3-2000

Design and Validation of an Accurate GPS Signal and Receiver Truth Model for Comparing Advanced receiver Processing Techniques

Phillip Martin Corbell

AFIT/GE/ENG/00M-07

DESIGN AND VALIDATION OF AN ACCURATE GPS SIGNAL AND RECEIVER TRUTH MODEL FOR COMPARING ADVANCED RECEIVER PROCESSING TECHNIQUES

THESIS

Phillip Martin Corbell, Second Lieutenant, USAF

AFIT/GE/ENG/00M-07

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

AFIT/GE/ENG/00M-07

DESIGN AND VALIDATION OF AN ACCURATE GPS SIGNAL AND RECEIVER TRUTH MODEL FOR COMPARING ADVANCED RECEIVER PROCESSING TECHNIQUES

THESIS

Presented to the Faculty of the Graduate School of Engineering and Management

of the Air Force Institute of Technology

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

Phillip Martin Corbell, B.S.E.E.

Second Lieutenant, United States Air Force

March, 2000

AFIT/GE/ENG/00M-07

DESIGN AND VALIDATION OF AN ACCURATE GPS SIGNAL AND RECEIVER TRUTH MODEL FOR COMPARING ADVANCED RECEIVER PROCESSING TECHNIQUES

Phillip Martin Corbell, B.S.E.E.
Second Lieutenant, United States Air Force

Approved:

Lieutenant Colonel Mikel M. Miller Assistant Professor, Thesis Advisor	Date
Captain John F. Raquet Assistant Professor, Committee Member	Date
Dr. Peter S. Maybeck Professor, Committee Member	Date
Dr. Michael A. Temple Assistant Professor, Committee Member	Date

Preface

First, I would like to give praise to the Lord my God and his Son Jesus Christ, who has blessed us beyond our ability to comprehend. He has provided many wonderful sources of strength and encouragement, none more consistent and loving than my wife Kristi. For all the support, understanding, "picnics," love, patience, and '*magic,*' I praise you—Truly you are *The Other Side of Me*. You are at least as responsible for the completion of this work as I—thanks for always believing in me.

At the risk of inadvertently missing some, I would like to thank my fellow classmates, especially George, Jeremy, Byron, Mike, Brian, Dan, Marcus, Paul, and Todd for bringing me up to speed on "the *REAL* Air Force." You have all provided me with a true education that AFIT can't take credit for—thanks for your support, advice, feedback, and your stories. I greatly enjoyed working with all of you, and appreciate the community we shared in the countless hours spent doing homework and finishing Theses. Thanks for your willingness to help me learn.

To the Promise Keepers at AFIT, I am eternally grateful. You have transformed my opportunities to learn into opportunities to *Grow*. Truly, you have nurtured my faith and lived lives worthy of imitation—I pray I will continue to be sharpened by your example. Thank you for having such an inspirational priority list.

Last, but certainly not least, I'd like to thank my Thesis Advisor, Lt. Col. Mikel Miller, my committee, and sponsor, Capt. Andy Proud. You have all provided feedback and advise which has proved invaluable in my constantly changing Thesis. Thank you for your support and wisdom.

Table of Contents

List of Figures

List of Tables

Page

Abstract

Recent increases in the computational power of computers and digital signal processors have made possible new, novel signal tracking techniques in GPS receivers. One such technique is known as Direct Correlator Output Processing (DCOP). This technique replaces individual traditional tracking loops with a single Kalman Filter, which jointly processes the received signals while exploiting their correlated noises. DCOP is innovative in its potential to *replace* the tried and true classical signal tracking loops. It is also an enabling technology for ultra-tightly coupled GPS/INS (Global Positioning System/Inertial Navigation System). Potential benefits of these new tracking techniques include an order-of-magnitude improvement in positional accuracy in environments of jamming and high dynamics. However, such performance gains are typically based on software simulations of conceptual GPS receiver designs, not working prototypes. Simulating these new designs requires the modeling of GPS signals and receiver tracking loops, instead of the traditional pseudorange and carrier-phase measurements, which many proven GPS simulation software packages accurately model. The purpose of this research has been to develop an accurate, user-friendly, and customizable GPS signal and receiver model to use for a fair and unbiased evaluation of advanced receiver designs. The result of this research is a Matlab® GPS signal simulator, a Simulink® GPS receiver model implementing true receiver DSP processing, and a Matlab® high-speed signal/receiver model that approximates the signal simulator and receiver model.

DESIGN AND VALIDATION OF AN ACCURATE GPS SIGNAL AND RECEIVER TRUTH MODEL FOR COMPARING ADVANCED RECEIVER PROCESSING TECHNIQUES

1. Introduction

1.1 Overview

Applications of the Navstar Global Positioning System (GPS) have been growing rapidly in recent years, with the completion of the entire system in 1993, and the remarkable demonstration of GPS (operating at only 66% of the full satellite constellation) during the Persian Gulf War 1991 [27]. Commercial GPS industries have made large investments in developing techniques to make GPS more accurate, robust, and useful [27]. Their efforts have resulted in an explosion of markets and applications, from millimeter level differential carrier-phase positioning to remote animal tracking. In developing these markets and the technologies necessary to support them, civilian receivers have achieved amazing advances in receiver design—resulting in higher positioning accuracy with increased robustness and reliability. Additionally, the doubling of computer processing power roughly every 18 months continues to fuel the rapid development of new innovative GPS Digital Signal Processing (DSP) related technologies. New alternatives to traditional signal tracking, once thought idealistic and unrealizable in real time, are now achievable with today's technology. Direct Correlator

Output Processing (DCOP) [39, 45, 59] and the ultra-tight integration of the GPS and Inertial Navigation System (INS) [35] are two such technologies that the Department Of Defense (DOD) is eager to acquire, develop, and deploy [2, 3]. The Government's eagerness has not gone unnoticed—many companies have responded with their own "new innovative technique" to compete for funding. Sorting through the resulting proposals is a laborious process, and meaningfully comparing projected performance gains is often an ambiguous process.

As the DOD struggles to modernize its GPS equipment amidst a sea of competing technologies, there exists an urgent need for simulation tools that level the playing field and allow direct comparisons between competing designs, early in the product development cycle. The general idea is not new: Simulation Based Acquisition (SBA) [47] is a program being developed by the Modeling and Simulation Information Analysis Center (MSIAC) [24] under the Defense Modeling and Simulation Office (DMSO) [9]. The advantages to this type of acquisition approach are apparent to the Government, and this process is actively being developed through these programs.

In the area of advanced GPS receiver design, one such badly needed tool is an easy-to-use, unbiased, accurate GPS receiver truth model. Such a model would allow simulation and analysis of new processing techniques earlier in the research and development (R&D) cycle, providing performance validation (or lack thereof) before funding occurs.

A direct application of such a tool would be the verification of a particular receiver processing technique that has the potential to unleash large gains in the area of carrier-phase precision level positioning in highly dynamic environments with signal

blockage, attenuation, and jamming. Known by several proprietary names [2, 3, 39, 59], the technique has been named by the Sensors Directorate of the Air Force Research Lab (to avoid proprietary favoritism) as Direct Correlator Output Processing (DCOP). The GPS signal simulator and receiver models developed in this research provide an easy-to-use tool for unbiased comparison and validation of new DCOP receiver designs against conventional signal tracking methods.

1.2 Problem Definition

With only a fraction of the development budget of the commercial sector, current military GPS receiver development lacks many of the latest receiver enhancements. Although some technologies are not suited for military applications (i.e., theatre Differential GPS, or DGPS), many offer substantial benefits. As the government endeavors to acquire and integrate new GPS technologies, it is constantly bombarded with technologies having "high potential" with few verified results. Verifying results in today's acquisition environment usually requires funding the development of these potentially promising technologies until a testable prototype is built. True Radio Frequency (RF) GPS signal simulators can then be used to verify performance claims. Many hours and millions of dollars in development costs are spent trying to validate the predicted performance of promising GPS technologies.

Validating potential technology earlier in the R&D cycle is highly desirable, as it allows for smarter funding distribution to technologies that are theoretically sound, reduces wasted time and money, and identifies potential technological caveats early on. The issue of conceptual validation is complicated by the fact that such validation is

typically done by the company pursuing the contract using their own in-house products and simulations—often developed by themselves for their own evaluation. For such "proof of concept analysis," it is often the job of the government contract manager to perform an unbiased validation of a company's technology, independent of their in-house analysis. Unfortunately, such managers rarely have the required tools to conduct an independent concept validation. In this situation, unsound technological concepts are not identified until the company goes to demonstrate capabilities of a working prototype.

In the area of advanced GPS receiver technologies, independent validation is often a challenge due to the non-traditional nature of new designs and a lack of Commercial-Off-The-Shelf (COTS) products that can validate new receiver processing schemes. This is a big problem when trying to validate DCOP related technologies. DCOP techniques involve processing demodulated signals rather than pseudorange and carrier-phase information, which are the classical observables generated by a GPS receiver. In order to process the demodulated signal, a DCOP algorithm must take over the responsibility of signal tracking, a process normally handled by traditional communication tracking loops. The input signal components used by the DCOP architecture are known as the In-phase (I) and Quadra-phase (Q) signal components, which are the sampled and processed components of the received GPS signal generated inside a GPS receiver. Figure 1-1 shows the structure of a generic GPS receiver, divided into four key areas (labeled A, B, C, and D). Typical COTS GPS software simulators model pseudoranges and carrier-phase observables [50], generated in block 'B' of a real GPS receiver and processed by section 'D' (refer to Figure 1-1). However, DCOP and ultra-tight INS/GPS architectures directly process the I and Q samples, which are also

generated by block 'B,' but are typically only given to block 'C'. In addition, these architectures typically assume control of Block 'B', generating the Digitally Controlled Oscillator (DCO) commands that traditionally originate from Block 'C'. Block 'C' is effectively replaced by these architectures, as they take over the responsibilities of signal tracking. DCOP architectures often integrate portions of Block 'D', but are capable of simply replacing Block 'C' entirely. Ultra-tight GPS/INS structures typically combine Blocks 'C' and 'D' with an INS or Inertial Measurement Units (IMUs).

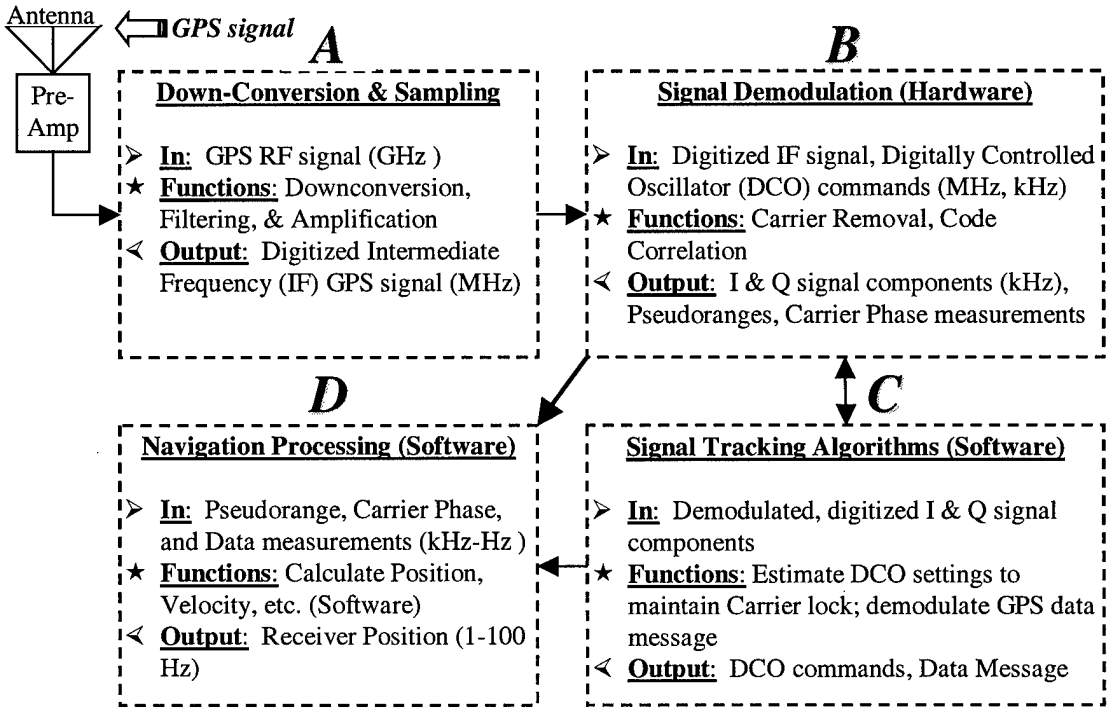


Figure 1-1. Generic GPS Receiver Structure

Although many software packages exist that accurately model traditional GPS observables (pseudorange and carrier-phase measurements, block "C" in Figure 1-1), few are available that model the signal tracking process (the right half of Figure 1-1), which is necessary to generate I and Q samples correctly. Ultra-tight integration simulation

requires I and Q samples be accurately simulated with the same truth trajectory and initial conditions as those given to the INS model. This ensures the closed-loop feedback in the integrated system is correlated correctly and stable. Traditional signal tracking algorithms are well understood and usually very predictable, thereby allowing robust simulation of pseudorange and carrier-phase measurements without tracking loop modeling. However, In the case of DCOP and ultra-tight integration, the signal tracking process is at least partially, if not totally, performed by additional hardware in the system, either augmenting or replacing traditional tracking loops. Therefore, evaluation of these techniques requires a receiver tracking loop model to simulate the highly integrated technologies.

Such in-depth GPS receiver modeling was not necessary in the past, as most simulation and modeling was/is still done using the traditional raw observables (pseudoranges and carrier-phase measurements). However, certain applications, i.e. those that must observe or intervene into the signal tracking process, require this level of modeling. Tetewsky and Soltz of Draper Laboratory identified which capabilities a software model/simulator of this type should have, specifically for a test tool capable of RF signal modeling, DSP, navigation and modeling simulation [50]:

> [A] detailed simulation of an entire GPS receiver . . . should generate base-band in-phase (I) and quadrature (Q) error signals at 2-kHz sample rates. These data mimic the integrate and dump outputs of the DSP chip found in all modern GPS receivers. The package should also provide some sample tracking-loop models. If it generates the actual sample values using a 4 MHz bandwidth for C/A code and 40 MHz for P(Y) simulation, which includes the front-end automatic gain control (AGC) and quantizer, this would be very valuable. Simulink, a product for block-diagram design of dynamic systems for use with MATLAB, is probably a good implementation option when combined with MEX files for speed. [50:54]

1.3 Problem Solution

The solution to this problem is to build a user-friendly, accurate truth model of a GPS receiver. A quality truth model in the hands of a Project Manager would allow for an unbiased, fair evaluation of competing technologies. As a deployable asset, companies could be required to demonstrate their technology using the same truth environment as competing technologies, allowing for a fair and direct comparison. If a company provides their own evaluation tool, the Project Manager can compare the company's tool with a verified in-house truth environment. The Project Manager could also replace the company's models with those developed by this research, in order to generate comparable results. These results would also be comparable to traditional receiver design performance evaluated with the same truth model.

The need for a GPS receiver simulation software model was identified in the previous section with a list of recommendations [50]. Every effort is made to address these recommendations. The main focus throughout this research is to incorporate error sources and receiver effects which have a measurable impact on the validity of the sampled I and Q statistics. Many errors that are typically modeled in the pseudorange do not have an appreciable impact on tracking performance because they are slowly changing biases that do not show up in the sampled I and Q values. Biases in the received signal are quickly removed in the tracking loops and do not significantly contribute to tracking loop errors, i.e. I and Q statistics are relative to the tracking errors, not the GPS position error.

1.4 Summary of Similar Research

Modern day GPS receiver designs were undoubtedly developed with the aid of computer aided design tools, both for actual hardware chip simulation and testing, and for advanced receiver algorithm development. In addition to being proprietary and typically unavailable to the government Project Manager, these tools typically are very costly, have steep learning curves, and/or are not well documented, since their original purpose was for development of receiver designs, not for generic receiver modeling and ease of use. This summary of similar research quickly covers some characteristics of commercial simulation tools.

In addition to these commercial and internally developed tools, there are some existing GPS receiver simulators that have been developed for purchase by the GPS community. These tools vary in complexity, flexibility, and cost. This summary also includes a short discussion on the existing GPS receiver simulators that are available.

1.4.1 Simulation and Modeling Tools

In pursuing a simulation of any system, there are many modeling packages from which to choose [10], more than can adequately be listed and described here. Many of the 138 packages listed at [10] are high fidelity commercial software products used by companies for product R&D. Commercial software often has a large operational learning curve, although companies that design these tools have endeavored to make them as user-friendly as possible. As a result, these packages typically implement block-diagram level interfaces in a feature laden Graphical User Interface (GUI), instead of older command

prompt "DOS" type interface. However, many of these packages are cost-prohibitive and unnecessarily complicated for use in validation.

There is a wide variance in the level of modeling and capabilities among these tools. The combination of features required for a particular application depend on many factors, including cost, signal processing needs, built-in function libraries, fixed-point analysis, real-time evaluation, software vs. hardware end-goals, and level of control [7,8]. There are many commercial resources for help in simulation and modeling [46,56], as well as government and military resources. Government resources include the Defense Modeling and Simulation Office (DMSO) [9] and the Modeling and Simulation Information Analysis Center (MSIAC) [24]. The purpose of these government agencies is to centralize government and military modeling and simulation knowledge base. One special interest item being pursued by MSIAC is a concept known as Simulation Based Acquisition (SBA) [47], which is a proposed method for validating and acquiring new technology. The results of this research contribute to that very effort. The tools developed in this research enable project managers to make acquisition decisions based on simulation of advanced GPS receiver designs from competing companies.

1.4.2 Existing GPS Receiver Simulators/Models

Some existing GPS receiver simulators/models were found in open literature. One of the publicly available products is the GPS Signal Simulation Toolbox for Matlab®, written by Navsys Corporation [28]. This product was reviewed in the article by Tetewsky and Soltz from Draper Laboratory [50] and is written for Matlab® version 4.2 (the current version is 5.31). The toolbox performs single-channel Monte Carlo

receiver simulation with faithful modeling of the front-end digital signal correlation and carrier conversion processing (using a 4-MHz model). According to [50], it was the only Matlab® simulation product available at the time of the article's publication (October 1998) that is capable of this type of analysis.

Another available model recommended by Navtech Inc. [30], is the GPS Correlator Simulator by Accord Software and Systems [1]. According to Navtech, Accord's product is a "very powerful software package for understanding the characteristics of the GPS satellite signal and its processing in a typical GPS receiver" [30]. Composed of two pieces, an Input Signal Simulator and a Signal Processor, it is able to simulate noise effects, receiver circuit parameters and baseband converter, satellite signal acquisition and tracking, and C/A code properties. However, it appears to be a single-channel simulator, which runs as a compiled stand-alone program on a PC. It implements the Costas Loop phase tracking algorithm, and can induce Doppler shift on the code and carrier. It does not appear to be easily interfacable with external tracking mechanizations, nor could it accept tracking corrections from external sources. Additionally, its truth environment is not driven by satellite position and receiver trajectory, a requirement for simulating ultra-tightly coupled navigation systems.

ACOLADE® [26] is a visual block-diagram oriented simulation package, used to model a GPS Receiver in an Application Note published by ICUCOM, the maker of ACOLADE®. In this note, the entire signal generation, error sources, and demodulation process is simulated inside ACOLADE. A review of the note reveals a much higher emphasis on modeling software flexibility than on accurate implementation of a receiver model. Although it is probable that a good quality model could be built inside this

system, the design as described in the application note is not particularly useful for purposes previously mentioned.

Mr. Ward developed a receiver model for running Monte Carlo simulations to predict Radio Frequency (RF) Interference Performance [58]. The design presented in [58] is an analytical representation of a GPS receiver. It is implemented in Matlab®, a mathematical programming environment. As such, it is not directly comparable to the other visual block-oriented time-ordered simulators previously discussed. This simulator does not actually generate the carrier or code waveforms, but keeps track of their phases and relationships to generate the I and Q samples with mathematical models. The models used in this paper are the same that are used in [44], and are also used in the Matlab® I and Q model developed during this research. The simulator in the paper only models a single channel, and the inputs are not based on satellite or receiver positions, but rather created to match the desired signal dynamics specified in the paper. While the approach and design are solid for the research performed in the paper, the design of the Monte Carlo simulator does not appear to be tailored towards the applications mentioned here. Nonetheless, the source was found after the Matlab® I and Q model was developed for this research, and the models used in the paper were found to be the same or very similar to those used in this research.

Probably the most promising GPS receiver model has been developed by the GPS/INS Branch of the Naval Air Warfare Center Weapons Division. Unfortunately, very little information on this tool is available. According to their website [31], they have developed a GPS receiver that models code and carrier tracking under dynamic conditions. It is capable of modeling jamming, "multiple types of discriminators, code,

and carrier loops" [31]. It is possible to incorporate INS aiding into the code and carrier loops, and it is available in Fortran® and Simulink® realizations. Of existing GPS receiver simulators found, this one seems to be the most promising existing product in terms of filling requirements defined in this thesis.

1.5 Scope

Truth models are only useful if their accuracy is verifiable. This requirement quickly limits the truth model design to one that mimics existing hardware. Thus, verification is performed against known effects and phenomena of the existing hardware and compared to real measured data. However, the design must be flexible enough to model a plethora of varying receiver designs—each company's GPS hardware is undoubtedly different. In addition to being accurate and adaptable, it must be easy-to-use with a short operational learning curve—i.e. a project manager must be able to understand and use the tool within a minimal amount of time.

To confidently model and simulate I and Q samples with any degree of accuracy, one must take an in-depth look at everything occurring in the signal reception and demodulation process that can affect these samples. Here, the focus is placed on factors affecting tracking loop performance and thresholds. The models developed are accurate to both the real-time operation of the system being modeled and inclusive of all the errors affecting old and new tracking algorithms. Thus, the truth environment does not favor traditional tracking methodologies over new competing tracking technologies and vice-versa.

The truth trajectory and satellite positions used in this research are simulated. This prevents measurement errors in recorded trajectories from resulting in unknown system dynamics. The signal simulation model accepts any trajectory in the specified format at any sampling rate. However, the receiver model fidelity is greatly affected by the sampling rate associated with its truth trajectory. This is due to the spline interpolation applied to the input trajectory to generate an estimate of the truth environment at each sample time. The I and Q signal components are generated at the sampling frequency. Therefore, a higher frequency in the truth environment relies less on interpolation and more on the truth trajectory, resulting in better modeling of receiver dynamics.

The receiver model does not incorporate GPS receiver navigation processing, satellite selection logic, or any of the higher-level software functions. It is meant to model the processing hardware and GPS received signal perturbations faithfully, impacting the realism of generated I and Q samples in the tracking process. It includes very little "decision making" logic, except as required to allow for receiver specific modeling parameters and user control.

A secondary application of this research is its probable deployment as teaching tool in the Advanced GPS course at AFIT. In light of this, the Matlab® and Simulink® models are designed such that they are segmentable and incrementally deployable, piece-by-piece. Every effort is made to comment the code and models well, to aid in the comprehension of GPS receiver design by the students in these courses and sponsors of this research alike.

1.6 Assumptions

It is assumed that Selective Availability (SA) effects [33] are avoided by some means in this problem. The impact of this assumption is a limitation on the hardware configuration of the GPS equipment in the real world. SA can be circumvented by using 1) a P(Y) code (keyed) receiver, 2) a C/A code receiver in a differential mode, or 3) a C/A code receiver with S/A turned off (tentatively scheduled for 2006). SA could also be bypassed by using a GPS signal simulator.

The receiver model assumes constant (but specific) received signal strength per satellite per run, avoiding complications of body and terrain masking on a GPS receiver's trajectory (all Satellites included in the simulation are assumed visible throughout the duration of the simulation). The receiver model does not incorporate satellite selection logic and does not change satellites and/or their channel assignments during a simulation. The simulation uses cubic spline fit interpolation techniques which inherently assume piece-wise continuous 3^{rd} order dynamics (jerk). The spline fitting interpolates all data points (determined by the sampling rate) between the provided data points in the input trajectory, thus smoothing very high dynamic effects from the input trajectory. This could possibly lead to better than realizable results. The best way to counter this phenomenon is to provide truth trajectories at higher sampling rates. Clock errors induced by receiver dynamics are not modeled. Two types of Automatic Gain Control (AGC) circuits are provided. The first type is implemented in Matlab® and calculates optimal, fixed thresholds over user-defined intervals. The second AGC is a configurable dynamic AGC model implemented in Simulink®, which more closely models the finite response time of an AGC to fluctuations in the received spectrum. The first AGC model is faster,

and accurate in situations in which the received noise is constant, while the second AGC model is to be used for analysis in jamming or other RF in-band interference scenarios.

1.7 Approach

The GPS modeling is broken into three distinguishable products. The first product is a GPS signal simulator model that simulates digitized and down-converted GPS signals. The second product is a GPS Receiver Correlator/Demodulator model, which comprises the entire (high-speed) DSP section of a typical GPS receiver. After developing these two products, an analytical model for accumulated I and Q samples is mathematically derived and implemented, based on the operating characteristics of the receiver design and the GPS signal structure.

The logical point to model/create the received GPS signal and noise spectrum is at the A/D converter output (output of block "A" in Figure 1-1), where the process becomes digital. There are inherent problems with accurately modeling high-frequency analog signals in a discrete environment such as Matlab® and Simulink®. Although Simulink® is designed to model continuous systems, it relies on the computationally expensive and time-intensive method of solving sets of differential equations at each time step. Analog simulation of tracking loops in Simulink® was previously explored by Baier [6], where many assumptions and compromises in modeling accuracy were made to obtain credible results in a reasonable amount of time in the pseudo-analog world of continuous-time Simulink® models. Furthermore, most modern GPS receivers digitize at a Intermediate Frequency (IF) and do all their processing digitally from that point on, so digital modeling is more accurate in this regard as well. Many of the errors and receiver effects

can be accurately modeled at this point by deriving analytical error expressions, which are evaluated at each sample time. The signal simulator does not depend on any fed-back information; it only requires the knowledge of pre-specified truth parameters that control simulator operation. With the lack of any feedback control, this model is ideally suited for mechanization in Matlab®.

The receiver model is inherently a time-ordered closed-loop process, which is better suited for implementation in Simulink®. Whereas Matlab® processes are code-ordered, Simulink® processes are time-ordered, such that all functions in a process are evaluated at each time step. This type of processing is difficult and inefficient to implement in Matlab®. Dynamic feedback is also difficult to provide via Matlab® programing, and is easier to build into Simulink® Diagrams. Therefore, the receiver model is implemented in Simulink®.

The final model represents an analytical approximation of the signal simulator and the receiver model. The purpose of this model is two-fold. First, it is used to provide theoretical validation of the first two models. Second, given that the analytical model suitably represents the truth environment provided by the signal and receiver models, it can be used as a verified (faster) generator of summed I and Q values. Since this model consists of pseudo-code, implementing equations to approximate the accumulated I and Q samples, it can be ported to different operating environments and runs much faster than the high fidelity Matlab® signal simulator and Simulink® receiver model. This analytical model was used in a thesis by Michael Presnar [35] to simulate an ultra-tightly integrated GPS/INS system.

1.8 Materials and Equipment

As recommended by Tetewsky and Soltz [50], Simulink® is used to model the digital demodulation section of the GPS receiver. Matlab® is utilized to simulate the digitized IF spectrum of GPS signals plus noise. Due to the amount of information readily available, and the educational and developmental emphasis of the product, Mitel's (formerly GEC/Plessey) GPS C/A code chipset is chosen as the GPS receiver design to model [20-23]. Although this research focuses on modeling this particular chipset, effort is made to make the design flexible enough to be easily reconfigured to other receiver designs as well. This chipset was also chosen due to the added bonus that real I and Q data has been collected and graciously provided by Tracking and Imaging Systems Inc. in Bloomington IL [52] for validation of the I and Q model.

The version of Matlab® used in this research is the most recent release, Release 11.1, version 5.3.1.29215a. The version of Simulink® used is also the most recent release, version 3.0.1. The Simulink® model requires only the standard Simulink® blocks. The Matlab® code developed requires no special toolboxes for its operation.

1.9 Thesis Organization

Chapter 2 details the GPS signal structure and system design, as well as, receiver design and signal tracking methods. A brief explanation of non-conventional GPS signal tracking techniques is provided and their potential benefits identified. Chapter 3 details the model development for the signal simulator software, GPS receiver, and the analytic I and Q model. It explains the models used for the downconverted and sampled GPS signal and associated errors. It also includes a discussion of Simulink® as a simulation

tool and a model-by-model explanation of the blocks comprising the receiver model. Finally, detailed equations for the accumulated I and Q values are provided. Chapter 4 provides a detailed analysis and validation of the models developed, cross-checking the analytical I and Q model against equivalently generated I and Q values using the signal simulator and receiver model. It also compares both results to real logged I and Q data graciously provided by Tracking and Imaging Systems Inc. Finally, Chapter 5 offers conclusions and recommendations for further research.

2. *Theory*

This chapter provides the background necessary to understand and apply the models derived in this thesis. The chapter begins with an overview of GPS and an introduction to Direct Sequence Spread Spectrum (DSSS) communications. After discussing DSSS signal properties, the GPS signal structure is defined. Following a review of signal power notation, a discussion of traditional receiver designs and tracking techniques is presented, followed by a few new and intriguing approaches to signal tracking. GPS errors are then explored and the rationale for ignoring specific errors in the model derivation is presented. Finally, some applications of this research are presented in context with the limitations of current GPS receivers in specific applications. This chapter does not attempt to discuss all types of GPS receivers, nor does it provide a comprehensive listing of all the applications of this research.

2.1 Global Positioning System Overview

The NAVSTAR Global Positioning System (GPS) is a modern-day triangulation-based positioning scheme. GPS satellites transmit ranging signals, that, when received and processed by a GPS receiver, provide distance measurements between the satellites and receiver. Because this ranging method relies on measuring the arrival time of radio waves, very small differences (10^{-6} seconds) between a receiver's clock and the satellite's clock create large errors (300 m) in these distance measurements. Receiver clock errors dominate the other errors in the distance measurements, known as pseudoranges (PR). While traditional triangulation suggests only three measurements are required to calculate

position, these clock errors add the time dimension to the triangulation process. Thus without any external positioning (or timing) information, a GPS receiver needs PR from *four* satellites to calculate position—solving for X, Y, Z coordinates, and time.

2.1.1 GPS History

GPS was officially approved for development on 17 December 1973. The first satellite was launched in February of 1978, and the entire constellation of 24 satellites was first complete in December 1993. All system components reached full operational capability in the spring of 1995, following completion of comprehensive testing on all production GPS satellites [33].

2.1.2 Satellite Constellation Overview

The GPS constellation of 24 satellites is arranged into 6 nearly circular orbital planes with 4 satellites per plane. The planes are inclined 55 degrees relative to the equator and evenly spaced about the equator every 60 degrees. With an orbit of 26,600 km from the earth's center, their orbital period is one-half a sidereal day or 11 hr, 58 min [14]. The resulting orbital velocities cause a maximum Doppler shift of 6 kHz for a stationary receiver on the earth's surface [33]. The spacing of the Satellite Vehicles (SVs) in their orbital planes is optimized to provide at least four visible satellites at any point on or around the earth at nearly all times. The coverage standard defines a >99.9% probability of 4 or more satellites in view over any 24 hour interval, averaged over all the locations on the globe [11]. This standard is predicated on 24 operational satellites; however, the current constellation contains 30 active (in working order) satellites [54], 24 of which are guaranteed by the control segment to be operational at any one time[11].

2.1.3 *Positioning Services*

GPS provides two positioning services, with differing accuracies and tolerances to interference. These services are known as the Standard Positioning Service (SPS) and the Precise Positioning Service (PPS). SPS is an unrestricted service, i.e., available to anyone worldwide. This positioning service provides signals modulated by a pseudo-random spreading code known as the Coarse Acquisition (C/A) code. It provides positional accuracies of 100 meters (2 drms, 95%) in the horizontal plane and 156 meters (2 drms, 95%) in the vertical plane [14]. The distance root mean square (drms) is a common measure of 3-dimensional positioning accuracy. A circle with a radius of $2 \cdot drms$ (twice the drms value) includes 95% of the positions that would be calculated from the SPS signals [14].

The Department of Defense is currently implementing an intentional degradation in the positional accuracy of SPS, known as Selective Availability (SA). Without SA, the standard deviation of estimated error in each PR drops from 33.3 meters to 8 meters. This equates to accuracy on the order of 25m (2 drms) and 43 m (2 drms) in the horizontal and vertical planes, respectively [14]. The U.S. Government has announced that SA is to be turned off in the year 2006.

PPS is a positioning service limited to the military that provides encrypted positioning signals. Only authorized receivers can decrypt PPS signals. However, the C/A code of the SPS system is (typically) needed to initially acquire the PPS signal; in fact, this was its primary original purpose [33]. The Horizontal accuracy of this service is at least 22 m (2 drms) and the vertical accuracy is at least 27.7 m (2 drms) [14]. The PPS system provides signals modulated on two different carrier frequencies, allowing for

direct ionoshperic error correction. The spreading code for this system is known as the P(Y) code, where the Y code is the encrypted version of the published P-code.

2.1.4 Differential GPS

Differential GPS (DGPS) is a technique using multiple receivers to cooperatively increase the positional accuracy of a GPS calculated position. It consists of comparing the GPS calculated position of a fixed (reference) receiver with its surveyed position, calculating the errors between the two positions, and broadcasting these error corrections to other (mobile) GPS receivers. According to Kruczynski [33], "Differential GPS was an integral part of range operations from the beginning [of GPS]." However, DGPS methods and receiver technologies have resulted in civilian positioning accuracies much greater than the designers of GPS had anticipated [33]. Differential positioning techniques attained much higher levels of accuracy when carrier-phase ambiguity resolution techniques were developed in the early 80's [38]. These techniques currently provide the highest level of accuracy achievable—without using information external to GPS measurements. Current GPS receivers using carrier-phase ambiguity resolution techniques are capable of *sub-centimeter* accuracy in low-dynamic environments. However, these techniques require all the receivers in the differential system to track the carrier-phase of the GPS signal, compared with tracking the carrier frequency. Thus carrier-phase tracking is required in order to achieve sub-centimeter positional accuracies.

2.2 Direct Sequence Spread Spectrum Communication

Because a GPS receiver requires multiple signals from separate satellites, the GPS signals use a multiple access technique to keep signals from interfering with each other. The technique used in GPS is called Direct Sequence Spread Spectrum (DSSS) modulation, which is a type of Code Division Multiple Access (CDMA). DSSS is the only CDMA technique that uses a continuous, fixed carrier frequency. This method modulates the carrier in such a way that multiple signals simultaneously occupy the same frequency spectrum, while still maintaining orthogonality in "code space," thus, they can be readily distinguished from one another at the receiver. Most importantly (to GPS), DSSS modulation provides the ability to measure signal propagation times which are used to form pseudoranges (PRs), the key ingredient to calculating user position. Other benefits include resistance to jamming and a relatively low amount of interference contributed by other DSSS signals within the system [34].

In a DSSS system, there are three basic components—the carrier frequency, the data modulation, and the pseudorandom spreading modulation. Figure 2-1 shows these three basic components in a very simplified diagram of a DSSS transmitter. The carrier is data modulated using Phase Shift Keying (PSK); where data bit values are used to shift the carrier signal phase (see Figure 2-2). The spreading modulation is a pseudorandom sequence of ones and zeros which applies additional Binary Phase Shift Keying (BPSK) modulation to the data modulated carrier. The spreading modulation frequency is typically at least an order of magnitude higher than the data modulation frequency. The spreading modulation "codes" the signal in the transmitter so that it can only be "un-

encoded" (or decoded) with the same spreading sequence locally generated in the receiver. The properties of the spreading modulation are the topic of the next section.

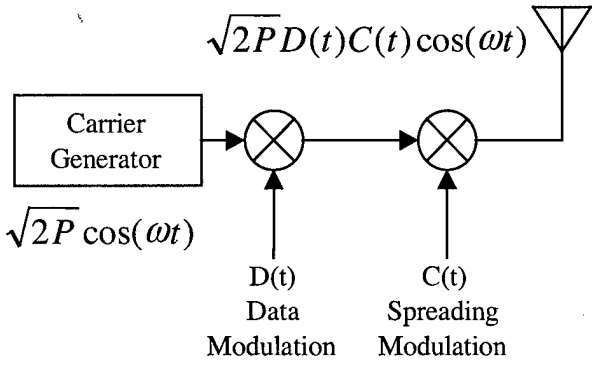


Figure 2-1. Basic spread spectrum transmitter

2.2.1 *Spreading Modulation Properties*

There are various terms used to describe the spreading modulation of a DSSS system. Pseudorandom spreading code sequence, pseudorandom sequence, pseudonoise sequence, pseudorandom noise sequence, code signal, spreading signal, ranging code, and spreading code are all terms that are used to describe the spreading modulation in a DSSS system (sequence and code are often used interchangeably). The term "PRN" (usually followed by a number) is used in GPS as an identifier for individual GPS satellites, however PRN stands for Pseudo-Random Noise, which again refers to the spreading modulation used in the GPS satellites. Pseudonoise (PN) sequence is used in the spread spectrum discussion to describe the spreading modulation.

A PN sequence is a deterministic string of "N" ones and zeros that appear random, but are purposely ordered to achieve noise-like auto- and cross-correlation properties. These auto- and cross-correlation properties are what prevent signals in the

same frequency spectrum from interfering with each other. The ones and zeros in these sequences are referred to as chips, and the number of chips in a sequence is the length (N) of that sequence. PN sequences always contain an odd number of chips, and the number of ones and zeros differ by no more than one (i.e., the sequence is *nearly* zero mean). Some classes of PN sequences include maximal-length codes, Gold codes, and non-linear codes [34, 48]. GPS uses Gold codes and non-linear codes in the SPS and PPS signal, respectively. Because this research does not focus on the PPS signal, the discussion here focuses on the PN sequence properties of the SPS signal, which are a family of Gold codes.

In a DSSS system, PN sequences are mapped to square waves, which are then clocked at a frequency known as the "chipping rate." The square wave has values of "–1" and "+1", and is known as a spreading waveform. In practice, the "zeros" of a sequence are mapped to "+1" while the "ones" of a sequence are mapped to "-1." BPSK modulation is then simply the multiplication of the spreading waveform with the carrier, as demonstrated in Figure 2-2. From Figure 2-2, it can be seen how multiplying the modulated carrier by the same modulation waveform reproduces the original carrier by re-inverting sections of the signal that were inverted during modulation. If BPSK is used for data modulation, such as in GPS, this process happens twice—once with the data modulation and once with the spreading modulation. In the receiver, the spreading waveform is known and regenerated, so once code synchronization has occurred, the phase reversals caused by spreading are reversed, leaving only a BPSK data modulated, waveform, which is readily demodulated using a BPSK demodulator.

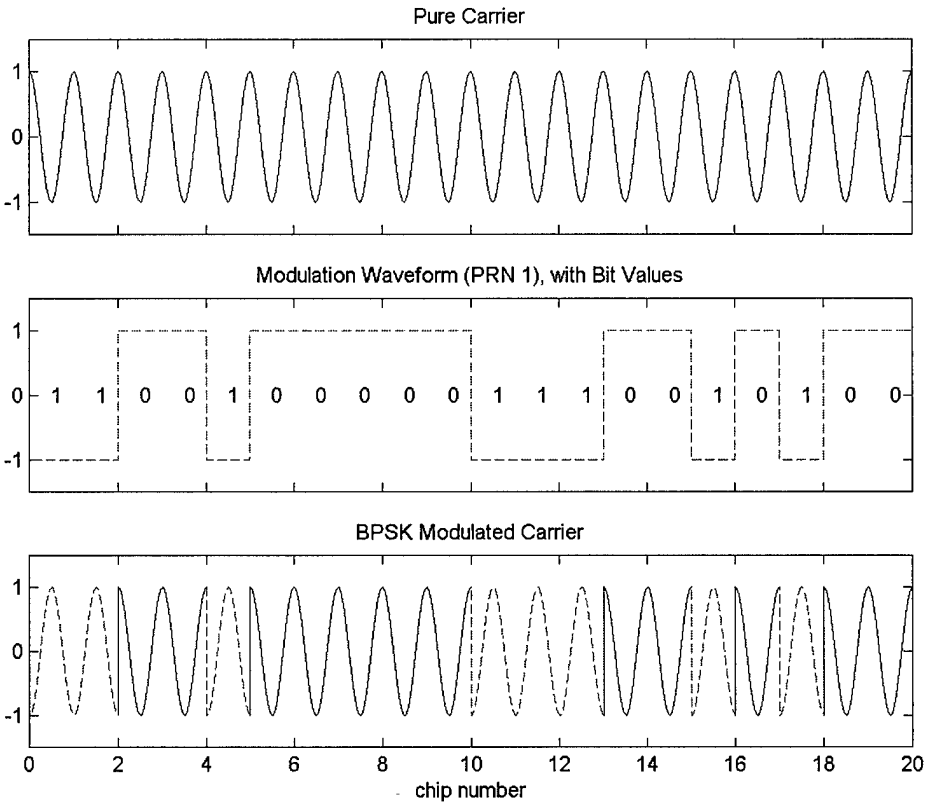


Figure 2-2. Example of BPSK modulation

An example of a typical DSSS multiple access system is depicted if Figure 2-3. This example includes three DSSS transmitters and one DSSS receiver. Each of the transmitters uses a unique spreading sequence, and the receiver can demodulate any transmitted signal by synchronizing a locally generated PN sequence with the same PN sequence used to generate it, taking into account the time delay induced by the propagation time. With this background, desirable correlation properties of the PN sequences are presented.

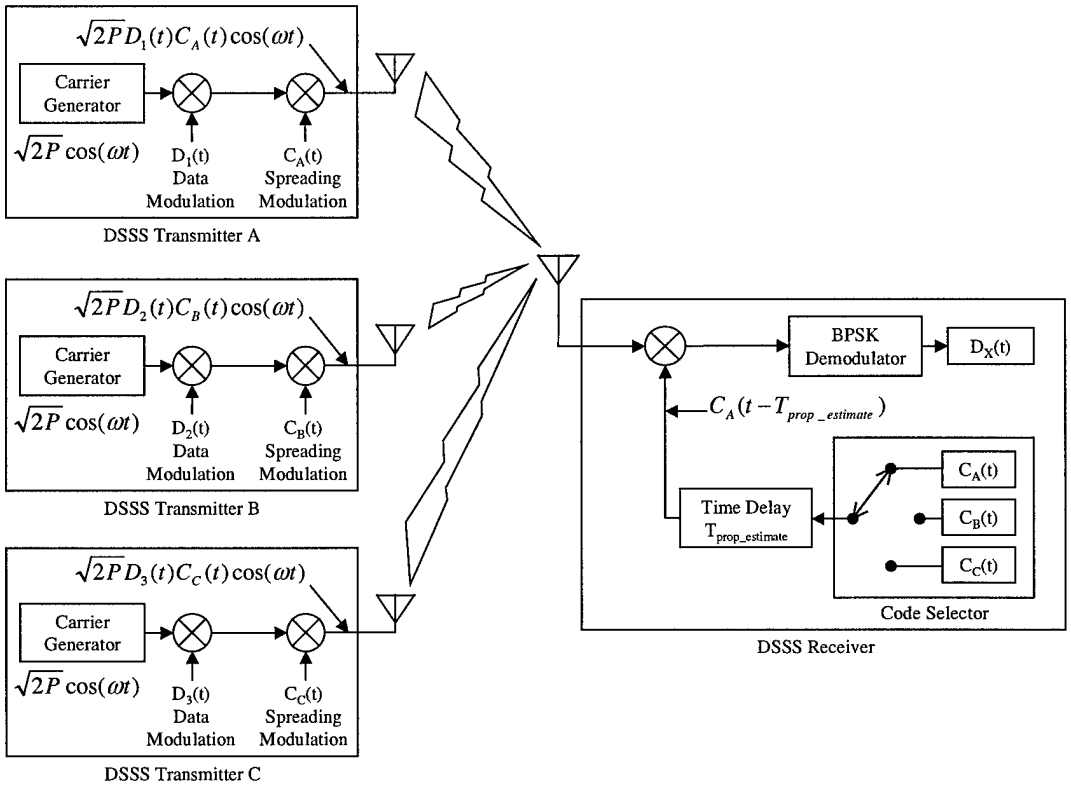


Figure 2-3. Typical multiple access DSSS block diagram

The autocorrelation function of white noise is zero everywhere except at a time difference of $\tau = 0$. Likewise, a vector of samples of a white noise process is uncorrelated with any shifted version of itself. PN sequences have autocorrelations that approximate those of white noise. Longer sequences exhibit closer approximations of white noise (in their correlation characteristics), but they also increase the time needed for a receiver to determine the correct "code phase" of the received signal. Determining the "code phase" is analogous to synchronizing with the received signal, which must occur for the signal to be detectable in the receiver. Longer sequences have more "phases" to search through in the synchronization process. Thus the shorter C/A code (repeating every millisecond) is used to acquire the longer P-code (repeating every

week). Because PN sequences are finite in length, the autocorrelation plot is periodic with a peak occurring at $\tau = 0$, and at $\tau = \pm n/T_C$, where T_C is the chip width in seconds and n is an integer value. A plot of the a normalized Autocorrelation function at $\tau = 0$ is shown in Figure 2-4.

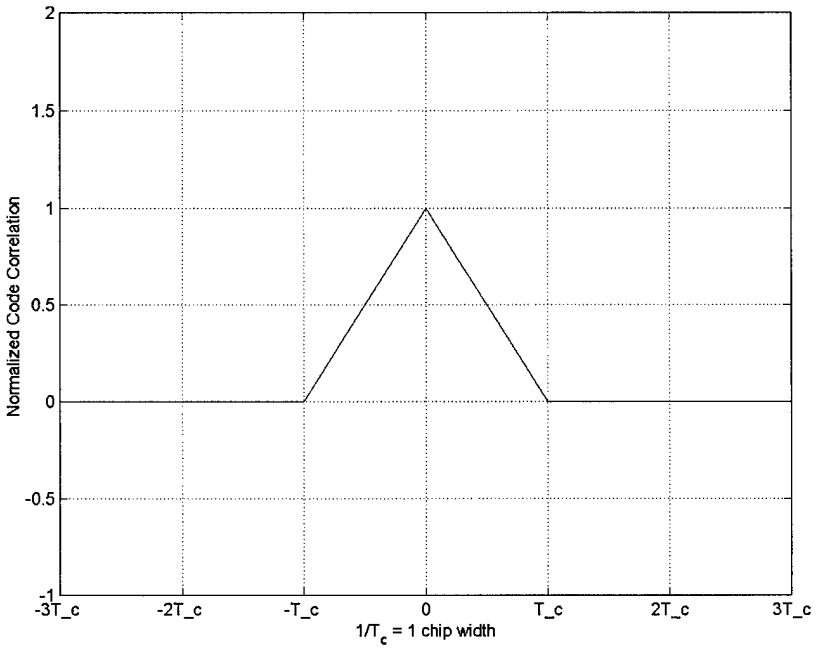


Figure 2-4. Normalized ideal code autocorrelation function

Two *independent*, discrete, finite-length, *white* noise vectors are uncorrelated with each other, making their circular cross-correlation values ideally zero [18]. Cross-correlations between Gold codes in the same family are not completely zero, but predictably and consistently low when compared to either code's autocorrelation peak. Any two Gold codes within the same family have the same cross-correlation values, which get lower (relative to a code's autocorrelation peak) as a code gets longer [34]. In the case of the C/A codes, the maximum power of any cross-correlation value is at least

–23.9 dB lower than the autocorrelation peak (only with zero Doppler offset, see [14]). This is also the power difference between the autocorrelation peak and its 2nd highest peak. Plots of the autocorrelation and cross-correlation of two GPS Gold codes (PRN 1 & 2) are shown in Figure 2-5. The auto and cross-correlation plots for all other PRNs have similar values.

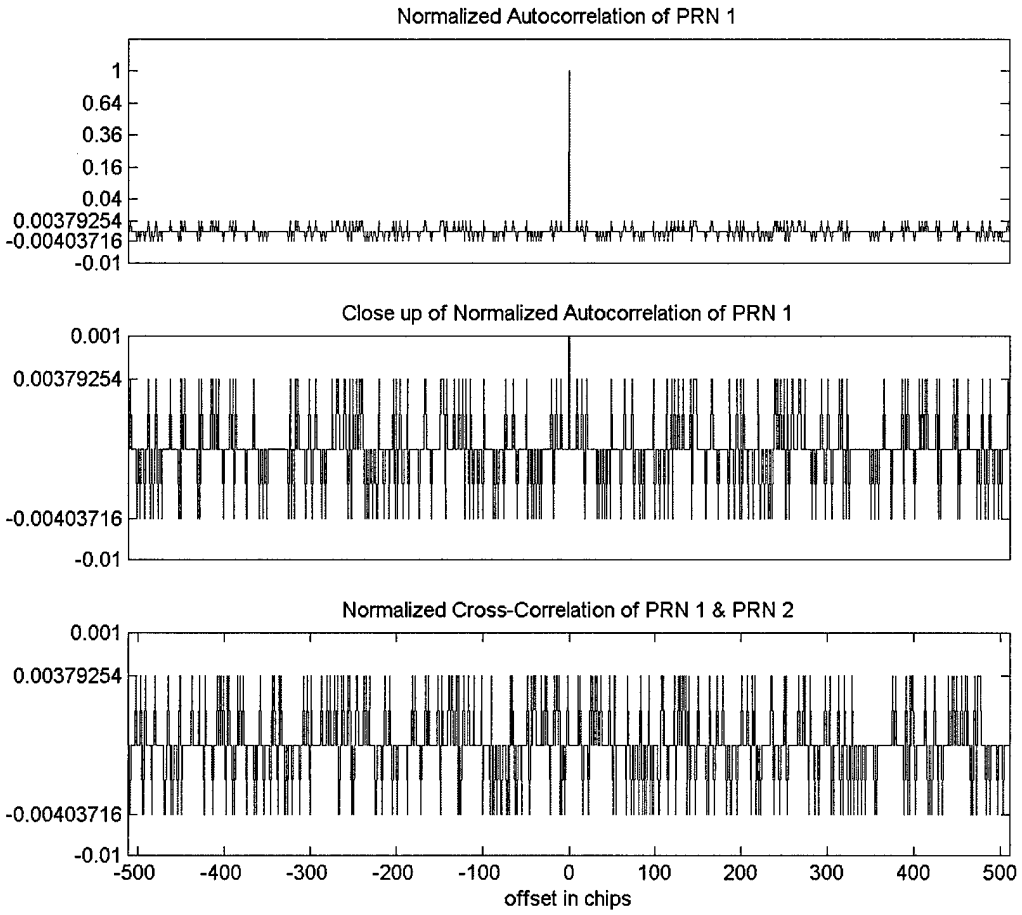


Figure 2-5. Auto and cross-correlation plots for PRN 1 & 2

2.2.2 *Signal Spreading and Anti-Jam Characteristics*

Any BPSK modulated signal has a single-sided Power Spectral Density (PSD) given by Equation (2.1) [34]. As mentioned earlier, A GPS signal is BPSK modulated by

both the data and the PN sequence. The following equations are for a generic BPSK modulation scheme, not specifically a GPS signal.

$$S(f) = P_C T_B \operatorname{sinc}^2\{2\pi(f - f_0)T_B\} \tag{2.1}$$

where:

P_C = Power in the carrier signal
T_B = Time duration of one "bit," or 1/"bit rate"
f_0 = Carrier frequency

This power spectrum contains ~90% of its power in the main lobe (between the first nulls), which is $2/T_B$ Hz wide [33]. This bandwidth is the null-to-null bandwidth of the signal, controlled by the period of the BPSK modulation. In a spread spectrum system (as mentioned earlier), the chipping rate is typically an order of magnitude higher than the data rate. If the data and PN waveforms transition synchronously, and the chipping rate is much greater than the data rate, the resulting DSSS modulated signal exhibits a bandwidth of twice the chipping rate and a PSD given by Equation (2.1). Given these conditions, the spreading modulation causes 90% of the power in the signal to be distributed over a bandwidth of 2 x [chipping rate] (Hz). This bandwidth is much wider than what is required for the data rate, hence the term, "spread spectrum." Frequencies outside of this bandwidth are typically filtered out in a GPS receiver.

DSSS modulation method also provides inherent anti-jam capability. Figure 2-3 illustrates that the same spreading code (PN sequence) used to spread the transmitted signal is used to *de-spread*, or reconstitute the original signal, in the receiver. However, any power received in the transmitted signal bandwidth which is not modulated by the

spreading code is effectively *spread* by the *de-spreading* process. Figure 2-6 illustrates this process.

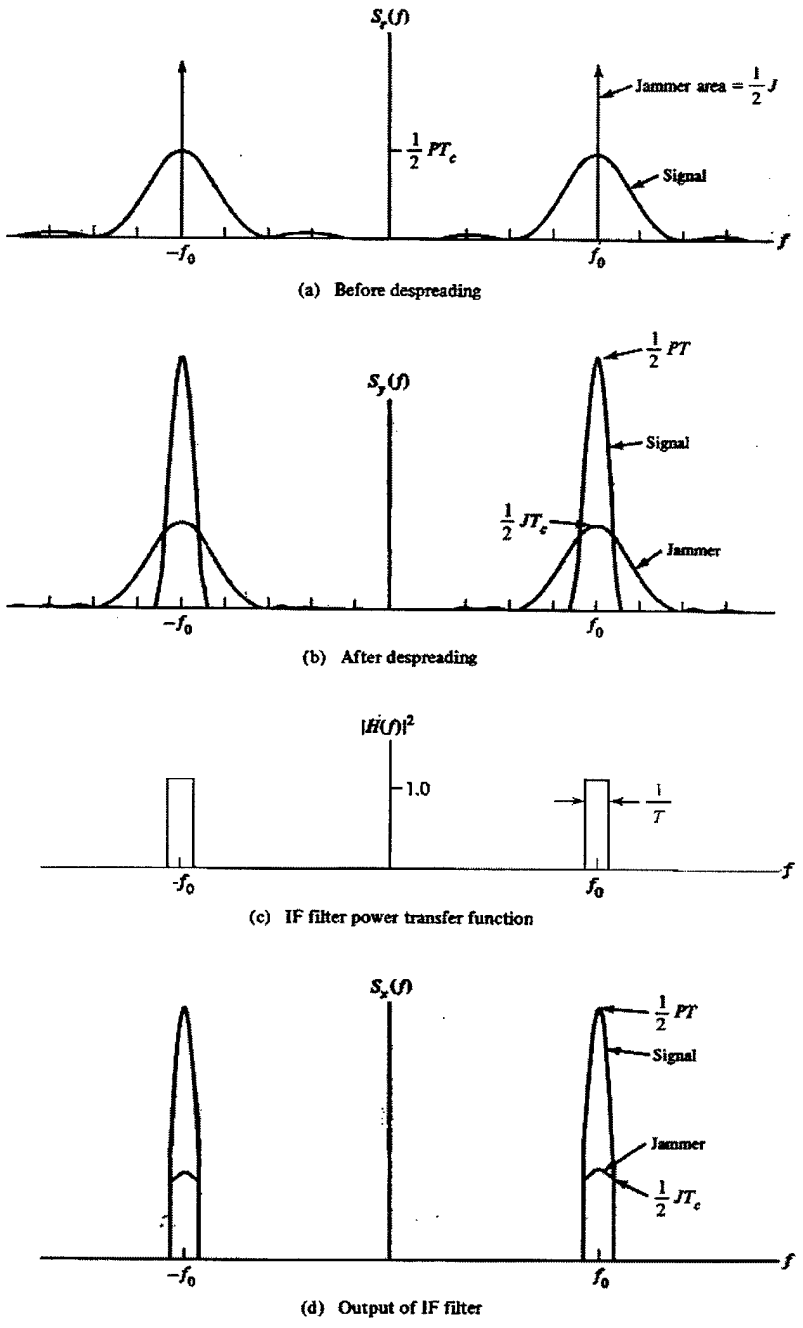


Figure 2-6. DSSS tone jammer effects [34]

Part (a) of the figure shows a tone jammer broadcasting at the center frequency of the transmitted DSSS signal. All the jammer power (J) passes through the initial bandpass filter of the DSSS receiver. However, in the same way that our original carrier was modulated by the spreading signal, the jamming signal is modulated by the de-spreading process, effectively distributing its power over the same bandwidth as the spread signal. Prior to additional filtering, the jammer power at the center frequency after de-spreading is reduced from its received power by a factor of 1/[chipping rate], illustrated in part (b). Likewise, the de-spread signal power increases by a factor of the [chipping rate]/[data rate]. After filtering (c), the jammer power is additionally reduced.

2.2.3 Processing Gain

The Processing Gain of a DSSS system is a measure of its ability to reject or tolerate interference. In considering the previous tone jamming scenario, the total amount of jamming power makeing it through the demodulation process is dependent on the signal data rate. The data rate determines the bandwidth of signal after de-spreading. For cases in which the chip rate is much higher than the data rate, the jammer power (in the above scenario) getting through the data bandpass filter is approximately the peak jamming power after spreading times the data bandwidth, or:

$$P_{JAMMER\_EFF} = P_{JAMMER\_RECVD} \cdot T_C \cdot BW_{DATA} \tag{2.2}$$

where:

P_{JAMMER_EFF}	=	Effective jammer power (after receiver spreading)
P_{JAMMER_RECVD}	=	Received jammer power (before receiver spreading)
T_C	=	Spreading chip width (seconds)
BW_{DATA}	=	Data bandwidth (Hz), typically twice the data rate

We saw previously that the chipping rate determined how much the jamming power was reduced. The processing Gain (G_P) is defined as a ratio between the chipping period and the data bit period [34].

$$G_P = \frac{T_D}{T_C} \tag{2.3}$$

2.3 GPS Signal Structure

Each GPS satellite broadcasts three signals using two specific center frequencies, L1 (1575.42 MHz) and L2 (1227.6 MHz). The SPS (C/A code) signal and the PPS (P(Y) code) signal are both broadcasted on the L1 frequency in phase-quadrature. Signals in phase-quadrature have carriers that are 90 degrees out of phase with each other. The satellite has the capability of broadcasting either the SPS or the PPS signal on L2, but in practice this frequency has never contained the SPS signal.

As mentioned earlier, SPS uses a family of Gold codes for its PN sequences, at a chipping rate of 1.023 MHz. PPS uses a non-linear spreading code called the P(Y) code, which has a chipping rate of 10.23 MHz. The Gold codes are 1023 chips long, and therefore repeat every millisecond. The P(Y) code repeats each week, giving it a period of 604800 seconds and a unique sequence of 6.1871×10^{12} chips. Non-linear codes are inherently more secure and can be much longer than linear codes. They are more complex to implement, however. Each GPS satellite BPSK modulates all signals with a 50 Hz data message 37500 bits long, which continuously repeats and periodically changes (the data message can be turned off on L2 at the government's discretion). An

entire data message from one satellite is received only after 12.5 minutes of continuous reception. Each satellite is assigned a unique C/A and P(Y)-code.

The SPS signal has a null-to-null bandwidth is 2.046 MHz. The PPS signal's bandwidth, modulated by the P(Y) code, is 20.46 MHz. Thus the PPS signal bandwidth is roughly 10 times the SPS signal bandwidth. Using Equation (2.3), it can be shown that the processing gain for the civilian signal is ~43 dB, where the military signal's processing gain is ~53 dB. The higher processing gain of the P(Y) code contributes to the PPS having a higher tolerance to jamming than the SPS.

All the frequencies previously mentioned can be generated from the fundamental frequency of GPS, which is 10.23 MHz. Table 2.1 shows the integer relationships. Having integer relationships maintains a fixed number of carrier cycles per chip and data bit, and made frequency plans easier to implement with one standard oscillator [33].

Table 2.1. Relationship of GPS Signal Parameters to the Fundamental Frequency

Fundamental Frequency of GPS: f_0 = 10.23 MHz		
Parameter	Relationship	Value
L1 Carrier Frequency	$154xf_0$	1575.42 MHz
L2 Carrier Frequency	$120xf_0$	1227.6 MHz
P code chipping rate	f_0	10.23 MHz
C/A code chipping rate	$f_0/10$	1.023 MHz
Data Rate	$f_0/204600$	50 Hz

The L1 signal's analytical representation generated by the "i^{th}" satellite vehicle (SV), with respect to GPS time ('t') is shown in Equation (2.4):

$$S_{L1i} = \sqrt{2P_t}G_i(t)D_i(t)\cos(\omega_{L1}t) - \sqrt{P_t}Y_i(t)D_i(t)\sin(\omega_{L1}t) \tag{2.4}$$

Where

S_{LIi} = GPS L1-band signal as transmitted ("sent") from SV_i
P_t = Transmitted signal power

$G_i(t)$ = Pseudorandom noise (PRN) spreading sequence (1.023 MHz)
$D_i(t)$ = Data message modulation (50Hz)
ω_{L1} = L1 carrier frequency (1.575.42 MHz)
t = GPS time
$Y_i(t)$ = Encrypted P-Code spreading sequence (10.23 MHz)

This signal representation is similar to Spilker's representation in [33:68] The expression in [14] has the correct lead-lag relationship, but is not exactly in terms of GPS time. The GPS control document [5], Kaplan [14], and Spilker [33] all agree that the C/A code bit transitions occur at the L1 carrier peaks, while the P-Code bit transitions occur at the zero crossings of L1 carrier. Both the C/A code and the P-code reset and synchronize their first bit transition with the reset of GPS time. This reset occurs on 12:00 Midnight on Saturday. Therefore, at "0" GPS time (t = 0), the spreading sequences execute their initial transition, at which time the C/A code carrier should peak, while the P-Code carrier should be at a null. Equation (2.4) satisfies all these requirements. The composite L1 signal and its components are illustrated pictorially in Figure 2-7. In practice, the GPS signal has 154 carrier cycles per P-code chip (and thus 1540 cycles per C/A code chip). In order to illustrate the relationships specified in Equation (2.4), each carrier cycle in Figure 2-7 *represents* 154 actual L1 carrier cycles. The spreading sequences shown are *true* C/A and P-codes starting at zero GPS time for PRN 1. The data transition shown was inserted for illustrative purposes, and would never occur at P-code chip #11. In the SPS and PPS signal plots, the dashed portions represent phase inversion. The data bit transition additionally inverts the spreading waveform modulation. This plot represents an ideal depiction of the signals for showing their relationships. In practice, phase transitions at these frequencies are never as instantaneous as that depicted in Figure 2-7. The composite signal appears similar to a Quaternary Phase Shift Keying (QPSK)

modulated carrier, exhibiting some 90 degree phase shifts in the carrier as opposed to the minimum 180 degree phase shifts of BPSK.

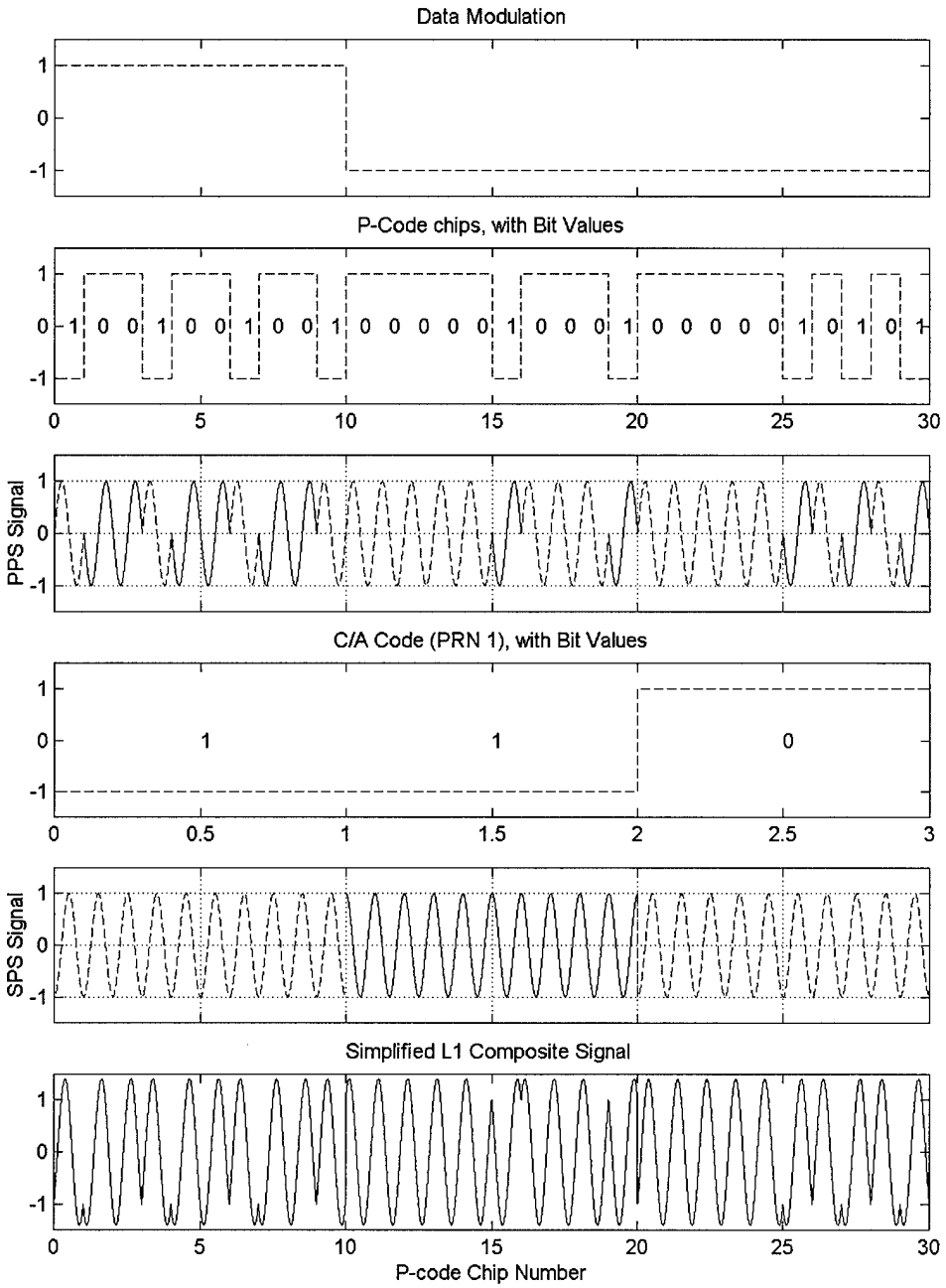


Figure 2-7. Representation of the L1 GPS signal components

2.3.1 *Overview of Signal Power Measurements and Notation.*

Signal power measurements are typically either referenced to one Watt or the relative noise power. The power levels specified in [5] and [14] provide the expected range of received signal power over a satellite's life, tabulated in Table 2.2 below.

Table 2.2. L1 and L2 Minimum received Power Levels [14]

	L1 C/A Code	L1 P-Code	L2 (P-Code or C/A code)
Minimum received power	-160.0 dBw	-163.0 dBw	-166.0 dBw
Maximum expected power	-153.0 dBw	-155.5 dBw	-158.0 dBw
Max. Expected Variation	7.0 dB	7.5 dB	8.0 dB

These measurements are referenced to one Watt. However, absolute signal power is not necessarily meaningful, because it does not consider the noise power. Receiver performance is more dependent on a signal to noise power ratio than the absolute signal power. The Carrier to Noise ratio (C/N_0) is a ratio of total carrier power to the noise power in one Hz of bandwidth. The equation for calculating the effective received C/N_0 is shown in Equation (2.5). This received power measurement is meaningful without specifying the bandwidth of a receiver, and is gives a more generic signal strength characterization.

$$\frac{C}{N_0} = S_r + G_a - 10\log(kT_0) - N_f - L \ \text{(dB-Hz)} \tag{2.5}$$

where:

S_r = Received GPS signal power (dBw)
G_a = Antenna gain toward the satellite (dBic)
$10log(kT_0)$ = Thermal noise density = -204 dBw-Hz
k = Boltzman's constant = 1.38 x 10^{-23} (watts-seconds/K)
T_0 = Thermal noise reference temperature = 290 K
N_f = Noise figure of receiver including antenna and cable losses (dB)

L = Implementation losses plus A/D converter losses (dB)
C = Total received signal power
N_0 = Noise power in a 1 Hz bandwidth

Perhaps the most meaningful measure of signal power when characterizing the performance of a receiver is the signal-to-noise ratio, S/N. This is a ratio of total signal power to total noise power, in the same bandwidth. For a BPSK modulated waveform with a null-to-null bandwidth B_n, an approximation relating C/N_0 to the S/N ratio is in given in Equation (2.6).

$$\frac{S}{N} = \frac{C}{N_0} - B_N \text{ (dB)} \tag{2.6}$$

where:

B_N = Bandwidth of the filter in the receiver to remove out of band noise (in dB)
S = Power of the signal in B_N bandwidth
N = Power of the noise in B_N bandwidth

Un-attenuated signal power varies according to Table 2.2, but received signal power after down-conversion depends on more factors than received signal power. Table 2.3 provides the range of power ratios previously discussed, based on the combination of received power, ambient noise power, antenna gains, and a typical range of receiver losses. The following information was compiled from [14] and [33]. Although the P-Code signal is not implemented in this research, it is included here for comparison. Table 2.3 does not account for signal attenuation by any unnatural interference, and the effects of parameters not listed (satellite elevation, antenna gains, etc.) are lumped into the maximum and minimum received signal power. The C/N_0 measurement in Table 2.3 is the carrier to noise ratio at the antenna, rather than the effective received C/N_0 given by Equation (2.5).

Table 2.3. Estimated range of received power measurements

	Minimum received power and maximum receiver losses		Maximum received power and minimum receiver losses	
Signal	L1 C/A Code	L1 P-Code	L1 C/A Code	L1 P-Code
Received signal power 'C' (dBW)	*-160.0*	*-163.0*	*-150.0*	*-152.5*
Noise floor 'N_0' (dBW)	*-204.0*	*-204.0*	*-205.2*	*-205.2*
C/N_0 (dB-Hz)	**44.0**	**41.0**	**55.2**	**52.7**
Noise Figure 'NF' (dB)	*4*	*4*	*2*	*2*
Receiver Losses 'L' (dB)	*2*	*2*	*1*	*1*
Pre-Correlation C/N_0 (dB-Hz)	**38.0**	**35.0**	**52.2**	**49.7**
Bandwidth 'B_N' (dB/Hz)	*60.1*	*70.1*	*60.1*	*70.1*
Pre-Correlation S/N (dB)	**-22.1**	**-35.1**	**-7.9**	**-20.4**
Processing Gain 'G_P'	*43.1*	*53.1*	*43.1*	*53.1*
Post-Correlation S/N (dB)	**21.0**	**18.0**	**35.2**	**32.7**

The above table provides a reasonable range of pre-correlation S/N ratios for typical GPS receivers. The *italicized* numbers are the high and low estimates previously discussed, where the **bold** numbers are the various power measures based on these parameters. Interference can always cause these levels to go lower, but for the purpose of this research, this table specifies the upper bound on received signal strength. Therefore, the highest pre-correlation S/N ratio that is considered reasonable is –7.9 dB.

2.4 Traditional GPS Receiver Designs

Categorizing every GPS receiver into any list of standard types is a difficult, if not impossible process. Receiver designs have evolved greatly over the 20+ years since the first design. The numerous application areas of GPS have generated a rich diversity of

receiver requirements that have resulted in the wide variety of GPS receivers available today. At the risk of excluding current designs, a "traditional receiver" is defined here as one that digitizes a down-converted signal prior to any processing, and tracks multiple GPS signals simultaneously. In Chapter 1, a typical receiver layout was given in Figure 1-1. This layout is expanded in Figure 2-8 to include block diagrams of each task. This discussion of traditional receiver designs examines each of these blocks in turn.

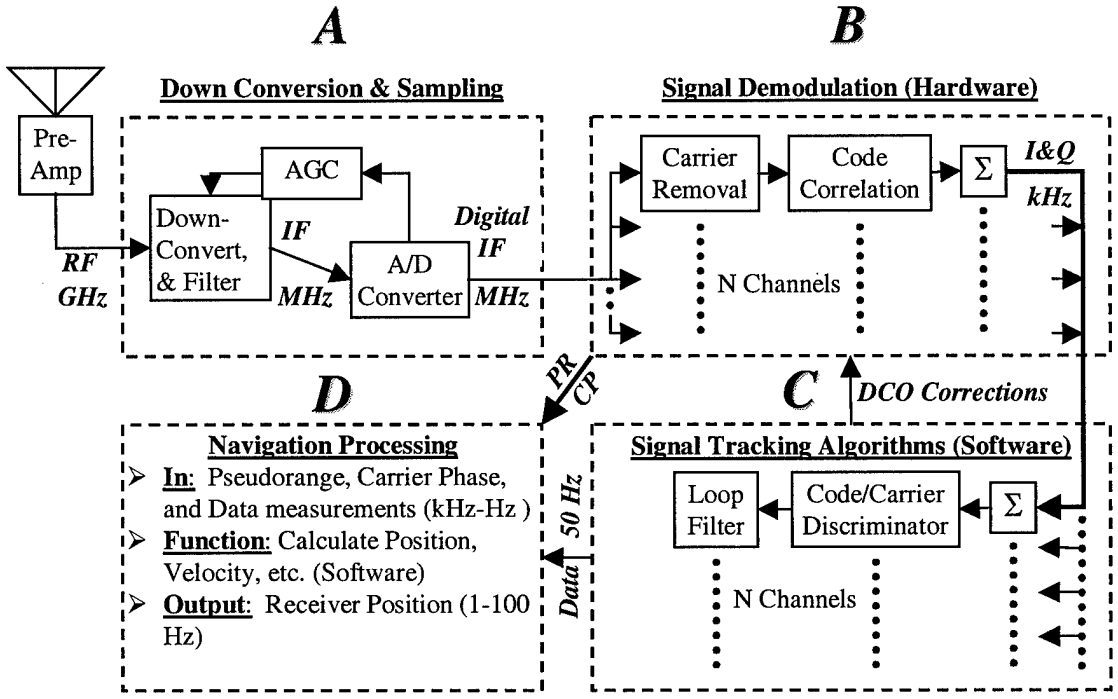


Figure 2-8. Layout of a typical GPS receiver.

The following discussion pertains to processing a single frequency and spreading code, although the structure depicted in Figure 2-8 could be duplicated for both GPS frequencies and spreading codes. In addition, these receiver designs include only the components required for basic GPS positioning. Effort is made to keep the discussion as

generic as possible and cover the most common components among most receiver designs. Much of the information presented here is summarized from [14] and [33].

2.4.1 *Signal Down Conversion and Sampling*

High carrier frequencies are necessary to propagate signals from space through the atmosphere. However, these high frequencies are not easy to filter and demodulate. Downconversion via cosine mixing is used to reduce the received carrier frequency to a more manageable frequency. Typically, several mixing operations are accomplished before reaching the final Intermediate Frequency (IF). Equation (2.7) depicts the process analytically, and includes errors induced by the mixing process.

$$\begin{aligned} S_{IF1} &= S_{GPS}\cos(\omega_{L1}t)\cdot 2\cos(\omega_{LO}t) \\ S_{IF1} &= 2S_{GPS}\left(\frac{1}{2}\cos((\omega_{L1}+\omega_{LO})t)+\frac{1}{2}\cos((\omega_{L1}-\omega_{LO})t)\right)+ \\ &\quad \text{harmonics} + \text{LO feedthrough} + \text{image noise} \end{aligned} \tag{2.7}$$

where:

S_{IF1} = Signal at a frequency of IF_1 after mixing and filtering
S_{GPS} = GPS received signal @ ω_{IF} (includes modulation components)

When down-converting the desired signal, the frequency difference term specifies the generated IF frequency and filtering is done to remove any other frequencies from the signal. Harmonic frequencies are generated in the non-linear mixing process, but are generally much higher than the IF frequency and are easily filtered out. Local oscillator (LO) feed through is a small local oscillator frequency component that "leaks through" the demodulation process. Image noise is caused by noise energy at the image frequencies of the mixing process, located at $\omega_{L1}-2\omega_{LO}$ and $\omega_{L1}-2\omega_{IF}$. The derivation of how these image frequencies propagate through the mixing interference is shown

dbelow; the final image noise terms are given in Equation (2.12). S_{IN2} is the negative image frequency, which can be thought of as the aliased IF frequency. When down-converting a signal, the IF frequency is defined as:

$$\omega_{IF} = \omega_{L1} - \omega_{LO} \tag{2.8}$$

Now, define the noise at the image frequencies mentioned above with noise "amplitudes" n_1 and n_2, and mix these noises with the local oscillator to get:

$$\begin{aligned} S_{IN1} &= n_1 \cos((\omega_{L1} - 2\omega_{LO})t) \cdot 2\cos(\omega_{LO}t) \\ S_{IN2} &= n_2 \cos((\omega_{L1} - 2\omega_{IF})t) \cdot 2\cos(\omega_{LO}t) \end{aligned} \tag{2.9}$$

Next expand the mixed noise expressions using trigonometric identities:

$$\begin{aligned} S_{IN1} &= 2n_1\left(\frac{1}{2}\cos((\omega_{L1} - 2\omega_{LO} + \omega_{LO})t) + \frac{1}{2}\cos((\omega_{L1} - 2\omega_{LO} - \omega_{LO})t)\right) \\ S_{IN2} &= 2n_2\left(\frac{1}{2}\cos((\omega_{L1} - 2\omega_{IF} + \omega_{LO})t) + \frac{1}{2}\cos((\omega_{L1} - \omega_{LO} - 2\omega_{IF})t)\right) \end{aligned} \tag{2.10}$$

Next substitute in Equation (2.8) and simplify:

$$\begin{aligned} S_{IN1} &= n_1\left(\cos((\omega_{IF})t) + \cos((\omega_{L1} - 3\omega_{LO})t)\right) \\ S_{IN2} &= n_2\left(\cos((-\omega_{L1} + 3\omega_{LO})t) + \cos((-\omega_{IF})t)\right) \end{aligned} \tag{2.11}$$

After bandpass filtering at ω_{IF}, this reduces to:

$$\begin{aligned} S_{IN1} &= n_1 \cos(\omega_{IF}t) \\ S_{IN2} &= n_2 \cos(-\omega_{IF}t) \end{aligned} \tag{2.12}$$

Image noise can be avoided by filtering the image frequencies before the mixing process. The frequency plan of a receiver specifies all the previously mentioned frequencies (LO, IF, carrier, image, and harmonic frequencies) so that only the desired signal is present and the other frequencies are rejected by filtering. The frequency plan

design is key to maintaining signal quality at the final IF. The performance of the filters used in the down-conversion process is also key to removing out-of-band interference, thereby increasing the signal-to-noise ratio at the final IF.

Once the signal has been down-converted and filtered, it is ready to be sampled. Sampling performs analog-to-digital conversion of the GPS signal. The analog signal is typically quantized using 1 to 5-bit resolution. As shown in [33], 5-bit resolution results in only 0.5 dB of loss and more bits do not yield appreciable improvement. An Automatic Gain Control (AGC) circuit is used to maximize sampling resolution. The AGC scales the received signal spectrum so that the A/D converter uses the full resolution of the A/D converter as the received signal and noise power fluctuates. The sampling frequency can be selected to convert the IF frequency directly to baseband or to some residual frequency offset. The sampling frequency can also alias the signal to some residual frequency offset by sampling the final IF frequency at less than the Nyquist rate. This type of sampling has the effect of causing a frequency and phase reversal, due to the high-side mixing process that occurs. This process occurs in the Mitel chipset [21] and is detailed in Chapter 3.

2.4.2 Digital Signal Demodulation

After downconversion and sampling, the digitized signal contains the GPS spreading and data modulation at a reduced (or zero) IF frequency. Demodulation is the process of removing any residual carrier and the spreading modulation via code correlation. Typically, the signal is split into its In-phase (I) and Quadrature-phase (Q) components, either in the final down-conversion before sampling, or in the final residual

frequency offset removal. The I and Q samples are necessary to implement digital signal tracking algorithms.

Carrier removal is typically accomplished one of two ways: cosine mixing or phase rotation. Cosine mixing is the same process previously covered for signal down conversion, except here it is performed digitally. Phase rotation is detailed in [33] and uses an algorithm that removes the high frequency component that would result from cosine mixing. Either process requires a digital cosine wave generated at the estimated IF signal phase or frequency. This cosine wave is generated by a Digitally Controlled Oscillator (DCO).

Code correlation is the process of de-spreading the signal, mentioned earlier in the DSSS discussion. Here the Pseudo-Random Noise (PRN) code used by the satellite is locally generated and correlated with the received signal. When the locally generated sequence is aligned (within one chip) with the received signal, the GPS signal is de-spread. The spreading waveform is also generated by a DCO.

As the samples propagate through the carrier removal and code correlation process, they are accumulated and "dumped" (latched) at fixed time interval, usually every millisecond. Because the carrier removal and code correlation process operate on the sampled IF signal—usually clocked at over 2 MHz [33]—this process must be implemented in dedicated hardware for speed. This block's output is a periodic dump of these accumulated high frequency samples, providing summed I and Q values at a lower frequency, which are then processed with embedded computers (software). This embedded software is discussed next.

2.4.3 *Signal Tracking Methods*

The signal tracking method (or loop) of a GPS receiver determines a receiver's highest possible accuracy and its level of robustness. While the highest positioning accuracies are chiefly a result of navigation processing (using differential carrier-phase ambiguity resolution), the signal tracking loop must continuously provide quality PR and Carrier Phase (CP) observables to the positioning algorithms with the precision that the algorithms require. Unfortunately, there is a trade-off between robust and precise tracking loops [57]. Much work has been done to design signal tracking loops to attain both precision and robustness [3,16,39,33,59]. There are two signal components that must be tracked—the spreading code and the residual carrier. Both must be simultaneously tracked in order to de-modulate the received signal. If lock is lost on either the code or the carrier, the receiver will lose lock on the signal and must re-acquire the signal. Of the two signal components, the code is easier to track in dynamic environments, due to the 293 meter equivalent range of one chip length. Carrier frequency lock is harder to maintain than code lock, and carrier-phase lock is the hardest to achieve. As previously mentioned, carrier-phase lock is required to provide the Carrier-Phase (CP) measurements crucial to performing carrier-phase ambiguity resolution. Since a carrier wavelength for L1 is 19 cm, carrier-phase tracking is both very precise and hard to maintain. A phase error of 90 degrees between the received signal and the DCO generated frequency equates to a ranging misalignment of only 4.7 cm. Adjusting the carrier DCO to maintain phase lock on this signal in environments with the large Doppler shifts is very difficult. As a result, phase lock is the weak link in the tracking process.

Code tracking is the process of using the PN sequence autocorrelation function to maintain alignment between the code modulation in the signal and a locally generated replica. Typically, a Delay Locked Loop (DLL) is used, in which the signal is split and fed into multiple correlators with locally generated codes that are offset by fractions of a chip. A Tau-Dither tracking Loop (TDL) is similar to a DLL, but only uses one correlator, which alternates between offset codes. Code tracking loops can be either coherent or non-coherent, which is to say they operate with or without carrier-phase information [33]. Misalignments between the received signal and the locally generated code are sensed by comparing the autocorrelations of the parallel correlators. These autocorrelations are the I and Q accumulated samples generated by block "B" in Figure 2-8. Accumulation here serves as a low pass filtering process, which effectively increases the signal S/N ratio. The I and Q samples generated by Block "B" are typically produced at 1 kHz. For C/A code receivers, the accumulating over 1 ms (analogous to a dumping rate of 1 kHz) equates to a processing gain (G_p) of 30 dB. Accumulating 20 1kHz samples utilizes the full 43 dB of processing gain available, but there is a trade-off between dumping the I and Q accumulated samples at faster rates (with lower S/N) and having observables with higher S/N ratios at a slower rate [14]. The summation box in block "C" of Figure 2-8 accomplishes this additional accumulation process.

Figure 2-9 depicts code tracking with three code replicas spaced at ½ chip increments:

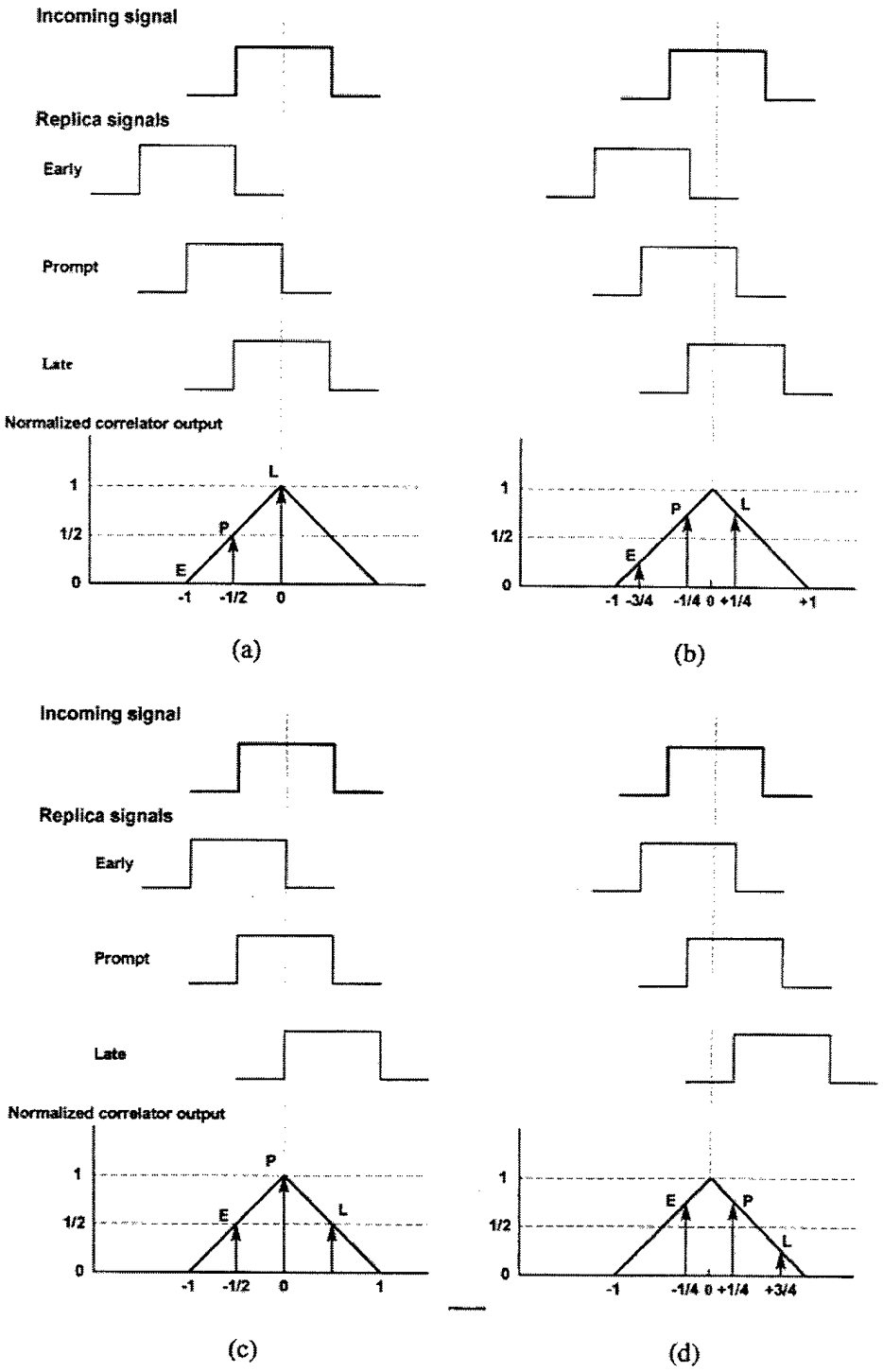


Figure 2-9. Early, prompt, and late code tracking [14]

The signal is tracked so that the three local waveforms are ½ chip late, perfectly synchronized, and ½ chip early when compared to the received signal. The autocorrelation values are run through a discriminator that estimates adjustments to the frequency of all replica waveforms in order to maintain this relationship. A list of code tracking discriminators can be found in [14]. Some other code tracking loops are detailed in chapter 4 of [31].

The received carrier frequency must also be tracked and removed from the sampled signal. The received frequency is constantly changing due to Doppler and receiver clock frequency drift. Carrier tracking can consist of either frequency or phase tracking. Frequency tracking is more robust than phase tracking, but is not as precise [57]. If available, the prompt I and Q values from the code correlation are used as the observables for the frequency or phase discriminator. Both phase and frequency discriminators are listed in [14]. These discriminators provide noisy frequency or phase error measurements between the received signal and the local DCO frequency. As with the code observables, the 1 kHz I and Q measurements generated in block "B" of Figure 2-8 are summed again, to increase the signal S/N ratio, thereby providing a better estimate of the phase or frequency error at the output of the discriminator. However, if the integration time is too long in periods of high dynamics, the tracking loops are not able to adapt fast enough to track the signal dynamics. This is another trade-off design point for a GPS receiver [14].

It was mentioned that the discriminators provide a noisy estimate of the receiver frequency or phase error, due to their low S/N ratios. The loop filter in block "D" of Figure 2-8 performs filtering on the perceived signal demodulation errors. This filter is

key to signal tracking performance. These filters are described well by Ward in [14] and [57]. These filters effectively control how fast the DCO frequencies can change, which in turn controls the dynamic range over which the receiver can operate. However, as the loop filter bandwidth is increased, the DCO corrections also increases, which lowers the quality of the PR and CP measurements.

Because the code tracking process is much less accurate than the carrier tracking process, the carrier tracking loop is often used to aid the code tracking loop [14]. This results in higher quality GPS measurements, but puts more strain on the carrier tracking loop. However, the benefits usually outweigh this drawback and this aiding configuration is commonly used in GPS receivers.

2.4.4 Navigation Processing

This block of the receiver receives the pseudoranges and carrier-phase measurements from the digital hardware and the data bits from the signal tracking loop. This block uses the GPS observables to calculate user position, velocity, and (possibly) acceleration. This process is usually accomplished by an iterative least-squares algorithm or a Kalman Filter [18,19]. The navigation algorithms used to calculate these quantities are not covered in this thesis. However, it is important to understand know which quantities are needed and where they originate—from inside blocks "B" and "C" in Figure 2-8.

The PR measurement comes from the prompt digital correlator of a channel. It is the speed of light times the time delay (computed from the receiver time—in seconds) that the local PN sequence is slewed in order to correlate perfectly with the received

signal. The Carrier-Phase (CP) measurement is the difference between the actual carrier DCO cycle count over a specified period and the nominal count (zero Doppler and clock error) over the same period. This measurement is fundamentally the integration of the perceived Doppler frequency, which provides a measure of the relative motion along each satellite line of sight path. This precise phase measurement is needed for carrier-phase ambiguity resolution. The data bits are typically estimated in the signal tracking algorithm section, from the processed I and Q samples.

2.5 Advanced GPS Receiver Designs

Since the first GPS satellite went into orbit, the signals broadcasted by the satellites have not changed. The services provided by GPS have remained relatively constant, but the technology developed to utilize them has grown remarkably over the last 20 years [4]. There are more advanced receiver technologies than those covered here, and this research does not attempt to touch on all of them. The techniques listed here are those that have the potential to aid a specific application of GPS—maintaining very high positional accuracy in highly dynamic environments with signal blockage, attenuation, and jamming. Of the technologies available and under development for this application, three are mentioned here: Direct Correlator Output Processing (DCOP), Ultra-Tight GPS/INS integration, and frequency domain tracking.

2.5.1 Direct Correlator Output Processing:

Most modern receivers have multiple parallel tracking channels, which independently demodulate GPS signals digitally after sampling a down-converted intermediate frequency. There is typically a trade-off in GPS designs between achieving

position accuracy and providing robustness in reception environments with high-dynamics and interference. It is currently difficult to maintain carrier-cycle level positioning in environments of high-dynamics, intermittent signal blockage, and low signal strengths. Direct Correlator Output Processing (DCOP) is a receiver architecture which is capable of maintaining carrier-phase signal tracking with periods of signal blockage and attenuation. DCOP architectures jointly process the received signals while still expressed in their sampled in-phase (I) and quadrature-phase (Q) components directly after accumulation. This technique uses an Extended Kalman Filter [19] to track the satellite signals received jointly, while taking advantage of the Kalman Filter's ability to estimate and remove correlated dynamic errors projected onto the line-of-sight signal propagation paths. There is also a net increase in signal power from combining signal components together prior to processing [39]. Using a Kalman Filter[18] for signal tracking approaches optimal signal tracking methods, and by jointly processing the signals, a GPS receiver can potentially eliminate cycle slips. This technique is capable of maintaining lock on the carrier-phase of every received signal even during short signal outages of some received signals [43]. Applications for this technology can be found in situations where robust sub-meter positioning systems are required in highly dynamic environments prone to signal blockage and attenuation [41]. Inertial aiding could be added as well, greatly boosting robustness in the same environments.

DCOP involves processing the raw In-phase (I) and Quadrature-phase (Q) signal components of each received signal directly in an optimum fashion. The I and Q signal components are those available at the GPS receiver accumulator output, after carrier removal, code correlation, and accumulation. The group of I and Q signal components

from each received signal are utilized collectively as the raw observables to an Extended Kalman Filter (EKF) [19], a frequently used first-order nonlinear estimation algorithm. The EKF jointly processes these quantities, taking advantage of correlated tracking errors among the received signals, caused mostly by receiver dynamics. As a receiver's antenna undergoes dynamic movement, each satellite signal incurs a Doppler shift due to the receiver's dynamic motion. With DCOP, these correlated errors are estimated and removed by jointly processing all signals, taking into account the transmitters' and receiver's geometry to determine the component of receiver motion on each line of sight (LOS) signal path. This method effectively combines the signals together to improve the pre-detection signal-to-noise ratio, resulting in better performance, especially in the aforementioned reception environments [39].

2.5.2 *GPS/INS Integration Techniques*

An Inertial Navigation System (INS) provides very accurate measurement of high dynamics, but is subject to long term drifts in position accuracy. GPS has excellent long-term positioning accuracy, but lacks very accurate measurement of position during high dynamics. Integrating these two navigation systems combines the best of both systems, yielding good positioning performance in high dynamics and over long time periods. There are typically three different architectures for integrating these systems [35], loose, tight, and ultra-tight integration.

Loose integration is the optimal combining of GPS-calculated position with an INS-calculated position. Tight integration optimally combines INS measurements with GPS pseudoranges and possibly carrier-phase measurements to calculate position. Ultra-

tight integration goes another step further into the GPS receiver and optimally combines processed GPSI and Q signal components with INS measurements to calculate position. One direct application of this thesis is providing a GPS I and Q truth model for simulating an ultra-tight GPS/INS positioning system. In ultra-tight coupling, the GPS signal is broken down into its sampled, digitally processed, in-phase *(I)* and quadraphase *(Q)* signal components. The digital signal components are first generated by down-converting the RF signal to an intermediate frequency (IF) and sampling. These samples are then digitally correlated and demodulated by the receiver's code and carrier DCOs, which are controlled by the INU-computed position and velocity to maintain code alignment and carrier lock. Simulating this type of integration requires the simulation of previously mentioned I and Q measurements as generated from a receiver. This research provided this capability for a fellow student's thesis [35], making possible the simulation of ultra tightly integrated INS/GPS architectures. This level of integration also allows DCOP architectures to be combined with INS at a very low level, with potentially the highest attainable level of non-differential positioning accuracy currently possible.

2.5.3 Frequency Domain Tracking

One method for providing GPS positioning in high dynamic environments is with frequency domain tracking techniques. Frequency domain tracking involves using the Fast Fourier Transform (FFT) to convert the GPS signal into the frequency domain where Doppler frequency estimation is performed for signal demodulation. Hurd, while working under a NASA contract, filed a patent on this concept in 1983 [12]. Although the patent is for the concept and is not based on any realization of the technology, it

claims anticipated performance of being able to track accelerations up to 50 g, with an expected tracking error of only 0.88 meters (added to the pseudorange errors) while receiving a signal 7.6 dB lower than the minimum specified satellite signal strength. Even if these claims are met, the potential accuracies of this technique are still much less than that of DCOP, especially when DCOP is combined with carrier-phase ambiguity resolution. This technique should be capable of tracking higher dynamics than DCOP, but with less accuracy.

2.6 GPS Error Sources

A distinction must be made between the errors that affect position accuracy, and those that affect GPS signal tracking. For example, errors that are slowly changing over time do not adversely affect the tracking performance of a GPS receiver, but they do affect the calculated position accuracy. An exploration of all the errors in GPS is warranted, but this research focuses on those errors that are important to model when comparing receiver designs.

According to Parkinson, [33] there are six classes of ranging errors, which are presented below in Table 2.4:

Table 2.4. GPS ranging error classes [33:478]

Ephemeris data	Errors in the calculated satellite location
Satellite clock	Errors in the transmitted clock, including SA
Ionosphere	Errors in the pseudorange corrections caused by ionospheric effects
Troposphere	Errors in the pseudorange corrections caused by tropospheric effects
Multipath	Errors caused by reflected signals entering the receiver antenna
Receiver	Errors in the receiver's measurement of range caused by thermal noise, software accuracy, and interchannel biases

Ephemeris errors are satellite position errors. In the data message from each satellite, there is a complete set of orbital parameters and refinement factors from which satellite positions are calculated. The error in this calculated position is due to satellite clock error and the quality of the orbital refinement factors provided by the GPS control segment. Ephemeris parameters are optimized for 4-hour time windows and will generate large satellite position errors if used outside this time window [36]. Ephemeris errors can be removed in post-processing applications if precise orbit information is obtained. Precise orbits are available for download from several organizations, including the international GPS service [13]. This error is the epitome of a slowly changing bias, and thus does not affect signal tracking performance.

Each GPS satellite carries an atomic clock with which to keep accurate time. These clocks are watched closely and maintained by the control segment to be within one μs of Universal Coordinated Time, (UTC, modulo 1 second) [5]. However, this accuracy is skewed by Selective Availability (SA). SA has a dominant effect on both position accuracy and receiver tracking. It inserts an unpredictable acceleration component into the tracking loop that limits the minimum loop bandwidth, thereby limiting the loop's tracking accuracy. Its contribution to SPS position error is larger than any other error term. SA can be removed by using differential GPS or by-passed with PPS. As stated in Chapter 1, it is assumed SA is removed for the purposes of this thesis. Because satellite clock errors are independent from satellite to satellite, they affect traditional and advanced receiver designs equally and are not modeled in this research.

The speed of any electromagnetic wave through a vacuum is the speed of light. However, a GPS signal must travel through the ionosphere and troposphere before it

reaches a GPS receiver (on earth). These mediums affect the propagation speed of a signal, resulting in a time-of-arrival error at the receiver. While the troposphere causes a net delay on the GPS signal, the ionosphere affects the signal code and carrier differently. The ionosphere causes a delay on the pseudonoise code, but an *advance* on the carrier. The phenomenon is known as carrier advance or code/carrier divergence, and is explained mathematically by Jorgensen, reprinted in [17]. Ionospheric and tropospheric errors exhibit some correlation between received signals, depending on whether the signals under question traveled through similar atmospheres on their way to the receiver. Signals that propagate through the same elevation angles typically share the same ionospheric and tropospheric errors. These errors can be also characterized as noisy, slowly changing biases, but do not greatly influence GPS signal tracking performance. However, since the errors are somewhat correlated, the DCOP receiver design has the potential of performing better in the presence of this error over an un-coupled approach, but time constraints prevented its implementation.

Multipath errors occur when the GPS signal is reflected and then received by the antenna. Multiple copies of the transmitted signal, attenuated and slightly retarded in time in relation to the direct signal, are present in the tracking loop. The tracking loop tries to track the resultant sum of the multiple signals, causing errors in the time-of-arrival estimation by the tracking loop [6]. Multipath errors are unique to each receiver, and uncorrelated between signals. Some receiver architectures are specifically designed to combat multipath [6], in which case modeling this error is essential to ensuring a fair comparison between receiver designs. However, no assertion has been made that a DCOP receiver performs better or worse in the presence of multipath than traditional

tracking loops, therefore it was not modeled in this research. Please see the recommendations in Chapter 5 for suggestions regarding its implementation.

Receiver errors can be divided into two categories. The first includes those errors and signal power losses associated with the type of implementation chosen. These effects include the AGC finite response time effects, sampling and quantization losses, losses in the code tracking design, and hardware resolution limitations. The second type of receiver errors originates from the practical limitations of component accuracy. These errors include receiver clock errors, filter-induced distortions on the signal, and inherent system noise. Typically, these errors are not completely minimized due to complexity and cost requirements.

Receiver errors are very important to this research. Fortunately, simply configuring the developed receiver model to match the chipset design of interest determines most errors that are implementation dependent (those listed above as the first type of errors). The second type of errors are difficult to model, but are usually comparable among all receiver designs (more so for similarly priced receivers). However, not all errors of this type can be dismissed due to their universal impacts, because some advanced receiver designs have implemented special techniques to reduce their effects and so a fair comparison warrants their modeling.

Table 2.5 provides a quantitative overview of the errors that have been discussed here. True GPS position accuracy cannot be fully understood without a treatment of Dilution of Precision (DOP) calculations. For the purposes of understanding the table, Horizontal Dilution of Precision (HDOP) is the horizontal equivalent positioning error for a particular satellite geometry, and Vertical Dilution of Precision (VDOP) is the

vertical equivalent positioning error for a given satellite geometry. A more in-depth discussion is beyond the scope of this research, but can be found in [14] and [33]. User Equivalent Range Error (UERE) is the Line-of-Sight (LOS) error in the distance from the receiver to the satellite.

Table 2.5. Standard GPS error tables [33]

	Standard Positioning Service (SPS) (one-sigma error, m)			**Precise Positioning Service (PPS) (one-sigma error, m)**			**Modeled in this research?**
Error Source	Bias	Random	Total	Bias	Random	Total	Yes/No
Ephemeris Data	2.1	0.0	2.1	2.1	0.0	2.1	No
Satellite clock	20	0.7	20.0	2.0	0.7	2.1	No
Ionosphere	4.0	0.5	4.0	1.0	0.7	1.2	No
Troposphere	0.5	0.5	0.7	0.5	0.5	0.7	No
Multipath	1.0	1.0	1.4	1.0	1.0	1.4	No
Receiver measurement	0.5	0.2	0.5	0.5	0.2	0.5	Yes
UERE, rms	20.5	1.4	20.6	3.3	1.5	3.6	
Vertical 1-σ errors, VDOP = 2.5			51.4			8.3	
Horizontal 1-σ errors, HDOP = 2.0			41.1			6.6	

2.7 Receiver Clock Performance Analysis

Perhaps the most prominent error source of a receiver is the oscillator. Receiver oscillator performance greatly impacts the overall receiver performance. Table 2.6 lists some of the oscillator effects complied by Vig [55]. Oscillator (or clock) errors in a receiver are inherently correlated between the received signals, and since these errors have such a large effect on receiver performance, they must be modeled to provide a fair comparison of the DCOP advanced processing architecture. Portable GPS receivers typically employ some type of a crystal-based oscillator, since they are lightweight, relatively cheap, small, and have acceptable stability. The discussion therefore focuses on crystal oscillators.

Table 2.6. Impacts of oscillator performance on GPS receivers.

Oscillator parameter	GPS performance parameter
Warmup time	Time to first fix
Power	Mission duration, logistics costs (batteries)
Size and weight	Manpack size and weight
Short term stability (0.1 s to 100 s)	Δ-range measurement accuracy, acceleration performance, jamming resistance
Short term stability (~15 min.)	Time to subsequent fix
Phase noise	Jamming margin, data demodulation, tracking performance
Acceleration sensitivity	See short term stability and phase noise effects

Accurate receiver clock modeling is not easily achieved. Crystal controlled oscillators contain a variety of noise components, and the combined effects of these components cannot be observed with classical variance calculations [32]. The Allan variance (AVAR—defined in Equation (3.3)) is one method used to specify clock performance because it can adequately specify a clock's performance statistics over varying time spans. Where the classical variance statistics diverge for some commonly observed noise processes (such as a random walk), the Allan variance converges for all noise processes observed in precision oscillators. Allan variance is easy to compute and is faster and more accurate in estimating noise processes than the FFT [55].

$$\sigma^2_{Allan}(\tau) = \frac{1}{2(n-1)} \sum_{k=1}^{n-1} (y_{k+1} - y_k)^2 \tag{2.13}$$

y_k = Observation at time t_k
$y_{k+1} - y_k$ = Change in observed quantity over time interval τ $(t_{k+1} - t_k)$
n = Number of samples

Various clock errors have been summarized pictorially by [55] and are shown in Figure 2-10. Oscillator drift is defined as "the systematic change in frequency with time of an oscillator [55]." Figure 2-10 details how clock errors vary over different time periods and due to dynamic effects. Temperature variations typically cause large

frequency offsets in quartz oscillators, and must be compensated by applying temperature sensing and error correction, or by "ovenizing" an oscillator to keep it at a constant temperature. Because a crystal oscillator is fundamentally a mechanical precisely vibrating device, vibrations to the oscillator itself cause errors in the output frequency. Additionally, oscillator short-term stability is a key performance parameter for maintaining carrier-phase lock. Long-term clock stability is important to a GPS receiver during periods of off/on operation, as an initial clock offset error impacts the time-to-first-fix. During a receiver's operation, long term stability is not as crucial, because of the stability of the satellite clock. Shock and 2-g tip-over errors are again caused by the mechanical nature of crystal vibration. Radiation errors typically only apply to satellite clocks, and turn-on to turn-on biases affect initial signal reacquisition the most, after which they add little or no net effect to receiver clock errors.

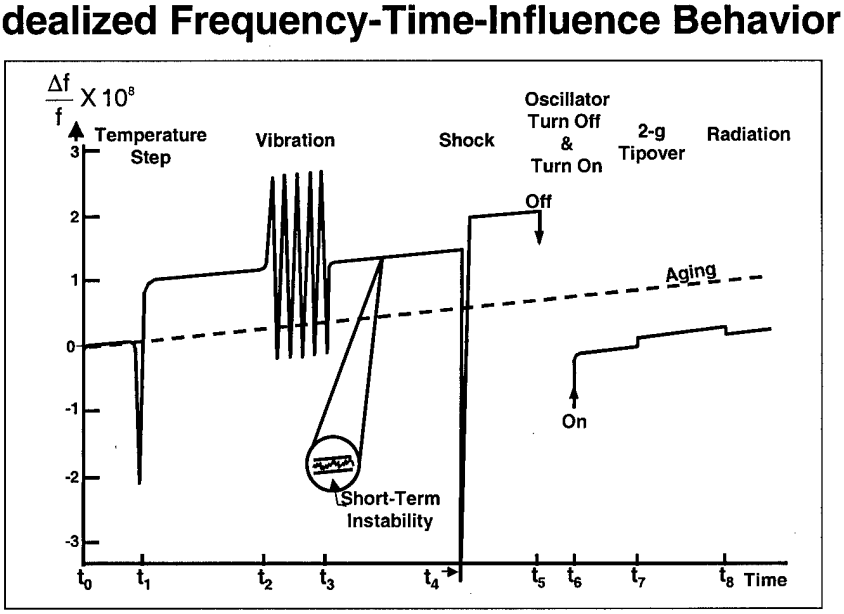


Figure 2-10. Example clock error characteristics [55]

In comparing phase noise errors, short-term frequency stability, and long-term frequency drift, one might understand why a technique that measures statistical variance over different time periods, such as Allan variance, is useful. An Allan variance plot illustrating the principle of variable frequency variance for a typical crystal oscillator is shown in Figure 2-11. The variance parameters are used as the metric by which the clock error model is defined. The clock error model based on the Allan variance is presented in Chapter 3.

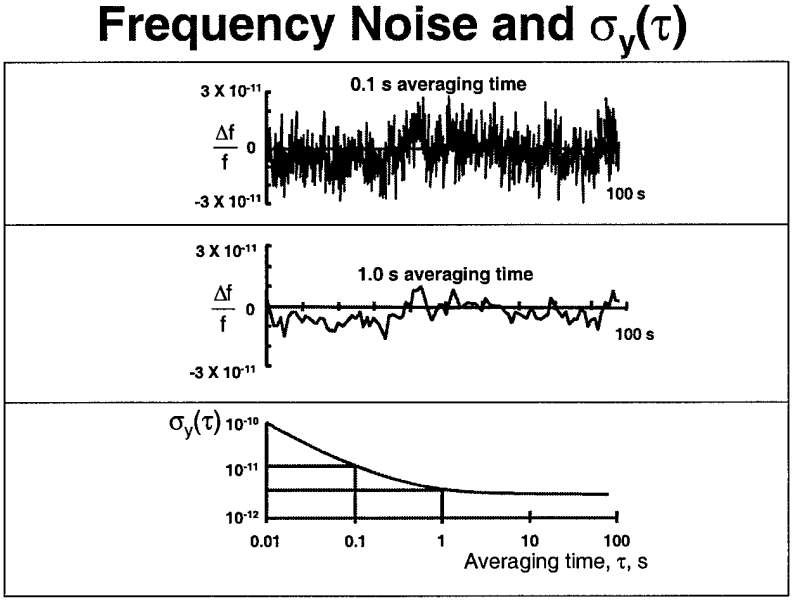


Figure 2-11. Illustration of Allan variance [55]

2.8 GPS Applications and Limitations

Tracking the carrier frequency of GPS signals has enabled differentially corrected receivers to achieve centimeter level positioning accuracies. Performing carrier tracking in stand-alone receivers also significantly enhances the position and velocity accuracies

by using techniques such as carrier smoothing of the code pseudoranges. Extending this level of accuracy to applications in which satellite signals regularly undergo blockage, attenuation, and interference has been problematic, as these environments are prone to cycle slips, which cause jumps in the position, velocity, acceleration (PVA) solutions. A cycle slip occurs when the carrier-tracking loop looses carrier-phase signal lock. When a cycle slip occurs, there is a large drop in the position accuracy, which remains until the carrier cycle ambiguities are again resolved. Harsh signal environments, like those previously mentioned, compound the problem of resolving these cycle ambiguities, thereby elongating the period of reduced accuracy. Examples of such environments include dynamic airborne platforms or metropolitan vehicular applications. The lack of robust carrier-phase positioning techniques has prevented their use in these environments.

While code tracking and frequency domain tracking techniques have demonstrated improved performance in these environments, they do so with the cost of higher position and velocity errors. For some applications these larger errors are acceptable, but nearly all these applications would benefit from increased accuracy. One application that could benefit from advances in carrier-phase tracking robustness is test range truth systems. As navigation systems grow more accurate over time, better truth positioning systems must be developed in order to accurately test these new systems accurately. A general rule of thumb in navigation test ranges is to have truth reference system be an order of magnitude more accurate than the system under test. For the next generation of navigation systems, this will become harder to accomplish. In addition, since the next generation of navigation systems will undoubtedly utilize GPS, and because the jamming susceptibility of these navigation systems is of high interest to the

armed forces, the truth positioning systems for test ranges must be able to operate in conjunction with GPS jamming testing.

Another concern in the advent of commercially available GPS jammers is the effectiveness of our GPS-guided munitions in a jamming environment. Techniques that increase a GPS receiver's robustness in a jamming environment are of great value to the United States (US) armed forces. Many US weapon systems rely on GPS, and most (if not all) of the aircraft in the US arsenal employ GPS as a navigation and/or weapon delivery system. Among the munitions that use GPS for their navigation are the GBU-31/32 Joint Direct Attack Munition (JDAM), the AGM-86C Air Launched Cruise Missile (ALCM), the AGM-158 Joint Air-to-Surface Stand-off Missile (JASSM), and the AGM-154 Joint Standoff Weapon (JSOW) [53].

2.9 Summary

This chapter has presented the fundamentals of GPS, DSSS communication systems, GPS signal structure, some traditional and advanced GPS receiver topologies, signal tracking methodologies, GPS error sources, crystal oscillator characteristics, and some applications of this research. With this diverse background, the next chapter delves into the signal generator model derivations, the receiver hardware, and an analytical I and Q simulator. The reader should be well prepared to understand the methodology and utility of the design process presented.

3. *Modeling Methodology*

3.1 Overview

This chapter details the development of the models used in this research. The models developed by this research are grouped into three independent products: (1) A digitized IF GPS signal simulator, (2) a realistic receiver signal processing model, and (3) a GPS accumulated I and Q model. The first and third tools were written in Matlab® and the receiver signal processing model was implemented in Simulink®.

The IF GPS signal simulator (SS) simulates the downconverted, filtered, and sampled spectrum of received GPS signals and noise. The receiver signal processing model (RM) is a complete implementation of the hardware Digital Signal Processing (DSP) processes that typically occur in a GPS receiver. These processes include carrier removal, code correlation, and accumulation.

The final model is an equation-based approximation of the other two models combined. The final product generated by the SS and RM when used together is the accumulated I and Q samples. The I and Q model takes in the same truth environment as the SS and the same DCO inputs as the RM to estimate the accumulated I and Q samples. This high-level, equation-based I and Q model is much faster at generating accumulated I and Q values than the SS and RM, but with less modeling accuracy. This model is hereafter referred to as the I and Q model (IQMDL).

3.2 Digitized IF GPS Signal Simulator

As previously mentioned in Chapter 1 (Approach Section), the location inside a GPS receiver at which to start modeling the GPS signal was chosen to be the IF sampling point. This is the point in a true receiver at which the analog spectrum of signals and noise becomes digital. This sampling process is naturally suited to be approximated in Matlab®, as continuous-time equations are "sampled" at the discrete points of a time vector. This is the same location at which NavSys's Correlator model generates its signals [28, 49].

With the digital computing power currently available, any modeling and simulation of a signal prior to sampling, while necessary and beneficial for some applications, is a digital approximation of an analog process. By modeling the signal where it is sampled in the receiver front end, one can avoid the time-intensive and computationally expensive analog signal modeling while increasing the potential for creating accurate and equivalent models of the real hardware. Some effects, such as the filtering of the received spectrum in the down-conversion, cannot be as accurately portrayed in the initial "sampling" of the continuous models, but digital filters and discrete Fourier techniques can approximate these effects after the sampled vector is created. If the modeling of the signal is accurately portrayed at this point, any subsequent processing can be easily and accurately equated to real DSP processing used inside working receivers. Any true digital process could be implemented in Matlab® accurately, since Matlab® is a digital processing environment.

The SS was designed to be reconfigurable, to model many types of single-frequency front-ends. As mentioned in Chapter 1, this research focused on modeling the

Mitel GPS Receiver Front End [22]. Although modeling the P-code signal is not included in this research, most of the necessary modeling is a scale factor different than that done for C/A code, so adding this capability should only require modification of the IF filter bandwidth, the frequency plan, and the addition of a P-code generation routine.

3.2.1 Signal Model Derivation

The theoretical basis for the down-converted IF GPS signal is based on the following analytical development. For the sake of clarity, start with the single *ideal* GPS L1 signal as presented in Chapter 2, with the exception of a new time variable. The 'T' in Equation (3.1) is the *true GPS system time*, introduced here for notational purposes. This ideal time scale is referred to as 'system time' in the derivation.

$$S_{L1i} = \sqrt{2P_i}G_i(T)D_i(T)\cos(\omega_{L1}T) - \sqrt{P_i}Y_i(T)D_i(T)\sin(\omega_{L1}T) \tag{3.1}$$

To derive an expression for the received signal just *prior to sampling* inside the receiver, Equation (3.1) is analytically propagated through the atmosphere, to the receiving antenna, and through the GPS RF front end. During this analytical propagation, all known observable errors, having a measurable impact on the signal, are included, followed by a discussion on each error's impact on this research. The set of errors is then reduced to those that fall within the scope of this research, and the result of this process is the signal model around which the simulator was written.

The effects of GPS signal perturbations have been studied in detail by the GPS community [6, 14, 33, 38, 55]. They have been excruciatingly analyzed and characterized as to their affect on the traditional GPS observables (pseudorange and

carrier-phase measurements). This development focuses on how these error sources affect the sampled IF representation of the received signal. Equations (3.2) and (3.3) are an analytical approximation of the perturbed received GPS signal at the antenna, prior to any antenna or receiver induced effects. These equations include an error term for all significant error sources in the GPS satellite and propagation environment. There are undoubtedly more signal perturbations than are represented here, but their effects on this level of signal modeling are negligible [33].

$$\begin{aligned} R_{L1_i} = {} & \sqrt{2P_r}G_i(T-\delta t_{iono}-T_d)D_i(T-\delta t_{iono}-T_d)\cdot \\ & \cos\{(\omega_{L1}+\omega_{e_{sv}})(T+\delta t_{iono}-T_d)+\phi_{e_{sv}}+2\pi d\frac{T^2}{2}+\varphi_{sv}(T)\}- \\ & \sqrt{P_r}Y_i(T-\delta t_{iono}-T_d)D_i(T-\delta t_{iono}-T_d)\cdot \\ & \sin\{(\omega_{L1}+\omega_{e_{sv}})(T+\delta t_{iono}-T_d)+\phi_{e_{sv}}+2\pi d\frac{T^2}{2}+\varphi_{sv}(T)\}+n(T)+R_{L1_{MP}} \end{aligned} \tag{3.2}$$

where:

$G_i(\ )$, $D_i(\ )$, ω_{L1}, T $Y_i(\ )$ are defined the same as in (2.4) and (3.1)

R_{L1_i}	=	Received GPS L1 signal from the i^{th} satellite
P_r	=	Received signal power (W)
δt_{iono}	=	Ionospheric perturbation (s)
ω_{e_sv}	=	Non-Stochastic L1 carrier frequency error
ϕ_{e_sv}	=	Fixed L1 carrier-phase error
$2\pi dT^2/2$	=	Frequency drift or aging effect
$\varphi_{sv}(T)$	=	Random phase error (SV Phase Jitter)
T_d	=	Total time delay (see Equation (3.3))
$n(T)$	=	Noise, modeled as Additive White Gaussian Noise (AWGN)
R_{L1_MP}	=	Received reflected signals (multipath)

The R_{L1_MP} term is composed of additional copies of R_{L1i} but with reduced amplitudes and additional delays to simulate the lower power levels and additional distance traveled by reflected signals (multipath) [6]. The clock error terms were taken from Spilker's clock error model in Chapter 4 of [33]. Three of the clock terms, ω_{e_sv}, ϕ_{e_sv}, and $2\pi dT^2/2$, are not random variables but deterministic quantities. The frequency term is representative

of clock environmental effects such as temperature, magnetic field, acceleration, and pressure [33]; only the $\varphi(T)$ term is random. Notice that the ionospheric error term has opposite signs for the carrier and code expressions. This is due to the ionospheric divergence/carrier advance phenomenon, explained in detail in the first appendix of [17]. The term T_d contains all those terms that affect propagation time (prior to receiver effects). It is broken out in detail in Equation (3.3).

$$T_d = R/c + \delta t_{sv} + \delta t_{tropo} + \delta t_{SA} + SV_{pos_{error}}/c + (\delta t_{MP}) \tag{3.3}$$

where:

R	=	True range from the phase centers of the satellite and receiver antennas (m)
c	=	Speed of light (299792458 m/s)
δt_{sv}	=	Satellite clock error (s)
δt_{tropo}	=	Tropospheric perturbation (s)
δt_{SA}	=	Selective Availability error (s)
SV_{pos_error}	=	Satellite ephemeris error (m)
(δt_{MP})	=	Multipath signal delay (only found inside the R_{L1_MP} term)

It may not be obvious that a Doppler shift is included in Equation (3.2), but the Doppler term is embedded in the product of T_d and ω_{L1}, as T_d contains the instantaneous relative velocity dR/dt along the line-of-site (LOS) path.

Before propagating these errors through the downconversion and sampling process, a reality check must be performed to limit our derivation to those sources that fall within the scope of this research. There are some applications of the SS where most, if not all, these errors need to be modeled. If the SS were used to generate pseudorange and carrier-phase measurements for calculating position, most of the above errors are expected to be present, and as such a navigation algorithm that accounts for these errors would not work correctly in their absence. The above equation is included for these

applications, in the event additional modeling is desired. In this research, the criteria for determining which error sources were modeled is stated below:

- Does the error have a measurable impact on the receiver's tracking threshold?
- Is the error correlated between multiple signals? In other words, does the error have the same or a similar impact on multiple channels in the receiver?
- Does the error vary greatly from one receiver design to another?

In-depth receiver hardware modeling is primarily useful for determining a receiver's tracking performance [50, 57, 58]. Navigation level analysis is not a proposed application of this research; only errors that impact a receiver's signal tracking performance are of interest. One of the research objectives is to provide an accurate truth model for comparing traditional receivers with DCOP equipped receivers. Since DCOP has the capability of removing correlated errors between satellites, a fair comparison must include errors that fall into this category. Since a research goal is to support comparisons of other new receiver designs, even those not yet known, receiver-induced errors are modeled as well.

Applying the above criteria, the satellite clock errors are not modeled, nor are tropospheric, ionospheric, multipath, or SA induced errors. It is postulated for the purposes of this research that these errors are either mainly uncorrelated between the received signals or have similar effects on receiver *tracking performance*. However, adding error models for the errors listed above would greatly increase the potential application areas of this research.

Ionospheric and tropospheric errors *are* somewhat correlated, as a function of the difference in elevation between the received signals, but they can be characterized as a noisy but slowly changing time delay (advance, for carrier-phase). Although slowly

changing time delays can have appreciable effects on position error, they do not add additional stress to the tracking loops, except in the most extreme conditions, when the 11 year sun cycle is at its peak [33].

The phase noise component of the satellite clock errors is a basic limiting factor in maintaining carrier lock at low signal to noise ratios; [16] however, the phase noise (or jitter) of the satellite clock is negligible in the presence of the phase noise of the receiver clock [16]. The other satellite clock errors—frequency drift, GPS time bias, environmental and aging effects, and SA clock dithering—can be removed or greatly reduced by using the clock correction parameters in the ephemeris message and DGPS techniques [33]. Recall that the removal of SA is an assumption of this research stated in Chapter 1.

Multipath error is usually minimized through the careful placement of a multipath limiting antenna. However, some receiver designs have been suggested which remove multipath effects with special receiver processing [6]. If such receivers were to be fairly compared with the tools develop here, a multipath model would need to be added to the SS. However, this error was not modeled in this research due to time constraints.

With the exception of multipath errors, differential applications effectively remove the errors in Equation (3.3) [38]. Therefore, if one assumes the use of DGPS, the modeled pseudorange and carrier-phase measurements can be used to approximate differentially corrected measurements. The net effect is that "corrections" are "applied" at a much earlier point in the process.

It is not intuitively obvious what effect the P(Y) coded signal has on the demodulation process, but its impact on this type of simulation is minimal. The inclusion

of the PPS signal in the derivation would show this analytically, but this has the undesirable effect of doubling the size of each equation in the derivation, making it harder to follow. The justification for removing this signal from the derivation is as follows. The power of the P-code signal is half of the C/A code signal, which is typically –15 dB below the noise floor to start with (see Table 2.3). Also, the same noise spreading properties of any DSSS system (discussed in Chapter 2) spread the P-Code signal in the same manner. Additionally, if the signal simulator filters the incoming signal at the C/A code bandwidth, the band-pass filtering of the signal would filter out nearly all (roughly (1-(2 MHz/20 MHz))x100% = 90%) of the P-code signal power. Given these facts, it should be seems reasonable to drop the PPS portion of the received signal from the derivation.

The resulting received signal model (at the antenna) is a subset of Equations (3.2) and (3.3), and is given by Equations (3.4) and (3.5).

$$R_{L1_i} = \sqrt{2P_r}G_i(T-T_d)D_i(T-T_d)\cos\{(\omega_{L1})(T-T_d)\} + n(T) \tag{3.4}$$

where:

$$T_d = R/c \tag{3.5}$$

Before propagating the expression into the receiver, it is useful to develop some more rigorous notation for expressing time. This is necessitated by the fact that the receiver clock be generating perfect system time. The sign convention that was followed for expressing these clock errors is that used in [14]. Let us define and relate the following quantities:

T = GPS system time (ideal time scale, previously defined)
T_t = System signal transmit time
T_r = System signal received time
$T_d = T_r - T_t$ = signal propagation time (delay)
δt_r = Advance of the receiver clock with respect to system time (receiver clock bias)
$t_r = T + \delta t_r$ = Receiver clock time

Note the sign convention for the receiver clock bias term indicates that a receiver clock that is ahead of system time by 0.005 ms would have a $\delta t_r = +0.005$ ms; this is opposite of the customary definition of an error as the 'true' minus the 'measured.' Note also a subtlety in the true range that is important when calculating perfect pseudoranges; the true range is the distance between the satellite at time T_t and the receiver at time T_r, not the instantaneous distance between the satellite and the receiver at one instant in time. Satellites can have LOS velocities on the order of 800 m/s, causing large (~50m) errors in the calculated 'perfect' pseudoranges if this subtlety is ignored.

A generic representation of a GPS front end is shown in Figure 3-1. This generic representation is the mold that GPS front-ends must fit to be modeled using the SS. The SS currently uses 2-bit (4 level) quantization, for analog to digital conversion. The modeling and effects of more (or less) resolution in the quantization process was not studied in this research (see [33] for this analysis). A dynamic AGC was implemented in Simulink® based on the limited information available in the Mitel literature [21]. It was designed to be a highly reconfigurable so that it could be tuned to portray the AGC operation accurately, should its dynamic performance be fully specified. At a minimum, this allows for the investigation of a receiver's operation in dynamic noise environments.

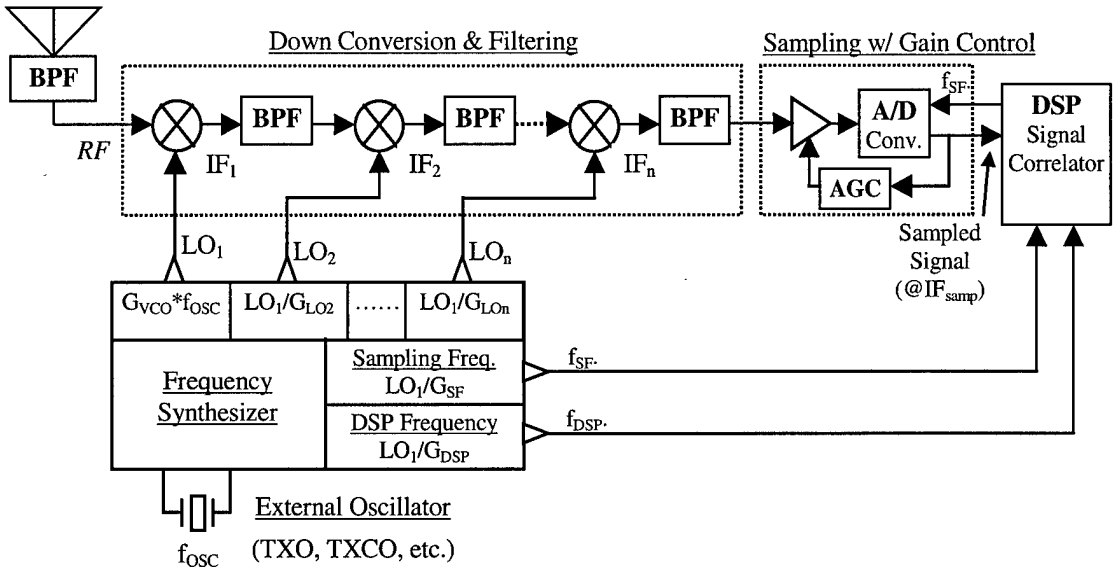


Figure 3-1. Generic GPS RF Front End Model for the Signal Simulator

The process of choosing the number of down-converter stages and the intermediate frequencies is a key design consideration in any GPS RF front-end design. Many factors must be considered and there is no universally accepted "optimal" design. This research does not attempt to model hardware effects such as image frequencies and local oscillator leak-through. The only hardware effects that are modeled are filtering and oscillator errors. Most receivers use crystal oscillators, which are susceptible to large frequency errors in dynamic environments [55]. These receiver oscillator errors have a significant impact on maintaining coherent signal tracking, and are common to all receiver channels, making it an important receiver error to model. The oscillator error model is described in a subsequent section.

Having defined the RF front-end model, Equation (3.4) is now propagated to the final IF frequency prior to sampling. The signal and noise amplitudes are modeled at the final IF expression, and as such, are omitted in the derivation. Since the spreading and

data modulation is not time or frequency shifted by the cosine mixing process, it is also omitted from the following derivation. The emphasis here is to track the time and frequency accurately through the reception process.

Starting with the first local oscillator of Figure 3-1, an expression In terms of receiver time is shown in Equation (3.6):

$$LO_1 = \cos(\omega_{LO_1} t_r) \tag{3.6}$$

The received signal is down-converted through cosine mixing, which is the multiplication of (3.6) and (3.4). Equation (3.4) is represented in terms of receiver time using the previously defined time relationships. Applying the appropriate trigonometric identity and including the relevant terms results in (3.7):

$$IF_1 = LO_1 R_{L1_i} = \cos\{\omega_{L1}(t_r - \delta t_r - T_d) - \omega_{LO_1} t_r\} + \cos\{\omega_{L1}(t_r - \delta t_r - T_d) + \omega_{LO_1} t_r\} \tag{3.7}$$

Filtering out the high frequency component generated by the mixing process, we have (3.8):

$$IF_1 = \cos\{(\omega_{L1} - \omega_{LO_1})t_r - \omega_{L1}(T_d + \delta t_r)\} \tag{3.8}$$

Repeating the process for LO_2 results in (3.9):

$$IF_2 = \cos\{(\omega_{L1} - \omega_{LO_1} - \omega_{LO_2})t_r - \omega_{L1}(T_d + \delta t_r)\} \tag{3.9}$$

Repeating once more for LO_3 gives us our final IF prior to sampling:

$$IF_3 = \cos\{(\omega_{L1} - \omega_{LO_1} - \omega_{LO_2} - \omega_{LO_3})t_r - \omega_{L1}(T_d + \delta t_r)\} \tag{3.10}$$

This expression represents the signal model prior to including any oscillator frequency errors. Such errors can have a significant impact on this expression, and the derivation of

the oscillator frequency errors and their incorporation into Equation (3.10) is given in the next section.

3.2.2 *Oscillator Error Model and Impact Analysis*

The oscillator error model in this research is intended to allow for a reasonable comparison of a receiver's ability to cope with clock frequency drift. The resulting model is not claimed to represent an exact characterization of the combined deterministic and random clock error sources. These include specific force and temperature induced errors, random frequency drift, and phase noise. Of these errors, the clock model used can represent the effects of these errors, with the exception of phase noise. Since the sampled IF signal and carrier DCO signal period is represented by only 4.06 (average) samples in the Mitel chipset [21], this research ignores the effects of Phase noise. The following paragraph presents the rationale.

With 2-bit amplitude quantization and essentially 2-bit phase quantization of the digitized signals, the effects of the phase noise are assumed to have a small effect on the sampled signal, and therefore a minimal effect on receiver comparisons. This assumption is invalid if special hardware to remove phase noise is incorporated into a receiver design, as is the case in [16]. As stated in [16], phase noise has an effect on tracking performance, but it is assumed here the clock error model and other error models have dominant effects. The inclusion of a phase noise model is partially coded in the SS, but it is currently incomplete and 'commented out.' The partial model is retained for future research. Its inclusion in the SS is listed as a recommendation in Chapter 5.

Frequency drift originates in the external crystal oscillator. As previously mentioned in the discussion of equation (3.2), oscillator frequency errors are made up of a deterministic portion and a random portion. According to [55], deterministic vibration (g-force induced) crystal oscillator errors dominate over random errors. For this reason, a clock model of only random oscillator errors is inadequate for testing the effects of oscillator errors on receiver designs. Equation (3.11) shows the output frequency of the external oscillator modeled as a nominal frequency component plus an error component:

$$f_{tcxo} = f_{tcxo\_nom} + f_{tcxo\_err} \tag{3.11}$$

The frequency DCO error in Equation (3.11) represents the cumulative effects of all the oscillator errors modeled. Due to the diverse nature of oscillator errors (see Figure 2-10) a comprehensive model was not developed. The approximation used is discussed following a derivation of a frequency clock error's impact on the sampled signal.

The net effect of the oscillator error is dependent on the total shift in received frequency and is independent of the number of downconversion stages. Aliasing in the sampling process increases the impact of a frequency error, as will be shown. The Mitel design uses aliasing in its sampling process. The derivation of the oscillator impact is done for the Mitel frequency plan, provided in Table 3.1. This table formulates the relationships between the actual local oscillator errors and the frequencies in the chipset. These relationships are key to deriving the impact of the local oscillator error on the sampled signal.

Table 3.1. GP2010 Frequency Plan

Frequency	Symbol	Description	How Produced
10.000 MHz	f_{OSC}	PLL reference signal	10.00 MHz TCXO
1400 MHz	LO_1	1st Local Oscillator	$f_{OSC}x(G_{VCO}=140)$
140 MHz	LO_2	2nd Local Oscillator	$LO_1/(G_{LO2}=10)$
40 MHz	f_{DSP}	GP2021 Clock Signal	$LO_1/(G_{DSP}=35)$
31.111 MHz	LO_3	3rd Local Oscillator	$LO_1/(G_{LO2}=45)$
5.7142 MHz	f_{SF}	Sampling Frequency	$LO_1/(G_{SF}=245)$
Nominal Received and Intermediate Frequencies			
1575.42 MHz	RF	L1 GPS Signal	By the Satellite
175.42 MHz	IF_1	1st Intermediate Freq.	RF- LO_1
35.42 MHz	IF_2	2nd Intermediate Freq.	RF-LO_1-LO_2-
4.3088 MHz	IF_3	3rd Intermediate Freq.	RF-LO_1-LO_2-LO_3
1.4053 MHz	N/A	Final (Aliased) IF	(RF-LO_1-LO_2-LO_3-f_{SF})

Using the relationships in Table 3.1 and Equation (3.11), the oscillator error can be substituted into the expression for the final (Aliased) IF frequency in order to express the final aliased IF frequency as a function of the crystal oscillator and its error:

$$\left[RF - (f_{OSC} + f_{OSC\_ERR}) \left(140 + \frac{140}{10} + \frac{140}{45} + \frac{140}{245} \right) \right] \tag{3.12}$$

The ratios in Equation (3.12) were derived from the relationships in Table 3.1. Substituting in the numbers for the GPS carrier frequency and a 10 MHz nominal oscillator frequency yields:

$$\left[1575.42MHz - 1576.825396MHz - f_{OSC\_ERR} \frac{1576.825396MHz}{10MHz} \right] \tag{3.13}$$

In Equation (3.13), the numerical subtraction of just the first two terms in Equation (3.13) results in a negative frequency. This negative frequency is a result of the high-side mixing process caused by aliasing [23]. The effects of aliasing in general are not

necessarily the same, as they depend on what frequencies are involved. The discussion on aliasing that follows is claimed only applicable to the Mitel chipset.

Simplifying Equation (3.13) and converting the frequency term into a unitless ratio (recall that the nominal frequency of the oscillator is 10 MHz) results in (3.14):

$$\text{Final (Aliased) IF frequency} = \left[-1.405396 - \frac{\Delta f}{f} 1576.825396 \right] MHz \tag{3.14}$$

Where the $\Delta f/f$ term is the normalized frequency error of the oscillator, usually specified in parts per million (ppm). This is the resulting frequency content of the signal after sampling. The frequency observed by the user is the mirror (positive) frequency of this negative frequency. The aliasing occurring in the sampling process reverses the frequency and phase error/offset in the received signal [21]. This impacts both the Doppler and the clock error induced frequency shifts. For example, if a positive Doppler Shift (receiver moving towards the transmitter) occurred such that the received frequency was 2000 Hz *higher* than the normal L1 frequency, the final IF frequency would be 2000 Hz *lower* than it's nominal ~1.4 MHz. In terms of the oscillator error, a positive local oscillator error (faster than the nominal) would normally shift the spectrum to a lower frequency than normal, resulting in a final IF *below* the nominal. The aliasing process reverses, and slightly increases this shift *above* the nominal. How does aliasing increase the error? The error in the final IF frequency is *proportional* to the total shift in frequency of the down-conversion process. Therefore, the larger the total frequency shift, the larger the net frequency error. Aliasing effectively causes a total frequency shift equal to the received frequency *plus* the final IF, because it converts the signal to a *negative IF*, of which we see the positive mirror frequency. Therefore, a signal sampled

at a frequency greater than the Nyquist frequency would have a slightly smaller frequency error due to oscillator frequency error, because the total frequency shift in the down-conversion process is less. The difference in error between an aliasing implementation and sampling meeting the Nyquist criteria is given by Equation (3.15): (again, it is emphasized this analysis is not valid for *aliasing in general*):

$$(\text{IF frequency before sampling - final (aliased) IF frequency}) * \frac{\Delta f}{f} \ Hz \tag{3.15}$$

As an illustration, suppose the GP2010's oscillator has a 2.5 ppm frequency error (maximum error rating of the oscillators recommended by Mitel). If the sampling frequency were greater than the pre-sampled IF's Nyquist frequency (~4.3 MHz), the sampled frequency would have an error ~14 Hz less *in magnitude* than incurred by the same oscillator frequency error in the aliasing case. If aliasing is not present in the sampling, a frequency error in the sampling frequency does not necessarily result in any additional shift of the sampled frequency, as long as the error does not cause the sampling frequency to go below the Nyquist sampling rate.

Table 3.2 lists the four types of oscillator-induced errors that occur in the Mitel GPS GP2010 RF front end modeled by the SS [21].

Table 3.2. Oscillator Induced Errors in the GP2010 GPS RF Front-End [21]

Four Main Effects on System Performance Caused by Clock Frequency Variations:		
Oscillator Error (Offset--ppm)	**Error at IF (Hz)**	**Effect on Receiver Performance**
Start-up Frequency Error (>0.3 ppm)	473 Hz	Start searching wrong Doppler bins for signal, results in longer Time to First Fix (TTFF)
Large Frequency Error (>50 ppm)	>~79 kHz	Significant frequency shift in the down-converted GPS signal; results in filtering out signal power.
Abrupt Frequency Changes (<<0.1 ppm)	<< 158 Hz	Cause the system to prevent or lose satellite-code lock.
Modulation of Reference Frequency (10s of Hz up to 10s of MHz)	n/a	Adds out-of-band noise to the signal spectrum and distorts the received signal, resulting in lower Signal-to-Noise ratios.

Mitel recommends a temperature controlled crystal oscillators (TCXO) with a maximum frequency error (over the entire temperature range) of 2.5 ppm [21]. This error bound only specifies the temperature-induced drift error; it does not include the vibration or shock induced frequency errors (see Figure 2-10). Allan variance parameters were not available for the oscillators listed, so typical values were taken from [37]. These values are used as the parameters for the clock error model.

The frequency error term in Equation (3.11) was modeled as a bounded random walk, using an Allan variance plot to determine the amount of walk, and a "bumper rail" bounding approach. The bound was implemented such that whenever the random walk crossed a bound, it was "reflected" back inside the bound, about the last point inside the bound. Thus the discrete magnitude of the simulated random walk step is not altered, only the direction, and only when the clock error reaches the user specified bounds. This bounding is done because the frequency error of an oscillator does not continually grow as a random walk does. The process could also be simulated using a Gauss-Markov first-

order lag noise model [18], and it might be more accurate to do so. This approach was used because it eliminates the specification of a time constant and no existing models for *generating* this frequency drift could be found.

Due to the limited information on modeling crystal oscillator clock performance with only Allan variance and temperature stability specifications, no claims are made to the realism of the clock model. However, given a time vector of frequency drifts, the effects of those drifts are fully simulated. So if an accurate (or real) drift vector is generated or provided (at the sampling rate) the effects are modeled accurately. Without a proven, realistic time vector of frequency clock drifts, the model is claimed to be reasonable *only* from a *performance comparison* standpoint. To compensate for this, the parameters controlling the oscillator frequency error model are easily modifiable via global variables, and a fixed oscillator frequency error implementation is also available. The user may specify the Allan variance parameters, the time interval (τ) of the Allan variance, and the bound on the random walk model. The time interval specifies which integration interval of an Allan variance is used for selecting the noise strength of the random walk or the variance of the random bias. Attempts were made to model a clock drift exhibiting typical Allan variance properties, but such a process is not easily simulated in Matlab®. The equation generating the Allan variance parameters was taken from [33] and is shown in Equation (3.16), with the h_1 and h_2 terms removed. These terms specify the long term accuracy of a oscillator, which is not of interest here. Typical temperature-compensated crystal oscillator (TCXO) Allan parameters [37] are individually plotted in Figure 3-2, and the composite Allan variance is shown in Figure 3-3.

$$\sigma_{AV}^2(\tau) = h_{-2}\frac{(2\pi)^2}{6}|\tau| + h_{-1}2\ln 2 + \frac{h_0}{2}\frac{1}{|\tau|} \tag{3.16}$$

Where: [33]

$\sigma^2_{AV}(\tau)$	=	Allan variance
h_{-2}	=	Allan parameter specifying the random walk in frequency
h_{-1}	=	Allan parameter specifying the flicker frequency noise
h_0	=	Allan parameter specifying the white frequency noise
τ	=	Sampling interval of the Allan variance

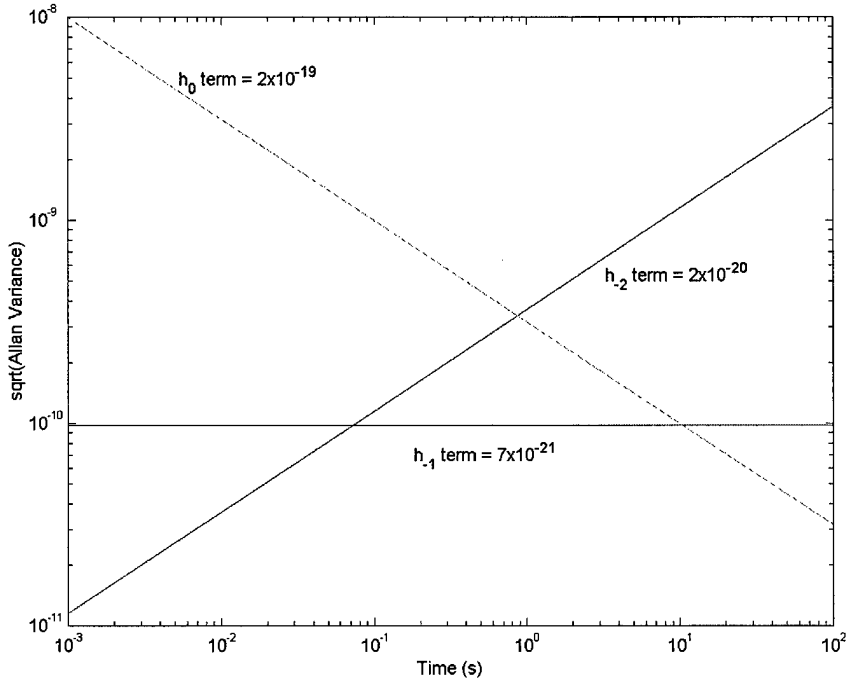


Figure 3-2. Individual Allan variance parameters

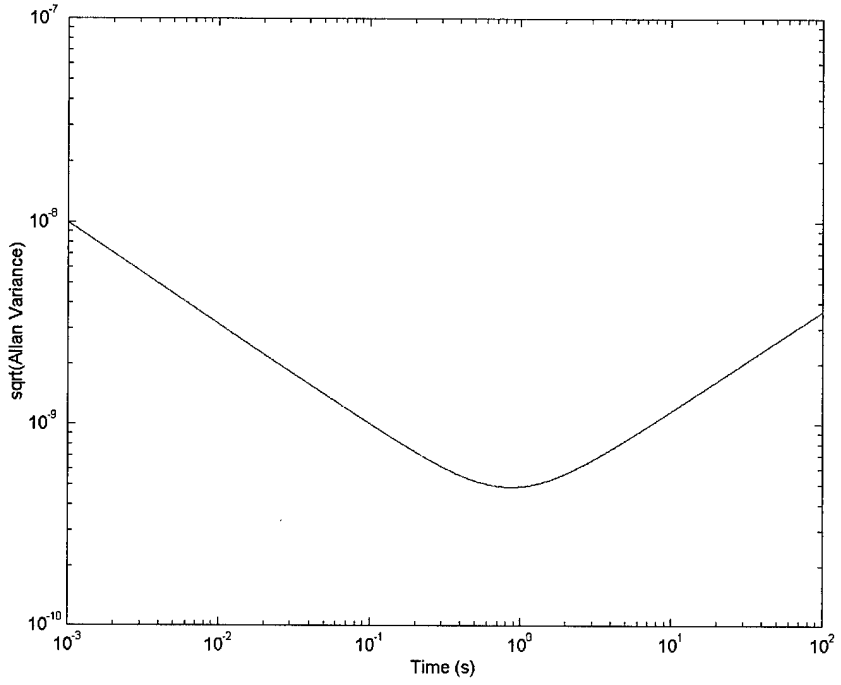


Figure 3-3. Typical crystal oscillator Allan variance plot

When implementing the random walk, the square-root of the desired variance (pulled from the Allan Variance plot) is multiplied by a Gaussian noise vector and divided by the sampling frequency before integrating. This integrated noise is the frequency drift, and the clock bias is the integral of the drift. In the random constant mode, the process is repeated except the drift is a constant rather than a random walk. The random constant's variance is specified by the Allan variance at a user specified time interval τ.

Given this model, it is high time to incorporate the oscillator effects into our equation modeling the down-converted signal. Applying the frequency relationships in Table 3.1, our previously derived signal expression (Equation (3.10)) can be expressed as:

$$IF_3 = \cos\{\omega_{IF_3} t_r - \omega_{L1}(T_d + \delta t_r)\} \tag{3.17}$$

Incorporating all the effects of an oscillator error is not without some implementation issues. Theoretically, the sampling frequency should be affected by the oscillator error just as the other LOs. Since aliasing effectively causes a down-conversion in the sampling process, these errors must be incorporated into the sampling times. In practice this would require the creation of a vector of sample times that vary according to the frequency drift. Using a non-uniform sampling time would significantly slow down the receiver model implementation in Simulink®. The same effect is more efficiently implemented by generating a vector of sampling times with a fixed delta time (which is easily done in Matlab®) and inserting the sampling frequency error component into the equation. This is accomplished by increasing the total frequency shift term (which is multiplied by the oscillator error) to include the shift caused by aliasing in the sampling process. In essence, Equation (3.18) illustrates the proper theoretical substitution for IF_3,

$$\omega_{IF_3} = \omega_{L1} - \omega_{LO_1} - \omega_{LO_2} - \omega_{LO_3} = 35.42MHz - \frac{\Delta f}{f}(140 + \frac{140}{10} + \frac{140}{45}) * 10MHz \tag{3.18}$$

and Equation (3.19) shows the substitution adjusted to include the effects of the sampling frequency error when a perfect nominal sampling frequency is used.

$$\omega_{IF_3} = \omega_{L1} - \omega_{LO_1} - \omega_{LO_2} - \omega_{LO_3} = 35.42MHz - \frac{\Delta f}{f}(140 + \frac{140}{10} + \frac{140}{45} + \frac{140}{245}) * 10MHz \tag{3.19}$$

The substitution given by Equation (3.19) is used in the SS. The final equation modeling a received GPS signal can now be formulated. Combining Equations (3.17) and (3.19), reinserting the data modulation, code modulation, noise, and amplitude terms,

and expressing all terms in receiver time yields the complete signal model, given by Equation (3.20):

$$R_{L1DC} = \sqrt{2P_r}G_i(t_r - T_d - \delta t_r)D_i(t_r - T_d - \delta t_r)\cos\{(\omega_{IF3} - \frac{\Delta f}{f} f_{TDC})t_r - \omega_{L1}(T_d + \delta t_r)\} + n(t_r) \quad (3.20)$$

In implementation, the receiver time in Equation (3.20) is a vector of sampling times generated at the fixed sampling frequency. Clock bias and drift vectors are simultaneously generated at the same sampling frequency and incorporate clock errors in the sampled signal.

3.2.3 Matlab® Realization

The Signal Simulator was configured to model the RF front-end chip of Mitel's GPS chipset, the GP2010. The GP2010 is fed by a GPS pre-amplified antenna and generates a 4-level (2-bit) down-converted representation of the received C/A code signal spectrum filtered to a bandwidth of 2 MHz. A block diagram of this chip is shown in Figure 3-4. As shown in Figure 3-4 and specified previously in Table 3.1, the final analog IF is ~4.3 MHz, and the sampling rate is ~5.7 MHz.

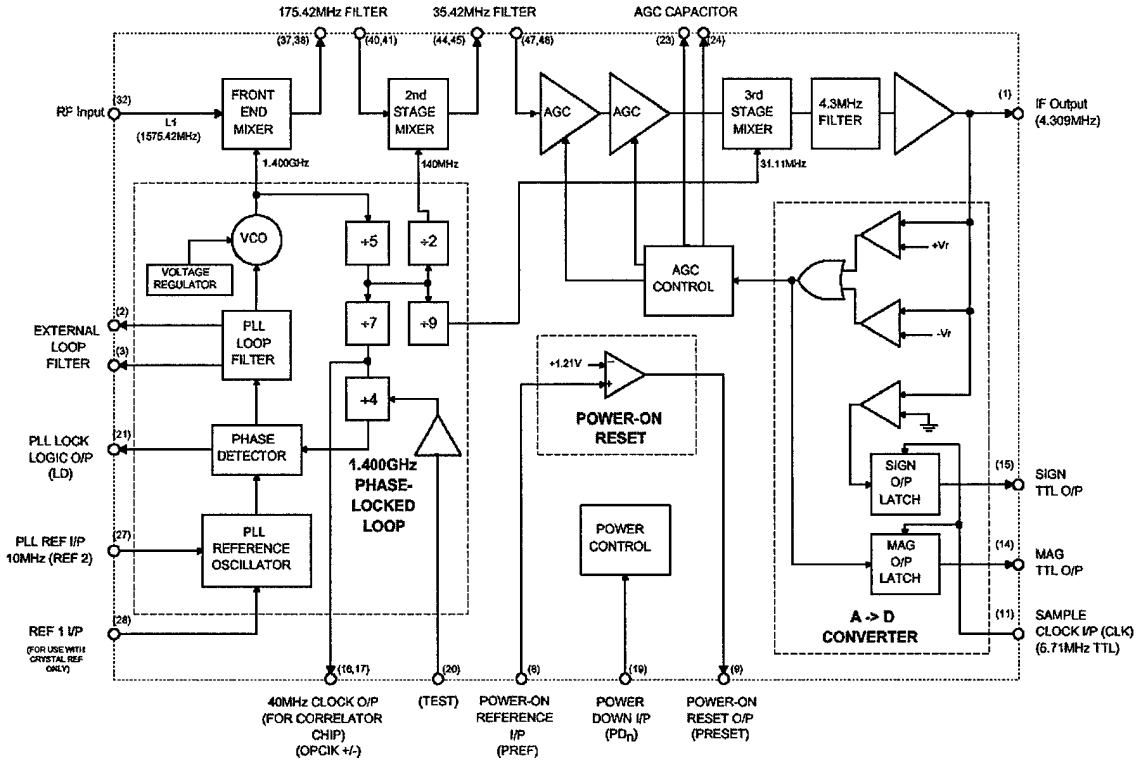


Figure 3-4. GP2010 Block Diagram

Currently, 2-bit quantization is all that is supported by the SS. However, quantization can be turned off, to generate a non-quantized but sampled signal spectrum for external quantization and AGC algorithms. The thresholds of the quantization process are fixed in both the Matlab® and Simulink® AGC implementations, but AGC parameters controlling the response of the Simulink® AGC are coded as global variables. The parameters of a chipset that are definable for both the SS and the RM are encoded in global variables, which are listed in Table 3.3. The SS and RM columns indicate if that particular variable is utilized by the respective model. These parameters can be easily changed to model other chipsets. By convention, global variables in Matlab® are all capital letters.

Table 3.3. Receiver hardware global variables

Receiver Hardware Dependent Variables:			
Variable Name	**SS**	**RM**	**Description / Value**
SAMPLING_FREQUENCY	Y	Y	40/7 MHz
SAMPLING_DT	Y	Y	Time between samples / 0.175 μs
IF_BEFORE_SAMP	Y	Y	IF prior to sampling / $4.30\overline{8}$ MHz
IF_AFTER_SAMP	Y	Y	Aliased IF frequency / 8854e4/63 MHz
CLOCK_PARAMS	Y	N	Structure containing Allan parameters, maximum frequency error, and τ
LATCHING_DT		Y	Synchronized latching interval of channels (s)
CHIP_SPACING		Y	Spacing between locally generated early, prompt, and late PRN sequences (s)

The following list describes the capabilities of the SS, listing all the error models, signal components, and truth environment parameters that were modeled in this research. Of these effects and errors, the SS was designed such that the following effects, with the exception of the smooth dynamic modeling, can be turned on and off easily as needs arise.

- Smooth (spline fit) modeling of receiver and satellite dynamics
- True C/A code spreading code modulation
- Data modulation (random data bits)
- Number of signals included in a simulated spectrum limited only by the maximum number of GPS signals available
- IF filtering of noise and signals in Matlab®
- Dynamic or fixed AGC implementation for 2-bit quantization
- Receiver clock frequency drift model, both variable and constant
- User defined frequency plan
- S/N ratio individually selectable for each channel

A block diagram of the signal simulator code, structured with respect to the hierarchy of functions, is shown in Figure 3-5. The order in which the functions are called is predominantly from the top down, then left to right. Some functions exist inside other functions, and do not comprise a file by themselves. Those functions are shown in

italics. However, throughout the chapter when these functions are listed in the text, they are italicized regardless of whether they are file names or just function names. The shading is to represent different functional sections of the code. The main signal generation routine is comprised of 6 functions, but interfaces with the main function (*gen_samples*) entirely through *sampled_recv_signal*. The shading illustrates they comprise a single 'branch' from gen_samples.

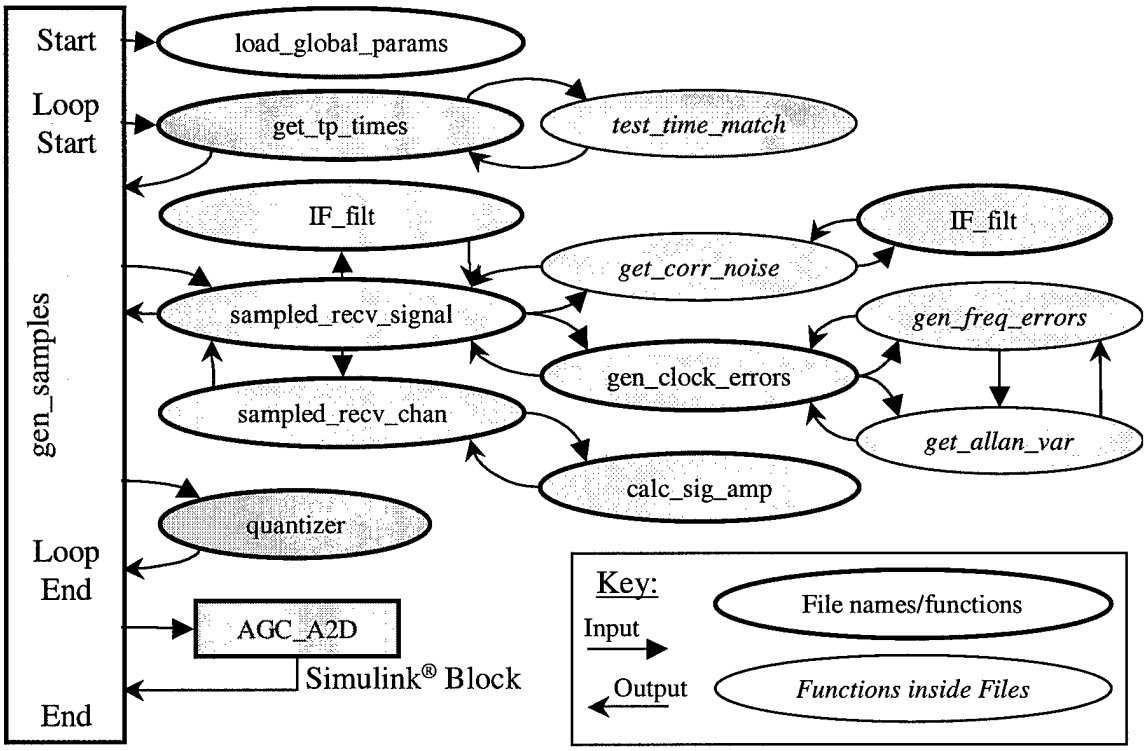


Figure 3-5. Signal Simulator function list

Some input parameters are directly specified in the function call to *gen_samples*, but both the signal simulator and the receiver model in Simulink® are mostly controlled by global variables, defined in the Matlab® file *load_global_params*. Most global variables are utilized by both the RM and the SS, but some are specific to either model. To avoid redundancy, the global variables of both the SS and the RM are listed in the

tables that follow. The global variables include GPS parameters (constants), the hardware-dependent variables for the chipset being modeled (see Table 3.3), as well as parameters that control the simulation, specify the truth environment, and switch various models on and off. Table 3.4 contains the GPS constants used in the SS. The CA_CODE variable contains the entire sequence of 1023 chips for each PRN stored in a Matlab® cell array.

Table 3.4. Global variables defining GPS constants

GPS System Constants:			
Variable Name	**SS**	**RM**	**Description**
CARRIER_FREQUENCY	Y	Y	GPS L1 Frequency, 1575.42 MHz
CARRIER_WAVELENGTH	Y	Y	GPS L1 Wavelength, 0.19029 meters
CHIPPING_RATE	Y	N	C/A code frequency, 1.023 MHz
T_C	Y	Y	C/A code chip duration, 0.9775 μs
SPEED_OF_LIGHT	Y	Y	299792458 m/s
CA_CODE	Y	Y	Cell array of all 37 PN Gold Code Sequences

Some global variables initialized in *load_global_params* are output variables, generated as the simulation runs. This method of extracting information from a run is more flexible than specifying output arguments in the function call to *gen_samples.* This allows a fixed number of output arguments per function call regardless of what variables are being exported during a simulation. The only output argument that is returned by *gen_samples* is the actual signal. The simulation control parameters are given in Table 3.5.

Table 3.5. Global variable control parameters

Simulation control parameters:			
Variable Name	**SS**	**RM**	**Description**
START_TIME	Y	N	Start time of the simulator, in GPS time (s)
RUN_TIME	Y	Y	Amount of time that simulator runs (s)
STOP_TIME	Y	N	Stop time of the simulator, in GPS time (s)
PIT_CNT	Y	N	Global loop counter (used during execution)
PARSE_SIG_INTERVAL	Y	N	Seconds per loop. Specifies how often the output truth parameters are generated and the time constant of the Matlab® AGC
NOISE_POWER	Y	N	Power of the noise (arbitrary, used as a reference point to generate relative signal strengths)
SNR_DB	Y	N	Signal-to-noise ratio in dB of each signal simulated
A2D_FACT	Y	N	Specifies the sampling rate of the noise before it is filtered and sampled to the final aliased frequency
PRN_VEC	Y	Y	Vector of PRNs to include in the simulator

The START_TIME variable specifies the location within a receiver trajectory at which to start the simulation. The trajectory file must be time-tagged in GPS time, which is a count of the seconds in a week, starting at ~midnight on Saturday. Obviously, the START_TIME must lie inside the times given in the trajectory file. The run time specifies the length of the simulation in seconds. With a ~5.7 MHz sampling rate, a one second simulation generates a time-tagged sampled signal requiring approximately ~90 MB of storage space. PARSE_SIG_INTERVAL essentially specifies the number of samples to process in one "loop" of the gen_samples routine. During a simulation, the entire sampled signal is generated in sections, the size of which are specified by PARSE_SIG_INTERVAL. For each section of the sampled signal, a set of truth parameters is recorded to global variables, and if using the Matlab® quantization routine, the section is ideally quantized from the distribution of the points in the section.

Therefore, the PARSE_SIG_INTERVAL variable controls both how often the truth parameters of the sampled signal are generated, and similarly the number of samples used to calculate the quantization thresholds in Matlab.

Some control parameters are defined as either a one or a zero, which are used to 'switch' functions in the SS on and off. These switches are given in Table 3.6.

Table 3.6. Global variable control switches

Modeling and debugging switches			
Variable Name	**SS**	**RM**	**Description**
APPLY_CA	Y	Y	Apply spreading modulation
QUANTIZE_SIG	Y	Y	Apply quantization to the signal
ADD_NOISE	Y	N	Add noise to the signal
ADD_OSC_ERRORS	Y	N	Apply oscillator error effects
APPLY_DATA	Y	N	Apply data modulation (random)
FILTER_SIG	Y	N	Apply filtering to the signal
FILTER_NOISE	Y	N	Apply filtering to the noise
DYNAMIC_AGC	Y	N	Use the dynamic AGC in Simulink®
SAVE_QUANT_THRESH	Y	N	Save the quantization thresholds in Matlab®
SAVE_CLK_ERRORS	Y	N	Save the clock and drift error vectors
REPEATABILITY	Y	N	Use the same noise seed for each run
SAVE_FEE_REF	Y	N	Save LOS radians from receiver to satellite
MIX_TO_BASEBAND	N	Y	Apply cosine mixing in Simulink
DUMP_ON_EPOCH	N	Y	Latch the accumulated I and Q samples on the code roll-over of each channel
SAVE_TIME	N	Y	Save the simulation time of the Receiver Model

Most of the variables are self-explanatory. The parameters starting with 'SAVE' trigger the SS (or RM) to do data logging on the respective quantities at the full sampling rate. Turning many of these on drastically increases the memory requirements to run the SS and limits the run time of the simulator. However, saving off some of these quantities is extremely helpful when debugging or taking an in-depth look at a run. The SS can generate a sampled signal spectrum without quantization, spreading modulation, and/or

data modulation. Likewise, the RM can turn off its cosine mixing and spreading demodulation functions. REPEATABILITY is a useful switch for debugging as it uses the same noise seed for each run. Turning on the data modulation automatically causes the SS to write the data bits to a global variable. However, the clock errors are not automatically saved if the ADD_OSC_ERRORS switch is set.

Without processing the generated signal with the receiver model, virtually no information is readily apparent looking at the simulated signal, especially when multiple signals are simulated at the appropriate S/N ratios (see Table 2.3). The truth environment (receiver positions and satellite positions) used to simulate the signals do not readily provide information on the signals contained in the output vector either. The SS provides this information via global variables created during the simulation. Table 3.7 lists all the potential information that can be recorded by *gen_samples* in global variables. The switched column specifies if a global switch controls the recording of the information.

Table 3.7. Simulation Parameters stored in global variables

Saved truth environment variables:			
Variable Name	**Switched**	**Rate**	**Description**
DOPPLER_FREQ	N	PI	Perceived Doppler frequency of each signal at the beginning of the Parsed Interval (PI)
RAD_TO_SV	N	PI	Perceived radians to satellite of each signal at the first sample of the PI
DOPPLER_FREQ_TRUE	Y	PI	True Doppler frequency of each signal at the beginning of the PI
RAD_TO_SV_TRUE	Y	PI	True radians to satellite of each signal at the first sample of the PI
DATA_BITS	Y	PI	Cell array of the data bit index, data bit value, and data bit transition time for each PI
CLOCK_ERRORS	Y	SF	Clock bias and drift recorded at the sampling frequency (SF)
ML_QUANT_THRESH	Y	PI	Matlab® quantization thresholds used on each section of the parsed signal
REAL_SNR	N	PI	Calculated SNR of each signal against the composite spectrum over the samples in one PI
IDEAL_SNR	N	PI	Calculated SNR for each signal against the calculated noise power excluding any signals over the samples in one PI

The difference between the REAL_SNR and IDEAL_SNR quantities is that REAL_SNR includes the cross-correlation noise in the measurement. The calculated SNR in the REAL_SNR variable is lower than the respective IDEAL_SNR calculated power as (1) the number of signals in the spectrum is increased, (2) the SNR of the signals in the spectrum is increased, and (3) the power difference between the signals in the spectrum increase. The Doppler measurements are generated from an average over the first 500 samples of each signal. The RAD_TO_SV variable contains the LOS radians, or $2\pi\omega_{L1}xT_d$ radians, for each signal at the first sample of the section being processed. This is simply the distance from the receiver to the satellite expressed in

radians of the carrier frequency. The RAD_TO_SV_TRUE and DOPPLER_FREQ_TRUE terms record the true values in the case that oscillator errors have been added in the simulation. If data modulation is activated, the SS generates an entire (random) GPS data message. The DATA_BITS variable contains only the data bits used for each of the signals during the run. Although it is overkill to generate a complete data set for the time scales of these runs, the simulator was designed around an entire data message in order that an actual data message could be used in the future, allowing for valid comparisons of simulated signals with real (data-modulated) sampled signals.

The last remaining parameters are specified in the function call of *gen_samples*. These include (1) the receiver trajectory, (2) the satellite position matrix, and (3) the file name to which the simulation results are written. The output file generated is a Matlab® 'MAT' file which contains the sampled signal spectrum, all the output global variables listed in Table 3.7, and the following control parameters: PRN_VEC, SNR_DB, RUN_TIME,and START_TIME.

The source code (Matlab® pseudocode) for the SS is provided in Appendix A., and available in electronic format via the CD in the back cover of this document. While a complete explanation of all the equations is beyond the scope of this document, certain sections of the code are covered that are considered key concepts to understanding the SS. These portions of the simulator are covered here in the order they are called in the main function *gen_samples*.

3.2.3.1 *Get_tp_times.m*

This function is the core of the dynamic modeling in the SS. This function uses the trajectory and satellite position matrices to derive a matrix of LOS distances for each GPS signal at each time in the input trajectory. This matrix of LOS distances is calculated once on the first function call, and used in all the subsequent calls. A cubic spline fit is performed on this matrix for a vector of sample times, specified by the SAMPLING_FREQUENCY global variable, generating a matrix of true LOS distances for each signal at each *sample time*, which are converted to propagation times using the speed of light specified in Table 3.4. This process, although computationally intensive, allows for smooth, 3rd order polynomial fit of the LOS dynamics. The Matlab® spline routine is used, which is a built-in function and not part of the spline toolbox®. The jerk, or 3rd order dynamic of the LOS path is piece-wise continuous. Because sampling frequencies are typically >4 MHz, this results in a smoothed but reasonable representation of the system dynamics.

3.2.3.2 *Get_corr_noise*

The filtering process in the downconversion band-limits the noise in the samples. The correlated noise is generated is the following manner. A Matlab® Gaussian noise sequence is generated at a higher multiple of the sampling rate, specified by A2D_FACT. This noise vector is filtered in Matlab® using an elliptical filter model. The model can either be selected to be a very high-order filter which approximates ideal filtering, or a more realistic filter modeled after the response of the dominate IF filter in the down-conversion process [21]. Either of these filters is easily selectable in the *if_filt* function.

The Matlab® generated frequency response of these filters is shown in Figure 3-6 and Figure 3-7.

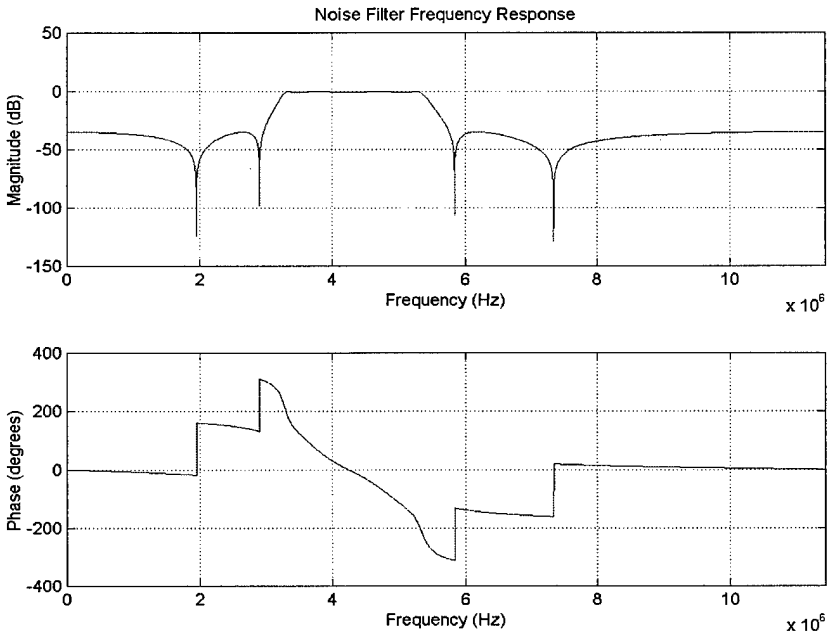


Figure 3-6. Elliptical filter approximating the Mitel chipset IF bandpass filtering

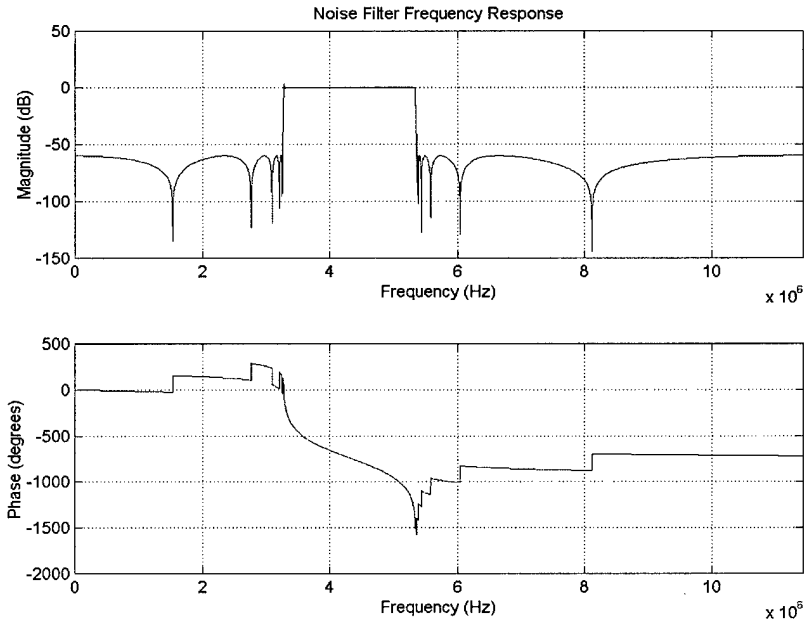


Figure 3-7. Elliptical filter approximating ideal IF bandpass filtering

Once the noise sequence has been filtered, it is 'sampled' through decimating the higher frequency noise sequence. The higher the A2D_FACT parameter, the more 'accurate' this filtering process becomes. The Power Spectral Density (PSD) of the filtered noise is shown before filtering in Figure 3-8, after filtering in Figure 3-9, and after 'sampling' in Figure 3-10. The ideal filtering parameters and an A2D_FACT of 4 were used to generate these noise PSD plots. The NOISE_POWER parameter specifies the power of the noise, as measured in Matlab®. Due to the loss of power in filtering, the initial noise power before filtering is increased such that the end power of the 'sampled' noise at the IF_AFTER_SAMP frequency meets the NOISE_POWER specification.

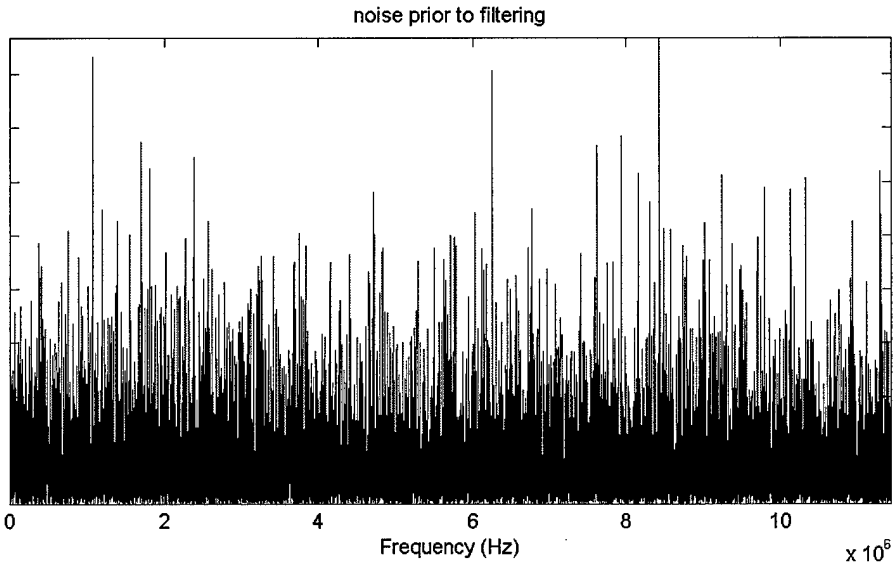


Figure 3-8. PSD of noise before filtering and sampling

Because only the signal to noise ratio is relevant in modeling the received signal with an AGC, the power of the noise can be scaled without loss of simulation accuracy. For this reason, the amplitudes of Figure 3-8, Figure 3-9, and Figure 3-10 are omitted. The bandwidth of the bandpass filter used is ~2 MHz. Figure 3-10 shows that the bandwidth

of the noise remains 2 MHz after sampling it to the final IF. In a discrete environment such as Matlab, the frequencies of a PSD plot greater than half of the sampling frequency are not defined. Thus, Figure 3-10 has an upper frequency limit a factor of 4 less than Figure 3-8, and Figure 3-9.

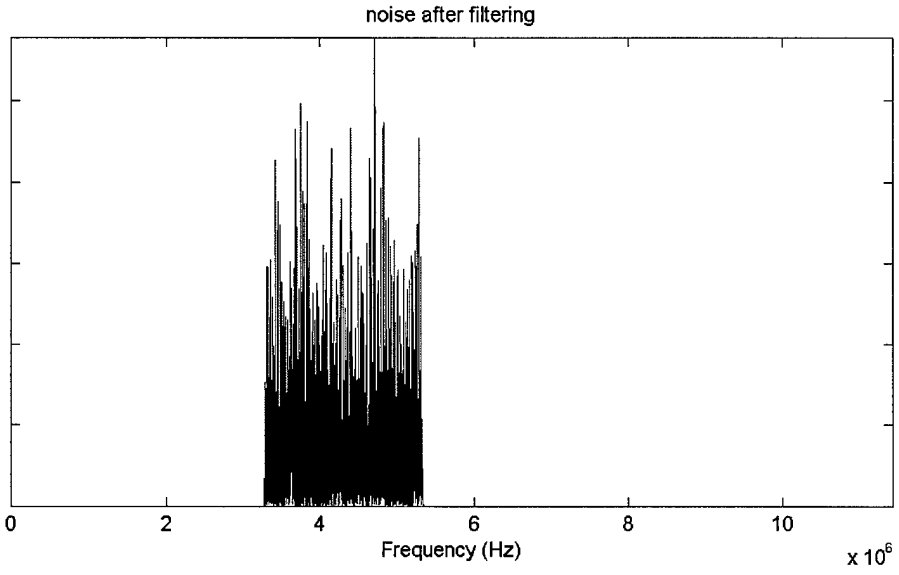


Figure 3-9. PSD of noise after filtering

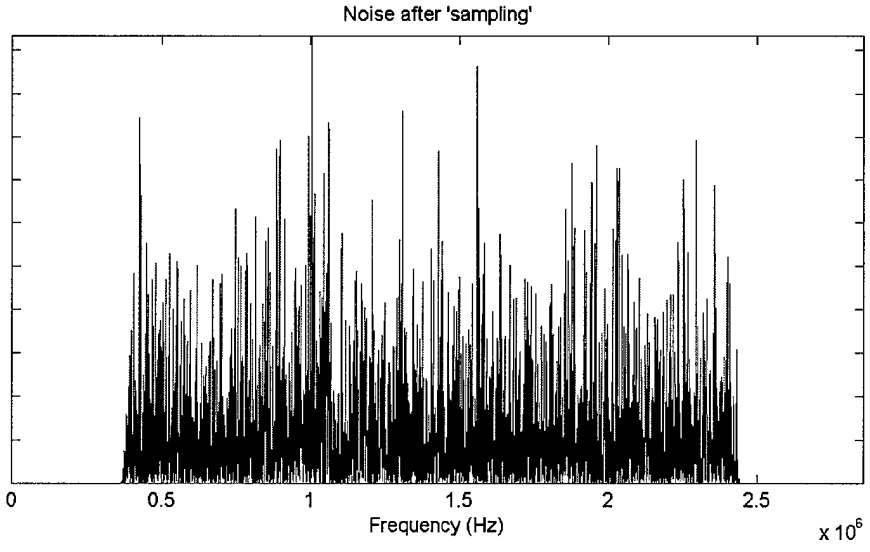


Figure 3-10. PSD of noise after filtering and sampling

3.2.3.3 *Sampled_recv_chan.m*

This function generates the sampled signal using the signal model derived previously and specified in Equation (3.20). The *calc_sig_amp* function calculates the amplitude of the signal according to the signal's specified SNR_DB value. This routine generates one GPS signal per function call, and the *sampled_recv_signal* combines the outputs of this function into a composite simulated sampled spectrum. Although the ability to filter the signal is available at the 'flip' of a global variable, the practice of aliasing the signal in the sampling process prevents the feature from being realistic. The frequency plan of the Mitel chipset results in the sampled signal incurring a chip transition approximately every 5.5 samples. For the filtering of this signal to approach realism, it would have to be simulated at a much higher frequency, filtered, and sampled similar to the correlated noise generation process. The computational burden of this process is significant, and the effect of the filtering on the signal (as long as the oscillator error is <<50 ppm, see Table 3.2) can be reasonably approximated as a decrease in signal power.

3.2.3.4 *Quantizer.m*

The Matlab® quantization routine iteratively determines the threshold values that meet the sample distribution goals of the Mitel design. The number of samples over which these thresholds are calculated is determined by the PARSE_SIG_INTERVAL variable (see discussion of Table 3.5 on page 3-27). This quantization approach yields acceptable results when the power of the received spectrum is relatively constant, however, if the noise power of the received spectrum fluctuates, this method does not adequately characterize the behavior of a real AGC. Thus, an AGC in Simulink® was

devised to approximate these environments. Although the existing SS does not have the built-in capability of generating fluxuations in the noise, quantization can be turned off, generating an unquantized but sampled signal (this is not possible in actual implementations). Now jamming signals and excess noise can be added to the SS generated signals to simulate changes in the received noise levels. The composite signal can then be run through the programmable Simulink® AGC to simulate the effects of a dynamic AGC.

3.2.3.5 Simulink® Automatic Gain Control

Simulink® is a graphical, block-diagram simulation environment. Systems are built by combining blocks that perform a given function on an input, generating the processed result on the output. Every block in a Simulink® diagram is executed at each time step, giving rise to the ability to implement feedback easily in a system. This feedback capability is the reason to implement an AGC model in Simulink® rather than in Matlab®.

Systems implemented in Simulink® are typically multiple layers deep, i.e. their functional blocks have a hierarchy of input-output relationships. As such, a complicated system can be represented at a higher level by a simple box with a few inputs and outputs. The Simulink® models developed in this thesis exhibit a hierarchy of processes logically grouped to reduce the perception of complexity and increase the modularity of the separable functions.

The top level of the AGC model is shown in Figure 1-1. The single input and output are clearly marked with *gps_sig*, the name of the variable generated by gen_samples that contains the sampled signal. The feedback path is not readily apparent

from this view of the system. The threshold for determining the magnitude is pulled from a memory location at each time step, which is updated by the AGC block in the diagram.

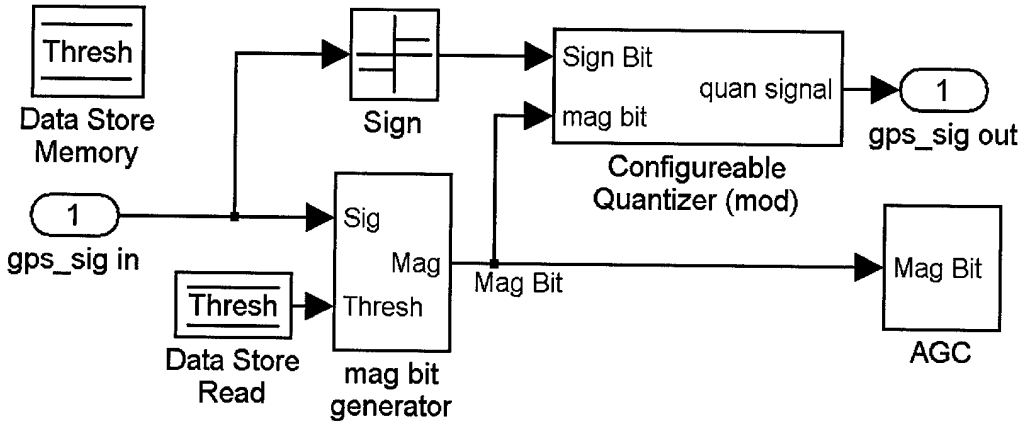


Figure 3-11. Simulink® Automatic Gain Control (top level)

The magnitude bit is generated by the "*mag bit generator*" block, which is displayed in Figure 3-12. Any sample having a magnitude greater than the current value of the *Thresh* memory location generates a logic "1" at the output.

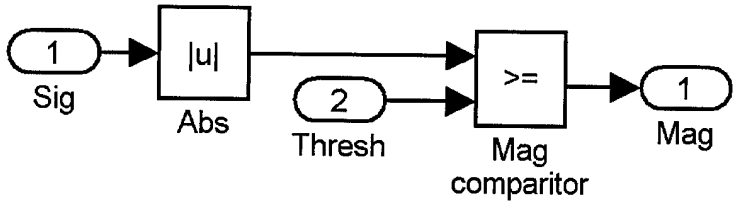


Figure 3-12. Magnitude bit generator block

The magnitude bit is fed into a switch in the *configurable quantizer* block, shown in Figure 3-13, which outputs the appropriate bit mapping. The sign bit effectively assigns the sign to the mapped magnitude value with a multiplication.

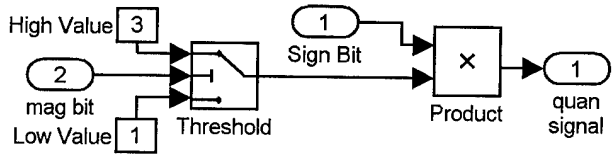


Figure 3-13. Configurable quantizer block

The AGC block (Figure 3-14) implements the control algorithm, which updates the *thresh* value. The design emphasis of the AGC was on flexibility and configurability; as such, many tools for analysis and debugging are included in the design. The basic operation is to accumulate the magnitude bit over a time interval to determine its duty cycle. The measured duty cycle is compared to a target duty cycle (30% for the Mitel chipset) and an error signal is generated which modifies the THRESH thus establishing the feedback of the system.

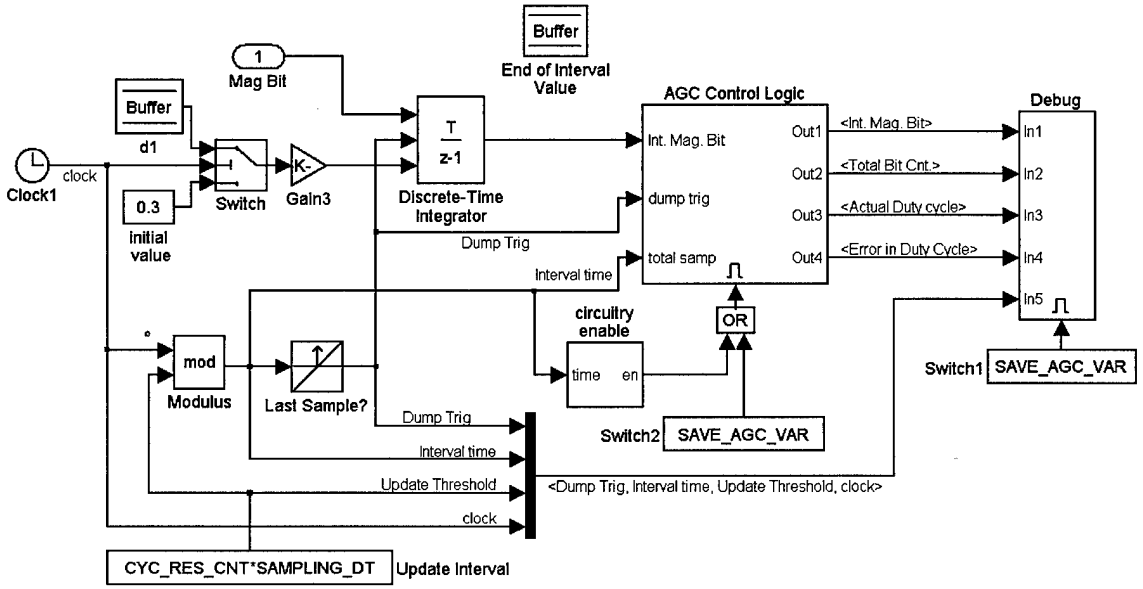


Figure 3-14. AGC block

The control parameters for the AGC are listed in Table 3.8. These parameters essentially specify the time constant and type of response (critically damped, over-damped, under-damped, etc.) of the system. These quantities were selected as probable design parameters for a truly digital AGC implementation, such as the AGC in the Mitel chipset. Little information of the exact method of operation of the Mitel AGC is available [21], but based on what is known, the parameters in Table 3.8 should be

equivalent to design parameters of the AGC in the Mitel chipset. Although the Mitel AGC controls the gain of the signal and not the threshold, controlling the threshold for a fixed gain is an equivalent process.

Table 3.8. Simulink® AGC control parameters

AGC control parameters:	
Variable Name	**Description**
CYC_RES_CNT	Number of samples between THRESH updates
GAIN	Scale factor applied to duty cycle error to generate correction to THRESH value
INIT_CYCLES	Number of cycles pre-loaded into the integrator at an update cycle, weighted by the last measured duty cycle. A measure of averaging the response over more than one update cycle.
AGC_DEBUG	Output variable for storing the internal signals of a run
CLOSED_LOOP	Switch specifying closed or open loop operation
SAVE_AGC_VAR	Switch that enables the storing of internal signals in AGC_DEBUG.

Figure 3-15 shows the control logic of the AGC:

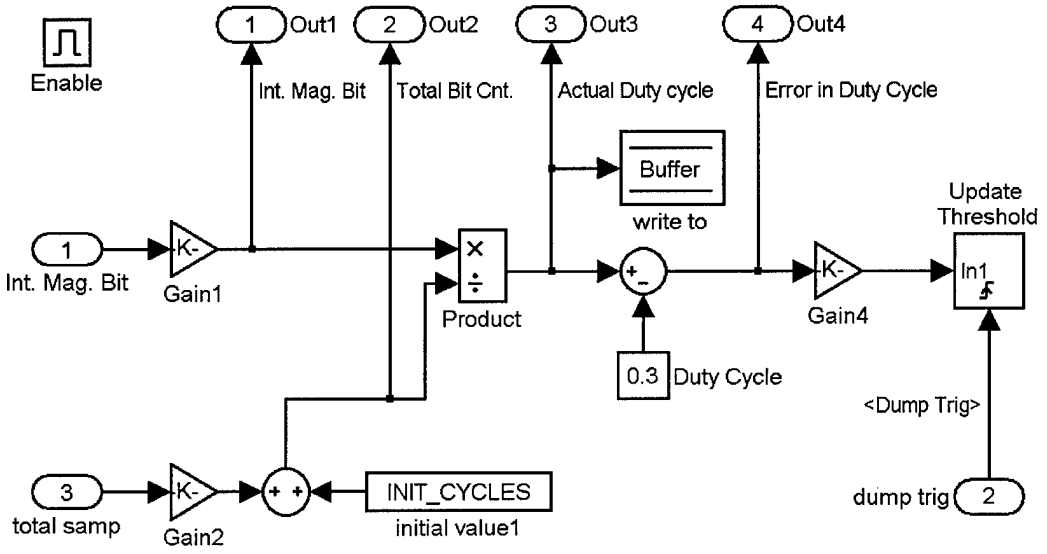


Figure 3-15. AGC control logic block

The accumulated samples are divided by the total number of samples, both of which were "padded" with a specified number of samples scaled from the final actual duty cycle from the previous run. This prevents large transients at the beginning of each integration interval in the actual measured duty cycle. These transients would normally occur from the division of a small number of samples at the output of the product block in Figure 3-15. The gain blocks *gain1* and *gain2* convert the integrated data bits and simulation time into integer samples. The *gain4* block contains the GAIN parameter in Table 3.8. If the CLOSED_LOOP parameter is set, the *Update Threshold* block (Figure 3-16) incorporates the error signal into the THRESH value. Otherwise, the AGC runs with a fixed threshold, which is useful for comparison purposes.

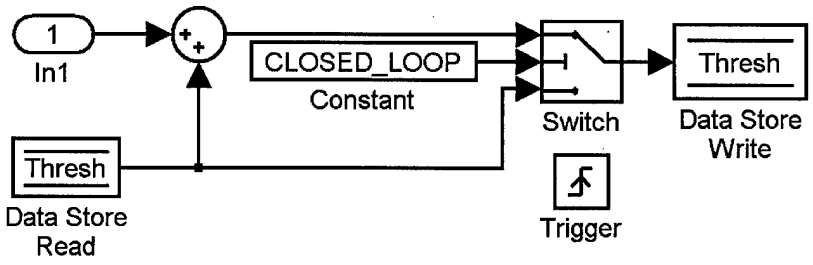


Figure 3-16. Update Threshold Block

The *Debug* block in Figure 3-14 allows for the tuning and "viewing" of the AGC operation as it runs. The contents of the *Debug* block are shown in Figure 3-17. If the SAVE_AGC_VAR switch is set, the block is activated and stores the signals entering into the multiplexer bar. The scope displays the signals as the simulation runs and allows for immediate analysis of the AGC operation following a run. The signal selector block allows the user to select from among the available signals so that only the signals of interest are displayed.

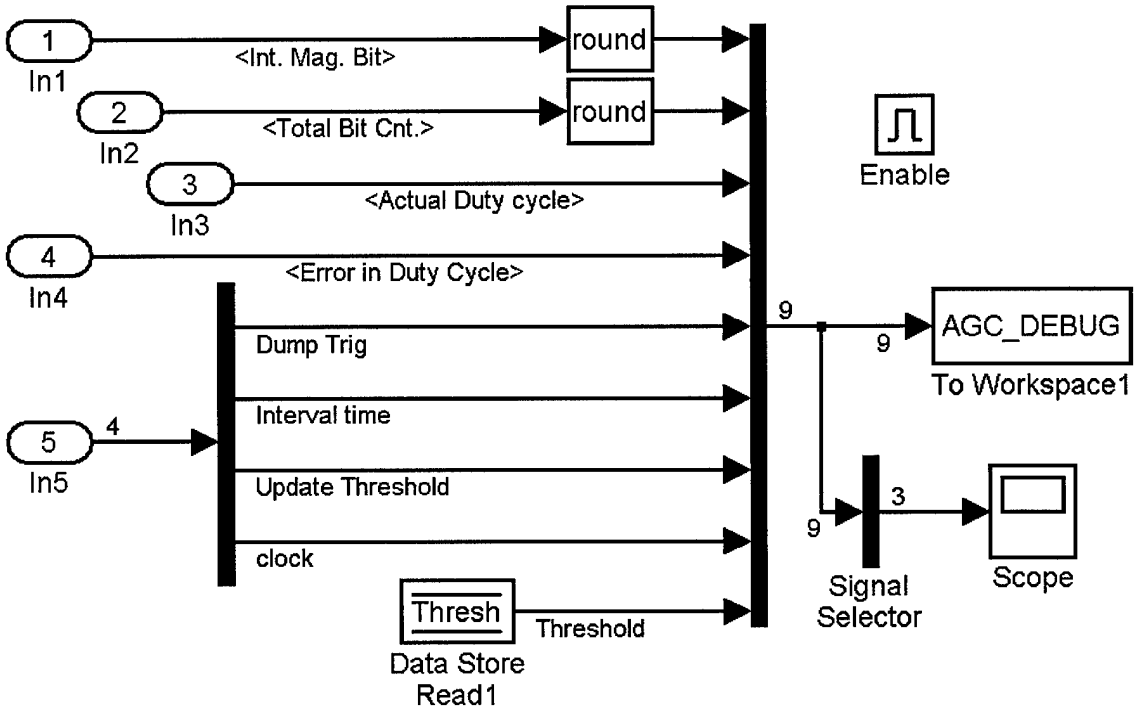


Figure 3-17. AGC data logging block (debug)

This concludes the discussion of the Signal Simulator (SS). The AGC implementation in Simulink® provides a good primer for understanding the design of the Receiver Model (RM), done entirely in Simulink®.

3.3 GPS Receiver Model

The GPS Receiver Model (RM) faithfully models a typical (configurable) 12-channel C/A code signal-processing chip. The functions of this model are those typically implemented in high-speed DSP chips due to the nature of the computations and the speed (4 MHz +, typical) at which they must be executed. The design emphasis was on model configurability and providing access to all the useful information inside the simulation. A summary of the design features of the RM are listed here:

- ✓ 12 parallel independent channels
- ✓ User-selected sampling frequency
- ✓ Auto-adapting algorithms for both Nyquist and non-Nyquist sampling
- ✓ Independent code and carrier DCOs
- ✓ User programmable phase and frequency registers for each DCO
- ✓ True C/A code waveform generation of PRNs 1-37
- ✓ Each channel performs the following functions:
    - Cosine mixing to baseband
    - Early, prompt, and late code correlation
    - Processed I and Q accumulation (summation)
    - Pseudorange measurement generation
    - Carrier-phase measurement generation
- ✓ The following quantities are configurable:
    - Correlator chip spacing
    - Channel PRN assignment
    - Latching rate of the accumulator
    - Amplitude of the DCO waveform
    - Trigger source—synchronized among channels or on code epoch
- ✓ Each channel outputs the following quantities:
    - Early, prompt, and late accumulated I and Q signal components
    - Pseudorange measurement
    - Carrier-phase measurement
- ✓ Each channel is capable of saving the following signals at any user specified decimation of the sampling frequency:
    - Carrier DCO phase (radians)
    - In-phase and quadrature-phase DCO waveforms (2 signals)
    - I and Q samples after cosine mixing (2 signals)
    - Early, prompt, and late code waveforms (3 signals)
    - I and Q samples after correlation (6 signals)
    - Pseudorange measurement
    - Carrier-phase measurement

The RM was used to model the 12-channel correlator chip of the Mitel GPS chipset, namely the GP2021. The GP2021 block diagram is shown in Figure 3-18. The RM does not model the I/O interface, synchronization signals, or control signals of the GP2021, but focuses on the 12-channel correlator model instead. The purpose of this model is not to provide an intermediate step towards real implementation, but rather to provide an accurate model of the signal processing functions of a working chip.

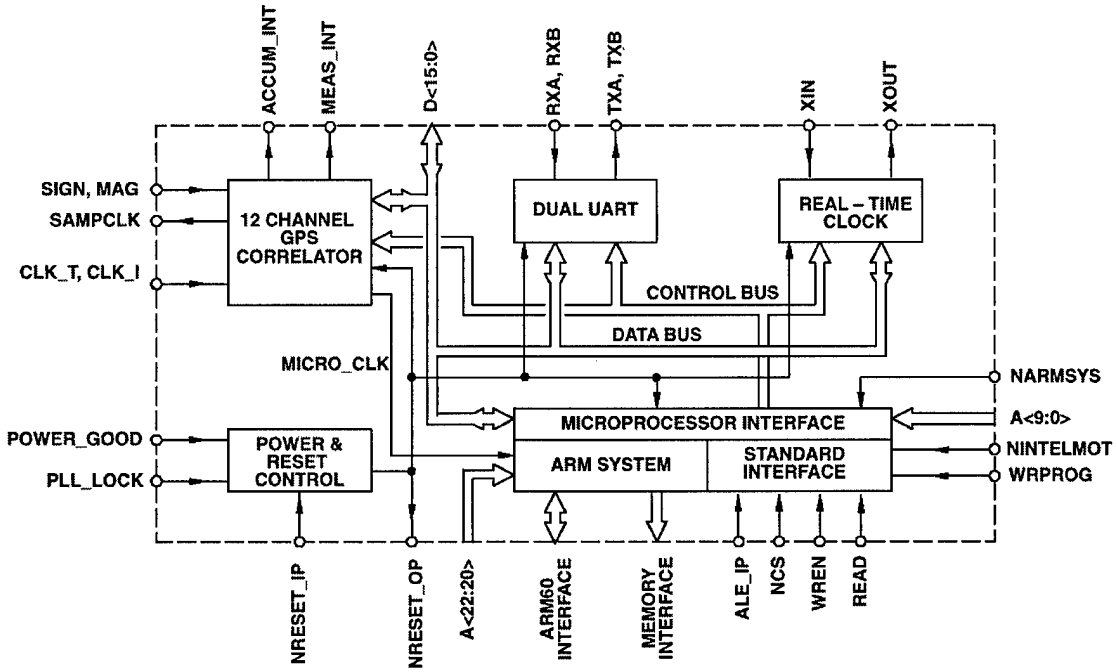


Figure 3-18. GP2021 block diagram [23]

The 12-channel correlator module in Figure 3-18 is faithfully modeled in the RM. A single channel of this module is shown in Figure 3-19. This diagram is explained in detail in [23], and it is discussed here in comparison to the RM.

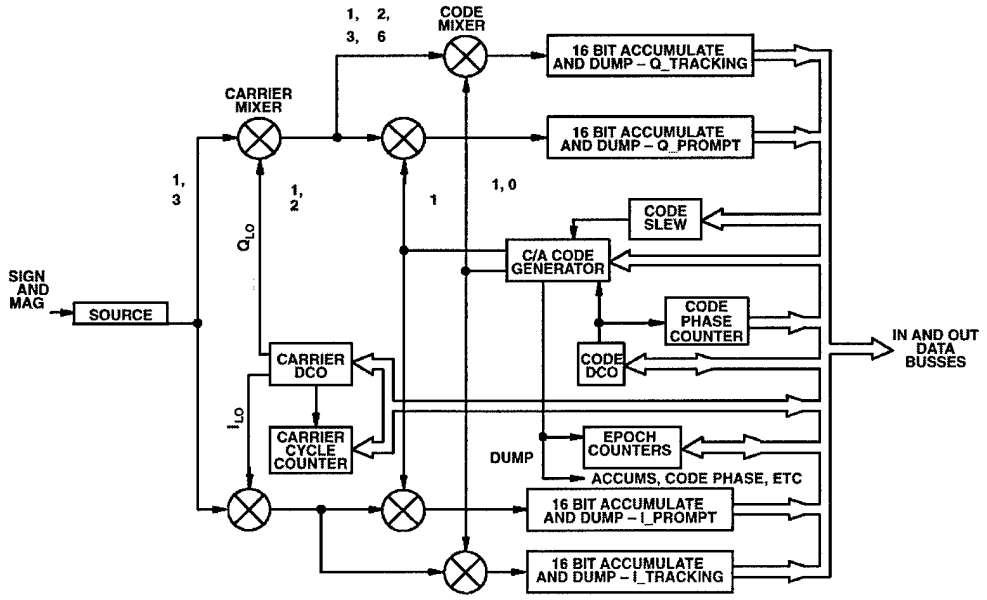


Figure 3-19. GP2021 tracking module block diagram [23]

The tracking module of the GP2021 has only two sets of correlators, described by Mitel as the "prompt arm" and "tracking arm" [23]. These 'tracking arms' can be configured to generate any two of the three signals available in the RM (early, prompt, and late) at a fixed ½ chip spacing between the sequences. If the correlators are set to generate early and late replicas, the code delay between the early and late remains ½ chip, which results in a net ¼ chip spacing of either sequence to the peak of the autocorrelation function (when perfectly aligned). Each of these code tracking configurations can be modeled by the RM, by modifying the CHIP_SPACING global variable and ignoring one of the correlator outputs.

The carrier DCO in the GP2021 generates a 4-valued cosine (or sine) wave composed of ± 1s and ± 2s. At the sampling frequency of ~5.7 MHz, there are approximately 4 DCO phases in the ~1.4 MHz IF frequency. These four values therefore have a nominally uniform distribution. The carrier DCO in the RM has been modeled to follow this design.

More comparisons between the Mitel GP2021 chip and the RM are made as the RM is presented. The next section will display and discuss the components of the RM as grouped in their respective Simulink® blocks. The RM is made up of approximately 33 unique blocks. Not all the blocks are presented here, but they are included in Appendix B. The blocks shown here offer both a theoretical and an operational view of the RM. For lack of an independent manual, Section 3.3.1 serves as a manual for the RM.

3.3.1 *Receiver Model Simulink® blocks*

The top-level view of the entire 12-channel Receiver Model is shown in Figure 3-20. This is what a user sees after first opening the model in Simulink®.

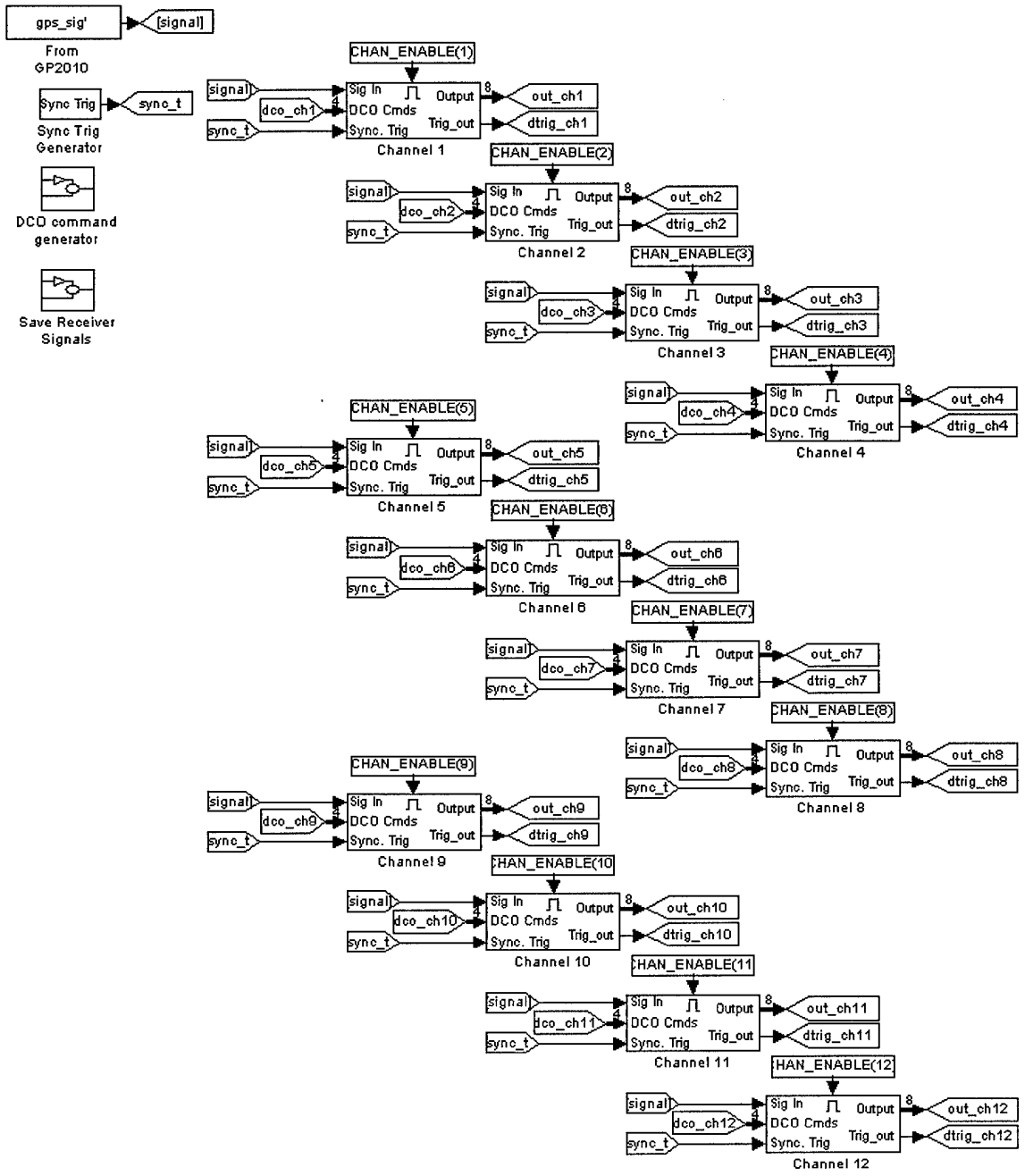


Figure 3-20. 12-channel Receiver Model (top level)

The collection of 12 channels makes the diagram very large, but it is useful to see the how blocks can be easily duplicated in Simulink. Each block is nearly the same, with only a few variables that change from channel to channel. In Simulink, connections are typically shown with lines from block to block, but here the "goto" and "from" shortcuts have been used extensively reduce clutter caused by many lines. These shortcuts provide the same function as individual lines.

The inputs to each block consist of the sampled GPS signal, the DCO commands for each channel, and the synchronized latch trigger. The outputs are the eight observables previously described in the bulleted list at the beginning of Section 3.3, and the trigger signal that latches those eight observables inside the channel.

Figure 3-21 displays the contents of a single channel. The progression of the input signal through the processing blocks to the output can be clearly seen.

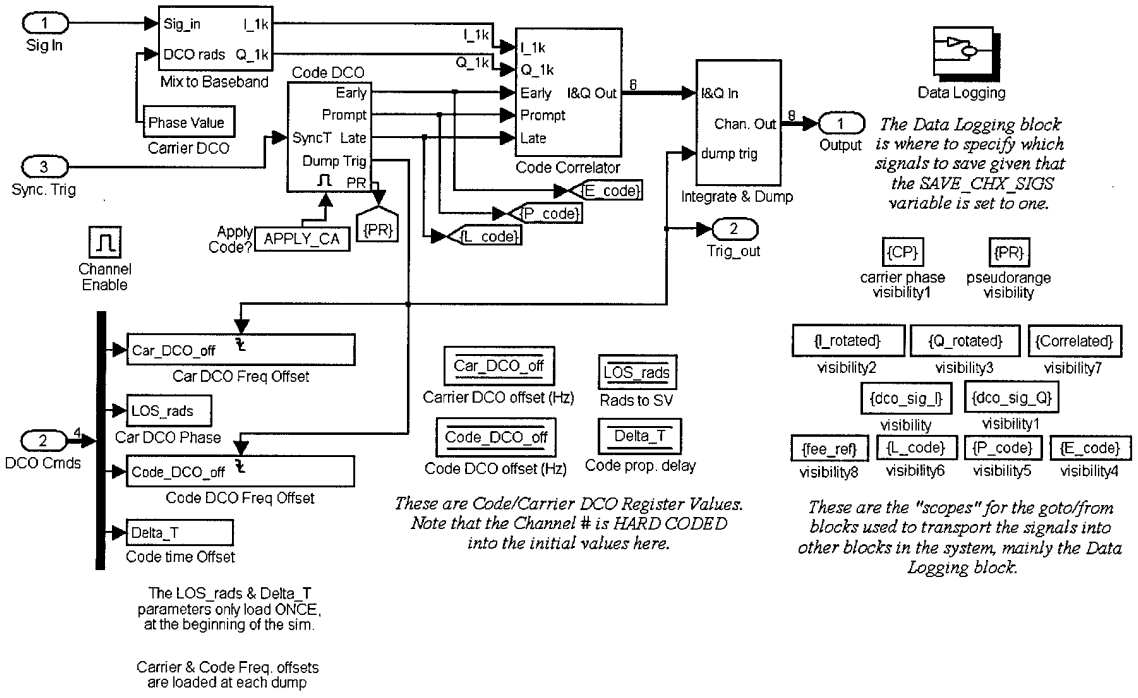


Figure 3-21. Single Receiver Channel

The discussion of the entire loop is be divided into analysis of the individual blocks. The first block to analyze is the carrier DCO block, shown in Figure 3-22. The block generates the phase of the carrier DCO in radians, based on the initial phase and frequency offset due to Doppler and any clock error. Logic is included that implements phase and Doppler reversal in the case that the Nyquist sampling criterion is not satisfied.

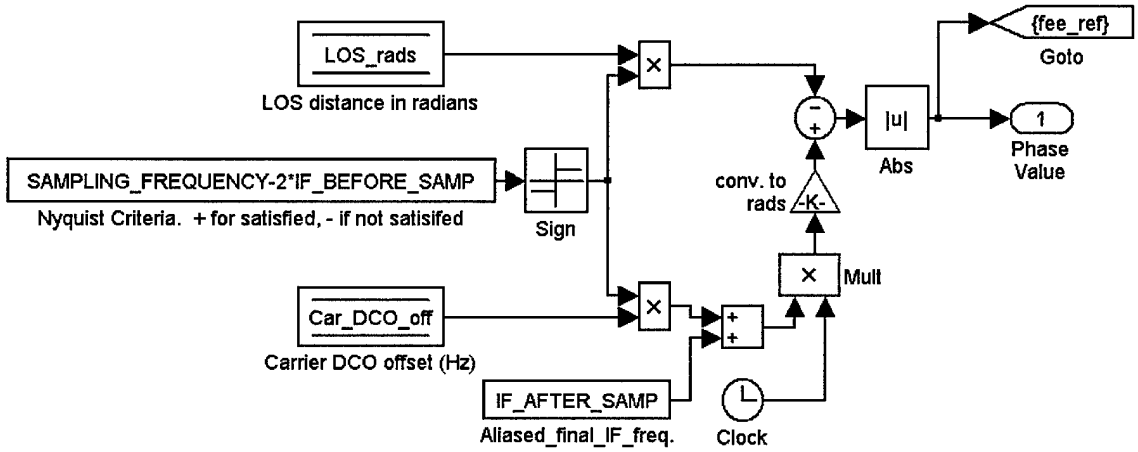


Figure 3-22. Carrier DCO block

The carrier (and code) DCO offset registers contain the *perceived* Doppler value of the received signal, defined as positive for decreasing LOS distances. If the Nyquist criterion is satisfied, the perceived Doppler is added to the nominal IF frequency, where if aliasing does occur, the Doppler frequency must be subtracted from the nominal IF frequency. The initial phase is reversed in the same fashion.

The carrier DCO radians are fed to the Mix to baseband block (Figure 3-23), where they are converted into the DCO waveforms by the DCO signal synthesizer (Figure 3-24). A custom quantization block (Figure 3-25) was built to generate the waveforms per the Mitel specification.

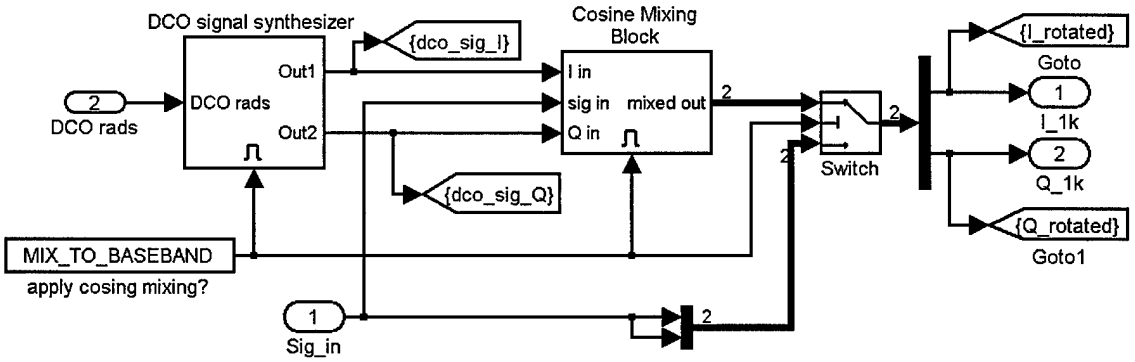


Figure 3-23. Mix to baseband block

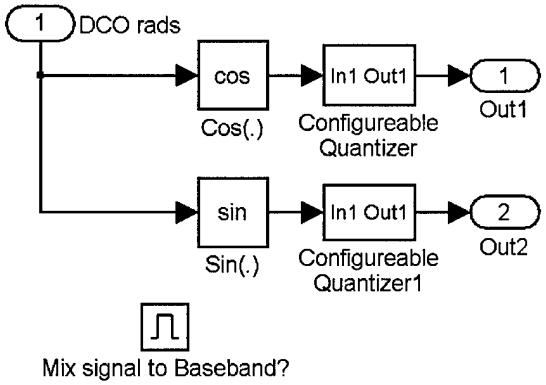


Figure 3-24. DCO signal synthesizer block

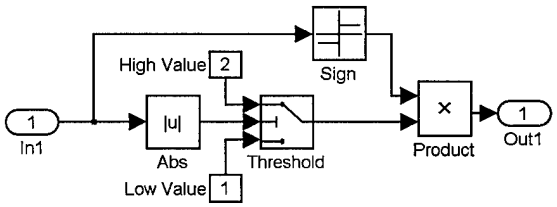


Figure 3-25. Configurable quantizer block

The DCO signals are then mixed with the received signal in the Cosine mixing block, shown in Figure 3-26. The MIX_TO_BASEBAND enables and disables this process.

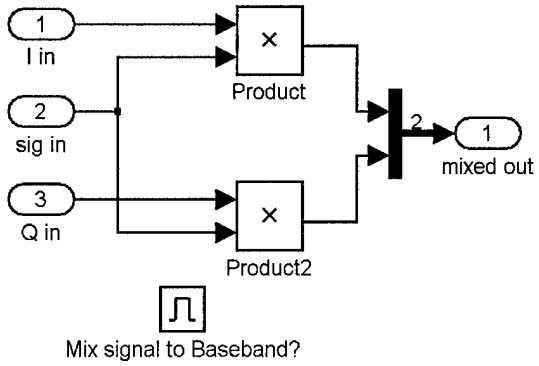


Figure 3-26. Cosine mixing block

After the signal is mixed to baseband, it is correlated with the local code replicas generated by the code DCO block, shown in Figure 3-27. This block generates the receiver's estimate of satellite time (Figure 3-28) and converts the satellite time into the appropriate chip number of the C/A code that was generated at that time. The chip number (1-1023) is then fed to a look-up table to generate the correct chip value, which is output for each sample time. The CHIP_SPACING parameter subtracts or adds the appropriate amount of time to advance or delay the code waveform by the desired chip spacing. This allows for the chips spacing to be precisely altered with the use of any sampling frequency.

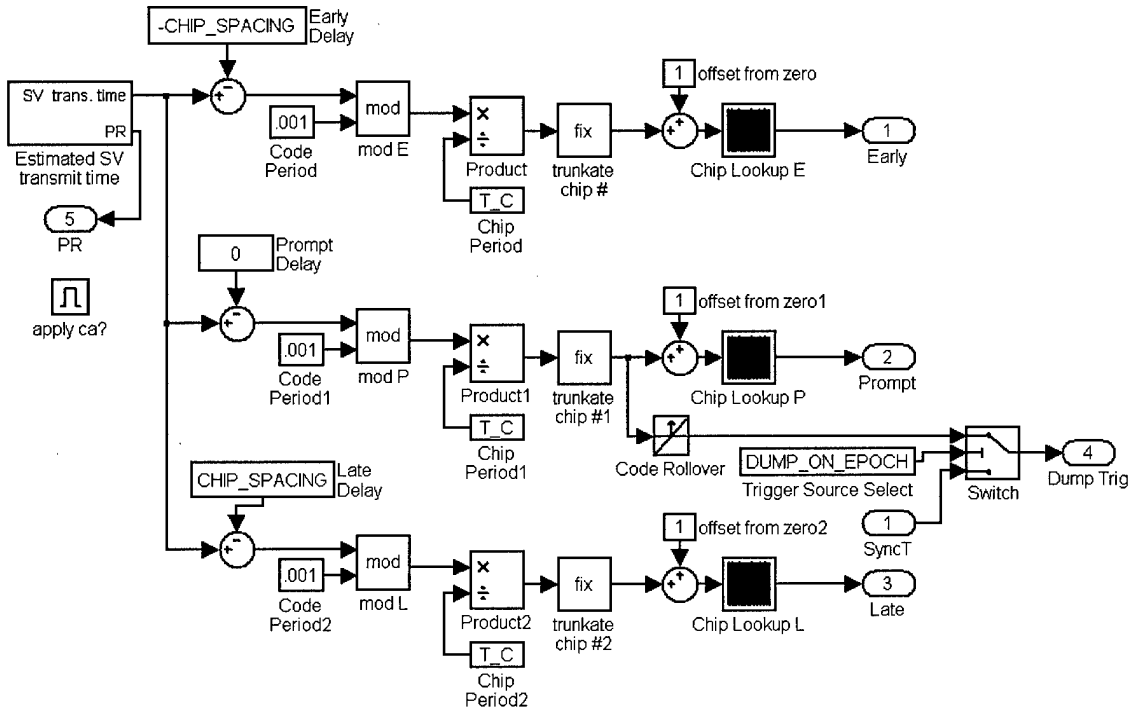


Figure 3-27. Code DCO block

The initial satellite time is derived from the "phase" of the code DCO, which is equivalent to the pseudorange. This time is updated by integrating the code Doppler,

which is a measure of the LOS velocity. The algorithm is pictorially displayed in Figure 3-28. A positive Doppler frequency indicates a decreasing distance; therefore a negative sign is required in the "Hz to s/s" conversion. This block also generates the pseudorange variable, as it is fundamentally defined as the delay in the code tracking loop multiplied by the speed of light.

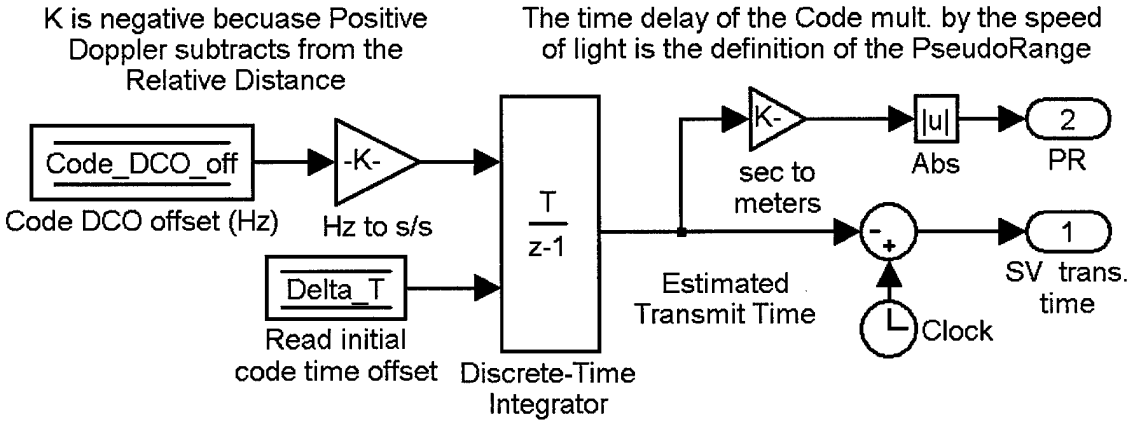


Figure 3-28. Estimated Satellite Vehicle (SV) transmit time block

The code correlation block (Figure 3-29) consists simply the signal multiplied with the locally generated codes. The ability exists to bypass this process by setting the APPLY_CA global variable to zero. The multiplication process is shown in Figure 3-30.

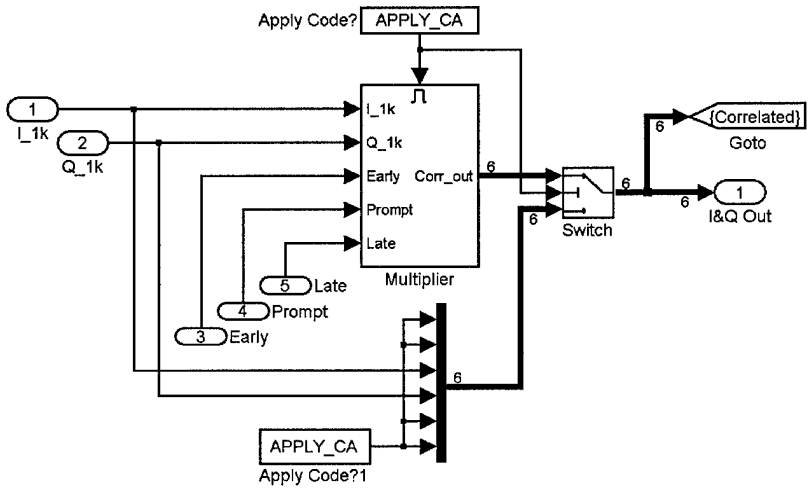


Figure 3-29. Code correlator block

In this block, the signals become multiplexed onto a single line. This is a very powerful feature of Simulink® that allows a single line to contain multiple signals, on which blocks operate on individually. This limits duplication of the same functional blocks for different signals.

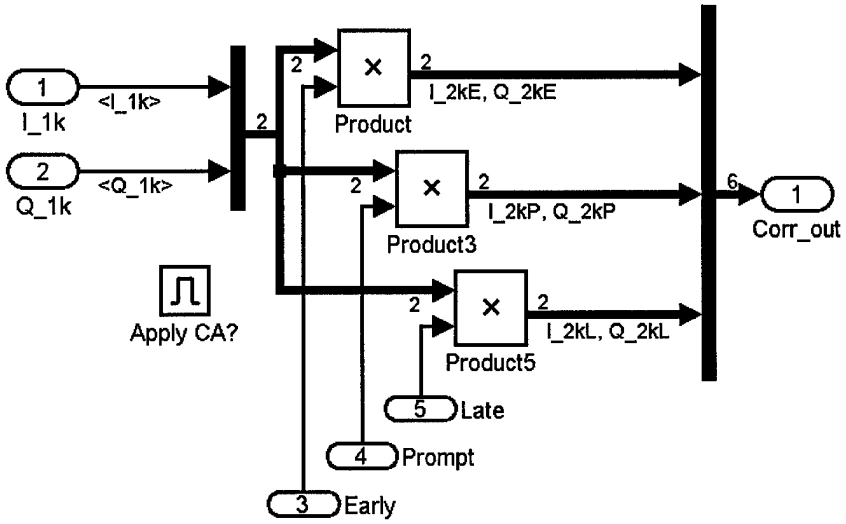


Figure 3-30. Multiplier block

After correlation, the samples have been completely processed and are ready to be accumulated. This is done in the integrate & dump block (Figure 3-31). Here the frequency of the information is reduced from the sampling frequency to the "latching rate," specified by the LATCH_DT global parameter. In the Mitel chipset, as is common in other receivers as well, the latching rate is 1 kHz, 'dumping' the contents of the accumulator every millisecond. This is usually done synchronously with the code epoch, or rollover of the code sequence, on each channel. This is because the data bit transitions occur at the same time, and latching at this point prevents the integration over a data bit transition, which could cause a loss of signal strength. However, in testing advanced receiver designs and developing new integration techniques, it is often useful to "latch"

all the channels at once. This is particularly useful when feeding a Kalman Filter, as implementation tends to be less complicated when measurements arrive at the same time. In such cases, data modulation is often not simulated.

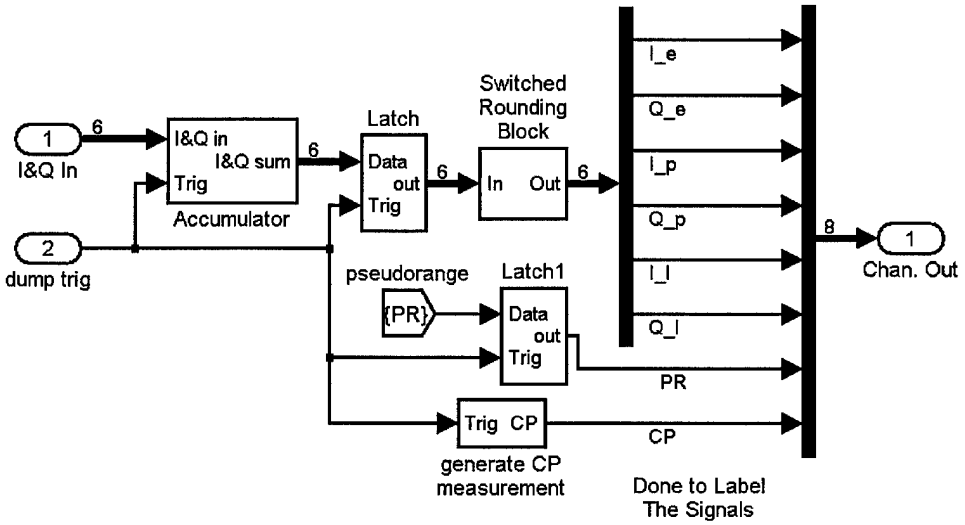


Figure 3-31. Integrate & dump block

The carrier-phase measurement generator is included in this block, and is detailed in Figure 3-32. This block integrates the perceived offset in frequency over the course of the run. Again, the need for the negative sign due to the way Doppler has been defined in this research.

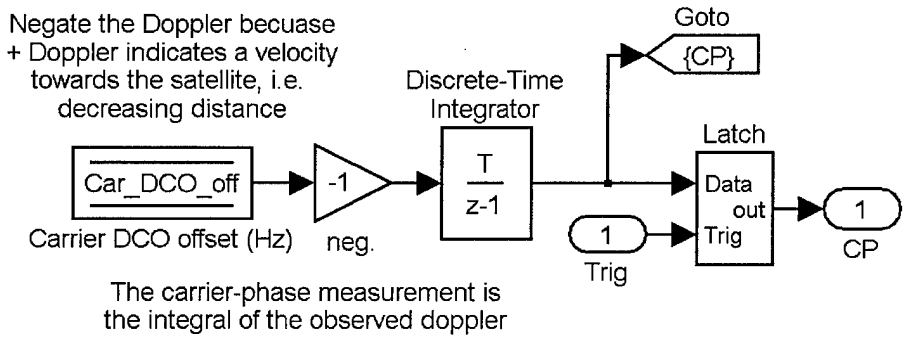


Figure 3-32. Carrier cycle counter block

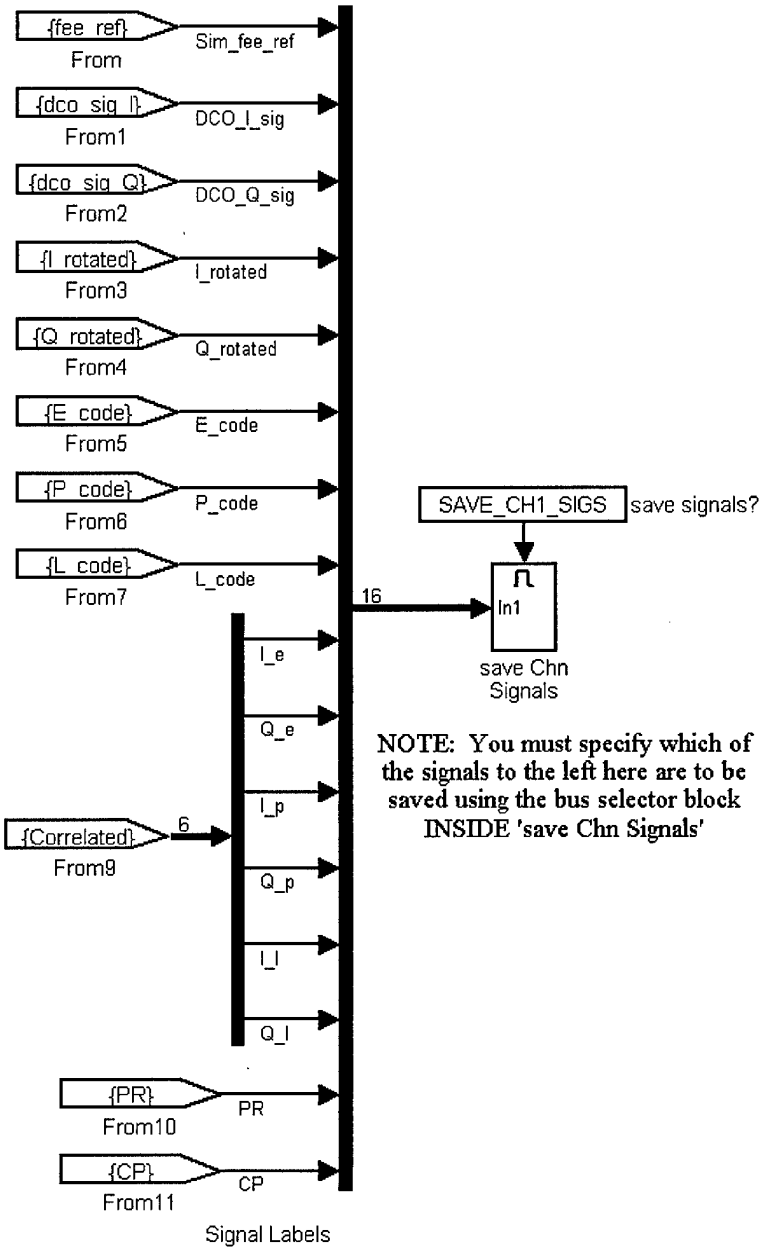
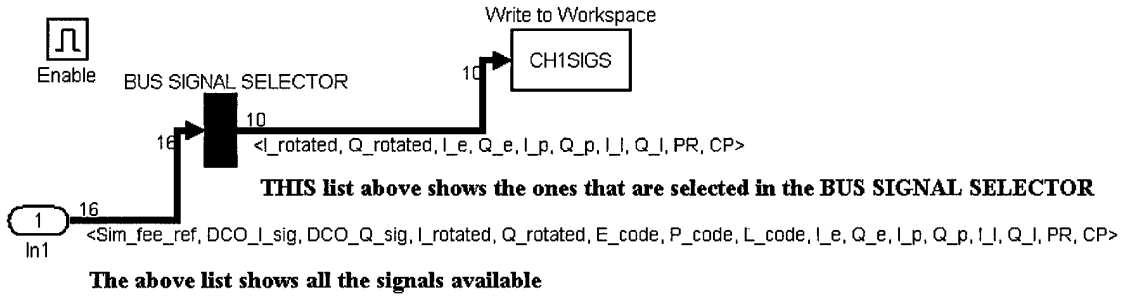


Figure 3-33. Data logging block

Figure 3-33 displays the data logging mechanism available in each channel. Many of the previous blocks have had "Goto" blocks that match these "From" blocks. All the signals in the model that are significant are brought into this block for potential data logging. The global variable SAVE_CH1_SIGS enables data logging on channel

one. If this variable is not set, no data is logged regardless of the settings inside "save Chn Signals." The inside of this block is shown in Figure 3-34. Here the signals are selected using the signal bus selector block, which is the solid black rectangle in Figure 3-34.



Directions for selecting signals:
1. **Double Click the BUS SIGNAL SELECTOR**
2. **Select the signals on the right that you want to record during the simulation.**
3. **Assign them to the output by clicking "select"**
4. **The order they appear in the Right pane is the order they will be stored in the output variable**

Figure 3-34. Save channel signals block

Figure 3-35 shows the dialog box for the bus selector. Here the user has the ability to select the signals individually to be saved during a simulation. These signals are saved at the decimated sampling rate, where the decimation factor is specified by the global variable DEBUG_DECIMATION.

This concludes the discussion of a single channel. There are a few blocks that interface with all the channels in the model. One such block is the synchronized trigger generator (Figure 3-36). This block generates a common trigger signal for latching the outputs of all the blocks in the system simultaneously. The rate at which the blocks are latched is specified by the global variable LATCHING_DT.

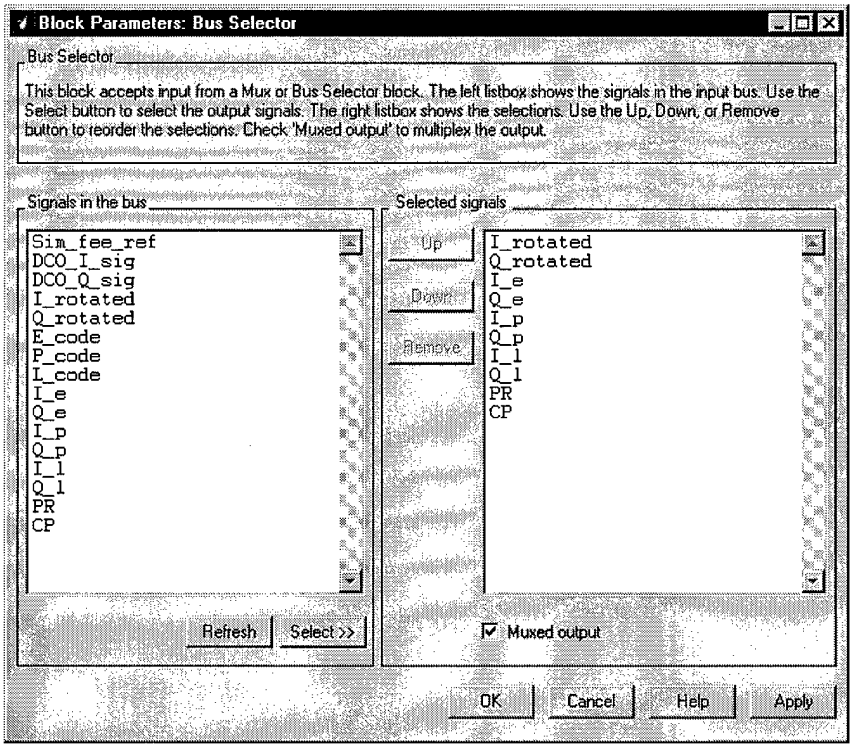


Figure 3-35. Bus selector dialog box

The "last sample" block detects when an input rises above a given threshold, generating a logic "1" for one sample time. The SAVE_TIME global variable allows the logging of the simulation time and the "mod time," which resets to zero after each trigger. The trigger signal is added to the "mod time" signal to reduce redundancy, as the trigger signal is zero most of the time.

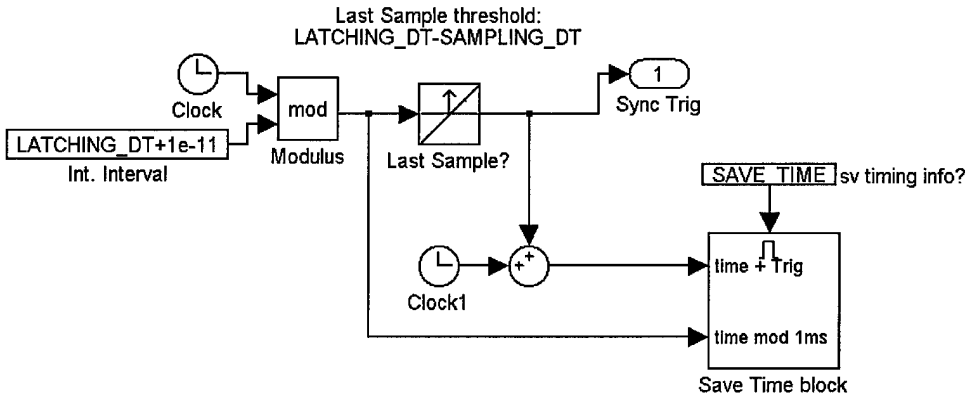


Figure 3-36. Synchronous trigger generator block

Because a full tracking loop is not within the scope of this research, the DCO values must be set manually and fixed during the runs performed in this thesis. To set the initial DCO values, the output variables from the SS are used to pre-set the DCOs to the perfect values at the start of the simulation. The block that accomplishes this is shown in Figure 3-37. This figure is provided to equate the outputs of the SS with the DCO values of the RM.

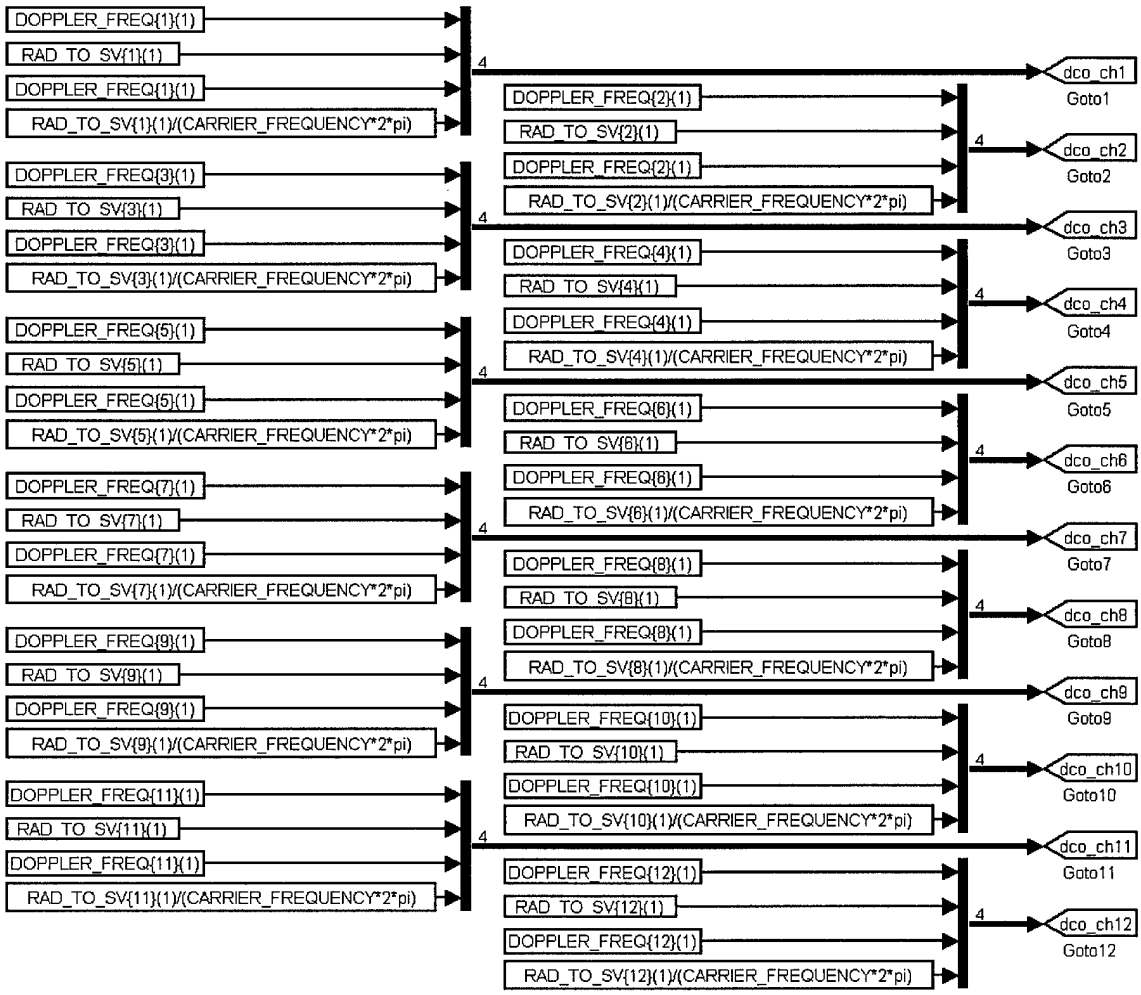


Figure 3-37. DCO command generator

The final block covered here is the "Save Receiver Signals" block, shown in Figure 3-38. This block receives and records the outputs of all the channels during a run.

It is like the "save channel signals" block in that it utilizes a bus selector for selecting which signals are to be saved, and the signals available in the "Save Receiver Signals" block are all available in the "save channel signals" block as well. This block however, saves the signals only when a trigger hits.

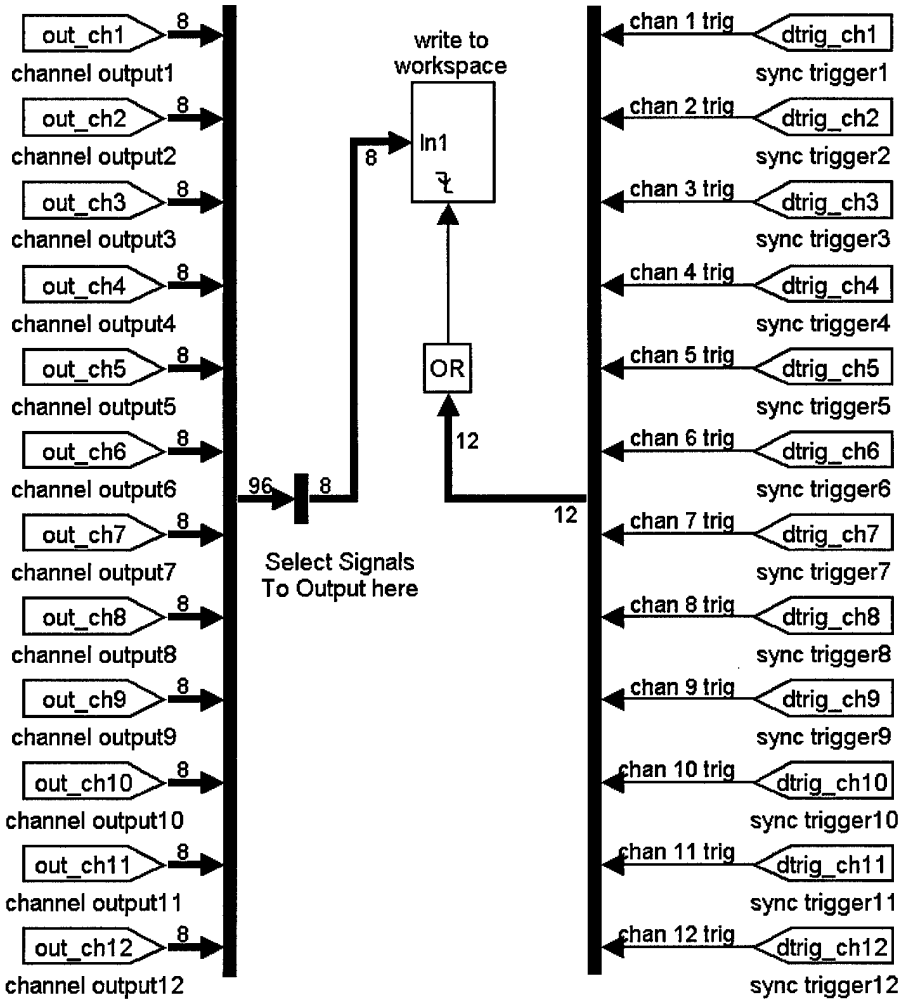


Figure 3-38. Save receiver signals block

This centralized signal storage block is intended for recording receiver output for most applications. If all 12 channels are simultaneously active, 96 signals are available for logging. If the synchronized trigger is used, these signals are saved at the rate

specified by LATCHING_DT. However, if the source of the trigger is from the code epoch in each channel, the signals are logged at 12 times the code epoch rate, or at 12 kHz. That is because a trigger from any channel saves all the signals, and all channels will experience a code epoch within a 1ms time window. Thus if the synchronized pulse is set to occur at the code epoch rate, latching the data per channel will cause the output file to increase by a factor of 12.

Figure 3-39 shows the bus selector dialog box for the bus selector in Figure 3-38. Here each of the channels show up as composite signals, which can be selected in whole or in part, as is shown in the figure. This allows the user the ultimate flexibility in picking which signals are to be logged from a given run.

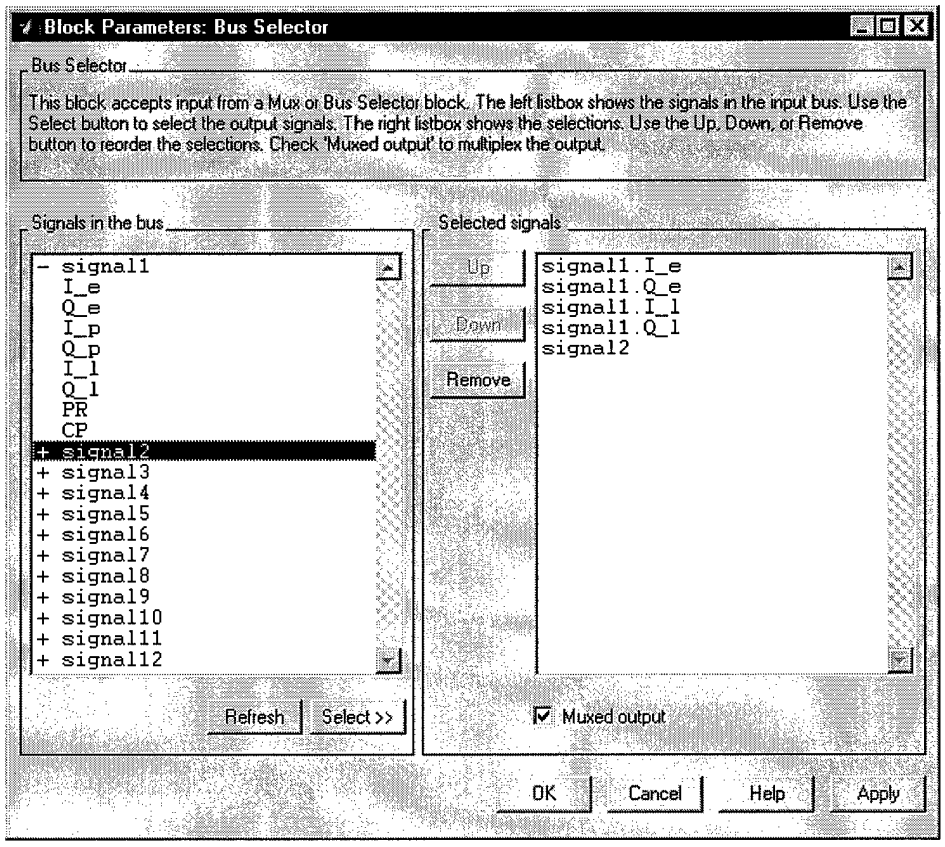


Figure 3-39. Save receiver signals bus selector block

3.3.2 *Noise floor derivation*

All the signals in the GP2021 tracking module are represented by integers, which allow fast integer math calculations in the GP2021. The integer values in the tracking channel are shown in Figure 3-19 next to the signal paths. Thus, the accumulated value is the sum of only 4 integer magnitudes. If no signal is present in the system, these accumulators continue to generate sums of these numbers at the output. In order to determine the presence of a signal, a noise floor estimate of this system must be found. This is accomplished by calculating the expected value of the system with no signal present. The derivation of the noise floor and the resulting equations for determining S/N, phase lock, and frequency lock for this chipset are provided by [20]

The calculation of the noise floor is the process of estimating the power of the accumulated samples with no signal present. The derivation is an exercise in random variable math, calculating the mean and variance of discrete values with a given probability of occurrence. The values involved in this calculation are the quantized signal levels and those generated by the DCO. With no signal present, the values and their distributions are given by Table 3.9.

Table 3.9. Distribution of sampled signal and DCO values

Quantized Signal: "S"		DCO: "D"	
Value	Probability	Value	Probability
-3	0.15	-2	0.25
-1	0.35	-1	0.25
1	0.35	1	0.25
3	0.15	2	0.25
$E\{S\} = 0$	$E\{S^2\} = 3.4$	$E\{D\} = 0$	$E\{D^2\} = 1.5$

These distributions apply to the signal with noise as well, the difference being in the correlation of adjacent samples. Without a signal, the samples representing the quantized signal are completely uncorrelated. The DCO values follow a deterministic sequence, representing a cosine (or sine) wave at a given IF frequency. The sequence is not perfectly repetitious, due to the changing frequency of the DCO to track the signal. The noise floor expected variance of the product of these two random sequences. This process happens twice for each channel, as the signal is split into it's I and Q components prior to processing. Let "I" and "Q" each represent the summation of a sequence of "N" products of the DCO and quantized signal. The expected noise floor is given by Equation (3.38):

$$E(I^2 + Q^2) \tag{3.21}$$

Given that the signal is completely random, this is equivalent to:

$$E(I^2) + E(Q^2) \tag{3.22}$$

Additionally, since the DCO sequences for the I and Q are identical, with one delayed in reference to the other, and with the provision that the signal is totally random, this is equivalent to:

$$2E(I^2) \tag{3.23}$$

Now, representing the Signal with a discrete random variable "S" and the DCO with a discrete random variable "D", given an accumulation of "N" samples, Equation (3.24) shows the appropriate substitutions for I:

$$2E\left\{\left(\sum_{i=1}^{N} S \cdot D\right)^2\right\} = 2E\left\{\left(\sum_{i=1}^{N} S \cdot D\right)\left(\sum_{i=1}^{N} S \cdot D\right)\right\} \tag{3.24}$$

Create a discrete random variable I_i, defined as the product of the i^{th} S and ith D realization of the random variables S and D:

$$I_1 = S_1 \cdot D_1, \quad I_2 = S_2 \cdot D_{21}, \quad I_3 = S_3 \cdot D_3, \cdots\cdots I_N = S_N \cdot D_N \tag{3.25}$$

Taking the product of the two random variables give us a discrete probability distribution for I_i, given in Table 3.10

Table 3.10. Distribution of samples after mixing

Distribution of I = (S)(D)	
Value	Probability
+6, -6	0.075
+3, -3	0.075
+2, -2	0.175
+1, -1	0.175
E{I} = 0	$E\{I^2\} = 8.5$

Substituting in for S and D, and expanding the summations, the equation is now:

$$2E\{(I_1 + I_2 + I_3 + \cdots I_N) \cdot (I_1 + I_2 + I_3 + \cdots I_N)\} \tag{3.26}$$

Now, recombine the terms under new summations:

$$2E\left\{\left(\sum_{i=1}^{N} I_i^2\right)_{for\ i=j} + \left(\sum_{i=1}^{N}\sum_{j=1}^{N} I_i I_j\right)_{for\ i\neq j}\right\} \tag{3.27}$$

Using the distributive property of the expected value operator, we arrive at Equation (3.28).

$$2\sum_{i=1}^{N} E\{I_i^2\} + 2\sum_{i=1}^{N}\sum_{j=1}^{N} E\{I_i I_j\} \tag{3.28}$$

Again applying the property of independence, based on our signal having independent samples, we have:

$$2\sum_{i=1}^{N} E\{I_i^2\} + 2\sum_{i=1}^{N}\sum_{j=1}^{N} E(I_i)E(I_j) \tag{3.29}$$

Because the mean of I is zero, the second term is eliminated, giving us a final noise floor value of:

$$2\sum_{i=1}^{N} E\{I^2\} = N \cdot E\{I^2 + Q^2\} = 2 \cdot N \cdot 8.5 \tag{3.30}$$

Where N is the number of samples summed into I and Q. The typical integrate and dump cycle in a GPS C/A code receiver is 1ms, and the sampling frequency of the Mitel chipset is ~5.714 MHz, so the expected noise floor given these provisions is nominally equal to 97142. This noise floor can be used to estimate the S/N ratio of the signal as follows:

$$\frac{S}{N} = 10 * LOG\left(\frac{I^2 + Q^2}{2 \cdot N_{samp} \cdot 8.5}\right) \tag{3.31}$$

This is the formula used by Mitel to express the S/N ratio of the received signal[20]. The number of samples determines the processing gain. However, this does not represent the actual S/N received. Consider the ideal case where the signal is phase locked such that all the power is in the I term, and the Q term averages to zero over the integration. Because the DCO has an amplitude of 2, the cosine mixing process does not alter the amplitude of the signal. Recall:

$$A\cos(\omega \cdot t) \cdot 2\cos(\hat{\omega} \cdot t) = \frac{2 \cdot A}{2}\left(\cos((\omega + \hat{\omega})t) + \cos(\omega - \hat{\omega})t)\right) = A + A\cos((\omega + \hat{\omega})t) \tag{3.32}$$

This mixing process occurs in the GP2021. The equation on the far right is valid when the frequencies of the cosine terms are equal. As the samples are accumulated, the cosine term averages to zero, and the resulting summation is ideally equal to A times number of samples. The normalization to the number of samples is taken care of in the N_{samp} term in the denominator of (3.31). Hence, the power in the signal here is represented as A^2, but the standard equation for the Power in a carrier wave is $A^2/2$. Thus, the S/N ratio power estimate generated by (3.31) is 3 dB higher than the actual received power. This phenomenon is clearly seen in Chapter 4 with the analysis of real data.

This concludes the discussion on the Receiver Model. The SS and RM work in concert to generate processed I and Q samples of the received signal. Both the SS and the RM run at the full sampling rate of a receiver, which is ~5.7 MHz in the case of the Mitel chips. With all the high-fidelity simulation and processing that goes on, a substantial price must be paid in the time and computing power required. This level of modeling is not practical for simulating an integrated GPS/INS system. A faster method for generating the same observables is required, and was created out of this research. The I and Q model (IQMDL) is the topic of the next section.

3.4 GPS Analytical Accumulated I and Q model

The analytical I and Q model is a complete stand-alone module which generates code-correlated, phase-rotated, accumulated I and Q samples. It is entirely managed using function calls in Matlab®, and all variables are passed through the function calls. It does not currently model *any* GPS error sources, and is only a L1 C/A code implementation. Nonetheless, it provides good dynamic modeling and faithfully

represents the effects of varying pre-integration times, S/N ratios, and DCO errors. An Ideal autocorrelation function is used, as well as a proven mathematical model for the accumulated I and Q values as a function of carrier DCO phase and frequency offsets.

The program implements several functions, selected by specifying the "task" to be performed as a string in the first input argument. The input and output arguments are dependent on the task that is issued. The four tasks currently implemented are initialization (init), re-start at a particular time (rstrt), DCO updating (updtdco), and I & Q generation (geniq). Their input/output arguments are listed in Table 3.11.

The very first call to the I/Q model should always be an init function. The algorithm for determining satellite selection is one that ensures that all the satellites are visible (above the specified cutoff angle) over the course of the simulation, defined by the times in the receiver position input file. The receiver trajectory must be in ECEF coordinates. The receiver trajectory file format is specified in the function listing in Appendix C. This first call requires a substantial amount of time to complete, as all of the satellite positions being simulated at each time step in the trajectory file are calculated. The satellite positions are generated up front so that the multiple runs of a simulation are not burdened with re-calculating SV positions at each call. As the methods for calculating satellite positions are outside the scope of this thesis, the functions for generating the satellite matrices are not covered here.

Table 3.11. Operational interface for the GPS I/Q model

Operational Summary for gps_iq_model:		
Function call:		
[argout1,argout2,argout3]=gps_iq_model(task, argin1,argin2,argin3,argin4)		
Task: 'init'		
Input/Output	**Variable name**	**Description**
argin1	eph_filename	Ephemeris filename, structure is that used at AFIT in the the GPS courses [37]
argin2	recv_pos_mat	receiver position file, including time
argin3	START_TIME	Start time in GPS seconds. Used only if the first time in the receiver file time vector is zero
argin4	elev_cutoff	Cutoff angle for satellite visibility. Default: 5 degrees.
argout1	sv_pos_mat	matrix of satellite positions for all visible satellites and each time step
Task: 'rstrt'		
Input/Output	**Variable name**	**Description**
argin1	init_time	Time, in GPS seconds, to re-initialize to
argout1	doppler_mat	Vector of Doppler values
Task: 'updtdco'		
Input/Output	**Variable name**	**Description**
argin1	car_dco_corr	Vector of carrier DCO frequency adjustments
argin2	code_dco_corr	Vector of code DCO frequency adjustments
argin3	UPDT_TIME	Time stamp that the DCO settings are updated
Task: 'geniq'		
Input/Output	**Variable name**	**Description**
argin1	LATCH_TIME	Simulation time to generate I and Q values (s)
argout1	EPL_IQ	Cell array of early, prompt, & late I and Q values
argout2	PR	Vector of pseudoranges
argout3	CP	Vector of carrier-phase measurements

The 'rstrt' task is used for restarting and initializing a simulation without recalculating the satellite positions. The DCO settings are cleared whenever rstrt is called. This function cannot be used to jump to another location in time while preserving the current DCO phases and frequencies.

The 'updtdco' task has two main steps. First it propagates the receiver environment to the time specified in the function call, then it updates the DCO values. The IQMOD adds the DCO frequency updates to the existing values.

The 'geniq' command is the main workhorse of the IQMOD. It is responsible for modeling the I and Q values and propagating the receiver environment forward in time. Before discussing the models used, the global variable list for this function must be revealed. The global variables are listed and defined in IQ_model_globals.m. The user should use caution when running the IQMOD with either the SS or RM in the same Matlab® workspace, because they do use some of the same global variables. The global variable list is given in Table 3.12.

The I and Q model is a function of the following quantities:

- ✓ Signal/Noise ratio of the signal
- ✓ Integration time over which the samples are "Accumulated"
- ✓ Code and Carrier DCO Phase and frequency errors at the beginning of the integration interval.

Although never true in the real world, for simulation purposes the S/N ratio is set at the beginning of a simulation and is therefore constant over the simulation. The integration time is the time that has lapsed since the last call to the function. Therefore, the IQMDL has to only to keep track of the Code and Carrier DCO phase and frequency errors. A cubic spline function is used to generate the truth LOS distances and velocities. Additionally, the DCO phase and frequency values are maintained and updated. The differencing of the truth LOS values and the DCO estimates generates the carrier-phase, pseudorange, and DCO errors.

Table 3.12. IQMOD global variables

Variable Name	Description / Value
CARRIER_FREQUENCY	L1 received frequency = 1575.42e6 (Hz)
CARRIER_WAVELENGTH	L1 carrier wavelength = 0.1902936 (m)
SPEED_OF_LIGHT	299792458 (m/s)
T_C	Length of one C/A code Chip in seconds: 9.775171e-7
CHIP_SPACING	Correlator chip spacing (s)
SAMPLING_FREQUENCY	Mitel GP2021 Sampling Frequency
SNR_DB	S/N ratio of the signals in PRN_VEC
NO_NOISE	Switch turning off the noise, giving perfect samples
PRN_VEC	Vector of PRNs, chosen by the IQMOD
LAST_LATCH_TIME	Stores the last time, in GPS seconds, that the model was updated (s)
TIME_OFF	Time offset of the internal time vector to GPS time
CAR_DCO_PHASE_ERR	Carrier phase error vector (radians)
TAU	Code correlation error vector (seconds)
CAR_DCO_FREQ_ERR	Carrier DCO frequency error vector (Hz)
CODE_DCO_FREQ_ERR	Code DCO frequency error vector (Hz)
CAR_DCO_FREQ_MAT	Vector of carrier DCO offsets from nominal for each signal (Hz)
CODE_DCO_FREQ_MAT	Vector of code DCO offsets from nominal for each signal (Hz)
CAR_DCO_PHZ_MAT	Vector of LOS distances to each SV as estimated by the DCO (rads)
CODE_DCO_DELAY_MAT	Vector of transmit time estimates for each SV as estimated by the DCO (sec)
LOS_RANGES_MAT	Calculated matrix of LOS ranges for each satellite at the user specified time (from the function call)
LOS_DIST_MAT	Matrix of perfect pseudoranges for each satellite at each timestep (includes time vector)
DOPPLER_FREQ_MAT	Calculated matrix of Doppler Frequencies at the user specified time (from the function call)

The true LOS range and velocity is calculated for each signal path using the same spline fit techniques used in the SS (see section 3.2.3.1). This guarantees the truth environment for both the SS and the IQMDL is the same. Code and carrier DCO frequency and phase errors are calculated by differencing the current DCO values with the truth environment. The DCO "frequencies" are only the offsets from an arbitrary

nominal frequency caused by Doppler, i.e. the DCO Doppler estimate. Both the code and carrier DCOs are expressed in L1 Doppler values. The "phase" of the DCOs are implemented as the DCO LOS distance estimate. The carrier DCO phase is expressed in LOS radians, where the code DCO is expressed in seconds of propagation time.

The derivation of the I and Q model is shown for one channel, as the process is the same for each. Thus, channel numbering is not included in the equations. The following analysis is done for the carrier DCO, the code DCO derivation is essentially identical. Sign conventions are very important here, especially when aliasing is involved. Aliasing effectively reverses the sign of frequency and phase errors from what they would be if the Nyquist sampling criteria were met. Equations (3.33), (3.34), and (3.35) define the integration time, the DCO frequency error, and the DCO phase error, respectively. The phase and frequency errors are defined for the last latch time, or the beginning of the integration interval.

$$\Delta T = t_{latch} - t_{last\_latch} \tag{3.33}$$

where:

t_{latch}	=	Latch time specified in the function call
t_{last_latch}	=	Latch time of the last function call
ΔT	=	Integration time; time lapsed between calls.

$$\delta\omega = \omega_{DOPP} - \omega_{DCO} \quad @\ t_{last\_latch} \tag{3.34}$$

where:

$\delta\omega$	=	DCO frequency error
ω_{DOPP}	=	True Doppler frequency
ω_{DCO}	=	DCO frequency setting

$$\delta\phi = \phi_{TRUE} - \phi_{DCO} \quad @\ t_{last\_latch} \tag{3.35}$$

where:

$\delta\phi$ = DCO phase error

ϕ_{TRUE} = True phase of the signal

ϕ_{DCO} = DCO phase of the signal

These equations are paralleled for each of the 12 channels in the IQMOD. Using the errors defined above, the I and Q error equations, shown in Equations (3.36) and (3.37), are applied to simulate the effects of the DCO phase and frequency offsets over the integration interval. These equations are similar to those first published by Sennott [42] and derived into a more robust form by Ward [58]. These equations adequately simulate the effects of the integration process, which is not taken into account by the equations in [33].

$$I = \sqrt{2P_{sig}}\,N\cos\left(\frac{\Delta T\delta\omega}{2}+\delta\phi\right)\mathrm{sinc}\left(\frac{\Delta T\delta\omega}{2}\right) \tag{3.36}$$

$$Q = \sqrt{2P_{sig}}\,N\sin\left(\frac{\Delta T\delta\omega}{2}+\delta\phi\right)\mathrm{sinc}\left(\frac{\Delta T\delta\omega}{2}\right) \tag{3.37}$$

where:

I, Q = uncorrupted I and Q magnitudes as a function of the DCO settings, signal power, and truth environment

P_{sig} = Power of the signal in *one* sample, using the variance defined previously in section 3.3.2.

N = Number of samples in the integration time

ΔT = Integration time

$\delta\omega, \delta\phi$ = Frequency and Phase errors

The assumption that makes Equations (3.36) and (3.37) realizable is that the true Doppler frequency is constant over the integration interval. Without this assumption, the equation that must be integrated has a t^2 term inside of a cosine expression, which makes a closed form solution very difficult, if possible at all. This assumption makes the dynamic model inaccurate during long integration times in high dynamics. To counter this effect, it is

recommended that the system interfacing with the IQMDL call it at a rate higher than that required by the system itself, in order to limit the errors caused by integrating for long time periods during high dynamics. The system would then sum the outputs over time to generate the rate desired.

With the proper carrier modeling done, the I and Q values are now scaled according to the code misalignment. The autocorrelation function "R( )", was described in section 2.2.1 and shown in Figure 2-4.

$$\begin{aligned} I_E &= I \cdot R\left(\tau - \frac{\Delta}{2}\right) + n(t) & Q_E &= Q \cdot R\left(\tau - \frac{\Delta}{2}\right) + n(t) \\ I_P &= I \cdot R(\tau) + n(t) & Q_P &= Q \cdot R(\tau) + n(t) \\ I_L &= I \cdot R\left(\tau + \frac{\Delta}{2}\right) + n(t) & Q_L &= Q \cdot R\left(\tau + \frac{\Delta}{2}\right) + n(t) \end{aligned} \tag{3.38}$$

where:

I_E, Q_E	=	Early I and Q values
I_P, Q_P	=	Prompt I and Q values
I_L, Q_L	=	Late I and Q values
τ	=	Code tracking error (s); misalignment of DCO code with the code of the "received" signal
Δ	=	One chip width (s)
$n(t)$	=	Noise, where the variance, σ_n^2, is related to the noise floor and the number of samples, N, by Equation (3.39):

$$\sigma_n^2 = N * 8.5 \tag{3.39}$$

The I and Q values now have been completely defined. The code DCO is "controlled" through the same equations as the carrier DCO, although expressed in different units. The phase of the carrier DCO is the LOS distance between the receiver and the respective satellite, expressed in radians of the L1 carrier frequency. The "phase" of the code DCO is simply expressed as the signal propagation time delay, which when multiplied by the speed of light yields the PR measurement. Although code Doppler should be defined

referenced to a code period by definition, the code DCO frequency is expressed in terms of the carrier frequency. s as the units of choice for defining LOS velocity. The code phase is essentially the PR stored as time, which is in error by τ seconds. After specifying the I and Q values, the only task remaining is to propagate the DCO values to the current time step, which is specified by the time input parameter to the function call. The propagation equations for the code and carrier DCOs are given in Equations (3.40). and (3.41).

$$\phi_{CARRIER\_DCO} = \phi_{CARRIER\_DCO} - \omega_{CARRIER\_DCO} \cdot \Delta T \tag{3.40}$$

$$\frac{PR_{DCO}}{c} = \varphi_{CODE\_DCO} = \varphi_{CODE\_DCO} - \omega_{CODE\_DCO} \cdot \Delta T \tag{3.41}$$

where

$\phi_{CARRIER\_DCO}$	=	Carrier DCO phase (radians)
$\omega_{CARRIER\_DCO}$	=	Carrier DCO frequency (rad/sec)
PR_{DCO}	=	Pseudorange estimated from code DCO (DCO phase)x(c)
φ_{CODE_DCO}	=	Code DCO phase (s)
ω_{CODE_DCO}	=	Code DCO frequency (rad/sec)
c	=	Speed of Light

As with most real receivers, the phase of the DCOs is not modifiable during a simulation. Only the frequency may be modified by the user. The phase of the DCO can only be updated in a reset call to the 'rstrt' task. When the 'updtdco' routine is called, the DCO values are first propagated up to the update time, and then the DCOs are modified using the equation below:

$$\omega_{DCO} = \omega_{DCO} + \omega_{CORR} \tag{3.42}$$

where

ω_{CORR} = Frequency correction to the DCO

Both the code and carrier DCOs use this same notation. Although the true definition of code Doppler would be defined in terms of the code frequency, the code DCO phase shown here is defined in terms of the L1 Carrier frequency, with the variables previously defined.

$$\phi_{DCO} = \phi_{DCO} - \omega_{DCO} \cdot \Delta T \tag{3.43}$$

This concludes the discussion on the I and Q model derivation. Chapter 4 will cover some performance aspects of the model, and Chapter 5 recommends several enhancements that would make it much more useful.

3.5 Chapter Summary

This chapter has presented the models developed in this thesis, providing some mathematical origins as well as a user-oriented discussion. The SS and RM models are high-fidelity models that model trajectory dynamics very accurately, especially when system LOS dynamics are very large. The analytical I and Q model is a faster, more feasible implementation of generating accumulated I and Q measurements. Both of these models are compared and analyzed in Chapter 4.

4. *Model Validation and Accuracy Analysis*

4.1 Overview

The very definition of the word "model" implies that something else is being represented. Models exist purposely to replicate in whole or in part a *real* object or idea. As such, their value is measured by their ability to faithfully represent the subject of their existence. This chapter provides a fundamental verification of the models developed in this research. It should be emphasized that any "new" model is rarely immediately validated, as it must be used and accepted by others over time. However, there are some tests that are done to assure basic functionality and technical competency. This chapter presents the results from several preliminary tests that were performed to determine the validity of the models developed in this research.

Real I and Q data recorded from the same Mitel chipset modeled here was provided by Tracking Systems and Imaging Inc. [52] and it is analyzed and compared with the outputs from the Signal Simulator. This data served as benchmark for determining typical receiver noise levels, which were then simulated for comparison. Comparisons are made between the output of Signal Simulator and Receiver Model and the output of the I and Q model. Mutual verification is achieved as these outputs are shown to be very similar. The generated pseudorange and carrier-phase measurements are compared with the truth environment, which shows the divergence of the LOS observables without signal tracking. Finally, other methods of verification are suggested that would increase the confidence in the models developed in this research.

4.2 Analysis of real data

Tracking Imaging and Systems Inc. [52] have a long history of GPS receiver research. The company's president, Dr. James Sennott, has patented a proprietary implementation of a DCOP algorithm [45], and has published many papers on his method, known as Integrated Demodulation Navigation, or IDN [40][41][42][43][44]. Their technology is implemented using the same Mitel chips modeled here, and they have the capability of data logging (storing) 1 kHz I and Q data output from the GP2021. They graciously provided a three minute long recording of I and Q samples, taken from the same Mitel chipset modeled in this research.

The I and Q data is the output of a single correlator, accumulated and dumped at ~1kHz. As discussed in Section 3.3, the GP2021 has two parallel correlators. The data provided by TISI was taken from the "prompt" correlator, with a code offset of ¼ chip. As a result, the signal magnitude in the I and Q samples is ¾'s of what it would be if the channel had been configured for "prompt" correlation. This must be taken into account when calculating the true received power—a 3/4 reduction in amplitude results in nearly half the power, or 3 dB [20].

At the start of the I and Q data file, no signal is present. For approximately two seconds, the I and Q values display only noise, averaging to zero. Figure 4-1 displays the I^2+Q^2 power calculation of the captured I and Q samples. As the signal is acquired, the detected I^2+Q^2 power jumps up, which is apparent in Figure 4-1. Notice the poor estimates of the S/N ratio, varying by roughly 5 dB. The quality of the S/N ratio estimate is directly related to the number of samples used to compute it.

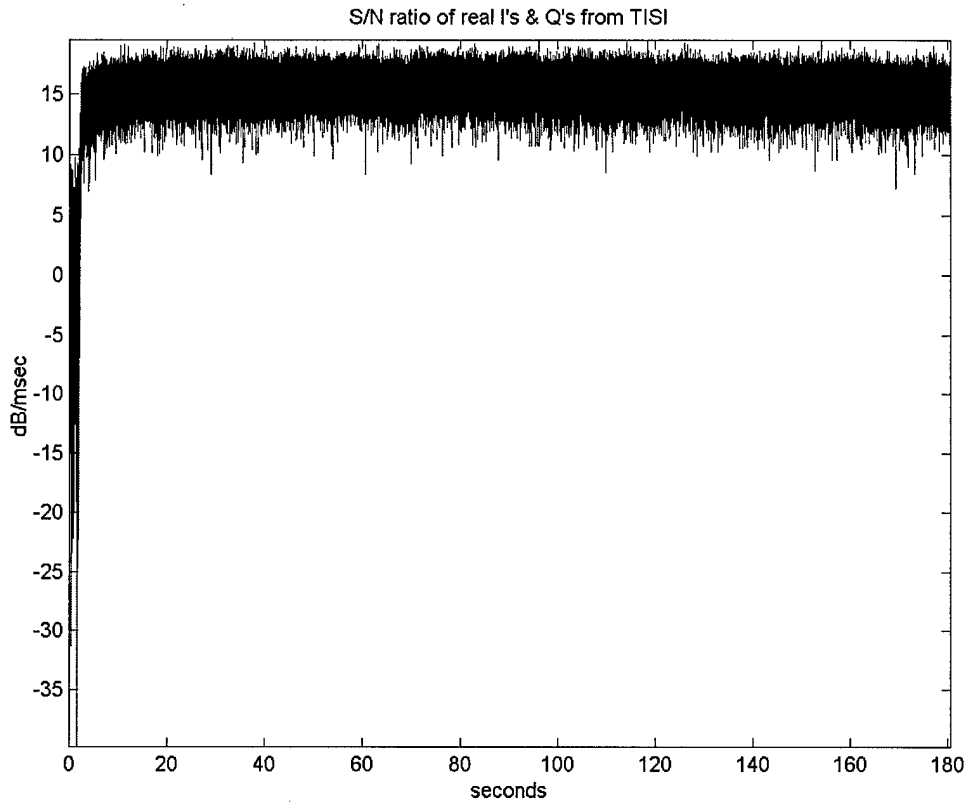


Figure 4-1. Calculated Power of real I and Q data

One of the benefits of signal simulation is the ability to turn off the data modulation. This luxury is not afforded to us with the real data, as data bit transitions are visible throughout. These transitions do not appear in the power plot due to the squaring process. The data rate limits how many samples can be accumulated to both increase the S/N ratio and estimate S/N. Without a priori information, only 20 milliseconds of data can be integrated without incurring an ambiguity at a data bit transition. Using the processing gain equation (2.3) in Section 2.2.3, the estimated pre-correlation S/N ratio can be calculated as follows:

$$\left(\frac{S}{N}\right)_{PRE-CORR.} = \left(\frac{S}{N}\right)_{POST-CORR.} - 10LOG_{10}\left(\frac{Int.time}{T_c}\right) \quad (4.1)$$

where:

Int. time = Time over which $S/N_{POST\text{-}CORR}$ was calculated
T_c = Time width of a C/A code chip.

The last term in (4.1) is equivalent to the processing gain for a given integration interval. Figure 4-2 shows the calculated S/N ratio, integrating over 20ms. The S/N is approximately 13 dB higher than that shown in Figure 4-1, due to the additional processing gain afforded by a longer integration time. The mean S/N ratio in Figure 4-2 is 28.2 dB. Due to the correlator spacing, the actual power of the signal that was recorded should be ~3 dB higher, or 31.2 dB.

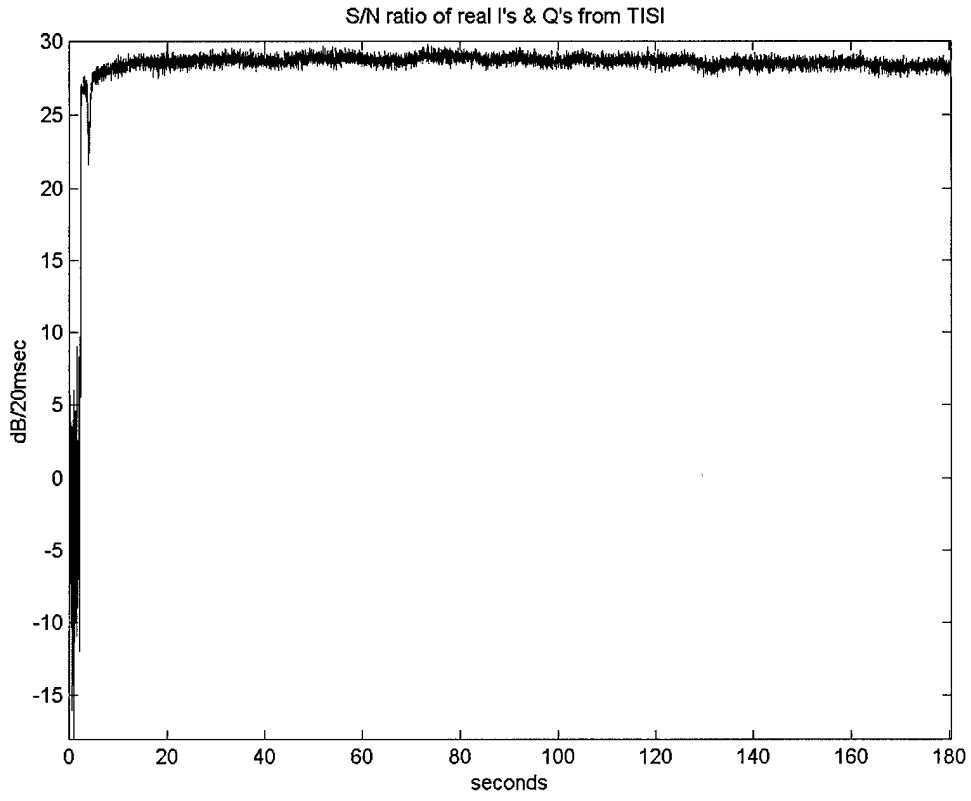


Figure 4-2. Calculated Power integrating over 20ms.

The receiver implemented a frequency tracking method, and did not do precise phase tracking, which makes comparing and displaying the data difficult. Additionally, the analysis is complicated since the phase of the I and Q samples is constantly rotating and the data transitions every twenty samples. Phase and Frequency discriminators were applied to the data, but the resulting plots exhibited little clarity and would not add any understanding to this presentation.

4.3 Noise floor validation

The derivation of the noise floor was given in Section 3.3.2. The receiver model must generate the expected noise floor accurately for the S/N estimation to be valid. The receiver model was fed a quantized signal of only noise, and the samples after cosine mixing were saved. The simulation used 100 ms of noise, which amounts to 571,428 samples. The variance of these samples was calculated and compared to the theoretical noise floor value of 8.5 (see section 3.3.2). The results indicated that the noise floor generated by the receiver model matched very well with the derived estimate. The comparison is shown in Table 4.1. The positive and negative numbers were combined because there was a maximum 0.0015 numerical difference between any positive and negative value for any given number. The samples are shown before and after code correlation, to show that in the absence of any signal these quantities have the same statistics. The reader is referred to section 3.3.2 for the theoretical derivation of the noise floor, and the origination of the component values in Table 4.1.

Table 4.1. Noise only signal statistics

Component	±6	±3	±2	±1	Mean	Var
Expected Values	**0.15**	**0.15**	**0.350**	**0.350**	**0**	**8.5**
I after Mixing	0.150055	0.150355	0.349947	0.349643	-0.002	8.505
Q after Mixing	0.150355	0.150055	0.349643	0.349947	-0.001	8.512
I after Corrleation	0.150055	0.150355	0.349947	0.349643	-0.002	8.505
Q after Correlation	0.150355	0.150055	0.349643	0.349947	-0.008	8.512

4.4 C-12 Flight Trajectory

All the data generated here used portions of a simulated trajectory based on an actual trajectory flown by a C-12 aircraft at Holloman AFB, NM. The trajectory was provided by Capt. John F. Raquet and previously used in coursework at AFIT [37]. The trajectory is ~28 minutes in length, and contains a full spectrum of dynamics. The trajectory begins with the aircraft stationary, after which it takes off and performs several box pattern landing approaches over the runway, eventually ending in some higher-dynamic maneuvers. The actual flight profile in any navigational frame is not useful for characterizing receiver dynamics. The true dynamic stresses on each channel occur in the line-of-sight (LOS) frame, where the dynamics of both the aircraft and the satellite are projected onto the signal propagation path. Accompanying this trajectory file was ephemeris information for eight satellites. The satellites were numbered 1-8, to lessen confusion in coding. The composite plot of LOS dynamics for all eight satellites over the entire 28 minutes of the trajectory file is shown in Figure 4-3. This plot is provided to give the reader an overall feel for LOS dynamics in this trajectory and the range of values for the LOS distances and velocities.

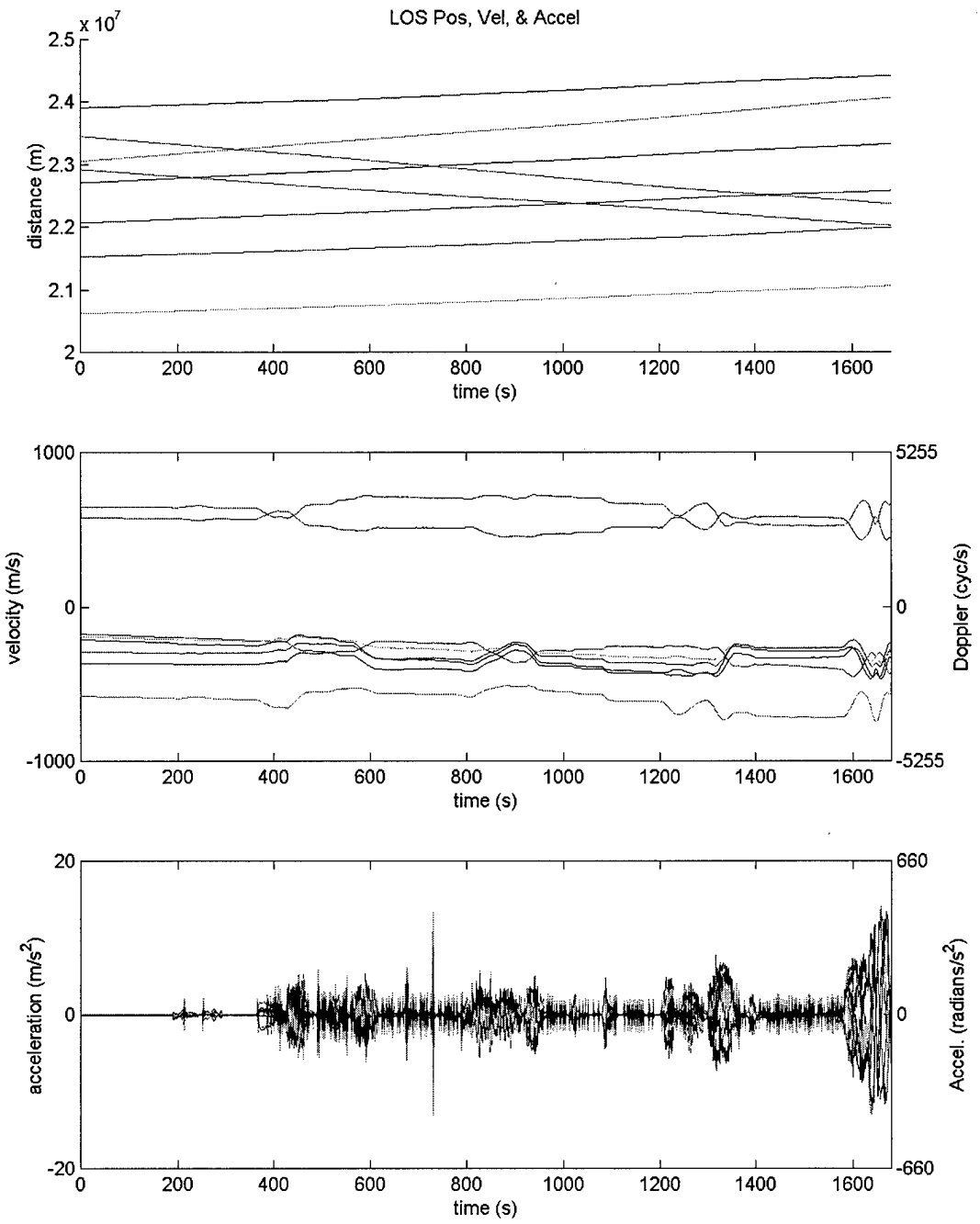


Figure 4-3. LOS dynamics for C-12 trajectory file

The trajectory was analyzed for its minimum and maximum dynamics. The maximum jerk (3rd order dynamic) experienced by any LOS path occurred on PRN 4 at around + 1657 seconds into the flight. A two second view of the LOS dynamics starting at this time is shown in Figure 4-4.

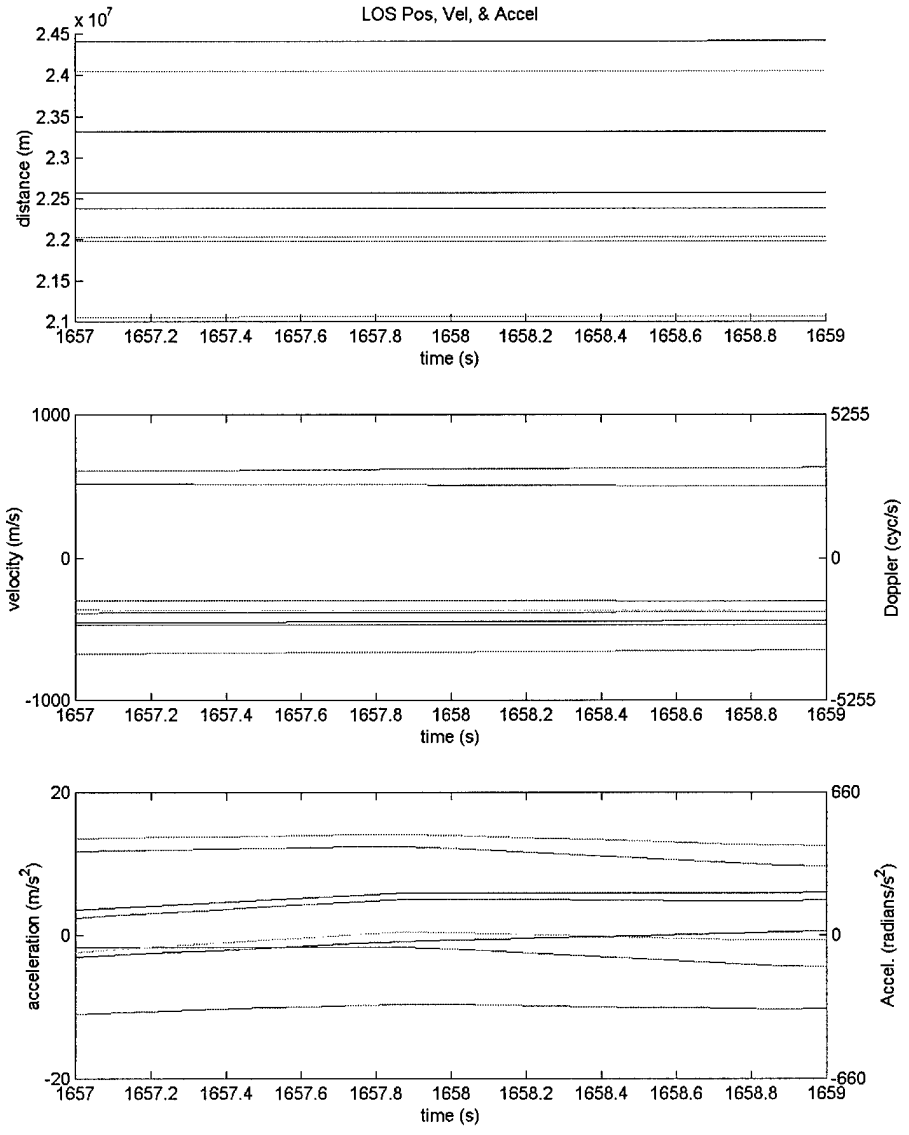


Figure 4-4. Maximum LOS dynamic environment

To analyze the low dynamic environment, any time prior to takeoff where the aircraft is stationary is sufficient, and an offset of 100 seconds from the start of the trajectory was chosen. The low dynamics in a two second time window are shown in Figure 4-5.

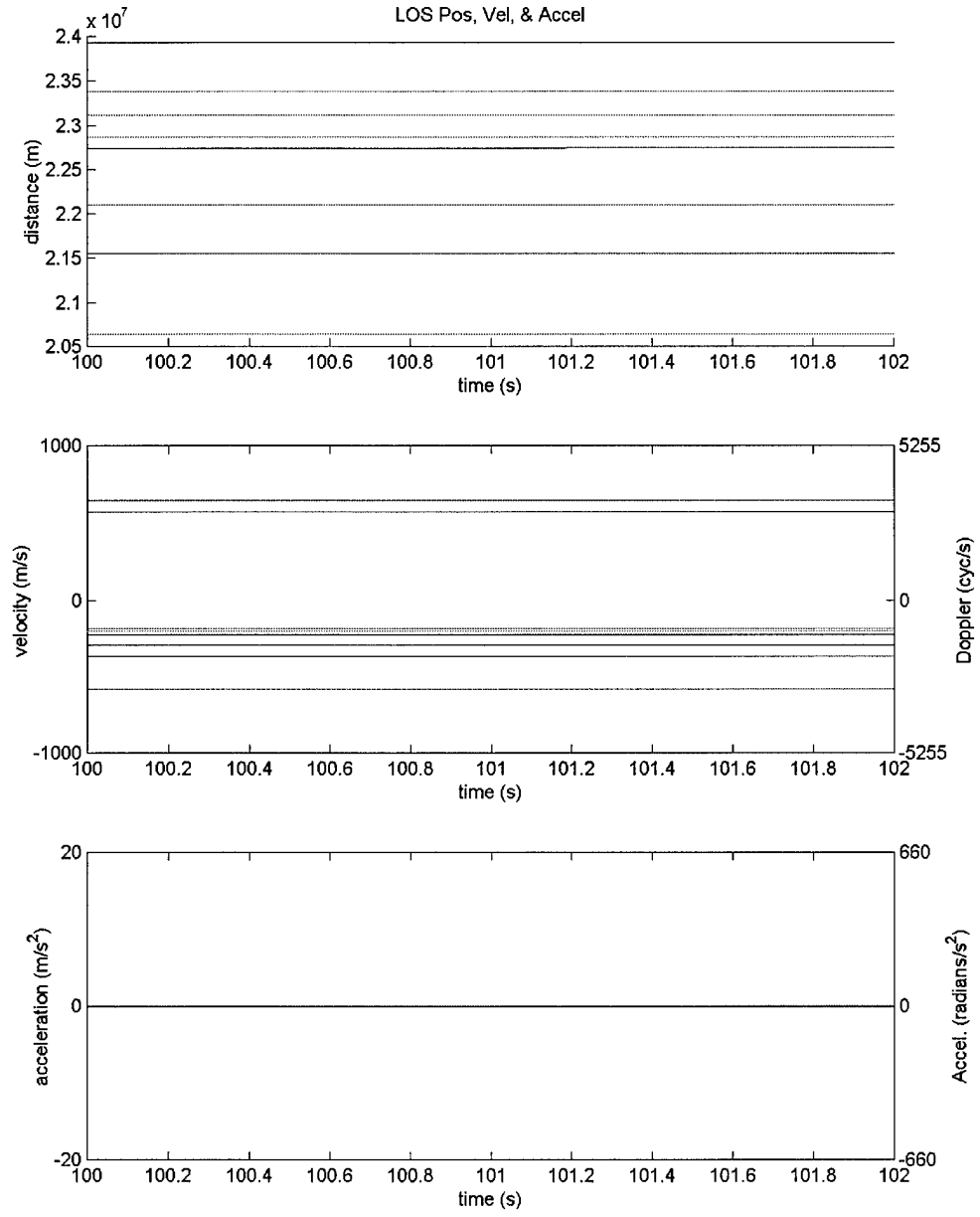


Figure 4-5. Minimum LOS dynamic environment

As PRN 4 is studied extensively in the results, a close up of just PRN 4's LOS dynamics is given in Figure 4-6.

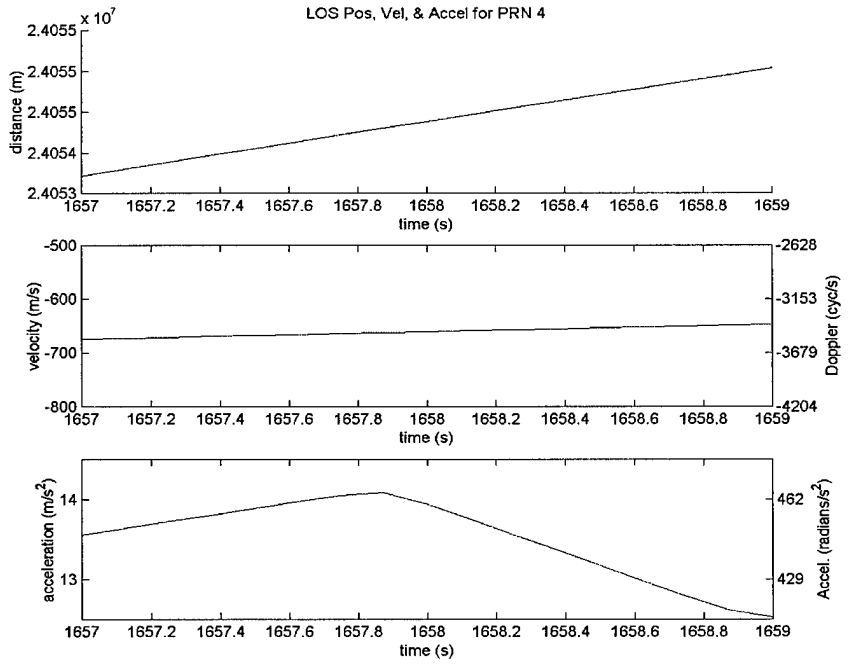


Figure 4-6. PRN 4 LOS dynamics

4.5 Simulation Parameters

To equate the output of the SS, RM, and IQMOD models, simulations were run to generate PR, CP, and I and Q values for two satellites in two different environments. PRN 4 was chosen to be the satellite used in the simulations. All simulations run were one second in length, generating accumulated I, Q, PR and CP measurements at a 1kHz rate. For the data simulated from the SS and RM, the SS generated 5.7 million samples per run, which reduced to the same final output amount as the I and Q model after processing by the RM. All the simulations were run with fixed DCO frequencies, and they perfectly synchronized to the truth phase and frequency at the beginning of each

simulation. These simulations included no clock errors or signal filtering because the IQ model does not have this capability.

The S/N ratio chosen for the run was based off the power of the samples provided by TISI. According to their correspondence and verified through calculation, the calculated S/N ratio of the data is ~ 15.5 dB/ms. This value only takes into account an integration equal to the code period of 1ms. At this integration rate, the processing gain is only 30 dB, compared with the 43 dB that can be achieved if the sample is integrated over 20 ms. This estimate is using Mitel's S/N formula, which generates an estimate 3dB higher than the true value of the signal (see section 3.3.2). However, the data provided is not the true prompt signal, but offset by ¼ chip. In practice, this ¼ chip code offset reduces the accumulated amplitude by ¾. This results in a loss of nearly half the power of a full correlation, given that power is proportional to the square of the amplitude. Therefore, the actual post-correlation power of the signal that was recorded is ~ 15.5 dB/ms. With the 30 dB of processing gain used to achieve this power, the signals below were simulated at a pre-correlation S/N ratio of –15 dB.

4.6 Pseudorange and Carrier-Phase measurement comparison

As a simple check, the PR and CP measurements generated by both the RM and the IQMOD were compared against the truth trajectories, to validate that the DCOs had been programmed correctly and to measure the errors generated from the Fixed DCO with the truth environment. In the plots that follow, the output generated from the Signal Simulator (SS) and Receiver Model (RM) is located in the top plot, whereas the bottom plot is the output of the I and Q model (IQMOD).

Figure 4-7 shows the phase error in the DCO over the course of the one second simulation. This error was calculated by subtracting the integrated carrier-phase measurements generated by the models from the true integrated carrier-phase measurements. The dynamics are appreciable, keeping in mind that the fixed DCO was perfectly synchronized at the beginning of the interval.

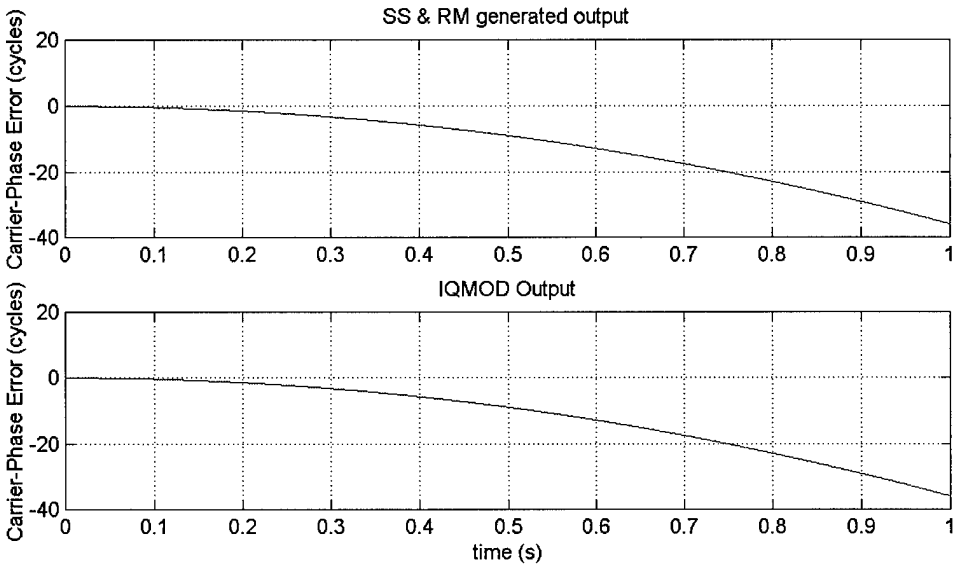


Figure 4-7. Comparing Carrier Phase errors for PRN 4 in high dynamics

These two plots should be *exactly* the same, as the DCO values are not changed over the interval and the generation of the truth LOS phase and frequency values is performed using essentially the same equations in both models. The above two plots, if subtracted, would yield a maximum error of 7×10^{-4} cycles over the entire second. Once again, this is just to show equivalency in the dynamic truth models and DCO error equations. For completeness, the pseudorange error plot is generated in the same manner and shown in Figure 4-8. The maximum difference between these two truth environments is slightly less than 6 mm.

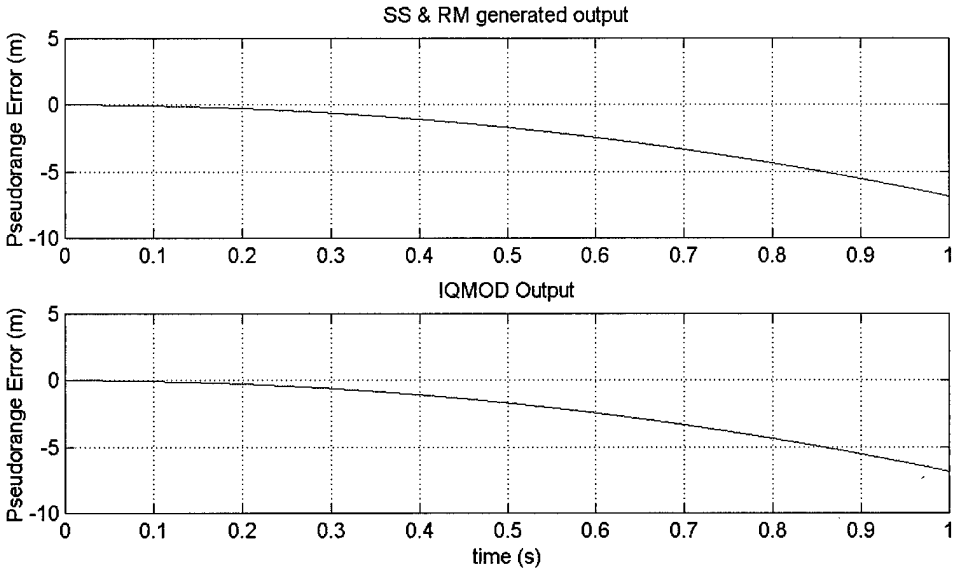


Figure 4-8. Comparing Pseudorange errors for PRN 4 in high dynamics

The frequency errors were generated in the same manner, by subtracting the DCO frequency from the LOS Doppler frequency of the truth environment. Either DCO could be subtracted (carrier or code), as for this example they were set to the same values. A comparison plot of the frequency errors is shown in Figure 4-9.

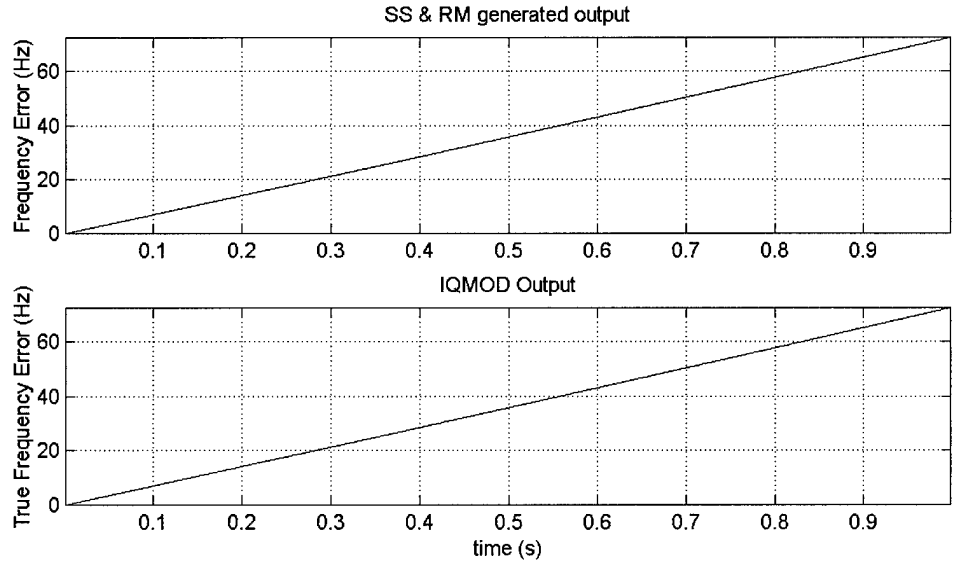


Figure 4-9. Comparison of frequency errors for PRN 4

Incurring approximately 70 Hz of frequency error after only one second is an indication of the effects dynamics can have on tracking errors. The frequency and phase errors were estimated from the I and Q data using a Costas 2-quadrent arctangent phase discriminator and a four-quadrant arctangent maximum likelihood frequency estimator [14]. These discriminators break down at ±90 degrees, and -±25 Hz, for the phase and frequency discriminators, respectively. The Phase estimation plot is shown in Figure 4-10.

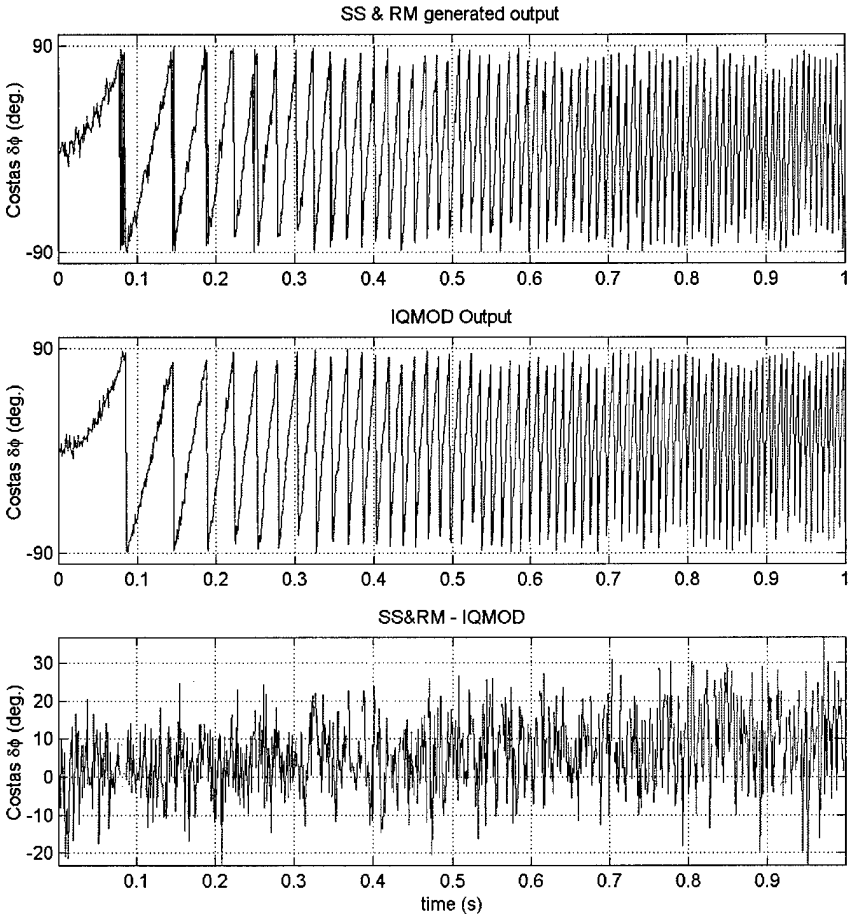


Figure 4-10. Costas phase discriminator output, applied to the I and Q samples.

The noises of the two models are independent, so a direct subtraction should yield a zero mean white noise, if the models were exactly equivalent. The third plot in Figure 4-10 shows this subtraction, with some points removed. The points which are not shown are those that occurred when one phase estimator experienced a 180 degree flip before the other. This results in points plotted at ~175-180 degrees, until the other phase estimator incurs its 180 degree flip. With these points removed, an increasing bias can be seen in the subtraction of the SS and RM estimated phase and the IQMOD estimated phase. This is due to the increased resolution of the SS and RM model over the IQMOD, which assumes a fixed Doppler offset over an integration interval.

The Frequency estimation plot is shown in Figure 4-11. It was mentioned previously that the discriminator breaks down at -±25 Hz, however the plot seems to indicate a break-down at 20 Hz instead. This is because the last sample to fall below 25 Hz was only slightly higher than 20 Hz, and the axis is set to the limits of the data. Some points were not plotted in the third plot of Figure 4-11 for the same reasons as before. The same trend in Figure 4-10 is not seen when subtracting the frequency estimates of the different models. This is due to the inherently less accurate nature of frequency measurement. The points that were generated in of Figure 4-11 were estimated from I and Q values accumulated over 20 ms, in order to reduce the noise of the measurements. With frequency estimation, the quality of the estimate is not nearly as high as the phase measurement, when compared in this context. Thus, in order to see visual correlation of the errors, the I and Q values were summed to generate better frequency error estimates. .This long integration time causes these estimators to work poorly in high dynamics, a classical trade off in GPS receiver design. [14].

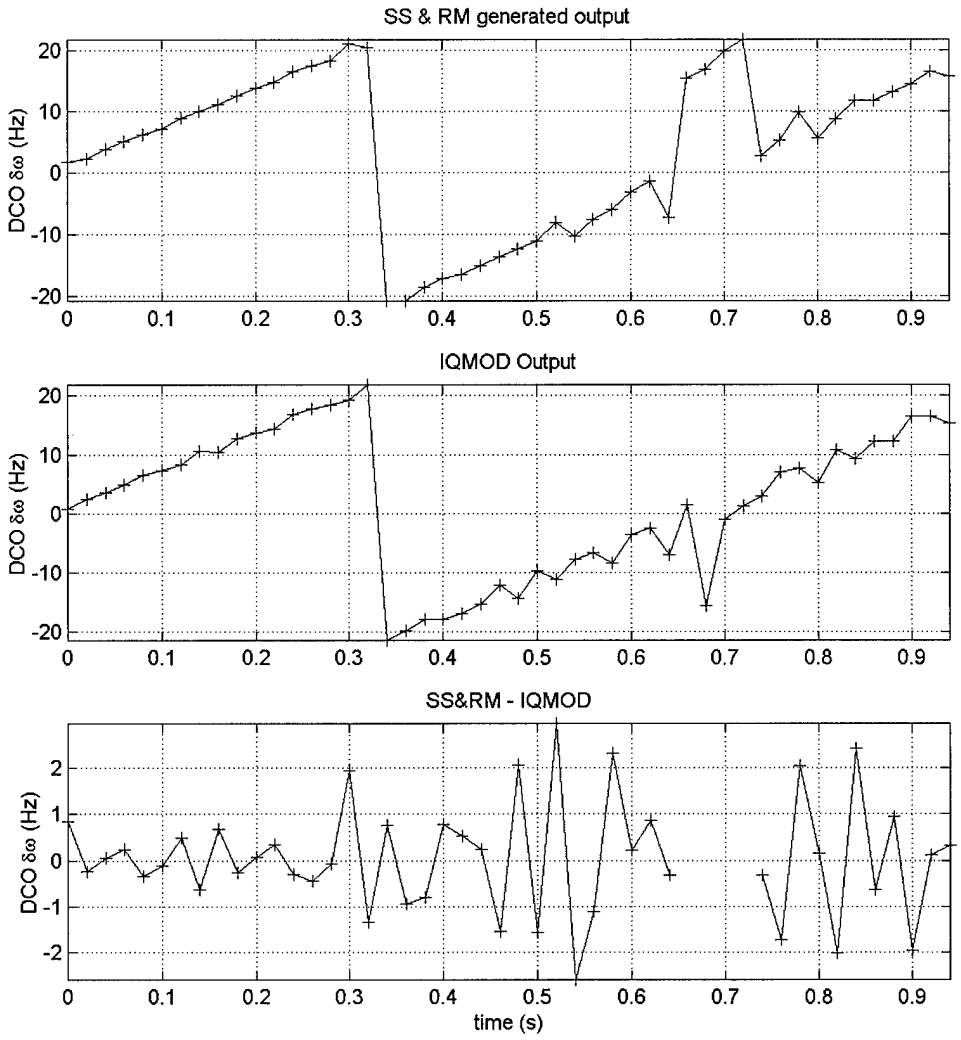


Figure 4-11. 4 Quadrant frequency discriminator applied to the I and Q samples

As the frequency offset gets very large, the frequency estimator starts breaking down, due to the ambiguity in its phase measurements.

In comparison with the true phase errors (Figure 4-7 and Figure 4-8), the estimators perform reasonably well. It can be seen already that the models match very closely with one another. The dynamics are so extreme here that detail is lost in the discriminators ability to report the errors towards the end of the one second simulation.

However, the phase discriminator does show a slight increase in modeling accuracy of the SS and RM over the IQMOD.

4.7 I & Q output comparison

The direct accumulated I and Q outputs from the prompt correlator are now compared. The Early and Late outputs were also recorded, but displaying them is redundant as they are linked to the prompt's reaction to the errors in the carrier DCOs. Over this one second, there are only 7 meters of Code offset, not enough to notice any difference of the early and late values, especially given that they were perfectly synchronized at the beginning of the simulation.

Figure 4-12 shows the I and Q outputs from both the SS/RM and the IQMOD.

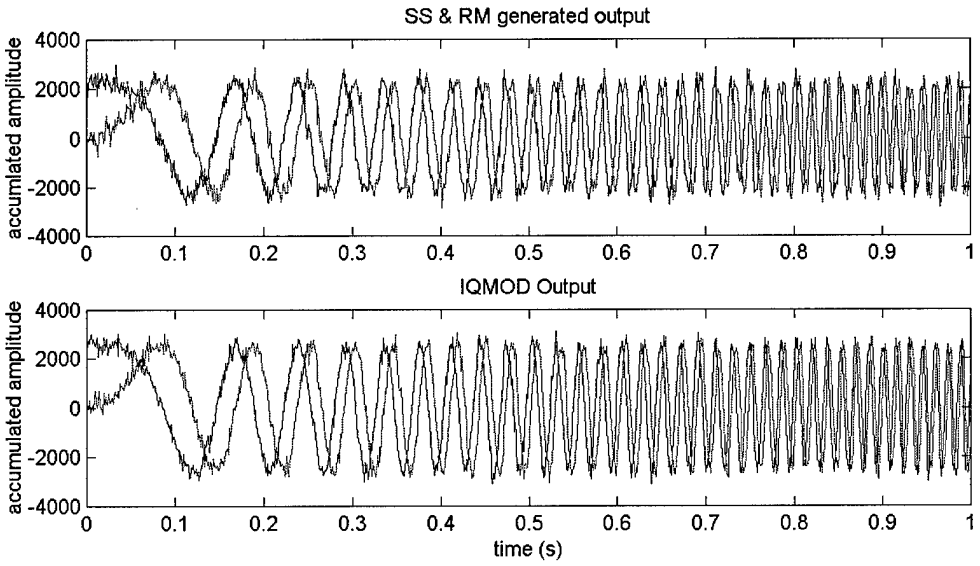


Figure 4-12. I and Q output for PRN 4 under high dynamics.

If they were not before, the dynamics of the system are readily apparent here. The fixed DCO values quickly cause the signal to loose phase lock. The zero frequency offset a the beginning of the run has grown into a ~70 Hz offset after one second. This is

not due to a high Doppler value, but to a high *rate of change* in Doppler, which can be seen from Figure 4-6. The Q value, (starting at zero), goes positive first, this indicates that the phase error is positive, and that the Doppler frequency is increasing. However, due to the aliasing in the sampling process, this is not the case. A quick look at Figure 4-6 reveals that the Doppler frequency is actually decreasing. This must be taken into account when applying feedback to this system. If the aliasing phase and Doppler reversal is ignored, the tracking loop will become unstable.

The effects of the frequency error are readily apparent in the integration of the samples, shown in Figure 4-13. Here the samples are accumulated over a data bit width of 20 ms, and then used to calculate signal power.

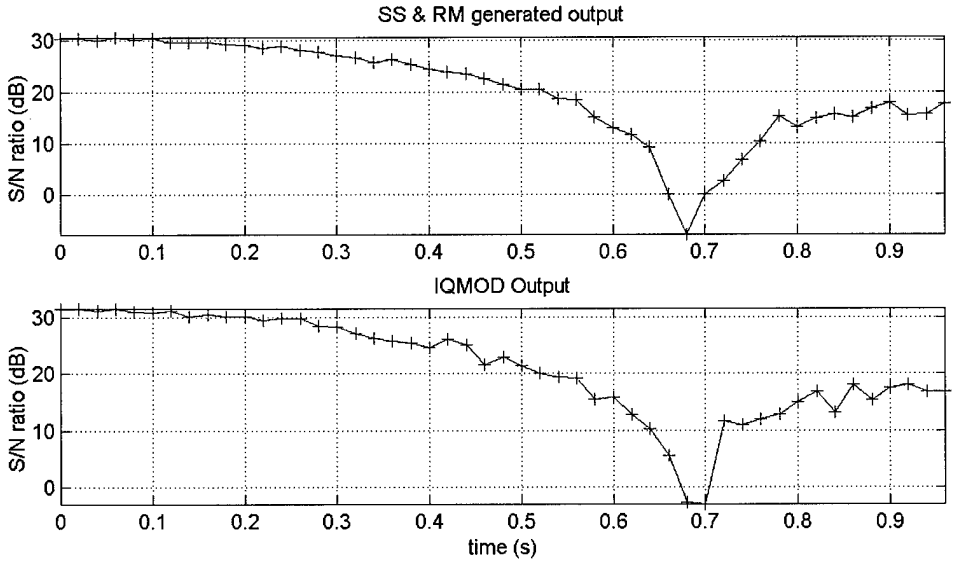


Figure 4-13. Comparison of accumulated S/N ratio, 20 ms integration time

As the frequency error increases, the integrated samples decrease, due to the rotation of the I and Q values being integrated. As the frequency increases, the vector sum of each I and Q value starts to converge back to zero. Essentially, the beat frequency

caused by the mismatch of the received frequency with the DCO frequency is integrated over a period of 20 ms. At approximately 0.7 seconds in Figure 4-13 the beat frequency reaches 50 hertz, at which point the samples integrate over a complete cosine wave, which averages to zero. As the beat frequency then increases past 50 Hz, the integration does not average to exactly zero and thus the signal estimate comes back up. This is the reason to maintain a zero frequency offset, so that signal power is not lost during accumulation.

To appreciate the impact of the dynamics seen above, the same plot is shown Figure 4-14 for the low dynamic case in. Here the setup is the same, but the aircraft is sitting stationary on the runway during this second. Here, the DCO values are fixed for the same amount of time with little more than 60 degrees phase tracking error. One can see that again the IQMOD output follows the SS & RM generated output.

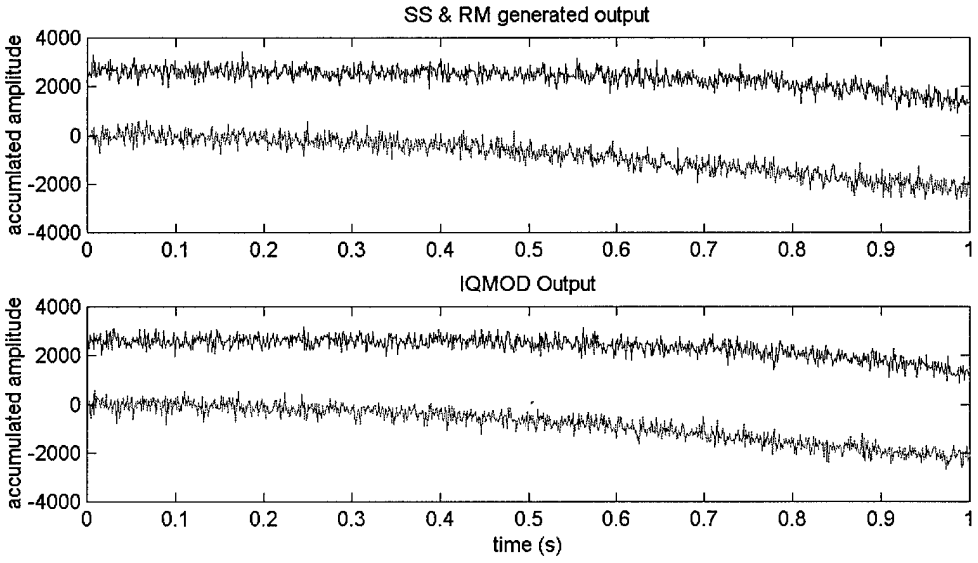


Figure 4-14. I and Q output for PRN 4 under low dynamics.

4.8 Statistical Analysis

The simulation with limited dynamics, illustrated in Figure 4-14, provides a good opportunity for statistical analysis. The 1 second simulations generated 6,000 I and Q measurements—1000 for each of the correlator channels—early, prompt, and late I and Q. To validate the equation based model, it is valid to compare the statistical mean and variance of the samples between models. It is also productive to look at other averages, such as S/N ratio, relative magnitudes between the early prompt and late samples, and the expected processing gains.

Phase locked I and Q values are necessary for calculating averages and statistics on the I and Q measurements. If the mean value of either the I or the Q changes, the validity of a mean and variance calculation drops. Therefore, only the first 300 milliseconds of the I and Q values in the low-dynamics simulation was used in the statistical analysis. The means and variances on the I and Q values are given in Table 4.2.

Table 4.2. I and Q statistics

	Theoretical:	**SS & RM PRN 4**	**IQMOD PRN 4**	**SS & RM PRN8**	**IQMOD PRN 8**
I Early Mean	1304.61	1296.02	1210.31	1292.95	1304.74
Q Early Mean	0.00	-28.63	-78.77	-38.01	-81.31
I Prompt Mean	2609.22	2615.01	2601.41	2618.76	2598.30
Q Prompt Mean.	0.00	-99.11	-146.43	-85.44	-140.24
I Late Mean	1304.61	1318.75	1226.96	1287.11	1309.62
Q Late Mean	0.00	-43.47	-66.66	-47.85	-50.29
I Early Var.	220.40	246.16	243.71	216.38	236.90
Q Early Var.	220.40	244.18	231.34	221.49	233.66
I Prompt Var.	220.40	236.00	239.26	221.55	239.20
Q Prompt Var.	220.40	261.95	278.09	252.28	236.49
I Late Var.	220.40	241.17	249.73	215.26	239.56
Q Late Var.	220.40	241.22	255.54	222.53	241.00

The theoretical column in Table 4.2 was calculated with the same equations used in the I and Q model. The slight positive bias in the variance of all the measurements is because although the first 300 samples were relatively unaffected by the slowly growing phase error, they do experience some of the effects. To prevent this, the DCO values should be continually updated so accurate statistics can be taken. The mean I values of both models are very close to the theoretical estimate, as well as each other. Some evidence of signal phase error is evident in the Q values, as they are all consistently negative and exhibit a slight bias. From Figure 4-14 we can see the signal start to migrate negatively, which support the conclusion on the negative Q values. Overall, however, there is good correlation between the models and theoretical values.

The data in the last table was averaged over the all the samples. The S/N ratios match almost perfectly. The rows containing the W and W/O are variables calculated inside the program, and differentiate between signal power with and without the other signals present. This gives us ability to directly measure the effects of the other signals in the system, especially useful for pseudolite research.

Table 4.3. S/N ratio Statistics.

	SS & RM PRN 4	IQMOD PRN 4	SS & RM PRN8	IQMOD PRN 8
S/N setpoint	-15.00	-15.00	-15.00	-15.00
W/O cross-corr.	-15.08	N/A	-15.08	N/A
W/ cross-corr.	-15.34	N/A	-15.34	N/A
Mean 1ms S/N	15.45345	15.46	15.44852	15.44852
Mean 20ms S/N	28.46185	28.48	28.46445	28.46445
Processing Gain	13.00839	13.01314	13.01593	13.01593

4.9 Summary

This chapter has demonstrated basic functionality and the fundamentals of the models developed in this research. Chapter 5 presents a summary of the work accomplished for this research and a list of recommendations for further validation and application of these models.

5. *Conclusions and Recommendations*

5.1 Conclusions

The development and initial testing of three contributions in GPS system and receiver modeling has been completed. The developed models are rare, if not unique, in their capabilities and adaptability. The purpose of this research was to design and validate GPS signal and receiver models that are accurate in all areas pertinent to comparing or evaluating new receiver technologies. It is recognized that validation is not an immediate process. Any new product must be used and tested by the community in order to be trusted. However, initial model validation is performed, and these results indicate a high level of modeling accuracy.

Three individual models were developed, that provide a new simulation and testing capability to Air Force Research Laboratory and the Air Force Institute of Technology. These models are (1) A GPS L1 C/A code Intermediate Frequency (IF) signal simulator (SS), (2) a realistic GPS receiver hardware model (RM), and (3) a fast, analytical, accumulated I and Q signal/receiver model (IQMOD). The result fills a void in the GPS industry identified by Tetwsky and Soltz of the Draper laboratory [50]. This chapter discusses the impact of these models first individually, then as a whole. Some potential applications of these models are suggested, and recommendations are given for furthering this research.

5.1.1 *GPS IF Signal Simulator*

The main strength of this model its ability to accurately generate a spectrum of downconverted GPS signals and noise that are correlated with an input trajectory and in phase with true GPS system time. The output of this model is completely representative of what is seen at the output of real hardware, and with a few modest enhancements should be capable of generating a signal that can be processed by real hardware. Many receiver effects are modeled, although many GPS system error sources are not. Receiver clock bias and drift are simulated, although the models for the errors themselves have not been verified. The effects of the errors have been properly handled, however, so in the event that error models are replaced with more realistic or proven models, their effects will be accurately portrayed.

Although many software simulators are available that generate pseudorange and carrier-phase measurements from receiver trajectories and satellite ephemeris, existing receiver tracking loop models capable of accepting these same inputs were not found in the literature review. The SS model developed here is unique in that is uses receiver trajectories and satellite ephemeris as inputs to simulate an accurate GPS signal spectrum at a sampled IF. The capability of generating sampled GPS signals at a down-converted IF from a receiver trajectory and satellite ephemeris was not found in the literature review. As far as is known, the contributions made by this software are original to the field of GPS simulation.

5.1.2 *Applications of the GPS IF Signal Simulator*

The output of the Signal Simulator is typically a quantized bit stream of one's and three's, which represents the spectrum of un-processed GPS signals and noise. Without sufficient receiver processing, this bit stream provides little, if any, information prior to processing. Some devices on the market today generate such sequences only to be transmitted to a central location. NavSys Corporation makes such a product, based on the same GPS RF front-end (GP2010) used in this research. The device is known as a TIDGIT® sensor, and the location system which uses it is called TRACKTAG. This simulator faithfully models the sampled spectrum that TIDGIT® sends to TRACKTAG.

The general application of this model is to simulate the GPS signals adequately for input by receiver simulators, such as the RM, and potentially for real hardware as well. As GPS RF simulators are very costly, it would be advantageous to test the digital section of a receiver using a software signal model, instead of using a GPS RF simulator and the RF front-end of a receiver to generate the digital samples. The SS generates a digitized signal directly, which can then be imported into actual hardware. In the case of the Mitel chipset, a 20 second signal would consist of 114 million samples, each of which only require two digital bits to store. This signal, as well as longer ones, could potentially be stored much more efficiently than what is done in the Matlab® "MAT" file format (the Matlab® .MAT file requires almost a gigabyte of storage space for this 20 second signal). It is expected that few enhancements to the SS would be necessary, depending on the desired application, for the SS output to be used in real hardware. The minimum enhancement is the insertion of an actual GPS data message into the software, replacing the currently randomly generated data bits. This data sequence is necessary for

proper synchronization in the receiver. The only component necessary to enable this capability is the data message itself, as the data modulation has been fully implemented and synchronized to GPS system time. Using real hardware to process the signal generated here would be the ultimate verification of this signal simulator, and would provide the ability to tweak the simulator to maximum accuracy.

5.1.3 Recommendations for further development of the GPS IF Signal Simulator

In order for real hardware to calculate a position from the simulated signal, some of the GPS system errors would need to be modeled in the software. While the absence of most of these errors would only increase the accuracy of the position calculated, some of the errors that are anticipated and accounted for would need to be implemented. A principal example of this type of error would be ionospheric error. A ionospheric model is nearly always applied, which reduces the error considerably. Modeling the GPS errors at this stage is a slightly different process than that modeling performed at the pseudorange level. Some models are easier and more accurately implemented, such as a multipath model. Multipath errors can be accurately simulated by including the simulated signal multiple times, with the multiple copies delayed and reduced in power. This type of error is difficult to model in the pseudorange (PR) and carrier-phase (CP) measurements [6]. In fact, the inclusion of multipath in this model could lead to advances in characterizing multipath errors in PR and CP measurements, for a variety of different receiver topologies. Other errors are not as easily modeled. The ability to characterize errors at the high sampling rates of the SS gives rise to a much more detailed model, and implementing simple models here can fail to recreate the characteristics of the

error after the samples are processed and accumulated. High fidelity error models would likely have to be developed and tested to ensure their validity.

A more complicated, but nonetheless valid approach to verifying the SS model would be through the use of RF satellite simulator. A highly accurate and configurable RF simulator could be fed the same environment and control parameters as the SS, which could be fed to a receiver using the Mitel chipset. Some comparisons would be valid here. Using this approach, the noise corrupting the real signals is not reproducible by the SS. Only after the true and simulated signals were processed could their results be adequately compared. If the output of the RM were to be compared to the output of the real hardware, the comparison would be limited by the effects of different noises on the tracking process. In addition, the receiver model would need to be expanded and the effects of the various timing signals used in the real hardware would need to be implemented. As the purpose of the RM was initially to dodge this level of hardware modeling, this approach to verification is not recommended. A better approach would be to data log the digitized samples (@ 5.7 MHz) out of the actual GP2010 (or GP2015), and run the same signal into the RM as is seen by the actual GP2021. A comparison of the I and Q values could then be made, if one was very careful to initialize both the RM and the GP2021 to exactly the same values. There are many quantities here to synchronize, requiring additional modeling in the RM as well. The DCO settings, dumping rates, timing signals, correlator configuration, and DCO adjustments would have to occur at precisely the same times during the simulation as in the real world in order to directly compare the accumulated I and Q samples. In this approach, the real hardware would process the signal first, all the while recording and storing all DCO

values, their corrections and timestamps, I and Q output and the dump times at which the I and Q data was latched. With this and undoubtedly other information about the chip as it processed the data, the Receiver model could theoretically generate the same I and Q values as the real hardware.

There are other enhancements to the SS that would make it very valuable and extendable into other applications. These include the ability to add and drop satellites during a run, vary the S/N of the signals individually and/or collectively over the course of a run, and developing an satellite visibility model as a function of the input trajectory and antenna mounting to simulate body frame masking. As the largest crystal oscillators are deterministic and g-force induced, the development of a g-sensitive clock model would more accurately reflect the performance of a receiver in high dynamics.

Some fundamental programming changes need to be made in order to generate longer signals. The current implementation requires the machine to have as much RAM as the final size of the signal being generated. This is due to the complicated nature of incrementally saving variables to disk in Matlab®. Low-level file writing commands are available in Matlab®, and could be used to overcome this limitation. Such an enhancement would be necessary to generate signals long enough to make a real hardware test practical.

A highly desirable enhancement to the software would be the development of a Graphical User Interface (GUI) for controlling the parameters of the simulation. Currently, the load_global_params.m file must be open in an editor and modified to control the parameters of the simulation. A GUI is easily developed in Matlab®, and would enhance the utility and lower the learning curve of the SS. This GUI should also

include the RM parameters, or a separate GUI should be developed for the receiver model as well.

5.2 GPS Digital Signal Processing Receiver Model

The literature review for this thesis suggests that the 12-channel C/A code RM is unique in its capacity to process GPS signals simultaneously; other products that were found with similar capabilities only consist of a single channel (see section 1.4.2). Simultaneous demodulation is necessary for some advanced receiver designs, such as DCOP receiver architectures. Additional capabilities of the model are highlighted here; the combined performance of the SS and RM model is given in subsequent sections.

The RM performs the high-frequency (>4MHz) baseband digital signal processing in a GPS receiver. These processes are typically very similar among receiver designs, and are usually implemented in custom digital circuitry. The RM is designed to allow the user to customize the portions of the design that are likely to vary from one design to the next. For this research, the RM was configured to model the Mitel GP2021 chip. Only the signal processing section shown in Figure 5-1 is faithfully modeled. The RM design focus was to accurately duplicate the signal processing functions, while minimizing the level of digital complexity encountered in an actual hardware chip. For the purposes of comparing fundamental design topologies, the in-depth digital operation is not necessary and often hinders the process. The Simulink® modeling environment allowed this goal to be met.

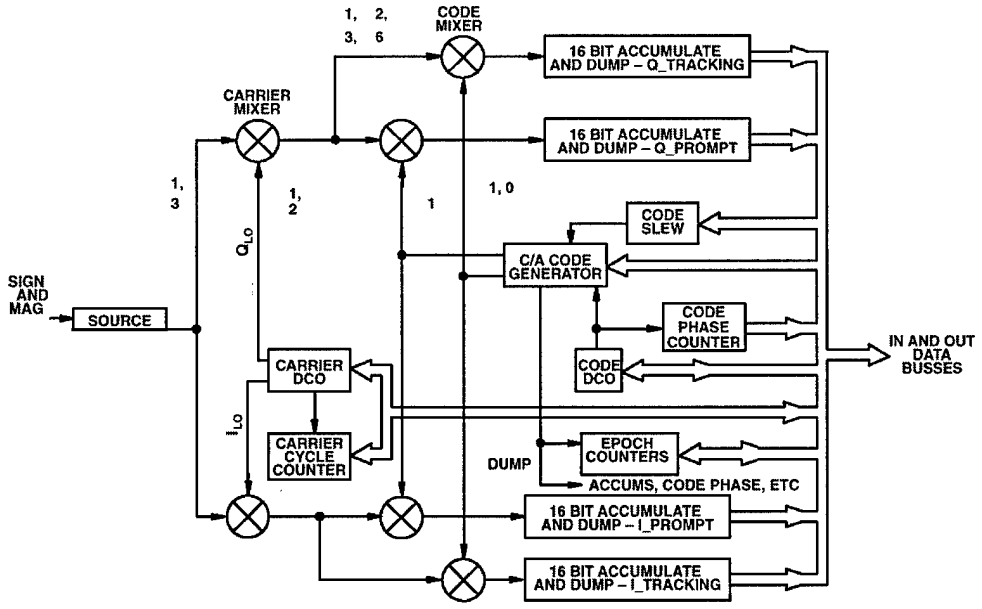


Figure 5-1. GP2021 tracking module block diagram [23]

Having an easy-to-use graphical interface to a detailed Receiver model is a major benefit of the RM. As recommended by Tetwsky and Soltz [50], Simulink® was used to model the GPS receiver model. Simulink® offers a good balance of user-friendliness and modeling flexibility. Simulink® allows the user to quickly see and understand the functional hierarchy and layout of a GPS receiver. While this document does not attempt to teach the operation of Simulink®, it is reasonable to assume that the learning curve is not prohibitively large. Working in Simulink® is comparable to dragging, dropping, and connecting block diagrams. There is no programming language to learn with Simulink®.

The RM is designed for closed loop operation, and is readily controllable by any feedback loop implemented in Simulink®. The tool is designed to save nearly every signal of interest, at up to the full sample rate, to be used for analysis and/or debugging of a simulation. Simulink® also allows a user to view signals as the simulation runs, which gives the user an animated view of the system response at a reduced time scale.

5.2.1 *Applications of the GPS Receiver Model*

The receiver model can only perform processing of an externally generated signal. The individual contribution of the RM, without any signals to process, is very limited. The verification that the processing is done correctly requires processing signals with known characteristics. Thus, the validation of the SS and RM must occur simultaneously, as neither the output of the SS nor the processing abilities of the RM are verifiable without using them together.

Signals for the RM are not limited to those generated by the SS. The RM could be used to process actual samples recorded out of the Mitel GP2010 or GP2015 front end. As the signals are pulled out of the composite signal through a time-frequency search process, the receiver could be configured to generate a set of pseudoranges and carrier-phase measurements for calculating position. The samples would need to be time tagged in order for this to occur. This again represents an application using the TRACKTAG system from NavSys, [29]. The RM could be experimentally used as the sample processing section of the system.

5.2.2 *Recommendations for further development of the GPS Receiver Model*

As mentioned in the recommendation section for the SS, a GUI interface would be very helpful for simplifying simulation control. The GUI interface would be additionally useful if it controlled the signal selection done by the bus selector blocks (see Figure 3-34). To speed up the simulation, some of the functions could be implemented in MEX files. MEX files are used in Matlab® and Simulink® to speed up processes that run slowly in either Matlab® or Simulink®. The files are of a specific format and are

complied to run faster than the pseudocode in Matlab® or user defined blocks in Simulink®.

5.3 Analytical I and Q signal/receiver model

The analytical I and Q model is an important contribution in the area of ultra-tight INS/GPS integration and DCOP system modeling. In fact, this model has already been used to generate I and Q values in simulations of an ultra-tight INS/GPS system performed by a fellow thesis student [35]. The analytical I and Q model was validated in Chapter 4 against the output of the SS and the RM, which performed simulated signal generation and processing. In both low and high dynamics the I and Q model generated I and Q values undistinguishable from those generated by the SS and RM. Direct mathematical comparison of the I and Q values is not performed due to the relatively high noise ambiguity of the samples. The visual comparisons of the two outputs in Chapter 4 provide suitable validation for the scope of this thesis. The equations for estimating the amplitude and noise variance were also validated against the SS and RM in Table 4.3.

The I and Q model generates I, Q , PR, and CP measurements. It simulates a 12 channel C/A code single frequency receiver at an arbitrary final IF frequency. The model can operate in environments were the update rate is not fixed, and the satellites used and their channel assignments can be changed at any function call.

5.3.1 Applications of the Analytical I and Q signal/receiver model

The I and Q model has direct application in the simulation of both DCOP and ultra-tight GPS/INS navigation systems. It consists of only a few Matlab® files, and has a

logical command line interface. It can be used in any Matlab® simulation of these types of systems. It can also be used for any high-level GPS receiver simulation, such as those done by Ward to determine tracing thresholds under jamming [57]. In fact, this model provides a multiple channel extension to the stated capabilities of the model used in his simulations, described in [58].

5.3.2 *Recommendations for further development of the I and Q model*

The I and Q model currently does not employ any GPS error models, system or otherwise. It is recommended that atmospheric error models be added, as well as receiver clock errors. Due to the high G-force environments that DCOP and ultra tight architectures are well suited for, the inclusion of a g-sensitive clock model is also required. Additionally, as trajectories of aircraft are simulated, body masking of the antenna should be modeled to accurately simulate the blockage of satellites as the aircraft maneuvers. Signal attenuation effects could also be added here, and in general the ability to vary the S/N ratio of the received signals during a run should also be added.

5.4 Performance and Accuracy analysis of all three models

The signal generation and processing models and their operation are inherently different from the I and Q modeling done in the analytical model. These models are linked only in that they should generate the same quantities. The SS and RM models have to do with IF signal generation and receiver processing functions, while the analytical I and Q model is based on deriving the theoretical effects of the processes implemented in the SS and RM. Thus, a comparison of their outputs provides a first level validity check in the modeling. This is the level of validation performed in this thesis.

More robust and valuable validation techniques were known, and they are listed here rather than performed due to the time constraints of this work.

The output of the RM model, fed by the SS model, was compared with the output of the I and Q model with identical input parameters. The results were impressive, the RM behaved nearly exactly as predicted by the equations. Phase and frequency discriminators were applied to the output I and Q values, which accurately estimated the true phase and frequency errors in the system (compared to the known errors). The results indicated that the extra modeling accuracy afforded by in-depth modeling of the signal creation and processing was noticeable from comparing the phase discriminator outputs of the two models. As this simulation was limited to the LOS dynamics of a commercial C-12 aircraft, and limited to a length of one second, there are some modeling applications where the high-fidelity dynamic modeling in the SS is needed. However, the I and Q model is a much more practical model due to its size, speed and portability. It was previously mentioned that the accuracy of the model could be increased by reducing the integration time parameter, allowing the assumption of a fixed Doppler frequency over the integration interval to have less of an impact on the modeling. Therefore, most applications will benefit from using the analytical I and Q model over the SS and RM, due the massive differences in processing and memory requirements between the models. However, the analytical I and Q model would be inherently less accurate at generating I and Q values perturbed by some GPS error sources such as multipath.

5.5 Final Remarks:

The results of this research are recognized as the models themselves, not only the results of the initial validation process. It is known that validation is key for these models to be accepted, but it is also known that only a very in-depth analysis of these models, outside the scope of this thesis, could possibly convince the community outright that they are, in fact, accurate. Therefore, the validation process is just beginning. It starts with the first person to use these models and never truly ends.

The work done was therefore focused on developing the capabilities and usefulness of the models to the community that identified their absence. If the models are not useful or easy to use, their validity does not matter. These models are the only ones known to exist that generate multiple channels simultaneously, and are fed by receiver trajectory and satellite positions. This makes them prime candidates for doing DCOP or ultra-tight GPS/INS integration research.

In closing, it should also be mentioned that the educational utility of these models is potentially tremendous. The ability of Simulink® to display the results of the simulation as it runs, combined with the ability to deploy the Simulink® models incrementally gives rise to the use of these models in teaching receiver design and spread spectrum principles.

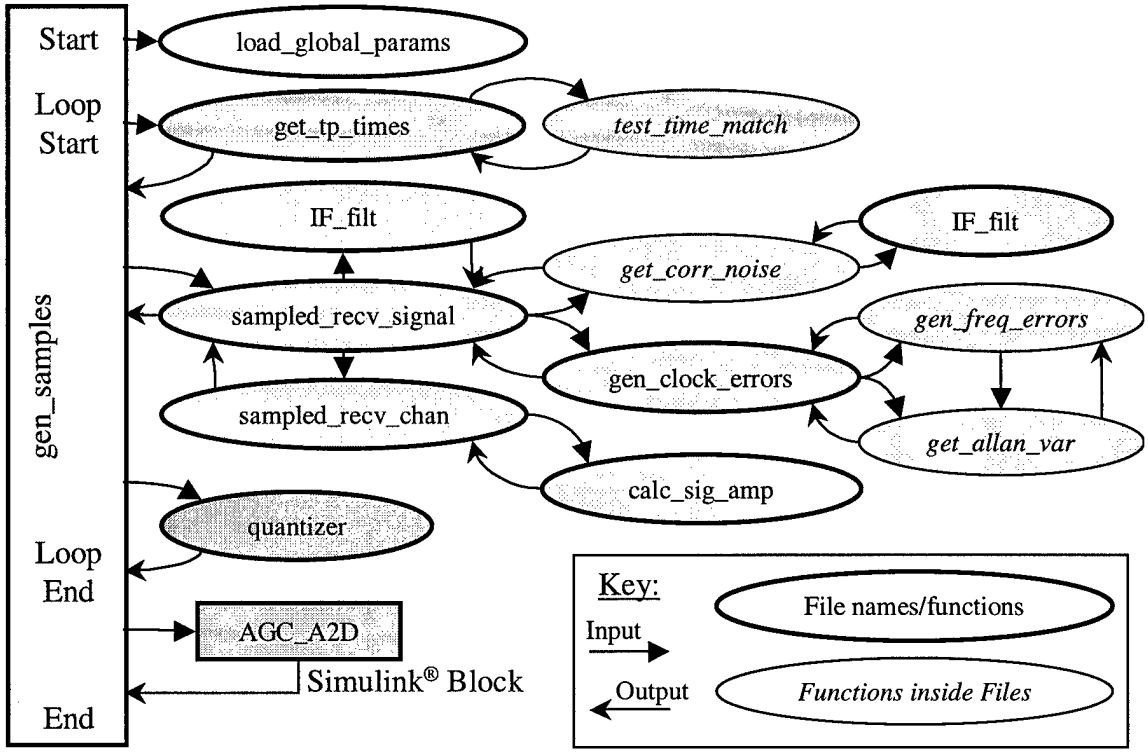
APPENDIX A Acronym List

AFIT	Air Force Institute of Technology
AGC	Automatic Gain Control
BPSK	Binary Phase Shift Keying
CDMA	Code Division Multiple Access
COTS	Commercial-Off-The-Shelf
CP	Carrier-Phase
DCO	Digitally Controlled Oscillator
DCOP	Direct Correlator Output Processing
DGPS	Differential Global Positioning System
DLL	Delay Locked Loop
DMSO	Defense Modeling and Simulation Office
DOD	Department of Defense
DSP	Digital Signal Processing
DSSS	Direct Sequence Spread Spectrum
EKF	Extended Kalman Filter
GPS	Global Positioning System
GUI	Graphical User Interface
I	In-phase
IF	Intermediate Frequency
IMUs	Inertial Measurement Units
INS	Inertial Navigation System
LO	Local Oscillator
LOS	Line of Sight
MEX	Compiled MATLAB function
MSIAC	Modeling and Simulation Information Analysis Center
PN	Pseudonoise
PPS	Precise Postioning Service
PR	Pseudoranges
PRN	Pseudorandom Noise
PSD	Power Spectral Density
PSK	Phase Shift Keying
Q	Quadra-phase
QPSK	Quaternary Phase Shift Keying
R&D	Research and Development
RF	Radio Frequency
SA	Selective Avalilability
SBA	Simulation Based Acquistion
SPS	Standard Positioning Service
SV	Satellite Vehicle
TDL	Tau-Dither tracking Loop
FFT	Fast Fourier Transform
UTC	Universal Coordinated Time

DOP	Dilution of Precision
HDOP	Horizontal Dilution of Precision
VDOP	Vertical Dilution of Precision
UERE	Equivalent Range Error
AVAR	Allan Variance

APPENDIX B Matlab® GPS Signal Simulator

Source code for the Satellite Simulator. Function listing:



gen_samples.m

```
function signal_out = gen_samples(output_file, sv_pos_mat, flight_traj)
% usage: signal_out = gen_samples(output_file, sv_pos_mat, flight_traj)
%
% This function generates down-converted sampled values of the L1 C/A GPS signal at
% the output of a GPS RF front end.  At present, the code is taylored to the Mitel
% GP2010 GPS Receiver RF front end.
%
% inputs:   output_file   name of file where the composite signal & time is stored
%                         The variable stored in the file is "gps_sig".
%
%           flight_traj   ECEF recv. position with time in true GPS time
%                         Col #:   1      2      3      4      5      6      7
%                         ------------------------------------------------------
%                                 time   X      Y      Z     V_x    V_y    V_z
%
%           sv_pos_mat    Satellite positions in ECEF with true GPS time
%                         Col #:   1      2      3      4      5
%                         ---------------------------------------
%                                 time  PRN#    X      Y      Z
%
% Output:  signal_out     Samples of down-converted C/A code spectrum
%
% the rest of inputs/parameters can be found in load_global_params.m  Note that the
% time in the receiver & satellite position files must be the same.  The rate can
% be variable (1Hz, 10Hz, etc.) but must be constant and common between the input
% files.  The gps_iq_model program will generate the sv_pos_mat file needed here
% in the correct format for a given trajectory file.
%
```

```
% This function calls the following functions:  load_global_params.m  (1)
%                                               get_tp_times.m        (2)
%                                                    test_time_match (2a)
%                                               sampled_recv_signal.m (3)
%                                                    get_corr_noise  (3a)
%                                               if_filt.m             (4)
%                                               gen_clock_errors.m    (5)
%                                                    gen_freq_errors (5a)
%                                                    get_allan_var   (5b)
%                                               sampled_recv_chan.m   (6)
%                                               calc_sig_amp.m        (7)
%                                               quantizer.m           (8)
%                              (simulink Block) agc_a2d.mdl           (9)
%
% This function is the 'parent' function for a set of functions comprising the model.
% The functions are called in the following way:
%-------------------------------------------------------------------------------------
% gen_samples
%           \
% |          load_global_params
%           \
% |          get_tp_times
%                      \
% |                     time_test_match
%           \
% |          sampled_recv_signal
%                           \
% |          |               get_corr_noise
%                                       \
% |          |                           if_filt.m
%                           gen_clock_errors
% |          |                           \
%                           |             gen_freq_errors
% |          |                           get_allan_var
%                           sampled_recv_chan
% |          |                           \
%                                        calc_sig_amp.m
% |          |               if_filt.m
%           \
% |          quantizer.m
%           \
%(Simulink)  agc_a2d.mdl
%-------------------------------------------------------------------------------------

% specific command for example 1:
% gen_samples('samples.mat',sv_pos_mat_ex1,trajectory_mat,PRN_VEC)
% gen_samples('samples.mat',sv_pos_mat(1:end-1,:),lab2,PRN_VEC)

tic
strtflop = flops;
%-------------------------------------------------------------------------------------
%  Perform input variable checking
%-------------------------------------------------------------------------------------
if nargin<2
   disp('Error: not enough input arguments');
   return
end
if isempty(output_file)|isempty(sv_pos_mat)|isempty(flight_traj)
   disp('Error: input argument(s) empty')
   return
end
%-------------------------------------------------------------------------------------
%  Clear global output variables from any previous run:
%-------------------------------------------------------------------------------------
clear global REAL_SNR IDEAL_SNR DATA_BITS PIT_CNT SNR_DB
clear global DOPPLER_FREQ_TRUE RAD_TO_SV_TRUE DOPPLER_FREQ RAD_TO_SV

%-------------------------------------------------------------------------------------
%  Start by loading/defining the global parameters with a call to:
%-------------------------------------------------------------------------------------
load_global_params;
% List the global parameters used in gen_samples: (note, the global is redudant here
% since all the global variables are present and initialized with the above function
% call.  However, this provides the ability to comment out the above line and
% interatively change global variables from call to call (without changing the file
% itself)
global START_TIME
global STOP_TIME
```

```
global RUN_TIME
global PIT_CNT
global PARSE_SIG_INTERVAL
global QUANTIZE_SIG
global DYNAMIC_AGC
global SAMPLING_DT
global AGC_DEBUG
global PRN_VEC
global ADD_OSC_ERRORS
global CARRIER_FREQUENCY
global SAMPLING_FREQUENCY
global DOPPLER_FREQ_TRUE
global RAD_TO_SV_TRUE
global DOPPLER_FREQ
global RAD_TO_SV
global DATA_MESG
global REAL_SNR
global IDEAL_SNR
global SNR_DB
global REPEATABILITY
global DATA_BITS
global CLOCK_ERRORS
global ML_QUANT_THRESH

% Commenting out the following line will return the GPS signal as a global variable
% in the workspace:
% global gps_sig

%-------------------------------------------------------------------------------
%  Do a "1st look" check to check time syncronization of the input files
%-------------------------------------------------------------------------------
index = find(sv_pos_mat(:,2)==PRN_VEC(1));
if sum(sv_pos_mat(index,1)-flight_traj(:,1))~=0
   disp('Error: time stamps in files not syncronized');
   return
end

%-------------------------------------------------------------------------------
% If START_TIME has not been initialized, then set start time to the first time
% in the input files, trunkated to the first millisecond
%-------------------------------------------------------------------------------
if isempty(START_TIME)
   START_TIME=ceil(sv_pos_mat(1,1)*1000)/1000;  %start at the first full millisecond
end
STOP_TIME = START_TIME+RUN_TIME; %calculate the real GPS time to stop the simulation
%-------------------------------------------------------------------------------
%  are start and stop times valid?
%-------------------------------------------------------------------------------
if isempty(find(START_TIME>=flight_traj(1,1)))|...
      isempty(find(ceil(STOP_TIME)<=flight_traj(end,1)))
   disp('Error: Start or Stop time is not included in range of input file')
   return
end

%-------------------------------------------------------------------------------
%  Random number initialization
%-------------------------------------------------------------------------------
if REPEATABILITY
   randn('state',200000); %resets it to the same state each time (200000 is arbitrary)
else
   randn('state',sum(100*clock)); %resets it to a different state each time.
end

%-------------------------------------------------------------------------------
%  Initialize local simulation parameters
%-------------------------------------------------------------------------------
t0=0; % simulation start time (see note at bottom of this file)
gps_sig = []; %initialize the final output variable
PIT_CNT=1;  %Pre-integration time counter, also an "code epoch" counter if PIT is 1ms

%-------------------------------------------------------------------------------
%  Begin main loop
%-------------------------------------------------------------------------------
while t0<RUN_TIME

   %----------------------------------------------------------------------------
   % Create the time vector for this section of samples
   %----------------------------------------------------------------------------
```

```
   time=[t0:SAMPLING_DT:(PIT_CNT*PARSE_SIG_INTERVAL)];
   % THE FOLLOWING LINE IS SPECIFIC TO THE MITEL 2010 SAMPLING RATE ONLY!!!!
   time=(round(time*10^9)/10^9);
   % one would think that the above line of code is unnecessry, however
   % experimentation has shown otherwise.  If you don't believe the following line
   % makes any difference, run diff.m on 'time' before and after executing the
   % following line.  Why the results are different, only Mathworks Knows.
   % If your still not convinced, run simulink with and without the following line,
   % and subtract the time vector in from the simulation time (generated by Simulink).
   % Note that maximum error is much less with the following line.
   % ****(DON'T TURN "interpolate data" off in the load from workspace block in
   % simulink--if the sample times are just slightly off, and this
   % box is unchecked, Simulink won't load those values.)****

   %-------------------------------------------------------------------------------
   % Get the signal propagation times (Tp) for each satellite for each sample time
   % Note:  These times multipled by the speed of light are the TRUE distances
   % (assuming no SV position error) from the recevier to each satellite at each
   % sample time
   %-------------------------------------------------------------------------------
   Tp_vec = get_Tp_times(time, sv_pos_mat, flight_traj);
   if isempty(Tp_vec)
      disp('Error:  Time stamps are not aligned, simulation aborted')
      return
   end

   %-------------------------------------------------------------------------------
   % Generate the sampled, non-quantized recevied signal @ the IF_AFTER_SAMP freq.
   % The vector returned here contains the L1 C/A code signals from each satellite
   % scaled to the approprate S/N ratio, sampled at the SAMPLING_FREQUENCY.  Other
   % switches in load_global_params control the other signal characteristics.
   %-------------------------------------------------------------------------------
   [composite_signal, clk_bias]=sampled_recv_signal(time, Tp_vec);

   %-------------------------------------------------------------------------------
   % Quantize the results--Quantization is currently 2-bit, 4-level, and implements
   % the Mitel sampling distribution (Magnitude bit has 30% duty cycle, and sign bit
   % has a 50% duty cycle.)  The MATLAB method selects the correct thresholds for
   % each section of the signal provided in the loop such that the resultant vector
   % is ~zero mean with all 1's & 3's (30% are 3's & 70% are 1's)
   %-------------------------------------------------------------------------------
   if QUANTIZE_SIG&(~DYNAMIC_AGC)
      % if using Dynamic AGC, wait until entire signal is generated, then quantize
      % it in Simulink
      composite_signal_quantized=quantizer(composite_signal);
   else
      composite_signal_quantized=composite_signal;
   end

   %-------------------------------------------------------------------------------
   % Concatenate the output variable with the receiver time.
   %-------------------------------------------------------------------------------
   samples = [time', composite_signal_quantized'];  %samples is a n X 2 vector

   %-------------------------------------------------------------------------------
   % Create (or append) to signal output variable
   %-------------------------------------------------------------------------------
   gps_sig = [gps_sig samples'];   %samples is a 2 X n*PIT_cnt vec.

   %-------------------------------------------------------------------------------
   %  Setup the correct starting sample time for the next time vector (next loop)
   %-------------------------------------------------------------------------------
   t0 = time(end) + SAMPLING_DT;
   PIT_CNT=PIT_CNT+1;  %increment the pre-integration time counter

   %-------------------------------------------------------------------------------
   % Occasionally output progress
   %-------------------------------------------------------------------------------
   if mod(time(end),0.003)>=(0.003-SAMPLING_DT)
      disp(sprintf('Done through time = %.3f   (%.1f%% completed)',time(end),...
         100*(time(end))/(RUN_TIME)))
      pause(0.001)
   end
   %-------------------------------------------------------------------------------
   % REPEAT LOOP
   %-------------------------------------------------------------------------------
end
```

```
%-------------------------------------------------------------------------------
% if using the Simulink AGC, perform quantization now:
%-------------------------------------------------------------------------------
if QUANTIZE_SIG&DYNAMIC_AGC
   global AGC_DEBUG
   % setup AGC parameters
   config_agc;
   disp('running simulink AGC, please wait');
   pause(.5);
   agc_a2d;
   % the following line speeds up the simulation:
   set_param('AGC_A2D','InvariantConstants','on');
   %set_param('AGC_A2D','LoadExternalInput','on');
   %set_param('AGC_A2D','ExternalInput','gps_sig'');
   % use the Matlab calculated Thresholds (without iteration) for initial guess
   set_param('AGC_A2D/Data Store Memory','InitialValue',...
      num2str(std(gps_sig(2,:))*(70/68.3),8));
   [T,X,Y]=sim('AGC_A2D',[gps_sig(1,1) gps_sig(1,end)],[],gps_sig');
   clear T X
   gps_sig(2,:) = Y'; %gps_sig_quan';
end

%-------------------------------------------------------------------------------
% Append the final doppler frequency and LOS distance in radians (RAD_TO_SV) to the
% global variables DOPPLER_FREQ and RAD_TO_SV.  NOTE:  the FINAL values are then for
% the END of the last sample time!!
%-------------------------------------------------------------------------------
for i=1:length(PRN_VEC)
   if ADD_OSC_ERRORS
      % Estimate the TRUE Received Doppler at the END of this correlation period
      DOPPLER_FREQ_TRUE{i}(PIT_CNT)=(Tp_vec(i,end-500)-Tp_vec(i,end))*...
         CARRIER_FREQUENCY*SAMPLING_FREQUENCY/500;
      %Record the END phase of the received waveform (LOS dist. expressed in Radians)
      RAD_TO_SV_TRUE{i}(PIT_CNT)=(2*pi*CARRIER_FREQUENCY*Tp_vec(i,end));
      Tp_vec = Tp_vec-repmat(clk_bias,length(Tp_vec(:,1)),1);
   end
   % Estimate the perceived Received Doppler for the END of this correlation period
   DOPPLER_FREQ{i}(PIT_CNT)=(Tp_vec(i,end-500)-Tp_vec(i,end))*CARRIER_FREQUENCY*...
      SAMPLING_FREQUENCY/500;
   %Record the perceived END phase of the signal (LOS dist. expressed in Radians)
   RAD_TO_SV{i}(PIT_CNT)=(2*pi*CARRIER_FREQUENCY*Tp_vec(i,end));
end


save(output_file, 'gps_sig','DOPPLER_FREQ','DOPPLER_FREQ_TRUE','RAD_TO_SV',...
   'RAD_TO_SV_TRUE','DATA_BITS','CLOCK_ERRORS','ML_QUANT_THRESH','IDEAL_SNR',...
   'REAL_SNR','PRN_VEC','SNR_DB','RUN_TIME','START_TIME');
if nargout>0;
   signal_out = gps_sig;
end
stopflp = flops;
mesg = ['number of flops: ' num2str(stopflp-strtflop)];
disp(mesg)
toc
clear global PIT_CNT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  Notes:  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% We must start our time vector at 0 rather than true GPS time due to the limitations
% of double precision numbers.  note that if:
% time=[0:SAMPLING_DT:(1000*SAMPLING_DT)];
% time(2)-time(1)
% yields: 1.750000000000000e-007
% but if:
% time=[400635:SAMPLING_DT:(1000*SAMLING_DT+400635)];
% time(2)-time(1)
% yields: 1.749722287058830e-007
% this lack of accuracy causes problems later.  So we will start by stripping and
% storing GPS time.
```

load_global_params.m

```
% These are the global variables that are visible by all other functions
% This file functions as the "control file" for the Matlab IF signal and Simulink
% Correlator models
```

```
% global "CONSTANTS" (i.e., functions of the hardware or GPS system parameters)
% GPS Constants:
global CARRIER_FREQUENCY      % L1 carrier frequency
global CARRIER_WAVELENGTH     % L1 wavelength in meters.
global CHIPPING_RATE
global T_C                     % the time period of one chip
global SPEED_OF_LIGHT
global CA_CODE                % cell array containing all the gold code chips.
% Hardware Dependent Constants:
global SAMPLING_FREQUENCY
global SAMPLING_DT            % sampling delta time, or the time between samples.
global IF_BEFORE_SAMP
global IF_AFTER_SAMP
global LATCHING_DT
global CHIP_SPACING
global CLOCK_PARAMS
% Simulink AGC parameters
global CYC_RES_CNT
global GAIN
global INIT_CYCLES
global AGC_DEBUG
% Control/Input parameters
global RUN_TIME               %DETERMINES THE NUMBER OF SAMPLES SIMULATED
global START_TIME             %specify start time in GPS seconds, see note below
global STOP_TIME              %declare here, define in gen_samples
global PARSE_SIG_INTERVAL
global NOISE_POWER
global SNR_DB
global A2D_FACT
global PRN_VEC
global PIT_CNT                %declare here, define in gen_samples
% Output parameters
global DOPPLER_FREQ           %declare & init. here, define in sampled_recv_chan(s)
global RAD_TO_SV              %declare & init. here, define in sampled_recv_chan(s)
global DOPPLER_FREQ_TRUE      %declare here, define in sampled_recv_chan(s)
global RAD_TO_SV_TRUE         %declare here, define in sampled_recv_chan(s)
global DATA_BITS              %declare here, define in sampled_recv_chan(s)
global REAL_SNR               %declare here, define in sampled_recv_signal
global IDEAL_SNR              %declare here, define in sampled_recv_signal
global CLOCK_ERRORS           %declare here, define in gen_clock_errors
global ML_QUANT_THRESH        %delcare here, define in quantizer

% switches: (boolean values)
% Signal Simulator & Receiver Model Switches
global APPLY_CA
global QUANTIZE_SIG
% Signal Simulator Switches
global ADD_NOISE
global ADD_OSC_ERRORS
global APPLY_DATA
global FILTER_SIG
global FILTER_NOISE
global DYNAMIC_AGC
global SAVE_QUANT_THRESH
global SAVE_CLK_ERRORS
global REPEATABILITY
% Receiver Model Switches
global MIX_TO_BASEBAND
global DUMP_ON_EPOCH
global SAVE_TIME
global CHAN_ENABLE
global SAVE_CH1_SIGS
global SAVE_CH2_SIGS
global SAVE_CH3_SIGS
global SAVE_CH4_SIGS
global SAVE_CH5_SIGS
global SAVE_CH6_SIGS
global SAVE_CH7_SIGS
global SAVE_CH8_SIGS
global SAVE_CH9_SIGS
global SAVE_CH10_SIGS
global SAVE_CH11_SIGS
global SAVE_CH12_SIGS
global DEBUG_DECIMATION
global SAVE_CORR              %single channel only
global SAVE_DCO_OUT           %single channel only
global SAVE_BASEBAND          %single channel only
global SAVE_SIM_CODE          %single channel only
```

```
global SAVE_FEE_REF          %single channel only
% Simulink AGC switches
global CLOSED_LOOP
global SAVE_AGC_VAR

%-------------------------------------------------------------------------------%
%                    define the 'constants' (for GPS)
%-------------------------------------------------------------------------------%
CHIPPING_RATE = 1.023E6;              % CA code chipping rate (Hz)
T_C = 1/CHIPPING_RATE;                % Length (in seconds) of a C/A code chip.
CARRIER_FREQUENCY = 1575.42E6;        % L1 received frequency (Hz)
SPEED_OF_LIGHT=299792458.0;           % (meters/second)
CARRIER_WAVELENGTH=SPEED_OF_LIGHT/CARRIER_FREQUENCY;  %L1 wavelength in Meters
load ca_codes;                        % load cell array containing ALL the gold
%                                     % sequences for all 37 PRNS
CA_CODE=CA_code;                      % switch case to global variable case
clear CA_code;                        % clear variable with incorrect case
%-------------------------------------------------------------------------------%
%        define the frequency plan and other hardware dependent variables
%-------------------------------------------------------------------------------%
SAMPLING_FREQUENCY = 40E6/7;           % From the Mitel GP2021 chip (Hz)
SAMPLING_DT = 1.75e-7;                 % 1/SAMPLING_FREQUENCY (sec/cycle)
                                       % "DT" = "Delta T" time between samples
IF_BEFORE_SAMP = 35.42E6-1400E6/45;    % about 4.309e6 Hz, From Mitel GP2010 chip
IF_AFTER_SAMP = 88540000/63;           % about 1.405e6 Hz, From Mitel GP2010 chip
LATCHING_DT = 0.001;                   % latch interval in Simulink when all
                                       % channels are latched at the same time (s)
CHIP_SPACING = T_C/2;                  % Correlator Chip spacing in Simulink model
CLOCK_PARAMS.ALLAN_VAR.H0 = 2e-19;     % Allan Variance h0 (h_zero) parameter
CLOCK_PARAMS.ALLAN_VAR.H_1 = 7e-21;    % Allan Variance h-1 (h minus 1) parameter
CLOCK_PARAMS.ALLAN_VAR.H_2 = 2e-20;    % Allan Variance h-2 (h minus 2) parameter
CLOCK_PARAMS.MAX_FREQ_ERR = 2.5e-6;    % Max frequency variation of Crystal (ppm)
CLOCK_PARAMS.TAU          = 1;         % Tau value to plug into Allan Var. Plot.
                                       % to determine noise for random walk or
                                       % random constant
%-------------------------------------------------------------------------------%
%             define the parameters that control and define the simulation
%-------------------------------------------------------------------------------%
if isempty(DOPPLER_FREQ)&isempty(RAD_TO_SV);  %make sure they are zero first.
   for i=1:12
      DOPPLER_FREQ{i}=0;               % initialize here to the max size
      RAD_TO_SV{i}=0;                  % initialize here to the max size
   end
else
   disp('DOPPLER_FREQ & RAD_TO_SV arrays already filled--not initialized')
end
RUN_TIME=0.02;                         % specifies the length of the simulation in
%                                      % seconds rounded to the nearest millisecond.
PARSE_SIG_INTERVAL=0.001;              % defines the size of the timevector
                                       % created in gen_samples.  Currently
                                       % defines the number of samples the Matlab AGC
                                       % processes at once. (seconds)
START_TIME=400657; % if you fix the start time here, it will be used, otherwise
                   % first time of the satellite and trajectory files will be used.
A2D_FACT = 4;       % for Correlated Noise Generation (see IF_filt.m)
PRN_VEC = [4,8];%:8;     % PRN_VEC defines both which signals are included in the
                       % composite signal and the channel asignments.  The index
                   % location of PRN_VEC corresponds to a particular Channel.  For
                   % example, the PRN# in PRN_VEC(3) will be assigned
                   % to the 3rd channel of the 12-channel correlator.
% To set the signal to noise ratios, use a loop for now (set them all the same)
% As with PRN_VEC, SRN_DB is a channel indexed vector, so that the first element
% defines the SNR ratio for CHANNEL 1, NOT PRN 1. SNR_DB(4) is the SNR of CHANNEL
% 4, not PRN 4.
% Note: many applications will require being able to vary the SNR over the
% course of run, but for right now the SNR will be fixed for the duration of
% run for each signal.  Future enhancements must be made here.
for i=PRN_VEC
   % SNR_DB represents the signal-to-noise ratio at the output of the GPS front-end
   SNR_DB(i==PRN_VEC)=-15;     %indexed by the channel assignments of PRN_VEC
end
clear i;
NOISE_POWER = 100; % specifies the power of the noise (arbitrary value from which
%                  % the signal amplitudes are derived by relation to maintain the
%                  % given SNR specified above.
%-------------------------------------------------------------------------------%
% define the SWITCHES that toggle models on and off and generate debugging info
%-------------------------------------------------------------------------------%
```

```
APPLY_CA      =1; % 1 =>apply CA-code to all measurements, 0 => no CA code mod.
                  %This switch also turns on/off the code correlation block
DUMP_ON_EPOCH =0; % 1 => latch a channel's I & Q values on its code epoch
                  % 0 => latch all I & Q values at 1ms time increments
APPLY_DATA    =0; % 1 => apply a Random Data modulation, 0 => don't
                   %Matlab Output Variable:  DATA_MESG & DATA_BITS
QUANTIZE_SIG  =1; % 1 => apply AGC quantization to signal, 0 => don't
ADD_NOISE     =1; % 1 => add noise to the signals, 0 => don't
ADD_OSC_ERRORS=0; % 1 => add oscillator errors to the signal, 0 => don't
CLOCK_PARAMS.CONSTANT_DRIFT = 0;  % Switch specifing constant frequency drift
                                  % "1" => a constant drift model is to be used
                                  % "0" => bounded random frequency drift.
FILTER_SIG    =0; % 1 => apply bandpass filtering to the signal, 0 => don't
FILTER_NOISE  =0; % 1 => apply bandpass filtering to the noise, 0 => don't
DYNAMIC_AGC   =0; % 1 => use simulink dynamic AGC to quantize samples, 0 => Matlab
REPEATABILITY = 1;  %if you want the same noise vector for each run, set to "1"
MIX_TO_BASEBAND=1;%switch for engaging cosine mixing block in simulink     (1=>ON)
SAVE_BASEBAND =0; %switch for saving off zero-frequency samples in Simulink(1=>ON)
                   %Simulink Output Variable:  Baseband
SAVE_CORR     =0; %switch for saving off Correlated values of Simulink run (1=>ON)
                   %Simulink Output Variable:  Correlated
SAVE_DCO_OUT  =0; %switch for saving off DCO values of simulink run        (1=>ON)
                   %Simulink Output Variable:  DCO_output
SAVE_TIME     =0; %switch for saving off time of Simulink run              (1=>ON)
                   %Simulink Output Variable:  Sim_times
SAVE_SIM_CODE =0; %switch for saving sumulated PRN waveforms in simulink   (1=>ON)
                   %Simulink Output Variable:  Sim_code
SAVE_FEE_REF  =0; %switch that saves the radian measurements for each sample
                  %time in Simulink                                        (1=>ON)
                   %Simulink Output Variable:  Sim_fee_ref
SAVE_CLK_ERRORS=0;%switch that saves the clock drift and error values      (1=>ON)
                   %Matlab Output Variable:    CLOCK_ERRORS
SAVE_QUANT_THRESH=1;%switch that saves the quantization thresholds in Matlab (1=>ON)
                   %Matlab Output Variable:    ML_QUANT_THRESH
%-------------------------------------------------------------------------------%
%         define the parameters that control and define the Simulink AGC
%-------------------------------------------------------------------------------%
CYC_RES_CNT    = 5714;     % Number Of samples between AGC updates
GAIN           = 8;       % AGC feedback gain (Multiplied by Normalized Errors)
INIT_CYCLES    = 1000;     % number of samples to pre-load integration
%                            (INIT_CYCLES + CYC_RES_CNT = averaging interval)
%-------------------------------------------------------------------------------%
%              define the SWITCHES that control the Simulink AGC
%-------------------------------------------------------------------------------%
CLOSED_LOOP    = 1;        % Operate AGC in Closed Loop Mode
SAVE_AGC_VAR   = 0;        % Debug Mode, save off parameters and view signals
                           % Matlab Output Variable:  AGC_DEBUG
%-------------------------------------------------------------------------------%
%              define the SWITCHES that control the GPS_rcvr_model
%-------------------------------------------------------------------------------%
CHAN_ENABLE      = [1;0;0;0;0;0;0;0;0;0;0;0];   % Enabling switches for each channel
SAVE_CH1_SIGS=0;           % Enable for saving off signals inside a channel
SAVE_CH2_SIGS=0;
SAVE_CH3_SIGS=0;
SAVE_CH4_SIGS=0;
SAVE_CH5_SIGS=0;
SAVE_CH6_SIGS=0;
SAVE_CH7_SIGS=0;
SAVE_CH8_SIGS=0;
SAVE_CH9_SIGS=0;
SAVE_CH10_SIGS=0;
SAVE_CH11_SIGS=0;
SAVE_CH12_SIGS=0;
DEBUG_DECIMATION = [1;1;1;1;1;1;1;1;1;1;1;1];   % Decimation intervals for the channel
                                                % signals
%
%
% Reasonably bad dynamics occur at t=400657.
% Good dynamics (non-existent) have been simulated at t=399100.
% lab2's first time:  399000
% lab2's last time:   400680
```

get_tp_times.m

```
function Tp_vec = get_Tp_times(time, sv_pos_mat, trajectory_mat);
% usage: Tp_vec = get_Tp_times(time, sv_pos_mat, trajectory_mat);
%
% time = vector of receiver sample times (True GPS time) (1xn)
%
% sv_pos_mat: (Error Free)          trajectory_mat: (Error Free)
% col1 = time (True GPS time)       col1 = time (True GPS time)
% col2 = PRN                        col2 = X ECEF position
% col3 = X ECEF coordinate          col3 = Y ECEF position
% col4 = Y ECEF coordinate          col4 = Z ECEF position
% col5 = Z ECEF coordinate          col5 = X ECEF velocity
%                                   col6 = Y ECEF velocity
% PRN_VEC = global vector of PRNs   col7 = Z ECEF velocity
%
% Tp_vec = Output matrix of Tp times, each row is for the respective PRN
%          in the PRN_VEC and the columns are the Tp times sync. to "time"
%
% from these files, the exact distance between each SV and the receiver can
% be determined at any time inside the time included in the files by using the
% spline function.  These exact distances are then divided by the speed of light
% to get the exact "Tp" times, or propogation times for each signal from each
% satellite

% future versions could just use the perfect pseudoranges instead of
% recalculating them from the SV positions and the Trajectory positions.

global SPEED_OF_LIGHT;
global START_TIME;
global RUN_TIME;
global SAMPLING_DT;
global PRN_VEC;
c=SPEED_OF_LIGHT;       % define shorter hand notation for light speed
persistent Tp XX

if time(1)<3*SAMPLING_DT  %If this file is being called the first time, initialize
   % get the 'data' rate of the input variables
   time_step=(trajectory_mat(2,1)-trajectory_mat(1,1));
   for PRN=PRN_VEC
      PRN_row_index = find(sv_pos_mat(:,2)==PRN);
      %sv_pos contains SV positions for current PRN at each time
      sv_pos=sv_pos_mat(PRN_row_index,[1 3:5]);
      % check to make sure time indexes between satellite positions & trajectory
      % match (checks for EACH prn, not just the first one, see start of gen_sample)
      not_ok = test_time_match(sv_pos,trajectory_mat);
      if not_ok;return;end;
      % get the transmit times for every "time" in the input files for the current PRN
      % Note:  Tp is a row vector (1xn)
      Tp(find(PRN_VEC==PRN),:)=(sum(((sv_pos(:,2:4)-...
         trajectory_mat(:,2:4)).^2)').^0.5)./c;
   end
   %form time vector, placing 0 at the START_TIME
   XX=[(trajectory_mat(1,1)-START_TIME):time_step:(trajectory_mat(end,1)-START_TIME)];
% Tp now contains the propagation time of signals from each satellite for each time
% step found in the trajectory and satellite position input files.  This step only
% needs to be done once.
end

% Now do the spline interpolation to get the propagation times for the current ms
% being generated.  Note that the rows of Tp_vec are equivalent with channel
% assignments, i.e. Row 4 contains the signal assigned to Channel 4

Tp_vec=spline(XX,Tp,time);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function no_match = test_time_match(sv_pos,trajectory_mat)
% just tests to see the time stamps on the satellite positions and the trajectory
% position match

if length(sv_pos(:,1))~=length(trajectory_mat(:,1));
   disp('different number of SV positions and trajectory time tags')
   Tp_vec = [];
   no_match=1;
end
common_time_test = sv_pos(:,1)~=trajectory_mat(:,1);
if sum(common_time_test)
   disp('SV positions and trajectory time tags are not aligned')
```

```
   Tp_vec = [];
   no_match=1;
end
no_match=0;

%code archive:
%global PSEUDORANGES % ????
   % old, slower way:
   %for i=1:length(sv_pos(:,1))
   %   % get the propagation times for each time step in the inputs files for one PRN
   %   Tp(1,i)=norm(sv_pos(i,2:4)-trajectory_mat(i,2:4))/c;
   %end
   %XX=[0:.001:(trajectory_mat(end,1)-START_TIME)];
   %YY=spline((trajectory_mat(:,1)-START_TIME),Tp,XX);
   %Tp=fix(Tp*1e16)/1e16;
%Tp_interp=spline(XX,Tp,time);
%Tp_vec(find(PRN_VEC==PRN),1:length(time))=Tp_interp;
```

sampled_recv_signal.m

```
function [composite_sig,clk_bias]=sampled_recv_signal(time, Tp_vec)
% usage: [composite_sig,clk_bias]=sampled_recv_signal(time, Tp_vec)
%
%
% Measurment simulator
% This function generates the GPS (sampled) signal over the desired time
% vector.

global PIT_CNT
global NOISE_POWER
global IF_AFTER_SAMP
global REAL_SNR
global IDEAL_SNR
global ADD_NOISE
global ADD_OSC_ERRORS
global PRN_VEC
persistent IDEAL_SNR_mat
persistent RUNNING_TOTAL
if PIT_CNT==1
   RUNNING_TOTAL=0;
end
global AVE_NOISE_POWER

% initialize the composite signal (with/without the noise, depending on ADD_NOISE):
composite_sig=get_corr_noise(length(time),NOISE_POWER);
if ADD_NOISE
   % calculate Noise Power:
   Pow_noise=sum(composite_sig.^2)/length(composite_sig);
   RUNNING_TOTAL=RUNNING_TOTAL+Pow_noise;
   AVE_NOISE_POWER = RUNNING_TOTAL/PIT_CNT;
end
% Generate clock errors:
if ADD_OSC_ERRORS
   % get the clock drift and clock bias terms for each sample time
   [clk_drft clk_bias phase_noise]=gen_clock_errors(time);
else
   clk_drft = zeros(1,length(time));
   clk_bias = zeros(1,length(time));
   phase_noise = zeros(1,length(time));
end

% process the channels individually.
for i=1:length(PRN_VEC)
   sig_chan = sampled_recv_chan(time, Tp_vec(i,:), PRN_VEC(i), clk_drft,...
      clk_bias, phase_noise);
   sig_chan_filt = IF_filt(sig_chan,'signal');
   if ADD_NOISE
      % calculate Signal Power:
      Pow_sig(i)=sum(sig_chan_filt.^2)/length(sig_chan_filt);
      % calculate S/N ratio:
      IDEAL_SNR{i}(PIT_CNT) = 10*log10(Pow_sig(i)/Pow_noise);
   end
   composite_sig = composite_sig + sig_chan_filt;
end
```

```
   % calculate the power in the composite signal
if ADD_NOISE
   Pow_composite_sig = sum(composite_sig.^2)/length(composite_sig);
   for i=1:length(PRN_VEC)
      REAL_SNR{i}(PIT_CNT) = 10*log10(Pow_sig(i)/Pow_composite_sig);
   end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function noise_vec=get_corr_noise(num_samples,noise_power);
%
% usage: noise_vec=get_corr_noise(num_samples,noise_power);
%
% generates a noise_vector "num_samples" long of correlated (band-limited)
% noise containing "noise_power" in the 2 MHz C/A code bandwidth.

global A2D_FACT
global SAMPLING_FREQUENCY
global CHIPPING_RATE
global IF_BEFORE_SAMP
global ADD_NOISE
global FILTER_NOISE

if ADD_NOISE
   if FILTER_NOISE
      % Pn needs to represent the power of the noise at the output of the GPS
      % front end, so if IF filtering is applied, we must scale it appropriately:
      % ( %1.02 is a "fudge factor" determined experimentally, and is due to the
      % characteristics of the filter used.)
      %Pn = SAMPLING_FREQUENCY/(2*CHIPPING_RATE)*noise_power*1.02;
      Pn = SAMPLING_FREQUENCY/(CHIPPING_RATE)*noise_power*1.02;
      % Calculate noise
      %A2D_FACT controls the resolution of the IF_filtering of the noise.
      In=randn(1,(A2D_FACT*num_samples))*sqrt(Pn);
      In=IF_filt(In,'noise');  %Bandpass filter the noise
      noise_vec=In(1:A2D_FACT:end);  %'sample' the noise back to IF (digitized IF).
   else
      % since the noise is not filtered, just use the specfied noise power:
      noise_vec=sqrt(noise_power)*randn(1,num_samples);
   end
else
   noise_vec=zeros(1,num_samples);
end
```

if_filt.m

```
function filtered_signal = IF_filt(signal,sig_or_noise)
% usage: filtered_signal = IF_filt(signal,sig_or_noise)
%
% filters the signal using an elliptical filter with parameters listed in this file.
%
% sig_or_noise is a string that either says "noise" or "signal" to tell the function
% how to filter.

global FILTER_SIG
global FILTER_NOISE

switch sig_or_noise

case 'noise'
   if FILTER_NOISE
      persistent a_nz b_nz
      % load only the global parameters needed . .
      global SAMPLING_FREQUENCY
      global IF_BEFORE_SAMP
      global CHIPPING_RATE
      global A2D_FACT
      if isempty(a_nz)  % if noise parameter is empty, define noise parameters
         fs=A2D_FACT*SAMPLING_FREQUENCY;  % Sampling rate for NOISE
         % at this point, the file is being run for the first time, in which case
         % a & b need to be defined.  The parameters used here are based off of the
         % specifications of the SAW filter used in the Mitel Receiver Chipset.  See
         % the spec. sheet on the DW9255 from Mitel for more specifications on this
         % filter.  (Note that this filter operates at a higher IF frequency, but it
```

```
        % has the sharpest cutoffs of any of the IF filters, so it's effects can be
        % simulated at lower IF frequencies with acceptable realism.
        %
        fc=IF_BEFORE_SAMP;      % Center frequency of the bandpass filter(Hz)
        Wnn=2*CHIPPING_RATE;    % Bandwidth (null-to-null for BPF) of the filter (Hz)
        %-------------------------------------------------------------------------
        %  Choose between a nearly ideal filter (very tight) or a more realistic
        %  filter, based on the SAW filter used with the Mitel chipset.
        %-------------------------------------------------------------------------
        ideal = 0;
        if ideal
           stop_width=Wnn/50;     % Difference between the bandpass corner frequency
           %                      % and the stopband frequency (Hz)
           Rp=0.05;               % Bandpass ripple (dB)
           Rs=60;      % Sidelobe level down from the peak value in the passband (dB)
        else
           stop_width=Wnn/4;      % Difference between the bandpass corner frequency
           %                      % and the stopband frequency (Hz)
           Rp=0.8;                % Bandpass ripple (dB)
           Rs=35;      % Sidelobe level down from the peak value in the passband (dB)
        end
        freq_pts=2^15;  % Number of frequency points for filter frequency response
        %           % (int) The default is 512.  Always specify a power of two for
        %           % fast computation of the FFT algorithm
        %
        % calulate the Matlab filter parameters from the above filter statistics.
        % Wp and Ws have values between 0 and 1, where 1
        % corrsponds to half the sampling frequency (fs/2)
        Wp=[(fc-Wnn/2) (fc+Wnn/2)]*2/fs;                        % Cutoff frequencies.
        Ws=[(fc-Wnn/2-stop_width) (fc+Wnn/2+stop_width)]*2/fs;% Stopband frequencies.
        % use an elliptic filter.  See Filt_ord.m for performance analysis of this
        % filter
        [n Wn]=ellipord(Wp,Ws,Rp,Rs);  % get the order and cuttoff frequencies
        [b_nz,a_nz]=ellip(n,Rp,Rs,Wn);
        display_response = 0;
        if display_response
           [H,f,units]=freqz(b_nz,a_nz,freq_pts,fs);
           freqzplot(H,f,units)
           title('Noise Filter Frequency Response')
        end
     end
     a=a_nz;
     b=b_nz;
     fs=A2D_FACT*SAMPLING_FREQUENCY;  % Sampling rate for NOISE
  else
     % If IF filtering is turned off . .
     filtered_signal = signal;
     return
  end

case 'signal'
  if FILTER_SIG
     persistent a_sig b_sig
     % load only the global parameters needed . .
     global SAMPLING_FREQUENCY
     global IF_AFTER_SAMP
     global CHIPPING_RATE
     if isempty(a_sig)  % if noise parameter is empty, define noise parameters
        fs=SAMPLING_FREQUENCY;           % Sampling rate for signal
        % at this point, the file is being run for the first time, in which case
        % a & b need to be defined.  The parameters used here are based off of the
        % specifications of the SAW filter used in the Mitel Receiver Chipset.  See
        % the spec. sheet on the DW9255 from Mitel for more specifications on this
        % filter.  (Note that this filter operates at a higher IF frequency, but it
        % has the sharpest cutoffs of any of the IF filters, so it's effects can be
        % simulated at lower IF frequencies with acceptable realism.
        %
        fc=IF_AFTER_SAMP;      % Center frequency of the bandpass filter(Hz)
        Wnn=2*CHIPPING_RATE;    % Bandwidth (null-to-null for BPF) of the filter (Hz)
        ideal =0;  %choose an (near) ideal filter (very tight)
        if ideal
           stop_width=Wnn/50;     % Difference between the bandpass corner frequency
           %                      % and the stopband frequency (Hz)
           Rp=0.05;               % Bandpass ripple (dB)
           Rs=60;      % Sidelobe level down from the peak value in the passband (dB)
        else
           stop_width=Wnn/4;      % Difference between the bandpass corner frequency
           %                      % and the stopband frequency (Hz)
```

```
            Rp=0.8;                % Bandpass ripple (dB)
            Rs=35;     % Sidelobe level down from the peak value in the passband (dB)
         end
         freq_pts=2^15;  % Number of frequency points for filter frequency response
         %           % (int) The default is 512.  Always specify a power of two for
         %           % fast computation of the FFT algorithm
         %
         % calulate the Matlab filter parameters from the above filter statistics.
         % Wp and Ws have values between 0 and 1, where 1
         % corrsponds to half the sampling frequency (fs/2)
         Wp=[(fc-Wnn/2) (fc+Wnn/2)]*2/fs;                      % Cutoff frequencies.
         Ws=[(fc-Wnn/2-stop_width) (fc+Wnn/2+stop_width)]*2/fs;% Stopband frequencies.
         % use an elliptic filter.  See Filt_ord.m for performance analysis of this
         % filter
         [n Wn]=ellipord(Wp,Ws,Rp,Rs);  % get the order and cuttoff frequencies
         [b_sig,a_sig]=ellip(n,Rp,Rs,Wn);
         display_response = 0;
         if display_response
            [H,f,units]=freqz(b_sig,a_sig,freq_pts,fs);
            freqzplot(H,f,units)
            title('Signal Filter Frequency Response')
         end
      end
      a=a_sig;
      b=b_sig;
      fs=SAMPLING_FREQUENCY;           % Sampling rate for signal
   else
      % If IF filtering is turned off . .
      filtered_signal = signal;
      return
   end
end % matches switch statement

% filter the signal.  when signal is a matrix, filter operates on the columns
filtered_signal = filter(b,a,signal')';  %elliptic filter

% Show PSD plots before and after filtering for debugging purposes
%******** Switch *********
plot_filt_sig = 0;
if plot_filt_sig
   get_psd(signal,fs,'signal before filtering')
   get_psd(filtered_signal,fs,'signal after filtering')
end
```

gen_clock_errors.m

```
function [clk_drft,clk_bias,clk_jitter]=gen_clock_errors(time);
% usage: [clk_drft,clk_bias,clk_jitter]=gen_clock_errors(time);
%
% for a given time vector of length n, generate clock drift & clock bias
% values for each sampletime in "time". Note that the contents of time
% are unimportant past the first call, which looks for a nearly zero start
% time to declare initalization.
%
%

global SAMPLING_DT
global CLOCK_PARAMS
global CLOCK_ERRORS
global SAVE_CLK_ERRORS
persistent LAST_CLK_DRFT
persistent LAST_CLK_BIAS

if time(1)<2*SAMPLING_DT
   LAST_CLK_DRFT = 0;
   LAST_CLK_BIAS = 0;
   CLOCK_ERRORS.DRIFT = [];
   CLOCK_ERRORS.BIAS = [];
end

% Clock modeling in this way is an evolving process.  I have not found anyone or any
% documents claiming to model clock errors at this level, in this way.  As such, this
% file shall be considered "a work in progress" and past iterations of ideas will be
% commented out, rather than deleted, for educational purposes.
```

```
%-------------------------------------------------------------------------------
% the following was the first cut, which has been modified/abandoned.  The reference
% was how to model clocks in a Kalman Filter, not in a truth environment.  There are
% some key differences.
% -------------------------------------------------------------------------------
% Get the power (Variance) of the noises in the clock model:
% source:  Brown and Hwang, Introduction of Random Signals and Applied Kalman Filtering,
% John Wiley & Sons, 1992
% E[Wc1^2] = Expected value of the Clock bias,
% wc1=2*CLOCK_PARAMS.ALLAN_VAR.H0/SAMPLING_DT;
% E[Wc2^2] = Expected value of the Clock drift,
% wc2=8*pi^2*CLOCK_PARAMS.ALLAN_VAR.H_2*SAMPLING_DT;

% for now, we will not model any clk_jitter
clk_jitter = zeros(1,length(time));

if CLOCK_PARAMS.CONSTANT_DRIFT
   if time(1)<2*SAMPLING_DT
      % setup random constant (fixed constant drift)
      % base it on the allan variance for .001 second.
      LAST_CLK_DRFT = sqrt(get_allan_var(CLOCK_PARAMS.TAU))*randn(1,1);
   end
   %Wc1_vec=sqrt(wc1)*randn(1,length(time)); % add to clock drift and integate to get
bias
   %clk_jitter=Wc1_vec;
   %Wc1_vec(1)=LAST_CLK_BIAS;  % inital cond. of integral, from last chunk of data.
   % model clock drift as a constant offset:
   clk_drft = repmat(LAST_CLK_DRFT,1,length(time));
   % add white noise to the clock drift, and integrate:
   %clk_bias = cumsum(clk_drft+Wc1_vec);
   clk_bias = SAMPLING_DT*cumsum(clk_drft)+LAST_CLK_BIAS;
   % save off the last values of clock drift and bias for the next time it's called:
   LAST_CLK_DRFT=clk_drft(end);
   LAST_CLK_BIAS=clk_bias(end);
else
   % Create the noise vectors:
   %Wc1_vec=sqrt(wc1)*randn(1,length(time)); % add to clock drift and integate to get
bias
   %clk_jitter=Wc1_vec; %mult (Delta t)/t
   %Wc2_vec=sqrt(wc2)*randn(1,length(time)); % integrate to get clock drift
   %Wc2_vec(1)=LAST_CLK_DRFT;  % initial condition of the integral, from last chunk of
   %Wc1_vec(1)=LAST_CLK_BIAS;  % data.
   % Integrate noise to get the clock drift:
   clk_drft = gen_freq_errors(length(time),LAST_CLK_DRFT);
   % add white noise to the integrated value, and integrate again:
   %clk_bias = SAMPLING_DT*cumsum(clk_drft+Wc1_vec);
   clk_bias = SAMPLING_DT*cumsum(clk_drft)+LAST_CLK_BIAS;
   % save off the last values of clock drift and bias for the next time it's called:
   LAST_CLK_DRFT=clk_drft(end);
   LAST_CLK_BIAS=clk_bias(end);
end

% Save off the clock errors?
if SAVE_CLK_ERRORS
   if CLOCK_PARAMS.CONSTANT_DRIFT
      CLOCK_ERRORS.DRIFT = [clk_drft(1)];
      CLOCK_ERRORS.BIAS = [CLOCK_ERRORS.BIAS clk_bias];
   else
      CLOCK_ERRORS.DRIFT = [CLOCK_ERRORS.DRIFT clk_drft];
      CLOCK_ERRORS.BIAS = [CLOCK_ERRORS.BIAS clk_bias];
   end
end



function freq_err = gen_freq_errors(numelemnts, last_freq_err)
%
%
% freq_err = delta_f/f (unitless) parameter of clock drift (of the crystal)

global SAMPLING_FREQUENCY
global CLOCK_PARAMS

max_error = CLOCK_PARAMS.MAX_FREQ_ERR;
%tau = logspace(log10(0.001),log10(0.01),10);  %create number of vectors
% get the scale factor for this vector
aln_std = sqrt(get_allan_var(CLOCK_PARAMS.TAU))/SAMPLING_FREQUENCY;
freq_err=zeros(1,numelemnts);
```

```
% get the random vector
randvec = aln_std*randn(1,numelemnts);
randvec = cumsum(randvec);
%figure(50)
%plot(randvec)
freq_err = last_freq_err + randvec;
%figure(51)
%plot(freq_err)
while (find(abs(freq_err)>max_error))>0;
   %figure(52)
   %plot(freq_err)
   if (find(freq_err>max_error))>0;
      pts_outside = find(freq_err>max_error);
      freq_err(pts_outside)=-freq_err(pts_outside);
      freq_err(pts_outside)=+2*freq_err(pts_outside(1)-1)+freq_err(pts_outside);
   end
   if (find(freq_err<-max_error))>0;
      pts_outside = find(freq_err<-max_error);
      freq_err(pts_outside)=-freq_err(pts_outside);
      freq_err(pts_outside)=+2*freq_err(pts_outside(1)-1)+freq_err(pts_outside);
   end
   %figure(53)
   %plot(freq_err)
end

%-------------------------------------------------------------------------------


function aln_var = get_allan_var(tau)
%
% returns the allan variance value for a given averaging interval tau

global CLOCK_PARAMS

% Reference GLOBAL POSITIONING SYSTEM:  THEORY AND APPLICATIONS, VOL 1.
% Parkinson & Spilker, P. 156

aln_var = CLOCK_PARAMS.ALLAN_VAR.H_2*((2*pi)^2/6).*tau+CLOCK_PARAMS.ALLAN_VAR.H_1*...
   2*log(2)+CLOCK_PARAMS.ALLAN_VAR.H0./(2*tau);

% plot the allan variance parameters
plot_al_var = 1;
if plot_al_var
tau = 0.001:0.01:100;
tau=logspace(log10(tau(1)),log10(tau(end)),length(tau));

all_var = CLOCK_PARAMS.ALLAN_VAR.H_2*((2*pi)^2/6).*tau+CLOCK_PARAMS.ALLAN_VAR.H_1*...
   2*log(2)+CLOCK_PARAMS.ALLAN_VAR.H0./(2*tau);

all_var1 = CLOCK_PARAMS.ALLAN_VAR.H_2*((2*pi)^2/6).*tau;
all_var2 = CLOCK_PARAMS.ALLAN_VAR.H_1*2*log(2);
all_var3 = CLOCK_PARAMS.ALLAN_VAR.H0./(2*tau);

figure(60)
loglog(tau,sqrt(all_var1),tau,sqrt(all_var2),'r',tau,sqrt(all_var3),'g')
figure(61)
loglog(tau,sqrt(all_var))
end
```

sampled_recv_chan.m

```
function signal = sampled_recv_chan(time, Tp, PRN, clk_drift, clk_bias, phz_noise)
% usage: signal = sampled_recv_chan(time, Tp, PRN, clk_drift, clk_bias, phz_noise)
%
% This function generates the raw (sampled) signal for a given channel over
% the specified time vector.
%
% time      sampling times in receiver clock time     (nx1)
% Tp        propagation time for PRN (True_dist/c)    (nx1)         def: 0
% PRN       Satellite PRN number                      (int 1-37)    def: 1
%
% clk_drift vector of (Delta_Freq)/Freq values for determining frequency error
% clk_bias  vector of time biases for each sample time (time offsets from GPS time)
% phz_noise vector of radian phase errors (Phase Jitter)
%
% signal    output signal vector
```

```
%
% The propagation time includes the dyanmics of the system, as it represents
% the Line-of-sight (LOS) distance.  The change in this distance defines the
% LOS dynamics.

% if variables aren't entered, set the defaults:
if nargin<3
   disp('Not enough parameters specified to generate signal');
   return
end

global CARRIER_FREQUENCY
global IF_BEFORE_SAMP
global IF_AFTER_SAMP
global SAMPLING_FREQUENCY
global SAMPLING_DT
global T_C
global SPEED_OF_LIGHT
global START_TIME;
global CA_CODE;
global PRN_VEC;
global APPLY_DATA;
global DATA_BITS;
global APPLY_CA;
global ADD_OSC_ERRORS
% initialize working global variables (dynamic):
global PIT_CNT;
% Load global variables formerly initialized but not defined in load_global_params:
global DOPPLER_FREQ;
global RAD_TO_SV;
global DOPPLER_FREQ_TRUE
global RAD_TO_SV_TRUE
persistent DATA_MESG;
c = SPEED_OF_LIGHT;

% If first time through, initialize persistent variables (generate DATA sequences)
if (time(1)<(2*SAMPLING_DT))&(PRN==PRN_VEC(1))&(APPLY_DATA)
   % if first time through, generate a full set of data (1 superframe)
   % it has a duration of 12.5 Min @ 50 Hz, or 37500 bits.  (overkill at this
   % point, but if a valid data sequence existed, it could be
   % inserted here and the correct data modulation would be simulated)
   % for now, make the data the same for all satellites
   rand_data = rand(1,37500);
   rand_data(find(rand_data>0.5)) = -1;
   rand_data(find(abs(rand_data)<0.5)) = 1;
   % for now, make the data the same for all satellites
   DATA_MESG=rand_data;
end

% if clock errors are being modeled, insert them now into the Tp vector.  This allows
% for the sampling time and the output time to maintain its time step consistency,
% effectively becoming the 'Receiver Time' rather than the perfect GPS time.
if ADD_OSC_ERRORS
   % before adding in the oscillator errors, record the true Doppler and LOS radian
   % measurments--before "corrupting" the truth environment
   % Estimate the TRUE Received Doppler at the start of this correlation period
   DOPPLER_FREQ_TRUE(find(PRN==PRN_VEC))(PIT_CNT)=(Tp(1)-Tp(501))*...
      CARRIER_FREQUENCY*SAMPLING_FREQUENCY/500;
   %Record the initial phase of the received waveform (LOS dist. expressed in Radians)
   RAD_TO_SV_TRUE(find(PRN==PRN_VEC))(PIT_CNT) = (2*pi*CARRIER_FREQUENCY*Tp(1));
   % clk_bias is defined as "the advance of the recever clock with respect to true
   % GPS time"  See Kaplan Chptr 2.
   Tp = Tp+clk_bias;
   % NOTE: FROM NOW ON Tp IS CORRUPTED BY CLOCK ERRORS!!!
   % the frequency shifts are different if the signal is aliased--test for Nyquist
   if 2*IF_BEFORE_SAMP>SAMPLING_FREQUENCY  %if true, then aliasing occurs
      TFDS = (CARRIER_FREQUENCY+IF_AFTER_SAMP); %Total Frequency Down Shift--the
      %                          total change in frequency from received to final IF
   else  % no aliasing, the final IF after sampling is the same as before sampling
      TFDS = (CARRIER_FREQUENCY-IF_BEFORE_SAMP); %Total Frequency Down Shift
   end
   % Expression including clock error:  clk_bias
   TRUE_FEE_REF = (2*pi*(IF_BEFORE_SAMP-clk_drift*TFDS)).*time-...
      (2*pi*CARRIER_FREQUENCY.*Tp) ;%+ phz_noise;
   % equivlalent expression:
   % TRUE_FEE_REF = (2*pi*(-IF_AFTER_SAMP-clk_drift*TFDS)).*time-...
   %   (2*pi*CARRIER_FREQUENCY.*Tp);
else
```

```
   % both of the TRUE_FEE_REF statements below give the same result, due to aliasing
   TRUE_FEE_REF = (2*pi*IF_BEFORE_SAMP).*time-(2*pi*CARRIER_FREQUENCY.*Tp);
   %%% Second TRUE_FEE_REF statement
   %TRUE_FEE_REF = (-2*pi*IF_AFTER_SAMP).*time-(2*pi*CARRIER_FREQUENCY.*Tp);
end
% Estimate the initial Received Doppler for the start of this correlation period
DOPPLER_FREQ{find(PRN==PRN_VEC)}(PIT_CNT)=(Tp(1)-Tp(501))*CARRIER_FREQUENCY*...
   SAMPLING_FREQUENCY/500;
%Record the initial phase of the received waveform (LOS dist. expressed in Radians)
RAD_TO_SV{find(PRN==PRN_VEC)}(PIT_CNT) = (2*pi*CARRIER_FREQUENCY*Tp(1));
amplitude=calc_sig_amp(find(PRN==PRN_VEC));
% Raw CA code carrier
signal = amplitude*cos(TRUE_FEE_REF);

if APPLY_CA
   %-------------------------------------------------------------------------------
   %  Apply CA code
   %-------------------------------------------------------------------------------

   % get the times in terms of SV time trunkated to the nearest millisecond
   time_dyn = mod((time-Tp), 0.001);
   chip_num = fix(time_dyn./T_C) +1;
   % chip_num now contains the correct chip number for each sample time

   % Sample the code at the same sampling frequency for the same number of samples
   % (Determine the actual code over all time)
   tcode = CA_CODE{PRN}(chip_num);

   % Apply the code to the carrier
   signal = signal.*tcode;
end

if APPLY_DATA
   %-------------------------------------------------------------------------------
   %  Apply Data Modulation
   %-------------------------------------------------------------------------------
   % determine the current bit index based on the START_TIME and current simulation
   % time for each sample time.  The data "repeats" every 12.5 minutes.
   if (fix(mod(START_TIME+(time(1)-Tp(1)),750)*50)+1)~=(fix(mod(START_TIME+...
         (time(end)-Tp(end)),750)*50)+1);
      data_bit = fix(mod(START_TIME+(time-Tp), [repmat(750,1,length(time))])*50)+1;
   else
      data_bit = repmat(fix(mod(START_TIME+(time(1)-Tp(1)),750)*50)+1,1,length(time));
   end

   % data_bit now contains the correct data bit index (inside a superframe) for
   % each sample time

   %-------------------------------------------------------------------------------
   % save off the "bit #", the index inside a superframe of the NEWEST bit observed
   % during this time interval
   %-------------------------------------------------------------------------------
   DATA_BITS{find(PRN==PRN_VEC)}(PIT_CNT,1) = data_bit(end);
   % save off the NEWEST bit value
   DATA_BITS{find(PRN==PRN_VEC)}(PIT_CNT,2) = DATA_MESG(data_bit(end));
   if sum(diff(data_bit))
      % get the first sample time at which the bit transistion occurs, i.e. the
      % first sample that "sees" the new bit value.  This time is the receiver
      % time at which the data transistion is OBSERVED
      DATA_BITS{find(PRN==PRN_VEC)}(PIT_CNT,3)=time((find(1==diff(data_bit))+1));
      % Sample the data at the same sampling frequency for the same number of
      % samples (Determine the actual data over all time)
      tdata = DATA_MESG(data_bit);
      %-------------------------------------------------------------------------------
      %  Modulate the signal
      %-------------------------------------------------------------------------------
      signal = signal.*tdata;
   else %no bit transition has occured
      DATA_BITS{find(PRN==PRN_VEC)}(PIT_CNT,3) = nan;
      % if no data bit has occured, implement faster "modulation"
      if DATA_MESG(data_bit(end))==(-1)
         signal = -signal;
      end
      % if the data bit has not changed and it is a "1", do nothing to the signal
   end
end
```

calc_sig_amp.m

```
function amplitude=calc_sig_amp(CHAN_VEC)
% usage: amplitude=calc_sig_amp(CHAN_VEC)
%
% This function returns the amplitude for the PRN listed in the function call
% according to the SNR_ratio & Noise power parameters set in the load_global_params
% file.  SNR_ratio is to be expressed in Db.

global SAMPLING_FREQUENCY
global CHIPPING_RATE
global NOISE_POWER
global FILTER_SIG
global SNR_DB

if nargin==1
   % Calculate signal power
   snr_ratio=10^(SNR_DB(CHAN_VEC)./10);
   Ps=NOISE_POWER*snr_ratio;
   if FILTER_SIG
       % kick up the power a little more to compensate for the filtered components.
      % Note: the main lobe contains 90.3% of the signal power, so to maintain the
      % same signal power specified by the SNR, the power must be increased here.
      Ps=Ps/.94;
   end
   amplitude=sqrt(Ps*2);
elseif nargin==0
   % if no input arguments are specified, return all the amplitudes for the signals
   % specifed by the global variables.  Take the noise powers from the SNR_DB variable
   global PRN_VEC
   snr_ratios=10.^(SNR_DB(length(PRN_VEC))./10);
   Pwr_sigs=NOISE_POWER.*snr_ratios;
   if FILTER_SIG
      % kick up the power a little more to compensate for the filtered components.
    % Note: the main lobe contains 90.3% of the signal power, so to maintain the
      % same signal power specified by the SNR, the power must be increased here.
      Pwr_sigs=Pwr_sigs./0.94;
   end
   amplitude=sqrt(Pwr_sigs*2);
elseif nargin>1
   disp('Error: too many input arguments, amplitude set to zero')
   amplitude=0;
end
```

quantizer.m

```
function sig_quant=quantizer(Sig_in)
% This function quantizes the I and Q values.  It's currently using 2-bit
% quantization, and assumes that there is an automatic gain control (AGC)
% that is making sure the gain is correct.  The AGC implements a 30%
% occurance of 3's & a 70% occurance of 1's.

global PIT_CNT
global ML_QUANT_THRESH
global SAVE_QUANT_THRESH

cutoff70=std(Sig_in)*(70/68.3); % approximation to get cutoffs to include 1.7%
                                % more points than that included within one
                                % Standard Deviation.
%scaled_Sig_in=Sig_in*1.85/sigma;

% set a default distribution error
dist_err = 0.01;
% set the gain
gain = cutoff70;
% use an iterative technique to arrive at 30%/70% distrbution within 0.1% error
while abs(dist_err) > 0.001
scaled_Sig_in=Sig_in;
minus3=find(scaled_Sig_in<=-cutoff70);
minus1=find(scaled_Sig_in>-cutoff70&scaled_Sig_in<=0);
plus1=find(scaled_Sig_in>0&scaled_Sig_in<cutoff70);
```

```
plus3=find(scaled_Sig_in>=cutoff70);

sig_quant=zeros(size(Sig_in));
sig_quant(minus3)=-3;
sig_quant(minus1)=-1;
sig_quant(plus1)=1;
sig_quant(plus3)=3;

dist1_3 = ([1 0 0 1;0 1 1 0]*(hist(sig_quant,[-3 -1 1 3])/length(sig_quant))');
% a positive dist_err here means the cutoff is too low, add to cutoff70
dist_err = dist1_3(1)-0.3;
if SAVE_QUANT_THRESH
   ML_QUANT_THRESH(PIT_CNT) = cutoff70;
end
cutoff70=cutoff70+gain*dist_err;
end


% For a Gaussian density, 68.3% of the points are within 1 standard deviation.  To
% get a 70% boundry for our AGC, we scale the distribution

% P. 354, BLUE BOOK:   talks about the AGC in the GEC/Plessy system, comparing it to
% the MAGR
```

APPENDIX C Source Code for Analytical I and Q Model

gps_iq_model.m

```
function [argout1,argout2,argout3]=gps_iq_model(task,argin1,argin2,argin3,argin4)
% usage: [argout1,argout2,argout3]=gps_iq_model(task,argin1,argin2,argin3,argin4)
%
% This program is a GPS receiver model which generates In-Phase (I) and Quadrature-
% Phase (Q) samples of the received signal.  These I's & Q's are demodulated and
% accumlated according to DCO values and latch intervals controlled by the user.
%
% The program has several modes of operation, selected by specifying the "task"
% to be performed as a string in the first input argument.  The input and
% output arguments are then specifed for a given task.  The four tasks currently
% implemented are initialization (init), restart at a particular time (rstrt),
% DCO updating (updtdco), and I & Q generation (geniq).  Their input/output
% arguments are listed below:
%
% task:  'init'
%|-------------------------------------------------------------------------
%  INPUTS:
%
%  argin1     eph_filename    filename of the ephemeris file, needed for SV positions
%  argin2     recv_pos_mat    the receiver position file, including time. structure:
%                             col/row 1 = time (if not starting at zero, in GPS time)
%                             col/row 2 = X ECEF position
%                             col/row 3 = Y ECEF position
%                             col/row 4 = Z ECEF position
%  argin3     START_TIME      GPS start time, if the time in the recv_pos_mat file is
%                             zero.  If the first time in the recv_pos_mat file is
%                             not zero, this input will be ignored
%  argin4     elev_cutoff     cutoff angle above the horizion for determining satelite
%                             visibility.  If not provided, 5 degrees is used
%  OUTPUTS:
%
%  argout1    sv_pos_mat      matrix of satelite positions for all satellites visible
%                             during the entire run at each timestep.
%|-------------------------------------------------------------------------
% task:  'rstrt'
%|-------------------------------------------------------------------------
%  INPUTS:
%
%  argin1     init_time       Time at which to initialize the I & Q generator, loading
%                             the initial errors specified in iq_model_globals
%  OUTPUTS:
%
%  argout1    doppler_mat     Matrix of Doppler frequencies for each channel @ init_time
%|-------------------------------------------------------------------------
% task:  'updtdco'
%|-------------------------------------------------------------------------
%  argin1     car_dco_corr    Carrier DCO correction in Hz.
%  argin2     code_dco_corr   Code DCO correction in Hz.
%  argin3     UPDT_TIME       simulation time of DCO update
%|-------------------------------------------------------------------------
% task:  'geniq'
%|-------------------------------------------------------------------------
%  argin1     LATCH_TIME  Simulation time (either GPS or offset from 0 start time)
%  argout1    EPL_IQ      Cell Array of Early, Prompt, & Late I & Q values
%                         For EPL_IQ_MAT{chan#}:
%                                 Col1     Col2     Col3
%                               ------------------------
%                         Row 1 |  I_E   |  I_P   |  I_L |
%                               ------------------------
%                         Row 2 |  Q_E   |  Q_P   |  Q_L |
%                               ------------------------
%
%  argout2                Vector of pseudorange measurements
%  argout3    CP_COUNT    vector of carrier-phase measurements
%
%  Alternate output format:  if the GENIQ_MATOUT bit in the IQ_model_globals
%                            function is set, the output will have the following
```

```
%|                          format:
%|
%|  argout1   MOD_OUTPT  Row vector containing earily, prompt, & late I & Q, PR, & CP
%|
%|                      Channel 1                    Channel 2
%|           /-----------------------------\/------------------- . . . . . ---\
%|            C1  C2  C3  C4  C5  C6  C7  C8  C9  C10 C11 C12 C13 . . . . . CXX
%|            ---------------------------------------------------------------------
%|            I_E Q_E I_P Q_P I_L Q_L PR  CP  I_E Q_E I_P Q_P I_L . . . . .  CP
%|
%|-------------------------------------------------------------------------------
%
% The file is further controlled by the global variables found in IQ_model_globals
% This file must accompany this one.  The function uses a collection of functions,
% some included at the bottom of the main routine in the "Utilities" section.  The
% functions used in this program are:
%
% ------------------------------------------------------------------------
% gps_iq_model
%             \
% |            IQ_model_globals (external)
%       {init}\
% |            calc_all_svs_pos (internal)
%              |               \
% |            |                load_eph_file   (external)
%              |                get_valid_prns  (internal)
% |                                           \
%              |                |              ecef2lla           (external)
% |                              |              calc_sv_pos        (external)
%              |                |              sv_elevation       (external)
% |                              |              current_ephemeris  (external)
%              |                calc_sv_pos_mat (external)
% |    {rstrt}\
%              IQ_model_globals (external)
% |  {updtdco}\
%              IQ_model_globals (external)
% |    {geniq}\
%              IQ_model_globals (external)
% |            gen_IQ (internal)
%              R      (internal)
% ------------------------------------------------------------------------
%
% Written By Phillip M. Corbell
% Last Update:  8 Februrary 2000

% example call statement:
%     gps_iq_model('init','lab2.eph',lab2,399100)

%persistent LOS_DIST_MAT
%persistent DOPPLER_FREQ_MAT
%persistent TIME_OFF
%tic
global LOS_DIST_MAT
global DOPPLER_FREQ_MAT
global TIME_OFF
persistent CP_COUNT;  %Carrier-phase counter--running carrier-phase total
global LOOKUP  % inverse mapping of PRN vec, PRNS to indicees

switch lower(task)

case {'init','rstrt'}
   %-------------------------------------------------------------------------------
   % at either initialization or a restart at a different time: (see after individual
   % init and rstrt functions for more common code)
   %-------------------------------------------------------------------------------
   % do some variable managment
   global DEFINE_GLOBALS
   DEFINE_GLOBALS=1;
   iq_model_globals
   % now that globals have been defined, turn that switch off!
   DEFINE_GLOBALS=0;
   % initialize carrier-phase counter to zero.
   CP_COUNT = 0;
switch lower(task)
   case 'init'
      %-------------------------------------------------------------------------------
      % initialize:
      %-------------------------------------------------------------------------------
```

```
% argin1  eph_filename  filename of the ephemeris file, needed for SV positions
% argin2  recv_pos_mat  the receiver position file, including time. structure:
%                       col/row 1 = time (if not starting at zero, in GPS time)
%                       col/row 2 = X ECEF position
%                       col/row 3 = Y ECEF position
%                       col/row 4 = Z ECEF position
% argin3  START_TIME    GPS start time, if the time in the recv_pos_mat file is
%                       zero.  If the first time in the recv_pos_mat file is
%                       not zero, this input will be ignored
% argin4  elev_cutoff   cutoff angle above the horizion for determining satelite
%                       visibility.  If not provided, 5 degrees is used
%
% argout1 sv_pos_mat    matrix of satelite positions for all satellites visible
%                       during the entire run at each timestep.
%
%-------------------------------------------------------------------------------
% in the initialization, clear persistent/global values from previous runs
LOS_DIST_MAT=[];
DOPPLER_FREQ_MAT=[];
LOS_RANGES_MAT=[];
LAST_LATCH_TIME=[];
TIME_OFF=[];

global EPHEMERIS
c=SPEED_OF_LIGHT; %shorthand notation

% assign input variable names
eph_filename = argin1; %string, name of the *.eph file in "Raquet *.eph Format"
sz=size(argin2);  %the following takes the input traj. in either orientation
if sz(1)<sz(2)    %and rotates it so that the time is the first column.
   recv_pos_mat = argin2'; %ECEF coord. for the receiver over the entire run
else
   recv_pos_mat = argin2; %ECEF coord. for the receiver over the entire run
end
if recv_pos_mat(1,1)==0;  %if the time starts from zero, get the GPS start time
   if (nargin<3)|isempty(argin3)|(argin3>604800)
      disp('Error:  GPS start time not specified');
   end
   START_TIME = argin3;  %start time in GPS seconds
   time = recv_pos_mat(:,1)+START_TIME; %vector of all times in GPS seconds
   END_TIME = recv_pos_mat(end,1); %assumes column one is time
   TIME_OFF = START_TIME;
else % if not, assume the time vector in the trajectory is GPS time
   START_TIME = recv_pos_mat(1,1);
   END_TIME = recv_pos_mat(end,1);
   time = recv_pos_mat(:,1); %vector of all times in GPS seconds
   TIME_OFF = 0;
end
if nargin<5
   elev_cutoff = 5; %degrees ELEVATION CUTOFF HARD CODED HERE (if not given)
else
   elev_cutoff = argin4;
end
LATCH_TIME=START_TIME;%initialize "Latch time"--a latch doesn't actually occur
% Calculate SV positions for SV's visible during entire run for all timesteps
[sv_pos_mat,PRN_LST]=calc_all_svs_pos(eph_filename,...
   recv_pos_mat,time,elev_cutoff);
% PRN_LST contains the PRN numbers of the SV positions in sv_pos_mat
% Create lookup variable to do inverse mapping of PRNs to indicees
LOOKUP(PRN_LST)=1:length(PRN_LST);
% Assign the first output variable, argout1, the matrix of satellite positions
argout1=sv_pos_mat;
% The number of PRN's will be used alot, make it a variable
num_prns=length(PRN_LST);
%create LOS TRUE RANGES for each SV at each time in the receiver position
LOS_DIST_MAT=time; %(column one will be times of the receiver trajectory)
for i=1:num_prns
   LOS_DIST_MAT(:,i+1)=(sum(((recv_pos_mat(:,2:4)-...
      sv_pos_mat(i:num_prns:end,3:5)).^2)').^0.5)';
end
% LOS_DIST_MAT is has length(PRN_LST)+1 columns and length(time) rows.  The col.
% contain the LOS distance between each respective satellite (indexed according
% to the PRN_LST) and the receiver at the GPS time specified in the first col.
% each row then contains the "perfect pseudoranges" for each timestep.

%now that we know the number of PRNs we're dealing with, we need to size our
%global variables to match:
CAR_DCO_PHASE_ERR=CAR_DCO_PHASE_ERR(1:num_prns);
```

```
      TAU=TAU(1:num_prns);
      CAR_DCO_FREQ_ERR=CAR_DCO_FREQ_ERR(1:num_prns);
      CODE_DCO_FREQ_ERR=CODE_DCO_FREQ_ERR(1:num_prns);
      SNR_DB=SNR_DB(1:num_prns);
      %get the initial LOS distances and doppler values at the start time.
      delta_time = 0.1e-3; % a tenth of a millisecond average time
      time_DT = [LOS_DIST_MAT(1,1) LOS_DIST_MAT(1,1)+delta_time];
      %-------------- END CASE INIT ------------------------------------------------
   case 'rstrt'
      %-------------------------------------------------------------------------------
      % restart at a given time:
      %-------------------------------------------------------------------------------
      % INPUTS:
      %
      % argin1   init_time   Time at which to initialize the I & Q generator, loading
      %                      the initial errors specified in iq_model_globals
      %
      % OUTPUTS:
      %
      % argout1  doppler_mat  Matrix of Doppler frequencies for each channel @
      %                       init_time
      %
      %-------------------------------------------------------------------------------
      % assign input arguments:
      init_time=argin1 + TIME_OFF;
      if isempty(LOS_DIST_MAT)
         disp('Line of Site matrix empty.  Run init to re-create')
         return
      end
      % make sure init time is within the time of the run
      if init_time>LOS_DIST_MAT(end,1)|init_time<LOS_DIST_MAT(1,1)
         disp('Error:  initalization time outside of run time window');
         argout1 = 0;
         return
      end
      c=SPEED_OF_LIGHT; %shorthand notation (handy)
      % clear values assocated with some other time, to reinitalize to the new time
      DOPPLER_FREQ_MAT=[];
      LAST_LATCH_TIME=[];
      % create a variable
      num_prns=length(PRN_LST);
      % Create lookup variable to do inverse mapping of PRNs to indicees
      LOOKUP(PRN_LST)=1:length(PRN_LST);
      %now that we know the number of PRNs we're dealing with, we need to size our
      %global variables to match:
      CAR_DCO_PHASE_ERR=CAR_DCO_PHASE_ERR(1:num_prns);
      TAU=TAU(1:num_prns);
      CAR_DCO_FREQ_ERR=CAR_DCO_FREQ_ERR(1:num_prns);
      CODE_DCO_FREQ_ERR=CODE_DCO_FREQ_ERR(1:num_prns);
      SNR_DB=SNR_DB(1:num_prns);

      %get the initial LOS distances and doppler values at the init time.
      delta_time = 0.1e-3; % a tenth of a millisecond average time
      time_DT = [init_time init_time+delta_time];
      LATCH_TIME=init_time;%initialize "Latch time"--a latch doesn't actually occur
      %-------------- END CASE RSTRT -----------------------------------------------
   end
   % now run more code common to both init and rstrt
   LOS_distances=spline(LOS_DIST_MAT(:,1)',LOS_DIST_MAT(:,2:end)',time_DT)';
   Delta_ranges=diff(LOS_distances);
   % Delta_ranges now contains the change in LOS range for each PRN over the time
   % period delta_time
   DOPPLER_FREQ_MAT = -(Delta_ranges./delta_time)./CARRIER_WAVELENGTH; %row vector
   LOS_RANGES_MAT = LOS_distances(1,:); % (m) row vector
   % initialize DCO values to the correct doppler plus inital errors.
   CAR_DCO_FREQ_MAT = DOPPLER_FREQ_MAT + CAR_DCO_FREQ_ERR; % (Hz) row vector
   CODE_DCO_FREQ_MAT = DOPPLER_FREQ_MAT + CODE_DCO_FREQ_ERR; % (Hz) row vector
   CAR_DCO_PHZ_MAT = 2*pi*LOS_RANGES_MAT./CARRIER_WAVELENGTH + CAR_DCO_PHASE_ERR;%rad
   CODE_DCO_DELAY_MAT = LOS_RANGES_MAT./c + TAU; % (s) row vector
   LAST_LATCH_TIME = LATCH_TIME;
   switch lower(task)
   case 'rstrt'
      % assign the output variable
      argout1 = DOPPLER_FREQ_MAT;
   end
   %-------------- END CASE INIT AND RSTRT ---------------------------------------

case {'updtdco','geniq'}
```

```
%-------------------------------------------------------------------------------
% at either a dco update or a generate IQ task:
%-------------------------------------------------------------------------------
iq_model_globals
c=SPEED_OF_LIGHT; %shorthand notation (handy)
if isempty(LOS_DIST_MAT)
   disp('Line of Site matrix empty.  Run init to re-create')
   return
end

switch lower(task)
%-------------- END CASE UPDTDCO OR GENIQ ----------------------------------------

case 'updtdco'
   %-------------------------------------------------------------------------------
   % update DCO values
   %-------------------------------------------------------------------------------
   % argin1    car_dco_corr   Carrier DCO correction in Hz.
   % argin2    code_dco_corr  Code DCO correction in Hz.
   % argin3    UPDT_TIME      simulation time of DCO update
   %-------------------------------------------------------------------------------
   car_dco_corr = argin1;
   code_dco_corr= argin2;
   UPDT_TIME   =  argin3+TIME_OFF;
   % make sure the user is not trying to go backwards in time!
   if UPDT_TIME<=LAST_LATCH_TIME
      disp('Error:  This Time input equals or preceeds the last time input')
      argout1 = [];
      return
   end
   % update the internal carrier phase and code delay values before updating DCO
   delta_t = LAST_LATCH_TIME - UPDT_TIME;
   CAR_DCO_PHZ_MAT = CAR_DCO_PHZ_MAT - 2*pi*CAR_DCO_FREQ_MAT.*delta_t; %rads
   CODE_DCO_DELAY_MAT = CODE_DCO_DELAY_MAT - (CODE_DCO_FREQ_MAT.*delta_t)./...
      CARRIER_FREQUENCY;  %seconds
   % update the DCO values
   CAR_DCO_FREQ_MAT = CAR_DCO_FREQ_MAT + car_dco_corr;
   CODE_DCO_FREQ_MAT = CODE_DCO_FREQ_MAT + code_dco_corr;
   % set last latch time here to the update time, so the same interval is not
   % integrated twice.
   LAST_LATCH_TIME = UPDT_TIME;
   %-------------- END CASE UPDTDCO -------------------------------------------

case 'geniq'
   %-------------------------------------------------------------------------------
   % dump I & Q values
   %-------------------------------------------------------------------------------
   % argin1    LATCH_TIME  Simulation time (either GPS or offset from 0 start time)
   % argin2    PRN_ASIGN   Vector of PRNS that specifies the assignments of
   %                       the PRNs to the 12 channels of the simulator.  The
   %                       index of the numbers in PRN_ASIGN correspond to the
   %                       channel, and the numbers at the 12 locations in the
   %                       PRN_ASIGN vector specify the PRN assigned to that
   %                       channel.
   %
   % argout1   EPL_IQ      Cell Array of Early, Prompt, & Late I & Q values
   %                       For EPL_IQ_MAT{chan#}:
   %                                Col1     Col2     Col3
   %                       Row 1    I_E      I_P      I_L
   %                       Row 2    Q_E      Q_P      Q_L
   %
   %
   % argout2               Vector of pseudorange measurements
   % argout3   CP_COUNT    vector of carrier-phase measurements
   %
   % Alternate output format:  if the GENIQ_MATOUT bit in the IQ_model_globals
   %                           function is set, the output will have the following
   %                           format:
   %
   % argout1   MOD_OUTPUT  Row vector containing earily, prompt, & late I & Q, PR,
   %                       & CP
   %
   %              Channel 1                   Channel 2
   %          /-----------------------------\/------------------- . . . . . ---\
   %           C1  C2  C3  C4  C5  C6  C7  C8  C9  C10 C11 C12 C13 . . . . . CXX
   %           ---------------------------------------------------------------
   %           I_E Q_E I_P Q_P I_L Q_L PR  CP  I_E Q_E I_P Q_P I_L . . . . .  CP
   %
```

```
%-------------------------------------------------------------------------
% assign input arguments
if nargin<2
   PRN_ASIGN = PRN_LST;
else
   PRN_ASIGN = argin2;
end
LATCH_TIME = argin1 + TIME_OFF;
% make sure the user is not trying to go backwards in time!
if LATCH_TIME<=LAST_LATCH_TIME
   disp('Error:  This Time input equals or preceeds the last time input')
   argout1 = [];
   return
end

% get the time lapse since the last call
delta_t = LATCH_TIME - LAST_LATCH_TIME;

% now get the TRUE LOS distances and signal DOPPLER values:

% use a tenth of a millisecond average time to calculate actual doppler
delta_time = 0.1e-3;
% need to calculate the phase error at the beginning of the integration interval
% therefore, get the truth Doppler values at that time
time_DT = [LAST_LATCH_TIME LAST_LATCH_TIME+delta_time];
LOS_distances=spline(LOS_DIST_MAT(:,1)',LOS_DIST_MAT(:,2:end)',time_DT)';
Delta_ranges=diff(LOS_distances);
% Delta_ranges now contains the change in LOS range for each PRN over the time
% period delta_time (one millisecond) just after LAST_LATCH_TIME
DOPPLER_FREQ_MAT = -(Delta_ranges./delta_time)./CARRIER_WAVELENGTH;
% create the truth LOS ranges matrix, at the beginning of the integration
% interval (starting at LAST_LATCH_TIME)
LOS_RANGES_MAT = LOS_distances(1,:); % m
% Calculate the phase/time errors:
CAR_DCO_PHASE_ERR = 2*pi*LOS_RANGES_MAT/CARRIER_WAVELENGTH - CAR_DCO_PHZ_MAT;
% TAU is the time error in the autocorrelation function, or code misallignment
% error between the local and received code waveform. (in seconds)
TAU = LOS_RANGES_MAT./c - CODE_DCO_DELAY_MAT;
% Calculate the frequency errors:
CAR_DCO_FREQ_ERR = DOPPLER_FREQ_MAT - CAR_DCO_FREQ_MAT;
CODE_DCO_FREQ_ERR = DOPPLER_FREQ_MAT - CODE_DCO_FREQ_MAT;
%%%%%%%%%%------------ PERFECT WORLD CHEAT ----------------%%%%%%%%%%%%
perfect_world=0;  %set this to 1 to allow perfect DCO Syncronization Each Time
if perfect_world
   CAR_DCO_FREQ_MAT = DOPPLER_FREQ_MAT;
   CODE_DCO_FREQ_MAT = DOPPLER_FREQ_MAT;
end
%%%%%%%%%%---------- END PERFECT WORLD CHEAT ----------------%%%%%%%%%%%%
% get the IQ values, w/o code correlation yet.
delta_t_vec = delta_t*ones(size(CAR_DCO_FREQ_ERR));
[I,Q] = gen_IQ(CAR_DCO_PHASE_ERR, CAR_DCO_FREQ_ERR, delta_t_vec, SNR_DB);
% I & Q contain the I & Q values for each LOS path, now scale them with the
% autocorrelation function. (I and Q are row vectors)
%apply Code correlation
I_Q_E = [I;Q].*(ones(2,1)*R(TAU,'e')); %I and Q are row vectors, and R returns
I_Q_P = [I;Q].*(ones(2,1)*R(TAU,'p')); %a column vector the same length as I or
I_Q_L = [I;Q].*(ones(2,1)*R(TAU,'l')); %Q.
sz=size(I_Q_E);
if sz(1)>sz(2)
   I_Q_E=I_Q_E';
   I_Q_P=I_Q_P';
   I_Q_L=I_Q_L';
end
% the above rotation ensures the I_Q_x paramters are 2x(num of sv's)
global NO_NOISE
global SAMPLING_FREQUENCY
I_or_Q_sqrd_var = 8.5;
noise_std = sqrt(I_or_Q_sqrd_var*SAMPLING_FREQUENCY.*delta_t_vec);
if ~NO_NOISE
    I_Q_E=I_Q_E+repmat(noise_std,2,1).*randn(size(repmat(noise_std,2,1)));
    I_Q_P=I_Q_P+repmat(noise_std,2,1).*randn(size(repmat(noise_std,2,1)));
    I_Q_L=I_Q_L+repmat(noise_std,2,1).*randn(size(repmat(noise_std,2,1)));
end

CP_COUNT = CP_COUNT - 2*pi*CAR_DCO_FREQ_MAT.*delta_t; %rads
% update the internal carrier phase and code delay values (since last update)
CAR_DCO_PHZ_MAT = CAR_DCO_PHZ_MAT - 2*pi*CAR_DCO_FREQ_MAT.*delta_t; %rads
CODE_DCO_DELAY_MAT = CODE_DCO_DELAY_MAT - (CODE_DCO_FREQ_MAT.*delta_t)./...
```

```
      CARRIER_FREQUENCY;  %seconds
   % CAR_DCO_PHZ_MAT now contains the expected LOS radians between each SV and the
   % receiver.  CODE_DCO_DELAY_MAT contains the expected signal travel times bet.
   % each SV and the Receiver.

   if ~MOD_OUTPUT
      for i=LOOKUP(PRN_ASIGN)  %1:length(I_Q_E(1,:))
         EPL_IQ{i} = round([I_Q_E(:,i) I_Q_P(:,i) I_Q_L(:,i)]);
      end
      argout1 = EPL_IQ;
      % assign carrier-phase and pseudorange output variables
      argout2 = CODE_DCO_DELAY_MAT(LOOKUP(PRN_ASIGN)).*c; %pseudorange measurement
      argout3 = CP_COUNT(LOOKUP(PRN_ASIGN));              %carrier-phase measurement
   else
      I_Q_E = I_Q_E(:,LOOKUP(PRN_ASIGN));
      I_Q_P = I_Q_P(:,LOOKUP(PRN_ASIGN));
      I_Q_L = I_Q_L(:,LOOKUP(PRN_ASIGN));
      TEMP1 = round([I_Q_E;I_Q_P;I_Q_L]);
      TEMP2 = [CODE_DCO_DELAY_MAT(LOOKUP(PRN_ASIGN)).*c;...
            CP_COUNT(LOOKUP(PRN_ASIGN))/(2*pi)];
      TEMP3 = [TEMP1;TEMP2];
      argout1 = TEMP3(:)';
%      argout1(1:8:num_prns) = I_Q_E(1,:);
%      argout1(2:8:num_prns) = I_Q_E(2,:);
%      argout1(3:8:num_prns) = I_Q_P(1,:);
%      argout1(4:8:num_prns) = I_Q_P(2,:);
%      argout1(5:8:num_prns) = I_Q_L(1,:);
%      argout1(6:8:num_prns) = I_Q_L(2,:);
%      argout1(7:8:num_prns) = CODE_DCO_DELAY_MAT.*c;
%      argout1(8:8:num_prns) = CP_COUNT;
   end


   % update last latch time parameter
   LAST_LATCH_TIME = LATCH_TIME;

   %-------------- END CASE GENIQ ----------------------------------------------
end

otherwise
   disp('unknown task')

end

%toc

%--------------------------------------------------------------------------------
% UTILITY FUNCTIONS:
%--------------------------------------------------------------------------------

%--------------------------------------------------------------------------------
function [sv_pos_mat,PRN_VEC]=calc_all_svs_pos(ephem_filename,...
   recv_pos_mat,time,elev_cutoff)
%
%
%
global EPHEMERIS
global SPEED_OF_LIGHT
c=SPEED_OF_LIGHT;

% Load the ephemerous parameters
load_eph_file(ephem_filename);
% Get a list of valid prns that are visible during the entire run
PRN_VEC = get_valid_prns(recv_pos_mat([1 end],:), elev_cutoff, time);
num_prns = length(PRN_VEC);
% Calculate satellite positions for all timesteps
% form a matrix of SV ECEF positions:
% col 1: Time, Col 2: PRN, Col 3: X cordECEF, Col 4: Y cordECEF, Col 5: Z cord ECEF
% first create the first two columns, time & PRN
sv_pos_mat=[];
col1 = ones(num_prns,1)*recv_pos_mat(:,1)';
col2 = PRN_VEC'*ones(1,length(recv_pos_mat(:,1)));
sv_pos_mat(:,1:2) = [col1(:) col2(:)];
true_ranges=sv_pos_mat;
% now complile the satellite positions for every row in sv_pos_mat
% note that we are calculating the SV positions for where they WERE at the time of
% transmission, not where they all ARE at a given GPS time.  In doing this, we must
% interatively solve for the transmission time.
```

```
for i=1:(length(time));
   % start off with the transmit times being the true receiver time of reception
   strt = (1+(i-1)*num_prns); stp = (num_prns+(i-1)*num_prns);
   transmit_time_vec=ones(num_prns,1)*(time(i)-0.07); %0.07 is a nominal prop. time
   true_recv_pos = recv_pos_mat(i,2:4);
   max_LOS_err = 41;
   while max_LOS_err>1;  % specify accuracy here in meters
      sv_pos_mat(strt:stp,3:5)=...
         calc_sv_pos_mat(PRN_VEC,transmit_time_vec,true_recv_pos);
      % now calculate the transmit times from the initial positions
      prop_times = (((sum(((sv_pos_mat(strt:stp,3:5)-...
         ones(num_prns,1)*true_recv_pos).^2)')')').^0.5)./c;
      new_transmit_time_vec = ones(num_prns,1)*time(i)-prop_times;
      % see if we've converged:
      max_LOS_err=max(abs(transmit_time_vec-new_transmit_time_vec))*c;
      transmit_time_vec=new_transmit_time_vec;
      true_ranges(strt:stp,3)=prop_times*c;
   end
   %occasionally output progress
   if mod(i,round(length(time)/4))==1;
      endofmes = '% done calculating SV positions';
      startofmes = num2str(100*i/length(time),3);
      disp([startofmes endofmes]);
   end
end
%-------------------------------------------------------------------------------

%-------------------------------------------------------------------------------
function prn_vec = get_valid_prns(srt_stp_time_pos, elev_cutoff, time_vec)
% returns the prns that are visible both from the starting location at the start time
% and from the ending location at the end time

global EPHEMERIS

time=[time_vec(1) time_vec(end)];  %start and stop times of the run in GPS seconds
% get list of valid PRN's in the Ephemeris file
prns = [];
for i=1:32
   if EPHEMERIS{i}.valid==1
      prns=[prns i];
   end
end
elvmat=zeros(length(time),length(prns));
% for the start position, determine the SV's above the cutoff angle
recv_pos_ECEF = srt_stp_time_pos(1,2:4);
[long, lati]=ecef2lla(recv_pos_ECEF);
for (i=prns)
   for (t=time)
      sv_pos = calc_sv_pos(i, t, recv_pos_ECEF);
      elvmat(find(time==t), find(i==prns)) = (180/pi)*sv_elevation(lati,...
         long, recv_pos_ECEF, sv_pos);
   end
end
% now find those PRNs that are above the cutoff
prnsvis = [];
elvmatvis = [];
for i=prns
   abvhor=find(elvmat(:,find(i==prns))>elev_cutoff); %SV's above the horizion
   if ~isempty(abvhor)
      prnsvis = [prnsvis i];
      elvmatvis = [elvmatvis elvmat(:,find(i==prns))];
   end
end
% for the end position, determine the SV's above the cutoff angle
recv_pos_ECEF = srt_stp_time_pos(end,2:4);
[long, lati]=ecef2lla(recv_pos_ECEF);
for (i=prns)
   for (t=time)
      sv_pos = calc_sv_pos(i, t, recv_pos_ECEF);
      elvmat(find(time==t), find(i==prns)) = (180/pi)*sv_elevation(lati,...
         long, recv_pos_ECEF, sv_pos);
   end
end
% now find those PRNs that are above the cutoff
prnsvis_end = [];
elvmatvis_end = [];
for i=prns
   abvhor=find(elvmat(:,find(i==prns))>elev_cutoff); %SV's above the horizion
```

```
   if ~isempty(abvhor)
      prnsvis_end = [prnsvis_end i];
      elvmatvis_end = [elvmatvis_end elvmat(:,find(i==prns))];
   end
end
% now find those PRNs that are above the cutoff at the end and beginning of the run
vec1(prnsvis)=1;
vec2(prnsvis_end)=1;
vec3=vec1&vec2;
PRN_VEC=find(vec3==1);
% now make sure the ephemeris for these satelites is valid over this time span
for j=PRN_VEC
   eph=current_ephemeris(j);
   toe_value(j)=eph.t0e;
end
%returns the indexes (PRN numbers) of the toe_value vector which has toe values
% within 2 hours of the start and stop times of the run
prn_vec=find((abs(toe_value-time(1))<2*60*60)&(abs(toe_value-time(end))<2*60*60));

% End Function
%-------------------------------------------------------------------------------

%-------------------------------------------------------------------------------
function [arg1,arg2,arg3]=R(tau_err,EPL_switch)
% usage: []=R(tau_err,EPL_switch)
% tau_err is the offset (in seconds) of the code tracking loop (in time) It is
%     equivalent to the absolute phase error in cycles, but expressed in seconds
% if EPL_switch isn't specified, arg1,arg2,arg3 =  E_value,P_value,L_value
% if EPL_switch is "E" then arg1 = E_value
% if EPL_switch is "P" then arg1 = P_value
% if EPL_switch is "L" then arg1 = L_value
%
% a POSITIVE tau_err will return a larger earily than late value, and vice versa
%

global CHIP_SPACING   %spacing between chips (in seconds)
global T_C            %length of one chip (in seconds)
if nargin<2
   E_value = 1-min(abs(tau_err-CHIP_SPACING)/T_C,1)
   P_value = 1-min(abs(tau_err)/T_C,1)
   L_value = 1-min(abs(tau_err+CHIP_SPACING)/T_C,1)
else
   switch lower(EPL_switch)
   case 'e'
      arg1 = 1-min(abs(tau_err-CHIP_SPACING)/T_C,1); %E_value
   case 'p'
      arg1 = 1-min(abs(tau_err)/T_C,1); %P_value
   case 'l'
      arg1 = 1-min(abs(tau_err+CHIP_SPACING)/T_C,1); %L_value
   end
end
% End Function
%-------------------------------------------------------------------------------

%-------------------------------------------------------------------------------
function [I,Q] = gen_IQ(DCO_phase_err, DCO_freq_err, Delta_T, pre_corr_SNR_dB)
% usage: [I,Q] = gen_IQ(DCO_phase_err, DCO_freq_err, Delta_T, pre_corr_SNR_dB)
%
% This function returns the simulated accumulated I & Q values for the GP2021 chipset
%
% DCO_phase_err      initial phase error in the DCO at the beginning of the
%                    Correlation interval of Delta_T, defined as: (in Radians)
%
%              DCO_phase_err = (True Signal Phase)-(Estimated Signal Phase)
%
%                    Note that for the GP2021, the subsampling causes a phase reversal
%                    such that a increase in DCO_phase_err must be compensated by a
%                    physical DECREASE in the ACTUAL DCO phase.  Note also that the
%                    GP2021 doesn't directly adjust the DCO phase, only the frequency,
%                    when it is in signal tracking mode.
%
% DCO_freq_err       inital frequency error in the DCO at the beginning of the
%                    Correlation interval of Delta_T, defined as: (in Hz)
%
%              DCO_freq_err = (True Recv. Freq.) - (Estimated Recv. Freq.)
%
%                    Note again due to the subsampling, the ACTUAL DCO frequency is
%                    LOWERED by a positive Doppler Frequency, and RAISED by a negative
```

```
%                    Doppler Frequency.  (+ Doppler = relative distance DECREASING)
%
% Delta_T            The time interval of the I's & Q's to be summed.  Normally an
%                    integer multiple of 1ms.
%
% pre_corr_SNR_dB    Signal To Noise Ratio (in Db) prior to correlation (without the
%                    43 Db of Processing Gain for a 20ms accumulation)
%
% This function will work with singlar values OR VECTORS.  Notes no VECTOR USE:
% DCO_phase_err, DCO_freq_err, pre_corr_SNR_dB, AND Delta_T vectors must be the same
% length, in THE SAME DIRECTION (i.e., row vectors or column vectors).

global SAMPLING_FREQUENCY
global NO_NOISE
global SUB_SAMP
% if the I/Q model is "modeling" a subsampling recevier, invert the errors
if SUB_SAMP
   DCO_freq_err=-DCO_freq_err;
   DCO_phase_err=-DCO_phase_err;
end

s_pls_n_var = 3.4;
I_or_Q_sqrd_var = 8.5;
% max_amplitude equates the output of the I & Q model to a single signal with no
% noise being run through the SS & RM for ideal comparisons
max_amplitude = 1;

% get the signal to noise ratio as a pure ratio
snr_precor = 10.^(pre_corr_SNR_dB./10);
% calculate the signal power in the discrete quantized samples
sig_pwr = s_pls_n_var.*(snr_precor./(snr_precor+1));
if max_amplitude&NO_NOISE
   sig_pwr = 4.5;
end
% the signal amplitude, as sumed over a second.  The Delta_T scaling is done below
%sig_amp = SAMPLING_FREQUENCY.*sqrt(2*sig_pwr);
sig_amp = Delta_T.*SAMPLING_FREQUENCY.*sqrt(2*sig_pwr);
noise_std = sqrt(I_or_Q_sqrd_var*SAMPLING_FREQUENCY.*Delta_T);

% get the indexes of each vector where the DCO_freq_err is zero
%i=find(DCO_freq_err==0);
% now get the indexes of each vector where the DCO_freq_err is not zero
%n=find(DCO_freq_err~=0);
n=1:length(DCO_freq_err);
% for those indicees that have zero frequency errors, use these equations
%I(i)=sig_amp(i).*Delta_T(i).*cos(DCO_phase_err(i));
%Q(i)=sig_amp(i).*Delta_T(i).*sin(DCO_phase_err(i));
% for those indicees that have non-zero frequency errors, use these equations
%I(n)=sig_amp(n).*cos((Delta_T(n).*2*pi.*DCO_freq_err(n))/2+DCO_phase_err(n)).*...
%      (2./(2*pi.*(DCO_freq_err(n)))).*sin((2*pi*DCO_freq_err(n)./2).*Delta_T(n));
%Q(n)=sig_amp(n).*sin((-Delta_T(n).*2*pi.*DCO_freq_err(n))/2+DCO_phase_err(n)).*...
%      (2./(2*pi.*(DCO_freq_err(n)))).*sin((2*pi*DCO_freq_err(n)./2).*Delta_T(n));

% use a sinc implementation
I(n)=sig_amp(n).*cos((Delta_T(n).*2*pi.*DCO_freq_err(n))/2+DCO_phase_err(n)).*...
      sinc((2*DCO_freq_err(n)./2).*Delta_T(n));
Q(n)=sig_amp(n).*sin((Delta_T(n).*2*pi.*DCO_freq_err(n))/2+DCO_phase_err(n)).*...
      sinc((2*DCO_freq_err(n)./2).*Delta_T(n));



%if ~NO_NOISE
%   I=I+noise_std.*randn(size(noise_std));
%   Q=Q+noise_std.*randn(size(noise_std));
%end

% End Function
%-------------------------------------------------------------------------------
```

iq_model_globals.m

```
% These are the global variables that control the Matlab IQ Model

% global "CONSTANTS" (i.e., functions of the hardware or GPS system parameters)
%%%---------  NOTE: THESE VARIABLES ARE COMMON WITH THE SIGNAL SIM & CORRELATOR MOD!!
% GPS Constants:
global CARRIER_FREQUENCY      % L1 carrier frequency
global CARRIER_WAVELENGTH     % L1 wavelength in meters.
global SPEED_OF_LIGHT
global T_C
% Hardware Dependent Constants:
global CHIP_SPACING
global SAMPLING_FREQUENCY
% Parameters to set
%global START_TIME
global SNR_DB
global SUBSAMP
%%%---------  NOTE: END VARIABLES THAT ARE COMMON W/ THE SIGNAL SIM & CORRELATOR MOD
% IQ_model_globals
global CAR_DCO_PHASE_ERR
global TAU
global CAR_DCO_FREQ_ERR
global CODE_DCO_FREQ_ERR
global LAST_LATCH_TIME
global CAR_DCO_FREQ_MAT
global CODE_DCO_FREQ_MAT
global CAR_DCO_PHZ_MAT
global CODE_DCO_DELAY_MAT
global LOS_RANGES_MAT
%global IQ_FREQ
global NO_NOISE
global MOD_OUTPUT

%DEFINED IN GPS_IQ_MOD
global LOS_DIST_MAT
global DOPPLER_FREQ_MAT
global TIME_OFF
global PRN_LST                %defined by the iq_model
global LOOKUP  % inverse mapping of PRN vec, PRNS to indicees

global DEFINE_GLOBALS

if DEFINE_GLOBALS==1;
%-------------------------------------------------------------------------%
%                    define the 'constants' (for GPS)
%-------------------------------------------------------------------------%
CHIPPING_RATE = 1.023E6;              % CA code chipping rate (Hz)
T_C = 1/CHIPPING_RATE;                % Length (in seconds) of a C/A code chip.
CARRIER_FREQUENCY = 1575.42E6;        % L1 received frequency (Hz)
SPEED_OF_LIGHT=299792458.0;           % (meters/second)
CARRIER_WAVELENGTH=SPEED_OF_LIGHT/CARRIER_FREQUENCY;  %L1 wavelength in Meters
%-------------------------------------------------------------------------%
%        define the frequency plan and other hardware dependent variables
%-------------------------------------------------------------------------%
CHIP_SPACING = T_C/2;                 % Correlator Chip spacing in Simulink model
SAMPLING_FREQUENCY = 40E6/7;          % From the Mitel GP2021 chip (Hz)
%-------------------------------------------------------------------------%
%              define the parameters that control and define the simulation
%-------------------------------------------------------------------------%
for i=1:12
   % SNR_DB represents the signal-to-noise ratio at the output of the GPS front-end
   SNR_DB(1,i)=-15;                   %indexed by channel
end
SUBSAMP=1;                            %if the final IF frequency is sub-sampled,
%                                     aliasing occurs and the PHASE AND FREQUENCY
%                                     ERRORS ARE REVERSED!
% the rest of these parameters are defined through function interface (i.e., PRN_VEC)
```

```
%-------------------------------------------------------------------------%
%              define the parameters that control and define the truth environment
%-------------------------------------------------------------------------%
CAR_DCO_PHASE_ERR(1:12) =0*ones(1,12);  %initial carrier phase error (radians)
TAU(1:12)               =0*ones(1,12);  %initial code correlation error (sec)
CAR_DCO_FREQ_ERR(1:12)  =0*ones(1,12);  %initial DCO carrier freq. error (Hz)
CODE_DCO_FREQ_ERR(1:12) =0*ones(1,12);  %initial DCO code freq. error (Hz)
LAST_LATCH_TIME         =[]; %(s)
CAR_DCO_FREQ_MAT        =[]; %(Hz)
CODE_DCO_FREQ_MAT       =[]; %(Hz)
CAR_DCO_PHZ_MAT         =[]; %(rads)
CODE_DCO_DELAY_MAT      =[]; %(sec)
NO_NOISE                =0;   % if set to "1", it doesn't add noise to the I's & Q's.
MOD_OUTPUT              =1;  % select alternate output out of IQ_mod.
% IQ_FREQ is defined through the function interface
end
```

Bibliography

[1] Accord. "GPS Correlator Simulator," Reference to a GPS Receiver Model, WWW Site. http://www.accord-soft.com/Crsm.htm. 25 January 2000.

[2] Air Force Research Laboratory Munitions Directorate. "A Request for Information – Miniature Integrated Navigation Technology (MINT)," *Commerce Business Daily*, 27 July 1999.

[3] Air Force Research Laboratory Munitions Directorate. *Miniature Integrated Navigation Technology (MINT) Technology Investment Plan*. Eglin AFB, FL, 1999.

[4] Anderson, Jon and Steve Lazar. "Lm and Lc: How the Military and Civilian Users of GPS Can Reuse Our Existing Spectrum," *Proceedings ION GPS 97*:1229:1235. Kansas City, Institute of Navigation, September 1997.

[5] Anon. GPS Interface Control Document GPS-IDC-200, with IRN-200C-PR002. Department of the Air Force. September 25, 1997.

[6] Baier, Fred P. *A GPS Code Tracking Receiver Design for Multipath Mitigation Using Maximum Likelihood Estimation*. MS Thesis, AFIT/GE/ENG/97D-18, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1997.

[7] Berkeley Design Technology, Inc. "Choosing Block-Diagram Tools for DSP Design," *DSP & Multimedia Technology,* April 1995.

[8] Berkeley Design Technology, Inc. "In Search of the Best Tool for DSP Design," *Electronic Engineering Times*, July 11, 1995.

[9] Defense Modeling and Simulation Office, Home Page, WWW Site. http://www.dmso.mil/. 26 January 2000.

[10] DSP Design Tools & Methodologies (Volume II), List of DSP graphical development programs, WWW Site. http://www.bdti.com/products/volume2.htm. 26 January 2000.

[11] GPS support Commission. *Global Positioning System Standard Positioning Service Signal specification*. DOD-4650.5/SPSSP V3, 1 August, 1998. WWW Site. http://www.peterson.af.mil/usspacecom/gps%5Fsupport/index.htm, 25 February 2000.

[12] Herd, William J. *High Dynamic Global Positioning System Receiver*, U.S. Patent 4,578.678. Washington: USPTO, 25 March, 1986

[13] International GPS Service Homepage. http://igscb.jpl.nasa.gov. 3 Feburary 2000.

[14] Kaplan, Elliott D. *Understanding GPS, Principles and Applications.* Boston: Artech House, Inc., 1996.

[15] Leap seconds, WWW Site. http://tycho.usno.navy.mil/leapsec.html. 14 February 2000.

[16] Leimer, Donald and Sanjai Kohli. "Receiver Phase-Noise Mitigation," *Proceedings ION GPS 98*:627:632. Nashville, Institute of Navigation, September 1998.

[17] Lukesh, John, "Optimal Use of Both GPS Carrier and Code Tracking Observations," Proceedings of the 1999 National Technical Meeting & 19th Biennial Guidance Test Symposium, Vision 2010: Present and Future:447:454. 1999

[18] Maybeck, Peter S. *Stochastic Models, Estimation and Control, I.* New York: Academic Press, Inc., 1979. Republished, Arlington, VA: Navtech, 1994.

[19] Maybeck, Peter S. *Stochastic Models, Estimation and Control, II.* New York: Academic Press, Inc., 1982. Republished, Arlington, VA: Navtech, 1994.

[20] Mitel Semiconductor. *GP2000 GPS Chipset – Designers Guide.* MS4395-2.3. April 1998. WWW Site. www.mitel.com.

[21] Mitel Semiconductor. *GP2000, GPS Receiver Hardware Design.* Application Note AN4855 Issue No 1.4. February 1999. WWW Site. www.mitel.com.

[22] Mitel Semiconductor. *GP2010, GPS Receiver RF Front End.* DS4056 ver 3.4. October 1996. WWW Site. www.mitel.com.

[23] Mitel Semiconductor. *GP2021, GPS 12 Channel Correlator.* DS4077 ver 2.6. July 1996. WWW Site. www.mitel.com.

[24] Modeling and Simulation Information Analysis Center, Home Page, WWW Site. http://www.msosa.mil.inter.net/home.htm. 26 January 2000.

[25] Modeling And Simulation of GPS Receivers Is Easier With New Software, *Wireless Systems Design*, April 1998. WWW Site. http://www.wsdmag.com/library/penton/archives/wsd/April1998/288.htm. 26 January 2000.

[26] *Modeling GPS Receivers in ACOLADE*, ACOLADE Application Note, PN – 3313-02-PD, ICUCOM Corporation, 1999.

[27] National Research Council. *The Global Positioning System, A Shared National Asset.* Library of Congress Catalog Card Number 95-69627. Washington, D.C.: National Academy Press, 1995

[28] NavSys Corporation, "GPS Receiver Modeling and Simulation Tools," Site explaining their GPS Signal Simulation Toolbox, WWW Site. http://www.navsys.com/Gpstools.htm. 29 January, 2000.

[29] NavSys Corporation, "TRACKTAG Autonomous TIGIT® Geolocation system," WWW Site. http://www.navsys.com/tracktag.htm. 10 March, 2000

[30] Navtech Seminars & GPS Supply/Development/Engineering/GPS Correlator Simulator, Reference on Navtech website to Accord's Correlator Simulator, WWW Site. http://navtechgps.com/supply/correlator_simulator.asp. 26 January 2000.

[31] NAWCWPNS GPS/INS Branch *Modeling & Simulation*, Reference to a GPS Receiver Model, WWW Site. http://sirius.chinalake.navy.mil/model.html. 25 January 2000.

[32] Oak Frequency Control Group, "Understanding AVAR," *Microwaves & RF,* Penton Media. December 1996. WWW Site. http://sites.penton.com/mwrf/library/penton/archives/mrf/December1996/403.htm

[33] Parkinson, Bradford W. and James J. Spiler, Jr. *Global Positioning System: Theory and Applications, Volume 1.* Washington, D.C.: American Institute of Aeronautics and Astronautics, Inc., 1996.

[34] Peterson, R.L., Rodger E. Ziemer, David E. Borth, *Introduction to Spread Spectrum Communications*, Englewood Cliffs, NJ: Prentice Hall, 1995.

[35] Presnar, Michael D. *Comparison of Ultra-Tightly Coupled Global Positioning System/Inertial Navigation System (GPS/INS) Integration Designs for Miniature Integrated Navigation Technology (MINT).* MS thesis, AFIT/GE/ENG/00M-12, School of Engineering and Management, Air Force Institute of Technology and Management (AU), Wright-Patterson AFB, OH, March 2000.

[36] Raquet, John F. Class Handouts, EENG 533, Navigation Using GPS. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, Spring 1999.

[37] Raquet, John F. Class Handouts, EENG 633, Advanced GPS Theory and Applications. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, Summer 1999.

[38] Raquet, John. *Development of a Method for Kinematic GPS Carrier-Phase Ambiguity Resolution Using Multiple Reference Receivers.* Ph.D. dissertation, UCGE Reports Number 20116, Department of Geomatics Engineering, University of Calgary, Calgary, Alberta, Canada, May 1998.

[39] Sennott, James W. "Receiver Architectures for Improved Carrier Phase Tracking in Attenuation, Blockage, and Interference," *GPS Solutions,* Vol. 3, No. 2: 40-47 1999

[40] Sennott, James W. and David M. Senffner. "A DGPS Signal Processor with Improved Blockage and Multipath Properties," *Proceedings ION GPS-93*: 923-928. Salt Lake City: Institute of Navigation, September 1993.

[41] Sennott, James W. and David M. Senffner. "A GPS Carrier Phase Processor for Real-Time High Dynamics Tracking," Proceedings of the 53rd Annual Meeting of ION:299-308. Albuquerque: Institute of Navigation, June 1997.

[42] Sennott, James W. and David M. Senffner. "Comparison of Continuity and Integrity Characteristics for Integrated and Decoupled Demodulation/Navigation Receivers," *Proceedings ION GPS-95*:1531-1537. Palm Springs: Institute of Navigation, September 1995.

[43] Sennott, James W. and David M. Senffner. "Robustness of Tightly Coupled Integrations for Real-Time Centimeter GPS Positioning," *Proceedings ION GPS-97*:655-663. Kansas City: Institute of Navigation, September 1997.

[44] Sennott, James W. and David M. Senffner. "The Use of Satellite Geometry for Prevention of Cycle Slips in a GPS Processor," *Journal of the Institute of Navigation*, Vol. 39 No. 2:217-235, Alexandria: Institute of Navigation, Summer 1992.

[45] Sennott, James W. and David M. Senffner. *Navigation Receiver with Coupled Signal-Tracking Channels*, U.S. Patent 5,343,209. Washington: USPTO, 30 August 1994.

[46] SimCentral, The Modeling and Simulation Information Network, Commercial Modeling and Simulation reference site, WWW Site. http://www.simcentral.com/. 26 January 2000.

[47] Simulation Based Acquisition Special Interest Area, Homepage, WWW Site. http://www.msosa.dmso.mil/sba/default.asp. 31 January 2000.

[48] Sklar, Bernard, *Digital Communications, Fundamentals and Applications*, Englewood Cliffs, NJ: Prentice Hall, 1988.

[49] Surathu, Mahesh. Resident GPS Signal Simulation Toolbox expert. Navsys Corporation. Telephone conversation. 28 January 2000.

[50] Tetwsky, Avram K. and Soltz, Arnold. "GPS MATLAB Toolbox Review," *GPS World*. 50-56, June 1998.

[51] The MathWorks, Inc., 21 Elliot Street, Natick, MA 01760. *MATLAB*. October 1999. Version 5.3.1.

[52] Tracking and Imaging Systems Inc., 418 N. Linden St. Bloomington IL 61701.

[53] United States Air Force Aircraft Fact Sheets. WWW Site. http://www.airpower.maxwell.af.mil/FactSheets

[54] USNO GPS Data & Information, WWW Site. http://tycho.usno.navy.mil/gps_datafiles.html. 25 February 2000.

[55] Vig, John R. *Quartz Crystal Resonators and Oscillators For Frequency Control and Timing Applications—A Tutorial*. AD-A328861 Rev. 8.3.7, September 1999. Updated version of report SLCET-TR-88-1, for the U.S. Army Communications-Electronics Command (CECOM), Fort Monmouth, NJ 07703, USA. WWW Site. http://www.ofc.com/, (click on App Notes.)

[56] VIS-SIM.ORG - The Visual Simulation Portal, reference site for visual simulation, WWW Site. http://www.vis-sim.org/. 26 January 2000.

[57] Ward, Phillip W. "Performance Comparisons Between FLL, PLL and a Novel FLL-Assisted-PLL Carrier Tracking Loop Under RF Interference Conditions," *ION GPS 98*:783:795. Nashville, Institute of Navigation, September 1998.

[58] Ward, Phillip W. "Using a GPS Receiver Monte Carlo Simulator to Predict RF Interference Performance," *ION GPS 97*:1473:1482. Kansas City, Institute of Navigation, September 1997.

[59] Zhodzinshsky, M. Yudanov, S., Veistel, V., and Ashjaee, J., "Co-Op Tracking for Carrier Phase," *Proceedings ION GPS 98*:653:664. Nashville, Institute of Navigation, September 1998.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
15-03-2000	Master's Thesis	Apr 1999 - Mar 2000

4. TITLE AND SUBTITLE
DESIGN AND VALIDATION OF AN ACCURATE GPS SIGNAL AND RECEIVER TRUTH MODEL FOR COMPARING ADVANCED RECEIVER PROCESSING TECHNIQUES

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Phillip M. Corbell, 2Lt, USAF

5d. PROJECT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
2950 P Street
Wright-Patterson AFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GE/ENG/00M-07

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Capt. Andrew W. Proud, Navigation Systems Engineer
AFRL/SNAR
2241 Avionics Circle
Wright-Patterson AFB, OH 45433-7318

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/SNAR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited

14. ABSTRACT
Recent increases in the computational power of computers and digital signal processors have made possible new, novel signal tracking techniques in GPS receivers. One such technique is known as Direct Correlator Output Processing (DCOP). DCOP is innovative in its potential to replace the tried and true classical signal tracking loops. It is also an enabling technology for ultra-tightly coupled GPS/INS (Global Positioning System/Inertial Navigation System). Potential benefits include improvement in positional accuracy in environments of jamming and high dynamics. However, such performance gains are typically based on software simulations of conceptual GPS receiver designs, not working prototypes. Simulating these new designs requires the modeling of GPS signals and receiver tracking loops, instead of the traditional pseudorange and carrier-phase measurements, which many proven GPS simulation software packages accurately model. The purpose of this research has been to develop an accurate, user-friendly, and customizable GPS signal and receiver model to use for a fair and unbiased evaluation of advanced receiver designs. The result of this research is a Matlab® GPS signal simulator, a Simulink® GPS receiver model implementing true receiver DSP processing, and a Matlab® high-speed signal/receiver model that approximates the signal simulator and receiver model

15. SUBJECT TERMS
GPS/INS, ultra-tight integration, receiver model, Matlab, Simulink, signal model, Direct Sequence Spread Spectrum, DSSS, DCOP, GPS, Global Positioning System, jamming, truth model, NAVSTAR, I and Q model, in-phase, quadrature-phase, sampled signal, advanced receiver design, DSP, receiver processing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Mikel M. Miller, Lt. Col, USAF
U	U	U	UU	227	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-6565 x4278

INSTRUCTIONS FOR COMPLETING SF 298

1. REPORT DATE. Full publication date, including day, month, if available. Must cite at least the year and be Year 2000 compliant, e.g. 30-06-1998; xx-06-1998; xx-xx-1998.

2. REPORT TYPE. State the type of report, such as final, technical, interim, memorandum, master's thesis, progress, quarterly, research, special, group study, etc.

3. DATES COVERED. Indicate the time during which the work was performed and the report was written, e.g., Jun 1997 - Jun 1998; 1-10 Jun 1996; May - Nov 1998; Nov 1998.

4. TITLE. Enter title and subtitle with volume number and part number, if applicable. On classified documents, enter the title classification in parentheses.

5a. CONTRACT NUMBER. Enter all contract numbers as they appear in the report, e.g. F33615-86-C-5169.

5b. GRANT NUMBER. Enter all grant numbers as they appear in the report, e.g. AFOSR-82-1234.

5c. PROGRAM ELEMENT NUMBER. Enter all program element numbers as they appear in the report, e.g. 61101A.

5d. PROJECT NUMBER. Enter all project numbers as they appear in the report, e.g. 1F665702D1257; ILIR.

5e. TASK NUMBER. Enter all task numbers as they appear in the report, e.g. 05; RF0330201; T4112.

5f. WORK UNIT NUMBER. Enter all work unit numbers as they appear in the report, e.g. 001; AFAPL30480105.

6. AUTHOR(S). Enter name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. The form of entry is the last name, first name, middle initial, and additional qualifiers separated by commas, e.g. Smith, Richard, J, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES). Self-explanatory.

8. PERFORMING ORGANIZATION REPORT NUMBER. Enter all unique alphanumeric report numbers assigned by the performing organization, e.g. BRL-1234; AFWL-TR-85-4017-Vol-21-PT-2.

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES). Enter the name and address of the organization(s) financially responsible for and monitoring the work.

10. SPONSOR/MONITOR'S ACRONYM(S). Enter, if available, e.g. BRL, ARDEC, NADC.

11. SPONSOR/MONITOR'S REPORT NUMBER(S). Enter report number as assigned by the sponsoring/ monitoring agency, if available, e.g. BRL-TR-829; -215.

12. DISTRIBUTION/AVAILABILITY STATEMENT. Use agency-mandated availability statements to indicate the public availability or distribution limitations of the report. If additional limitations/ restrictions or special markings are indicated, follow agency authorization procedures, e.g. RD/FRD, PROPIN, ITAR, etc. Include copyright information.

13. SUPPLEMENTARY NOTES. Enter information not included elsewhere such as: prepared in cooperation with; translation of; report supersedes; old edition number, etc.

14. ABSTRACT. A brief (approximately 200 words) factual summary of the most significant information.

15. SUBJECT TERMS. Key words or phrases identifying major concepts in the report.

16. SECURITY CLASSIFICATION. Enter security classification in accordance with security classification regulations, e.g. U, C, S, etc. If this form contains classified information, stamp classification level on the top and bottom of this page.

17. LIMITATION OF ABSTRACT. This block must be completed to assign a distribution limitation to the abstract. Enter UU (Unclassified Unlimited) or SAR (Same as Report). An entry in this block is necessary if the abstract is to be limited.