“Analysis and Implementation of Line Codes on Different Platforms”

By

Abhishek Kumar Malviya (111080)

Hemant Sharma (111083)

Under the supervision of

Prof. Tapan Jain

May-2015

Dissertation submitted in partial fulfillment

Of the requirement for the degree of

BACHELOR OF TECHNOLGY

IN

ELECTRONICS & COMMUNICATION ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,

WAKNAGHAT, SOLAN- 173234, INDIA

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

(Established under the Act 14 of Legislative Assembly of Himachal Pradesh)

Waknaghat, P.O. DomeharBani. Teh. Kandaghat, Distt. Solan- 173234(H.P.)

Phone: 01792-245367, 245368, 245369

Fax- 01792-245362

DECLARATION

We hereby declare that the work reported in the B. Tech report entitled "**Analysis and Implementation of Line Codes on Different Platforms"** submitted by "**Abhishek Kumar Malviya and Hemant Sharma"** at Jaypee University of Information Technology, Waknaghat is an authentic record of our work carried out under the supervision of **Prof. Tapan Jain.** This work has not been submitted partially or wholly to any other university or institution for the award of this or any other degree or diploma.

Abhishek Kumar Malviya

Hemant Sharma

Department of Electronics and Communication Engineering
Jaypee University of Information Technology (JUIT)
Waknaghat, Solan- 173234, India

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

(Established under the Act 14 of Legislative Assembly of Himachal Pradesh)

Waknaghat, P.O. DomeharBani. Teh. Kandaghat, Distt. Solan- 173234(H.P.)

Phone: 01792-245367, 245368, 245369

Fax- 01792-245362

CERTIFICATE

This is to certify that the work titled "**Analysis and Implementation of Line Codes on Different Platforms**" submitted by "**Abhishek Kumar Malviya and Hemant Sharma**" in the partial fulfilment of the degree of Bachelor of Technology (ECE) of Jaypee University of Information Technology, Waknaghat has been carried out under my supervision. This work has not yet been submitted partially or wholly to any other university or institution for the award of this or any other degree or diploma.

Prof. Tapan Jain

(Assistant Professor)

Department Of Electronics and Communication Engineering

Jaypee University of Information Technology (JUIT)

Waknaghat, Solan- 173234, India

(Supervisor)

ACKNOWLEDGEMENT

It is divine, grace and blessing of God that today we have successfully reached yet another milestone of our journey in this endless path of learning that has just begun. After the completion of our project work, we feel to convey our indebtness to all those who helped us to reach our goal. We take this opportunity to express our profound gratitude and deep regards to our guide (Mentor Prof. Tapan Kumar Jain) for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him time to time shall carry us a long way in the journey of life on which we are about to embark. We are obliged to all our faculty members of JUIT, for the valuable information provided by them in their respective fields. We are grateful for their cooperation during the period of our project. Lastly, we thank almighty, our Dean Prof. T.S. Lamba, HOD Prof. S.V. Bhoosan, Project coordinator Dr.Rajiv Kumar, our parents, brothers, sisters and friends for their constant encouragement without which this project would not be possible.

Abhishek Kumar Malviya

Hemant Sharma

ABSTRACT

In telecommunication, a **line code** is a code chosen for use within communication systems for baseband transmission purposes. Line coding is often used for digital data transport.

Line coding consists of representing the digital signal to be transported by an amplitude- and time-discrete signal that is optimally tuned for the specific properties of the physical channel (and of the receiving equipment). The waveform pattern of voltage or current used to represent the 1s and 0s of a digital data on a transmission link is called *line encoding*. The common types of line encoding are unipolar, polar, bipolar, and Manchester encoding.

The implementation of these line codes can be done using various techniques. We will be first implementing it using coding techniques. The next step would be the simulation of the circuits of each line code technique. After we are done with the simulation we will be doing it on hardware. The hardware implementation includes the implementation of these line codes using lab components. The next step will be to implement them using VHDL. Further we will implement it on FPGA kit.

We will see the implementation of each technique using different methods and will compare the results. The hardware implementation give us the opportunity to see the working model of the line codes.

INDEX

List of Figure

List of Acronyms

NRZ	Non return to zero
OOK	On-off keying
FPGA	Field Programmable Gate Array
PRBS	Pseudorandom binary sequence
DSP	Digital Signal Processing
PPG	Pulse Pattern Generator
SMA	Sub Miniature version A
PLD	Programmable Logic Device
IC	Integrated Circuit
SoC	Systems-on-Chip
HDL	Hardware Description Language
VHDL	Very High Speed Integrated Circuit HDL
LAB	Logic Array Block
LE	Logic Element
LUT	Lookup table
SRAM	Static Random Access Memory
ALM	Adaptive Logic Module

CHAPTER-1

INTRODUCTION

1.1 Introduction

Networks have been experiencing an exponential growth of data traffic demand over the recent years. This growth has been caused by the dramatic increases in the number of users and the increasing bandwidth requirements for applications that have been developed (high resolution video, online gaming, etc.). According to forecasts, data traffic will continue to increase over the next years [1]. Therefore, it is necessary that next telecommunication networks are built taking into account this projection.

The terminology line coding originated in telephony with the need to transmit digital information across a copper telephone *line;* more specifically, binary data over a digital repeated line [2]. The concept of line coding, however, readily applies to any transmission line or channel. In a digital communication system, there exists a known set of symbols to be transmitted [2].

Different channel characteristics, as well as different applications and performance requirements, have provided the impetus for the development and study of various types of line coding. For example, the channel might be ac coupled and, thus, could not support a line code with a dc component or large dc content. Synchronization or timing recovery requirements might necessitate a discrete component at the data rate.

Hence the use of different line Coding Techniques is necessary to implement different line codes depending upon the type of application they are put to [1] [2].

For example, if a signal needs to be tracked on the basis of its time recovery then in that case Manchester line code is suitable [3], whereas if the implementation is just limited to sending and receiving of bits then any line code like Polar or Unipolar may work [4] [2].

1.2 Problem Statement

The implementation of line codes is important in the sense that the coming age is of communication techniques. Every day we encounter new discoveries related to communication in which the data sent and received is judged and put to application on the basis of its ease of implementation and its reliability.

In this way the line codes are used for data transmission. The line codes follow a particular set of rules to send data in linear form or we can say bit by bit form.

Also since we are aware of the fact that the line codes can be implemented using both active and passive components, in our project we would also like to look at its implementation on various application software like MATLAB, MULTISIM, and VERILOG.

The implementation on MULTISIM cannot be directly implemented to a chip that can be put to multiple application, so for that purpose we will do its hardware realization using Verilog.

Further to see its real world implementation we will be burning the Verilog code over the FPGA kit. This will give us the necessary hardware required for multiple applications.

1.3 Methodology

The methodology that we have adopted is the step by step realization of the line codes.
Taking the very first step we will first be realizing the line codes using the Matlab application. We would be implementing the codes on the basis of the logic or the rule that each line code follows.
The second step will be to verify the resulting waveforms of each of the line code by designing circuits and verifying their results by simulating on simulation tools like MULTISIM. This will give us the real time simulation of the circuits that we have implemented before actually realizing them on hardware.
The third step will be to realize the circuits simulated using MULTISIM to hardware components, and observing the results on DSO.
The next step would be realizing the hardware from the logics of the line codes using HDL verilog. This will give us the clear picture of the hardware components required for the realization of each line code. The biggest advantage of using verilog is that we can realize it for n number of repetitions of the same unit of line code. Also the test bench unit of this line code will also help us to observe the output generated in form of the waveform just like we obtain it on DSO.
The last and final step would be burning the Verilog code over the FPGA kit. The kit helps us to burn different types of codes. Thus it helps to get a hardware that can generate any line code if we know its verilog code. This saves a lot of time and cost since single hardware is available to support different application.

CHAPTER-2

Theoretical Background

2.1 Background Material

2.1.1 Line Codes

i. Unipolar nrz

Unipolar encoding is a line code. A positive voltage represents a binary 1, and zero volts indicates a binary 0. It is the simplest line code, directly encoding the bit stream, and is analogous to on-off keying in modulation [2].

ii. Polar nrz

A **non-return-to-zero** (**NRZ**) line code is a binary code in which 1s are represented by one significant condition (usually a positive voltage) and 0s are represented by some other significant condition (usually a negative voltage), with no other neutral or rest condition [2].

iii. Polar rz

Return-to-zero (RZ) describes a line code used in telecommunications signals in which the signal drops (returns) to zero between each pulse. This takes place even if a number of consecutive 0s or 1s occur in the signal. The signal is self-clocking. This means that a separate clock does not need to be sent alongside the signal, but suffers from using twice the bandwidth to achieve the same data-rate as compared to non-return-to-zero format [2].

iv. Manchester line code

In telecommunication and data storage, **Manchester coding** is a line code in which the encoding of each data bit has at least one transition and occupies the same time. It therefore has no DC component, and is self-clocking, which means that it may be inductively or capacitively coupled, and that a clock signal can be recovered from the encoded data. As a result, electrical connections using a Manchester code are easily galvanically isolated using a network isolator [2].

2.2 MATLAB

MATLAB (**mat**rix **lab**oratory) is a multi paradigm numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java,Fortran and Python [5].

Although MATLAB is intended primarily for numerical computing, an optional toolbox uses the MuPAD symbolic engine, allowing access to symbolic computing capabilities. An additional package, Simulink, adds graphical multi-domain simulation and Model-Based Design for dynamic and embedded systems [6].

In 2004, MATLAB had around one million users across industry and academia. MATLAB users come from various backgrounds of engineering, science, and economics. MATLAB is widely used in academic and research institutions as well as industrial enterprises.

MATLAB can call functions and subroutines written in the C programming language or Fortran. A wrapper function is created allowing MATLAB data types to be passed and returned. The dynamically loadable object files created by compiling such functions are termed "MEX-files" (for **M**ATLAB **ex**ecutable) [5][6].

Libraries written in Perl, Java, ActiveX or .NET can be directly called from MATLAB,and many MATLAB libraries (for example XML or SQL support) are implemented as wrappers around Java or ActiveX libraries. Calling MATLAB from Java is more complicated, but can be done with a MATLAB toolbox which is sold separately by MathWorks, or using an undocumented mechanism called JMI (Java-to-MATLAB Interface), (which should not be confused with the unrelated Java Metadata Interface that is also called JMI) [5].

2.3 MutiSim

NI Multisim (formerly **MultiSIM**) is an electronic schematic capture and simulation program which is part of a suite of circuit design programs, along with NI Ultiboard. Multisim is one of the few circuit design programs to employ the original Berkeley SPICE based software simulation. Multisim was originally created by a company named Electronics Workbench, which is now a division of National Instruments. Multisim includes microcontroller simulation (formerly known as MultiMCU), as well as integrated import and export features to the Printed Circuit Board layout software in the suite, NI Ultiboard [7] [8].

Multisim is widely used in academia and industry for circuits education, electronic schematic design and SPICE simulation [7] [8].

2.4 VHDL

(**VHSIC Hardware Description Language**) is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate array and integrated circuits. VHDL can also be used as a general purpose parallel programming language [9].

Advantages

The key advantage of VHDL, when used for systems design, is that it allows the behavior of the required system to be described (modeled) and verified (simulated) before synthesis tools translate the design into real hardware (gates and wires).

Another benefit is that VHDL allows the description of a concurrent system. VHDL is a dataflow language, unlike procedural computing languages such as BASIC, C, and assembly code, which all run sequentially, one instruction at a time.

A VHDL project is multipurpose. Being created once, a calculation block can be used in many other projects. However, many formational and functional block parameters can be tuned (capacity parameters, memory size, element base, block composition and interconnection structure).

A VHDL project is portable. Being created for one element base, a computing device project can be ported on another element base, for example VLSI with various technologies [9].

2.5 VHDL vs VERILOG

✓ ***Capability***

When modeling abstract hardware, the capability of VHDL can sometimes only be achieved in Verilog when using the PLI. The choice of which to use is not therefore based solely on technical capability but on:

- personal preferences
- EDA tool availability
- commercial, business and marketing issues

The modeling constructs of VHDL and Verilog cover a slightly different spectrum across the levels of behavioral abstraction.

✓ ***Compilation***

VHDL. Multiple design-units (entity/architecture pairs), that reside in the same system file, may be separately compiled if so desired. However, it is good design practice to keep each design unit in its own system file in which case separate compilation should not be an issue.

Verilog. The Verilog language is still rooted in its native interpretative mode. Compilation is a means of speeding up simulation, but has not changed the original nature of the language. As a result care must be taken with both the compilation order of code written in a single file and the compilation order of multiple files. Simulation results can change by simply changing the order of compilation.

✓ ***Data types***

VHDL. A variety of language or user defined data types can be used. Dedicated conversion functions are needed to convert objects from one type to another. The choice of which data types to use should be considered wisely, especially enumerated (abstract) data types. It makes models easier to write, clearer to read and avoid unnecessary conversion functions that can clutter the code [10].

Verilog. Compared to VHDL, Verilog data types are very simple, easy to use and very much geared towards modelling hardware structure as opposed to abstract hardware modelling. Unlike VHDL, all data types used in a Verilog model are defined by the Verilog language and not by the user. There are net data types, for example wire, and a register data type called reg. A model with a signal whose type is one of the net data types has a corresponding electrical wire in the implied modeled circuit. Objects, that is signals, of type reg hold their value over simulation delta cycles and should not be confused with the modeling of a hardware register. Verilog may be preferred because of its simplicity [10].

- ✓ ***Design reusability***

VHDL. Procedures and functions may be placed in a package so that they are available to any design-unit that wishes to use them.

Verilog. There is no concept of packages in Verilog. Functions and procedures used within a model must be defined in the module. To make functions and procedures generally accessible from different module statements the functions and procedures must be placed in a separate system file and included using the `include compiler directive.

- ✓ ***Easiest to Learn***

Starting with zero knowledge of either language, Verilog is probably the easiest to grasp and understand. This assumes the Verilog compiler directive language for simulation and the PLI language is not included. If these languages are included they can be looked upon as two additional languages that need to be learned. VHDL may seem less intuitive at first for two primary reasons. First, it is very strongly typed; a feature that makes it robust and powerful for the advanced user after a longer learning phase. Second, there are many ways to model the same circuit, especially those with large hierarchical structures [10].

- ✓ ***Forward and back annotation***

A spin-off from Verilog is the Standard Delay Format (SDF). This is a general purpose format used to define the timing delays in a circuit. The format provides a bidirectional link between, chip layout tools, and either synthesis or simulation tools, in order to provide more accurate timing representations. The SDF format is now an industry standard in its own right.

- ✓ ***High level constructs***

VHDL. There are more constructs and features for high-level modelling in VHDL than there are in Verilog. Abstract data types can be used along with the following statements:

* Package statements for model reuse,

* Configuration statements for configuring design structure,

* Generate statements for replicating structure,

* Generic statements for generic models that can be individually characterized, for example, bit width.

Verilog. Except for being able to parameterize models by overloading parameter constants, there is no equivalent to the high-level VHDL modelling statements in Verilog.

✓ ***Language Extensions***

The use of language extensions will make a model non standard and most likely not portable across other design tools. However, sometimes they are necessary in order to achieve the desired results.

VHDL. Has an attribute called 'foreign that allows architectures and subprograms to be modelled in another language.

Verilog. The Programming Language Interface (PLI) is an interface mechanism between Verilog models and Verilog software tools. For example, a designer, or more likely, a Verilog tool vendor, can specify user defined tasks or functions in the C programming language, and then call them from the Verilog source description. Use of such tasks or functions make a Verilog model nonstandard and so may not be usable by other Verilog tools. Their use is not recommended.

✓ ***Libraries***

VHDL. A library is a store for compiled entities, architectures, packages and configurations. Useful for managing multiple design projects.

Verilog. There is no concept of a library in Verilog. This is due to it's origins as an interpretive language.

✓ ***Low Level Constructs***

VHDL. Simple two input logical operators are built into the language, they are: NOT, AND, OR, NAND, NOR, XOR and XNOR. Any timing must be separately specified using the after clause. Separate constructs defined under the VITAL language must be used to define the cell primitives of ASIC and FPGA libraries.

Verilog. The Verilog language was originally developed with gate level modelling in mind, and so has very good constructs for modelling at this level and for modelling the cell primitives of ASIC and FPGA libraries. Examples include User Defined Primitive s (UDP), truth tables and the specify block for specifying timing delays across a module [9].

2.6 FPGA

2.6.1 HARDWARE IMPLEMENTATION ON FPGA

FPGA: A field-programmable gate array (FPGA) is an integrated circuit (IC) that can be programmed in the field after manufacture. FPGAs are similar in principle to, but have vastly wider potential application than, programmable read-only memory (PROM) chips [10].

ATLYS BOARD: The Atlys circuit board is a complete, ready-to-use digital circuit development platform based on a Xilinx Spartan 6 LX45 FPGA. The on-board collection of high-end peripherals, including Gbit Ethernet, HDMI Video, 128Mbyte DDR2 memory array, audio and USB ports make the Atlys board an ideal host for complete digital systems built around embedded processors like Xilinx’s MicroBlaze. Atlys is fully compatible with all Xilinx CAD tools, including ChipScope, EDK, and the free WebPack, so designs can be completed with no extra costs. The Atlys board can be communicated with and programmed by the Digilent Adept Software .In addition, the board can be programmed by Xilinx's iMPACT using the Digilent Plugin for Xilinx Tools [13].

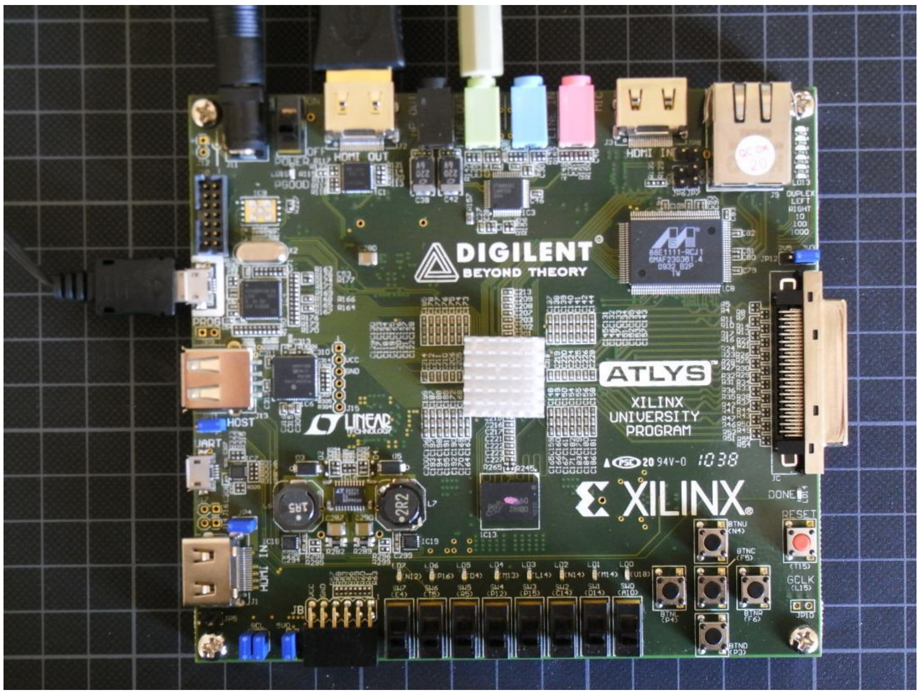


Figure 2.1:Atlys Board

The Spartan-6 LX45 is optimized for high-performance logic and offers:

- 6,822 slices each containing four 6-input LUTs and eight flip-flops
- 2.1Mbits of fast block RAM
- 4 clock tiles (8 DCMs & 4 PLLs)
- 6 phased-locked loops
- 58 DSP slices
- 500MHz+ clock speeds

2.6.2 XILINX PLATFORM STUDIO

Xilinx Platform Studio (XPS) is a key component of the ISE Embedded Edition Design Suite, helping the hardware designer to easily build, connect and configure embedded processor-based systems; from simple state machines to full-blown 32-bit RISC microprocessor systems. XPS employs graphical design views and sophisticated correct-by-design wizards to guide developers through the steps necessary to create custom processor systems within minutes[15].

The true potential of XPS emerges with its ability to configure and integrate plug and play IP cores from the Xilinx Embedded IP catalogue, with custom or 3rd party Verilog and VHDL designs. Now, highly-custom processors can be designed according to project-specific needs include; peripheral and IO requirements, real-time responsiveness, general purpose processing power, floating point performance, on-chip or off-chip memory, minimal power consumption and much more [15] [16].

2.6.3 XILINX SOFTWARE DEVELOPMENT KIT

The Software Development Kit (SDK) is the Xilinx Integrated Design Environment for creating embedded applications on any of Xilinx' award winning microprocessors for Zynq 7000 All Programmable SoCs, and the industry-leading MicroBlaze. The SDK is the first application IDE to deliver true homogenous and heterogeneous multi-processor design and debug.

Firmware and software developers benefit from XPS integration with Xilinx SDK which allows the automatic generation of critical system software such as boot loaders, bare metal BSP, and Linux BSPs. This capability ensures that OS porting and applications development can begin without delay caused by firmware development [20].

2.6.4 Steps for recognizing and programming the FPGA

Once you have compiled your program on Xilinx ISE, following steps will help you program the FPGA:

1. Connect the FPGA board with the power supply and switch it on. An LED near the power supply port should start glowing.

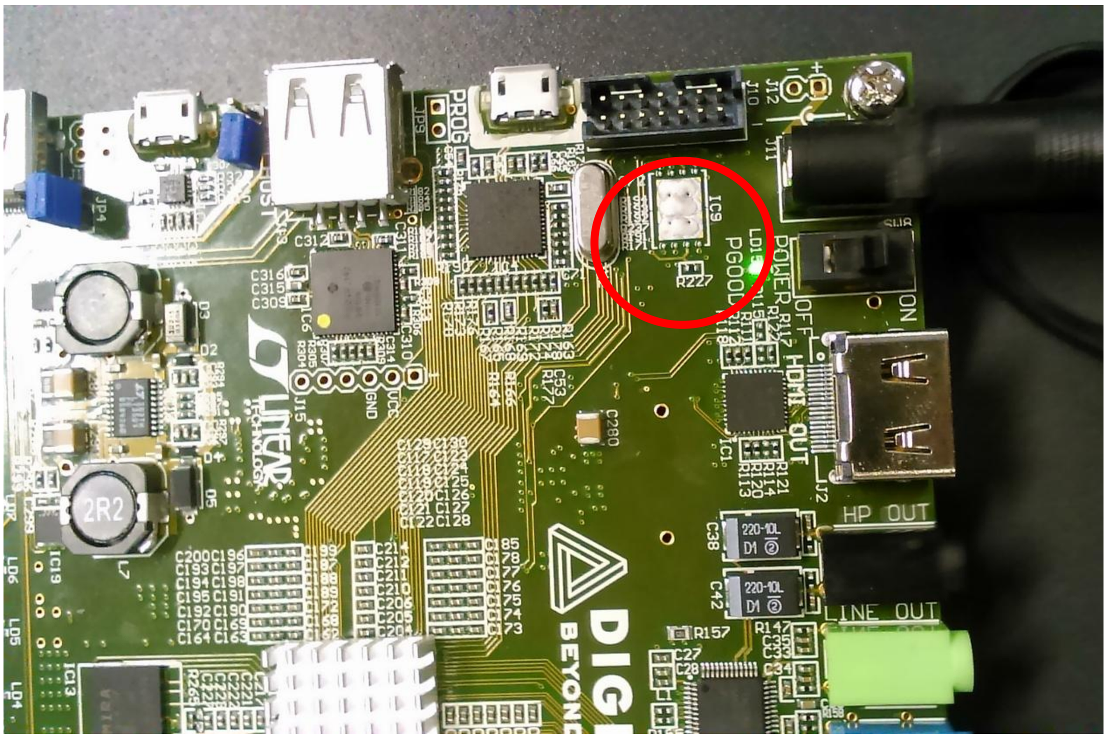


Figure 2.2:On LED blinking

2. Now, start the ISE Design Suite where your VHDL/Verilog code is present that needs to be programmed on FPGA.
3. After you have compiled the code successfully, remove your testbench from the code and just keep your main code file.
4. Now, you need to build a .ucf file which will define the inputs/outputs present in your code is connected to which ports of fpga. Expand the user constraints tab and click on I/O Pin Planning (post synthesis). This will open a PlanAhead window.

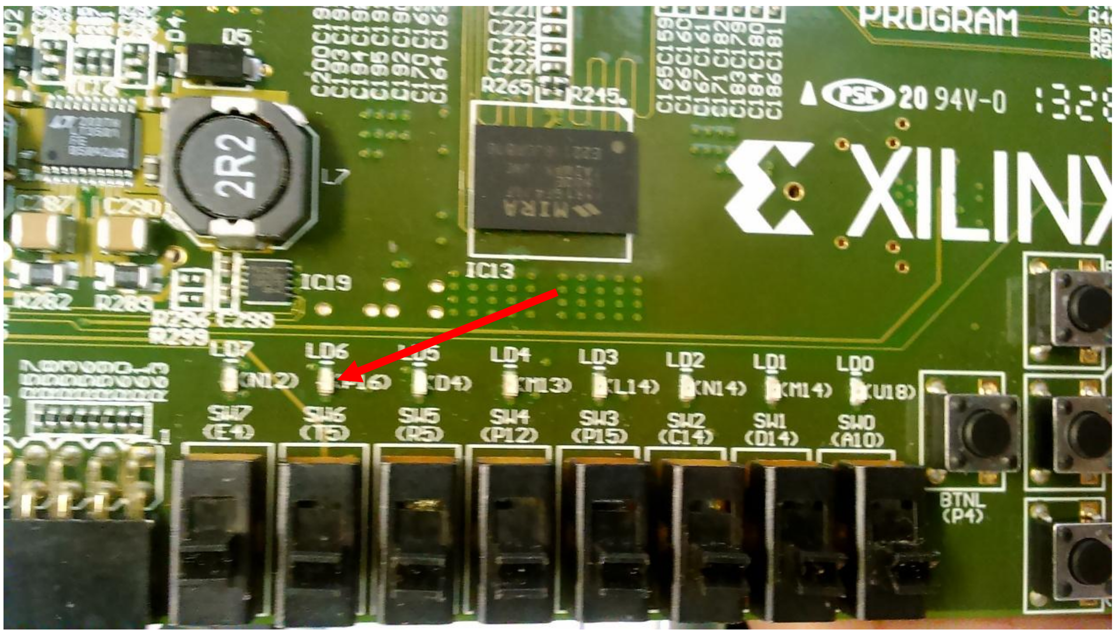
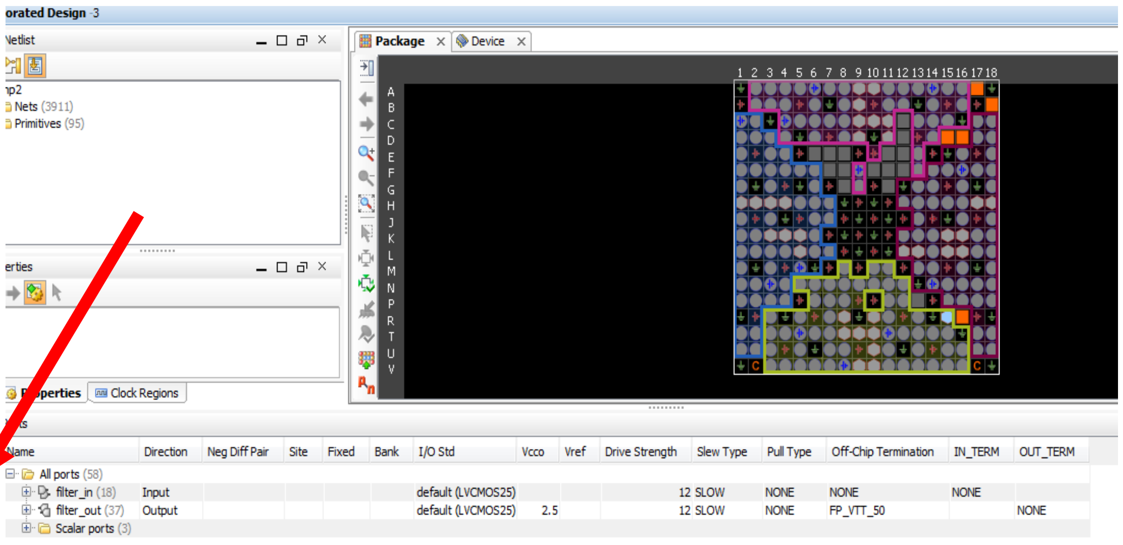


Figure 2.3: Port numbers written in brackets



Name	Direction	Neg Diff Pair	Site	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM	OUT_TERM
All ports (58)														
filter_in (18)	Input					default (LVCMOS25)			12	SLOW	NONE	NONE	NONE	
filter_out (37)	Output					default (LVCMOS25)	2.5		12	SLOW	NONE	FP_VTT_50		NONE
Scalar ports (3)														

Figure 2.4: PlanAhead Window for Pin Planning

5. Now, you will see your input and output variables below. If they are more than 1 bit long , expand them and assign each bit input and output to the required port on FPGA board by typing it manually or selecting ports under the "**SITE**" option present. After this save the design and exit.

6. Now, in the ISE Design Suite , Click on the generate programming file tab on the left side as shown in the figure below.

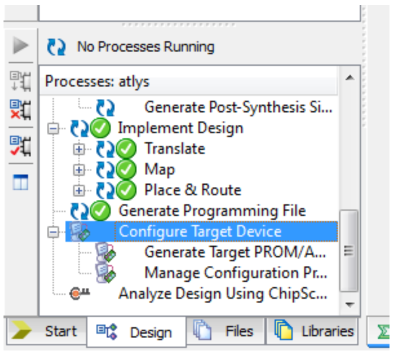

No Processes Running
Processes: atlys
Generate Post-Synthesis Si...
Implement Design
Translate
Map
Place & Route
Generate Programming File
Configure Target Device
Generate Target PROM/A...
Manage Configuration Pr...
Analyze Design Using ChipSc...
Start
Design
Files
Libraries

Figure 2.5: Xilinx ISE Design Suite

7. This will run all the processes above it and if it compiles successfully, it will give a green tick mark for completion.
8. Now, click on configure target device, this will ask you to open an impact tool click on ok and proceed. Click on Boundary Scan on the left side. A blank screen will open up and then right click on anywhere on that screen and click on **Initialize chain** from the set of options.

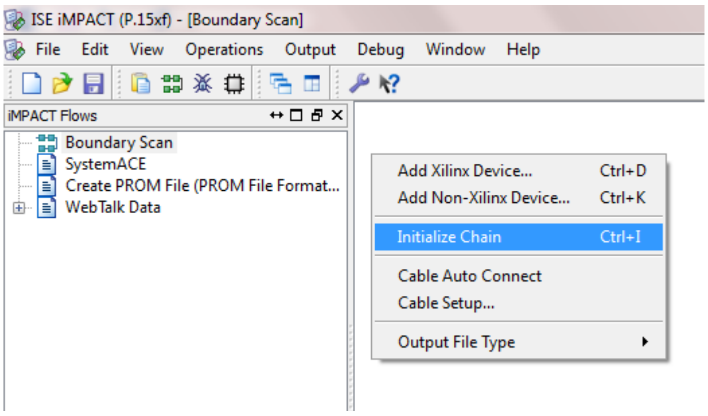


Figure 2.6: Configuring the FPGA

9. This will show a Xilinx device present (make sure your FPGA is connected to the computer and switched on).Now, assign configuration file and select the .bit file that has been generated inside your Xilinx project. After the configuration file has been assigned right click on the device to program it [17] [18].

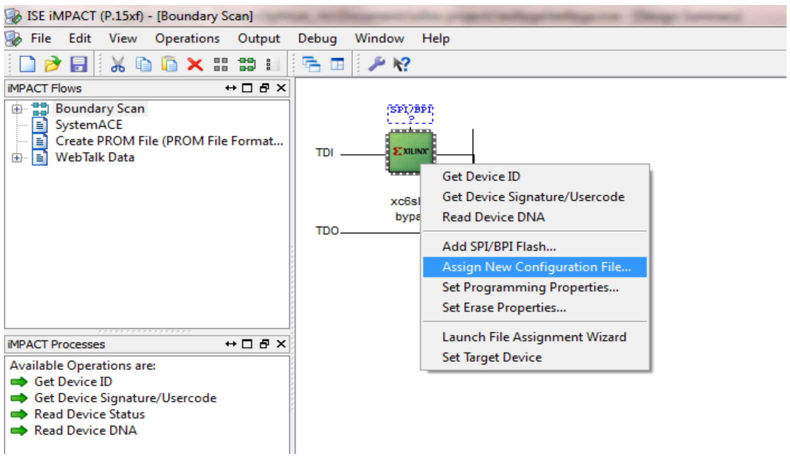
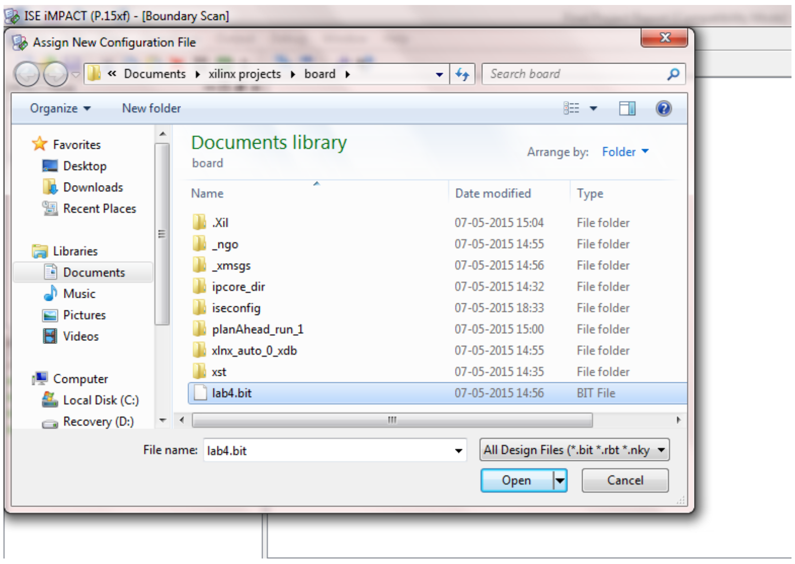

ISE iMPACT (P.15xf) - [Boundary Scan]
File Edit View Operations Output Debug Window Help
iMPACT Flows
Boundary Scan
SystemACE
Create PROM File (PROM File Format...
WebTalk Data
iMPACT Processes
Available Operations are:
Get Device ID
Get Device Signature/Usercode
Read Device Status
Read Device DNA
SPI/BPI
TDI
TDO
xc6sl
bypa
Get Device ID
Get Device Signature/Usercode
Read Device DNA
Add SPI/BPI Flash...
Assign New Configuration File...
Set Programming Properties...
Set Erase Properties...
Launch File Assignment Wizard
Set Target Device

Figure 2.7: Assigning configuration file



Name	Date modified	Type
.Xil	07-05-2015 15:04	File folder
_ngo	07-05-2015 14:55	File folder
_xmsgs	07-05-2015 14:56	File folder
ipcore_dir	07-05-2015 14:32	File folder
iseconfig	07-05-2015 18:33	File folder
planAhead_run_1	07-05-2015 15:00	File folder
xlnx_auto_0_xdb	07-05-2015 14:55	File folder
xst	07-05-2015 14:35	File folder
lab4.bit	07-05-2015 14:56	BIT File

File name: lab4.bit

All Design Files (*.bit *.rbt *.nky

Figure 2.8: Path for .bit file

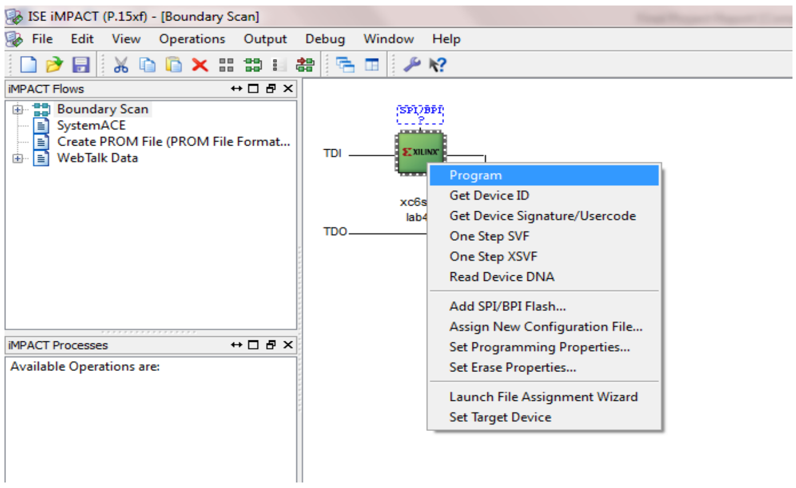

ISE iMPACT (P.15xf) - [Boundary Scan]
File Edit View Operations Output Debug Window Help
iMPACT Flows
Boundary Scan
SystemACE
Create PROM File (PROM File Format...
WebTalk Data
iMPACT Processes
Available Operations are:
SPI/BPI
TDI
TDO
xc6s
lab4
Program
Get Device ID
Get Device Signature/Usercode
One Step SVF
One Step XSVF
Read Device DNA
Add SPI/BPI Flash...
Assign New Configuration File...
Set Programming Properties...
Set Erase Properties...
Launch File Assignment Wizard
Set Target Device

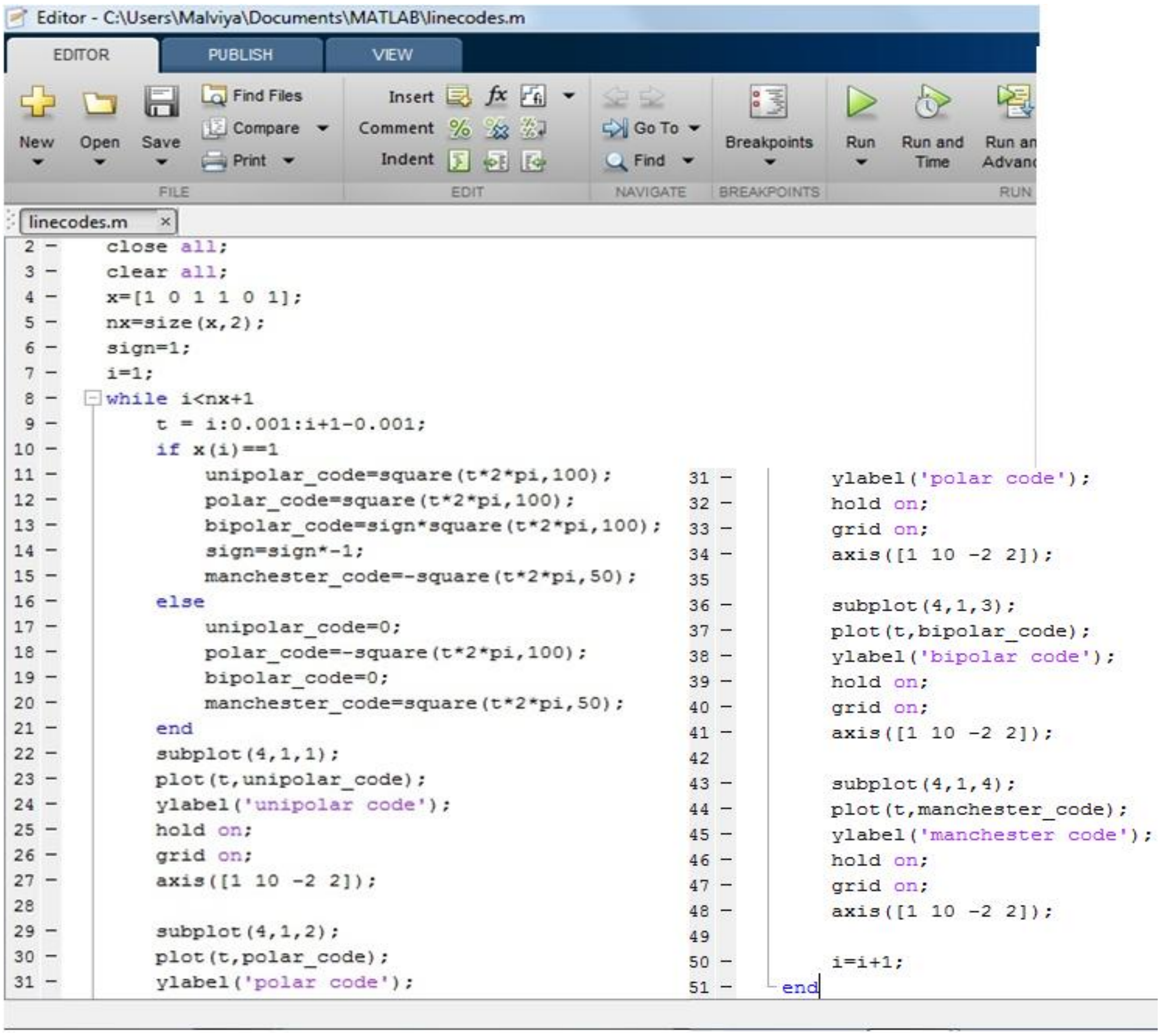
Figure 2.9: Programming the device

CHAPTER-3

Work Description

3. Work Description

3.1 Implementation in MATLAB



```
close all;
clear all;
x=[1 0 1 1 0 1];
nx=size(x,2);
sign=1;
i=1;
while i<nx+1
    t = i:0.001:i+1-0.001;
    if x(i)==1
        unipolar_code=square(t*2*pi,100);
        polar_code=square(t*2*pi,100);
        bipolar_code=sign*square(t*2*pi,100);
        sign=sign*-1;
        manchester_code=-square(t*2*pi,50);
    else
        unipolar_code=0;
        polar_code=-square(t*2*pi,100);
        bipolar_code=0;
        manchester_code=square(t*2*pi,50);
    end
    subplot(4,1,1);
    plot(t,unipolar_code);
    ylabel('unipolar code');
    hold on;
    grid on;
    axis([1 10 -2 2]);

    subplot(4,1,2);
    plot(t,polar_code);
    ylabel('polar code');
    hold on;
    grid on;
    axis([1 10 -2 2]);

    subplot(4,1,3);
    plot(t,bipolar_code);
    ylabel('bipolar code');
    hold on;
    grid on;
    axis([1 10 -2 2]);

    subplot(4,1,4);
    plot(t,manchester_code);
    ylabel('manchester code');
    hold on;
    grid on;
    axis([1 10 -2 2]);

    i=i+1;
end
```

Figure 3.1:Implimentation code of matlab

3.2 Implementation on MULTISIM

i. Unipolar NRZ

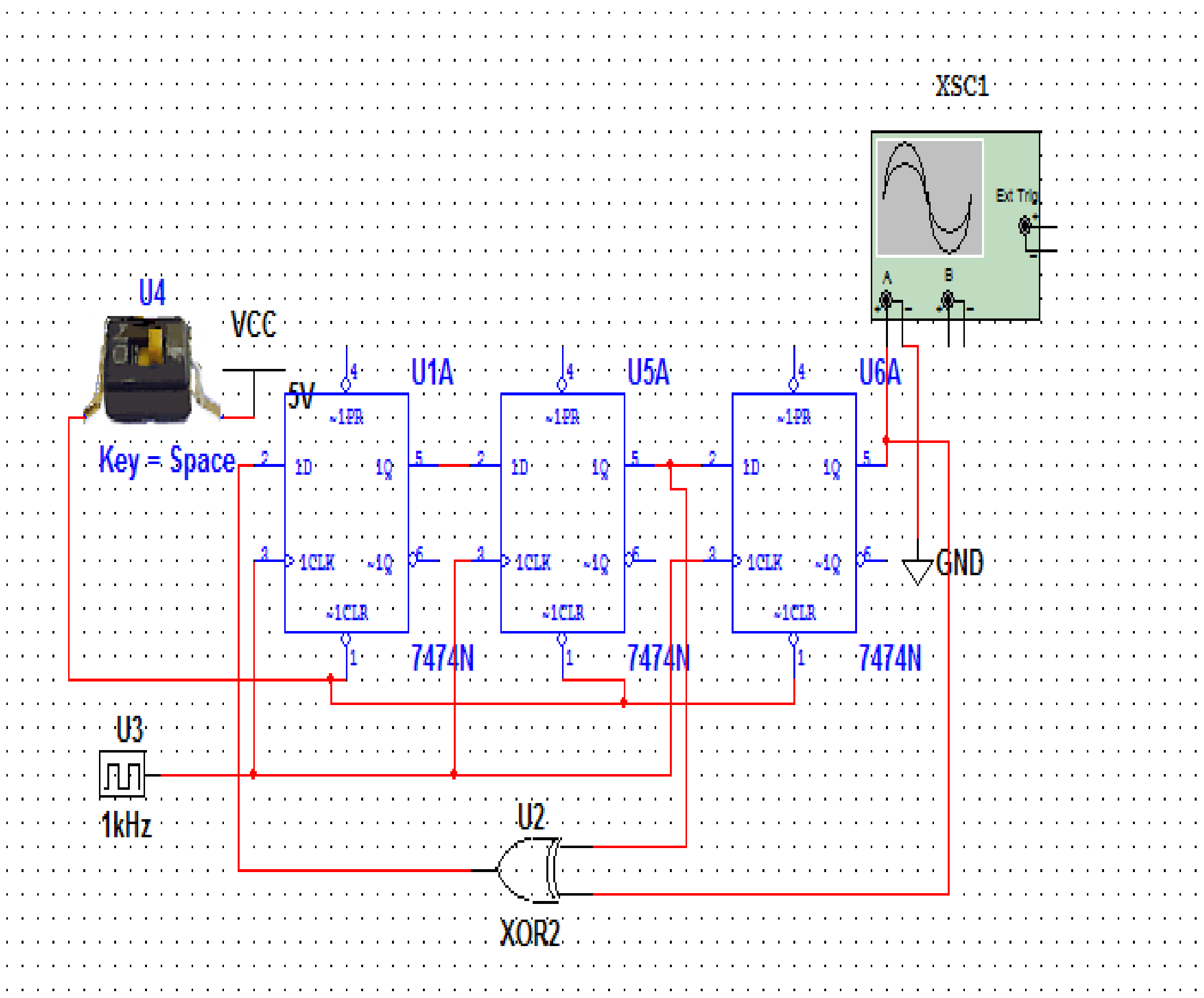


Figure 3.2:Implimentaion of Unipolar NRZ on Multisim

ii. Polar NRZ

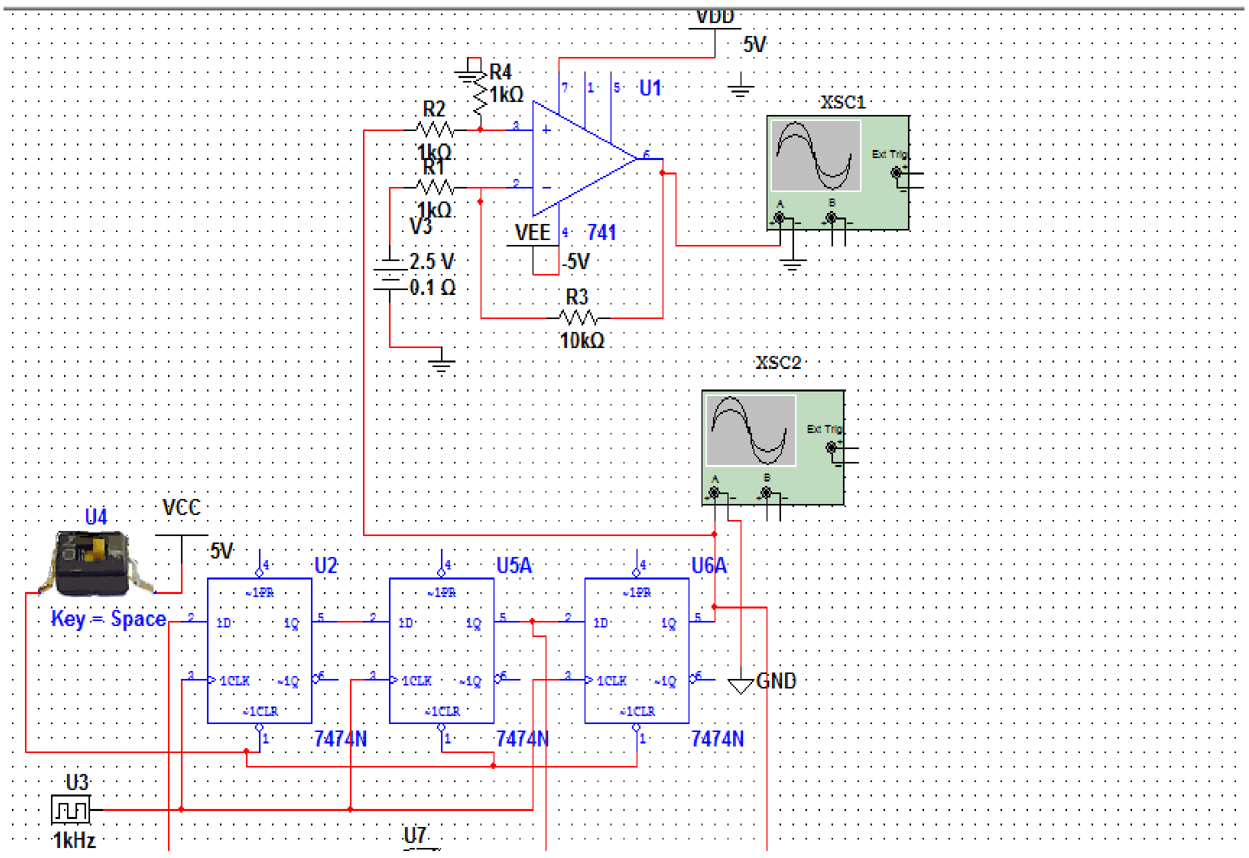


Figure 3.3:Implimentation of Polar NRZ on Multisim

iii. Polar RZ

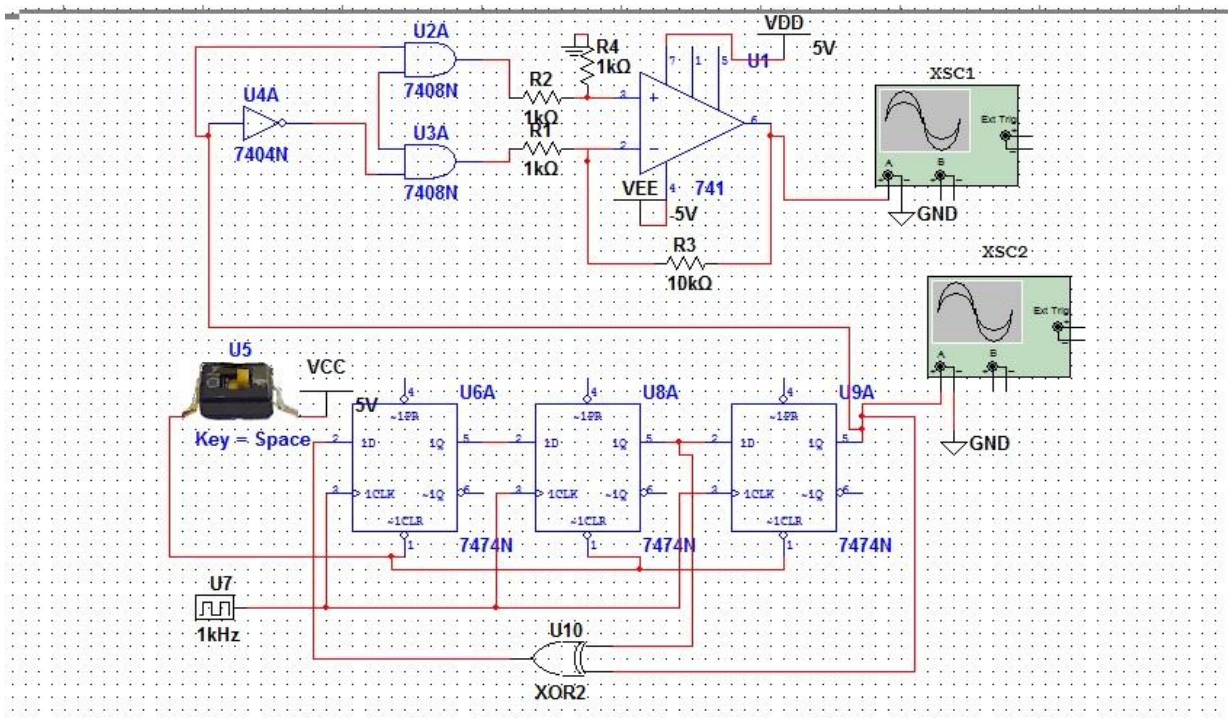


Figure 3.4:Implimentation of Polar RZ on Multisim

iv. Manchester line code

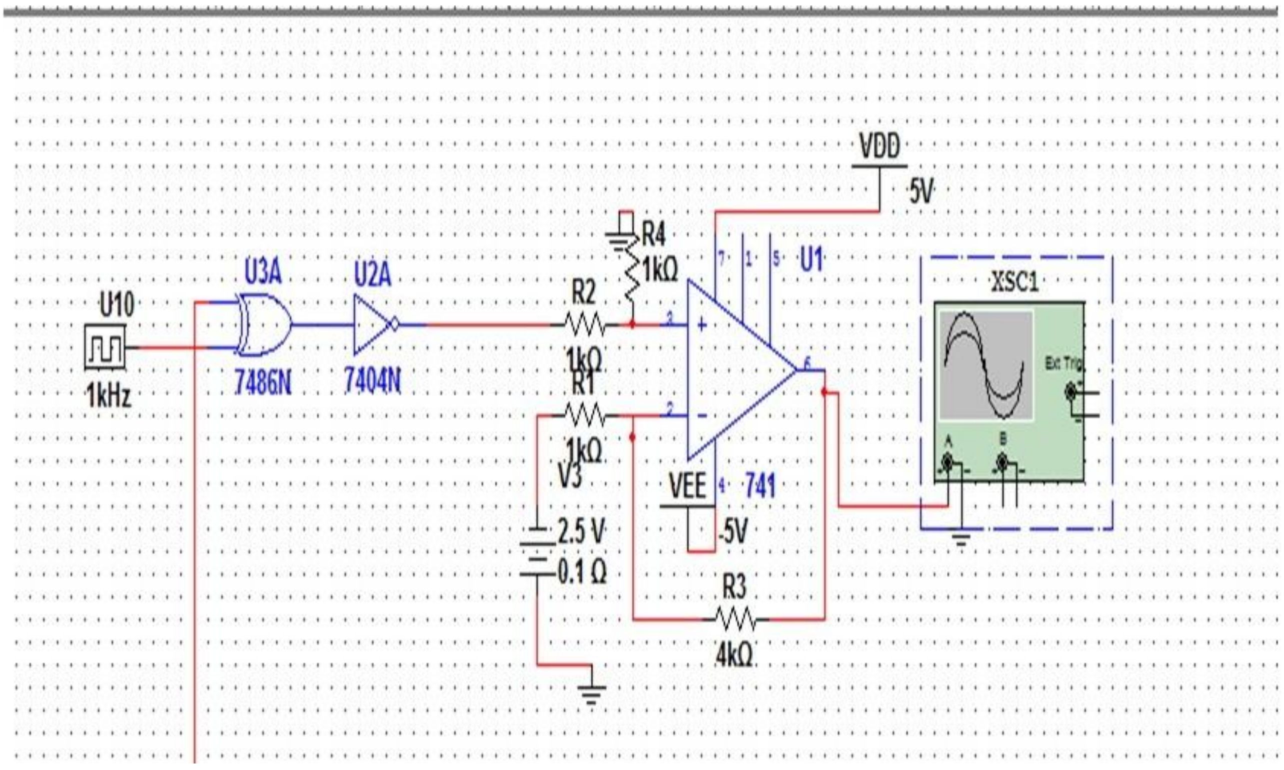


Figure 3.5:Implimentation of Manchester on Multisim

3.3 Implementation on Hardware Using Lab Components

i. Pseudo Random Number Generator

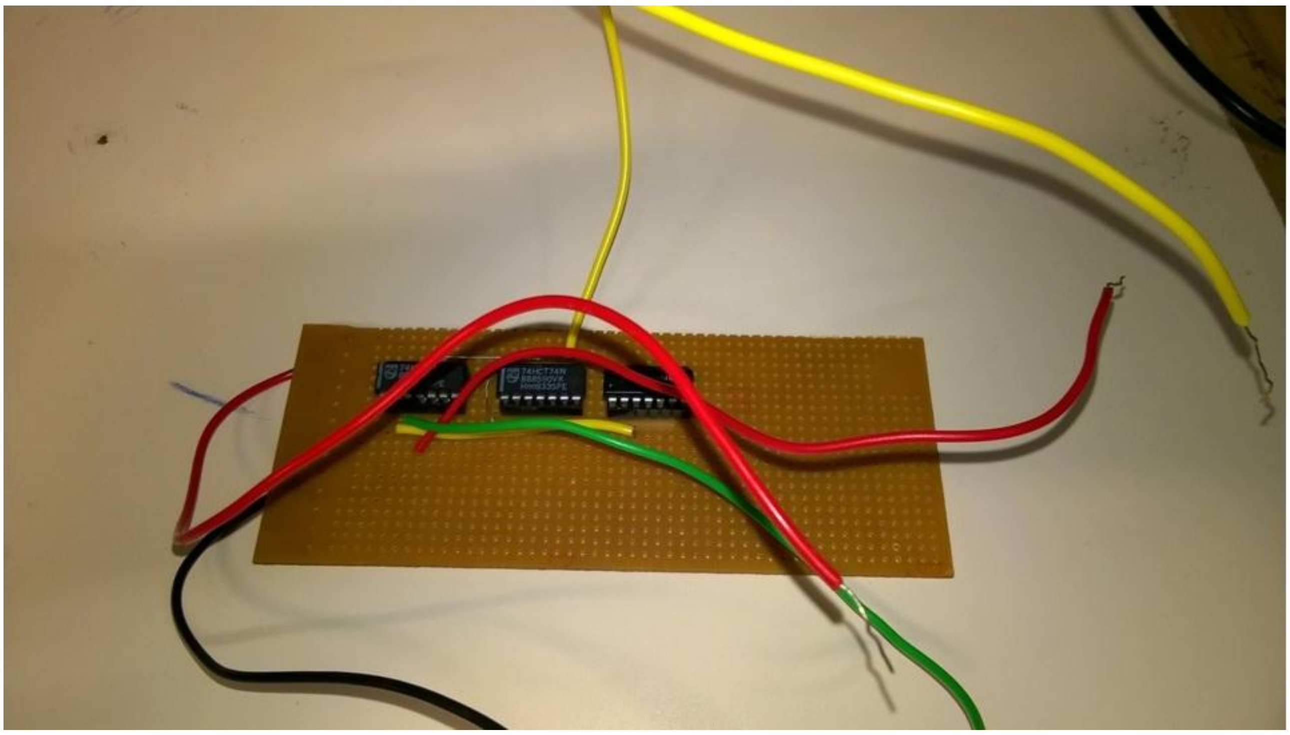


Figure 3.6:PRNG Hardware

ii. Manchester Line Code Implementation

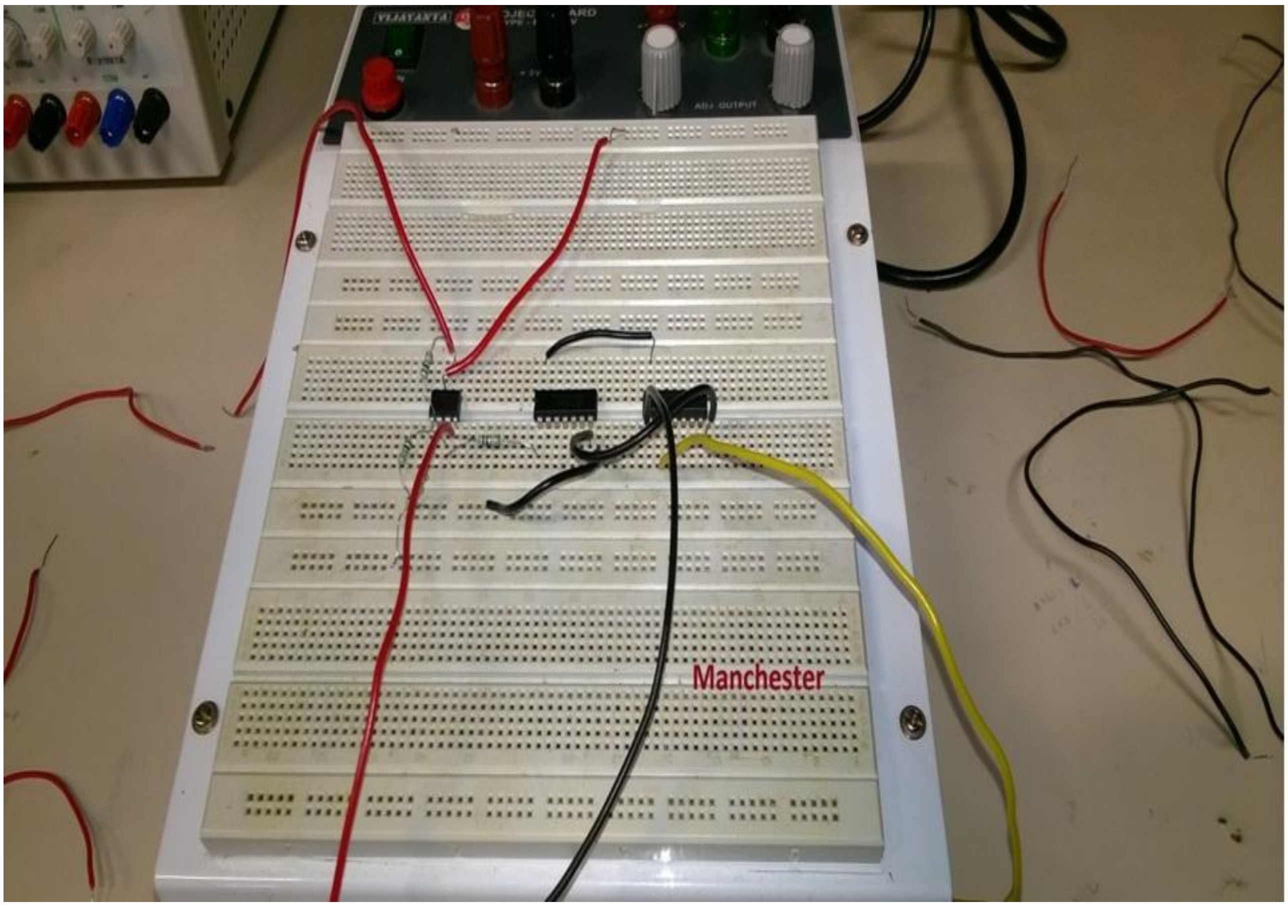
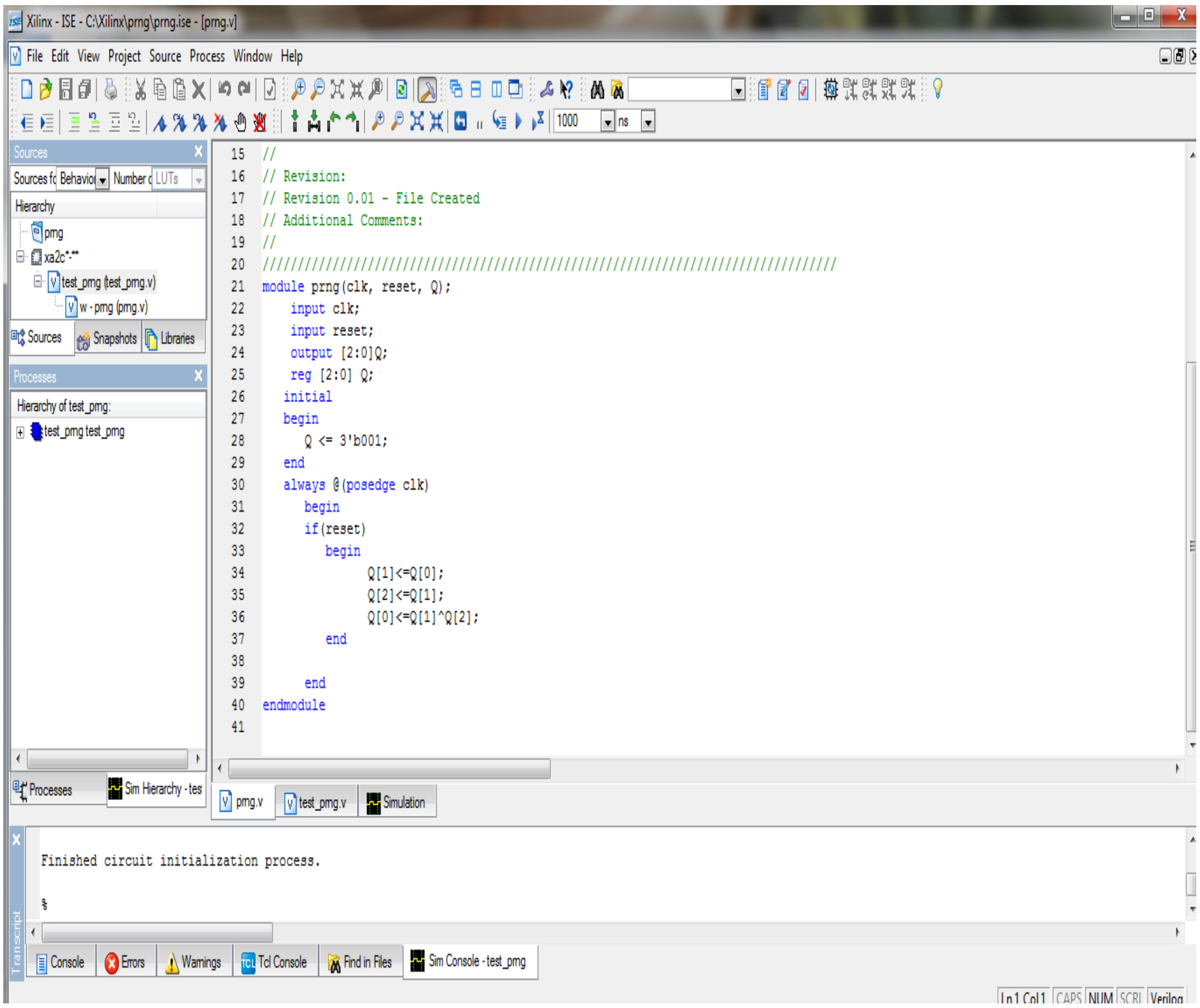


Figure 3.7:Manchester Hardware

3.4 Implementation using Verilog

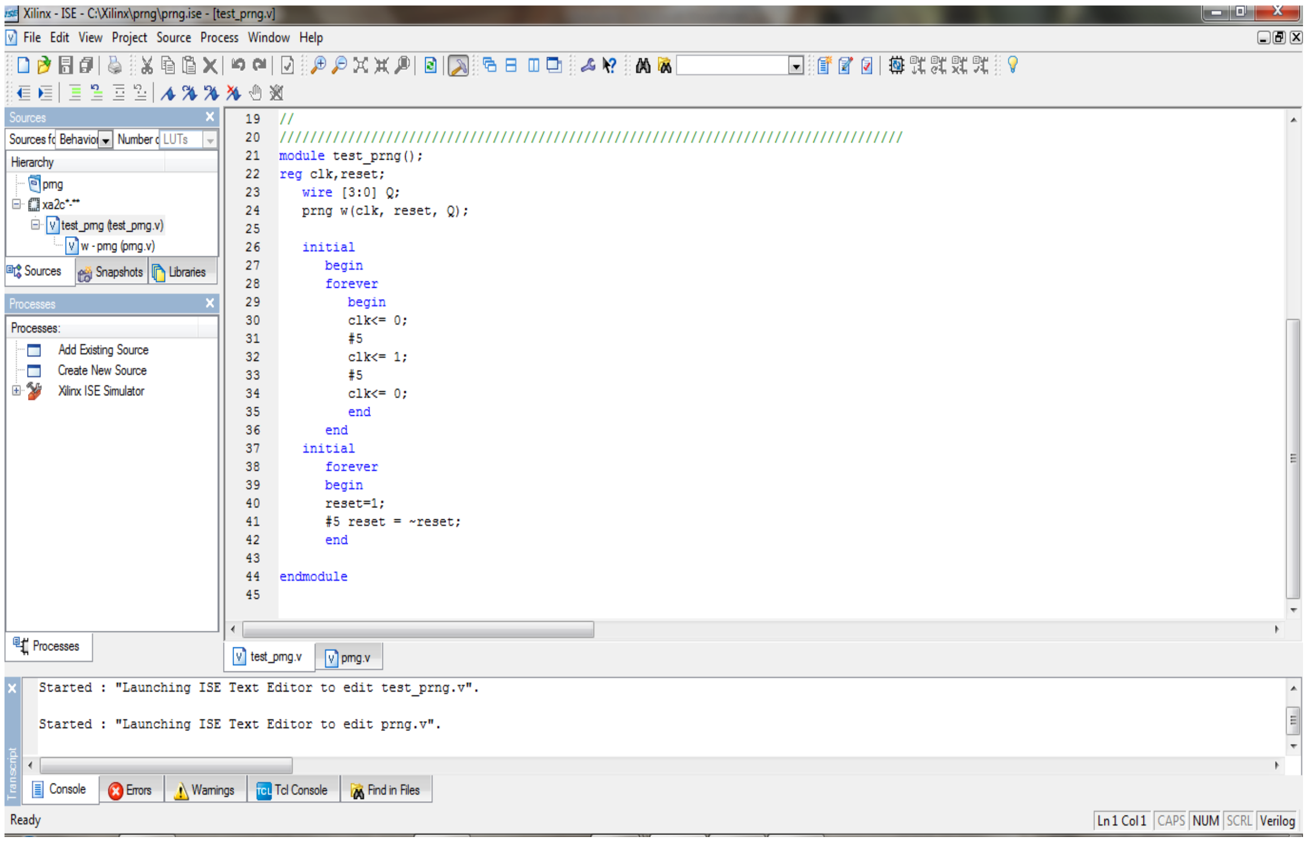
i. Pseudo Random number Generator


Xilinx - ISE - C:\Xilinx\prng\prng.ise - [prng.v]
File Edit View Project Source Process Window Help
Sources
Hierarchy
prng
xa2c*-**
test_prng (test_prng.v)
w - prng (prng.v)
Sources Snapshots Libraries
Processes
Hierarchy of test_prng:
test_prng test_prng
Processes Sim Hierarchy - tes
prng.v test_prng.v Simulation
Console Errors Warnings Tcl Console Find in Files Sim Console - test_prng

```
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module prng(clk, reset, Q);
    input clk;
    input reset;
    output [2:0]Q;
    reg [2:0] Q;
   initial
   begin
      Q <= 3'b001;
   end
   always @(posedge clk)
      begin
      if(reset)
         begin
               Q[1]<=Q[0];
               Q[2]<=Q[1];
               Q[0]<=Q[1]^Q[2];
         end

      end
endmodule
```

```
Finished circuit initialization process.

%
```

Figure 3.8: Verilog code for PRNG

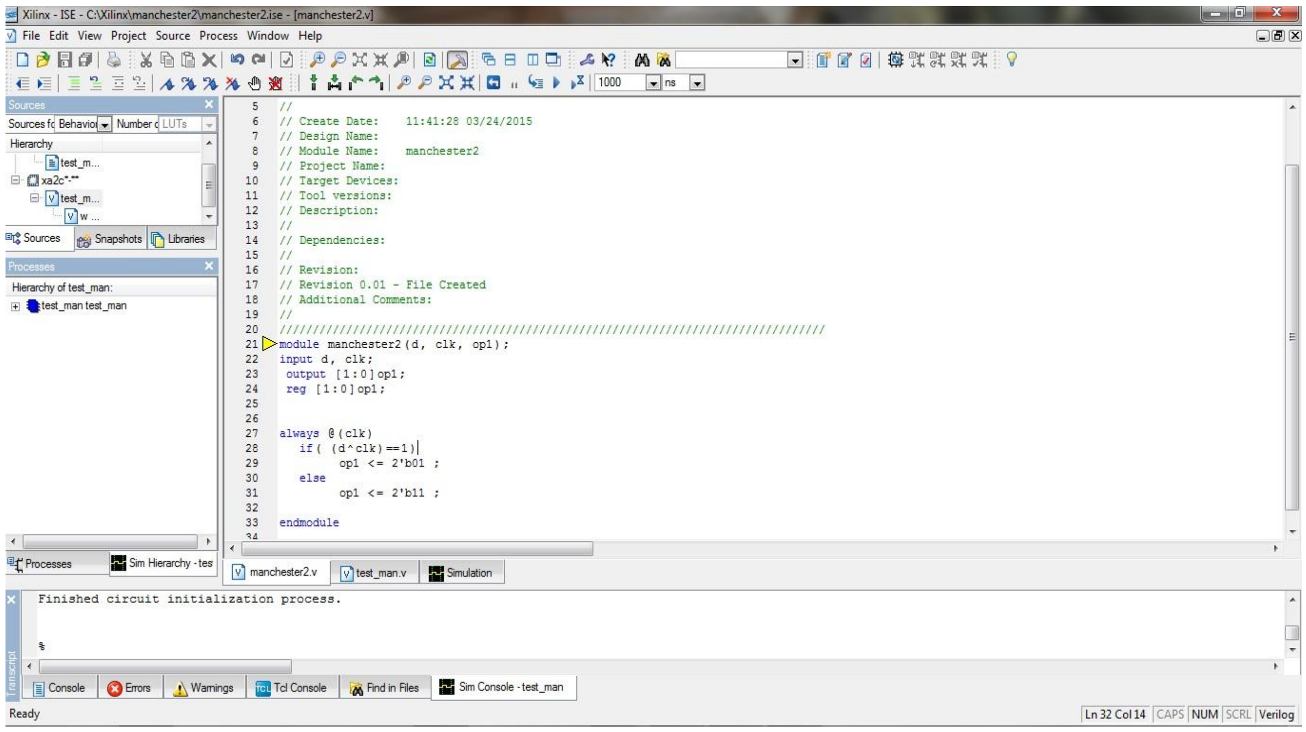


```
//
////////////////////////////////////////////////////////////////////////////////
module test_prng();
reg clk,reset;
   wire [3:0] Q;
   prng w(clk, reset, Q);

   initial
      begin
      forever
         begin
         clk<= 0;
         #5
         clk<= 1;
         #5
         clk<= 0;
         end
      end
   initial
      forever
      begin
      reset=1;
      #5 reset = ~reset;
      end

endmodule

```

```
Started : "Launching ISE Text Editor to edit test_prng.v".

Started : "Launching ISE Text Editor to edit prng.v".
```

Figure 3.9 Test Bench code for PRNG

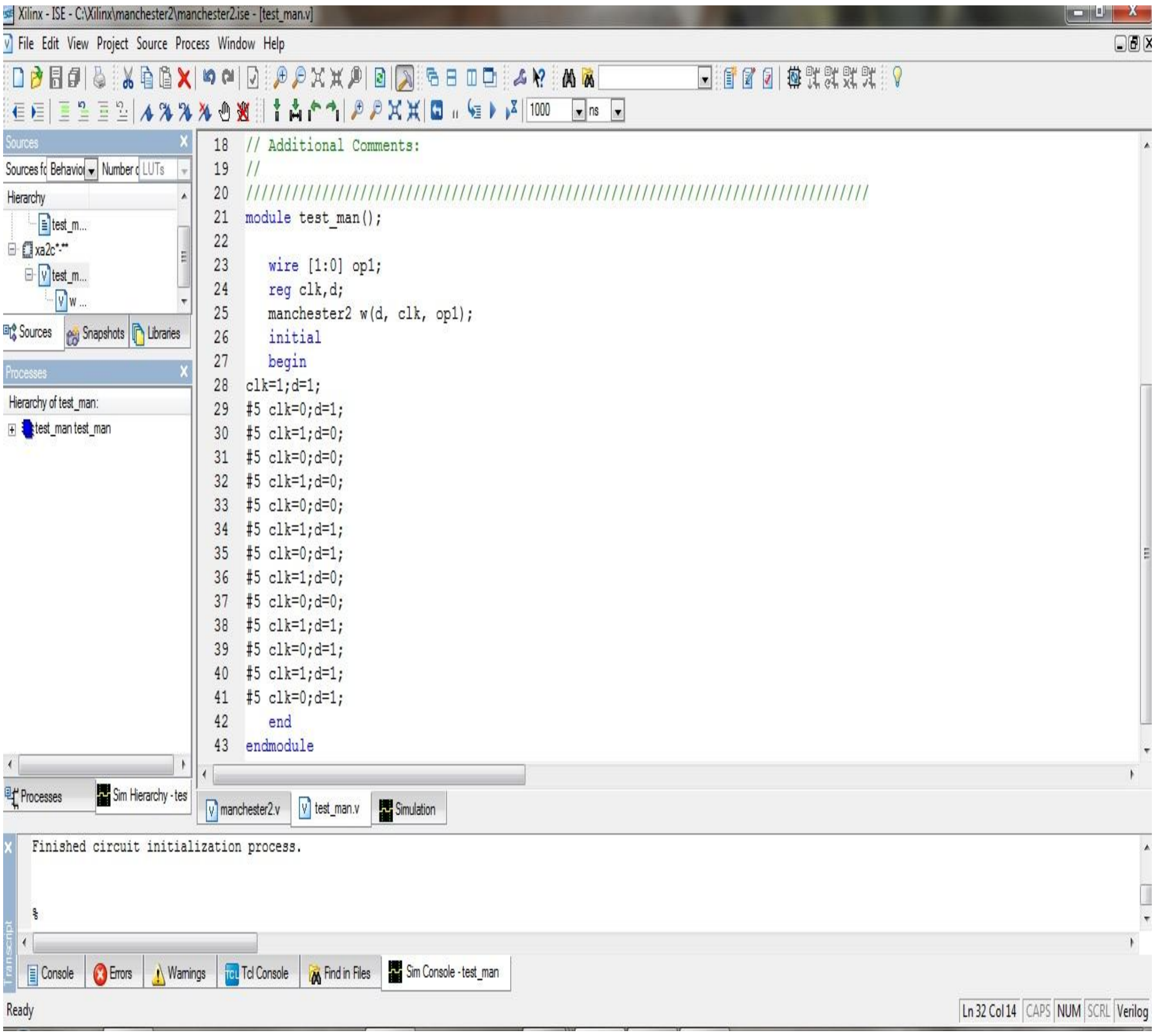
ii. Manchester Line code



```
//
// Create Date:    11:41:28 03/24/2015
// Design Name:
// Module Name:    manchester2
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module manchester2(d, clk, op1);
input d, clk;
 output [1:0]op1;
 reg [1:0]op1;


always @(clk)
   if( (d^clk)==1)
         op1 <= 2'b01 ;
   else
         op1 <= 2'b11 ;

endmodule
```

```
Finished circuit initialization process.

%
```

Figure 3.10 Verilog code for Manchester

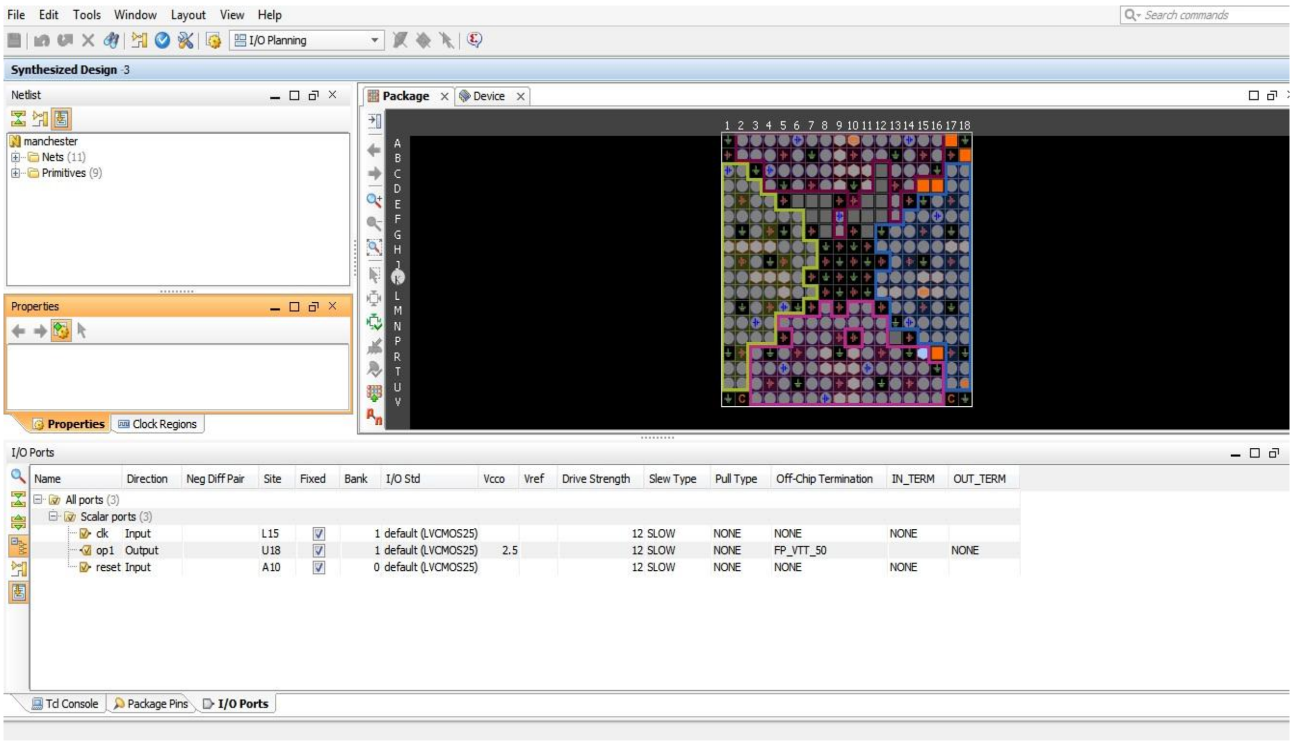


```
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module test_man();

  wire [1:0] op1;
  reg clk,d;
  manchester2 w(d, clk, op1);
  initial
  begin
clk=1;d=1;
#5 clk=0;d=1;
#5 clk=1;d=0;
#5 clk=0;d=0;
#5 clk=1;d=0;
#5 clk=0;d=0;
#5 clk=1;d=1;
#5 clk=0;d=1;
#5 clk=1;d=0;
#5 clk=0;d=0;
#5 clk=1;d=1;
#5 clk=0;d=1;
#5 clk=1;d=1;
#5 clk=0;d=1;
  end
endmodule
```

```
Finished circuit initialization process.

%
```

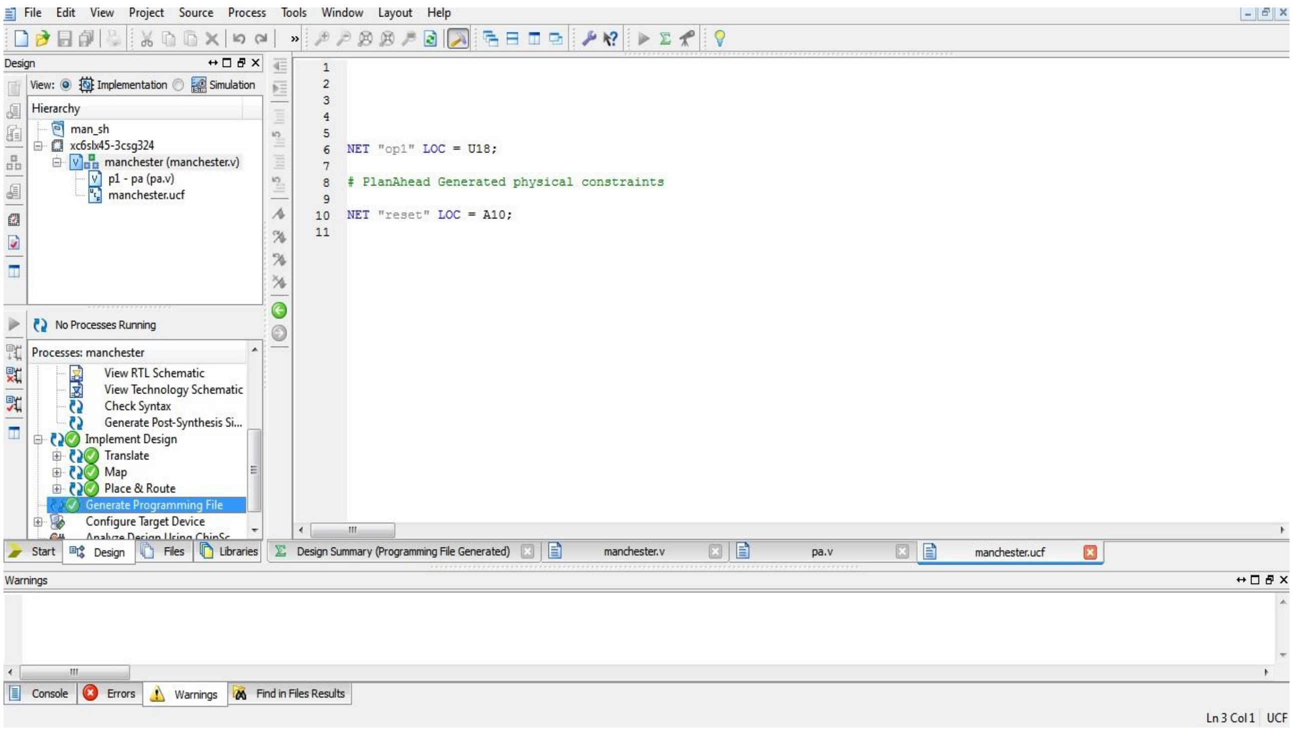
Figure 3.11:Test Bench Code for Manchester

3.5 Implementation on FPGA kit



Name	Direction	Neg Diff Pair	Site	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM	OUT_TERM
All ports (3)														
Scalar ports (3)														
clk	Input		L15	✓	1	default (LVCMOS25)			12	SLOW	NONE	NONE	NONE	
op1	Output		U18	✓	1	default (LVCMOS25)	2.5		12	SLOW	NONE	FP_VTT_50		NONE
reset	Input		A10	✓	0	default (LVCMOS25)			12	SLOW	NONE	NONE	NONE	

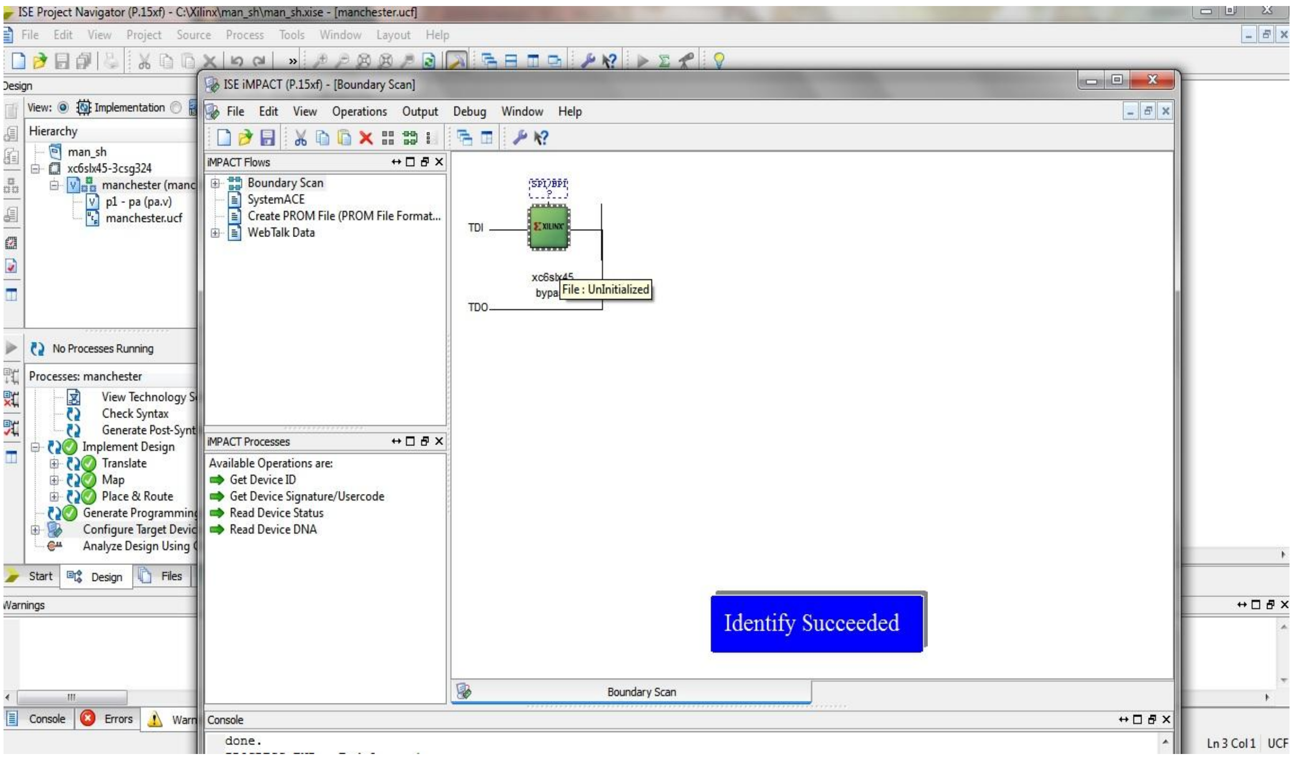
Figure 3.12:Pin Planning on FPGA



File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
man_sh
xc6slx45-3csg324
manchester (manchester.v)
p1 - pa (pa.v)
manchester.ucf
No Processes Running
Processes: manchester
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Si...
Implement Design
Translate
Map
Place & Route
Generate Programming File
Configure Target Device
Start Design Files Libraries
Design Summary (Programming File Generated)
manchester.v
pa.v
manchester.ucf
Warnings
Console Errors Warnings Find in Files Results
Ln 3 Col 1 UCF

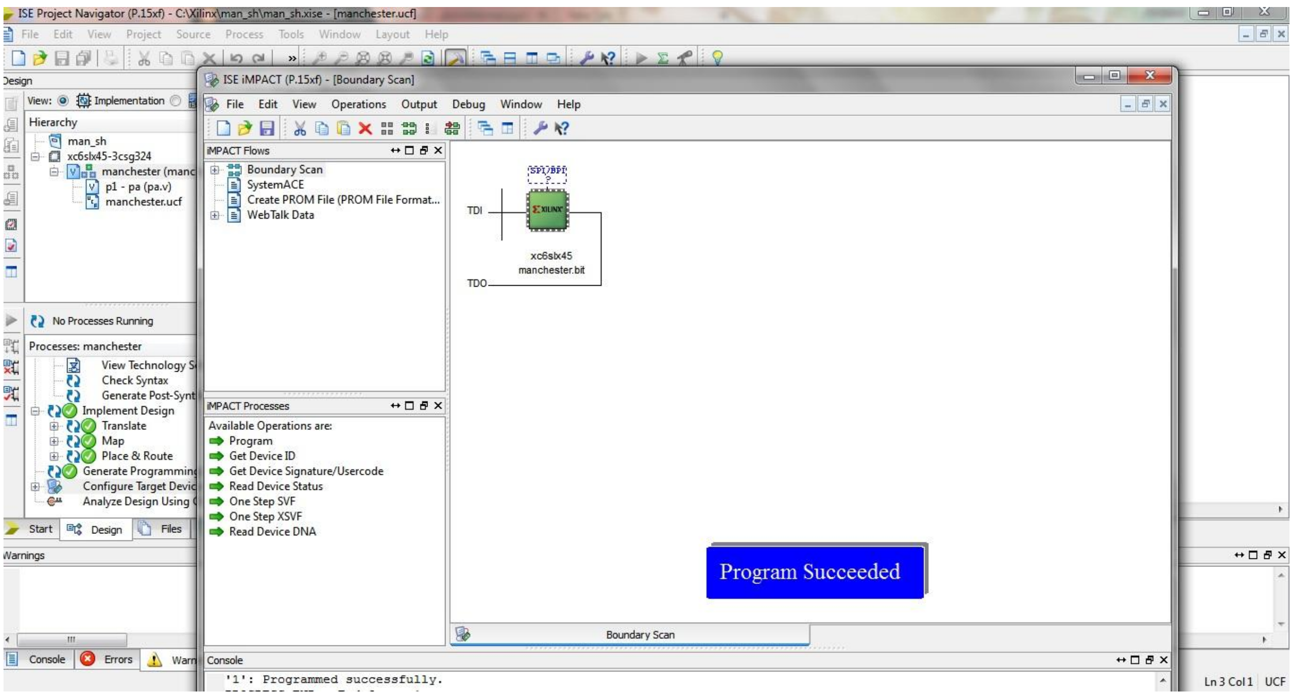
```
NET "op1" LOC = U18;

# PlanAhead Generated physical constraints

NET "reset" LOC = A10;
```

Figure 3.13:UCF File Generation


ISE Project Navigator (P.15xf) - C:\Xilinx\man_sh\man_sh.xise - [manchester.ucf]
ISE iMPACT (P.15xf) - [Boundary Scan]
File Edit View Operations Output Debug Window Help
iMPACT Flows
Boundary Scan
SystemACE
Create PROM File (PROM File Format...
WebTalk Data
iMPACT Processes
Available Operations are:
Get Device ID
Get Device Signature/Usercode
Read Device Status
Read Device DNA
TDI
TDO
xc6slx45
bypa
File : UnInitialized
Identify Succeeded
Boundary Scan
Console

```
done.
```

Figure 3.14:Boundary Scan


ISE Project Navigator (P.15xf) - C:\Xilinx\man_sh\man_sh.xise - [manchester.ucf]
ISE iMPACT (P.15xf) - [Boundary Scan]
File Edit View Operations Output Debug Window Help
iMPACT Flows
Boundary Scan
SystemACE
Create PROM File (PROM File Format...
WebTalk Data
iMPACT Processes
Available Operations are:
Program
Get Device ID
Get Device Signature/Usercode
Read Device Status
One Step SVF
One Step XSVF
Read Device DNA
TDI
TDO
xc6slx45
manchester.bit
Program Succeeded
Boundary Scan
Console

```
'1': Programmed successfully.
```

Figure 3.15:Programming File Burned to FPGA

CHAPTER-4

Results

4.1 Results from Matlab Implementation

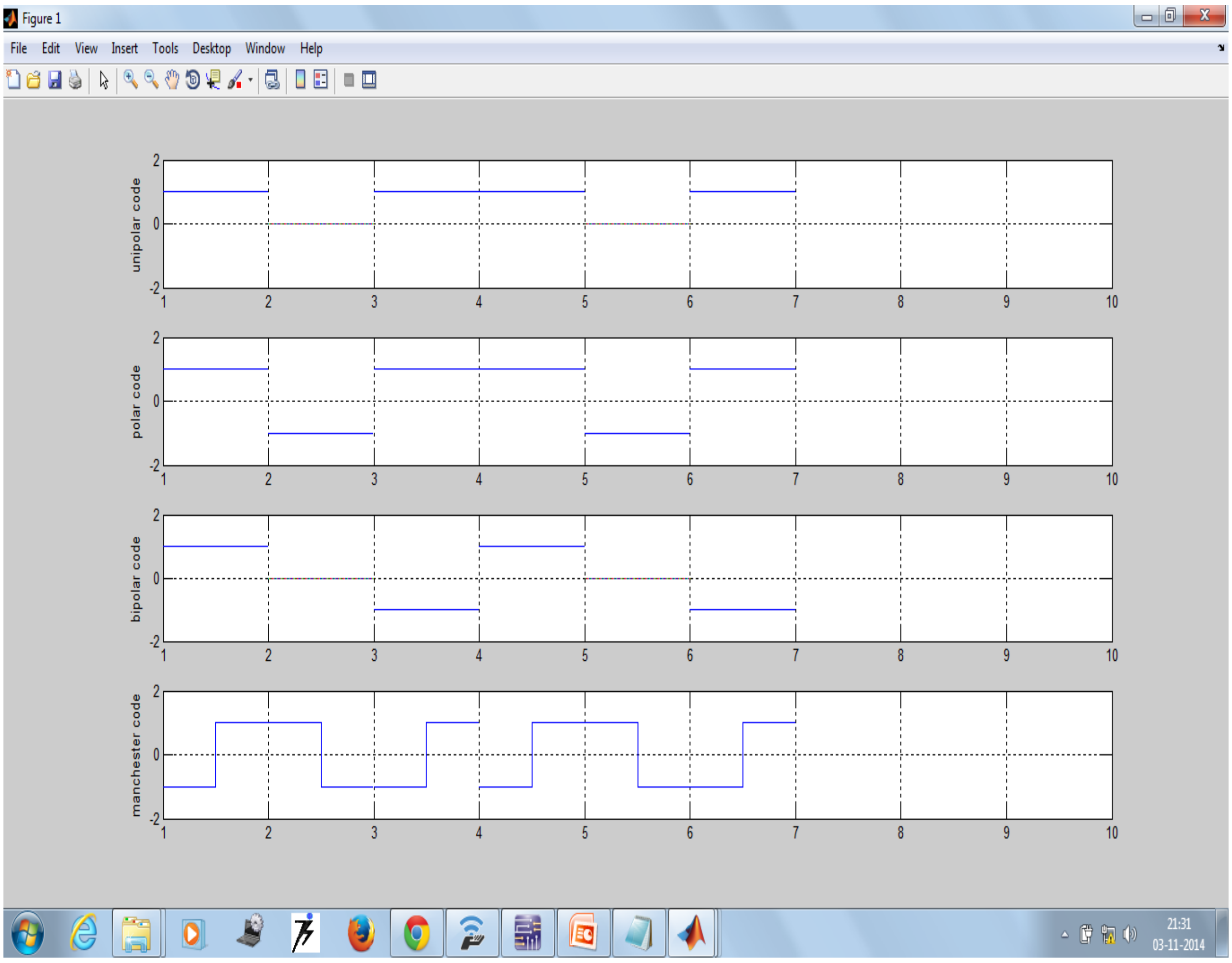
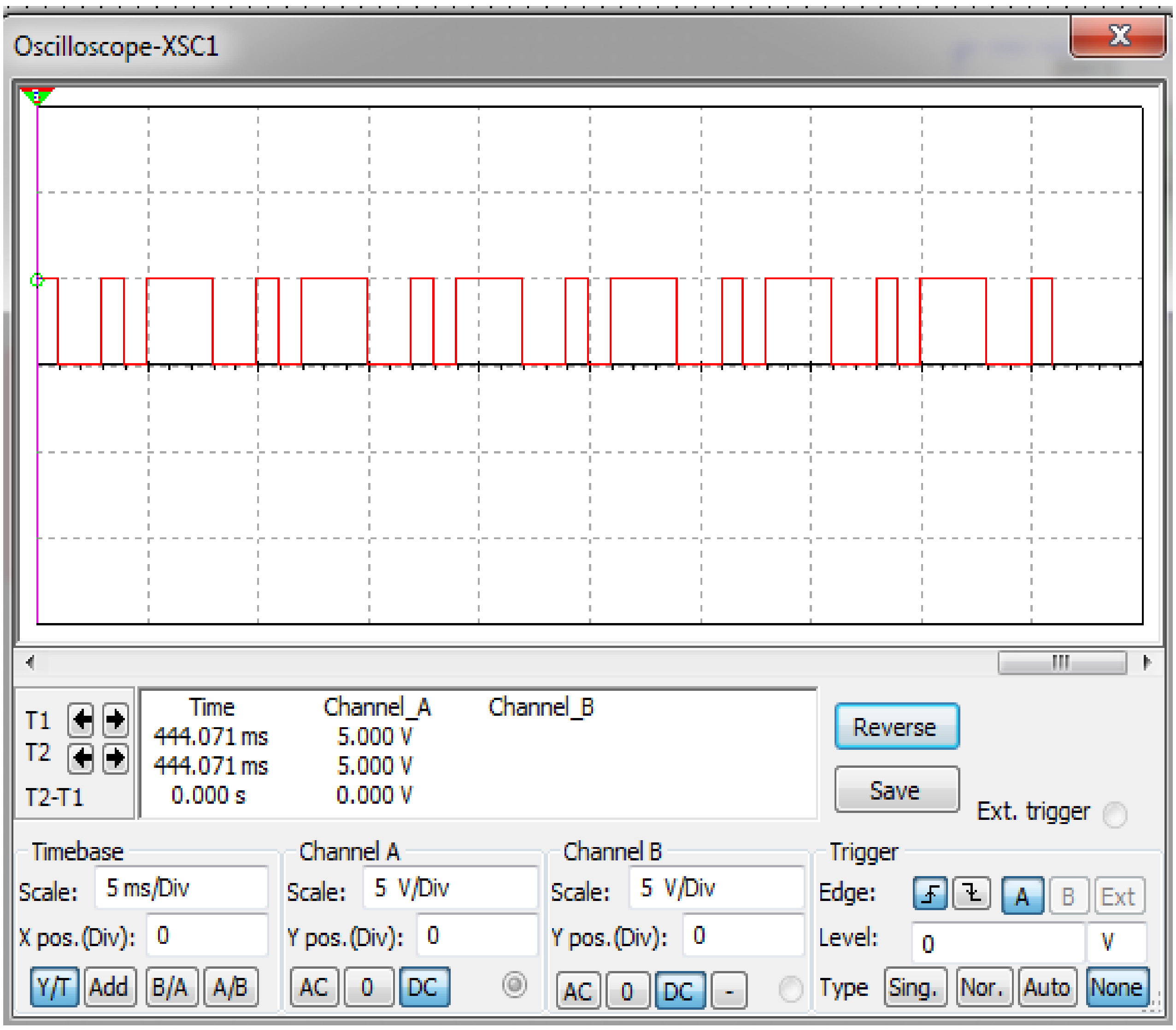


Figure 4.1:Output Waveform of Matlab

4.2 Results from Multisim implementation

i. Unipolar NRZ



	Time	Channel_A	Channel_B
T1	444.071 ms	5.000 V	
T2	444.071 ms	5.000 V	
T2-T1	0.000 s	0.000 V	

Ext. trigger

Timebase
Scale: 5 ms/Div
X pos.(Div): 0

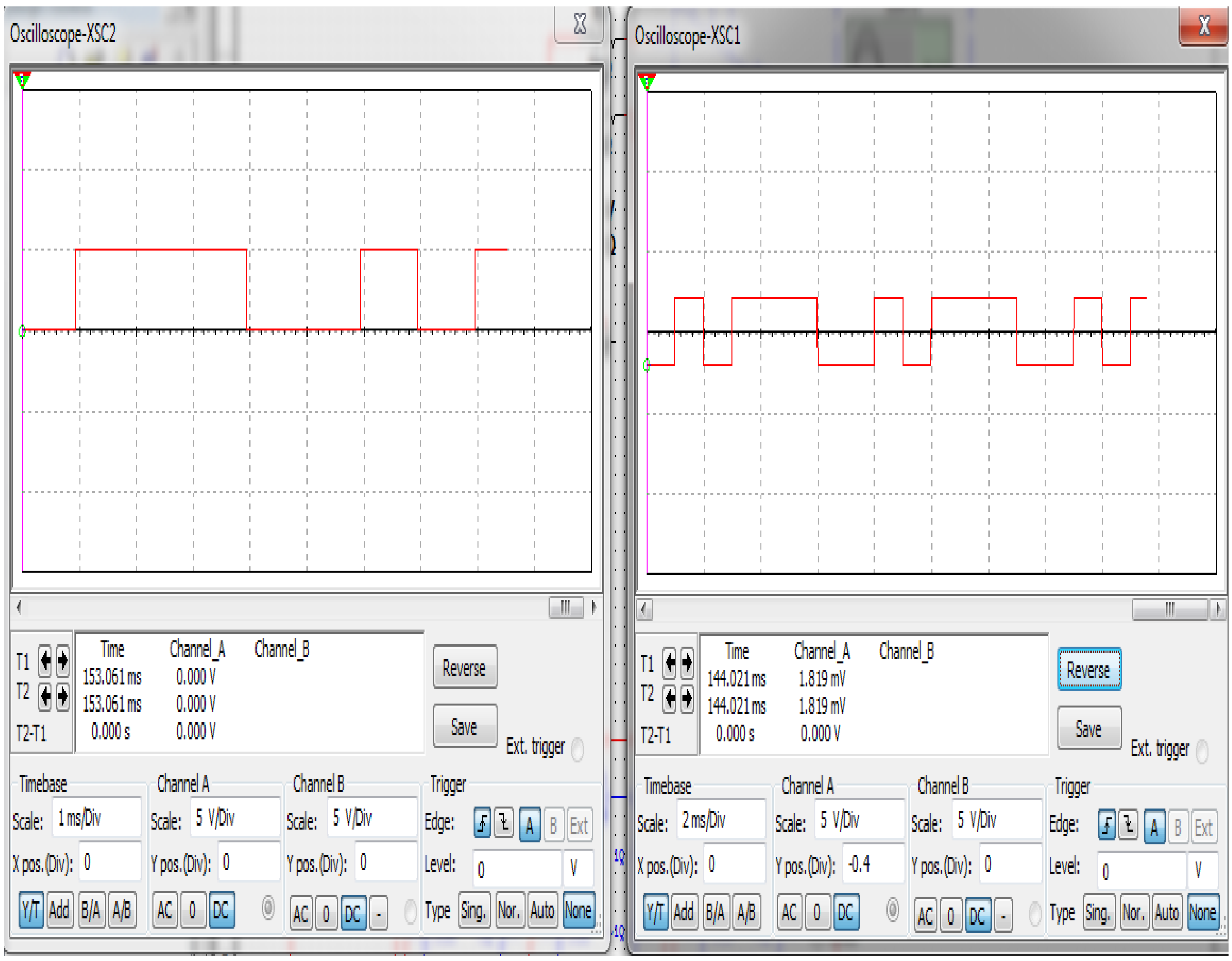
Channel A
Scale: 5 V/Div
Y pos.(Div): 0

Channel B
Scale: 5 V/Div
Y pos.(Div): 0

Trigger
Edge:
Level: 0 V
Type

Figure 4.2:Unipolar NRZ Waveform

ii. Polar NRZ


Oscilloscope-XSC2

	Time	Channel_A	Channel_B
T1	153.061 ms	0.000 V	
T2	153.061 ms	0.000 V	
T2-T1	0.000 s	0.000 V	

Timebase Scale: 1 ms/Div X pos.(Div): 0
Channel A Scale: 5 V/Div Y pos.(Div): 0
Channel B Scale: 5 V/Div Y pos.(Div): 0
Trigger Level: 0 V

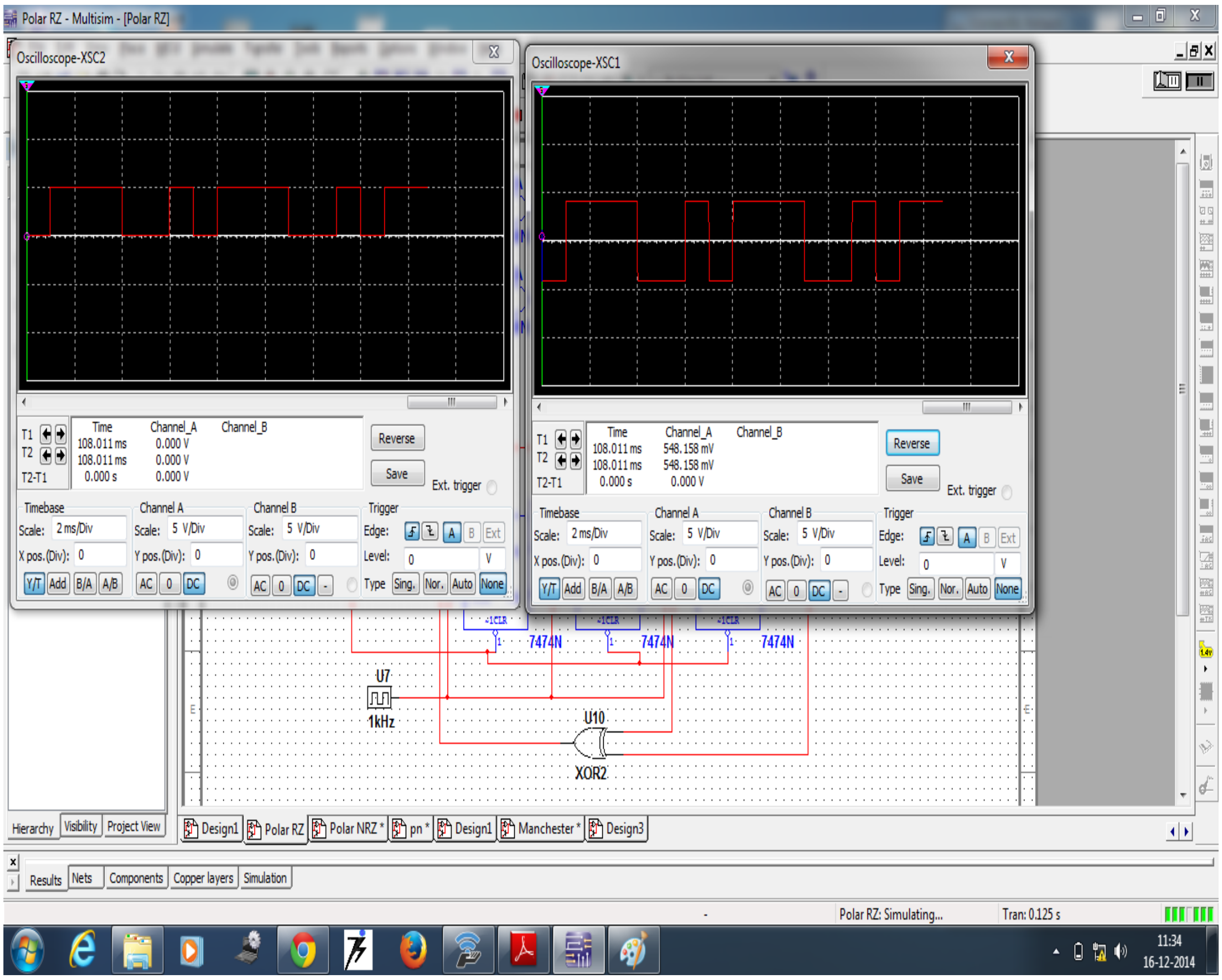
Oscilloscope-XSC1

	Time	Channel_A	Channel_B
T1	144.021 ms	1.819 mV	
T2	144.021 ms	1.819 mV	
T2-T1	0.000 s	0.000 V	

Timebase Scale: 2 ms/Div X pos.(Div): 0
Channel A Scale: 5 V/Div Y pos.(Div): -0.4
Channel B Scale: 5 V/Div Y pos.(Div): 0
Trigger Level: 0 V

Figure 4.3:Polar NRZ Waveform

iii. Polar RZ



Oscilloscope-XSC2

```
       Time        Channel_A   Channel_B
T1     108.011 ms  0.000 V
T2     108.011 ms  0.000 V
T2-T1  0.000 s     0.000 V
```

Timebase: Scale: 2 ms/Div; X pos.(Div): 0; Y/T
Channel A: Scale: 5 V/Div; Y pos.(Div): 0; DC
Channel B: Scale: 5 V/Div; Y pos.(Div): 0; DC
Trigger: Edge: A; Level: 0 V; Type: None

Oscilloscope-XSC1

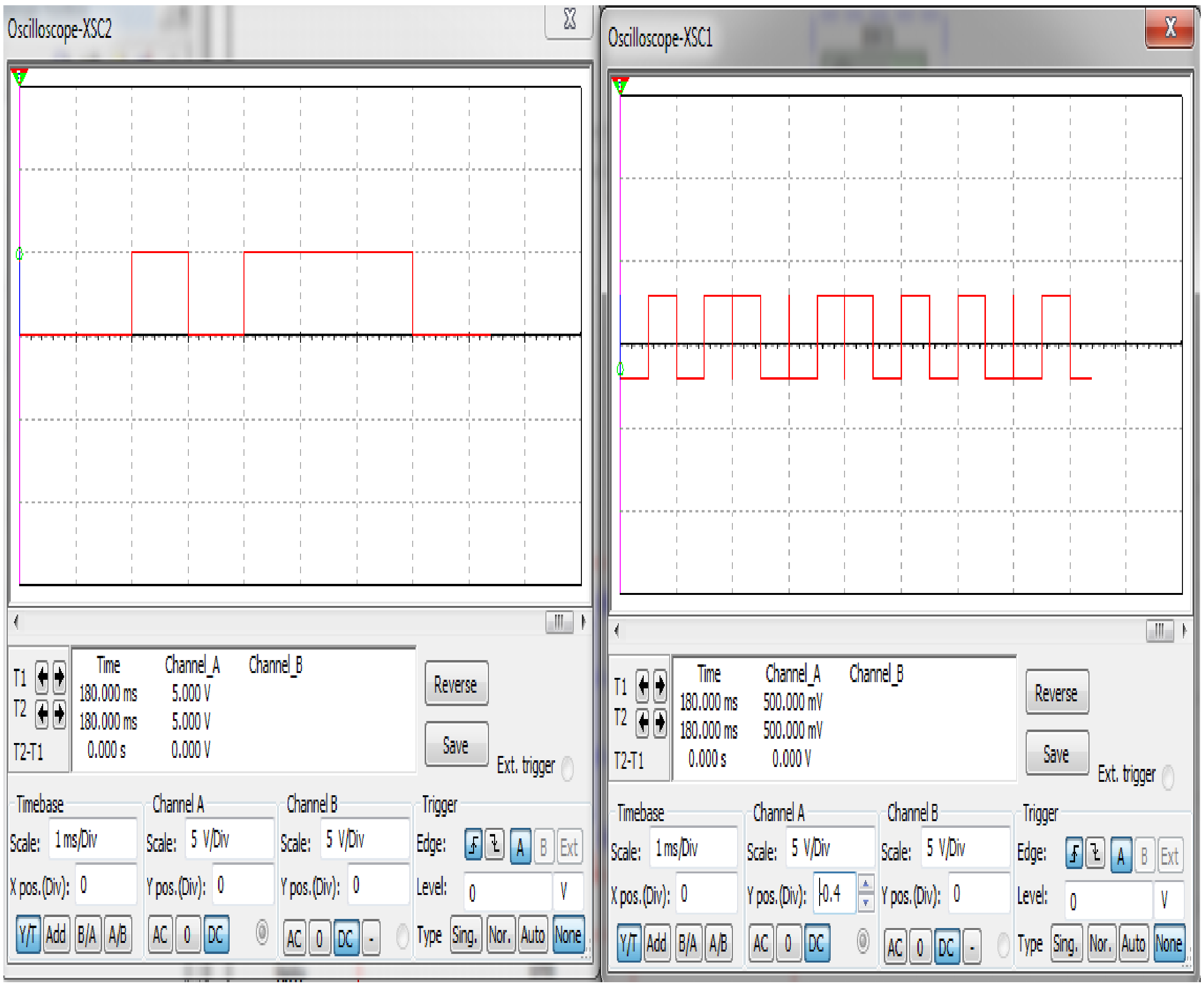
```
       Time        Channel_A   Channel_B
T1     108.011 ms  548.158 mV
T2     108.011 ms  548.158 mV
T2-T1  0.000 s     0.000 V
```

Timebase: Scale: 2 ms/Div; X pos.(Div): 0; Y/T
Channel A: Scale: 5 V/Div; Y pos.(Div): 0; DC
Channel B: Scale: 5 V/Div; Y pos.(Div): 0; DC
Trigger: Edge: A; Level: 0 V; Type: None

Figure 4.4:Polar RZ Waveform

•

iv. Manchester line code


Oscilloscope-XSC2

	Time	Channel_A	Channel_B
T1	180.000 ms	5.000 V	
T2	180.000 ms	5.000 V	
T2-T1	0.000 s	0.000 V	

Timebase Scale: 1 ms/Div X pos.(Div): 0
Channel A Scale: 5 V/Div Y pos.(Div): 0
Channel B Scale: 5 V/Div Y pos.(Div): 0
Trigger Level: 0 V

Oscilloscope-XSC1

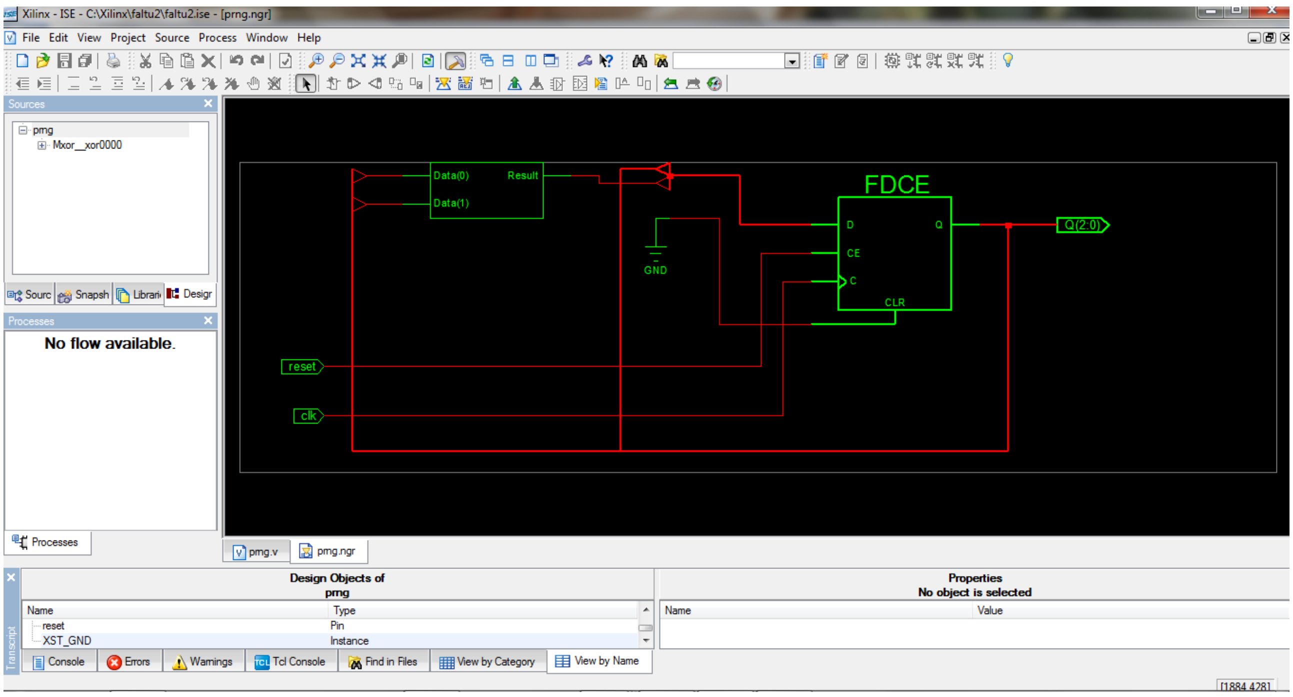
	Time	Channel_A	Channel_B
T1	180.000 ms	500.000 mV	
T2	180.000 ms	500.000 mV	
T2-T1	0.000 s	0.000 V	

Timebase Scale: 1 ms/Div X pos.(Div): 0
Channel A Scale: 5 V/Div Y pos.(Div): 0.4
Channel B Scale: 5 V/Div Y pos.(Div): 0
Trigger Level: 0 V

Figure 4.5:Manchester Waveform

4.3 Results from verilog Implementation

i. Pseudo Random number Generator



No flow available.

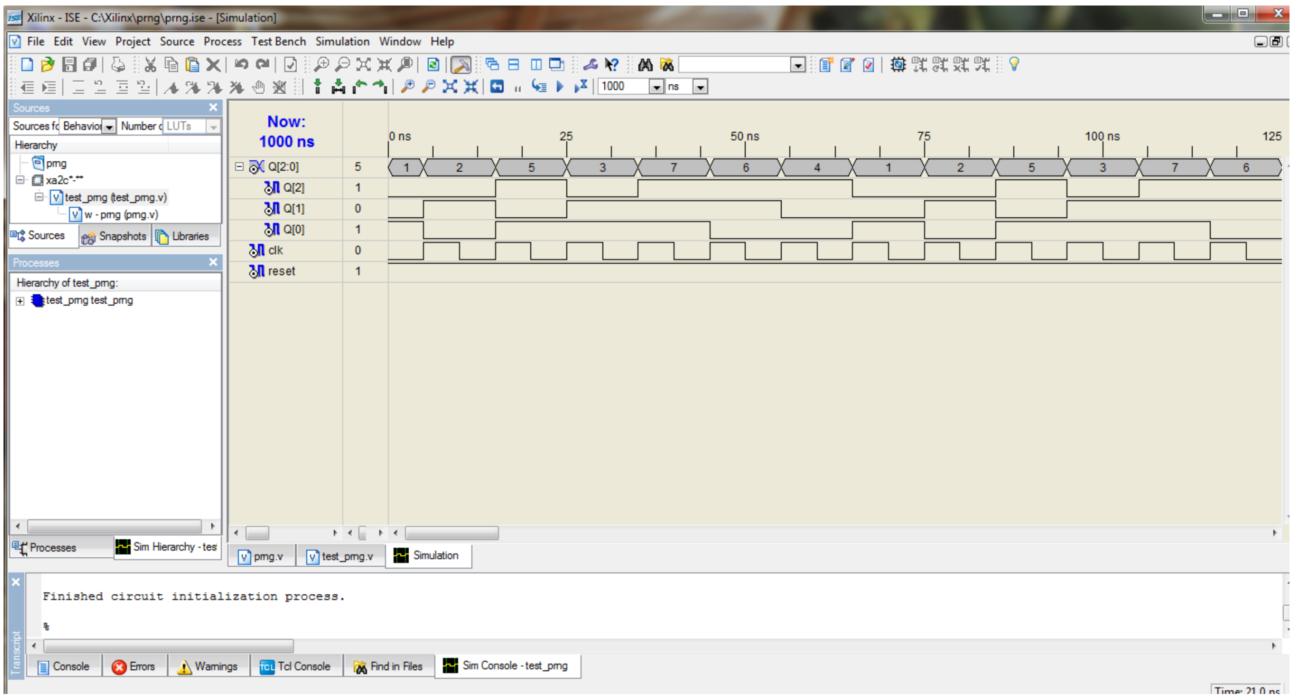
Design Objects of prng

Name	Type
reset	Pin
XST_GND	Instance

Properties
No object is selected

Name	Value

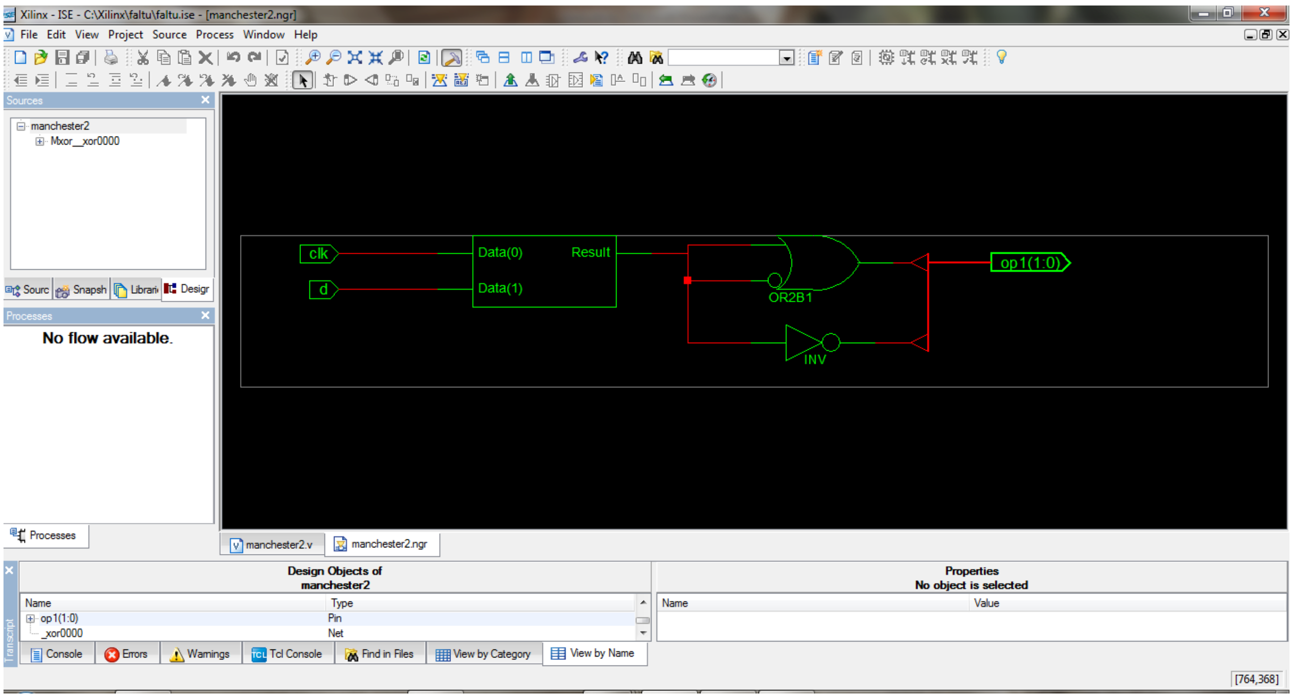
Figure 4.6:RTL Schematic of PRNG



```
Finished circuit initialization process.
%
```

Figure 4.7:Testbench Waveform of PRNG

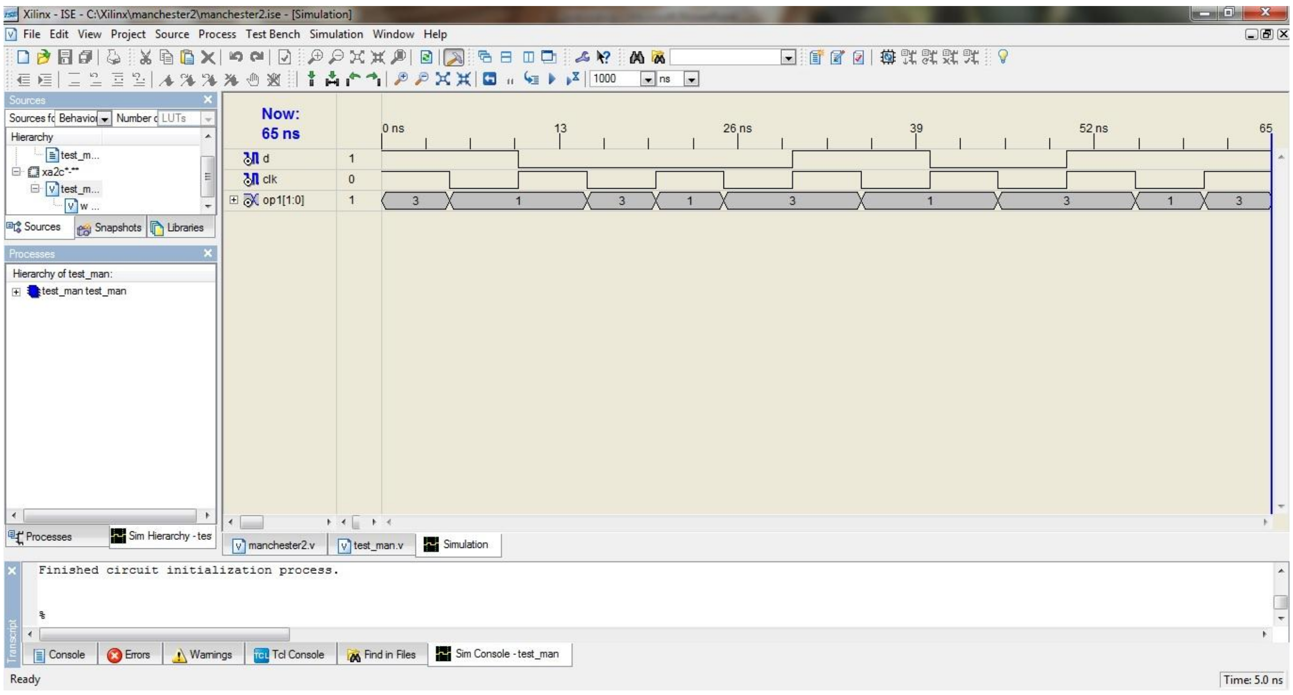
ii. Manchester Line code


Xilinx - ISE - C:\Xilinx\faltu\faltu.ise - [manchester2.ngr]
File Edit View Project Source Process Window Help
manchester2
Mxor__xor0000
clk
d
Data(0)
Data(1)
Result
OR2B1
INV
op1(1:0)
manchester2.v
manchester2.ngr
Design Objects of manchester2
Properties
No object is selected

No flow available.

Name	Type
op1(1:0)	Pin
_xor0000	Net

Figure 4.8:RTL Schematic of Manchester



```
Finished circuit initialization process.

%
```

Figure 4.9:Testbench Waveform of Manchester

4.4 Results from FPGA kit Implementation

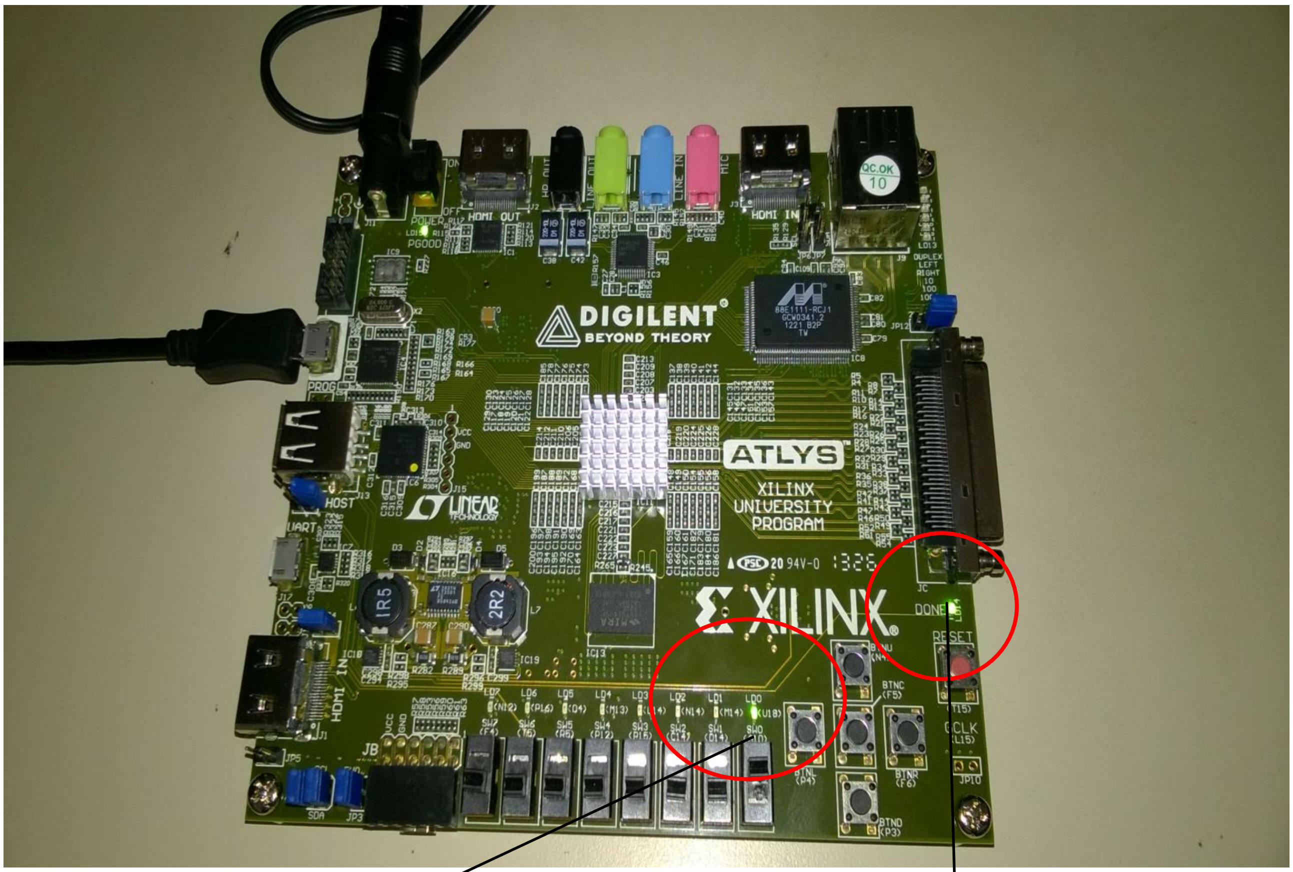


Figure 4.10: Output on FPGA Kit

CHAPTER 5

CONCLUSION

In this thesis we have mainly focused on the implementation of the line codes on different platforms. We have first implemented the line code on MATLAB using their mathematical modeling and then analyzed the output waveforms. Further we have implemented the and worked on the circuits and analyzed their output first on simulation tools and then on the hardware. The results that we obtained were output waveforms and then we have verified the results. In the next step we worked on the hardware modeling of these line codes using hardware descriptive language using the basic logic of the line codes. The hardware schematic generated were verified with the circuits that we had already simulated. Also the test bench waveforms were similar to that of the waveforms from the simulation. Lastly we implemented the verilog code on FPGA kit so as to have a generalized hardware for implementation of different line codes. The results have been successfully verified.

References

[1] Cisco Visual Networking Index (VNI) Global IP Traffic Forecast, 2010 –2015.

[2] Cattermole, K. W. "Invited paper Principles of digital line coding." *International Journal of Electronics* 55, no. 1 (1983): 3-33.

[3] Fox, Trevor R. "Decoder for Manchester encoded data." U.S. Patent 4,688,232, issued August 18, 1987.

[4] Gehlot, Narayan L. "System and method for generating NRZ signals from RZ signals in communications networks." U.S. Patent 6,404,819, issued June 11, 2002.

[5] http://in.mathworks.com/products/matlab/ (Accessed September 2014)

[6] Kamen, Edward, and Bonnie Heck. *Fundamentals of Signals and Systems: With MATLAB Examples*. Prentice Hall PTR, 2000.

[7] McKinley, Philip K., and Christian Trefftz. "Multisim: A simulation tool for the study of large-scale multiprocessors." In *In Proceedings of the 1993 International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Networks (MASCOTS*. 1993.

[8] http://www.ni.com/multisim/ (Accessed November 2014)

[9] Navabi, Zainalabedin. *VHDL: Analysis and modeling of digital systems*. McGraw-Hill, Inc., 1997.

[10] Christen, Ernst, and Kenneth Bakalar. "VHDL-AMS-a hardware description language for analog and mixed-signal applications." *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on* 46, no. 10 (1999): 1263-1272.

[11] “FPGA and Structured ASICs-Low Risk SoC for the masses”, Altera Corporation

[12] http://www.tc.etc.upt.ro/teaching/dc-pi/Course3.pdf (Accessed March 2015)

[13] Bush, Ian, Martyn Guest, Miles Deegan, Igor Kozin, and Christine Kitchen. *An overview of FPGAs and FPGA programming: Initial experiences at Daresbury*. Council for the Central Laboratory of the Research Councils, 2006.

[14] Bacon, David F., Rodric Rabbah, and Sunil Shukla. "FPGA Programming for the Masses." *Communications of the ACM* 56, no. 4 (2013): 56-63.

[15] Mayer-Lindenberg, Fritz. "High-level FPGA programming through mapping process networks to FPGA resources." In *Reconfigurable Computing and FPGAs, 2009. ReConFig'09. International Conference on*, pp. 302-307. IEEE, 2009.

[16] http://digilentinc.com/Products/Detail.cfm?NavPath=2,400,836&Prod=ATLYS (Accessed April 2015)

[17] http://digilentinc.com/Products/Detail.cfm?NavPath=2,729,969&Prod=GETTING-STARTED-CHIPKIT (Accessed April 2015)

[18] Janocha, H., D. Pesotski, and K. Kuhnen. "FPGA-based compensator of hysteretic actuator nonlinearities for highly dynamic applications." *Signal* 1, no. 2 (2008): 3.

[19] Andrews, David, Douglas Niehaus, Razali Jidin, Michael Finley, Wesley Peck, Michael Frisbie, Jorge Ortiz, Ed Komp, and Peter Ashenden. "Programming models for hybrid FPGA-CPU computational components: a missing link."*IEEE Micro* 4 (2004): 42-53.

[20] Klingman, Ed. "FPGA programming step by step." *Embedded Systems Programming* 17, no. 4 (2004): 29-37.