Department of Electronics & Telecommunication Engineering

Program Outcomes

Graduates will be able to

1. Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems. [Engineering knowledge]
2. Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.[Problem analysis]
3. Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations. [Design/development of solutions]
4. Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions. [Conduct investigations of complex problems]
5. Create, select, and apply appropriate techniques, resources, and modem engineering and IT tools includin6 prediction and modelling to complex engineering activities with an understanding of the limitations. [Modem tool usage]
6. Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice. [The engineer and society]
7. Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development. [Environment and sustainability]
8. Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice. [Ethics]
9. Function effectively as an individual, and as a member or leader in diverse teams, and in multi-disciplinary settings. [Individual and team work]
10. Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions. [Communication]
11. Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments. [Project management and finance]
12. Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change. [Life-long learning]

INDEX

TITLE: INTRODUCTION TO XILINX	
NAME:	**SUBJECT: VLSI DESIGN AND TECHNOLOGY**
CLASS: BE	**ROLL NO:**
SEMESTER/YEAR: - 7TH SEM	**EXAM NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED: -	**REMARKS:**

EXPERIMENT NO – 01

AIM: INTRODUCTION TO XILINX

OBJECTIVE:

To study the procedure to write a VHDL code in XILINX and various operations for its analysis

THEORY:

This experiment provides Xilinx PLD designers with a quick overview of the basic design process using ISE 8.1i or any other versions of it. After you have completed the tutorial, you will have an understanding of how to create, verify, and implement a design.

Note: This tutorial is designed for ISE 8.1i on Windows.

This tutorial contains the following sections:

• "Getting Started"
• "Create a New Project"
• "Create an HDL Source"
• "Design Simulation"
• "Create Timing Constraints"
• "Implement Design and Verify Constraints"
• "Reimplement Design and Verify Pin Locations"
• "Download Design to the Spartan™-3 Demo Board"

Getting Started

Software Requirements

To use this tutorial, you must install the following software:

• ISE 8.1i

Hardware Requirements

To use this tutorial, you must have the following hardware:

• Spartan-3 Start-up Kit, containing the Spartan-3 Start-up Kit Demo Board

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 8.1i → Project Navigator
Note: Your start-up path is set during the installation process and may differ from the one above.

Accessing Help

At any time during the tutorial, you can access online help for additional information about the ISE software and related tools.

To open Help, do either of the following:

• Press F1 to view Help for the specific tool or function that you have selected or highlighted.
• Launch the ISE Help Contents from the Help menu. It contains information about creating and maintaining your complete design flow in ISE.

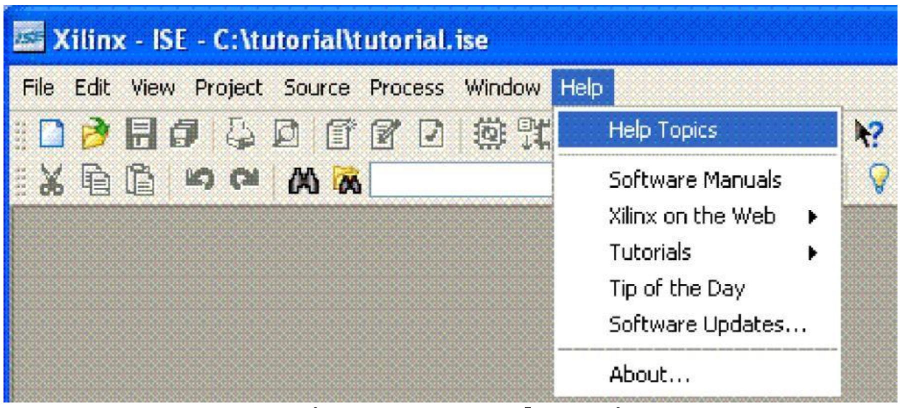


Figure 1: ISE Help Topics

Create a New Project

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

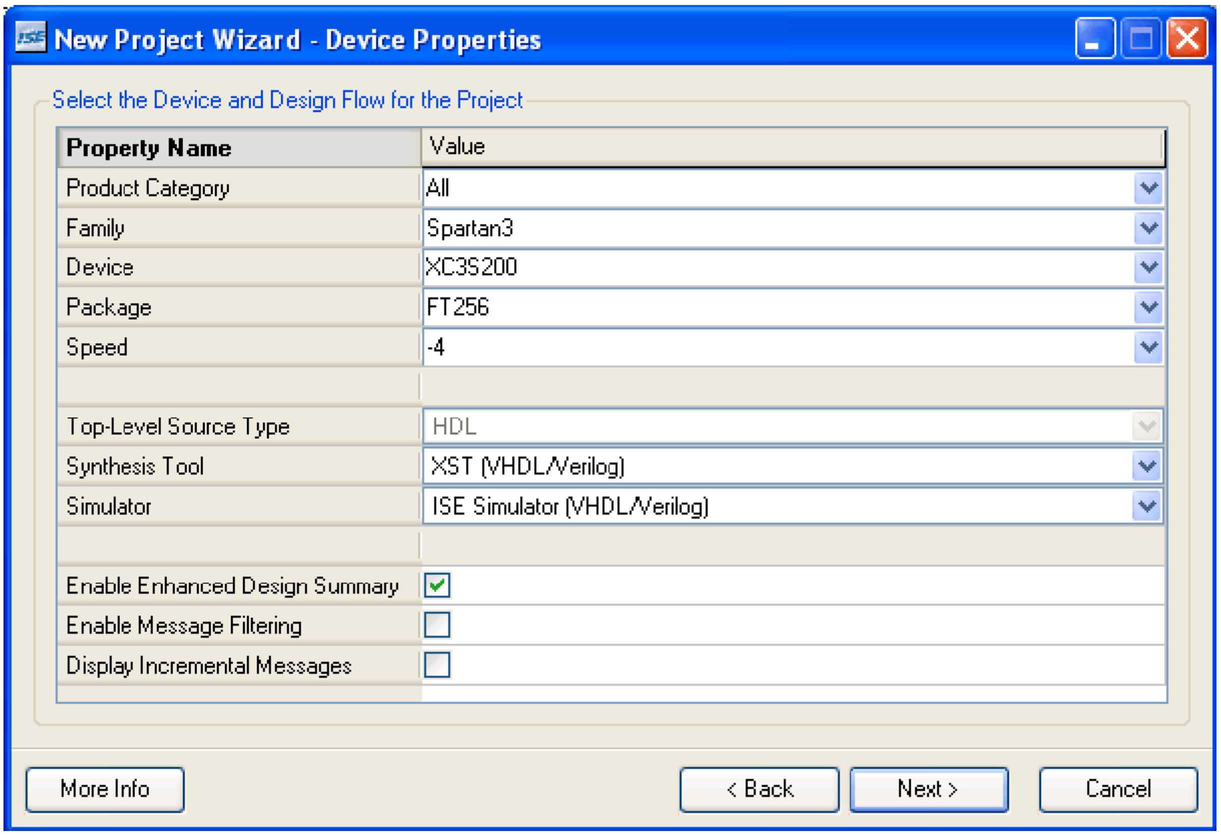
To create a new project:

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

♦ Product Category: All
♦ Family: Spartan3
♦ Device: XC3S200
♦ Package: FT256
♦ Speed Grade: -4
♦ Top-Level Module Type: HDL
♦ Synthesis Tool: XST (VHDL/Verilog)
♦ Simulator: ISE Simulator (VHDL/Verilog)
♦ Verify that Enable Enhanced Design Summary is selected.

Leave the default values in the remaining fields.

When the table is complete, your project properties will look like the following:


New Project Wizard - Device Properties
Select the Device and Design Flow for the Project

Property Name	Value
Product Category	All
Family	Spartan3
Device	XC3S200
Package	FT256
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISE Simulator (VHDL/Verilog)
Enable Enhanced Design Summary	✓
Enable Message Filtering	☐
Display Incremental Messages	☐

More Info
< Back
Next >
Cancel

Figure 2: Project Device Properties

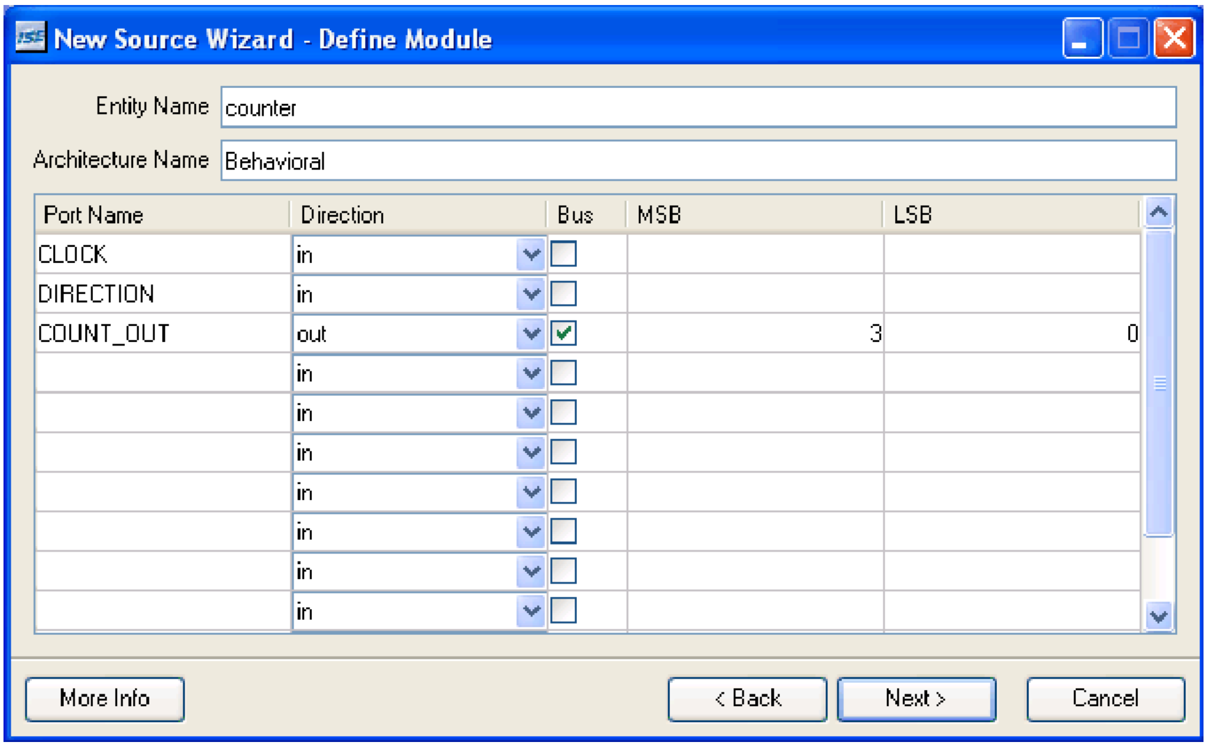
7. Click Next to proceed to the Create New Source window in the New Project Wizard. At the end of the next section, your new project will be complete.

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:


New Source Wizard - Define Module

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	☐		

Figure 3: Define Module

7. Click Next, then Finish in the New Source Information 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 Sources tab, as shown below:

Using Language Templates (VHDL)

The next step in creating the new source is to add the behavioral description for the counter. To do this you will use a simple counter code example from the ISE Language Templates and customize it for the counter design.
1. Place the cursor just below the begin statement within the counter architecture.
2. Open the Language Templates by selecting Edit → Language Templates…
Note: You can tile the Language Templates and the counter file by selecting Window → Tile Vertically to make them both visible.
3. Using the "+" symbol, browse to the following code example:
VHDL → Synthesis Constructs → Coding Examples → Counters → Binary → Up/Down Counters → Simple Counter
4. With Simple Counter selected, select Edit → Use in File, or select the Use Template in File toolbar button. This step copies the template into the counter source file.
5. Close the Language Templates.

Final Editing of the VHDL Source

1. Add the following signal declaration to handle the feedback of the counter output below the architecture declaration and above the first begin statement:
signal count_int : std_logic_vector(3 downto 0) := "0000";
2. Customize the source file for the counter design by replacing the port and signal name placeholders with the actual ones as follows:

♦ replace all occurrences of <clock> with CLOCK
♦ replace all occurrences of <count_direction> with DIRECTION
♦ replace all occurrences of <count> with count_int
3. Add the following line below the end process; statement:
COUNT_OUT <= count_int;
4. Save the file by selecting File → Save.
When you are finished, the counter source file will look like the following:

```
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 instantiating
-- any Xilinx primitive in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity counter is
Port ( CLOCK : in STD_LOGIC;
DIRECTION : in STD_LOGIC;
COUNT_OUT : out STD_LOGIC_VECTOR (3 downto 0));
end counter;
architecture Behavioral of counter is
signal count_int : std_logic_vector(3 downto 0) := "0000";
begin
process (CLOCK)
begin
if CLOCK='1' and CLOCK'event then
if DIRECTION='1' then
count_int <= count_int + 1;
else
count_int <= count_int - 1;
end if;
end if;
end process;
COUNT_OUT <= count_int;
end Behavioral;
```

You have now created the VHDL source for the tutorial project. Skip past the Verilog sections below, and proceed to the "Checking the Syntax of the New Counter Module"section.

Checking the Syntax of the New Counter Module

When the source files are complete, check the syntax of the design to find errors and typos.
1. Verify that Synthesis/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.

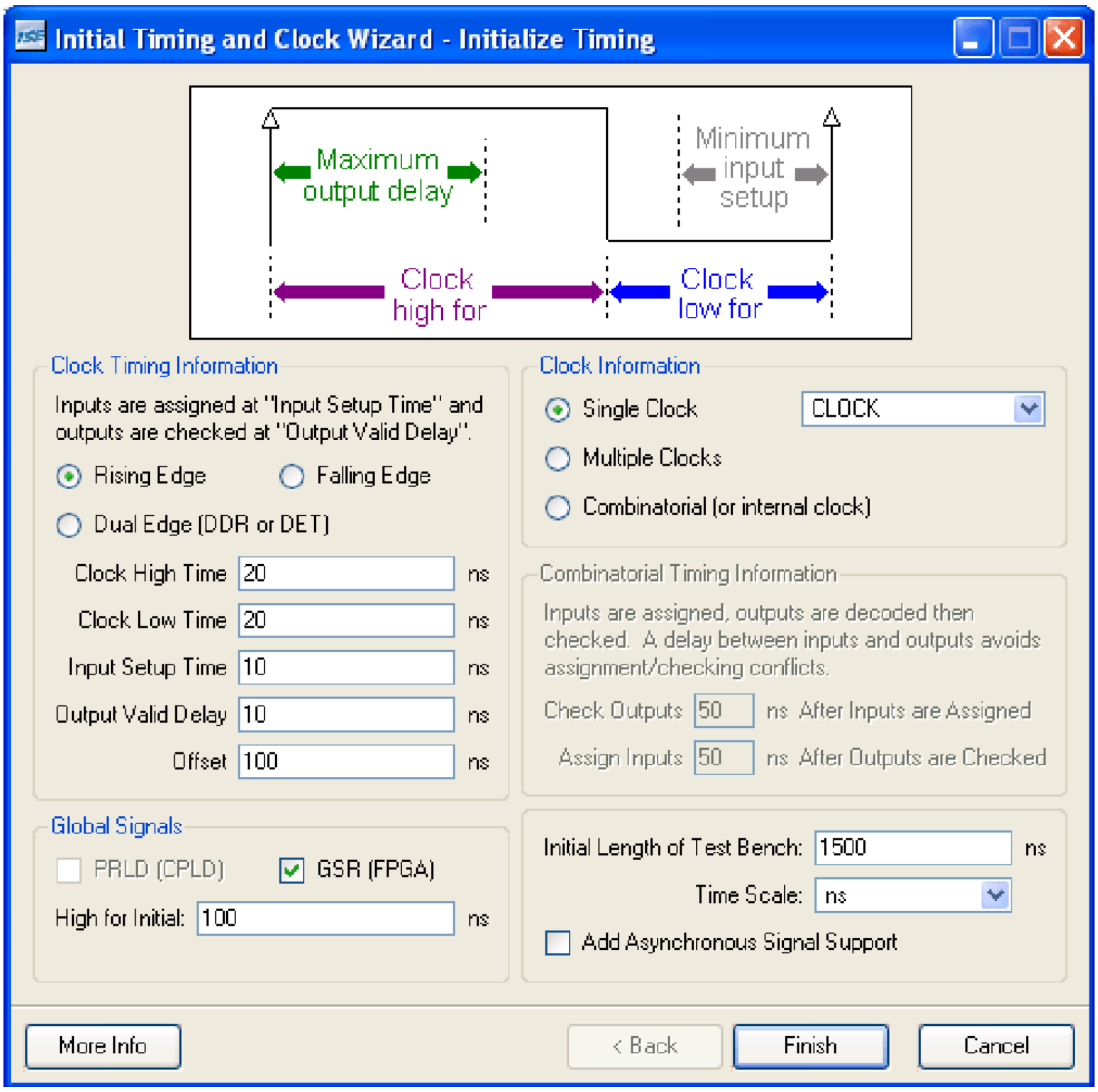
Fill in the fields in the Initialize Timing dialog box with the following information:

♦ Clock Time High: 20 ns.
♦ Clock Time Low: 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.

Leave the default values in the remaining fields.



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	100	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

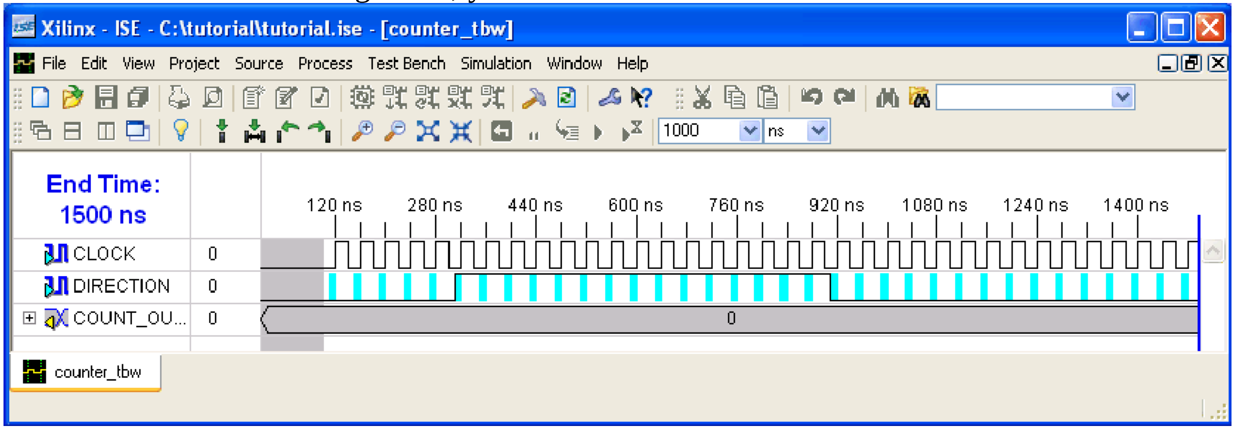
Figure 7: Initialize Timing

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 high so that the counter will count down.

Note: For more accurate alignment, you can use the Zoom In and Zoom Out toolbar buttons


Xilinx - ISE - C:\tutorial\tutorial.ise - [counter_tbw]
File Edit View Project Source Process Test Bench Simulation Window Help
1000
ns
120 ns
280 ns
440 ns
600 ns
760 ns
920 ns
1080 ns
1240 ns
1400 ns
0
counter_tbw

End Time: 1500 ns

CLOCK	0
DIRECTION	0
COUNT_OU...	0

Figure 8: Test Bench Waveform

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.

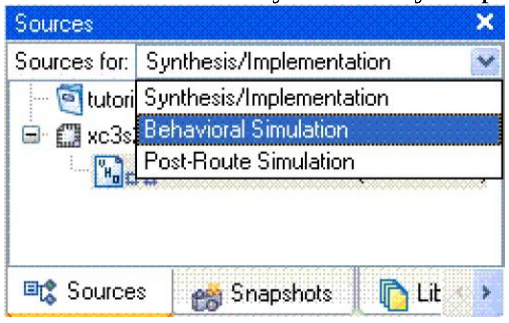
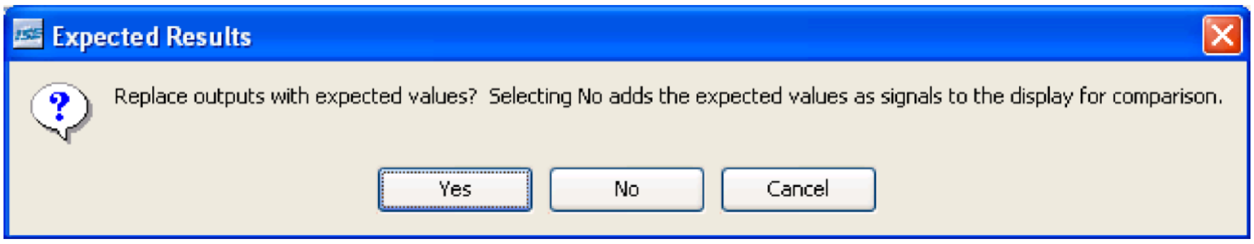


Figure 9: Behavior Simulation Selection

12. Close the test bench waveform.

Create a Self-Checking Test Bench Waveform

Add the expected output values to finish creating the test bench waveform. This transforms the test bench waveform into a self-checking test bench waveform. The key benefit to a self-checking test bench waveform is that it compares the desired and actual output values and flags errors in your design as it goes through the various transformations, from behavioral HDL to the device specific representation.
To create a self-checking test bench, edit output values manually, or run the Generate Expected Results process to create them automatically. If you run the Generate Expected Results process, visually inspect the output values to see if they are the ones you expected for the given set of input values.
To create the self-checking test bench waveform automatically, do the following:
1. Verify that Behavioral Simulation is selected from the drop-down list in the Sources window.
2. Select the counter_tbw file in the Sources window.
3. In the Processes tab, click the "+" to expand the Xilinx ISE Simulator process and double-click the Generate Expected Simulation Results process. This process simulates the design in a background process.
4. The Expected Results dialog box opens. Select Yes to annotate the results to the test bench.


Expected Results

Replace outputs with expected values? Selecting No adds the expected values as signals to the display for comparison.

Figure 10: Expected Results Dialog Box

5. Click the "+" to expand the COUNT_OUT bus and view the transitions that correspond to the Output Delay value (yellow cells) specified in the Initialize Timing dialog box.

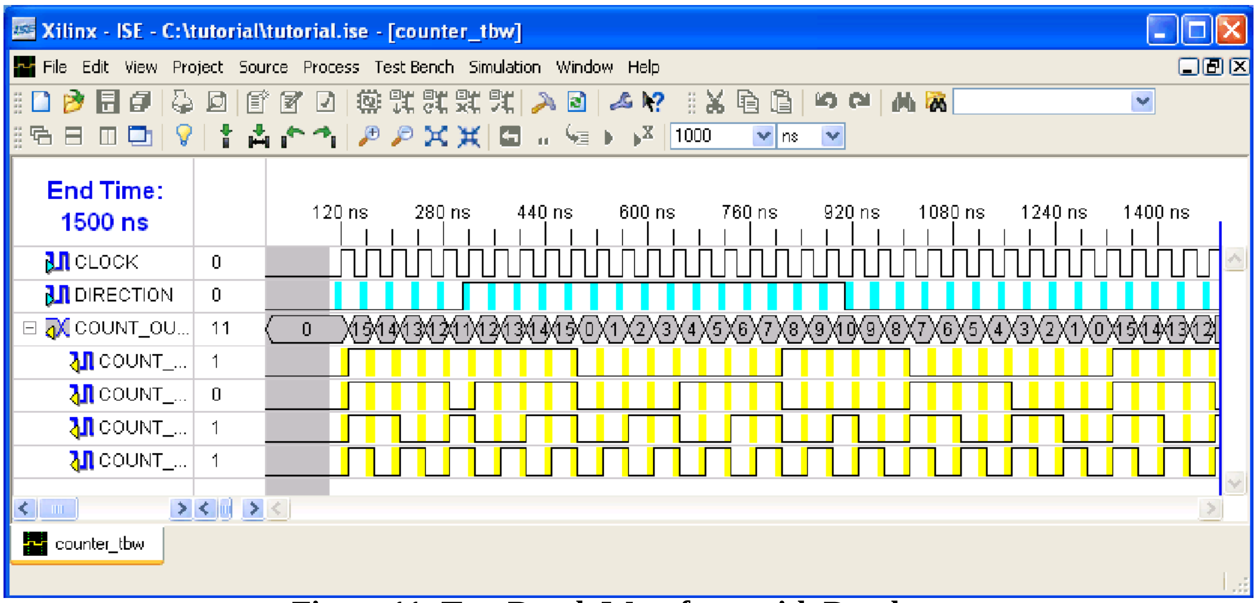

Xilinx - ISE - C:\tutorial\tutorial.ise - [counter_tbw]
File Edit View Project Source Process Test Bench Simulation Window Help
End Time: 1500 ns
CLOCK 0
DIRECTION 0
COUNT_OU... 11
COUNT_... 1
COUNT_... 0
COUNT_... 1
COUNT_... 1
counter_tbw

Figure 11: Test Bench Waveform with Results

6. Save the test bench waveform and close it.
You have now created a self-checking 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.
The simulation waveform results will look like the following:

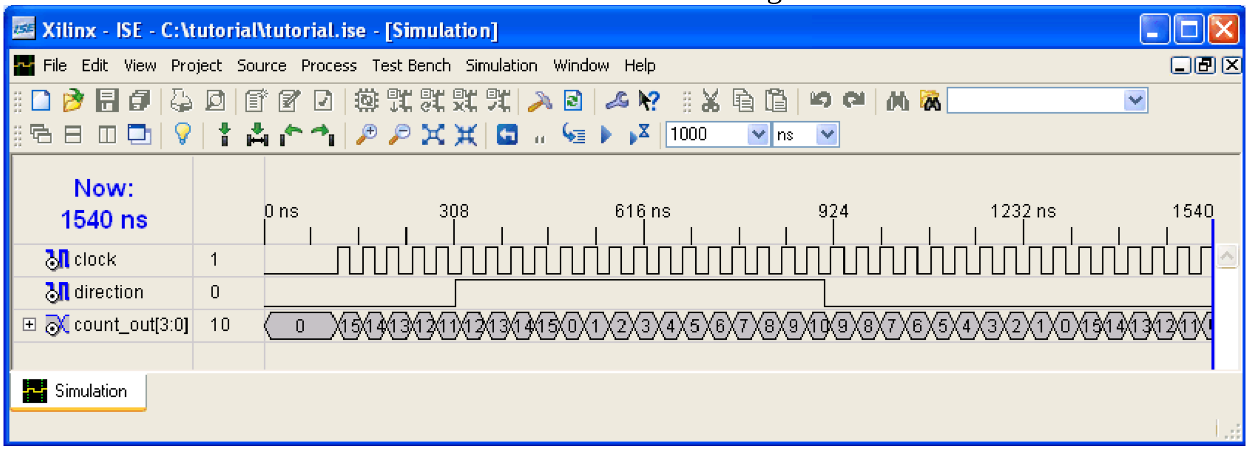

Xilinx - ISE - C:\tutorial\tutorial.ise - [Simulation]
File Edit View Project Source Process Test Bench Simulation Window Help
Now: 1540 ns
clock 1
direction 0
count_out[3:0] 10
Simulation

Figure 12: Simulation Results

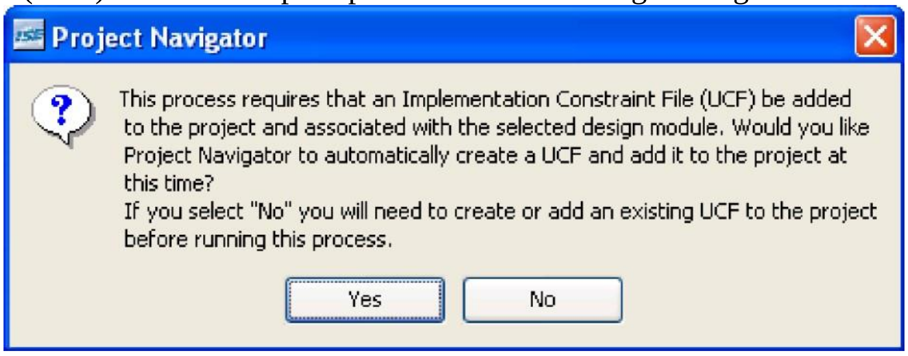
Note: You can ignore any rows that start with TX.
4. Verify that the counter is counting up and down as expected.
5. Close the simulation view. If you are prompted with the following message, "You have an active simulation open. Are you sure you want to close it?", click Yes to continue.
You have now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

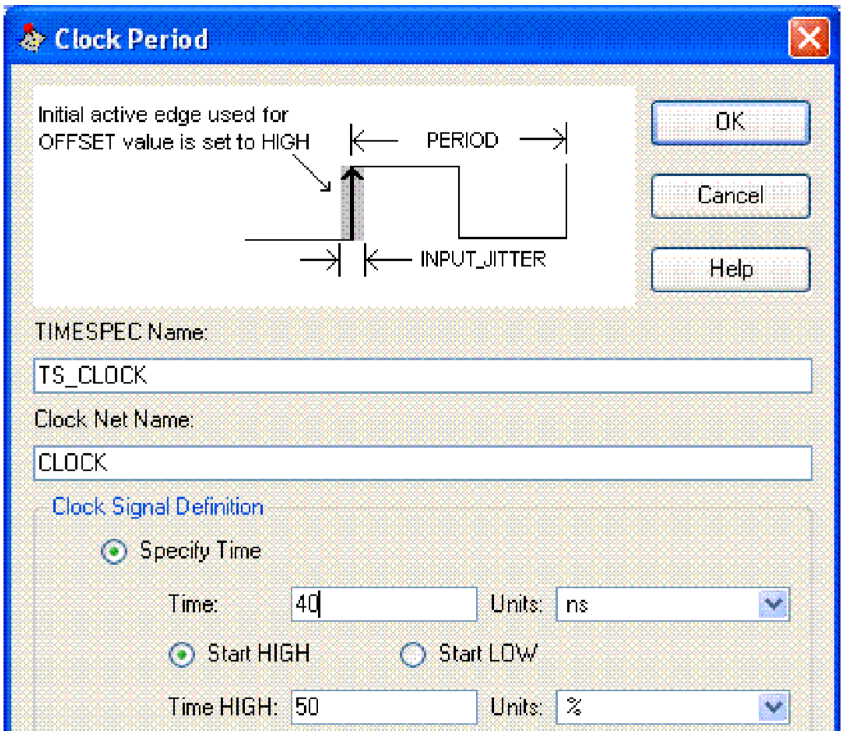

Project Navigator

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.

Figure 13: Prompt to Add UCF File to Project

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.
In the next step, enter values in the fields associated with CLOCK in the Constraints Editor Global tab.
5. Select CLOCK in the Clock Net Name field, then select the Period toolbar button or double-click the empty Period field to display the Clock Period dialog box.

6. Enter 40 ns in the Time field.


Clock Period

Initial active edge used for OFFSET value is set to HIGH

TIMESPEC Name: TS_CLOCK

Clock Net Name: CLOCK

Clock Signal Definition

[✓ Specify Time]

Time: 40 Units: ns

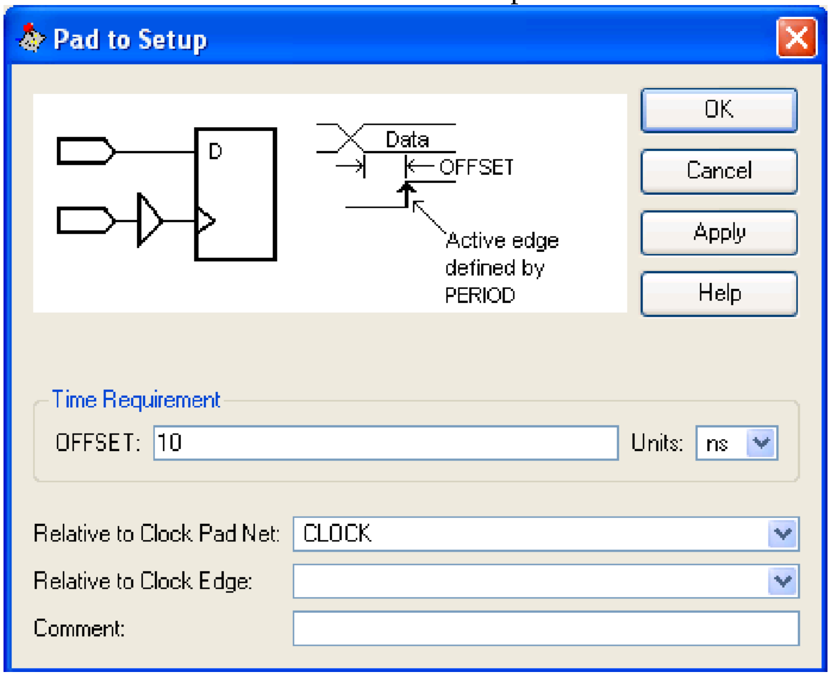
[✓ Start HIGH] Start LOW

Time HIGH: 50 Units: %

Figure 14: Clock Period

7. Click OK.
8. Select the Pad to Setup toolbar button or double-click the empty Pad to Setup field to display the Pad to Setup dialog box.

9. Enter 10 ns in the OFFSET field to set the input offset constraint.


Pad to Setup

Time Requirement

OFFSET: 10 Units: ns

Relative to Clock Pad Net: CLOCK

Relative to Clock Edge:

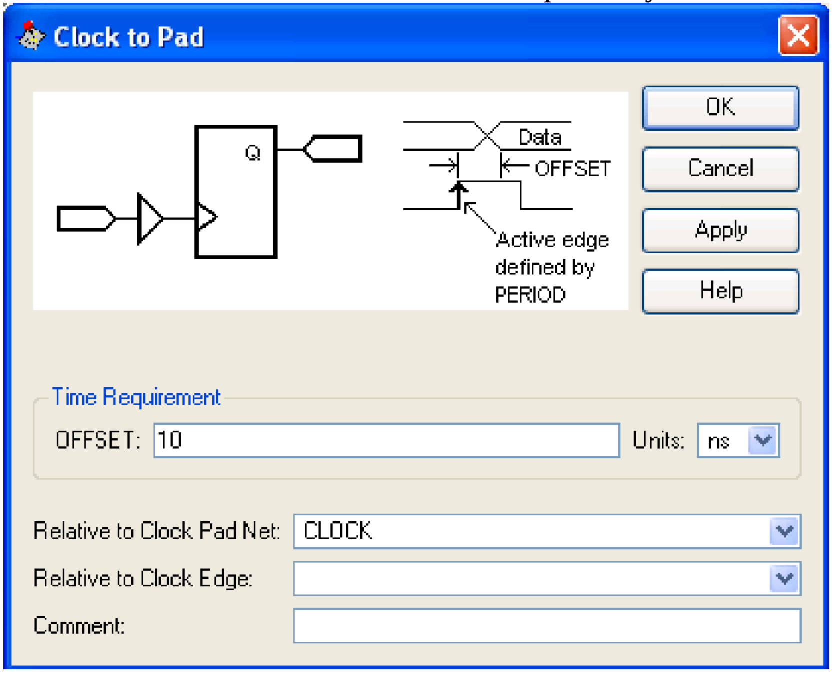
Comment:

Figure 15: Pad to Setup

10. Click OK.

11. Select the Clock to Pad toolbar button or double-click the empty Clock to Pad field to display the Clock to Pad dialog box.

12. Enter 10 ns in the OFFSET field to set the output delay constraint.



OFFSET: 10 Units: ns

Relative to Clock Pad Net: CLOCK

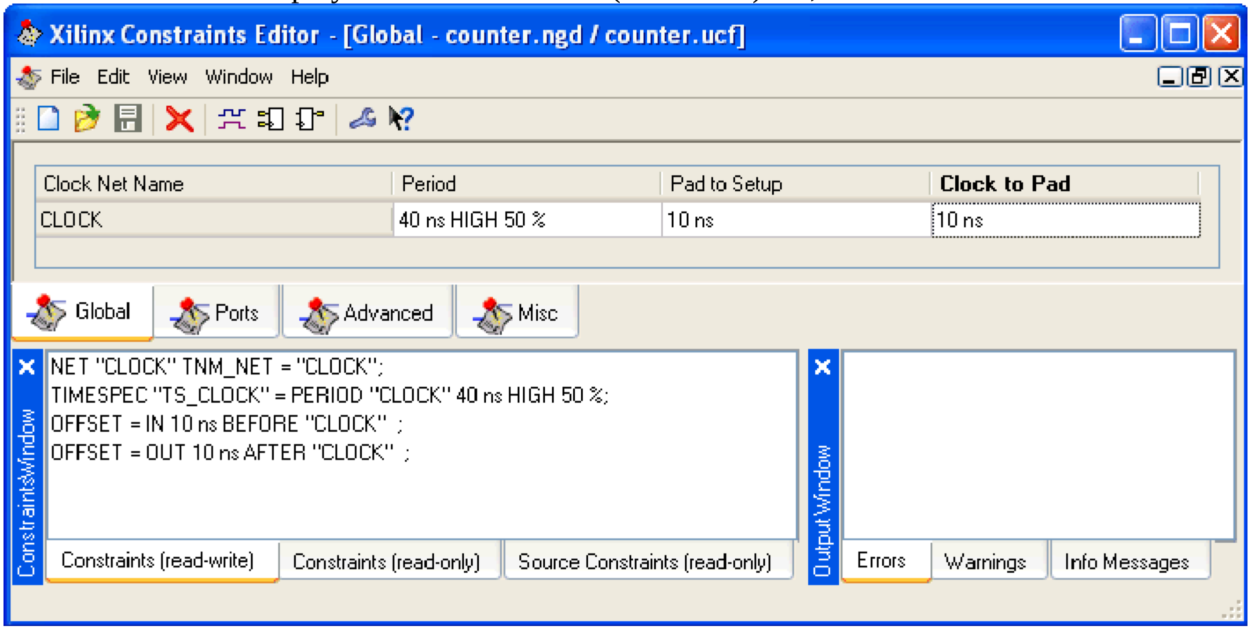
Relative to Clock Edge:

Comment:

Figure 16: Clock to Pad

13. Click OK.

The constraints are displayed in the Constraints(read-write) tab, as shown below:



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

```
NET "CLOCK" TNM_NET = "CLOCK";
TIMESPEC "TS_CLOCK" = PERIOD "CLOCK" 40 ns HIGH 50 %;
OFFSET = IN 10 ns BEFORE "CLOCK" ;
OFFSET = OUT 10 ns AFTER "CLOCK" ;
```

Figure 17: Timing Constraints

14. Save the timing constraints. If you are prompted to rerun the TRANSLATE or XST step, click OK to continue.

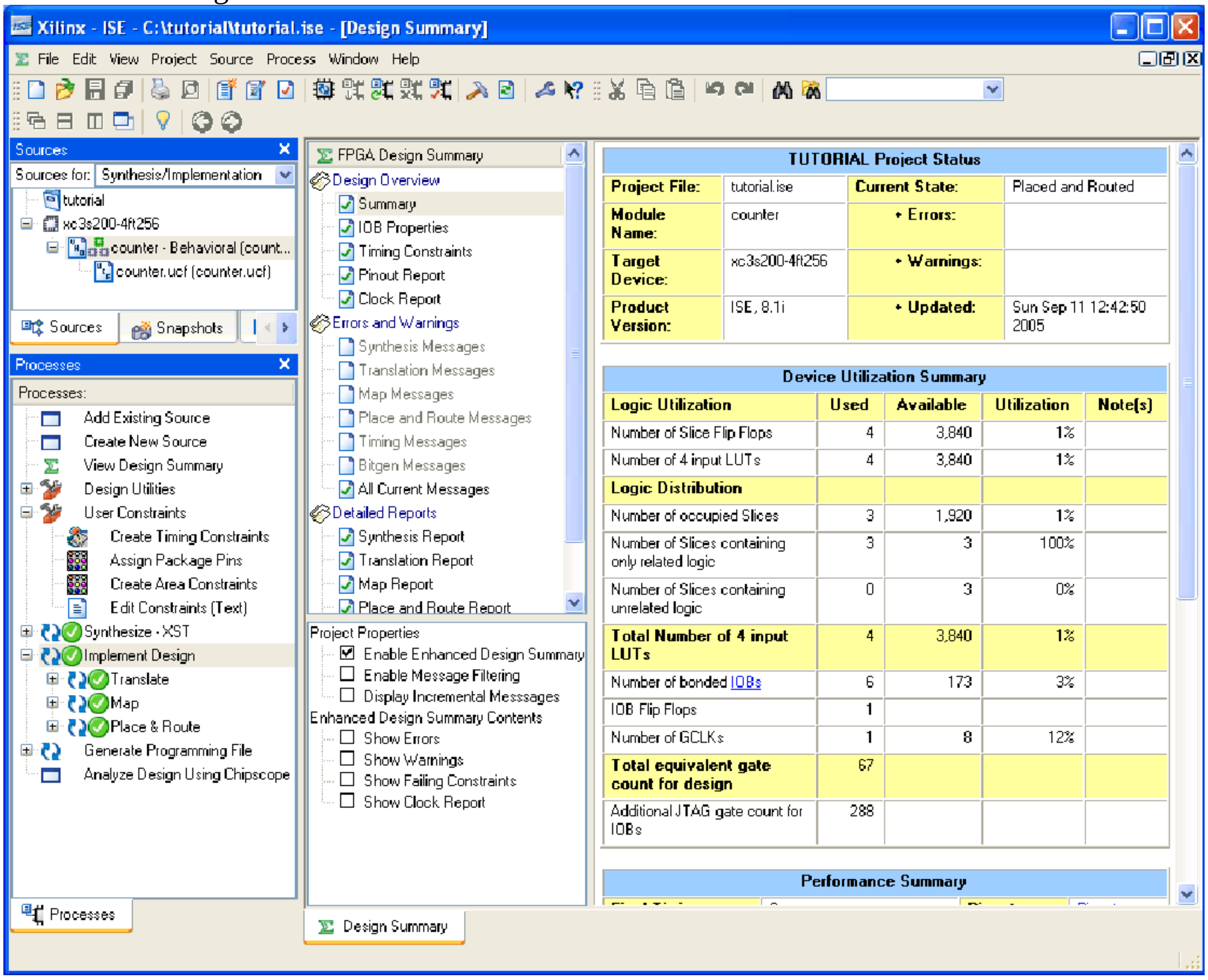
15. Close the Constraints Editor.

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.



Project Properties
- [✓] Enable Enhanced Design Summary
- [ ] Enable Message Filtering
- [ ] Display Incremental Messsages

Enhanced Design Summary Contents
- [ ] Show Errors
- [ ] Show Warnings
- [ ] Show Failing Constraints
- [ ] Show Clock Report

TUTORIAL Project Status

Project File:	tutorial.ise	**Current State:**	Placed and Routed
Module Name:	counter	**• Errors:**	
Target Device:	xc3s200-4ft256	**• Warnings:**	
Product Version:	ISE, 8.1i	**• Updated:**	Sun Sep 11 12:42:50 2005

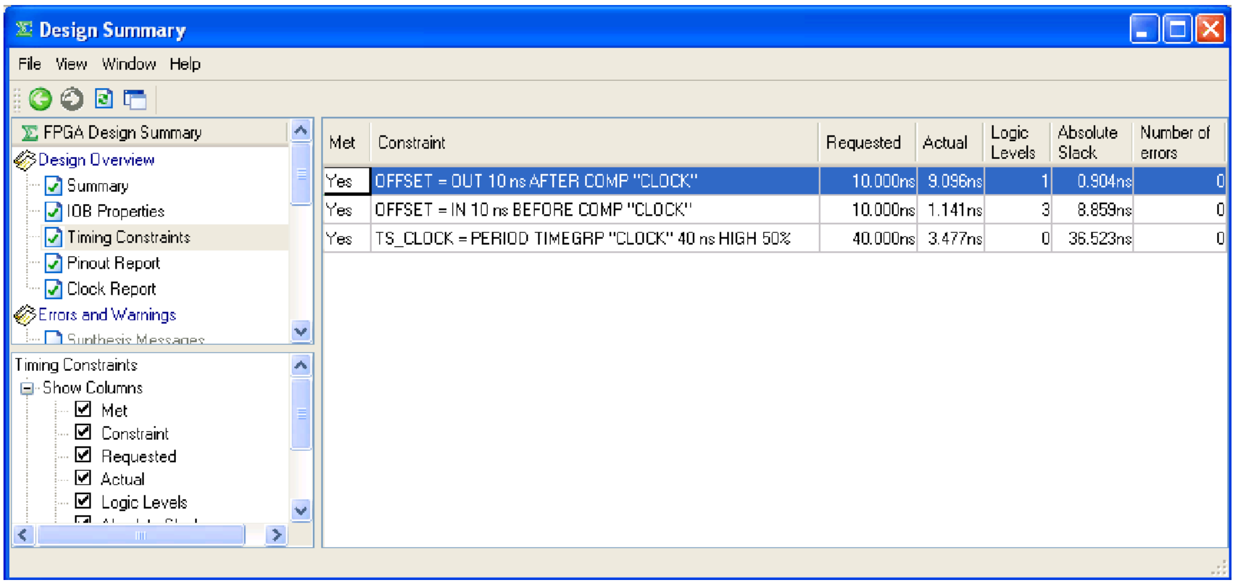
Device Utilization Summary

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				
Number of occupied Slices	3	1,920	1%	
Number of Slices containing only related logic	3	3	100%	
Number of Slices containing unrelated logic	0	3	0%	
Total Number of 4 input LUTs	4	3,840	1%	
Number of bonded IOBs	6	173	3%	
IOB Flip Flops	1			
Number of GCLKs	1	8	12%	
Total equivalent gate count for design	67			
Additional JTAG gate count for IOBs	288			

Performance Summary

Figure 18: Post Implementation Design Summary

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.



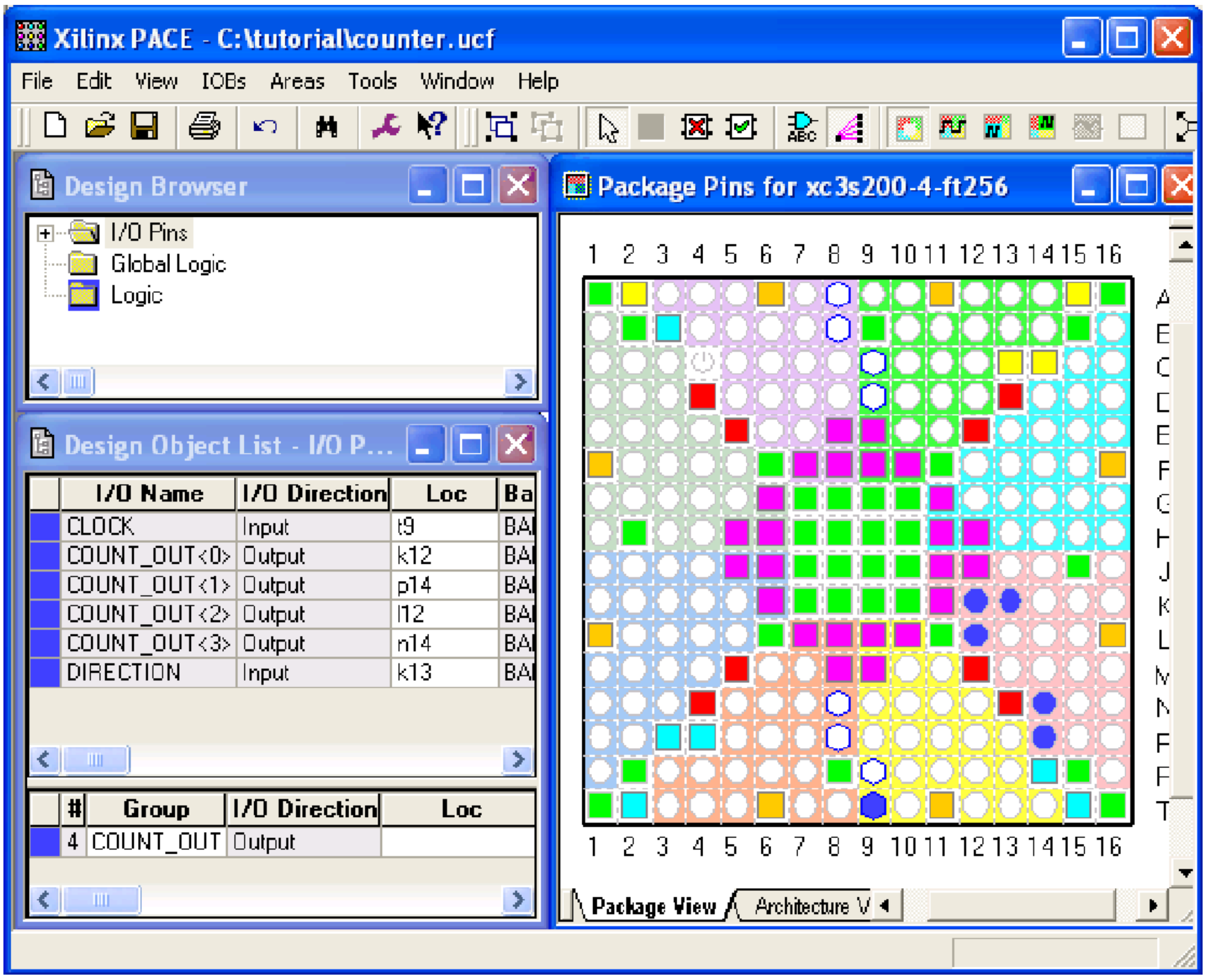
Met	Constraint	Requested	Actual	Logic Levels	Absolute Slack	Number of errors
Yes	OFFSET = OUT 10 ns AFTER COMP "CLOCK"	10.000ns	9.096ns	1	0.904ns	0
Yes	OFFSET = IN 10 ns BEFORE COMP "CLOCK"	10.000ns	1.141ns	3	8.859ns	0
Yes	TS_CLOCK = PERIOD TIMEGRP "CLOCK" 40 ns HIGH 50%	40.000ns	3.477ns	0	36.523ns	0

Figure 19: All Constraints Met Report

7. 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 Assign Package Pins 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:
♦ CLOCK input port connects to FPGA pin T9 (GCK0 signal on board)
♦ COUNT_OUT<0> output port connects to FPGA pin K12 (LD0 signal on board)
♦ COUNT_OUT<1> output port connects to FPGA pin P14 (LD1 signal on board)
♦ COUNT_OUT<2> output port connects to FPGA pin L12 (LD2 signal on board)
♦ COUNT_OUT<3> output port connects to FPGA pin N14 (LD3 signal on board)
♦ DIRECTION input port connects to FPGA pin K13 (SW7 signal on board)
Notice that the assigned pin locations are shown in blue:


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	

Figure 20: Package Pin Locations

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. Close PACE.

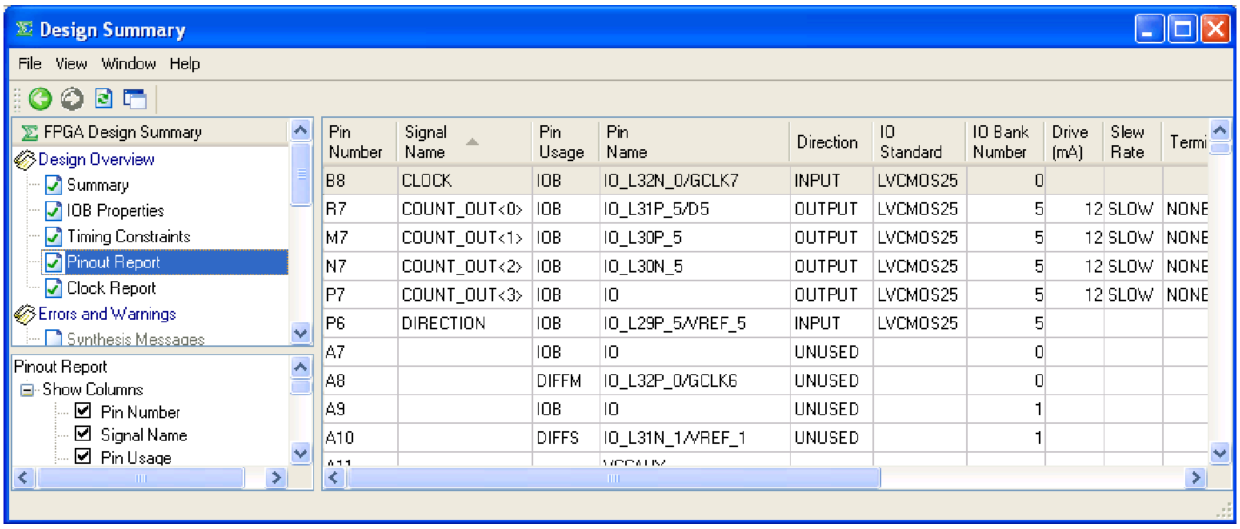
Notice that the Implement Design processes have an orange question mark next to them, indicating they are out-of-date with one or more of the design files. This is because the UCF file has been modified.

Re-implement Design and Verify Pin Locations

Re-implement the design and verify that the ports of the counter design are routed to the package pins specified in the previous section.

First, review the Pinout Report from the previous implementation by doing the following:

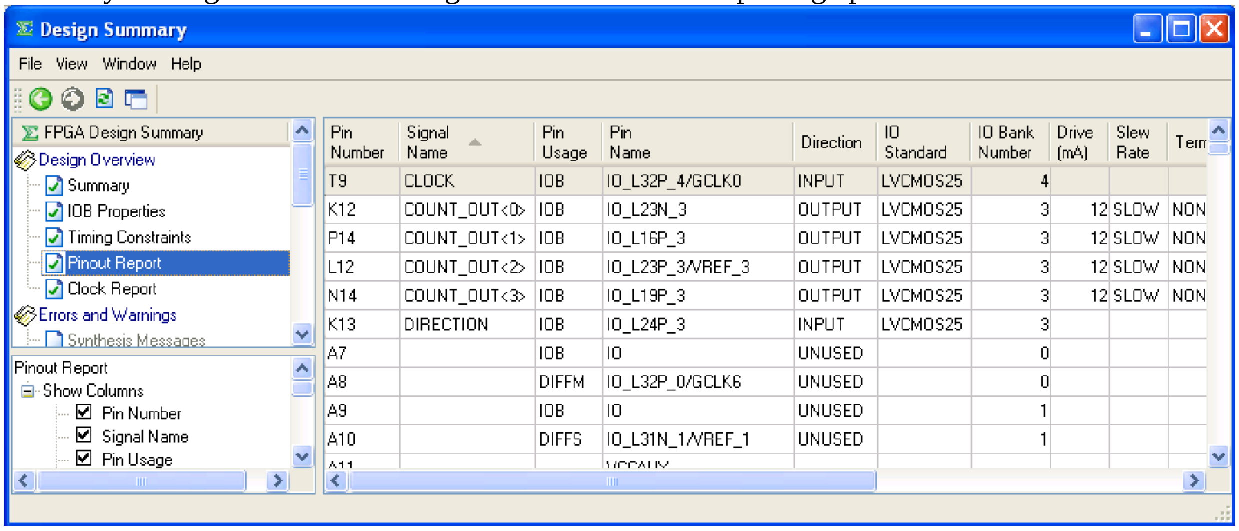
1. Open the Design Summary by double-clicking the View Design Summary process in the Processes window.
2. Select the Pinout Report and select the Signal Name column header to sort the signal names. Notice the Pin Numbers assigned to the design ports in the absence of location constraints.



Pin Number	Signal Name	Pin Usage	Pin Name	Direction	IO Standard	IO Bank Number	Drive (mA)	Slew Rate	Termi
B8	CLOCK	IOB	IO_L32N_0/GCLK7	INPUT	LVCMOS25	0			
R7	COUNT_OUT<0>	IOB	IO_L31P_5/D5	OUTPUT	LVCMOS25	5	12	SLOW	NONE
M7	COUNT_OUT<1>	IOB	IO_L30P_5	OUTPUT	LVCMOS25	5	12	SLOW	NONE
N7	COUNT_OUT<2>	IOB	IO_L30N_5	OUTPUT	LVCMOS25	5	12	SLOW	NONE
P7	COUNT_OUT<3>	IOB	IO	OUTPUT	LVCMOS25	5	12	SLOW	NONE
P6	DIRECTION	IOB	IO_L29P_5/VREF_5	INPUT	LVCMOS25	5			
A7		IOB	IO	UNUSED		0			
A8		DIFFM	IO_L32P_0/GCLK6	UNUSED		0			
A9		IOB	IO	UNUSED		1			
A10		DIFFS	IO_L31N_1/VREF_1	UNUSED		1			

Figure 21: Package Pin Locations Prior to Pin Location Constraints

3. Re-implement the design by double-clicking the Implement Design process.
4. Select the Pinout Report again and select the Signal Name column header to sort the signal names.
5. Verify that signals are now being routed to the correct package pins.



Pin Number	Signal Name	Pin Usage	Pin Name	Direction	IO Standard	IO Bank Number	Drive (mA)	Slew Rate	Term
T9	CLOCK	IOB	IO_L32P_4/GCLK0	INPUT	LVCMOS25	4			
K12	COUNT_OUT<0>	IOB	IO_L23N_3	OUTPUT	LVCMOS25	3	12	SLOW	NON
P14	COUNT_OUT<1>	IOB	IO_L16P_3	OUTPUT	LVCMOS25	3	12	SLOW	NON
L12	COUNT_OUT<2>	IOB	IO_L23P_3/VREF_3	OUTPUT	LVCMOS25	3	12	SLOW	NON
N14	COUNT_OUT<3>	IOB	IO_L19P_3	OUTPUT	LVCMOS25	3	12	SLOW	NON
K13	DIRECTION	IOB	IO_L24P_3	INPUT	LVCMOS25	3			
A7		IOB	IO	UNUSED		0			
A8		DIFFM	IO_L32P_0/GCLK6	UNUSED		0			
A9		IOB	IO	UNUSED		1			
A10		DIFFS	IO_L31N_1/VREF_1	UNUSED		1			

Figure 22: Package Pin Locations After Pin Location Constraints

6. Close the Design Summary.

Verify Design using Timing Simulation

Use the same self-checking test bench waveform you created in the previous section to verify the counter design after it has been completely implemented. Timing simulation Verifies that the design operates within the constraints specified after routing and logic delays are accounted for.

Run timing simulation as follows:

1. Select the Post-Route Simulation view from the drop-down list in the Sources window.
2. Select counter_tbw in the Sources window.
3. Run timing simulation by double-clicking the Simulate Post-Place & Route Model process found in the Xilinx ISE Simulator process group.

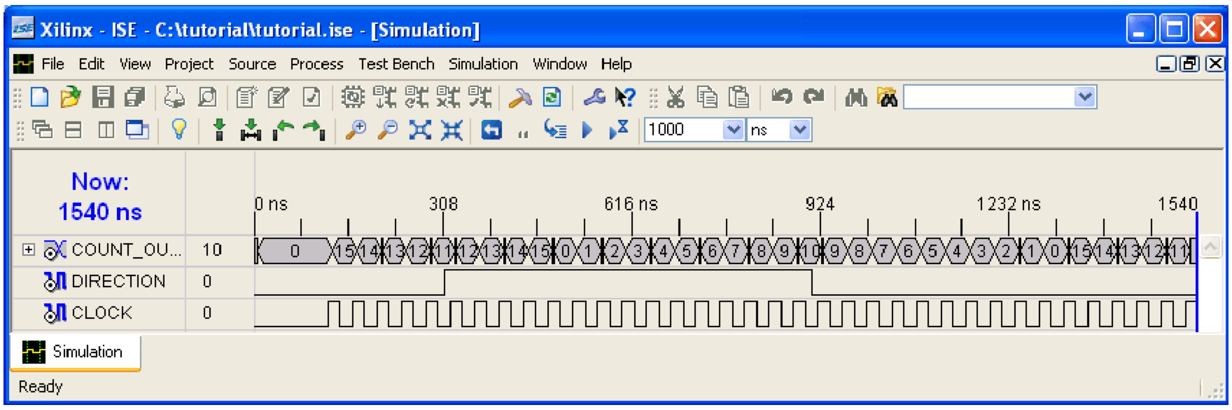

Xilinx - ISE - C:\tutorial\tutorial.ise - [Simulation]
File Edit View Project Source Process Test Bench Simulation Window Help
1000
ns
Now:
1540 ns
0 ns
308
616 ns
924
1232 ns
1540
COUNT_OU...
10
DIRECTION
0
CLOCK
0
Simulation
Ready

Figure 23: Post-Place & Route Simulation

4. Verify that the counter is counting up and down as directed by the stimulus on the DIRECTION port.
5. Verify that there are no ERRORs reported in the Simulator transcript window.
6. Zoom in to view the actual delay from the rising edge of CLOCK to a valid COUNT_OUT output change.
7. Close the simulation view.

You have completed timing simulation of your design using the ISE Simulator.

Download Design to the Spartan™-3 Demo Board

This is the last step in the design verification process. This section provides simple instructions for downloading the counter design to the Spartan-3 Starter Kit demo board.

1. Connect the 5V DC power cable to the power input on the demo board (J4).
2. Connect the download cable between the PC and demo board (J7).
3. Select Synthesis/Implementation from the drop-down list in the Sources window.
4. Select counter in the Sources window.
5. In the Processes window, click the "+" sign to expand the Generate Programming File processes.
6. Double-click the Configure Device (iMPACT) process.

iMPACT opens and the Configure Devices dialog box is displayed.

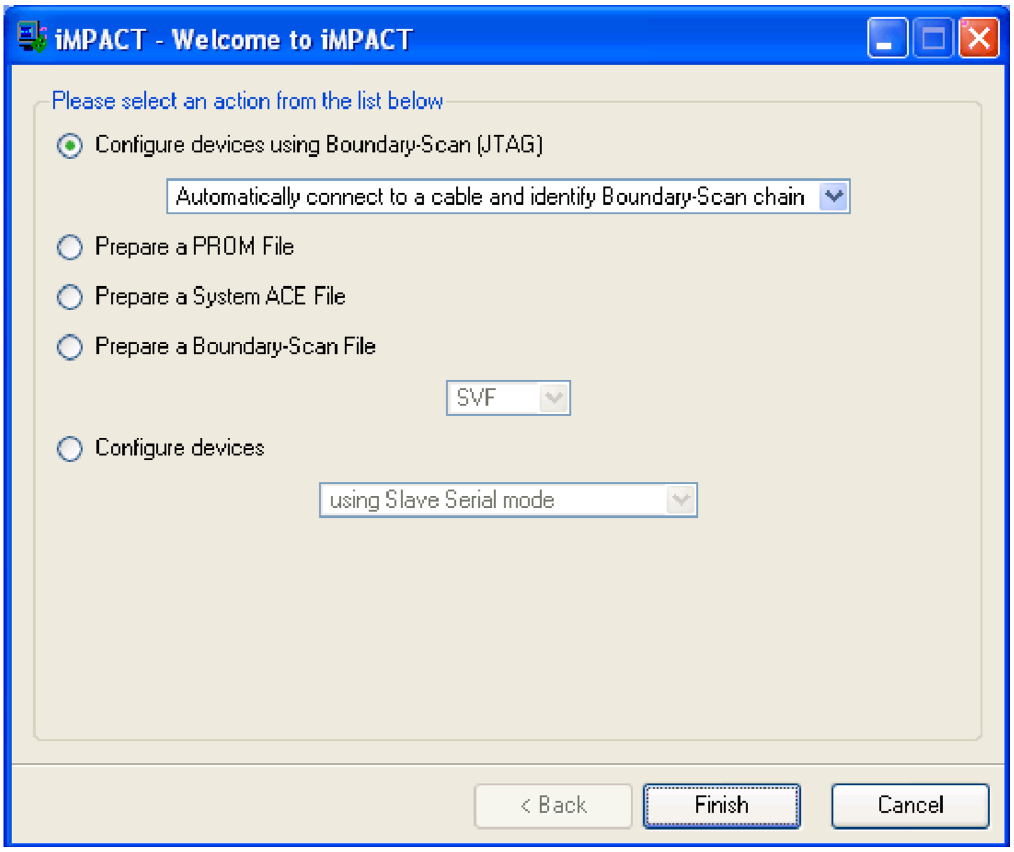
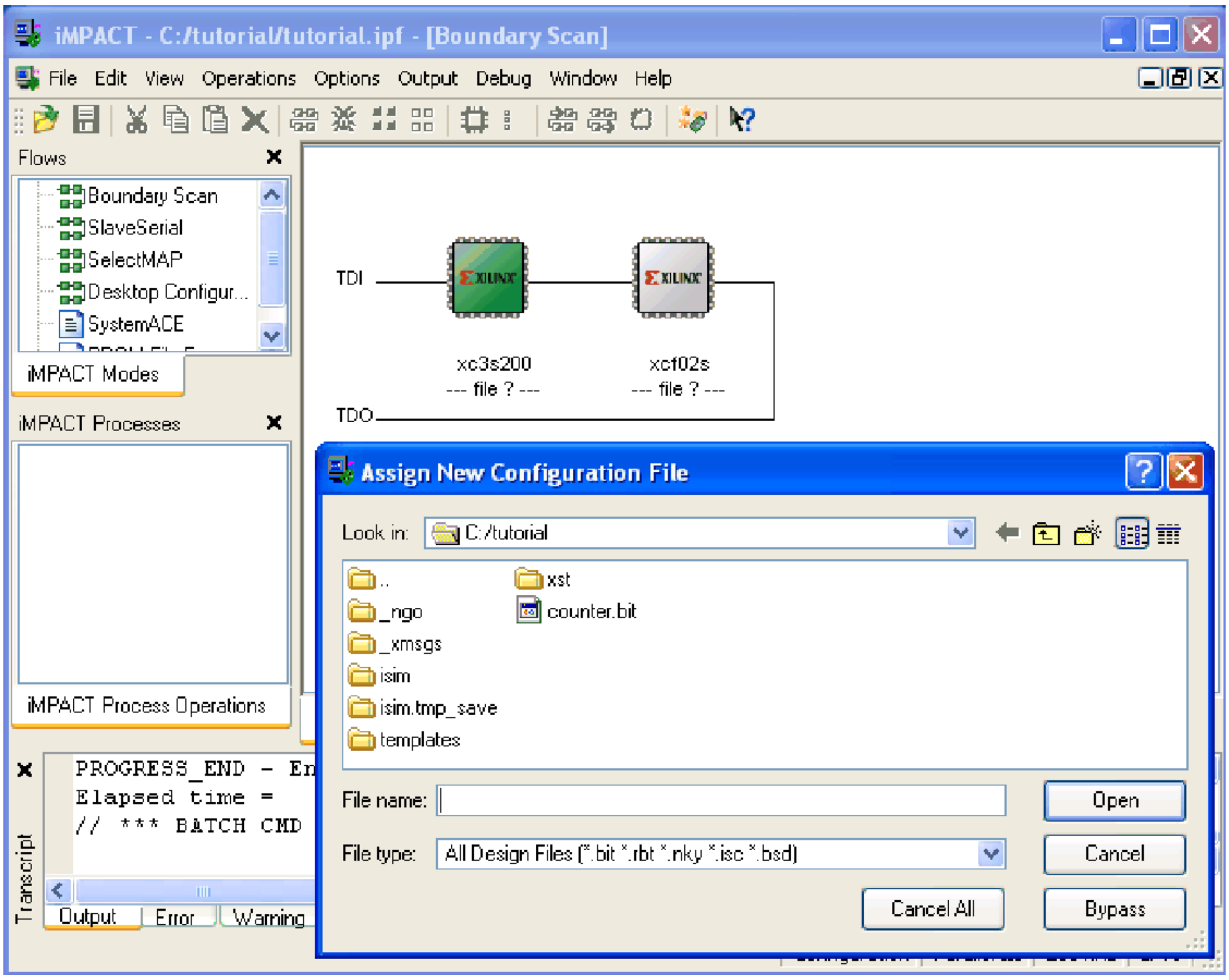

iMPACT - Welcome to iMPACT
Please select an action from the list below
Configure devices using Boundary-Scan (JTAG)
Automatically connect to a cable and identify Boundary-Scan chain
Prepare a PROM File
Prepare a System ACE File
Prepare a Boundary-Scan File
SVF
Configure devices
using Slave Serial mode
< Back
Finish
Cancel

Figure 24: iMPACT Welcome Dialog Box

7. In the Welcome dialog box, select Configure devices using Boundary-Scan (JTAG).
8. 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.



```
PROGRESS_END - En
Elapsed time =
// *** BATCH CMD
```

Figure 25: Assign New Configuration File

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.

Conclusion:

We have studied the procedure to write a VHDL code in XILINX software and various operations for its analysis which includes getting started, create a new project, create an HDL source design simulation etc.

TITLE: Implement 4-bit ALU for Add, Subtract, AND, NAND, XOR, XNOR, OR and ALU Pass	
NAME:	**SUBJECT**
CLASS: BE	**ROLL NO:**
SEMESTER/YEAR: - 7TH SEM	**EXAM NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED: -	**REMARKS:**

EXPERIMENT NO – 02

AIM: Implement 4 bit ALU for Add, Subtract, AND, NAND, XOR, XNOR, OR and ALU Pass.

OBJECTIVE:

1. To write VHDL code for up- down 4 bit ALU.
2. To modify above code for different operations.
3. To verify operation of above counters through its test benches.

PREREQUISITE:

Students should know the constructs of VHDL, ISE Simulator and Spartan 3 board.

TOOLS USED:

1. EDA front end tool- Xilinx
2. VLSI trainers with FPGA and CPLD.

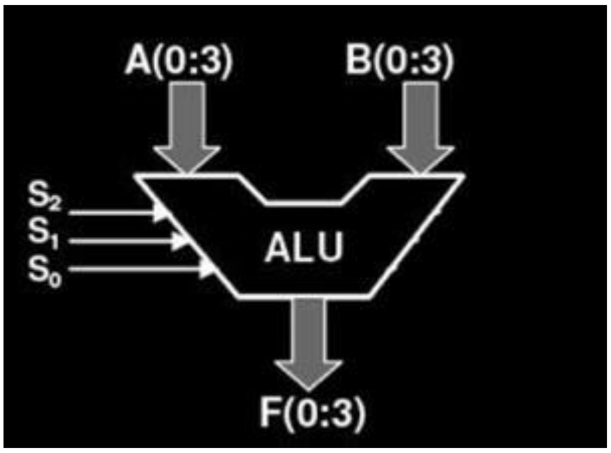
THEORY:

An **arithmetic logic unit** (ALU) is a digital electronic circuit that performs arithmetic and bitwise logical operations on integer binary numbers. This is in contrast to a floating-point unit (FPU), which operates on floating point numbers. ALU is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing units (GPUs). A single CPU, FPU or GPU may contain multiple ALUs.

The inputs to an ALU are the data to be operated on, called operands, and a code indicating the operation to be performed; the ALU's output is the result of the performed operation. In many designs, the ALU also exchanges additional information with a status register, which relates to the result of the current or previous operations

Opcode or Selection Line:

The opcode input is a parallel bus that conveys to the ALU an operation selection code, which is an enumerated value that specifies the desired arithmetic or logic operation to be performed by the ALU. The opcode size (its bus width) is related to the number of different operations the ALU can perform; for example, a four-bit opcode can specify up to sixteen different ALU operations. Generally, an ALU opcode is not the same as a machine language opcode, though in some cases it may be directly encoded as a bit field within a machine language opcode.



Arithmetic operations:

- Add: A and B are summed and the sum appears at Y and carry-out.
- Add with carry: A, B and carry-in are summed and the sum appears at Y and carry-out.
- Subtract: B is subtracted from A (or vice-versa) and the difference appears at Y and carry-out. For this function, carry-out is effectively a "borrow" indicator. This operation may

also be used to compare the magnitudes of A and B; in such cases the Y output may be ignored by the processor, which is only interested in the status bits (particularly zero and negative) that result from the operation.

- Subtract with borrow: B is subtracted from A (or vice-versa) with borrow (carry-in) and the difference appears at Y and carry-out (borrow out).
- Two's complement (negate): A (or B) is subtracted from zero and the difference appears at Y.
- Increment: A (or B) is increased by one and the resulting value appears at Y.
- Decrement: A (or B) is decreased by one and the resulting value appears at Y.
- Pass through: all bits of A (or B) appear unmodified at Y. This operation is typically used to determine the parity of the operand or whether it is zero or negative.

Bitwise logical operations:

- AND: the bitwise AND of A and B appears at Y.
- OR: the bitwise OR of A and B appear at Y.
- Exclusive-OR: the bitwise XOR of A and B appear at Y.
- One's complement: all bits of A (or B) are inverted and appear at Y

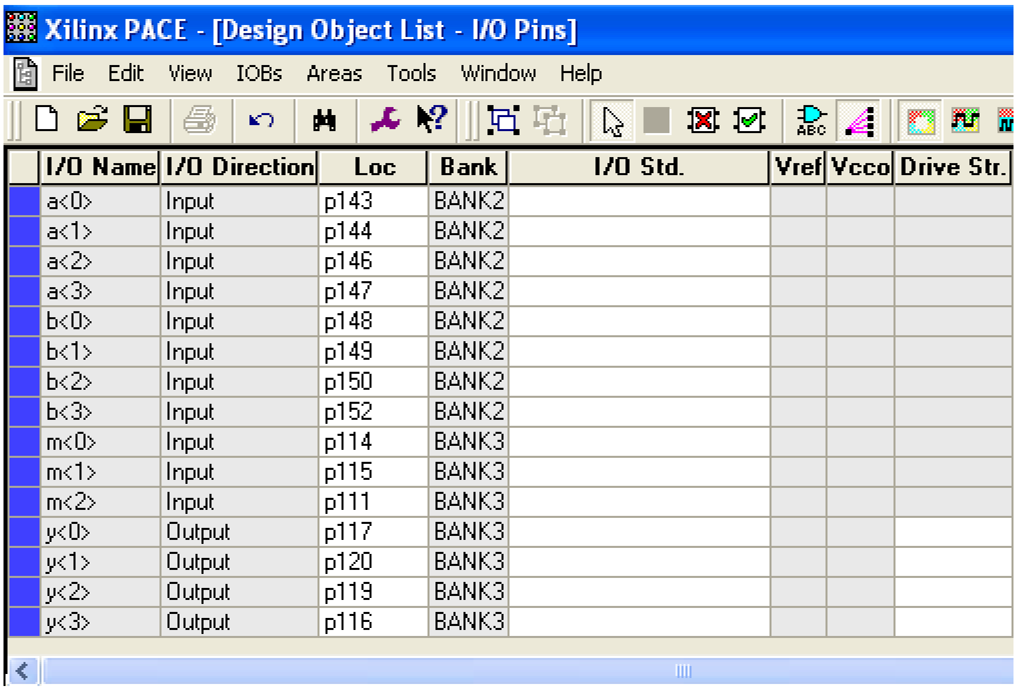
TRUTH TABLE:

S2	S1	S0	Function
0	0	0	A+B
0	0	1	A-B
0	1	0	A AND B
0	1	1	A NAND B
1	0	0	A XOR B
1	0	1	A XNOR B
1	1	0	A OR B
1	1	1	A

CONCLUSION:

We have implemented 4 bit ALU for Add, Subtract, AND, OR, NAND, XOR, XNOR, and ALU Pass using Xilinx software and learnt to write VHDL code for up- down 4 bit ALU, modify above code for different operations and to verify operation of above counters through its test benches.

ALU PIN PACKAGE:


Xilinx PACE - [Design Object List - I/O Pins]
File Edit View IOBs Areas Tools Window Help

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.
a<0>	Input	p143	BANK2				
a<1>	Input	p144	BANK2				
a<2>	Input	p146	BANK2				
a<3>	Input	p147	BANK2				
b<0>	Input	p148	BANK2				
b<1>	Input	p149	BANK2				
b<2>	Input	p150	BANK2				
b<3>	Input	p152	BANK2				
m<0>	Input	p114	BANK3				
m<1>	Input	p115	BANK3				
m<2>	Input	p111	BANK3				
y<0>	Output	p117	BANK3				
y<1>	Output	p120	BANK3				
y<2>	Output	p119	BANK3				
y<3>	Output	p116	BANK3				

Main Code:

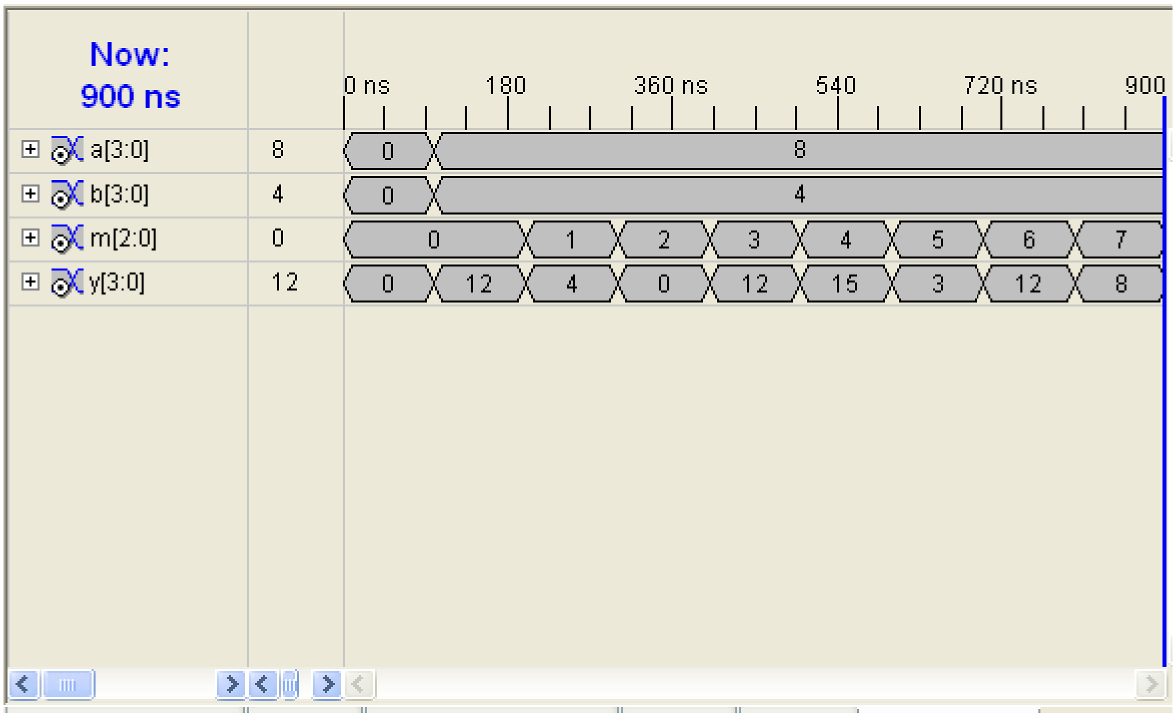
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity asd is
   Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
          b : in  STD_LOGIC_VECTOR (3 downto 0);
          m : in  STD_LOGIC_VECTOR (2 downto 0);
          y : out  STD_LOGIC_VECTOR (3 downto 0));
end asd;

architecture Behavioral of asd is
begin
process(a,b,m)
begin
case m is
when "000"=> y <= a+b;
when "001"=> y <= a-b;
when "010"=> y <= a and b;
when "011"=> y <= a or b;
when "100"=> y <= a nand b;
when "101"=> y <= a nor b;
when "110"=> y <= a xor b;
when "111"=> y <= a;
when others => y <= "0000";
end case;
end process;
end Behavioral;
```

TEST BENCH:

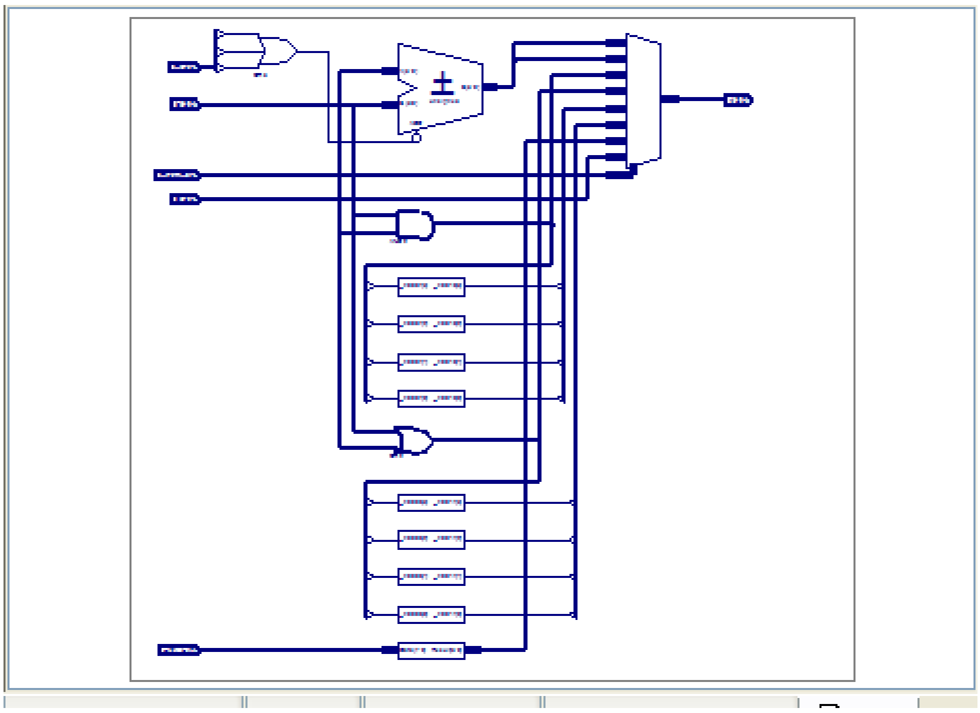
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY ertrt_vhd IS
END ertrt_vhd;

ARCHITECTURE behavior OF ertrt_vhd IS

          -- Component Declaration for the Unit Under Test (UUT)
          COMPONENT asd
          PORT(
                    a : IN std_logic_vector(3 downto 0);
                    b : IN std_logic_vector(3 downto 0);
                    m : IN std_logic_vector(2 downto 0);
                    y : OUT std_logic_vector(3 downto 0)
                    );
          END COMPONENT;

          --Inputs
          SIGNAL a : std_logic_vector(3 downto 0) := (others=>'0');
          SIGNAL b : std_logic_vector(3 downto 0) := (others=>'0');
          SIGNAL m :  std_logic_vector(2 downto 0) := (others=>'0');

          --Outputs
          SIGNAL y :  std_logic_vector(3 downto 0);
```

```
BEGIN
        -- Instantiate the Unit Under Test (UUT)
        uut: asd PORT MAP(
                a =>  a,
                b => b,
                m => m,
                y => y
        );
       tb : PROCESS
        BEGIN

                -- Wait 100 ns for global reset to finish
                wait for 100 ns;
                a<="1000";
                b<="0100";
                m<="000";
                wait for 100 ns;
    a<="1000";
                b<="0100";
                m<="001";
                wait for 100 ns;
                a<="1000";
                b<="0100";
                m<="010";
                wait for 100 ns;
                a<="1000";
                b<="0100";
                m<="011";
                wait for 100 ns;
                a<="1000";
                b<="0100";
                m<="100";
                wait for 100 ns;
                a<="1000";
                b<="0100";
                m<="101";
                wait for 100 ns;
                a<="1000";
                b<="0100";
                m<="110";
                wait for 100 ns;
                a<="1000";
                b<="0100";
                m<="111";
                wait for 100 ns;
                wait; -- will wait forever
        END PROCESS;
END;
```

TEST BENCH WAVE:



SIMULATION:



Final Results :

RTL Top Level Output File Name	: asd.ngr
Top Level Output File Name	: asd
Output Format	: NGC
Optimization Goal	: Speed
Keep Hierarchy	: NO

Design Statistics

```
# IOs                          : 15

Cell Usage :
# BELS                          : 33
#      LUT2                     : 1
#      LUT3                     : 16
#      LUT4                     : 4
#      MUXF5                      : 8
#      MUXF6                      : 4
# IO Buffers                     : 15
#      IBUF                     : 11
#      OBUF                      : 4
=========================================
```

TITLE: Implement Universal shift register with mode selection input for SIPO, SISO, PISO, PIPO modes	
NAME:	**SUBJECT: VLSI DESIGN AND TECHNOLOGY**
CLASS: BE	**ROLL NO:**
SEMESTER/YEAR: - 7TH SEM	**EXAM NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED: -	**REMARKS:**

EXPERIMENT NO – 03

AIM: Implement Universal shift register with mode selection input for SIPO, SISO, PISO, PIPO modes.

OBJECTIVE:

1. To write a VHDL code for Universal shift register and its test bench.
2. To verify the operation from the timing diagrams.
3. Experimental testing on hardware prototype for Universal shift register.

PREREQUISITE:
Students should know the constructs of VHDL, ISE Simulator and Spartan 3 board.

TOOLS USED:

1. EDA front end tool- Xilinx
2. VLSI trainers with FPGA and CPLD

THEORY:

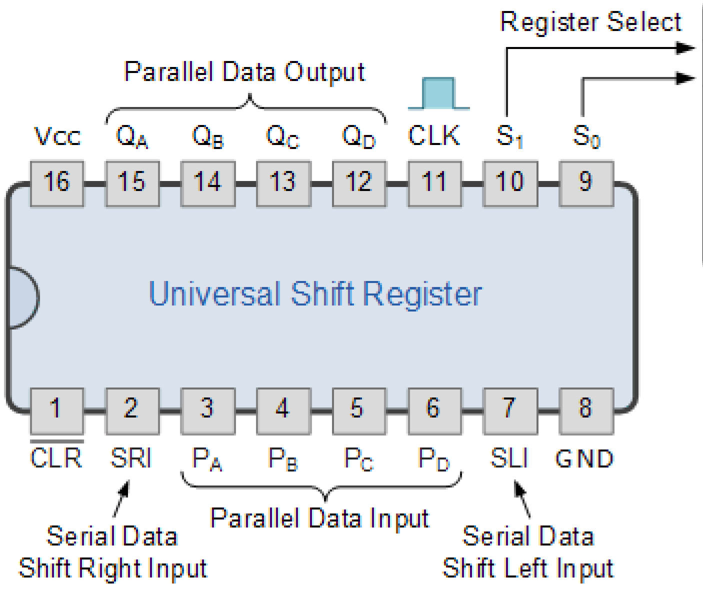
There are many high speed bi-directional “universal” type **Shift Registers** available such as the TTL 74LS194, 74LS195 or CMOS 4035.

They are available as 4-bit multi-function devices that can be used in either serial-to-serial, left shifting, right shifting, serial-to-parallel, parallel-to-serial, or as a parallel-to-parallel multifunction data register, hence the name “Universal”.

These universal shift registers can perform any combination of parallel and serial input to output operations but require additional inputs to specify desired function and to pre-load and reset the device. Universal shift registers are very useful digital devices. They can be configured to respond to operations that require some form of temporary memory storage or for the delay of information such as the SISO or PIPO configuration modes or transfer data from one point to another in either a serial or parallel format. Universal shift registers are frequently used in arithmetic operations to shift data to the left or right for multiplication or division.

A commonly used universal shift register is the TTL 74LS194 as shown below:

4-bit Universal Shift Register 74LS194



Mode Control		Register Type
S_1	S_0	
0	0	Hold
0	1	Shift to right
1	0	Shift to left
1	1	Parallel Mode

The register capable of shifting both right and left is called a bidirectional shift register.
The SN54/74LS194A is a High Speed 4-Bit Bidirectional Universal Shift Register. As a high speed multifunctional sequential building block, it is useful in a wide variety of applications. It may be used in serial-serial, shift left, shift right, serial-parallel, parallel-serial, and parallel-parallel data register transfers.

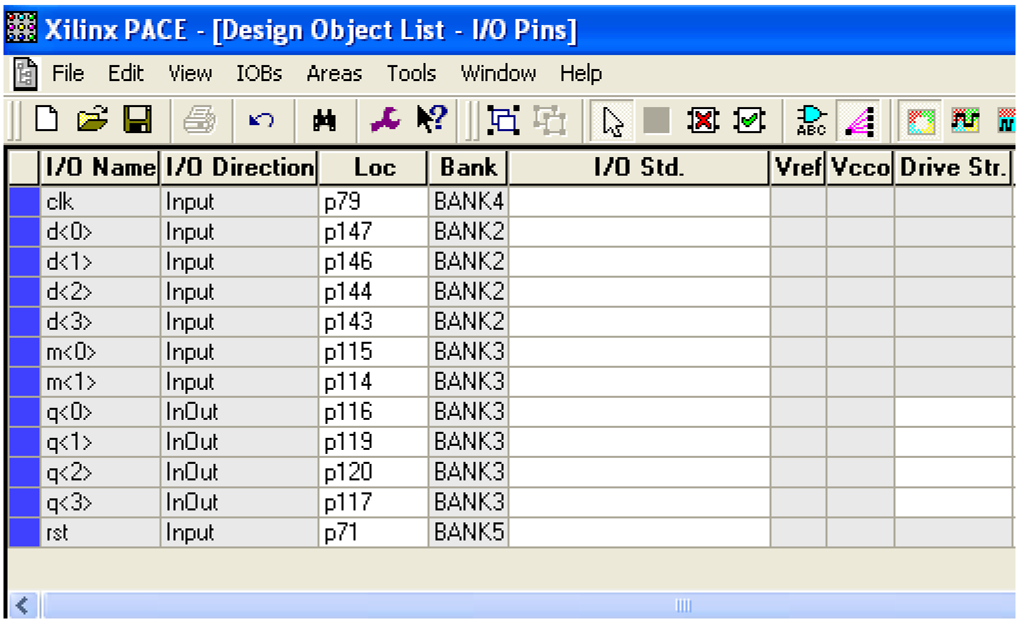
PIN NAMES:-

S0, S1	Mode Control Inputs
P0–P3	Parallel Data Inputs
DSR	Serial (Shift Right) Data Input
DSL	Serial (Shift Left) Data Input
CP	Clock (Active HIGH Going Edge) Input
MR	Master Reset (Active LOW) Input
Q0–Q3	Parallel Outputs

CONCLUSION:

We have implemented a Universal shift register with mode selection input for SIPO, SISO, PISO, PIPO modes using Xilinx software and understood the function of shift registers along with its different modes of operations.

PIN PACKAGE:


Xilinx PACE - [Design Object List - I/O Pins]
File Edit View IOBs Areas Tools Window Help

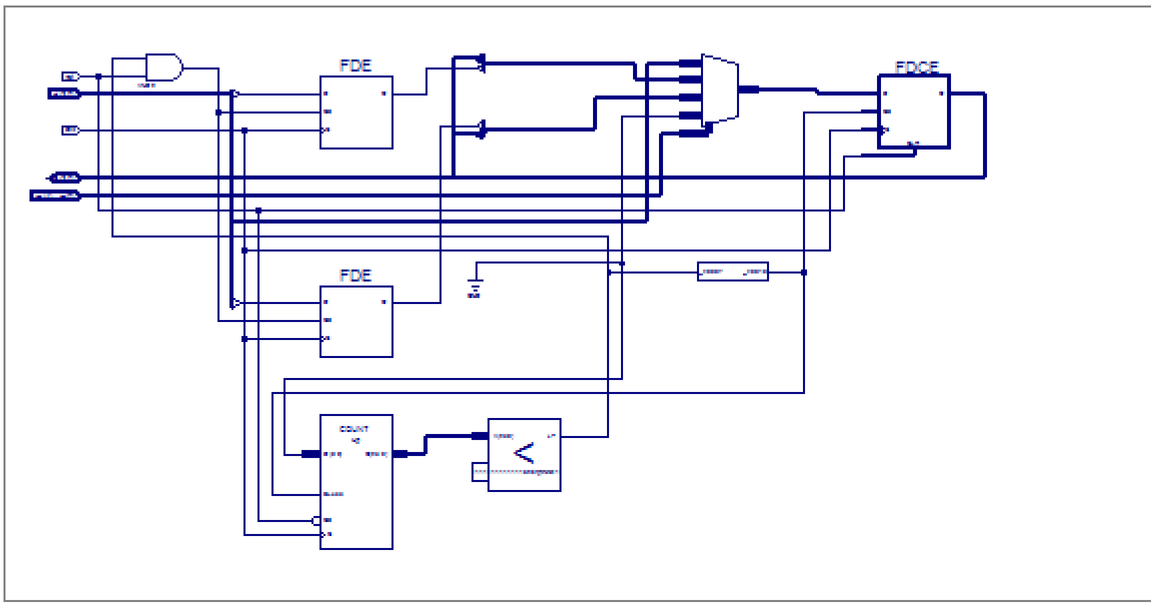
I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.
clk	Input	p79	BANK4				
d<0>	Input	p147	BANK2				
d<1>	Input	p146	BANK2				
d<2>	Input	p144	BANK2				
d<3>	Input	p143	BANK2				
m<0>	Input	p115	BANK3				
m<1>	Input	p114	BANK3				
q<0>	InOut	p116	BANK3				
q<1>	InOut	p119	BANK3				
q<2>	InOut	p120	BANK3				
q<3>	InOut	p117	BANK3				
rst	Input	p71	BANK5				

MAIN CODE:

```
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 instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity qwedweer is
   Port ( d : in STD_LOGIC_VECTOR (3 downto 0);
          clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          q : inout STD_LOGIC_VECTOR (3 downto 0);
          m : in  STD_LOGIC_VECTOR (1 downto 0));
end qwedweer;

architecture Behavioral of qwedweer is
signal lsbin,msbin:std_logic;
signal clk_divider: std_logic_vector(23 downto 0);
begin
process(clk,rst)
begin
if rst='1' then
q<="0000";
elsif clk'event and clk='1' then
if clk_divider<x"3fffff" then
clk_divider<= clk_divider +'1';
else
clk_divider<=x"000000";
msbin<=d(3);
lsbin<=d(0);
case m is
when "00"=> q <= d;
when "01"=> q <= q (2 downto 0)& lsbin;
when "10"=> q <= msbin & q(3 downto 1);
when others=> q <= "0000";
```

```
end case;
end if;
end if;
end process;
end Behavioral;
```

RTL SCHEMATIC:



FINAL RESULT:

```
RTL Top Level Output File Name     : qwedweer.ngr
Top Level Output File Name         : qwedweer
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 12

Cell Usage :
# BELS                             : 99
#      GND                         : 1
#      INV                         : 3
#      LUT2                        : 1
#      LUT2_D                      : 1
#      LUT2_L                      : 25
#      LUT3                        : 4
#      LUT4                        : 4
#      LUT4_L                      : 5
#      MULT_AND                    : 1
#      MUXCY                       : 30
#      VCC                         : 1
#      XORCY                       : 23
# FlipFlops/Latches                  30
#      FDCE                        : 4
#      FDE                         : 26
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 11
#      IBUF                        : 7
#      OBUF                        : 4
```

TITLE: Design and implement FIFO memory	
NAME:	**SUBJECT: VLSI DESIGN AND TECHNOLOGY**
CLASS: BE	**ROLL NO:**
SEMESTER/YEAR: - 7TH SEM	**EXAM NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED: -	**REMARKS:**

EXPERIMENT NO – 04

AIM: Design and implement FIFO memory.

OBJECTIVE:

1. To write a VHDL code for FIFO memory.
2. To verify the operation from the timing diagrams.
3. Experimental testing on hardware prototype for FIFO memory.

PREREQUISITE:

Students should know the constructs of VHDL, ISE Simulator and Spartan 3 board.

TOOLS USED:

1. EDA front end tool- Xilinx
2. VLSI trainers with FPGA and CPLD.

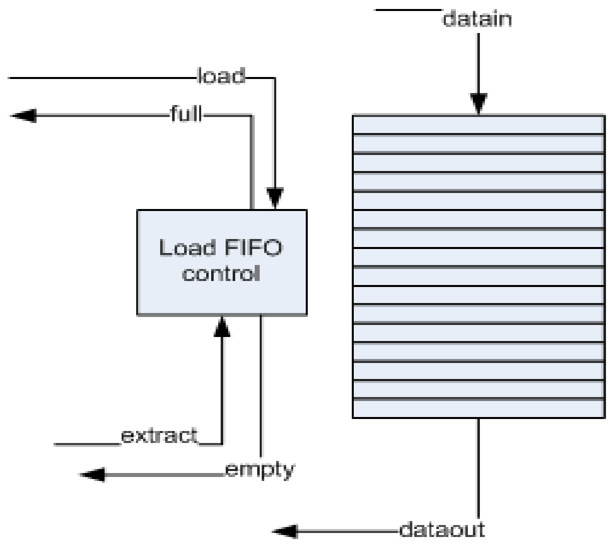
THEORY:

FIFO is an acronym for First In, First Out data organization method. FIFOs (First In, First Out) are essentially memory buffers used to temporarily store data until another process is ready to read it. As their name suggests the first byte written into a FIFO will be the first one to appear on the output. Typically, FIFOs are used when you have two processes that operate and a different rate. A common example is a high-speed communications channel that writes a burst of data into a FIFO and then a slower communications channel that read the data as need to send it at a slower rate. FIFOs are widely used in logic design for buffering, queuing and management of rate, priorities and flow control in data applications

The FIFO module below has two settings that can be configured to adjust the width and depth of the FIFO. The DATA_WIDTH variable adjusts the size of the DataIn and DataOutbuses so that you can write different sizes of bytes if needed and the FIFO_DEPTHvariable adjusts how big the internal memory of the FIFO is. A FIFO consists of a read pointer and a write pointer, pointing to entries in a storage array typically, made of flip-flops

In order to write data into the FIFO first push the data onto the DataIn bus and then strobe the WriteEn input high for one clock cycle. This will write whatever is on DataIn into the FIFOs internal memory. If writing in bulk the WriteEn signal can be left high while changing the data on the DataIn bus each clock cycle. When the Full flag goes high, this means that the FIFO's memory is full and will not accept any more writes until data is read using theReadEn input. If data is written while the Full flag is high it will be ignored.

BLOCK DIAGRAM:



For a standard FIFO when you write the first byte into the FIFO nothing happens on the DataOut bus until the ReadEn signal is pulsed high for at-least one clock cycle to. Once a byte has been written into the FIFO the Empty flag will go low. To read the next byte from the FIFO strobe the ReadEn signal high for one clock cycle and the next byte of data will be available to read on the next clock cycle. When the last byte of data is pushed onto theDataOut bus the Empty flag will go high.

CONCLUSION:

We have Designed and implement VHDL code for FIFO memory using Xilinx software and tested the implemented design using the hardware kits, also learnt the about the operations of FIFO memory like DataOut, DataIn, ReadEn and WriteEn.

MAIN CODE:

```
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 instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity pppp is
   Port ( clk : in STD_LOGIC;
          en : in  STD_LOGIC;
          din : in STD_LOGIC_VECTOR (7 downto 0);
          dout : out STD_LOGIC_VECTOR (7 downto 0);
          full : out  STD_LOGIC;
          empty : out  STD_LOGIC);
```

```
end pppp;

architecture Behavioral of pppp is

type mem is array (0 to 255)of std_logic_vector(7 downto 0);
signal memory:mem;
signal readptr,writep tr:std_logic_vector(7 downto 0);
SIGNAL CLK_DIVIDER :std_logic_vector(23 downto 0);

begin
process(clk)
begin
if clk' event and clk='1' then
if clk_divider < x"7fffff" then
clk_divider<= clk_divider +'1';
else
clk_divider<= x"000000";

if en='1' then
dout<=memory(conv_integer(readptr));
readptr<=readptr+'1';
else
memory(conv_integer(writeptr))<=din;
writeptr<=writeptr+'1';
end if;
if writeptr="11111111" then
full<='1';
writeptr<="00000000";
else
full<='0';
end if;
if writeptr="00000000" then
empty<='1';
else
empty<='0';
end if;
end if;
end if;
end process;
end Behavioral;
```

FINAL RESULT:

```
RTL Top Level Output File Name     : pppp.ngr
Top Level Output File Name         : pppp
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              20

Cell Usage :
```

```
# BELS                          : 118
#      GND                      : 1
#      INV                      : 5
#      LUT1                     : 11
#      LUT1_L                   : 12
#      LUT2                     : 8
#      LUT2_D                   : 1
#      LUT2_L                   : 3
#      LUT3                     : 2
#      LUT3_L                   : 3
#      LUT4                     : 9
#      LUT4_L                   : 9
#      MUXCY                    : 30
#      VCC                      : 1
#      XORCY                    : 23
# FlipFlops/Latches             : 42
#      FDE                      : 10
#      FDR                      : 24
#      FDRE                     : 8
# RAMS                          : 1
#      RAMB16_S9_S9             : 1
# Clock Buffers                 : 1
#      BUFGP                    : 1
# IO Buffers                    : 19
#      IBUF                     : 9
#      OBUF                     : 10
```

Main code:

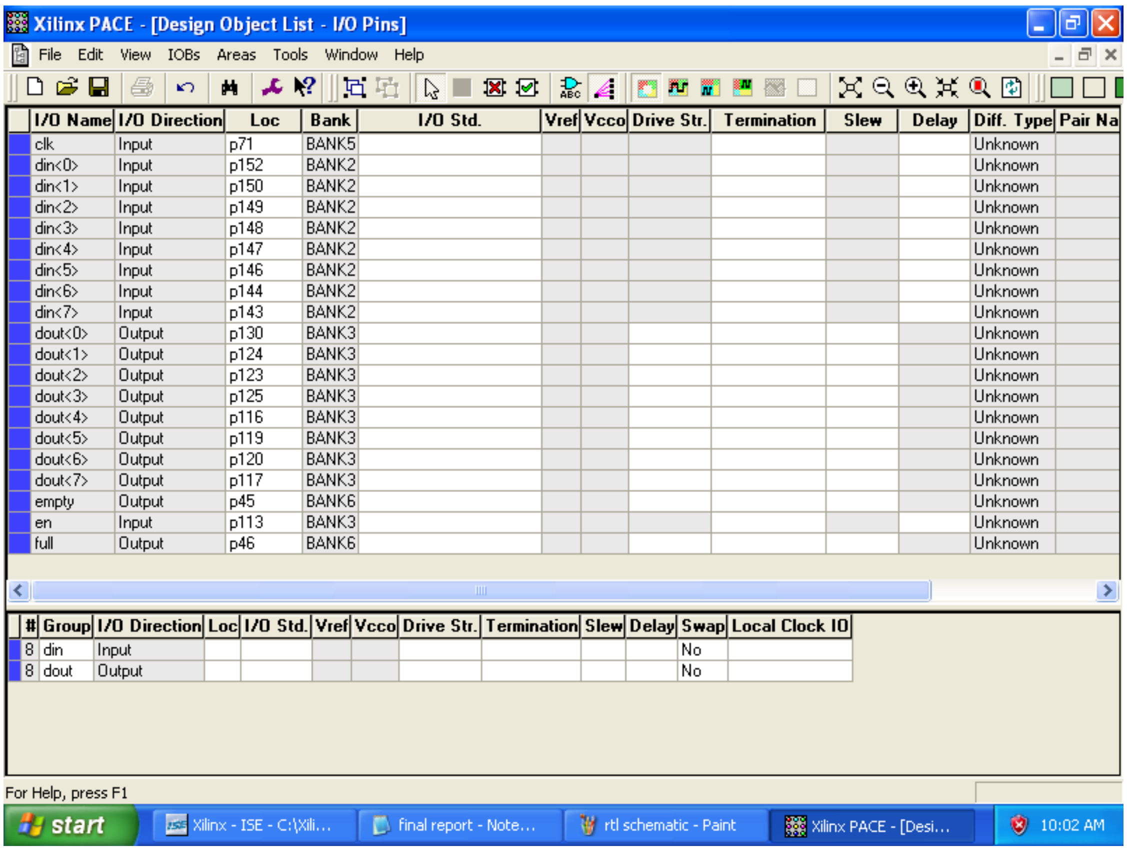
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity fiffo is
   Port ( din : in STD_LOGIC_VECTOR (7 downto 0);
          clk : in  STD_LOGIC;
          en : in STD_LOGIC;
          full : out  STD_LOGIC;
          empty : out  STD_LOGIC;
          dout : out STD_LOGIC_VECTOR (7 downto 0));
end fiffo;

architecture Behavioral of fiffo is
type mem is array(0 to 255)of std_logic_vector(7 downto 0);
signal memory:mem;
signal readptr,writeptr: std_logic_vector(7 downto 0);
signal clk_divider :std_logic_vector(23 downto 0);
```

```
begin
process(clk)
begin
if clk' event and clk='1' then
if clk_divider< x"0fffff" then
 clk_divider<= clk_divider+'1';
else
clk_divider <=x"000000";
if en='1' then
memory (conv_integer(writeptr))<= din;
writeptr <= writeptr+'1';
else
dout<= memory(conv_integer(readptr));
readptr<= readptr+'1';
end if;
if readptr="11111111" then
readptr <= "00000000";
end if;
if writeptr="11111111" then
writeptr <= "00000000";
full <='1';
else
full<='0';
end if;
if writeptr= "00000000" then
empty<='1';
else
empty<='0';
end if;
end if;
end if;
end process;

end Behavioral;
```

FINAL RESULT:

```
RTL Top Level Output File Name     : fiffo.ngr
Top Level Output File Name         : fiffo
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              20

Cell Usage :
# BELS                             : 118
#      GND                         : 1
#      INV                         : 3
#      LUT1                        : 11
#      LUT1_L                      : 12
#      LUT2                        : 8
#      LUT2_D                    : 2
#      LUT2_L                    : 3
#      LUT3                      : 2
#      LUT3_L                    : 2
#      LUT4                      : 11
#      LUT4_L                    : 10
#      MUXCY                     : 29
#      VCC                       : 1
#      XORCY                     : 23
# FlipFlops/Latches              : 42
#      FDE                       : 2
#      FDR                       : 24
#      FDRE                      : 16
# RAMS                           : 1
#      RAMB16_S9_S9              : 1
# Clock Buffers                  : 1
#      BUFGP                     : 1
# IO Buffers                     : 19
#      IBUF                      : 9
#      OBUF                      : 10
```

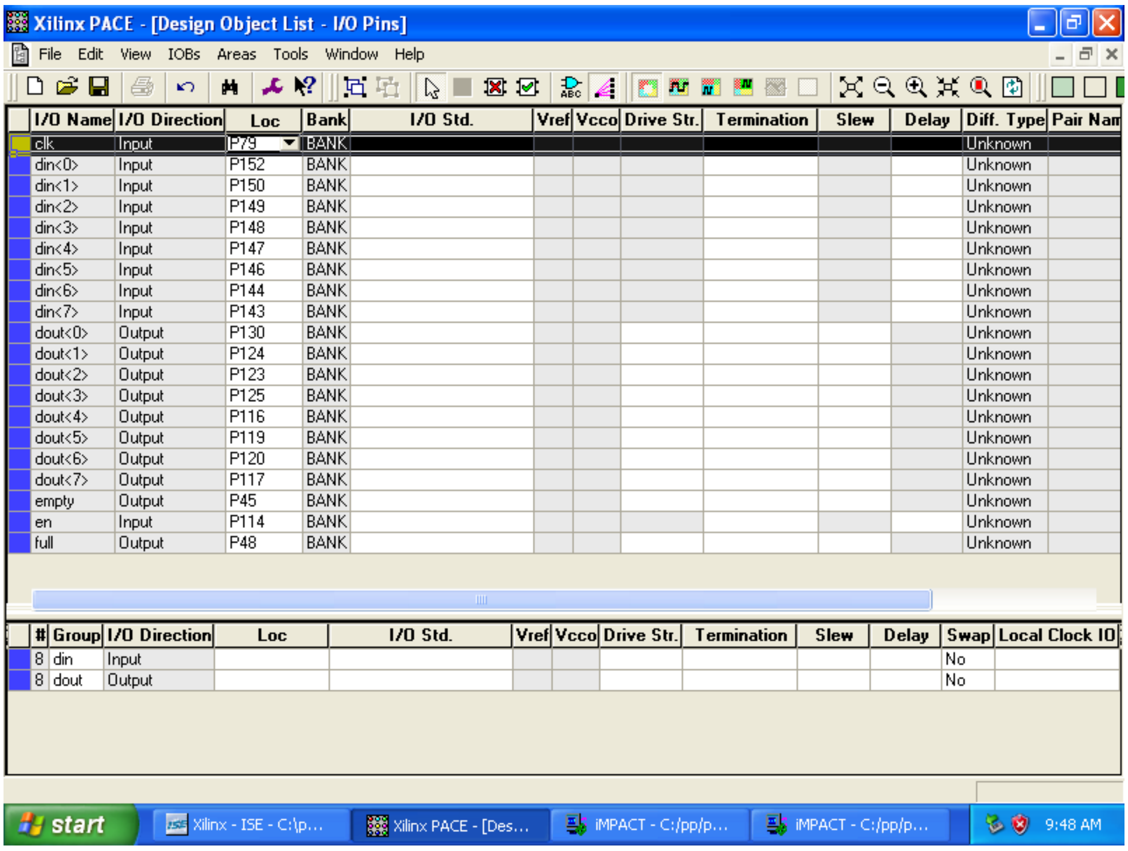
PIN PACKAGE:


Xilinx PACE - [Design Object List - I/O Pins]
File Edit View IOBs Areas Tools Window Help
start
Xilinx - ISE - C:\Xili...
final report - Note...
rtl schematic - Paint
Xilinx PACE - [Desi...
10:02 AM

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Na
clk	Input	p71	BANK5								Unknown	
din<0>	Input	p152	BANK2								Unknown	
din<1>	Input	p150	BANK2								Unknown	
din<2>	Input	p149	BANK2								Unknown	
din<3>	Input	p148	BANK2								Unknown	
din<4>	Input	p147	BANK2								Unknown	
din<5>	Input	p146	BANK2								Unknown	
din<6>	Input	p144	BANK2								Unknown	
din<7>	Input	p143	BANK2								Unknown	
dout<0>	Output	p130	BANK3								Unknown	
dout<1>	Output	p124	BANK3								Unknown	
dout<2>	Output	p123	BANK3								Unknown	
dout<3>	Output	p125	BANK3								Unknown	
dout<4>	Output	p116	BANK3								Unknown	
dout<5>	Output	p119	BANK3								Unknown	
dout<6>	Output	p120	BANK3								Unknown	
dout<7>	Output	p117	BANK3								Unknown	
empty	Output	p45	BANK6								Unknown	
en	Input	p113	BANK3								Unknown	
full	Output	p46	BANK6								Unknown	

#	Group	I/O Direction	Loc	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Swap	Local Clock IO
8	din	Input									No	
8	dout	Output									No	

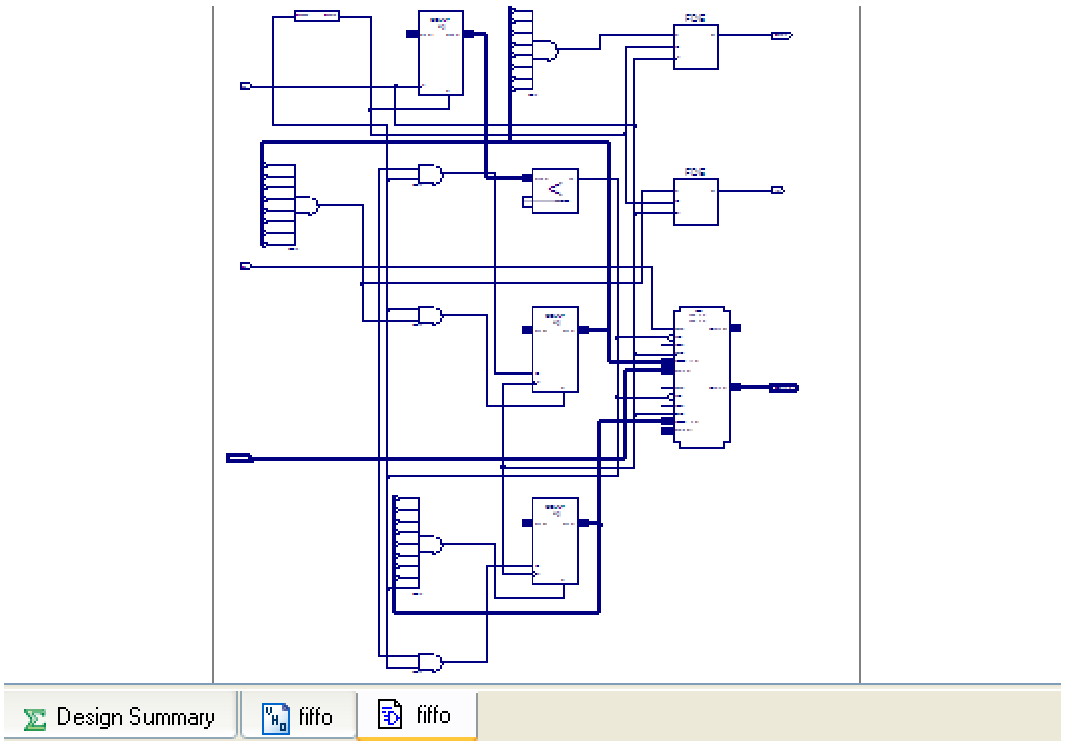
For Help, press F1



I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Diff. Type	Pair Nam
clk	Input	P79	BANK								Unknown	
din<0>	Input	P152	BANK								Unknown	
din<1>	Input	P150	BANK								Unknown	
din<2>	Input	P149	BANK								Unknown	
din<3>	Input	P148	BANK								Unknown	
din<4>	Input	P147	BANK								Unknown	
din<5>	Input	P146	BANK								Unknown	
din<6>	Input	P144	BANK								Unknown	
din<7>	Input	P143	BANK								Unknown	
dout<0>	Output	P130	BANK								Unknown	
dout<1>	Output	P124	BANK								Unknown	
dout<2>	Output	P123	BANK								Unknown	
dout<3>	Output	P125	BANK								Unknown	
dout<4>	Output	P116	BANK								Unknown	
dout<5>	Output	P119	BANK								Unknown	
dout<6>	Output	P120	BANK								Unknown	
dout<7>	Output	P117	BANK								Unknown	
empty	Output	P45	BANK								Unknown	
en	Input	P114	BANK								Unknown	
full	Output	P48	BANK								Unknown	

#	Group	I/O Direction	Loc	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Delay	Swap	Local Clock IO
8	din	Input									No	
8	dout	Output									No	

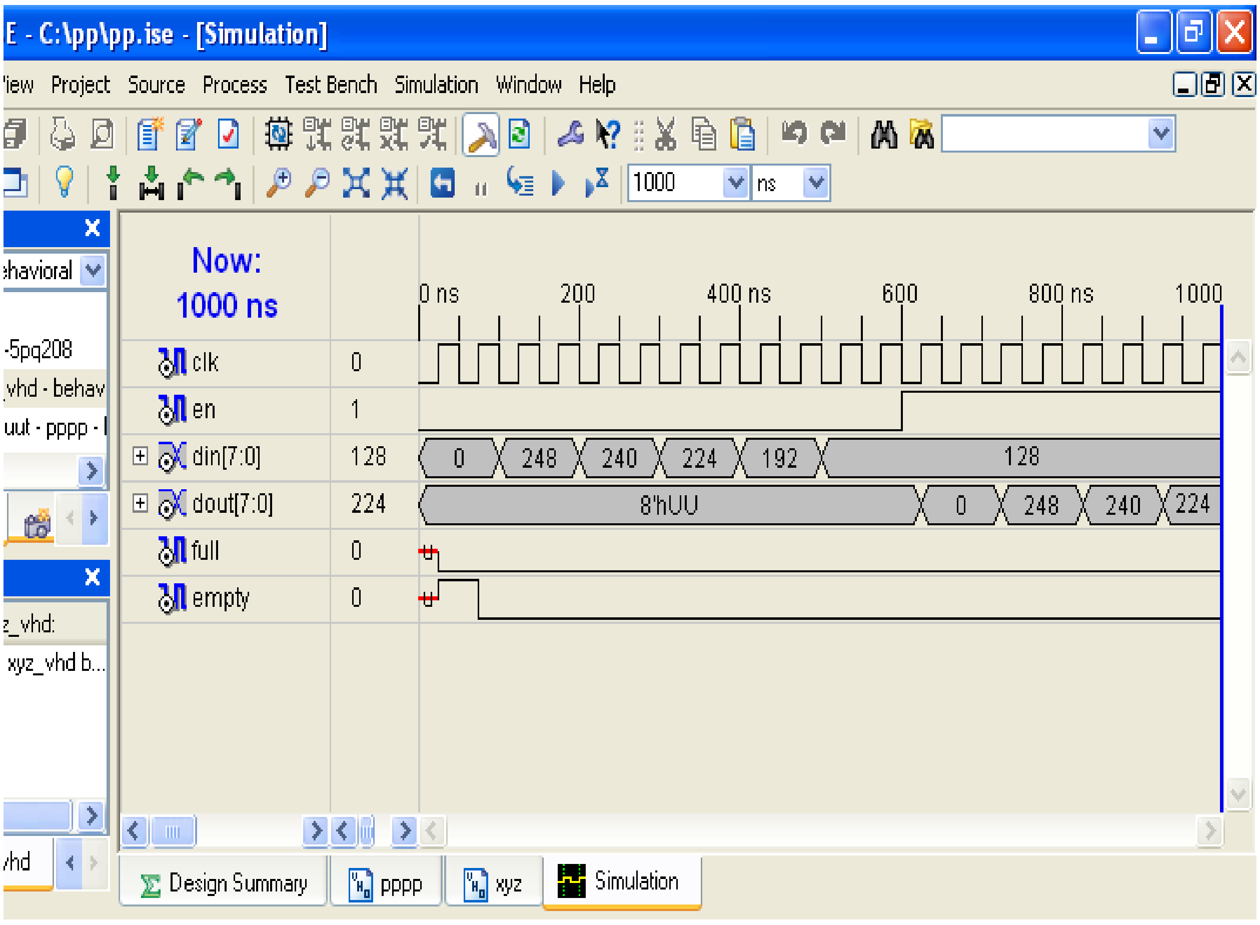
RTL SCHEMATIC:



TEST BENCH:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY xyz_vhd IS
END xyz_vhd;

ARCHITECTURE behavior OF xyz_vhd IS

        -- Component Declaration for the Unit Under Test (UUT)
        COMPONENT pppp
        PORT(
                clk : IN std_logic;
                en : IN std_logic;
                din : IN std_logic_vector(7 downto 0);
                dout : OUT std_logic_vector(7 downto 0);
                full : OUT std_logic;
                empty : OUT std_logic
                );
        END COMPONENT;

        --Inputs
        SIGNAL clk : std_logic := '0';
        SIGNAL en :  std_logic := '0';
        SIGNAL din :  std_logic_vector(7 downto 0) := (others=>'0');

        --Outputs
        SIGNAL dout : std_logic_vector(7 downto 0);
        SIGNAL full :  std_logic;
        SIGNAL empty : std_logic;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
        uut: pppp PORT MAP(
                clk => clk,
                en => en,
                din => din,
                dout => dout,
                full => full,
                empty => empty
        );

        tb :
        process
        begin
        clk<='0';
        wait for 25 ns;

        clk<='1';
        wait for 25 ns;

        end process;


        process
        begin

                -- Wait 100 ns for global reset to finish
                wait for 100 ns;
                en<='0';
                din<="11111000";

                wait for 100 ns;

                din<="11110000";

                wait for 100 ns;
                din<="11100000";

                wait for 100 ns;

                din<="11000000";

                wait for 100 ns;

                din<="10000000";

                wait for 100 ns;
                en<='1';

                wait; -- will wait forever
        END process;

END;
```

TEST BENCH WAVE:


E - C:\pp\pp.ise - [Simulation]
iew Project Source Process Test Bench Simulation Window Help
1000 ns
Now: 1000 ns
0 ns 200 400 ns 600 800 ns 1000
clk 0
en 1
din[7:0] 128
0 248 240 224 192 128
dout[7:0] 224
8'hUU 0 248 240 224
full 0
empty 0
Design Summary pppp xyz Simulation

TITLE: To write a VHDL code for Keypad interface, Simulate, Synthesis and Implement on SPARTAN-3 board	
NAME:	**SUBJECT: VLSI DESIGN AND TECHNOLOGY**
CLASS: BE	**ROLL NO:**
SEMESTER/YEAR: - 7TH SEM	**EXAM NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED: -	**REMARKS:**

EXPERIMENT NO – 05

AIM: To write a VHDL code for Keypad interface, Simulate, Synthesis and Implement on SPARTAN-3 board.

OBJECTIVE:

1. To write VHDL code for Keypad Interface
2. To verify operation of above through its test benches.

PREREQUISITE:

Students should know the constructs of VHDL, ISE Simulator and Spartan 3 board.

TOOLS USED:

1. EDA front end tool- Xilinx
2. VLSI trainers with FPGA and CPLD.

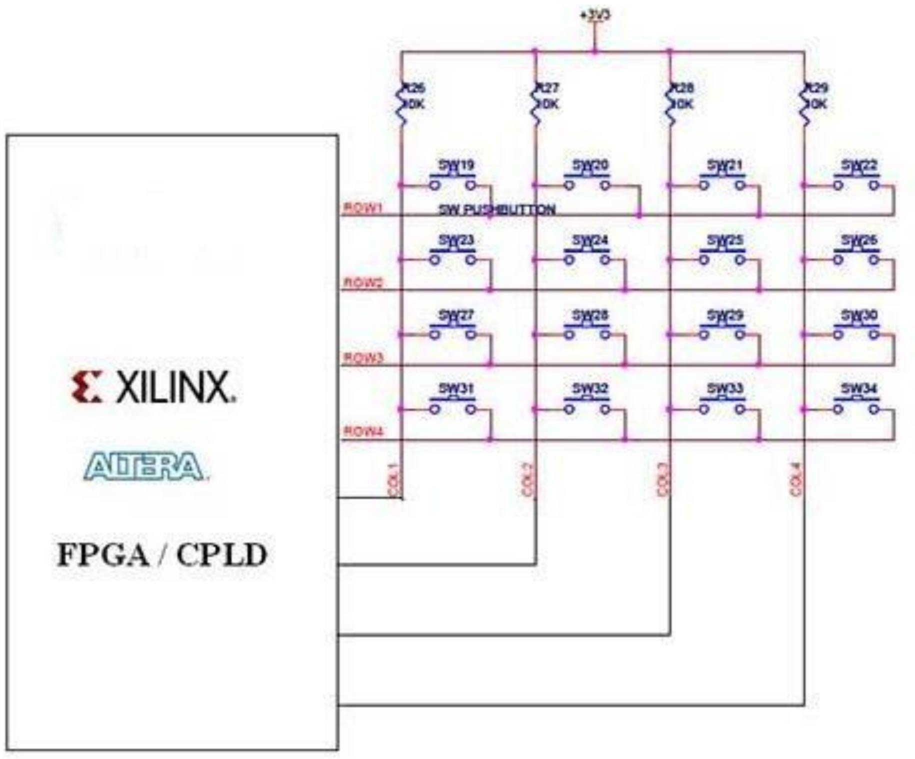
THEORY:

Matrix Keypad

Matrix keypad consists of a set of buttons similar to an alphanumeric keyboard provided with keys usually marked with letters or numbers and various extra keys. Embedded systems which require user interaction must be interfaced with devices that accept user input such as a keypad.

Interfacing Keypad with Spartan-3 Primer FPGA.

The Spartan-3 Primer board has 4x4 Matrix Keypad, indicated as in Figure. Keypads arranged by matrix format, each row and column section pulled by high, all row lines and column lines connected directly by the I/O pins.



CONCLUSION:

We have performed the VHDL code for Keypad interface, Simulate, Synthesis and Implement it on SPARTAN-3 board ,studied about matrix keypad and its interfacing keypad with spartan-3 primer FPGA.

MAIN CODE:

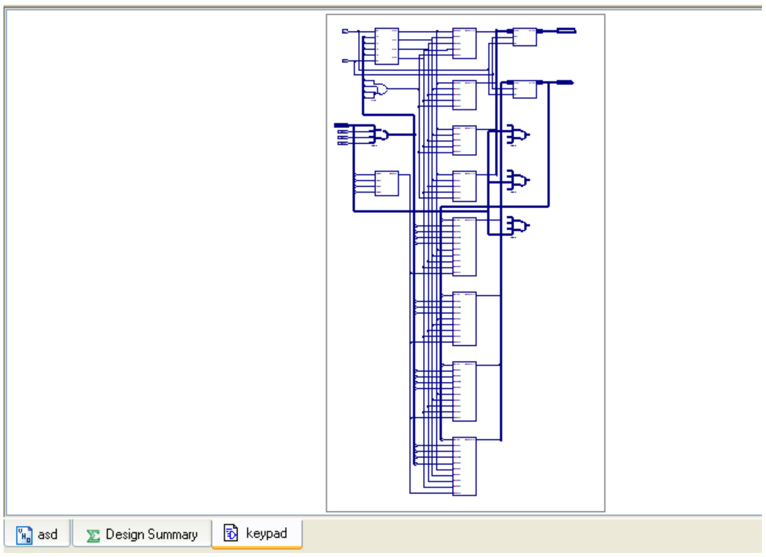
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity keypad is
   Port ( rst : in STD_LOGIC;
          clk : in  STD_LOGIC;
          row : in STD_LOGIC_VECTOR (3 downto 0);
          column : out STD_LOGIC_VECTOR (3 downto 0);
          data : out  STD_LOGIC_VECTOR (3 downto 0));
end keypad;

architecture Behavioral of keypad is
type state_type is(co1,co2,co3,co4);
signal coltest: state_type:= co1;
begin
process(clk,rst)
begin
if  rst='1'then
coltest <= co1;
column <= "1110";
data <="0001";
elsif clk' event and clk='1' then
case coltest is
when co1=> column <="1110";
case row is
when"1110"=> data <= "0001";
when"1101"=> data <= "0100";
when"1011"=> data <= "0111";
when"0111"=> data <= "1110";
when others => coltest <= co2;
column <="1101";
```

```
end case;

when co2=> column <="1101";
case row is
when"1110"=> data <= "0010";
when"1101"=> data <= "0101";
when"1011"=> data <= "1000";
when"0111"=> data <= "0000";
when others => coltest <= co3;
column <="1011";
end case;

when co3=> column <="1011";
case row is
when"1110"=> data <= "0011";
when"1101"=> data <= "0110";
when"1011"=> data <= "1001";
when"0111"=> data <= "1111";
when others => coltest <= co4;
column <="0111";
end case;

when co4=> column <="0111";
case row is
when"1110"=> data <= "1010";
when"1101"=> data <= "1011";
when"1011"=> data <= "1100";
when"0111"=> data <= "1101";
when others => coltest <= co1;
column <="1110";
end case;
end case;
end if ;
end      process;
end Behavioral;
```

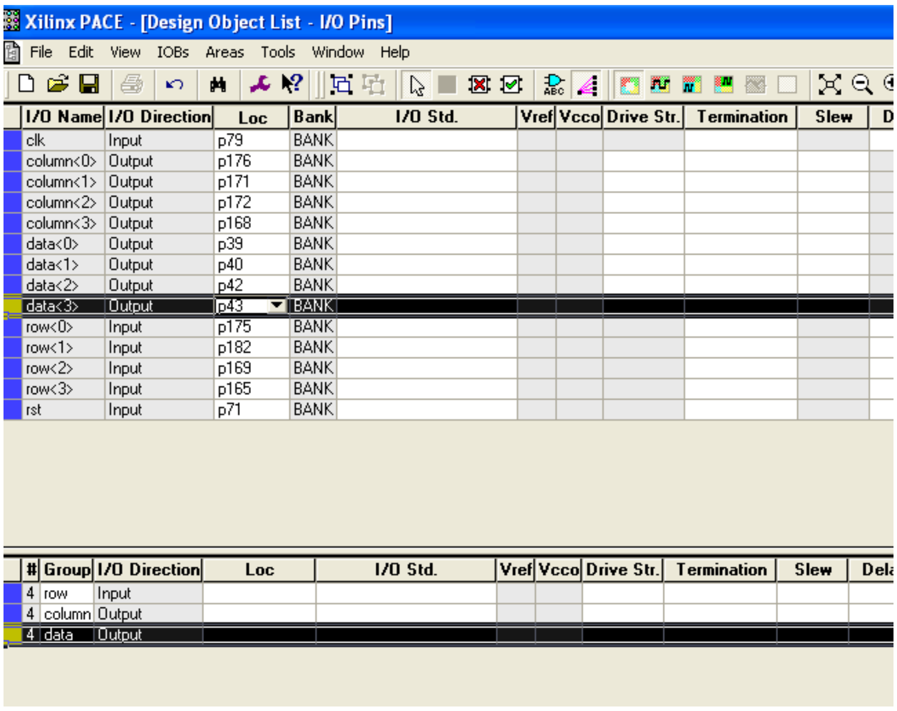
FINAL RESULT:

```
RTL Top Level Output File Name     : keypad.ngr
Top Level Output File Name         : keypad
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 14

Cell Usage :
# BELS                             : 30
#      INV                         : 1
#      LUT2                        : 4
#      LUT2_L                      : 1
#      LUT3                        : 1
#      LUT3_L                      : 5
#      LUT4                        : 12
#      LUT4_L                      : 6
# FlipFlops/Latches                  10
#      FDC                         : 4
#      FDCE                        : 2
#      FDP                         : 4
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 13
#      IBUF                        : 5
#      OBUF                        : 8
```

RTL SCHEMATIC:



PIN PACKAGE:


Xilinx PACE - [Design Object List - I/O Pins]
File Edit View IOBs Areas Tools Window Help

I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	D
clk	Input	p79	BANK							
column<0>	Output	p176	BANK							
column<1>	Output	p171	BANK							
column<2>	Output	p172	BANK							
column<3>	Output	p168	BANK							
data<0>	Output	p39	BANK							
data<1>	Output	p40	BANK							
data<2>	Output	p42	BANK							
data<3>	Output	p43	BANK							
row<0>	Input	p175	BANK							
row<1>	Input	p182	BANK							
row<2>	Input	p169	BANK							
row<3>	Input	p165	BANK							
rst	Input	p71	BANK							

#	Group	I/O Direction	Loc	I/O Std.	Vref	Vcco	Drive Str.	Termination	Slew	Dela
4	row	Input								
4	column	Output								
4	data	Output								

TITLE: To design CMOS Inverter, simulate with and without capacitive load	
NAME:	**SUBJECT: VLSI DESIGN AND TECHNOLOGY**
CLASS: BE	**ROLL NO:**
SEMESTER/YEAR: - 7TH SEM	**EXAM NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED: -	**REMARKS:**

EXPERIMENT NO – 06(A)

AIM: To design CMOS Inverter, simulate with and without capacitive load.

OBJECTIVE:
1. To design CMOS Inverter logic
2. To implement the design in micro wind.
3. To simulate the design in micro wind.

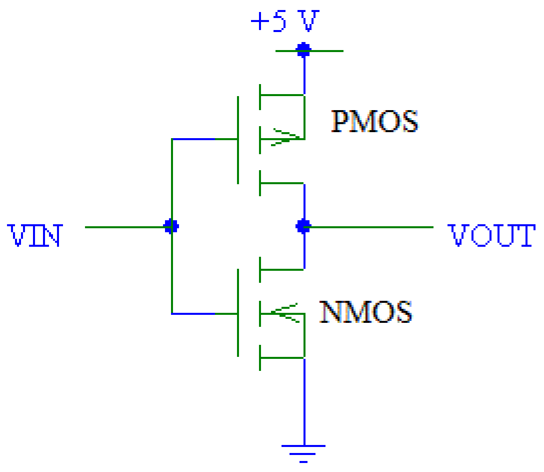
PREREQUISITE:

Students should know the constructs of micro wind software.

TOOLS USED:

1. Micro wind software with version 3 and above

PROBLEM STATEMENT:
Design a CMOS inverter and implement it in micro wind for simulation.



CMOS inverters (Complementary MOSFET Inverters) are some of the most widely used and adaptable MOSFET inverters used in chip design. They operate with very little power loss and at relatively high speed. Furthermore, the CMOS inverter has good logic buffer characteristics, in that, its noise margins in both low and high states are large.

This short description of CMOS inverters gives a basic understanding of the how a CMOS inverter works. It will cover input/output characteristics, MOSFET states at different input voltages, and power losses due to electrical current.

A CMOS inverter contains a PMOS and a NMOS transistor connected at the drain and gate terminals, a supply voltage VDD at the PMOS source terminal, and a ground connected at the NMOS source terminal, were VIN is connected to the gate terminals and VOUT is connected to the drain terminals.(See diagram). It is important to notice that the CMOS does not contain any resistors, which makes it more power efficient that a regular resistor-MOSFET inverter.As the voltage at the input of the CMOS device varies between 0 and 5 volts, the state of the NMOS and

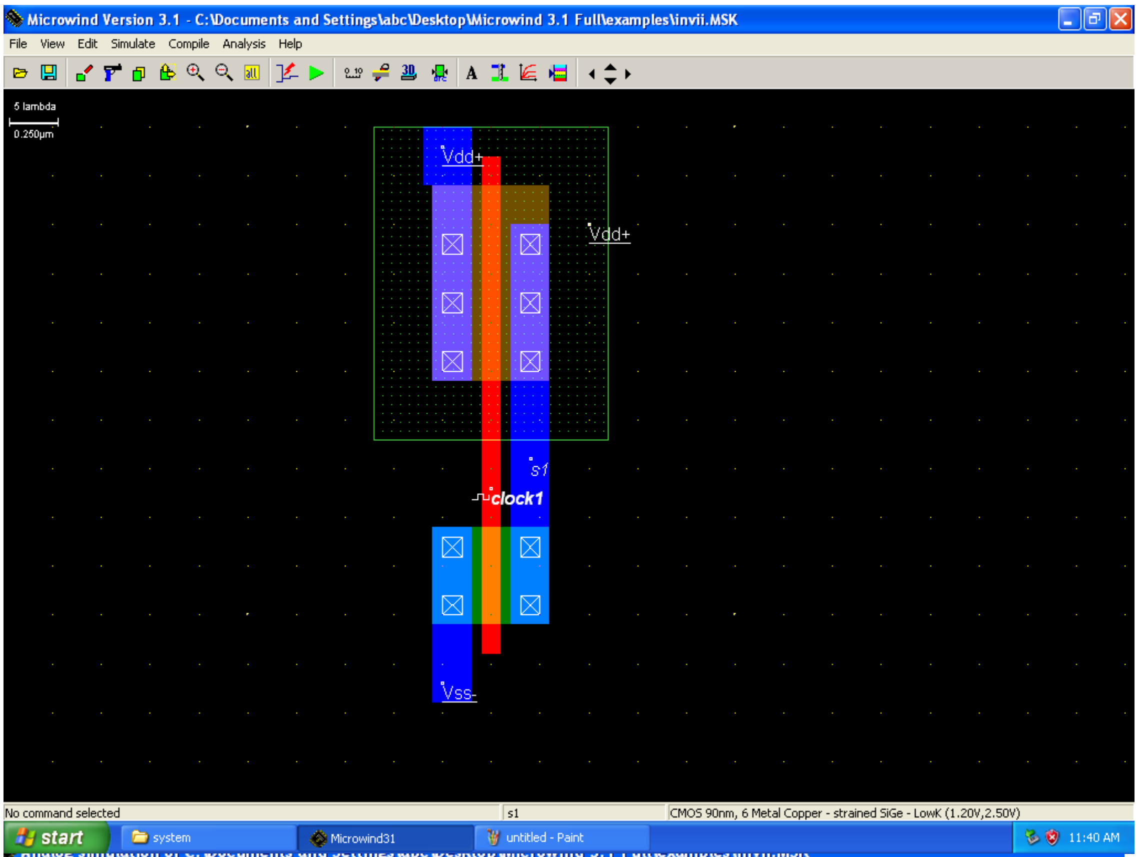
PMOS varies accordingly. If we model each transistor as a simple switch activated by VIN, the inverter’s operations can be seen very easily:

MOSFET	**Condition on MOSFET**	**State of MOSFET**
NMOS	Vgs<Vtn	OFF
NMOS	Vgs>Vtn	ON
PMOS	Vsg<Vtp	OFF
PMOS	Vsg>Vtp	ON

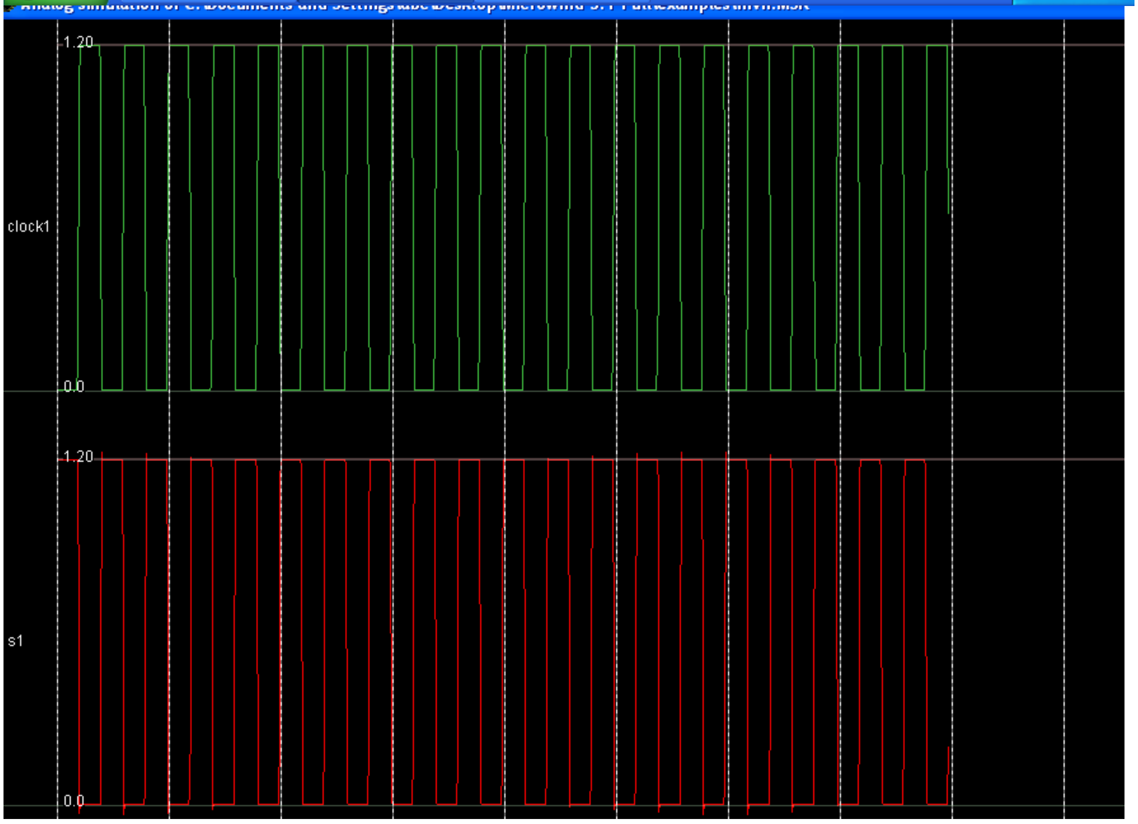
When VIN is low, the NMOS is "off", while the PMOS stays "on": instantly charging VOUT to logic high. When Vin is high, the NMOS is "on and the PMOS is "on: draining the voltage at VOUT to logic low.

CONCLUSION:
We have performed the design of CMOS Inverter, simulate with and without capacitive load using microwind software and we have learnt about CMOS inverter along with its working and truth table

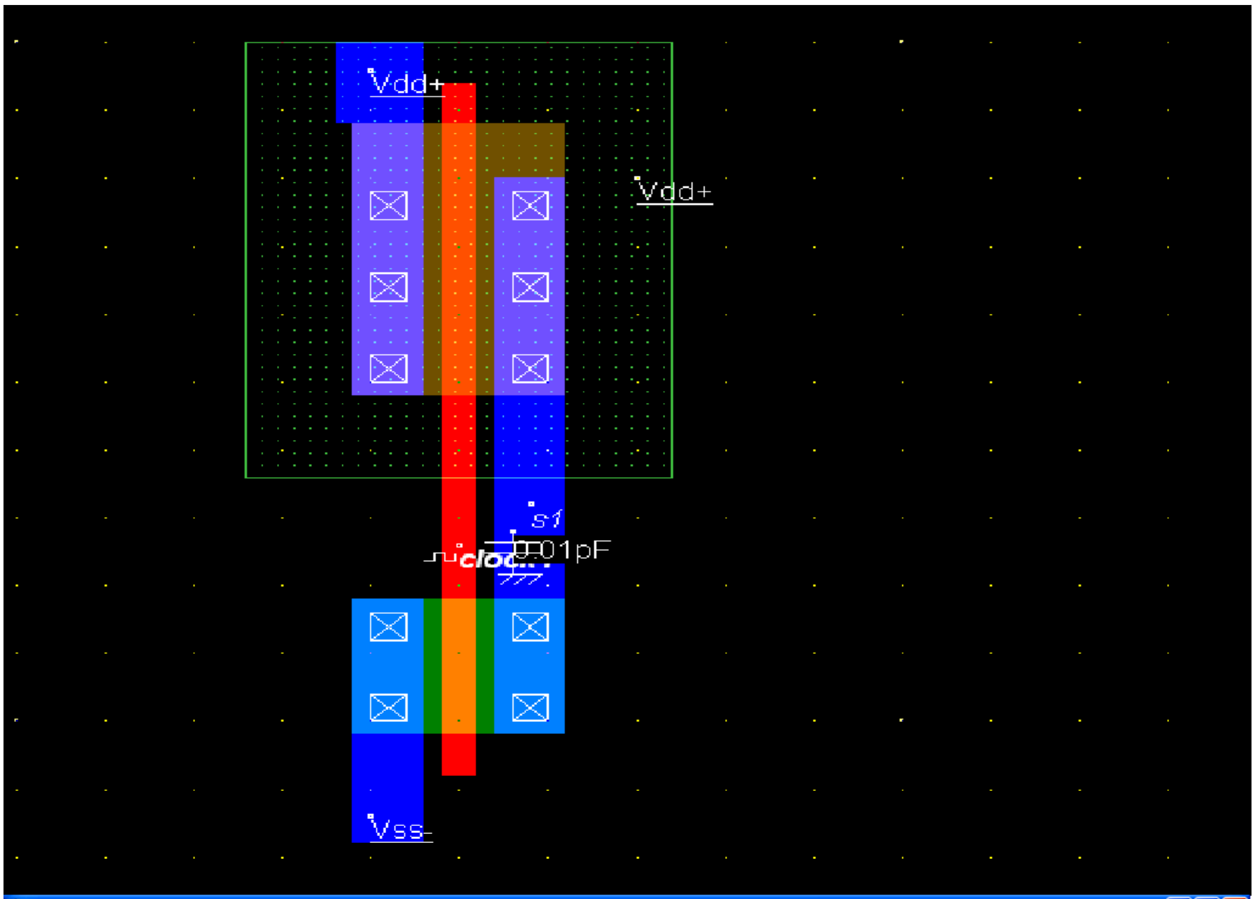
CMOS INVERTER WITHOUT CAPACITOR:

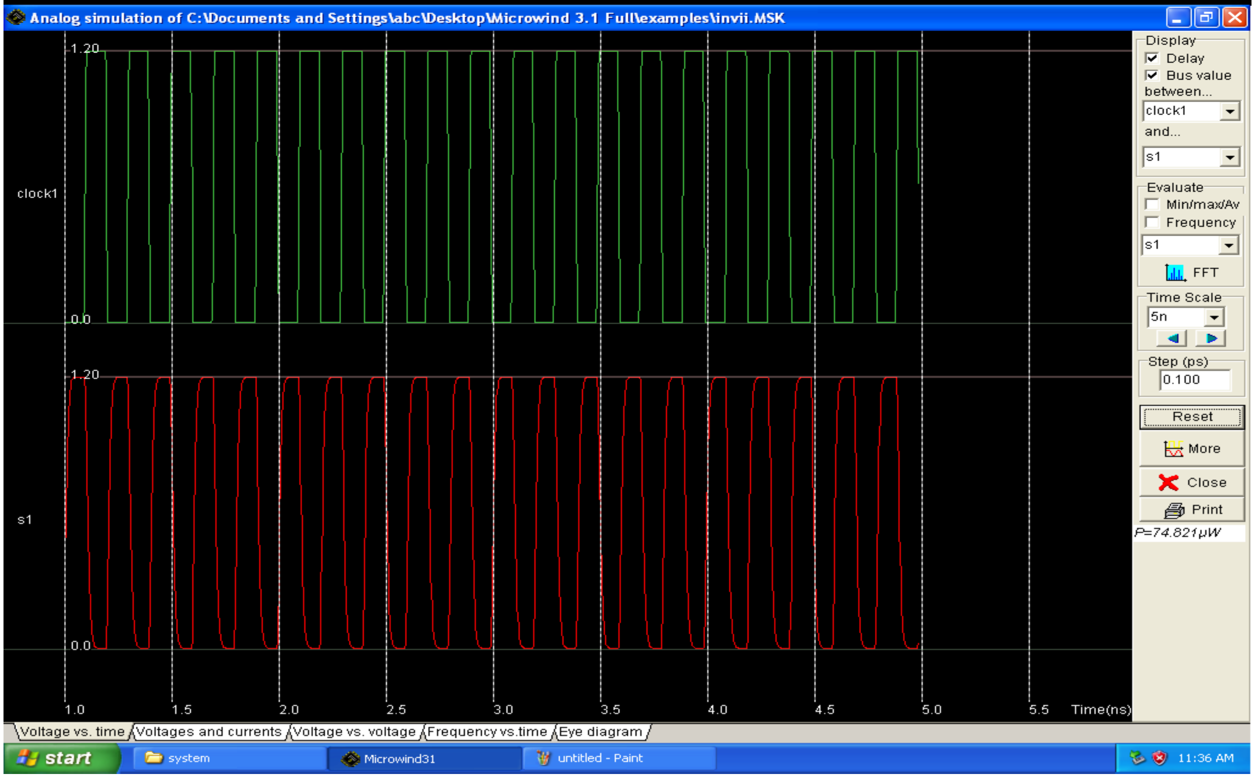

Microwind Version 3.1 - C:\Documents and Settings\abc\Desktop\Microwind 3.1 Full\examples\invii.MSK
File View Edit Simulate Compile Analysis Help
5 lambda
0.250µm
Vdd+
Vdd+
s1
clock1
Vss-
start
system
Microwind31
untitled - Paint
11:40 AM

No command selected | s1 | CMOS 90nm, 6 Metal Copper - strained SiGe - LowK (1.20V,2.50V)



CMOS INVERTER WITH CAPACITOR :




Analog simulation of C:\Documents and Settings\abc\Desktop\Microwind 3.1 Full\examples\invii.MSK

Display: [✓ Delay] [✓ Bus value] between... clock1 and... s1

Evaluate: Min/max/Av, Frequency, s1

Time Scale: 5n

Step (ps): 0.100

P=74.821µW

TITLE: To design CMOS NAND and NOR gate, simulate with and without capacitive load	
NAME:	**SUBJECT: VLSI DESIGN AND TECHNOLOGY**
CLASS: BE	**ROLL NO:**
SEMESTER/YEAR: - 7TH SEM	**EXAM NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED: -	**REMARKS:**

EXPERIMENT NO – 06(B)

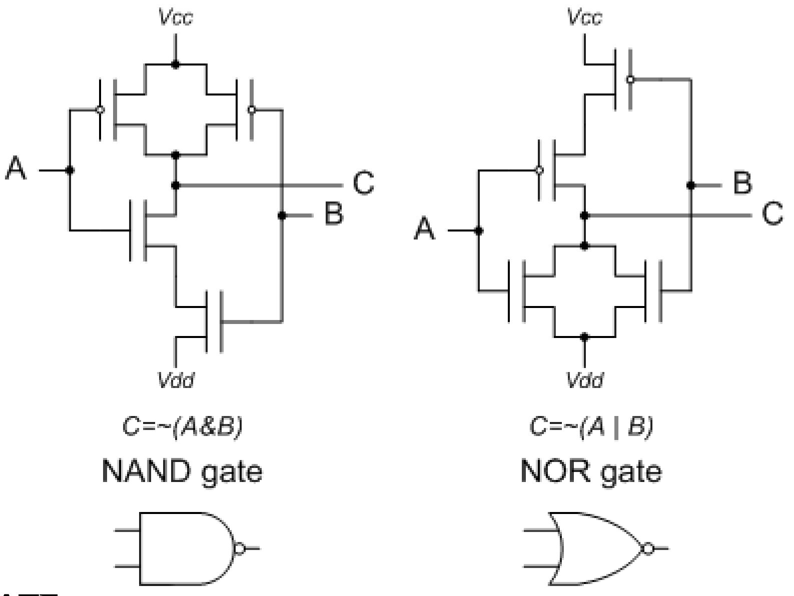
AIM: To design CMOS NAND and NOR gate, simulate with and without capacitive load.

OBJECTIVE:
1. To design CMOS NAND and NOR logic
2. To implement the design in micro wind.
3. To simulate the design in micro wind.

PREREQUISITE:
Students should know the constructs of micro wind software.

TOOLS USED:
1. Micro wind software with version 3 and above

PROBLEM STATEMENT:
Design a CMOS NAND and NOR and implement it in micro wind for simulation.



CMOS NAND GATE:

When a path consists of two transistors in series, both transistors must have low resistance to the corresponding supply voltage, modelling an AND. When a path consists of two transistors in parallel, either one or both of the transistors must have low resistance to connect the supply voltage to the output, modelling an OR.

If both of the A and B inputs are high in the circuit, then both the NMOS transistors (bottom half of the diagram) will conduct, neither of the PMOS transistors (top half) will conduct, and a conductive path will be established between the output and V_{ss} (ground), bringing the output low. If either of the A or B inputs is low, one of the NMOS transistors will not conduct, one of the PMOS transistors will, and a conductive path will be established between the output and V_{dd} (voltage source), bringing the output high.

An advantage of CMOS over NMOS is that *both* low-to-high and high-to-low output transitions are fast since the pull-up transistors have low resistance when switched on, unlike the load resistors in NMOS logic. In addition, the output signal swings the full voltage between the low and high rails. This strong, more nearly symmetric response also makes CMOS more resistant to noise.

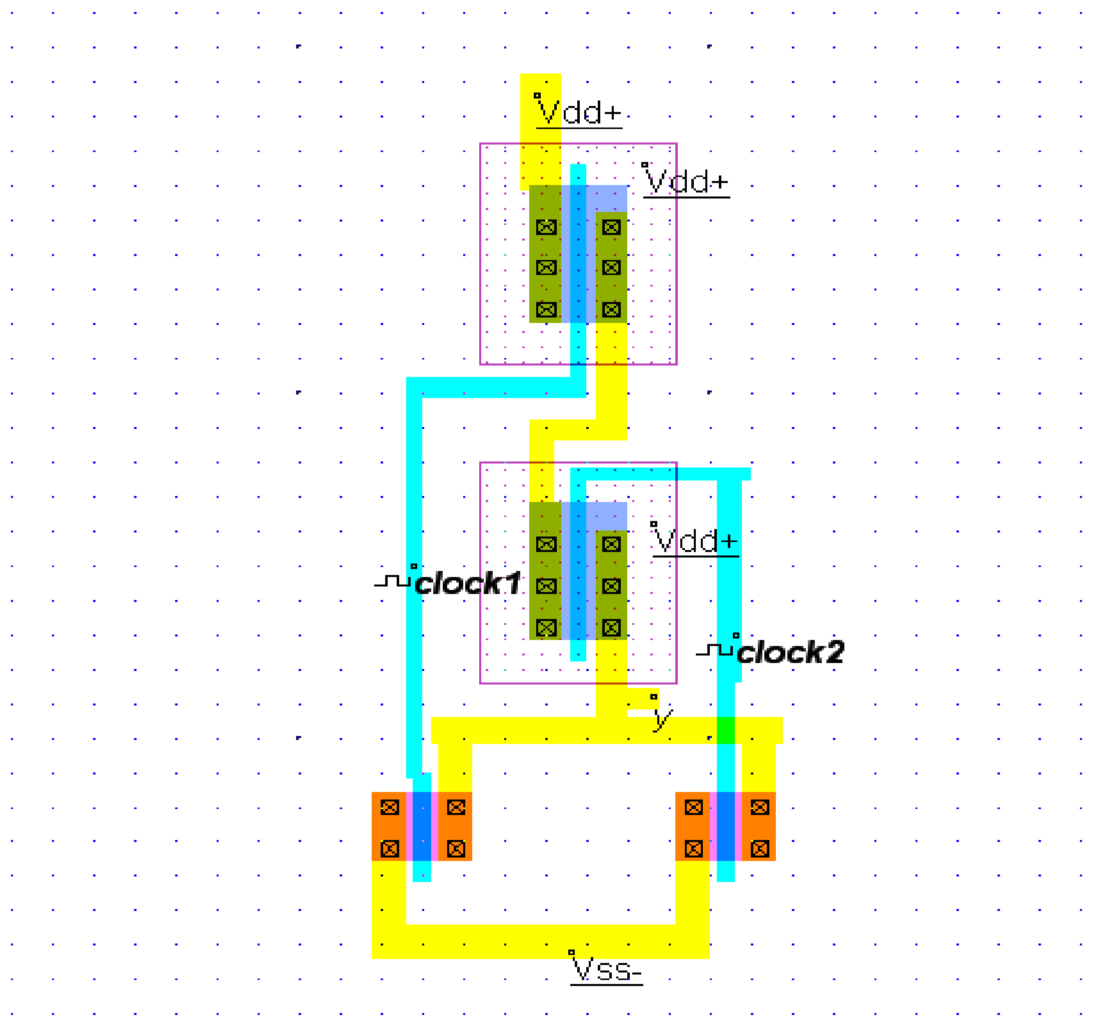
CMOS NOR GATE:
A CMOS NOR gate circuit uses four MOSFETs just like the NAND gate, except that its transistors are differently arranged. Instead of two paralleled sourcing (upper) transistors connected to Vdd and two series-connected sinking (lower) transistors connected to ground, the NOR gate uses two series-connected sourcing transistors and two parallel-connected sinking transistors

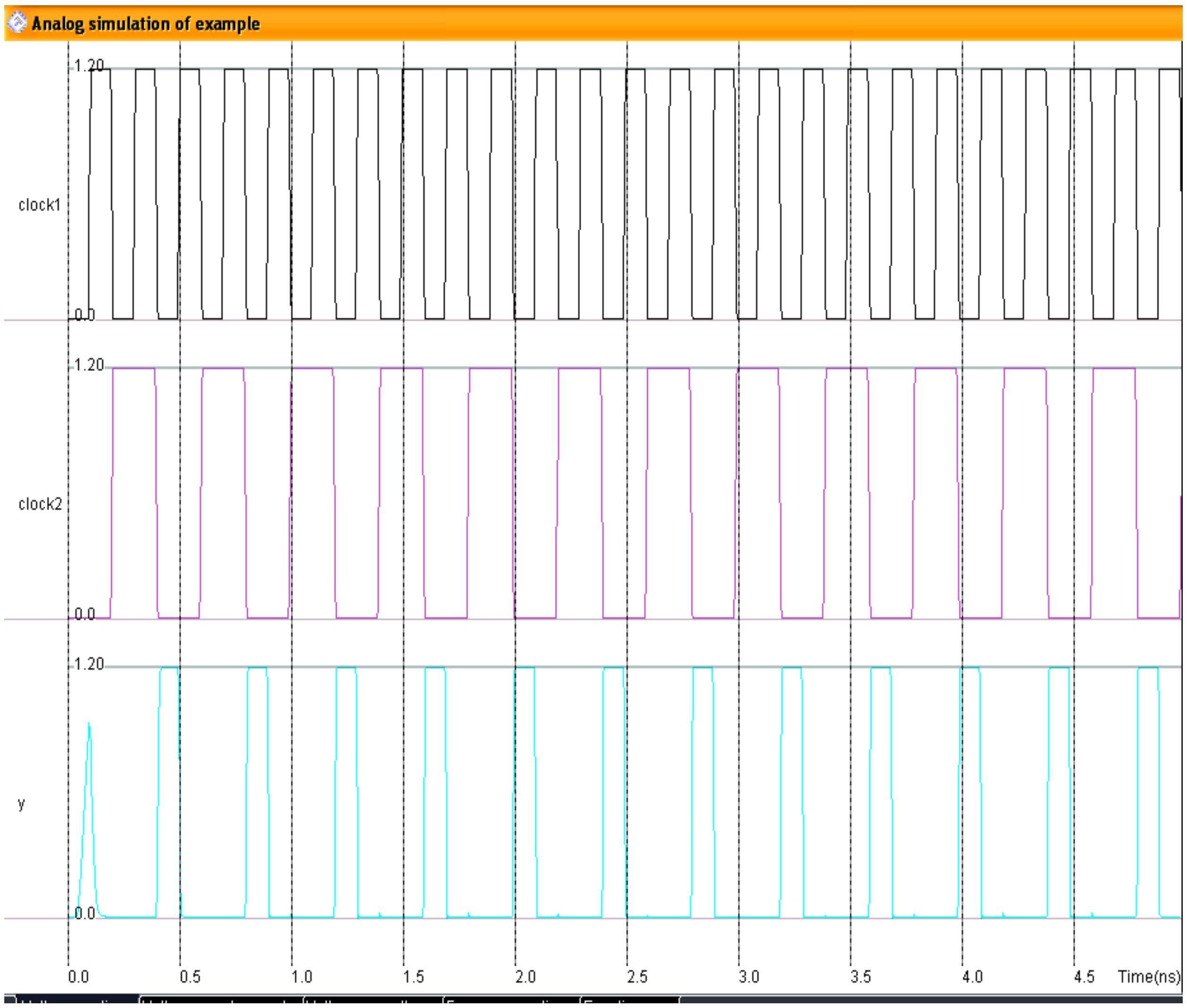
As with the NAND gate, transistors Q1 and Q3 work as a complementary pair, as do transistors Q2 and Q4. Each pair is controlled by a single input signal. If either input A or input B are "high" (1), at least one of the lower transistors (Q3 or Q4) will be saturated, thus making the output "low" (0). Only in the event of both inputs being "low" (0) will both lower transistors be in cut-off mode and both upper transistors be saturated, the conditions necessary for the output to go "high" (1).
This behavior, of course, defines the NOR logic function.

CONCLUSION:

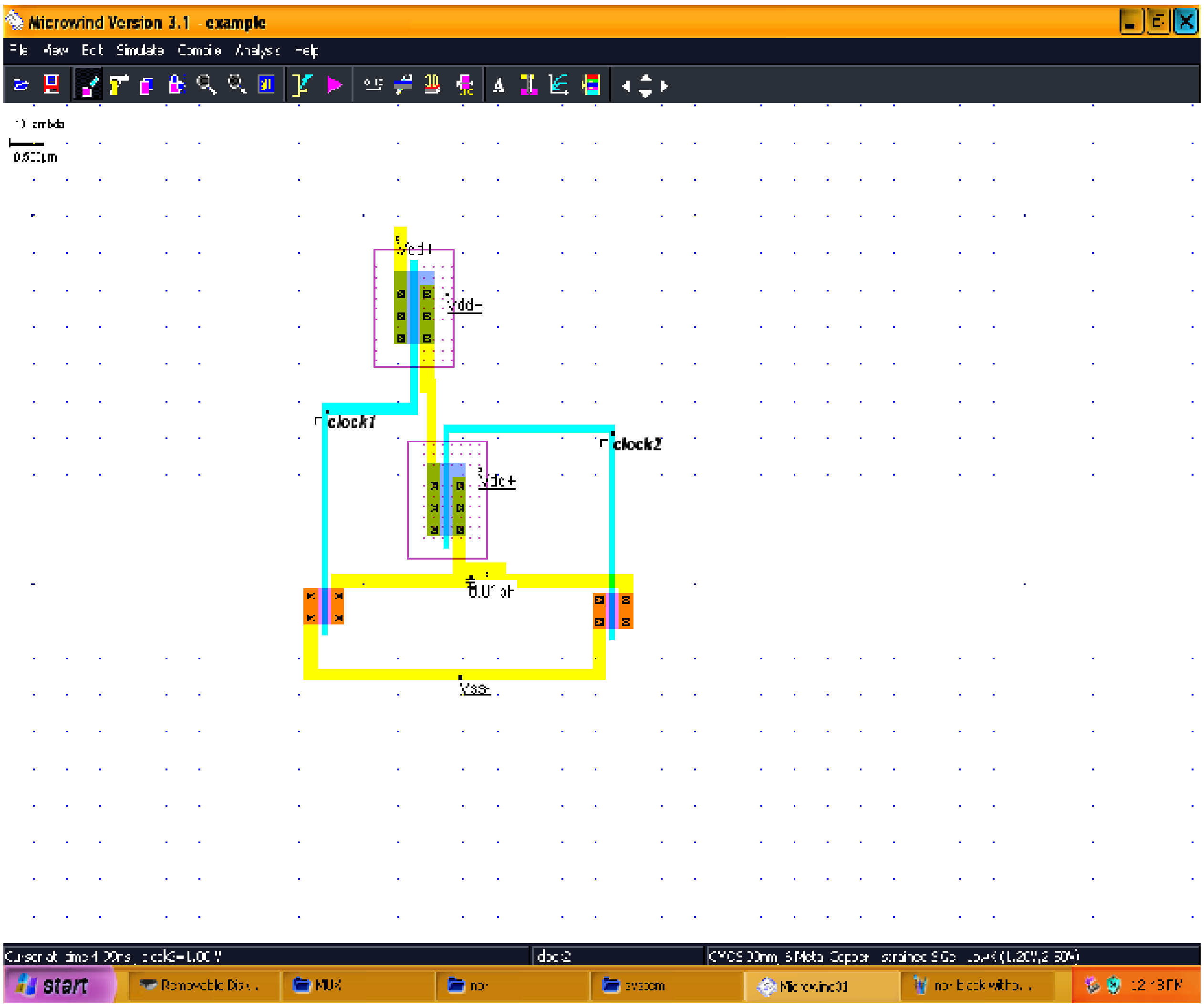
We have performed the design of CMOS NAND and NOR gate, simulate with and without capacitive load using micro wind software and learnt about CMOS NAND and NOR gate circuit uses four MOSFETs, its construction ,working and proved its truth table with the help of simulation.

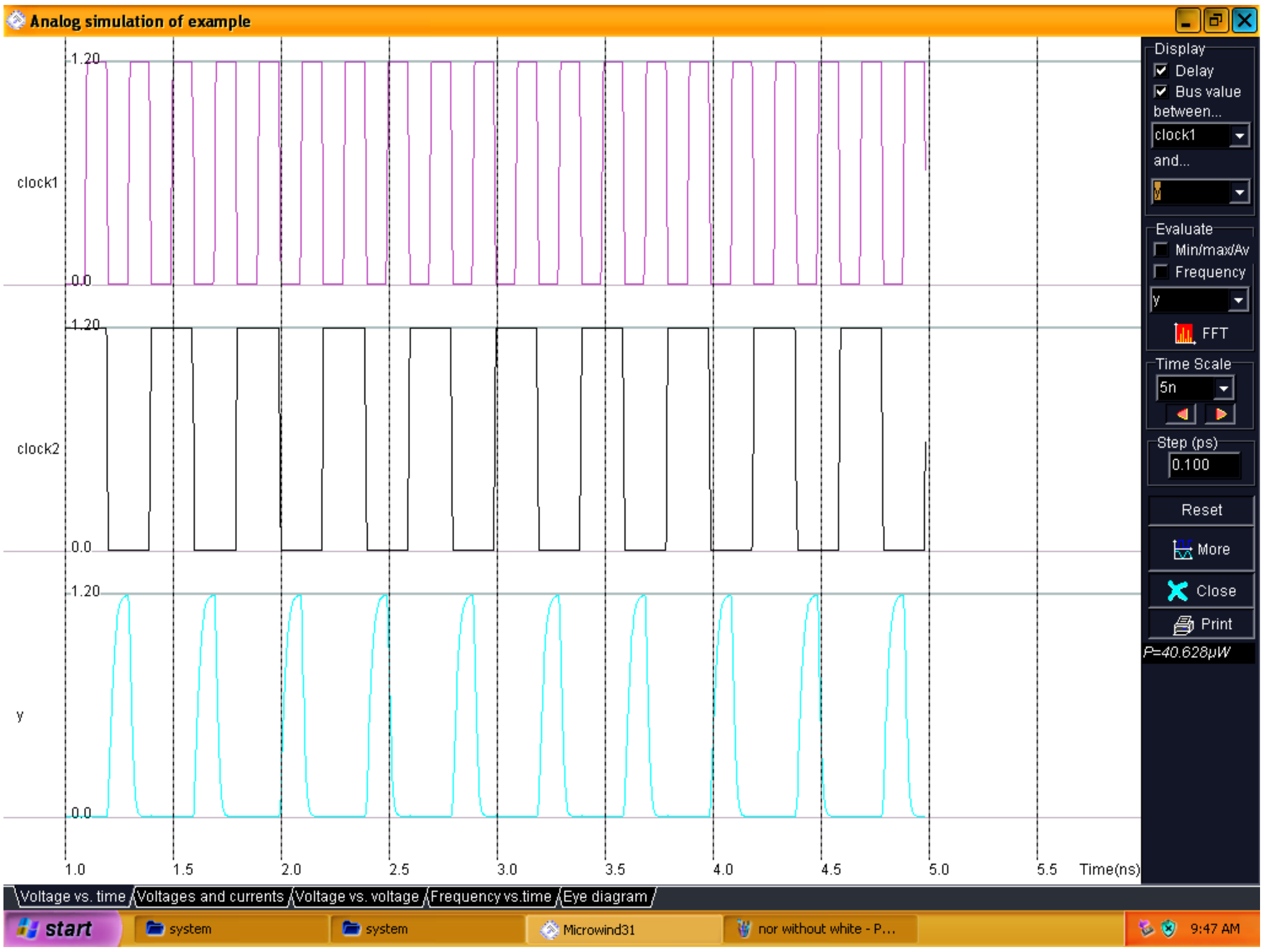
NOR : WITHOUT CAPACITOR:



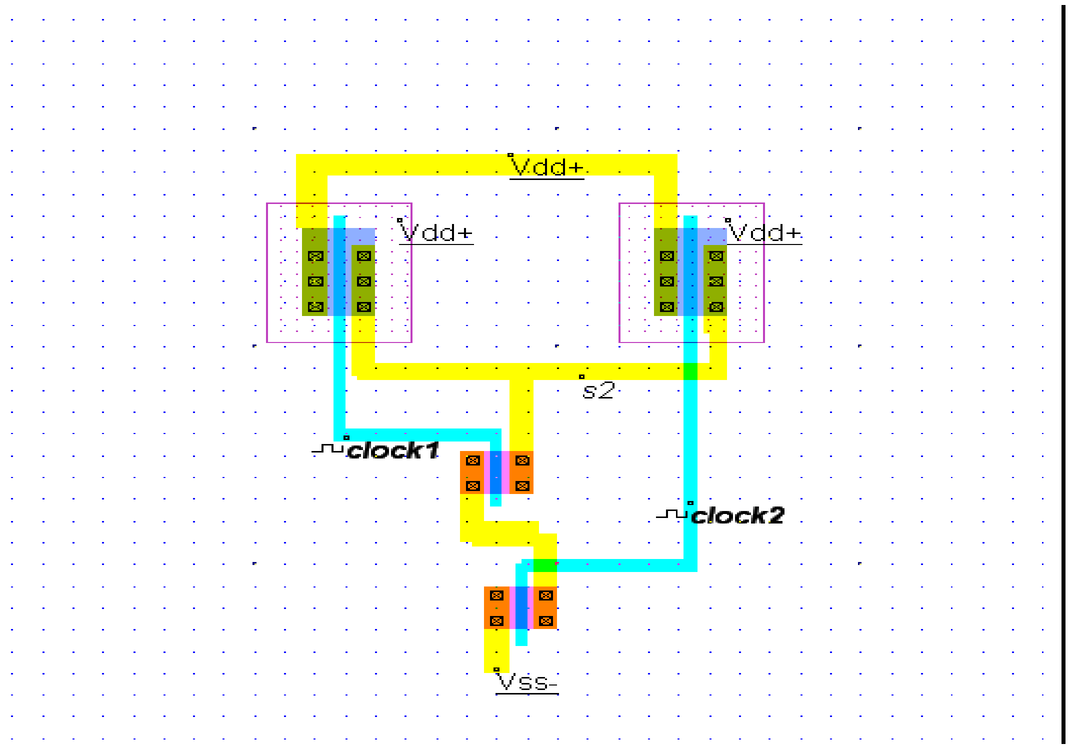

Analog simulation of example
clock1
clock2
y
1.20
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
Time(ns)

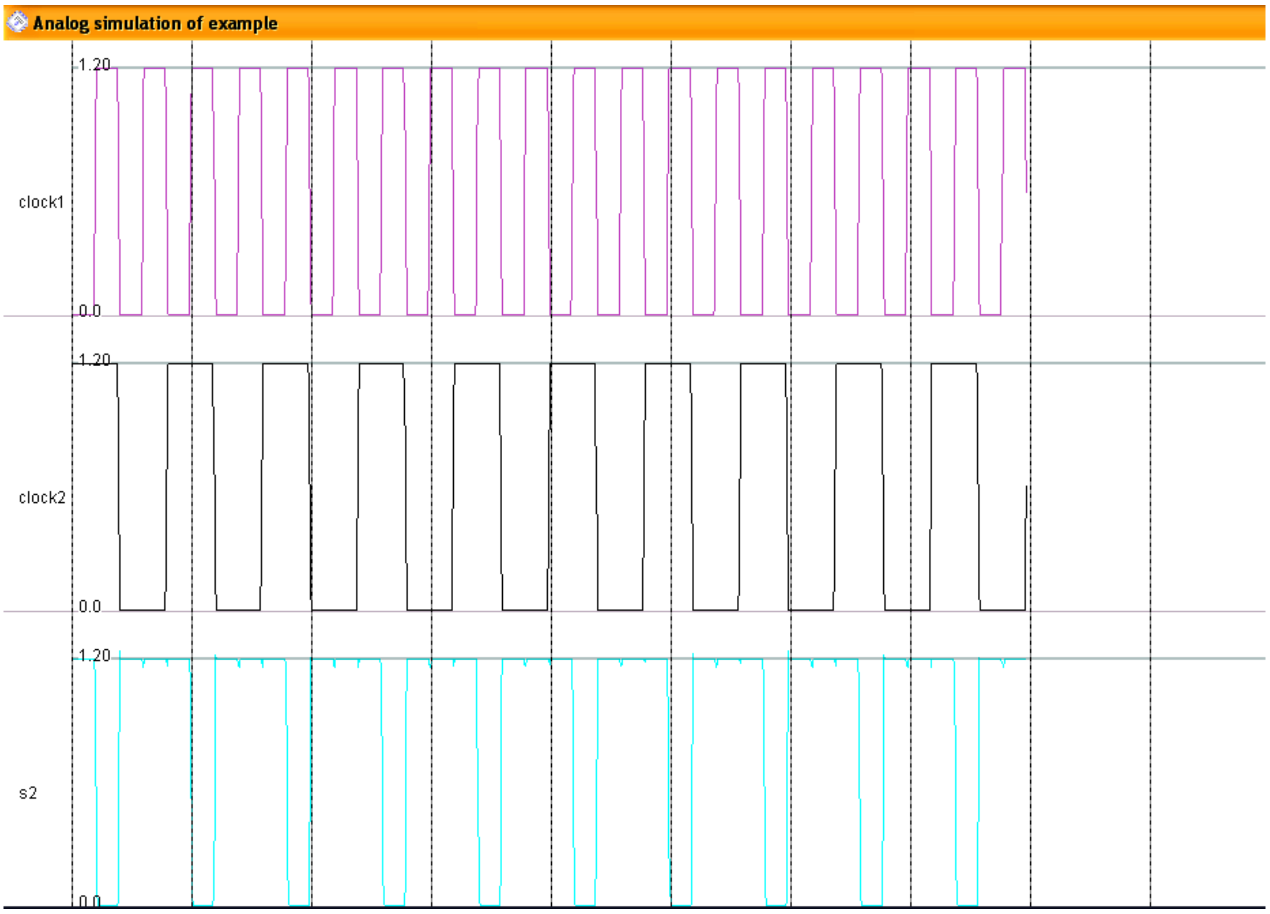
NOR : WITH CAPACITOR


Microwind Version 3.1 example
File View Edit Simulate Compile Analysis Help
Vdd+
Vdd+
clock1
clock2
Vdd+
0.01 pF
Vss-
start

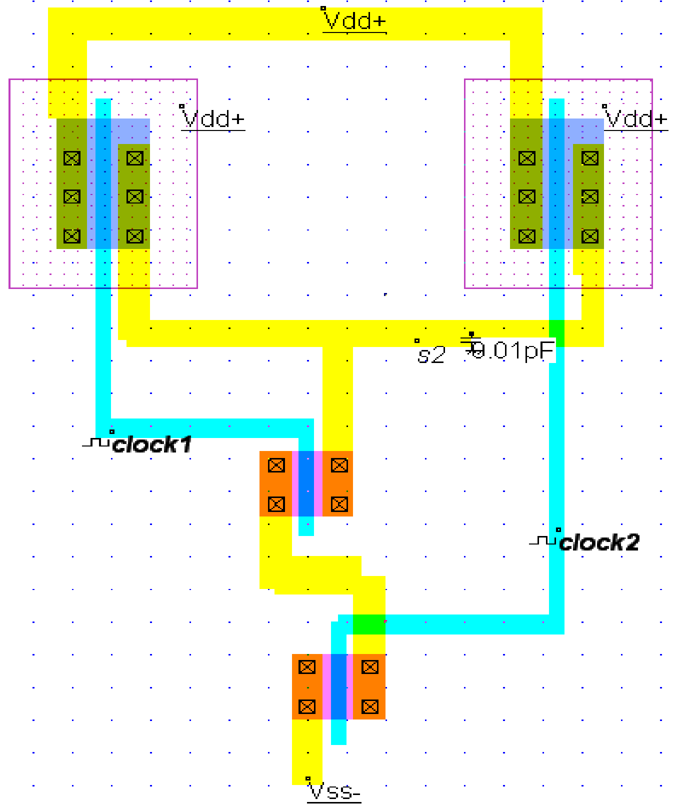


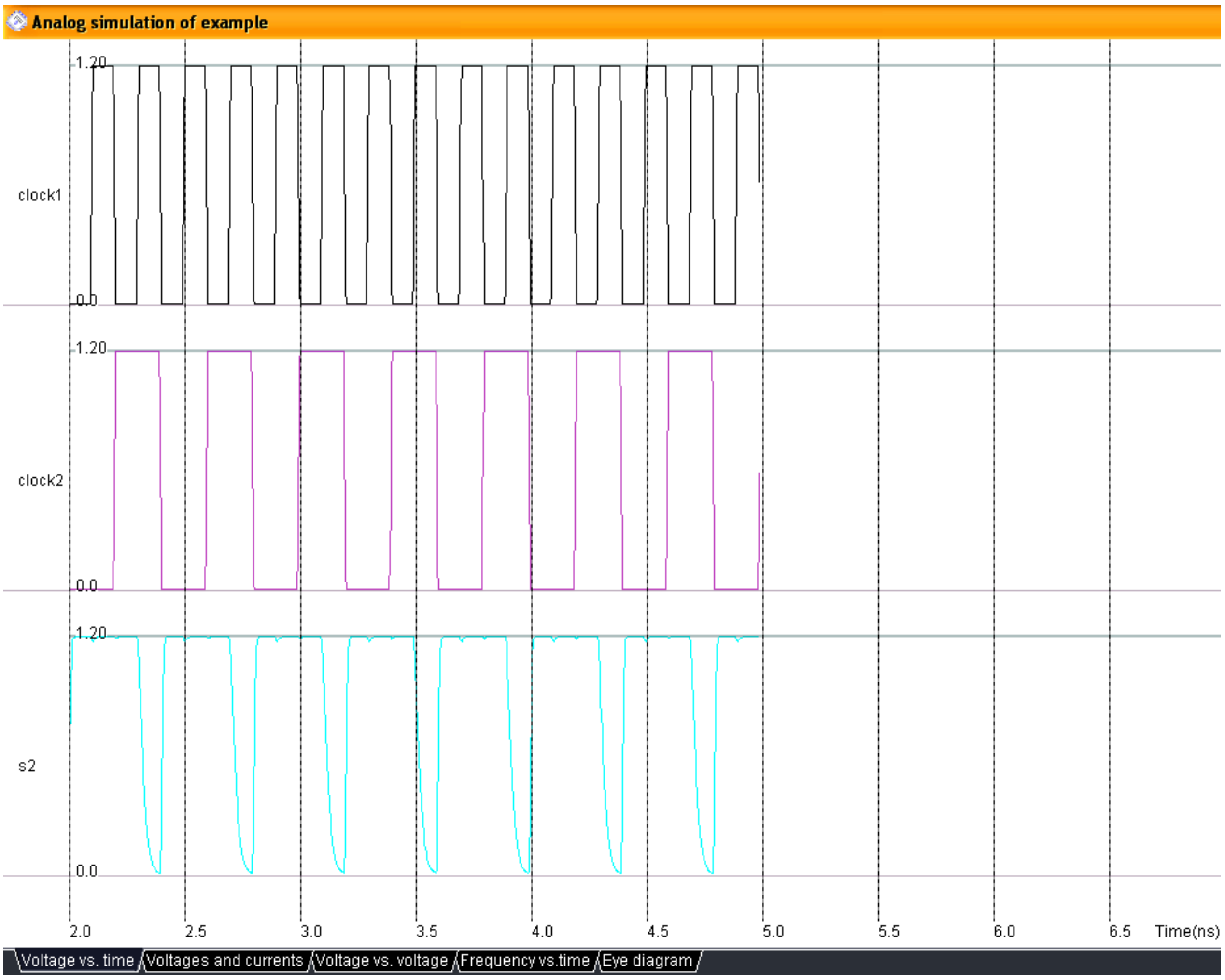
NAND : WITHOUT CAPACITOR





NAND : WITH CAPACITOR




Analog simulation of example
clock1
1.20
0.0
clock2
1.20
0.0
s2
1.20
0.0
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
Time(ns)
Voltage vs. time
Voltages and currents
Voltage vs. voltage
Frequency vs.time
Eye diagram

TITLE: To design Half adder by using transmission gates and Simulate using micro wind	
NAME:	**SUBJECT: VLSI DESIGN AND TECHNOLOGY**
CLASS: BE	**ROLL NO:**
SEMESTER/YEAR: - 7TH SEM	**EXAM NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED: -	**REMARKS:**

EXPERIMENT NO – 06(C)

AIM: To design Half adder by using transmission gates and Simulate using micro wind.

OBJECTIVES:

1. Designing of Half Adder using transmission gates.

PRE-LAB REQUISITES:

1. Concept of transmission gate
2. Comparison of transmission gate with CMOS.
3. Designing of digital logic using transmission gates.

TOOLS USED:

1. PC
2. Micro wind backend tool.

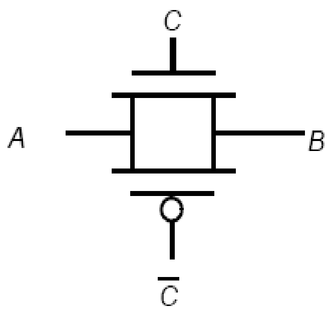
THEORY:

The most widely used solution to deal with the voltage drops induced by pass transistors is the use of *transmission gates*. The primary limitation of NMOS or PMOS only pass gate is the threshold drop (NMOS pass device pass a strong 0 while passing a weak 1and PMOS pass devices pass a strong 1 while passing a weak 0). The ideal approach is to use the NMOS device to pull-down and the PMOS device to pull-up. The transmission gate combines the best of both device by placing a NMOS device in parallel with a PMOS device. The control signals to the transmission gate (*C* and *C*) are complementary. The transmission gate acts as a bidirectional switch controlled by the gate signal *C*. When *C* = 1, both MOSFETs are on, allowing the signal to pass through the gate. In short,

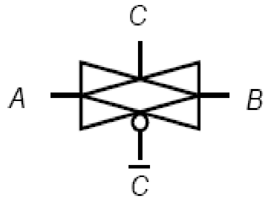
A=B if C=1

On the other hand, *C* = 0 places both transistors in cut-off, creating an open circuit between Nodes A and B

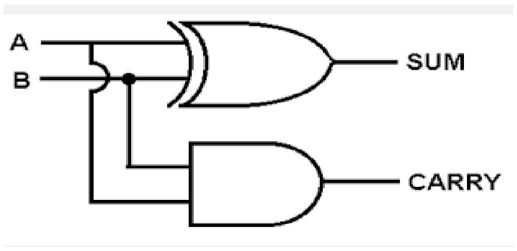
DIGRAMS:



(a) Circuit



(b) Symbolic representation



Input		Output	
A	B	Sum	Carry
1	1	0	1
1	0	1	0
0	1	1	0
0	0	0	0

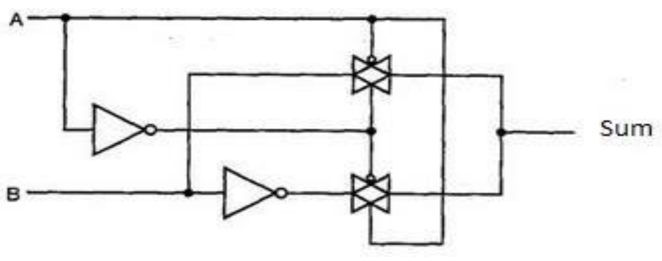


Fig a: XOR gate using TG

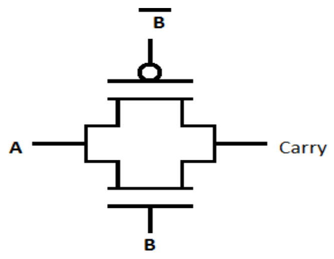


Fig b: AND gate using TG

Figure 1: Half Adder using Transmission Gate

PROCEDURE:

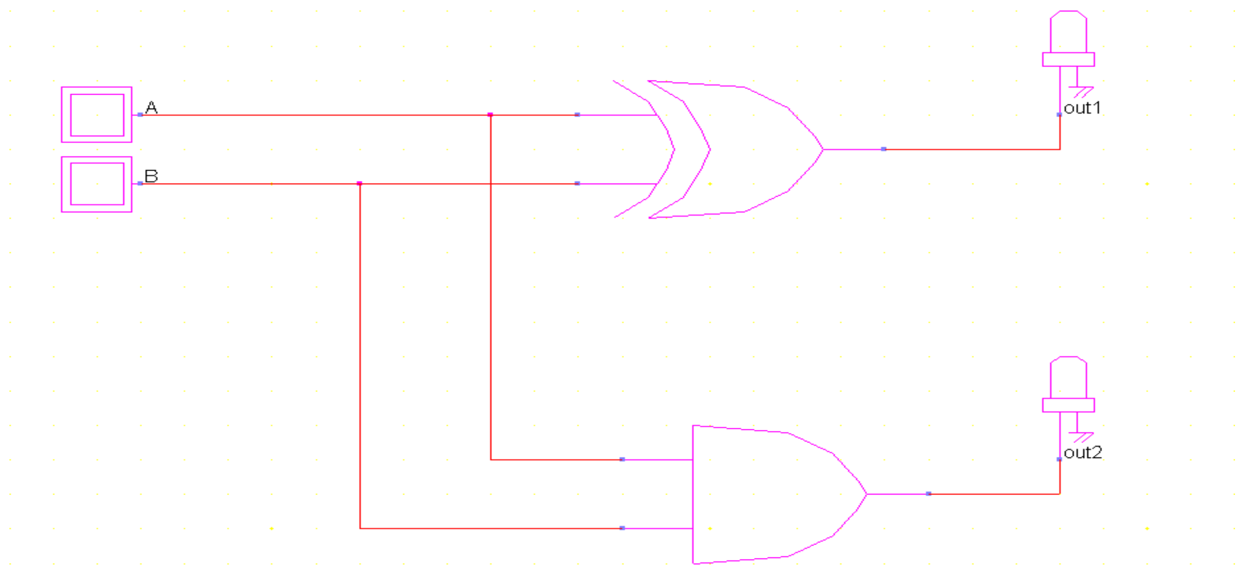
1. Construct Half Adder using transmission gate in Micro wind software.
2. Simulate this.
3. Check Output

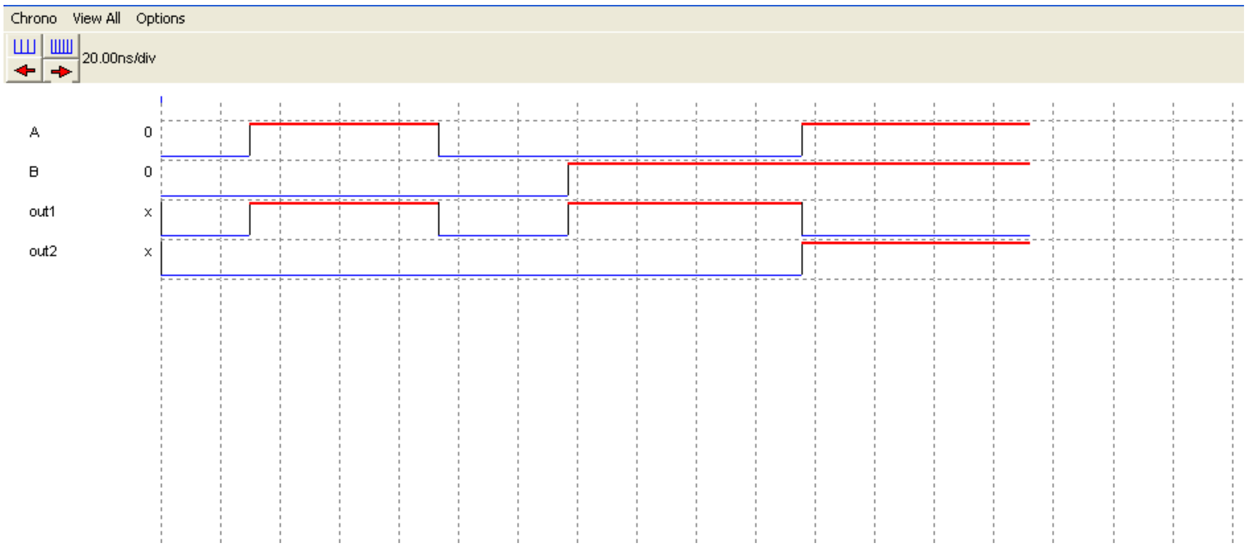
OBSERVATIONS:

In simulation we verify the truth table of Half Adder.

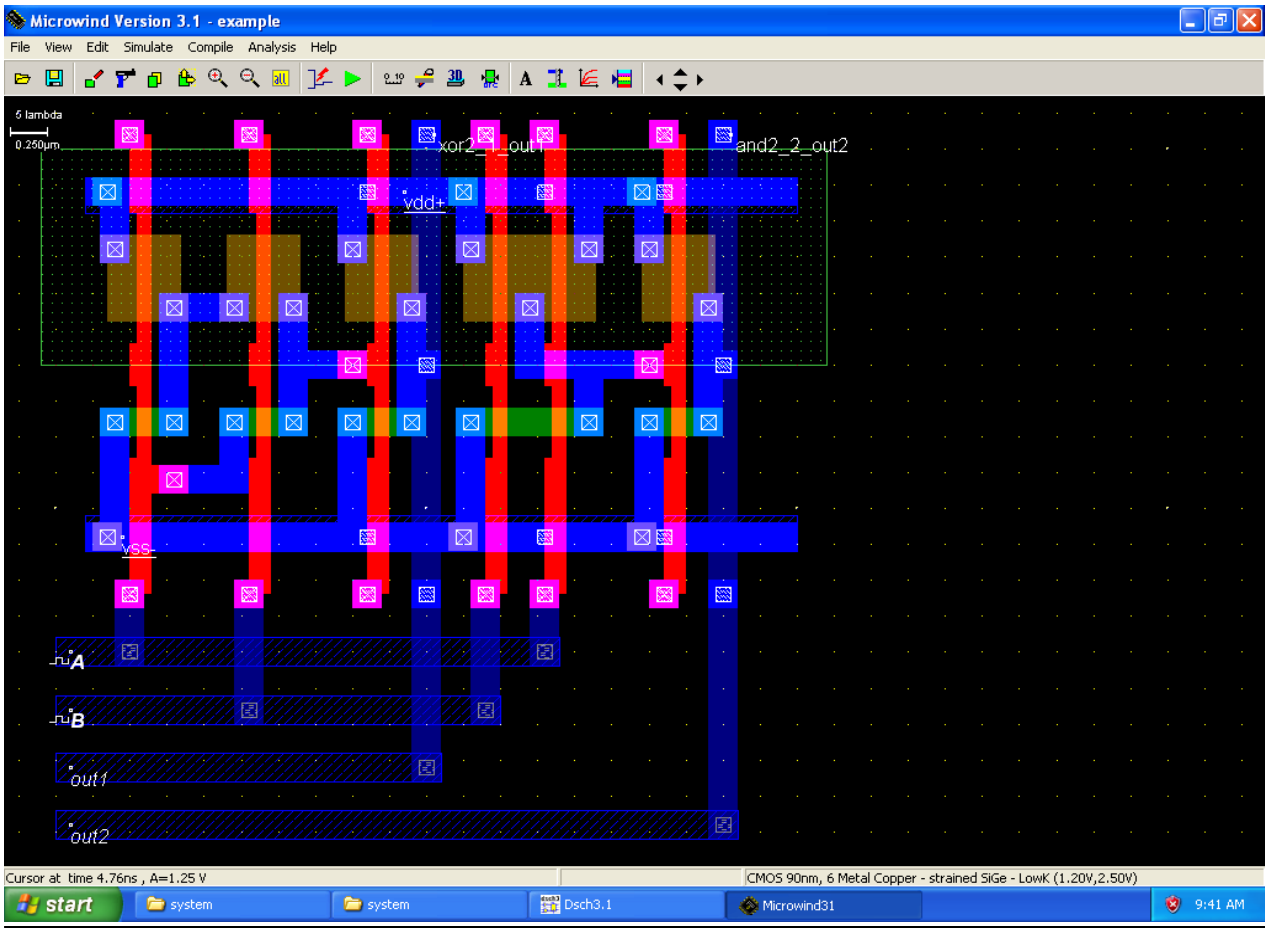
CONCLUSIONS:

We have performed the design of Half adder by using transmission gates and simulated using micro wind software, studied about the concept of transmission gate, comparison of transmission gate with CMOS and designing of digital logic using transmission gates



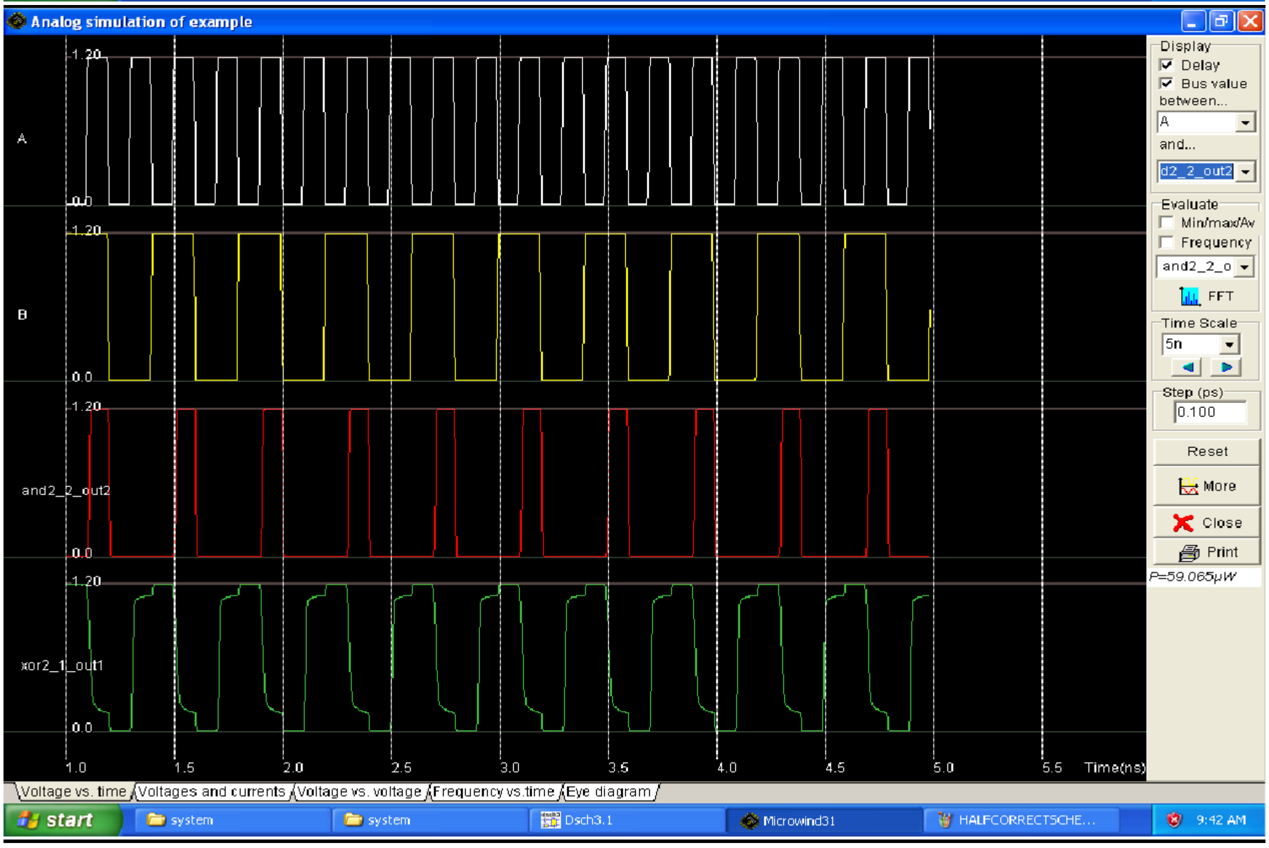

Chrono
View All
Options
20.00ns/div
A 0
B 0
out1 x
out2 x

HALF ADDER :



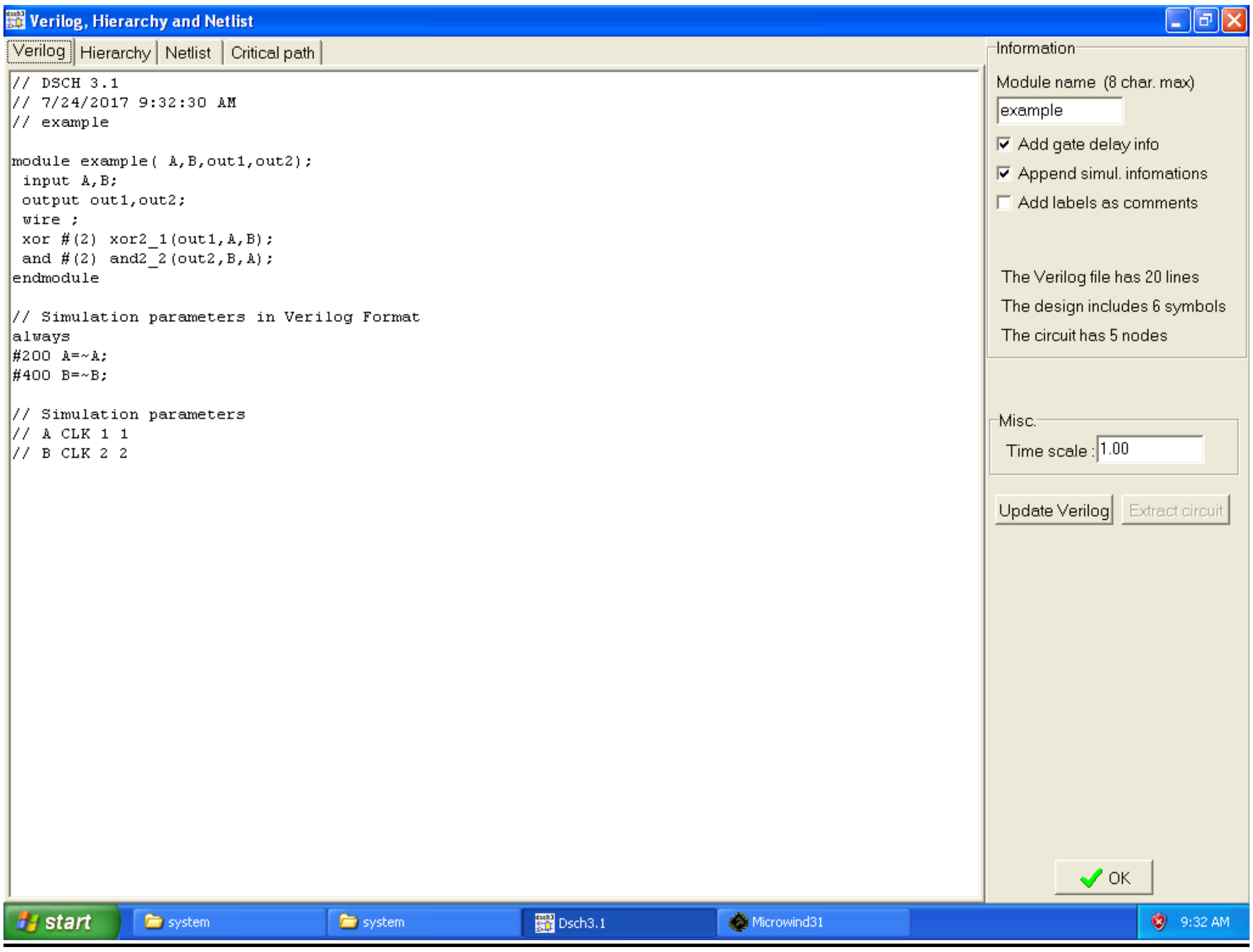
Cursor at time 4.76ns , A=1.25 V

CMOS 90nm, 6 Metal Copper - strained SiGe - LowK (1.20V,2.50V)



Display
☑ Delay
☑ Bus value
between...
A
and...
d2_2_out2
Evaluate
☐ Min/max/Av
☐ Frequency
and2_2_o
Time Scale
5n
Step (ps)
0.100
P=59.065µW

VERILOG:



```
// DSCH 3.1
// 7/24/2017 9:32:30 AM
// example

module example( A,B,out1,out2);
 input A,B;
 output out1,out2;
 wire ;
 xor #(2) xor2_1(out1,A,B);
 and #(2) and2_2(out2,B,A);
endmodule

// Simulation parameters in Verilog Format
always
#200 A=~A;
#400 B=~B;

// Simulation parameters
// A CLK 1 1
// B CLK 2 2
```

Information

Module name (8 char. max): example

- [✓] Add gate delay info
- [✓] Append simul. infomations
- [ ] Add labels as comments

The Verilog file has 20 lines
The design includes 6 symbols
The circuit has 5 nodes

Misc.

Time scale : 1.00

TITLE: To design 2:1 Mux by conventional method and by using Transmission gates, comparison of them, prepare layout in multi metal layers and simulate	
NAME:	**SUBJECT: VLSI DESIGN AND TECHNOLOGY**
CLASS: BE	**ROLL NO:**
SEMESTER/YEAR: - 7TH SEM	**EXAM NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED: -	**REMARKS:**

EXPERIMENT NO – 07

AIM: To design 2:1 Mux by conventional method and by using Transmission gates, comparison of them, prepare layout in multi metal layers and simulate.

OBJECTIVE:
1. To design 2:1 Mux by conventional method and by using Transmission gates
2. Comparison of them.
3. To simulate the design in micro wind.

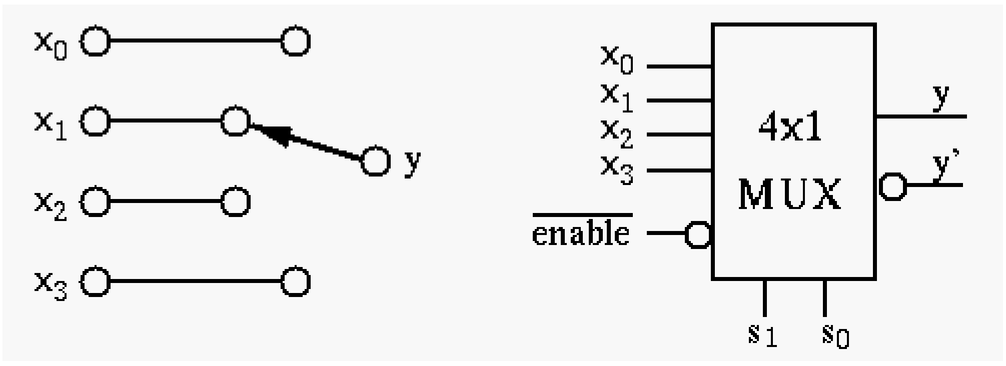
PREREQUISITE:
Students should know the constructs of micro wind software.

TOOLS USED:
1. Micro wind software with version 3 and above.

PROBLEM STATEMENT:
Design 2:1 Mux by conventional method and by using Transmission gates, compare them, prepare layout in multi metal layers and simulate

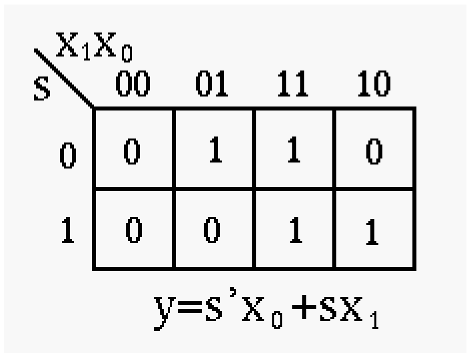
Multiplexer (MUX)

An MUX has N inputs and one output. Under the control of $n = log_2 N$ *selection* signals, one of the inputs is passed on to the output.



First consider the truth table of a 2x1 MUX with three inputs x_0, x_1 and s and only one output y :

	s	x_1	x_0	y
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1



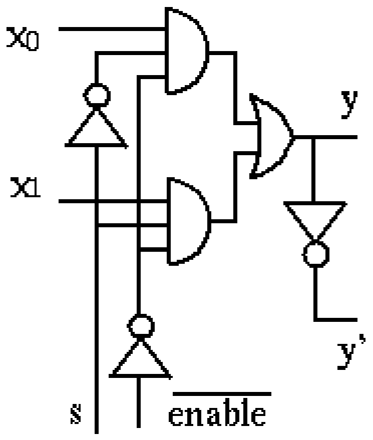
$y = s'x_0 + sx_1$

This truth table can be simplified by allowing Don't-cares in the table:

s	x_1	x_0	y
0	x	0	0
0	x	1	1
1	0	x	0
1	1	x	1

Finally, if we allow variables in the truth table (variable-entered map VEM), the truth table can be further simplified to be

s	y
0	x_0
1	x_1



2x1 MUX

A 4x1 MUX has $N = 2^n = 4$ inputs x_0, x_1, x_2 and x_3, and $n = log_2 4 = 2$ selections s_0 and s_1. Its output y is one of the four inputs depending on the selections. The truth table for a 4x1 MUX:

s_1	s_0	x_3	x_2	x_1	x_0	y
0	0	x	x	x	0	0
0	0	x	x	x	1	1
0	1	x	x	0	x	0
0	1	x	x	1	x	1
1	0	x	0	x	x	0
1	0	x	1	x	x	1
1	1	0	x	x	x	0
1	1	1	x	x	x	1

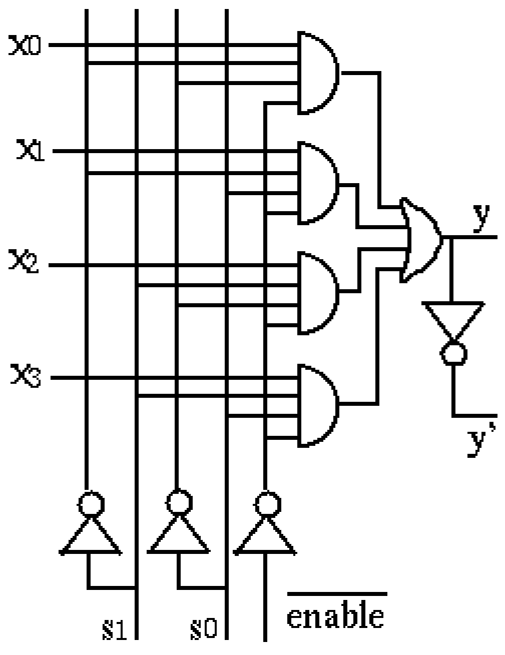
This truth table can be simplified by the VEM:

s_1	s_0	y
0	0	x_0
0	1	x_1
1	0	x_2
1	1	x_3

We see that each minterm of the two selection bits corresponds to an input:

$$y = \sum_{i=0}^{2^n-1} m_i x_i = (s_1' s_0')x_0 + (s_1' s_0)x_1 + (s_1 s_0')x_2 + (s_1 s_0)x_3$$

where m_i is the ith minterm of $\{s_1, s_0\}$.



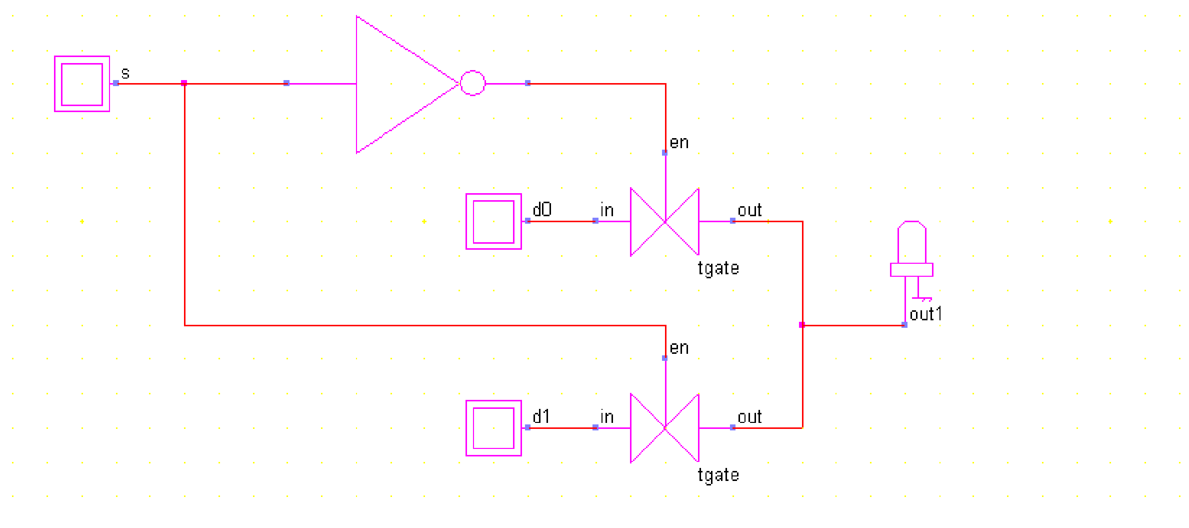
4x1 MUX

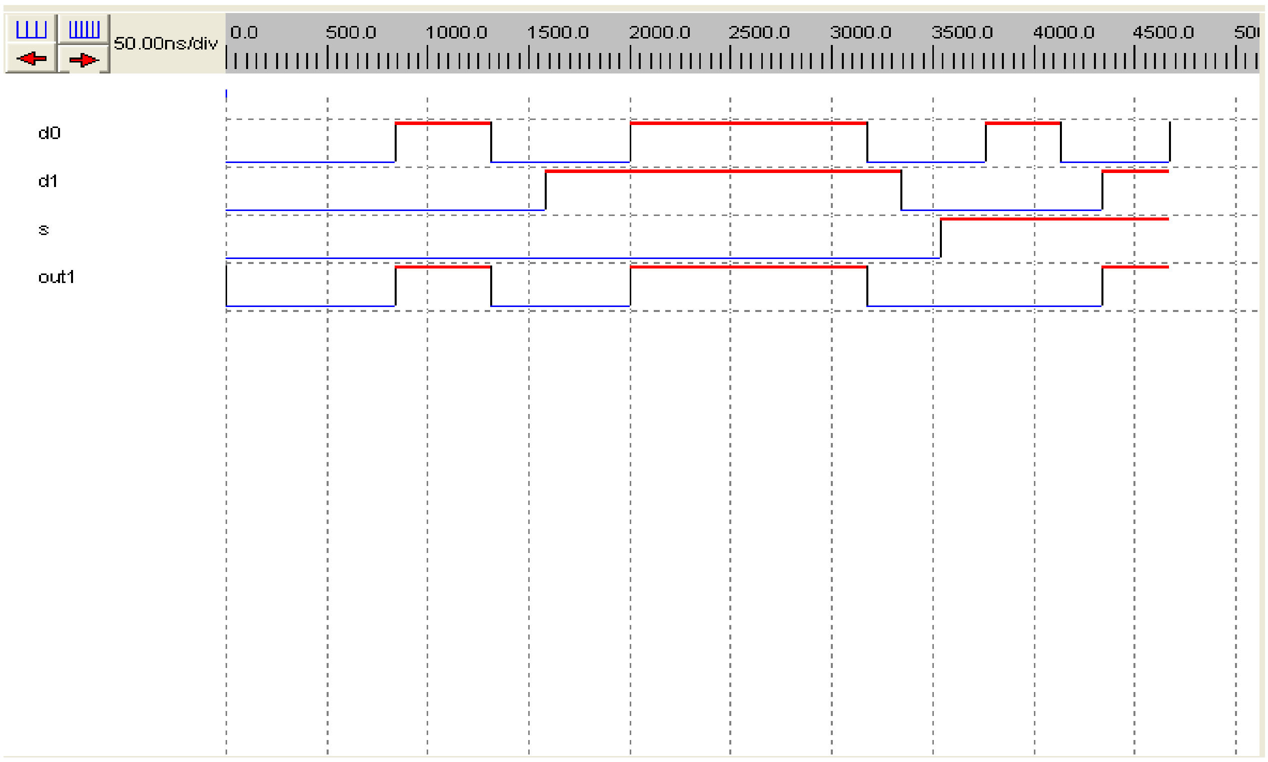
This approach can be generalized to any MUX of 2^n inputs with n selections.

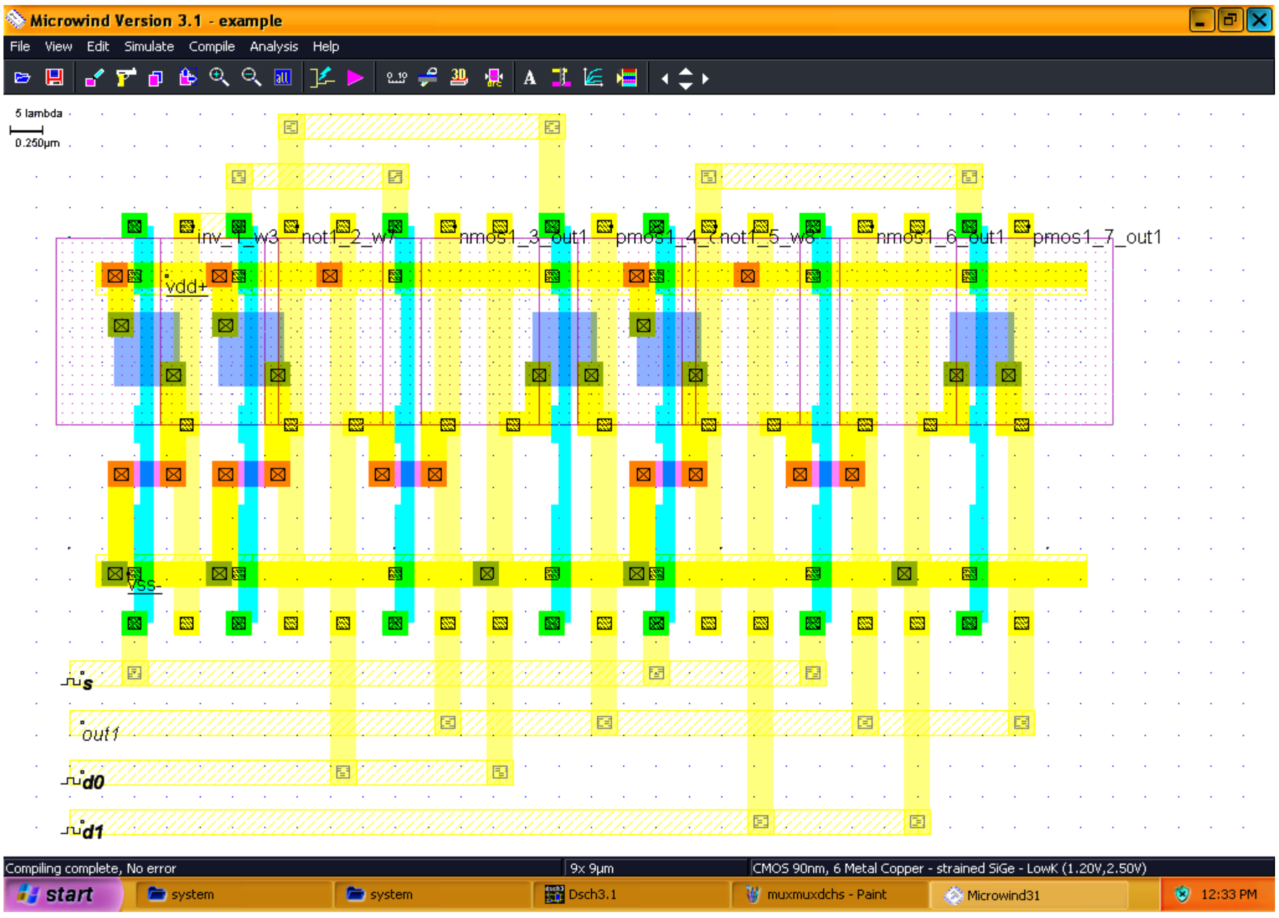
CONCLUSION:

We have designed 2:1 Mux by conventional method and by using Transmission gates and compared them, prepared layout in multi metal layers and simulated with micro wind software also learnt the concepts of multiplexes along with its truth tables.

MUX 2:1


s
en
d0
in
out
tgate
out1
en
d1
in
out
tgate

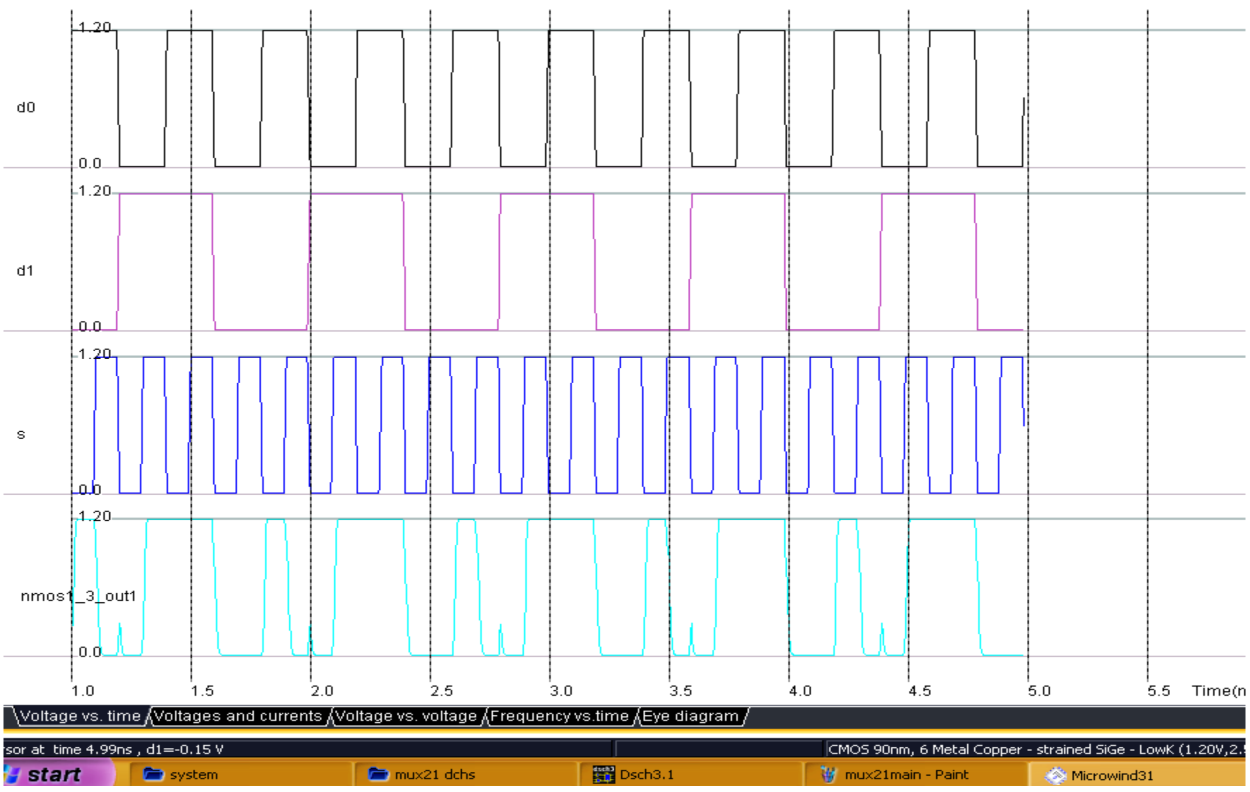

50.00ns/div
0.0
500.0
1000.0
1500.0
2000.0
2500.0
3000.0
3500.0
4000.0
4500.0
d0
d1
s
out1



Compiling complete, No error

9x 9µm

CMOS 90nm, 6 Metal Copper - strained SiGe - LowK (1.20V,2.50V)



sor at time 4.99ns , d1=-0.15 V

CMOS 90nm, 6 Metal Copper - strained SiGe - LowK (1.20V,2.

TITLE: Single bit SRAM cell	
NAME:	**SUBJECT: VLSI DESIGN AND TECHNOLOGY**
CLASS: BE	**ROLL NO:**
SEMESTER/YEAR: - 7TH SEM	**EXAM NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED: -	**REMARKS:**

EXPERIMENT NO – 8

AIM: Single bit SRAM cell.

OBJECTIVES:

1. Designing of SRAM memory using CMOS logic.

PRE-LAB REQUISITES:

1. Working of SRAM memory using PMOS & NMOS.
2. Output SRAM memory.

TOOLS USED:

1. PC
2. Micro wind backend tool.

THEORY:

The read-write (R/W) memory circuits are designed to permit the writing and reading of data bits to be stored in the memory array. The memory circuit is said to be static if the stored data can be retained as long as a sufficient power supply voltage is provided without any need for a periodic refresh operation. The data storage cell or another word the 1-bit memory cell in static Random-access memory consists of a simple latch circuit with two stable operating states. Depending on the conserved state of the two-inverter latch circuit, the data being held in the memory cell will be interpreted either as logic "0" or as logic "1." To access the data contained in the memory cell via the bit line, we need at least one switch, which is controlled by the corresponding word line, two complementary access switches consist of nMOS pass transistors are implemented to connect the 1-bit SRAM cell. The five -transistor SRAM cell can be implemented by using a polysilicon and n+ diffusion layer and p+ diffusion layer and metal layer. The two stable operating points of this basic latch circuit are used to store a one bit piece of information. The pair of cross-coupled inverters make up the central component of the SRAM cell to perform read and write operations. We use one nMOS pass transistors which driven the by the word line signal. The memory cell by raising its word line voltage to logic "1," hence, the pass transistors M3 is turned on.

Once the memory cell is selected, four basic operations may be performed on this cell. Figure 1 shows the schematic of 5T SRAM in which used a three NMOS and two PMOS transistor. Which transistor is connected to word line are called access transistor. With the help of these transistor controlling the bistable circuitry of SRAM. When bit line is activated and word line is low, no operation is performed by the circuit. When raising the word line of 5T SRAM the read and write operation is performed. Bit line basically an information.

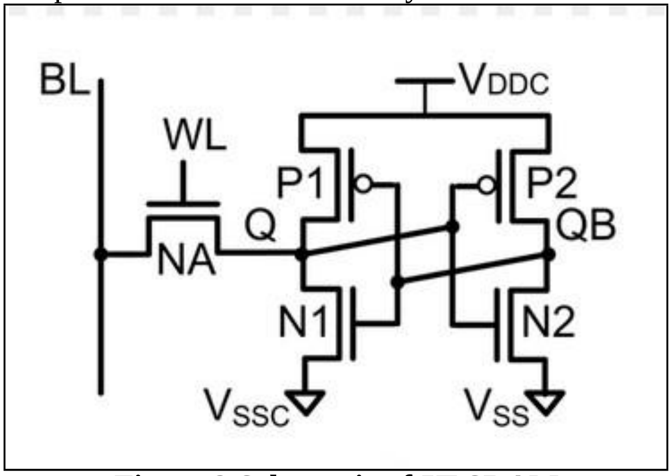


Figure 2 Schematic of 5T SRAM

PROCEDURE:

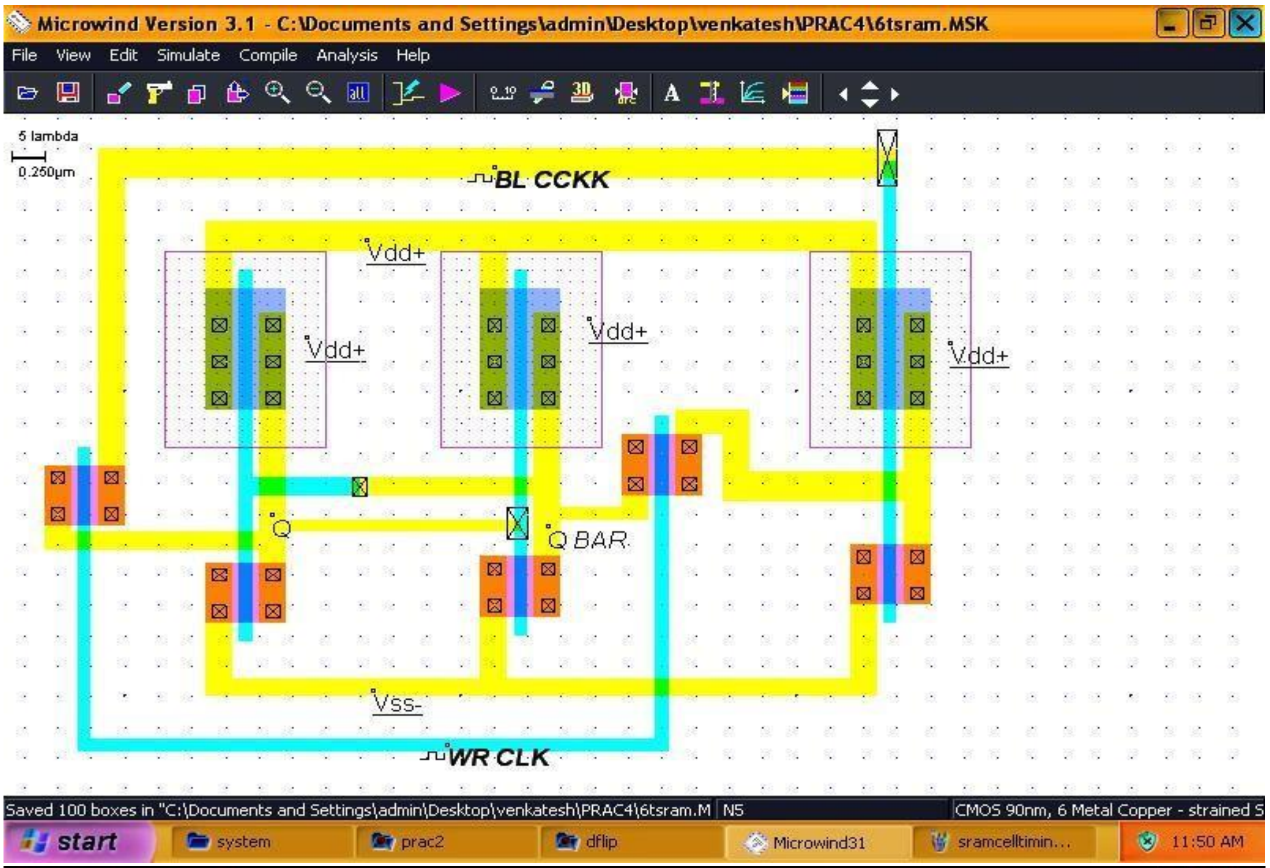
1. Construct SRAM memory using CMOS logic in Micro wind software.
2. Simulate this.
3. Check Output

OBSERVATIONS:

In simulation we verify the output of SRAM.

CONCLUSIONS:

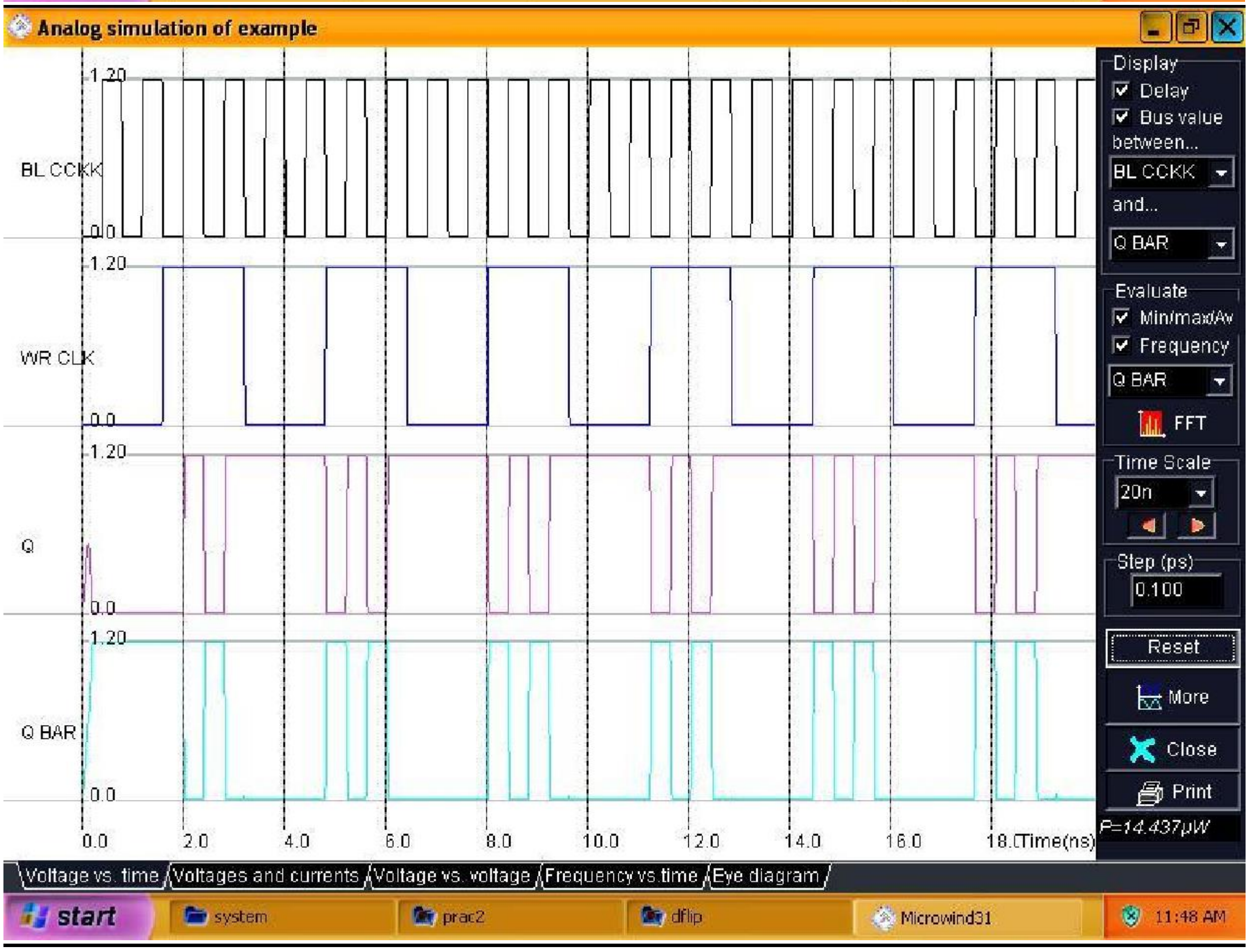
Thus we have designed SRAM memory by using CMOS logic and simulated it using microwind software and also learnt about construction and working of SRAM along with the schematic diagram of 6T SRAM cell.


Microwind Version 3.1 - C:\Documents and Settings\admin\Desktop\venkatesh\PRAC4\6tsram.MSK

File View Edit Simulate Compile Analysis Help
5 lambda
0.250µm
BL CCKK
Vdd+
Vdd+
Vdd+
Vdd+
Q
Q BAR
Vss-
WR CLK

Saved 100 boxes in "C:\Documents and Settings\admin\Desktop\venkatesh\PRAC4\6tsram.M N5 CMOS 90nm, 6 Metal Copper - strained S

start system prac2 dflip Microwind31 sramcelltimin... 11:50 AM


Analog simulation of example

TITLE: To design D- Flip Flop by using transmission gates and Simulate using micro wind

NAME:	**SUBJECT: VLSI DESIGN AND TECHNOLOGY**
CLASS: BE	**ROLL NO:**
SEMESTER/YEAR: - 7TH SEM	**EXAM NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED: -	**REMARKS:**

EXPERIMENT NO – 9 (Contents beyond Syllabus)

AIM: To design D- Flip Flop by using transmission gates and Simulate using micriwind.

OBJECTIVES:

1. Designing of D-Flip Flop using transmission gates.

PRE-LAB REQUISITES:

1. Concept of transmission gate
2. Comparison of transmission gate with CMOS.
3. Designing of digital logic using transmission gates.

TOOLS USED:

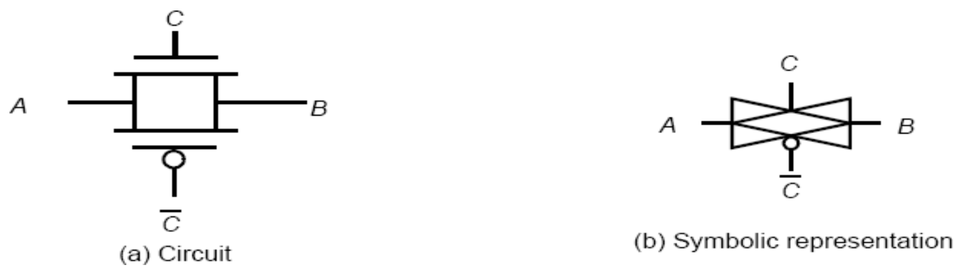
1. PC
2. Micro wind backend tool.

THEORY:

The most widely used solution to deal with the voltage drops induced by pass transistorsis the use of *transmission gates*. The primary limitation of NMOS or PMOS only pass gate is the threshold drop (NMOS pass device pass a strong 0 while passing a weak 1and PMOS pass devices pass a strong 1 while passing a weak 0). The ideal approach is to use the NMOS device to pull-down and the PMOS device to pull-up. The transmission gate combines the best of both device by placing a NMOS device in parallel with a PMOS device. The control signals to the transmission gate (*C* and *C*) are complementary. The transmission gate acts as a bidirectional switch controlled by the gate signal *C*. When *C* = 1, both MOSFETs are on, allowing the signal to pass through the gate. In short,

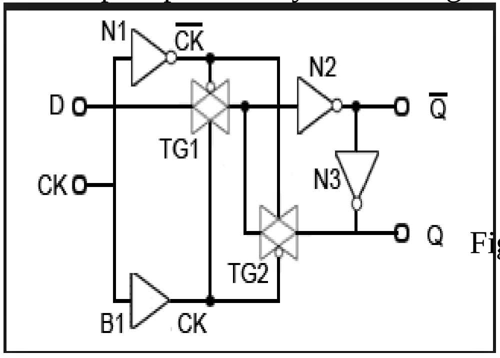
A=B if C=1

On the other hand, *C* = 0 places both transistors in cut-off, creating an open circuit between node A and



B

Following figure shows a basic circuit for a single flip-flop, which operates as a level triggered D Type flip-flop. Apart from the NOT gate (N1) and the buffer (B1) controlling the CK input, the basic flip-flop uses only two NOT gates (N2 and N3) and two transmission gates (TG1 and TG2).



OR

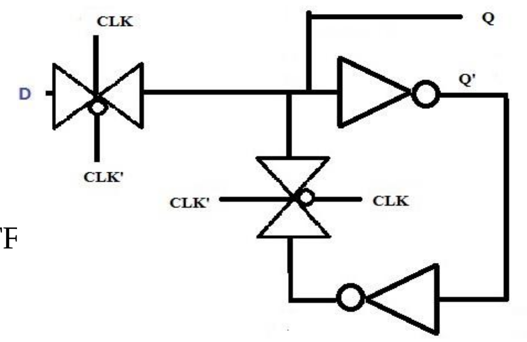


Figure 1: D-FF

The inverter N1 and the Buffer B1 create clock pulses CK and inverted clock pulses CK, which (because N1 and B1 have identical propagation delays), will exactly coincide in time when applied to the transmission gates of the flip-flop circuit. Initially, assuming that the CK and D are both at logic 0, CK will be at logic 1, so transmission gate TG1 will be in its high impedance state, preventing D from having any effect upon the flip-flop.

When CK is logic 1 and CK is logic 0, TG1 will conduct and the logic 0 from D will be inverted by N2, so the output Q will become logic 1. The logic 1 at Q will be inverted by N3 to become logic 0 at the Q output.

The logic 1 at Q will not affect the logic 0 at the input to N2 as TG2, connected in opposite polarity to the CK and CK clock signals will be turned off. This condition will remain stable irrespective of any further clock pulses being applied, as whenever TG1 is turned on, TG2 is turned off.

If input D is now changed to logic 1 between the occurrence of clock pulses, the rising edge of the first clock pulse after the change at D will turn on TG1, transmitting the logic 1 from D to the input of N2, causing Q to change to logic 1 and (via N3) Q to change to logic 0.

Whilst the CK input is high, any changes at D will be transmitted via TG1 and N2 to the outputs, indicating that the flip-flop is level triggered, but the moment the falling edge of the clock pulse occurs, TG1 will turn off and TG2 will turn on, isolating N1 and N2 from any further changes at the D input and leaving the output of N3 connected via TG2 to the input of N1.

As both these points will be at the same logic state (the logic state existing at D before the falling edge of the CK pulse) the flip-flop outputs will remain in a stable mode until the next clock pulse, when Q will take up the same state as input D once more.

PROCEDURE:

1. Construct D Flip Flop using transmission gate in Micro wind software.
2. Simulate this.
3. Check Output

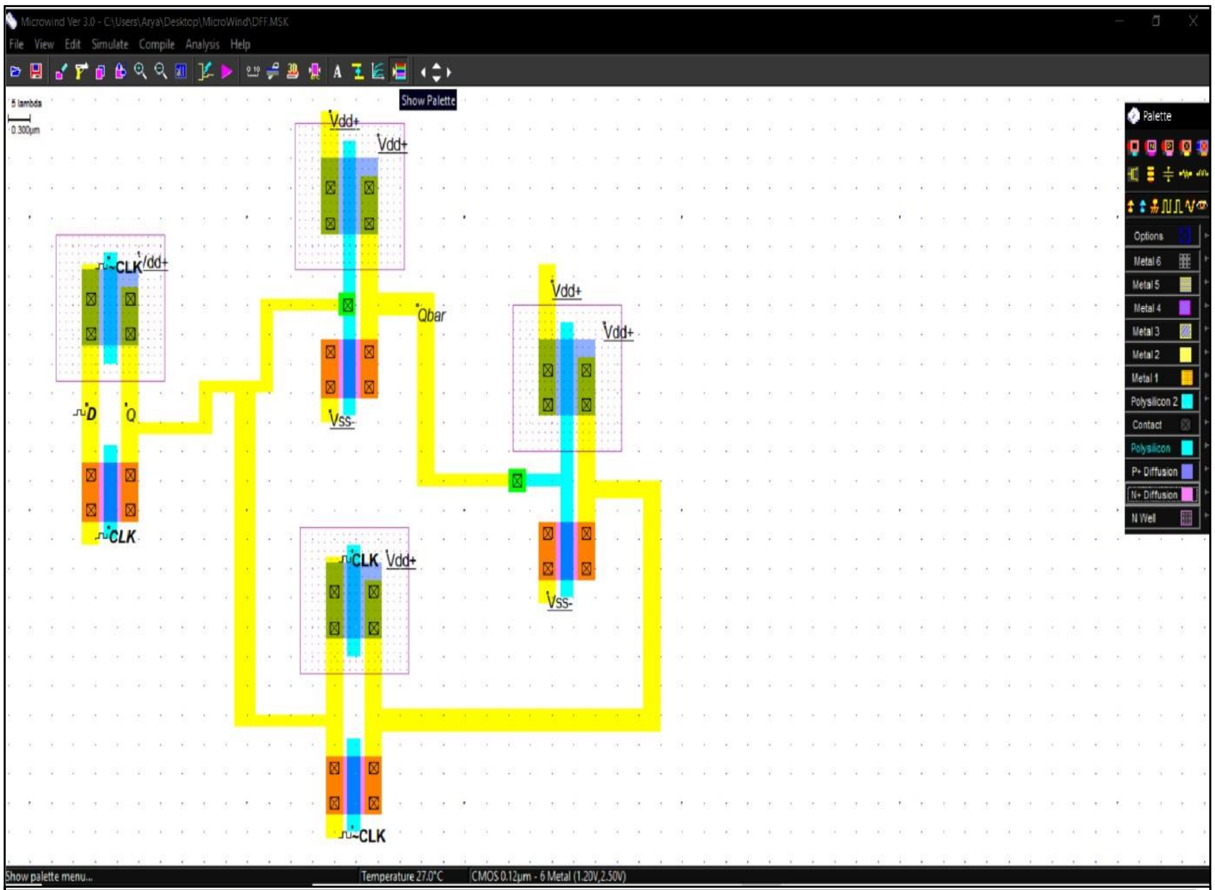
OBSERVATIONS:

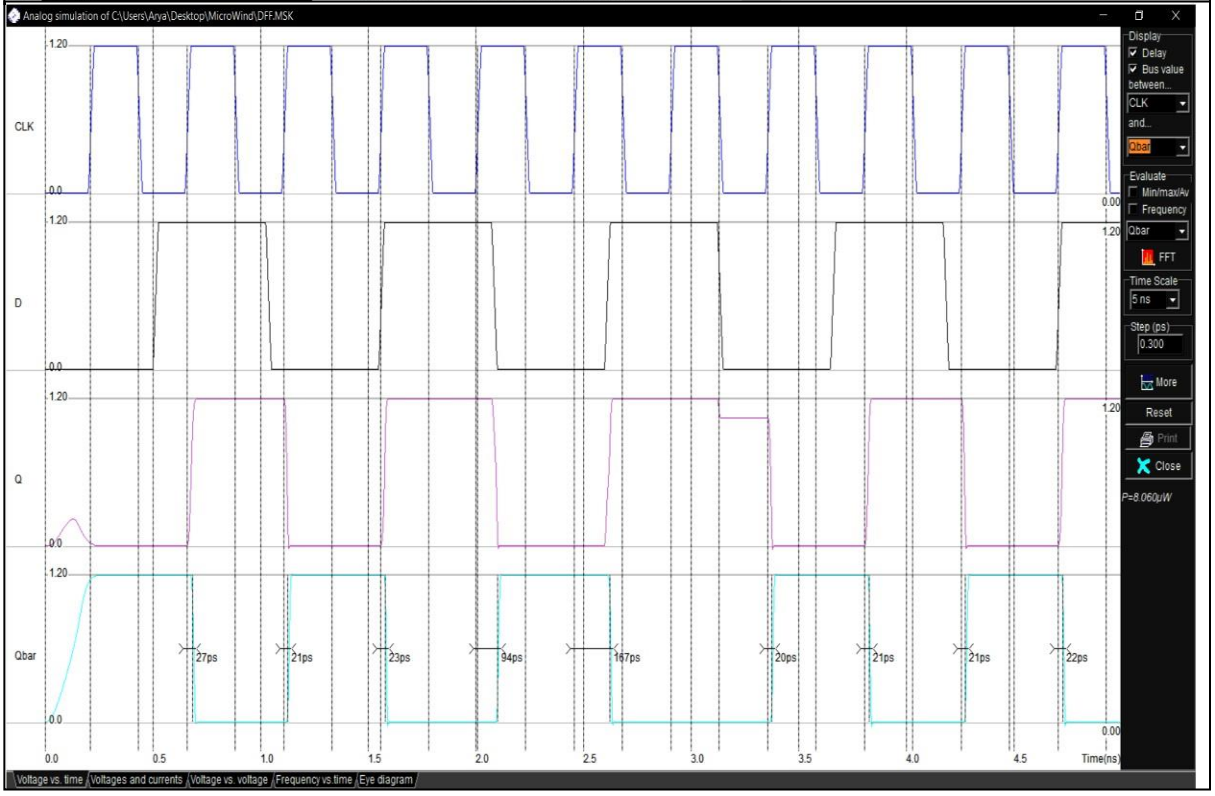
In simulation we verify the truth table of Half Adder.

CONCLUSIONS:

Thus we have studied how to design D- Flip Flop by using transmission gates and simulated it using micro wind software and studied its working with its truth table and also learned about the transmission gates.

D FLIP FLOP:


Microwind Ver 3.0 - C:\Users\Arya\Desktop\MicroWind\DFF.MSK
File View Edit Simulate Compile Analysis Help
Show Palette
5 lambdas
0.300µm
Vdd+
CLK
D
Q
Qbar
Vss-
~CLK
Palette
Options
Metal 6
Metal 5
Metal 4
Metal 3
Metal 2
Metal 1
Polysilicon 2
Contact
Polysilicon
P+ Diffusion
N+ Diffusion
N Well
Show palette menu...
Temperature 27.0°C
CMOS 0.12µm - 6 Metal (1.20V,2.50V)


Analog simulation of C:\Users\Arya\Desktop\MicroWind\DFF.MSK
CLK
D
Q
Qbar
1.20
0.0
27ps
21ps
23ps
94ps
167ps
20ps
21ps
21ps
22ps
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
Time(ns)
Display
Delay
Bus value
between...
CLK
and...
Qbar
Evaluate
Min/max/Av
Frequency
Qbar
FFT
Time Scale
5 ns
Step (ps)
0.300
More
Reset
Print
Close
P=8.060µW
Voltage vs. time
Voltages and currents
Voltage vs. voltage
Frequency vs.time
Eye diagram