University of California, Davis
Department of Electrical and Computer Engineering

EEC180 **DIGITAL SYSTEMS II** **Winter Quarter 2024**

Lab 1: Modelsim/Quartus Tutorial

Objective:
This tutorial/lab covers the complete design flow for implementing a high-level Verilog design on the DE10-Lite board. The tools covered in this tutorial include System Builder, Quartus Prime and ModelSim. Part 0 uses a simple design example to illustrate the design flow. In Part I, you will then practice implementing simple logic gates on the DE10-Lite board. In Part II, you will design and implement a combinational logic circuit to display a 4-bit input in decimal on 7-segment displays.

Resources:
The Terasic webpage for the DE10-Lite board contains links to information and resources for the board. These resources include:

- DE-10-Lite User Manual, Version 1.4, Release Date 01-24-2017
- DE10-Lite CD-ROM, Version 2.0.0, Release Date 01-24-2017

http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=218&No=1021&PartNo=4

You can download the DE10-Lite CD-ROM without creating a Terasic account here:
https://download.terasic.com/downloads/cd-rom/de10-lite/

Download the DE10-Lite_v.2.1.0_SystemCD.zip file and extract it to a safe location on your Windows PC or VM. Note that the CD-ROM image has a copy of the user manual under **Manual/**.

Prelab – Tool Installation

We strongly recommend you install the tools on your local machine. While there are options to use the ECE department machines, you will not be able to program your hardware outside of the lab unless you have a local installation. The local machine should either run Windows natively or in a virtual machine.

DE-10 Lite Boards:
Your DE10-Lite kit will be distributed during the first week of the quarter. Your kit will include a DE10-Lite board, and USB-Blaster cable. If you notice any inactive LEDs, or other components that fail throughout the quarter, please notify a TA so they can swap your board.

Installing the Quartus and ModelSim Tools on your Personal Computer:
The preferred configuration this quarter is Intel Quartus Prime 19.1 for design and ModelSim-Intel FPGA Edition 19.1 for simulation. The following guide describes how to install the tools on your local system. We strongly recommend you install the tools locally.

Setup guide:
https://canvas.ucdavis.edu/courses/853816/files/folder/Labs/ToolsSetup

Kemper 2110 Lab Machines:
The machines in the lab run Windows and include Intel Quartus Prime 19.1 for design and ModelSim-Intel FPGA Edition 19.1. These machines can be used during the lab session to run your labs. However, they cannot be accessed remotely.

<u>Part 0 – Design Flow Using System Builder, Quartus Prime and ModelSim</u>

A. Proper file organization

In this course, there will be a wide variety of tool-generated files, design files, and testbench files. **Placing all these files into a single folder will cause hard-to-fix tooling issues and frustration**. To prevent these issues, we have provided a folder with the recommended file organization for the course in Canvas under Files > Labs > ProjectTemplate. Download this folder as shown below:

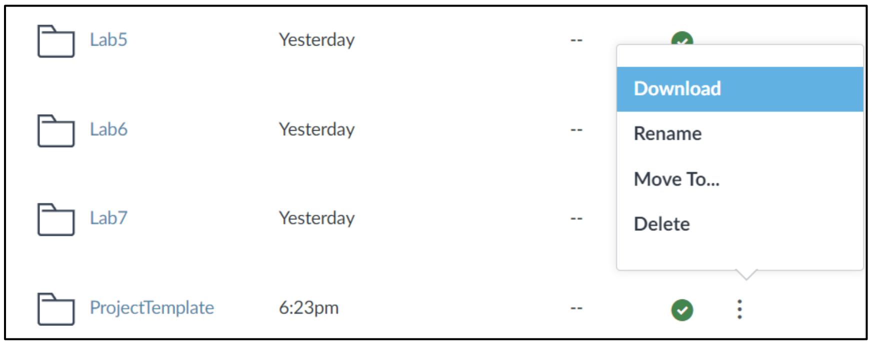

Lab5	Yesterday	--
Lab6	Yesterday	--
Lab7	Yesterday	--
ProjectTemplate	6:23pm	--

Figure 0. System Builder Configuration

Extract the downloaded .zip folder. It is recommended to keep a fresh copy of the ProjectTemplate folder on your system so that you can copy the folder for each new lab exercise.

B. Creating Project Files Using System Builder

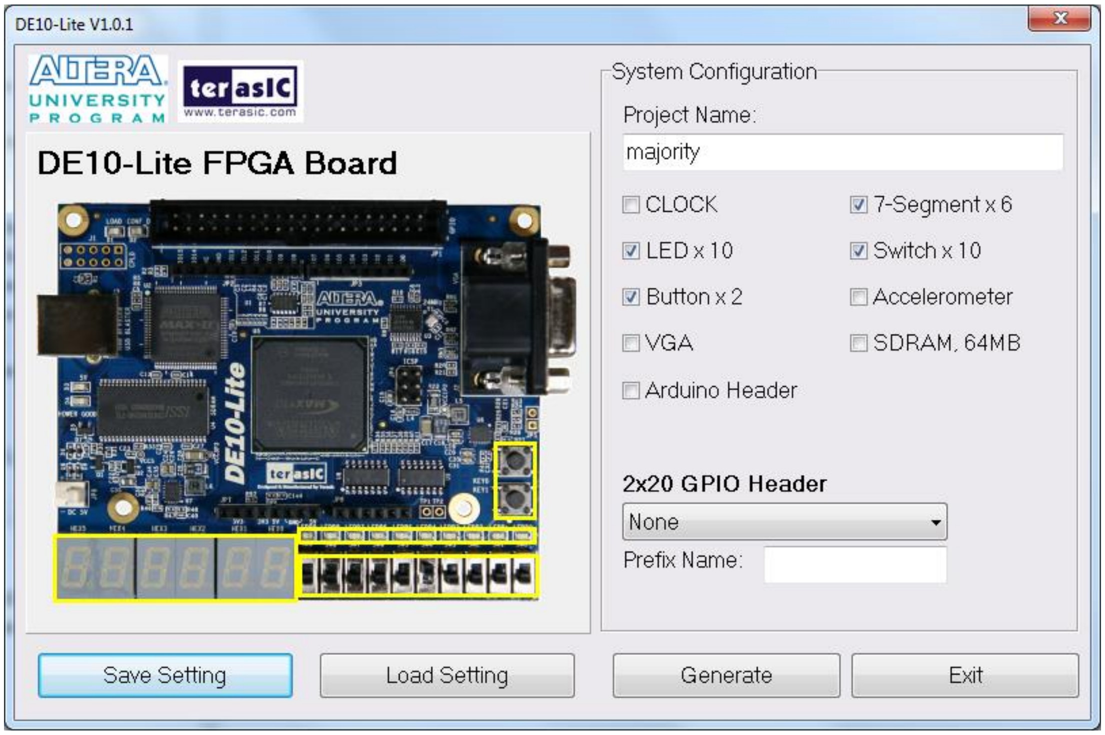
Make a copy of the **ProjectTemplate** folder and rename it **lab1**. The goal of this project organization is to separate tool-generated files and your design and testbench files and to ensure that multiple Quartus/ModelSim projects never conflict and/or overwrite one another. Review the README.txt files in the main directory and each subfolder.

In all labs, you will want to create new subdirectories in **synthesis/** and **simulation/** for each exercise that requires a new Quartus/ModelSim project. Make a new folder in the **lab1/synthesis/** directory and name it **majority** Make another new folder in the **lab1/simulation/** directory and name it **tb_majority**. Both Part I and Part II will require Quartus and ModelSim projects, so you will create **partI**, **tb_partI**, **partII**, **tb_partII** subdirectories.

System Builder is a Windows-only utility included in the DE10-Lite CD-ROM and described in Chapter 4 of the User Manual. This tool generates Quartus Prime project files and does the pin assignment mapping for the I/O pins on the DE10-Lite board.

1. Run **DE10_Lite_SystemBuilder.exe** located in the DE10-Lite_v.2.1.0_SystemCD folder under Tools > SystemBuilder.

2. Specify the project name and the I/O devices that you will use in your project.
   a. For this tutorial project specify the Project Name as **majority** and select the LED, Button, Switch, and 7-Segment I/O devices, as shown in Figure 1.
   b. Then click the Generate button and save the project files to the copied ProjectTemplate folder under the newly created synthesis directory (i.e. **lab1/synthesis/majority**). Once you have successfully generated your Quartus Prime project files, exit System Builder.



System Configuration

Project Name: majority

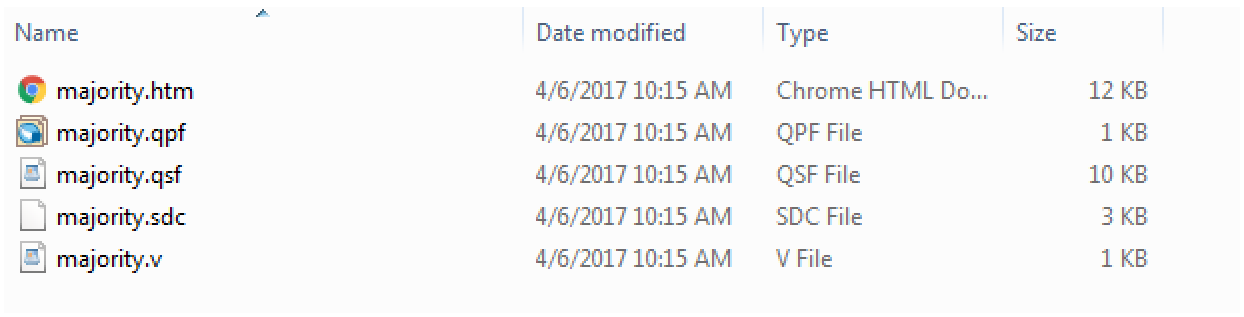
[ ] CLOCK
[✓] 7-Segment x 6
[✓] LED x 10
[✓] Switch x 10
[✓] Button x 2
[ ] Accelerometer
[ ] VGA
[ ] SDRAM, 64MB
[ ] Arduino Header

2x20 GPIO Header: None

Prefix Name:

Figure 1. System Builder Configuration

System Builder will create a set of files in your project directory, as shown in Figure 2. You will need to be concerned with only a few of these files in this lab.



Name	Date modified	Type	Size
majority.htm	4/6/2017 10:15 AM	Chrome HTML Do...	12 KB
majority.qpf	4/6/2017 10:15 AM	QPF File	1 KB
majority.qsf	4/6/2017 10:15 AM	QSF File	10 KB
majority.sdc	4/6/2017 10:15 AM	SDC File	3 KB
majority.v	4/6/2017 10:15 AM	V File	1 KB

Figure 2. Files Generated by System Builder

The **majority.v** file is the template for your top-level Verilog design entity. The template generated by System Builder is shown in Figure 3. System Builder maps the I/O devices with the proper pin assignments that are hardwired on the DE10-Lite board. This is a great convenience so that you don’t need to manually do the pin mapping. This is one of the biggest advantages of using System Builder to create your project file and Verilog template file. Notice that the declarations and structural coding sections are left blank for you to complete.

```
//=======================================================
//  This code is generated by Terasic System Builder
//=======================================================

module majority(

      //////////// SEG7 //////////
      output                 [7:0]        HEX0,
      output                 [7:0]        HEX1,
      output                 [7:0]        HEX2,
      output                 [7:0]        HEX3,
      output                 [7:0]        HEX4,
      output                 [7:0]        HEX5,

      //////////// KEY //////////
      input                  [1:0]        KEY,

      //////////// LED //////////
      output                 [9:0]        LEDR,

      //////////// SW //////////
      input                  [9:0]        SW
);



//=======================================================
//  REG/WIRE declarations
//=======================================================




//=======================================================
//  Structural coding
//=======================================================



endmodule
```

Figure 3. System Builder Verilog Template (majority.v)

In this example, we will illustrate the design of a majority gate, as described in Section 3.6 of your text, Digital Design – A Systems Approach by Dally. We will use switches for the inputs and an LED for the output.

C. Compiling and Programming Using Quartus Prime

1. Run Quartus Prime and open the project file that you just created using System Builder.

2. Select File > Open Project and navigate to the folder where you stored your files. Select the **lab1/synthesis/majority/majority.qpf** file.

3. Double-click the majority entity instance in the Project Navigator pane to open the template created by System Builder for your top-level design.

4. Add the following line of code in the Structural coding section of the **majority.v** file:

```
assign LEDR[1] = (SW[0]&SW[1]) | (SW[0]&SW[2]) | (SW[1]&SW[2]);
```

As described in lecture, posted notes, and your text (p. 50), Verilog uses the symbols &, |, ^, and ~ to represent the logic operations AND, OR, XOR, and NOT, respectively. The keyword assign is used to describe a combinational logic function. In this case, LEDR[1] will be asserted (logic high) when at least 2 of the 3 switch inputs, SW[0]-SW[2], are high.

Note that the LEDs are active high, as shown in Figure 4.

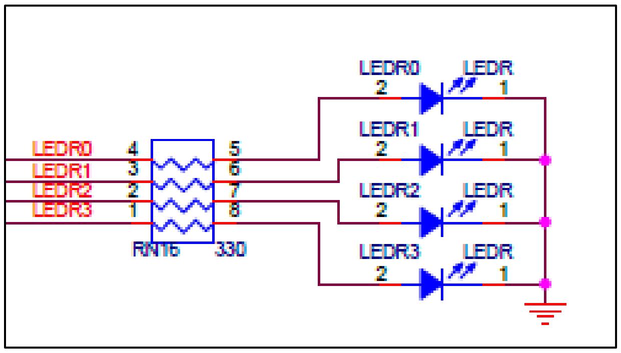
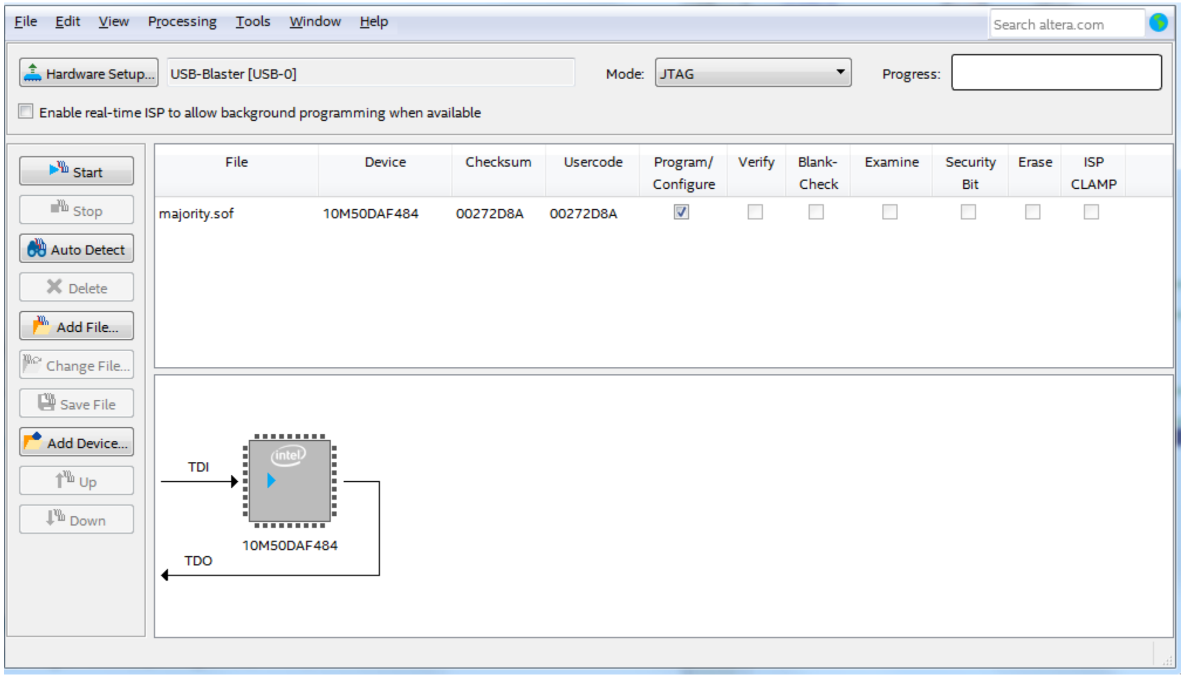


Figure 4. Excerpt from DE10-Lite Schematic Showing Active-High LEDs

5. Once you have entered your code, click Processing > Start Compilation. The design should compile with 0 errors.

6. Select Tools > Programmer. Connect your DE10-Lite board to your host computer with the USB cable. The Hardware Setup should be set to USB-Blaster[USB-0], as shown in Figure 5. Make sure the majority.sof file is listed and the Program/Configure box is checked as shown in Figure 5. Then click Start.



Hardware Setup... USB-Blaster [USB-0] Mode: JTAG Progress:

☐ Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine	Security Bit	Erase	ISP CLAMP
majority.sof	10M50DAF484	00272D8A	00272D8A	☑	☐	☐	☐	☐	☐	☐

Figure 5. Programmer Tool Configuration

You can now test the majority gate design on the DE10-Lite board using SW[0], SW[1] and SW[2] as inputs. When 2 of the 3 switch inputs are high, then the output, LEDR[1] should be on; otherwise, it should be off.

Verify that the majority gate design works as expected.

D. Simulating in ModelSim-Intel Using a Testbench

An enormously powerful debugging technique is to simulate a design using either functional or timing simulation. This is generally a better way to debug a design than simply downloading and testing because in a functional simulation you can see internal signals as well as I/O signals. We will start by doing a functional simulation of the majority gate design using a testbench. Although it is possible to manually give stimulation commands or draw waveforms, the best approach is to write a testbench program in Verilog to generate the stimulation inputs and monitor the outputs.

An example testbench for the majority gate module is shown in Figure 6. This testbench code has been adapted slightly from the code given in Figure 3.8 (p. 50) in your text. The changes were needed only because the names of the I/O signals are different for the DE10-Lite board than the signals used in the text.

```
module tb_majority;

      reg [2:0] count;

      wire  [7:0] HEX0;
      wire  [7:0] HEX1;
      wire  [7:0] HEX2;
      wire  [7:0] HEX3;
      wire  [7:0] HEX4;
      wire  [7:0] HEX5;
      wire  [1:0] KEY;
      wire  [9:0] LEDR;
      wire  [9:0] SW;

      assign SW[2:0] = count;

      majority UUT (.HEX0(HEX0), .HEX1(HEX1), .HEX2(HEX2),
                  .HEX3(HEX3), .HEX4(HEX4), .HEX5(HEX5),
                  .KEY(KEY), .LEDR(LEDR), .SW(SW));


      initial begin
         count = 3'b000;
         repeat (8) begin
            #100
            $display("in = %b, out = %b", count, LEDR[1]);
            count = count + 3'b001;
         end
      end
endmodule
```

Figure 6. Testbench Module – tb_majority.v

Note that the testbench Verilog code is **not** intended to be synthesized; it is for simulation only. Therefore, it can use non-synthesizable Verilog constructs such as time delay (#) and the initial and $display statements. The testbench instantiates the module to be tested, in this case the majority gate design, as a component named UUT (Unit Under Test) and provides stimuli for the inputs and monitors the outputs, In ModelSim, the Waveform viewer is a convenient tool for checking the proper operation of a circuit.

To perform a functional simulation using ModelSim, do the following:

1. Enter the **tb_majority.v** into a file and save it to the **lab1/test**/ subdirectory.

2. In the **lab1** directory, make sure you have created a folder for your simulation project. This subdirectory should be located under **simulation/** (i.e. **lab1/simulation/tb_majority/**).

3. Run ModelSim from the start menu or a desktop icon. (Do not launch it from within Quartus Prime since the testbench will not be part of such a project.)

4. Select File > New> Project to open the Create Project dialog box. Give the project a name and specify the project location as the subdirectory that you created above. Leave the other settings in the default state. (i.e. Default Library Name = work, Copy Settings From = Copy Library Mappings). Click OK.

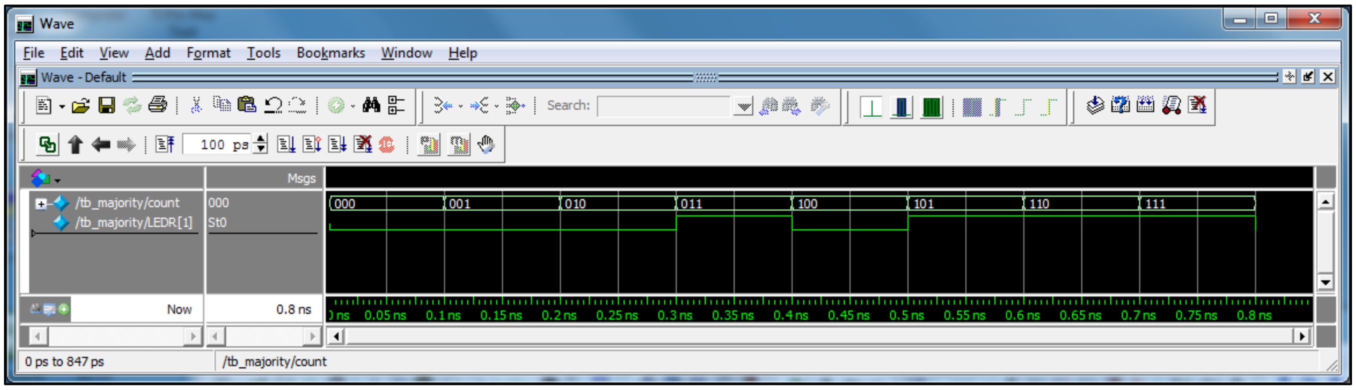
5. Next an Add Items to Project Window opens. Add both your **lab1/test/tb_majority.v** and your **lab1/synthesis/majority/majority.v** files. Leave the open set to “Reference from current location”.

6. Select Compile > Compile All. Your project should be compiled without errors.

7. Click Simulate > Start Simulation. Select Design > work > tb_majority

8. To view a simulation waveform, click View > Wave. Then in the Objects window, select count, right click on it and select Add Wave. This will add the count signal to your Wave window. Next expand the LEDR signal, select LEDR[1] and add it to the Wave window. This way you can see the waveform for the input and output signals of the majority gate.

9. You can run the simulation by selecting Simulate > Run > Run 100 or by simply typing run in the command window. This will run your simulation for 100 time units, which is 100 ps by default. See Figure 7 for a screenshot of the Wave window after simulating for 800 ps.


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
100 ps
Msgs
/tb_majority/count
/tb_majority/LEDR[1]
000
St0
000
001
010
011
100
101
110
111
Now
0.8 ns
0 ns 0.05 ns 0.1 ns 0.15 ns 0.2 ns 0.25 ns 0.3 ns 0.35 ns 0.4 ns 0.45 ns 0.5 ns 0.55 ns 0.6 ns 0.65 ns 0.7 ns 0.75 ns 0.8 ns
0 ps to 847 ps
/tb_majority/count
Figure 7. Functional Simulation Waveform for Majority Gate Design

You should also see output in the ModelSim console window from the $display command in the testbench. The console output should be a text version of the waveform output and should be the same as in Figure 3.9 in your text and shown in Figure 8.

```
# in = 000, out = 0
# in = 001, out = 0
# in = 010, out = 0
# in = 011, out = 1
# in = 100, out = 0
# in = 101, out = 1
# in = 110, out = 1
# in = 111, out = 1
```

Figure 8. ModelSim Console Output

Part I – Implementing Basic Combinational Logic Gates (40 pts)

Following the example given in Part 0, create the subdirectories **lab1/synthesis/partI** and **lab1/simulation/tb_partI**. Then, create a new project using SystemBuilder (this time called **partI**) and implement some logic gates of your choice using the switches, buttons, LEDs and 7-segment displays on the DE10-Lite board. Download your design and verify proper operation.

There are many possible design options. A few examples are 2-, 3- or 4-input AND, NAND, OR, NOR gates. You could also use a pushbutton to directly control an LED or segment of a 7-segment display. Note that the segments of the 7-segment display are active-low, as shown in Figure 9. Also note that bit 7 corresponds to the decimal point. For example, in order to permanently turn off all the segments of the HEX0 7-segment display, you could use the following assign statement:

```
assign HEX0 = 8'b11111111;
```

You can verify experimentally or from the DE10-Lite board schematic that the push-button switches are active-low.

Design a "testbench" verilog module (save it to **lab1/test/tb_partI.v**) which instantiates a copy of your module, exercises the circuit through all possible input combinations, and prints inputs and outputs alongside each other in a format that is easy to read.

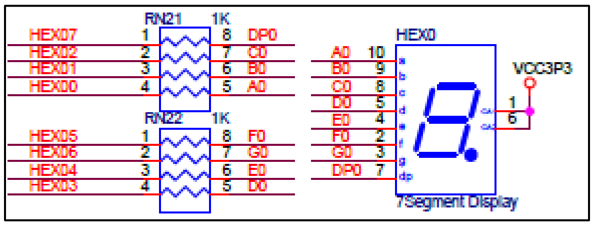


Figure 9. Excerpt from DE10-Lite Schematic Showing Active-Low 7-Segment Display. Active low operation is visible because the common nodes on pins 1 and 6 are tied to Vcc.

Once your design works on the DE10-Lite board, demonstrate it to your TA for checkoff.

Part II – Implementing a Decimal Display for 4-bit Switch Inputs (60 pts)

In this part, you will design a simple combinational circuit so that the value of the 4-bit switch input, SW[3] – SW[0], is displayed in decimal on HEX[1] and HEX[0]. Thus, your display will show the values 00, 01, 02, … 09, 10, 11, 12, 13, 14, 15 depending on the positions of SW[3]-SW[0]. The other 7-segment displays (HEX5-HEX2) segments should be off. All decimal point segments should be off.

As design complexity increases, hierarchical design practices become necessary to simplify HDL development and debug. For very simple designs, such as the circuits constructed in Part 0 and Part I, placing all design logic into the top-level file generated by SystemBuilder is not an issue.

In this part, you will implement your decimal display using hierarchical design. You will design a separate (reusable) module implementing the decimal 7-segment decoder logic. Then, you will instantiate this module at the top-level.

Go through the following steps:

1. Write a truth table for the output functions (i.e. the segments on the HEX1 and HEX0 displays) based on the 4-bit input (SW[3]-SW[0]).

2. Hint: most of the segments for HEX1 will either be constants or shared.

3. Draw a Karnaugh map for each output function.

4. Use the Karnaugh map to generate logic equations for each output.

5. Following the example given in Part 0 and Part 1, create a new project named **partII**. Be sure to save the SystemBuilder output files to **lab1/synthesis/partII**.

6. Implement your logic equations using Verilog **assign** statements.

7. Write these assign statements in a new module, saved in a new file (i.e. **lab1/hdl/dec_7seg_decoder.v**). The template for this module is provided in Figure 10.
   - See Appendices B and C for details on how to add, remove, and move Verilog design files.

8. Instantiate your decimal 7-segment decoder module at the top level (i.e. within **lab1/synthesis/partII.v**). Note that placing the decimal 7-segment decoder logic into a new module allows this module to be reused in later projects without any modifications.

9. Assign all unused I/O at the top-level (i.e. HEX2-HEX5).

10. Design a "testbench" Verilog module (save it to **lab1/test/tb_partII.v**) which instantiates a copy of your module, exercises the circuit through all possible input combinations, and prints inputs and outputs alongside each other in a format that is easy to read. Be sure to create the **lab1/simulation/tb_partII/** subdirectory before simulating the testbench.

11. Compile your Verilog design and test it on the DE10-Lite board.

Later, you will learn other ways to describe combinational logic in Verilog, such as using a **case** statement to directly implement a truth table without solving for the logic equations.

Demonstrate your working design on the DE10-Lite board to your TA for verification.

```
module dec_7seg_decoder
(
     input [3:0]
          bin,
     output [7:0]
          hex0,
          hex1
);

// Your logic here

endmodule
```

Figure 10. Module template for a 4-bit decimal 7-segment decoder

Submission (100 pts total)

Submit the following items for verification and grading:

- Report containing
  - Page1: Lab cover sheet with TA verifications for
    1) Part I – HW demonstration & testbench
    2) Part II – HW demonstration & testbench
  - Printed outputs from testbenches showing inputs alongside outputs for Parts I and II
  - Truth table, Karnaugh maps and output equations for your Part II design
- Verilog design/testbench files
  - partI.v, tb_partI.v
  - partII.v, tb_partII.v
  - dec_7seg_decoder.v

Appendix - Quartus Tips

A. Using external text editors

Depending on your preferences, you may want to use an external text editor to create and edit HDL files. Several popular text editors provide support for Verilog/SystemVerilog either by default or using extensions (ex. VSCode, Sublime Text, Notepad++, VIM, Emacs). After setting up your preferred editor you can tell Quartus to open files using that text editor by performing the following steps:

1. Navigate to Tools > Options.
2. Select the Preferred Text Editor category.
3. Select Custom from the text editor drop-down.
4. Browse to the executable location for your preferred editor.
5. You will need to add %f (and potentially some command line switches) after the executable as shown in Figure A1.

B. Adding new/existing Quartus Project design files

Throughout this lab, SystemBuilder was used to create the Verilog file containing the top-level module (i.e. top-level design entity) for each project. It is good practice to place additional design modules (ex. **dec_7seg_decoder.v**) in a separate file within the **hdl/** folder.

To **create a new Verilog file** in Quartus perform the following steps:

1. Navigating to File > New (or CTRL+N)
2. Select Verilog HDL File.
3. Be sure to save all HDL design files to **hdl/**

To **add existing files** to a Quartus Project perform the following steps:

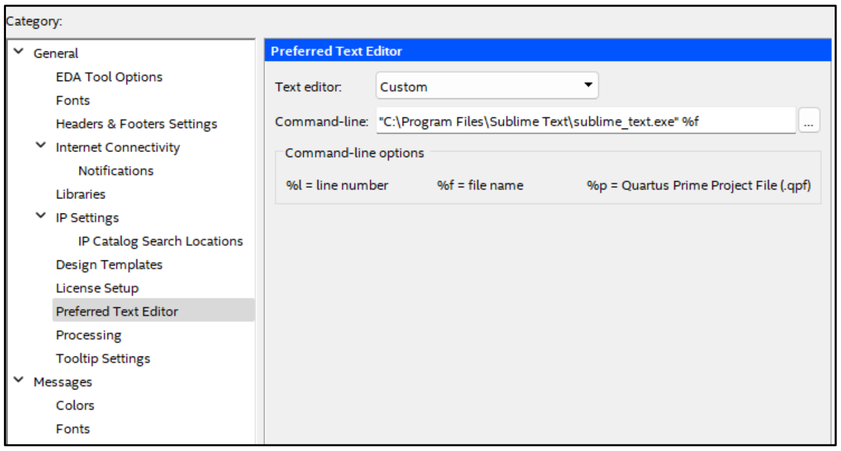
1. Be sure the file has been saved or copied to the hdl/ folder
2. In Quartus, navigate to Project > Add/Remove Files in Project…
3. Browse for your file by clicking …
4. Click Add
5. Click OK

C. Removing/moving Quartus project design files

It is important to note that removing files from Quartus projects does not remove them from your system. Either removing or moving a file included in a Quartus project requires that the file is first removed from the project:

1. In Quartus, navigate to Project > Add/Remove Files in Project…
2. Select the file
3. Click Remove
4. Click OK

After removing the file from the project, you can either delete it using the Windows File Explorer or move it to a new location and add the file back to the project (see adding existing files in Appendix B).



Text editor: Custom

Command-line: "C:\Program Files\Sublime Text\sublime_text.exe" %f

%l = line number   %f = file name   %p = Quartus Prime Project File (.qpf)

Figure A1. Configuring Sublime Text as the default text editor.

D. Setting top-level design entity

SystemBuilder will automatically configure the top-level design entity for your project. However, it is possible that you accidentally set a different module as the top-level design entity. This will result in a design that has no pin connections (i.e. no I/O), causing Quartus to optimize away all logic. This causes hard-to-find bugs because the tool will not indicate any errors! To check and select the top-level design entity perform the following steps:

1. Navigate to Assignments > Settings
2. Click the General category and verify that your top-level design entity is correct (ex. majority would be the top-level design entity for Part 0).
3. If not change to the correct design entity and click OK