12-2005

Optimization of DSSS Receivers Using Hardware-in-the-Loop Simulations

Balbir Kaur Dhillon
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a thesis written by Balbir Kaur Dhillon entitled "Optimization of DSSS Receivers Using Hardware-in-the-Loop Simulations." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Mostofa K. Howlader, Major Professor

We have read this thesis and recommend its acceptance:

Michael J. Roberts, Donald Bouldin, Miljko Bobrek

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Balbir Kaur Dhillon entitled "Optimization of DSSS Receivers Using Hardware-in-the-Loop Simulations." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Mostofa K. Howlader
Major Professor

We have read this thesis
and recommend its acceptance:

Michael J. Roberts

Donald Bouldin

Miljko Bobrek

Accepted for the Council:

Anne Mayhew
Vice Chancellor and
Dean of Graduate Studies

(Original signatures are on file with official student records.)

Optimization of DSSS Receivers Using Hardware-in-the-Loop Simulations

A Thesis Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Balbir Kaur Dhillon

December 2005

Dedication

This thesis is dedicated first and foremost to my parents, Sukhdev and Parminder, for always encouraging me to pursue my dream. Also, I would like to contribute this thesis to my loving family, my professors who have inspired me over the years, my friends who have enriched my college experiences and my fellow engineering students for providing an entertaining atmosphere.

Acknowledgements

I would like to thank everyone who has helped me attain a Masters of Science degree in Electrical Engineering. I would especially like to thank Dr. Howlader for giving me the opportunity to work as a research assistant and for encouraging me to strive for the best. I would like to thank Dr. Bobrek for helping me better understand how to apply my communications knowledge in real world applications. I could not have completed my research without his help. I would also like to acknowledge my committee members, Dr. Roberts and Dr. Bouldin. I am grateful for taking Dr. Bouldin's classes since they have helped me in the hardware implementation aspect of my thesis. I am grateful to Dr. Roberts for providing me with excellent advice when I was an undergraduate. Also, I would like to thank fellow students in the Wireless Communication Research Group for all their help.

This research work has been performed in the Wireless Communications Research Group (WCRG) at University of Tennessee, Knoxville, sponsored by the RF & Microwave System Group (RFMSG) of the Oak Ridge National Laboratory (ORNL) under the contract UT-B 4000025441 and UT research account number R011344113. I am also especially indebted to Paul Ewing, leader of the RFMSG, who has provided me with this great opportunity.

Abstract

Over the years, there has been significant interest in defining a hardware abstraction layer to facilitate code reuse in software defined radio (SDR) applications. Designers are looking for a way to enable application software to specify a waveform, configure the platform, and control digital signal processing (DSP) functions in a hardware platform in a way that insulates it from the details of realization.

This thesis presents a tool-based methodolgy for developing and optimizing a Direct Sequence Spread Spectrum (DSSS) transceiver deployed in custom hardware like Field Programmble Gate Arrays (FPGAs). The system model consists of a tranmitter which employs a quadrature phase shift keying (QPSK) modulation scheme, an additive white Gaussian noise (AWGN) channel, and a receiver whose main parts consist of an analog-to-digital converter (ADC), digital down converter (DDC), image rejection low-pass filter (LPF), carrier phase locked loop (PLL), tracking locked loop, down-sampler, spread spectrum correlators, and rectangular-to-polar converter.

The design methodology is based on a new programming model for FPGAs developed in the industry by Xilinx Inc. The Xilinx System Generator for DSP software tool provides design portability and streamlines system development by enabling engineers to create and validate a system model in Xilinx FPGAs. By providing hierarchical modeling and automatic HDL code generation for programmable devices, designs can be easily verified through hardware-in-the-loop (HIL) simulations.

HIL provides a significant increase in simulation speed which allows optimization of the receiver design with respect to the datapath size for different functional parts of the receiver. The parameterized datapath points used in the simulation are ADC resolution, DDC datapath size, LPF datapath size, correlator height, correlator datapath size, and rectangular-to-polar datapath size. These parameters are changed in the software enviornment and tested for bit error rate (BER) performance through real-time hardware simualtions. The final result presents a system design with minimum harware area occupancy relative to an acceptable BER degradation.

Table of Contents

List of Figures

List of Abbreviations

ADC	Analog to Digital Converter
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
BER	Bit Error Rate
BPF	Bandpass Filter
CE	Clock Enable
CHARIOT	Changeable Advanced Radio for Inter-Operable Telecommunications
CORDIC	Coordinate Rotational Digital Computer
CPLD	Complex Programmable Logic Device
DA	Distributed Arithmetic
DAC	Digital to Analog Converter
DCM	Device Clock Manager
DDC	Digital Down Conversion
DDS	Digital Down Synthesizer
DIME	DSP and Image processing Modules for Enhanced FPGAs
DoD	Department of Defense
DRC	Design Rule Check
DSP	Digital Signal Processing
DSSS	Direct Sequence Spread Spectrum
DUC	Digital Up Conversion
ECS	Engineering Capture System
FF	Flip Flops
FFT	Fast Fourier Transform
FHSS	Frequency Hopping Spread Spectrum
FIFO	First In First Out
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
FUSE	Field Upgradeable Software Environment
GCD	Greatest Common Divisor

GPP	General Purpose Processors
GUI	Graphic User Interface
HIL	Hardware-in-the-Loop
I	In-phase Component
IF	Intermediate Frequency
IP	Intellectual Property
ISE	Integrated Software Environment
ISI	Inter-symbol Interference
JTAG	Joint Test Access Group
JTRS	Joint Tactical Radio System
LED	Light Emitting Diode
LFSR	Linear Feedback Shift Register
LNA	Linear Noise Amplifier
LO	Local Oscillator
LPF	Lowpass Filter
LUT	Lookup Table
MAC	Multiplier and Accumulate
MCSS	Multi Carrier Spread Spectrum
MXE	ModelSim Xilinx Edition
NCO	Numerically Controlled Oscillator
NGD	Native Generic Database
PAR	Place and Route
PCI	Peripheral Controller Interface
PLL	Phase Lock Loop
PN	Pseudo noise
Q	Quadrature Component
QPSK	Quadrature Phase Shift Keying
RAF	Running Average Filter
RAM	Read Access Memory
RF	Radio Frequency
ROM	Read Only Memory
RTL	Register Transfer Language
SCA	Software Communication Architecture

SDR	Software Defined Radio
SS	Spread Spectrum
THSS	Time Hopping Spread Spectrum
UCF	User Constraint File
USB	Universal Serial Bus
VCO	Voltage Controlled Oscillator
VHDL	Verilog Hardware Description Language
VHF	Very High Frequency
WITS	Wireless Information Transfer System
XCF	Xilinx Timing Constraint File
XCO	CORE generator log file
XST	Xilinx Synthesis Technology
ZTB	Zirconium T-Butoxide (memory)

Chapter 1
Introduction

1.1 The Context

The commercial wireless industry is constantly evolving and therefore facing many challenges. These problems arise due to the constant change of link-layer protocols and existence of incompatible wireless technologies in different parts of the world. Software Defined Radio (SDR) technology provides solutions to these problems by implementing radio functionality such as modulation/demodulation, signal generation, coding and link-layer protocols in the form of software modules running on generic hardware platforms [1]. The software modules provide flexibility to the SDR system. By allowing the capability of over-the-air downloads of software modules, the issue of compatibility of different standards is eliminated. To provide key features such as reconfigurability, flexibility, and inter-operability, the SDR architecture needs to be constructed such that applications can function in various environments. Therefore, design verification is of utmost importance.

1.2 The Problem

Software simulation can provide designers insight into system behavior under various internal and external conditions. However, for complex system, software simulation is unable to accurately model every characteristic of a system's behavior. Also, use of software-based true cycle simulators is impractical due to the large number of cycles needed to achieve accurate data statistics. This is especially apparent in simulation of SDR applications that may involve millions of states. Although many solutions have been proposed to increase simulation speed, the necessary computations needed to simulate a radio of low-level complexity exceed the capabilities of mainstream office and lab computers. Therefore, hardware-in-loop (HIL) simulation is provided as an alternative solution due to its significant increase in simulation speed [2]. Many methods for HIL simulation of complex communication systems have been proposed which differ in structure due to emphasis on simulation speed, accuracy, or flexibility [3], [4], [5], and [6].

Implementation of HIL simulations expands the possible applications of time domain simulators and provides hardware-specific results, which are not easily obtained by other techniques. HIL simulation supports development, verification, and integration of complex systems in a systematic process. Integration of physical subsystems into the simulation can provide true system behavior in real-time for verification purposes. Along with verification, the accuracy of the system can be refined by changing system parameters in software.

1.3 Thesis Objective

The objective of this thesis is to provide an optimized communication system design that is completely described in software and implemented on a hardware platform. The communication system will model a parametric Direct Sequence Spread Spectrum transceiver. To speed up the simulation, the design is synthesized and downloaded to a hardware platform to obtain bit-error-rate (BER) performance statistics. Using HIL simulations, the receiver is optimized with respect to datapath size of significant functional blocks. The BER degradation, obtained through real-time simulations, is used as the basis for creating a receiver with minimum hardware area occupancy.

1.4 Thesis Structure

The thesis is organized into seven chapters. Chapter one, which corresponds to this introduction, gives the context, states the problem, and describes the thesis objective. Chapter two provides the motivation for this thesis, and therefore, focuses on SDR. Since hardware implementation is part of the thesis objective, chapter two also reviews the technology background on Field Programmable Gate Arrays (FPGAs). Chapter three provides technical background on Spread Spectrum (SS) systems and describes each functional component of the transceiver used to model the communication system. It also explains the methodology used for optimizing the system design. Chapter four provides extensive details on System Generator, the software package used to model the entire communication system. Also, detailed analysis of digital signal processing (DSP) techniques is discussed in this chapter. Chapter five lays the foundation of the hardware implementation, describing the FPGA architecture and other necessary implementation tools. Chapter six presents the obtained results and explains their

significance. Finally, Chapter seven summarizes the findings of this thesis and gives direction to future work. A set of appendices is provided for details on some of the subjects discussed in this thesis. Appendix A and B provide pin layouts of the I/O devices of the hardware platform. Appendix C is a published paper from the GSP2005 conference that is titled "Optimization of a DSSS receiver Using Hardware Co-Simulation." It provides results of a preliminary DSSS design used for optimization.

Chapter 2
Integration of SDR and FPGAs

Software Defined Radio (SDR) has become the focus of attention in the continuously changing wireless technology. Due to its inherent flexibility and adaptability SDR provides a secure path for wireless transmission. This chapter lays the foundation of SDR, explaining its definition, need, benefits, design architecture, and history. But before presenting SDR, the traditional radio structure and its limitations will be discussed. Since SDR was developed as a solution to these limitations, extensive details on the SDR architectural structure will be presented. Efficient and effective SDR design requires a standard programmable hardware platform that helps designers navigate through tough system requirements. Since FPGAs have been the leading contender in this area, technical background on FPGAs is provided to ensure efficient hardware implementation. Discussion of these topics will provide a glimpse ahead into the core technical contributions of this thesis.

2.1 Traditional Radio Systems

Traditional radios are based on the based on the super-heterodyne receiver circuit. Other than demodulation, receivers must perform carrier-frequency tuning to select the desired signal, filtering to separate the desired signal from other received signals, and amplification to compensate for transmission and implementation loss. As shown in Figure 2.1, a received message carrying an RF signal is down-converted (or mixed down) to baseband in multiple stages. The incoming signal, $x_c(t)$, is received by the antenna and amplified by a radio-frequency (RF) section tuned to the desired carrier frequency f_c. The relatively large bandwidth, B_{RF}, of the amplifier allows some adjacent channel signals to pass through. Next, the RF frequency is brought down to an intermediate frequency (IF) by a frequency converter composed of a mixer and local oscillator (LO). The LO frequency tracks with the RF tuning such that $f_{LO} = f_c \pm f_{IF}$. The signal is then filtered in an IF section to isolate the message-carrying IF carrier and reject the images at ($f_c \pm n \cdot f_{IF}$), where n is an integer greater than zero. Figure 2.2 clarifies this concept. Finally, the signal is sent to a demodulator for message recovery [7].

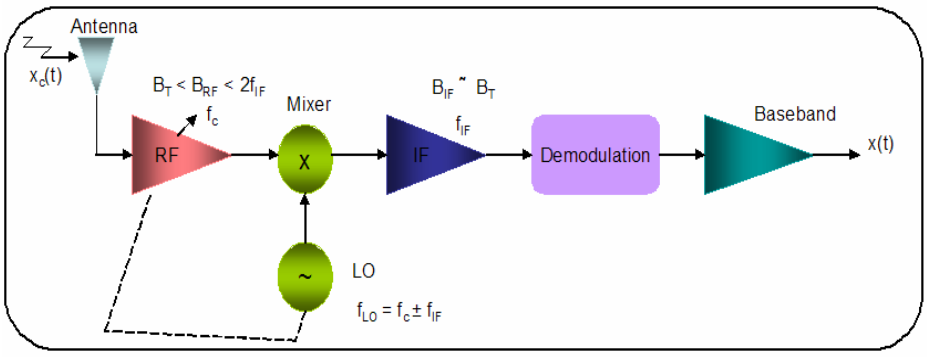

Figure 2.1: Superheterodyne Receiver

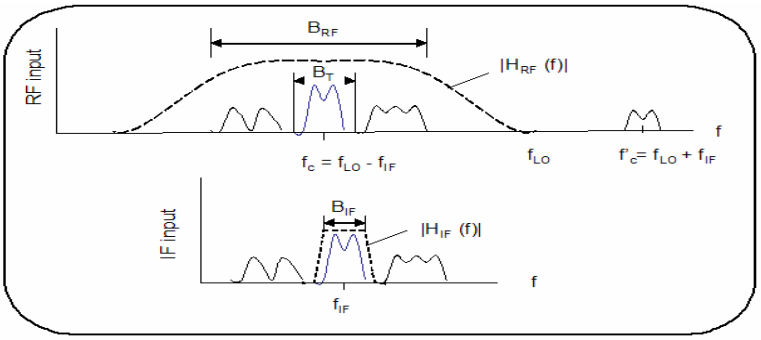

Figure 2.2: Spectral Drawings

This simplified scheme covers the functions carried out by simple devices like traditional AM/FM receivers. Modern transceivers such as base stations and cellular phones require added hardware components that perform more complicated functions such as equalization, frequency hopping and error detection. These modules require more time-consuming and more expensive development and production processes.

The hardware-oriented approach of traditional radios imposes a set of limitations. First, traditional radios have low flexibility to adapt to new services and standards. As shown in the previous paragraphs, each hardware element of the radio chain performs a radio function. These components are designed to operate in a particular frequency band (RF) and standard. When the frequency or any of the parameters of the standard changes, traditional radios cannot correctly extract the information. Before being able to operate under the new conditions, the system must be redesigned and hardware modules have to be replaced. Redesigning, manufacturing and replacing hardware components require high times and costs. Due to the inherent difficulty and

limitations in design implementation of analog signal processing components, designers have migrated to developing a software-based approach to radios.

2.2 A Software Based Approach

A software based approach for radio design, known as software radio, was developed to counteract the drawbacks of traditional technology. By using this methodology, software modules are constructed instead of hardware components to extract information from signals. Figure 2.3 illustrates the architecture for a software-based radio. The chain of hardware components of the traditional radio is replaced by analog-to-digital converters (ADCs), digital-to-analog converters (DACs), and general purpose processors (GPPs) that run the software. ADCs digitalize the analog IF or RF signals. Software modules perform signal processing techniques to extract the information from the digitalized samples. DACs are used to convert the message back into a more suitable form for the user. The use of software enhances flexibility to conform to new features and standards [8].

The transition from traditional radio to software radio can be viewed as a gradual evolution. Over the years designers have been trying to move the digitalization of the signal closer to the antenna. Digitalization right before the RF filter would allow for the most flexibility since the signal would be handled entirely in software. However, this type of digitalization is difficult to implement for high carrier frequencies. Digitalization after the IF filter is the approach currently used in software defined radios. This design requires an RF front end which brings the signal down from the RF frequency to an IF frequency [9].

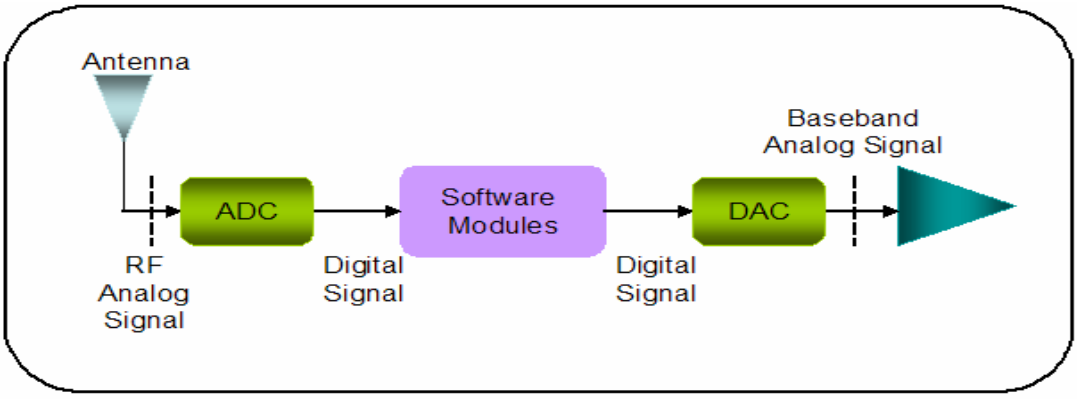

Figure 2.3: Software Based Receiver Design

2.3 SDR Realm

Every designer's fantasy is to have a wireless system free of any air interface constraints and future modifications. This realm of SDR is described by Broadband Magazine as a wireless network where "new frequency allocations or new modulation schemes could be adopted at a keystroke. The network could acquire new spectrum as soon as it becomes available or even utilize spectrum from another network operator on a temporary basis. Frequencies could be reused with a high degree of aggressiveness within the same cell, where line-of-sight placement of subscriber terminals becomes essentially irrelevant, and antenna gain can be varied dynamically to adapt to changing network conditions. In such a network, new standards-based protocols governing framing, network restoration and bandwidth reservation could be downloaded network-wide over the air interface with no interruption in service and no manual reconfiguration required on the part of the operator or the subscriber [10]." In such a network wireless transmission and reception would become just another computing function. Although SDR is not in full realizable form, it is emerging as the hottest new technology.

2.3.1 SDR Concept

SDR is a fast developing technology that has accumulated extensive recognition and interest in the telecommunication industry. The concept of "software radio" has been around for some time, having initially been discussed in the field of military research. Now, however, with the increasing capabilities of DSP on one hand, and the requirements for fast time to market on the other, it is emerging as an important commercial technology. Digital radio systems with programmable hardware modules are being used to build an open-architecture based radio system software. Radio applications such as Bluetooth, WLAN, GPS, Radar, WCDMA, GPRS, etc. can be implemented using SDR technology [1].

.

SDR provides an efficient and comparatively inexpensive solution to the problem of building multi-mode, multi-band, multi-functional wireless devices that can be enhanced using software upgrades. It implements via software, functional modules of a radio system such as signal generation, modulation/demodulation, coding and link layer protocols. SDR-enabled

devices and equipment can be dynamically programmed in software to reconfigure the characteristics of equipment. This allows manufacturers to concentrate development efforts on a common hardware platform. Similarly, it permits network operators to differentiate their service offerings without having to support a myriad of handhelds. Also, software modules that implement new features can be downloaded over the air onto the handsets. Finally, SDR provides the user with a single piece of scalable hardware that is compatible at a global scale [11].

2.3.2 Software Defined Radio Definition

The term "software radio" (SR) has various definitions since no consensus has been reached about the level of reconfigurable architecture needed to qualify a radio as a software radio. Joe Mitola, who coined the phrase software radio, would describe software radio as:

"*a radio whose channel modulation waveforms are defined in software. That is, waveforms are generated as sampled digital signals, converted from digital to analog via a wideband DAC and then possibly unconverted from IF to RF. The receiver, similarly, employs a wideband ADC that captures all of the channels of the software radio node. The receiver then extracts, downconverts, and demodulates the channel waveform using software on a general purpose processor [12].*"

The SDR Forum, a non-profit association of different software radio players, describes SDR technology as:

"*radios that provide software control of a variety of modulation techniques, wide-band or narrow-band operation, communications security functions, and waveform requirements of current and evolving standards over a broad frequency range [12].*"

A well-established definition of software radio is

"a radio that is substantially defined in software and whose physical layer behavior can be significantly altered through changes to its software [13]."

In the radio industry, the terms SR and SDR are used to refer to radios exhibiting the above characteristics. The term SDR is commonly used in technical literature and therefore in this thesis.

2.3.3 Evolution of SDR

Software Defined Radio is a promising technology and has gained worldwide interest and support from commercial industries and government agencies. SDR concept started in the late 1970s with the introduction of multimode radios operating in very high frequency (VHF) band. One of the first software radios was a military project named SPEAKeasy. The primary goal of the SPEAKeasy project was to use programmable processing to emulate more than 10 existing military radios, operating in frequency bands between 2 and 200 MHz. Further, another design goal was to be able to easily incorporate new coding and modulation standards in the future, so that military communications can keep pace with advances in coding and modulation techniques [13].

The birth of SDR was a result of the Department of Defense's (DoD's) initiative Joint Tactical Radio System (JTRS). Evolving from SPEAKeasy, JTRS is motivated by the same issues identified by the SPEAKeasy program. DoD's desire to obtain a more flexible approach to achieving diverse communication led to the development of JTRS with digital signal processors and general purpose processors [14]. Hence, the baseline structure for JTRS is software communication architecture (SCA), which has allowed it to shift away from a hardware dependent architecture.

Other developments of SDR include Motorola's Wireless Information Transfer System (WITS) radio, the SDR-3000 produced by Spectrum Signal Processing Inc., the SpectrumWare System, and the CHARIOT (Changeable Advanced Radio for Inter-Operable Telecommunications) software radio developed by Virginia Tech as part of DARPA's GloMo programs. The WITS radio was the first instantiation of the JTRS/SDR Forum architecture. The SDR-3000 was an example of a system fully compliant with the JTRS SCA. The SpectrumWare program justified the use of GPPs in a software radio design. CHARIOT's layered architecture structure created a structure for running reconfigurable hardware into a software radio [13].

As shown in Figure 2.4, SDR and its architecture continue to evolve as new technologies become available. Initially developed as a solution to interoperability problems of the military, SDR has developed well beyond its early role. SDR is now viewed as an enabling platform for a vast array of technologies.

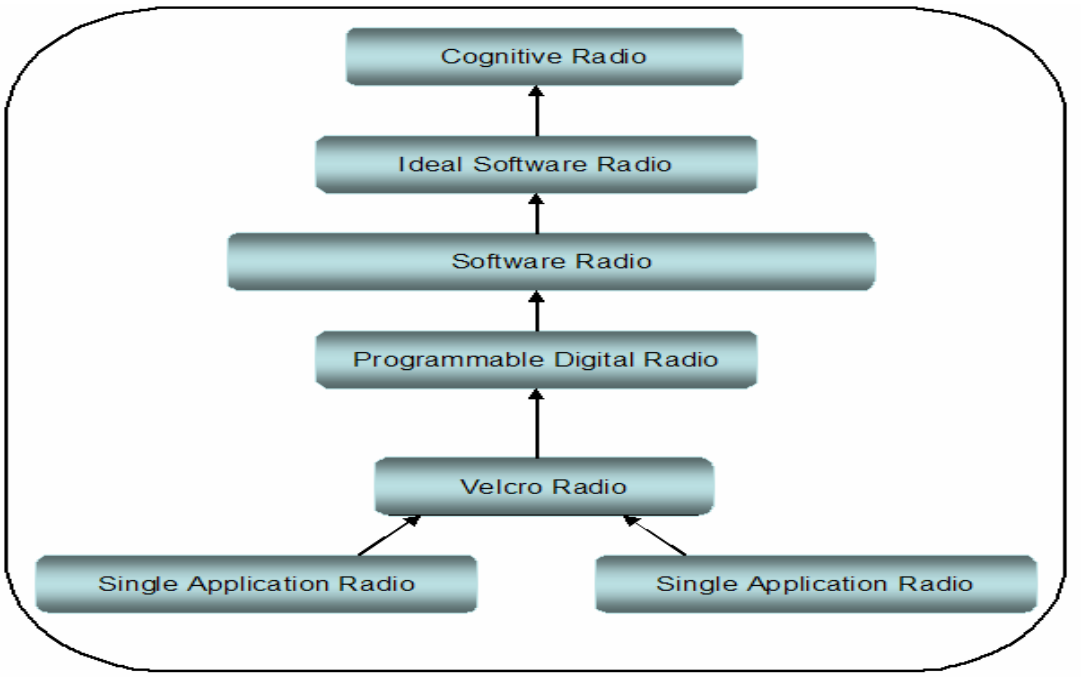

Figure 2.4: Development of Software Radio

2.3.4 Advantages and Benefits of SDR Technology

The multitude of wireless network standards hinders seamless interoperability by requiring different physical devices to inter-operate with different networks. A new challenge to the mobile communication industry is the integration of multiple systems and applications on a single device. Although third generation wireless communication concepts address the goal of global standardization, a more realistic approach in the intermediate term is to develop transceivers that will work with several standards and in several frequency bands on a common hardware platform. Such a platform would allow flexible and programmable transceiver operations. This type of software radio is expected to be a key technology in several application scenarios of wireless communications. The following factors illustrate motivation in advancing SDR technology in the telecommunication industry.

- Multifunctionality: The existence of various technologies increases the incompatibility between devices, requiring users to purchase additional hardware that supports new standards. The reconfigurable capability of SDR allows users to support various standards and services on a single system.
- Global Mobility: Network standards are continuously evolving and differ significantly in link-layer protocol standards causing widespread problems. SDR allows compatibility

between all these standards so users do not face problems during migration of the network from one generation to the next.

- Compactness and Power Efficiency: Multifunction radios requiring separate silicon for each additional system can attribute bulkiness and inefficiency to the device. SDR's reconfigurable attribute can reduce the size and power of the device since the hardware can be reprogrammed to implement various systems.
- Ease of Manufacture: The complexity of standardizing RF components delays the product introduction. Digitalization of the signal can result in the reduction of hardware components and therefore reduce the time to market.
- Ease of Upgrade: The evolution of standards requires the enhancement of current devices. The flexible architecture of SDR allows systems to be upgraded easily and permits new devices to be integrated easily into existing infrastructures [13].

2.3.5 Design Principles

To ensure the benefits of SDR, such as flexibility, reconfiguribilty, and scalability are maintained, its development must allow for interaction between various subsystems of the radio design. Therefore, the SDR architecture design is important. The following steps illustrate the design principles in developing SDRs.

- System Engineering: Understanding communication link and network protocol constraints to ensure allocations of sufficient resources to establish service under system constraints.
- RF Chain Planning: Incorporation of flexibility in selecting power gain, bandwidth, center frequency, sensitivity, and dynamic range is necessary in designing SDRs. Achieving strict flexibility is impossible, and hence tradeoffs must be made.
- Analog to Digital Conversion and Digital to Analog Conversion: ADC and DAC are difficult to achieve and thus requires tradeoffs in power consumption, dynamic range, and bandwidth. Due to the weakness of current conversion technology, post-digitalization techniques can be used to improve flexibility of the digitalization stage.
- Software Architecture Selection: Software architecture should allow for hardware independence through use of appropriate interfaces between software-oriented applications and the hardware layer.

- DSP Hardware Architecture Selection: Since DSP hardware can be implemented through microprocessors, FPGAs, or Application Specific Integrated Circuits (ASICs), selection of hardware will depend on the algorithms and their computational and throughput requirements.
- Radio Validation: It is necessary to ensure that the communication system is operating correctly and that minor glitches do not cause system failure [13].

2.3.6 Future of SDR

SDR technology has many applications. The military wants smart radios that can flexibly work in whatever country they are deployed. Cell phone makers want to consolidate the four or more radios that are building into their handsets and provide bug fixes with downloaded software. And public safety professionals see SDR as a way to solve interagency communications problems during a crisis. "Our history is hardwired technologies and standards, but our future is software-defined systems and standards for reconfigurability," said Mark Cummings, chairman of the SDR Forum. The next step is to work on a standard for fixed-access systems that use so-called cognitive radio techniques to flexibly tap unused swaths of spectrum. Cognitive radio is a paradigm for wireless communication in which either the network or wireless node itself changes particular transmission or reception parameter to fulfill specific tasks. This parameter alteration is based on observations of various factors from external and internal cognitive radio environment, like radio frequency spectrum, user behavior, network state, etc [14].

2.4 Hardware Platform

A transition from traditional analog RF hardware to digital systems has occurred in the field of communications due to SDR development. These systems perform digitization of baseband RF signals and signal processing through use of DSPs. The increase of power available for DSPs in SDR applications has conditioned designers to digitize the RF signal closer to the antenna. This technique has been connected with advancement in the re-programmability of SDR interfaces [15].

2.4.1 GPP, DSP, or FPGA

SDR implementation requires efficient hardware and software architecture. Typically, the architecture will be split across a GPP, DSP, and dedicated hardware (implemented in the FPGA) [17]. Figure 2.5 shows typical functions of SDR supported by each of these devices.

The downfall of DSPs is their inability to handle an immense flow of data inherent in SDR applications. Therefore, FPGAs seem to be the prominent choice since they can be used to offload the GPP or DSP with application-specific hardware acceleration units [18]. The advancement of FPGAs from being just flexible logic design platforms to rapid signal processing engines has had a hand in revolutionizing the SDR market.

FPGAs are well suited for high-speed parallel multiply and accumulate (MAC) instructions. The Cots journal states currently most "FPGAs can perform an 18 x 18 multiplication operation at speeds in excess of 200 MHz", which make FPGAs an ideal candidate "for operations such as Fast Fourier Transform (FFT), Finite Impulse Response (FIR) filters, Digital Down Converters (DDC), Digital Up Converters (DUC), correlators, pulse compression, etc." Although, FPGAs can implement many DSP functionalities they have difficulty implementing floating point operations due to excessive area required in the device [10]. Therefore, such factors will allow for co-existence of FPGAs and DSPs for some time, but with FPGAs assuming increasing importance.

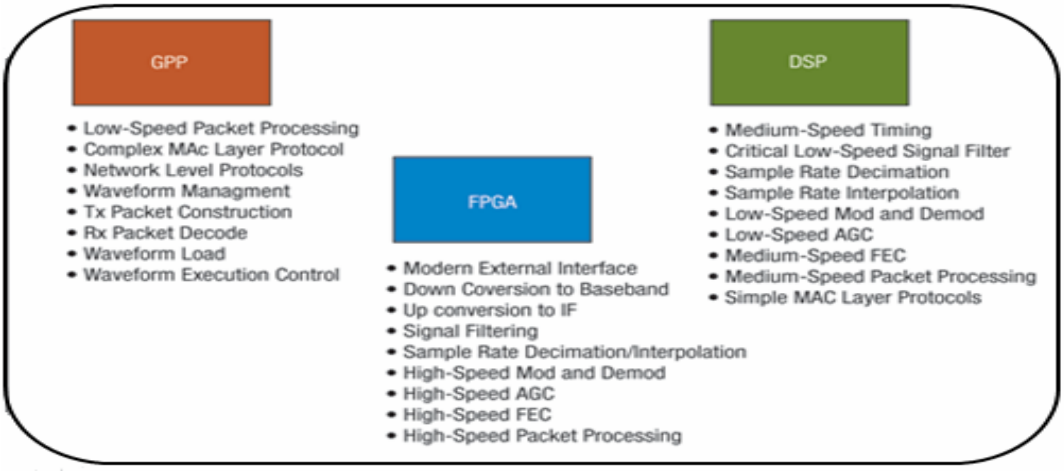

Figure 2.5: Architecture Splitting SDR Functions across GPPs, DSPs, and FPGAs
Source: [10]

2.4.2 FPGAs

Although ASICs and DSP processors are available for SDR implementation, many companies offer a far superior alternative. This alternative choice is implemented through usage of larger FPGAs, efficient DSP algorithms, and automated design process tools [19]. The reprogrammable standard FPGAs offer high performance parallel processing, equivalent to any custom silicon devices. Parallel processing techniques provide an effective way to achieve high sampling rates while increasing algorithmic complexity. When compared to custom chip alternatives, these off-the-shelf devices are cost effective and easier to implement. The FPGAs offer design tools that access high level DSP building blocks (IP cores) to reduce implementation complexity [19].

Compared to DSP processors, the high performance FPGAs are more flexible. FPGAs offer parameterized building blocks to increase adaptability of design implementation for any real world application. The ability to act as co-processors increases the performance of the FPGAs when complex calculations or high sample rate signals must be processed. This allows the DSP processor to concentrate on executing the code portions of the algorithm [19].

Even though DSPs offer highly efficient MACs for digital signal processing, downfalls arise in high frequency applications as a result of being able to handle only few simultaneous calculations. While parallel pipelining increases efficiency, it is not the best way to increase performance. To increase performance of standard DSP processors after a performance limit has been reached, DSP clock speed or number of processors used must be increased. Increase in clock speed is an inefficient solution due to the rise in DSP cost and power consumption [20]. Adding processors is inefficient when comparing the performance gained with the increase in power consumption, board area, cost, development time, and design complexity. The increase in MACs through usage of multiple processors comes at too high a cost. Many companies offer a solution for this problem. They allow millions of MACs to be executed in a single component through the usage of parallel processing. In Figure 2.6, a comparison of Xilinx FPGAs with traditional DSP processors for MAC operations is shown [19]. It is apparent that FPGAs outperform traditional DSPs. As a result, designers use FPGAs to provide widespread MAC functionality.

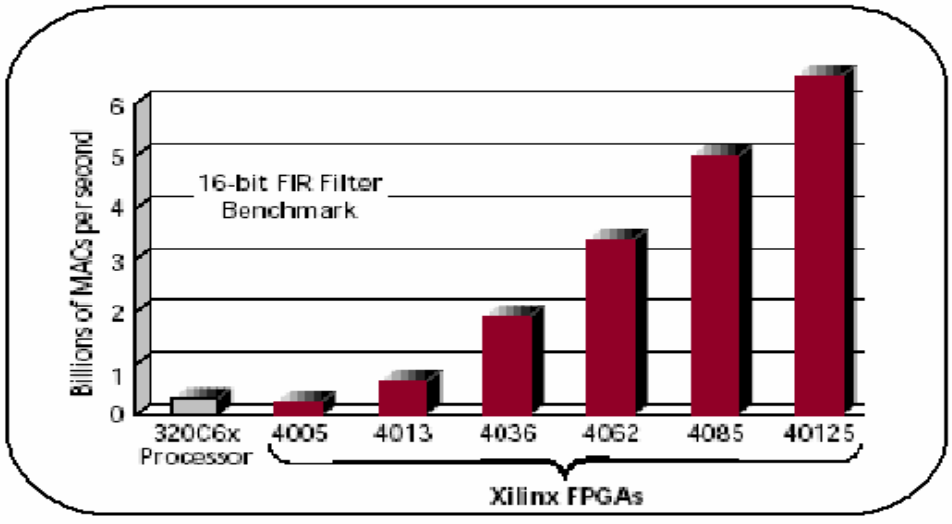

Figure 2.6: Comparison of Performance of Xilinx FPGAs to DSP Processors

Source: [19]

Implementation of FPGAs for high-speed and complex signal processing requirements in SDR applications decrease complexity of high-speed operations such as down conversion, decimation, and interpolation [20]. Usage of Distributed Arithmetic (DA) algorithms matched with the distributed random access memory (RAM) structure allows for efficient placement on FPGAs. Other complex calculation issues are resolved by using look-up tables (LUTs) and adders [19].

The distributed RAM available on the board is useful in buffering data streams, such as in a finite impulse response (FIR) filter. A filter requiring hundreds of taps would require a large number of flip-flops, more than the available amount on most FPGAs. But the distributed RAM on FPGAs allows for development of large shift registers with intermediate taps [19].

Another advantage of using FPGAs is lower power consumption, which is an important aspect of DSP applications. The reduction of size of metal lines used to interconnect programmable logic blocks lowers the power dissipation. Therefore, more MACs are performed before reaching the power limit. Figure 2.7 shows the comparison of power consumption between Xilinx segmented routing architecture and non-segmented routing architecture.

The segmented routing structure allows for the specification of the size and performance of cores before design implementation. The segmented routing architecture also allows for

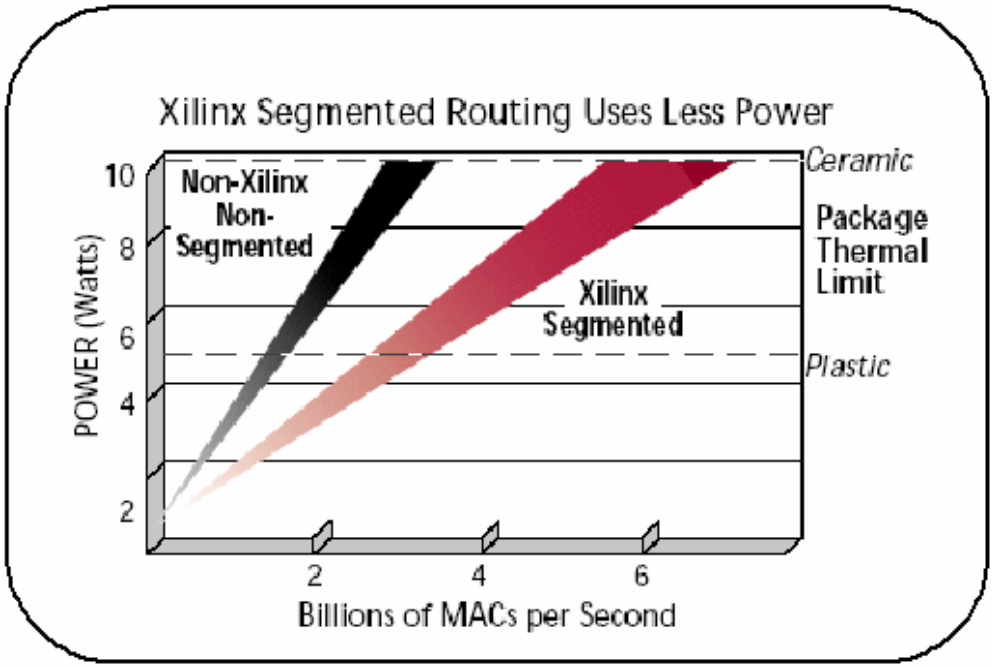

Figure 2.7: Power Consumption of Segmented and Non-Segmented Routing Architecture
Source: [19]

consistency of performance as cores are added to a device. Cores implemented in FPGAs without segmented routing suffer from unpredictable performance degradation as additional cores are added to the device. In addition, the long metal lines in non-segmented FPGAs must get even longer as the device size becomes larger and this results in a 30% reduction in performance between the smallest and the largest device [19].

An additional benefit of using FPGAs is that the CORE Generator and DSP LogiCORE products develop system level DSP functional blocks automatically. The performance characterized cores are selected from a hierarchical library and parameterized to user specifications for usage with standard hardware design environments such as VHDL, Verilog, or schematic capture. The output of CORE generator is a logic netlist and a behavioral model for schematic capture or instantiation code for VHDL or Verilog [19].

FPGAs also come with system design tools that implement, simulate and test the design system. These tools extract hardware description language (HDL) code and form a high-level, block-architectural design. The integration of Xilinx system level tools provides rapid translation

from implementation phase to simulation phase to testing phase and back. This increase in the development cycle can often save a designer 50% or more time for final implementation [17].

2.4.3 Advantages of Using HDLs to Design FPGAs

Hardware Description Languages are used to describe the behavior and structure of system and circuit designs. Usage of HDLs to design FPGAs provides the following advantages:

- Top-Down Approach for Large Projects: Large projects require many designers to work together. The top-down approach to system design, supported by HDLs, allows designers to work independently on separate sections of the code.
- Functional Simulation Early in the Design Flow: By implementing HDL simulation, the functionality of the design can be verified earlier in the design flow. By testing the design before RTL or gate level implementation of design allows necessary changes to be made early in the design process.
- Synthesis of HDL Codes to Gates: Hardware description can be synthesized to gate level implementation of design, eliminating the need to define each gate. This reduces the overall design time and errors that can occur in translation of hardware description to schematic design. Also, efficiency can be increased by applying the automation techniques used by the synthesis tool during the optimization phase of the design to original HDL code.
- Early testing of Various Design Implementations: Different implementations of the design can be tested early in the design process by using HDLs. Since synthesis tools can be used to perform the logic synthesis into gates in a short amount of time, designers can experiment with different architectural possibilities at the Register Transfer Level (RTL).
- Reuse of RTL Code: RTL code can be retargeted to new FPGA architectures with minimum recoding [22].

FPGAs play an important role in implementing designs cost-effectively for real-world applications. With introduction of 3G wireless technology, support for multiple air interfaces and modulation techniques will become a necessity for future communication devices. With enhancement of FPGA technology and intellectual property (IP) cores, SDR is becoming the most optimal solution. Also, SDR's key features such as flexibility and adaptability have enabled it to be a leading contender in the race of providing secure path services as GPRS (General Packet

Radio Service), EDGE (Enhanced Data rates for GSM Evolution), and 3G standards become realities. Through SDSR, development of adaptable high-speed communication equipment can be enhanced. Optimization of designs to meet performance, cost, and power requirements can be met if designers understand the analog signal interactions from the RF front-end to ADC and DSP processing subsystems implemented on FPGAs.

Chapter 3
Theory and System Model

The implementations of spread spectrum (SS) systems have become steadily more important due to their widespread adoption. This field covers the art of secure digital communications that is now being exploited for commercial and industrial purposes. Applications for commercial SS range from wireless LAN's to integrated bar code scanner, computer/radio, and modem devices for warehousing, to digital dispatch, to digital cellular telephone communications, to country-wide networks for passing faxes, computer data, email, or multimedia data.

This chapter presents the theoretical background on the SS and the modulation technique used to create the digital communication system model in software. Therefore, it will provide extensive details on Direct Sequence spread spectrum (DSSS) and Quadrature Phase Shift Keying (QPSK). Due to the architectural complexity of the communication system, simulation is employed for design implementation and verification. Furthermore, this chapter will explain all functional components used to model the transmitter, channel, and receiver. Finally, the modifications made to the design to acquire an optimized design will be elaborated upon.

3.1 Spread Spectrum

The implementation of spread spectrum implies that bandwidth of the transmitted signal is several orders of magnitude greater than the minimum bandwidth, B_{min}, required for transmission. The reasoning behind the increase is to transform a signal with bandwidth B into a noise-like signal of much larger bandwidth B_{ss}. This type of system is inefficient for a single user, but is very bandwidth efficient in a multiple-user, multiple-access interference (MAI) environment because many users can use the same bandwidth simultaneously without interfering significantly with one another [23].

Apart from occupying a large bandwidth, spread spectrum signals are pseudorandom and have noise-like properties. Figure 3.1 shows the basic blocks in a spread spectrum system. The

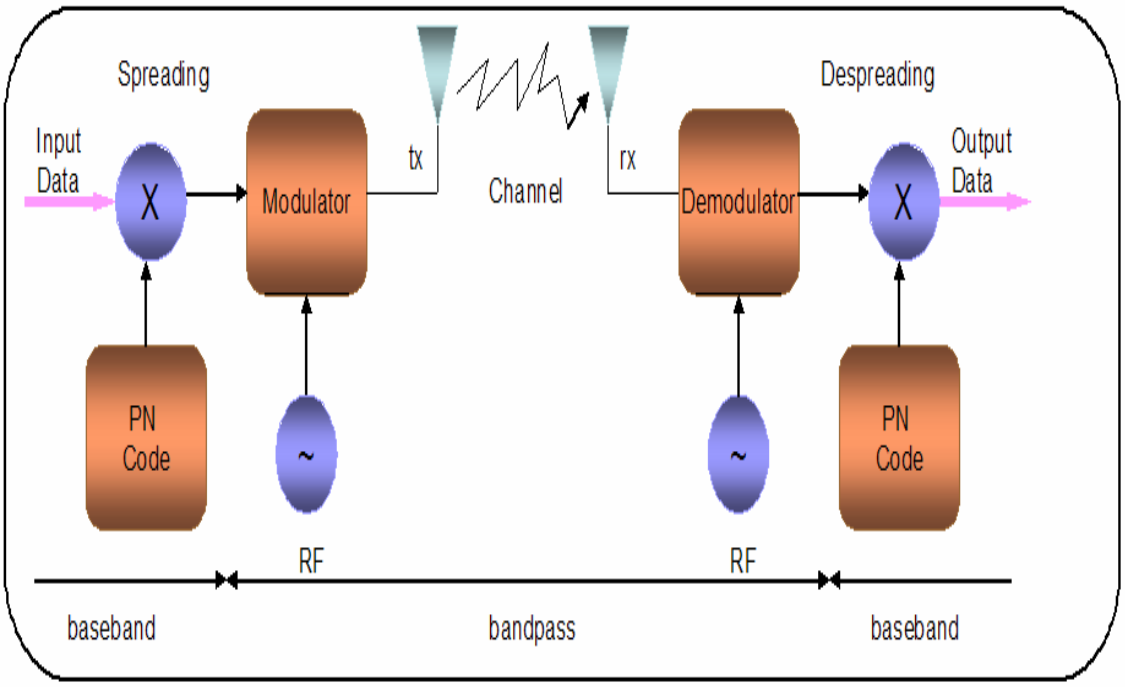

Figure 3.1: Spread Spectrum Model

data is scattered (spread) across the available frequency band in a pseudo random pattern. This spreading of the baseband signal *m(t)* is done by modulating the signal with a pseudo-noise (PN) code sequence *p(t)*. The code sequence *p(t)* is independent of the data sequence *m(t)*. At the receiver, de-spreading of the signal is done by cross-correlating the received signal *r(t)* by a locally generated version of the PN sequence *p(t)* [24].

3.1.1 Advantages of SS

There are many applications of SS due to its numerous benefits, which include antijamming capability, low probability of intercept, multiple access capability, multipath protection, low PSD, and interference limited operation [23].

- Antijamming Capability: SS provides anti-jamming capability due to the unpredictable nature of the carrier signal. Since narrowband interference affects only a small portion of the spectrum, jamming the entire spectrum is extremely difficult. This anti-jamming capability made SS an appealing candidate for military applications.

- Multiple Access Capability: SS systems are used for random and multiple access systems. With SS, users can start their transmission at an arbitrary time without worrying about channel saturation.
- Multipath Protection: SS reduces the effects of multipath, and hence reduces the effects of fading. The multipath resistance properties is due to the fact that delayed versions of the transmitted PN sequence will have poor correlation with the original PN sequence, and thus will appear as another uncorrelated receiver.
- Low PSD: Spreading over a large frequency band reduces power spectral density (PSD), while Gaussian noise level increases. This may result in improved spectral efficiency in some cases.
- Interference Limited Operation: Unlike conventional systems, with SS performance is limited by interference rather than by noise. Transmitter-receiver pairs using independent random carriers can operate in the same bandwidth with minimal co-channel interference.

3.1.2 Types of SS

The many variations of spread spectrum include direct sequence spread spectrum (DSSS), frequency hopping spread spectrum (FHSS), time hopping spread spectrum (THSS), multi-carrier spread spectrum (MCSS), and hybrid forms of spread spectrum. In DSSS, a signal is modulated using a wideband spreading signal (PN sequence). In FHSS, the carrier frequency (f_c) is randomly switched from one band to another during radio transmission according to some specified algorithm. FHSS can be further classified into fast frequency hopping and slow frequency hopping. In THSS, the signal hops within a particular time frame, where only one time slot in a frame is modulated. In MCSS, different carriers are used to transmit the signal. Even though these techniques implement SS in various ways, all of them require signal spreading by means of a code, synchronization between pairs of users, power control to minimize near-far effect, and source and channel coding to optimize performance [23]. The most popular SS techniques are DSSS and FHSS. This thesis will focus on only direct sequence implementation of spread spectrum since that it is the spreading technique employed in the system model.

3.1.3 DSSS

Direct sequence spread spectrum systems are so called because they employ a high-speed code sequence, along with the basic information being sent, to modulate their RF carrier. It can be assumed that the information signal in DSSS transmission is spread at baseband, and the spread signal is then modulated in a second stage. By using this approach the modulation is separate from the spreading and "stretching" operation and the baseband spreading and "stretching" can be discussed separately. At the receiver, the signal is first demodulated and then "stretched" back to recover original information.

A simple example of spreading and "stretching" of DSSS signal is illustrated in Figure 3.2. The data waveform, *m(t)*, is a time sequence of non-overlapping rectangular pulses, each with amplitude of ± 1. Each data symbol represented by *m(t)* has a period of T_s. Its Fourier transform is a *sinc* function with zero values at $1/T_s$. Each pulse in the PN spreading sequence *p(t)* represents a chip with amplitude of ± 1 and period T_c. The transitions of *m(t)* and *p(t)* are such that the ratio of T_s and T_c is an integer. The spreading due to *p(t)* makes the bandwidth B_{ss} of $s_{ss}(t)$ much larger that the bandwidth *B* of a conventionally modulated signal $m(t)\cos(2\pi f_c t)$ [12]. Figure 3.3 illustrates the frequency domain analysis of the DSSS signal described in Figure 3.2.

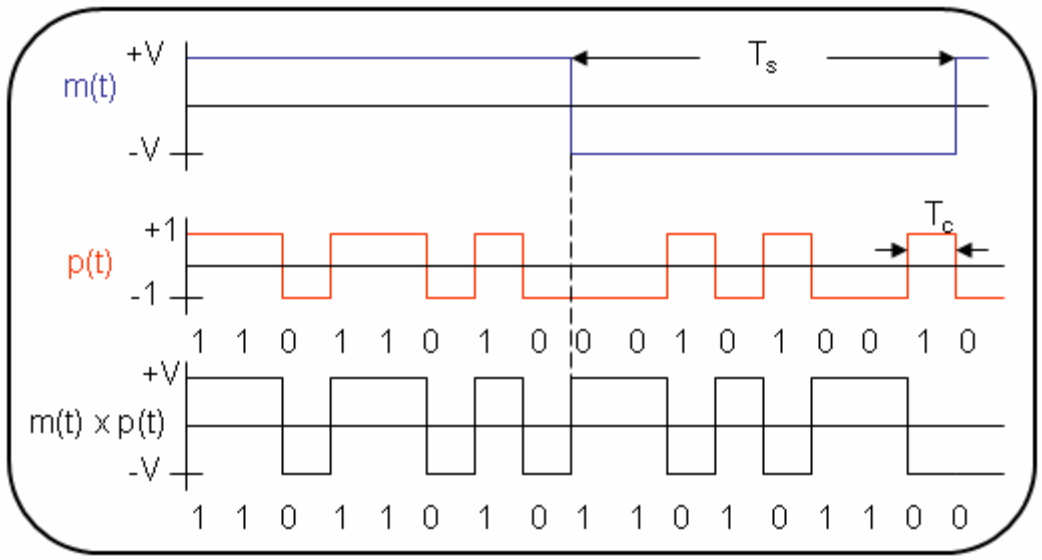

Figure 3.2: Effect of Spreading on Message Stream

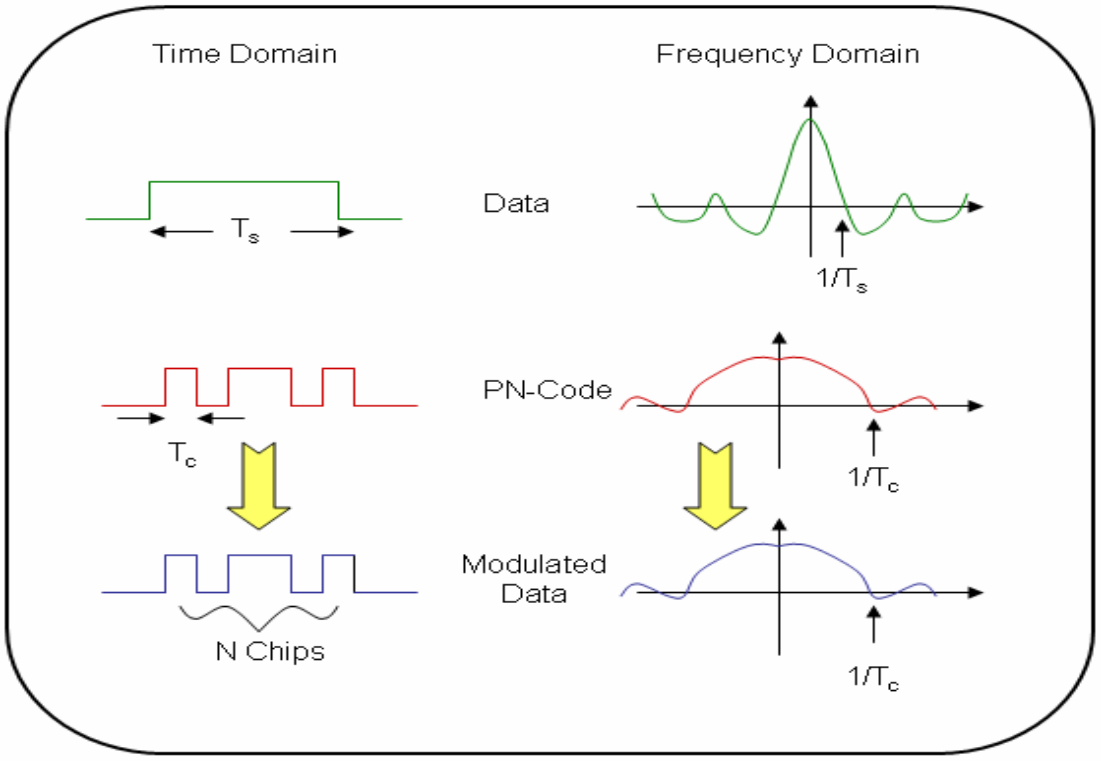

Figure 3.3: Time/Frequency Analysis

Figures 3.4 and 3.5 represent a typical DSSS transmitter and receiver design. The transmitter is composed of a PN code generator, binary adder and balanced modulator. The binary output of the PN generator is added in modulo-2 fashion to the binary message, and the sum is used to modulate a carrier. The result of modulating an RF carrier with such a code sequence is to produce a signal centered at the carrier frequency, direct sequence modulated spread spectrum. Direct sequence spectra vary somewhat in spectral shape depending upon the actual carrier and data modulation used. If a coherent phase shift keying modulation is used in the receiver, the received spread spectrum signal can be represented as

$$r(t) = \sqrt{2P}m(t)p(t)\cos(2\pi f_c t + \theta) + n(t), \qquad P = \frac{E_s}{T_s} \tag{3.1}$$

where *m(t)* is the data sequence, *p(t)* is the PN spreading sequence, f_c is the carrier frequency and θ is the carrier phase angle at t = 0 [23]. SS signals are demodulated at the receiver through cross-correlation with a locally generated version of the pseudorandom carrier. De-spreading of the signal is attained when the signal is cross-correlated with the correct PN sequence. This also

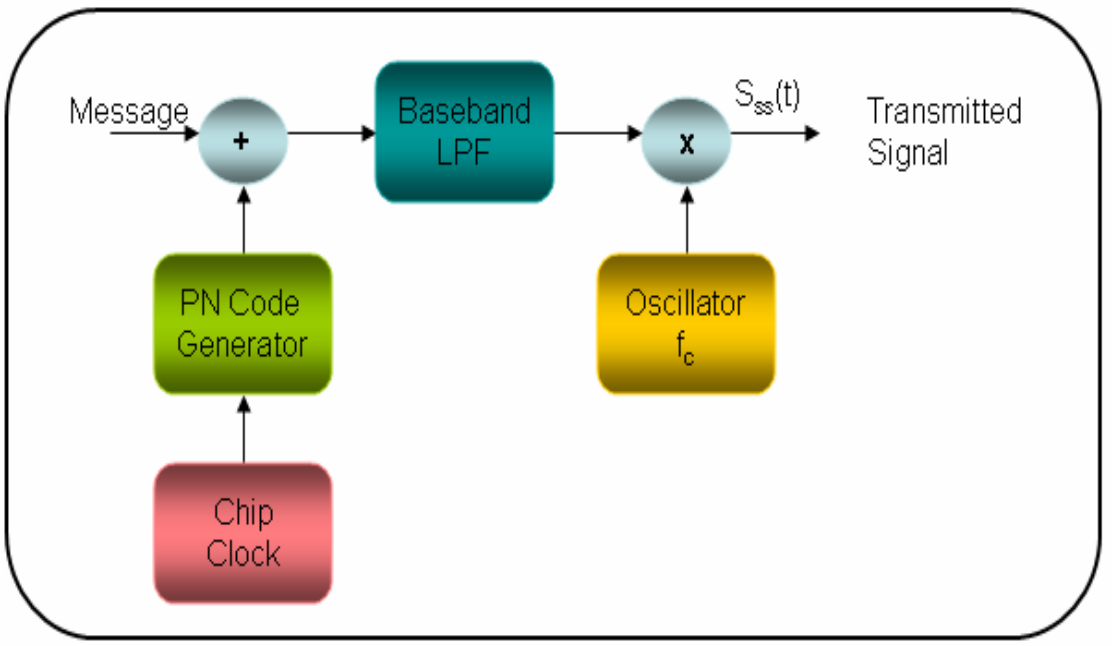

Figure 3.4: Design of Transmitter

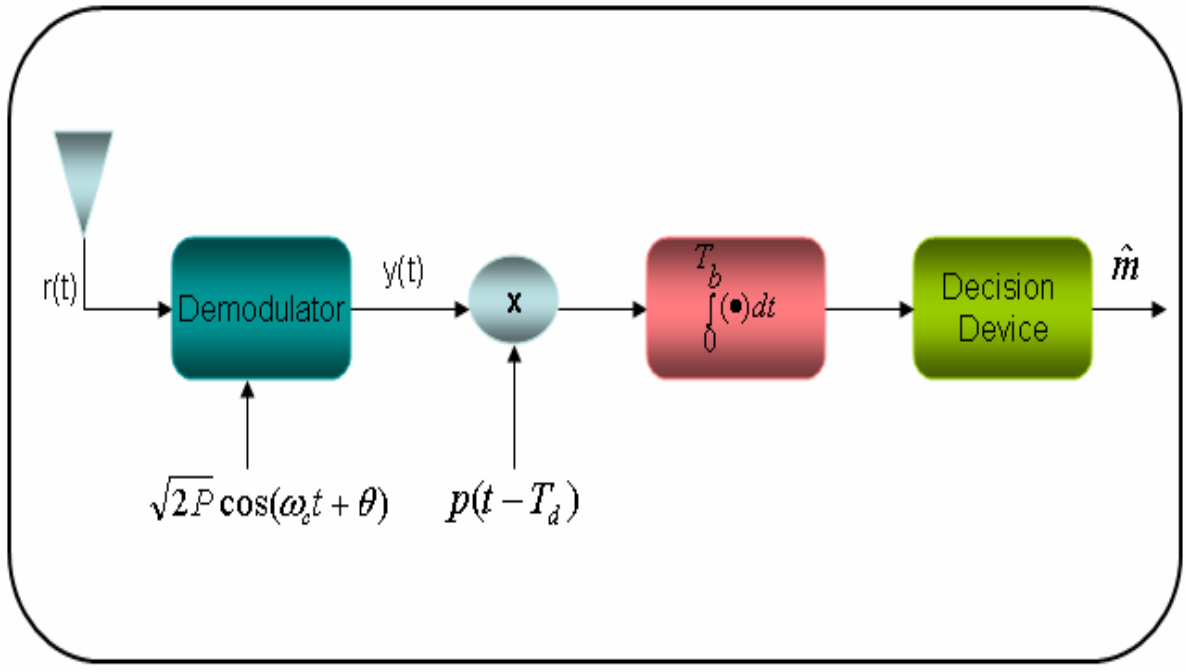

Figure 3.5: Design of Receiver

restores the modulated signal into the same narrow band as the original data [23]. An SS correlator can be thought of as a very special matched filter (MF) that responds only to signals that are encoded with a pseudo noise code that matches its own code. Thus, an SS correlator can be "tuned" to different codes simply by changing its local code.

After demodulation, the signal bandwidth is reduced to B, while the interfering energy is spread over an RF bandwidth exceeding B_{ss}. Thus, most of the original interference energy is eliminated by the spreading and minimally affects the desired receiver signal. The measure of interference rejection capability is defined as the processing gain (PG) or spreading factor, given by

$$PG = \frac{Spread\ \ Bandwidth}{Information\ Bandwidth} = \frac{B_{ss}}{B}, \tag{3.2}$$

which is equivalent to

$$\frac{T_s}{T_c} = \frac{R_c}{R_s} = \frac{B_{ss}}{2R_s}. \tag{3. 3}$$

The greater the processor gain of the system, the greater will be its ability to suppress in-band interference.

3.1.4 PN Generator

Pseudonoise code generators are periodic since the produced sequence repeats itself after a certain period of time. A PN code generator, shown in the Figure 3.6, is generated using sequential logic circuits. In this feedback circuit, binary sequences are shifted through the shift registers in response to clock pulses. The output, which is dependent to the logical combination of the various stages, is fed back as the input to the first stage. Ideally, the spreading code should be designed so that the chip amplitudes are statistically independent of one another. The entire period of PN sequence consists of N time chips. In case of maximal linear PN generator, the value on N is $2^n - 1$, where n is the number of stages in the code generator. Another important reason for using PN generator to modulate a signal is the properties of the resulting signal's autocorrelation function. As illustrated in Figure 3.7, it has a maximal value of one repeating itself every period, and a constant value of $-(1/N)$ in between the peaks [24].

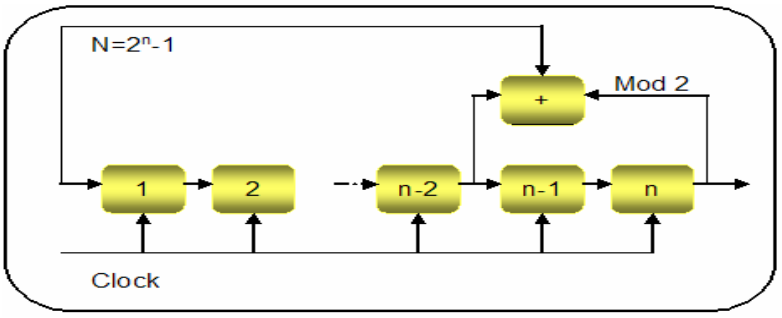

Figure 3.6: PN Generator Model

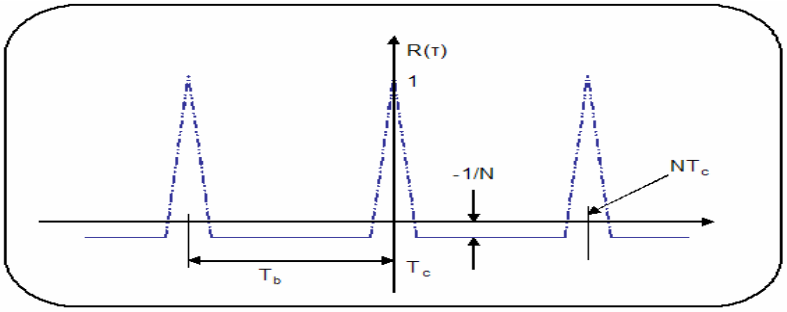

Figure 3.7: Autocorrelation of PN Code

Although PN sequences are deterministic, they have similar properties as random binary sequences, such as equal number of ones as zeros, low correlation between shifted versions of the PN sequence, low cross-correlation between any two sequences. Different classes of periodic PN sequences exist. They include Maximal-Length Linear Shift register Sequences (m Sequences), Quadratic Residue Sequences (q-r Sequences), Hall Sequences, and Twin Primes.

3.2 Quadrature Phase Shift Keying (QPSK)

In an M-ary system, one of *M* possible signals may be transmitted during each *T*-second period, where $M \geq 2$. Each possible transmitted signal of an M-ary message sequence is referred as a symbol. The rate at which M-ary symbols are transmitted through the channel is called the baud rate. Therefore, $M = 4$ is termed as quadrature phase shift keying (QPSK).

Quadrature phase-shift keying (QPSK) is one of the prevalent modulation scheme used in digital communication systems. QPSK is a method for transmitting digital information across an analog channel. Data bits are grouped into pairs, and each pair is represented by a particular waveform, called a symbol, to be sent across the channel after modulating the carrier. The QPSK

transmitter system uses both the sine and cosine at the carrier frequency to transmit two separate message signals, $s_I[n]$ and $s_Q[n]$, referred to as the in-phase and quadrature signals.

As represented by Figure 3.8, the phase of the carrier will take on one of four values: 0, $\pi/2$, π, and $3\pi/2$, where each phase value corresponds to a unique pair of message bits. Considering this set of symbol states, the QPSK signal can be defined as

$$s_{QPSK}(t) = \sqrt{\frac{2E_s}{T_s}} \cos[2\pi f_c t + (i-1)\frac{\pi}{2}] \qquad 0 \le t \le T_{s,} \quad i = 1,2,3,4, \qquad (3.4)$$

where T_s represents the symbol duration and has a value of twice the bit period. Using trigonometric identities, the equivalent form of equation 3.3 is

$$s_{QPSK}(t) = \sqrt{\frac{2E_s}{T_s}} \cos[(i-1)\frac{\pi}{2}]\cos(2\pi f_c t) - \sqrt{\frac{2E_s}{T_s}} \sin[(i-1)\frac{\pi}{2}]\sin(2\pi f_c t). \qquad (3.5)$$

For the QPSK signal set, if the basis vectors are defined over the interval $0 \le t \le T_s$ as

$$\phi_1(t) = \sqrt{\frac{2}{T_s}} \cos(2\pi f_c t), \quad \phi_2(t) = \sqrt{\frac{2}{T_s}} \sin(2\pi f_c t), \qquad (3.6)$$

then the four signals in the set can be expressed in terms of basis signals as

$$s_{QPSK}(t) = \sqrt{E_s} \cos[(i-1)\frac{\pi}{2}]\phi_1(t) - \sqrt{E_s} \sin[(i-1)\frac{\pi}{2}]\phi_2(t) \qquad i = 1,2,3,4. \qquad (3.7)$$

According to the constellation diagram of a QPSK signal (Figure 3.8), the minimum distance between adjacent points is $\sqrt{2E_s}$. Since each symbol corresponds to two bits, then

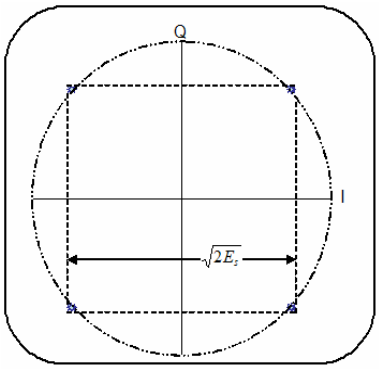

Figure 3.8: QPSK Constellation Diagram

$E_s = 2E_b$. Consequently, the distance between two neighboring points is $2\sqrt{E_b}$. Therefore, the probability of bit error in an additive white Gaussian noise (AWGN) channel is

$$P_{e,Qpsk} = Q\left(\sqrt{\frac{2E_b}{N_o}}\right). \tag{3.8}$$

Since the bit error probability of QPSK is equivalent to the bit error probability of BPSK, twice as much data can be sent in the same bandwidth. Hence compared to BPSK, QPSK provides twice the spectral efficiency with exactly the same energy efficiency [23].

3.3 System Model

The overall system design used to model the communication system is illustrated in Figure 3.9. The system is comprised of the following blocks: transmitter, channel, receiver, bit counter, error counter, and start/stop simulation. The transmitted signal is sent through a noisy channel before it is demodulated in the receiver. The error counter compares the transmitted and received signal and accumulates errors. The Bit Error Rate (BER) is evaluated according to the number of bits transmitted, which is controlled by the bit counter. The start/stop simulation block is created for rerunning simulations to obtain a good average for BER results. The focus of this thesis is the hardware implementation of the DSSS transceiver modeled in software. Therefore, the design is modeled with a software package called System Generator, which is explained in detail in Chapter 4.

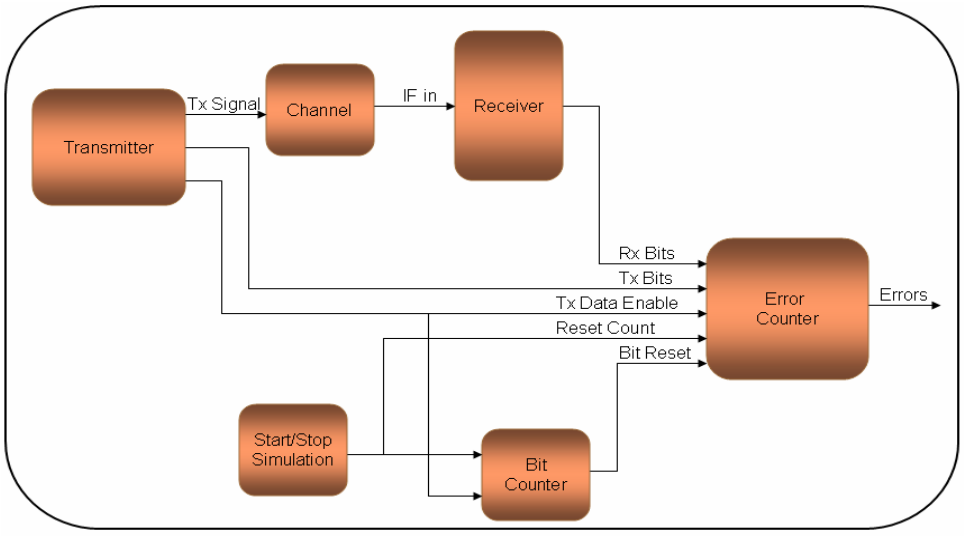

Figure 3.9: Overview of System Model

3.4 QPSK Transmitter

Figure 3.10 shows the block diagram of the QPSK transmitter employed in this system model. The transmitter consists of a scheduler that constructs the data into a packet structure. The details of the packet structure are discussed shortly. The random binary message stream embedded within a packet structure with bit rate R_b is split into two bit streams m_I and m_Q, each with bit rate $R_b / 2$. The m_I stream represents the in-phase component and is referred to as the "even" stream while the m_Q stream represents the quadrature component and is called the "odd" stream. Next, the m_I and m_Q data streams are individually spread by a PN sequence to significantly increase the bandwidth of the transmitted signal. The two bit streams are separately pulse shaped before being modulated with a carrier. Finally, the two modulated signals, each of which can be considered to be a BPSK signal, are summed to form a QPSK signal. The signal is scaled before being sent into a channel composed of additive white Gaussian noise (AWGN).

3.4.1 Creating Packets

In a packet-based communication system, data must be transmitted as a packet. A packet is a self-contained parcel of bytes that is part of a larger block of data that travels as a sequence of bits from a transmitter to a receiver. Sending data in packets offers a mechanism of coordination between sender and receiver and provides a guarantee of fairness, which is very important in obtaining accurate BER results [25]. Usually, a packet consists of three parts: header, payload, and trailer. The header contains instructions about the data carried by the packet, which may include length of packet, synchronization bits, packet number, destination address or originating address. The payload, also referred to as data or body, is comprised of the actual data the packet is sending. If the data is a fixed-length, then the payload may be padded with blank information to obtain the correct size. This is known as data stuffing. The trailer typically contains a few bits that inform the receiver that the packet has ended. It also may include some type of error checking, such as Cyclic Redundancy Check (CRC) [25]. In this design, the packet is simplified to include a header comprised of preamble and Data Delimiter (DDW) bits followed by the actual data sequence. As shown in Figure 3.11, the 512-bit packet contains 65 bits of preamble, 63 bits of DDW, and 384 bits of random data. The preamble, consisting of all ones, is used to define the

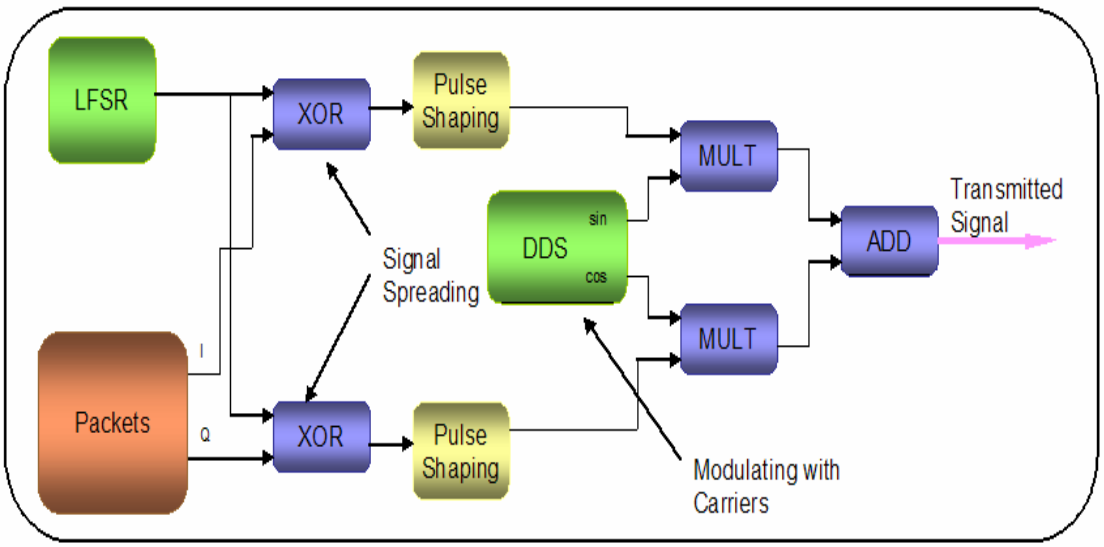

Figure 3.10: QPSK Transmitter

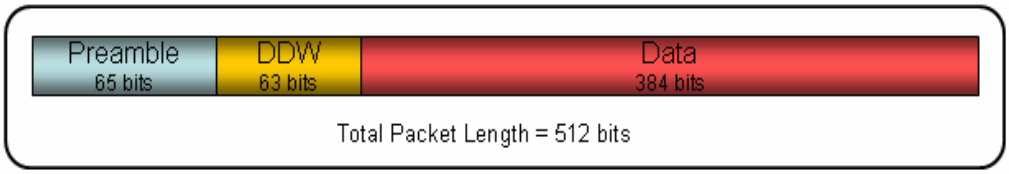

Figure 3.11: Packet Structure

start of a transmit packet. Also, it allows time for lock of the receiver phase lock loop, which is used to synchronize the receiver data clock to the transmitter data clock. The DDW is used for packet synchronization purposes and defines the start of the actual data. When the receiver starts receiving data, it may have an arbitrary phase for its lock clock. During the course of the preamble it learns the correct phase, but may miss or gain a number of bits. Therefore, a special pattern is constructed in the DDW to mark the start of the data. As shown in Figure 3.12, a scheduler is constructed to create the packets as described in the paragraphs above. The scheduler is comprised of a counter, relational blocks, logic blocks, and constant blocks. In addition, the scheduler creates a *select signa*l and *enable signals* for the DDW and data.

The DDW is created by using a Linear Feedback Shift Register (LFSR) that contains a 63-length PN code. The LFSR block is enabled only for the duration of the DDW to ensure that each packet contains the same sequence to represent the DDW. The random data are created by a

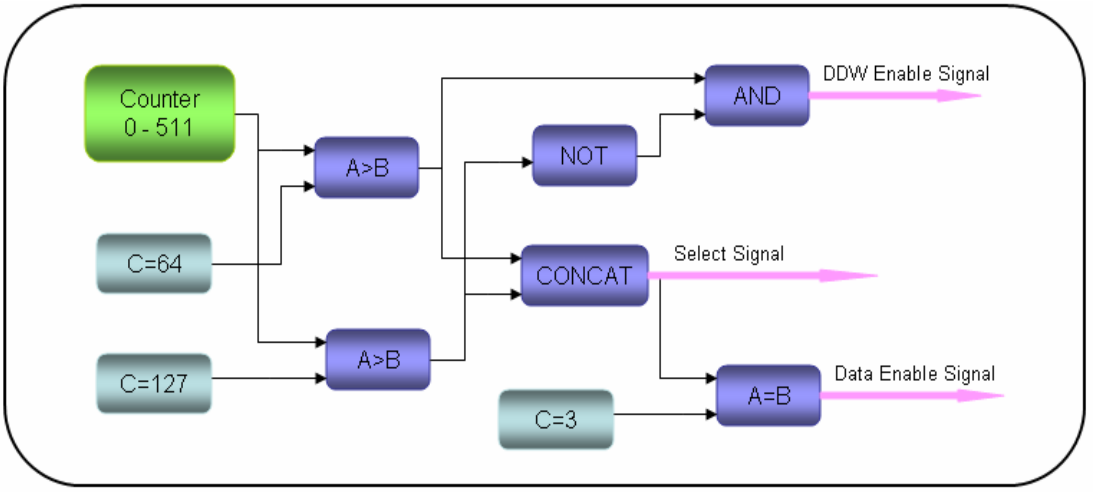

Figure 3.12: Packet Scheduler

combination of blocks: a counter, a parallel to serial, and slice blocks. The counter is designed to output eight bits. The parallel-to-serial block takes these eight bits as input and outputs two bits at a time. Finally, slice blocks are used to send one of the two bits into the I channel, and the other into the Q channel. The m_I and m_Q data streams represent the in-phase and quadrature components needed for QPSK modulation as explained above.

The *select signal,* generated by the scheduler, informs the multiplexers whether to send the preamble, DDW, or data bits. As shown in Figure 3.13, when *select signal* equals zero or one, the preamble is transmitted. When it equals two, the DDW are transmitted. When it equals three, the data are transmitted. As a result, both m_I and m_Q data streams packets will be equivalent for the duration of the preamble and the DDW, but will differ for the data portions.

The m_I and m_Q data streams are processed at a specific sample rate, or clock period, as they flow through the dataflow system. Typically, each block detects the input sample rate and produces the correct sample rate on its output. Rather than using the default sampling period, an explicit sample period of 63 is selected to create the packets. This implies that each bit within the packet will have a period of 63 and therefore each packet will have duration of 32256 $(63 * 512)$. The significance of this value will be explained shortly. Figure 3.14 shows both the m_I and m_Q packets in simulation for duration of one packet length. The simulation result validates that both the m_I and m_Q packets are equivalent until the random data start. It also shows that both packets are sent consecutively and no delay is added between transmission of packets.

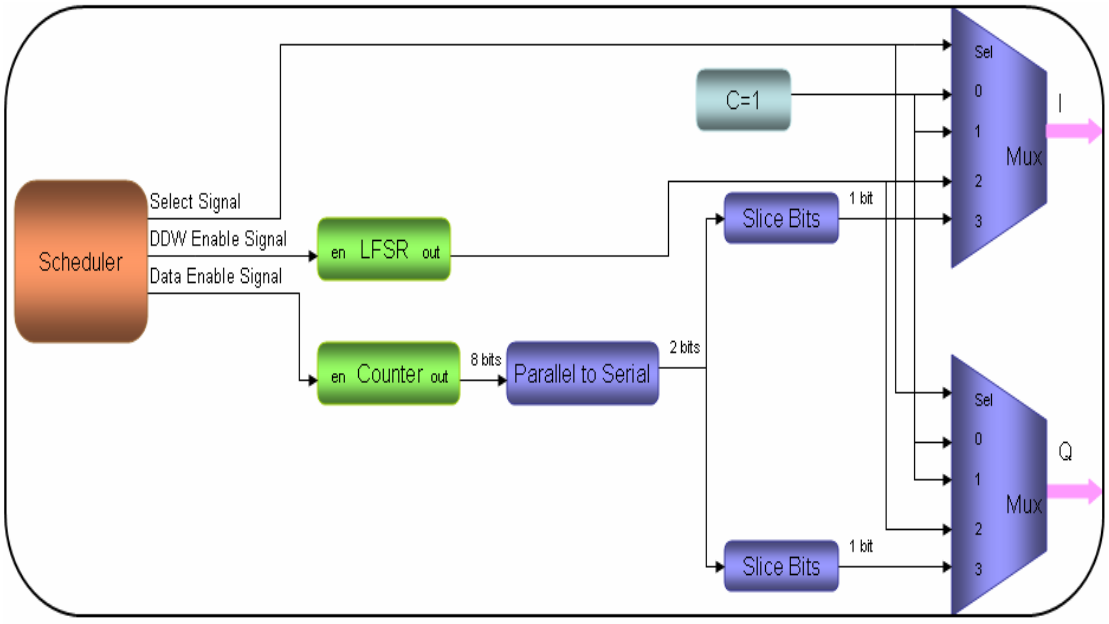

Figure 3.13: I and Q Packets

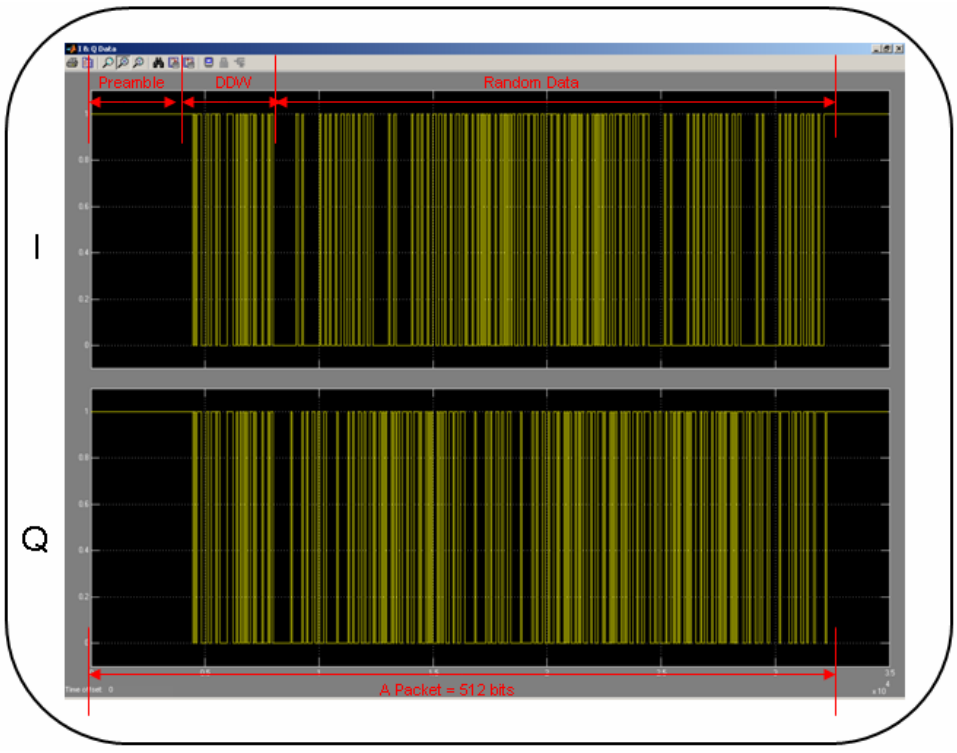

Figure 3.14: I and Q Packets in Simulation

3.4.2 Signal Spreading

To implement DSSS in the system, the incoming packets must be spread by a PN sequence. Figure 3.15 illustrates that before spreading the signal with a PN code, both m_I and m_Q data streams must be up sampled by 63. The up sampling is needed to ensure that the local PN code runs at much higher rate than the data rate. The up sampling block is modified to over-sample the input signal by placing every *n*th input sample at the output instead of presenting it once with (*n-1*) zeros inserted interspersed [6]. Sampling period and up sampling are explained in further detail in chapter 4. The value of 63 is chosen since the length of the spreading sequence generated by the Linear Feedback Shift Register (LFSR) is 63.

Since the sampling period of the incoming packets is 63 at the output of the packet generator, the up sampling increases the sample rate of the packets from 63 to 1 ($63/63=1$). This is done to match the sampling time of the incoming m_I and m_Q data streams to the sampling period of the LFSR used to create the PN sequence. Other features of the LFSR block are discussed shortly. The up sampling of the packets and matching of the sampling rates is of utmost importance in the system model. Without the up sampling, the packets will not be spread by the PN code and without the matching of the sampling rates, the design will not simulate. The addition in modulo–2 fashion of the data symbols to the chips is performed by the XOR block. Finally, the packets are sent to be pulse shaped individually after spreading is completed.

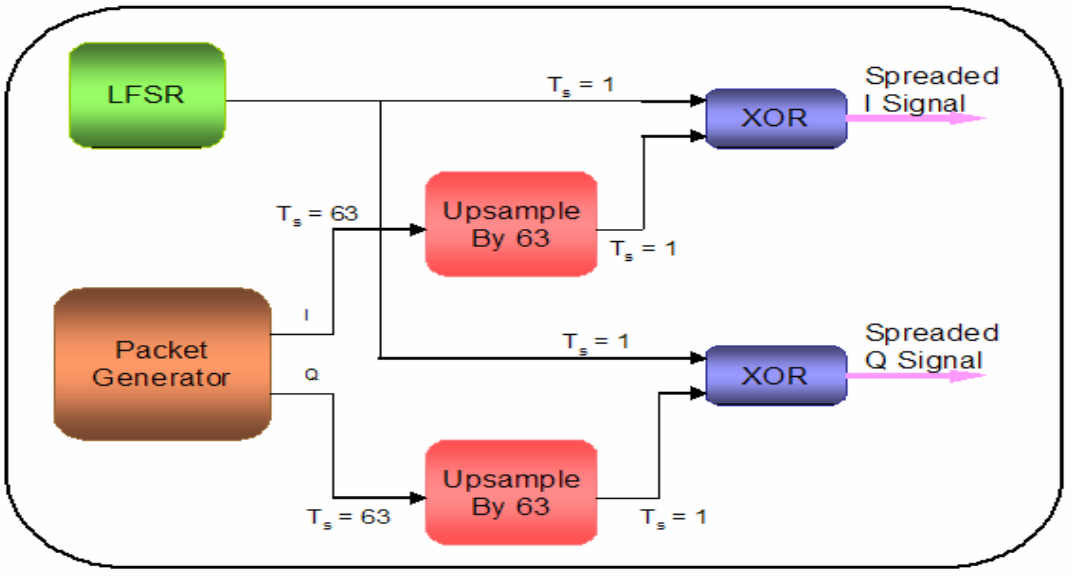

Figure 3.15: Spreading Packets with PN Sequence

To implement signal spreading, a PN sequence must be generated. The PN generator polynomial governs all the characteristics of the generator. Therefore, to implement a particular PN code, correct initialization of the block parameters is essential. For a given generator polynomial, there are two ways of implementing LFSR. Galois feedback generator uses only the output bit to add (in Galois field) several stages of the shift register and is desirable for high speed hardware implementation as well as software implementation. The other known as Fibonacci feedback generator can generate several delays of sequences without any additional logic [26].

A PN generator produces a sequence of bits that appears random. The sequence will repeat with period $2^B - 1$, where B is the width in bits of the shift register. Therefore, only 6 bits are necessary to represent a 63-length PN code. Figure 3.16 illustrates that to create a 63-length PN sequence, bits 4 and 5 of the shift-register are XORed together and the result is shifted into the highest bit of the register. The lowest bit, which is shifted out, is the output of the PN generator [26]. To create this sequence in a LFSR, the LFSR block parameters must be defined as shown is Figure 3.17.

The initial contents of the memory stages and the feedback logic determine the successive memory contents. If a linear shift register falls into zero state, it will always remain in that state, and the output would subsequently be all zeros. Therefore, the LFSR is set to be a 6-bit Fibonacci feedback generator with a hex value of '27' for the feedback polynomial. This value is computed by looking up the feedback polynomial for a 63 length PN sequence and converting it to a hex value according to the specifications of the LFSR block in System Generator. In general the PN sequence has $\frac{2N}{2}$ binary ones and $\frac{2N}{2} - 1$ binary zeros, where N is number of binary stages. Thus, the resultant PN sequence is
111111010111000110011101100000111100100101010011010000100010110.

Since the sample period of the LFSR is 1, each bit of the packets will be spread by the 63 length PN sequence by directly multiplying it with the baseband data pulses. As a result, if a bit is zero it will have the shape of the PN code and if it is one, it will have the inverted shape. Figure 3.18 shows the I and Q packets in simulation after spreading for the duration of two bits. Each bit still has a Simulink period of 63 since it is spread by a 63-length PN sequence that has a sampling period of one.

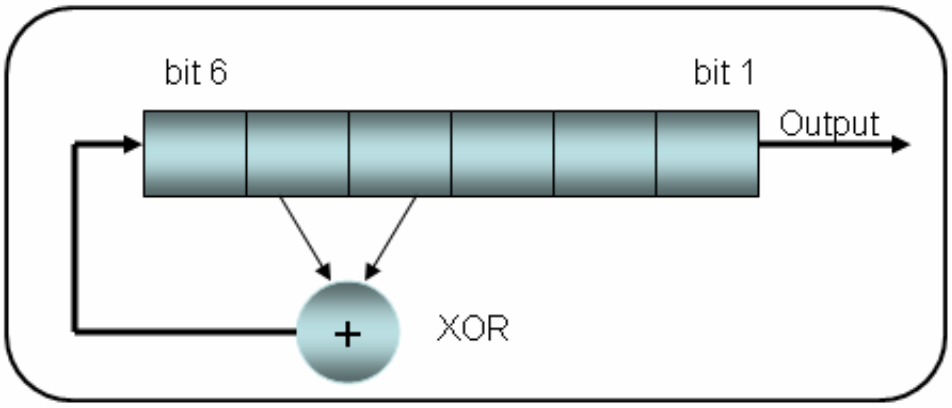

Figure 3.16: 63-Length PN Generator

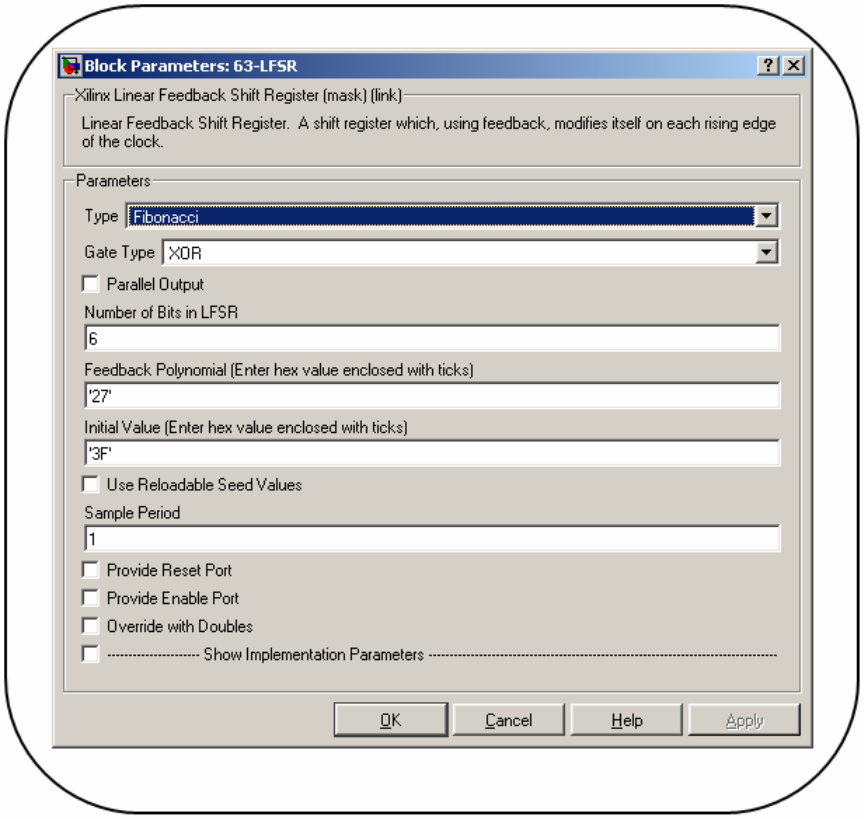

Figure 3.17: Block Parameters for LFSR

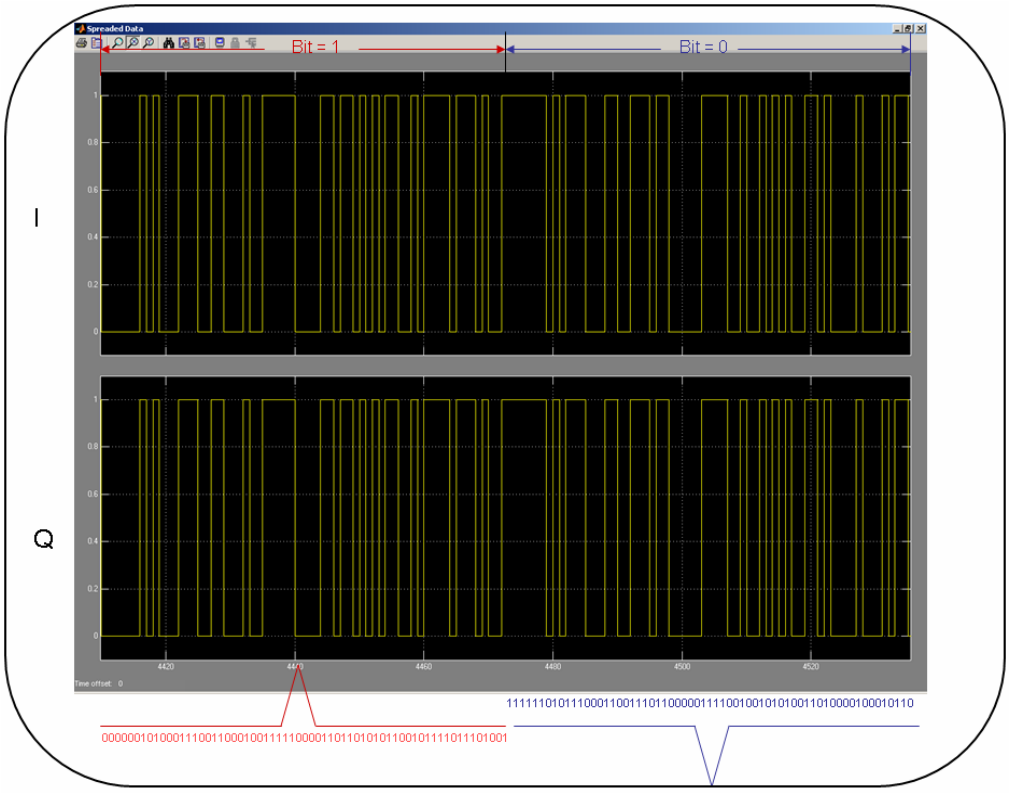

Figure 3.18: Data after Spreading in Simulation (for 2 bits)

3.4.3 Pulse Shaping

In general, the MPSK (M'ary phase shift keying) spectrum consists of a main lobe representing the middle of the spectrum and various side lobes located on either side of the main lobe. Shaping the spectrum should satisfy two criteria: The main lobe should be as narrow as possible, and the maximum side lobe level should be as small as possible relative to the main lobe [27]. Therefore, pulse shaping is used to improve spectral efficiency.

In a bandlimited channel, the rectangular pulses that represent the symbols will spread in time into the succeeding symbols causing intersymbol interference (ISI) and increasing the error probability of the receiver during symbol detection. One way of combating ISI is to increase channel bandwidth. Due to the difficulty of manipulating signals at RF frequencies, spectral shaping is implemented through baseband or IF processing [23]. A number of pulse shaping techniques can be used to reduce ISI such as Gaussian, Nyquist, and Raised Cosine.

Before the data streams are pulse shaped, they are both up sampled by ten to obtain ten samples per chip. Up sampling is employed to minimize spectral re-growth. At this time the Simulink system period will be 1/10. The spread data is shaped with an intermittent and jitter free (IJF) function, which is implemented through ROM blocks as shown in Figure 3.19. The ROM blocks are addressed by the counter and defined to have a depth equivalent to the length of the IJF shaping function. Correct addressing is necessary because if the depth of the ROM is longer than the vector length, the ROM's trailing words are set to zero. If the vector length is longer that the ROM depth, the vector's trailing elements are discarded [22].

The IJF pulse shaping technique is chosen because it is highly bandwidth efficient and easy to implement in System Generator. The IJF – QPSK scheme, also known as FQPSK-1, is based on defining odd and even functions, $s_o(t)$ and $s_e(t)$, over the symbol interval $-T_s/2 < t < T_s/2$ and using their negatives as a 4-ary signal set for transmission [27]. If d_{In} denotes the I channel data symbols, then the transmitted waveform, $x_I(t)$, would be determined as follows:

$$\left.\begin{array}{ll} x_I(t) = s_e(t - nT_s) = s_0(t - nT_s), & \text{if } d_{I,n-1} = 1,\ d_{I,n} = 1 \\ x_I(t) = -s_e(t - nT_s) = s_1(t - nT_s), & \text{if } d_{I,n-1} = -1,\ d_{I,n} = -1 \\ x_I(t) = s_o(t - nT_s) = s_2(t - nT_s), & \text{if } d_{I,n-1} = -1,\ d_{I,n} = 1 \\ x_I(t) = -s_o(t - nT_s) = s_3(t - nT_s), & \text{if } d_{I,n-1} = 1,\ d_{I,n} = -1. \end{array}\right\} \quad (3.9)$$

The Q channel waveform, $x_Q(t)$, is generated using the Q channel data symbols, d_{Qn}, by the same mapping scheme as used in Equation 3.9 and then delaying the resulting waveform by a half-symbol [3]. Thus, s_o and s_e are defined as

$$\left.\begin{array}{ll} s_o = \sin\left(\dfrac{\pi * t}{T_s}\right), & \dfrac{-T_s}{2} \le t \le \dfrac{T_s}{2} \\ s_e = 1, & \dfrac{-T_s}{2} \le t \le \dfrac{T_s}{2}\ . \end{array}\right\} \quad (3.10)$$

The above equations result in the pulse shape shown in Figure 3.20. The simulation results of applying this shaping function to the rectangular pulses is demonstrated in Figure 3.21.

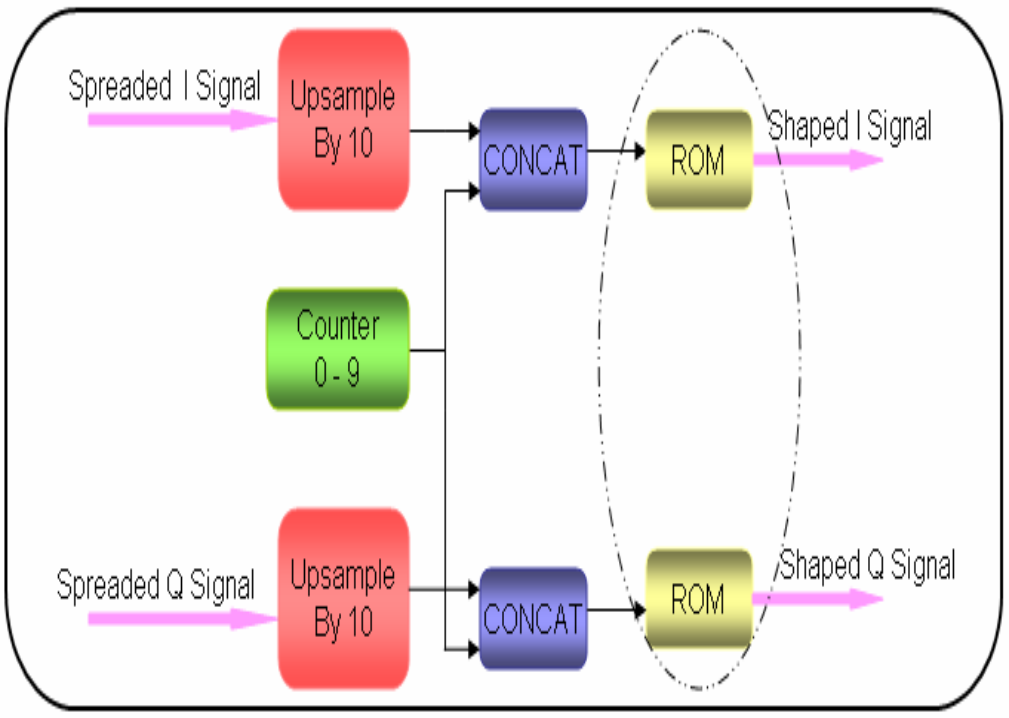

Figure 3.19: Pulse Shaping Model

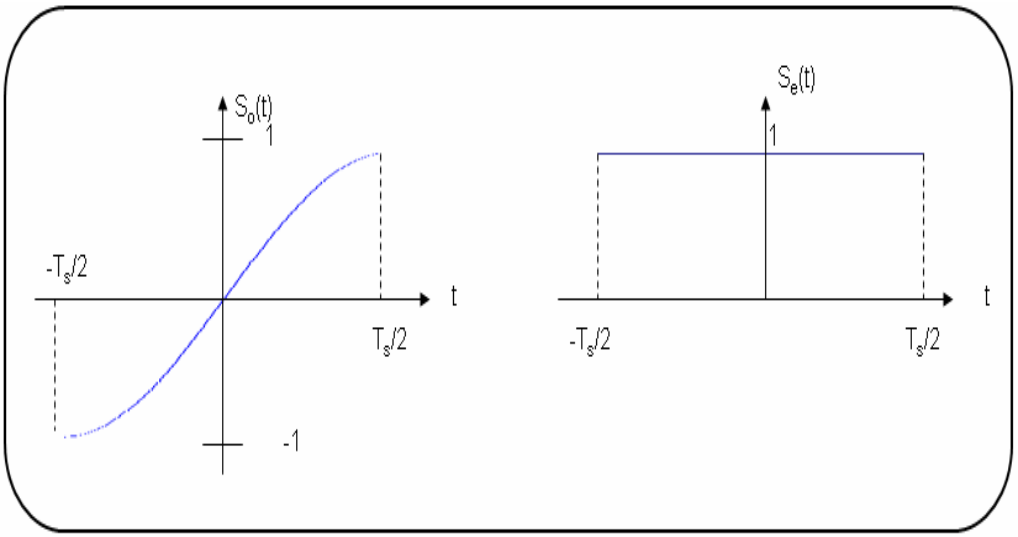

Figure 3.20: S_o and S_e Pulse Shape

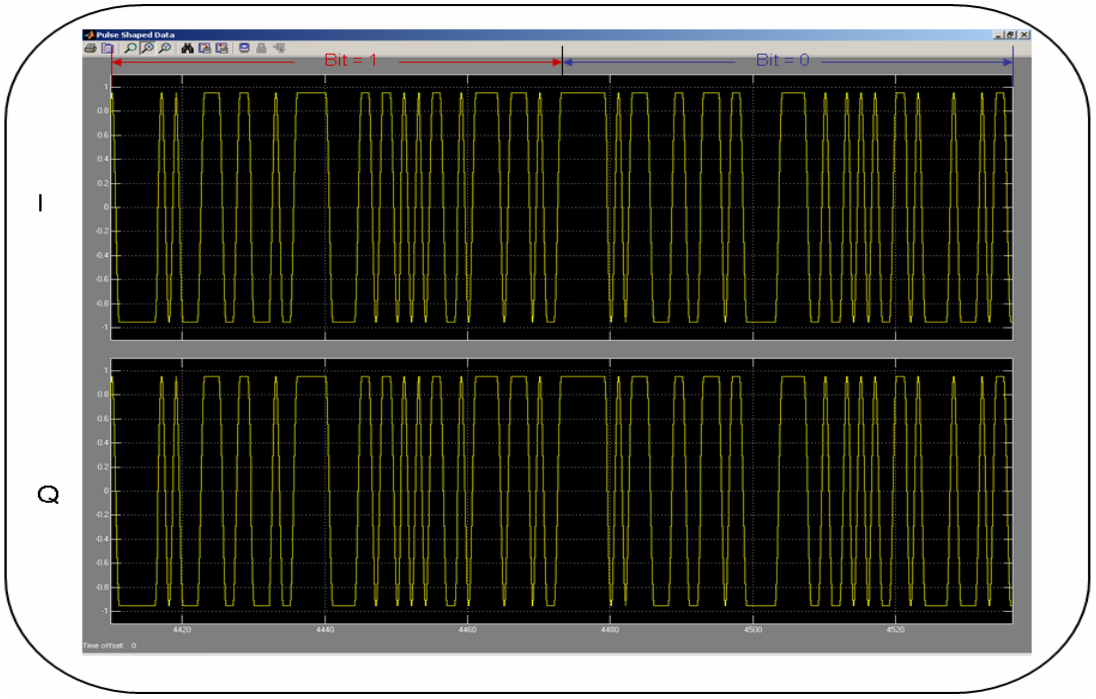

Figure 3.21: Signal Pulse Shaping in Simulation

3.4.4 Modulation

Modulation is the process by which symbols are transformed into waveforms that are compatible with the characteristics of the channel. One of the three key characteristics of a signal is usually modulated: its phase, frequency, or amplitude. Modulation can be used to minimize the effects of interference. Modulation can also be used to place a signal in a frequency band where design requirements, such as filtering and amplification, can be easily met. This is the case when radio-frequency (RF) signals are converted to an intermediate frequency (IF) in a receiver [23].

A phase shift keying (PSK) modulation scheme is employed in this system. When $M = 4$ in an M-ary phase shift keying (MPSK) modulation scheme, it is defined as Quadrature phase shift keying (QPSK). A phase-modulated waveform can be generated by using the digital data to change the phase of a signal while its frequency and amplitude stay constant. The term "quadrature" implies that there are four possible phases (4-PSK) which the carrier can have at a given time corresponding to one of {0, 90, 180, 270} degrees. In each time period, the phase can change once. Since there are four possible phases, there are 2 bits of information conveyed within each time slot [23]. Modulation of the information sequence implies analysis of the system in the

time and frequency domain. The message $m_I(t)$ and $m_Q(t)$ is modulated with a carrier frequency, f_c, and amplitude, A_c. Adding the in-phase and quadrature components together formulates the QPSK signal, which is described as

$$x_c(t) = Am_I(t)\cos\omega_o t + Am_Q(t)\sin\omega_o t\,. \tag{3.11}$$

As Figure 3.22 illustrates, the Xilinx DDS Block implements a direct digital synthesizer (DDS), also commonly called a numerically controlled oscillator (NCO) to modulate the signal. The block employs a look-up table scheme to generate real or complex valued sinusoids. An internal look-up table stores samples representing one period of a sinusoid. A digital integrator (accumulator) is then used to generate a suitable phase argument that is mapped by the look-up table into the desired output waveform. Finally, combining or adding the upper (I) and lower (Q) parts will represent the QPSK modulated output [22].

The result of modulating an RF carrier with such a code sequence is to produce a signal centered at the carrier frequency. The main lobe of this spectrum has a bandwidth twice the clock rate of the modulating code, from null to null. The side lobes have a null to null bandwidth equal to the code's clock rate.

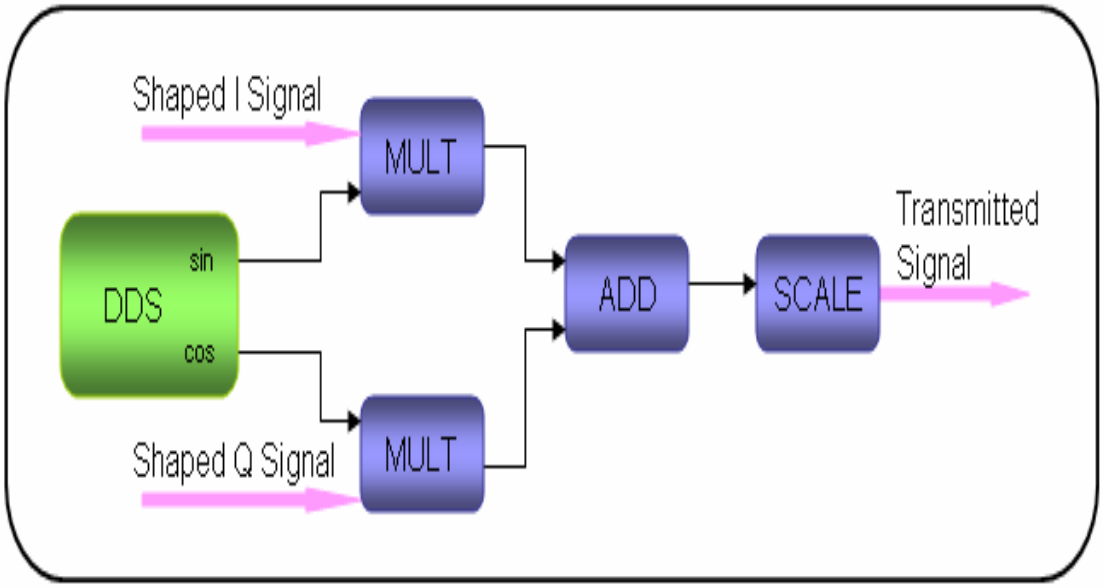

Figure 3.22: Modulation

Since the hardware platform has a 65MHz clock and the overall Simulink System Period is 1/10 according to the model specifications, the block parameters of the DDS block are selected to have a constant phase increment of 1/5 cycles per sample and a sampling period of 1/10 to obtain the following:

$$\left.\begin{aligned}
&\text{Chip Rate} = 6.5\ \text{MHz},\\
&\text{Tx Sample Rate} = 6.5\ \text{MHz} * 10 = 65\ MHz,\\
&\text{DDS Block} = \frac{1}{\frac{1}{5} cycle/sample} = 5\ \text{samples per cycle},\\
&\text{Carrier Frequency} = \frac{65MHz}{5} = 13\ MHz,\\
&\text{Symbol Rate} = \frac{6.5MHz}{63} = 103.2\ KHz.
\end{aligned}\right\} \quad (3.12)$$

Since the symbol rate is 103.2 KHz, each bit will have a period of 9.69 microseconds $(1/103.2KHz)$. The 65 MHz clock value is obtained from the hardware specifications and will be explained in detail in chapter 5 along with the Simulink System Period. The rate of change (baud) in this signal determines the signal bandwidth, but the throughput or bit rate for QPSK is twice the baud rate. Figure 3.23 shows the signal after it has been modulated, added together, and

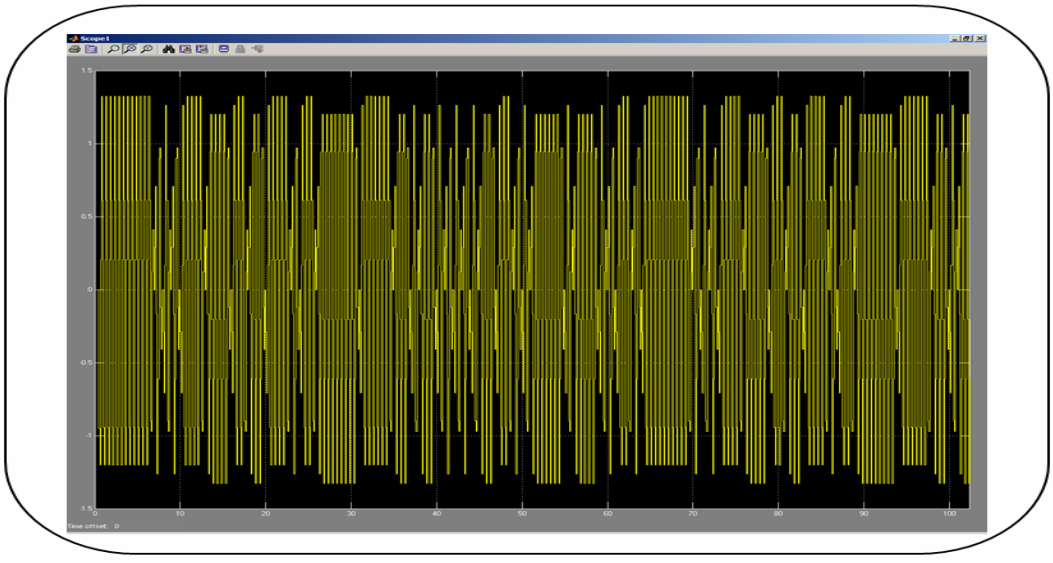

Figure 3.23: Signal after Modulation in Simulation

scaled down in amplitude for transmission. Ideally the amplitude of a QPSK signal is constant. However, pulse shaping the QPSK signals causes them to lose their constant envelope property [23]. This fluctuation of the envelope is apparent in Figure 3.23.

3.5 Channel

Before arriving at the intended receiver, the signal must go through a channel that adds noise and creates distortion effects. Even though the channel can be created in various ways, the ultimate result is that it degrades the signal transmitted to the receiver. Noise is characterized into two forms: external and internal. Internal noise results from components such as resistors and electron tubes. External noise results from outdoor sources such as the atmosphere [28]. In this system model, the channel only adds additive white Gaussian noise (AWGN) to the system. Typical characteristics of white Gaussian noise are a statistically independence of any two noise samples, a constant power spectral density $P_N(f)$ and an autocorrelation function $R_N(\tau)$ that consists of a weighted delta function , which are described as [29]

$$P_N(f) = \frac{N_o}{2} \ , \qquad R_N(\tau) = \frac{N_o}{2}\delta(\tau) \ . \tag{3.13}$$

The channel noise is characterized so that it ranges between a signal to noise ratio (SNR) of -7dB to 7dB. SNR is a measure of signal strength relative to background noise. Assuming that the input to the receiver is signal plus white Gaussian noise the channel output is

$$y(t) = Am_I(t)\cos\omega_o t + Am_Q(t)\sin\omega_o t + n(t) \ . \tag{3.14}$$

3.5.1 Noise Generation

Noise is generated through the Matlab *awgn* function, which adds Gaussian white noise to the signal relative to the SNR value defined by the user. The Gaussian wideband noise is sent through a 101-tap bandpass filter to constrain it to the same bandwidth as occupied by the signal. The bandpass filter coefficients are generated by the FDA toolbox with the following constraints:

$$
\left.
\begin{array}{ll}
F_s = 65 \text{ MHz} & A_{pass} = 1 \text{ dB}, \\
A_{stop1} = 80 \text{ dB} & A_{stop2} = 80 \text{ dB}, \\
F_{pass1} = 13 - 5 & F_{pass2} = 13 + 5, \\
F_{stop1} = 13 - 7 & F_{stop2} = 13 + 7.
\end{array}
\right\} \tag{3.15}
$$

Figure 3.24 illustrates the significance of these parameters in the filter design.

The resulting bandpass filter has a transfer function defined as

$$H_{BP}(f) = [H_1(f - f_o) + H_1(f + f_o)]e^{-j\omega t_o}, \tag{3.16}$$

where $H_1(f) = H_o \prod (f / B)$ and B is the single sided bandwidth. The magnitude and impulse response for this filter is given in Figures 3.25 and 3.26. The corresponding impulse response for this filter is

$$h_{BP}(t) = 2H_o B \; sinc \; B(t - t_o)\cos(t - t_o). \tag{3.17}$$

The MATLAB *filter* function filters the incoming noise with the filter described by coefficients generated by the FDA toolbox. Figure 3.27 and 3.28 show the noise before and after the filter, which represent the wideband and bandpass noise.

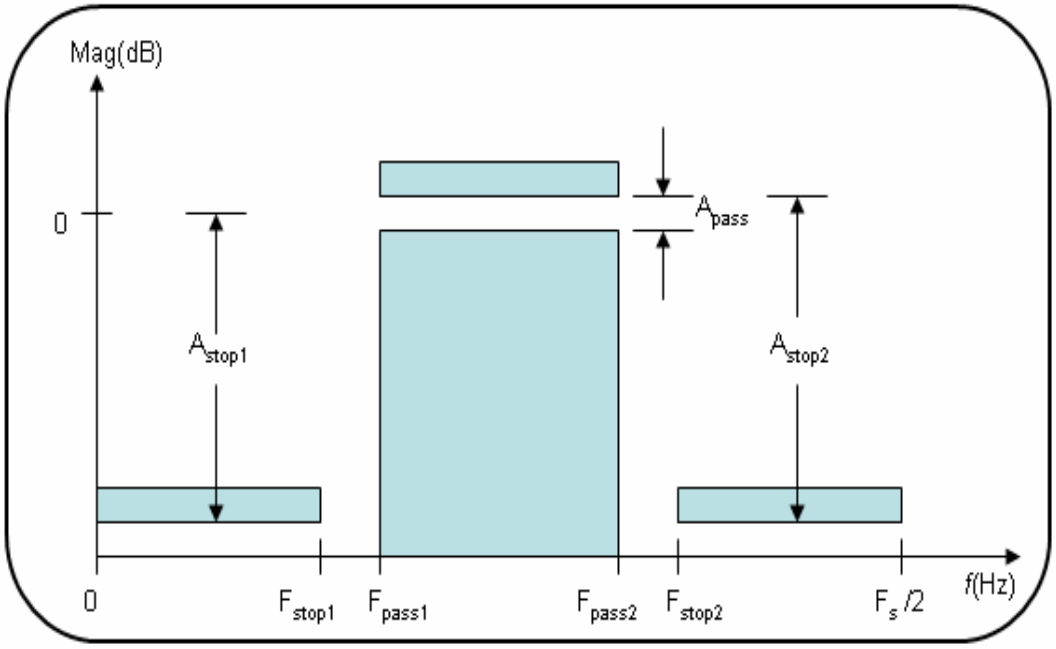

Figure 3.24: Bandpass Filter Specifications

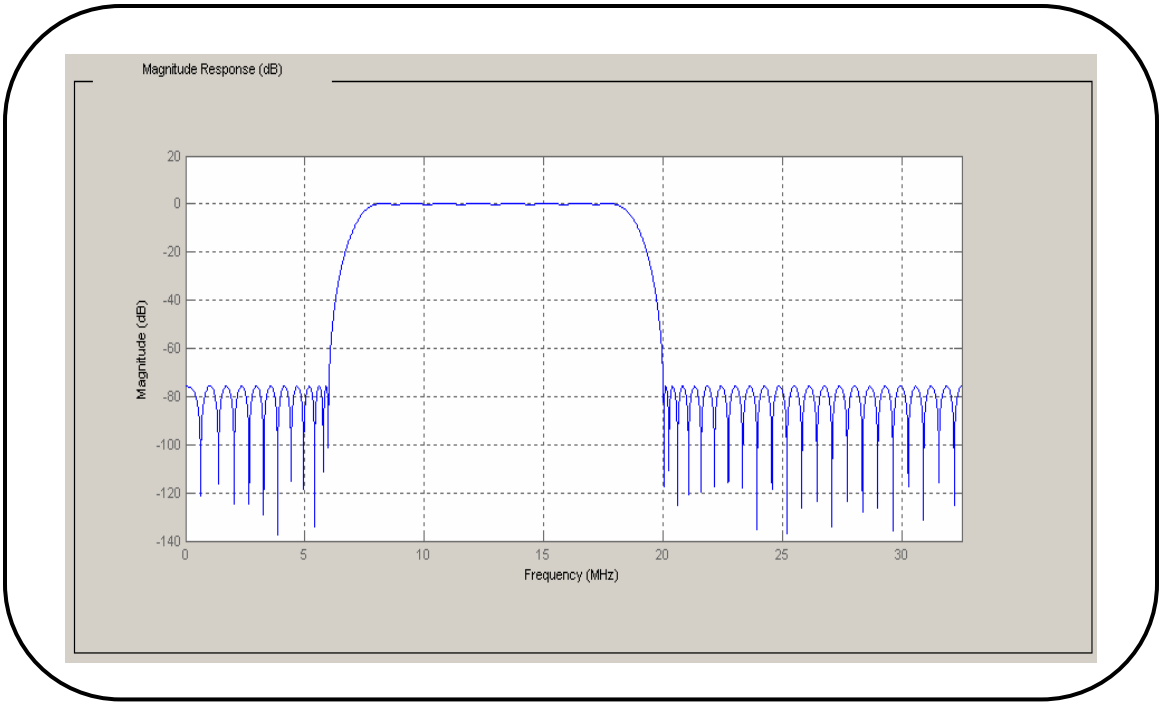

Figure 3.25: Magnitude Response of Bandpass Filter

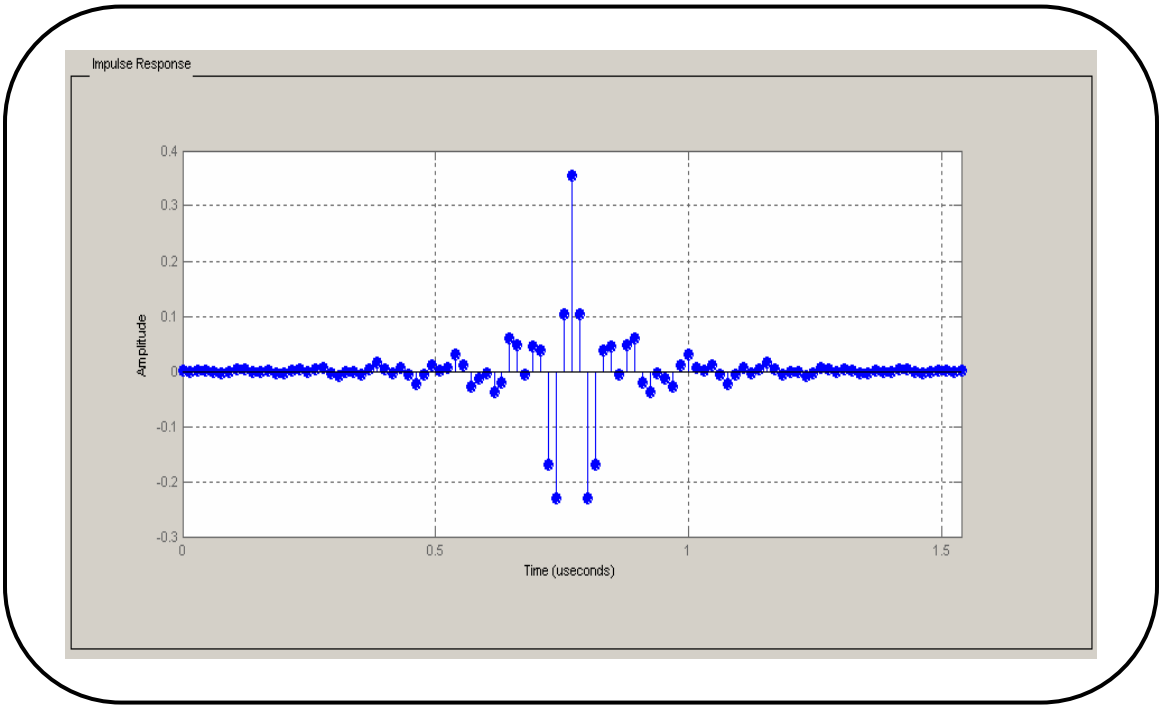

Figure 3.26: Impulse Response of Bandpass Filter

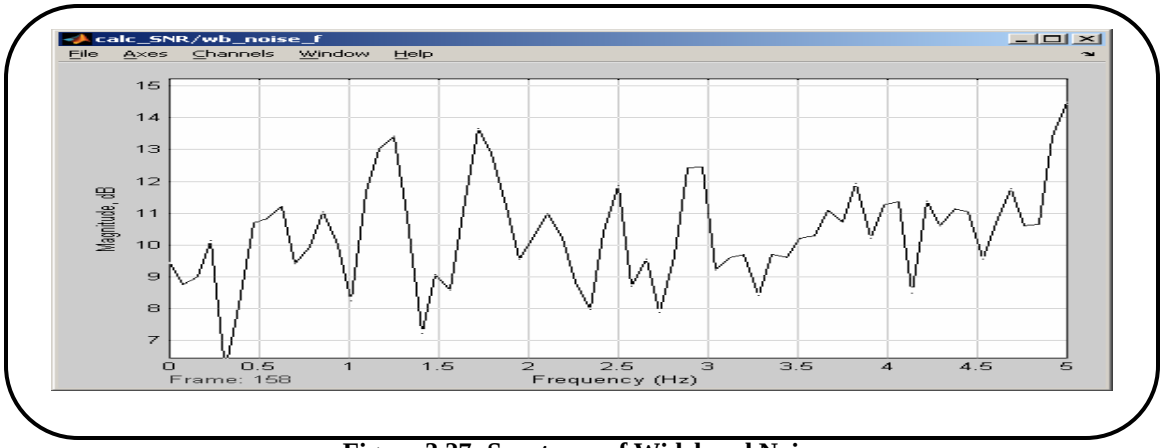

Figure 3.27: Spectrum of Wideband Noise

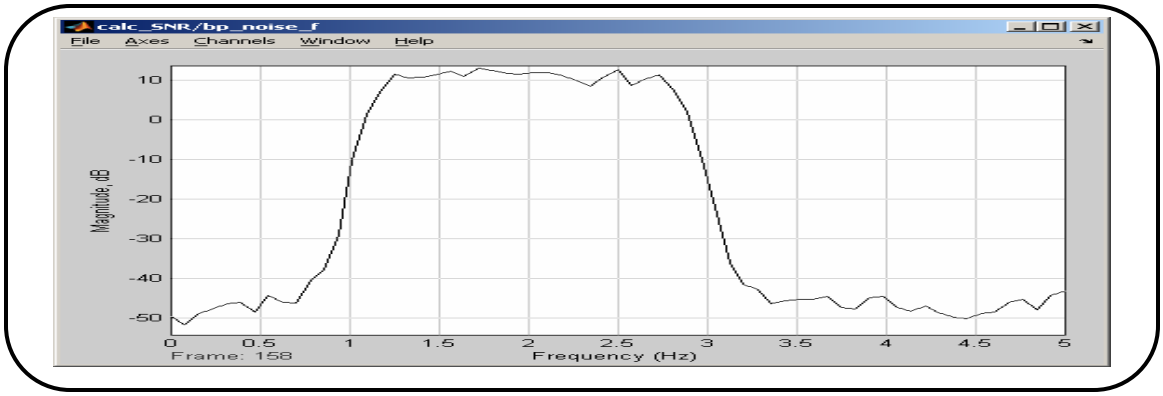

Figure 3.28: Spectrum of Bandpass Noise

The construction of the AWGN in the channel is illustrated in Figure 3.29. The resultant bandpass noise is stored in a vector and placed in a Xilinx *read only memory* (*ROM)* block, which is addressed through a counter. The depth of the *ROM* corresponds to the length of the input vector, the baseband noise. For Virtex devices, the maximum timing performance is possible only if the depth of the ROM is less than 16,384 [22]. Therefore, the depth of the ROM and the length of the baseband noise vector are set to 16,000. The baseband noise vector is repeatedly added to the transmitted signal and then scaled down before it is sent to the receiver. The spectrum of the signal before and after the noise is shown in Figures 3.30 and 3.31. Scaling of the signal is necessary since the signal will be passed through a digital to analog converter (DAC) and analog to digital converter (ADC) before being sent to the receiver. The signal must be scaled to an unsigned 14-bit signal due to the hardware specifications of the ADC and DAC. Further details regarding the hardware specifications of the ADCs and DACs are provided in chapter 5.

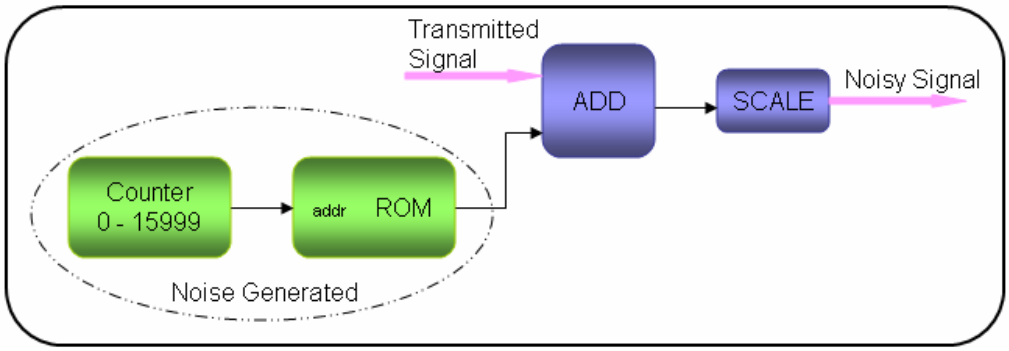

Figure 3.29: Design of Channel

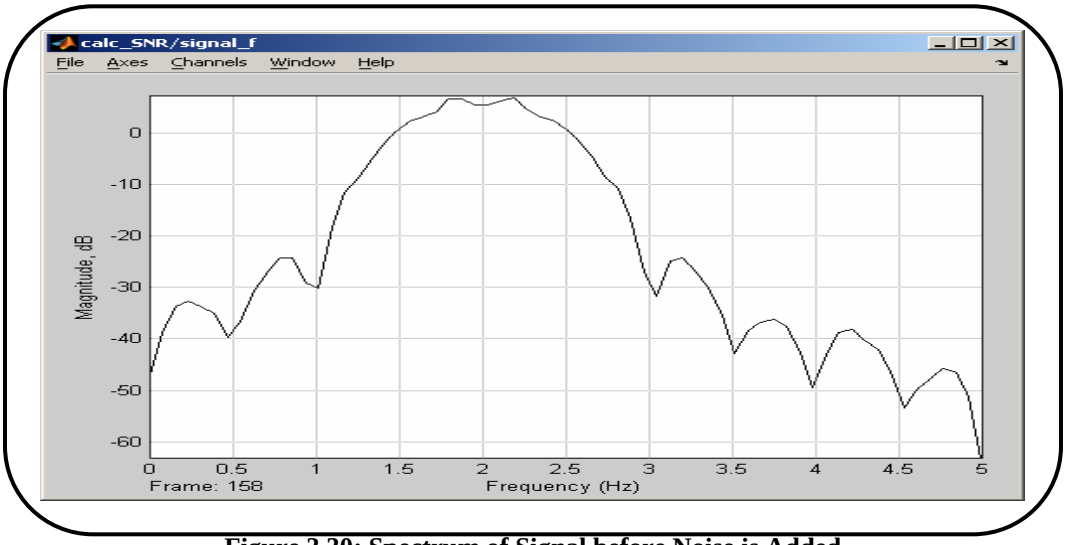

Figure 3.30: Spectrum of Signal before Noise is Added

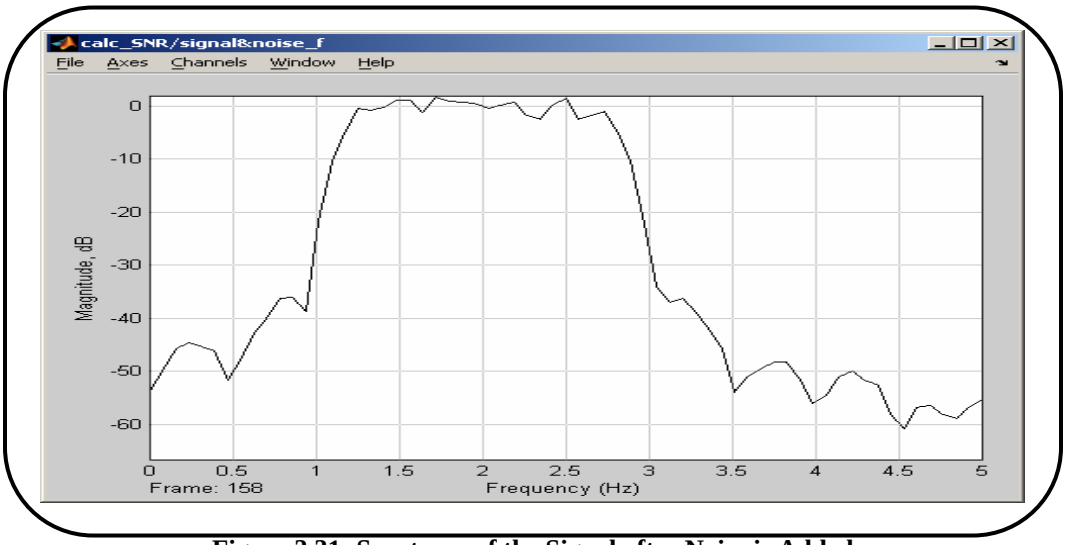

Figure 3.31: Spectrum of the Signal after Noise is Added

3.5.2 Calculating SNR

The noise performance of various types of systems is examined by evaluating the signal-to-noise power ratio at the receiver output. Let the output of the transmitter be defined as the transmitted signal, *S(t)*, and the output of the *ROM* block as the baseband noise signal, *N(t)*. Then, the SNR values are obtained by squaring and summing the signal and noise values over the period *K* and then converting the value into dB scale, which are described as

$$SNR = \sum_{t=0}^{K} \frac{S(t)^2}{N(t)^2} \quad , \tag{3.18}$$

$$SNR_{dB} = 10\log_{10}(SNR). \tag{3.19}$$

The value of *K* is selected such that the difference between *K* summations and $K+1$ summations is less than one percent.

Eight different noise signals corresponding to SNR values of {-7, -5, -3, -1, 1, 3, 5, 7} dB are generated and stored in *ROM* blocks. The eight different SNR values needed in the *awgn* function to obtain SNR values of {-7, -5, -3, -1, 1, 3, 5, 7} dB are found by performing several simulations and adjusting the values until desired results are obtained.

3.5.3 Changing Noise Levels

Instead of having the user change the noise value in software for each new simulation, the model is designed so that the noise value can be changed in hardware. As a result, the user saves time by not needing to generate a bit file for the system model each time the noise value is changed. As shown in Figure 3.32, a Xilinx *gateway in* block is used as a select input to a multiplexer that chooses various noise levels. The ROM blocks are initialized to eight different noise levels to be selected by the user. Since there are 8 different inputs to the mux, the *gateway in* block must be allocated to three pins on the Nallatech board. Through these pins the user will be able to select the desired SNR. The noise settings will be controlled by a user with a dipswitch that is wired to the corresponding pins on the hardware platform. Further explanation of pin allocation and hardware setup will be addressed in Section 5.2.5 of Chapter 5 and Section 4.5 of Chapter 4.

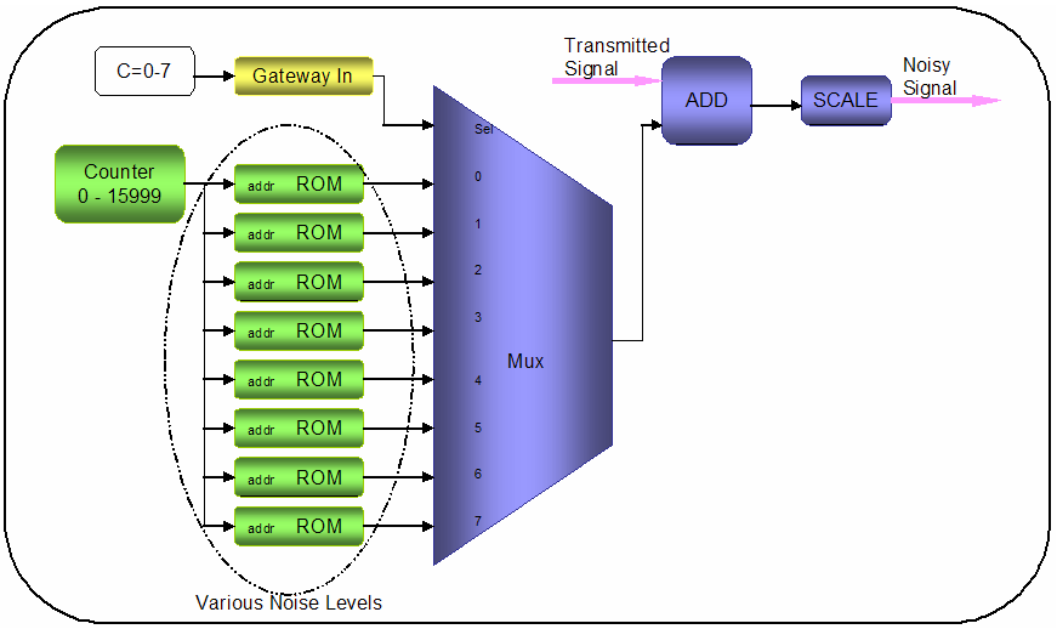

Figure 3.32: Design for Implementing Various Noise Levels

3.6 QPSK Receiver

The main function of the receiver is to extract the desired signal from the received signal at the channel output. Figure 3.33 shows that the primary components of a QPSK receiver which include down conversion, baseband processor and packet processor. The downcoversion block includes blocks to perform digital down conversion and low pass filtering. The baseband processor is composed of the following blocks: carrier phase lock loop, I and Q correlators, tracking lock loop, peak detector, rectangular to polar converter, and phase decoder. In a receiver, the received signal is first coherently demodulated and low-pass filtered to recover the message signals (in-phase and quadrature channels). The next step for the receiver is to sample the message signals at the symbol rate and decide which symbols were sent. Although the symbol rate is typically known to the receiver, the receiver does not know when to sample the signal for the best noise performance. The objective of the symbol-timing recovery loop is to find the best time to sample the received signal [28]. The presence of noise complicates this operation. The signal is de-spread by correlation with the original PN spreading sequence and then sent to the packet processor.

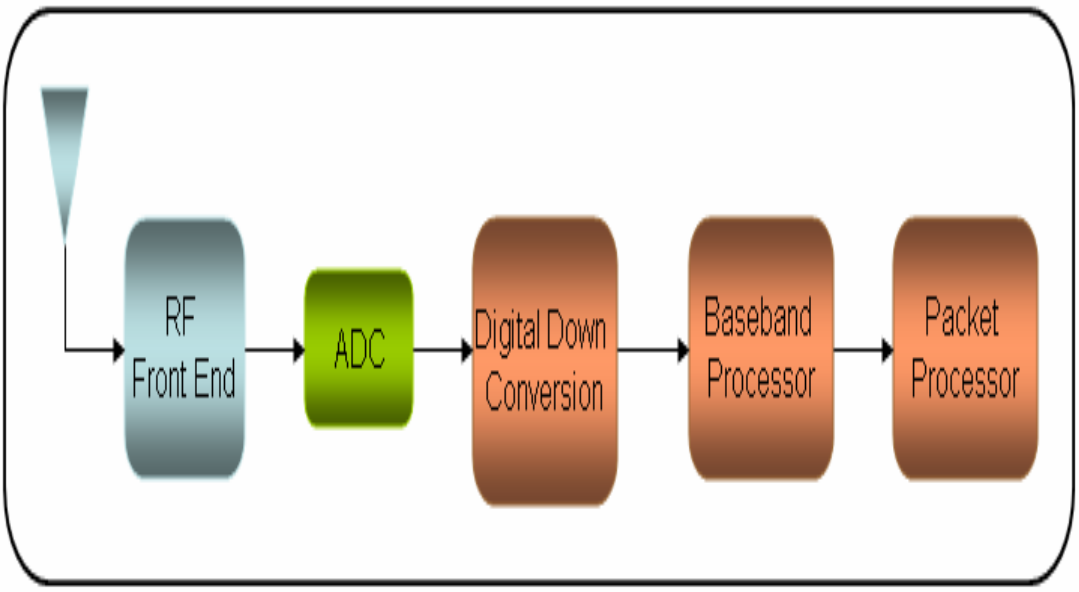

Figure 3.33: QPSK Receiver

3.6.1 Down Conversion

The QPSK modulated 13 MHz IF signal is the input to the receiver. As illustrated in Figure 3.34, the transmitted signal is first coherently demodulated with both a sine and cosine, and then filtered to remove the double-frequency terms, yielding the recovered in-phase and quadrature signals, $s_I[n]$ and $s_Q[n]$. The DDS block is designed with a constant phase increment of 1/5 cycles per sample and a sampling period of 1/10 just as it was designed in the transmitter for modulation purposes.

Figure 3.35 represents the analysis on the signal in the frequency domain after down conversion. Down conversion brings the signal from the RF frequency to baseband. As illustrated by the figure, some type of filtering is necessary to eliminate the double frequency components occurring at $-2f_c$ and $2f_c$.

The inclusion of a channel adds a delay between the transmitter and receiver. Therefore there is a difference in the clock cycles of the transmitter and receiver. This is evident in Figure 3.36, which shows the resultant I and Q signals in simulation after down conversion.

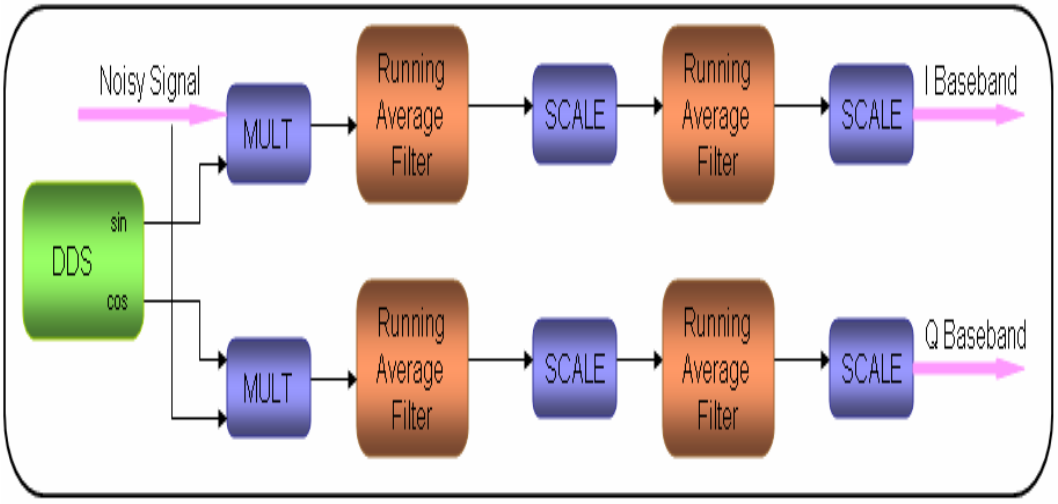

Figure 3.34: Digital Down Conversion Block

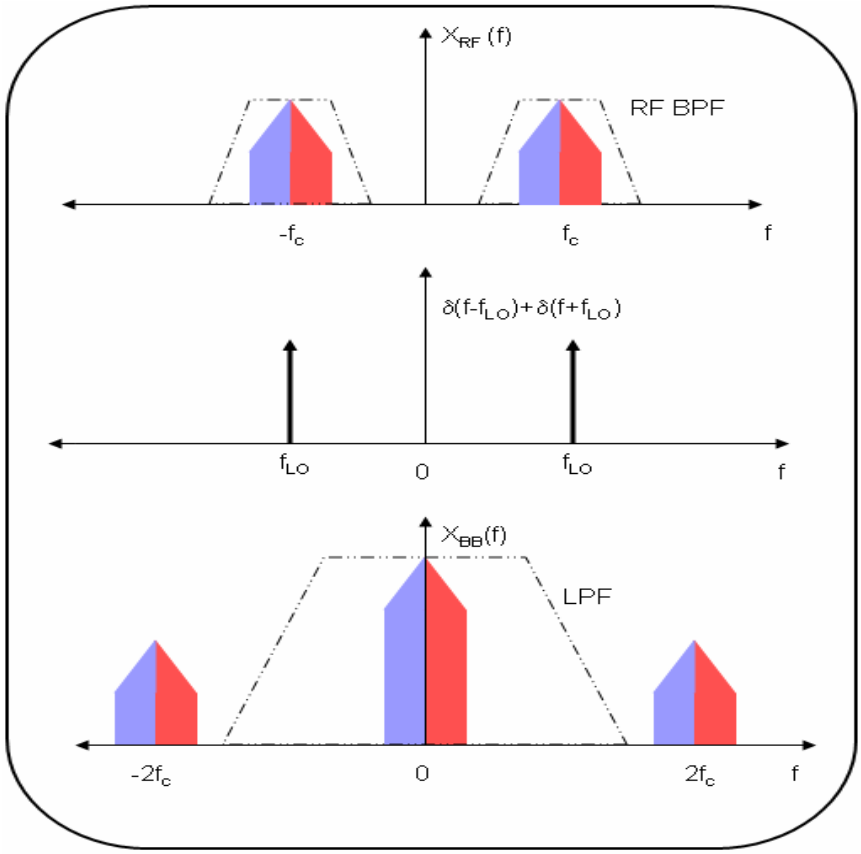

Figure 3.35: Frequency Domain Analysis for Down Conversion

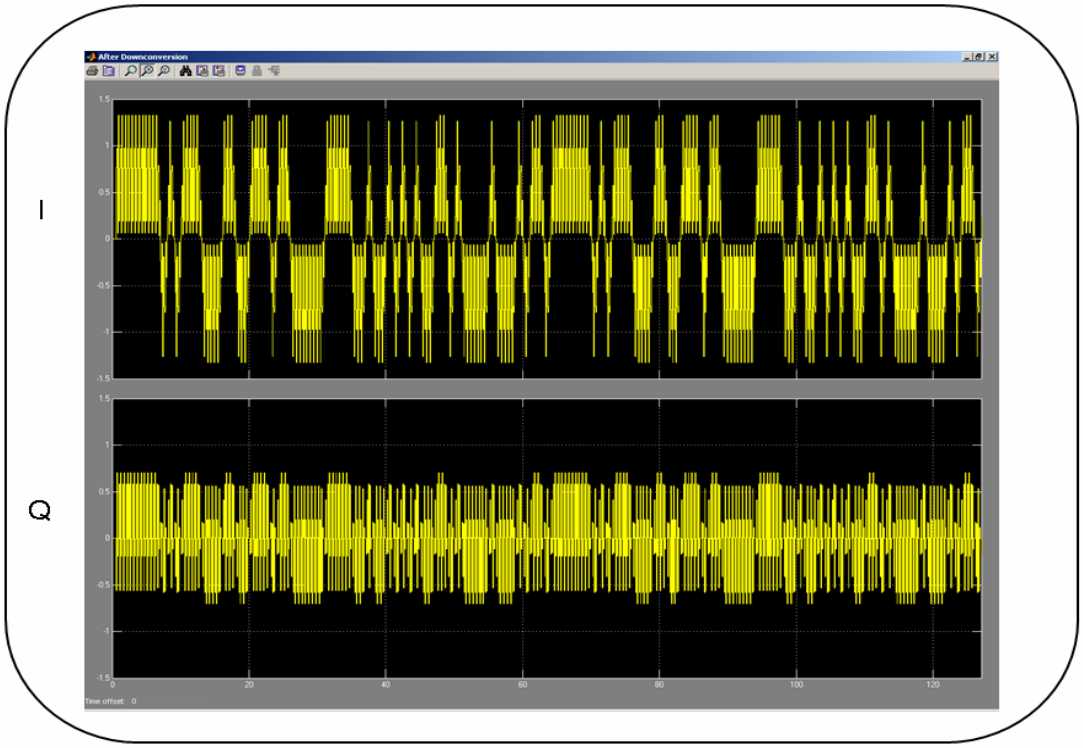

Figure 3.36: Signal after Down Conversion

3.6.2 Running Average Filter

After down conversion, low pass filtering is necessary to get rid of the double frequency components. Since designing a lowpass filter occupies a significant amount of hardware space, an alternative design known as running average filter is chosen. In the case of a digital signal, a Finite Impulse Response (FIR) filter that would have as an output the average of the last N values of an input signal can be easily created. Such a system is sometimes called a running average filter. The running average filter can be imagined as a window of size N moving along the array, one element at a time [30]. The impulse response of this filter is

$$h(u) = [1111.....1]. \tag{3.20}$$

Recall that the general difference equation of an FIR filter is

$$y(n) = \sum_{k=0}^{\infty} b_k * x(n-k). \tag{3.21}$$

Equation 3.21 shows that in an LTI system the output *y(k)* is the resultant of the convolutional sum of the input *x(u)* and the channel response *h(u)*, which is equivalent to

$$y(k) = \sum_{u=0}^{N-1} x(u) * h(k-u). \tag{3.22}$$

The properties of convolution allow Equation 3.21 to be equivalent to

$$y(k) = \sum_{u=0}^{N-1} x(k-u) * h(u). \tag{3.23}$$

Since $h(u) = [1111.....1]$, *y(k)* can be simplified to

$$y(k) = \sum_{u=0}^{N-1} x(k-u) * 1. \tag{3.24}$$

Therefore, *y(0)* would just be the summation to the first *N-1* values, given by

$$y(0) = \sum_{u=0}^{N-1} x(u). \tag{3.25}$$

As shown in Figure 3.37, *y(1)* would be the summation of the next *N-1* vectors shifted over one, which is described as

$$y(1) = \sum_{u=0}^{N-1} x(u-1). \tag{3.26}$$

This is equivalent to taking output of *y(0)* and adding element *x(u-1)* and subtracting element *x(N-1)*. Therefore,

$$y(1) = y(0) + x(u-1) - x(N-1). \tag{3.27}$$

This next summation is computed by taking the previous summed value and adding the previous component and subtracting the last component. Thus, the difference equation for the running average filter would be

$$y(n+1) = y(n) + x(u-n) + x(N-n). \tag{3.28}$$

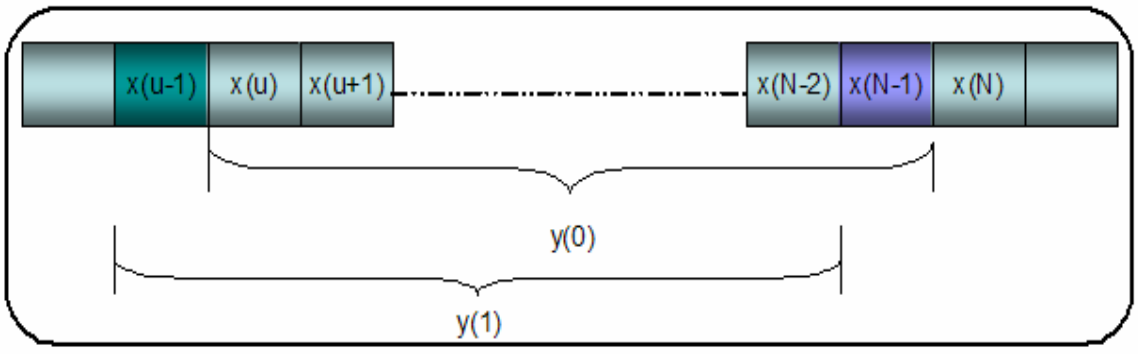

Figure 3.37: Running Average Filter Example

Figure 3.38 shows the model of a running average filter (RAF). The accumulator sums the *x(n)* from 0 to N-1 and stores the output in a register. An addressable shift register is used for tracking the last component of *x(n)*. A delay block is used to index the next component of *x(n)* to be added.

The running average filter has the effect of "smoothing out" fluctuations in a signal. The effect of smoothing out the fluctuations is equivalent to a low pass filter; that is, a filter which removes higher frequency components in a signal while leaving behind lower frequency components. Thus, a running average filter is equivalent to one form of a low pass filter.

Changing the window range varies the position of the notch of the filter in the frequency response. The best window range, *N*, was evaluated by determining which value would result in the notch eliminating the double frequency components. Considering the magnitude response of the signal before filtering (Figure 3.39), it is obvious where the notches need to be placed.

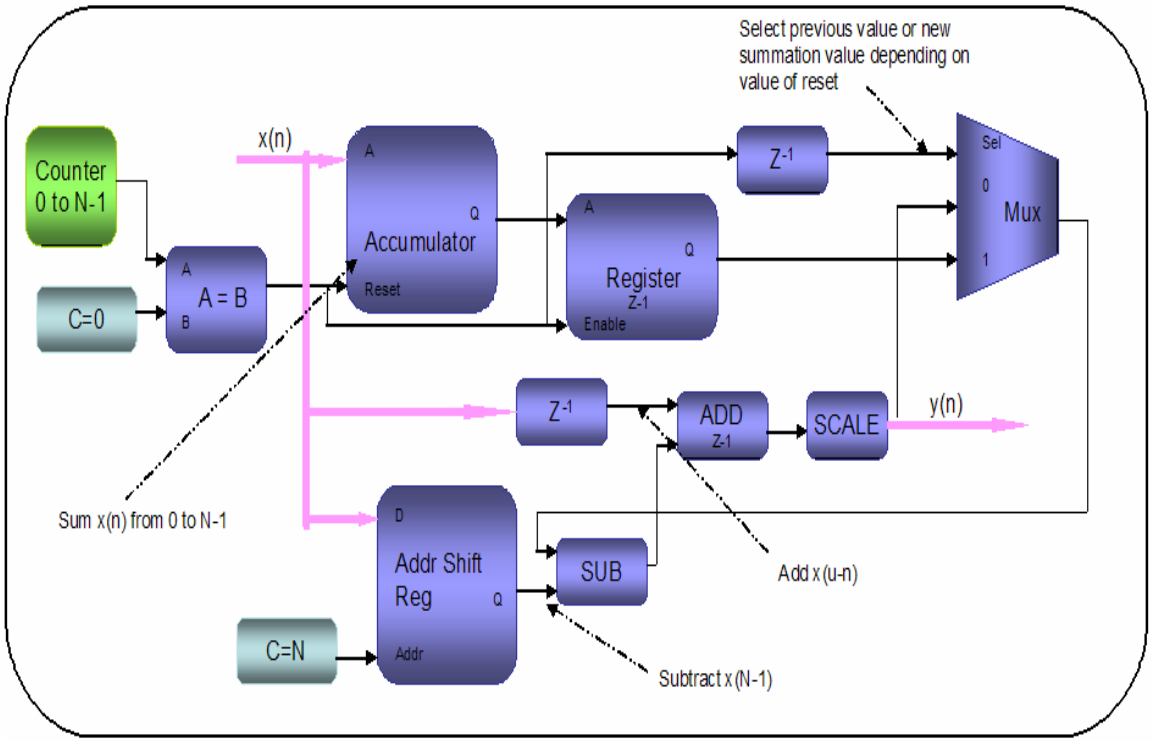

Figure 3.38: Running Average Filter Model

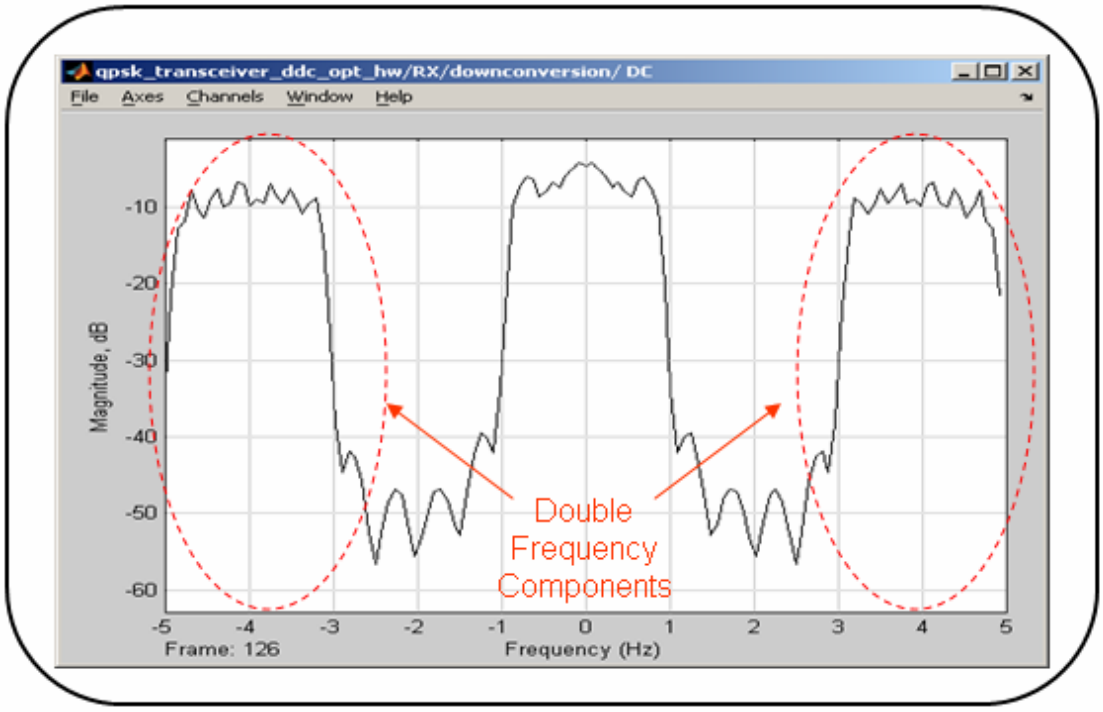

Figure 3 39: Magnitude Response after Down Conversion

The window range is varied until the notch is placed at the center of the double frequency components. Figure 3.40 shows the magnitude response of the RAF when *N=5*. The magnitude response of the signal after the first RAF is shown in Figure 3.41. Figure 3.41 illustrates that having just one running average filter does not remove enough of the higher frequency components. Therefore, another RAF is added in the system design. Since the window range *N* can only be integers, it is strategically calculated to obtain the best result. Figure 3.42 shows that *N=3* provides the best result. The magnitude response of the signal after the second RAF is shown in Figure 3.43. This result reflects the effects of both filters.

Since the filters occupy a significant amount of hardware space, scaling the output after the first RAF is necessary. The accumulator increases the resolution of the output which is not needed. Scaling the output also prevent overflow from occurring in the second RAF.

The simulation results of the signal after RAF1 and RAF2 are shown in Figures 3.44 and 3.45. As indicated by the results, the second RAF improves the smoothness of the signal significantly.

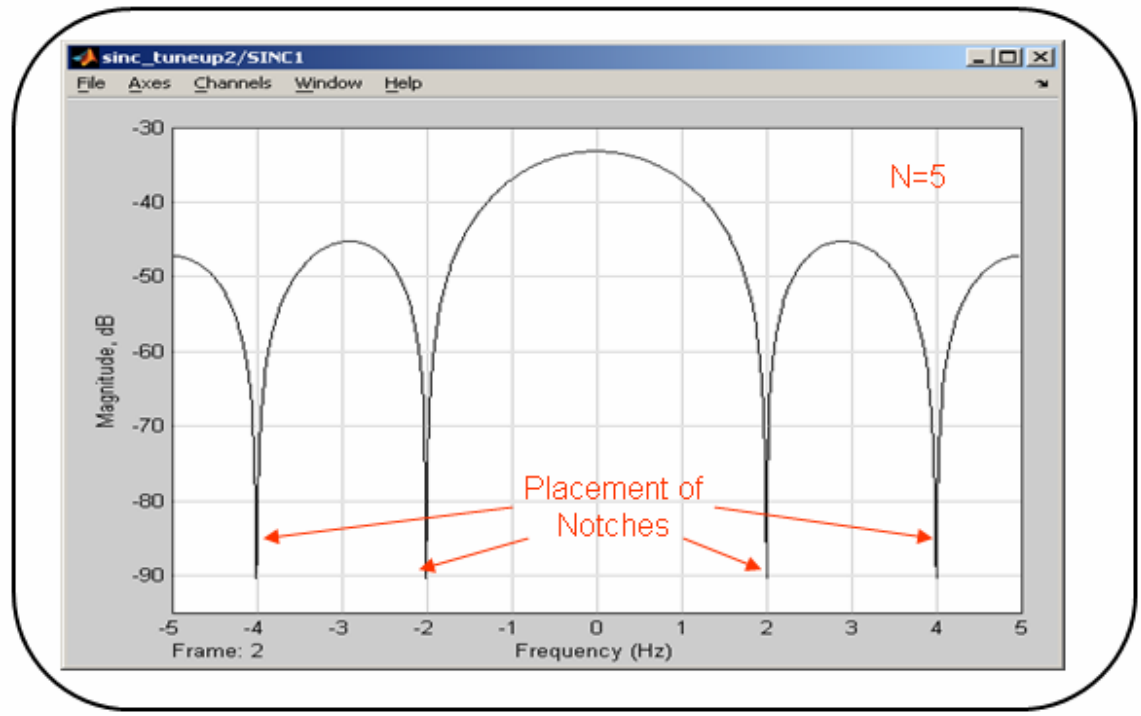

Figure 3.40: Magnitude Response of Filter 1

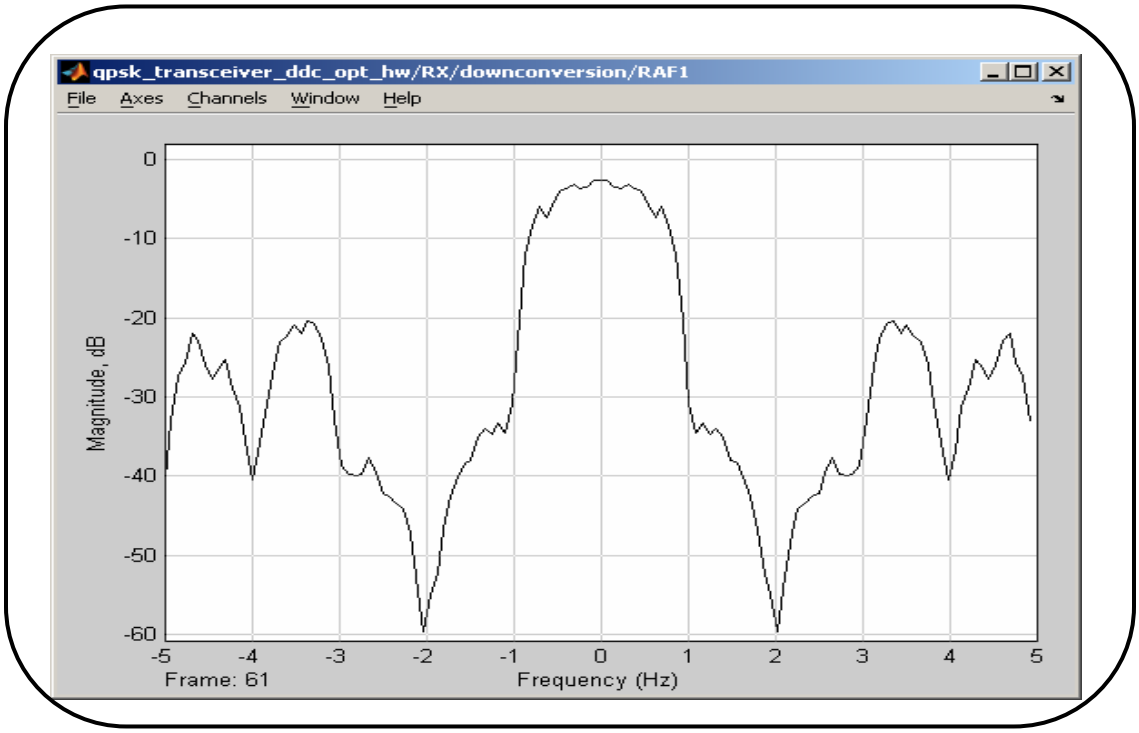

Figure 3.41: Magnitude Response of Signal after Filter 1

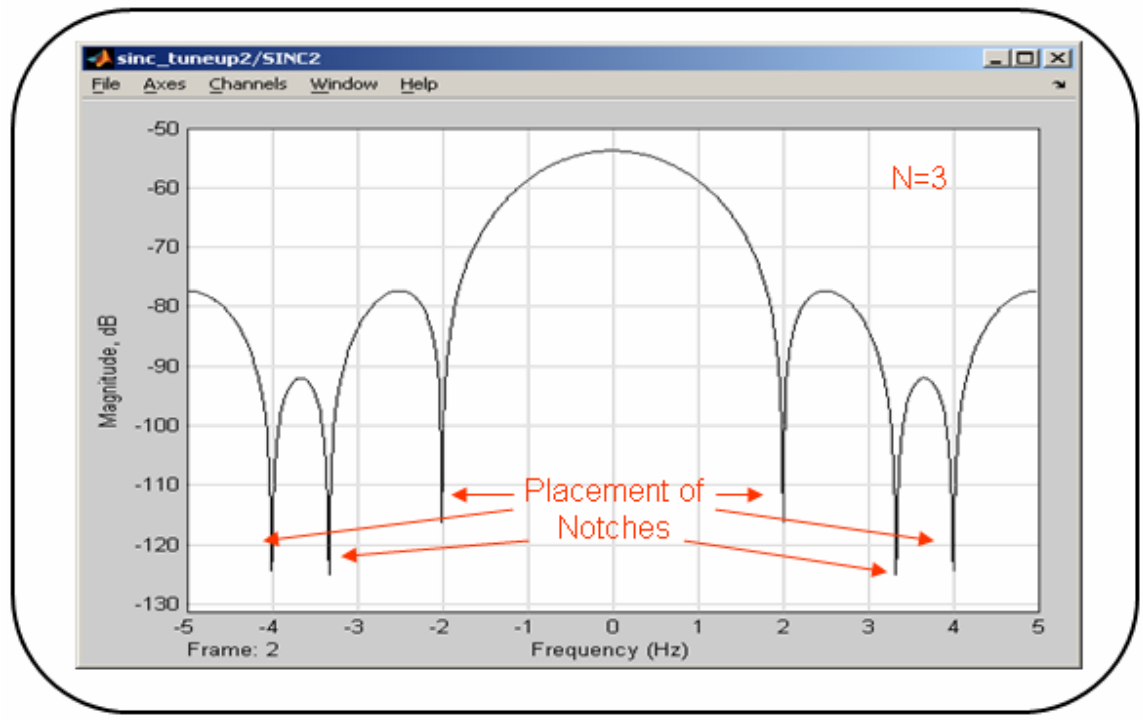

Figure 3.42: Magnitude Response of Filter 1 and 2

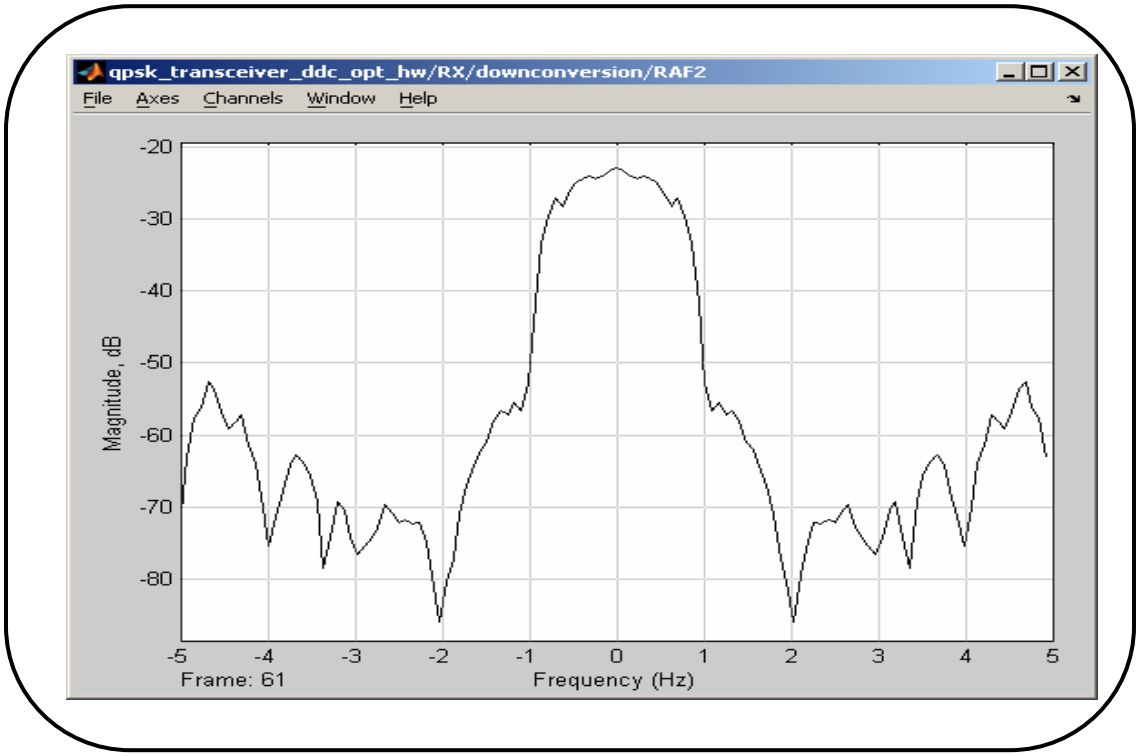

Figure 3.43: Magnitude Response of Signal after Filter 2

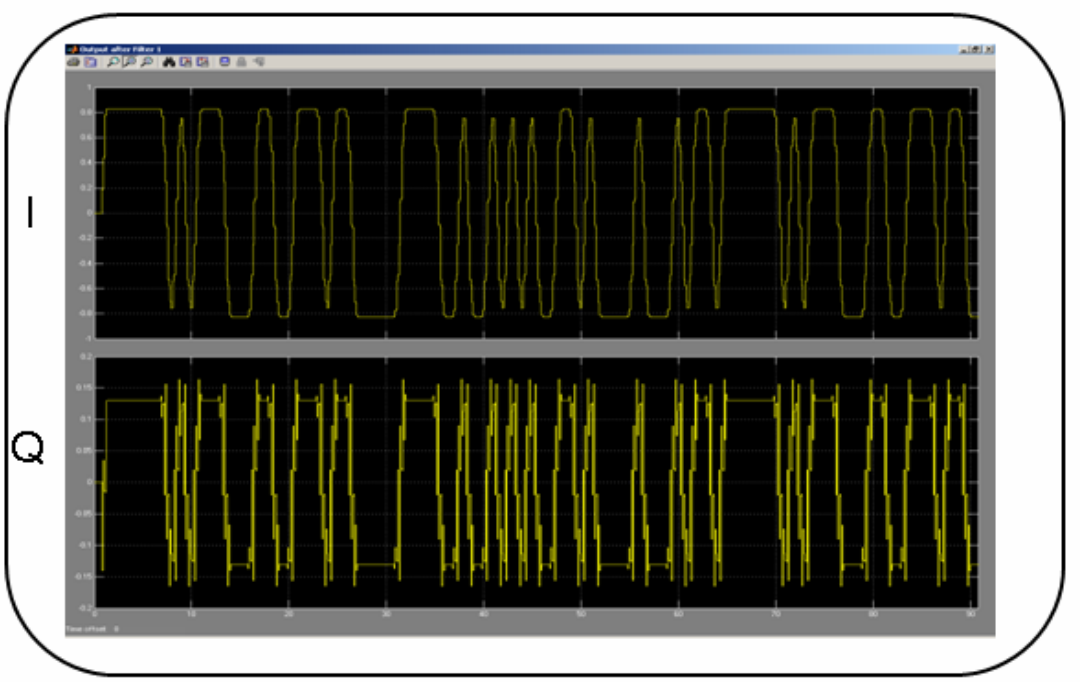

Figure 3.44: Signal after Running Average Filter 1

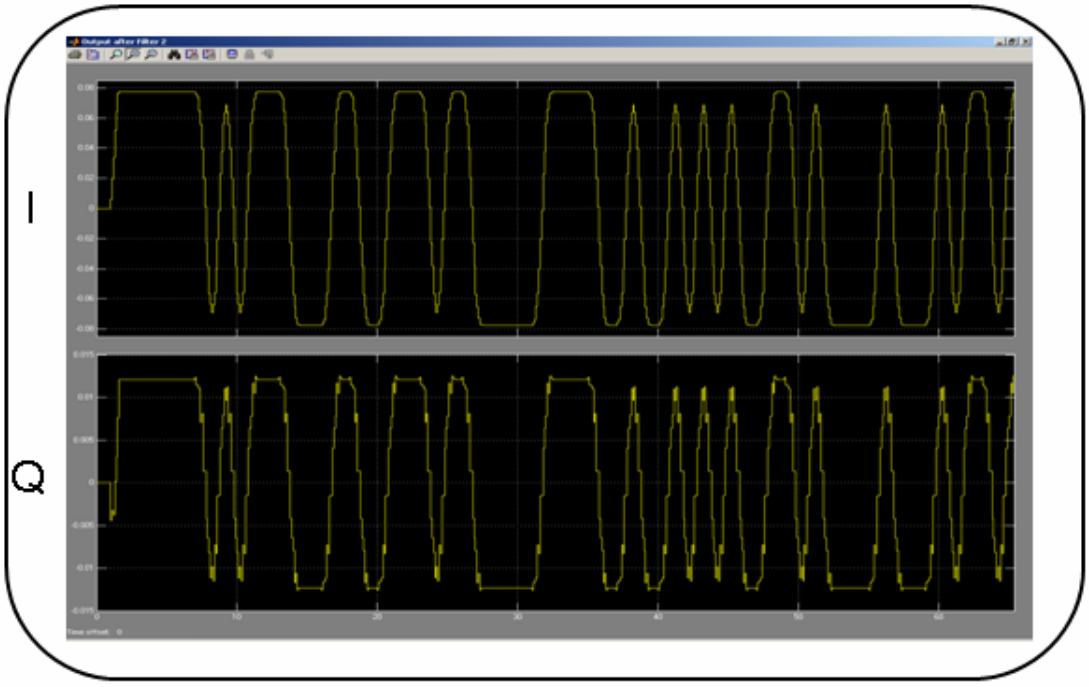

Figure 3.45: Signal after Running Average Filter 2

3.6.3 Baseband Processor

After the signal has been downconverted from the IF frequency to the zero-IF complex base band signal by the DDC and filtered by the LPF, it is sent to the baseband processor. As shown in Figure 3.46, the baseband processor consists of the following blocks: carrier phase lock look, tracking phase lock loop, I&Q correlators, CORDIC for rectangular to polar conversion, peak detector and phase decoder. The carrier phase locked loop locks to the zero-IF baseband signal, while the tracking loop is performing symbol tracking. Both the carrier phase lock loop and the tracking loop are feedback loops that must be accurately modeled. Two separate correlators constructed from FIR blocks serve as match filters for I and Q baseband signals. A rectangular-to-polar block performing a CORDIC algorithm provides the magnitude and phase of the received signals. The magnitude value is sent to the peak detector to provide early and late gate values to the tracking loop for synchronization purposes, while the phase value is sent to the phase decoder to evaluate the value of the received symbol.

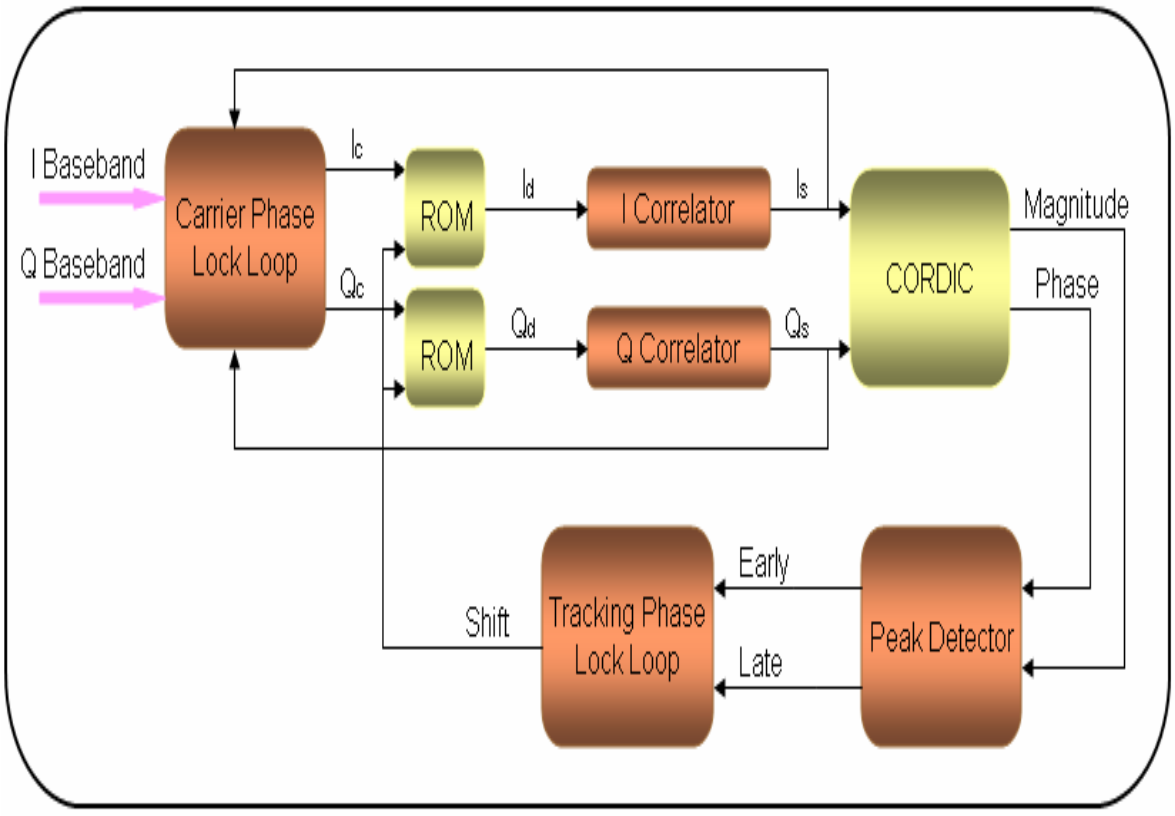

Figure 3.46: Baseband Processor

3.6.4 Carrier Phase Lock Loop

In digital communications, to recover the transmitted signal, the output of the demodulator must be sampled once per symbol interval. Since the receiver does not know the delay between the transmitter and receiver, symbol timing must be derived from the received signal for synchronous sampling of the output from the demodulator. The delay in the transmitted signal also causes an offset to the carrier, which the receiver must estimate.

Therefore, a carrier phase lock loop (CPLL) must be implemented to generate a version of the local oscillator that is matched in both frequency and phase to the oscillator employed in the transmitter. A Costas Loop, developed by Costas, is one method of implementing a phase-locked loop (PLL). Figure 3.47 represents the design of the Costas Loop.

The PLL is a critical component in coherent communications receivers that is responsible for locking on to the carrier of a received modulated signal. The PLL consists of two basic functional blocks: a numerically controlled oscillator (NCO) and a loop filter (LF). Ideally, the transmitted carrier frequency is known exactly and only the phase needs to known to demodulate correctly. However, due to imperfections at the transmitter, the actual carrier frequency may be slightly different from the expected frequency. This difference between the expected and actual carrier frequencies can be modeled as a time-varying phase. Provided that the frequency

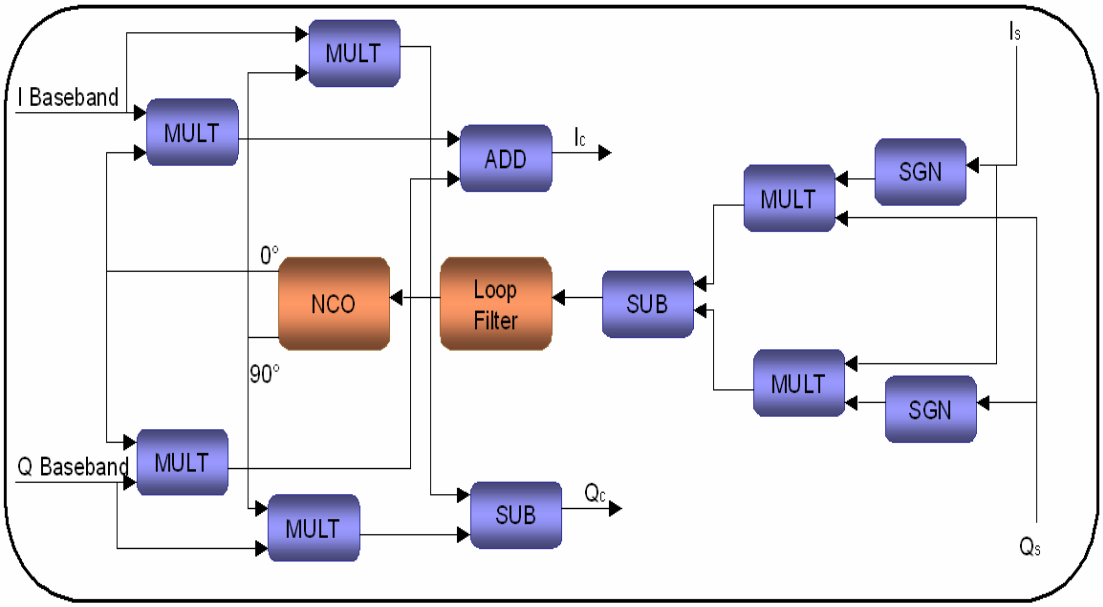

Figure 3.47: Costas Loop

mismatch is small relative to the carrier frequency, the feedback control of an appropriately calibrated PLL can track this time-varying phase, thereby locking on to both the correct frequency and the correct phase [32].

In an analog system this recovery is often implemented with a voltage-controlled oscillator (VCO) that allows for precise adjustment of the carrier frequency based on the output of a phase-detecting circuit. In our digital application, this adjustment is performed with a numerically-controlled oscillator (NCO) as shown in Figure 3.48. The NCO is basically a sinusoidal signal generator.

The received signal *s(t)* is multiplied by the outputs of the NCO, which are

$$\cos(2\pi f_c t + \hat{\phi}) \text{ and } \sin(2\pi f_c t + \hat{\phi}), \tag{3.29}$$

where $\hat{\phi}$ represents the estimate of ϕ. The product of these two implies that the phase error is equivalent to the difference between the two, given by

$$\Delta\phi = \hat{\phi} - \phi \tag{3.30}$$

The error signal is evaluated by finding the absolute magnitude difference between the incoming signals I_s and Q_s. This error signal is filtered by the loop filter, whose output is the control voltage that derives the NCO. Figure 3.49 models the structure of the loop filter. The constants K_P and K_I of the loop filter control the way the loop responds to its initial excitation, whether it is overdamped, underdamped, or critically damped.

In the synchronized (called locked) state the phase error between the oscillator's output signal and the reference signal is zero or very small. If a phase error builds up, a control mechanism acts on the oscillator in such a way that the phase error is again reduced to a minimum. In such a feedback control system the phase of the output signal is actually locked to the phase of the reference signal. This is why it is referred to as a phase-locked loop. The loop is set to reset at the end of each packet to prevent error buildup. Figure 3.50 illustrates the error that arises due to the phase difference. The constants of the loop filter K_P and K_I are adjusted to obtain the minimum error.

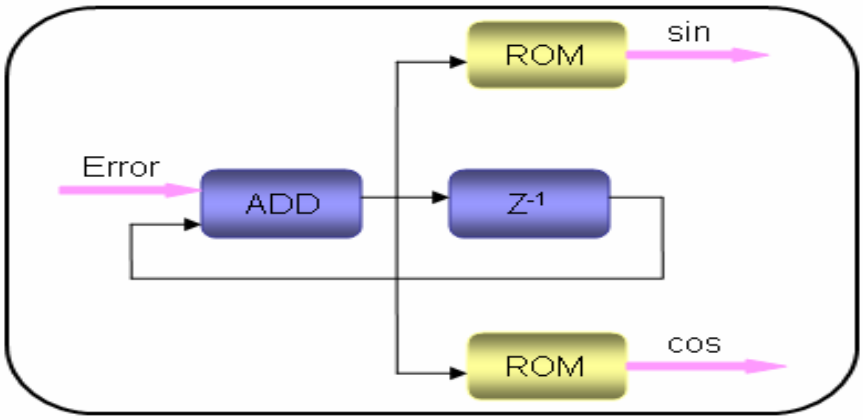

Figure 3.48: NCO

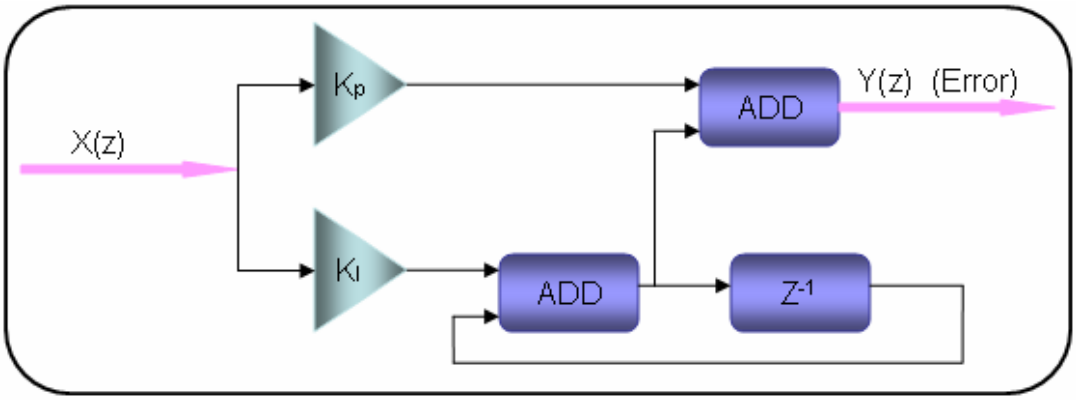

Figure 3.49: Loop Filter

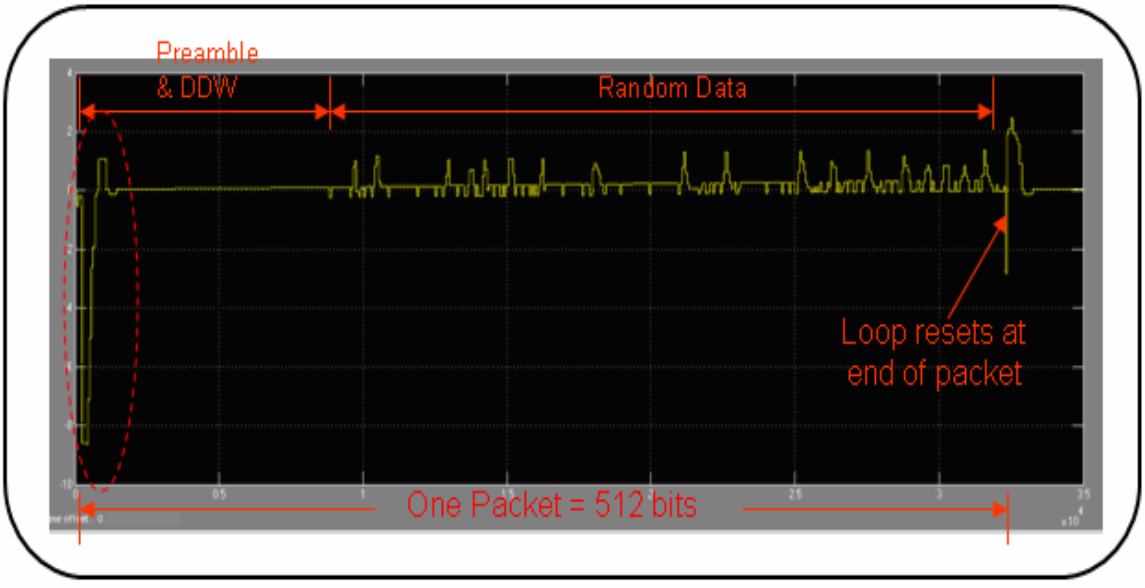

Figure 3.50: Error of PLL

3.6.5 Parallel Correlator

In spread-spectrum systems, the receiver must synchronize onto the transmitted PN code and de-spread the received signal into the original symbol by calculating the correlation of input data and the PN sequence. The receiver adjusts the timing offset to search the maximum correlation value. The completely parallel architecture provides the fastest synchronization and good accuracy, however in the full implementation requires a tap length equal to the spreading sequence length. Figure 3.51 illustrates that the parallel correlators are implemented by two *M*-tap finite impulse response (FIR) filter, defined by *M* filter coefficients, or taps, each represented as a Xilinx fixed-point number [22].

An FIR filter with *M*-length input *x(n)* and output *y(n)* is described by the following difference equation

$$\begin{aligned} y(n) &= b_o x(n) + b_1 x(n-1) + \ldots + b_{M-1} x(n-m+1) \\ &= \sum_{k=0}^{M-1} b_k x(n-k), \end{aligned} \tag{3.31}$$

where b_k is the set of filter coefficients. Alternatively, the output, *y(n)*, can be expressed as the convolution of the unit impulse response *h(n)* of the system with the input signal, which is described as

$$y(n) = \sum_{k=0}^{M-1} x(n-k) * h(k). \tag{3.32}$$

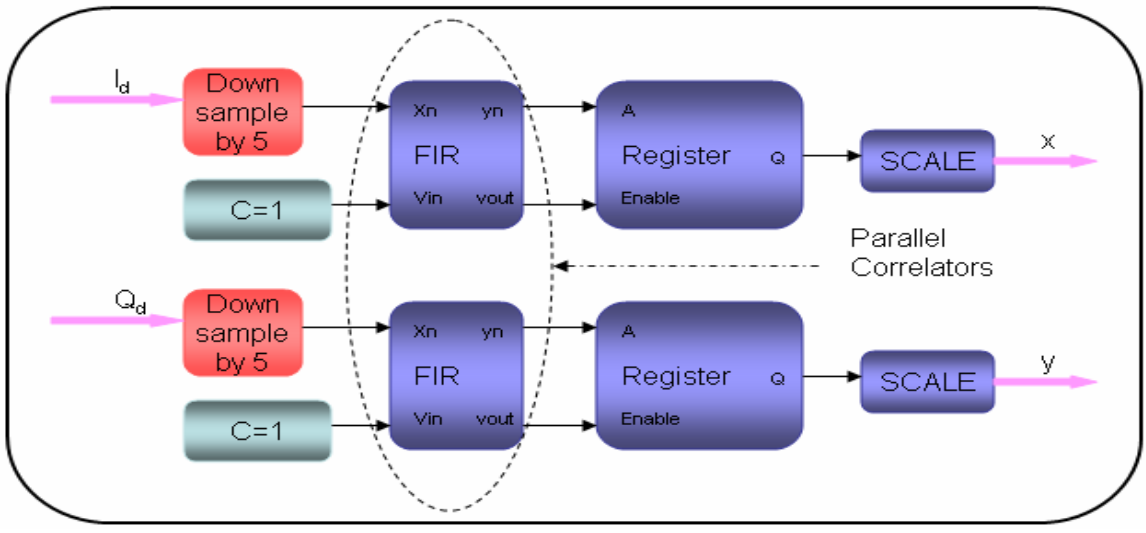

Figure 3.51: Implementation of Parallel Correlators

Equation 3.31 and 3.32 are identical in form, and thus $b_k = h(k)$, where *h(k)* are the set of user-defined coefficients which represent the PN sequence [23]. The filter block accepts a stream of Xilinx fixed-point data samples x(0), x(1), ..., and at time *n* computes the convolution sum defined by equation 3.32. The conventional tapped delay line realization of this inner-product calculation is shown in Figure 3.52.

The input samples are serially shifted into a shift register. The shift register's taps drive the memory block's address buses. At each clock cycle, the sum of memory blocks outputs gives an intermediate sum-of-multiplications result. The accumulator at the end of the adder tree gives the complete FIR filter result after n clock cycles (*n* is the resolution of the input sample). Figure 3.53 shows a correlator example. The correlator slides the code sequence to the right of the received samples and searches for one of the correlation points that has the maximum correlation value.

Perfect correlation results in peaks as shown in Figure 3.54. The correlation peaks occur at increments of $T = 63$ and represent a duration of one packet length. As illustrated by the simulation results, the correlation peaks vary in amplitude until the PLL locks. Variations in magnitudes of correlation peaks occur due to noise added by the channel.

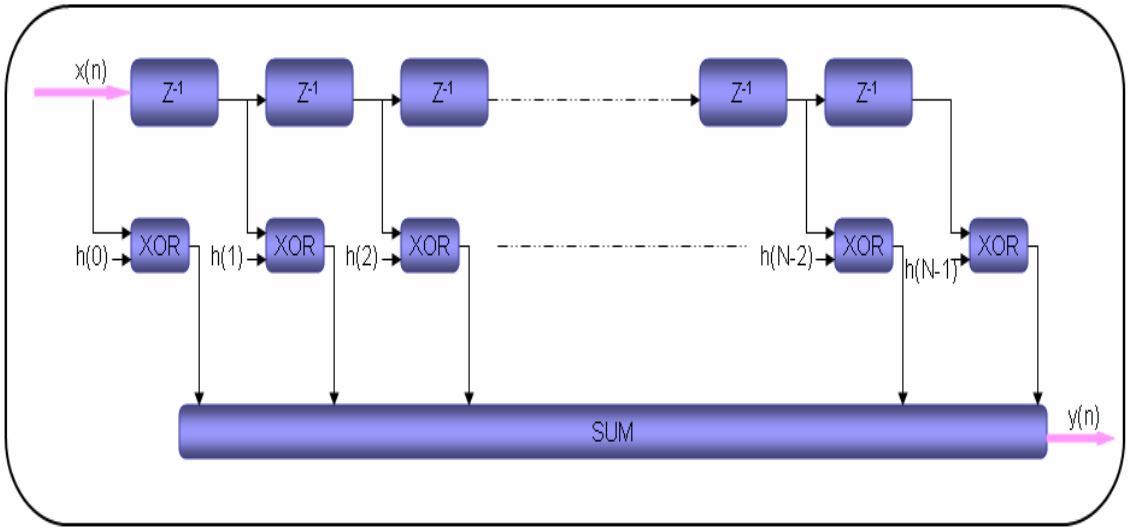

Figure 3.52: FIR Filter

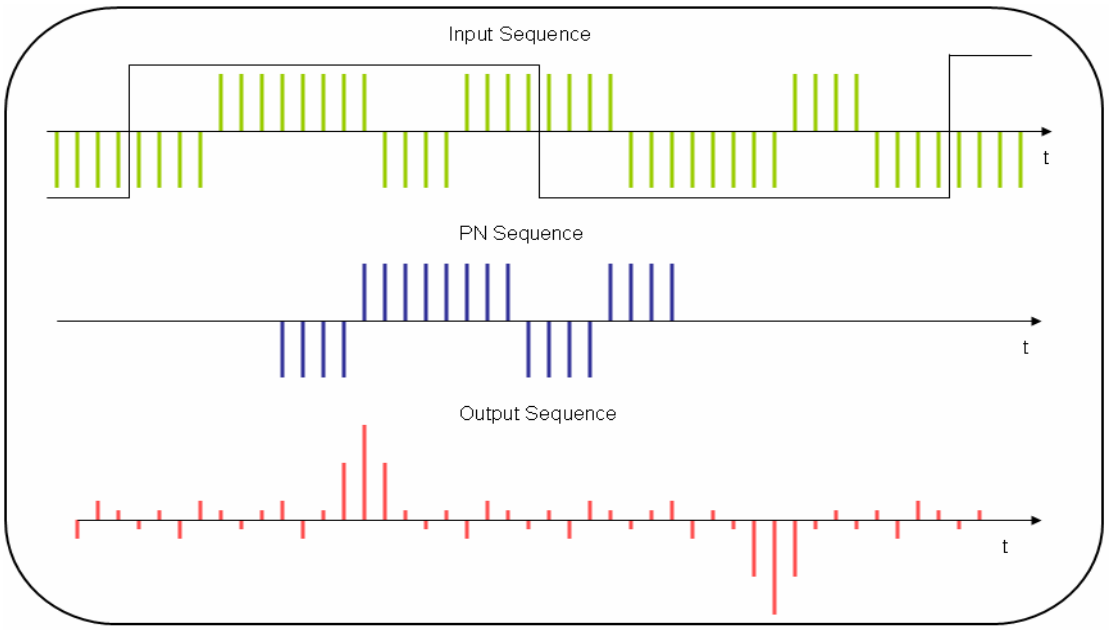

Figure 3.53: Example of Correlator

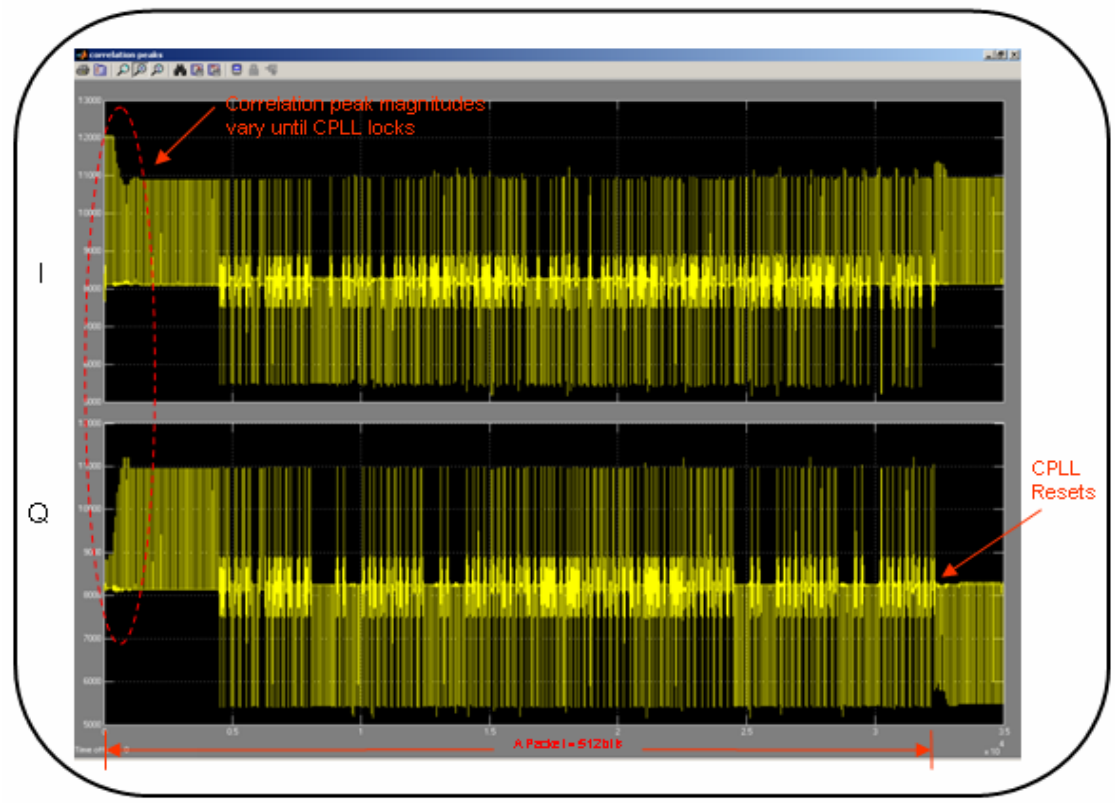

Figure 3.54: Output of Correlator in Simulation

3.6.6 CORDIC

Coordinate Rotation Digital Computer (CORDIC), shown in Figure 3.55, is an iterative algorithm for calculating trigonometric functions including sine, cosine, magnitude and phase. It is particularly suited for hardware implementations because it does not require any multiplies. CORDIC revolves around the idea of rotating the phase of a complex number, by multiplying it by a succession of constant values. However, the multiplies can all be powers of 2, so in binary arithmetic they can be done using just shifts and adds (no actual multiplier is needed) [23]. Given a complex-input <*x,y*>, CORDIC computes a new vector <*m,a*>, where magnitude and angle are defined as

$$m = \sqrt{x^2 + y^2}\ , \tag{3.33}$$

$$a = \tan^{-1}(\frac{y}{x})\ . \tag{3.34}$$

The *x* and *y* inputs to the CORDIC block must have the same data width and binary point. These two constraints are defined in the block parameters of the CORDIC and will ensure that the output values (*m* and *a*) have the same precision [23]. Therefore, the *x* and *y* inputs must be scaled prior to the CORDIC to ensure enough bits to represent the phase values accurately. The CORDIC block also adds latency, where *latency = 3 + number of processing elements*. The number of processing elements is a user defined parameter that indicates the number of iterations performed for fine angle rotation [6]. This added latency must be taken into account when the signal is sent to the peak detector.

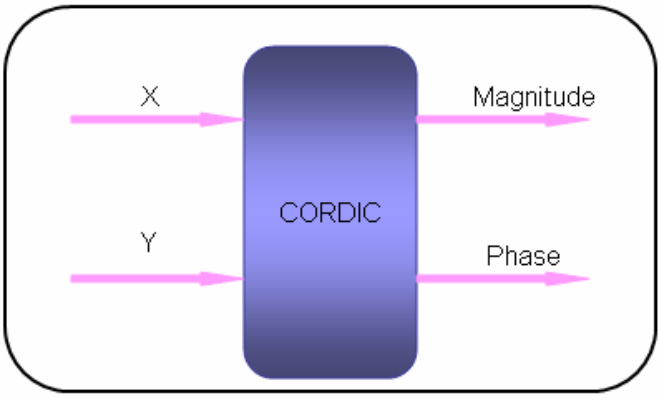

Figure 3.55: CORDIC

3.6.7 Peak Detector

The goal of the peak detector is to sample the waveform at the peak points in order to obtain the best performance in the presence of noise. The peak detector finds peaks without assistance from the user. When it begins running, it arbitrarily selects a sample, called the on-time sample, from the correlator output. The sample from the time-index one greater than that of the on-time sample is the late sample, and the sample from the time-index one less than that of the on-time sample is the early sample. As shown in Figure 3.56, the peak detector outputs the on-time, early and late sample. The on-time sample is used as an enable signal to indicate when to extract the phase values outputted from the CORDIC. The early and late sample values are sent to the tracking phase lock loop for fine synchronization. Also, a Max-Latch value is evaluated and fed back into the CPLL.

Figure 3.57 shows the system model for the peak detector. It is comprised of mainly delay, register, and logic blocks. The Max-Latch value takes into account the signal delay obtained from the CORDIC. The Max Latch value acts as an enable signal for the register, informing it of the location of the maximum peak relative to the output from the correlators. These resultant I and Q values from the registers are fed into the CPLL, where their absolute value difference defines the error signal. The integrated values from the correlator are delayed by one to represent the optimum sampling time and by two for the late sampling time relative to the early sampling time. Delay blocks are used to create enable signals for register blocks for correct referencing.

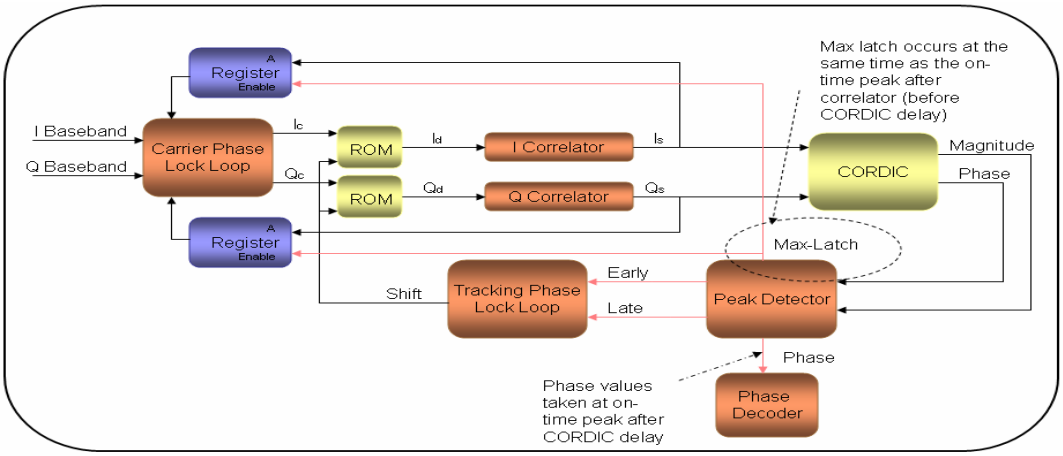

Figure 3.56: Peak Detector Outputs

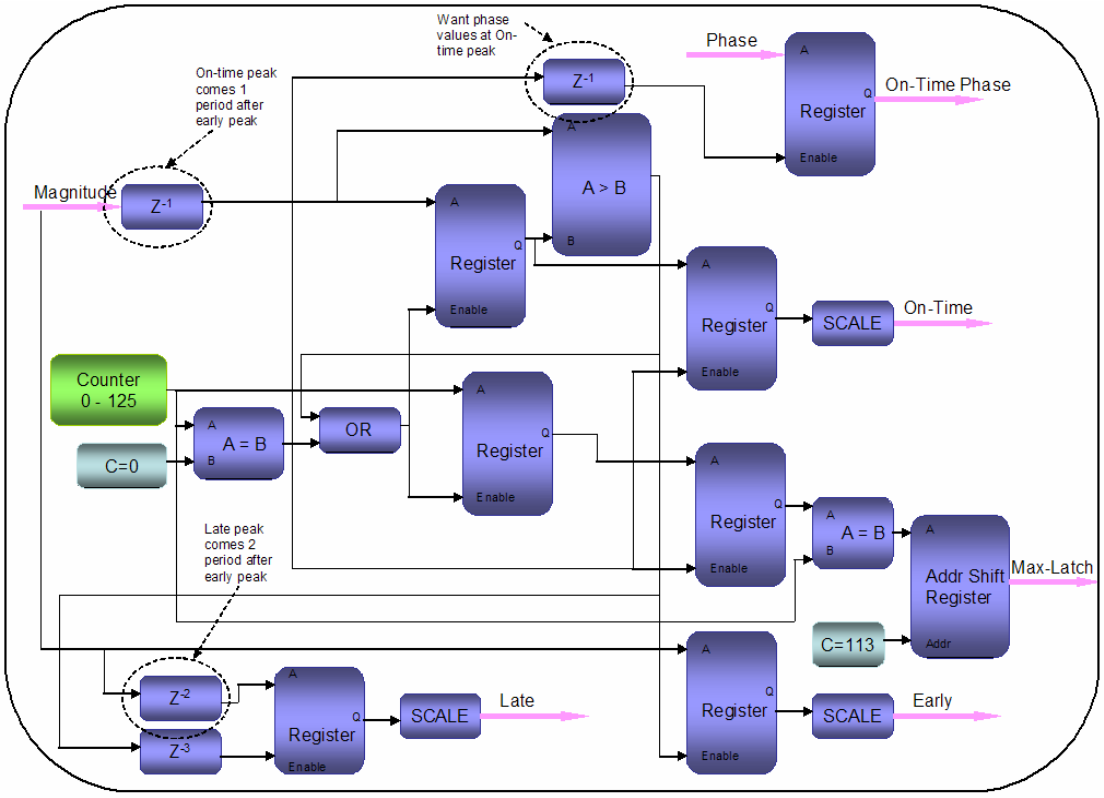

Figure 3.57: Peak Detector Model

3.6.8 Tracking Phase Lock Loop

The fundamental goal of the tracking phase lock loop or early-late gate synchronizer is symbol tracking. The early-late gate synchronizing technique exploits the symmetry of the signal. The output from the correlator attains its maximum value at time $t = T$. Thus the output is the time autocorrelation function of the pulse *s(t)*. Therefore, the proper time to sample the output is at the peak of the correlation function. Addition of noise from the channel increases difficulty of identifying the peak value of the signal. Instead of sampling the signal at the peak, early and late samples are taken. These samples occur at $t = T - \delta$ and $t = T + \delta$. Figure 3.58 clarifies this concept. On average, the absolute value of these samples will be smaller than the peak value. Since the autocorrelation function is symmetric, the absolute value of the early and late samples should be equivalent. Thus, the proper sampling point is midway between $t = T - \delta$ and $t = T + \delta$. This condition forms the basis of the early-gate synchronizer or tracking phase lock loop (TPLL) [23].

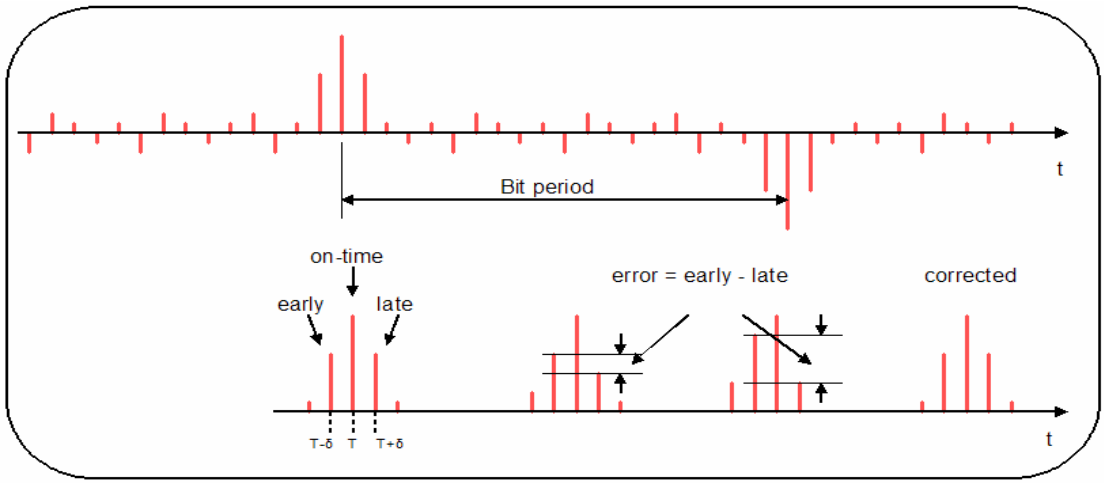

Figure 3.58: Early, Late, and On-time Samples

Figures 3.59 and 3.60 illustrate the block diagram of the TPLL. The early and late peaks are sent to the TPLL for calculating the difference between the two, which is denoted as the error signal. Therefore, the error signal is

$$\Delta = (T + \delta) - (T - \delta). \tag{3.35}$$

If the on-time sample occurs at the peak, the difference between early and late is zero. But when the peaks shift and result in an incorrect on-time sample due to timing errors, the TPLL must adjust the timing of on-time samples to coincide with peaks in the waveform.

The difference between the absolute values of the early and late gates formulates the error signal. To smooth the noise corrupted signals, the error signal is passed through a loop filter. This loop filter works in the same manner as the one in the CPLL. The constants K_P and K_I of the loop filter are adjusted to obtain the best results in simulation.

Any timing offset relative to the optimum sample time will result in a nonzero output of the error signal at the output of the filter. This smoothed error signal also derives the VCC output. Depending on if $\Delta > \Delta_{thresh\ early}$, or $\Delta < \Delta_{thresh\ late}$, the signal should be retarded or advanced [23]. Therefore, the pulse is either advanced or delayed depending on the error value and then fed back into the FIR filters for correlation [23]. Driving the error signal to zero by means of a feedback loop leads to maximum likelihood timing recovery. As shown in Figures below, this closed-loop control resets at the end of each packet.

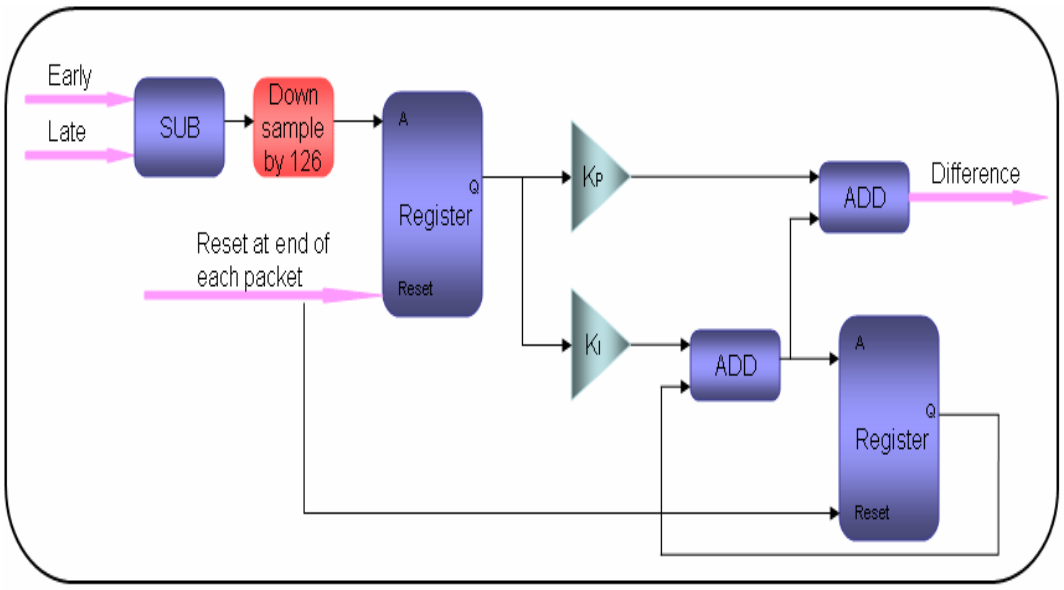

Figure 3.59: Tracking Phase Lock Loop Model

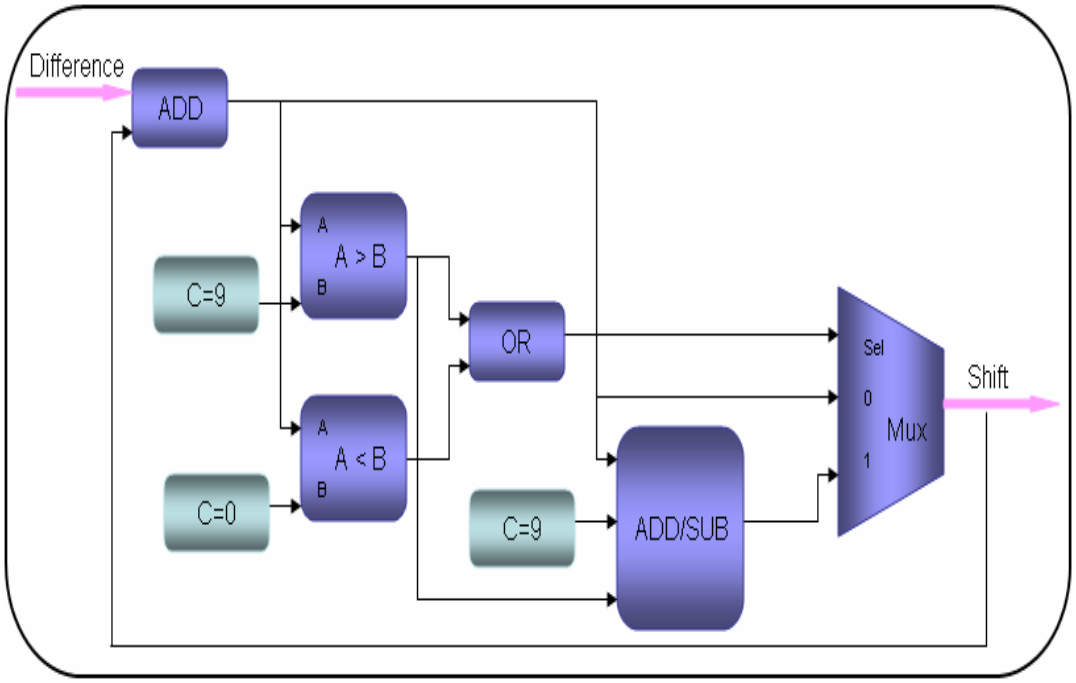

Figure 3.60: Tracking Phase Lock Loop Model (2)

3.6.9 Phase Decoder

The phase decoder retains the phase values from the CORDIC only at the maximum peak. As shown in Figure 3.61, the phase values for QPSK for bits 00, 10, 11, and 01 correspond to phase values of π/4, 3π/4, 5π/4, and 7π/4 in an ideal environment.

The CORDIC algorithm converges only for angles between - π/2 and π/2. Therefore, if *x*< *zero*, the input vector is reflected to the 1st or 3rd quadrant by making the x-coordinate non-negative [22]. As a result, in simulation the phase values for a one in the I channel is ±π/4 and ±3π/4 for a zero. The phase values for a one in the Q channel is -π/4 or -3π/4 and +π/4 or +3π/4 for a zero. Therefore, the bits 00, 11, 10, and 01 correspond to phase value 3π/4, -π/4, π/4, and -3π/4. Since the cosine of a positive and negative number is always positive, the sign of the phase value doesn't factor when determining the bit value for the I channel. Therefore, the phase boundary will be 0. On the other hand, the sign value is significant when taking the sine of the phase value. Therefore, the boundary will be π/2, halfway between π/4 and 3π/4, and compared to the absolute value of the phase when determining the bit value for the Q channel. Figure 3.62 illustrates the logic described above to determine the phase value.

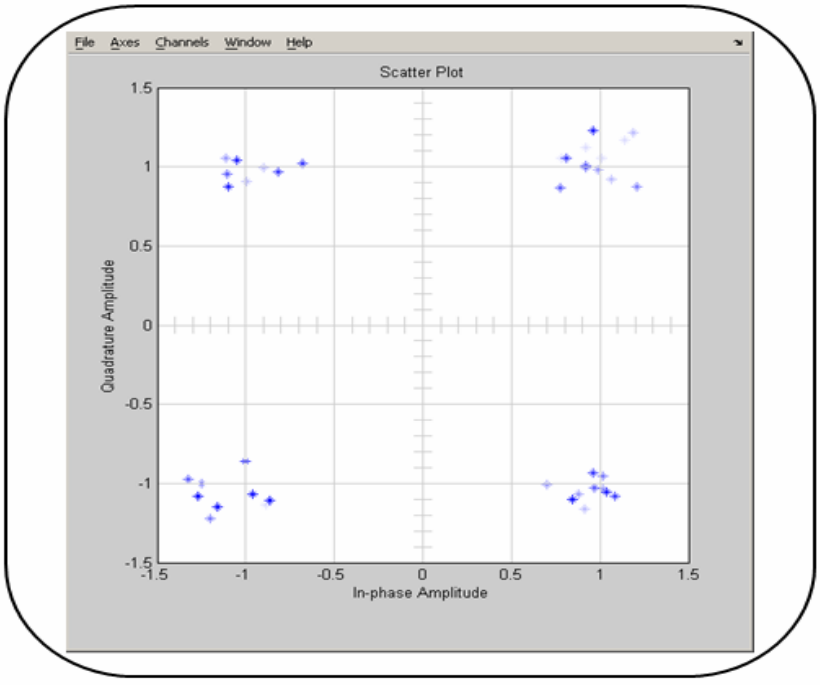

Figure 3.61: QPSK Phase Values

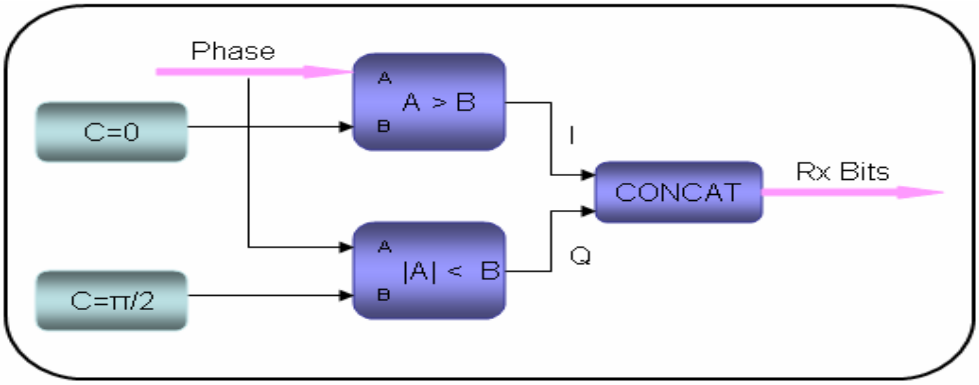

Figure 3.62: Phase Decoder Model

3.6.10 Packet Processor

To obtain accurate bit error rate results, the received and transmitted signal needs only to be compared for the actual random data and not during the preamble of DDW. Therefore, an enable signal must be constructed to indicate where the actual random data starts. This is the function of the packet processor. Since both the I and Q data bits are equivalent for the duration of the DDW, only one of them is needed.

In the DDW block shown in Figure 3.63, the incoming packet is first, up sampled by 63, just as it was done for the original spreading. The data stream is correlated with the original 63-length PN sequence used to create the DDW in the transmitter. The output is accumulated and stored in a register. The result is down sampled by 63 to bring the sampling period back to the original value.

Figure 3.64 represents the complete packet processor design. Perfect correlation results in positive and negative peaks of values 0 or 63. The correlation peaks value is compared to the values of 3 and 60 to provide a little leeway. Perfect synchronization provides an index for where the DDW has ended and random data is started. When this occurs, the data enable signal is set high to indicate where the actual data starts within the packet. This signal remains high for the duration of the data, 384 bits, and then goes low during the preamble and DDW. The simulation results for the data enable signal from the packet processor are shown in Figure 3.65.

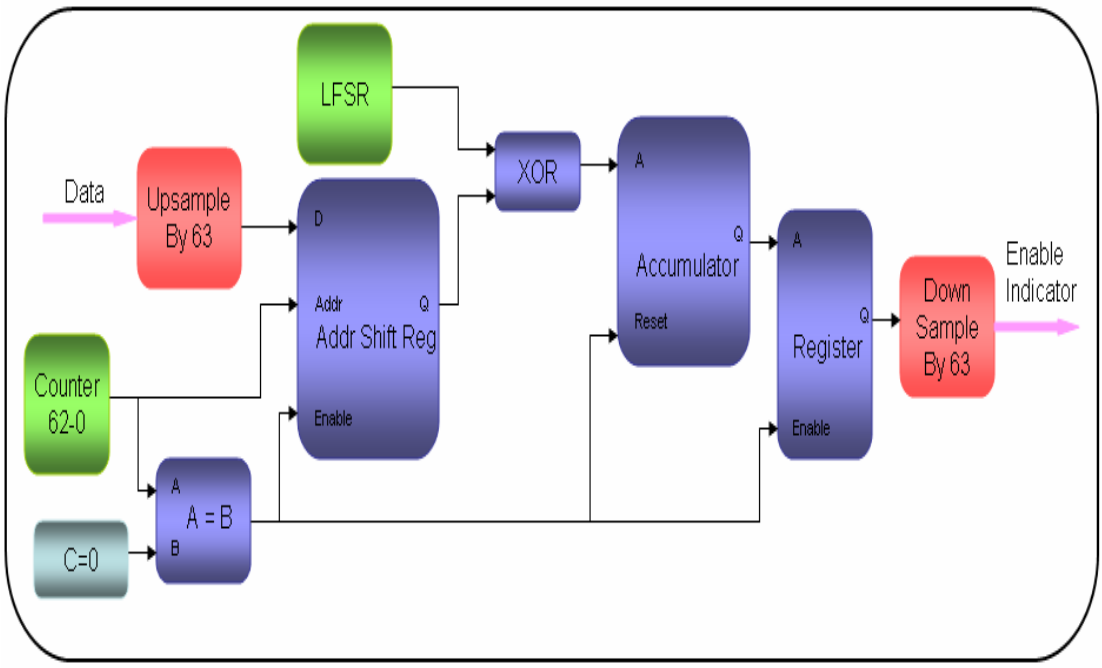

Figure 3.63: DDW Block within the Packet Processor

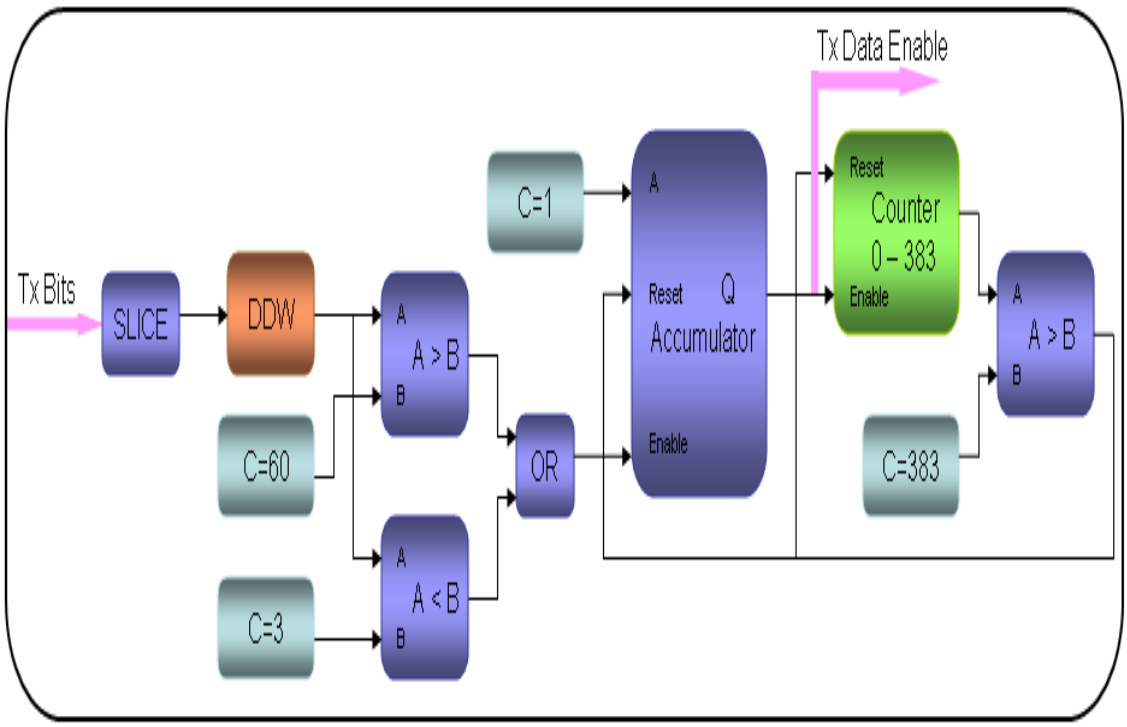

Figure 3.64: Packet Processor

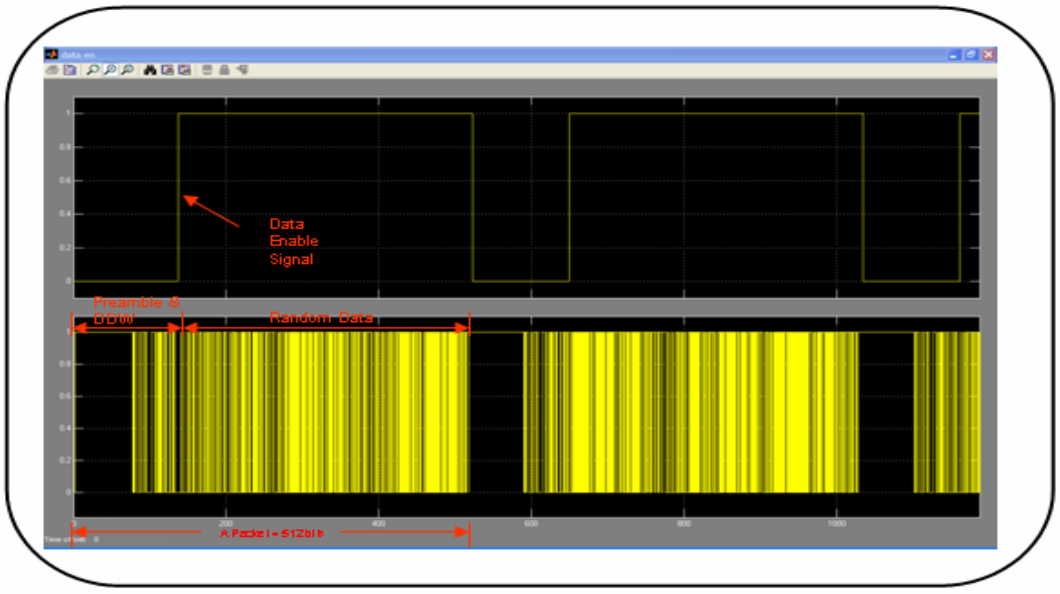

Figure 3.65: Data Enable Signal and Packeted Data

3.7 Calculating BER

BER calculations are performed for various SNR levels to see if they match theoretical calculations. The results conclude that errors occur uniformly across any data packet, independent of packet size, and that there are no correlations evident between the positions of errors within the frame. This implies that errors are highly localized within a frame and the error- inducing events occur over small (bit-time) time scales.

3.7.1 Bit Counter

In Figure 3.66, the bit counter is created to provide a reference point for the errors. In this design the bit counter is adjusted for a one million bits simulation. For BER results to be accurate, the simulation must be run for at least one million bits. The bit counter is enabled only when *Tx Data Enable* signal is high and therefore only accumulates the data bits and not the preamble or DDW bits. When the counter reaches a million, the *Bit Reset* signal is set to high to indicate the error counter to stop counting errors. Also, a LED is blinked to notify the user that the simulation is complete and error result can be recorded.

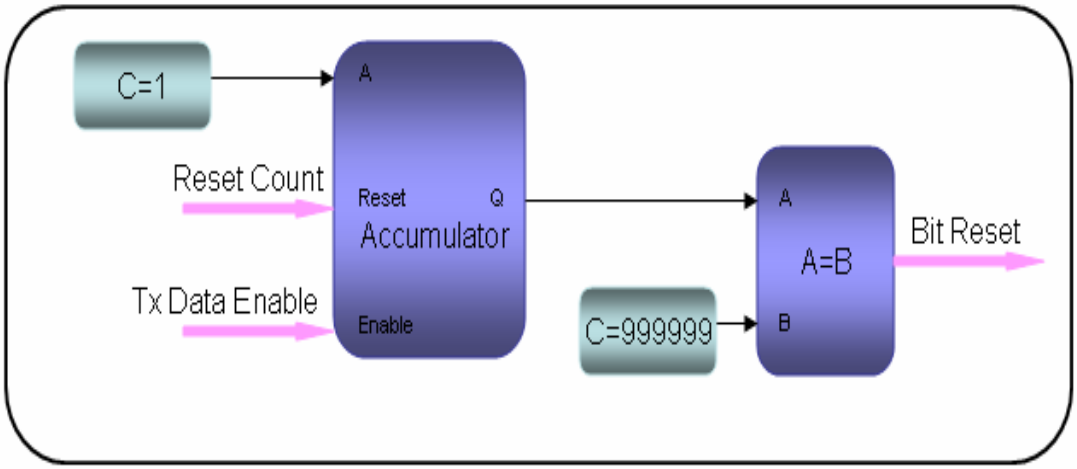

Figure 3.66: Bit Counter

The accumulator is reset by the *Reset Count* signal, which is controlled by the user through a pin on the board that is allocated in a *gateway in* block. Further explanation of this user defined reset signal is in Chapter 5.

3.7.2 Error Counter

The error counter, illustrated in Figure 3.67, compares the transmitted and received data to see if any errors have occurred due to the noise added in the channel. Since there is a delay between the transmitter and receiver, both the *Tx Bit* and *Rx Bit* signals must be adjusted before comparison. First, both signals must be down sampled by two to bring the sampling period back to 63. Also, the *Tx Bit* and *Tx Data Enable* signals must be delayed so that they line up correctly with the received signal. The matching of these three signals is essential for obtaining correct BER curves. A slight offset of the *Tx Data Enable* signal can result in comparison of the wrong portions of the transmitted and received packets.

Just as in the bit counter, the transmitted and received bits are compared only when the *Tx Data Enable* signal is high. This implies that only the data portions of the packet are compared for errors. After the bit counter reaches a million bits, the error counter holds its value. Since the error counter is specified to have ten output bits, a *gateway out* is assigned to ten pins. These pins are wired to LEDs that indicate the error value. The error counter is reset when the bit counter is reset. Details on the display board created to view BER results and reset the simulation are provided in Chapter 5.

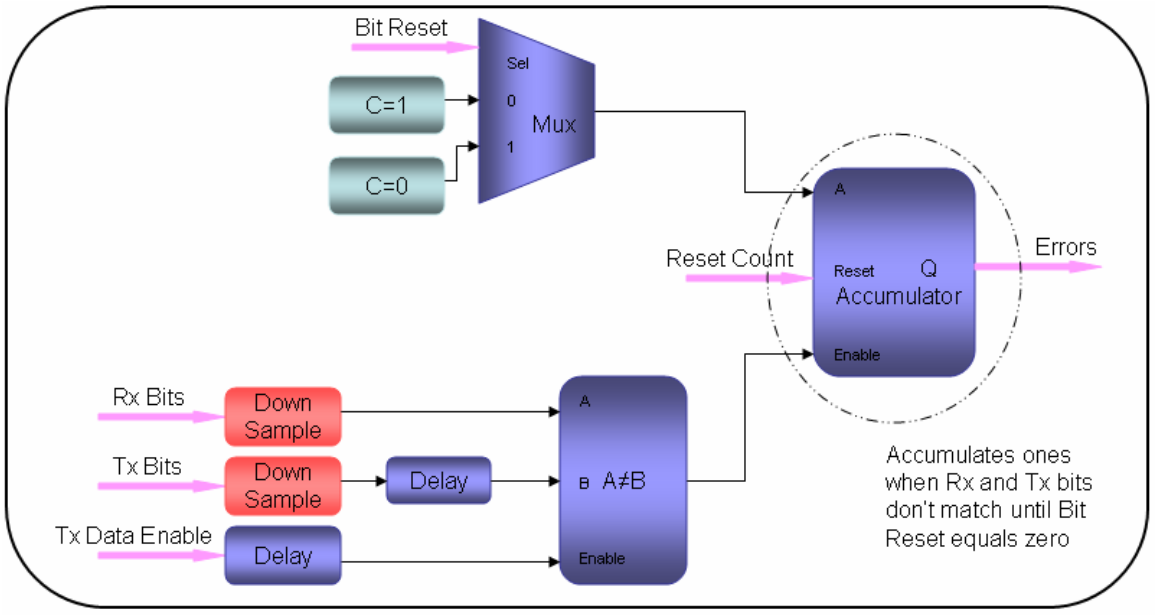

Figure 3.67: Error Counter

The performance evaluation of Quadature phase-shift-keying communication systems have been analyzed in a great variety of papers. Quaternary phase-shift-keying (QPSK or 4-PSK) systems have the greatest practical interest of all no binary (multiposition) systems of digital transmission of messages by phase-modulated signals.

3.8 Optimization of System Model

Since the system model of this communication system is quite large, it will consume a significant area on the hardware. The resource estimator block in System Generator gives an estimate of the area usage. Further details of this block are provided in Section 4.3.14 of Chapter 4.

The area usage can be decreased by optimizing the sections of the system model that take the most area. This implies that the receiver section should be optimized with consideration to the ADC, DDC, filter, parallel correlator, and CORDIC blocks. Therefore, 6 sections of the receiver are altered for optimization, as shown in Figure 3.68. The signal is optimized before analog to digital conversion, after digital down conversion, after the two running average filters, after the carrier loop, after the parallel correlators, and after the CORDIC.

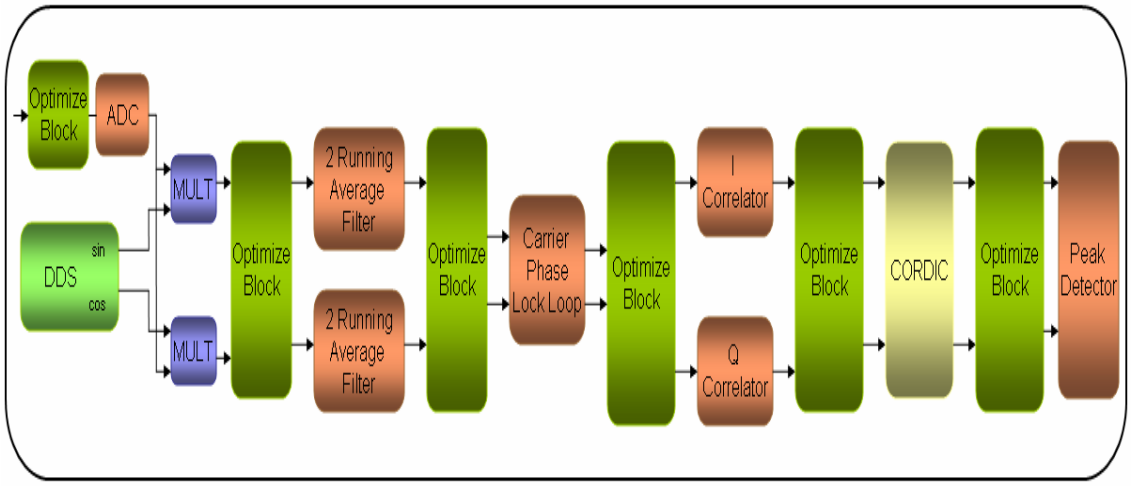

Figure 3.68: Location of Optimization Points in Receiver

Optimization of the system is dependent on the datapath size of the signal at various points in the receiver. Minimizing the datapath size decreases the area the system model requires on hardware, but also increases the BER of the system. Therefore, a relationship must be established between these two parameters to achieve an optimized design. As a result, the BER of the system is recorded for varied word lengths of the input signal of the optimization block. To evaluate the significance of each section of the receiver, all other signals are kept at full precision in other areas of the receiver for each simulation.

A multiplexer is used to select the various datapath sizes, varying from the input sequence having all '*A*' bits to the input sequence being composed of just two bits. In Figure 3.69, the select signal to the multiplexer is controlled by the user through a *gateway in* block. This block is allocated to pins corresponding to the hardware platform. This scheme is similar to the one used in the channel to create varying noise levels. Further explanations of this block are provided in Chapter 4. For the first iteration, all the bits of the input sequence with word length '*A*' are used to evaluate the BER. For the next iteration, one bit of the input signal is sliced off and instead inserted with a zero. The extra zeros are inserted to bring the signal back to its original length of '*A*' bits. This methodology is chosen because a multiplexer requires all input signals to be of same length. For the next iteration, two bits are sliced off and two zeros are inserted at the end of the signal. The user continues performing these iterations until just two bits are left in the input signal.

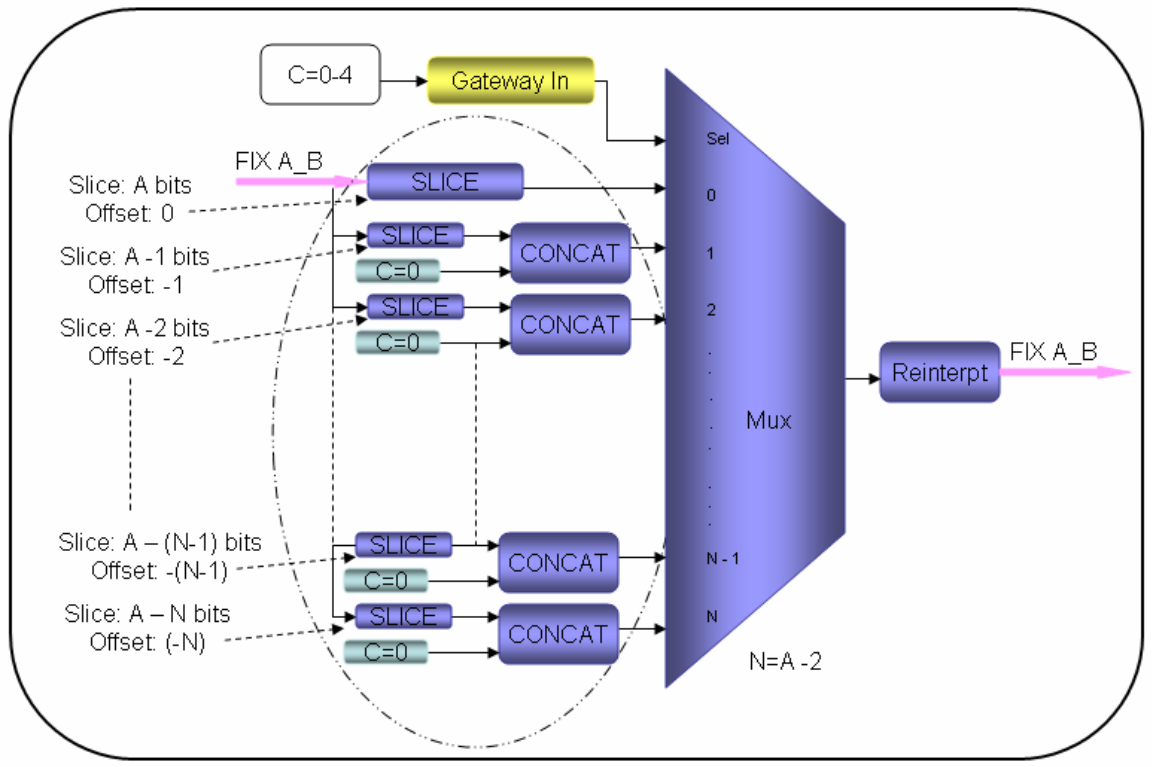

Figure 3.59: Optimization Block

The results are recorded and the input signal to that section is brought back to the original precision. The next area of the receiver is chosen for optimization and the same methodology is applied until all six sections of the receiver have been optimized individually. By using these results, several minimum area solutions are constructed to find an optimum design whose performance maintains certain specifications.

Since optimizing the system model is the ultimate goal of this thesis, accurately modeling the DSSS transceiver in System Generator is very important. Therefore, each component of the transmitter, channel, and receiver is tested for functionality before obtaining any performance results. The signal output is viewed in simulation after each significant block of the system model is incorporated in the design. Each block is finely tuned to obtain desired results by changing its parameters in the software. These parameters are discussed in Chapter 4. The signal outputs of the system model are tested in hardware to verify compliance to simulation results. Finally, block parameters are changed in the software environment to obtain an optimized DSSS transceiver design.

Chapter 4
DSP Analysis and System Generator

Over the years, the trend in the industry has migrated towards platform chips (FPGAs, DSP) to reduce costs. Designers are leaning towards implementing system on chips for increased design flexibility. As a result, challenges in modeling and implementing an entire platform arise. Therefore, DSP aspects of the system model of the transmitter, channel, and receiver used to describe the communication system are very important. This chapter will explain the different types of design flows available for DSP and the steps required in implementing them.

The overall approach to simulating a communication system is to create a system model consisting of functional blocks from a set library, which are interconnected to produce particular results. The parameters of each block are specified before execution based on system specification. Since the system design, described in chapter 3, is modeled with a software package called System Generator, this chapter will provide background on its features and highlight key facets of the software structure. Also, HDL co-simulation which supports legacy code will be discussed in view of the fact that having the ability to include legacy code is essential for many DSP system designers. Finally, details on implementation of hardware-in-the-loop (HIL) simulations will be provided since HIL is used for fast design verification.

4.1 DSP Design Flow

DSP design flow consists of the following steps: DSP Systems Modeling, System Generation, HDL Synthesis, Simulation/Verification, FPGA Implementation and In-System Debug. Figure 4.1 illustrates the steps for DSP design flow. Although, the Xilinx System Generator for DSP software platform is a critical component of DSP design flow, other tools are necessary to enable simulation, translation, and verification. The additional software may be a combination of Xilinx XST, Synplify Pro from Synplicity, Leonardo Spectrum from Mentor Graphics, ModelSim from Mentor Graphics, Xilinx MXE, Xilinx ISE, and Xilinx ChipScope Pro.

- DSP System Modeling: By using familiar tools like MATLAB and Simulink, users can develop models of their DSP systems. System Generator includes a Xilinx blockset that

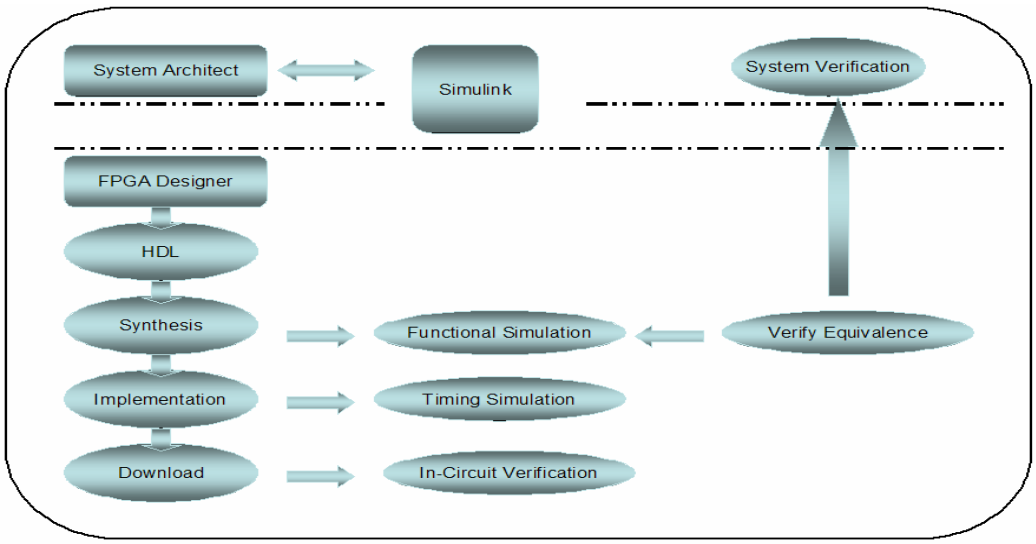

Figure 4.1: Traditional Simulink FPGA Flow

comprises basic level building blocks like FFTs, and advanced DSP algorithms like digital down converters. Users can also bring in their own HDL Modules via HDL co-simulation, or write MATLAB code for combinational control logic or state machine.

- System Generation: System Generation for DSP is invoked from Simulink through the System Generator for DSP token. This token generates VHDL and cores for all the Xilinx Blocks on the sheet containing the token, and on any sheets beneath it in the design hierarchy. FPGA designs are generated using Xilinx optimized LogiCOREs, ensuring that the most efficient implementation is being produced.
- HDL Synthesis: Once VHDL has been generated by System Generator for DSP, users may want to synthesize this for optimal FPGA implementations whether it is for high performance or optimal area. Users can choose from one of three popular synthesis engines including Xilinx's XST, Synplify Pro from Synplicity and FPGA Advantage from Mentor Graphics.
- Simulation/Verification: A VHDL testbench and data vectors can also be created by System Generator for DSP. These vectors represent the inputs and expected outputs seen in the Simulink simulation, and allow the designer to easily see any discrepancies between the Simulink and VHDL simulation results. FPGA Advantage can be used to conduct simulations of DSP systems prior to implementation. If doing HDL co-simulation, ModelSim is required.

- FPGA Implementation: Finally, designers can use ISE implementation tools to place route and verify the design in a FPGA. ISE allows users to use VHDL and design schematic entry tools to perform behavioral and timing simulations.
- In-System Debug: Hardware Co-Simulation capability can be used to accelerate simulation and verify the design in hardware. Including ChipScope Pro to your design flow will allow real-time debugging at system speed [32].

4.1.1 Types of Design Flows

The three types of design flows for DSP are VHDL-based designs, CORE Generator based designs, and System Generator based designs. All three have various advantages and disadvantages. VHDL-based designs offer portability, complete control of the design implementations and tradeoffs, and easy debugging. However, they are time consuming and require users to have familiarity of the algorithm and how it is written. CORE Generator based designs provide quick access to existing functions and optimized IP for the specified design. Nonetheless, they might not have the exact functionality. System Generator based designs are very attractive for FPGA novices. They offer high productivity and ability to simulate at a system level. The hardware in the loop simulation feature improves productivity and accelerates verification. The downfall of this type of design is that it doesn't always provide the best results from an area usage point of view. It is also not well suited for multiple clock designs. Due to its numerous advantages, the communication model design for implementing SDR is based on System Generator flow [22].

4.2 Simulink

Simulink is a platform for multi-domain simulation and model-based design for dynamic systems. The Simulink environment provides an alternative to using programming languages for system design. It is a software package that supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two. Simulink enables user to visualize the dynamic nature of the system by providing a graphical user interface (GUI) and a customizable set of block libraries. The designs follow a system of hierarchy allowing designers to build

systems using both the top-down approach and the bottom-up approach. Simulink illustrates the design in a realistic fashion with respect to the hardware design. Most hardware designs start out with a block diagram description and specifications of the system, very similar to the Simulink design. Unlike the sequential manner of software code, the Simulink model can be seen to be executing sections of a design at the same time [33]. This notion of parallelism is fundamental to implementing high-performance hardware implementations. System Generator, a system level modeling toolbox running under the Simulink environment, allows user to move from the software environment to hardware implantation with ease by providing high level abstractions that are automatically compiled into an FPGA.

4.3 System Generator

System Generator for DSP is fast becoming the preferred framework for developing and debugging high-performance DSP systems using the industry's most advanced FPGAs. It is a Xilinx software package that allows a designer to develop high performance DSP systems for Xilinx Virtex, Virtex-II, and Spartan-II FPGAs via MATLAB and Simulink. System Generator allows a designer to generate a system-level abstraction of FBGA circuits. These circuits are composed of common functions available in the Simulink library. The Simulink model automatically generates VHDL code that can be used with HDL testbenches. Figure 4.2 shows how System Generator interacts within the MATLAB environment. Simulink provides a block library that contains block sets used to model systems. System Generator provides an additional library that contains blocksets similar to those in the Simulink library. The advantage of this library is that these blocks can be implemented in hardware through the System Generator token. This token allows the generation on VHDL code, cores, and test vectors when HDL code generation is selected. FPGA implementation can be obtained by applying a synthesis tool, such as ISE, to the generated VHDL code. After mapping and place and route, ISE produces a bitstream that can be downloaded to a FPGA device using the FUSE software [23].

System Generator provides system-level designers with a portal into the FPGA, tapping into existing technologies to provide the foundations for system design for implementation. The following reasons illustrate that System Generator is an excellent choice for DSP platform designs.

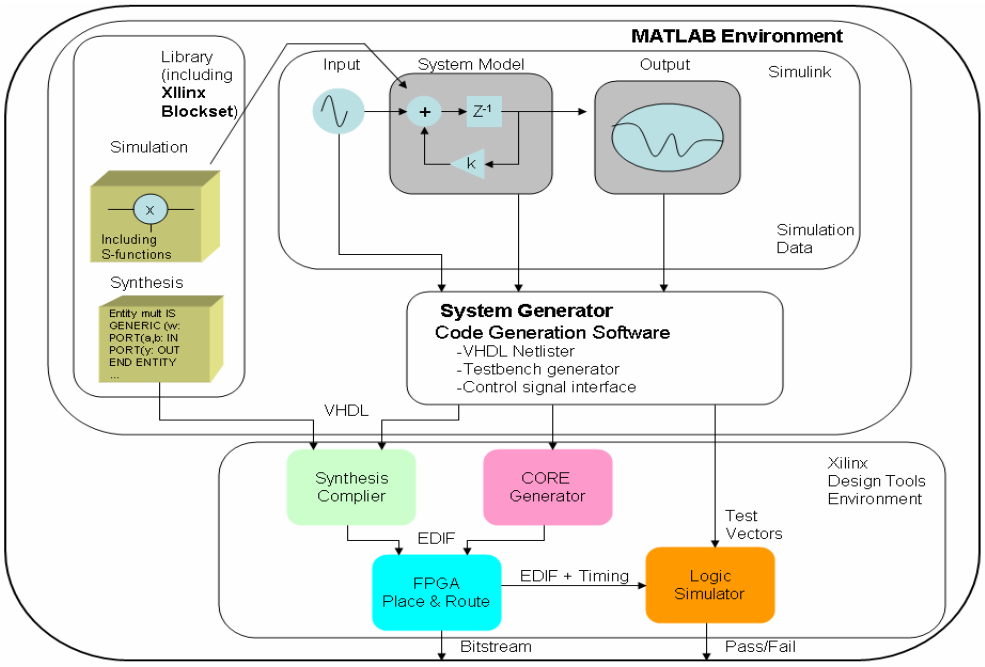

Figure 4.2: System Generator Flow Diagram

- Huge productivity gains through high-level modeling
- Ability to simulate the complete designs at system level
- Very attractive for FPGA novices
- Excellent capabilities for designing complex testbenches
- HDL Testbench, test vector, and data written automatically
- Hardware-in-the-Loop simulation improves productivity and provides quick verification of whether the system is functioning correctly.

4.3.1 Xilinx Blockset Library

The Xilinx Blockset is a Simulink library, accessible from the Simulink library browser. It consists of building blocks that can be instantiated within a Simulink model, and like other Simulink blocksets, elements can be combined to form subsystems and arbitrary hierarchies. The Xilinx Gateway blocks are used to interface between the Xilinx Blockset fixed-point data type and other Simulink blocks. Every Xilinx Blockset element can be configured via a parameterization GUI, with few exceptions even during simulation. Many blocks share common parameters, which are described later in this chapter. Most also have parameters specific to the function computed. The System Generator has the ability to generate an FPGA implementation

consisting of RTF VHDL and Xilinx Smart-IP Cores from a Simulink subsystem built from the Xilinx Blockset [23]. The overall design, including test environment, may consist of arbitrary Simulink blocks. However the portion of a Simulink model to be implemented in an FPGA must be built exclusively of Xilinx Blockset elements, with the exception of subsystems denoted as black boxes.

The following lists some of the most important blocks included in the Xilinx Blockset Library and where they can be found.

• *Basic Elements*
- System Generator
- Black Box
- Concat
- Constant
- Convert
- Counter
- Delay
- Down Sample
- Get Valid Bit
- Mux
- Register
- Set Valid Bit
- Slice
- Sync
- Up Sample
- Accumulator
- AddSub
- CMult
- Inverter
- Logical
- Mult
- Negate
- Relational
- Scale

• *DSP*
- DDS
- FFT
- FIR

• *Memory*
- Dual Port RAM
- FIFO
- ROM
- Single Port RAM

• *MATLAB I/O*
- Clear Quantization Error
- Display
- Enable Adapter
- Gateway In
- Gateway Out
- Quantization Error
- Sample Time

• *Math*
- Accumulator
- AddSub
- CMult
- Inverter
- Logical
- Mult
- Negate
- Relational
- Scale
- SineCosine
- Threshold

- Shift
- SineCosine

4.3.2 Bit True and Cycle True Representation

The Xilinx System Generator supports bit true and cycle true modeling of hardware. System Generator is bit true in the sense that signals in System Generator are represented as arbitrary precision fixed point data, which in VHDL corresponds to standard logic vectors. A fixed point value in a System Generator signal in Simulink consists of the same bits as its corresponding bits of the standard logic vector in VHDL. In addition to the fixed point value, every System Generator signal is sampled, and has an associated sample period. This implies that transitions occur only at multiples of the sample period for the block that drives the signal. In the VHDL generated by System Generator, the corresponding standard logic vector is driven by a block that is clocked (or if combinational, has an "inherited" clock period from its inputs) at a particular clock rate. The corresponding sample period in Simulink is guaranteed to be a multiple of the hardware clock period. At the clock transitions, this correspond to sample period multiples the bits in the standard logic vector (VHDL) match the fixed point data in the Simulink signal (software). This is an example of how System Generator is cycle true [23].

4.3.3 Hierarchy and Subsystems

All large designs will utilize hierarchy by implementing subsystems. This is a useful feature for maintaining the readability and reducing complexity of the design. Hierarchy in the VHDL code generated is determined by subsystems. When Simulink creates a subsystem, additional blocks known as in ports and out ports, are added as hierarchy connectors. Also, when analyzing a design in the Xilinx implementation tools, the name of a subsystem will be added to the component and signal names in that subsystem.

Subsystems can be personalized through masking. This allows the user to generate custom macro blocks with a custom icon, create a parameter dialog box for the block, shield complexity of the internal components of the block, and protect the contents of a block from being altered by unauthorized users.

4.3.4 Configuring Blocks

Most Xilinx blocks have parameters that can be configured. The typical element has a parameterization GUI with several common parameters (common to most blocks in the blockset) and some specific parameters (specific to the particular block only). Block parameters can be defined as equations which are calculated in the beginning of the simulation. This is useful when simulations must be done for varying parameter values. The configurable parameters of any block can be viewed or changed through block parameters. It is important to keep in mind that although all parameters can be simulated, not all are realizable. The following is a list of the configurable parameters [23].

- Arithmetic Type: Unsigned, Signed, Twos Complement, Boolean
- Latency: This parameter defines the number of input sample periods required for an input to affect a block output. Since System Generator doesn't perform extensive pipelining, additional latency is implemented as a shift register on the output of the block.
- Overflow and Quantization: Saturate, Wrap, Truncate, Round
- Precision: Full or User Defined with the number of bits and decimal point placement.
- Sample Period: Inherited with '-1' or User Defined with integer value to process data streams at a specific sample rate as they flow through the system.
- Override with doubles (Simulation only): With this option, the fixed-point simulation is bypassed and instead is executed in doubles. This is useful in examining the effects of quantization on the system design.
- Implement: With Xilinx Smart-IP Core or Generate Core

The simulation model of a functional block is a transformation of the form

$$\{y[k], y[k-1],....y[k-m] = F\{x[k-j], x[k-j-1],....x[k-j-n]; k; p_{1,} p_{2,} ...p_q\}, \qquad (4.1)$$

where *x[k]* represents input samples, *y[k]* represents output samples, p_1, p_2,.... p_q represents configurable parameters of the block, and $k = m, 2m, 3m,....$ is a time index. In such a representation, n samples of the input are used to generate m samples of the output of the model according to the transformation F, which is defined in terms of the input samples, the parameters of the block, and the time index k. The model is considered time-invariant if the transformation does not depend on the time index k. If $m = 0$ the block is evaluated on a sample-by sample basis.

If $n = 0$, the block is memoryless. The construction and execution of each block must take all such parameters in consideration [30]

4.3.5 Parametric Designs

Parametric designs provide flexibility on the IP within the block. Therefore, the same block can be used at different places in the design with different parameters. Not having to recreate the same blocks, saves the user valuable time. Also, it simplifies the comprehension of the design if parameters are used instead of numbers which other users may not understand. Parametric designs can be created by using functions such as get_param, set_param, find_system, add_block, delete_block, add_line, delete_line, etc [23]. Parameterization is useful in optimization of design which involves finding the optimum value of critical parameters such as number of quantization levels to be used in the receiver.

4.3.6 Quantization and Overflow

Most Xilinx blocks are polymorphic since they are able to deduce their output types based on their input types. When the full precision option is chosen, the block ensures that the output has no loss in precision. Therefore, sign extension and zero padding occur automatically when necessary. In Simulink the numbers are represented in double-precision whereas in Xilinx Blockset, the numbers are represented in fixed-point. Since resources are valuable and cost money in FPGAs, it is necessary to maximize the dynamic range of the design by using only the required number of bits. A user specified precision allows the user to set the output type for a block and to specify quantization handling and overflow. Figure 4.3 illustrates an example of overflow and quantization.

Quantization is a process of handling higher-precision number representation with a lower-precision number representation. Truncate and rounding are the two options available to handle it. An overflow occurs when a large number is represented in a smaller range representation. Saturate, Wrap the value, and Flag an error are the three options available to handle overflow. Efficient implementation of quantization and overflow is critical when datapath sizes are reduced for optimization purposes.

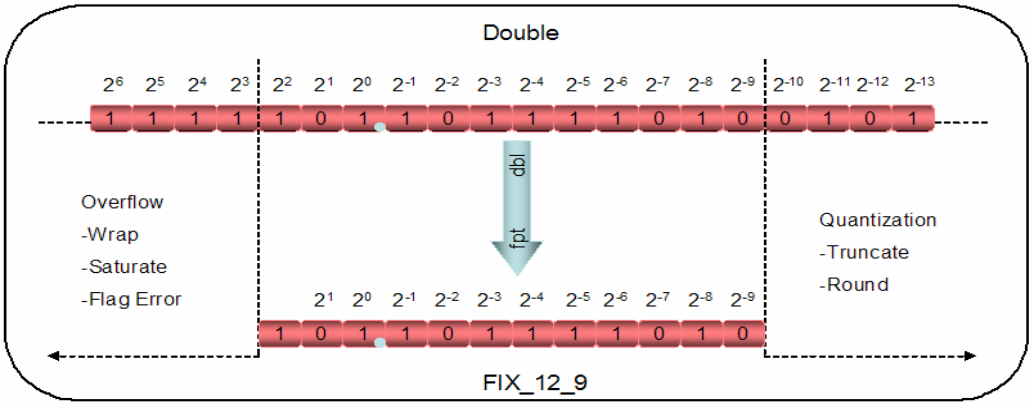

Figure 4.3: Quantization and Overflow Example

4.3.7 Bit Picking

Bit picking is necessary when there is need to combine two data buses together to form a new bus, force a conversion of data type including the number of bits and binary bits, reinterpret unsigned data as signed, or extract certain bits of data, especially when there is bit growth. The four Xilinx library blocks available for manipulation and re-interpretation of data are Concat, Convert, Reinterpret, and Slice. Understanding overflow and quantization is necessary when using the Convert block. Saturating the overflow may change the fractional number to get the saturated value, while rounding the quantization may affect the value to the left of the binary point [23].

4.3.8 Control Mechanism

The two mechanisms available in System Generator to control the data flow are enable and reset ports and valid and invalid data ports. An enable port, if available, ensures that the block holds its current state until it is asserted or the reset signal is asserted. The reset port, if available, is connected to a signal that places the block in its initial state when asserted. Both, enable and reset signals, must be of Boolean type and run at a multiple of the sample rate of the block. Invalid data ports may be required for data burst applications, one-shot FFTs, and latency output from high-level cores. Valid bit ports are used as control signals to the data input. They may be accompanied by a valid out signal that signals whether the output data is valid. Such ports may be necessary to ensure that a Xilinx block doesn't produce indeterminate data [23].

Several control blocks are provided to facilitate a high-level control mechanism and implement state machines. The Mcode block executes the supplied MATLAB M code to calculate the values the block delivers to a Simulink simulation. The expression block performs a bitwise logical expression. The Mealy State Machine block implements a state machine whose output depends on both the current state and the input, while the Moore State Machine block implements a state machine whose output only depends on the current state.

4.3.9 Sampling Period and Propagation Rules

System Generator designs are discrete time systems. Therefore, the data streams are processed at a specific sample rate, or clock period, as they flow through a dataflow system. A block's sample rate determines how often the block is updated. Typically, each block detects the input sample rate and produces the correct sample rate on its output. This "explicitly inherited" sample period tells Simulink to inherit the first encountered sample time. Feedback loops cause problems for Simulink's propagation algorithms. Therefore, the user must set at least one explicit sample time in every feedback loop. By selecting Explicit Sample Period rather than the default, the user can set the sample period required for all the block outputs, which supplies a hint to the feedback loop. The following blocks can change the sample period: Up Sample, Down Sample, Parallel to Serial and Serial to Parallel [23].

Increasing the sample rate (up sampling) by an integer factor *I* is called interpolation and decreasing the sample rate (down sampling) by an integer factor *D* is known as decimation. The up sample block either replicates the same number *M-1* times or inserts *M-1* zeros to achieve the higher sampling rate. The down sample block extracts *M-1* samples to achieve the lower sampling rate. Figure 4.4 shows the effects that up sampling and down sampling have on a signal [30].

When establishing a suitable sampling rate factors, such as aliasing error, frequency warping in filters, and bandwidth expansion due to nonlinearities, need to be taken into consideration. Increasing the sample rate can subdue these effects, but will also increase the computational load. As a result, there is a tradeoff between accuracy and simulation time.

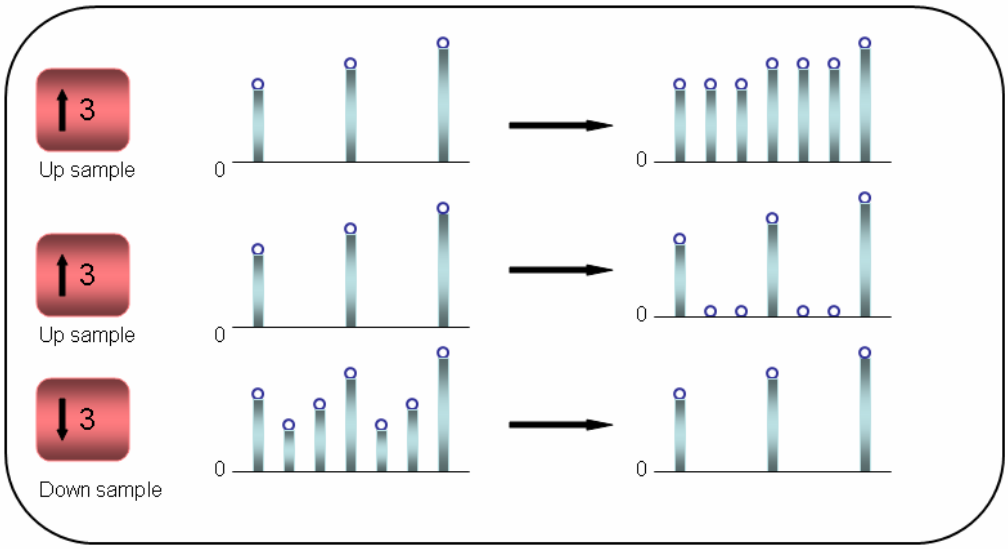

Figure 4.4: Up Sampling and Down Sampling Example

4.3.10 Multi-rate Systems and Sample Rate Conversion

DSP involves up conversion and down conversion of frequency and this can be associated by different sample rates. It is necessary to be able to convert between different sample rates in System Generator so that every subsystem is clocked only at a rate necessary to compute its input-output relation. This leads to more efficient use of resources and implementations that require less power. Significant resource savings can be accomplished by time-division multiplexing a data path to service multiple data streams that operate at a lower frequency than the processor.

In the digital domain, the change of the sample rate can be viewed as a linear filtering operation. If F_x is considered as the sampling rate of an input *x(n)* and F_y is the sampling rate of the output y(n), then the ratio of the sampling rates F_y/F_x must be rational [30]. Therefore,

$$\frac{F_y}{F_x} = \frac{I}{D}, \tag{4.2}$$

where *I* and *D* are relatively prime numbers [1]. The sample rate conversion can be explained as digital re-sampling of the same analog signal. If *x(t)* is the analog signal with sampled at a rate F_x to generate *x(n)* and *y(m)* is obtained from sampling *x(n)* at a sample rate F_y, then *y(m)* is a time-shifted version of *x(n)*. Figure 4.5 depicts this view of sample rate conversion. The time difference between the *x(n)* and *y(m)* sample is denoted as τ_i.

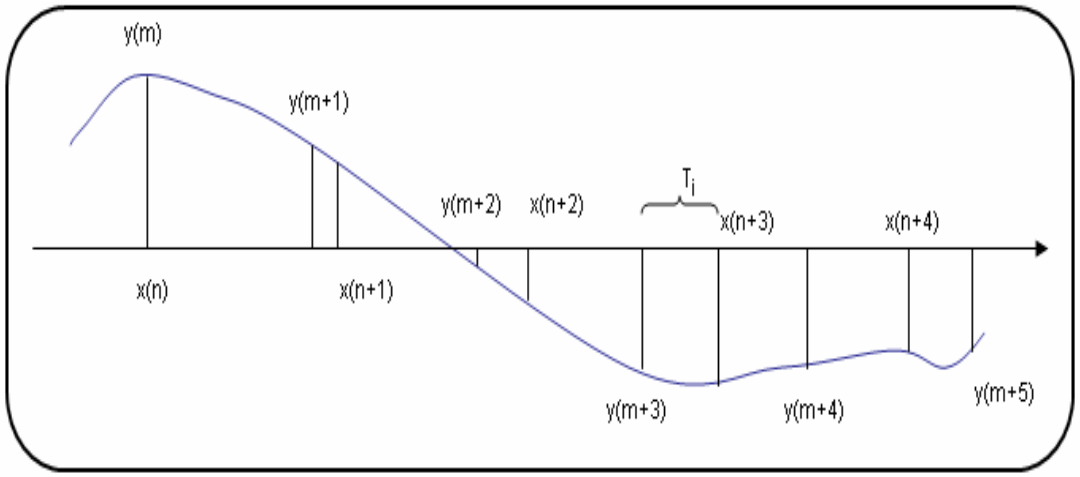

Figure 4.5: Sample Rate Conversion Example

4.3.11 Hardware Clock and Over-clocking

System Generator infers the clock from the design sample times, and abstracts away the clock enables. In multi-rate systems, further clock enables are inferred due to the more complex hardware elaboration scheme. Every block receives the same system clock signal, the fastest clock, but is enabled at its relative rate which is defined in the Simulink environment.

In hardware, the sample period acts as the number of clock pulses between clock executions. System Generator examines every sample time in the entire system and computes the greatest common divisor (GCD). The system clock corresponds to the GCD and each sample period is then normalized to a multiple of this value. System Generator circuitry that periodically asserts a clock enable (CE) for every required multiple is generated in a .vhd file. The entire system is referred to as a synchronous clock enable scheme. CE is modeled to reflect the hardware behavior, and therefore, the signal must come one clock cycle earlier. This is a familiar behavior to hardware designers, but is unusual for system-level designers. Figure 4.6 illustrates the behavior of CE. The CE pulses are referred to as the "Normalized Sample Times." Figure 4.6 shows that System Generator designs uses only one CLK and lower CLK speeds are derived from CE for different rate blocks. The *period* constraint is based on the system sample period and the FPGA system clock period specified in the System Generator token [23]. This type of clocking scheme requires implementation tools to be informed of the clocking speed of each flip-flop. Therefore, System Generator places timing constraints based on CE signals in a XCF file. Further details on timing constraints will be discussed in Chapter 5.

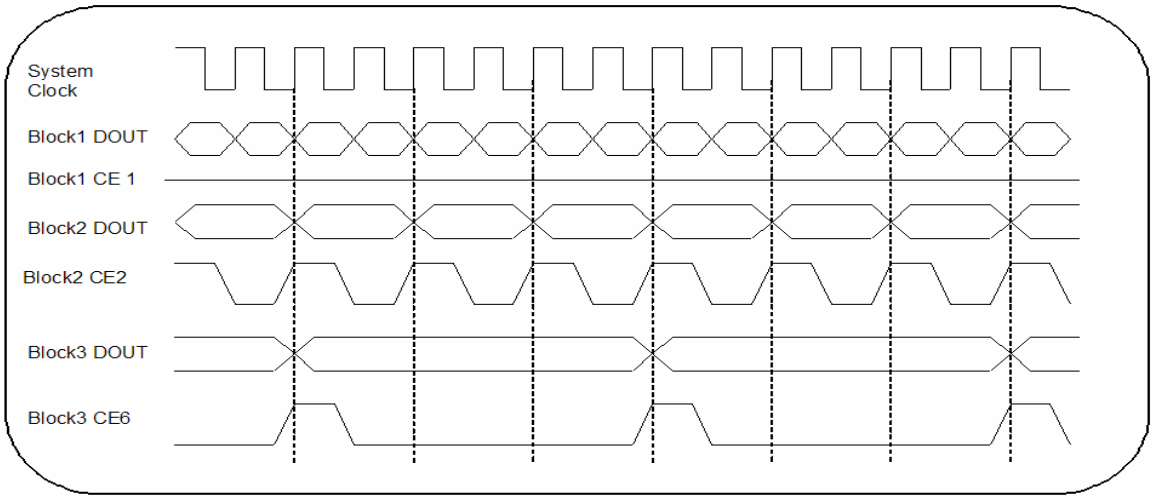

Figure 4.6: Clock Enable Behavior Example

Some System Generator blocks, such as a multiplier, require over-clocking. This implies that the block's internal processing is run at a faster rate than its data rates. In hardware, this signifies that more than one clock cycle is necessary for the block to process a data sample. Therefore, the internal processing of the over-clocked blocks need to be considered before specifying the Simulink sample period in the System Generator token.

4.3.12 Gateway In and Gateway Out Blocks

The Gateway In and Gateway Out blocks provide an interface between the Xilinx blocksets and the Simulink blocksets. They also act as input and output ports for the FPGA. The Gateway blocks also handle type conversions since MATLAB uses double precision floating-point and Xilinx uses fixed-point precision. This conversion from double to fixed point causes quantization effects.

4.3.13 System Generator Token

The System Generator token resides on the highest hierarchy level of the design. All System Generator designs must include a System Generator token. System Generator also offers the option of having numerous System Generator tokens in System Generator designs. This provides the software the ability to test lower levels of the design. For the simulation to work correctly, the Simulink System Period must be defined correctly. This value defines the smallest

period at which the system can run. All other sample periods are evaluated from this sample period. In hardware, this value equates to the System CLK that drives the design. As a result, the FPGA System CLK requires a value in nanoseconds to pass onto the timing constraints. These two fields define the scaling factor between time in Simulink simulation and time in actual hardware implementation. Also, they are necessary to achieve the desired timing performance in the place and implement part of the design. VHDL code can be generated by selecting HDL Netlist in the System Generator token. The target device is selected to correspond to the hardware board being used [23]. Figure 4.7 illustrates the parameters defined by the System Generator token.

A System Generator block that lies in the scope of another system generator block is called a slave. Otherwise it is called a master. Most system parameters can only be set in the master block. System generator will automatically synchronize slave blocks to the parameters specified in their master block. System parameters specified in the System Generator block affect the Simulink behavior, the hardware realization or the relation between the two for every block in the Xilinx blockset. Consequently, every element in the blockset must lie in the scope of a System Generator block. Therefore, every Simulink model that contains any element from the blockset must contain at least one System Generator block.

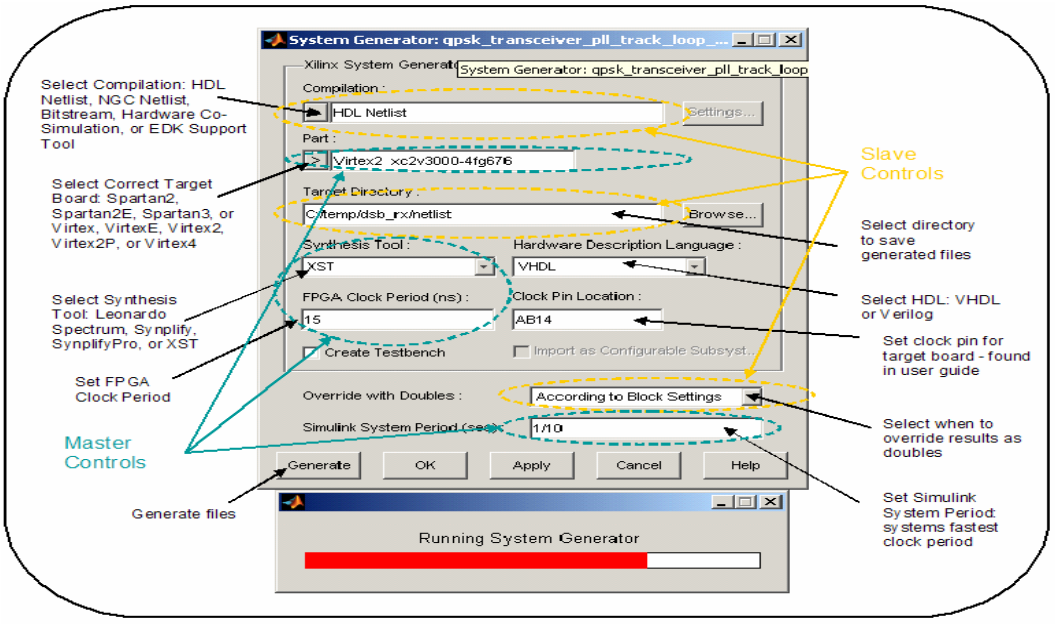

Figure 4.7: System Generator Token Block

System Generator can compile the model into various low level representations depending on the System Generator settings. The tool generates auxiliary files in addition to generating HDL files for hardware description. Some generated files are necessary for assisting downstream tools, while others are needed for design verification. The following describes a list of output files generated by System Generator.

- Design files: VHD (VHDL files); EDN (core implementation files); XCF (Xilinx timing constraint file)
- Project Files: NPL (Project Navigator project file); TCL (scripts for Synplify and Leonardo project creation
- Simulation Files: DO (simulation scripts for MTI); DAT (data files containing test bench for System Generator; VHD (simulation testbench)

4.3.14 Resource Estimator

Xilinx resource estimator provides fast estimates of the FPGA resources required to implement a System Generator model. The estimates are computed by invoking block-specific estimators for Xilinx blocks, and summing these values to obtain aggregated estimates of lookup tables (LUTs), flip flops (FFs), block memories (BRAM), 18 x 18 multipliers, three-state buffers, and I/Os. The Resource Estimator block provides three types of estimation: Estimate Area, Quick Sum, and Post-Map Area. The Estimate Area button invokes block estimation functions top-down for each block and subsystem recursively. The Quick Sum button causes the resource estimator block to sum all the FPGA Area fields on the block and subsystems at or below the current subsystem. No underlying estimation functions are invoked. The Post-Map Area button opens a file browser and lets the designer select the map report file. The design needs to be generated and implemented through synthesis, translate, and mapping phases prior to selecting this option.

4.4 HDL Co-Simulation

HDL co-simulation provides users a means to incorporate legacy code in a Simulink based system DSP design. Legacy code simulated in the Simulink tool can significantly reduce development time, resources, and cost because designers no longer need to write S-functions for Simulink. It also allows compilation of HDL designs into FPGA hardware which can be co-

simulated in the ModelSim environment. HDL co-simulation is supported by the following System Generator blocks: Black Box, Simulation Multiplexer, and ModelSim.

Figure 4.8 illustrates the required steps for HDL Co-simulation. A Black block is used as a way to incorporate non-Xilinx blockset functions into a System Generator model. This black box must be associated with a VHDL or Verilog file. Before compiling the design for co-simulation, a ModelSim block that represents the interface to the HDL co-simulator must be placed in the design. Finally, a testbench can be created to verify the functionality of the system in ModelSim through simulation.

HDL co-simulation of the design in FPGA hardware accelerates simulation speed since large HDL designs involve lengthy simulation times. The ModelSim co-simulation option allows the netlist portion of the design to be co-simulated in hardware along with traditional HDL components. A System Generator design can be compiled for ModelSim hardware co-simulation provided the resource requirements of the design do not exceed the available resources of the underlying hardware platform. The time matching between Simulink and ModelSim makes it easier for the designer to compare times at which events occur in the two settings. On a larger scale, it is useful since it allows System Generator to schedule events without running into issues related to timing characteristics of the HDL model.

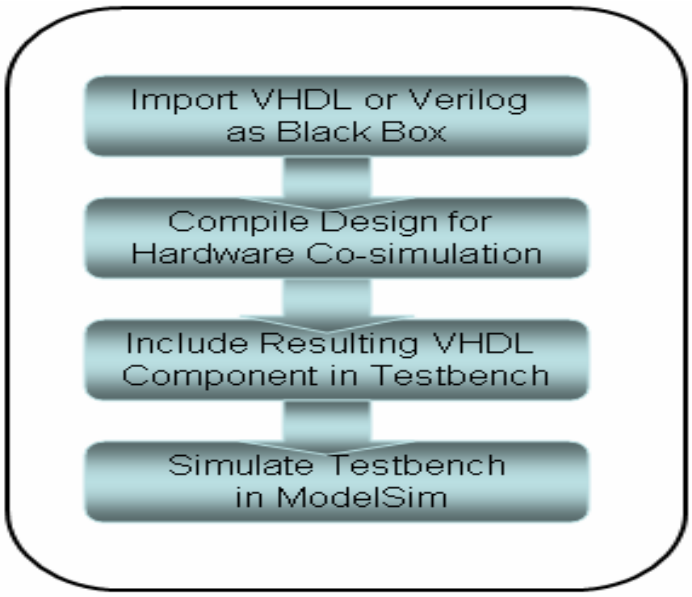

Figure 4.8: Required Steps for HDL Co-simulation through ModelSim

4.5 HIL Simulations

Hardware in the loop (HIL) is a Simulink hardware accelerator which enables design verification in hardware. It is a Simulink-to-bitstream-to-Simulink push-button flow to simulate HDL and EDIF-based designs. As a result, HIL simulations reduce design time and cost by allowing designers to verify designs in the hardware directly from the Simulink tool. Also, HIL simplifies hardware verification since it mirrors traditional DSP processor design flows and allows designers to accelerate the simulation when required, without the need of expensive emulation hardware, or long simulation times.

Hardware in the Loop simulations provides the following benefits when compared to emulated platform tests:

- Timing: Timing problems are not apparent in software simulations because they might not take into account the real time of code execution or data acquisition. Since HIL tests are performed in real time, any timing errors that occur will be apparently noticeable to the user.
- Concurrency: True concurrency of code execution on hardware cannot be simulated in pure software simulations. Therefore, problems such as hidden race conditions may go undetected and detailed event and state behaviors cannot be simulated.
- Hardware-specific Code: Since pure software simulations ignore hardware specific routines, any errors in the code can only be detected through actual hardware simulation.
- Communication Details: Communication protocols are simplified in simulators. Therefore, any errors in the code can only be detected through hardware simulation.
- Hardware Upgrades/Modifications: Simulators will not represent all aspects of the hardware. As a result, HIL tests are necessary to get accurate results when any hardware modifications or upgrades have been made.
- Performance Tuning: As a result of the above problems, performance of any application will be different in pure simulation from HIL testing. Therefore, the performance can only be adjusted when HIL testing is performed [22].

HIL simulations can be induced provided that the model meets the requirements of the underlying hardware specified by the System Generator token. This allows the Xilinx implementation tool flow to run in the background to create a BIT file and a library component. It also generates HDL and netlist files for the model and runs the downstream tools necessary for producing a FPGA configuration file. The configuration bitstream, shown in Figure 4.9, contains the necessary hardware for the model and interfacing logic to allow communication between the PC and the hardware platform through a physical interface such as a universal serial bus (USB). During simulation, a hardware co-simulation block interacts with the underlying FPGA platform, automating tasks such as device configuration, data transfers, and clocking.

The hardware generation produces a library block which must be dragged onto the design window and connected to all source and sink blocks before being simulated. The block assumes the external interface of the model and matches its port names to the port names on the original design. Therefore, the block produces the same type of signals as other System Generator blocks. The block can be driven by either Xilinx fixed-point signals or Simulink doubles. If Simulink doubles option is selected, quantization in the input is handled by rounding and overflow is handled by saturation. The data of the input ports is sent to its corresponding location in hardware when a value is written. Similarly, the output port retrieves data from the hardware when there is an event. The parameters of the block, such as hardware co-simulation clocking, need to be specified according to selected FPGA platform for implementation.

As, shown in Figure 4.10, HIL simulations can also be performed by generating a HDL netlist in the System Generator token for the design. The generated output files are used in an implementation tool such as ISE to create a BIT file for the design. This bit file is downloaded on the board through the software FUSE. For debugging and design verification, system design outputs can be viewed through HIL. To view results on devices such as logic analyzer, oscilloscope, frequency spectrum, etc. netlist generation through System Generator token and BIT file generation through ISE is necessary. Further details on the hardware aspects of HIL simulations and clocking synchronization are provided in Chapter 5.

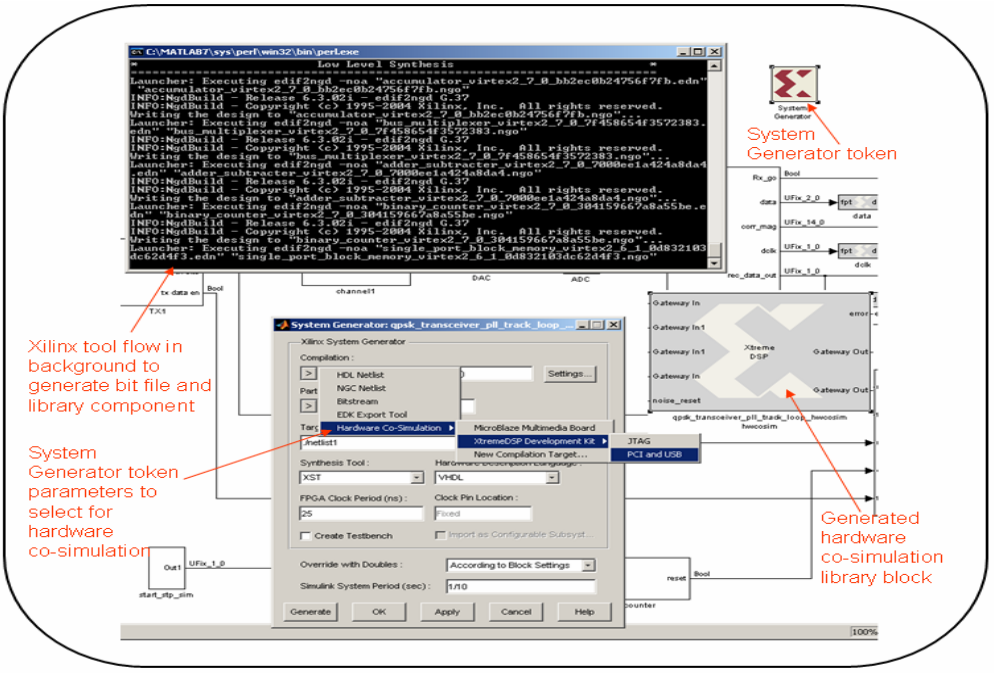

Figure 4.9: HIL Emulation

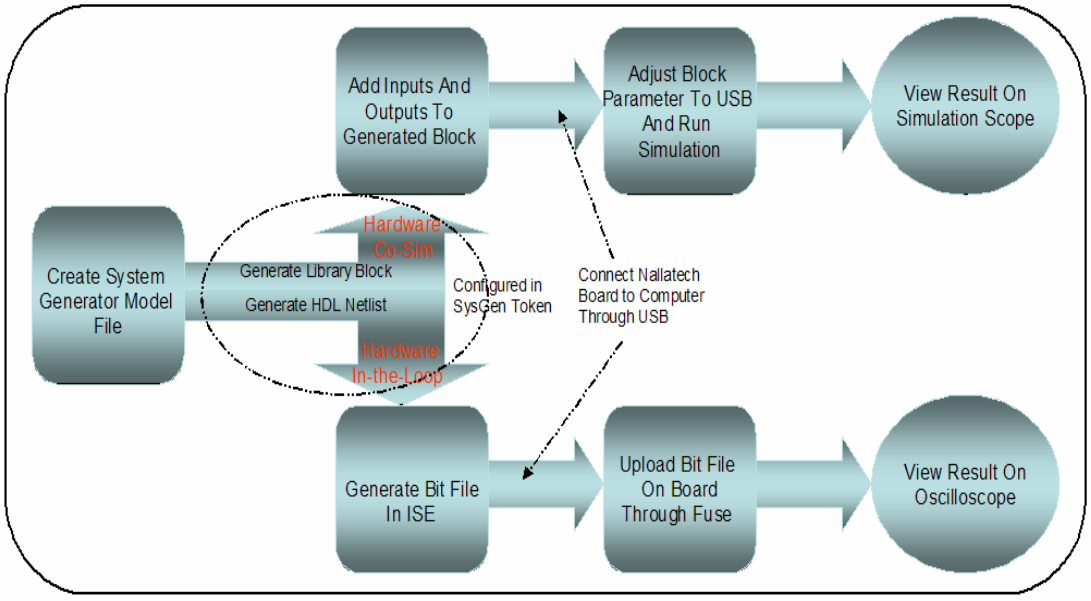

Figure 4.10: Steps for HIL and Hardware Co-Simulation

Xilinx System Generator tool ensures that the behavior of the design in hardware is guaranteed to be bit and cyclic true just as it is in the pure software environment. Xilinx's System Generator toolbox converts the system design into HDL code that can be placed on hardware. In co-simulation implementation, System Generator automatically produces a custom co-simulation library block that interfaces with the hardware when a simulation is run. System Generator also ensures that the appropriate FPGA configuration files are produced for the targeted hardware platform. The incorporation of hardware allows for increase in simulation speed, provides incremental hardware verification capabilities, and removes the difficulty of learning to program a FPGA.

Chapter 5
Hardware Implementation and Analysis

Since FPGAs play a critical role in enabling software defined radio technology cost-effectively in real-world applications, Xilinx, a worldwide expert in digital signal processing development solutions and leader in hardware-in-the-loop co-simulation, has developed a product to reduce time-to-market and development costs for designers of digital solutions. As described in Chapter 4, this tool automatically translates DSP systems developed using MATLAB and Simulink from The MathWorks into highly optimized VHDL and IP cores for Xilinx FPGAs such as the Virtex-II series and Spartan-3. Therefore, this chapter will provide extensive information on the features of the Virtex-II device used for hardware implementation. Next, this chapter will highlight the steps of Project Navigator, the software used to generate bit-streams directly from Simulink for FPGAs. Project Navigator is the user interface that helps designers manage the entire design process including design entry, simulation, synthesis, implementation and finally download the configuration of the FPGA device. Also, details will be provided on a software package called FUSE. It acts as a direct interface between the MATLAB/Simulink environment and a hardware platform, allowing users to directly view the data output from the IP cores in the MATLAB environment. Finally, the hardware architecture of the BER board constructed to evaluate bit error rate of the system will be explained. The functionality of this board is described in Chapter 4.

5.1 Parallelism

Conventional DSPs use a common architecture known as the Von Neumann architecture. This architecture's serial structure limits its performance. The MACs within conventional DSPs are typically shared resources. The increased number of MAC operations provides for more accurate results. FPGA implementation based on sequential MAC can be very efficient for low sample rates. To achieve higher sample rates, Xilinx uses parallel processing [34]. Figure 5.1 depicts the architecture difference in MACs between conventional DSPs and FPGAs.

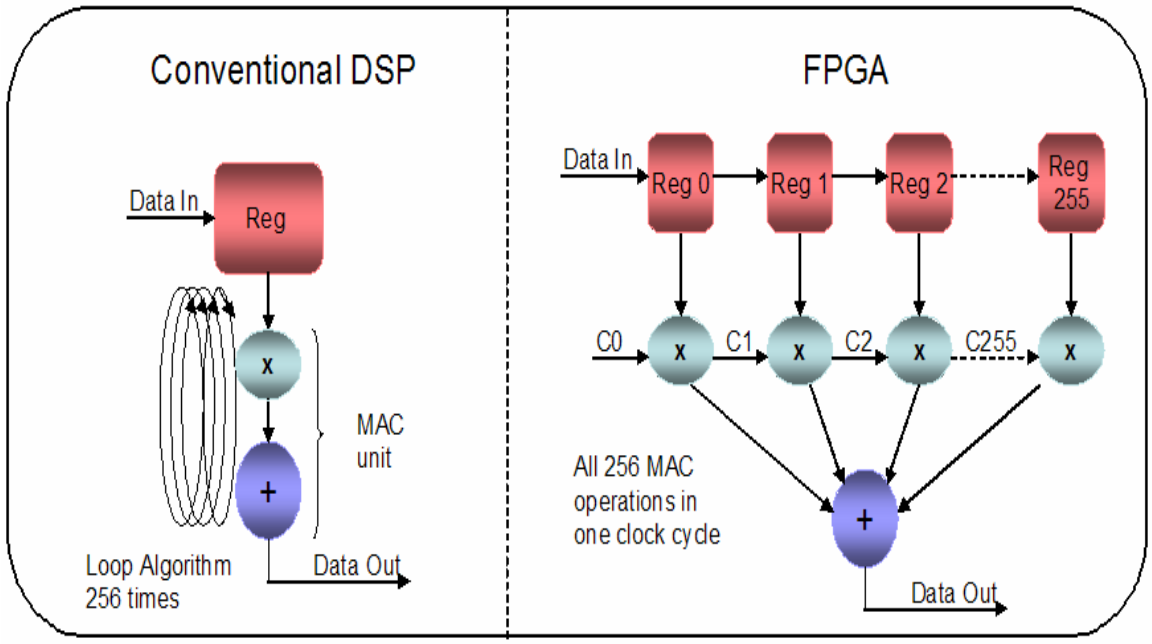

Figure 5.1: Architecture Difference in MACs between DSPs and FPGAs

Within a fixed MAC unit, the maximum sample rate is related to the algorithmic complexity as

$$Max\ sample\ rate = \frac{Fixed\ \ processor\ clock\ rate}{Number\ of\ \ opertions\ \ per\ sample}\ . \qquad (5.1)$$

According to this equation, the sample rate must decrease as the algorithm complexity increases and requires more clock cycles to process each sample. Using multiple processor engines is the only way to increase the algorithm complexity and the sample rate. Parallel processing maximizes data throughput and provides the optimal performance versus cost tradeoff [1].

The parallel architecture allows performance of FPGAs to reach up to 500-billion MACs per second in the largest Xilinx Virtex-II FPGA, which is significantly higher than the conventional DSPs. Figure 5.2 illustrates in detail the increase in performance of Xilinx FPGAs as compared to conventional DSPs.

Another advantage of FPGAs is that they provide flexibility in design for a wide spectrum of sample rates, from multi-cycle implementation to single cycle. This is highly crucial when designing a communication system, as illustrated by the system model in Chapter 3.

Function	Industry's fastest DSP Processor CORE	Virtex-II	Virtex-II PRO	Spartan - 3
8 x 8 MAC	5.7 billion MAC/s	0.5 Tera MAC/s	1 Tera MAC/s	0.27 Tera MAC/s
FIR Filter - 256 taps, linear phase - 16-bit data/coefficients	11.16 MSPS 720 MHZ	180 MSPS 180 MHZ	300 MSPS 300 MHZ	140 MSPS 140 MHZ
Complex FFT - 1024 point, 16-bit data	8.5 µs 720 MHZ	0.914 µs 140 MHZ	0.853 µs 150 MHZ	0.914 µs 140 MHZ

Figure 5.2: Performance Comparison of Xilinx FPGAs and Traditional DSPs

Source: [35]

5.2 Xilinx Xtreme DSP

A user programmable Virtex-II device and high performance ADCs and DACs made the Xtreme DSP Development Kit-II an ideal candidate for implementing signal processing applications such as Software Defined Radio. The Xtreme DSP board contains a motherboard and a module, which are referred as "BenONE-Kit Motherboard" and "BenADDA DIME-II Module". DIME is a modular standard for FPGAs that allows the system to be re-programmable and allows alteration of design partitioning at any time. The BenONE-Kit Motherboard contains the Spartan-II FPGA for 3.3V/5V PCI or USB interface, JTAG configuration headers and user pitch pin headers [35].

5.2.1 Physical Description

This device contains over two million system gates, enough to handle the types of complicated algorithms used in leading-edge digital communications and imaging solutions today. The board also offers flexible, high-speed, high-resolution data conversion for both baseband and direct IF applications, including:

- Two Analog Devices AD9772A digital-to-analog converters, operating at up to 160 MSPS, directly controlled by the on-board FPGA, allowing maximum operating flexibility.

- Two Analog Devices AD6644(5) analog-to-digital converters which interface directly to the on-board FPGA. The AD6644(5) is a high-speed, high-performance, monolithic 14-bit device operating at up to 65 MSPS.
- A dedicated PCI and USB interface, used for interfacing between the PC system and the user application running on the Virtex-II FPGA. This is complemented with drivers, (Windows 95/98/NT/2000 and Linux) which offer a complete foundation for system development.
- A dedicated clock management FPGA (Virtex-II), along with the on board oscillator and external clock input. This device provides source selection and routing of programmable system clocks for low jitter.

The hardware of this kit is contained in a blue case which provides EMI shielding and protection for the board. Figure 5.3 displays the front of the case and highlights the location of the following components: ADCs, DACs, LEDs, USB connection, power input, JTAG cable access, and fan vent. Configuration of large designs can significantly overheat the User FPGA running at full potential. Therefore, a temperature sensor is provided on the kit to monitor heat levels. Also, a fan is installed to provide cooling to the User FPGA. Figure 5.4 emphasize the key features of the motherboard: I/O headers, JTAG headers, DACs, ADCs, LEDs, ZBT memory, main user FPGA, interface FPGA, crystal oscillator, PCI connection, USB connection, and power connections. The clock FPGA is not visible in this figure since it is located on the underside of this module [35].

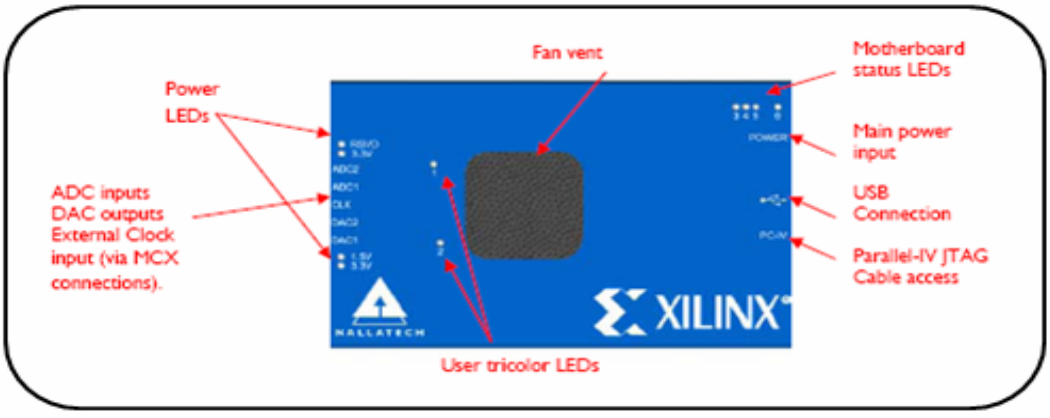

Figure 5.3: Front Case of XtremeDSP Board

Source: [35]

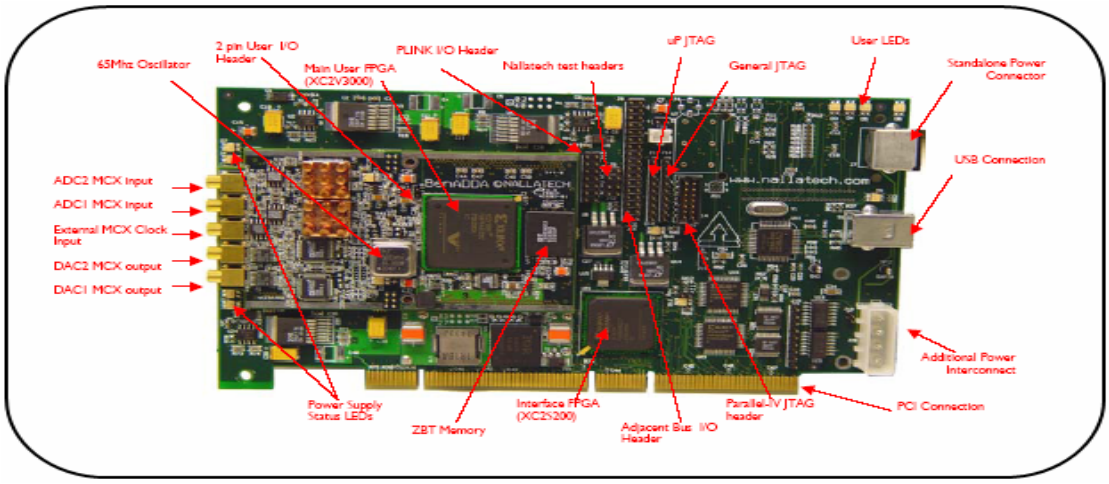

Figure 5.4: Key Features of the Motherboard

Source: [35]

5.2.2 Virtex-2 Architecture

The Virtex-II device is embedded with user-programmable gate arrays to optimize for high-density and high-performance logic designs. Figure 5.5 illustrates that the Virtex-II device is comprised of input/output blocks (IOBs) and internal configurable logic blocks (CLBs). Interfacing between package pins and the internal configurable logic is provided by programmable I/O blocks. The internal configurable logic consists of the following elements:

- CLBs are responsible for supplying functional elements for combinatorial and synchronous logic.
- Block Select RAM memory modules are equipped with large 18-Kbit storage elements of dual-port RAM.
- Multiplier blocks are 18-bit x 18-bit dedicated multipliers.
- Digital clock manager (DCM) blocks assign digital solution for clock distribution delay compensation, clock multiplication and division, and coarse and fine tuned clock phase shifting.

The CBL resources contain four slices and two 3-state buffers, where each slice contains two function generators, two storage elements, arithmetic logic gates, large multiplexers, wide function capability, fast carry look-ahead chain and horizontal cascade chains [36].

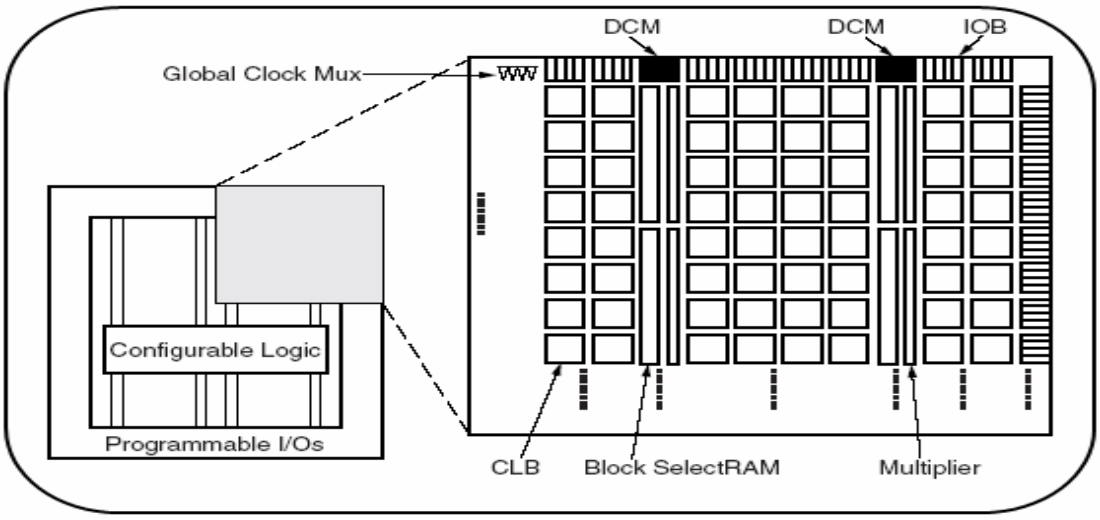

Figure 5.5: Virtex-II Architecture Overview

Source: [36]

5.2.3 XtremeDSP Kit Highlights

Creating high performance DSP designs requires a fast platform FPGA for design implementation, easily accessible software tools and IP, and a pre-engineered high-performance hardware platform for quick functionality verification. The XtremeDSP kit provides a complete development solution, allowing users to develop powerful DSP algorithms. The following reasons validate that XtremeDSP kit is a reliable candidate for creating DSP designs with exceptionally high performance.

- High Performance: The dual-channel high-performance ADCs and DACs, as well as the user-programmable Virtex-II FPGA, are ideal for implementing high-performance signal processing applications such as Software Defined Radio, 3G Wireless, networking, HDTV or video imaging.
- Scalability: The modular system is based on Nallatech's latest DIME-II technology and is an ideal stepping stone if one wants to scale-up later for more demanding application requirements. Nallatech offers unparalleled off-module I/O capabilities and flexible FPGA device support, coupled with extreme bandwidth capabilities for next generation systems design.
- Flexibility: Communication and control of the Xtreme DSP demo board is provided via a PCI interface for embedded environments, or via a USB interface for stand alone

applications. The board also includes multiple clock drivers including an external clock, an on board oscillator, and a programmable clock.

- Ease of Use: Provides simple and well-integrated design flow from algorithm concept to hardware verification. The Xilinx System Generator for DSP interfaces with MATLAB/Simulink and a large selection of intellectual property (IP) from Xilinx, allowing users to solve complex DSP design problems quickly. Also, Nallatech FUSE (Field Upgradeable Systems Environment) software is provided to control and configure the on-board FPGA, and allows the user to transfer data between the motherboard and a host PC.
- Time to Market Advantages: Increases speed in implementing a complete system for applications such as digital communications and image processing. Thus, the user can focus on the design without worrying about prototyping [3].

5.2.4 Clocking Configurations

The Xtreme DSP Development Kit-II has an intricate, yet flexible clock management system. The 65MHz oscillator provides a low jitter clock source for the analog devices. The kit also contains two soft programmable clock sources, which can be set to various frequencies. Figure 5.6 displays the location on hardware of the devices and inputs related to the clock sources and Figure 5.7 presents an overview of the clock structure of the kit.

The BenADDA module can use three system clocks (CLKA, CLKB, and CLKC) fed from the motherboard to the user FPGA. The DIME-II motherboard generates these signals and routes them to the modules for placement. These clocks can be controlled by the user and are routed to Global Clock pins. The Fuse software controls the programmable oscillators, which only operate at the following frequencies: 20 MHz; 25 MHz; 30 MHz; 33.33 MHz; 40 MHz; 45 MHz; 50 MHz; 60 MHz; 66.66 MHz; 70 MHz; 75 MHz; 80 MHz; 90MHz; 100 MHz; 120 MHz. The firmware selects the numerically closest available frequency when the requesting frequency doesn't match one of the above frequencies [35].

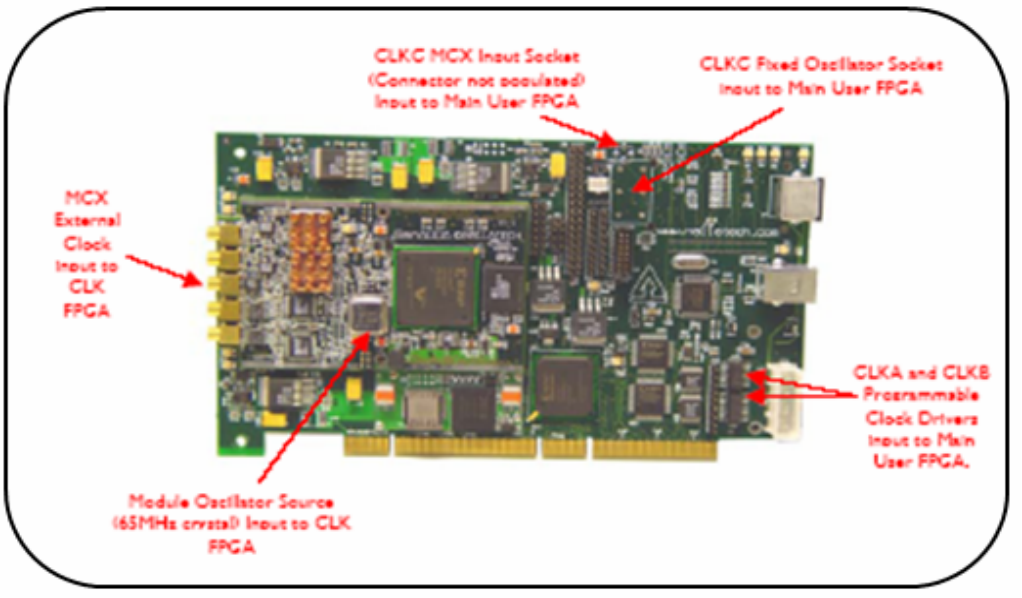

Figure 5.6: Inputs Related to Clock Sources on Hardware

Source: [35]

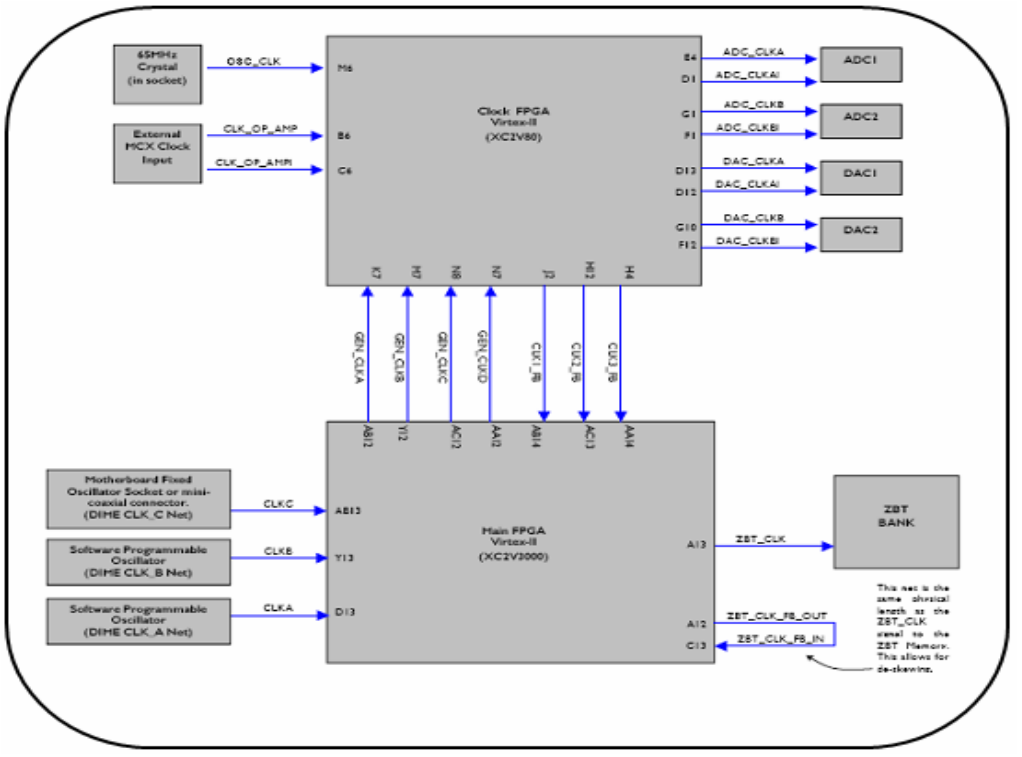

Figure 5.7: Overview of Clock Structure

Source: [35]

5.2.5 ADCs and DACs

The BenADDA module used in the XtremeDSP Development Kit-II has two analogue input channels, with each channel providing independent data and control signals to the FPGA. Two sets of 14-bit wide data are fed from two ADCs (AD6644) devices to two DACs (AD9772A devices), each of which has an isolated supply and ground plane. The14-bit ADC resolution is represented in 2's complement format. The ADCs can handle up to 65MSPS sampling data rate and are clocked differentially. The inputs to the ADC devices are connected via MCX connectors on the front of the module. The standard shipped configuration exhibits 50Ω single-ended inputs, each featuring a 3rd order anti-aliasing filter with a -3dB point at 34.5MHz. The ADC has a full range input specification of 2.2 V peak to peak (p-p). The recommended maximum signal magnitude at the MCX input to attain best performance characteristics is 2 V p-p or +/- 1 V [35].

The 14-bit DAC resolution in offset binary format can handle a maximum of 160 MSPS input data rate. The BenADDA is configured to have single ended DC coupled outputs from the DACs. Each DAC device is clocked directly by an independent differential, LVPECL signal. This LVPECL signal is driven from Virtex-II XC2V80 FPGA (Clock FPGA) which is solely dedicated to managing the various methods for clocking each ADC and DAC device. The way the DACs are clocked depends on the bitfile that is assigned to the dedicated Clock FPGA [35].

5.2.6 Digital I/O

Digital I/O is provided on the board for interfacing with other hardware or for debugging purposes through use of hardware such as logic analyzers. Digital IO is available on the board through the following:

- A 14-pin PLINK Bus header on the motherboard. This header contains 12 bi-directional pin connections to the main User FPGA, while the other two are used as GND connections.
- A 34 pin Adjacent Bus header on the motherboard. This header contains 28 bi-directional pin connections to the main User FPGA, while the remaining are reserved for a 3.3V connection, a GND, and 'no connects'(NC).
- A 2 pin user I/O header on the module. This header contains 2 bi-directional connections to the main User FPGA.

In system design, the digital I/O is configured according to the specific I/O of the FPGA. The datasheets of the I/O standards supported by pins on the Virtex-II device can be found in the Appendix. Each pin on the header corresponds to a particular pin number on the User FPGA. These pin numbers are needed when the user wishes to view a particular software output on hardware. These pin numbers are assigned in gateway out blocks as explained in Chapter 3 [35].

5.2.7 JTAG

The BenADDA module contains a JTAG based Plug and Play (PnP) facility to automatically detect modules already present in the system. The JTAG chain, which is used for test and configuration purposes, connects to the General JTAG header via the standard JTAG pins on the User FPGA. The General JTAG header supports flying lead connections for Xilinx Parallel-III or Parallel-IV pods. The Parallel-IV JTAG header is necessary when the Xilinx JTAG Co-simulation option is selected in Xilinx System Generator token [35].

5.3 ISE

Xilinx Integrated Software Environment (ISE) provides the user with a powerful and well integrated environment toolbox for the following steps of design flow: design entry, synthesis, verification, implementation and configuration and board level integration. Figure 5.8 depicts the FPGA design flow process.

Project Navigator is the primary user interface for Xilinx ISE, which allows users to create, define and compile a FPGA or CPLD design using a suite of tools accessible. Each step of the design process, from design entry to downloading the design to the device, is managed from Project Navigator as part of a project. The top-level source defines the inputs and outputs that will be mapped into the device, and references the logic descriptions contained in lower-level sources. A project must contain at least one source as the top-level source. All source files and their accompanying icon are displayed in the Sources in Project window below the project file. Figure 5.9 displays the Project Navigator window and highlights the key features involved for all the steps from design entry to configuration.

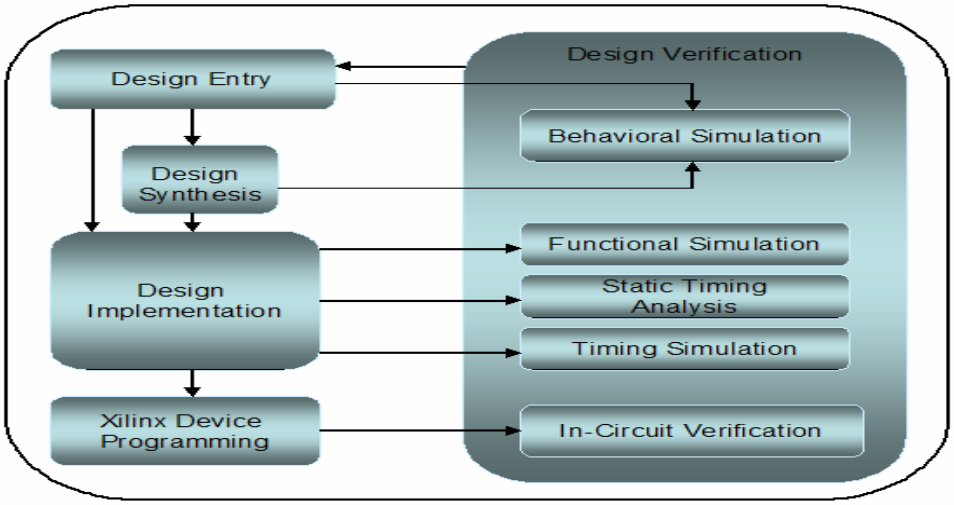

Figure 5.8: System Generator Based Design Flow

Source: [36]

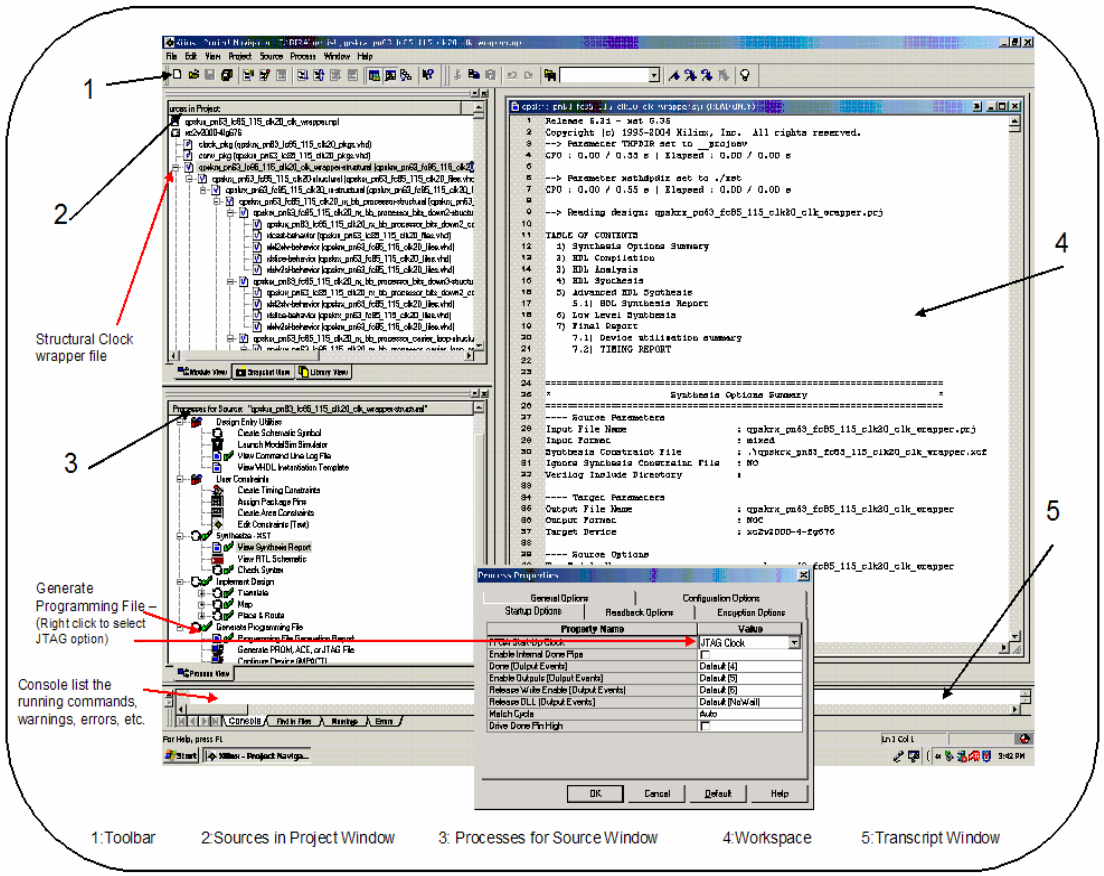

Figure 5.9: Project Navigator Window

5.3.1 FPGA Flow in ISE

ISE flow for FPGAs consists of three different types: push button flow, basic flow, and advanced flow. ISE is designed to provide a rapid design path, or "push button flow," for integrated circuit designs. These designs are usually smaller having fewer design elements and fewer timing constraints. It is often necessary to set design constraints, process properties and reiterate some of the steps in the flow in order to meet the timing requirements for the design. When the designs are usually of moderate complexity having more design elements and more timing constraints, it is defined as basic flow. ISE provides a suite of tools necessary to create very complex designs and ensure that the design will meet the design requirements. These designs are usually moderate to very complex and can have a very dense population of design elements and timing constraints. The design can be in VHDL, Verilog, ABEL, Schematic, or in some cases, a mixture of an HDL language and schematic design [36].

When the ISE process is run, the source files will be analyzed to determine if any files are out of date or have been modified. Only the necessary processes will run to update and process the design. The design will be synthesized and implemented and a programming file will be created. All output files (.map, .ngd, .bit, etc) are put in the project directory. The (.bit) file is used to configure the device for debugging purposes or for creating and downloading a PROM, ACE or JTAG file to the device.

5.3.2 Design Entry

Design entry, the first step of ISE design flow, allows users to create source files based on design objectives. A top-level design file can be created by using either a HDL, such as VHDL, Verilog, or ABEL, or a schematic. The top level module type is specified by creating a project. A project is a collection of all files necessary to create and download the design to a selected device. This process is applicable to FPGA and CPLD designs. The new project file, (project_name).NPL, will be put in the new project directory. Project Navigator will manage the project based on the target device and design flow the user selected when the project was created. It organizes all the parts of the design, and keeps track of the processes necessary to move the design from the conceptual stage through implementation in the targeted Xilinx device [36].

5.3.2.1 Using Design Constraints

Xilinx software enables the user to specify several types of constraints to help with the construction of the design. Constraints can be used in a design to control or modify the behavior of the timing within a design. Constraints will allow for specific placement of elements defined within a design. Also, the synthesis process can be controlled through the use of constraints in the synthesis constraints file. In order for the project to use constraints in ISE, the user must first create an implementation constraints file (UCF) or add a constraints file from another project if one already exists. The user can create area constraints that apply to the placement of logic on the device. Area (or placement) constraints are a way of restricting where place and route (PAR) can place a particular piece of logic. By reducing PAR's search area for placing logic, PAR's performance may be improved. Area constraints for each type of logic element, such as flip-flops, ROMs and RAMs, FMAPs, F5MAPs, and HMAPs, CLBMAPs, BUFTs, CLBs, IOBs, I/Os, edge decoders, and global buffers can be created in FPGA designs [36].

Finally, precise timing constraints for any nets or paths can be specified in the design or globally. One way of specifying path requirements is to first identify a set of paths by identifying a group of start and end points. The start and end points can be flip-flops, I/O pads, latches, or RAMs. One can then control the worst-case timing on the set of paths by specifying a single delay requirement for all paths in the set. The primary method of specifying timing constraints is by entering them in the design (HDL and schematic). However, one can also specify timing constraints in constraints files (UCF, NCF, PCF, XCF). Once the user defines timing specifications and maps the design, PAR places and routes the design based on these requirements. The results of the timing specifications can be analyzed through the command line tool TRACE (TRCE) or the GUI tool Timing Analyzer [36].

5.3.3 Performing Synthesis

Synthesis of the design can be performed after the design has been successfully analyzed. The synthesis process translates the design into gates and optimizes it for the target architecture. One can view the results of the synthesis process in the synthesis report. The synthesis report contains many sections that indicate how the design is optimized. If the design is not optimized to the user’s specifications he/she can modify the synthesis properties. The synthesis report

contains the following sections: Synthesis Options Summary, HDL Compilation, HDL Analysis, HDL Synthesis, HDL Synthesis Report, Low Level Synthesis, Final Report, Device Utilization Summary, and Timing Report [36].

5.3.4 Verifying a Design

The functionality of the design can be tested at various points of the design flow, which include behavioral simulation prior to synthesis, post-translate simulation, post-map simulation, and post-place and route simulation. Functionality and timing verification of the design can be instigated through simulator software or by a portion of the design. The simulator translates VHDL or Verilog into equivalent circuitry and reports the results based on HDL description [36].

5.3.4.1 Performing a Behavioral Simulation

Register Transfer Level (behavioral) simulation can be completed prior to synthesizing the design. This simulation is typically performed to verify code syntax and to confirm that the code is functioning as intended. Behavioral simulation can be performed on either VHDL or Verilog designs. To do this, it is necessary to create a testbench and a testbench waveform file (.TBW file), which is passed to ModelSim for simulation.

5.3.4.2 Performing a Post-Translate Simulation

Post-Translate simulation model can be generated that will contain a mapping for CLBs and IOBs in the design. This creates a (module_translate).VHD or (module_translate).V simulation file. The simulation model generated by this process can be used as input to ModelSim Xilinx Edition (MXE), HDL Bencher, or the user's own installed simulation program. Post-Translate (functional) simulation can be performed prior to mapping the design. This simulation process allows one to verify that the design has been synthesized correctly.

5.3.4.3 Performing a Post-Map Simulation

Post-Map simulation can be carried out prior to placing and routing the design. This simulation process allows one to see block delays for the design. Routing delays are not identified

in this type of simulation. This simulation passes the (test_bench).TBW or (test_bench).VHD or VER file to ModelSim for simulation.

5.3.4.4 Performing Post-Place & Route Simulation

Post-Place and Route simulation can be executed after the design has been placed and routed. After the design has been through all of the Xilinx implementation tools, a timing simulation netlist can be created. This simulation process allows one to see how the design will behave in the circuit.

5.3.5 Implementing a Design

After a design source is created, the Implement Design process converts the logical design represented in that source (and all sources in the hierarchy from that source down) into a physical file format that can be implemented in the selected target device. In Project Navigator, the implementation process can be run in one step or each step separately. The default property values are used for the implementation process unless one modifies them. Properties for the Implement Design process can be set in the Process Properties dialog. The Translate process merges all of the input netlists and design constraint information and outputs a Xilinx NGD (Native Generic Database) file. The output NGD file can then be mapped to the targeted device family [36].

The MAP process first performs a logical DRC (Design Rule Check) on the design in the NGD file produced by the Translate process. MAP then maps the logic to the components (logic cells, I/O cells, and other components) in the target Xilinx FPGA. The output design is an NCD (Native Circuit Description) file physically representing the design mapped to the components in the Xilinx FPGA. The NCD file can then be placed and routed. The Place and Route process (PAR) takes a mapped NCD file, places and routes the design, and produces an NCD file to be used by the programming file generator (BitGen). The Create Programming File process will run BitGen and create a bitstream, (module_name).BIT and place it in the project directory [36].

5.3.5.1 Translating a Design

Translate is the first step in the implementation process. The Translate process merges all of the input netlists and design constraint information and outputs a Xilinx NGD (Native Generic Database) file. The output NGD file can then be mapped to the targeted device family. It uses the default property values for the translation process unless they are modified. All processes necessary to successfully complete the translate process will run automatically and if completed successfully will result in a green checkmark next to the Translate process. The NGD file created by the translate process can be opened in the Xilinx Floorplanner (for FPGA) or ChipViewer (for CPLD) [36].

5.3.5.2 Floorplanning a Design

Xilinx Floorplanner can be used to view and edit location constraints in the design. One can manually or automatically place logic into a floorplan of the selected FPGA. In the Xilinx modular design flow, one can use the Floorplanner to assign location constraints for each module in the design. The Floorplanner can be used at several points during the design process: Prior to Mapping, Prior to Place and Route, and After Place and Route.

5.3.5.3 Viewing a Translating Report

Translate Report can be observed after running the implementation process. The translate process runs automatically during implementation or it can be run independent of the implementation process. The translate report contains warning and error messages from the three translation processes: conversion of the EDIF or XNF style netlist to the Xilinx NGD netlist format, timing specification checks, and logical design rule checks.

5.3.5.4 Mapping a Design

The Map process can be run after the design has been translated. The Map process creates an NCD file. The NCD file will be used by the PAR process for further processing. All processes necessary to successfully complete the Map process will run automatically. After the process is successfully completed, the Map Report (module_name).MRP can be viewed.

5.3.5.5 Viewing a Post-Map Static Timing Report

The output from the Post-Map Static Timing process inspected in the Post-Map Timing Report (module_name_preroute).TWX. The Post-Map Static Timing Report gives a calculated worst-case timing for all signal paths in the design. It optionally includes a complete listing of all delays on each individual path in the design. It does not include insertion of stimulus vectors. The FPGA design must be mapped and can be partially or completely placed, routed, or both.

5.3.5.6 Analyzing Post-Map Static Timing

One can analyze the timing results of the Post-Map process. Post-Map timing reports can be very useful in evaluating timing performance. Although route delays are not accounted for, the logic delays can provide valuable information about the design. If logic delays account for a significant portion (> 50%) of the total allowable delay of a path, the path may not be able to meet the timing requirements when routing delays are added. Routing delays typically account for 45% to 65% of the total path delays. By identifying problem paths, one can mitigate potential problems before investing time in place and route.

The user can redesign the logic paths to use fewer levels of logic, tag the paths for specialized routing resources, move to a faster device, or allocate more time for the path. If logic-only-delays account for much less (<35%) than the total allowable delay for a path or timing constraint, then the place-and-route software can use very low placement effort levels. In these cases, reducing effort levels allows for the decrease in runtimes while still meeting performance requirements.

5.3.5.7 Placing and Routing a Design

The place and route (PAR) process can be executed after the design has been mapped. The Map process creates an NCD file which PAR accepts as input to place and route the design. One can view the results of the place and route process. The guide report is included in the PAR report file and as a separate report. The report describes the criteria used to select each component and signal used to guide the design. It may also enumerate the criteria used to reject some subset

of the components and signals that were eliminated as candidates. One can view the output from the post-place and route timing process in the Post-Place and Route Timing Report (module_name .twx). The Post-Place & Route Static Timing Report gives a calculated worst-case timing for all signal paths in the design. It optionally includes a complete listing of all delays on each individual path in the design. One can analyze the timing results of the Post-Place and Route process. Post-PAR timing reports incorporate all delays to provide a comprehensive timing summary. If a placed and routed design has met all of the timing constraints, then one can proceed by creating configuration data and downloading a device. On the other hand, if the user identifies problems in the timing reports, he/she can try fixing the problems by increasing the placer effort level, using re-entrant routing, or using multi-pass place and route. One can also redesign the logic paths to use fewer levels of logic, tag the paths for specialized routing resources, move to a faster device, or allocate more time for the paths [36].

5.3.6 Generating a Programming File

The user can run the Generate Programming File process after the design has been completely routed. The Generate Programming File process runs BitGen, the Xilinx bitstream generation program, to produce a bitstream (.BIT) or (.ISC) file for Xilinx device configuration. The (.BIT) or (.ISC) files can then be configured by the iMPACT program for debugging the design, or creating a PROM, ACE or JTAG file to download to the device.

5.3.6.1 Configuring a Device

Configuration involves download of the output from the Generate Programming File process, (.BIT) or (.ISC) file, from a host computer to a hardware platform to configure the device for debugging or downloading to the device. The (.BIT) and (.ISC) files contain all of the configuration information from the NCD file defining the internal logic and interconnections of the FPGA, plus device-specific information from other files associated with the target device. The binary data in the BIT or ISC file can then be downloaded into the FPGA's memory cells, or it can be used to create a PROM, ACE or JTAG file.

5.4 FUSE

FUSE (Field Upgradeable Software Environment) is Nallatech's Reconfigurable Computing Operating System. FUSE facilitates flexible and scalable control and configuration of FPGA-based systems and allows data communication between the motherboard and Host PC, hence allowing data transfer to and from designs running in on-board Xilinx FPGAs. FUSE provides a number of interfaces, including the scripting language DIMEscript, the FUSE Probe GUI application and FUSE development APIs for C/C++ supplied as standard. An overview of the FUSE operating software is provided by Figure 5.10 [37].

DIMEScript has been developed by Nallatech as a simple method of accessing cards without the need to resort to programming. DIMEScript is an interpreted language, which means that the language is read in line-by-line and appropriate actions taken. This, in turn, means that any errors in the script are only found when the relevant line is executed. This is in contrast to a compiled language where the required action is checked in advance and made into a more machine friendly form. In the case of the compiled language, syntax and other features can be fully checked before running the code [6]. DIMEScript allows users to open a Nallatech card, read data from the card, write data to the card, and access various specific card functions [37].

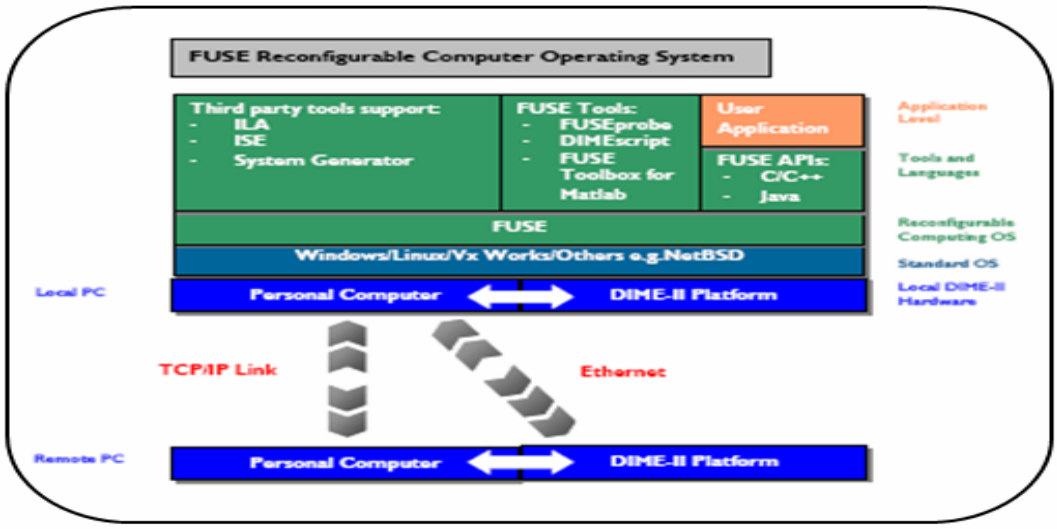

Figure 5.10: Overview of FUSE

Source: [37]

Figure 5.11 displays the program window of FUSE. As shown in the left side of the FUSE GUI, a set of user programmable buttons are provided for automating various functions such as loading system files, configuring all devices, toggling all resets, opening DIMEScript files, and running executable files.

When the user interface is loaded there are no cards open. Before opening a Nallatech card, the board must be powered and connected to a host PC through a USB port. The Card Control\Open card option must be selected from the menu in the FUSE Probe to open a card. After the card has been opened, two .BIT files must be assigned to the devices. It is important that these files be compiled specifically for the targeted board in System Generator token (see Chapter 4). The clock file (osc_clock_2v80.bit) is assigned to the Virtex2 2v80, while the bit file created by ISE for the particular design is assigned to Virtex 2 2v3000. This bitfile is used to configure the onboard FPGA on the card that was opened. Assigning bit files to the devices is done exactly in the same way when hardware Co-simulation is performed. Finally, probes must be connected to the digital I/O or the DACs on the hardware platform to be viewed on an oscilloscope or logic analyzer. Figure 5.12 illustrates the overall hardware setup.

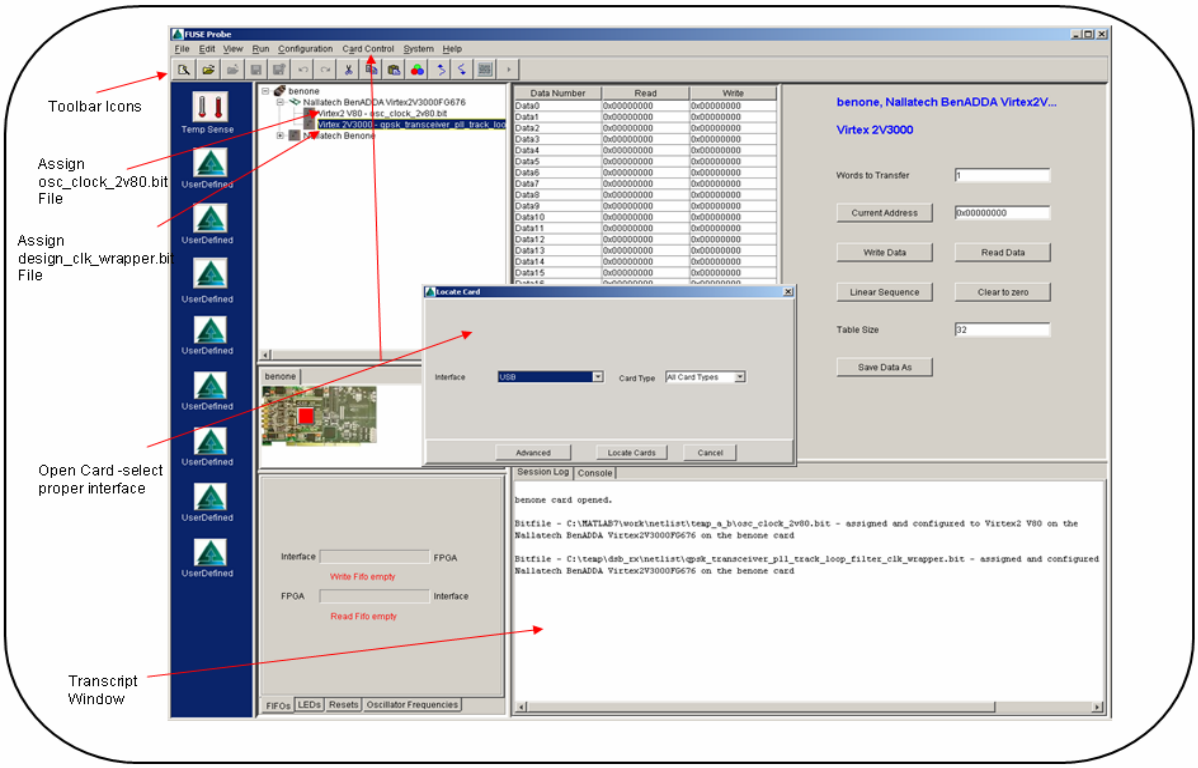

Figure 5.11: FUSE Window

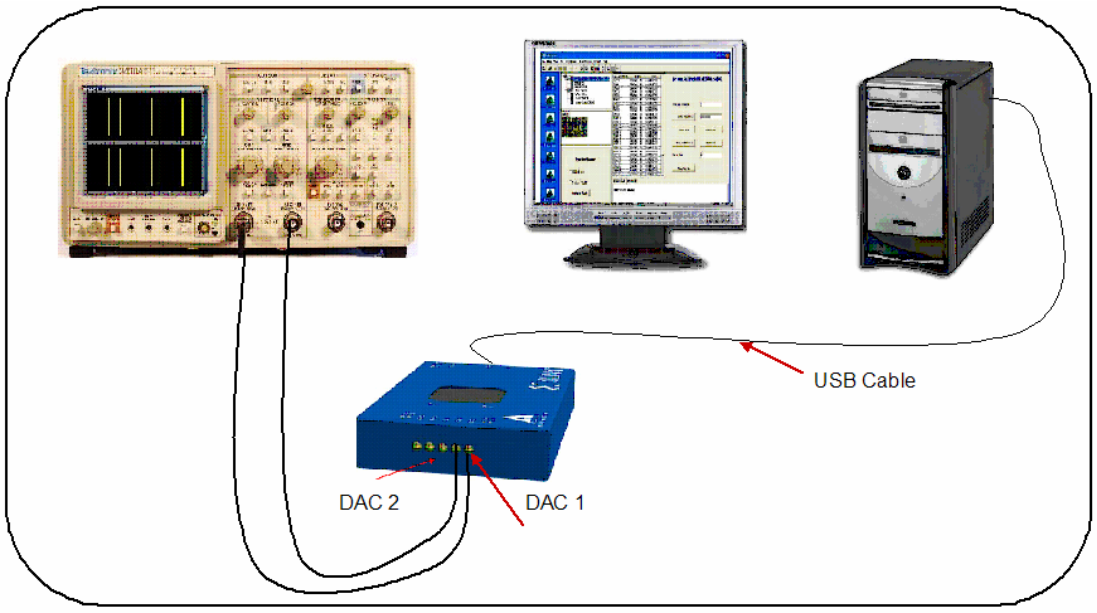

Figure 5.12: Overall Hardware Setup

5.5 BER Board Design

To view the bit error rate results of the design, a board was designed to interface with the XtremeDSP board via the pins on the digital I/O header J8. As described previously in Chapter 3, some pins of the J8 header were assigned to display the error count in the gateway out block, while other pins were assigned to represent the user inputs through the gateway in block. As described in Chapter 3, the selected user input defined the number of bits selected in the optimization block.

In Figure 5.13, the BER board, constructed with LEDs, resistors, a push button switch, and a DIP switch, was wired to interface with the J8 header through a PCI connector cable. Figure 5.14 shows the schematic of the board. The 10 red LEDs represent the 10-bit error count value. The DIP switch is used to select the user input to the gateway in block to specify the number of bits to be used in the optimization block. The green LED is used to notify the user when the simulation is complete. The push button switch is used to reset the simulation to obtain new results. Since the J8 header is connected to the board via the PCI cable, it is very important the each allocated pin of the J8 header be matched to the corresponding pin on the PCI connector to obtain correct results. Since there is only one pin allocated for GND and +3.3V, all components requiring such connections are wired to those pins.

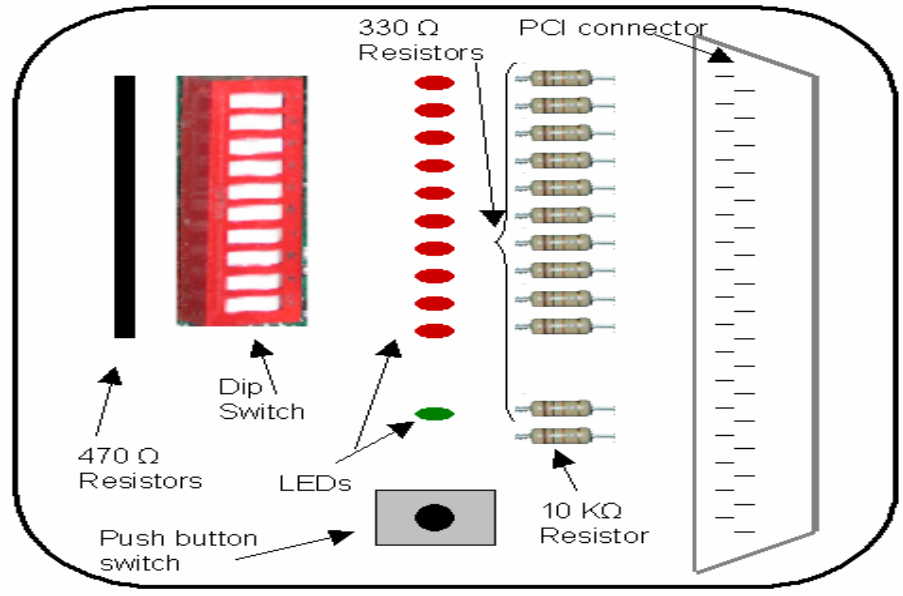

Figure 5.13: Top View of BER Board

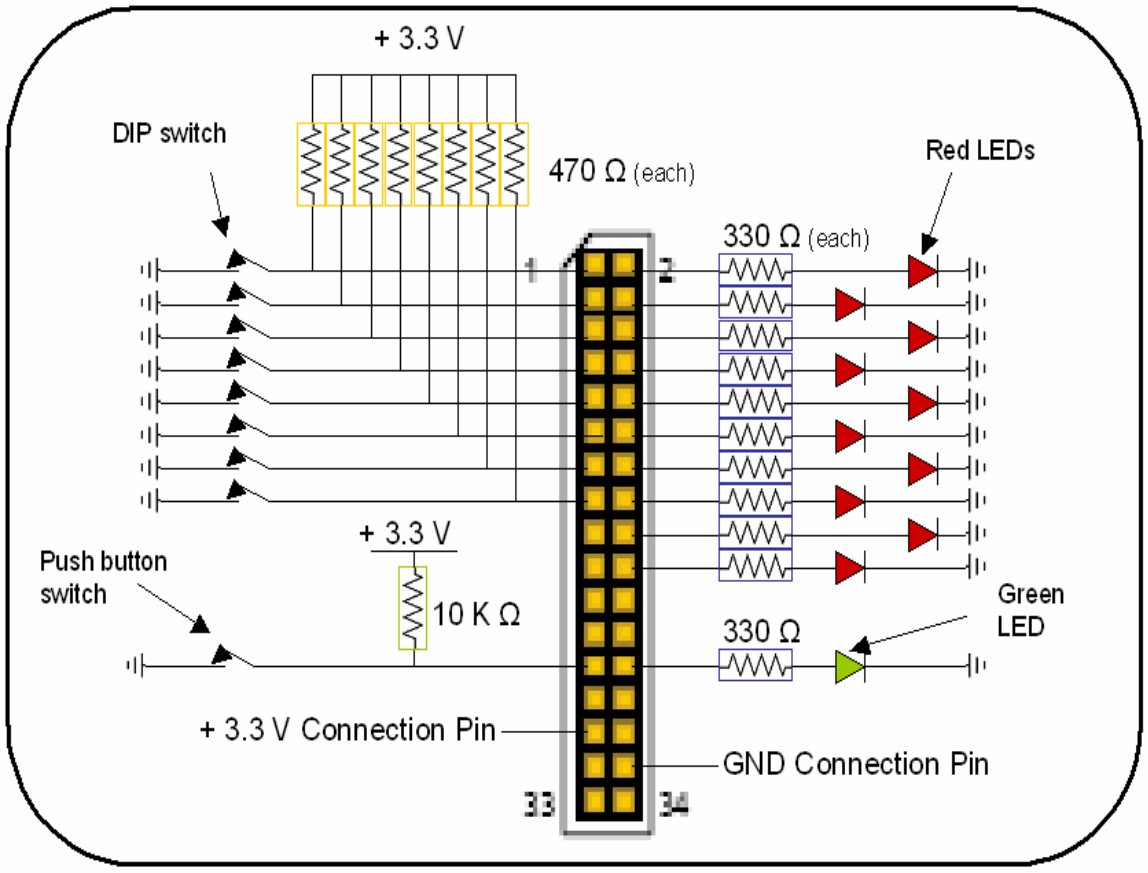

Figure 5.14: Circuit Diagram for BER Board

Xilinx Integrated Software Environment (ISE) provides the user with a powerful and well integrated environment toolbox for implementing design flow. FUSE is provided to configure the FPGA with the BIT file generated in ISE. These Xilinx design tools enable users to verify designs and accelerate the speed of simulations through hardware in the loop (HIL) simulations using PCI, USB, or JTAG interface. The computed outputs are either displayed through the digital I/O available on the board or routed back to the software environment via PCI or USB cable.

Chapter 6

Simulation Results and Analysis

Simulation plays a critical role in the design of the communication system depicted in Chapter 3. The simulation results are used for the detailed design of various system components and system level performance evaluation. Simulation of communication systems involves driving the models with input waveforms to produce outputs that can be analyzed to optimize design parameters and evaluate performance measures such as bit error rates (BER). Therefore, signal processing operations are performed by functional blocks of the communication model to generate required inputs and process them at sampled values. Some components of the system model are theoretically based and therefore quantitative in nature, while others involve approaches that are not quantifiable and are heuristic in nature. The simulation results must be validated by comparison to analytical bounds or measured results. Therefore, this chapter provides the BER results for the DSSS transceiver modeled in System Generator. Based on these results, the most optimum receiver design that maintains a specified BER performance is provided.

Floating-point mathematics is used for DSP algorithm and communication system development because it offers extensive dynamic range and accuracy and it accommodates virtually limitless word-widths and precision. Converting floating-point C or C++ code to fixed-point code is the mandatory first step in the creation of reusable algorithms since most implementations in hardware and software will limit the word-length and precision of operations. Typical effects of using fixed-point math include both overflow and quantization. A value to be stored could be too large to fit the fixed word-width (overflow). Or its precision could exceed that of the fixed-point specification or it could have too few precision bits (quantization). In order to ensure that the fixed-point algorithm behaves in an acceptable way, modeling and analyzing those effects correctly is very important.

The optimization of the design requires reduction of datapath size and hence results in quantization of the signal value by truncation. Quantization occurs whenever a value needs to be

stored with less precision than is required to represent the actual value. Therefore, an error is induced which degrade system performance. The system performance is measured by calculating BER for various datapath sizes.

BER results have been computed from real-time simulations implemented on Xilinx FPGAs. BER sensitivity of the receiver varies with the datapath size of the specific blocks shown in Figure 6.1. As explained in Chapter 3, parameterized datapath sizes are controlled from the software environment in the following points: ADC resolution, DDC datapath size, LPF datapath size, correlator height, correlator datapath size and Rectangular-to-Polar datapath size. These optimization points in the receiver are chosen relative to functionality and hardware area occupancy. To find the optimum design, each hardware simulation BER result is computed as a function of a single parameter while all other parameters are kept constant.

In each hardware simulation, the BER result is evaluated from running 1,000,000 bits for every datapath setup. Each simulation is repeated to obtain average data statistics for each parameter. The maximum datapath size at the optimization points is either 8, 14, or 16 bits, while the minimum datapath size is determined by the 1 dB BER degradation limit.

6.1 Results for Each Optimization Block

Figure 6.2 depicts BER versus ADC resolution. Since the input to the ADC is an unsigned 14 bit signal (UFix_14_0), the datapath size is varied from 14 to 5 bits. As illustrated in Figure 6.2, the BER value does not vary greatly until it reaches a precision of less than 6 bits. Therefore, the minimum number of bits is 5 with respect to 1dB degradation. In a CDMA environment, this resolution would need to be higher to accommodate required number of simultaneous users [38].

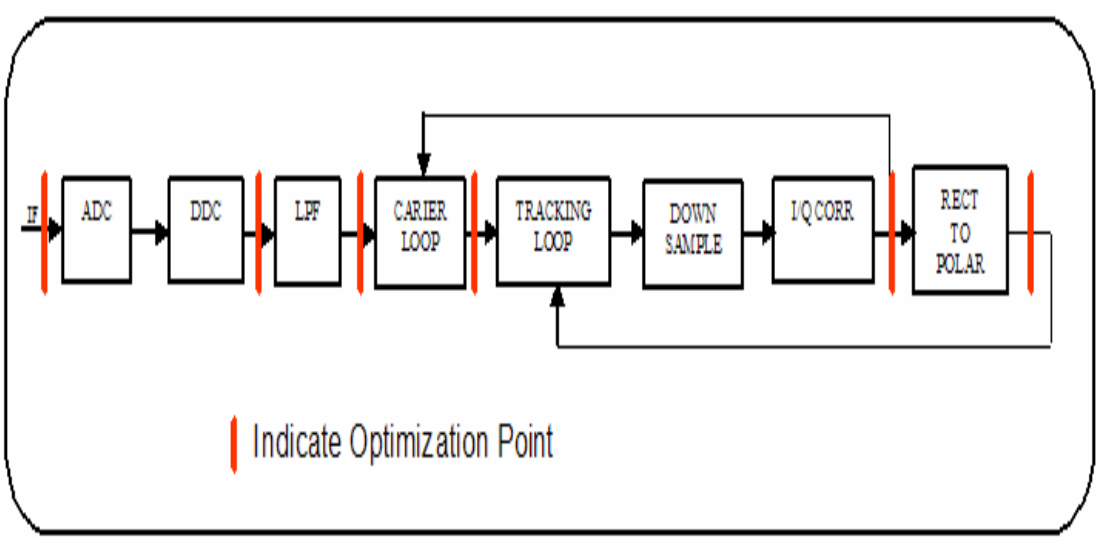

Figure 6.1: Optimization Points in DSSS Receiver

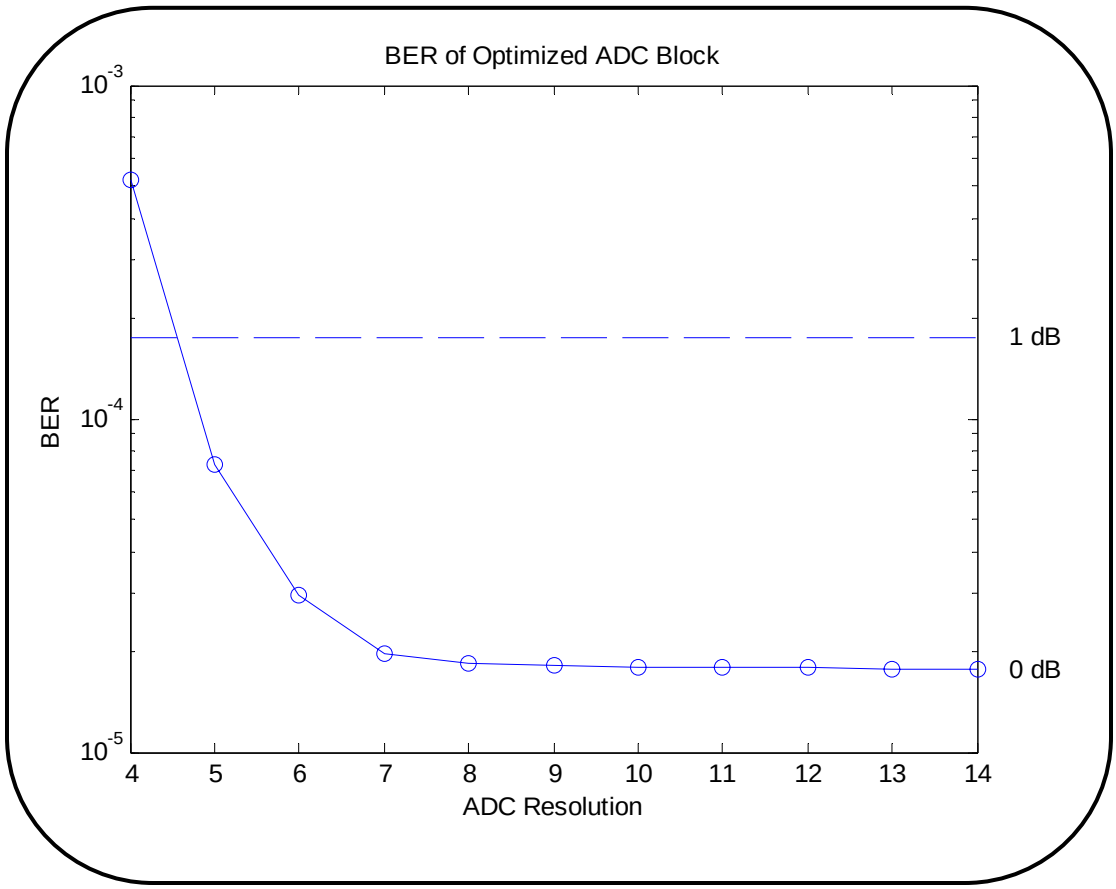

Figure 6.2: BER versus ADC Resolution

Optimization of the DDC datapath size is necessary since it is the input to the low pass filter, which occupies a considerable area on hardware. The variation of the datapath size determines a level of round off error in the block. Figure 6.3 shows that 6 bits provide enough precision to stay within the 1 dB degradation limit.

The low pass filter in Figure 6.1 is implemented as a second order running average to reduce hardware area occupancy. Figure 6.4 shows that eight bits of resolution is sufficient to provide less than one dB of performance degradation.

In Figure 6.5, BER is shown as a function of I and Q correlator input size. The number of bits used for these inputs corresponds to the cell size required in implementation of a parallel correlator. Since the correlators occupy a significant portion of the hardware area, it is necessary to minimize the size of their inputs. By minimizing the input size, the computational complexity of the MAC operations is reduced. The results show that as few as three bits are more than enough to maintain well below the 1 dB degradation limit.

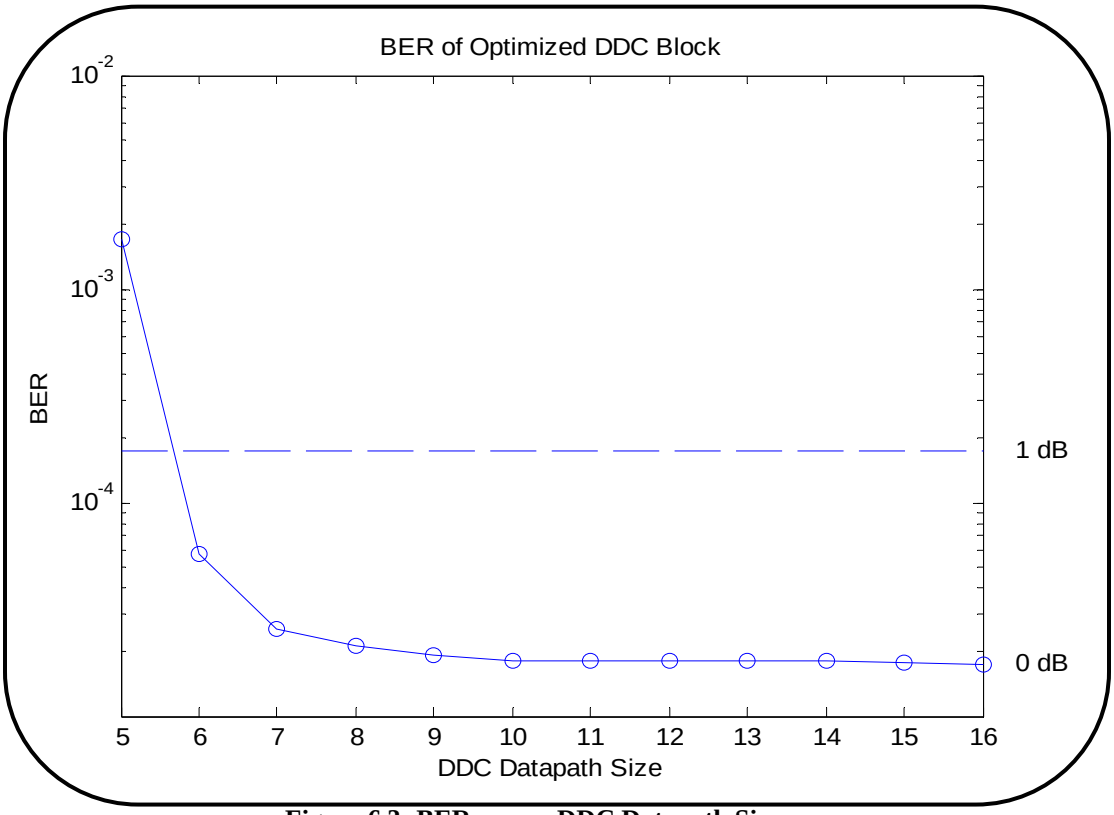

Figure 6.3: BER versus DDC Datapath Size

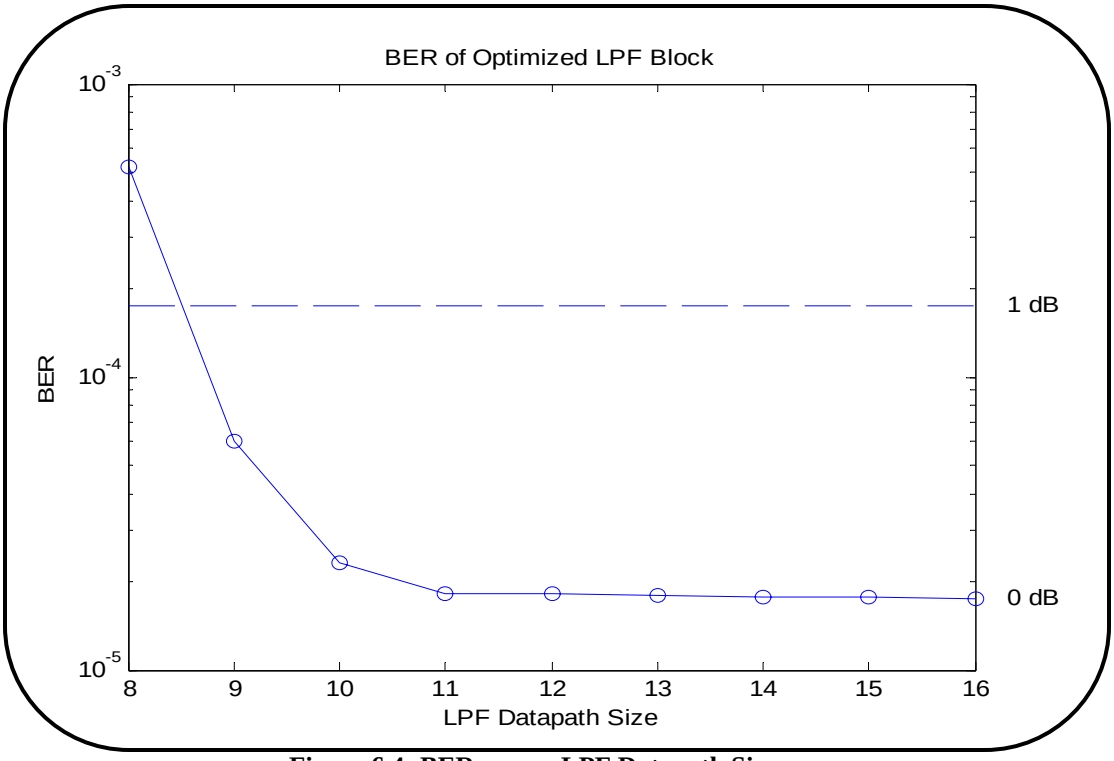

Figure 6.4: BER versus LPF Datapath Size

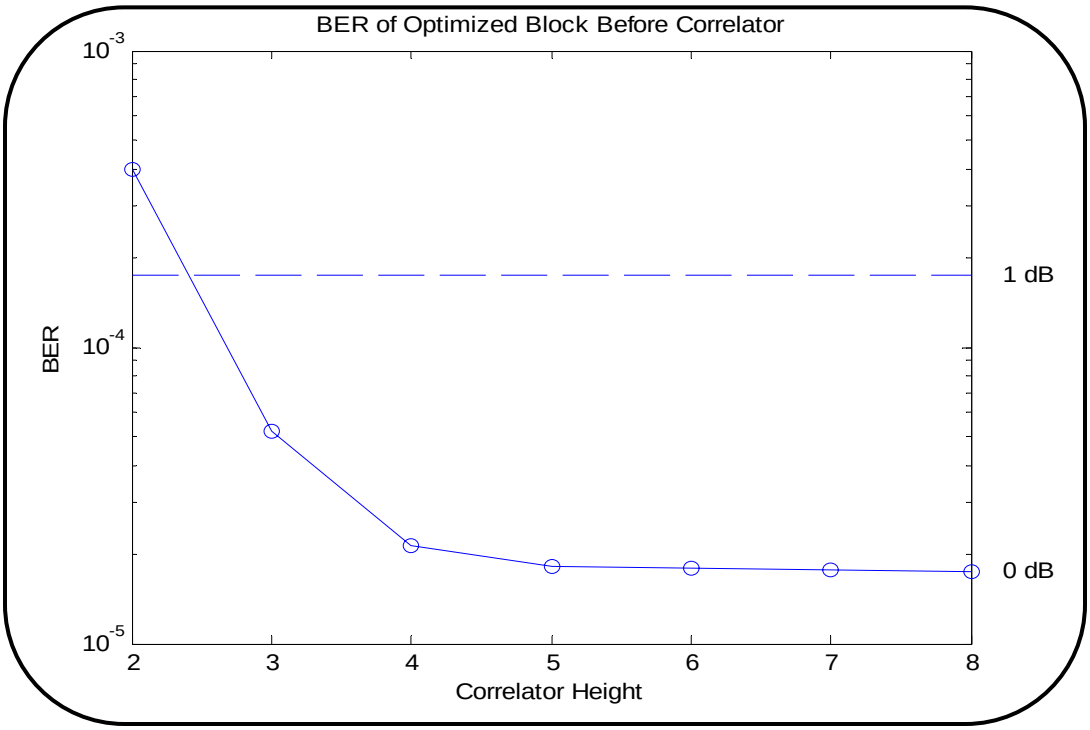

Figure 6.5: BER versus Correlator Datapath Height

Another critical parameter for determining the size of the parallel correlator is the datapath size of the correlator's adding tree. The parallel architecture for the correlators is implemented through FIR filter blocks. At each clock cycle, the sum of memory blocks outputs gives an intermediate sum-of-multiplications result. The accumulator at the end of the adder tree gives the complete FIR filter result. Therefore, minimizing the critical path of the accumulator is necessary to optimize the design. As Figure 6.6 shows, 7-bit addition arithmetic is needed to satisfy the 1 dB degradation limit.

Finally, the relationship between BER and datapath size of the rectangular-to-polar block is displayed in Figure 6.7. This block is implemented using the CORDIC algorithm. Refer to Chapter 3 for more details on CORDIC. Given a complex input, this block outputs an equivalent vector in magnitude and angle format. The datapath size of the CORDIC is important since the phase values used to decode the symbol is obtained from the CORDIC output. Therefore, there is a direct relationship between precision needed to decode the symbol in presence of noise and the BER. Figure 6.6 illustrates that at least 6 bits are necessary in the datapath size to remain within the 1 dB degradation limit.

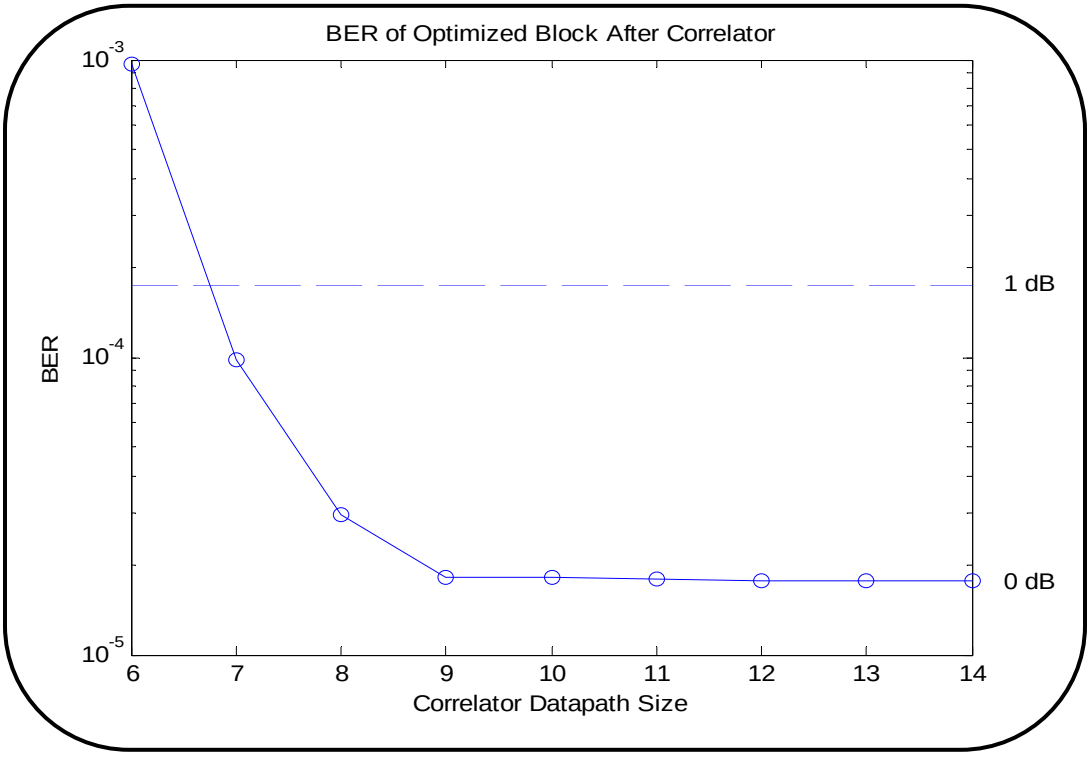

Figure 6.6: BER versus Correlator Datapath Size

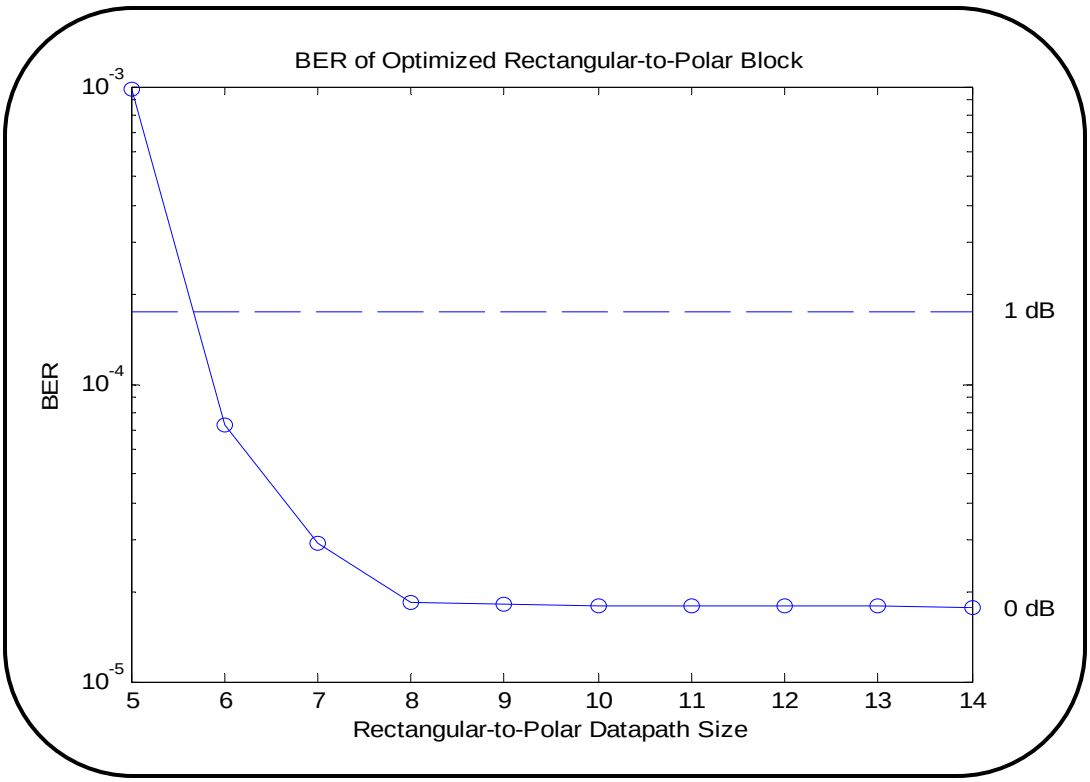

Figure 6.7: BER versus Rectangular-to-Polar Datapath Size

6.2 Minimum Area Solutions

Using the hardware co-simulation results presented in Figure 6.3 through Figure 6.7, one can determine a set of minimum values for the datapath sizes so that the DSSS receiver implementation size is minimized for a given implementation loss. Table 6.1 shows several possible minimum area solutions together with the full precision case. The second column of the table represents the full precision case where BER is the smallest and the implementation area is the largest. Four other minimum area cases are shown in consecutive columns. The third column shows the smallest area case with the highest BER. Other three cases can be considered sub-optimal with respect to the area and BER. The results indicate that a 1-dB degradation can be maintained with an 8- bit ADC resolution, a DDC datapath size of 6 bits, a filter datapath size of 11 bits, a 3 bit correlator height, a 9 bit correlator datapath size, and a 8 bit rectangular-to-polar datapath size. A design implemented with these parameters occupies 6894 FPGA slices. This implies an area reduction of 38% when compared to the full precision case which occupies 10775 FPGA slices.

Table 6.1: BER and Implementation Area versus Datapath Size

ADC Resolution	14	5	8	8	8
DDC Datapath	16	6	6	6	6
Filter Datapath	16	9	11	11	12
Correlator Height	8	3	3	3	3
Correlator Datapath	14	7	9	9	10
Rec-to-pol Datapath	14	6	8	9	8
BER	1.75e-5	5.09e-4	1.74e-4	1.69e-4	1.37e-4
FPGA Slices	10775	6888	6894	6895	6896

6.3 Effects of Optimization

The minimization of datapath size results in quantization error which can be modeled as noise. The quantized value is the summation of the original value and the error induced by the quantization process. When fixed-point arithmetic is used, quantization is an inevitable side effect that typically exhibits itself as "noise." Similarly, the results generated by the vast numbers of multiply-and-accumulate operations used in digital signal processing and communications algorithms are frequently larger than the fixed word-width that has been specified by the design, causing overflow. Overflow can cause the signal to be distorted or can introduce unpredictable non-linear behavior. A general mechanism needs to be established to define specific overflow and quantization behavior. By default "TRUNCATED" quantization and "WRAPPED" overflow are used since this is the behavior of hardware designed to perform 2's complement arithmetic [39].

6.4 Truncation

The problem of quantizing arises when computations which are either fixed-point or floating-point arithmetic are performed. Quantization via truncation results in a lower level of precision and introduces errors that depend on the number of bits in the original value relative to

number of bits after quantization [30]. In fixed-point representation, truncation error occurs from the quantization of b_u bits representing a value x into b bits. Thus the number

$$x = \overbrace{0.1011\ldots\ldots 11}^{b_u} \tag{6.1}$$

consisting of b_u bits before quantization is represented as

$$x = \overbrace{0.101\ldots.1}^{b} \tag{6.2}$$

after quantization, where $b<b_u$. The truncation of the value x results in a truncation error defined as

$$E_t=Q_t(x)-x. \tag{6.3}$$

Considering sign-magnitude and two's-complement representation, the positive numbers have identical representation in both forms. Therefore, truncation results in a value that is smaller than the unquantized value for positive numbers. As a result, the reduction of significant bits from b_u to b results in truncation error of

$$-(2^{-b} - 2^{-bu}) \le E_t \le 0. \tag{6.4}$$

According to this equation, the largest error would occur from discarding $b_u - b$ bits, all of which are ones. When considering negative fixed-point numbers based on sign-magnitude representation, the truncation error is positive since it just reduces the magnitude of the numbers. Therefore, the truncation error for negative numbers is

$$0\le E_t \le -(2^{-b} - 2^{-bu}) . \tag{6.5}$$

In the two's complement representation of negative numbers, subtraction of the corresponding positive number from 2 results in the negative of a number. Therefore, truncation results in an increase of the magnitude of the negative number. Since $x > Q_t(x)$, the truncation error is

$$-(2^{-b} - 2^{-bu}) \le E_t \le 0. \tag{6.6}$$

Consequently, truncation error for sign-magnitude representation is symmetric about zero and falls in the range

$$-(2^{-b} - 2^{-bu}) \le E_t \le (2^{-b} - 2^{-bu}) . \tag{6.7}$$

Alternatively, the truncation error for two's complement representation is always negative and falls in the range

$$-(2^{-b} - 2^{-bu}) \le E_t \le 0. \tag{6.8}$$

Figure 6.8 shows the quantization errors due to truncating for the two's-complement representation and sign-magnitude representation.

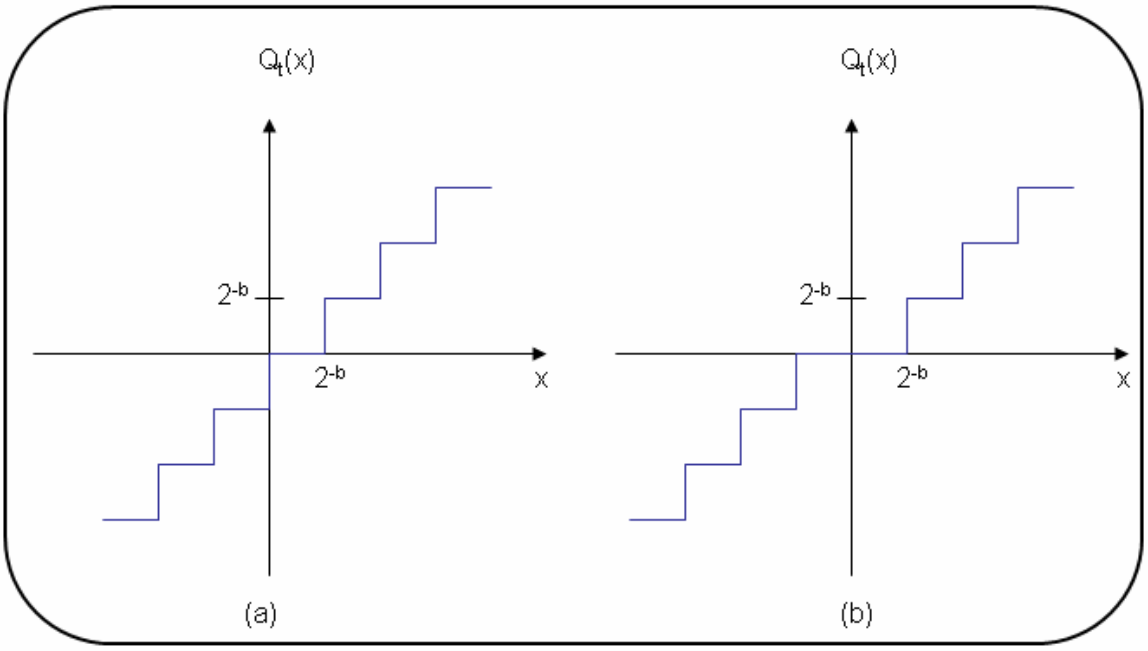

Figure 6. 8: Effects of Quantization for (a) two's complement representaion and (b) sign-magnitude representation

Although, error calculations can be evaluated for quantization effects, it is difficult to construct an algorithm that computes the performance degradation due to the complexity of the design and the fact that the error accumulates over time.

The results obtained in Table 6.1 define sub-optimum solutions with respect to BER and area. To find the global optimum, one needs to assign weights to each datapath size corresponding to their contribution to the implementation area, and then implement a multivariable constrained optimization. Similarly, one can optimize the receiver with respect to the power by weighting the datapath sizes according to their individual contributions to the total receiver power.

Chapter 7

Summary and Future Work

In this thesis a DSSS transceiver is designed in a completely software environment with System Generator and implemented on a hardware platform via Xilinx implementation tools such as ISE. Further, HIL simulations are performed to find an optimized system design with a specified performance level. The DSSS transceiver consists of a transmitter that performs QPSK modulation, an AWGN channel and a receiver comprised of a digital down converter, low pass filter, carrier phase lock loop, I and Q correlators, tracking lock loop, peak detector, and rectangular-to-polar converter.

7.1 Summary

The concept of a software-defined-radio (SDR) has been of considerable academic and industrial interest for several years. Started in the military, SDR now serves many commercial purposes. SDR provides features such as flexibility, scalability, and inter-operability that were not available in traditional radio based on a hardware approach. SDR allows a radio to be described by its software; thus, a single radio can change its operation to suit the current needs of the system. In software defined radios, FPGAs are being used increasingly as a general-purpose computational fabric to implement hardware acceleration units that boost performance while lowering cost and power requirements. Software defined radios require extensive processing power to realize the portability of waveforms and reconfigurability that has been promised. The use of FPGAs for hardware acceleration offers promising architectural options that are helping to make SDRs a reality. Hardware implementation speeds up the design verification process.

Pure simulation is often used to understand the behavior of a system, or to predict an outcome under different internal and external influences. But if the simulation is being used as a basis for proving control feasibility, the risk of investment can be further reduced utilizing a HIL simulation approach. Good system engineering practice would begin with a pure simulation and as components become better defined (with the aid of simulation), they can be fabricated and

replaced in the control loop. Once physical components are added to the loop, un-modeled characteristics can be investigated, and controls can be further refined. The use of HILS eliminates expensive and lengthy iterations in machining and fabrication of parts, and speeds development towards a more efficient design.

System Generator is a system level modeling tool that facilitates FPGA hardware design. System Generator extends Simulink in numerous ways in order to provide a powerful modeling environment that is well suited to hardware design. The tool provides high-level abstractions that are automatically compiled into an FPGA at the push of a button. The tool also provides access to underlying FPGA resources through lower level abstractions, allowing designers to implement highly efficient FPGA designs. System Generator provides hardware co-simulation interfaces making it possible to incorporate a design running in an FPGA directly into a Simulink simulation. Hardware co-simulation compilation targets automatically create a bitstream, and associate it to a block. When the design is simulated in Simulink, results for the compiled portion are calculated in hardware. This allows the compiled portion to be tested in actual hardware, and can speed up simulation dramatically.

ISE integrates everything a designer may need in a complete logic design environment for all leading Xilinx FPGA and CPLD products. Easy-to-use, built-in tools and wizards also make I/O assignment, power analysis, timing-driven design closure, and HDL simulation quick and intuitive. FUSE software allows the design created in System Generator and implemented in HDL by ISE to be downloaded on a hardware platform.

The infusion of System Generator and ISE enables HIL emulation for design verification and performance evaluation. Bit error rate is used as a measure of performance. BER sensitivity of the receiver is determined from varying ADC resolution, DDC datapath size, LPF datapath size, correlator height, correlator datapath size and rectangular-to-polar datapath size. The simulation results are used to obtain a minimum area solution. The results indicate that less than one dB degradation can be maintained with an 8 bit ADC resolution, 6 bit DDC datapath, 11 bit filter datapath, 3 bit correlator height, 9 bit correlator datapath, and 9 bit rectangular-to-polar datapath.

7.2 Future Work

Although a minimum area solution has been found, it is not the optimum solution. To obtain an optimum solution, an optimization code needs to be performed that takes all the obtained results and finds the global minimum in accordance to a specified BER performance. Therefore, it will generate all the possible combinations of datapath sizes for the functional blocks and evaluate hardware occupancy. The optimum solution will be the one that gives the lowest number of FPGA slices used and still maintains less than one dB of BER degradation.

Also, the design process can be automated, eliminating the need of a designer to change parameters and record results. Such automation would download the system parameters on the board, perform BER calculations, report the results back to the simulation environment, update the system parameters and repeat the process over again.

List of References

[1] Wipro Technologies, "Software Defined Radio," *White Paper*, August 2002. http://www.wipro.com

[2] M. Bobrek, M. Howlader, and J. Suh, "Hardware Optimization of a DSSS Receiver Using Simulink Model," Proceedings of the GSPx Conference, Santa Clara, CA, 2004.

[3] P. Shaumont and I. Verbauwhede, "Interactive Cosimulation with Partial Evaluation," *Proceedings of the Conference on Design, Automation and Test*, Paris, France, 2004.

[4] C. Kreiner, C. Steger, and R.Weiss, "A Hardware/Software Cosimulation Environment for DSP Applications," *Proceedings of EUROMICRO5 Conference*, 1999.

[5] S. Yoo and K. Choi, "Optimistic Distributed Timed Cosimulation based on Thread Simulation Model" *Proceedings of the 6th International Workshop on Hardware/Software Codesign*, Seattle, WA, 1998.

[6] W. Sung and S. Ha, "Optimized Timed Hardware Software Cosimulation without Roll-back," *Proceedings of the Conference on Design, Automation and Test*, Paris, France, 1998.

[7] P. Crilly, J. Rutledge, and B. Carlson, *Communication Systems*. New York: McGraw Hill, 2002.

[8] K. Sienski and C. Field, "Digital Transceiver Software Defined Radio Applications"

[9] McCarthy, Darren, "Software Defined Radio: Integration for innovation," RF Design, September 2005.

[10] D. Sweeny, "Software Defined Radio: A Revolution in the Making," Broadband Wireless Magazine, Vol 4, No.1, Jan/Feb. 2003.

[11] A. Rudra, "The Rising Importance of FPGA Technology in Software Defined Radio," *COTS Journal*, January 2005.

[12] SD-Radio: A Software Defined Radio www.qsl.net/padan/sdradio

[13] J. Reed, *Software Radio: A Modern Approach to radio Engineering*. New Delhi: Pearson Education, 2002.

[14] R. Sathapan, and C. Flemming, "SDR and JTRC Ride the DSP.FPGA Wave." *COTS Journal*, September 2005.

[15] P. Mackenzie, L. Doyle, D. O' Mahony, and K. Nolan, "Software Radio on General Purpose Processors," Tinity College, Dublin.

[16] B. Wong, “Filling the Generation Gap with Software-Defined Broadband Radio,” *CTI*, Volume 4 Number 9.

[17] D. Boppana, and J. Seely, “FPGAs Help Software-Defined Radios Adapt,” *Wireless System Design*, 2005.

[18] J. Seely, “Using Hardware Acceleration Units in Software Defined Radio Modem Functions,” *COTS Journal*, January 2005.

[19] Xilinx Inc., “Xilinx High Performance Signal Processing” January 1998. http://www.nalanda.nitc.ac.in/industry/appnotes/xilinx/documents/products/logicore/docs

[20] N. Cravotta, “Managing High Speed Analog Signals for SDR Applications Using FPGAs,” Avnet Electronic Marketing.

[21] Xilinx Products and Services http://www.reconfigurable.com/products/software/sysgen/hw_loop.htm

[22] System Generator User Guide www.xilinx.com

[23] T. Rappaport, *Wireless Communications: Principles and Practic*e. New Delhi: Pearson Education, 2002.

[24] R. Roberts, “The ABCs of Spread Spectrum,” http://www.sss-mag.com/ss.html

[25] HowStuffWorks Website http://www.howstuffworks.com/question525.htm

[26] CDMA Interactive Website http://www.cdmaonline.com/interactive/workshops/terms1/1008.htm

[27] M. K. Simon, “Bandwidth-Efficient Digital modulation with Application to Deep Space Communications,” Hoboken, NJ: John Wiley & Sons, 2003.

[28] Ziemer/Tranter, *Principles of Communications: Systems, Modulation, and Noise*, 2nd Ed. Dallas: Houghton Mifflin Company, 1985.

[29] L. Couch, *Digital and Analog Communication Systems*. Upper Saddle River: Prentince Hall, 2001.

[30] J. Proakis and D. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*. New Delhi: Prentince Hall, 2003.

[31] W. Tranter, K. Shanmugan, T. Rappaport, and K. Kosbar, *Communication Systems Simulation with Wireless Applications*. Upper Saddle River: Prentince Hall, 2004.

[32] Xilinx Products and Services
http://www.xilinx.com

[33] Matlab Help Guide

[34] Xilinx Software Manual

[35] Xtreme DSP Development Kit-II User Guide
www.nallatech.com

[36] Xilinx ISE 6 Software Manuals
http://www.xilinx.com

[37] FUSE System Software User Guide
http://www.nallatech.com

[38] M. Bobrek, M. Howlader, and B. Dhillon, "Optimization of a DSSS Receiver Using Hardware Co-Simulation„" Proceedings of the GSPx Conference, Santa Clara, CA, 2005.

[39] D. Johnson, "Development of Reusable Algorithms Based on C and C++," http://archive.chipcenter.com/asic/tn003.html

Appendix

Appendix A

Header Pin Number	Name	User FPGA (2V3000FG676) PIN No
1	ADJIN<12>	Y6
2	ADJIN<13>	Y5
3	ADJIN<10>	W6
4	ADJIN<11>	W7
5	ADJIN<8>	U4
6	ADJIN<9>	U3
7	ADJIN<6>	U6
8	ADJIN<7>	U5
9	ADJIN<4>	P6
10	ADJIN<5>	P5
11	ADJIN<2>	P4
12	ADJIN<3>	P3
13	ADJIN<0>	F14
14	ADJIN<1>	G14
15	ADJIN<14>	AA4
16	ADJIN<15>	AA3
17	ADJIN<16>	T4
18	ADJIN<17>	T3
19	ADJIN<18>	T5
20	ADJIN<19>	T6
21	ADJIN<20>	V5
22	ADJIN<21>	V4
23	ADJIN<22>	V6
24	ADJIN<23>	V7
25	ADJIN<24>	W5
26	ADJIN<25>	W4
27	ADJIN<26>	Y1
28	ADJIN<27>	W1
29	3.3V	NC
30	GND	NC
31	NC	NC
32	NC	NC
33	NC	
34	NC	

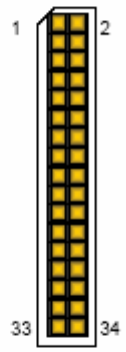

Table 17: Adjacent Bus Digital I/O Header

Appendix B

Motherboard Main Power LEDs

In addition to the power supplies for the module the Kit contains LEDs which indicate the status of the main power supplies for the motherboard itself. Table 19 on page 52 defines their use.

LED	Purpose	General Operation State
D10	2.5V power indicator	GREEN
D14	3.3V power indicator	GREEN
D15	5V power indicator	GREEN

Table 19: Motherboard Main Power LEDs

8.2.3 User LEDs

Module User LEDs

The BenADDA DIME-II module has two user tricolor LEDs, which can be used for specific design purposes.

Signal Description	User LED	Signal Name	Main User FPGA (XC2V3000) Pin
Green Diode for LED2	D2	LED_Green2	Y23
Red Diode for LED2	D2	LED_Red2	Y24
Green Diode for LED1	D1	LED_Green1	Y11
Red Diode for LED1	D1	LED_Red1	AA11

Table 20: Module User LEDs

Motherboard User Status LEDs

The BenONE DIME-II motherboard has 4 software controllable LEDs. These LEDs are connected to the interface FPGA (XC2S200) and can only be controlled via software calls to the card and NOT via signals from the main User FPGA (XC2V3000). Please see "Software" on page 54 for details of the relevant software calls.

Appendix C

Design Optimization of a DSSS Receiver Using Hardware Co-Simulation

Miljko Bobrek, PhD.
Oak Ridge National Laboratory
P.O.Box 2008, M.S. 6003
Oak Ridge, TN 37831-6003
865-574-5694
bobrekm@ornl.gov

Mostofa Howlader, PhD.
The University of Tennessee
ECE Department
Knoxville, TN 37919
865-974-5415
howlader@utk.edu

Balbir Dhillon
The University of Tennessee
ECE Department
Knoxville, TN 37919
bdhillon@utk.edu

ABSTRACT

A Direct Sequence Spread Spectrum (DSSS) receiver has been designed in Matlab-Sysgen® environment and implemented on a Software Defined Radio (SDR) hardware platform. Using hardware co-simulation, the receiver was optimized with respect to the datapath size of its main building blocks. The hardware co-simulation offered a significant increase in the simulation speed over the software simulation. Running the hardware at bit rates of several hundreds of Kbit/s allowed a quick turnaround even if tens of millions of bits were required for a single-point bit error rate (BER) measurement. The datapath size was varied for different functional parts of the receiver, such as analog-to-digital converter (ADC), digital down-converter (DDC), low-pass filter, correlator, carrier phase locked loop (PLL), and the tracking loop. As a final result, a minimum-hardware receiver with an acceptable BER degradation has been designed.

Keywords

Hardware co-simulation, Software Defined Radio, Digital down-converter, matched filter, analog-to-digital converter round-off error.

1. INTRODUCTION

Use of software-based true-cycle simulators is often prohibitive due to a large number of cycles needed to achieve a sufficient data statistics. This is especially true in the case of simulation of SDR applications that sometimes involve millions of states. Even though different methods have been proposed to speed up the software simulation [1], the total number of computations necessary to simulate even a medium complexity radio significantly exceeds capabilities of mainstream office and lab computers. As an alternative, the hardware simulation offers significant improvement with respect to the simulation time. Recently, many alternative approaches for hardware co-simulation of complex communication systems have been proposed [2], [3], [4], and [5]. They differ in their emphasis on conflicting goals such as simulation speed, accuracy, flexibility, and so on. In this paper, a Simulink® model for a DSSS receiver was implemented and verified for its correct functionality. To speed up the BER performance testing, the design was synthesized and downloaded to a hardware platform. The software environment was used to change simulation parameters.

2. SIMULATION MODEL

Figure 1 represents a simplified DSSS architecture where the main parts of the receiver are the analog-to-digital converter (ADC), digital down-converter (DDC) , image rejection low-pass filter (LPF), carrier phase locked loop, tracking locked loop, down-sampler, spread spectrum correlator, and rectangular-to-polar block. This architecture was implemented in Simulimk® using Sysgen® library, and then downloaded to a Nallatech® hardware platform. In the implementation, the input to the receiver was a QPSK modulated 13 MHz IF signal. The chip rate of the spread spectrum baseband signal was 3.25 MHz, and the PN code length was 63. The DDC block of Figure 1 down-converts the IF signal to the zero-IF complex base-band signal that is filtered by the low-pass filter. The carrier phase locked loop locks to the zero-IF baseband signal, while the tracking loop is performing symbol tracking. Two separate correlators serve as matched filters for I and Q baseband signals. At the end, the rectangular-to-polar block provides the magnitude and the phase of the received symbols.

To determine the BER sensitivity of the receiver to the datapath size of specific blocks, a separate BER counter was implemented in hardware. A number of parameterized datapath points were also implemented, and their sizes were controlled from the Simulink® environment. The following datapath points were used in the simulation:

- ADC resolution
- DDC datapath size
- LPF datapath size
- Correlator height
- Correlator datapath size
- Rectangular-to-polar datapath size

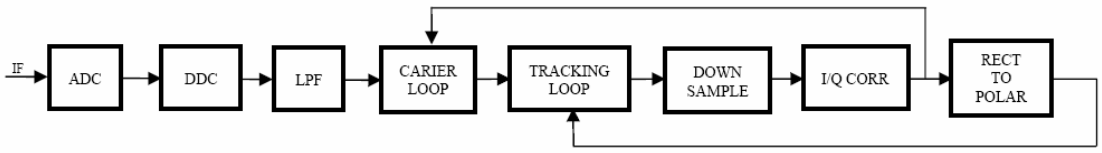

Figure 1. DSSS receiver block diagram

Hardware simulation was run for 1,000,000 bits for every datapath setup. The maximum datapath size was either 14 or 16 bits, while the minimum datapath size was determined by 1 dB BER degradation limit.

3. SIMULATION RESULTS

In the following figures, the BER is shown as a function of a single parameter while the other parameters were kept constant. Figure 2 shows BER versus the ADC resolution. With the respect of the 1dB degradation line, the minimum number of bits is 5. In a CDMA environment, the resolution would be higher to accommodate required number of simultaneous users.

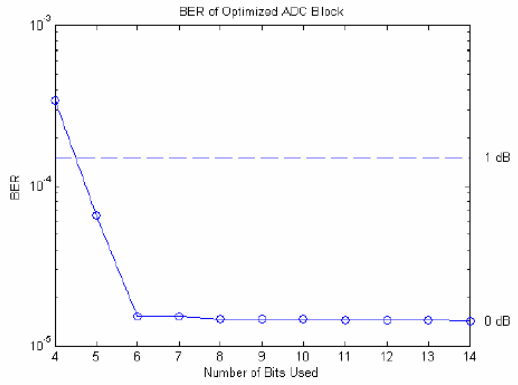

Figure 2. BER versus ADC resolution

Figure 3 shows BER dependence on the DDC datapath size which determines the level of the round-off error in the block. Here, the 6-bit arithmetic is sufficient to provide less then one dB of performance degradation.

The low-pass filter of Figure 1 is implemented as a second order running average. As Figure 4 shows, 8 bits of resolution is sufficient for maintaining the 1 dB BER degradation.

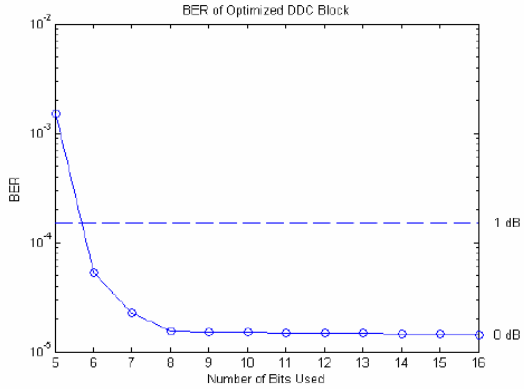

Figure 3. BER versus DDC datapath size

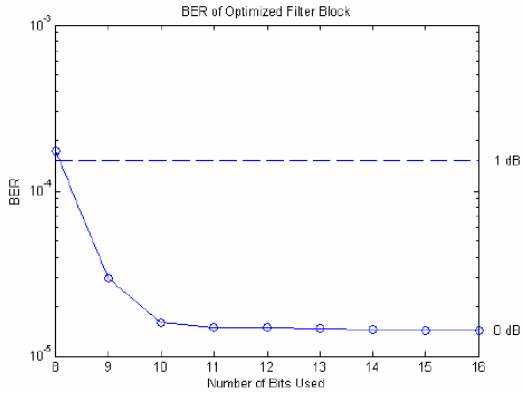

Figure 4. BER versus LPF datapath size

In Figure 5, BER is shown as the function of number of bits at the correlator I and Q inputs. The size of the correlator inputs is critical because of the size of memory cells required in implementation of a parallel correlator.

Here, as few as 2 bits at the correlator inputs are sufficient to have almost 1 dB of BER degradation.

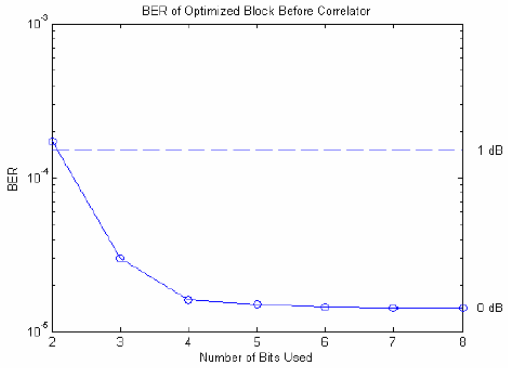

Figure 5. BER versus correlator height

Another important parameter that determines the size of a parallel correlator is the datapath size of the correlator's adding tree. As Figure 6 shows, 7-bit addition arithmetic is needed to satisfy the 1 dB degradation limit.

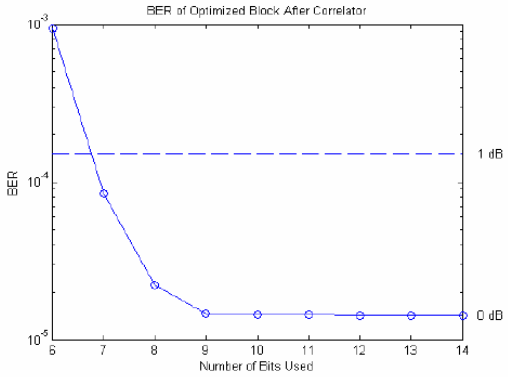

Figure 6. BER versus correlator datapath size

Finally, Figure 7 shows correlation between the BER and the datapath size in the rectangular-to-polar block. In this particular example, the block is implemented using CORDIC algorithm. To keep the BER degradation below 1 dB, this block needs to have at least 6 bits in its datapath.

Using the hardware co-simulation results presented in Figure 3 through Figure 7, one can determine a set of minimum values for the datapath sizes so that the DSSS receiver implementation size is minimized for a given

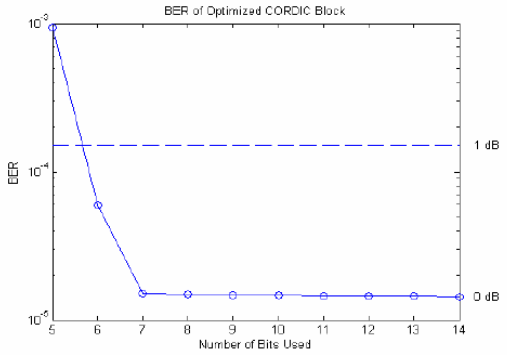

Figure 7. BER versus rec-to-pol datapath size

implementation loss. Table 1 shows several possible minimum area solutions together with the full precision case. The second column of the table represents the full precision case where BER is the smallest and the implementation area is the largest. Four other minimum

Table 1. BER and implementation area versus datapath

ADC Resolution	14	5	8	8	8
DDC Datapath	16	6	6	6	6
Filter Datapath	16	9	11	11	12
Correlator Height	8	3	3	3	3
Correlator Datapath	14	7	9	9	10
Rec-to-pol Datapath	14	6	8	9	8
BER(10^{-5})	1.42	46.9	14.2	14.0	11.5
FPGA Slices	6167	4367	4382	4385	4383

area cases shown in consecutive columns. The third column shows the smallest area case with the highest BER. Other three cases can be considered sub-optimal with respect to the area and BER. To find the global optimum, one needs to assign weights to each datapath size corresponding to their contribution to the implementation area, and then implement a multivariable constrained optimization. Similarly, one can optimize the receiver with respect to the power by weighting the datapath sizes according to their individual contributions to the total receiver power.

4. CONCLUSIONS

In this paper, the hardware co-simulation is used to determine BER sensitivity to the datapath size in various parts of a DSSS receiver. The co-simulation runs at the real-time data rate speed, and it is therefore orders of magnitude faster than the software-only simulation. Using hardware co-simulation designers can determine minimum-area or minimum-power implementation of complex designs such as SDR applications in a relatively short period of time.

5. REFERENCES

[1] Bobrek, M., Howlader, M., Suh, J., "Hardware Optimization of a DSSS Receiver Using Simulink® Model," *Proceedings of the GSPx conference,* Santa Clara, CA, 2004.

[2] Shaumont, P., Verbauwhede, I., "Interactive Cosimulation with Partial Evalution," *Proceedings of the Conference on Design, Automation and Test*, Paris, France, 2004.

[3] Kreiner, C., Steger, C., Weiss, R., "A Hardware/Software Cosimulation Environment for DSP Applications," *Proceedings of EUROMICRO conference*, 1999.

[4] Yoo, S., Choi, K., "Optimistic distributed timed cosimulation based on thread simulation model," *Proceedings of the 6th International Workshop on Hardware/Software Codesign*, Seatle, Wa, 1998.

[5] Sung, W., Ha, S., "Optimized timed hardware software cosimulation without roll-back," *Proceedings of the Conference on Design, Automation and Test*, Paris, France, 1998.

Vita

Balbir Dhillon was born in Amritsar, India and lived there until she was eight years old. She moved to the states with her family to purse better education opportunities. She finished her elementary schooling at Lincoln Elementary School and Winston Park Junior High in Palatine, IL. She attended two years at Palatine High School and then moved to Memphis to finish the rest of her schooling. She obtained her Bachelors of Science Degree in Computer Engineering in spring of 2004 from University of Tennessee, Knoxville (UTK). She began graduate school in fall of 2004 at UTK where her major concentration was Digital Communications. She acquired a research position and worked at Oak Ridge National Lab for the duration of graduate school. She obtained her Masters of Science in Electrical engineering in fall of 2005.