SAVEETHA SCHOOL OF ENGINEERING
SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES
DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Engineer to Excel

Laboratory Manual

For

ECA11

VLSI Design Laboratory

E.C.E.Department

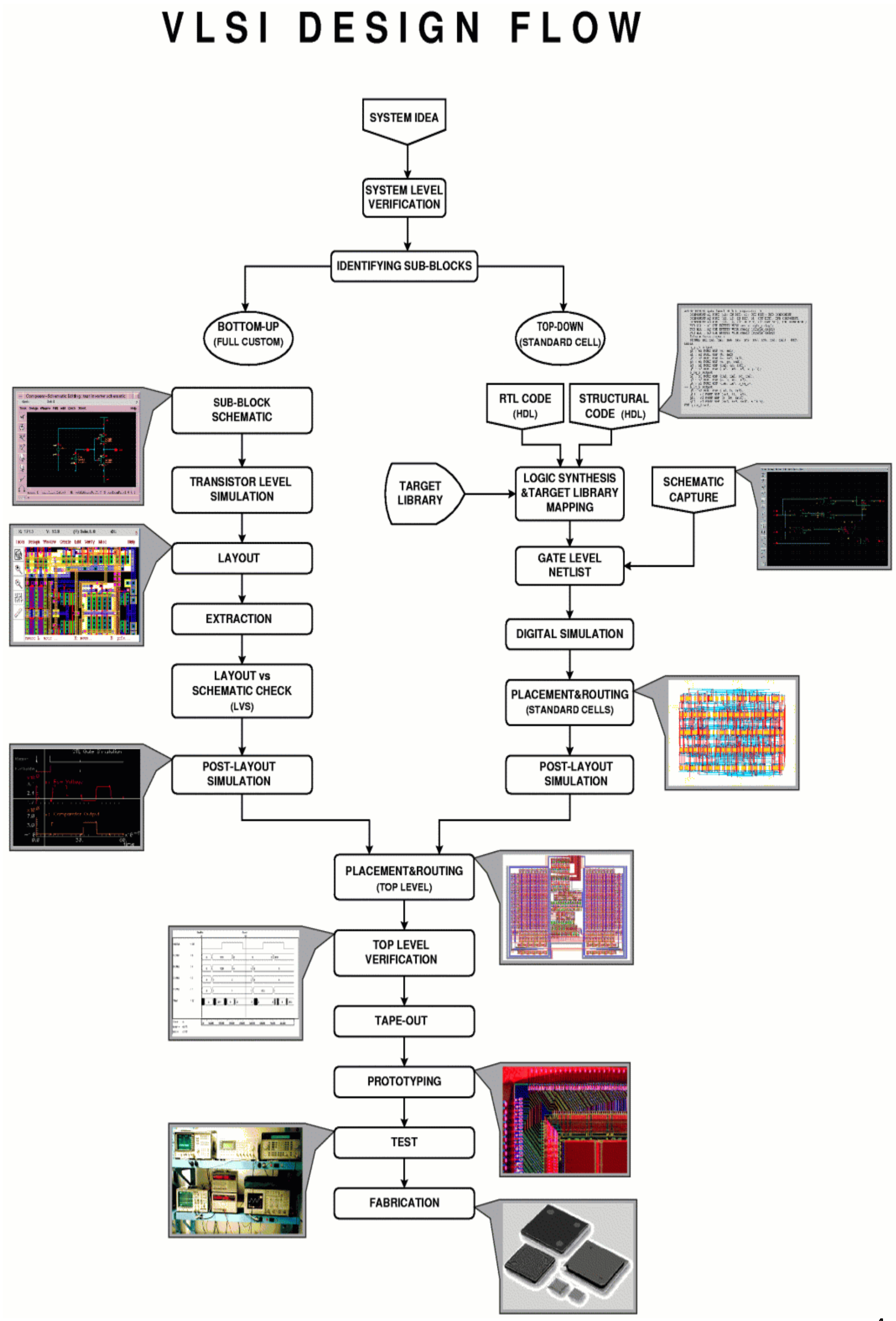
SAVEETHA SCHOOL OF ENGINEERING
SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES
DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

ECA 11 VLSI DESIGN LABORATORY

NAME	**:**
REG. No	**:**
YEAR	**:**
SEMESTER	**:**
BRANCH	**:**
LAB INCHARGE	**:**
MARKS	**:**

LIST OF EXPERIMENTS:

1. Study of Xilinx electronic design automation (EDA) tool.
2. Study of Development tool for FPGA
3. Simulate and verify the basic logic gates using Verilog code in Xilinx with FPGA trainer kit
4. Design and simulation of Half Adder using Xilinx. with FPGA trainer kit
5. Design and simulation Full Adder using Xilinx. with FPGA trainer kit
6. Design and simulation of Half Subtractor using Xilinx with FPGA trainer kit
7. Design and simulation of Full subtractor using Xilinx with FPGA trainer kit
8. Design and simulation of a 4-bit Ripple carry adder using Xilinx.
9. Design and simulation of T Flip Flop using Xilinx.
10. Design and simulation of Delay Flip Flop using Xilinx.
11. Design and simulation of 4:1 Multiplexer using Xilinx.
12. Design and simulation of 1:4 Demultiplexer using Xilinx.
13. Design and simulation of 8:3 Encoder using Xilinx.
14. Design and simulation of 3:8 Decoder using Xilinx.
15. Simulation and verification of 16 bit Arithmetic Logic Unit using Xilinx.
16. Simulate and verify up-down counter using Xilinx
17. Design and simulation of magnitude comparator using Xilinx.
18. Design and simulation of 4-bit Ripple counter using Xilinx.
19. Design and simulation of priority encoder using Xilinx.
20. Testing of LEDs and Switches on the FPGA board.
21. Testing of buzzer on FPGA board.


VLSI DESIGN FLOW
SYSTEM IDEA
SYSTEM LEVEL VERIFICATION
IDENTIFYING SUB-BLOCKS
BOTTOM-UP (FULL CUSTOM)
TOP-DOWN (STANDARD CELL)
SUB-BLOCK SCHEMATIC
TRANSISTOR LEVEL SIMULATION
LAYOUT
EXTRACTION
LAYOUT vs SCHEMATIC CHECK (LVS)
POST-LAYOUT SIMULATION
RTL CODE (HDL)
STRUCTURAL CODE (HDL)
TARGET LIBRARY
LOGIC SYNTHESIS &TARGET LIBRARY MAPPING
SCHEMATIC CAPTURE
GATE LEVEL NETLIST
DIGITAL SIMULATION
PLACEMENT&ROUTING (STANDARD CELLS)
POST-LAYOUT SIMULATION
PLACEMENT&ROUTING (TOP LEVEL)
TOP LEVEL VERIFICATION
TAPE-OUT
PROTOTYPING
TEST
FABRICATION

Expt.No: 1 **STUDY OF SIMULATION TOOLS**

AIM: To study the Simulation tools, the Synthesis tools, the Place and Root and Back annotation for FPGAs.

THEORY:

STARTING THE ISE SOFTWARE

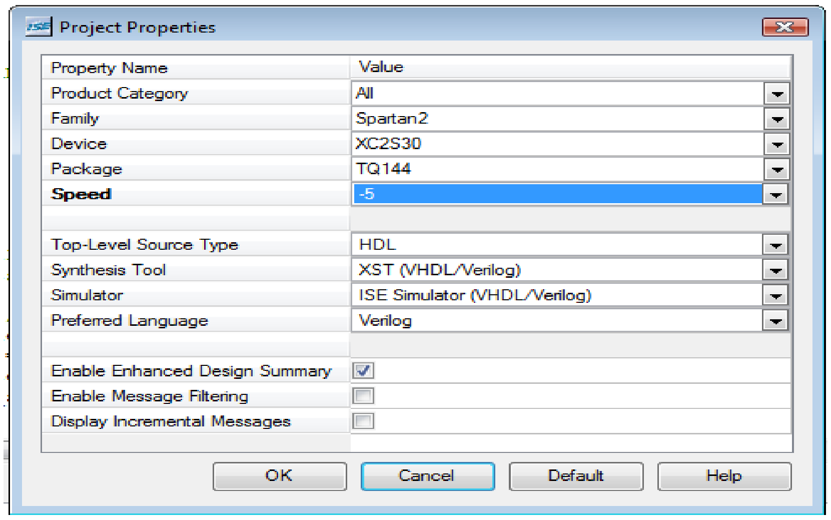
To start ISE, double-click the desktop icon,

or start ISE from the Start menu by selecting:
Start → All Programs → Xilinx ISE 10.1→ Project Navigator

CREATE A NEW PROJECT

Create a new ISE project which will target the FPGA device on the Spartan-3 Startup Kit demo board.

1. To create a new project:
2. Select **File** > **New Project...** The New Project Wizard appears.
3. Type **tutorial** in the Project Name field.
4. Enter or browse to a location (directory path) for the new project. A tutorial subdirectory is created automatically.
5. Verify that **HDL** is selected from the Top-Level Source Type list.
6. Click **Next** to move to the device properties page.
7. Fill in the properties in the table as shown below:



Property Name	Value
Product Category	All
Family	Spartan2
Device	XC2S30
Package	TQ144
Speed	-5
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISE Simulator (VHDL/Verilog)
Preferred Language	Verilog
Enable Enhanced Design Summary	✓
Enable Message Filtering	☐
Display Incremental Messages	☐

8. Click **Next** to proceed to the Create New Source window in the New Project Wizard. At the end of the next section, your new project will be complete

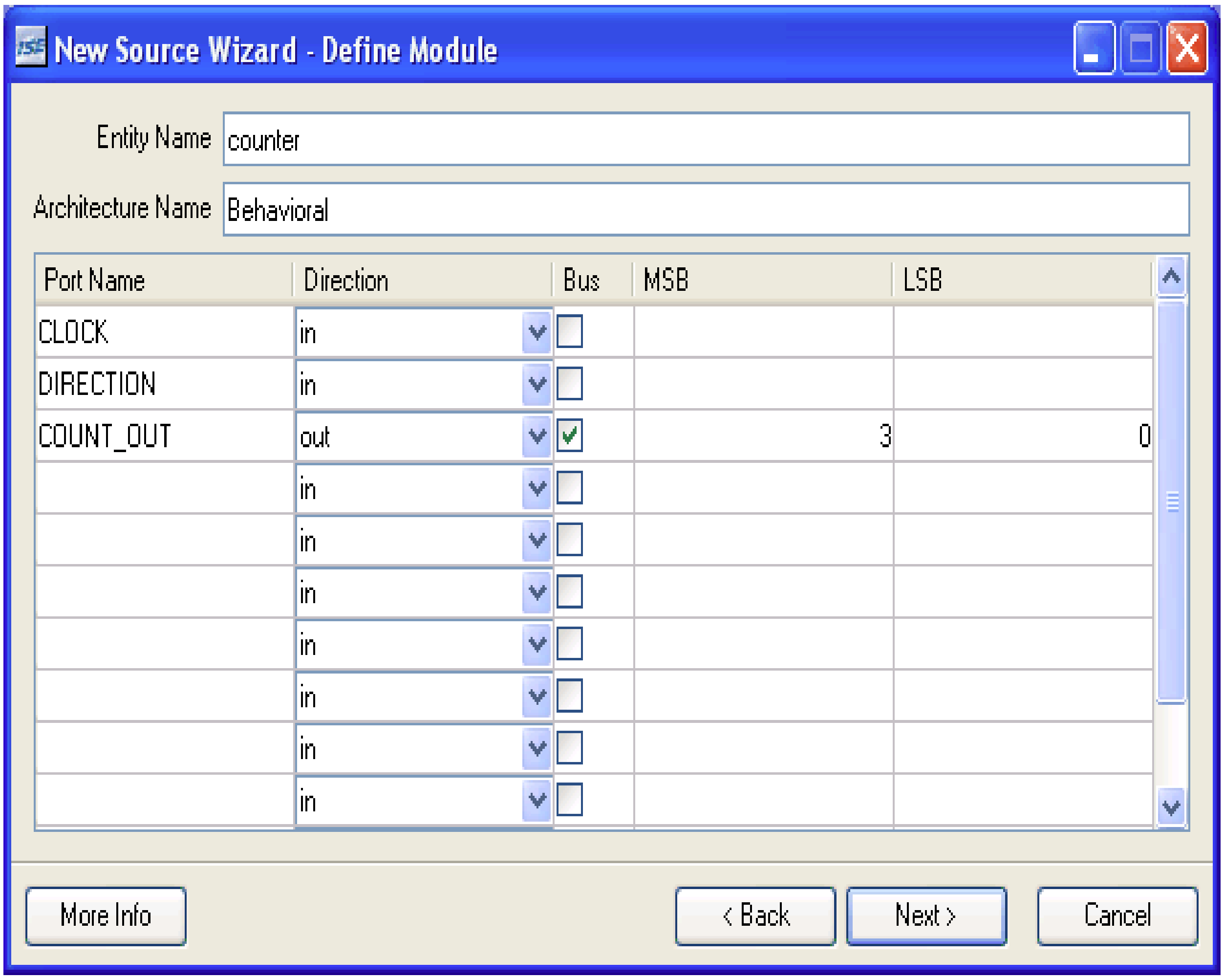
CREATE AN HDL SOURCE:

In this section, you will create the top-level HDL file for your design. Determine the language that you wish to use for the tutorial. Then, continue either to the “Creating a VHDL Source” section below, or skip to the “Creating a Verilog Source” section. Creating a VHDL Source

Create a VHDL source file for the project as follows:

1. Click the **New Source** button in the New Project Wizard.
2. Select **VHDL Module** as the source type.
3. Type in the file name **counter**.
4. Verify that the **Add to project** checkbox is selected.
5. Click **Next**.
6. Declare the ports for the counter design by filling in the port information as shown below:
7. Click **Next**, then **Finish** in the New Source Wizard - Summary dialog box to complete the new source file template.
8. Click **Next**, then **Next**, then **Finish**.

The source file containing the entity/architecture pair displays in the Workspace, and the counter displays in the Source tab, as shown below:

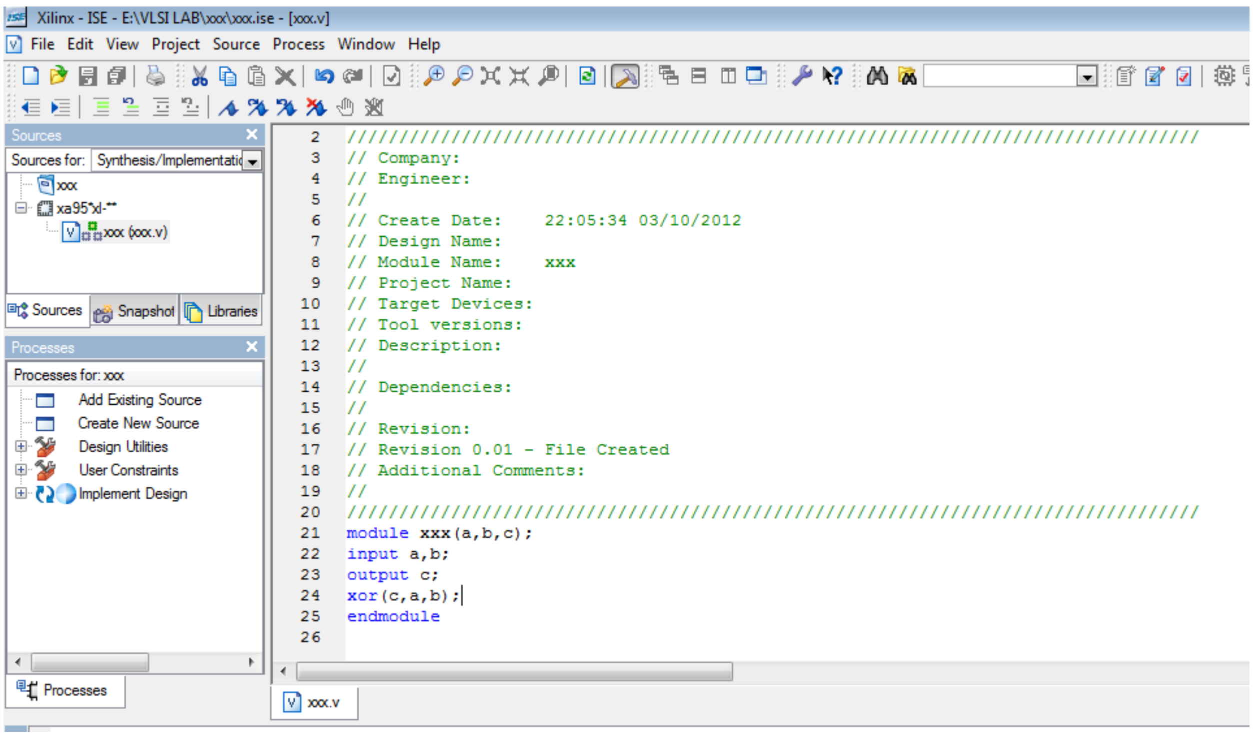


Entity Name: counter

Architecture Name: Behavioral

Port Name	Direction	Bus	MSB	LSB
CLOCK	in	☐		
DIRECTION	in	☐		
COUNT_OUT	out	✓	3	0
	in	☐		
	in	☐		
	in	☐		
	in	☐		
	in	☐		
	in	☐		
	in	☐		

7. Click **Next**, then **Finish** in the New Source Wizard - Summary dialog box to complete the new source file template.
8. Click **Next**, then **Next**, then **Finish**.



```
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:05:34 03/10/2012
// Design Name:
// Module Name:    xxx
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module xxx(a,b,c);
input a,b;
output c;
xor(c,a,b);
endmodule
```

CHECKING THE SYNTAX OF THE MODULE

When the source files are complete, check the syntax of the design to find errors and typos.

1. Verify that **Implementation** is selected from the drop-down list in the Sources window.
2. Select the **counter** design source in the Sources window to display the related processes in the Processes window.
3. Click the "+" next to the Synthesize-XST process to expand the process group.
4. Double-click the **Check Syntax** process.

Note: You must correct any errors found in your source files. You can check for errors in the Console tab of the Transcript window. If you continue without valid syntax, you will not be able to simulate or synthesize your design.

5. Close the HDL file.

DESIGN SIMULATION

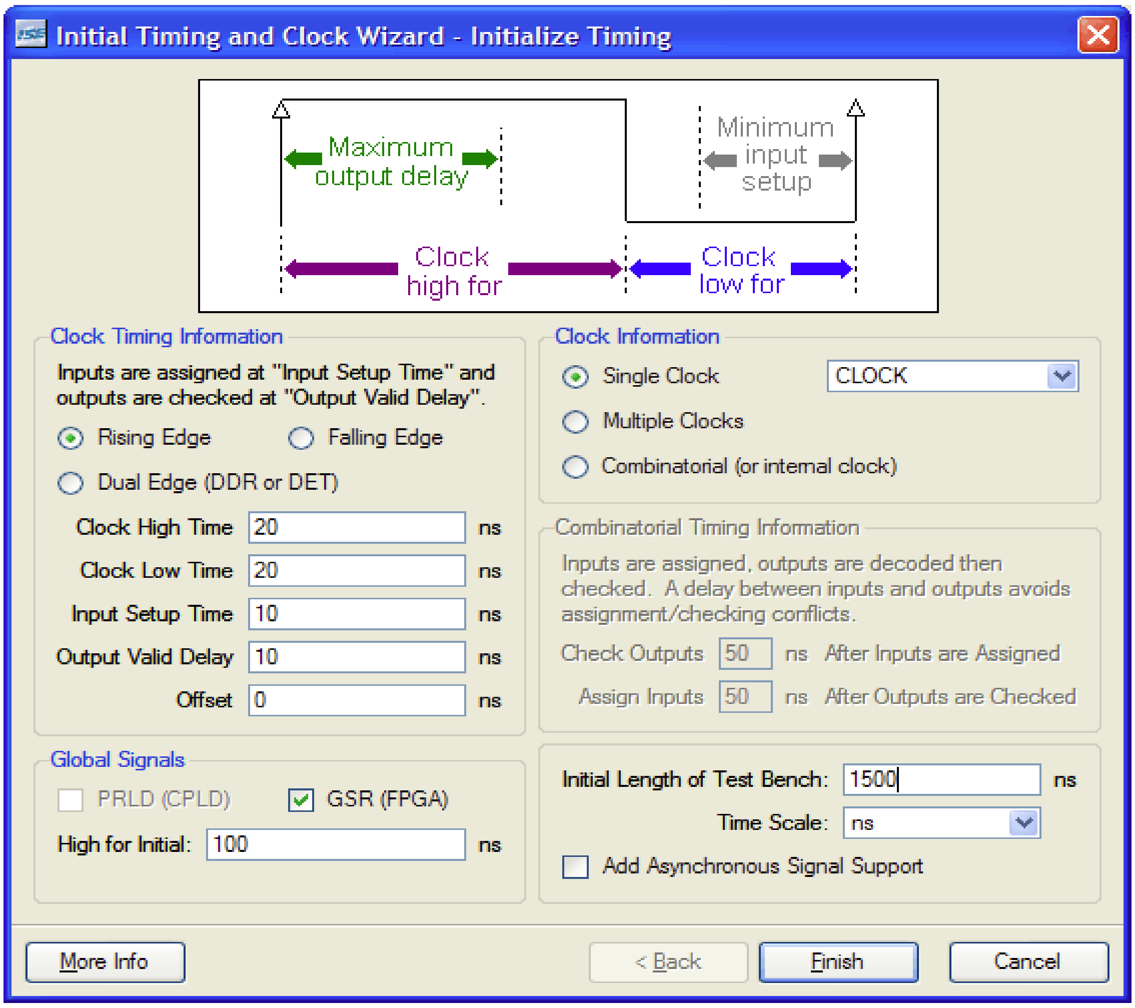
VERIFYING FUNCTIONALITY USING BEHAVIORAL SIMULATION

Create a test bench waveform containing input stimulus you can use to verify the functionality of the counter module. The test bench waveform is a graphical view of a test bench.

Create the test bench waveform as follows:

1. Select the **counter** HDL file in the Sources window.
2. Create a new test bench source by selecting **Project** → **New Source**.
3. In the New Source Wizard, select **Test Bench WaveForm** as the source type, and type **counter_tbw** in the File Name field.
4. Click **Next**.

5. The Associated Source page shows that you are associating the test bench waveform with the source file counter. Click **Next**.
6. The Summary page shows that the source will be added to the project, and it displays the source directory, type, and name. Click **Finish**.
7. You need to set the clock frequency, setup time and output delay times in the Initialize Timing dialog box before the test bench waveform editing window opens.
The requirements for this design are the following:
♦ The counter must operate correctly with an input clock frequency = 25 MHz.
♦ The DIRECTION input will be valid 10 ns before the rising edge of CLOCK.
♦ The output (COUNT_OUT) must be valid 10 ns after the rising edge of CLOCK.
The design requirements correspond with the values below.
Fill in the fields in the Initialize Timing dialog box with the following information:
♦ Clock High Time: **20** ns.
♦ Clock Low Time: **20** ns.
♦ Input Setup Time: **10** ns.
♦ Output Valid Delay: **10** ns.
♦ Offset: **0** ns.
♦ Global Signals: **GSR (FPGA)**
Note: When GSR(FPGA) is enabled, 100 ns. is added to the Offset value automatically.
♦ Initial Length of Test Bench: **1500** ns.



Clock Timing Information

Inputs are assigned at "Input Setup Time" and outputs are checked at "Output Valid Delay".

[✓ Rising Edge] Falling Edge

Dual Edge (DDR or DET)

Clock High Time 20 ns
Clock Low Time 20 ns
Input Setup Time 10 ns
Output Valid Delay 10 ns
Offset 0 ns

Clock Information

[✓ Single Clock] CLOCK

Multiple Clocks

Combinatorial (or internal clock)

Combinatorial Timing Information

Inputs are assigned, outputs are decoded then checked. A delay between inputs and outputs avoids assignment/checking conflicts.

Check Outputs 50 ns After Inputs are Assigned
Assign Inputs 50 ns After Outputs are Checked

Global Signals

PRLD (CPLD) [✓ GSR (FPGA)]

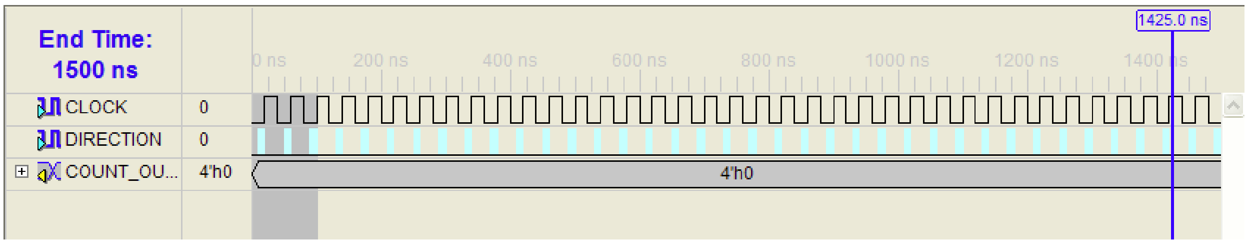
High for Initial: 100 ns

Initial Length of Test Bench: 1500 ns

Time Scale: ns

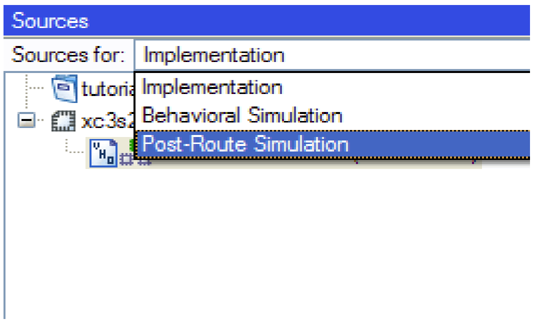
Add Asynchronous Signal Support

8. Click **Finish** to complete the timing initialization.
9. The blue shaded areas that precede the rising edge of the CLOCK correspond to the Input Setup Time in the Initialize Timing dialog box. Toggle the DIRECTION port to define the input stimulus for the counter design as follows:
♦ Click on the blue cell at approximately the **300** ns to assert DIRECTION high so that the counter will count up.
♦ Click on the blue cell at approximately the **900** ns to assert DIRECTION low so that the counter will count down.



10. Save the waveform.

11. In the Sources window, select the **Behavioral Simulation** view to see that the test bench waveform file is automatically added to your project.

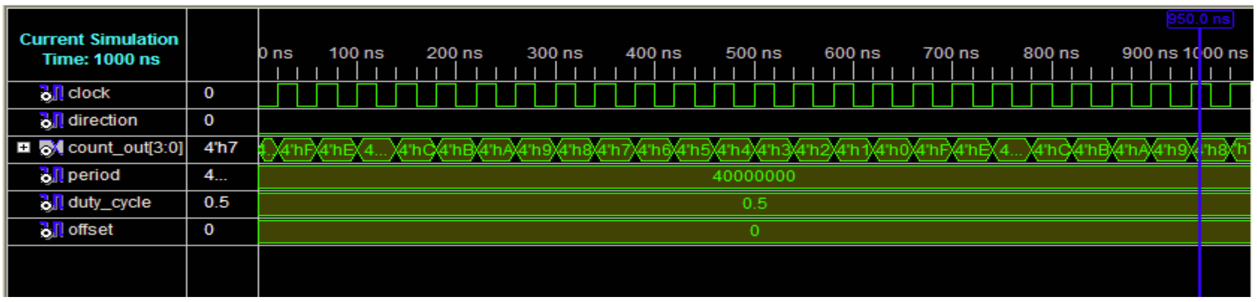


12. Close the test bench waveform.

SIMULATING DESIGN FUNCTIONALITY

Verify that the counter design functions as you expect by performing behavior simulation as follows:
1. Verify that **Behavioral Simulation** and **counter_tbw** are selected in the Sources window.
2. In the **Processes** tab, click the "+" to expand the Xilinx ISE Simulator process and double-click the **Simulate Behavioral Model** process. The ISE Simulator opens and runs the simulation to the end of the test bench.
3. To view your simulation results, select the **Simulation** tab and zoom in on the transitions.

Note: You can ignore any rows that start with **TX**.
4. Verify that the counter is counting up and down as expected.
5. Close the simulation view. If you are prompted with the following message, "You have an active simulation open. Are you sure you want to close it?", click **Yes** to continue. You have now completed simulation of your design using the ISE Simulator.

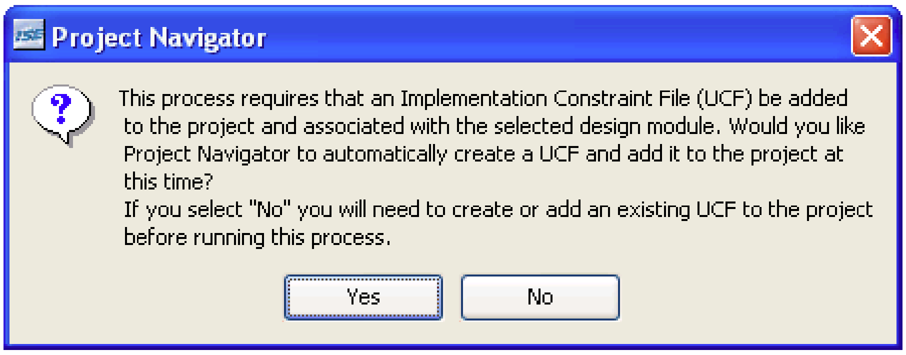


CREATE TIMING CONSTRAINTS

Specify the timing between the FPGA and its surrounding logic as well as the frequencythe design must operate at internal to the FPGA. The timing is specified by entering constraints that guide the placement and routing of the design. It is recommended that you enter global constraints. The clock period constraint specifies the clock frequency at which your design must operate inside the FPGA. The offset constraints specify when to expect valid data at the FPGA inputs and when valid data will be available at the FPGA outputs.

ENTERING TIMING CONSTRAINTS

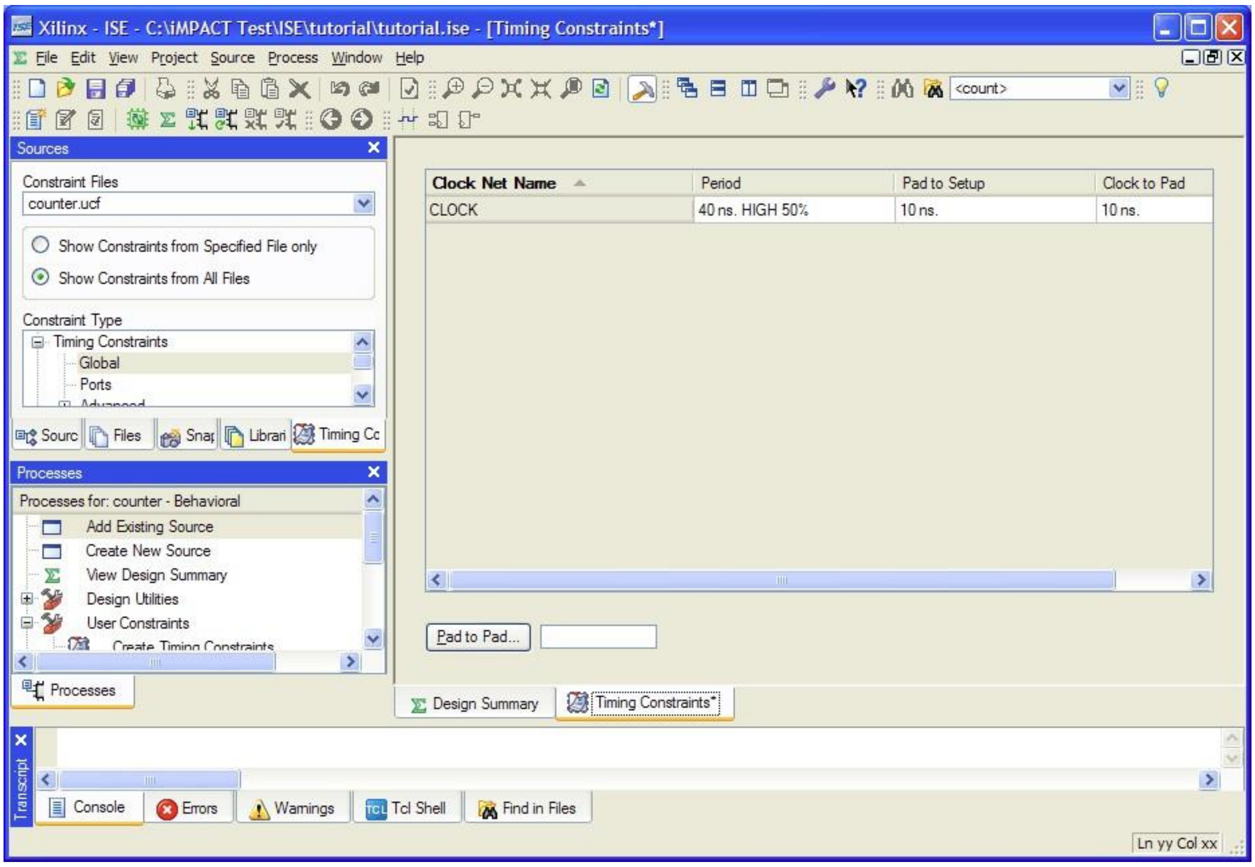
To constrain the design do the following:
1. Select **Implementation** from the drop-down list in the Sources window.
2. Select the **counter** HDL source file.
3. Click the "+" sign next to the User Constraints processes group, and double-click the **Create Timing Constraints** process.
ISE runs the Synthesis and Translate steps and automatically creates a User Constraints File (UCF). You will be prompted with the following message:


Project Navigator

This process requires that an Implementation Constraint File (UCF) be added to the project and associated with the selected design module. Would you like Project Navigator to automatically create a UCF and add it to the project at this time?
If you select "No" you will need to create or add an existing UCF to the project before running this process.

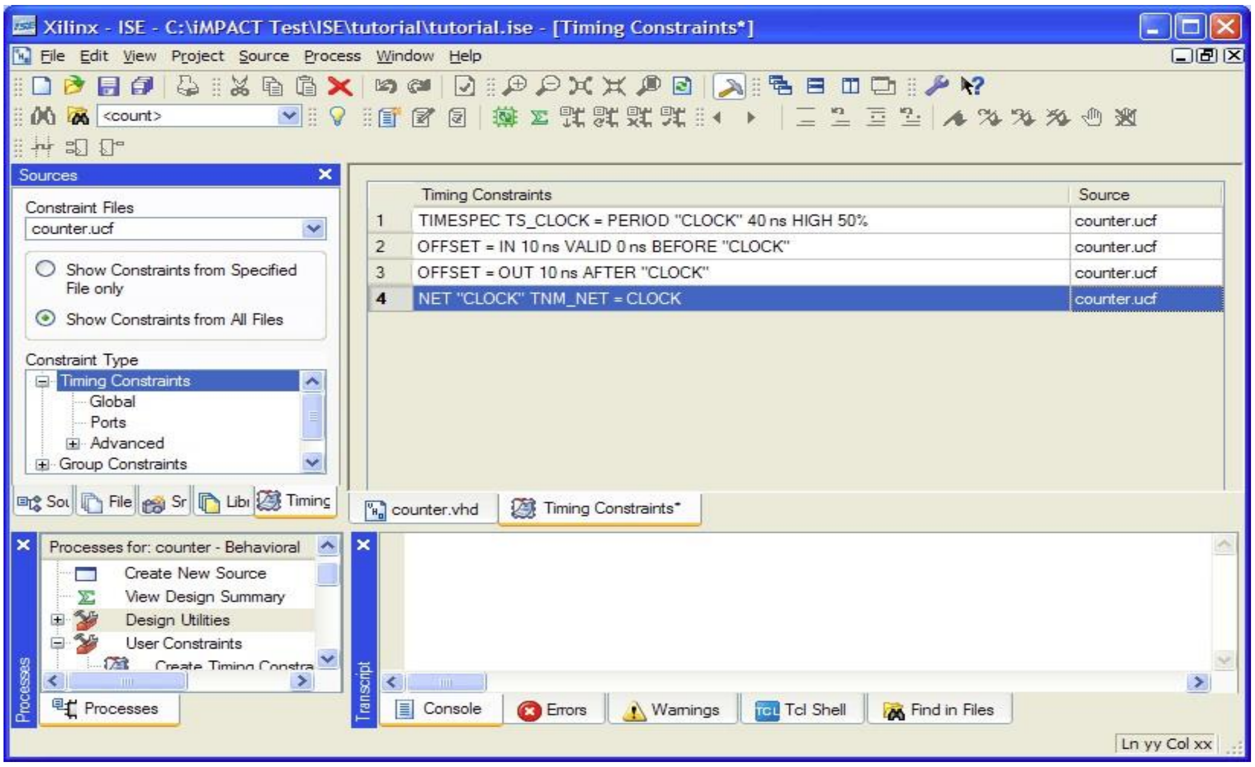
4. Click **Yes** to add the UCF file to your project.
The counter.ucf file is added to your project and is visible in the Sources window. The Xilinx Constraints Editor opens automatically.
Note: You can also create a UCF file for your project by selecting **Project → Create New Source**.
5. In the Timing Constraints dialog, enter the following in the Period, Pad to Setup, and CLock to Pad fields:
♦ **Period: 40**
♦ **Pade to Setup: 10**
♦ **Clock to Pad: 10**
6. Press **Enter**.

After the information has been entered, the dialog should look like what is shown below..



Clock Net Name	Period	Pad to Setup	Clock to Pad
CLOCK	40 ns. HIGH 50%	10 ns.	10 ns.

7. Select **Timing Constraints** under Constraint Type in the Timing Constraints tab and the newly created timing constraints are displayed as follows:



	Timing Constraints	Source
1	TIMESPEC TS_CLOCK = PERIOD "CLOCK" 40 ns HIGH 50%	counter.ucf
2	OFFSET = IN 10 ns VALID 0 ns BEFORE "CLOCK"	counter.ucf
3	OFFSET = OUT 10 ns AFTER "CLOCK"	counter.ucf
4	NET "CLOCK" TNM_NET = CLOCK	counter.ucf

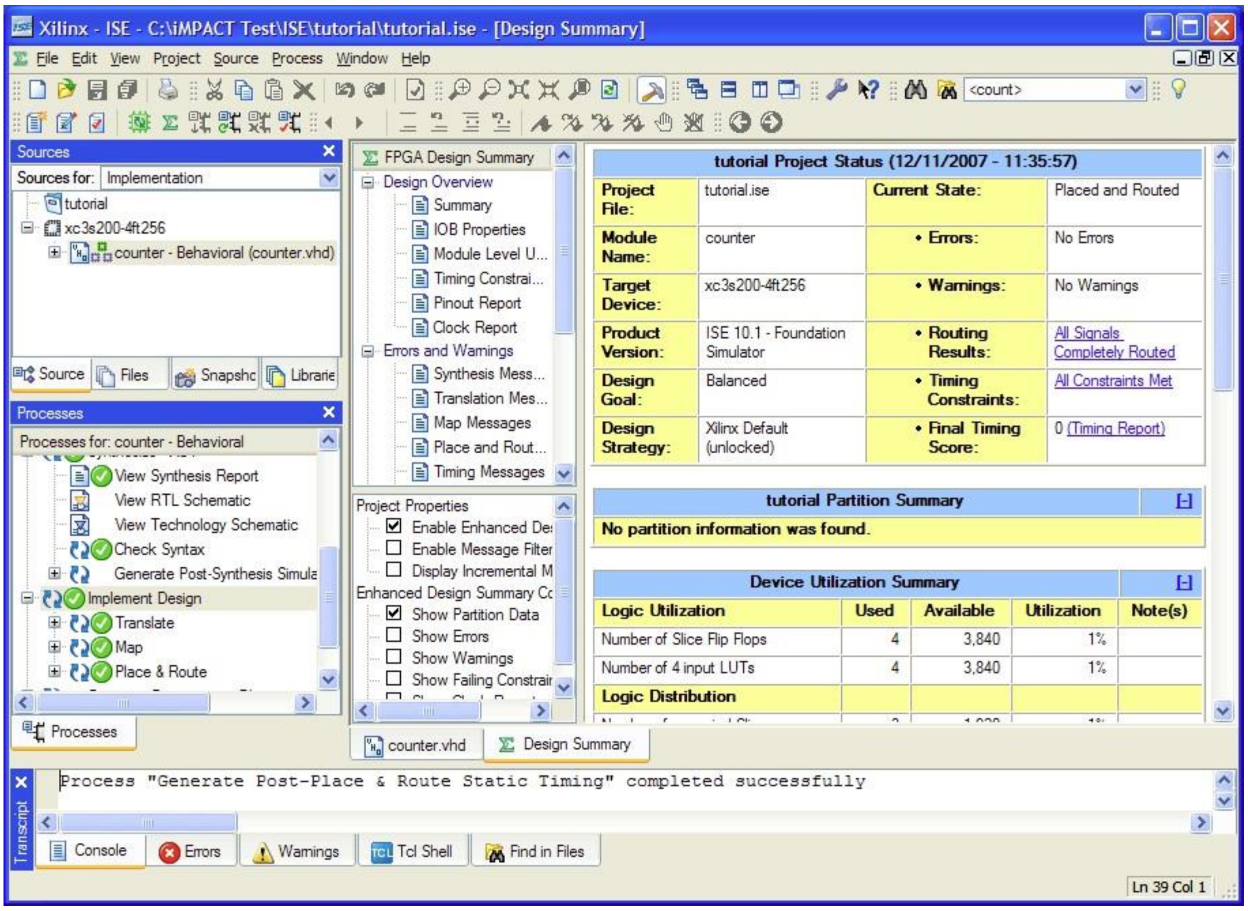
8. Save the timing constraints. If you are prompted to rerun the TRANSLATE or XST step, click **OK** to continue.
   8. Close the Constraints Editor.

IMPLEMENT DESIGN AND VERIFY CONSTRAINTS

Implement the design and verify that it meets the timing constraints specified in the previous section.

IMPLEMENTING THE DESIGN

1. Select the **counter** source file in the Sources window.
2. Open the Design Summary by double-clicking the **View Design Summary** process in the Processes tab.
3. Double-click the **Implement Design** process in the Processes tab.
4. Notice that after Implementation is complete, the Implementation processes have a green check mark next to them indicating that they completed successfully without Errors or Warnings.



tutorial Project Status (12/11/2007 - 11:35:57)

Project File:	tutorial.ise	Current State:	Placed and Routed
Module Name:	counter	• Errors:	No Errors
Target Device:	xc3s200-4ft256	• Warnings:	No Warnings
Product Version:	ISE 10.1 - Foundation Simulator	• Routing Results:	All Signals Completely Routed
Design Goal:	Balanced	• Timing Constraints:	All Constraints Met
Design Strategy:	Xilinx Default (unlocked)	• Final Timing Score:	0 (Timing Report)

tutorial Partition Summary

No partition information was found.

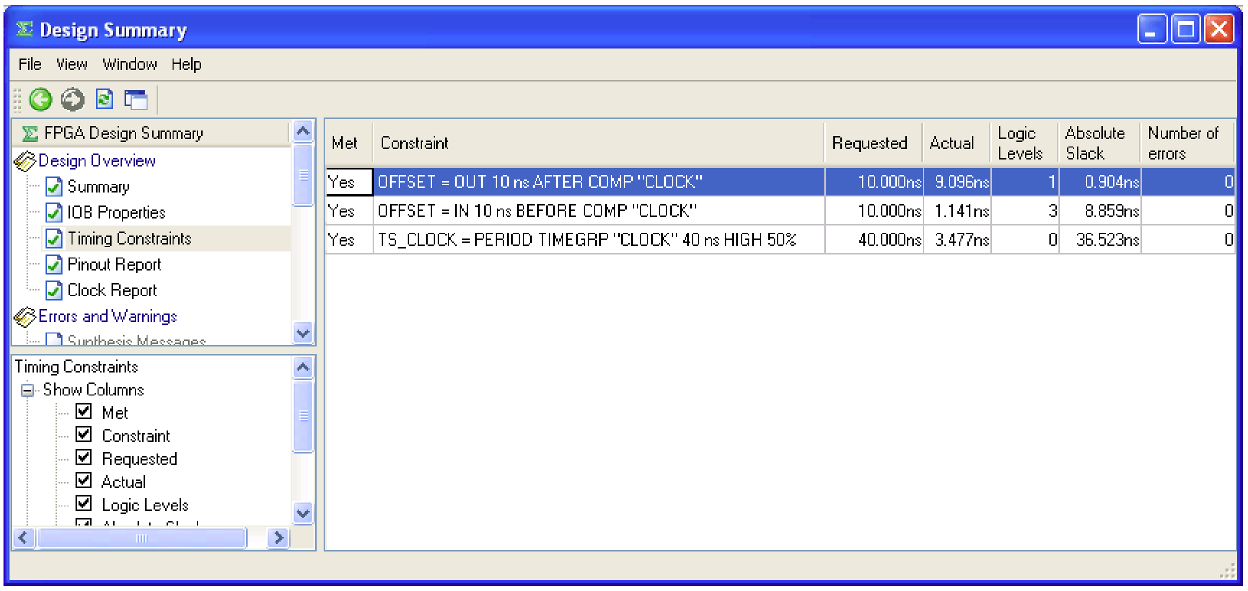
Device Utilization Summary

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	4	3,840	1%	
Number of 4 input LUTs	4	3,840	1%	
Logic Distribution				

```
Process "Generate Post-Place & Route Static Timing" completed successfully
```

5. Locate the **Performance Summary** table near the bottom of the Design Summary

6. Click the **All Constraints Met** link in the Timing Constraints field to view the Timing Constraints report. Verify that the design meets the specified timing requirements.

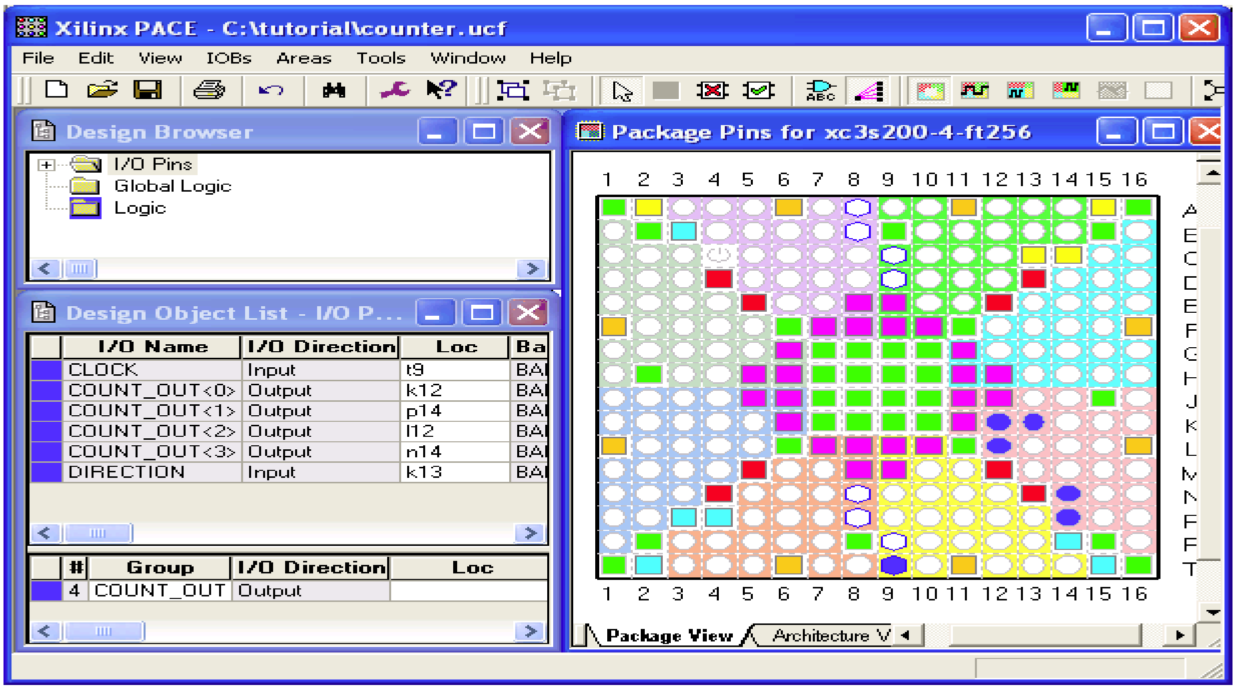


Met	Constraint	Requested	Actual	Logic Levels	Absolute Slack	Number of errors
Yes	OFFSET = OUT 10 ns AFTER COMP "CLOCK"	10.000ns	9.096ns	1	0.904ns	0
Yes	OFFSET = IN 10 ns BEFORE COMP "CLOCK"	10.000ns	1.141ns	3	8.859ns	0
Yes	TS_CLOCK = PERIOD TIMEGRP "CLOCK" 40 ns HIGH 50%	40.000ns	3.477ns	0	36.523ns	0

9. Close the Design Summary.

ASSIGNING PIN LOCATION CONSTRAINTS

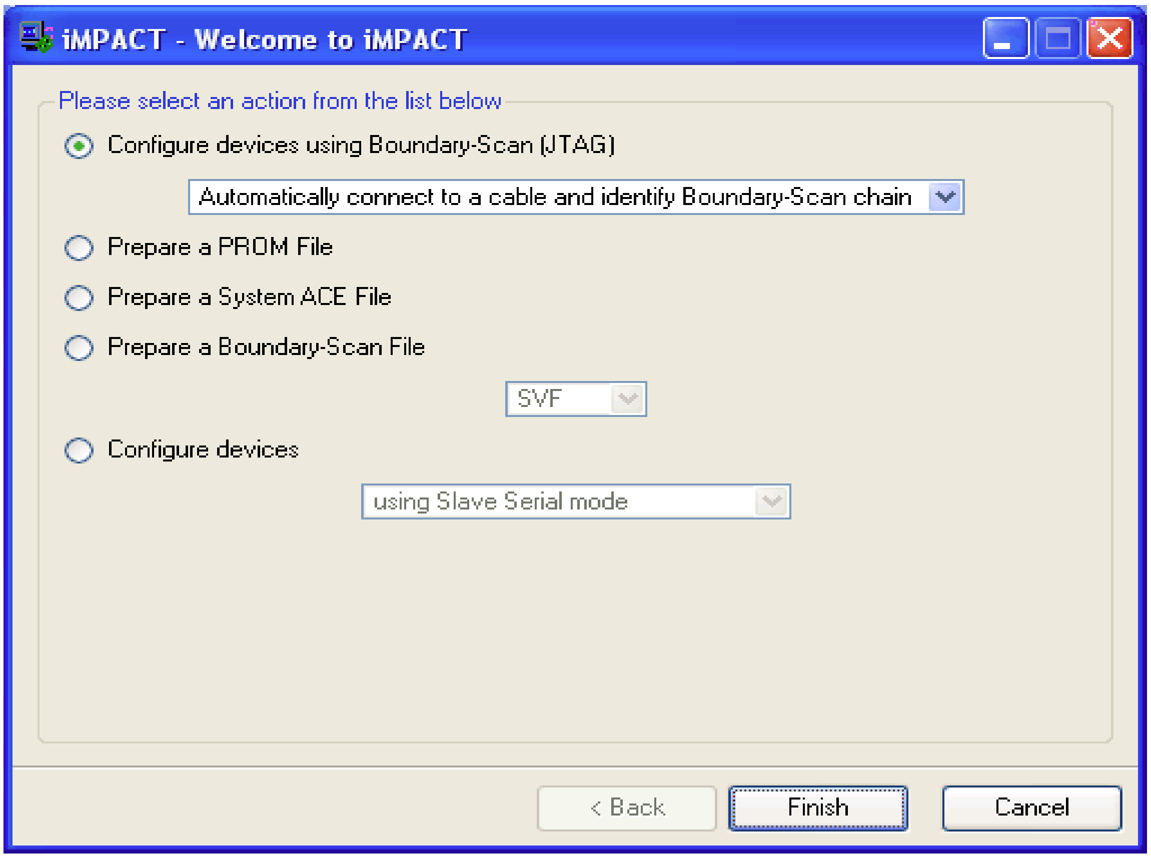
Specify the pin locations for the ports of the design so that they are connected correctly on the Spartan-3 Startup Kit demo board.
To constrain the design ports to package pins, do the following:
1. Verify that **counter** is selected in the Sources window.
2. Double-click the **Floorplan Area/IO/Logic - Post Synthesis** process found in the User Constraints process group. The Xilinx Pinout and Area Constraints Editor (PACE) opens.
3. Select the **Package View** tab.
4. In the Design Object List window, enter a pin location for each pin in the **Loc** column using the following information:


Xilinx PACE - C:\tutorial\counter.ucf
File Edit View IOBs Areas Tools Window Help
Design Browser
I/O Pins
Global Logic
Logic
Design Object List - I/O P...
Package Pins for xc3s200-4-ft256
Package View
Architecture V

I/O Name	I/O Direction	Loc	Ba
CLOCK	Input	t9	BA
COUNT_OUT<0>	Output	k12	BA
COUNT_OUT<1>	Output	p14	BA
COUNT_OUT<2>	Output	l12	BA
COUNT_OUT<3>	Output	n14	BA
DIRECTION	Input	k13	BA

#	Group	I/O Direction	Loc
4	COUNT_OUT	Output	

5. Select **File** → **Save**. You are prompted to select the bus delimiter type based on the synthesis tool you are using. Select **XST Default** <> and click **OK**.
6. Generate program file → Generate PROM, ACE or JTAG file



Please select an action from the list below

[✓ Configure devices using Boundary-Scan (JTAG)]

Automatically connect to a cable and identify Boundary-Scan chain

Prepare a PROM File

Prepare a System ACE File

Prepare a Boundary-Scan File

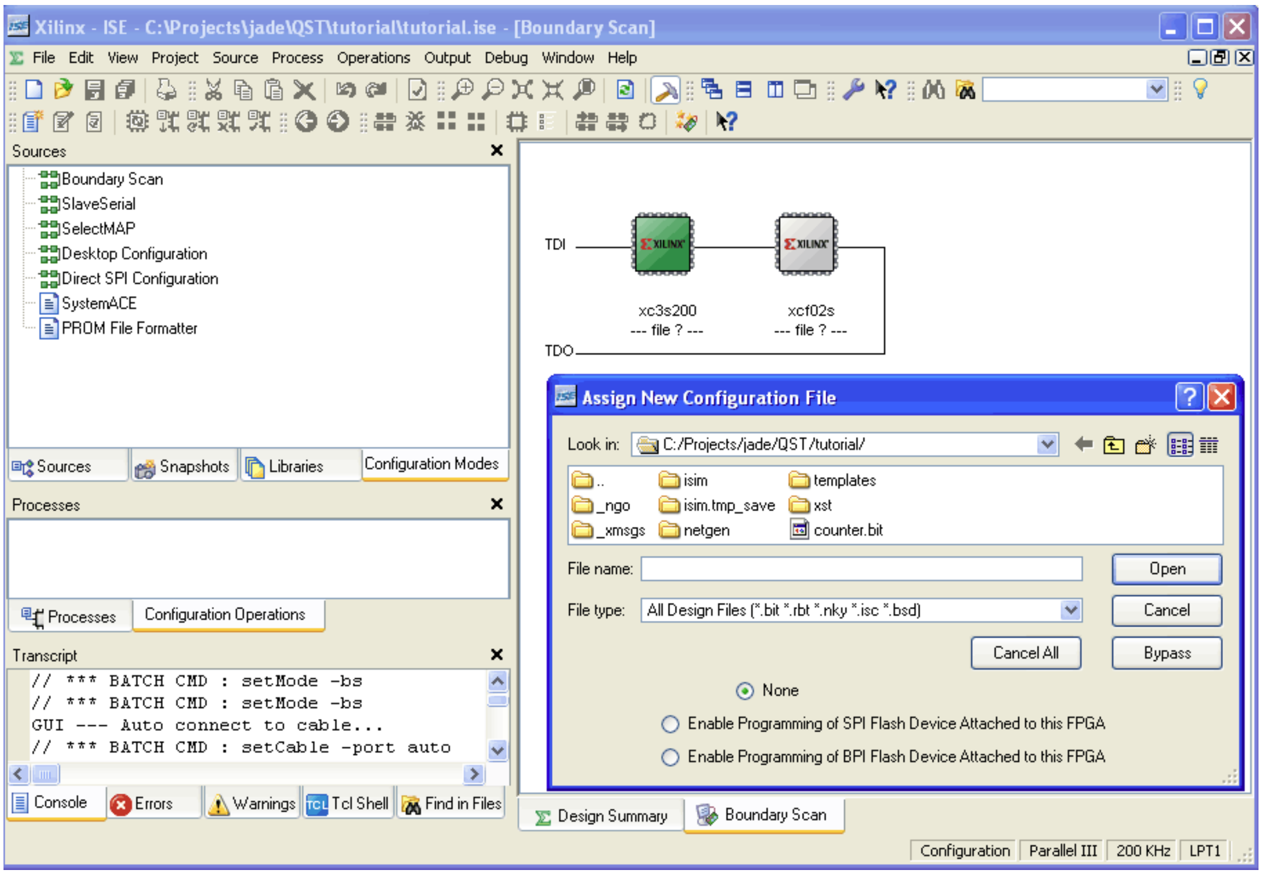
SVF

Configure devices

using Slave Serial mode

Verify that **Automatically connect to a cable and identify Boundary-Scan chain** is selected.
9. Click **Finish**.
10. If you get a message saying that there are two devices found, click **OK** to continue.
The devices connected to the JTAG chain on the board will be detected and displayed in the iMPACT window.

11. The **Assign New Configuration File** dialog box appears. To assign a configuration file to the xc3s200 device in the JTAG chain, select the counter.bit file and click **Open**.


Xilinx - ISE - C:\Projects\jade\QST\tutorial\tutorial.ise - [Boundary Scan]
File Edit View Project Source Process Operations Output Debug Window Help
Sources
Boundary Scan
SlaveSerial
SelectMAP
Desktop Configuration
Direct SPI Configuration
SystemACE
PROM File Formatter
Sources Snapshots Libraries Configuration Modes
Processes
Processes Configuration Operations
Transcript
Console Errors Warnings Tcl Shell Find in Files
TDI
TDO
xc3s200
--- file ? ---
xcf02s
--- file ? ---
Assign New Configuration File
Look in: C:/Projects/jade/QST/tutorial/
..
_ngo
_xmsgs
isim
isim.tmp_save
netgen
templates
xst
counter.bit
File name:
File type: All Design Files (*.bit *.rbt *.nky *.isc *.bsd)
Open
Cancel
Cancel All
Bypass
None
Enable Programming of SPI Flash Device Attached to this FPGA
Enable Programming of BPI Flash Device Attached to this FPGA
Design Summary
Boundary Scan
Configuration
Parallel III
200 KHz
LPT1

```
// *** BATCH CMD : setMode -bs
// *** BATCH CMD : setMode -bs
GUI --- Auto connect to cable...
// *** BATCH CMD : setCable -port auto
```

12. If you get a Warning message, click **OK**.
13. Select **Bypass** to skip any remaining devices.
14. Right-click on the xc3s200 device image, and select **Program...** The **Programming Properties** dialog box opens.
15. Click **OK** to program the device.
When programming is complete, the Program Succeeded message is displayed.

Program Succeeded

On the board, LEDs 0, 1, 2, and 3 are lit, indicating that the counter is running.
16. Close iMPACT without saving.

Result:

Thus, the Simulation tools, the Synthesis tools, the Place and Root and Back annotation for FPGAs was studied.

EX. NO 02 STUDY OF DEVELOPMENT TOOL FOR FPGA

Spartan-6 Trainer Kit – VPTB – 20

Introduction

Vi Microsystems Xilinx Spartan-6 Trainer Kit (VPTB – 20) is a demonstration platform intended for you to become familiar with the new features and availability of the Spartan-6 FPGA family. This Kit provides a low-cost, easy to use development and evaluation platform for Spartan-6 FPGA designs.

Features

➔ XC6SLX16 Based on Xilinx Spartan-6 FPGA features:
  - o 14579 Logic cells
  - o 2278 Kb of Distributed RAM
  - o 136 Kb of Block RAM
  - o 32 NOS of DSP Slices
  - o 2 NOS of Clock Management Tiles (CMT)
  - o 2 NOS of Memory Controller Block (MCB)
  - o 186 User I/O lines

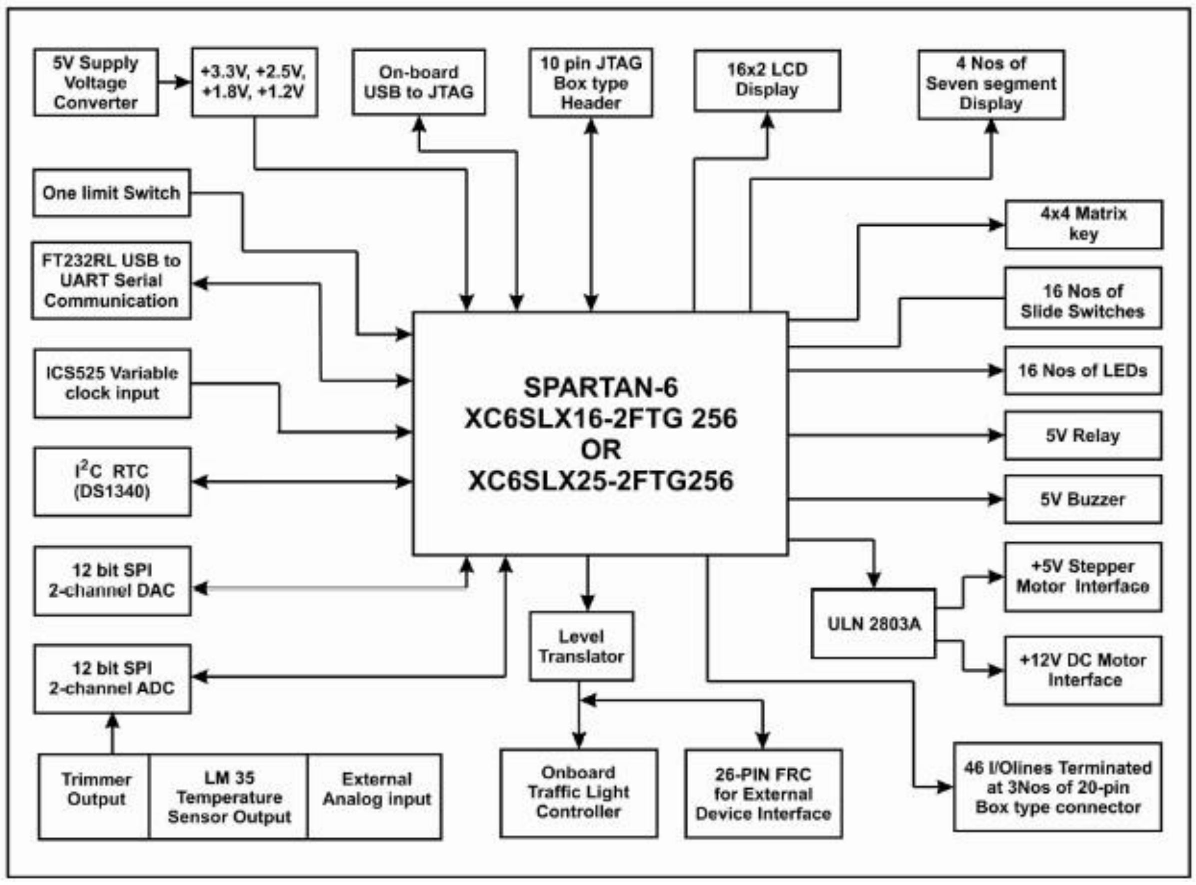
➔ 16 Nos. of digital input using slide switches with LED indication
➔ 16 Nos. of digital output using discrete LEDs
➔ 16× 2 LCD is provided for display the text message.
➔ One Reset Switch
➔ One Limit Switch for giving manual clock
➔ FPGA configuration through
  - On-board Xilinx USB to JTAG Programmer with isolation to configure FPGA
  - On board Flash Prom XCF04S (programmable through USB to JTAG programmer)

➔ Total 186 I/O Pins: 100 Pins used for integrating peripheral like LED, Switches etc., balance pins are available to user in 3 nos of 20-pin headers.
➔ 1 No of 26 pin header to interface VLIM cards like Traffic Light Controller / On-board TLC factory settable.
➔ On board programmable PLL oscillator from1 MHz to 100MHz using jumpers.
➔ 4 Nos of 7 segment LED display
➔ One relay and Buzzer provided.
➔ Stepper & DC motor driver provided on board (Motor Optional)
➔ 4x4 matrix key provided
➔ SPI Based Analog to Digital Converter
  - Resolution → 12 bits
  - Number of Channel → 2 Channels
  - Input Range → 0 to 5V
  - Speed → 1 Msps

➔ SPI Based Digital to Analog converter

- Resolution →12 bits
- Number of Channel → 2 Channels
- Output Range → 0 to 5V
- Speed → 8.5 µS settling time

Block Diagram


5V Supply Voltage Converter
+3.3V, +2.5V, +1.8V, +1.2V
On-board USB to JTAG
10 pin JTAG Box type Header
16x2 LCD Display
4 Nos of Seven segment Display
One limit Switch
FT232RL USB to UART Serial Communication
ICS525 Variable clock input
I2C RTC (DS1340)
12 bit SPI 2-channel DAC
12 bit SPI 2-channel ADC
SPARTAN-6 XC6SLX16-2FTG 256 OR XC6SLX25-2FTG256
4x4 Matrix key
16 Nos of Slide Switches
16 Nos of LEDs
5V Relay
5V Buzzer
+5V Stepper Motor Interface
ULN 2803A
+12V DC Motor Interface
Level Translator
46 I/Olines Terminated at 3Nos of 20-pin Box type connector
Trimmer Output
LM 35 Temperature Sensor Output
External Analog input
Onboard Traffic Light Controller
26-PIN FRC for External Device Interface

Clock FPGA Pin = E7

Power Switch (Slide Switch (SW1)) & LED (L1)
The Spartan-6 Trainer Kit has a slide power switch. Moving the power switch Up for Power ON, the LED L1 will glow and down for power OFF, the LED L1 will off.

Configuration Switch & LED (SW2, L2)
The Spartan-6 Trainer Kit has a push button Switch (named as CONFIG) to Configure the FPGA from Xilinx Serial Flash PROM. During configuration process of the FPGA the LED L2 will be off and after successful configuration the LED L2 will glow.

Reset Switch (SW38)

The Spartan-6 Trainer kit has a push button switch (named as RESET) which can be configured as input by assigning the corresponding FPGA pin location.

Name	FPGA
RESET (SW34)	P175

Limit Switch (SW21)

This switch is used to increment the speed or used to give manual clock. FPGA location for the switch is given below

Name	SW21
FPGA Pin	R7

Input Switches

The Spartan-6 Trainer Kit has 16 nos of slide switches with led indication for giving inputs to the FPGA I/O lines. The slide switches are located in the bottom corner of the board and are labeled as SW22 through SW37. Switch SW22 is the left-most switch, and SW37 is the right-most switch**.** When in the UP or ON position, a switch connects the FPGA pin to 3.3V, a logic High. When DOWN or in the OFF position, the switch connects the FPGA pin to ground, a logic Low. The switches typically exhibit about 2 ms of mechanical bounce. There is no active de-bouncing circuitry, although such circuitry could easily be added to the FPGA design programmed on the board.

Slide Switch connections with FPGA

SWITCH	SW22	SW23	SW24	SW25	SW26	SW27	SW28	SW29
FPGA Pin	B8	D8	A7	F7	B6	D6	F6	B5
SWITCH	SW30	SW31	SW32	SW33	SW34	SW35	SW36	SW37
FPGA Pin	D5	A4	A3	C3	B2	C2	D3	E3

Matrix Keypad

There are 16 The VPTB-10 board has 16 momentary-contact push button switches are arranged in 4 x 4 matrix format, and multiplexed by 8 IO pins to FPGA. There are 4 output lines named a3 to a0 (row) and 4 input lines are available named as b3 to b0 (column).

Matrix Keypad Connections with FPGA

Signal	a<0>	a<1>	a<2>	a<3>	b<0>	b<1>	b<2>	b<3>
FPGA Pin	J1	H1	H2	H3	H4	H5	G1	G3

Output LEDs

The Spartan-6 Trainer Kit board has sixteen individual surface-mount LEDs located immediately above the slide switches. The LEDs are labeled L40 through L55. L40 is the left-most LED, L55 the right-most LED. Each LED has one side connected to ground and the other side connected to a pin on the Spartan6 device via a 270O current limit ing resistor. To light an individual LED, drive the associated FPGA control signal

LED connections with FPGA

LED	L40	L41	L42	L43	L44	L45	L46	L47
FPGA Pin	A8	C8	E8	C7	A6	C6	E6	A5
LED	L48	L49	L50	L51	L52	L53	L54	L55
FPGA Pin	C5	F5	E4	B3	A2	B1	C1	D1

Seven Segment Display

The VPTB-20 board has a four-character, seven segment LED display controlled by FPGA user- I/O pins. Each individual character has a separate common pin, and this pin is connected to J10 3-pin jumper. When using common cathode control input seven segment display short this jumper to ground and when using common anode control input seven segment display short this jumper to +3.3V. The LED control signals are connected to the individual line of the FPGA I/O.

Character	a	b	c	d	e	f	g
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	0	0	1	1
A	1	1	1	0	1	1	1
B	0	0	1	1	1	1	1
C	1	0	0	1	1	1	0
D	0	1	1	1	1	0	1
E	1	0	0	1	1	1	1
F	1	0	0	0	1	1	1

SEVEN SEGMENT Connections with FPGA

Disp1

DISPLAY1	seg-a	seg-b	seg-c	seg-d	seg-e	seg-f	seg-g	seg-dp
FPGA Pin	T7	P7	N5	P1	P2	L8	N8	N4

Disp3

DISPLAY2	seg-a	seg-b	seg-c	seg-d	seg-e	seg-f	seg-g	seg-dp
FPGA Pin	P5	P4	L4	L5	L7	R1	R2	L3

Disp2

DISPLAY3	seg-a	seg-b	seg-c	seg-d	seg-e	seg-f	seg-g	seg-dp
FPGA Pin	P6	N6	M6	N1	N3	T6	M7	M5

Disp4

DISPLAY4	seg-a	seg-b	seg-c	seg-d	seg-e	seg-f	seg-g	seg-dp
FPGA Pin	T4	T3	M2	M3	M4	T5	R5	M1

RELAY & BUZZER

The VPTB-20 contains Buzzer & Relay circuit, and both works in 5V DC Voltage. A buzzer or beeper is a signaling device, usually electronic, typically used in automobiles, household appliances such as a microwave oven, or game shows. Buzzer generates different tone Generator. Relay is used as On/Off switch depending upon user needs.

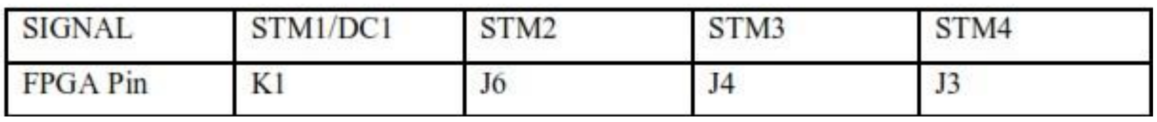
BUZZER & RELAY Connections with FPGA

SIGNAL	Buzzer	Relay
FPGA Pin	K2	L1

STEPPER MOTOR & DC MOTOR

The VPTB-20 Contains Stepper Motor Interface Connector to Control the Stepper Motor Speed and it also controls the Step Angle of the Motor. VPTB-20 board has the provision to interface the 5V stepper motor and +12V DC motor through the ULN2803A driver IC. P9 2-pin J801 connector has the option to give external supply voltage to the driver IC and J13 jumper will choose the driver voltage is be the on-board +5V or be the external voltage. Driver output lines for stepper motor are terminated in P11 5-pin RMC connector and for DC motor lines are terminated in P7 2-pin J801 connector. But we cannot execute the stepper motor and dc motor alone not in the same time.

Motor connections with FPGA



SIGNAL	STM1/DC1	STM2	STM3	STM4
FPGA Pin	K1	J6	J4	J3

Result:

Thus, the development tool for FPGAs was studied.

Expt.No: 3 BASIC LOGIC GATES

AIM:

To implement basic logic gates using Verilog HDL.

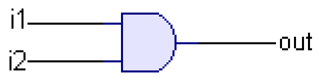
APPARATUS REQUIRED:

PC with Windows XP.
XILINX, ModelSim software.
FPGA kit.
RS 232 cable.

PROCEDURE:

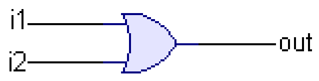
- Write and draw the Digital logic system.
- Write the Verilog code for above system.
- Enter the Verilog code in Xilinx software.
- Check the syntax and simulate the above verilog code (using ModelSim or Xilinx) and verify the output waveform as obtained.
- Implement the above code in Spartan III using FPGA kit.

AND Gate: Truth table



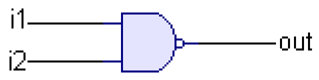
Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate:



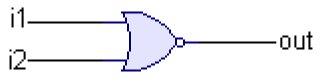
Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

NAND Gate:



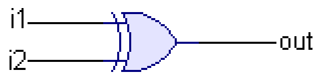
Input 1	Input 2	Output
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate:



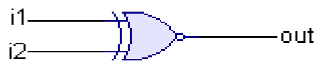
Input 1	Input 2	Output
0	0	1
0	1	0
1	0	0
1	1	0

XOR Gate:



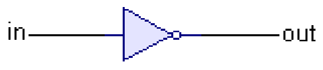
Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

XNOR Gate:



Input 1	Input 2	Output
0	0	1
0	1	0
1	0	0
1	1	1

NOT Gate:

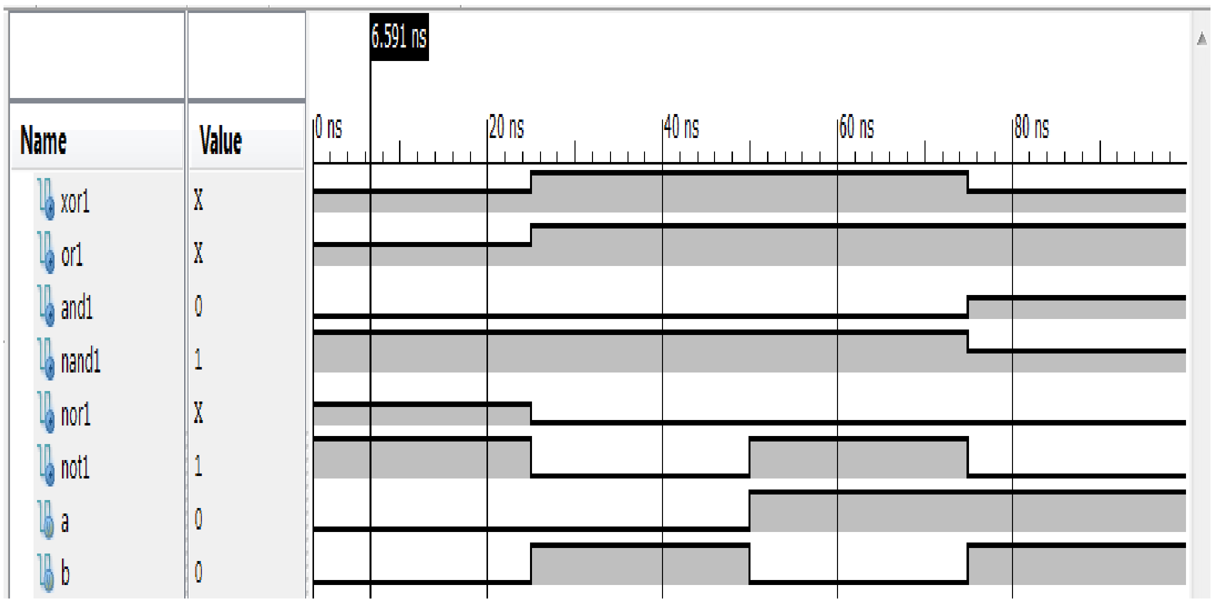


Input	Output
0	1
1	0

PROGRAM

```
module logic_gates(a,b,xor1,or1,and1,nand1,nor1,not1);
input a,b;
output xor1,or1,and1,nand1,nor1,not1;
and (and1,a,b);
or (or1,a,b);
not (not1,a);
nor (nor1,a,b);
nand (nand1,a,b);
xor (xor1,a,b);
endmodule
```

WAVEFORM:



RESULT:

Thus, the basic logic gates are simulated and synthesized using Verilog HDL. And implement in FPGA.

Expt.No: 4 HALF ADDER

AIM:

To implement half adder using Verilog HDL.

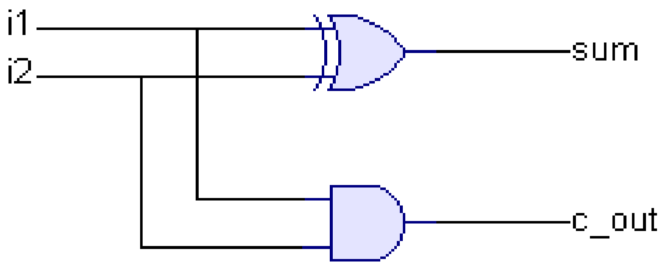
APPARATUS REQUIRED:

PC with Windows XP.
XILINX, ModelSim software.
FPGA kit.
RS 232 cable.

PROCEDURE:

- Write and draw the Digital logic system.
- Write the Verilog code for above system.
- Enter the Verilog code in Xilinx software.
- Check the syntax and simulate the above verilog code (using ModelSim or Xilinx) and verify the output waveform as obtained.
- Implement the above code in Spartan III using FPGA kit.

Half adder:



Output:

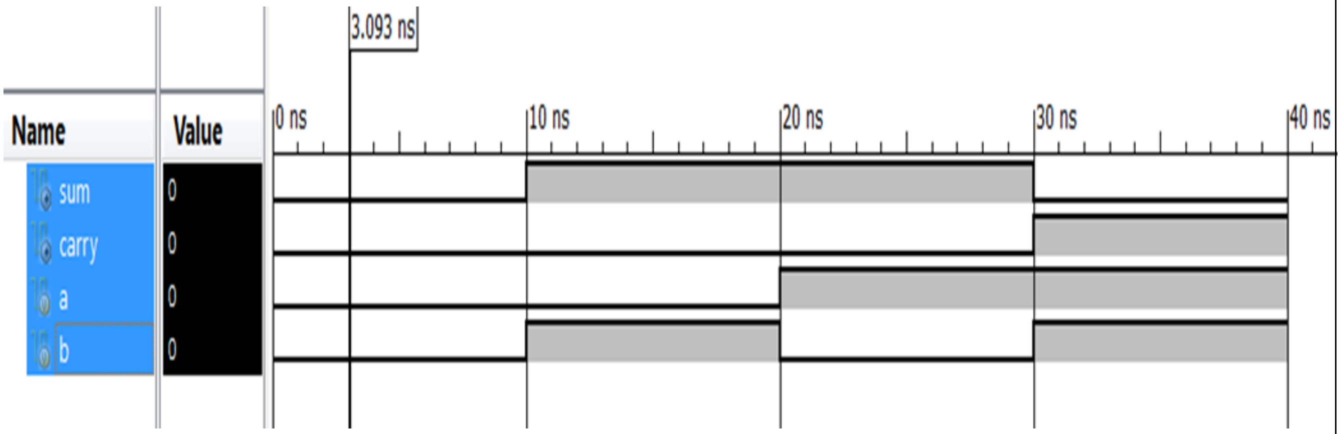
Input 1	Input 2	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

PROGRAM

GATE LEVEL MODELING:

```
module half_adder(a,b,sum,carry);
input a,b;
output sum,carry;
xor(sum,a,b);
and (carry,a,b);
endmodule
```

WAVEFORM:



RESULT:

Thus, the half adder are simulated and synthesized using Verilog HDL and implement in FPGA.

Expt.No: 5 FULL ADDER

AIM:

To implement full adder using Verilog HDL.

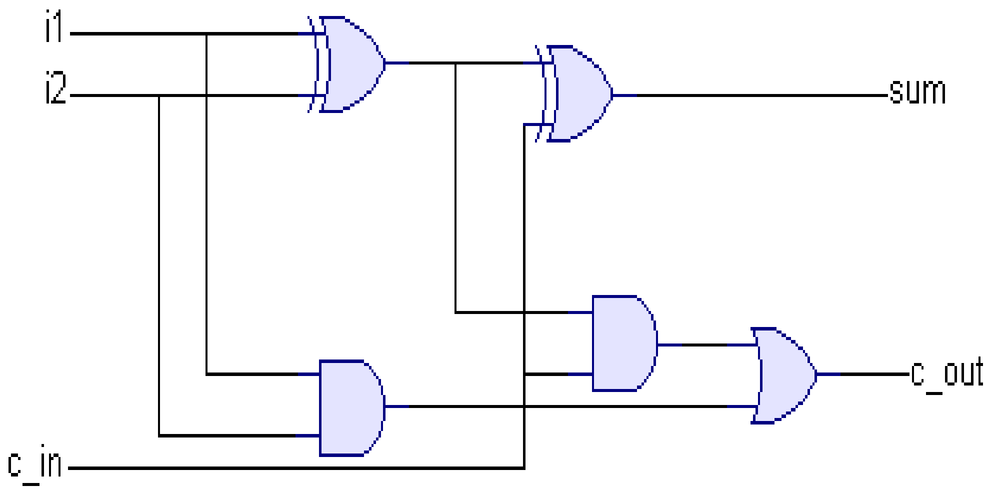
APPARATUS REQUIRED:

PC with Windows XP.
XILINX, ModelSim software.
FPGA kit.
RS 232 cable.

PROCEDURE:

- Write and draw the Digital logic system.
- Write the Verilog code for above system.
- Enter the Verilog code in Xilinx software.
- Check the syntax and simulate the above verilog code (using ModelSim or Xilinx) and verify the output waveform as obtained.
- Implement the above code in Spartan III using FPGA kit.

Full adder:



Output:

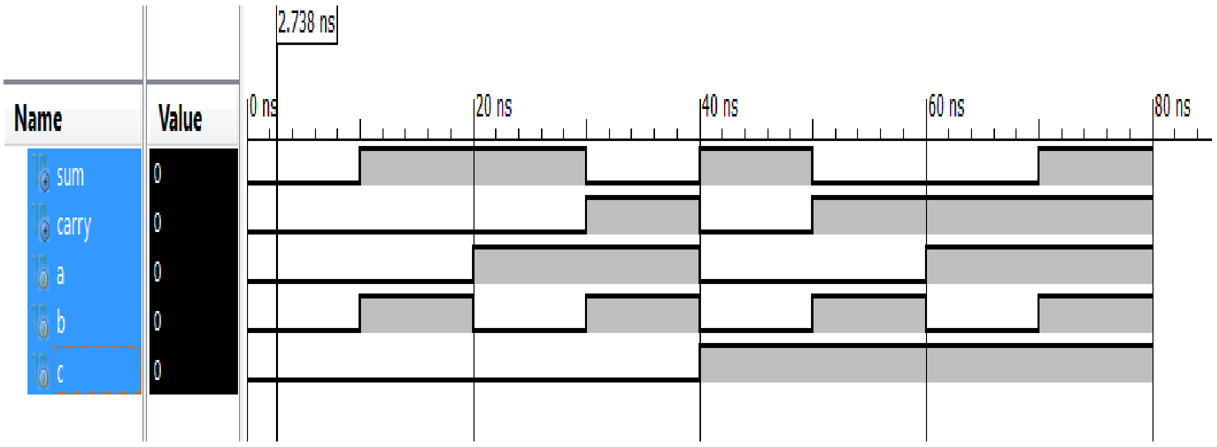
Input 1	Input 2	Cin	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

PROGRAM

BEHAVIORAL LEVEL MODELING:

```
module full_adder(a,b,c,sum,carry);
input a,b,c;
output reg sum,carry;
always @ (a or b or c)
begin
sum=a^b^c;
carry=(a&b)|(b&c)|(c&a);
end
endmodule
```

WAVEFORM:



RESULT:

Thus, the full adder are simulated and synthesized using Verilog HDL. And implement in FPGA.

Expt.No: 6 HALF SUBTRACTOR

AIM:

To implement half subtractor using Verilog HDL.

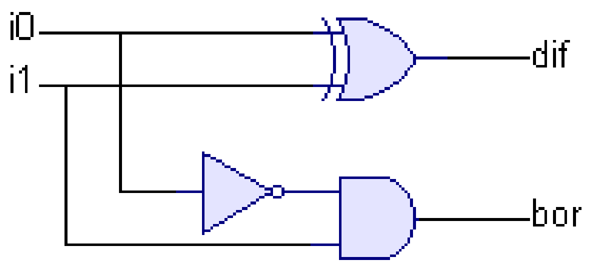
APPARATUS REQUIRED:

PC with Windows XP.
XILINX, ModelSim software.
FPGA kit.
RS 232 cable.

PROCEDURE:

- Write and draw the Digital logic system.
- Write the Verilog code for above system.
- Enter the Verilog code in Xilinx software.
- Check the syntax and simulate the above verilog code (using ModelSim or Xilinx) and verify the output waveform as obtained.
- Implement the above code in Spartan II using FPGA kit.

Half subtractor:



Output:

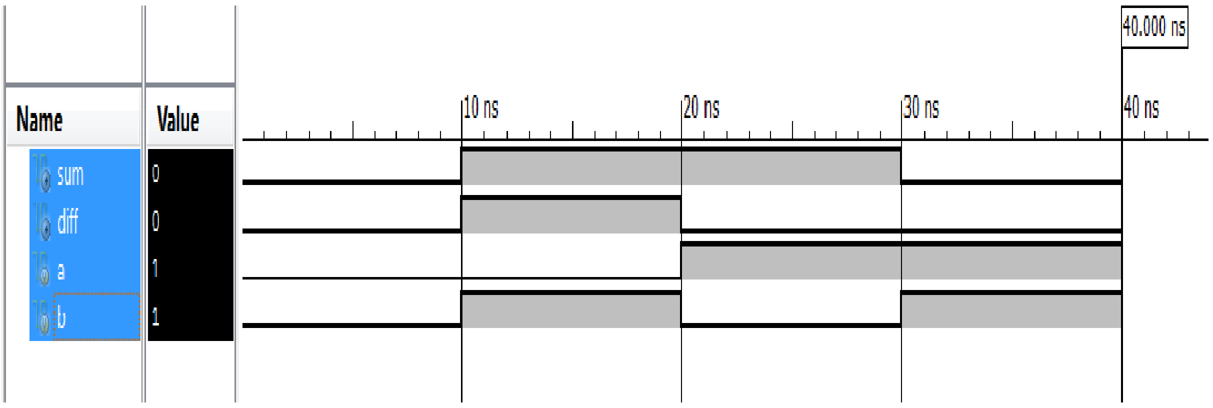
Input 1	Input 2	Borrow	Difference
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

PROGRAM

DATAFLOW MODELING:

```
module half_subtractor(a,b,sum,diff);
input a,b;
output sum,diff;
assign sum=a^b,
          diff=((~a)&b);
endmodule
```

WAVEFORM:



RESULT:

Thus, the half subtractor are simulated and synthesized using Verilog HDL. And implement in FPGA.

Expt.No: 7 **FULL SUBTRACTOR**

AIM:

To implement full subtractor using Verilog HDL.

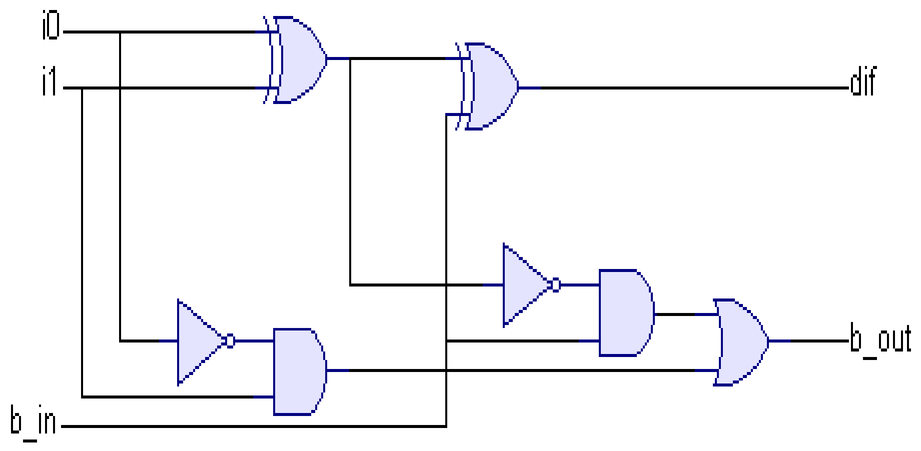
APPARATUS REQUIRED:

PC with Windows XP.
XILINX, ModelSim software.
FPGA kit.
RS 232 cable.

PROCEDURE:

- Write and draw the Digital logic system.
- Write the Verilog code for above system.
- Enter the Verilog code in Xilinx software.
- Check the syntax and simulate the above verilog code (using ModelSim or Xilinx) and verify the output waveform as obtained.
- Implement the above code in Spartan II using FPGA kit.

Full subtractor:



Output:

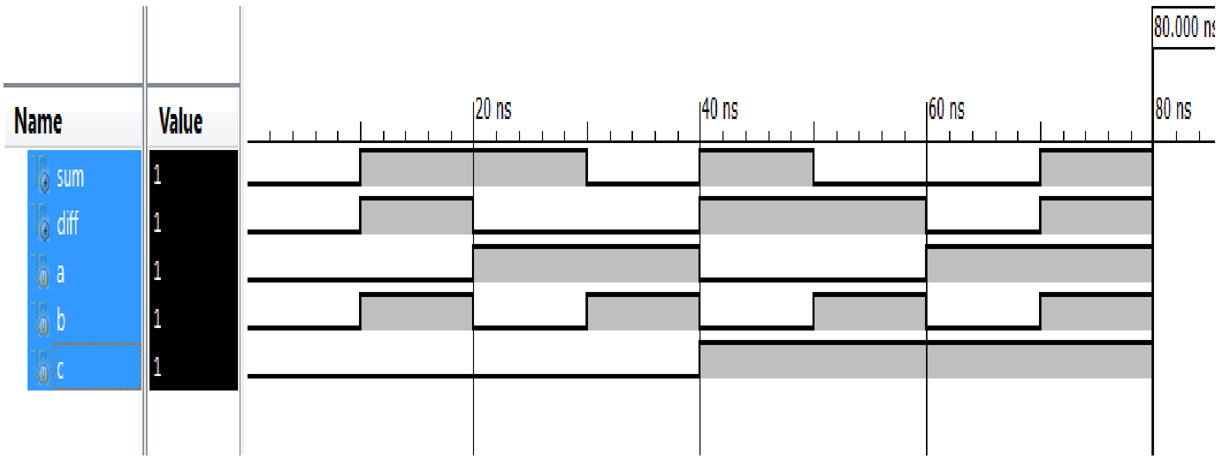
Input 1	Input 2	Cin	Borrow	Difference
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

PROGRAM

BEHAVIORAL LEVEL MODELING:

```
module full_ subtractor (a,b,c,sum,diff);
input a,b,c;
output reg  sum, diff;
always @ (a or b or c)
begin
sum=a^b^c;
diff =((~a)&b)|(b&c)|(c&(~a));
end
endmodule
```

WAVEFORM:



RESULT:

Thus, the full subtractor are simulated and synthesized using Verilog HDL. And implement in FPGA.

Expt.No: 8 RIPPLE CARRY ADDER

AIM:

To implement Ripple carry adder using Verilog HDL.

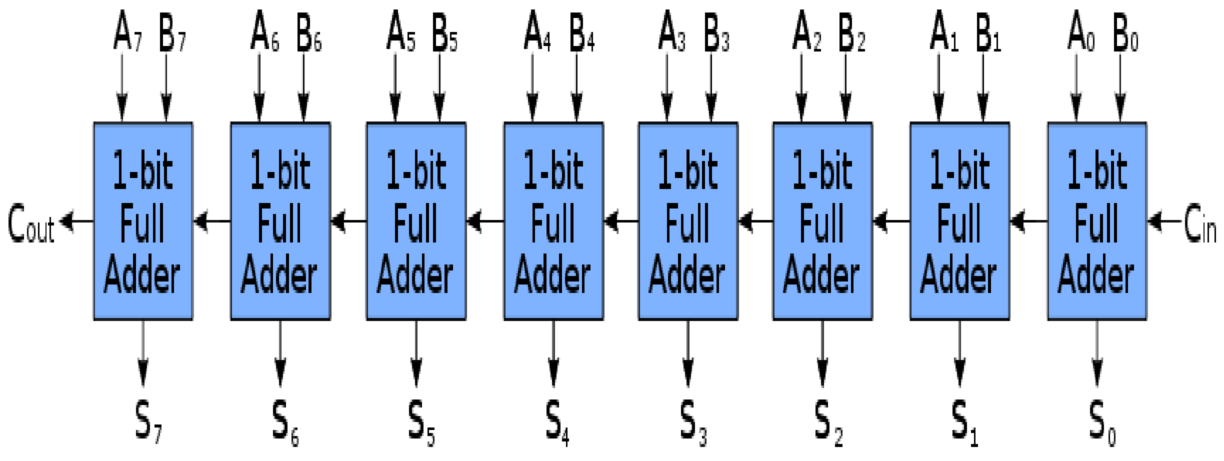
APPARATUS REQUIRED:

PC with Windows XP.
XILINX, ModelSim software.
FPGA kit.
RS 232 cable.

PROCEDURE:

- Write and draw the Digital logic system.
- Write the Verilog code for above system.
- Enter the Verilog code in Xilinx software.
- Check the syntax and simulate the above verilog code (using ModelSim or Xilinx) and verify the output waveform as obtained.
- Implement the above code in Spartan II using FPGA kit.

Ripple carry adder



PROGRAM

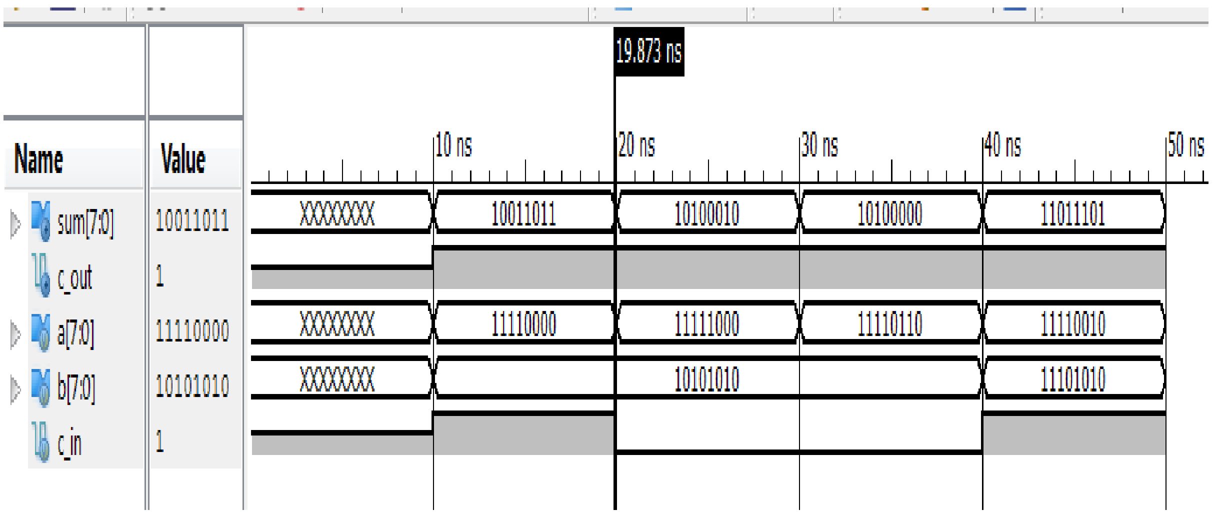
BEHAVIORAL LEVEL MODELING:

```
module fulladd(sum, c_out, a, b, c_in);
output sum, c_out;
input a, b, c_in;
wire s1, c1, c2;
xor (s1, a, b);
and (c1, a, b);
xor (sum, s1, c_in);
and (c2, s1, c_in);
xor (c_out, c2, c1);
endmodule


// PORT MAPPING //

module fulladd8(sum, c_out, a, b, c_in);
output [7:0] sum;
output c_out;
input[7:0] a, b;
input c_in;
wire c1, c2, c3 ,c4,c5,c6,c7;
fulladd fa0(sum[0], c1, a[0], b[0], c_in);
fulladd fa1(sum[1], c2, a[1], b[1], c1);
fulladd fa2(sum[2], c3, a[2], b[2], c2);
fulladd fa3(sum[3], c4, a[3], b[3], c3);
fulladd fa4(sum[4], c5, a[4], b[4], c4);
fulladd fa5(sum[5], c6, a[5], b[5], c5);
fulladd fa6(sum[6], c7, a[6], b[6], c6);
fulladd fa7(sum[7], c_out, a[7], b[7], c7);
endmodule
```

WAVEFORM:



RESULT:

Thus, the Ripple carry adder are simulated and synthesized using Verilog HDL. And implement in FPGA.

Expt.No: 9 T - FLIP FLOP

AIM:

To implement T flip flop using Verilog HDL.

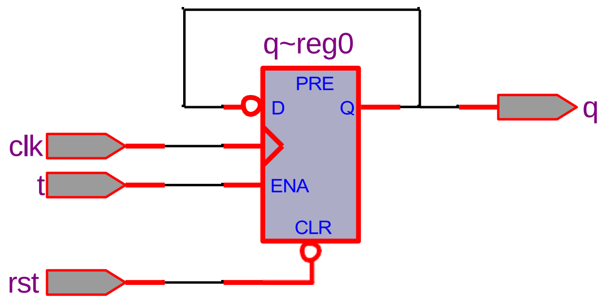
APPARATUS REQUIRED:

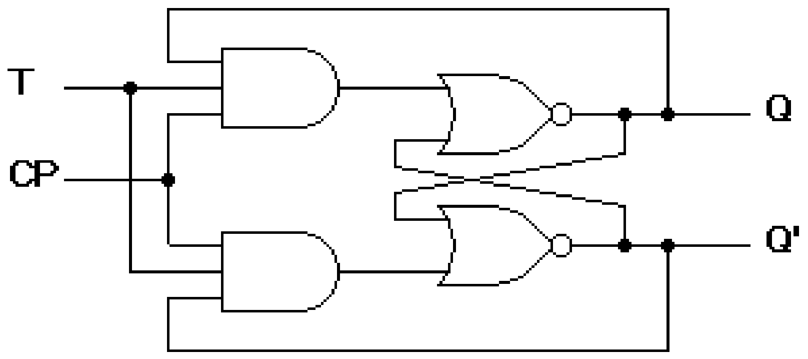
PC with Windows XP.
XILINX, ModelSim software.
FPGA kit.
RS 232 cable.

PROCEDURE:

- Write and draw the Digital logic system.
- Write the Verilog code for above system.
- Enter the Verilog code in Xilinx software.
- Check the syntax and simulate the above verilog code (using ModelSim or Xilinx) and verify the output waveform as obtained.
- Implement the above code in Spartan II using FPGA kit.

T Flip flop:





Output:

RST	Q	T	Qn+1
0	0	1	0
1	0	1	1
1	1	0	1
1	1	1	0

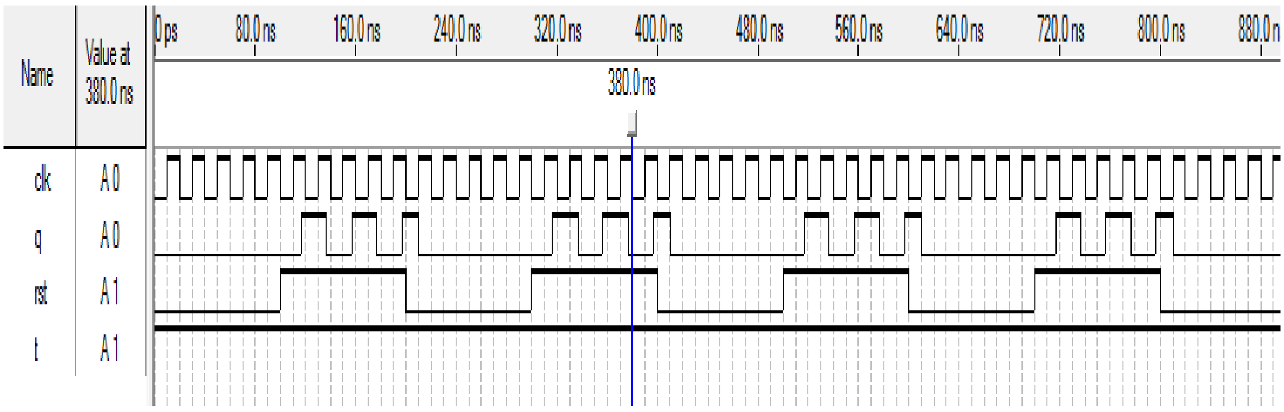
PROGRAM

BEHAVIORAL LEVEL MODELING:

T FLIP FLOP:

```
module TF (t,rst,clk,q);
 input t,rst,clk;
output reg q;
always @ (posedge clk or negedge rst)
begin
if  (rst==0)
   q<=1'b0;
else
q<=q^t;
end
endmodule
```

WAVEFORM:



RESULT:

Thus, the T- flip flop is simulated and synthesized using Verilog HDL. And implement in FPGA.

Expt.No: 10 **DELAY FLIP FLOP**

AIM:

To implement D flip flops using Verilog HDL.

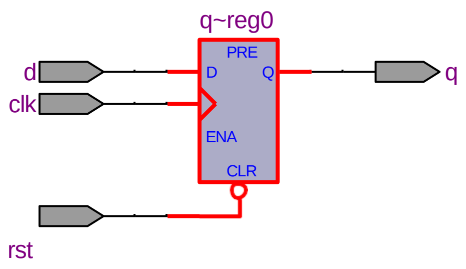
APPARATUS REQUIRED:

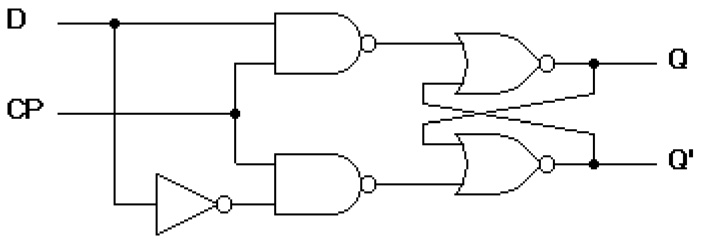
PC with Windows XP.
XILINX, ModelSim software.
FPGA kit.
RS 232 cable.

PROCEDURE:

- Write and draw the Digital logic system.
- Write the Verilog code for above system.
- Enter the Verilog code in Xilinx software.
- Check the syntax and simulate the above verilog code (using ModelSim or Xilinx) and verify the output waveform as obtained.
- Implement the above code in Spartan II using FPGA kit.

D Flip flop:





Output:

Q	D	Qn+1
0	0	0
0	1	1
1	0	0
1	1	1

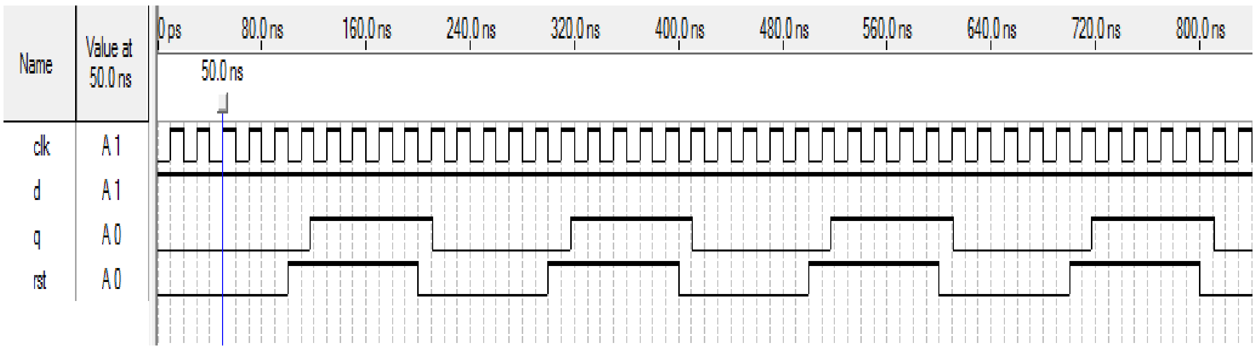
PROGRAM

BEHAVIORAL LEVEL MODELING:

D FLIP FLOP:

```
module df (d,rst,clk,q);
 input d,rst,clk;
output reg q;
always @ (posedge clk or negedge rst)
begin
if  (rst==0)
   q<=1'b0;
else
q<=d;
end
endmodule
```

WAVEFORM:



RESULT:

Thus, the D- flip flop is simulated and synthesized using Verilog HDL. And implement in FPGA.

Expt.No: 11 MULTIPLEXER

AIM:

To implement multiplexer using Verilog HDL.

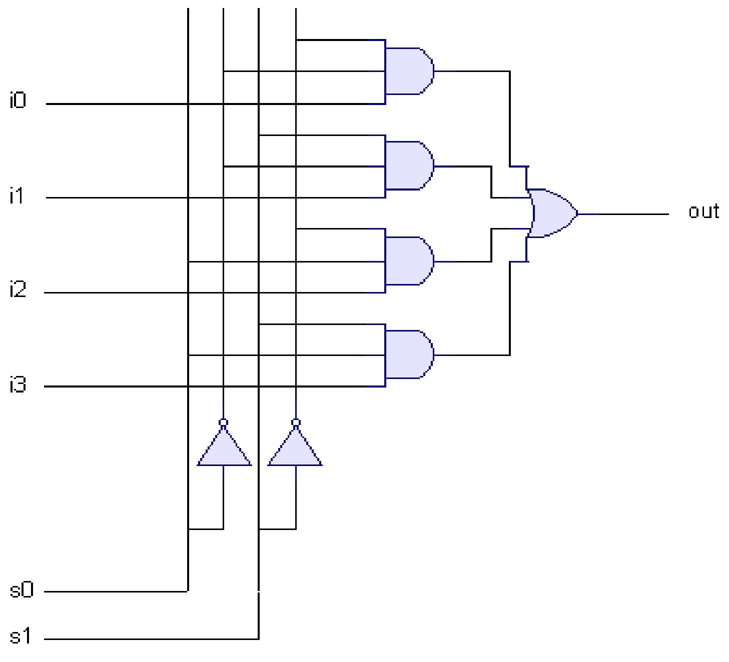
APPARATUS REQUIRED:

PC with Windows XP.
XILINX, ModelSim software.
FPGA kit.
RS 232 cable.

PROCEDURE:

- Write and draw the Digital logic system.
- Write the Verilog code for above system.
- Enter the Verilog code in Xilinx software.
- Check the syntax and simulate the above verilog code (using ModelSim or Xilinx) and verify the output waveform as obtained.
- Implement the above code in Spartan II using FPGA kit.

Multiplexer



S0	S1	OUTPU
0	0	I0
0	1	I1
1	0	I2
1	1	I3

PROGRAM

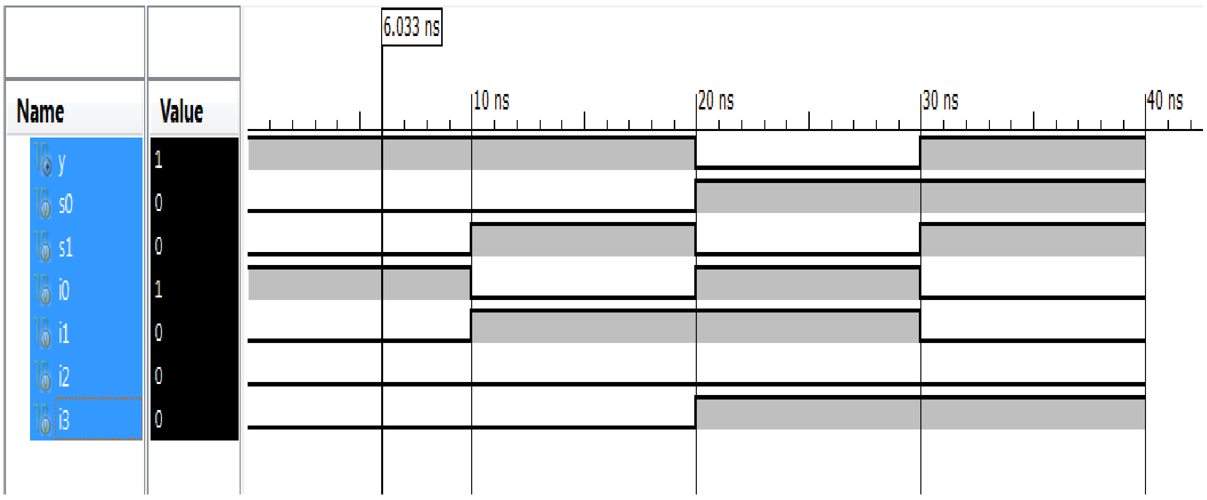
DATA FLOW LEVEL MODELING:

```
module mux (i0,i1,i2,i3,s0,s1,y);
input s0,s1,i0,i1,i2,i3;
output  y;
assign y= (~s0 & ~s1 &i0),
             (~s0 & s1 &i1),
             (s0 & ~s1 &i2),
             (s0 & s1 &i3);
endmodule
```

Conditional statement

```
module mux (i0,i1,i2,i3,s0,s1,y);
input s0,s1,i0,i1,i2,i3;
output  y;
assign y= s1?(s0?i3:i2)☹s0?i1:i0);
endmodule
```

WAVEFORM:



RESULT:

Thus, the multiplexer are simulated and synthesized using Verilog HDL. And implement in FPGA.

Expt.No: 12 DEMULTIPLEXER

AIM:

To implement demultiplexer using Verilog HDL.

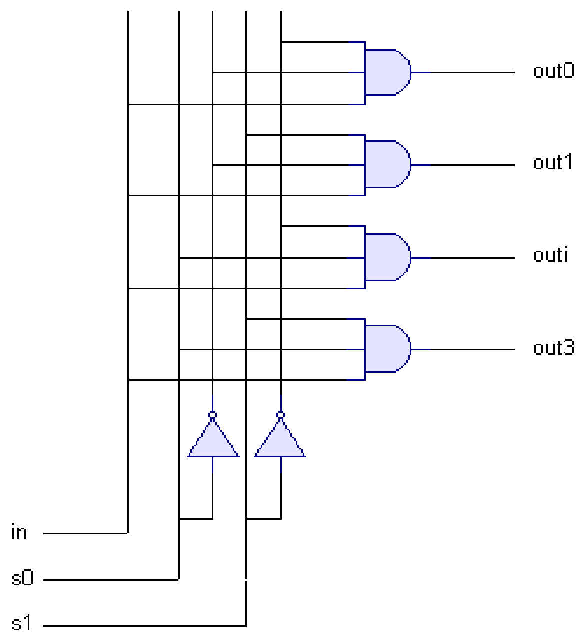
APPARATUS REQUIRED:

PC with Windows XP.
XILINX, ModelSim software.
FPGA kit.
RS 232 cable.

PROCEDURE:

- Write and draw the Digital logic system.
- Write the Verilog code for above system.
- Enter the Verilog code in Xilinx software.
- Check the syntax and simulate the above verilog code (using ModelSim or Xilinx) and verify the output waveform as obtained.
- Implement the above code in Spartan II using FPGA kit.

Demultiplexer



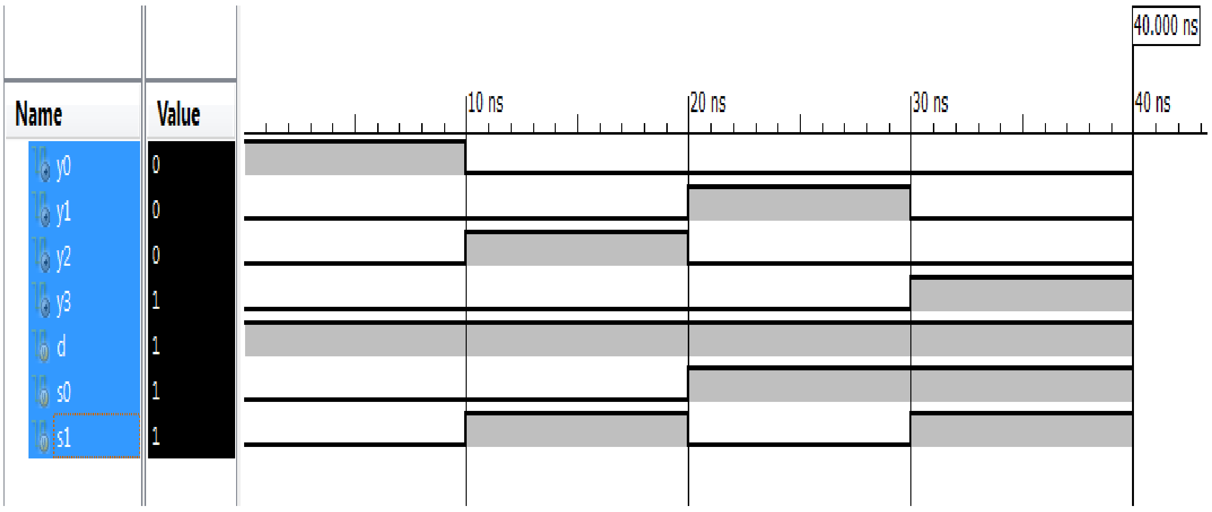
S0	S1	OUTPUT
0	0	Y0=1000
0	1	Y1=0100
1	0	Y2=0010
1	1	Y3=0001

PROGRAM

BEHAVIORAL LEVEL MODELING:

```
module  demux (d,s,y);
input d;
input [2:0]s;
output reg [3:0]y;
always @ (s or d)
case ({s})
2'b00 : begin y[0]=d; y[1]=0; y[2]=0; y[3]=0; end
2'b01 : begin y[0]=0; y[1]=d; y[2]=0; y[3]=0; end
2'b10 : begin y[0]=0; y[1]=0; y[2]=d; y[3]=0; end
2'b11 : begin y[0]=0; y[1]=0; y[2]=0; y[3]=d; end
endcase
endmodule
```

WAVEFORM:



RESULT:

Thus, the demultiplexer are simulated and synthesized using Verilog HDL. And implement in FPGA.

Expt.No: 13 8:3 ENCODER

AIM:

To implement 8:3 encoder using Verilog HDL.

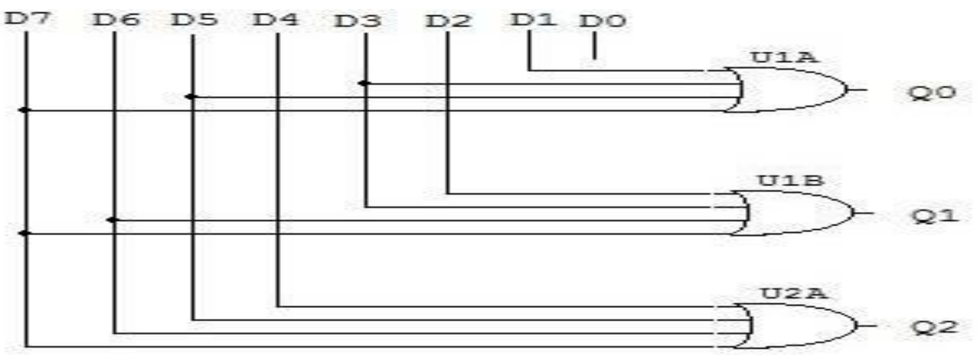
APPARATUS REQUIRED:

PC with Windows XP.
XILINX, ModelSim software.
FPGA kit.
RS 232 cable.

PROCEDURE:

- Write and draw the Digital logic system.
- Write the Verilog code for above system.
- Enter the Verilog code in Xilinx software.
- Check the syntax and simulate the above verilog code (using ModelSim or Xilinx) and verify the output waveform as obtained.
- Implement the above code in Spartan II using FPGA kit.

8:3 encoder



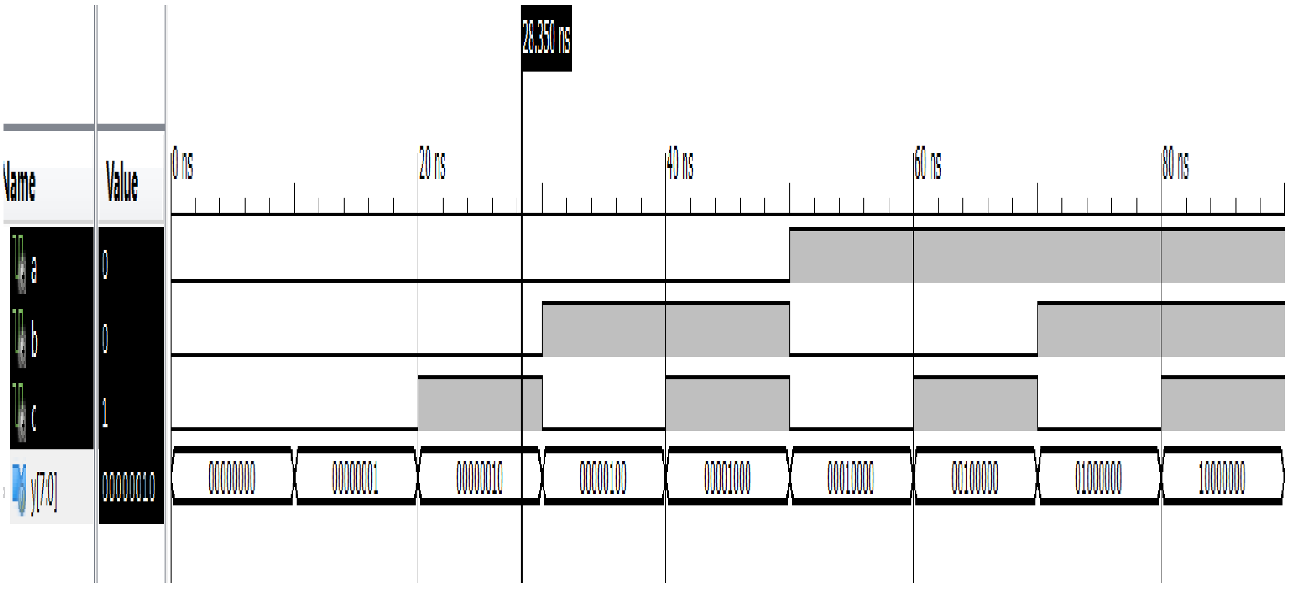
Input								Output		
D0	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**X**	**Y**	**Z**
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

PROGRAM

GATE LEVEL MODELING:

```
module encoder (a,b,c,y);
input [7:0]y;
output  a,b,c;
or (a,y[4],y[5],y[6],y[7]);
or (b,y[2],y[3],y[6],y[7]);
or (c,y[1],y[3],y[5],y[7]);
endmodule
```

WAVEFORM:



RESULT:

Thus, the 8:3 encoder are simulated and synthesized using Verilog HDL. And implement in FPGA.

Expt.No: 14 3:8 DECODER

Date :

AIM:

To implement 3:8 decoder using Verilog HDL.

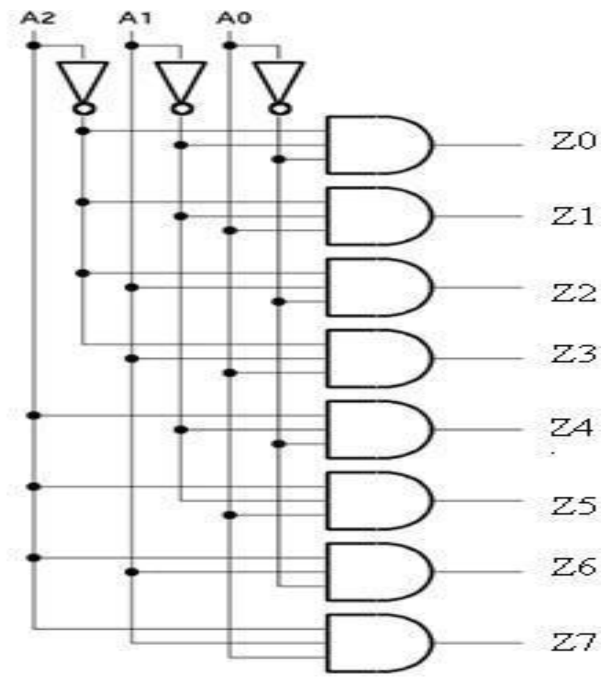
APPARATUS REQUIRED:

PC with Windows XP.
XILINX, ModelSim software.
FPGA kit.
RS 232 cable.

PROCEDURE:

- Write and draw the Digital logic system.
- Write the Verilog code for above system.
- Enter the Verilog code in Xilinx software.
- Check the syntax and simulate the above verilog code (using ModelSim or Xilinx) and verify the output waveform as obtained.
- Implement the above code in Spartan II using FPGA kit.

3:8 decoder



Output :

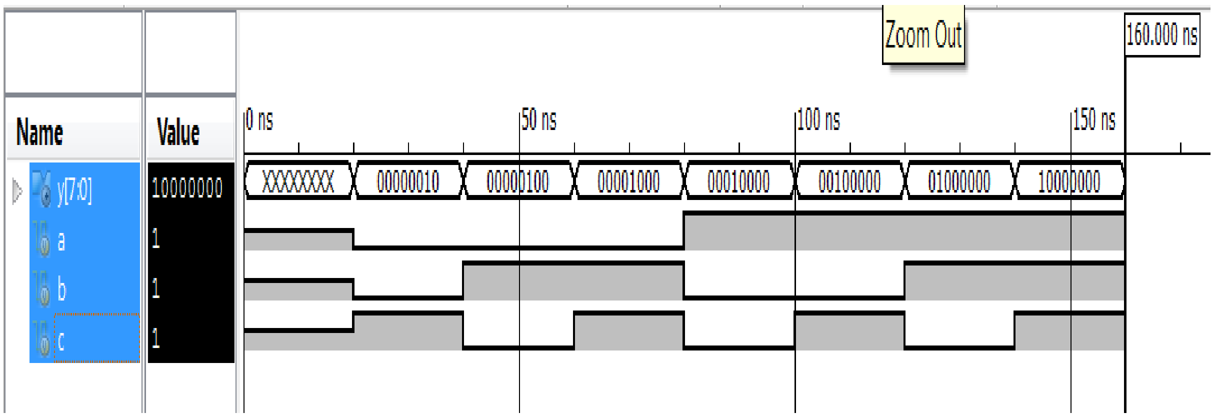
Output								Input		
D0	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**X**	**Y**	**Z**
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

PROGRAM

DATA FLOW LEVEL MODELING:

```
module decoder (a,b,c,y);
output  [7:0]y;
input  a,b,c;
assign y[0]= ~a  & ~b & ~c;
assign y[1]= ~a  & ~b & c;
assign y[2]= ~a  & b & ~c;
assign y[3]= ~a  & b & c;
assign y[4]= a  & ~b & ~c;
assign y[5]= a  & ~b & c;
assign y[6]= a  & b & ~c;
assign y[7]= a  & b & c;
endmodule
```

WAVEFORM:



RESULT:

Thus, the 3:8 decoder are simulated and synthesized using Verilog HDL. And implement in FPGA.

SIMULATE AND VERIFY 16 BIT ARITHMETIC LOGIC UNITS USING VERILOG CODE IN XILINX

EXP NO: 15

AIM: To simulate and verify the operation of the 16 bit arithmetic logic unit using Verilog code.

COMPONENTS REQUIRED:

- Xilinx
- FPGA kit
- System

PROCEDURE:

1. Verilog language was used to code the given module.
2. Corresponding input and clock signal given
3. Program compiled for the error.
4. Then it is simulated for the given data.
5. Corresponding output waveform obtained and verified.

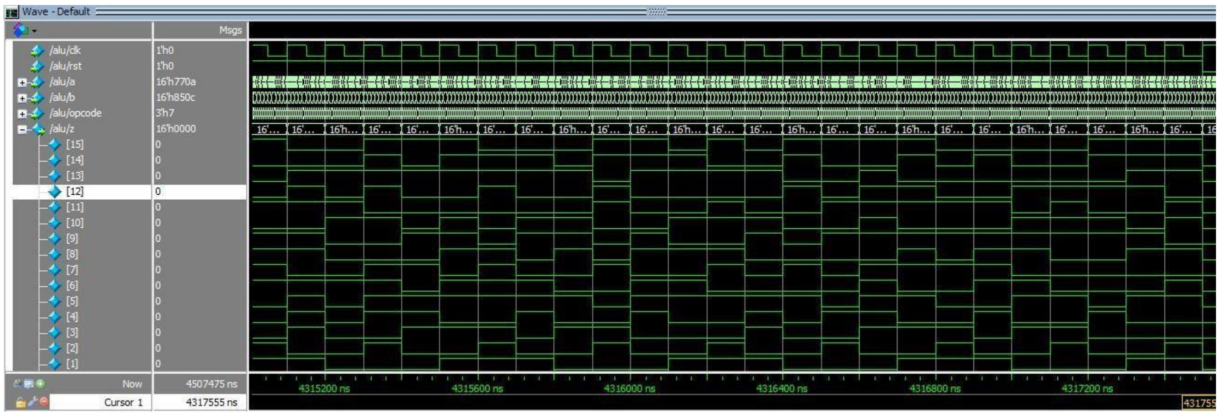
PROGRAM:

```
module alu(clk,rst,a,b,opcode,z);
input clk;
input rst;
input [15:0]a,b;
input [2:0]opcode;
output [15:0]z;
reg [15:0]z;
always @(posedge clk or negedge rst)
begin
        if(~rst)
                z <= 16'b0;
        else
                case(opcode)
                        3'd0:z <= a + b;
                        3'd1:z <= a - b;
                        3'd2:z <= a * b;
                        3'd3:z <= a / b;
                        3'd4:z <= a & b;
                        3'd5:z <= a | b;
                        3'd6:z <= a ^ b;
```

```
                    3'd7:z <= ~a;
            endcase
end

endmodule
```

RESULTANT WAVE:


```
Wave - Default
                 Msgs
/alu/clk         1'h0
/alu/rst         1'h0
/alu/a           16'h770a
/alu/b           16'h850c
/alu/opcode      3'h7
/alu/z           16'h0000
  [15]           0
  [14]           0
  [13]           0
  [12]           0
  [11]           0
  [10]           0
  [9]            0
  [8]            0
  [7]            0
  [6]            0
  [5]            0
  [4]            0
  [3]            0
  [2]            0
  [1]            0
Now              4507475 ns
Cursor 1         4317555 ns
```

RESULT:

Thus the operation was verified with the resultant waveform.

SIMULATE AND VERIFY UP-DOWN COUNTER USING VERILOG CODE IN XILINX

EXP NO: 16

AIM: To simulate and verify the operation of up-down counter using Verilog language in xilinx tool.

COMPONENTS REQUIRED:

- Xilinx
- FPGA kit
- System

PROCEDURE:

1. Verilog language was used to code the given module.
2. Corresponding input and clock signal given
3. Program compiled for the error.
4. Then it is simulated for the given data.
5. Corresponding output waveform obtained and verified.

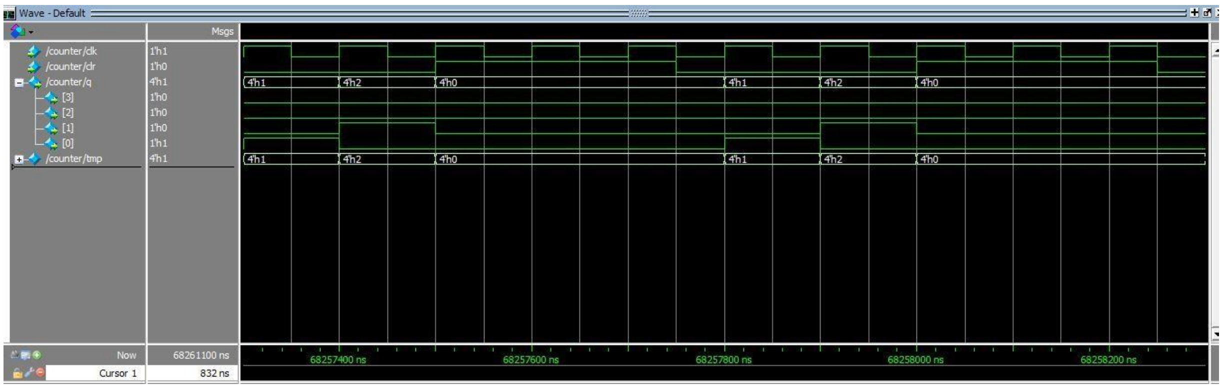
PROGRAM:

```
module up_down_counter(clk,rst,upn_down,out);
input clk;
input rst;
input upn_down;
output [3:0]out;
reg [3:0]count_reg;

always @(posedge clk or negedge rst)
begin
        if(~rst)
                count_reg <= 4'b0;
        else if(upn_down) //down counter
                count_reg <= count_reg - 1'b1;
        else                                    //up counter
                count_reg <= count_reg + 1'b1;
end

assign out = count_reg;
endmodule
```

RESULTANT WAVE:



RESULT:

Thus, the operation of the up – down counter was verified using xilinx tool with the help of resultant waveform.

Expt No:17 **DESIGN OF MAGNITUDE COMPARATOR**

Aim:

Design 4 bit Magnitude Comparator using Verilog and Verify with Test Bench

COMPONENTS REQUIRED:

- Xilinx
- FPGA kit
- System

PROCEDURE:

6. Verilog language was used to code the given module.
7. Corresponding input and clock signal given
8. Program compiled for the error.
9. Then it is simulated for the given data.
10. Corresponding output waveform obtained and verified.

Block diagram

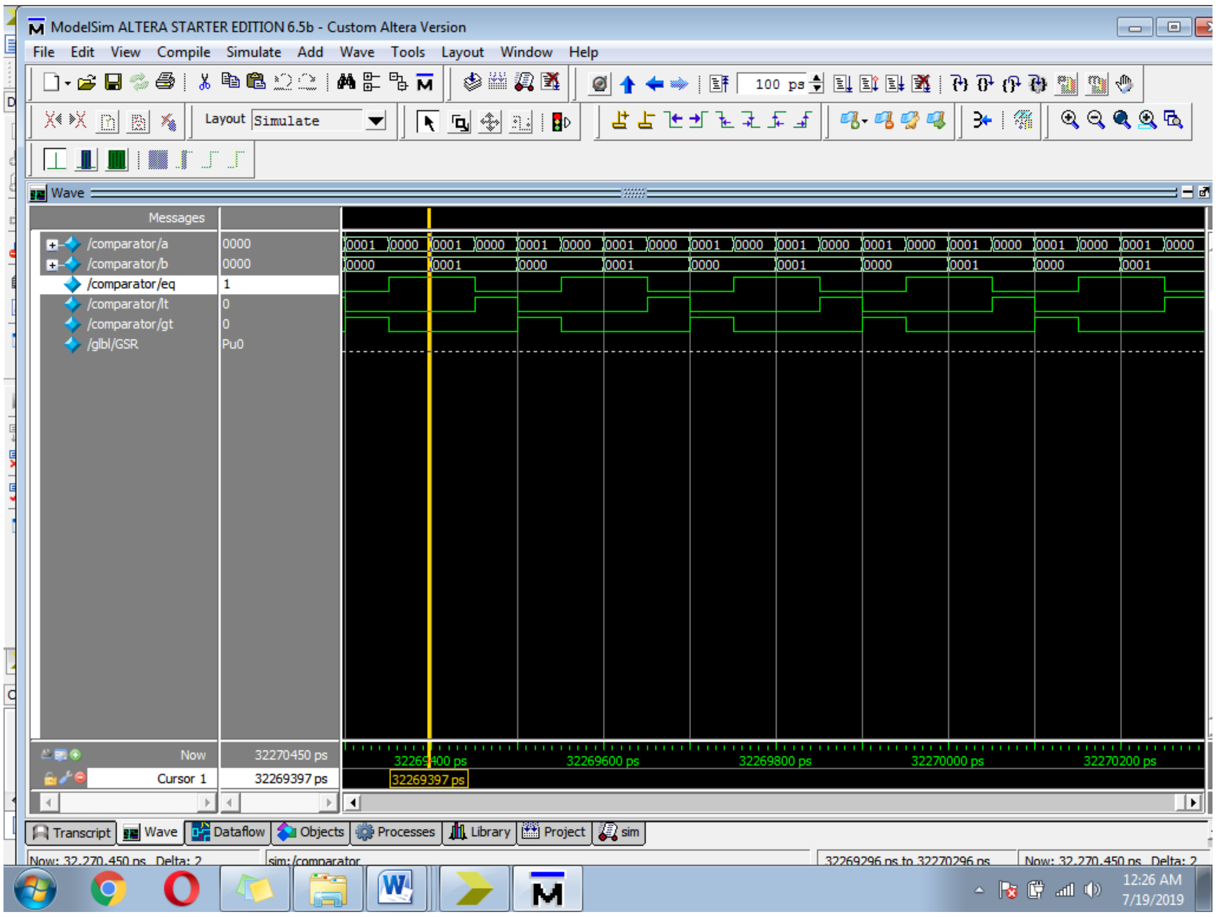
```
module comparator(a,b,eq,lt,gt);

input [3:0] a,b;

output reg eq,lt,gt;

always @(a,b)
begin
 if (a==b)
 begin
  eq = 1'b1;
 lt = 1'b0;
 gt = 1'b0;
 end
 else if (a>b)
 begin
  eq = 1'b0;
  lt = 1'b0;
  gt = 1'b1;
 end
 else
 begin
  eq = 1'b0;
 lt = 1'b1;
 gt = 1'b0;
 end
end
endmodule
```

RESULTANT WAVE:



ModelSim ALTERA STARTER EDITION 6.5b - Custom Altera Version

```
Messages
/comparator/a    0000
/comparator/b    0000
/comparator/eq   1
/comparator/lt   0
/comparator/gt   0
/glbl/GSR        Pu0

Now        32270450 ps
Cursor 1   32269397 ps
```

RESULT:

Thus, the operation of Magnitude Comparator was verified using xilinx tool with the help of resultant waveform.

EXPT NO:18 DESIGN OF 4 BIT RIPPLE COUNTER

Aim:

Design a 4 bit ripple carry counter using T Flip flop using the following features,

1] Negative edge-triggered.
2] Asynchronous active-low reset.
3] 4-bit Type.

COMPONENTS REQUIRED:

- Xilinx
- FPGA kit
- System

PROCEDURE:

1. Verilog language was used to code the given module.
2. Corresponding input and clock signal given
3. Program compiled for the error.
4. Then it is simulated for the given data.
5. Corresponding output waveform obtained and verified.

Ripple carry counter:

```
module ripple_carry_counter(q, clk, reset);
output [3:0] q;
input clk, reset;
T_FF tff0(q[0], clk, reset);
T_FF tff1(q[1], q[0], reset);
T_FF tff2(q[2], q[1], reset);
T_FF tff3(q[3], q[2], reset);
endmodule
```

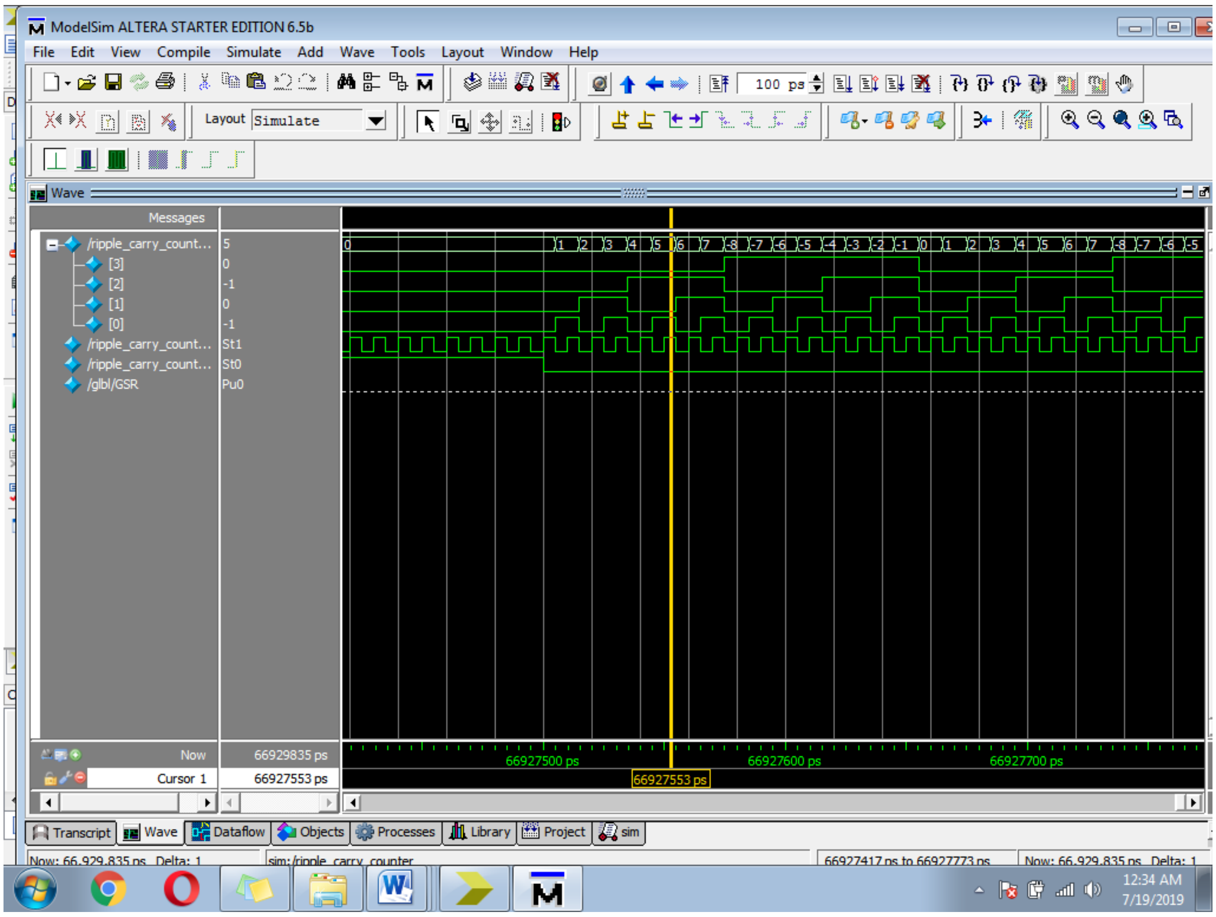
T flip-flop:

```
module T_FF(q, clk, reset);
output q;
input clk, reset;
wire d;
D_FF dff0(q, d, clk, reset);
not n1(d, q); // not is Verilog-provided primitive. Case sensitive.
endmodule
```

D flip-flop:

```
module D_FF(q, d, clk, reset);
output q;
input d, clk, reset;
reg q;
always @(posedge reset or negedge clk)
if (reset)
q = 1'b0;
else
q = d;
endmodule
```

RESULTANT WAVE:


ModelSim ALTERA STARTER EDITION 6.5b
File Edit View Compile Simulate Add Wave Tools Layout Window Help
Layout Simulate
Wave
Messages
/ripple_carry_count... 5
[3] 0
[2] -1
[1] 0
[0] -1
/ripple_carry_count... St1
/ripple_carry_count... St0
/glbl/GSR Pu0
Now 66929835 ps
Cursor 1 66927553 ps
66927500 ps
66927600 ps
66927700 ps
66927553 ps
Transcript Wave Dataflow Objects Processes Library Project sim
Now: 66,929,835 ns Delta: 1
sim:/ripple_carry_counter
66927417 ps to 66927773 ps
12:34 AM
7/19/2019

RESULT:

Thus, the 4 bit ripple carry counter was verified using xilinx tool with the help of resultant waveform.

EX. NO 19 DESIGN OF PRIORITY ENCODER

Aim:

To design and verify the 3 bit 1 of 9 priority encoder using behavioural model and verify the results.

COMPONENTS REQUIRED:

- Xilinx
- FPGA kit
- System

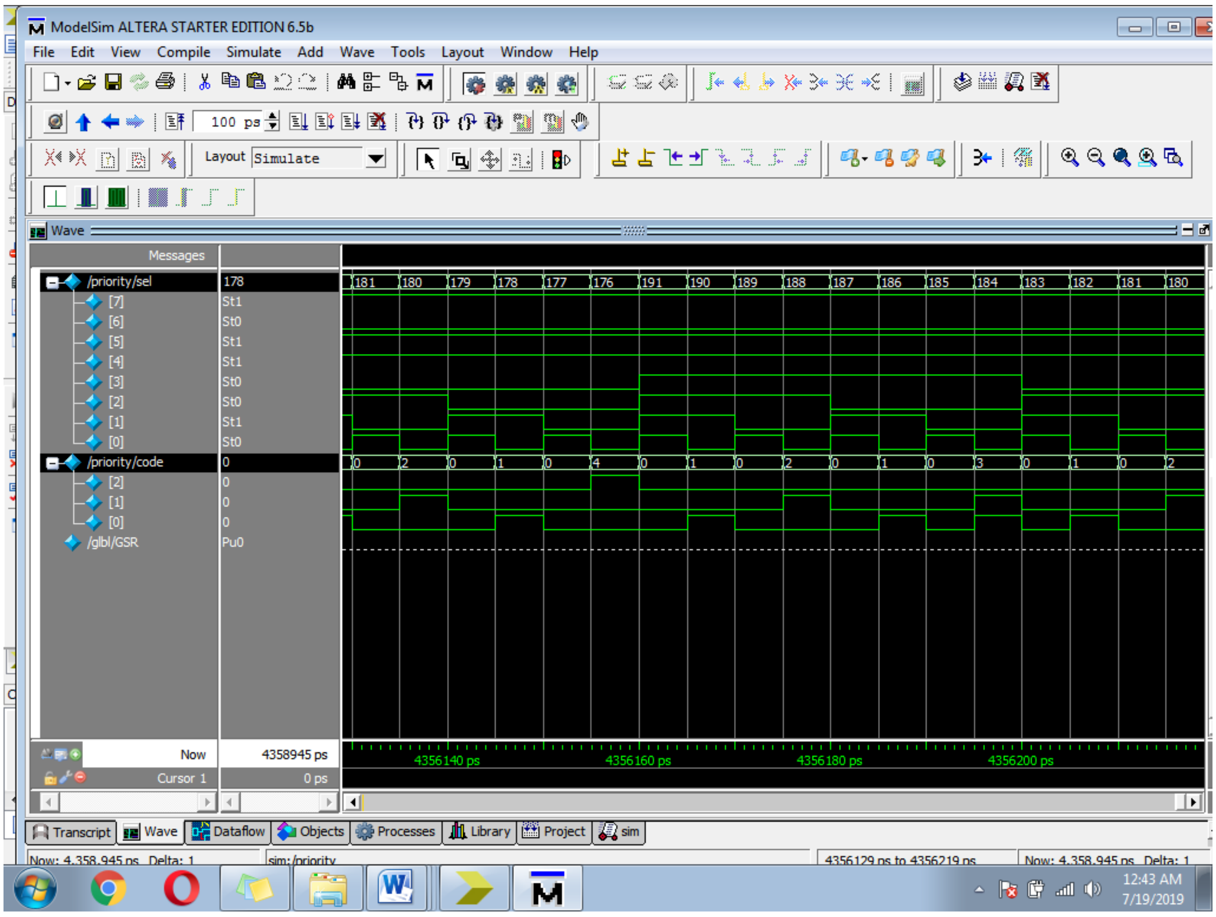
PROCEDURE:

6. Verilog language was used to code the given module.
7. Corresponding input and clock signal given
8. Program compiled for the error.
9. Then it is simulated for the given data.
10. Corresponding output waveform obtained and verified.

```
module priority (sel, code);
  input [7:0] sel;
  output [2:0] code;
  reg [2:0] code;

  always @(sel)
  begin
        if (sel[0]) code <= 3'b000;
   else if (sel[1]) code <= 3'b001;
   else if (sel[2]) code <= 3'b010;
   else if (sel[3]) code <= 3'b011;
   else if (sel[4]) code <= 3'b100;
   else if (sel[5]) code <= 3'b101;
   else if (sel[6]) code <= 3'b110;
   else if (sel[7]) code <= 3'b111;
   else code <= 3'bxxx;
  end
endmodule
```

RESULTANT WAVE:



RESULT:

Thus, priority encoder was verified using xilinx tool with the help of resultant waveform.

TESTING ON FPGA BOARD LEDS AND SWITCHES USING VHDL CODES

EXP NO: 20

AIM: To test and verify the operation of the LEDs using switches present in the Spartan6 FPGA board.

COMPONENTS REQUIRED:

- Xilinx
- FPGA kit
- System

PROCEDURE:

1. Xilinx software was initialised with the Spartan 6 FPGA kit.
2. VHDL module was selected
3. New project with new file name created.
4. VHDL code was written and compiled.
5. Its corresponding UCF (User Constraint File) also coded.
6. After successful compilation corresponding bit file generated automatically.
7. This bit file was burned in the temporary module of the FPGA kit.
8. Now the FPGA kit was interfaced with the Xilinx and we can verify the operation performed by the VHDL code.

PROGRAM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity sw_led is
port (   sw  : in std_logic_vector(15 downto 0);
                        led : out std_logic_vector(15 downto 0)
                        );
end sw_led;
```

```
architecture Behavioral of sw_led is
begin
led <= sw;
end Behavioral;
```

UCF FILE:

```
net     "sw<15>"        loc     =       "B8";   #SW22
net     "sw<14>"        loc     =       "D8";   #SW23
net     "sw<13>"        loc     =       "A7";   #SW24
net     "sw<12>"        loc     =       "F7";   #SW25
net     "sw<11>"        loc     =       "B6";   #SW26
net     "sw<10>"        loc     =       "D6";   #SW27
net     "sw<9>"         loc     =       "F6";   #SW28
net     "sw<8>"         loc     =       "B5";   #SW29
net     "sw<7>"         loc     =       "D5";   #SW30
net     "sw<6>"         loc     =       "A4";   #SW31
net     "sw<5>"         loc     =       "A3";   #SW32
net     "sw<4>"         loc     =       "C3";   #SW33
net     "sw<3>"         loc     =       "B2";   #SW34
net     "sw<2>"         loc     =       "C2";   #SW35
net     "sw<1>"         loc     =       "D3";   #SW36
net     "sw<0>"         loc     =       "E3";   #SW37

net     "led<0>"        loc     =       "D1";   #L55
net     "led<1>"        loc     =       "C1";   #L54
net     "led<2>"        loc     =       "B1";   #L53
net     "led<3>"        loc     =       "A2";   #L52
net     "led<4>"        loc     =       "B3";   #L51
net     "led<5>"        loc     =       "E4";   #L50
net     "led<6>"        loc     =       "F5";   #L49
net     "led<7>"        loc     =       "C5";   #L48
net     "led<8>"        loc     =       "A5";   #L47
net     "led<9>"        loc     =       "E6";   #L46
net     "led<10>"   loc     =       "C6";   #L45
net     "led<11>"   loc     =       "A6";   #L44
net     "led<12>"   loc     =       "C7";   #L43
net     "led<13>"   loc     =       "E8";   #L42
net     "led<14>"   loc     =       "C8";   #L41
net     "led<15>"   loc     =       "A8";   #L40
```

RESULT:

Thus, the operation of LEDs was verified in the Spartan 6 FPGA kit and the application of the board were studied.

TESTING OF BUZZER ON FPGA BOARD

EXP NO: 21

AIM: To simulate the buzzer operation using Spartan 6 FPGA board with VHDL code.

COMPONENTS REQUIRED:

- Xilinx
- FPGA kit
- System

PROCEDURE:

1. Xilinx software was initialised with the Spartan 6 FPGA kit.
2. VHDL module was selected
3. New project with new file name created.
4. VHDL code was written and compiled.
5. Its corresponding UCF (User Constraint File) also coded.
6. After successful compilation corresponding bit file generated automatically.
7. This bit file was burned in the temporary module of the FPGA kit.
8. Now the FPGA kit was interfaced with the Xilinx and we can verify the operation performed by the VHDL code.

PROGRAM:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity buzzer is
port (   clk : in std_logic;
                        buzzer : out std_logic
                        );
end buzzer;

architecture Behavioral of buzzer is
```

```
signal k : integer := 0;
signal clk_div : std_logic := '0';
signal delay : std_logic_vector(1 downto 0);

begin

process(clk)
begin

if rising_edge(clk) then

        k <= k + 1;
                if k = 10000000 then
                        clk_div <= not(clk_div);
                        k <= 0;
                end if;
end if;

if rising_edge(clk_div) then

delay   <= delay + 1;
        case delay(1 downto 0) is
                when "00"       =>
                        buzzer <= '0';
                when "10"       =>
                        buzzer <= '1';
                when others =>
        end case;

--buzzer <= '1';

end if;

end process;

end Behavioral;
```

UCF CODE:

```
NET "clk"               LOC     = "E10"         ;
NET "buzzer" LOC        = "K2"  ;
```

RESULT:

Thus, the operation was verified.