# EDUCATIONAL APPLICATIONS OF PARTIAL RECONFIGURATION OF FPGAS

By

Maysam Sarfaraz

A Thesis
Submitted to the Faculty of the
University of Tennessee at Chattanooga
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Electrical Engineering

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

May 2011

To the Graduate Council:

I am submitting herewith a thesis written by Maysam Sarfaraz entitled “Educational Applications of Partial Reconfiguration of FPGAs” I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Dr. Stephen Craven
Major Professor

We have read this thesis
and recommend its acceptance:

Dr. Ahmed Eltom

Dr. Raziq Yaqub

Accepted for the Council:

Dr. Jerald Ainsworth
Dean of the Graduate School

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

## ABSTRACT

The use of partial reconfiguration (PR) in reconfigurable systems such as Field Programming Gate Arrays (FPGAs) has gained a lot of attention during the past ten years. Recently, Xilinx has released the first commercially available PR implementation for its FPGAs. However, there is a lack of educational tools for PR instruction. In addition, the design and implantation of PR on FPGAs can be beneficial compared with the current communications tools within academia. This thesis presents the design and simulation of several basic modulation schemes within Simulink and System Generator for educational applications in communications classes. In addition, the implementation process of creating and testing additional partial bitstreams will take a few minutes rather than a few hours, which make this reconfigurable system a suitable system for educational applications. Furthermore, each modulation scheme can be implemented without any use of Hardware Description Language (HDL), embedded, and software development design. After implementing these models on an FPGA the results of these implementations are analyzed and compared with the simulation results. The results demonstrate the proper implementation of these PR designs on FPGAs.

## ACKNOWLEDGEMENTS

This thesis was written under the direction and supervision of Dr. Stephen Craven. I would personally like to thank Dr. Stephen Craven as my advisor for his patient guidance throughout the course of my thesis at the University of Tennessee at Chattanooga. My appreciations also go to the committee members of this thesis, Dr. Ahmed Eltom and Dr. Raziq Yaqub.

# TABLE OF CONTENTS

## LIST OF TABLES

# LIST OF FIGURES

## LIST OF ABBREVIATIONS

| | |
|---|---|
| **AM** | Amplitude Modulation |
| **ASIC** | Application-specific integrated circuit |
| **BEE** | Berkeley Emulation Engine |
| **CLB** | Configurable Logic Block |
| **CPLD** | Complex Programmable Logic Device |
| **CPU** | Central processing units |
| **DSB-SC** | Double SideBand-Suppressed Carrier |
| **DSP** | Digital Signal Processing |
| **EDK** | Embedded Development Kit |
| **FM** | Frequency Modulation |
| **FPGA** | Field Programming Gate Array |
| **HDL** | Hardware Description Language |
| **IC** | Integrated Circuit |
| **ICAP** | Internal Configuration Access Port |
| **LAB** | Logic Array Blocks |
| **MATLAB** | Matrix Laboratory |
| **OSSIE** | Open-Source SCA Implementation Embedded |
| **PM** | Phase Modulation |

| | |
|---|---|
| **PR** | Partial Reconfiguration |
| **RP** | Reconfigurable Partition |
| **SCA** | Software Communications Architecture |
| **SDR** | Software Defined Radio |
| **SDK** | Software Development Kit |
| **WARP** | Wireless Open Access Research Platform |
| **XST** | Xilinx Synthesis Technology |

# CHAPTER I

# INTRODUCTION

Nowadays, reconfigurable systems such as Field Programming Gate Arrays (FPGAs) are becoming popular for different applications in areas such as defense, automotive, broadcasting, high performance computing and data storage, wired communications, wireless communication, etc. This fact provides great opportunities for research and educational communities to enhance and create more efficient ways to leverage the use of these reconfigurable devices. One of the capabilities of these systems is partial reconfiguration (PR) in which allow designers to overcome lack of resources of FPGAs. This thesis looks into the advantages of partial reconfiguration capabilities of FPGAs and leverages its usage in educational and research applications.

## *Advantages of Reconfigurable Systems*

There are several reconfigurable systems in the market. FPGAs are the most common reconfigurable systems that used among all other reconfigurable systems. The FPGAs usually are compared to Application-Specific Integrated Circuits (ASICs). ASIC is an Integrated Circuit (IC) for a specific purpose, and its hardware configuration cannot change after its production. They are more expensive to design compared with FPGAs. However, they can operate at higher frequencies with lower power consumption compared with FPGAs.

FPGAs give an opportunity to the user to change the hardware reconfiguration unlike the ASIC. This capability can avoid series of delays due to hardware conflicts [1]. For example, an operation can be delayed due to lack of certain hardware resources such as DSP block in an ASIC. Moreover, this reconfigurable capability made these devices an ideal tool for teaching digital design in academia. In addition, FPGAs have a fast and cheap implementation time versus ASIC designs.

There are several companies that produce FPGAs in the market. Some of these companies are Xilinx [2], Altera [3], Lattice [4], Actel [5]. In addition, several educational and evaluation kits are available from different vendors such as Avnet [6] and Digilent [7] for Xilinx FPGAs.

## *Advantages of Partial Reconfiguration*

Partial reconfiguration helps to reduce cost and board space, providing a flexibility to change a portion of the system dynamically without shutting down the entire system [8]. Figure 1 illustrates how the use of PR will reduce the implementation size of the design on the FPGAs. This figure illustrates an FPGA with a design that three partitions of that are dedicated for static regions and the rest of the FPGA can be used for PR implementation. The left portion of this figure shows this FPGA without PR implementation. This portion can hold only few different designs within partial reconfiguration region of the FPGA. However, after PR implementation (right portion of the figure) a lot of additional designs can be stored outside of the FPGA and swapped if there is need for change during operation of the FPGA.

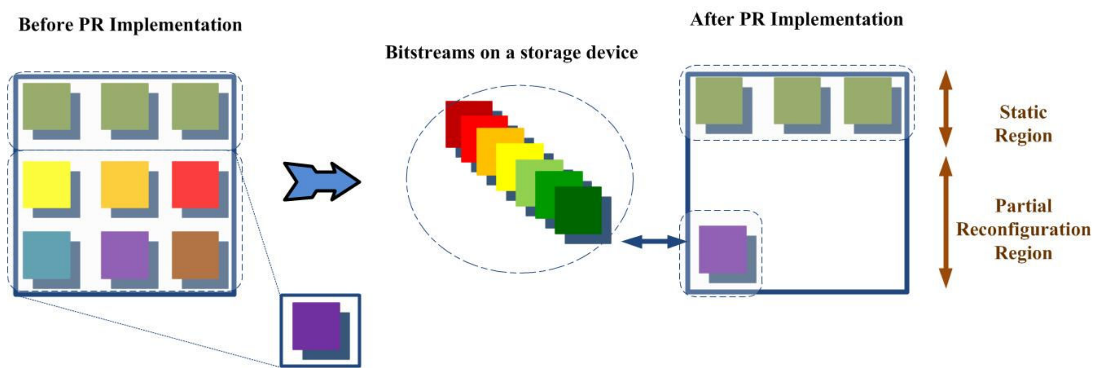

Before PR Implementation
Bitstreams on a storage device
After PR Implementation
Static Region
Partial Reconfiguration Region

Figure 1. Reducing Size after Implementing Partial Reconfiguration

In addition, after PR implementation the remaining unused resources of the FPGA can be used for expanding the design with more additional features. The other advantages of the PR implementation would be reducing the time that required for completing the entire design since adding additional configurations to existing design can be process a lot faster than starting a design from the beginning. Also, using PR implementations allow the user to use smaller FPGAs rather than the bigger ones and that will reduce the power consumption of entire system.

*Existing Educational and Research Projects*

This section surveys different types educational and research project that are available to the academia that could be used for PR implementation such as Software Communications Architecture (SCA) Implementation Embedded (OSSIE) [9], Wireless Open Access Research Platform (WARP) [10], Berkeley Emulation Engine (BEE) [11-13]. None of these projects uses PR implementation; however, these boards are capable of PR and they can be used for future research.

One of the configurable systems for educational use in areas of communications is TIMS 301, which is made by the EMONA [14]. TIMS can be used only for educational purposes in the areas of telecommunication unlike an FPGA that can be used in wider areas such as digital design, communications, signal processing, image processing, networking, etc. As a result, using partial reconfiguration on FPGAs can be beneficial for educational applications in wider areas. In addition, FPGAs are used for building actual radio systems unlike the TIMS systems. In addition, the FPGAs less expensive compared with the TIMS systems. Figure 2 shows TIMS-301 basic system model.

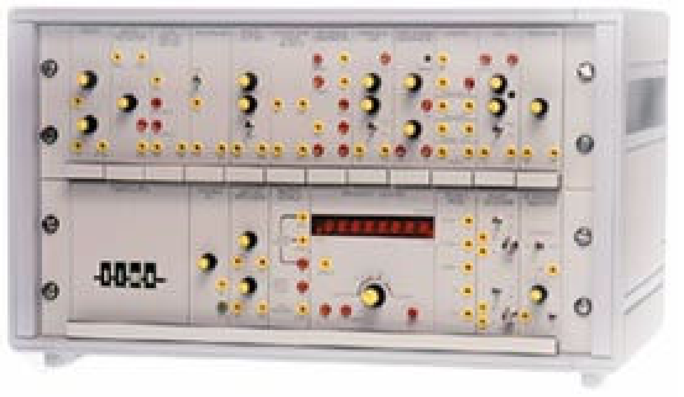


Figure 2. TIMS-301 Basic System Model [15]

*Thesis Goals and Objectives*

This educational and research project is an interdisciplinary design that draws from electrical and computer engineering disciplines. The main goal of this research is to design, analyze, and setup a partial reconfigurable system with emphasis on communications and Digital Signal Processing (DSP) for educational and research applications. This system will be used for instructors to demonstrate design process and configuration of different modulation schemes on an FPGA by using System Generator. Using PR implementation will allow students to design and test additional partial bitstreams in few minutes compared to a few hours of the HDL design process.

## *Thesis Overview*

Chapter I presents an overview of the partial reconfigurable systems and discusses project goals and objectives. Chapter II presents theoretical background of reconfigurable systems and partial reconfiguration. Chapter III addresses different design methods for analysis of performance of proposed reconfigurable systems and discusses pros and cons of simulation tools such as Simulink [16] and System Generator [17] for the scope of this research. In addition, Chapter III provides procedural description of Simulink and System Generator simulation tools for modeling and analysis of several modulation designs for educational applications. Chapter IV after comparing simulation results from Chapter III describes how to implement each modulation scheme within an FPGA by using partial reconfiguration. In addition, Chapter IV shows that adding additional partial bitstreams will take only a few minutes rather a few hours. Finally, Chapter V presents conclusions of this research effort and recommends further research toward enhancement of this educational reconfigurable system.

# CHAPTER II

# BACKGROUND

This chapter provides an overview of FPGAs such as different types of FPGA technologies, and their capabilities. It also discusses some related theories for partial reconfiguration process and its implementation. In addition, different design tools for FPGAs are introduced. Finally, Xilinx design tools are describes in details particularly those tools that are required for implementation of partial reconfiguration on FPGA.

## *FPGA Architecture and Technology*

The FPGA architecture consists of logic blocks for implementing logic functions, interconnection resources for routing and input/output blocks for outside connections [18-20]. About 90% of the FPGA is made of programmable interconnects the rest of the FPGA is made of logic blocks [1]. Figure 3 shows a generic FPGA architecture. In addition, additional resources such as, memory, DSP blocks, embedded processors [21], etc. maybe available on a FPGA.

Configurable Logic Blocks (CLBs) [19], [20], [22] are the basic blocks of Xilinx FPGAs. Also, Altera refers to these logic blocks as Logic Array Blocks (LABs) [23]. In addition, each CLB contains several look up tables (LUTs). The LUT is used by most of FPGA vendors mainly because an n-input LUT is capable of implementing of any n-input logic function [24]. In other word, A LUT is a small memory that directly stores the truth table for a digital circuit.  The number of CLBs and their configuration depend on the size and type of an FPGA.  For example each CLB in Virtex-5 the FPGAs that used in implementation of PR is made up two slices [25].  In addition, each slice in Virtex-5 family has four function generators (or LUTs) and four storage elements [25].

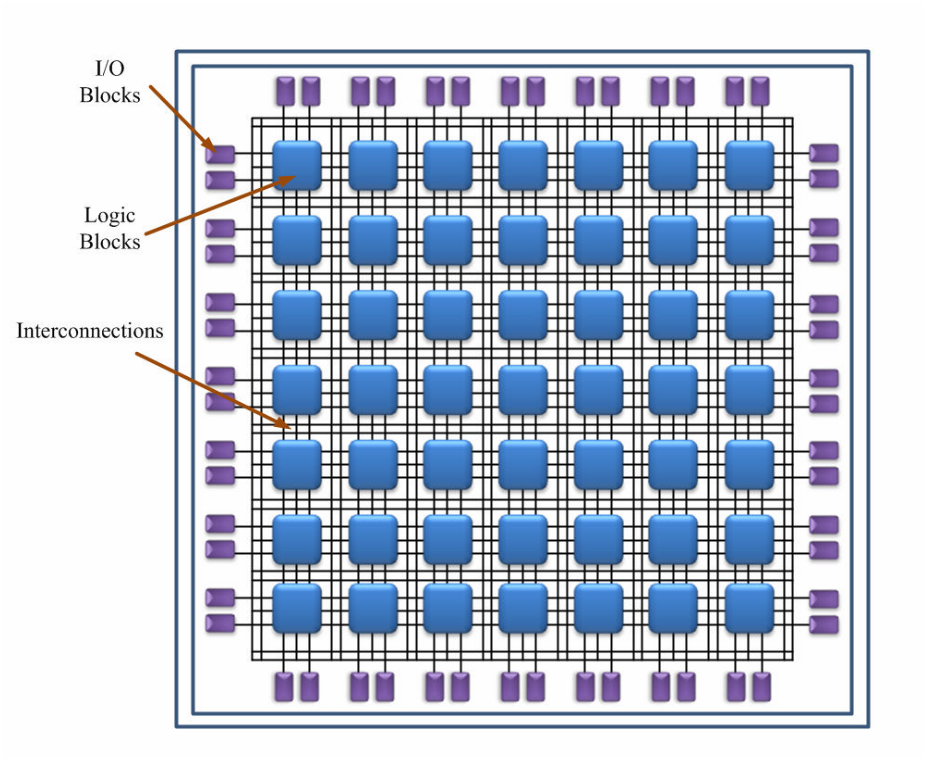


Figure 3. Generic FPGA Architecture

Figure 4 illustrates configurable logic block components of the XUPV5 board. The 6-Input LUT Architecture is shown on the left side of this figure and a flip-flop on the right.  Also, in this figure CIN and COUT refer to fast carry adder input and output chain which is shown in the middle. The 6-Input LUT Architecture allows the implementation of truth tables with up to 64 combinations of six different input signals. In addition, these LUTs can be used as shift registers or RAM. [24].

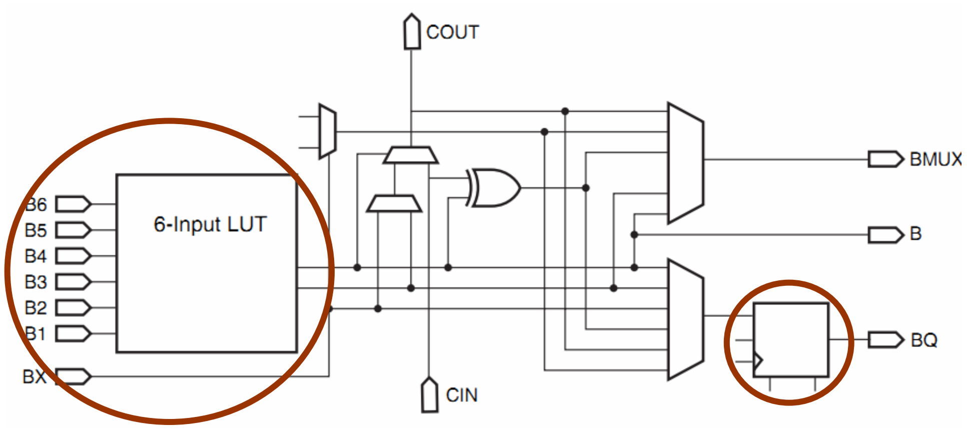


Figure 4. Configurable Logic Block Components [26]

There are couples of educational boards from Digilent, which have PR capabilities. These boards are Genesys [27] and Virtex-5 OpenSPARC (XUPV5) [28]. Table 1 compares the architecture of these FPGA boards.

Table 1. Comparison of Two FPGA Boards from Digilent

| Board | FPGA | CLBs | DSP48E Slices | Academic Price |
|---|---|---|---|---|
| Genesys | XC5VLX50T | 3,600 [25] | 48 [25] | $449.00 [27] |
| Virtex-5 OpenSPARC (XUPV5) | XC5VLX110T | 8640[25] | 64 [25] | $750.00 [28] |

*Partial Reconfigurations in FPGAs*

There are several types of reconfiguration for reconfigurable systems. These types are configurable, reconfigurable, run-time reconfigurable/dynamically reconfigurable and partially runtime reconfigurable [29]. The configurable configuration can be configured one time by using fuse or anti-fuse technology. Reconfigurable system can be erasing and programmed over and over. Run-time reconfigurable on other hand, allows the reconfiguration of the system during its operation [29]. Finally, the partial reconfiguration allows reconfiguration of portion of hardware while the static part is operational [29]. PR can be used in wide variety of applications such as image processing, Software Define Radio (SDR), encryption, etc.

The concept of partial reconfiguration in reconfigurable system has gained a lot of attentions within academia during past few years. However, the partial reconfiguration never been used for educational applications. Xilinx recently released the first commercially available PR implantation for its FPGAs [8]. This implementation supported for Virtex-4, Virtex-5, and Virtex-6 FPGAs. In addition, Xilinx has several tutorials for this PR implementation [30]. However, at this point these tutorials are not covering a detail PR implementation in the area of communications. Using partial reconfiguration in communications classes allow the user to design, build, implement, and test communication modules in a few minutes rather than a few hours on an FPGA.

### *FPGA Design Tools*

The current version of Xilinx ISE design suite tools is version 13.1 [31]. The ISE consists of several tools such as PlanAhead [32] for design analysis, ISE project manager as a Synthesizer, ISim [33] as a simulator, Embedded Development Kit (EDK) [34], [35] for embedded designs, Software Development Kit (SDK) [34], [35] for software applications targeting soft and embedded processors within FPGAs, System Generator [36] for DSP applications, etc. In this section, the ISE design tools is discussed since the

Xilinx provided these tools to universities free of charge for educational and research applications.

# CHAPTER III

# DESIGNS AND METHODS

This chapter looks into several simulation results within Simulink and System Generator. First, the theory behind some of the most common modulations such as AM and Double SideBand-Suppressed Carrier (DSB-SC) modulations is presented. Next these modulations were built and tested within Simulink. Next, the audio signal was imported to the Simulink and the simulation result was analyzed before implementation of the design within System Generator. Finally, these designs were implemented within System Generator for educational applications.

## *Modulation in Communication Systems*

Communications system can transmit and receive analog or digital signals. Analog modulation can be characterized as Amplitude Modulation (AM), Frequency Modulation (FM), or Phase Modulation (PM). In these modulations the information or baseband signal will be attached to corresponding properties of carrier signal, for example in AM modulation the information is attached to the carrier amplitude [37]. This section covers two modulation schemes. These modulations are AM and DSB-SC modulations. First, AM modulation is discussed then looked into DSB-SC modulation for educational applications. In order to analyze AM modulation first a message $m(t)$ and a carrier wave $c(t)$ are defined. These waves are illustrated in Equation 1 [38]. In this equation, $A_m$ is m message signal amplitude, $A_c$ is carrier signal amplitude, $f_m$ is message signal frequency, and $f_c$ is carrier signal frequency.

$$\begin{gathered} m(t) = A_m \cos(2\pi f_m t), \\ c(t) = A_c \cos(2\pi f_c t) \end{gathered} \tag{1}$$

Now the AM modulated wave $s(t)$ is defined by [38]

$$s(t) = A_c[1 + k_a m(t)]\cos(2\pi f_c t) \tag{2}$$

In Equation 2 the $k_a$ is called the amplitude sensitivity. In order to detect envelope of modulated wave $s(t)$ properly and avoid envelope distortion the amplitude of $k_a m(t)$ should be less than one.

Demodulation of the AM modulated wave is done by envelope detection method. This method and its circuit illustrate in Figure 5. This method of the demodulation recovers the baseband signal by removing half of the envelope, and after this process uses a low-pass filter to remove high frequency components of the recovered signal.

AM modulation was one the first modulations that was developed but it is still used in standard AM frequency spectrum from 535 to 1605 kHz [39]. Next, the AM modulation and demodulation is constructed and analyzed.

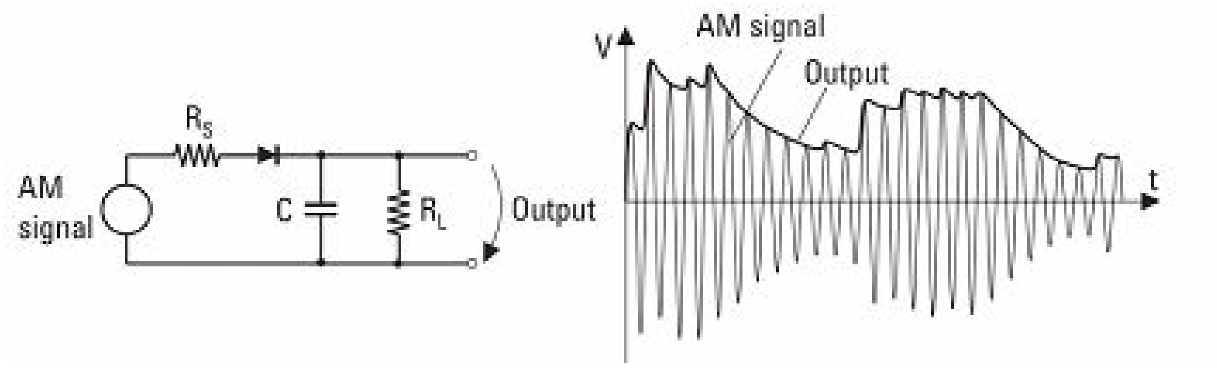


Figure 5. AM Demodulator using an Envelope Detector in the Time Domain (right) and its Circuit Configuration (left) [37]

Next, the characteristic of the Double SideBand-Suppressed Carrier (DSB-SC) modulation is discussed. The DSB-SC has an advantage of simple modulation which is multiplying the message signal and carrier signal directly. However, it required a much more complex demodulator circuit. In addition, this modulation reaches to zero when the message signal is turned off [38]. In this section only coherent detection or synchronous demodulation method is discussed, which assumed that local oscillator is synchronized with AM modulated signal. This method also can be implemented on FPGAs since both modulator and demodulator partitions use the same clock. Figure 6 shows the block diagram of coherent detector.

*Simulation within Matlab / Simulink*

AM modulation can be implemented in several different ways within Simulink; two of these methods are demonstrated in Figure 7. The first method uses the DSB AM modulator blocks from Communication Blockset of Simulink.

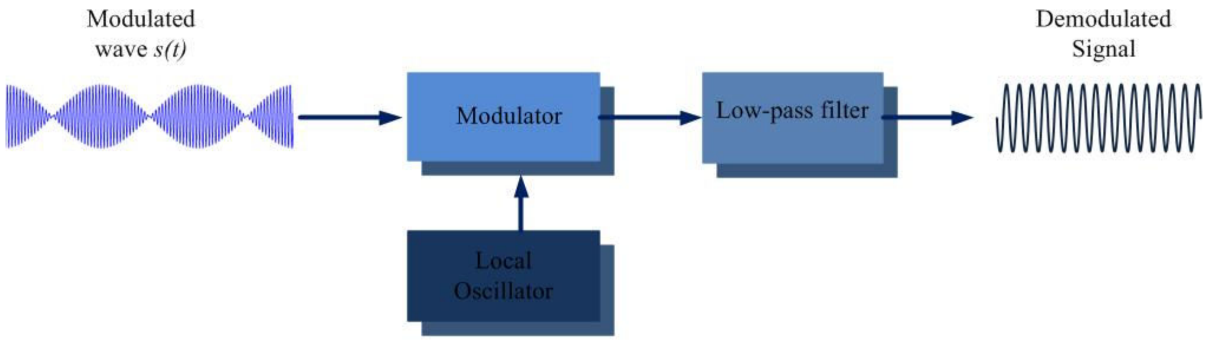


Figure 6. Block Diagram of Coherent Detector or Synchronous Demodulator

The other method leverages the use of product block to mix the message signal $m(t)$ with carrier signal $c(t)$. First, the envelope detector method simulated within Simulink which is shown in the lower section of Figure 7. Again, there are several methods that can be used to implement the envelope detection process. One of these methods is called DSB AM demodulator which is marked as Method 4 in this figure. The other methods are implemented by squaring the modulated signal and low-pass filtering [40] which is illustrated in Method 1s through 3. Among these modulation and demodulation methods, the Method 1 AM modulation and Method 3 AM demodulation methods can be transferred into System Generator and later implemented on an FPGA for educational applications since currently the similar blocks are exist in System Generator library.

Simulation results from Simulink are shown in Figure 8 and Figure 9. In Figure 8 the input sine signal is shown on the top, the modulation Method 1 is illustrated in the middle, and the modulation Method 2 is shown on the bottom.

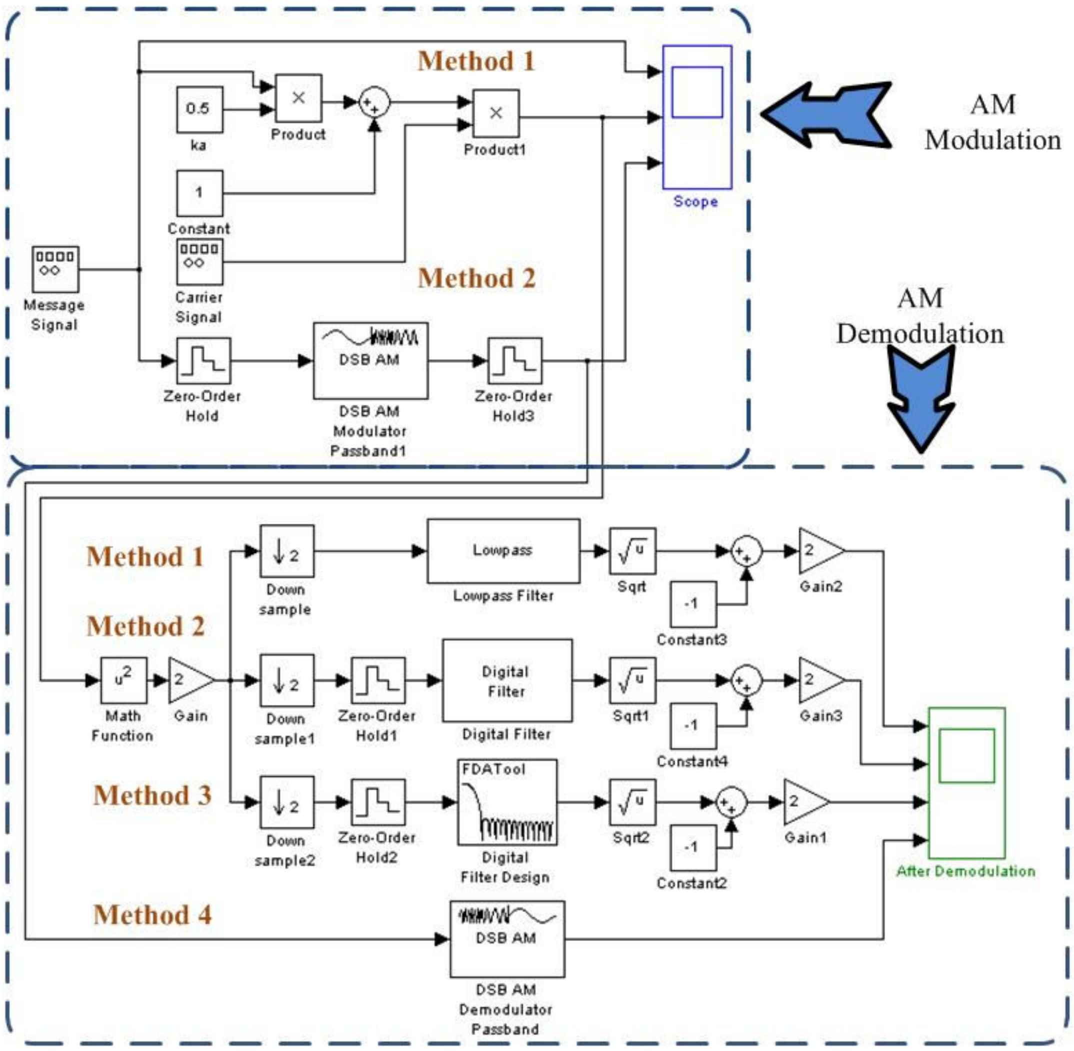


Figure 7. AM Modulation and Demodulation within Simulink

Figure 9 shows the demodulation result for Method 1s through 4. In addition, this figure shows that using each method will lead to a similar result.

Next, the selected modulation and demodulation methods analyzed with an input audio wav file by comparing differences between original audio, modulated, and demodulated signals.

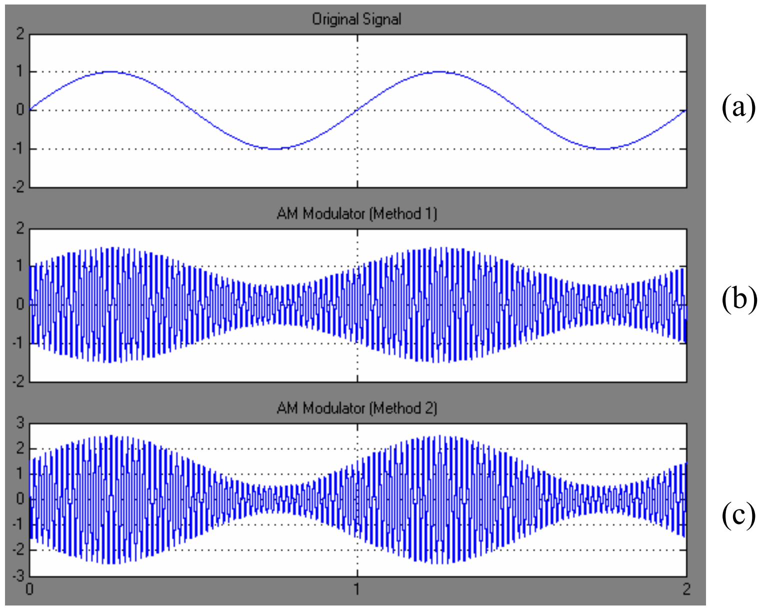


Figure 8. Simulation Results of AM Modulation in Simulink (a) Input Sine Signal (b) Method 1 Modulation (c) Method 2 Modulation

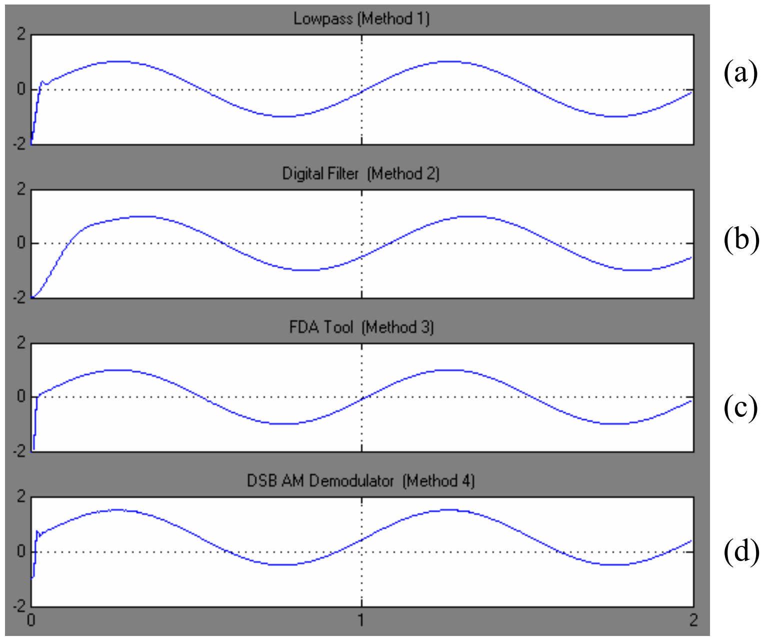


Figure 9. Simulation Results of AM Modulation in Simulink (a) Method 1 Demodulation (b) Method 2 Demodulation (c) Method 3 Demodulation (d) Method 1 Demodulation

The input audio file is mono format with sample rate of 22.05 kHz. Figure 10 illustrates the AM modulation and demodulation of this sample audio signal in Simulink.

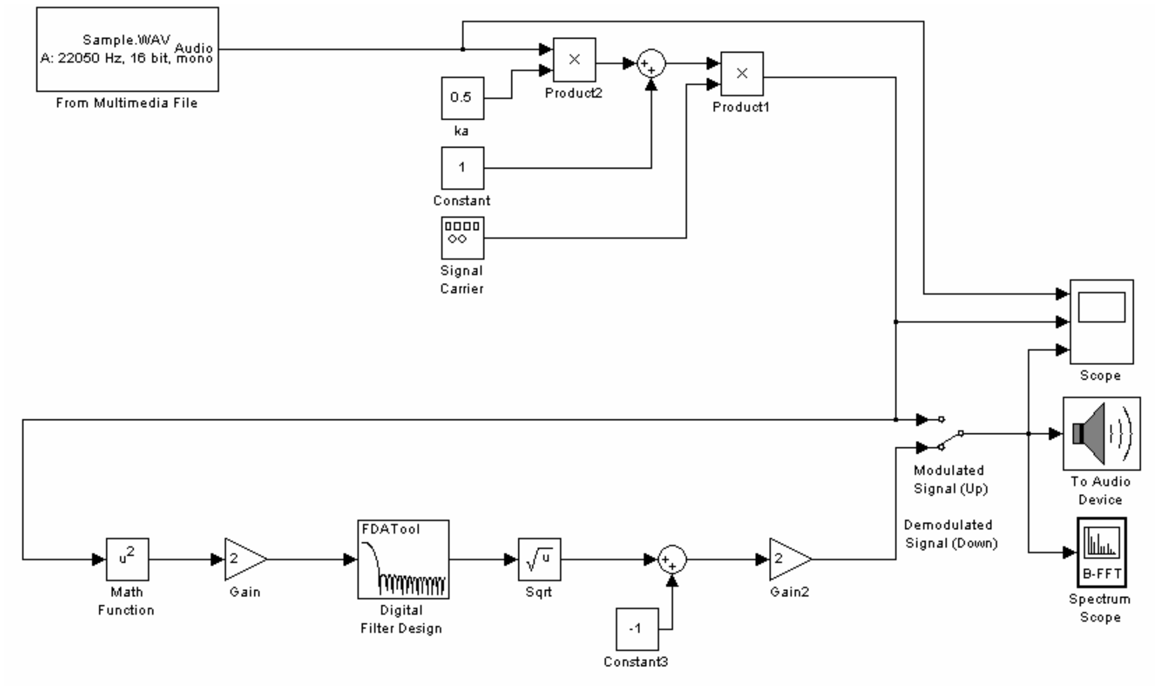
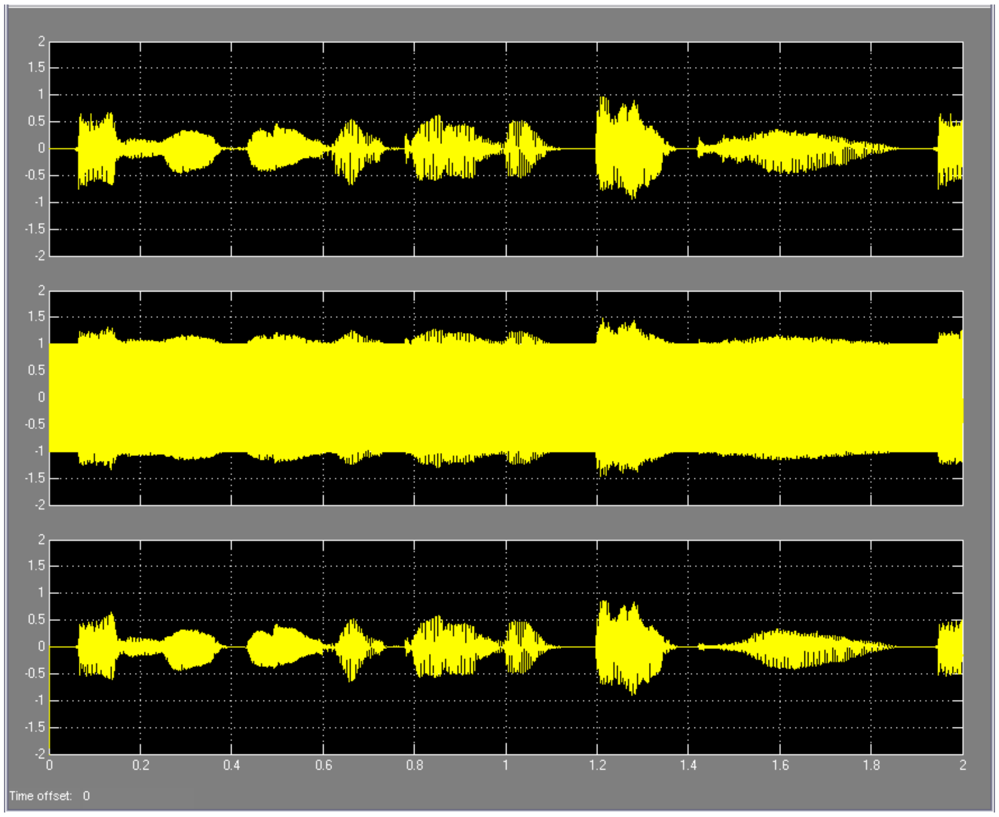


Figure 10. AM Modulation and Demodulation of an Audio Signal in Simulink

The demodulated audio signal compared with original audio signal by observing them on a scope and spectrum scope tools within Simulink. The results show the recovery of original signal after demodulation process. Figure 11 shows the comparison of a portion of this sample audio signal with its demodulated signal in a time domain. This figure also illustrates the spectrum of modulated signal. Figure 12 illustrates the frequency spectrum of this sample audio signal and its modulated signal.

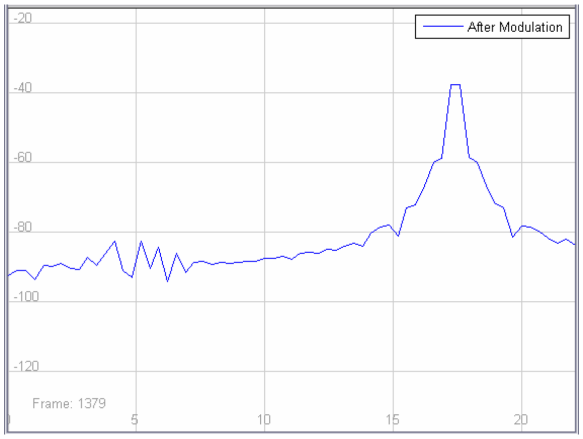


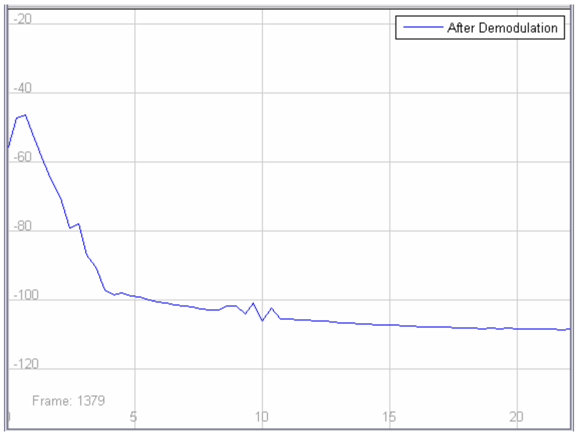
(a)

(b)

(c)

Figure 11. Simulation Results of AM Modulation of an Audio Signal in Simulink (a) Input Audio Signal (b) Audio Signal after Modulation (c) Recovered Audio Signal after Demodulation





(a) (b)

Figure 12. Frequency Spectrum of an Audio Signal after Modulation (a) and Demodulation (b) for AM Modulation

In this section, coherent detection was built and simulated in Simulink. Similarly, several methods for modulation and demodulation process were considered and simulated within Simulink. Figure 13 illustrates these DSB-SC modulations and demodulations within Simulink. In the same way, observed the simulation results of these blocks with sine wave inputs.

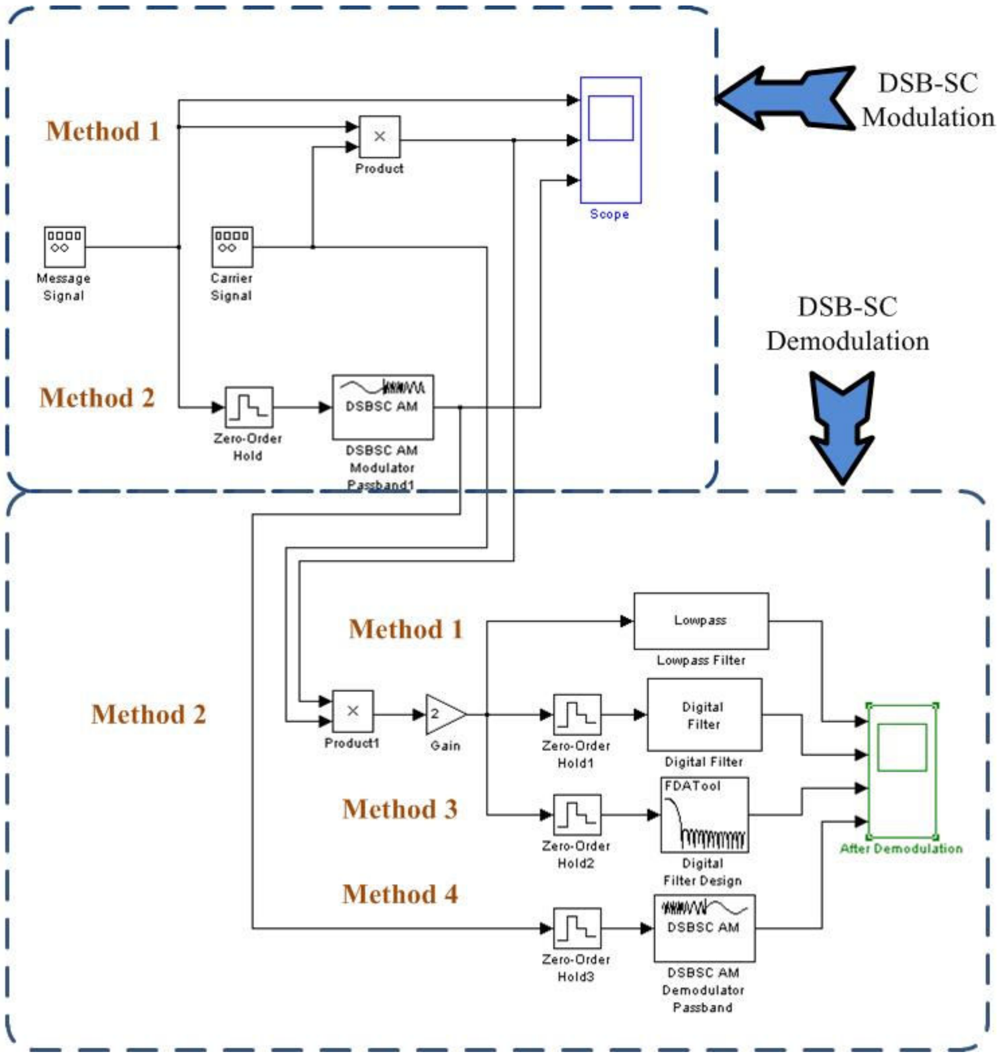


Figure 13. DSB-SC Modulation and Demodulation within Simulink

Figure 14 shows simulation results of input sine wave on the top, the DSB-SC modulation Method 1 in the middle, and the DSB-SC modulation Method 2 on the bottom. Figure 15 shows the demodulation result for the methods 1 through 4. In addition, this figure shows that using each method will lead to a similar result.

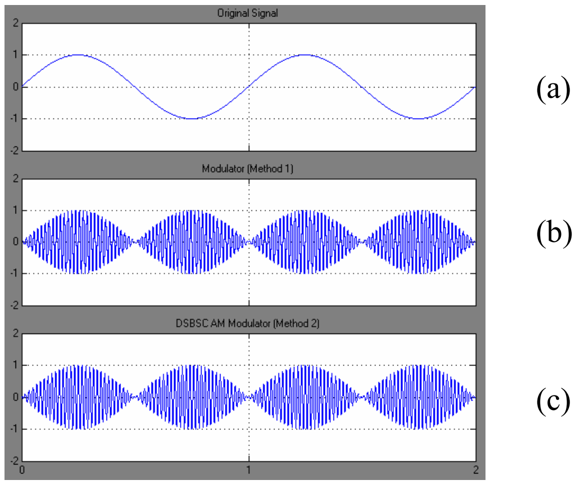


Figure 14. Simulation Results of DSB-SC Modulation in Simulink (a) Input Sine Signal (b) Method 1 Modulation (c) Method 2 Modulation

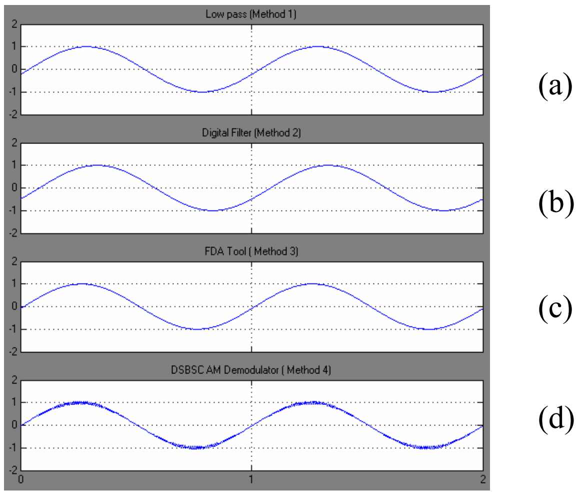


Figure 15. Simulation Results of DSB-SC Modulation in Simulink (a) Method 1 Demodulation (b) Method 2 Demodulation (c) Method 3 Demodulation (d) Method 4 Demodulation

Similarly, the selected modulation and demodulation methods analyzed with the same sample input audio signal by comparing differences between original audio, modulated, and demodulated signals. Figure 16 illustrates the representation of this DSB-SC modulation and demodulation in Simulink.

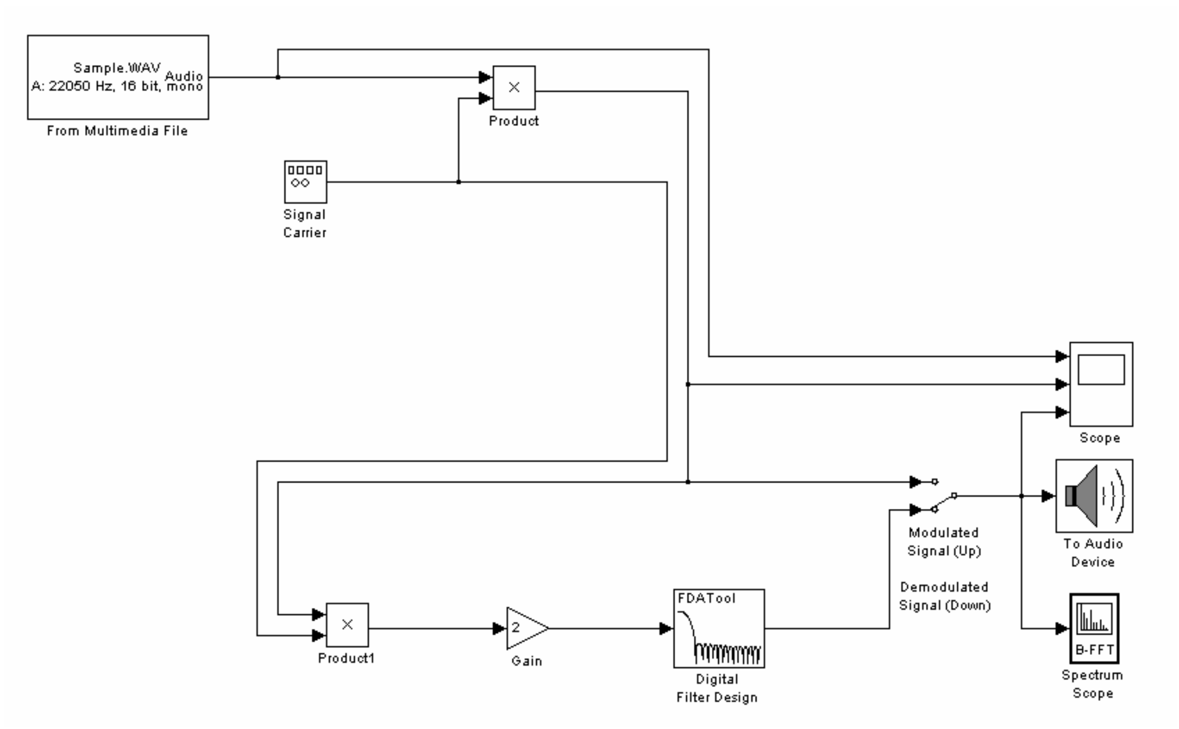


Figure 16. DSB-SC Modulation and Demodulation of an Audio Signal in Simulink

Figure 17 again illustrates the comparison of this sample audio signal with its demodulated signal in a time domain. The results show that original signal was recovered after demodulation process. Figure 18 illustrates the similar comparison of this sample audio signal with its modulated signal in frequency domain.

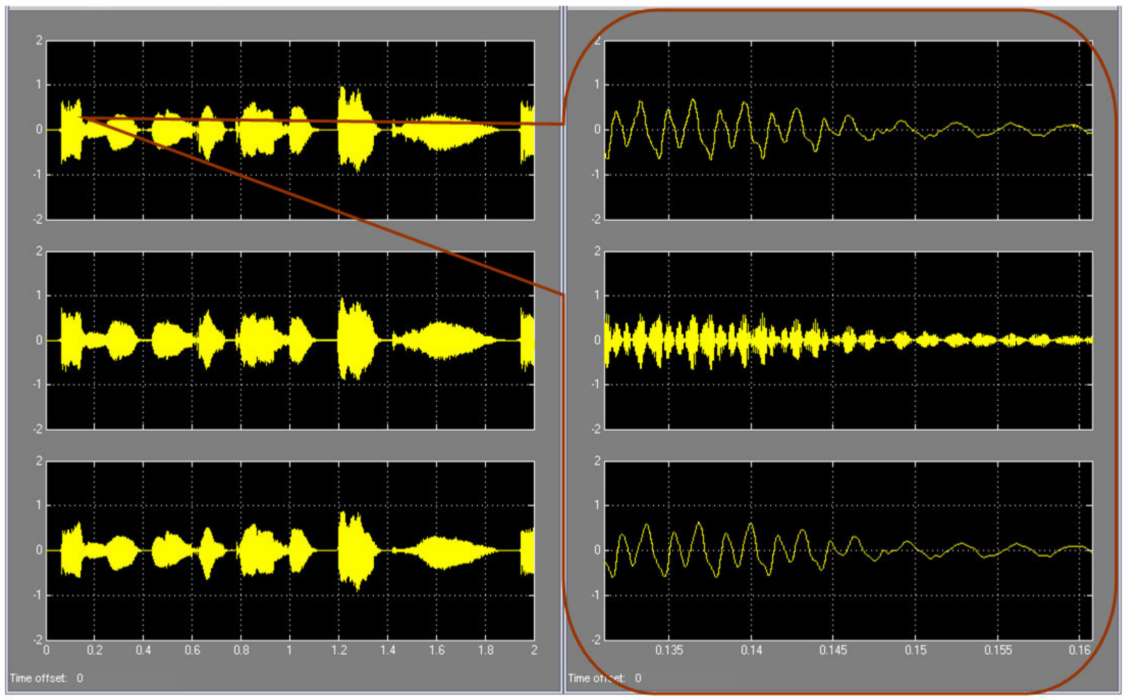
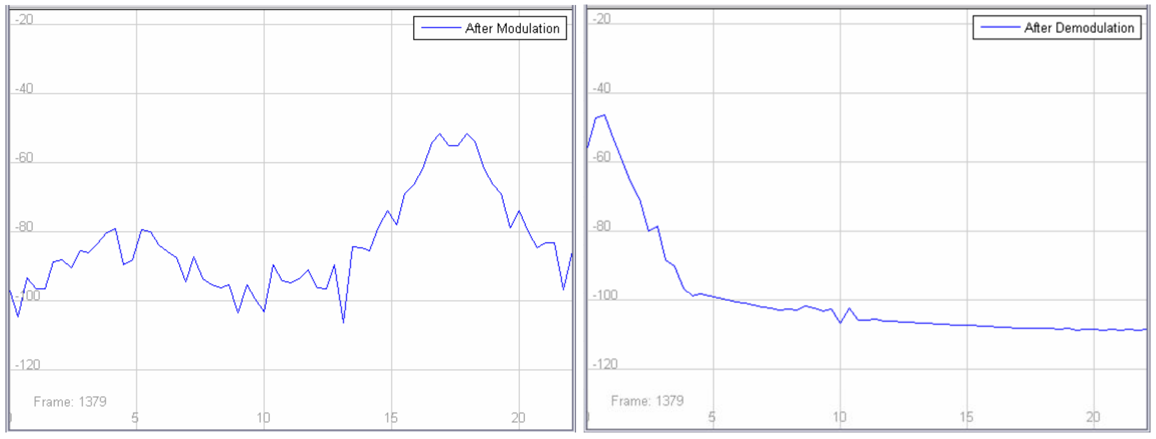


Figure 17. Simulation Results of DSB-SC Modulation of an Audio Signal in Simulink (a) Input Audio Signal (b) Audio Signal after Modulation (c) Recovered Audio Signal after Demodulation



(a) (b)

Figure 18. Frequency Spectrum of an Audio Signal after Modulation (a) and Demodulation (b) for DSB-SC

## *System Generator Design Process*

This section covers the System Generator in details. System Generator is a useful tool for designers with no back ground in Hardware Description Languages (HDLs). This capability allows the designer to reduce design times, which is a suitable feature for the educational applications. In this step the entire AM modulator using envelope detector is built in System Generator which is shown in Figure 19. This figure shows that System Generator uses two different types of blocks. Some of these blocks such as Scope are directly imported to System Generator from Simulink and other blocks provided by Xilinx.

In order to simulate the behavior of an FPGA the user should build each modulation scheme within In and Out Xilinx blocks. System Generator does not provide as many as blocks like Simulink; therefore, the designer must map his design from Simulink modules to System Generator modules. For example System Generator does not have Square block within its library; however, the user instead can use Mult Xilinx block to get the square of a signal. The other Xilinx block that is used in this figure is CORDIC SQRT which is similar to Sqrt block of Simulink which is shown in Figure 10.

Figure 20 shows System Generator simulation results of input sine wave on the top, the AM modulation in the middle, and the recovered signal after demodulation process on the bottom.

In the next step, the entire DSB-SC modulation is built in System Generator which is shown in Figure 21. This figure has a lot of similarities to Figure 16 which is showing the Simulink version of this modulation.

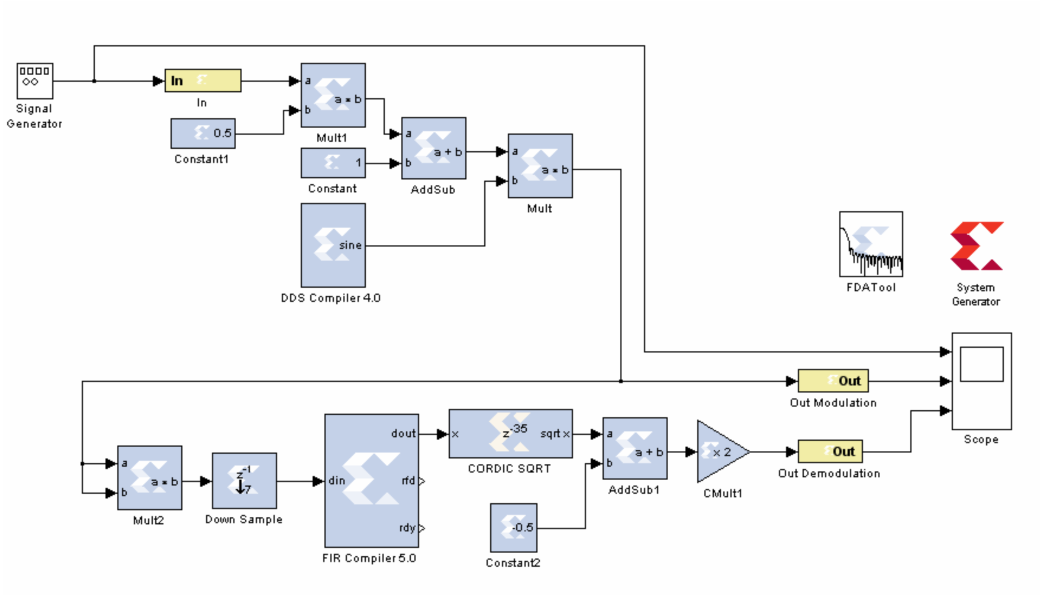


Figure 19. AM Modulation and Demodulation of a Sine Wave in System Generator

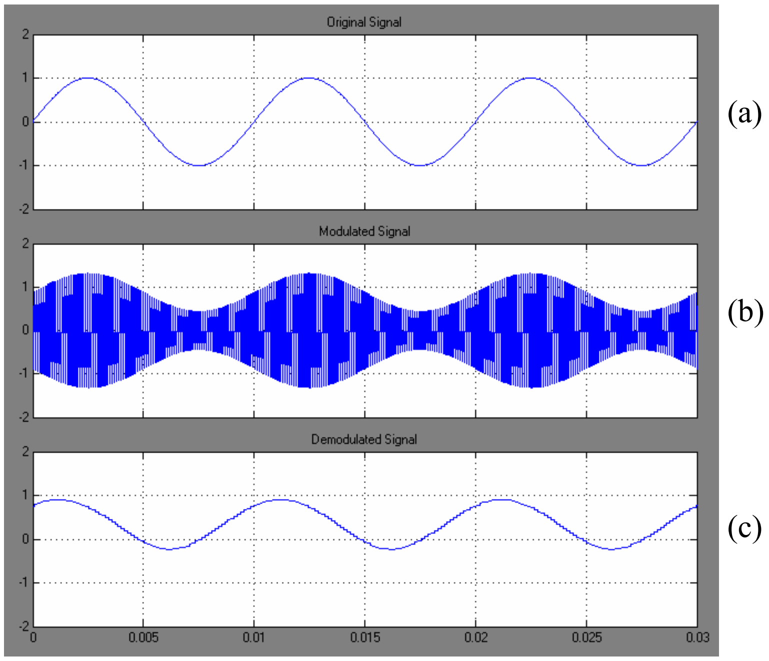


Figure 20. Simulation Results of AM Modulation in System Generator (a) Input Sine Signal (b) Signal after Modulation (c) Recovered Signal after Demodulation

In Figure 21 DDS Compiler 4.0 is used to generate the carrier signal. DDS stands for Direct Digital Synthesizer [41]. This block is generating digital sine/cosine wave within System Generator. This is the main difference between these figures. Figure 22 shows simulation results of input sine wave on the top, the DSB-SC modulation in the middle, and the recovered signal after demodulation process on the bottom for the System Generator.

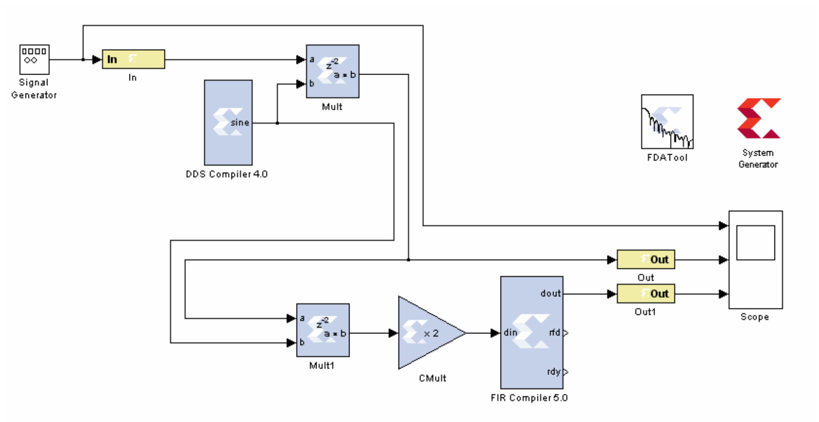
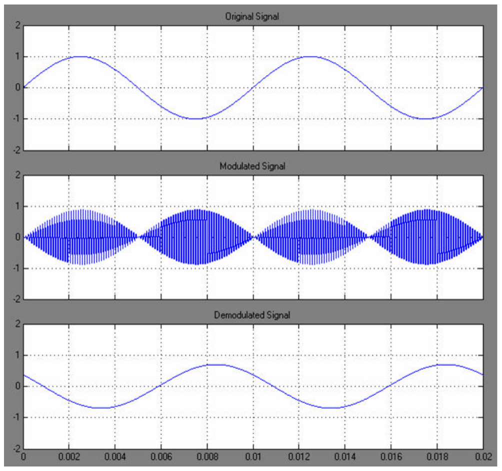


Figure 21. DSB-SC Modulation and Demodulation of a Sine Wave in System Generator



(a)

(b)

(c)

Figure 22. Simulation Results of DSB-SC Modulation in System Generator (a) Input Sine Signal (b) Signal after Modulation (c) Recovered Signal after Demodulation

In the next chapter, these modules are implemented on an FPGA and the output of the FPGA is tested and compared to the simulation results in System Generator and Simulink.

# Chapter IV

# RESULTS AND DISCUSSION

This chapter analyzes the performance of the implemented designs on an FPGA. In addition, it describes how these designs were enhanced by modifying them. This chapter shows that several design tools such as System Generator, and PlanAhead from Xilinx are required at the same time in order to complete entire PR implementation process. As mentioned earlier, System Generator is needed for designing and implementing DSP and communications designs. This chapter first shows the process of implementing partial reconfiguration through PlanAhead and finally validates the successful PR implementation on XUPV5 board for educational applications in communications classes.

In the previous chapters it was mentioned that Xilinx has several tutorials on partial reconfigurations [30]. However, none of these tutorials address using of partial reconfiguration for educational applications in communications classes. One of these tutorials describes the implementation of PR of an audio filter on FPGAs. During the process of PR implementation of a modulation and demodulation majority of the embedded system design and parts of the software components were borrowed from Xilinx tutorial. The software for this reconfigurable system was expanded from approximately 750 lines of code to about 1,800 lines of code. The original portions of the design are responsible for driving of audio chip on XUPV5 board and allowing the access to Compact Flash memory card through Internal Configuration Access Port (ICAP) port. ICAP allows writing software for an embedded processor (MicroBlaze) to modify the circuit structure during the circuit's operation [42]. Figure 23 summarizes the entire PR implementation process of this reconfigurable system for educational applications in communications classes. The upper portion of this figure illustrates the embedded system design process which is completed within EDK and SDK tools.

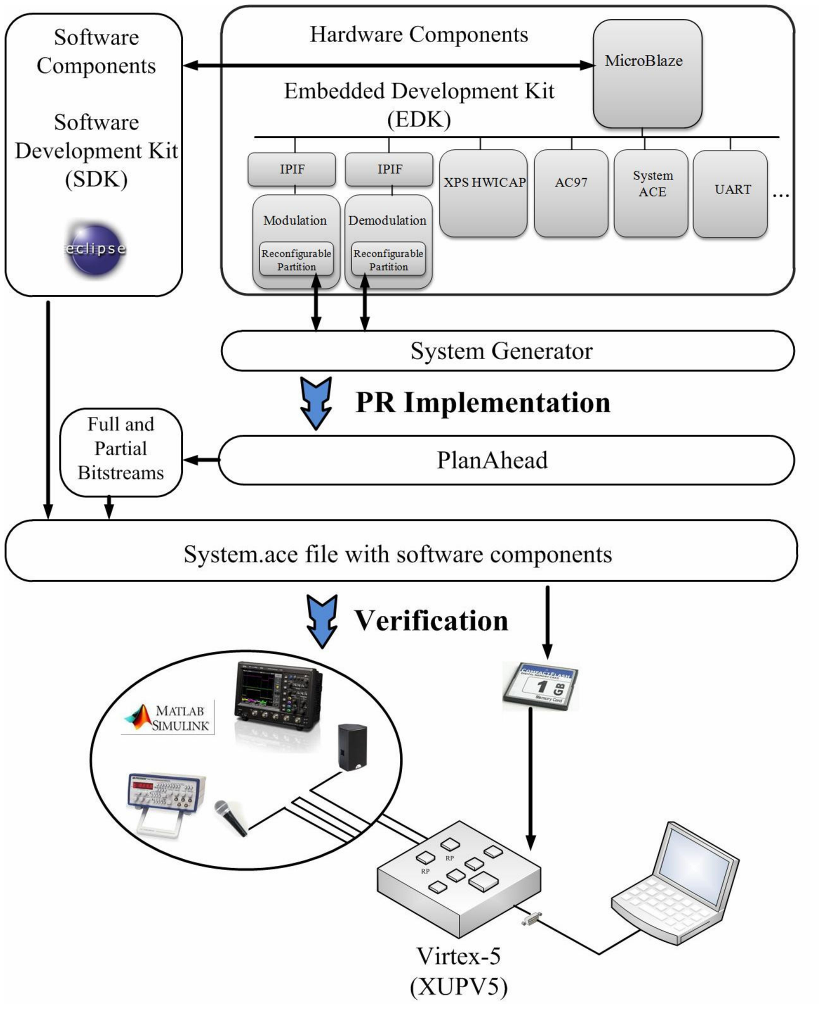


Figure 23. Representation of PR Implementation on Virtex-5 (XUPV5) Board

In addition, this figure shows that the netlist files for each reconfigurable partition (RP) are generated by System Generator then the entire PR implantation process is completed within PlanAhead. Also, after generating partial bitstreams and System ACE file, they are transferred into a Compact Flash memory card for implementation on the FPGA. System ACE stands for System Advanced Configuration Environment [43]. System ACE was developed by Xilinx to provide a cost saving alternative over traditional PROMs by using Compact Flash memory cards [43]. The verification process is done by observing that each partial bitstreams can swap on the FPGA without delaying the operation of the board. In addition, a function generator is used for transmitting signal through the system and observing the output signals through oscilloscope. In one part of this process, the Simulink tool is used for modulating the signal and FPGA is used to act as receiver by demodulating the signal. Next, the entire PR implementation process describes in the following sections.

### *Partial Reconfiguration Implementation Process*

The netlist files for each RP are generated within System Generator. In order to create a PR design within PlanAhead it is required to have partial reconfiguration license which was provided by Xilinx free of charge to universities for research and educational applications. The following steps are required to generate partial bitstreams and System ACE file for an FPGA [30]. This process is implemented for this reconfigurable system. The users only need to follow some of these steps for their educational applications, which are described in the next section.

(1) Setting-up PlanAhead

The first step is selecting PR option at the beginning of each PR project within PlanAhead. Next the specific FPGA on the board is selected which is xc5vlx110tff1136-1 for XUPV5 board.

(2) Selecting Top-level Netlist and Constraint file

In this step, the user should select the top netlist and constraint files of this design that was generated by embedded design tool.

(3) Defining Reconfigurable Partitions

After selecting the top netlist, the next step is defining reconfigurable partitions by right clicking on the instance and selecting Set Partition option.

(4) Adding Reconfigurable Modules

Next step is adding reconfigurable partitions for this design by right clicking on the partition and selecting Add Reconfigurable Module. Next, the user should select those netlist files that were generated by System Generator as the top level netlist for this reconfigurable module.

(5) Defining RP Regions

In this step, the RP regions are defined. This process is part of a floorplanning which refers to using AREA_GROUP constraints for controlling the design placement on an FPGA [44]. Using AREA_GROUP constraints allows the user to keep all associated logic with a partition concentrated in one area. This process will reduce the risk of routing conflict, improve runtime, and improve timing results [44]. In PlanAhead the AREA_GROUP constraints are defined by a Pblock. Figure 24 shows the process of defining the reconfigurable regions. In this figure, the Pblock size should be set under the Physical Constraints pane. After right clicking in this pane and selecting Set Pblock Size the user will be able to draw a rectangular shape for each RP.

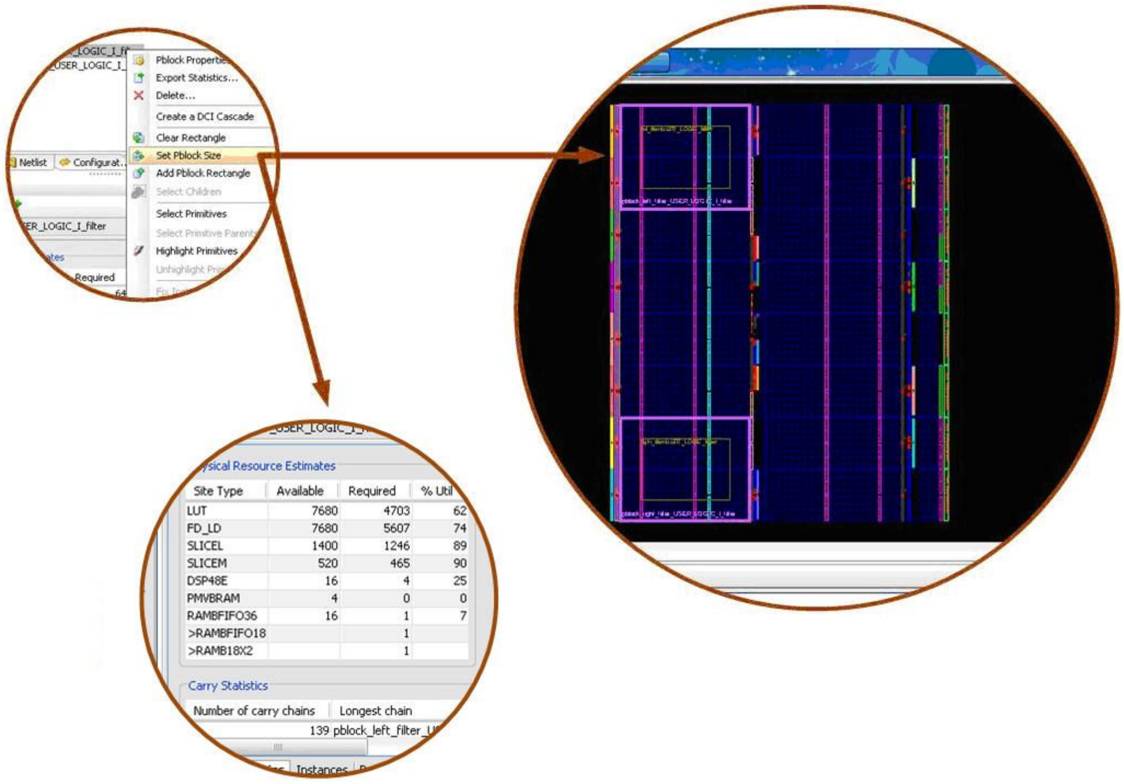
In addition, PlanAhead will provide estimate of required resources for Pblock s of each RP. The user should provide enough resources in this section by viewing physical resource estimates for each partition in order complete the PR implementation successfully. After saving the reconfigurable regions configuration will automatically will save within the constraints file (system.ucf). These configurations can be edited and updated within constraints file as well.

(6) Running Design Rule Checker

PlanAhead has option for catching the partial reconfiguration design issues at this stage of implementation which is called Design Rule Checker (DRC).

(7) Creating Configuration

In this step, first implementation configuration is created by setting a new synthesis strategy by using Xilinx Synthesis Technology (XST) tool. XST is part of ISE Design Suite allowing synthesis of an HDL design. At this point the user can assign a name to this configuration and select the appropriate partitions for it and then create it by right clicking this configuration in Design Runs tab and selecting the Launch Runs. After creating the first configuration (active configuration), this configuration should be promoted to set the static logic of the design for entire PR implementation process.



ysical Resource Estimates

| Site Type | Available | Required | % Util |
|---|---|---|---|
| LUT | 7680 | 4703 | 62 |
| FD_LD | 7680 | 5607 | 74 |
| SLICEL | 1400 | 1246 | 89 |
| SLICEM | 520 | 465 | 90 |
| DSP48E | 16 | 4 | 25 |
| PMVBRAM | 4 | 0 | 0 |
| RAMBFIFO36 | 16 | 1 | 7 |
| >RAMBFIFO18 | | 1 | |
| >RAMB18X2 | | 1 | |

Carry Statistics

| Number of carry chains | Longest chain |
|---|---|
| 139 | pblock_left_filter_U |

Figure 24. Defining the Reconfigurable Regions

(8) Creating Additional Configurations

In this step, after promoting the first configuration, the additional configurations can be generated by selecting Create Multiple Runs option under tools. In order to ensure that the static regions and all other configurations interfaces all consistent, the user should run the Verify Configuration option by right clicking on Configuration pane in PlanAhead.

(9) Verifying PR Implementation

In this step, after verifying the configurations, the bitstreams can be generated by right clicking on all of these configurations in Design Runs tab and selecting the Generate Bitsteam option.

(10) Generating Bitstream Files and Image Files

In this step, the image file is created and implemented on the FPGA. This PR design is using embedded soft (MicroBlaze) processor; therefore, the hardware bitstream file must be merged with the software components. The main core of the software components for this design is already created in the SDK. At this point, if there is need for updating the software components that can be finalized by lunching SDK tool within EDK and update the software components.

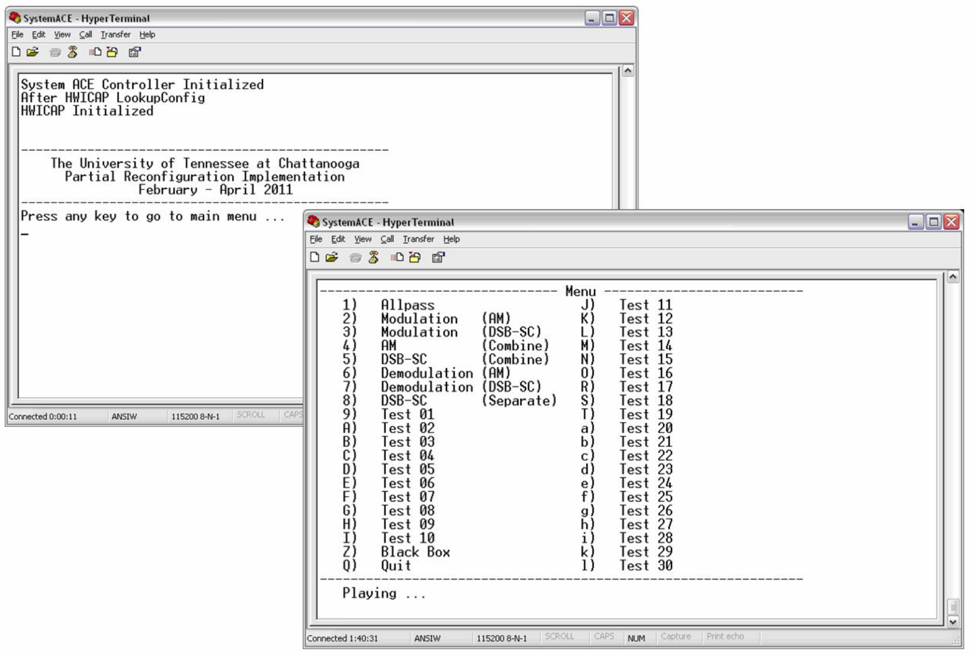
After updating the software components, these software components should merge with the hardware bitstream. This process cannot be implemented within GUI of PlanAhead and the user Xilinx bash shell will allow merging these components. The bash shell can be access from EDK or SDK tools.

Finally, the partial bitstreams and system.ace files are copied on a compact flash card for implantation on the XUPV5 board. In addition, if there is a need to add more configurations later, they can be added at any time after completing the main PR project.

In order to add additional configurations, the user should only follow the following steps. These steps are adding a new reconfigurable module, creating additional multiple runs, selecting the Lunch Runs, and finally creating partial bitstreams for the new configurations which makes this reconfigurable system a suitable system to use for educational or research applications in communications classes.

*Results of PR Implementation on the FPGA*

In this section, the implementation results will be analyzed. In this section the user can swap each partial bitstreams through HyperTerminal. If the PR implementation did not complete properly the terminal will freeze during exchanging the partial bitstreams. Figure 25 illustrates the process of changing partial bitstreams through HyperTerminal.


SystemACE - HyperTerminal
File Edit View Call Transfer Help
Connected 0:00:11 ANSIW 115200 8-N-1 SCROLL CAPS
SystemACE - HyperTerminal
File Edit View Call Transfer Help
Connected 1:40:31 ANSIW 115200 8-N-1 SCROLL CAPS NUM Capture Print echo

```
System ACE Controller Initialized
After HWICAP LookupConfig
HWICAP Initialized


--------------------------------------------------
    The University of Tennessee at Chattanooga
      Partial Reconfiguration Implementation
             February - April 2011
--------------------------------------------------
Press any key to go to main menu ...
_
```

```
------------------------------- Menu --------------------------
   1)   Allpass                     J)   Test 11
   2)   Modulation   (AM)           K)   Test 12
   3)   Modulation   (DSB-SC)       L)   Test 13
   4)   AM           (Combine)      M)   Test 14
   5)   DSB-SC       (Combine)      N)   Test 15
   6)   Demodulation (AM)           O)   Test 16
   7)   Demodulation (DSB-SC)       R)   Test 17
   8)   DSB-SC       (Separate)     S)   Test 18
   9)   Test 01                     T)   Test 19
   A)   Test 02                     a)   Test 20
   B)   Test 03                     b)   Test 21
   C)   Test 04                     c)   Test 22
   D)   Test 05                     d)   Test 23
   E)   Test 06                     e)   Test 24
   F)   Test 07                     f)   Test 25
   G)   Test 08                     g)   Test 26
   H)   Test 09                     h)   Test 27
   I)   Test 10                     i)   Test 28
   Z)   Black Box                   k)   Test 29
   Q)   Quit                        l)   Test 30
---------------------------------------------------------------
   Playing ...
```

Figure 25. Process of Changing Partial Bits through HyperTerminal

In the following steps, Simulink was used to generate an input sine signal that was transmitted through wire connection to the FPGA, and finally the output result was captured on an oscilloscope. This configuration is illustrated in Figure 26.

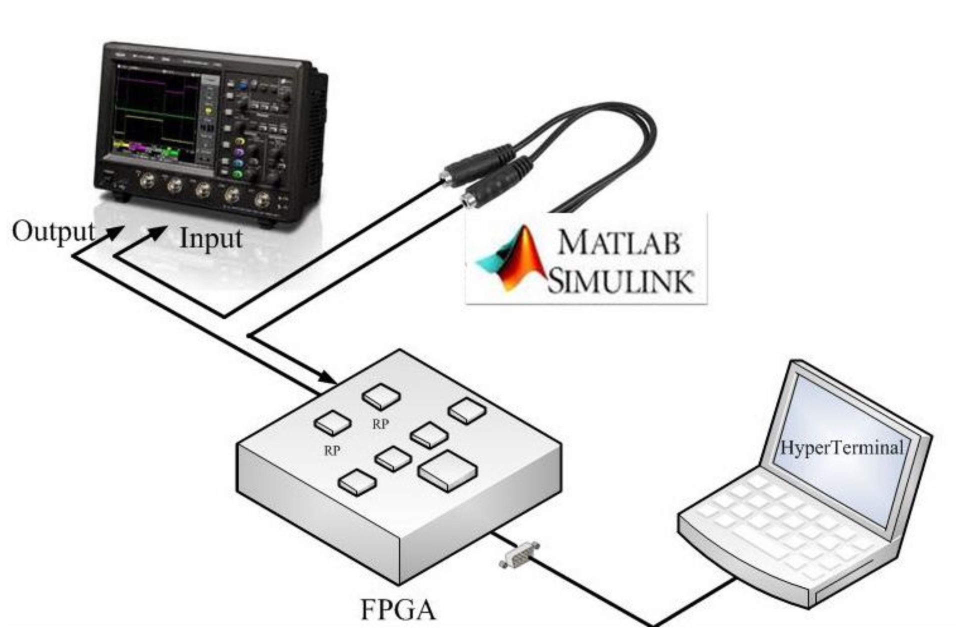


Figure 26. FPGA Input/Output Configurations

Next, DSB-SC modulation is implemented on the FPGA which is shown in Figure 27. The processing of implementing this modulation is conducted within System Generator by generating netlist file and then used the generated netlist file for generating partial bitstreams in PlanAhead. Figure 28 shows the input signal and the modulated signal DSB-SC modulation. In this figure, the signal in the upper portion of the figure represents the input sine wave which is generated from Simulink and lower signal shows the output signal of the FPGA. The carrier frequency is generated within System Generator by DDS block.

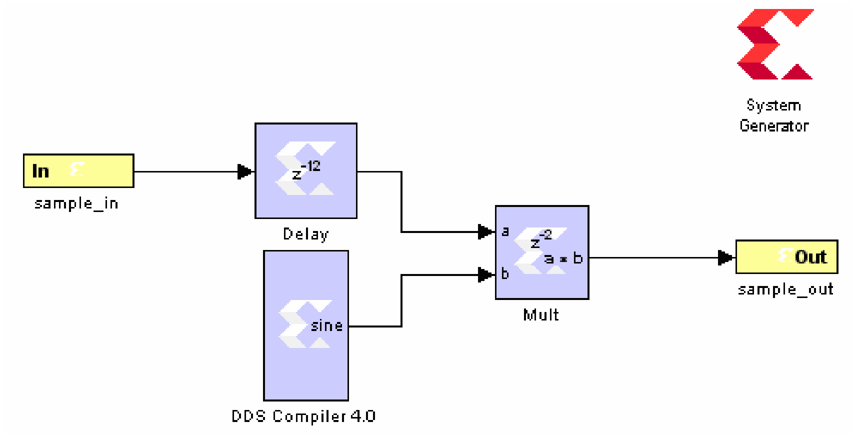


Figure 27. DSB-SC Modulation Process

In addition, this configuration was tested with an audio signal input, and the output signal was observed on the oscilloscope. Figure 29 shows the output results of a DSB-SC modulated of an audio signal on the FPGA. The results in this section are compared with the Simulink and the System Generator simulation results which show the similar results among all of them.

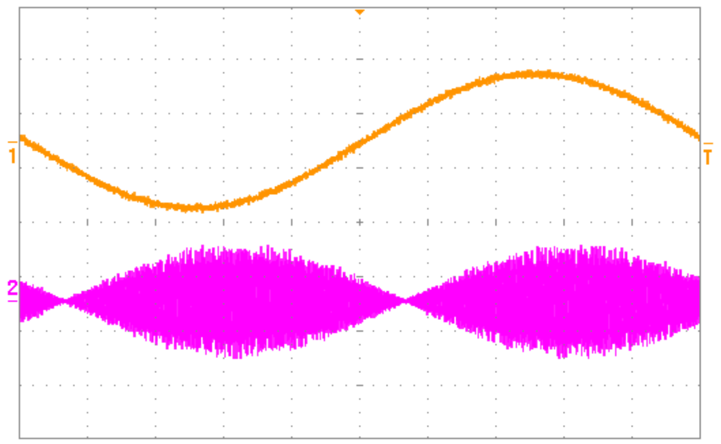


Figure 28. Input Signal and Output Results of Modulated Signal (DSB-SC Modulation) on the FPGA (1) Input Signal (2) Output Signal

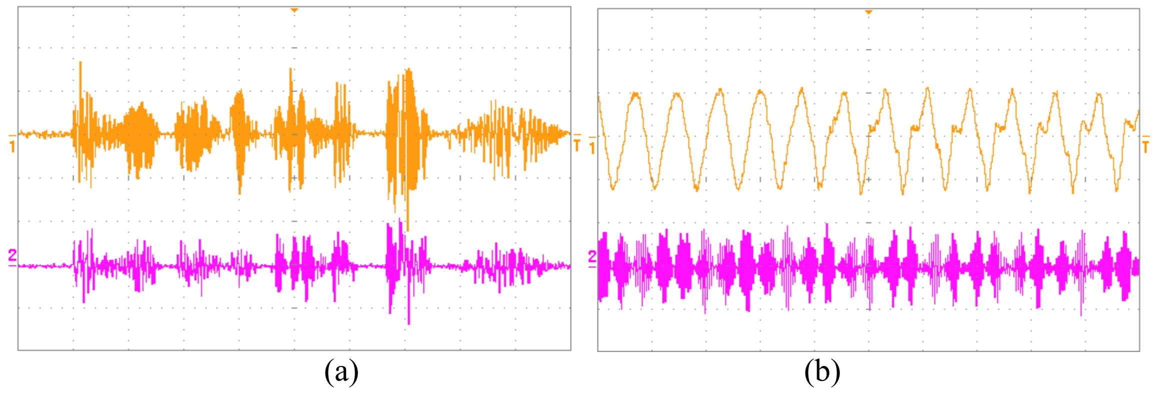


Figure 29. Output Results of Modulated Signal (DSB-SC Modulation) on the FPGA (a) 200ms Sec/Div (b) 5ms Sec/Div

Similarly, the next radio implemented is modulating an AM signal which is shown in Figure 30. In this figure, Constant1 block with value of 0.5 represents $k_a$ for AM modulation. The input signal and output result of this modulation was captured on the same oscilloscope.

Figure 31 illustrates the output result of this modulation process. Again, the signal in the upper portion of this figure represents the input sine signal which is generated from Simulink and lower signal shows the output modulated signal. The carrier frequency is generated within System Generator by DDS block. Moreover, this configuration was tested with an audio signal input and the output signal was captured on the oscilloscope which is shown in Figure 32.

The results of these measurements after implementation on an FPGA well matched to the simulation results of Simulink and System Generator in previous chapter which shows this reconfigurable system is suitable system for educational applications in communications classes.

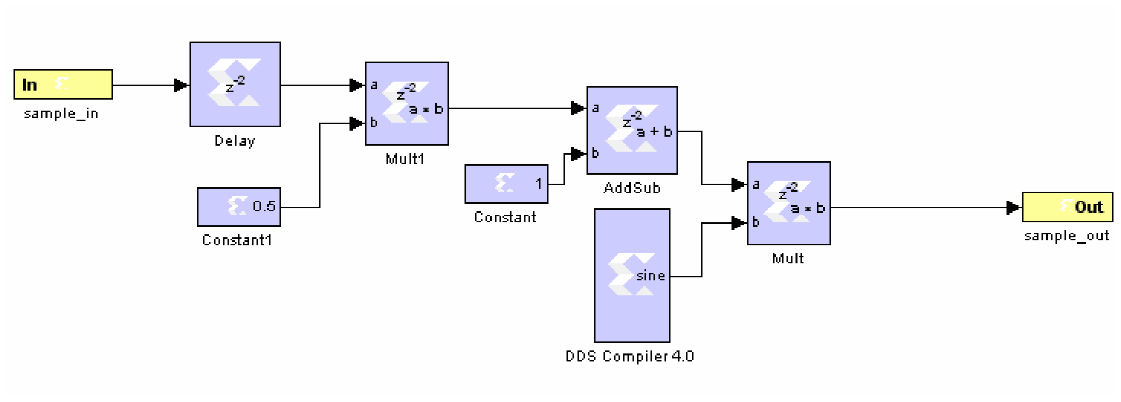


Figure 30. AM Modulation Block Diagram in System Generator

In the next configuration, the entire of DSB-SC modulation and demodulation is implemented within a same module which is shown in Figure 33. This figure shows that the carrier frequency is generated within System Generator by DDS block and then directly multiplied by the input signal (message signal) similar to Figure 13 and Figure 16.

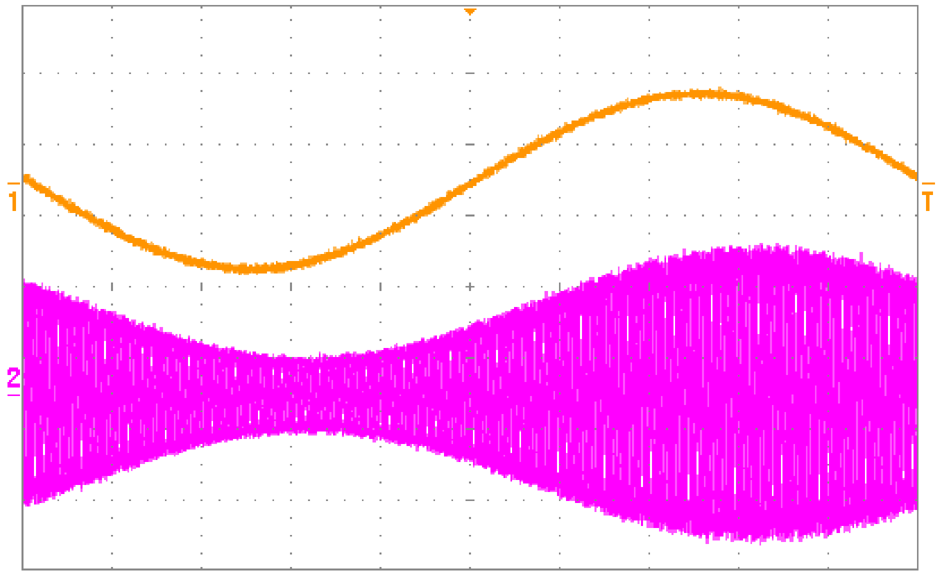


Figure 31. Output Results of Modulated Signal (AM Modulation) on the FPGA (1) Input Signal (2) Output Signal

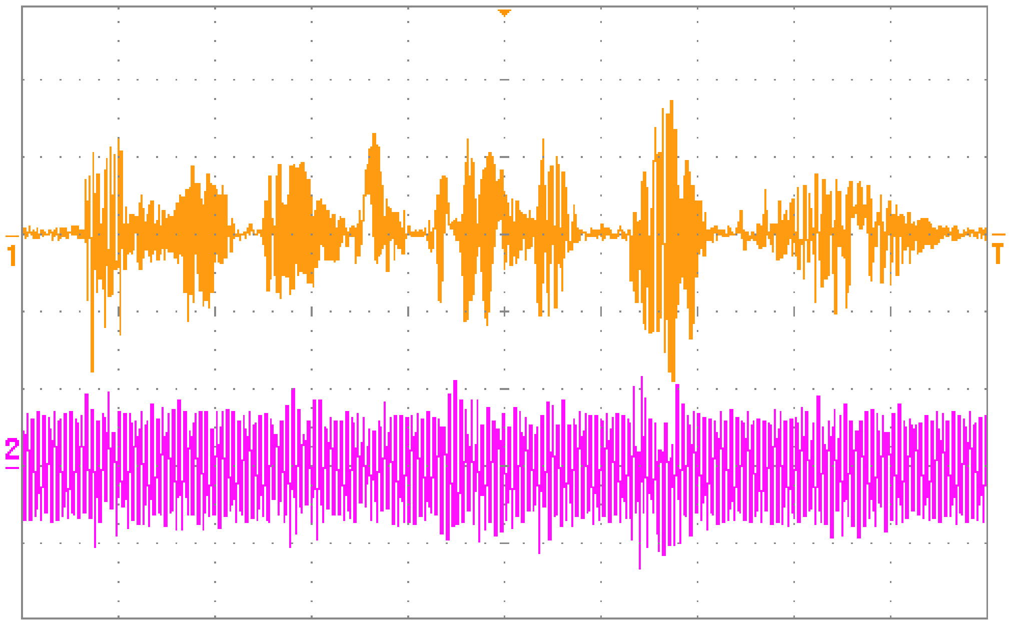


Figure 32. Output Results of Modulated Signal (AM Modulation) on the FPGA (1) Input Audio Signal (2) Output Audio Signal

Since the recovered signal has a lower gain compared with the input signal the CMult block is used in this design to amplify the output signal. CMult in System Generator is similar to Gain block in Simulink. In addition, the FIR Compiler 5.0 and Xilinx FDAtool are used to remove higher frequency components of the recovered signal. The output result of this modulation was captured and the result is shown in Figure 34. Again, the signal in the upper portion of this figure represents the input sine signal which is generated from Simulink and lower signal shows the recovered signal after demodulation process. Also, this configuration is tested with an audio signal input and the output signal is captured on the oscilloscope which is shown in Figure 35. The results of these measurements well matched to the simulation results of Simulink and System generator in previous chapter.

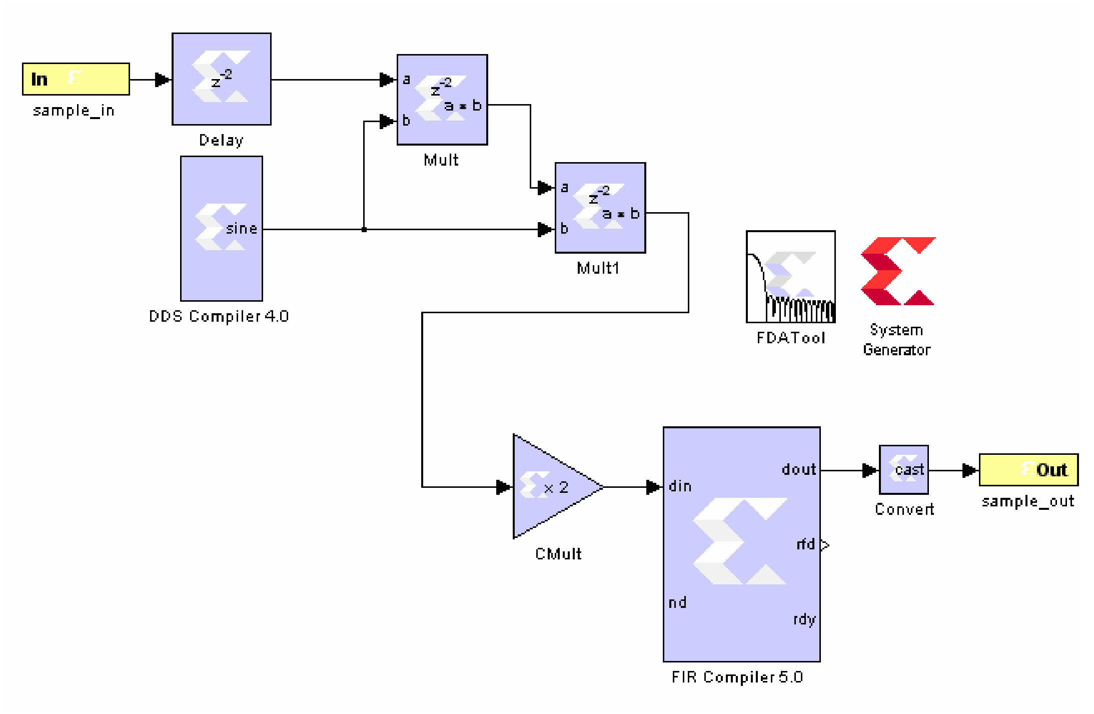
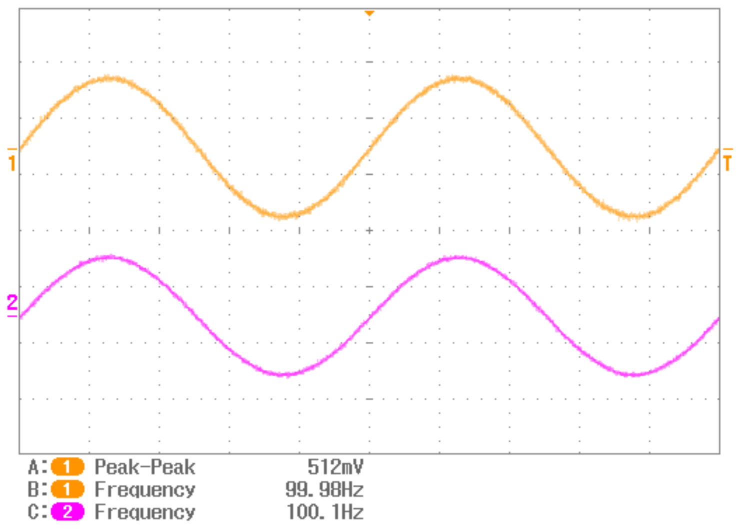


Figure 33. Entire DSB-SC Modulation and Demodulation on a Same Module



```
A: 1 Peak-Peak      512mV
B: 1 Frequency      99.98Hz
C: 2 Frequency      100.1Hz
```

Figure 34. Output Results of DSB-SC Modulation on the FPGA (1) Input Signal (2) Output Signal

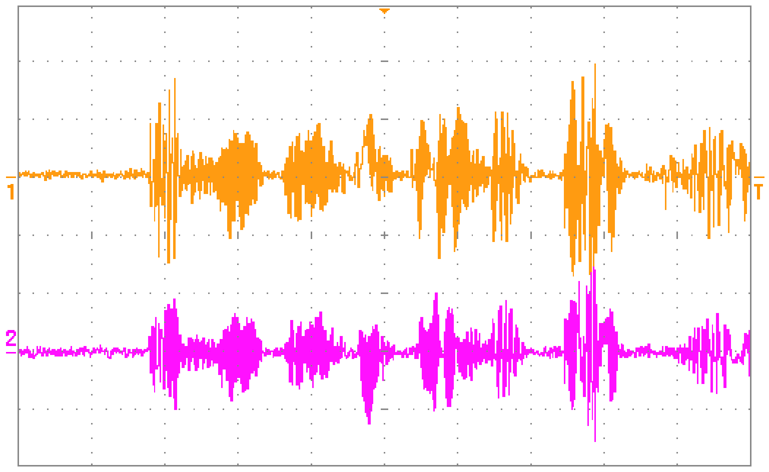


Figure 35. Output Results of DSB-SC Modulation on the FPGA (1) Input Audio Signal (2) Output Audio Signal

In the next configuration, entire AM modulation and demodulation is implemented within a same module which is shown in Figure 36.

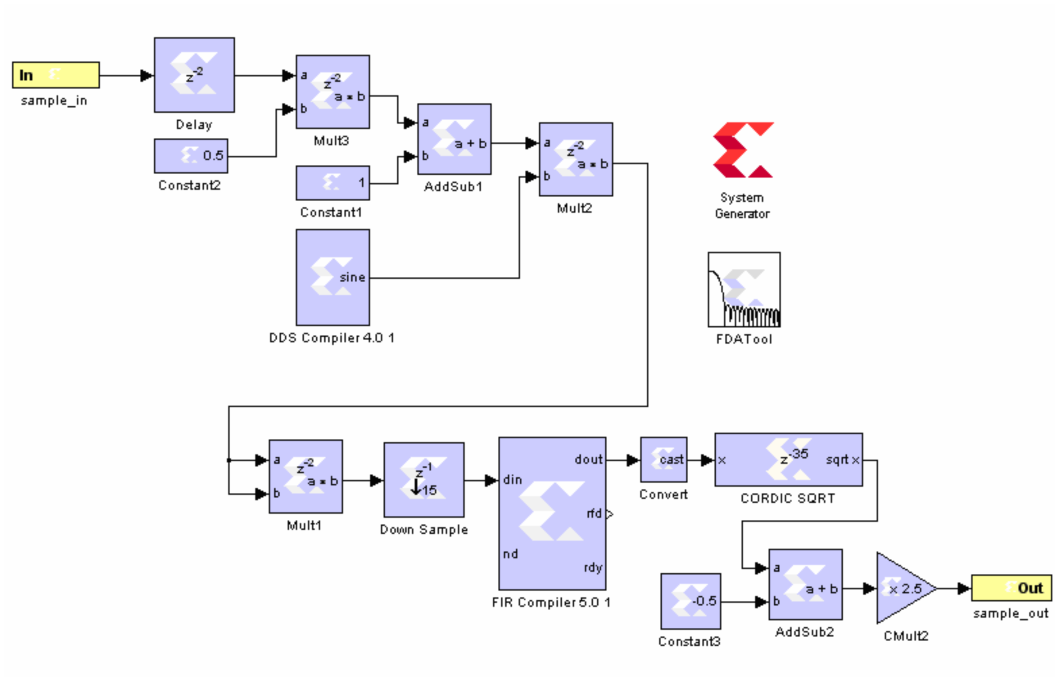


Figure 36. Entire AM Modulation and Demodulation on a Same RP

In this figure, the FIR Compiler 5.0 and Xilinx FDAtool are used to remove higher frequency components of the recovered signal. The output result of this modulation process is illustrated in Figure 37. Again, the signal in the upper portion of this figure represents the input sine signal which is generated from Simulink and lower signal shows the recovered signal after demodulation process. The carrier frequency is generated within System Generator. Also, this configuration was tested with an audio signal input and the output signal was captured on the oscilloscope which is shown in Figure 38. The results of these measurements are similar to the simulation results of Simulink and System generator in previous chapter.

In the following step, a DSB-SC modulated signal is generated within Simulink and transmitted to the FPGA through a wire connection. The FPGA now works as a DSB-SC demodulator (radio receiver). The implementation of this demodulation is illustrated in Figure 39. The output result of this demodulation process is shown in Figure 40.

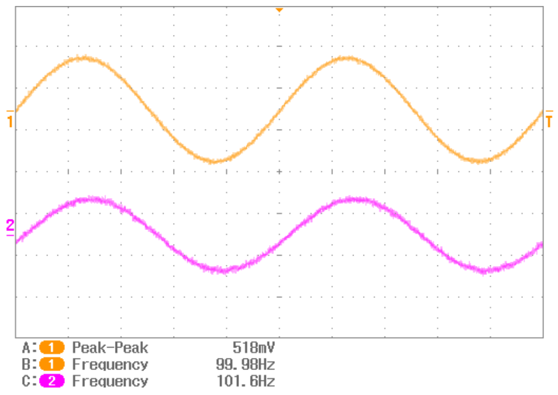


Figure 37. Output Result of AM Modulation on the FPGA (1) Input Signal (2) Output Signal

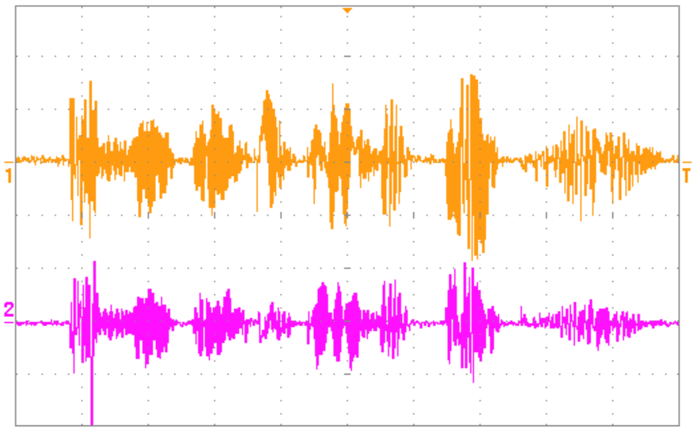


Figure 38. Output Result of AM Modulation on the FPGA (1) Input Signal (2) Output Signal

Since, the Simulink (Computer) clock frequency and FPGA clock are not well synchronized in this configuration the modulation and recovering of the signal did not occur at the exact desired frequency. In addition, this configuration was tested with a modulated audio signal input and the output signal was viewed on the oscilloscope which is illustrated in Figure 41.

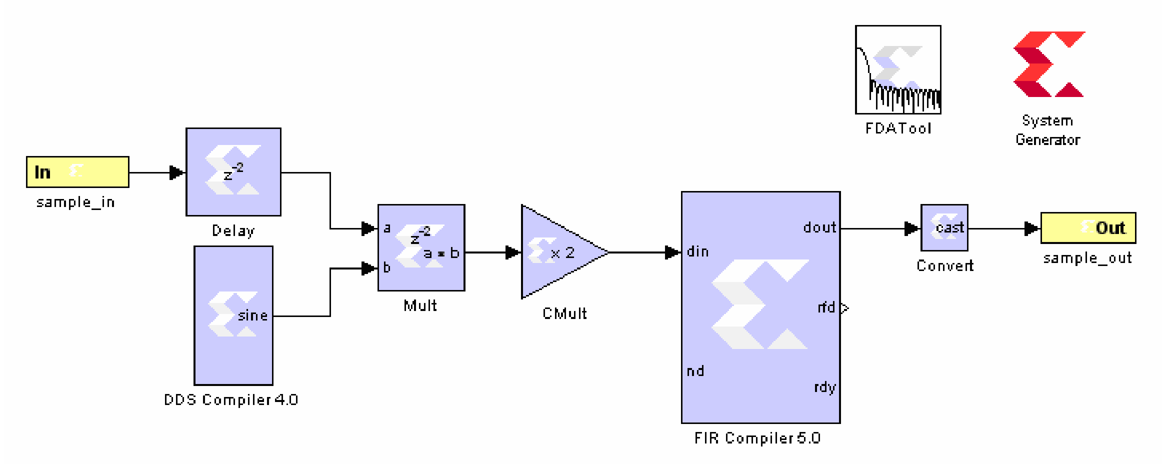


Figure 39. DSB-SC Demodulator (Radio Receiver) in System Generator

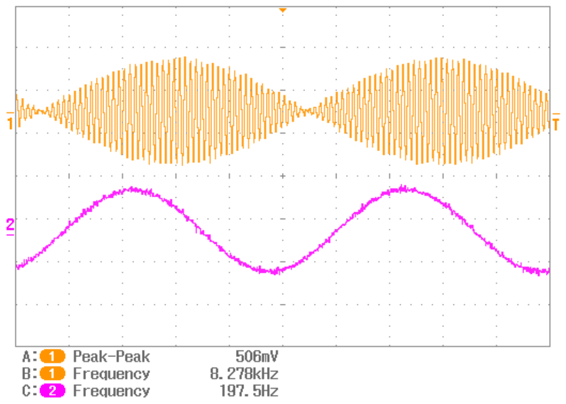


Figure 40. Output Result of DSB-SC Demodulation on the FPGA (1) Input Signal (2) Output Signal

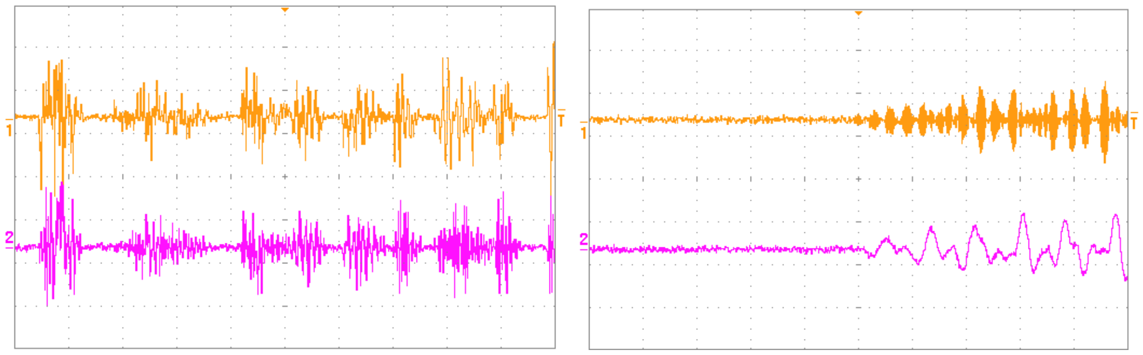


Figure 41. Output Result of DSB-SC Demodulation on the FPGA (1) Input Audio Signal (2) Output Audio Signal

# Chapter V

# CONCLUSIONS AND RECOMMENDATIONS

## *Conclusions*

The objective of this thesis was to design, simulate, and implement partial reconfigurable system for educational purposes on FPGAs. This objective was met by investigating and modeling several designs within Simulink simulation tool. These designs later were transferred and analyzed within System Generator, then implemented within a PR design and finally implemented on a FPGA. After comparing the simulations' results of several modulation schemes on Simulink and System Generator to the actual PR implementation on an FPGA, the results show that PR implementation was done properly. Furthermore, the PR implementation shows that the additional configurations can be design and implemented on an FPGA in few minutes rather than few hours compared with the entire design implementation process. This capability will make the PR implementation a useful tool for under graduate students in several of their classes such as communications, digital signal processing, digital design, etc.

## *Future Recommendations*

The current PR implementation does not take advantage of Tool Command Language (Tcl). The PR implementation can be automated with use of Tcl commands and reduce the time that is required to implement additional configurations. Also, due to lack of additional XUPV5 Virtex board at the University of Tennessee at Chattanooga, the performance of this partial reconfiguration was evaluated only on one XUPV5 Virtex board. For the future work, these designs can be implemented and verified using several FPGA boards. Another recommendation would be expanding these PR design in other areas with SDR systems such as wireless network design.

## REFERENCES

[1] M. Lanzagorta, S. Bique, and R. Rosenberg, *Introduction to Reconfigurable Supercomputing*. Morgan & Claypool Publishers, 2010.

[2] Xilinx, Inc., *http://www.xilinx.com/*. .

[3] Altera, Corp., *http://www.altera.com/*. .

[4] Lattice Semiconductor Corp., *http://www.latticesemi.com/*. .

[5] Actel Corp., *http://www.actel.com/*. .

[6] Avnet, Inc., *http://www.avnet.com/*. .

[7] Digilent Inc., *http://www.digilentinc.com/*. .

[8] D. Dye, *Partial Reconfiguration of Virtex FPGAs in ISE 12*. San Jose, CA: Xilinx, Inc., 2010.

[9] "OSSIE, SCA-Based Open Source Software Defined Radio," *http://ossie.wireless.vt.edu/*. .

[10] "Rice University WARP - Wireless Open-Access Research Platform," *http://warp.rice.edu/trac/*. .

[11] "BEE1," *http://bwrc.eecs.berkeley.edu/Research/BEE/BEE1/index.htm*. .

[12] "BEE2 - Berkeley Emulation Engine 2," *http://bee2.eecs.berkeley.edu/*. .

[13] "BEEcube Inc. - Products," *http://www.beecube.com/products/#*. .

[14] "TIMS Telecommunication, University, Modulation, Transmission, Wireless...," *http://www.tims.com.au/tims-main.taf?_UserReference=05E7E1CF034518C54D8F46E1*. .

[15] "About TIMS," *http://www.tims.com.au/tims-main.taf?do=page&page=about_tims&_UserReference=05E7E1CF034518C54D8F46E1*. .

[16] "Simulink - Simulation and Model-Based Design," *http://www.mathworks.com/products/simulink/*. .

[17] "System Generator for DSP," *http://www.xilinx.com/tools/sysgen.htm*. .

[18] I. Kuon, R. Tessier, and J. Rose, *FPGA Architecture*. Now Publishers Inc, 2008.

[19] B. Zeidman, *Designing with FPGAs and CPLDs*. Focal Press, 2002.

[20] P.-A. Hsiung, M. D. Santambrogio, and C.-H. Huang, *Reconfigurable System Design and Verification*, 1st ed. CRC Press, 2009.

[21] "POWER to the people," *http://www.ibm.com/developerworks/power/library/pa-powerppl/*, 15-Dec-2005. .

[22] "Xilinx : About Xilinx : Getting Started," *http://www.xilinx.com/company/gettingstarted/index.htm*. .

[23] "High-Performance FPGA Architecture," *http://www.altera.com/products/devices/stratix-fpgas/about/fpga-architecture/stx-architecture.html?GSA_pos=3&WT.oss_r=1&WT.oss=FPGA%20technology%20Logic%20Array%20Blocks*. .

[24] R. Woods and J. McAllister, *FPGA-based implementation of signal processing systems*. John Wiley and Sons, 2008.

[25] Xilinx, Inc., *Virtex-5 Family Overview, Product Specification*. San Jose, CA: Xilinx, Inc., 2009.

[26] A. Percey, *Advantages of the Virtex-5 FPGA 6-Input LUT Architecture*. San Jose, CA: Xilinx, Inc., 2007.

[27] "Genesys Virtex, 5 FPGA Development Kit," *http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,819&Prod=GENESYS*. .

[28] "Virtex-5 OpenSPARC Evaluation Platform," *http://www.digilentinc.com/Products/Detail.cfm?Prod=XUPV5&Nav1=Products&Nav2=Programmable*. .

[29] P. Sirisuk, *Reconfigurable Computing: Architectures, Tools and Applications: 6th International Symposium, ARC 2010, Bangkok, Thailand, March 17-19, 2010, Proceedings*. Springer, 2010.

[30] "Xilinx University Program," *http://www.xilinx.com/university/index.htm*. .

[31] "Xilinx: Downloads," *http://www.xilinx.com/support/download/index.htm*. .

[32] Xilinx, Inc., *PlanAhead User Guide*. San Jose, CA: Xilinx, Inc., 2010.

[33] Xilinx, Inc., *ISim User Guide*. San Jose, CA: Xilinx, Inc., 2011.

[34] Xilinx, Inc., *Embedded System Tools Reference Manual*. San Jose, CA: Xilinx, Inc., 2011.

[35] Xilinx, Inc., *EDK Concepts, Tools, and Techniques*. San Jose, CA: Xilinx, Inc., 2010.

[36] Xilinx, Inc., *System Generator for DSP User Guide*. San Jose, CA: Xilinx, Inc., 2010.

[37] A. V. Raisanen, *Radio Engineering for Wireless Communication and Sensor Applications*. Boston, MA: Artech House, 2003.

[38] S. Haykin and M. Moher, *An Introduction to Analog and Digital Communications*, 2nd ed. Wiley, 2006.

[39] S. Gibilisco, *The Illustrated Dictionary of Electronics*, 8th ed. New York: McGraw-Hill, 2001.

[40] "Signal Processing Blockset - Envelope Detection Demo," *http://www.mathworks.com/products/sigprocblockset/demos.html?file=/products/demos/shipping/dspblks/signalblksenvdet.html*. .

[41] "DDS Compiler," *http://www.xilinx.com/products/intellectual-property/DDS_Compiler.htm*. .

[42] Xilinx, Inc., *LogiCORE IP XPS HWICAP*. San Jose, CA: Xilinx, Inc., 2010.

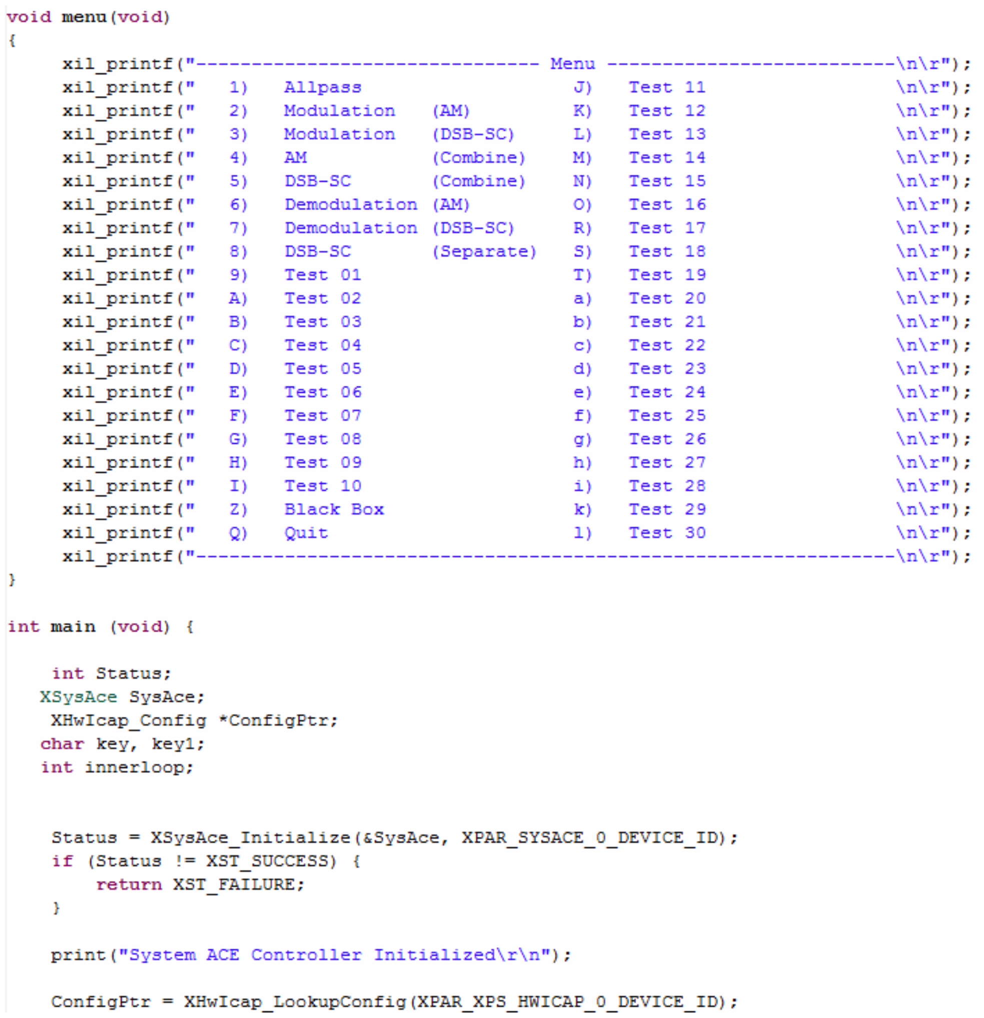
[43] Xilinx, Inc., *System ACE CompactFlash Solution*. San Jose, CA: Xilinx, Inc., 2008.

[44] Xilinx, Inc., *Hierarchical Design Methodology Guide*. San Jose, CA: Xilinx, Inc., 2010.

# APPENDIX A

# SOFTWARE COMPONENTS

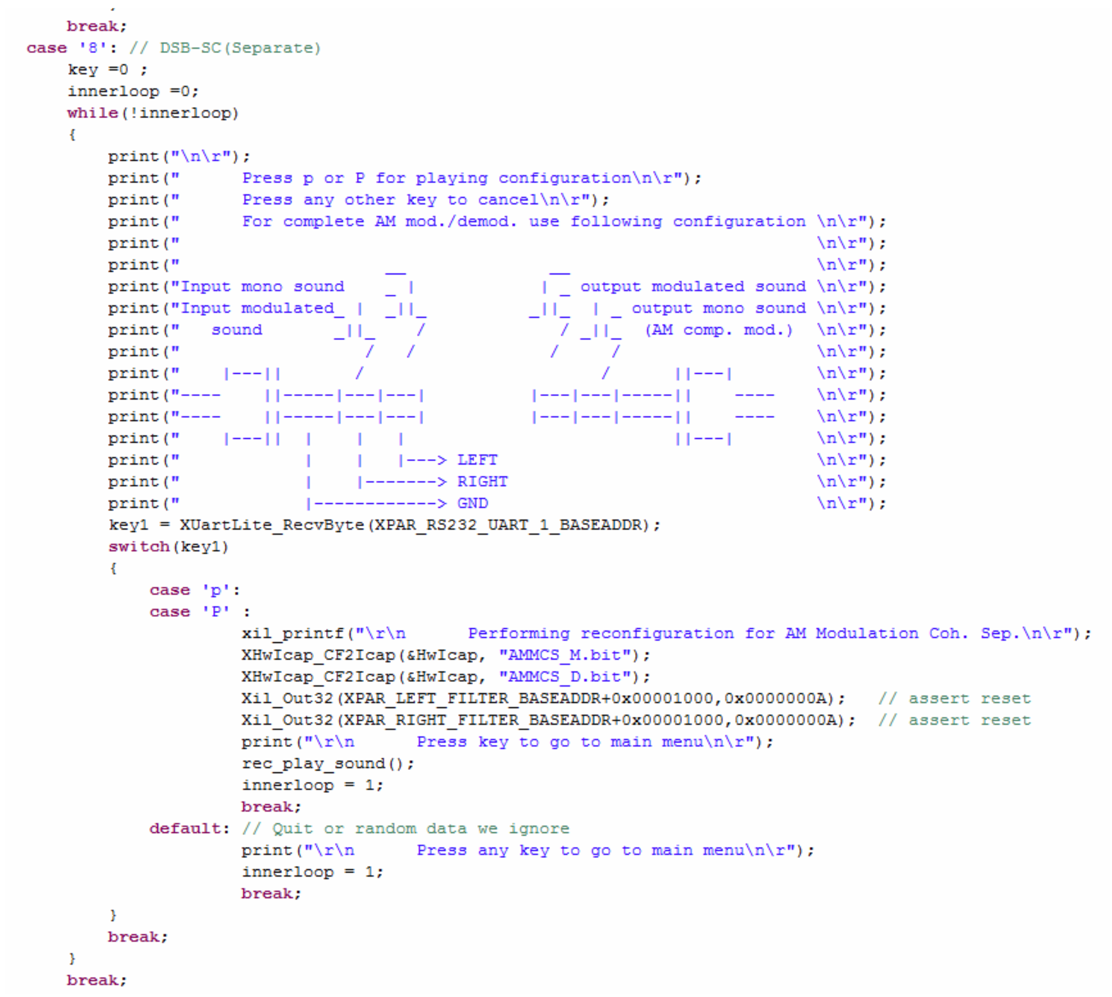
The main portion of the software components for this reconfigurable system was developed by the Xilinx. The software components are expanded from approximately 750 lines of code to about 1,800 lines of codes. The following section shows some part of this expansion.



```
void menu(void)
{
     xil_printf("------------------------------- Menu -------------------------\n\r");
     xil_printf("   1)   Allpass                      J)   Test 11               \n\r");
     xil_printf("   2)   Modulation    (AM)           K)   Test 12               \n\r");
     xil_printf("   3)   Modulation    (DSB-SC)       L)   Test 13               \n\r");
     xil_printf("   4)   AM            (Combine)      M)   Test 14               \n\r");
     xil_printf("   5)   DSB-SC        (Combine)      N)   Test 15               \n\r");
     xil_printf("   6)   Demodulation  (AM)           O)   Test 16               \n\r");
     xil_printf("   7)   Demodulation  (DSB-SC)       R)   Test 17               \n\r");
     xil_printf("   8)   DSB-SC        (Separate)     S)   Test 18               \n\r");
     xil_printf("   9)   Test 01                      T)   Test 19               \n\r");
     xil_printf("   A)   Test 02                      a)   Test 20               \n\r");
     xil_printf("   B)   Test 03                      b)   Test 21               \n\r");
     xil_printf("   C)   Test 04                      c)   Test 22               \n\r");
     xil_printf("   D)   Test 05                      d)   Test 23               \n\r");
     xil_printf("   E)   Test 06                      e)   Test 24               \n\r");
     xil_printf("   F)   Test 07                      f)   Test 25               \n\r");
     xil_printf("   G)   Test 08                      g)   Test 26               \n\r");
     xil_printf("   H)   Test 09                      h)   Test 27               \n\r");
     xil_printf("   I)   Test 10                      i)   Test 28               \n\r");
     xil_printf("   Z)   Black Box                    k)   Test 29               \n\r");
     xil_printf("   Q)   Quit                         l)   Test 30               \n\r");
     xil_printf("--------------------------------------------------------------\n\r");
}

int main (void) {

    int Status;
   XSysAce SysAce;
    XHwIcap_Config *ConfigPtr;
   char key, key1;
   int innerloop;


    Status = XSysAce_Initialize(&SysAce, XPAR_SYSACE_0_DEVICE_ID);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }

    print("System ACE Controller Initialized\r\n");

    ConfigPtr = XHwIcap_LookupConfig(XPAR_XPS_HWICAP_0_DEVICE_ID);
```

```
 if (ConfigPtr == NULL) {
     return XST_FAILURE;
 }
 print("After HWICAP LookupConfig\r\n");

 Status = XHwIcap_CfgInitialize(&HwIcap, ConfigPtr,
             ConfigPtr->BaseAddress);
 if (Status != XST_SUCCESS) {
     return XST_FAILURE;
 }
 print("HWICAP Initialized\r\n");

  ac97_init();
/* Print out title bar. */
 xil_printf("\r\n\r\n");
 xil_printf("--------------------------------------------------\r\n");
 xil_printf("    The University of Tennessee at Chattanooga    \r\n");
 xil_printf("      Partial Reconfiguration Implementation      \r\n");
 xil_printf("                February - April 2011             \r\n");
 xil_printf("--------------------------------------------------\r\n");

 print("Press any key to go to main menu ...\r\n");

 /* Wait for information from UART (keypress). */
key = XUartLite_RecvByte(XPAR_RS232_UART_1_BASEADDR);
menu();

/* Keep performing user requested function until quit (9) is pressed. */
while (key != 'Q') {
    switch (key) {
        case '1': // Allpass
            key =0 ;
            innerloop =0;
            while(!innerloop)
            {
                print("\n\r");
                print("      Press p or P for playing configuration\n\r");
                print("      Press any other key to cancel\n\r");
                key1 = XUartLite_RecvByte(XPAR_RS232_UART_1_BASEADDR);
                switch(key1)
                {
                    case 'p':
                    case 'P' :
                             xil_printf("\r\n      Performing reconfiguration for Allpass \n\r");
                             XHwIcap_CF2Icap(&HwIcap, "all_l.bit");
                             XHwIcap_CF2Icap(&HwIcap, "all_r.bit");
                             Xil_Out32(XPAR_LEFT_FILTER_BASEADDR+0x00001000,0x0000000A);  // assert reset
                             Xil_Out32(XPAR_RIGHT_FILTER_BASEADDR+0x00001000,0x0000000A);  // assert reset
                             print("\r\n      Press key to go to main menu\n\r");
                             rec_play_sound();
                             innerloop = 1;
                             break;
                    default: // Quit or random data we ignore
                             print("\r\n      Press any key to go to main menu\n\r");
                             innerloop = 1;
                             break;
                }
                break;
            }
            break;
            case '2': // Modulation (AM)
            key =0 ;
            innerloop =0;
            while(!innerloop)
            {
                print("\n\r");
                print("      Press p or P for playing configuration\n\r");
                print("      Press any other key to cancel\n\r");

                key1 = XUartLite_RecvByte(XPAR_RS232_UART_1_BASEADDR);
                switch(key1)
        {
            case 'p':
            case 'P' :
                 xil_printf("\r\n      Performing reconfiguration for Modulation (AM) \n\r");
                 XHwIcap_CF2Icap(&HwIcap, "amem_l.bit");
                 XHwIcap_CF2Icap(&HwIcap, "amem_r.bit");
                 Xil_Out32(XPAR_LEFT_FILTER_BASEADDR+0x00001000,0x0000000A);  // assert reset
                 Xil_Out32(XPAR_RIGHT_FILTER_BASEADDR+0x00001000,0x0000000A);  // assert reset
                 print("\r\n      Press key to go to main menu\n\r");
                 rec_play_sound();
                 innerloop = 1;
                 break;
            default: // Quit or random data we ignore
                     print("\r\n      Press any key to go to main menu\n\r");
                     innerloop = 1;
                     break;
        }
        break;
    }
    break;
    case '3': // Modulation (DSB-SC)
    key =0 ;
    innerloop =0;
    while(!innerloop)
    {
        print("\n\r");
        print("      Press p or P for playing configuration\n\r");
        print("      Press any other key to cancel\n\r");

        key1 = XUartLite_RecvByte(XPAR_RS232_UART_1_BASEADDR);
        switch(key1)
        {
            case 'p':
            case 'P' :
                     xil_printf("\r\n      Performing reconfiguration for Modulation (DSB-SC) \n\r");
                     XHwIcap_CF2Icap(&HwIcap, "amcm_l.bit");
                     XHwIcap_CF2Icap(&HwIcap, "amcm_l.bit");
                     Xil_Out32(XPAR_LEFT_FILTER_BASEADDR+0x00001000,0x0000000A);  // assert reset
                     Xil_Out32(XPAR_RIGHT_FILTER_BASEADDR+0x00001000,0x0000000A);  // assert reset
                     print("\r\n      Press key to go to main menu\n\r");
                     rec_play_sound();
                     innerloop = 1;
                     break;
            default: // Quit or random data we ignore
                     print("\r\n      Press any key to go to main menu\n\r");
                     innerloop = 1;
                     break;
        }
        break;
    }
    break;
case '4': // AM (Combine)
    key =0 ;
    innerloop =0;
    while(!innerloop)
    {
        print("\n\r");
        print("      Press p or P for playing configuration\n\r");
        print("      Press any other key to cancel\n\r");
```

Furthermore, the following section shows the portion of the code that is written for operating DSB-SC within separate modules.



```
    break;
case '8': // DSB-SC(Separate)
    key =0 ;
    innerloop =0;
    while(!innerloop)
    {
        print("\n\r");
        print("      Press p or P for playing configuration\n\r");
        print("      Press any other key to cancel\n\r");
        print("      For complete AM mod./demod. use following configuration \n\r");
        print("                                                               \n\r");
        print("                    __              __                         \n\r");
        print("Input mono sound    _ |            | _ output modulated sound  \n\r");
        print("Input modulated_ |  _||_          _||_   | _ output mono sound \n\r");
        print("   sound       _||_    /             / _||_   (AM comp. mod.)  \n\r");
        print("                  /   /             /     /                    \n\r");
        print("    |---||       /                       /      ||---|         \n\r");
        print("----    ||-----|---|---|          |---|---|-----||     ----    \n\r");
        print("----    ||-----|---|---|          |---|---|-----||     ----    \n\r");
        print("    |---||  |    |   |                          ||---|         \n\r");
        print("            |    |   |---> LEFT                                \n\r");
        print("            |    |-------> RIGHT                               \n\r");
        print("            |-------------> GND                                \n\r");
        key1 = XUartLite_RecvByte(XPAR_RS232_UART_1_BASEADDR);
        switch(key1)
        {
            case 'p':
            case 'P' :
                     xil_printf("\r\n      Performing reconfiguration for AM Modulation Coh. Sep.\n\r");
                     XHwIcap_CF2Icap(&HwIcap, "AMMCS_M.bit");
                     XHwIcap_CF2Icap(&HwIcap, "AMMCS_D.bit");
                     Xil_Out32(XPAR_LEFT_FILTER_BASEADDR+0x00001000,0x0000000A);    // assert reset
                     Xil_Out32(XPAR_RIGHT_FILTER_BASEADDR+0x00001000,0x0000000A);   // assert reset
                     print("\r\n      Press key to go to main menu\n\r");
                     rec_play_sound();
                     innerloop = 1;
                     break;
            default: // Quit or random data we ignore
                     print("\r\n      Press any key to go to main menu\n\r");
                     innerloop = 1;
                     break;
        }
        break;
    }
    break;
```