LAB 2

THE DESIGN HIERARCHY

In Lab 2, you build combinational circuits using *hierarchical design* (a technique that emphasizes modularity). In addition, you use Karnaugh maps to ensure that your circuits are optimized. Before the lab, you prepare by deriving truth tables and logic functions, and by drawing schematics using Logisim Evolution. During the lab, you simulate your designs to demonstrate your understanding of the circuit.

This document describes what you need to prepare and demonstrate for Lab 2. Section 2.3 briefly describes hierarchical design in Logisim Evolution. Section 2.4 describes the tasks you must complete *before* your lab session. Section 2.5 describes the tasks you complete *during* your lab session. The next section describes lab logistics in more detail.

2.1 Logistics

Even though you work in pairs during your lab session, you are assessed individually on your Lab Preparation ("pre-lab") and Lab Demonstration ("demo"). All pre-lab exercises are submitted electronically before your lab (see the course website for exact due dates, times, and the submission process). So, **before** each lab, you must read through this document and complete all the pre-lab exercises. During the lab, use your pre-lab designs to help you complete all the required in-lab actions. The more care you put into your pre-lab designs, the faster you will complete your lab.

Before beginning the pre-lab, read Section 2.3. The Lab Preparation must be completed individually and submitted online by the due date. Follow the steps in Section 2.4 for the pre-lab. Remember to **download the starter files** (if provided).

You must upload *every required file* for your pre-lab submission to be complete. But you do *not* need to include images that are not on the list of required files (even if those images are in your lab report). If you have questions about the submission process, please ask ahead of time. The required files for Lab 2's pre-lab (Section 2.4) are:

- Your lab report: `lab2_report.tex`, `lab2_report.pdf` (as generated from the tex file)
- Your digital designs: `lab2_part1.circ`, `lab2_part2.circ`

The Lab Demonstration must be completed during the lab session that you are enrolled in. During a lab demonstration, your TA may ask you to: go through parts of your pre-lab, run and simulate your designs in Logisim, and answer questions related to the lab. You may not receive outside help (e.g., from your partner) when asked a question.

2.2 Marking Scheme

Each lab is worth 4% of your final grade, where you will be graded out of 4 marks for this lab, as follows.

- Prelab: 1 mark
- Part I (in-lab): 1 mark
- Part II (in-lab): 1 mark
- Part III (in-lab): 1 mark

2.3 Hierarchical Design in Logisim Evolution

In Lab 1 you designed a 2-to-1 multiplexer. Multiplexers are a common building block in digital circuits. In this section, we first describe how to convert a circuit, like a 2-to-1 multiplexer, into a building block. We then describe how to test your circuits. Testing smaller circuits (e.g., 2-to-1 multiplexer) helps with the testing of your larger circuits (e.g., that use multiplexers).

2.3.1 Subcircuits

In Logisim Evolution, a `.circ` file can be composed of multiple circuits. There is a top-level circuit, typically called `main`, which you should be familiar with from Lab 1. But it is possible to add more circuits called *subcircuits* (or *modules*). From your programming experience, this is analogous to creating a function.

Subcircuits are typically building blocks, like a 2-to-1 multiplexer, used in other subcircuits and/or the top-level module. Each time a subcircuit is used in another circuit, it is an *instance* of that module. Note that, if you label an instance, the name should be unique. We can create large circuits by using multiple modules of the same type. From your programming experience, this is analogous to function reuse.

Logisim Evolution includes a tutorial on hierarchical design. You can find it by launching Logisim Evolution and navigating to `Help > User's Guide`. There, you will find a section (and its subsections) called `Subcircuits`.

2.3.2 Testing Circuits

It is a good idea to verify the correctness of your circuits and subcircuits. Logisim Evolution provides a method to compare what your circuit actually outputs with what you expect it to output. You can accomplish this by navigating to `Simulate > Test Vector...`.

A test vector is essentially a plain text file containing a truth table. The truth table maps a set of inputs to the expected output. The first line of the file specifies the names of the inputs and outputs - these names must match those found in your circuit. In the lines that follow, you include the values to use for the input(s) and the value you expect at the output(s).

Figure 2.1 shows a circuit that consists of only an AND gate. Listing 2.1 shows the corresponding test vector for the circuit. The test vector must be saved as a text file before it can be loaded into Logisim Evolution. Once loaded, the status of each row is given (Figure 2.2).

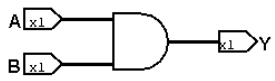

Figure 2.1: Contrived Circuit

```
# my_test_vector.txt
A B Y
0 0 0
0 1 0
1 0 0
1 1 1
```

Listing 2.1: Test Vector

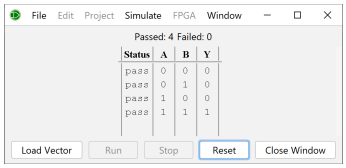

Figure 2.2: Test Outcomes

2.4 Lab Preparation

Make sure you have read Section 2.3 before beginning your pre-lab.

The pre-lab for Lab 2 consists of two parts. In Part I, you follow a tutorial on Hierarchical Design to implement a 4-to-1 multiplexer. In Part II, you apply hierarchical design to create a decoder that visualizes hexadecimal characters on a 7-segment display.

2.4.1 Part I

In this part, you construct a module for the 4-to-1 multiplexer shown in Figure 2.3 with the truth table shown in Table 2.1 using hierarchical design. Note that the truth table in Table 2.1 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.

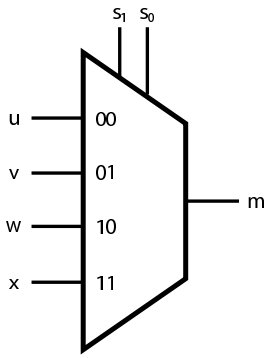

Figure 2.3: Symbol for a 4-to-1 multiplexer

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

Table 2.1: Truth table for a 4-to-1 multiplexer

1. If the truth table in Table 2.1 was given in full, how many rows would it have?

2. In a file called `lab2_part1.circ`, follow the `Subcircuits` tutorial in Logisim Evolution to create a 4-to-1 multiplexer out of multiple instances of 2-to-1 multiplexers. However, make sure you meet the following requirements and naming schemes:

   - You must have a subcircuit of a 2-to-1 multiplexer called `mux2to1`.
   - You must have a subcircuit of a 4-to-1 multiplexer called `mux4to1`. This module can only contain instances of the `mux2to1` subcircuit (as well as wires, inputs, and outputs).

- You must have a top-level circuit called `main`. This module should only contain one instance of the `mux4to1` subcircuit (as well as wires, inputs, and outputs).

Export the schematic of the `mux4to1` subcircuit as an image and include it in your report. Also remember to upload the designs (i.e., the `.circ` file) as part of your required files.

2.4.2 Part II

In this part, you design a decoder for a 7-segment display (Figure 2.4). As the name implies, the display has seven segments (lines) that can be ON or OFF. The 7-segment display shown in Figure 2.4 has a number associated with each segment. So we can refer to a segment by its number. For example, the top-most horizontal segment is S_0, which we can turn ON or OFF by driving pin 0 (third dot from the left at the very top of the display).

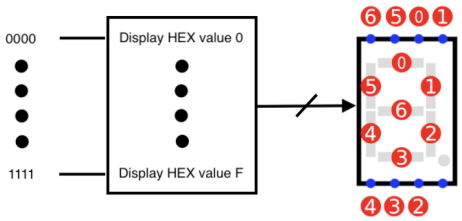

Figure 2.4: A decoder driving a 7-segment display

$D_{3:0}$	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

Table 2.2: The desired behaviour of the decoder

The segments are arranged in a way such that you can visualize the numbers 1 to F in hexadecimal. See Table 2.2 for a mapping of decoder inputs to the characters that need to be displayed. The decoder you are designing takes, as input, 4 bits and produces a 7-bit output that controls the LEDs in the 7-segment display. To design the decoder, you need to identify when a segment in the display is on or off.

Since there are seven segments, you should first derive seven Boolean functions. Then, use Karnaugh maps to ensure these Boolean functions are minimal. Next, implement each Boolean function as its own subcircuit. And, finally, use hierarchical design to construct the decoder and drive a 7-segment display.

CAUTION

Example 2.10 in the textbook is a very useful starting point. But be careful, the textbook example is only concerned with visualizing the characters 0 to 9. This lab is also asking you to visualize the outputs associated with A, b, c, d, E, F (Table 2.2).

Perform the following steps:

1. In your lab report, derive seven truth tables, one for each segment of the 7-segment decoder. Another way to ask this question is: which segments should be on (and which should be off) for a given character?

2. Use Karnaugh maps to write seven Boolean functions for each segment so that they are optimized. You do not need to include these in your report, but **you should save these Karnaugh maps for your Lab Demonstration.**

3. In a file called `lab2_part2.circ`, create seven subcircuits, one for each optimized function. Use the naming scheme `HEX0`, `HEX1`, ..., `HEX6` for each subcircuit. Each subcircuit should have input pins called `D3, D2, D1, D0` and an output pin `SX`, where X corresponds to which segment is being driven (e.g., `S0` for segment 0). Export each subcircuit schematic as an image and include it in your report.

4. In the same file, create a subcircuit called `HEX_DECODER` using hierarchical design. This module should only contain instances of your `HEX#` subcircuits (as well as wires, inputs, and outputs).

5. In the same file, in your top-level circuit, create a circuit like the one shown in Figure 2.4. The circuit should have four input pins called `SW3, SW2, SW1, and SW0`. Once again, use hierarchical design to instantiate your `HEX_DECODER` subcircuit. In Logisim Evolution, you can find the 7-Segment Display under `Input/Output`; your top-level circuit should not have an output pin, only the 7-segment display. When you are done, remember to upload your designs (i.e., the `.circ` file) as part of your required files.

2.5 Lab Demonstration

The lab demonstration consists of three parts. Each part corresponds to a part from the pre-lab. Part II-B involves *synthesising* the `HEX_DECODER` so that it runs on the DE1-SoC board. When demonstrating a part to your TA, be ready to answer questions **individually**.

2.5.1 Part I

Test your top-level circuit in Logisim Evolution with the `Poke` tool. Test your subcircuits with test vectors (Section 2.3.2). **Make sure you are ready to, if asked, explain your testing methodology and/or how you optimized your Boolean functions with Karnaugh maps.** When you are done, demonstrate your working design to your TA.

2.5.2 Part II

Test your top-level circuit in Logisim Evolution with the `Poke` tool. Test your subcircuits with test vectors (Section 2.3.2). **Make sure you are ready to, if asked, explain your testing methodology and/or how you optimized your Boolean functions with Karnaugh maps.** When you are done, demonstrate your working design to your TA.

2.5.3 Part III

Synthesise your `HEX_DECODER` design and download it onto the DE1-SoC board. You will need to perform the steps below. When you are done, demonstrate your working circuit to your TA.

2.6 Uploading to the DE1-SOC Board

CAUTION

Make sure you are using a lab computer (i.e., a desktop computer in BA3135, BA3145, BA3155, or BA3165) and have 'installed' Logisim 3.6.1. Also make sure you have `DE1_SOC_7seg.xml` downloaded on the lab computer. Finally, make sure the DE1-SoC board is turned on.

1. Navigate to `FPGA > Synthesize & Download`. A small window should open (Figure 2.5).

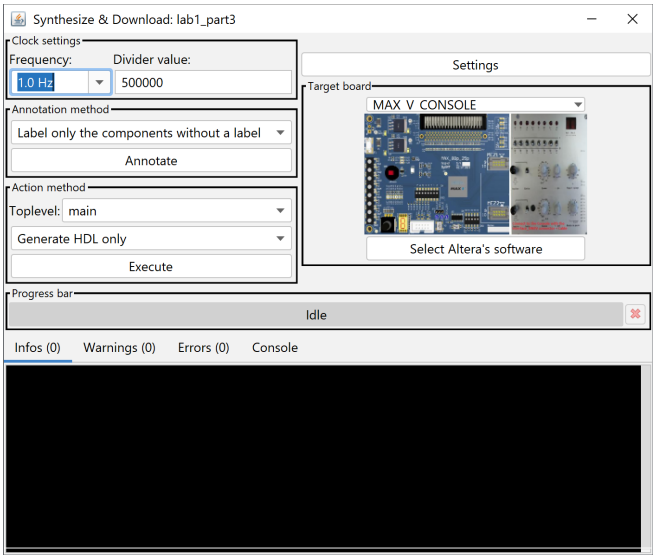

Figure 2.5: The 'Synthesize & Download' window

2. In the `Target board` panel of the window, select `Other` from the pull-down menu. An open file dialogue should open.

3. Navigate to, and `Open`, the `DE1_SOC_7seg.xml` file that you downloaded.

4. Click on the `Settings` button just above the `Target board` panel. A Preferences window should open (Figure 2.6). You should see `DE1_SOC_7seg` under `External FPGA Board list:`.

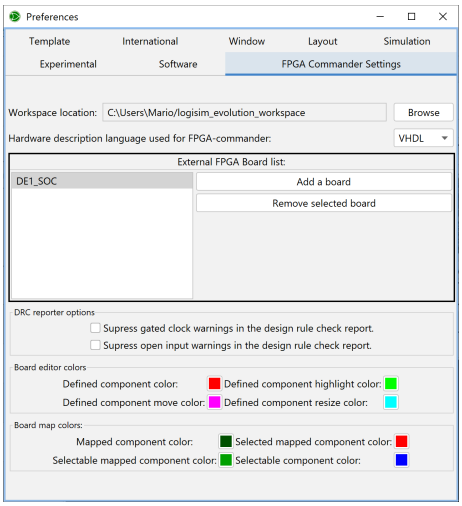

Figure 2.6: The Preferences: FPGA Commander tab

5. Click on the `Browse` button beside `Workspace location`. An open file dialogue should open.

6. Create a new subdirectory in your home directory and select it as your workplace location.

7. Navigate to the `Software` tab (Figure 2.7). Click on `Browse` next to the `Altera/Intel Quartus toolpath`. An open folder dialogue should open.

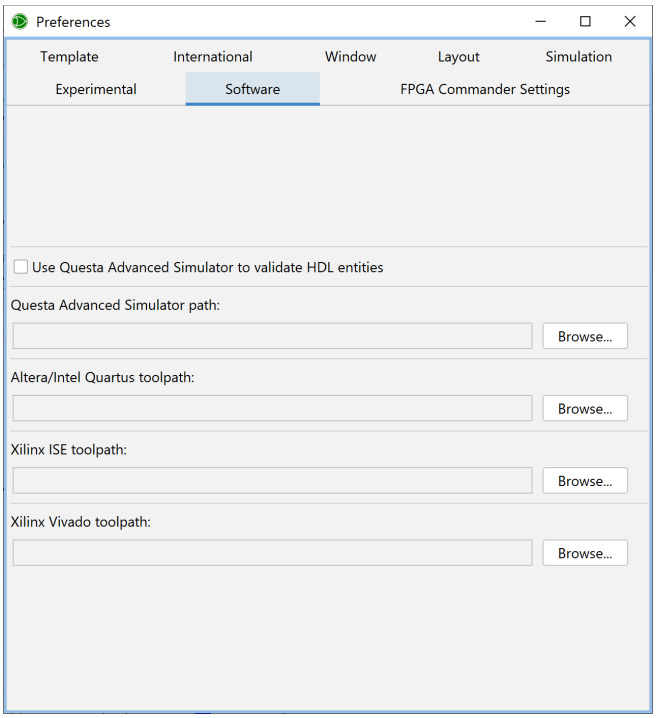

Figure 2.7: The Preferences: Software tab

8. Navigate to, and `Open`, the `C:\DESL\Quartus18\quartus\bin64` folder.

9. Close the `Preference` window. Everything should now be configured.

10. In the `Action method` panel (Figure 2.5), select the appropriate Toplevel circuit (typically main). From the pull-down menu, make sure that `Synthesize & Download` (not `Generate HDL only`) is selected.

11. Click on `Execute` under `Action method`. A window should open (Figure 2.8).

    With this version of Logisim Evolution, your circuit's inputs can be mapped to the buttons and switches on the physical DE1-SoC board. Similarly, your circuit's outputs can be mapped to the LEDs, or individual segments on the 7 segment displays. **Make sure that you change your circuit's output in the Logisim file to output pins so you can map each pin to a separate segment.**

12. Map the pins in the `Unmapped List:`. First, select one of the pins. The components that this pin can be mapped to are highlighted in translucent red on the image of the board. Click on one to map the pin. Repeat until all pins have been mapped to a physical component on the DE1-SoC board.

13. Click on `Done` to start synthesising your circuit. This may take some time.

14. Select `Yes, download` when presented with the option.

15. Test your circuit by toggling switches and observing the LEDs and 7 segment displays.

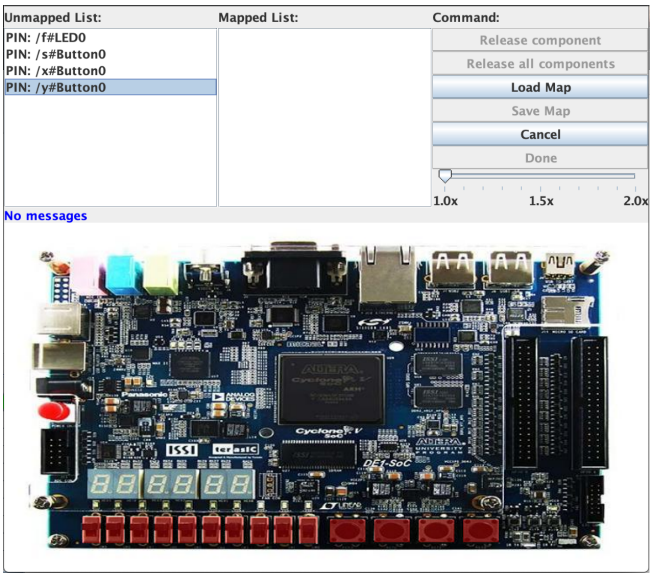

Figure 2.8: The board mapping window