BEHAVIOURAL LEVEL MODELING

LAB # 05

Spring 2021

CSE-308L Digital System Design Lab

Submitted by: **ASHIQ ULLAH**

Registration No.: **18PWCSE1695**

Class Section: **B**

"On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work."

Student Signature: ______________

Submitted to:

Engr. Madiha Sher

May 23, 2021

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

OBJECTIVES:

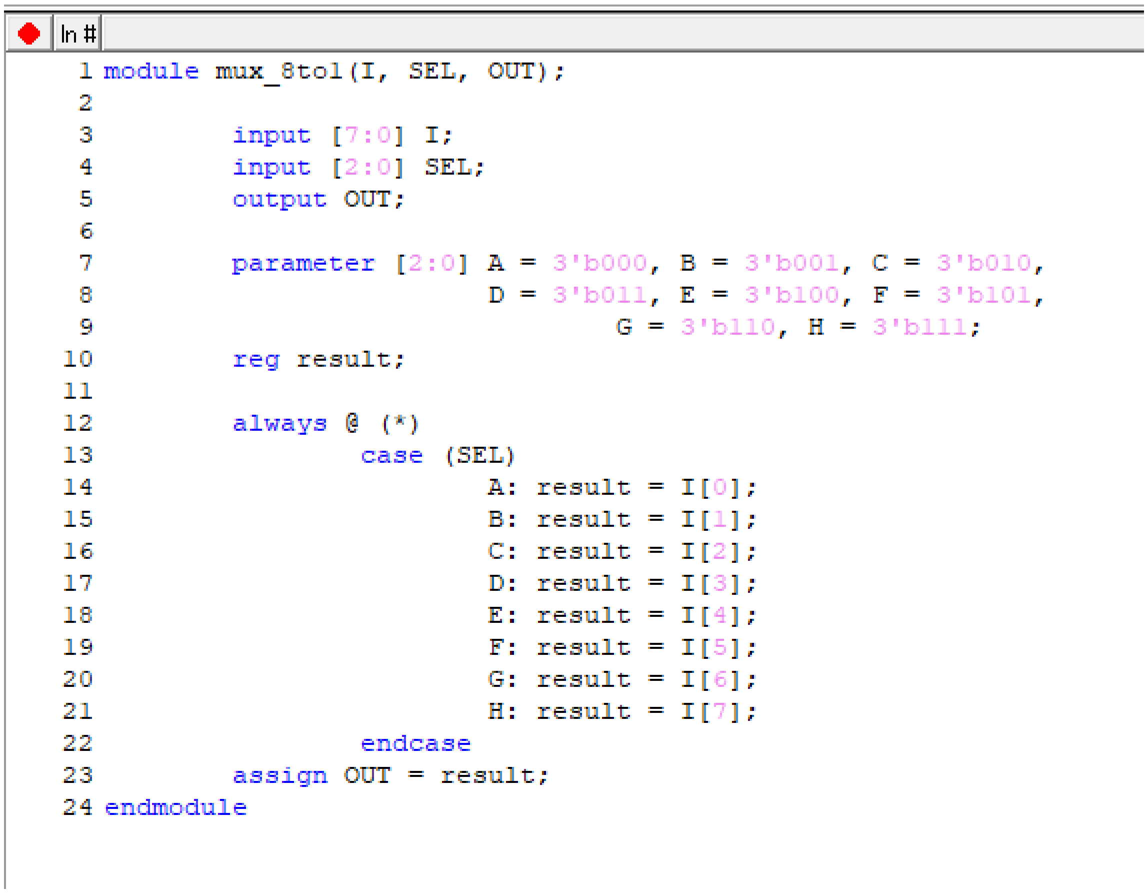
This lab will enable students to:

- Code using Behavioral level modeling
- Implement multiplexer and demultiplexer and decoder

TASK01:

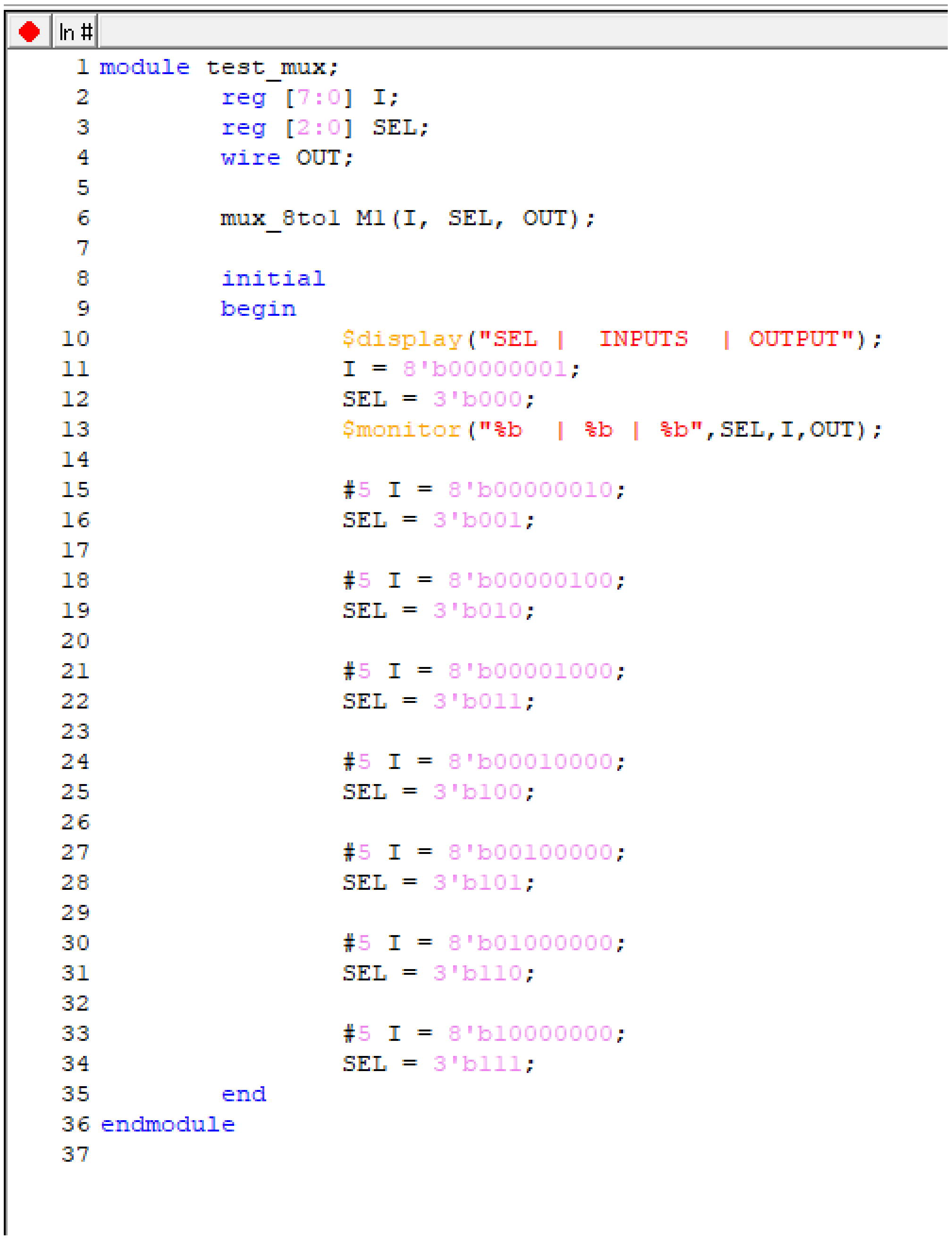
Implementation of 8x1 multiplexer (using case)

CODE:



```
module mux_8to1(I, SEL, OUT);

        input [7:0] I;
        input [2:0] SEL;
        output OUT;

        parameter [2:0] A = 3'b000, B = 3'b001, C = 3'b010,
                        D = 3'b011, E = 3'b100, F = 3'b101,
                                G = 3'b110, H = 3'b111;
        reg result;

        always @ (*)
                case (SEL)
                        A: result = I[0];
                        B: result = I[1];
                        C: result = I[2];
                        D: result = I[3];
                        E: result = I[4];
                        F: result = I[5];
                        G: result = I[6];
                        H: result = I[7];
                endcase
        assign OUT = result;
endmodule
```

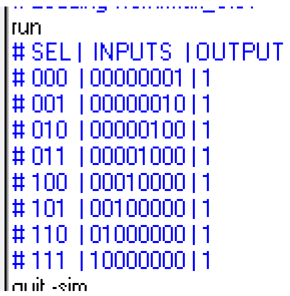
TestBench:


ln #

```
module test_mux;
        reg [7:0] I;
        reg [2:0] SEL;
        wire OUT;

        mux_8to1 M1(I, SEL, OUT);

        initial
        begin
                $display("SEL |  INPUTS   | OUTPUT");
                I = 8'b00000001;
                SEL = 3'b000;
                $monitor("%b  | %b | %b",SEL,I,OUT);

                #5 I = 8'b00000010;
                SEL = 3'b001;

                #5 I = 8'b00000100;
                SEL = 3'b010;

                #5 I = 8'b00001000;
                SEL = 3'b011;

                #5 I = 8'b00010000;
                SEL = 3'b100;

                #5 I = 8'b00100000;
                SEL = 3'b101;

                #5 I = 8'b01000000;
                SEL = 3'b110;

                #5 I = 8'b10000000;
                SEL = 3'b111;
        end
endmodule
```

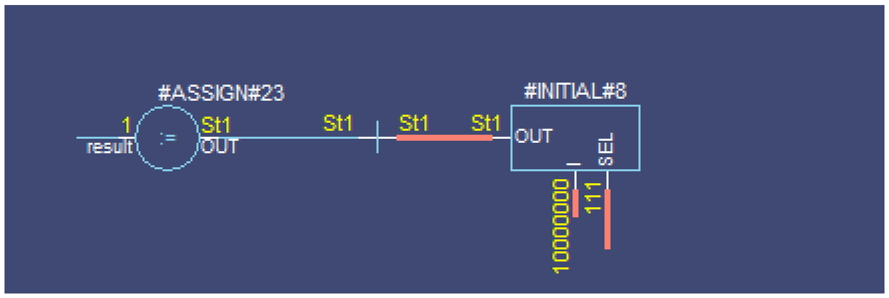
OUTPUTS

Truth Table:

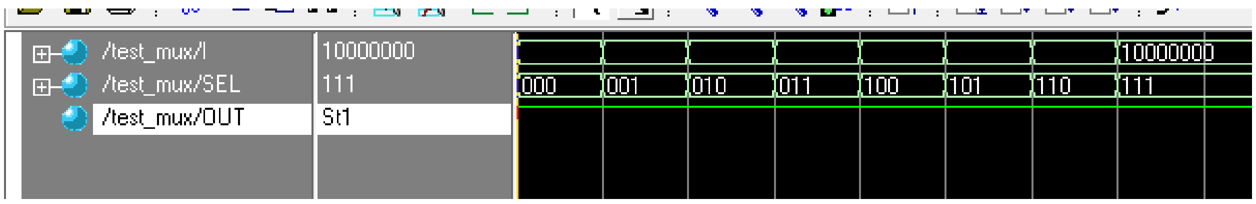


```
run
# SEL | INPUTS  | OUTPUT
# 000  | 00000001 | 1
# 001  | 00000010 | 1
# 010  | 00000100 | 1
# 011  | 00001000 | 1
# 100  | 00010000 | 1
# 101  | 00100000 | 1
# 110  | 01000000 | 1
# 111  | 10000000 | 1
quit -sim
```

Data Flow:



Wave Form:

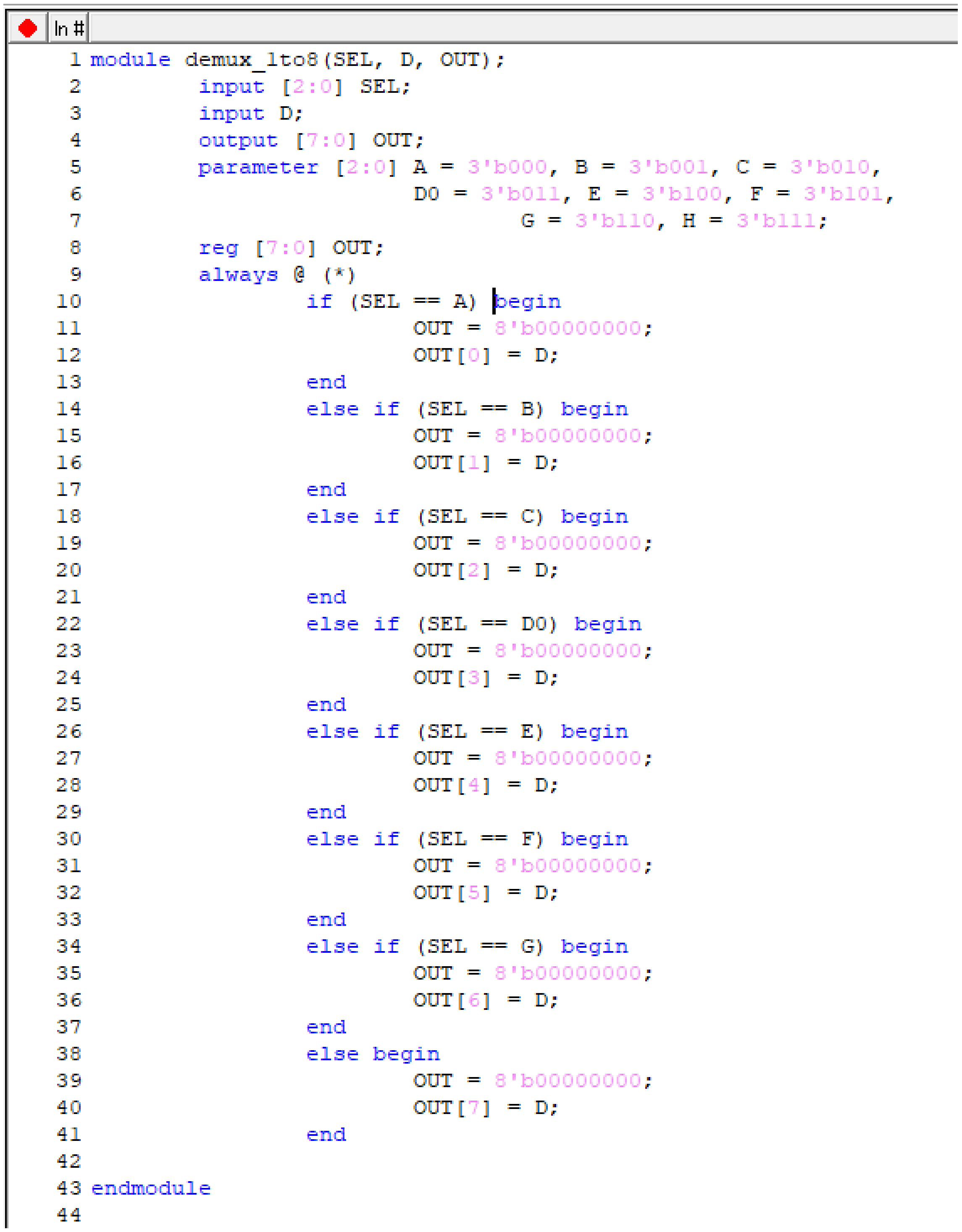


```
/test_mux/I    10000000
/test_mux/SEL  111
/test_mux/OUT  St1
```

<u>TASK02</u>:

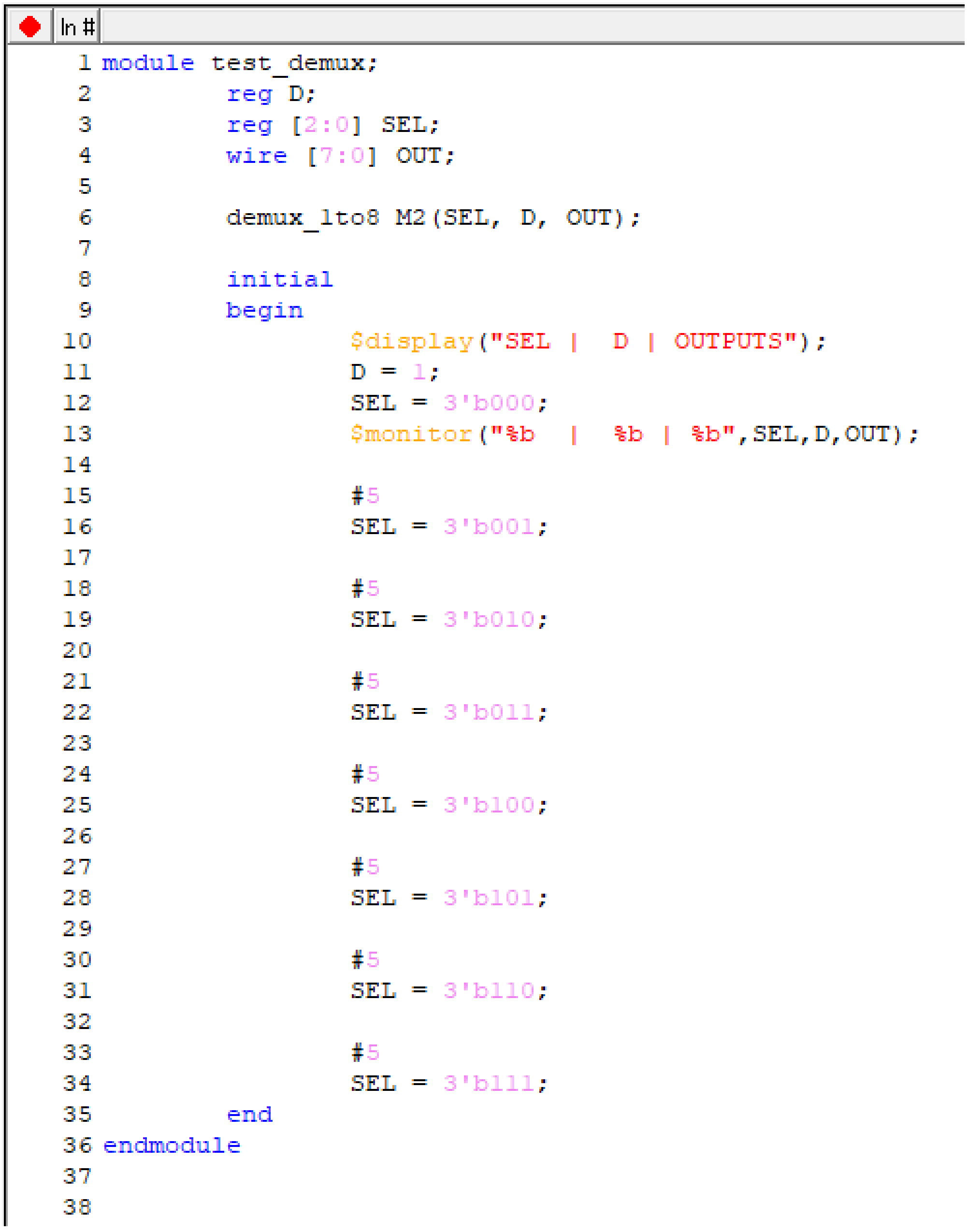
Implementation of 1x8 demultiplexer (using if/else)

<u>CODE</u>:



```
module demux_1to8(SEL, D, OUT);
        input [2:0] SEL;
        input D;
        output [7:0] OUT;
        parameter [2:0] A = 3'b000, B = 3'b001, C = 3'b010,
                        D0 = 3'b011, E = 3'b100, F = 3'b101,
                                G = 3'b110, H = 3'b111;
        reg [7:0] OUT;
        always @ (*)
                if (SEL == A) begin
                        OUT = 8'b00000000;
                        OUT[0] = D;
                end
                else if (SEL == B) begin
                        OUT = 8'b00000000;
                        OUT[1] = D;
                end
                else if (SEL == C) begin
                        OUT = 8'b00000000;
                        OUT[2] = D;
                end
                else if (SEL == D0) begin
                        OUT = 8'b00000000;
                        OUT[3] = D;
                end
                else if (SEL == E) begin
                        OUT = 8'b00000000;
                        OUT[4] = D;
                end
                else if (SEL == F) begin
                        OUT = 8'b00000000;
                        OUT[5] = D;
                end
                else if (SEL == G) begin
                        OUT = 8'b00000000;
                        OUT[6] = D;
                end
                else begin
                        OUT = 8'b00000000;
                        OUT[7] = D;
                end

endmodule

```

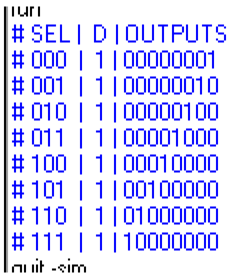
TestBench:



```
module test_demux;
        reg D;
        reg [2:0] SEL;
        wire [7:0] OUT;

        demux_1to8 M2(SEL, D, OUT);

        initial
        begin
                $display("SEL |  D | OUTPUTS");
                D = 1;
                SEL = 3'b000;
                $monitor("%b  |  %b | %b",SEL,D,OUT);

                #5
                SEL = 3'b001;

                #5
                SEL = 3'b010;

                #5
                SEL = 3'b011;

                #5
                SEL = 3'b100;

                #5
                SEL = 3'b101;

                #5
                SEL = 3'b110;

                #5
                SEL = 3'b111;
        end
endmodule
```

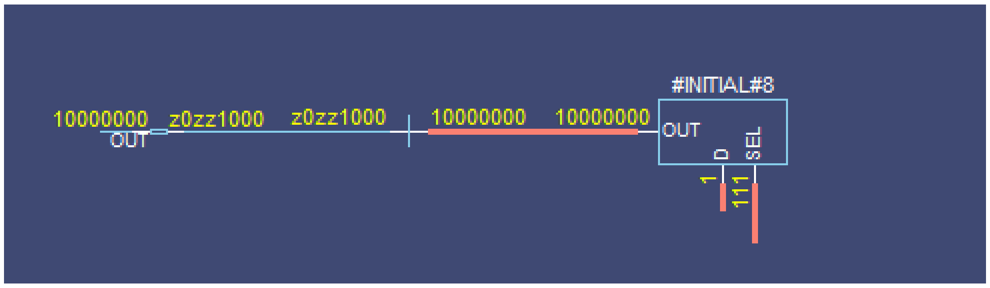
OUTPUTS

Truth Table:

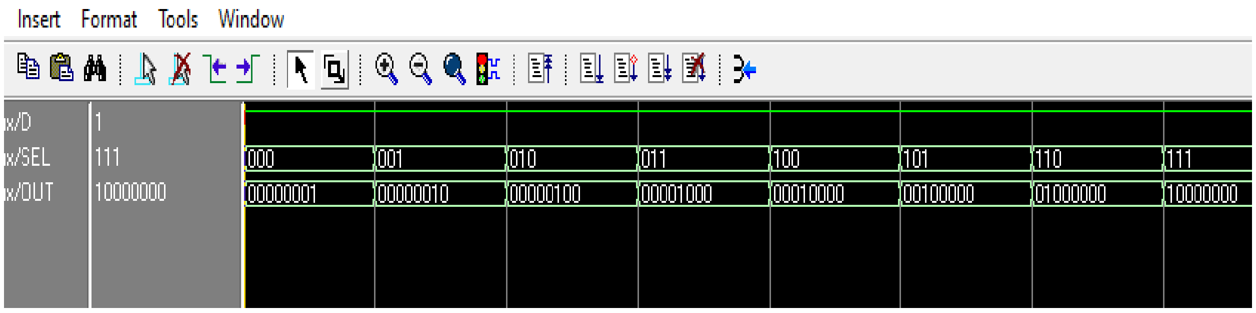


```
run
# SEL |  D | OUTPUTS
# 000  |  1 | 00000001
# 001  |  1 | 00000010
# 010  |  1 | 00000100
# 011  |  1 | 00001000
# 100  |  1 | 00010000
# 101  |  1 | 00100000
# 110  |  1 | 01000000
# 111  |  1 | 10000000
quit -sim
```

Data Flow:



Wave Form:

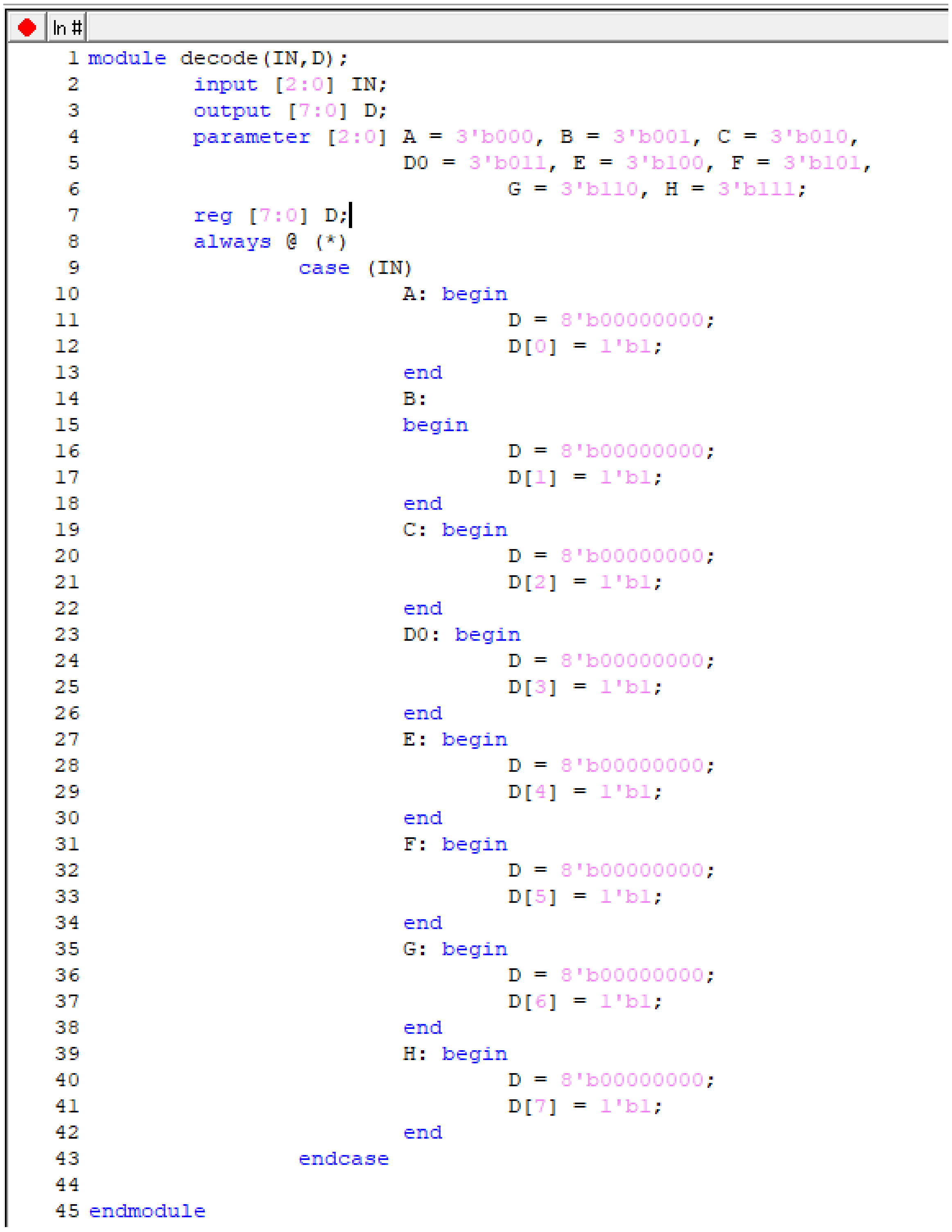


```
x/D    1
x/SEL  111       000       001       010       011       100       101       110       111
x/OUT  10000000  00000001  00000010  00000100  00001000  00010000  00100000  01000000  10000000
```

TASK03:

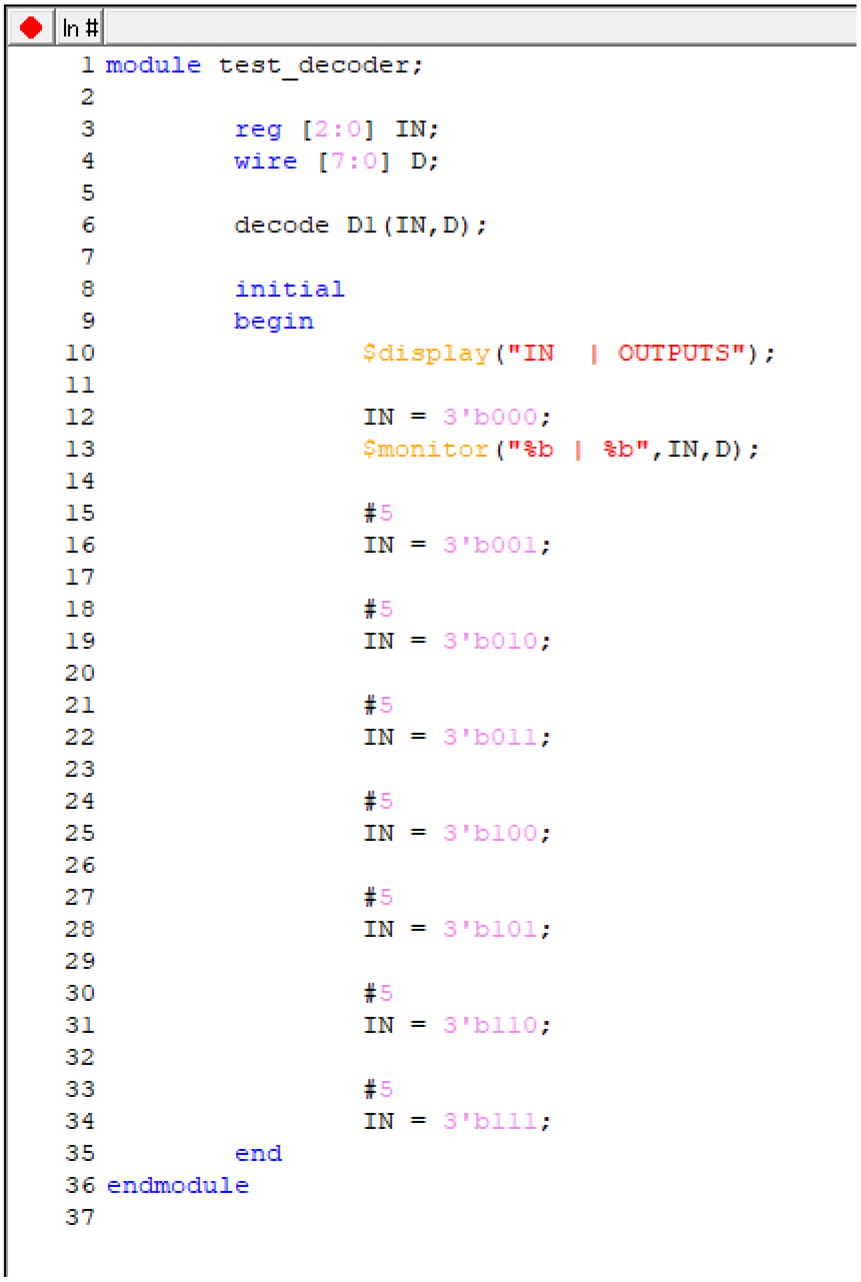
Implementation of 3x8 decoder

CODE:



```verilog
module decode(IN,D);
        input [2:0] IN;
        output [7:0] D;
        parameter [2:0] A = 3'b000, B = 3'b001, C = 3'b010,
                        D0 = 3'b011, E = 3'b100, F = 3'b101,
                                G = 3'b110, H = 3'b111;
        reg [7:0] D;
        always @ (*)
                case (IN)
                        A: begin
                                D = 8'b00000000;
                                D[0] = 1'b1;
                        end
                        B:
                        begin
                                D = 8'b00000000;
                                D[1] = 1'b1;
                        end
                        C: begin
                                D = 8'b00000000;
                                D[2] = 1'b1;
                        end
                        D0: begin
                                D = 8'b00000000;
                                D[3] = 1'b1;
                        end
                        E: begin
                                D = 8'b00000000;
                                D[4] = 1'b1;
                        end
                        F: begin
                                D = 8'b00000000;
                                D[5] = 1'b1;
                        end
                        G: begin
                                D = 8'b00000000;
                                D[6] = 1'b1;
                        end
                        H: begin
                                D = 8'b00000000;
                                D[7] = 1'b1;
                        end
                endcase

endmodule
```

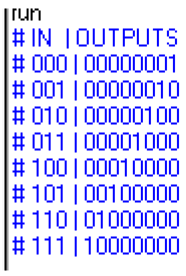
TestBench:



```
module test_decoder;

        reg [2:0] IN;
        wire [7:0] D;

        decode D1(IN,D);

        initial
        begin
                $display("IN  | OUTPUTS");

                IN = 3'b000;
                $monitor("%b | %b",IN,D);

                #5
                IN = 3'b001;

                #5
                IN = 3'b010;

                #5
                IN = 3'b011;

                #5
                IN = 3'b100;

                #5
                IN = 3'b101;

                #5
                IN = 3'b110;

                #5
                IN = 3'b111;
        end
endmodule
```

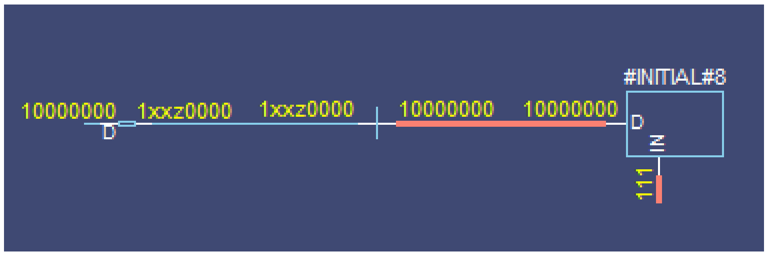
OUTPUTS

Truth Table:

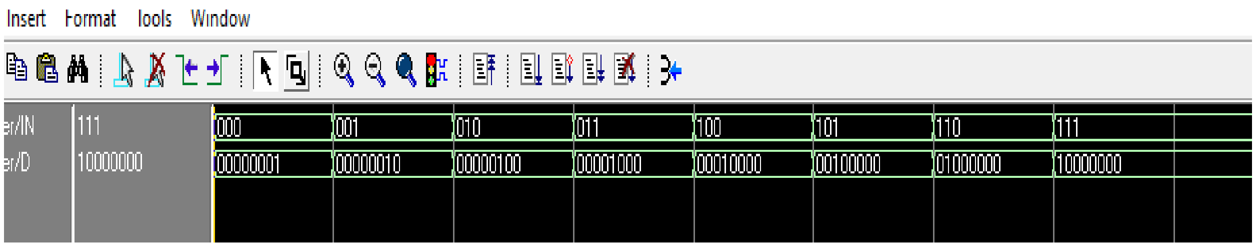


```
run
# IN  | OUTPUTS
# 000 | 00000001
# 001 | 00000010
# 010 | 00000100
# 011 | 00001000
# 100 | 00010000
# 101 | 00100000
# 110 | 01000000
# 111 | 10000000
```

Data Flow:



Wave Form:



```
er/IN  111       000      001      010      011      100      101      110      111
er/D   10000000  00000001 00000010 00000100 00001000 00010000 00100000 01000000 10000000
```