Techno India NJR Institute of Technology

Department of Electronics & Communication Engineering

B.Tech. VII Semester

Lab: VLSI Design Lab (7EC4-21)

7EC4-21: VLSI Design Lab

Credit: **Max. Marks: 100(IA:60, ETE:40)**

0L+0T+4P

SN	Contents
1	Introduction: Objective, scope and outcome of the course.
PART-A	Step1 Write the VHDL/Verilog code using VHDL software for following experiment and simulate them. Step 2. Burn the Written code in Xilling Board and test the output with real input signal
1	Design and simulate all the logic gates with 2 inputs using VHDL/Verilog.
2	Design and simulate 2-to-4 decoder,3-to-8 encoder and 8X1 multiplexer using VHDL/Verilog.
3	Design and simulate half adder and full adder using VHDL (data flow method)/Verilog.
4	Design and simulate D, T and J-K flip flop using VHDL/Verilog.
5	Design a 4bit binary Asynchronous and synchronous counter. Obtain its number of gates, area, and speed and power dissipation.
6	Design a 4- bit Serial in-serial out shift register. Obtain its number of gates, area, and speed and power dissipation.
PART-B	Step-1 Design and simulate following experiment using ECAD software Viz. Mentor graphics, Orcade Pspice, Cadence etc. Step-2 Draw the layout (without any DRC error)of the schematic obtain in step 1 and obtain post layout simulation using appropriate ECAD software.
1	Design and simulate all the logic gates (NOT, NAND and NOR) with 2 inputs in CMOS Technology.
2	Design and simulate Y = AB (C+D), Y = A+B(C+D) and 4X1 multiplexer using CMOS Technology.
3	Design and simulate half adder and full adder using CMOS Technology.
4	Design and simulate SR flip flop using CMOS Technology.
5	Design and Simulate any DRAM cell.

Course Outcomes:

Course Code	Course Name	Course Outcomes	Details
7EC4-21	**VLSI Design Lab**	**CO1**	Understand the physical design process of Digital Integrated Circuits.
		CO2	Describe procedure for designing of programmable circuits.
		CO3	Demonstrate the ability to use various EDA tools for digital system design
		CO4	Implement various combinational and sequential circuits using VHDL on FPGA.
		CO5	Implement schematic and layout of various digital CMOS logic circuits using EDA tools.

Course Outcome Mapping with Program Outcome:

Course Outcome	**Program Outcomes (PO's)**											
CO. NO.	**Domain Specific**					**Domain Independent**						
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
CO1	2	2	1	2	2	1				2	1	2
CO2	2	2	2	2	2	1				2	2	1
CO3	2	3	2	2	2	2				2	2	2
CO4	2	2	2	2	2	1				2	2	1
CO5	2	2	2	2	2	2				2	1	2
1: Slight (Low) , 2: Moderate (Medium), 3: Substantial (High)												

ISE Quick Start Tutorial

Getting Started

Starting the ISE Software

For Windows users, start ISE from the Start menu by selecting:

Start _ Programs _ Xilinx ISE 7 _ Project Navigator

The ISE Project Navigator opens. The Project Navigator lets you manage the sources and processes in your ISE project. All of the tasks in the Quick Start Tutorial are managed from within Project Navigator.

Stopping and Restarting a Session

At any point during this tutorial you can stop your session and continue at a later time.
To stop the session:

- Save all source files you have opened in other applications.
- Exit the software (ISE and other applications).

The current status of the ISE project is maintained when exiting the software.
To restart your session, start the ISE software again. ISE displays the contents and state of your project with the last saved changes.

Accessing Help

At any time during the tutorial, you can access online help for additional information about a variety of topics and procedures in the ISE software as well as related tools.
To open Help you may 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.

Creating a New Project in ISE

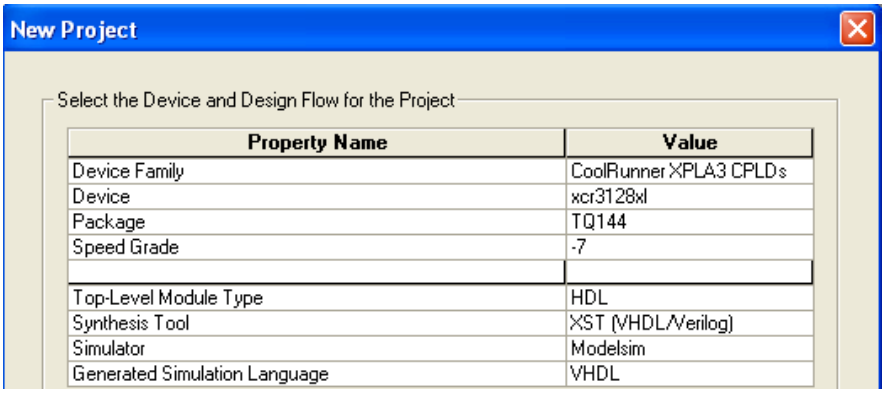
In this section, you will create a new ISE project. A project is a collection of all files necessary to create and to download a design to a selected FPGA or CPLD device.

To create a new project for this tutorial:

1. Select **File > New Project**. The New Project Wizard appears.
2. First, enter a location (directory path) for the new project.
3. Type **tutorial** in the Project Name field. When you type **tutorial** in the Project Name field, a tutorial subdirectory is created automatically in the directory path you selected.
4. Select **HDL** from the Top-Level Module Type list, indicating that the top-level file in your project will be HDL, rather than Schematic or EDIF.
5. Click **Next** to move to the project properties page.
6. Fill in the properties in the table as shown below

   Device Family: **CoolRunner XPLA3 CPLDs**
   Device: **xcr3128xl**
   Package: **TQ144**
   Speed Grade: **-7**
   Top-Level Module Type: **HDL**
   Synthesis Tool: **XST (VHDL/Verilog)**
   Simulator: **ModelSim**
   Generated Simulation Language: **VHDL** or **Verilog**, depending on the language you want to use when running behavioral simulation.

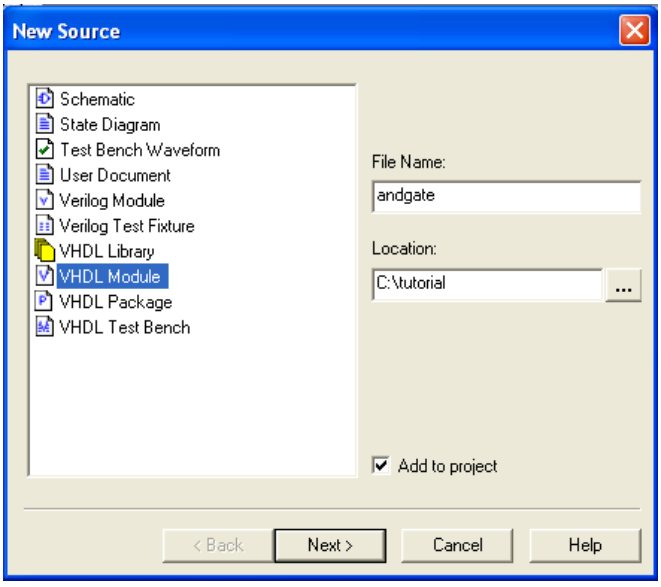
When the table is complete, your project properties should look like the following:



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 created.

Creating an HDL Source

In this section, you will create a 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.

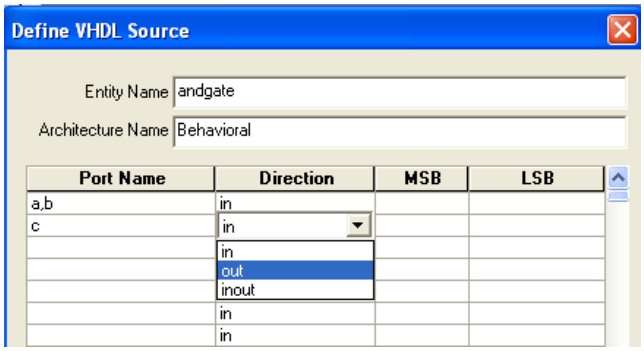


This simple AND Gate design has two inputs: A and B. This design has one output called C

1. Click **New Source** in the New Project Wizard to add one new source to your project.
2. Select **VHDL Module** as the source type in the New Source dialog box.
3. Type in the file name **andgate**.
4. Verify that the **Add to project** checkbox is selected.
5. Click **Next**.
6. Define the ports for your VHDL source.

   In the Port Name column, type the port names on three separate rows: **A**, **B** and **C**.

   In the Direction column, indicate whether each port is an input, output, or inout. For A and B, select **in** from the list. For C, select **out** from the list.



7. Click **Next** in the Define VHDL Source dialog box.
8. Click **Finish** in the New Source Information dialog box to complete the new source file template.
9. Click **Next** in the New Project Wizard.
10. Click **Next** again.
11. Click **Finish** in the New Project Information dialog box.

    ISE creates and displays the new project in the Sources in Project window and adds the andgate.vhd file to the project.
12. Double-click on the **andgate.vhd** file in the Sources in Project window to open the VHDL file in the ISE Text Editor.

    The andgate.vhd file contains:

    - Header information.
    - Library declaration and use statements.
    - Entity declaration for the counter and an empty architecture statement.
13. In the header section, fill in the following fields:

    Design Name: **andgate.vhd**

    Project Name: **andgate**

    Target Device: **xcr3128xl- TQ144**

    Description: **This is the top level HDL file for an up/down counter.**

    Dependencies: **None**

    Note: It is good design practice to fill in the header section in all source files.
14. Below the `end process` statement, enter the following line:

```
C <= A and B;
```

15. Save the file by selecting **File > Save**.

Checking the Syntax of the New Counter Module

When the source files are complete, the next step is to check the syntax of the design. Syntax errors and typos can be found using this step.

1. Select the **counter** design source in the ISE Sources window to display the related processes in the Processes for Source window.
2. Click the "**+**" next to the Synthesize-XST process to expand the hierarchy.
3. Double-click the **Check Syntax** process.

When an ISE process completes, you will see a status indicator next to the process name.

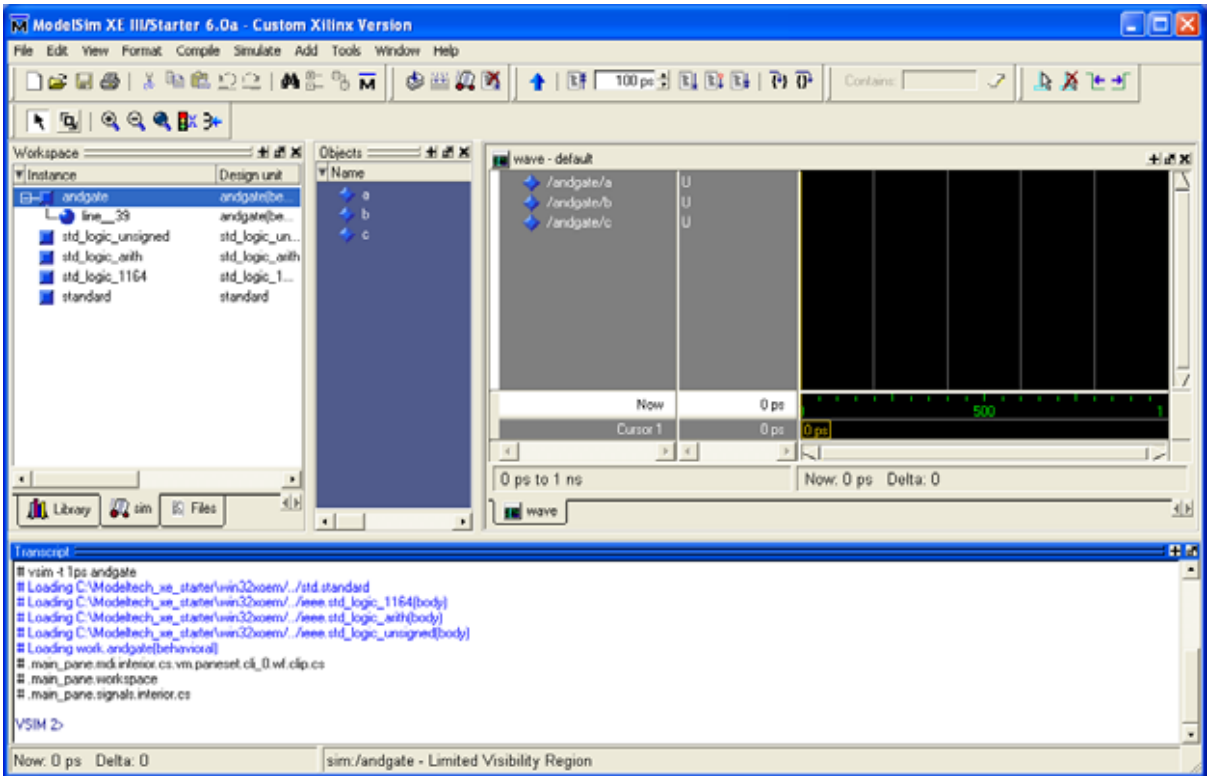
- If the process completed successfully, a green check mark appears.
- If there were errors and the process failed, a red X appears.
- A yellow exclamation point means that the process completed successfully, but some warnings occurred.
- An orange question mark means the process is out of date and should be run again.

4. Look in the **Console** tab of the Transcript window and read the output and status messages produced by any process that you run.

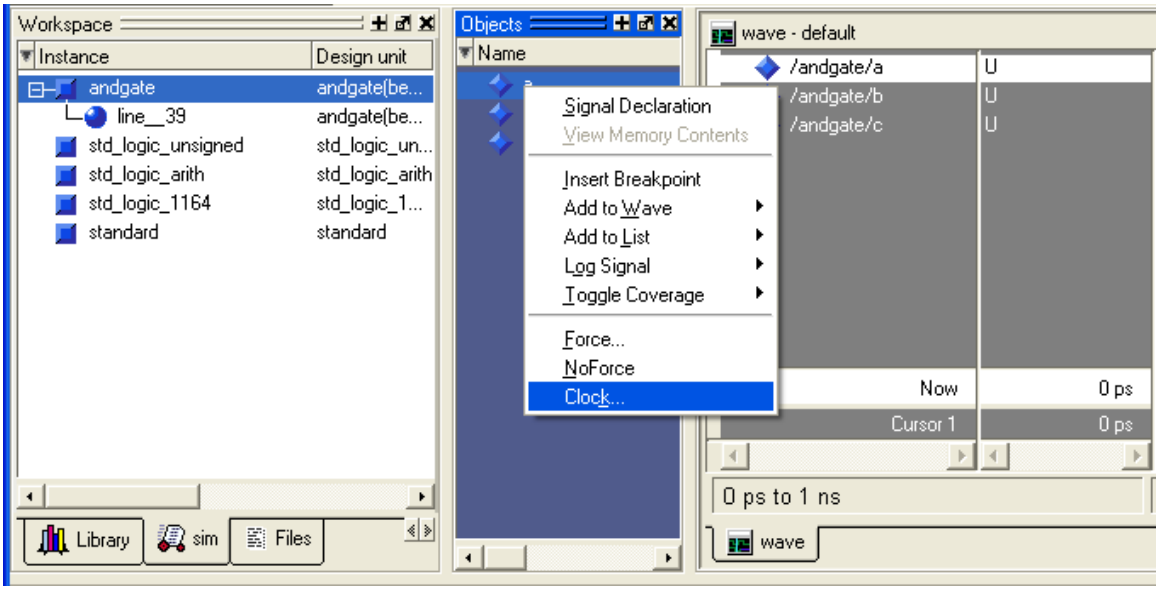
   Caution! You must correct any errors found in your source files. If you continue without valid syntax, you will not be able to simulate or synthesize your design.

Simulation

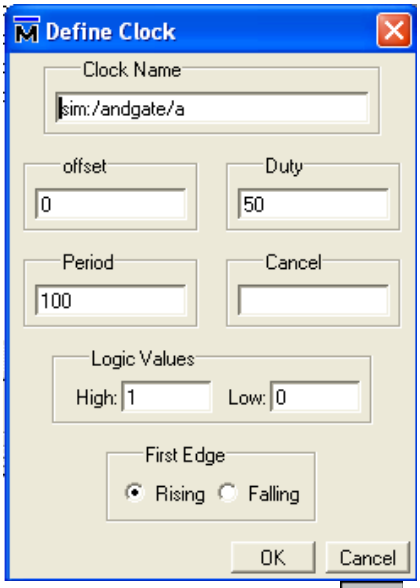
1. Double click Launch ModelSim Simulator in the Process View window.



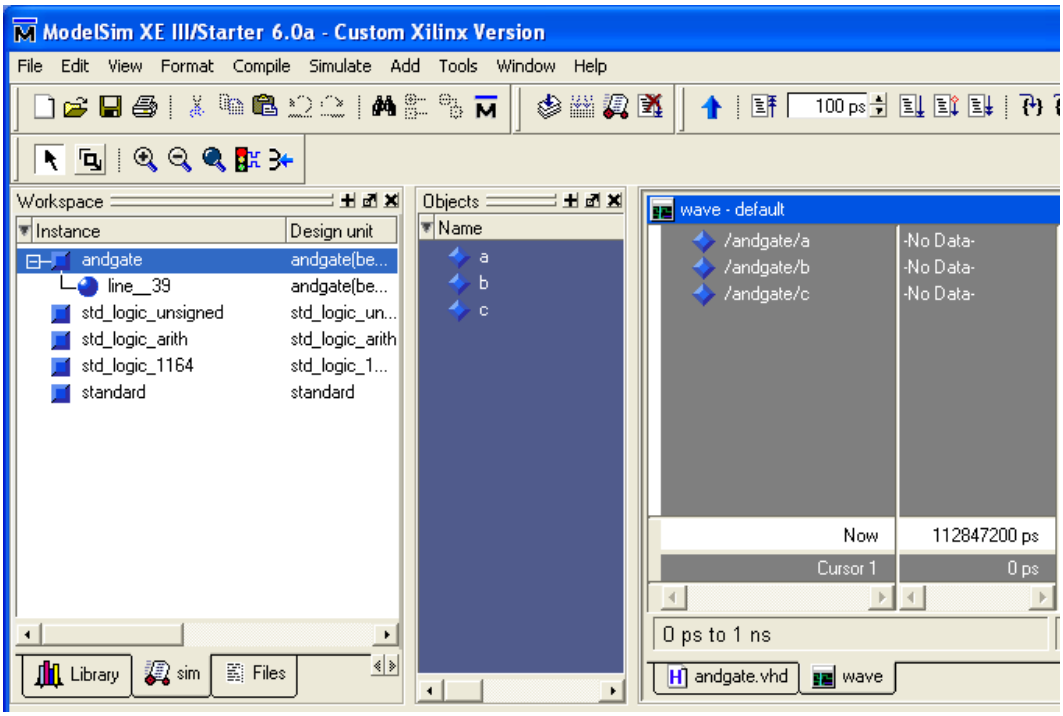
2. Right Click '*a*' to open a context menu.
3. Select Force or Clock to add the signal.



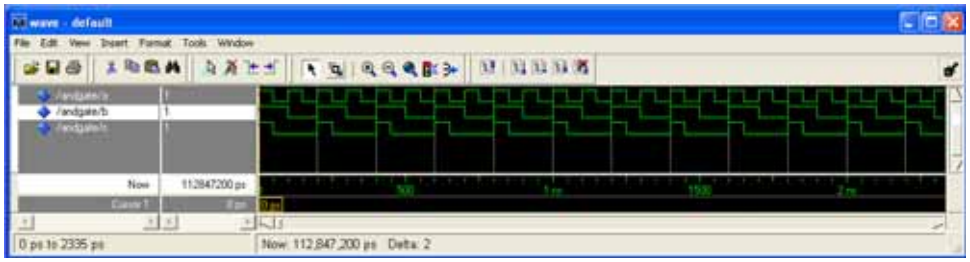
4. Define the Clock or Force signal to load appropriate signal



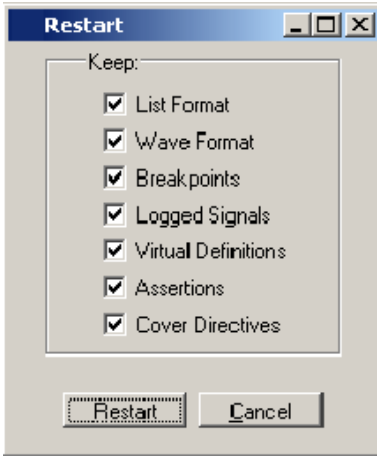
5. Run the simulation by clicking the Run icon in the Main or Wave window toolbar. 



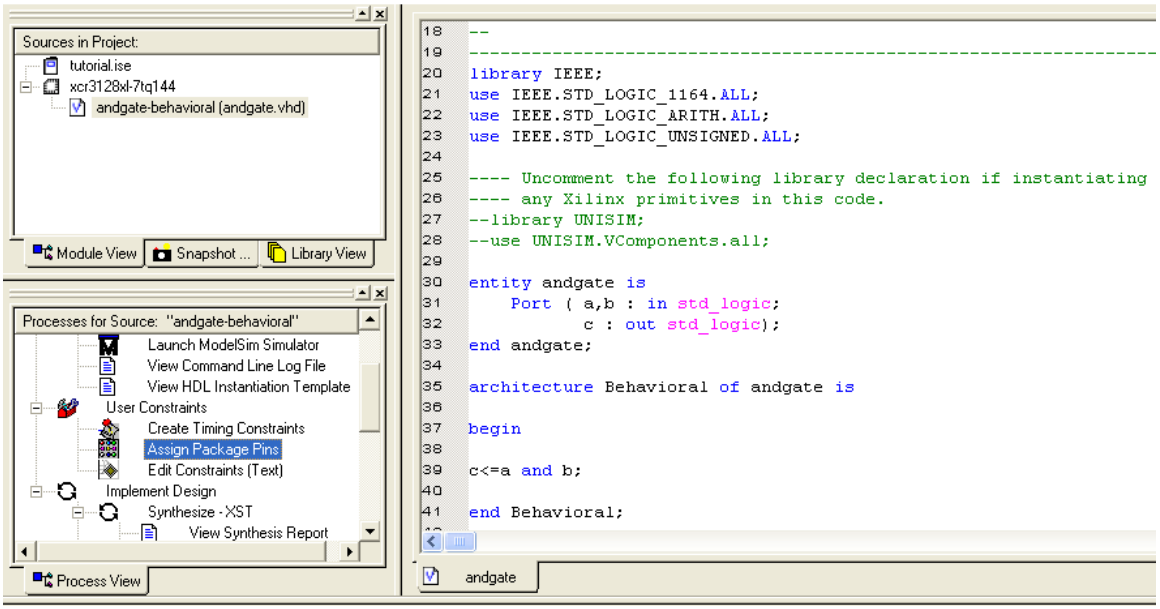
6. Waveform can be observed in the wave window



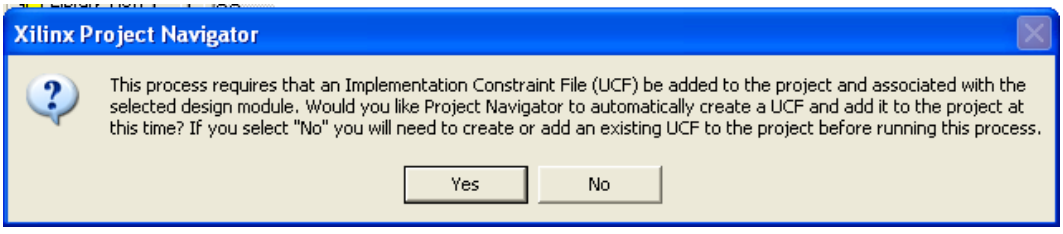
7. Click the Run -All icon on the Main or Wave window toolbar. The simulation continues running until you execute a break command. 
8. Click the Break icon. The simulation stops running. 
9. To restart the simulation, click the Restart icon to reload the design elements and reset the simulation time to zero. The Restart dialog that appears gives you options on what to retain during the restart.  Click the Restart button in the Restart dialog.



Assigning Pin Location



1. Double-click the **Assign Package Pins** process found in the User Constraints process group. ISE runs the Synthesis and Translate steps and automatically creates a User Constraints File (UCF). You will be prompted with the following message:



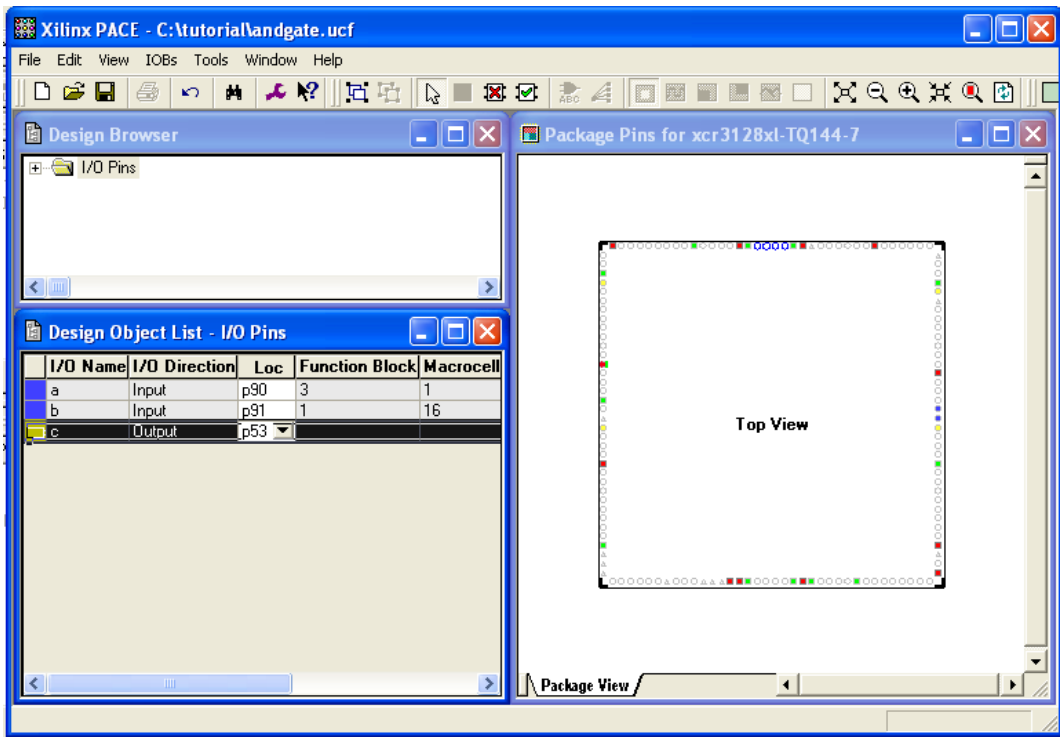
2. Click **Yes** to add the UCF file to your project. The `counter.ucf` file is added to your project and is visible in the Sources in Project window. The Xilinx Constraints Editor opens automatically.
3. Now the Xilinx Pinout and Area Constraints Editor (PACE) opens.
4. You can see your I/O Pins listed in the Design Object List window. Enter a pin location for each pin in the Loc column as specified below:

   A: p90

   B: p91

   C: p53

5. Click on the **Package View** tab at the bottom of the window to see the pins you just added. Put your mouse over grid number to verify the pin assignment.



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.

Creating Configuration Data

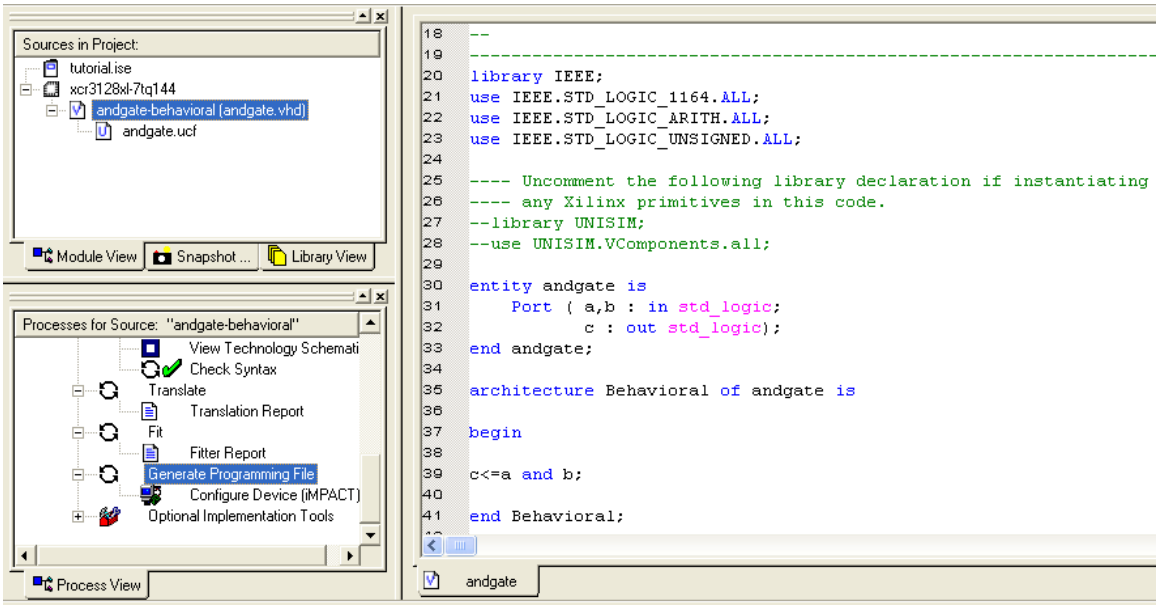
The final phase in the software flow is to generate a program file and configure the device.

Generating a Program File

The Program File is a encoded file that is the equivalent of the design in a form that can be downloaded into the CPLD device.

1. Double Click the **Generate Programming File** process located near the bottom of the Processes for Source window.

   The Program File is created. It is written into a file called `andgate.jed`. This is the actual configuration data.



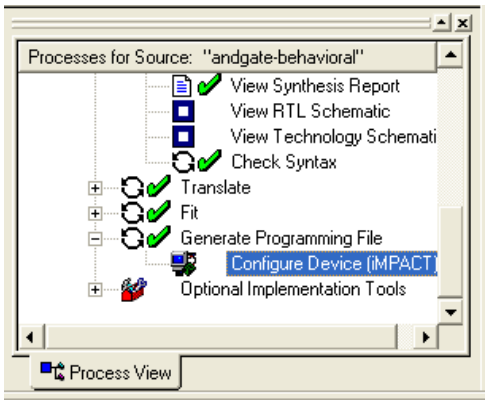
Configuring the Device

iMPACT is used to configure your FPGA or CPLD device. This is the last step in the design process. This section provides simple instructions for configuring a Spartan-3 xc3s200 device connected to your PC.

Note: Your board must be connected to your PC before proceeding. If the device on your board does not match the device assigned to the project, you will get errors. Please refer to the iMPACT Help for more information. To access the help, select **Help > Help Topics**.

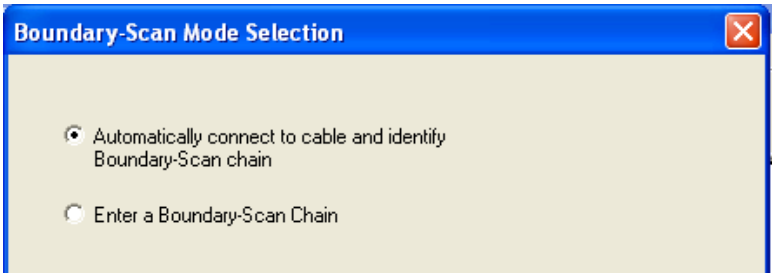
To configure the device:

1. Click the "**+**" sign to expand the **Generate Programming File** processes.



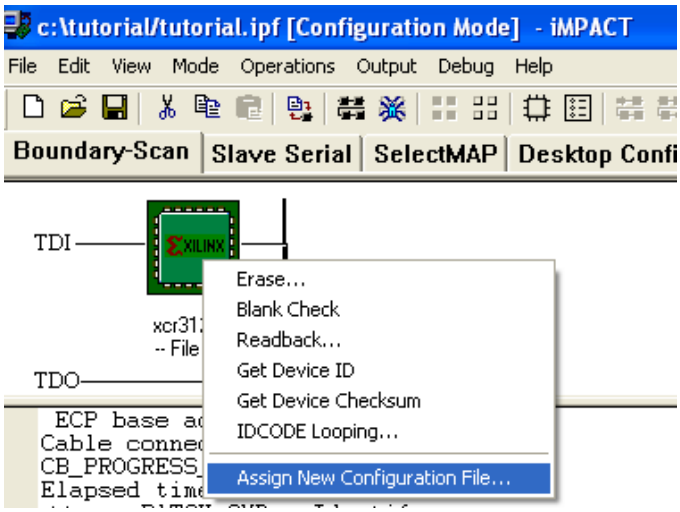
2. Double-click the **Configure Device (iMPACT)** process. iMPACT opens and the Configure Devices dialog box is displayed.

3. In the Configure Devices dialog box, verify that **Boundary-Scan Mode** is selected and click **Next**.
4. Verify that **Automatically connect to cable and identify Boundary-Scan chain** is selected and click **Finish**.



5. If you get a message saying that there was one device found, click **OK** to continue.

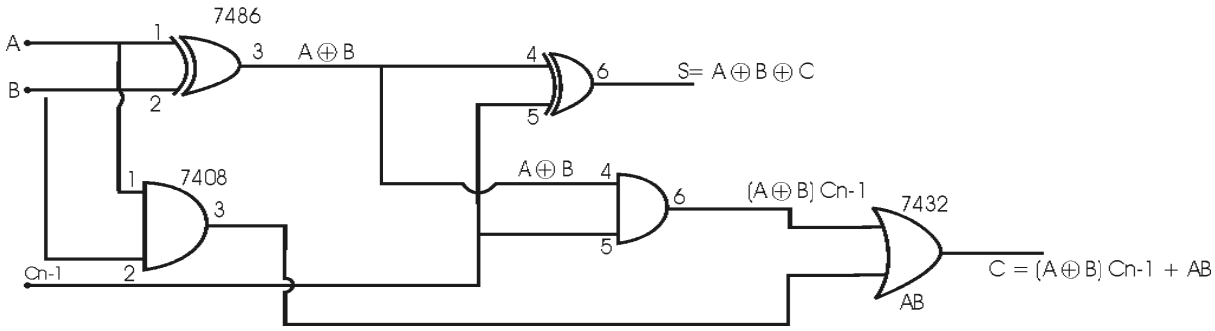
6. The iMPACT will now show the detected device, right click the device and select **New Configuration File.**



7. The **Assign New Configuration File** dialog box appears. Assign a configuration file to each device in the JTAG chain. Select the `andgate.jed` file and click **Open**.
8. Right-click on the counter device image, and select **Program...** to open the **Program Options** dialog box.
9. Click **OK** to program the device. ISE programs the device and displays Programming Succeeded if the operation was successful.
10. Close iMPACT without saving.

PROGRAMS

1. VHDL CODE FOR FULL ADDER DATA FLOW:



Full Adder				
A	B	Ci	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

EXPRESSIONS:

$S = A \oplus B \oplus Ci$

$CO = (A \oplus B)Ci + AB$

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity fa1 is
          Port ( a,b,ci : in  STD_LOGIC; s,co : out  STD_LOGIC);
        end fa1;
architecture Behavioral of fa1 is
begin
        s<=a xor b xor ci;
        co<=(a and b)or (b and ci)or (ci and a);
end Behavioral;
```

VHDL CODE FOR FULL ADDER BEHAVIORAL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity fa1 is
          Port ( a,b,ci : in  STD_LOGIC; s,co : out  STD_LOGIC);
        end fa1;
architecture Behavioral of fa1 is
begin
        process(a,b,ci)
begin
        s<=a xor b xor ci;
        co<=(a and b)or (b and ci)or (ci and a);
end process;
end Behavioral;
```

2. VHDL CODE FOR FULL ADDER STRUCTURAL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity fa1 is
          Port ( a,b,cin : in  STD_LOGIC;
               s,cout : out  STD_LOGIC);
        end fa1;
architecture struct of fa1 is
        component and21
        port(a,b:in std_logic;          ---components, entity and architecture
        c:out std_logic);               --- must be declared separately
        end component;
                component xor21
                port(a,b:in std_logic;  ---components, entity and architecture
                c:out std_logic);       --- must be declared separately
                end component;
        component or31
        port(a,b:in std_logic;          ---components, entity and architecture
        d:out std_logic);               --- must be declared separately
        end component;
signal s1,s2,s3:std_logic;
begin
u1:xor21 port map(a,b,s1);
u2:xor21 port map(s1,cin,s);
u3:and21 port map(a,b,s2);
u4:and21 port map(s1,cin,s3);
u6:or31 port map(s2,s3,cout);
end struct;
```

3. VHDL CODE FOR MULTIPLEXER (8:1):

INPUTS								SELECT LINES			O/P
d(7)	d(6)	d(5)	d(4)	d(3)	d(2)	d(1)	d(0)	s(2)	s(1)	s(0)	f
X	X	X	X	X	X	X	0	0	0	0	0
X	X	X	X	X	X	X	1	0	0	0	1
X	X	X	X	X	X	0	X	0	0	1	0
X	X	X	X	X	X	1	X	0	0	1	1
X	X	X	X	X	0	X	X	0	1	0	0
X	X	X	X	X	1	X	X	0	1	0	1
X	X	X	X	0	X	X	X	0	1	1	0
X	X	X	X	1	X	X	X	0	1	1	1
X	X	X	0	X	X	X	X	1	0	0	0
X	X	X	1	X	X	X	X	1	0	0	1
X	X	0	X	X	X	X	X	1	0	1	0
X	X	1	X	X	X	X	X	1	0	1	1
X	0	X	X	X	X	X	X	1	1	0	0
X	1	X	X	X	X	X	X	1	1	0	1
0	X	X	X	X	X	X	X	1	1	1	0
1	X	X	X	X	X	X	X	1	1	1	1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity mux1 is
           Port ( d : in  STD_LOGIC_VECTOR (7 downto 0);
                s : in  STD_LOGIC_VECTOR (2 downto 0);
                f : out  STD_LOGIC);
        end mux1;
architecture Behavioral of mux1 is
begin
        f<= d(0) when s="000" else
        d(1) when s="001" else
        d(2) when s="010" else
        d(3) when s="011" else
        d(4) when s="100" else
        d(5) when s="101" else
        d(6) when s="110" else
        d(7) when s="111";
end Behavioral;
```

4. VHDL CODE FOR Demultiplexer (1:8):

SELECT LINES			I/P	OUTPUT							
s(2)	s(1)	s(0)	f	d(7)	d(6)	d(5)	d(4)	d(3)	d(2)	d(1)	d(0)
0	0	0	0	X	X	X	X	X	X	X	0
0	0	0	1	X	X	X	X	X	X	X	1
0	0	1	0	X	X	X	X	X	X	0	X
0	0	1	1	X	X	X	X	X	X	1	X
0	1	0	0	X	X	X	X	X	0	X	X
0	1	0	1	X	X	X	X	X	1	X	X
0	1	1	0	X	X	X	X	0	X	X	X
0	1	1	1	X	X	X	X	1	X	X	X
1	0	0	0	X	X	X	0	X	X	X	X
1	0	0	1	X	X	X	1	X	X	X	X
1	0	1	0	X	X	0	X	X	X	X	X
1	0	1	1	X	X	1	X	X	X	X	X
1	1	0	0	X	0	X	X	X	X	X	X
1	1	0	1	X	1	X	X	X	X	X	X
1	1	1	0	0	X	X	X	X	X	X	X
1	1	1	1	1	X	X	X	X	X	X	X

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

        entity dmux is
        port(f:in std_logic;
        s:in std_logic_vector(2 downto 0);
        y:out std_logic_vector(7 downto 0));
        end demux;

architectural behavioral of dmux is
begin
        y(0)<=f when s="000"else'0';
        y(1)<=f when s="001"else'0';
        y(2)<=f when s="010"else'0';
        y(3)<=f when s="011"else'0';
        y(4)<=f when s="100"else'0';
        y(5)<=f when s="101"else'0';
        y(6)<=f when s="110"else'0';
        y(7)<=f when s="111"else'0';
end behavioral;
```

5. VHDL CODE FOR ENCODER WITHOUT PRIORITY (8:3):

SELECT LINES			OUTPUT							
s(2)	s(1)	s(0)	y(7)	y(6)	y(5)	y(4)	y(3)	Y(2)	y(1)	y(0)
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	X	X	X	X	X	X	X	X

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity enco is
           Port ( i : in  STD_LOGIC_VECTOR (7 downto 0);
                y : out  STD_LOGIC_VECTOR (2 downto 0));
        end enco;
architecture Behavioral of enco is
begin
        with i select
        y<="000" when "00000001",
        "001" when "00000010",
        "010" when "00000100",
        "011" when "00001000",
        "100" when "00010000",
        "101" when "00100000",
        "110" when "01000000",
        "111" when others;
end Behavioral;
```

6. VHDL CODE FOR ENCODER WITH PRIORITY (8:3):

SELECT LINES			OUTPUT							
s(2)	s(1)	s(0)	y(7)	y(6)	y(5)	y(4)	y(3)	y(2)	y(1)	y(0)
0	0	0	0	0	0	0	0	1	1	1
0	0	1	0	0	0	0	0	1	1	0
0	1	0	0	0	0	0	0	1	0	1
0	1	1	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	1	1	0
1	0	1	0-	0	0	0	0	0	1	0
1	1	0	0	0	0	0	0	1	0	0
1	1	1	X	X	X	X	X	X	X	X

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity enco1 is
           Port ( i : in  STD_LOGIC_VECTOR (7 downto 0);
                y : out  STD_LOGIC_VECTOR (2 downto 0));
        end enco1;
architecture Behavioral of enco1 is
begin
        with i select
        y<="000" when "00000111",
        "001" when "00000110",
        "010" when "00000101",
        "011" when "00000100",
        "100" when "00000011",
        "101" when "00000010",
        "110" when "00000001",
        "111" when others;
end Behavioral;
```

7. VHDL CODE FOR 3:8 DECODER:

SELECT LINES			OUTPUT							
s(2)	s(1)	s(0)	y(7)	y(6)	y(5)	y(4)	y(3)	y(2)	y(1)	y(0)
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0
X	X	X	0	0	0	0	0	0	0	0

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity dec1 is
           Port ( s : in  STD_LOGIC_VECTOR (2 downto 0);
                y : out  STD_LOGIC_VECTOR (7 downto 0));
        end dec1;
architecture Behavioral of dec1 is
begin
        with sel select
        y<="00000001" when "000",
        "00000010" when "001",
        "00000100" when "010",
        "00001000" when "011",
        "00010000" when "100",
        "00100000" when "101",
        "01000000" when "110",
        "10000000" when "111",
        "00000000" when others;
end Behavioral;
```

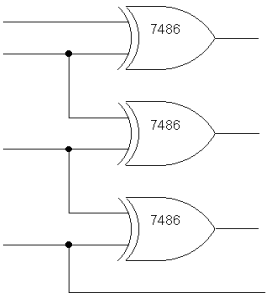
8. VHDL CODE FOR 2-bit comparator:

A1	A0	B1	B0	Y1 (A > B)	Y2 (A = B)	Y3 (A < B)
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity comp is
           Port ( a,b : in  STD_LOGIC_VECTOR (1 downto 0);
                y : out  STD_LOGIC_VECTOR (2 downto 0));
        end comp;
architecture Behavioral of comp is
begin
        y<= "100" when a>b else
        "001" when a<b else
        "010" when a=b;
end Behavioral;
```

9. VHDL CODE FOR Binary to gray (USING EXOR GATES):

CIRCUIT DIAGRAM:



Using EX-OR gates

EXPRESSIONS:

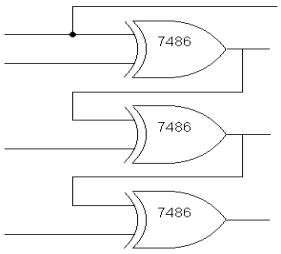
G(0)=B(0) ⊕ B(1)
G(1)=B(1) ⊕ B(2)
G(2)=B(2) ⊕ B(3)
G(3)=B(3)

Inputs				Outputs			
B (3)	B (2)	B (1)	B (0)	G (3)	G (2)	G (1)	G (0)
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity Binary_Gray is
        port(  b: in std_logic_vector(3 downto 0);          --Binary Input
             g: out std_logic_vector(3 downto 0));          --Gray Output
        end  binary_gray;
architecture behavioral of  Binary_gray is
begin
        b(3)<= g(3);
        b(2)<= g(3) xor g(2);
        b(1)<= g(2) xor g(1);
        b(0)<= g(1) xor g(0);
end behavioral;
```

10. VHDL CODE FOR GRAY TO BINARY:

CIRCUIT DIAGRAM:



Using EX-OR gates

EXPRESSIONS:

$B(0)=G(0) \oplus G(1) \oplus G(2) \oplus G(3)$
$B(1)=G(1) \oplus G(2) \oplus G(3)$
$B(2)=G(2) \oplus G(3)$
$B(1)=G(3)$

INPUT				OUTPUT			
G(3)	G(2)	G(1)	G(0)	B (3)	B (2)	B (1)	B (0)
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity gb1 is
           Port ( g : in  STD_LOGIC_VECTOR (3 downto 0);
                b : out  STD_LOGIC_VECTOR (3 downto 0));
        end gb1;
architecture Behavioral of gb1 is
begin
        b(3)<= g(3);
        b(2)<= g(3) xor g(2);
        b(1)<= g(3) xor g(2) xor g(1);
        b(0)<= g(3) xor g(2) xor g(1) xor g(0);
end behavioral;
```

11. VHDL CODE FOR JK FLIP FLOP WITH ASYNCHRONOUS RESET: (Master Slave JK Flip-Flop)

Reset	J	K	Clock	Qn+1	$\overline{Qn+1}$	Status
1	X	X	X	0	1	Reset
0	0	0	⎍	Qn	$\overline{Qn}$	No Change
0	0	1	⎍	0	1	Reset
0	1	0	⎍	1	0	Set
0	1	1	⎍	$\overline{Qn}$	Qn	Toggle

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity jkff1 is
          Port ( j,k,clk,reset : in  STD_LOGIC;
               Q : inout  STD_LOGIC);
        end jkff1;
architecture Behavioral of jkff1 is
        signal div:std_logic_vector(22 downto 0);
        signal clkd:std_logic;
begin
process(clk)
        begin
        if rising_edge(clk)then
        div<= div+1;
end if;
end process;
        clkd<=div(22);
process(clkd,reset)
begin
        if(reset='1')then
        Q<= '0';
        elsif(clkd'event and clkd='1')then
        if(j='0' and k='0')then
        Q<= Q;
        elsif(j='0' and k='1')then
        Q<= '0';
        elsif(j='1' and k='0')then
        Q<= '1';
        elsif(j='1' and k='1')then
        Q<= not Q;
end if;
end if;
end process;
end Behavioral;
```

12. VHDL CODE FOR T FLIP FLOP:

Clear	T	Clock	Qn+1	$\overline{Qn+1}$
1	0	0	0	$\overline{Qn}$
0	1	⎍	$\overline{Qn}$	Qn

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity tff is
           Port ( t,clk,rst : in  STD_LOGIC;
                q : inout  STD_LOGIC);
        end tff;
architecture Behavioral of tff is
        signal div:std_logic_vector(22 downto 0);
        signal clkd:std_logic;
begin
process(clk)
begin
        if rising_edge(clk)then
        div<= div+1;
end if;
end process;
        clkd<=div(20);
process(clkd,rst)
begin
        if(rst='1')then
        q<='0';
        elsif (clkd'event and clkd='1' and t='1') then
        q<= not q;
        else q<=q;
end if;
end process;
end Behavioral;
```

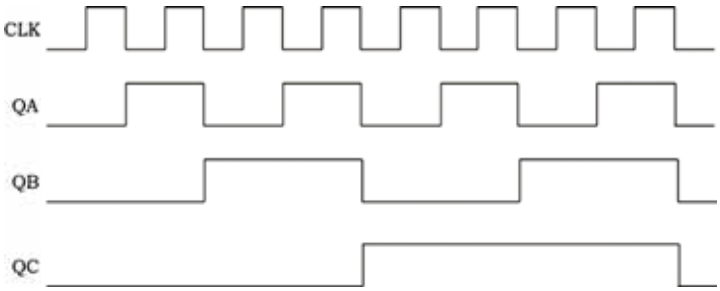
13. VHDL CODE FOR D FLIP FLOP:

Clear	D	Clock	Qn+1	$\overline{Qn+1}$
1	0	0	0	1
0	1	⎍	1	0

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity dff is
           Port ( d,res,clk : in  STD_LOGIC;
                q : out  STD_LOGIC);
        end dff;
architecture Behavioral of dff is
begin
process(clk)
begin
        if (res ='1')then q<='0';
        elsif clk'event and clk='1'
        then q<=d;
        end if;
end process;
end Behavioral;
```

14. ASYNCHRONOUS BINARY UP COUNTER:

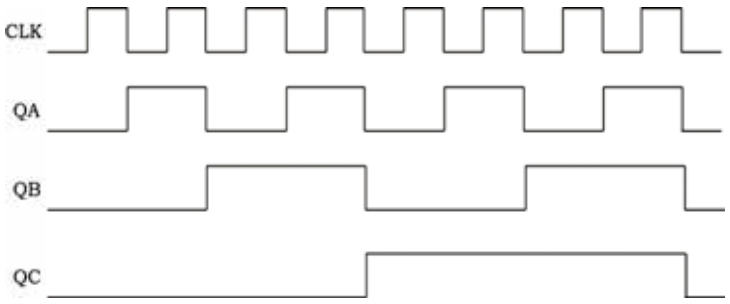
3-bit Asynchronous up counter			
Clock	QC	QB	QA
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0
9	0	0	1




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity asybin1 is
           Port ( rs,clk : in  STD_LOGIC;
                q : inout  STD_LOGIC_VECTOR (3 downto 0));
        end asybin1;
architecture Behavioral of asybin1 is
        signal div:std_logic_vector(22 downto 0);
        signal temp:STD_LOGIC_VECTOR (3 downto 0);
        signal clkd:std_logic;
begin
process(clk)
begin
        if rising_edge(clk)then
        div<= div+1;
        end if;
end process;
        clkd<=div(22);
process(clkd,rs)
begin
        if(rs='1')then temp<=(others=>'0');      --for down counter temp=>"1111";
        elsif(clkd='1' and clkd'event) then
        temp<=temp+1;                            --for down counter temp<= temp-1;
        q<= temp;
        end if;
end process;
end Behavioral;
```

15. SYNCHRONOUS BINARY UP COUNTER:

3-bit Asynchronous up counter			
Clock	QC	QB	QA
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0
9	0	0	1




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity synbicount is
           Port ( rs,clk : in  STD_LOGIC;
                q : inout  STD_LOGIC_VECTOR (3 downto 0));
        end synbicount;
architecture Behavioral of synbicount is
        signal div:std_logic_vector(22 downto 0);
        signal temp:STD_LOGIC_VECTOR (3 downto 0);
        signal clkd:std_logic;
begin
process(clk)
begin
        if rising_edge(clk)then
        div<= div+1;
        end if;
end process;
        clkd<=div(22);
process(clkd)
begin
        if(clkd='1' and clkd'event) then
        if(rs='1')then temp<=(others=>'0');  ---for down counter temp<="1111"
        else temp<=temp+1;                    ---for down counter temp<= temp-1;
        end if;
        q<= temp;
        end if;
end process;
end Behavioral;
```

16. BCD UP COUNTER:

Clock	QD	QC	QB	QA
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	0	0	0	0

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity bcdupcount is
        Port ( clk,rst : in  STD_LOGIC;
        q : inout  STD_LOGIC_VECTOR (3 downto 0));
        end bcdupcount;
architecture Behavioral of bcdupcount is
        signal div:std_logic_vector(22 downto 0);
        signal clkd:std_logic;
begin
process(clkd)
begin
        if rising_edge(clk)then
        div<= div+1;
        end if;
end process;
        clkd<=div(22);
process(clkd,rst)
begin
        if rst='0' or q="1010" then
        q<="0000";
        elsif clkd'event and clkd='1' then
        q<=q+1;
        end if;
end process;
         q<=q;
 end Behavioral;
```

17. BCD DOWN COUNTER:

Clock	QD	QC	QB	QA
0	1	0	0	1
1	1	0	0	0
2	0	1	1	1
3	0	1	1	0
4	0	1	0	1
5	0	1	0	0
6	0	0	1	1
7	0	0	1	0
8	0	0	0	1
9	0	0	0	0

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity bcddowncounter1 is
        Port ( clk,rst : in  STD_LOGIC;
        q : inout  STD_LOGIC_VECTOR (3 downto 0));
        end bcddowncounter1;
architecture Behavioral of bcddowncounter1 is
        signal div:std_logic_vector(22 downto 0);
        signal clkd:std_logic;
begin
process(clkd)
begin
        if rising_edge(clk)then
        div<= div+1;
        end if;
end process;
        clkd<=div(22);
process(clkd,rst)
begin
        if rst='0' or q="1111" then
        q<="1001";
        elsif clkd'event and clkd='1' then
        q<=q-1;
        end if;
end process;
        q<=q;
end Behavioral;
```

18. VHDL CODE FOR DC MOTOR INTERFACE

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;
        Entity dcmotor is
        Port (start,dir,clk:in std_logic;
                pwm_out:out std_logic;
                out_dc:out std_logic_vector(1 downto 0));
        end dcmotor;
architecture dcmotor_a of dcmotor is
        signal clk1:std_logic;
        signal div:std_logic_vector (24 downto 0);
begin
process (clk,start)
begin
        if(start='0') then
        div<="00000000000000000000000000";
        elsif(clk' event and clk='1')then
        div<=div+1;
        end if;
        clk1<=div(19);
end process;
process(clk1)
begin
        if(clk1'event and clk='1')then
        if start='0' then
        out_dc<="00";
        elsif start='1' and dir='1' then
        out_dc<="10";
        pwm_out<='1';
        elsif start='1' and dir='0' then
        out_dc<="01";
        pwm_out<='1';
        end if;
        end if;
end process;
end dcmotor_a;
```

PIN ASSIGNMENT

XC3128TQ144		
Connector	Device pin	Property
	128	Clk
P 18/6	6	Dir
P 18/13	14	Out_dc(0)
P 18/14	15	Out_dc(1)
P 18/11	11	Pwm_out
P 18/5	5	Start

XC2S100TQ144-5		
Connector	Device pin	Property
	18	Clk
P 18/6	44	Dir
P 18/13	54	Out_dc(0)
P 18/14	56	Out_dc(1)
P 18/11	50	Pwm_out
P 18/5	43	Start

19. VHDL CODE FOR STEPPER MOTOR INTERFACE

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;
        entity stm_st is
        port (clk:in std_logic; rst:in std_logic;
    out_stm:out std_logic_vector(3 downto 0));
        end stm_st;
        architecture stm_st_a of stm_st is
        type state_type is (s0,s1,s2,s3);
        signal state:state_type;
        signal div:st_logic_vector(20 downto 0);
        signal lk,clkwise,start:std_logic;
begin
process(clk,rst)
begin
        if (rst='0') then
        div<=(others=>'0');
        elsif(clk'event and clk='1') then
        div<=div+1;
        end if;
end process;
        lk<=dic(15);
process(lk,rst,clkwise)
begin
        if(rst='1')then
        state<=s0;
        elsif lk'event and lk='1' then
        if clkwise='0' then
        case state is
        when s0=>state<=s1;
        when s1=>state<=s2;
        when s2=>state<=s3;
        when s3=>state<=s0;
        when others=>null;
        end case
        end if;
        end if;
end process;
        with state select
        out_stm<="0110" when s0,
        "1010" when s1, "1001" when s2,
        "0101" when s3;
End stm_st_a;
```

PIN ASSIGNMENT

XC3128TQ144		
Connector	Device pin	Property
	128	Clk
P 14/1	88	Dir
P 15/1	132	Out_stm(0)
P 15/2	131	Out_stm(1)
P 15/3	121	Out_stm(2)
P 15/4	120	Out_stm(3)
	125	Rst

XC2S100TQ144-5		
Connector	Device pin	Property
	18	Clk
P 18/5	43	Dir
P 18/21	62	Out_stm(0)
P 18/22	65	Out_stm(1)
P 18/19	60	Out_stm(2)
P 18/20	64	Out_stm(3)
	86	Rst

20. VHDL CODE FOR SEVEN SEGMENT DISPLAY INTERFACE

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;
        Entity mux_disp is
        Port (clk:in std_logic                                  ---4mhz
        Rst: in std_logic;
        seg: out std_logic_vector(6 downto 0)
        base: out std_logic_vector(3 downto 0));
        end mux_disp;
architecture mux_disp_arch of mux_disp is
        signal count : std_logic_vector(25 downto 0);
        signal base_cnt: std_logic_vector(1 downto 0);
        signal seg_cnt: std_logic_vector(3 downto 0);
begin
process(clk,rst)
begin
        if rst = '1' then
        count<=(others=>'0');
        elsif
        clk='1' and clk'event then
        count<= + '1';
end if;
end process;
        base_cnt<=count(12 downto11);
        seg_cnt<=count(25 downto 22);
Base<= "1110" when base_cnt="00" else
        "1101" when base_cnt="01" else
        "1011" when base_cnt="10" else
        "0111" when base_cnt="11" else
        "1111";
Seg<=   "0111111" when seg_cnt="0000" else          ---0
        "0000110" when seg_cnt="0001" else          ---1
        "1011011" when seg_cnt="0010" else          ---2
        "1001111" when seg_cnt="0011" else          ---3
        "1100110" when seg_cnt="0100" else          ---4
        "1101101" when seg_cnt="0101" else          ---5
        "1111101" when seg_cnt="0110" else          ---6
        "0000111" when seg_cnt="0111" else          ---7
        "1111111" when seg_cnt="1000" else          ---8
        "1100111" when seg_cnt="1001" else          ---9
        "1110111" when seg_cnt="1010" else          ---A
        "1111100" when seg_cnt="1011" else          ---B
        "0111001" when seg_cnt="1100" else          ---C
        "1011110" when seg_cnt="1101" else          ---D
        "1111001" when seg_cnt="1110" else          ---E
        "1110001" when seg_cnt="0000" else          ---F
        "0000000";
        End mux_disp_arch;
```

21. VHDL CODE FOR RELAY INTERFACE

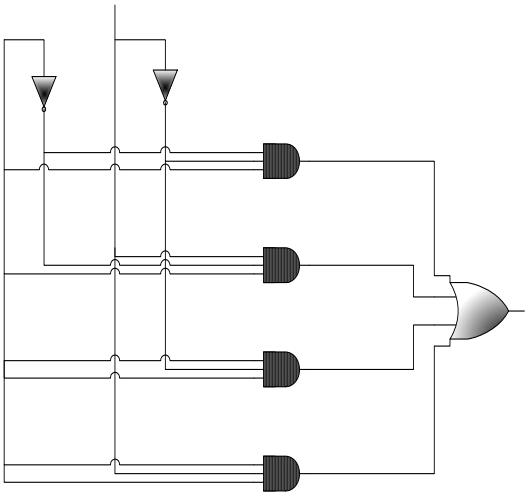
```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;
        Entity relay is
        Port (sw : in std_logic;
                Rl1,led : out std_logic);
        End realy;
Architecture behavioral of relay is
Begin
        Rl1 <= ws;
        Led <= sw;
End behavioral;
```

PIN ASSIGNMENT

XC3128TQ144		
Connector	Device pin	Property
P 18/3	41	Sw
P 18/5	43	Rl1
P 18/21	62	Led

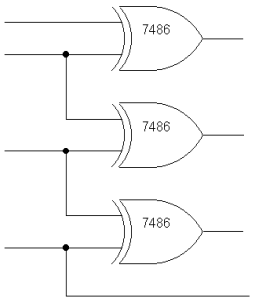
XC2S100TQ144-5		
Connector	Device pin	Property
P 18/3	1	Sw
P 18/5	5	Rl1
P 18/21	23	Led

22. VHDL CODE FOR MULTIPLEXER (4:1)(STRUCTURAL):



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity mux is
        Port ( i0,i1,i2,i3,s0,s1 : in  STD_LOGIC;
        y : out  STD_LOGIC);
        end mux;
architecture struct of mux is
        component and1                          ----components, entity and architecture
        port (l,m,u:in std_logic;n:out std_logic);      --- must be declared separately
        end component;
        component or1                           ----components, entity and architecture
        port (o,p,x,y:in std_logic;q:out std_logic);    --- must be declared separately
        end component;
        component not1                          ----components, entity and architecture
        port (r:in std_logic;s:out std_logic);          --- must be declared separately
        end component;
        signal s2,s3,s4,s5,s6,s7:std_logic;
begin
        u1:and1 port map(i0,s2,s3,s4);
        u2:and1 port map(i1,s2,s1,s5);
        u3:and1 port map(i2,s0,s3,s6);
        u4:and1 port map(i3,s0,s1,s7);
        u5:not1 port map(s0,s2);
        u6:not1 port map(s1,s3);
        u7:or1 port map(s4,s5,s6,s7,y);
end struct;
```

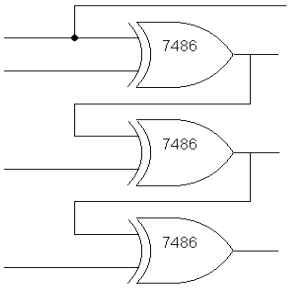
23. VHDL CODE FOR BINARY TO GRAY (Structural):



Using EX-OR gates

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity bg is
        Port ( b : in  STD_LOGIC_VECTOR (3 downto 0);
        g : out  STD_LOGIC_VECTOR (3 downto 0));
        end bg;
architecture struct of bg is
        component xor1 port(a,b:in std_logic;   ----components, entity and architecture
        c:out std_logic);                       --- must be declared separately
        end component;
        component and1 port(d,e:in std_logic;   ----components, entity and architecture
        f:out std_logic);                       --- must be declared separately
        end component;
begin
        u1:and1 port map(b(3),b(3),g(3));
        u2:xor1 port map(b(3),b(2),g(2));
        u3:xor1 port map(b(2),b(1),g(1));
        u4:xor1 port map(b(1),b(0),g(0));
end struct;
```

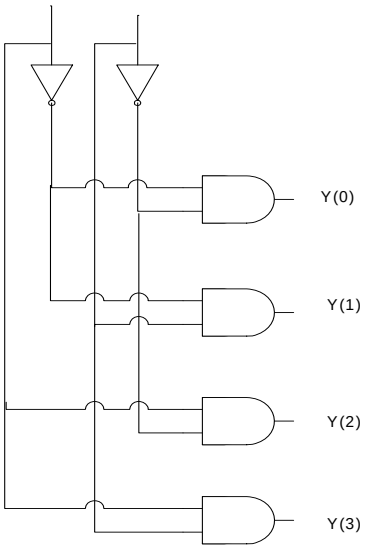
24. VHDL CODE FOR GRAY TO BINARY (Structural):



Using EX-OR gates

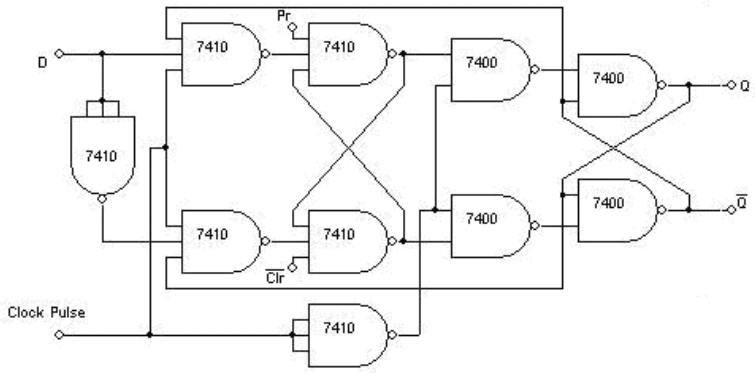
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity gb is
        Port ( g : in  STD_LOGIC_VECTOR (3 downto 0);
        b : out  STD_LOGIC_VECTOR (3 downto 0));
        end gb;
architecture struct of gb is
        component xor1 port(a,b:in std_logic;    ----components, entity and architecture
        c:out std_logic);                        --- must be declared separately
        end component;
        component and1 port(d,e:in std_logic;    ----components, entity and architecture
        f:out std_logic);                        --- must be declared separately
        end component;
        component xor12 port(g,h,i:in std_logic; ----components, entity and architecture
        k:out std_logic);                        --- must be declared separately
        end component;
        component xor13 port(l,m,n,o:in std_logic; ----components, entity and architecture
        p:out std_logic);                        --- must be declared separately
        end component;
begin
u1:and1 port map(g(3),g(3),b(3));
u2:xor1 port map(g(3),g(2),b(2));
u3:xor12 port map(g(3),g(2),g(1),b(1));
u4:xor13 port map(g(3),g(2),g(1),g(0),b(0));
end struct;
```

25. VHDL CODE FOR DECODER (Structural):



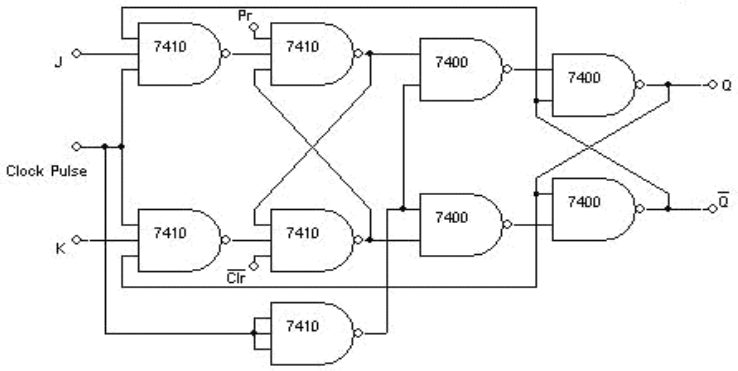

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity dec is
        Port ( a,b : in  STD_LOGIC;
        y: out  STD_LOGIC_VECTOR (3 downto 0));
        end dec;
architecture Behavioral of dec is
        component and1 is                       ----components, entity and architecture
        port(p,q:in std_logic;r:out std_logic);  --- must be declared separately
        end component;
        component not1 is                        ----components, entity and architecture
        port (d:in std_logic;e:out std_logic);   --- must be declared separately
        end component;
        signal s1,s2:std_logic;
begin
        u1:and1 port map(s1,s2,y(0));
        u2:and1 port map(s1,b,y(1));
        u3:and1 port map(a,s2,y(2));
        u4:and1 port map (a,b,y(3));
        u5:not1 port map(a,s1);
        u6:not1 port map(b,s2);
end Behavioral;
```

26. VHDL CODE FOR D FLIP FLOP (Structural):



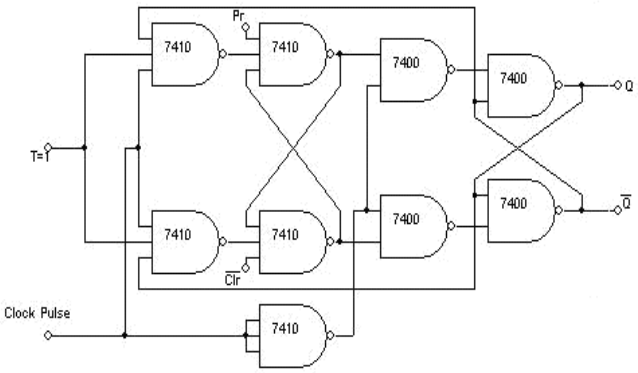

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity dff1 is
        Port ( d,clk,pr,clr : in  STD_LOGIC;
        q,qn : inout  STD_LOGIC);
        end dff1;
architecture struct of dff1 is
        component clkdiv is                     ----components, entity and architecture
        port(clk:in std_logic;clk_d:out std_logic);   --- must be declared separately
        end component;
        component nand1 is
        port(a,b,c:in std_logic;                ----components, entity and architecture
        d:out std_logic);                       --- must be declared separately
        end component;
        component nand12 is
        port(x,y:in std_logic;                  ----components, entity and architecture
        z:out std_logic);                       --- must be declared separately
        end component;
        component nand13 is
        port(e:in std_logic;                    ----components, entity and architecture
        f:out std_logic);                       --- must be declared separately
        end component;
        signal s1,s2,s3,s4,s5,s6,s7,s8,s9:std_logic;
begin
        u10:clkdiv port map(clk,s7);
        u1:nand1 port map(d,s7,qn,s1);
        u2:nand1 port map(s9,s7,q,s2);
        u3:nand1 port map(pr,s1,s4,s3);
        u4:nand1 port map(s2,clr,s3,s4);
        u5:nand12 port map(s3,s8,s5);
        u6:nand12 port map(s8,s4,s6);
        u7:nand12 port map(s5,qn,q);
        u8:nand12 port map(s6,q,qn);
        u9:nand13 port map(s7,s8);
        u11:nand13 port map(d,s9);
end struct;
```

27. <u>VHDL CODE FOR JK FLIP FLOP (Structural):</u>



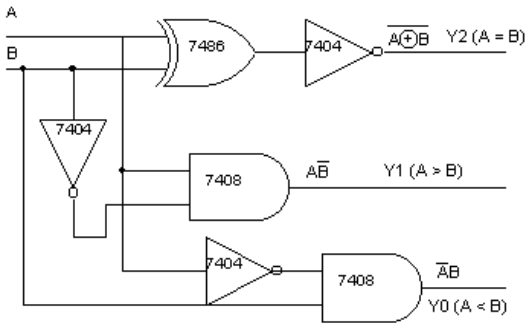

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity jkff is
        Port ( j,k,clk,pr,clr : in  STD_LOGIC;
        q,qn : inout  STD_LOGIC);
        end jkff;
architecture struct of jkff is
        component clkdiv is                     ----components, entity and architecture
        port(clk:in std_logic;clk_d:out std_logic); --- must be declared separately
        end component;
        component nand1 is                      ----components, entity and architecture
        port(a,b,c:in std_logic;                --- must be declared separately
        d:out std_logic);
        end component;
        component nand12 is                     ----components, entity and architecture
        port(x,y:in std_logic;                   --- must be declared separately
        z:out std_logic);
        end component;
        component nand13 is
        port(e:in std_logic;                    ----components, entity and architecture
        f:out std_logic);                       --- must be declared separately
        end component;
        signal s1,s2,s3,s4,s5,s6,s7,s8:std_logic;
begin
        u10:clkdiv port map(clk,s7);
        u1:nand1 port map(j,qn,s7,s1);
        u2:nand1 port map(k,s7,q,s2);
        u3:nand1 port map(pr,s1,s4,s3);
        u4:nand1 port map(s2,clr,s3,s4);
        u5:nand12 port map(s3,s8,s5);
        u6:nand12 port map(s8,s4,s6);
        u7:nand12 port map(s5,qn,q);
        u8:nand12 port map(s6,q,qn);
        u9:nand13 port map(s7,s8);
end struct;
```

28. VHDL CODE FOR T FLIP FLOP (Structural):




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity tff1 is
        Port ( t,clk,pr,clr : in  STD_LOGIC;
        q,qn : inout  STD_LOGIC);
        end tff1;
architecture struct of tff1 is
        component clkdiv is                     ----components, entity and architecture
        port(clk:in std_logic;clk_d:out std_logic); --- must be declared separately
        end component;
        component nand1 is
        port(a,b,c:in std_logic;                ----components, entity and architecture
        d:out std_logic);                       --- must be declared separately
        end component;
        component nand12 is
        port(x,y:in std_logic;                  ----components, entity and architecture
        z:out std_logic);                       --- must be declared separately
        end component;
        component nand13 is
        port(e:in std_logic;                    ----components, entity and architecture
        f:out std_logic);                       --- must be declared separately
        end component;
        signal s1,s2,s3,s4,s5,s6,s7,s8:std_logic;
begin
        u10:clkdiv port map(clk,s7);
        u1:nand1 port map(t,qn,s7,s1);
        u2:nand1 port map(t,s7,q,s2);
        u3:nand1 port map(pr,s1,s4,s3);
        u4:nand1 port map(s2,clr,s3,s4);
        u5:nand12 port map(s3,s8,s5);
        u6:nand12 port map(s8,s4,s6);
        u7:nand12 port map(s5,qn,q);
        u8:nand12 port map(s6,q,qn);
        u9:nand13 port map(s7,s8);
end struct;
```

29. <u>1 BIT COMPARATOR (structural):</u>



A	B	Y1 (A>B)	Y2 (A = B)	Y3 (A < B)
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
        entity comp is
        Port ( a,b : in  STD_LOGIC; y : out  STD_LOGIC_VECTOR (2 downto 0));
        end comp;
architecture struct of comp is
        component and1
        port(l,m:in std_logic;              ----components, entity and architecture
        n:out std_logic);                    --- must be declared separately
        end component;
        component xnor1 is
        port(p,q:in std_logic;              ----components, entity and architecture
        r:out std_logic);                   --- must be declared separately
        end component;
        component notgate1 is
        port(s:in std_logic;                ----components, entity and architecture
        t:out std_logic);                   --- must be declared separately
        end component;
        signal s1,s2:std_logic;
begin
        u1:and1 port map(a,s2,y(0));
        u2:and1 port map(s1,b,y(1));
        u3:xnor1 port map(a,b,y(2));
        u4:notgate1 port map(a,s1);
        u5:notgate1 port map(b,s2);
end struct;
```

Lab-01 Schematic

"**Introduction to DSCH and a Simple Gate Implementation in DSCH"**

1. Objective

In this lab students will be introduced to a schematic based EDA tool "DSCH" and the introduction will be accompanied with an implementation of simple Gate at Gate level and at CMOS switch level. The tool used in this lab is DSCH. The goals for this Lab are:

- Familiarity and Hands on Example using the tool.
- Schematic Design in using the tool.
- Schematic Design Verification.
- Simulation of the design

2. Theory (NOR Gate)

As per discussion and design on white board in the Lab, a NOR gate can be implemented using four FETS i.e. two pFETs and two nFETs as the inputs of the gate is two. pFETs are connected in series while nFETs are connected in parallel, Vdd is supplied to the series combination of pFETs while the parallel combination of nFETs is grounded. Inputs a & b are applied to the gate terminals of all FETs, and the output f is obtained from the common junction of these parallel and series combinations as illustrated in NOR circuit under the heading of Circuit

3. Block Diagram and Logic Diagram

- **Symbol, Truth Table and CMOS circuit of NOR Gate**

4. Lab Instructions

a) Open the **DSCH** by double clicking it located in the installed directory of **dsch2-7**

The following Screen will be appeared

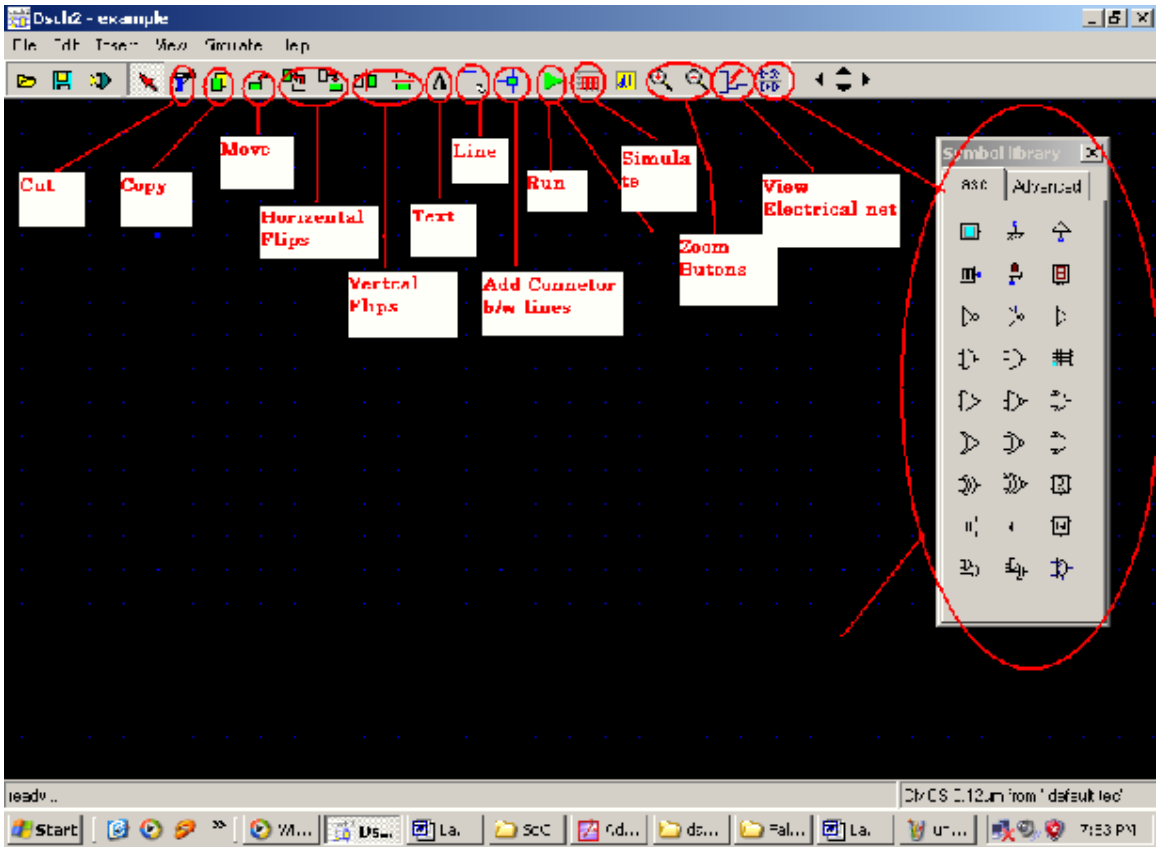


Figure 1.1 Main Window DSCH

b) Select the foundry using the command **File > Select Foundry**

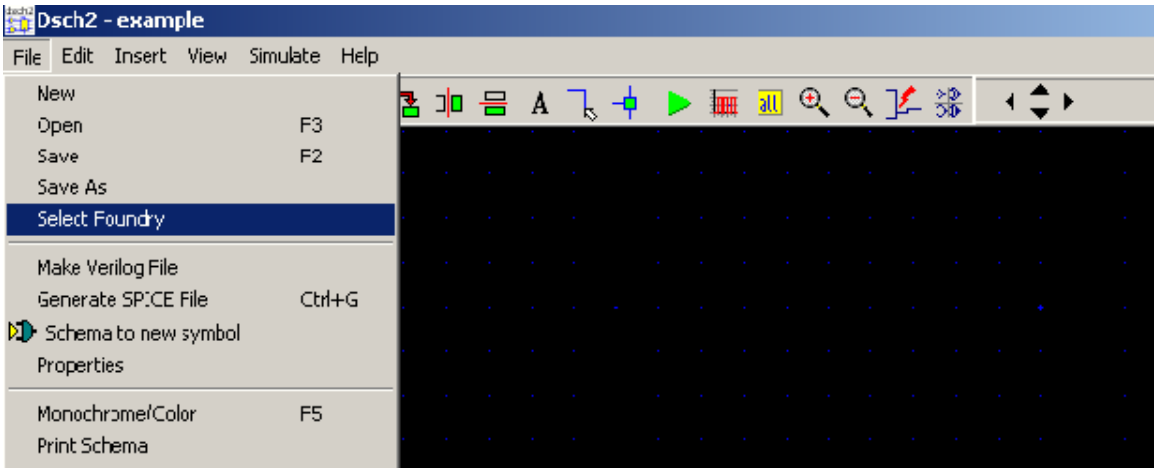


Figure 1.2 Select Foundry Window

c) Select **0.25-micron** process by selecting **"cmos025.tec"** file. Click Open tab to continue.

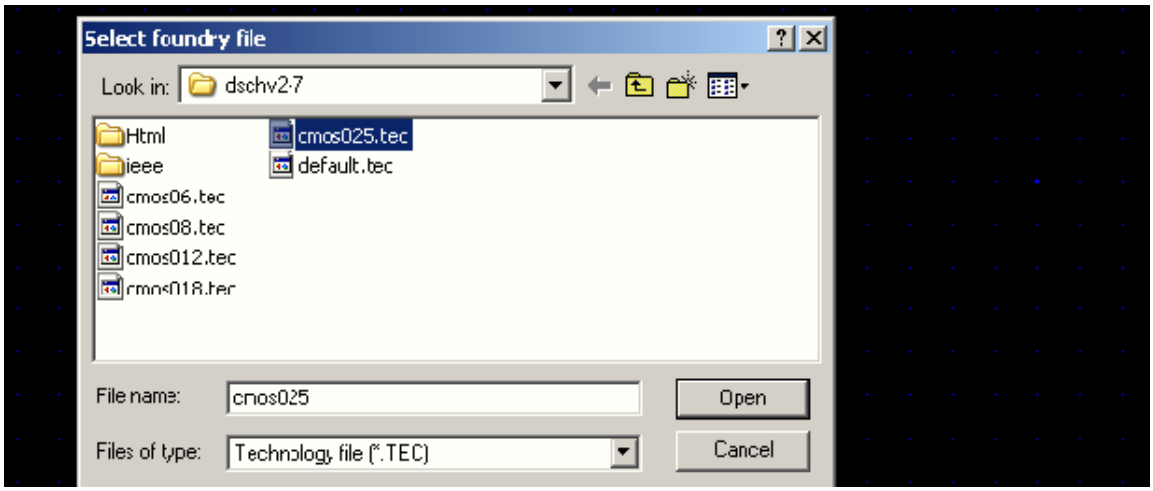


Figure 1.3 Select Foundry File

d) **Save** the design as "Lab05" using the command **File > Save as.**
e) **Open** a schematic design using the command **Insert > Another Schema (.SCH).**

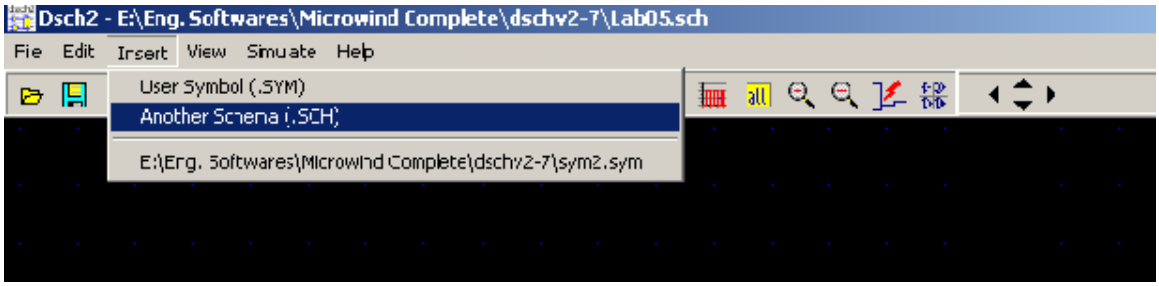


Figure 1.4 Insert Schematics

Select "NOR2_Cir.sch" click on Open tab to open the schematic. The following schematic of half adder will be open. Analyze the schematic carefully, especially the connecting wires.

f) Click on the **Run** Tab on the Tool bar menu to start the simulation or using the command **Simulate > Start Simulation**.

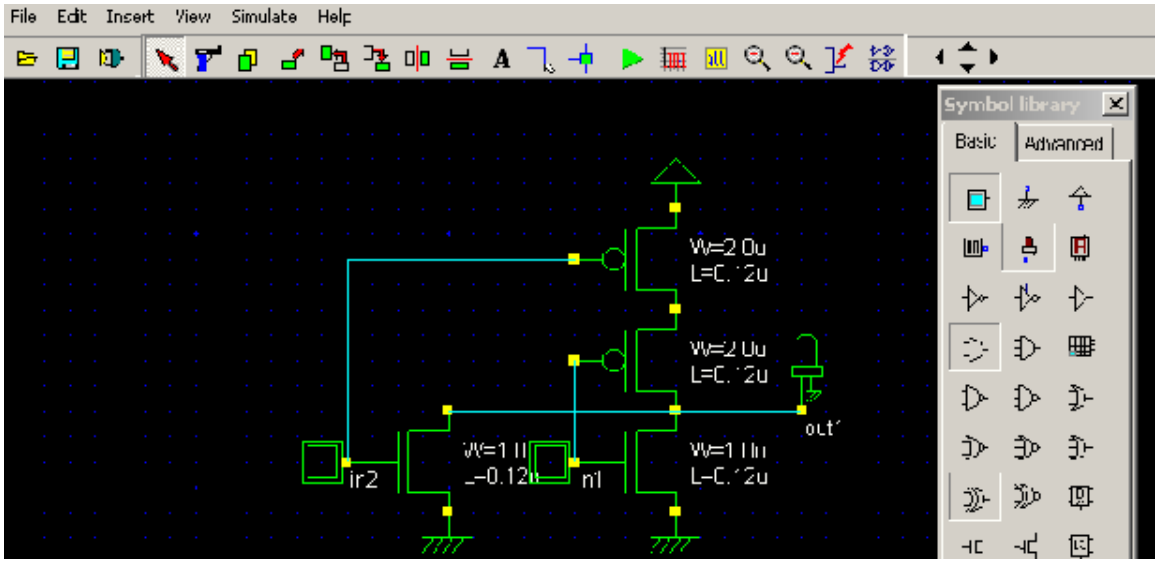


Figure 1.5 Working with DSCH

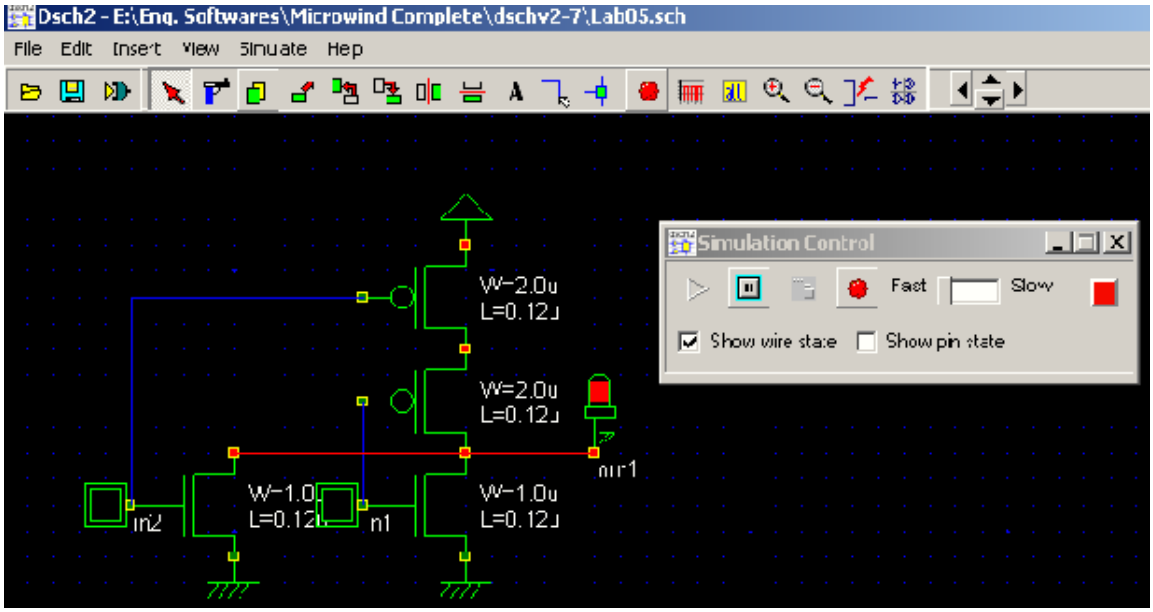


Figure 1.6 Simulating the Circuit

g) Check for different input combinations by using mouse click on the inputs buttons to be on or off. Like in the above diagram the inputs in1 and in2 is "0" on as indicated while the output out1 on as indicated by red color.
h) Now we will make the above schematics manually. Delete the existing diagram by selecting the **Cut** command button from the icon menu bar and then select the whole diagram.
i) Click on the **pMOS** and **nMOS** symbol button in the Symbol Library and drag it the schematic design areas as indicated in the figure.

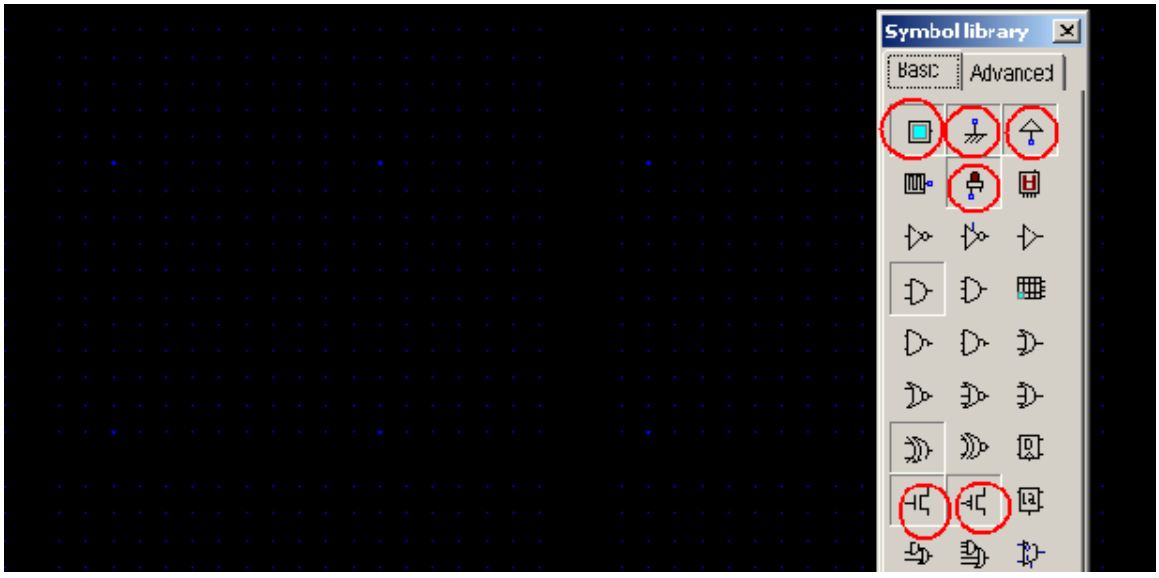


Figure 1.7 Symbol Library

Similarly complete the half adder schematic by adding and gate from symbol library and input buttons and output lights at the respective nodes from the symbol library as given in the following figure. Also add a name to your schematic for evaluation purpose.

j) You can view the Timing diagram by clicking on Timing Diagram command button on the icon bar.

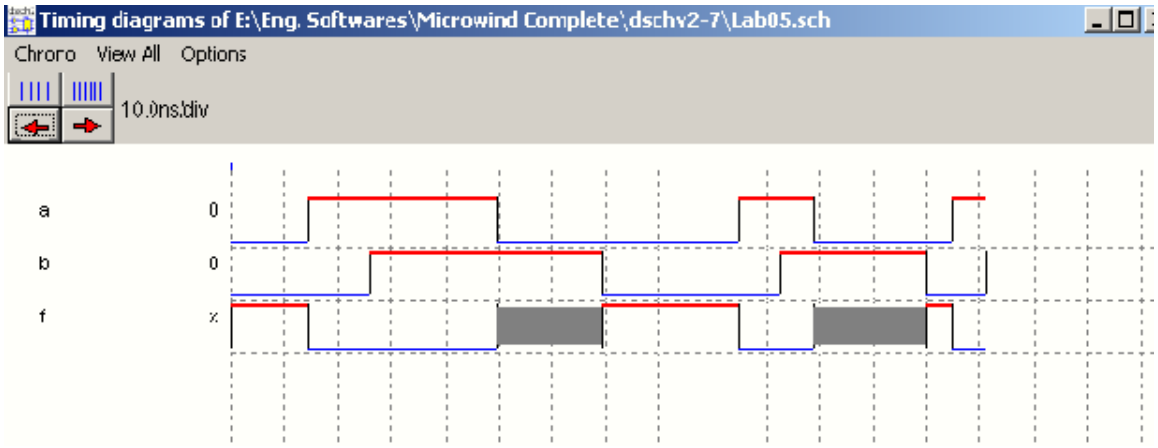


Figure 1.8 Timing Diagram

k) **Save** the design.

l) Explore the Schema to new symbol, Make Verilog File, Generate Spice netlist in the File menu, delays of each symbol and the Advanced Tab in the symbol library

5. Lab Report

- Give a short description of the contents of the lab
- Include block diagram/diagrams of your design in the lab report
- Describe your design approach and explain the working of each the schematic
- Include schematic of your design Include the test bench module in your design that must include at least six test inputs
- Include the results in your report
- Only follow the provided cover page format.

Lab-02 Layout

"Introduction to MicroWind and Analysis of MOSFETs"

1. Objective

In this lab students will be introduced to a Layout based EDA tool "MicroWind" and the introduction will be accompanied with analysis of MOS transistors. The tool used in this lab is MicroWind. The goals for this Lab are:

- Familiarity and Hands on Example using the tool.
- Layout Design using the tool.
- Study of MOSFET Characteristics.
- Analog Simulation of MOSFETs.

2. Theory

2.1. MOSFET

The Metal Oxide Semiconductor Field Effect Transistor is very important part of Digital Integrated Circuits. It is mostly used as switch in digital design. MOSFET is a four terminal device. The voltage applied to the gate terminal determine the current flow between drain and source terminals. The body/substrate of the transistor is the fourth terminal. Mostly the fourth terminal (body/substrate) of the device is connected to dc supply that is identical for all devices of the same type (GND fro nMOS and Vdd for pMOS). Usually this terminal is not shown on the schematics.

2.2. nMOS

The nMOS transistor consists of n+ drain and source diffusion regions, which are embedded in a p-type substrate. The electrons in the channel beneath the gate between source and drain terminal are responsible for the current flow.

2.3. pMOS

The pMOS transistor consists of p+ drain and source diffusion regions, which are embedded in an n-type substrate. The holes in the channel beneath the gate between source and drain terminal are responsible for the current flow.

2.4. CMOS

The CMOS (Complementary MOS) consist of both p-type and n-type MOS. The advantage of CMOS is its low power design due to its Static behavior.

3. Design/ Diagram/Circuit

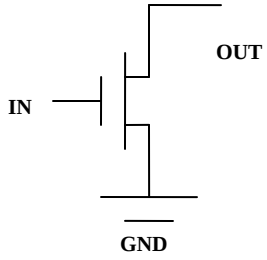


Figure 2.1 NMOS Symbol

4. Lab Instructions

a) Open the Microwind2 by double clicking it located in the installed directory of **dsch2-7**

The following screen will be appeared

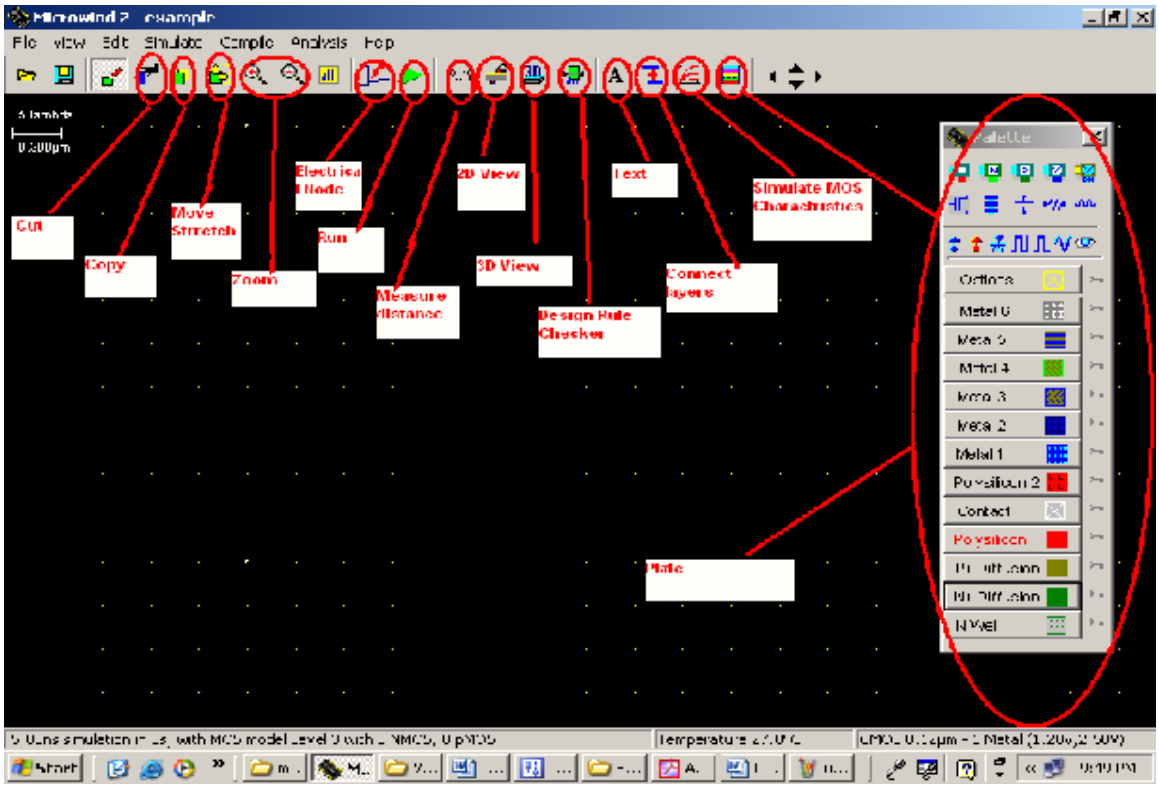


Figure 2.2 Main Window MicroWind

b) Select the foundry using the command **File > Select Foundry**

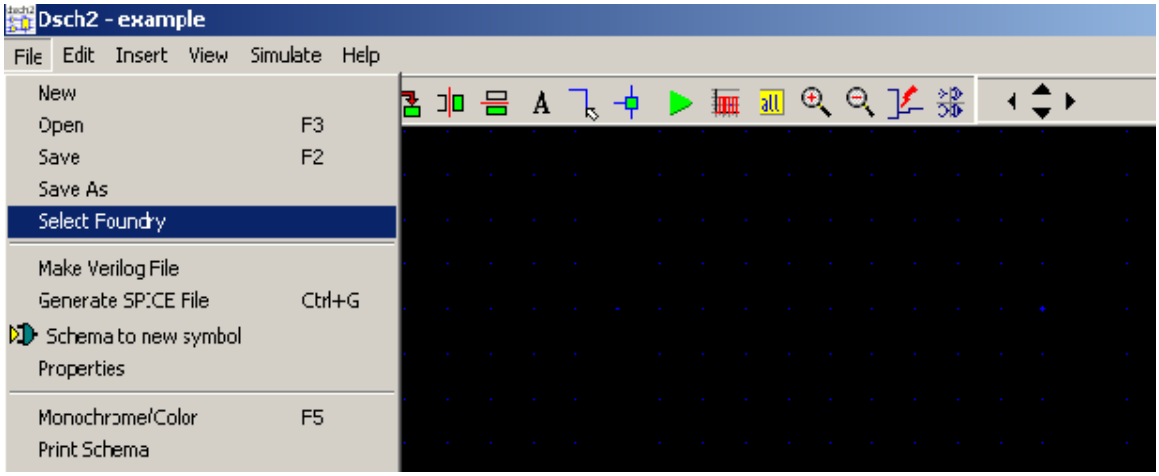


Figure 2.3 Select Foundry Window in MicroWind

c) Select 0.25-micron process by selecting **"cmos025.tec"** file. Click Open tab to continue.

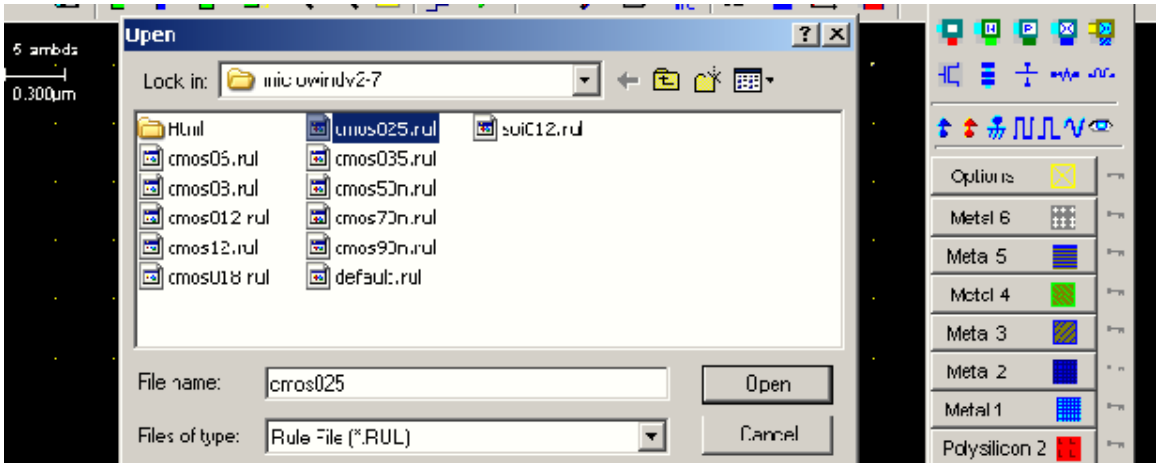


Figure 2.4 Foundry Selection Window MicroWind

d) **Save** the design as "Lab02" using the command **File > Save as.**
e) **Create** an nMOS by using the **nMOS generator** button in the Palate

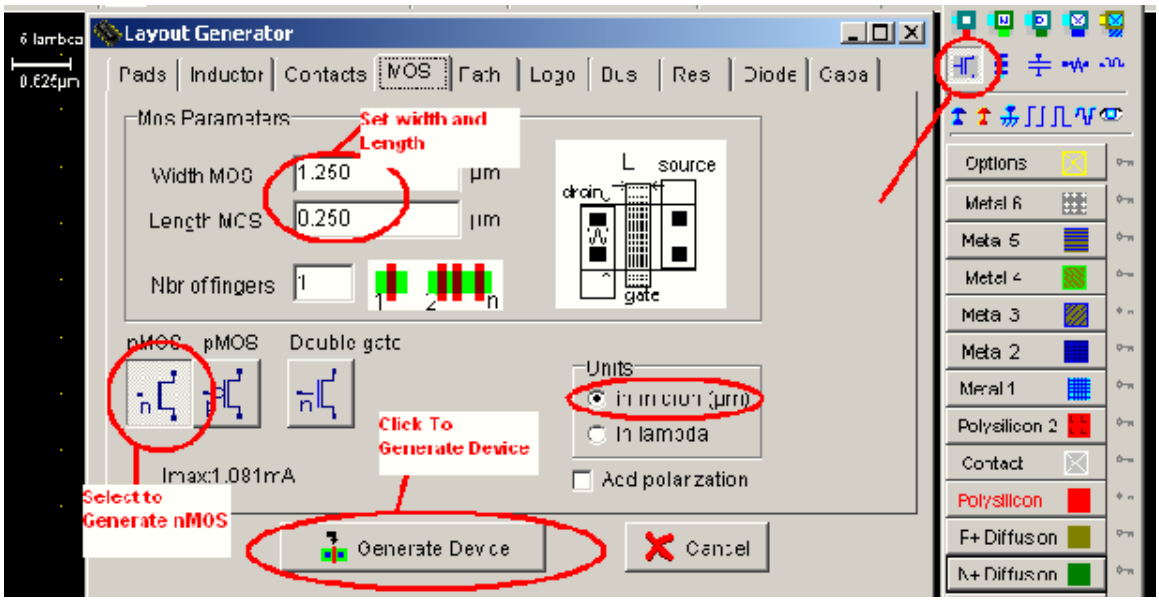


Figure 2.4 Layout Generator MicroWind

You can set the width and length of MOS by typing in the fields **Width MOS and Length MOS** either in micron or in lambda units as indicated in the above figure.

f) **Click** on Generate Device Tab to generate the device.

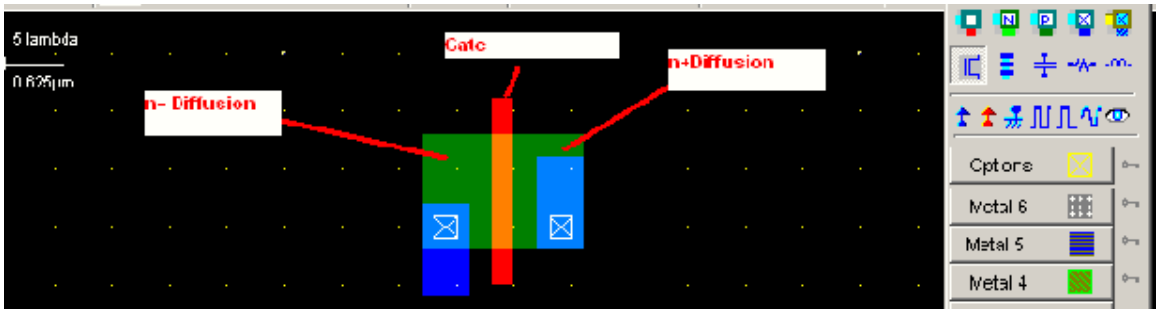


Figure 2.5 NMOS Symbol in MicroWind

g) Apply the voltages and output node using the symbol buttons Vdd, Gnd, Add a Pulse, and Visible node in the Palate menu, as indicated in the following figure. You can use the **Stretch/Move** command button for these actions.

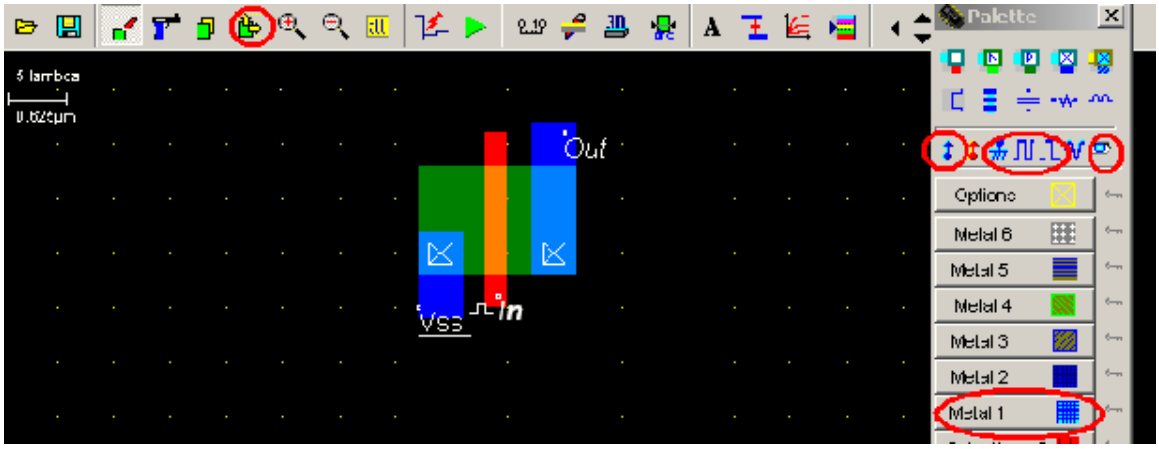


Figure 2.5 Applying Input to NMOS

h) **Click** on the **Run Tab** on the Tool bar menu to start the simulation or using the command **Simulate > Run Simulation**.

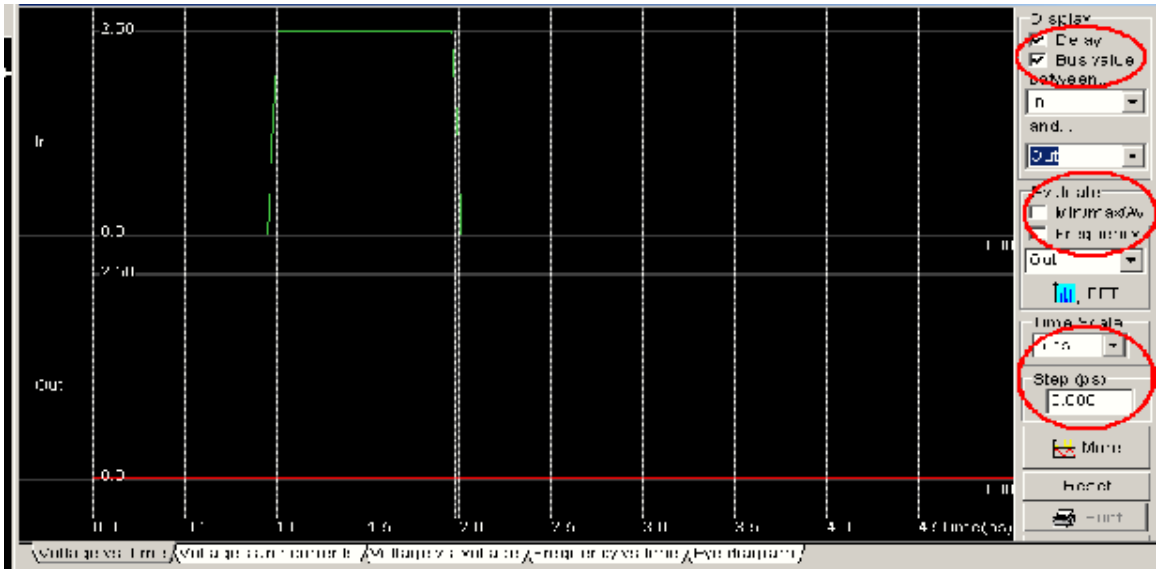


Figure 2.6 Timing Diagram in MicroWind

i) Now apply the Vdd to the n+ diffusion or drain terminal instead of Vss, run the Simulation again

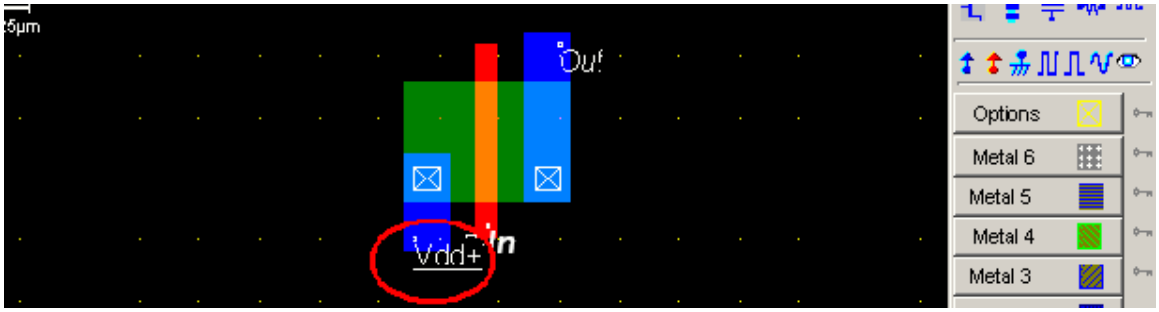


Figure 2.7 NMOS in MicroWind

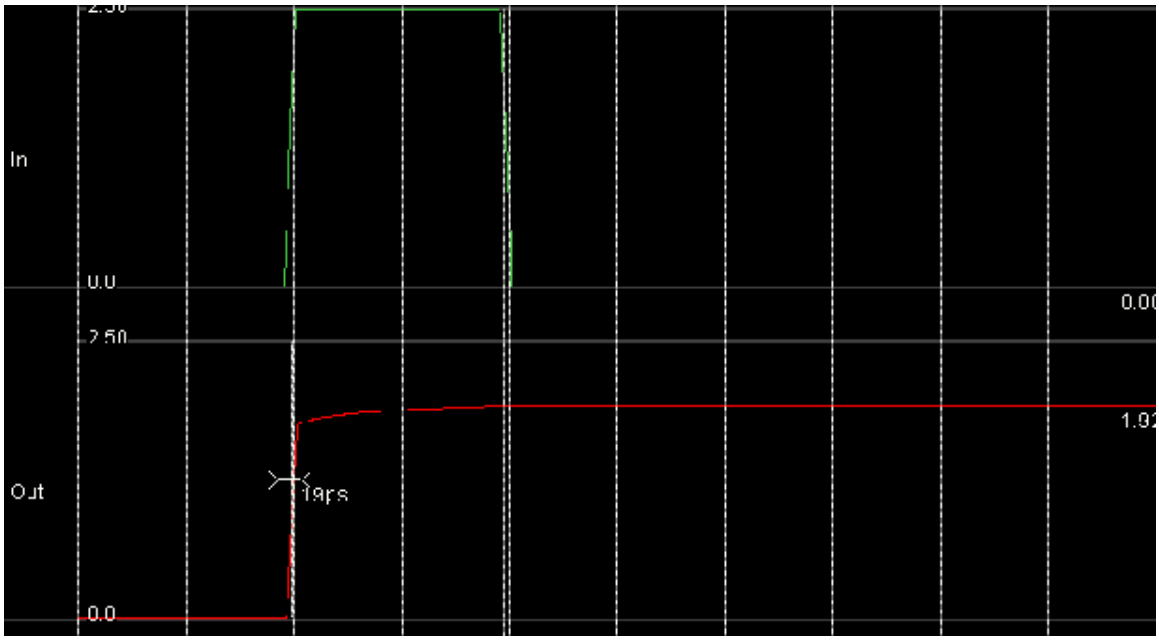


Figure 2.8 Analysis of NMOS in MicroWind

Analyze the simulation waveform, use different values of voltages for Vdd by double clicking on it and set the voltage level. Now we will make the above schematics.

j) Now increase the width of nMOS either by using the stretch command for the previous layout, by generating again the nMOS with significantly (5 to 10 times more) increased width, or by using manual layout from the Palate menu. In the below figure it is 50 (lambda) or 6.250µ m, and run the simulation again, analyze the effect of width on the propagation delay.

Figure 2.9 Changed Width of NMOS MicroWind

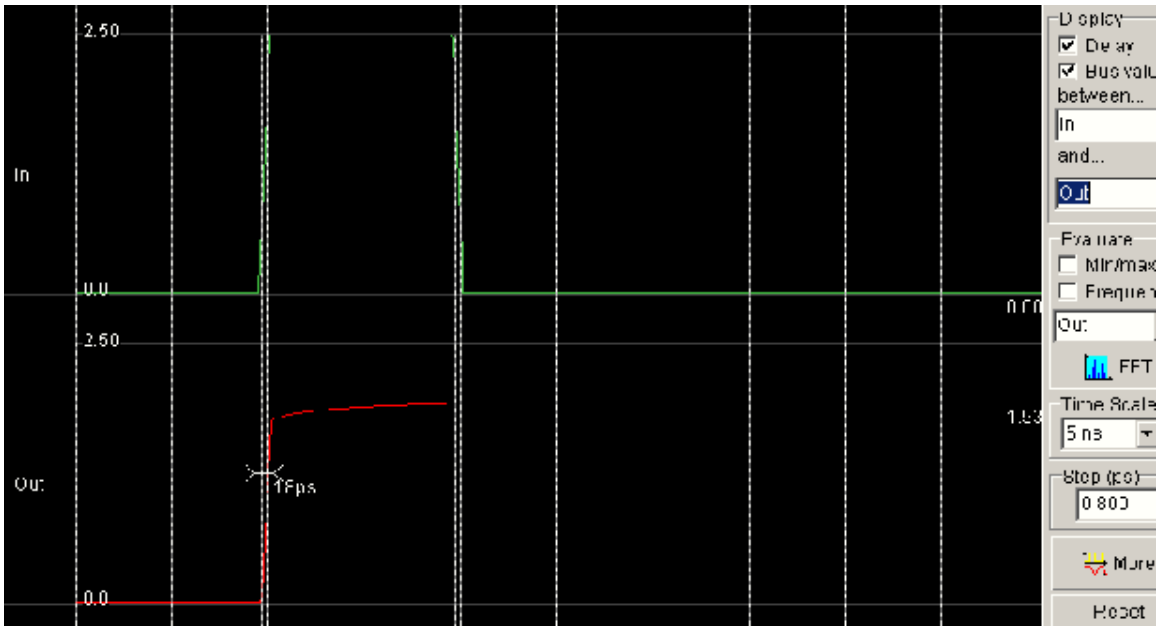


Figure 2.10 Response of NMOS after changing its width

Similarly the nMOS can be analyzed using different widths and different input voltages.

k) **Save** the design.

The characteristics of the pMOS are similar to the nMOS. Design the pMOS Layout and analyzed in the similar way as nMOS Explore the Simulation Graphs. Explore different device parameters and the commands in the drop down menus.

4. Lab Report

- Give a short description of the contents of the lab
- Include block diagram/diagrams of your design in the lab report
- Describe your layout design approach parameters and explain the effects of each the parameter
- Include layout of your design I
- Include the results in timing waveform format in your report
- Only follow the provided cover page format.

5. Simulation Anlysis (Include in your Lab Report)

SR. #	**Width of nMOS**	**Drain term voltage**	**Input Voltage Level**	**Propagation Delay**	**Output voltage Level**	**Current I_D**
1	**10 λ**	V_{SS}	**2.5v**			
2			**0.5v**			
3			**1.5v**			
4			**5.5v**			
5		V_{DD}	**2.5v**			
6			**0.5v**			
7			**1.5v**			
8			**5.5v**			
9	**50 λ**	V_{SS}	**2.5v**			
10			**0.5v**			
11			**1.5v**			
12			**5.5v**			
13		V_{DD}	**2.5v**			
14			**0.5v**			
15			**1.5v**			
16			**5.5v**			

You can increase the table and also the entries for in depth analysis.

A properly presented in depth analysis with graph based on the table entries will be highly appreciated.

Discuss the Effects of width design parameter of the MOS devices on their behavior.

Lab-03

"MOS Device Characteristics CMOS Layout Simulation and Parametric Analysis"

1. Objectives

To become acquainted with MOS device characteristics and full custom design by using Micro Wind tools. This lab consists of three parts. Part I is to analyse NMOS and PMOS devices. Part II is to draw a layout of the gate for the logic function y=a. (b+c).Part III is to simulate this layout and do its parametric analysis.

2.Theory

2.1 The MOS device

The n-channel MOS is built using polysilicon as the gate material and N+ diffusion to make the source and drain. The p-channel MOS is built using polysilicon as the gate material and P+ diffusion to make the source and drain.

Figure 3.1 NMOS and PMOS Symbols

2.2 The MOS Model

In this lab, the model 3 is employed. For the evaluation of the current **Ids** as a function of **Vd**, **Vg** and **Vs** between Drain and Source, we use the equations on page 16 and 17 in the manual of Micro Wind. The equations are derived from Model 1 and take into account a set of physical limitations in a semi-empirical way.

3. Labs

3.1 Part I: MOS Device Characteristics (30 minutes)

- ***Double click the icon "Micro Wind"*** The Micro Wind window pops up.

- **File>Select foundry>**

An "**Open**" window pops up. Select "**cmos025.rul**" and push "**open**" button.
In this lab, we use 0.25um CMOS technology.

- **Simulate>MOS characteristics>**

A "**MOS Viewer**" window pops up. In this window, we will analyse the characteristics of MOS devices.

Select "**NMOS**" (lower right), "**level3**" (upper right), "**W=10um L=10um**", "**Id vs.Vd**". Push the button "**Draw Curve**", the display window displays several curves.
Find the Ids value with Vgs = 2V and Vds = 2V and fill it into table 1.

Select "**NMOS**" (lower right), "**level3**" (upper right), "**W=0.60um L=10um**", "**Id vs.Vd**". Push the button "**Draw Curve**", the display window displays several curves.
Find the Ids value with Vgs = 2V and Vds = 2V and fill it into table 1.

Select "**NMOS**" (lower right), "**level3**" (upper right), "**W=10um L=0.30um**", "**Id vs.Vd**". Push the button "**Draw Curve**", the display window displays several curves.
Find the Ids value with Vgs = 2V and Vds = 2V and fill it into table 1.

Select "**NMOS**" (lower right), "**level3**" (upper right), "**W=0.60um L=0.30um**", "**Id vs.Vd**". Push the button "**Draw Curve**", the display window displays several curves.
Find the Ids value with Vgs = 2V and Vds = 2V and fill it into table 1.
From this part, we can conclude that:

If W increases (L, Vds, Vgs are fixed), then Ids_____
If L increases (W, Vds, Vgs are fixed), then Ids_____
If Vds increases (W, L, Vgs are fixed), then Ids_____
If Vgs increases (W, L, Vds are fixed), then Ids_____

TABLE 1. NMOS

W	L	Vgs	Vds	Ids
10um	10um	2V	2V	
0.6um	10um	2V	2V	
10um	0.3um	2V	2V	
0.6um	0.3um	2V	2V	

Select "**PMOS**" (lower right), "**level3**" (upper right), "**W=10um L=10um**", "**Id vs.Vd**". Push the button "**Draw Curve**", the display window displays several curves. Find the Ids value with Vgs = -2V and |Vds| = 2V and fill it into table 2.

Select "**PMOS**" (lower right), "**level3**" (upper right), "**W=0.60um L=10um**", "**Id vs.Vd**". Push the button "**Draw Curve**", the display window displays several curves.

Find the Ids value with Vgs = -2V and |Vds| = 2V and fill it into table 2.
Select "**PMOS**" (lower right), "**level3**" (upper right), "**W=10um L=0.30um**", "**Id vs.Vd**". Push the button "**Draw Curve**", the display window displays several curves.

Find the Ids value with Vgs = -2V and |Vds| = 2V and fill it into table 2.
Select "**PMOS**" (lower right), "**level3**" (upper right), "**W=0.60um L=0.30um**", "**Id vs.Vd**". Push the button "**Draw Curve**", the display window displays several curves.

Find the Ids value with Vgs = -2V and |Vds| = 2V and fill it into table 2.
From this part, we can conclude that

if W increases (L, Vds, Vgs are fixed), then |Ids|_____
if L increases (W, Vds, Vgs are fixed), then |Ids|_____
if |Vds| increases (W, L, Vgs are fixed), then |Ids|_____
if |Vgs| increases (W, L, Vds are fixed), then |Ids|_____

TABLE 2. PMOS

W	L	Vgs	Vds	Ids
10um	10um	-2V	2V	
0.6um	10um	-2V	2V	
10um	0.3um	-2V	2V	
0.6um	0.3um	-2V	2V	

Push"OK" to close the simulation window.

3.2 PART II: Layout and Design Rule Checker

Note: If you can finish the layout, please go ahead to PART III and use your layout to do the simulation and parametric analysis in PART III.
Figure 1 gives a schematic of the gate for the logic function y=/a.(b+c).

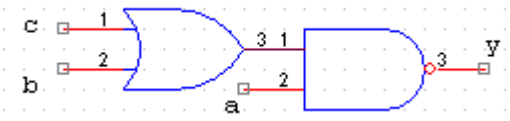


Figure 3.2 Schematics of y=[a.(b+c)]'

According to this schematic please work out your layout.

Begin to draw your layout with Micro wind layout editor.
When you **finish** the layout, please check your layout:

- **Analysis > Design Rule Checker**

If your layout is correct, then no messages will appear. If there are some errors, then the warning messages will appear near the errors. Please modify your layout until no error messages appear. Save your layout.

3.3 Part III: Simulation and Parametric Analysis (1.5 hours)
If your layout has been finished and no errors are found, then go ahead to step 2. If not, follow the step1, which will show you how to generate a layout automatically.

Step1:

Using "cut" command, clear everything in the display window. Make sure that nothing remains in the window. Go to main menu:

- **Compile>Compile one line>**

A "**CMOS Cell Complier**" window pops up. Input "**y=/ (a. (b+c))**", push "**Compile**", a layout of y=a. (b+c) will appear in the window.

- **Analysis>design rule checker**

It is to check the layout. It should be correct. If any error appears, please let the assistant know it. Save the layout.

Step2: Add properties to input signals for simulation

- **click on the clock icon and then click the "a" node on the layout.**

The clock window appears; make sure the properties on the window are below:

Low level 0.0V High level 2.5V Time low40 Rise time0.5 Time high40 Fall time0.5 ns

- Push "**Assign**".

- **click on the clock icon and click the "b" node on the layout.**

The clock window appears; make sure the properties on the window are below:

low level 0.0V High 2.5V Time low 20 Rise time 0.5 Time high 20 Fall time 0.5 ns

- Push "**Assign**".

- **click on the clock icon and click the "c" node on the layout.**

The clock window appears; make sure the properties on the window are below:

low level 0.0V High level 2.5V Time low 10Rise time 0.5 Time high 10Fall time 0.5 ns

- Push "**Assign**".

- **click on the node visible icon and then click the "y" node on the layout.**

The node window appears. Make sure "**visible in simulation**" is active.

- Push "**Assign**".

- **Simulate>Simulate Option**

An **Extraction** window pops up. Select" **Purge and merge**" for "**database**", "**generate a SPICE file after extraction**" for "**Options**".

- Push button "**Models, Parameters**"

Make sure: "**use MOS Model**" is "**Berkeley Spice level 3**" "**power Supply**" is **2.5**V "**Temperature**" is **27.0**

- Push "**Extract**" and then push "**Quit**".

>Simulate>Start Simulate

A **"simulation of example"** window pops up. Select **"Voltage vs. time",** and step is 2 ps,
time scale is 100ns. Check your simulation results. If you are sure that your simulation results are correct, then draw the waves in figure 2**.**

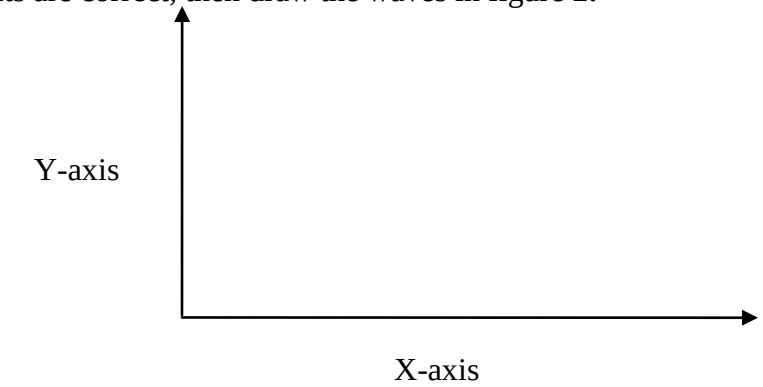


Figure 3.3: Voltage vs. Time

- **Click "Back to Editor" to quit the simulation window.**

>Analysis>Parametric Analysis

Click on the output node, and then a window "**Start Analysis**" pops up. Select "**Power Supply**", the **range** is from 0.0V to 5.0V, the **step** is 1.0V. The "**measurement**" is "**dissipation**", push "**Start analysis**"; curve (dissipation vs. Vdd supply) appears. Click "Large" button to zoom in and draw the curve in figure 3.

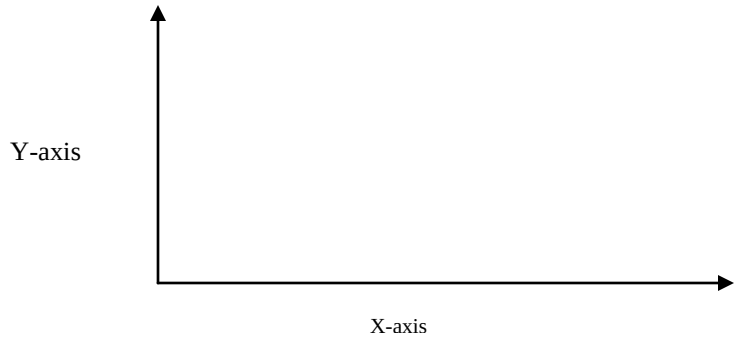


Figure 3.4: Dissipation vs. power supply (Vdd= 0V to 5.0 V)

Select "**Power Supply**", the **range** is from 0.0V to 5.0V, the **step** is 1.0V. The "**measurement**" is "**rise delay**", push "**Start analysis**", a curve (rise delay vs. Vdd supply) appears. Draw the curve in figure 4.

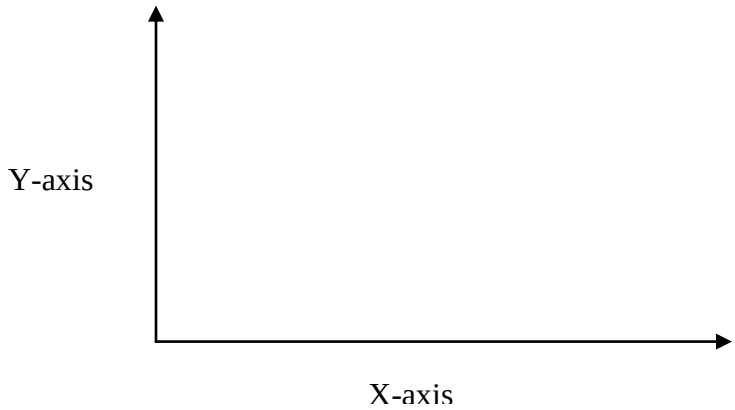


Figure 3.5: Rise Delay & Fall Delay vs. power supply

Select "**Power Supply**", the **range** is from 0.0V to 5.0V, the **step** is 1.0 V. The "**measurement**" is "**fall delay**", push "**Start analysis**", a curve (fall delay vs. Vdd supply) appears. Draw the curve in figure 4.

Select "**Node capacitance**", the **range** is from 0 to 100 fF, the **step** is 20 fF. The "**measurement**" is "**rise delay**", push "**Start analysis**", a curve (rise delay vs. load capacitance) appears. Draw the curve in figure5.

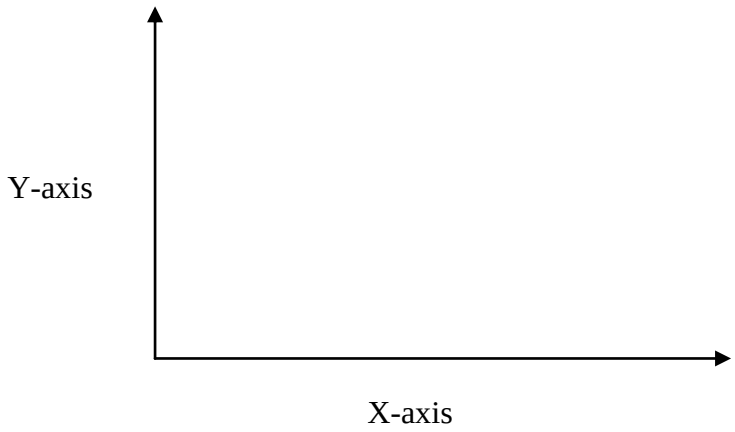


Figure 3.6: Rise Delay & Fall Delay vs. load capacitance

Select "**Node capacitance**", the **range** is from 0 to 100 fF, the **step** is 20 fF. The "**measurement**" is "**fall delay**", push "**Start analysis**", a curve (fall delay vs. load capacitance) appears. Draw the curve in figure5.

Please check all of your results in the figures.

Lab-04 Layout

"M**OSFET Inverter Characteristics and Layout in Micro wind"**

1. Objective

In this lab students will design and implement a CMOS Inverter. Different design Parameter's effects like transistor sizing, supply voltages etc well be analyzed and delay, area, power and currents will be observed. This lab assumed that students are familiar with MicroWind and Lambda based design rules. The tool used in this lab is MicroWind. The goals for this Lab are:

• Design of CMOS Inverter and transistor sizing.
• Layout Design using the tool.
• Gate delay, area, power and current analysis and the effects of transistor sizing on these parameters.

2. Theory

2.1 MOSFET

The Metal Oxide Semiconductor Field Effect Transistor is very important part of Digital Integrated Circuits. It is mostly used as switch in digital design. MOSFET is a four terminal device. The voltage applied to the gate terminal determine the current flow between drain and source terminals. The body/substrate of the transistor is the fourth terminal. Mostly the fourth terminal (body/substrate) of the device is connected to dc supply that is identical for all devices of the same type (GND fro nMOS and Vdd for pMOS). Usually this terminal is not shown on the schematics

2.2 nMOS

The nMOS transistor consists of n+ drain and source diffusion regions, which are embedded in a p-type substrate. The electrons in the channel beneath the gate between source and drain terminal are responsible for the current flow.

2.3 pMOS

The pMOS transistor consists of p+ drain and source diffusion regions, which are embedded in an n-type substrate. The holes in the channel beneath the gate between source and drain terminal are responsible for the current flow.

2.4 CMOS

CMOS Inverter/NOT gate is considered to be the heart of VLSI circuits, based on the understanding of NOT gate we can extend it easily to NAND and NOR gates which are

the basic building blocks of more complex circuits e.g. multipliers and microprocessors. As per discussion and design on white board in the Lab, a NOT gate can be implemented using two FETs i.e. a pFET and an nFET both connected in series, in which Vdd is supplied to pFET and nFET is grounded, input x is applied to the gate terminals of both and the output is obtained at node y.

3. Design/ Diagram/Circuit

Symbol, Truth Table and CMOS circuit of NOT Gate

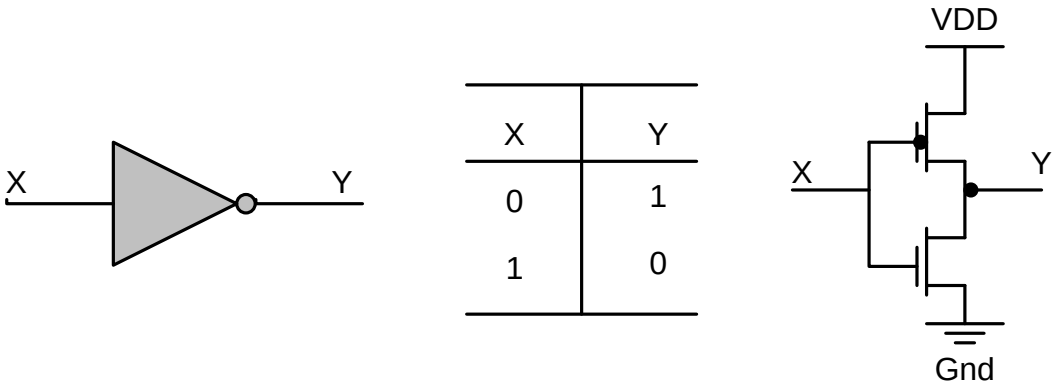


Figure 4.1 Inverter Symbol

4. Lab Instructions

a. Open MicroWind and select the foundry cmos025.
b. Save the design as "Save as" as "Lab04", and save the design frequently during the lab session.
c. Draw the layout of nMOS using MOS Generator
d. Draw the layout of pMOS using MOS Generator by setting the appropriate width of pMOS
e. Connect the two transistors using Medal 1as per diagram.
f. Draw the rails of Vdd and Gnd above and below.
g. Connect the n well with Vdd.
h. Add input and output to your design.
i. Save the layout.
j. Apply design rule checker.
k. Simulate the design using run Command.
l. Analyze configuration delay, gate delay, current, power, and midpoint voltage.
m. Repeat the design for different values

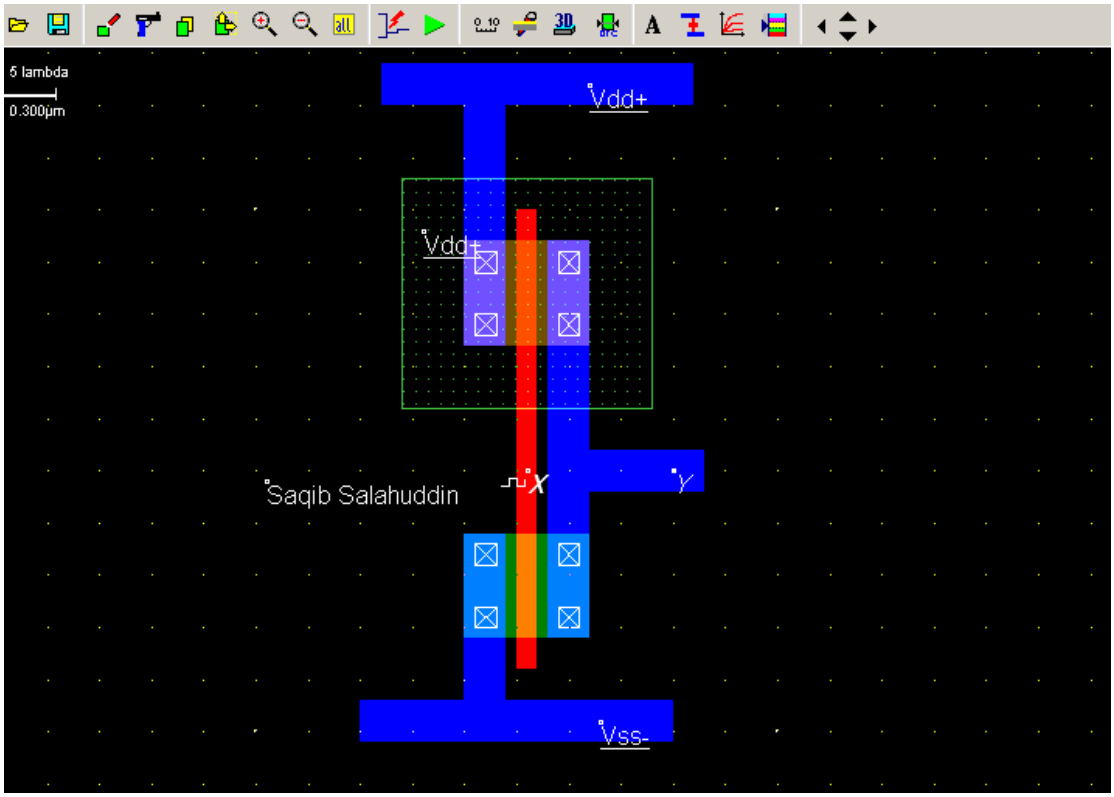


Figure 4.2 Inverter Symbol in MicroWind

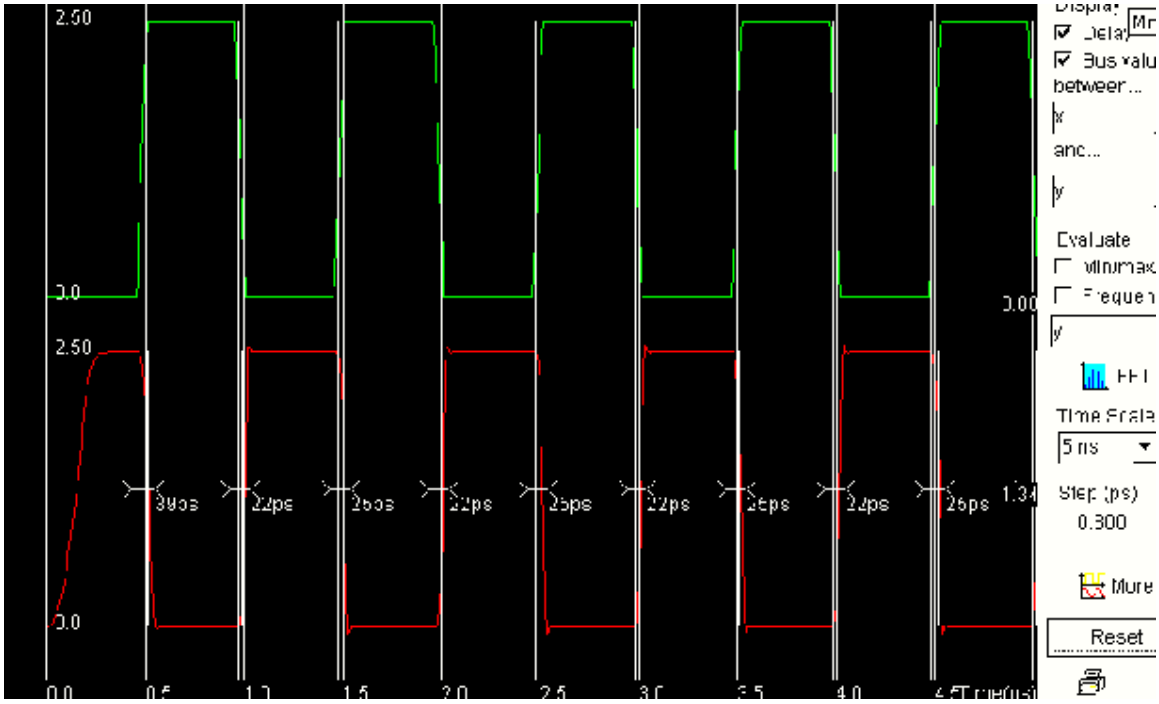


Figure 4.3 Running the Simulation 1

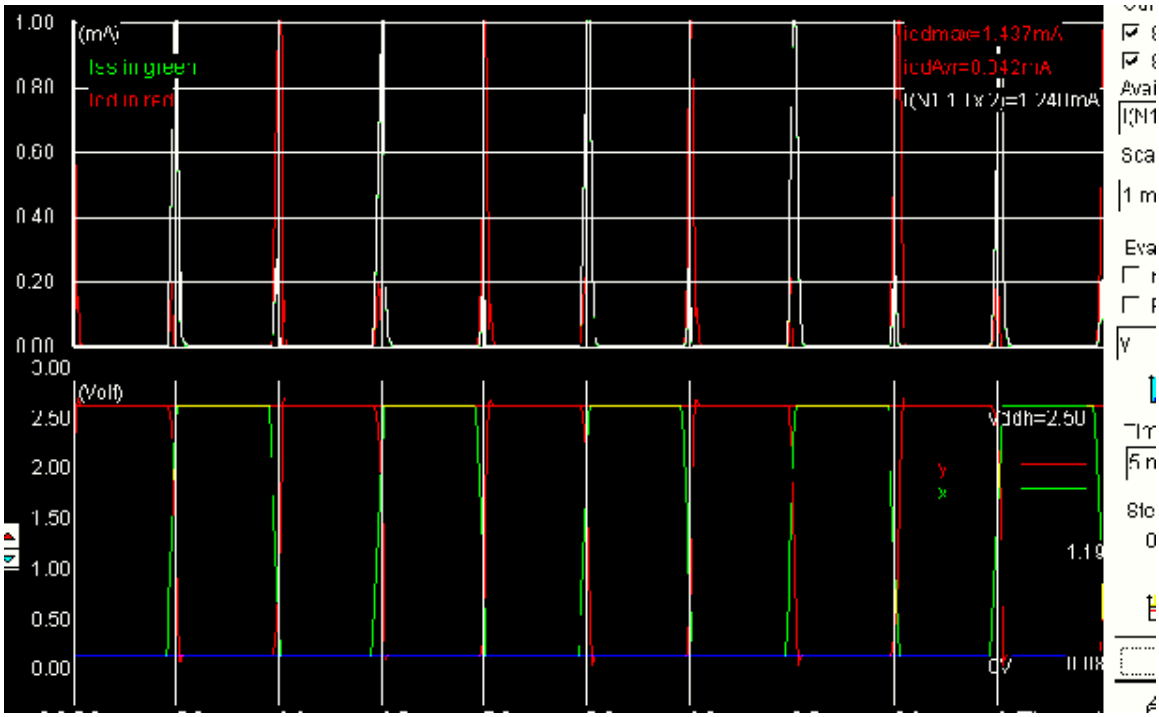


Figure 4.4 Running the Simulation 2

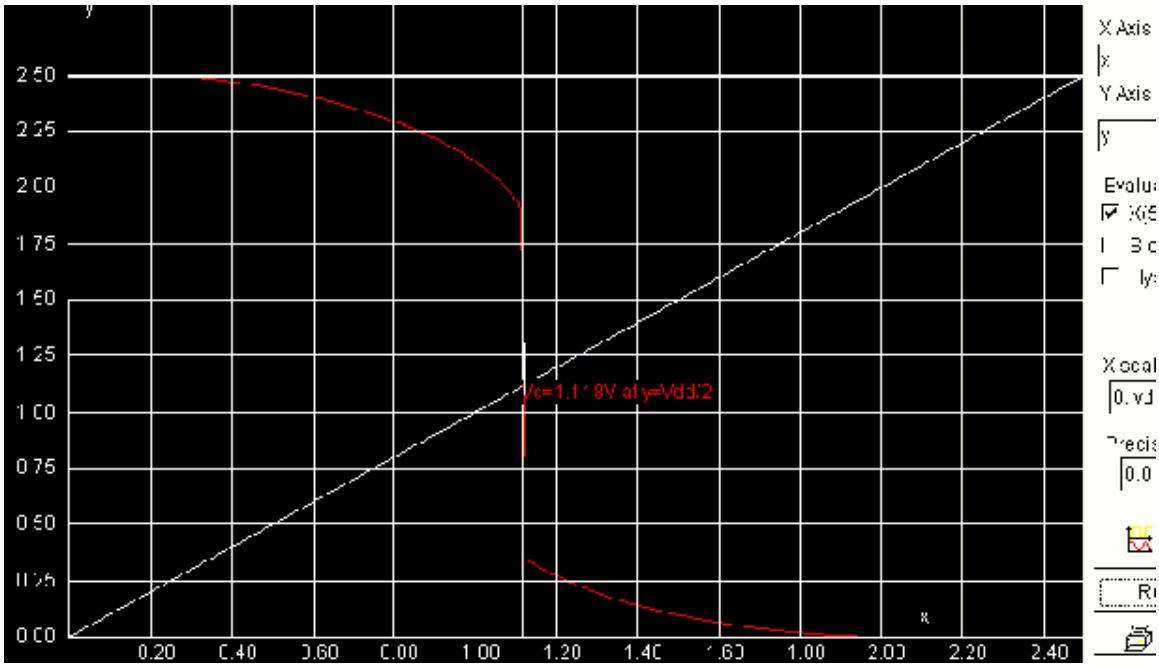


Figure 4.5 Running the Simulation 3

5. Lab Report

- Give a short description of the content of the Lab
- Include block Diagram of your design in the lab report.
- Include your name in the layout for evaluation purpose
- Include the results in timing waveform in your report.
- Only follow the cover page format.

Simulation Anlysis (Include in your Lab Report)

SR. #	**Width of nMOS**	**Drain term voltage**	**Input Voltage Level**	**Propagation Delay**	**Output voltage Level**	**Current I_D**
1	**10 λ**	V_{SS}	**2.5v**			
2			**0.5v**			
3			**1.5v**			
4			**5.5v**			
5		V_{DD}	**2.5v**			
6			**0.5v**			
7			**1.5v**			
8			**5.5v**			
9	**50 λ**	V_{SS}	**2.5v**			
10			**0.5v**			
11			**1.5v**			
12			**5.5v**			
13		V_{DD}	**2.5v**			
14			**0.5v**			
15			**1.5v**			
16			**5.5v**			

You can increase the table and also the entries for in depth analysis.

A properly presented in depth analysis with graph based on the table entries will be highly appreciated.

Discuss the Effects of width design parameter of the MOS devices on their behavior.

Lab-05 Layout

"Layout of Basic Gates using 0.25 micron Technology in Microwind"

1. Objective

In this lab students will design and implement the layouts of different CMOS gates, which includes NAND, NOR. The tool used in this lab is Microwind. The goals for this Lab are:

- Design of CMOS NAND and NOR Gate.
- Layout Design using the tool.
- Gate delay, area, power and current analysis and the effects of transistor sizing on these parameters.

2. Theory

2.1 NAND Gate

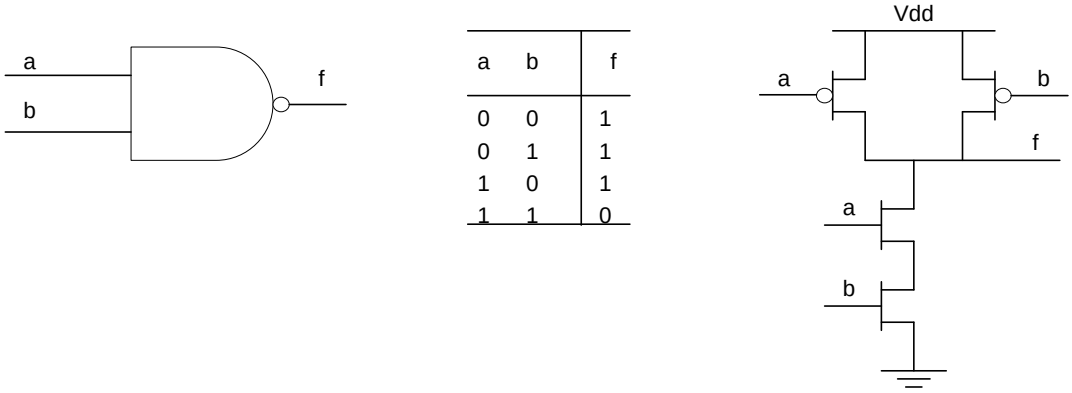
As per discussion and design on white board in the Lab, a NAND gate can be implemented using four FETS i.e. two pFETs and two nFETs as the inputs of the gate is two. pFETs are connected in parallel while nFETs are connected in series, Vdd is supplied to the parallel combination of pFETs while the series combination of nFETs is grounded. Inputs a & b are applied to the gate terminals of all FETs, and the output f is obtained from the common junction of these series and parallel combinations as illustrated in NAND circuit under the heading of **Design Diagram / Circuit.**

2.2 NOR Gate

As per discussion and design on white board in the Lab, A NOR Gate can be implemented using fours FETS i.e two pFETs and two nFETs as the inputs of the gate is two. pFETs are connected in series while nFETs are connected in parallel, Vdd is supplied to the series combination of pFETs while the parallel combination of nFETs is grounded. Inputs a & b are applied to the gate terminals of all FETs, and the output f is obtained from the common junction of these parallel and series combinations as illustrated inNOR circuit under the heading of **Design Diagram/Circuit.**

3. Design Diagram / Circuit

a) Symbol, Truth Table and CMOS circuit of NAND Gate



b) **Symbol, Truth Table and CMOS circuit of NOR Gate**

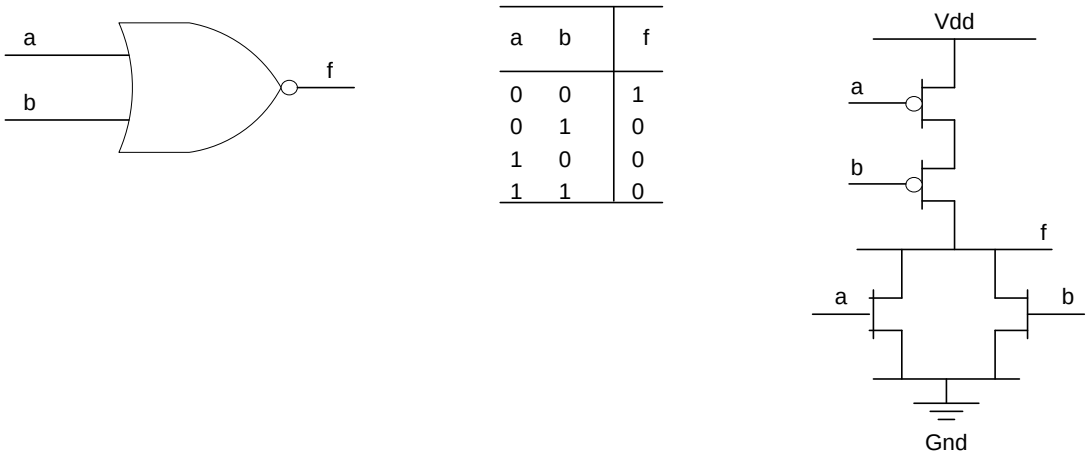


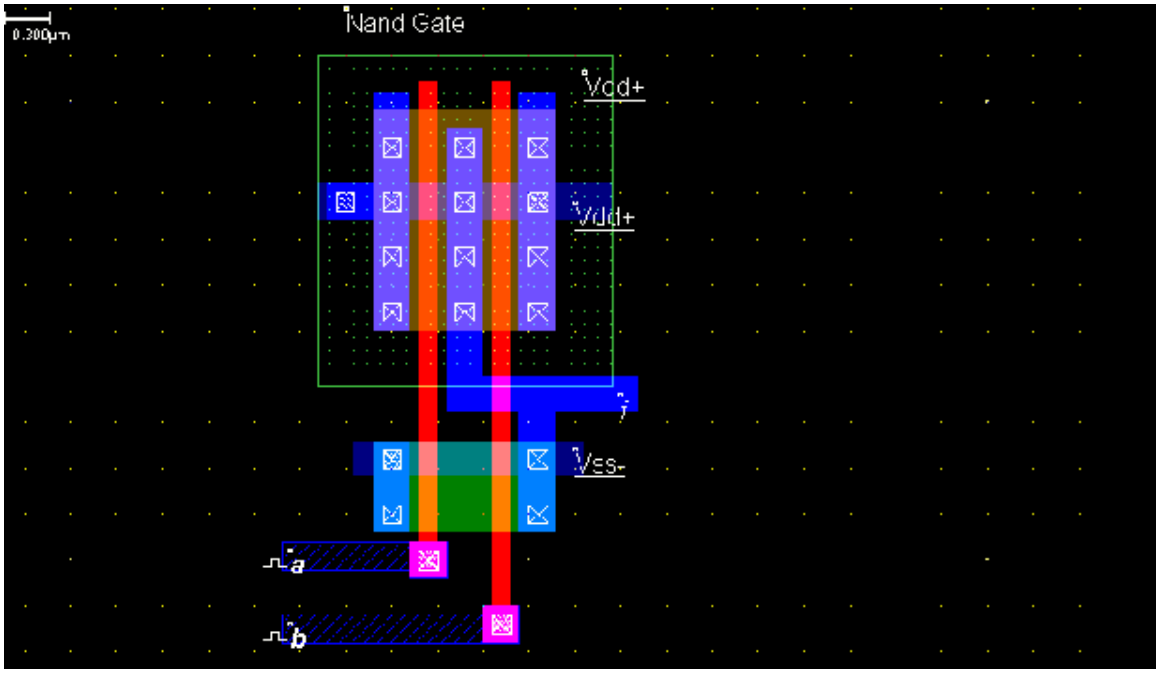
Figure 5.1 NOR Gate, Symbols and Truth Table

4. Lab Instructions

1. Open Microwind and select the foundry cmos025
2. Save the design as "Save as" as "Lab05", and save the design frequently during the lab session.
3. Draw the layout of nMOS using MOS Generator
4. Draw the layout of pMOS using MOS Generator by setting the appropriate width of pMOS
5. Connect the transistors using Metal 1 as per design.
6. Draw the rails of V_{DD} and ground rails above and below.
7. Connect the nWell to V_{DD}
8. Check the design using DRC for any design rule violation and correct the

design in case of error, again run the DRC and check for errors. Or run the DRC after each change in the layout.

9. Check for Electrical connections to be valid.
10. Add inputs and outputs to the design; also add virtual capacitance at the output in your design.
11. Simulate the Design. Observe the values of configuration delay, gate delay, power, current, VTC, and area.
12. Repeat the design using for different values of transistor's dimensions, supply voltages. And observe the changes in configuration delay, gate delay, power, current, VTC, and area carefully. Make a conclusion of your observations.



NAND Gate using Metal 3 for inputs, Metal 2 for Vdd and Gnd, and Metal 1 for diffusion interconnection and CMOS 0.12 micron process

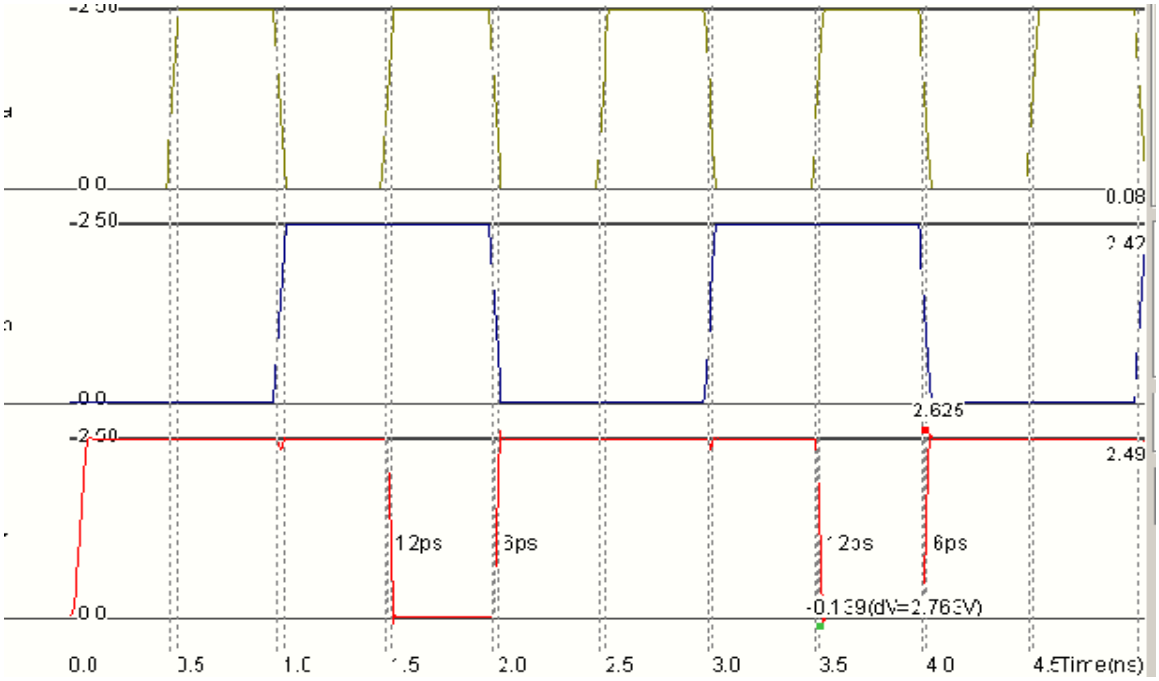


Figure 5.2 Simulating the NOR gate

5. Lab Report

- Give a short description of the contents of the lab
- Include block diagram/diagrams of your design in the lab report
- Describe your layout design approach parameters and explain the effects of each the parameter
- Include layout of your design also add your name on the design for evaluation purpose.
- Include the results in timing waveform format in your report. Only follow the provided cover page format.

6. Simulation Analysis (Include in your Lab Report)

- A properly presented in depth analysis with graph based on the table entries will be highly appreciated.
- Why the Low to High propagation delay is smaller than High to Low propagation delay for NAND gate.
- Why the Low to High propagation delay is larger High to Low propagation delay for NOR gate.
- How for NAND Gate the delay can be made symmetric. Explain, and what will be its effects on power consumption.
- How for Complex Gate the delay can be made symmetric. Explain, and what will be its effects on power consumption.

Lab 06 Layout

"Layout of a Complex gate using 0.25 micron Technology in MicroWind"

1. Objective

In this lab students will design and implement the layouts of a complex CMOS gate. The tool used in this lab is MicroWind. The goals for this Lab are:

- Design of CMOS Complex Gate.
- Layout Design using the tool.
- Gate delay, area, power and current analysis and the effects of transistor sizing on these parameters.

2. Theory

2.1 Complex Gate

The expression for the complex gate is given as under

$$\overline{F = ad + b(cd + a)}$$

As per discussion and design on white board in the Lab, this complex gate can be implemented as under

For pFETs Array

Group1: Two pFETs with inputs "c" & "d" at its gate terminals are connected in parallel.
Group2: A pFET with input "a" at its gate terminal is in series with Group1.
Group3: A pFET with input "b" at its gate terminal is parallel to Group1-Group2.
Group4: Two pFETs with inputs "a" and "d" are in parallel and is connected in series with Group1-Group2-Group3

For nFETs Array

Group1: Two nFETs with inputs "c" & "d" at its gate terminals are connected in series.
Group2: An nFET with input "a" at its gate terminal is in parallel with Group1.
Group3: An nFET with input "b" at its gate terminal is in seies to Group1-Group2.
Group4: Two nFETs with inputs "a" and "d" are in series and is connected in parallel with Group1-Group2-Group3

3. Design Diagram / Circuit

Expression and CMOS circuit of a complex Gate

$\overline{F = ad + b(cd + a)}$

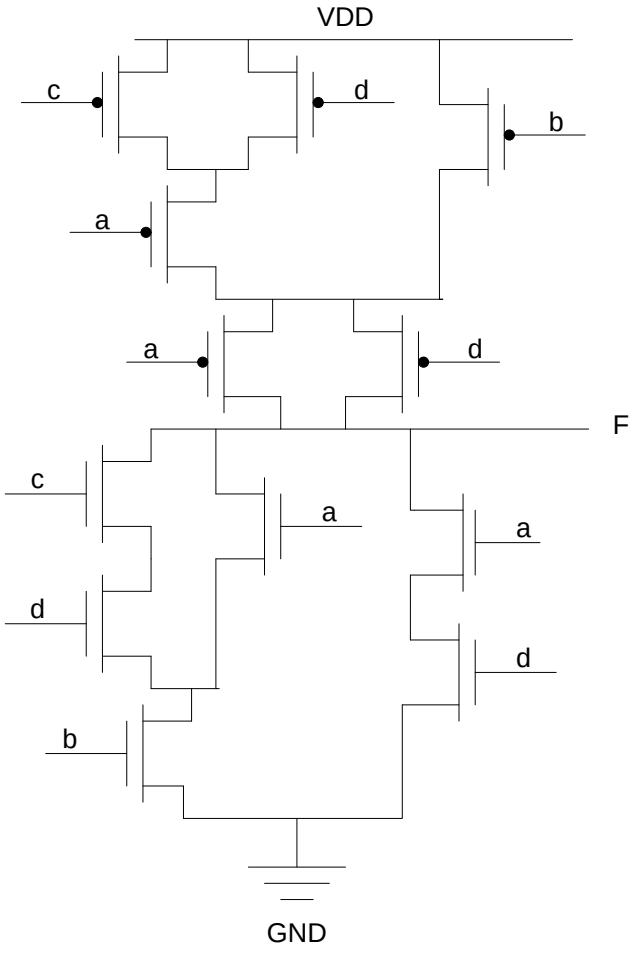


Figure 6.1 Schematics of $\overline{F = ad + b(cd + a)}$

4. Lab Instructions

1. Open MicroWind and select the foundry cmos025
2. Save the design as "Save as" as "Lab04", and save the design frequently during the lab session.
3. Draw the layout of nMOS using MOS Generator
4. Draw the layout of pMOS using MOS Generator by setting the appropriate width of pMOS
5. Connect the transistors using Metal 1 as per design.
6. Draw the rails of V_{DD} and ground rails above and below.
7. Connect the nWell to V_{DD}
8. Check the design using DRC for any design rule violation and correct the

design in case of error, again run the DRC and check for errors. Or run the DRC after each change in the layout.

9. Check for Electrical connections to be valid.
10. Add inputs and outputs to the design; also add virtual capacitance at the output in your design.
11. Simulate the Design. Observe the values of configuration delay, gate delay, power, current, VTC, and area.
12. Repeat the design using for different values of transistor's dimensions, supply voltages. And observe the changes in configuration delay, gate delay, power, current, VTC, and area carefully. Make a conclusion of your observations.

5. Lab Report

- Give a short description of the contents of the lab
- Include block diagram/diagrams of your design in the lab report
- Describe your layout design approach parameters and explain the effects of each the parameter
- Include layout of your design also add your name on the design for evaluation purpose.
- Include the results in timing waveform format in your report
- Only follow the provided cover page format.

6. Simulation Analysis (Include in your Lab Report)

- A properly presented in depth analysis with graph based on the table entries will be highly appreciated.
- How for the Complex Gate the delay can be made symmetric. Explain and what will be its effects on the power consumption.

Lab-07 Layout

"Technology Scaling Effects in CMOS"

1. Introduction

Integrated circuits fabrication technology is continuously improving. As a result, the internal dimensions of semiconductor devices are steadily decreasing. At the same time, the size of circuits that can be fabricated economically continues to increase. Besides increasing the number of devices per IC, scaling has a profound effect on the performance of a circuit in term of speed and power. In this lab, we will demonstrate these effects using the layout edit tool MicroWind (from INSA, Toulouse, France) and circuit simulation tool PSPICE (from MicroSim Corp., USA). This lab will demonstrate the influence of technology scaling on area, delay and power. The first part (section 2) is about **device scaling**. The second part (section 3) is about **interconnect scaling**. **Please note** -- *Some of the formulas used in this lab are explained in detail in Rabaey's book "Digital Integrated Circuits -- A design perspective", as you will see in the following sections.*
Read the related pages carefully before you start the lab.

2. Device Scaling

The following table-1 and table-2 are from Rabaey's book *"Digital Integrated Circuits -- A design perspective"*. They show the ideal scaling effect. This can be used to do scaling effect estimation as is needed later (S is the scaling factor. S>1. U is the voltage scaling factor, and 1<U<S). **Full scaling** means that the supply voltage is scaled in the same ratio as device size. **Fixed-voltage** scaling means that the supply voltage is kept the same while the size is scaled down. **General scaling** means that the supply voltage is scaled while the size is scaled down, but not as much as the size scaling ratio.

Table-7.1 Scaling relation for long-channel devices

Parameter	Relation	Full scaling	General scaling	Fixed-voltage scaling
W, L, t_{ox}		1/S	1/S	1/S
VDD		1/S	1/U	1
Area/Device	WL	$1/S^2$	$1/S^2$	$1/S^2$
C_{ox}	$1/t_{ox}$	S	S	S
C_L	C_{ox} WL	1/S	1/S	1/S
T_p	$C_L V/I_{av}$	1/S	U/S^2	$1/S^2$
P_{av}	$C_L V^2/t_p$	$1/S^2$	S/U^3	S

Table-7.2 Scaling relation for short-channel devices
(the parameters not listed here are the same as in table-1)

Parameter	Relation	Full scaling	General scaling	Fixed-voltage scaling
T_p	$C_L V/I_{av}$	1/S	1/S	1/S
P_{av}	$C_L V^2/t_p$	$1/S^2$	$1/U^2$	1

2.1 Area Scaling

- Open Microwind. Open an example named "ram44". Select the 1.2um foundry. Here, the "1.2um" means the feature size (or minimum gate length of the transistor. *Please note that in 0.35 um foundry and 0.25um foundry the minimum gate lengths are 0.4um and 0.3um respectively. So their scaling factor should be calculated according to 0.4um and 0.3um*). Use 'view->statistics' to get the size of this full-adder. **Fill in the following table**. Then choose foundry of 0.8um, 0.6um, 0.35um and 0.25um, and check their area respectively. Verify that the area scaled as predicted.

Table-7.3 Area scaling effect

	1.2um foundry	0.8um foundry	0.6um foundry	0.35um foundry	0.25um foundry
Process scaling factor -- S (compared with 1.2um)	1	1.5 (1.2um/0.8um)	2 (1.2um/0.6um)	3 (1.2um/0.4um)	4 (1.2um/0.3um)
Estimated area scaling factor -- $1/S^2$ (S is scaling factor)	1	0.4444	0.25	0.1111	0.0625
Area (um^2)					
Real area scaling factor -- $Area/Area_{1.2um}$	1				

2.2 Vertical Size Scaling

It should be mentioned that the device's vertical dimension is not scaled in the same degree as thelateral. Normally the vertical size is less scaled than the lateral size. This is particularly apparent in deep sub-micron process. The following figure shows this phenomenon:

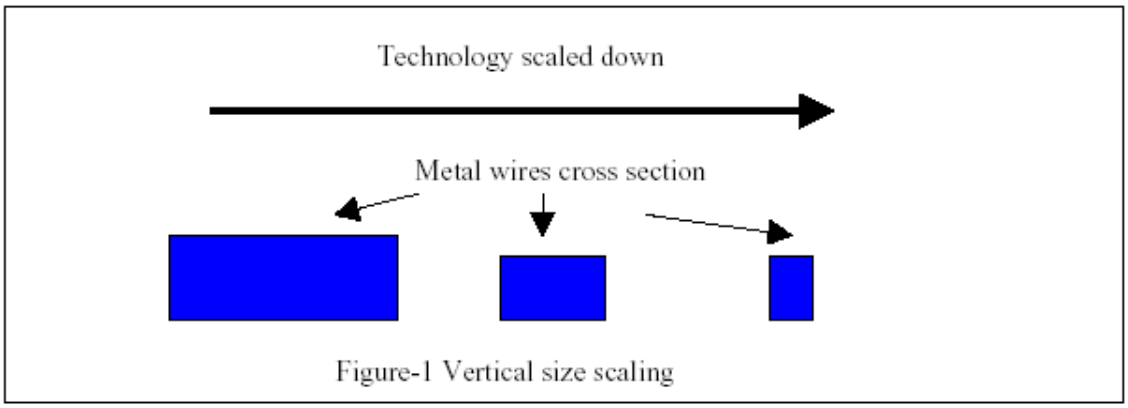


Figure 7.1 Vertical Size Scaling

Open the "ram44" again with 1.2um foundry. Choose any Metal-1 line, use "process view" to check the thickness of metal-1. **Fill in table-4**. Then choose foundry of 0.8um, 0.6um, 0.35um and 0.25um and perform the same checking. Note that with scaled technology, the same metal wire looks 'thicker'.

Table-7.4 Metal-1 thickness

	1.2um	0.8um	0.6um	0.35um	0.25um
Metal-1 thickness (um)					

Question: Why the thickness does not scaled down as the process does?

2.3 Propagation Delay Scaling

2.3.1 **Read page 133-135 of Rabaey's book "Digital Integrated Circuits -- A design perspective",** which is about the first order propagation delay estimation of an inverter.
2.3.2 Estimate the propagation time scaling factor (according to table-1, table-2) from 1.2um to 0.8um, 0.6um, and from 0.6um to 0.35um and 0.25um. **Fill in the related row in table-5**.

Hints: Scaling from 1.2um to 0.8um and 0.6um can be treated as fixed-voltage long channel scaling because the supply voltages for them are all fixed at 5V. From 0.6um to 0.35um and 0.25um can be treated as short channel full scaling because the supply voltage is 5V for 0.6um and 2.5V for 0.35um and 0.25um processes.
2.3.3 Open Microwind. Choose 1.2um foundry, and draw two same inverters as shown in the following diagram (the second one works as the load of the first one). You can find a demonstration of inverter layout by selecting "file->open", and selecting "inv3.msk".

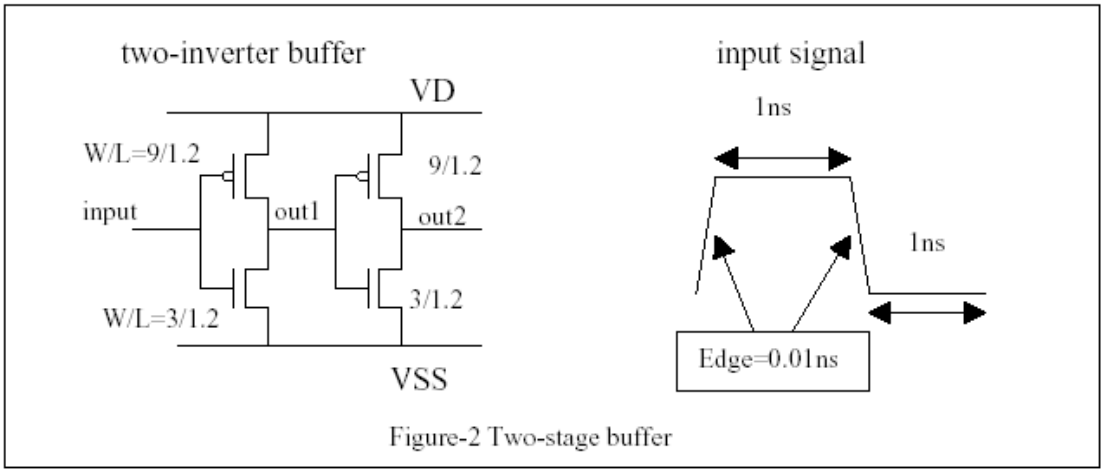


Figure 7.2 Two-stage buffer

- Set the size (width/length) of both NMOS transistors to be **3um/1.2um** and both PMOS transistor to be **9um/1.2um**. Set the input to be a clock signal that has **1ns** high and low periods, **0.01ns** rising and falling edges (the edges are set to be very sharp to measure the propagation accurately). 2.3.4 Simulate the inverters for **20ns** and set the simulation step to be **1ps**.

- Check the propagation time of inverter1 (from ***input*** to ***out1***). The propagation time should be the average of the rising time and falling time. **Fill in table-5**. Save the design with the name "buffer1". 2.3.5 Open the design "buffer1" with foundry of 0.8, 0.6, 0.35 and 0.25um respectively. Repeat step 2.3.4.

Table-7.5 Propagation Delay

	1.2um	0.8um	0.6um
Process scaling factor – S (compare with 1.2um)	1	1.5	2
Your estimation of t_p scaling factor	1		
Measured propagation time -- t_p (ps)			
t_p / $t_{p\text{-}1.2um}$ (propagation delay scaling factor)	1		

	0.6um	0.35um	0.25um
Process scaling factor – S (compare with 0.6um)	1	1.5	2
Your estimation of t_p scaling factor	1		
Measured propagation time -- t_p (ps)			
t_p / $t_{p\text{-}0.6um}$ (propagation scaling factor)	1		

Question: Does the measured propagation scaling factor meets your estimation? It should be mentioned that the deviation could be quite large. There are two main reasons causing this deviation. The first is that many assumptions used to deduce the relationships in table-1, table-2 do not hold in practice. The second is that the process parameters are different from company to company, so the scaling factor can also be quite different in different companies.

2.4 Power Scaling

2.4.1 Estimate the power scaling effect of this "buffer1" with hints in 2.3.2. **Fill in the related row in table-6**. ***Please note that here the power consumption means the power at their highest working frequency. To evaluate this highest-frequency-power, we first use a fixed frequencyinput, measure the power at this frequency, and divide this power by the propagation delay to normalize it. This is the same method as used in Table-1 to deduce the relation of Pav: CLV2/tp.***

2.4.2 Choose different foundry (1.2um, 0.8um, 0.6um, 0.35um and 0.25um) and simulate 'buffer1' in Microwind. **Note that the input signal frequency is kept the same (set in section 2.3.3) in different foundry simulations**. In the simulation waveform window, choose "voltages and currents". There's average power consumption estimated. We should compare the relative values of power to see the scaling effect. Set the simulation time to 20ns (simulate 10 cycles to average the power consumption).

Please note – *The power consumption P_1 you get by this way corresponds to a specific working frequency -- frequency of 'input'. Power consumption of CMOS circuit is directly proportional to the working frequency. To evaluate the power consumption of their respective highest work frequency, we should divide this P_1 by the propagation time t_p (measured in section 2.3 in table-5) to normalize them. So, $P_N=P_1/t_p$. Here, P_N is not a real physical value. It is just a relative number.*

Table-7.6 Power scaling factor

	1.2um	0.8um	0.6um
Process scaling factor – S (compared with 1.2um)	1	1.5	2
Your estimation of power scaling factor	1		
P_1(uW)			
t_p (ps) (from table-5)			
$P_N = P_1/t_p$ (normalized power)			
$P_N / P_{N\text{-}1.2um}$ (power scaling factor)	1		

	0.6um	0.35um	0.25um
Scaling factor – S (compared with 0.6um)	1	1.5	2
your estimation of power scaling factor	1		
P_1(uW)			
t_p (ps) (from table-5)			
$P_N = P_1/t_p$ (normalized power)			
$P_N / P_{N\text{-}0.6um}$ (power scaling factor)	1		

3. Interconnect Scaling

Interconnect scaling has a different aspect as device scaling. If a metal wire is scaled the same ratio in all dimensions, the resistance of the wire will increase to S*R and the capacitance will reduce to C/S. So the RC delay should keep the same. But the real case is that there're a lot of long wires connecting internal modules, so the average wire length will not scaled down as the process does. If a wire is not scaled in length but scaled in the other two dimensions, the RC delay will be S2 times larger (as shown in figure-3)! This causes the phenomenon: **the wire delay becomes larger than the gate delay**.

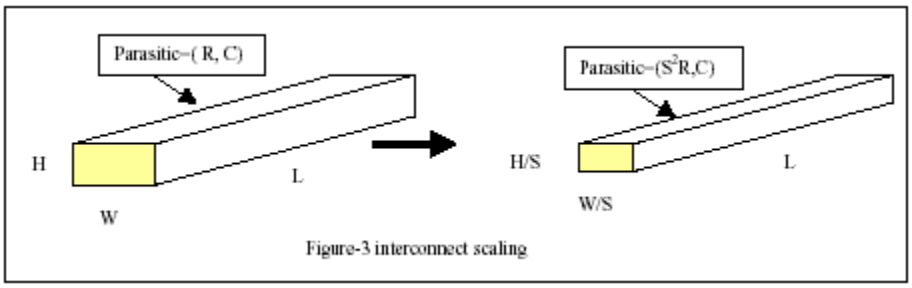


Figure 6.3 Interconnect Scaling

Design Target: ***Design a two-stage buffer to drive a 1cm metal-1 wire (5-lambda width) using 1.2um foundry and 0.25 um foundry respectively. Compare the gate delay and the wire delay in both cases.***

Read page 449 of Rabaey's book "Digital Integrated Circuits -- A design perspective", which is about how to calculate the optimal size of a two-stage buffer. ***The formulas used***

in this section are from this page. Read this page carefully and understand how the formulas are induced.

3.1.1 First, design the buffer in 1.2 um foundry according to step 3.1.2 to 3.1.7 and **fill in table-7**.

3.1.2 The first inverter of the two-stage buffer has a minimum size of 9um/1.2um for PMOS and 3um/1.2um for NMOS (these values are for 1.2um foundry. When open this design again in other foundries, the size will be scaled automatically, but the ratio will stay the same). It is the same size as the one we have designed in section 2.3. So we can open this design directly in 1.2um foundry. Double click the input gate of the first inverter. Find the input capacitance **Ci** and **fill in table-7**.

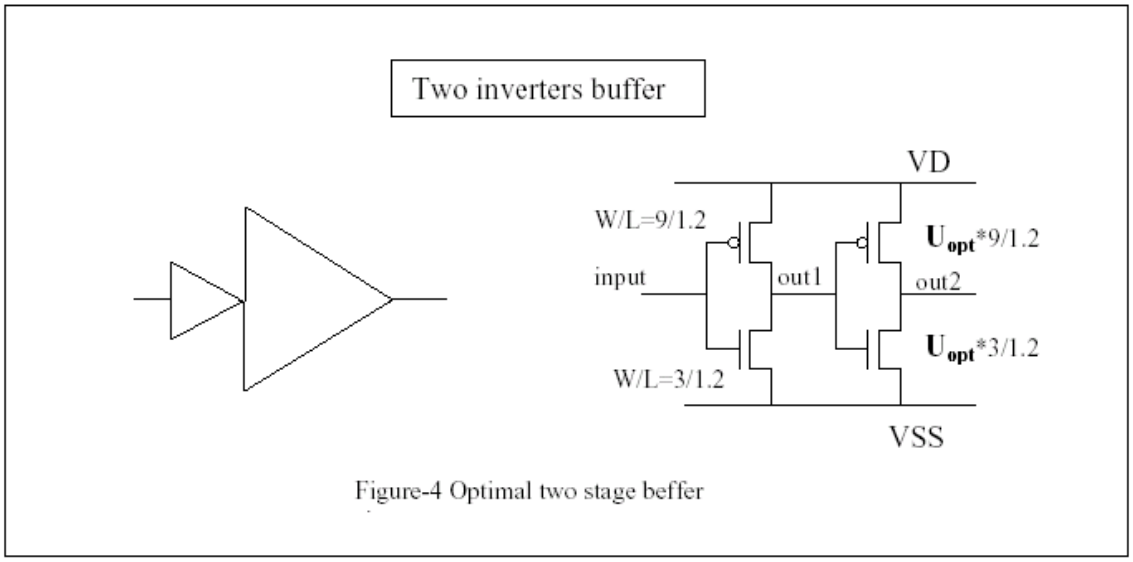


Figure 7.4 Optimal two stage buffer

3.1.3 Calculate the resistance and capacitance of the metal-1 wire using Microwind. It is not easyto draw a 1cm long metal wire. So, in Microwind, draw a 5-lambda width, 500um long metal-1 wire. Double click it to get the capacitance and resistance. Multiply these values by 20 to get the respective **Rwire** and **Cwire** for 1cm wire. **Fill them in tabel-7**.

3.1.4 The RC delay of the wire is **tp,wire** = 0.38*Rwire*Cwire (**Fill it in table-7**)

3.1.5 Calculate the optimal size of the second inverter and **fill it in table-7.7**:
Uopt = X1/2 where Uopt = the optimal ratio of the second inverter size to the first one.
X = Cwire/Ci

3.1.6 The propagation delay of the optimal buffer is **tp,buffer** = 2*tp*Uopt. **Fill it in tabel-7.7**. Here tp is the propagation delay of the minimum size inverter with a load of same size. This value was measured in section 2.3.

3.1.7 Compare the delay of wire and buffer: $t_{p,wire}$ and $t_{p,buffer}$. Which one is larger?

3.1.8 **Repeat step 3.1.2 to 3.1.7 with 0.25um foundry**. Compare the delays again. Which one is larger now? Why?

Table-7.7 Propagation delay in wire and buffer

	R_{wire}(ohm)	C_{wire} (pF)	$t_{p,wire}$ (ns)	C_i (fF)	U_{opt}	$t_{p,buffer}$ (ns)
1.2um						
0.25um						

3.2 Verification of the design in PSPICE (only for 0.25um)

3.2.1 Select 0.25 um foundry in Microwind. Draw the optimal two-stage buffer and generate a SPICE netlist (you have to draw the layout to get the parasitic parameters such as input/output capacitance). 3.2.2 In the generated netlist, manually add the resistance and capacitance to be the load of the buffer. You have to do this manually because Microwind can not handle wire parasitic parameters. In a real wire, the resistance and capacitance are distributed rather than lumped. Therefore, we use the following model to simulate the real case.

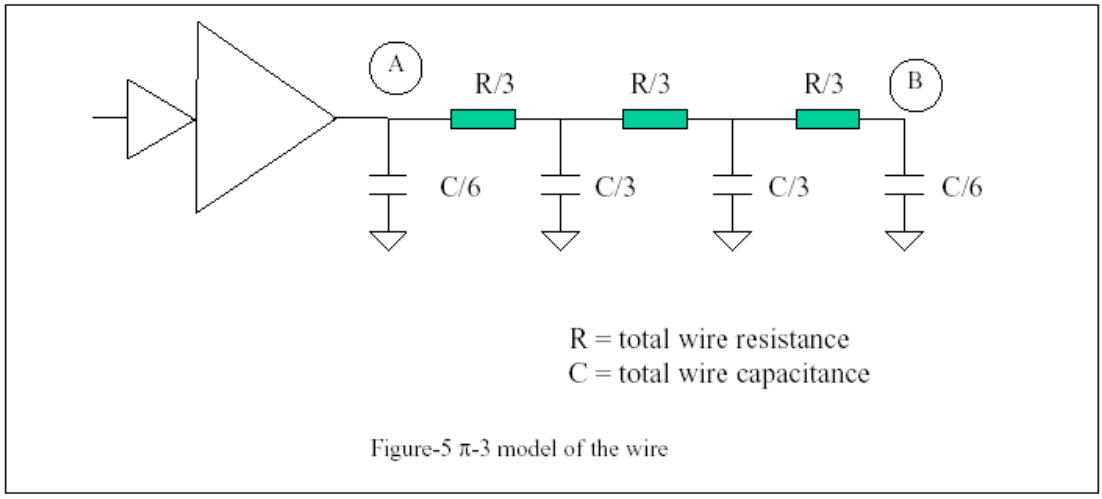


Figure-5 π-3 model of the wire

Figure 7.5: π-3 Model of the wire

3.2.3 Simulate this netlist in PSPICE. Check the waveform and propagation delay of the bufferand the RC wire (from point A to point B in Figure-5). Do they meet your calculation in section 3.1?

Table-7.8 Wire and buffer delay in 0.25um process

	Wire delay (ns)	Buffer delay (ns)
Your calculation		
Measured value		

Hints: *You might find that the buffer has a much smaller propagation time than you have calculated. One of the reasons is that the calculation of* U_{opt} *assumes the capacitance to be lumped, while in real case (and in this simulation) the capacitance is distributed. This will overestimate* U_{opt}.

Lab 08 Layout

"Design and Implementation of Full Adder at Layout Level in Microwind"

1. Objective

In this lab students will design and implement the layout of a CMOS Full Adder. Delay, area, power and currents of Full Adder will be observed. This lab assumed that students are familiar with Microwind and Lambda based design rules. The tool used in this lab is Microwind. The goals for this Lab are:

- Design of CMOS Full Adder Layout.
- Layout Design using the tool.
- Gate delay, area, power and current analysis

2. Theory

CMOS Full Adder

A Full Adder is an important building bock of arithmetic circuits in a system. Full adder accepts three inputs and produces the outputs sum and carry by adding the binary bits with the help of logic gates. The Full Adder can be optimized using XOR and NAND Gates only in the following way.

3. Design Diagram / Circuit

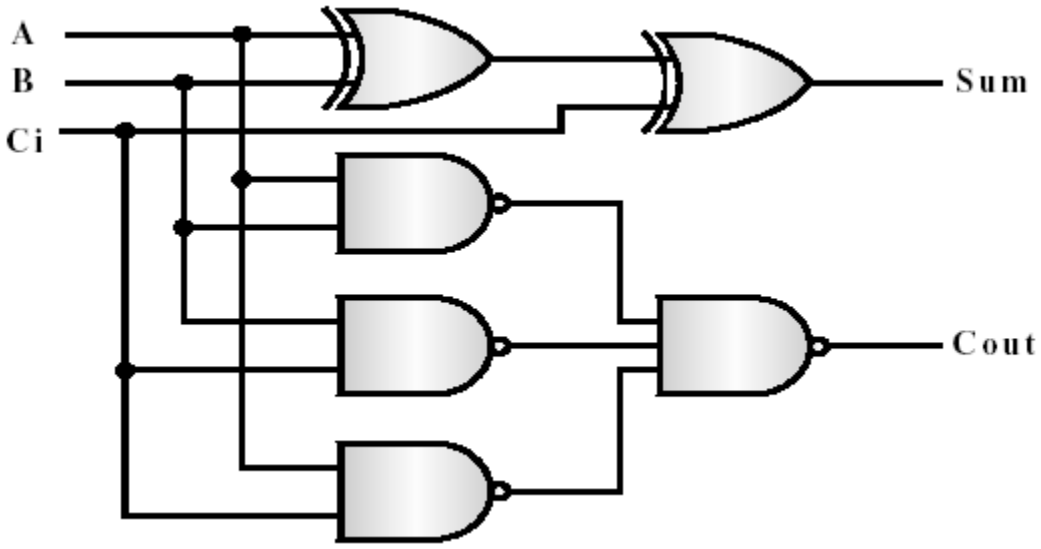


Figure 8.1: Full Adder Gate Level Diagram

Schematic and Layout of the NAND Gate has been done in one or more of the previous labs. There are many ways to construct the XOR schematic e.g. using expression, using Transmission gate. We will construct the schematic in the following way. From the table of XOR Gate.

Inputs		Outputs
A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Table 6.1: Truth Table for XOR

The XOR can be read from the Table as follows: IF B=0, OUT=A, IF B=1, OUT=Inv (A). The principle of the circuit presented below is to enable the A signal to flow to node W1 if ***B=1 and to enable Inv (A) the signal to flow to node W1 if B=0. The output inverts the node W1 so that we can get the XOR operator.***

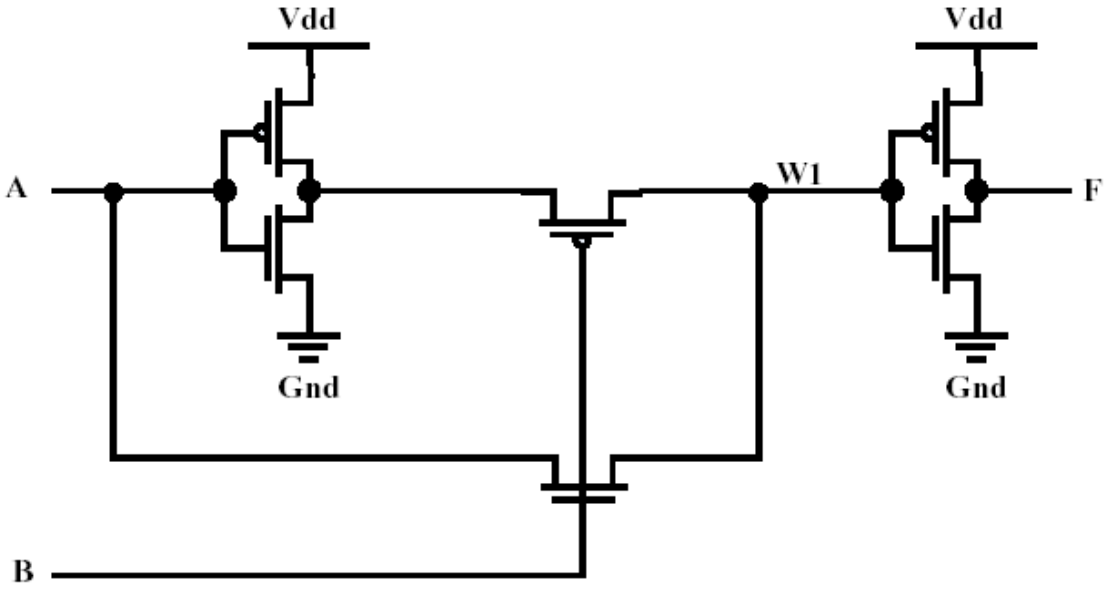


Figure: Schematic of XOR Gate

Figure 8.2: Schematic of XOR Gate

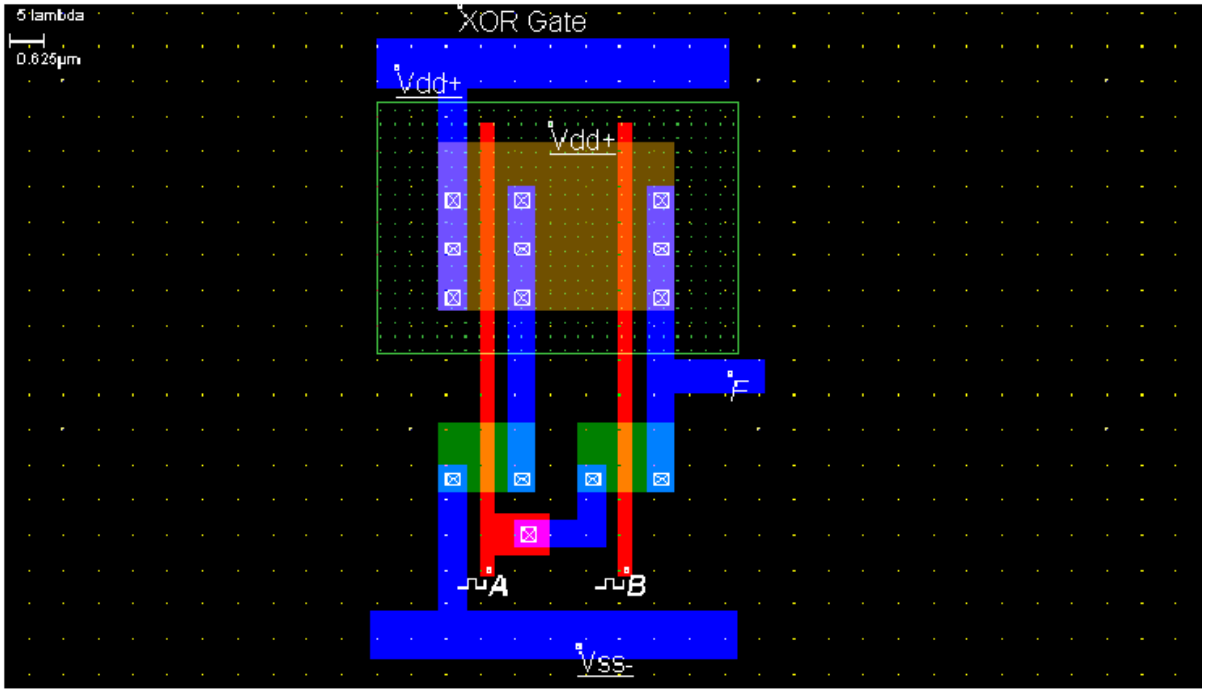


Figure 8.3 : Layout of XOR Gate

4. Lab Instructions

1. Open Microwind and select the foundry cmos025
2. Save the design as "Save as" as "Lab05", and save the design frequently during the lab session.
3. Draw the layout of nMOS using MOS Generator
4. Draw the layout of pMOS using MOS Generator by setting the appropriate width of pMOS
5. Connect the transistors using Metal 1 as per design.
6. Draw the rails of V_{DD} and ground rails above and below.
7. Connect the nWell to V_{DD}
8. Check the design using DRC for any design rule violation and correct the design in case of error, again run the DRC and check for errors. Or run the DRC after each change in the layout.
9. Check for Electrical connections to be valid.
10. Add inputs and outputs to the design; also add virtual capacitance at the output in your design.
11. Simulate the Design. Observe the values of configuration delay, gate delay, power, current, VTC, and area.
12. Repeat the design using for different values of transistor's dimensions, supply voltages. And observe the changes in configuration delay, gate delay, power, current, VTC, and area carefully. Make a conclusion of your observations.

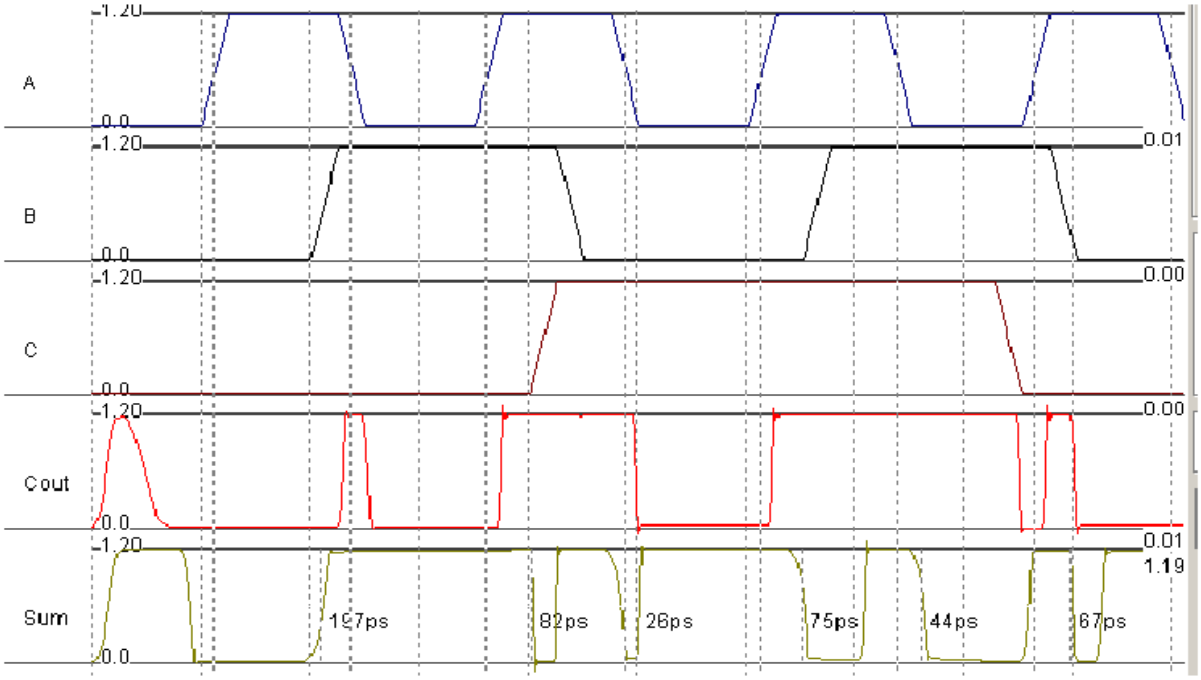


Figure 8.4: Simulating Full Adder

5. Lab Report

- Give a short description of the contents of the lab
- Include block diagram/diagrams of your design in the lab report
- Describe your layout design approach parameters and explain the effects of each the parameter
- Include layout of your design also add your name on the design for evaluation purpose.
- Include the results in timing waveform format in your report
- Only follow the provided cover page format.

Lab-09 Layout

"Design and Implementation of Static RAM Cell Layout using CMOS 0.12 micron Technology in Microwind"

1. Objective

In this lab students will design and implement the layout of a six-transistor static memory cell.
The technology for this lab is cmos012 micron process. The tool used in this lab is Microwind.
The goals for this Lab are:

- Design of Static RAM Cell schematic.
- Design of Static RAM Cell Layout using the tool.
- Gate delay, area, power and current analysis.

2. Theory (Static RAM Cell)

The basic cell for static memory design is based on 6 transistors, with two pass gates instead of one. The circuit consists of two cross coupled inverters, but uses two pass transistors instead of one. The cell has been designed to be duplicated in X and Y in order to create a large arrays of cells. Usual sizes of Megabit SRAM memories are 256 columns x 256 rows or higher. The selection lines WL concern all the cells of one row. The bit lines BL and ~BL concerns all the cells of one column.

3. Design/ Diagram/Circuit

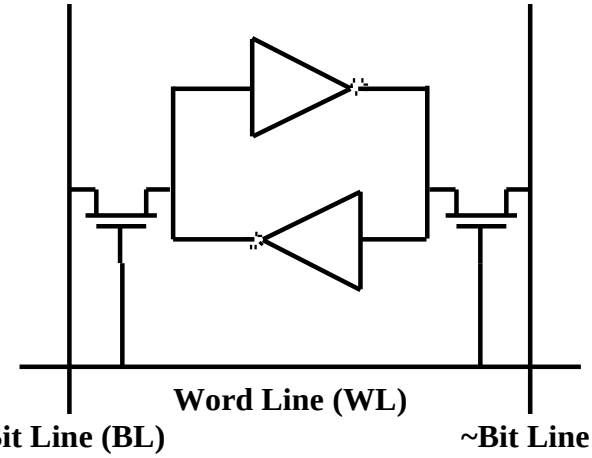


Figure 9.1 : Static Memory Cell

4. Lab Instructions

4.1. Manual Layout of Static memory Cell

1. Open Microwind and select the foundry cmos012.
2. Save the design as "Save as" as "Lab08", and save the design frequently during the lab session.
3. Draw the layout of nMOS using MOS Generator
4. Draw the layout of pMOS using MOS Generator by setting the appropriate width of pMOS
5. Connect the transistors using Metal 1 as per design.
6. Draw the rails of V_{DD} and ground horizontal rails with Metal 3.
7. Draw the layout of signals BL and ~BL using Metal 2.
8. Connect the nWell to V_{DD}
9. Check the design using DRC for any design rule violation and correct the design in case of error, again run the DRC and check for errors. Or run the DRC after each change in the layout.
10. Check for Electrical connections to be valid.
11. Add inputs and outputs to the design; also add virtual capacitance at the output in your design.
12. Simulate the Design. Observe the values of configuration delay, gate delay, power, current, VTC, and area.
13. Repeat the design using for different values of transistor's dimensions, supply voltages. And observe the changes in configuration delay, gate delay, power, current, TC, and area carefully. Make a conclusion of your observations.

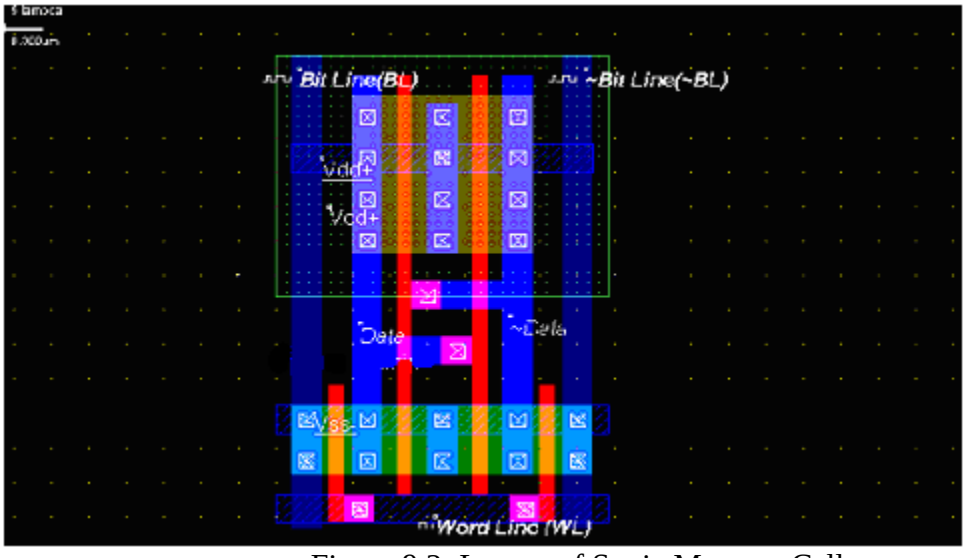


Figure 9.2: Layout of Static Memory Cell

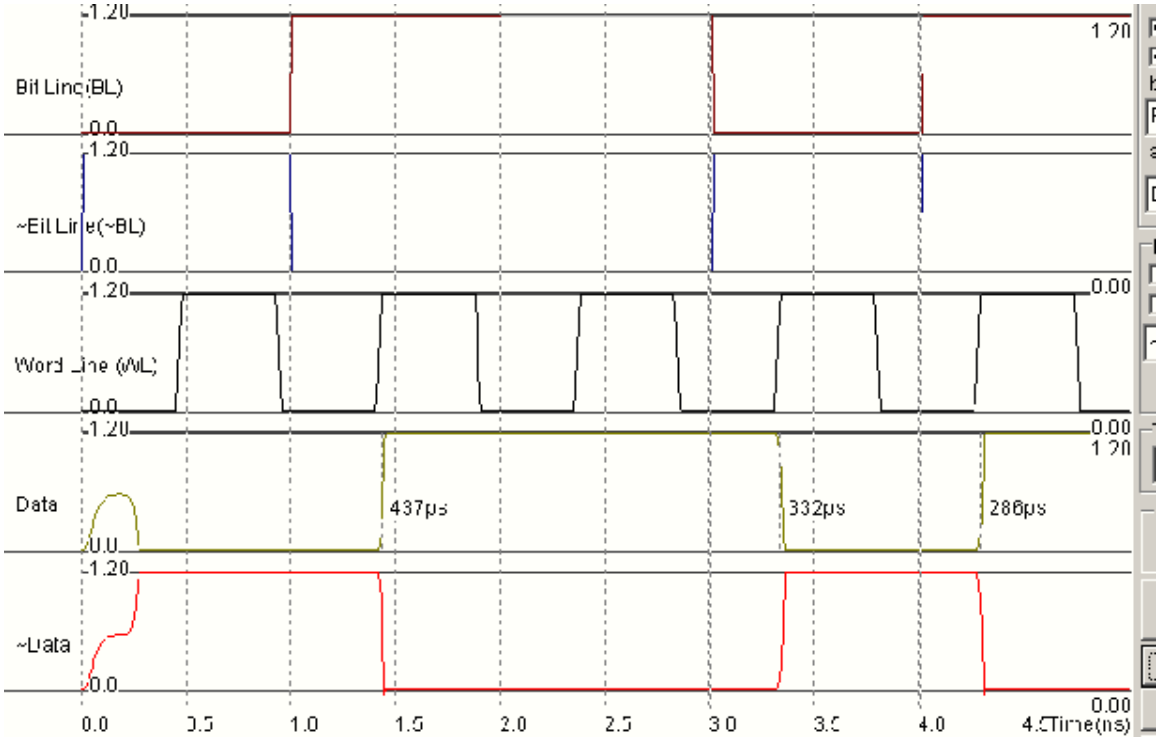


Figure 9.3: Write Cycle of Static Memory Cell

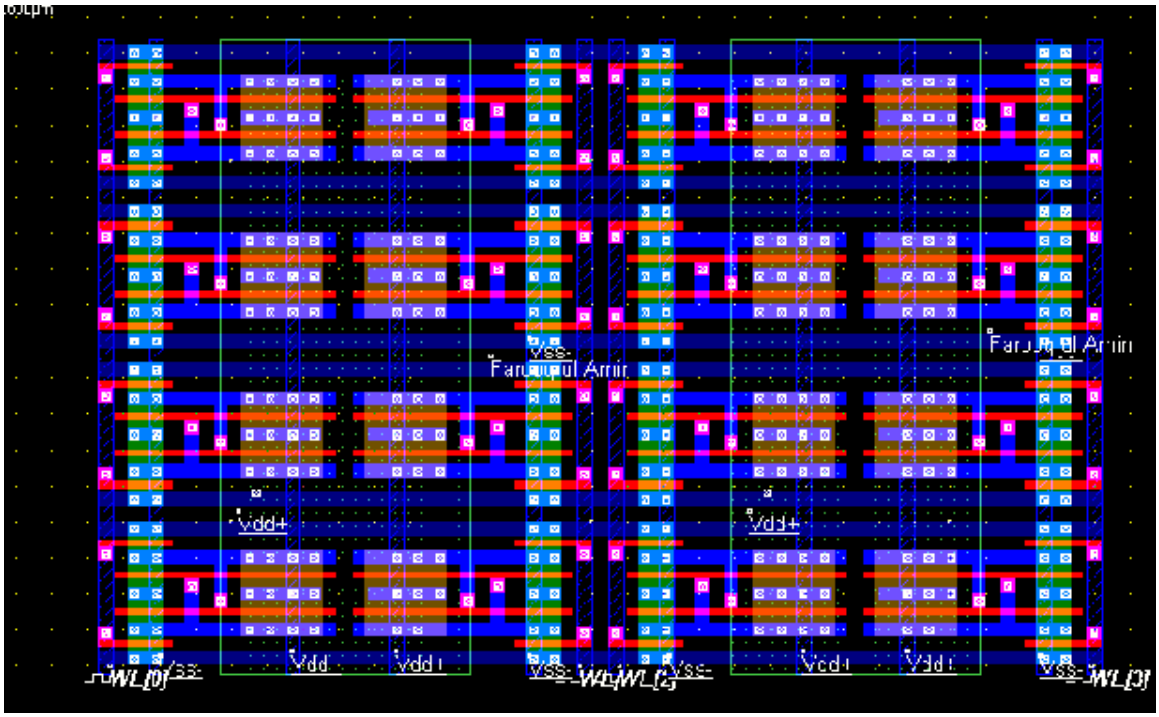


Figure 9.4: Layout of 4x4 Array of Static RAM Cells

5. Lab Report

- Give a short description of the contents of the lab
- Include block diagram/diagrams of your design in the lab report
- Describe your layout design approach parameters and explain the effects of each the parameter
- Include layout of your design also add your name on the design for evaluation purpose.
- Include the results in timing waveform format in your report
- Only follow the provided cover page format.
- You can implement using any other optimized method.

Lab-10 Layout

"Design and Implementation of Look Up Table (LUT) Layout using CMOS 0.25 micron Technology in Microwind"

1. Objective

In this lab students will design and implement the layout of 4 bit 2-input Look UP Table (LUT).
The technology for this lab is cmos025 micron process. The tool used in this lab is Microwind.
The goals for this Lab are:

- Design of Static D Register schematic.
- Design of Static D Register Layout using the tool.
- Design of 4-to-1 Multiplexer using Pass Transistor Logic
- Gate delay, area, power and current analysis.

2. Theory (Static RAM Cell)

Look-up Tables are almost invariably used in FPGAs to produce combinational functions. A look-up table is composed of permanent memory (RAM) connected to a multiplexer. All possible outputs of the desired function are stored in the ram cells. The inputs of combinational function are the inputs of multiple
For this, students will design a 2-input look-up table. For that we will need 4 D-Registers and one 4-1 multiplexer with 02 select inputs.
A 4-1Multiplexer can be constructed using pass transistor logic as shown in the Diagram section.

3. Design/ Diagram/Circuit

4. Lab Instructions

- Open Microwind and select the foundry cmos012.
- Save the design as "Save as" as "Lab09", and save the design frequently during the lab session.
- Draw the layout of nMOS using MOS Generator
- Draw the layout of pMOS using MOS Generator by setting the appropriate width of pMOS
- You are free to choose automated or manual method for your latch design.
- For your convenience schematics and Layout of D-latch and MUX are already copied in your Labs folder. You are free to use them. But they are not optimized and may not give the best results. So you are strongly advised to design your own optimized layout.

- Connect the transistors using **Metal 1** as per design.
- Draw the rails of V_{DD} and ground horizontal rails with **Metal 3**
- Connect the nWell to V_{DD}
- Check the design using DRC for any design rule violation and correct the design in case of error, again run the DRC and check for errors. Or run the DRC after each change in the layout.
- Check for Electrical connections to be valid.
- Add inputs and outputs to the design; also add virtual capacitance at the output in your design.
- Simulate the Design. Observe the values of configuration delay, gate delay, power, current, VTC, and area.
- First stage of simulation will be configuration where you will give a positive pulse to the clock of your RAM cells and load them with four possible combinations of output.
- Second stage will be simulation of you configured XOR function where you will give different combinations to sel inputs and observe the output.

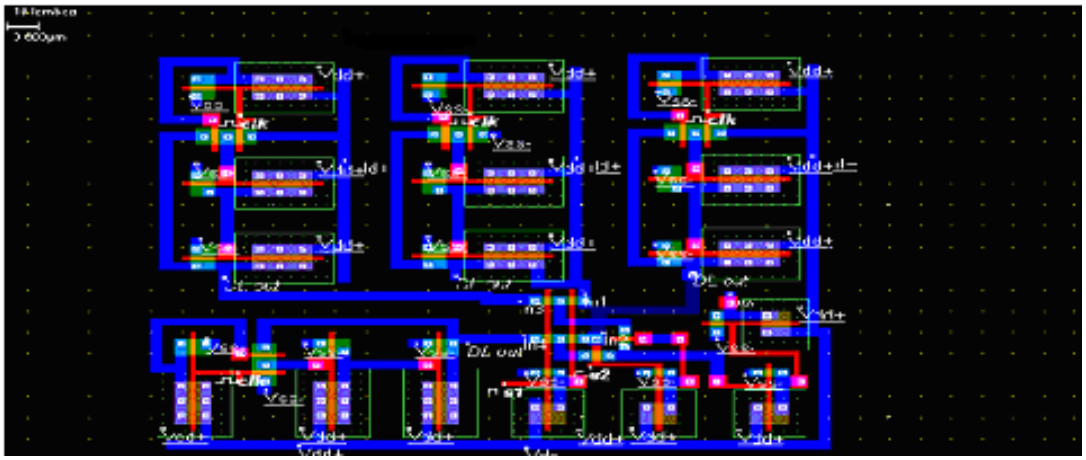


Figure 10.1: Layout of 4-bit 2-Input Look UP Table (LUT)

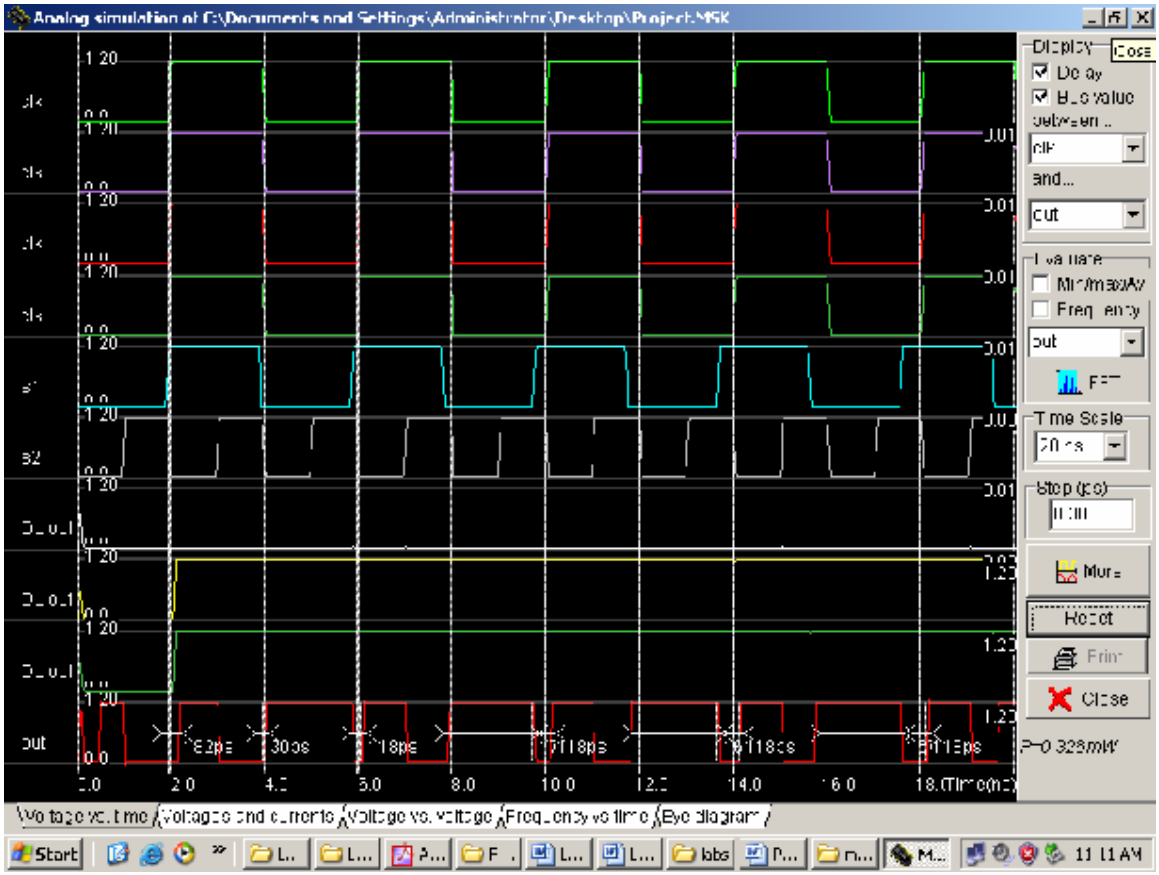


Figure 10.2: Simulation of 4-bit 2-Input Look UP Table (LUT)

5. Lab Report

- Give a short description of the contents of the lab
- Include block diagram/diagrams of your design in the lab report
- Describe your layout design approach parameters and explain the effects of each the parameter
- Include layout of your design also add your name on the design for evaluation purpose.
- Include the results in timing waveform format in your report
- Only follow the provided cover page format.
- You can implement using any other optimized method.
- How much RAM and what kind of MUX will you need to construct an LUT that has 05-inputs

Lab-11

"Chip Design"

1. Objectives

This lab consists of designing a complete chip with a buffered 8-bit barrel shifter on layout level (Lab4) and I/O pads. I/O pad ring consists of 32 pads which are 16 input pads, 8 output pads, 4 V_{DD} pads and 4 V_{SS} pads, respectively. Each basic bonding pad size is 100 x 100 μm^2.

Note: This lab is VERY time consuming and the only way to be able to finish the lab in time is to prepare the homework tasks carefully.

2. Pre-study and preparation

You must carefully read Appendix to know how the chip can be generated. Meanwhile you must finish the lab3 since the lab4 is based on the lab3. If you have not finished the lab3, don't do this lab4 first.

3. MicroWind manual

Study the MicroWind manual thoroughly, so that you are very familiar with the features of the programme.

4. Lab instructions

Select the cmos0.25 foundry.

4.1 ESD pad protections

Electrostatic discharge (ESD) exists in input and output pads. In order to protect the chip, ESD protections are required. One of the most simple ESD protection is made up of a set of two diodes and a resistance (Appendix Fig.5). One diode handles the negative voltage Vss flowing inside the circuit (N+/Psubstrate), the other diode (P+/N well) handles the positive voltage Vdd. The layout of ESD protection pad with the N+/P substrate diode and P+/N well) diode is shown in Fig1.

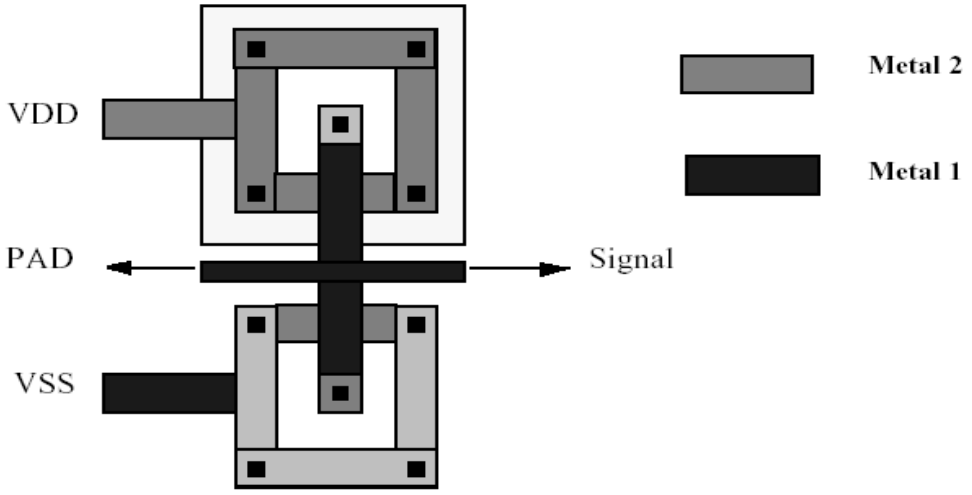


Figure 11.1.Create a diode for I/O pad protections

One simple way to add a diode in the layout is to click on the cell library icon, then click on the Contacts menu and to assert the options: <Diff P + Metal1> and <Diff N+ on Nwell>. This creates a P+/ Nwell diode with its appropriate contacts (Fig. 2). To select <Diff N+ to Metal1> and <Diff P+ on Psubstrate> generate N+/P substrate diode.

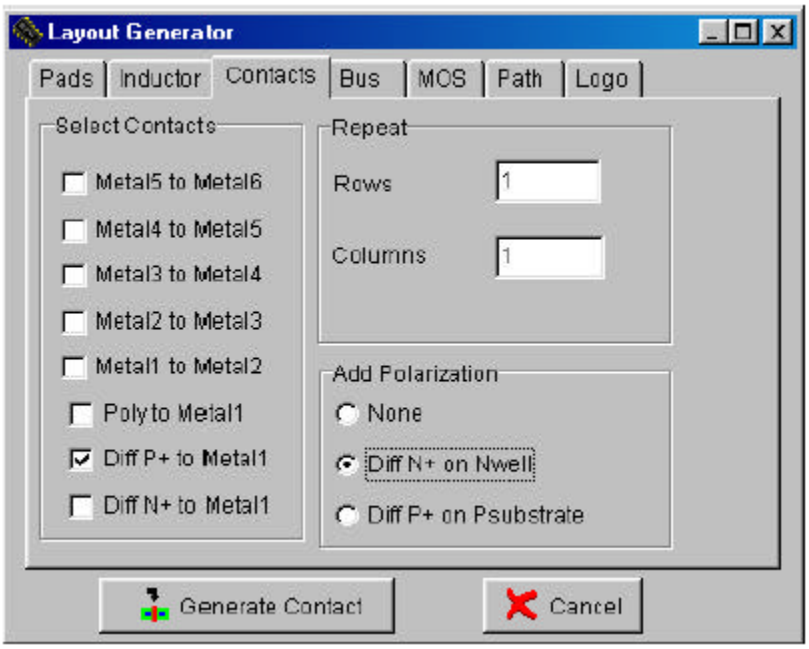


Figure 11.2: Layout Generator.

Note that to change **Rows** and **Columns** with any more than 1 numbers can enlarge the diode size. Save your design as ESDprotections.msk in your directory.

4.2 Design the I/O pad ring

To generate the pad ring: Edit -> Generate -> I/O Pads.

A windows similar to Fig. 3 pops up. The <Pad ring> option should be selected. The values <Pads in X: 8>, <Pads in Y: 8>, <Ring Width: 15 µm> and <Vdd, Vss pairs: 4> should be filled in.

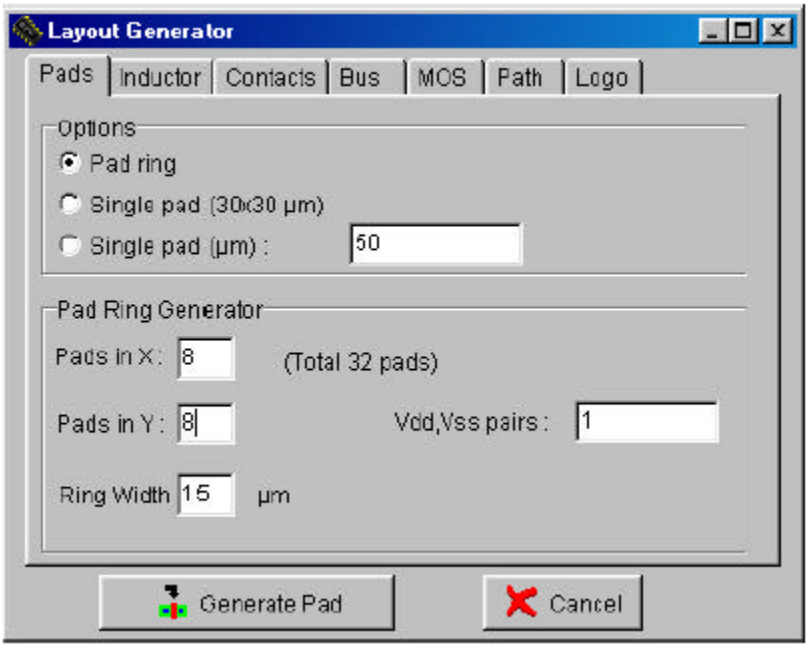


Figure 11.3: Generate Pads Window

A pad ring similar (not exactly the same) to Fig. 4 in the lab instruction is generated.You can now inspect the generated pad ring. It has four pairs of Vdd-Vss pads,correctly connected. Make sure that the supply voltage is correct, 2.5 V. You can also see that the ESD protection circuits already exist in the generated pad ring.

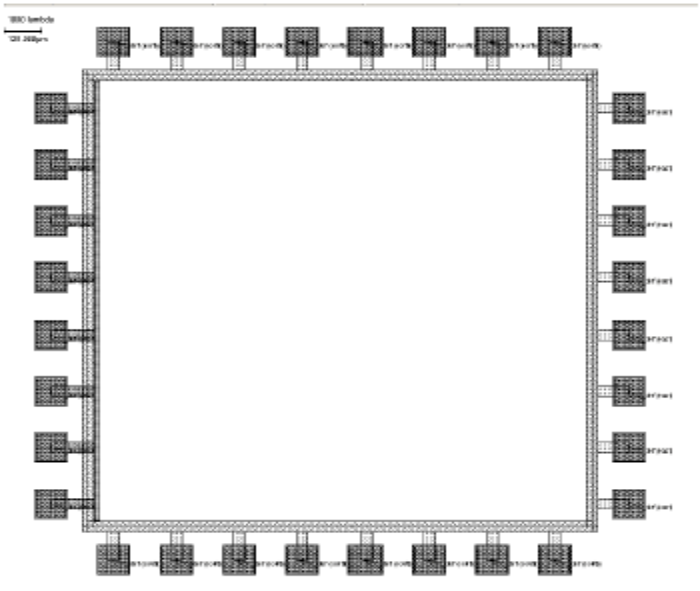


Figure 11.4: Overview of Chip

The pads and the inner ring (Vdd) have all six metal layers. We only need two in this lab. Extra metal layers require more masks, resulting in a higher cost. Therefore, we remove the unnecessary metal layers.

1. From Edit -> Protect all.

2. Select M 6, M 5, M 4 and M 3 on the Palette, and remove all these layers in the layout (pads and pad ring).

3. Select M 2 on the Palette and remove the Metal 2 on the left side of the inner ring.

4. After removing the metal layers, make sure that the power supply pads are still correctly connected, the 4 Vdd pads should be connected to the inner ring and the

4 Vss pads to the outer.

5. Just as in the lab instruction, besides that the power supply pads are now already properly connected. You only need to distribute the pads to the signals (A0-A7, Sh0-Sh7 and the outputs s0-s7).

4.3 Assembling the chip

1. Open lab4.msk from File.

2. Click File -> Insert ->open your ESDprotections.msk file. It appears on the right corner of the pad ring window. Put it in each I/O pad. The connection way can be referred Fig.1.

3. Click File -> Insert ->open your lab3 msk file. The barrel shifter is on the right corner of the pad ring window. Moving it into the centre of the pad ring.

4. Connecting all I/O,Vdd and Vss of lab 3 cell to the relative pads with Metal 1 and Metal 2. Please refer to **Path between Pads and Cells** in appendix. The Width Metal can be filled in any more than 10 lamda in order to visible on the whole chip.
Note that the work need you careful and patient. It consumes lots of time.

Hint: repeating using <Zoom in>, <all> to enlarge the part of chip.

Steps for connection between Pads and Cells.

1. Click the stretch icon;

2. Stretch some path edge with mouse;

3. Hold down one of arrow keys or move Mouse to needed direction;

4. put the VIA to make M1/M2 contacted as need two metal wires;

5. Check node if identified between pads and cells;

6. Do Design Rule Check. If errors, correct them;

7. If right, save your design;

4.4 I/O stimuli

- Name Pad the same with Lab4.

- Name the inputs to the barrel shifter A7 through A0 in the order displayed in figure 4. Use the **Visible node** in the palette for this and select the properties to Vdd or Vss so that You set a clever selection of fixed input values to enable Your investigation on how the output is depending on the control signals.

- Name the control inputs to the barrel shifter Sh0 through Sh7 in the order displayed in figure 6. Use the **Visible node** in the palette for this and select the property to pulse in order to apply interesting control signals. Let **one and only one** control signal be high at each instant. The pulse should have a rise and fall time of 0.5ns. The pulse width should stay high for at least 2ns

- Name Your buffer outputs s7 through s0 in such manner that the index corresponds to that of the barrel shifters input signals. Use the **Visible node** in the palette for this and select the property to variable. Apply Vdd and Vss voltages to the buffer, using **Vdd supply** and **Ground** from the palette.

5. Simulate

5.1 Simulate functionality

Check that Your design is working properly.

Study maximum time delay.

Maximum time delay: _____

Check the average wire length, by doing:

File -> Statistics -> Interconnects.

Average length: _____

Check the capacitance on the following nodes:

A7: _____ F
A6: _____ F

A5: _____ F
A4: _____ F
A3: _____ F
A2: _____ F
A1: _____ F
A0: _____ F

Also check the capacitance, on the barrel shifters output before the buffer:

B7: _____ F
B6: _____ F
B5: _____ F
B4: _____ F
B3: _____ F
B2: _____ F
B1: _____ F
B0: _____ F

6. Discussion

Explain the difference between Lab4 and Lab5.

_____
_____
_____
_____
_____

Explain the simulation You have carried out.

_____
_____
_____
_____

Motivate Your time delay result.

_____
_____

Motivate why the obtained capacitance values on A7 through A0 are not equal.

_____
_____
_____

Explain the obtained results regarding the capacitance values on B7 through B0.

_____
_____

You can now call an assistant and **demonstrate** results and discuss Your conclusions.