



REVA
UNIVERSITY

Bengaluru, India

CMOS VLSI Laboratory Manual

VI Semester
BTEC15F6300
2017-21



School of Electronics and
Communication Engineering

School of Electronics and Communication Engineering

Vision of the School

The School of Electronics and Communication Engineering is envisioned to be a leading centre of higher learning with academic excellence in the field of electronics and communication engineering blended by research and innovation in tune with changing technological and cultural challenges supported with leadership qualities, ethical and moral values.

Mission of the School

M1 Establish a unique learning environment to enable the students to face the challenges in the field of Electronics and Communication Engineering and explore multidisciplinary which serve the societal requirements.

M2 Create state-of-the-art laboratories, resources, and exposure to the current industrial trends to enable students to develop skills for solving complex technological problems of current times and provide a framework for promoting collaborative and multidisciplinary activities.

M3 Promote the establishment of Centres of Excellence in niche technology areas to nurture the spirit of innovation and creativity among faculty and students.

M4 Offer ethical and moral value-based education by promoting activities which inculcate the leadership qualities, patriotism and set high benchmarks to serve the society

Program Educational Objectives

The Program Educational Objectives of B. Tech in Electronics and Communication Engineering are as follows:

PEO -1: To have successful professional careers in industry, government, academia, and military as innovative engineers.

PEO -2: To successfully solve engineering problems associated with the lifecycle of Electronics and Communication Systems either leading a team or as a team member.

PEO -3: To continue to learn and advance their careers through activities such as participation in professional organizations, attainment of professional certification for lifelong learning and seeking higher education.

PEO -4: To be active members ready to serve the society locally and internationally and will undertake entrepreneurship for the growth of economy and to generate employment.

BTEC15F6300	CMOS VLSI	L	T	P	C
Duration :16 Wks		3	0	1	4

COURSE CONTENT

PART-A

Write Verilog code for the following circuits and their test bench for verification, observe the waveform and synthesize the code with technology library with given constraints. Usage of tool can be demonstrated by taking: nmos and pmos ID vs. VDS characteristics.

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

PART-B

1. Design a CMOS Inverter circuit with the given specifications.
Draw the schematic and verify DC Analysis, Transient Analysis.
Draw the Layout and verify the DRC, LVS and Extract RC.

COURSE OBJECTIVES

1. Be able to understand the ASIC design flow.
2. Be able to write switch level coding for digital circuits.
3. Be able to develop the digital circuits at different abstract levels.
4. Be able to design CMOS combinational and sequential logic at the transistor level, with layout.
5. Compare the performance analysis based on area, power and delay for the digital circuits.

COURSE OUTCOMES

On the successful completion of CMOS VLSI Lab, the student shall be able to:

1. Acquire knowledge on modern engineering tools design.
2. Examine MOSFETs functionality through experimentation.
3. Demonstrate logic circuits digitally.
4. Construct digital system using the basic digital blocks.
5. Distinguish between various circuit implementation techniques through experimentation.

Challenge experiment:

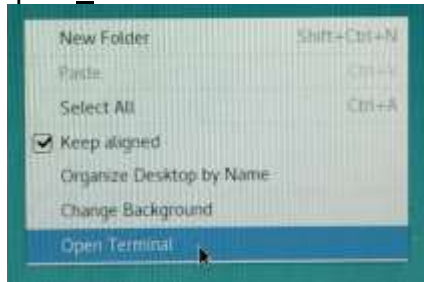
1. Write a Verilog description for 4X4 bit booth algorithm and its test bench for verification, observe the waveform.
2. Design a 3 bit resistor string DAC.

SI No.	Description	Page No.
1	General procedure	5
	List of Programs	
1	Write Verilog Code for the Inverter circuit and its Test Bench for verification, perform switch level simulation and observe the waveform.	8
2	Write Verilog Code for the Buffer circuit and its Test Bench for verification, perform switch level simulation and observe the waveform.	9
3	Write Verilog Code for the Transmission Gate and its Test Bench for verification, perform switch level simulation and observe the waveform.	10
4	Write Verilog Code for the NAND Gate and its Test Bench for verification, perform switch level simulation and observe the waveform.	11
5	Write Verilog Code for the NOR Gate and its Test Bench for verification, perform switch level simulation and observe the waveform.	12
6	Write Verilog Code for the OR Gate and its Test Bench for verification, perform switch level simulation and observe the waveform.	13
7	Write Verilog Code for the AND Gate and its Test Bench for verification, perform switch level simulation and observe the waveform.	14
8	Write Verilog Code for the XOR Gate and its Test Bench for verification, perform switch level simulation and observe the waveform.	15
9	Write Verilog Code for the XNOR Gate and its Test Bench for verification, perform switch level simulation and observe the waveform.	16
10	Write Verilog Code for the RS flip flop and its Test Bench for verification, perform simulation, observe the waveform and synthesize.	17
11	Write Verilog Code for the D flip flop and its Test Bench for verification, perform simulation, observe the waveform and synthesize.	19
12	Write Verilog Code for the T flip flop and its Test Bench for verification, perform simulation, observe the waveform and synthesize.	20
13	Write Verilog Code for the JK flip flop and its Test Bench for verification, perform simulation, observe the waveform and synthesize.	21
14	Write Verilog Code for the MS JK flip flop and its Test Bench for verification, perform simulation, observe the waveform and synthesize.	23

15	Write Verilog Code for the Parallel adder and its Test Bench for verification, perform simulation, observe the waveform and synthesize.	25
16	Write Verilog Code for the Asynchronous Counters and its Test Bench for verification, perform simulation, observe the waveform and synthesize.	26
17	Write Verilog Code for the Ripple Counter Structural description and its Test Bench for verification, perform simulation, observe the waveform and synthesize.	27
18	Write Verilog Code for the Ripple Counter and its Test Bench for verification, perform simulation, observe the waveform and synthesize.	28
19	Write Verilog Code for the Synchronous Counters and its Test Bench for verification, perform simulation, observe the waveform and synthesize.	30
20	Write Verilog Code for the Successive Approximation Register [SAR] and its Test Bench for verification, perform simulation, observe the waveform and synthesize.	31
21	Gate level Code	32
22	Waveforms.	34
23	General Notes for analog virtuoso	42
24	Design a CMOS Inverter circuit with the given specifications. Draw the schematic and verify DC Analysis, Transient Analysis. Draw the Layout and verify the DRC, LVS and Extract RC	43
	Challenge experiment:	
25	Write a Verilog description for 4X4 bit booth algorithm and its test bench for verification, observe the waveform.	69
26	Design a 3 bit resistor string DAC	70

General procedure for Digital design:

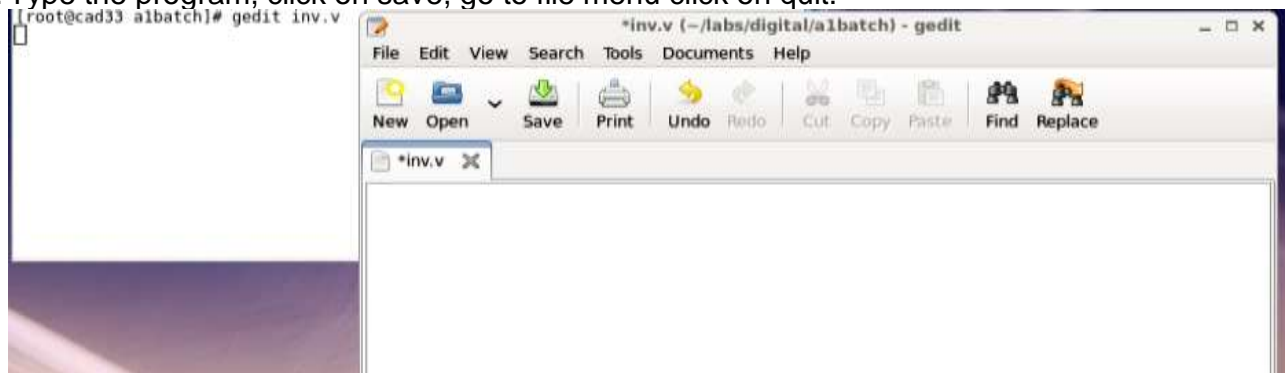
1. Log in to workstation using not listed?
2. User name: **root**
3. Password: **root123**
4. Right click on the Desktop
5. Click on Open Terminal



6. Type **csch** <press enter button>
7. Type **source cschrc** <press enter button> the **"Welcome to cadence tools"** will display in the next line.
8. Type **cd labs** <press enter button> changes the directory to labs (folder).
9. Type **cd digital** <press enter button> changes the directory to digital(folder)
10. Type **cd <batch_name>** (ex: a1batch, b1batch etc..) <press enter button> changes to directory_name(folder)
11. Type **gedit <file_name.v>** <press enter button> (file name followed with the extension **.v(dot v)** ex: **inv.v**)
repeat the step 11 to create test bench **gedit <file_name_test.v>**

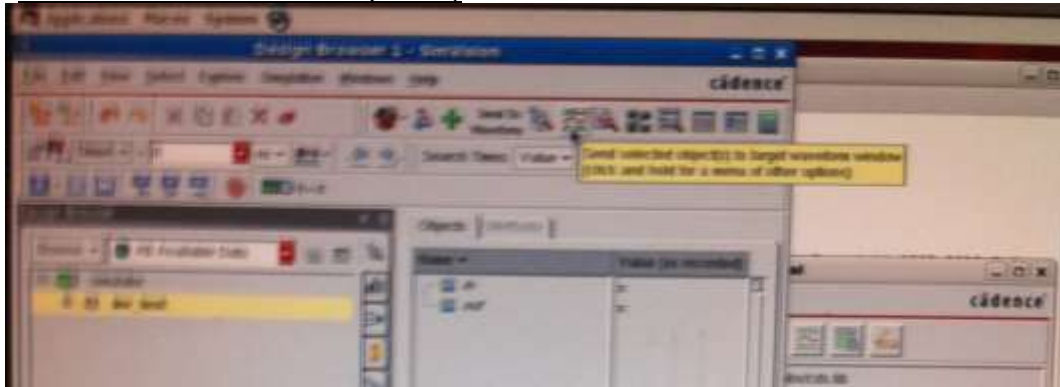


12. Type the program, click on save, go to file menu click on quit.

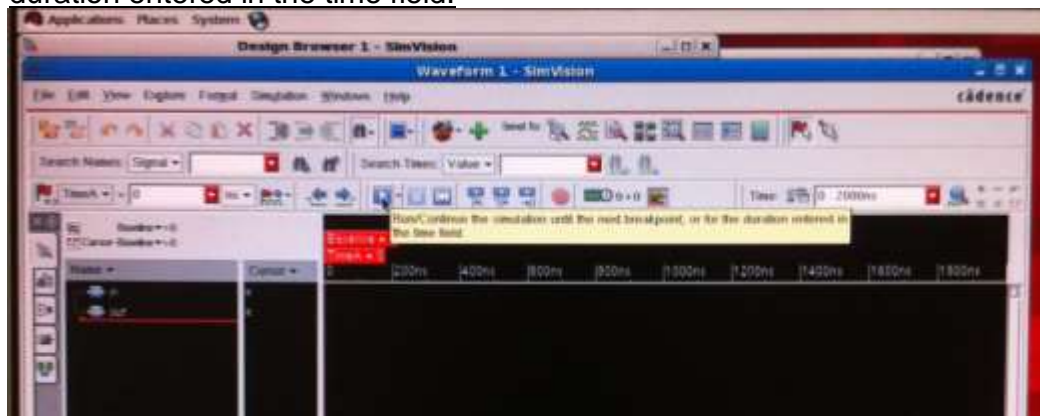


Procedure for simulation:**command for simulation:****irun <filename.v> -access +rwc -mess****irun <filename_test.v> -access +rwc -mess -gui**

1. Click on the **icon** send selected object(s) to target waveform window (click and hold for a menu of other options)



2. Click on the **icon** Run/Continue the simulation until the next breakpoint, or for the duration entered in the time field.



3. The o/p wave form window opens.



Procedure for synthesis (gate level only):

(top level program and test bench should be in separate file)

close all the windows except terminal window, then in terminal window type the following:

```
genus
set_db library slow_normal.lib
read_hdl <filename.v> (top level program)
elaborate
read_sdc constraints.sdc (optional for combinational circuits)
synthesize -to_mapped -effort medium
report area
report timing -unconstrained (for gates and combinational circuits)
report timing (for sequential circuits)
report power
report gates
write_hdl > <filename_netlist.v>
gui_show
gui_hide
exit
#####
# slow_normal.lib file should be in the same folder where the .v files are kept.
# constraints_top.g file(which is in /rclabs/work) is to be renamed as constraints.sdc and this
# file should be kept in the same folder where the .v files are kept, and then edit the i/o pins,
# clk etc. in this file.
# for synthesize the top module file and test bench should be in different files.
#####
the constraints file contents are:

create_clock -name clk -period 10 -waveform {0 5} [get_ports "clk"]
set_clock_transition -rise 0.1 [get_clocks "clk"]
set_clock_transition -fall 0.1 [get_clocks "clk"]
set_clock_uncertainty 1.0 [get_ports "clk"]
set_input_delay -max 1.0 [get_ports "M"] -clock [get_clocks "clk"]
set_input_delay -max 1.0 [get_ports "rst"] -clock [get_clocks "clk"]
set_input_delay -max 1.0 [get_ports "rw"] -clock [get_clocks "clk"]
set_input_delay -max 1.0 [get_ports "data_in"] -clock [get_clocks "clk"]
set_input_delay -max 1.0 [get_ports "addr_load"] -clock [get_clocks "clk"]
set_input_delay -max 1.0 [get_ports "load"] -clock [get_clocks "clk"]
set_output_delay -max 1.0 [get_ports "data_out"] -clock [get_clocks "clk"]
```

in place of "M" "rst" "rw" "data_in" "addr_load" "load" are the editable names as for ex: dff:- M-
clk, rst-n_rst, data_in-din, data_out-q like this we have to comment the line if the name is not
there.

1. Write Verilog Code for the Inverter circuit and its Test Bench for verification, perform switch level simulation and observe the waveform.

```

module inverter (out, in);

// Declarations of I/O, Power and Ground Lines

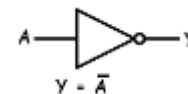
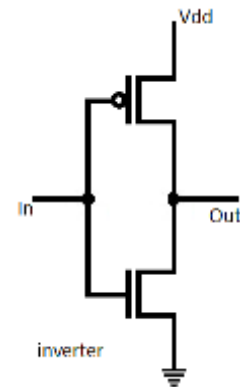
output out;
input in;
supply1 pwr;
supply0 gnd;

// Instantiate pmos and nmos switches

pmos (out,pwr,in);
nmos (out,gnd,in);

endmodule

```



TEST BENCH

```

module inv_test;
  wire out ;
  reg in ;

// Instantiate inverter Module

  Inverter i1(out, in) ;

// Apply Stimulus

  initial
  begin
    in = 1'b0 ; #10 ;
    in = 1'b1 ; #10 ;
    in = 1'bx ; #10 ;
    in = 1'bz ; #10 ;
  end

endmodule

```

Truth Table	
Input	Output
0	1
1	0
x	x
z	x

2. Write Verilog Code for the Buffer circuit and its Test Bench for verification, perform switch level simulation and observe the waveform.

```

module inverter (Y, A);

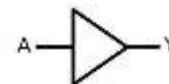
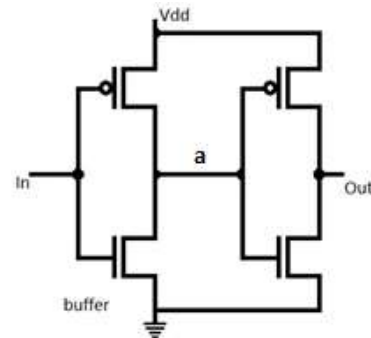
// Declarations of I/O ,Power and Ground Lines
output Y;
input A;

supply1 pwr;
supply0 gnd;

// Instantiate pmos and nmos switches
pmos (Y,pwr,A);
nmos (Y,gnd,A);

endmodule

```



```

// Define our own Buffer

```

```

module buffer (out, in);

```

```

    output out;
    input in;
    wire a;

```

```

    inverter i1 (a,in);
    inverter i2 (out,a);
endmodule

```

TEST BENCH

```

module buf_test;

```

```

    wire out ;
    reg in ;

```

```

// Instantiate Buffer Module
    buffer b1 ( out, in);

```

```

// Apply Stimulus

```

```

    initial
    begin
        in = 1'b0 ; #10 ;
        in = 1'b1 ; #10 ;
        in = 1'bx ; #10 ;
        in = 1'bz ; #10 ;
    end
endmodule

```

Truth Table	
Input	Output
0	0
1	1
x	x
z	x

3. Write Verilog Code for the Transmission Gate and its Test Bench for verification, perform switch level simulation and observe the waveform.

```
module trangate (out, in, cntrl1, cntrl2);
```

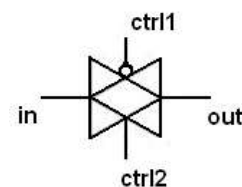
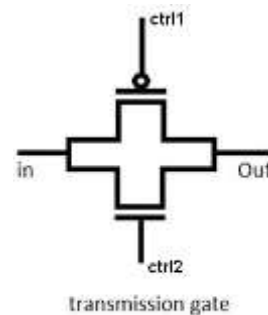
```
// Declarations of I/O and Control Lines
```

```
output out;
input in;
input cntrl1,cntrl2;
```

```
// Instantiate pmos and nmos switches
```

```
pmos (out,in,cntrl1);
nmos (out,in,cntrl2);
```

```
endmodule
```



TEST BENCH

```
module trangate_test;
```

```
wire out ;
reg in ;
reg cntrl1,cntrl2;
```

```
// Instantiate trangate Module
trangate t1(out, in, cntrl1, cntrl2) ;
```

```
// Apply Stimulus
```

```
initial
begin
in = 1'b0 ; cntrl1 = 1'b0 ; cntrl2 = 1'b1 ; #10 ;
in = 1'b0 ; cntrl1 = 1'b1 ; cntrl2 = 1'b0 ; #10 ;
in = 1'b1 ; cntrl1 = 1'b0 ; cntrl2 = 1'b1 ; #10 ;
in = 1'b1 ; cntrl1 = 1'b1 ; cntrl2 = 1'b0 ; #10 ;
end
endmodule
```

CONTROL		INPUT		OUTPUT	
pMOS	nMOS	IN		OUT	
0	1	0		0	
		1		1	
1	0	0		Z	
		1			

4. Write Verilog Code for the NAND Gate and its Test Bench for verification, perform switch level simulation and observe the waveform.

```

module nandgate (out, in1, in2);

// Declarations of I/O ,Power and Ground Lines

output out;
input in1,in2;
supply1 pwr;
supply0 gnd;

// Declaration of Wire

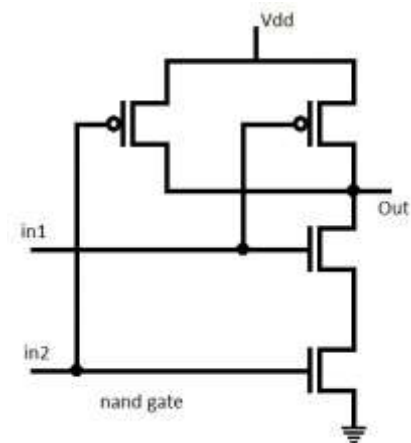
wire contact;

// Instantiate pmos and nmos switches

pmos (out,pwr,in1);
pmos (out,pwr,in2);
nmos (out,contact,in1);
nmos (contact,gnd,in2);

endmodule

```



TEST BENCH

```

module nand_test;

wire out ;
reg in1,in2 ;

// Instantiate NAND Gate Module

nandgate n1(out, in1, in2) ;

// Apply Stimulus

initial
begin
    in1 = 1'b0 ; in2 = 1'b0 ; #10 ;
    in1 = 1'b0 ; in2 = 1'b1 ; #10 ;
    in1 = 1'b1 ; in2 = 1'b0 ; #10 ;
    in1 = 1'b1 ; in2 = 1'b1 ; #10 ;
end
endmodule

```

Truth Table		
Input		Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

5. Write Verilog Code for the NOR Gate and its Test Bench for verification, perform switch level simulation and observe the waveform.

```
module norgate ( out, in1, in2);
```

```
// Declarations of I/O ,Power and Ground Lines
```

```
output out;
input in1,in2;
supply1 pwr;
supply0 gnd;
```

```
// Declaration of Wire
```

```
wire contact;
```

```
// Instantiate pmos and nmos switches
```

```
pmos (contact,pwr,in1);
pmos (out,contact,in2);
nmos (out,gnd,in1);
nmos (out,gnd,in2);
```

```
endmodule
```

TEST BENCH

```
module nor_test;
```

```
wire out ;
reg in1,in2 ;
```

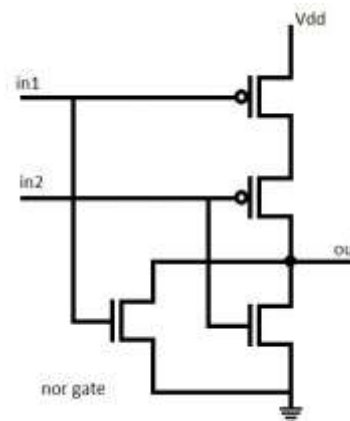
```
// Instantiate NOR Gate Module
```

```
norgate n1( out, in1, in2) ;
```

```
// Apply Stimulus
```

```
initial
begin
in1 = 1'b0 ; in2 = 1'b0 ; #10 ;
in1 = 1'b0 ; in2 = 1'b1 ; #10 ;
in1 = 1'b1 ; in2 = 1'b0 ; #10 ;
in1 = 1'b1 ; in2 = 1'b1 ; #10 ;
end
```

```
endmodule
```



Truth Table		
Input		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

6. Write Verilog Code for the OR Gate and its Test Bench for verification, perform switch level simulation and observe the waveform.

```
module orgate (out , in1, in2);
```

```
output out;
input in1,in2;
supply1 pwr;
supply0 gnd;
```

```
// Declaration of Wires
```

```
wire contact;
wire nout;
```

```
// Instantiate pmos and nmos switches for OR gate
```

```
pmos (contact,pwr,in1);
pmos (nout,contact,in2);
nmos (nout,gnd,in1);
nmos (nout,gnd,in2);
```

```
// Instantiate pmos and nmos switches for NOT gate
```

```
pmos (out,pwr,nout);
nmos (out,gnd,nout);
```

```
endmodule
```

TEST BENCH

```
module or_test;
```

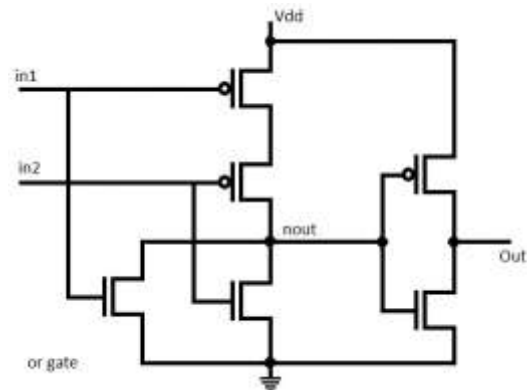
```
wire out ;
reg in1,in2 ;
```

```
orgate n1( out, in1, in2) ;
```

```
// Apply Stimulus
```

```
initial
begin
in1 = 1'b0; in2 = 1'b0; #10;
in1 = 1'b0; in2 = 1'b1; #10;
in1 = 1'b1; in2 = 1'b0; #10;
in1 = 1'b1; in2 = 1'b1; #10;
end
```

```
endmodule
```



Truth Table		
Input		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

7. Write Verilog Code for the AND Gate and its Test Bench for verification, perform switch level simulation and observe the waveform.

```
module andgate (out, in1, in2);
```

```
output out;
input in1,in2;
supply1 pwr;
supply0 gnd;
```

```
// Declaration of Wires
```

```
wire contact;
wire nout
```

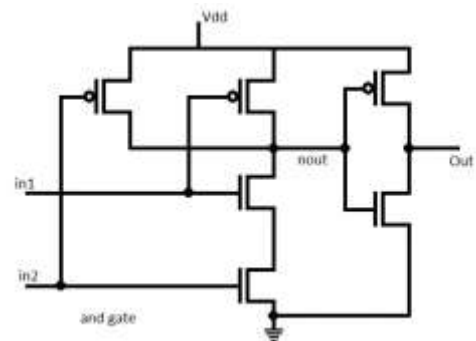
```
// Instantiate pmos and nmos switches to form AND gate
```

```
pmos (nout,pwr,in1);
pmos (nout,pwr,in2);
nmos (nout,contact,in1);
nmos (contact,gnd,in2);
```

```
// Instantiate pmos and nmos switches to form NOT gate
```

```
pmos (out,pwr,nout);
nmos (out,gnd,nout);
```

```
endmodule
```



TEST BENCH

```
module and_test;
```

```
wire out ;
reg in1,in2 ;
```

```
// Instantiate AND gate module
```

```
andgate a1(out, in1, in2) ;
```

```
// Apply Stimulus
```

```
initial
begin
in1 = 1'b0 ; in2 = 1'b0 ; #10 ;
in1 = 1'b0 ; in2 = 1'b1 ; #10 ;
in1 = 1'b1 ; in2 = 1'b0 ; #10 ;
in1 = 1'b1 ; in2 = 1'b1 ; #10 ;
end
```

```
endmodule
```

Truth Table		
Input		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

8. Write Verilog Code for the XOR Gate and its Test Bench for verification, perform switch level simulation and observe the waveform.

```

module xorgate ( out, in1, in2);

// Declarations of I/O ports

output out;
input in1,in2;
wire in2bar;

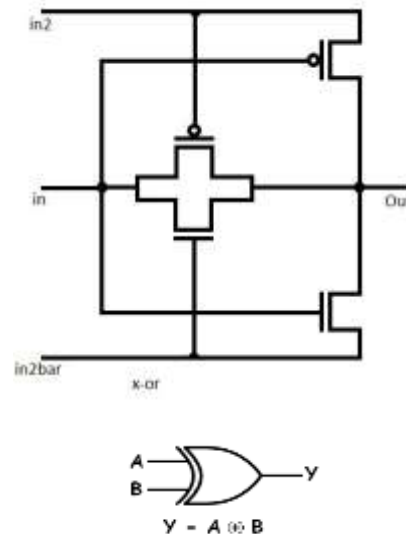
assign in2bar = ~in2;

// Instantiate pmos and nmos switches :

pmos (out,in2,in1);
nmos (out,in2bar,in1);
pmos (out,in1,in2);
nmos (out,in1,in2bar);

endmodule

```



TEST BENCH

```

module xor_test;

wire out ;
reg in1,in2 ;

// Instantiate xorgate Module

xorgate x1 ( out, in1, in2) ;

// Apply Stimulus

initial
begin
    in1 = 1'b0 ; in2 = 1'b0 ; #10 ;
    in1 = 1'b0 ; in2 = 1'b1 ; #10 ;
    in1 = 1'b1 ; in2 = 1'b0 ; #10 ;
    in1 = 1'b1 ; in2 = 1'b1 ; #10 ;
end
endmodule

```

Truth Table		
Input		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

9. Write Verilog Code for the XNOR Gate and its Test Bench for verification, perform switch level simulation and observe the waveform.

```

module xnorgate ( out, in1, in2);

// Declarations of I/O ports

output out;
input in1,in2;
wire in2bar;

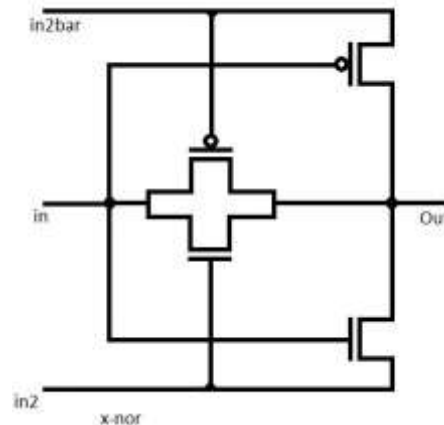
assign in2bar = ~in2;

// Instantiate pmos and nmos switches :

pmos (out,in2bar,in1);
nmos (out,in2,in1);
pmos (out,in1,in2bar);
nmos (out,in1,in2);

endmodule

```



TEST BENCH

```

module xnor_test;

wire out ;
reg in1,in2 ;

// Instantiate xnor gate Module

xnorgate x1 (out, in1, in2) ;

// Apply Stimulus

initial
begin
    in1 = 1'b0 ; in2 = 1'b0 ; #10 ;
    in1 = 1'b0 ; in2 = 1'b1 ; #10 ;
    in1 = 1'b1 ; in2 = 1'b0 ; #10 ;
    in1 = 1'b1 ; in2 = 1'b1 ; #10 ;
end
endmodule

```

Truth Table		
Input		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

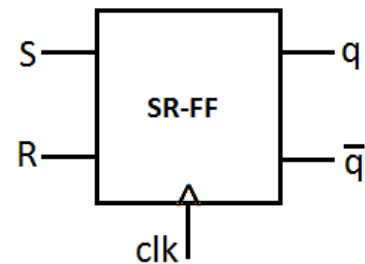
10. Write Verilog Code for the SR flip flop and its Test Bench for verification, perform simulation, observe the waveform and synthesize.

```

module SR_ff(q,qbar,s,r,clk);
output q,qbar;
input clk,s,r;
reg tq;

always @(posedge clk or tq)
begin
    if (s == 1'b0 && r == 1'b0)
        tq <= tq;
    else if (s == 1'b0 && r == 1'b1)
        tq <= 1'b0;
    else if (s == 1'b1 && r == 1'b0)
        tq <= 1'b1;
    else if (s == 1'b1 && r == 1'b1)
        tq <= 1'bx;
end
assign q = tq;
assign qbar = ~tq;
endmodule

```



TEST BENCH

```

module SR_ff_test;
reg clk,s,r;
wire q,qbar;

    SR_ff sr1(q,qbar,s,r,clk);

    initial
        clk = 1'b0;

    always
        #10 clk = ~clk;

    initial
    begin
        s = 1'b0; r = 1'b0;
        #30 s = 1'b1;
        #29 s = 1'b0;
        #1 r = 1'b1;
        #30 s = 1'b1;
        #30 r = 1'b0;
        #20 s = 1'b0;
        #19 s = 1'b1;
    end
endmodule

```

Truth Table					
clk	S	R	q	qbar	
	0	0	Previous state		
	0	1	0	1	reset
	1	0	1	0	set
	1	1	forbidden		

```
        #200 s = 1'b1; r = 1'b1;
            #50 s = 1'b0; r = 1'b0;
            #50 s = 1'b1; r = 1'b0;
            #10 ;
    end

    initial
        #500 $finish;

endmodule
```

11. Write Verilog Code for the D flip flop and its Test Bench for verification, perform simulation, observe the waveform and synthesize.

```

module d_ff(q,clk,n_rst,din);
    output q;
    input clk,din,n_rst;
    reg q;
    always @(posedge clk or negedge n_rst)
    begin
        if(!n_rst)
            q <= 1'b0;
        else
            q <= din;
    end
endmodule

```

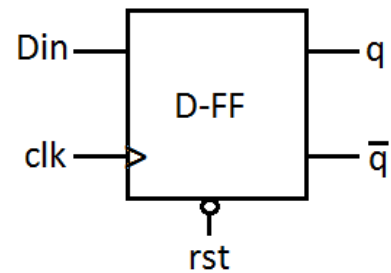
TEST BENCH

```

module d_ff_test;
    reg clk, din, n_rst;
    wire q;
    d_ff df1 (q, clk, n_rst, din);

    initial
        clk = 1'b0;
    always
        #10 clk = ~clk;
    initial
    begin
        din = 1'b0;
        n_rst = 1'b1;
        #20 n_rst = 1'b0;
        #10 din = 1'b1;
        #20 n_rst = 1'b1;
        #18 din = 1'b0;
        #1 din = 1'b1;
        #20 din = 1'b0;
        #10 ;
    end
    initial
        #100 $finish;
endmodule

```



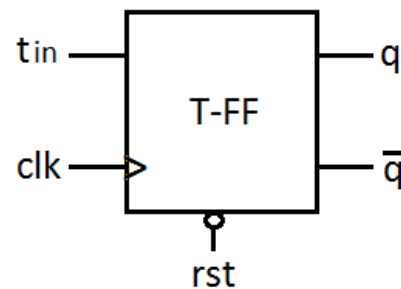
Truth Table				
rst	clk	Din	q	qbar
0		x	0	1
1		1	1	0
1		0	0	1

12. Write Verilog Code for the T flip flop and its Test Bench for verification, perform simulation, observe the waveform and synthesize.

```

module t_ff(q,qbar,clk,tin,rst);
    output q,qbar;
    input clk,tin,rst;
    reg tq;
    always @(posedge clk or negedge rst)
    begin
        if(!rst)
            tq <= 1'b0;
        else
            begin
                if (tin)
                    tq <= ~tq;
            end
        end
        assign q = tq;
        assign qbar = ~q;
    end
endmodule

```



Truth Table			
rst	clk	t in	q
0		x	0
1		0	q
1		1	toggle

TEST BENCH

```

module t_ff_test;
    reg clk,tin,rst;
    wire q,qbar;
    t_ff t1(q,qbar,clk,tin,rst);
    initial
        clk = 1'b0;
    always
        #10 clk = ~clk;
    initial
    begin
        rst = 1'b0; tin = 1'b0;
        #30 rst = 1'b1;
        #10 tin = 1'b1;
        #205 tin = 1'b0;
        #300 tin = 1'b1;
        #175 tin = 1'b0;
        #280 rst = 1'b0;
        #20 rst = 1'b1;
        #280 tin = 1'b1;
        #10 ;
    end
    initial
        #2000 $finish;
endmodule

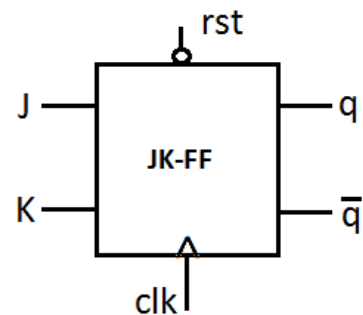
```

13. Write Verilog Code for the JK flip flop and its Test Bench for verification, perform simulation, observe the waveform and synthesize.

```
module jk_ff(q,qbar,clk,rst,j,k);
```

```
input clk,rst,j,k;
output q,qbar;
reg q,tq;
```

```
always @(posedge clk or negedge rst)
begin
if (!rst)
begin
q <= 1'b0;
tq <= 1'b0;
end
else
begin
if (j == 1'b1 && k == 1'b0)
q <= j;
else if (j == 1'b0 && k == 1'b1)
q <= 1'b0;
else if (j == 1'b1 && k == 1'b1)
begin
tq <= ~tq;
q <= tq;
end
end
end
assign qbar = ~q;
endmodule
```



Truth Table					
clk	rst	J	K	q	qbar
	0	x	x	0	1
	1	0	0	previous state	
	1	0	1	0	1
	1	1	0	1	0
	1	1	1	toggle	

TEST BENCH

```
module jk_ff_test;
reg clk,rst,j,k;
wire q,qbar;
jk_ff inst(q,qbar,clk,rst,j,k);
initial
clk=1'b0;
always
#10 clk=~clk;
initial
begin
j=1'b0;k=1'b0;rst=1'b0;
#10 rst=1'b1;
#10 j=1'b0; k=1'b0;
#10 j=1'b0; k=1'b1;
#10 j=1'b1; k=1'b0;
#50 j=1'b1; k=1'b1;
#10 j=1'b0; k=1'b0;
#50 j=1'b1; k=1'b1;
#10;
end
initial
#300 $finish;
endmodule
```


14. Write Verilog Code for the MS JK flip flop and its Test Bench for verification, perform simulation, observe the waveform and synthesize.

```
module ms_jkff(q,q_bar,clk,j,k);
    output q,q_bar;
    input clk,j,k;
    reg tq,q,q_bar;

    always @(clk)
    begin
        if (!clk)
            begin
                if (j==1'b0 && k==1'b1)
                    tq <= 1'b0;
                else if (j==1'b1 && k==1'b0)
                    tq <= 1'b1;
                else if (j==1'b1 && k==1'b1)
                    tq <= ~tq;
            end
        if (clk)
            begin
                q <= tq;
                q_bar <= ~tq;
            end
        end
    end
endmodule
```

TEST BENCH

```
module tb_ms_jkff;
    reg clk,j,k;
    wire q,q_bar;

    ms_jkff inst(q,q_bar,clk,j,k);

    initial
        clk = 1'b0;

    always #10
        clk = ~clk;

    initial
    begin
        j = 1'b0; k = 1'b0;
        #60 j = 1'b0; k = 1'b1;
        #40 j = 1'b1; k = 1'b0;
        #20 j = 1'b1; k = 1'b1;
        #40 j = 1'b1; k = 1'b0;
        #5 j = 1'b0; #20 j = 1'b1;
        #10;
    end

    initial
        #250 $finish;
endmodule
```

15. Write Verilog Code for the Parallel adder and its Test Bench for verification, perform simulation, observe the waveform and synthesize.

// FULL ADDER

```
module fulladd (cin,x,y,s,cout);
  input cin,x,y;
  output s,cout;

  assign s = x^y^cin;
  assign cout =( x & y) | ( x & cin) |( y & cin);

endmodule
```

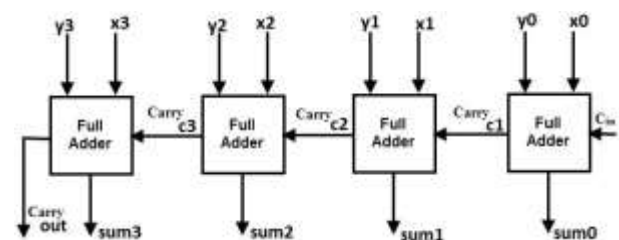
// PARALLEL ADDER

```
module adder4 ( carryin,x,y,sum,carryout);

  input carryin;
  input [3:0] x,y;
  output [3:0] sum;
  output carryout;

  fulladd stage0 (carryin,x[0],y[0],sum[0],c1);
  fulladd stage1 (c1,x[1],y[1],sum[1],c2);
  fulladd stage2 (c2,x[2],y[2],sum[2],c3);
  fulladd stage3 (c3,x[3],y[3],sum[3],carryout);

endmodule
```



TEST BENCH

```
module adder4_t ;
  reg [3:0] x,y;
  reg carryin;
  wire [3:0] sum;
  wire carryout;

  adder4 a1 ( carryin,x,y,sum,carryout);

  initial
  begin
    x = 4'b0000; y= 4'b0000;carryin = 1'b0;
    #20 x =4'b1111; y = 4'b1010;
    #40 x =4'b1011; y =4'b0110;
    #40 x =4'b1111; y=4'b1111;
    #150 $finish;
  end
endmodule
```

16. Write Verilog Code for the Asynchronous Counters and its Test Bench for verification, perform simulation, observe the waveform and synthesize.

D FF

```
module D_FF (q, d, clk, reset) ;

output q;
input d, clk, reset;
reg q;

always @(posedge reset or negedge clk)
if (reset)
    q <= 1'b0;
else
    q <= d;
endmodule
```

T FF

```
module T_FF(q, clk, reset);

output q;
input clk, reset;
wire d;
D_FF dff0 (q,d,clk,reset);
not n1(d, q);

endmodule
```

17. Write Verilog Code for the Ripple Counter Structural description and its Test Bench for verification, perform simulation, observe the wave form and synthesize.

```
module ripple_counter_str (q,clk,reset);
```

```
output [3:0] q ;
```

```
input clk, reset;
```

```
T_FF tff0 (q[0],clk,reset);
```

```
T_FF tff1 (q[1],q[0],reset);
```

```
T_FF tff2 (q[2],q[1],reset);
```

```
T_FF tff3 (q[3],q[2],reset);
```

```
endmodule
```

TEST BENCH

```
module ripple_str_t ;
```

```
reg clk;
```

```
reg reset;
```

```
wire [3:0] q;
```

```
ripple_counter_str r2(q, clk, reset);
```

```
initial
```

```
    clk = 1'b0;
```

```
always
```

```
    #5 clk = ~clk;
```

```
initial
```

```
begin
```

```
    reset = 1'b1;
```

```
    #15 reset = 1'b0;
```

```
    #180 reset = 1'b1;
```

```
    #10 reset = 1'b0;
```

```
    #250 $finish;
```

```
end
```

```
endmodule
```

18. Write Verilog Code for the Ripple Counter and its Test Bench for verification, perform simulation, observe the wave form and synthesize.

```
module rip_ctr (clock, toggle, reset, count);
  input clock, toggle, reset;
  output [3:0] count;
  reg [3:0] count;
  wire c0, c1, c2;
  assign c0 = count[0], c1 = count[1], c2 = count[2];

  always @ (posedge reset or posedge clock)
    if (reset == 1'b1) count[0] <= 1'b0;
    else if (toggle == 1'b1) count[0] <= ~count[0];

  always @ (posedge reset or negedge c0)
    if (reset == 1'b1) count[1] <= 1'b0;
    else if (toggle == 1'b1) count[1] <= ~count[1];

  always @ (posedge reset or negedge c1)
    if (reset == 1'b1) count[2] <= 1'b0;
    else if (toggle == 1'b1) count[2] <= ~count[2];

  always @ (posedge reset or negedge c2)
    if (reset == 1'b1) count[3] <= 1'b0;
    else if (toggle == 1'b1) count[3] <= ~count[3];
endmodule
```

TEST BENCH

```
module rip_ctr_t ;
reg clock,toggle,reset;
wire [3:0] count ;

rip_ctr a1 (clock,toggle,reset,count);

initial
    clock = 1'b0;
always
    #5 clock = ~clock;

initial
begin
    reset = 1'b0;toggle = 1'b0;
    #10 reset = 1'b1; toggle = 1'b1;
    #10 reset = 1'b0;
    #190 reset = 1'b1;
    #20 reset = 1'b0;
    #100 reset = 1'b1;
    #40 reset = 1'b0;
    #450 $finish;
end
endmodule
```

19. Write Verilog Code for the Synchronous Counters and its Test Bench for verification, perform simulation, observe the waveform and synthesize.

```
module counter_behav ( count,reset,clk);
input wire reset, clk;
output reg [3:0] count;

always @(posedge clk)
if (reset)
count = 4'b0000;
else
count = count + 4'b0001;

endmodule
```

TEST BENCH

```
module mycounter_t ;
wire [3:0] count;
reg reset,clk;

initial
clk = 1'b0;

always
#5 clk = ~clk;

counter_behav m1 ( count,reset,clk);

initial
begin
reset = 1'b0 ;

#15 reset =1'b1;
#30 reset =1'b0;

#300 $finish;

end
endmodule
```


20. Write Verilog Code for the Successive Approximation Register [SAR] and its Test Bench for verification, perform simulation, observe the waveform and synthesize.

```

module shiftrne ( R,L,E,w,clock,q);
parameter n=8;
input [n-1:0] R;
input L,E,w,clock;
output [n-1:0] q;
reg [n-1:0] q;
integer k;

always @(posedge clock)
if (L)
    q <= R;
else if (E)
    begin
        for (k=n-1;k>0;k=k-1)
            q[k-1] <= q[k];
        q[n-1] <= w;
    end
endmodule

module sar_test;
reg [7:0] r;
reg l;
reg e;
reg w;
reg clk;
wire [7:0] q;

shiftrne sf(.R(r),.L(l),.E(e),.w(w),.clock(clk),.q(q));

initial
begin
    clk = 1'b0;
    l = 1'b1;
    w = 1'b0;
    e = 1'b0;
#5   r = 8'b1111_0000;
#10  l = 1'b0;
    e = 1'b1;
    w = 1'b0;
#10  w = 1'b0;
#10  w = 1'b0;
#10  w = 1'b0;
#10  w = 1'b1;
#10  w = 1'b1;
#10  w = 1'b1;
#10  w = 1'b1;
#10  $finish;
end
always #5 clk = ~clk;
endmodule

```

GATE LEVEL CODE:**INVERTER:**

```
module inverter ( out, in);  
output out;  
input in;  
assign out=~in;  
endmodule
```

NAND GATE:

```
module nandgate ( out, in1, in2);  
output out;  
input in1, in2;  
assign out=~(in1 & in2);  
endmodule
```

NOR GATE:

```
module norgate ( out, in1, in2);  
output out;  
input in1, in2;  
assign out=~(in1 | in2);  
endmodule
```

OR GATE:

```
module orgate ( out, in1, in2);  
output out;  
input in1, in2;  
assign out=(in1 | in2);  
endmodule
```

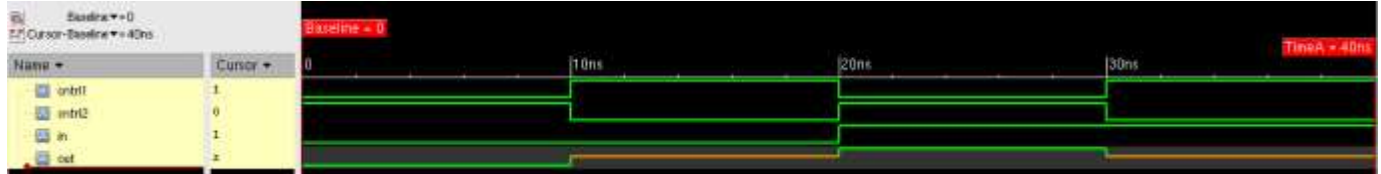
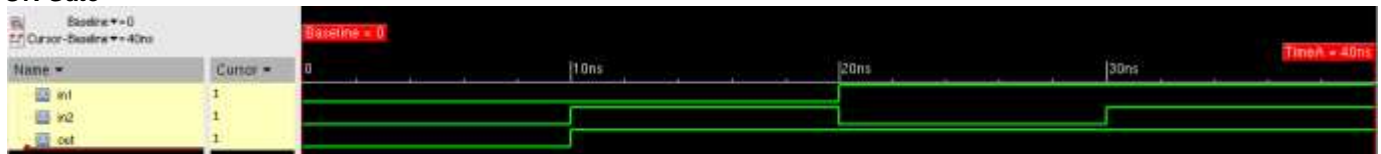
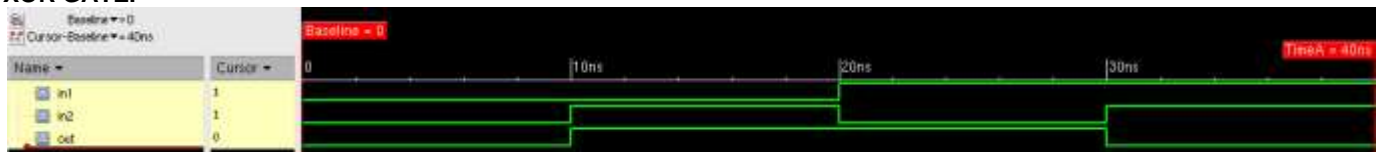
AND GATE:

```
module andgate ( out, in1, in2);  
output out;  
input in1, in2;  
assign out=(in1 & in2);  
endmodule
```

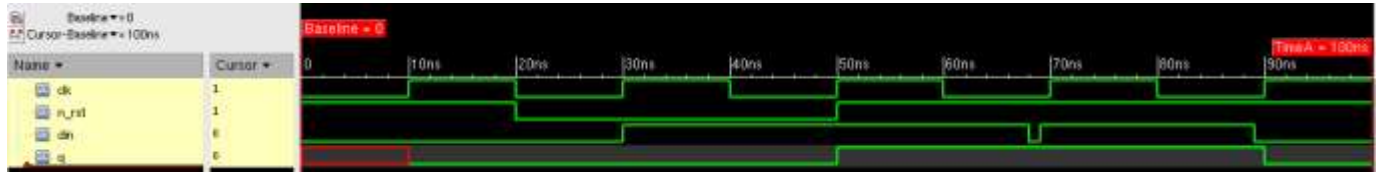
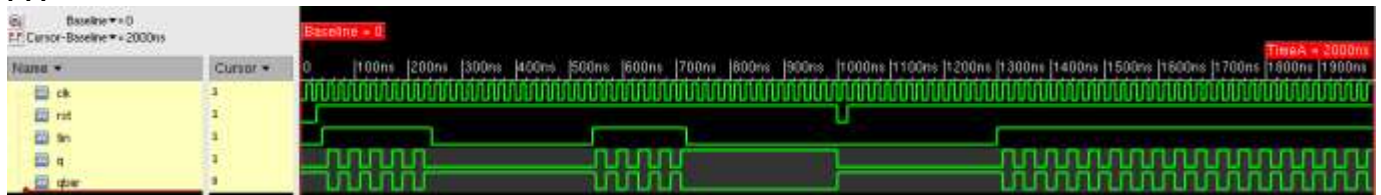
XOR GATE:

```
module xorgate ( out, in1, in2);  
output out;  
input in1, in2;  
assign out=(in1 ^ in2);  
endmodule
```

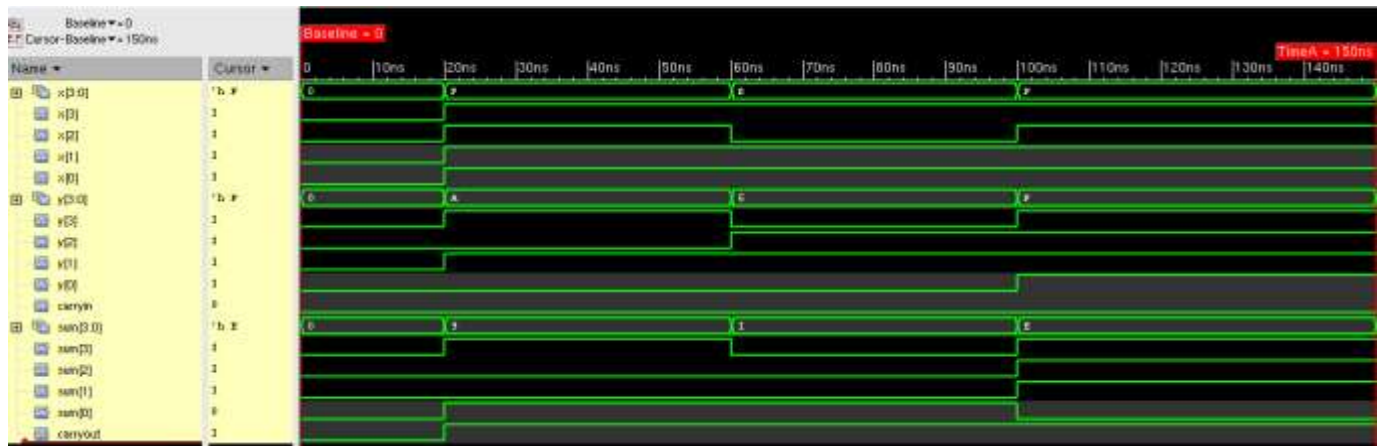
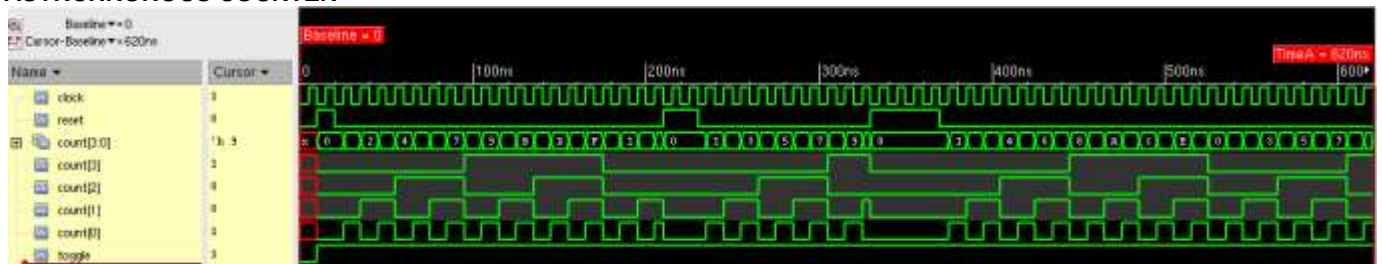
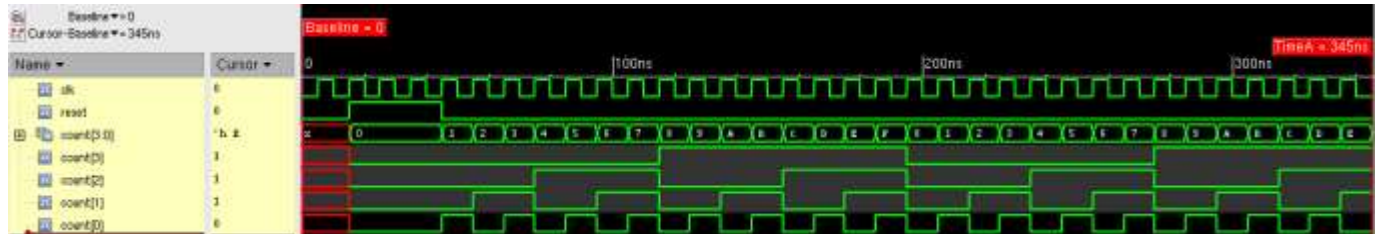
This page is intentionally left blank

Inverter**Buffer****Transmission Gate****NAND Gate****NOR GATE****OR Gate****AND Gate****XOR GATE:****XNOR GATE:**

This page is intentionally left blank

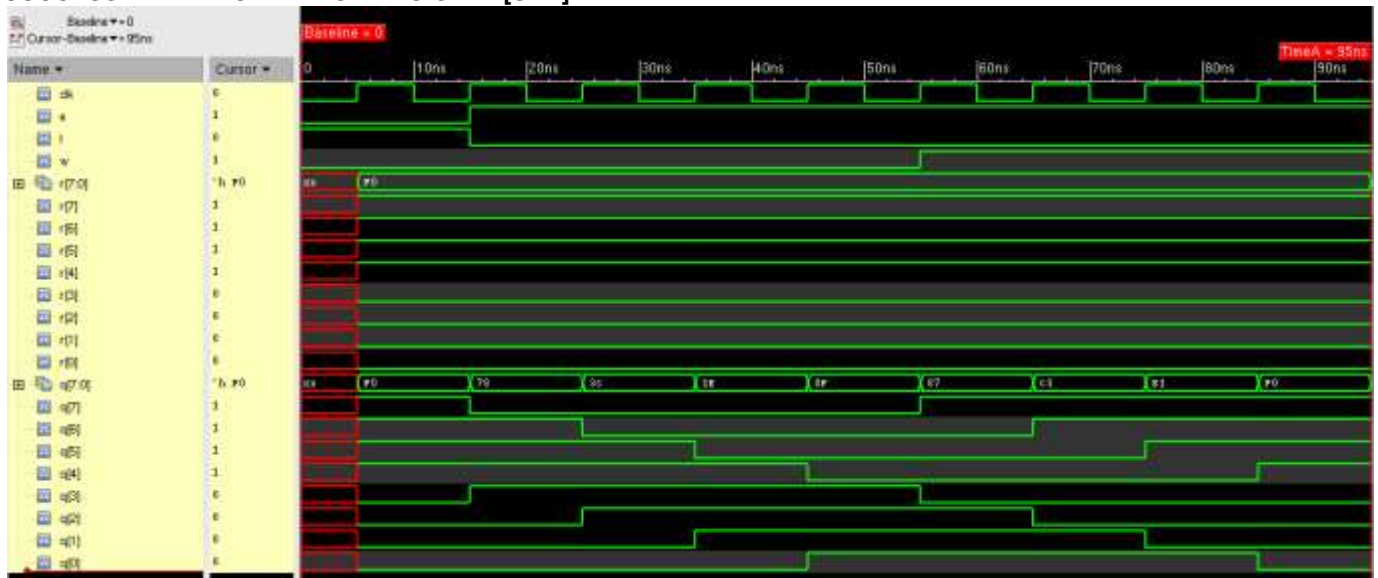
SR FF**D FF****T FF****JK FF****MS JK FF**

This page is intentionally left blank

PARALLEL ADDER**ASYNCHRONOUS COUNTER****SYNCHRONOUS COUNTER**

This page is intentionally left blank

SUCCESSIVE APPROXIMATION REGISTER [SAR]



This page is intentionally left blank

General Notes for analog virtuoso

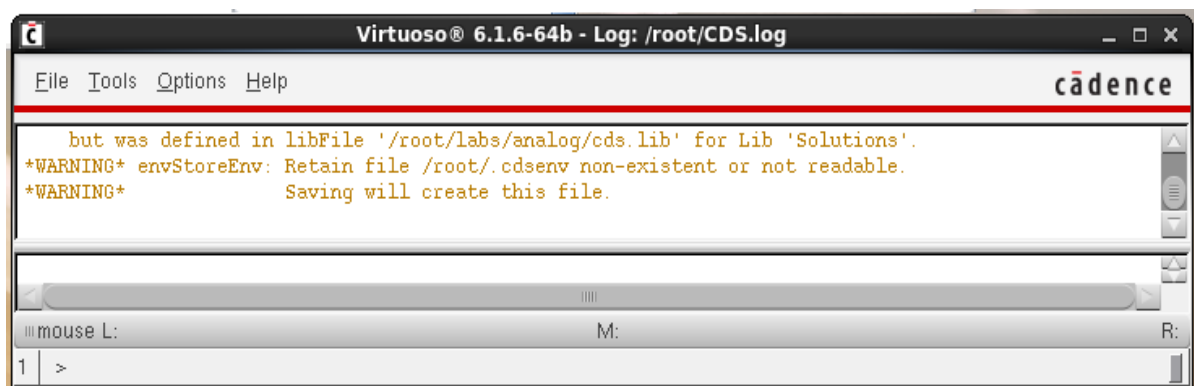
Lab Getting Started

1. Log in to your workstation using the username and password.
2. In a terminal window, type **cs**h at the command prompt to invoke the C shell.
 >csh
 >source cshrc
 >cd labs
 >cd analog

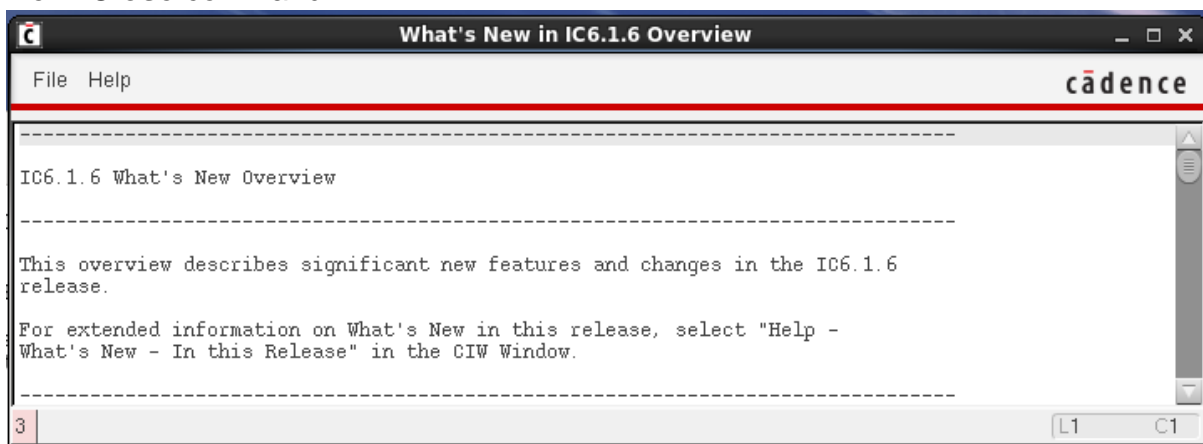
Starting the Cadence Software

1. In the terminal window, enter:
 >virtuoso &

The virtuoso or Command Interpreter Window (CIW) appears at the bottom of the screen.



2. If the “What’s New ...” if window appears, close it with the **File— Close** command.

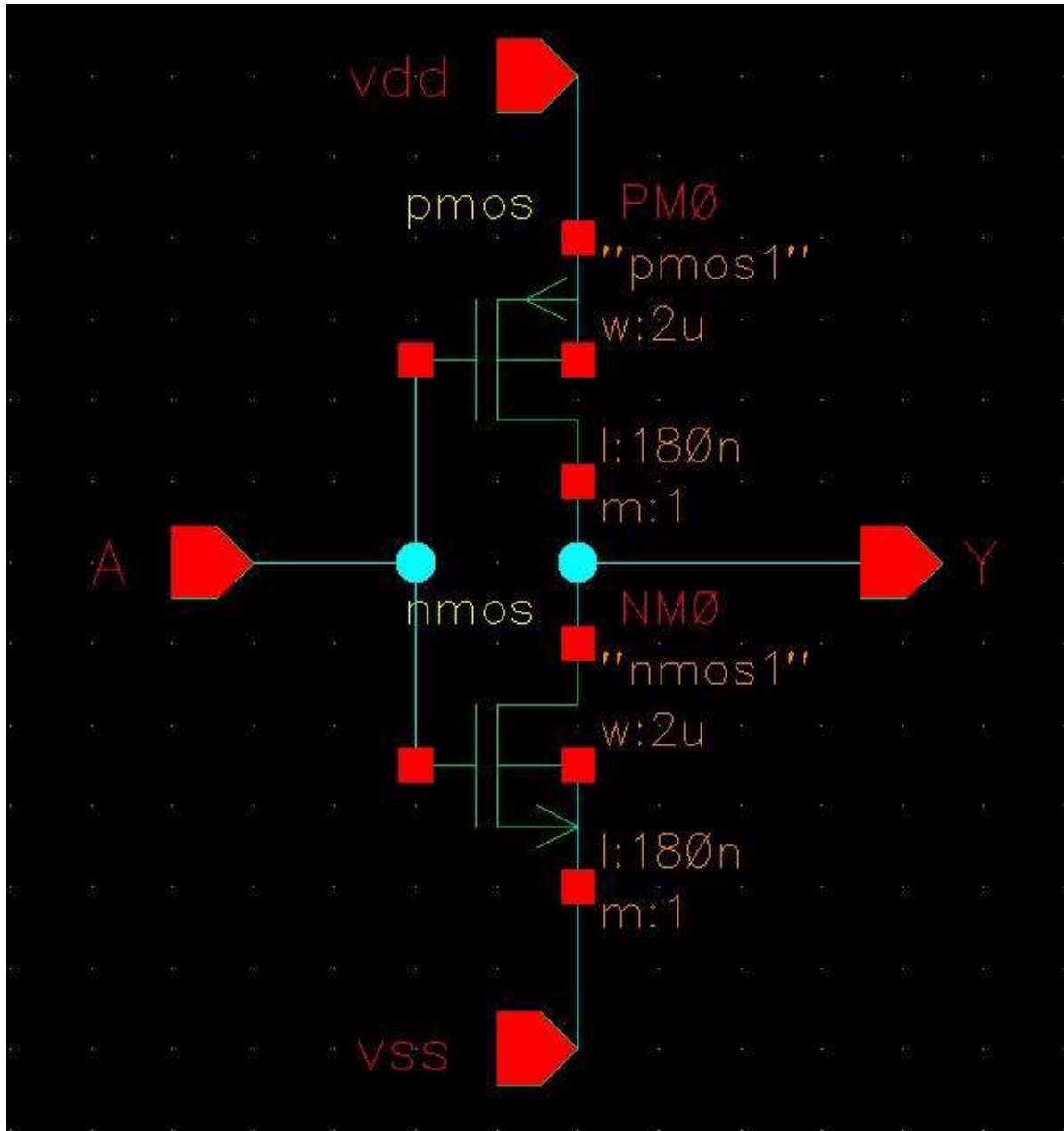


3. Keep opened CIW window for the labs.

End of General Notes

1. Design a CMOS Inverter circuit with the given specifications.
Draw the schematic and verify DC Analysis, Transient Analysis.
Draw the Layout and verify the DRC, LVS and Extract RC.

INVERTER Schematic



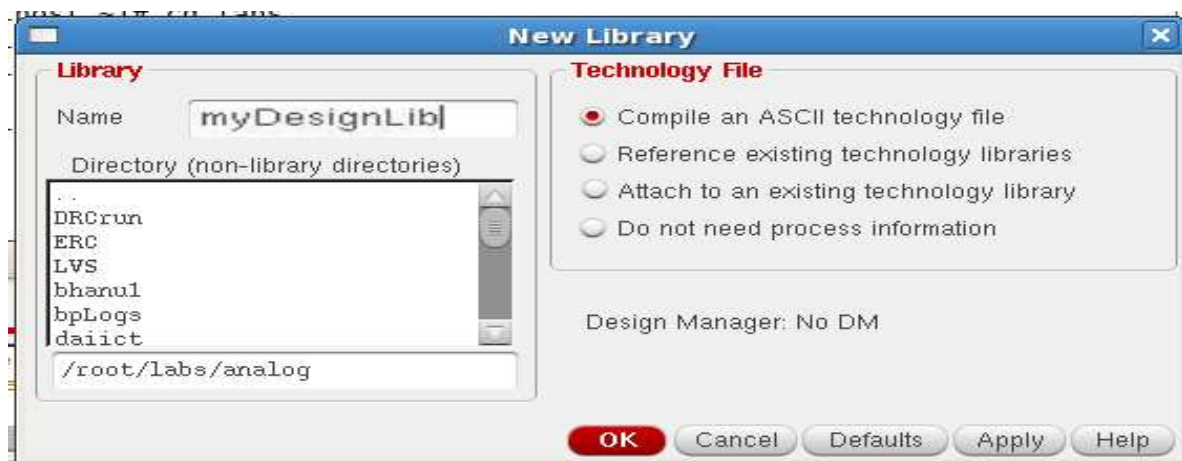
Schematic Entry

Objective: To create a library and build a schematic of an Inverter

Below steps explain the creation of new library “**myDesignLib**” and we will use the same throughout this course for building various cells that we going to create in the next labs. Execute **Tools – Library Manager** in the CIW or Virtuoso window to open Library Manager.

Creating a New library

1. In the Library Manager, execute **File - New – Library**. The new library form appears.
2. In the “New Library” form, type “**myDesignLib**” in the Name section.



3. In the field of Directory section, verify that the path to the library is set to **~/labs/analog** and click **OK**.

Note: A technology file is not required if you are not interested to do the layouts for the design

5. In the next “**Technology File for New library**” form, select option **Attach to an existing techfile** and click **OK**.



6. In the “**Attach Design Library to Technology File**” form, select **gpd180** from the cyclic field and click **OK**.



7. After creating a new library you can verify it from the library manager.

8. If you right click on the “**myDesignLib**” and select properties, you will find that **gpd180** library is attached as techlib to “**myDesignLib**”.



Creating a Schematic Cellview

In this section we will learn how to open new schematic window in the new “**myDesignLib**” library and build the inverter schematic as shown in the figure at the start of this lab.

1. In the CIW or Library manager, execute **File – New – Cellview**.
2. Set up the New file form as follows:



Do not edit the **Library path file** and the one above might be different from the path shown in your form.

3. Click **OK** when done the above settings. A blank schematic window for the **Inverter** design appears.

Adding Components to schematic

1. In the Inverter schematic window, click the **Instance** fixed menu icon to display the Add Instance form.

Tip: You can also execute **Create — Instance** or press **i**.

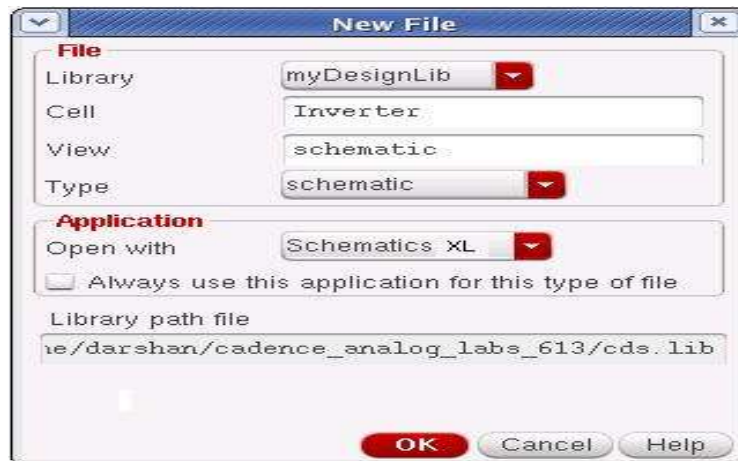
2. Click on the **Browse** button. This opens up a Library browser from which you can select components and the **symbol** view .

You will update the Library Name, Cell Name, and the property values given in the table on the next page as you place each component.

Creating a Schematic Cellview

In this section we will learn how to open new schematic window in the new “**myDesignLib**” library and build the inverter schematic as shown in the figure at the start of this lab.

1. In the CIW or Library manager, execute **File – New – Cellview**.
2. Set up the New file form as follows:



Do not edit the **Library path file** and the one above might be different from the path shown in your form.

3. Click **OK** when done the above settings. A blank schematic window for the **Inverter** design appears.

Adding Components to schematic



1. In the Inverter schematic window, click the **Instance** fixed menu icon to display the Add Instance form.



Tip: You can also execute **Create — Instance** or press **i**.

2. Click on the **Browse** button. This opens up a Library browser from which you can select components and the **symbol** view.

You will update the Library Name, Cell Name, and the property values given in the table on the next page as you place each component.

3. After you complete the Add Instance form, move your cursor to the schematic window and click **left** to place a component.

This is a table of components for building the Inverter schematic.

Library name	Cell Name	Properties/Comments
gpd180	pmos	Model name = pmos1, W= 2u, L=180n
gpd181	nmos	Model name = nmos1, W= 2u, L=180n

If you place a component with the wrong parameter values, use the **Edit— Properties— Objects** command to change the parameters. Use the **Edit— Move** command if you place components in the wrong location.



You can rotate components at the time you place them, or use the **Edit— Rotate** command after they are placed.

4. After entering components, click **Cancel** in the Add Instance form or press **Esc** with your cursor in the schematic window.

Adding pins to Schematic

1. Click the **Pin** fixed menu icon in the schematic window.


You can also execute **Create — Pin** or press **p**. 

The Add pin form appears.

2. Type the following in the Add pin form in the exact order leaving space between the pin names.

Pin Names	Direction
vin	Input
vout	Output

Make sure that the direction field is set to **input/output/inputOutput** when placing the **input/output/inout** pins respectively and the Usage field is set to **schematic**.

3. Select **Cancel** from the Add – pin form after placing the pins. 

In the schematic window, execute **Window— Fit** or press the **f** bindkey.



Adding Wires to a Schematic

Add wires to connect components and pins in the design.

1. Click the **Wire (narrow)** icon in the schematic window.

You can also press the **w** key, or execute **Create — Wire (narrow)**.



2. In the schematic window, click on a pin of one of your components as the first point for your wiring. A diamond shape appears over the starting point of this wire.

3. Follow the prompts at the bottom of the design window and click **left** on the destination point for your wire. A wire is routed between the source and destination points.

4. Complete the wiring as shown in figure and when done wiring press **ESC** key in the schematic window to cancel wiring.

Saving the Design

1. Click the **Check and Save** icon in the schematic editor window.



2. Observe the CIW output area for any errors.

Symbol Creation

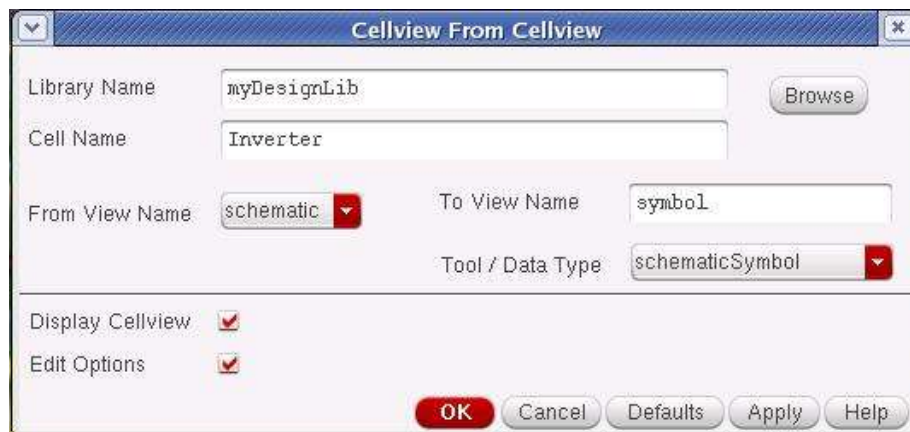
Objective: To create a symbol for the Inverter

In this section, you will create a symbol for your inverter design so you can place it in a test circuit for simulation. A symbol view is extremely important step in the design process. The symbol view must exist for the schematic to be used in a hierarchy. In addition, the symbol has attached properties (cdsParam) that facilitate the simulation and the design of the circuit.

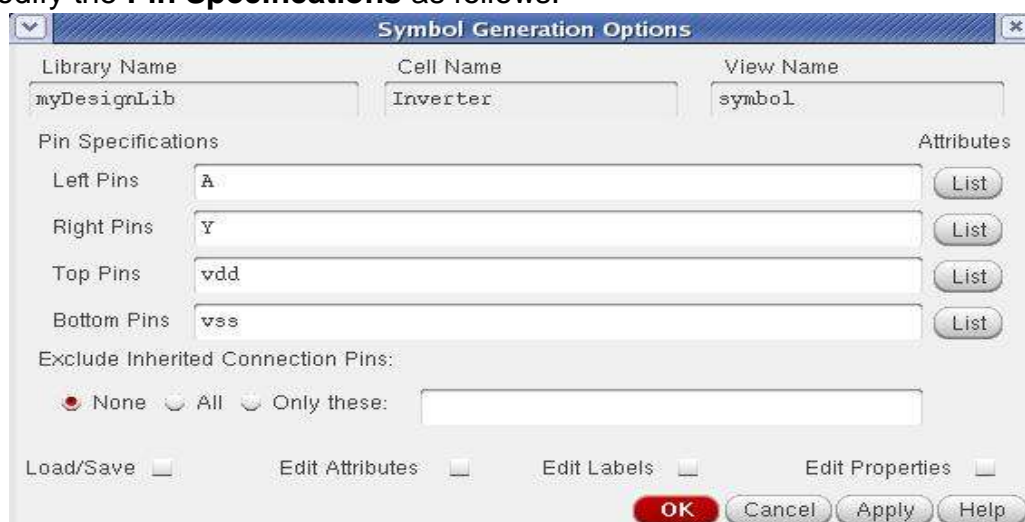
1. In the Inverter schematic window, execute **Create — Cellview— From Cellview**.

The **Cellview From Cellview** form appears. With the Edit Options function active, you can control the appearance of the symbol to generate.

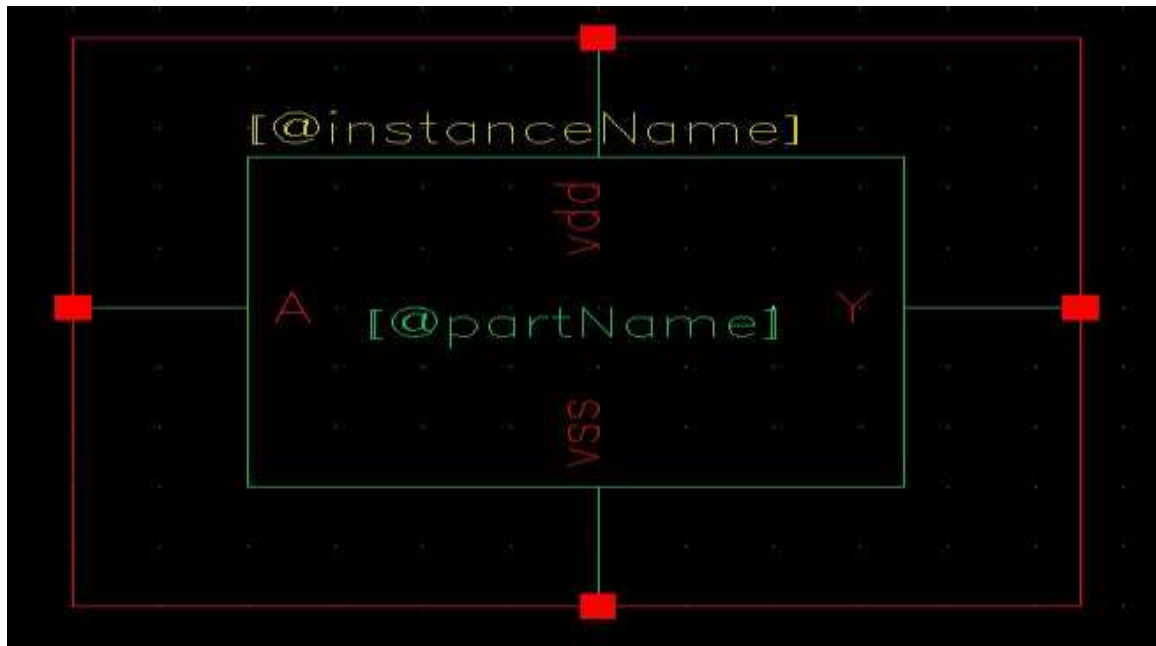
2. Verify that the **From View Name** field is set to **schematic**, and the **To View Name** field is set to **symbol**, with the **Tool/Data Type** set as **SchematicSymbol**.



3. Click **OK** in the **Cellview From Cellview** form.
The Symbol Generation Form appears
4. Modify the **Pin Specifications** as follows:



5. Click **OK** in the Symbol Generation Options form.
6. A new window displays an automatically created Inverter symbol as shown here.

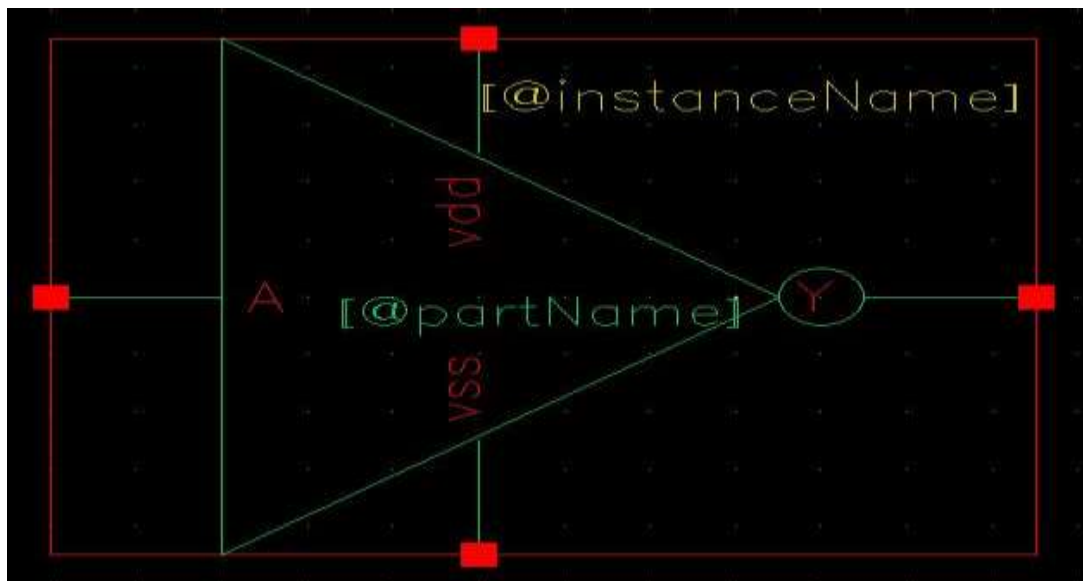


Editing a Symbol

In this section we will modify the inverter symbol to look like a Inverter gate symbol.



1. Move the cursor over the automatically generated symbol, until the green rectangle is highlighted, click **left** to select it.
2. Click **Delete** icon in the symbol window, similarly select the red rectangle and delete that.
3. Execute **Create – Shape – polygon**, and draw a shape similar to triangle.
4. After creating the triangle press **ESC** key.
5. Execute **Create – Shape – Circle** to make a circle at the end of triangle.
6. You can move the pin names according to the location.
7. Execute **Create – Selection Box**. In the Add Selection Box form, click **Automatic**. A new red selection box is automatically added.
8. After creating symbol, click on the **save** icon in the symbol editor window to save the symbol. In the symbol editor, execute **File – Close** to close the symbol view window.



Building the Inverter_Test Design

Objective: To build an Inverter Test circuit using your Inverter

Creating the Inverter_Test Cellview

You will create the Inverter_Test cellview that will contain an instance of the Inverter cellview. In the next section, you will run simulation on this design

1. In the CIW or Library Manager, execute **File— New— Cellview**.
2. Set up the New File form as follows:



3. Click **OK** when done. A blank schematic window for the **Inverter_Test** design appears.

Building the Inverter_Test Circuit

1. Using the component list and Properties/Comments in this table, build the **Inverter_Test** schematic.

Library name	Cellview name	Properties/Comments
myDesignLib	Inverter	Symbol
analogLib	vpulse	v1=0, v2=1.8, Period=20n, PW=10n
analogLib	vdc, gnd	vdc=1.8

Note: Remember to set the values for **VDD** and **VSS**. Otherwise, your circuit will have no power.

2. Add the above components using **Create — Instance** or by pressing **I**.



3. Click the **Wire (narrow)** icon and wire your schematic.

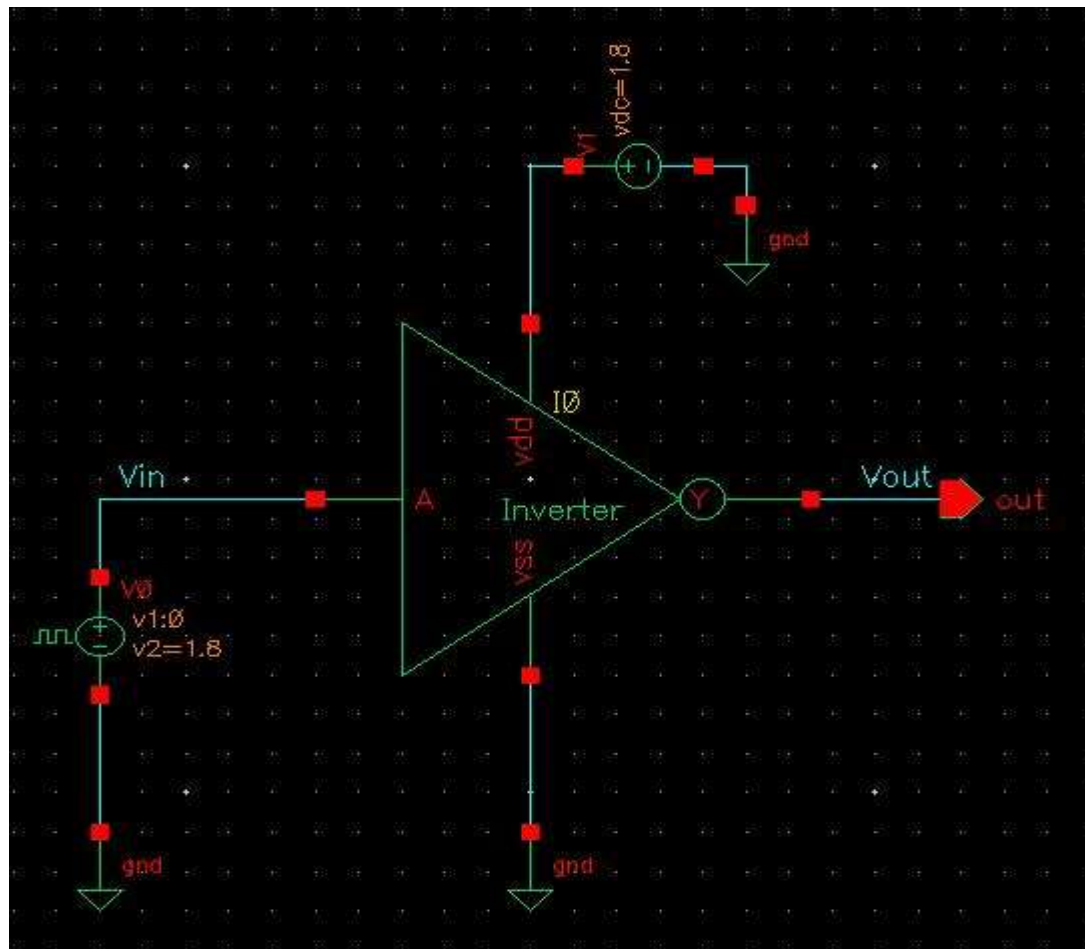
Tip: You can also press the **w** key, or execute **Create— Wire (narrow)**.



4. Click **Create — Wire Name** or press **L** to name the input (**Vin**) and output (**Vout**) wires as in the below schematic.



5. Click on the **Check and Save** icon to save the design.



6. Leave your **Inverter_Test** schematic window open for the next section.

Analog Simulation with Spectre

Objective: To set up and run simulations on the Inverter_Test design

In this section, we will run the simulation for Inverter and plot the transient, DC characteristics and we will do Parametric Analysis after the initial simulation.

Starting the Simulation Environment

Start the Simulation Environment to run a simulation.

1. In the **Inverter_Test** schematic window, execute **Launch – ADE L**

The **Virtuoso Analog Design Environment (ADE)** simulation window appears.

Choosing Analyses

This section demonstrates how to view and select the different types of analyses to complete the circuit when running the simulation.

1. In the Simulation window (ADE), click the **Choose - Analyses** icon. You can also execute **Analyses - Choose**.



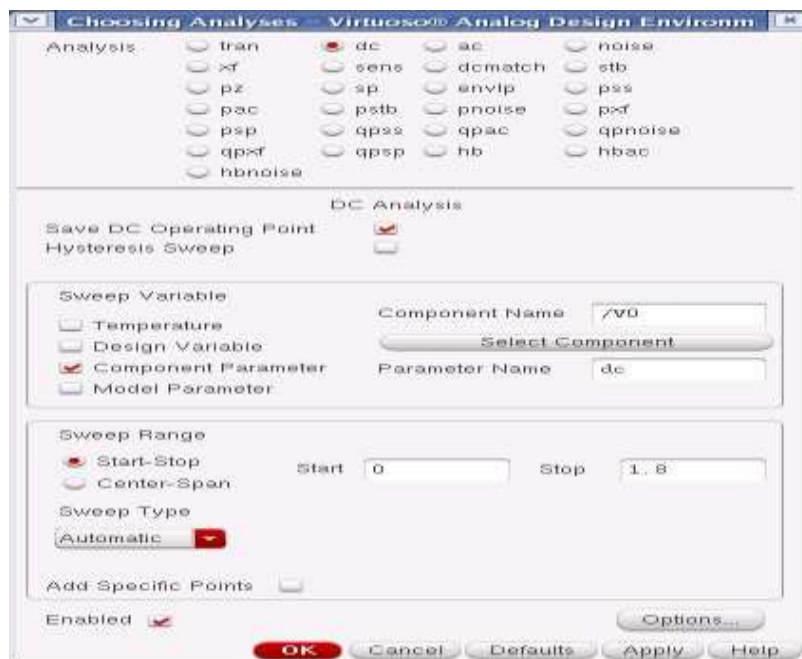
The Choosing Analysis form appears. This is a dynamic form, the bottom of the form changes based on the selection above.

2. To setup for transient analysis
 - a. In the Analysis section select **tran**
 - b. Set the stop time as **200n**
 - c. Click at the **moderate** or **Enabled** button at the bottom, and then click **Apply**.



3. To set up for DC Analyses:


- In the Analyses section, select **dc**.
- In the DC Analyses section, turn on **Save DC Operating Point**.
- Turn on the **Component Parameter**.
- Double click the **Select Component**, Which takes you to the schematic window.
- Select input signal **vpulse source** in the test schematic window.
- Select **—DC Voltage||** in the **Select Component Parameter** form and click OK.
- In the analysis form type **start** and **stop** voltages as **0** to **1.8** respectively.
- Check the enable button and then click **Apply**.



4. Click **OK** in the Choosing Analyses Form.

Setting Design Variables

Set the values of any design variables in the circuit before simulating. Otherwise, the simulation will not run.

- In the Simulation window, click the **Edit Variables** icon.  The Editing Design Variables form appears.

- Click **Copy From** at the bottom of the form. The design is scanned and all variables found in the design are listed. In a few moments, the **wp** variable appears in the Table of Design variables section.

- Set the value of the **wp** variable:

With the **wp** variable highlighted in the Table of Design Variables, click on the variable name **wp** and enter the following:

Value(Expr)	2u
-------------	----

Click **Change** and notice the update in the Table of Design Variables.

3. Click **OK** or **Cancel** in the Editing Design Variables window.

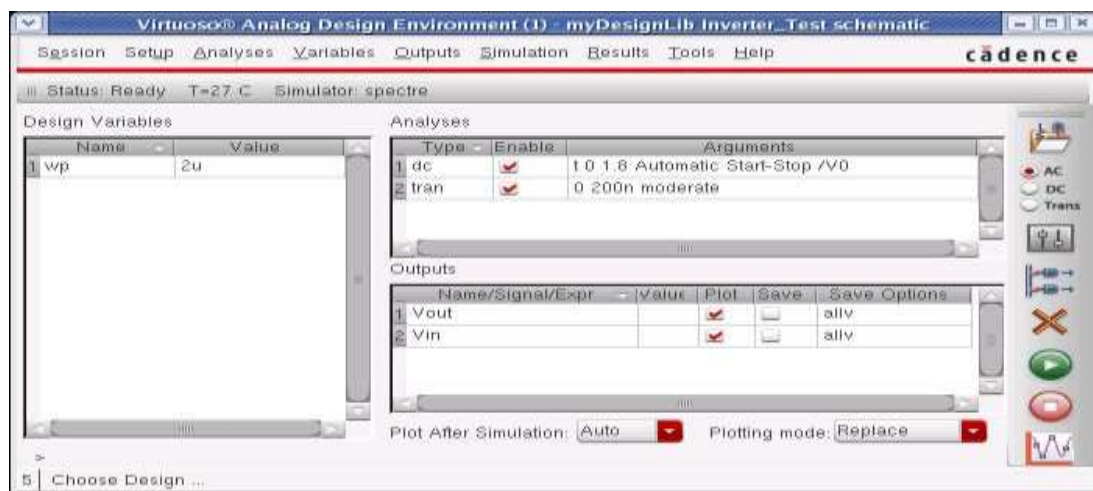
Selecting Outputs for Plotting

1. Execute **Outputs – To be plotted – Select on Schematic** in the simulation window.

2. Follow the prompt at the bottom of the schematic window, Click on output net **Vout**,

input net **Vin** of the Inverter. Press **ESC** with the cursor in the schematic after selecting it.

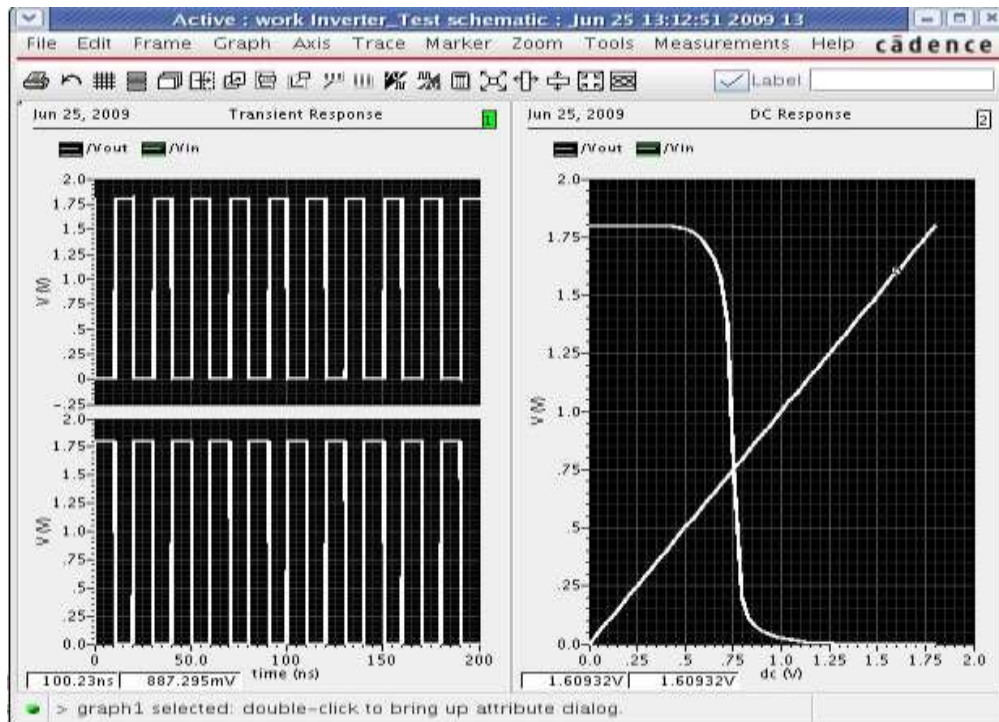
Does the simulation window look like this?



Running the Simulation



1. Execute **Simulation – Netlist and Run** in the simulation window to start the Simulation or the icon, this will create the netlist as well as run the simulation.
2. When simulation finishes, the Transient, DC plots automatically will be popped up along with log file.



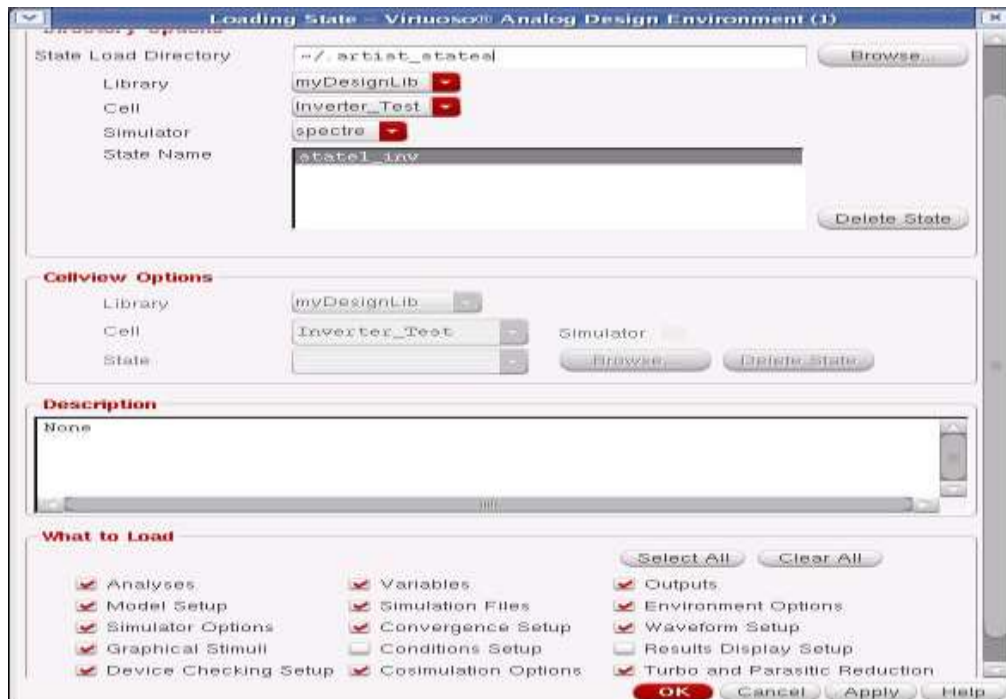
Saving the Simulator State

We can save the simulator state, which stores information such as model library file, outputs, analysis, variable etc. This information restores the simulation environment without having to type in all of setting again.

1. In the Simulation window, execute **Session – Save State**.
The Saving State form appears.
2. Set the **Save as** field to **state1_inv** and make sure all options are selected under what to save field.
3. Click **OK** in the saving state form. The Simulator state is saved.

Loading the Simulator State

1. From the ADE window execute **Session – Load State**.
2. In the Loading State window, set the State name to **state1_inv** as shown



3. Click **OK** in the Loading State window

Parametric Analysis

Parametric Analysis yields information similar to that provided by the Spectre® sweep feature, except the data is for a full range of sweeps for each parametric step. The Spectre sweep feature provides sweep data at only one specified condition.

You will run a parametric DC analysis on the **wp** variable, of the PMOS device of the Inverter design by sweeping the value of **wp**.

Run a simulation before starting the parametric tool. You will start by loading the state from the previous simulation run.

Run the simulation and check for errors. When the simulation ends, a single waveform in the waveform window displays the DC Response at the **Vout** node.

Starting the Parametric Analysis Tool

1. In the Simulation window, execute **Tools—Parametric Analysis**.
The Parametric Analysis form appears.

2. In the Parametric Analysis form, execute
Setup—Pick Name For Variable—Sweep 1.

A selection window appears with a list of all variables in the design that you can sweep. This list includes the variables that appear in the Design Variables section of the Simulation window.

3. In the selection window, double click left on **wp**.
The Variable Name field for Sweep 1 in the Parametric Analysis form is set to **wp**.

4. Change the Range Type and Step Control fields in the Parametric Analysis form as shown below:

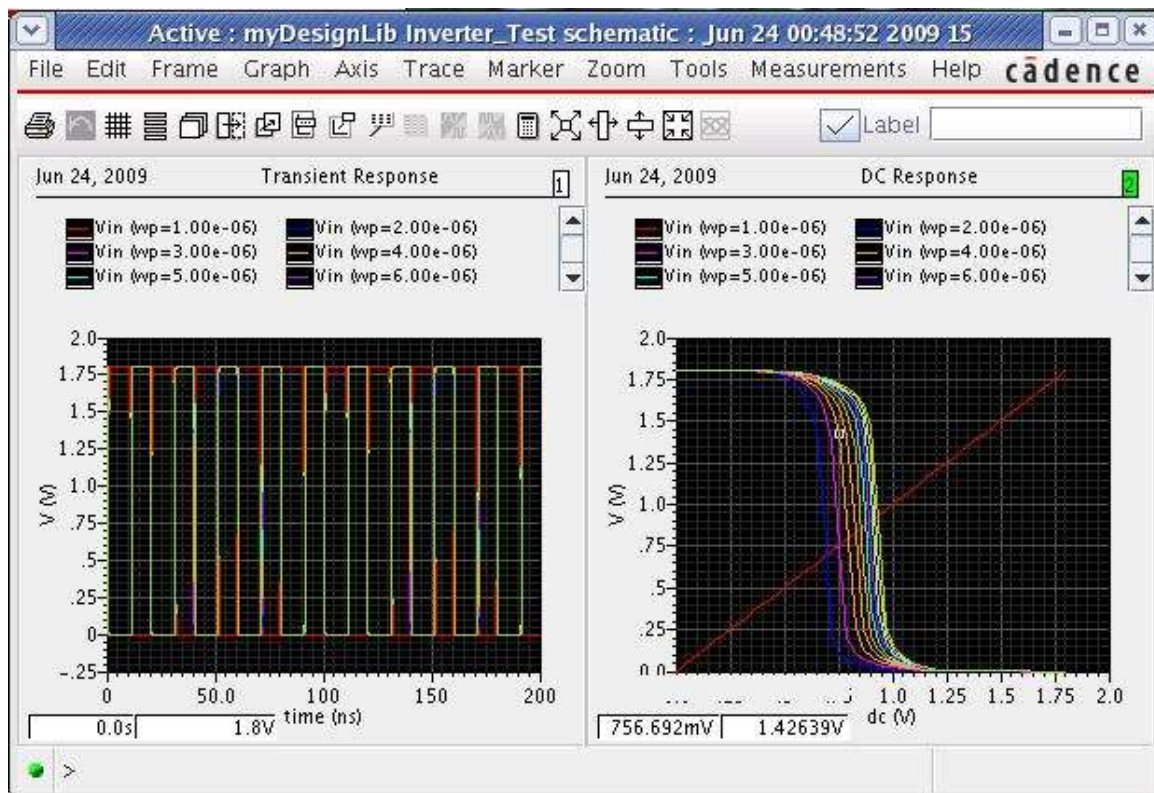
Range Type	From/To	From	1u	To	10u
Step Control	Auto	Total Steps	10		

These numbers vary the value of the **wp** of the pmos between 1um and 10um at ten evenly spaced intervals.



5. Execute **Analysis—Start**.

The Parametric Analysis window displays the number of runs remaining in the analysis and the current value of the swept variable(s). Look in the upper right corner of the window. Once the runs are completed the wavescan window comes up with the plots for different runs.



Note: Change the wp value of pmos device back to 2u and save the schematic before proceeding to the next section of the lab. To do this use edit property option.

Creating Layout View of Inverter

1. From the **Inverter** schematic window menu execute **Launch – Layout XL**. A **Startup Option** form appears.
2. Select **Create New** option. This gives a New Cell View Form
3. Check the Cellname (**Inverter**), Viewname (**layout**).
4. Click **OK** from the New Cellview form.

LSW and a blank layout window appear along with schematic window.

Adding Components to Layout.



1. Execute **Connectivity – Generate – All from Source** or click the icon in the layout editor window, **Generate Layout** form appears. Click **OK** which imports the schematic components in to the Layout window automatically.
2. Re arrange the components with in PR-Boundary as shown in the next page.
Press shift+F to change the view,
Place –pin placement, click on place as in schematic,
Create H-rail-select vdd, vss click on H-rail.
Create wire **
3. To rotate a component, Select the component and execute **Edit –Properties**.
now select the degree of rotation from the property edit form.



4. To Move a component, Select the component and execute **Edit -Move** command.
5. To Extend the PR boundary right click stretch & click on the line which is to be extend.

Making interconnection



1. Execute **Connectivity –Nets – Show/Hide selected Incomplete Nets** or click the icon in the Layout Menu.
2. Move the mouse pointer over the device and click **LMB** to get the connectivity information, which shows the guide lines (or flight lines) for the inter connections of the components.
3. From the layout window execute **Create – Shape – Path/ Create wire** or **Create – Shape – Rectangle** (for vdd and gnd bar) and select the appropriate Layers from the **LSW** window and Vias for making the inter connections
Shift-z, Ctrl-z –zoom.

Creating Contacts/Vias

You will use the contacts or vias to make connections between two different layers.

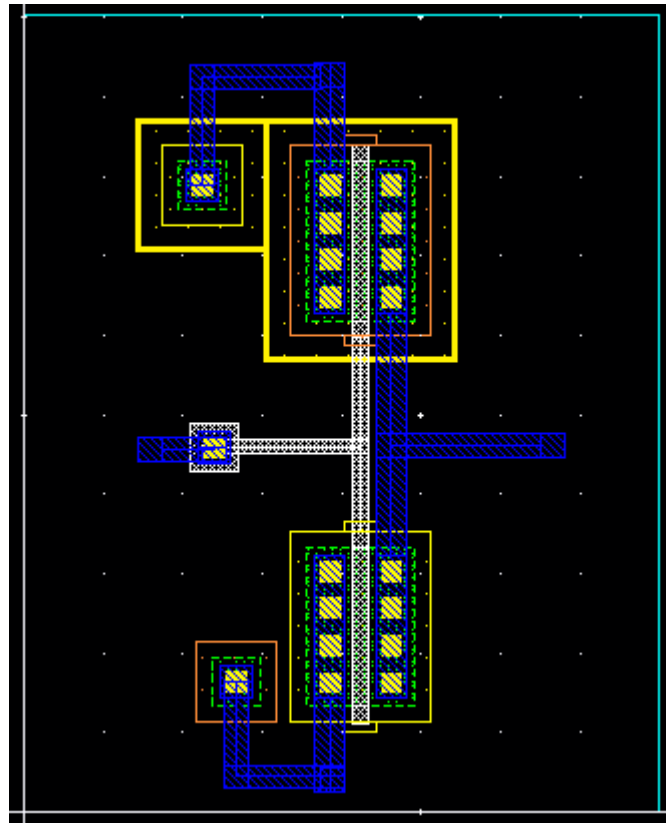
1. Execute **Create — Via** or select command to place different Contacts, as given in below table **

Connection	Contact Type
For Metal1- Poly Connection	Metal1-Poly
For Metal1- Psubstrate Connection	Metal1-Sub
For Metal1- Nwell Connection	Metal1-Nwell

(to extend the PR boundary right click stretch and click on the line which is to be Extended)

Saving the design

1. Save your design by selecting **File — Save** or click  to save the layout, and layout should appear as below



Physical Verification

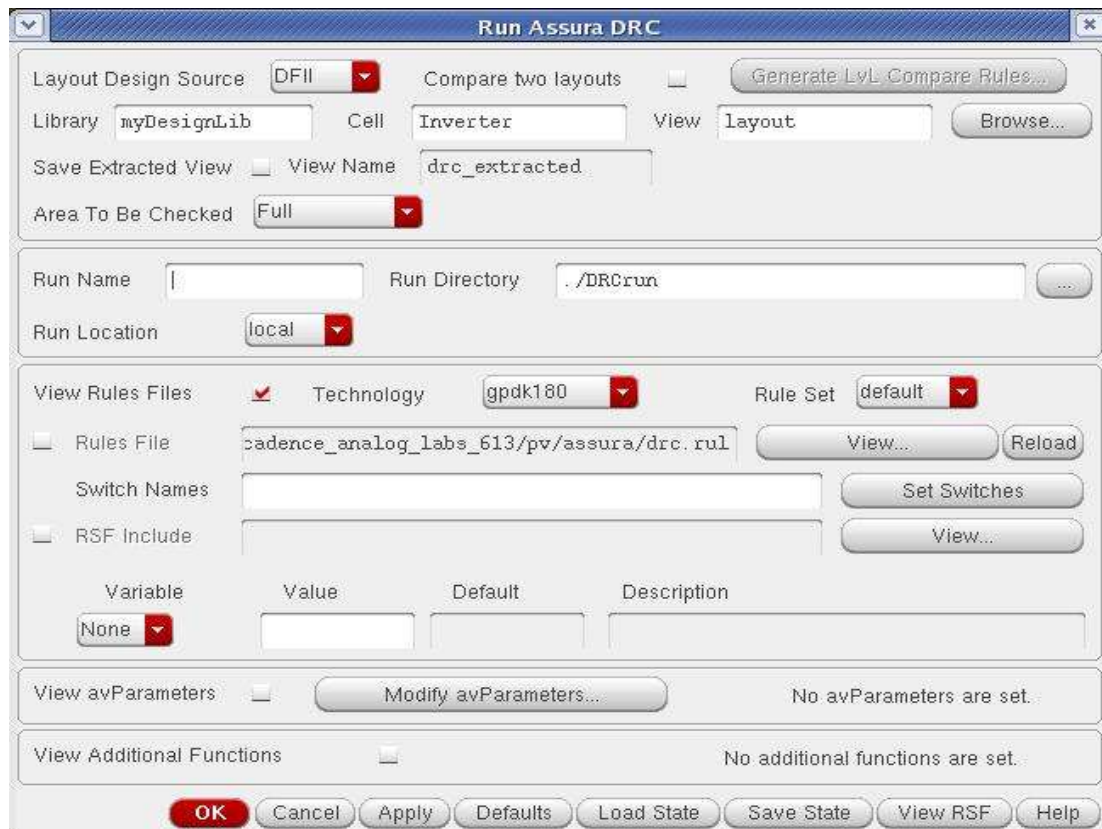
Assura DRC

Running a DRC

1. Open the Inverter layout from the CIW or library manager if you have closed that. Press **shift – f** in the layout window to display all the levels.
2. Select **Assura - Run DRC** from layout window.

The DRC form appears. The Library and Cellname are taken from the current design window, but rule file may be missing. Select the Technology as **gpd180**. This automatically loads the rule file.

Your DRC form should appear like this



3. Click **OK** to start DRC.
4. A Progress form will appear. You can click on the watch log file to see the log file.
5. When DRC finishes, a dialog box appears asking you if you want to view your DRC results, and then click **Yes** to view the results of this run.
6. If there are any DRC errors in the design **View Layer Window (VLW)** and **Error Layer Window (ELW)** appears. Also the errors highlight in the design itself.
7. Click **View – Summary** in the ELW to find the details of errors.
8. You can refer to rule file also for more information, correct all the DRC errors and **Re – run** the DRC.
9. If there are no errors in the layout then a dialog box appears with **No DRC errors found** written in it, click on **close** to terminate the DRC run.

ASSURA LVS

In this section we will perform the LVS check that will compare the schematic netlist and the layout netlist.

Running LVS

1. Select **Assura – Run LVS** from the layout window.
The Assura Run LVS form appears. It will automatically load both the schematic and layout view of the cell.
2. Change the following in the form and click **OK**.

3. The LVS begins and a Progress form appears.
4. If the schematic and layout matches completely, you will get the form displaying **Schematic and Layout Match**.
5. If the schematic and layout do not matches, a form informs that the LVS completed successfully and asks if you want to see the results of this run.
6. Click **Yes** in the form. LVS debug form appears, and you are directed into LVS debug environment.
7. In the **LVS debug form** you can find the details of mismatches and you need to correct all those mismatches and **Re – run** the LVS till you will be able to match the schematic with layout.

Assura RCX

In this section we will extract the RC values from the layout and perform analog circuit simulation on the designs extracted with RCX.

Before using RCX to extract parasitic devices for simulation, the layout should match with schematic completely to ensure that all parasites will be back annotated to the correct schematic nets.

Running RCX

1. From the layout window execute **Assura – Run RCX**. or **QRC**
2. Change the following in the Assura parasitic extraction form. Select **output** type under **Setup** tab of the form.

The screenshot shows the 'QRC (Assura) Parasitic Extraction Run Form' dialog box with the 'Setup' tab selected. The 'Technology' is set to 'gpdkt180' and 'RuleSet' is 'default'. The 'Setup Dir' is '/home/darshan/cadence_analog_labs_613/pv/assura/rcx'. The 'Output' is set to 'Extracted View'. The 'Lib' is 'DesignLib', 'Cel' is 'Inverter', and 'View' is 'av_extracted'. The 'Parasitic Res Component' is 'presistor', 'Parasitic Cap Component' is 'pcapacitor', 'Parasitic Ind Component' is 'pinductor', and 'Parasitic M Component' is 'pmind'. The 'Inductance L1 Prop Id' is 'ind1' and 'Inductance L2 Prop Id' is 'ind2'. The 'Substrate Extract' checkbox is checked. The 'Extract MOS Diffusion AP' checkbox is checked. The 'Substrate Profile' is 'NONE'. The 'Library Prefix' and 'Library Directory' are empty. The 'OK' button is highlighted in red.

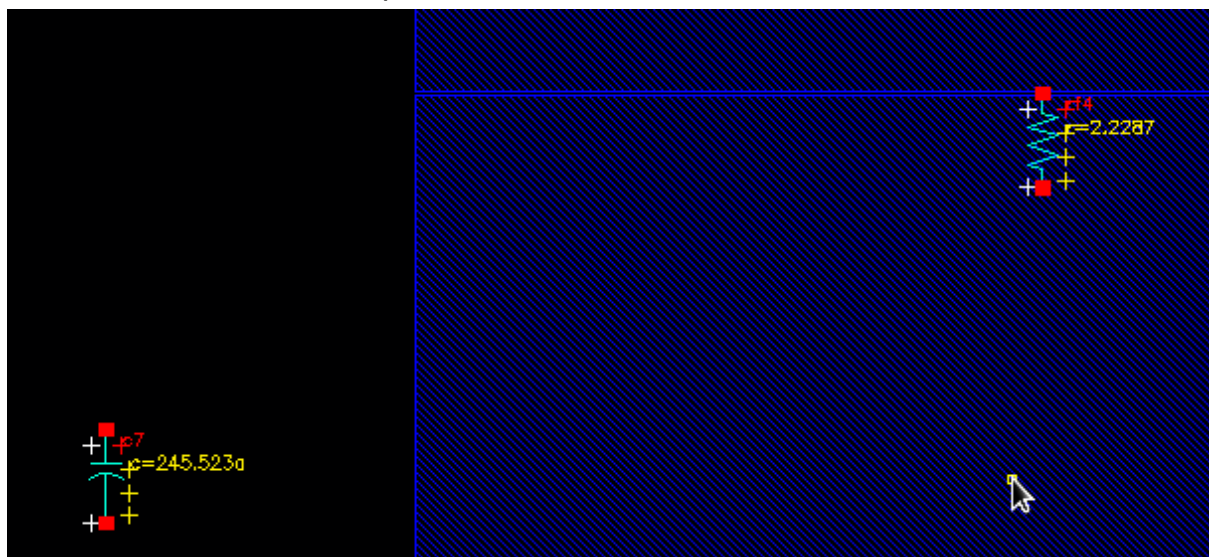
3. In the **Extraction** tab of the form, choose Extraction type, Cap Coupling Mode and specify the Reference node for extraction.



4. In the **Filtering** tab of the form, **Enter Power Nets** as vdd!, vss! and **Enter Ground Nets** as gnd!



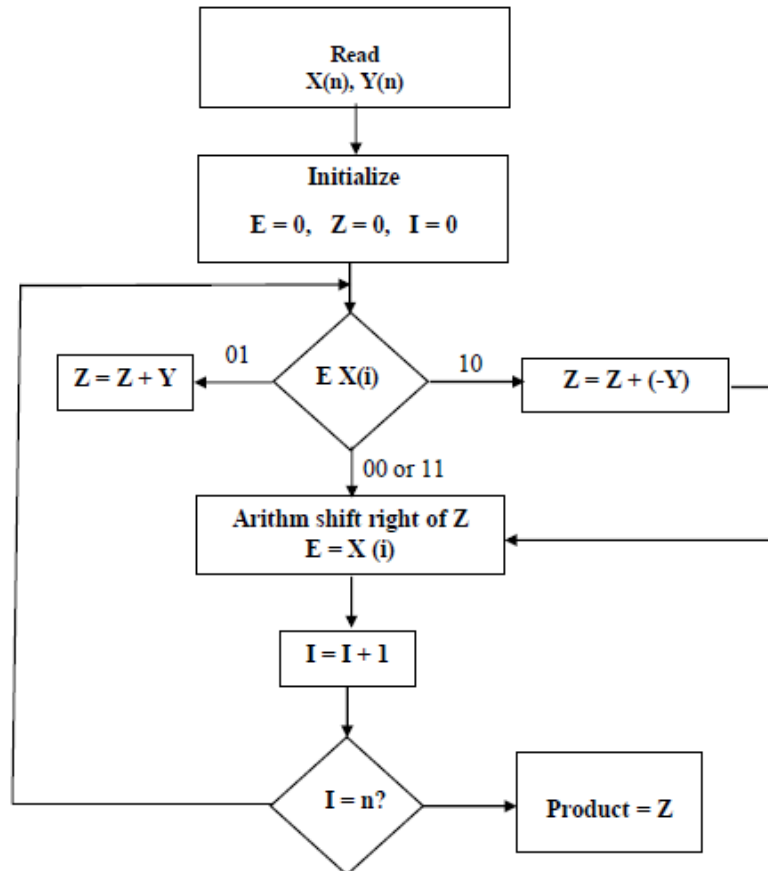
5. Click **OK** in the Assura parasitic extraction form when done.



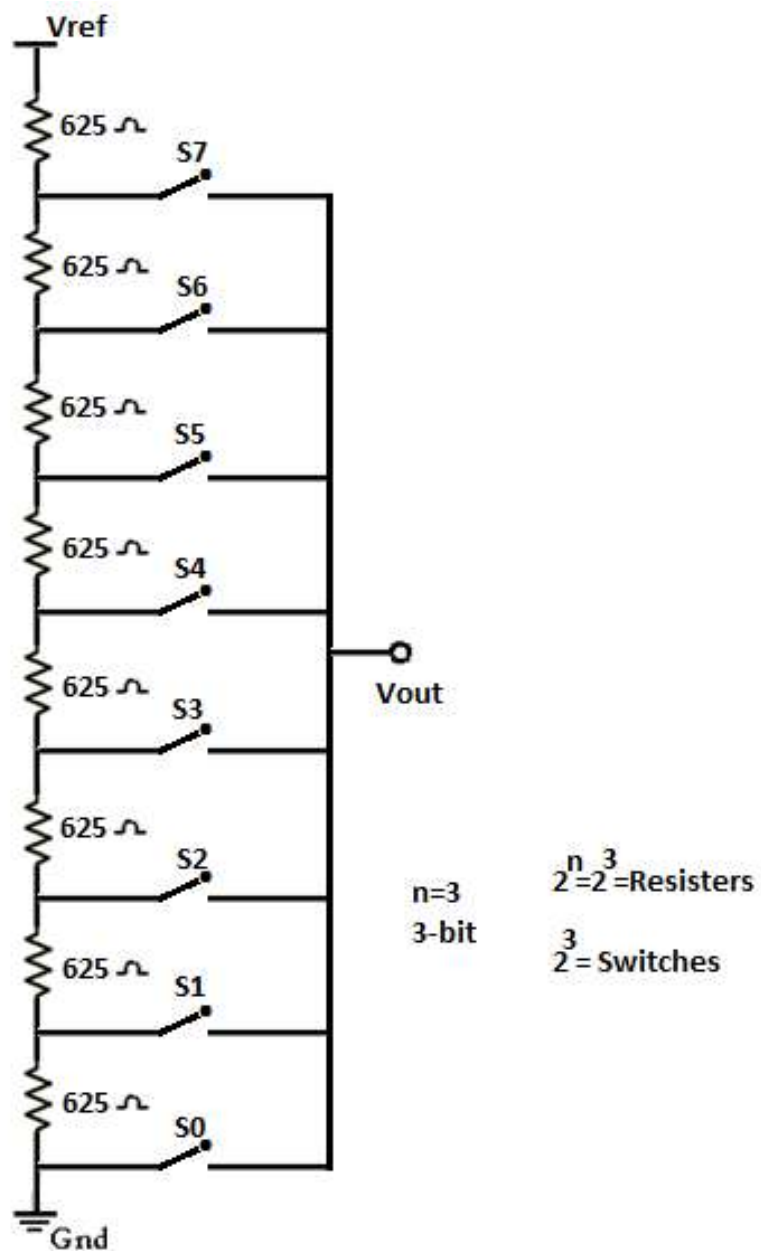
Challenge experiment:

1. Write a Verilog description for 4X4 bit booth algorithm and its test bench for verification, observe the waveform.

Flow chart:



2. Design a 3 bit resistor string DAC.



www.reva.edu.in

REVA University

Rukmini Knowledge Park, Kattigenahalli, Yelahanka, Bengaluru - 560 064