EX.NO: 1 DESIGN ENTRY AND SIMULATION OF COMBINATIONAL LOGIC CIRCUITS

DATE: 24/01/2022

AIM:

To verify the functionality and timing of your design or portion of your design. To interpret Verilog code into circuit functionality and displays logical results of the described HDL to determine correct circuit operation and to create and verify complex functions in a relatively small amount of time.

TOOLS REQUIRED:

1. Xilinx Vivado Design Suite: WebEdition

PROCEDURE:

1. Start by clicking **Vivado Design Suite: WebEdition**

2. Go to **File → new project → Enter project name**, select the top level source as **HDL** & click **next**.

3. Enter **device properties** as

   Product Category : General Purpose
   Family : Kintex 7
   Device : xc7k70t
   Package :fbg484
   Speed : -1
   Top Level Source Type : HDL
   Synthesis Tool : XST (VHDL/Verilog)
   Simulator : ISim (VHDL/Verilog)
   Preferred Language : Verilog

   & **Click next**.

4. **Right click** the device name (XCS3540) in the source window

   to create **new source**.

5. Select **Verilog module** and enter **file name** in the new source window & click **next**.

6. Write the **Verilog code** in the **Verilog editor window**.

7. Write the **testbench** by create **new source simulation source.**

8. Run Check syntax through **Process window→ synthesize→** double click **Behavioral Check Syntax→** and removes error if present, with proper syntax & coding.

9. Click on the symbol of FPGA device and then right click→ click on **new source**.

10. Select the desired parameters for simulating the design. In this case **combinational circuit** and **simulation time** click **finish**.

11. Assign **all input signal** (**high or low**) using just **click** on this and save file.

12. From the **source process window**. Click **Behavioral simulation**

from drop-down menu.

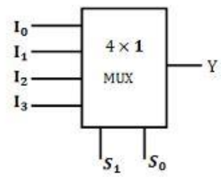
13. Double click the **Simulation Behavioral Model**.

14. Verify your design in **wave window** by seeing behavior of output signal with respect to input signal.

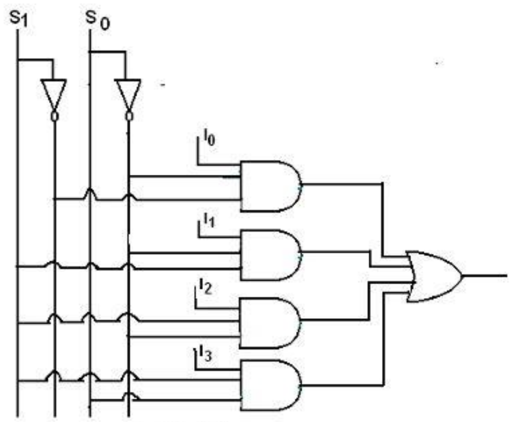
4x1 Mux:

MULTIPLEXER (4x1):

Logic Symbol



Logic Circuit



4 x 1 Multiplexer

Truth Table

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Expression

$$\text{Output} = I_0 S_1' S_0' + I_1 S_1' S_0 + I_2 S_1 S_0' + I_3 S_1 S_0$$

PROGRAM

Gate Level

```
//1960628 PREM
//4x1 Mux gate level
module mux_4x1(
    input s0,
    input s1,
    input i0,
    input i1,
    input i2,
    input i3,
    output y
    );
wire w0,w1,w2,w3,w4,w5;
not no1(w0,s0);
not no2(w1,s1);
and and1(w2,i0,w0,w1);
and and2(w3,i1,w0,s1);
and and3(w4,i2,w1,s0);
and and4(w5,i3,s1,s0);
or or1(y,w2,w3,w4,w5);
endmodule
```

Data Flow

```
//1960628 PREM
//4x1 Mux Data Flow
module mux_4x1(
    input s0,
    input s1,
    input i0,
    input i1,
    input i2,
    input i3,
    output y
    );
assign not_0 = ~s0;
assign not_1 = ~s1;
assign and1 = not_1&not_0&i0;
assign and2 = s1&not_0&i1;
assign and3 = not_1&s0&i2;
assign and4 = s1&s0&i3;
assign y = and1|and2|and3|and4;

endmodule
```

Behavioural

```
//1960628 Prem
//4x1 mux Behavioural
module mux_4x1(
    input s0,
    input s1,
    input i0,
    input i1,
    input i2,
    input i3,
    output reg y
    );
always @(s0 or s1 or i0 or i1 or i2 or i3)
    begin
            case({s1,s0})
            2'b00: y = i0;
            2'b01: y = i1;
            2'b10: y = i2;
            2'b11: y = i3;
            default: y = 0;
            endcase
    end

endmodule
```

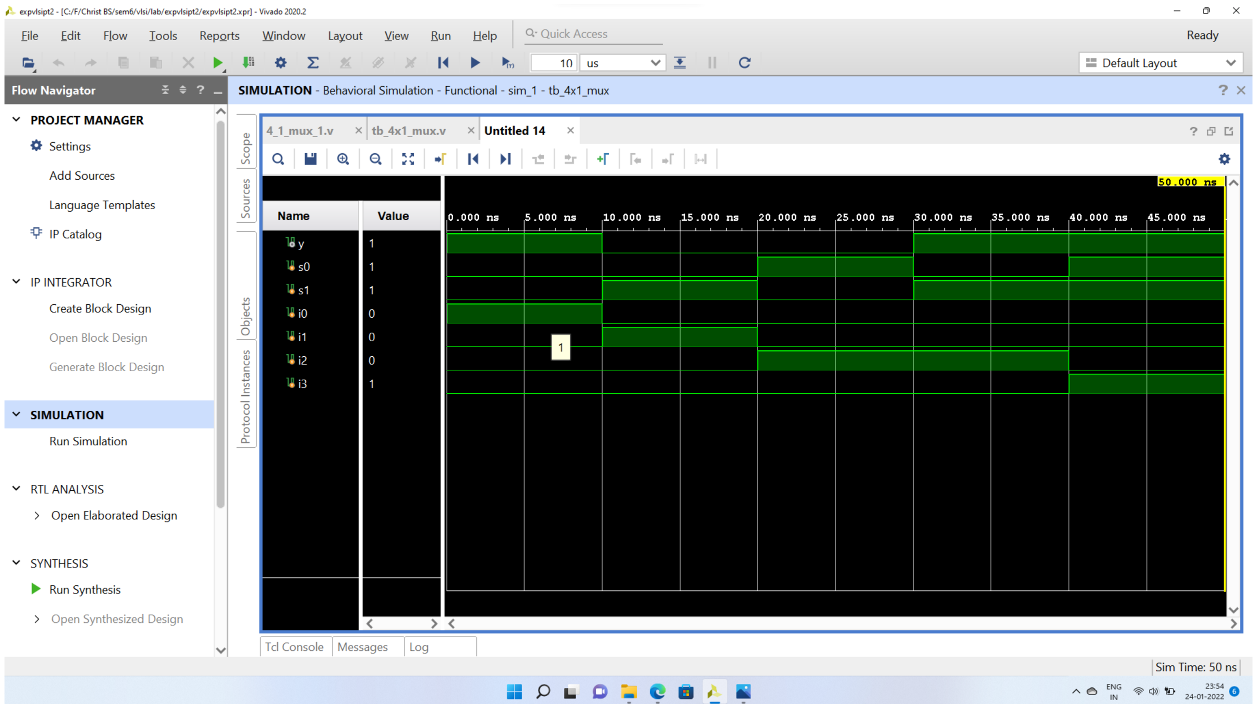
TESTBENCH

```
//1960628 PREM
//4x1 Mux Testbench
module tb_4x1_mux();
wire y;
reg s0,s1,i0,i1,i2,i3;

mux_4x1 I1(s0,s1,i0,i1,i2,i3,y);
initial
begin
s0 = 1'b0;
s1 = 1'b0;
i0 = 1'b1;
i1 = 1'b0;
i2 = 1'b0;
i3 = 1'b0;
#10
s0 = 1'b0;
s1 = 1'b1;
i0 = 1'b0;
i1 = 1'b1;
i2 = 1'b0;
i3 = 1'b0;
#10
```

```
s0 = 1'b1;
s1 = 1'b0;
i0 = 1'b0;
i1 = 1'b0;
i2 = 1'b1;
i3 = 1'b0;
#10
s0 = 1'b0;
s1 = 1'b1;
i0 = 1'b0;
i1 = 1'b0;
i2 = 1'b01;
i3 = 1'b0;
#10
s0 = 1'b1;
s1 = 1'b1;
i0 = 1'b0;
i1 = 1'b0;
i2 = 1'b0;
i3 = 1'b1;
#10
$finish();
end
```

OUTPUT



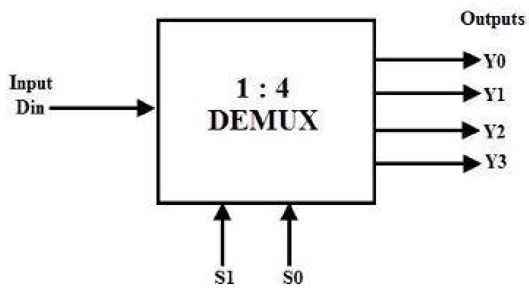
Result:

Thus the Verilog code was scripted, simulated and waveforms were verified using Xilinx Vivado Webpack.

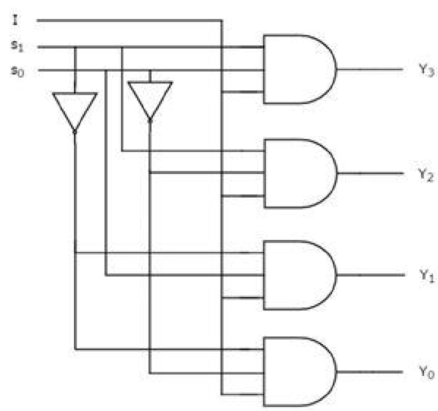
1x4 Demux:

DEMULTIPLEXER (1x4):

Logic Symbol



Logic Circuit



Truth Table

Data Input	Select Inputs		Outputs			
D	S_1	S_0	Y_3	Y_2	Y_1	Y_0
D	0	0	0	0	0	D
D	0	1	0	0	D	0
D	1	0	0	D	0	0
D	1	1	D	0	0	0

Expression

Y0 = S1' S0' D
Y1 = S1' S0 D
Y2 = S1 S0' D
Y3 = S1 S0 D

PROGRAM

Gate Level

```
//1960628 Prem
//Demux 1x4 GATE LEVEL
module demux(
    input din,
    input s0,
    input s1,
    output y0,
    output y1,
    output y2,
    output y3
    );
wire w0,w1;
not no1(w0,s0);
not no2(w1,s1);
and out(y0,w1,w0,din);
and out1(y1,w1,s0,din);
and out2(y2,s1,w0,din);
and out3(y3,s1,s0,din);

endmodule
```

Data Flow

```
//1960628 Prem
//Demux 1x4 Data Flow
module demux(
    input din,
    input s0,
    input s1,
    output y0,
    output y1,
    output y2,
    output y3
    );
wire w0,w1;
assign w0 = ~s0;
assign w1 = ~s1;
assign y0 = w1&w0&din;
assign y1 = w1&s0&din;
assign y2 = s1&w0&din;
assign y3 = s1&s0&din;
endmodule
```

Behavioural

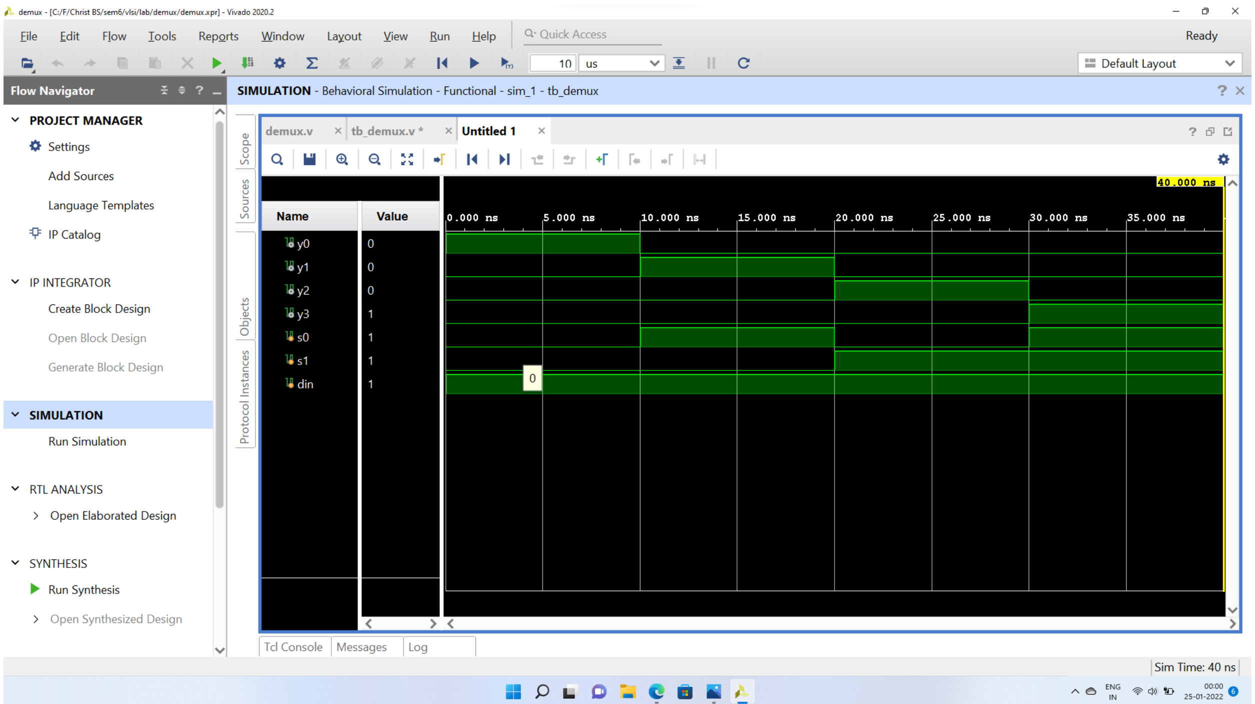
```
//1960628 Prem
//Demux 1x4 Behavioural
module demux(
    input din,
    input s0,
    input s1,
    output reg y0,
    output reg y1,
    output reg y2,
    output reg y3
    );
always @(din or s0 or s1)
    begin
        case({s1,s0})
            2'b00: begin y0 = din;y1 = 0;y2 = 0;y3 = 0; end
            2'b01: begin y0 = 0;y1 = din;y2 = 0;y3 = 0; end
            2'b10: begin y0 = 0;y1 = 0;y2 = din;y3 = 0; end
            2'b11: begin y0 = 0;y1 = 0;y2 = 0;y3 = din; end
            default: begin  y0 = 0;y1 = 0;y2 = 0;y3 = 0; end
        endcase
    end

endmodule
```

TESTBENCH

```
//1960628 Prem
// Demux 1x4 testbench
module tb_demux();
wire y0,y1,y2,y3;
reg s0,s1,din;
demux I1(din,s0,s1,y0,y1,y2,y3);
initial
begin
din = 1'b1;
s0 = 1'b0;
s1 = 1'b0;
#10
din = 1'b1;
s0 = 1'b1;
s1 = 1'b0;
#10
din = 1'b1;
s0 = 1'b0;
s1 = 1'b1;
#10
din = 1'b1;
s0 = 1'b1;
s1 = 1'b1;
#10
$finish();
```

OUTPUT



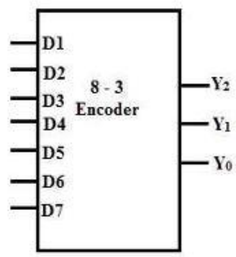
Result:

Thus the Verilog code was scripted, simulated and waveforms were verified using Xilinx Vivado Webpack.

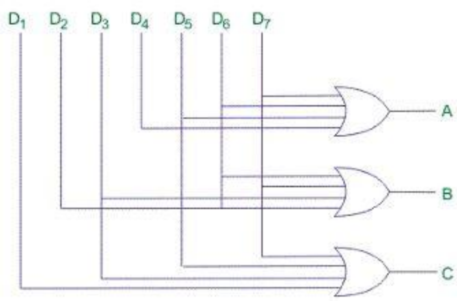
8x3 Encoder:

Encoder:

Logic Symbol



Logic Circuit



Truth Table

Inputs								Outputs		
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	y_2	y_1	y_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Expression

y0 = I1 + I3 + I5 + I7

y1 = I2 + I3 + I6 + I7

y2 = I4 + I5 + I6 + I7

PROGRAM

Gate Level

```
//1960628 Prem
//encoder 8x3 Gate level
module encoder(
    input d0,
    input d1,
    input d2,
    input d3,
    input d4,
    input d5,
    input d6,
    input d7,
    output y0,
    output y1,
    output y2
    );
or out1(y0,d7,d5,d3,d1);
or out3(y2,d7,d6,d5,d4);
or out2(y1,d7,d6,d2,d3);
endmodule
```

Data Flow

```
//1960628 Prem
//encoder 8x3 Data Flow
module encoder(
    input d0,
    input d1,
    input d2,
    input d3,
    input d4,
    input d5,
    input d6,
    input d7,
    output y0,
    output y1,
    output y2
    );
assign y0 = d7+d5+d3+d1;
assign y1 = d7+d6+d2+d3;
assign y2 = d7+d6+d5+d4;
endmodule
```

Behavioural

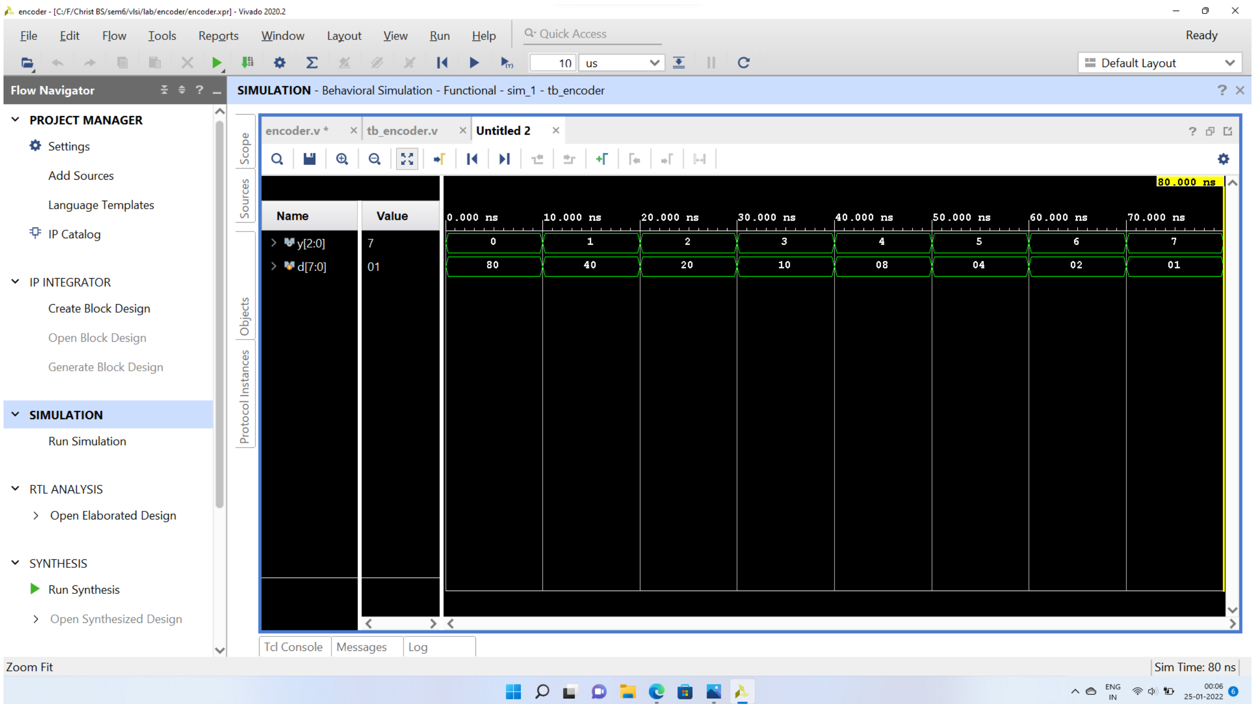
```
//1960628 Prem
//encoder 8x3 Behavioural
module encoder(
    output reg [2:0]y,
    input [7:0]d
    );
always @(d,y)
    begin
        case(d)
        8'b10000000:begin y=3'b000; end
        8'b01000000:begin y=3'b001; end
        8'b00100000:begin y=3'b010; end
        8'b00010000:begin y=3'b011; end
        8'b00001000:begin y=3'b100; end
        8'b00000100:begin y=3'b101; end
        8'b00000010:begin y=3'b110; end
        8'b00000001:begin y=3'b111; end
        endcase
    end
endmodule
```

TESTBENCH

```
//1960628 Prem
//encoder 8x3 Testbench
module tb_encoder();
wire [2:0]y;
reg [7:0]d;

encoder I1(y,d);
initial
begin
d = 8'b10000000;
#10
d = 8'b01000000;
#10
d = 8'b00100000;
#10
d = 8'b00010000;
#10
d = 8'b00001000;
#10
d = 8'b00000100;
#10
d = 8'b00000010;
#10
d = 8'b00000001;
#10
$finish();
end
endmodule
```

OUTPUT



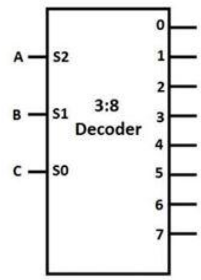
Result:

Thus the Verilog code was scripted, simulated and waveforms were verified using Xilinx Vivado Webpack.

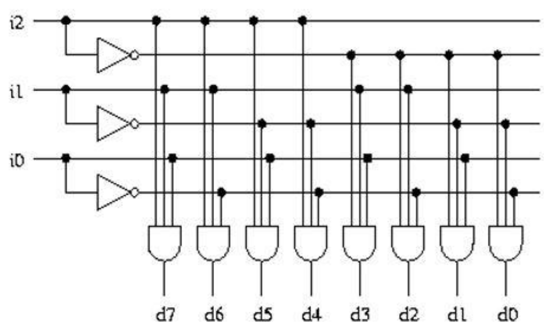
3x8 Decoder:

Decoder:

Logical Symbol



Logic Circuit



Truth Table

A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Expression:

D0 = A2' A1' A0'

D1 = A2' A1' A0

D2 = A2' A1 A0'

D3 = A2' A1 A0

D4 = A2 A1' A0'

D5 = A2 A1' A0

D6 = A2 A1 A0'

D7 = A2 A1 A0

PROGRAM

Gate Level

```
//1960628 Prem
//Demux 3x8 Gate level
module decoder_3x8(
    input s0,
    input s1,
    input s2,
    output d0,
    output d1,
    output d2,
    output d3,
    output d4,
    output d5,
    output d6,
    output d7
    );
wire w0,w1,w2;
not no1(w0,s0);
not no2(w1,s1);
not no3(w2,s2);

and and0(d0,w2,w1,w0);
and and1(d1,w2,w1,s0);
and and2(d2,w2,s1,w0);
and and3(d3,w2,s1,s0);
and and4(d4,s2,w1,w0);
and and5(d5,s2,w1,s0);
and and6(d6,s2,s1,w0);
and and7(d7,s2,s1,s0);
endmodule
```

Data Flow

```
//1960628 Prem
//3x8 decoder Dataflow
module decoder_3x8(
    input s0,
    input s1,
    input s2,
    output d0,
    output d1,
    output d2,
    output d3,
    output d4,
    output d5,
    output d6,
    output d7
    );

assign w0 = ~s0;
assign w1 = ~s1;
assign w2 = ~s2;
assign d0 = w2&w1&w0;
assign d1 = w2&w1&s0;
assign d2 = w2&s1&w0;
assign d3 = w2&s1&s0;
assign d4 = s2&w1&w0;
assign d5 = s2&w1&s0;
assign d6 = s2&s1&w0;
assign d7 = s2&s1&s0;
endmodule
```

Behavioural

```
//1960628 Prem
//3x8 Decoder Behavioural Modelling
module decoder_3x8(
    input s0,
    input s1,
    input s2,
    output reg d0,
    output reg d1,
    output reg d2,
    output reg d3,
    output reg d4,
    output reg d5,
    output reg d6,
    output reg d7
    );
always @(s0,s1,s2)
    begin
        case ({s2,s1,s0})
        3'b000: begin d0=0;d1=0;d2=0;d3=0;d4=0;d5=0;d6=0;d7=0;end
        3'b001: begin d0=0;d1=1;d2=0;d3=0;d4=0;d5=0;d6=0;d7=0;end
        3'b010: begin d0=0;d1=0;d2=1;d3=0;d4=0;d5=0;d6=0;d7=0;end
        3'b011: begin d0=0;d1=0;d2=0;d3=1;d4=0;d5=0;d6=0;d7=0;end
        3'b100: begin d0=0;d1=0;d2=0;d3=0;d4=1;d5=0;d6=0;d7=0;end
        3'b101: begin d0=0;d1=0;d2=0;d3=0;d4=0;d5=1;d6=0;d7=0;end
        3'b110: begin d0=0;d1=0;d2=0;d3=0;d4=0;d5=0;d6=1;d7=0;end
        3'b111: begin d0=0;d1=0;d2=0;d3=0;d4=0;d5=0;d6=0;d7=1;end
        default: begin d0=0;d1=0;d2=0;d3=0;d4=0;d5=0;d6=0;d7=0;end
        endcase
    end

endmodule
```

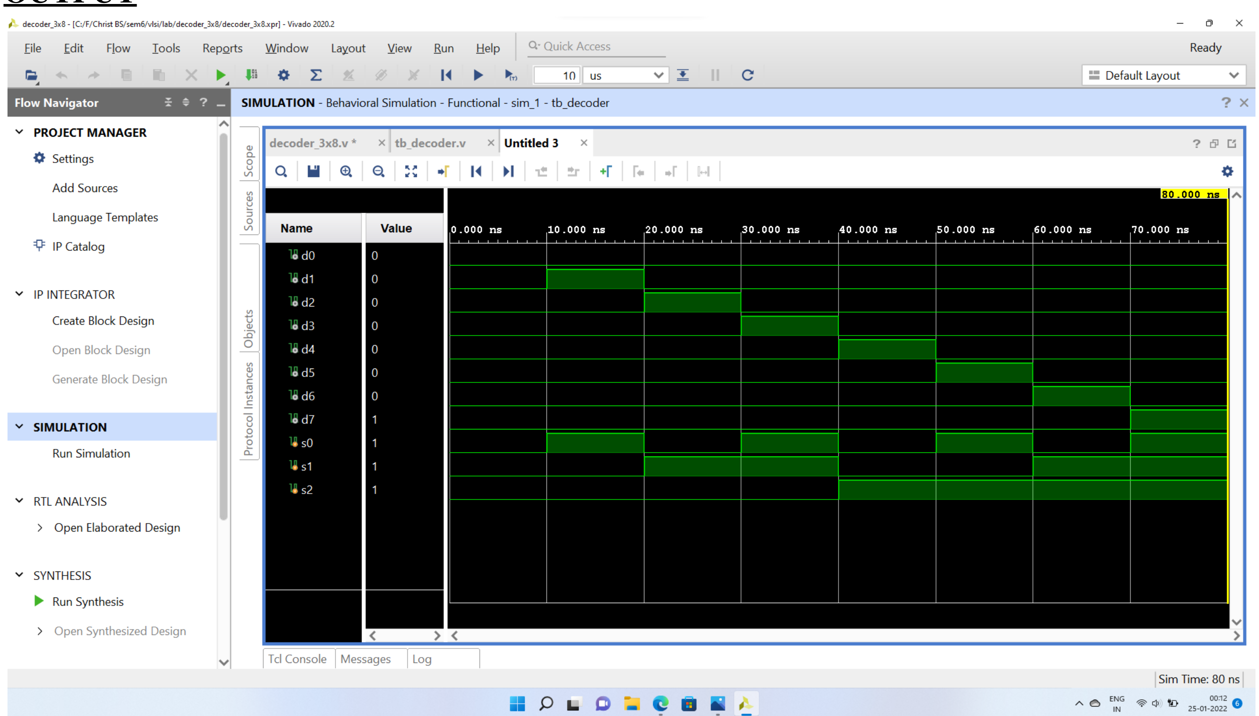
TESTBENCH

```
//1960628 Prem
//3x8 decoder testbench
module tb_decoder();
wire d0,d1,d2,d3,d4,d5,d6,d7;
reg s0,s1,s2;

decoder_3x8 I1(s0,s1,s2,d0,d1,d2,d3,d4,d5,d6,d7);
initial
begin
s0 = 1'b0;
s1 = 1'b0;
s2 = 1'b0;
#10
s0 = 1'b1;
s1 = 1'b0;
s2 = 1'b0;
#10
s0 = 1'b0;
s1 = 1'b1;
s2 = 1'b0;
#10
s0 = 1'b1;
s1 = 1'b1;
s2 = 1'b0;
#10
s0 = 1'b0;
s1 = 1'b0;
s2 = 1'b1;
#10
s0 = 1'b1;
s1 = 1'b0;
s2 = 1'b1;
#10
s0 = 1'b0;
s1 = 1'b1;
s2 = 1'b1;
```

```
#10
s0 = 1'b1;
s1 = 1'b1;
s2 = 1'b1;
#10
$finish();
end
endmodule
```

OUTPUT



Result:

Thus the Verilog code was scripted, simulated and waveforms were verified using Xilinx Vivado Webpack.