STARS 2023: Experiment No. 6

Introduction to System Verilog

(Materials are adapted from ECE270 and ECE337 course materials)

Objective:

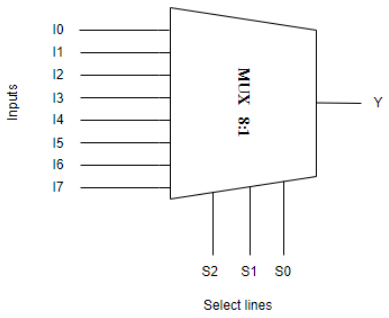
- To learn how to convert a given digital combinational circuit design into System Verilog.
- To learn how to use the System Verilog simulator to synthesize and simulate these designs.
- To learn how to write a design that can drive the output LEDs based on the values asserted via the input pushbuttons of the virtual FPGA breakout board.

Lab Tasks:

1. Implement an 8:1 multiplexer
2. Implement an 8:3 encoder and an 8:3 priority encoder with strobe
3. Implement a 3:8 decoder

Lab task1: 8:1 multiplexer

A multiplexer is a digital switch that uses s select lines to determine which of $n = 2^s$ inputs is connected to its output. The equation implemented by an s-select line multiplexer is the sum-of-products form of a general s-variable function. Figure 1 shows the truth table of an 8:1 multiplexer. It has 8 data inputs I0-I7, 3 select inputs S0, S1, and S2, and produces output Y. Use case statements to implement your circuit using System Verilog.



Select lines			Outputs
S2	S1	S0	Y
0	0	0	I0
0	0	1	I1
0	1	0	I2
0	1	1	I3
1	0	0	I4
1	0	1	I5
1	1	0	I6
1	1	1	I7

Figure 1: 8:1 Multiplexer truth table

Required module name: mux8to1
Required port name:

Port name	Direction
I [7:0]	input
S [2:0]	input
Y	output

On your top module, you may add the following information to map your inputs and outputs to the push buttons and LEDs on the physical hardware.

```
mux8to1 u1(.I(pb[7:0]), .S(pb[10:8]), .Y(red));
```

Note: Examples of behavioral modeling using case statements are provided in the study material. Find a 2:1 multiplexer using case statement below. On your top module, you will want to connect the inputs and outputs of your circuit to the appropriate pushbuttons and LEDs to see the output on the simulator.

```
module mux2to1(input logic [1:0] I, input logic S, output logic Y);

 always_comb

   case (S)

   1'b0: Y= I[0];

   1'b1: Y= I[1];

   default: Y=1'b0;

   endcase

endmodule
```

TA check-off point: Show your 8:1 multiplexer output to your TA for check-off.

Note**:** System Verilog supports conditional ternary operators, by the help of which you can write your 8:1 multiplexer module in an easier way. For example, output Y of a 2:1 multiplexer with data inputs I0, I1, and select input S can be written as-

```
assign Y=S? I0:I1;
```

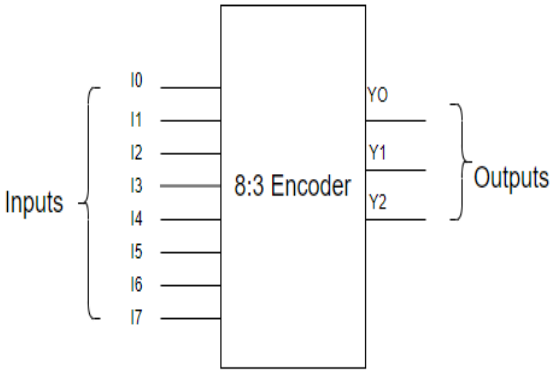
Similarly, the assignment statement for a 4:1 multiplexer using the ternary operator is-

```
assign Y=S[1] ? (S[0] ? I3:I2) : (S[0] ? I1:I0)
```

Try writing the assignment statement for an 8:1 multiplexer and show it to your TA.

Lab task2: 8:3 encoder, 8:3 priority encoder with strobe

An encoder is an inverse decoder, the role of inputs and outputs is reversed, and there are more input code bits than output code bits. The simplest encoder to build is a 2^n-to-n or binary encoder. A basic 8:3 encoder truth table is given in Figure 2 below. In a basic encoder, each input activates the binary encoding that corresponds to its number.



Inputs								Outputs		
I7	I6	I5	I4	I3	I2	I1	I0	Y2	Y1	Y0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Figure 2: An 8:3 encoder truth table

Required module name: encode8to3
Required port name:

Port name	Direction
I [7:0]	input
Y[2:0]	output

On your top module, instantiate your encode8to3 module and add the information to map your inputs and outputs to the push buttons and LEDs on the physical hardware.

The basic encoder does not work if it is possible for more than one input to be asserted simultaneously. Read about the priority encoder in the study materials carefully and implement an 8:3 priority encoder with a strobe signal using System Verilog.

Required module name: pri_enc
Required port name:

Port name	Direction
I [7:0]	input
Y[2:0]	output
G	output

TA check-off point: Show your 8:3 encoder and 8:3 priority encoder with a strobe signal output to your TA for check-off.

Lab task3: Implement a 3-to-8 decoder

The decoder should be constructed so that out[0] signal is asserted if and only if the in[2:0] bus value is 3'b000, the out[1] signal is asserted if and only if the in[2:0] bus value is 3'b001, and so on up to out[7]. Outputs of the decoder module should be active-high. There is no enable line for this decoder. Therefore, exactly one of its outputs is always asserted.

Required module name: decode3to8
Required port name:

Port name	Direction
in[2:0]	input
out[7:0]	output

Create an instance of the 'decode3to8' module in your 'top' module. Use an instance name of your choosing. Make the following connections:

1. Connect pb[2:0] to the instances 'in' port.
2. Connect each of the decimal points on the seven-segment displays (ss7, ... , ss0) to the 'out' port. The decimal point is element 7 of each of the ssx buses. You will need to concatenate element 7 of each bus into a single 8-bit bus connected to the 'out' port.

The result is that if no buttons are pressed, the decimal point of ss0 should be illuminated. If button 0 is pressed, to send a 3'b001 to the decoder input, the decimal point of ss1 should illuminate (and the decimal point of ss0 should turn off). If buttons '2', '1', and '0' are pressed, only the decimal point of ss7 should be illuminated.

TA check-off point: Show your 3:8 decoder output to your TA for check-off.