SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Department of Electronics and Communication Engineering

18ECC206J-VLSI DESIGN LAB

Laboratory Manual

College of Engineering & Technology

SRM Institute of Science and Technology

Tiruchrappalli – 621 105,

Near Samamyapuram Toll Plaza,

Tamilnadu

SRM Institute of Science and Technology
Department of Electronics and Communication Engineering

Vision of the Department

1. To create and disseminate knowledge in the area of Electronics and Communication Engineering through national and international accredited educational process.
2. To facilitate a unique learning and research experience to the students and faculty.
3. To prepare the students as highly ethical and competent professionals.

Mission of the Department

1. Build an educational process that is well suited to local needs as well as satisfies the national and international accreditation requirements.
2. Attract the qualified professionals and retain them by building an environment that fosters work freedom and empowerment.
3. With the right talent pool, create knowledge and disseminate, get involved in collaborative research with reputed universities and produce competent grandaunts.

Program Educational Objectives (PEO)

Program Educational Objectives for the Electronics and Communication Engineering program describes accomplishments that graduates are expected to attain within a few years of graduation. Graduates will be able to:

PEO1: Establish themselves as successful and creative practicing professional engineers both nationally and globally in the related fields of Electronics and Communication Engineering.

PEO2: Apply the acquired knowledge and skills in solving real-world engineering problems; develop novel technology and design products, which are economically feasible and socially relevant.

PEO3: Develop an attitude of lifelong learning for sustained career advancement and adapt to the changing multidisciplinary profession.

PEO4: Demonstrate leadership qualities, effective communication skills, and to work in a team of enterprising people in a multidisciplinary and multicultural environment with strong adherence to professional ethics.

18ECC206J VLSI Design

Sl. No.	Description of experiments	Session
Software's: Synthesis tool: Xilinx ISE; Simulation tool: Modelsim Simulator /HSPICE		
1	Lab 0: Verilog Operators	1
2	Lab 1: Realization of Combinational and Sequential Circuits	2
3	Lab 2: (a) Realization of digital circuits using behavioral modeling (b) Realization of MOS Circuit using Switch level Modeling	3
4	Lab 3: Design using FSM and ASM charts	4
5	Lab 4: Realization of VLSI Adders-I	5
6	Lab 5: Realization of VLSI Adders-II	6
7	Lab 6: Realization of VLSI Multiplier-I	7
8	Lab 7: Realization of VLSI Multiplier-II	8
9	Lab 8: Realization of RAM and ROM	9
10	Lab 9: Design and Analysis of CMOS inverter using HSPICE	10
11	Lab 10: (a) Design and Analysis of Complex CMOS gate using HSPICE (b) Design and Analysis of Pseudo NMOS gate using HSPICE	11
12	Lab 11: (a) Design and Analysis of AND/NAND gate in DCVSL using SPICE (b) Design and Analysis of Pass transistor gate and CPL gates using HSPICE	12
13	Lab 12: Design and analysis of 4- input Dynamic NAND gate using HSPICE	13
14	Lab 13: Model Practical Examination	14
15	Lab 14: End Semester Practical Examination	15

1.Evaluation Scheme

Component	Assessment Tool	Weightage
Practical	Course Learning Assessment P1	5%
	Course Learning Assessment P2	7.5%
	Course Learning Assessment P3	7.5%
	Course Learning Assessment P4 (Model Practical Exam and Mini Project)	5%
End-Semester Lab Examination		**25%**

2.Assessment Method – Lab Report

Component	Assessment Tool	Weightage
Lab Report – Practical	Prelab and Post lab	10
	Design, HDL Code, In-Lab Performance	15
	Simulation and Results	10
	Lab Report	05
	Total	40

- Soft copy of manual (student's Copy) will be posted in the Google classroom.
- Lab report format should follow the following details
    - Aim
    - Software required
    - Pre-Lab answers
    - Circuit Diagram, Truth table, Boolean Expression for each problem statement
    - Program code
    - Test bench code
    - Simulated output
    - Post-Lab answers
    - Results
- Lab report has to be uploaded in the Google classroom with in immediate next week.
- Design of Microprocessor components has been suggested for Mini Project Assessment.

Laboratory Report Cover Sheet

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY College of Engineering and Technology Department of Electronics and Communication Engineering
18ECC206J VLSI Design VI Semester, 2023-2024 (Even Semester)

Name:

Register No.:

Title of Experiment:

Date of Conduction:

Date of Submission:

Particulars	Max. Marks	Marks Obtained
Pre-Lab and Post Lab	10	
Design, HDL Code, In-Lab Performance	15	
Output verification &viva	10	
Lab Report	05	
Total	**40**	

REPORT VERIFICATION

Staff Name :

Signature:

Lab Experiment #1

Realization of Combinational and Sequential Circuits Using Gate-Level and Dataflow Modeling

1.1 Objective: To learn the design of combinational and sequential circuits using Gate-Level and Dataflow modeling in Verilog then simulating and synthesizing using EDA tools

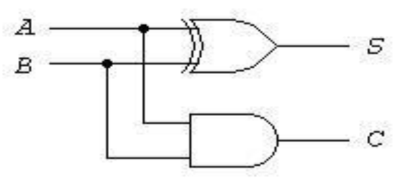
1.2 Software tools Requirement
Equipment's:
Computer with Xilinx and Modelsim Software Specifications:
HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk
Software's: Synthesis tool: Xilinx ISE
Simulation tool: Modelsim Simulator

1.3 Prelab Questions
(write pre lab Q & A in an A4 sheet)

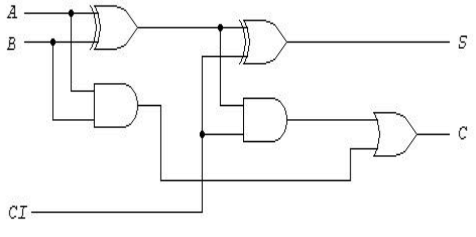
1. List the types of design methodologies for digital design with an example?
2. Give the difference between module and module instance.
3. What are built in gate primitives?
4. Give the use of net, reg. and wire data types.
5. Declare the following variables in Verilog:

1.4.1 Problem 1: Write a Verilog code to implement Ripple Carry Adder.
The following points should be taken care of:
1. Use assign statement to design sub module half adder
2. Use gate primitives to design full adder using half adder
3. Construct the top module 4-bit ripple carry adder using 1-bit full adder

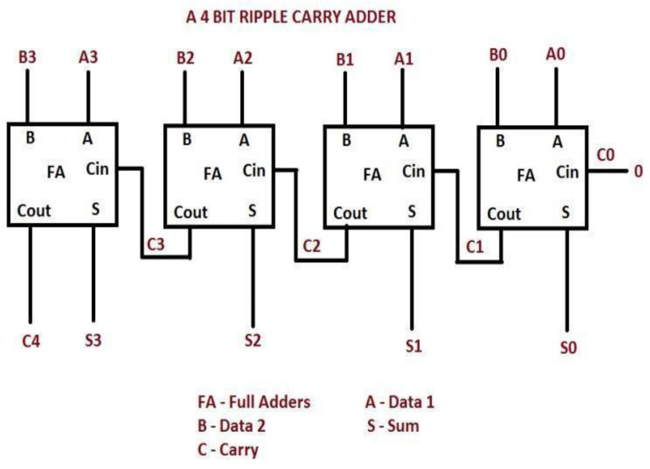
Logic Diagram – Problem 1
Half adder:



Full adder:



Ripple Carry Adder



Verilog Code - Problem 1

1.(a)HALF ADDER USING DATA FLOW MODEL

```
module Half_adder(s, cout, x, y);
output s;
output cout;
input x;
input y;
assign s=x^y;
assign cout=x&y;
endmodule
```

// TEST BENCH

```
module half_adder_tb_v;
//      Inputs

reg x;
// reg y;

//Outputs

wire s; wire cout;
// Instantiate the Unit Under Test (UUT)
Half_adder uut (.s(s), .cout(cout), .x(x), .y(y) );
initial begin
// Initialize Inputs
x = 0;              y = 0;
Wait 100 ns for global reset to finish #100; x=0; y=1;

#100; x=1; y=0; #100; x=1; y=1;
Add stimulus here

end
endmodule
```

1.(b) FULL ADDER USING STRUCTURAL MODEL

```
module Full_adder(sum, carry, x, y, cin);
output sum;
output carry;
input x;
input y;
input cin;
wire w1,w2,w3;
Half_adder h1(w1,w2,x,y);
Half_adder h2(sum,w3,w1,cin);
or g1(carry,w2,w3);
endmodule
```

// TEST BENCH

```
module full_adder_tb_v;
// Inputs
reg x; reg y; reg cin;
// Outputs
wire sum; wire carry;
// Instantiate the Unit Under Test (UUT)
Full_adder uut (.sum(sum), .carry(carry), .x(x), .y(y), .cin(cin)); initial begin
//Initialize Inputs x = 0; y = 0; cin = 0; Wait 100 ns for global reset to finish #100; x=0;
y=0; cin=1;
//#100; x=0; y=1; cin=0; #100; x=0; y=1; cin=1; #100; x=1; y=0; cin=0; #100; x=1; y=0;
cin=1; #100; x=1; y=1; cin=0; #100; x=1; y=1; cin=1;
//Add stimulus here
end
endmodule
```

1.(c) RIPPLE CARRY ADDER USING 1-BIT FULL ADDER

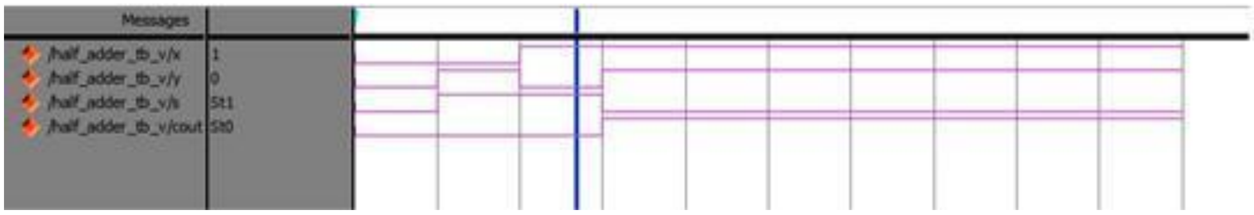
```
module ripple_carry_adder(S, cout, A, B, cin);
output [3:0]S;
output cout;
input [3:0]A;
input [3:0]B;
input cin;

wire c1,c2,c3;
full_adder f1(S[0],c1,A[0],B[0],cin);
full_adder f2(S[1],c2,A[1],B[1],c1);
full_adder f3(S[2],c3,A[2],B[2],c2);
full_adder f4(S[3],cout,A[3],B[3],c3);
endmodule
```

// TEST BENCH

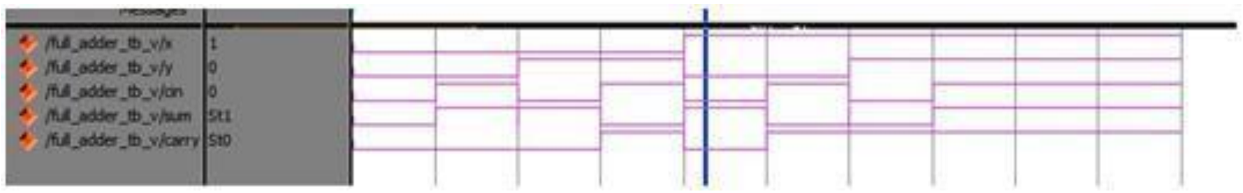
```
module ripple_carry_adder_tb_v;
// Inputs
reg [3:0] A; reg [3:0] B; reg cin;
```

```
// Outputs
wire [3:0] S; wire cout;
// Instantiate the Unit Under Test (UUT)
ripple_carry_adder uut (.S(S), .cout(cout), .A(A), .B(B), .cin(cin)); initial begin

//Initialize Inputs A = 0; B = 0; cin = 0;
//Wait 100 ns for global reset to finish #100; A=0; B=0; cin=1;
#100; A=0; B=1; cin=0; #100; A=0; B=1; cin=1; #100; A=1; B=0; cin=0; #100; A=1;
B=0; cin=1; #100; A=1; B=1; cin=0; #100; A=1; B=1; cin=1;
//Add stimulus here
End
endmodule
```

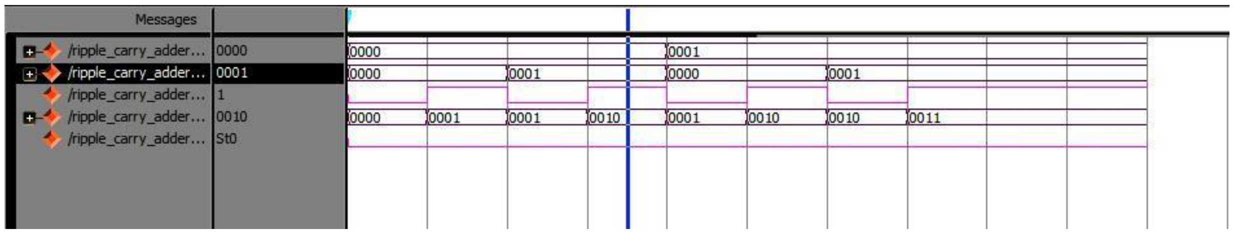
Waveforms – Problem 1



Half Adder using Data Flow Model



Full Adder using Structural Model

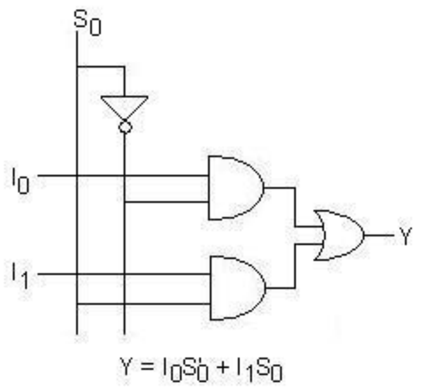

Messages
/ripple_carry_adder... 0000
/ripple_carry_adder... 0001
/ripple_carry_adder... 1
/ripple_carry_adder... 0010
/ripple_carry_adder... St0
0000
0001
0000
0001
0000
0001
0000
0001
0001
0010
0001
0010
0010
0011

Ripple Carry Adder using 1-Bit Full Adder

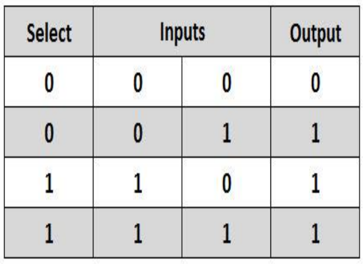
1.4.2 Problem 2: Write a Verilog code to implement Multiplexer.
The following points should be taken care of:
1.Use data flow model to design 2x1 Multiplexer
2.Use conditional statement to design 4x1 Multiplexer
3.Use Gate Level Model to design 8x1 Mux using two 4x1 Mux and one 2x1 Mux

Logic Diagram – Problem 2

2 x 1 Multiplexer

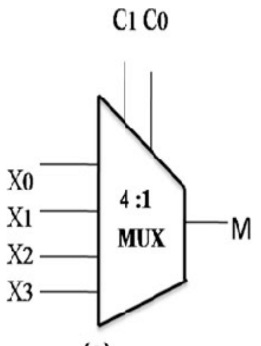


$Y = I_0S_0' + I_1S_0$

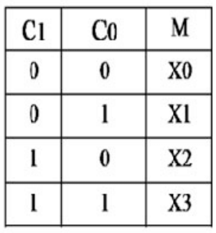


Select	Inputs		Output
0	0	0	0
0	0	1	1
1	1	0	1
1	1	1	1

4 x 1 Multiplexer



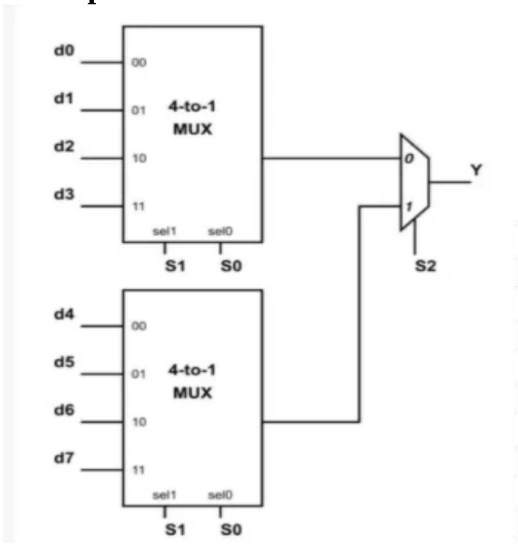
(a)

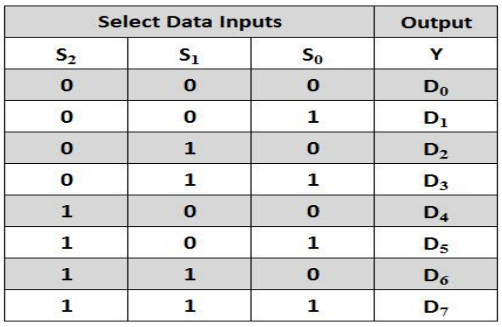


C1	C0	M
0	0	X0
0	1	X1
1	0	X2
1	1	X3

(b)

8 x 1 Multiplexer





Select Data Inputs			Output
S_2	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

Verilog Code - Problem 2

2.(a) MUX 2 X 1 DATA FLOW MODEL

```
module mux2_1(out, A, B, s);
output out;
input A;
input B;
input s;
assign out=(s)?A:B;
endmodule
```

```
// TEST BENCH
module mux2_1_tb_v;
// Inputs
reg A; reg B; reg s;
 //Outputs wire out;
 //Instantiate the Unit Under Test (UUT)
mux2_1 uut (.out(out), .A(A), .B(B),            .s(s));
initial begin
 //Initialize Inputs A = 0; B = 0; s = 0;
 //Wait 100 ns for global reset to finish #100; A=1; B=1; s=0;
#100; A=1; B=1; s=1;
//Add stimulus here
  end
endmodule
```

2.(b) MUX 4 X 1 conditional statement

```
module mux4_1(out, i, s);
output out;
input [3:0]i;
input [1:0]s;
assign out = s[0]?(s[1]?i[3]:i[2]):(s[1]?i[1]:i[0]);
endmodule
```

```
// TEST BENCH
module mux4_1_tb_v;
// Inputs
reg [3:0] i; reg [1:0] s;
//Output wire out;
//Instantiate the Unit Under Test (UUT) mux4_1 uut (.out(out), .I(I), .s(s)); initial begin
//Initialize Inputs
//Wait 100 ns for global reset to finish #100; I=4'b1101; s=2'b01;
#100; I=4'b0100; s=2'b10; #100; I=4'b0111; s=2'b11;
//Add stimulus here
end
endmodule
```

MUX 8 X 1 USING 4 X 1 MUX AND 2 X 1 MUX

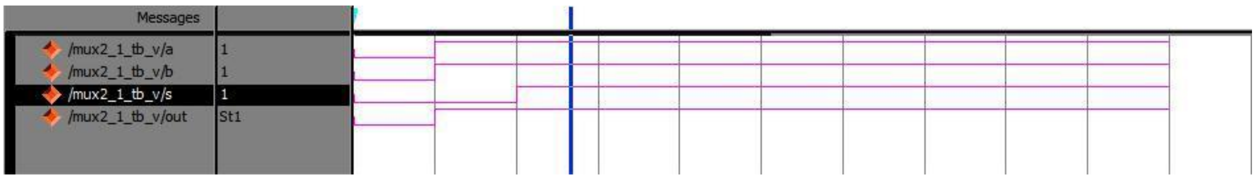
```
module mux8_1(out, i, s);
```

```
output out;
input [2:0]s, [7:0]i;
wire [1:0]w;
mux4__1 m1(w[0], i[3:0],  s[1:0]);
 mux4__1 m2(w[1], i[7:4], s[1:0]);
 mux2_1 m3(out,w[0],w[1], s[2]);
endmodule
```

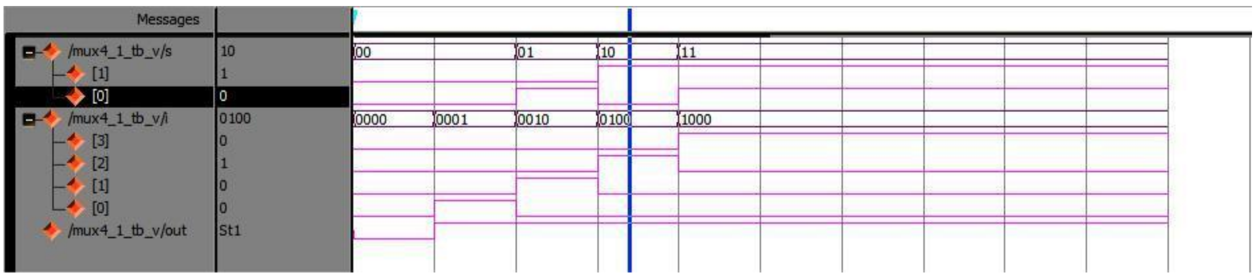
// TEST BENCH

```
module mux8_1_tb_v;
// Inputs
reg [2:0] s; reg [7:0] i;
//Outputs

wire out;

 //Instantiate the Unit Under Test (UUT) mux8_1 uut (.out(out), .s(s), .i(i)); initial begin
 //Initialize Inputs s = 0; i = 0;
 //Wait 100 ns for global reset to finish #100; s=3'b000; i=8'b00000001; #100; s=3'b001;
    i=8'b00000010; #100; s=3'b010; i=8'b00000100; #100; s=3'b011; i=8'b00001000; #100;
    s=3'b100; i=8'b00010000;
#100; s=3'b101; i=8'b00100000;
#100; s=3'b110; i=8'b01000000;
#100; s=3'b111; i=8'b10000000;
end
endmodule
```

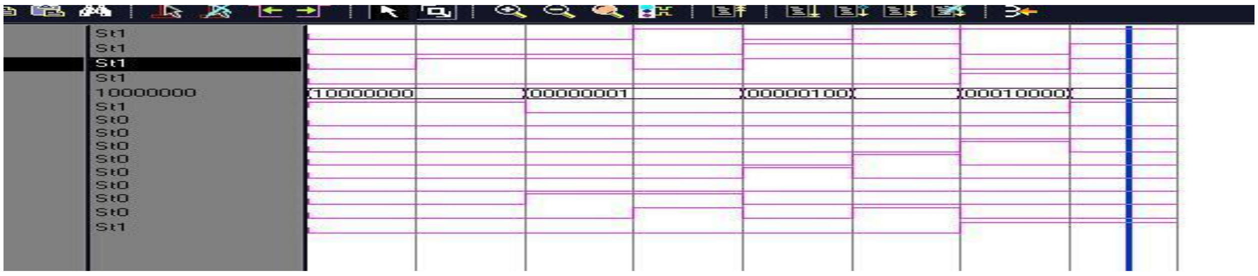
Waveforms – Problem 2



2 X 1 MUX Using Data Flow Model



4 X 1 MUX using conditional Operator



```
St1
St1
St1
St1
10000000      10000000      00000001      00000100      00010000
St1
St0
St0
St0
St0
St0
St0
St0
St0
St1
```

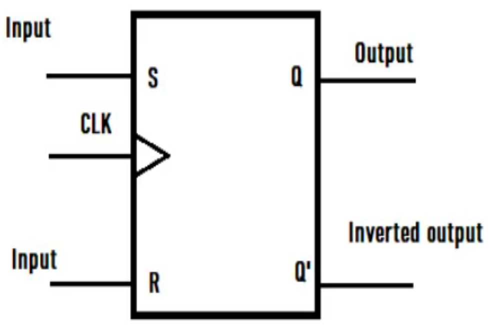
8 X 1 MUX Using 4 X 1 MUX and 2 X 1

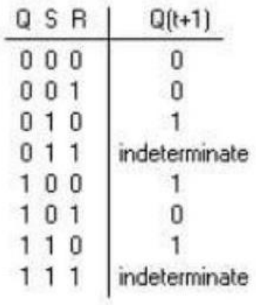
1.4.3 Problem 3: Design of Sequential Circuits: Write a Verilog code to implement Flip flops.

The following points should be taken care of:

1. Use gate level model to design positive edge triggered SR flip
2. Use data flow model to design negative edge triggered JK flip flop

Logic Diagram – Problem 3

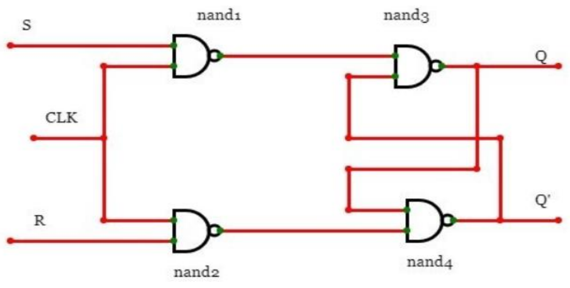


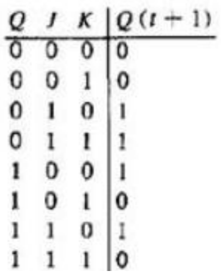


Q S R	Q(t+1)
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	indeterminate
1 0 0	1
1 0 1	0
1 1 0	1
1 1 1	indeterminate

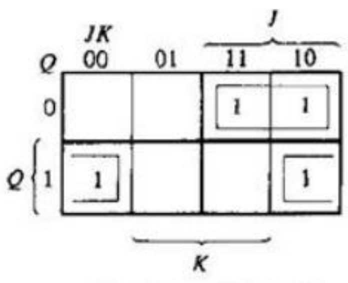
(b) Truth table

Clocked SR flip-flop

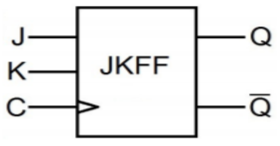




Q	J	K	$Q(t+1)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



$Q(t+1) = JQ' + K'Q$



$$Q^{+} = J\,\overline{Q} + \overline{K}\,Q.$$

Verilog Code - Problem 3

3.(a) POSITIVE EDGE TRIGGERED SR FLIPFLOP USING GATE LEVEL MODEL

```
module srff(q, qbar, s, r, clk);
output q, qbar;
input s;
input r;
input clk;
 wire nand1_out;
wire nand2_out;
nand n1(nand1_out, clk, s);
nand n2(nand2_out, clk, r);
nand n3(q, nand1_out, qbar);
nand n4(qbar, nand2_out, q);
endmodule
```

// **TEST BENCH**

```
module srff_tb_v;
// Inputs
reg s;
reg r;
reg clk;
// Outputs
wire q;
wire qbar;
// Instantiate the Unit Under Test (UUT)
srff uut (
.q(q),
.qbar(qbar),
.s(s),
.r(r),
.clk(clk)
);
initial begin
// Initialize Inputs
s = 0;r = 1;clk = 1;
// Wait 100 ns for global reset to finish
#100 s = 1;r = 0;  clk = 1;
#100 s = 0;r = 0; clk = 1;
#100 s = 1;r = 1;  clk = 1;
// Add stimulus here
end
endmodule
```

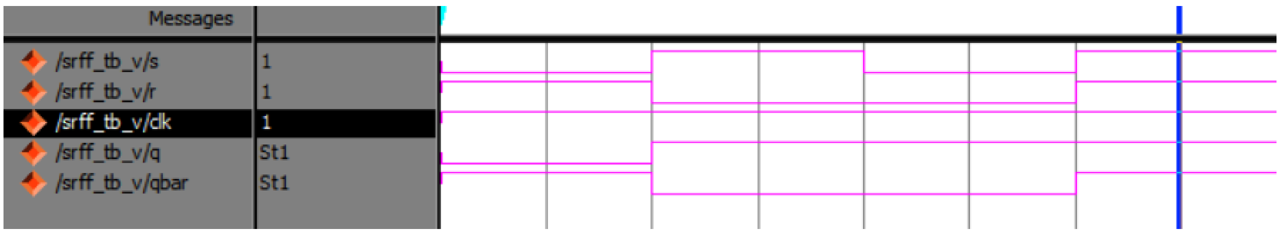
3.(b) NEGATIVE EDGE TRIGGERED JK FLIPFLOP USING DATA FLOW MODEL

```
module jkff(q,qb, clk, clr, j, k);
output q, qb;
input clk;
input clr;
input j, k;
reg q;
initial
q=0;
always@(negedge clk)
begin
q= (j & (~q)) | ((~k) & q);
end
assign qb=~q;
endmodule
```

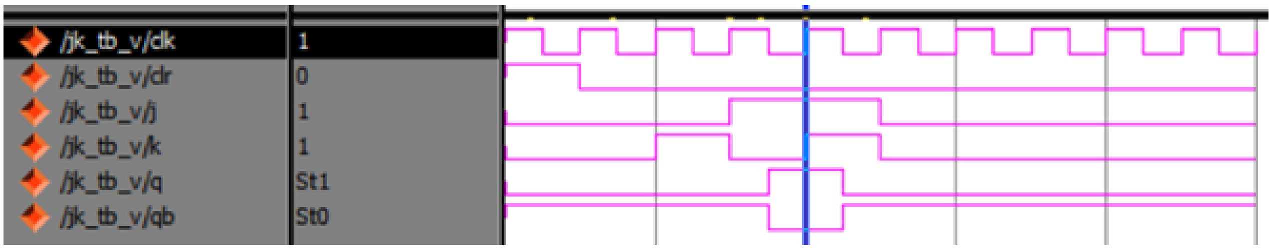
TEST BENCH

```
module jk_tb_v;
// Inputs
reg clk;
reg clr;
reg j;
reg k;

// Outputs
wire q;
wire qb;

// Instantiate the Unit Under Test (UUT)
jkff uut (
.q(q),
.qb(qb),
.clk(clk),
.clr(clr),
.j(j),
.k(k)
);

initial begin
// Initialize Inputs
clk = 1;clr = 1;j = 0;k = 0;
// Wait 100 ns for global reset to finish
#100 clr = 0;j = 0;k = 0;
#100 clr = 0;j = 0;k = 1;
#100 clr = 0;j = 1;k = 0;
#100 clr = 0;j = 1;k = 1;
#100 clr = 0;j = 0;k = 0;
// Add stimulus here
end
always #50 clk=~clk;
endmodule
```

Waveforms – Problem 3



SR Flip-flop using Gate level model



JK Flip-flop using Data flow model

1.5 Post lab

1.Draw the block diagram of 4x16 decoder. Use 3x8 decoder to design 4x16 decoder using Gate level model.

1.6 Result:

Thus, the design of combinational and Sequential logic circuits was simulated in Verilog and synthesized using EDA tools.

Lab Experiment #2
Realization of Digital Circuits Using Behavioral and Switch Level Modeling

2.1 Objective: To realize the design of digital circuits in Verilog using behavioral and switch level modelling then simulating and synthesizing using EDA tools.

2.2 Software tools Requirement

Equipments:

Computer with Xilinx and Modelsim Software Specifications:
HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk
Softwares: Synthesis tool: Xilinx ISE.
Simulation tool: ModelSim Simulator

2.3 Prelab Questions

(write pre lab Q & A in an A4 sheet)

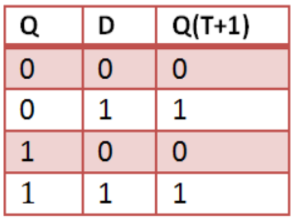
1. Write the difference between initial and always block.
2. List the Reduction and Logical Operators.
3. Give the use of Blocking and Nonblocking statments.
4. Differentiate case, casex and casez statements.

2.4.1 Problem 1: Write a Verilog code to implement Flip flops.
The following points should be taken care of:

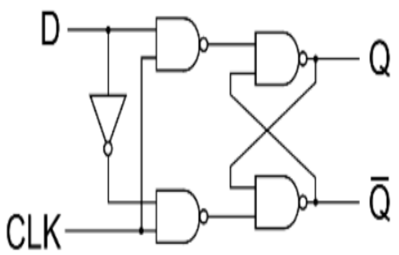
1. Use If statement to design positive edge triggered D flip
2. Use case statement to design negative edge triggered T flip flop

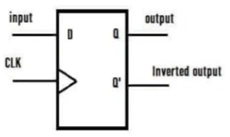
Logic Diagram

D flip flop

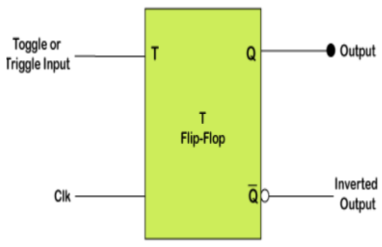


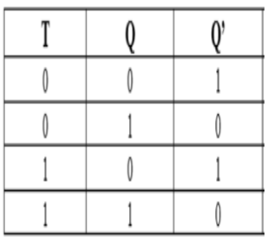
Q	D	Q(T+1)
0	0	0
0	1	1
1	0	0
1	1	1



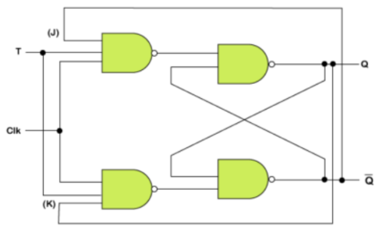


T flip flop





T	Q	Q'
0	0	1
0	1	0
1	0	1
1	1	0



Verilog Code - Problem 1

1.(a) POSITIVE EDGE TRIGGERED D FLIPFLOP USING IF STATEMENT

```
module dff(q, qbar, d, clk, clear);
```

```
 output q;
 output qbar;
 input d;
 input clk;
 input clear;
 reg q, qbar;
always@(posedge clk or posedge clear)
begin
if(clear== 1)
begin
q <= 0;
qbar <= 1;
end
else
begin
q <= d;
qbar = !d;
end
end
endmodule
```

// **TEST BENCH**

```
module dff_tb_v;
// Inputs
reg d;
reg clk;
reg clear;
// Outputs
wire q;
wire qbar;

// Instantiate the Unit Under Test (UUT)
dff uut (
.q(q),
.qbar(qbar),
.d(d),
.clk(clk),
.clear(clear)
);
initial begin
// Initialize Inputs
d = 0;clk = 0;clear = 1;
// Wait 100 ns for global reset to finish
#100 d = 0;clear = 0;
#100 d = 1;clear = 0;
#100 d = 1;clear = 1;
// Add stimulus here
end
always #50 clk=~clk;
endmodule
```

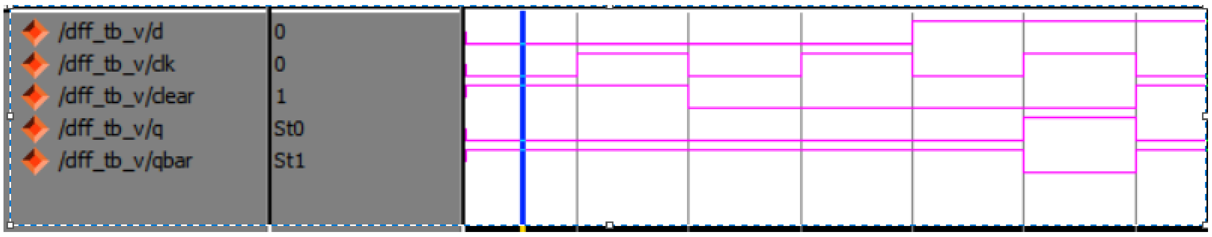
1.2) NEGATIVE EDGE TRIGGERED T FLIPFLOP USING CASE STATEMENT

```
module tffcase(q, clk, clr, t);
output q;
input clk;
input clr;
input t;
reg q;
initial
q=0;
always@(negedge clk)
begin
case({clr,t})
2'b10: q=0;
2'b00: q=q;
2'b01: q=~q;
endcase
end
endmodule
```

// TEST BENCH

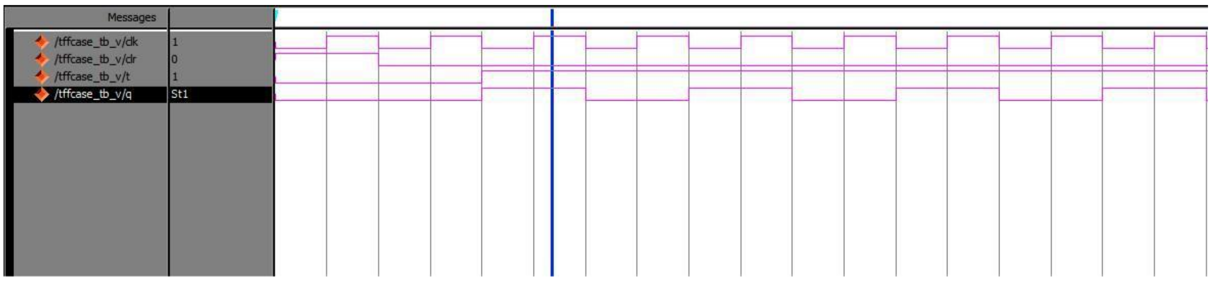
```
module tffcase_tb_v;
// Inputs
reg clk; reg clr; reg t;
//Outputs wire q;
//Instantiate the Unit Under Test (UUT) tffcase uut (.q(q),.clk(clk),.clr(clr),.t(t)); initial begin
//Initialize Inputs clk = 0; clr = 1; t = 0;
//Wait 100 ns for global reset to finish #100; clr=0;
#100; clr=0; t=1;
 //Add stimulus here
end
always
#50 clk=~clk;
endmodule
```

Waveforms – Problem 1



```
/dff_tb_v/d      0
/dff_tb_v/clk    0
/dff_tb_v/clear  1
/dff_tb_v/q      St0
/dff_tb_v/qbar   St1
```

D Flipflop using IF statement



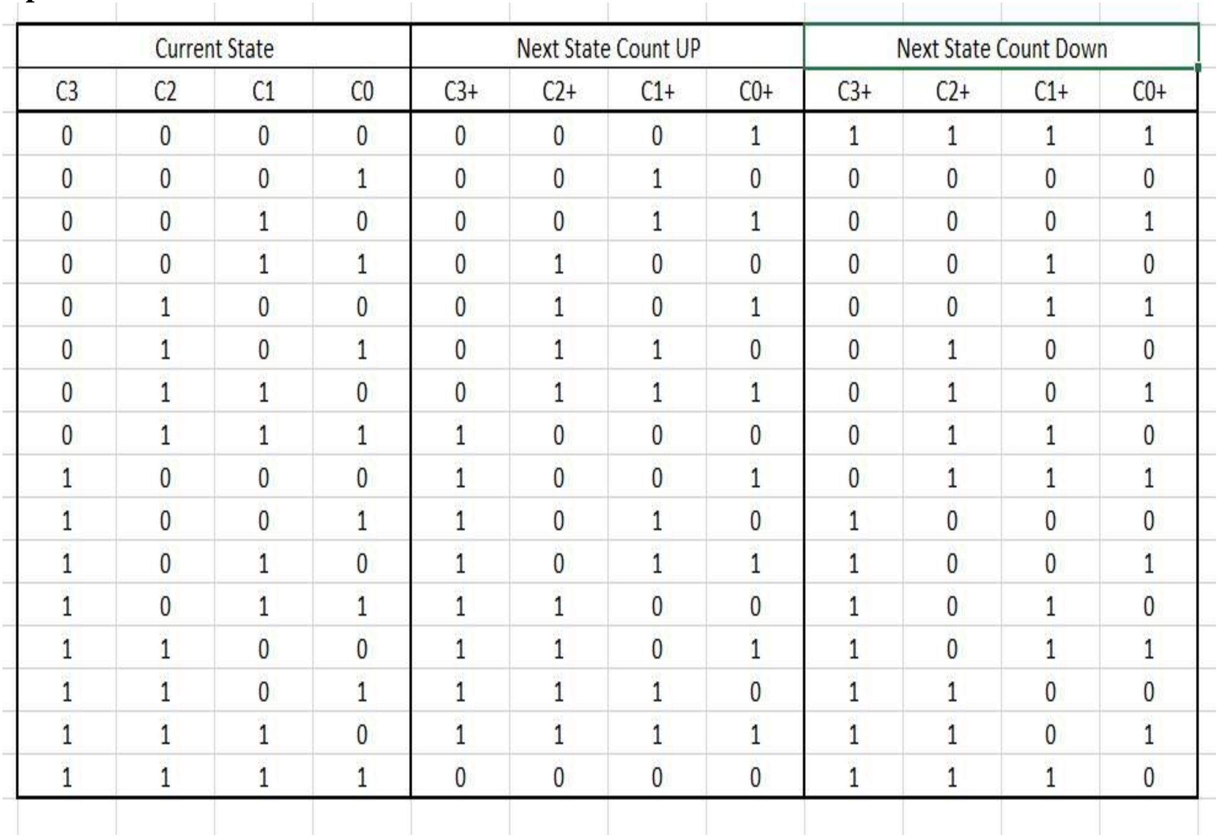
```
Messages
/tffcase_tb_v/clk   1
/tffcase_tb_v/clr   0
/tffcase_tb_v/t     1
/tffcase_tb_v/q     St1
```

T Flipflop using CASE statement

2.4.2 Problem 2: Write a Verilog code to implement Counters.
The following points should be taken care of:

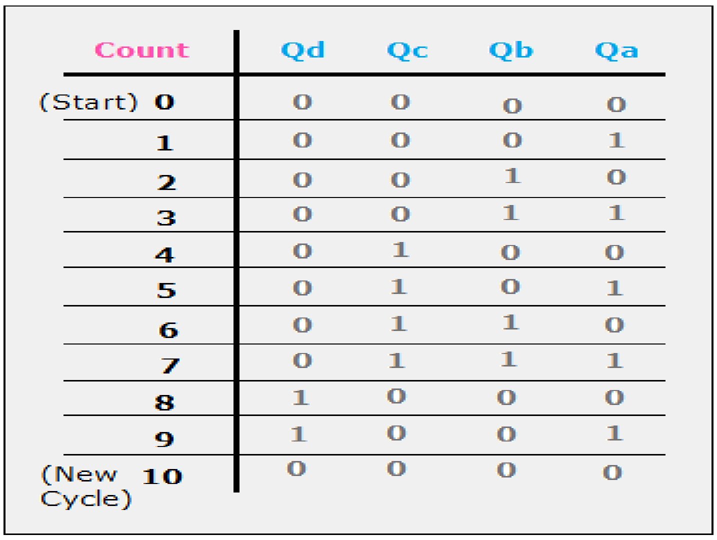
1. Use behavioral model to design Up-Down Counter. When mode ='1' do up counting and for mode='0' do down counting.
2. Use behavioral model to design Mod-N counter.
3. Design SISO registers using behavioral model.

Logic Diagram – Problem 2
Up-Down Counter:



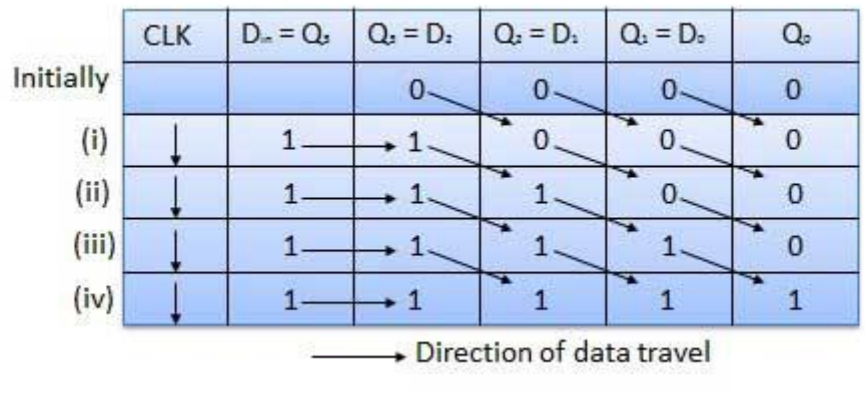
Current State				Next State Count UP				Next State Count Down			
C3	C2	C1	C0	C3+	C2+	C1+	C0+	C3+	C2+	C1+	C0+
0	0	0	0	0	0	0	1	1	1	1	1
0	0	0	1	0	0	1	0	0	0	0	0
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	0	1	0
0	1	0	0	0	1	0	1	0	0	1	1
0	1	0	1	0	1	1	0	0	1	0	0
0	1	1	0	0	1	1	1	0	1	0	1
0	1	1	1	1	0	0	0	0	1	1	0
1	0	0	0	1	0	0	1	0	1	1	1
1	0	0	1	1	0	1	0	1	0	0	0
1	0	1	0	1	0	1	1	1	0	0	1
1	0	1	1	1	1	0	0	1	0	1	0
1	1	0	0	1	1	0	1	1	0	1	1
1	1	0	1	1	1	1	0	1	1	0	0
1	1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	0	0	0	0	1	1	1	0

Mod-N Counter:



Count	Qd	Qc	Qb	Qa
(Start) 0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
(New Cycle) 10	0	0	0	0

SISO Register:



	CLK	$D_{in} = Q_3$	$Q_3 = D_2$	$Q_2 = D_1$	$Q_1 = D_0$	Q_0
Initially			0	0	0	0
(i)	↓	1	1	0	0	0
(ii)	↓	1	1	1	0	0
(iii)	↓	1	1	1	1	0
(iv)	↓	1	1	1	1	1

$\longrightarrow$ Direction of data travel

Verilog Code - Problem 2

2.(1) UP DOWN COUNTER USING BEHAVIOURAL MODEL

```
module updowncntr(q, clr, clk, mode);
 output reg [3:0] q;
 input clr;
 input clk;
 input mod;
 always@(posedge clk)
 begin
 case({clr,mod})
 2'b11 : q=0;
 2'b10 : q=0;
 2'b01 : q=q+1;
 2'b00 : q=q-1;
 endcase
 end
endmodule
```

// TEST BENCH

```
module updowncntr_tb_v;
 // Inputs
 reg clr; reg clk; reg mod;
 //Outputs wire [3:0]q;

 //Instantiate the Unit Under Test (UUT) updowncntr uut (.q(q),.clr(clr),.clk(clk),.mod(mod));
 initial begin
 // Initialize Inputs
 clr = 1; clk = 0; mod = 1;
 //Wait 100 ns for global reset to finish #100; clr=0;
 #1000; mod=0;
 //Add stimulus here
  end
  always
  #50 clk=~clk;
  endmodule
```

2.(2) Mod N-COUNTER USING BEHAVIOURAL MODEL

```
module modN_ctr
# (parameter N = 10,
parameter WIDTH = 4)
( input   clk,
input   rstn,
output  reg[WIDTH-1:0] out);
always @ (posedge clk) begin
if (rstn) begin
out <= 0;
end else begin
if (out == N-1)
out <= 0;
else
out <= out + 1;
end
end
endmodule
```

// TEST BENCH

```
module modncounter_tb_v;

// Inputs
reg clk;
reg rstn;

// Outputs
wire [3:0] out;

// Instantiate the Unit Under Test (UUT)
modN_ctr uut (
.clk(clk),
.rstn(rstn),
.out(out)
);
initial begin
// Initialize Inputs
clk = 1;rstn = 1;
// Wait 100 ns for global reset to finish
#100 rstn=0;
// Add stimulus here
end
lways #50 clk=~clk;
endmodule
```

2.(3) SISO REGISTER USING BEHAVIOURAL MODEL

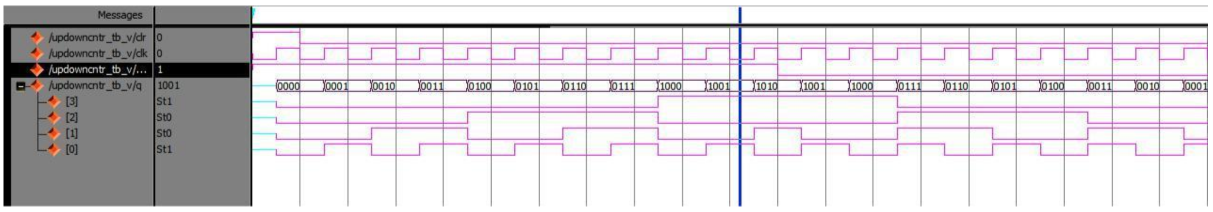
```
module siso_register(shift_out, shift_in, clk);
input  shift_in;
input clk;
output  shift_out;
reg shift_out;
reg  [2:0] data;
always @(negedge clk)
begin
```

```
data[0] <= shift_in ;
data[1] <= data[0];
data[2] <= data[1];
shift_out <= data[2];
end
endmodule
```

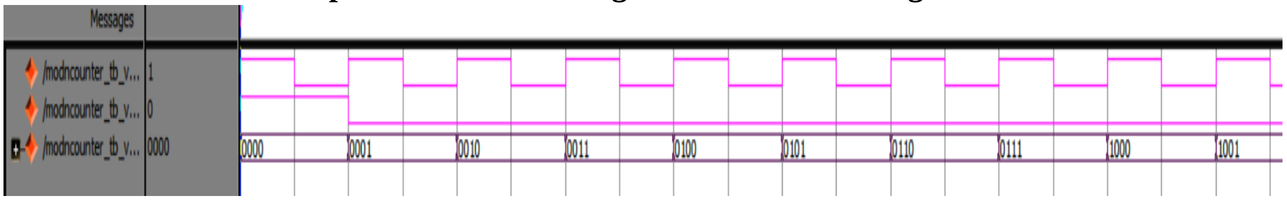
// TEST BENCH

```
module siso_tb_v;

// Inputs
reg shift_in;
reg clk;
// Outputs
wire shift_out;
// Instantiate the Unit Under Test (UUT)
siso_register uut (
.shift_out(shift_out),
.shift_in(shift_in),
.clk(clk)
);

initial begin
// Initialize Inputs
shift_in = 1;
clk = 1;

// Wait 100 ns for global reset to finish

#100 shift_in = 0;
#100 shift_in = 1;
#100 shift_in = 0;
// Add stimulus here

end
always #50 clk=~clk;
endmodule
```

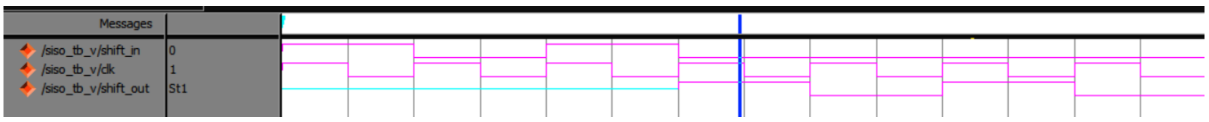
Waveforms – Problem 2



Up Down counter using behavioral modelling



Mod-N counter using behavioral modelling

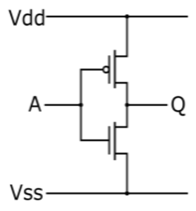


SISO register using behavioral modelling

Problem 2.4.3:

1. Design inverter logic using Verilog switch level modeling and verify the simulation result using test bench.
2. Design two inputs CMOS NAND logic using Verilog switch level modeling and verify the simulation result using test bench.
3. Design 2:1 Mux using CMOS switches and write Verilog coding using switch level modeling and verify the simulation result.

Logic Diagram-Problem 1



2.4.3 Verilog Code - Problem 1

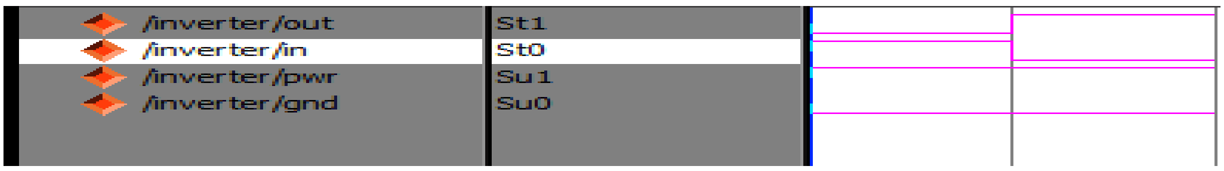
CMOS Inverter

```
module inverter(out,data);
output out;
input data;
supply1 vdd;
supply0 gnd;
pmos p1(out,vdd,data);
nmos n1(out,gnd,data);
endmodule
```

CMOS Inverter_TB

```
module inverter_tb_v;
reg data;
inverter uut (.out(out), .data(data));
initial begin
data = 0;
#100 data = 1;
end
endmodule
```

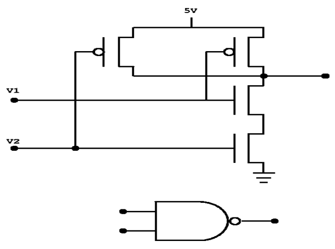
6.4 Waveforms – Problem 1


/inverter/out St1
/inverter/in St0
/inverter/pwr Su1
/inverter/gnd Su0

CMOS inverter

Logic Diagram-Problem 2

Two input NAND



2.4.3 Verilog Code - Problem 2

CMOS NAND Logic

```
module nandcmos(out,a,b);
wire t;
output out;
input a,b;
supply1 vdd;
supply0 gnd;
pmos m1(out,vdd,a);
pmos m2(out,vdd,b);
nmos m3(out,t,a);
nmos m4(t,gnd,b);
endmodule
```

CMOS NAND Logic_TB

```
module nandcmos_tb_v;
reg a; reg b;
wire out;
nandcmos uut (
.out(out),  .a(a), .b(b)
);
initial begin
a = 0;  b = 0;
#100 a = 0;  b = 1;
#100 a = 1;  b = 0;
#100 a = 1;  b = 1;
end
endmodule
```

2.4.3Waveforms – Problem 2

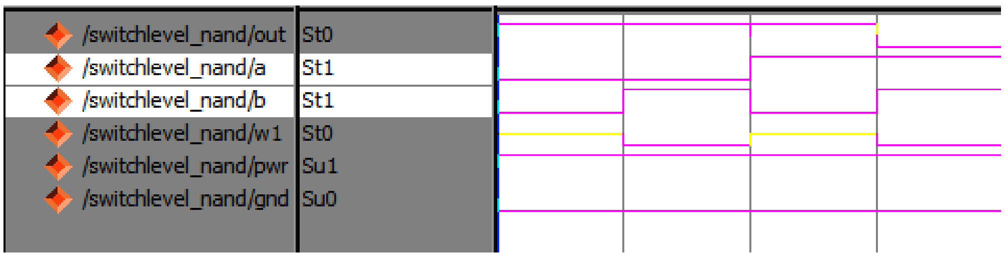
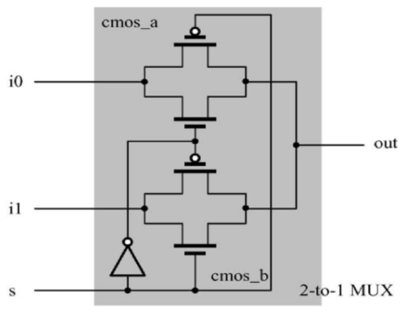

/switchlevel_nand/out St0
/switchlevel_nand/a St1
/switchlevel_nand/b St1
/switchlevel_nand/w1 St0
/switchlevel_nand/pwr Su1
/switchlevel_nand/gnd Su0

Fig 2.4.3 Two input NAND

Logic Diagram-Problem 3
2:1 Mux using CMOS



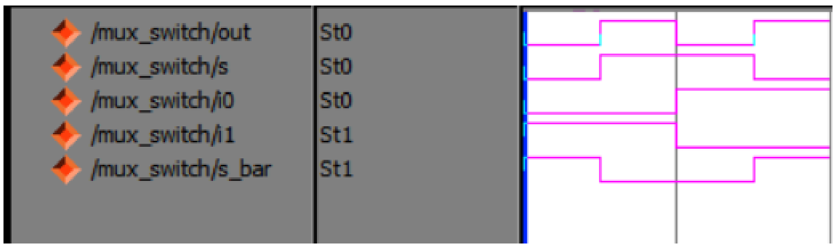
2.4.3 Verilog Code - Problem 3
CMOS Mux 2X1

```
module cmos_mux(out,s,i1,i0);
output out;
input s,i0,i1;
wire sbar;
not n1(sbar,s);
cmos(oout,i0,sbar,s);
cmos(out,i1,s,bar);
endmodule
```

CMOS Mux 2X1_TB

```
module cmos_mux_tb_v;
reg s;reg i1;reg i0;
wire out;
cmos_mux uut (.out(out), .s(s),
.i1(i1),.i0(i0));
initial begin
s = 0;i1 = 1;i0 = 0; #100;
s = 0;i1 = 0;i0 = 1; #100;
s = 1;i1 = 0;i0 = 1; #100;
s = 1;i1 = 1;i0 = 1;
end
endmodule
```

2.4.3Waveforms – Problem 3



2X1 Multiplexer using CMOS

2.5 Post Lab :

Write a Verilog HDL Code to implement a SIPO and PIPO shift registers.

2.6 Result:

Thus, the design of Digital circuits using behavioral and Switch level modelling is simulated in Verilog and synthesized using EDA tools.

Lab Experiment #3
Design of Finite State Machine (FSM) and Algorithmic State

3.1 Objective:

To learn the design of Finite State Machine (FSM) and Algorithmic State Machine (ASM) for any application in Verilog, the simulating and synthesizing using EDA tools.

3.2 Tools Required:

1. Synthesis tool: Xilinx ISE
2. Simulation tool: Modelsim simulator

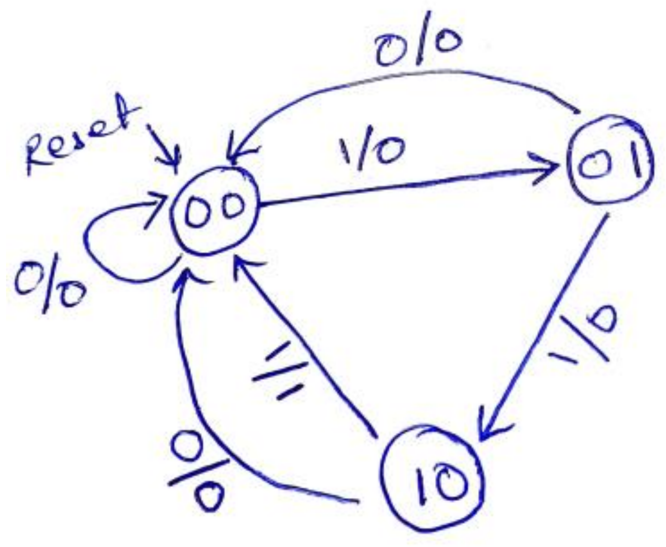
3.3 Pre Lab Questions:

1. Draw the simple model of FSM.
2. What is the basic Algorithm of Sequence Detector?
3. List the difference between Mealy and Moore Model.
4. What is ASM Chart and what are its main components?

3.4.1 Problem:

1. Implement Sequence recognizer for detecting three successive 1's using Mealy Model (Behavioural Modelling).

Mealy model for 111 Detector (FSM):



Verilog code for Mealy Model FSM: (Behavioral Modelling)

moule fsm_111_mealey(o,reset,a,clk);

output reg o;

input reset,a,clk;

reg[1:0]pre_s, nxt_s;

initial begin

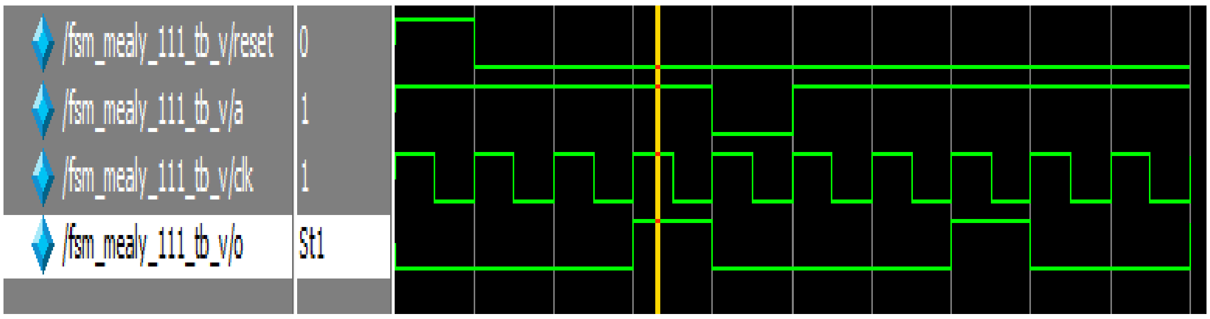
```
pre_s=2’b00;
end
always@(posedge clk)
begin
if(reset==1)
begin pre_s=2’b00; o=0;end
else
begin
case(pre_s)
2’b00:begin if (a==1)begin o=0; nxt_s=2’b01;pre_s=nxt_s; end
                    else begin o=0; nxt_s=2’b00;pre_s=nxt_s; end end
2’b01:begin if (a==1)begin o=0; nxt_s=2’b10;pre_s=nxt_s; end
                    else begin o=0; nxt_s=2’b00;pre_s=nxt_s; end end
2’b10:begin if (a==1)begin o=1; nxt_s=2’b00;pre_s=nxt_s; end
                    else begin o=0; nxt_s=2’b00;pre_s=nxt_s; end end
              default:pre_s2’b00;
              endcase
              end
end
endmodule
```

Test Bench:

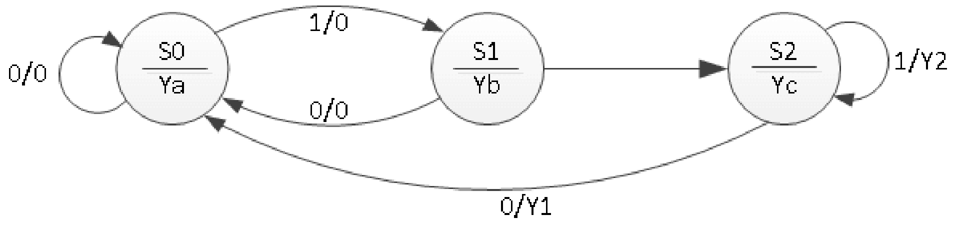
```
module fsm_mealy_111_tb_v;
// Inputs
reg reset;
reg a;
reg clk;
// Outputs
wire o;
// Instantiate the Unit Under Test (UUT)
fsm_mealy_111 uut (
.o(o),
.reset(reset),
.a(a),
.clk(clk)
);
initial begin
// Initialize Inputs
```

```
reset = 1;
a = 1;
clk = 1;
// Wait 100 ns for global reset to finish
#100;
      reset = 0;
a = 1;
#100;
reset = 0;
a = 1;
#100;
reset = 0;
a = 1;
#100;
 reset = 0;
a = 0;
#100;
 reset = 0;
a = 1;
#100;
// Add stimulus here
end
  always #50 clk=~clk;
endmodule
```

Simulation Result:

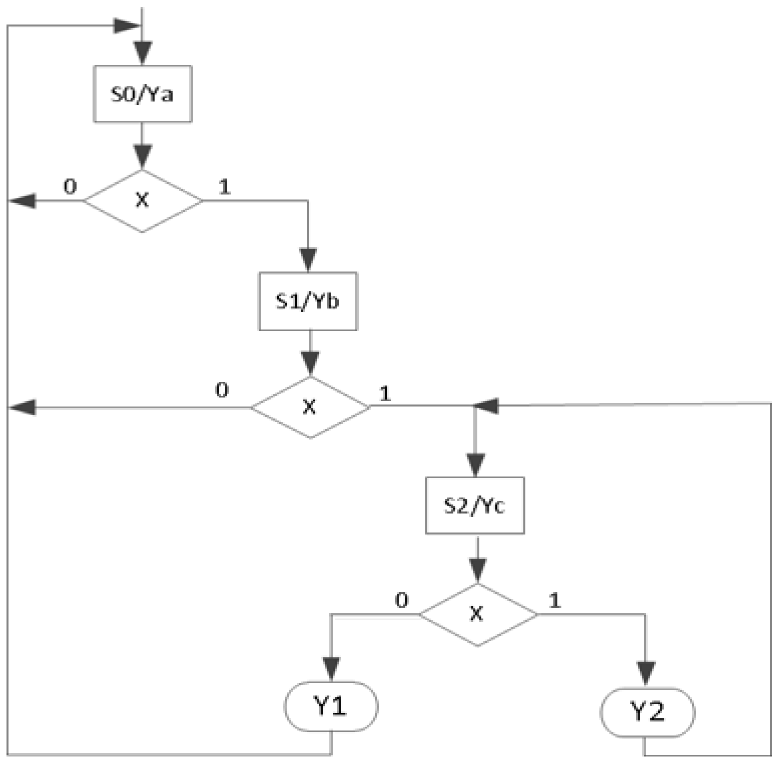


3.4.2 Consider the following state graph of a sequential network which has both Mealy and Moore outputs. The outputs Y1 and Y2 are the Mealy outputs and so should be conditional outputs. The Ya, Yb, and Yc are the Moore outputs so they should be part of state box. Input X can either be “0” or “1” and hence it should be part of the decision box.



Draw the ASM Chart for the above state graph and implement using Verilog.

ASM Chart:



Verilog Program:

```
module asm (input clk, input x, output reg ya, output reg yb, output reg yc, output reg
y1,

output reg y2);

    reg [1:0] state, nextstate;

    parameter [1:0] S0=0, S1=1, S2=2;

    always @(posedge clk)// always block to update state

    state <= nextstate;

    always @(state or x) // always block to compute both Mealy output & next state

    begin

        y1 = 1'b0; y2 = 1'b0;

    case (state)

    S0: if(x)

        nextstate = S1;

     else

        nextstate = S0;
```

```
S1: if(x)
       nextstate = S2;
    else
       nextstate = S0;
S2: if(x)
begin
       y2 = 1'b1;
       nextstate = S1;
end
else
begin
        y1 = 1'b1;
        nextstate = S0;
 end
default:nextstate = S0;
endcase
end
always @(state) // always block to compute Moore output
begin
ya = 1'b0; yb = 1'b0; yc = 1'b0;
case(state)
//begin
S0: ya = 1'b1;
S1: yb = 1'b1;
 S2: yc = 1'b1;
 //end
default: begin
ya = 1'b0;
yb = 1'b0;
yc = 1'b0;
end
endcase
```

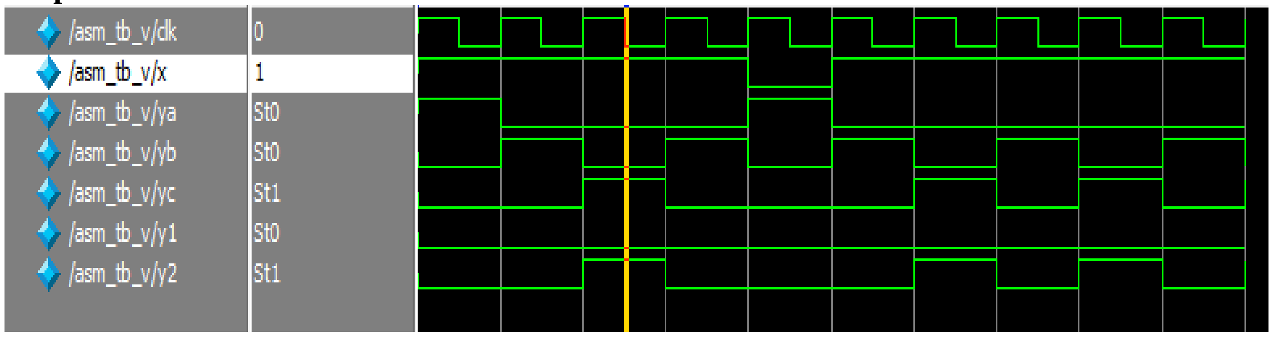
```
                end
                endmodule
```

Test Bench:

```
module asm_tb_v;
// Inputs
reg clk;
reg x;
// Outputs
wire ya;
wire yb;
wire yc;
wire y1;
wire y2;
// Instantiate the Unit Under Test (UUT)
asm uut (
.clk(clk),
.x(x),
.ya(ya),
.yb(yb),
.yc(yc),
.y1(y1),
.y2(y2)  );
initial begin
// Initialize Inputs
x = 1;
clk = 1;
// Wait 100 ns for global reset to finish
#100;
    x = 1;
#100;
x = 1;
#100;
```

```
x = 1;
#100;
x = 0;
#100;
// Add stimulus here
end
  always #50 clk=~clk;
        // Add stimulus here
endmodule
```

Output:



3.5 Post lab Questions:

1. Write Verilog code to implement an FSM using Moore Machine.

3.6 Result:

All the FSM Logic and ASM logic were executed and verified.

Lab Experiment # 4
Realization of Carry Look-Ahead Adder

4.1 Objective: To design and simulate the carry look-ahead adder (4-bit) in verilog and synthesize using EDA tools.

4.2 Software tools Requirement

Equipment's:

Computer with Xilinx and Modelsim Software Specifications:
HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk
Software's: Synthesis tool: Xilinx ISE
Simulation tool: Modelsim Simulator

4.3 Prelab Questions

(Write pre lab Q & A in an A4 sheet)

1.Write the expression for Propagate and generate term in a CLA.
2.List the efficient method for implementing 64- adder using CLA technique?

4.4 Problem: Write a Verilog code to implement the 4-bit Carry Look-ahead adder using structural model.

Logic Diagram:

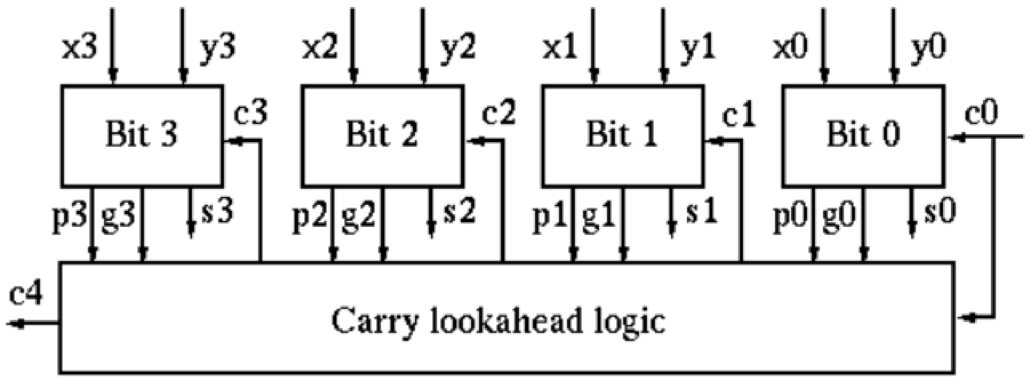


Fig. 4.1: 4-bit Carry Look-ahead Adder

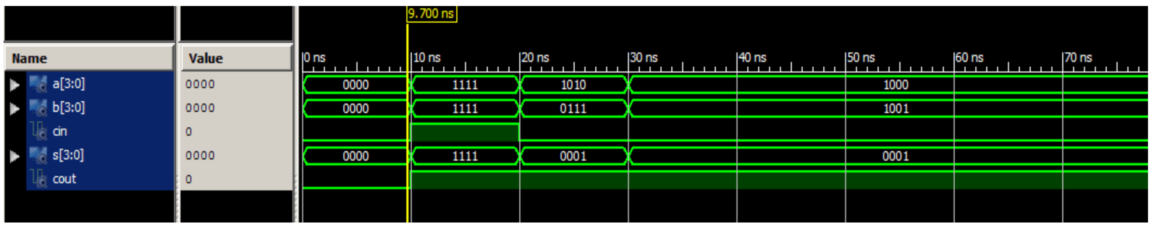
Source code:

```
Module CLA_4bit(sum,c_4,a,b,c_0);
input [3:0]a,b;
input c_0;
output [3:0]sum;
output c_4;
wire p0,p1,p2,p3,g0,g1,g2,g3;
wire c1,c2,c3,c4;
assign
p0=a[0]^b[0],
p1=a[1]^b[1],
p2=a[2]^b[2],
p3=a[3]^b[3],
g0=a[0]&b[0],
g1=a[1]&b[1],
g2=a[2]&b[2],
g3=a[3]&b[3];
assign
c1=g0|(p0&c_0),
c2=g1|(p1&g0)|(p1&p0&c_0),
c3=g2|(p2&g1)|(p2&p1&g0)|(p2&p1&p0&
c_0),
c4=g3|(p3&g2)|(p3&p2&p1&g1)|(p3&p2&
p1&g0)|(p3&p2&p1&p0&c_0);
assign
sum[0]=p0^c_0,
sum[1]=p1^c1,
sum[2]=p2^c2,
sum[3]=p3^c3,
c_4=c4;
endmodule
```

Test bench:

```
module carry_lookahead_adder_tb ();
parameter WIDTH = 3;
reg [WIDTH-1:0] r_ADD_1 = 0;
reg [WIDTH-1:0] r_ADD_2 = 0;
wire [WIDTH:0]  w_RESULT;
CLA_4bit #(.WIDTH(WIDTH)) carry_lookahead_inst
( .i_add1(r_ADD_1), .i_add2(r_ADD_2),
.o_result(w_RESULT)  );

initial
begin
#10;
r_ADD_1 = 3'b000;
r_ADD_2 = 3'b001;
#10;
r_ADD_1 = 3'b010;
r_ADD_2 = 3'b010;
#10;
r_ADD_1 = 3'b101;
r_ADD_2 = 3'b110;
#10;
r_ADD_1 = 3'b111;
_ADD_2 = 3'b111;
#10;
end
endmodule // carry_lookahead_adder_t
```

Simulation Waveform:



Simulation waveforms of carry look ahead adder

4.5 Post Lab:

1. Compare the area, delay and power report of ripple carry & carry look-ahead adder in Xilinx ISE. Create a comparison chart and justify the results.
2. List the application of CLA in VLSI Design.
3. Prepare the synthesis Chart for a 4-bit CLA.
4. Can retiming mechanism improve the speed further in CLA architecture?

4.6 Result

Thus, the design of 4-bit carry look-ahead adder circuit was simulated in Verilog and synthesized using EDA tools.

Lab Experiment # 5
Realization of Carry Skip Adder

5.1 Objective: To design and simulate the carry skip adder (4-bit) in Verilog and synthesize using EDA tools.

5.2 Software tools Requirement

Equipment's:

Computer with Xilinx and Modelsim Software Specifications:
HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk
Software's: Synthesis tool: Xilinx ISE
Simulation tool: Modelsim Simulator

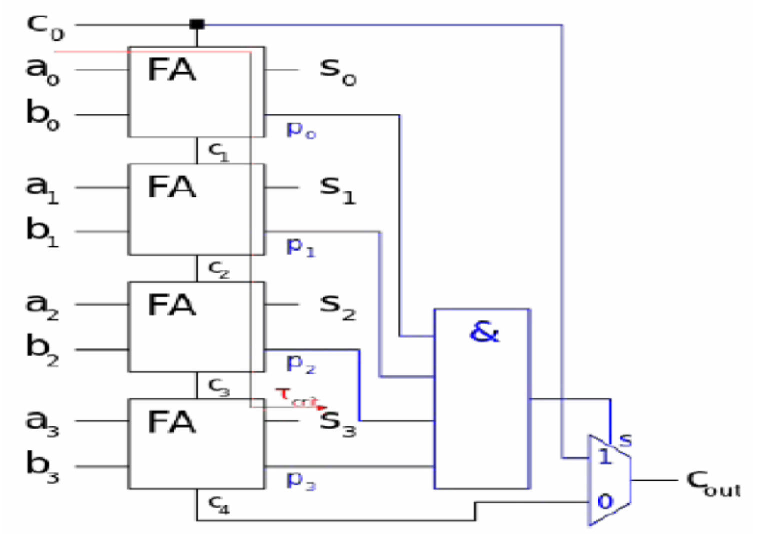
5.3 Pre Lab Questions

(Write pre lab Q & A in an A4 sheet)

1. Why Carry-Skip adder has the same critical path as regular Carry-Ripple adder.
2. List the efficient method for implementing 16 Bit - adder using Carry-Skip technique?

5.4 Problem: Write a Verilog code to implement the 4-bit Carry Look-ahead adder using structural model.

Logic Diagram:



4-bit Carry Look-ahead Adder

Source code:

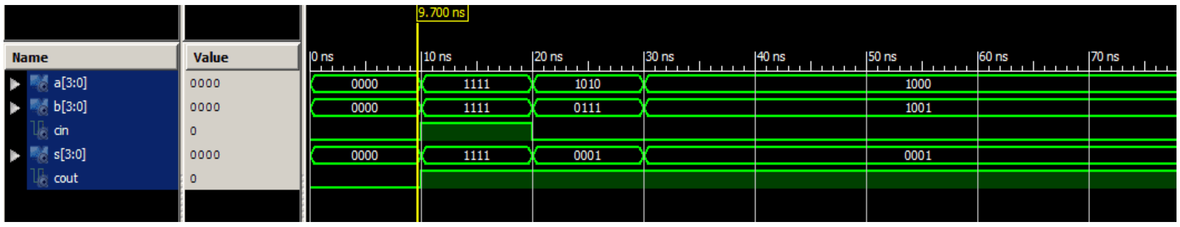
```
module carry_skip_4bit(a, b, cin, sum, cout);
input [3:0] a,b;
input cin;
output [3:0] sum;
output cout;
wire [3:0] p;
wire c0;
wire bp;
ripple_carry_4_bit rca1 (a[3:0],b[3:0],cin, sum[3:0],c0);
propagate_p p1(a, b, p, bp);
mux2X1 m0(c0,cin,bp,cout);
endmodule

module propagate_p(a,b,p,bp);
input [3:0] a,b;
output [3:0] p;
output bp;
assign p= a^b;//get all propagate bits
assign bp= &p;// and p0p1p2p3 bits
endmodule

module ripple_carry_4_bit(a, b, cin,sum, cout);
input [3:0] a,b;
input cin;
wire c1,c2,c3;
output [3:0] sum;
output cout;
full_adder fa0(a[0], b[0],cin,
sum[0],c1);
full_adder fa1(a[1], b[1],c1,
sum[1],c2);
full_adder fa2(a[2], b[2],c2,
sum[2],c3);
full_adder fa3(a[3], b[3],c3,
sum[3],cout);
endmodule
```

TEST BENCH:

```
module carry_skip_4bit_addertb;
wire [7:0]Y;
wire carryout;
reg [7:0]A,B;
reg carryin;
carry_skip_4bit csa1 (Y,carryout,A,B,carryin);
initial
begin
```

```
$display("RSLT\tA\tB\tCYIN\t\tCYOUT\tSUM");
//A = 4'b0101; B = 4'b1101; carryin = 0; #50; // Set inputs and add delay
A = 0; B = 0; carryin = 0; #50; // Set inputs and add delay
A = 3; B = 2; carryin = 1; #50; // Set inputs and add delay
A = 7; B = 10; carryin = 0; #50; // Set inputs and add delay
A = 15; B = 15; carryin = 1; #50; // Set inputs and add delay
A = 255; B = 55; carryin = 0; #50; // Set inputs and add delay
A = 255; B = 255; carryin = 1; #50; // Set inputs and add delay
/*
//if ( (carryout == 1 ) && (Y === 4'b0010) )
if ( (carryout == 1 ) && (Y === 2) )
$display("PASS\t%p\t%p\t%d\t=\t%d\p%p",A,B,carryin,carryout,Y);
else
$display("FAIL\t%p\t%p\t%d\t=\t%d\t%p",A,B,carryin,carryout,Y);
*/
end
//enabling the wave dump
initial begin
$dumpfile("dump.vcd"); $dumpvars;
end
endmodule
```

Simulation Waveform:



Simulation waveform of carry skip adder

5.5 Post Lab:

1. Write a Verilog code to design 4 Bit Carry Save Adder.

5.6 Result

Thus, the design of 4-bit carry look-ahead adder circuit was simulated in Verilog and synthesized using EDA tools.

Lab Experiment #6
Realization of Multiplier-1

6.1 Objective: To design and simulate the Braun array multiplier in Verilog and synthesize using EDA tools.

6.2 Software tools Requirement:

Equipment's:

Computer with Xilinx and Modelsim Software Specifications:

HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk

Software's: Synthesis tool: Xilinx ISE

Simulation tool: Modelsim Simulator

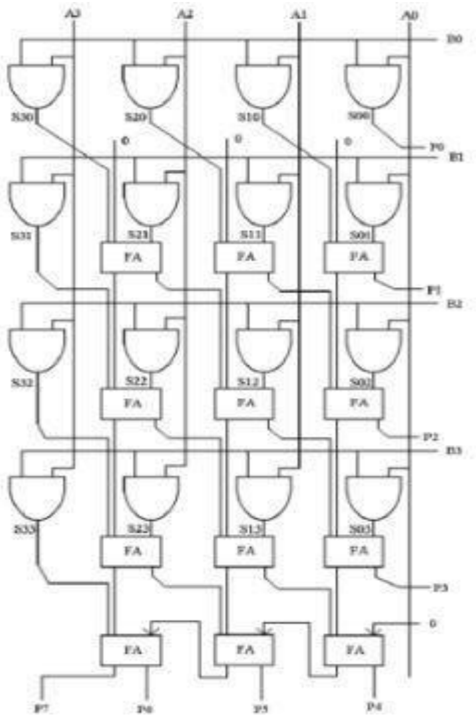
6.3 Prelab Questions:

(write pre lab Q & A in an A4 sheet)

1. What is called Booth recoding?
2. Give the booths recoding transformation of the following number 01111001(0).
3. Give the difference between Booth's recoding and modified booth's recoding?

6.4 Problem: Write a Verilog code to implement the 4-bit Braun Array multipliers using structural model.

Logic Diagram



4-bit Braun Array multiplier

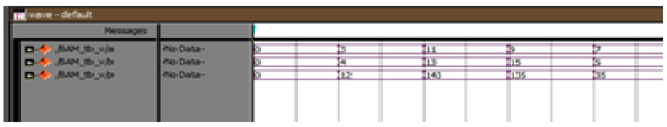
Verilog Code

```
module braun(a, b, p); input [3:0] a;
    input [3:0] b;
    output [7:0] p;
wire w0,w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11,w12,w13,w14,w15,w16,w17,
w18, w19,w20,w21,w22,w23,w24,w25,w26,w27,w28,w29;

 and g0 (w0,  a[3], b[0]);
 and g1 (w1,  a[2], b[0]);
 and g2 (w2,  a[1], b[0]);
 and g3 (p[0],a[0], b[0]);
 and g4 (w6,  a[3], b[1]);
 and g5 (w3,  a[2], b[1]);
 and g6 (w4,  a[1], b[1]);
 and g7 (w5,  a[0], b[1]);
 and g8 (w12, a[3], b[2]);
 and g9 (w13, a[2], b[2]);
 and g10(w14, a[1], b[2]);
 and g11(w15, a[0], b[2]);
 and g12(w21, a[3], b[3]);
 and g13(w22, a[2], b[3]);
 and g14(w23, a[1], b[3]);
 and g15(w24, a[0], b[3]);

 fa_df f0 (w8 , w7 , w0 , w3 , 1'b0);
 fa_df f1 (w10 , w9 , w1 , w4 , 1'b0);
 fa_df f2 (p[1], w11 , w2 , w5 , 1'b0);
 fa_df f3 (w17 , w16 , w6 , w13, w7);
 fa_df f4 (w19 , w18 , w8 , w14, w9);
 fa_df f5 (p[2], w20 , w10, w15, w11);
 fa_df f6 (w26 , w25 , w12, w22, w16);
 fa_df f7 (w28 , w27 , w17, w23, w18);
 fa_df f8 (p[3], w29 , w19, w24, w20);
 fa_df f9 (p[6], p[7], w21, w25, w31);
 fa_df f10(p[5], w31 , w26, w27, w30);
 fa_df f11(p[4], w30 , w28, w29, 1'b0);
 endmodule
```

// TEST BENCH

```
 module braun_test_v;
 // Inputs
 reg [3:0] a;
 reg [3:0] b;
 // Outputs
 wire [7:0]p;
// Instantiate the Unit Under Test (UUT) braun uut (
            .a(a),
            .b(b),
            .p(p));
 initial
 begin
 // Initialize Inputs
            a = 0;
            b = 0;
            #100;a=1101; b=1001;
            end
            endmodule
```

Simulation waveforms:



Braun Array multiplier waveform

6.5 Post Lab:

1. Write the booth multiplier code in Verilog and implement using EDA tools.

6.6 Result

Thus, the design of 4-bit Braun array multiplier circuit was simulated in Verilog and synthesized using EDA tools.

Lab Experiment #7
Realization of Multiplier-II

7.1 Objective: To design and simulate the Wallace tree multiplier in Verilog and synthesize using EDA tools

7.2 Software tools Requirement

Equipment's:

Computer with Xilinx and Modelsim Software Specifications:

HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk

Software's: Synthesis tool: Xilinx ISE

Simulation tool: Modelsim Simulator

7.3 Pre Lab Questions

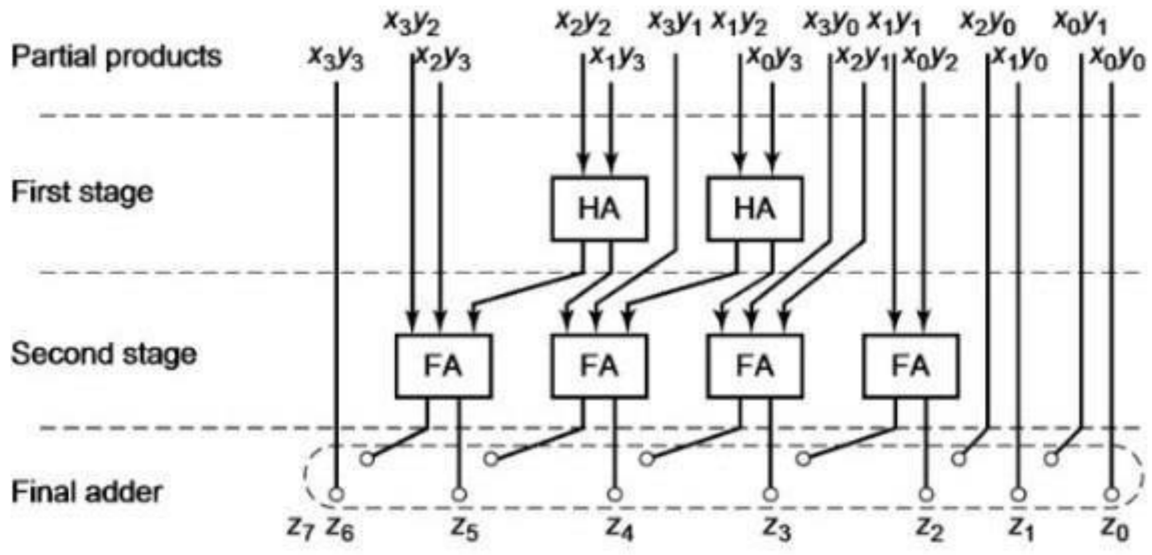
(Write pre lab Q & A in an A4 sheet)

1.How carry save multiplier differs from array multiplier?
2.Why the partial sum adders are arranged in tree like fashion?

7.4 Problem:

Write a Verilog code to implement the 4-bit Wallace tree multipliers using structural model.

Logic Diagram:



7.1 4-bit Wallace tree multiplier

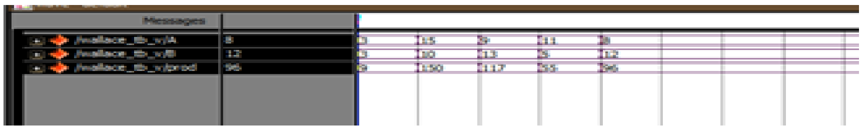
Source code:

```
module wallace_tree_mul(a, b, z); input [3:0] a;
input  [3:0]  b;
output [7:0]  z;
wire   [32:0] w;
 and g1 (z[0] , a[0], b[0]);
 and g2 (w[27], a[0], b[1]);
 and g3 (w[0] , a[1], b[0]);
 and g4 (w[1] , a[2], b[0]);
 and g5 (w[2] , a[0], b[2]);
 and g6 (w[3] , a[1], b[1]);
 and g7 (w[4] , a[1], b[1]);
 and g8 (w[5] , a[3], b[0]);
 and g9 (w[6] , a[0], b[3]);
 and g10(w[7] , a[1], b[2]);
 and g11(w[8] , a[3], b[1]);
 and g12(w[9] , a[1], b[3]);
 and g13(w[10], a[2], b[2]);
 and g14(w[11], a[2], b[3]);
 and g15(w[12], a[3], b[2]);
 and g16(w[13], a[3], b[3]);
 ha_gate h0(w[16], w[17], w[10], w[9]);
 ha_gate h1(w[14], w[15],  w[7], w[6]);
 ha_gate h2(z[1] , w[28], w[27], w[0]);
fa_df f0(w[19] , w[20] , w[2] , w[3] , 1'b0);
fa_df f1(w[21]  , w[22]  , w[4]  , w[5]  , w[14]);
fa_df f2(w[23]  , w[24]  , w[15] , w[8]  , w[15]);
fa_df f3(w[25]  , w[26]  , w[17] , w[11] , w[12]);
fa_df f4(z[2]   , w[29]  , w[0]  , w[1]  , w[28]);
fa_df f5(z[3]   , w[30]  , w[20] , w[21] , w[29]);
fa_df f6(z[4]   , w[31]  , w[22] , w[23] , w[30]);
fa_df f7(z[5]   , w[32]  , w[24] , w[25] , w[31]);
fa_df f8(z[6] , z[7] , w[26] , w[13] , w[32]);
endmodule
```

Test bench:

```
module wallace_test_v;
// Inputs reg [3:0] a;
reg [3:0] b;
// Outputs wire [7:0] z;
// Instantiate the Unit Under Test (UUT) wallace_tree_mul uut (
.a(a),
.b(b),
.z(z));
initial begin
// Initialize Inputs a = 0;
b = 0;
//wait 100 ns for global reset to finish
#100a=1101; b=1001;
end
endmodule
```

Simulation Waveform:



```
Messages
/wallace_tb_v/A     8    0   15   9    11  8
/wallace_tb_v/B     12   0   10   13   5   12
/wallace_tb_v/prod  96   0   150  117  55  96
```

Simulation waveform of Wallace tree multiplier

7.5 Post Lab:

1. Write the Verilog code for 4-bit Baugh Woolley multiplier in structural modelling and implement using EDA tools.

7.6 Result

Thus, the design of 4-bit Wallace tree multiplier circuit was simulated in Verilog and synthesized using EDA tools

Lab Experiment #8
Realization of Memory

8.1 Objective:

Design memory using Verilog and Simulate using Xilinx Tool.

8.2 Software tools Requirement:

Equipment's:

Computer with Xilinx and Modelsim Software Specifications.
HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk.

Software's:

Synthesis tool: Xilinx ISE.
Simulation tool: Modelsim Simulator.

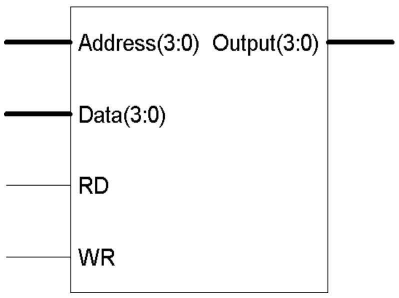
8.3 Prelab:

1. What are the different types of memory? Compare its performance.
2. Write the difference between static and dynamic memory.
3. Define port for 32*64k RAM memory.

8.4.1 Problem Statement 1:

1. Design a 16X4 RAM Memory.

Logic Diagram:



Program:

```
module Memory_Design( Data, RD, WR, Address, Output);input[3:0] Data;
input RD, WR; input[3:0] Address; output reg[3:0] Output;reg[3:0] RAM[15:0];
 reg[3:0] Temp;
 always@(Address)begin
 if(WR)
 begin
 RAM[Address]=Data; Temp=RAM[Address];
 end
 if(RD)
```

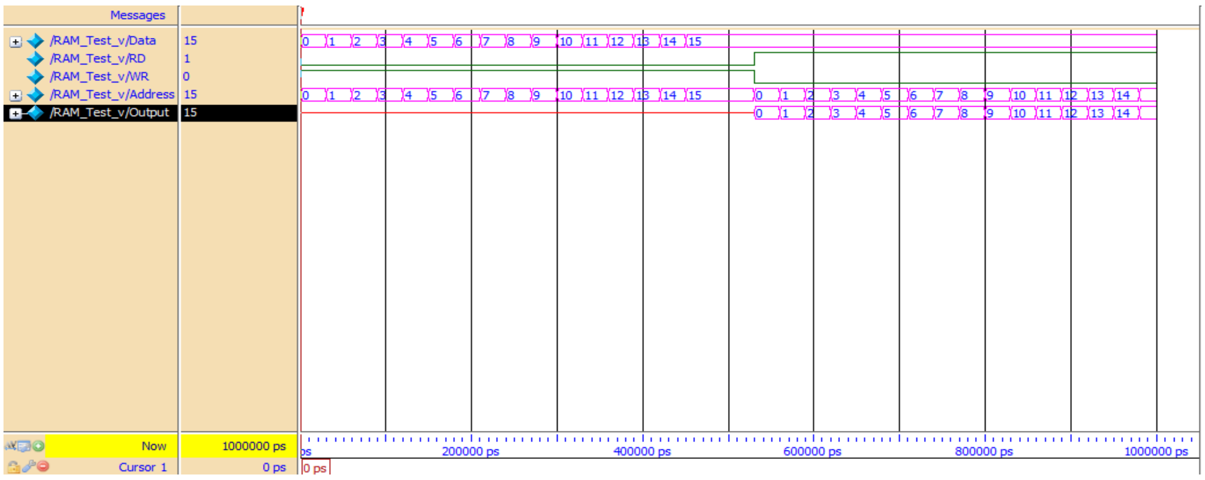
```
begin
Output=RAM[Address];
end
end
endmodule
```

TEST BENCH:

```
module RAM_Test_v;
// Inputs
reg [3:0] Data;reg RD;
reg WR;
reg [3:0] Address;
// Outputs
wire [3:0] Output;
// Instantiate the Unit Under Test (UUT)
emory_Design uut
(.Data(Data),.RD(RD),.WR(WR),.Address(Address),.Output(Output));
initial begin
            Data = 4'b0000;          RD = 0;        WR = 1;        Address = 4'b0000; #30;

            Data = 4'b0001;          RD = 0;        WR = 1;        Address = 4'b0001; #30;

            Data = 4'b0010;          RD = 0;        WR = 1;        Address = 4'b0010; #30;

            Data = 4'b0011;          RD = 0;        WR = 1;        Address = 4'b0011; #30;

            Data = 4'b0100;          RD = 0;        WR = 1;        Address = 4'b0100; #30;

            Data = 4'b0101;          RD = 0;        WR = 1;        Address = 4'b0101; #30;

            Data = 4'b0110;          RD = 0;        WR = 1;       Address = 4'b0110; #30;

            Data = 4'b0111;          RD = 0;        WR = 1;       Address = 4'b0111; #30;

            Data = 4'b1000;          RD = 0;        WR = 1;       Address = 4'b1000; #30;

            Data = 4'b1001;          RD = 0;        WR = 1;       Address = 4'b1001; #30;

            Data = 4'b1010;          RD = 0;        WR = 1;       Address = 4'b1010; #30;

            Data = 4'b1011;          RD = 0;        WR = 1;       Address = 4'b1011; #30;

            Data = 4'b1100;          RD = 0;        WR = 1;       Address = 4'b1100; #30;

            Data = 4'b1101;          RD = 0;        WR = 1;       Address = 4'b1101; #30;

            Data = 4'b1110;          RD = 0;        WR = 1;       Address = 4'b1110; #30;
              Data = 4'b1111;        RD = 0;        WR = 1;Address = 4'b1111; #30;

                   RD = 1;             WR = 0;                  Address = 4'b0000; #30;

                   RD = 1;            WR = 0;                   Address = 4'b0001; #30;
```

```
                RD = 1;         WR = 0;             Address = 4'b0010; #30;
                RD = 1;         WR = 0;             Address = 4'b0011; #30;
                RD = 1;         WR = 0;             Address = 4'b0100; #30;
                RD = 1;         WR = 0;             Address = 4'b0101; #30;
                RD = 1;         WR = 0;             Address = 4'b0110; #30;
                RD = 1;         WR = 0;             Address = 4'b0111; #30;
                RD = 1;         WR = 0;             Address = 4'b1000; #30;
                RD = 1;         WR = 0;             Address = 4'b1001; #30;
                RD = 1;         WR = 0;             Address = 4'b1010; #30;
                RD = 1;         WR = 0;             Address = 4'b1011; #30;
                RD = 1;         WR = 0;             Address = 4'b1100; #30;
                RD = 1;         WR = 0;             Address = 4'b1101; #30;
                RD = 1;         WR = 0;             Address = 4'b1110; #30;
                RD = 1;         WR = 0;             Address = 4'b1111; #30;
                #100;
    End
      endmodule
```

Simulation Output:



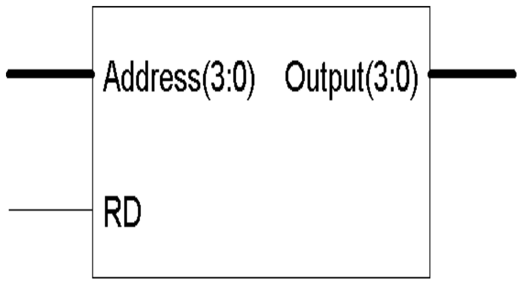
```
Messages
/RAM_Test_v/Data      15
/RAM_Test_v/RD        1
/RAM_Test_v/WR        0
/RAM_Test_v/Address   15
/RAM_Test_v/Output    15
Now                   1000000 ps
Cursor 1              0 ps
```

Statement 2:

8.4.2 Problem Design 16X4 ROM Memory.

To Design a 16x4 ROM in Verilog and Simulate using Xilinx Tool.

Block Diagram:



Program:

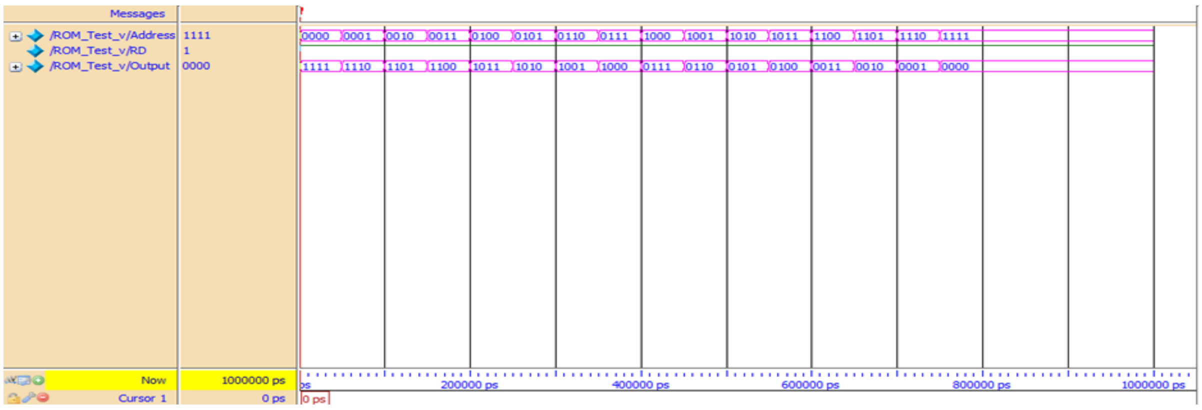
```
Module ROM_16x4(Address, RD, Output);input[3:0] Address;
input RD;
output reg[3:0] Output;reg[3:0] ROM[15:0];
initial
 begin
 ROM[4'b0000]=4'b1111;ROM[4'b0001]=4'b1110;
ROM[4'b0010]=4'b1101;          ROM[4'b0011]=4'b1100;          ROM[4'b0100]=4'b1011;
 ROM[4'b0101]=4'b1010;          ROM[4'b0110]=4'b1001;          ROM[4'b0111]=4'b1000;
 ROM[4'b1000]=4'b0111;          ROM[4'b1001]=4'b0110;          ROM[4'b1010]=4'b0101;
 ROM[4'b1011]=4'b0100;          ROM[4'b1100]=4'b0011;          ROM[4'b1101]=4'b0010;
 ROM[4'b1110]=4'b0001; ROM[4'b1111]=4'b0000;
 end
always@(RD , Address)begin
if(RD)
begin
  Output=ROM[Address];
end
end
   endmodule
```

Test Bench:

```
   module ROM_Test_v;
          // Inputs
          reg [3:0] Address;reg RD;
          // Outputs
          wire [3:0] Output;
          // Instantiate the Unit Under Test (UUT)
          ROM_16x4 uut (.Address(Address),.RD(RD),.Output(Output));initial begin
                 // Initialize Inputs
```

```
                Address = 4'b0000;                    RD = 1;      #50;
                Address = 4'b0001;                    RD = 1;      #50;
                Address = 4'b0010;                    RD = 1;      #50;
                Address = 4'b0011;                    RD = 1;      #50;
                Address = 4'b0100;                    RD = 1;      #50;
                Address = 4'b0101;                    RD = 1;      #50;
                Address = 4'b0110;                    RD = 1;      #50;
                 Address = 4'b0111;                   RD = 1;      #50;
                 Address = 4'b1000;                   RD = 1;      #50;
                   Address = 4'b1001;                              #50;
                   Address = 4'b1010;                              #50;
                   Address = 4'b1011;                              #50;
                   Address = 4'b1100;                              #50;
                   Address = 4'b1101;                              #50;
                   Address = 4'b1110;                              #50;
                   Address = 4'b1111;                              #50;
            end
        endmodule
```

Simulation:



```
Messages
/ROM_Test_v/Address  1111   0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
/ROM_Test_v/RD       1
/ROM_Test_v/Output   0000   1111 1110 1101 1100 1011 1010 1001 1000 0111 0110 0101 0100 0011 0010 0001 0000
Now      1000000 ps    ps   200000 ps   400000 ps   600000 ps   800000 ps   1000000 ps
Cursor 1 0 ps          0 ps
```

8.5 Post Lab:

Draw the block diagram and explain about FIFO memory.

8.6 Result:

Design of a 16x4 ROM in Verilog is Performed and Verified using Xilinx Tool.

Lab Experiment #9
Design and Analysis of CMOS inverter using LTSPICE

9.1 Objective: To learn the design of CMOS inverter in circuit level and get the spice netlist using LTSPICE tool.

9.2 Software tools Requirement

Equipment's:

LTSPICE software

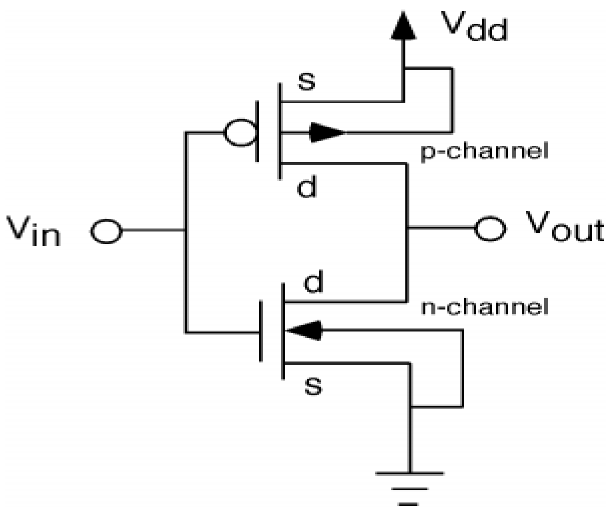
9.3 Prelab Questions

(Write pre lab Q & A in an A4 sheet)

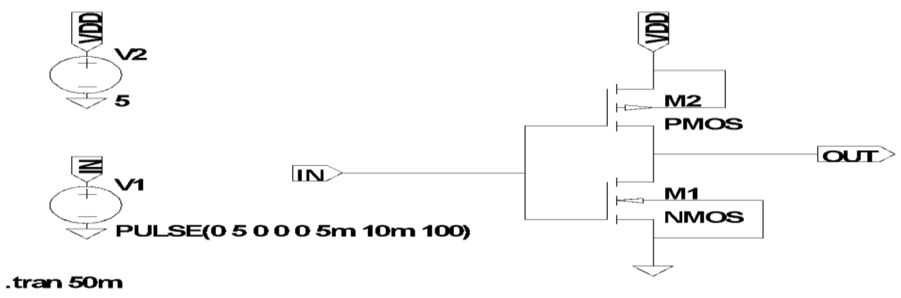
1. What are the advantages of SPICE software?
2. What are the differences between enhancement and depletion mode transistors?
3. What is the difference between strong 1 and weak 1?
4. What is meant by transistor sizing?

9.4 Problem 1: Design and analyse the CMOS inverter in circuit level, verify the transfer characteristics and infer the SPICE netlist using LTSPICE.

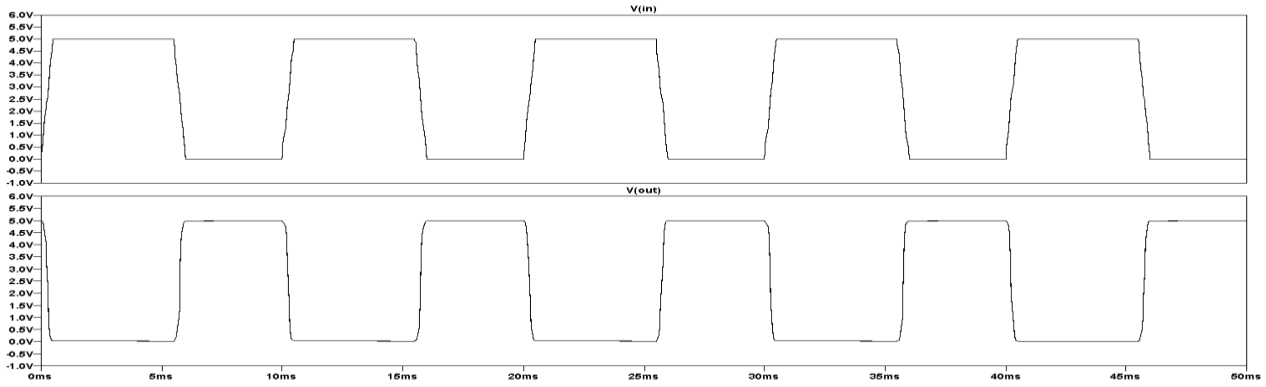
9.4.1 Logic Diagram



9.4.2 Circuit design of CMOS Inverter in LTSPICE:



9.4.3 Output of CMOS INVERTER:



9.4.4 SPICE NETLIST:

```
* C:\Users\PRITHIVI\Documents\LTspiceXVII\INVERTER.asc
V1 IN 0 PULSE(0 5 0 0 0 5m 10m 100)
V2 VDD 0 5
M1 OUT IN 0 0 NMOS
M2 VDD IN OUT VDD PMOS
.model NMOS NMOS
.model PMOS PMOS
.lib C:\Users\PRITHIVI\Documents\LTspiceXVII\lib\cmp\standard.mos
.tran 50m
.backanno
.end
```

9.5 Post lab

1. Perform DC Analysis for CMOS Inverter in LTSPICE.

9.6 Result:

Thus, the design and analysis of CMOS Inverter has been performed, transfer characteristics have been verified using LTSPICE tool.

Lab Experiment #10

Design and Analysis of Complex CMOS gates and Pseudo NMOS gates using LTSPICE

10.1 Objective: To learn the design of Complex CMOS gates and Pseudo NMOS gates in circuit level and get the spice netlist using LTSPICE tool.

10.2 Software tools Requirement

Equipment's:

LTSPICE software

10.3 Prelab Questions

(Write pre lab Q & A in an A4 sheet)

1. What is meant by Pseudo NMOS logic?
2. What are the advantages and drawbacks of Pseudo NMOS?
3. In Pseudo NMOS logic, PMOS transistor operates in __________.
4. What is meant by body effect?
5. What are the different types of operating region in MOSFET?

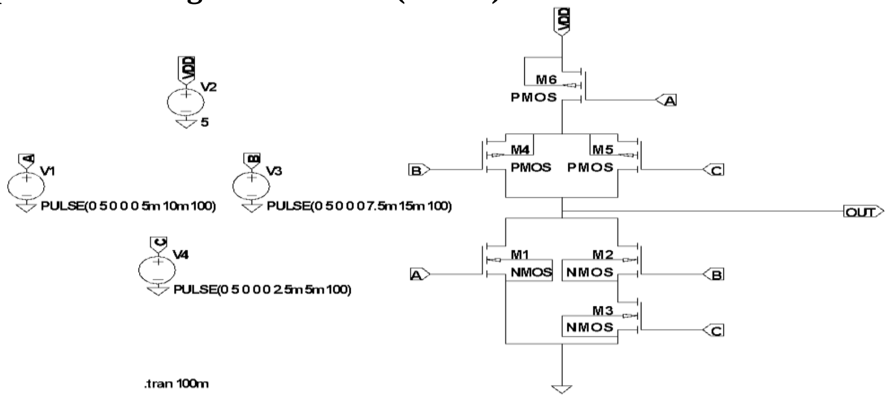
10.4 Problem 1:

Design and analyse the Complex CMOS gate in circuit level, verify the transfer characteristics and infer the SPICE netlist using LTSPICE.

10.4.1 Problems:

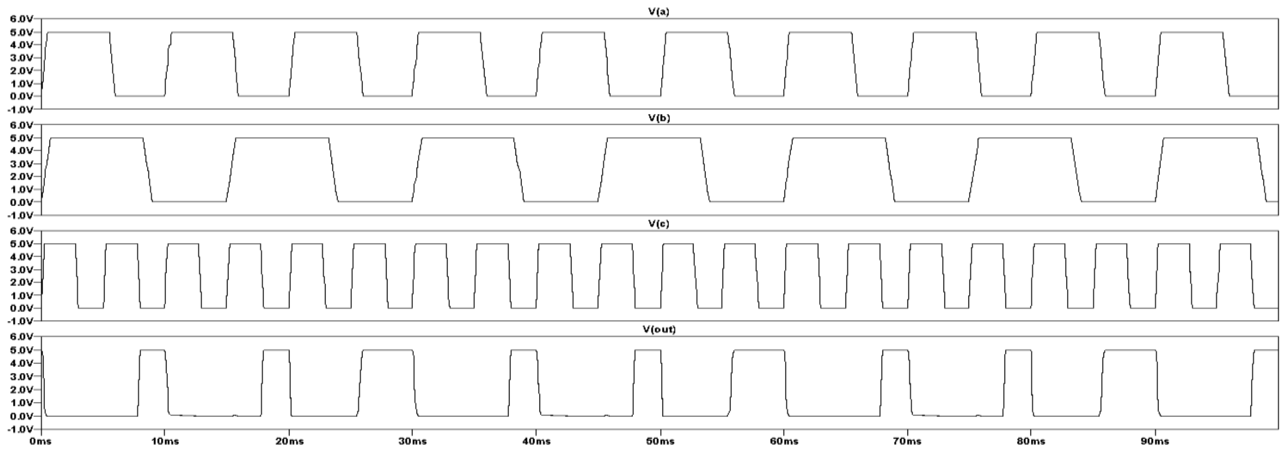
1. Design complex CMOS logic for OUT= ~(A+BC).
2. Design and analysis of Pseudo NMOS gate Inverter using LTSPICE.

10.4.2 Complex CMOS logic for OUT= ~(A+BC)



VDD
V2
5
A
V1
PULSE(0 5 0 0 0 5m 10m 100)
B
V3
PULSE(0 5 0 0 0 7.5m 15m 100)
C
V4
PULSE(0 5 0 0 0 2.5m 5m 100)
.tran 100m
M6
PMOS
M4
PMOS
M5
PMOS
M1
NMOS
M2
NMOS
M3
NMOS
OUT

10.4.3 Output of complex CMOS logic for OUT= ~(A+BC)

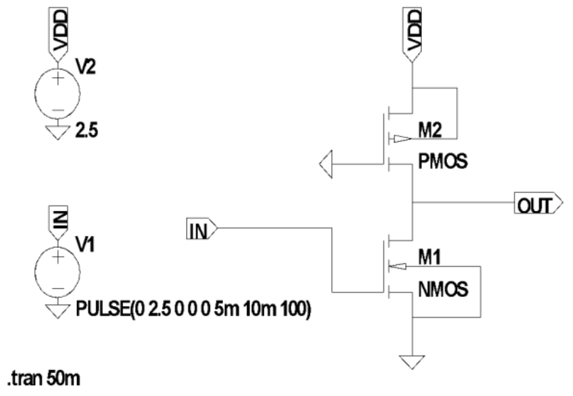


10.4.4 SPICE NETLIST:

```
* C:\Users\PRITHIVI\Documents\LTspiceXVII\A+BC_COMPLEXGATE.asc
M1 OUT A 0 0 NMOS
M2 OUT B N002 N002 NMOS
M3 N002 C 0 0 NMOS
M4 N001 B OUT N001 PMOS
M5 N001 C OUT N001 PMOS
M6 VDD A N001 VDD PMOS
V1 A 0 PULSE(0 5 0 0 0 5m 10m 100)
V2 VDD 0 5
V3 B 0 PULSE(0 5 0 0 0 7.5m 15m 100)
V4 C 0 PULSE(0 5 0 0 0 2.5m 5m 100)
.model NMOS NMOS
.model PMOS PMOS
.lib C:\Users\PRITHIVI\Documents\LTspiceXVII\lib\cmp\standard.mos
.tran 100m
.backanno
.end
```

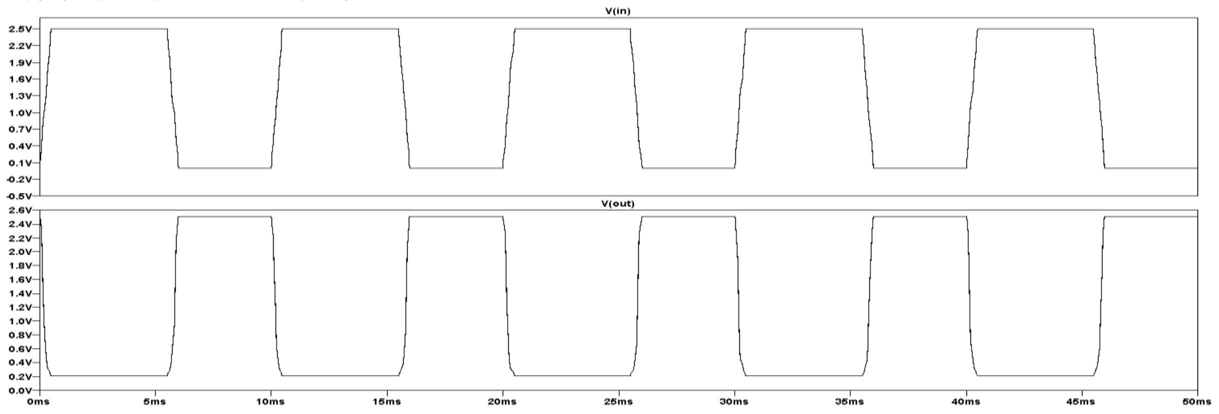
10.4.5 Pseudo NMOS gate Inverter:

W/L ratio of PMOS: 4/2, NMOS:24/2.



10.4.6 Pseudo NMOS Inverter output:

10.4.7 SPICE NETLIST :



```
* C:\Users\PRITHIVI\Documents\LTspiceXVII\PSEUDO NMOS INVERTER.asc
V1 IN 0 PULSE(0 2.5 0 0 0 5m 10m 100)
V2 VDD 0 2.5
M1 OUT IN 0 0 NMOS l=2 w=24
M2 VDD 0 OUT VDD PMOS l=2 w=4
.model NMOS NMOS
.model PMOS PMOS
.lib C:\Users\PRITHIVI\Documents\LTspiceXVII\lib\cmp\standard.mos
.tran 50m
.backanno
.end
```

10.5 Post lab:

1. Design Complex CMOS logic Out= ~(AB+CD) in LTSPICE.

2. Design Pseudo NMOS NAND gate in LTSPICE.

10.6 Result:

Thus, the design and analyse of Complex CMOS gate and Pseudo NMOS gate have been performed and its transfer characteristics is verified using LTSPICE tool.

Lab Experiment # 11

Design and Analysis of DCVSL, CPL and Pass Transistor Logic using LTSPICE

11.1 Objective: To learn the design of **DCVSL, CPL and Pass Transistor Logic** in circuit level and get the spice net list using LTSPICE tool.

11.2 Software tools Requirement
Equipment's:
LTSPICE software

11.3 Pre lab Questions
(Write pre lab Q & A in an A4 sheet)

1. Differentiate Dynamic and Domino CMOS logic
2. Explain Domino CMOS logic with an example.

11.4 Problem 1:

1a) Design and analyze Differential Cascode Voltage Switch Logic(DCVSL) using LTspice

1b) Design and analyze the complementary Pass transistor Logic for a 2 input AND/NAND in circuit level, verify the characteristics and infer the SPICE netlist using LTSPICE.

Problem 2:

2a) Design and analyze the complementary Pass transistor Logic for a 2 input OR/NOR in circuit level, verify the characteristics and infer the SPICE netlist using LTSPICE.

Problem 3:

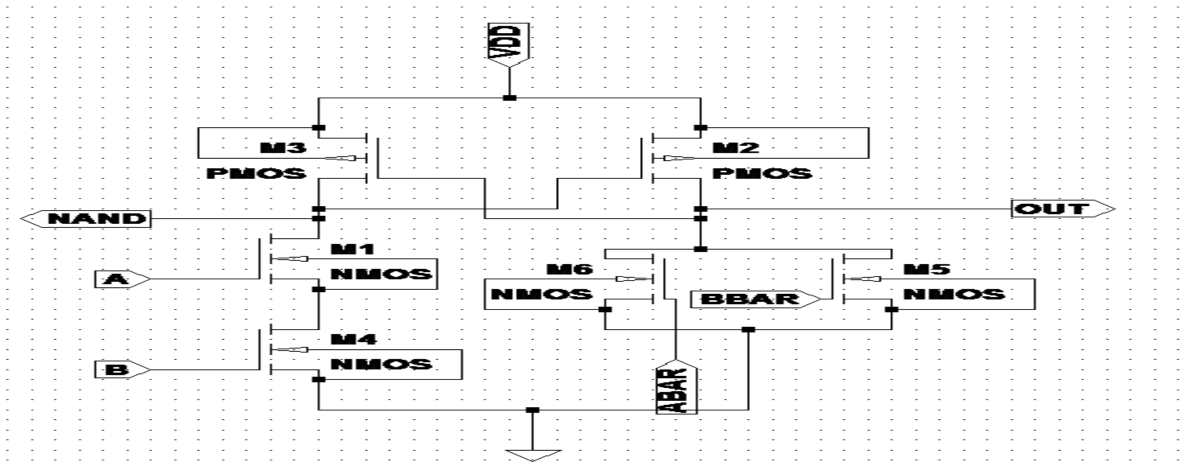
3a) Design and analyze the Pass transistor Logic for a 2 input AND in circuit level, verify the characteristics and infer the SPICE netlist using LTSPICE.

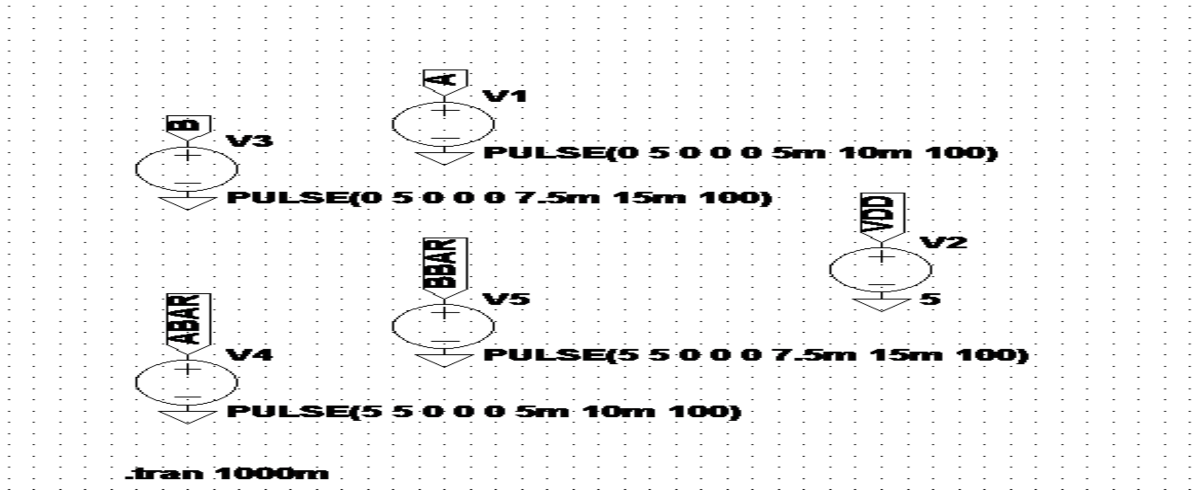
Problem 4:

4a) Design and analyze the Pass transistor Logic for a 2 input XOR in circuit level, verify the characteristics and infer the SPICE netlist using LTSPICE.

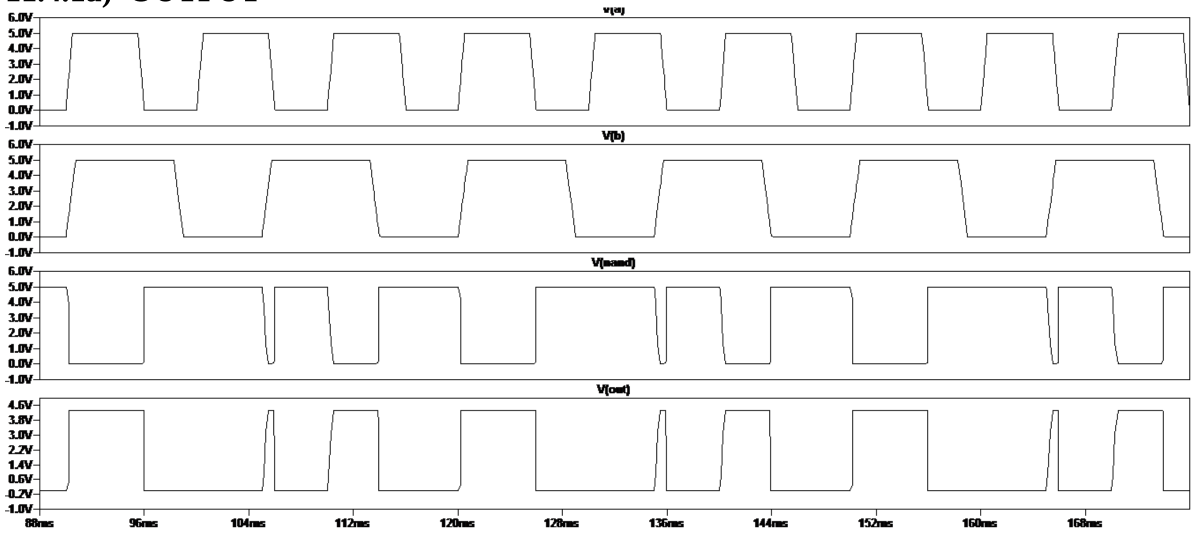
11.4.1 Circuit Diagram

a.)DCVSL –NAND gate




A
V1
PULSE(0 5 0 0 0 5m 10m 100)
B
V3
PULSE(0 5 0 0 0 7.5m 15m 100)
VDD
V2
5
BBAR
V5
PULSE(5 5 0 0 0 7.5m 15m 100)
ABAR
V4
PULSE(5 5 0 0 0 5m 10m 100)
.tran 1000m

11.4.1a) OUTPUT

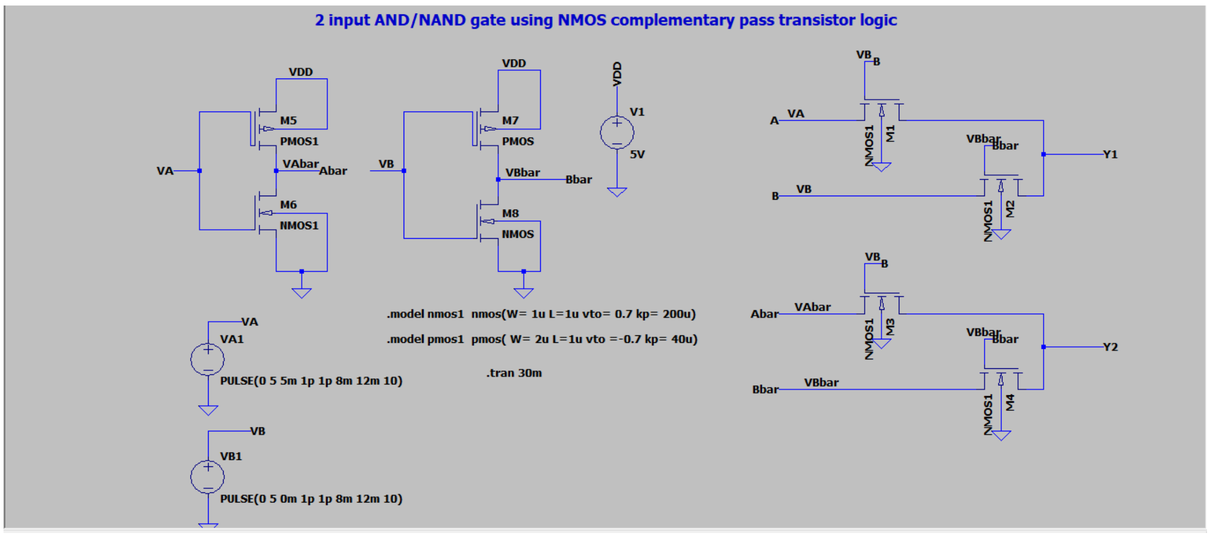


11.4.1a)Spice Netlist:

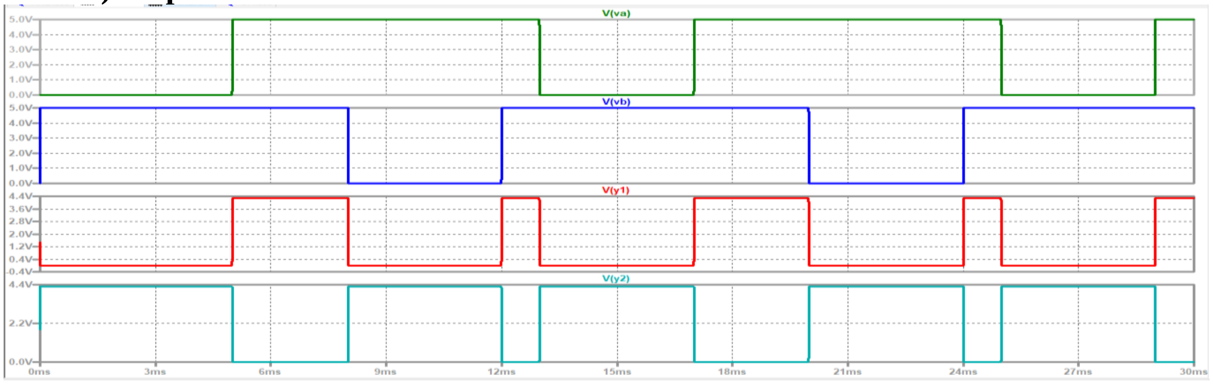
* C:\Users\PRITHIVI\Documents\LTspiceXVII\DCVSL.asc
M2 VDD NAND OUT VDD PMOS l=2 w=80

```
M3 VDD OUT NAND VDD PMOS l=2 w=4
M1 NAND A N001 N001 NMOS l=2 w=20
M4 N001 B 0 0 NMOS l=2 w=20
M5 OUT BBAR 0 0 NMOS l=2 w=10
M6 OUT ABAR 0 0 NMOS l=2 w=10
V1 A 0 PULSE(0 5 0 0 0 5m 10m 100)
V2 VDD 0 5
V3 B 0 PULSE(0 5 0 0 0 7.5m 15m 100)
V4 ABAR 0 PULSE(5 5 0 0 0 5m 10m 100)
V5 BBAR 0 PULSE(5 5 0 0 0 7.5m 15m 100)
.model NMOS NMOS
.model PMOS PMOS
.lib C:\Users\PRITHIVI\Documents\LTspiceXVII\lib\cmp\standard.mos
.tran 1000m
.backanno
.end
```

1b) AND/NAND gate using CPL:


2 input AND/NAND gate using NMOS complementary pass transistor logic
VDD
M5
PMOS1
VA
VAbar
Abar
M6
NMOS1
VDD
M7
PMOS
VB
VBbar
Bbar
M8
NMOS
VDD
V1
5V
VB
B
A
VA
NMOS1
M1
VBbar
Bbar
Y1
B
VB
NMOS1
M2
VB
B
Abar
VAbar
NMOS1
M3
VBbar
Bbar
Y2
Bbar
VBbar
NMOS1
M4
VA
VA1
PULSE(0 5 5m 1p 1p 8m 12m 10)
VB
VB1
PULSE(0 5 0m 1p 1p 8m 12m 10)
.model nmos1 nmos(W= 1u L=1u vto= 0.7 kp= 200u)
.model pmos1 pmos(W= 2u L=1u vto =-0.7 kp= 40u)
.tran 30m

11.4.1b)Output

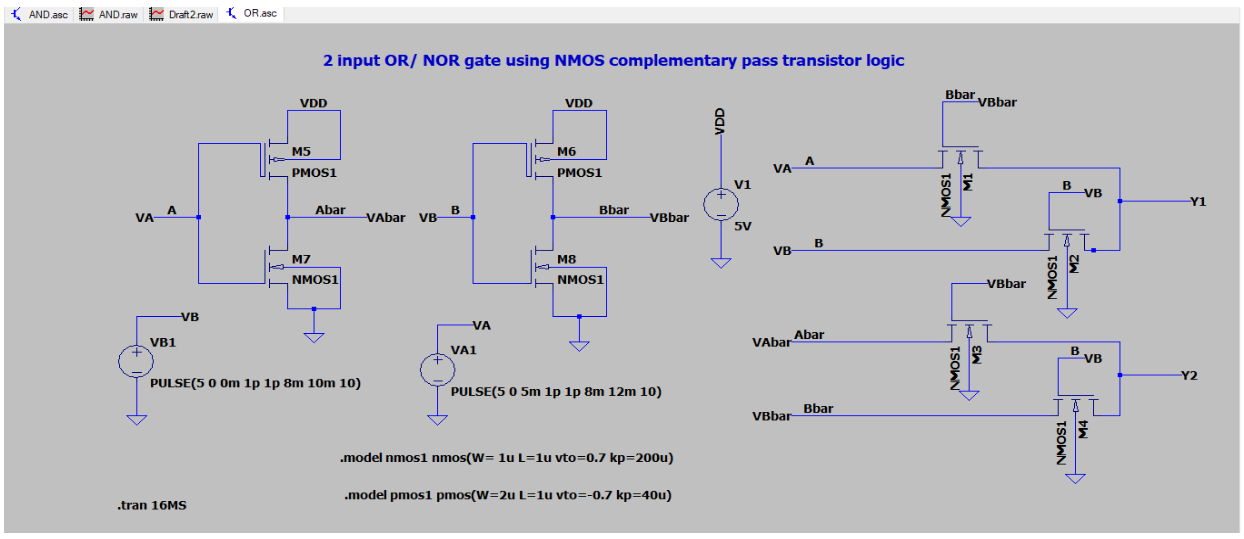


SPICE NETLIST:

- * F:\AND.asc

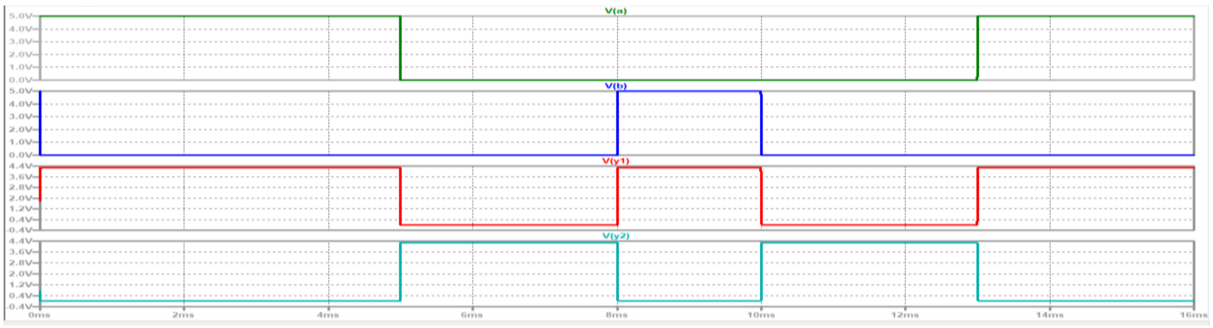
- M1 Y1 VB VA 0 NMOS1 l=1u w=1u
- M2 Y1 VBbar VB 0 NMOS1 l=1u w=1u
- M3 Y2 VB VAbar 0 NMOS1 l=1u w=1u
- M4 Y2 VBbar VBbar 0 NMOS1 l=1u w=1u
- M5 VDD VA VAbar VDD PMOS1
- M6 VAbar VA 0 0 NMOS1 l=1u w=1u
- V1 VDD 0 5V
- VA1 VA 0 PULSE(0 5 5m 1p 1p 8m 12m 10)
- VB1 VB 0 PULSE(0 5 0m 1p 1p 8m 12m 10)
- M7 VDD VB VBbar VDD PMOS
- M8 VBbar VB 0 0 NMOS
- .model NMOS NMOS
- .model PMOS PMOS
- .lib C:\Users\AJAY GOVINDAN\Documents\LTspiceXVII\lib\cmp\standard.mos
- .model nmos1 nmos(W= 1u L=1u vto= 0.7 kp= 200u)
- .model pmos1 pmos(W= 2u L=1u vto =-0.7 kp= 40u)
- .tran 30m
- * 2 input AND/NAND gate using NMOS complementary pass transistor logic
- .backanno
- .end

11.4.2) OR/NOR CPL LOGIC:



```
PULSE(5 0 0m 1p 1p 8m 10m 10)
PULSE(5 0 5m 1p 1p 8m 12m 10)
.model nmos1 nmos(W= 1u L=1u vto=0.7 kp=200u)
.model pmos1 pmos(W=2u L=1u vto=-0.7 kp=40u)
.tran 16MS
```

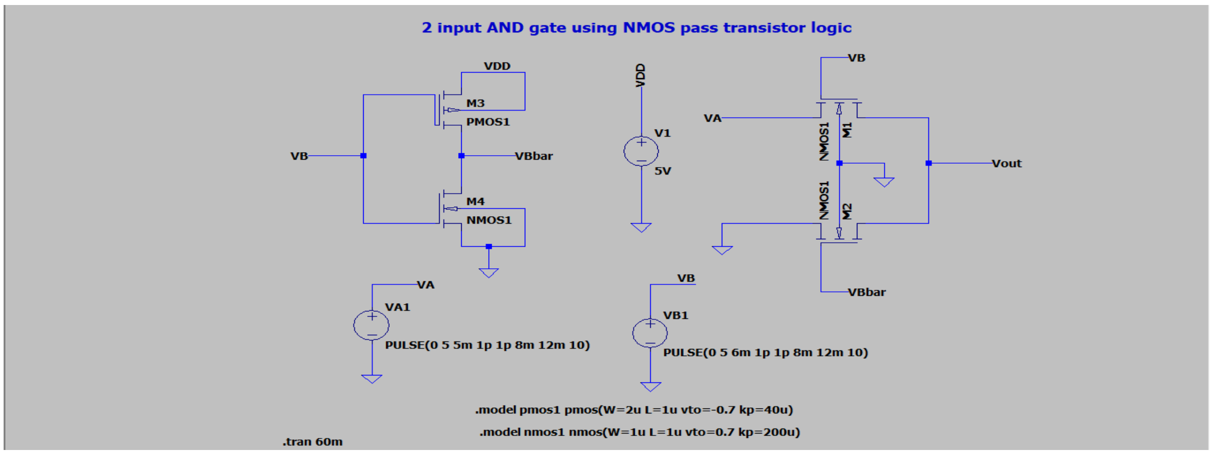
11.4.2 OUTPUT:



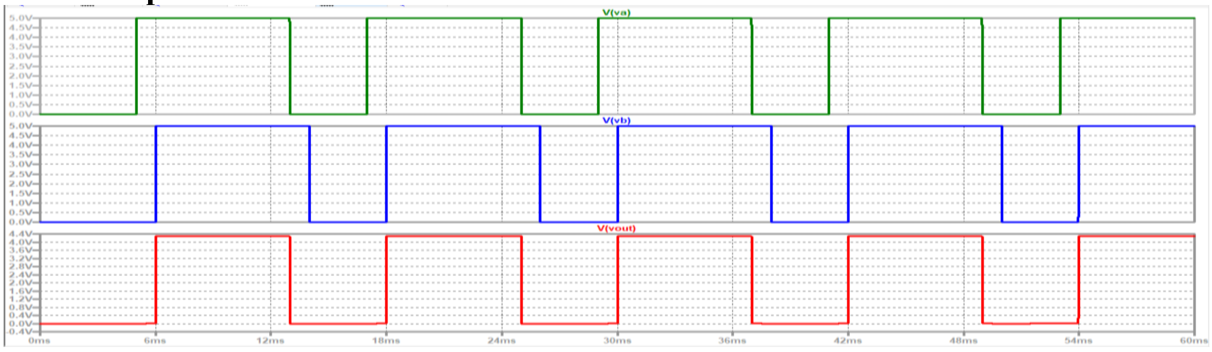
11.4.2 Spice Netlist:

- * F:\OR.asc
- M1 Y1 Bbar A 0 NMOS1
- M2 Y1 B B 0 NMOS1
- M3 Y2 Bbar Abar 0 NMOS1
- M4 Y2 B Bbar 0 NMOS1
- M5 VDD A Abar VDD PMOS1
- M6 VDD B Bbar VDD PMOS1
- M7 Abar A 0 0 NMOS1
- M8 Bbar B 0 0 NMOS1
- V1 VDD 0 5V
- VA1 A 0 PULSE(5 0 5m 1p 1p 8m 12m 10)
- VB1 B 0 PULSE(5 0 0m 1p 1p 8m 10m 10)
- .model NMOS NMOS
- .model PMOS PMOS
- .lib C:\Users\AJAY GOVINDAN\Documents\LTspiceXVII\lib\cmp\standard.mos
- * 2 input OR/ NOR gate using NMOS complementary pass transistor logic
- model nmos1 nmos(W= 1u L=1u vto=0.7 kp=200u)
- model pmos1 pmos(W=2u L=1u vto=-0.7 kp=40u)
- tran 16MS
- backanno
- end

11.4.3 AND GATE USING PTL:


2 input AND gate using NMOS pass transistor logic
VDD
M3
PMOS1
VB
VBbar
M4
NMOS1
VDD
V1
5V
VB
VA
NMOS1
M1
NMOS1
M2
Vout
VBbar
VA
VA1
PULSE(0 5 5m 1p 1p 8m 12m 10)
VB
VB1
PULSE(0 5 6m 1p 1p 8m 12m 10)
.model pmos1 pmos(W=2u L=1u vto=-0.7 kp=40u)
.model nmos1 nmos(W=1u L=1u vto=0.7 kp=200u)
.tran 60m

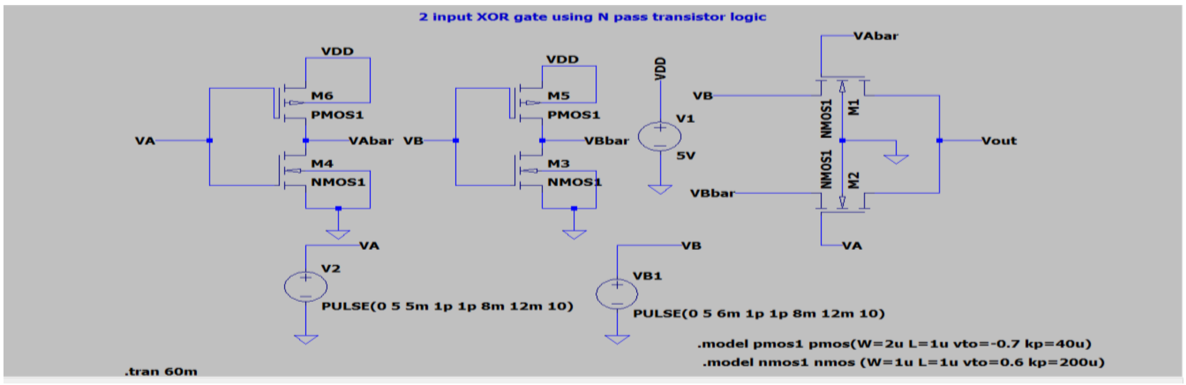
11.4.3 Output



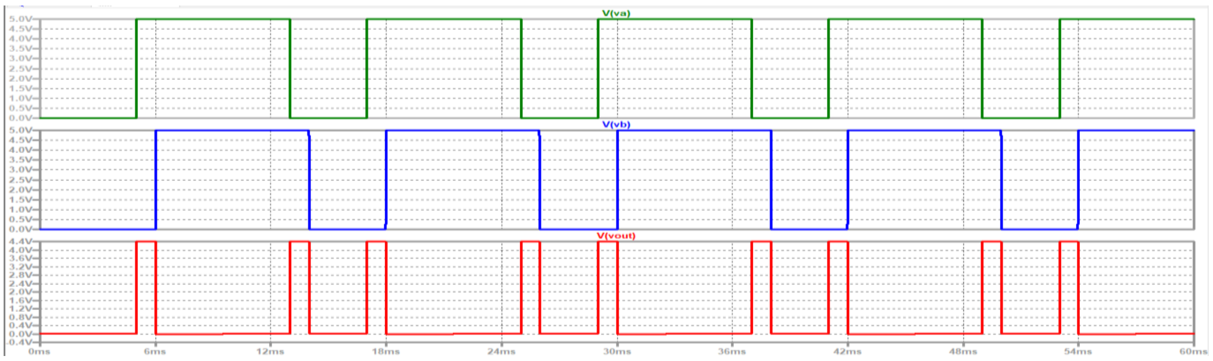
11.4.3 SPICE NETLIST:

- * F:\ANDPTL.asc
- M1 Vout VB VA 0 NMOS1
- M2 Vout VBbar 0 0 NMOS1
- M3 VDD VB VBbar VDD PMOS1
- M4 VBbar VB 0 0 NMOS1
- V1 VDD 0 5V
- VB1 VB 0 PULSE(0 5 6m 1p 1p 8m 12m 10)
- VA1 VA 0 PULSE(0 5 5m 1p 1p 8m 12m 10)
- .model NMOS NMOS
- .model PMOS PMOS
- .lib C:\Users\AJAY GOVINDAN\Documents\LTspiceXVII\lib\cmp\standard.mos
- * 2 input AND gate using NMOS pass transistor logic
- .model pmos1 pmos(W=2u L=1u vto=-0.7 kp=40u)
- .model nmos1 nmos(W=1u L=1u vto=0.7 kp=200u)
- .tran 60m
- .backanno
- .end

<u>11.4.4 XOR GATE USING PTL</u>


2 input XOR gate using N pass transistor logic
VDD
M6
PMOS1
VA
VAbar
M4
NMOS1
VDD
M5
PMOS1
VB
VBbar
M3
NMOS1
VDD
V1
5V
VAbar
VB
NMOS1
M1
NMOS1
M2
Vout
VBbar
VA
VA
V2
PULSE(0 5 5m 1p 1p 8m 12m 10)
VB
VB1
PULSE(0 5 6m 1p 1p 8m 12m 10)
.model pmos1 pmos(W=2u L=1u vto=-0.7 kp=40u)
.model nmos1 nmos (W=1u L=1u vto=0.6 kp=200u)
.tran 60m

11.4.4 OUTPUT :



<u>11.4.4 Spice Netlist:</u>

- * F:\XORPTL.asc
- M1 Vout VAbar VB 0 NMOS1
- M2 Vout VA VBbar 0 NMOS1
- M3 VBbar VB 0 0 NMOS1
- M4 VAbar VA 0 0 NMOS1

- M5 VDD VB VBbar VDD PMOS1
- M6 VDD VA VAbar VDD PMOS1
- V1 VDD 0 5V
- VB1 VB 0 PULSE(0 5 6m 1p 1p 8m 12m 10)
- V2 VA 0 PULSE(0 5 5m 1p 1p 8m 12m 10)
- .model NMOS NMOS
- .model PMOS PMOS
- .lib C:\Users\AJAY GOVINDAN\Documents\LTspiceXVII\lib\cmp\standard.mos
- * 2 input XOR gate using N pass transistor logic
- .model pmos1 pmos(W=2u L=1u vto=-0.7 kp=40u)
- .model nmos1 nmos (W=1u L=1u vto=0.6 kp=200u)
- .tran 60m
- .backanno
- .end

11.5 Post lab:
1. Realize tri-state buffer circuit.

11.6 Result:
Thus, the design and analysis of DCVSL, CPL and Pass transistor design have been performed and its characteristics are verified using LTSpice tool.

LAB EXPERIMENT # 12

Dynamic NAND gate

12.1 Objective: To study and design the two inputs n-type dynamic NAND gate using HSPICE and verify the simulation result.

12.2 Software required: Hspice

12.3 Pre-lab Questions

1. Differentiate static and dynamic CMOS logic.
2. Explain NORA CMOS logic.

12.4 Design two input dynamic NAND Gate

Circuit Diagram:

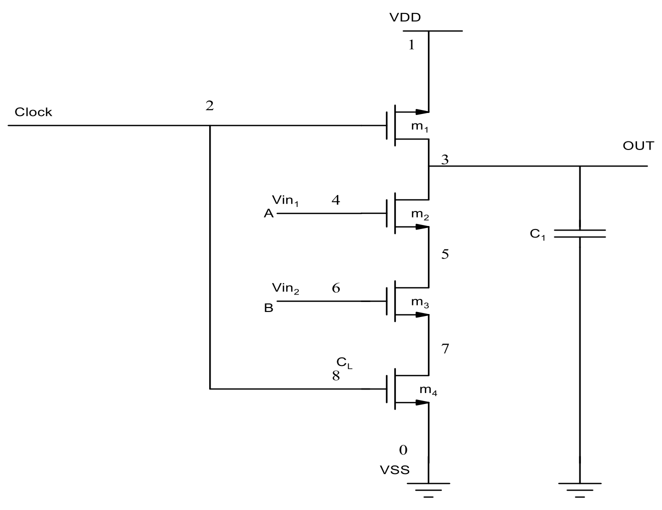


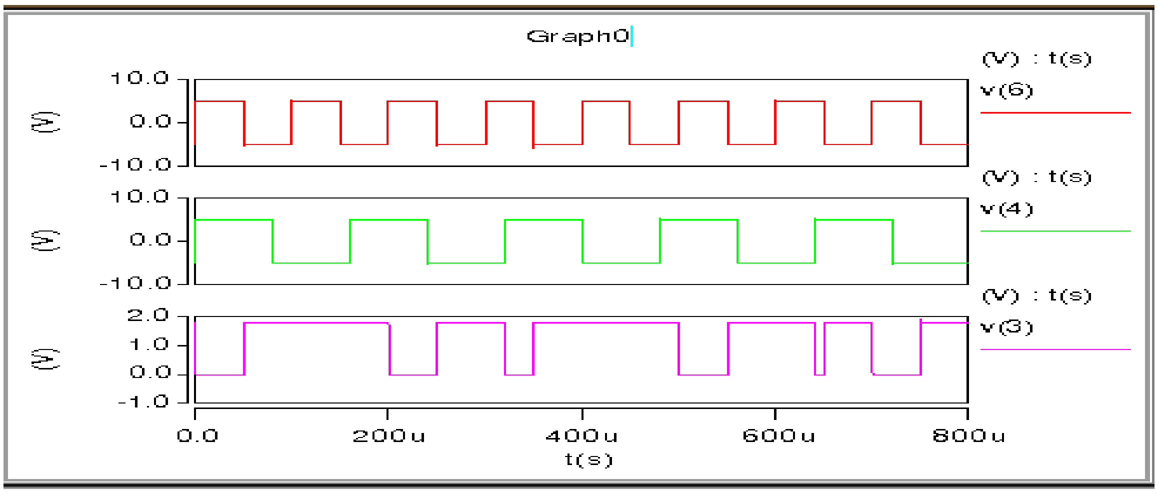
Figure 1. Two input dynamic NAND gate

12.4 Spice Netlist:

```
.option post
.include c:\synopsys\tsmc018.lib
vdd 1 0 dc 1.8v
vin1 4 0 pulse(-5 5 2ns 2ns 2ns 80us 160us)    //(input A)
vin2 6 0 pulse(-5 5 2ns 2ns 2ns 50us 100us)    //(input B)
vin3 2 0 pulse(-5 5 2ns 2ns 2ns 50us 100us)    //(CLOCK)
vin4 8 0 pulse(-5 5 2ns 2ns 2ns 50us 100us)    //(CLOCK)
m1 3 2 1 1  cmosp w=4u l=180nm
m2 3 4 5 5 cmosn w=2u l=180nm
m3 5 6 7 7  cmosn w=2u l=180nm
m4 7 8 0 0   cmosn w=2u l=180nm
```

```
c1 3 0 100p    // (CAPACITOR→ Vary the capacitor value to see the dynamic performance)
.tran 100u 800u
.plot v(3) v(4) V(6)
.end
```

OUTPUT : N TYPE DYNAMIC- NAND GATE



12.7 Netlist :

```
.option post
.include c:\synopsys\tsmc018.lib
*  t58f spice bsim3 version 3.1 parameters
*
* spice 3f5 level 8, star-hspice level 49, utmost level 8
*
 * temperature_parameters=default
 *
vdd 1 0 dc 1.8v
vin1 4 0 pulse(-5 5 2ns 2ns 2ns 80us 160us)
vin2 6 0 pulse(-5 5 2ns 2ns 2ns 50us 100us)
vin3 2 0 pulse(-5 5 2ns 2ns 2ns 50us 100us)
vin4 8 0 pulse(-5 5 2ns 2ns 2ns 50us 100us)
m1 3 2 1 1  cmosp w=4u l=180nm
m2 3 4 5 5 cmosn w=2u l=180nm
m3 5 6 7 7  cmosn w=2u l=180nm
m4 7 8 0 0   cmosn w=2u l=180nm
c1 3 0 100p
.tran 100u 800u
.plot v(3) v(4) v(6)
.end

******
 dynamic nand
  ******  mos   model parameters          tnom=  25.000 temp=  25.000
 ******
 ***************************************************************************
```

```
***       model parameters  model name:    0:cmosn     model type:nmos    ***
*****************************************************************************

 *** general parameters ***
  deriv=   0.

 ***  level 49  model parameters  ***

    hspver=    2006.03              level=         49
   version=        3.1           paramchk=          0
    apwarn=          1               lite=          0
    vgslim=          0            binUnit=          1
    capMod=          2              xpart=        0.5
    mobMod=          1            nqsMod=          0
    stiMod=          0                elm=          5
  sfvtflag=          0                tox=   4.1e-009 meter
        xj=     1e-007 meter      binflag=          0
      lmin=          0 meter         lmax=          0 meter
      wmin=          0 meter         wmax=          0 meter
      lref=          0 meter         wref=          0 meter
      lint=1.69004e-008 meter        wint=          0 meter
     lmult=          1              wmult=          1
        ll=          0                lln=          1
        lw=          0                lwn=          1
       lwl=          0                 wl=          0
       wln=          1                 ww=          0
       wwn=          1                wwl=          0
       dwg=-4.72872e-009 m/V          dwb=-2.45241e-009 m/V^1/2
        a0=          2                ags=   0.444996 V^-1
        b0=1.90108e-007 meter          b1=4.99995e-006 meter
      keta= -0.0164863 V^-1          voff= -0.0948017 V
     ngate=          0 cm^-3          vbx=          0 V
       vbm=         -3 V               xt=  1.55e-007 meter
      vth0=   0.366247 V              nch=2.3549e+017 cm^-3
      nsub=     6e+016 cm^-3          nlx=1.63068e-007 meter
    gamma1=          0 V^-1/2      gamma2=          0 V^-1/2
        k1=     0.5865 V^1/2           k2= 0.00112727
        k3=      0.001                k3b=  0.0294061 V^-1
        w0=     1e-007 meter         dvt0=    1.20646
      dvt1=   0.421549               dvt2=  0.0197749 V^-1
     dvt0w=          0 meter^-1     dvt1w=          0 meter^-1
     dvt2w=          0 V^-1          dsub=  0.0145467
      eta0= 0.00223093               etab=6.02898e-005 V^-1
        u0=    273.809 cm^2/V/sec      ua=-1.40499e-009 m/V
        ub=2.40832e-018 (m/V)^2        uc=6.50483e-011 V^-1
      vsat=     135501 m/sec           a1=0.000386877 v^-1
        a2=   0.464027              delta=       0.01 V
      rdsw=    123.338 ohm-um        prwg=        0.5 V^-1
      prwb=  -0.197728 V^-1/2          wr=          1
      pclm=    1.38221            pdiblc1=   0.176279
    pdiblc2= 0.00166653            pdiblcb=       -0.1 V^-1
     pscbe1=8.91287e+009 V/m          pscbe2=7.34961e-009 V/m
      drout=   0.769469               pvag= 0.00168592
    nfactor=    2.18601               cdsc=    0.00024 f/m^2
      cdscb=          0 f/V/m^2       cdscd=          0 f/V/m^2
        cit=          0 f/m^2       alpha0=          0 m/V
      beta0=         30 V              dlc=1.69004e-008 meter
        dwc=          0 meter          clc=     1e-007 meter
        cle=        0.6               cgso=  8.23e-010 f/m
       cgdo=  8.23e-010 f/m            cgbo=     1e-012 f/m
       cgsl=          0 f/m           cgdl=          0 f/m
     ckappa=        0.6                 cf=          0 f/m
       tnom=     300.15 K              kt1=      -0.11 V
       kt1l=          0                kt2=      0.022
        ute=       -1.5                ua1=  4.31e-009 m/V
        ub1= -7.61e-018 (m/V)^2        uc1=  -5.6e-011 m/V^2
         at=      33000 m/s            prt=          0
        using Berkeley noise model       noiMod=          2
       noia=1.31826e+019              noib=     144544
       noic=-1.24516e-012               em=   4.1e+007
         ef=       0.92                 af=          1
         kf=          0            gdsnoi=-1.23457e-029
        using Hspice diodes            using ACM
        acm=          0               hdif=          0 meter
       ldif=          0 meter           js=          0 amp/m^2
        jsw=          0 amp/m          xti=          0
          n=          1               cj=0.000946643 f/m^2
         mj=  0.382027                 pb=        0.8 V
       cjsw=2.60815e-010 f/m          mjsw=  0.102322
        php=        0.8 V              pbsw (not used)
     cjgate=2.60815e-010 f/m           cjswg (not used)
      mjswg (not used)                 pbswg (not used)
      lketa= 0.00716508               wketa=-0.00288098
      pketa=-0.00675928               pvth0=-0.00219937
        pk2= 0.00159325               peta0=0.000100316
        pu0=    6.77752                 pua=5.50542e-012
        pub=8.84133e-025              pvsat=    2006.29
      prdsw=  -0.936896
*******************************************************************************
***       model parameters  model name:   0:cmosp    model type:pmos    ***
*******************************************************************************

  *** general parameters ***
   deriv=   0.

  ***  level 49  model parameters  ***

     hspver=    2006.03              level=         49
    version=        3.1           paramchk=          0
     apwarn=          1               lite=          0
  vgslim=         0           binUnit=         1
  capMod=          2              xpart=       0.5
  mobMod=          1              nqsMod=         0
  stiMod=         0               elm=          5
sfvtflag=         0                tox=   4.1e-009 meter
      xj=     1e-007 meter       binflag=          0
    lmin=          0 meter         lmax=         0 meter
    wmin=          0 meter         wmax=         0 meter
    lref=          0 meter         wref=          0 meter
    lint=2.94454e-008 meter          wint=          0 meter
   lmult=          1              wmult=          1
      ll=          0              lln=          1
      lw=          0              lwn=          1
     lwl=          0               wl=          0
     wln=          1               ww=          0
     wwn=          1               wwl=          0
     dwg=-2.79872e-008 m/V             dwb=-4.83797e-010 m/V^1/2
      a0=    1.99766                 ags=   0.418694 V^-1
      b0=1.94918e-007 meter            b1=6.42291e-007 meter
    keta=  0.0166345 V^-1           voff=  -0.095236 V
   ngate=          0 cm^-3          vbx=          0 V
     vbm=         -3 V               xt=  1.55e-007 meter
    vth0=  -0.390601 V               nch=4.1589e+017 cm^-3
    nsub=     6e+016 cm^-3           nlx=1.19407e-007 meter
  gamma1=          0 V^-1/2      gamma2=          0 V^-1/2
      k1=   0.534131 V^1/2           k2=  0.0395326
      k3=          0              k3b=    7.49162 V^-1
      w0=     1e-006 meter          dvt0=   0.506055
    dvt1=   0.242384               dvt2=        0.1 V^-1
   dvt0w=          0 meter^-1      dvt1w=          0 meter^-1
   dvt2w=          0 V^-1          dsub= 0.00181655
    eta0=  0.0010355               etab=-0.00043584 V^-1
      u0=    115.689 cm^2/V/sec         ua=1.57375e-009 m/V
      ub=1.87431e-021 (m/V)^2          uc=    -1e-010 V^-1
    vsat=     113098 m/sec            a1=   0.474915 v^-1
      a2=   0.300003              delta=       0.01 V
    rdsw=    198.321 ohm-um         prwg=        0.5 V^-1
    prwb=  -0.498665 V^-1/2          wr=          1
    pclm=    1.32999             pdiblc1= 0.00176656
 pdiblc2=7.72839e-007             pdiblcb=     -0.001 V^-1
  pscbe1=4.87218e+010 V/m          pscbe2=     5e-010 V/m
   drout= 0.00101189               pvag=  0.0209921
 nfactor=          2              cdsc=    0.00024 f/m^2
   cdscb=          0 f/V/m^2      cdscd=          0 f/V/m^2
     cit=          0 f/m^2      alpha0=          0 m/V
   beta0=         30 V              dlc=2.94454e-008 meter
     dwc=          0 meter          clc=     1e-007 meter
     cle=        0.6              cgso=  6.35e-010 f/m
    cgdo=  6.35e-010 f/m            cgbo=     1e-012 f/m
    cgsl=          0 f/m          cgdl=          0 f/m
```

```
       ckappa=        0.6                  cf=          0 f/m
         tnom=     300.15 K               kt1=      -0.11 V
         kt1l=          0                kt2=      0.022
          ute=       -1.5                ua1=  4.31e-009 m/V
          ub1= -7.61e-018 (m/V)^2         uc1=  -5.6e-011 m/V^2
           at=      33000 m/s             prt=          0
          using Berkeley noise model       noiMod=          2
         noia=3.57457e+018                noib=       2500
         noic=2.6126e-011                  em=   4.1e+007
           ef=     1.1388                  af=          1
           kf=          0             gdsnoi=-1.23457e-029
          using Hspice diodes            using ACM
          acm=          0                hdif=          0 meter
         ldif=          0 meter            js=          0 amp/m^2
          jsw=          0 amp/m           xti=          0
            n=          1                cj=0.00114452 f/m^2
           mj=  0.409952                  pb=  0.846869 V
         cjsw=2.49075e-010 f/m            mjsw=  0.347856
          php=  0.846869 V               pbsw (not used)
       cjgate=2.49075e-010 f/m            cjswg (not used)
        mjswg (not used)                pbswg (not used)
        lketa=-0.00149587               wketa=  0.0222457
        pketa=-0.00253656               pvth0= 0.00230202
          pk2= 0.00182191               peta0=2.82779e-005
          pu0=   -1.55806               pua=-6.36889e-011
          pub=     1e-021             pvsat=     49.842
        prdsw=    9.05753

Model:    0:cmosn
 W = 1.99e-006, L = 1.8e-007

dynamic nand
  ******  operating point information      tnom=  25.000 temp=  25.000
 ******
 ***** operating point status is voltage   simulation time is      0.
    node    =voltage      node    =voltage      node    =voltage

 +0:1       =   1.8000  0:2       =  -5.0000  0:3       =   1.8000
 +0:4       =  -5.0000  0:5       =   1.2000  0:6       =  -5.0000
 +0:7       = 600.0000m 0:8       =  -5.0000

 ******
 dynamic nand
  ******  transient analysis              tnom=  25.000 temp=  25.000
 ******
a
 legend:
  a: v(3)
  b: v(4)
  c: v(6)
```

```
    time       v(3)
 (abc     )    -10.0000      -5.0000       0.          5.0000      10.0000
                  +            +            +           +            +
   0.        1.800 -+------+------2------+------+----a-+------+------+------+-
  100.0000u    1.800  +      +      2      +      +    a +      +      +      +
  200.0000u    1.800  +      +      c      +      +    a +      b      +      +
  300.0000u    1.800  +      +      2      +      +    a +      +      +      +
  400.0000u    1.800  +      +      c      +      +    a +      b      +      +
  500.0000u    1.800  +      +      c      +      +    a +      b      +      +
  600.0000u    1.800  +      +      2      +      +    a +      +      +      +
  700.0000u    1.800  +      +      c      +      +    a +      b      +      +
  800.0000u    1.800  +      +      2      +      +    a +      +      +      +
                  +            +            +           +            +
B
dynamic nand
  ******  job statistics summary           tnom=  25.000 temp=  25.000
 ******

        total memory used        161 kbytes

  # nodes =       9 # elements=      10
  # diodes=       0 # bjts    =       0 # jfets   =       0 # mosfets =       4
  # va device =       0

    analysis          time    # points   tot. iter  conv.iter

    op point          0.03         1         32
    transient         0.05         9       2016        560 rev=       114
    readin           0.01
    errchk           0.00
    setup            0.00
    output           0.00
         total cpu time          0.14 seconds
            job started at  12:36:05 02/26/2020
            job ended   at  12:36:05 02/26/2020
```

12.5 Post Lab Questions:

1.Design the n-type dynamic NOR gate using the HSPICE.
2.Realize n –type dynamic AND gate using the HSPICE.

12.6 Result:

Thus the design of two input dynamic NAND gate, MOS transistor level using HSPICE was studied and simulated.