Homework Exercise 1

Switches, Lights, and Multiplexers

TCES 330
Spring 2018

Introduction

The purpose of this exercise is to gain more experience with ModelSim and Quartus. You will also 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 DE-series boards as inputs to the circuit. We will use light emitting diodes (LEDs) and 7-segment displays as output devices.

It is important that you realize from the beginning that you must allocate plenty of time to finish these exercises by the due date. There are six parts to this homework; five of them will involve testbenches and ModelSim simulations. Read the directions carefully.

Before you begin this exercise, set up a directory structure that looks exactly like

Figure 1. Make sure every file required for your circuits is somewhere in your HW1 directory tree. Because I will completely recompile your project and anything missing or that cannot be found will result in a compile error. I stop checking your work when I find such an error.

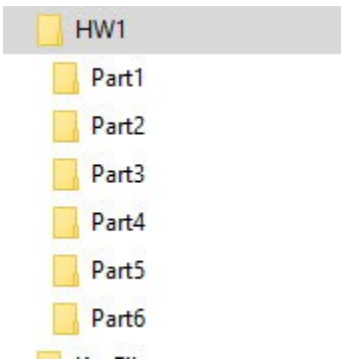


Figure 1. HW1 Directories.

When you turn in your assignment on Canvas, you will Zip up HW1 and all those sub-directories and turn in the complete package.

Part 1

The DE2-115 provides eighteen switches and lights. The switches can be used to provide inputs, and the lights can be used as output devices. Figure1shows 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 DE-series boards have hardwired connections between its FPGA chip and the switches and lights. To use the switches and lights it is necessary to include in your Quartus 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 Quartus software the pin assignment file for your board, which is provided for you on Canvas in the Files section for this course. The procedure for making pin assignments has been described in class and is described in the tutorial *Quartus Introduction using Verilog Design*, which is also available from Intel.

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] to SW[17] and LEDR[0] to LEDR[17], then these are the names that must be used for input and output ports in the Verilog code, as we have done in Figure 2.

```
// Module that connects ten switches and lights
module part1 (SW, LEDR);
input [17:0] SW;    // slide switches
output [17:0] LEDR; // red LEDs
assign LEDR= SW;
endmodule
```

Figure 2. Verilog code that uses switches and lights.

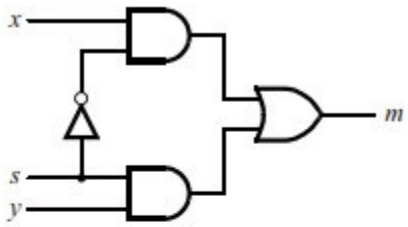
Note: since this module interfaces with the DE2-115 board directly, it is a **top-level** module. As such we will not need a testbench for this module since it can be tested using the DE2 board.

Perform the following steps to implement a circuit corresponding to the code in Figure1on the DE-series boards.

1. Create a new Quartus project for your circuit. Select the target chip that corresponds to your DE-series board. Refer to Table1for a list of devices.
2. Create a Verilog module for the code in Figure 2 and include it in your project.
3. Include in your project the required pin assignments for your DE-series board, as discussed above. Compile the project.
4. Download the compiled circuit into the FPGA chip by using the Quartus Programmer tool (the procedure for using the Programmer tool is described in the tutorial Quartus Introduction). Test the functionality of the circuit by toggling the switches and observing the LEDs.

Part 2

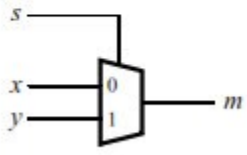
Figure 3a shows a sum-of-products circuit that implements a 2-to-1 multiplexer with a select input s. If s = 0 the multiplexer's output m is equal to the input x, and if s = 1 the output is equal to y. Part b of the figure gives a truth table for this multiplexer, and part c shows its circuit symbol.



a) Circuit

s	m
0	x
1	y

b) Truth table



c) Symbol

Figure 3. A 2-to-1 multiplexer.

The multiplexer can be described by the following SystemVerilog statement:

```
assign m = (~S & X) | (S & Y);  // a 2-to-1 multiplexer
```

You are to write a SystemVerilog module that includes four assignment statements like the one shown above to describe the circuit given in Figure 4a. 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 4b, in which X, Y, and M are depicted as four-bit wires. Call this module Mux_4w_2_to_1() and the file you write it in Mux_4w_2_to_1.sv.

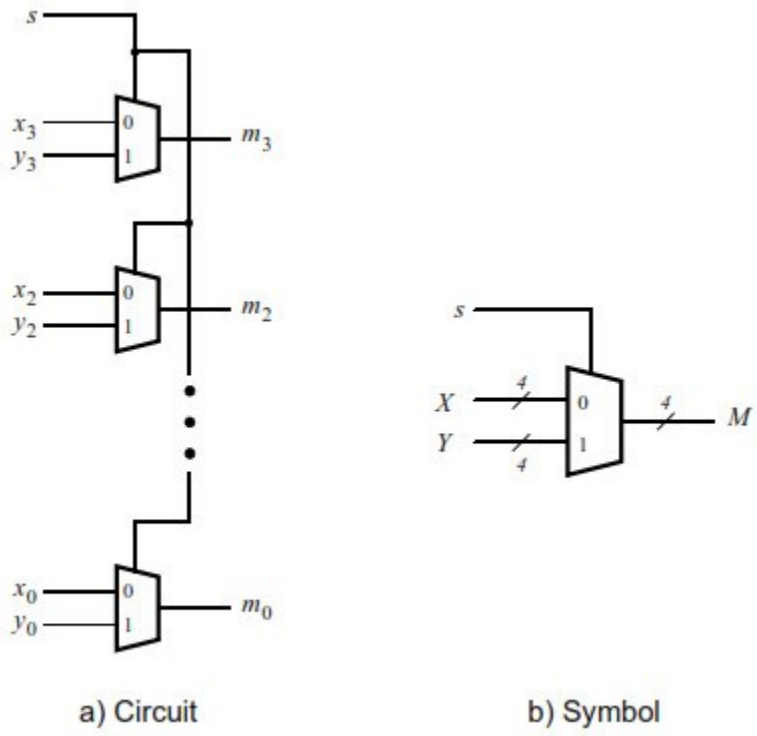


Figure 4. A four bit wide 2-to-1 multiplexer.

Perform the steps listed below.

1. Place your Mux_4w_2_to_1.sv file in your Part2 folder.

2. In your Mux_4w_2_to_1 file, write a testbench for your circuit similar to what we discussed in class. Pick some values for X and some different values for Y and toggle S from 0 to 1 for each set of values.
3. Copy the runlab.do, wave.do, and Launch_ModelSim.bat files into your Part2 folder. Modify them as we discussed in class.
4. Run ModelSim (using Launch ModelSim.bat) and the simulation for your circuit (using do runlab.do). Correct any errors you may find. Repeat until you get a successful simulation run.
5. Create a new Quartus project for your circuit (in your Part2 folder).
6. Create a new SystemVerilog module called Part2 that instantiates your Mux_4w_2_to_1 module and connects all the inputs and outputs to the DE2-115 board as follows.
    a. S: switch SW_{17}
    b. X: switches SW_{3-0}
    c. Y: switches SW_{7-4}
    d. M: LEDs $LEDG_{3-0}$.
    e. Connect all of the LEDR lights to the SW switches like we did in Part 1.
7. Include in your project the required pin assignments for your DE2-115 board. As discussed in Part 1, 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 the LEDR lights.
8. Compile the project and then download the resulting circuit into the FPGA chip. Test the functionality of your project by toggling the switches and observing the LEDs in a fashion similar to your ModelSim simulation.
9. When you are finished, your Part2 folder should contain the files shown in Figure 5.
10. The runlab.do you turn in should run a testbench for your Mux_4w_2_to_1 module. It should contain a line that calls the appropriate *wave.do for this module.

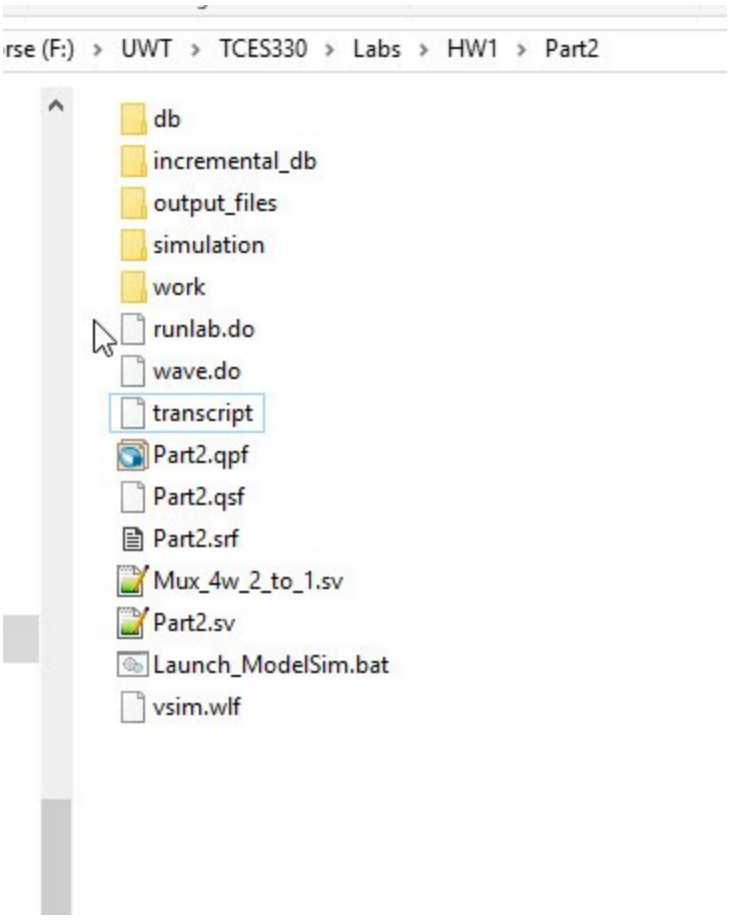


Figure 5. Par2 directory contents.

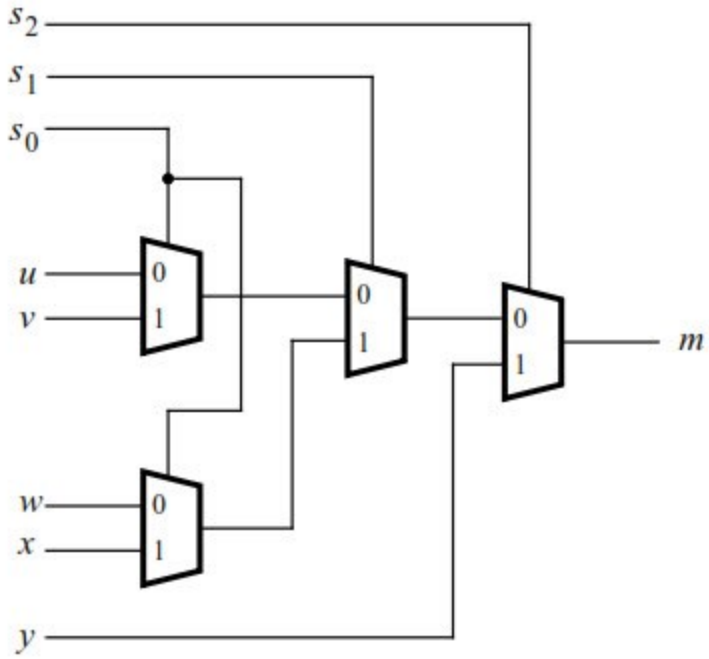
Note that whenever you need a 4-bit wide 2-to-1 multiplexer, you can now simply and safely use your Mux_4w_2_to_1 module.

Part 3

In Figure 3 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 five inputs u, v, w, x, and y. Part a of Figure 6 shows how we can build the required 5-to-1 multiplexer by using four 2-to-1 multiplexers. The circuit uses a 3-bit select input s and implements the truth table shown in Figure 6 b. A circuit symbol for this multiplexer is given in part c of the figure.

Recall from Figure 4 that a four-bit wide 2-to-1 multiplexer can be built by using four instances of a 2-to-1 multiplexer. Figure 7 applies this concept to define a

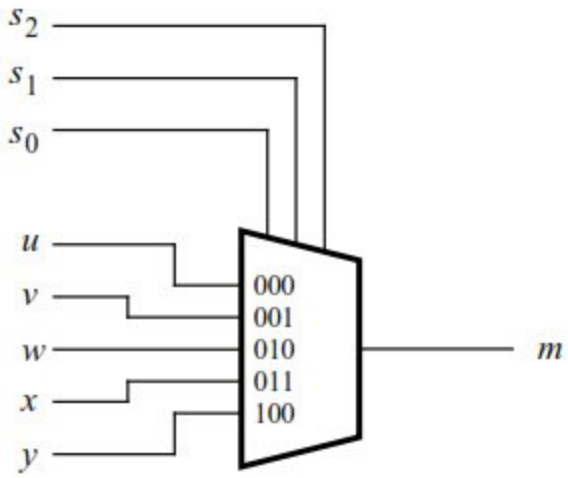
three-bit wide 5-to-1 multiplexer. It contains three instances of the circuit in Figure 6 a.



a) Circuit

s_2	s_1	s_0	m
0	0	0	u
0	0	1	v
0	1	0	w
0	1	1	x
1	0	0	y
1	0	1	y
1	1	0	y
1	1	1	y

b) Truth table



c) Symbol

Figure 6. A 5-to-1 multiplexer.

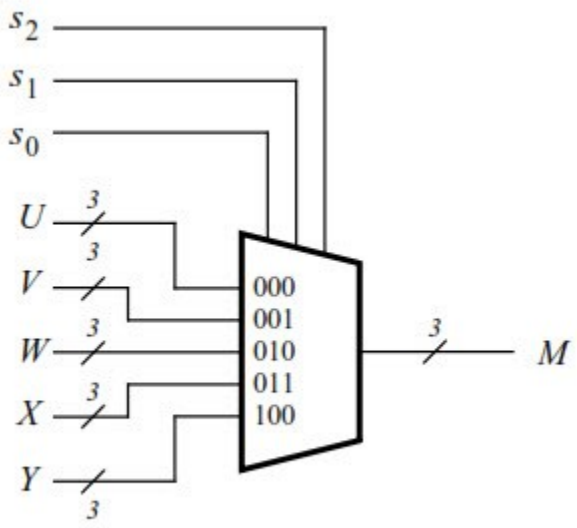


Figure 7. A three-bit wide 5-to-1 multiplexer.

Perform the following steps to implement the three-bit wide 5-to-1 multiplexer.

1. Using the Quartus or some other text editor, make a file called Mux_5_to_1.sv and store it in your Part3 folder. Enter your code for the Mux_5_to_1 multiplexer (Figure 6) in this file.
2. Write a testbench for your Mux_5_to_1 module in the same file. Call this Mux_5_to_1_testbench. Ensure that each of the inputs U through Y can be properly selected as the output, M, using an appropriate value of S.
3. Copy Launch_ModelSim.bat and runlog.do into your Part3 folder. Since we are going to have a couple of testbenches in this folder, we will have to do some inventive naming. Make a copy of runlab.do and rename it runmux5.do. Modify runmux5.do to run the testbench for Mux_5_to_1 as follows.
    a. Make sure you have a 'vlog' line for each *.sv module you are using in this testbench.
    b. Change the end of the 'vsim' line to read `Mux_5_to_1_testbench`.
    c. Change the do *_wave.do line to read
    `do Mux_5_to_1_wave.do`
4. Launch ModelSim using the .bat file. Run rumux5.do and fix any problems. You will get at least one error saying it can't find Mux_5_to_1_wave.do. You will have to drag some variables into the Wave Window and save the result

to a Mux_5_to_1_wave.do file just like we did in class. Run runmux5.do and fix errors until you obtain a successful simulation.

5. Using the Quartus or some other text editor, make a file called Mux_3w_5_to_1.sv and store it in your Part3 folder.
6. Enter your code for the multiplexer required (Figure 7). This module should be called Mux_3w_5_to_1().
7. Write a testbench for your Mux_3w_5_to_1 module in the same file. Ensure that each of the inputs U through Y can be properly selected as the output, M, using an appropriate value of S.
8. Modify runlab.do to run the testbench for Mux_3w_5_to_1 as follows.
    a. Make sure you have a 'vlog' line for each *.sv module you are using in this testbench (at least Mux_5_to_1.sv and Mux_3w_5_to_1.sv).
    b. Change the end of the 'vsim' line to read `Mux_3w_5_to_1_testbench`.
    c. Change the do *_wave.do line to read
    `do Mux_3w_5_to_1_wave.do`
9. Launch ModelSim using the .bat file and fix any problems. You will get at least one error saying it can't find Mux_3w_5_to_1_wave.do. You will have to drag some variables into the Wave Window and save the result to a Mux_3w_5_to_1_wave.do file just like we did above. Run runlab.do and fix errors until you obtain a successful simulation.
10. In your Part3 folder create a new Quartus project for your circuit.
11. Create a top-level SystemVerilog module called Part3 that connects your Mux_3w_5_to_1 to switches on the DE2-115 board as follows:
    a. S2: SW_{17}
    b. S1: SW_{16}
    c. S1: SW_{15}
    d. U: $SW_{14\text{-}12}$
    e. V: $SW_{11\text{-}9}$
    f. W: $SW_{8\text{-}6}$
    g. X: $SW_{5\text{-}3}$
    h. Y: $SW_{2\text{-}0}$
    i. M: $LEDG_{2\text{-}0}$
    j. Hook up all switches SW to the red LEDRs like we did in Part1.
12. Include in your project the required pin assignments for your DE2-115 board. Compile the project.
13. Download the compiled circuit to the FPGA chip. Test the functionality of Mux_3w_5_to_1 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.
14. When you are done, your Part3 folder should look something like Figure 8.
15. The runlab.do you turn in should run a testbench for your Mux_3w_5_to_1 module. It should contain a line that calls the appropriate *wave.do for this module.

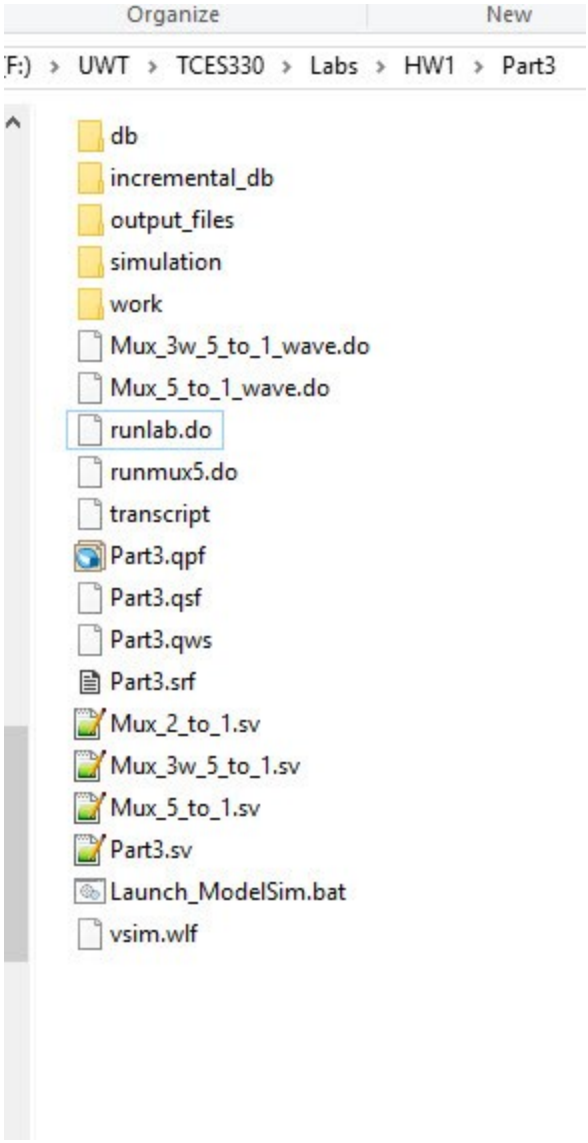


Figure 8. Part3 directory contents.

Part 4

Figure 6 shows a 7-segment decoder module that has the three-bit input c_1 c_2 c_3 . This decoder produces seven outputs that are used to display a character on a 7-segment display. Figure 9 lists the characters that should be displayed for each valuation of c_1 c_2 c_3. To keep the design simple, only four characters are

included in the table (plus the 'blank' character, which is selected for codes 100 – 111). The seven segments in the display are identified by the indices 0 to 6 shown in Figure 10.

Each segment is illuminated by driving it to the logic value 0. You are to write a SystemVerilog module that implements logic functions that represent circuits needed to activate each of the seven segments. You should make a truth table similar to Table 1,. You can then use simple Verilog assign statements in your code to specify each logic function (the output columns) using a Boolean expression.

$$Seg0 = f_0(c2, c1, c0)$$
$$Seg1 = f_1(c2, c1, c0)$$
$$\ldots$$

You do not have to simplify these expressions (using K-maps, etc.) since the Quartus synthesizer will do that job for you. The most likely mistake you'll make here is to forget that a logic 0 turns a segment ON and a logic 1 turns it OFF.

$c_2 c_1 c_0$	Character
000	H
001	E
010	L
011	O
100	
101	
110	
111	

Figure 9. Character codes.

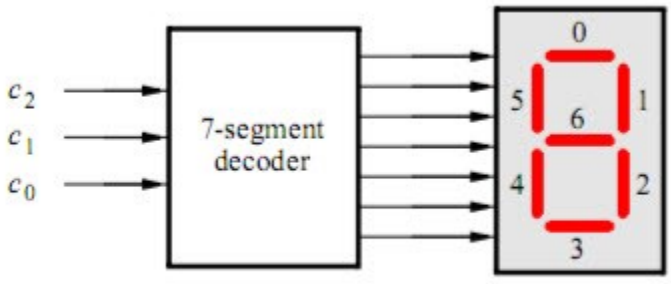


Figure 10. A 7-segment decoder.

c2	c1	c0	Seg0	Seg1	…	Seg6	Char
0	0	0	0	1		1	H
0	0	1					E
0	1	0					L
0	1	1					O
1	0	0					_
etc.							_

Table 1. Truth table for HELO.

Perform the following steps.

1. In your Part4 folder make a file called HexHELO.sh and in this file place your seven segment decoder logic. The module could use a signature something like
   ```
   module HexHELO( s0, s1, s2, s3, s4, s5, s6, c0, c1, c2 );
   ```
   But why write all these variables? We have already met the vector, and vectors were made for situations like this. So, instead, let's use the much simpler signature
   ```
   module HexHELO( Hex, C );
   input  [2:0] C;   // input lines
   output [0:6]Hex;  // the seven segments
   ```
2. We won't make a testbench for Part4 since this is more efficient to test directly on the DE2 board. So make a Quartus project called Part4 and a top-level module that connects your decoder inputs and outputs to the board as follows
   a. C[2]: switch SW_{17}
   b. C[1]: switch SW_{16}
   c. C[0]: switch SW_{15}
   d. S_0 through S_6: HEX0[0] through HEX0[6]
      You should declare the 7-bit port
      ```
      output [0:6] HEX0;
      ```
      to match the pin assignments in the DE2_115_pin_assignments.csv file.
3. After making the required DE2-115 board pin assignments, compile the project. Correct errors and serious warnings.

4. Download the compiled circuit to the FPGA chip. Test the functionality of your circuit by toggling the switches SW17-15 through all possible values and comparing HEX0 to Figure 9. Correct any errors.
5. No ModelSim files are required for this part.

Part 5

Consider the circuit shown in x, where the variables H = 3'b0, E = 3'b1, and so on according to Figure 9. It uses a three-bit wide 5-to-1 multiplexer to enable the selection of one of five characters that are displayed on a 7-segment display. Using the 7-segment decoder from Part 4 this circuit can display any of the characters H, E, L, L, or O simply by changing SW_{17-15}. When SW_{17-15} = 3'b0, 'H' is displayed, when SW_{17-15} = 3'b1 'E' is displayed, etc.

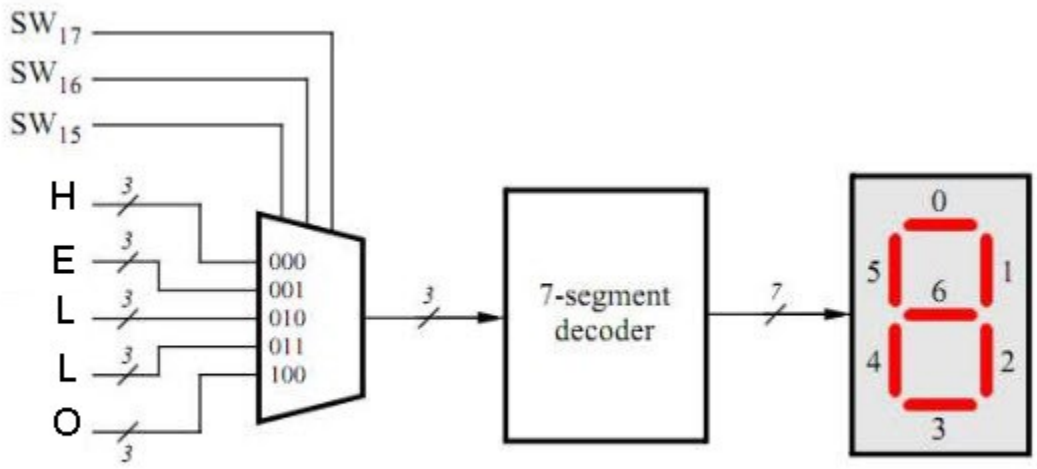


Figure 11. A circuit that can select and display one of five characters.

SW17-15	HEX4	HEX3	HEX2	HEX1	HEX0
000	H	E	L	L	O
001	E	L	L	O	H
010	L	L	O	H	E
011	L	O	H	E	L
100	O	H	E	L	L

Table 2. Rotating the word HELLO on five displays.

Note we have already built the 3-bit 5-to-1 multiplexer in Part 3 and the 7-segment decoder in Part 4. This part, Part 5, simply uses combinations of these circuits that we already have built. Your circuit will repeat the basic circuit of Figure 11 five times, each copy with different inputs to the mux to form the display patterns of Table 2.

Perform the following steps.

1. Create a new Quartus project called Part5 in your Part5 folder.
2. Create a SystemVerilog file called Part5.sv containing a module called Part5().
3. Write your code for Part5 by instantiating copies of your Mux_3w_5_to_1 module and your HexHELO module and wiring them together appropriately.
4. Use switches to form the variables H, E, L, and O.
   H: SW_{14-12}
   E: SW_{11-9}
   L: SW_{8-6}
   O: SW_{5-3}
   Note: you only need to set the switches for 'L' **ONCE**; you can use it in multiple places. And as discussed previously, use SW_{17-15} to change the display (Table 2).
5. Include the required pin assignments for the DE2-115 board for all switches, LEDs, and 7-segment displays. Compile the project.
6. Download the compiled circuit to the FPGA chip. Test the functionality of the circuit by setting the proper character code switches (see 4), above) and toggling SW_{17-15} to observe the rotation of the characters.
7. No ModelSim files are required for this part.

Part 6

Extend your design from Part 5 so that it uses all eight 7-segment displays on the DE2-115 board. Your circuit should be able to rotate the displayed word when the switches SW17-15 are toggled as shown in Table 3. And, yes, you will need to build a three-wide eight to one multiplexor module to implement this circuit.

SW_{17} SW_{16} SW_{15}	Character pattern							
000				H	E	L	L	O
001			H	E	L	L	O	
010		H	E	L	L	O		
011	H	E	L	L	O			
100	E	L	L	O				H
101	L	L	O				H	E
110	L	O				H	E	L
111	O				H	E	L	L

Table 3. Rotating the word HELLO on eight displays.

Perform the following steps.

1. Since you are controlling eight independent displays in this part, you will need to a Mux_3w_8_to_1 circuit. Design one of these along with its testbench in a file called Mux_3w_8_to_1.sv. Test it similar to the way you tested your Mux_3w_5_to_1 mux.
2. Create a new Quartus project called Part6 in your Part6 folder.
3. Create a SystemVerilog file called Part6.sv containing a module called Part6().
4. Write your code for Part6 by instantiating copies of your Mux_3w_8_to_1 module and your HexHELO module and wiring them together appropriately.
5. Use switches to form the variables H, E, L, and O.
   H: SW_{14-12}
   E: SW_{11-9}
   L: SW_{8-6}
   O: SW_{5-3}
   blank: SW_{2-0}
   Note: you only need to set the switches for 'L' **ONCE**; you can use it in multiple places. And as discussed previously, use SW_{17-15} to change the display (Table 3).
6. Include the required pin assignments for the DE2-115 board for all switches, LEDs, and 7-segment displays. Compile the project.
7. Download the compiled circuit to the FPGA chip. Test the functionality of the circuit by setting the proper character code switches (see 4), above) and toggling SW_{17-15} to observe the rotation of the characters.

8. The runlab.do you turn in should run a testbench for three wide eight to one mux module. It should contain a line that calls the appropriate *wave.do for this module.

When you are finished, zip up your entire HW1 directory and submit the zip file on Canvas.