Laboratory Exercise 1

Switches, Lights, and Multiplexers

The purpose of this exercise is to learn how to connect simple input and output devices to an FPGA chip and implement a circuit that uses these devices. We will use the switches on the FPGA boards as inputs to the circuit. We will use light emitting diodes (LEDs) and 7-segment displays as output devices.

Part I

The FPGA boards provide switches and lights, (e.g., SW_{9-0} and $LEDR_{9-0}$). The switches can be used to provide inputs, and the lights can be used as output devices. Figure 1 shows a simple Verilog module that uses ten switches and shows their states on the LEDs. Since there are multiple switches and lights it is convenient to represent them as vectors in the Verilog code, as shown. We have used a single assignment statement for all *LEDR* outputs, which is equivalent to the individual assignments:

```
. . .
assign LEDR[2] = SW[2];
assign LEDR[1] = SW[1];
assign LEDR[0] = SW[0];
```

The FPGA boards have hardwired connections between its chip and the switches and lights. To use the switches and lights it is necessary to include in your project the correct pin assignments, which are given in your board's user manual. A good way to make the required pin assignments is to import into the software the pin assignment file for your board.

It is important to realize that the pin assignments in the file are useful only if the pin names that appear in this file are exactly the same as the port names used in your Verilog module. For example, if the pin assignment file uses the names *SW*[0], . . ., *SW*[9] and *LEDR*[0], . . ., *LEDR*[9], then these are the names that must be used for input and output ports in the Verilog code, as we have done in Figure 1.

```
// Module that connects ten switches and lights
module  part1 (SW, LEDR);
    input [9:0] SW;          // slide switches
    output [9:0] LEDR;       // red LEDs

    assign LEDR = SW;
endmodule
```

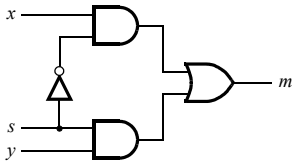
Figure 1: Verilog code that uses ten switches and lights.

Perform the following steps to implement a circuit corresponding to the code in Figure 1 on the FPGA boards.

1. Create a new project for your circuit. Select the target chip that corresponds to your FPGA board.
2. Create a Verilog module for the code in Figure 1 and include it in your project.
3. Include in your project the required pin assignments for your FPGA board, as discussed above. Compile the project.
4. Simulate the circuit through the test bench. Test the functionality of the circuit by toggling the switches and observing the LEDs.

Part II

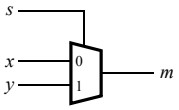
Figure 2a shows a sum-of-products circuit that implements a 2-to-1 *multiplexer* with a select input s. If $s = 0$ the multiplexer's output m is equal to the input x, and if $s = 1$ the output is equal to y. Part b of the figure gives a truth table for this multiplexer, and part c shows its circuit symbol.



a) Circuit

s	m
0	x
1	y

b) Truth table



c) Symbol

Figure 2: A 2-to-1 multiplexer.

The multiplexer can be described by the following Verilog statement:

assign m = (∼s & x) | (s & y);

You are to write a Verilog module that includes four assignment statements like the one shown above to describe the circuit given in Figure 3a. This circuit has two four-bit inputs, X and Y, and produces the four-bit output M. If $s = 0$ then $M = X$, while if $s = 1$ then $M = Y$. We refer to this circuit as a four-bit wide 2-to-1 multiplexer. It has the circuit symbol shown in Figure 3b, in which X, Y, and M are depicted as four-bit wires.

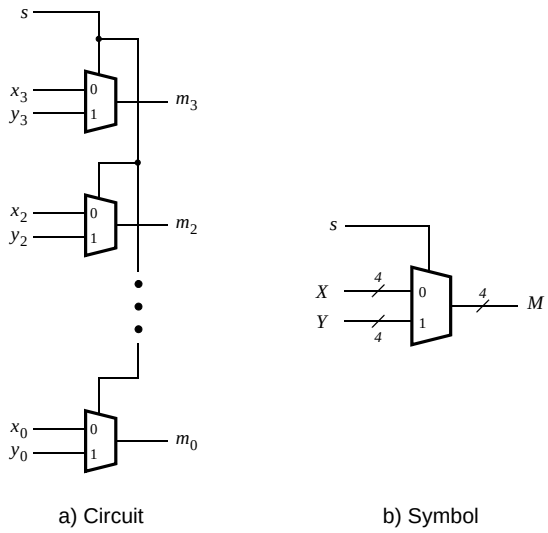


Figure 3: A four-bit wide 2-to-1 multiplexer.

Perform the steps listed below.

1. Create a new project for your circuit.
2. Include your Verilog file for the four-bit wide 2-to-1 multiplexer in your project. Use switch SW_9 as the s input, switches SW_{3-0} as the X input and SW_{7-4} as the Y input. Display the value of the input s on $LEDR_9$, connect the output M to $LEDR_{3-0}$, and connect the unused LEDR lights to the constant value 0.
3. Include in your project the required pin assignments for your FPGA board. As discussed in Part I, these assignments ensure that the ports of your Verilog code will use the pins on the FPGA chip that are connected to the *SW* switches and *LEDR* lights.
4. Simulate the circuit through the test bench. Test the functionality of the four-bit wide 2-to-1 multiplexer by toggling the switches and observing the LEDs.

Part III

In Figure 2 we showed a 2-to-1 multiplexer that selects between the two inputs x and y. For this part consider a circuit in which the output m has to be selected from four inputs u, v, w, and x. Part a of Figure 4 shows how we can build the required 4-to-1 multiplexer by using three 2-to-1 multiplexers. The circuit uses a 2-bit select input $s_1 s_0$ and implements the truth table shown in Figure 4b. A circuit symbol for this multiplexer is given in part c of the figure.

Recall from Figure 3 that a four-bit wide 2-to-1 multiplexer can be built by using four instances of a 2-to-1 multiplexer. Figure 5 applies this concept to define a two-bit wide 4-to-1 multiplexer. It contains two instances of the circuit in Figure 4a.

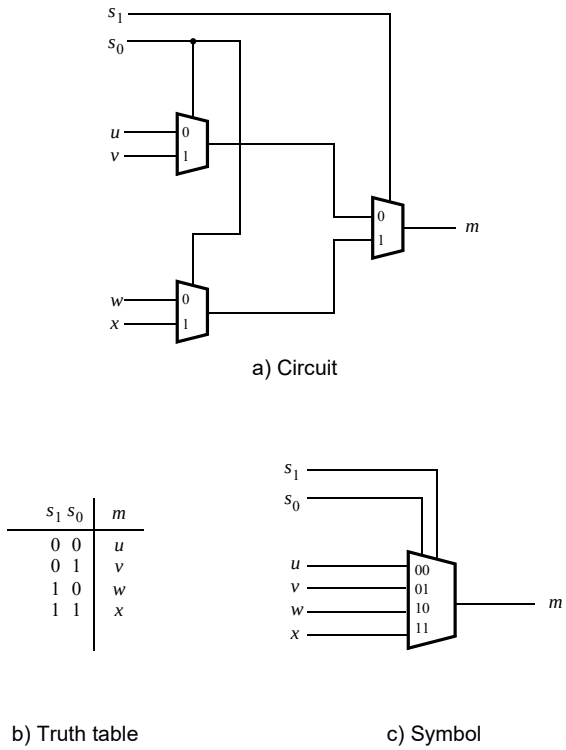


Figure 4: A 4-to-1 multiplexer.

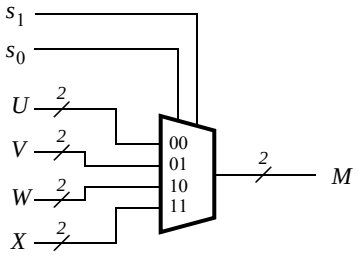


Figure 5: A two-bit wide 4-to-1 multiplexer.

Perform the following steps to implement the two-bit wide 4-to-1 multiplexer.

1. Create a new project for your circuit.
2. Create a Verilog module for the two-bit wide 4-to-1 multiplexer. Connect its select inputs to switches SW_{9-8}, and use switches SW_{7-0} to provide the four 2-bit inputs U to X. Connect the output M to the red lights $LEDR_{1-0}$.
3. Include in your project the required pin assignments for your FPGA board. Compile the project.
4. Simulate the circuit through the test bench. Test the functionality of the two-bit wide 4-to-1 multiplexer by toggling the switches and observing the LEDs. Ensure that each of the inputs U to X can be properly selected as the output M.

Part IV

The objective of this part is to display a character on a 7-segment display. The specific character displayed depends on a two-bit input. Figure 6 shows a *7-segment decoder* module that has the two-bit input c_1c_0. This decoder produces seven outputs that are used to display a character on a 7-segment display. Table 1 lists the characters that should be displayed for each valuation of c_1c_0 for your FPGA board. Note that in some cases the 'blank' character is selected for code 11.

The seven segments in the display are identified by the indices 0 to 6 shown in the figure. Each segment is illuminated by driving it to the logic value 0. You are to write a Verilog module that implements logic functions to activate each of the seven segments. Use only simple Verilog **assign** statements in your code to specify each logic function using a Boolean expression.

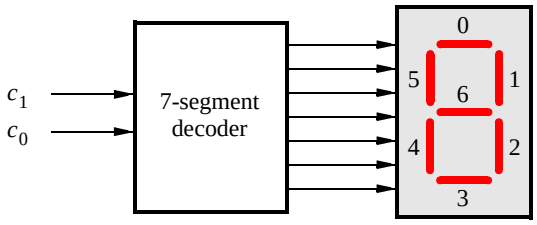


Figure 6: A 7-segment decoder.

c_1c_0	Character
00	d
01	E
10	1
11	0

Table 1: Character codes for the FPGA boards.

Perform the following steps:

1. Create a new project for your circuit.
2. Create a Verilog module for the 7-segment decoder. Connect the c_1c_0 inputs to switches SW_{1-0}, and connect the outputs of the decoder to the *HEX0* display on your FPGA board. The segments in this display are called $HEX0_0$, $HEX0_1$, ..., $HEX0_6$, corresponding to Figure 6. You should declare the 7-bit port

   output [0:6] HEX0;

   in your Verilog code so that the names of these outputs match the corresponding names in your board's user manual and pin assignment file.
3. After making the required pin assignments, compile the project.
4. Download the compiled circuit into the FPGA chip. Test the functionality of the circuit by toggling the SW_{1-0} switches and observing the 7-segment display.

Part V

Consider the circuit shown in Figure 7. It uses a two-bit wide 4-to-1 multiplexer to enable the selection of four characters that are displayed on a 7-segment display. Using the 7-segment decoder from Part IV this circuit can display the characters d, E, 0, 1, 2, or 'blank' depending on your FPGA board. The character codes are set according to Table 1 by using the switches SW_{7-0}, and a specific character is selected for display by setting the switches SW_{9-8}.

An outline of the Verilog code that represents this circuit is provided in Figure 8. Note that we have used the circuits from Parts III and IV as subcircuits in this code. You are to extend the code in Figure 8 so that it uses four 7-segment displays rather than just one. You will need to use four instances of each of the subcircuits. The purpose of your circuit is to display any word on the four 7-segment displays that is composed of the characters in Table 1, and be able to rotate this word in a circular fashion across the displays when the switches SW_{9-8} are toggled. As an example, if the displayed word is dE10, then your circuit should produce the output patterns illustrated in Table 2.

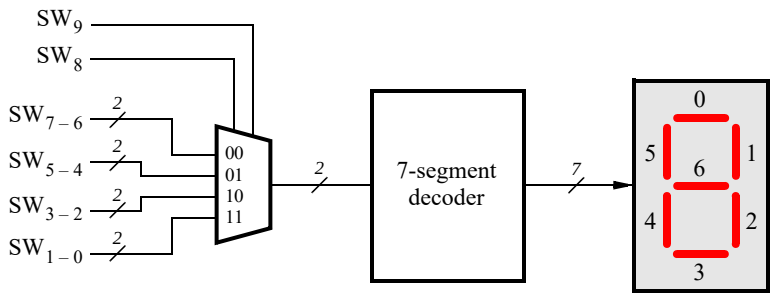


Figure 7: A circuit that can select and display one of four characters.

SW_{9-8}	Characters			
00	d	E	1	0
01	E	1	0	d
10	1	0	d	E
11	0	d	E	1

Table 2: Rotating the word dE10 on four displays.

Perform the following steps.

1. Create a new project for your circuit.
2. Include your Verilog module in the project. Connect the switches SW_{9-8} to the select inputs of each of the four instances of the two-bit wide 4-to-1 multiplexers. Also connect SW_{7-0} to each instance of the multiplexers as required to produce the patterns of characters shown in Table 1. Connect the SW switches to the red lights LEDR, and connect the outputs of the four multiplexers to the 7-segment displays *HEX3*, *HEX2*, *HEX1*, and *HEX0*.
3. Include the required pin assignments for your FPGA board for all switches, LEDs, and 7-segment displays. Compile the project.
4. Download the compiled circuit into the FPGA chip. Test the functionality of the circuit by setting the proper character codes on the switches SW_{7-0} and then toggling SW_{9-8} to observe the rotation of the characters.

```
module  part5 (SW, LEDR, HEX0);
    input [9:0] SW;                          // slide switches
    output [9:0] LEDR;                       // red lights
    output [0:6] HEX0;                       // 7-seg display

    wire [1:0] M0;

    mux_2bit_4to1 U0 (SW[9:8], SW[7:6], SW[5:4], SW[3:2], SW[1:0], M0);
    char_7seg H0 (M0, HEX0);
    . . .
endmodule

// implements a 2-bit wide 4-to-1 multiplexer
module mux_2bit_3to1 (S, U, V, W, X, M);
    input [1:0] S, U, V, W, X;
    output [1:0] M;
    . . . code not shown

endmodule

// implements a 7-segment decoder for d, E, 1 and 0
module char_7seg (C, Display);
    input [1:0] C;              // input code
    output [0:6] Display;    // output 7-seg code
    . . . code not shown

endmodule
```

Figure 8: Verilog code for the circuit in Figure 7.

.