VLSI System Design

ELE301P

LAB - 4 - Report

Praveen B R

COE19B007

Submission Date: 15/09/2021

INDEX

- Q1) 4x1 MUX
  - i) Section 1
    - Main Code
    - Testbench
    - Output
    - Waveform
  - ii) Section 2
    - Main Code
    - Testbench
    - Output
    - Waveform
- Q2) Decoder
  - Main Code
  - Testbench
  - Output
  - Waveform
- Q3) Encoder
  - Main Code
  - Testbench
  - Output
  - Waveform
- Other Questions
  - Question 1
  - Question 2

Q1) MUX

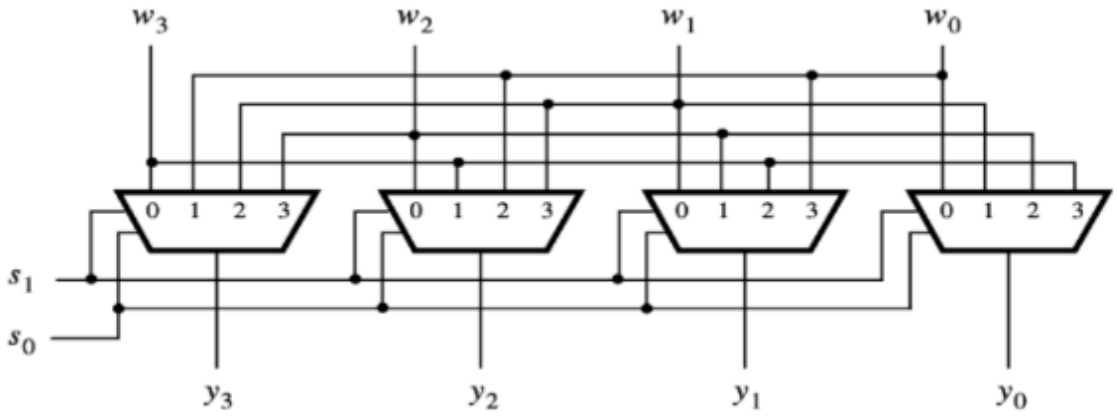
i)4 bit barrel shifter

Objective:

To implement a 4 bit barrel shifter according to give truth table,

s_1	s_0	y_3	y_2	y_1	y_0
0	0	w_3	w_2	w_1	w_0
0	1	w_0	w_3	w_2	w_1
1	0	w_1	w_0	w_3	w_2
1	1	w_2	w_1	w_0	w_3

Theory:



A barrel shifter is **a digital circuit that can shift a data word by a specified number of bits** without the use of any sequential logic, only pure combinational logic, i.e. it inherently provides a binary operation.

Input: One 4 bit number ([3:0] w)
One 2 bit selection variable ([1:0] x)

Output: One 4 bit number ([3:0] y])

Main Code:

```
//structural
module mx41_bs(w0, w1, w2, w3, sel, out );
output [3:0]out;
input w0, w1, w2, w3, s1, s2;
input [1:0] sel;
wire s0bar, s1bar, T1, T2, T3, T4;
wire [3:0]E1;
wire [3:0]E2;
wire [3:0]E3;
wire [3:0]E4;

assign E0 = {w3, w2, w1, w0};
assign E1 = {w0, w3, w2, w1};
assign E2 = {w1, w0, w3, w2};
assign E3 = {w2, w1, w0, w3};

assign out = (sel[0] == sel[1]) ? ((sel[0] == 0) ? E0 : E3) : ((sel[0] ==
0) ? E2 : E1) ;
/*
not u1(s1bar, sel[1]);
not u2(s0bar, sel[0]);
and u3(T1, E0, s0bar, s1bar);
and u4(T2, E1, s0, s1bar);
and u5(T3, E2, s0bar, s1);
and u6(T4, E3, s0, s1);
or u7(out, T1, T2, T3, T4);
*/
endmodule
```

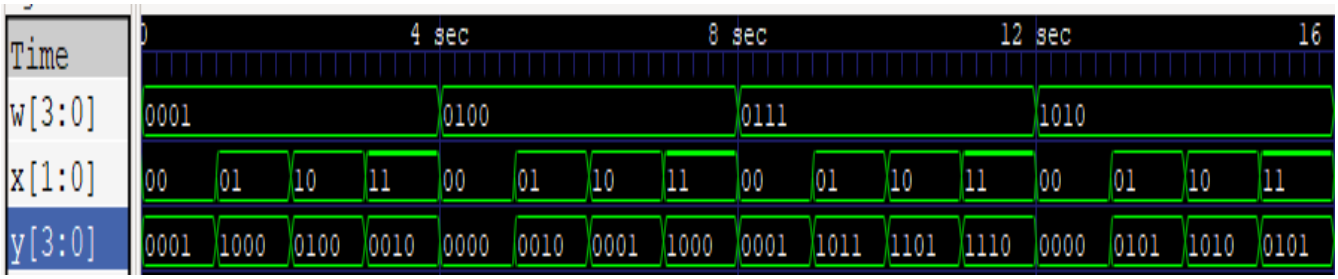
Testbench:

```
module bs_tb;
  // Initialization
  reg [3:0] w;
  reg [1:0] x;
  wire [3:0] y;

  // Instantiate the Unit Under Test (UUT)
  //mux41_bs tb1(w[0],w[1],w[2],w[3],x,y);
  mx41_bs tb2(w[0],w[1],w[2],w[3],x,y);

  initial begin
    // Initialize Inputs
    #0 w=4'b0001;x=2'b00;
    #19
    $finish;
  end

  always begin
    #4 w+=3; //$monitor("\n");
  end

  always begin
    #1 x+=1;
  end

  initial begin
    $dumpfile("q1.vcd");
    $dumpvars(0, bs_tb);
    $monitor("w = %b \tx = %b\t\ty = %b ", w, x, y);
  end

endmodule
```

Output Terminal:

```
PS E:\Sem 5\VLSI\Lab\lab4> iverilog q1.v
PS E:\Sem 5\VLSI\Lab\lab4> vvp a.out
VCD info: dumpfile q1.vcd opened for output.
w = 0001        x = 00          y = 0001
w = 0001        x = 01          y = 1000
w = 0001        x = 10          y = 0100
w = 0001        x = 11          y = 0010
w = 0100        x = 00          y = 0000
w = 0100        x = 01          y = 0010
w = 0100        x = 10          y = 0001
w = 0100        x = 11          y = 1000
w = 0111        x = 00          y = 0001
w = 0111        x = 01          y = 1011
w = 0111        x = 10          y = 1101
w = 0111        x = 11          y = 1110
w = 1010        x = 00          y = 0000
w = 1010        x = 01          y = 0101
w = 1010        x = 10          y = 1010
w = 1010        x = 11          y = 0101
w = 1101        x = 00          y = 0001
w = 1101        x = 01          y = 1110
w = 1101        x = 10          y = 0111
q1.v:57: $finish called at 19 (1s)
w = 1101        x = 11          y = 1011
PS E:\Sem 5\VLSI\Lab\lab4>
```

Waveform:



Conclusion:

Thus, a barrel shift has been implemented successfully using 4x1 MUX and simulation waveforms of more than 10 cases have been shown.

Application:

A barrel shifter is often used to shift and rotate n-bits in modern microprocessors, typically within a single clock cycle.

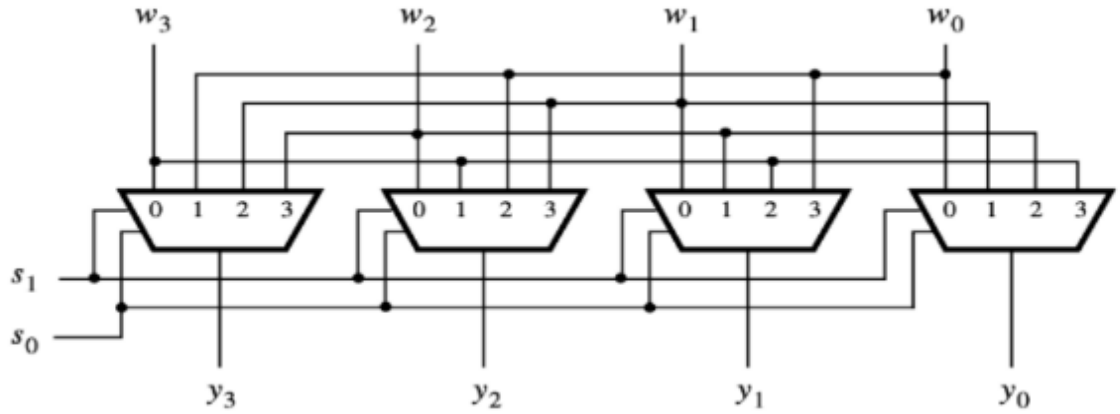
ii)

Objective:

To implement a 4 bit barrel shifter according to give truth table,

S_1	S_0	Operation
0	0	Right shift by 1 bit 1101→(0110)
0	1	Left shift by 1 bit 1101→1010
1	0	Right circular shift 1101→1110
1	1	Left circular shift 1101→1011

Theory:



A barrel shifter is **a digital circuit that can shift a data word by a specified number of bits** without the use of any sequential logic, only pure combinational logic, i.e. it inherently provides a binary operation.

Input: One 4 bit number ([3:0] w)
One 2 bit selection variable ([1:0] x)

Output: One 4 bit number ([3:0] y])

Main Code:

```
module MUX(w1, w2, w3, w4, sel0, sel1, y);
     input w1, w2, w3, w4;
     input sel0, sel1;
     output y;
     wire negsel0, negsel1;
     wire r1, r2, r3, r4;

     assign negsel0 = ~sel0;
     assign negsel1 = ~sel1;

     and a0(r1,sel0,sel1,w4);
     and a1(r2,sel0,negsel1,w3);
     and a2(r3,negsel0,sel1,w2);
     and a3(r4,negsel0,negsel1,w1);
     or o1(y, r1, r2, r3, r4);

endmodule


module que1_2(M, sel0, sel1, y);
    input [3:0] M;
    input sel0, sel1;
    output [3:0] y;
    wire y1, y2, y3, y4;
    reg w1, w2, w3, w4;
    wire [1:0] sel;
    assign sel = {sel0, sel1};

    always @ (*) begin

        case(sel)
            2'b00: begin
                w1 = M[1]; // 1
                w2 = M[2]; // 1
                w3 = M[3]; // 0
                w4 = 0;    // 1
            end
            2'b01: begin
                w1 = M[2]; // 1
```

```
                w2 = 0;     // 1
                w3 = M[0]; // 0
                w4 = M[1]; // 1
            end
            2'b10: begin
                w1 = M[3]; // 1
                w2 = M[0]; // 1
                w3 = M[1]; // 0
                w4 = M[2]; // 1
            end
            2'b11: begin
                w1 = M[0]; // 1
                w2 = M[1]; // 1
                w3 = M[2]; // 0
                w4 = M[3]; // 1
            end
        endcase

    end

    MUX mux1(w1, w2, w3, w4, sel0, sel1, y1); // 0
    MUX mux2(w2, w3, w4, w1, sel0, sel1, y2); // 1
    MUX mux3(w3, w4, w1, w2, sel0, sel1, y3); // 2
    MUX mux4(w4, w1, w2, w3, sel0, sel1, y4); // 3

    assign y[0] = y1;
    assign y[1] = y2;
    assign y[2] = y3;
    assign y[3] = y4;

endmodule
```

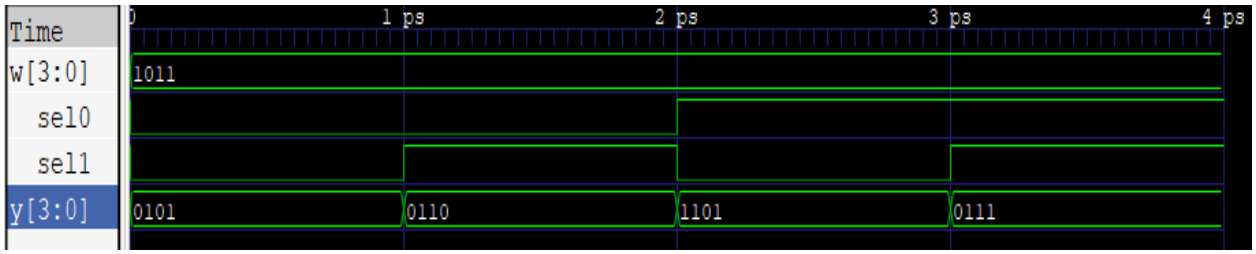
Testbench:

```
module que12_tb;
  reg[3:0] w;
  reg sel0, sel1;
  wire [3:0] y;

    que1_2 op(w, sel0, sel1, y);

    initial begin

    #0  w=4'b1011;sel0=1'b0;sel1=1'b0;
    #1  w=4'b1011;sel0=1'b0;sel1=1'b1;
    #1  w=4'b1011;sel0=1'b1;sel1=1'b0;
    #1  w=4'b1011;sel0=1'b1;sel1=1'b1;
    #1
    $finish;

    end

    initial begin

    $dumpfile("q1.vcd");
    $dumpvars(0,que12_tb);
    $monitor("w=%b \t s=%b%b \ty=%b ",w,sel0,sel1,y);

    end
endmodule
```

Output terminal:

```
PS E:\Sem 5\VLSI\Lab\lab4> iverilog q2.v
PS E:\Sem 5\VLSI\Lab\lab4> vvp a.out
VCD info: dumpfile q1.vcd opened for output.
w=1011   s=00   y=0101
w=1011   s=01   y=0110
w=1011   s=10   y=1101
w=1011   s=11   y=0111
PS E:\Sem 5\VLSI\Lab\lab4>
```

Waveform:



Conclusion:

Thus, a barrel shift has been implemented successfully using 4x1 MUX and simulation waveforms of more than 10 cases have been shown.

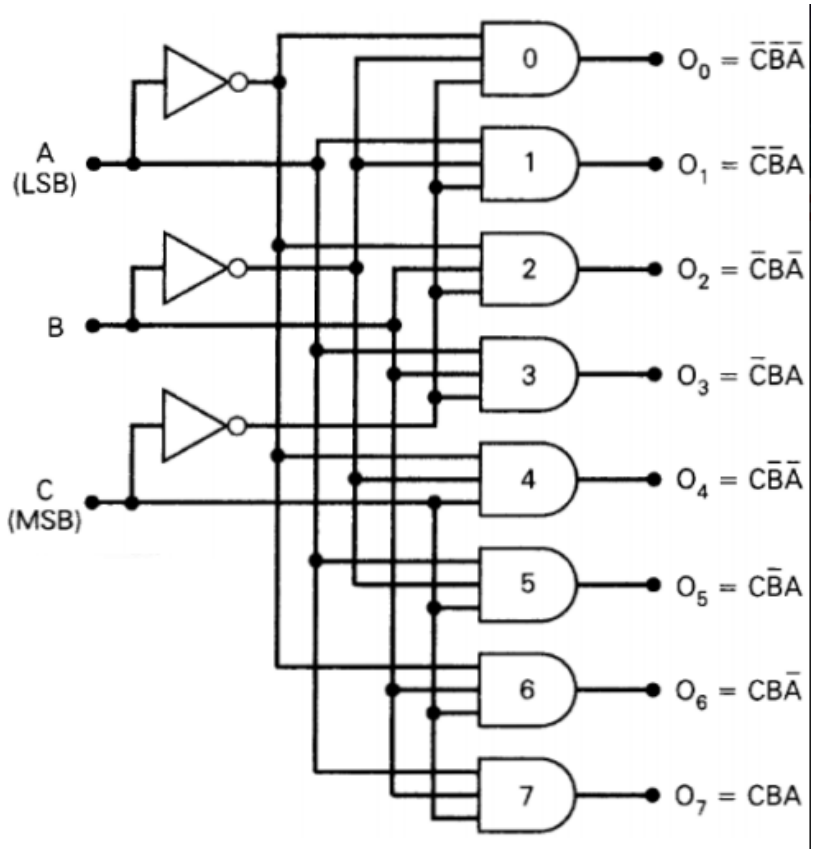
Application:

A barrel shifter is often used to shift and rotate n-bits in modern microprocessors, typically within a single clock cycle.

Q2) 3x8 decoder

Objective:
To implement a 3x8 decoder using verilog

Theory:



A decoder is a combinational logic circuit that is used to change the code into a set of signals. It is the reverse process of an encoder. A decoder circuit takes multiple inputs and gives multiple outputs. A decoder circuit takes binary data of ‘n’ inputs into ‘2^n’ unique output. In addition to input pins, the decoder has a enable pin. This enables the pin when negated, to make the circuit inactive. in this article, we discuss 3 to 8 line Decoder and demultiplexer.

Input : Three 1 bit numbers
Output: 2^3 outputs (8)

Main Code:

```
`timescale 1ps/1ps

module Decoder(a,b,c,d0,d1,d2,d3,d4,d5,d6,d7);
input a,b,c;
output d0,d1,d2,d3,d4,d5,d6,d7;
wire abar, bbar, cbar;
not n1(abar, a);
not n2(bbar, b);
not n3(cbar, c);
and a1(d0, abar, bbar, cbar);
and a3(d1, abar, bbar, c);
and a5(d2, abar, b, cbar);
and a7(d3, abar, b, c);
and a9(d4, a, bbar, cbar);
and a11(d5, a, bbar, c);
and a13(d6, a, b, cbar);
and a15(d7, a, b, c);
endmodule
```

Testbench:

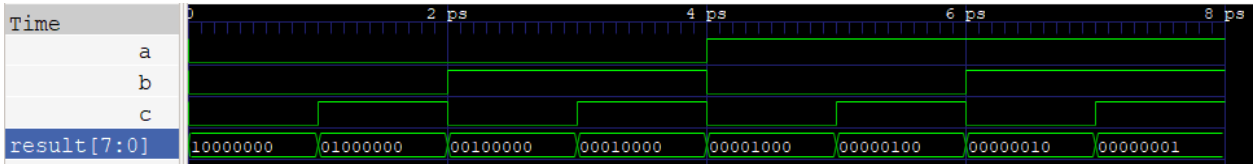
```
module decoder_tb ();

reg a, b, c;
wire d0,d1,d2,d3,d4,d5,d6,d7;
integer i=0;
Decoder dec1(a,b,c,d0,d1,d2,d3,d4,d5,d6,d7);

initial begin
#0;
for(i=0; i<8; i++)
begin
c=i[0];
b=i[1];
a=i[2];
#1;
end
end

initial begin
```

```
$monitor("%0d\t%d%d%d \t %d%d%d%d%d%d%d%d",
$time,a,b,c,d0,d1,d2,d3,d4,d5,d6,d7); //to monitor changes
$dumpfile("decoder.vcd");
$dumpvars(0,decoder_tb);
end

endmodule
```

Output Terminal:

```
PS E:\Sem 5\VLSI\Lab\lab4> iverilog q3.v
PS E:\Sem 5\VLSI\Lab\lab4> vvp a.out
VCD info: dumpfile q3.vcd opened for output.
0       000      10000000
1       001      01000000
2       010      00100000
3       011      00010000
4       100      00001000
5       101      00000100
6       110      00000010
7       111      00000001
PS E:\Sem 5\VLSI\Lab\lab4>
```

Waveform:



Conclusion:

Thus a 3x8 decoder has been implemented successfully using verilog and all test cases have been shown

Application:

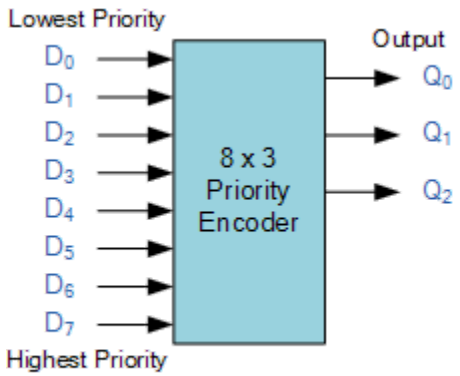
- The Decoders were used in analog to digital conversion in analog decoders.
- Used in electronic circuits to convert instructions into CPU control signals.
- They mainly used in logical circuits, data transfer.

Q3) 8x3 encoder

Objective:

To implement a 8x3 encoder using verilog

Theory:



Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

X = dont care

An Encoder is a combinational circuit that performs the reverse operation of Decoder.It has maximum of 2^n input lines and ‘n’ output lines, hence it encodes the information from 2^n inputs into an n-bit code. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes 2^n input lines with ‘n’ bits.

Main Code:

```
module encoder(in0, in1, in2, in3, in4, in5, in6, in7, res);
    input in0, in1, in2, in3, in4, in5, in6, in7;
    output [2:0] res;

    wire in0bar, in1bar, in2bar;

    assign in0bar = ~in0;
    assign in1bar = ~in1;
    assign in2bar = ~in2;

    or o0(res[2],in0, in1, in2, in3);
    or o1(res[1],in0, in1, in4, in5);
    or o2(res[0],in0, in2, in4, in6);


endmodule
```

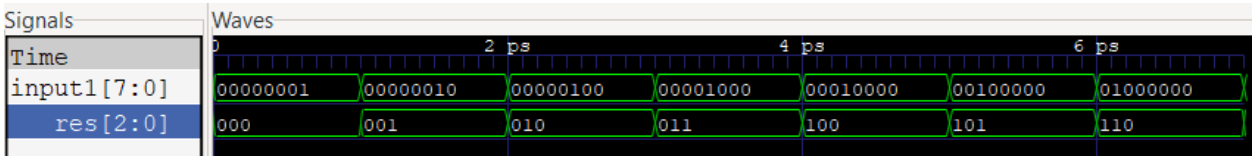
Testbench:

```
module enc_tb;
  reg s0, s1, s2, s3, s4, s5, s6, s7;
  wire [2:0] res;
  wire [7:0]input1;

    encoder en(s0, s1, s2, s3, s4, s5, s6, s7, res);
    assign input1 = {s0,s1,s2,s3,s4,s5,s6,s7};
    initial begin

    #0  s0=1'b0; s1=1'b0; s2=1'b0; s3=1'b0; s4=1'b0; s5=1'b0; s6=1'b0;
s7=1'b1;
    #1  s0=1'b0; s1=1'b0; s2=1'b0; s3=1'b0; s4=1'b0; s5=1'b0; s6=1'b1;
s7=1'b0;
    #1  s0=1'b0; s1=1'b0; s2=1'b0; s3=1'b0; s4=1'b0; s5=1'b1; s6=1'b0;
s7=1'b0;
    #1  s0=1'b0; s1=1'b0; s2=1'b0; s3=1'b0; s4=1'b1; s5=1'b0; s6=1'b0;
s7=1'b0;
    #1  s0=1'b0; s1=1'b0; s2=1'b0; s3=1'b1; s4=1'b0; s5=1'b0; s6=1'b0;
s7=1'b0;
    #1  s0=1'b0; s1=1'b0; s2=1'b1; s3=1'b0; s4=1'b0; s5=1'b0; s6=1'b0;
s7=1'b0;
    #1  s0=1'b0; s1=1'b1; s2=1'b0; s3=1'b0; s4=1'b0; s5=1'b0; s6=1'b0;
s7=1'b0;
    #1  s0=1'b1; s1=1'b0; s2=1'b0; s3=1'b0; s4=1'b0; s5=1'b0; s6=1'b0;
s7=1'b0;

    end

    initial begin

    $dumpfile("q4.vcd");
    $dumpvars(0,enc_tb);
    $monitor("%d\t %b%b%b%b%b%b%b%b\t %b ",$time,s0, s1, s2, s3, s4, s5,
s6, s7, res);

    end
endmodule
```

Output Terminal:

```
PS E:\Sem 5\VLSI\Lab\lab4> iverilog q4.v
PS E:\Sem 5\VLSI\Lab\lab4> vvp a.out
VCD info: dumpfile q4.vcd opened for output.
                   0     00000001        000
                   1     00000010        001
                   2     00000100        010
                   3     00001000        011
                   4     00010000        100
                   5     00100000        101
                   6     01000000        110
                   7     10000000        111
PS E:\Sem 5\VLSI\Lab\lab4>
```

Waveform:



Conclusion:

Thus a 8x3 encoder has been implemented successfully using verilog and all test cases have been shown

Application:

- Encoders are very common electronic circuits used in all digital systems.
- Encoders are used to translate the decimal values to the binary in order to perform the binary functions such as addition, subtraction, multiplication, etc.
- Other applications especially for Priority Encoders may include detecting interrupts in microprocessor applications.

Other Questions

1. Write the Verilog syntax for if-else conditional statements with one example Verilog code.

Syntax:

```
Initial begin
        if(condition) begin
                statement
        end
        else begin
                statement
        end
end
```

Code:

```
module IF();
    wire input1;
    assign input1 = 10;
    initial
    begin
        if(input1%2 == 0)
        begin
            $display("Even\n");
        end
    else
        begin
            $display("Odd\n");
        end
    end
endmodule
```

Output:

```
PS E:\Sem 5\VLSI\Lab\lab4> vvp a.out
Even

PS E:\Sem 5\VLSI\Lab\lab4>
```

2. Explain the Case statement in the Verilog with the help of example Verilog code.

Syntax:
initial begin
case(**expression**) begin
case1: begin
statement
end
case2: begin
statement
end
default:
statement
endcase
end

Code:

```
module CASE();
    wire input1;
    assign input1 = 10;
    initial
    begin
        case(input1%2)
            1'b0:
                $display("Even");
            1'b1:
                $display("Odd");
            default:
                $display("default case");
        endcase
    end
endmodule
```

Output:

```
PS E:\Sem 5\VLSI\Lab\lab4> vvp a.out
Even
PS E:\Sem 5\VLSI\Lab\lab4>
```