## CONTENTS

<table>
<thead>
<tr>
<th>S.No</th>
<th>Title</th>
<th>Page No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Syllabus</td>
<td>02</td>
</tr>
<tr>
<td>2.</td>
<td>Course objective</td>
<td>03</td>
</tr>
<tr>
<td>3.</td>
<td>Course outcome</td>
<td>04</td>
</tr>
<tr>
<td>4.</td>
<td>Do’s &amp; Don’t’s</td>
<td>05</td>
</tr>
<tr>
<td>5.</td>
<td>List of experiments</td>
<td>06</td>
</tr>
<tr>
<td>6.</td>
<td>Viva questions</td>
<td>82</td>
</tr>
</tbody>
</table>
1. SYLLABUS

Subject Code : 10ECL77
IA Marks : 25
No. of Practical Hrs/Week : 03
Exam Hours : 03
Total no. of Practical Hrs. : 42
Exam Marks : 50

PART – A

DIGITAL DESIGN

1. Write Verilog Code for the following circuits and their Test Bench for verification, observe the waveform and synthesize the code with the technological library, with the given Constraints*. Do the initial timing verification with gate level simulation.

   1. An inverter
   2. A Buffer
   3. Transmission Gate
   4. Basic/universal gates
   5. Flip flop - RS, D, JK, MS, T
   6. Serial & Parallel adder
   7. 4-bit counter [Synchronous and Asynchronous counter]
   8. Successive approximation register [SAR]

* An appropriate constraint should be given

PART - B

ANALOG DESIGN

1. Design an Inverter with given specifications*, completing the design flow mentioned below:

   a. Draw the schematic and verify the following: i) DC Analysis ii) Transient Analysis
   b. Draw the Layout and verify the DRC, ERC
   c. Check for LVS
   d. Extract RC and back annotate the same and verify the Design
   e. Verify & Optimize for Time, Power and Area to the given constraint***

2. Design the following circuits with the given specifications*, completing the design flow mentioned below:

   a. Draw the schematic and verify the following: i) DC Analysis ii) AC Analysis
      iii) Transient Analysis
   b. Draw the Layout and verify the DRC, ERC
   c. Check for LVS
   d. Extract RC and back annotate the same and verify the Design.
1. A Single Stage differential amplifier
2. Common source and Common Drain amplifier

3. Design an **op-amp** with the given specification* using given differential amplifier, Common source and Common Drain amplifier in library** and completing the design flow as mentioned below:
   a. Draw the schematic and verify the following:   i) DC Analysis ii) AC Analysis iii) Transient Analysis
   b. Draw the Layout and verify the DRC, ERC
   c. Check for LVS
   d. Extract RC and back annotate the same and verify the Design.

4. Design a **4 bit R-2R based DAC** for the given specification and completing the design flow mentioned using given op-amp in the library**.
   a. Draw the schematic and verify the following:   i) DC Analysis ii) AC Analysis iii) Transient Analysis
   b. Draw the Layout and verify the DRC, ERC
   c. Check for LVS
   d. Extract RC and back annotate the same and verify the Design.

5. For the **SAR based ADC** mentioned in the figure below, draw the mixed signal schematic and verify the functionality by completing ASIC Design flow. [Specifications to GDS-II]

* Appropriate specification should be given.
** Applicable Library should be added & information should be given to the Designer.
*** An appropriate constraint should be given.
2. COURSE OBJECTIVES

- To educate students with the knowledge of verilog coding and test bench, to write verilog code for all logic gates, flip-flops, counters and adders etc.

- Students will be able to compile, simulate and synthesize the verilog code.

- From this lab the students will be able to draw the schematic diagram and layout for the inverter and amplifiers and verify their functionality.
3. COURSE OUTCOMES

- Write Verilog Code for the all logic gate circuits and their Test Bench for verification, observe the waveform and synthesize the code with the technological library, with the given Constraints.

- Write Verilog Code for the SR, JK, D, T flip-flop circuits and their Test Bench for verification, observe the waveform and synthesize the code with the technological library, with the given Constraints.

- Write Verilog Code for the counters adder circuits and their Test Bench for verification, observe the waveform and synthesize the code with the technological library, with the given Constraints.

- Design an Inverter with given specifications, completing the design flow mentioned below:
  
  a. Draw the schematic and verify the following: i) DC Analysis ii) Transient Analysis
  b. Draw the Layout and verify the DRC, ERC
  c. Check for LVS
  d. Extract RC and back annotate the same and verify the Design
  e. Verify & Optimize for Time, Power and Area to the given constraint

- Design the following circuits with the given specifications, completing the design flow mentioned below:
  
  a. Draw the schematic and verify the following: i) DC Analysis ii) AC Analysis
  iii) Transient Analysis
  b. Draw the Layout and verify the DRC, ERC
  c. Check for LVS
  d. Extract RC and back annotate the same and verify the Design.
     i) A Single Stage differential amplifier and op-amp
     ii) Common source and Common Drain amplifier
4. **DO’S AND DON’TS**

**DO’S**

- Do log off the computer when you finish the work.
- Bring observation, manual, pen etc, with you.
- Make sure that your hands are clean and dry when you use the computer.
- Do ask the staff for assistance if you need help.
- You should be in time in lab.
- Do keep your voice low when speaking to others in the lab.
- All users of the laboratory are to follow the directions of faculty.
- Before leaving lab, you should save your work, collect your belongings.

**DON’TS**

- Do not eat or drink in the laboratory
- Do not use pen drive or similar kind devices without permission.
- Do not take your baggages inside
- Do not change computer preference settings or endeavor to hack into unauthorized areas
- Do not install any programs without the faculty permission.
- Do not create any usr accounts without faculty permission.
- Avoid stepping on electrical wires or any other computer cables.
- Do not touch, connect or disconnect any plug or cable without your lecturer/ laboratory technician’s permission
- Do not misbehave in the computer laboratory.
- Don’t use chat rooms, online games or multiuser domains.
## 5. LIST OF EXPERIMENTS

<table>
<thead>
<tr>
<th>Expt. No.</th>
<th>Name of the Experiment</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Inverter</td>
<td>10</td>
</tr>
<tr>
<td>02</td>
<td>Buffer</td>
<td>11</td>
</tr>
<tr>
<td>03</td>
<td>Transmission gate</td>
<td>13</td>
</tr>
<tr>
<td>04</td>
<td>a) Basic gates</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>b) Universal gates</td>
<td>18</td>
</tr>
<tr>
<td>05</td>
<td>a) D Flip-flop</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>b) SR Flip-flop</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>c) JK Flip-flop</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>d) T Flip-flop</td>
<td>29</td>
</tr>
<tr>
<td>06</td>
<td>Parallel adders</td>
<td>32</td>
</tr>
<tr>
<td>07</td>
<td>a) 4-bit counters asynchronous counter</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>b) 4-bit counters synchronous counter</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td><strong>Part B : Analog design</strong></td>
<td>39</td>
</tr>
<tr>
<td>08</td>
<td>Inverter</td>
<td>44</td>
</tr>
<tr>
<td>09</td>
<td>Common source amplifier</td>
<td>69</td>
</tr>
<tr>
<td>10</td>
<td>Common drain amplifier</td>
<td>72</td>
</tr>
<tr>
<td>11</td>
<td>Single stage differential amplifier</td>
<td>75</td>
</tr>
<tr>
<td>12</td>
<td>Operational amplifier</td>
<td>78</td>
</tr>
</tbody>
</table>
1. **PART A – DIGITAL DESIGN**

**PROCEDURES FOR DIGITAL DESIGN**

**INITIAL PROCEDURE:**

1. After logging in, *right click* and *open terminal*.
   
   ```
   cd ..
   cd cadence_db
   ```

2. Get into the *c shell* by typing the command –
   
   ```
   csh
   ```

3. Run the shellscript by typing the command –
   
   ```
   source cshrc
   ```

4. Check system is connected to LAN
   
   ```
   Mount -a
   ```

**I. STEPS FOR DESIGN ENTRY:**

1. Move inside *NCO* using the *cd* command -
   
   ```
   cd NCO
   ```

2. Similarly move inside *rclab* and *rtl* using the *cd* command -
   
   ```
   cd rclabs
   cd rtl
   ```

3. Write the verilog program for your design (e.g.: Codefile1.v)
   
   ```
   gedit codefile1.v
   ```

4. Write the verilog test bench program for your design (e.g.: Codefile1_tb.v). Now, the design entry using HDL gets finished.
   
   ```
   gedit codefile1_tb.v
   ```

**II. STEPS FOR SIMULATION:**

1. Initially, both of your verilog programs have to be compiled

2. After compilation, you have to elaborate the top module
All this is done using the following steps

```bash
cd ..
cd Simulation
nclaunch
```

### III. STEPS FOR SYNTHESIS:

1. Move into the *work* directory using the command –

```bash
cd ..
cd work
rc
```

2. After the above steps you will in rc command prompt then to synthesize your code execute the following command

```bash
set_attribute lib_search_path ../library
set_attribute library slow_normal.lib
set_attribute hdl_search_path ../rtl
read_hdl Codefile1.v
elaborate
read_sdc constraints_top.g
synthesize --to_mapped
gui_show
report power
report gate
report timing
report area
write_hdl> codefile1.v
write_sdc> codefile1.sdc
exit
gedit codefile1.v
```
Experiment No: 01                Date:

INVERTER

AIM:
To write verilog code for an inverter circuit and its test bench for verification, observe the waveform and synthesize the code with technological library with given Constraints.

TOOL REQUIRED:
Cadence Tool

THEORY:

The NOT gate or an inverter is an electronic circuit that produces an inverted version of the input at its output. It is also known as an inverter. If the input variable is A, the inverted output is known as NOT A. This is also shown as A’, or A with a bar over the top, as shown at the outputs. The diagrams below show two ways that the NAND logic gate can be configured to produce a NOT gate. It can also be done using NOR logic gates in the same way.

<table>
<thead>
<tr>
<th>A</th>
<th>NOT gate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

PROGRAM:
Module inverter (a,Y_INV):
Input a;
Output Y_INV;
Assign Y_INV =~a;
End module

TEST BENCH:
Module inverter_tb();
reg a; wire Y_INV;
inverter inverter_ins (a, Y_INV);
initial
begin
a=0;
#10  a=1;
#10  a=0; #10  a=1;
End
End module

RESULT:
Verilog code for the inverter circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.
Experiment No: 02

AIM:
To write verilog code for a buffer circuit and its test bench for verification, observe the waveform and synthesize the code with technological library with given Constraints.

TOOL REQUIRED:
Cadence Tool

THEORY:

"Buffer" gate

![Buffer gate diagram]

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

A special logic gate called a buffer is manufactured to perform the same function as two inverters. Its symbol is simply a triangle, with no inverting “bubble” on the output terminal. Buffer gates merely serve the purpose of signal amplification: taking a “weak” signal source that isn’t capable of sourcing or sinking much current, and boosting the current capacity of the signal so as to be able to drive a load.

PROGRAM:
Module buffer (a,Y):
Input a;
Output Y;
Assign Y=a;
End module
TEST BENCH:
Module buffer_tb();
Reg a;
Wire Y;
buffer buffer_ins(a,Y);
Initial
Begin
a=0;
#10 a=1;
#10 a=0;
#10 a=1;
End
End module

RESULT:
Verilog code for the buffer circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.
Experiment No: 03

TRANSISION GATE

AIM:
To write verilog code for an Transmission Gate circuit and its test bench for verification, observe the waveform and synthesize the code with technological library with given Constraints.

TOOL REQUIRED:
Cadence Tool

THEORY:
A transmission gate, or analog switch, is defined as an electronic element that will selectively block or pass a signal level from the input to the output.

Basic Operation
This solid-state switch is comprised of a pMOS transistor and nMOS transistor. The control gates are biased in a complementary manner so that both transistors are either on or off.

When the voltage on node A is a Logic 1, the complementary Logic 0 is applied to node active-low A, allowing both transistors to conduct and pass the signal at IN to OUT. When the voltage on node active-low A is a Logic 0, the complementary Logic 1 is applied to node A, turning both transistors off and forcing a high-impedance condition on both the IN and OUT nodes. This high-impedance condition represents the third "state" (high, low, or high-Z) that the channel may reflect downstream. The schematic diagram (Figure 1) includes the arbitrary labels for IN and OUT, as the circuit will operate in an identical manner if those labels were reversed. This design provides true bidirectional connectivity without degradation of the input signal.

Figure 1. Schematic representation of a transmission gate.

PROGRAM:

module trans(a,ctrl,Y);
input a,ctrl;

GCEM
output Y;
reg Y;
always @(ctrl)
begin
if(ctrl==1)
Y<=a;
else
Y<=1’bz;
end
endmodule

TEST BENCH:

module trans_tb();
wire Y;
trans trans_ins(a,ctrl,Y);
initial
begin
a=0;
#10 a=1;
#10 a=0;
#10 a=1;
#10 a=1;
#5 $finish
end
initial
begin
ctrl=0;
#15 ctrl=1;
#15 ctrl=0;
#25 ctrl=1;
#5 $finish
end
endmodule

RESULT:
Verilog code for the transmission gate circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.
Experiment No: 04a

BASICS GATES

AIM:
To write verilog code for an basics Gate circuit and its test bench for verification, observe the waveform and synthesize the code with technological library with given Constraints.

TOOL REQUIRED:
Cadence Tool

THEORY:
Digital systems are said to be constructed by using logic gates. These gates are the AND, OR, NOT, NAND, NOR, EXOR and EXNOR gates. The basic operations are described below with the aid of truth tables.

AND gate

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A.B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The AND gate is an electronic circuit that gives a high output (1) only if all its inputs are high. A dot (.) is used to show the AND operation i.e. A.B. Bear in mind that this dot is sometimes omitted i.e. AB

OR gate

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A+B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The OR gate is an electronic circuit that gives a high output (1) if one or more of its inputs are high. A plus (+) is used to show the OR operation.
NOT gate

The NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an inverter. If the input variable is A, the inverted output is known as NOT A. This is also shown as A', or A with a bar over the top, as shown at the outputs. The diagrams below show two ways that the NAND logic gate can be configured to produce a NOT gate. It can also be done using NOR logic gates in the same way.

EXOR gate

The 'Exclusive-OR' gate is a circuit which will give a high output if either, but not both, of its two inputs are high. An encircled plus sign (⊕) is used to show the EOR operation.

EXNOR gate

The 'Exclusive-NOR' gate circuit does the opposite to the EOR gate. It will give a low output if either, but not both, of its two inputs are high. The symbol is an EXOR gate with a small circle on the output. The small circle represents inversion.
PROGRAM:
module basic1(a,b,Y\_AND,Y\_OR,Y\_NOT,Y\_XOR,Y\_XNOR);
input a,b;
output Y\_AND,Y\_OR,Y\_NOT,Y\_XOR,Y\_XNOR;
assign Y\_AND = a & b;
assign Y\_OR = a | b;
assign Y\_NOT =~ a ;
assign Y\_XOR = a ^ b;
assign Y\_XNOR = ~(a ^ b);
endmodule

TEST BENCH:
module basic1\_tb();
wire Y\_AND,Y\_OR,Y\_NOT,Y\_XOR,Y\_XNOR;
reg a,b;
basic1 basic1\_ins(a,b,Y\_AND,Y\_OR,Y\_NOT,Y\_XOR,Y\_XNOR);
initial
begin
a=0; b=0;
#5 a=0; b=1;
#5 a=1; b=0;
#5 a=1; b=1;
end
endmodule

RESULT:
Verilog code for the basic gates circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.
Experiment No: 04b

UNIVERSAL GATES

AIM:
To write verilog code for an universal Gate circuit and its test bench for verification, observe the waveform and synthesize the code with technological library with given Constraints.

TOOL REQUIRED:
Cadence Tool

THEORY:

NAND gate

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A*B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate. The outputs of all NAND gates are high if any of the inputs are low. The symbol is an AND gate with a small circle on the output. The small circle represents inversion.

NOR gate

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A+B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate. The outputs of all NOR gates are low if any of the inputs are high. The symbol is an OR gate with a small circle on the output. The small circle represents inversion.
PROGRAM:
module unigate(a,b, Y\textsubscript{NAND}, Y\textsubscript{NOR});
input a,b;
output Y\textsubscript{NAND}, Y\textsubscript{NOR};
assign Y\textsubscript{NAND} = ~(a & b);
assign Y\textsubscript{NOR} = ~(a | b);
endmodule

TEST BENCH:
module unigate\_tb();
wire Y\textsubscript{NAND}, Y\textsubscript{NOR};
reg a,b;
unigate unigate\_ins(a,b, Y\textsubscript{NAND}, Y\textsubscript{NOR});
initial
begin
a=0; b=0;
#5 a=0; b=1;
#5 a=1; b=0;
#5 a=1; b=1;
end
endmodule

RESULT:
Verilog code for the universal gates circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.
Experiment No: 05a

D FLIP FLOP

AIM:
To write verilog code for an D flip flop circuit and its test bench for verification, observe the waveform and synthesize the code with technological library with given Constraints.

TOOL REQUIRED:
Cadence Tool

THEORY:
In electronics, a flip-flop or latch is a circuit that has two stable states and can be used to store state information. A flip-flop is a bistable multivibrator. The circuit can be made to change state by signals applied to one or more control inputs and will have one or two outputs. It is the basic storage element in sequential logic. Flip-flops and latches are fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems.

Flip-flops and latches are used as data storage elements. A flip-flop stores a single bit (binary digit) of data; one of its two states represents a "one" and the other represents a "zero". Such data storage can be used for storage of state, and such a circuit is described as sequential logic.

D flip-flop

The D flip-flop is widely used. It is also known as a "data" or "delay" flip-flop.

The D flip-flop captures the value of the D-input at a definite portion of the clock cycle (such as the rising edge of the clock). That captured value becomes the Q output. At other times, the output Q does not change. The D flip-flop can be viewed as a memory cell, a zero-order hold, or a delay line.

D flip-flop symbol

Truth table:

<table>
<thead>
<tr>
<th>Clock</th>
<th>D</th>
<th>Q_{next}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rising edge</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rising edge</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Non-Rising</td>
<td>X</td>
<td>Q</td>
</tr>
</tbody>
</table>

('X' denotes a Don't care condition, meaning the signal is irrelevant)
PROGRAM:
module dff1(d,clk,rst,q,qb);
input d,clk,rst;
output q,qb;
reg q;
always @ (posedge clk)
begin
if(rst)
q<=1'b0;
else
q<=d;
end
assign qb=~q;
endmodule

TEST BENCH:
module dff1_tb();
reg d,clk,rst;
wire q,qb;
dff1 dff1_ins(d,clk,rst,q,qb);
initial
begin
clk=0; forever #5 clk=~clk;
end
initial
begin
rst=1;
#10 rst=0;
#70 rst=1;
#10 rst=0;
#70 rst=1;
end
initial
begin
d=0;
#15 d=1;
end
#15 d=0;
#15 d=1;
#30 d=0;
end
endmodule

**RESULT:**
Verilog code for the D flip-flop circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.
Experiment No: 5b

SR FLIP FLOP

AIM:
To write verilog code for an SR flip flop circuit and its test bench for verification, observe the waveform and synthesize the code with technological library with given Constraints.

TOOL REQUIRED:
Cadence Tool

THEORY:

Truth Table

<table>
<thead>
<tr>
<th>State</th>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>Q</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Set Q \to 1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>no change</td>
</tr>
<tr>
<td>Reset</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Reset Q \to 0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>no change</td>
</tr>
<tr>
<td>Invalid</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Invalid Condition</td>
</tr>
</tbody>
</table>

It can be seen that when both inputs S = “1” and R = “1” the outputs Q and \(\overline{Q}\) can be at either logic level “1” or “0”, depending upon the state of the inputs S or R BEFORE this input condition existed. Therefore the condition of S = R = “1” does not change the state of the outputs Q and \(\overline{Q}\).
However, the input state of \( S = 0 \) and \( R = 0 \) is an undesirable or invalid condition and must be avoided. The condition of \( S = R = 0 \) causes both outputs \( Q \) and \( Q \) to be HIGH together at logic level “1” when we would normally want \( Q \) to be the inverse of \( Q \). The result is that the flip-flop looses control of \( Q \) and \( Q \), and if the two inputs are now switched “HIGH” again after this condition to logic “1”, the flip-flop becomes unstable and switches to an unknown data state based upon the unbalance as shown in the following switching diagram.

**PROGRAM**:

```verilog
module srff(s,r,clk,rst,q,qb);
input s,r,clk,rst;
output q,qb;
wire s,r,clk,rst,qb;
reg q;
always @ (posedge clk)
begin
  if(rst)
    q<=1'b0;
  else
    if (s==1'b0 && r==1'b0)
      q<=q;
    else if (s==1'b0 && r==1'b1)
      q<=1'b0;
    else if (s==1'b1 && r==1'b0)
      q<=1'b1;
    else if (s==1'b1 && r==1'b1)
      q<=1'bx;
  end
  assign qb=~q;
endmodule
```
TEST BENCH:
module srff_tb();
wire q,qb;
reg s,r,clk,rst;
srff srff_ins(s,r,clk,rst,q,qb);
initial
begin
clk=0;
forever   #5 clk=~clk;
end
initial
begin
rst=1;
    #10 rst=0;
    #80  rst=1;
    #10 rst=0;
end
initial
begin
s=0; r=0;
    #15 s=0; r=1;
    #15 s=1; r=0;
    #15 s=1; r=1;
    #15 s=0; r=0;
end
endmodule

RESULT:
Verilog code for the SR flip-flop circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.
Experiment No: 5c

JK FLIP FLOP

AIM:
To write verilog code for an JK flip flop circuit and its test bench for verification, observe the waveform and synthesize the code with technological library with given Constraints.

TOOL REQUIRED:
Cadence Tool

THEORY:
The Truth Table

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>K</td>
<td>Q</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

same as for the SR Latch

Toggles on leading edge of clock signal

SR flip-flop

Symbol

Circuit
Then the JK flip-flop is basically an SR flip flop with feedback which enables only one of its two input terminals, either SET or RESET to be active at any one time thereby eliminating the invalid condition seen previously in the SR flip flop circuit. Also when both the J and the K inputs are at logic level “1” at the same time, and the clock input is pulsed either “HIGH”, the circuit will “toggle” from its SET state to a RESET state, or visa-versa. This results in the JK flip flop acting more like a T-type toggle flip-flop when both terminals are “HIGH”.

Although this circuit is an improvement on the clocked SR flip-flop it still suffers from timing problems called “race” if the output Q changes state before the timing pulse of the clock input has time to go “OFF”. To avoid this the timing pulse period (T) must be kept as short as possible (high frequency). As this is sometimes not possible with modern TTL IC’s the much improved **Master-Slave JK Flip-flop** was developed.

**PROGRAM:**

```
Module jkff (j,k,clk,rst,q,qb);
Input j,k,clk,rst;
Output q,qb;
Wire j,k,clk,rst,qb;
Reg q;
Always @ (posedge clk)
Begin
  if(rst)
    q<=1'b0;
  else if (j==0 && k==0)
    q=q;
  else if (j==0 && k==1)
    q<=1'b0;
  else if (j==1 && k==0)
    q<=1'b1;
  else if (j==1 && k==1)
    q<=~q;
```
end
assign qb=~q;
endmodule

**TEST BENCH:**

module JKff_tb();
wire q,qb;
reg J,K,,clk,rst;
JKff jkff_ins(J,K,clk,rst,q,qb);
initial
begin
clk=0;
forever #5 clk=~clk;
end
initial
begin
rst=1;
#15 rst=0;
#80 rst=1;
#15 rst=0;
end
initial
begin
J=0; K=0;
#15 J=0; K=1;
#15 J=1; K=0;
#15 J=1; K=1;
#15 J=0; K=0;
end
endmodule

**RESULT:**

Verilog code for the JK flip-flop circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.
Experiment No: 5b

T FLIP FLOP

AIM:
To write verilog code for an T flip flop circuit and its test bench for verification, observe the waveform and synthesize the code with technological library with given Constraints.

TOOL REQUIRED:
Cadence Tool

THEORY:

T flip-flop

A circuit symbol for a T-type flip-flop

If the T input is high, the T flip-flop changes state ("toggles") whenever the clock input is strobed. If the T input is low, the flip-flop holds the previous value. This behavior is described by the characteristic equation:

<table>
<thead>
<tr>
<th>Characteristic table</th>
<th>Excitation table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>Comment</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

When T is held high, the toggle flip-flop divides the clock frequency by two; that is, if clock frequency is 4 MHz, the output frequency obtained from the flip-flop will be 2 MHz. This "divide by" feature has application in various types of digital counters. A T flip-flop can also be built using a JK flip-flop (J & K pins are connected together and act as T) or a D flip-flop (T input XOR Qprevious drives the D input).
PROGRAM:
module Tff (T,clk,rst,q,qb);
input T,clk,rst;
output q,qb;
wire T,clk,rst,qb;
reg tq;
always @ (posedge clk)
begin
if(rst)
tq<=1'b0;
else
begin
if(T==1'b1)
tq<=~tq;
end
end
assign q=~tq;
assign qb=~q;
endmodule

TEST BENCH:
module Tff_tb();
wire q,qb;
reg T,clk,rst;
Tff Tff_ins(T,clk,rst,q,qb);
initial
begin
clk=0;
forever 
#5 clk=~clk;
end
initial
begin
rst=1;
#10 rst=0;
#80 rst=1;
#10 rst=0;
end
end
initial
begin
  t=0;
  #15 T=1;
  #15 T=0;
  #15 T=1;
  #15 T=0;
end
endmodule

RESULT:
Verilog code for the T flip-flop circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.
Experiment No: 06

PARALLEL ADDER

AIM:
To write verilog code for an parallel adder circuit and its test bench for verification, observe the waveform and synthesize the code with technological library with given Constraints.

TOOL REQUIRED:
Cadence Tool

THEORY:

Addition is a fundamental operation for any digital system, digital signal processing or control system. A fast and accurate operation of a digital system is greatly influenced by the performance of the resident adders. Adders are also very important component in digital systems because of their extensive use in other basic digital operations such as subtraction, multiplication and division.

Parallel adder is a combinatorial circuit (not clocked, does not have any memory and feedback) adding every bit position of the operands in the same time. Thus it is requiring number of bit-adders(full adders + 1 half adder) equal to the number of bits to be added. The Parallel adder is constructed by cascading full adders (FA) blocks in series. One full adder is responsible for the addition of two binary digits at any stage of the ripple carry. The carryout of one stage is fed directly to the carry-in of the next stage.

PROGRAM:

```verilog
module adder(x,y,cin,sum,cout);
input[3:0]x,y;
input cin;
wire c1,c2,c3;
output[3:0] sum;
output cout;
```

GCEM 32
```verilog
fulladd stage0(x[0], y[0], cin, sum[0], c1);
fulladd stage 1(x[1], y[1], c1, sum[1], c2);
fulladd stage 2(x[2], y[2], c2, sum[2], c3);
fulladd stage 3(x[3], y[3], c3, sum[3], cout);
endmodule
module fulladd(x, y, carryin, s, carryout);
input x, y, carryin;
output s, carryout;
assign s = x ^ y ^ carryin;
assign carryout = (x & y) | (y & carryin) | (carryin & x);
endmodule

TEST BENCH:
module adder_tb();
reg[3:0] x, y;
reg cin;
wire[3:0] sum;
wire cout;
adder adder_ins(x, y, cin, sum, cout);
initial
begin
  x = 4'd0110;  y = 4'd1100;  cin = 1'b0;
  #15 x = 4'd1111;  y = 4'd1010;  cin = 1'b1;
  #15 x = 4'd1011;  y = 4'd0110;  cin = 1'b0;
  #15 x = 4'd0111;  y = 4'd1110;  cin = 1'b1;
  #15 $ finish
end
endmodule

RESULT:
Verilog code for the parallel adder circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.
Experiment No: 7a

ASYNCHRONOUS COUNTER

AIM:
To write verilog code for asynchronous counter circuit and its test bench for verification, observe the waveform and synthesize the code with technological library with given Constraints.

TOOL REQUIRED:
Cadence Tool

THEORY:

A four-bit “up” counter

This circuit would yield the following output waveforms, when “clocked” by a repetitive source of pulses from an oscillator:

<table>
<thead>
<tr>
<th>Clock</th>
<th>Q0</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1</td>
<td>0 0 1 1 0 0 1 1 0 0 1 1</td>
<td>0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1</td>
<td></td>
</tr>
</tbody>
</table>

A ripple counter is an asynchronous counter where only the first flip-flop is clocked by an external clock. All subsequent flip-flops are clocked by the output of the preceding flip-flop. Asynchronous
counters are also called ripple-counters because of the way the clock pulse ripples it way through the flip-flops.

The MOD of the ripple counter or asynchronous counter is $2^n$ if $n$ flip-flops are used. For a 4-bit counter, the range of the count is 0000 to 1111 ($2^4$-1). A counter may count up or count down or count up and down depending on the input control. The count sequence usually repeats itself. When counting up, the count sequence goes from 0000, 0001, 0010, ... 1110, 1111, 0000, 0001, ... etc. When counting down the count sequence goes in the opposite manner: 1111, 1110, ... 0010, 0001, 0000, 1111, 1110, ... etc.

Four J-K flip-flops connected in such a way to always be in the “toggle” mode, we need to determine how to connect the clock inputs in such a way so that each succeeding bit toggles when the bit before it transitions from 1 to 0. The Q outputs of each flip-flop will serve as the respective binary bits of the final, four-bit count.

If we used flip-flops with negative-edge triggering (bubble symbols on the clock inputs), we could simply connect the clock input of each flip-flop to the Q output of the flip-flop before it, so that when the bit before it changes from a 1 to a 0, the “falling edge” of that signal would “clock” the next flip-flop to toggle the next bit.

**PROGRAM:**

```verilog
decler module counter(clk,rst,count);
input clk,rst;
output[3:0]count;
reg[3:0]count;
always @(negedge clk)
if(rst)
    count[0]<=1’b0;
else
    count[0]<=~count[0];
always @(negedge count[0])
if(rst)
    count[1]<=1’b0;
else
    count[1]<=~count[1];
always @(negedge count[1])
if(rst)
    count[2]<=1’b0;
else
    count[2]<=~count[2];
always @(negedge count[2])
```

GCEM
if (rst)
  count[3]<=1’b0;
else
  count[3]<=~count[3];
endmodule

TEST BENCH:
module counter_tb();
wire [3:0] count;
reg clk, rst;
counter counter_ins(clk,rst,count);
initial
begin
  clk=0
  forever #5 clk=~clk;
end
initial
begin
  rst=1
  #10 rst=0;
  #150 rst=1;
  #10 rst=0;
  $finish
end
endmodule

RESULT:
Verilog code for the asynchronous counter circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.
Experiment No: 7b

SYNCHRONOUS COUNTER

AIM:
To write verilog code for synchronous counter circuit and its test bench for verification, observe the waveform and synthesize the code with technological library with given Constraints.

TOOL REQUIRED:
Cadence Tool

THEORY:

A four-bit synchronous “up” counter

A synchronous counter, in contrast to an asynchronous counter, is one whose output bits change state simultaneously, with no ripple. The only way we can build such a counter circuit from J-K flip-flops is to connect all the clock inputs together, so that each and every flip-flop receives the exact same clock pulse at the exact same time.

The figure shows a four-bit synchronous “up” counter. Each of the higher-order flip-flops are made ready to toggle (both J and K inputs “high”) if the Q outputs of all previous flip-flops are “high.” Otherwise, the J and K inputs for that flip-flop will both be “low,” placing it into the “latch” mode where it will maintain its present output state at the next clock pulse. Since the first (LSB) flip-flop needs to toggle at every clock pulse, its J and K inputs are connected to V_{cc} or V_{dd}, where they will be “high” all the time. The next flip-flop need only “recognize” that the first flip-flop’s Q output is high to be made ready to toggle, so no AND gate is needed. However, the remaining flip-flops should be made ready to toggle only when all lower-order output bits are “high,” thus the need for AND gates.
PROGRAM:
module counter(clk,rst,count);
input clk,rst;
output[3:0] count;
reg [3:0] count;
always@(posedge.clk)
begin
if(rst)
count<=4’b0000;
else
count<=count+4’b0001;
end
endmodule

TEST BENCH:
module counter_tb();
reg clk,rst;
wire[3:0]count;
counter counter_ins(clk,rst,count);
initial
begin
clk=0;
forever #5 clk=~clk;
end
initial
begin
rst=1; #10 rst=0; #70 rst=1; #10 rst=0; $finish
end
endmodule

RESULT:
Verilog code for the synchronous counter circuit and its test bench for verification is written, the waveform is observed and the code is synthesized with the technological library and is verified.
PART – B ANALOG DESIGN

PROCEDURES FOR ANALOG DESIGN

CUSTOM IC DESIGN FLOW

Specifications

Schematic entry

Circuit simulation

Layout

Design rule check

Layout versus schematic

Parasitic RC extraction

Back annotation

GDS-II to foundry
INITIAL PROCEDURES:

1. After logging in, right click and open terminal.
   
   cd ..
   
   cd cadence_db

2. Get into the c shell by typing the command
   
   csh

3. Run the shellscript by typing the command
   
   source cshrc

4. Check system is connected to LAN
   
   Mount -a

5. Move inside the respective directory using the cd command
   
   cd cadence_ms_labs_614

6. Invoke the analog design tool by using the command
   
   virtuoso

   After the virtuoso console opens up, maximize it. The linux terminal can be minimized.

7. In the virtuoso console, create your own library by following the steps –File→New→Library
8. In the “New Library” window that opens up, fill in your library name (e.g.: Design2), and then click on the option –

**Attach to an existing technology library**

9. A selection box named “Attach Library to Technology Library” will open. Select “gpdk180” and click on OK.
I. STEPS FOR DESIGN ENTRY:
1. In the virtuoso console, select the following –

   **File → New → Cellview**

2. In the “New File” form that opens, browse for your library name, and in front of the *Cell*, fill in the name of the schematic that is going to be entered (e.g.: inverter). Later, click on *OK*.

3. A new design window opens, in which the schematic has to be entered.
4. After the schematic entry, a symbol for the schematic has to be created in the schematic editor window, by selecting the following –

**Create → Cellview → From Cellview**

During these steps, one more editor window will open up for the symbol entry. The symbol generation procedures will be elaborated while describing the experiments.

5. After symbol creation, both the editor windows can be closed. Now for the test circuit, a new *Cellview* has to be created in the virtuoso console. Again, the detailed procedures for the test circuit are elaborated while discussing the respective experiments.

II. STEPS FOR SIMULATION AND LAYOUT:

The test circuit has to be simulated by launching ADE-L in the schematic editor window, and then by choosing the respective analyses in ADE. The three main analyses that are performed are *transient*, *dc* and *ac*. With output plotted in *Y*-axis, the details of these analyses are summarized below –

<table>
<thead>
<tr>
<th>Type of analysis</th>
<th>X-axis</th>
<th>Observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transient</td>
<td>Time</td>
<td>Waveform</td>
</tr>
<tr>
<td>DC</td>
<td>Input</td>
<td>Transfer characteristics</td>
</tr>
<tr>
<td>AC</td>
<td>Frequency</td>
<td>Bandwidth</td>
</tr>
</tbody>
</table>

After the circuit verification, the layout for the schematic has to be prepared using Layout-XL, and the same has to be physically verified. The detailed procedures are explained with the experiments.
Experiment No:08

AIM:
To simulate the schematic of the CMOS inverter, and then to perform the physical verification for the layout of the same.

TOOL REQUIRED:
Cadence Tool

THEORY:
The inverter is universally accepted as the most basic logic gate doing a Boolean operation on a single input variable. Fig.1 depicts the symbol, truth table and a general structure of a CMOS inverter. As shown, the simple structure consists of a combination of an pMOS transistor at the top and a nMOS transistor at the bottom.

CMOS is also sometimes referred to as complementary-symmetry metal–oxide–semiconductor. The words "complementary-symmetry" refer to the fact that the typical digital design style with CMOS uses complementary and symmetrical pairs of p-type and n-type metal oxide semiconductor field effect transistors (MOSFETs) for logic functions. Two important characteristics of CMOS devices are high noise immunity and low static power consumption. Significant power is only drawn
while the transistors in the CMOS device are switching between on and off states. Consequently, CMOS devices do not produce as much waste heat as other forms of logic, for example transistor-transistor logic (TTL) or NMOS logic, which uses all n-channel devices without p-channel devices.

**PROCEDURE:**
The three initial steps before simulation are: schematic entry, symbol entry and test circuit entry. The procedures are as detailed below –

1. **DESIGN ENTRY:**

   1. In the schematic editor window, for the addition of instances, press “i”. This will open the “Add Instance” window. In that window, browse for the library gpdk180, select the cell *pmos* and then select the view *symbol*. Click on *close*.

   2. The properties of the selected instance are displayed in the “Add Instance” window. There is no need to modify any properties for this particular experiment. Click on *Hide*.
3. The `pmos` symbol will move along with the cursor. Place it in the top mid-position, left-click and then press `Esc`.

4. Similarly place `nmos` device, and then press `Esc`. 

5. Now, press “w” for placing wire, click on the respective nodes and connect them through wire. Place the input and output wires as well. Complete the substrate connections also.

6. After pressing Esc, press “p” for adding pins to the schematic diagram. In the “Add Pin” window, enter the name of the pin (e.g.: in), and ensure its direction as input.

7. Click on Hide, and place the pin at the input. Later press Esc.
8. Similarly, place the output pin with name “out” and direction output.

9. Complete the schematic by placing the instances “vdd” and “gnd”, which are in the analogLib library. Finally, click on “check & save” icon and observe the errors in the virtuoso console. In the schematic window, the errors will be highlighted with yellow boxes. Move those boxes, correct the errors, and click on “check & save”. Correct all the errors that are reported.
10. After the schematic entry is finished, a symbol for the design has to be created. For this purpose, click on *Create* and follow the procedure –

*Create → Cellview → From Cellview*

The name comes by default, along with the other options. Click on *OK*.

11. Another window opens, which shows the input and output pin configurations. Click on *OK*. 
12. A symbol editor window opens up with the default symbol.

13. The default symbol can be deleted and the symbol as per our convention can be created. For this purpose, click on the green colored edge of the symbol. Now the color of the selected part
14. On the pulldown menu, Select Create → Shape → Line and click on the editor window. Use the mouse to create the shape of a triangle. Alternatively, you can select Create → Shape → Polygon as well. Use “right click” for making the line slant.

15. After the triangle is complete, select Create → Shape → Circle, and then bring the cursor in front of the triangle, and click once. Release the finger and move the mouse. After the circle is generated, click once again to place it.
16. Now, click on the output wire and drag it in front of the circle. Similarly, join the output pin to the wire. Later, select Create → Selection box and click on Automatic. The tool will adjust the selection box around the created symbol automatically. You can similarly select and drag the port names to the respective places. Finally, “check & save” and observe the errors.

17. After the schematic and symbol entries, both the editor windows can be closed. Now for creating a test circuit, create a new cellview from the virtuoso console, and give the name as “inverter_test”. When the editor window opens, press “i”, browse for your library and select the inverter symbol which was created earlier. Place it in the middle of the screen.

To ensure that the symbol is loaded correctly, you can click on the symbol and then press “Shift E”. The schematic editor will move one level down, and the inverter’s circuit entered
earlier will be displayed on the screen. To come back to the symbol, press “Ctrl E”; the symbol will be displayed back. Press Esc to unselect the symbol.

18. Place wires at the input and output, and place an output pin as well. These wires are needed during simulation, to plot the voltage waveforms.

20. In the property window, enter \textit{Voltage1} as 0 and \textit{Voltage2} as 1.8. Similarly enter \textit{Period} as 40n and \textit{Pulse width} as 20n, without the space in between. No need to enter the units; they appear automatically. Place the “vpulse” at the input wire. Connect “gnd” at the other end.

21. Similarly, browse for the instance “vdc”, and enter its \textit{DC voltage} as 1.8.
22. Place “vdc” at the front, and connect “vdd” and “gnd” accordingly. Finally “check & save”. The test circuit is complete now, and ready to be simulated.

**Note:** The library *gpdk180* contains the technology dependent components (180nm), and the library *analogLib* contains the technology independent components.

Whenever a component needs to be selected, place the cursor on the component and click on it. The selected component’s boundary turns into magenta color. Now the properties of the component can be verified by pressing “q”, after which the property window opens.

To zoom a particular portion of the screen, right click, hold, and move the mouse. A yellow colored boundary will be drawn on the screen. When the finger is released, the highlighted portion gets zoomed. To come back to the original screen, press “f”. Alternatively, “Ctrl Z” and “Shift Z” can be used, to zoom in and zoom out.

After the symbol is entered, *Shift* and *E* can be used together to move one level down, to view the schematic diagram. Later, *Ctrl* and *E* can be used together to move one level up, to the symbol.

The hot key functions that are used during design entry are summarized as follows –

<table>
<thead>
<tr>
<th>Hot key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Instance</td>
</tr>
<tr>
<td>w</td>
<td>Wire</td>
</tr>
<tr>
<td>p</td>
<td>Pin</td>
</tr>
<tr>
<td>m</td>
<td>Stretch</td>
</tr>
<tr>
<td>q</td>
<td>Property</td>
</tr>
<tr>
<td>r</td>
<td>Rotate</td>
</tr>
<tr>
<td>u</td>
<td>Undo</td>
</tr>
<tr>
<td>c</td>
<td>Copy</td>
</tr>
<tr>
<td>f</td>
<td>Fit to screen</td>
</tr>
<tr>
<td>Esc</td>
<td>Exit</td>
</tr>
</tbody>
</table>
II. **SIMULATION:**

1. In the test circuit’s editor window, click on **Launch → ADE L**. A new window will open.

2. Click on **Analyses → Choose**. A new window will open, in which select “tran”. Fill the stop time as **100n**, and select **liberal**. Later, click on **Apply**.
3. Now select “dc” on the Choosing Analyses window, and click on Save DC Operating Point. Click on the Component Parameter. A Select Component option will pop up.

4. Double click on Select Component. The ADE window gets minimized, and the schematic is shown. Click on the component “vpulse”. In the new window that opens up, click on the top most parameter dc and then click on OK.

5. Ensure that the component name and parameter name are updated. Now in the Sweep range, enter 0 and 1.8 in the Start and Stop options respectively. Later, click on Apply.
6. In the ADE window, ensure that the Analyses fields are updated for **tran** and **dc**.
7. Now to select the stimulus and response points, in the ADE window, click on Outputs → To be plotted → Select on Schematic. In the schematic window, click on input and output wires. These wires will become dotted lines when selected. Later, press Esc.

8. In the ADE window, check that the Outputs fields are updated. Now click on Simulation → Netlist and Run. The waveform window will open and the simulation results are displayed. Transient response is displayed on the left side and DC response on the right.

Click on the transient response waveform and then click on the fourth icon at the top (Strip chart mode). The input and output waveforms are displayed separately. You can “right click” on each waveform, and then edit the properties of the display such as color and appearance. Similarly, the transfer characteristics can be observed at the right hand side.
III. LAYOUT:

1. For preparing the layout of the inverter, all the other windows can be closed, except for the virtuoso console. In the console, open the schematic of the inverter and click on Launch → Layout XL. In the Startup Option, click on OK.

2. In the New File option, the tool selects the view as layout by default. Click on OK.
3. The tool opens the LSW and the Layout suite.

4. Maximize the layout suite and click on Connectivity → Generate → All from Source. A Generate Layout window will open, with default attributes. Click on OK.
5. The layout suite displays a cyan colored box in the first quadrant, which is the Photo-Resist boundary. In addition, in the fourth quadrant, the default layouts of pmos and nmos transistors are displayed, along with four blue squares, which are the nodes - vdd, gnd, input & output.

6. Press “Shift F” to see all the layers within the default layout. Hold the “right click” and move the mouse to zoom a selected portion, and observe the layout carefully. The color details are – **Orange border**: n-well, **Red border**: p-diffusion’s boundary, **Yellow border**: n-diffusion’s boundary, **Green**: diffusion, **Rose**: polysilicon, **Yellow square**: contact cut, **Blue**: metal1.

7. Click on the pmos device and drag it into the PR boundary. The layout can be moved either vertically or horizontally, not diagonally. During this movement, the tool keeps displaying the
connections of the terminals with the nodes. After placing the pmos device, place the nmos device below it. If the space is insufficient, the PR boundary can be enlarged, through the top and the right edges. For this purpose, press “s” and click on the edge of the PR boundary. (“s” is for stretch, in the layout suite). The selected edge will turn into magenta color. Now release the finger and move the mouse till the desired area, and click again. Later, press Esc.

8. After placing the devices, zoom the space in between the transistors. In the LSW, select Poly. Now in the layout suite, press “p”, place the mouse at the middle of the gate’s lower contact of pmos device, and click once. (“p” is for path, in the layout suite). Release the finger and move the mouse downwards. The poly path will move along with the mouse. Move the mouse until the gate area of the nmos device gets overlapped. Bring the cursor exactly to the middle of the path and double click. The poly path between the gates gets realized. The area can be zoomed further, and the devices can be moved, for the exact overlapping of the poly layers.
9. In the LSW, select Metal1. Using the same procedure, draw the paths for “vdd” at the top and “gnd” at the bottom. Later, using the same metal path, connect the source of pmos device to “vdd” and that of nmos device to “gnd”. Finally, connect both the drains for the output path.

10. Now move to the fourth quadrant where the four blue squares are displayed. Click on one of them; it will turn into magenta color. Press “q”, and then click on Connectivity, to see its properties. If it is vdd, drag it and place it on the upper metal path. Later, place the gnd on the lower path; similarly, place the output pin. Now, place the input pin in front of the poly and connect through a poly path.
11. Now, for connecting the input metal pin to the poly path, a via needs to be placed. Hence, in the layout suite click **Create → Via**. In the Via Definition pull-down menu, select the via M1_POLY1. Click on Hide, and place the via on the input pin. Press Esc.

12. Similarly, for the substrate connections, select the via M1_NWELL and place it touching the n-well, and connect it to “vdd” through a metal path. Later, place the via M1_PSUB on the “vss” path, for the substrate connection of nmos device; the **Black** background itself indicates the p-substrate. (p-device resides on n-well and n-device resides directly on p-substrate).
13. As the layout is now complete, its verification can be performed. In the layout suite, click on **Assura → Run DRC**. Give the run name as “inverter” and verify the output. If there are errors, the tool will highlight those areas in **White** color. The errors will be displayed in the ELW, and the location of each error can be known, by selecting the error in ELW, and then clicking on the arrow mark available in ELW. Correct those errors and rerun DRC.

14. After the DRC check, click on **Assura → Run LVS**, and verify the output. Correct the errors.
15. After the LVS check, click on Assura → Run RCX. Click OK on the form that appears.

16. After the RCX is run, the output is saved in your library as av_extracted. In the virtuoso console, open the “inverter” file with view as av_extracted, and observe the output. The layout can be enlarged and the parasitic components can be observed. Each components value can be checked, by selecting the component and pressing “q”.

If the parasitic component values are beyond the limits, then the layout can be optimized in the layout suite, for the reduction of the parasitic component values; later on, the layout can be back-annotated with the existing parasitic components, and simulation can be performed, for verifying the output.
RESULT:
   a. The schematic for the inverter is drawn and verified the following: DC Analysis, Transient Analysis
   b. The Layout for the inverter is drawn and verified the DRC, LVS, RC Extraction.
Experiment No: 09 Date:

COMMON SOURCE AMPLIFIER

AIM:
To simulate the schematic of the common source amplifier, and then to perform the physical verification for the layout of the same.

TOOL REQUIRED:
Cadence Tool

THEORY:
In electronics, a common-source amplifier is one of three basic single-stage field-effect transistor (FET) amplifier topologies, typically used as a voltage or transconductance amplifier. The easiest way to tell if a FET is common source, common drain, or common gate is to examine where the signal enters and leaves. The remaining terminal is what is known as "common". In this example, the signal enters the gate, and exits the drain. The only terminal remaining is the source. This is a common-source FET circuit. The analogous bipolar junction transistor circuit is the common-emitter amplifier.

The common-source (CS) amplifier may be viewed as a transconductance amplifier or as a voltage amplifier. (See classification of amplifiers). As a transconductance amplifier, the input voltage is seen as modulating the current going to the load. As a voltage amplifier, input voltage modulates the amount of current flowing through the FET, changing the voltage across the output resistance according to Ohm's law. However, the FET device's output resistance typically is not high enough for a reasonable transconductance amplifier (ideally infinite), nor low enough for a decent voltage amplifier (ideally zero). Another major drawback is the amplifier's limited high-frequency response.

CIRCUIT DIAGRAM:

1. Circuit diagram in schematic entry
2. Design entry for the test circuit

<table>
<thead>
<tr>
<th>Library name</th>
<th>Cellview name</th>
<th>Properties/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>myDesgnLib</td>
<td>cs_amplifier</td>
<td>Symbol</td>
</tr>
<tr>
<td>analogLib</td>
<td>vsin</td>
<td>Define pulse specification as AC Magnitude=1; DC Voltage=0; Offset Voltage=0; Amplitude=5m; Frequency=1K</td>
</tr>
<tr>
<td>analogLib</td>
<td>vdd,vss,gnd</td>
<td>vdd=2.5; vss=-2.5; vbias=-2.5</td>
</tr>
</tbody>
</table>

3. Output waveform
RESULT:

a. The schematic for the common source amplifier is drawn and verified the following: DC Analysis, DC Analysis, Transient Analysis.

b. The Layout for the common source amplifier is drawn and verified the DRC, LVS, RC Extraction.
Experiment No: 10

COMMON DRAIN AMPLIFIER

AIM:
To simulate the schematic of the common drain amplifier, and then to perform the physical verification for the layout of the same.

TOOL REQUIRED:
Cadence Tool

THEORY:
Common drain amplifier is a source follower or buffer amplifier circuit using a MOSFET. The output is simply equal to the input minus about 2.2V. The advantage of this circuit is that the MOSFET can provide current and power gain; the MOSFET draws no current from the input. It provides low output impedance to any circuit using the output of the follower, meaning that the output will not drop under load.

Its output impedance is not as low as that of an emitter follower using a bipolar transistor (as you can verify by connecting a resistor from the output to -15V), but it has the advantage that the input impedance is infinite. The MOSFET is in saturation, so the current across it is determined by the gate-source voltage. Since a current source keeps the current constant, the gate-source voltage is also constant.

CIRCUIT DIAGRAM:
1. Circuit diagram in schematic entry

![Circuit Diagram](image-url)
2. Design entry for the test circuit

<table>
<thead>
<tr>
<th>Library name</th>
<th>Cellview name</th>
<th>Properties/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>myDesignLib</td>
<td>cd_amplifier</td>
<td>Symbol</td>
</tr>
<tr>
<td>analogLib</td>
<td>vsin</td>
<td>Define pulse specification as AC Magnitude=1; DC Voltage=0; Offset Voltage=0; Amplitude=5m; Frequency=1K</td>
</tr>
<tr>
<td>analogLib</td>
<td>vdd,vss,gnd</td>
<td>vdd=2.5; vss= -2.5;</td>
</tr>
</tbody>
</table>

3. Test circuit
RESULT:

c. The schematic for the common drain amplifier is drawn and verified the following: DC Analysis, DC Analysis, Transient Analysis.
d. The Layout for the common drain amplifier is drawn and verified the DRC, LVS, RC Extraction.
Experiment No: 11

Differential Amplifier

AIM:
To simulate the schematic of the differential amplifier, and then to perform the physical verification for the layout of the same.

TOOL REQUIRED:
Cadence Tool

THEORY:
The differential amplifier is probably the most widely used circuit building block in analog integrated circuits, principally op amps. We had a brief glimpse at one back in Chapter 3 section 3.4.3 when we were discussing input bias current. The differential amplifier can be implemented with BJTs or MOSFETs. A differential amplifier multiplies the voltage difference between two inputs \((V_{in}^+ - V_{in}^-)\) by some constant factor \(A_d\), the differential gain. It may have either one output or a pair of outputs where the signal of interest is the voltage difference between the two outputs. A differential amplifier also tends to reject the part of the input signals that are common to both inputs \((V_{in}^+ + V_{in}^-)/2\). This is referred to as the common mode signal.

CIRCUIT DIAGRAM:
1. Circuit diagram in schematic entry
2. Design entry for the test circuit

<table>
<thead>
<tr>
<th>Library name</th>
<th>Cellview name</th>
<th>Properties/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>myDesugnLib</td>
<td>Diff_amplifier</td>
<td>Symbol</td>
</tr>
<tr>
<td>analogLib</td>
<td>vsin</td>
<td>Define pulse specification as AC Magnitude=1; DC Voltage=0; Offset Voltage=0; Amplitude=5m; Frequency=1K</td>
</tr>
<tr>
<td>analogLib</td>
<td>vdd,vss,gnd</td>
<td>vdd=2.5; vss=-2.5;</td>
</tr>
<tr>
<td>analogLib</td>
<td>Idc</td>
<td>Dc current = 30u</td>
</tr>
</tbody>
</table>

3. Test circuit

![Test circuit diagram](image-url)
4. Output waveform

RESULT:
   a. The schematic for the differential amplifier is drawn and verified the following: DC Analysis, DC Analysis, Transient Analysis.
   b. The Layout for the differential amplifier is drawn and verified the DRC, LVS, RC Extraction.
Experiment No: 12  

OPERATIONAL AMPLIFIER

AIM:
To simulate the schematic of the operational amplifier, and then to perform the physical verification for the layout of the same.

TOOL REQUIRED:
Cadence Tool

THEORY:
An operational amplifier (often op-amp or opamp) is a DC-coupled high-gain electronic voltage amplifier with a differential input and, usually, a single-ended output.\[^{[1]}\] In this configuration, an op-amp produces an output potential (relative to circuit ground) that is typically hundreds of thousands of times larger than the potential difference between its input terminals. Operational amplifiers had their origins in analog computers, where they were used to perform mathematical operations in many linear, non-linear and frequency-dependent circuits. The popularity of the op-amp as a building block in analog circuits is due to its versatility. Due to negative feedback, the characteristics of an op-amp circuit, its gain, input and output impedance, bandwidth etc. are determined by external components and have little dependence on temperature coefficients or manufacturing variations in the op-amp itself.

Op-amps are among the most widely used electronic devices today, being used in a vast array of consumer, industrial, and scientific devices. Many standard IC op-amps cost only a few cents in moderate production volume; however some integrated or hybrid operational amplifiers with special performance specifications may cost over $100 US in small quantities. Op-amps may be packaged as components, or used as elements of more complex integrated circuits. The op-amp is one type of differential amplifier.

The amplifier's differential inputs consist of a non-inverting input (+) with voltage $V_+$ and an inverting input (−) with voltage $V_-$; ideally the op-amp amplifies only the difference in voltage between the two, which is called the differential input voltage. The output voltage of the op-amp $V_{\text{out}}$ is given by the equation:

$$V_{\text{out}} = A_{\text{OL}}(V_+ - V_-)$$

where $A_{\text{OL}}$ is the open-loop gain of the amplifier (the term "open-loop" refers to the absence of a feedback loop from the output to the input).
CIRCUIT DIAGRAM:

1. Circuit diagram in schematic entry

2. Design entry for the test circuit

<table>
<thead>
<tr>
<th>Library name</th>
<th>Cellview name</th>
<th>Properties/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>myDesignLib</td>
<td>Diff_amplifier</td>
<td>Symbol</td>
</tr>
<tr>
<td>analogLib</td>
<td>vsin</td>
<td>Define pulse specification as AC Magnitude=1; DC Voltage=0; Offset Voltage=0; Amplitude=5m; Frequency=1K</td>
</tr>
<tr>
<td>analogLib</td>
<td>vdd,vss,gnd</td>
<td>vdd=2.5; vss=-2.5;</td>
</tr>
<tr>
<td>analogLib</td>
<td>Idc</td>
<td>Dc current = 30u</td>
</tr>
</tbody>
</table>
3. Test circuit

4. Output waveform
RESULT:

a. The schematic for the operational amplifier is drawn and verified the following: DC Analysis, DC Analysis, Transient Analysis.

b. The Layout for the operational amplifier is drawn and verified the DRC, LVS, RC Extraction.
6. VIVA QUESTIONS

1. Why don’t we use just one NMOS or PMOS transistor as a transmission gate?
2. What are set up time & hold time constraints? What do they signify?
3. Explain Clock Skew?
4. Why is NAND gate preferred over NOR gate for fabrication?
5. What is Body Effect?
6. Why is the substrate in NMOS connected to Ground and in PMOS to VDD?
7. What is the fundamental difference between a MOSFET and BJT?
8. Why PMOS and NMOS are sized equally in a Transmission Gate?
9. What happens when the PMOS and NMOS are interchanged with one another in an inverter?
10. Why are pMOS transistor networks generally used to produce high signals, while nMOS networks are used to produce low signals?
11. What is Latch Up? Explain Latch Up with cross section of a CMOS Inverter. How do you avoid Latch Up?
13. What is DRC?
14. What is LVS?
15. What is RCX?
16. What are the differences between SIMULATION and SYNTHESIS?
17. What is a counter?
18. What are the differences between flipflop and latch?
19. How can you convert JK flipflop into Jk?
20. What are different types of adders?
21. Give the excitation table for JK flipflop?
22. Give the excitation table for SR flipflop?
23. Give the excitation table for D flipflop?
24. Give the excitation table for T flipflop?
25. What is the race around condition?
26. What is an amplifier?
27. What is an op-amp?
28. What is differential amplifier?
29. What is elaboration?
30. What is transient analysis?
31. What is DC analysis?
32. What is AC analysis?