JADAVPUR UNIVERSITY

IC DESIGN AND FABRICATION CENTRE

SUMMER TRAINING ON MICRO-ELECTRONICS TECHNOLOGY AND VLSI DESIGN

PRACTICAL REPORT
ON
VERILOG & FPGA LAB

BY~

INSTITUTE OF ENGINEERING AND MANAGEMENT, KOLKATA(IEM)

GROUP – 1

TEAM MEMBERS

NAMES	STREAM/ YEAR
APARAJITA ROY	ECE/ 3rd Year
SOMAK KUMAR	ECE/ 3rd Year
ABHISHEK ANAND	ECE/ 3rd Year
SAYANTAN SARKAR	ECE/ 3rd Year
SOURAJIT NAYAK	ECE/ 3rd Year
LAWAZ ISLAM	ECE/ 3rd Year
DEBAYAN SENGUPTA	ECE/ 3rd Year
VISHWAVIJAY SINGH	ECE/ 3rd Year

ACKNOWLEDGEMENT

We received a lot of information and advice from some well-respected people who deserve our gratitude for helping us with our lab studies. As a result, I'd want to express our heartfelt gratitude to everyone who has helped us along the way, both directly and indirectly. First and foremost, I want to express my gratitude to respected Professors, who introduced us to the work approach and whose sincere advice, inspiration, and participation opened the path for the effective completion of our report. I'd like to express my gratitude to him in particular for sharing the ups and downs of the development process and putting up with the inconvenience. I'd also like to offer my heartfelt gratitude to all the members of Jadavpur University for their cooperation, recommendations, and unwavering support. Most significantly, by allowing us the access to the IC centre and its technologies to us, have provided valuable feedback on this proposal, inspiring us to prepare ourselves for a better future. Last but not the least, I appreciate everyone's assistance in performing our lab experiments.

VERILOG HDL

VERILOG HDL

Verilog is a Hardware Description Language (HDL) which is used to model electronic systems. It is commonly used for design and verification of digital circuits. It is similar in syntax to C programming language. Designers with C programming experience will find it easy to grasp this language. Other than other HDL, Verilog runs syntax concurrently but in other HDL syntax is followed stepwise.

Verilog allows different levels of abstraction to be mixed in the same model. Thus, a designer can mix hardware model in terms of gates, switches, RTL, behavioural or structural code.

Verilog supports a design at many levels of abstraction. The major three are –

- **Behavioural level**
- **Data Flow level**
- **Gate level**

BEHAVIOURAL OR ALGORITHMIC LEVEL

This is the highest level of abstraction provided by Verilog HDL. It uses procedural statements (use of the "always" block, case, if statements) to describe the circuit. At behavioural level, the circuit is described using an algorithm which consists of a set of instructions executed one after the another in a sequential manner.

REGISTER-TRANSFER LEVEL

Register Transfer Level (RTL) is an abstraction for defining the digital portions of a design. It is the principle abstraction used for defining electronic systems today and often serves as the golden model in the design and verification flow. The RTL design is usually captured using a hardware description language (HDL) such as Verilog or VHDL. While these languages are capable of defining systems at other levels

of abstraction, it is generally the RTL semantics of these languages, and indeed a subset of these languages defined as the synthesizable subset.

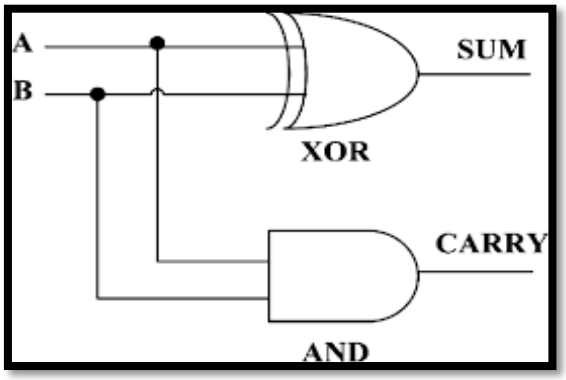
GATE LEVEL OR STRUCTURAL LEVEL

The module is implemented in terms of logic gales and interconnection between these gates. It resembles a schematic drawing with components com1ected with signals. Since logic gate is most popular component, Verilog has a predefined set of logic gates known as primitives. Any digital circuit can be built from these primitives.

❖ HALF ADDER

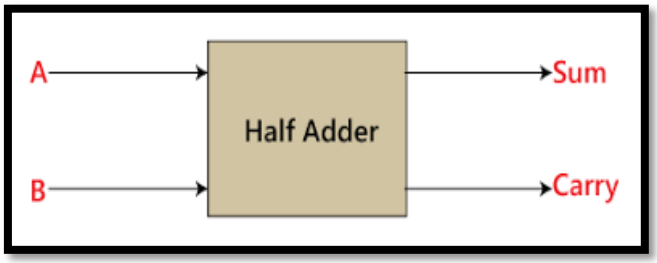
A half adder is a type of adder, an electronic circuit that performs the addition of numbers. The half adder is able to add two single binary digits and provide the output plus a carry value. It has two inputs, called A and B, and two outputs S (sum) and C (carry). The common representation uses a XOR logic gate and an AND logic gate.

CIRCUIT DIAGRAM:



`

BLOCK DIAGRAM:



TRUTH TABLE:

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Sum=A'B+AB'

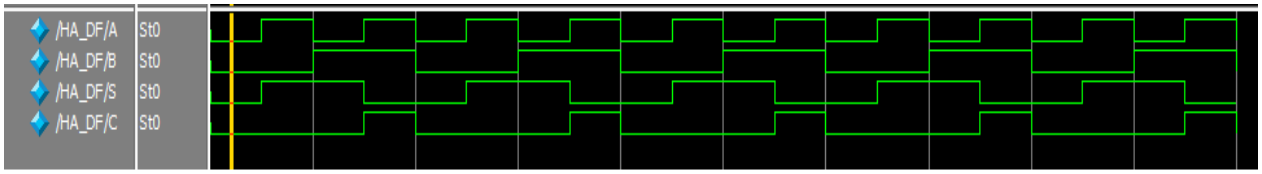
Carry = AB

➢ HALF ADDER IN DATAFLOW:

- **CODE:**

```
module HA_DF(input A,B, output S,C);
assign S=A^B;
assign C=A&B;
endmodule
```

- **GRAPH**:



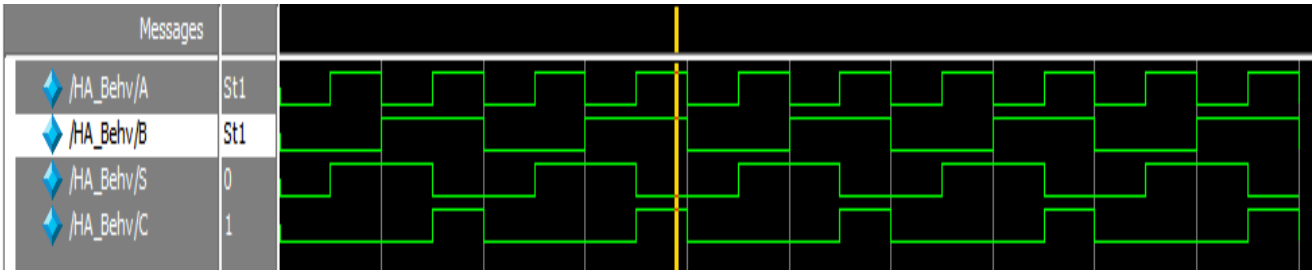
➢ HALF ADDER IN BEHAVIOURAL:

- **CODE:**

```
module HA_Behv(input a, b, output reg s, c); always @(a,b)
if(a==0&&b==0||a==1&&b==1)
s=0;
else s=1
end
```

```
always @(a,b) begin
if(a==1&&b==1)
C=1;
else c=0;
end
endmodule
```

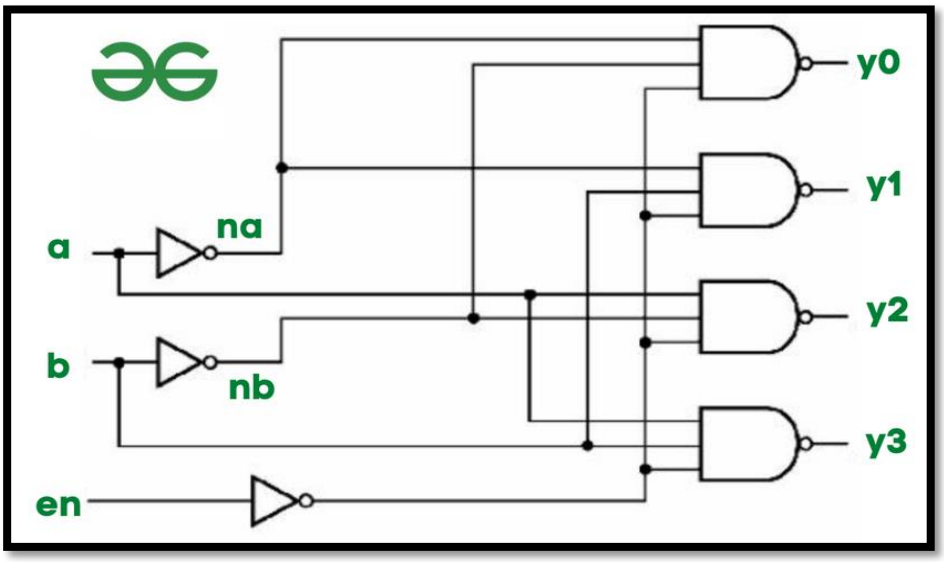
- **GRAPH:**



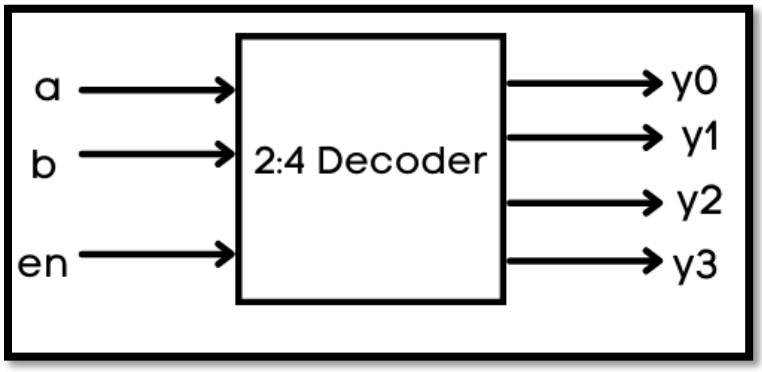
❖ 2 TO 4 DECODER:

The combinational circuit that change the binary information into 2N output lines is known as Decoders. The binary information is passed in the form of N input lines. The output lines define the 2N-bit code for the binary information. In simple words, the Decoder performs the reverse operation of the Encoder. At a time, only one input line is activated for simplicity. The produced 2N-bit output code is equivalent to the binary information. In the 2 to 4 line decoder, there is a total of three inputs, i.e., A0, and A1 and E and four outputs, i.e., Y0, Y1, Y2, and Y3. For each combination of inputs, when the enable 'E' is set to 1, one of these four outputs will be 1.

CIRCUIT DIAGRAM:



BLOCK DIAGRAM:



TRUTH TABLE:

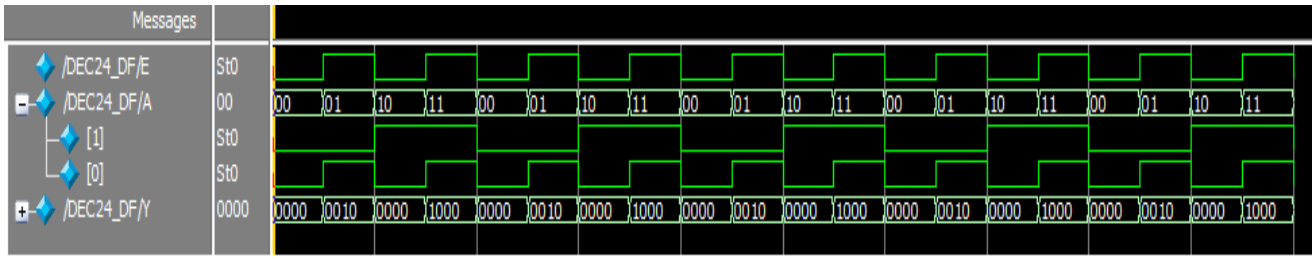
Enable	Inputs		Outputs			
e	a	b	Y3	Y2	Y1	Y0
1	x	x	1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

➢ 2 TO 4 DECODER IN DATAFLOW:

- **CODE:**

```
module DEC 24_DF (input E,[1:0]A, output [3:0]Y);
assign Y[3]=E & A[1] & A[0];
assign Y[2]=E & A[1] & (~A[0]);
assign Y[1]=E & A(~[1]) & A[0];
assign Y[0]=E & A(~[1]) & (~A[0]);
endmodule;
```

- **GRAPH**:

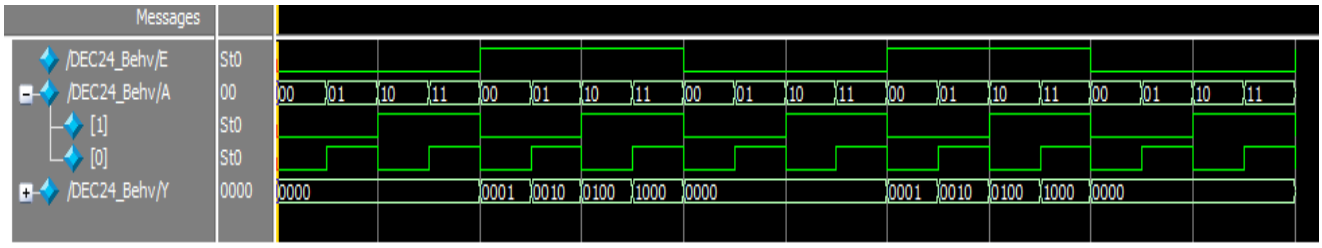


➢ 2 TO 4 DECODER IN BEHAVIOURAL:

- **CODE:**

```
module DEC 24_Behv(input E,[1:0]A, output reg [3:0]Y);
always@ (E,A)
begin
if(E==1 && A==0)
Y=4'b0001;
else if(E==1&& A==1)
Y=4'b0010;
else if(E==1&&A==2)
Y=4'b0100;
else if(E==1&&A==3)
Y=4'b1000;
else Y=4'b0000;
end
endmodule
```

- **GRAPH:**

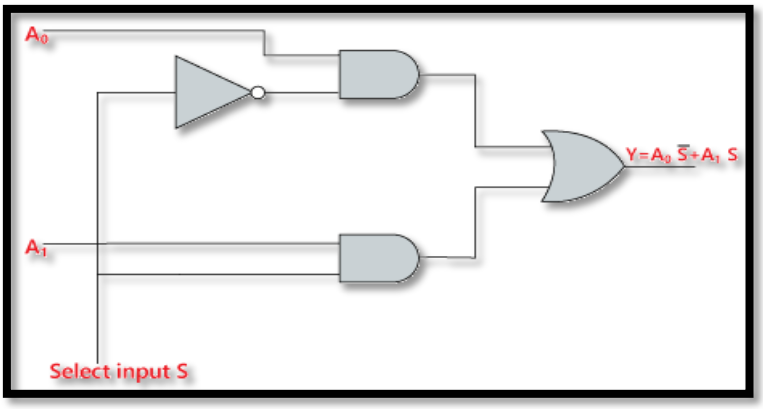


❖ 2 TO 1 MUX

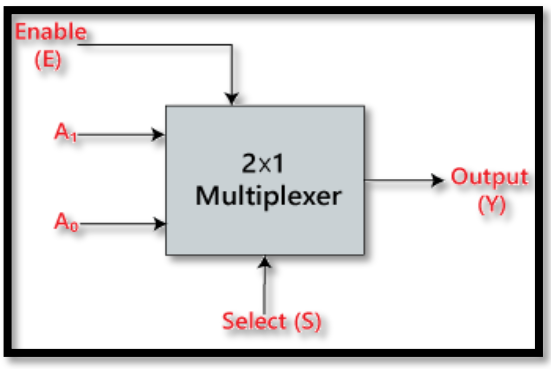
A multiplexer is a combinational circuit that has 2n input lines and a single output line. Simply, the multiplexer is a multi-input and single-output combinational circuit. The binary information is received from the input lines and directed to the output line. The multiplexer or MUX is a digital switch, also called as data selector. It is a Combinational Logic Circuit with more than one input line, one output line and more than one select line.

Y=S0'.A0+S0.A1

CIRCUIT DIAGRAM:



BLOCK DIAGRAM:



TRUTH TABLE:

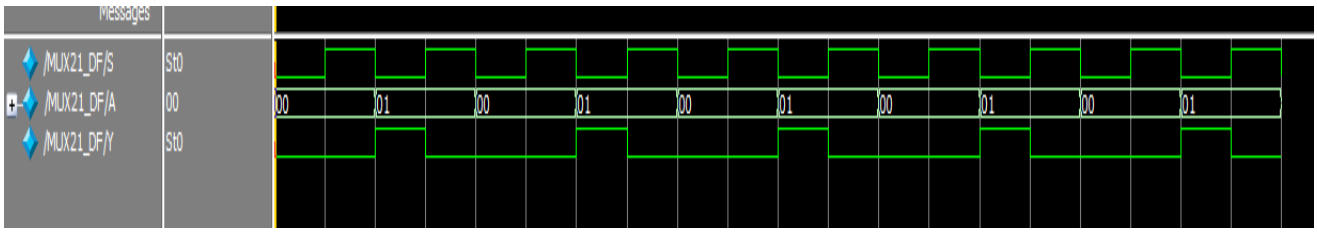
INPUTS	Output
S_0	Y
0	A_0
1	A_1

➢ 2 TO 1 MUX IN DATAFLOW:

- **CODE:**

```
module MUX 21_DF(input S,[1:0]A, output Y);
assign Y=(S & A[1]) | (~S & A[0]);
endmodule
else if (E==1 && A==3)
Y=4'b1000;
else Y=4'b000;
end
endmodule
```

- **GRAPH:**



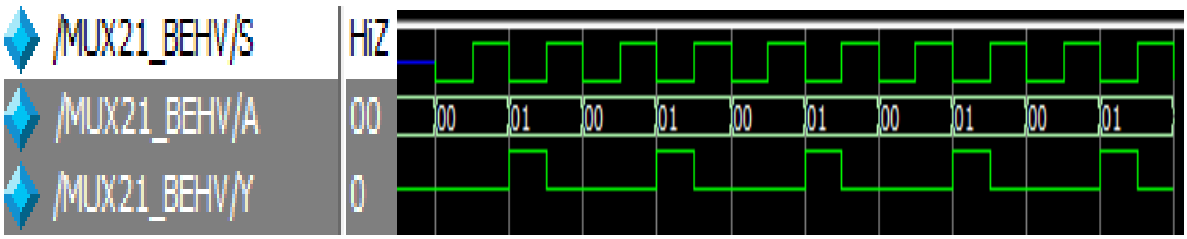
➢ 2 TO 1 MUX IN BEHAVIOURAL:

- **CODE:**

```
module MUX 21_Behv(input S,[1:0]A, output reg Y);
always@ (S,A)
begin
if(S==0)
Y=A[0];
else Y=A[1];
```

end
endmodule

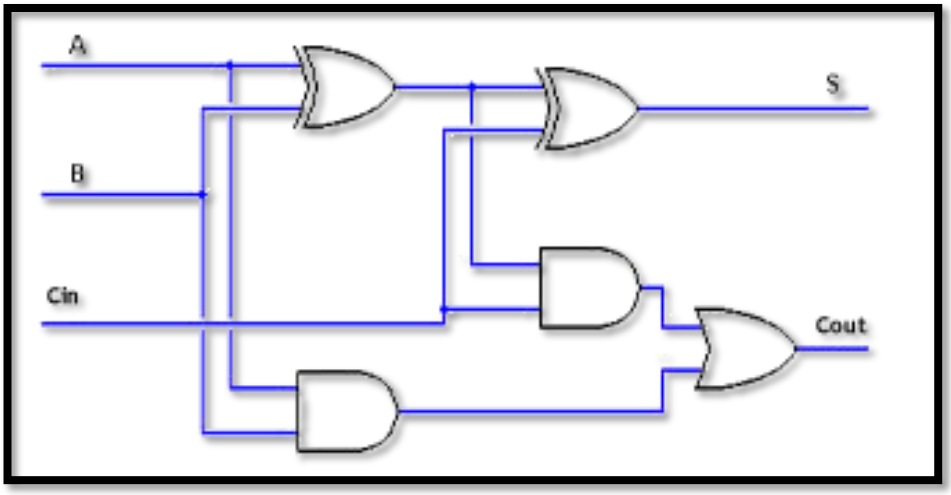
- **GRAPH:**



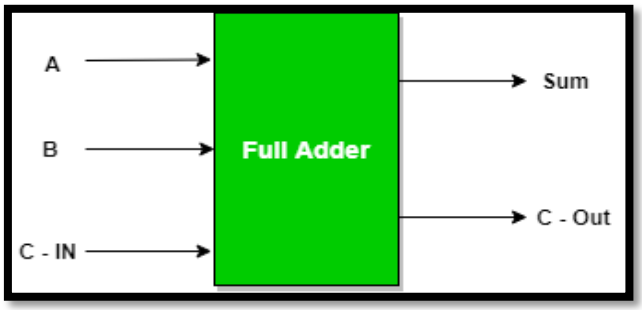
❖ 1-BIT FULL ADDER

A full adder is a digital circuit that performs addition. Full adders are implemented with logic gates in hardware. A full adder adds three one-bit binary numbers, two operands and a carry bit. The adder outputs two numbers, a sum and a carry bit. The term is contrasted with a half adder, which adds two binary digits.

CIRCUIT DIAGRAM:



BLOCK DIAGRAM:



TRUTH TABLE:

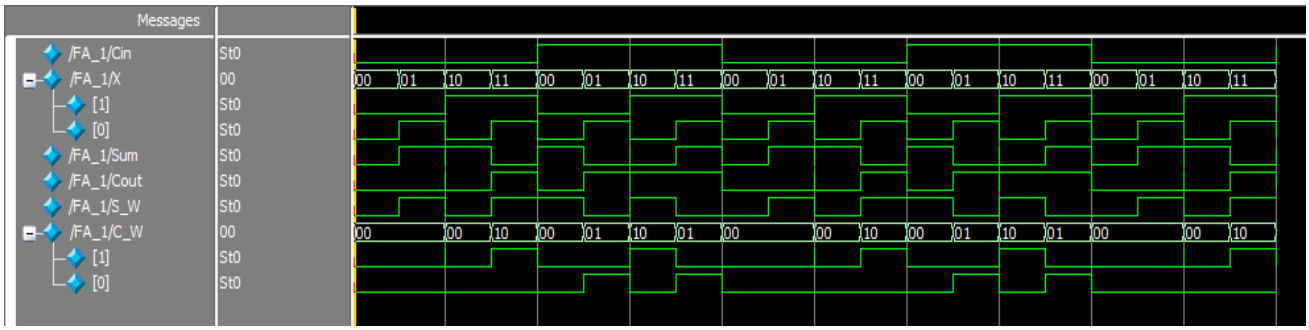
Inputs			Outputs	
A	B	C – IN	Sum	C - Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

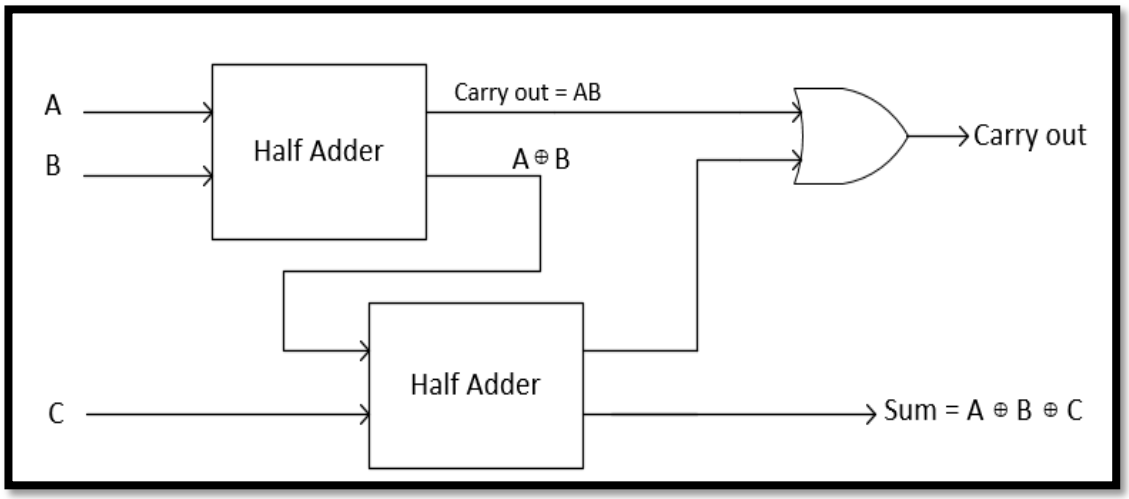
❖ 1-BIT FULL ADDER USING HALF ADDER:

- **CODE:**

```
module FA_1(input Cin,[1:0]X, output sum, Cout);
wire S_W;
wire [1:0]C_W;
HF_DF U0(.A(Cin), .B(X[0]), .C(C_W[0]), .S(S_W));
HF_DF U1(.A(S_W), .B(X[1]), .C(C_W[1]), .S(sum));
assign Cout = C_W[1] | C_W[0];
endmodule
```

- **GRAPH:**

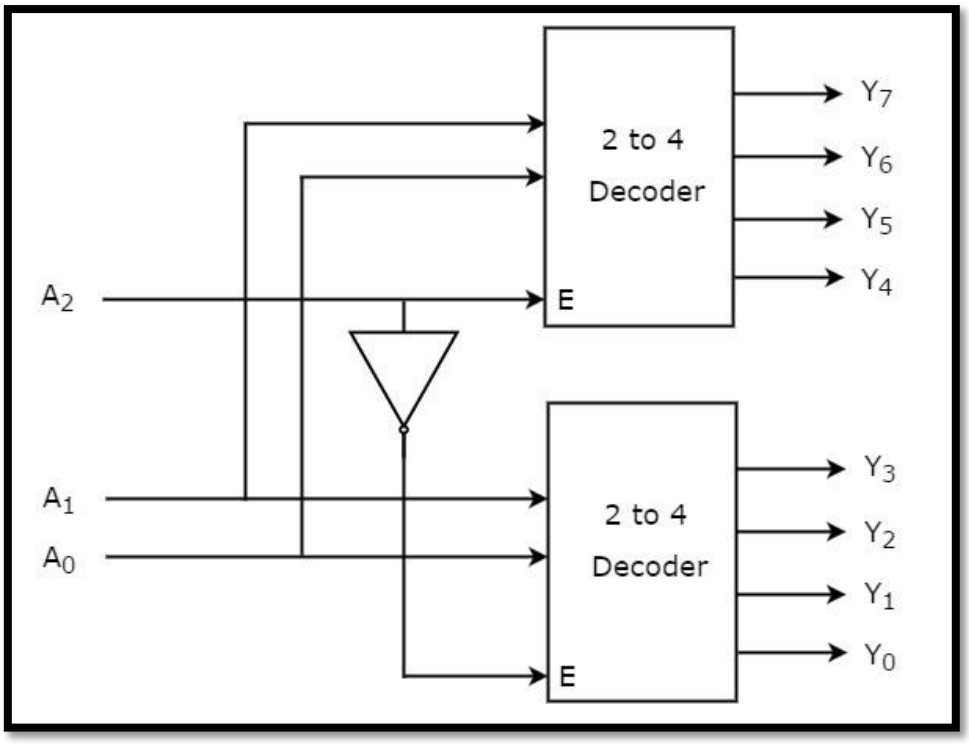




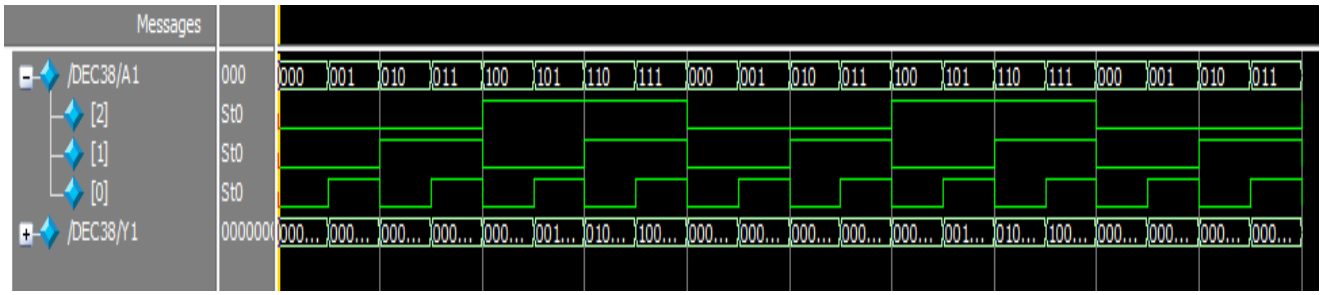
❖3 TO 8 DECODER USING 2 TO 4 DECODER:

- **CODE:**

```
module DEC 38(input [2:0] A1, output [7:0]Y1);
DEC 24_DF U0(.E(A1[2], .A(A1[1:0]), .Y(Y1[7:4]));
DEC 24_DF U1(.E(~A1[2]), .A(A1[1:0]), .Y(Y1[3:0]));
endmodule
```



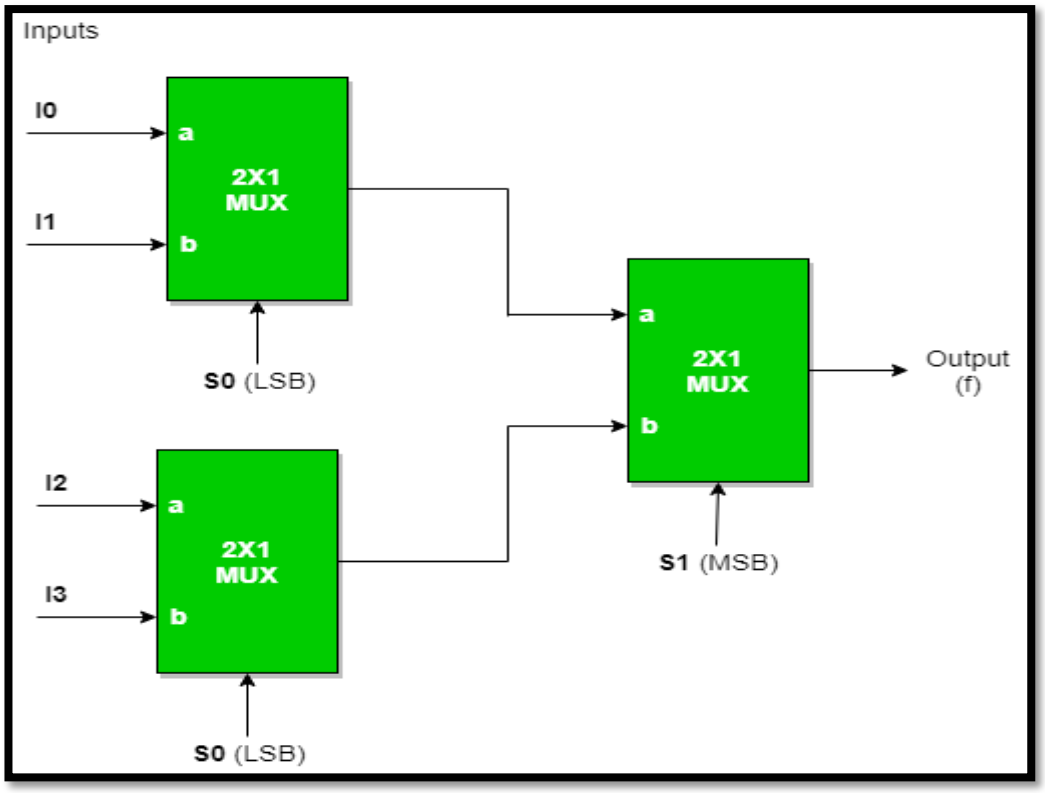
- **GRAPH:**



❖ 4 TO 1 MUX USING 2 TO 1 MUX:

- **CODE:**

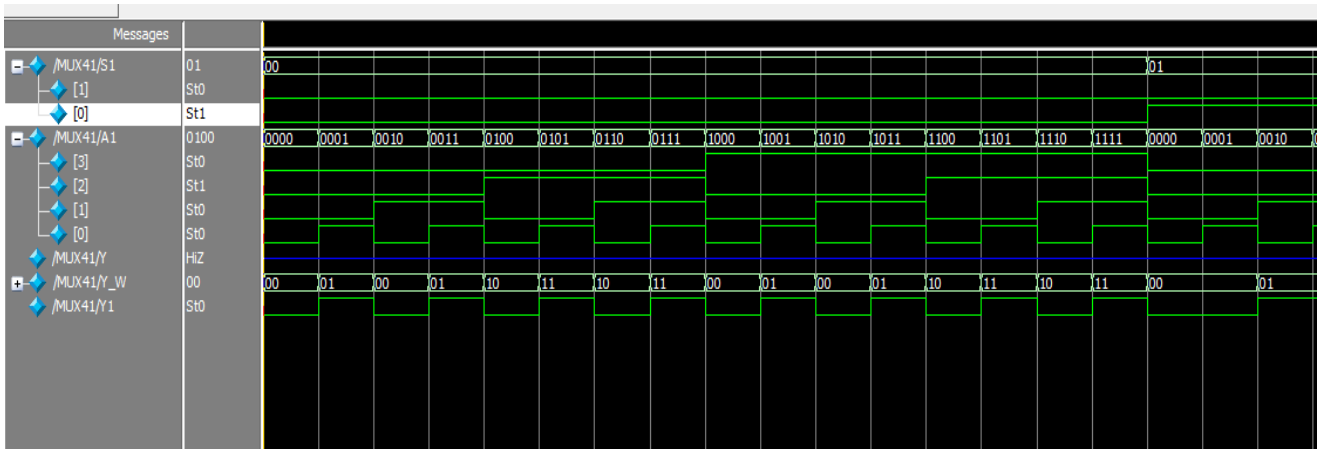
```
module MUX 41(input [1:0] S1,[3:0] A1, output Y1);
wire [1:0] Y_W;
MUX21_DF U0(.S(S1[0], .A(A1[3:2]), .Y(Y_W[1]));
U1(.S(S1[0], .A(A1[1:0]), .Y(Y_W[0]));
U2(.S(S1[1], .A(Y_W), .Y(Y1));
endmodule
```



Truth Table

S0	S1	f
0	0	I0
0	1	I1
1	0	I2
1	1	I3

- **GRAPH:**



FPGA

FPGA

A **Field Programmable Gate Array**, or FPGA, is a type of integrated circuit (IC) that enables the development of custom logic for rapid prototyping and final system design. FPGAs are different than other custom or semi-custom ICs due to their inherent flexibility that allows it to be programmed and reprogrammed via software download to adapt to the evolving needs of the larger system in which it is designed into. FPGAs are ideally suited for today's fastest growing applications, like edge computing, artificial intelligence (AI), system security, 5G, factory automation, and robotics.
Examples include serial transceiver circuitry, Gigabit Ethernet MAC and IOR dual registers to support DDR memory interfaces.

PERFORMANCE:

More recently, FPGAs such as the Xilinx Virtex-7 or the Altera Stratix 5 provides significantly reduced power usage, increased speed, lower materials cost, minimal implementation real estate, and increased possibilities for re-
configuration. A design that included 6 to 10 ASICS can now be achieved using only one FPGA.

ADVANTAGES OF FPGA:

It includes the ability to re-program when already deployed
to fix bugs, and often include shorter time to market and lower non-recurring engineering costs. FPGAs flexibility makes malicious modifications during fabrication a lower risk.

DEVICE WE HAVE USED IN OUR LAB:

Xilinx Vivado, S/W
Family = Artix 7
Device = XC7A100TCSG324-1
Package CSG324
Speed = -1

❖ 4 BIT UP COUNTER:

In digital logic and computing, a Counter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal. Counters are used in digital electronics for counting purpose, they can count specific event happening in the circuit. For example, in UP counter a counter increases count for every rising edge of clock. Not only counting, a counter can follow the certain sequence based on our design like any random sequence 0,1,3,2... They can also be designed with the help of flip-flops. It is a group of flip-flops with a
clock signal applied.

- **CODE:**

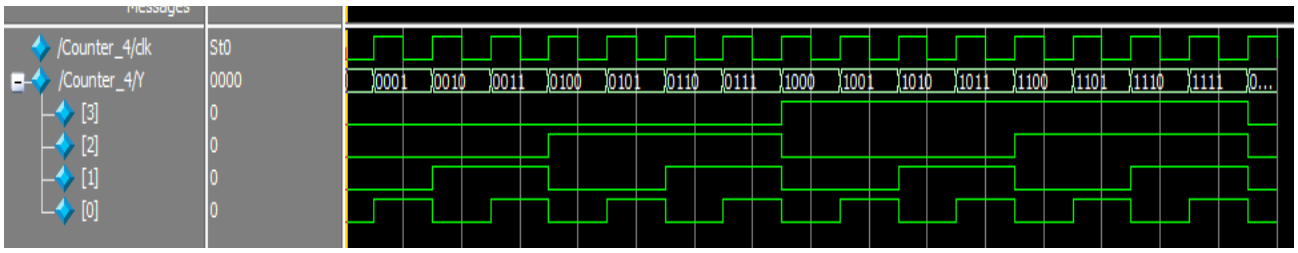
```
module counter(
input clk,
output reg[3:0] y=4'b0000);
reg clk1=0;
integer c=0;
always @(posedge clk)
begin c=c+1;
if(c=25000000)
begin
c=0;
clk1=~clk1;
end
end
always@(posedge clk1)
begin
y=y+1'b1;
end
endmodule
```

- **PIN CONFIGURATIONS:**

Clk-E3
Y[0]=H17
Y[1]=K15

Y[2]=J13
Y[3]=N14

- **GRAPH:**



❖ 4 BIT LEFT SHIFT RESISTER:

A group of flip flops which is used to store multiple bits of data and the data is moved from one flip flop to another is known as Shift Register. The bits stored in registers shifted when the clock pulse is applied within and inside or outside the registers. To form an n-bit shift register, we have to connect n number of flip flops. So, the number of bits of the binary number is directly proportional to the number of flip flops. The flip flops are connected in such a way that the first flip flop's output becomes the input of the other flip flop. A Shift Register can shift the bits either to the left or to the right.

- **CODE:**

```
module left_shift(
input clk,
output reg [3:0] y =4' b0001);
reg clk1 = 0;
integer c =0: always@(posedge clk)
begin
c = c + 1 ;
if(c = 25000000)
begin
c = 0
clk1 = (~clk1);
end
end
```

```
always@(posedge clk1)
begin
y = {y[2/0], y[3]}
end
endmodule
```

- **PIN CONFIGURATION:**

Clk-E3
Y[0]=H17
Y[1]=K15
Y[2]=J13
Y[3]=N14

- **GRAPH:**

❖ 4 BIT COMBINATIONAL CIRCUITS:

- **CODE:**

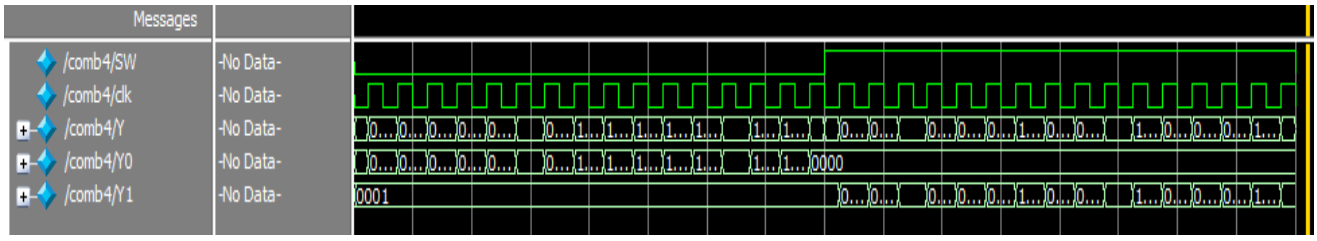
```
module comb4(
input sw, input clk,
output reg [3:0] y);
reg clk1 = 0;
integer c=0;
reg [3:0]y0=4'b0000;
reg [3:0]y1=4'b0001;
always@(posedge clk) begin
c=c+1;
if(c= 25000000)
begin
```

```
c=0;
clk1= clk1;
end
end
always@(posedge clk1)
begin
if(sw==0)
y0=y0+1'b1;
else y1={y1[2:0],y[3]};
end
always@(sw,y0,y1)
begin
if(sw==0)
y=y0;
else y=y1;
end
endmodule
```

- **PIN CONFIGURATION:**

SW=J15
Clk-E3
Y[0]=H17
Y[1]=K15
Y[2]=J13
Y[3]=N14

- **GRAPH:**



❖ 7 SEGMENT DISPLAY:

Seven anodes of the seven segments in a single LED are connected together to one common anode node while its cathodes are separate as shown in the following figure. DP-segment is to illuminate the dot so we omit the DP- segment for now since it is not contributing to the number value of the seven-segment display. To illuminate LED segments such as A-G, anodes of these segments need to be at the 'high' logic level and cathodes should be at the 'low' logic level. Thus, the common anode node should be high to activate a single seven-segment LED display.

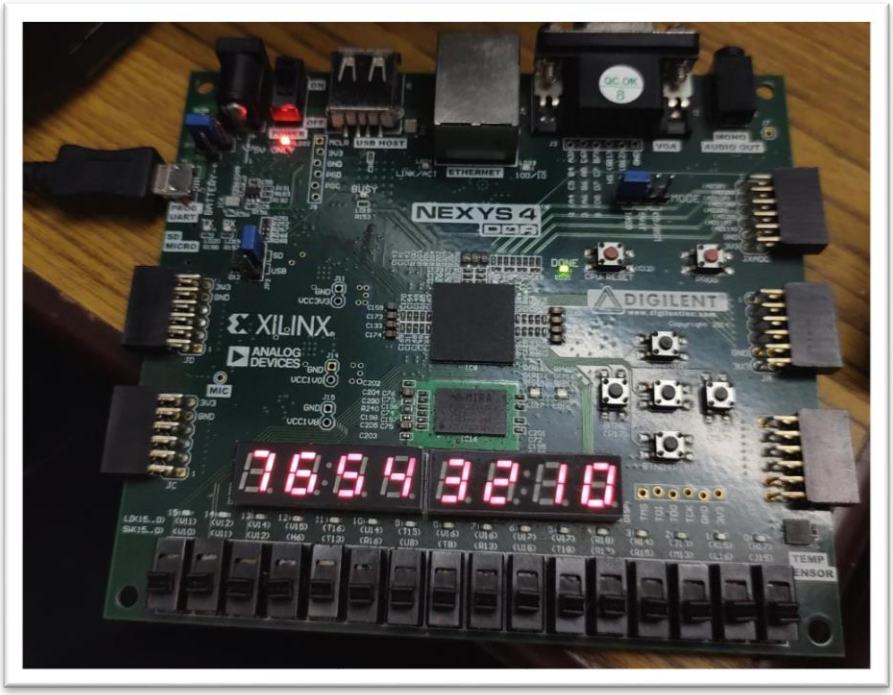
- **CODE:**

```
module seg7(
input clk, output reg [7:0]d=8'hFE, reg [7:0]seg );
reg clk1 =0;
integer c = 0;
reg[2/0] * n = 0 always@(posedge clk)
begin
c = c + 1 ;
f(c = 100000)
begin
c = 0
clk1=~clk1;
end
end
always@(posedge clk1)
begin
n =n+1'b1 ;
d = \{d[6/0], d[7]\} ;
end
always@(n)
begin
case(n)
0:seg=8'h03;
1:seg=8h9F;
2:seg=8'h25;
3:seg-8'h0D;
```

```
4:seg-8'h99;
5:seg-8'h49;
6:seg-8'h41;
7:seg=8'h1F;
default:seg=8'hFF;
endcase
end
endmodule
```

- **PIN CONFIGURATION:**

Seg(0)=H15
Seg(1)=L18
Seg(2)=T11
Seg(3)=P15
Seg(4)=K13
Seg(5)=K16
Seg(6)=R10
Seg(7)=T10
D(0)=J17
D(1) =J18
D(2)=T9
D(3)=J14
D(4) =P14
D(5)=T14
D(6)=K2
D(7) =V13
D(3)=J14
D(4)=P14
D(5)=T14
D(6)=K2
D(7)=U13



❖ LCD DISPLAY:

Display which is used to display the character. The character is represented as the ASCII value (American Standard Code for Information Interchange). To display the character only the ASCII values are sent to LCD. The ASCII is 8bit. Here we have used 16x2 LCD with 5x8 pixel matrix (per character). The definition of 16×2 is the LCD contains 2 rows and 16 characters can be displayed per line.

- **CODE:**

```
module lcd(
input clk, output rw, reg rs, e,reg [7:0] db);
reg clk1=0;
integer c = 0;
integer n=0;
always@(posedge clk)
begin
c = c + 1;
if(c = =7500000)
begin
```

```
c =0;
clk1=~clk1;
end
end
always@(posedge clk1)
begin
n = n + 1
if(n>9)
n = n;
end
always@(n)
begin
if(n<7)
rs=0;
else rs=1;
if(n<=9)
E=clk1;
else E=0;
end
always @(n)
begin
case(n)
0: db =8' h30;
1: db =8' h30:
2: db =8' h30:
3: db =8' h38;
4: db = 8'h0C;
5: dh =8' h06;
6: db =8' h0 ;
7: db =8' h45;
8: db =8' h44;
9: db =8' h41;
default: db =8' h00
endcase
end
assign rw=0;
endmodule
```

- **PIN CONFIGURATION:**

CLK-E3
RS=E16
RW=F13
E=G13
DB(0)=C17
DB(1)=D18
DB(2)=E18
DB(3)=G17
DB(4)=D17
DB(5)=E17
DB(6)=F18
DB(7)=G18

CONCLUSION

The design of VLSI electronic circuits can be achieved at many different abstraction levels, wherein each level represents a different model of the same information and processes, but with varying amounts of detail, starting from system behaviour to the most detailed, physical layout level.

The Electric VLSI Design System is an open-source Electronic Design Automation system that can handle many forms of circuit design, including Custom IC layout, digital and analog Schematic Capture, Textual Languages such as VHDL and Verilog, etc. Electric also provides an excellent way of integrating different environments of design.