R.M.K. ENGINEERING COLLEGE
(An Autonomous Institution)
RSM NAGAR
KAVARAIPETTAI – 601 206
Academic Year 2021-2022 Even Semester

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

LABMANUAL

VLSI DESIGN LABORATORY

R.M.K. ENGINEERING COLLEGE
(An Autonomous Institution)
RSM NAGAR
KAVARAIPETTAI – 601 206
Academic Year 2021-2022 Even Semester

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

LAB MANUAL

Subject Code : EC8661
Subject Name: VLSI DESIGN LABORATORY

Prepared by

Ms. P. Latha ASP/ECE
Mr. V.Ramkumar AP/ECE
Mr.K.Nareshkumar Thapa AP/ECE

Approved by

HOD/ECE

R.M.K. ENGINEERING COLLEGE
(An Autonomous Institution)
RSM NAGAR
KAVARAIPETTAI – 601 206
Academic Year 2021-2022 Even Semester

VISION OF THE INSTITUTE

- To be the most preferred destination in the country for pursuing education in Engineering and it allied fields, at the undergraduate and post graduate levels, and for undertaking doctoral research.
- To transform learners into achievers at the global level with the right attitude towards changing societal needs.

MISSION OF THE INSTITUTE

M1: To develop the needed resources and infrastructure, and to establish a conducive ambience for the teaching- learning process

M2: To nurture in the students professional and ethical values and instill in them a spirit of innovation and entrepreneurship

M3: To encourage in the students a desire for higher learning and research to equip them to face the global challenges

M4: To provide opportunities for students to get the needed additional skills to make them industry ready

M5: To interact with industries and other organizations to facilitate transfer of knowledge and know- how

R.M.K. ENGINEERING COLLEGE
(An Autonomous Institution)
RSM NAGAR
KAVARAIPETTAI – 601 206
Academic Year 2021-2022 Even Semester

VISION OF THE DEPARTMENT

- To be one of the most sought after ***Centres of Excellence*** in the field of Electronics and Communication Engineering by providing ***high quality education***.
- To mould the students to compete internationally and to become ***excellent researchers and innovators*** who can provide solution to societal issues.

MISSION OF THE DEPARTMENT

M1: To provide the needed resources and infrastructure and to establish a ***conducive ambience for the teaching-learning and research processes*** and to meet with the technological developments.

M2: To create ***high quality professionals and entrepreneurs*** in the field of Electronics and Communication Engineering with the right attitude to serve the society with ethical values.

M3: To ***modernize the laboratories*** on par with industry standards and to collaborate with them to improve the skill set of the students for providing***innovative solutions to the industry.***

R.M.K. ENGINEERING COLLEGE
(An Autonomous Institution)
RSM NAGAR
KAVARAIPETTAI – 601 206
Academic Year 2021-2022 Even Semester

PROGRAMME EDUCATIONAL OBJECTIVES

PEO 1: Graduates will develop the ability to identify, formulate and solve challenging problems in the field of Electronics and Communication Engineering.

PEO 2: Graduates will acquire the professional skills that make them ready for immediate employment or to pursue higher studies in the related disciplines.

PEO 3: Graduates will be molded with a strong educational foundation that prepares them for leadership roles.

PEO 4: Graduates will show the understanding of impact of engineering solutions in the society and also will be aware of contemporary issues.

PEO 5: Graduates will develop confidence to communicate their ideas effectively to industry and society.

R.M.K. ENGINEERING COLLEGE
(An Autonomous Institution)
RSM NAGAR
KAVARAIPETTAI – 601 206
Academic Year 2021-2022 Even Semester

PROGRAM OUTCOMES

Program Outcome		Engineering Graduates will be able to
PO1	Engineering Knowledge	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	Problem Analysis	Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences
PO3	Design/Development of Solutions	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	Conduct Investigations of Complex Problems	Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	Modern Tool Usage	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	The Engineer and Society	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	Environment and Sustainability	Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PROGRAM OUTCOMES		
Program Outcome		**Engineering Graduates will be able to**
PO8	**Ethics**	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
PO9	**Individual and Team Work**	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings
PO10	**Communication**	Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions
PO11	**Project Management and Finance**	Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments
PO12	**Life-Long Learning**	Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

R.M.K. ENGINEERING COLLEGE
(An Autonomous Institution)
RSM NAGAR
KAVARAIPETTAI – 601 206
Academic Year 2021-2022 Even Semester

PROGRAM SPECIFIC OUTCOMES

Program Specific Outcomes (PSOs)	Electronics and Communication Engineering Graduates will be able to
PSO 1	Apply the principles of Semiconductor Devices, Digital Systems, Microprocessor and Signal Processing in the fields of Consumer Electronics, Medical, Defense and Spacecraft Electronics industry.
PSO 2	Design a variety of Computer-based components and systems for applications including Communications, Networking and Control Systems.
PSO 3	Identify indigenous components and methods for producing high quality, compact, energy efficient and eco-friendly consumer goods at an affordable price.
PSO 4	Apply acquired knowledge in Information and Communication technologies to the benefit of society

R.M.K. ENGINEERING COLLEGE
(An Autonomous Institution)
RSM NAGAR
KAVARAIPETTAI – 601 206
Academic Year 2021-2022 Even Semester

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

SYLLABUS

EC6612 VLSI DESIGN LABORATORY L T P C 0 0 3 2

OBJECTIVES:

- To learn Hardware Descriptive Language (Verilog/VHDL)
- To learn the fundamental principles of VLSI circuit design in digital and analog domain
- To familiarize fusing of logical modules on FPGAs
- To provide hands on design experience with professional design (EDA) platforms.

LIST OF EXPERIMENTS FPGA BASED EXPERIMENTS.

Part I: Digital System Design using HDL & FPGA (24 Periods)

1. Design an Adder (Min 8 Bit) using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA
2. Design a Multiplier (4 Bit Min) using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA
3. Design an ALU using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA
4. Design a Universal Shift Register using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA
5. Design Finite State Machine (Moore/Mealy) using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA
6. Design Memories using HDL. Simulate it using Xilinx/Altera Software and implement by Xilinx/Altera FPGA

Compare pre synthesis and post synthesis simulation for experiments 1 to 6.
Requirements: Xilinx ISE/Altera Quartus/ equivalent EDA Tools along with Xilinx/Altera/equivalent FPGA Boards

Part-II Digital Circuit Design (24 Periods)

7 Design and simulate a CMOS inverter using digital flow
8 Design and simulate a CMOS Basic Gates & Flip-Flops
9 Design and simulate a 4-bit synchronous counter using a Flip-Flops

Manual/Automatic Layout Generation and Post Layout Extraction for experiments 7 to 9 Analyze the power, area and timing for experiments 7 to 9 by performing Pre Layout and Post Layout Simulations.

Part-III Analog Circuit Design (12 Periods)

10. Design and Simulate a CMOS Inverting Amplifier.

11. Design and Simulate basic Common Source, Common Gate and Common Drain Amplifiers.

Analyze the input impedance, output impedance, gain and bandwidth for experiments 10 and 11 by performing Schematic Simulations.

12. Design and simulate simple 5 transistor differential amplifier. Analyze Gain, Bandwidth and CMRR by performing Schematic Simulations.

Requirements: Cadence/Synopsis/ Mentor Graphics/Tanner/equivalent EDA Tools

TOTAL :60 PERIODS

S.NO	LIST OF EXPERIMENTS
1.	Design of Half Adder and Full Adder
2.	Design of Half Subtractor and Full Subtractor
3.	Design of Encoder and Decoder
4.	Design on Multiplexer and De-Multiplexer
5.	Design of Ripple carry adder
6.	Design of Multiplier
7.	Design of Flipflops
8.	Design of Counters
9.	Design of Carry Select Adder
10.	Design of Mealy and Moore State Machines
11.	FPGA implementation of Half Adder
12.	Design of MOS differential amplifier using LT Spice
13.	Design of NOT, NAND and NOR gate using Microwind
14.	Design of CMOS inverter using CADENCE
	CONTENT BEYOND SYLLABUS
15.	Design of Pseudo Random Sequence Generator

1. DESIGN OF HALF ADDER AND FULL ADDER

AIM:

To design a half adder and a full adder using verilog HDL.

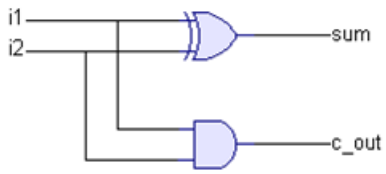
TOOLS REQUIRED:

Xilinx 8.1 version

PROCEDURE:

1. Start the Xilinx tool
2. Create project using the project wizard
3. Select verilog module for writing the code in verilog
4. Initialize the entity declaration by input and output ports entry
5. Write the architecture part of the coding in verilog for adder
6. Synthesize and rectify the errors
7. Create a test bench waveform
8. Simulate the output after giving the inputs
9. Verify the operation of the adder.

HALF ADDER CIRCUIT:



PROGRAM:

Data flow model:

```
module halfadder(i1, i2B,Sum,C_out) ;
input i1, i2;
output Sum, C_out;
assign Sum=i1^i2;
assign Carry=i1&i2;
endmodule
```

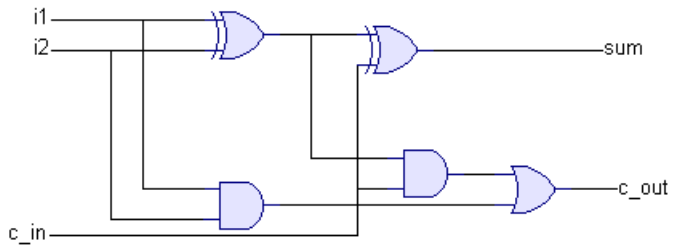
Structural model:

```
   module halfadder(sum, c_out, i1, i2);
   output sum;
   output c_out;
   input i1;
   input i2;
 xor(sum,i1,i2);
 and(c_out,i1,i2);
endmodule
```

Truth table:

i1	i2	C_out	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Full adder circuit:



PROGRAM:
Data flow model:

```
full_adder( A, B, Cin, S, Cout);
 input wire A, B, Cin;
 output reg S, Cout;
 always @(A or B or Cin)
  begin
   S = A ^ B ^ Cin;
   Cout = A&B | (A^B) & Cin;
  end
endmodule
```

Structural model:

```
module fullAdder(i1, i2, cin, cout, sum);
   input i1;
```

```
    input i2;
    input c_in;
    output c_out;
    output sum;
        wire s1,c1,c2;
        xor n1(s1,i1,i2);
        and n2(c1,i1,i2);
        xor n3(sum,s1,c_in);
        and n4(c2,s1,c_in);
        or n5(c_out,c1,c2);
endmodule
```

Truth Table:

i1	i2	C_in	C_out	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

RESULT:

Thus a half adder and a full adder were designed using verilog and verified using test bench waveform.

2. DESIGN OF HALF SUBTRACTOR AND FULL SUBTRACTOR

AIM:

To Design Half Subtractor & Full Subtractor using Verilog HDL.

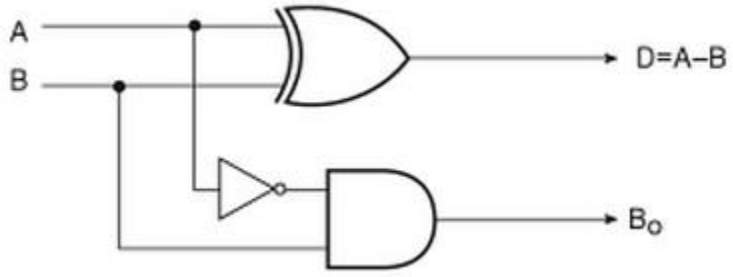
TOOLS REQUIRED:

Xilinx 8.1 version

PROCEDURE:

1. Start the Xilinx tool
2. Create project using the project wizard
3. Select verilog module for writing the code in verilog
4. Initialize the entity declaration by input and output ports entry
5. Write the architecture part of the coding in verilog for subtractor
6. Synthesize and rectify the errors
7. Create a test bench waveform
8. Simulate the output after giving the inputs
9. Verify the operation of the subtractor.

Half Subtractor Circuit:



PROGRAM:

Data flow model:

```
module hs(i1, i2B,D,B) ;
input i1, i2;
output D, B;
assign D=i1^i2;
assign Carry=~i1&i2;
endmodule
```

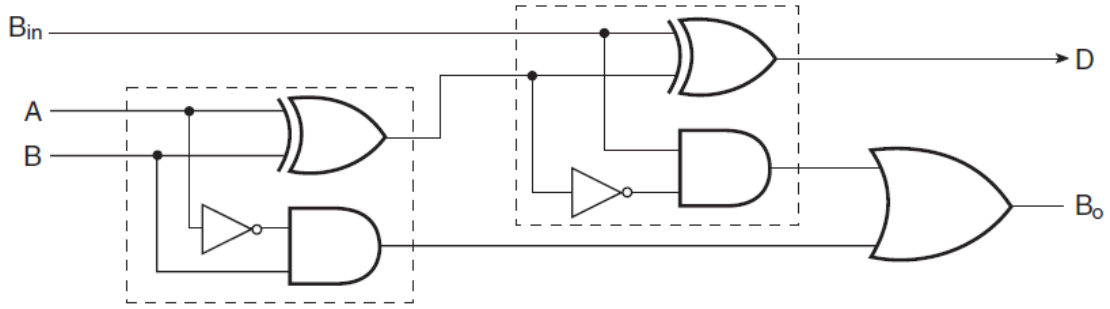
Structural model:

```
module hs(D, B, i1, i2);
    output D;
    output B;
    input i1;
    input i2;
    wire x,i1;
  xor(D,i1,i2);
  and(B,x,i2);
endmodule
```

Truth table:

i1	i2	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Full Subtractor circuit:



Program:

Data flow model:

```
module fs(i1,i2,i3,d,b);
input       i1;
input       i2;
input       i3;
output       d;
output       b;
assign d=a^b^c;
assign b = (~a)b|(~b)c|(~c)a;
endmodule
```

Truth Table:

i1	i2	i3	d	B
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

RESULT:

Thus a half Subtractor and a full Subtractor were designed using verilog and verified using test bench waveform.

3. DESIGN OF ENCODER AND DECODER

AIM:

To implement 4 x 2 Encoder and 2 x 4 Decoder Verilog HDL.

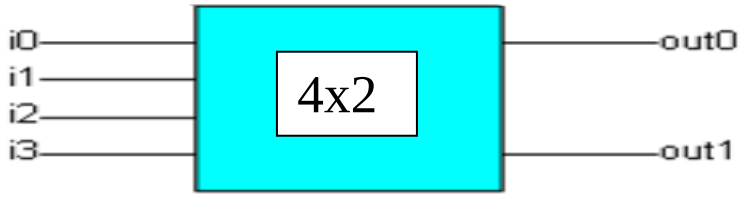
TOOLS REQUIRED:

Xilinx 8.1 version

ALGORITHM:

1. A 4 x 2 Encoder has 4 inputs and 2 outputs and 2 x 4 decoder has 2 inputs and 4 output.
2. Set the inputs i0,i1,i2 and i4 and outputs out0 and out1for encoder and inputs i0 and i1 and outputs out0,out1out2 and out3 for decoder.
3. Output is verified as per truth table.

Encoder:



Program:

```
module Encd2to4(i0, i1, i2, i3, out0, out1);
   input i0;
   input i1;
   input i2;
   input i3;
   output out0;
   output out1;
         reg out0,out1;
         always@(i0,i1,i2,i3)
           case({i0,i1,i2,i3})
                        4'b1000:{out0,out1}=2'b00;
                        4'b0100:{out0,out1}=2'b01;
                        4'b0010:{out0,out1}=2'b10;
                        4'b0001:{out0,out1}=2'b11;
                        default: $display("Invalid");
```

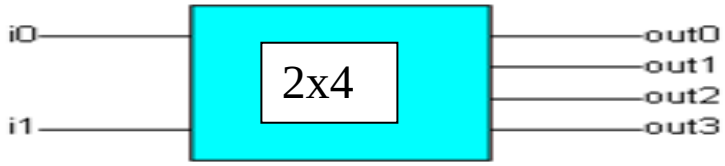
```
            endcase
endmodule
```

Truth Table:

4to2 Encoder

Input				output	
i0	i1	i2	i3	out0	out1
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

Decoder:



Program:

```
module Decd2to4(i0, i1, out0, out1, out2, out3);
   input i0;
   input i1;
   output out0;
   output out1;
   output out2;
   output out3;
        reg out0,out1,out2,out3;
        always@(i0,i1)
                case({i0,i1})
                        2'b00: {out0,out1,out2,out3}=4'b1000;
                        2'b01: {out0,out1,out2,out3}=4'b0100;
                        2'b10: {out0,out1,out2,out3}=4'b0010;
                        2'b11: {out0,out1,out2,out3}=4'b0001;
                  default: $display("Invalid");
                endcase
endmodule
```

Truth Table:

2 to 4 Decoder

Input		Output			
i0	i1	out0	out1	out2	out3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

RESULT:

Thus a 4 x 2 encoder and 2 x 4 decoder were designed using verilog and verified using test bench waveform.

4. DESIGN OF MULTIPLEXER AND DEMULTIPLEXER .

AIM:

To design a 4:1 multiplexer and 1: 4 demultiplexer using Verilog HDL.

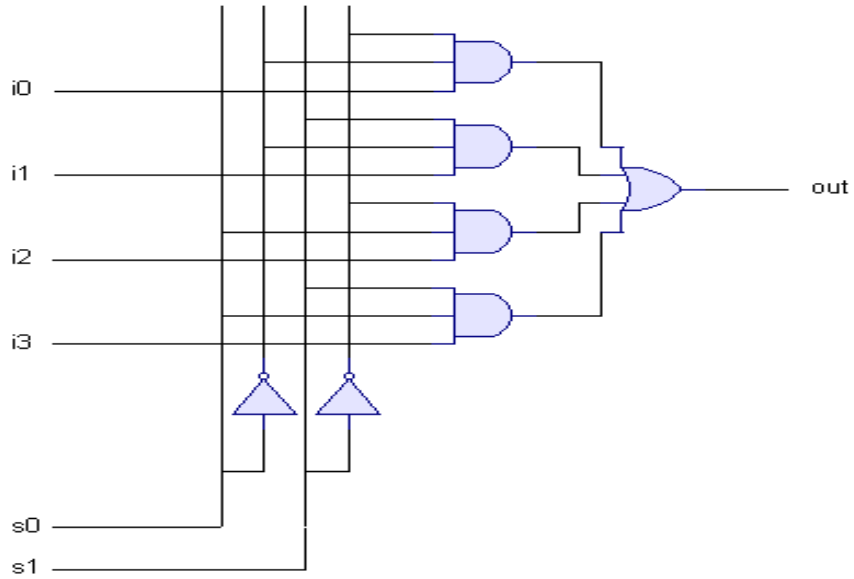
TOOLS REQUIRED:

Xilinx 8.1 version

ALGORITHM:

1. Set the inputs as i1,i2,i3,i4 and select lines as s1 & s2..
2. Set the output as out.
3. A Multiplexer is a data selector which selects one of the (‘2^n’) data based on the status of the (n)select lines and passes it to the output.

CIRCUIT DIAGRAM



PROGRAM:

Using if statement:

```
module MUX(i1,i2,i3,i4,s1,s2,out) ;
output out;
input i1,i2,i3,i4,s1,s2;
reg out;
always@(i1 or i2 or i3 or i4 or s1 or s2)
begin
    if (s == 2'b00)
      o = a;
    else if (s == 2'b01)
      o = b;
    else if (s == 2'b10)
      o = c;
    else
      o = d;
  end
endmodule
```

Using case statement

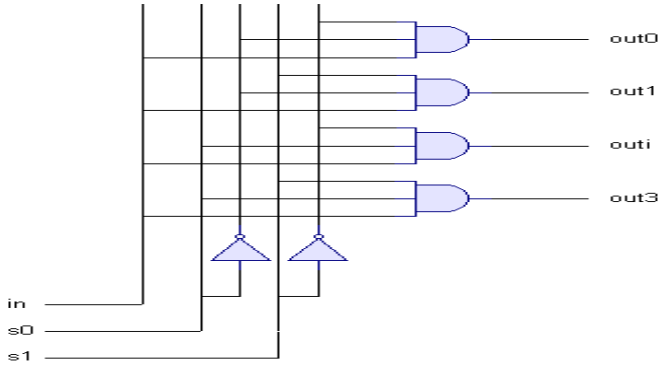
```
module MUX(i1,i2,i3,i4,s1,s2,out) ;
output out;
input i1,i2,i3,i4,s1,s2;
reg out;
always@(i1 or i2 or i3 or i4 or s1 or s2)
begin
        case({s1,s2})
        2'b00:out = i1;
        2'b01:out = i2;
        2'b10:out = i3;
        2'b11:out = i4;
        endcase
end
endmodule
```

Truth Table:

4to1 Multiplexer

Select lines		output
s0	s1	out
0	0	i0
0	1	i1
1	0	i2
1	1	i3

Demultiplexer:



Program:

```
module Dux1to4(in, s0, s1, out0, out1, out2, out3);
   input in;
   input s0;
   input s1;
   output out0;
   output out1;
   output out2;
   output out3;
         wire s0n,s1n;
         not(s0n,s0);
         not(s1n,s1);
         and (out0,in,s1n,s0n);
         and (out1,in,s1n,s0);
         and (out2,in,s1,s0n);
         and (out3,in,s1,s0);
```

endmodule

Truth Table:

Demultiplexer

Input	Select lines		output			
in	s0	s1	out0	out1	out2	out3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

RESULT:

A 4:1 Multiplexer and 1:4 demultiplexer were designed using verilog and verified using test bench waveform.

5. DESIGN OF RIPPLE CARRY ADDER

AIM:

To design a 8 bit Ripple Carry Adder using verilog HDL.

TOOLS REQUIRED:

Xilinx 8.1 version

ALGORITHM:

1. Set the inputs as A & B, where A & B are 8 bit binary numbers.
2. Set the outputs as sum & carry, each 8 bit.
3. A Ripple carry adder is an n-bit adder in which the carry in one stage is used (rippled) in the next stage as the third input for addition.
4. Using the half adder and full adder modules as templates, a ripple carry adder is designed using instantiation.

Program:

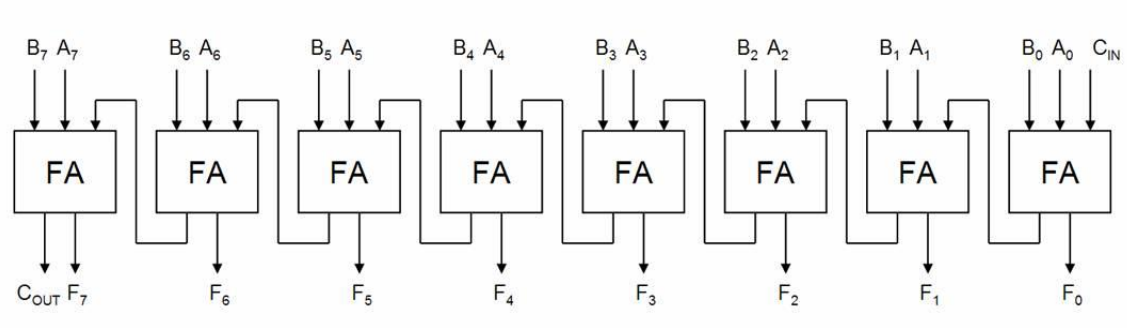
Structural model

```
module RCA(A,B,Sum,Carry) ;
output [7:0]Sum, [7:0]Carry;
input [7:0]A,[7:0] B;
halfadder add1(A[0],B[0],Sum[0],Carry[0]);
full_adder add2(A[1],B[1],Carry[0],Sum[1],Carry[1]);
full_adder add3(A[2],B[2],Carry[1],Sum[2],Carry[2]);
full_adder add4(A[3],B[3],Carry[2],Sum[3],Carry[3]);
full_adder add4(A[4],B[4],Carry[3],Sum[4],Carry[4]);
full_adder add4(A[5],B[5],Carry[4],Sum[5],Carry[5]);
full_adder add4(A[6],B[6],Carry[5],Sum[6],Carry[6]);
full_adder add4(A[7],B[7],Carry[6],Sum[7],Carry[7]);
endmodule
```

Data flow model

```
module adder(a, b, ci, sum, co);
      input ci;
      input  [7:0] a;
      input  [7:0] b;
      output [7:0] sum;
      output co;
wire [8:0] tmp;
        assign tmp = a + b + ci;
        assign sum = tmp [7:0];
        assign co  = tmp [8];
      endmodule
```

Circuit:



RESULT:

Thus a Ripple carry adder was designed using verilog and verified using test bench waveform.

6. DESIGN OF A PARALLEL MULTIPLIER

AIM:

To design and simulate a 4 bit multiplier using verilog.

TOOLS REQUIRED:

Xilinx 8.1 version

ALGORITHM:

1. Set the inputs as a & b, where a & b are 4 bit binary numbers.
2. Set the outputs as out, where out is the 8 bit product
3. Using data flow model, multiplier is designed.

VERILOG CODE:

```
module multiplier(a, b, out);
   input [4:0] a;
   input [4:0] b;
   output [9:0] out;
assign out=(a*b);
endmodule
```

PROGRAM:

Component:

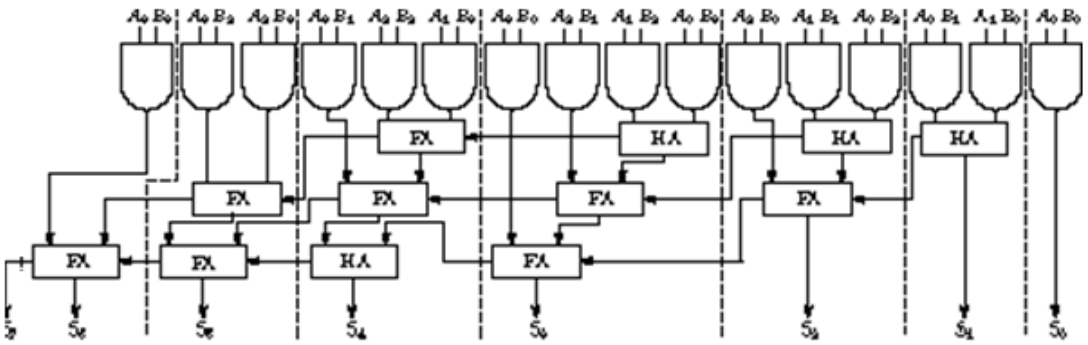
```
module HA(sout,cout,a,b);
output sout,cout;
input a,b;
assign sout=a^b;
assign cout=(a&b);
endmodule
module FA(sout,cout,a,b,cin);
```

```
output sout,cout;
input a,b,cin;
assign sout=(a^b^cin);
assign cout=((a&b)|(a&cin)|(b&cin));
endmodule
```

Top Module:

```
module multiply4bits(product,inp1,inp2);
output [7:0]product;
input [3:0]inp1, [3:0]inp2;
assign product[0]=(inp1[0]&inp2[0]);
wirex1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17;
 HA HA1(product[1],x1,(inp1[1]&inp2[0]),(inp1[0]&inp2[1]));
FA FA1(x2,x3,inp1[1]&inp2[1],(inp1[0]&inp2[2]),x1);
FAFA2(x4,x5,(inp1[1]&inp2[2]),(inp1[0]&inp2[3]),x3);
HA HA2(x6,x7,(inp1[1]&inp2[3]),x5);
HA HA3(product[2],x15,x2,(inp1[2]&inp2[0]));
FA FA5(x14,x16,x4,(inp1[2]&inp2[1]),x15);
FA FA4(x13,x17,x6,(inp1[2]&inp2[2]),x16);
FA FA3(x9,x8,x7,(inp1[2]&inp2[3]),x17);
HA A4(product[3],x12,x14,(inp1[3]&inp2[0]));
FAFA8(product[4],x11,x13,(inp1[3]&inp2[1]),x12);
FA FA7(product[5],x10,x9,(inp1[3]& inp2[2]),x11);
FAFA6(product[6],product[7],x8,(inp1[3]&inp2[3]),x10); endmodule
```

Circuit:



RESULT:

Thus a 4 bit multiplier was designed using verilog and verified using test bench waveform.

7. DESIGN OF D FLIPFLOP AND T LATCH USING VERILOG

AIM:

To design a data flipflop and a data latch using verilog.

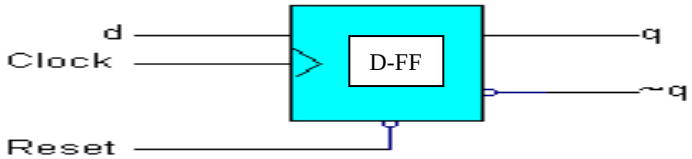
TOOLS REQUIRED:

Xilinx 8.1 version

PROCEDURE:

1. Start the Xilinx tool
2. Create project using the project wizard
3. Select verilog module for writing the code in verilog
4. Initialize the entity declaration by input and output ports entry
5. Write the architecture part of the coding in verilog for D and T flipflop
6. Synthesize and rectify the errors
7. Create a test bench waveform
8. Simulate the output after giving the inputs

D Flip-Flop block diagram:



PROGRAM:

// **D FLIPFLOP**

```
module dff(d,Reset,clock,q1,q2) ;
input d,clock,Reset;
output q1,q2;
reg q1,q2;
always@(posedge clock)
begin
if(Reset)
```

```
begin
q1=0;
q2=1;
end
else
begin
q1=d;
q2=~d;
end
end
endmodule
```

TRUTH TABLE:

Input			Output	
Reset	clock	d	q1	q2
1	↑	0	0	1
1	↑	1	0	1
0	↑	0	0	1
0	↑	1	1	0

T FLIPFLOP:

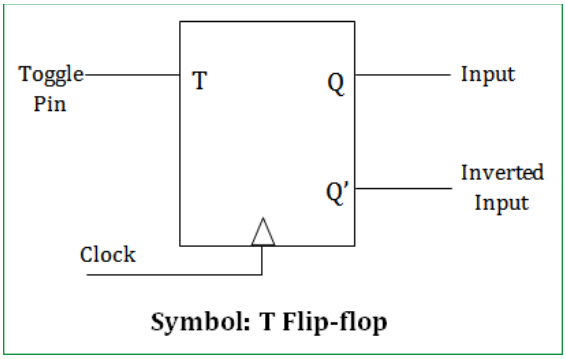
PROGRAM:

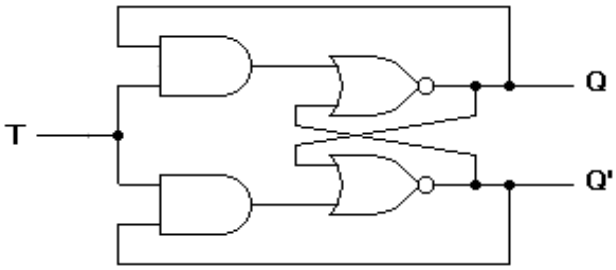
```
module TFF(Clock, Reset, t, q);
  input Clock;
  input Reset;
  input t;
  output q;
   reg q;
   always@(posedge Clock , negedge Reset)
        if(~Reset) q=0;
        else if (t) q=~q;
```

else q=q;

endmodule

BLOCK DIAGRAM:





TRUTH TABLE:

Q(t)	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

RESULT:

Thus a data flipflop and a toggle flipflop were designed using Verilog and verified using test bench waveform.

8. DESIGN OF SYNCHRONOUS COUNTER.

AIM:

To design a 4 bit synchronous counter using verilog HDL.

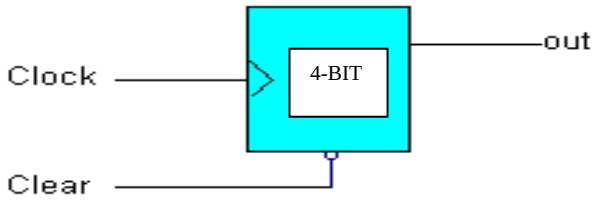
TOOLS REQUIRED:

Xilinx 8.1 version

ALGORITHM:

1. A counter is a circuit used to count the number of events.
2. It consists of two inputs clk and clear and one 4 bit output q.
3. This counter counts from 0000 to 1111 in increasing order

BLOCK DIAGRAM:



PROGRAM:

```
module upcountert(clk, clear, q);
   input clk;
   input clear;
   output [3:0] q;
    reg [3:0] q;
   always@(posedge clear or posedge clk)
   begin
   if(clear)
   q <=4'b0000;
   else
   q <= q+1'b1;
   end
endmodule
```

TRUTH TABLE:

clear	clk	q3	q2	q1	q0
1	↑	0	0	0	0
0	↑	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

RESULT:

Thus a 4 bit up counter was designed using verilog and verified using test bench waveform.

9. Design of 8 bit carry select adder

AIM

To Design a 8 bit carry select adder by using verilog HDL.

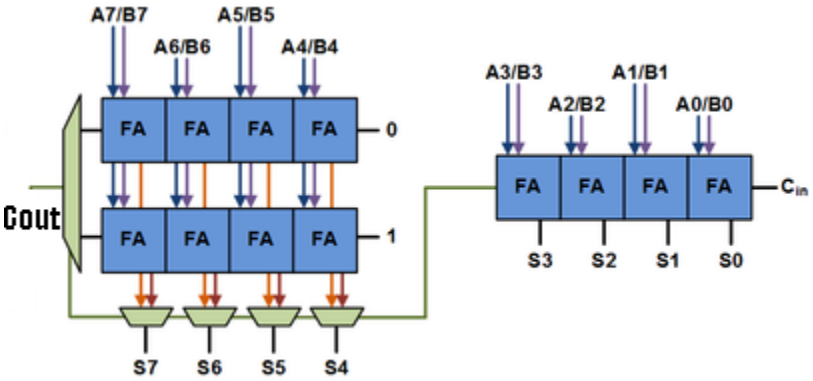
TOOLS REQUIRED:

- Xilinx 8.1 version

THEORY

A simple carry select adder is a digital circuit that produces the arithmetic sum of n bit numbers. To implement 8 bit addition, three ripple carry adders are required. One ripple carry adder is used to perform the least 4bits addition and remaining two ripple carry adders are to calculate most 4bits addition in order to reduce the propagation delay. In this two 4-bit ripple carry adders are multiplexed together, where the resulting carry and sum bits are selected by the carry-in. Since one ripple carry adder assumes a carry-in of 0, and the other assumes a carry-in of 1, selecting which adder had the correct assumption via the actual carry-in yields the desired result.

LOGIC CIRCUIT:



PROCEDURE:

1. Start the Xilinx tool
2. Create project using the project wizard

3. Select Verilog module for writing the code in verilog
4. Initialize the entity declaration by input and output ports entry
5. Write the architecture part of the coding for 8bit carry select adder.
6. Synthesize and rectify the errors
7. Create a test bench waveform
8. Simulate the output after giving the inputs
9. Verify the operation of the 8bit carry select adder.

PROGRAM

```
module csa(a, b, cin, sum, cout);
   input [7:0] a;
   input [7:0] b;
   input cin;
   output [7:0]sum;
   output cout;
         wire c,c_top,c_bot;
         wire [3:0]s_top,s_bot;
         rcast
         r1(a[3:0],b[3:0],cin,sum[3:0],c),
         r2(a[7:4],b[7:4],0,s_bot[3:0],c_bot),
         r3(a[7:4],b[7:4],1,s_top[3:0],c_top);
         mux2
         m1(s_top[0],s_bot[0],c,sum[4]),
         m2(s_top[1],s_bot[1],c,sum[5]),
         m3(s_top[2],s_bot[2],c,sum[6]),
         m4(s_top[3],s_bot[3],c,sum[7]),
         m5(c_top,c_bot,c,cout);
         endmodule
```

Sub Module 1:

```
module rcast(a, b, c, s, ca);
   input [3:0] a;
   input [3:0] b;
   input c;
   output [3:0] s;
   output ca;
         wire c1,c2,c3;
         fulladddataflow
         f1(a[0],b[0],c,s[0],c1),
         f2(a[1],b[1],c1,s[1],c2),
         f3(a[2],b[2],c2,s[2],c3),
         f4(a[3],b[3],c3,s[3],ca);
endmodule
```

Sub Module 2:

```
module mux2(i0, i1, s0, y);
   input i0;
   input i1;
   input s0;
   output y;
          wire s0bar,p,q;
          not
          n1(s0bar,s0);
          and
          a1(p,i0,s0bar),
          a2(q,i1,s0);
          or
          o1(y,p,q);
endmodule
```

Sub Module 3:

```
module fulladddataflow( a , b, cin , sum, cout );
input a , b , cin;
output sum, cout;
assign sum = a ^ b ^cin;
assign cout = (a&b)|(b&cin)|(cin&a);
endmodule
```

RESULT:

Thus a 8 carry save adder was designed using verilog and verified using test bench waveform.

10. DESIGN OF MEALY AND MOORE STATE MACHINES

AIM

To develop the source code for Mealy and Moore state machines by using verilog..

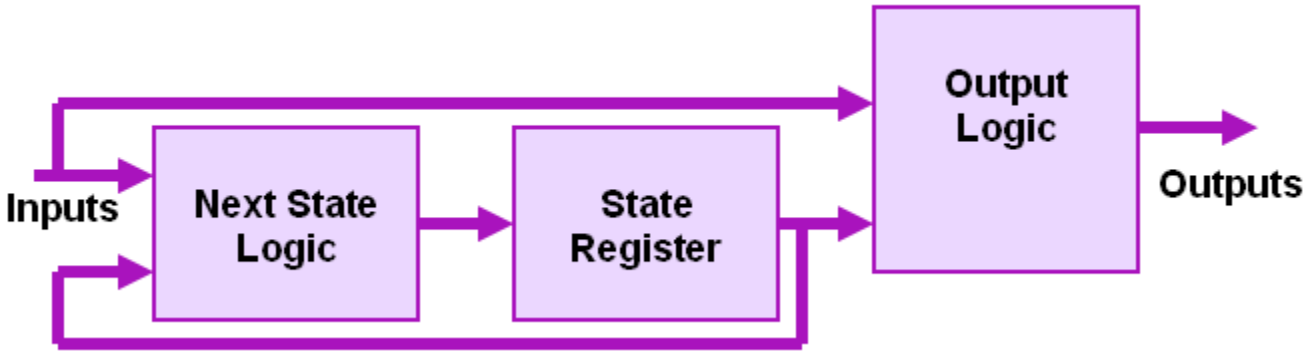
TOOLS REQUIRED:

- Xilinx 8.1 version

THEORY:

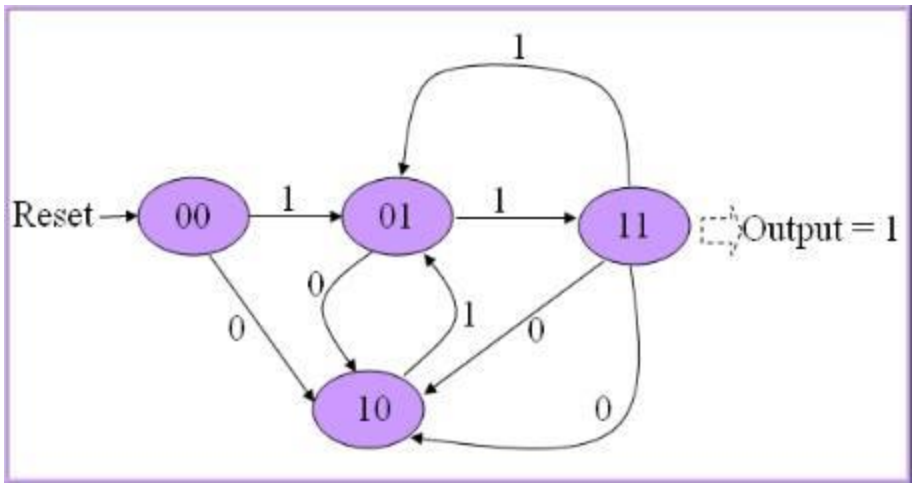
There are two basic ways to implement sequential circuit. They are Moore and Mealy state machines. In Moore state machine, the output depends on the present state, whereas in Mealy state machine, the output depends on both present state and inputs.

Moore State Machine



Mealy State Machine

Moore State machine:



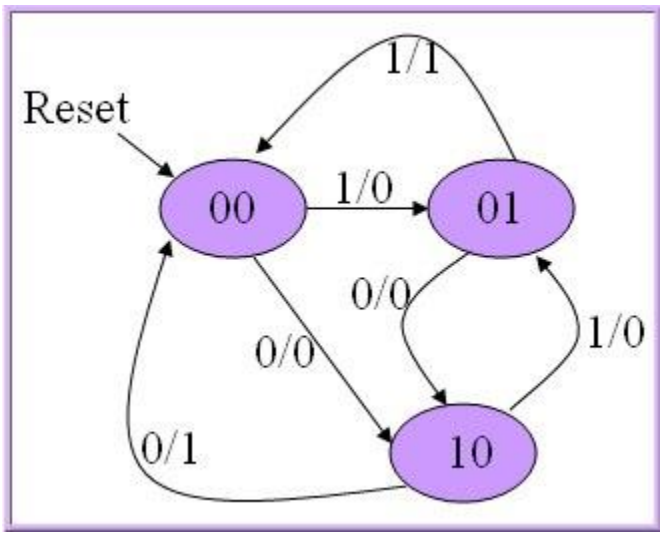
Here is a Moore type state transition diagram for the circuit. When reset, state goes to 00; If input is 1, state will be 01 and if input is 0, state goes to 10. State will be 11 if input repeats. After state 11, goes to 10 state or 01 depending on the inp, since overlapping pair should not be considered. That is, if 111 comes, it should consider only one pair.

Program for Moore Machine:

```
module fsm( clk, rst, inp, outp);
input clk, rst, inp;
output outp;
reg [1:0] state;
reg outp;
always @( posedge clk, posedge rst )
begin
if( rst )
    state <= 2'b00;
else
begin
case( state )
    2'b00:
begin
if( inp ) state <= 2'b01;
```

```
else state <= 2'b10;
end
    2'b01:
begin
if( inp ) state <= 2'b11;
else state <= 2'b10;
end
    2'b10:
begin
if( inp ) state <= 2'b01;
else state <= 2'b11;
end
    2'b11:
    begin
if( inp ) state <= 2'b01;
else state <= 2'b10;
end
endcase
end
end
always @(posedge clk, posedge rst)
begin
if( rst )
    outp <= 0;
elseif( state == 2'b11 )
    outp <= 1;
else outp <= 0;
end
endmodule
```

Mealy State machine:



When reset, state becomes idle, that is 00. Next, if 1 comes, state becomes 01 and if 0 comes state becomes 10 with output 0. We have showed input 1, output 0 as 1/0. If input bit repeats, output becomes 1 and state goes to 00.

Program for Mealy Machine:

```
module mealy( clk, rst, inp, outp);
input clk, rst, inp;
output outp;
reg [1:0] state;
reg outp;
always @( posedge clk, posedge rst ) begin
if( rst ) begin
      state <= 2'b00;
      outp <= 0;
end
elsebegin
case( state )
      2'b00: begin
 if( inp ) begin
            state <= 2'b01;
            outp  <= 0;
end
elsebegin
             state <= 2'b10;
             outp <= 0;
end
end
```

```
      2'b01: begin
if( inp ) begin
              state <= 2'b00;
              outp  <= 1;
end
elsebegin
             state <= 2'b10;
             outp <= 0;
end
end

      2'b10: begin
if( inp ) begin
              state <= 2'b01;
              outp  <= 0;
end
elsebegin
             state <= 2'b00;
             outp <= 1;
    end
end

default: begin
           state <= 2'b00;
           outp <= 0;
end
endcase
end
end

endmodule
```

PROCEDURE:

1. Start the Xilinx tool
2. Create project using the project wizard
3. Select verilog module for writing the code in verilog
4. Initialize the entity declaration by input and output ports entry
5. Write the architecture part of the coding for Moore and Mealy state machines.
6. Synthesize and rectify the errors
7. Create a test bench waveform
8. Simulate the output after giving the inputs

9. Verify the operation of the Moore and Mealy state machines.

SIMULATION OUTPUT:

RESULT: Thus the Mealy and Moore state machines was designed using verilog and verified using test bench waveform.

11.FPGA IMPLEMENTATION OF HALF ADDER

AIM:

To design a half adder and implement it in FPGA kit.

TOOLS REQUIRED:

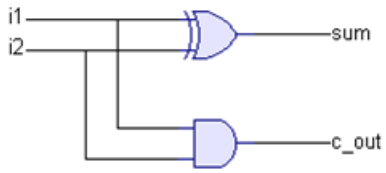
Xilinx 8.1 version

Spartan 3E trainer kit

PROCEDURE:

1. Start the Xilinx tool
2. Create project using the project wizard
3. Select verilog module for writing the code in verilog
4. Initialize the entity declaration by input and output ports entry
5. Write the architecture part of the coding in verilog for adder
6. Synthesize and rectify the errors
7. Create a test bench waveform
8. Simulate the output after giving the inputs
9. Verify the operation of the adder.
10. Assign the package pins
11. Perform place, map and routing functions.
12. Download the program on the kit and verify its functionality

Half Adder circuit:



VERILOG CODE:

Data flow model:

```
module halfadder(i1, i2B,Sum,C_out) ;
input i1, i2;
output Sum, C_out;
assign Sum=i1^i2;
assign Carry=i1&i2;
endmodule
```

Structural model:

```
module halfadder(sum, c_out, i1, i2);
output sum;
output c_out;
input i1;
input i2;
xor(sum,i1,i2);
 and(c_out,i1,i2);
endmodule
```

Truth table:

i1	i2	C_out	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

RESULT:

Thus a half adder was designed using verilog and verified using FPGA kit.

12. LAYOUT DESIGN OF NOT, NAND AND NOR GATE USING MICROWIND

Aim:

To develop the layout design for CMOS inverter, NAND and NOR gate by using MICROWIND and obtain the simulation results.

Software tools required:

The MICROWIND program allows designing and simulating an integrated circuit at physical description level. The package contains a library of common logic and analog ICs to view and simulate.

Theory:

PURPOSE OF DESIGN RULES:

As all the layouts are prepared with minimum λ design rules .With minimum λ , layout will require less amount of area on chip, so larger circuits can be implemented with in less area. Makes the circuit to consumes less amount power i.e. as area and polysilicon usage can also reduced, resulting less resistance ,which in turn causes to dissipate less amount of power. Design rules will give optimized layout.

(i) PMOS TRANSISTOR

1. The complete window can be considered as P-well.
2. The N-well is placed over the p-well in a rectangular box.
3. Next we need to place the P-diffusion over the N-well.
4. Width of the P-diffusion should be 4λ.
5. The metal contacts can be drawn by selecting a (4λ*4λ) square.

(ii) NMOS TRANSISTOR

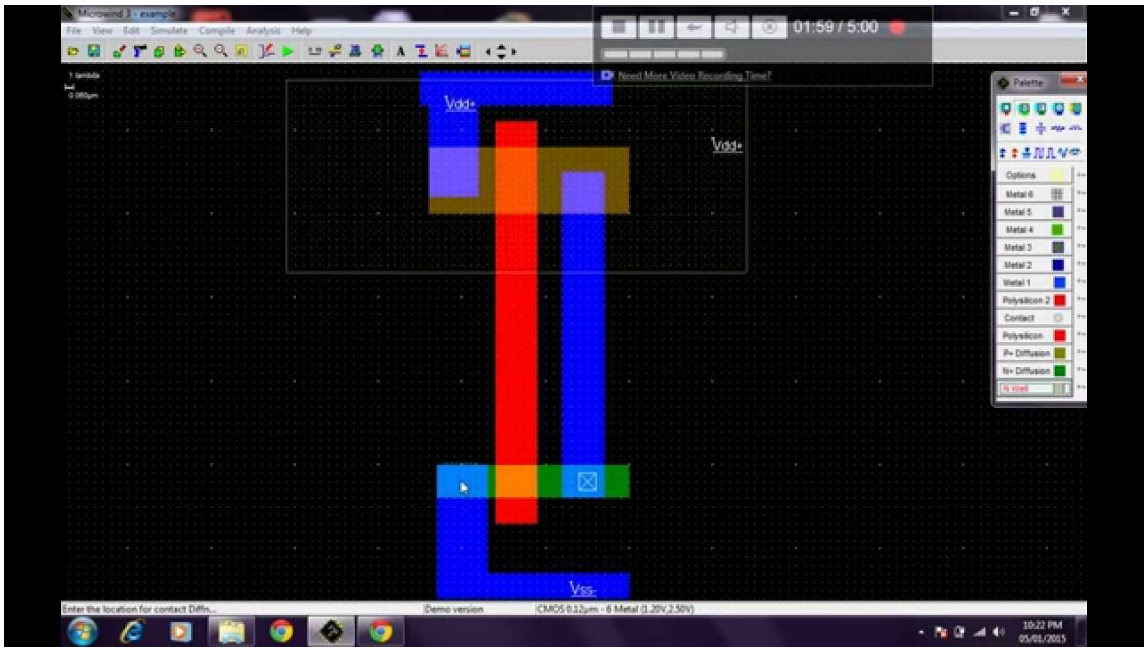
1. There should be minimum 6λ space between N-well and N-diffusion.
2. The same steps can be followed for NMOS as in PMOS.
3. The grid itself is P-well, we can start directly with n-diffusion layer.

4. The PMOS and NMOS should be connected by using polysilicon layer of width 2λ.
5. The polysilicon layer should be extended up to 3λ length above and below P and N diffusion layers.
6. Leave minimum 1λ space on both sides of polysilicon between the contacts.
7. The source to drain connection is made using a metal of width 3λ.

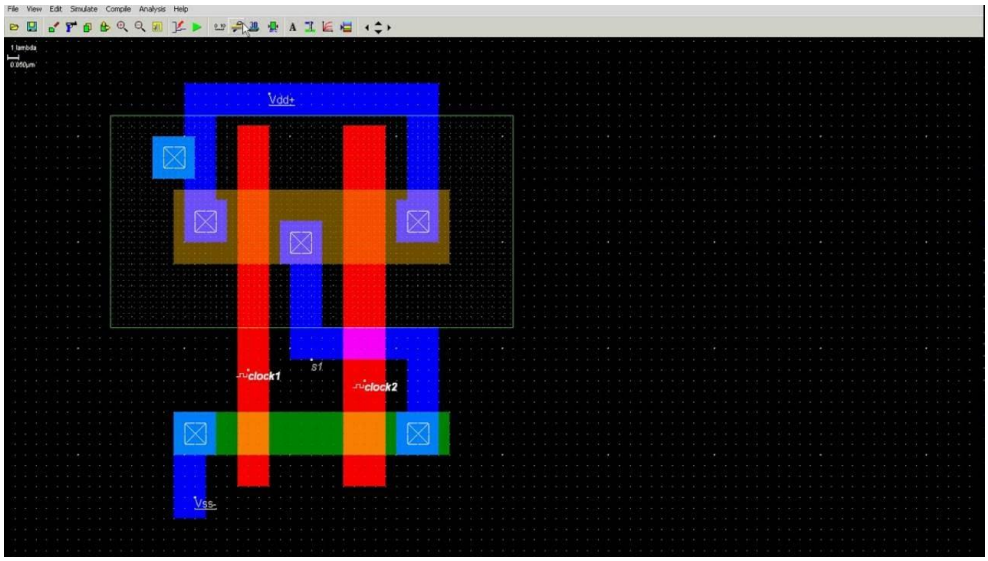
PROCEDURE:

1. Select Microwind, microwind tools expert DCSH2DSCH
2. Select filenew and draw inverter diagram
3. Start simulation ,then click simulation
4. Choose the option shown at the input pins
5. By clicking on the value of X in the fig, we can view the output of inverter for input 0 and 1.
6. Then save and give the file name by clicking the file
7. Then convert this in to verilog file by clicking the file and make as verilog file
8. Compile the verilog file and generate the layout
9. Select the layout and run the process

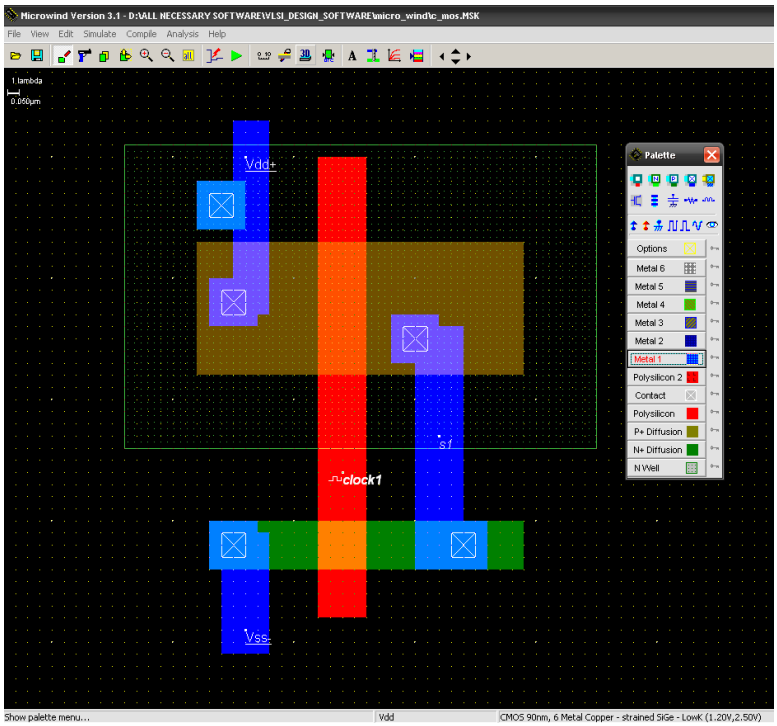
Inverter:



NAND :



NOR :



Result :

Thus the CMOS Inverter, NAND and NOR gate are designed using MICROWIND and the results are simulated.

13. DESIGN OF MOS DIFFERENTIAL AMPLIFIER USING LT SPICE

AIM:

To design a MOS differential amplifier using LT SPICE.

TOOLS REQUIRED:

LT SPICE

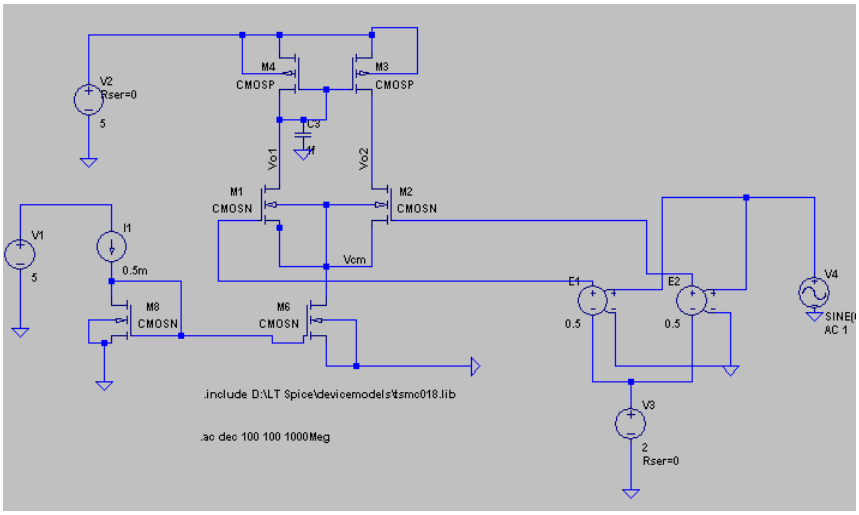
THEORY:

1. MOS differential amplifier is an amplifier which amplifies the difference between the voltages of two signals.
2. It acts in two modes namely the common mode and differential mode.
3. CMRR can be calculated from the difference between common mode gain and differential mode gain.
4. Common mode gain and differential mode gain can be calculated from the response.

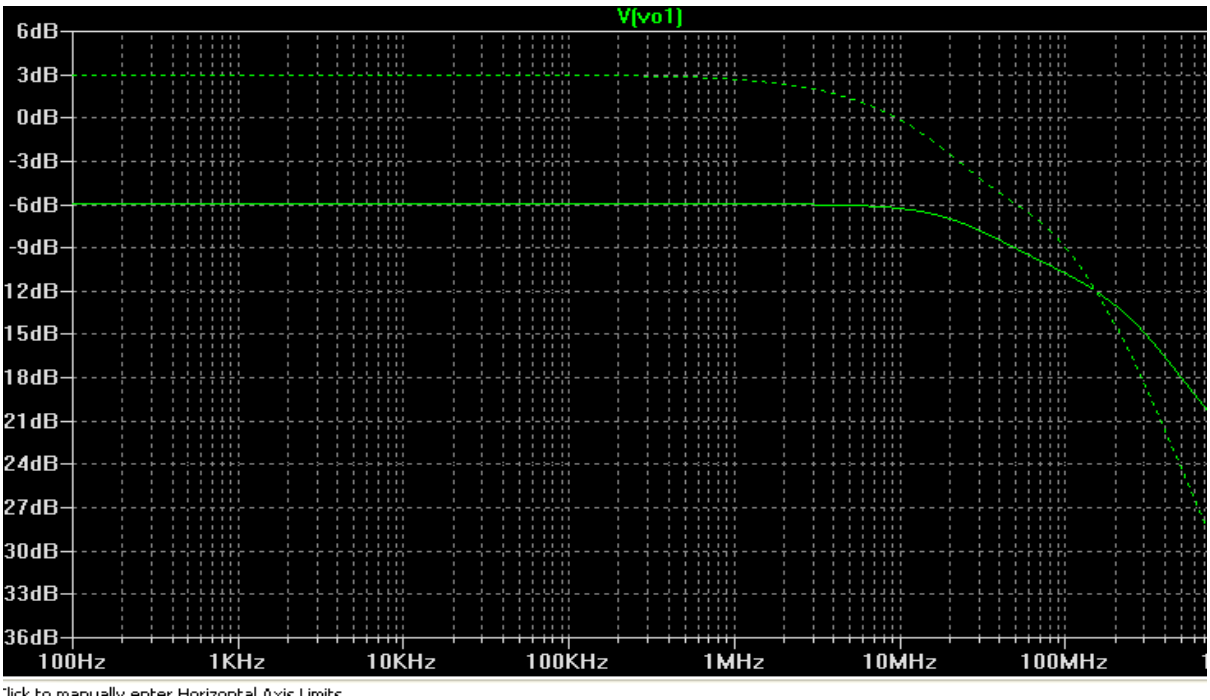
PROCEDURE:

1. Open LT spice.
2. Given the connection as per the circuit diagram.
3. Give the inputs for common mode and differential mode separately.
4. Notice the response from the Graph and determine the gain for both differential mode and common mode
5. Calculate CMRR from the formula.

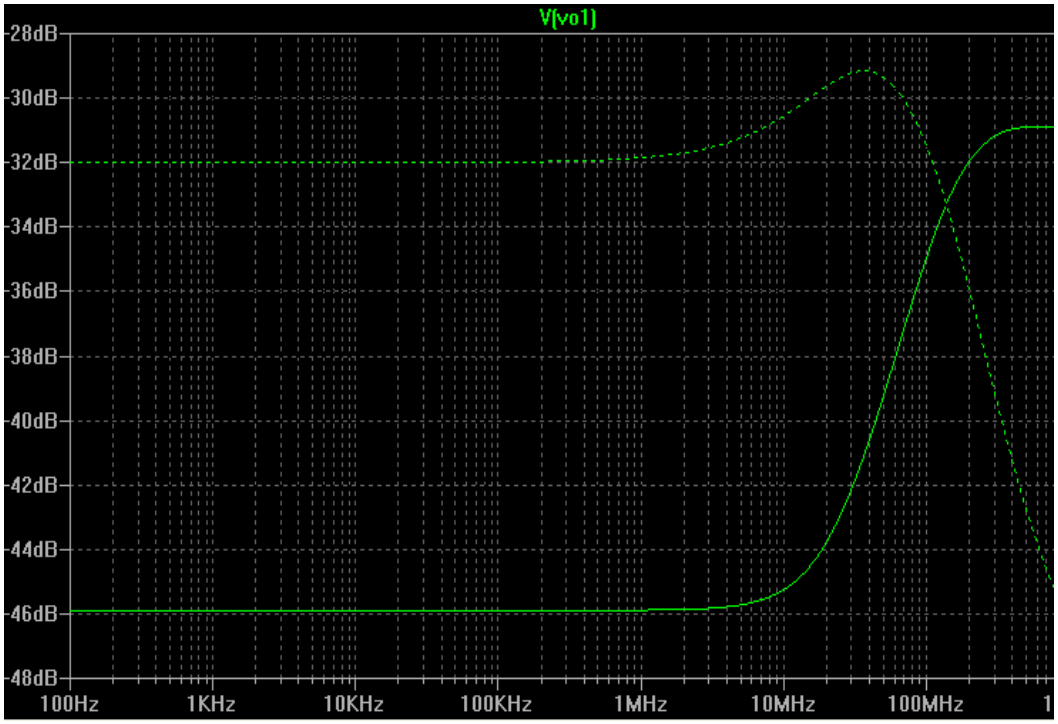
CIRCUIT DIAGRAM:



DIFFERENTIAL MODE RESPONSE



COMMON MODE RESPONSE:



CALCULATION:

CMRR = A_d - A_c

A_d – Differential Mode Gain

A_c – Common Mode Gain

RESULT :

Thus a MOS differential amplifier using LT SPICE was designed and CMRR was calculated.

14. DESIGN OF CMOS INVERTER USING CADENCE

AIM :

To design an CMOS inverter using CADENCE Virtuoso.

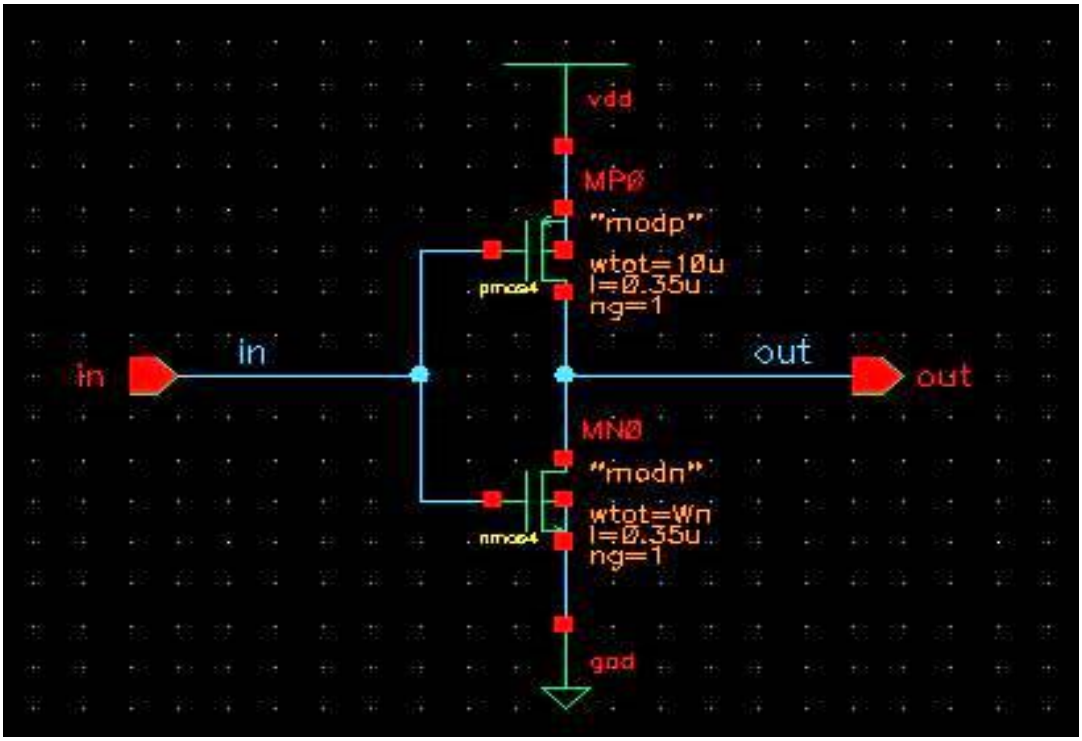
SOFTWARE REQUIRED :

- CADENCE Virtuoso software
- Linux OS

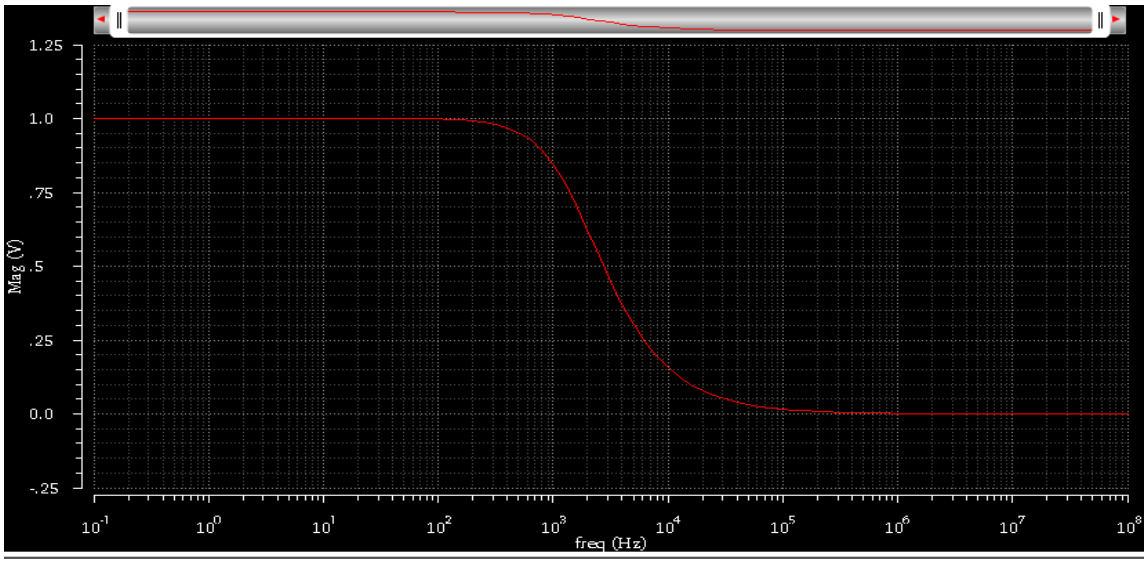
PROCEDURE :

- Start the CADENCE Virtuoso in linux environment.
- Create a new file and add gpdk 180 library to it.
- Place the components required for inverter using instance tool box.
- Connect the components using wire command.
- Give values for the power supply and input.
- Check the design for any errors.
- Run transitent analysis and verify the functionality of inverter using the waveform.

CIRCUIT :



WAVEFORM :



RESULT :

Thus the CMOS inverter is designed and verified using CADENCE Virtuoso.

15. DESIGN OF PRBS GENERATOR

AIM:

To design a PRBS generator using verilog.

TOOLS REQUIRED:

Xilinx 8.1 version

ALGORITHM:

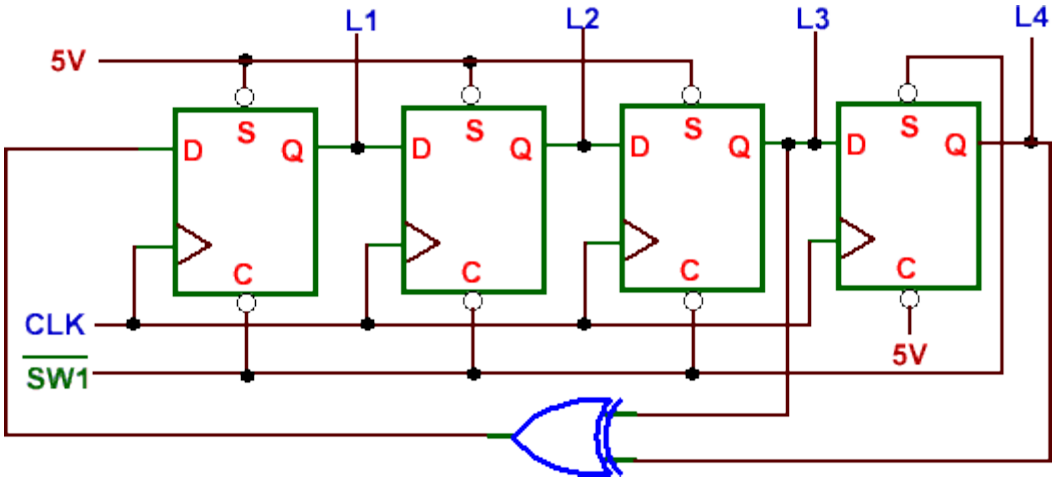
1. PRBS (Pseudo Random Binary Sequence) generator is constructed using LFSR (Linear Feedback Shift Register).
2. An LFSR shifts the input bits to the right. The last bit is XNOR with first bit.
3. It consists of inputs d, clk and reset and 4-bit output q.
4. Initially acc is 8 bit zero and adds the input repeatedly.

VERILOG CODE:

```
module prbs(d, clk, reset, q);
  input [3:0] d;
  input clk;
  input reset;
  output [3:0] q;
        reg [3:0]q;
        reg [3:0]temp;
        always @(clk)
        begin
        if(reset)
        q=4'b0000;
        else
        begin
        temp=d;
        temp[3]=temp[0]^temp[3];
        q=temp<<1;
        end
```

```
end
endmodule
```

CIRCUIT:



RESULT:

Thus a PRBS generator was designed using verilog and verified using test bench waveform.