SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Department of Electronics and Communication Engineering

21ECC311L-VLSI DESIGN LAB

Laboratory Manual

College of Engineering & Technology SRM Institute of Science and Technology

Tiruchrappalli – 621 105,

Near Samayapuram 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.

21ECC311L VLSI Design

Sl. No.	Description of experiments	Session
Software's: Xilinx vivado		
1	Exp 1: Realization of Combinational and Sequential Circuits using gate level modeling	1
2	Exp 2: Realization of digital circuits using behavioral modeling	2
3	Exp 3: Design using FSM and ASM charts	3
4	Exp 4: Realization of VLSI Adders-I	4
5	Exp 5: Realization of VLSI Adders-II	5
6	Exp 6: Realization of VLSI Multiplier-I	6
7	Exp 7: Realization of VLSI Multiplier-II	7
8	Exp 8: Realization of RAM and ROM	8
9	Exp 9: Design and Analysis of CMOS inverter using HSPICE	9
10	Exp 10: Design and analysis of 4- input Dynamic NAND gate using HSPICE	10

1.Evaluation Scheme

Component	**Assessment Tool**	**Weightage**
Practical	Course Learning Assessment P1	20%
	Course Learning Assessment P2	20%
	Course Learning Assessment P3	20%
	Course Learning Assessment P4 (Model Practical Exam and Mini Project)	40%
End-Semester Lab Examination		**100%**

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

Laboratory Report Cover Sheet

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

College of Engineering and Technology

Department of Electronics and Communication Engineering

21ECC311L VLSI Design

V Semester, 2024-2025 (Odd 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 GateLevel 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 Xilinx vivado tools

1.2 Software tools Requirement

Software's: Xilinx vivado

1.3 Prelab Questions

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

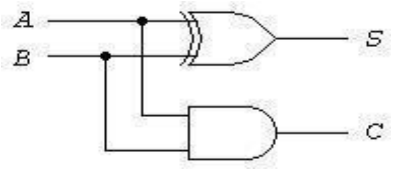
1. List the types of design methodologies for digital design with an example? 2.Give the difference between module and module instance.
3. What are built in gate primitives?
4. Give the use of net, reg. and wire data types. 5.Declare the following variables in Verilog:

1.4 Problem: Write a Verilog code to implement Ripple Carry Adder.

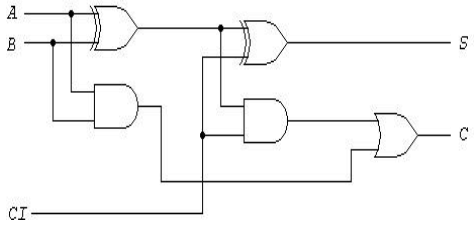
The following points should be taken care of:

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

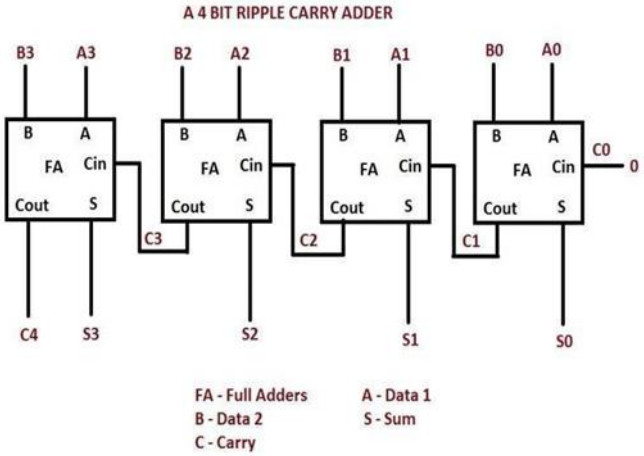
Logic Diagram – Problem 1 Half adder:



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
```

Testbench:

```
module half_adder_tb_v;
reg x;
reg y;
wire s;
wire cout;
Half_adder uut (.s(s),.cout(cout),.x(x),.y(y));
initial begin
x = 0; y = 0;
#100;
x = 0; y = 1;
```

```
#100;
x = 1; y = 0;
#100;
x = 1; y = 1;
#100;
$finish;
end
```

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 (.s(w1),.cout(w2),.x(x),.y(y));
Half_adder h2 (.s(sum),.cout(w3),.x(w1),.y(cin));
or g1 (carry, w2, w3);
endmodule
```

Testbench

```
module full_adder_tb_v;
reg x;
reg y;
reg cin;
wire sum;
wire carry;
Full_adder uut (. sum (sum), carry(carry),.x(x), y(y), cin(cin));
initial begin
$display ("x y cin | sum carry");
x = 0; y = 0; cin = 0; #100;
$display ("%b %b %b | %b %b", x, y, cin, sum, carry);
x = 0; y = 0; cin = 1; #100;
$display ("%b %b %b | %b %b", x, y, cin, sum, carry);
x = 0; y = 1; cin = 0; #100;
$display ("%b %b %b | %b %b", x, y, cin, sum, carry);
x = 0; y = 1; cin = 1; #100;
$display ("%b %b %b | %b %b", x, y, cin, sum, carry);
```

```
x = 1; y = 0; cin = 0; #100;
$display ("%b %b %b | %b %b", x, y, cin, sum, carry);
x = 1; y = 0; cin = 1; #100;
$display ("%b %b %b | %b %b", x, y, cin, sum, carry);
x = 1; y = 1; cin = 0; #100;
$display ("%b %b %b | %b %b", x, y, cin, sum, carry);
x = 1; y = 1; cin = 1; #100;
$display ("%b %b %b | %b %b", x, y, cin, sum, carry);
$stop;
end
endmodule
```

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

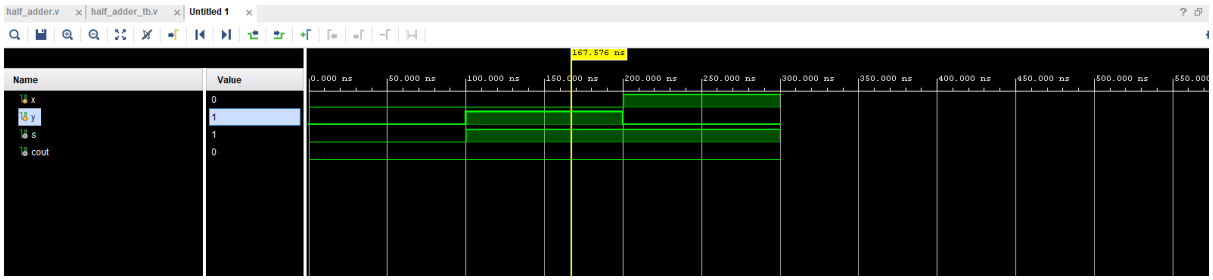
```
module ripple_carry_adder(S, cout, A, B, cin);
output [3:0] S;
output cout;
input [3:0] A;
input [3:0] B;
input cin;
wire c1, c2, c3;
full_adder f1 (S[0], c1, A[0], B[0], cin); // Least significant bit
full_adder f2 (S[1], c2, A[1], B[1], c1);
full_adder f3 (S[2], c3, A[2], B[2], c2);
full_adder f4 (S[3], cout, A[3], B[3], c3); // Most significant bit
endmodule
```

Testbench

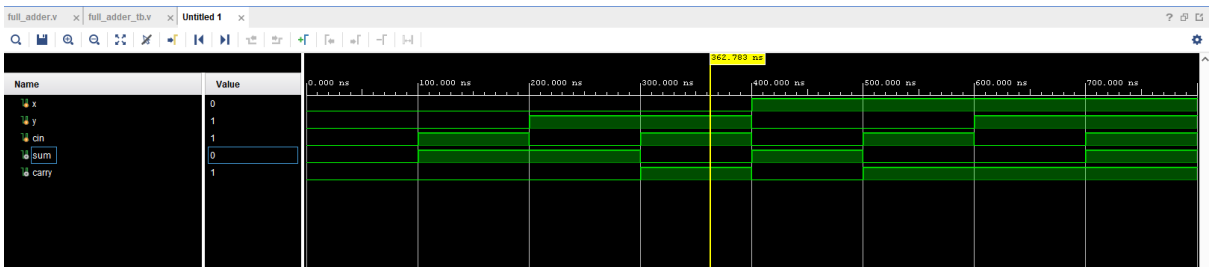
```
module ripple_carry_adder_tb_v;
reg [3:0] A;
reg [3:0] B;
reg cin;
wire [3:0] S;
wire cout;
ripple_carry_adder uut (.S(S),.cout(cout),.A(A),.B(B),.cin(cin));
initial begin
$display("A B cin | S cout");
A = 4'b0000; B = 4'b0000; cin = 0; #100;
$display("%b %b %b | %b %b", A, B, cin, S, cout);
A = 4'b0000; B = 4'b0000; cin = 1; #100;
```

```
$display("%b %b %b | %b %b", A, B, cin, S, cout);
A = 4'b0000; B = 4'b0001; cin = 0; #100;
$display("%b %b %b | %b %b", A, B, cin, S, cout);
A = 4'b0000; B = 4'b0001; cin = 1; #100;
$display("%b %b %b | %b %b", A, B, cin, S, cout);
A = 4'b0001; B = 4'b0000; cin = 0; #100;
$display("%b %b %b | %b %b", A, B, cin, S, cout);
A = 4'b0001; B = 4'b0000; cin = 1; #100;
$display("%b %b %b | %b %b", A, B, cin, S, cout);
A = 4'b0001; B = 4'b0001; cin = 0; #100;
$display("%b %b %b | %b %b", A, B, cin, S, cout);
A = 4'b0001; B = 4'b0001; cin = 1; #100;
$display("%b %b %b | %b %b", A, B, cin, S, cout);
A = 4'b1111; B = 4'b1111; cin = 1; #100;
$display("%b %b %b | %b %b", A, B, cin, S, cout);
$finish;
end
endmodule
```

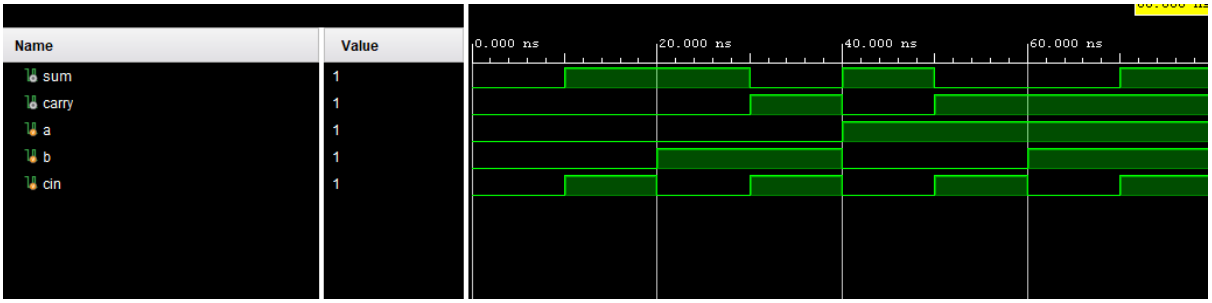
Waveforms – Problem 1



Half Adder using Data Flow Model



Full Adder using Structural Model



Ripple Carry Adder using 1-Bit Full Adder

1.5 Result:

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

Lab Experiment #2
Realization of Digital Circuits Using Behavioral Modeling

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

2.2 Software tools Requirement

Software's: Xilinx vivado

2.3 Prelab Questions

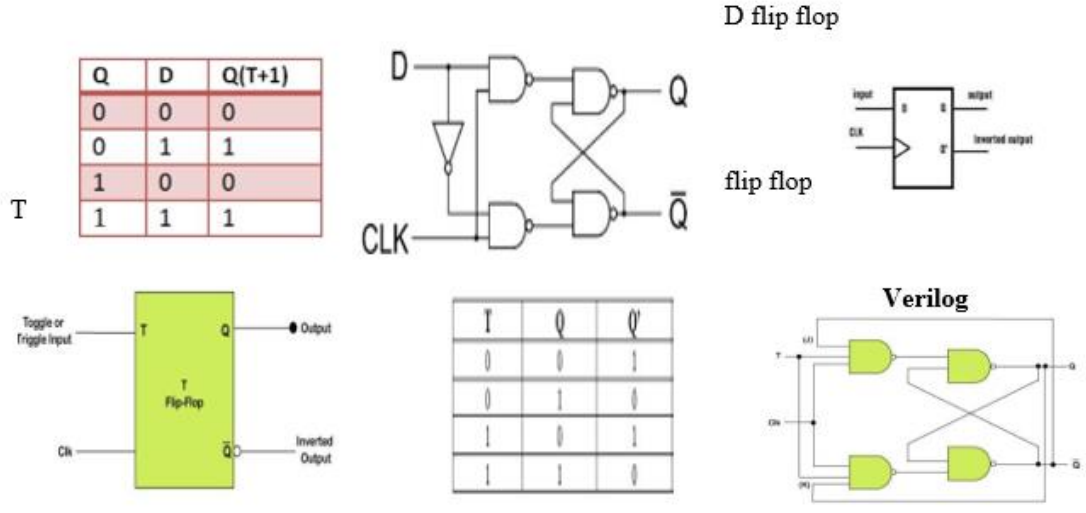
(write pre lab Q & A in an A4 sheet)

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



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
```

Testbench

```
module dff_tb_v;
reg d;
reg clk;
reg clear;
wire q;
wire qbar;
dff uut (.q(q),.qbar(qbar),.d(d),.clk(clk),.clear(clear));
initial begin
d = 0; clk = 0; clear = 1;
#100 d = 0; clear = 0;
#100 d = 1; clear = 0;
#100 d = 1; clear = 1;
end
always #50 clk = ~clk;
endmodule
```

1.(b) NEGATIVE EDGE TRIGGERED T FLIPFLOP USING CASE STATEMENT

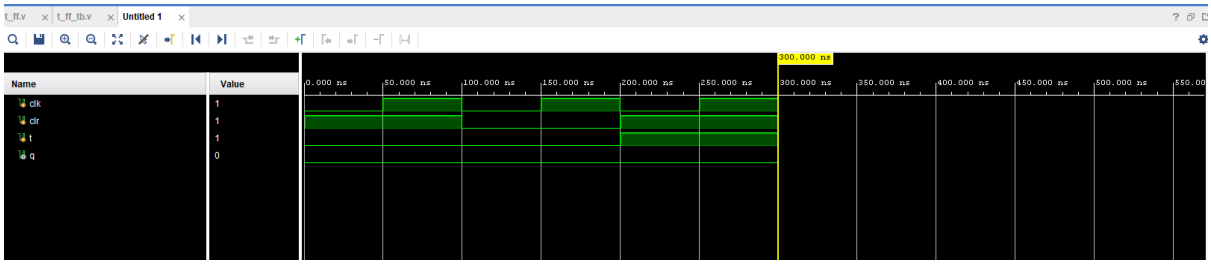
```
module tffcase(q, clk, clr, t);
output q;
input clk;
input clr;
input t;
reg q;
initial q = 0;
always @(negedge clk) begin
case ({clr, t})
2'b10: q = 0;
2'b00: q = q;
2'b01: q = ~q;
endcase
end
endmodule
```

Testbench

```
module tffcase_tb_v;
reg clk;
reg clr;
reg t;
wire q;
tffcase uut (.q(q),.clk(clk),.clr(clr),.t(t));
initial begin
clk = 0; clr = 1; t = 0;
#100; clr = 0;
#100; clr = 0; t = 1;
end
always #50 clk = ~clk;
endmodule
```

Waveforms – Problem 1

D Flipflop using IF statement



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.

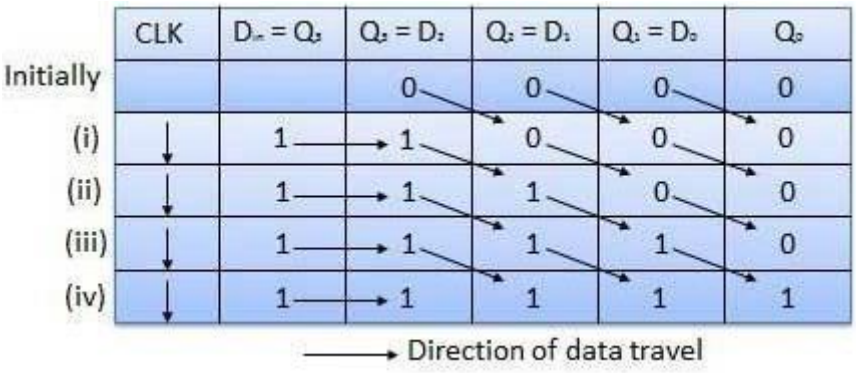
Logic Diagram – Problem 2 Up-Down Counter:

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:



Verilog Code - Problem 2

2. (a) UP DOWN COUNTER USING BEHAVIOURAL MODEL

```
module updowncntr(q, clr, clk, mode);
output reg [3:0] q;
input clr;
input clk;
input mode;
always @(posedge clk) begin
case ({clr, mode})
2'b11: q = 0;
```

```
2'b10: q = 0;
2'b01: q = q + 1;
2'b00: q = q - 1;
endcase
end
endmodule
```

Testbench

```
module updowncntr_tb_v;
reg clr;
reg clk;
reg mod;
wire [3:0] q;
updowncntr uut (.q(q),.clr(clr),.clk(clk),.mode(mod));
initial begin
clr = 1; clk = 0; mod = 1;
#100;
clr = 0;
#1000;
mod = 0;
end
always #50 clk = ~clk;
endmodule
```

2.(b) 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
```

Testbench

```
module modncounter_tb_v;
reg clk;
reg rstn;
wire [3:0] out;
modN_ctr uut (.clk(clk),.rstn(rstn),.out(out));
initial begin
clk = 1;
rstn = 1;
#100;
rstn = 0;
end
always #50 clk = ~clk;
endmodule
```

2.(c) SISO REGISTER USING BEHAVIOURAL MODEL

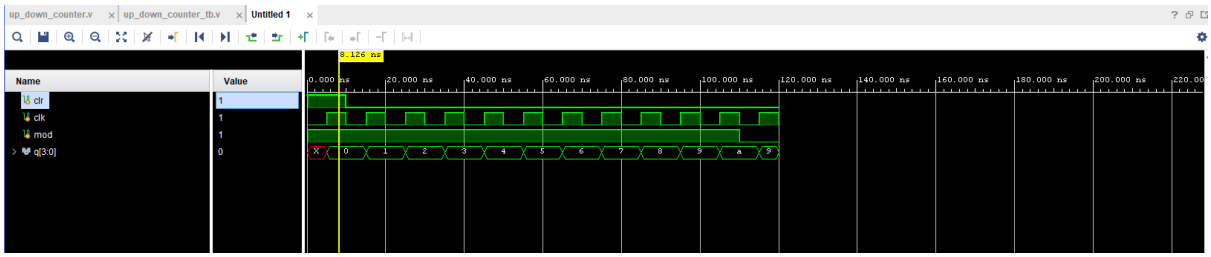
```
module siso_register(
output reg shift_out,
input shift_in,
input clk
);
reg [2:0] data;
always @(negedge clk) begin
data[0] <= shift_in;
data[1] <= data[0];
data[2] <= data[1];
shift_out <= data[2];
end
endmodule
```

Testbench

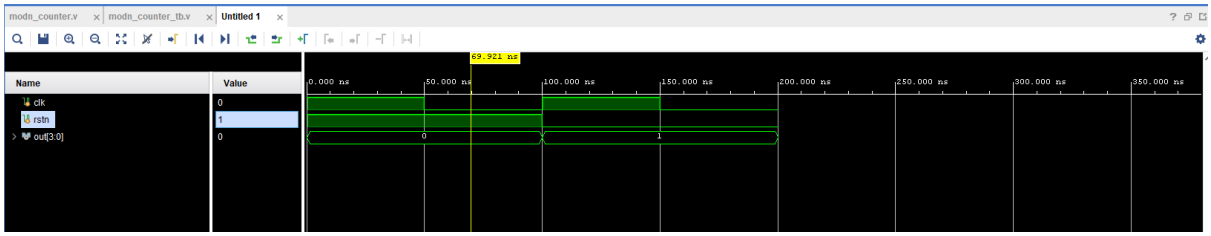
```
module siso_tb_v;
reg shift_in;
reg clk;
wire shift_out;
siso_register uut (.shift_out(shift_out),.shift_in(shift_in),.clk(clk));
initial begin
shift_in = 1;
```

```
clk = 1;
#100;
shift_in = 0;
#100;
shift_in = 1;
#100;
shift_in = 0;
end
always #50 clk = ~clk;
endmodule
```

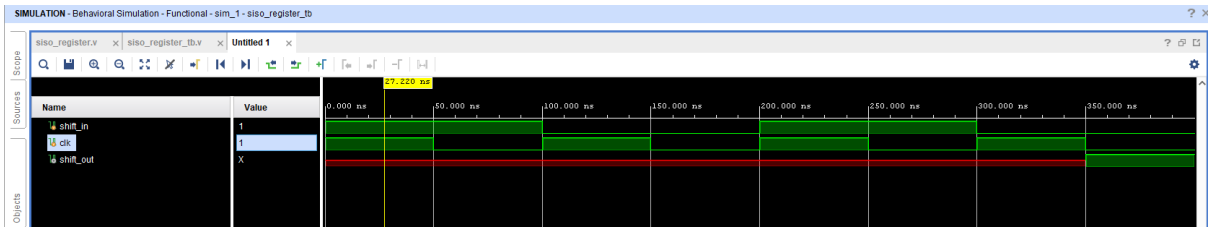
Waveforms – Problem 2



Up Down counter using behavioral modelling



Mod-N counter using behavioral modelling



SISO register using behavioral modelling

2.2 Post Lab :

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

2.3 Result:

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

Lab Experiment #3

Design of Finite State Machine (FSM) and Algorithmic State Machine

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 Xilinx vivadotools.

3.2 Tools Required:

Software's: Xilinx vivado

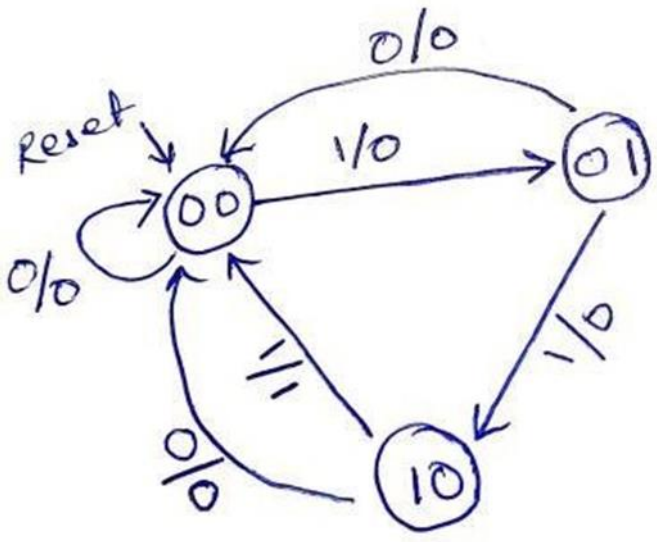
3.3 Pre Lab Questions:

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)

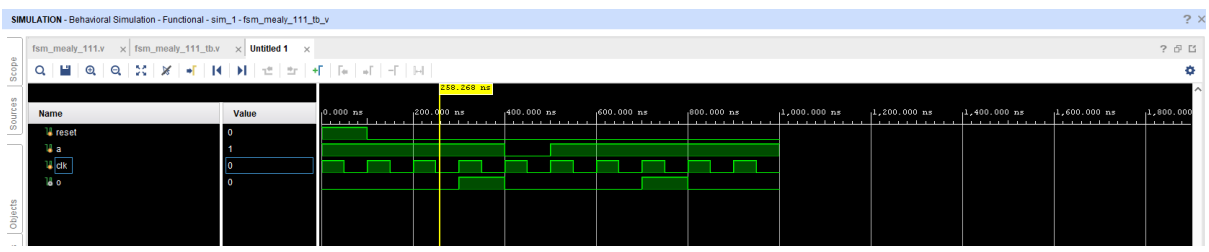
```
module fsm_mealy_111(o,reset,a,clk);
output reg o;
input reset,a,clk;
reg[1:0]pre_s, nxt_s;
initial
begin
pre_s=2'b00;
end
always@(posedge clk)
begin if(reset==1)
begin
pre_s=2'b00; o=0;
end
else
begin
case(pre_s)2'b00:
begin if (a==1)
begin o=0;
nxt_s=2'b01;
pre_s=nxt_s;
end
else
begin o=0;
nxt_s=2'b00;
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_s=2'b00;
endcase
end
end
endmodule
```

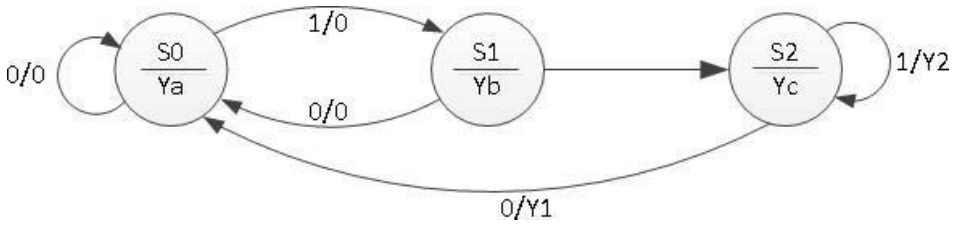
Testbench

```
module fsm_mealy_111_tb_v;
reg reset; reg a; reg clk;
wire o;
fsm_mealy_111 uut (.o(o),.reset(reset),.a(a),.clk(clk));
initial
begin
reset = 1; a = 1;clk = 1;
#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;
end
always #50 clk=~clk;
endmodule
```

Simulation Result:

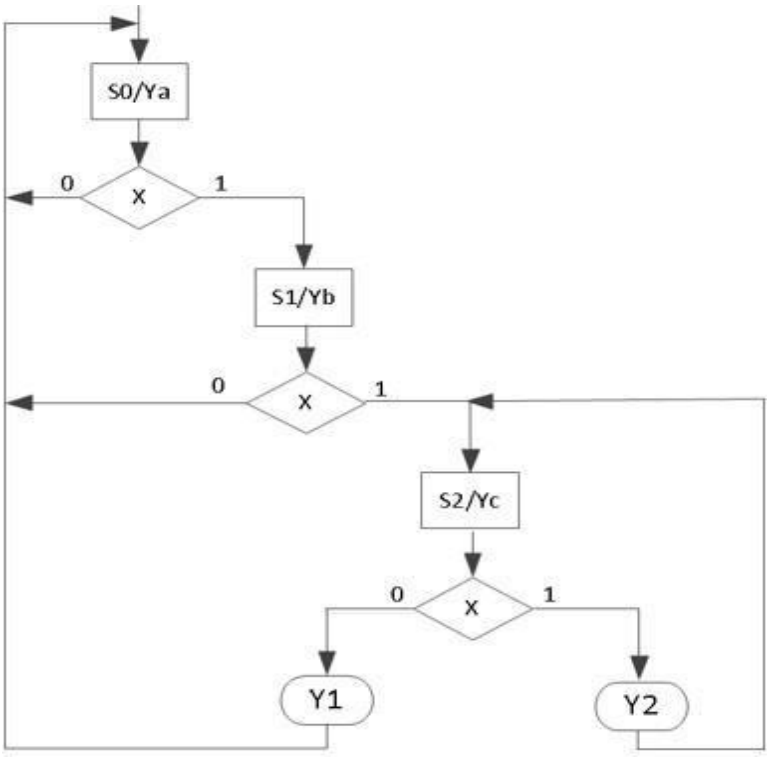


3.4.2 Consider the following state graph of a sequential network which has both Mealy and Moore outputs. The outputs Y1 and Y2 are the Mealy outputs and so should be conditional outputs. The Ya, Yb, and Yc are the Moore outputs so they should be part of state box. Input X can either be “0” or “1” and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

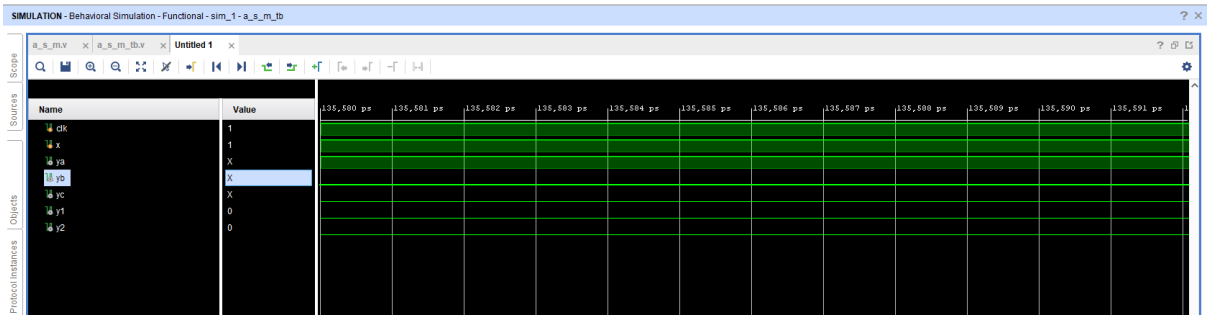
```
module asm(
input clk, x,
output reg ya,yb,yc,
output reg y1,y2
);
reg [1:0] state, nextstate;
parameter [1:0] S0=0, S1=1, S2=2;
always @(posedge clk)
state <= nextstate;
always @(state or x)
begin
y1 = 1'b0;
y2 = 1'b0;
case (state)
S0:
if(x) nextstate = S1;
else nextstate = S0;
S1: if(x)
nextstate = 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)
```

```
begin ya = 1'b0;
yb = 1'b0;
yc = 1'b0;
case(state)
S0: ya = 1'b1;
S1: yb = 1'b1;
S2: yc =1'b1;
default:
begin
ya = 1'b0;
yb= 1'b0;
yc =1'b0;
end
endcase
end
endmodule
```

Testbench

```
module asm_tb();
reg clk;
reg x;
wire ya;
wire yb;
wire yc;
wire y1;
wire y2;
asm uut ( .clk(clk),.x(x),.ya(ya),.yb(yb),.yc(yc),.y1(y1), .y2(y2));
initial begin
x= 1; clk = 1;
#100;
x = 1;
#100;
x = 1;
#100;
x = 1;
#100;
x = 0;
#100;
end always #50
clk=~clk;
endmodule
```

Simulation Result:



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 Xilinx vivado tools.

4.2 Software tools Requirement

Software's: Xilinx vivado

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:

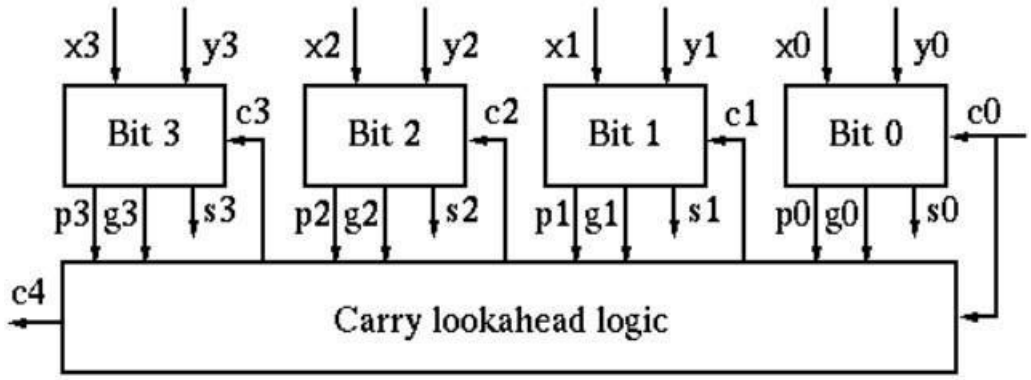


Fig. 4.1: 4-bit Carry Look-ahead Adder Source

Verilog code:

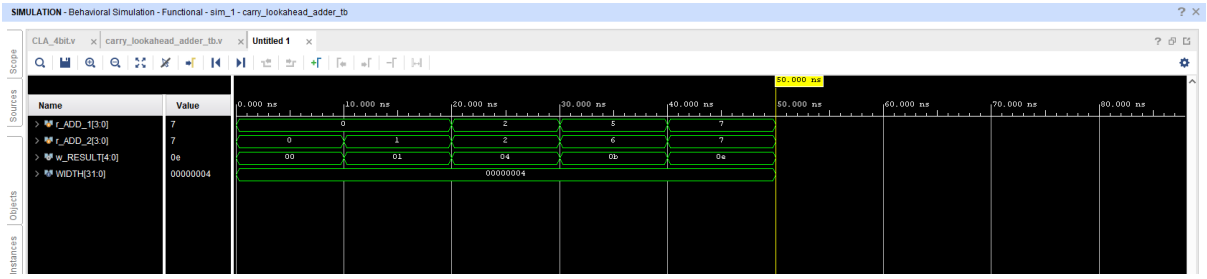
```
module CLA_4bit(
output [3:0] sum, // 4-bit sum output
output c_4, // Carry-out output
input [3:0] a, // First 4-bit input
input [3:0] b, // Second 4-bit input
input c_0 // Initial carry-in
);
// Internal wire declarations
wire p0, p1, p2, p3; // Propagate signals
wire g0, g1, g2, g3; // Generate signals
wire c1, c2, c3, c4; // Carry signals
// Propagate and Generate calculations
assign p0 = a[0] ^ b[0];
assign p1 = a[1] ^ b[1];
assign p2 = a[2] ^ b[2];
assign p3 = a[3] ^ b[3];
assign g0 = a[0] & b[0];
assign g1 = a[1] & b[1];
assign g2 = a[2] & b[2];
assign g3 = a[3] & b[3];
// Carry calculations using Carry Lookahead logic
assign c1 = g0 | (p0 & c_0);
assign c2 = g1 | (p1 & g0) | (p1 & p0 & c_0);
assign c3 = g2 | (p2 & g1) | (p2 & p1 & g0) | (p2 & p1 & p0 & c_0);
assign c4 = g3 | (p3 & g2) | (p3 & p2 & g1) | (p3 & p2 & p1 & g0) | (p3 & p2 & p1 & p0 &
c_0);
// Sum calculations
assign sum[0] = p0 ^ c_0;
assign sum[1] = p1 ^ c1;
assign sum[2] = p2 ^ c2;
assign sum[3] = p3 ^ c3;
```

```
// Assign carry-out
assign c_4 = c4;
endmodule
//Testbench
module CLA_4bit_tb();
parameter WIDTH = 3; // Set the width of the operands
// Declare the registers for the two addends
reg [WIDTH-1:0] r_ADD_1 = 0;
reg [WIDTH-1:0] r_ADD_2 = 0;
// Declare the wire for the result
wire [WIDTH:0] w_RESULT;
// Instantiate the Carry Lookahead Adder
CLA_4bit #(.WIDTH(WIDTH)) carry_lookahead_inst (
.a(r_ADD_1),
.b(r_ADD_2),
.sum(w_RESULT[WIDTH-1:0]), // Connect the sum to the lower bits of w_RESULT
.c_4(w_RESULT[WIDTH]) // Connect the carry-out to the MSB of w_RESULT
);
initial begin
// Apply test vectors with a delay of 10 time units
#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;
r_ADD_2 = 3'b111;
#10;
// Add more test cases if needed
```

end

endmodule

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 inXilinx 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 andsynthesized using Xilinx vivado 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 Xilinx vivado tools.

5.2 Software tools Requirement

Equipment's: Software's: Xilinx vivado

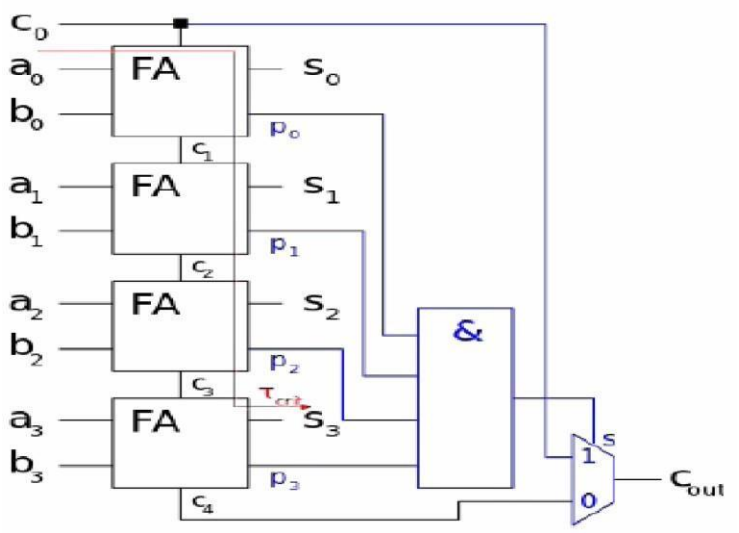
5.3 Pre Lab-Questions

(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?
2.

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

Verilog code

```
// 4-bit Carry Skip Adder module
module carry_skip_4bit(
input [3:0] a, // First 4-bit input
input [3:0] b, // Second 4-bit input
input cin, // Carry-in input
output [3:0] sum, // 4-bit sum output
output cout // Carry-out output
);
wire [3:0] p; // Propagate signals
wire c0; // Carry from ripple carry adder
wire bp; // Block propagate signal
// Instantiate ripple carry adder
ripple_carry_4_bit rca1 (.a(a),.b(b),.cin(cin,.sum(sum),.cout(c0));
// Instantiate propagate signal calculation
propagate_p p1 (.a(a),.b(b),.p(p),.bp(bp));
// Multiplexer to select between the carry-out of RCA and the input carry
mux2X1 m0 (.a(c0),.b(cin),.sel(bp),.y(cout));
endmodule

// Propagate calculation module
module propagate_p(
input [3:0] a,
input [3:0] b,
output [3:0] p,
output bp
);
assign p = a ^ b; // Generate propagate signals (p0, p1, p2, p3)
assign bp = &p; // Block propagate signal (1 if all propagate bits are 1)
endmodule
```

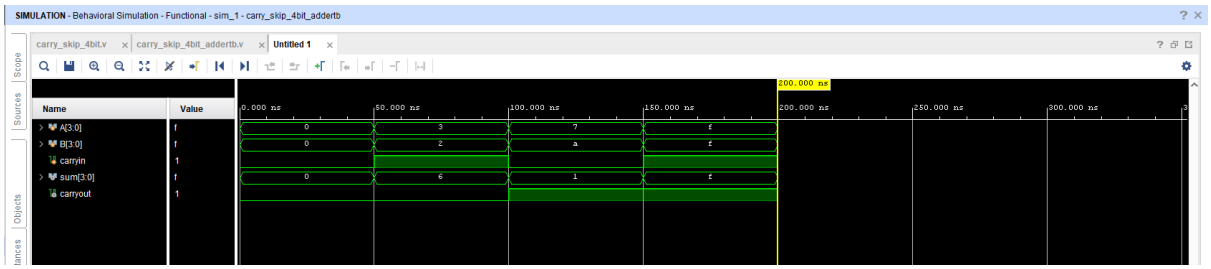
```
// 4-bit Ripple Carry Adder module
module ripple_carry_4_bit(
input [3:0] a,
input [3:0] b,
input cin,
output [3:0] sum,
output cout
);
wire c1, c2, c3; // Intermediate carry wires
// Instantiate four full adders
full_adder fa0 (.a(a[0]),.b(b[0]),.cin(cin),.sum(sum[0]),.cout(c1));
full_adder fa1 (.a(a[1]),.b(b[1]),.cin(c1),.sum(sum[1]),.cout(c2));
full_adder fa2 (.a(a[2]),.b(b[2]),.cin(c2),.sum(sum[2]),.cout(c3));
full_adder fa3 (.a(a[3]),.b(b[3]),.cin(c3),.sum(sum[3]),.cout(cout));
endmodule
```

Testbench

```
module carry_skip_4bit_adder_tb;
// Declare wires and registers
wire [7:0] Y; // Output sum
wire carryout; // Carry out
reg [7:0] A, B; // Input operands
reg carryin; // Carry in
// Instantiate the Carry Skip Adder
carry_skip_4bit csa1 (
.sum(Y), // Sum output
.cout(carryout), // Carry out
.a(A), // First input
.b(B), // Second input
.cin(carryin) // Carry in
);
```

```
// Initial block for stimulus
initial begin
// Display the header
$display("RSLT\tA\t\tB\t\tCYIN\tCYOUT\tSUM");
// Test cases
A = 8'b00000000; B = 8'b00000000; carryin = 0; #50; // Test case 1
A = 8'b00000011; B = 8'b00000010; carryin = 1; #50; // Test case 2
A = 8'b00000111; B = 8'b00001010; carryin = 0; #50; // Test case 3
A = 8'b00001111; B = 8'b00001111; carryin = 1; #50; // Test case 4
A = 8'b11111111; B = 8'b00111000; carryin = 0; #50; // Test case 5
A = 8'b11111111; B = 8'b11111111; carryin = 1; #50; // Test case 6
// Display results for verification
if (carryout == 1) begin
$display("PASS\t%h\t%h\t%d\t=\t%d\t%h", A, B, carryin, carryout, Y);
end else begin
$display("FAIL\t%h\t%h\t%d\t=\t%d\t%h", A, B, carryin, carryout, Y);
end
end
// Enable the waveform dump for simulation
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 Xilinx vivado tools.

Lab Experiment #6

Realization of Multiplier-1

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

6.2 Software tools Requirement:

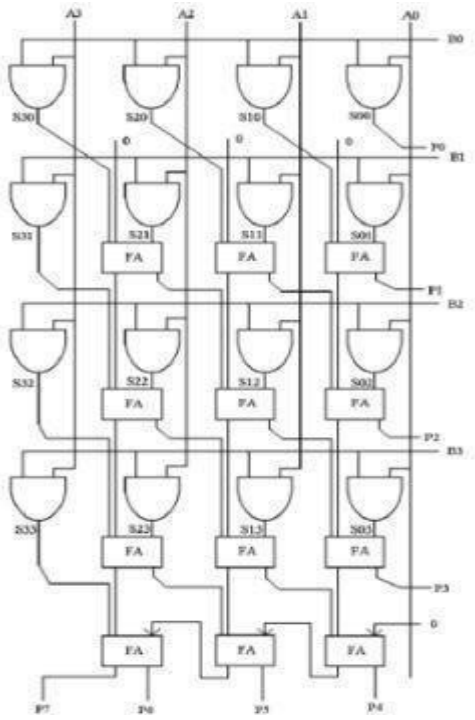
Software's: Xilinx vivado

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



Verilog code

```
module braun(a, b, p);
input [3:0] a; // 4-bit input a
input [3:0] b; // 4-bit input b
output [7:0] p; // 8-bit output product p
wire w0, w1, w2, w3, w4, w5, w6, w7;
wire w8, w9, w10, w11, w12, w13, w14, w15;
wire w16, w17, w18, w19, w20, w21, w22;
wire w23, w24, w25, w26, w27, w28, w29;
wire w30, w31;
// AND gates to generate partial products
and g0 (w0, a[0], b[0]);
and g1 (w1, a[1], b[0]);
and g2 (w2, a[2], b[0]);
and g3 (w3, a[3], b[0]);
and g4 (w4, a[0], b[1]);
and g5 (w5, a[1], b[1]);
and g6 (w6, a[2], b[1]);
and g7 (w7, a[3], b[1]);
and g8 (w8, a[0], b[2]);
and g9 (w9, a[1], b[2]);
and g10(w10, a[2], b[2]);
and g11(w11, a[3], b[2]);
and g12(w12, a[0], b[3]);
and g13(w13, a[1], b[3]);
and g14(w14, a[2], b[3]);
and g15(w15, a[3], b[3]);
// Full adders for addition of partial products
fa_df f0 (w16, w17, w0, w4, 1'b0); // Column 1
fa_df f1 (w18, w19, w1, w5, 1'b0); // Column 2
fa_df f2 (p[1], w20, w2, w6, 1'b0); // Column 3
fa_df f3 (w21, w22, w3, w7, w16); // Column 4
fa_df f4 (w23, w24, w8, w12, w17); // Column 5
```

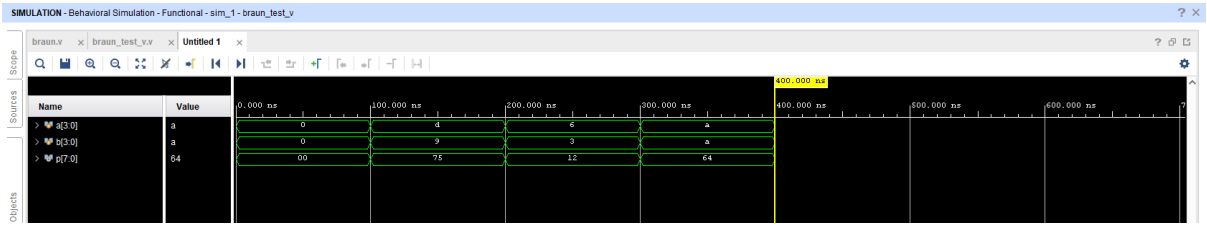
```
fa_df f5 (p[2], w25, w9, w13, w19); // Column 6
fa_df f6 (w26, w27, w10, w14, w20); // Column 7
fa_df f7 (p[3], w28, w11, w15, w22); // Column 8
fa_df f8 (p[4], w29, w12, w23, w24); // Carry addition
fa_df f9 (p[5], p[6], w21, w26, w27); // More additions
fa_df f10(p[7], w30, w28, w29, 1'b0); // Final addition for MSB
endmodule

// Full adder module definition (assumed)
module fa_df(sum, cout, a, b, cin);
input a, b, cin;
output sum, cout;
wire w1, w2, w3;
assign sum = a ^ b ^ cin; // Sum is the XOR of inputs
assign w1 = a & b; // Intermediate carry
assign w2 = a & cin; // Intermediate carry
assign w3 = b & cin; // Intermediate carry
assign cout = w1 | w2 | w3; // Carry out is the OR of intermediate carries
endmodule
```

Testbench

```
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 = 4'b0000; // Test case 1
b = 4'b0000;
#100; // Wait for some time to observe outputs
```

```
// Test case 2
a = 4'b1101; // 13 in decimal
b = 4'b1001; // 9 in decimal
#100; // Wait for outputs
// Test case 3
a = 4'b0011; // 3 in decimal
b = 4'b0101; // 5 in decimal
#100; // Wait for outputs
// Test case 4
a = 4'b1111; // 15 in decimal
b = 4'b1111; // 15 in decimal
#100; // Wait for outputs
// Test case 5
a = 4'b0001; // 1 in decimal
b = 4'b0010; // 2 in decimal
#100; // Wait for outputs
// Test case 6
a = 4'b0101; // 5 in decimal
b = 4'b0011; // 3 in decimal
#100; // Wait for outputs
// Finish simulation
$finish;
end
// Optional: Monitor the outputs
initial begin
$monitor("Time: %0t | A: %b | B: %b | P: %b", $time, a, b, p);
end
endmodule
```

Simulation Waveform



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 Xilinx vivado tools.

Lab Experiment #7

Realization of Multiplier-II

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

7.2 Software tools Requirement

Software's: Xilinx vivado

7.3 Pre Lab-Questions

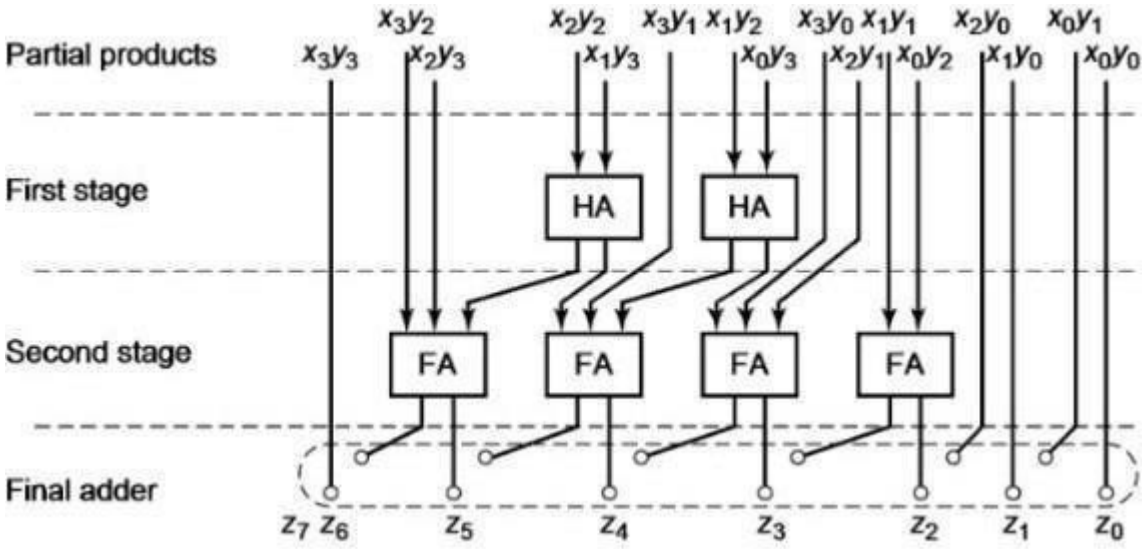
(Write pre lab Q & A in an A4 sheet)

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

Verilog Code:

```
module wallace_tree_mul(
input [3:0] a,
input [3:0] b,
output [7:0] z
);
wire [15:0] p; // Intermediate products
wire [32:0] w; // Internal wires for addition
// Generate partial products
and g0 (p[0], a[0], b[0]);
and g1 (p[1], a[0], b[1]);
and g2 (p[2], a[0], b[2]);
and g3 (p[3], a[0], b[3]);
and g4 (p[4], a[1], b[0]);
and g5 (p[5], a[1], b[1]);
and g6 (p[6], a[1], b[2]);
and g7 (p[7], a[1], b[3]);
and g8 (p[8], a[2], b[0]);
and g9 (p[9], a[2], b[1]);
and g10(p[10], a[2], b[2]);
and g11(p[11], a[2], b[3]);
and g12(p[12], a[3], b[0]);
and g13(p[13], a[3], b[1]);
and g14(p[14], a[3], b[2]);
and g15(p[15], a[3], b[3]);
// Level 1: Half adders and full adders
ha_gate h0 (w[0], w[1], p[5], p[4]); // Partial product from a[1] * b[1] and a[1] * b[0]
ha_gate h1 (w[2], w[3], p[7], p[6]); // From a[1] * b[3] and a[1] * b[2]
fa_df f0 (w[4], w[5], p[3], p[2], 1'b0); // From a[0] * b[3] and a[0] * b[2]
// Second level addition
fa_df f1 (w[6], w[7], w[1], p[1], 1'b0);
fa_df f2 (w[8], w[9], w[3], p[8], 1'b0);
fa_df f3 (w[10], w[11], w[5], p[9], 1'b0);
```

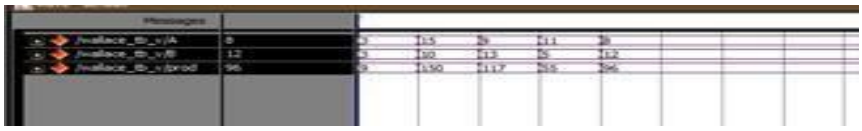
```
// Final sum
fa_df f4 (z[1], w[12], w[4], p[10], 1'b0);
fa_df f5 (z[2], w[13], w[6], w[11], 1'b0);
fa_df f6 (z[3], w[14], w[8], p[12], 1'b0);
fa_df f7 (z[4], z[5], w[9], p[13], 1'b0);
fa_df f8 (z[5], z[6], w[10], p[14], 1'b0);
// Last output
assign z[0] = p[0];
assign z[6] = w[12];
assign z[7] = w[14];
// Carry-out
assign z[7] = w[14]; // last carry
endmodule
```

Testbench

```
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 = 4'b0000; // Initialize a to 0
b = 4'b0000; // Initialize b to 0
// Wait 100 ns for global reset to finish
#100;
// Apply test cases
a = 4'b1101; // a = 13
b = 4'b1001; // b = 9
#100; // Wait for 100 ns
```

```
a = 4'b0011; // a = 3
b = 4'b0101; // b = 5
#100; // Wait for 100 ns
a = 4'b1111; // a = 15
b = 4'b0001; // b = 1
#100; // Wait for 100 ns
a = 4'b1010; // a = 10
b = 4'b1100; // b = 12
#100; // Wait for 100 ns
// You can add more test cases as needed
// Finish simulation
$finish;
end
// Monitor the outputs
initial begin
$monitor("Time: %t | A: %b | B: %b | Result (Z): %b", $time, a, b, z);
end
endmodule
```

Simulation Waveform:



7.5 Post Lab:

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

7.6 Result:

Thus, the design of 4-bit Wallace tree multiplier circuit was simulated in Verilog andsynthesized using Xilinx vivado tools

Lab Experiment #8

Realization of Memory

8.1 Objective:

Design memory using Verilog and Simulate using Xilinx Tool.

8.2 Software tools Requirement:

Software's: Xilinx vivado

8.3 Prelab:

1.What are the different types of memory? Compare its performance.

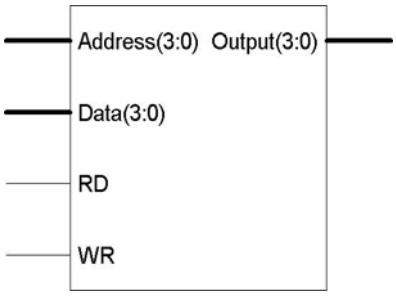
2.Write the difference between static and dynamic memory.

3.Define port for 32*64k RAM memory.

8.4.1 Problem Statement 1:

1.Design a 16X4 RAM Memory.

Logic Diagram:



Verilog code:

```
Module RAM (
input [3:0] Data,
input RD,
input WR,
input [3:0] Address,
output reg [3:0] Output
);
// RAM declaration: 16 locations of 4 bits each
reg [3:0] RAM [15:0];
always @(posedge WR or posedge RD) begin
if (WR) begin
// Write data to RAM at specified address
RAM[Address] <= Data;
end
if (RD) begin
// Read data from RAM at specified address
Output <= RAM[Address];
end
end
endmodule
```

Testbench

```
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)
Memory_Design uut (.Data(Data),.RD(RD),.WR(WR),.Address(Address),.Output(Output));
```

```
initial begin
// Write data to RAM
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;
// Read data from RAM
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;

// End simulation

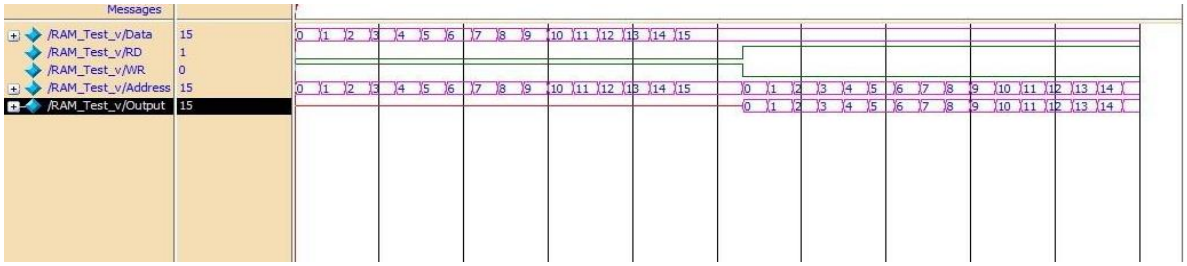
#100;

$finish;

end

endmodule

Simulation Waveform:

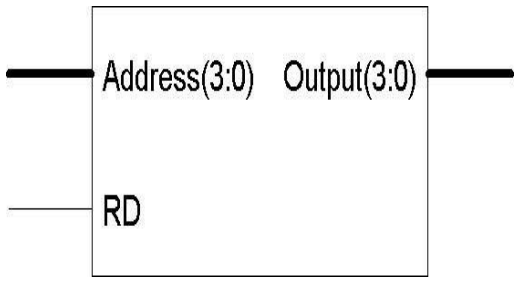


Statement 2:

8.4.2 Problem Design 16X4 ROM Memory.

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

Block Diagram



Verilog code:

```
module ROM_16x4(
input [3:0] Address, // 4-bit address input
input RD, // Read enable signal
output reg [3:0] Output // 4-bit output data
);
// ROM declaration with 16 entries of 4 bits each
reg [3:0] ROM [15:0];
// Initialize ROM contents
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; // Ensure this address is initialized correctly
ROM[4'b1110] = 4'b0001;
ROM[4'b1111] = 4'b0000;
end
// Read operation
always @(RD, Address) begin
if (RD) begin
Output = ROM[Address]; // Output data from the ROM
end
else begin
Output = 4'bzzzz; // Set to high impedance if not reading
```

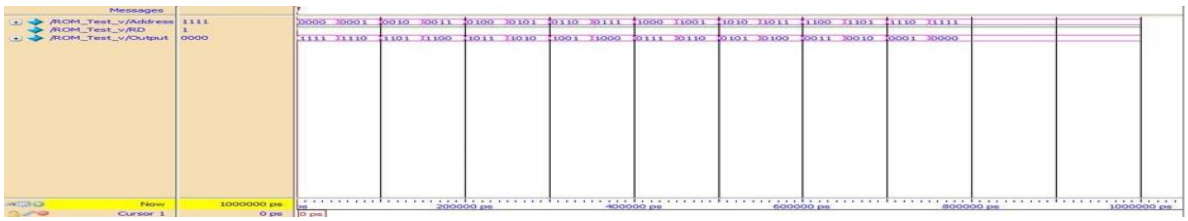
```
end
end
endmodule
```

Testbench

```
module ROM_16x4_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
RD = 1; // Set RD to 1 for read operations
Address = 4'b0000; #50;
$display("Address: %b, Output: %b", Address, Output);
Address = 4'b0001; #50;
$display("Address: %b, Output: %b", Address, Output);
Address = 4'b0010; #50;
$display("Address: %b, Output: %b", Address, Output);
Address = 4'b0011; #50;
$display("Address: %b, Output: %b", Address, Output);
Address = 4'b0100; #50;
$display("Address: %b, Output: %b", Address, Output);
Address = 4'b0101; #50;
$display("Address: %b, Output: %b", Address, Output);
Address = 4'b0110; #50;
$display("Address: %b, Output: %b", Address, Output);
Address = 4'b0111; #50;
$display("Address: %b, Output: %b", Address, Output);
Address = 4'b1000; #50;
```

```
$display("Address: %b, Output: %b", Address, Output);
Address = 4'b1001; #50;
$display("Address: %b, Output: %b", Address, Output);
Address = 4'b1010; #50;
$display("Address: %b, Output: %b", Address, Output);
Address = 4'b1011; #50;
$display("Address: %b, Output: %b", Address, Output);
Address = 4'b1100; #50;
$display("Address: %b, Output: %b", Address, Output);
Address = 4'b1101; #50;
$display("Address: %b, Output: %b", Address, Output);
Address = 4'b1110; #50;
$display("Address: %b, Output: %b", Address, Output);
Address = 4'b1111; #50;
$display("Address: %b, Output: %b", Address, Output);
// End of test
$finish;
end
endmodule
```

Simulation Waveform:



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 # 10

Dynamic NAND gate

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

10.2 Software required: Hspice

10.3 Pre-lab Questions

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

10.4 Design two input dynamic NAND Gate Circuit Diagram:

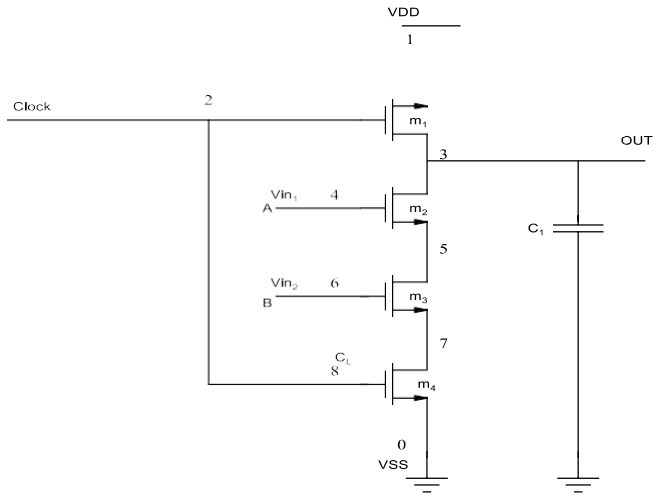
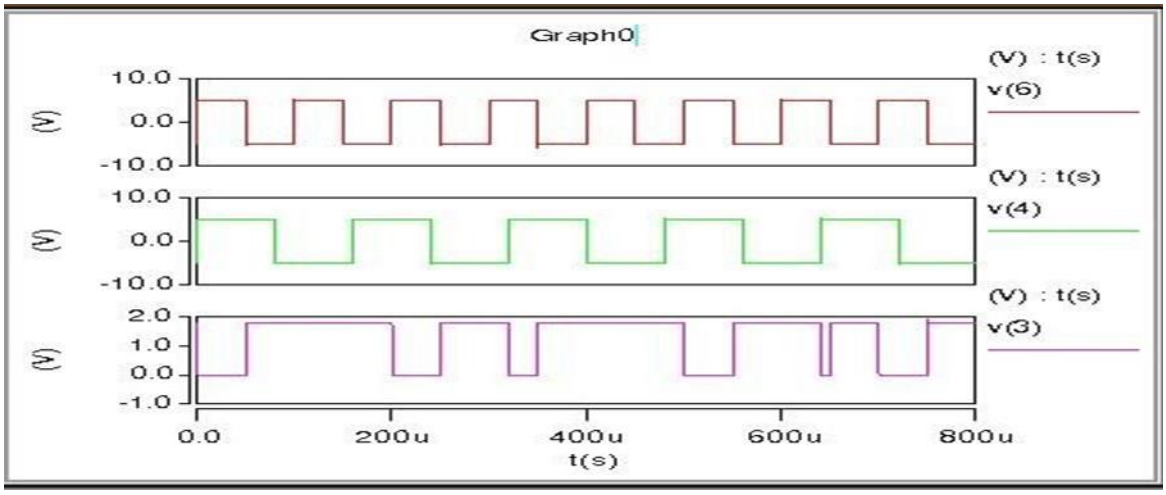


Figure 1. Two input dynamic NAND gate

10.5 Spice Netlist:

```
.option post
.include c:\synopsys\tsmc018.lib    * Include TSMC 180nm technology library
vdd 1 0 dc 1.8                      * Supply voltage (1.8V)

* Input pulses for A, B, and CLOCK signals
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 for precharge

* MOSFETs (using TSMC 180nm technology)
m1 3 2 1 1 cmosp w=4u l=180n        * PMOS transistor (pull-up network)
m2 3 4 5 5 cmosn w=2u l=180n        * NMOS transistor (first part of pull-down network)
m3 5 6 7 7 cmosn w=2u l=180n        * NMOS transistor (second part of pull-down
network)
m4 7 8 0 0 cmosn w=2u l=180n        * NMOS transistor (third part of pull-down network)

* Load capacitor to observe dynamic performance
c1 3 0 100p                         * Capacitor at output node (vary for dynamic performance)

* Simulation parameters
.tran 100u 800u                     * Transient analysis from 0 to 800us with 100us step size

* Plot signals
.plot v(3) v(4) v(6)                * Plot output (v(3)) and input signals (v(4), v(6))

.end                                * End of netlist
```

OUTPUT: N TYPE DYNAMIC – NAND GATE



10.6 Netlist:

.option post

* Include TSMC Model Library

.include c:\synopsys\tsmc018.lib

* TSMC018 CMOS Model Parameters (BSIM3 Version 3.1)

* Power Supply

vdd 1 0 dc 1.8V

* Input Signals

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)

* MOSFETs

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

```
* Capacitor
c1 3 0 100p

* Analysis Commands
.tran 100u 800u
.plot v(3) v(4) v(6)

.end

* Dynamic NAND Gate
****** MOS Model Parameters - PMOS and NMOS Models ********
**********************************************************************
**

* NMOS Model
.model cmosn nmos level=49
+ tox=4.1e-9
+ xj=1e-7
+ vth0=0.366247
+ u0=273.809
+ rdsw=123.338
+ vsat=135501
+ ... (additional NMOS parameters as needed)

* PMOS Model
.model cmosp pmos level=49
+ tox=4.1e-9
+ xj=1e-7
+ vth0=-0.390601
+ u0=115.689
+ rdsw=198.321
+ vsat=113098
+ ... (additional PMOS parameters as needed)
```

```
**********************************************************************
**
* Operating Point Information
* Simulation Time = 0
.node voltages
vdd = 1.8V, vin1 = -5V, vin2 = -5V, vin3 = -5V, vin4 = -5V, ...
* (other node voltages can be listed here)

* Transient Analysis - Node Voltages at Specific Times
time          v(1)    v(2)    v(3)    ...
100.000u      1.8             1.2             ...
200.000u      1.8             1.2             ...
300.000u      1.8             1.2             ...
...
* Job Statistics Summary
* Total Memory Used = 161 kB
* Number of Nodes = 9, Elements = 10, MOSFETs = 4

* Analysis Summary
.analysis time      #points total_iter conv_iter
op point            0.03    1               32
transient           0.05    9               2016            560

* End of Netlist
```

10.7 Post Lab Questions:

1.Design the n-type dynamic NOR gate using the HSPICE.

2.Realize n –type dynamic AND gate using the HSPICE.

10.8 Result:

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