Embedded Systems

CS429 Lab7

Name: Dipean Dasgupta **ID:202151188**

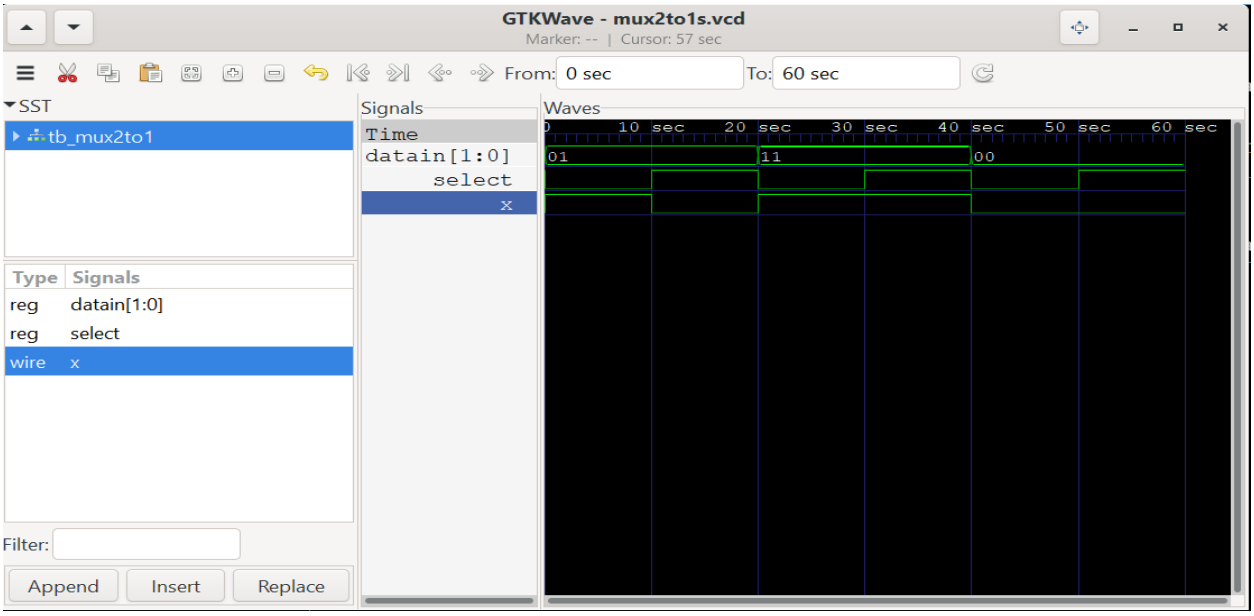
Task1(a): Design 2:1 multiplexer in Verilog.The module specifications are as follows:
module mux2to1(output x, input [1:0] datain, input select);

```
module and_gate(output and_out, input a, input b);
    assign and_out = a & b;
endmodule

module or_gate(output or_out, input a, input b);
    assign or_out = a | b;
endmodule

module mux2to1(output x, input [1:0] datain, input select);
    wire and1_out, and2_out;
    wire or_out;

    // AND gates for datain[0] and ~select, and datain[1] and select
    and_gate AND1 (.and_out(and1_out), .a(datain[0]), .b(~select));
    and_gate AND2 (.and_out(and2_out), .a(datain[1]), .b(select));

    // OR gate=combine the outputs of AND
    or_gate OR1 (.or_out(or_out), .a(and1_out), .b(and2_out));

    // Output
    assign x = or_out;
endmodule

module tb_mux2to1();
    reg [1:0] datain;
    reg select;
    wire x;

    mux2to1 mux_inst (.x(x), .datain(datain), .select(select));

    initial begin
        $dumpfile("mux2to1s.vcd");
        $dumpvars(0, tb_mux2to1);
        // Test Case 1: When select is 0, output=datain[0]
        datain = 2'b10;
```

```
        select = 0;
        #10; // Wait for a few simulation cycles
        $display("Test Case 1: datain=%b, select=%b, x=%b", datain, select, x);

        // Test Case 2: When select is 1, output= datain[1]
        datain = 2'b01;
        select = 1;
        #10; // Wait for a few simulation cycles
        $display("Test Case 2: datain=%b, select=%b, x=%b", datain, select, x);

        // Test Case 3:both i/p are 1, o/p  be 1 regardless of select
        datain = 2'b11;
        select = 0;
        #10;
        $display("Test Case 3a: datain=%b, select=%b, x=%b", datain, select, x);

        select = 1;
        #10;
        $display("Test Case 3b: datain=%b, select=%b, x=%b", datain, select, x);

        // Test Case 4:both i/p are 0, o/p should be 0
        datain = 2'b00;
        select = 0;
        #10;
        $display("Test Case 4a: datain=%b, select=%b, x=%b", datain, select, x);

        select = 1;
        #10;
        $display("Test Case 4b: datain=%b, select=%b, x=%b", datain, select, x);
        $finish;
    end
endmodule
```

Output:

```
[Running] lab7_1A.v
VCD info: dumpfile mux2to1s.vcd opened for output.
Test Case 1: datain=01, select=0, x=1
Test Case 2: datain=01, select=1, x=0
Test Case 3a: datain=11, select=0, x=1
Test Case 3b: datain=11, select=1, x=1
Test Case 4a: datain=00, select=0, x=0
Test Case 4b: datain=00, select=1, x=0
lab7_1A.v:72: $finish called at 60 (1s)
[Done] exit with code=0 in 0.314 seconds
```



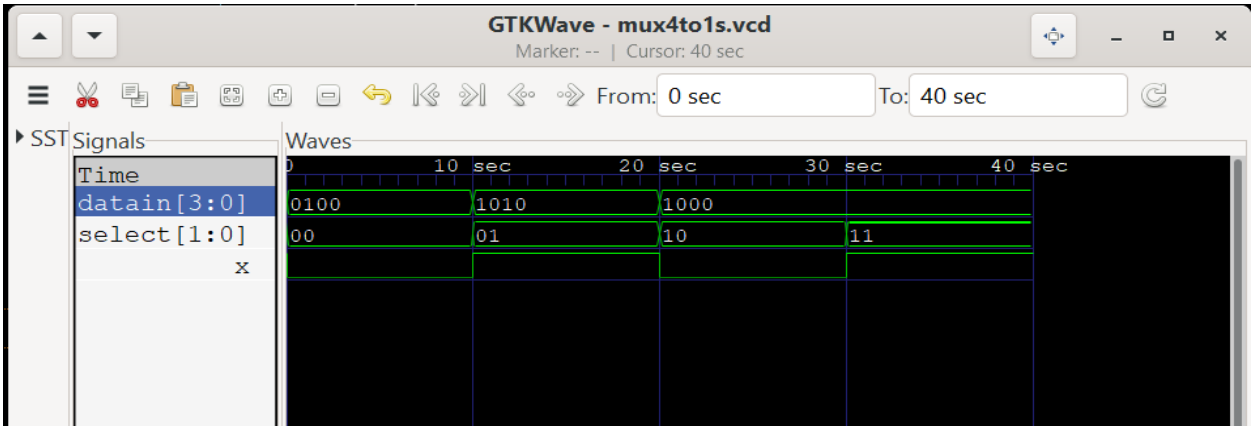
Task1(b): Design 4:1 multiplexer in Verilog.The module specifications are as follows:module mux4to1(output x, input [3:0]datain, input [1:0]select);Using 2:1 multiplexer as instance model.

```
module mux2to1(output y, input a, input b, input select);
    assign y = (select) ? b : a;
endmodule

module mux4to1(output x, input [3:0] datain, input [1:0] select);
    wire m0, m1;
    mux2to1 u1 (.y(m0), .a(datain[0]), .b(datain[1]), .select(select[0]));
    mux2to1 u2 (.y(m1), .a(datain[2]), .b(datain[3]), .select(select[0]));
    mux2to1 u3 (.y(x), .a(m0), .b(m1), .select(select[1]));
endmodule

module tb_mux4to1();
    reg [3:0] datain;
    reg [1:0] select;
    wire x;
    mux4to1 mux_inst (.x(x), .datain(datain), .select(select));

    initial begin
        $dumpfile("mux4to1s.vcd");
        $dumpvars(0, tb_mux4to1);
        // Test1: Selecting datain[0]
        datain = 4'b0100;
        select = 2'b00;
```

```
        #10;
        $display("Test Case 1: datain=%b, select=%b, x=%b", datain, select, x);
        // Test2: Selecting datain[1]
        datain = 4'b1010;
        select = 2'b01;
        #10;
        $display("Test Case 2: datain=%b, select=%b, x=%b", datain, select, x);
        // Test3: Selecting datain[2]
        datain = 4'b1000;
        select = 2'b10;
        #10;
        $display("Test Case 3: datain=%b, select=%b, x=%b", datain, select, x);
        // Test4: Selecting datain[3]
        datain = 4'b1000;
        select = 2'b11;
        #10;
        $display("Test Case 4: datain=%b, select=%b, x=%b", datain, select,
x);
        $finish;
    end
endmodule
```

OUTPUT:

```
[Running] lab7_1B.v
VCD info: dumpfile mux4to1s.vcd opened for output.
Test Case 1: datain=0100, select=00, x=0
Test Case 2: datain=1010, select=01, x=1
Test Case 3: datain=1000, select=10, x=0
Test Case 4: datain=1000, select=11, x=1
lab7_1B.v:44: $finish called at 40 (1s)
[Done] exit with code=0 in 0.284 seconds
```



Task2: Design a 3:8 decoder with all 3 levels of abstraction.

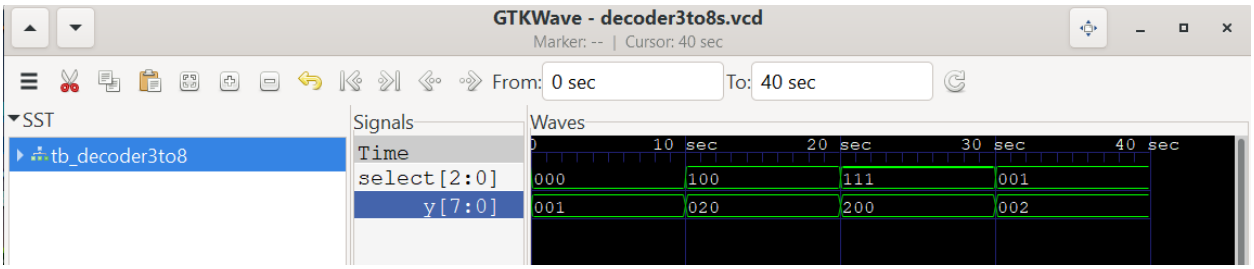
Structural level

```
module and_gate(output and_out, input a, input b);
    assign and_out = a & b;
endmodule
module decoder3to8(output [7:0] y, input [2:0] select);
    wire [2:0] inv_select;
    wire [7:0] and_outputs;
    // Invert the select lines
    assign inv_select = ~select;
    and_gate u0 (.and_out(and_outputs[0]), .a(inv_select[2]), .b(inv_select[1] &
inv_select[0]));
    and_gate u1 (.and_out(and_outputs[1]), .a(inv_select[2]), .b(inv_select[1] &
select[0]));
    and_gate u2 (.and_out(and_outputs[2]), .a(inv_select[2]), .b(select[1] &
inv_select[0]));
    and_gate u3 (.and_out(and_outputs[3]), .a(inv_select[2]), .b(select[1] &
select[0]));
    and_gate u4 (.and_out(and_outputs[4]), .a(select[2]), .b(inv_select[1] &
inv_select[0]));
    and_gate u5 (.and_out(and_outputs[5]), .a(select[2]), .b(inv_select[1] &
select[0]));
    and_gate u6 (.and_out(and_outputs[6]), .a(select[2]), .b(select[1] &
inv_select[0]));
    and_gate u7 (.and_out(and_outputs[7]), .a(select[2]), .b(select[1] &
select[0]));
    assign y = and_outputs;
endmodule

module tb_decoder3to8();
    reg [2:0] select;
    wire [7:0] y;

    decoder3to8 decoder_inst (.y(y), .select(select));

    initial begin
        $dumpfile("decoder3to8s.vcd");
        $dumpvars(0, tb_decoder3to8);
        // Test1: Selecting line 0 (000)
        select = 3'b000;
        #10;
        $display("Test Case 1: select=%b, y=%b", select, y);
        // Test2: Selecting line 4 (100)
        select = 3'b100;
```

```
        #10;
        $display("Test Case 2: select=%b, y=%b", select, y);
        // Test3: Selecting line 7 (111)
        select = 3'b111;
        #10;
        $display("Test Case 3: select=%b, y=%b", select, y);
        // Test4: Selecting line 1 (001)
        select = 3'b001;
        #10;
        $display("Test Case 4: select=%b, y=%b", select, y);
        $finish;
    end
endmodule
```

OUTPUT:

```
VCD info: dumpfile decoder3to8s.vcd opened for output.
Test Case 1: select=000, y=00000001
Test Case 2: select=100, y=00010000
Test Case 3: select=111, y=10000000
Test Case 4: select=001, y=00000010
lab7_3a.v:45: $finish called at 40 (1s)
[Done] exit with code=0 in 0.358 seconds
```



RTL level

```
module decoder3to8 (
  input [2:0] in,
  output [7:0] out
);
reg [7:0] out;

  always @(in) begin
    case (in)
      3'b000: out <= 8'b00000001;
      3'b001: out <= 8'b00000010;
      3'b010: out <= 8'b00000100;
```

```
      3'b011: out <= 8'b00001000;
      3'b100: out <= 8'b00010000;
      3'b101: out <= 8'b00100000;
      3'b110: out <= 8'b01000000;
      3'b111: out <= 8'b10000000;
      default: out <= 8'b00000000;
    endcase
  end
endmodule

module tb_decoder3to8;
  reg [2:0] in;
  wire [7:0] out;
  decoder3to8 decoder (in, out);
  initial begin
    $display("Input | Output");
    $display("------+--------");
    in = 3'b000;
    #1;
    $display("%b   | %b", in, out);
    in = 3'b001;
    #1;
    $display("%b   | %b", in, out);
    in = 3'b111;
    #1;
    $display("%b   | %b", in, out);
    $finish;
  end
endmodule
```

OUTPUT:

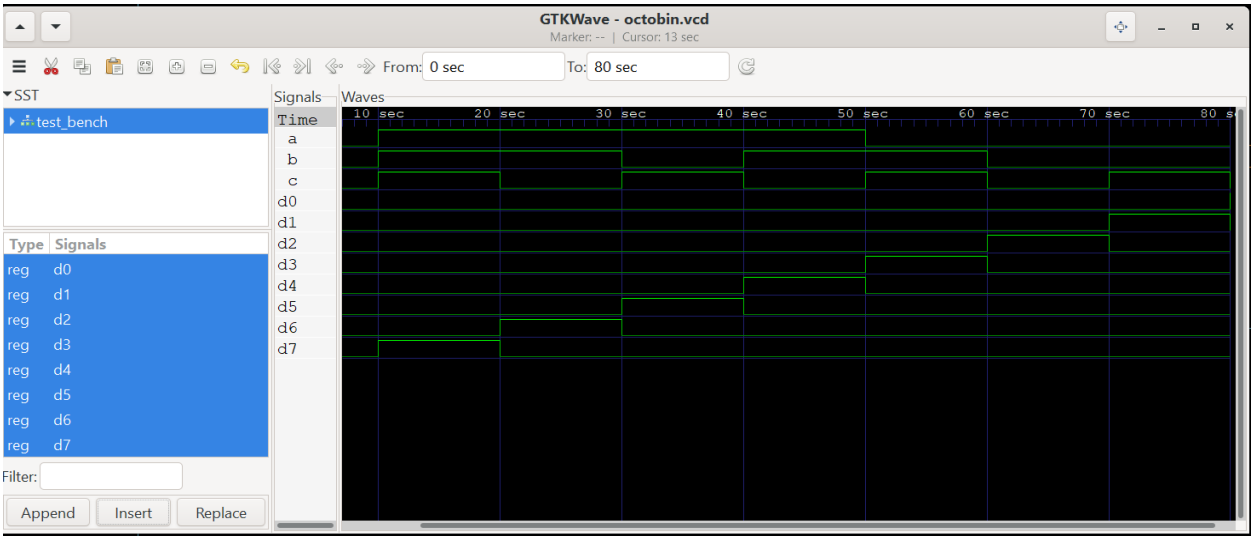
```
[Running] lab7_3.v
Input | Output
------+--------
000   | 00000001
001   | 00000010
111   | 10000000
lab7_3.v:97: $finish called at 3 (1s)
```

Task3: Implement octal to binary encoder.

Code:

```
module octal_to_binary_encoder(a,b,c,d0,d1,d2,d3,d4,d5,d6,d7);
output a,b,c;
input d0,d1,d2,d3,d4,d5,d6,d7;
or(a,d4,d5,d6,d7);
or(b,d3,d4,d6,d7);
or(c,d1,d3,d5,d7);
endmodule

module test_bench;
reg d0,d1,d2,d3,d4,d5,d6,d7;
wire a,b,c;
octal_to_binary_encoder uut(a,b,c,d0,d1,d2,d3,d4,d5,d6,d7);
initial begin
$dumpfile("dump.vcd");
$dumpvars(0,test_bench);
d0=0;d1=0;d2=0;d3=0;d4=0;d5=0;d6=0;d7=0; #10;
d0=0;d1=0;d2=0;d3=0;d4=0;d5=0;d6=0;d7=1; #10;
d0=0;d1=0;d2=0;d3=0;d4=0;d5=0;d6=1;d7=0; #10;
d0=0;d1=0;d2=0;d3=0;d4=0;d5=1;d6=0;d7=0; #10;
d0=0;d1=0;d2=0;d3=0;d4=1;d5=0;d6=0;d7=0; #10;
d0=0;d1=0;d2=0;d3=1;d4=0;d5=0;d6=0;d7=0; #10;
d0=0;d1=0;d2=1;d3=0;d4=0;d5=0;d6=0;d7=0; #10;
d0=0;d1=1;d2=0;d3=0;d4=0;d5=0;d6=0;d7=0; #10;
d0=1;d1=0;d2=0;d3=0;d4=0;d5=0;d6=0;d7=0;
$finish;
end
```

OUTPUT:



Task4: Design a 1:4, 1:8, 1:16 and 1:32 demultiplexer in all three levels of abstraction.

```
module demux_1to4 (
    input wire enable,
    input wire [1:0] select,
    input wire data_in,
    output reg [3:0] demux_out
);
    always @(select, enable) begin
        if (enable) begin
            case(select)
                2'b00: demux_out = 4'b0001;
                2'b01: demux_out = 4'b0010;
                2'b10: demux_out = 4'b0100;
                2'b11: demux_out = 4'b1000;
                default: demux_out = 4'b0000;
            endcase
        end
        else begin
            demux_out = 4'b0000;
        end
    end
endmodule
module demux_1to8 (
    input wire enable,
    input wire [2:0] select,
    input wire data_in,
    output reg [7:0] demux_out
);
    always @(select, enable) begin
        if (enable) begin
            case(select)
                3'b000: demux_out = 8'b00000001;
                3'b001: demux_out = 8'b00000010;
                3'b010: demux_out = 8'b00000100;
                3'b011: demux_out = 8'b00001000;
                3'b100: demux_out = 8'b00010000;
                3'b101: demux_out = 8'b00100000;
                3'b110: demux_out = 8'b01000000;
                3'b111: demux_out = 8'b10000000;
                default: demux_out = 8'b00000000;
            endcase
        end
        else begin
```

```
            demux_out = 8'b00000000;
        end
    end
endmodule
module demux_1to16 (
    input wire enable,
    input wire [3:0] select,
    input wire data_in,
    output reg [15:0] demux_out
);
    always @(select, enable) begin
        if (enable) begin
            case(select)
                4'b0000: demux_out = 16'b0000000000000001;
                4'b0001: demux_out = 16'b0000000000000010;
                4'b0010: demux_out = 16'b0000000000000100;
                4'b0011: demux_out = 16'b0000000000001000;
                4'b0100: demux_out = 16'b0000000000010000;
                4'b0101: demux_out = 16'b0000000000100000;
                4'b0110: demux_out = 16'b0000000001000000;
                4'b0111: demux_out = 16'b0000000010000000;
                4'b1000: demux_out = 16'b0000000100000000;
                4'b1001: demux_out = 16'b0000001000000000;
                4'b1010: demux_out = 16'b0000010000000000;
                4'b1011: demux_out = 16'b0000100000000000;
                4'b1100: demux_out = 16'b0001000000000000;
                4'b1101: demux_out = 16'b0010000000000000;
                4'b1110: demux_out = 16'b0100000000000000;
                4'b1111: demux_out = 16'b1000000000000000;
                default: demux_out = 16'b0000000000000000;
            endcase
        end
        else begin
            demux_out = 16'b0000000000000000;
        end
    end
endmodule
module demux_1to32 (
    input wire enable,
    input wire [4:0] select,
    input wire data_in,
    output reg [31:0] demux_out
);
    always @(select, enable) begin
        if (enable) begin
```

```
            case(select)
                5'b00000: demux_out = 32'b00000000000000000000000000000001;
                5'b00001: demux_out = 32'b00000000000000000000000000000010;
                5'b00010: demux_out = 32'b00000000000000000000000000000100;
                5'b00011: demux_out = 32'b00000000000000000000000000001000;
                5'b00100: demux_out = 32'b00000000000000000000000000010000;
                5'b00101: demux_out = 32'b00000000000000000000000000100000;
                5'b00110: demux_out = 32'b00000000000000000000000001000000;
                5'b00111: demux_out = 32'b00000000000000000000000010000000;
                5'b01000: demux_out = 32'b00000000000000000000000100000000;
                5'b01001: demux_out = 32'b00000000000000000000001000000000;
                5'b01010: demux_out = 32'b00000000000000000000010000000000;
                5'b01011: demux_out = 32'b00000000000000000000100000000000;
                5'b01100: demux_out = 32'b00000000000000000001000000000000;
                5'b01101: demux_out = 32'b00000000000000000010000000000000;
                5'b01110: demux_out = 32'b00000000000000000100000000000000;
                5'b01111: demux_out = 32'b00000000000000001000000000000000;
                5'b10000: demux_out = 32'b00000000000000010000000000000000;
                5'b10001: demux_out = 32'b00000000000000100000000000000000;
                5'b10010: demux_out = 32'b00000000000001000000000000000000;
                5'b10011: demux_out = 32'b00000000000010000000000000000000;
                5'b10100: demux_out = 32'b00000000000100000000000000000000;
                5'b10101: demux_out = 32'b00000000001000000000000000000000;
                5'b10110: demux_out = 32'b00000000010000000000000000000000;
                5'b10111: demux_out = 32'b00000000100000000000000000000000;
                5'b11000: demux_out = 32'b00000001000000000000000000000000;
                5'b11001: demux_out = 32'b00000010000000000000000000000000;
                5'b11010: demux_out = 32'b00000100000000000000000000000000;
                5'b11011: demux_out = 32'b00001000000000000000000000000000;
                5'b11100: demux_out = 32'b00010000000000000000000000000000;
                5'b11101: demux_out = 32'b00100000000000000000000000000000;
                5'b11110: demux_out = 32'b01000000000000000000000000000000;
                5'b11111: demux_out = 32'b10000000000000000000000000000000;
                default: demux_out = 32'b00000000000000000000000000000000;
            endcase
        end
        else begin
            demux_out = 32'b00000000000000000000000000000000;
        end
    end
endmodule

module tb_demux;
    parameter ENABLE_DELAY = 1;
    reg enable;
```

```
    reg [4:0] select;
    reg data_in;

    wire [3:0] demux_out_1to4;
    wire [7:0] demux_out_1to8;
    wire [15:0] demux_out_1to16;
    wire [31:0] demux_out_1to32;

    demux_1to4 uut_1to4 (
        .enable(enable),
        .select(select[1:0]),
        .data_in(data_in),
        .demux_out(demux_out_1to4)
    );
    demux_1to8 uut_1to8 (
        .enable(enable),
        .select(select[2:0]),
        .data_in(data_in),
        .demux_out(demux_out_1to8)
    );
    demux_1to16 uut_1to16 (
        .enable(enable),
        .select(select[3:0]),
        .data_in(data_in),
        .demux_out(demux_out_1to16)
    );
    demux_1to32 uut_1to32 (
        .enable(enable),
        .select(select),
        .data_in(data_in),
        .demux_out(demux_out_1to32)
    );

    initial begin
        // Enable Demultiplexers
        enable = 1'b1;

        // Test 1:4 Demultiplexer
        select = 2'b00;
        data_in = 1'b1;
        #ENABLE_DELAY;
        $display("1:4 Demux Output: %b", demux_out_1to4);
        // Test 1:8 Demultiplexer
        select = 3'b001;
        data_in = 1'b1;
```

```
        #ENABLE_DELAY;
        $display("1:8 Demux Output: %b", demux_out_1to8);
        // Test 1:16 Demultiplexer
        select = 4'b0101;
        data_in = 1'b1;
        #ENABLE_DELAY;
        $display("1:16 Demux Output: %b", demux_out_1to16);
        // Test 1:32 Demultiplexer
        select = 5'b11001;
        data_in = 1'b1;
        #ENABLE_DELAY;
        $display("1:32 Demux Output: %b", demux_out_1to32);
        $finish;
    end
endmodule
```

OUTPUT:

```
[Running] lab7_5.v
1:4 Demux Output: 0001
1:8 Demux Output: 00000010
1:16 Demux Output: 0000000000100000
1:32 Demux Output: 00000010000000000000000000000000
lab7_5.v:205: $finish called at 4 (1s)
[Done] exit with code=0 in 0.265 seconds
```