FEASIBILITY STUDY FOR THE IMPLEMENTATION OF GLOBAL POSITIONING SYSTEM BLOCK PROCESSING TECHNIQUES IN FIELD PROGRAMMABLE GATE ARRAYS

A Thesis Presented to

The Faculty of the

Fritz J. and Dolores H. Russ

College of Engineering and Technology

Ohio University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Sanjeev Gunawardena

November, 2000

THIS THESIS ENTITLED

"FEASIBILITY STUDY FOR THE IMPLEMENTATION OF GLOBAL POSITIONING SYSTEM BLOCK PROCESSING TECHNIQUES IN FIELD PROGRAMMABLE GATE ARRAYS"

by Sanjeev Gunawardena

has been approved

for the School of Electrical Engineering and Computer Science

and the Russ College of Engineering and Technology

Janusz A. Starzyk
Professor of Electrical Engineering

Jerrel R. Mitchell, Dean
Fritz J. and Dolores H. Russ
College of Engineering and Technology

ACKNOWLEDGMENTS

First and foremost, I must thank God for giving me the wisdom and strength to complete this work. I firmly believe that my faith in Him has brought me through this and many trials in life. I continue to give Him all the glory and honor He so rightly deserves.

Special thanks to my wife, Amali, for standing by my side, believing in me and encouraging me through this project. Without her prayers and undying support, this work would not have been completed. Thanks also to my daughter, Serendipity (7 months) who probably did not appreciate those times when Thathi (Daddy) had to go work on his thesis rather than play with her. Thanks to my parents for their prayers, support, and encouragement; my mother, who at times gave most of her paycheck to fund my electronics projects as a child, and my father who showed me the value of education and was instrumental in giving me the opportunity to study in the US.

A special thanks to my advisor, Dr. Starzyk, whom I have known and respected since my days as a sophomore at Ohio University. Not only is Dr. Starzyk responsible for teaching me most of what I know about hardware design during those six years, but he has been there for me numerous times; from understanding why an assignment was late, to making sure I had the opportunity to go to graduate school by nominating me to receive teaching assistantships. I appreciate the patience Dr. Starzyk had with me especially during the final days of completing this thesis. I also thank him for having faith in me to teach his VLSI design sequence while he was on sabbatical. That experience boosted my self esteem, and helped me discover talents I didn't know I had.

Special thanks to Dr. Frank van Graas for giving me the opportunity to work on this project. Without that summer internship, this work would not have been possible. I appreciate all those insightful discussions we had and thank him for letting me sit-in on his classes where I learned much more than GPS. Frank has been an invaluable resource and a great motivator.

Thanks to my graduate committee: Dr. Celenk, Dr. Curtis and Dr. Hunt for their time and helpful comments and suggestions. Dr. Hunt has given me many opportunities to shine while I worked for him in the Physics Department. Special thanks to Dr. Maarten Uijt de Haag for his suggestions, advice, and support; and for attending the defense of this thesis.

I must thank Ms. Gang Feng for providing the GPS dataset used in this work, and some of the results presented in Table 4.1. Thanks also to Dr. Chris Bartone for his helpful comments. Thanks to Jonathon Sayre for the many insightful discussions we had about block processing, and for showing me how to cool the Triple7 computer in order to run my simulations. Finally, thanks to all my friends and colleagues of the Avionics Engineering Center at Ohio University for their help and support in so many ways that are too numerous to mention.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SYMBOLS AND ABBREVIATIONS

σ	Standard Deviation
2D	two-Dimensional
3D	three-Dimensional
A&D	Accumulate-and-Dump (signal processing operation)
AD	Accumulated Doppler
ADC	Analog-to-Digital Converter
AGC	Automatic Gain Control
AM	Amplitude Modulation
ASIC	Application Specific Integrated Circuit
ASSP	Application Specific Standard Products
BPF	Bandpass Filter
bps	bits per second
BPSK	Binary Phase Shift Keying
B_W	Bandwidth
C/A	Coarse/Acquisition (GPS ranging PRN code)
C/N_0	Carrier to Noise Ratio in dB-Hz
CDMA	Code Division Multiple Access
CLB	Configurable Logic Block
CMOS	Complementary Metal Oxide Semiconductor
COTS	Commercial off the Shelf

CPLD	Complex Programmable Logic Device
CW	Continuous Wave
DDS	Direct Digital Synthesis
DFF	D-type Flip-Flop
DFT	Discrete Fourier Transform
DLL	Delay Locked Loop
DoD	Department of Defense (United States)
DPRAM	Dual-Port Random Access Memory
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processing
FFT	Fast Fourier Transform
FLL	Frequency Locked Loop
FM	Frequency Modulation
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GAEM	GPS Anomalous Event Monitor
GLONASS	Global Orbiting Navigation Satellite System
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GPS-PPS	Global Positioning System – Precise Positioning Service
GPS-SPS	Global Positioning System – Standard Positioning Service
GUI	Graphical User Interface

HOW	Hand-Over Word
I/O	Input/Output
IDFT	Inverse Discrete Fourier Transform
IEEE	Institute of Electrical and Electronics Engineers Inc.
IF	Intermediate Frequency
IFFT	Inverse Fast Fourier Transform
IMU	Inertial Measurement Unit
IOB	Input/Output Block
IP	Intellectual Property
kHz	kilohertz
km	kilometers
LAAS	Local Area Augmentation System
LGF	LAAS Ground Facility
LNA	Low Noise Amplifier
LUT	Lookup Table
MAC	Multiply-and-Accumulate (signal processing operation)
MDB	Minimum Detectable Bias
MHz	Megahertz
mHz	millihertz
ms	milliseconds
N/A	Not Applicable
NCO	Numerically Controlled Oscillator

NF	Noise Figure
NRE	Non-recoverable Expenditure
OCXO	Oven Compensated Crystal Oscillator
PAL	Programmable Array Logic
PC	Personal Computer
PCI	Peripheral Component Interconnect (computer bus standard)
PDF	Probability Density Function
PE	Processing Element
PLA	Programmable Logic Array
PLD	Programmable Logic Device
PLL	Phase Locked Loop
PR	Pseudorange
PRN	Pseudo-Random Noise
PSOC	Programmable System on a Chip
RAM	Random-Access Memory
RF	Radio Frequency
ROM	Read-Only Memory
RTL	Register Transfer Level
RTOS	Real Time Operating System
SA	Selective Availability
SDF	Standard Delay Format
SOC	System on a chip

SRAM	Static Random Access Memory
SS	Spread Spectrum
SV	Space Vehicle (satellite)
TOA	Time of Arrival
TOW	Time of Week
UI	User Interface
UTC	Coordinated Universal Time
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLSI	Very Large Scale Integration
w.r.t.	with respect to
µs	microseconds

1 INTRODUCTION

Satellite based navigation systems represents the pinnacle of navigation technology for the 21st century. Presently, there are two such global navigation satellite systems (GNSSs) in operation: the US developed Navstar Global Positioning System (GPS) and the Russian Global Orbiting Navigation Satellite System (GLONASS). Initially developed by the US department of Defense (DoD), the popularity of GPS has soared over the past decade due to new and ingenious commercial and consumer applications that include surveying, civil aviation, rescue operations, recreational uses, and emergency 911 location services to name but a few. As these applications become more widespread, users demand increased reliability, accuracy and availability that is not always possible with the current capabilities of GPS. This is particularly true when the system is used in urban environments, canyons, and inside buildings where the GPS signal is too weak to detect, or its performance is severely hampered by multipath reflections. These issues have driven the development of new receiver architectures that strive to improve the accuracy and reliability of GPS under these severe operating environments.

Modern GPS receivers use sequential digital signal processing (DSP) techniques. That is, the signal is processed sample by sample. These types of sequential receivers use traditional phase-locked loops (PLLs), frequency-locked loops (FLLs), delay-locked loops (DLLs), and correlators to process the received signal. The front ends of these receivers require high-precision analog components that are expensive and difficult to manufacture. In addition, the component values used deviate from one to another, which means that the receivers need to be individually calibrated. Furthermore, the component values drift over time and reduce reliability and accuracy as the receivers age. Even though modern GPS receivers are built using dedicated hardware due to their application specific nature and the need for tight integration, the sequential processing scheme has limited the performance of these receivers when used in severe operating environments.

A novel receiver architecture known as a software radio has emerged in the last decade. Due to its radically new architecture, the software radio has been referred to as the most significant development in receiver technology since the superheterodyne concept. The goal of the software radio is to sample the received signal as close as possible to the antenna using an analog to digital converter (ADC) and perform all processing in software. By sampling the signal close to the antenna, most of the analog processing components such as mixers, oscillators, filters, and their associated non-linearities can be removed since the signal is processed entirely in the digital domain using relatively cheap general-purpose microprocessors or DSP processors. Thus, the software radio offers increased reliability and repeatability (i.e. manufacturability) compared to traditional receivers. In practice, sampling at the antenna requires an ADC capable of operating at the Nyquist rate of the incoming signal, while having exceptionally low sampling noise and good stability. Such components are either not available today, or are too expensive for mass production. Processing the huge amounts of data generated through such a scheme presents another more severe problem. Due to these technology limitations, a compromise is reached by having at least one stage of analog downconversion before sampling.

The software radio is the ultimate receiver architecture since it offers infinite flexibility for implementing the processing algorithms. In other words, a software radio is a universal receiver that is programmable for virtually any application imaginable (within spectrum limitations). This receiver architecture is ideal for today's fast paced world, where product life cycles are measured in months and telecommunication standards and user requirements keep changing regularly.

The software radio has also unleashed a new breed of DSP techniques available for processing GPS signals. Instead of the DSP equivalents of the various sequential processing techniques used in the past, blocks of incoming data may be processed at once, similar to the techniques used for image processing. When applied to GPS, these block-processing techniques have been successful in pushing the envelope of its capability. For example, a GPS signal with a carrier-to-noise ratio lower than 44 dB-Hz is too weak and difficult for a standard receiver to detect. However, block-processing techniques have been used to successfully acquire signals as low as 20 dB-Hz [UijtdeHaag99]. This capability enables GPS positioning to work inside buildings and terrain with thick vegetation where GPS would otherwise be unusable.

The drawback of block processing techniques is their huge computational requirement. Currently developed block processing techniques require the computation of large discrete Fourier transforms (DFTs) using the fast Fourier transform (FFT) algorithm, and takes on the order of several seconds to process a single millisecond of data in a fast (~500 MHz) computer. This means even dedicated DSP processors would fail to handle the task in real time, because they do not offer such a leap in performance compared to general-purpose microprocessors. A custom hardware solution, such as an application specific integrated circuit (ASIC) could handle the computational requirement, but would defeat the philosophy of the software radio since it would then lack the desired flexibility. The ideal implementation platform for the GPS software radio is one that possesses the processing power of ASICs with the programming flexibility of general-purpose microprocessors.

Field Programmable Gate Arrays (FPGAs) are the latest addition to the family of programmable logic devices (PLDs) that includes programmable logic arrays (PLAs), programmable array logic (PALs) and complex programmable logic devices (CPLDs). The Xilinx™ Corporation invented the FPGA in 1987 [Roelandts99]. FPGAs are manufactured using Static Random Access Memory (SRAM) technologies, and can therefore be considered as specialized memory circuits. The functionality of the FPGA is determined by programming SRAM cells that work as look up tables (LUTs) to determine the logic function, or drive CMOS pass transistors that act as switches to make connections between the various logic functions. Present day FPGAs have relatively large equivalent gate counts (between 100,000 and 10 million gates), operate at high system clock rates (~300 MHz for the Virtex-E™ device), and are capable of supporting large system-on-chip (SOC) applications. The rapid evolution of the FPGA to an above-million-gate device has made it a viable contender in applications that were once dominated by the ASIC. By far the most advantageous feature of the FPGA is that it can be programmed quickly and easily with any design an infinite number of times, whereas ASICs take several months to develop and manufacture. FPGAs are widely used today as prototyping tools to test and validate large designs before they are manufactured in ASIC technologies. They have also found a niche in the data communications industry, where FPGAs are used for complex data switching and routing applications. Another new area of research that involves FPGAs is called dynamic reconfiguration. Here, a section of the processor can be made to change its architecture on the fly based on intelligence about the application that is currently being processed in real time. The FPGA is the only technology available today that simultaneously fulfils the processing power and flexibility requirements needed for a real-time block-processing GPS software radio.

This thesis studies the feasibility of implementing a real-time block-processing GPS receiver using FPGAs. This is the first time an effort has been made to migrate the novel algorithms developed in software to custom designed digital hardware to ultimately realize such a receiver. The actual hardware implementation of the block processing algorithms is an immense task that would require years to develop a working prototype, and is beyond the scope of this work. The goal of this work was as follows: 1) Determine how the block processing algorithms can be implemented in FPGA hardware. 2) Develop a high-level hardware architecture that is optimized as much as possible for implementation in an FPGA, which can also be easily broken down to lower levels of abstraction within a hardware design flow. 3) Determine what type of throughput could be obtained from this architecture based on a high level machine-cycle based evaluation. 4) Determine the optimum datapath precision needed for acceptable performance based on a behavioral simulation of the developed architecture using real GPS data. 5) Determine if it is feasible to implement this architecture with currently existing FPGA technology. Since the architecture developed in this work is modeled at a high level of abstraction and not largely implementation platform specific, it is possible to take the results presented in this thesis to target a technology other than FPGAs (such as ASICs). However, the overall architecture described was meant for a FPGA implementation because of its obvious advantages.

This thesis is organized as follows: Chapters 2 covers the necessary GPS background. Chapter 3 describes the sequential GPS receiver architecture that is used in all modern receivers. Chapter 4 describes the GPS block-processing algorithms, examines some of their potential applications, and presents a concept model for a real-time block processing GPS receiver. Chapter 5 introduces FPGAs and presents their internal architecture. Chapter 6 covers FFT theory and describes the development of the 5000-point mixed-radix FFT/IFFT algorithm that is the basis for the GPS block processing techniques. Chapter 7 covers the hardware architectures of the individual block-processing subsystems developed in this work and evaluates the feasibility of the real time requirement based on a machine-cycle based analysis. Chapters 8 presents simulation results based on real GPS data that was used to test and validate the hardware architecture and determine its optimal parameters. Finally, Chapter 9 gives a summary of the work done for this thesis and discusses future work for realizing the long-term goals of this project.

2 THE GLOBAL POSITIONING SYSTEM

Global navigation satellite systems (GNSSs) are the latest and most advanced class of navigation systems based on the simple time-of-arrival (TOA) ranging concept. GNSSs use satellites in space to implement TOA raging using a principal known as tri-lateration. Presently, two such systems exist: the United States' Global Positioning System (GPS) and the Russian Global Orbiting Navigation Satellite System (GLONASS). Both systems were developed in parallel during the latter years of the Cold War. The scope of this thesis is limited to GPS.

GPS was developed by the DoD primarily for military use. However, since an initial positioning capability needed to be incorporated to acquire the precise positioning component of GPS, that initial positioning service was made available to civilian users. This lower-precision component is known as the standard positioning service (GPS-SPS), whereas the military component is known as the precise positioning service (GPS-PPS). GPS-SPS has found a tremendous wealth of applications ranging from recreational uses such as hiking to commercial and civil aviation, and today is the basis for a rapidly growing industry that is expected to double in the next three years to $16 Billion annually [AP00]. This is even truer after May 2, 2000 when the US government turned off Selective Availability (SA), which intentionally limited positioning accuracy to 150 Meters (2σ) for civilian users [WH00].

The following sections describe how the system is used to obtain a position solution; along with the GPS link budget, its signal structure and dynamics, as applicable to this thesis.

2.1 Positioning With GPS

As described above, GPS is based on TOA ranging. TOA ranging involves measuring the time taken for a signal transmitted by an emitter (foghorn, radiobeacon, or satellite) in a known location to reach an observer (receiver) at an unknown location (i.e. propagation time).

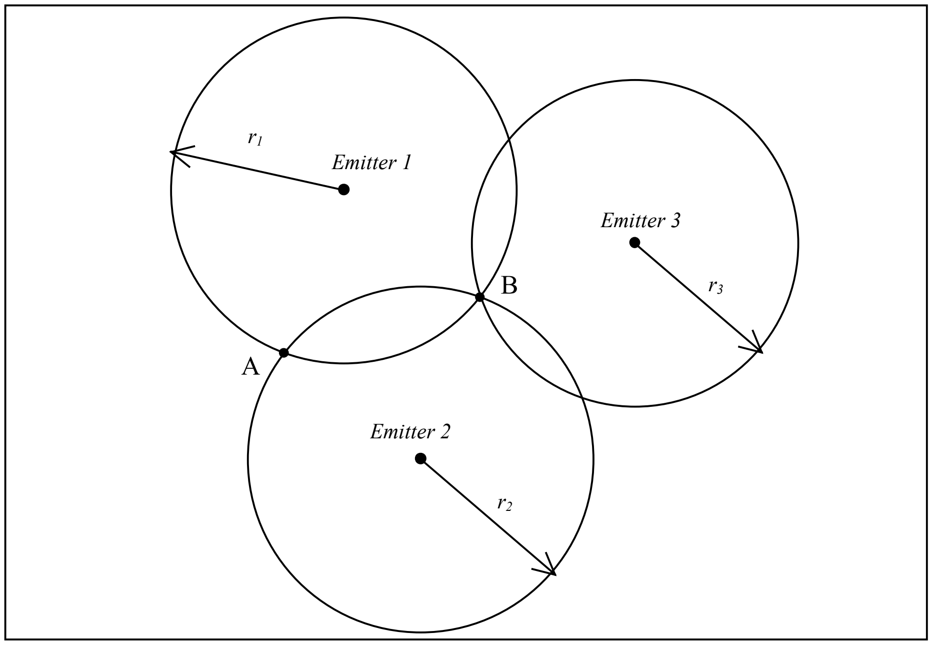


Figure 2.1 Graphical Illustration of Tri-Lateration

For the two dimensional (2D) case depicted in Figure 2.1, for one emitter, the position of the observer can be placed anywhere on a circle with radius r given by the measured range from the emitter. The difference between the time of reception, t_{re}, and the time of transmission, t_{tr}, times the propagation speed of the signal gives the range from the emitter to the observer. If the range from two emitters with known locations can be measured, the position of the observer is one of two solutions, *A* or *B*. Three range measurements results in a unique solution, *B*, for the position of the observer. Hence, this type of positioning is referred to as tri-lateration. For the three-dimensional (3D) case, the circles in Figure 2.1 are replaced with spheres. For satellite navigation, three range measurements alone would give rise to an ambiguity for the observer's position. This is resolved without the use of a fourth measurement by realizing that one solution is close to the surface of the earth whereas the other is not.

The GPS constellation (GPS space segment) consists of 24 satellites, called space vehicles (SVs), in six orbital planes, as illustrated in Figure 2.2.

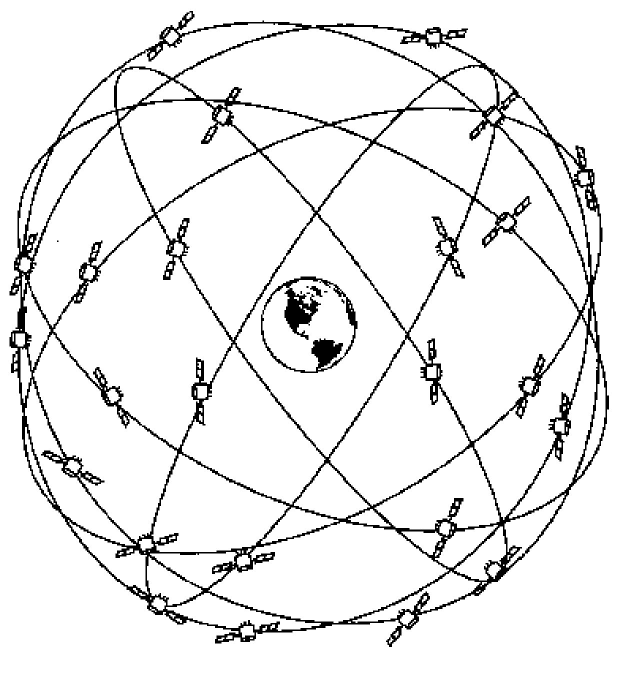


Figure 2.2 GPS Satellite Constellation (Not to Scale)

The orbits are configured in such a manner as to have at least four satellites visible to a user near the surface of the earth at any given location and time. Practically, it is common to receive between seven to nine SVs. Figure 2.3 shows the GPS SV global visibility profile for a 24-hour period [USDOT95].

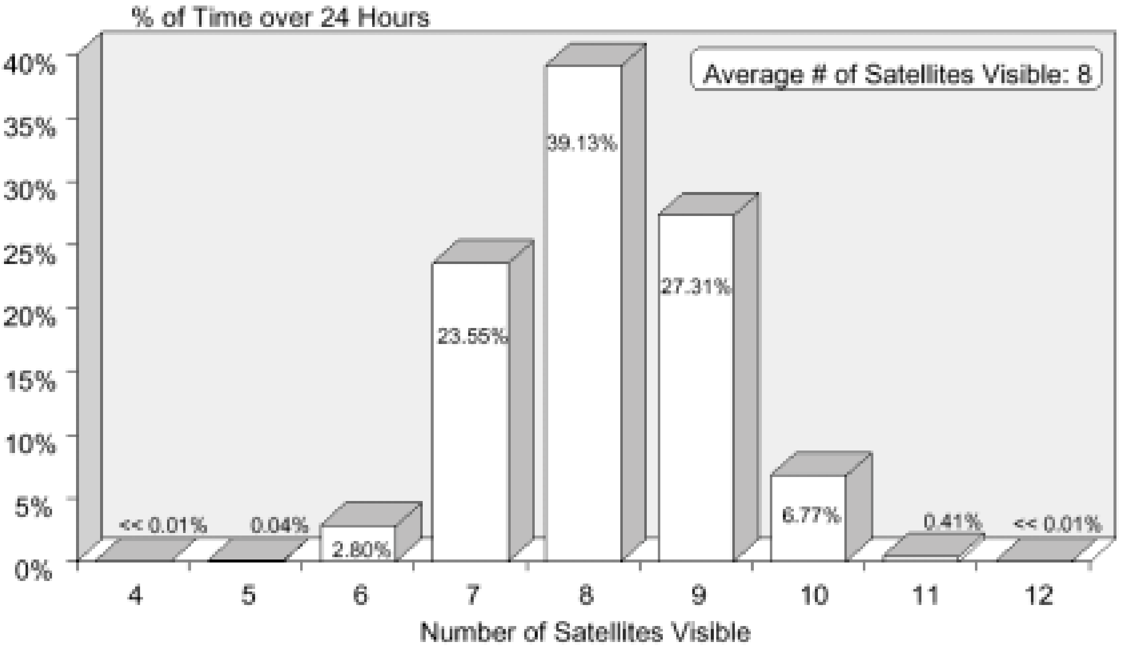


Figure 2.3 GPS Satellite Global Visibility Profile

For the i^{th} satellite, the true SV-to-user range, R_i is given by:

$$R_i = c\left(t_{re} - t_{tr_i} + \Delta t_{SV_i} + \Delta t_{Rcvr}\right) \tag{2.1}$$

Where:

t_{tr_i}	:	time of transmission of SV$_i$
t_{re}	:	time of reception relative to the receiver clock
Δt_{SV_i}	:	clock offset of SV$_i$ w.r.t. transmitted timing signals
Δt_{Rcvr}	:	offset in the receiver's internal clock w.r.t. received timing signals
c	:	GPS propagation constant (2.99792458×10^8 m/s)

A pseudorandom noise (PRN) code embedded within the GPS broadcast is used to perform timing measurements. When the code-tracking loop of the GPS receiver is time aligned to the incoming PRN code, the delay between the receiver's replica code epoch and the incoming code epoch (hence known as the *code phase*) gives the time of transmission, t_{tr}, of the signal. Since the range measurement for SV_i obtained from t_{tr} still contains the clock biases of the receiver and satellite, it is known as the *pseudorange* (PR) to SV_i and is given by:

$$PR_i = c\left(t_{re}(n) - t_{tr_i}(n)\right) \tag{2.2}$$

Where:

$t_{re}(n)$	:	receiver time corresponding to epoch *n* of the GPS receiver's clock
$t_{tr_i}(n)$	:	transmit time based on the SV$_i$ clock

Hence, when the receiver picks any replica code epoch to make range measurements for all SVs, $t_{tr_i}(n)$ is the natural measurement (obtained from the code tracking loop) that is used to calculate the pseudorange. The receiver and satellite clock biases are subsequently corrected when the receiver solves for the navigation position solution. The data stream embedded within the GPS broadcast contains the SV clock offset, Δt_{SV}. Once the receiver obtains the pseudoranges of at least four SVs, it can solve for position, *(x,y,z)*, and the clock bias, *B*, from:

$$PR_i = \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} + cB \tag{2.3}$$

Where:

(x_i,y_i,z_i):	position of SV$_i$ (from SV$_i$ ephemeris data)

When the receiver clock bias is corrected, the local reference clock is accurately synchronized to GPS time. Thus, GPS provides an accurate timing capability in addition to positioning. The absolute time (GPS time) is obtained from the time-of-week (TOW) contained in the data broadcast. The TOW count has a resolution of 1.5 seconds and is reset every Sunday at 00.00 hrs, GPS time. The offset between GPS time and UTC is also contained within the data broadcast.

2.2 GPS Link Budget

Figure 2.4 shows the link budget for GPS. A block IIR GPS SV transmits an equivalent power of 480 Watts towards the earth. Most of this power is lost according to the inverse-square law when the signal travels approximately 20,000 km to the earth's surface. By the time the GPS signal is received at the antenna, it is about 20 dB below the receiver's thermal noise floor.

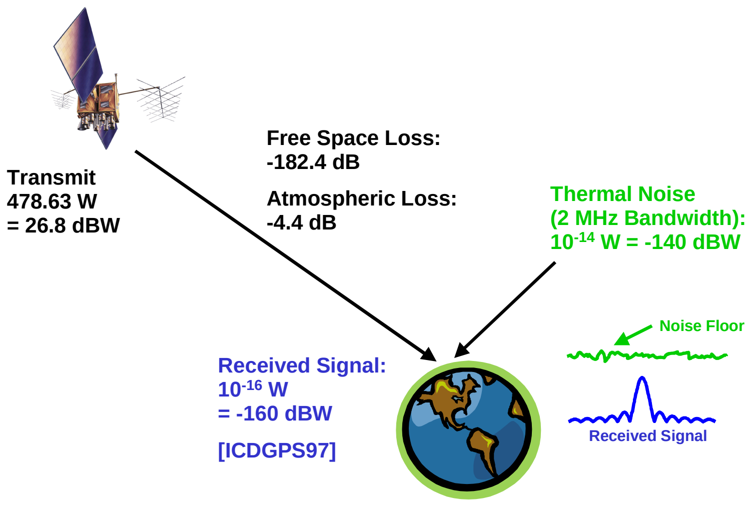


Figure 2.4 GPS Link Budget

Hence, traditional signal demodulation schemes such as those used for FM or AM cannot be used for spread-spectrum (SS) signals such as GPS because the signal is buried within the noise. Instead, the signal must be observed (integrated) over time so that the noise is averaged out, thereby raising the signal above the noise floor. To detect a GPS signal, integration is initially performed using *estimates* for the signal's code phase and carrier frequency using local synthesizers. This signal detection process is known as *acquisition*.

The received signal strength is guaranteed to be a minimum of –160 dBW [ICDGPS97]. The noise floor is approximated by $N = kTB_W$, where k is Boltzmann's constant ($1.3807\times10^{-23} J/K$), T is the Kelvin temperature, and B_W is the input bandwidth of the receiver in Hertz. Because the signal-to-noise ratio, *S/N*, is dependent on the input bandwidth, it is not a good measure of the true received signal strength. Hence, the carrier-to-noise ratio, given by $C/N_0 = S/kT$ [Braash91] is used instead. C/N_0 is expressed in dB-Hz. Given the minimum GPS received signal strength of −160 dBW, and a temperature of 300K, the minimum received C/N_0 is 44 dB-Hz.

2.3 GPS Signal Structure

The scope of this thesis is limited to the GPS-SPS broadcast. The signal structure for the GPS-SPS signal, S(t), can be modeled as:

$$S(t) = A(t)G(t)D(t)\sin(2\pi(f_0 + \Delta f_{SV}) + \phi_0) \tag{2.4}$$

Where:

$A(t)$	:	time dependant amplitude of the transmitted signal
$G(t)$	:	C/A code for the satellite (1.023 Mbps)
$D(t)$	:	navigation data stream (50 bps)
f_0	:	carrier frequency of broadcast (1575.42 MHz)
Δf_{SV}	:	satellite frequency offset
ϕ_0	:	carrier phase offset

Each SV transmits at the same L_1 carrier frequency of 1575.42 MHz. Spread-spectrum code division multiple access (CDMA) technology is used to demodulate the signal of each SV received at the antenna. The PRN code used for GPS-SPS is called the coarse acquisition (C/A) code. The 1023-chip long C/A code has a chipping rate of 1.023 Mbps, and hence a period, T_c, of 1 ms. The C/A code belongs to the family of Gold codes, possessing good multiple access properties for its period [Spilker78], an important factor for CDMA systems. However, unlike communications systems that use CDMA technology mainly for its inherent security and multiple-access capabilities, GPS uses its PRN code for ranging as well [Equation 2.2]. C/A code generation details are described in [ICDGPS97].

Modulo-2 added onto the C/A code is the 50 bps data stream, $D(t)$. The databits are generated in synchronous to the C/A code. The data message structure consists of a 1,500-bit long frame that is made up of five subframes, 1 through 5. Each subframe is 300 bits long, and consists of ten 30-bit words. The full data message consists of 25 full frames (12.5 minutes) since subframes 4 and 5 are subcommutated 25 times. However, decoding a full frame (30 seconds) is sufficient for a position calculation since it contains the necessary clock and ephemeris data in subframes 1 through 3. The second word of each subframe is the hand-over-word (HOW), which gives the precise GPS time-of-week (TOW) that will be valid at the starting edge of the next subframe. Thus, TOW updates are available every 6 seconds.

The bitstream containing the SV PRN code and the embedded data message is Binary Phase Shift Keying (BPSK) modulated onto the L_1 carrier.

2.4 SV-to-User Dynamics

Even though a GPS SV transmits at a constant L_1 carrier frequency, f_0, of 1575.42 MHz, SV-to-user dynamics causes the carrier frequency to vary depending on the relative line-of-sight velocity, v_r, causing a Doppler frequency shift, Δf_D, given by:

$$\Delta f_D = \pm\left(\frac{v_r}{c - v_r}\right) f_0 \tag{2.5}$$

For a stationary user, v_r is typically less than 1,500 m/s, which corresponds to a Doppler frequency offset of ±8 kHz.

The goal of this chapter was to explain the basic principals of GPS operation. In practice, many parameters such as geometry, atmospheric and ionospheric group delays, and multipath reflections affect the performance of GPS. The analysis of these effects and the errors they introduce to the final position calculation are beyond the scope of this work. The next chapter describes the architecture of a GPS receiver.

3 GPS RECEIVER ARCHITECTURE

In order to apply new signal processing techniques to GPS receivers, it is important to understand the architecture and functionality of traditional GPS receivers. Almost all modern GPS receivers sample the downconverted RF signal and perform all signal processing algorithms in the digital domain [Parkinson96]. The DSP techniques process the incoming data one sample at a time; that is, the processing technique is sequential. In order to differentiate this type of receiver from block processing receivers described in Chapter 4, the term *sequential receiver* is used in this thesis. This chapter describes the architecture and operation of a generic GPS-SPS sequential receiver.

3.1 Overview of Generic GPS-SPS Receiver

Figure 3.1 shows the block diagram of a generic GPS receiver. This diagram represents the minimum system required to receive the GPS-SPS broadcast of Equation 2.4 and calculate a navigation solution.

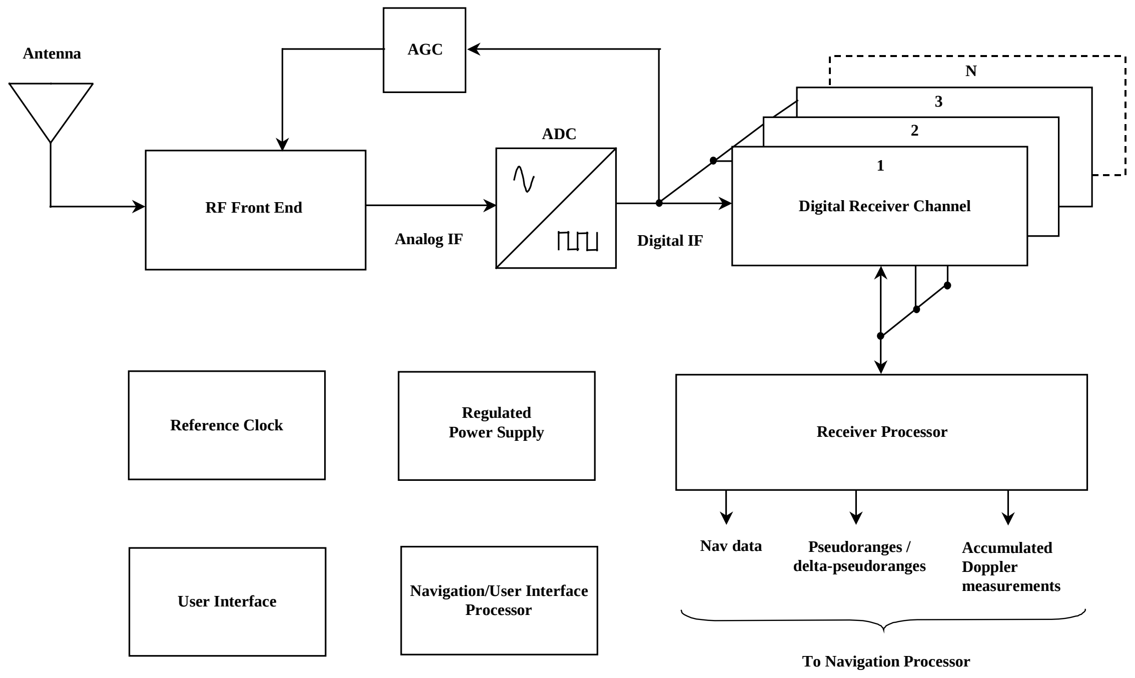


Figure 3.1 Block Diagram of a Generic GPS Receiver

The receiver can be sectioned into three main parts: 1) the RF front end, digitizing and automatic gain control (AGC), 2) the individual receiver channels and controlling processor, hereafter referred to as the *receiver processor*, and 3) the navigation/user interface processor and peripheral components such as power supply, user interface (UI), etc.

The RF front end amplifies the weak signals received at the antenna while rejecting unwanted out-of-band noise and downconverts the L_1 signal to the intermediate frequency (IF). All signal Doppler offsets and code phase information is preserved after downconversion (only the carrier frequency is lowered). An analog-to-digital converter

(ADC) samples the downconverted signal. The signal may also be digitally downconverted by the sampling process by using bandpass sampling techniques [Chapter 4]. Assuming the signal parameters do not change over the integration time of the receiver channel, the resulting digital IF signal, r_k, where k is the sample point *(k = 1,2,3...)*, can be represented as follows:

$$r_k = AG_{k,\tau}D_\tau \sin\left(2\pi\left(f_{IF} + \Delta f\right)T_S(k) + \phi\right) + n_k \tag{3.1}$$

Where:

τ	:	time delay due to signal transmission
A	:	signal amplitude
$G_{k,\tau}$	:	sampled, time-delayed C/A code
D_τ	:	time delayed navigation data stream
f_{IF}	:	digital IF frequency
Δf	:	frequency offset
T_S	:	sampling period $(1/f_S)$
ϕ	:	phase offset
n_k	:	sampled noise term

The received noise, n, is commonly modeled as additive white Gaussian noise [Parkinson96]. The frequency offset of the received signal, Δf, is the sum of four components:

$$\Delta f = \Delta f_{SV} + \Delta f_D + \Delta f_{LO} + \Delta f_S \tag{3.2}$$

Where:

Δf_{SV}	:	satellite frequency offset [Equation 2.4]
Δf_D	:	Doppler frequency offset [Equation 2.5]
Δf_{LO}	:	frequency offset due to the local oscillator
Δf_S	:	frequency offset due to sample clock error

The receiver has no means of distinguishing between Δf_D, Δf_{LO}, and Δf_S. Hence, it is important to minimize local oscillator and sample clock errors in the RF front end and sampling process respectively, so that they do not degrade the Doppler measurement. For this reason, the stable clock reference shown in Figure 3.1 is a critical component of any GPS receiver and influences the accuracy of the overall navigation solution.

The AGC circuit sets the amplitude of the sampled signal such that the dynamic range of the ADC is optimally utilized. Parallel processing is performed on the digital IF by the individual receiver channels. Each channel performs acquisition and tracking of individual satellites under the control of the receiver processor.

The natural measurements of a GPS receiver are the replica code phase, and the replica carrier Doppler frequency (if the receiver is in frequency lock with the incoming carrier signal) or the replica carrier Doppler phase (if the receiver is in phase lock with the incoming carrier signal). The receiver processor takes these measurements and calculates pseudoranges and delta-pseudoranges. In addition, if the receiver is in phase lock, it calculates the accumulated Doppler (AD) phase, which is used for millimeter-level positioning in surveying applications [Kaplan96]. The pseudorange and delta-pseudorange measurements are considered the GPS observables. The navigation processor uses the GPS observables, along with the navigation data to find a position solution. The navigation processor is also the intelligence component of the receiver that contains various algorithms to handle the complex decision making processes involved in a GPS receiver.

The most computation intensive process in the receiver (excluding the graphics processing for the UI) is the correlation operation that is performed within each receiver channel. For the correlation, the channel must perform multiply-and-accumulate (MAC) operations in real time at the digital IF data rate, f_S. In most modern receivers however, this is no longer considered an issue owing to the availability of ASICs that integrate all the math-intensive operations of the receiver channel. Such chips are available at relatively low cost due to volume production.

The following sections will describe tracking and acquisition of the GPS signal respectively.

3.2 Tracking Loops

As described in Section 2.2, the GPS signal must be integrated over time to raise it above the noise floor. The initial process of locating the GPS signal in terms of its code phase and carrier frequency is known as *acquisition*. Following acquisition, the code phase and carrier frequency estimates are fed into code and carrier tracking loops respectively. Provided the initial estimates are within their lock-in range, the tracking loops will keep the signal in lock, i.e. the loops will follow subsequent changes in code and carrier frequency (or phase). This is known as *tracking*. During tracking, if the code phase or carrier frequency suddenly changes value (i.e. steps), or they drift beyond the range of the tracking loops, the tracking operation will fail. This condition is known as *loss-of-lock*. When the receiver loses lock, it must reacquire the signal.

Figure 3.2 shows the simplified architecture within a single receiver channel. As shown in the figure, the carrier and code tracking loops operate together to lock code phase and carrier frequency simultaneously. The following subsections will describe the operation of the carrier and code tracking loops respectively.

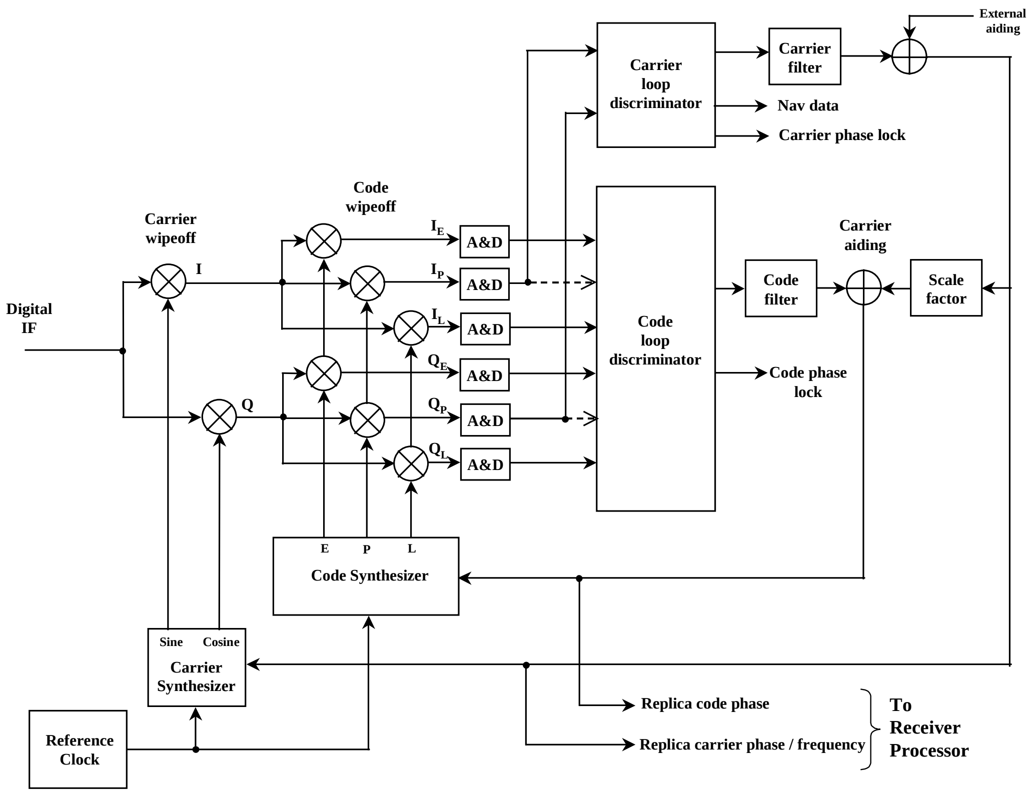


Figure 3.2 Block Diagram of a GPS Receiver Channel

3.2.1 Carrier Tracking

The digital IF is first mixed in quadrature with the replica carrier to convert it to baseband. A baseband signal is one that has been stripped of its carrier except for the residual, Δf. The purpose of the carrier tracking loop is to keep the carrier synthesizer (local oscillator) locked to the incoming carrier frequency (f_{IF} + Δf) so that the mixer output has no carrier component. Hence, this operation is known as carrier wipeoff. The signal is then correlated with the aligned replica C/A code (code wipeoff) and integrated using accumulate-and-dump (A&D) circuits. Since the A&D circuits have a low-pass characteristic, the double frequency term generated in the quadrature mixer is filtered out, leaving only the correlation value. The carrier loop discriminator outputs the frequency (or phase) offset magnitude and direction determined from the in-phase and quadrature components of the signal. After low-pass filtering, this error is fed back to the carrier synthesizer. GPS receivers use two types of carrier tracking loops: phase locked loops (PLL) and frequency locked loops (FLL). Both have their own advantages and disadvantages in terms of carrier tracking performance. The loop bandwidth is determined by the integration time (known as the pre-detection integration time, or dwell time, T_{DW}) and the characteristics of the filter. The filtered error value used to control the carrier synthesizer gives the replica carrier frequency offset, Δf. The carrier frequency offset is the basis for the delta-pseudorange measurement.

3.2.2 Code Tracking

The operation of the code-tracking loop is similar to the carrier-tracking loop in that the local replica code is compared to that of the incoming signal to produce a code offset that is filtered and used to control the code synthesizer. However, in the case of the code loop the replica code must be accurate to within one chip of the incoming code, since an offset greater than one chip produces zero correlation. By using the early and late correlation envelopes (and optionally the prompt correlation envelope, depending on the discriminator function), the code phase discriminator determines the magnitude and direction of the code phase error relative to that of the incoming signal. The early and late code streams are offset ½ chip relative to the prompt code stream. The noisy code phase error output of the discriminator is smoothed by the code loop filter.

Using the filtered code phase offset, the code synthesizer must advance or retard the replica code to align it with the incoming code. Hence, this type of tracking loop is called a delay locked loop (DLL). The code phase offset is the basis for the pseudorange measurement [Equation 2.2].

The difference in the pseudorange over a measurement time interval gives the SV's relative line-of-sight velocity. However obtaining this information from the code loop gives rise to noisy measurements. Instead, a 1000 times or 100 times less noisy delta-pseudorange measurement can be obtained from the carrier PLL or FLL respectively. Hence, the filtered replica carrier frequency (or phase) offset is used to smooth the error in the code loop. This is known as carrier smoothing (or carrier aiding, shown in Figure 3.2). The carrier smoothing technique is mathematically equivalent to including the second order term of an expansion series. In addition, for high accuracy and reliability, the tracking loops can be externally aided by a positioning system who's principal of operation is not related to GPS, such as an Inertial Measurement Unit (IMU).

The design of tracking loops is a complex subject and only a brief outline was given here. Detailed descriptions of the tracking loops used in GPS can be found in [Parkinson96].

3.3 Signal Acquisition

Figure 3.3 shows a block diagram of the acquisition architecture within the receiver channel. Much of the same hardware is used for acquisition, as apparent when comparing Figure 3.2 with Figure 3.3.

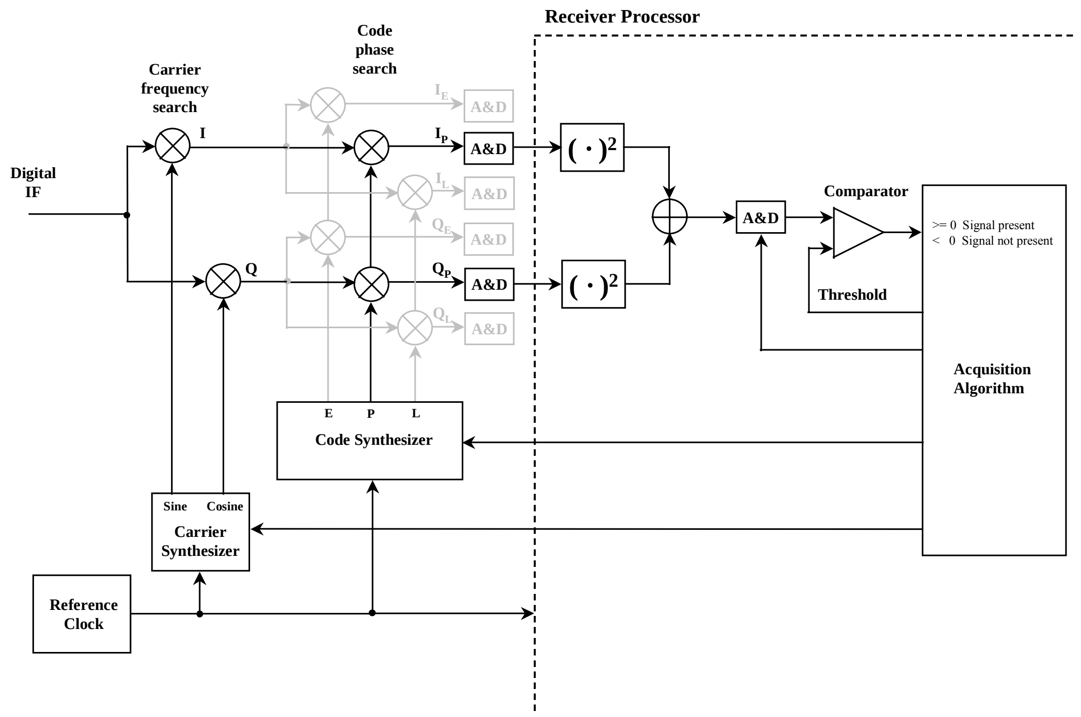


Figure 3.3 GPS Acquisition Configuration within a Receiver Channel

During acquisition, the channel operates in an open loop configuration where the acquisition algorithm tests successive code phase and carrier frequency combinations for a correlation peak, given by $I^2 + Q^2$ (i.e. the correlation energy). A second integration stage is used to sum correlations over multiple code periods to detect signals with low C/N$_0$. The preset threshold for determining the presence of a signal is calculated based on the application's required probabilities of false acquisition, P_{FA}, and missed detection, P_{MD}.

The receiver detects a correlation peak by comparing the correlation energy amplitude against a preset threshold, T_D. If this threshold is set too high, the receiver will not detect weak signals, thus increasing the probability of a missed detection. If the threshold is too low, an increase in the probability of false acquisitions due to noise will result. Figure 3.4 uses Gaussian probability density functions (PDFs) to illustrate the relationship between P_{FA}, P_{MD} and the minimum detectable bias, *MDB*. Based on the statistical distribution of the signal and noise parameters, and the required P_{FA}, the detection threshold can be calculated [Feng99].

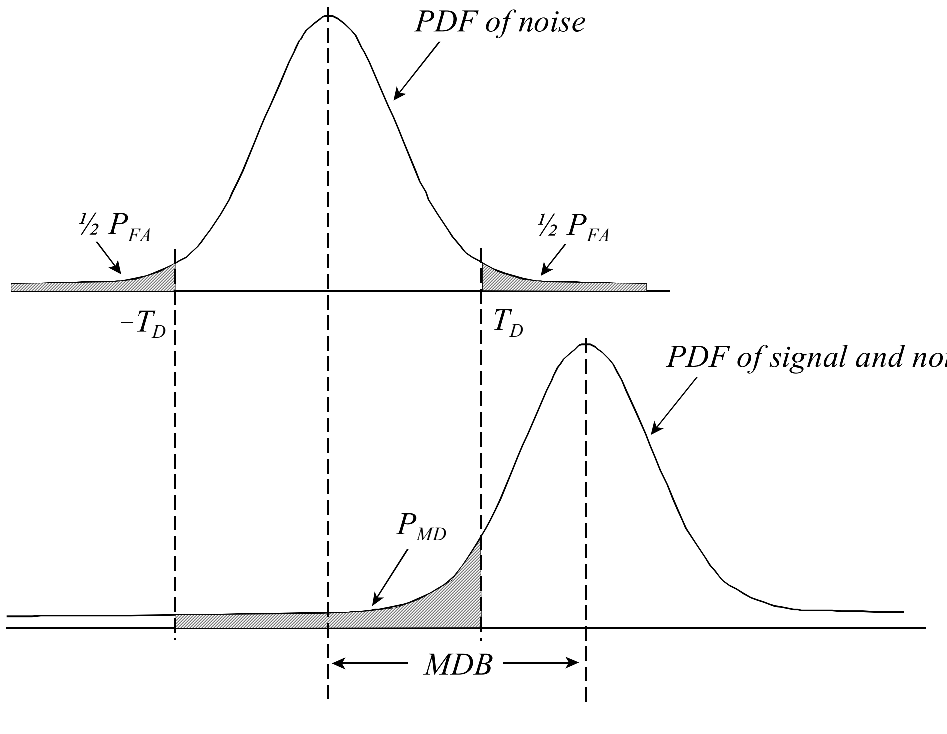


Figure 3.4 Illustration of P_{MD}, P_{FA}, and *MDB* using Gaussian PDFs

Acquisition is a three-dimensional search process, where the search dimensions are: 1) The CDMA PRN code of the SV, 2) The incoming signal's carrier frequency, and 3) The incoming signal's PRN code phase. If the receiver has knowledge of the approximate satellite orbits (almanac data), it can start searching for a probable SV based on the current time and the receiver's last known position. If this is not possible, the receiver must start by searching for any one of the 24 possible PRN codes. This is known as a "cold start". Assuming the receiver is searching for a given SV, the acquisition search space is shown in Figure 3.5.

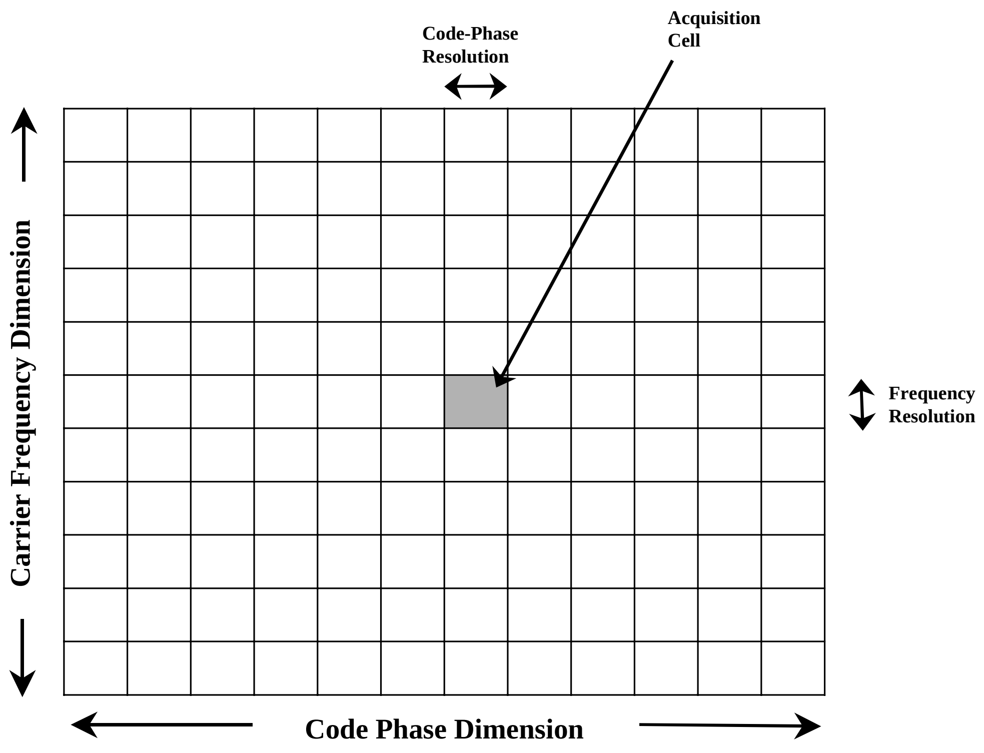


Figure 3.5 GPS Acquisition Search Space

Section 2.4 showed that the Doppler frequency offset is approximately ±8 kHz for a stationary observer. Indeed, for most typical applications Δf_D is less than ±10 kHz. If the receiver cannot estimate the Doppler, it must perform the entire search between L_1-10 kHz and L_1+10 kHz in steps of the Doppler frequency search resolution.

Since the replica code needs to align within one chip to obtain a correlation peak, the maximum code phase resolution is ½ chip. Because the GPS C/A code is 1023 chips long, there exists a minimum of 2046 search bins in the code phase dimension. If the receiver cannot estimate the pseudorange to the SV due to lack of a-priory information, it must check all 2046 bins.

Since the receiver must integrate to at least an entire C/A code period, the minimum dwell time for a correlation is 1 ms, which is adequate for high C/N_0 signals. However, during non-coherent acquisition (when the receiver code epoch is not synchronized to the SV code epoch), if a databit transition occurs within the dwell time, the correlation sign changes, effectively nulling the correlation result. Hence, a longer dwell time must be used during a cold start acquisition. Weak signals need longer dwell times per cell. This is accomplished by the second A&D operation in Figure 3.4.

The Doppler search resolution must be fine enough to land within the main lobe of the $Sinc^2$ correlation function (whose width is a function of the dwell time). The Doppler bin resolution can be estimated by $2/3T_{DW}$, where T_{DW} is the dwell time per cell. The minimum dwell time of 1 ms gives a Doppler bin resolution of 667-Hz. To acquire weak signals, T_{DW} can be as much as 20 ms, needing a Doppler bin resolution of 33-Hz [Kaplan96]. For example, during a cold start, if a typical sequential receiver searches the L_1-10 kHz to L_1+10 kHz carrier frequency interval in 1 kHz steps, it needs to search through 21 combinations in the carrier frequency dimension. Since the minimum code-phase dimension is 2046 ½-chip intervals, the total cold start acquisition search space for the receiver is:

$$2(1023)\times\left(2\left(\frac{10KHz}{1KHz}\right)+1\right)=(2046)(21)\approx 40{,}000\ \textit{acquisition cells}$$

Since the minimum dwell time is 1 ms (assuming no databit transitions), this operation will take about 40 seconds to complete in the worst case. Because every acquisition is not done in cold start mode, and the acquisition is performed in parallel in each of the receiver channels for candidate SVs, and because of the efficiency of the acquisition algorithms themselves, the mean time to acquire a signal is considerably smaller than this worst case estimate in a practical receiver. However, due to the inherent processing scheme in a sequential receiver, the mean time to acquisition is still in the order of a few seconds.

When GPS receivers are used in environments where the signal suffers severe attenuation (such as in dense foliage, inside buildings, or urban environments), or in high-dynamic environments (such as in fast aircraft or rockets), sequential receivers spend a significant amount of time in the acquisition phase, because under these conditions the receiver will often lose lock. Hence, the long acquisition time of a sequential receiver and its inability to cope with sudden signal steps in its tracking loops is a serious drawback for such applications. In these situations, a block processing architecture becomes a viable alternative to the sequential processing used in current GPS receivers. The next chapter describes the architecture of a block processing GPS receiver.

4 BLOCK PROCESSING GPS RECEIVER

In the GPS receiver of Chapter 3, the processing was performed sequentially. This method of processing, known as *stream processing* [Ackenhusen99], has inherent disadvantages such as relatively long acquisition times and the receiver constantly having to reacquire the signal following a loss-of-lock condition. Because of these shortcomings, it may not be the optimal architecture for positioning under challenging conditions when the GPS signal is severely attenuated (inside buildings, under dense foliage, etc.), is subject to high dynamics (in fast aircraft or rockets), multipath reflections (in urban environments, canyons, etc.), or interference conditions (during intentional and unintentional jamming). An alternate processing technique, known as *block processing*, has been shown to provide better performance under these conditions as described in [Moeglein98] and [UijtdeHaag99].

A block-processing scheme stores incoming samples in groups as they arrive. After the arrival of a sufficient number of samples (as set by the nature of the implementation), processing on the group begins. Because all input samples of the group, or block, are available simultaneously, processing can access the samples in a random manner rather than being restricted to sequential access. Figure 4.1 shows blocks of five samples being stored as they arrive. When the final sample of block *i* arrives, processing begins. At this point, two activities proceed together: the processing of block *i*, and the input and storage of block *i+1*. Block processing can be used when the input sample rate is much greater than the output sample rate [Ackenhusen99]. For a GPS receiver, processing is performed on a block of digital IF data to obtain code phase and Doppler offset values for each of the SVs being received. Since the block must cover at least a whole PRN code period (i.e. several thousand samples), and the output is only two values per SV, the conditions for block processing are satisfied.

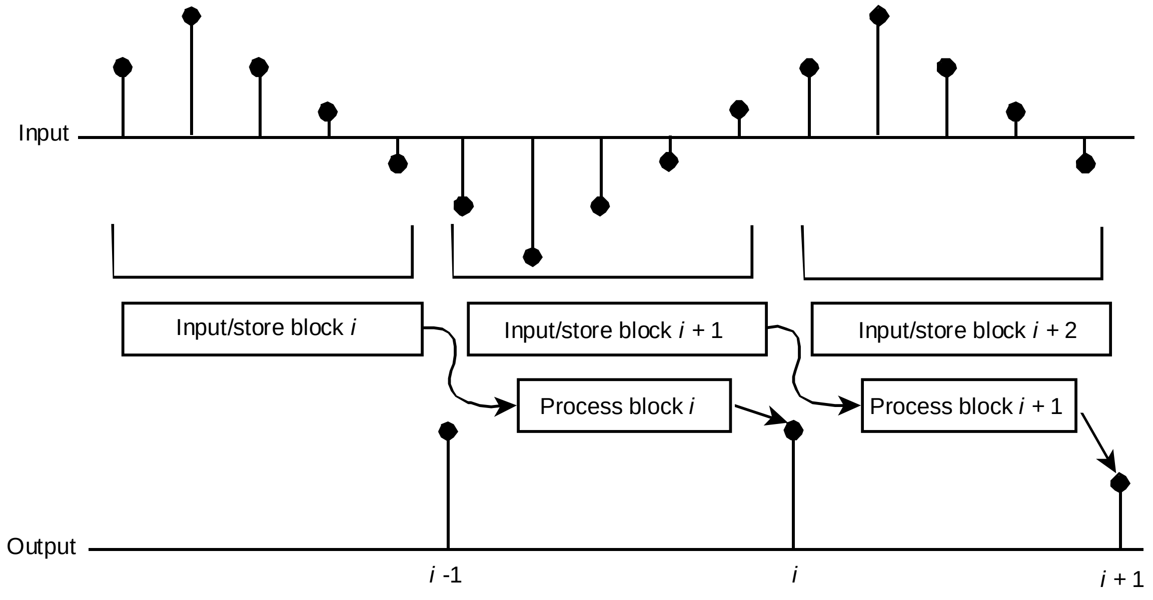


Figure 4.1 Illustration of Block Processing for a Block of Five Samples

4.1 GPS Data Collection System

As a precursor towards developing a block-processing GPS/GLONASS receiver, a relatively novel receiver architecture known as a software radio was designed by Akos [Akos97]. Based on this work, and subsequent revisions, a standard GPS data collection system for developing and testing block-processing algorithms designed by the Ohio University Avionics Engineering Center was used for this work. Figure 4.2 shows the block diagram of this setup.

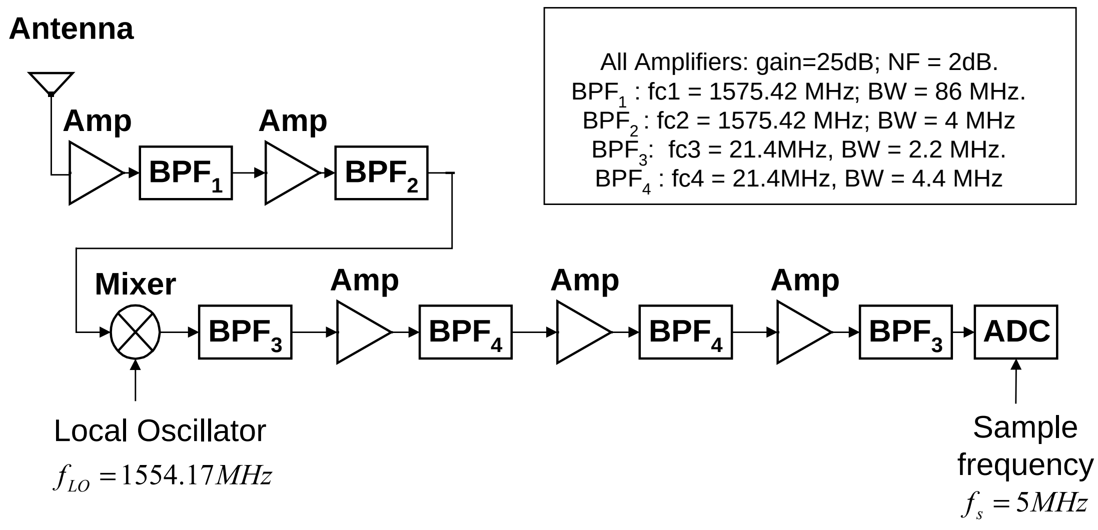

Antenna
Amp
BPF$_1$
Amp
BPF$_2$
All Amplifiers: gain=25dB; NF = 2dB.
BPF$_1$: fc1 = 1575.42 MHz; BW = 86 MHz.
BPF$_2$: fc2 = 1575.42 MHz; BW = 4 MHz
BPF$_3$: fc3 = 21.4MHz, BW = 2.2 MHz.
BPF$_4$: fc4 = 21.4MHz, BW = 4.4 MHz
Mixer
BPF$_3$
Amp
BPF$_4$
Amp
BPF$_4$
Amp
BPF$_3$
ADC
Local Oscillator
$f_{LO} = 1554.17MHz$
Sample frequency
$f_s = 5MHz$

Figure 4.2 GPS-SPS Data Collection System

Satellite data is collected through a roof-mounted antenna with an integrated 25 dB low noise amplifier (LNA). The first filter has a center frequency of 1575.42 MHz and a 3dB bandwidth of 86 MHz. The signal is amplified and filtered again before being mixed with the local oscillator frequency, f_{LO}, of 1554.17 MHz, resulting in a first standardized IF of 21.25 MHz. The IF is filtered to remove the double-frequency term followed by three more amplification and bandpass filtering stages to boost the signal to the sampling range of the ADC, while keeping out-of-band noise and spurious responses to a minimum. The signal is then sub-sampled at f_S = 5 MHz, which results in the 21.25 MHz IF aliasing to 1.25 MHz, whilst preserving the 2.2 MHz information bandwidth of the original signal. This intentional undersampling of a bandpass filtered signal is known as *bandpass sampling*, and results in a digital downconversion to the 1.25 MHz IF. Further details of bandpass sampling can be found in [Vaughan91]. The resulting digital IF data stream is currently stored in the hard disc of the PC housing the ADC card for post-processing [Snyder99]. The ADC has a quantization of 12 bits. Phase-noise due to sampling is kept to a minimum by using a Rubidium-referenced oven compensated crystal oscillator (OCXO).

The GPS-SPS sampled signal model presented in Equation 3.1 for the sequential case is modified for block processing as follows:

$$r_{i,k} = A_i G_{k,\tau_i} D_{i,\tau_i} \sin\left[2\pi\left(f_{IF} + \Delta f_i\right) T_S (m_i + k) + \phi_i\right] + n_{i,k} \qquad (4.1)$$

Where:

i	:	ith block ($i = 1,2,3,...$)
k	:	kth sample point starting at the ith block($k = 1,2,...,M$)
m_i	:	total number of samples before the ith block $m_i = (i-1)\times M$
M	:	number of samples per block, $M=5000$
τ_i	:	time delay due to signal transmission for block i
A_i	:	signal amplitude for block i
G_{k,τ_i}	:	sampled, time delayed C/A code
D_{i,τ_i}	:	navigation databit sign for block i
f_{IF}	:	center frequency of the digital IF, f_{IF} = 1.25 MHz
Δf_i	:	frequency offset for block i
T_S	:	sampling period ($1/f_S$), T_S = 0.2 μs
ϕ_i	:	phase offset for block i
$n_{i,k}$	:	sampled noise term

Then, the frequency offset for block *i*, Δf_i, becomes:

$$\Delta f_i = \Delta f_{SV,i} + \Delta f_{D,i} + \Delta f_{LO,i} + \Delta f_{S,i} \tag{4.2}$$

Where:

$\Delta f_{SV,i}$	:	satellite frequency offset for block *i*
$\Delta f_{D,i}$	:	Doppler frequency offset for block *i*
$\Delta f_{LO,i}$	:	local oscillator frequency offset for block *i*
$\Delta f_{S,i}$	:	sample clock error for block *I*

Since the sampled data must contain at least an entire C/A code period of 1 ms and the sample frequency is 5 MHz, the block size is chosen to be 5000 samples (i.e. M=5000).

A dataset of 1-second duration (*i*=1,2,3,…I; where I=1000) acquired from the above setup was used throughout this work. A NovAtel™ Millennium™ GPS Receiver was connected in parallel with the data acquisition system to monitor GPS parameters independently. Table 4.1 shows detectable SVs in the data along with their C/N$_0$ and Doppler frequencies [Feng99].

Table 4.1 GPS Data Set Parameters

	C/N$_0$ (dB-Hz)		Doppler frequency (Hz)		
SV#	NovAtel Receiver	Block Processing	NovAtel Receiver	Block Processing	Ad-hoc method
10	50.377	51.0603	939.9	1259	1261
24	48.209	47.1914	-650.5	-330	-329.6
30	46.497	46.3453	-428.3	-109	-108.4
13	46.011	45.649	1928.3	2258	2256
4	44.544	44.5948	-2274.5	-1954	-1951
5	N/A	N/A	N/A	N/A	-1931

The block processing techniques used to calculate these parameters is described in [Feng99]. For the Doppler frequency case, the results were independently verified using an ad-hoc method where the carrier frequency was spanned at 10 Hz intervals to find the frequency with the highest correlation energy. This was done for all blocks in the data excluding the blocks where possible databit transitions occurred. Since no prior knowledge of the SV data was available, the databit transitions were assumed to be at modulo 20 ms intervals of the block having the lowest correlation energy. Representative plots of this process are shown in Figures 4.3 and 4.4. The figures show the correlation energy for each block, and the result with the databit transitions removed, respectively. Table 4.1 shows that the results obtained by Feng and the results obtained from the ad-hoc method above agree to within 1% and verifies the accuracy of the formal block processing calculations applied to this data set. However, the values reported by the NovAtel™ receiver differ significantly. This discrepancy is attributed to the local clock offset that was not corrected in the block processing case. For simulations in this thesis, the Doppler frequencies were fixed to the values obtained by the ad-hoc method.

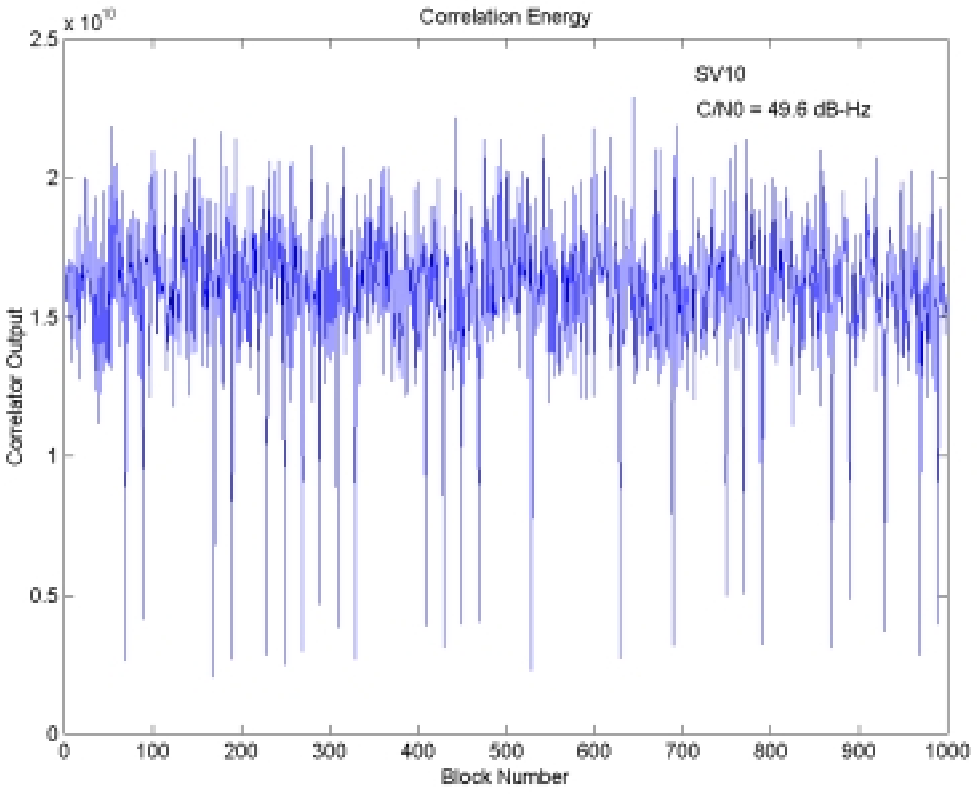


Figure 4.3 Correlation Energy Amplitudes for Each Block in Data Set

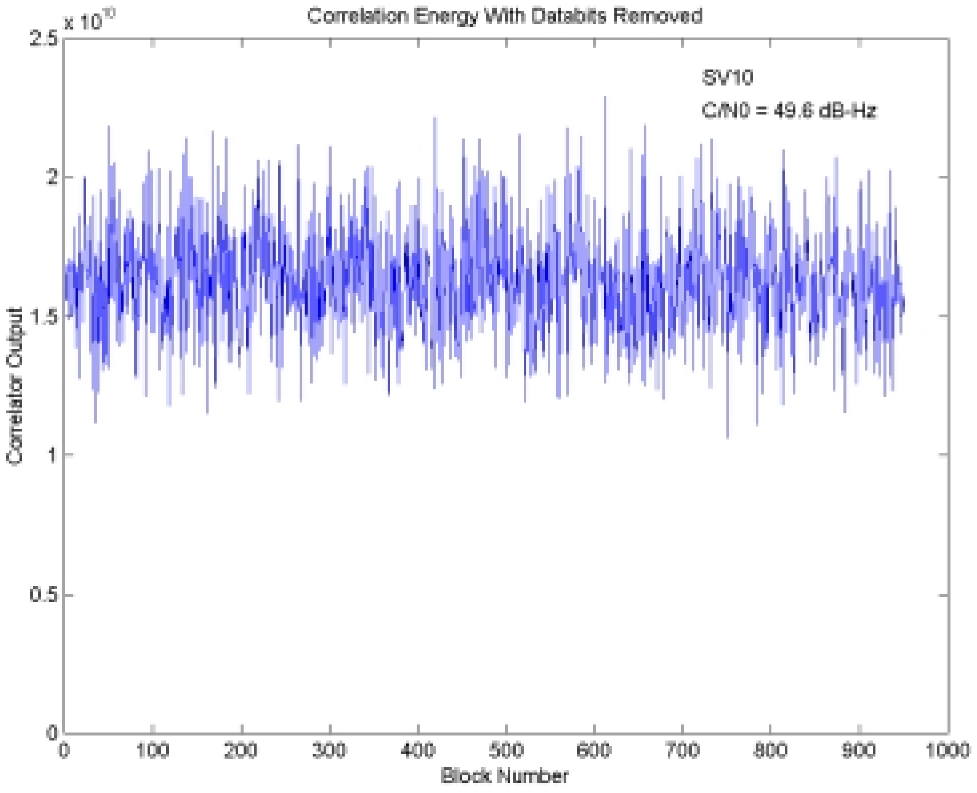


Figure 4.4 Correlation Energy Amplitudes with Databit-Transitioning Blocks Removed

4.2 GPS Fast Correlator

This section describes the GPS fast correlator that is the main signal processing component used within the GPS block-processing receiver.

As shown in Equation 4.1, since the data block $r_{i,k}$ and the replica C/A code, $G_{i,k}$ are both finite length sequences, their correlation can be performed by a slight modification of the circular convolution operation. The circular convolution of two finite-length sequences $x_1[n]$ and $x_2[n]$ can be obtained by taking the product of their discrete Fourier transforms (DFTs), $X_1[k]$ and $X_2[k]$, as given in Equation 4.3 [Oppenheim89]:

$$x_1[n] * x_2[n] \xleftrightarrow{DFT} X_1[k]X_2[k] \tag{4.3}$$

Where circular convolution is symbolized by *. By realizing that correlation is equivalent to convolution when one sequence is time reversed, that is:

$$x_1[n] * x_2[-n] \xleftrightarrow{DFT} X_1[k]X_2[k]^* \tag{4.4}$$

The correlation operation can be performed in the frequency domain as:

$$x_1[n] * x_2[-n] \equiv IDFT\{DFT\{x_1[n]\} \cdot DFT^*\{x_2[n]\}\} \tag{4.5}$$

Where *IDFT* is the inverse *DFT*, and $X_2[k]^*$ represents the complex conjugate of $X_2[k]$. Therefore, the correlation, R_i, of the i^{th} input signal block, $r_{i,k}$, with the upsampled replica C/A code, $G_{i,k}$ can be performed in the frequency domain as:

$$R_i \equiv IDFT\{DFT\{r_{i,k}\} \cdot DFT^*\{G_{i,k}\}\} \tag{4.6}$$

In practice, the fast Fourier transform (FFT) is used to compute the DFT. [vanNee91] was one of the first publications that documented this fast correlation technique applied to the GPS signal.

Figure 4.5 shows the result of a fast correlation based acquisition performed on SV10 by spanning the carrier frequency ±10 kHz about the 1.25 MHz IF center frequency in steps of 1 kHz.

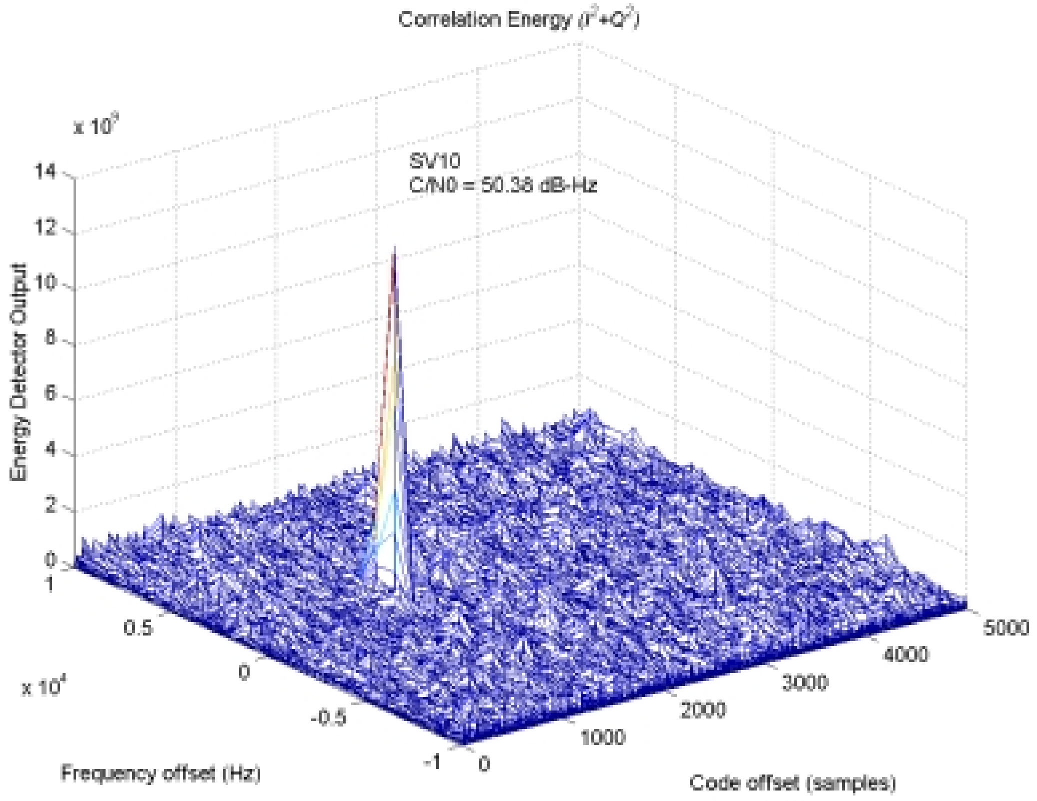


Figure 4.5 Fast Correlation Based GPS Signal Acquisition

The figure shows the triangular correlation peak in the code-offset direction when the carrier frequency approximately matches that of the incoming signal. Since only the carrier frequency dimension of the acquisition space [Figure 3.5] needs to be searched, and the code phase is immediately apparent from the location of the triangular correlation peak, the acquisition time is theoretically reduced by a factor of 2046. A block diagram of the fast correlator is shown in Figure 4.6.

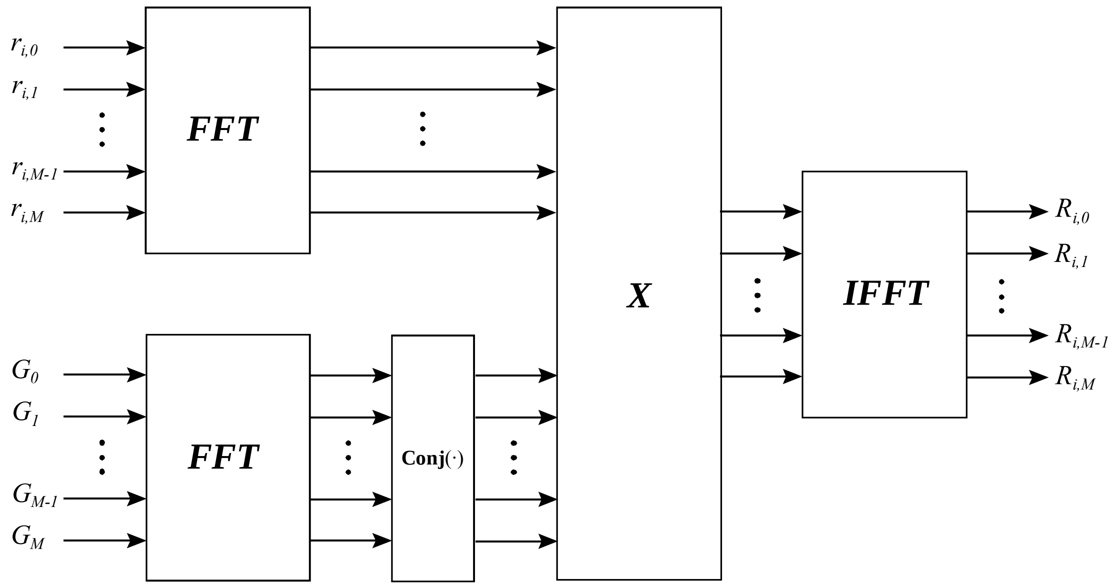


Figure 4.6 Block Diagram of GPS Fast Correlator

It should be noted here that any suitable transform that operates on an entire block of samples could be used to implement the fast correlation operation. The FFT is used in this work since it is the most well understood and straightforward technique.

The primary goal of this work was to study the feasibility of implementing the FFT based GPS fast correlator in FPGA hardware for real time processing. A minimal real time processing solution can be achieved only if the FFT/IFFT pair and multiplication operations can be performed within the dwell time (block update time) of the receiver.

4.3 Block Processing Hardware Performance Measures

As mentioned before, the goal of this work was to study the feasibility of implementing block-processing techniques in FPGA hardware. Since all of these algorithms had been developed, tested, and proven in the Matlab™ programming environment as documented in [UijtdeHaag99] and [Feng99], the double-precision processed results of Matlab™ are considered the 'truth' in this work. When these algorithms are migrated to finite-precision hardware such as that implemented in an FPGA, dynamic range limitations and rounding effects cause significant errors that result in deviations from the truth. In order to characterize these deviations and determine if these are still within acceptable limits for a given application, hardware processing performance measures were defined for this research. These performance measures evaluate how well a given hardware architecture can acquire the GPS signal and track its code phase. The ability to track carrier phase and carrier frequency were not studied since these were beyond the scope of this work. The following subsections describe the acquisition and code tracking performance measures, respectively.

4.3.1 Acquisition Margin and Implementation Loss

In order to characterize the relative amplitude of the signal correlation peak with respect to false correlations due to signal noise and finite precision round off noise, the Acquisition Margin, M_{Aq}, is defined. Figure 4.7 shows the correlation peak verses code phase for a 50 dB-Hz signal. Figure 4.8 shows the correlation of a 44 dB-Hz signal in the same data set (plotted in the same scale).

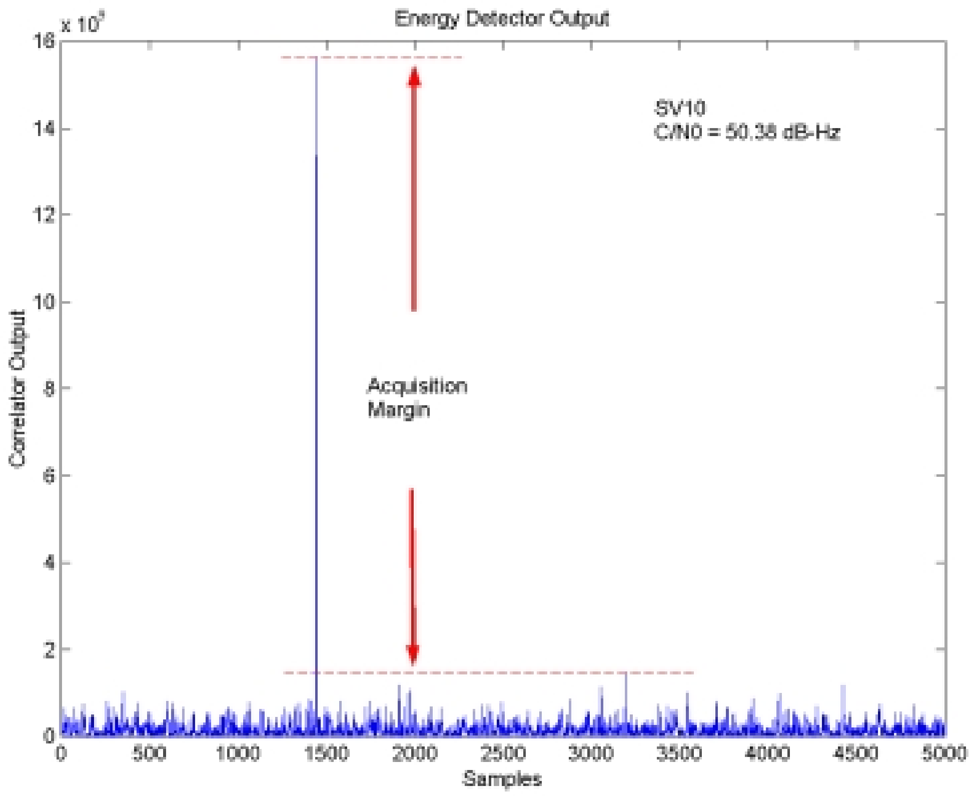


Figure 4.7 Correlation Peak for a 50 dB-Hz Signal

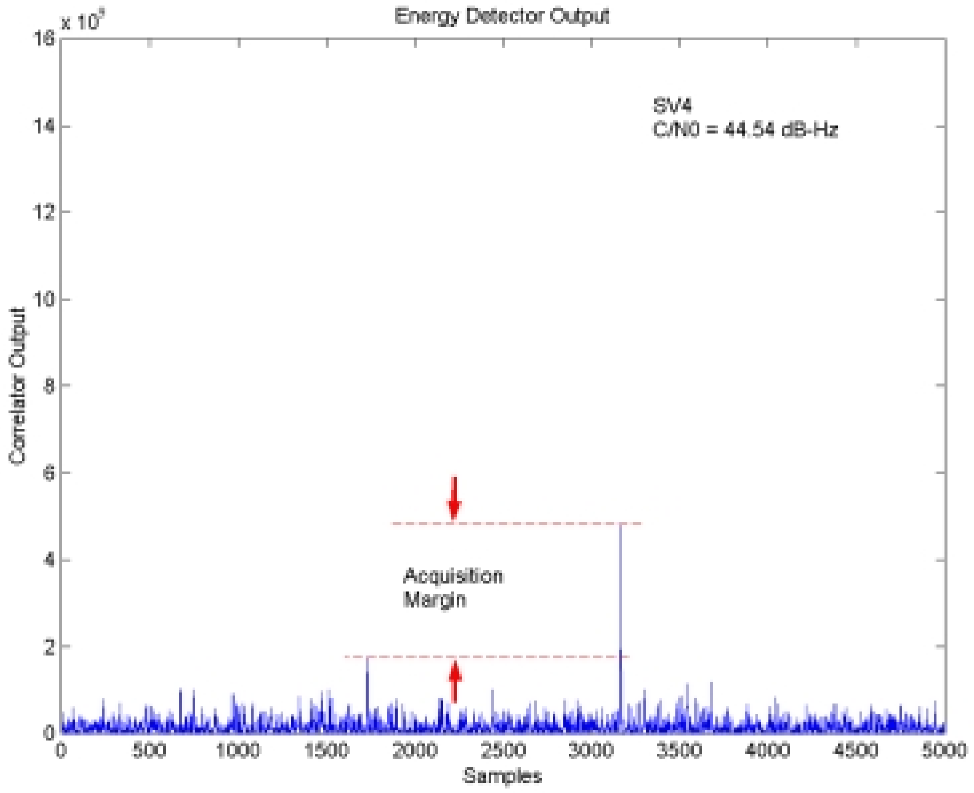


Figure 4.8 Correlation Peak for a 45 dB-Hz Signal

In order to detect the acquisition of *both* signals, the detection threshold [Section 3.3] must be set below the peak value of the weak signal, but above the highest noise peak. The acquisition margin is a measure of the amount of 'headroom' available in order to set the detection threshold, $T_{D,}$ such that an acquisition with reasonable P_{FA} and P_{MD} values is attainable. The acquisition margin is defined as:

$$M_{Aq} = 10 \cdot \log\left(\frac{R_P}{R_N}\right) \quad (dB) \tag{4.7}$$

Where:

R_P	:	amplitude of the correlation peak
R_N	:	amplitude of the largest noise peak in the correlated block

Therefore, the acquisition margin for the 50 dB-Hz signal of Figure 4.7 is 10.62 dB. For the 45 dB-Hz signal of Figure 4.8, it is 5.42 dB.

When the correlation is performed with finite-precision arithmetic (i.e. in hardware), processing "noise" due to roundoff errors is added to the signal being processed. It is assumed that the processing noise is uncorrelated to the thermal and sampling noise components. This is a reasonable assumption since the noise sources are due to completely different physical processes. Hence, when the signal is correlated using finite precision arithmetic, the noise peaks increase in amplitude and the actual correlation peak decreases, hence lowering M_{Aq}. The difference between mean acquisition margins for the truth, $M_{Aq,truth}$ (which contains only the thermal and sampling noise components), and hardware processing, $M_{Aq,HW}$ (which contains the processing noise in addition to the noise present in the truth), gives the implementation loss, L_{HW}, for the hardware architecture under consideration, as given below.

$$L_{HW} = E\left(M_{Aq,Truth}\right) - E\left(M_{Aq,HW}\right) \tag{4.8}$$

A significantly large sample size is needed to calculate the mean value. Since a databit transition within a block drastically reduces the correlation energy and hence affects the statistical results, blocks with databit transitions are removed, as described in Section 4.1.

4.3.2 Code Phase Detection and Range Error

As described in Section 2.1, the replica code phase gives the time of transmission, t_{tr}, of the signal relative to the receiver code epoch. For non-coherent detection (where the receiver clock is not synchronized to the incoming code epoch), the location, $\tau_{P,i}$ (i.e.

the code phase offset corresponding to the prompt C/A code for block *i*), which is the maximum output of the envelope detector provides the approximate C/A code phase in samples, as shown in Figure 4.9. The envelope value is given by $Y = \sqrt{I^2 + Q^2}$.

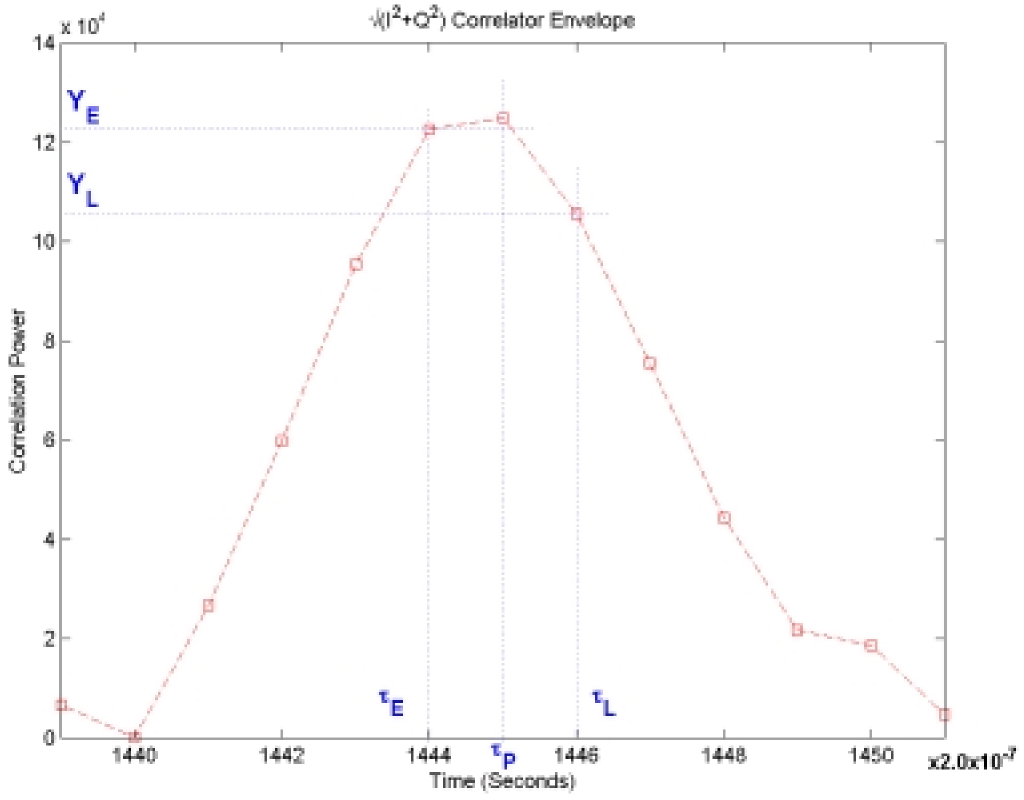


Figure 4.9 Illustration of Code Phase Measurement Parameters

Figure 4.10 shows the quantity c(τ_P), over a duration of one second for SV4 in the dataset.

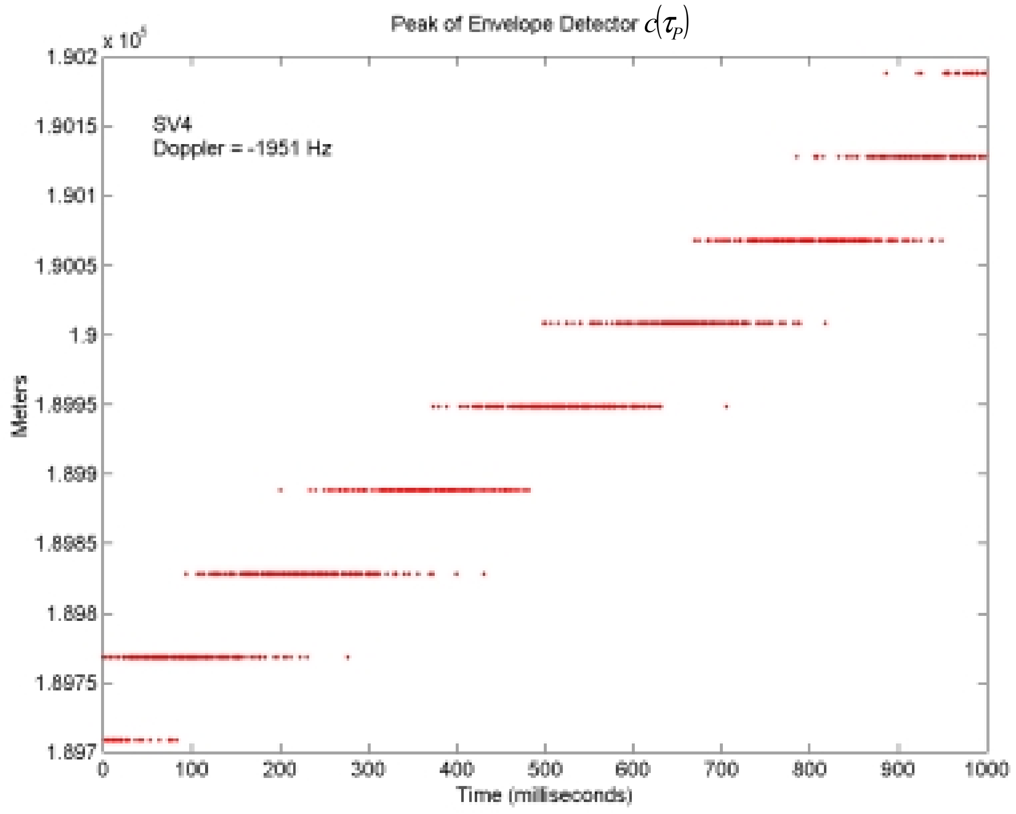


Figure 4.10 Envelope Detector Peak Location Showing Relative Motion of SV

The code phase changes over time due to the relative line-of-sight velocity, v_r, as given by Equation 2.5. Due to sampling quantization, range uncertainties, and noise, the code phase does not change smoothly from one sample point to the next. Therefore, t_{tr} is estimated from Equation 4.8 which is derived from [Tsui97] and [Feng99].

$$t_{tr} = T_S \left(\tau_P - \frac{M}{1023} \left(\frac{Y_E - Y_L}{Y_E + Y_L} \cdot \frac{2-d}{d} \right) \right) \tag{4.9}$$

Where:

d	:	correlator spacing in chips (d=0.4 chips for M=5000)
Y_E	:	envelope value when the reference C/A code is d/2 chip early
Y_L	:	envelope value when the reference C/A code is d/2 chip late

Figure 4.11 shows the quantity $c(t_{tr})$ (i.e. the time of transmission expressed as a range) estimated from Equation 4.9 for the i^{th} ms, in meters, along with a (least squares) quadratic fit of the data points. The standard deviation of $c(t_{tr})$ relative to the fit is 17.84 meters.

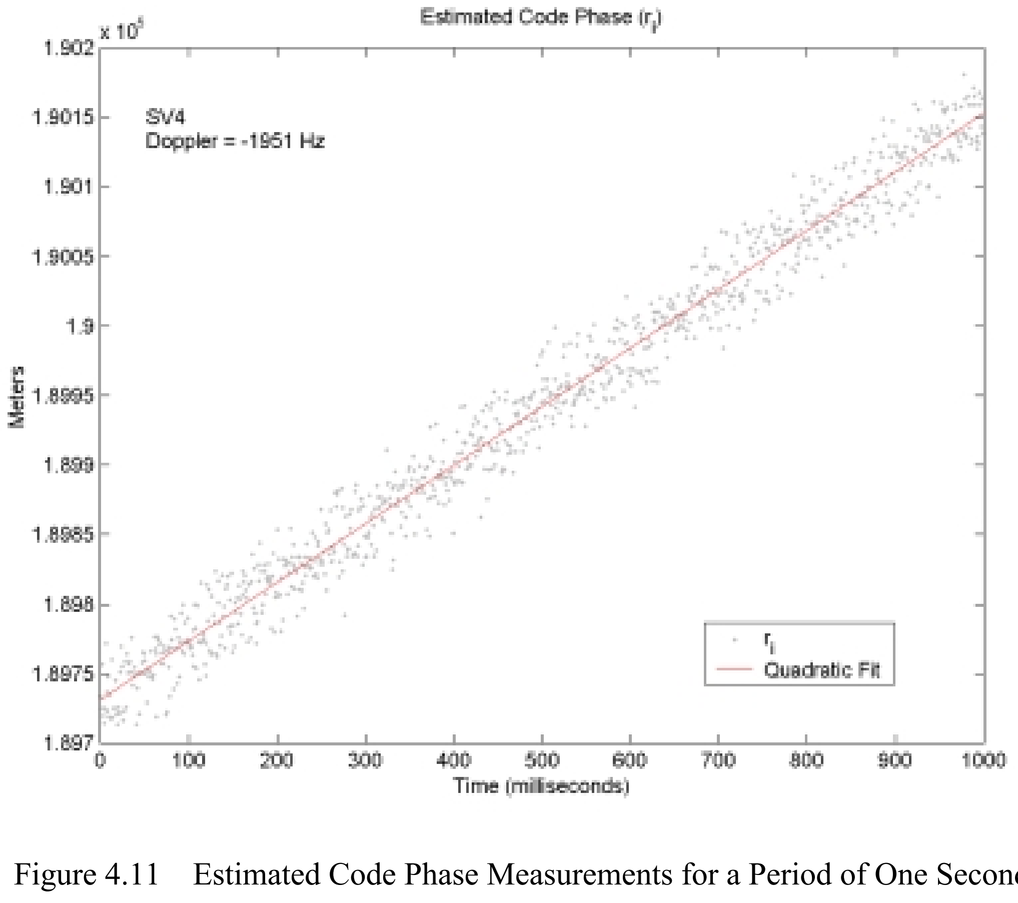


Figure 4.11 Estimated Code Phase Measurements for a Period of One Second

The standard deviation of the *$c(t_{tr})$* values with respect to the quadratic fit gives the amount of error in the pseudorange measurement, and is hence used as a performance measure to evaluate the amount of pseudorange error that will result when the block processing techniques are implemented with finite precision hardware.

4.4 Applications of Block Processing

Block processing has been shown to improve the performance of GPS under severe conditions such as when the received signal is too weak to be detected, the signal is received in high-dynamic environments, and in the presence of various types of interference. In addition, block processing can be used for the sophisticated receivers that are used in LAAS ground facilities (LGFs) to monitor the GPS signals and detect anomalies, in order to increase the reliability of a GPS based landing system [Snyder99 and Feng99]. An overview of one of these applications, acquiring low C/N_0 GPS signals, is presented below.

As shown in Table 4.1, SV5 is present in the dataset used for this work, but is too weak to be acquired by the NovAtel™ Millennium™ GPS receiver. Hence, it is assumed that SV5 has a C/N_0 lower than 44 dB-Hz. Figure 4.12 shows the block processing acquisition performed for SV5.

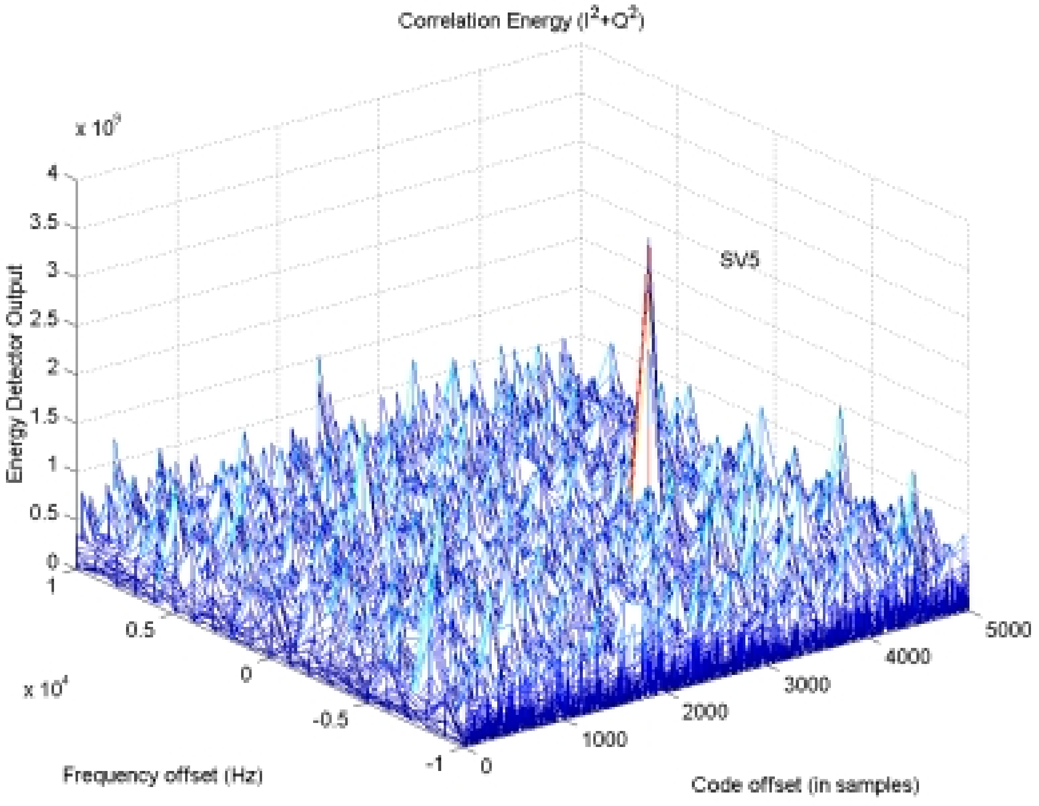


Figure 4.12 Acquisition of a weak GPS signal

When compared to the 50 dB-Hz signal acquisition of Figure 4.5, it can be seen that the correlation peak is too weak to detect reliably. Figure 4.13 shows the peak at the Doppler frequency offset of –1351 Hz (the average Doppler offset for the SV, as shown in Table 4.1). The acquisition margin for this case is only 3.9 dB.

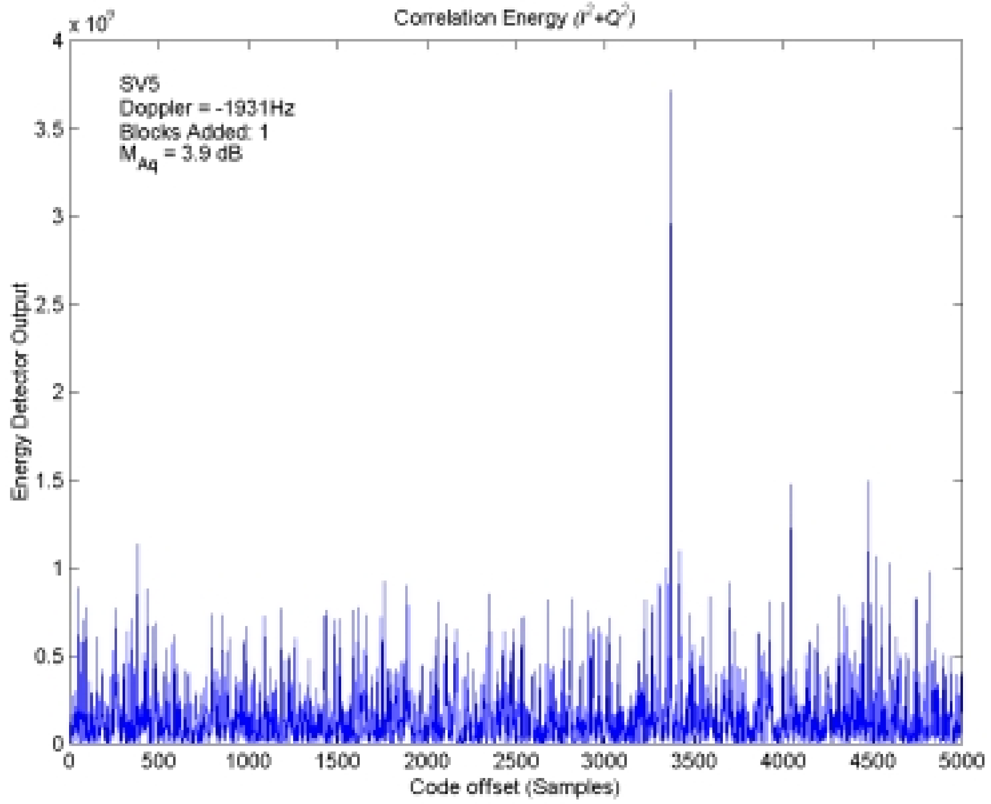


Figure 4.13 Fast Correlator Output for Weak GPS signal

Figure 4.14 shows the correlation result when five concurrent fast correlated blocks are coherently summed. The block addition process averages out the noise and increases the magnitude of the correlation peak. This technique increases the acquisition margin to 8.2 dB, roughly double that of the previous case, and greatly improves the detectability of the signal.

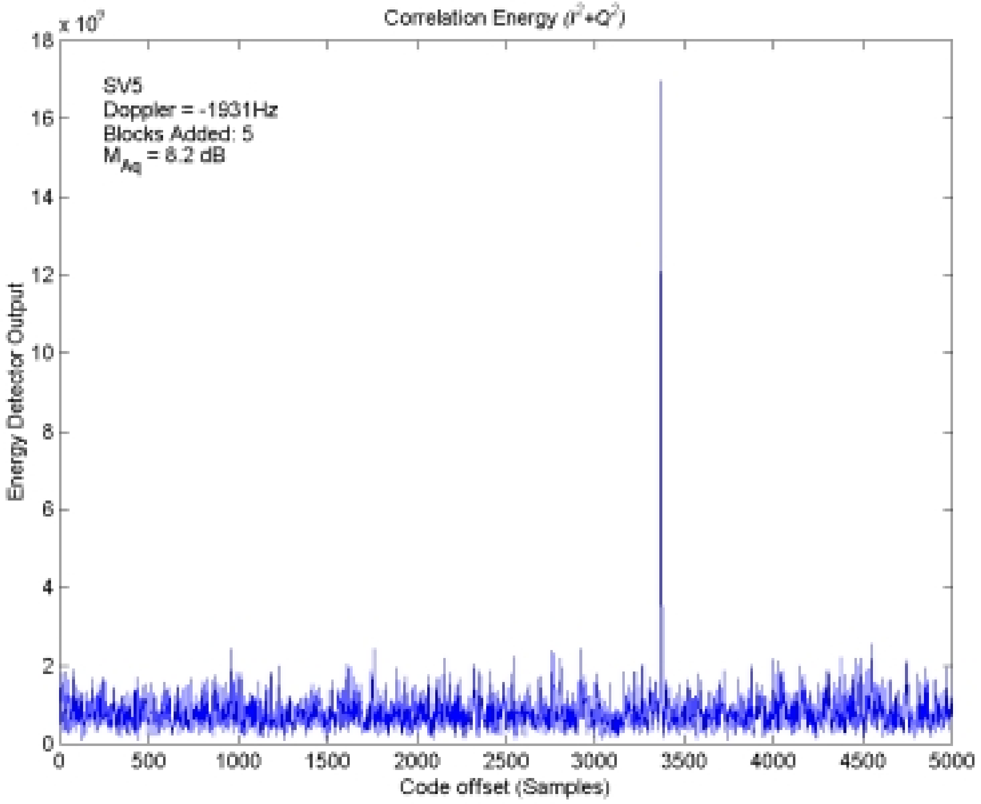


Figure 4.14 Result of Block Addition Technique Applied to Weak GPS Signal

Techniques such as this can be used to acquire GPS signals with very low C/N_0, and has been proven to acquire signals as low as 20 dB-Hz [UijtdeHaag99]. However, this technique can be used for positioning only if the receiver has *a-priory* information such as carrier Doppler offset and the GPS data message. The SnapTrack™ server aided positioning system [Moeglein98] uses a datalink to send these parameters to the receiver so that the receiver only needs to find the time of transmission based on the block added correlation peak. Since block addition techniques effectively increase the dwell time of the correlation, a receiver optimized for receiving weak signals will lack the ability to operate well under high-dynamic conditions. Further details of block addition techniques for receiving weak GPS signals can be found in [UijtdeHaag99].

4.5 Real Time Block Processing GPS Receiver

The previous section mentioned the various applications of a block processing GPS receiver. The algorithms for these applications have already been developed and tested in software, as described in [Feng99] and [UijtdeHaag99]. However, the immense computational burden of these algorithms has kept them from being implemented in real time. The main obstacle for real-time processing has been the computation of the 5000-point FFTs and IFFTs needed for the fast correlator. Even with the latest generation of multiprocessor-based computers, the best achievable processing throughput has been roughly two orders of magnitude lower than what is required for a real time solution. Even if general-purpose microprocessors eventually become fast enough to block-process GPS at a useful rate, such a solution would not be able to complete with current GPS receivers in terms of size, power consumption and portability. The reason is that the general-purpose microprocessor performs operations on an instruction-by-instruction basis, and is therefore not optimized for any given application. On the other hand, ASICs are ‘hardwired’ for a given application and hence offer the most optimal solution in terms of speed, power and size. However, once fabricated, the ASIC cannot be changed even to accommodate a small change in the algorithm. Furthermore, the non-recoverable expenditure (NRE) associated with ASICs is too restrictive for research purposes.

FPGAs combine the best features of microprocessors and ASICs and are hence the implementation platform of choice for a block processing GPS receiver.

Figure 4.15 shows a proposed system-level diagram of a real-time block processing GPS receiver that targets airborne applications.

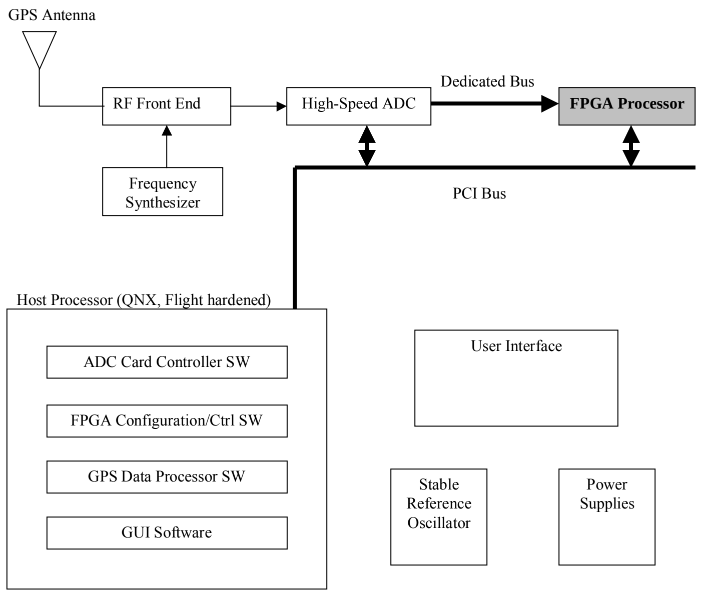


Figure 4.15 System Diagram of Real-Time Block Processing GPS Receiver

The entire system is housed within a flight hardened airborne computer that runs a flight certified real time operating system (RTOS) for high reliability. The RF front-end module (a miniaturized version of the scheme in Figure 4.2) amplifies and downconverts the GPS signal. A high speed ADC card attached to the host system bus (PCI) samples the downconverted signal at a quantization that is high enough (i.e. ~ 12-bits), so that AGC is not needed due to the high input dynamic range. Because of the high input quantization and the sampling rate (5 MHz), the data throughput of the ADC will be too high to send reliably via the system bus. For example, if 12-bit data is packed into a 16-bit integer, the throughput will be 10 Mbytes/sec. Hence, a dedicated bus is used to route the data directly to the FPGA processor. This ensures a sustained data transfer and helps minimize latency in the system. The host processor controls the operating parameters of the ADC card via the system bus.

The FPGA hardware processes the incoming data in a block-by-block basis, as illustrated in Figure 4.1. The main task of the FPGA processor is to perform the GPS fast correlation presented in Section 4.2. The output of this stage is the pseudorange, delta-pseudorange, and accumulated Doppler measurements, similar to the sequential receiver shown in Figure 3.1. The configuration and control of the FPGA processor is performed by the host processor via the system bus.

The host computer runs a multitasking RTOS that handles the following tasks: 1) setup of the ADC during startup and active monitoring of ADC card for intelligent AGC and fault monitoring. 2) FPGA configuration and embedded processor software uploads at power up and runtime based on the given application (i.e. dynamic reconfiguration). 3) Intelligent navigation processing based on decoded navigation data stream, pseudorange, delta-pseudorange, and accumulated Doppler values obtained from the FPGA processor, and supplemental positioning systems (such as an IMU). 4) Graphical user interface (GUI) software execution. 5) Overall system integrity monitoring.

The ultimate goal of this work was to implement the FPGA processor component of the real-time block-processing GPS receiver. The design of the FPGA processor is described in Chapter 7. The following chapter is an introduction to FPGAs.

5 FIELD PROGRAMMABLE GATE ARRAYS

Chapter 4 stated that FPGAs were the only platform currently available that was ideally suited for implementing a real time GPS block-processing receiver. This chapter introduces FPGAs and describes the Xilinx XC4000 series device as a representative architecture. In addition, it highlights some of the unique features FPGAs possess that make them ideal for DSP applications.

5.1 Introduction to FPGAs

FPGAs are the newest addition to the programmable logic device (PLD) family. In addition to FPGAs, PLDs comprise of simple programmable logic devices (SPLDs) and complex programmable logic devices (CPLDs). Of these, FPGAs are the most complex devices in terms of density, architecture, and functionality. They are suitable for implementing designs of varying complexities, ranging anywhere from a thousand up to several million system gates. In terms of density and speed, FPGAs have come of age in recent years, and are beginning to replace conventional ASIC technologies in select applications such as networking hardware. Perhaps the most important reasons FPGAs have become a popular platform for digital design are its rapid and infinite re-programmability, high gate density and speed-to-power ratio and low cost. Re-programmability has also made it a popular device for research where FPGAs are used for rapid prototyping.

Because FPGAs approach the performance of ASICs while incorporating the flexibility of software reconfiguration, it is the ideal architecture for implementing a software radio. This is why FPGAs were the focus in this work. That said, it should be noted that the architectures presented in this thesis could also be implemented in an ASIC technology since the architectures are described at a high-level of abstraction and are not technology specific.

During the course of this work, an FPGA based commercial off-the-shelf (COTS) design platform known as the PCI Pamette™ was used for implementing digital designs [Compaq97]. The Pamette board consists of four Xilinx XC4010E-HQ208 FPGAs, each containing 10,000 maximum usable gates available for reconfiguration. Even though new and better architectures such as the Virtex™, Virtex-E™ and Virtex-EM™ families of FPGAs have emerged, they are still largely based on the XC4000 series architecture, which Xilinx™ introduced in 1991 [Roelandts99]. As such, this thesis focuses on the XC4000 device as a possible implementation candidate. The following section presents a brief introduction to the XC4000 architecture.

5.2 Architecture of Xilinx XC4000 Series FPGAs

As shown in Figure 5.1, an FPGA consists of an array of programmable logic structures known as configurable logic blocks (CLBs). Along the periphery of the device are structures that control the input and output to the device. These input/output logic structures are called I/O blocks (IOBs). All blocks are enveloped by a rich combination of routing resources known as the programmable interconnect. The CLBs implement the logic functions of the hardware. The IOBs provide connectivity to and from the outside world. The programmable interconnect routes the connections between the blocks to make up the digital circuit. The CLBs, IOBs, and interconnect are configurable via SRAM based memory cells in the device. The interconnections are made using pass-transistor switches with repeaters (amplifiers) placed at suitable intervals to restore logic levels and minimize delay for long signal paths.

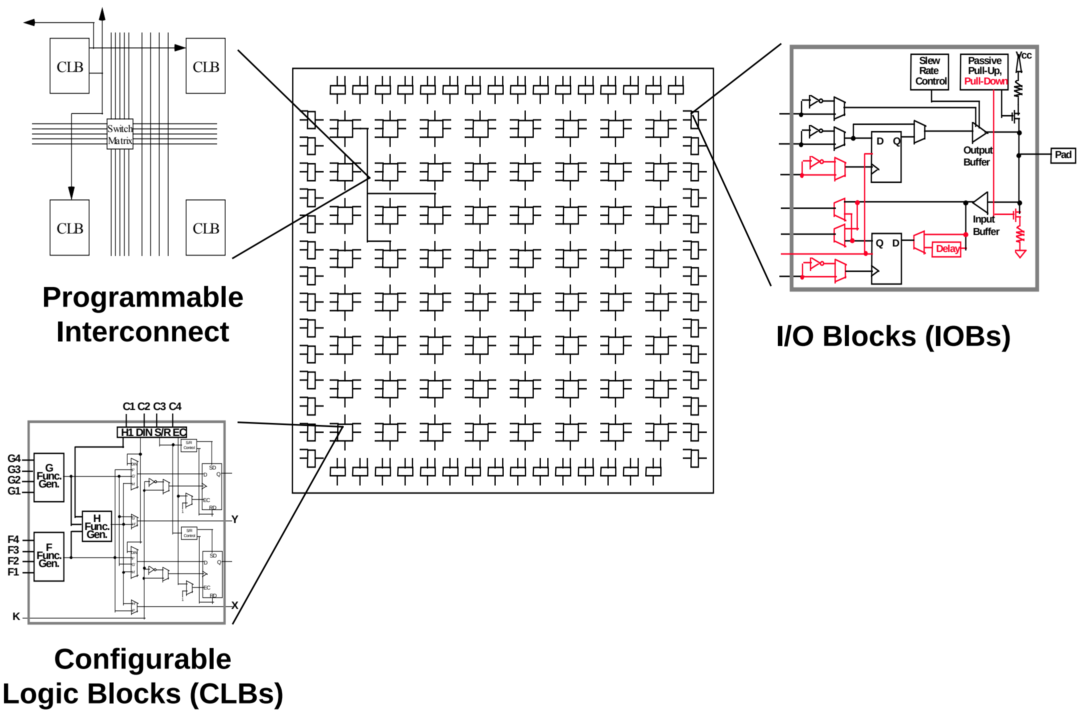


Figure 5.1 Anatomy of an FPGA [Xilinx99]

Digital systems are implemented by mapping the design into the framework of the FPGA architecture. During the design phase, many solutions to this mapping problem can result due to the generic architecture of the FPGA. The challenging task of FPGA design lies in the efficient structuring of the design logic such that the device and its architectural features are best utilized to yield an optimal implementation solution. This philosophy sets FPGA design and ASIC design distinctively apart from each other. For an ASIC, the goal is to implement the given logic using the lowest transistor count, physical area, and effective switched capacitance as possible (to reduce propagation delay and power consumption). When an FPGA is the target device, the designer must thoroughly study the architecture of the device and picture the logic in terms of the resources available in the FPGA. For example, a commutator switch in a pipeline is best implemented with pass transistor logic in an ASIC technology. For an FPGA, the same function is efficiently implemented using dual port RAM (DPRAM), which is abundant in the device.

Figure 5.2 shows the architecture of a CLB. The CLB is comprised of two 16×1-bit and one 8×1-bit SRAM based look-up-tables (LUTs), called function generators, that can be combined to form universal logic functions of up to nine inputs. In addition, the outputs of the F and G function generators can be registered within the CLB’s two register elements, which, depending on device, can be configured to be either edge-triggered D-type flip-flops (DFFs), or level-triggered latches. This logic-register structure is the basis for implementing bit-sliced pipelined datapaths necessary for most DSP functions.

Apart from the ability to implement fixed logic functions through the use of look-up-tables, special architectural features of the XC4000 enable the function generators to be configured as RAM, DPRAM, or high-speed adders (through the use of fast carry logic structures that traverse vertically through CLB boundaries). Since such higher order functions can be implemented in a single CLB, this type of architecture is referred to as one that is ‘coarse-grained’. This coarse-grained architecture made the XC4000 family one of the first and popular devices for implementing DSP hardware.

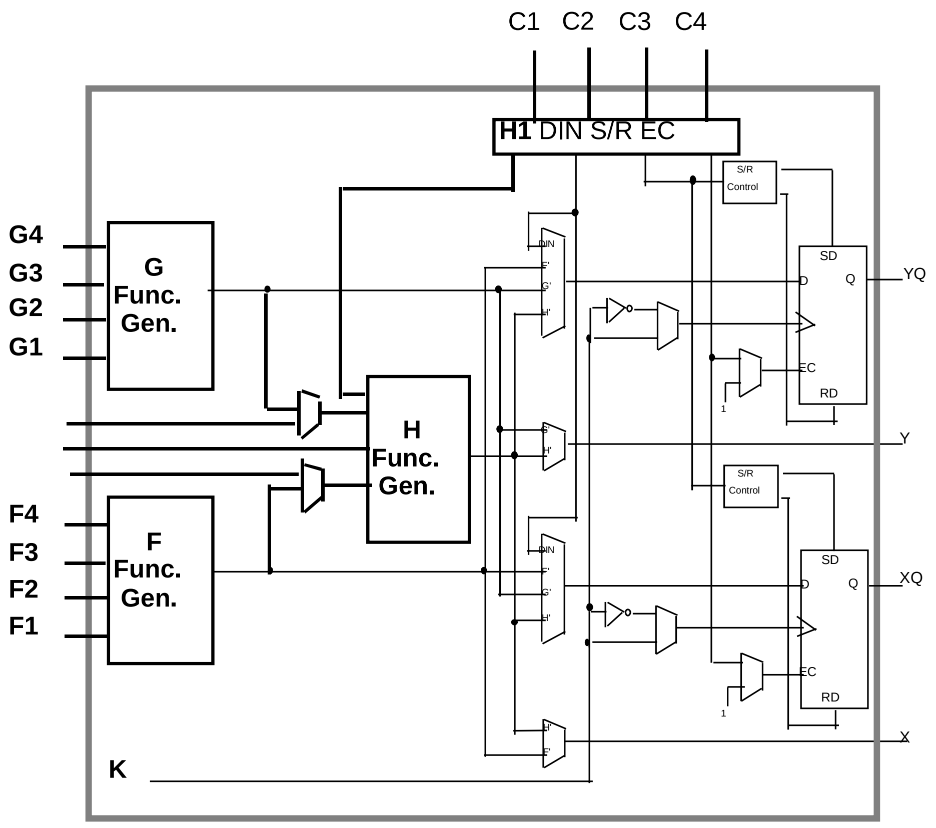


Figure 5.2 XC4000 Configurable Logic Block [Xilinx99]

Figure 5.3 shows the internal structure of the XC4000 IOB. Each IOB controls one package pin. The pin can be an input, output, or bi-directional depending on how the IOB is configured. In addition, the inputs and outputs can be registered. This enables the device to meet fast I/O throughput requirements when mated with external pipelined hardware. Each signal pin can also be tri-stated. This feature is invaluable when the device needs to be connected to bi-directional data busses.

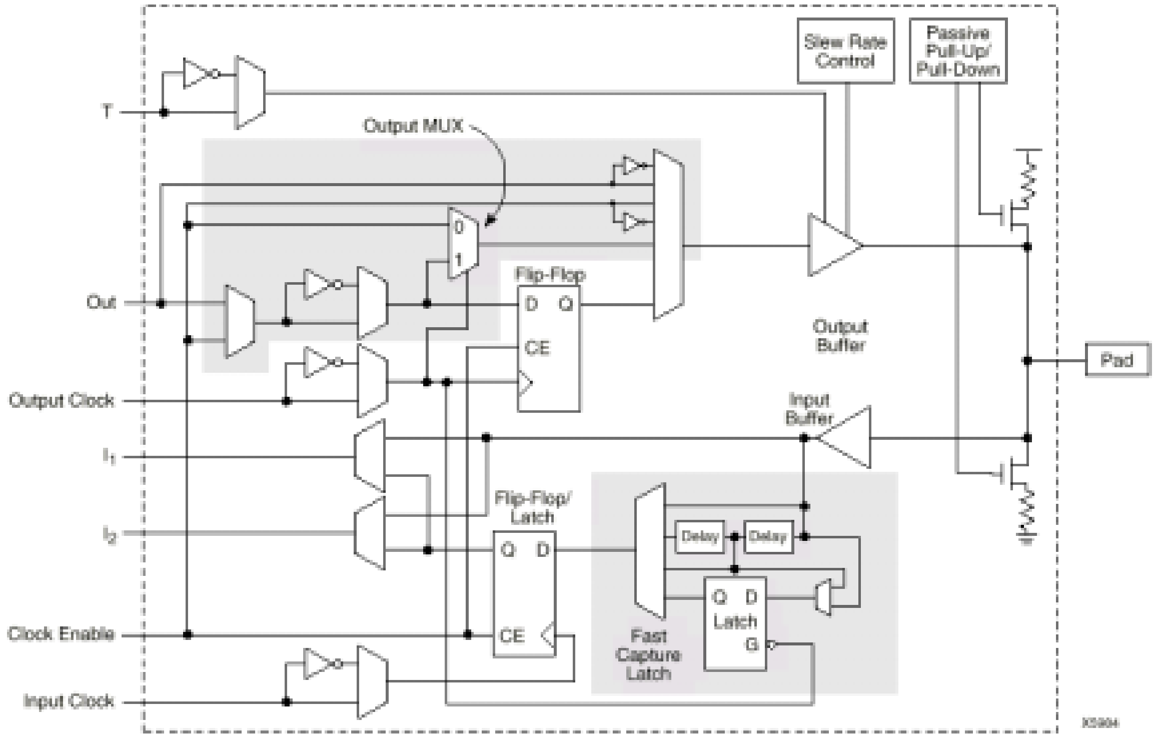


Figure 5.3 XC4000X IOB (Shaded Areas Indicate Differences from XC4000E) [Xilinx99]

Figure 5.4 illustrates how programmable routing is achieved through switching matrices within the FPGA. Each interconnecting node within the switch matrix has six pass transistors to route connections to and from any direction. Several types of routing resources comprise the FPGA's complex routing structure. These include short (and fast) CLB-to-CLB routing, general purpose interconnect that use the switch matrices, and low-capacitance long lines spanning vertically and horizontally across the chip to establish longer connections. The horizontal long lines can be configured as tri-stated busses for sending data to and from their adjacent rows of CLBs to IOBs. In addition, dedicated metal layers driven by powerful buffers (BUFGs, [Xilinx99]) are used to route low-skew clock signals to the individual registers within the CLBs and IOBs.

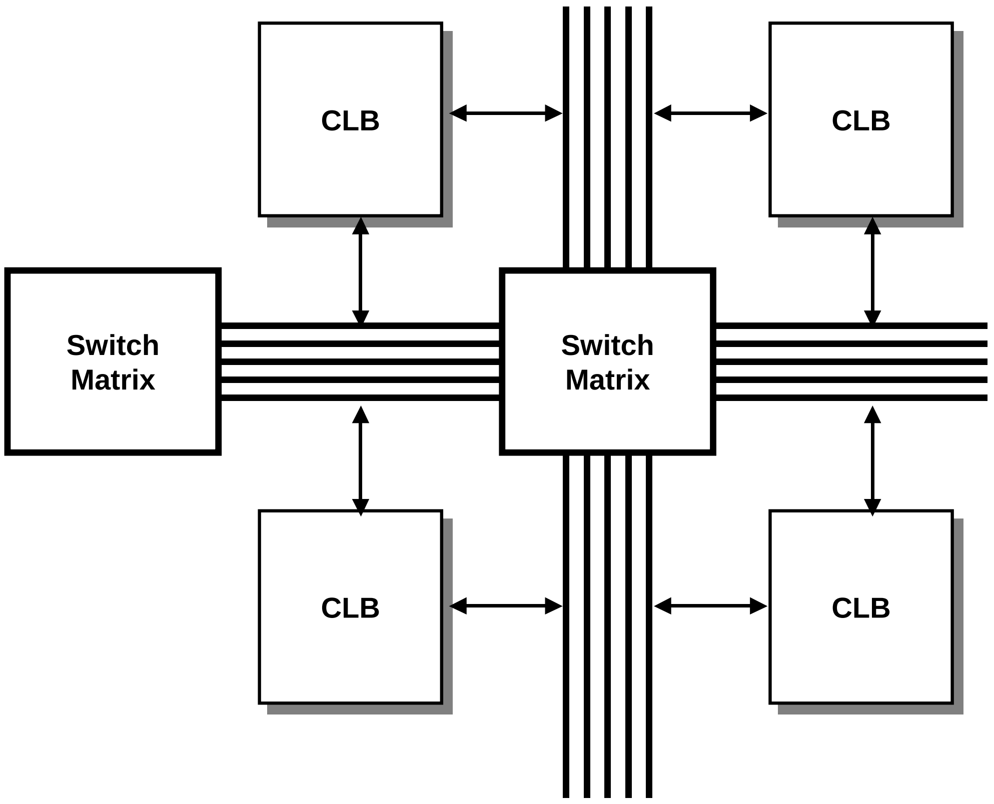


Figure 5.4 XC4000 Interconnect Scheme [Xilinx99]

5.3 FPGA Based Design Flow

As with any digital design process, FPGA based design involves a set of procedures that needs to be followed for optimum results. Figure 5.5 shows a typical FPGA design flow.

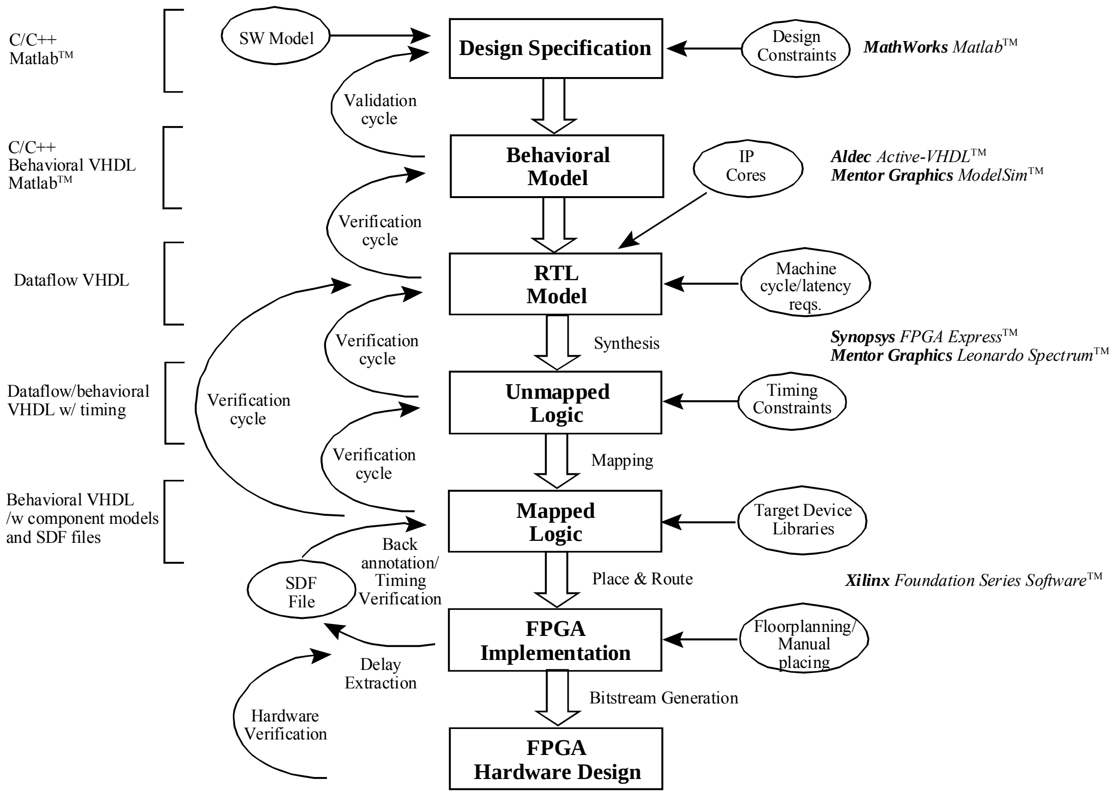


Figure 5.5 Typical FPGA Design Flow

The input to the process is a design specification that usually includes a software model of the system that needs to be implemented in hardware. The specification includes design constraints such as overall system error bounds, size, power consumption, cost, etc. The hardware designer starts by developing a model that simulates the behavior of the hardware (hence called the *behavioral model*) and validates it against the design specification. Once the behavioral model is fully validated to meet the constraints of the design specification, subsystems within the behavioral model are replaced with register transfer level (RTL) models. RTL models describe the system in terms of its dataflow and the operations to be performed with respect to machine (clock) cycles. In order to simplify the design process, pre-designed intellectual property (IP) cores can be integrated into the design at this stage. Examples of IP cores include PCI bus interfaces, Dynamic RAM (DRAM) controllers, and various DSP functional blocks. The RTL models are then verified for correct functionality with respect to the behavioral model. Following the verification process, the RTL description of the system is synthesized into a circuit comprising generic hardware elements such as NAND/NOR gates and DFFs. This is normally performed with a synthesis tool. At this stage, the logic elements do not necessarily have a correspondence to components in the actual target hardware. The gate-level design is then verified against the RTL model for proper functionality. The synthesis tool, in its effort to minimize the speed-area product, can sometimes incorrectly merge or "optimize-away" necessary functionality. Verification with the RTL model reveals these errors.

Following synthesis, the generic elements are mapped to the given FPGA device. The mapping tool has access to a database of the target device's components, their configuration rules, fan-in and fan-out limits, delay properties, hardware cost, etc. The mapped result is then verified against the unmapped or RTL models for proper functionality. The mapped design is then inputted to the FPGA development tools for placement and routing.

The place-and-route tools repeatedly try different placement and routing schemes until it finds a solution that meets the given timing constraints. The tool then generates a file containing information about the FPGA's internal interconnect delays. This file is generated in an industry standard called the standard delay format (SDF). The SDF file is back-annotated to the mapped hardware description and simulated for timing compliance.

Left to its own devices, the place and route tool can fail to find a configuration that meets the given timing constraints. In this case, the designer can manually place components in critical paths, or arrange all components so that the tool has only to route the design. This latter procedure is called floorplanning, and is an important step for harnessing peak performance from an FPGA design. Depending on how seriously the place-and-route step failed, the designer may have to go all the way back to the synthesis phase, or even the RTL design phase and iterate many times in order to get an optimal design that passes timing constraints. If timing compliance is impossible to achieve, some timing constraints may need to be relaxed for slower performance, or the target device may need to be changed to one with a higher speed grade.

Once the design passes timing verification, it is ready to be programmed into the FPGA. The design tools output a configuration bitstream to program the FPGA. A design completed in this way usually works first time in the target hardware. This is due in part to the thorough cross-verification between successive stages in the design, and the conservative unit delay values used for the SDF files. However, final hardware verification using digital test and measurement equipment in a variety of operational extremes (such as temperature, vibration, radiation, etc.) is essential to insure that the hardware will function as per the original design specification. Figure 5.5 shows typical languages and software tools used for the various stages of the design process.

In this work, the complete high-level behavioral model for the GPS fast correlator was developed and validated against the design specifications obtained from [Feng99] and [UijtdeHaag99]. The behavioral simulation was performed in Matlab™ by effectively 'crippling' its floating-point capabilities to mimic the functionality of the hardware. Matlab™ rather than a hardware description language such as VHDL was chosen for behavioral simulation because initial VHDL simulations took too much time to run, even for a relatively small subset of the entire design. The design validation process was performed for several hardware parameters as documented in Chapter 8. Due to time limitations, a complete translation to the RTL level was not performed. Instead, design partitioning schemes for eventual RTL conversion was studied, and is documented in Chapter 6 and Chapter 7.

6 DESIGN OF 5000 POINT FFT/IFFT

As described in Section 4.1, at the heart of the GPS fast correlator is the computation of 5,000-point FFTs and IFFTs. This chapter describes the development of the 5000-point FFT/IFFT algorithm used for this work.

6.1 The Fast Fourier Transform Algorithm

The DFT for an N-point finite duration sequence *{x(n)}*, where $0 \leq n \leq N\text{-}1$ is given by:

$$X(k) = \mathop{}\limits_{n=0}^{N-1} x(n)e^{-j(2\pi/N)nk} \qquad k = 0,1,...,N-1 \tag{6.1}$$

This can be tersely written as:

$$X(k) = \mathop{}\limits_{n=0}^{N-1} x(n)W^{nk} \tag{6.2}$$

Where $W=e^{-j(2\pi/N)}$. When *x(n)* is a complex sequence, the direct evaluation of the DFT requires $(N\text{-}1)^2$ complex multiplications and *N(N-1)* complex additions.

In 1964, J.W. Cooley and J.W. Tukey rediscovered an optimization to the DFT that revolutionized its use in signal processing applications [Cooley65]. In short, the optimization involves breaking up the original N-point sequence into two *N/2* sequences and evaluating the DFT for each and combining the result to obtain the original N-point transform. The shorter sequences now require on the order of $(N/2)^2 \times 2 = N^2/2$ complex multiplications, a factor of two savings over the direct evaluation. This splitting process can be iterated to end up with *N/2* 2-point transforms. A 2-point transform requires only two complex additions and no complex multiplications. This procedure requires *log_2N* splits and *$N/2$* complex multiplications to combine the results of the individual stages. Hence, the total multiplications needed is approximately *$(N/2)log_2N$*. This is an approximation because some of the multiplications are trivial and do not actually require multiplication operations. However, as *N* becomes large, the number of non-trivial multiplications approaches this value.

The method described above can be performed only if *N* is a radix-2 number. This is by far the most popular and optimal FFT algorithm. To apply the radix-2 FFT for non radix-2 sequences, various techniques such as zero padding is often used. This results in artifacts that may or may not be acceptable depending on the application [Madisetti98]. Such techniques are unacceptable for the circular convolution based fast correlator described in this work because the artifacts generated would interfere with the precise ranging measurements that need to be made with the correlation peak.

The same split-and-combine technique used for the radix-2 FFT can be applied to non-radix-2 sequences. However, this requires the calculation of non radix-2 transforms and the combination of mixed-radix transforms to obtain the final transform. To implement the 5,000-point FFT, radix-2, radix-4 and radix-5 transforms were used. Sections 6.2, 6.3, and 6.4 describe these building block FFTs respectively, and Section 6.5 describes how they are combined to realize the 5000-point mixed-radix FFT.

6.2 The Inverse FFT

Consistent with the notation of Equation 6.2, the inverse discrete Fourier transform is given by:

$$x(n) = \frac{1}{N}{}_{k=0}^{N-1} X(k)W^{-nk} \tag{6.3}$$

The IDFT can be computed using the same algorithm steps by replacing the original twiddle factors (i.e. W^{nk} coefficients) with their complex conjugates.

In the case of the fast correlator used in this work, the FFT hardware implementation introduces a scaling factor, *1/S* to the result. The IFFT is implemented by the same hardware structure used for the FFT merely by conjugating the twiddle factors as described above. Hence, the IFFT is also scaled by the *1/S* factor. For the purposes of hardware implementation, the *1/N* scaling of the IFFT is not performed. The result of the correlation can be scaled to give the correct absolute value since the hardware scaling factors are known. However, this is not done for the GPS fast correlator since only the relative magnitudes of the peaks are of interest.

The following sections describe the design of the 5000-point FFT/IFFT mixed-radix algorithm that was used for the fast correlator. The mixed radix algorithm is constructed from radix-2, radix-4, and radix-5 FFT building blocks. As such, these will be described first, followed by the 5000-point algorithm and how it is folded for implementation in FPGA hardware. For simplicity, only the FFT is considered in the following description.

6.3 Radix-2 FFT Building Block

The radix-2 FFT represents the most fundamental element used for the computation of larger FFTs. This algorithm is known as the *butterfly* because of its distinctive structure. The signal flow graph for a radix-2 complex FFT is shown in Figure 6.1, where *a(n)* represents the n^{th} time domain component and *A(k)* represents the k^{th} frequency domain component, and the subscripts *R* and *I* represent the real and imaginary components respectively.

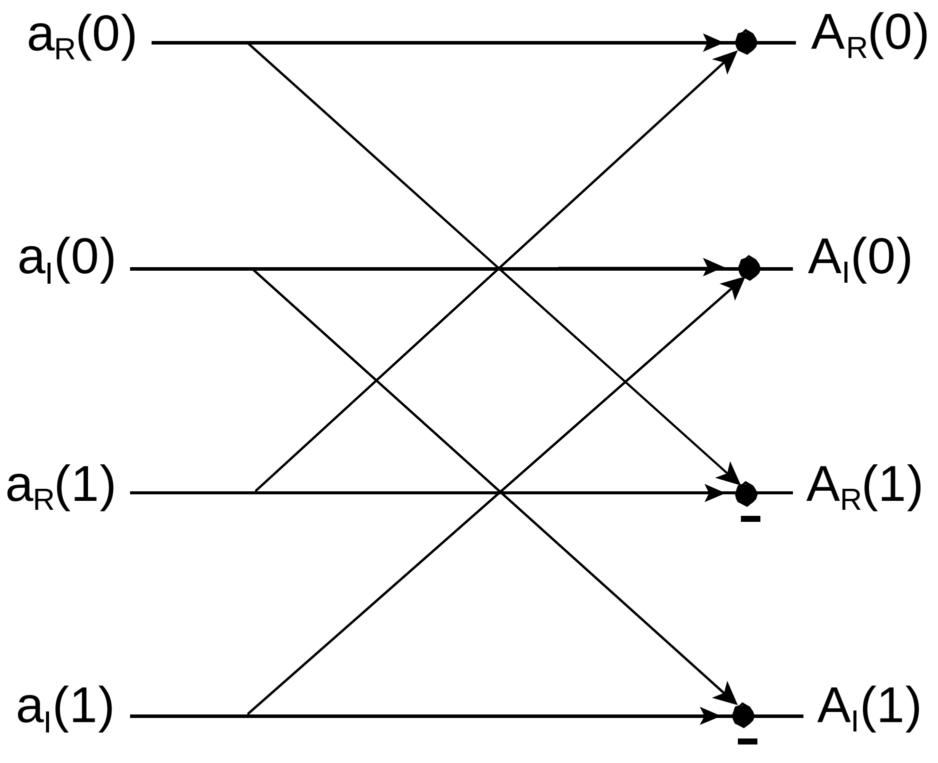


Figure 6.1 Radix-2 FFT Signal Flow Graph

The nodes in Figure 6.1 represent additions, and the – sign represents a subtraction. For a subtraction operation, the horizontal arrow always represents the subtrahend.

Figure 6.2 shows the signal flow graph for the radix-2 butterfly when it is implemented in finite precision hardware of width *W*, where *W* represents W-bit signed binary numbers in 2's complement form. In order to preserve a W-bit hardware pipeline and ensure that the computed values do not overflow the W-bit range, scaling is performed prior to sending the result of the computation to the subsequent stage. The scaling is a divide-by-two operation that is performed by shifting the *W+1* bit word one place to the right and rounding the result (the scheme for rounding in hardware is described in Section 7.2). However, depending on the final architecture, the scaling may or may not be required for a given stage. Hence, it is shown here as a switchable operation. In the final hardware implementation, the scaling operation will be hard-wired into the architecture.

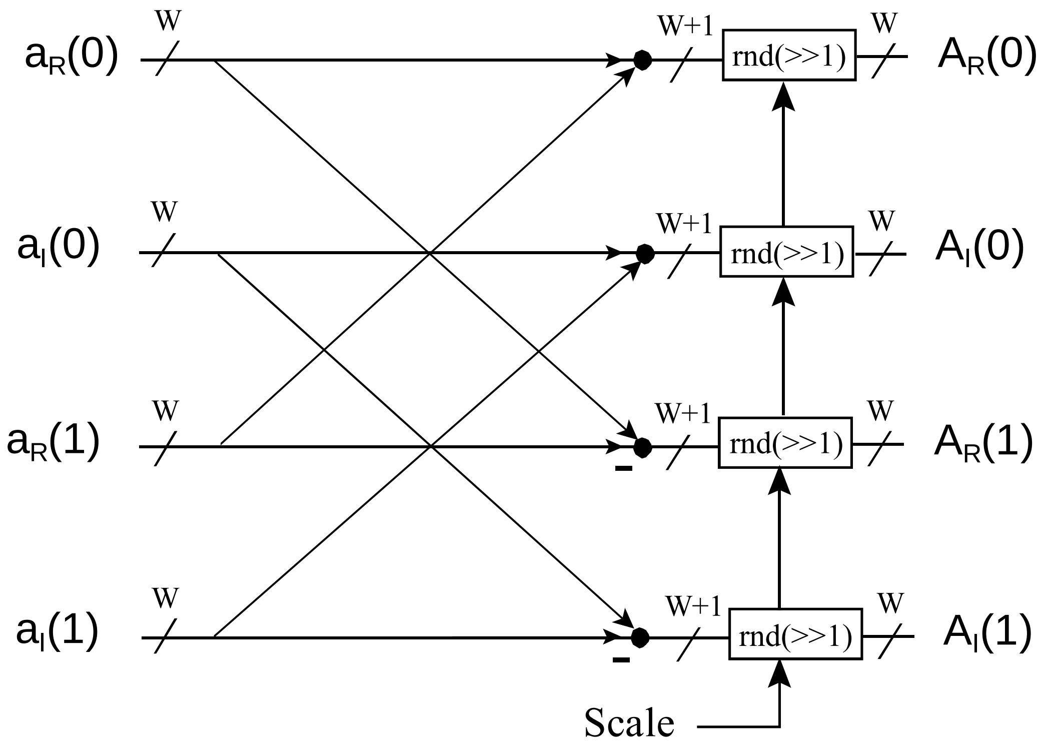


Figure 6.2   Radix-2 FFT Computation in Hardware

6.4   Radix-4 FFT Building Block

The 4-point FFT computed with Equation 6.2 requires no complex multiplies and 12 complex additions for a total of 24 complex additions. The circular convolution, complex conjugate symmetry, and 90° and 180° symmetry optimization approaches to algorithm construction all lead to the same algorithm whose signal flow graph is presented in Figure 6.3 [Smith95]. The radix-4 FFT algorithm requires a single multiplication by *j*, which is performed without a multiplication operation by crossing $b_i(3)$ and $b_r(3)$, as shown in the figure. Because of its distinctive structure, the radix-4 FFT element is also known as the *dragonfly*.

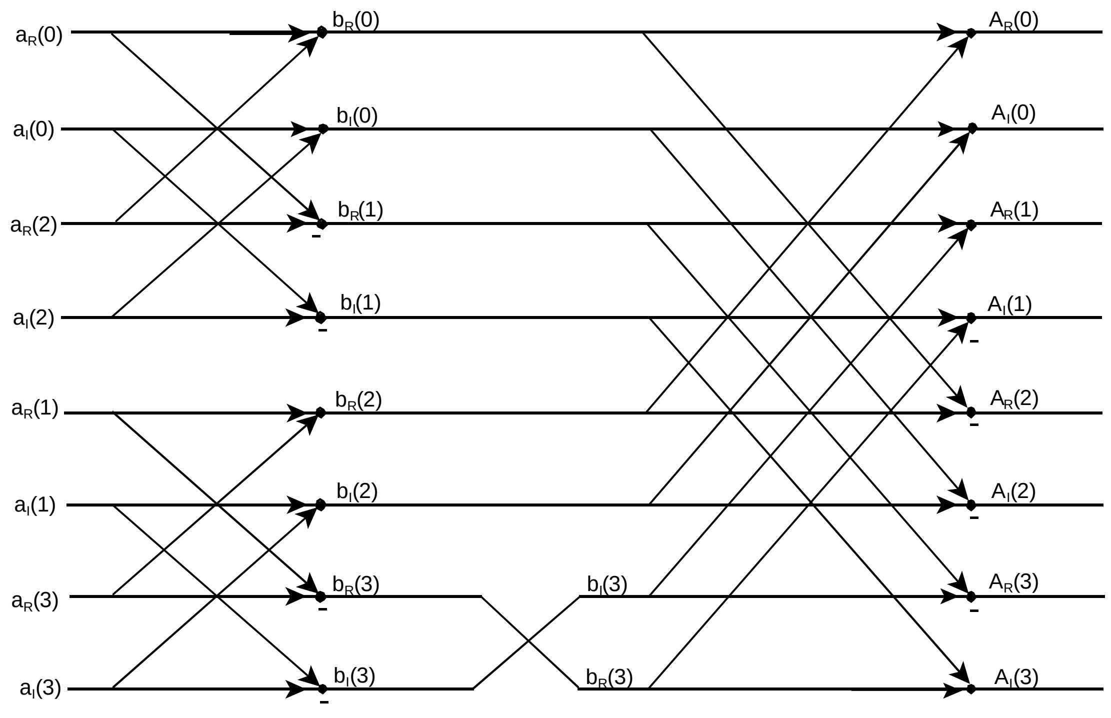


Figure 6.3 Radix-4 FFT Signal Flow Graph

The finite precision hardware mapping for the radix-4 algorithm is shown in Figure 6.4.

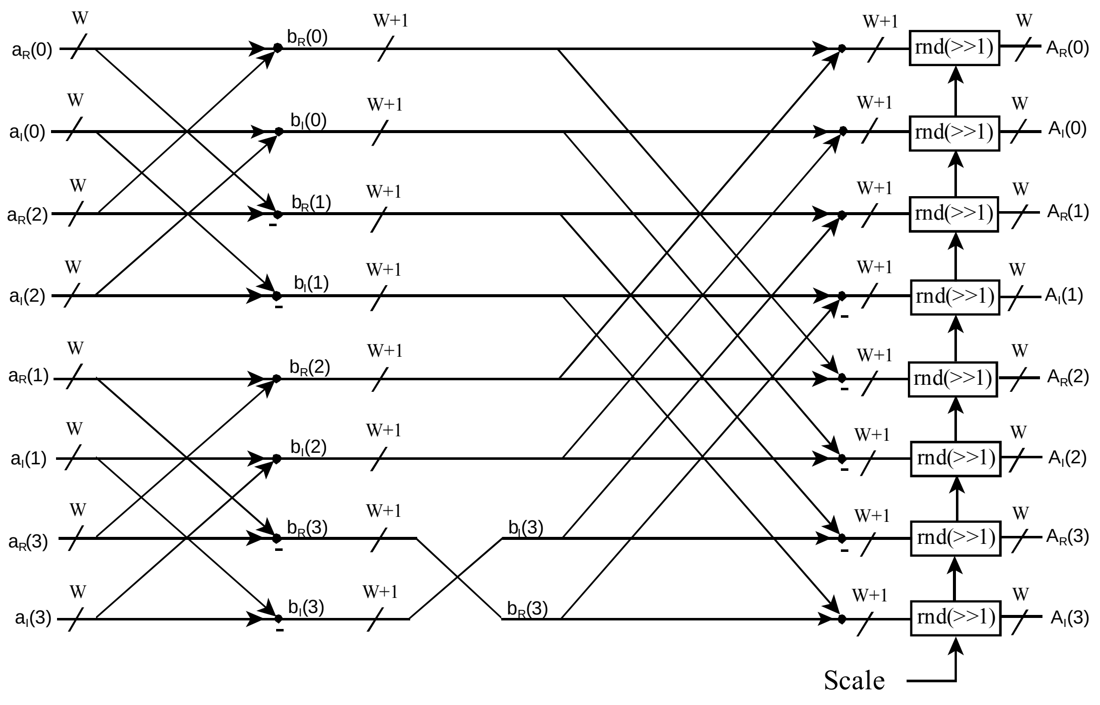


Figure 6.4 Radix-4 FFT Computation in Hardware

As for the radix-2 FFT, scaling was performed according to the final algorithm. When used for building higher order FFTs, the dragonfly structure is more efficient for hardware implementation than the butterfly structure since it eliminates an intermediate multiplication stage that would otherwise be needed to combine two butterfly operations. This comes at the expense of a slightly more complex data switching scheme that is needed for interchanging $b_I(3)$ and $b_R(3)$ before passing to the next stage of computation.

6.5 Winograd Radix-5 FFT Building Block

Several algorithms for implementing the 5-point FFT were considered. These included the Singleton, Rader and Winograd algorithms. The Winograd algorithm was chosen because it had the least number of constant coefficient multiplies (ten) and multiplier constants (five) as compared to the other algorithms [Smith95]. Unlike the radix-2 and radix-4 structures, the Winograd radix-5 FFT shown in Figure 6.5 has a complicated structure that does not easily lend itself to segmenting. That is, it cannot be broken down into a smaller structure (computational element) that can be used to compute the larger algorithm using multiple passes.

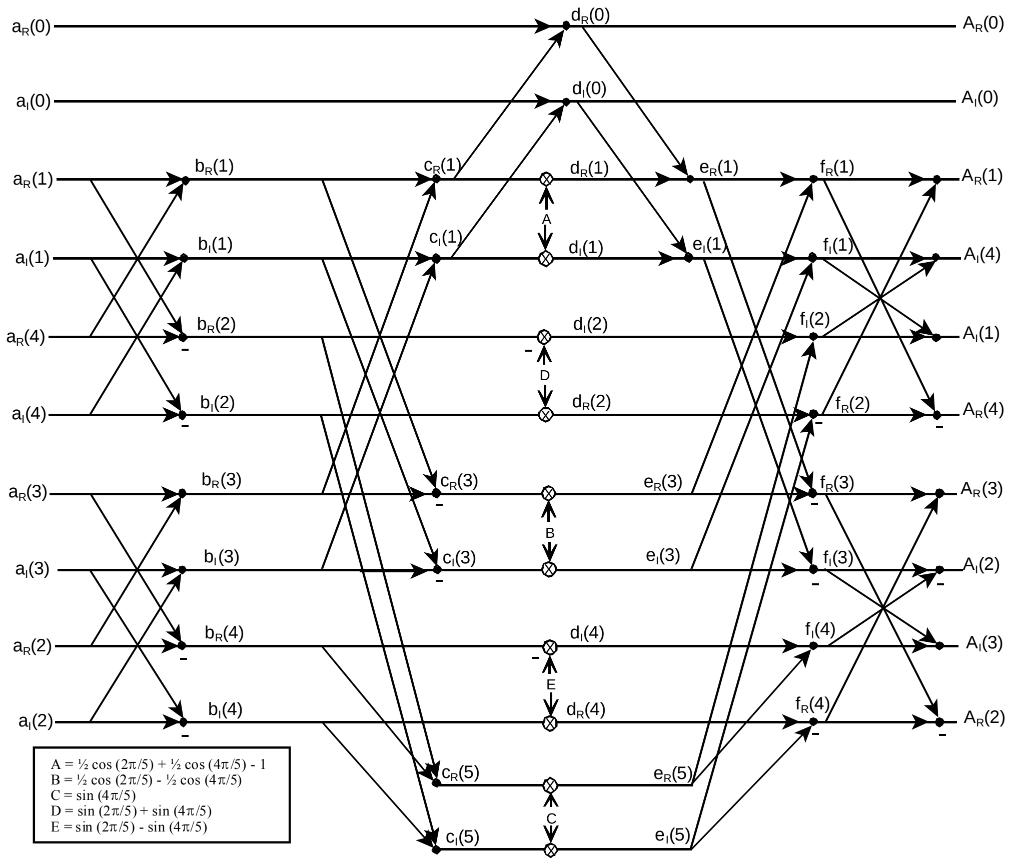
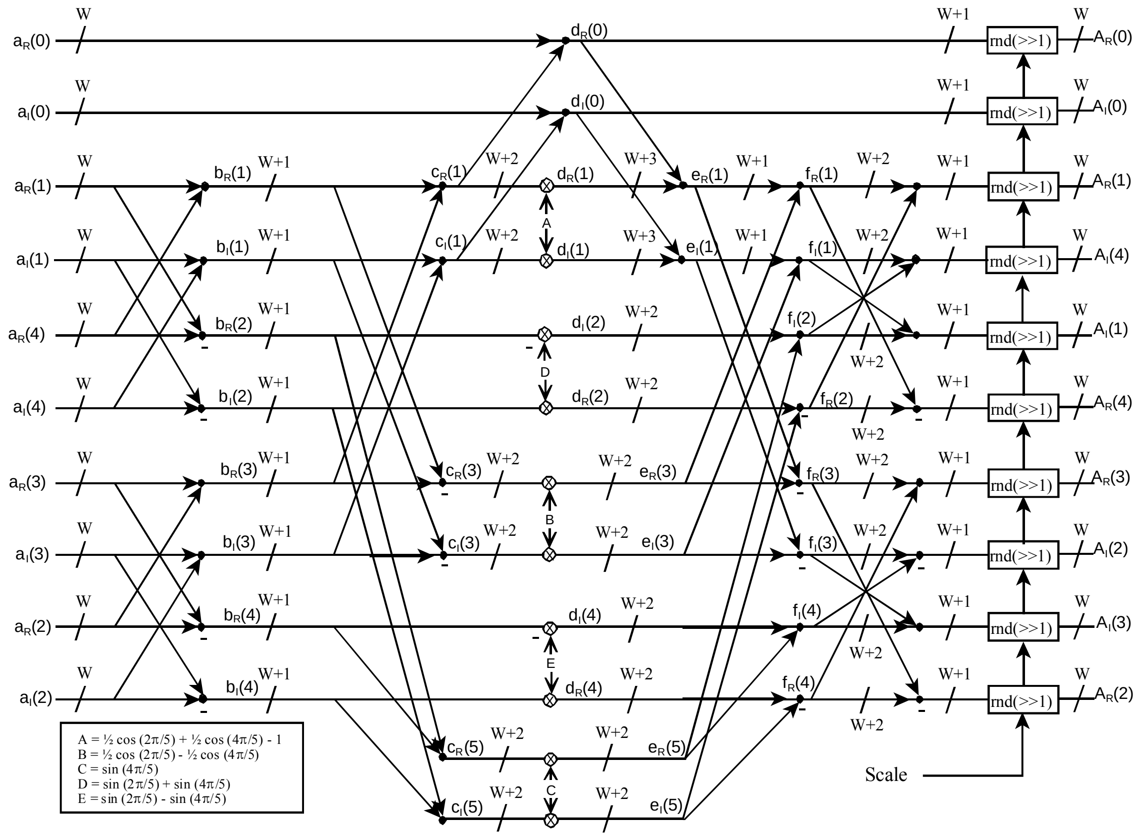


Figure 6.5 Winograd Radix-5 FFT Signal Flow Graph

The W-bit finite precision mapping for the Winograd radix-5 FFT is shown in Figure 6.6.



A = ½ cos (2π/5) + ½ cos (4π/5) - 1
B = ½ cos (2π/5) - ½ cos (4π/5)
C = sin (4π/5)
D = sin (2π/5) + sin (4π/5)
E = sin (2π/5) - sin (4π/5)
Scale
rnd(>>1)

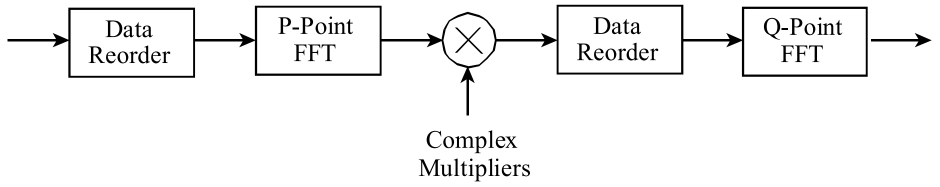
Figure 6.6 Winograd Radix-5 FFT Computation in Hardware

6.6 Mixed-Radix Approach to FFT Algorithm Construction

This section describes how multiple-radix FFT building blocks can be combined to form larger FFTs. Three approaches exist for combining smaller FFT building blocks: convolution, prime factor, and mixed-radix [Smith95]. Of these, the mixed-radix approach is the most straightforward and universal (it can be used for all transform lengths), and hence was the technique used for this work.

The mixed-radix approach to FFT algorithm construction exploits the complex conjugate symmetry properties of the DFT. The algorithms are characterized by a sequence of small-point building blocks with complex multipliers between them. The sequence is developed by factoring the transform of length, *N*, into two numbers, $N = P * Q$, and computing the N-point transform from the P-point and Q-point transforms. This is illustrated in Figure 6.7a.

(a)



(b)

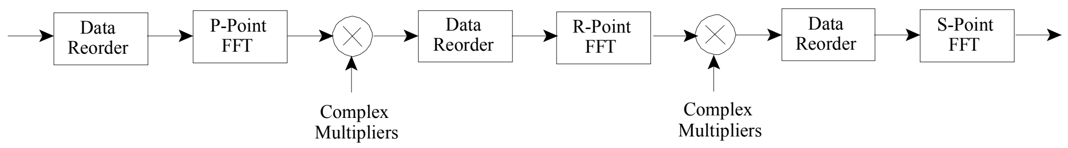


Figure 6.7 Two-Factor and Three-Factor Mixed-Radix FFT Construction

Similarly, if Q can be further factorized such that $Q = R * S$, then the Q-point transform can be constructed from R-point and S-point building blocks, as shown in Figure 6.7b. This process can be extended to an N-point transform made up of *N*'s prime factors. The order of the individual transforms determines the multiplier constants used between the stages, but does not change the number of multiplies and additions needed to perform the operation.

The complex multiplies needed between two building blocks have a predictable pattern. This is illustrated in Figure 6.8 for the n^{th} P-point building block.

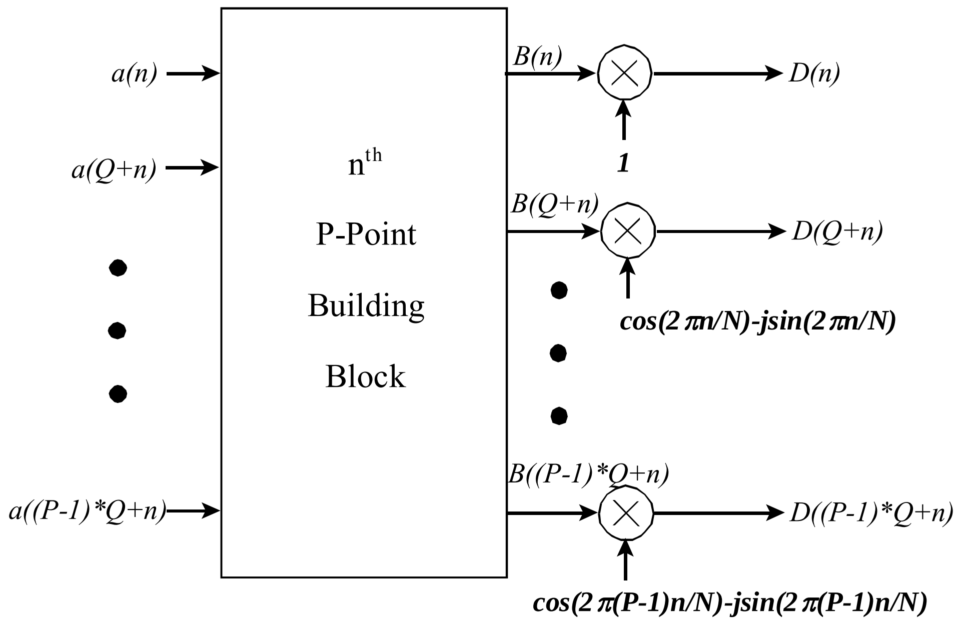


Figure 6.8 n^{th} P-point Building-Block Output's Complex Multipliers

If the multipliers are viewed as connected to the output of the P-point building block of Figure 6.7a then the following four rules apply:

1. The complex multiplier of the k^{th} output, *B(k * Q + n),* of the n^{th} P-point building block is given by $e^{-j2\pi kn/N}$.

2. The *0th* P-Point block has all 1's as the output multipliers since $n=0$ for the coefficient expression given in Rule 1.
3. The *0th* outputs, $D(n)$, of the remaining $(Q-1)$ P-point building blocks also have 1 as the multiplier since $k=0$ for the coefficient expression given in Rule 1. This gives $(P-1)$ complex multiplies per building block, for a total of $(Q-1)*(P-1)$ multiplications needed to connect the Q P-point blocks to the P Q-point blocks.
4. After multiplication, the k^{th} output, $D(k * Q + n)$, of the n^{th} P-point building block is connected to the n^{th} input of the k^{th} Q-point building block as shown in Figure 6.9.

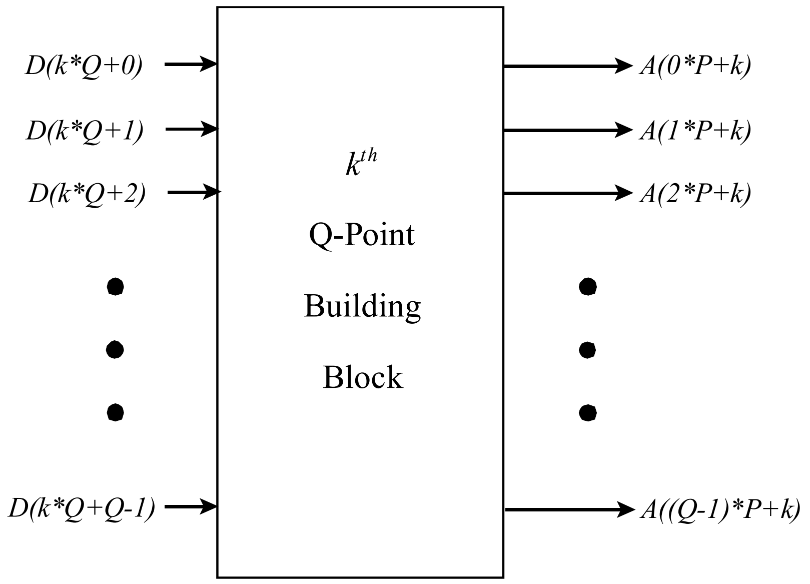


Figure 6.9 k^{th} Q-point building block input's connections

Figure 6.10 illustrates the mixed-radix combination technique for a 20-point FFT made from five 4-point, and four 5-point building blocks. The constant multipliers are shown next to the 4-point transform outputs ($W^{nk} = e^{-j2\pi nk/N}$).

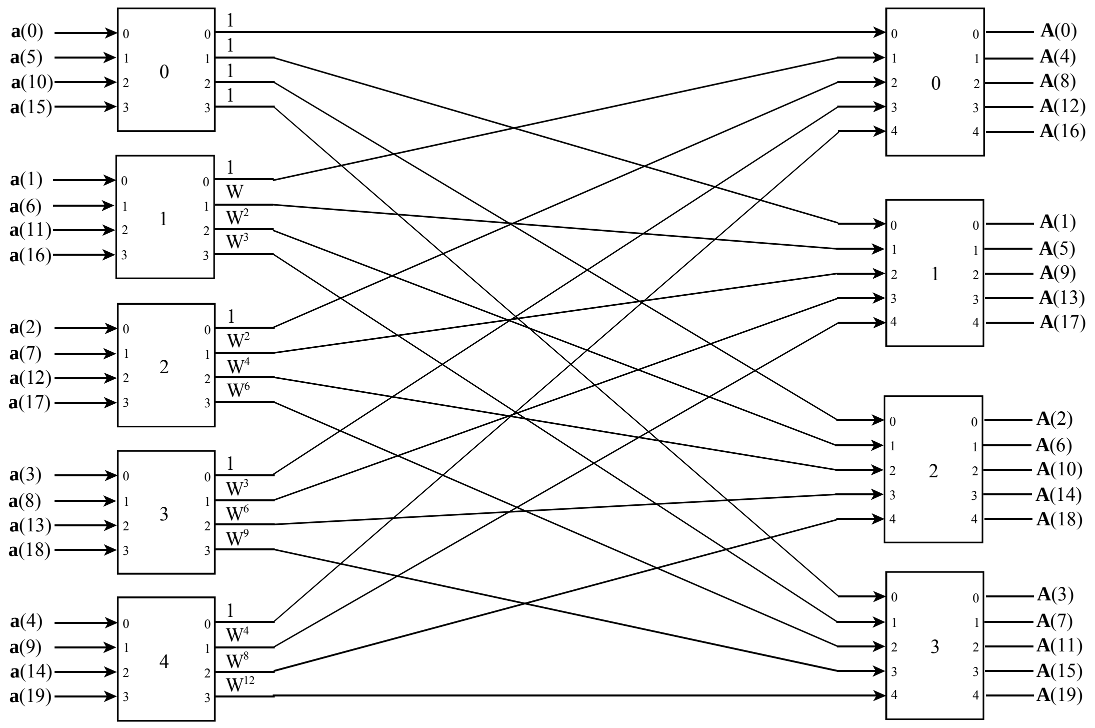


Figure 6.10 20-Point Mixed-Radix FFT Constructed from 4-Point and 5-Point Building Blocks

6.7 The 5000-Point FFT/IFFT Algorithm

The mixed-radix approach was used to construct the 5000-point FFT using the radix-2, radix-4, and radix-5 building blocks described in Sections 6.2, 6.3 and 6.4 respectively. This was based on a *[5000 = 2 * 4 * 5 * 5 * 5 * 5]* factoring scheme. It is evident why the radix-4 was used here since it saves an extra complex multiplication step and related data ordering when compared to a *[5000 = 2 * 2 * 2 * 5 * 5 * 5 * 5]* scheme. The block diagram of the 5000-point mixed-radix algorithm is shown in Figure 6.11.

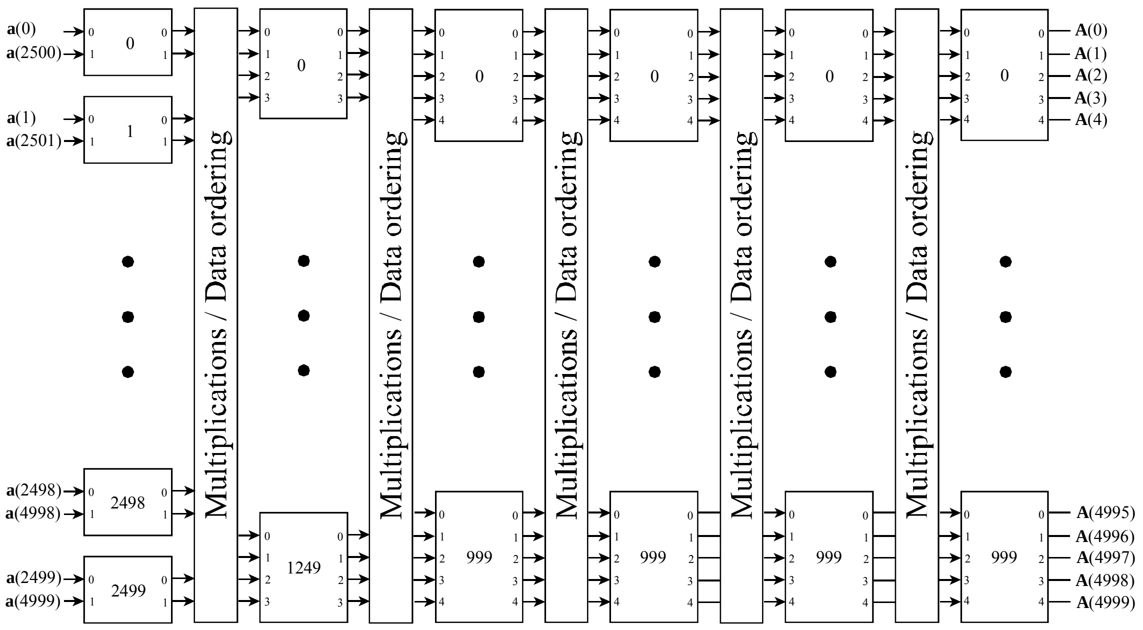


Figure 6.11   5000-Point FFT Block Diagram

The algorithm, as shown in Figure 6.11 is unsuitable for hardware implementation since it contains too many building-block elements *(2500 + 1250 + 1000 + 1000 + 1000 + 1000 = 7750)*. Such an implementation would take a restrictive amount of area, and would not exploit the processing speed of the FPGA, which is able to run at core clock frequencies in the order of several hundred MHz (for the Xilinx™ Virtex™ device). Therefore, the structure is projected vertically [Pirsch98], as shown in Figure 6.12.

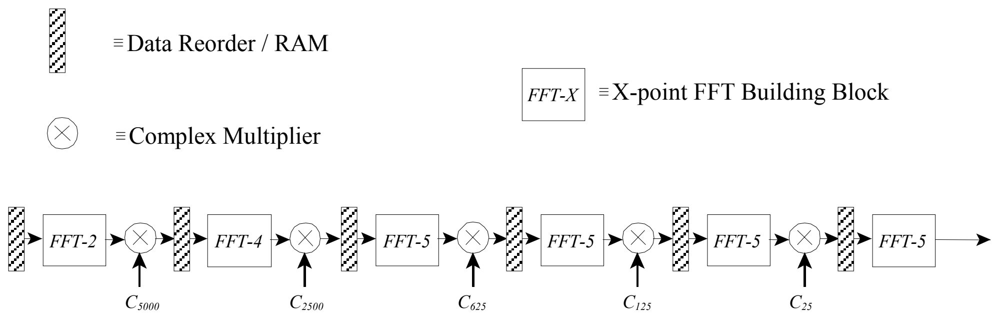


Figure 6.12 5000-Point FFT Algorithm after Vertical Projection

In Figure 6.12, the complex coefficients given by c_{25}, c_{125}, etc. represent the coefficients needed for constructing that FFT stage (for example, the c_{25} coefficients combine to make a 25-point FFT). The vertical projection reduces the number of processing elements to six and reduces the throughput approximately by a factor of 2500 since the radix-2 FFT must cycle the input data 2500 times. However, even this structure can be further optimized. The details of this optimization process are described in Section 7.3.

7 DESIGN OF GPS BLOCK PROCESSING DATAPATH IN FPGA

Chapter 4 described the novel GPS block processing receiver algorithms. So far, it has been impossible to achieve real time implementation of these algorithms owing to the huge computational requirement. This work studied the feasibility of implementing GPS block processing algorithms in FPGA hardware. Transforming an algorithm into a hardware implementation is a very challenging task. In comparison, it is relatively easy to develop an algorithm in software, and modern tools such as Matlab™ have taken the task of writing low-level code away from the user so that he or she can concentrate on developing the algorithm. In recent times, these tools have incorporated efficient compilers that make them comparable to hand-optimized software implementations [Matlab99]. When moving to the custom designed hardware domain however, it is not possible to directly convert the software algorithm into hardware. This is because custom hardware resources are limited and increased complexity results in slower processing speeds and higher cost.

This chapter describes the hardware design of the FPGA processor module of the real-time block processing GPS receiver described in Section 4.5. The overall system will be described first, followed by the design of the major individual subsystems. The design phase described here corresponds to the first steps in converting the design specification into an RTL description, as shown in the FPGA design flow of Figure 5.5. It must be explicitly noted that the system was not implemented in FPGA hardware due to the immense complexity of the task and the time restraints for this work. However, software simulations with real GPS data that models the exact functionality of the hardware architecture described in Chapter 6 and this chapter were used to verify the validity of the architecture. These simulation results are described in Chapter 8.

A system level block diagram of the FPGA processor from Section 4.5's Figure 4.15 is shown in Figure 7.1.

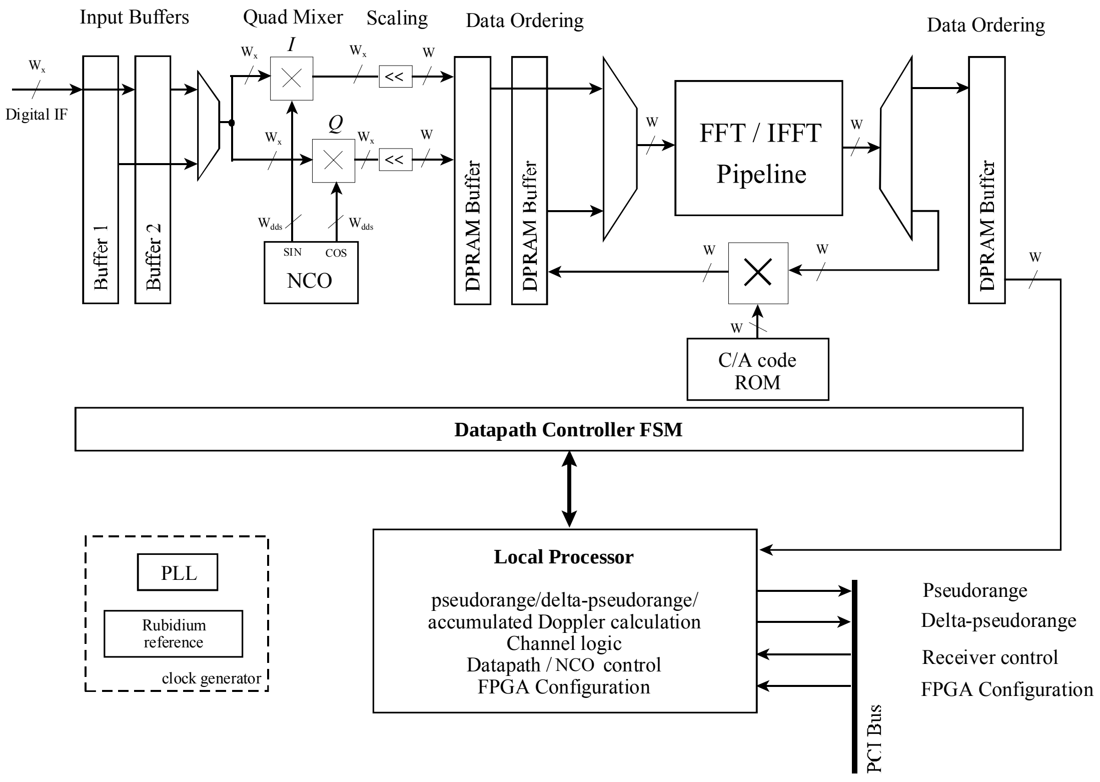


Figure 7.1 Block Diagram of FPGA Fast Correlator Processor

The FPGA processor consists of two major components: the FPGA hardware datapath, and an embedded microprocessor (hereafter referred to as the *local processor*). The datapath handles the high throughput math intensive processing while the local processor handles the low throughput software calculations. This type of partitioning is called hardware-software co-design. The co-design procedure must effectively partition the hardware verses software processing tasks for optimum efficiency in terms of speed and cost (area). For this application, the partitioning scheme is obvious, as shown in Figure 7.1.

The local processor manages the configuration of the FPGA during startup, and controls the hardware via the datapath controller finite state machine (FSM). In addition, it performs the double-precision floating-point calculations required for processing the fast correlated I and Q outputs of the hardware datapath. Hence, the microprocessor needs to be an adequately fast low-power device that supports floating-point arithmetic. A mobile Intel™ 486 or Motorola™ PowerPC™ processor is a suitable candidate. Alternatively, if enough resources exist, the microprocessor could be integrated into the FPGA. This is unlikely since floating-point hardware is too costly for FPGA implementation. The ideal for this application would be a programmable system-on-a-chip (PSOC) which incorporates programmable logic (FPGA) and an embedded microprocessor in a single device. PSOCs are currently under development by FPGA vendors and represent the next revolution in programmable logic technology.

The hardware datapath performs the quadrature mixing and fast correlation operations on the stored digital IF block. The carrier NCO, quadrature mixer, FFT/IFFT pipeline, complex multipliers and datapath FSM are all implemented in FPGA hardware.

Consistent with Figure 4.1, the incoming digital IF is stored in two DPRAM buffers such that when one buffer is busy storing the next block, the current block stored in the other buffer is being processed. The pipeline can perform many correlations by cycling through the current block of data. The number of processing cycles performed during the block time will depend on how fast the pipelined datapath can process a single correlation. On each processing pass, the correlation will be performed with different parameters. For example, the carrier NCO can produce a different carrier frequency, and/or a different C/A code transform may be used. The local processor managers these parameters according to tracking and acquisition algorithms in software, and from higher level commands received by the host processor. For example, the local processor may consecutively track several SVs and perform acquisition to find other SVs. Hence, the separate functionality of the receiver channels shown in Figure 3.1 is absorbed into a single FPGA processor. If the FPGA datapath cannot perform enough block processing cycles to meet the requirement of the GPS receiver, many such FPGA processors may need to be implemented in parallel.

The NCO generates the given carrier frequency and this is multiplied with the current block as it is been addressed out of the buffer. The result is stored in another DPRAM buffer to perform data ordering for the FFT/IFFT pipeline.

The FFT and IFFT are both processed by the same hardware pipeline. On the first pass, the FFT is computed, and this result is complex multiplied with the C/A code transform. The C/A code ROM contains the fast Fourier transformed, conjugated, upsampled, ordered, and appropriately scaled C/A codes for all 24 SVs. The appropriate C/A code can be selected by changing the page address of the ROM. On the second pass, the IFFT of the multiplied result is calculated and stored in a DPRAM buffer/sorter, which provides the result of the correlation to the local processor.

Since this is a large design, the DPRAM buffers may have to be external to the FPGA. The design will have many clocks operating at different rates. A PLL can be used to generate all the necessary clocks for the system using the master Rubidium reference clock. The new Xilinx™ Virtex™ devices have built-in DLLs for clocking the FPGA, and it may be possible to use these instead of an external PLL.

The following sections describe the design of the carrier NCO, complex multipliers, and the FFT/IFFT pipeline, respectively. Since the following is a high-level description, datapath controller details are not considered.

7.1 Numerically Controlled Oscillator

As shown in Figure 7.1, the buffered digital IF signal is quadrature multiplied by a locally generated carrier frequency to perform the carrier wipeoff operation [Section 3.2.1]. A numerically controlled oscillator (NCO) is used to generate the in-phase and quadrature carrier signals with a 1.25 MHz center frequency. The carrier NCO must be capable of spanning the ±10 kHz Doppler offset as described in Section 3.3, and have a frequency resolution much smaller than the residual phase error being outputted by the carrier phase discriminator. In addition, the carrier NCO must have low phase noise (i.e. carrier phase jitter) since otherwise it would distort the carrier phase and frequency measurements. Figure 7.2 shows a block diagram of the NCO. NCO-based frequency synthesizers, known as direct digital synthesizers (DDS) are quickly replacing their PLL counterparts because they are cheap and simple to implement in digital hardware.

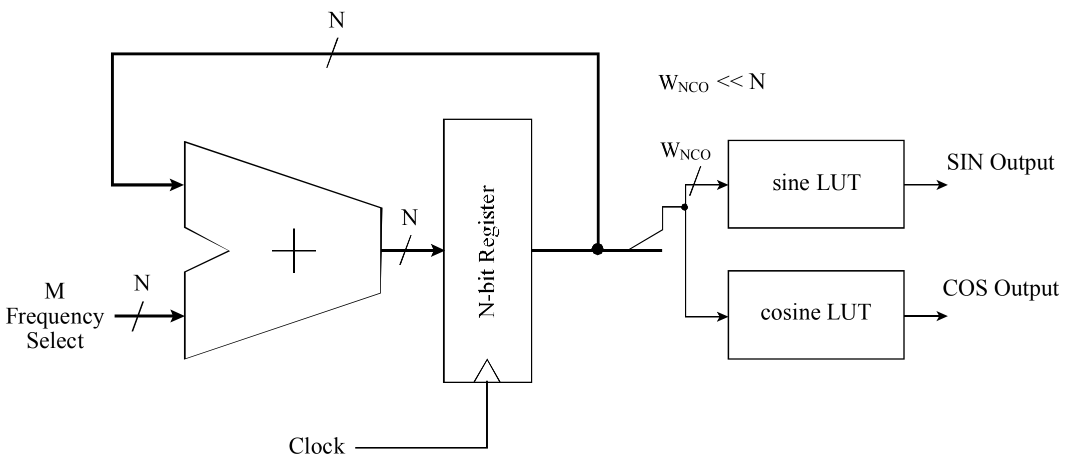


Figure 7.2 Block Diagram of Carrier NCO

The NCO is made up of two components: the phase accumulator and the output function look-up tables (LUTs). The phase accumulator (adder + register) of width N is continuously clocked by a stable reference oscillator with frequency f_{clk}. The digital frequency selection value, M, is fed to the accumulator. At each clock cycle, M is added to the previous sum of the accumulator resulting in a periodic numerical stair-step function at the output of the NCO. When the maximum value of the accumulator is reached, it overflows, completing one NCO cycle. The output frequency, f_{out}, of the NCO is limited by the Nyquist criterion ($\frac{1}{2} f_{clk}$), and is given by:

$$f_{out} = \frac{M \cdot f_{clk}}{2^N} \tag{7.1}$$

The stair-step waveform can be interpreted as phase information if the range of *N* is mapped from *0* to *2π* radians. Hence, a phase-to-amplitude converter can be used to generate any desired periodic waveform whose frequency is variable depending on the value *M*. Frequency modulation can be performed if *M* is a function of time.

For a desired output frequency, $f_{desired}$, the value of M is calculated as:

$$M = round\left[\frac{f_{desired} \times 2^N}{f_{clk}}\right] \tag{7.2}$$

The maximum frequency attainable from the NCO, that is, by setting *M* to its maximum value is dependent on the desired phase resolution. Hence, M_{max} is given by:

$$M_{\max} = 2^{(N - W_{NCO})} \tag{7.3}$$

Where W_{NCO} is such that, for *K* phase points, $K = 2^{W_{NCO}}$. Hence the maximum obtainable frequency is:

$$f_{out,\max} = \frac{f_{clk} \cdot 2^{(N - W_{NCO})}}{2^N} \tag{7.4}$$

The frequency resolution, Δf_{out}, of the NCO is given by:

$$f_{out} = \frac{f_{clk}}{2^N} \tag{7.5}$$

From Equation 6.5, it can be seen that the phase noise of the NCO is directly dependant on that of the reference clock. Since $f_{out} < f_{clk}$, and the desired center frequency of the carrier NCO is 1.25 MHz, the highly stable 5 MHz reference frequency used for the ADC can be used for the carrier NCO. This clock is derived from a highly stable OCXO for short-term stability and is referenced to a Rubidium source for long-term stability.

Since the block processing algorithms for carrier phase measurements require a very fine frequency resolution [Feng99], a 32-bit *($N=32$)* NCO was chosen for the design. This gives a frequency resolution of 1 mHz. To generate a mean frequency of 1.25 MHz, $M = 2^{30}$.

The amplitude resolution of the carrier NCO is determined by the amount of most significant bits, W_{NCO}, taken from the N-bit register for the phase to amplitude converter. For example, if only the first two most significant bits are used ($W_{NCO}=2$), only four discrete values are available for representing the amplitude of the sine function. Hence the sine function with a 2-bit LUT will have values: 0, 1, 0, –1, which is a triangular wave approximation to a sinusoid.

The phase resolution, $\Delta\phi_{out}$, of the carrier NCO is also given by W_{NCO} where:

$$\Delta\phi_{out} = \frac{2\pi}{2^{W_{NCO}}} \tag{7.6}$$

Hence, for the $W_{NCO}=2$ case described above, the phase resolution is π/2 radians. Phase resolution of the NCO should not be confused with its frequency resolution. Phase and amplitude resolution refer to the quantization that results from taking less than N bits to represent the entire 0 to 2π phase range generated by the NCO, whereas the frequency resolution determines how accurately the NCO can generate a given frequency.

As shown in Figure 7.2, two LUTs are used to map the stair-step functions to sinusoids. For a hardware implementation, this can be performed with only one ROM that is time-shared. Furthermore, only one quadrant of the cycle needs to be stored in the ROM since the remaining three quadrants can be generated from reverse addressing and inverting the stored values. However, these optimizing techniques would not save much hardware resources for the given application since W_{NCO} is not significantly large. Section 8.2 shows the amount of acquisition margin lost and code ranging noise introduced due to the carrier NCO amplitude quantization. NCO design is a complex subject and only a description within the scope of this thesis was presented here.

7.2 Complex Multiplier

The complex multiplier is a vital component used in the GPS fast correlator hardware implementation. It is used in the quadrature mixer and in the intermediate processing stages of the FFT/IFFT datapath for the twiddle factor multiplications.

The multiplication of two complex terms ***x*** and ***a*** is given by:

$$\boldsymbol{x} \cdot \boldsymbol{a} = \boldsymbol{u} =$$

$$(x_r + jx_i)(a_r + ja_i) = (x_r a_r - x_i a_i) + j(x_i a_r + x_r a_i) \tag{7.7}$$

In this form, the multiplication requires four real-valued multiplications and two real-valued additions. A modification can be used to reduce the overhead to three real-valued multiplications the five real valued additions, as shown in Figure 7.3 [Pirsch98].

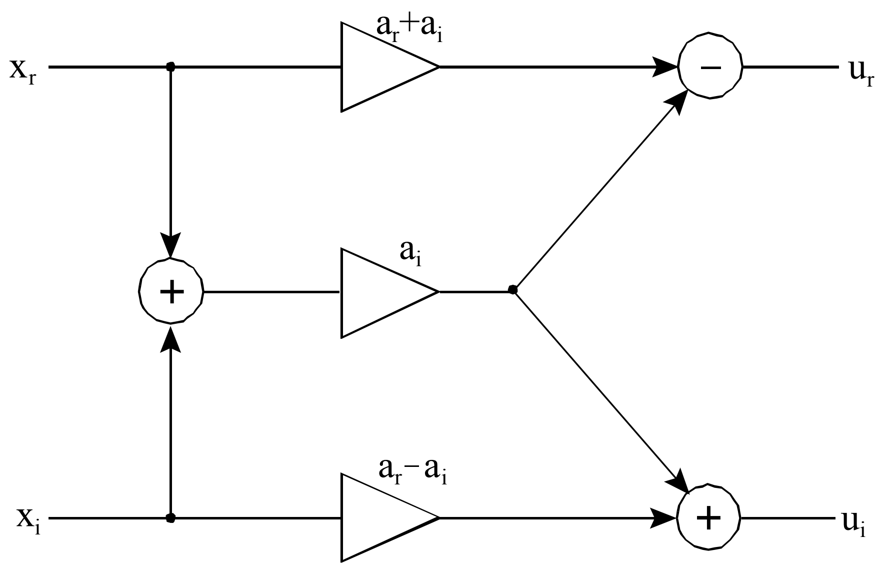


Figure 7.3 Complex Multiplier Structure Using Three Real Multipliers

This technique is attractive for FPGAs since addition can be performed efficiently and with fewer resources compared to multiplication. As described in Section 5.2, an adder requires only one CLB for two bits and uses the fast carry logic of the FPGA. Furthermore, in the case of the complex multiplications needed for the mixed-radix FFT structure, since the coefficients are stored in memory, the a_r+a_i and a_r-a_i values can be pre-calculated, thus avoiding more additions.

Figure 7.4 shows the hardware design of the complex multiplier.

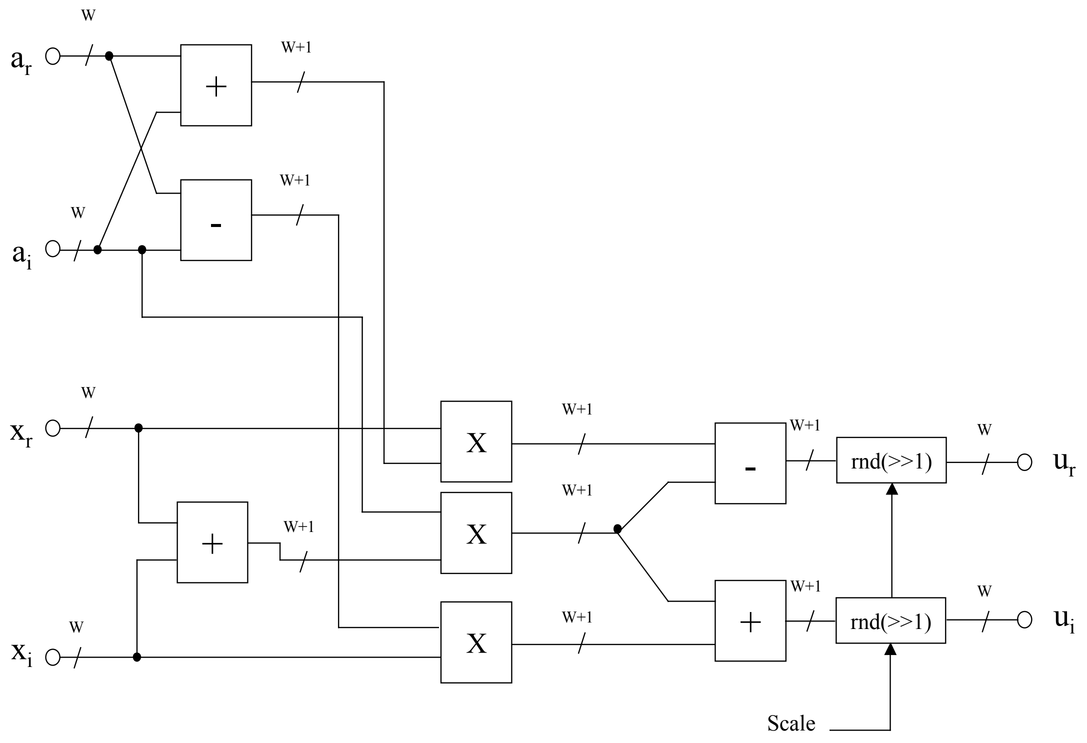


Figure 7.4 Block Diagram of Hardware Complex Multiplier

For W-bit ***a*** and ***x*** inputs, the first stage adders need *W+1* bits to represent the output. After multiplication with the a_r+a_i, $a_r–a_i$, and a_i values, the result does not exceed *W+1* bits for the worst case. Furthermore, the post-multiply additions also result in a *W+1* bit range. Since the datapath is *W* bits wide, the *W+1* bit outputs of the final adders are scaled by shifting 1-bit to the right and rounded to maintain W-bit precision throughout the hardware datapath. The scaling is done only at certain points in the datapath to avoid overflows. Hence, this is shown as a switchable operation in Figure 7.4.

The rounding operation is important since truncation leads to large error buildups in the FFT processing. Rounding can be achieved by examining the first binary point and the sign bit of the W-bit word to determine if the result needs to be incremented by 1. If so, the carry input of the next processing stage is set, effectively adding 1 to the result.

If at least three cycles are available for performing the multiplication operation, the structure shown in Figure 7.4 can be folded to a structure of two adders and a multiplier.

The multipliers required for the operation must be $(W+1)\times W$ bits wide. However, only the upper $W+1$ bits of the resulting 2W-bit output is used. Hence, a significant amount of FPGA hardware can be saved if an optimal multiplier can be custom designed for $W+1$ output bits. An attempt to design a Booth recorded multiplier [Wolf94] was made, however only slight area improvements were realized compared to Xilinx™'s parameterizable $N\times N$ multiplier core which contains the logic for all $2N-1$ outputs. This is because the Xilinx™ core is highly optimized and floorplanned for the FPGA architecture. Nevertheless, it should be possible to design a smaller multiplier. An optimally designed multiplier would save a considerable amount of FPGA resources since the design needs a large number of multipliers, and hence warrants further study.

7.3 FFT/IFFT Datapath

Section 6.6 described the vertical projection of the 5000-point FFT to give an architecture suitable for hardware implementation. Simulations were performed for this structure in Matlab™, and it was found that the structure needed to be scaled by $1/2^6 = 1/64$ for both the FFT and IFFT, in order to prevent overflows during processing. Several attempts were made to optimally distribute the $1/64$ scaling factor in the hardware datapath. The scaling scheme shown in Figure 7.5 yielded the best results, and is the scheme used for simulations in this thesis. The scaling is applied to the individual FFT building blocks shown in the hardware structure figures in Sections 6.2, 6.3, and 6.4.

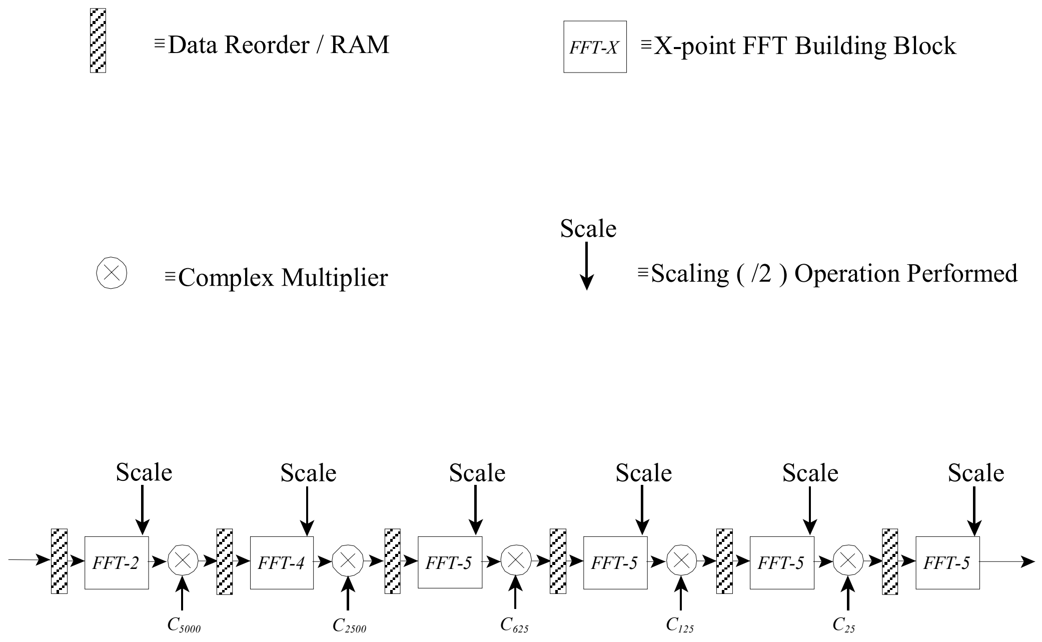


Figure 7.5 5000-Point FFT/IFFT Structure Showing Scaling Distribution

Each building block of the structure of Figure 7.5 takes a different number of clock cycles to complete its operations. Hence, the faster building blocks would sit idle while the slow ones complete their processing. Hence, the structure of Figure 7.5 needs to be further optimized to equalize the number clock cycles needed for each stage.

From Figure 6.6, it can be seen that the complicated structure of the 5-point FFT is such that it cannot be broken down into smaller components. Furthermore, this is the main operation that comprises the 5000-point FFT, needing 4000 radix-5 computations per FFT. Hence, a single 5-point FFT processing element (PE) can be implemented just as it is given in Figure 6.6. The extra area this requires is justified since only one PE is required. Of course, the PE is pipelined to achieve maximum throughput possible. When the 5-point FFT PE is implemented in this way, it will perform a complete FFT/IFFT operation in one clock cycle with a latency given by the number of pipelined stages (latency issues are neglected in this work since no attempt was made to implement the architecture in an FPGA). Since four complex multiplication operations follow the 5-point FFT computation, the multipliers will also need to perform their operations in one clock cycle. Hence, the multipliers can be implemented as shown in Figure 7.4 with pipelining.

Since the 5-point FFT PE will take at least 4000 cycles to perform the 4000 5-point FFTs (not counting any memory access or data reorder cycles), the remaining 2-point and 4-point FFTs can be folded to equalize the number of clocks cycles needed for performing all FFT operations to about 4000 cycles. The radix-4 FFT of Figure 6.4 can be easily folded to perform only one butterfly operation per clock cycle. When the radix-4 FFT is folded in this manner, it takes four cycles to perform a single FFT and hence would take 5000 cycles to perform all 1250 4-point FFTs.

For the radix-2 algorithm of Figure 6.2, the operation is broken down into two sweeps through two adders, thus taking 2 cycles per FFT. Hence, the 2-point FFT PE takes 5000 cycles to perform its 2500 FFTs. Since all the folded PEs takes approximately the same number of cycles to perform a complete 5000-point FFT (except for the 5-point PE which takes 4000 cycles, however an extra cycle per FFT could be needed for the complex data ordering), the datapath is now cycle optimized. This optimized structure is shown in Figure 7.6.

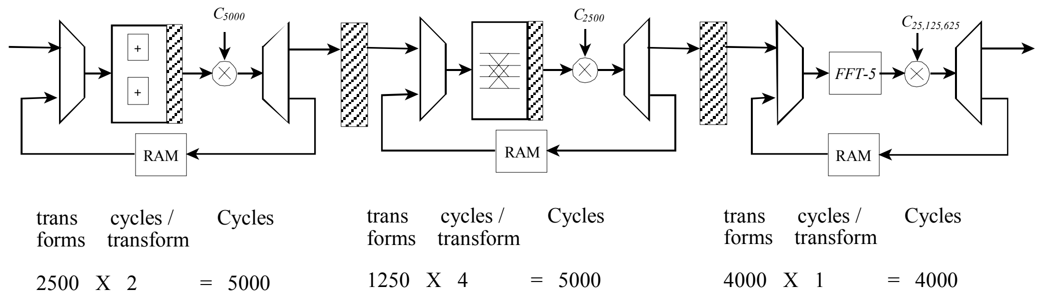


Figure 7.6 Processing-Cycle optimized 5000-Point FFT/IFFT Datapath

Assuming this structure can be implemented in FPGA hardware and the hardware is operating at a core clock frequency of 30 MHz (typical for a slow XC4000E device), the datapath would process *30,000,000/5000 = 6,000* FFTs per second, which means the pipeline will process 3 complete correlations per 1 ms block of data. If this structure is implemented in a fast Xilinx™ Virtex-E™ device, approximately an order of magnitude increase in processing throughput can be realized. Hence, given the datapath of Figure 7.6 can be implemented in an FPGA and can be made to execute in approximately the same number of clock cycles as derived above (i.e. 5000 cycles per FFT); then, given the clock speeds of current FPGAs (~300 MHz), it is entirely feasible to realize a GPS real-time block processing receiver.

In order to implement the derived FFT/IFFT architecture given above, it is necessary to study the amount of precision needed in the datapath (i.e. the number of bits needed to represent the data). This analysis is described in the following chapter.

8 BLOCK PROCESSING HARDWARE SIMULATIONS

Chapters 6 and 7 described the underlying algorithms and the hardware architecture required for implementing a GPS block-processing receiver. As mentioned earlier, the block processing algorithms described in [UijtdeHaag99] and [Feng99] were developed in software, where double-precision floating-point arithmetic was used (i.e. Matlab™). In order to migrate these techniques into hardware, it is important to study their performance degradation due to finite precision arithmetic. The purpose of this is two fold: 1) To find the optimum hardware architectural parameters, and 2) Determine the maximum error bound for a given architecture. The main concern is the computational errors of the FFT/IFFT datapath, since it is the largest component to be implemented in hardware. Even though the analytical error bound for finite-precision FFTs is well documented in literature such as [Rabiner75] and [Knight79], its direct application to the circular convolution based correlator becomes too complex. Instead, this work studies error statistics from simulations using real GPS data. By running simulations for different satellites with varying signal strengths, the hardware complexity verses performance tradeoff can be analyzed statistically. At the end of the analysis, questions such as: "how wide does the hardware pipeline have to be in order to reliably acquire the GPS signal?" or "how much performance does one sacrifice if the carrier NCO amplitude quantization is reduced to 2 bits?" can be answered based on the results of the simulations. Indeed, one goal of this work was to determine such architectural parameters once and for all, since earlier attempts to approach this problem from the bottom up (i.e. develop some kind of hardware first and determine if it could be used for

GPS block processing) proved to be a nonproductive task since it was impossible to get a handle on the problem. Furthermore, with this approach, it was difficult to estimate hardware resource requirements so that a suitable FPGA COTS product could be purchased. Nor was it possible to judge how much performance degradation could be expected for a particular architecture.

The hardware simulations were performed in Matlab™. This is done by scaling and rounding values such that the operations matched those of the hardware architecture described in Chapter 7. After each stage of processing, the results are checked carefully for their validity based on the results that would have been obtained from an actual hardware stage. The effect of overflow error was not simulated. Instead, the architecture is defined such that overflows do not occur. Matlab™ was used because of its simplicity and efficiency. VHDL simulations were attempted, but proved too slow because VHDL models too many parameters (such as each signal's transaction history), which was not needed at this level of abstraction. To illustrate the complexity and computational intensity of the hardware simulation modeled in Matlab™, a single simulation run consisting of one second of collected data evaluated for three SVs and a datapath width ranging from 8-bits to 22-bits took approximately 90 hours to complete on a PC with dual Pentium™ II 300 MHz processors. This was after all sequential loops were eliminated and the code was optimized to use only matrix operations.

Three main simulations were performed to study hardware processing accuracies. The first simulation studied the effects of ADC quantization on code tracking accuracy and acquisition margin. Since the ADC quantization sets the dynamic range of the input samples, this study was done to find the minimum range needed to represent the raw GPS data (i.e. digital IF). Furthermore, this simulation gave insight into the robustness of the GPS signal and the correctness of the processing algorithms.

The second simulation studied the effects of the carrier NCO amplitude quantization. That is, the number of bits representing the in-phase and quadrature carrier components in the carrier NCO Sine and Cosine LUTs, as described in Section 7.1. This simulation was mainly performed to determine if the digital quadrature mixer could be implemented without multipliers by using only a multiplexer and a two's complementor, and, if so, how much performance degradation would be caused by such a scheme.

The third simulation studies effects of the FFT/IFFT hardware pipeline width on system performance. Since the 5000-point FFT/IFFT pipeline involves many operations and takes the most amount of FPGA resources (Chapter 7), and each bit of extra precision results in an exponential growth in area and complexity of the digital hardware, it is important to determine the least possible pipeline width necessary that will get the job done for a given application.

The simulations are compared to the results obtained from Matlab's double precision processed results. For the latter two simulations, the ADC quantization is fixed at 8-bits (i.e. $W_x=8$). This was done so that the effect of only the quantity being evaluated (i.e. the hardware structure under test) could be studied. The setup and results for each of the simulations are described in the following sections.

For each simulation, two main quantities are studied. These are the GPS acquisition margin and its associated implementation loss, and the code phase based ranging error. These quantities were defined in Section 4.3. Whenever the term ‘range’ is used in this chapter, it refers to the quantity $c(t_{tr})$, which cannot be called range nor pseudorange according to the definitions of Chapter 3. The following section describes the effect of truncation verses rounding in the hardware datapath.

8.1 Effect of Truncation

This simulation studies the effect of truncation in the fast correlator hardware datapath. The justification for truncation is the fact that rounding is more complex than merely truncating the results. Hence, if rounding can be avoided, the design can be made simpler.

The behavioral model for each processing stage was written to include a rounding/truncation switch. When truncation is enabled, the results of each stage is truncated rather than rounded when any scaling is performed, or less precision than that of a processed result is sent to the next processing stage. Figure 8.1 shows the single-block simulation results of correlations for *SV4* from $W=8$ through $W=15$ bits when truncation is enabled.

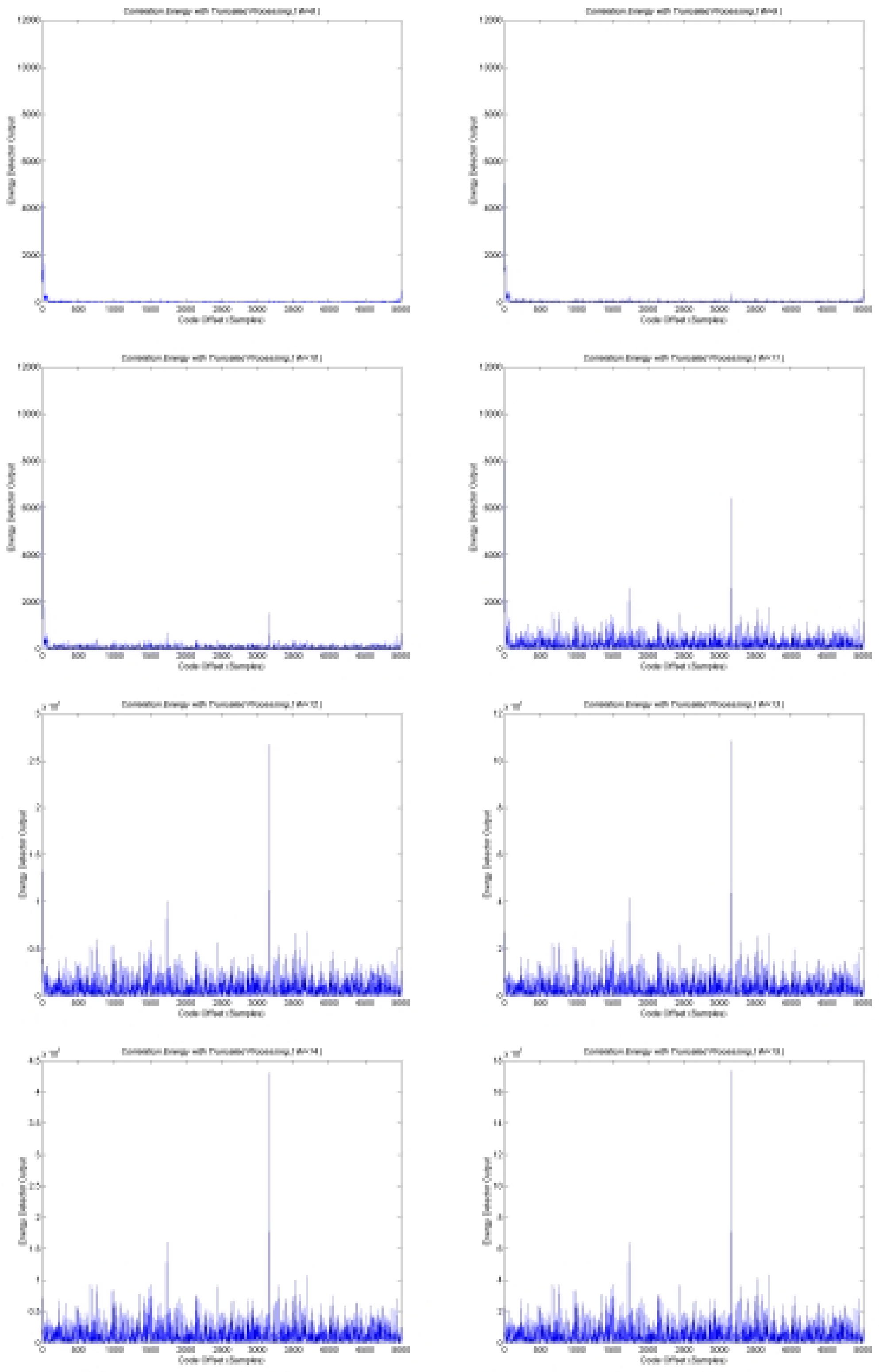


Figure 8.1   Plots Showing the Effect of Truncation on the Datapath from W=8 to W=15

As seen from the plots, when the datapath precision is low, the truncation error produces a large DC peak that seriously interferes with the correlation peak. However, as the datapath precision is increased gradually, the truncation error is offset to the point where the DC peak becomes lower than the noise peaks, essentially having no effect on the processing. However, this happens only when *W≥14*. Figure 8.2 shows the truncation error relative to the result obtained from rounding for *SV4* when *W=15*, which illustrates a significant error even at this high precision.

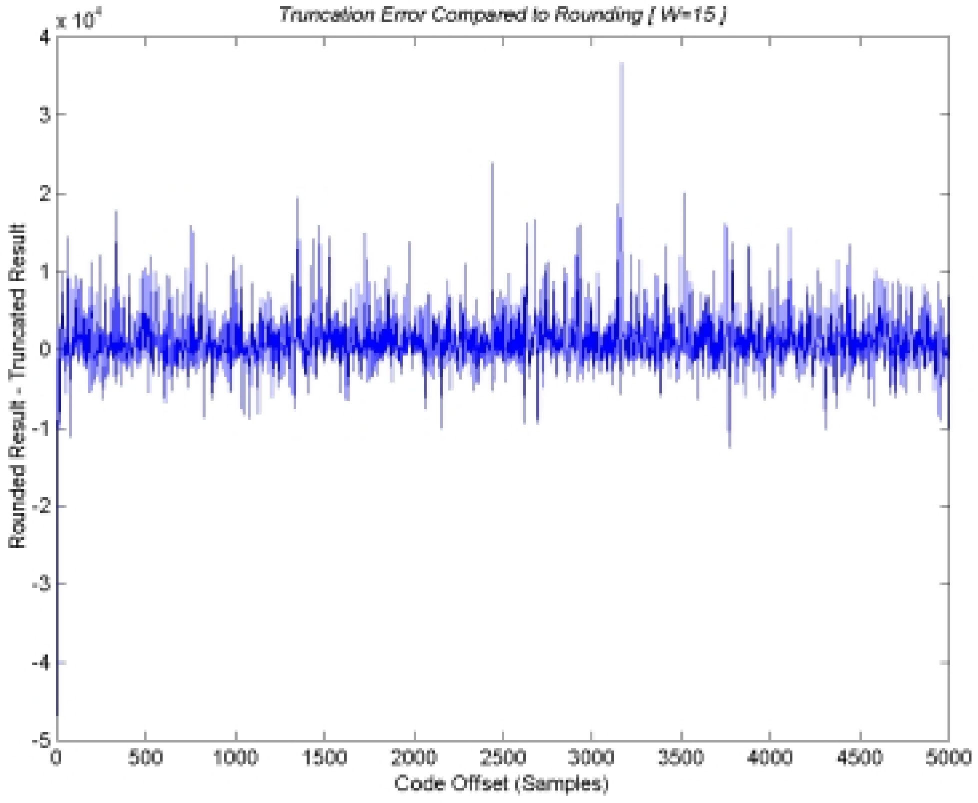


Figure 8.2 Difference Between Rounded and Truncated Processing Schemes

Since better accuracy can be achieved with rounding for a smaller datapath precision, and the cost of rounding is much cheaper than a wider datapath, the conclusion is: it is better to round than to truncate. For all subsequent simulations, the datapath simulation is set for rounding.

8.2 Effect of ADC Quantization

This simulation studies the effect of ADC sampling quantization on the resulting correlation. As mentioned before, the GPS data acquisition system uses a 12-bit ADC. To simulate lower quantization values, the least significant bits of the input data are ignored (i.e. truncated). Block processing simulations for ADC quantization values from 2-bits to 12-bits was performed. The results of these simulations are presented in this section.

For this simulation, the truth is the result obtained from processing the full precision data set ($W_x = 12$). Figure 8.3 shows the estimated range measurements for SV10 (from Equation 4.8) along with a quadratic fit of the data. Figure 8.4 shows the range data when the ADC quantization is two bits wide (i.e. $W_x = 2$). The deviation of the range estimates about the mean is clearly visible. Hence, observing the standard deviation of the range estimates shows the degradation in the measured values as a function of the hardware parameter (ADC quantization in this case). Therefore, only the standard deviation of the range measurements relative to the curve fit will be presented henceforth.

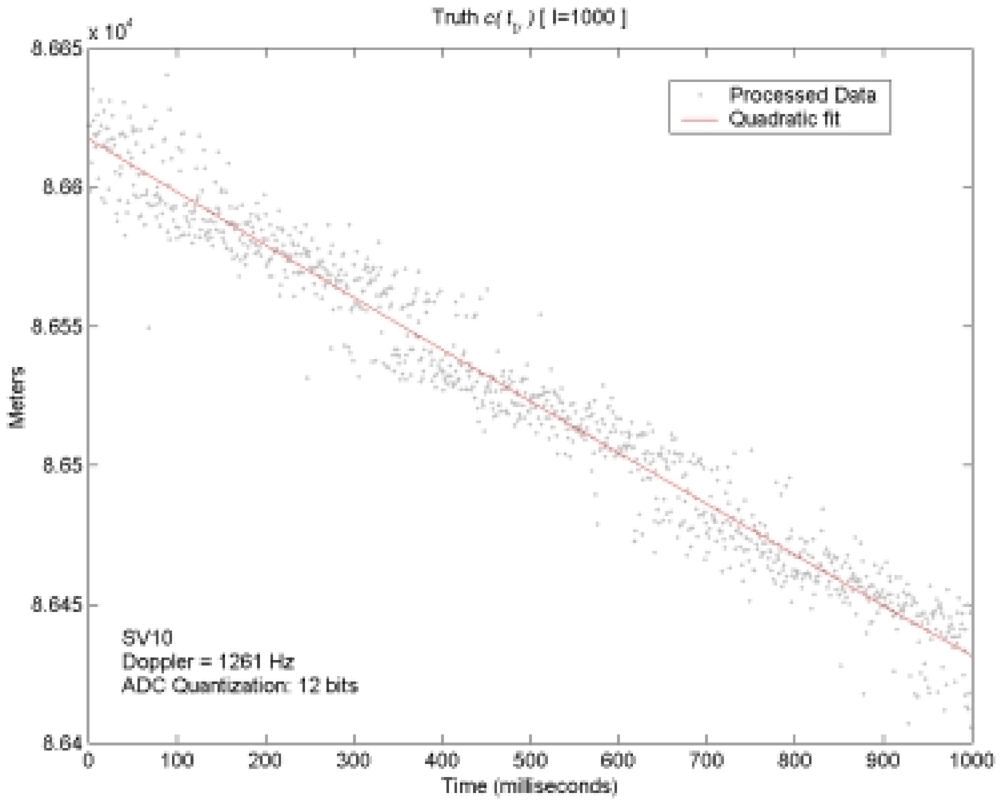


Figure 8.3 *$c(t_{tr})$* for SV10 With 12-bit ADC

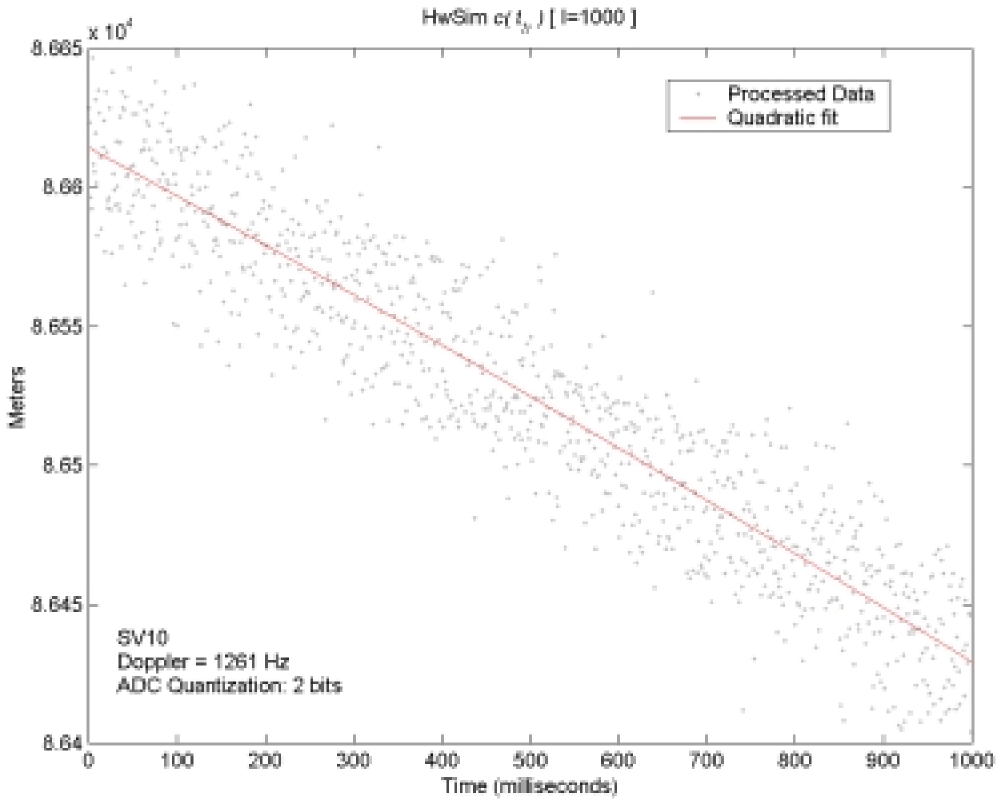


Figure 8.4 *$c(t_{tr})$* for SV10 With 2-bit ADC

Figure 8.5 shows the effect of ADC quantization on the standard deviation of the range estimates. At $W_x = 5$, the deviations begin to approach the truth. This is evident in Figure 8.6 where the inflation of the deviations with respect to the truth is shown as a percentage. The deviations settle to within 10% of the truth for $W_X = 5$.

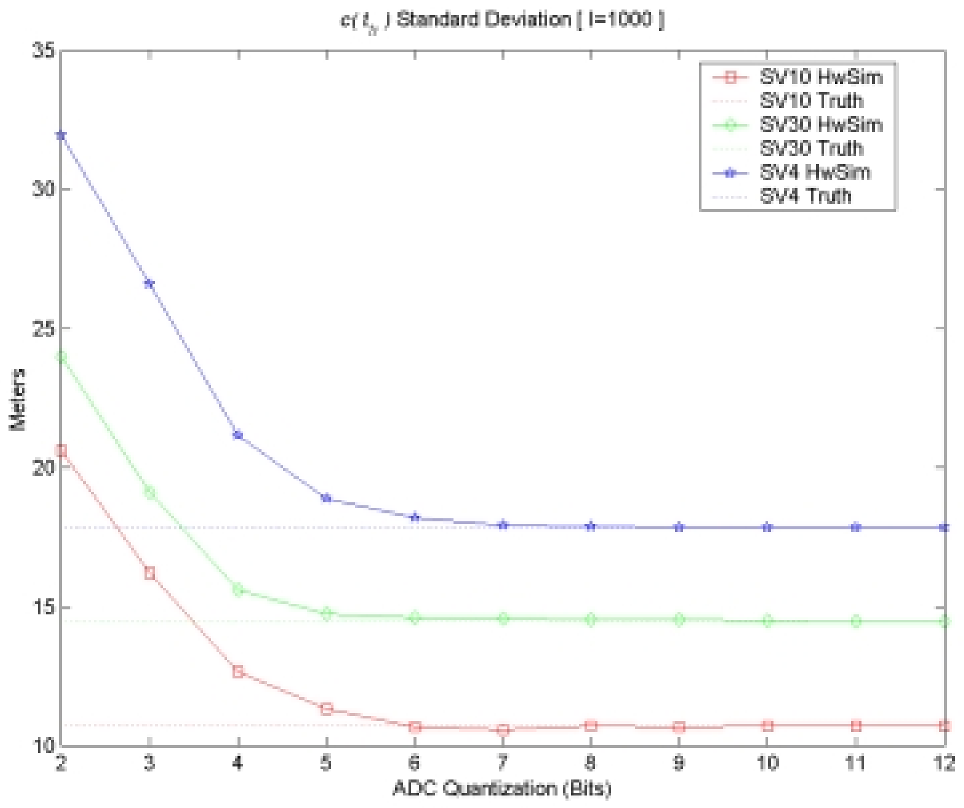


Figure 8.5 Effect of ADC Quantization on *$c(t_{tr})$* Standard Deviation

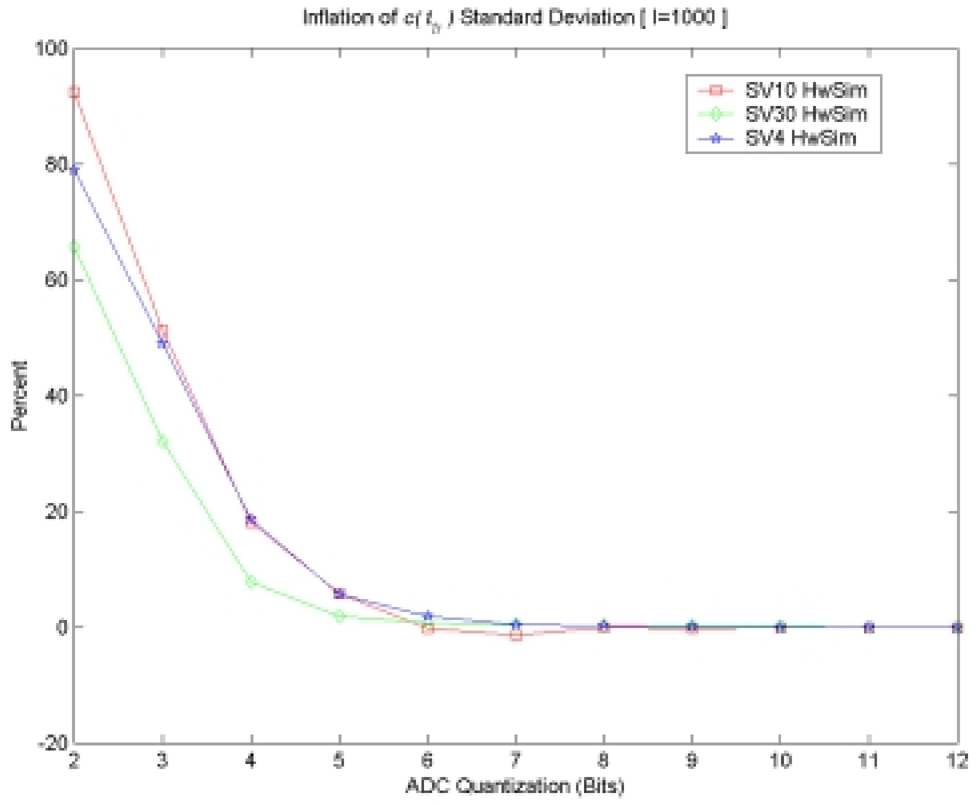


Figure 8.6 Inflation of $c(t_{tr})$ Standard Deviation With Respect to the Truth

The standard deviations are calculated with respect to the 2nd order fit to the data, in a least-squared sense. A second order fit is used because the user-SV dynamics consists of speed and acceleration components. Any higher order components are ignored because this tends to model the noise as well. The difference between the curve fits indicates any range measurement biases introduced by the ADC quantization. Figure 8.7 plots these biases. It can be seen that the bias is largely zero from $W_x=2$ to $W_x=12$ bits for *SV30* and a few meters for the other two SVs when W_x is less than 4 bits. This can be attributed to the statistical sample size (1000 measurements) and the fact that *SV4* gives more noisy measurements because it is the weakest signal. This can be justified through Figure 8.8, which plots the biases based on a 100-block sample size. Here, the range biases appear to be more severe when in fact the biases are due to the small sample size. Therefore, judging by the results shown in Figure 8.7 and 8.8 alone, it can be deduced that the system has little or no biases.

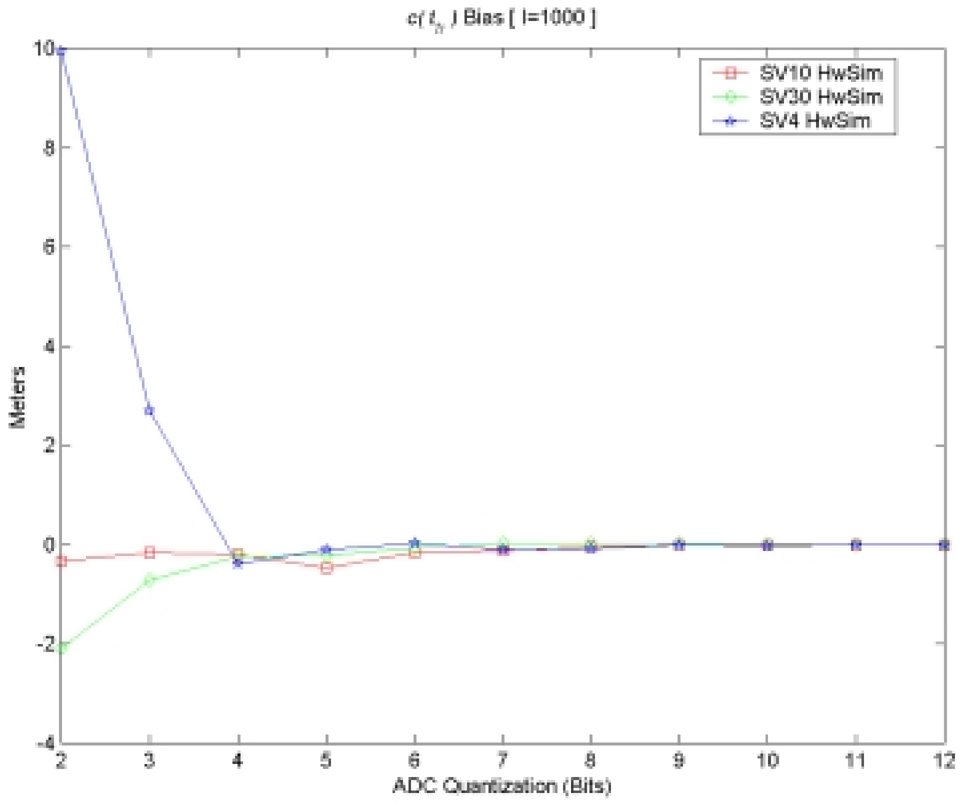


Figure 8.7 Ranging Bias Due to ADC Quantization (I=1000)

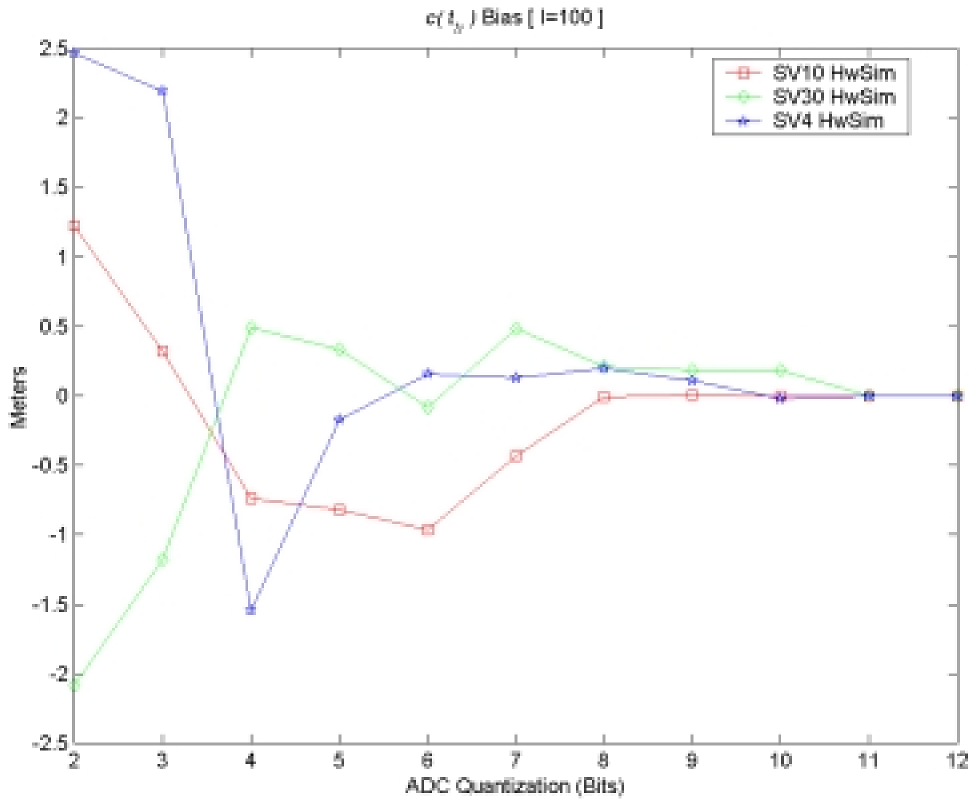


Figure 8.8 Ranging Bias Due to ADC Quantization (I=100)

Figures 8.9 and 8.10 show how the acquisition margin is degraded across all SVs as the ADC quantization is reduced. The acquisition margin suffers a loss of only 2 dB when a 2-bit quantizing ADC is used. These figures clearly show how inexpensive consumer GPS receivers are able to use a 1-bit ADC to sample the RF signal and still successfully acquire GPS satellites despite a 3 dB loss in gain. The results give a feel for the robustness of the GPS signal and its ability to provide service even under reasonably severe conditions (hardware limitations in this case). However, it should be noted that a 2-bit quantizing ADC seriously degrades the interference rejection performance of the receiver since any type of sufficiently strong interference, such as a continuous wave (CW) signal can saturate the ADC and completely blank out the signal of interest.

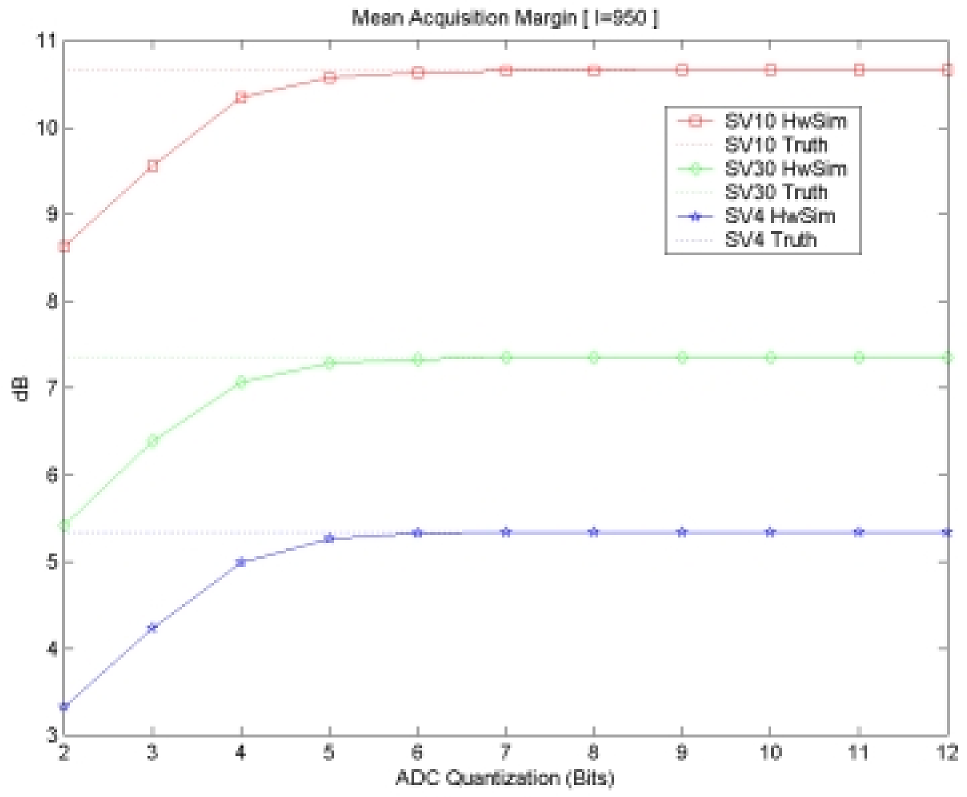


Figure 8.9 Effect of ADC Quantization on the Signal Acquisition Margin

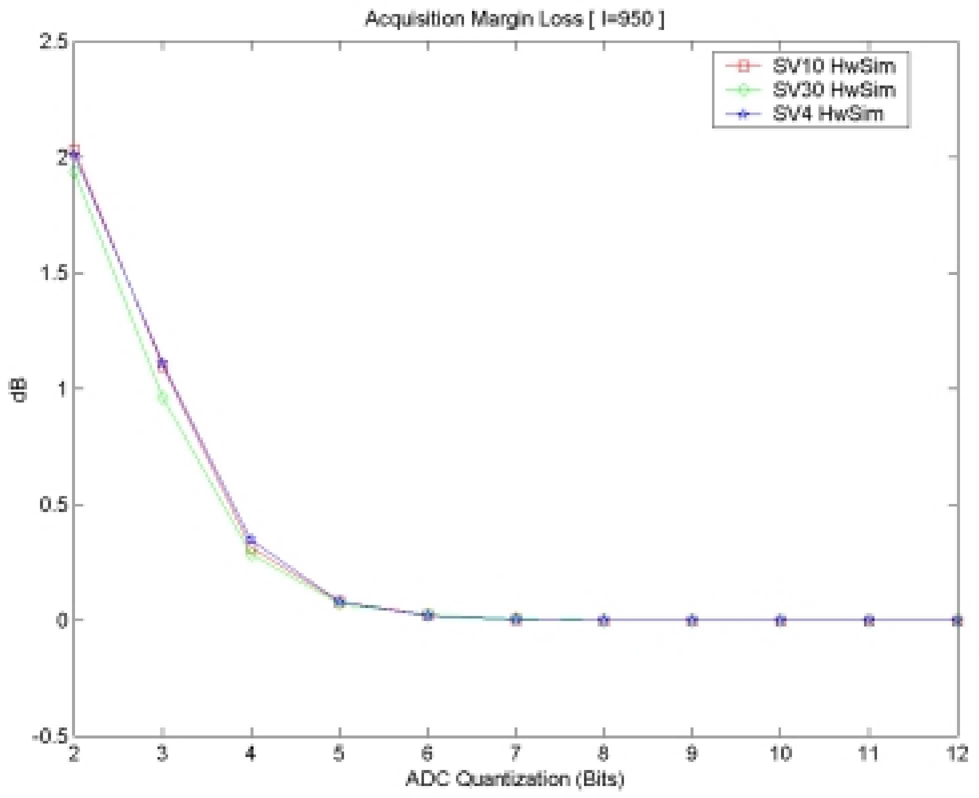


Figure 8.10 Loss of Acquisition Margin due to ADC Quantization

8.3 Effect of Carrier NCO Amplitude Quantization

For this and the subsequent simulation in Section 8.4, the ADC quantization, W_x, is fixed at eight bits. This was done because only the effect of the carrier NCO amplitude quantization was of interest. After the incoming signal is mixed with the NCO generated carrier, it was processed exactly as the truth in Matlab™. Hence, only the effect of the carrier NCO quantization can be seen in these simulations.

Figures 8.11 and 8.12 show the standard deviation of the range in absolute values and as an inflation percentage, respectively.

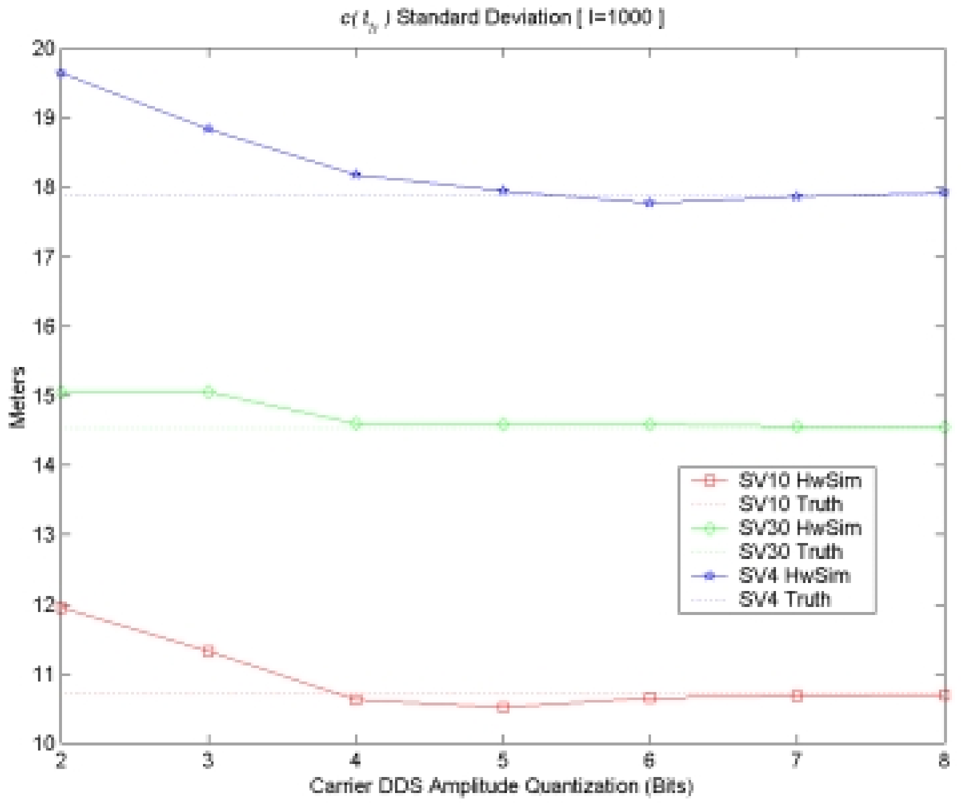


Figure 8.11 Effect of Carrier NCO Amplitude Quantization on $c(t_{tr})$ Standard Deviation

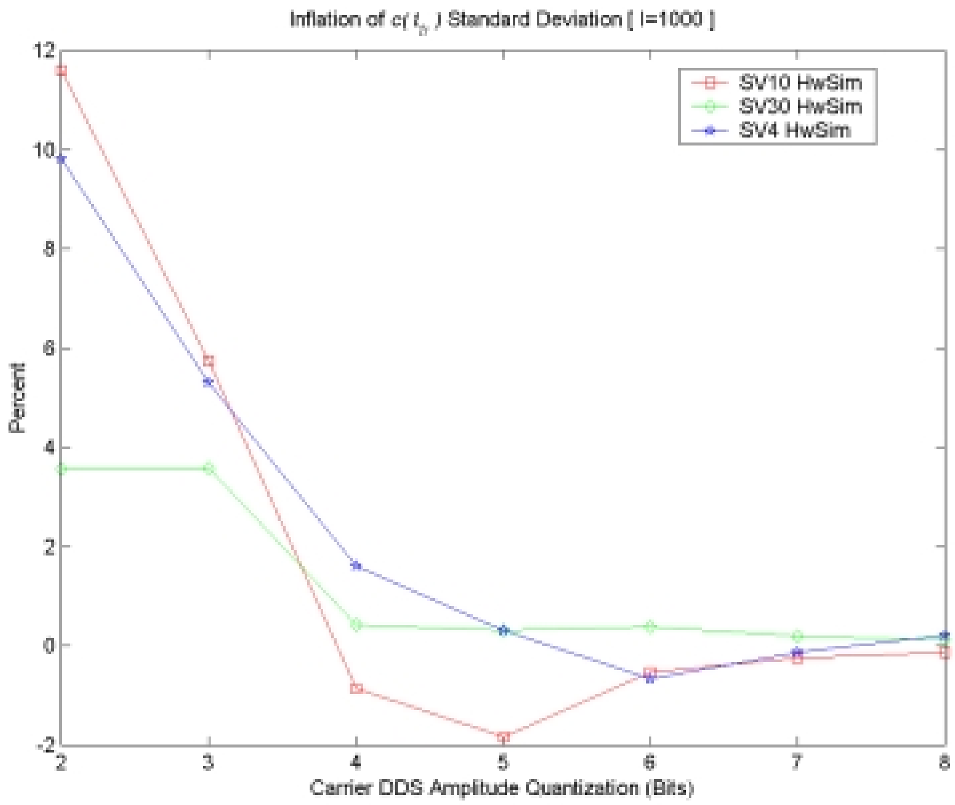


Figure 8.12 Inflation of $c(t_{tr})$ Standard Deviation With Respect to the Truth

A 2-bit NCO quantization gives only a 12% deviation in range deviation compared to the truth. Since this is acceptable for most applications, the quadrature mixer can be implemented without multipliers using only a multiplexer (to switch the Digital IF signal between the *I* and *Q* components) and a W_x-bit adder, which is configured as a two's complementor (to negate the data value).

Figure 8.13 shows the bias in the range deviation compared to the truth. The figure shows some bias in the processing, however it is small enough to be safely neglected.

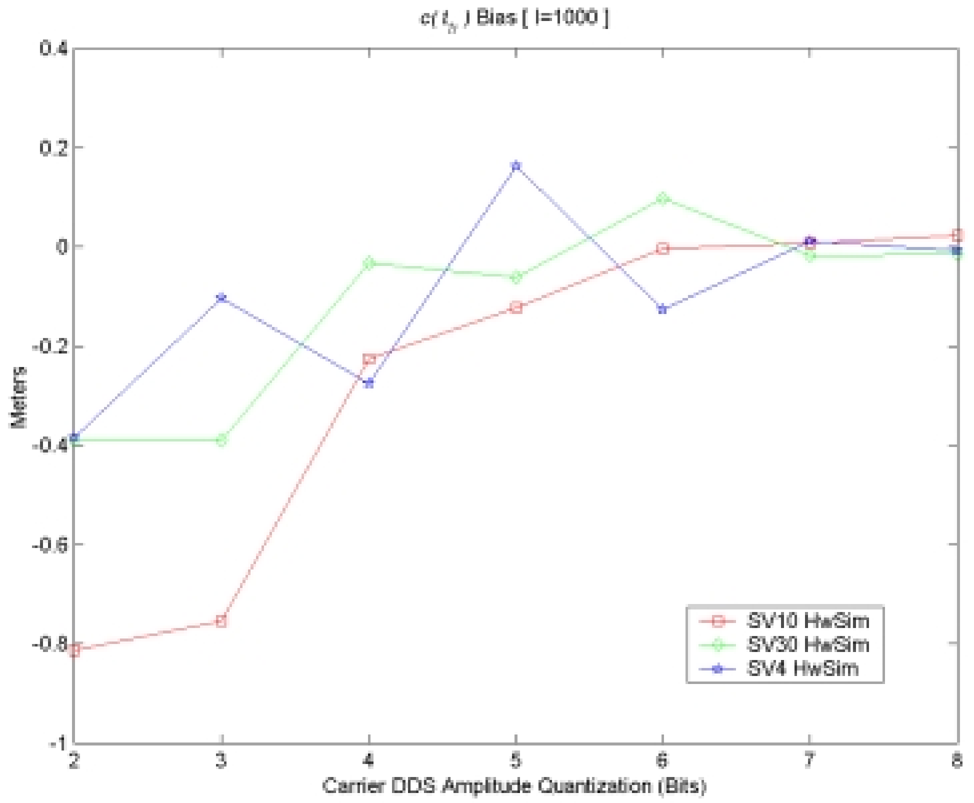


Figure 8.13 Ranging Bias Due to Carrier NCO Amplitude Quantization (I=1000)

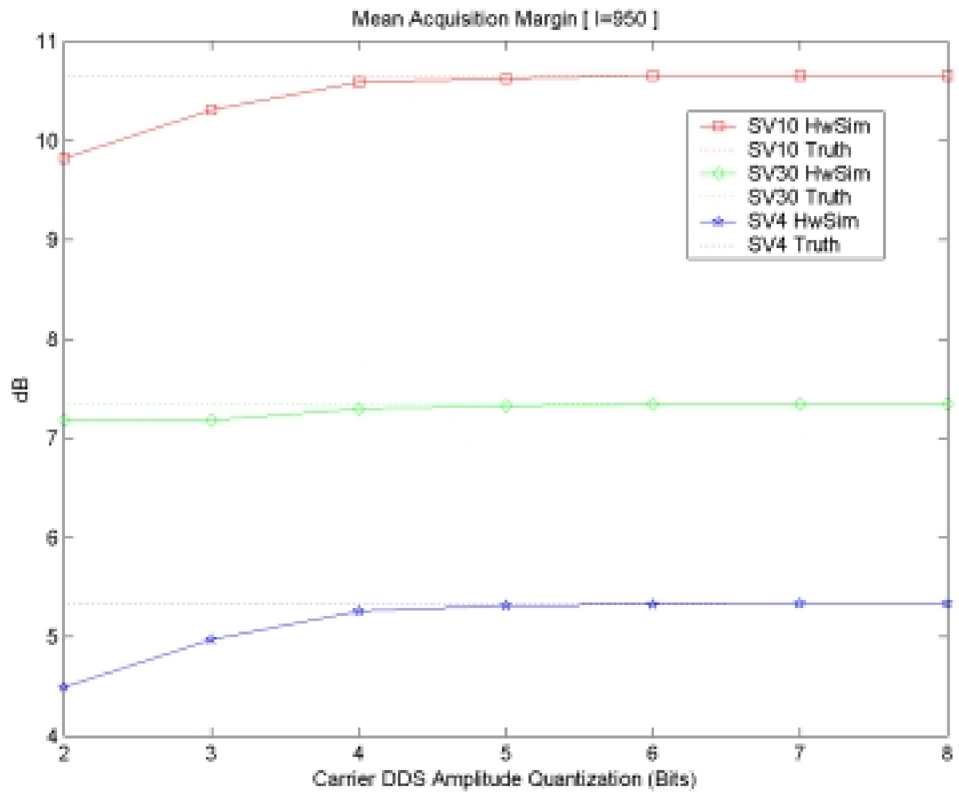


Figure 8.14 Effect of Carrier NCO Amplitude Quantization on the Signal Acquisition Margin

Figures 8.14 and 8.15 show the mean acquisition margin, and the associated implementation loss due to NCO quantization. It can be seen that the effect due to NCO quantization can be safely neglected for relatively strong signals (above 46 dB-Hz), but does pose a problem for weak signals such as the 44 dB-Hz *SV4* signal in the dataset. However, the 0.9 dB loss can be recovered through block addition techniques [Section 4.4] to detect weak signals even with a 2-bit NCO quantization.

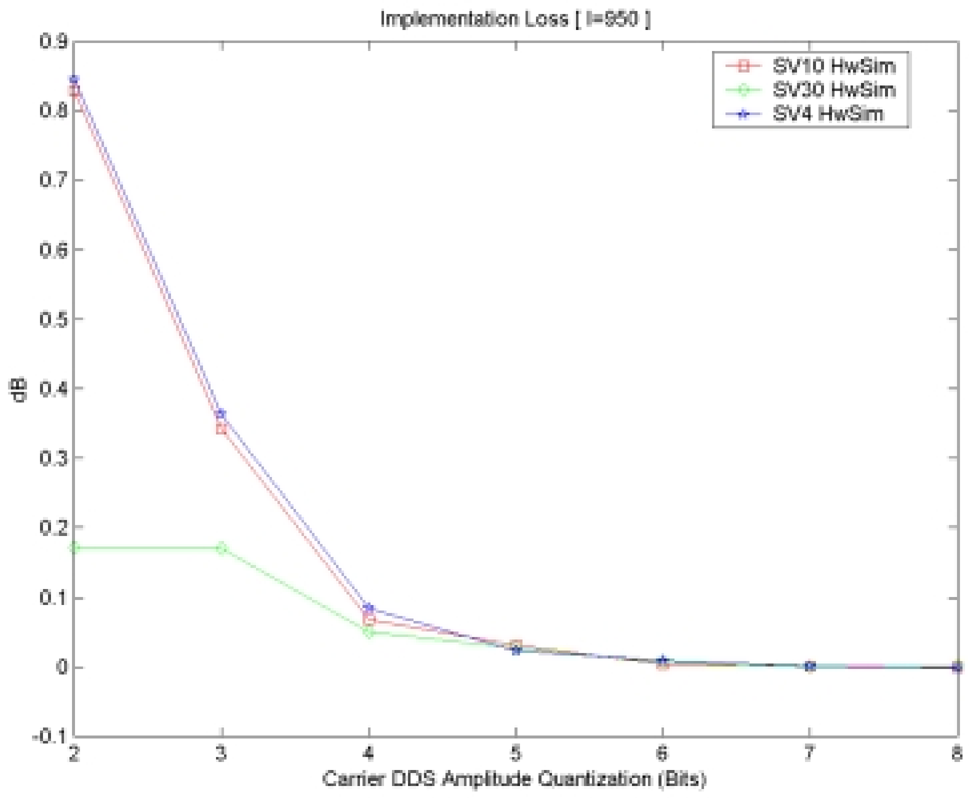


Figure 8.15 Implementation Loss due to Carrier NCO Amplitude Quantization

8.4 Effect of Finite-Precision Hardware Processing

To study the effect of finite precision processing for the GPS fast correlator, the hardware datapath section shown in Figure 7.1 was simulated. For this simulation, both the ADC quantization and the carrier NCO amplitude quantization are set to 8 bits (i.e. W_x=W_{NCO}=*8*). The hardware datapath width was varied from 8 to 22 bits. As the hardware datapath is increased, the *I* and *Q* components from the quadrature mixer are scaled such that the optimum dynamic range of the hardware datapath is utilized as shown in Figure 7.1. Scaling is also performed for the C/A code transform values. Figures 8.16 and 8.17 show the standard deviation absolute values and the inflation percentages compared to the truth, respectively. The figures show that the range deviation is approximately 10% when the datapath is 10 bits wide.

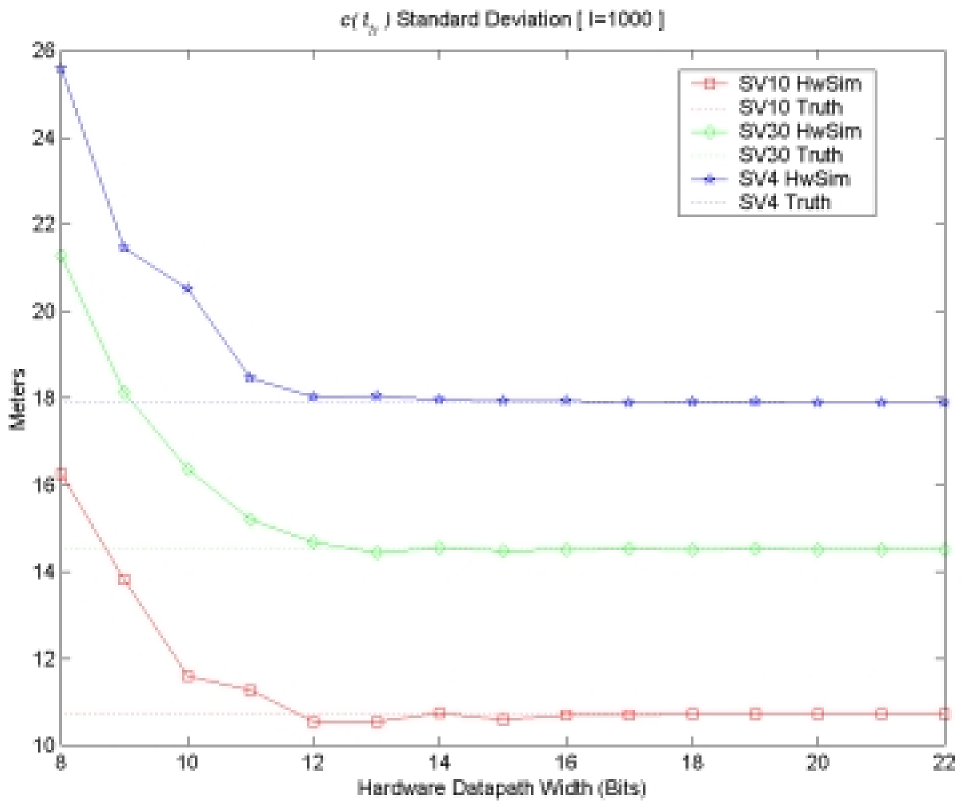


Figure 8.16 Effect of Hardware Pipeline Width on $c(t_{tr})$ Standard Deviation

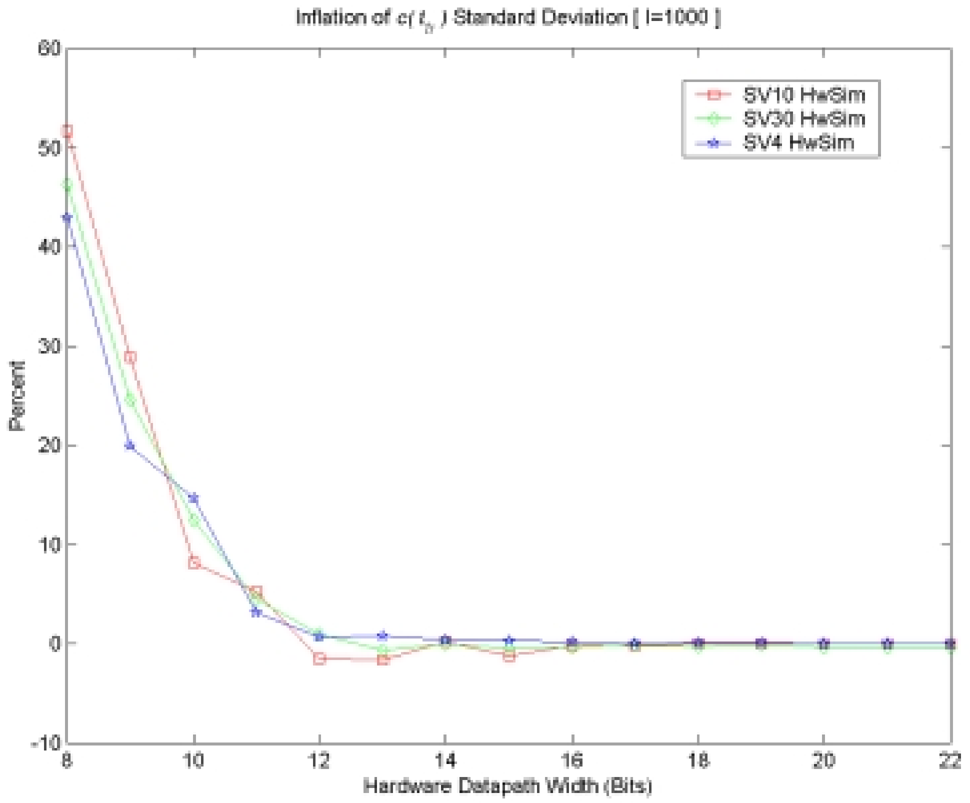


Figure 8.17 Inflation of $c(t_{tr})$ Standard Deviation With Respect to the Truth

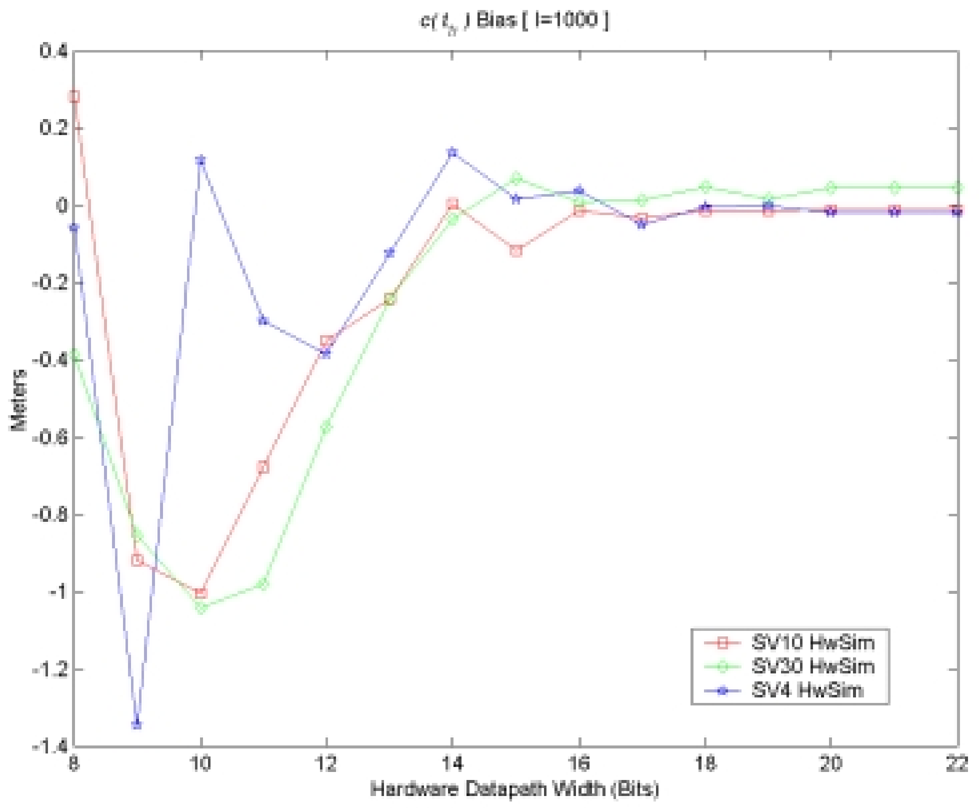


Figure 8.18 Ranging Bias Due to Finite Precision Processing

Figure 8.18 shows the bias in the range deviation measurements compared to the truth. It can be seen that for datapath widths less than 13 bits, the bias is around 1.5 meters for the weakest signal. Since the bias has not increased significantly compared to the NCO quantization simulation, and since this processing was subject to much more finite precision processing steps than the NCO quantization case, the bias can be safely neglected.

Figures 8.19 and 8.20 show how the finite precision processing has affected the acquisition margin of the signal and added implementation loss, respectively. An 8-bit hardware datapath adds approximately 3 dB of implementation loss. However, the loss drastically reduces to 0.5 dB for a 10-bit datapath. Since not much performance is gained beyond 10 bits of precision for either range deviation (<5%) or implementation loss (<0.5 dB), 10 bits seems to be the ‘magic number’ for the width of the hardware datapath.

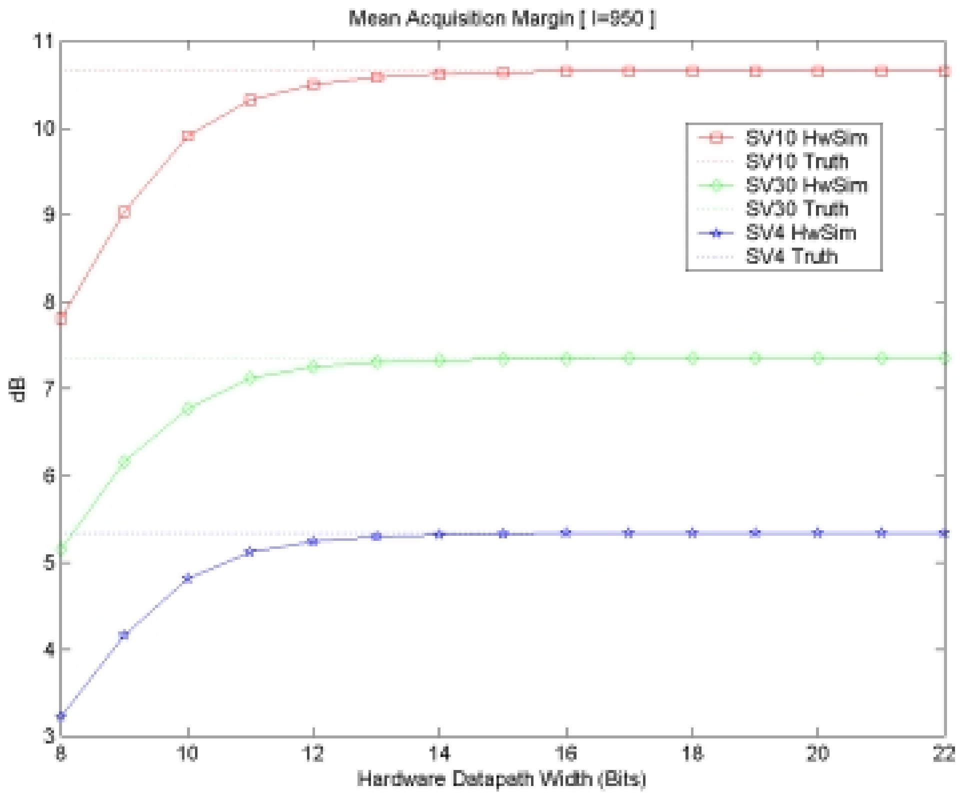


Figure 8.19 Effect of Pipeline Width on the Signal Acquisition Margin

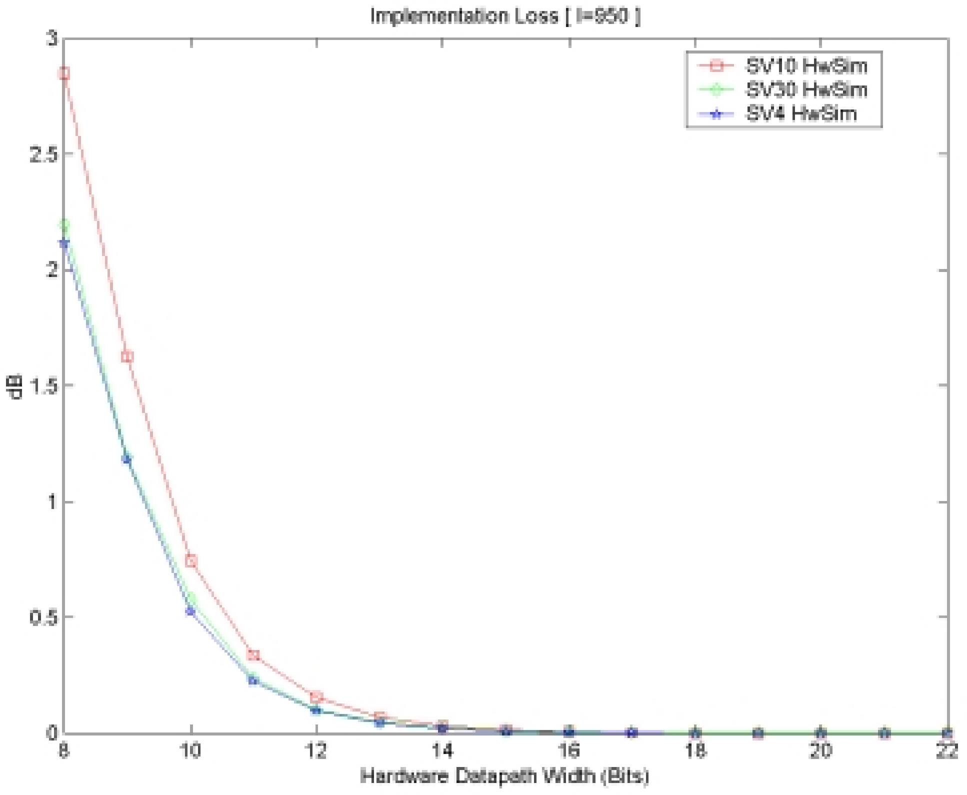


Figure 8.20 Implementation Loss due to Finite Precision

8.5 The Optimum Block Processing Hardware Architecture

From all of the preceding simulations, the following deductions can be made:

1) The hardware datapath must implement rounding rather than merely truncating the partial results. The cost associated with rounding is negligible compared to the accuracy gained.

2) The ADC quantization can be as low as 2-bits without affecting performance. However there is a definite need to have more bits for the sake of interference rejection. If $W<W_x$, W sets the upper bound for the ADC quantization.

3) The carrier NCO amplitude quantization can be limited to 2-bits without sacrificing much performance. The biggest advantage to this is the fact that the quadrature mixer can be implemented without multipliers, hence saving hardware resources.
4) Using a hardware datapath greater than 10 bits does not improve performance compared to the cost of implementation since the hardware area, interconnections and overall complexity grows exponentially with every bit of precision. The simulations show that $W=10$ is optimal for the hardware datapath.

Based on these deductions, the optimum hardware architecture that minimizes processing error and hardware resources simultaneously is one that has:

- 8-bit ADC quantization ($W_x = 8$)
- 2-bit carrier NCO amplitude quantization ($W_{NCO} = 2$)
- 10-bit FFT/IFFT datapath ($W = 10$)

Figures 8.21 through 8.25 show the simulation results for range deviation, range deviation inflation, ranging bias, acquisition margin, and implementation loss for this optimum hardware architecture, respectively.

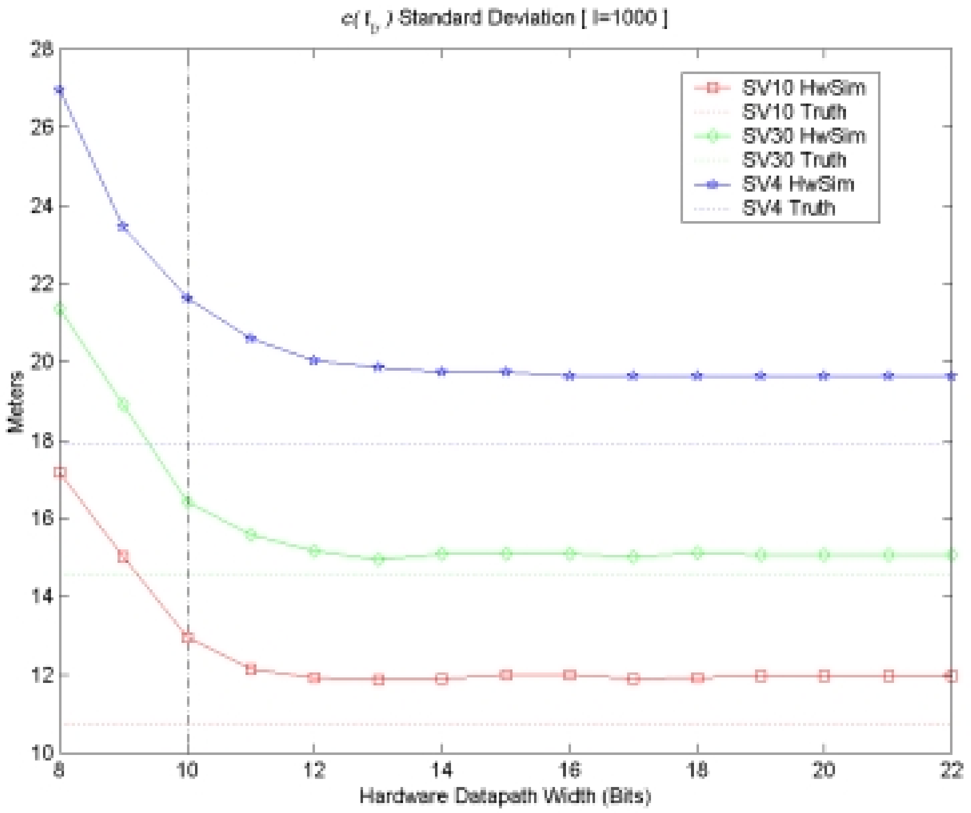


Figure 8.21 Range Deviation of Target Hardware Architecture

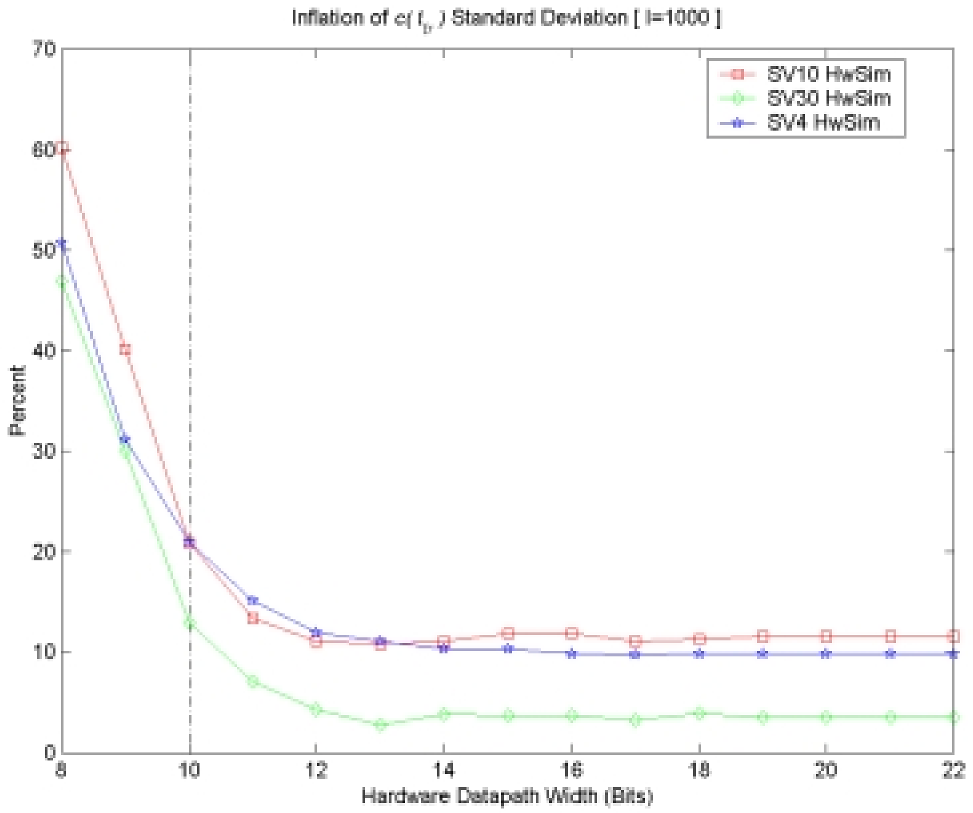


Figure 8.22 Range Deviation Inflation for Target Hardware Architecture

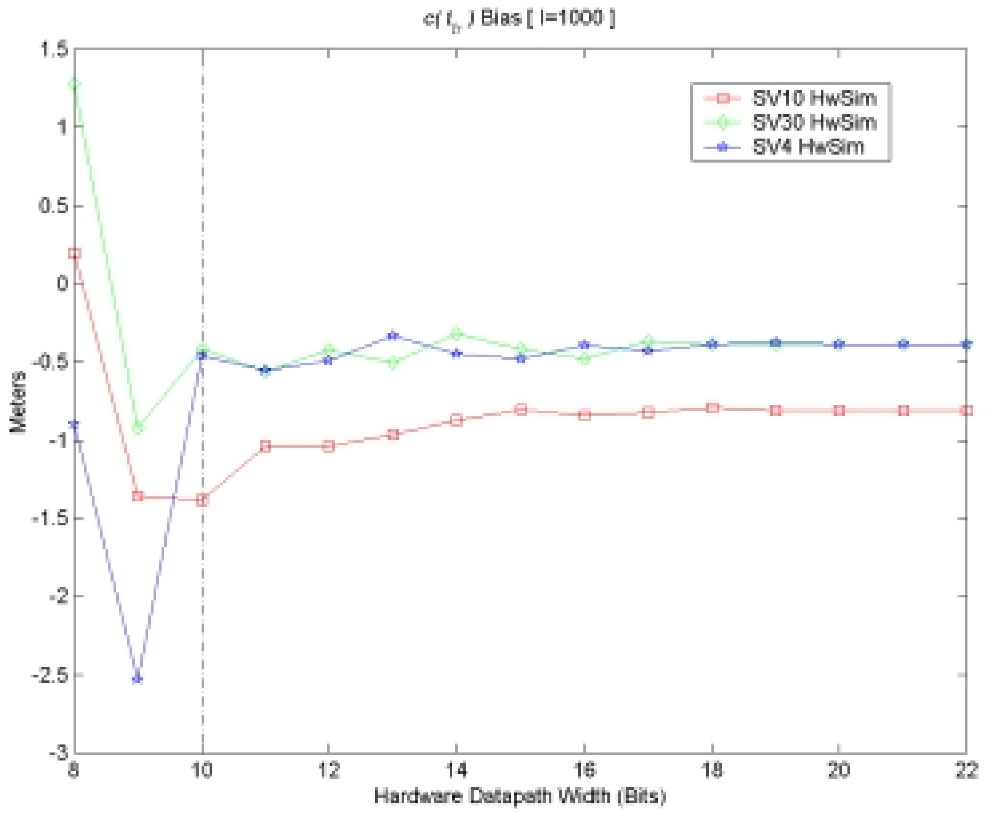


Figure 8.23 Ranging Bias for Target Hardware Architecture

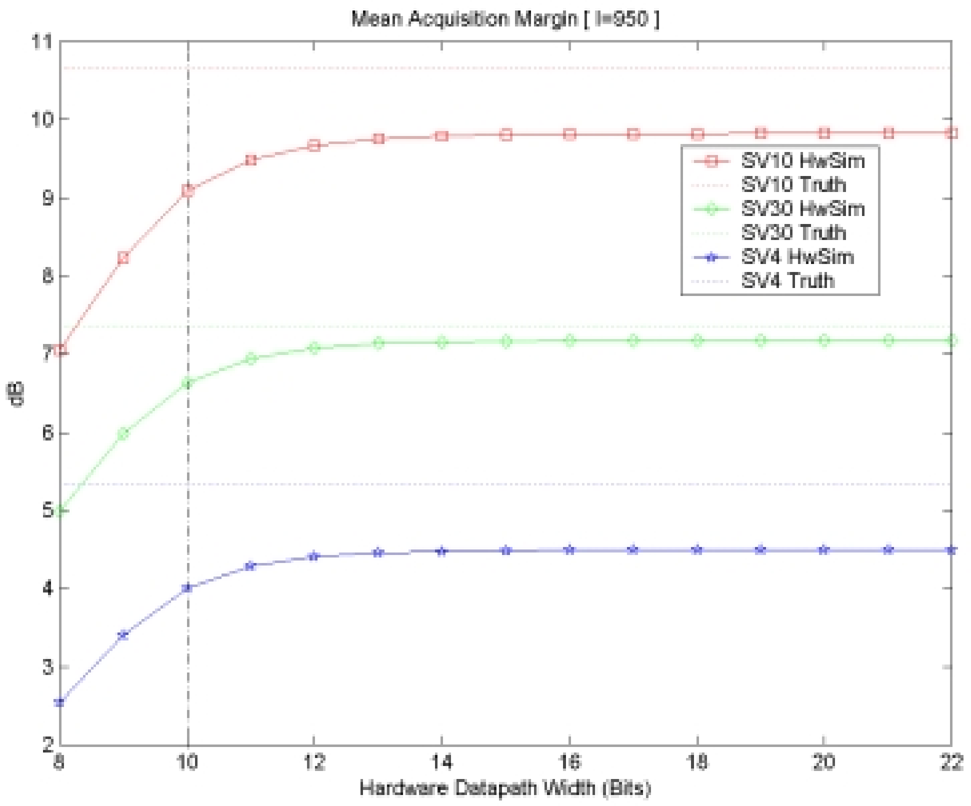


Figure 8.24 Acquisition Margin for Target Hardware Architecture

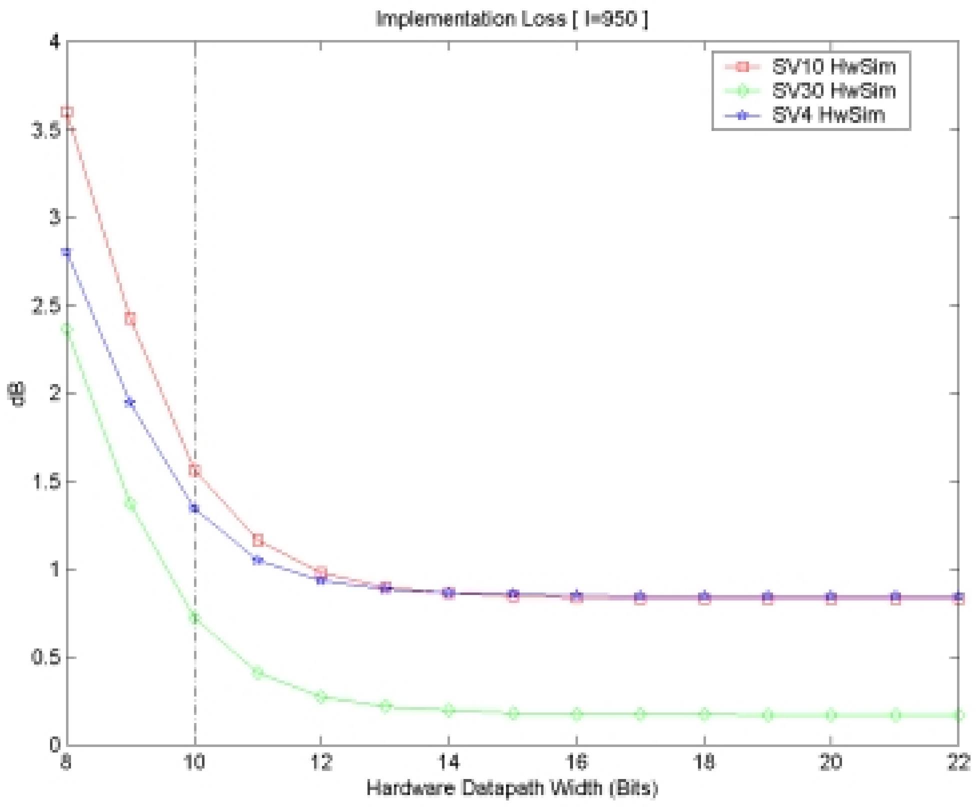


Figure 8.25 Implementation Loss for Target Hardware Architecture

9 SUMMARY AND CONCLUSIONS

This work studied the feasibility of implementing GPS block processing techniques in FPGA hardware, for the ultimate goal of realizing a fully functional real-time block-processing GPS receiver. This is the first time a study to migrate the block processing algorithms previously developed in software, to custom designed FPGA hardware has been carried out. The FPGA was chosen as the implementation platform due to its ASIC-like performance, software-like programmability, and because it is the ideal platform for research. The rapid development of FPGA technology has replaced ASICs in many areas. It is believed that it is only a matter of time before the hardware architectures presented in this thesis can be made to run in real-time using FPGAs.

This thesis presented the architecture of traditional GPS receivers and described the novel block processing approach. The architecture of a real-time block-processing GPS receiver as applied to an airborne application was presented. The high-level system architecture of the module of interest in this receiver, namely the FPGA processor, was described. Subsystems that make up the FPGA processor, such as the carrier NCO and complex multipliers and their internal architectures were described. Implementing the 5000-point FFT in hardware represents the biggest challenge for achieving the real-time processing goal. The design of a 5000-point FFT/IFFT algorithm based on the mixed radix approach was described. The hardware design of each FFT building block that comprised the 5000-point FFT was presented. Methods for optimizing the 5000-point FFT for FPGA implementation such as vertical projection, PE partitioning, datapath cycle optimization, and speed/performance estimation were presented. Lastly, results of behavioral simulations based on the developed hardware architecture using real GPS data were presented. From these results, the optimum hardware pipeline parameters were deduced.

This work is considered a feasibility study since only the high level architecture of the block processing techniques were developed and simulated. Only acquisition margin loss and pseudorange error deviations (from code tracking) were considered as performance measures in this work. Other performance characteristics, such as carrier phase and frequency tracking ability, accumulated Doppler measurement accuracy, and navigation data extraction reliability remains to be tested for the hardware simulation models developed in this research. If the presented architecture passes these tests, then the work started in this thesis can be continued towards the development of an RTL description of the hardware. Once this is completed, the presented FPGA design flow can be continued. Furthermore, when the RTL descriptions are complete, accurate insight into the hardware resources needed and knowledge of other architectural features such as memory size and throughput requirements, board and bus architecture requirements, and embedded processor requirements will be gained. Armed with this knowledge, a suitable FPGA COTS product for implementing the hardware can be acquired.

Through high-level architecture design and simulations, this research has laid the groundwork for achieving the goal of an FPGA-based real time block processing GPS receiver. Ultimate realization of this new GPS receiver will elevate the performance and capabilities of GPS to a hitherto unattained dimension in terms of novel applications and integration into existing and future technologies.

REFERENCES

[Ackenhusen99] J. G. Ackenhusen, Real-Time Signal Processing: Design and Implementation of Signal Processing Systems, Prentice Hall PTR, 1999.

[Akos97] D. A. Akos, A Software Radio Approach to Global Navigation Satellite System Receiver Design, *Ph.D. Dissertation*, Ohio University, August 1997.

[AP00] Associated Press, "U.S. to Offer More Accurate Satellite Navigation To Everyone," *Press Release*, May 2, 2000.

[Braash91] M. S. Braasch and F. van Graas, "Guidance Accuracy Considerations for Real-Time GPS Interferometry," *Proc. ION GPS-91*, Albuquerque, September 1991.

[Compaq97] Compaq Corporation, "PCI Pamette V1," *webpage*, www.research.digital.com/SRC/pamette/, 1997.

[Cooley65] J.W. Cooley and J.W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, vol. 19, pp. 297-301, April 1965.

[Feng99] G. Feng and F. van Graas, "GPS Receiver Block Processing", *Proc. ION GPS-99*, pp. 307-316, Nashville, September 1999.

[ICDGPS97] GPS Joint Program Office, "Navstar GPS Space Segment / Navigation User Interfaces ICD" Rev. IRN-200C-002," September 25, 1997.

[Kaplan96] E. D. Kaplan, ed., Understanding GPS: Principals and Applications, pp 193-198, Artech House Publishers, 1996.

[Knight79] W. R. Knight and R. Kaiser, "A Simple Fixed-Point Error Bound for the Fast Fourier Transform," *IEEE Trans. on ASSP*, vol. ASSP-27, no. 6, December 1979.

[Madisetti98] V. K. Madisetti and D. B. Williams, eds., The Digital Signal Processing Handbook, IEEE Press, 1998.

[Matlab99] "Matlab® Compiler 2.0 & Matlab C/C++ Math Library 2.0," *Datasheet*, The Math Works Inc., www.mathworks.com, 1999.

[Moeglein98] M. Moeglein and N. Krasner, "An Introduction to SnapTrack™ Server-Aided GPS Technology," *Proc. ION GPS-98*, pp. 333-342, Nashville, September 1998.

[Oppenheim89] A. V. Oppenheim and R. W. Schafer, Discrete-Time Signal Processing, Prentice Hall, 1989.

[Parkinson96] B. W. Parkinson and J. J. Spilker, eds., Global Positioning System: Theory and Applications, vol. 1, pp. 329-407, American Institute of Aeronautics and Astronautics, 1996.

[Pirsch98] P. Pirsch, Architectures for Digital Signal Processing, John Wiley and Sons, 1998.

[Rabiner75] L. R. Rabiner and B. Gold, Theory and Application of Digital Signal Processing, pp. 587-594, Prentice Hall Publishers, 1975.

[Roelandts99] W. Roelandts, "15 Years of Innovation," *Xcell Magazine*, Iss. 32, Second Quarter, Xilinx Publication, 1999.

[Smith95] W. W. Smith and J. M. Smith, Handbook of Real-Time Fast Fourier Transforms, IEEE Press, 1995.

[Snyder99] C. A. Snyder, G. Feng, and F. van Graas, "GPS Anomalous Event Monitor (GAEM)," *Proc. ION 55th Annual Meeting*, pp. 185-189, Cambridge, June 99.

[Spilker78] J. J. Spilker, "GPS Signal Structure and Performance Characteristics," *Navigation*, vol. 25, no. 2, pp. 121-130, Summer 1978.

[Tsui97] J.B.Y. Tsui, M. Stockmaster, and D. Akos, "Block Adjustment of Synchronizing Signal for GPS Receiver Signal Processing," *Proc. ION GPS-97*, pp. 637-644, Kansas City, September 1997.

[UijtdeHaag99] M. Uijt de Haag, An Investigation into the Application of Block Processing Techniques for the Global Positioning System, *Ph.D. Dissertation*, Ohio University, August 1999.

[USDOT95] U.S. Department of Transportation, Global Positioning System Standard Positioning Service Signal Specification, ed. 2, June 2 1995.

[vanNee91] D.J.R. van Nee and A. J. R. M. Coenen, "New Fast GPS Code-Acquisition Technique Using FFT," *Electronic Letters*, vol. 27, no. 2, January 1991.

[Vaughan91] R. G. Vaughan, "The Theory of Bandpass Sampling," *IEEE Trans. on Signal Processing*, vol. 39, no. 9, September 1991.

[WH00] The White House, “Statement by the President Regarding the United States’ Decision to Stop Degrading Global Positioning System Accuracy,” *Press Release*, May 1, 2000.

[Wolf94] W. Wolf, Modern VLSI Design: A Systems Approach, pp. 231-239, Prentice Hall, 1994.

[Xilinx99] The Programmable Logic Data Book 1999, Xilinx Publication, 1999.

ABSTRACT

Gunawardena, Sanjeev M.S. November 2000
Electrical Engineering

FEASIBILITY STUDY FOR THE IMPLEMENTATION OF GLOBAL POSITIONING SYSTEM BLOCK PROCESSING TECHNIQUES IN FIELD PROGRAMMABLE GATE ARRAYS (132 pp.)

Director of Thesis: Dr. Janusz A. Starzyk

The Global Positioning System represents the pinnacle of navigation technology for the 21st century. As new technologies integrate GPS services, the limited availability of GPS in environments where the signal is severely attenuated, subject to strong multipath or high dynamics becomes an obstacle to a rapidly growing industry. A novel scheme for processing the GPS signal, namely a software radio employing block-processing techniques similar to those used for image processing has proven to enhance the usability of GPS in such environments. However, these techniques have huge computational requirements that are impossible to meet with a microprocessor. Custom designed hardware, such as an application specific integrated circuit (ASIC) would handle the processing requirement, but defeats the philosophy of a software radio since the algorithms cannot be changed. Field programmable gate arrays (FPGAs) are beginning to replace ASICs in certain applications since they feature software-like re-programmability while approaching ASIC-like performance. FPGAs are excellent candidates for research since they lack the NRE costs associated with ASICs. Hence, FPGAs are the most attractive implementation platform for developing a real-time block-processing GPS receiver.

This work lays the groundwork for the implementation of a real-time block-processing GPS receiver in FPGA hardware. It is a feasibility study since the problem is approached at a high-level of abstraction. The original block-processing approach is re-analyzed for implementation in FPGA hardware. Implementing the 5000-point FFTs in finite-precision hardware represents one of the biggest challenges in this work. This requires analysis of the FFT error bound to determine the minimum precision required that would yield acceptable results while minimizing hardware cost. Even though the analytical error bound for finite-precision FFTs is well documented in past literature, its direct application to the block-processing problem becomes too complex. This work employs statistical results of simulations to deduce the optimum hardware architecture and concludes that real-time capability can be achieved with currently available technology.

Approved: ____
Signature of Director