Rapid Prototyping of Software Defined Radios using Model Based Design for FPGAs

Sabares Moola Sreedaranath

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

July 22, 2010
Blacksburg, Virginia

Keywords: Software Defined Radio, Model Based Design, Platform Independent Model
Platform Specific Model, IEEE 802.11a, IEEE 802.16e, MIMO, OFDM

Rapid Prototyping of Software Defined Radios using Model Based Design for FPGAs

Sabares Moola Sreedaranath

Abstract

With the rapid migration of physical layer design of radio towards software, it becomes necessary to select or develop the platform and tools that help in achieving rapid design and development along with flexibility and reconfigurability. The availability of field programmable gate arrays (FPGAs) has promoted the concept of reconfigurable hardware for software defined radio (SDR). It enables the designer to create high speed radios with flexibility, low latency and high throughput. Generally, the traditional method of designing FPGA based radios limits productivity. Productivity can be improved using Model based design (MBD) tools. These tools encourage a modular way of developing waveforms for radios. The tools based on MBD have been the focus of recent research exploring the concept of the platform independent model (PIM) and portability across platforms by the platform specific model (PSM). The thesis presented here explores the tools based on MBD to achieve prototyping for wireless standards like IEEE 802.11a and IEEE 802.16e on reconfigurable hardware. It also describes the interfacing of the universal software radio peripheral (USRP2), acting as a radio frequency (RF) front end, with an additional FPGA board for baseband processing.

Acknowledgments

First and foremost, I would like to thank my advisors Dr. Jeffrey H. Reed and Dr. Carl B. Dietrich without whom this work would not have been possible. I would like to thank them for instilling confidence in me to accomplish this task. I could see the journey over the last two years as the most memorable one in terms of learning, knowledge and taking up a challenge with confidence. I have to thank Dr. Dietrich again for getting me all the tools and equipments required for accomplishing this work.

I would also like to thank Dr. Shajedul Hasan and Dr. Tim Newman for providing valuable guidance and advice whenever I went to them with trouble. I really appreciate their patient observation of my findings during the course of this work.

I take this opportunity to thank my ex-colleagues Mr. Satish Vishwanath Bhat, T V Ramalakshman, Kiran Kumar Maheshwarrappa, Prashanth Mayya, Ravichandran Shivakumar, Deepak Menon and Ravi Jandial. Your encouragement helped me to come to this level. The technology I learned while working with you people at Sasken Communication Technologies Limited formed as a solid backbone for my accomplishments in the last 2 years.

On top of all these, I would like to express my gratitude to my father, mother and sister for their wonderful support during my graduate studies.

I had the pleasure of working with wonderful people like Barathram Ramkumar, Dinesh Datla, Haris Volos, Shereef Sayed and Joseph Gaeddert at MPRG. In particular, the interaction I had with Joesph and Haris helped me to understand the basics of MIMO and OFDM. This was instrumental in the success of this work.

Lastly, I would like to thank my friends and roomies who patiently went through my thesis and suggested corrections.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Goals

The primary goals of this work are to implement the signal processing blocks required for building a transceiver chain for MIMO-OFDM on reconfigurable hardware and to explore the use of model based design in implementing these blocks. The model based design flow is an alternative to traditional design flow. The efficiency of employing model based design flow over the traditional design flow for wireless communication system design is addressed in this work.

The signal processing blocks created for MIMO-OFDM are targeted for the SDR platform whose baseband processing element is FPGA. USRP2 will serve as the RF front end required for the MIMO-OFDM based SDR radio. The FPGA present in the USRP2 has limited space for additional signal processing algorithms. Therefore, USRP2 acting as a RF front end is interfaced with the auxiliary FPGA board that performs baseband processing.

The key issue of interfacing the USRP2 directly with the auxiliary FPGA board has not been addressed in any of the existing works. But, the framework required for implementing the same is available in the GNURadio community. The work presented here uses this framework to achieve the interfacing between a USRP2 and the auxiliary FPGA board. This interfacing is necessary to transfer the data samples between the USRP2 and the FPGA board.

This work employs *Alamouti STBC* as a MIMO technique in its physical layer implementation. The reference wireless standards for the implementation is taken from IEEE 802.11a(WiFi/WLAN) and IEEE 802.16e(WiMAX). These standards specify orthogonal frequency division multiplexing (OFDM) as one of its air-interface techniques. Therefore, a MIMO-OFDM physical layer chain is realized in this work. In this work, there is a significant change in the manner these standards are implemented. None of the standards are implemented completely. Only the *physical layer* concepts of these standards are used for the prototyping of the work. The outcome of this work will act as a solid backbone for making the complete standard to work on the FPGA.

1.2 Background Information

The methodology used in this work is based on the rapidly growing field of high-level synthesis tools for the abstraction of architecture of the lower level platforms[1]. One such technique in which most of the high-level synthesis tools are based is *model based design (MBD)*. Tools based on this technique are targeted for embedded systems development, signal processing algorithm development, rapid system integration and analyzing the behavior of complex mechanical systems for a wide variety of use cases. This has resulted in a considerable reduction in the development of lead time and time to market for a wide variety of systems. This has been proven from the case studies of Rockwell Collins, Lockheed Martin, The Mathworks, etc. With the concept of *software defined radio (SDR)* making a significant impact in the military and civilian communities, there is a need to identify the tools and processes that will reduce the development time for these products. The greater part of this work involves using some of the tools based on MBD techniques to design, implement and validate the system based on FPGA for wireless standards like IEEE 802.11a. The system prototype developed in this work acts as a proof-of-concept for the rapid development of future SDR based radios on FPGA platforms. Moreover, the tools used in this work have been shown to produce a model that will be portable across different FPGA platforms, thereby providing portability and scalability for the well proven models. The outcome of this work will also provide a feeder for developing open source intellectual property (IP) cores for the existing wireless standards independent of the target platform used.

In this work, we completely overhaul the design process. The traditional method focused on creating a simulation model, writing register transfer logic or a behavioral model in hardware description language (HDL), synthesizing it and creating a bitstream from it. But, the ramp up period required for this design process is enormous. On top of this, a wireless communication engineer should focus only on developing signal processing algorithms rather than thinking about their portability to platforms like FPGAs. Additional disadvantages include reworking the whole traditional design process when there is a change in the underlying hardware platform. In order to counter this, a high level MBD based architecture was considered for rapid prototyping of SDR for FPGA. The primary examples of MBD tools for FPGA include the Xilinx system generator for DSP^{TM}, Simulink HDL coder^{TM} and Synplify DSP^{TM}. In this work, Simulink HDL coder^{TM} and System generator for DSP^{TM} were used extensively to explore the concept of MBD design flow for SDR.

The MBD design flow gave rise to the idea of creating a high level model based wrapper for integrating the waveform components deployed on the FPGA with SCA based SDR architectures. The methodology for integrating this wrapper onto MBD tools like Simulink^{TM} is presented as future work. The workflow required for this integration is taken from the work done by Carrick [2]. This integration will provide a path for realizing Distributed Wireless computing in SCA based architecture like OSSIE with FPGA as a platform for deploying computation intensive signal processing algorithms.

1.3 Accomplishments

The accomplishments of this thesis are:

- Application of MBD design flow for SDR applications on FPGA.
- High level model designs for various blocks of the transceiver chain for MIMO-OFDM applications.
- Methodology for integrating these blocks to implement a physical layer chain for wireless standards like IEEE 802.11a or IEEE 802.16e on the FPGA.
- A simplified method for interfacing the USRP2 with FPGA development boards.

1.4 Organization

The work presented in this thesis is organized as follows:

- *Chapter 2* describes the basics of *software defined radio* and *model based design*. It briefly introduces the mathematical model behind the mapping of these tools for FPGA based systems. A later section provides an overview of the implications of these tools for SDR based radios.
- *Chapter 3* talks about the different design flows available for performing the task of implementing DSP algorithms on FPGA. It provides a detailed description of the MBD design flow used in this work and a brief description of the tools used in this work.
- *Chapter 4* gives a detailed description of the individual modules required for design for an MIMO-OFDM based system using IEEE 802.11a and IEEE 802.16e standards. It will provide detailed implementation issues along with potential drawbacks in each module. There is a results section associated with each module giving detailed information about the logic resource used in the FPGA.

- *Chapter 5* provides the framework details for implementing these baseband signal processing modules for MIMO-OFDM system in conjunction with USRP2 as a RF front end. It describes the methodology for interfacing the USRP2 with auxiliary FPGA board with PowerPC® processor.

- *Chapter 6* presents the conclusion of this work. It provides remarks about this work and gives directions for further research.

Chapter 2

Software Defined Radios and Model Based Design

2.1 Software Defined Radio

The term software defined radio (SDR) is used to describe radios whose components have been defined with the emphasis on software. The interest in these radios has been driven by technical feasibility, flexibility and commercial advantages.

In general, SDR is a wireless communication system in which the particular communication and transmission characteristics are realized through specialized software running on flexible signal processing hardware. This characteristic is in complete contrast to conventional radio that relies on specific hardware components for realizing the goal of communication. The use of SDR provides flexibility in terms of reuse of components on multiple platforms and using same platform for multiple SDR based communication systems. Other advantages include instant re-configurability in the field and ease of maintenance within the physical limits of underlying hardware.

SDR radios cannot be built in the same manner as conventional radios. They must be built to support a variety of functions like multiple waveforms in the same hardware, instant re-configurability, digitized IP-based data transmission and support for security. The incorporation of these functions creates a model that separates the waveform providers from the platform suppliers. This kind of SDR platform has a lot of relevance in the military, public safety and civilian radio markets.

The volatility of the market for radios and availability of innumerable standards has forced radio manufacturers to think about a cost-effective methodology for the adoption of new wireless standards onto radios. This is crucial from the perspective of the customer who can avoid the exploding costs associated with the upgrading of radios.

The SDR concept has been applied to the military market, in United States of America, through the Joint Tactical Radio System (JTRS) specification that provides inter-operability among radios, used by warfighters and supports multiple waveforms from the multiple defense contractors. In the same manner, there has to be support for the civilian radio market that has seen an influx of radio technologies like GSM, GPRS, CDMA, UMTS, WiMAX, LTE, WLAN, and Bluetooth. A single radio has to support these diverse wireless technologies concurrently that have different requirements for throughput and bandwidth.

SDR radios allow the over-the-air re-configuration of software that will ease the manner in which technology upgrades and bug-fixes can be applied to radio. Moreover, there is an apparent shift in the radio technologies required to support digitized IP based transmission. This makes the shift from point to point links to networking among different radios with diverse radio technologies. Because of SDR flexibility, security functions can be continuously improved to counter new and evolving threats and keep the radio functionality safe. There is cost-effectiveness, when the waveform developer provides flexible waveform application that will run on a wide variety of platforms.

2.2 Model Based Design

The motivation for the *model based design* for FPGAs rose from the necessity of designing complex DSP systems that require dedicated multipliers, compute units for specialized operations like Galois field arithmetic, and Add compare select unit for Viterbi decoding, etc. These dedicated and complex computing units obviated the requirement for finer level optimizations of the FPGA design circuit. This level of optimization is generally associated with traditional digital design for communication systems. MBDs are used to express the typical characteristic of the communication system called *timeliness* that describes the way a system is going to handle the "concurrency, liveliness, heterogeneity, interfacing and reactivity" [1]. The MBD design is basically an attempt to describe the way a system is going to interact with the real time analog world.

The aim of MBD system design is to convert the system model from its mathematical specification to an executable specification. This executable specification helps in realizing a platform independent model (PIM). The PIM is then used to create an elaborate hardware and software specific model for the completion of the system design using a automation code generation tool. Thus, the platform independent model (PIM) is converted into a platform specific model (PSM).

MBD provides a common framework for the integration of different phases of the development process. This reduces the lead time to create a self-sufficient model. The design phases associated with the MBD allow the designer to locate and correct errors prior to system prototyping. Thus, MBD effectively serves as a tool for rapid prototyping, system validation and testing. Moreover, the ability to create hardware-in-the-loop testing during the design development phase enables testing for dynamic effects on the system that is not possible with the traditional design methodology.

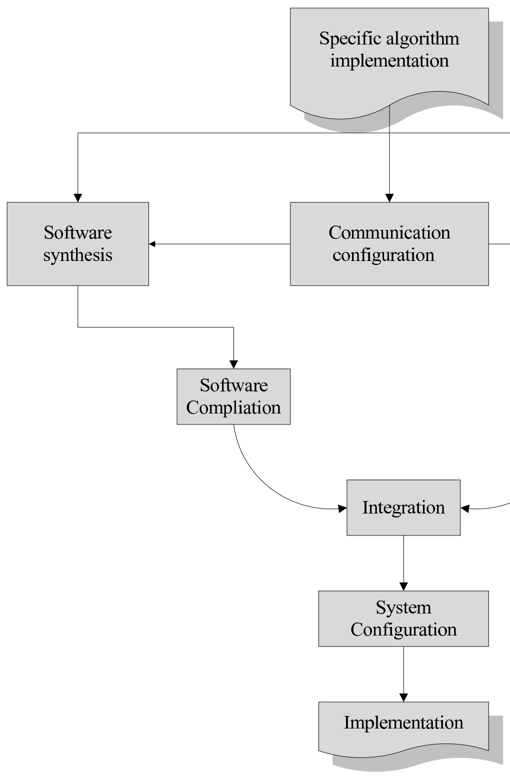


Figure 2.1: Flowgraph for Generic DSP System Design [1]

2.2.1 Dataflow Heterogeneous System Prototyping

Dataflow heterogeneous system prototyping specifies the dataflow modeling required for system design on FGPA or system-on-chip (SOC). The data flow associated with this kind of modeling closely reflects the rapid system design model required for SDR. The motivation behind this modeling is that by closely matching the behavioral semantics of the implementation to the semantics of the high-level specification, a direct translation from the functional domain to the implementation domain can be achieved. This technique can be viewed as the rapid system development and integration technique. The emphasis of the current work will based on this model. MATLAB^{TM}, Simulink®, Xilinx ISE^{TM}, Xilinx System generator for DSP^{TM}, and Simulink® HDL coder ^{TM} are some of the tools that accurately match this design flow. Thus, these tools are used for the rapid integration of baseband algorithms required for SDR. The general requirements for this approach are:

- Modeling domain for the specification.
- A tool for rapid integration.
- Capability for analysis and high-level optimization for the implementation.

The flow of this model is represented in the Figure 2.1. The model relies on high-level algorithm specification that has to be accurate mathematically. In general, one has to take care of floating-point, fixed-point and integer arithmetic operations in the system at the time of modeling itself. After algorithm verification and numerical modeling, the system can be accurately represented by model based design tools like Simulink® HDL coder^{TM} or System Generator for DSP^{TM}. The usage of Simulink® HDL coder^{TM} provides us the ability to create a PIM model like Veriolog/ VHDL code for FPGA. The PIM model can be adapted as a PSM model after further verification for word length effects, and precise numerical modeling required for the target platform. Similarly, for generating a embedded processor code, there is a *real time workshop tool (RTW)* in Simulink® Simulink® HDL coder^{TM} that assists in generating a PSM.

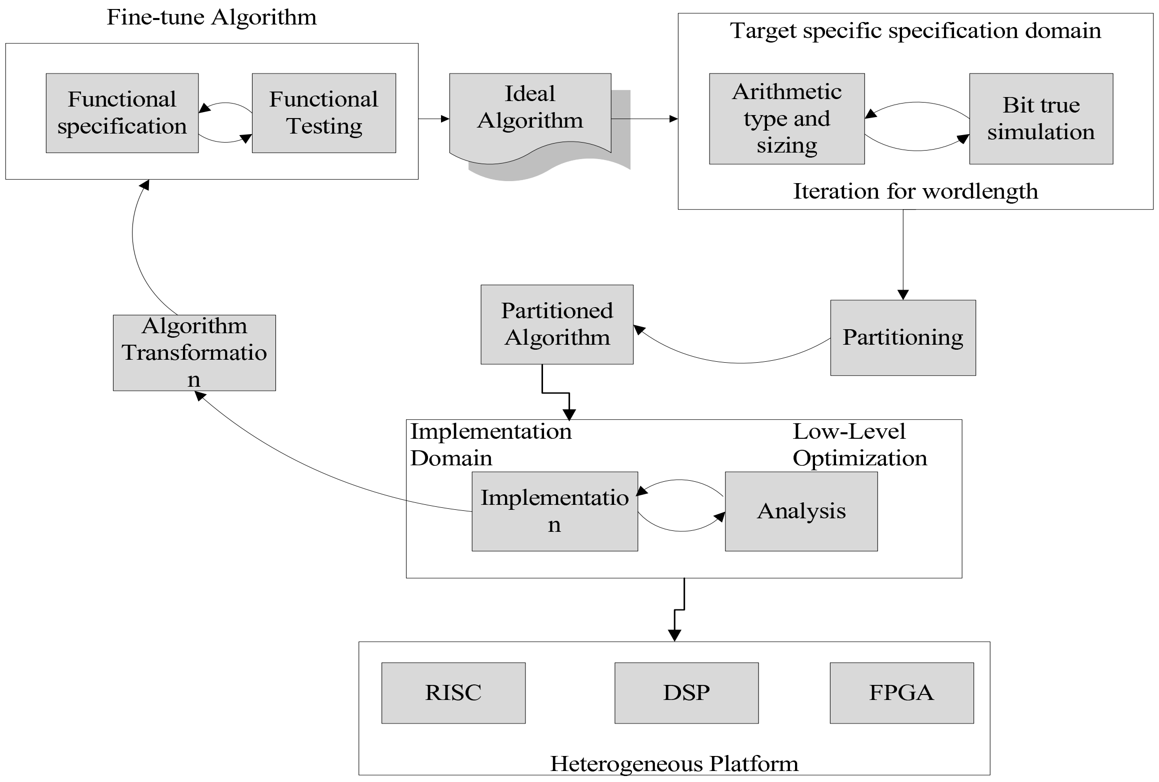


Figure 2.2: Flowgraph for DSP System Design with Heterogeneous Platform [1]

2.2.2 Partitioned Algorithm Implementation

The partitioned algorithm implementation pools the specific tasks that have to be implemented in embedded microprocessors and FPGA. It is also responsible for inter-communication among different hardware components. You can view this as a complete embedded system from a functional point of view. The computational units are pooled according to three major classifications. They are:

- Software synthesis for Embedded DSP/Microprocessors
- Hardware synthesis for FPGA
- Inter-processor communication required for creating a complete fabric

This work focuses on synthesizing a hardware circuit to be built on FPGA. But in a real time scenario a hardware platform is made up of multiple hardware components. The MBD model comes in handy in those situations. Thus, a set of pre-compiled libraries or primitive building blocks can be used to create a system model that should be realizable in hardware or software. The next stage is to create a software and hardware code that can be ported into any platform. In the case of the hardware, the aim is to create a Verilog/VHDL code while in the case of the later the aim is to obtain fixed point C model or floating point model depending upon the target platform. At this stage, the PIM model is readily available for various components. The PIM model is then converted to a PSM model based on design considerations, hardware platform characteristics, and code portability. Finally, we obtain a complete executable code that will be able to run on target hardware platforms.

Thus the whole dataflow model helps us to create rapidly [3, 4, 5] deployable code onto the hardware platform. This allows designers to focus on designing efficient algorithms or an accurate mathematical model.

2.3 Implications of MBD for Software Defined Radio

In this work, the focus is on performing baseband signal processing on stand alone FPGA. The FPGA present in the USRP2 deals with the rest of the operations like up/down conversion, up/down sampling, filtering and transmitting/receiving at the carrier frequency. Thus the USRP2 SDR platform is now leveraged using the additional processing capability of FPGA.

The developmental approach of SDR for FPGA is driven towards MBD because of the necessity to support multiple waveforms in a single SDR platform or supporting a single waveform on multiple hardware platforms. Following a traditional approach for deploying individual waveforms on FPGA is going to result in a significant cost in terms of money and time to market. With this approach, the inherent advantages of SDR are lost. A MBD design approach will allow designers to carry out the rapid prototyping of waveforms on FPGA. It provides a viable approach to implementation and reuse of intellectual property. It allows the designers to reuse the same high level design model for porting to different FPGA platforms

A typical SDR platform will consist of a combination of a general purpose processor (GPP), an embedded DSP processor and an FPGA [6]. Generally, the GPP handles the control and configuration of the software, and the DSP processor handles the low level signal processing algorithm, while the FPGA handles the computationally intensive signal processing algorithms. In the case of the embedded processor/GPP, the software code running on it has to be changed while switching to a different waveform, but in the case of FPGA, the circuit running on it has to be completely reconfigured. Thus the circuit on FPGA provides a co-processor capability that can work in tandem with the GPP and embedded DSP processor. In the heterogeneous SDR platform, there are plenty of options available to distribute the processing across different platforms based on necessity.

The use of MBD design flow allows the designer to focus on algorithm development and optimization rather than concentrating on platform implementation issues.

2.4 Tools for MBD Design

There are a number of tools available for designing MBD-based FPGA systems. A key aspect in these tools is the application of a well-defined MoC language. Most of these tools take advantage of the standardized *unified modeling language* (UML). Since UML differs in the way it defines a system across different domains, these tools differ in the way they explore system characteristics and describe a system. Some implementations may be less efficient than hand-coded ones, but they provides the rapid prototyping of the system, resulting in efficiency in terms of time. This section will describe some of the model based design tools that will help in creating a platform independent code. The selection of tools is dependent on factors such as the level of flexibility, availability of pre-built libraries/blocks, and comprehensive understanding of these blocks.

Some of the tools based on UML include Real time Studio from Artisan, Rhapsody from I-logix, MATLAB® and Simulink® Realtime workshop. These tools are used for designing an embedded multiprocessor environment. There are also modeling languages like GRAPE-II[7], Ptolemy project[6] and GEDAE[8].

The tools available for HDL code generation for the FPGA can be classified as *block based* and *C-based*. Block based tools generate HDL code from the block diagram, which is then fed to the hardware synthesis tool to implement system design in FPGA. Though the approach provides bit-accurate simulation and timing verification, the designer has to take care of the control and timing aspects of the core. Most of the block based tools are based on Simulink® and MATLAB® environments. Some examples include Synplify® DSP, Xilinx's System generator for DSP, Altera's DSP builder and Simulink® HDL coder. These tools provide a high-level modeling environment for signal processing algorithms. Simulink® library blocks are used along with the IP cores of the FPGA vendors to create a platform specific HDL code. The core issues of system implementation like latencies and pipelining are not considered in this model. It is up to the designer to incorporate this into the system model at both the specification and modeling stages. Tools like Simulink® HDL coder^{TM} and Synplify®DSP give more flexibility to the designer by integrating MATLAB® functions, m-block files and state. With these tools, the designer designs the algorithm in the Simulink® environment and slowly refines this step to the FPGA environment in a sequence of steps. An example of this sequence of steps is shown in further chapters.

C-based MBD tools are dependent upon the C programming language to create an abstraction for FPGA design. Some of the tools include Mentor Graphics® Catapult C and Celoxica's Handel-C. The major motivation behind these tools is that the C language being used commonly for DSP algorithm implementation gives a productivity gain compared to Verilog/VHDL code.

Other design tools include Synopsys Behavioral compiler^{TM}, MMAlpha and JHDL. These tools achieve the goal of MBD design flow at different levels. For example, the tool from Synopsys is a behavior synthesis tool that allows designers to quickly evaluate alternate architecture with gate-level optimization and create designs consisting of datapath, memory, and finite state machines. On the other hand, MMAlpha created using C and Mathematica is a programming language to translate the ALPHA programming language. It converts a high-level specification into synthesizable VHDL code. It is programmed using the ALPHA programming language whose syntax allows programming for pipelining, timing, and scheduling. ALPHA programming also allows for defining systolic arrays and can generate HDL code at the RTL level. Contrary to these tools, there is a just-another hardware description language (JHDL) based on Java HDL. It was originally developed by Birmingham Young University. Its motivation is to describe a circuit based on a structural design environment that can be dynamically configured over time. Its major drawback is that it assumes globally synchronous design and does not support multi-clock system designs. Moreover, it does not support behavioral synthesis.

Apart from these tools, there are certain top-level system design tools for FPGA. These include *Compaan, ESPAM, Daedalus* and *Koski* [1]. Compaan and ESPAM tools follow the same design principles and help to convert a sequential programming language like MATLAB® and C/C++ to a parallel process network model by automated code generation tool [1]. The difference between the *Compaan* and *ESPAM* tool is that the former is targeted towards a specific platform design while the later can be targeted for a heterogeneous platform. Both these tools are system level synthesis tools which are useful for rapid system prototyping. The addition of a system level simulation environment on top of an *ESPAM* tool gave rise to a tool called *Daedalus* [1]. Basically, this tool automatically performs mapping from the platform specifications of the ESPAM synthesis tool. *Koski* is a another programming language based on a unified-UML modeling infrastructure for the automatic design exploration and synthesis that is targeted towards wireless sensor network applications. The tool performs synthesis for prototyping the final application onto FPGA

[1].

In this work, extensively used tools were the Simulink® HDL coder^{TM} and the Xilinx System Generator for DSP^{TM}. These two tools played a major role in supporting the architecture design, simulation and synthesis for the target architecture. The key features of these tools are described in the sections below.

2.4.1 Simulink® HDL coder^{TM}

Simulink® is a MBD tool used for modeling, analyzing and simulating dynamically varying systems. It provides a well-defined graphic environment for the designer to create a high-level design of a complex system using the commonly used blocks from Simulink®. Moreover, this tool allows the user to create flexible user-defined blocks from the well-defined MATLAB® functions.

The Simulink® HDL coder^{TM} feature of Simulink® allows the designer to create bit-accurate and synthesizable HDL code from the Simulink® models. These models can include Simulink® models, Stateflow models and Embedded MATLAB^{TM} code. The generated HDL code can be verified using tools such as Cadence® Incisive®, Mentor Graphics® ModelSim®, and Synopsys® VCS®. The HDL code can also be synthesized and mapped onto target the FPGA using tools such as Altera Quartus® II, Cadence Encounter® RTL Compiler, Mentor Graphics® Precision®, Synopsys Design Compiler®, Synplicity® Synplify®, and Xilinx® ISE^{TM}. The Simulink® HDL coder^{TM} also generates a testbench for the purpose of verification and validation with the HDL simulation tools.

The Simulink® HDL coder^{TM} has primitive libraries built for HDL code generation. Some of the pre-built libraries include adder, multiplier, accumulator, integrator, multi-port switch, rate transition block, delays, matrix operation blocks, lookup tables, etc. Most of these libraries acts as building blocks for implementing the higher level signal processing blocks. The application of these libraries for building models will be explained in subsequent chapters.

2.4.2 Xilinx System Generator for DSP^{TM}

The Xilinx System generator for DSP^{TM} is a system level modeling tool targeted towards Xilinx FPGA architectures. It extends the capabilities of the Simulink® environment. As with Simulink® it provides higher level abstractions of the system. This tool can automatically create a low level FPGA code, for Xilinx platforms, at the click of a button. The tool uses in-built DSP IP cores of Xilinx, Inc along with blocks for common functionalities. Its major contrast from the Simulink® HDL coder^{TM} tool is the generation of low-level code targeted towards Xilinx platforms (e.g. Virtex^{TM}). This tool also allows for hardware co-simulation. Hardware co-simulation is used for testing the cores created by the user onto the hardware, in tandem with the model present in the Simulink® environment.

Chapter 3

Design Flow for FPGA Based System Design

The previous chapter described model based design and the tools available for using the same. In this chapter, the focus will be on designing a *FPGA based system*, primarily for signal processing applications, using the concept of *model based system design.* This work uses the Simulink® HDL Coder^{TM} and Xilinx System Generator for DSP^{TM} as the primary tools for MBD based design. The motivation for the FPGA based system-on-chip (SoC) design for wireless applications arises from the expansion of FPGA capabilities of Xilinx and Altera FPGA families. As the silicon density in these FPGA families increases, the complexity of the system design also increases, resulting in the convergence of multiple disciplines and technologies. For example, the work presented here requires expertise in the following areas:

- Signal processing
- Wireless communication theory
- FPGA architecture
- HDL

- Software engineering

Thus, a system designer has to focus on integrating all these components onto a single chip with less flexibility for delving further into system design for functionality and performance. Therefore, a good system design should involve abstraction mechanisms for designing and integrating various components. The designer should focus on the parameters of the components affecting the system performance.

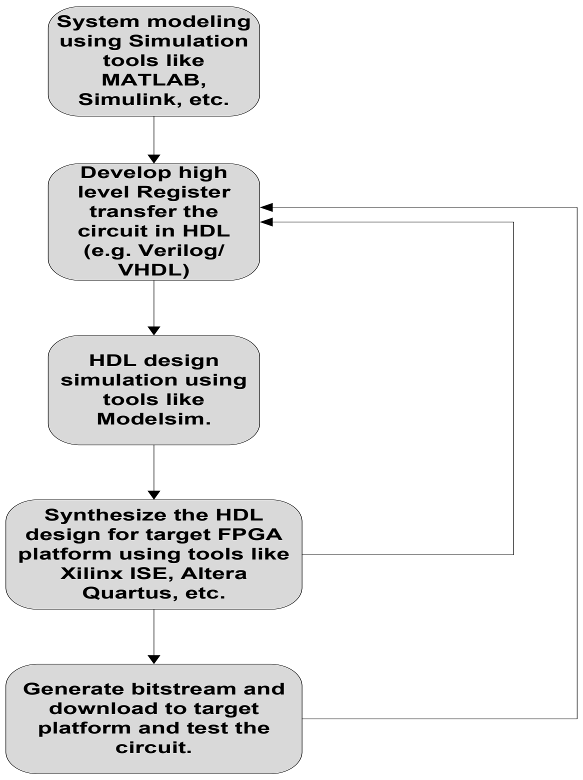

System modeling using Simulation tools like MATLAB, Simulink, etc.
Develop high level Register transfer the circuit in HDL (e.g. Verilog/ VHDL)
HDL design simulation using tools like Modelsim.
Synthesize the HDL design for target FPGA platform using tools like Xilinx ISE, Altera Quartus, etc.
Generate bitstream and download to target platform and test the circuit.

Figure 3.1: Traditional Design Flow

3.1 Traditional Design Flow

As shown in Figure 3.1, the traditional method of designing a system based on FPGA starts with the design specification and simulating/modeling the system based on the specification. The system specification/simulated system design is abstracted into a higher level hardware description by means of Verilog/VHDL. This step is crucial in describing the behavior of the system in the real world but it is time consuming. Finally, the HDL code describing the system, is synthesized, implemented, validated and tested. Though the last few steps are common with any kind of FPGA based system design, the lack of validation of the system at the simulation/modeling level is a major hindrance to a shorter development time. The determination of errors, in terms of timing and functionality during or after the implementation stage, results in a tremendous effort to re-iterate the whole design process. An example design flow is shown in Figure 3.1.

In order to counter the disadvantages of traditional design flow, the MBD design flow that integrates system simulation/modeling and validation in a single step is increasingly adopted. It is described in detail in the next section.

3.2 Design Flow for MBD Based Design

In this work, the MBD design methodology is used to make an open source generic IP core for the common blocks of WiFi/WiMax standards. A typical MBD design flow for the implementation of the transceiver chain on the FPGA is explained in this section. It is represented in Figure 3.2.

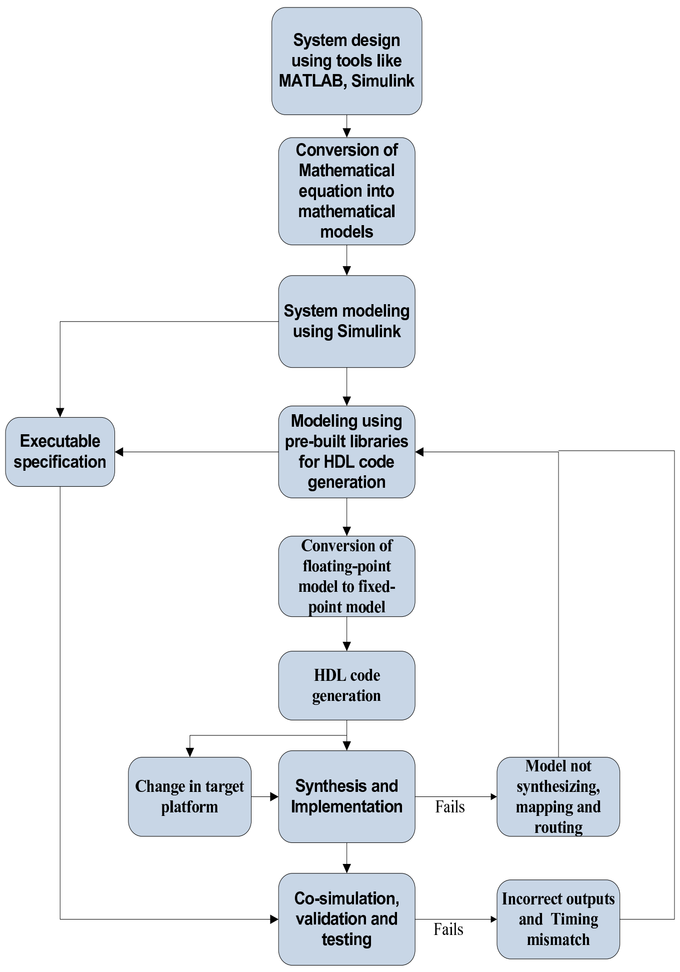

System design using tools like MATLAB, Simulink
Conversion of Mathematical equation into mathematical models
System modeling using Simulink
Executable specification
Modeling using pre-built libraries for HDL code generation
Conversion of floating-point model to fixed-point model
HDL code generation
Change in target platform
Synthesis and Implementation
Fails
Model not synthesizing, mapping and routing
Co-simulation, validation and testing
Fails
Incorrect outputs and Timing mismatch

Figure 3.2: Model Based Design Flow for FPGA

3.2.1 Design Specification

The initial step involved in the design of any DSP block is the design specification and its implication for the rest of the system. This is an abstract way of describing the operation of the block. The parameters obtained from this step includes the type of input or output and identification of the core mathematical functionality of the block. Optionally, the opportunity for *reuse of the design* of the block can be identified.

As an example, consider the development of the convolutional encoder block based on the

IEEE 802.11a standard [9]. In this step, the parameters are the number of inputs, the number of outputs, puncturing, and constraint length, and its operation on a sample based signal or frame based signal. The parameters are shown in Table 3.2.1.

Table 3.1: Design Parameters for Convolutional Encoder

Number of inputs	1
Number of outputs	2
Input type	Boolean
Output type	Boolean
Code rate	1/2 or 2/3 or 3/4
Constraint length	7
Puncturing	Yes/No
Frame/Sample based signal	Sample based signal

Similarly, the design parameters for the fast Fourier transform (FFT) are described in Table 3.2.1.

Table 3.2: Design Parameters for Fast Fourier Transform

Number of inputs	64
Number of outputs	64
Input type	Complex
Output type	Complex
Memory requirement	Dual port memory
Butterfly arithmetic type	Table lookup or CORDIC
Frame/Sample based signal	Frame based signal

3.2.2 Design Requirement Analysis

The core functionality of the block is identified in the preceding step. The next step in the design process is to determine the most suitable algorithm required for implementation. The algorithm should be chosen based on measurable parameters like complexity, computational latency, and logic resource occupancy. The chosen algorithm is then broken down into smaller mathematical functionalities. This plays a critical role in the performance of FPGA in terms of its circuit area and power dissipation.

The *core functionalities* that have to be implemented as IP cores have to be decided here. This core functionality will vary from application to application, and possible additional blocks will need to be added, taking into account future changes in the applications. Examples include constraint length of the convolutional code, number of outputs based on the code rate chosen, number of inputs to FFT, implementation of table lookup and CORDIC based butterfly arithmetic processing, higher throughput and lower memory requirements. This step is necessary to provide generic IP core design.

Once the algorithms for the core functionalities are determined, the effect of fixed-point operations on the chosen algorithm need to be determined. This analysis is performed in this step. The analysis will help in the determination of the optimal wordlength, optimal fixed point format, and scaling required at the various steps of the algorithm computation.

At the end of this step, the input/output format, rounding modes like saturation or overflow, implementation constraints in the form of DSP48 blocks for certain Xilinx devices, scaling factors, number of sub-blocks required for the computation, memory requirement, and FIFO buffer blocks for multi-rate circuit. The latency of each block results from the combination of the above factors.

3.2.3 Simulation and Modeling

The design specification and its implementation specific details now have to be converted into a high level model using MBD design tools. In this work, the majority of the blocks were created using the Simulink® HDL Coder^{TM} and System generator for the DSP^{TM} from Xilinx, Inc. The present step realizes the mathematical equations in the form of mathematical models. The libraries/blocks present in the tool facilitate this representation. The model chosen in this step serves as a platform independent model (PIM) because the model can be targeted for wide variety of platforms. For example, the models shown in Figures B.1, B.2 and B.3 can be targeted for both the embedded DSP platform and FPGA platform using Simulink® Real Time Workshop^{TM} and Simulink® HDL coder^{TM} respectively.

Then the model is tested for functionality using the environment that will best help in validating the model. Generally, observing the output for different input test vectors will suffice. Though Simulink® offers signal processing and communications blocks with a wide variety of functionalities, only certain blocks are supported by its HDL conversion environment. This means that all the blocks that cannot be converted to HDL by the code generation tool need to be removed from the model; it has to be replaced or built using the primitive libraries from the HDL coder tool.

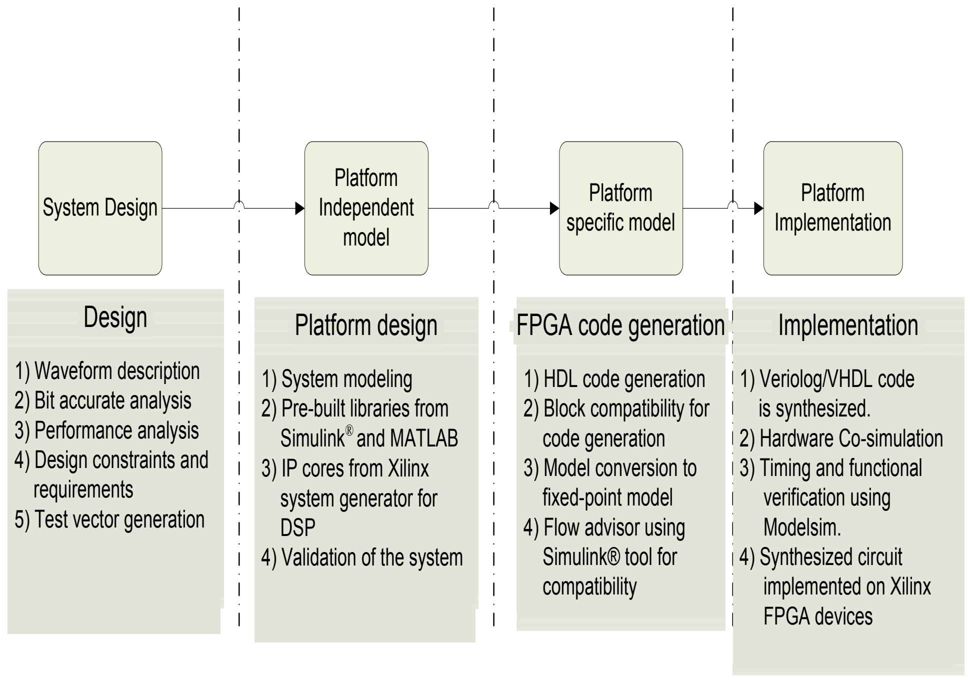
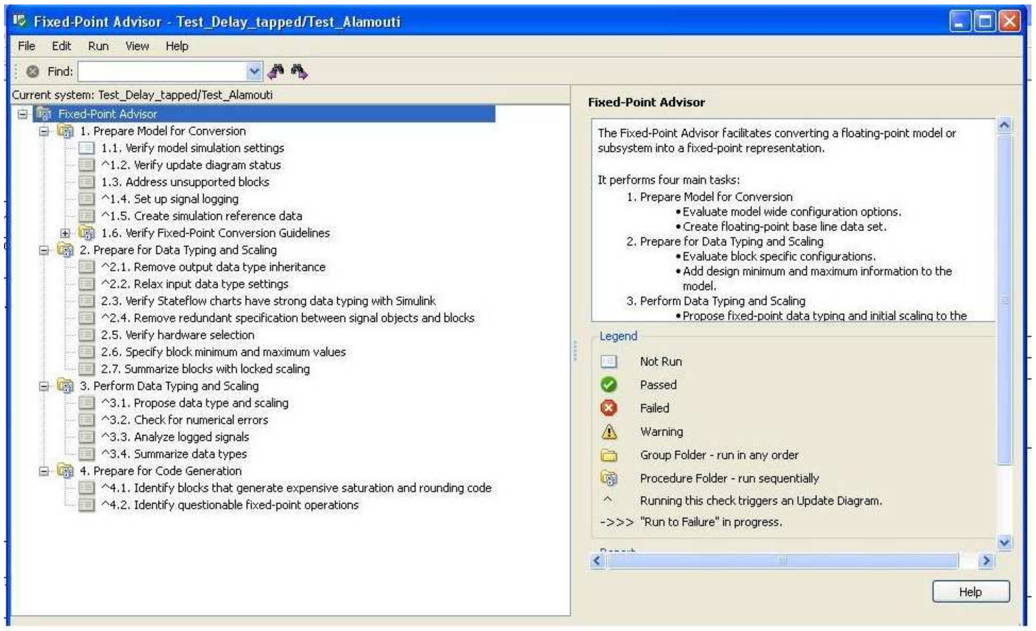

System Design
Platform Independent model
Platform specific model
Platform Implementation
Design
1) Waveform description
2) Bit accurate analysis
3) Performance analysis
4) Design constraints and requirements
5) Test vector generation
Platform design
1) System modeling
2) Pre-built libraries from Simulink® and MATLAB
3) IP cores from Xilinx system generator for DSP
4) Validation of the system
FPGA code generation
1) HDL code generation
2) Block compatibility for code generation
3) Model conversion to fixed-point model
4) Flow advisor using Simulink® tool for compatibility
Implementation
1) Veriolog/VHDL code is synthesized.
2) Hardware Co-simulation
3) Timing and functional verification using Modelsim.
4) Synthesized circuit implemented on Xilinx FPGA devices

Figure 3.3: Model Based Design Flow for Platform Implementation

3.2.4 System Implementation

System implementation is the last and most important step. This step involves the conversion of the PIM model into a platform specific code (PSM), resulting in an executable implementation of the model. It requires the use of an automatic code generation mechanism provided by the MBD design tools. In the case of the Simulink® HDL coder^{TM}, the flow advisor tool helps in checking the compatibility of the models for code generation, conversion of floating-point model into a fixed-point model, potential issues in terms of feedback loops and clock settings, input/output data types in the fixed point model, data scaling, options for generating HDL/Embedded C code, and generating test benches.

During the check for each of the tasks mentioned above, the failure points are identified by the Simulink® tool. The feedback coming from the tool at the failure points helps the designer reiterate the design process for reaching compatibility to generate platform specific code. Furthermore, the Simulink® HDL coder^{TM} helps in performing hardware in-the-loop testing for performing circuit level simulation using blocks like the EDA Simulator Link^{TM}. Once the HDL code has been generated, it can be synthesized in the Simulink® HDL coder^{TM} tool. The Simulink® HDL coder^{TM} supports simulator tools like the Mentor Graphics® Modelsim®, Cadence Incisive, and Synposys Discovery. It also supports the synthesis tools of Xilinx®, Altera®, Mentor Graphics®, Synplicity®, and Synopsys®. The vendor specific tools are used for synthesizing, mapping and placing/routing the IP core on the target FPGA. The tool then displays the logic resources used in the target FPGA platform. The complete flow from the model conversion to the code generation using the Simulink® is shown in Figures 3.4, 3.5 and 3.6.



Current system: Test_Delay_tapped/Test_Alamouti

- Fixed-Point Advisor
  - 1. Prepare Model for Conversion
    - 1.1. Verify model simulation settings
    - ^1.2. Verify update diagram status
    - 1.3. Address unsupported blocks
    - ^1.4. Set up signal logging
    - ^1.5. Create simulation reference data
    - 1.6. Verify Fixed-Point Conversion Guidelines
  - 2. Prepare for Data Typing and Scaling
    - ^2.1. Remove output data type inheritance
    - ^2.2. Relax input data type settings
    - 2.3. Verify Stateflow charts have strong data typing with Simulink
    - ^2.4. Remove redundant specification between signal objects and blocks
    - 2.5. Verify hardware selection
    - 2.6. Specify block minimum and maximum values
    - 2.7. Summarize blocks with locked scaling
  - 3. Perform Data Typing and Scaling
    - ^3.1. Propose data type and scaling
    - ^3.2. Check for numerical errors
    - ^3.3. Analyze logged signals
    - ^3.4. Summarize data types
  - 4. Prepare for Code Generation
    - ^4.1. Identify blocks that generate expensive saturation and rounding code
    - ^4.2. Identify questionable fixed-point operations

Fixed-Point Advisor

The Fixed-Point Advisor facilitates converting a floating-point model or subsystem into a fixed-point representation.

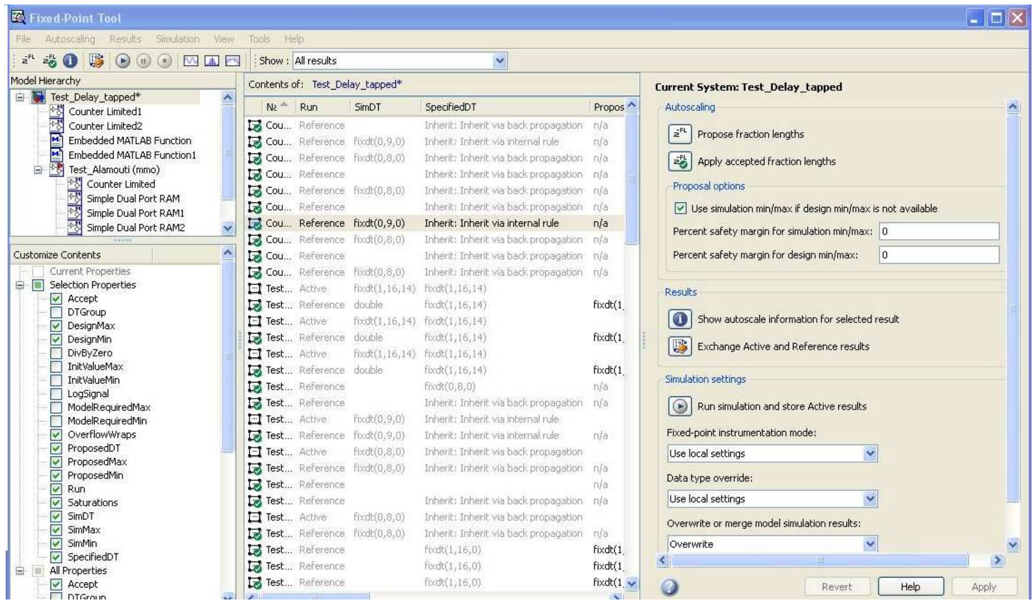
It performs four main tasks:

1. Prepare Model for Conversion
   - Evaluate model wide configuration options.
   - Create floating-point base line data set.
2. Prepare for Data Typing and Scaling
   - Evaluate block specific configurations.
   - Add design minimum and maximum information to the model.
3. Perform Data Typing and Scaling
   - Propose fixed-point data typing and initial scaling to the

Legend

☐	Not Run
✓	Passed
✗	Failed
⚠	Warning
▭	Group Folder - run in any order
▣	Procedure Folder - run sequentially
^	Running this check triggers an Update Diagram.
->>>	"Run to Failure" in progress.

Figure 3.4: Capture of Fixed Point Advisor Tool from Simulink®



Show : All results

Model Hierarchy

- Test_Delay_tapped*
  - Counter Limited1
  - Counter Limited2
  - Embedded MATLAB Function
  - Embedded MATLAB Function1
  - Test_Alamouti (mmo)
    - Counter Limited
    - Simple Dual Port RAM
    - Simple Dual Port RAM1
    - Simple Dual Port RAM2

Customize Contents

- Current Properties
- Selection Properties
  - [✓] Accept
  - [ ] DTGroup
  - [✓] DesignMax
  - [✓] DesignMin
  - [ ] DivByZero
  - [ ] InitValueMax
  - [ ] InitValueMin
  - [ ] LogSignal
  - [ ] ModelRequiredMax
  - [ ] ModelRequiredMin
  - [✓] OverflowWraps
  - [✓] ProposedDT
  - [✓] ProposedMax
  - [✓] ProposedMin
  - [✓] Run
  - [✓] Saturations
  - [✓] SimDT
  - [✓] SimMax
  - [✓] SimMin
  - [✓] SpecifiedDT
- All Properties
  - [✓] Accept
  - [ ] DTGroup

Contents of: Test_Delay_tapped*

Na	Run	SimDT	SpecifiedDT	Propos
Cou...	Reference		Inherit: Inherit via back propagation	n/a
Cou...	Reference	fixdt(0,9,0)	Inherit: Inherit via internal rule	n/a
Cou...	Reference	fixdt(0,8,0)	Inherit: Inherit via back propagation	n/a
Cou...	Reference		Inherit: Inherit via back propagation	n/a
Cou...	Reference	fixdt(0,8,0)	Inherit: Inherit via back propagation	n/a
Cou...	Reference		Inherit: Inherit via back propagation	n/a
Cou...	Reference	fixdt(0,9,0)	Inherit: Inherit via internal rule	n/a
Cou...	Reference	fixdt(0,8,0)	Inherit: Inherit via back propagation	n/a
Cou...	Reference		Inherit: Inherit via back propagation	n/a
Cou...	Reference	fixdt(0,8,0)	Inherit: Inherit via back propagation	n/a
Test...	Active	fixdt(1,16,14)	fixdt(1,16,14)	
Test...	Reference	double	fixdt(1,16,14)	fixdt(1
Test...	Active	fixdt(1,16,14)	fixdt(1,16,14)	
Test...	Reference	double	fixdt(1,16,14)	fixdt(1
Test...	Active	fixdt(1,16,14)	fixdt(1,16,14)	
Test...	Reference	double	fixdt(1,16,14)	fixdt(1
Test...	Reference		fixdt(0,8,0)	n/a
Test...	Reference		Inherit: Inherit via back propagation	n/a
Test...	Active	fixdt(0,9,0)	Inherit: Inherit via internal rule	
Test...	Reference	fixdt(0,9,0)	Inherit: Inherit via internal rule	n/a
Test...	Active	fixdt(0,8,0)	Inherit: Inherit via back propagation	
Test...	Reference	fixdt(0,8,0)	Inherit: Inherit via back propagation	n/a
Test...	Reference			n/a
Test...	Reference		Inherit: Inherit via back propagation	n/a
Test...	Active	fixdt(0,8,0)	Inherit: Inherit via back propagation	
Test...	Reference	fixdt(0,8,0)	Inherit: Inherit via back propagation	n/a
Test...	Reference		fixdt(1,16,0)	fixdt(1
Test...	Reference		fixdt(1,16,0)	fixdt(1
Test...	Reference		fixdt(1,16,0)	fixdt(1

Current System: Test_Delay_tapped

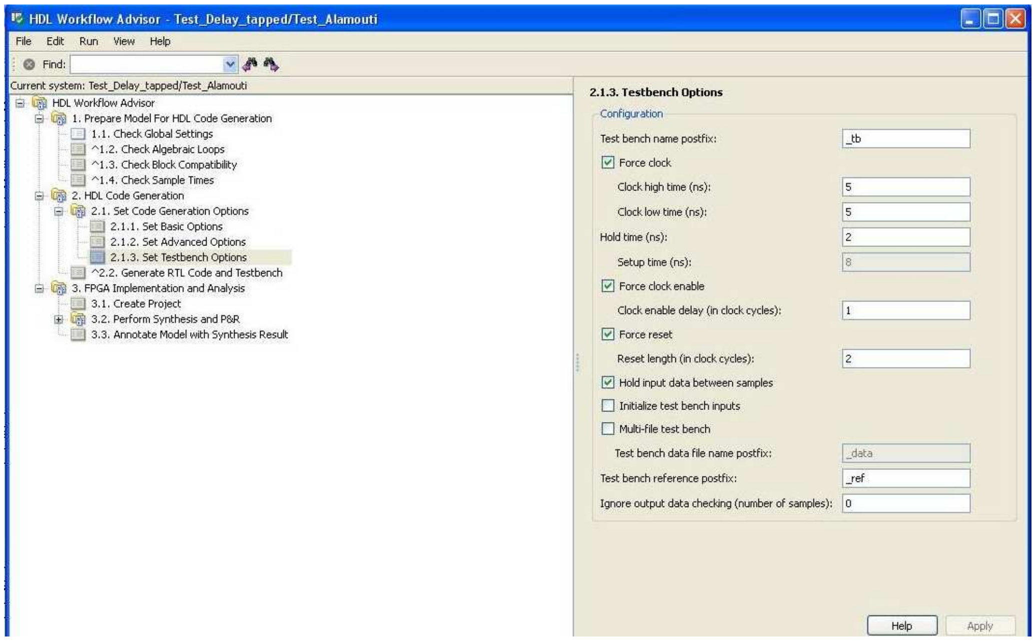
Autoscaling
- Propose fraction lengths
- Apply accepted fraction lengths

Proposal options
- [✓] Use simulation min/max if design min/max is not available
- Percent safety margin for simulation min/max: 0
- Percent safety margin for design min/max: 0

Results
- Show autoscale information for selected result
- Exchange Active and Reference results

Simulation settings
- Run simulation and store Active results
- Fixed-point instrumentation mode: Use local settings
- Data type override: Use local settings
- Overwrite or merge model simulation results: Overwrite

Figure 3.5: Capture of Fixed Point Tool from Simulink®


HDL Workflow Advisor - Test_Delay_tapped/Test_Alamouti

File Edit Run View Help

Find:

Current system: Test_Delay_tapped/Test_Alamouti

- HDL Workflow Advisor
  - 1. Prepare Model For HDL Code Generation
    - 1.1. Check Global Settings
    - ^1.2. Check Algebraic Loops
    - ^1.3. Check Block Compatibility
    - ^1.4. Check Sample Times
  - 2. HDL Code Generation
    - 2.1. Set Code Generation Options
      - 2.1.1. Set Basic Options
      - 2.1.2. Set Advanced Options
      - 2.1.3. Set Testbench Options
    - ^2.2. Generate RTL Code and Testbench
  - 3. FPGA Implementation and Analysis
    - 3.1. Create Project
    - 3.2. Perform Synthesis and P&R
    - 3.3. Annotate Model with Synthesis Result

2.1.3. Testbench Options

Configuration

Test bench name postfix:	_tb
[✓] Force clock	
Clock high time (ns):	5
Clock low time (ns):	5
Hold time (ns):	2
Setup time (ns):	8
[✓] Force clock enable	
Clock enable delay (in clock cycles):	1
[✓] Force reset	
Reset length (in clock cycles):	2
[✓] Hold input data between samples	
[ ] Initialize test bench inputs	
[ ] Multi-file test bench	
Test bench data file name postfix:	_data
Test bench reference postfix:	_ref
Ignore output data checking (number of samples):	0

Help Apply

Figure 3.6: Capture of HDL Workflow Advisor Tool from Simulink®

As mentioned in the previous section, this work also utilizes the other tool called Xilinx System generator for DSP^{TM}. This tool generates IP cores specifically targeted for Xilinx FPGA devices. It uses high level Simulink® models along with Xilinx blocksets for widely used signal processing functions, memories, error correction, and digital logic. Additionally, it supports Ethernet and JTAG communication between the hardware development board and Simulinks®. This provides the necessary communication link for real-time hardware verification and testing. The data type required for the model has to be predetermined while using the Xilinx System generator for DSP^{TM}. The major advantage of using this tool is the availability of IP cores developed by Xilinx, Inc. For example, some of the cores like the first in first out (FIFO) generator and FFT/IFFT are modeled and tested using this tool. In addition, the core for Ethernet communication called the tri-mode Ethernet medium access control (TEMAC) wrapper is used from the Core Generator^{TM} tool to make an Ethernet link between a RF front end and FPGA development board. By using the in-built IP core, the tool can take advantage of the hardware characteristics of the target FPGA device.

The target hardware platform used for testing and validation is the Xilinx ML403 development board. It uses the Virtex^{TM}-4 FPGA device with a PowerPC® hard core running on it. Its operational clock speed should meet the demand for baseband processing required for the IEEE802.11a/IEEE802.16e standard.

Chapter 4

Space Time Block Coding and OFDM System Design

4.1 MIMO

The rapid growth of mobile communication along with scarce availability of radio spectrum, has forced wireless communication engineers to come up with a technique that will provide high data rate services, measured in bits per second (bps), along with high spectrum efficiency, measured in bps/Hz. *Multiple input multiple output*(MIMO) antenna techniques help in achieving the high data rate needed for mobile services in a capacity constrained environment [5]. It also provides effective tolerance against fading and multipath by means of transmit and receiver diversity. The theoretical background for the MIMO technique was developed by Teletar [3] and Foschini [4]. Further work from Tarokh et al [10] and Alamouti [11] opened up new avenues of research in this area. The high point of MIMO technology was the demonstration of Bell laboratories that introduced Bell laboratories layered space time code (BLAST) coding technique [12], achieving spectral efficiency as high as 42 bps/Hz.

This chapter will describe, in brief, the MIMO technique. The chapter also covers the implementation details of the various components involved in designing a system with the *Alamouti STBC* MIMO technique. Current work uses the IEEE 802.11a and IEEE 802.16e standards as the reference transceiver chain. Some of the blocks implemented based on these standards are the convolutional encoder, Viterbi decoder, Alamouti encoder and decoder, modulation schemes, and fast Fourier transform/inverse fast Fourier transform. Apart from these blocks a few other blocks were used for completing this system. This includes the CORDIC algorithm IP core for performing trigonometric operations, FIFO buffers for interleaving and multi-clock systems, and digital clock managers (DCM) blocks for managing systems with multiple clocks. All the individual blocks required for the transceiver chain have been designed using MBD design methodology. Therefore, the MIMO system implemented in this work can be ported to a multitude of platforms. Some Xilinx proprietary IP cores that were used in this work are the DCM manager and FIFO blocks. The induction of proprietary cores along with the MBD design created by the user indicates that the conversion of PIM to PSM has to satisfy certain characteristics of the target platform.

Before describing the components designed in this work, the basic technology behind a MIMO system is explained in the next few sections. They are described in order to appreciate the effectiveness of the MIMO system.

4.1.1 Diversity Gain

Diversity techniques are generally employed to combat multipath fading. Diversity techniques involve transmitting replicas of the signal over frequency, time or space. In MIMO systems only spatial diversity is achieved. The combination of the Alamouti STBC with OFDM results in diversity gain over both frequency and space. The three different types of diversity schemes are explained below.

4.1.1.1 Temporal Diversity

Temporal diversity involves the transmission of replicas of the signal over time. Primary examples include channel coding and interleaving. Temporal diversity can be applied in those cases where the coherence time of the channel is much smaller than the symbol duration. The above criteria makes the symbols transmitted across time independent of each other.

4.1.1.2 Frequency Diversity

In the case of frequency diversity, the replicas of the signal are transmitted across different frequencies. This is applicable in those cases in which the coherence bandwidth of the channel is much smaller than the bandwidth of the signal. This condition is necessary to make sure that the signal transmitted across different frequency bands suffers independent fades.

4.1.1.3 Spatial Diversity

Spatial diversity refers to the use of multiple transmit or receive antennas. The multiple copies of the signal are transmitted from different antennas. This technique allows for the exploitation of the signal in space and time. The only condition required for this case is that the spacing between antenna elements is larger than the coherent distance, so the signal from different antennas undergoes independent fades.

The spatial diversity can be categorized based on the technique applied at the transmitter or receiver. The transmitter diversity can be taken advantage of by advanced signal processing algorithms at the receiver. The addition of controlled redundancies at the transmitter can be exploited at the receiver. This technique relies on complete channel information at the transmitter. The use of space-time block coding techniques like Alamouti [11] made it possible to implement transmit diversity *without any knowledge* of channel information [5].

In the case of receiver spatial diversity, well known algorithms like *Maximal ratio combining*, *selection combining* and *equal gain combining* are used. It is not practically feasible to implement these techniques at thes mobile receiver. Thus transmit diversity techniques became popular due to their ease of implementation at the base station.

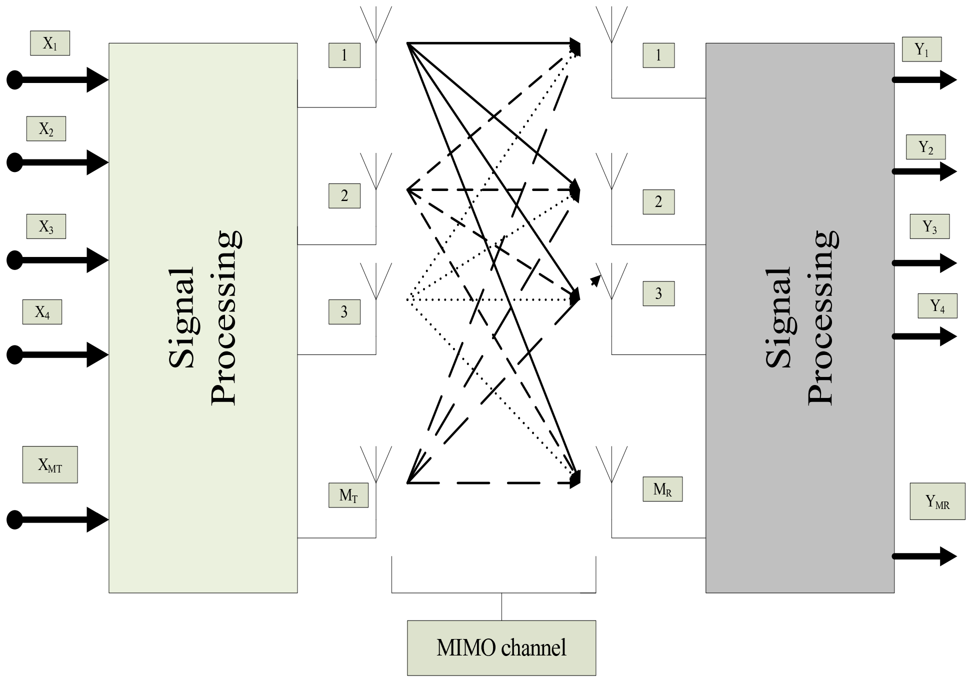


Figure 4.1: MIMO System Model [5]

4.2 MIMO-OFDM

As mentioned before, the signal processing blocks required for constructing a transceiver chain based on MIMO-OFDM are implemented in this work. The primary reason for choosing OFDM, as an air interface, is due to its simplicity of implementation. Additionally, current wireless standards for WLAN/WiMAX/LTE specify OFDM as its primary or as one of its air interface technique. The MIMO block implemented in this work is Alamouti STBC [11].

4.2.1 OFDM

OFDM is a multi-carrier modulation technique that employs narrow band wireless channels for combating communication impairments. The spacing between the sub-carriers is required to maintain orthogonality that allows for overlap of the sub-carrier frequencies. This results in high spectral efficiency. The efficiency in spectrum utilization made OFDM as preferred air-interface technique in most of the current wireless standards. Additionally, the ease of implementation of the OFDM modulator using fast Fourier transform techniques is one of the major strengths of the OFDM technique. The addition of cyclic-prefix plays a major role in reducing the effects of multipath fading. The sub-carrier spacing is chosen in relation to the coherence bandwidth of the channel. OFDM converts the wide-band frequency selective channel into narrow band flat fading channel. This obviates the necessity for complex equalization techniques at the receiver. For example, in the IEEE802.11a standard, the sub-carrier frequency spacing is fixed at 0.3125MHz. .

A simplified diagram of the OFDM modulator is shown in Figure 4.2. The modulator consists of an IFFT that takes N complex data symbols coming out of a phase shift keying (PSK)/ quadrature amplitude modulator (QAM) and converts it into an equivalent N-point time domain signal. The generated OFDM symbol can be written as,

$$\mathbf{X}_k = \sum_{n=0}^{N-1} \mathbf{x}_n exp^{\frac{j2\pi kn}{N}} \quad k = 0, 1, 2, ...N-1 \tag{4.1}$$

where $\mathbf{x}_n$ is the PSK/QAM modulated symbol, $\mathbf{X}_k$ are the time-domain coefficients, and N is the number of data symbols per OFDM symbol.

The cyclic prefix (CP) is added to the front of each OFDM symbol as shown in Figure 4.2 to mitigate inter-symbol interference (ISI). The length of the CP is directly related to the length of the channel. Since the cyclic prefix is a repetition of the last few samples of the data part of OFDM symbol, there is an integer number of cycles inside each OFDM symbol. This repetition allows the FFT operation to start at any point inside the cyclic prefix at the

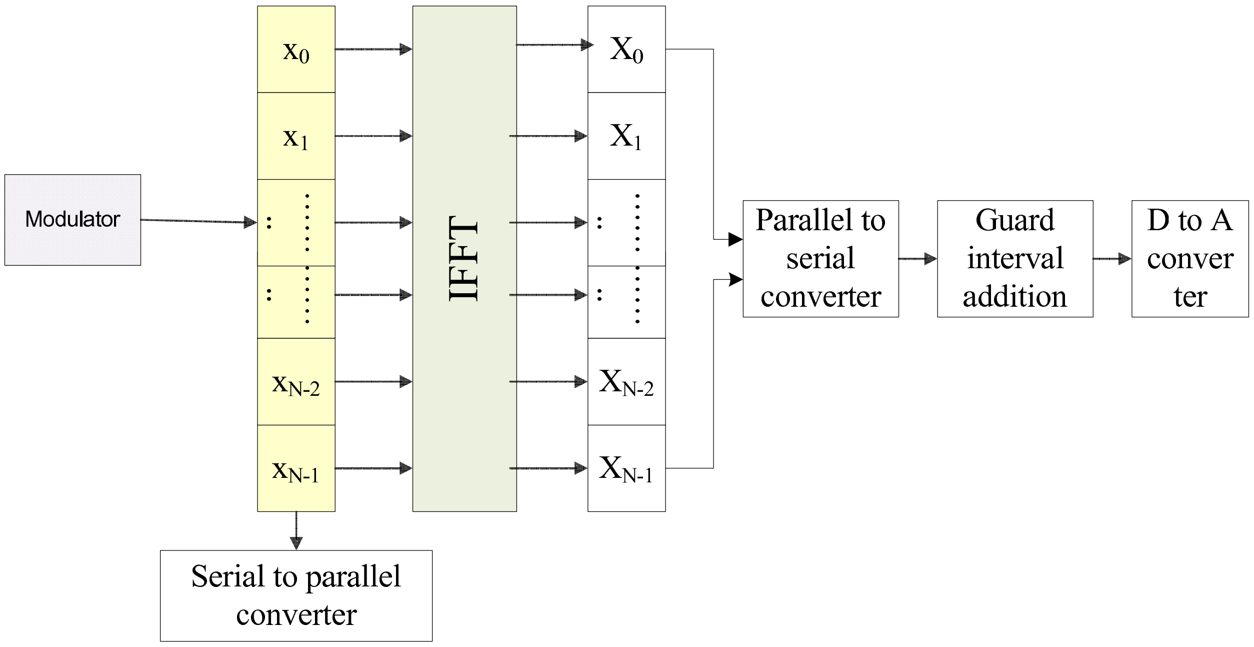


Figure 4.2: Block Diagram for OFDM Symbol Generation

receiver. By keeping the length of the cyclic prefix larger than the length of the channel, the *linear convolution* of the transmitted sequence with discrete-time channel has now been converted into a *circular convolution*.

At the receiver, the symbols are demodulated after the FFT operation. Then the symbol de-mapper is used to detect the transmitted data bits. The receiver part also has complex synchronization mechanisms (both time and frequency), channel estimation, and phase correction mechanisms. The blocks that make up a transceiver chain for MIMO-OFDM is shown in Figure 4.3 and Figure 4.4.

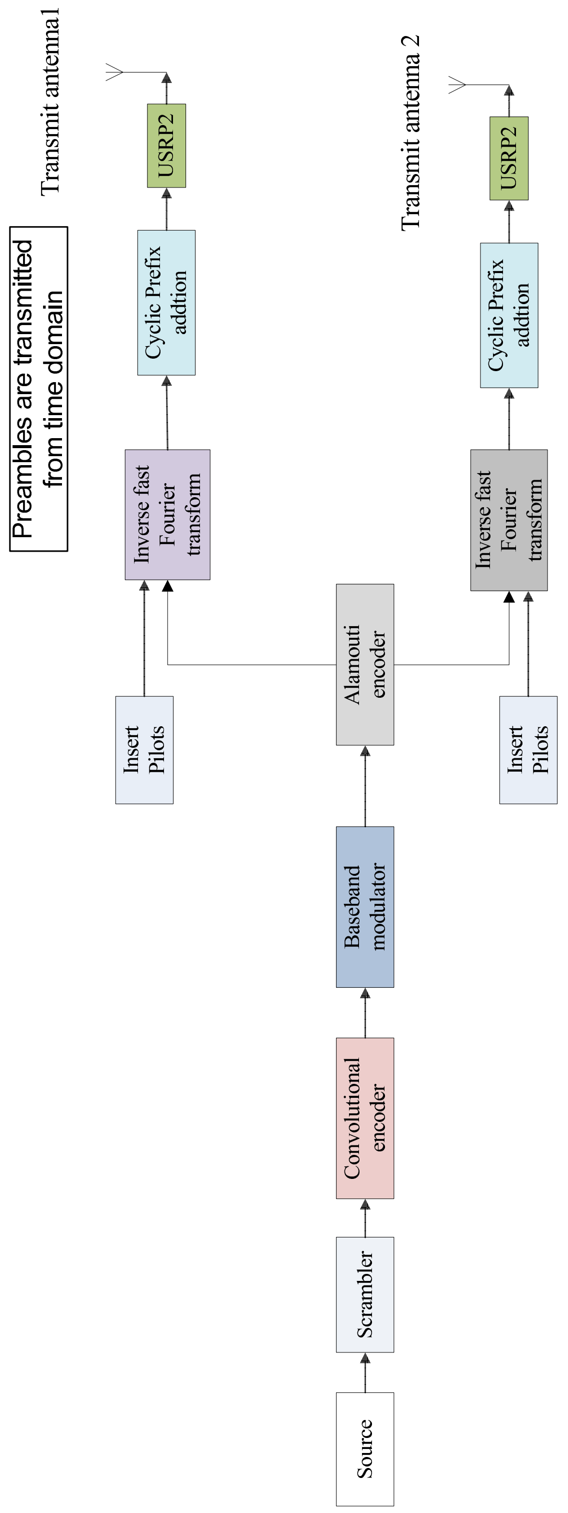


Figure 4.3: Trasmitter for MIMO-OFDM

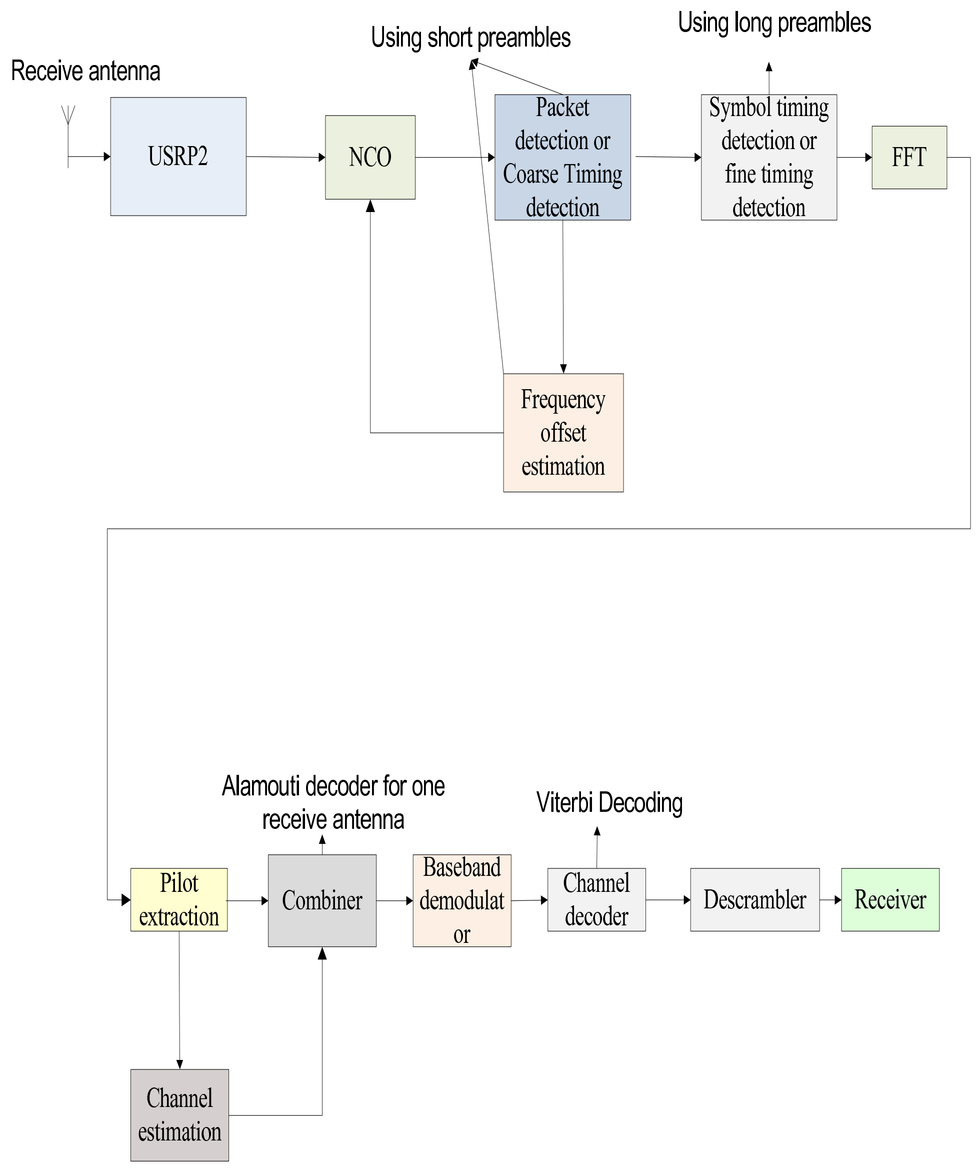


Figure 4.4: Receiver for MIMO-OFDM

4.3 System Overview

This section describes the overall system design employed in this work. This system design forms as an implementation platform for various blocks described in the previous section. The MIMO-OFDM system model is realized using the USPR2's and the Xilinx ML403 development board. The USRP2 is acting as an RF front end and is connected via Gigabit Ethernet cable to the Xilinx ML403 development board. The FPGA present on the ML403 development board is used for baseband processing. On the current system model, there are two USRP2s on the transmit side and one USRP2 on the receive side. In the transmit side, the synchronization of USRP2's is done using a MIMO connector cable. The MIMO connector cable is an auxiliary cable provided by Ettus Research to synchronize multiple USRP2s with a common reference clock. This synchronization can also be achieved by an externally generated clock.

On the receiver side, the USRP2 down converts the RF signal into the baseband signal. The data type of the baseband signal that is buffered to the host machine is configurable using the FPGA of the USRP2. The current work uses 16-bit I and Q data as baseband samples. The ML403 board acts as a proper embedded device with the PowerPC^{TM} processor and various peripherals attached to it. Further details about the ML403 board is given in the next chapter. Details about the interfacing between the ML403 board and USRP2 are also given in the next chapter.

The subsequent sections will describe the baseband elements required for implementing the complete transceiver chain. The wireless standards IEEE 802.11a and the OFDM mode of IEEE 802.16e have identical baseband processing elements. They differ in certain implementation parameters. Some of these include the number of data points for FFT/IFFT operation, the type of channel coding techniques, and the occupying bandwidth. In this work, most of the elements are initially targeted for the IEEE 802.11a standard with scalability options for the IEEE 802.16e standard.

At the end of this chapter, the advantages of using MBD design tools like the Simulink® HDL coder^{TM} and the Xilinx System Generator for DSP^{TM} for designing a complex DSP system will be clearly demonstrated.

4.4 Alamouti STBC

In this work, the Alamouti STBC scheme is incorporated into the physical layer design. Alamouti STBC is a simple transmit diversity technique with two antennas at the transmitter and one or more antennas at the receiver. Advantages include absence of feedback information from the receiver to transmitter, no bandwidth expansion, and attainment of full rate encoding. It provides identical performance as the maximal ratio combining (MRC) scheme provided that the total radiated power from both the antennas is double that of the MRC scheme [11]. The performance curves are shown in Figure 4.6(a) and 4.6(b). The Alamouti decoder at the receiver employs a simple maximum likelihood (ML) decoding scheme. The Alamouti scheme provides the diversity gain of $2 \times M_R$ without the channel information at the transmitter. The Alamouti model used in the current work is shown in Figure 4.5.

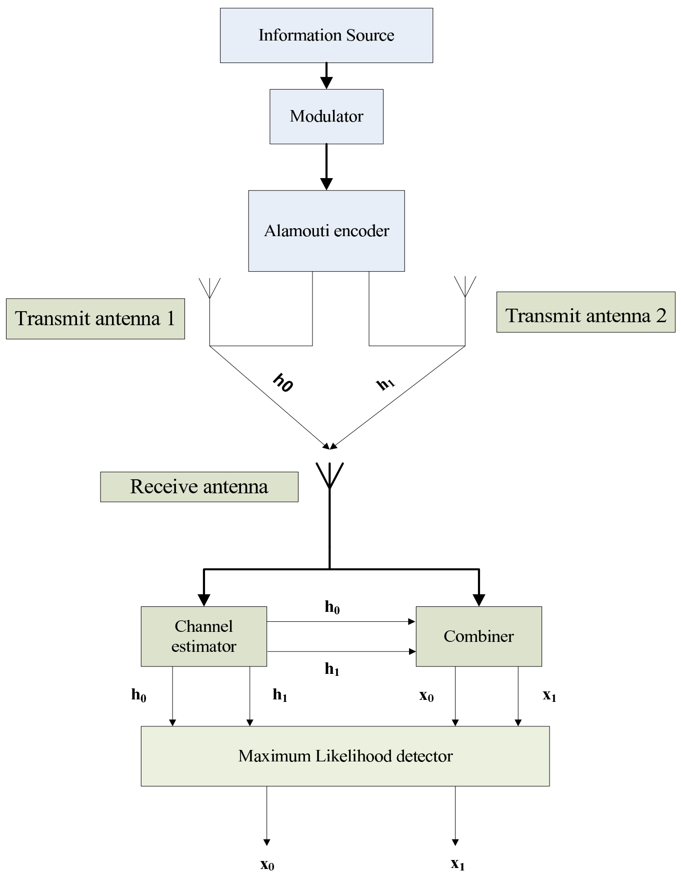


Figure 4.5: Alamouti Encoding Scheme with One Receiver
[11]

In Alamouti encoding, 2 symbols are transmitted over 2 symbol periods. Thus the *rate of the code* of the Alamouti scheme is 1. The symbols transmitted from 2 antennas in two consecutive symbol durations is shown in Table 4.1.

Because of the above arrangement of symbols over two consecutive symbol durations, the symbols are orthogonal to each other. This orthogonality principle helps in decoding at the receiver. From Figure 4.5, the channel between the transmitter and the receiver can be defined as in Table 4.2.

The terms h_0 and h_1 represent complex multiplicative distortion for the symbols received at the receiver antenna Rx from the transmit antenna Tx1 and Tx2, respectively. The basic

Table 4.1: Alamouti Encoding at the Transmitter

	Symbols transmitted from Antenna 1	Symbols transmitted from Antenna 2
Transmitted symbol at time t	x_0	x_1
Transmitted symbol at time t+T	$-\mathrm{x}_1^*$	x_0^*

Table 4.2: Definition of Channel Between the Transmitter and Receiver

	Receive antenna
Transmit antenna 1	h_0
Transmit antenna 2	h_1

assumption is that the channel remains constant over two consecutive symbol durations and it can be modeled as follows [11]:

$$\begin{aligned} h_0(t) &= h_0(t+T) = h_0 = \alpha_0 e^{j\theta_0} \\ h_1(t) &= h_1(t+T) = h_1 = \alpha_1 e^{j\theta_1} \end{aligned} \tag{4.2}$$

Then the received signal at the receiver at time t and $(t+T)$ can be expressed as [11]:

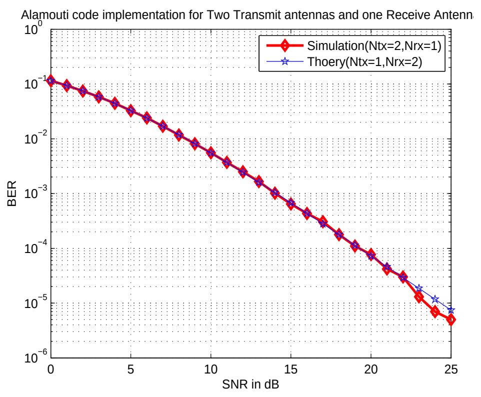
$$\begin{aligned} \mathbf{r}_0 &= \mathbf{h}_0\mathbf{x}_0 + \mathbf{h}_1\mathbf{x}_1 + \mathbf{n}_0 \\ \mathbf{r}_1 &= -\mathbf{h}_0\mathbf{x}_1^* + \mathbf{h}_1\mathbf{x}_0^* + \mathbf{n}_1 \end{aligned} \tag{4.3}$$

where $\mathbf{n}_0$ and $\mathbf{n}_1$ are random variables representing noise and/or interference.

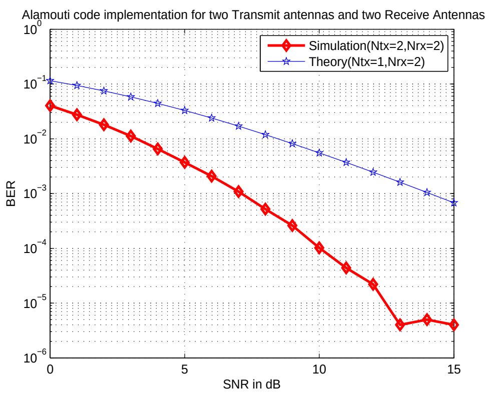
As shown in Figure 4.5, the combiner combines the received signal in the following manner,

$$\begin{aligned} \widehat{\mathbf{x_0}} &= \mathbf{h}_0^*\mathbf{r}_0 + \mathbf{h}_1\mathbf{r}_1^* \\ \widehat{\mathbf{x_1}} &= \mathbf{h}_1^*\mathbf{r}_0 - \mathbf{h}_0\mathbf{r}_1^* \end{aligned} \tag{4.4}$$

The resulting combined signal is then sent to the *ML detector* for making decisions about the symbol. The decision rule for the *ML detector* is based on the modulation scheme employed. The performance curves shown in Figure 4.6(a) show that the diversity order obtained from the two-branch Alamouti encoding scheme is the same as the two-branch MRC combining scheme.



(a) With one receive antenna



(b) With two receive antenna

Figure 4.6: Simulated and Theoretical Performance of Alamouti Transmit Diversity Scheme

The same scheme can be extended for two or more antennas with considerable increase in performance when compared to the MRC combining scheme [11]. The performance curves for the configuration of two transmit antennas and two receiver antennas is shown in Figure 4.6(b). The block diagram for the Alamouti scheme with two transmit antennas and two receive antennas is shown in Figure 4.7.

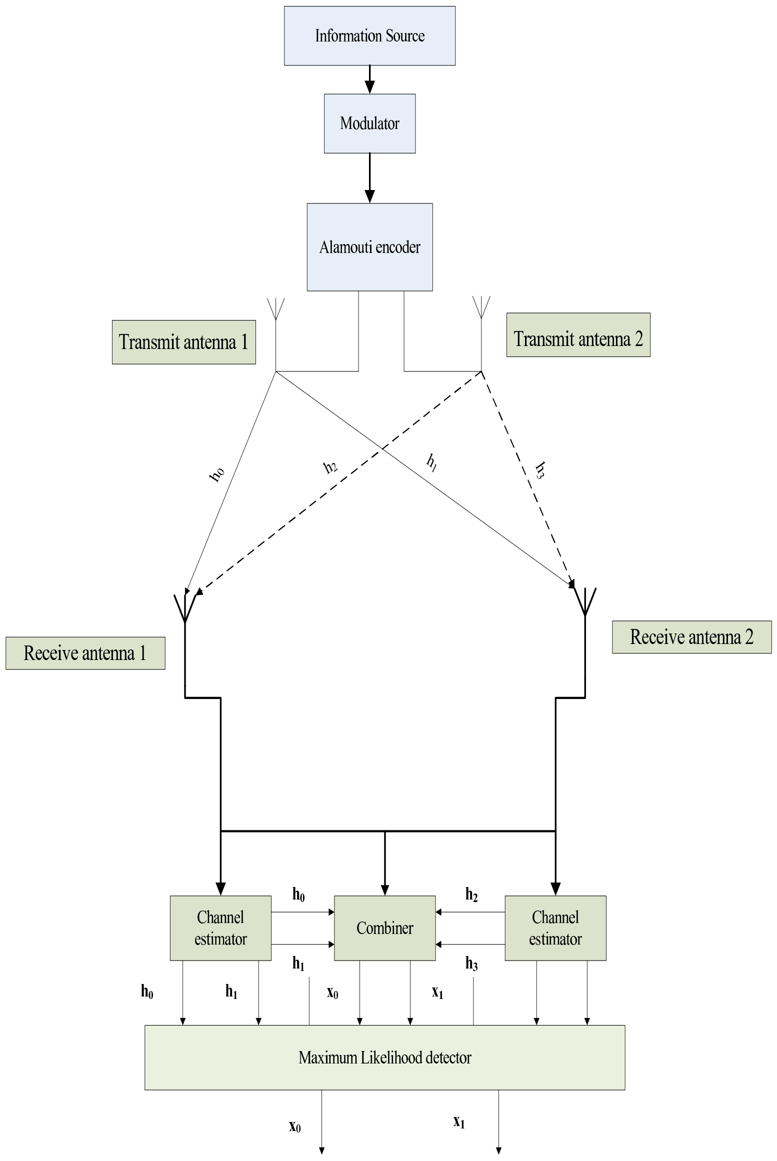


Figure 4.7: Alamouti Encoding with Two Receiver Antennas

4.4.1 Implementation

In this work, the high level modeling for the Alamouti encoder was realized using Simulink® which is then converted into the HDL code. After synthesizing the HDL code, the bitstream is obtained using the vendor specific tool. The bitstream was flashed onto the FPGA of the

Xilinx ML403 development platform and tested.

The implementation of Alamouti encoding at the transmitter is shown in Figure B.1. It uses the *multi-port switch blocks* to switch the symbols to be transmitted across the antennas. The whole design has been split to handle I and Q samples individually. There is a Dual port random access memory (RAM) that stores the incoming elements. The Dual port RAM helps in reading and accessing the memory elements at the same time. Generally, the depth of the RAM is chosen based on the specific application. In this case, the depth is chosen to accommodate 64 incoming samples corresponding to the FFT/IFFT depth. There is a *subsytem* block that controls the operation of the whole Alamouti system block. The subsystem block controls the timing and the read/write addresses.

The block diagram built out of the Simulink® is shown in Appendix B. The sub-blocks shown in Figure B.1 are pre-built libraries from Simulink® that produce synthesizable HDL code using the HDL code generation tool. It uses the dual-port RAM to store the incoming data streams as well as the outgoing data streams to the IFFT block. The generated HDL code is tested for functionality and timing using the ModelSim® simulation tool. The results are shown in Figure D.2. After this step in the validation process, the HDL code is synthesized and the bitstream is generated for downloading onto a target FPGA. The logic resources consumed in the target platform are shown in the following section.

4.4.2 Results

The sample output waveform from the Simulink^{TM} and Modelsim® is shown in Figure C.2 and Figure D.2 respectively. The logic resource consumed by this module along with its timing constraints are shown in Table 4.3 and Table 4.4 respectively. The target of this module is to produce 2 outputs per clock cycle corresponding to two antennas. But in this case, the samples are I and Q samples. Therefore, a Dual port RAM is used to hold the incoming samples until the processing of the previously arrived I and Q samples. The buffering and concurrent processing at the different switching elements helps in achieving the target of two outputs (I and Q samples dealt separately) per clock cycle.

Table 4.3: Logic Resources Consumed by Alamouti Encoder in **XCV4FX12**

Logic utilization	Used	Available	Utilization
Number of Slice Flip flops	148	10944	1%
Number of 4-input LUTs	253	10944	2%
Number of occupied slices	172	5472	3%

Table 4.4: Timing Summary for Alamouti Encoder in **XCV4FX12**

Minimum period	2.888ns
Maximum frequency	346.278 MHz
Maxim delay	6.554ns

4.5 Fast Fourier Transform

In any OFDM system, IFFT/FFT plays an important role in modulating/demodulating the multiple sub-carriers onto/from a single OFDM symbol. It takes up a major chunk of the circuit as well as power consumption. FFT/IFFT can be viewed as a mathematical operation to perform discrete Fourier transform (DFT). Then the N-point DFT can be represented as:

$$\mathbf{X_k} = \sum_{n=0}^{N-1} x_n e^{\frac{j2\pi n}{N}} k = 0, 1, 2, ...N-1 \tag{4.5}$$

Implementing this formula directly requires complexity to the order of O_{N^2}. Therefore, Cooley-Turkey proposed the algorithm that decomposes the whole DFT operation into a number of smaller operations that can be computed recursively. An example of this algorithm is the computation of a 2-point DFT recursively. This operation is referred to as radix-2 FFT. There are two types of FFT: Decimation-in-time (DIT) FFT and Decimation-in-frequency

(DIF) FFT. In the case of DIT FFT, the input sequences are not read in sequential order, while in the case of DIF FFT the output sequences are not read in sequential order.

Both DIT and DIF reduces to a primitive arithmetic operation called "butterfly" arithmetic. This involves additions, subtractions and a complex multiplication. For computing a N-point DFT through FFT there will be $\log_2 N$ stages with each stage involving $\frac{N}{2}$ butterflies.

In this work, memory based architecture is employed for performing the IFFT/FFT operations. There is an input memory, a computational unit, and an output memory for storing re-ordered bits. The basis of this work is the model provided by Simulink^{TM}. A 64-point FFT model is constructed based on this reference model.

4.5.1 Implementation

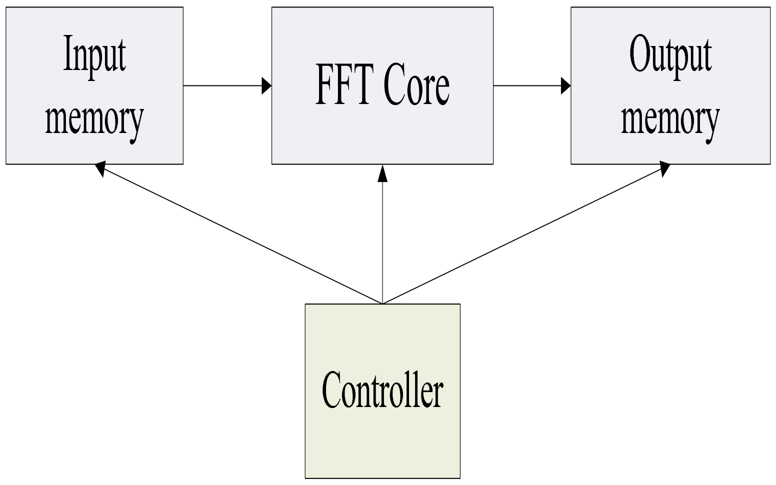


Figure 4.8: Block Diagram of FFT/IFFT Design

The FFT/IFFT can be implemented through streaming mode, serial mode, or burst mode. In the case of the *streaming mode*, the data to be transformed is received at the rate of one data per cycle and then de-serialized (converting serial data to parallel data). Dual port RAM is used to perform the de-serialization operation that helps store the previous data values and read the currently written values. The writing/reading to/from the same memory location behaves differently in different target FPGA devices. Therefore, the model created as PIM should reflect the target architecture before converting to PSM. For example, in Virtex^{TM} devices, reading and writing at the current location will result in reading the old value while writing the new value. The author would like to acknowledge the work of The Mathworks^{TM} Inc. The basic model created for this work is based on the sample provided by the company.

Implementation of this FFT is divided into stages. The number of stages is dependent upon the number of points of FFT. It is possible to implement a single stage to do all the butterfly arithmetic. The FPGA circuit created for the single stage is then used for the rest of the stages with iterations running over it. This implementation mechanism will result in less FPGA area but increased latency in the output. But in the MIMO-OFDM system, latency is of primary concern because the latency of the FFT/IFFT operation determines the OFDM symbol duration. Therefore, each individual stage is implemented separately in this work. Though it increases the chip area and power consumption, it produces the cycle count or latency that corresponds to the OFDM symbol duration of the IEEE 802.11a standard. The author of this work has written the controller for the data manipulation across stages. The Simulink® fixed point tool is used for the analysis of the blocks for fixed point operation. The Fixed Point Tools advisor^{TM} is helpful for performing this operation. A screen shot of this tool is shown in Figure 3.5. The HDL coder tool is then used for generating the HDL (Verilog/VHDL) code. The Modelsim® is then used to check the functionality and timing of the model. Once the timing and functional verification are done using Modelsim®, the HDL code is synthesized and implemented onto the target platform.

4.5.2 Results

A screen shot of the timing diagram is shown in Figure D.3. Figure D.3 validates the expected output. As determined from the Modelsim® output and by testing the actual FPGA circuit, the delay/latency experienced by the FFT module is around 188-192 clock cycles for a 64-point 16-bit input/output. The maximum rate at which the OFDM modulator/demodulator can run is determined from this clock cycle count. While the OFDM symbol duration is expected to be around 3.2 μs, the clock cycle required for the operation of the transmitter/receiver circuit is expected to be around 58-60 MHz. To demonstrate the portability of the models across different platforms, the FFT module results are shown from its implementation on the Pico computing FX60 board that has Virtex^{TM}-4 devices, **XCV4FX60**, with larger area and resource than the ML403 **XCV4FX12**. The logic resource consumed by the FFT module along with its other important timing constraints are shown in Table 4.5 and Table 4.6.

Table 4.5: Logic Resources Consumed by FFT in **XCV4FX60**

Logic utilization	Used	Available	Utilization
Number of Slices	2862	25280	1%
Number of 4-input LUTs	4266	50560	8%
Number of occupied slices	172	5472	3%
Number of FIFO/RAMs	4	232	37%
Number of DSP48's	80	128	62%

Table 4.6: Timing Summary for FFT in **XCV4FX60**

Minimum period	10.700ns
Maximum frequency	93.458 MHz
Maximum delay	6.976ns

The number of cycles taken by the FFT is key in the OFDM system design since it creates the OFDM symbol required for transmission. In this work, the delay of the FFT operation is 188-192 cycles for the 64-point operation. Thus, for creating an OFDM symbol of duration 3.2 μs, the clock rate of the system should be around 58.75 MHz to 60 MHz. Assuming the lower end of the operation, the OFDM circuit should run at a 60 MHz clock cycle.

4.6 Synchronization

In any digital communication system *synchronization* is a major task. Though most of the data in the system are digital in nature, the transmission over physical media are continuous in nature. Therefore, there is a conversion of the baseband digital signal to a higher-frequency carrier signal. This necessitates tuning the receiver to the same local oscillator frequency and phase to perform *coherent demodulation.* In most cases, the receiver is not synchronized with the transmitter and does not have accurate reference for the carrier signal and sampling clock. Any offset in the local oscillator frequency and sampling time will result in distortion, and in the case of OFDM, it shows up in the form of inter-carrier interference (ICI) and inter-symbol interference (ISI), resulting in the loss of orthogonality. Additionally, the unknown propagation delay between the transmitter and receiver results in *phase offset.* Doppler shift imposes further frequency shift on the received signal. As described before, OFDM that relies on the orthogonality of sub-carrier signals is more vulnerable to synchronization errors.

Generally, in a OFDM based receiver system, synchronization can be applied either in the time domain or frequency domain. The time or frequency synchronization techniques are chosen based on the transmission type chosen, latency, and system performance [13]. The IEEE 802.11a standard operates in burst transmission mode that makes the synchronization tasks more sensitive and difficult. Moreover, WLAN based on IEEE 802.11a assumes that the channel response does not change rapidly during the burst duration. Since the transmitted data packets over the burst are of short duration, in milliseconds, this assumption holds. Therefore the synchronization, either time or frequency, can be done only at the start of the preamble. Estimated parameters are not changed during the burst duration. Most of the synchronization tasks are done in the time domain because of the latency involved in converting to frequency domain due to FFT. The periodic repetitions in the preamble produce a good auto-correlation property that makes the time domain synchronization as the preferred choice [13].

Appendix A describes the preamble structure of IEEE 802.11a and the subsequent sections will describe the techniques employed in the synchronization.

4.6.1 Packet Detection

The receiver should detect the start of the OFDM symbol in burst mode transmission like IEEE 802.11a. As explained in the previous section, the periodic repetition of symbols at the start of the preamble is exploited to detect the starting of the burst data. This technique is also referred to as coarse symbol timing synchronization.

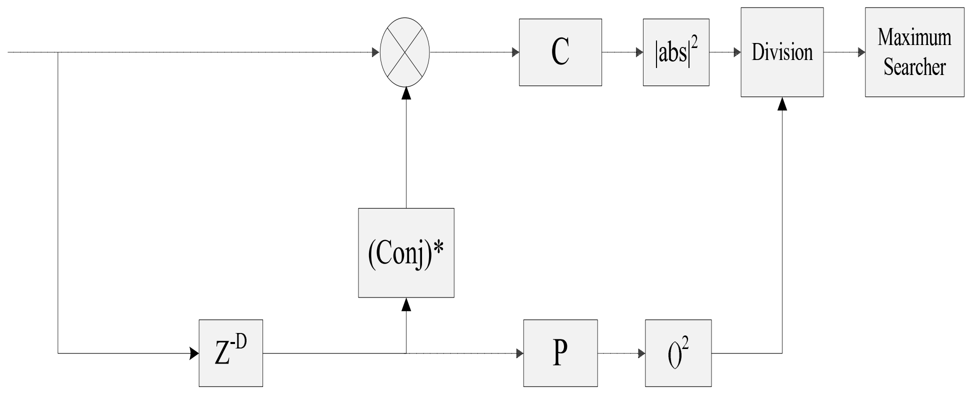
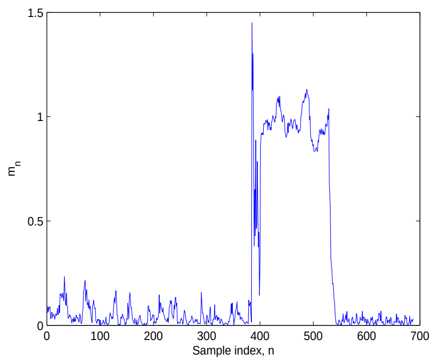
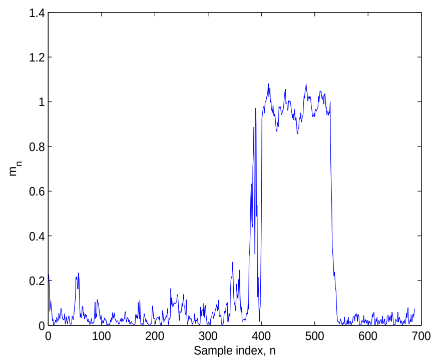


Figure 4.9: Packet Detection Block

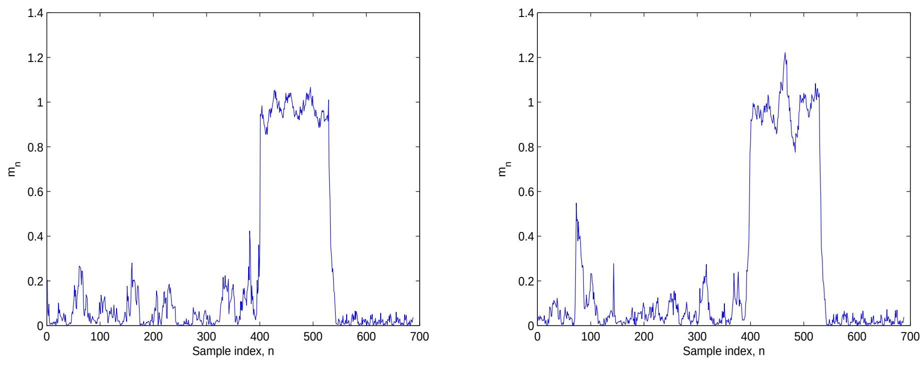


(a) with SNR= 1 dB



(b) with SNR = 2 dB

Figure 4.10: Packet Detection using the Short Preamble



(a) with SNR = 5 dB (b) with SNR = 10 dB

Figure 4.11: Packet Detection using the short Preamble

The *delay and correlate* algorithm [14] is used in this work to perform packet detection. It searches the start of the symbol by means of the *correlator*. The time index at which the packet starts is determined using the *maximum searcher* [15]. The delay and correlation algorithm can be viewed as a modified version of Schmidl's [16] algorithm. The delay and correlation block is represented in Figure 4.9. There are two sliding windows. One window cross-correlates the received signal and the delayed version of the received signal. The other window is used to calculate the received signal energy during the cross-correlation duration. The second sliding window makes the decision statistic independent on the absolute received power level. The decision statistic is computed as below [15]:

$$c_n = \sum_{k=0}^{L-1} r_{n+k} r^*_{n+k+D} \tag{4.6}$$

$$p_n = \sum_{k=0}^{L-1} r_{n+k+D} r^*_{n+k+D} = \sum_{k=0}^{L-1} |r_{n+k+D}|^2 \tag{4.7}$$

In the above equations, the value D represents the separation between two periodic symbol intervals and L represents the repetition period length. Then the decision metric, m_n, is calculated as follows [15],

$$m_n = \frac{|c_n|^2}{p_n^2} \tag{4.8}$$

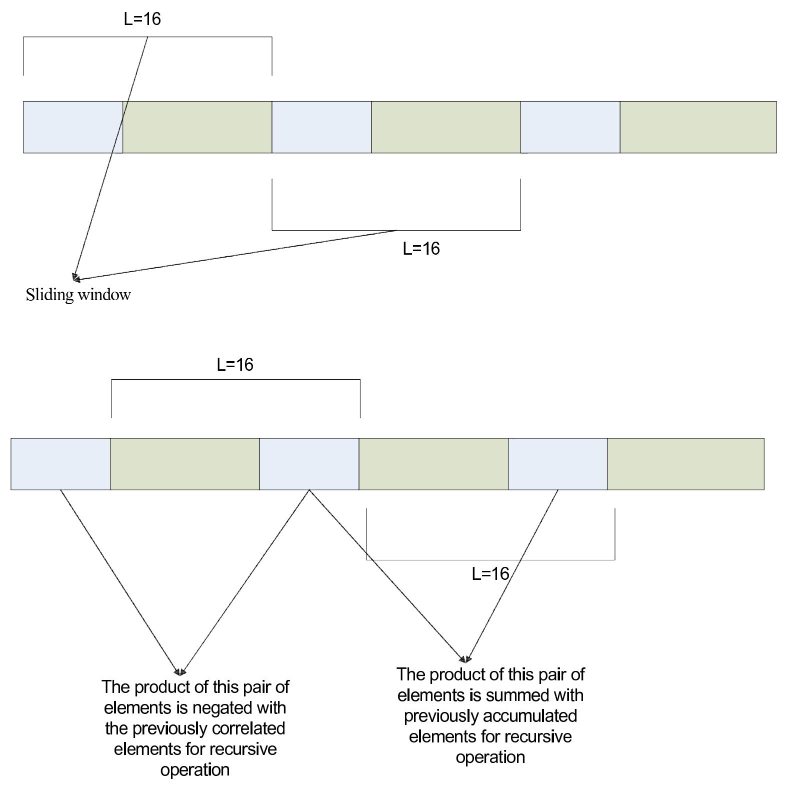

L=16
L=16
Sliding window
L=16
L=16
The product of this pair of elements is negated with the previously correlated elements for recursive operation
The product of this pair of elements is summed with previously accumulated elements for recursive operation

Figure 4.12: Recursive Procedure Represented in Block Diagram

Example decision statistics are shown in Figures 4.10(a), 4.10(b), 4.11(a) and 4.11(b). The step is the indication of the start of the packet. The response prior to the start of the packet consists of only noise resulting in almost zero correlation. Thus the decision statistic remains at a low level before the start of the packet. Because of the repetition of short preambles in IEEE 802.11a and IEEE 802.16e, there is a small plateau or width indicating the ISI free region. The presence of noise components causes distortion in the width of the plateau. Having a longer correlation length can help to reduce the distortion. Therefore, longer period length in the preamble and longer correlation length effectively increases the robustness of the decision metric in terms of SNR and timing detection.

4.6.1.1 Implementation

The arrangement of the preamble structure and the sliding window mechanism helps to reduce the number of arithmetic operations per correlation output. A recursive implementation is exploited for the synchronization algorithms. For example, consider the Equation 4.7. There are L complex multiplications and L+1 complex additions in each step. In order to reduce this computation, a recursive method is used as follows:

$$c_{n+1} = c_n + r_{n+L} \times r^*_{n+2\times L} - r_n \times r^*_{(n+L)} \tag{4.9}$$

Without the recursive operation the number of operations required for computing one output is L complex multiplications and L+1 complex additions. This cycle is repeated for the number of samples buffered. By doing the recursive operation, the correlation computation following the first output sample point requires 2 complex multiplications and 3 complex additions. Thus the author of this work found significant improvement in the latency of this block using the recursive operation.

4.6.1.2 Results

The logic resources consumed by the packet detection block are given in Table 4.7 and the timing summary in Table 4.8. The packet detection Simulink® block and output are shown in Figure B.4 and Figure C.4 respectively. The table values indicate the lower maximum operational clock frequency. The major consideration of this design block is the logic resource utilization rather than the latency. The operational frequency determines the samples that it can consume per cycle. In this case it is necessary to search for the 160 symbols of the short preambles.

Table 4.7: Logic Resource Utilization of Packet Detection Algorithm in **XCV4FX12**

Logic utilization	Used	Available	Utilization
Number of Slices	501	5472	9%
Number of slice Flip flops	525	10944	4%
Number of 4-input LUTs	805	10944	7%
Number of GCLKs	1	32	3%
Number of DSP48's	21	32	65%

Table 4.8: Timing Summary of the Packet Detection Algorithm in **XCV4FX12**

Minimum period	16.446ns
Maximum frequency	60.806 MHz
Maximum delay	6.987ns

4.6.2 Symbol Timing Estimation

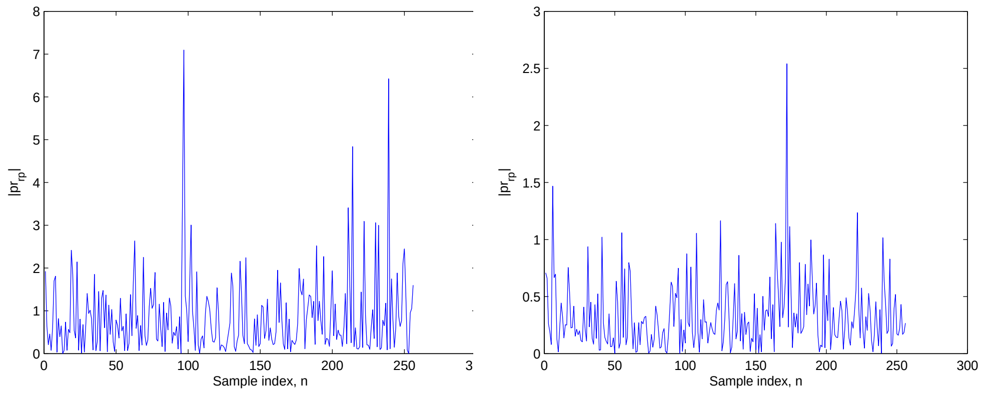
The coarse symbol timing estimation/packet detection is followed by the symbol timing estimation. In the OFDM based system, it becomes necessary to determine the sample point at which FFT can be effectively taken. Basically, the estimated point is the place at which the data part of the OFDM symbol starts. This can be viewed as fine symbol timing synchronization. In the case of packet based systems like IEEE 802.11a, the start of the exact symbol timing has to be determined as soon as possible in order to perform channel estimation and packet header determination. The packet header contains information like code rate, modulation scheme, and data rate.

The symbol timing estimation is determined by cross-correlating the received signal with the known *preamble*. Basically, this step helps in determining the channel impulse response. The starting of the exact symbol is important in demodulating the data on the sub-carriers. The cross-correlation can be computed by the following function,

$$\widehat{p_{rp}}(k) = \sum_{k=0}^{L-1} r_{n+k} p_k \tag{4.10}$$

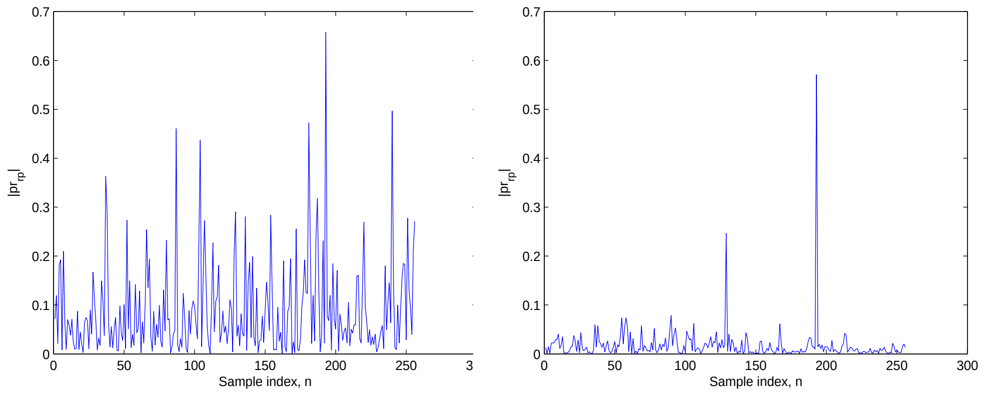
where p is the known preamble with L samples and r is the received signal. Then we determine the position at which maximum magnitude occurs:

$$\widehat{\phi_{max}} = \arg\max_{k} |\widehat{p_{rp}}(k)| \tag{4.11}$$



(a) with SNR= 1 dB (b) with SNR = 5 dB

Figure 4.13: Symbol Timing Detection using the Long Preamble



(a) with SNR = 10 dB (b) with SNR = 20 dB

Figure 4.14: Symbol Timing Detection using the Long Preamble

The ideal point at which the maximum value should occur is at the end of the cyclic prefix and the start of the data part of OFDM symbol. This is difficult to duplicate in practice.

4.6.2.1 Implementation

The implementation of symbol timing detection is also done using the recursive procedure as explained for packet detection. This reduces the latency for the estimation of the symbol timing. The Simulink® block for symbol timing estimation is shown in Figure B.5. An example output of the Simulink® block for symbol timing estimation is shown in Figure C.5. The recursive procedure reduces the latency and computational load for the symbol timing block. This is clearly shown in the following results section.

4.6.2.2 Results

The logic resources consumed by the symbol timing block is shown in Table 4.9 and the timing summary in Table 4.10. This block mostly works with the streaming symbols. Thus, the operational frequency of this circuit can be higher than its predecessor.

Table 4.9: Logic Resources Consumed by Symbol Timing Estimation Algorithm in **XCV4FX12**

Logic utilization	Used	Available	Utilization
Number of Slices	134	5472	2%
Number of slice Flip flops	137	10944	1%
Number of 4-input LUTs	241	10944	2%
Number of GCLKs	1	32	3%

Table 4.10: Timing Summary of the Symbol Timing Estimation Algorithm in **XCV4FX12**

Minimum period	6.845ns
Maximum frequency	146.090 MHz
Maximum delay	14.517ns

4.7 Frequency Offset Estimation

Frequency offset results from the mismatch between the oscillators of the transmitters and receivers, Doppler shifts, and phase noise in non-linear channels. This results in ICI and amplitude scaling due to the loss of orthogonality.

The estimation of frequency offset can operate in tandem with coarse symbol timing synchronization/packet detection. It also operates on the periodicity of the short training symbols present in the preamble of both the IEEE 802.11a and IEEE 802.16e standards. The *Maximum Likelihood Estimation* for frequency offset is given by [14]:

$$\widehat{\theta} = \sum_{k=0}^{L-1} r_n r_{n+D}^{*} \tag{4.12}$$

Then the expression for frequency offset is given by,

$$\widehat{f_{\Delta}} = -\frac{1}{2\pi D\, T_s} \angle \widehat{\theta} \tag{4.13}$$

where T_s is the sampling period. In general, the frequency offset is normalized with respect to sub-carrier spacing $f_s = \frac{1}{NT_s}$. The normalized estimate consists of two parts. One is the integer offset and other is the fractional offset. The initial coarse estimation from the above equations will give only the fractional offset, i.e. within the plus half or minus half of the sub-carrier spacing. This is because the angle that can be resolved is $[-\pi, \pi]$. The range of frequency offset that can be resolved is directly related to the length, L, of repeated symbols. The delay (D) and length of the symbol (L) together determine the range of frequency offset correction. Basically, the smaller the length of the repeated symbols, the larger the range of frequency that can be corrected.

For example, consider the short preamble of IEEE 802.11a with the sample time 0.05μs and delay 16 samples. The maximum delay that can be estimated is:

$$\begin{aligned} \widehat{f_{\Delta max}} &= \frac{1}{2DT_s} \\ &= \frac{1}{(2 \times 0.05 \times 10^{-6} \times 16)} \\ &= 625KHz \end{aligned} \tag{4.14}$$

This maximum error meets the requirement of the IEEE 802.11a standard specification. IEEE 802.11a operates at the carrier frequency of around 5.2-5.3GHz with maximum oscillator error specified as ±20ppm. Thus, with maximum oscillator error at both the transmitter and receiver, the maximum error amounts to 40 ppm. The frequency error resulting from this deviation is equivalent to 212 KHz. Thus, maximum frequency error is within the range of this algorithm. As the repetition length of the training symbol increases, there is a decrease in the range of the frequency error that can be corrected.

The above step results only in coarse frequency correction by making use of the short training symbols. In order to deal with integer frequency offsets, the fractional frequency offset compensated signal is then cross-correlated with long training symbols following the short training symbols. This can work in tandem with fine symbol timing synchronization. The output of the correlator will be maximum at the point at which the long preamble is modulated by the signal with correct integer frequency offset. The number of correlators required is limited by the number of integer offsets that can be corrected. Otherwise, the same correlator can be matched to a different integer frequency offset.

There is also phase error in the OFDM symbol due to phase noise and phase offset. Therefore the carrier phase has to be tracked at every OFDM symbol. This results in the rotation of the constellation resulting in the crossing of decision boundaries. This is corrected in the frequency domain using the *pilot sub-carriers* transmitted in each OFDM symbol. The pilot sub-carriers in the IEEE 802.11a and IEEE 802.16e standards are used for channel estimation and phase tracking. In both of these standards, the pilots are transmitted with higher power. This makes the identification of the pilot sub-carriers easier at the receiver. Because of the FFT operation at the receiver, the resulting OFDM symbol at the receiver will be the multiplication of channel frequency response and the transmitted pilot and data sub-carriers. In the case of residual frequency error, the pilot carrier would have undergone rotation equivalent to this frequency error as given by the following equation.

$$R_{n,k} = H_{n,k} P_{n,k} e^{j2\pi f_{\Delta}} \tag{4.15}$$

where H_k is the channel frequency response. If the channel estimate $\widehat{H_k}$ is available, then the phase estimate is given by [5]:

$$\widehat{\phi_n} = \angle \sum_{k=1}^{N_p} R_{n,k} (\widehat{H_{n,k}} P_{n,k})^* \tag{4.16}$$

where N_p is the number of sub-carriers. If the channel estimate is perfect,$H_k = \widehat{H_{n,k}}$, then the equation applies:

$$\widehat{\phi_n} = \angle \sum_{k=1}^{N_p} |H_{n,k}|^2 |P_{n,k}|^2 e^{j2\pi f_{\Delta}} \tag{4.17}$$

With pilots amplitude equal to unity, the phase estimate is given by [5]:

$$\widehat{\phi_n} = \angle \sum_{k=1}^{N_p} |H_{n,k}|^2 e^{j2\pi f_{\Delta}} \tag{4.18}$$

There will be distortion in the phase estimate if the channel estimate is not known.

Frequency offset correction is the next major task. As with estimation, this can be implemented in the time domain or frequency domain. In most of the current digital baseband processing techniques, changing the local oscillator (LO) frequency based on the frequency offset estimation is a tedious process. On the other hand, compensating for phase rotation in the frequency domain can be easily implemented in real time baseband processing systems.

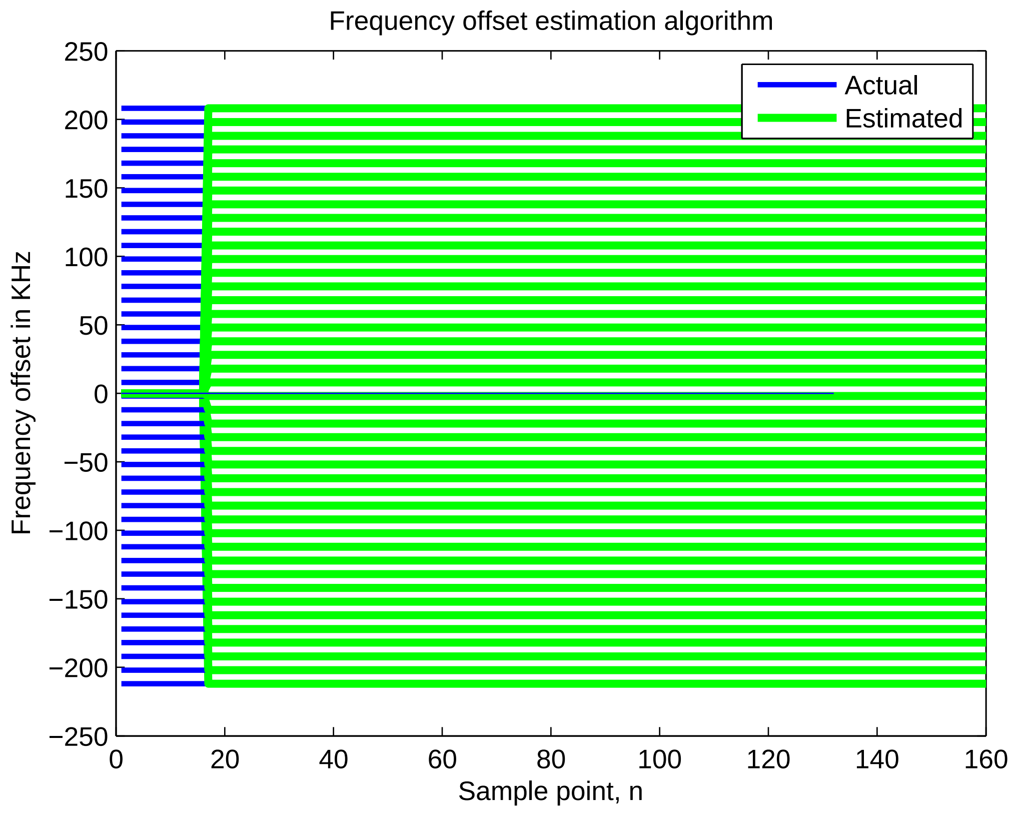


Figure 4.15: Performance of Frequency Offset Estimation Algorithm

The phase derotator is required for the correction. It can be implemented using a frequency domain interpolator. The interpolation will produce a signal at different frequencies. The interpolated output is then multiplied with the FFT output signal to reduce the ICI [17].

4.7.1 Implementation

The performance of the frequency offset estimation algorithm is shown in Figure 4.15. It indicates that after the first 16 samples, the estimated value for the frequency offset converges to the actual offset value. Basically, it takes 16 samples, equal to the period of short training sequence, to converge. The blocks necessary for the implementation of frequency offset estimation are derived from the Xilinx System generator for DSP^{TM}. Though the Simulink® HDL coder^{TM} has a block for computing trigonometric functions, it is not currently working/supported by The Mathworks, Inc. So the CORDIC IP core from Xilinx, Inc. is used in this work. The basis of the frequency offset correction is the numerically controlled oscillator (NCO) working in the time domain. The NCO operating in the time domain acts as a time-domain de-rotator. It is basically a lookup table with entries from the sinusoidal wave over the quarter of the period. The outputs from the NCO are complex multiplied with the incoming received signal. This work attempts to correct only fractional frequency offset.

4.7.2 Results

The logic resources consumed by the CORDIC IP core provided by Xilinx, Inc. are shown in Table 4.11. The timing summary is tabulated in the Table 4.12. From the experience of building models from Simulink® HDL coder^{TM}, it is clear that the resources consumed by this particular core are high.

Table 4.11: Logic Resources Consumed by CORDIC IP in **XCV4FX12**

Logic utilization	Used	Available	Utilization
Number of slices	608	5472	11%
Number of slice flip flops	1018	10944	9%
Number of 4-input LUTs	1042	10944	9%
Number of GCLKs	1	322	3%

Table 4.12: Timing Summary for CORDIC IP in **XCV4FX12**

Minimum period	4.554 ns
Maximum frequency	219.569 MHz
Maximum delay	Not applicable

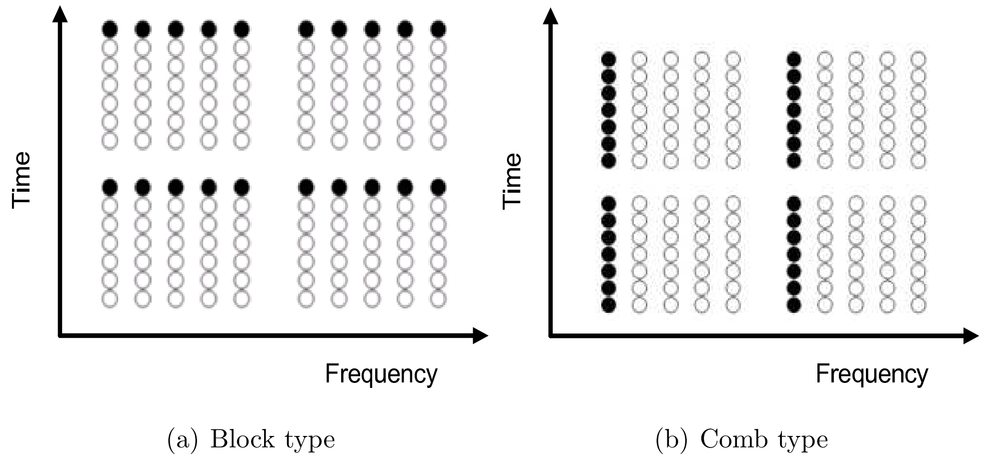
4.8 Channel Estimation

Channel estimation is necessary for the decoding of Alamouti encoded symbols. Inaccurate estimation of the channel will result in considerable degradation of the performance of the OFDM based wireless system. In particular, Alamouti encoded symbols rely on accurate channel estimation for decoding at the receiver.

There is an apparent problem in the implementation of channel estimation techniques employed in the generic OFDM scheme for the MIMO-OFDM case. In MIMO systems, it is clearly known that the received signal is the combination of signals transmitted from multiple antennas at the receiver. In order to separate the pilot symbols at the receiver a *pseudo random generator* is specified in the IEEE 802.16e standard. This makes the pilot symbols transmitted from multiple antennas orthogonal to each other. This makes the symbols placed in a two dimensional time frequency grid orthogonal to each other in time.

Once the pilots position has been defined, it can be taken advantage of by using well-known channel estimation algorithms. This is referred to as pilot-aided channel estimation. However this kind of estimation cannot be applied directly in this work because of the diversity technique employed at the transmitter. Therefore, several channel estimation techniques by Li et al [18],[19] dedicated for MIMO-OFDM systems were considered. The techniques employed in the literature require intense computational complexity. So, we decide to employ a modified estimation mechanism for the MIMO-OFDM case using the techniques mentioned by Coleri et al [20].

In this case, the pilot symbols are arranged in the OFDM symbol based on the IEEE 802.16e specification [21]. The polarity of the pilot symbol transmitted from each antenna is determined by the linear feedback shift register that makes the pilot symbols transmitted in each OFDM symbol orthogonal to each other.



(a) Block type (b) Comb type

Figure 4.16: Pilot Arrangement for Channel Estimation

Generically, the pilot symbols can be arranged in *block-type* or *comb-type*. Examples of this arrangement are shown in Figure 4.16. In the block-type arrangement, all the sub-carrier frequencies in the ODFM symbol are used as pilots, whereas in the comb-type arrangement only certain frequencies are used as pilot frequencies. In both the block-type and comb-type arrangement the estimation at pilot frequencies can be done either by the least squares (LS) estimation technique or minimum mean square error (MMSE) estimation technique. In this work, only the LS estimation technique is employed because of its reduced complexity. In the case of the comb-type arrangement the pilot frequencies are repeated periodically in each OFDM symbol and the estimates obtained at pilot frequencies are used to compute the estimates at non-pilot frequencies. Some of the interpolation techniques used are linear, low-pass, and spline cubic. The mathematics behind each of the techniques is explained in Appendix E.

The performance of comb-type estimation under various channel conditions is shown in Figure 4.17, Figure 4.18, and Figure 4.19.

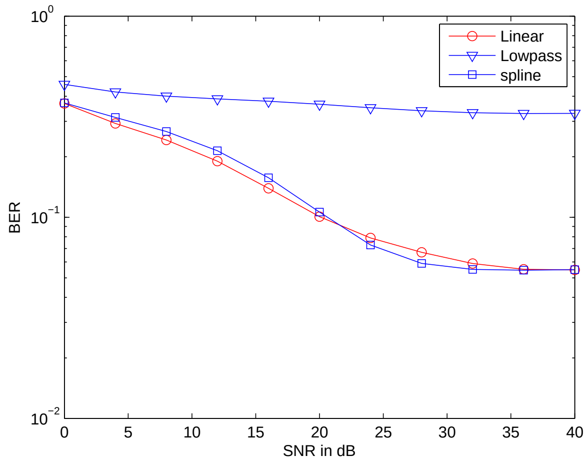


Figure 4.17: Performance of Channel Estimation Algorithm in Slow Fading Channel

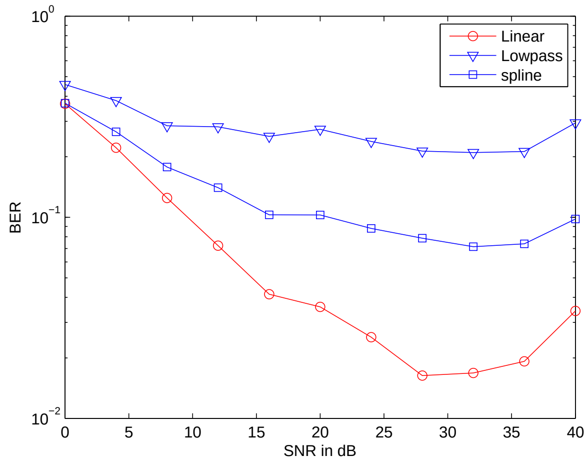


Figure 4.18: Performance of Channel Estimation Algorithm in Fast Fading Channel

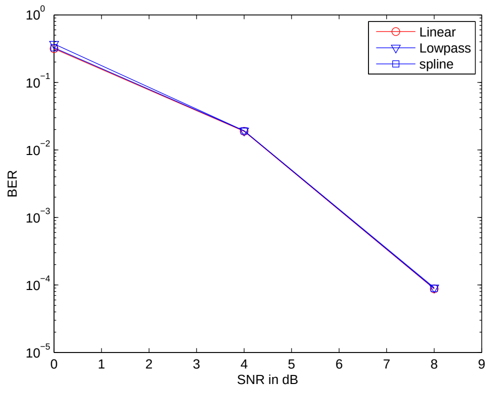


Figure 4.19: Performance of Channel Estimation Algorithm in AWGN channel

4.8.1 Implementation

Based on the performance analysis of various channel estimation techniques it is clear that the *linear interpolation* technique is an optimal channel estimation mechanism for the Alamouti encoded OFDM system. While working in this module, the author realized that it takes less time to directly code in HDL for certain simple modules like these. The primary reason being that the MBD tools used in this work do not have the necessary blocks to implement the channel estimation mechanism. The linear interpolation technique is implemented by behavioral modeling or direct HDL coding.

4.9 Convolutional Encoder

Both the IEEE 802.11a and IEEE 802.16e standard use a *convolutional encoder* as a channel coding technique. Its implementation is very straight forward. In general, a *convolutional encoder* generates its coded information by convolving the input message sequence with a set of coefficients. This set of coefficients depends on the *code rate* employed. For the IEEE 802.11a standard, a generic rate $\frac{1}{2}$ convolutional encoder has been defined. The other code rates like $\frac{2}{3}$ and $\frac{3}{4}$ are realized using a separate *puncturing block*. This puncturing block also performs an *interleaving* operation.

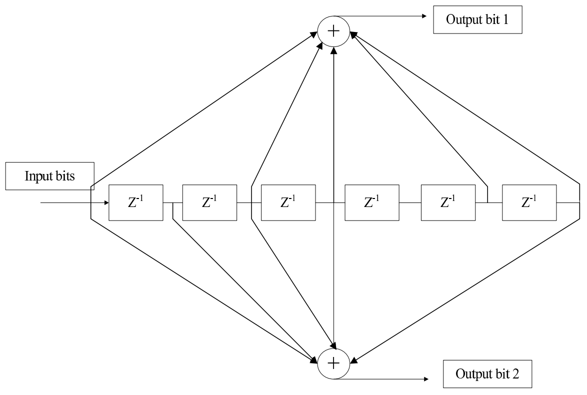


Figure 4.20: Block Diagram of Convolutional Encoder used in the IEEE 802.11a Standard

A generic (n, k) convolutional encoder takes k input message bits at each clock cycle and produces n coded bits as output. One can view each coded bit as the sum of k convolution outputs. The convolutional encoder is defined by the parameter k called the *constraint length* of the code. It defines the number of coded bits that depends on the current input bits. A $\frac{1}{2}$ convolutional encoder is shown in Figure 4.20. The constraint length determines the memory depth of the encoder. This has direct effect on the complexity of the *Viterbi Decoding* mechanism. The shift registers shown in Figure 4.20 are initialized to zero before the start of the message sequence and it has to be returned to an all-zero state after the encoding of all message bits. The output sequence is obtained by convolving the input bit sequence with the impulse response of the convolutional encoder. In this case, the convolutional sum is operated through the *Galois field* operation. It reduces to a bit XOR operation in this case. In the case of IEEE 802.11a, the rate of the code is $\frac{1}{2}$ with the constraint length $k = 7$ and the generator polynomials/impulse response specified as $\mathbf{g_0} = 133_8$ and $\mathbf{g_1} = 171_8$. These generator polynomial sequences specify the tapping points in the encoder. Implementation is detailed in the subsequent section.

4.9.1 Implementation

Implementation of the Convolutional encoder is very straight forward. Generally, a convolutional encoder can be implemented by either the *lookup table* method or the encoder is represented using *shift registers.* The shift register method is employed in the work because of the circuit's ability to operate at a faster clock rate. The logic resource occupancy of the lookup table resulted in resource crunch during the mapping process of the circuit implementation. In most of the cases here, the focus is on using continuous streaming operations rather than using a complex sequential logic to implement the circuit. Implementation involves the use of *delay elements* as the elements of the shift register and the Galois field addition operation that reduces to XOR. A snapshot of the model created using Simulink^{TM} is shown in Figure B.3. The corresponding output is shown in Figure C.1. The model is then converted to HDL code using the automatic code generation tool. The options going into this code generation play a key role in creating a synthesizable HDL code. The generated HDL code is simulated using ModelSim®. A screen snapshot is shown in Figure D.1. The logic resources consumed, and other details that affect the performance of this circuit in the overall system are shown in the results section.

4.9.2 Results

As shown in Figure 4.20, the convolutional encoder can be viewed as a shift register with outputs of registers at various locations being tapped to produce the output. Since the circuit is mostly sequential, comprised only of delay elements, the circuit is expected to be able to operate at a high frequency.

Table 4.13: Logic Resources Consumed by Convolutional Encoder in **XCV4FX12**

Logic utilization	Used	Available	Utilization
Number of slices	27	5472	0%
Number of slice flip flops	48	10944	0%
Number of 4-input LUTs	24	10944	0%
Number of occupied slices	172	5472	3%

Table 4.14: Timing Summary for Convolutional Encoder in **XCV4FX12**

Minimum period	0.759ns
Maximum frequency	1317.176 MHz
Maxim delay	6.814ns

4.10 Scrambler/De-scrambler

A *scrambler* is used in the IEEE 802.11a standard for the DATA, SERVICE, pad, and tail parts using a length-127 frame-synchronous scrambler. The purpose of the scrambler in most cases is to provide an extra level of encryption. But in this case, it is used to scramble the data so that the probability of transmission of data bit sequence that resembles the short

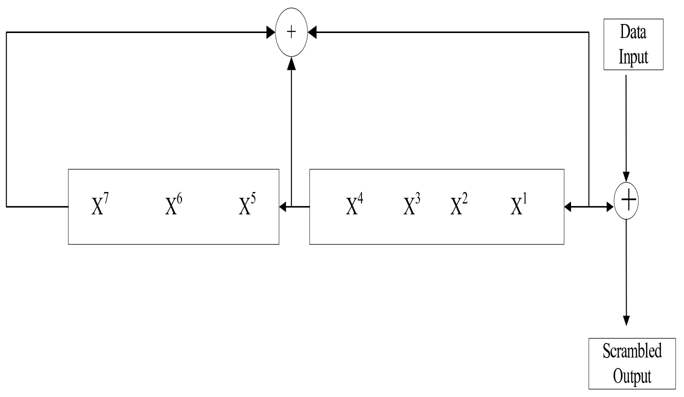


Figure 4.21: Block Diagram of Scrambler/De-scrambler used in the IEEE 802.11a Standard

training and long training sequences is very small. This helps to avoid the false alarm of packet detection. The IEEE 802.11a standard specifies the scrambler to be initialized with any non-zero pseudo-random sequence. The same scrambler structure, as shown in Figure 4.21, is used for *descrambling* at the receiver. The generator polynomial for this scrambler is given as,

$$S(x) = x^7 + x^4 + 1 \tag{4.19}$$

The initial state of the scrambler gets added to the preamble that helps the descrambler in estimating its initial state. As with the convolutional encoder, the mathematical operation behind this scrambler is based on Galois field arithmetic, which essentially reduces the operations to shifting and XORing. The type of scrambler used here can be referred to as *additive scrambler*. The two things that define the scrambler are the *generator polynomial* and *initial state*. IEEE 802.11a specifies the initial state to be either all-ones or 1011101. This initial state determines the 127-bit sequence generated by the scrambler. As mentioned before, the initial state has to be known at the receiver to correctly de-scramble the data.

4.10.1 Implementation

The mathematics behind the scrambler/descrambler is similar to the convolutional encoder. The snapshot of the Simulink® is shown in Figure B.2. The corresponding Simulink® and

Modelsim® is shown in Figure C.3 and Figure D.4 respectively. The results sections will show the logic resources consumed in the target FPGA platform. The numbers present in the table are of significance while doing overall system integration.

4.10.2 Results

The logic resources consumed by the scrambler module along with its timing specification are shown in the following tables.

Table 4.15: Logic Resources Consumed by Scrambler/De-scrambler in **XCV4FX12**

Logic utilization	Used	Available	Utilization
Number of slices	4	5472	0%
Number of slice Flip flops	7	10944	0%
Number of 4-input LUTs	2	10944	2%

Table 4.16: Timing Summary for Scrambler/De-scrambler in **XCV4FX12**

Minimum period	1.179ns
Maximum frequency	848.033 MHz
Maximum delay	6.165ns

4.11 Other Modules

In the course of making an OFDM based radio, there were a few other modules that were created in addition to the modules mentioned above. Some of the blocks include BPSK, QPSK, and Viterbi decoding. Among these, the implementation of Viterbi decoding is the most complex. Its logic resource occupancy and timing summary are given in Tables 4.17 and 4.18. Its significance is manifested from its resource occupancy in the FPGA chip. Moreover, its maximum operational frequency is the lowest.

Table 4.17: Logic Resources Consumed by Viterbi Decoder in **XCV4FX12**

Logic utilization	Used	Available	Utilization
Number of slices	3469	5472	64%
Number of slice Flip flops	3411	10944	30%
Number of 4-input LUTs	4374	10944	41%
Number of GCLKs	1	32	3%

Table 4.18: Timing Summary for Viterbi Decoder in **XCV4FX12**

Minimum period	14.943 ns
Maximum frequency	60.656 MHz
Maximum delay	6.156 ns

For implementing the system in real-time, there are other modules that are required. This includes CORDIC and FIFO buffers. Both these modules were implemented using Xilinx IP cores. The author attempted to implement the CORDIC algorithm using Simulink® blocks but the results were not satisfactory. The CORDIC module played a key role in the demodulation block. The FIFO buffers along with the DCM clock manager are very specific to the Xilinx FPGA implementation. These are necessary for operating the blocks in multiple clock domains.

4.12 Implementation Issues

4.12.1 Validation and Testing

The author tested the individual modules using the stimuli generated from the Simulink® tool and MATLAB^{TM} simulations. Initially the model is validated for output using the Simulink® tool. Then the module is tested using the pre-synthesis Modelsim^{TM} tool. The output obtained from the simulation tool is used for the verification of the circuit. In order to verify the testing, the HDL code generated by the Simulink® HDL coder ^{TM} is synthesized, mapped, placed, and routed using Xilinx ISE^{TM} tools. There are certain issues associated with this kind of testing. There are cases where the circuit synthesized using the Xilinx ISE^{TM} tool will not work on the target FPGA platform. The bitstream generated from the Xilinx ISE^{TM} tool is then used with the EDK environment of the ML403 development board for testing. The circuit generated by the author was added as a peripheral using the Peripheral Local Bus (PLB) of the PowerPC^{TM} 405. While creating this bus there are a number of options available to configure the peripheral. One such option is to configure the input and output FIFO's required for sending and receiving the data from the peripheral. The author of this work created the file with input stimuli from MATLAB^{TM} and then send it to the input FIFO buffer of the peripheral and read the output created by the peripheral from the output FIFO. This is one of the basic ways to test the circuit. In order to test the multiple sub-parts of the system, the results created by each individual module are used as the input to the next subsequent module of the system. A typical block diagram for performing this operation is shown in Figure 4.22.

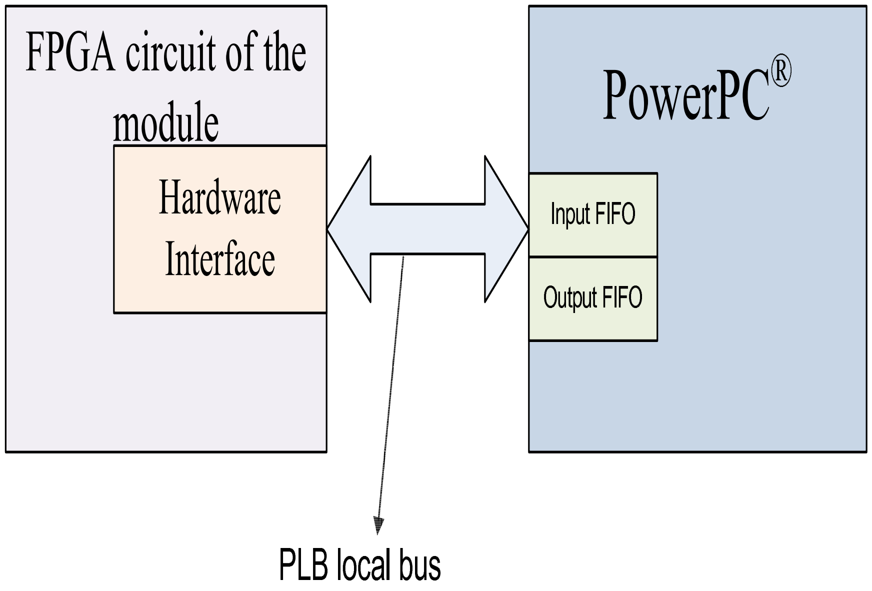


Figure 4.22: Interfacing Between PowerPC® and FPGA Circuit

4.12.2 Timing Constraints

The major part of the FPGA system design process is the verification of the *timing constraints*. This is very important from the system perspective. There are two important times that determines the validity of the signal in the sequential circuit. They are *setup time* and *hold time*. The setup time determine the amount of time the input signal should arrive before the arrival of the clock. Similarly, the hold time determines the amount of time the input should be held high in order for it to be realized as valid input and propagated to the next flip-flop or memory. Any jitters in these times or clock will cause the flip flops to go in a metastable state, resulting in erroneous output propagating to the entire circuit. Most of these verification and validation mechanisms cannot be automated. The user who is designing the system should work in conjunction with the person who is doing verification to make sure that the timing requirements required for his/her system are met. Additionally, there is a *propagation delay* associated with the combinational logic between the flip-flop elements. This determines the amount of delay that is going to be experienced by the user defined circuit while propagating the signal from the input to the output. This determines the minimum delay and the maximum frequency at which this circuit can operate.

4.12.3 Training Sequences

The author of this work realized that designing a complex system like a MIMO-OFDM based radio requires lot of understanding of the real time scenarios that will affect the performance. For example, transmitting a training sequence simultaneously over multiple antennas will make the synchronization process very difficult at the receive antenna. The basic assumption made in this design is that the training sequences are transmitted orthogonally in time. The design of this sequence is shown in Figure 4.23. This is important for estimating the channel across multiple transmit antennas and receive antennas. This does not avoid the problem of overlapping pilot symbols transmitted from multiple antennas at the same sub-carrier location. For this case, the pilot symbols have to be orthogonal to each other in the

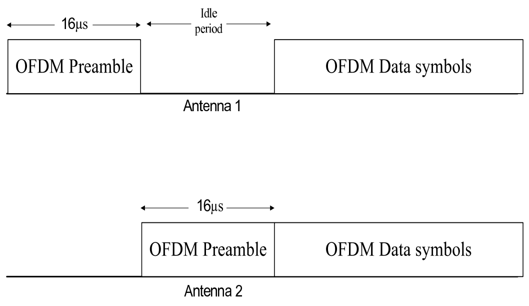


Figure 4.23: Arrangement of Training Sequences in the Preamble

frequency domain. The pilot arrangement to deal with this situation was explained in the channel estimation section.

Chapter 5

System Configuration for FPGA Board with USRP2 as RF Front End

As mentioned in Chapter 1, the intention of this work is to perform baseband processing for a MIMO system in the USRP2 SDR platform. Because of implementation constraints, the goal of this work was modified towards USRP2 acting as an RF front end with an auxiliary FPGA board performing the baseband processing.

The overall system consists of a USRP2 and Xilinx ML403 development board with Virtex^{TM}-4 on it. The PowerPC®-405 core present in the FPGA of the development board is helpful for controlling the configuration of the USRP2. Thus the overall setup is shown in the following Figure 5.1.

5.1 ML403 Development Board and Its Interfaces

The ML403 development board is powered by a Virtex®-4FX12 FPGA and is used for embedded system development. Its base clock runs at the speed of 100MHz. It has a PowerPC®-405 core on it that can run at the speed of 300MHz. It is a simple embedded platform for testing and prototyping. It does not have an analog-to-digital(ADC) converter or a digital-to-analog (DAC) converter. It has peripherals like Ethernet, USB, UART, etc. The major emphasis in this work is to interface the PowerPC®-405 core with the Ethernet core present in the ML403 system and to make this interface take critical control of the Ethernet core present in the USRP2. Its major connection bus includes CoreConnect^{TM} architecture that is helpful in making different types of bus connections between the PowerPC® and peripherals. For example, the buses include a processor local bus (PLB), on-chip peripheral bus (OPB), and device control register (DCR) bus. The PLB bus has an intellectual property interface (IPIF) that provides a connection to user-created peripherals. This interface can be viewed as a memory-mapped I/O that aids in accessing the peripherals, like addressing the register for reading and writing.

The peripherals can interface with a PowerPC® through an intellectual property interface (IPIF). The user can also write his/her own driver for the interface. The circuit performing baseband processing is interfaced with the PowerPC® using the IPIF. The IPIF is responsible for simplifying the CoreConnect^{TM} bus system by representing it as simple control registers. It helps in maintaining the modularity of the custom IP block created by making its IPIF interface independent of the bus. All the features, like interrupts, DMA, and status registers, necessary for the operation of the custom IP or existing peripheral in the ML403 board are provided by IPIF. The IPIF drives the IP block with the same clock that its bus is connected to.

Then comes the protocol required for talking to the UDP packets sent externally from the USRP2. The advantage of using the ML403 board is the ready availability of a Linux operating system to be run on the PowerPC® and drivers required for interfacing the peripherals. The procedure for installing Linux on the PowerPC® is based on references [22] and [23].

The clocks required for the operation of both the PowerPC® processor and bus are driven by the single system clock source running at 100Mhz. The Virtex^{TM}-4 uses digital clock manager (DCM) to generate multiple frequency clocks.

5.1.1 Device Interfacing

Once the Linux operating system is installed on the PowerPC®, it is ready to access the peripherals present in the board. The devices present in the board are accessed as the memory-mapped I/O. On top of it, the Linux kernel has pre-built drivers for accessing Ethernet MAC and various other devices. The lightweight Internet protocol (LwIP) is required for accessing the UDP packets. LwIP requires Xilkernel to manage the threads, semaphores, etc. The Linux operating system controls the memory management unit (MMU) that takes the address of the register representing the device and converts it into a physical address for accessing the device. The significance of this approach is that the user can provide the user level code to access these devices rather than writing drivers at the kernel level.

The addressing mechanism for accessing the devices is very straight forward. It takes the base address of the peripheral or device and uses the offset to read and write from it. For a developer, this is just like referencing and de-referencing the pointer, as used in C programming language.

5.1.2 Lightweight Internet Protocol

LwIP is a networking stack developed for low memory footprint embedded applications. The Xilinx EDK environment has customized this protocol to run on the ML40x development boards using a PowerPC® processor. The EDK user manual from Xilinx, Inc. helps us understand this integration. It uses Ethernet MAC to send and receive packets. EDK uses two types of Ethernet IP's: *xps_ethernetlite* and *xps_ll_temac*. The former is used for low bandwidth applications while the later is used for high bandwidth applications. It requires a periodic interrupt controller to look for the packets in the buffer. These interrupts inform the software for packet reception and transmission.

The Linux present on the PowerPC® acts as a controlling point. It reads at the speed of 300MHz, and the system has an Ethernet core interfaced with a FIFO buffer. The block diagram is shown in Figure 5.1. Thus the PowerPC® is responsible for all the data transfers happening between the baseband processing core and the Ethernet MAC core. This implies that data transfer is very slow with around 50 cycles required for accessing the memory mapped I/O peripherals. This forced the processor to be clocked at higher speed of 300 MHz for working with other peripherals running at 100 MHz or lesser speed.

5.2 USRP2 and Its Drivers

The USRP2 is an SDR platform useful for building radios with a wide range of applications. There are different kinds of daughter boards available to target different kinds of applications. The USRP2 is a successor to the USRP, with added capabilities in terms of FPGA and Gigabit Ethernet connection. It uses a Xilinx Spartan-3 FPGA with its ADC producing 14-bit samples at the rate of 100 MS/s and its DAC capable of producing 16-bit 400 MS/s. The interpolation and decimation factors determine the final data rate delivered to the host machines. There is a separate channel that operates between the USRP2 and hosts that control the various configuration parameters of the FPGA present in the USRP2.

5.2.1 UHD Driver

The interfacing of the USRP2 with the host happens through two kinds of drivers. They are raw Ethernet sockets and universal data protocol (UDP) packets. In this work, interfacing with the FPGA is attempted through UDP packets. The primary reason being that the "Universal Hardware Driver" released recently by the GNURadio community can be used to interface USRP2 with all types of operating systems. It uses the UDP packet protocol. It also makes deterministic handling of data packets at the FPGA board possible. It operates independent of the GNURadio framework and only requires application interfaces (API) to interact with the core of the driver.

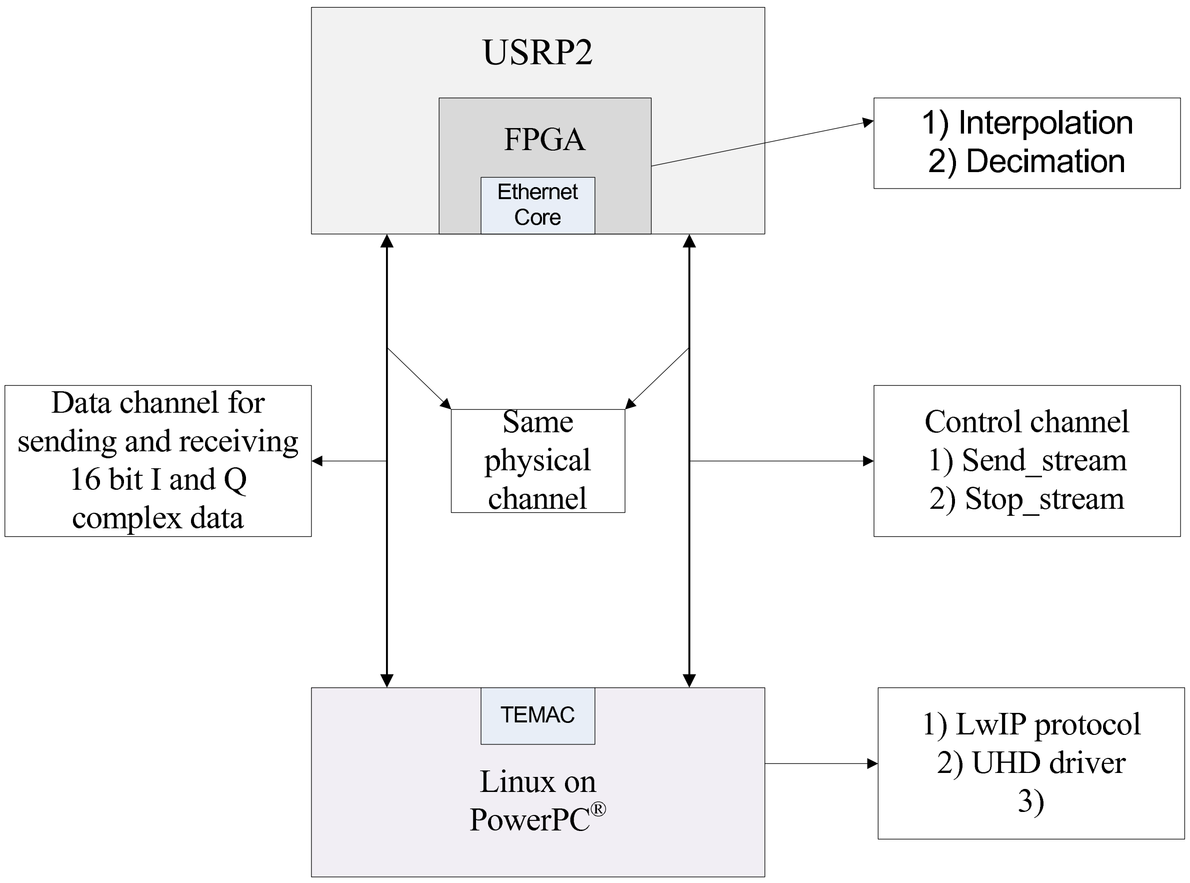


Figure 5.1: Interfacing Between USRP2 and PowerPC®

The testing of the peripherals and bus signals during interfacing is done through the Chip-Scope Pro^{TM} tool of Xilinx, Inc. An interesting observation was that the FPGA present on the USRP2 is not configured to send the data packets without the configuration of its FPGA using the control channel. This control channel is responsible for indicating to the USRP2 for starting/stopping the transmission/reception.

5.2.2 Channels

The design of the USRP2 indicates that the data to be received from the USRP2 can be in the form of a complex short or int. The chosen data type and decimation factor are indicated to the USRP2 by using the control channel. Once the control signal is received by the USRP2, the data are formatted to send to the host. The size of the receiving buffer dictates the number of samples that can be processed by the receiver or transmitter. There is a challenge in making the USRP2 transmit samples to reach the host at a data rate that can be handled by the FPGA of the developmental board. The amount of buffer given by the PowerPC® for each of the peripherals in the board varies from 2k to 32K bytes of memory. This limited memory acts as FIFO that can be configured for transmission and reception of samples on the developmental board. Each time the PowerPC® accesses the peripheral there is a 50 cycles cost for it. Though the processor is running at a speed of 300 MHz, it is not sufficient to achieve the processing speed required for high end wireless applications. However the work presented here acts as a prototype for offloading the signal processing applications onto the FPGA. There is a high speed connectivity option available to connect the USRP2 with the FPGA board. It is explained in the conclusion section. The USRP2 with its own firmware is capable of achieving the MIMO system configuration. But the GNURadio community does not yet support a MIMO configuration of the USRP2 using the current drivers.

While transferring data using the LwIP protocol, the RAW API mode is used. Instead if the socket mode is used with the LwIP protocol, then system cannot operate at 1000 Mbps. The present USRP2 configuration supports only 1000 Mbps operation. The RAW API model requires the OS to take care of the processing of the headers. This gives the flexibility for handling the header format required for transmitting and receiving data specific to the transport protocol of the USRP2. The channels shown in Figure 5.1 operate over the same physical channel i.e. Ethernet cable.

Chapter 6

Conclusions and Future Work

This chapter presents the concluding remarks and directions for future work that can be based on this work.

6.1 Concluding Remarks

The work presented here acts as a proof-of-concept for the model based design and the deployment of signal processing algorithms required for SDR on an FPGA. Over the years, the development of SDR has required rapid re-configurability and wide bandwidth processing capability. The majority of the current wireless standards like WLAN, WiMAX, digital video broadcast for terrestrial (DVB-T), long term evolution (LTE), etc. offer a data rate in the range of 10 to 100 Mb/s and require a higher processing capability than that of GPPs or DSPs. At the same time, the level of re-configurability has to excess that of applications written on ASICs. Incorporating these diverse standards on the FPGA of SDR platform satisfies the need for re-configurability and latency sensitive processing requirements.

With establishment of the interfacing, as described in the Chapter 5, this work promotes the idea of design and development of SDR on the lines of GNURadio with an FPGA acting as an underlying processing platform. There is a large number of open source signal processing blocks being developed for the GNURadio framework. Converting these signal processing blocks into a FPGA specific code is going to take tremendous effort and time. In order to reduce the development time for the applications targeting SDR applications, a MBD based design flow is explained in Chapter 3. The work presented in Chapter 4 shows the effectiveness in the application of MBD design flow for developing wireless baseband algorithms on the FPGA. From Chapter 4, it is clear that the user with his/her capability can create an IP core rather than relying on the third party IP core for development. These tools helps in reducing the development time and cost. The sections presented in the Chapter 4 prove that the individual signal processing block can be developed on the lines of the GNURadio framework using MBD based design flow. Thus the design flow presented in Chapter 3 is an efficient and robust means of developing a FPGA based SDR system.

With the rapid advancement of communication technologies, there are continuous changes in the underlying hardware platform, characteristics of the hardware, and the operating conditions of the system. The use of tools and design flow as mentioned in Chapter 3, avoids the need for a complete re-design of the system. The use of these tools promotes the idea of adaptability, portability, and scalability. The designs created as examples in Chapter 4 can be used for porting to different platforms from multiple vendors. The effect of change in the operating conditions of the system design can be easily tested by the tools used in this work.

The work flow presented in this work can be used for the rapid prototyping and development of the FPGA based SDR system. The models created for constructing a MIMO-OFDM transceiver prove the ability to convert a mathematical model into an executable specification that represents a PIM. A PIM model is used for converting to PSM after considering the hardware characteristics.

6.1.1 Hardware Platform

The hardware platform used in this work has a lot of limitations. It is limited by the logic resources available in it. Putting a large receiver like an OFDM radio on the FPGA of ML403 is a challenging task. The PowerPC^{TM} along with its peripherals takes up more than 60-70% of the logic resources available on the FPGA with very few slices remaining for the user generated IP circuit. In the future, FPGA platforms like higher versions of Virtex^{TM}-4, and Virtex^{TM}-5 devices can be used. Moreover, the transmission of data packets over the Ethernet cable from the USRP2 is the bottleneck for latency sensitive applications like OFDM. A development board that has a serial connector and communicates with the USRP2 using the MIMO connector port on it is the better choice. This connection is very much necessary for high bandwidth applications like WiMAX,WiFi, etc.

6.1.2 Implementation Issues

There are certain basic assumptions made in this work. Some of them may result in the performance degradation of the system. This includes the assumption that the OFDM symbols transmitted from both the antennas overlap with each other without lag. This will not be the case when there is an issue of multipath fading affecting each antenna independently.

There are some issues with the MATLAB^{TM} and Simulink® HDL coder tools used for HDL code generation. This can be seen in its Fixed Point advisor tool^{TM} of Simulink® that proposes some of the fixed point format for the model. The acceptance of the proposed fixed point format creates unexpected behavior in the operation of the system. Though the Fixed point advisor tool^{TM} is necessary for converting the floating point model to a fixed point model suitable for code generation, the user has to take care of his/her fixed point conversion guidelines for the expected output of the circuit.

There is an issue with USRP2 MIMO configuration. Presently, the firmware present on the USRP2 does not allow the clocks of multiple USRP2's to synchronize with each other. For the synchronized transmission of the signal from the slave USRP2 to the master USRP2 a MIMO connector cable is supplied by Ettus Research. But the current firmware present on it is not helping USRP2s sync with each other. The GNURadio community released the firmware for synchronizing USRP2 using the UHD driver a week before the defense of this work, this issue could not be resolved on time. The author of this work is currently looking into it, and the work will be completed in the next few weeks. Addressing this issue will make a significant impact on the open source community, providing an additional platform for the user to put his/her work on the FPGA.

There are some issues with mapping the HDL generated code in the real platform. The user is expected to have some knowledge of the FPGA architecture in terms of its blocks, clocking capability, timing constraints, etc. The HDL code running on one particular platform will not run on another platform because the basic architecture may be completely different.

6.2 Directions for Future Work

The present work deals with developing a prototype for SDR with active baseband processing provided by FPGA. This is accomplished using USRP2 as a front end and the ML403 as the baseband processing element. There are certain directions in which the present work can be extended.

6.2.1 Hardware/Software Co-design

The present work mostly explores baseband processing on the hardware. It is very much possible in the embedded environment to offload the less computational signal processing components on the less power intensive, slower processing elements like the GPP/Embedded processor. The models created using model based design will work for both embedded development and FPGA based hardware development. Thus, the given work can be extended to identifying the blocks that can go in embedded processors and blocks that can be offloaded for FPGA processing.

6.2.2 USRP2 Firmware

As mentioned in the previous chapters, the firmware currently provided for the USRP2 is highly unstable for MIMO applications. At start of this work it was widely assumed that the firmware along with the MIMO connector cable is enough to support MIMO configurations using the USRP2. But, over the course of the work it was realized that the connector cable did not serve any purpose until there is a firmware change. The author of this work is not able to identify the issue related to this firmware bug. Moreover, future work should look into the possibility of using the MIMO connector cable for transferring the data from/to USRP2 to/from FPGA evaluation board. There is a huge bottleneck in terms of the data rate if the Ethernet cable connector is used along with the PowerPC®. Future work should incorporate both the issues of the firmware and data transfer.

6.2.3 IEEE Standards Compliance

The present work does not cover all options of puncturing mechanisms as mentioned in the standards of IEEE 802.11a. This work can be extended to include these options in order to make a complete IEEE 802.11a transceiver chain. Moreover, the given system is not tested for power spillage outside the spectral band that may affect the operation as mentioned by the regulatory bodies.

6.2.4 MAC Layer Protocol

The present work mainly concerns the physical layer part of the baseband processing elements and there is no active support for the medium access control (MAC) layer protocol. This means that there is no mechanism for informing the transmitter about an erroneous packet being transmitted and also to perform carrier sensing before transmission. Thus, the work can be extended to accommodate different MAC protocol mechanisms like carrier sensing and acknowledgments.

6.2.5 Physical Layer Design

The work presented here can be extended to include other OFDM wireless standards like LTE, and DVB-T/H. Since most of the processing blocks implemented here are common to most OFDM based standards, this work can be extended to include changing the parameters of some of the blocks. The most challenging the blocks will be channel coding blocks like the Reed Solomon encoder/Decoder, Block Turbo coding and low density parity codes (LDPC) are becoming common in most of the current wireless standards. The implementation of these standards will make the user appreciate the usefulness of FPGA.

Bibliography

[1] R. Woods, J. McAllister, Y. Yi, and G. Lightbody, *FPGA-based implementation of Signal Processing Systems*. Wiley, 2008.

[2] M. Carrick, “Logical representation of FPGAs and FPGA circuits within SCA,” 2009.

[3] E. Teletar, “Capacity of Multiantenna Gaussian Channels,” in *European Transactions on Telecommunicaitons*, vol. 10, pp. 585–595, November/December 1999.

[4] G. J. Foschini and M. J. Gans, “On Limits of Wireless Communication in a Fading Environment when using Multiple Antennas,” in *Wireless Personal Communications*, vol. 6, pp. 311–335, 1998.

[5] M. Janakiraman, *Space-time codes and MIMO Systems*. Artech house universal personal communications, 2004.

[6] J. Reed, *Software Radio- A Modern approach to Radio engineering*. Pearson Education, 2006.

[7] R. Lauwereins, M. Engels, M. Ad, and J. A. Peperstraete, “Grape-II:Gaphical Rapid Prototyping Environment for Digital Signal processing systems,” in *The International Conference on Signal Processing Applications and Technology*, pp. 18–21, 1994.

[8] Gedae Inc. `http://www.gedae.com`.

[9] IEEE standards, “IEEE 802.11a-1999 Wireless LAN Medium Access Control (MAC) and Physical layer (phy) specifications-High speed physical layer in the 5 GHz band.”

[10] V. Tarokh, N. Seshadri, and A. Calderbank, "Space-Time codes for High Data Rate Wireless Communication: Performance Criterion and Code Construction," in *IEEE Transactions on Information Theory*, vol. 44, pp. 744–765, March 1998.

[11] S. M. Alamouti, "A simple Transmit Diversity Technique for Wireless Communication," in *IEEE Journal Selected Areas of Communication*, vol. 16, pp. 1451–1458, October 1998.

[12] G. J. Foschini, "Layered Space-time Architecture for Wireless Communications in a Fading Environment when using Multiple Antennas," in *Bell Labs Technology Journal*, vol. 6, pp. 41–59, 1996.

[13] T.-D. Chiueh and P.-Y. Tsai, *OFDM Baseband Receiver Design for Wireless Communication*. John Wiley and sons, 2007.

[14] J. Heiskala and J. Terry, *OFDM Wireless LANs: A Theoretical and Practical Guide*. Sams Publishing, 2002.

[15] T. Keller, L. Piazzo, P. Mandarini, and L. Hanzo, "Orthogonal Frequency Division Multiplex Synchronization Techniques for Frequency-selective Fading Channels," in *IEEE Journal on Selected Areas of Wireless Communication*, vol. 19, pp. 999–1008, 2001.

[16] T. M. Schimdl and D. C. Cox, "Robust Frequency and Timing Synchronization for OFDM," in *IEEE Transactions on Communications*, vol. 45, pp. 1613–1621, 1997.

[17] G. J. Foschini, "Layered Space-Time Architecture for Wireless Communications in a Fading environment when using Multiple Antennas," in *Bell Labs Technology Journal*, vol. 6, pp. 41–59, 1996.

[18] Y. Li, N. Seshadri, and S. Ariyavisitakul, "Channel Estimation for OFDM systems with Transmitter Diversity in Mobile Wireless Channels," in *IEEE Journal on Selected areas of Communication*, vol. 17, pp. 461–471, 1999.

[19] Y. Li, “Pilot-Symbol-Aided Channel Estimation for OFDM in Wireless Systems,” in *IEEE Transactions on Vehicular Technology*, vol. 49, pp. 1207–1215, 2000.

[20] S. Coleri, Mustafa, Ergen, A. Puri, and A. Bahai, “Channel Estimation Techniques based on Pilot Arrangement in OFDM systems,” in *IEEE Transactions on Broadcasting*, vol. 48, pp. 223–229, September 2002.

[21] IEEE standards, “IEEE 802.16-2004 Part-16 Air Interface for Fixed Broadband Wireless Access Systems.”

[22] “Configuring and Installing Linux on Xilinx FPGA Boards.” `http://www.ccl.ee.byu.edu/projects/LinuxFPGA/configuring.htm`.

[23] Xilinx, “git.xilinx.com Git.” `http://www.git.xilinx.com/cgi-bin/gitweb.cgi`.

Appendix A

IEEE 802.11a Packet Structure

This section provides the description of IEEE 802.11a packet structure ans some of its physical layer parameters. The understanding of the packet structure is necessary to appreciate its influence on the algorithms for packet detection, symbol timing estimation, and frequency offset estimation.

The PHY layer of IEEE 802.11a is divided into three entities based on functionality. They are physical management dependent(PMD) function, PHY convergence layer and layer management entity function.

The PMD layer is responsible for sending and receiving data between stations. This layer is responsible for physical layer functions like modulation, scrambling, channel coding , etc. The PHY convergence layer is necessary to make IEEE 802.11 MAC layer to work with minimum dependency on PMD layer. The layer management entity is responsible for management functions of the physical layer. In some configurations of IEEE 802.11, the state machines operating in the MAC layer requires input from the PHY layer. This exchange is controlled by between management entity functions of the MAC layer and PHY layer.

The procedure involved in the convergence function is called physical layer convergence procedure (PLCP). It defines a method for mapping PHY sublayer service data units (PSDU)

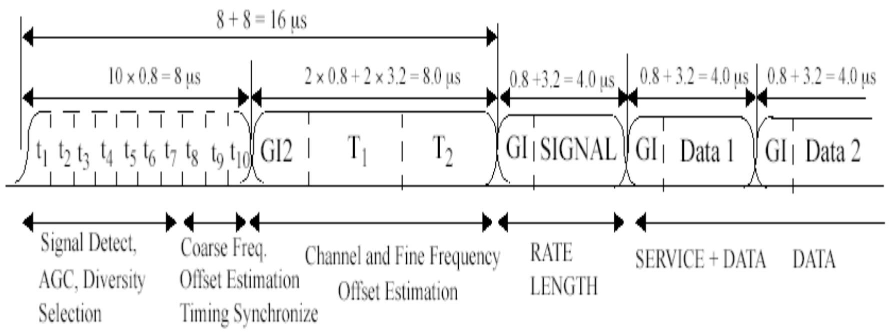

8 + 8 = 16 μs
10 × 0.8 = 8 μs
2 × 0.8 + 2 × 3.2 = 8.0 μs
0.8 + 3.2 = 4.0 μs
0.8 + 3.2 = 4.0 μs
0.8 + 3.2 = 4.0 μs
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10
GI2
T1
T2
GI
SIGNAL
GI
Data 1
GI
Data 2
Signal Detect, AGC, Diversity Selection
Coarse Freq. Offset Estimation Timing Synchronize
Channel and Fine Frequency Offset Estimation
RATE LENGTH
SERVICE + DATA
DATA

Figure A.1: Preamble Structure of IEEE 802.11a [9]

coming from the MAC layer into the frame format suitable for transmission and reception between two stations. During the transmission, the PSDU data are converted into physical layer protocol data units (PPDU). This includes PSDU data long with preamble, header, tail bits and pad bits. The PLCP frame format is shown in the following Figure A.1.

It is made up of OFDM PLCP preamble, OFDM PLCP header, PSDU, tail bits and pad bits. The PCLP header consists of modulation and data rate specific parameters. The header constitutes a separate OFDM symbol which is BPSK modulated and coded with the coder rate $\frac{1}{2}$. The PCLP preamble field consists of repetitions of short sequence and 2 repetitions of long sequence. According to the specification, the short sequence are used for automatic gain control (AGC) convergence, timing acquisition and coarse frequency synchronization while the long sequence aids in channel estimation and fine frequency synchronization.

A short OFDM symbol consists of 12 carriers modulated by the sequence S given by,

$$\begin{aligned} S_{-26,26} = \sqrt{\frac{13}{6}}\{&0,0,1+j,0,0,0,-1-j,0,0,0,1+j,0,0,0,-1-j,0,0,0,-1-j,0,0,0,\\ &1+j,0,0,0,0,0,0,0,-1-j,0,0,0,-1-j,0,0,0,1+j,0,0,0,1+j,0,0,0,1+j,0,0,\\ &0,0,1+j,0,0\} \end{aligned} \tag{A.1}$$

The normalization factor $\sqrt{\frac{13}{6}}$ is used to average power of the resulting OFDM symbol that uses only 12 out of the 52 sub-carriers. Since $S_{-26,26}$ produces spectral lines at multiples of 4 it produces periodicity of $\frac{T_{FFT}}{4} = 0.8\mu s$. Thus the short preamble, with 10 repetitions, (T_{Short}) has the duration of $8\mu s$

The long OFDM symbol is given as,

$$L_{-26,26} = \{1, 1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 0, 1, -1, \\ -1, 1, 1, -1, 1, -1, 1, -1, -1, -1, -1, -1, 1, 1, -1, -1, 1, -1, 1, -1, 1, 1, 1, 1\} \quad \text{(A.2)}$$

The long duration(T_{Long}) has the duration of $1.6 + 2 \times 3.2 = 8\mu s$. Thus, the total length of the training symbols are $16\mu s$.

There are other fields in the header. They are the SIGNAL field with LENGTH, RATE and tail bits. The LENGTH field indicates the number of octets transferred between MAC and physical layer in single time unit. The RATE field helps to identify the data rate of the transmission and the tail bits are used to return the convolutional encoder state back to zero in order to improve error performance.

Some of the timing related parameters for OFDM PLCP are shown in following table.

Table A.1: Physical Layer Parameters

Parameter	Value
Number of data sub-carriers	48
Number of pilot sub-carriers	4
Total number of sub-carriers	52
Total number of sub-carriers	64
Sub-carrier spacing	0.3125MHz
FFT duration	$3.25\mu s$
Preamble duration	$16\mu s$
Guard interval duration	$0.8\mu s$
OFDM symbol duration	$4\mu s$

According to the specification, 12 out of the 64 sub-carriers present in the fringes of the OFDM symbol are not used for transmission in order to act as a smoothing function. This will reduce the sidelobes in the spectral function. In this work, all the 60 data sub-carriers are used for transmitting data. The performance degradation due to side lobes is not taken into account here.

Appendix B

Simulink® HDL coder compliant block diagrams

B.1 Alamouti Encoder

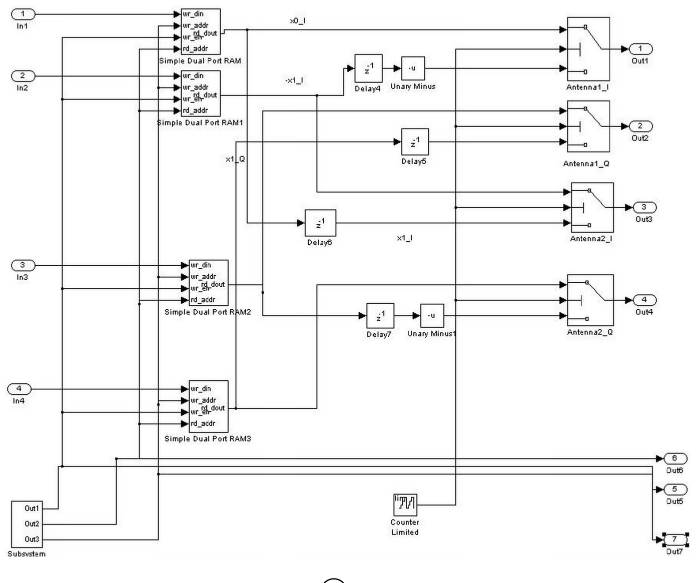


Figure B.1: Simulink® Block for Alamouti Encoder

B.2 Scrambler/De-Scrambler

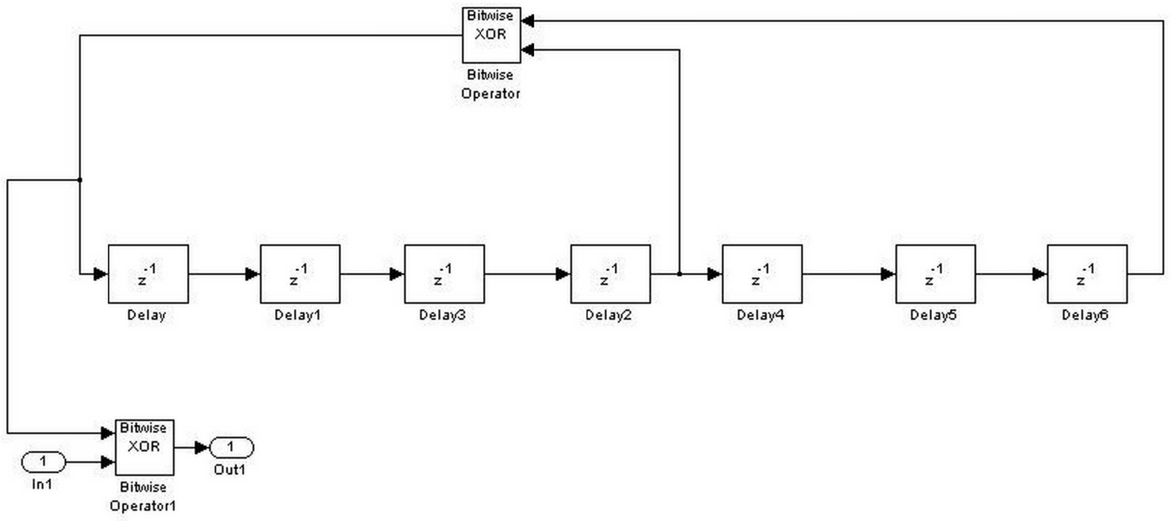


Figure B.2: Simulink® Block for Scrambling/De-scrambler

B.3 Convolutional Encoder

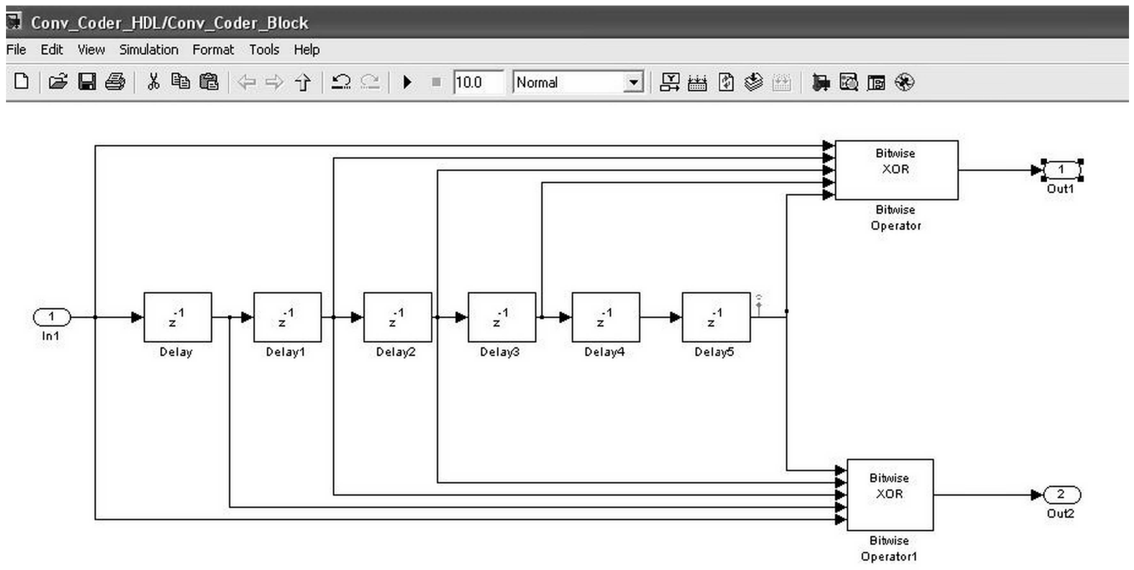


Figure B.3: Simulink® Block for Convolutional Encoder

B.4 Packet Detection

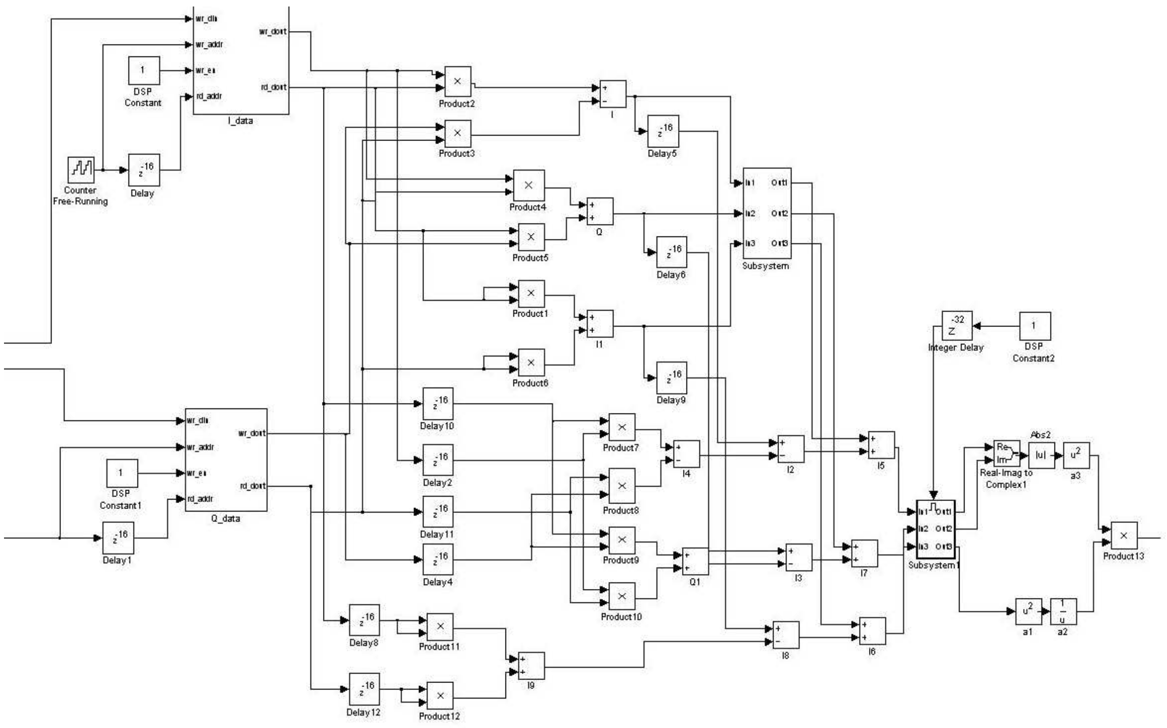


Figure B.4: Simulink® Block for Packet Detection Algorithm

B.5 Symbol Timing Estimation

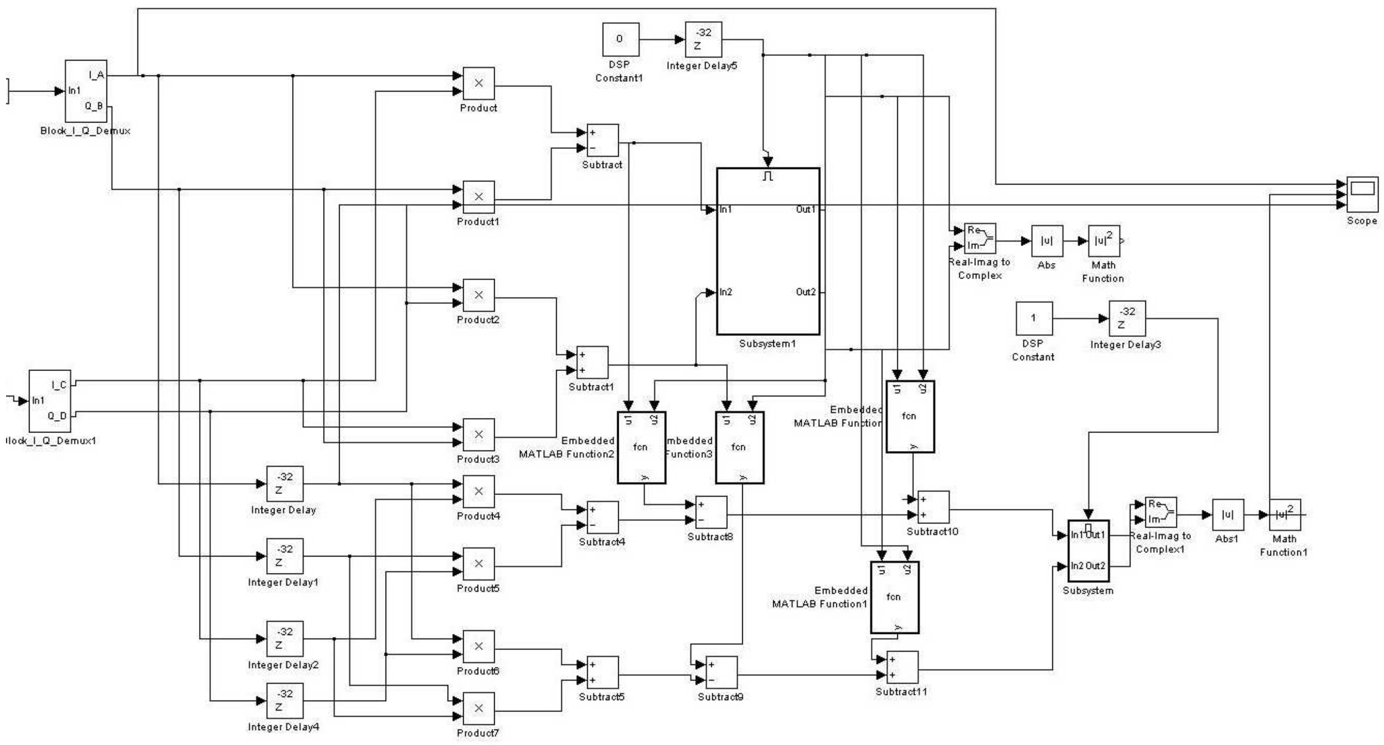


Figure B.5: Simulink® Block of Symbol Timing Estimation Algorithm

Appendix C

Simulink® Output Examples

C.1 Convolutional Encoder

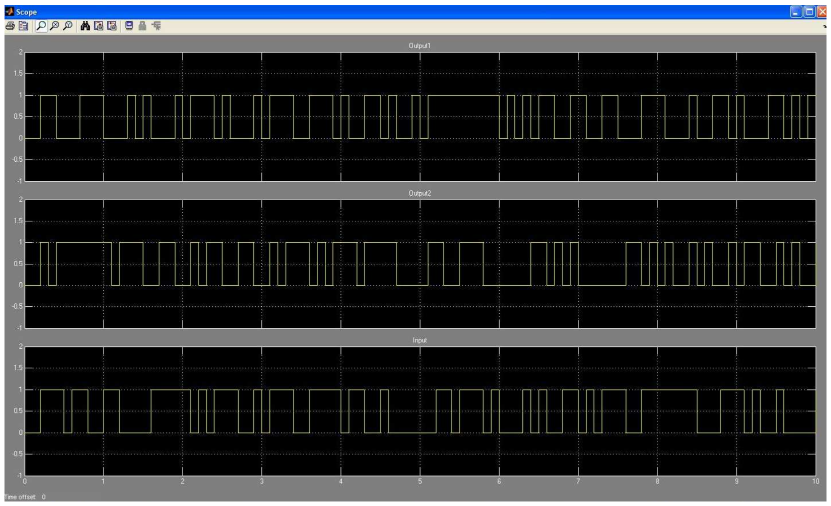


Figure C.1: Simulink® Output of Convolutional Encoder

C.2 Alamouti Encoder

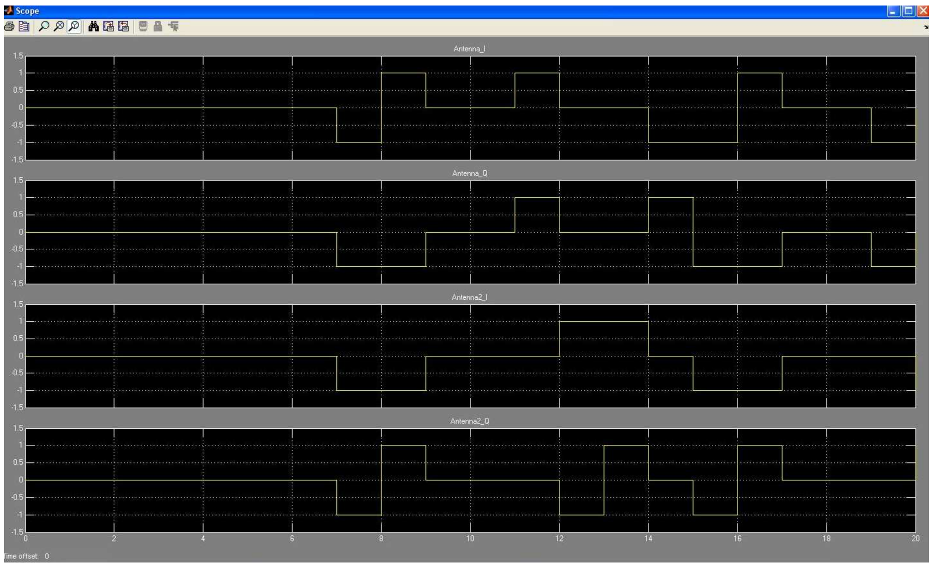


Figure C.2: Simulink® Output of Alamouti Encoder

C.3 Scrambler

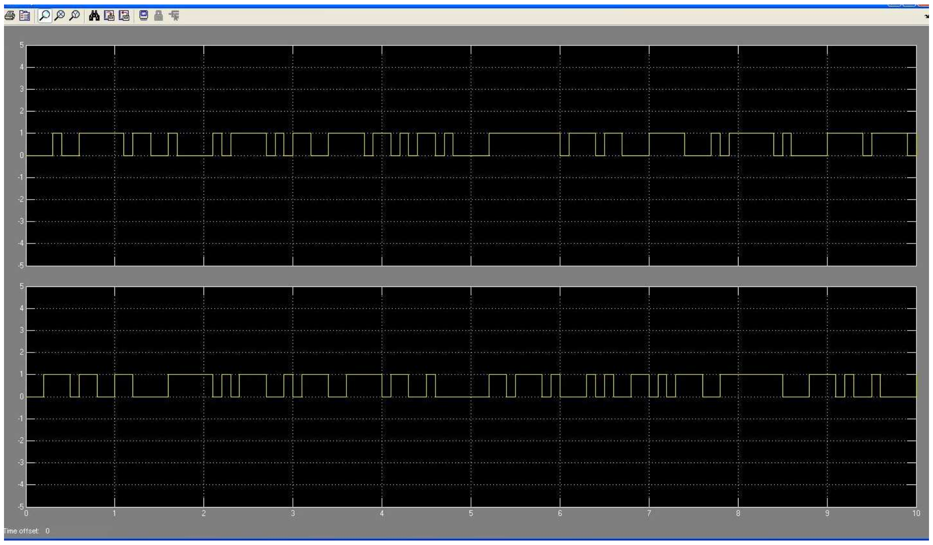


Figure C.3: Simulink® Output of Scrambler/De-scrambler

C.4 Packet Detection

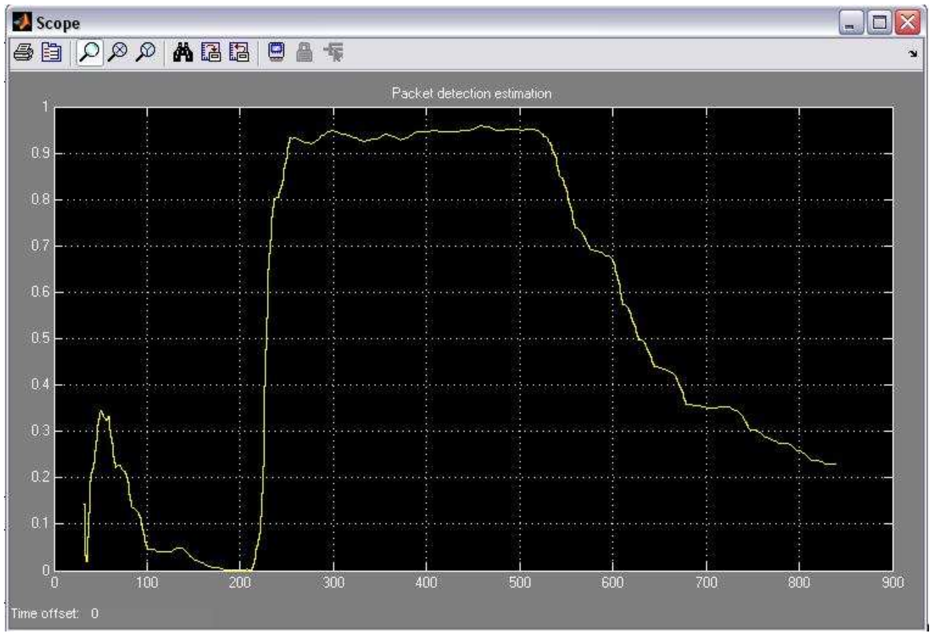


Figure C.4: Simulink® Output of Packet Detection Algorithm

C.5 Symbol Timing Estimation

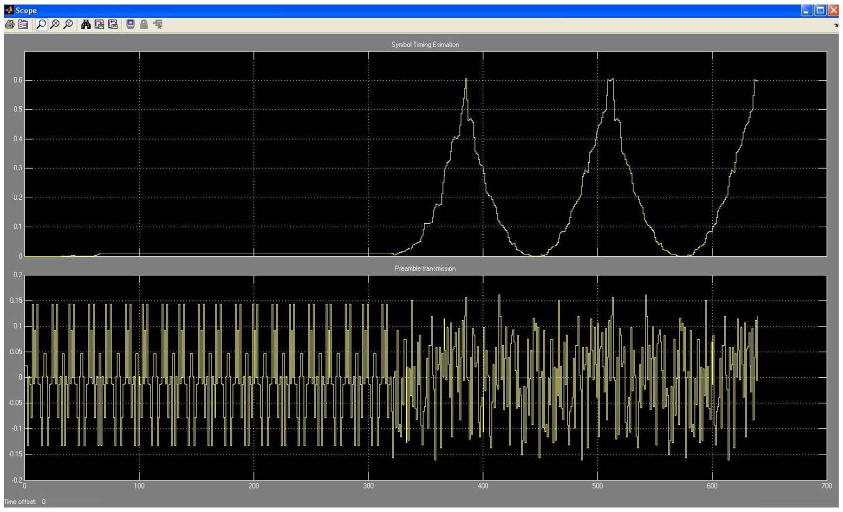
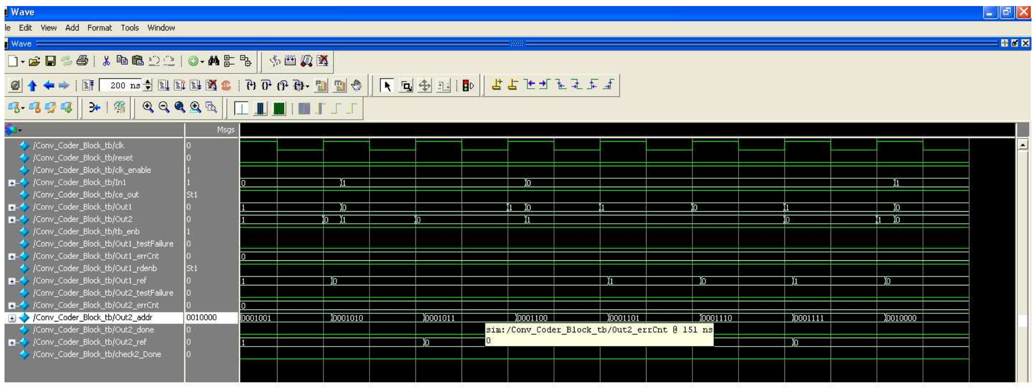


Figure C.5: Simulink® Output of Symbol Timing Estimation Algorithm

Appendix D

Modelsim® Output Examples

D.1 Convolutional Encoder



```
/Conv_Coder_Block_tb/clk                 0
/Conv_Coder_Block_tb/reset               0
/Conv_Coder_Block_tb/clk_enable          1
/Conv_Coder_Block_tb/In1                 1
/Conv_Coder_Block_tb/ce_out              St1
/Conv_Coder_Block_tb/Out1                0
/Conv_Coder_Block_tb/Out2                0
/Conv_Coder_Block_tb/tb_enb              1
/Conv_Coder_Block_tb/Out1_testFailure    0
/Conv_Coder_Block_tb/Out1_errCnt         0
/Conv_Coder_Block_tb/Out1_rdenb          St1
/Conv_Coder_Block_tb/Out1_ref            0
/Conv_Coder_Block_tb/Out2_testFailure    0
/Conv_Coder_Block_tb/Out2_errCnt         0
/Conv_Coder_Block_tb/Out2_addr           0010000
/Conv_Coder_Block_tb/Out2_done           0
/Conv_Coder_Block_tb/Out2_ref            0
/Conv_Coder_Block_tb/check2_Done         0
```

```
0001001  0001010  0001011  0001100  0001101  0001110  0001111  0010000
sim:/Conv_Coder_Block_tb/Out2_errCnt @ 151 ns
0
```

Figure D.1: Modelsim® Output of Convolutional Encoder

D.2 Alamouti Encoder

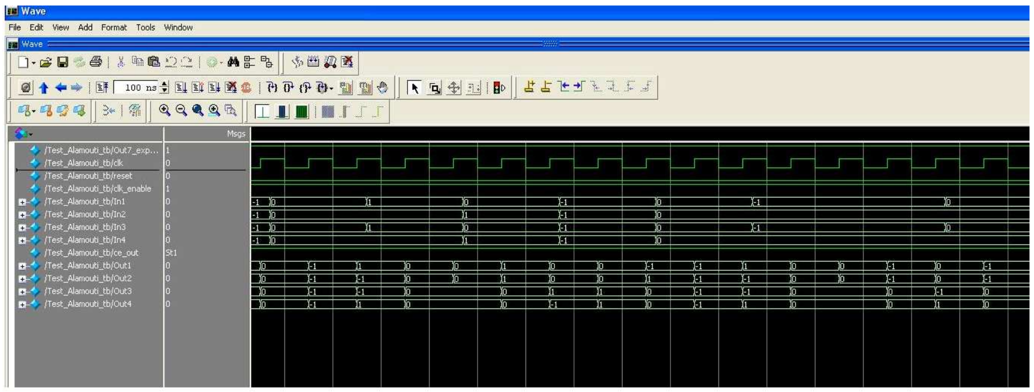


Figure D.2: Modelsim® Output of Alamouti Encoder

D.3 FFT

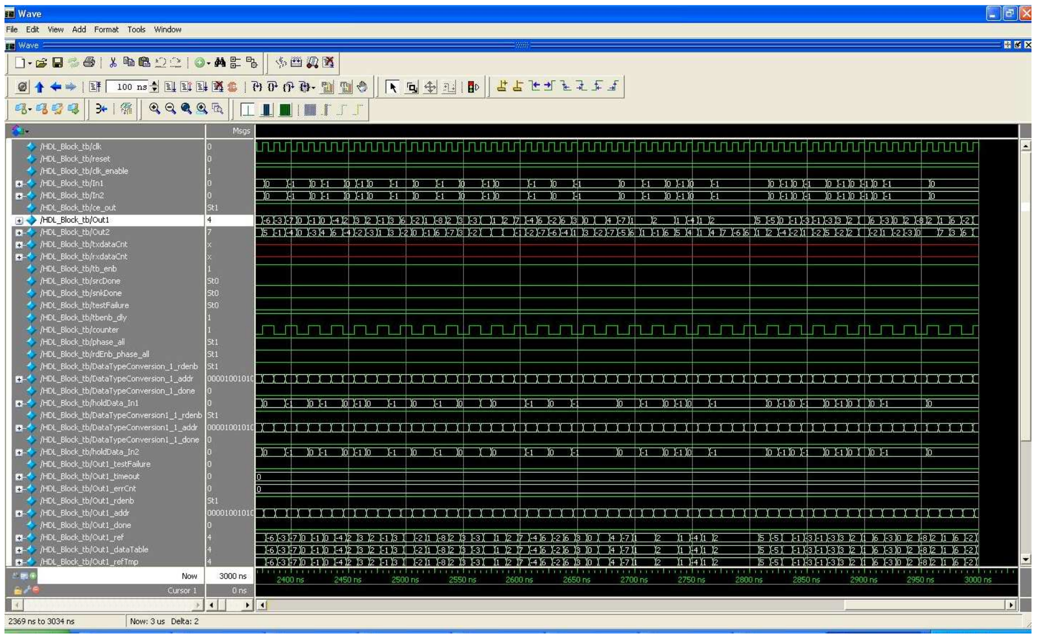
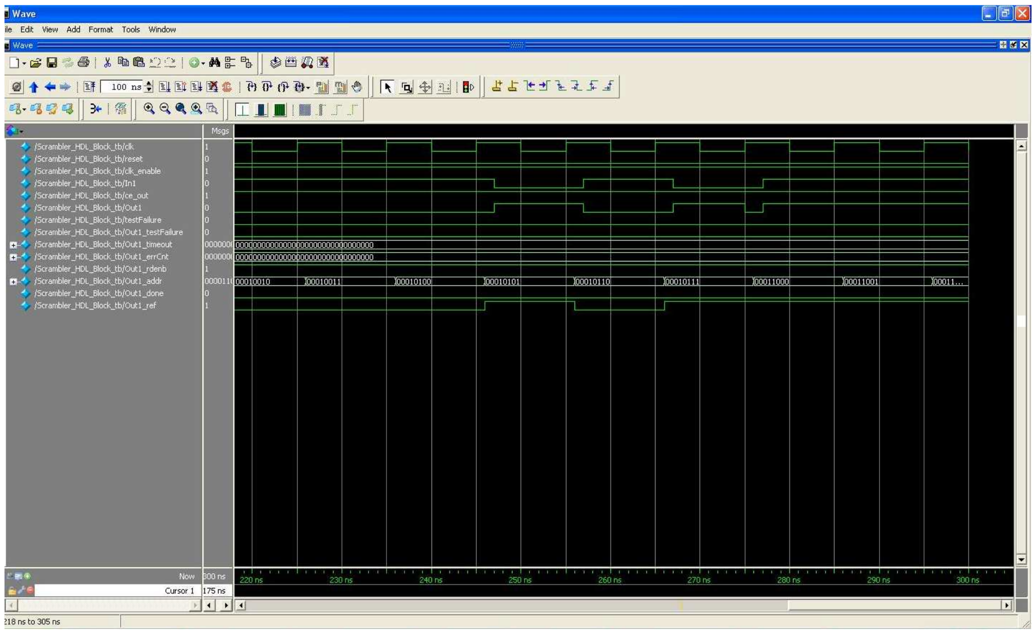


Figure D.3: Modelsim® Output of FFT

D.4 Scrambler/De-Scrambler



```
/Scrambler_HDL_Block_tb/clk              1
/Scrambler_HDL_Block_tb/reset            0
/Scrambler_HDL_Block_tb/clk_enable       1
/Scrambler_HDL_Block_tb/In1              0
/Scrambler_HDL_Block_tb/ce_out           1
/Scrambler_HDL_Block_tb/Out1             0
/Scrambler_HDL_Block_tb/testFailure      0
/Scrambler_HDL_Block_tb/Out1_testFailure 0
/Scrambler_HDL_Block_tb/Out1_timeout     0000000 00000000000000000000000000000000
/Scrambler_HDL_Block_tb/Out1_errCnt      0000000 00000000000000000000000000000000
/Scrambler_HDL_Block_tb/Out1_rdenb       1
/Scrambler_HDL_Block_tb/Out1_addr        0000110 00010010 00010011 00010100 00010101 00010110 00010111 00011000 00011001 00011...
/Scrambler_HDL_Block_tb/Out1_done        0
/Scrambler_HDL_Block_tb/Out1_ref         1
```

Figure D.4: Modelsim® Output of Scrambler/De-scrambler

Appendix E

Comb-type Channel Estimation Technique

The comb-type channel estimation is performed for the channels that varies over every OFDM symbol duration. The estimation of the channel over every symbol is possible because of the insertion pilot carriers at equal interval inside the OFDM symbol. The pilot based estimation is recommended for both IEEE 802.11a and IEEE 802.16e standard. In the case of former, 4 pilot carriers are recommended over 64 carriers per OFDM symbol while in the case of the latter 8 to 64 is recommended. As the number of pilot sub-carrier increases, the performance of the estimator increases.

If O is the OFDM symbol, then N pilot carriers are inserted into the symbol according to the follwing formula:[20]

$$O(k) = O(mL + 1) \tag{E.1}$$

At the pilot sub-carrier frequencies, the *least squares* estimation technique was used whose output is interpolated to other sub-carrier frequencies. It is given by : [20]

$$H_{pilot} = \frac{Y_{pilot}}{X_{pilot}} \tag{E.2}$$

where Y_{pilot} and X_{pilot} are outputs and inputs at pilot frequencies.

E.1 Interpolation Techniques

An efficient interpolation techniques are required for the correct operation of the multi antenna systems. In this work, three channel estimation techniques were considered based on the computation complexity and latency. They are *linear interpolation, spline cubic interpolation and second order interpolation.* The estimation of the channel at the non-pilot frequencies using the linear interpolation technique[20] is given by,

$$\begin{aligned} H_{np}(k) &= H_{pilot}(mL + \mathit{1})0 \leq l < L \\ &= H_{pilot}(m + \mathit{1}) - H_{pilot}(m) \times \frac{l}{L} + H_{pilot}(m) \end{aligned} \tag{E.3}$$

In the case of second order interpolation, the estimated channel is given by,

$$\begin{aligned} H_{np}(k) &= H_{pilot}(mL + \mathit{1})0 \leq l < L \\ &= c_{\mathit{1}} \times H_{pilot}(m - \mathit{1}) + c_{\mathit{0}} \times H_{pilot}(m) + c_{-\mathit{1}} \times H_{pilot}(m + \mathit{1}) \end{aligned} \tag{E.4}$$

where

$$\begin{aligned} c_{\mathit{1}} &= \frac{\alpha}{2}, \\ c_{\mathit{0}} &= -(\alpha - 1)(\alpha + 1) \\ c_{-\mathit{1}} &= \frac{\alpha(\alpha + 1)}{2} \end{aligned} \tag{E.5}$$

$$\tag{E.6}$$

Here $\alpha = \frac{l}{L}$.

In the case of the *low-pass filtering* approach, the pilot symbols has to converted again to time domain and the interpolated with the low pass filter with inserted between the pilot location. By this manner, the minimum mean square between the interpolated symbols are reduced. The cost of conversion to time domain itself makes this approach infeasible.

In *spline-cubic* interpolation, the pilot symbols are interpolated along the spline function to produce the smooth polynomial.