

**COLLEGE OF ENGINEERING, GUINDY
ANNA UNIVERSITY
CHENNAI 600025**

**EC5611 VLSI LABORATORY –SEMESTER VI
(R-2019)**

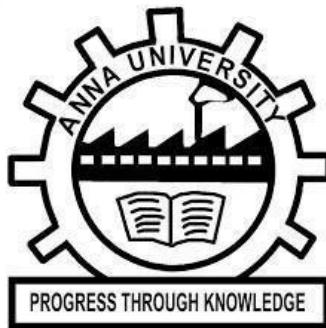
RECORD

SUBMITTED BY:

BHARATH KANNAN L

2021105006

In partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
IN
ELECTRONICS AND COMMUNICATION ENGINEERING



**ANNA UNIVERSITY
COLLEGE OF ENGINEERING GUINDY**

BONAFIDE CERTIFICATE

NAME: Bharath Kannan L

DEPT: B.E. Electronics and Communication Engineering

REGISTER NUMBER: 2021105006

It is certified that this is the bonafide record of the work done by the above-mentioned student in EC5611 – VLSI LABORATORY- SEMESTERVI (R-2019) during the period January 2024 - May 2024.

Signature of Lab-In-Charge

*Signature of the
Head of the Department*

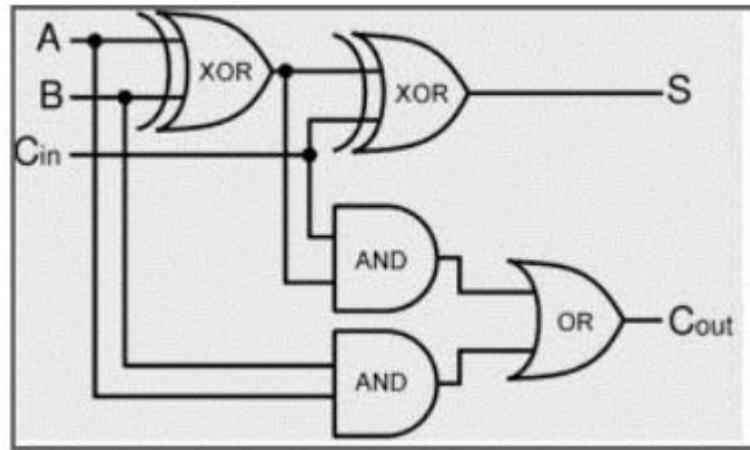
Submitted for the practical exam held on _____

Signature of Internal Examiner

INDEX

S.N O	DATE	TITLE OF THE EXPERIMENT	MARKS	SIGN
1		FULL ADDER WITHOUT CLOCK		
2		FULL ADDER WITH CLOCK		
3		ARITHMETIC LOGIC UNIT WITH AND WITHOUT CLOCK		
4		UP COUNTER USING PLL		
5		COMBINATIONAL CIRCUIT <ul style="list-style-type: none"> • DECODER • MULTIPLEXER • COMPARATOR 		
6		SEQUENTIAL CIRCUITS (D-FLIP FLOP) <ul style="list-style-type: none"> • BLOCKING • NON-BLOCKING 		
7		SEQUENCE DETECTOR (MOORE FSM)		
8		TIMING ANALYSIS		
		CADENCE EXPERIMENTS		
9		VOLTAGE TRANSFER CHARACTERISTICS <ul style="list-style-type: none"> • PMOS • NMOS 		
10		CMOS INVERTER		
11		COMMON SOURCE		

FULL ADDER CIRCUIT:



TRUTH TABLE:

A	B	C	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

EXPT NO

DATE:

FULL ADDER WITHOUT CLOCK

AIM:

To implement a four bit adder using Verilog and FPGA board.

REQUIREMENT:

SOFTWARE : Vivado 2022.2 , Waveforms

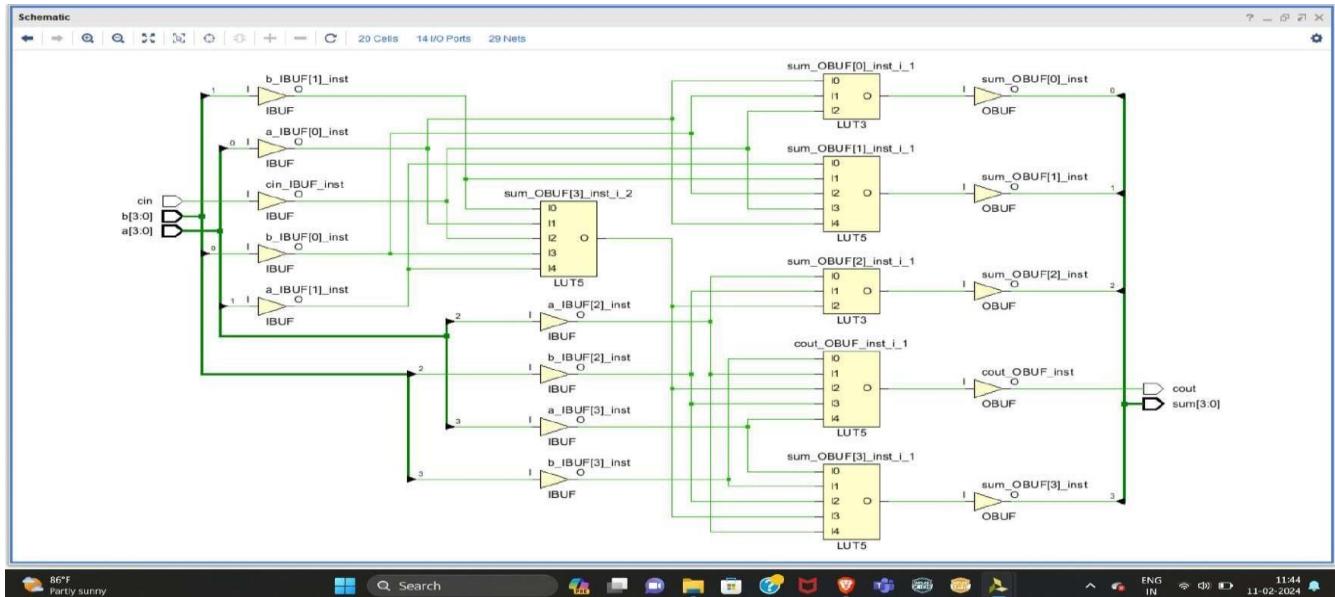
HARDWARE : Basys 3 , analog discovery

CODE:

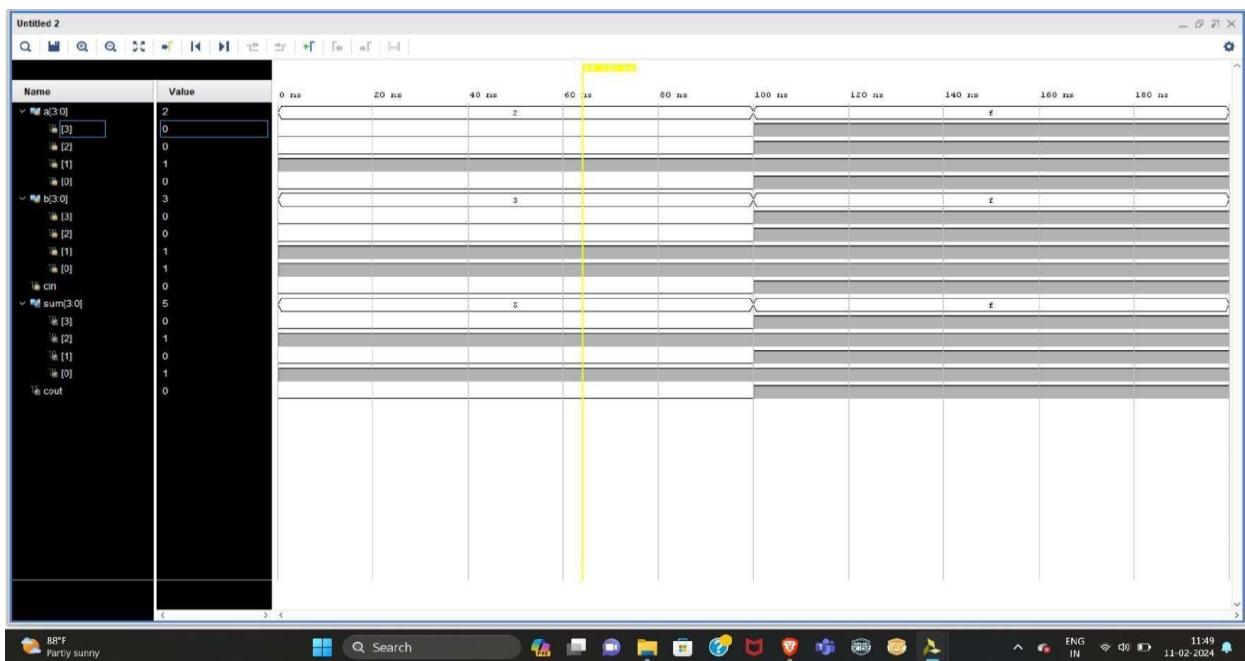
FULL ADDER:

```
module Adder_2(
    input a,
    input b,
    input c,
    output sum,
    output carry
);
reg sum,carry;4
always @(a or b or c)
begin
    sum <= a^b^c;
    carry <= (a&b) | (b&c) | (a&c);
end
endmodule
```

SCEMATIC:



BEHAVIORAL SIMULATION:



```

module Full_Adder(
    input [3:0] A,B,
    input Cin,
    output [3:0] Sum,
    output cout
);
    wire c1,c2,c3;
    Adder_2 fa0(A[0],B[0],Cin,Sum[0],c1);
    Adder_2 fa1(A[1],B[1],c1,Sum[1],c2);
    Adder_2 fa2(A[2],B[2],c2,Sum[2],c3);
    Adder_2 fa3(A[3],B[3],c3,Sum[3],cout);
Endmodule

```

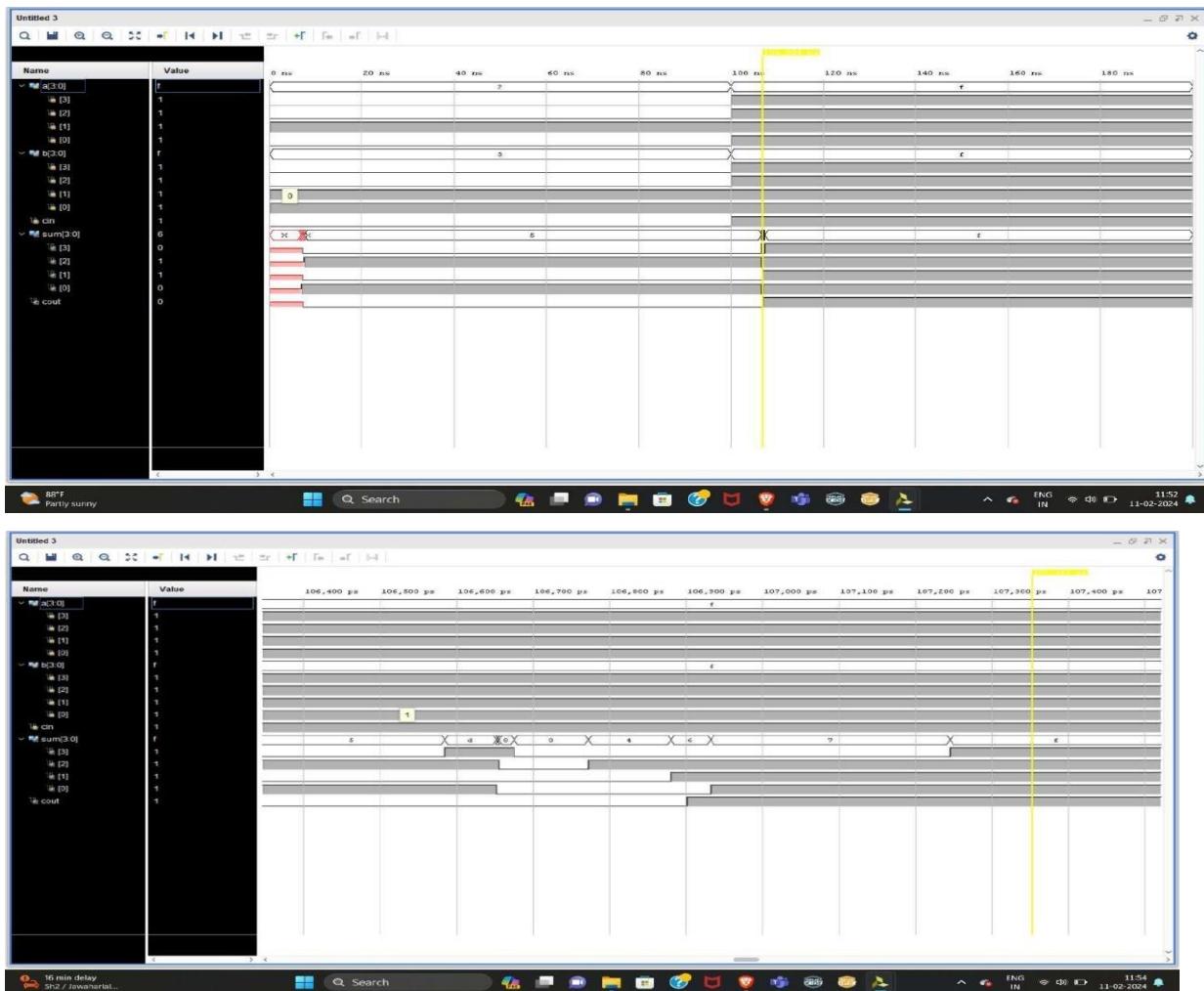
TESTBENCH

```

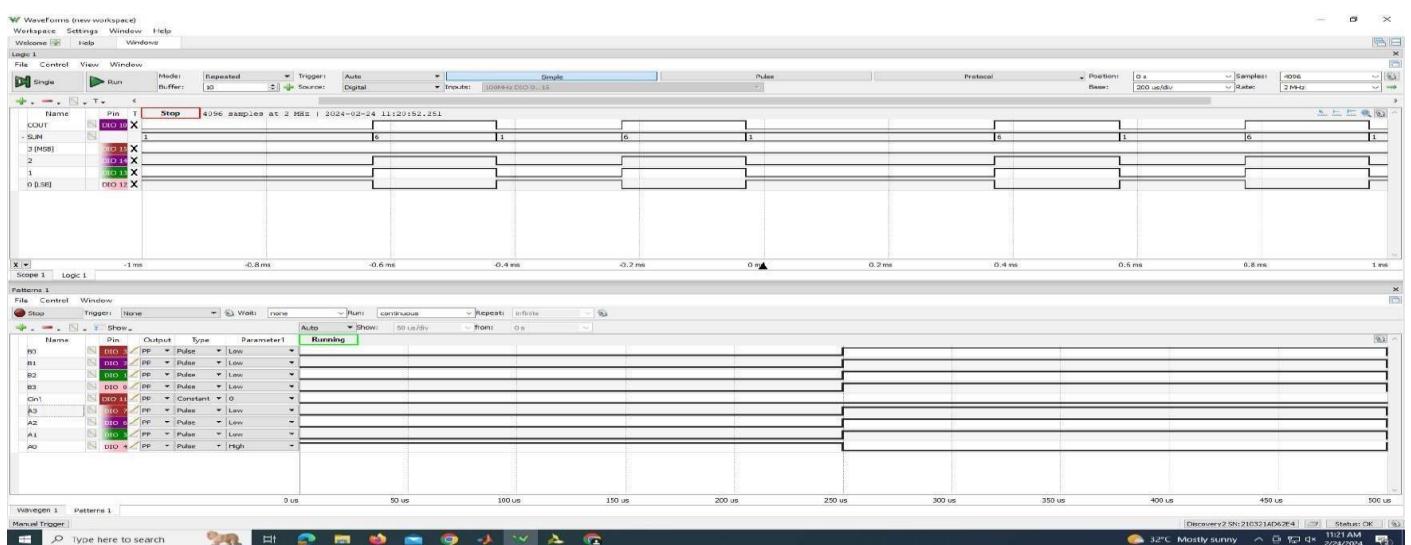
module Full_Adder_tb();
    reg[3:0] a;
    reg[3:0] b;
    reg cin;
    wire[3:0] sum;
    wire cout;
    Full_Adder uut(a,b,(cin,sum,cout));
    initial
    begin
        a=4'b0001;
        b=4'b0000;
        cin=1'b1;
        #10
        a=4'b0011; b=4'b0000; cin=1'b1;
        #10
        $finish;
    end
endmodule

```

POST TIMING IMPLEMENTATION:



WAVEFORMS OUTPUT:



INFERENCE:

The full adder circuit is implemented from single bit adder and carry from the previous sum as another input.

The adder circuit implements and generates sum and carry using the logical equations,

- Sum= $a \wedge b \wedge c$
- Carry= $ab + bc + ac$

RESULT:

Thus 4 bit adder is implemented using Verilog and the verification is done using simulation.

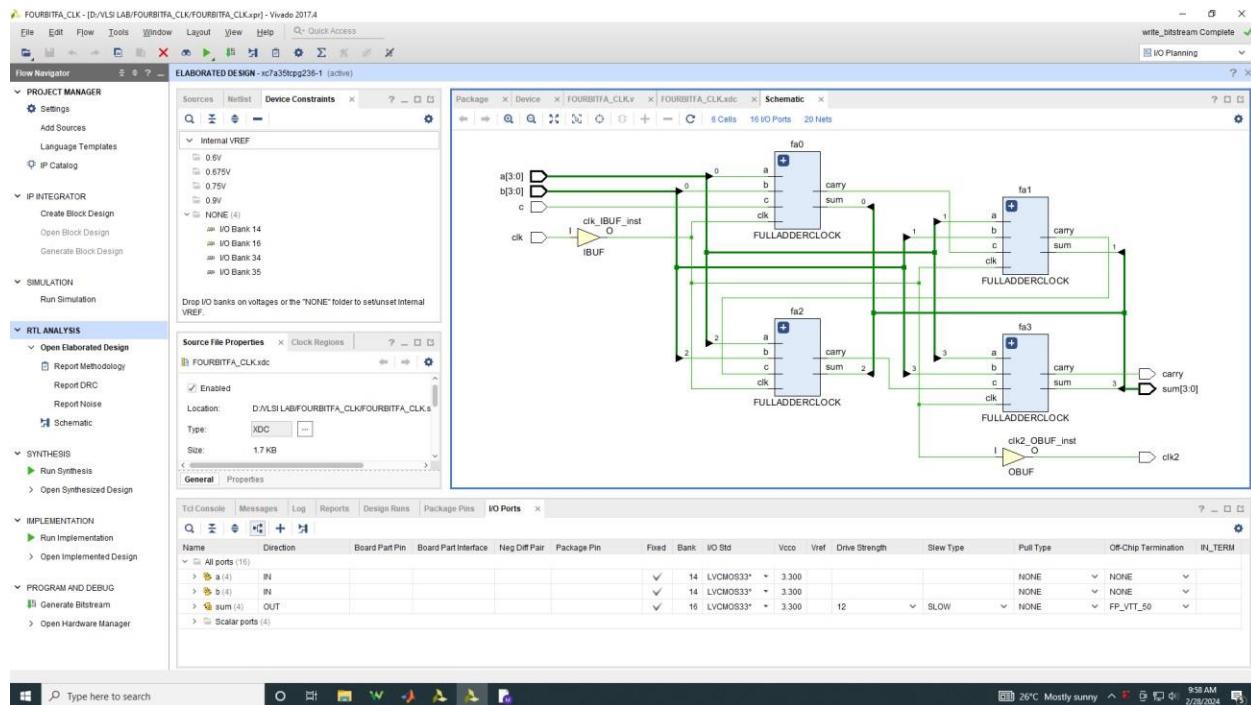
TABULATION:

INPUTS			OUTPUTS	
a	b	c	s	cy
A	2	0	C	0
D	3	1	1	1

AREA REPORT:

Resources	Utilization	Available	Utilization%
LUT	4	20800	0.02
FF	8	41600	0.02
IO	15	105	15.09

SCHEMATIC DIAGRAM:



EXPT NO:

DATE:

FULL ADDER WITH CLOCK

AIM:

To design and simulate full adder with clock using verilog and Basys 3 FPGA board and the waveforms to be observed using analog discovery.

REQUIREMENT:

SOFTWARE : Vivado 2022.2 , Waveforms

HARDWARE : Basys 3 , analog discovery

CODE:

FULL ADDER:

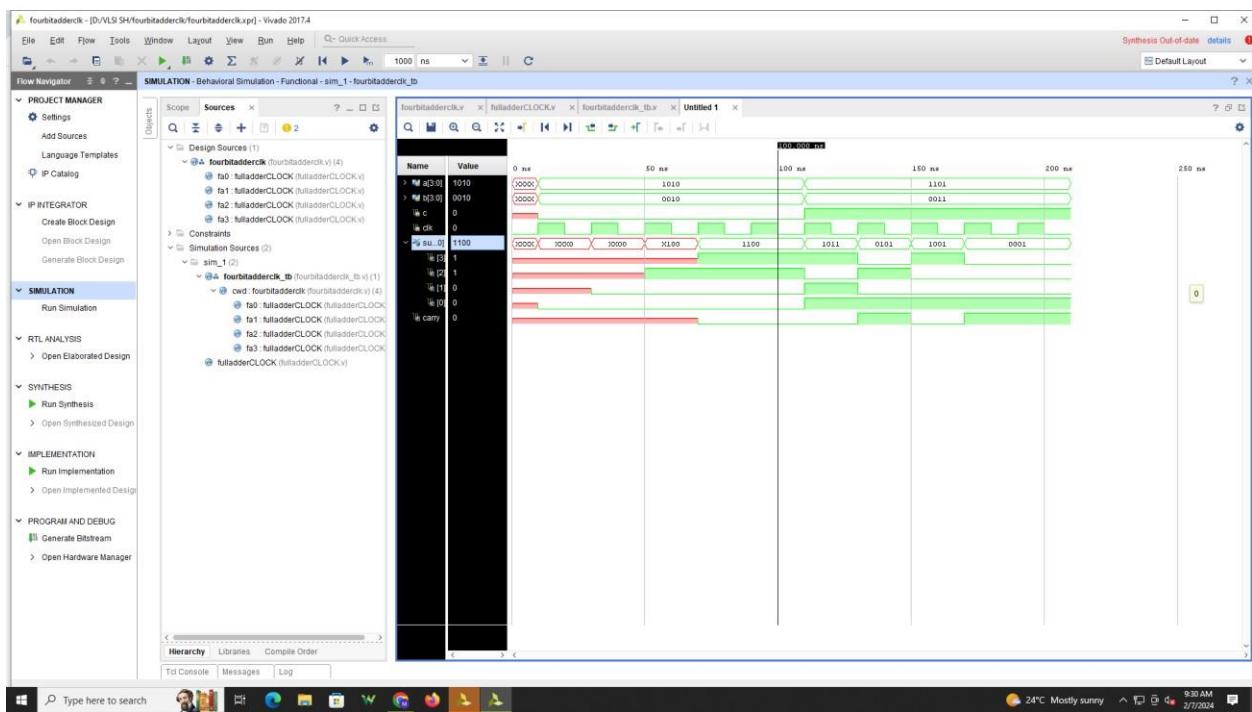
```
module fulladder( input a,b,c,clk, output reg s,cy);
always @(posedge clk)
begin
{cy , s}= a+b+c;
end
endmodule
```

FOURBIT FULLADDER:

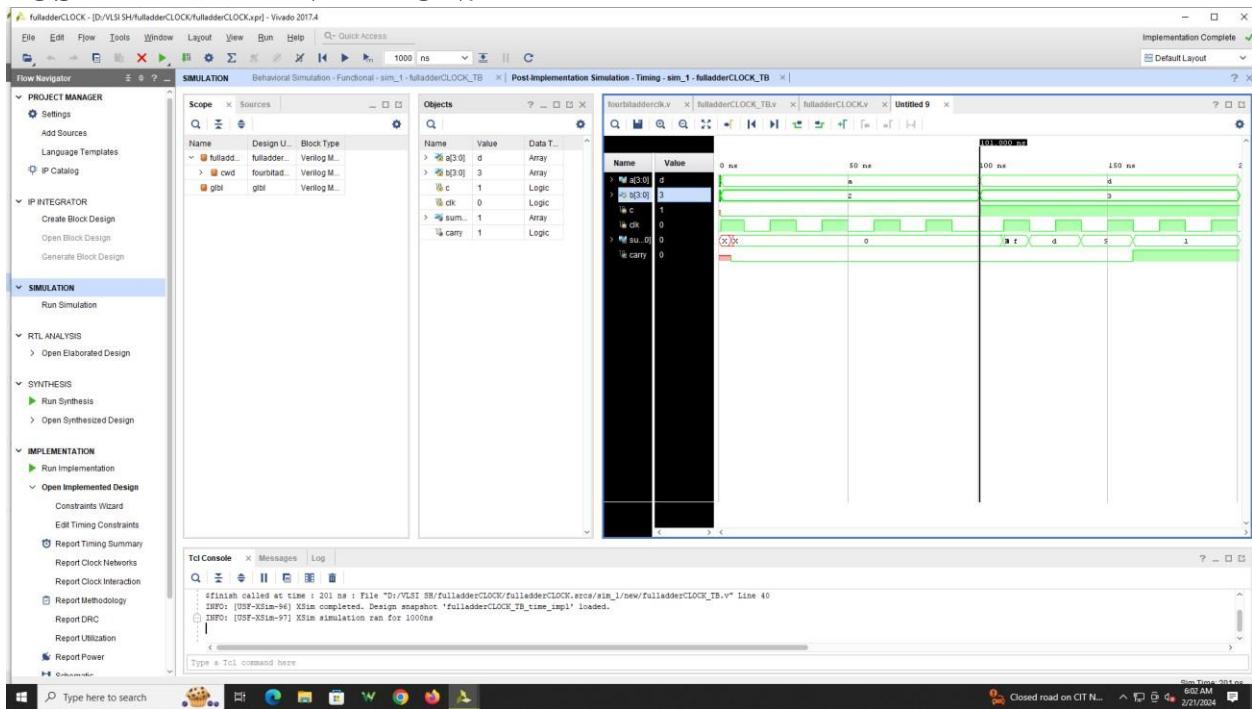
```
module fourbitadder( input [3:0]a,b, input c,clk, output reg [3:0]s, output reg cy);
wire c1,c2,c3
fulladder fa0 (a[0], b[0], c, clk, s[0], c1);
fulladder fa1 (a[1], b[1], c1, clk, s[1], c2);
fulladder fa2 (a[2], b[2], c2, clk, s[2], c3);
fulladder fa3 (a[3], b[3], c3, clk, s[3], cy);
endmodule
```

SIMULATION OUTPUT:

BEHAVIOURAL SIMULATION



POST IMPLEMENTATION:

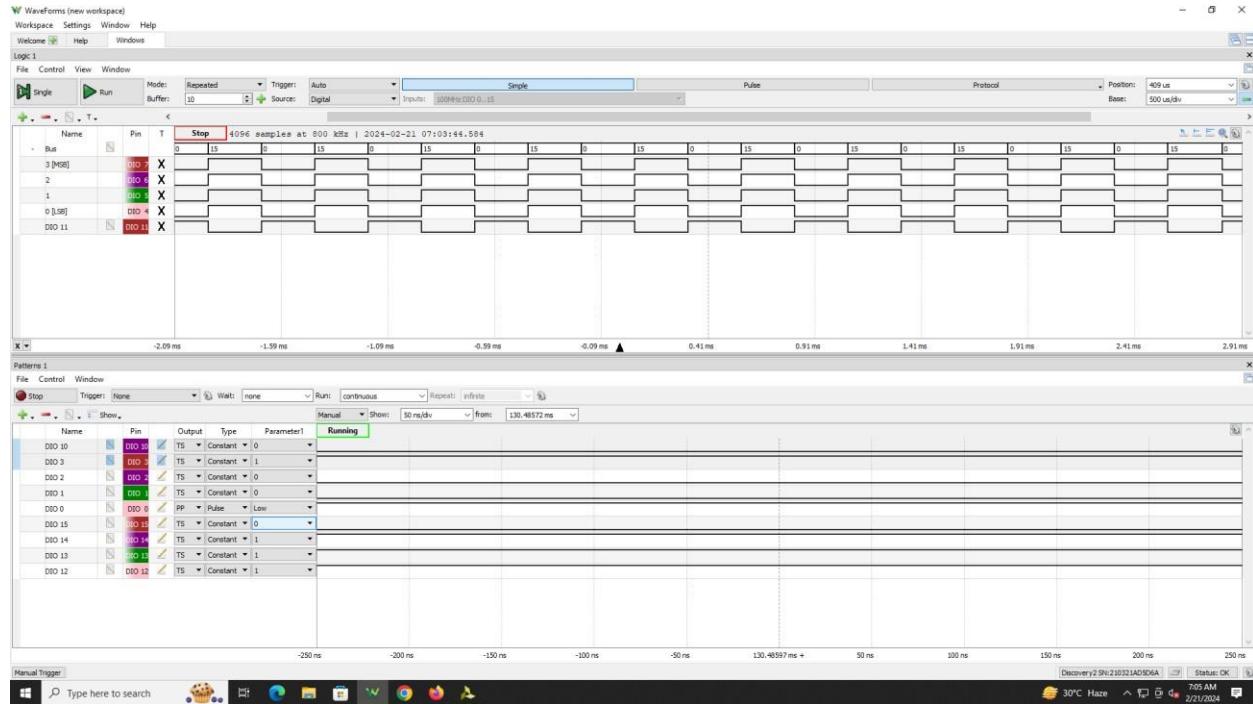


TESTBENCH:

```
module fourbitaddertb();
reg [3:0]a,b; reg c,clk; wire [3:0]s; wire cy;
fourbitadder cwd( a,b,c,clk,s,cy);
always #10 clk= ~clk;
initial begin
clk=0;
#10;
a=4'b1010;b=4'b0010;c=1'b0;
#100
a=4'b1101;b=4'b0011;c=1'b1;
#100 $finish;
end
endmodule
```

WAVEFORMS OUTPUT:

WITH INTERNAL CLOCK:



WITH EXTERNAL CLOCK:



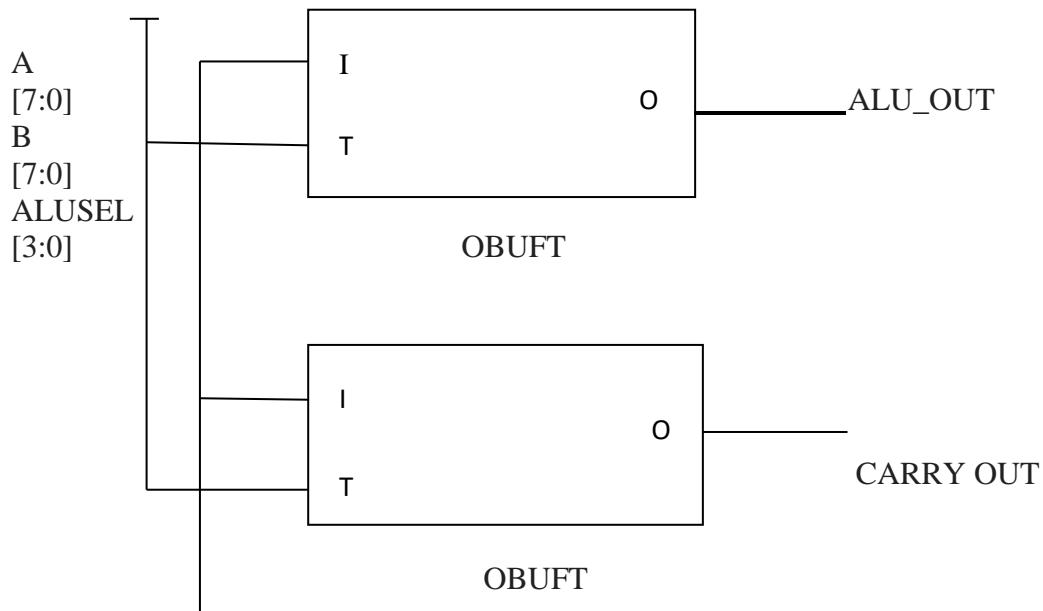
INFERENCE:

- From the post implementation , we can say that there exists some latency in the output due to both input and clock changes simultaneously so that there is some delay in the input to be stable.
- There exist some intermediate output
- We also observe the output using analog discovery using both internal and external clock.

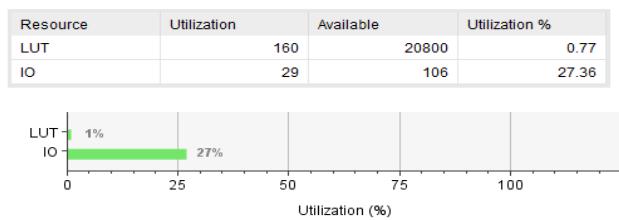
RESULT:

Thus we have designed and simulated arithmetic and logic units with clock using verilog and Basys 3 FPGA board and the waveforms is observed using analog discovery.

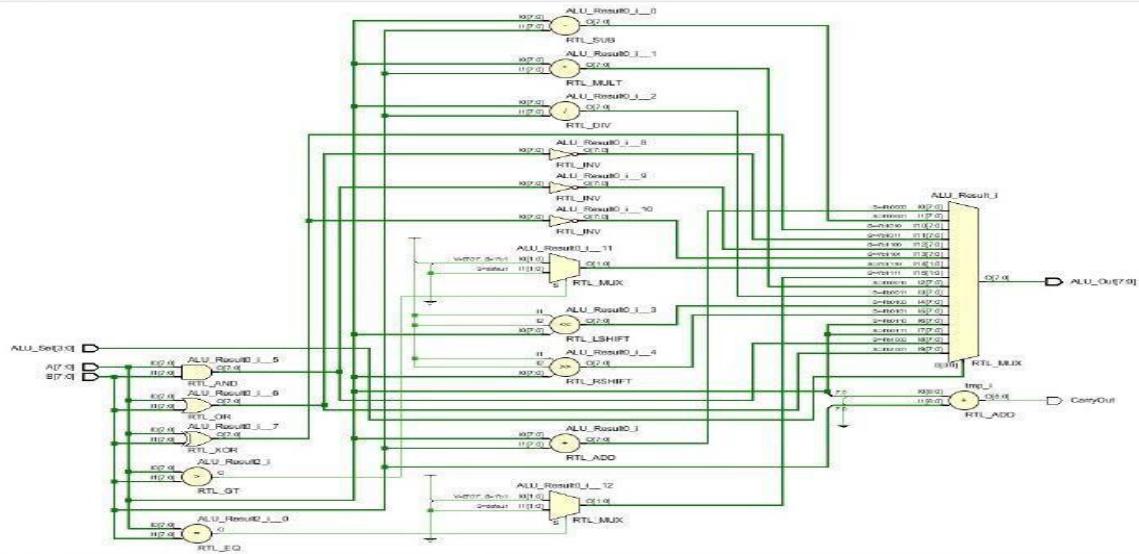
LOGIC DIAGRAM:



AREA REPORT:



SCHEMATIC WITHOUT CLOCK:



EXPT NO

DATE:

ALU WITH AND WITHOUT CLOCK

AIM:

To design and simulate arithmetic and logic units with and without clock using verilog and Basys 3 FPGA board and the waveforms to be observed using analog discovery.

REQUIREMENT:

SOFTWARE : Vivado 2022.2 , Waveforms

HARDWARE : Basys 3 , analog discovery

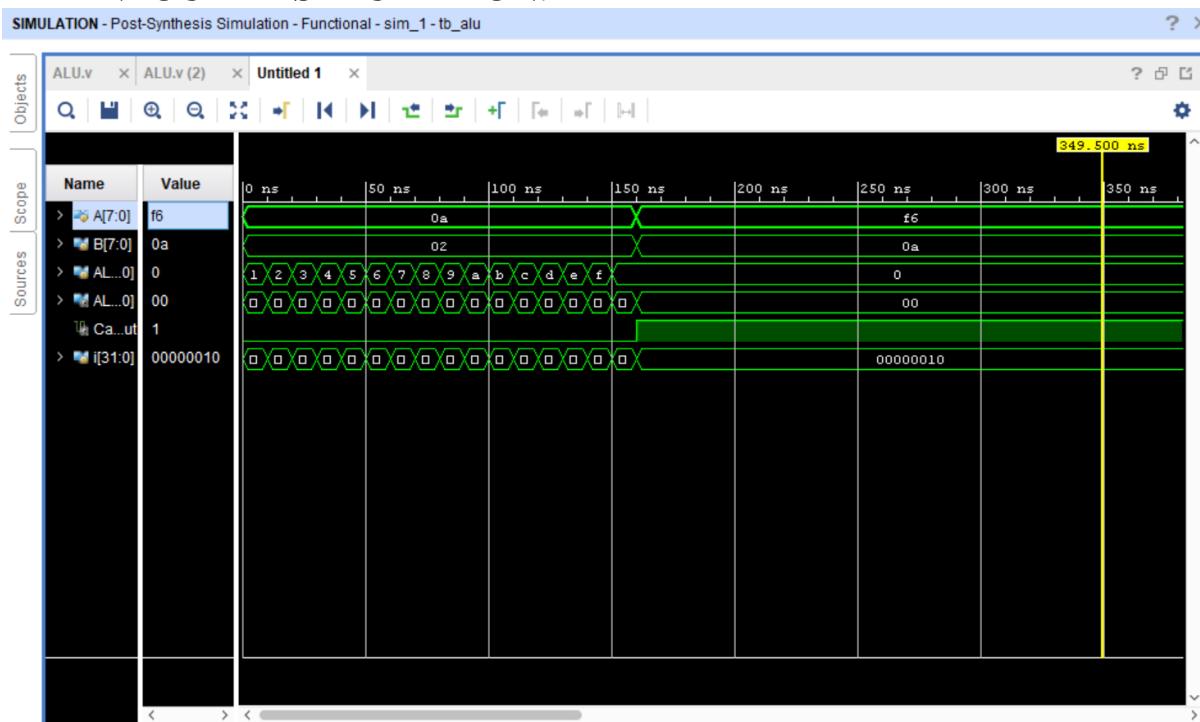
CODE:

ALU WITHOUT CLOCK:

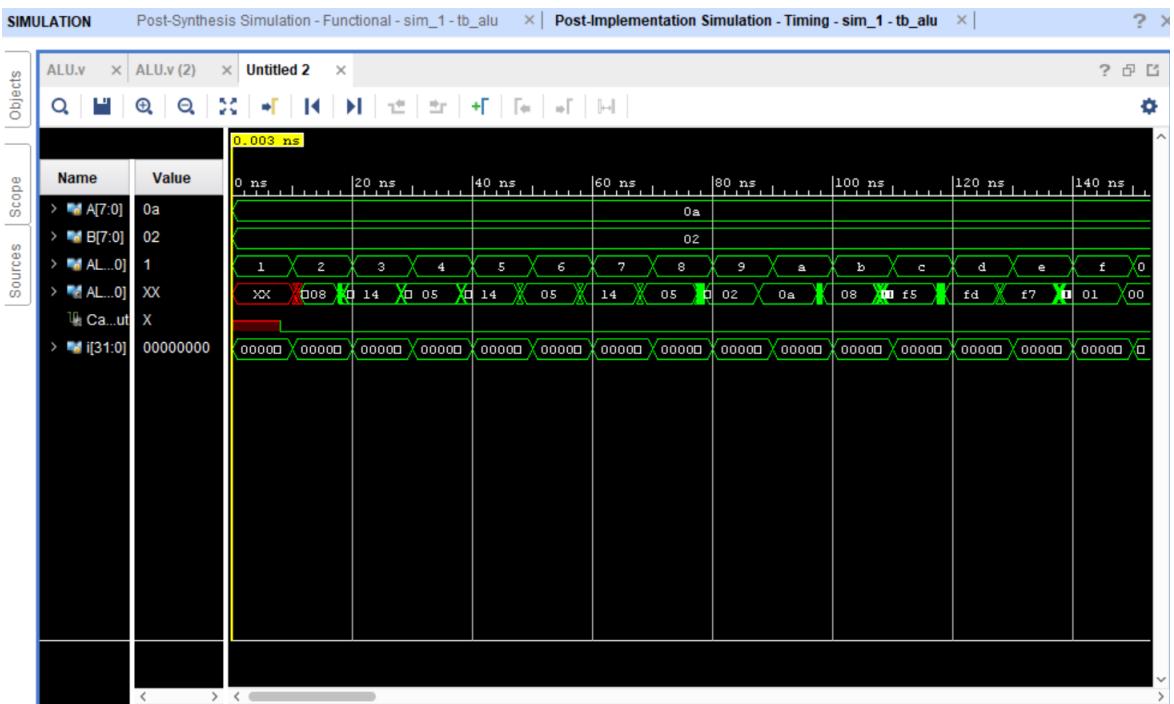
```
module ArithmeticLU(
    input [7:0] A,B,
    input [3:0] ALU_Sel,
    output [7:0] ALU_Out,
    output CarryOut
);
reg [7:0] ALU_Result;
wire [8:0] tmp;
assign ALU_Out = ALU_Result;
assign tmp = {1'b0,A} + {1'b0,B};
assign CarryOut = tmp[8];
begin
    case(ALU_Sel)
        4'b0000: // Addition
            ALU_Result = A + B ;
        4'b0001: // Subtraction
            ALU_Result = A - B ;
        4'b0010: // Multiplication
            ALU_Result = A * B;
        4'b0011: // Division
            ALU_Result = A/B;
        4'b0100: // Logical shift left
            ALU_Result = A<<1;
        4'b0101: // Logical shift right
            ALU_Result = A>>1;
        4'b0110: // Rotate left
            ALU_Result = {A[6:0],A[7]};
        4'b0111: // Rotate right
    endcase
end
endmodule
```

SIMULATION OUTPUT FOR WITHOUT CLOCK:

BEHAVIOURAL SIMULATION:



POST IMPLEMENTATION:



```

ALU_Result = {A[0],A[7:1]};
4'b1000: // Logical and
ALU_Result = A & B;
4'b1001: // Logical or
ALU_Result = A | B;
4'b1010: // Logical xor
ALU_Result = A ^ B;
4'b1011: // Logical nor
ALU_Result = ~(A | B);
4'b1100: // Logical nand
ALU_Result = ~(A & B);
4'b1101: // Logical xnor
ALU_Result = ~(A ^ B);
4'b1110: // Greater comparison
ALU_Result = (A>B)?8'd1:8'd0 ;
4'b1111: // Equal comparison
ALU_Result = (A==B)?8'd1:8'd0 ;
default: ALU_Result = A + B ;
endcase
end
endmodule

```

TESTBENCH:

```

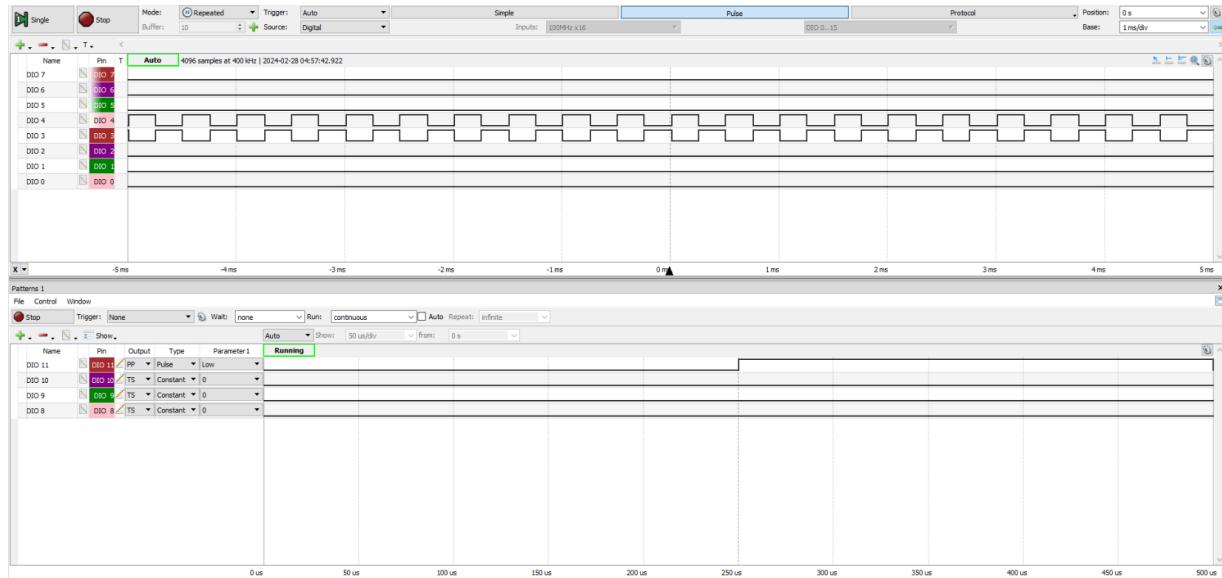
module tb_alu;
//Inputs
reg[7:0] A,B;
reg[3:0] ALU_Sel;
//Outputs
wire[7:0] ALU_Out;
wire CarryOut;
// Verilog code for ALU
integer i;
ArithmeticLU test_unit(
    A,B, // ALU 8-bit Inputs
    ALU_Sel,// ALU Selection
    ALU_Out, // ALU 8-bit Output
    CarryOut // Carry Out Flag
);
initial begin
    // hold reset state for 100 ns.
    A = 8'h03;
    B = 4'h02;
    ALU_Sel = 4'h0;

```

POWER ANALYSIS:



WAVEFORMS OUTPUT:



```

for (i=0;i<=15;i=i+1)
begin
#40;
ALU_Sel = ALU_Sel + 8'h1;
end
end
endmodule

```

ALU WITH CLOCK:

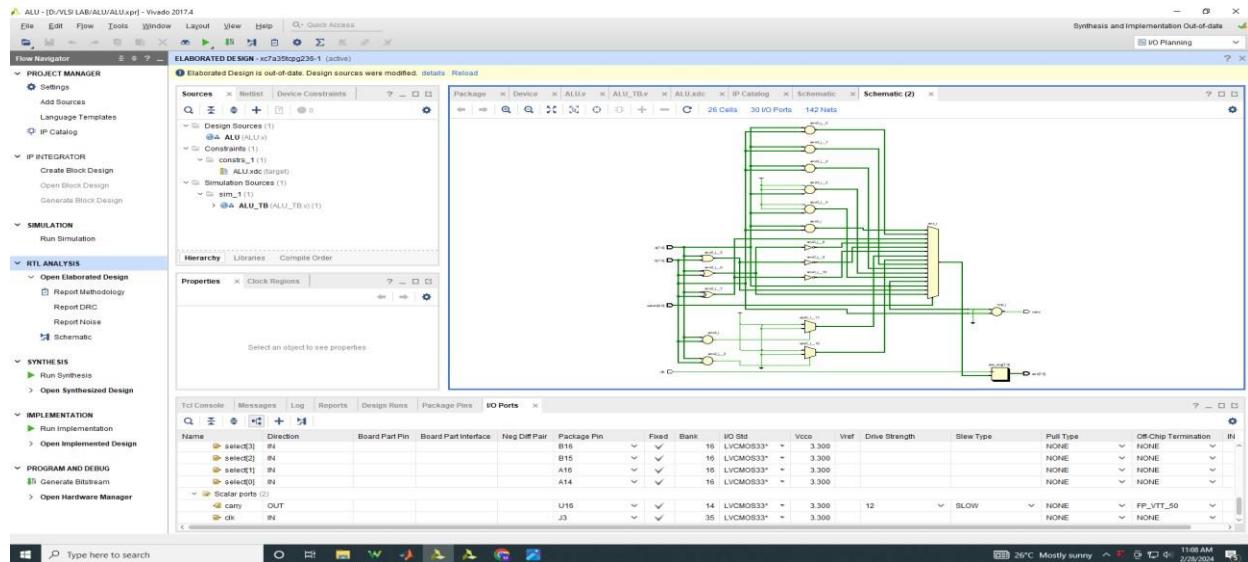
```

module ArithmeticLU(
    input [7:0] A,B,
    input [3:0] ALU_Sel,
    input clk,
    output [7:0] ALU_Out,
    output CarryOut
);
reg [7:0] ALU_Result;
wire [8:0] tmp;
assign ALU_Out = ALU_Result;
assign tmp = {1'b0,A} + {1'b0,B};
assign CarryOut = tmp[8];
always @(posedge clk)
begin
    case(ALU_Sel)
        4'b0000: // Addition
            ALU_Result = A + B ;
        4'b0001: // Subtraction
            ALU_Result = A - B ;
        4'b0010: // Multiplication
            ALU_Result = A * B;
        4'b0011: // Division
            ALU_Result = A/B;
        4'b0100: // Logical shift left
            ALU_Result = A<<1;
        4'b0101: // Logical shift right
            ALU_Result = A>>1;
        4'b0110: // Rotate left
            ALU_Result = {A[6:0],A[7]};
        4'b0111: // Rotate right
            ALU_Result = {A[0],A[7:1]};
        4'b1000: // Logical and
            ALU_Result = A & B;
        4'b1001: // Logical or
            ALU_Result = A | B;
        4'b1010: // Logical xor
            ALU_Result = A ^ B;
    endcase
end

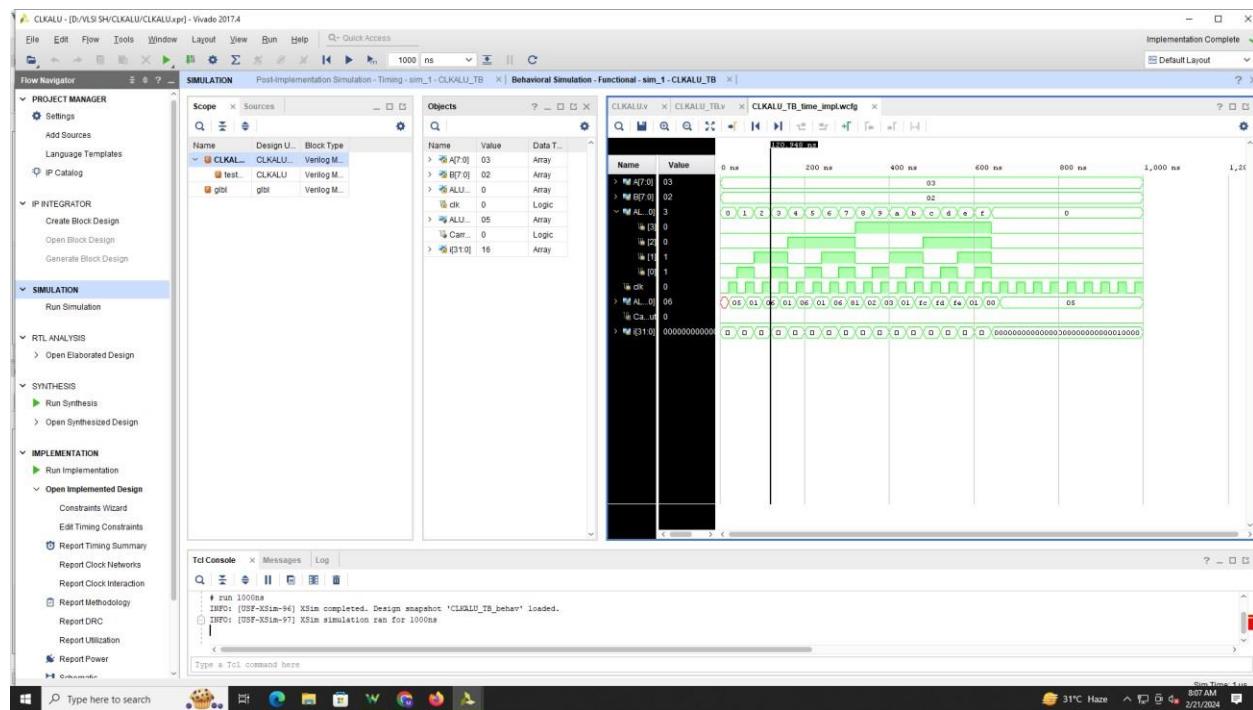
```

SIMULATION OUTPUT FOR WITH CLOCK:

SCHEMATIC:



BEHAVIOURAL SIMULATION:



```

4'b1011: // Logical nor
    ALU_Result = ~(A | B);
4'b1100: // Logical nand
    ALU_Result = ~(A & B);
4'b1101: // Logical xnor
    ALU_Result = ~(A ^ B);
4'b1110: // Greater comparison
    ALU_Result = (A>B)?8'd1:8'd0 ;
4'b1111: // Equal comparison
    ALU_Result = (A==B)?8'd1:8'd0 ;
default: ALU_Result = A + B ;
endcase
end
endmodule

```

TESTBENCH:

```

module tb_alu;
//Inputs
reg[7:0] A,B;
reg[3:0] ALU_Sel;
reg clk;
//Outputs
wire[7:0] ALU_Out;
wire CarryOut;
// Verilog code for ALU
integer i;
ArithmeticLU test_unit(
    A,B, // ALU 8-bit Inputs
    ALU_Sel,clk,// ALU Selection
    ALU_Out, // ALU 8-bit Output
    CarryOut // Carry Out Flag
);

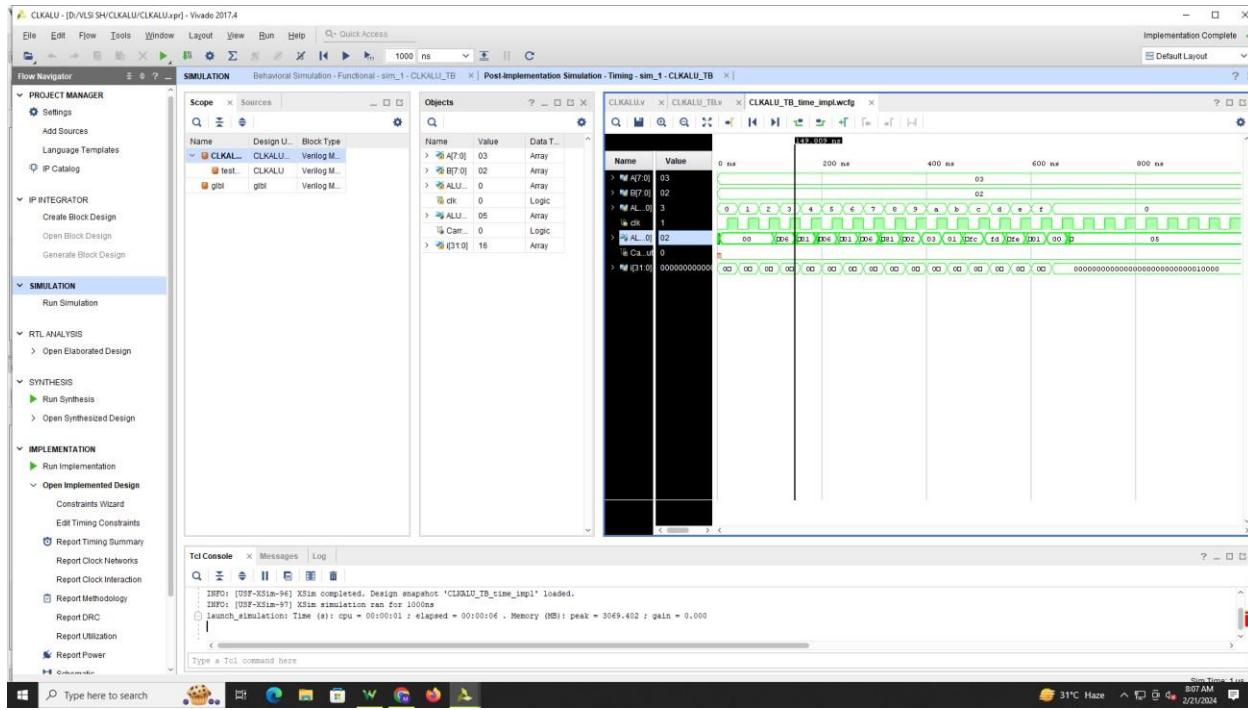
```

```

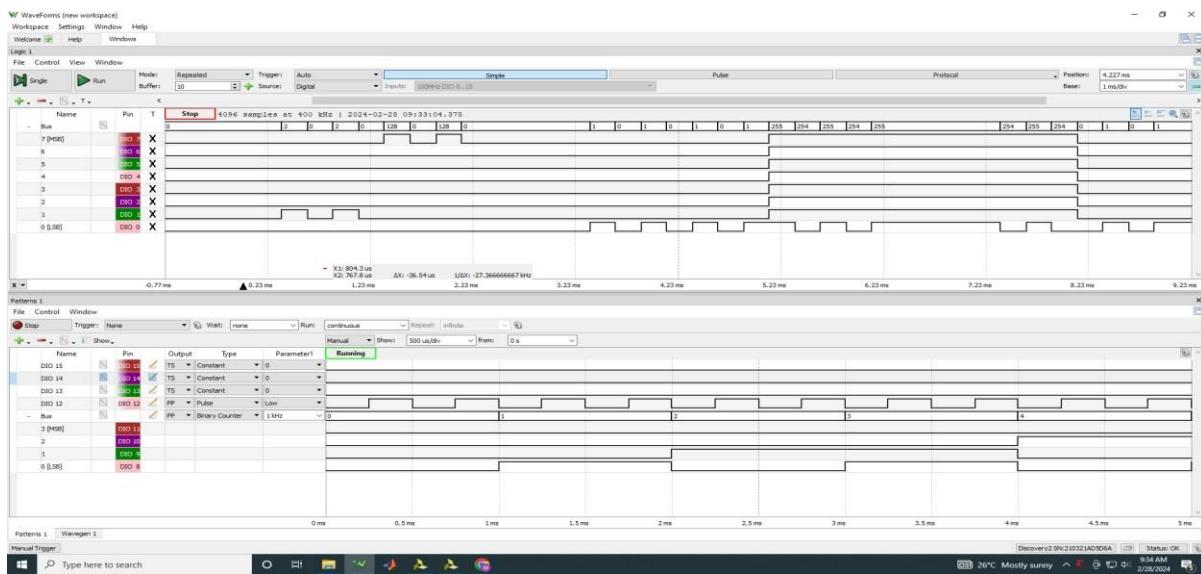
initial begin
    // hold reset state for 100 ns.
    A = 8'h03; B = 4'h02; ALU_Sel = 4'h0; clk=0;
    for (i=0;i<=15;i=i+1)
    begin
        #40;
        ALU_Sel = ALU_Sel + 8'h1;
    end;
    end
    always #20 clk=~clk;
endmodule

```

OUTPUT : POST IMPLEMENTATION:



WAVEFORMS OUTPUT: WITH INTERNAL CLOCK:



INFERENCE:

- From the post implementation , we can say that there exists some latency in the output due to both input and clock changes simultaneously so that there is some delay in the input to be stable.
- There exist some intermediate output
- We also observe the output using analog discovery using both internal and external clock.

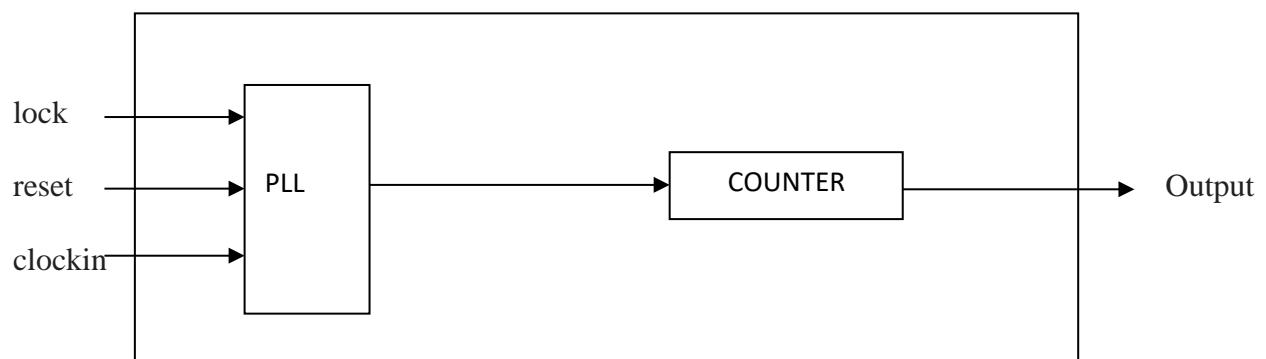
RESULT:

Thus we have designed and simulated arithmetic and logic units with clock using verilog and Basys 3 FPGA board and the waveforms is observed using analog discovery.

TRUTH TABLE:

Rst	CLK	O3	O2	O1	O0
1	↑	0	0	0	0
0	↑	0	0	0	1
0	↑	0	0	1	0
0	↑	0	0	1	1
0	↑	0	1	0	0
0	↑	0	1	0	1
0	↑	0	1	1	0
0	↑	0	1	1	1
0	↑	1	0	0	0
0	↑	1	0	0	1
0	↑	1	0	1	0
0	↑	1	0	1	1
0	↑	1	1	0	0
0	↑	1	1	0	1
0	↑	1	1	1	0
0	↑	1	1	1	1
0	↑	0	0	0	0

BLOCK DIAGRAM:



EXPT NO

DATE:

UP COUNTER WITH PLL

AIM:

To design and simulate up counter using PLL.out clock frequency

SOFTWARE REQUIREMENT:

Vivado 2022.2

CODE:

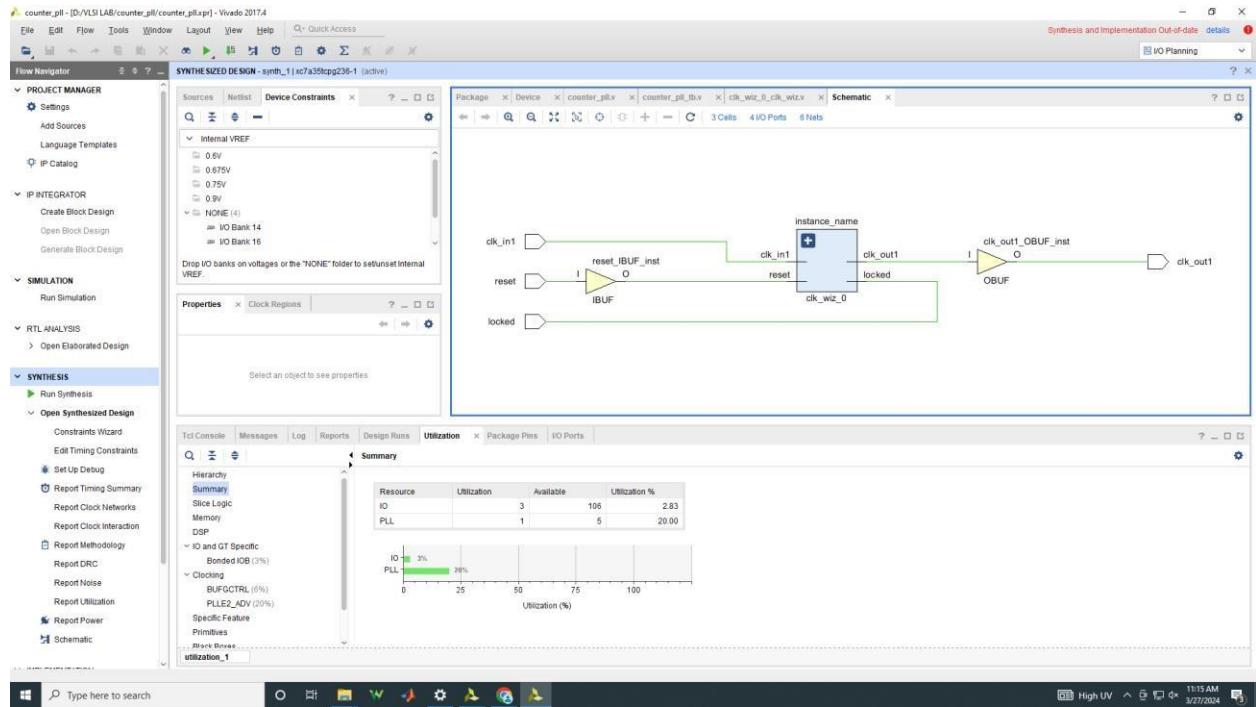
PLL CODE:

```
module pll(
    input clk_in1,
    input reset,
    output clk_out1,
    input locked
);
    clk_wiz_0 instance_name
        (.clk_out1(clk_out1),
        .reset(reset),
        .locked(locked),
        .clk_in1(clk_in1));
endmodule
```

PLL TESTBENCH:

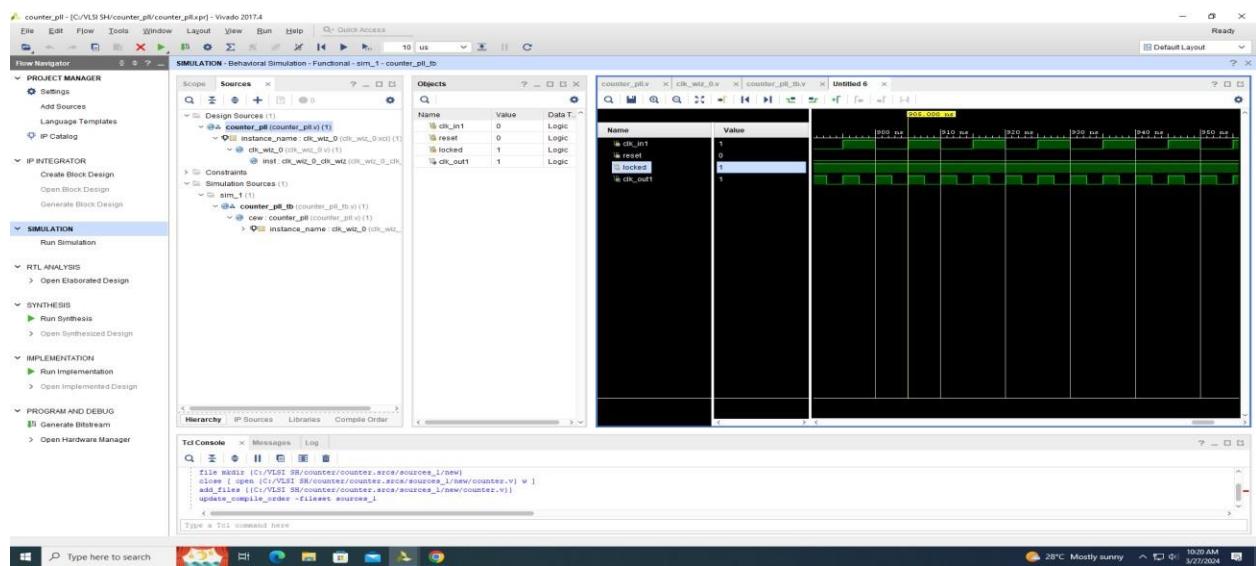
```
module pll_tb();
    reg clk_in1;
    reg reset;
    wire clk_out1;
    reg locked;
    pll uut(.clk_in1(clk_in1),.reset(reset),.clk_out1(clk_out1),.locked(locked));
    always #5 clk_in1 = ~clk_in1;
    initial begin
        clk_in1 = 0;
        locked = 0;
        reset = 0;
        #1
        reset = 1;
        #1
        reset = 0;
        #1 locked = 1;
```

SCHEMATIC:



SIMULATION OUTPUT:

BEHAVIOURAL SIMULATION:



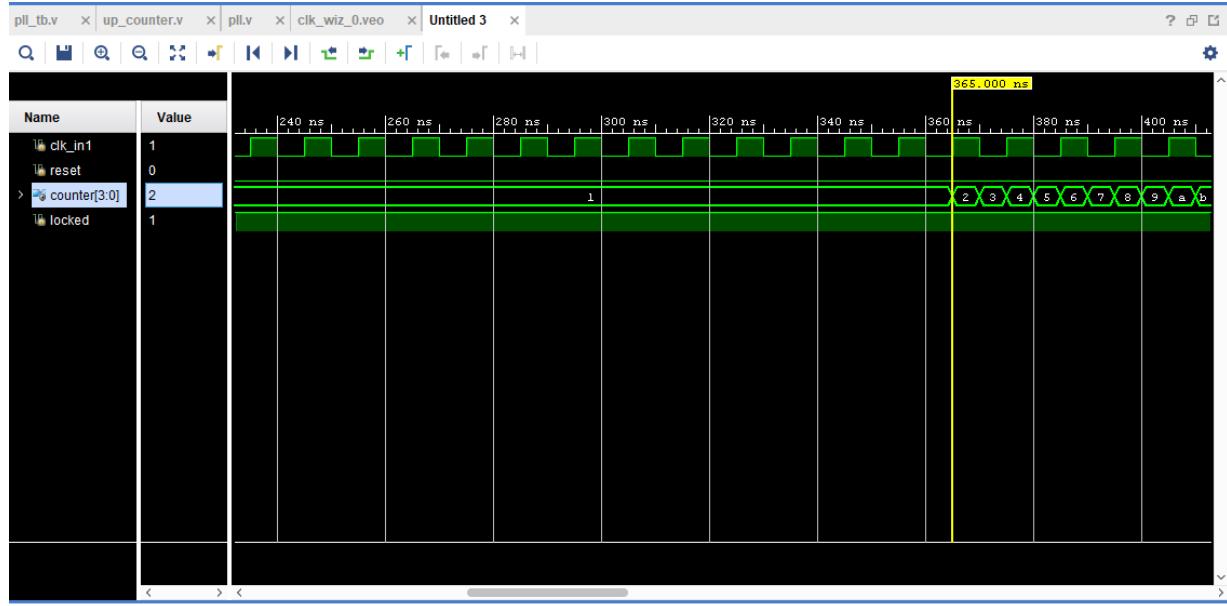
```
end  
endmodule
```

COUNTER CODE:

```
module counter(  
  
    // Inputs  
    input      clk_in1,locked,reset,  
    output reg [3:0] count  
  
)  
wire clk_out1;  
pll uut(  
.clk_in1(clk_in1),  
.reset(reset),.clk_out1(clk_out1),.locked(locked));  
always @ (posedge clk_out1 or posedge reset) begin  
if (reset == 1'b1) begin  
    count <= 0;  
end else begin  
    case (count)          // Gray code state  
        4'b0000: #1 count <= 4'b0001; // 0  
        4'b0001: #1 count <= 4'b0010; // 1  
        4'b0010: #1 count <= 4'b0011; // 2  
        4'b0011: #1 count <= 4'b0100; // 3  
        4'b0100: #1 count <= 4'b0101; // 4  
        4'b0101: #1 count <= 4'b0110; // 5  
        4'b0110: #1 count <= 4'b0111; // 6  
        4'b0111: #1 count <= 4'b1000; // 7  
        4'b1000: #1 count <= 4'b1001; // 8  
        4'b1001: #1 count <= 4'b1010; // 9  
        4'b1010: #1 count <= 4'b1011; // 10  
        4'b1011: #1 count <= 4'b1100; // 11  
        4'b1100: #1 count <= 4'b1101; // 12  
        4'b1101: #1 count <= 4'b1110; // 13  
        4'b1110: #1 count <= 4'b1111; // 14  
        4'b1111: #1 count <= 4'b0000; // 15  
        default: #1 count <= 4'b0000;  
    endcase  
end  
end
```

```
endmodule
```

POST IMPLEMENTATION:



COUNTER TESTBENCH:

```
module counter_pll();
    reg clk_in1;
    reg reset;
    wire [3:0] count;
    reg locked;

    counter uut(.clk_in1(clk_in1),.reset(reset),.count(count),.locked(locked));
    always #5 clk_in1 = ~clk_in1;
    initial begin
        clk_in1 = 0;
        locked = 0;
        reset = 0;
        #1
        reset = 1;
        #1
        reset = 0;
        #15
        locked = 1;
        end
    endmodule
```

INFERENCE:

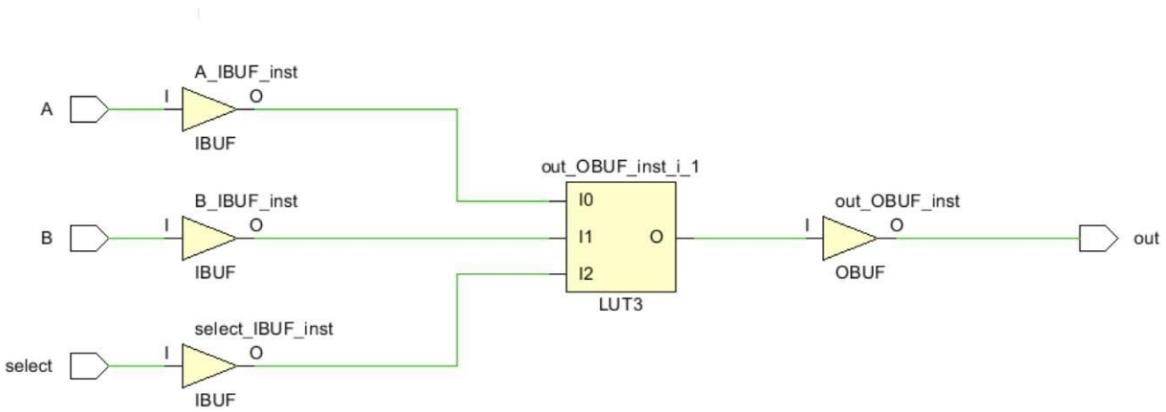
RESULT:

Thus , the up counter using pll is designed and simulated using Vivado.2022.2

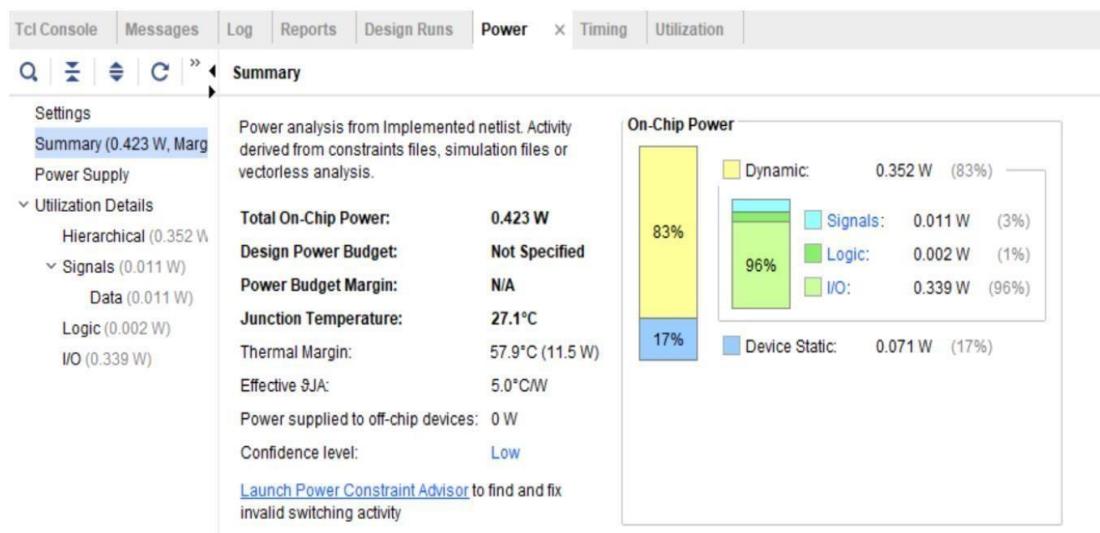
TRUTH TABLE:

SELECT LINE	INPUTS		OUTPUT
0	0	0	0
1	0	1	1
0	1	0	1
1	1	1	1

SCHEMATIC:



POWER ANALYSIS:



EXPT NO

DATE:

COMBINARIONAL CIRCITS

MULTIPLEXER

AIM:

To design, simulate and implement multiplexer using vivado.

SOFTWARE REQUIRED:

VIVADO 2022-2

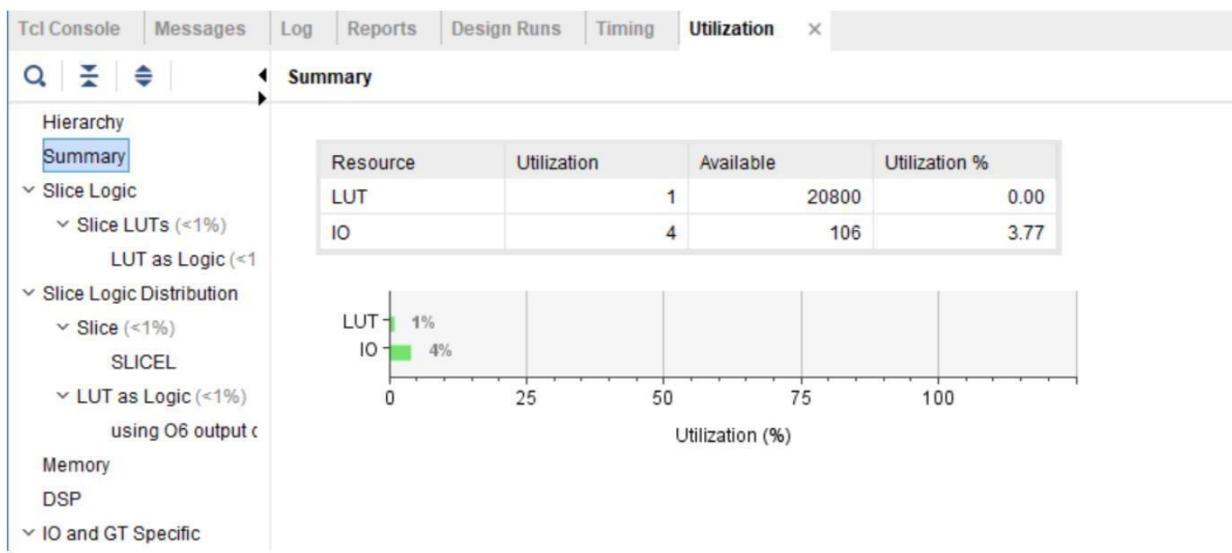
CODE:

```
module mux(A,B,select,out);
input A,B,select;
output reg out;
always@(select or A or B)
    if (select == 1)
        out = A;
    else
        out = B;
endmodule
```

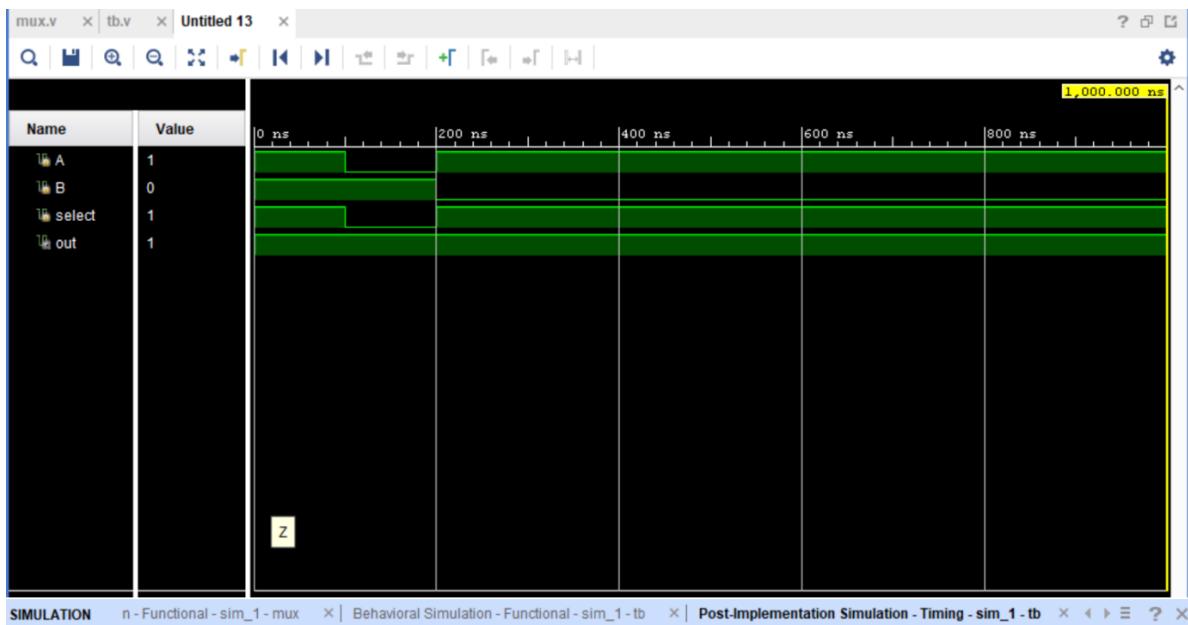
TEST BENCH:

```
module tb();
reg A,B,select;
wire out;
mux ut(A,B,select,out);
initial begin
A=1'b1;B=1'b1;
select = 1'b1;#100;
A = 1'b0;B=1'b1;
select = 1'b0; #100; A=1'b1;B=1'b0;
select = 1'b1; #100; A=1'b1;B=1'b0;
select = 1'b1;#100;
end
endmodule
```

DESIGN UTILIZATION:



OUTPUT:



INFERENCE:

The Verilog code is syntactically correct and implements a 2x1 multiplexer functionality. The test bench provides input stimuli to validate the functionality of the MUX under different scenarios. The design can be further verified through simulation in vivado to ensure its correctness. The waveform generated from simulation can be analyzed to confirm that the multiplexer behaves as expected, selecting the correct input based on the value of select.

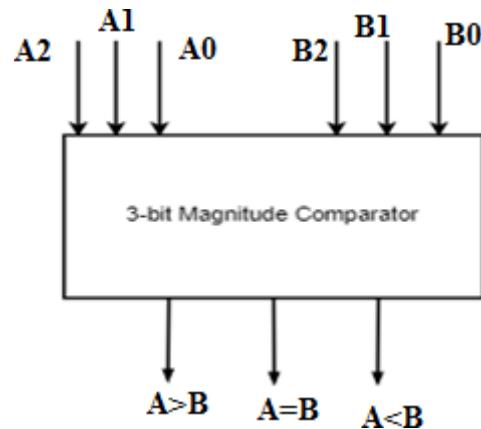
RESULT:

Thus a 2x1 multiplexer was designed and implemented using VIVADO.

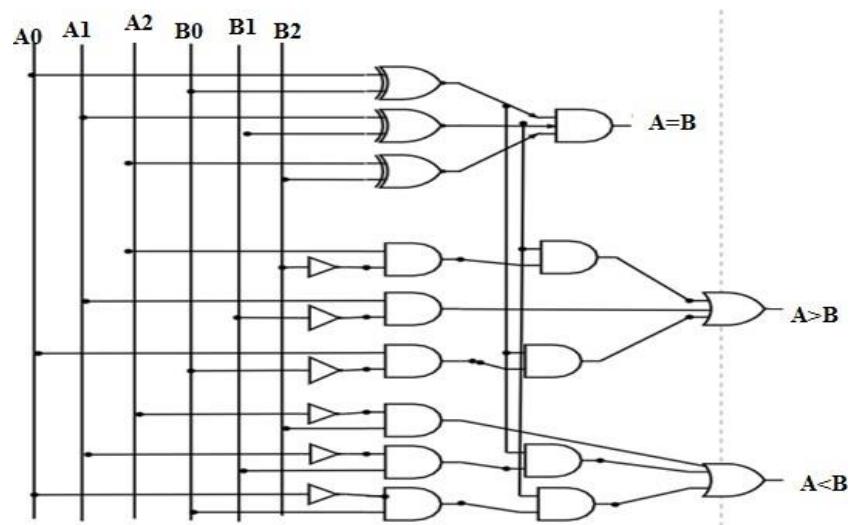
TRUTH TABLE:

INPUT						OUTPUT		
<u>A₂</u>	<u>A₁</u>	<u>A₀</u>	<u>B₂</u>	<u>B₁</u>	<u>B₀</u>	<u>A_grt_B</u>	<u>A less B</u>	<u>A_eq_B</u>
0	0	0	0	0	0	0	0	1
0	0	0	0	0	1	0	1	0
1	1	0	0	0	1	1	0	0
0	1	0	0	1	1	1	0	0
1	1	0	0	1	1	1	0	0
0	1	1	0	1	1	0	0	1

BLOCK DIAGRAM:



CIRCUIT DIAGRAM:



EXPT NO

DATE:

COMBINARIONAL CIRCITS

MAGNITUDE COMPARATOR

AIM:

To implement three bit magnitude comparator using Verilog code and Basys-3 board..

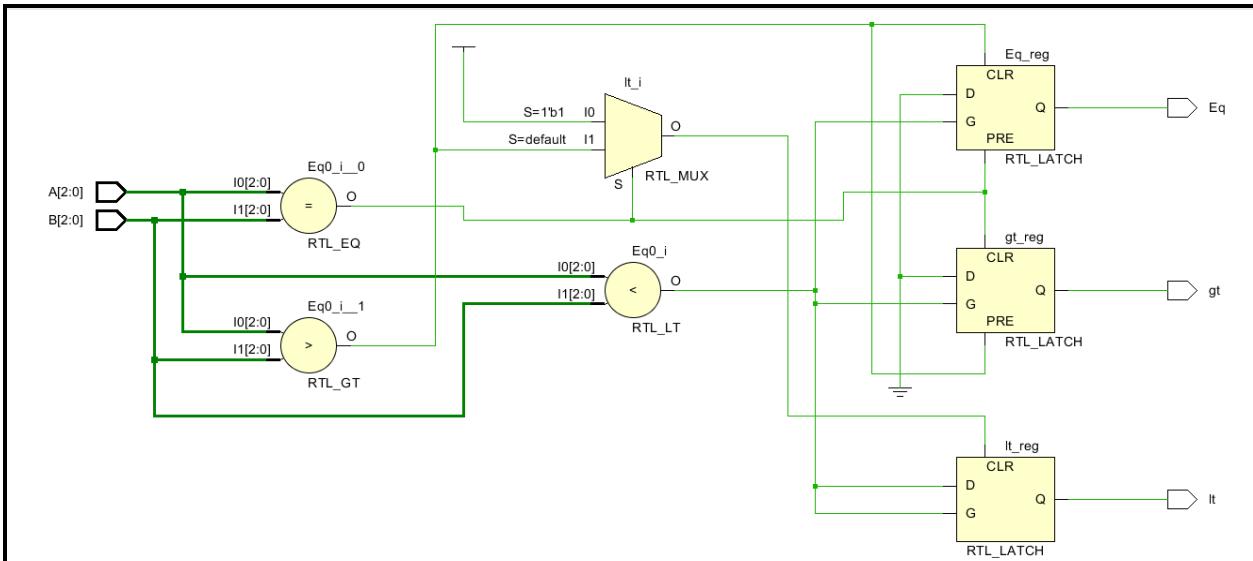
SOFTWARE REQUIRED:

VIVADO 2022-2

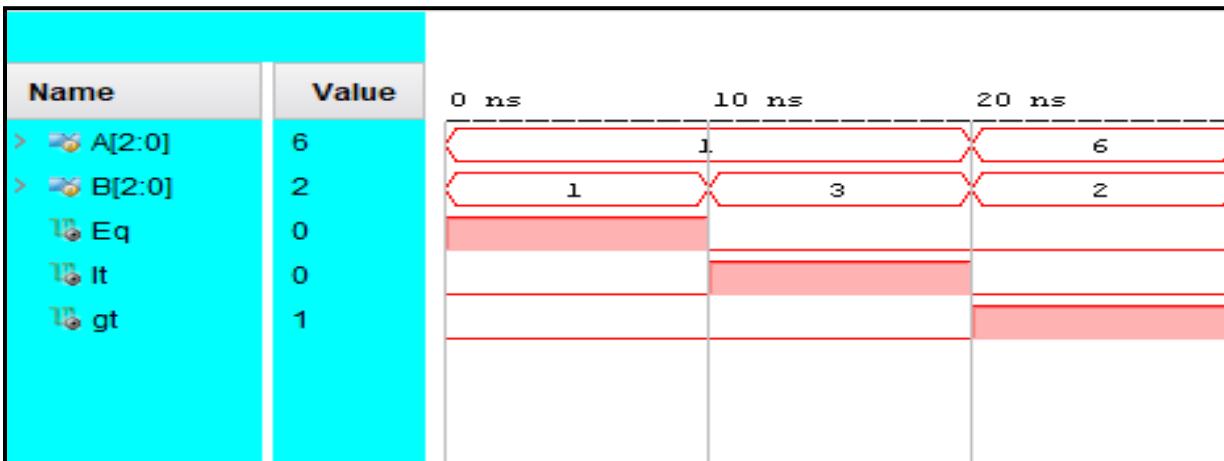
CODE:

```
module threebitcomp(  
    input [2:0]A,B, output reg A_grt_B,A_less_B,A_eq_B      );  
    always@(*)  
    begin  
        if(A>B) begin  
            A_grt_B=1'b1;  
            A_less_B=1'b0;  
            A_eq_B=1'b0;  
        end  
        else if(A<B) begin  
            A_grt_B=1'b0;  
            A_less_B=1'b1;  
            A_eq_B=1'b0;  
        end  
        else begin  
            A_grt_B=1'b0;  
            A_less_B=1'b0;  
            A_eq_B=1'b1;  
        end  
    end  
endmodule
```

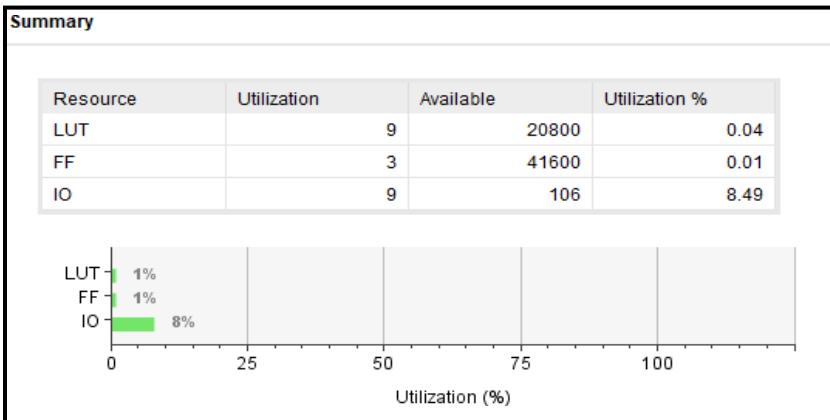
SCHEMATIC:



SIMULATION OUTPUT:



DESIGN UTILIZATION:



TEST BENCH:

```
module 3bitcomptb();
reg[2:0] A;
reg[2:0] B;
wire A_grt_B,A_less_B,A_eq_B;
threebitcomp z(.A(A), .B(B),. A_grt_B(A_grt_B), .A_less_B(A_less_B), A_eq_B(A_eq_B));
initial begin
A=3'b000;
B=3'b000;
#10
A=3'b000;
B=3'b000;
#10
A=3'b000;
B=3'b001;
#10
A=3'b110;
B=3'b001;
#10
A=3'b010;
B=3'b110;
#10
$finish;
end
endmodule
```

POWER ANALYSIS:

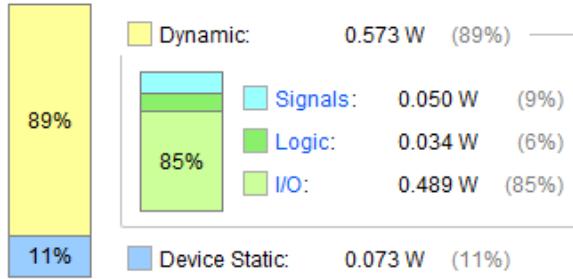
Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power: 0.646 W
Design Power Budget: Not Specified
Power Budget Margin: N/A
Junction Temperature: 28.2°C
Thermal Margin: 56.8°C (11.3 W)
Effective θJA: 5.0°C/W
Power supplied to off-chip devices: 0 W
Confidence level: Low

[Launch Power Constraint Advisor](#) to find and fix invalid switching activity

On-Chip Power



DELAY ANALYSIS:

Setup delay:

Total Delay	Logic Delay	Net Delay
6.927	3.669	3.258

Hold delay:

Total Delay	Logic Delay	Net Delay
2.099	1.340	0.759

INFEERENCE:

In magnitude comparator input signal of three bit of two signals are compared weather they are equal, greater than or less than one another and the output is set accordingly.

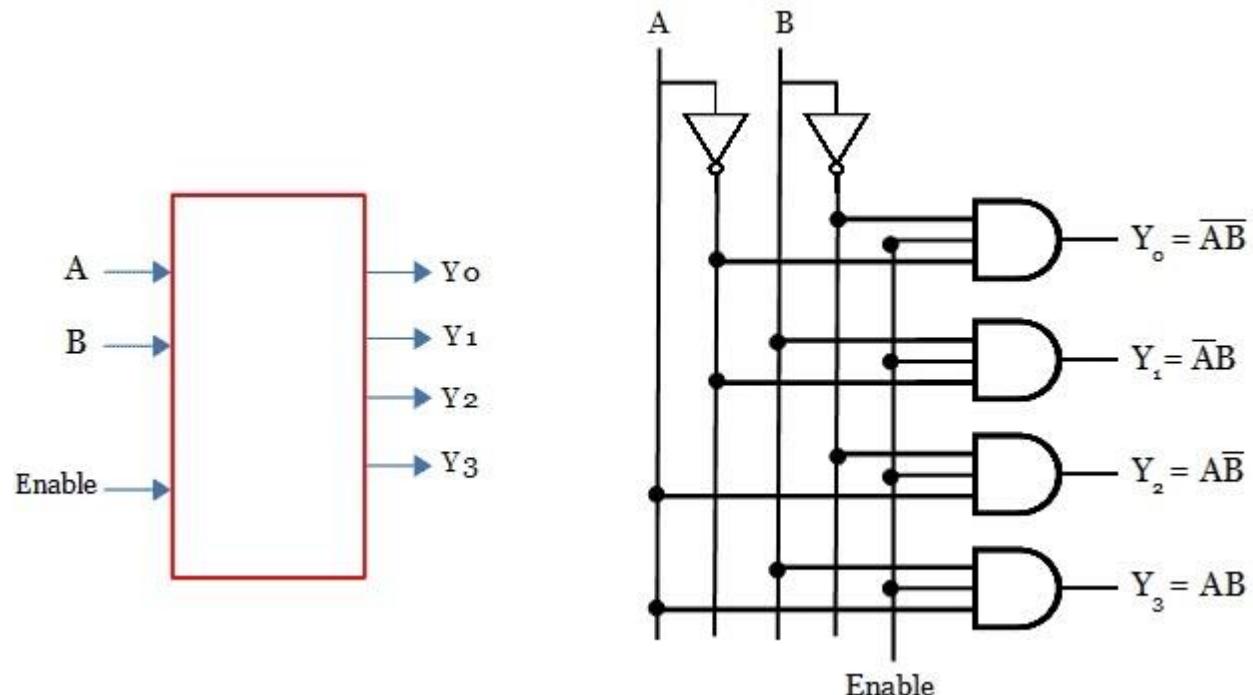
RESULT:

Thus the design and simulation of three bit magnitude comparator has been implemented in VIVADO.

TRUTH TABLE:

Inputs			Outputs			
EN	A	B	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

CIRCUIT DIAGRAM:



EXPT NO

DATE:

DECODER

AIM:

To design, simulate and implement 2X4 decoder using vivado.

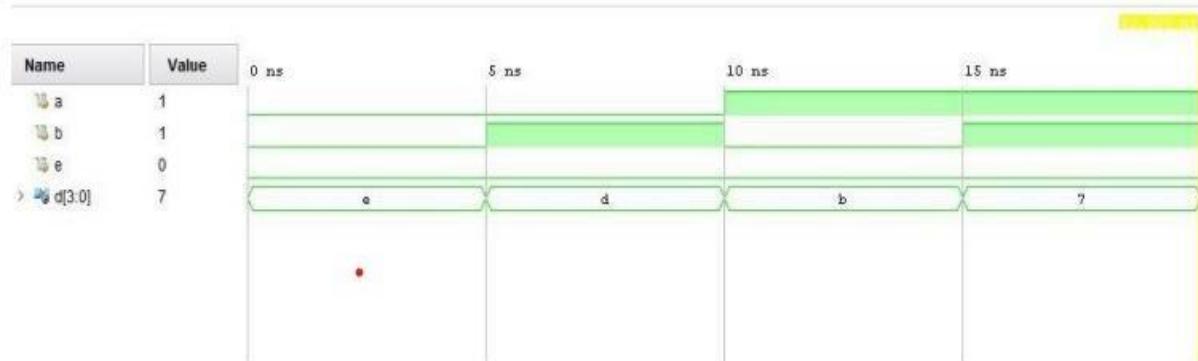
SOFTWARE REQUIRED:

VIVADO

CODE:

```
module decoder_df (A,B,E,D);  
input A,B,E;  
output [0:3] D;  
assign D[0] = ~(~A & ~B & ~E),  
      D[1] = ~(~A & B & ~E),  
      D[2] = ~(A & ~B & ~E),  
      D[3] = ~(A & B & ~E);  
endmodule
```

BEHAVIOURAL OUTPUT:



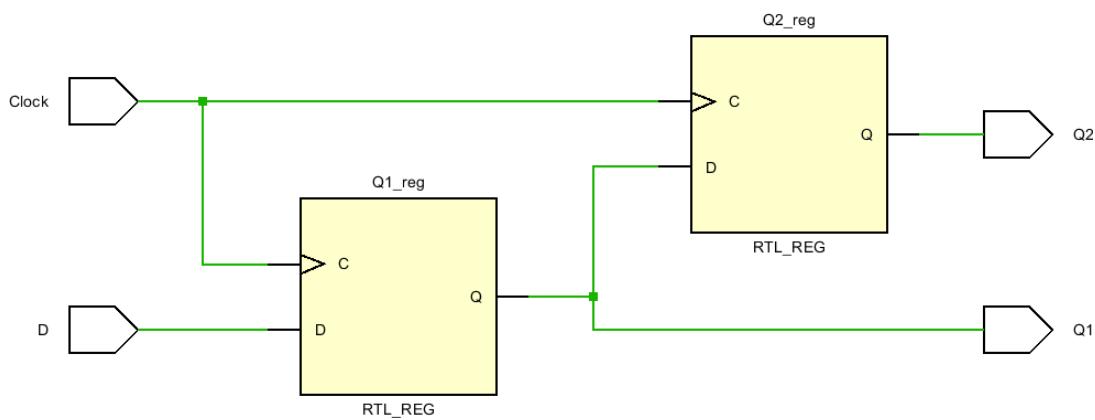
INFERENCE:

In this type of decoders, decoders have two inputs namely A, B, and four outputs denoted by D0, D1, D2, and D3. As in the truth table – for every input combination, one o/p line is turned on.

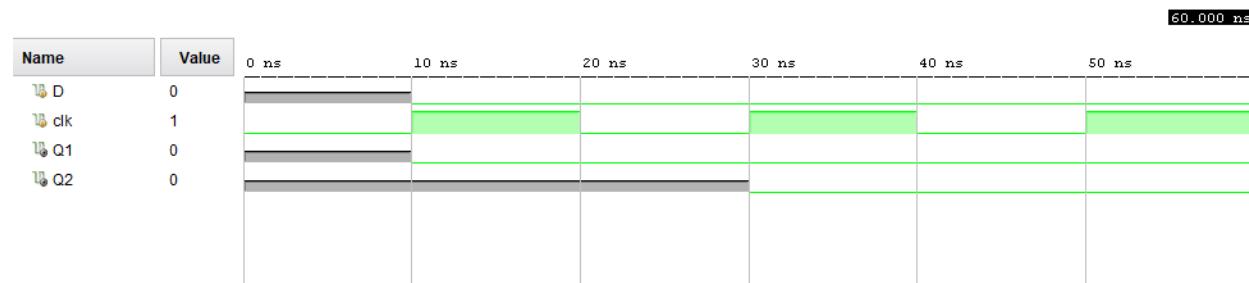
RESULT:

Thus 2x4 decoder is implemented in VIVADO.

SCHEMATIC - BLOCKING



BEHAVIOURAL SIMULATION:



POST TIMING IMPLEMENTATION:



EXPT NO

DATE:

SEQUENTIAL CIRCUIT

AIM:

To design, simulate and implement D flip flop(blocking and non-blocking) using vivado.

SOFTWARE REQUIRED:

VIVADO

CODE:

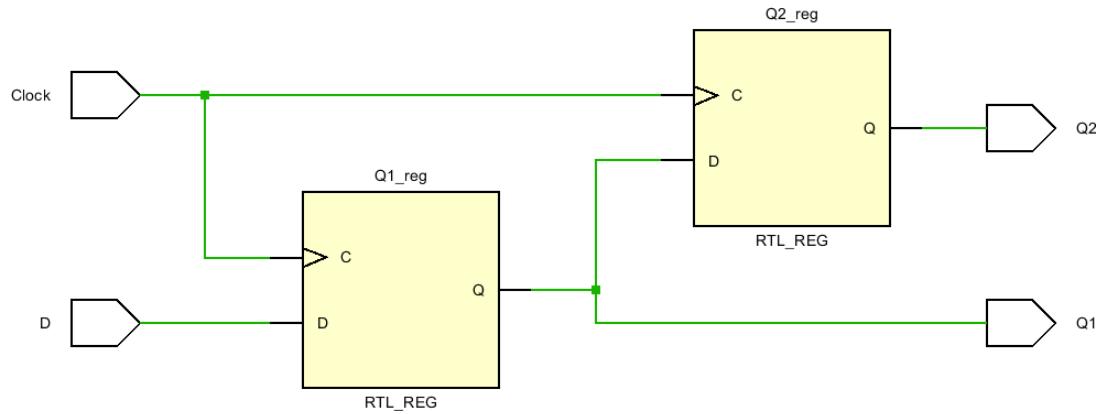
Non-blocking:

```
module NBDFF(D, Clock, Q1, Q2);
input D,Clock;
output Q1,Q2;
reg Q1, Q2;
always @(posedge Clock)
begin
// non-blocking assignment - series
Q1<= D;
Q2<= Q1;
end
endmodule
```

TEST BENCH:

```
module nbffd_tb();
reg D,clk;
wire Q1,Q2;
NBDFF cwd(D, clk, Q1, Q2);
always #10clk =~clk ;
initial
begin
clk=0;
#10
D=1'b0;
#50 $finish;
end
```

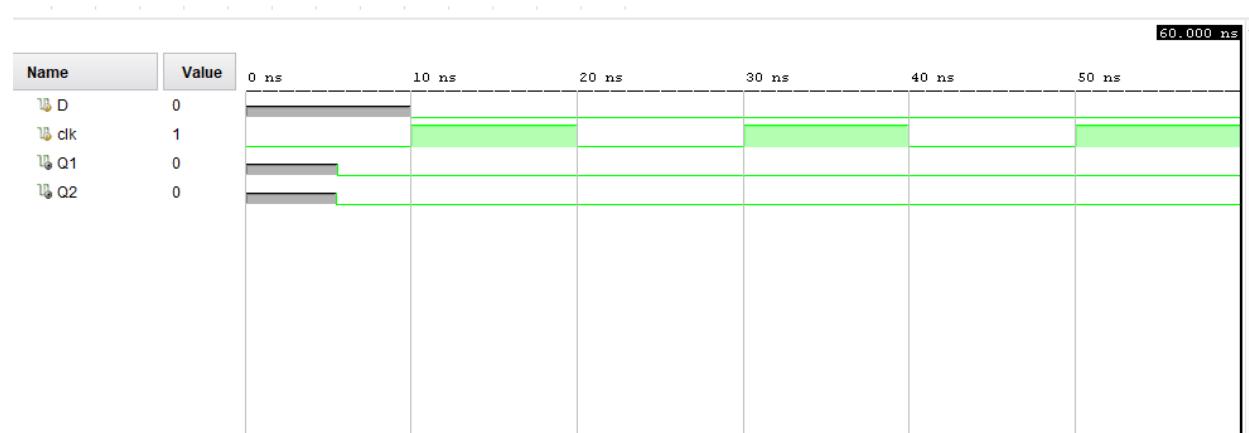
SCHEMATIC – BLOCKING



BEHAVIOURAL SIMULATION:



POST TIMING IMPLEMENTATION:



Blocking :

```
module BDFF(D, Clock, Q1, Q2);
input D,Clock;
output Q1,Q2;
reg Q1, Q2;
always @(posedge Clock)
begin
// blocking assignment - series
Q1 = D;
Q2 = Q1;
end
endmodule
```

Testbench:

```
module bffd_tb();
reg D,clk;
wire Q1,Q2;
BDFF cwd(D, clk, Q1, Q2);
always #10clk =~clk ;
initial
begin
clk=0;
#10
D=1'b0;
#50 $finish;
end
```

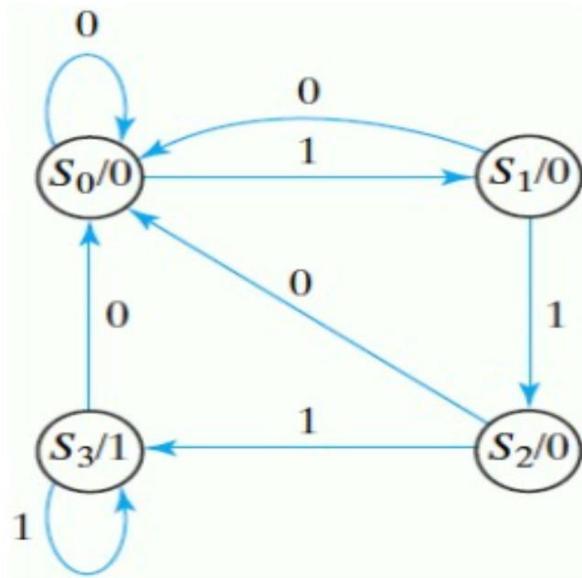

INFERENCE:

The Verilog code is syntactically correct and implements a blocking and non blocking D flip flop functionality. The test bench provides input stimuli to validate the functionality of the D flip flop under different scenarios. The design can be further verified through simulation in vivado to ensure its correctness.

RESULT:

Thus blocking and non blocking D flip flop was designed and implemented using vivado

STATE DIAGRAM:



TRUTH TABLE:

INITIAL STATE	INPUT		OUTPUT		NEXT STATE	
S0	0	1	0	0	S0	S1
S1	0	1	0	0	S0	S2
S2	0	1	0	1	S0	S3
S3	0	1	0	1	S0	S3

EXPT NO

DATE:

SEQUENCE DETECTOR (MOORE FSM)

AIM:

To design, implement and simulate Sequence detector (Moore FSM) using vivado.

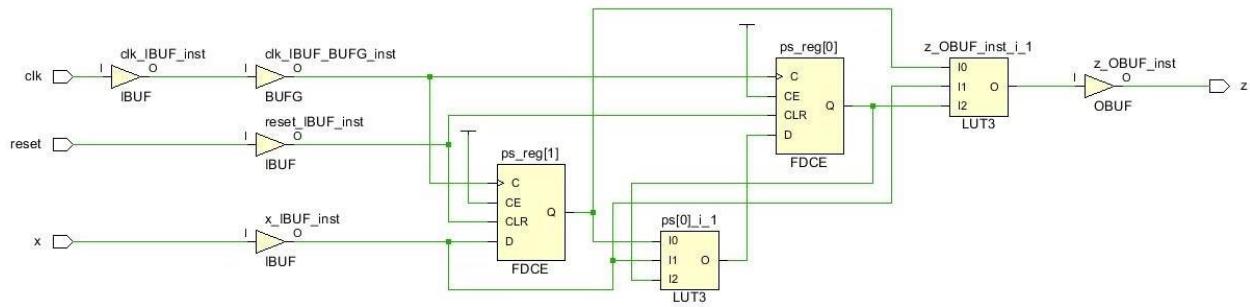
SOFTWARE REQUIRED:

VIVADO

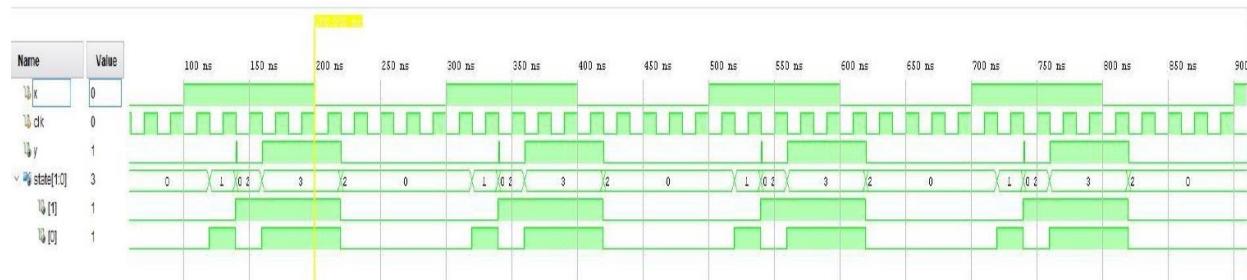
CODE:

```
module fsm(
    input in,
    input clk,
    input reset,
    output reg out,
    output reg [1:0] state
);
parameter s0=2'b00;
parameter s1=2'b01;
parameter s2=2'b10;
parameter s3=2'b11;
always @(*)
begin
if (reset)
begin
    state<=s0;
    out=1'b0;
end
case (state)
s0:
if (in==1'b1)
begin
    state<=s1;
    out<=1'b0;
end
else
begin
    state<=s0;
    out<=1'b0;
```

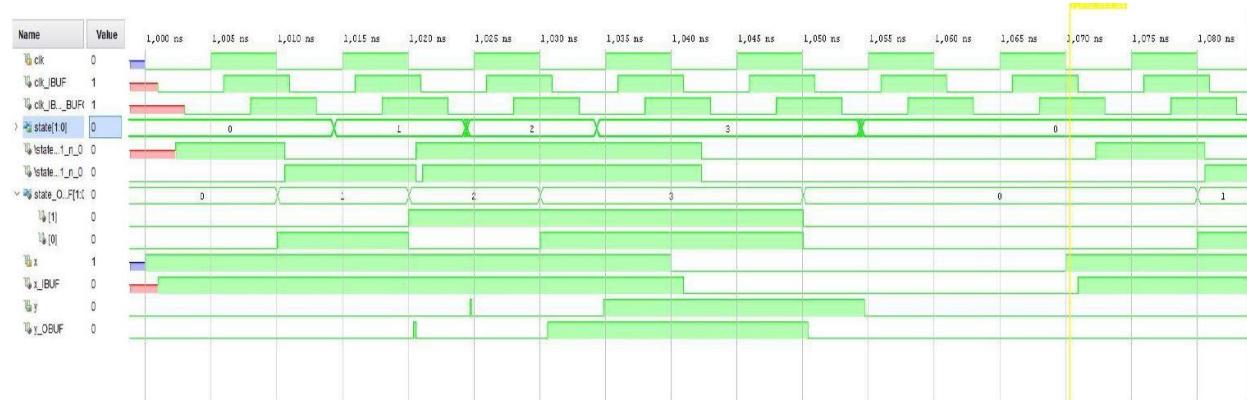
SCHEMATIC:



BEHAVIORAL:



POST IMPLEMENTATION:



```

end
s1:
if (in==1'b1)
begin
    state<=s2;
    out<=1'b0;
end
else
begin
    state<=s0;
    out<=1'b0;
end
s2:
if (in==1'b1)
begin
    state<=s3;
    out<=1'b1;
end
else

```

Test Bench:

```

module fsm_tb();
reg clk;
reg in;
reg reset;
wire [1:0]s;
wire out;
fsm uut(in,clk,reset,out,s);
always #10 clk=~clk;
initial
begin
    clk=1'b0;
    reset=1'b1;#10
    reset=1'b0;
    in=1'b1;#10
        in=1'b1;#10
    in=1'b1;#10

    $finish;
end
endmodule

```



```

begin
state<=s0;
    out<=1'b0;
end
s3:
if (in==1'b1)
begin
    state<=s3;
    out<=1'b1;
end
else
begin
    state<=s0;
    out<=1'b0;
end
endcase
end
endmodule

```

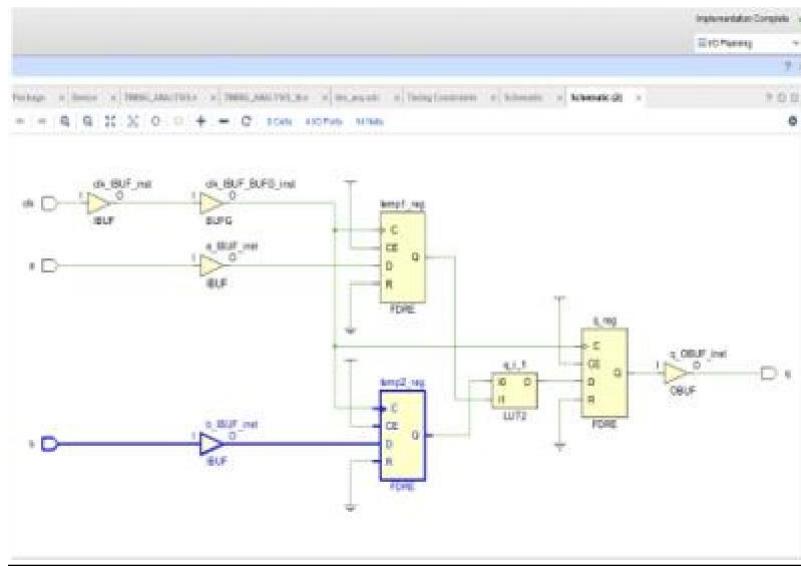
INFERENCE:

When the input changes from one bit to another the state changes from to another based on the state diagram. A Moore finite state machine (FSM) is a type of FSM where the outputs are associated with the states, rather than with the transitions.

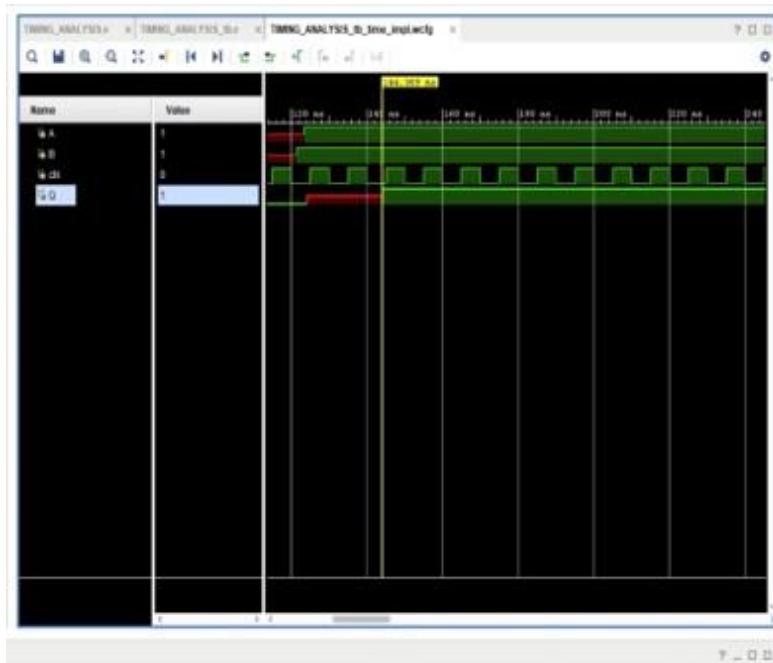
RESULT:

Thus Moore FSM is designed for the given state diagram and implemented using VIVADO.

SCHEMATIC:



BEHAVIOURAL SIMULATION:



EXPT NO

DATE:

TIMING ANALYSIS

AIM:

To design, implement and simulate timing analysis using AND gate and D flip flop in input and output.

SOFTWARE REQUIRED:

VIVADO

CODE:

```
module TIMING_ANALYSIS(A,B,clk,Q);
    input A,B;
    input clk;
    output reg Q;
    reg temp1, temp2;
    always @(posedge clk)
    begin
        temp1<=A;
        temp2<=B;
        Q<=temp1 & temp2;
    end
endmodule
```

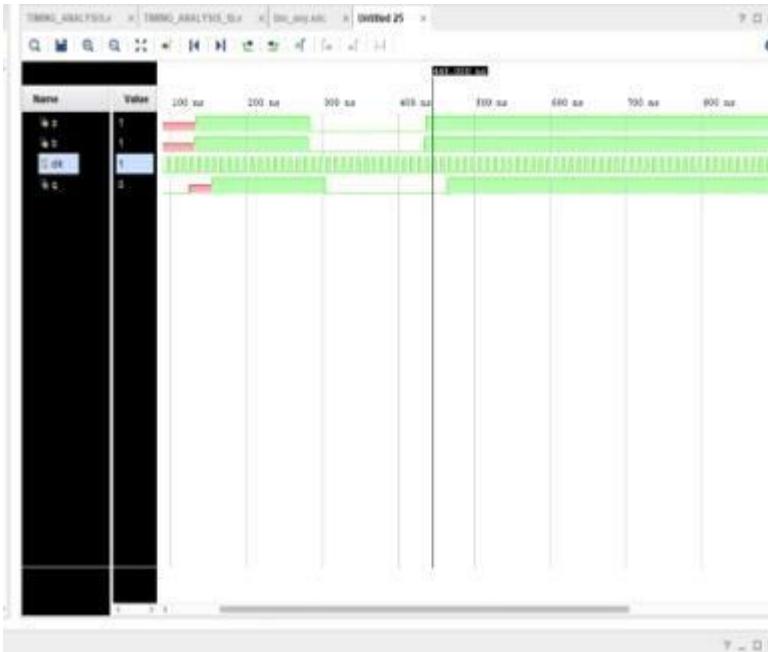

TEST BENCH

```
module TIMING_ANALYSIS_tb(
    );
    reg A,B,clk;
    wire Q;
    TIMING_ANALYSIS cwd(A,B,clk,Q);
    initial begin
        clk=0;
        forever #10 clk = ~ clk;
    end
    initial begin
        A<=0; B<=1; #100;
        B<=0;A<=0;#100;
        A<=1; B<=1; #100;
        A<=1; B<=1; #100;
    end
endmodule
```

TEST BENCH FOR WORST CASE DELAY:

```
module TIMING_ANALYSIS_tb();
    reg A,B,clk;
    wire Q;
    TIMING_ANALYSIS cwd(A,B,clk,Q);
    initial begin
        clk=0;
        forever #5 clk = ~ clk;
    end
    initial begin
        //A<=0; B<=1;
        //#10;
        //B<=0;A<=0;//#20;
        //A<=1; B<=1;// #30;
        // A<=1; B<=1;
        //#40;
        //B<=1;
        #121.5; B<=1;
        //A<=1;
        #2; A<=1;
    end
endmodule
```

POST IMPLEMENTATION:

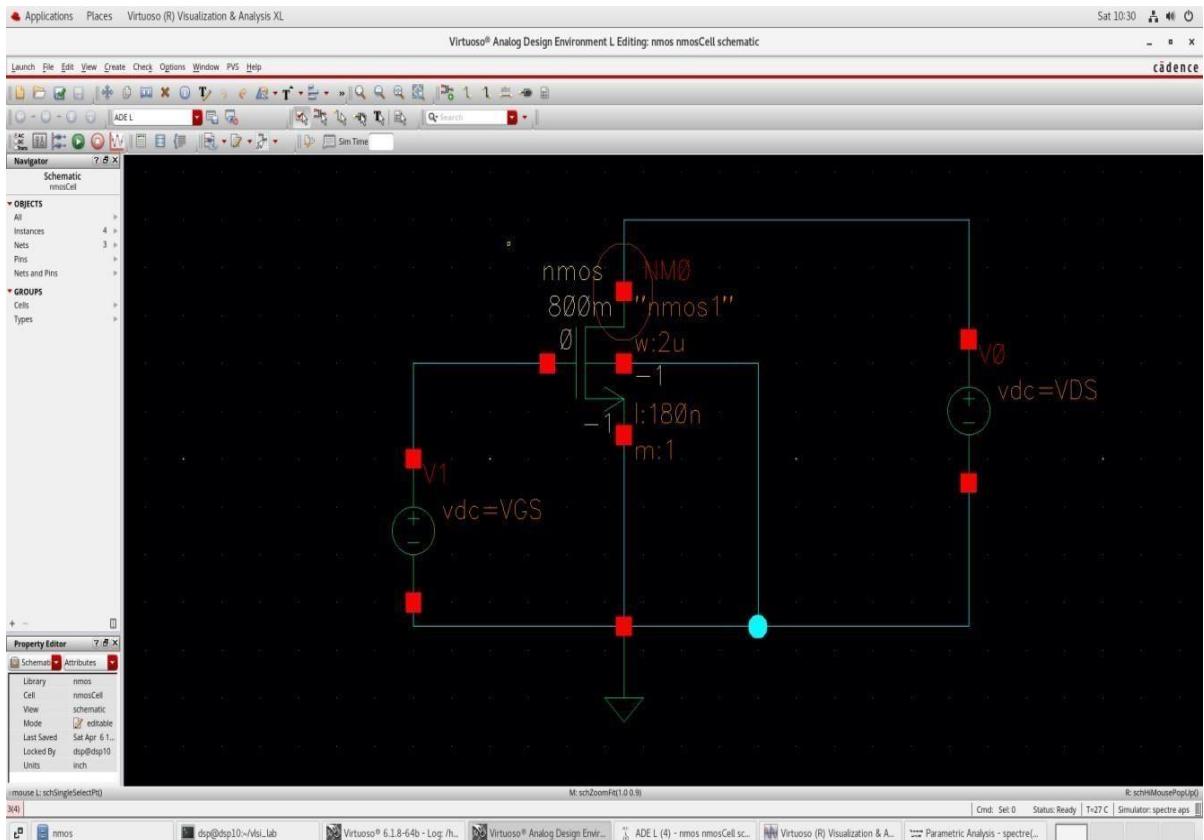
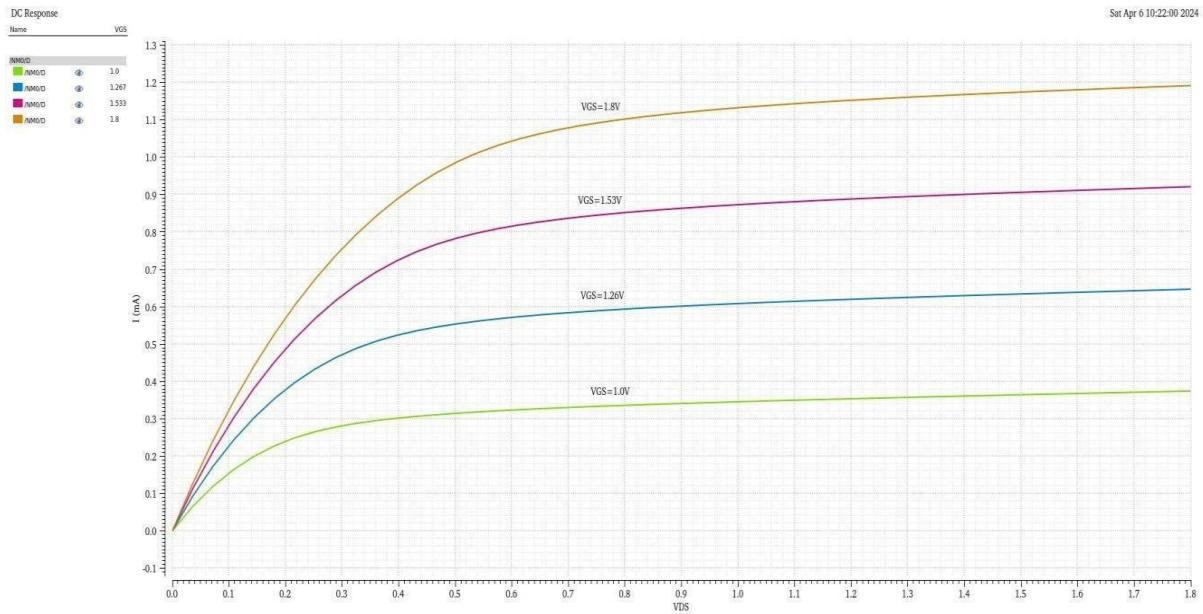


Setup	Hold
Worst Negative Slack (WNS): 0.534 ns	Worst Hold Slack (WHS): 0.142 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 4	Total Number of Endpoints: 4
All user specified timing constraints are met.	

RESULT:

Thus timing analysis for AND gate is implemented and behavioural simulation is seen by setting hold, set up to clock Q time and reducing slack time.

NMOS:



EXPT NO

DATE:

VOLTAGE TRANSFER CHARACTERISTICS

AIM:

To determine and plot the voltage transfer characteristics of n-mos and p-mos.

SOFTWARE REQUIRED:

CADENCE

PROCEDURE:

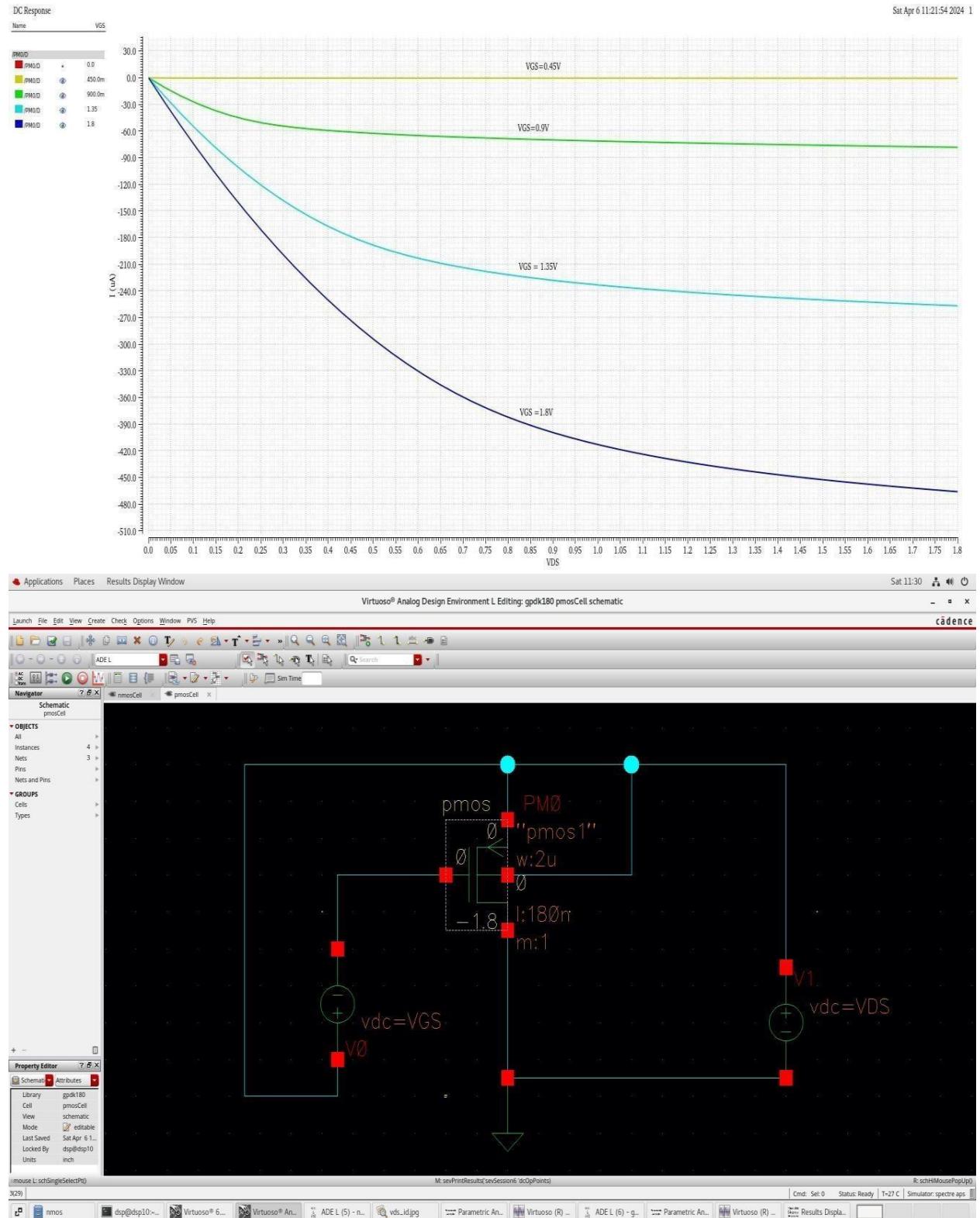
For opening the cadence

- 1) Create new folder in the home of the red hat.
- 2) Open the terminal.
- 3) Type csh.
- 4) Then type source /home/install/cshrc.
- 5) Welcome tab opens. Type virtuoso.
- 6) Log-in window opens and it opens a cadence design page
- 7) Then design the circuit using respective instances.

CIRCUIT PARAMETERS:

- Varying V_{gs} .
Therefore $V_{ds}=V_{gs}(in)$.
- Plot V_{ds} vs I_d .
Therefore $V_{dc}=V_{ds}$ (V_{dd}).
- Connect the body of the transistor to ground in n-mos and to V_{dd} in p-mos.
- W/L ratio = $2u/180n$ (as given).
- $V_{gs}=0.45V$
 $V_{gs}=0.5V$
 $V_{gs}=1.35V$
 $V_{gs}=1.8V$ for p-mos.
- $V_{gs}=1.0V$
 $V_{gs}=1.26V$
 $V_{gs}=1.53V$
 $V_{gs}=1.8V$ for n-mos.
- Perform transient analysis for input 2 output.

PMOS:



INFERENCE:

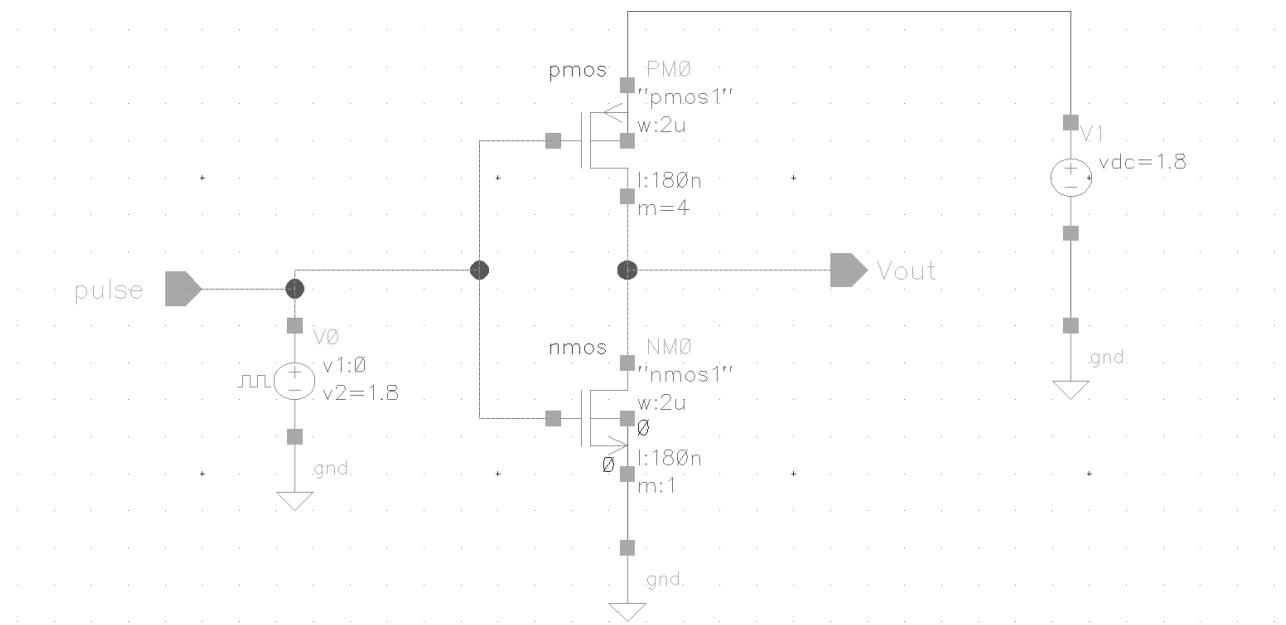
- The drain current (I_{ds}) increases as the gate-to-source voltage (V_{gs}) becomes more positive, and the transistor enters the saturation region when V_{gs} exceeds V_{th} .
- The drain current (I_{ds}) decreases as the gate-to-source voltage (V_{gs}) becomes more positive, and the transistor enters the saturation region when V_{gs} is less than V_{th} .
- The graph shows that the drain current increases (for NMOS) or decreases (for PMOS) as the gate-to-source voltage becomes more positive, and the transistor enters the saturation region when V_{gs} exceeds (for NMOS) or is less than (for PMOS) V_{th} .

RESULT:

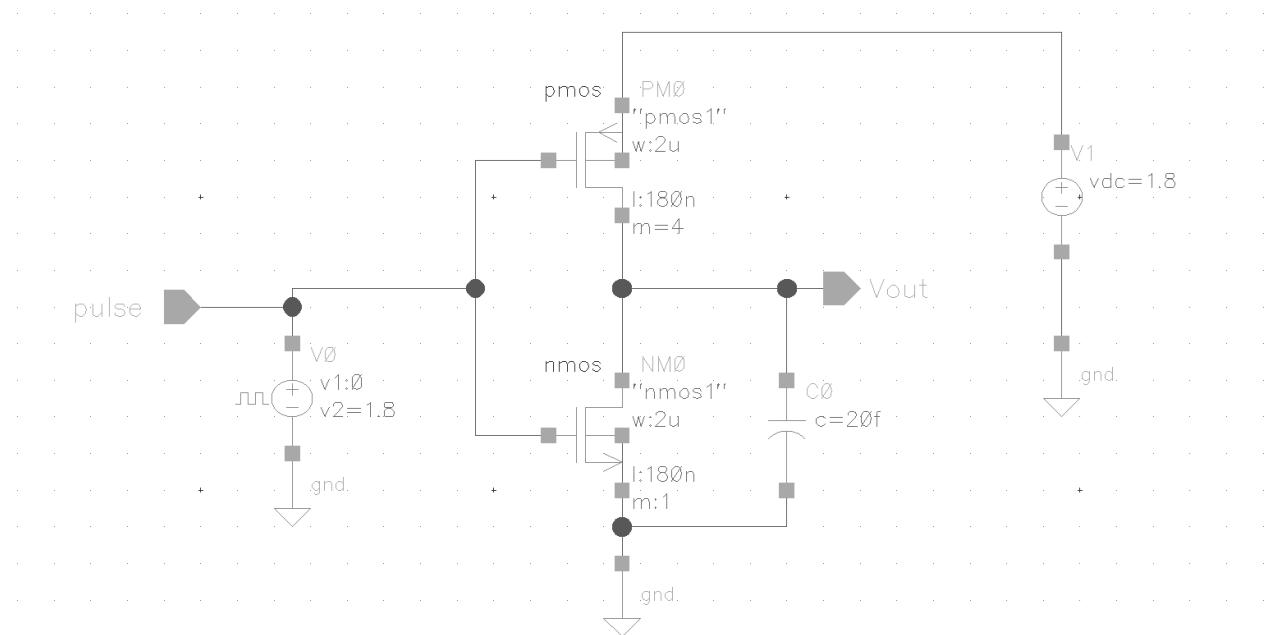
Therefore, the voltage transfer characteristics of n-mos and p-mos transistors are studied and analyzed.

CIRCUIT DIAGRAM:

Without load



With load



EXPT NO

DATE:

CMOS INVERTER

AIM:

To design a CMOS Inverter with load and without load and verify the propagation delay.

SOFTWARE REQUIRED:

CADENCE

CALCULATION:

Given: $V_{dd} = 1.8V$, $W/L = 2\mu/180n$, $V_{in} = 1.8V$

Without Load

Fall time(input) =

Rise time(output) =

$\Delta t_{LH} =$

Rise time(input) =

Fall time(output) =

$\Delta t_{HL} =$

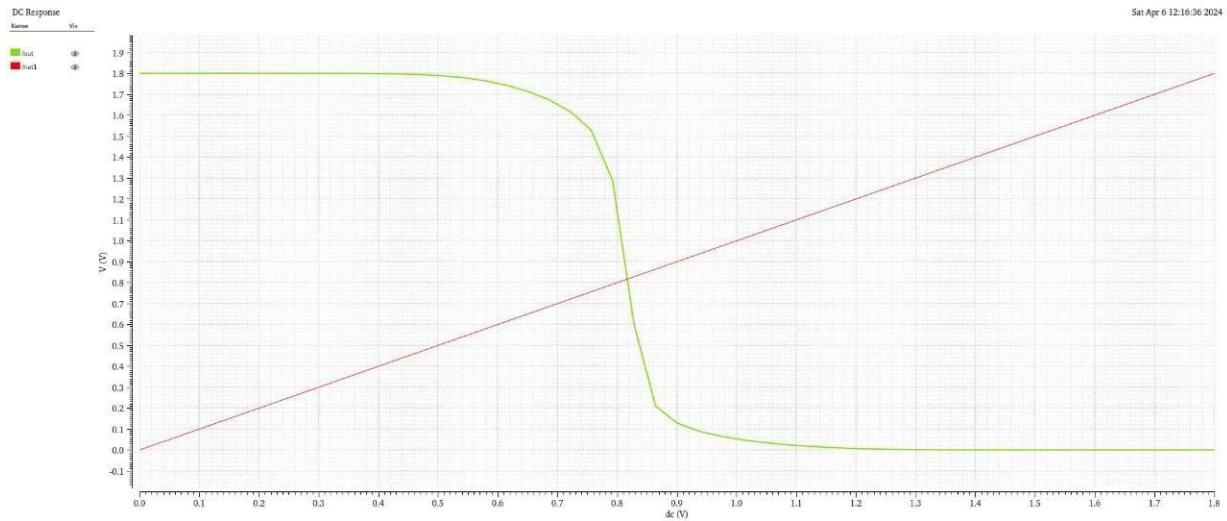
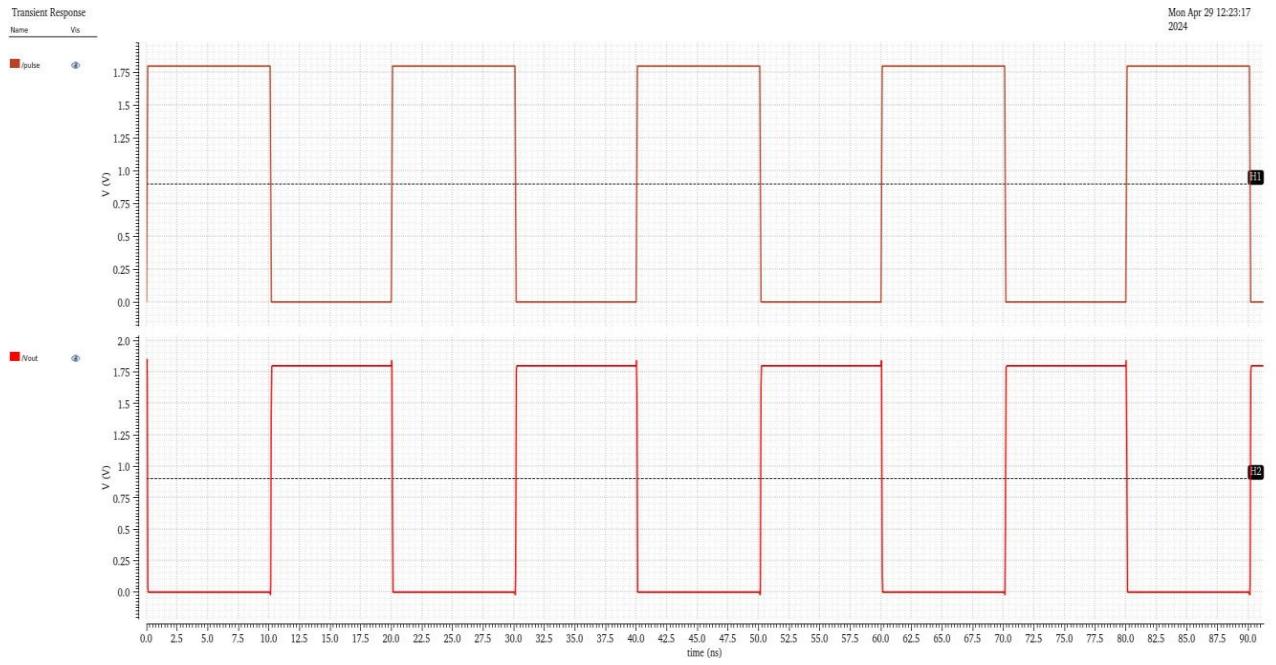
$t_{p0} =$

=

$V_H = 1.8/2 = 0.9 V$

OUTPUT:

Without load



With Load

Fall time(input) =

Rise time(output) =

$\Delta t_{LH} =$

Rise time(input) =

Fall time(output) =

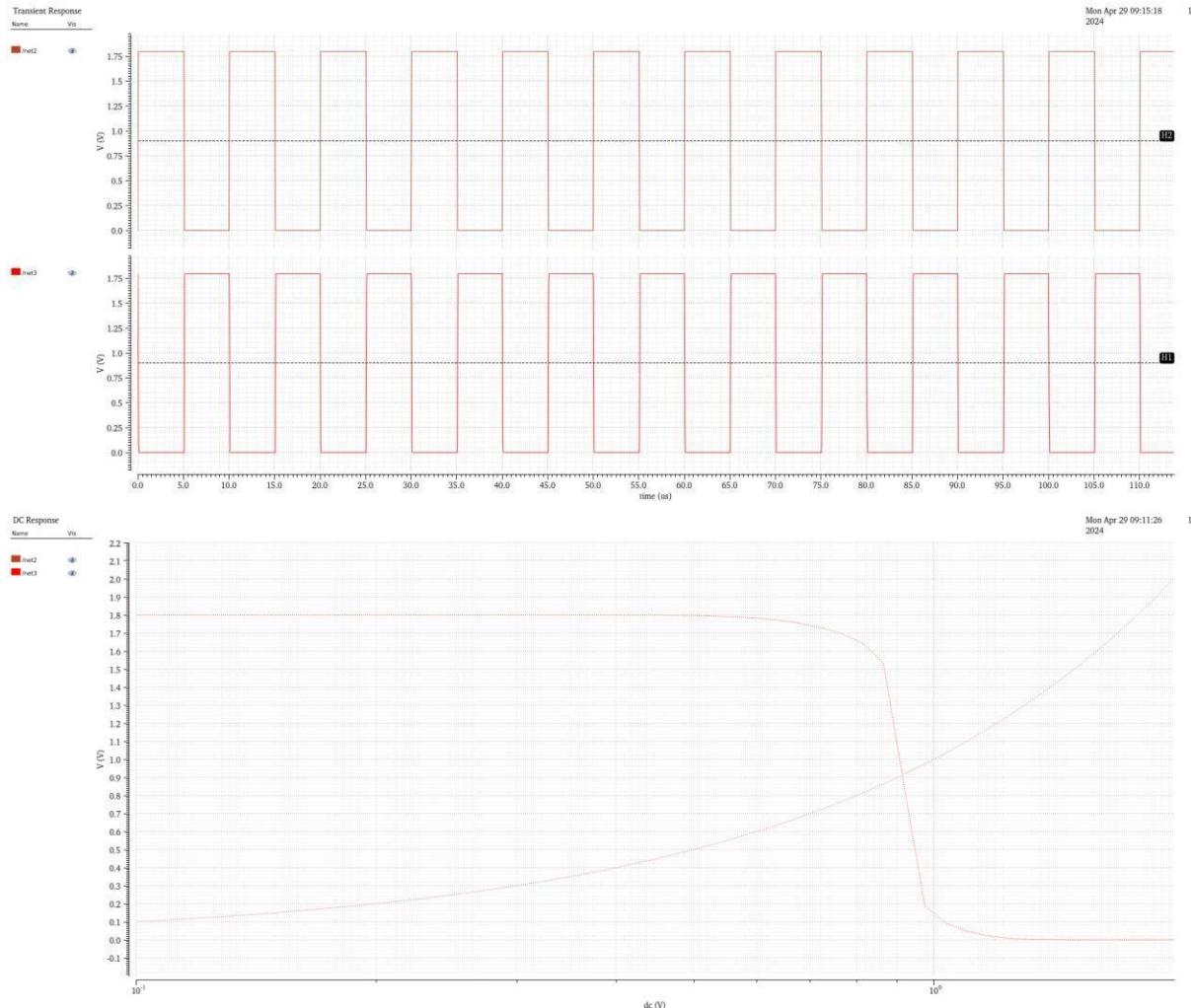
$\Delta t_{HL} =$

$t_{p0} =$

=

$$V_H = 1.8/2 = 0.9 \text{ V}$$

With load



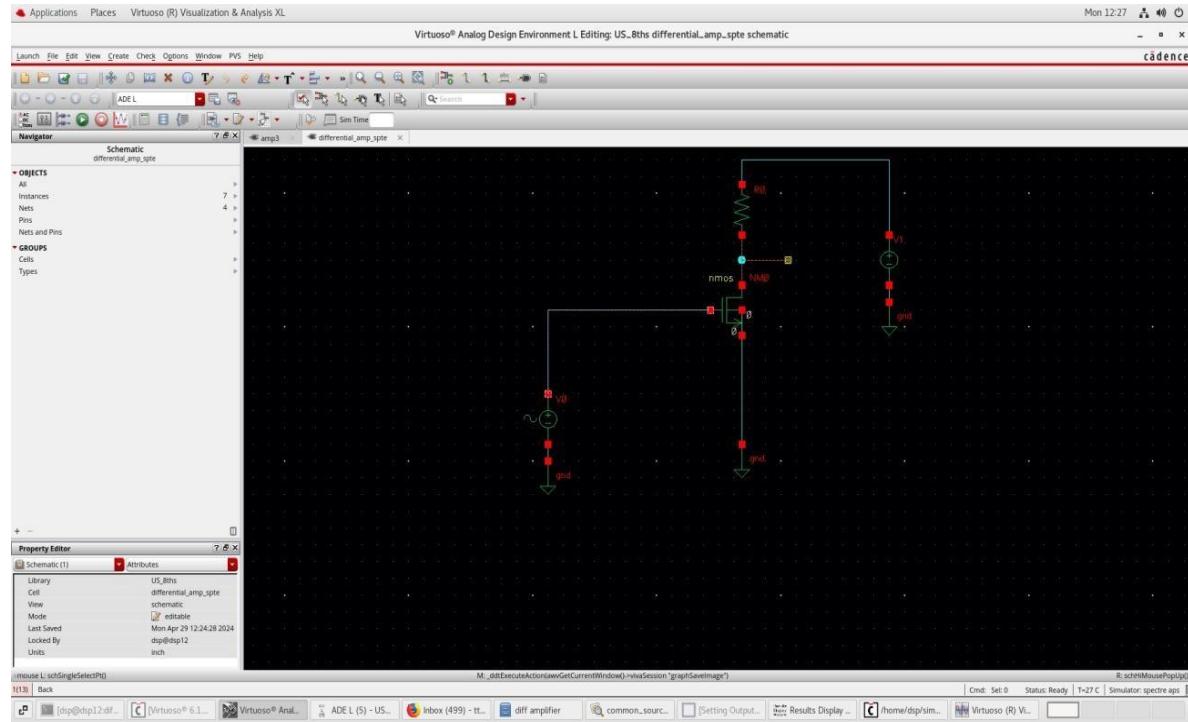
INFEERENCE:

With load, CMOS Inverter has propagation delay larger than reference inverter without load. The propagation delay is proportional to Capacitance. Without load, the propagation delay was dependent on only intrinsic capacitance, with load, propagation delay was dependent on bot intrinsic capacitance and extrinsic capacitance

RESULT:

Thus CMOS inverter with and without load is implemented in cadence.

SCHEMATIC:



DC ANALYSIS VALUE:

```

tmp1 = "signal" OPT("\\/NMO\\\" ??\\")"
tmp2 = "beff" 378.05u "
tmp3 = "betaeff"
tmp4 = "cbb"
tmp5 = "cbd" 329.02u "
tmp6 = "cbbbi"
tmp7 = "cbg" 3.57697f "
tmp8 = "cbs"
tmp9 = "cbsbi" 151.219a "
tmp10 = "cdb"
tmp11 = "cdd" -546.583a "
tmp12 = "cddb" 85.0966a
tmp13 = "cdg"
tmp14 = "cds"
tmp15 = "cgb" -735.561a "
tmp16 = "cgbovl" 7.19031f "
tmp17 = "cgd"
tmp18 = "cgdbi" 6.29694f "
tmp19 = "cgdovl"
tmp20 = "cgg"
tmp21 = "cggb" 0 "
tmp22 = "cgs" -1.57368f "
tmp23 = "cgbsbi"
tmp24 = "cgsov" -51.6698a "
tmp25 = "cjdf"
tmp26 = "cjs" NaN "
tmp27 = "csb" 1.3402G "
tmp28 = "csd"
tmp29 = "csg" 1.94327u "
tmp30 = "css" 60.5478u "
tmp31 = "cssbi"
tmp32 = "ft"
tmp33 = "fug" 17.5801u "
tmp34 = "gbd"
tmp35 = "gbs" 17.5801u
tmp36 = "gds" NaN "
tmp37 = "gm"
tmp38 = "gmb"
tmp39 = "gmbs" 0 "
tmp40 = "gmoverid" -78.3611a "
tmp41 = "i1" 78.3278a "
tmp42 = "i2" NaN "
tmp43 = "i3" NaN "
tmp44 = "i4" NaN "
tmp45 = "i5" NaN "
tmp46 = "i6" NaN "
tmp47 = "i7" NaN "
tmp48 = "i8" NaN "
tmp49 = "i9" NaN "
tmp50 = "i10" NaN "
tmp51 = "i11" NaN "
tmp52 = "i12" NaN "
tmp53 = "i13" NaN "
tmp54 = "igb" 0 "
tmp55 = "igcd"
tmp56 = "igcs"
tmp57 = "igd"
tmp58 = "ige" 0 "
tmp59 = "igidl" -7.47674u "
tmp60 = "igisl" -0 "
tmp61 = "igs" 7.47674u "
tmp62 = "is"
tmp63 = "isb" 33.2908z "
tmp64 = "ise"
tmp65 = "isub"
tmp66 = "pwr" 0 "
tmp67 = "qb" -152.11a "
tmp68 = "qbd"
tmp69 = "qbi" -16.3696a "
tmp70 = "qbs"
tmp71 = "qd" 5.86186f "
tmp72 = "qdi" 2 "
tmp73 = "cg" "
tmp74 = "ogi" "
tmp75 = "qinv" 0 "
tmp76 = "qsi" NaN "
tmp77 = "qsrco"
tmp78 = "region" 37.3348m "
tmp79 = "reversed" 0 "
tmp80 = "ron" 304.648m "
tmp81 = "rout" 304.648m "
tmp82 = "self_gain"
tmp83 = "tk"
tmp84 = "trise" 226.965m "
tmp85 = "type"
tmp86 = "ueff" 374.352m "
tmp87 = "vbs" 679m "
tmp88 = "vdb"
tmp89 = "vsat_marg" 77.6835m "
tmp90 = "vsb" -0 "
tmp91 = "vsl" 480.118m "

```

EXPT NO

DATE:

COMMON SOURCE

AIM:

To design and implement the common source NMOS circuit and perform dc-transient and ac analysis.

SOFTWARE REQUIRED:

CADENCE

CALCULATION:

$$I_d = 10 \mu A ; V_{gs} - V_{tn} = 0.2V ; A_v = 20$$

$$G_m = 2I_d/V_{gt} = 100 \mu$$

$$A_v = g_m R_d$$

$$R_d = 200 \text{ kohm}$$

Practical values:

$$V_{tn} = 479.3 \text{ mV}$$

$$V_{gs} = 0.69 \text{ V}$$

$$K_n' = 500 \mu \text{A/V}^2$$

$$I_d = K_n' (W/L) ((V_{gs} - V_t)^2 / 2) \quad (\text{Saturation region})$$

This implies

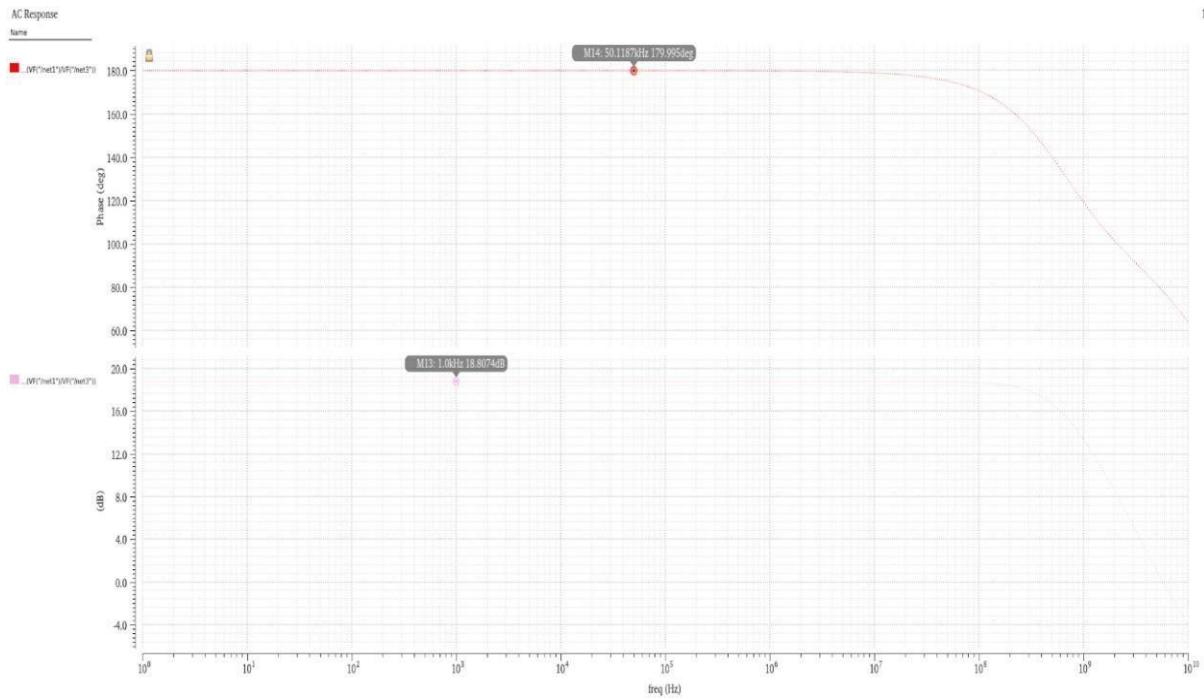
$$10 \mu = 500 \mu (W/L) (0.2/2)^2$$

$$W/L = 1$$

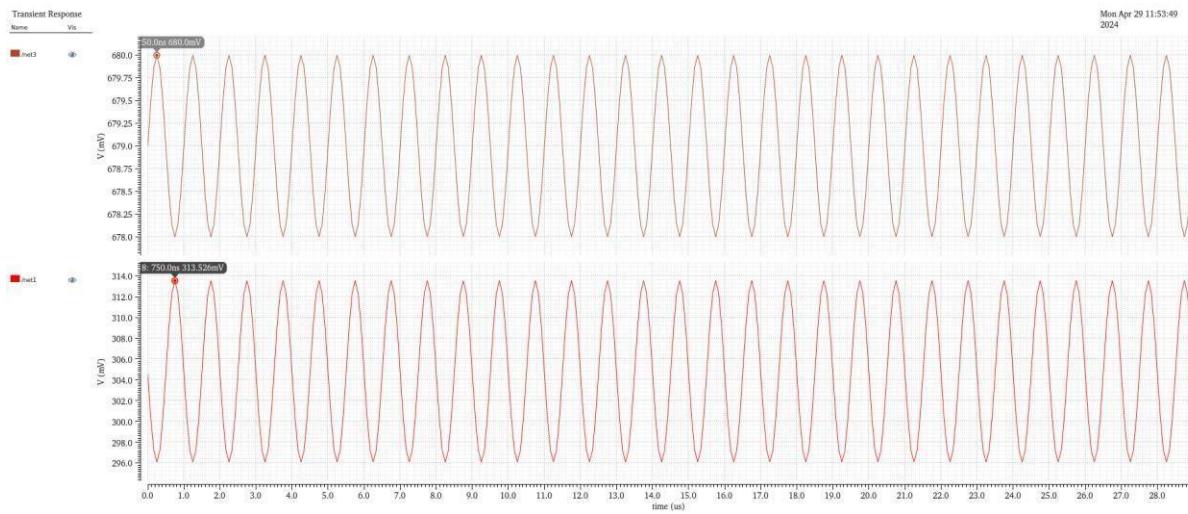
For W/L=1

$$I_d = ((60-64)(0.679-0.48))/2$$

AC ANALYSIS:



TRANSIENT ANALYSIS:



Considering R_o (internal resistance) of n-mos

$$R_{out}=514.595k$$

$$A_v = g_m(R_d/R_{out}) = 60.5(200/514.595)$$

$$A_v = 8.76$$

AC PLOT:

$$\text{Magnitude} = 18.807 \text{ dB}$$

$$3\text{-dB magnitude} = 15.807 \text{ dB}$$

$$\text{Gain} = 18.807 \text{ dB}$$

Theoretical values:

$$A_v = 20\log(A_v) = 20\log(8.76)$$

$$= 20(0.97)$$

$$= 18.85$$

INFERENCE:

Analysing a common-source amplifier involves examining its behavior under various conditions: DC, transient, and AC.

1. DC analysis

DC analysis helps determine the operating point (Q-point) of the amplifier. This Q-point defines the biasing conditions for the transistor. A properly chosen Q-point ensures that the transistor operates in the active region

2. Transient analysis

Transient analysis reveals the amplifier's response time and how it handles input signal changes. It helps identify any transient distortions or delays in the output signal, which are crucial for ensuring signal fidelity in practical applications.

3. AC analysis

AC analysis provides insights into the amplifier's gain, bandwidth, input and output impedances, and frequency response. It helps determine the amplifier's suitability for specific frequency ranges and applications.

RESULT:

Thus the common source NMOS circuit is implemented and analyzed using candence software.