CSCB58 - Lab 1

Intro to Verilog

Learning Objectives

This lab will serve as an introduction to Verilog, and show you how to get Verligo code running on the DE2 board, and use that code to manipulate switches and LEDs, as well as the 7-Segment displays.

Note that we may refer to signals as SW_{17-0}, i.e., with the subscripts, but when you write your Verilog, you will need to use `SW[0]`, `SW[1]`, etc.

Marks

Your TA must record the marks on this page as you complete each section of the lab. It is your responsibility to ensure that by the end of the lab, all work has been recorded appropriately.

Prelab	/3
Part I (in-lab)	/1
Part II (in-lab)	/1
Part III (in-lab)	/2
Clean work-space with all materials returned to their original state	/1
TOTAL	/8

Write your name, UTorID, and student ID:

Name: ____________________________________

Student ID: ________________________________

UTorID: __________________________________

Write your partner's name, UTorID, and student ID:

Partner name: ______________________________

Partner student ID: __________________________

Partner UTorID: ____________________________

Preparation Before the Lab

For this lab, and all future labs, you will be asked to prepare schematics (not in Quartus) and Verilog code for your prelab. The schematics should show the structure of your Verilog code, much like the schematics in Lab 0 showed how your circuit should be built. Your Verilog code will consist of a number of **modules** and the schematic should show how those modules are wired together, as well as the input and output ports of your circuit, i.e., connections to switches, LEDs, 7-segment hex displays, etc. Think of **modules** as just complex *gates*. In addition, all port names of the modules, wires and I/O ports should be clearly labeled in your schematics. Figure 2 below is an example. Finally, your Verilog code should be well-commented.

DE2 Resources

You can find information and resources (such as user manual and software) for Altera DE2-115 board here: `https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=502`.

The User Manual for the DE2 board can be downloaded from here: `https://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=China&No=502&FID=cd9c7c1feaa2467c58c9aa4cc02131af`

Drawing Schematics

Here are some CAD tools that you might fine useful to draw your logic circuits and schematics. These are just suggestions. If you have another tool that you like to use, you are welcome to use that instead.

- TinyCAD: `https://sourceforge.net/projects/tinycad/`
- RFFlow: `https://www.rff.com/index.php`
- Logicly: `https://logic.ly/demo/`

Part I

Running a Verilog File (.v):

The DE2 board provides 18 toggle switches, called SW_{17-0}, that can be used as inputs to a circuit, and 10 red lights, called $LEDR_{9-0}$, that can be used to display output values.

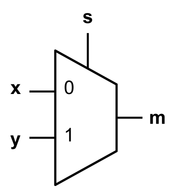
A Verilog file for a 2-to-1 multiplexer, named `mux.v`, has already been provided to you. The top module *mux* has 3 inputs. SW[0] is the input 0 signal, SW[1] is the input 1 signal, and SW[9] is the select signal. The output is displayed on LEDR[0].

```
module mux (SW, LEDR); // module name and port list
```

The top module, *lab_1*, is a very trivial example of a design hierarchy, as it instantiates a single *mux2to1* module. In the more general case, any module can instantiate a number of interconnected modules. However, in any circuit you build, there must be only one top-level module. The `.port(connection)` notation matches the port name from the *mux2to1* module to a connection (e.g., port or internal wire) inside the *mux* top-level module. Note that every instance has to have a unique name. In our example below we named our instance of the mux2to1 module *my_mux*.

```
mux2to1 my_mux(
    .x(SW[0]),     // connect port SW[0] to port x of the module
    .y(SW[1]),     // connect port SW[1] to port y of the module
    .s(SW[9]),     // connect port SW[9] to port s of the module
    .m(LEDR[0])    // connect port LEDR[0] to port m of the module
);
```

Figure 1 shows the symbol for a 2-to-1 multiplexer. 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 our example below, input *s* is the select signal. It controls which of the inputs *x* and *y* will appear on the output *m*. If *s* is 0, the output *m* will be equal to *x*, while if s is 1, it will be equal to *y*.



S	m
0	x
1	y

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

The boolean expression for a 2-to-1 multiplexer is M = XS'+ YS, where S' means NOT(S). One way you can express this in Verilog is the following:

```
assign m = x & ~s | y & s;
```

The following table describes the bitwise Verilog operators.

Operator	Description
|	bitwise OR
&	bitwise AND
~	bitwise negation (NOT)
^	bitwise XOR

Perform the following steps to run your verilog file on the DE2 board (you may want to keep this page handy until you get comfortable with this series of steps):

1. Open Quartus
2. Run the new project wizard
   (a) Choose a name and working directory
      - Note that your project must be named `lab_1`, as that is the name of the top-level module

   (b) Choose “Empty Project"
   (c) Add the file `mux.v` to your project
   (d) Choose the appropriate device (Cyclone IV E → EP4CE115F29C7)
   (e) Make sure your simulation is using ModelSim-Altera and your format is set to Verilog HDL)
3. Map your variable names to the board pin assignments
   - To save you doing this all the time, we have supplied a default mapping in the file `de2.sqf`
   - Simply choose Assignments → Import Assignments, and it should map all of the standard variable names to the appropriate pins
4. Run synthesis on your code
   - Processing → Start → Start Analysis & Synthesis
5. Compile the project
   - Project → Start Compilation
6. Power on your DE2 board
7. Run the code on the DE2 board
   (a) Tools → Programmer
   (b) Make sure "Hardware Setup" is set to "USB-Blaster"
   (c) Add the output file from your compilation (Add File → Find your .sof file in your output folder)
   (d) Click “Start"

Test your code on the DE2 board. Now, try changing the switches/LEDs that the code will use to make sure you really understand what the verilog file is doing. Try making it work with switch 17 being `s`, and use one of the green LEDs for output. **Show your work to the TA.**

Part II

Start with the code 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 – the concept of creating higher level circuits from lower level cells – and is a good practice especially for larger designs where the Verilog code you write can become more difficult to debug.

To complete this section, you will need to use the **wire** declaration to create wires that can be used to connect the multiple blocks together.

```
wire Connection; // creates a wire called Connection
```

The wire created above is called *Connection* and it can be used to connect the output of a module to the input of a module, like the wires you drew in Lab 0 to connect the output of one gate to the input of another gate. Figure 2 shows a schematic of two modules using the wire *Connection*. You may also use *Connection* as input to other logical expressions within the module where the wire is defined.

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

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

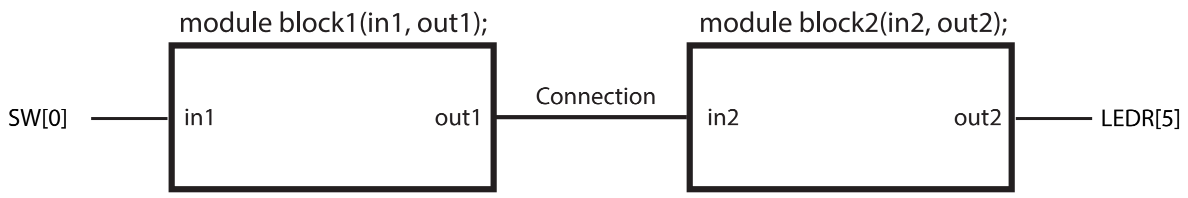


Figure 2: Using the wire *Connection* to make a connection between two modules

The following code fragment corresponds to Figure 2. It creates *instances* of modules *block1* and *block2*, named *B1* and *B2*, respectively. The wire *Connection* is used to wire the module instances together.

```
block1 B1 (
.in1(SW[0]),         // assign port SW[0] to port in1
.out1(Connection)    // assign wire Connection to port out1
);

block2 B2 (
.in2(Connection),    // assign wire Connection to port in2
.out2(LEDR[5])       // assign port LEDR[5] to port out2
);
```

Another way to make a connection is to use the **assign** statement. For example, if we wanted to connect the **wire** called *Connection* to *LEDR*$_0$, we do the following:

```
assign LEDR[0] = Connection; // joins wire Connection to LEDR[0]
```

Now construct a module for the 4-to-1 multiplexer shown in Figure 3 with the truth table shown in Table 1 using the **wire** construct and multiple instances of the mux2to1 module.

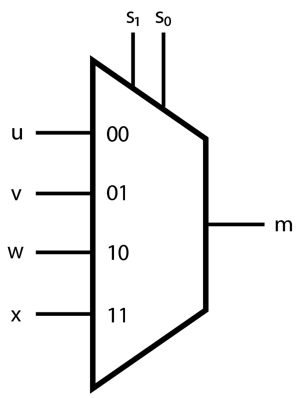


Figure 3: Symbol for a 4-to-1 multiplexer

c_3c_2 \ c_1c_0	00	01	11	10
00				
01				
11				
10				

0. 0 2 3 5 6 7 8 9 10 12 14 15
1: 0 1 2 3 4 7 8 9 10 13
2: 0 1 3 4 5 6 7 8 9 10 11 13
3: 0 2 3 5 6 8 9 11 12 13 14
4: 0 2 6 8 10 11 12 13 14 15
5: 0 4 5 6 8 9 10 11 12 14 15
6: 2 3 4 5 6 8 9 10 11 13 14 15

Table 2: Truth table for HEX decoder

$c_3c_2c_1c_0$	Character	
0000	0	0 1 2 3 4 5
0001	1	1 2
0010	2	0 1 3 4 6
0011	3	0 1 2 3 6
0100	4	1 2 5 6
0101	5	0 2 3 5 6
0110	6	0 2 3 4 5 6
0111	7	0 1 2
1000	8	0 1 2 3 4 5 6
1001	9	0 1 2 3 5 6
1010	A	0 1 2 4 5 6
1011	b	2 3 4 5 6
1100	C	0 3 4 5
1101	d	1 2 3 4 6
1110	E	0 3 4 5 6
1111	F	0 4 5 6

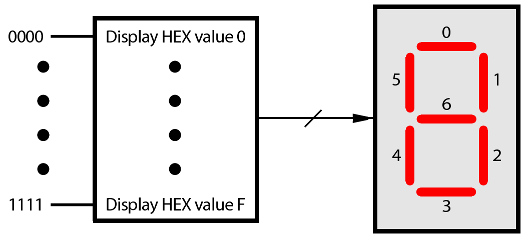


Figure 4: HEX decoder

Perform the following steps:

1. Draw a schematic (not in Quartus) 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 write your Verilog code. **(PRELAB)**
2. Create a new Quartus II project for your circuit and write the Verilog code.
3. Include your Verilog file for the circuit in your project. Use switches SW_{9-8} on the DE2 board as the 2-bit s input, switches SW_{3-0} as the inputs. Connect the output to $LEDR_0$. Do not forget that you will need the `de2.qsf` file to define how the switches and LEDs connect to the pins.
4. Compile the project.
5. Download the compiled circuit into the FPGA chip. Test the functionality of the circuit by toggling the switches and observing the LEDs.

Once done, show your work to your TA

Part III

In this part of the lab, you are to 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 displays hexadecimal digits.

Remember that in order to light up a segment of a 7-segment display, its value must be set *low*. e.g., in order to draw a 1 on the 0th display, $HEX0_1$ and $HEX0_2$ should be set to 0, while the others (HEX_0 and HEX_{2-6}) should be set to 1.

Debugging Strategy: Before diving into the full Part III implementation, you may find it helpful to start by creating a very simple Verilog module which turns on segment 0 of the 7-segment display, i.e., pin $HEX0_0$, based on the input from SW_0.

Perform the following steps:

1. Create boolean expressions for each segment of the 7 segment display. Use K-maps (Karnaugh maps) to simplify them. **(PRELAB)**

2. Draw a schematic of the circuit you want to build and be prepared to explain it to the TA as part of your prelab. The schematic should reflect how you are going to write your Verilog code. You need not show every single gate in your design but you should show the overall circuit structure using hierarchy. **(PRELAB)**
3. Create a new Quartus II project for your circuit and write the Verilog code.
4. Create a Verilog module for the 7-segment decoder. Connect the $c_3c_2c_1c_0$ inputs to switches SW_{3-0}, and connect the outputs of the decoder to the *HEX0* display on the DE2 board. The segments in this display are called $HEX0_0$, $HEX0_1$, ..., $HEX0_6$. You should declare the 7-bit port like this in your Verilog code:

   ```
   output [6:0] HEX0;
   ```

   so that the names of these outputs match the corresponding names in the *DE2 User Manual* and the pin assignment `de2.qsf` file.
5. Compile the project.
6. Download the compiled circuit into the FPGA chip. Test the functionality of the circuit by toggling the SW_{3-0} switches and observing the 7-segment display.

Show your work to the TA.

Finishing Up

1. Zip the files and take them with you. You may need it in future labs!
2. Clean up you work space and return all materials.
3. Don't forget to upload to Quercus. All partners have to submit to Quercus individually.