CSCB58 - Lab 2

Multiplexers, Design Hierarchy, and HEX Displays

1   Learning Objectives

The purpose of this lab exercise is to use gates to build simple circuits, and to learn the importance of simulations and hierarchies. This exercise is also meant to illustrate the importance of Karnaugh maps in designing circuits, and introduce important devices such as the multiplexer and seven-segment decoder.

This lab also introduces components of the DE1-SoC board and FPGA design in general. The DE1-SOC board is the hardware used in the physical labs to which we upload circuit designs. For this lab we will simulate the use of switches SW_{9-0} on the DE1-SoC board as inputs to the circuit, and Light Emitting Diodes (LEDs) and 7-segment displays as output devices.

This lab is also meant to introduce new notation and terminology when we mimic physical hardware. For instance, we may refer to signals on the input switches on the DE1-SoC as SW_{9-0}, *i.e.*, with the subscripts. In place of these physical switches, we will approximate these inputs and outputs in Logisim instead.

*** Note that we will not be using the actual DE1-SOC board in Winter 2021.

2   Resources

You can find many resources about the DE1-SoC board here `http://cd-DE1-soc.terasic.com/`. The User Manual for the DE1-SoC board can be downloaded from here: `http://www-ug.eecg.toronto.edu/desl/manuals/DE1-SoC_User_manual.pdf`

3   Marking Scheme

This lab is worth 4% of your final grade, but you will be graded out of 8 marks for this lab, as follows:

- Prelab + Testing: 3 marks
- Part I (in-lab): 1 mark
- Part II (in-lab): 2 marks
- Part III (in-lab): 2 marks

4   Preparation Before the Lab

For this lab and all future labs, your prelab report should include schematics (which can either be drawn by hand on paper or in Logisim), and simulations of your circuits (like the test vector file in Lab 1 and the test result).

The schematics should show the structure of your circuit, much like the schematics in Lab 1 showed how your circuit should be built. Figure 1 and Figure 3 are examples of how to draw the schematic for multiplexers. For the schematics you will need to show the names of each input and output (in your Logisim circuits) and the switches or LEDs to which they would be mapped on the DE1-SoC board (for the top-level module).

Your circuit design in Logisim will consist of a number of **modules**, where a module is like the circuitry equivalent of a class in Java or Python, encapsulating and implementing a circuit instead of code. Think of **modules** as *logic functions* consisting of multiple gates, such as the logic functions in Lab 1.

Modules can contain other modules. The top-level module would be the one that is uploaded to the DE1-SOC board. In addition to showing how its component modules are wired together, the top-level module would also have inputs and outputs that correspond to signals on the DE1-SOC board, *i.e.*, connections to switches, LEDs, 7-segment hex displays, etc. The port names of all modules, wires and I/O ports should be clearly labeled in your schematics.

For your simulations, you should have a test vector file or files, that tests important aspects of your design. Print out the test result on Logisim and include that into your prelab report (*e.g.* by taking a screenshot).

If the simulation is very long, make sure to print out the salient parts to demonstrate that the key elements of your circuit are working and that you have done the simulations. It is not necessary to have pages and pages of results. However, occasionally, you will be asked to demonstrate and explain any part of simulation to the TA in the lab, so be prepared for this.

As an example, if your circuit implements a logic function with three or four inputs, it is reasonable to show waveforms demonstrating the functionality of all possible combinations of input values. However, if your circuit implements a logic function with ten inputs, it would be unreasonable to simulate all 2^{10} possible input values. Instead, use similar approaches to testing that you would use for software; find groups of inputs that are logically related and test each group independently. For instance, if you have a multiplexer module and an adder module, it is reasonable to assume that you can test each independently instead of testing all combinations of all their inputs combined.

5   Part I

This part of the lab expands slightly on the mux circuit from Lab 1, introducing the concept of creating a module and associating the input and outputs to the physical hardware of the physical DE1-SOC board.

Logisim File (.circ)

The DE1-SoC board in the on-campus labs provide 10 toggle switches, called SW_{9-0} that can be used as inputs to a circuit, and 10 red lights, called $LEDR_{9-0}$, that can be set high or low to display output values. These subscript ranges indicate the names of the individual inputs and outputs. For instance, your circuit could take signal SW[0] as an input and send a result to LEDR[0] and LEDR[1]. As far as your Logisim design is concerned, we will use the same input and output pins from the tool bar that you used in Lab 1. You don't have to submit any **(PRELAB)**for Part I.

If we were using the DE1-SOC boards in the on-campus lab rooms, we would need to map the inputs and outputs for the top-level Logisim module to the SW_{9-0} and $LEDR_{9-0}$ pins on the DE1-SOC board respectively before uploading your design to the board. Unfortunately, at this point we do not have access to the boards in the on-campus labs, but we will still use the DE1-SOC ideas and terminology to approximate the way your design would map to the physical board. A Logisim file for a 2-to-1 multiplexer, named *mux.circ*, has already been provided to you on Quercus.

After you open this mux file in Logisim, the main is a very trivial example of a design hierarchy, as it only instantiates a single *mux2to1* module. In the more general case, a single module can instantiate a number of interconnected modules. However, in any circuit you build, there must be only one *top-level* module (called `main` in Logisim). The names of the left side of the rectangular module symbol match the input ports of the *mux2to1* module and on the right side, it is the output port of the *mux2to1* module. Note that you can have multiple modules of the same type in a single circuit, similar to how several objects of the same type can be used together in the implementation of a larger object. And similar to object-oriented programming, each use of a module is called an *instance* and every instance in a single circuit must have a unique label. In our example below we used only one instance of the mux2to1 module, which we named *u*. There are two modules, main and mux2to1 in mux.circ, you can check this in *Components*. Double-clicking on either of them to view the module on canvas:

Currently *main* has 3 inputs, corresponding to the three multiplexer inputs. The input labeled x on
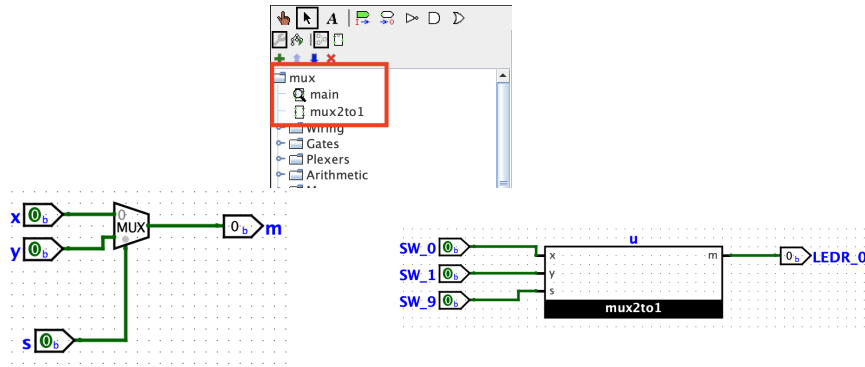
the multiplexer would be connected to an input called SW_0 in Logisim (which maps this to $SW[0]$ on DE1-SOC); the input labeled y on the multiplexer would be connected to the input SW_1 in Logisim (which maps to $SW[1]$ on DE1-SOC) and the input signal labeled s would be connected to the input SW_9 in Logisim (map this to $SW[9]$ on DE1-SOC). The output is the output $LEDR_0$ in Logisim, which would map to the $LEDR_0$ on the DE1-SOC board.

Figure 1 shows the symbol for a 2-to-1 multiplexer. As mentioned in Lab 1, a multiplexer is a device that uses a select signal to select which one of multiple inputs should appear on the output of the device. In Figure 1, input s will control which of the inputs x and y will appear on the output m. If s is 0, x will appear on the output, while if s is 1, y will appear on the output.
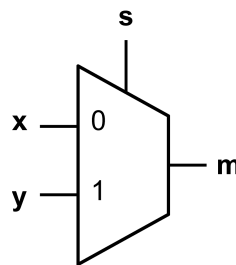

Figure 1: Symbol for a 2-to-1 multiplexer

The Boolean expression for a 2-to-1 multiplexer is m = xs'+ ys.

Test Vector File (.txt)

After building the circuit in Logisim, and to verify the circuit's functionality properly, we can perform testing using a test vector file like in Lab1. This file is also provided to you (on Quercus).

This test file includes the truth table for the 2to1 mux. And you should be able to run the tests through *Simulate > Test Vector...*

```
# Test Vector for mux2to1
SW_0  SW_1  SW_9  LEDR_0
0  0  0  0
0  1  0  0
1  0  0  1
1  1  0  1
0  0  1  0
0  1  1  1
1  0  1  0
1  1  1  1
```

3

6 Part II

Start with the design given in Part I and modify the design to make it a 4-to-1 multiplexer. You must use multiple instantiations of the *mux2to1* module given to you in Part I. This is known as *hierarchical design* and is a good practice especially for larger designs where the circuits you build can become more difficult to debug. Smaller submodules are generally easier to test thoroughly and debug.

Create a new circuit by clicking on the green plus sign just above the components. Enter "mux4to1" for "Circuit Name". Then, simply click on the *mux2to1* module and drop it on the canvas. You can then connect the input and output of the instantiated model with input and output of your current module with wires. **Note: All inputs and outputs of the module have to be the default type from the tool bar.** But you can do some experiments yourselves to see what could happen when we instantiate a module with other types of input output in *Input/Output*.

Figure 2 shows a schematic of two modules connected with wires.
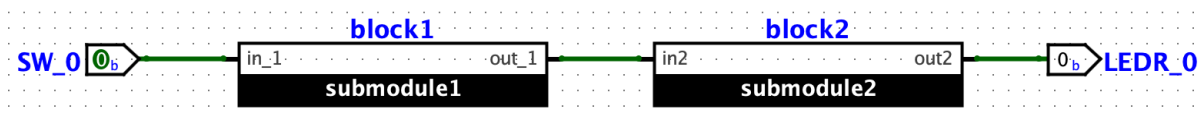

Figure 2: Using wires to make a connection between two modules

Now construct a module for the 4-to-1 multiplexer shown in Figure 3 with the truth table shown in Table 1 using wires and multiple instances of the mux2to1 module. Note that the truth table in Table 1 is given in a short-hand form. A real truth table would consist of rows enumerating all possible combinations of values of inputs u, v, w, x, in addition to s_0 and s_1, and show the value (0 or 1) of the output m for each row of the truth table. Since this would result in a truth table with a large number of rows, it is written in short-hand form instead.
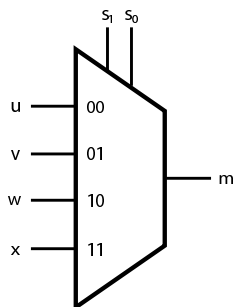

Figure 3: Symbol for a 4-to-1 multiplexer

$s_1 s_0$	m
00	u
01	v
10	w
11	x

Table 1: Truth table for a 4-to-1 multiplexer

Perform the following steps:

1. Answer the following question: if the truth table in Table 1 was given in full, how many rows would it have? **(PRELAB)**

2. Draw a schematic (not in Logisim) showing how you will connect the *mux2to1* modules to build the 4-to-1 multiplexer. Be prepared to explain it to the TA as part of your prelab. The schematic should reflect how you are going to create your Logisim circuit. **(PRELAB)**

3. Build your circuit in Logisim **(PRELAB)**

4. Test your circuit with different values of s, u, v, w and x. Do enough testing to convince yourself that the circuit is working. You must show these to the TA as part of your prelab. **(PRELAB)**

Answers to questions and hand-drawn schematics should be included in your prelab report and submitted online before the lab

7 Part III

In this part of the lab, you will design a decoder for the 7-segment HEX display as shown in Figure 4. The output of the HEX display is determined by the value at the input of the decoder as shown in Table 2. We call this a HEX display because it can display all hexadecimal digits.

When you build your circuit in Logisim, use default inputs and *7-Segment Display* as output.

HINT: In order to solve this part you need to first identify which segment needs to be illuminated for every input and then write a Boolean function for each one of the seven segments of the HEX display so they are turned on when needed. You must use Karnaugh maps to optimize those Boolean expressions before you build the corresponding circuits in Logisim.
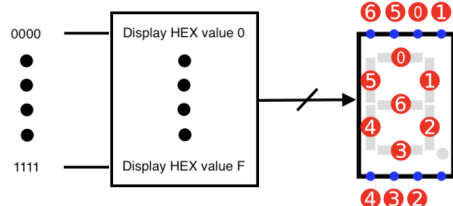
$c_3c_2c_1c_0$	Character
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	b
1100	C
1101	d
1110	E
1111	F

Figure 4: HEX decoder   Table 2: Desired behaviour of HEX decoder

Perform the following steps:

1. Write the expressions for seven Boolean functions, one for each segment of the 7-segment decoder. You must use Karnaugh maps for optimization. You should be able to explain to the TA how you generated these expressions by showing them the truth-tables you wrote, and Karnaugh maps you used to optimize your circuits. **(PRELAB)**

2. Build the circuit for the 7-segment decoder in Logisim taking advantage of the aforementioned expressions. HINT: you can change the number of inputs for each gate by selecting it and go to *Properties*. Also it might be helpful to separate the circuits for each of the 7 segments into different modules. **(PRELAB)**

3. Check whether your implementation is correct. To verify the correctness of each segment, you need to create a test vector file for each segment's truth table. To test whether your circuits for each segment connect well with the *7-Segment Display* in Logisim, use 'Poke' on the tool bar. Show the screenshots of your simulation to your TA as part of the prelab. **(PRELAB)**