

## Lab 4: 7-Segment Display

*Digital Design and Computer Architecture: RISC-V Edition (Harris & Harris, Elsevier © 2021)*

### Objective

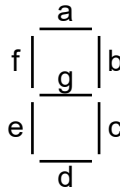
The purpose of this lab is to learn to design a combinational circuit using **behavioral** SystemVerilog, write your own self-checking testbench, and debug your design.

In this lab may you will design a circuit that implements a 7-segment display decoder. You will write a testbench to test the circuit and then implement it on your DE0-CV FPGA board. Remember that testbenches are only used in simulation.

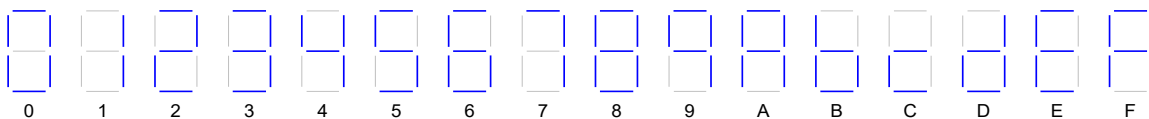
### 1. 7-Segment Displays

Many digital devices use seven-segment displays to represent numbers. Some examples include digital clocks and speedometers. One or more of the seven segments light up to display the desired number. In this lab, you will construct the seven-segment display decoder. It is called a decoder because it takes the 4-bit input (that indicates what value to display) and decodes them into seven outputs.

You will now design and simulate a seven-segment display, as shown in Figure 1. Each of the seven segments is labeled a through g. The numbers 0 through F light up the segments shown in Figure 2. For example, the number 0 lights up all but the middle segment, segment g.

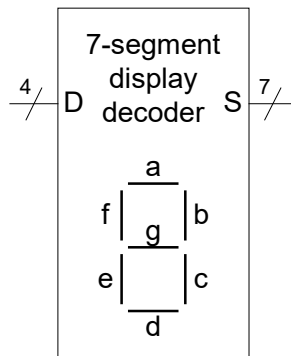


**Figure 1. Seven-segment display**



**Figure 2. Seven-segment display function**

You will build a seven-segment display decoder, shown in Figure 3. The circuit has four input bits,  $D_{3:0}$  (representing a hexadecimal number between 0 and F), and produces seven output bits,  $S_{a:g}$ , that drive the seven segments to display the number. In SystemVerilog, we will name the outputs segments[6:0]. A segment of the display turns on when it is 0. Such an output is called a *low-asserted output* or *active low output*.



**Figure 3. 7-segment display decoder**

To design your seven-segment display decoder, you will first write the truth table specifying the output values for each input combination. We have started the truth table for you in Table 1. For example, when the input is  $D_{3:0} = 0000$ , all of the segments except  $g$  should be on. Because the outputs are active low, they must be  $S_{g:a} = 1000000$ . Complete the truth table for the 7-segment display decoder circuit. You will need to turn in your completed truth table.

**Table 1. Truth table for 7-segment display decoder**

Hexadecimal Digit	Inputs				Outputs							(in hex)
	$D_3$	$D_2$	$D_1$	$D_0$	$S_g$	$S_f$	$S_e$	$S_d$	$S_c$	$S_b$	$S_a$	
0	0	0	0	0	1	0	0	0	0	0	0	40
1	0	0	0	1								
2	0	0	1	0								
3	0	0	1	1								
4	0	1	0	0								
5	0	1	0	1								
6	0	1	1	0								
7	0	1	1	1								
8	1	0	0	0								
9	1	0	0	1								
A	1	0	1	0								
B	1	0	1	1								
C	1	1	0	0								
D	1	1	0	1								
E	1	1	1	0								
F	1	1	1	1								

Now implement your design using SystemVerilog. You may implement your design using combinational logic ( $\&$ ,  $|$ ,  $\sim$ ,  $\wedge$  operators) or a case statement inside an always block. It is your choice. Name the file sevenseg.sv Your module declaration should be:

```
module sevenseg(input logic [3:0] data,  
                output logic [6:0] segments);
```

## 2. Testbench and Simulations

Now write a testbench and testvector file for your 7-segment display decoder. Name them `testbench_sevenseg.sv` and `sevenseg.txt`, respectively. **Hint:** For your testvector file, write the values in hexadecimal, with the outputs listed first. This will make it easier to assign to the inputs and expected output. For example, the first testvector (corresponding to the first row in the truth table) would be:

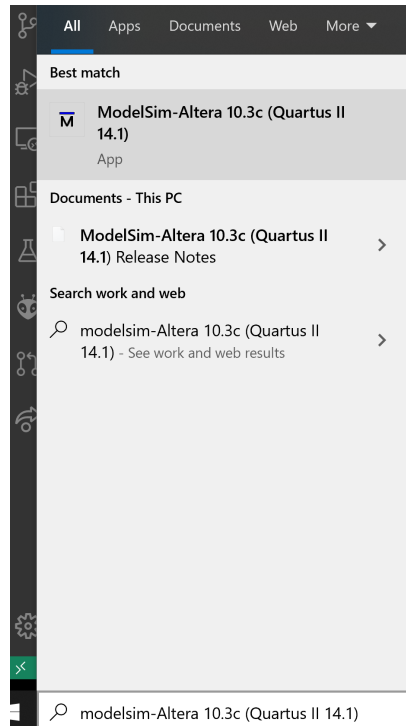
```
40_0 // segments_data
```

Simulate your design in ModelSim using the testbench and testvector file.

You **must always simulate your design before** implementing it in hardware on the FPGA.

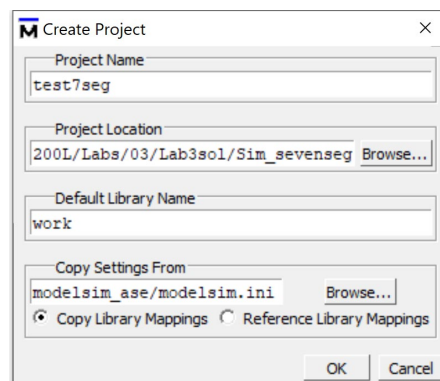
Display the inputs and outputs, from top to bottom on the waveform, in hexadecimal. Take a screenshot of the simulation with the inputs being set in counting order (0, 1, 2, etc.). To display the inputs and outputs in hexadecimal, right-click on the signal (after you have dragged them over the Wave pane) and select Radix -> Hexadecimal.

To simulate the testbench, open ModelSim from the Start menu.



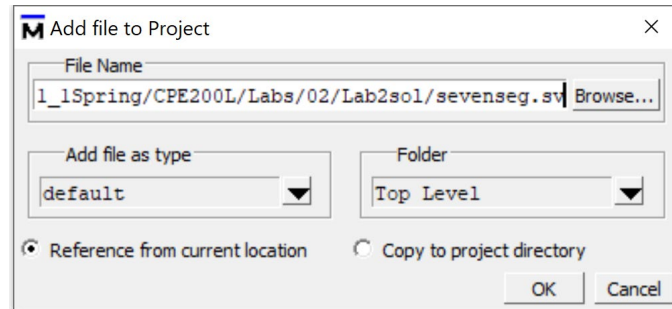
**Figure 4. Open ModelSim**

Now create a new ModelSim project as you did in Labs 2 and 3. Name the project test7seg and select a Project Location. Click OK.



**Figure 5. Name Project**

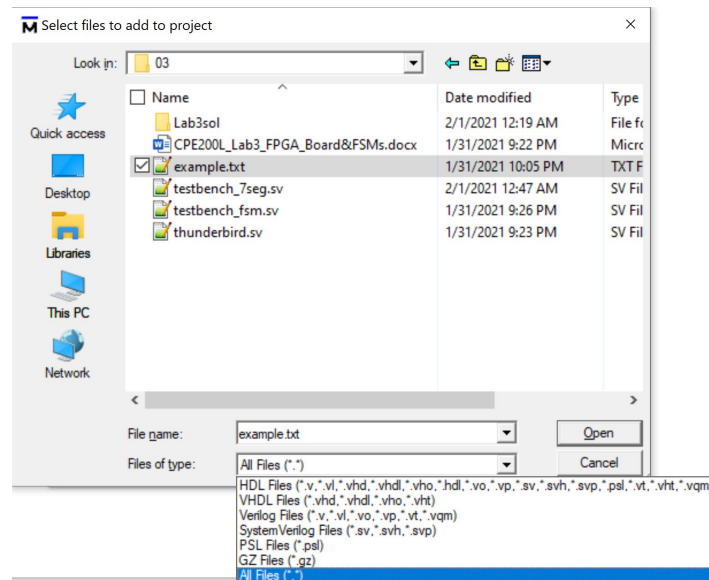
Click on Add Existing File, and add sevenseg.sv and testbench\_sevenseg.sv. Add these files by reference by keeping the **Reference from current location** radio button, as shown below in **Figure 6**. Then click OK. (If you clicked on Copy to project directory, ModelSim would make a copy of the file in the project directory. You'd end up with multiple copies and have a hard time keeping of track of which one you're actually working on.)



**Figure 6. Add sevensseg.sv and testbench\_7seg.sv to ModelSim project**

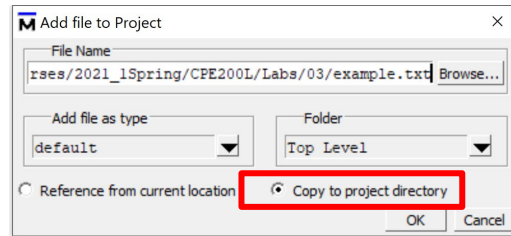
Now you will add the testvector file. This is an exception where you must add it by copying it to the project directory. Otherwise ModelSim can't find it during simulation.

So, click on Add Existing File again and browse to where you placed example.txt (that you downloaded from Canvas – you'll need to click on All Files under Files of type – as shown below in **Figure 7**). Then click Open.



**Figure 7. Add example.txt to ModelSim project**

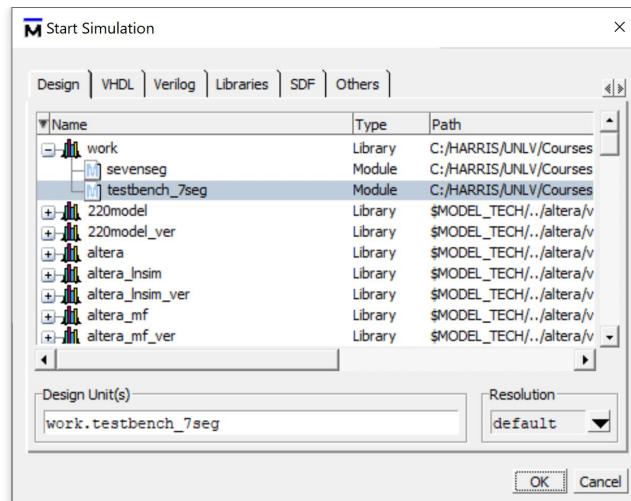
This time – as an exception – you will copy the file to the project by clicking on the **Copy to project directory** radio button, as shown below in **Figure 8**. Then click OK.



**Figure 8. Add example.txt to ModelSim project – copy to directory**

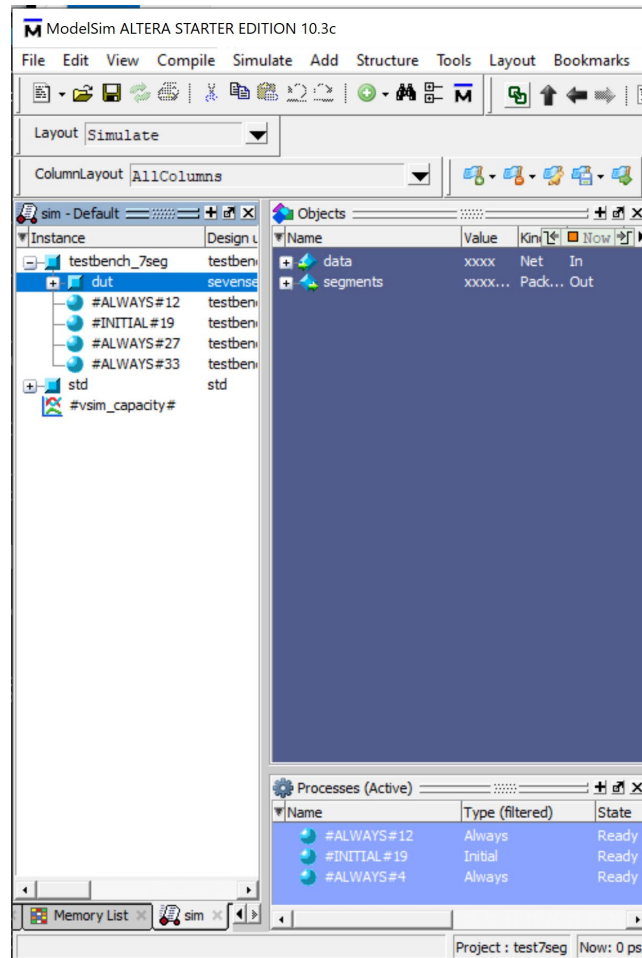
Then close the Add items to the Project window. Compile the files. If any errors occur, view the Transcript window for errors and warnings and fix the SystemVerilog files.

Now you will simulate the testbench (**not** the sevenseg module). Choose Simulate → Start Simulation from the top File menu. Then expand the work folder and select testbench\_7seg (**not** sevenseg) and click OK (see **Figure 9**).



**Figure 9. Simulate testbench\_7seg**

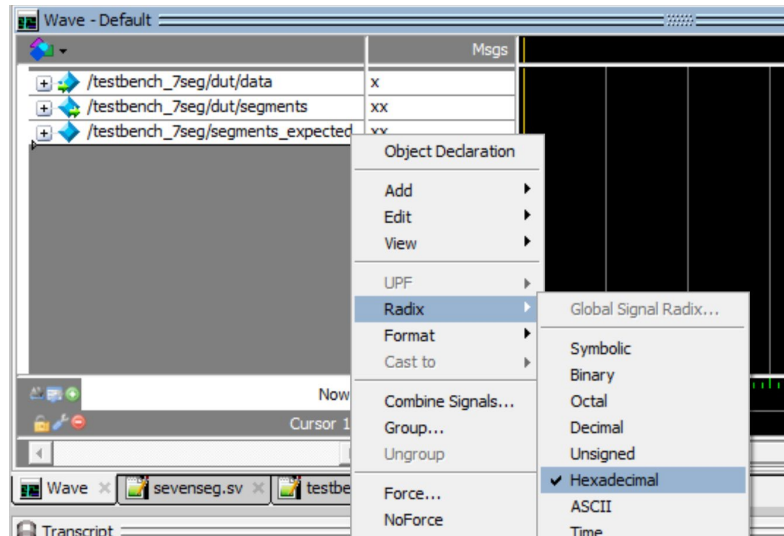
Now in the sim pane you will see the hierarchy of modules, with module testbench\_7seg instantiating the dut (device under test) – see **Figure 10**. This module only has two levels of hierarchy: testbench\_7seg and dut. But as your projects get bigger you'll have more complex project structures. You can use the sim window to navigate around the hierarchy of modules and to dive deeper into the hierarchy to grab signals you'd like to view as you run the simulation.



**Figure 10. View hierarchy and module signals**

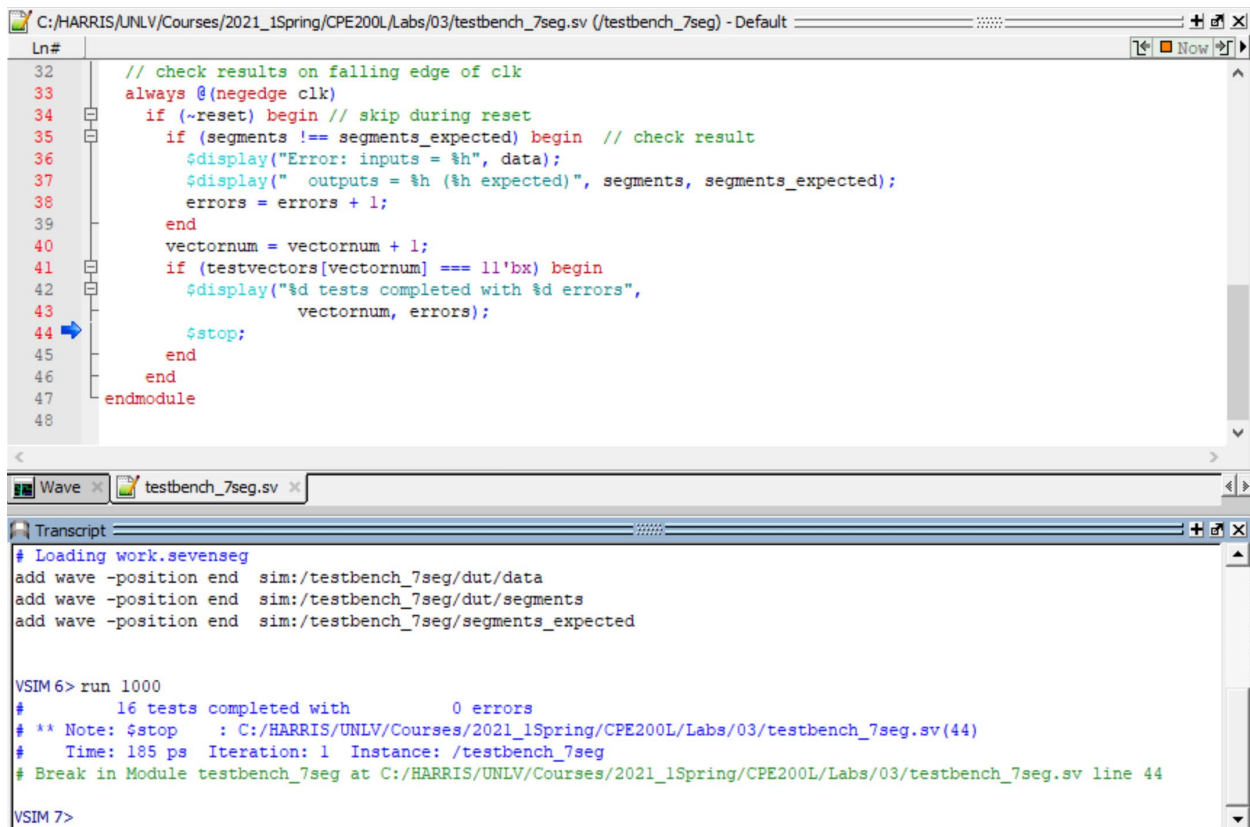
Click on the dut and copy all of that module's signals (data and segments) to the Wave pane. You may need to click on the Wave pane to make it come to the front. If you don't see a Wave pane at all, click on View → Wave in the File menu.

Click on testbench\_7seg in the sim pane and also copy segments\_expected to the Wave pane. Now right-click on all of the signals in the Wave window and change their radix is hexadecimal (see **Figure 11**).



**Figure 11. View signals as hexadecimal values**

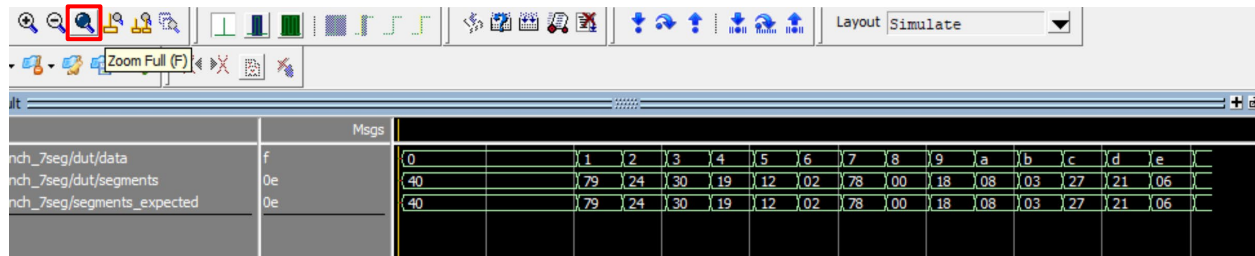
When the simulation completes, view the Transcript pane. It should say that 16 tests completed with 0 errors (see **Figure 12**). You'll also see where simulation stopped in the testbench file (testbench\_7seg.sv) at the \$stop command.



**Figure 12. run simulation for 1000 simulation ticks**



Also view the Wave window. With the Wave window active, click on Zoom Full (see **Figure 13**) to view the full waveform. Use the adjacent Zoom In and Zoom Out buttons to view the waveform as desired. You should see that the output from the module (segments) is the same as the expected output from the testvector (shown in the segments\_expected signal).

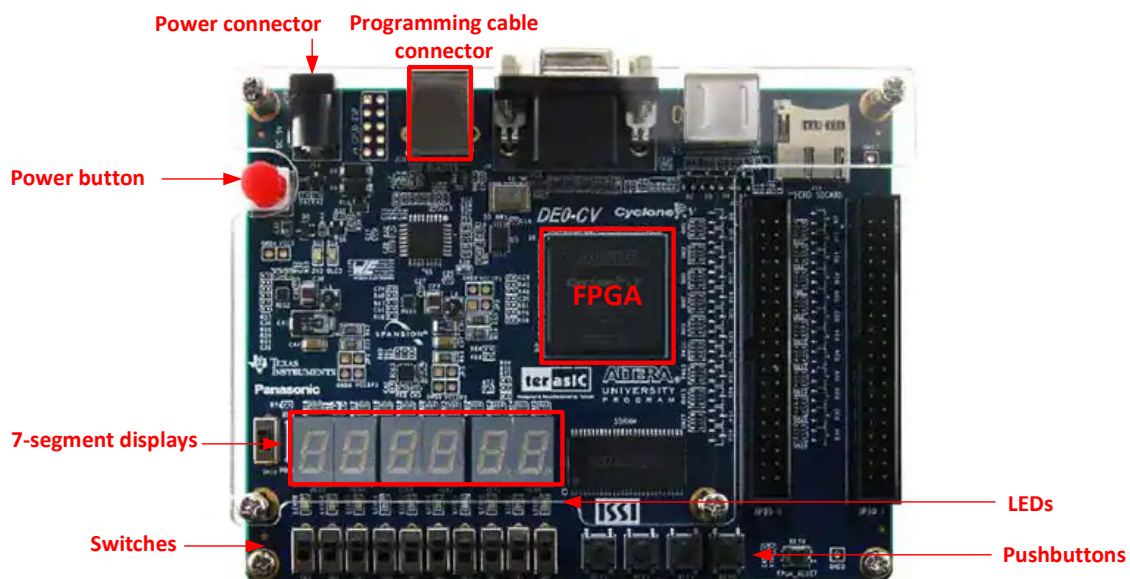


**Figure 13. Simulation waveform**

Once you're done viewing the waveform, close ModelSim.

### 3. Test 7-Segment Display Decoder in Hardware

Now you are ready to test your 7-segment display decoder in hardware. You will be using the DE0-CV board. The board, as shown in **Figure 14**, has an FPGA that drives the peripherals, which include switches, LEDs, pushbuttons, and 7-segment displays. Also notice the location of the programming cable and power connectors and the power button on the top left of the board.



**Figure 14. DE0-CV FPGA board and peripherals**

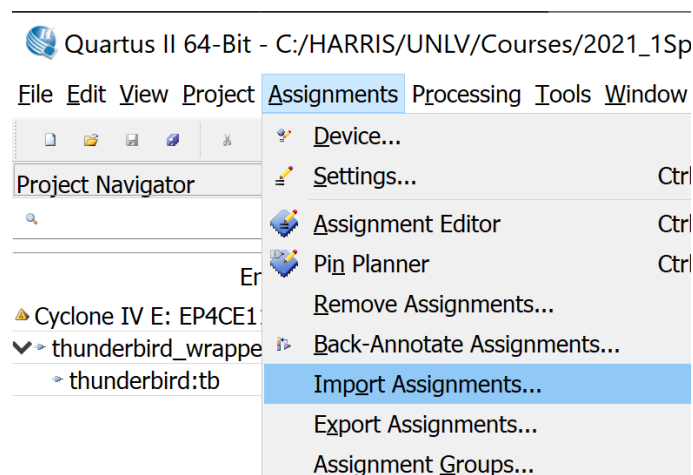
You will be programming the FPGA with your 7-segment display decoder, and you will drive the right-most 7-segment display. You will need to map your module's signals to the FPGA pins that are physically connected to the switches (named SW) and the right-most 7-segment display (named HEX0) on the DE0-CV board. To do so, create a wrapper module called `sevenseg_wrapper` that has inputs `SW[3:0]` and outputs `HEX0[6:0]`. This wrapper module should instantiate your `sevenseg` module and connect `SW[3:0]` to `data[3:0]` and `HEX0[6:0]` to `segments[6:0]`.

Open Quartus. Create a new project as you did in Labs 2 and 3. Name the project `sevenseg_wrapper`. In Quartus, the top-level module and the name of the project are the same. Add `sevenseg.sv` and `sevenseg_wrapper.sv` to the project.

Remember to select the correct target FPGA: Select **Pin Count** → **484**, then **Device** → **Cyclone V E Base**; this will greatly reduce the choices. Click **5CEBA4F23C7** in available devices and click next.

### Import Pin Assignments

Import the pin assignments from the Quartus Settings File (`de0_cv.qsf`) by clicking on **Assignments** → **Import Assignments**, as shown in Figure 15. The `de0_cv.qsf` file lists all of the connections between the FPGA pins and the peripherals (switches, LEDs, 7-segment displays, etc.).



**Figure 15. Importing pin assignments**

In the next window, browse to where you have placed the `de0_cv.qsf` file. Select that file (click on it) and click Open.



After compilation/synthesis is complete, the bottom window in Quartus II should say the Full Compilation was successful. Note that you will see some warnings that you can ignore: for example, “Timing requirements not met” (there were no timing requirements), “Ignored locations or region assignments to the following nodes” (if you expand that warning, you will see all the pin assignments/signals that were not used in your design but that were in the `.qsf` file). But not all warnings can be ignored (for example, if an inadvertent latch were created). If there are errors in your compilation, look at the errors (in the window at the bottom of Quartus – scroll up if needed) and fix them. Then recompile.

Now you are ready to download your design onto the FPGA board. Make sure that your DE0-CV board is connected to your computer and powered on. Then program your board as you have done in previous labs.

Now toggle the right-most switches and watch the digit on the 7-segment display change.

## 4. What to Turn In

1. Please indicate how many hours you spent on this lab. This will be helpful for calibrating the workload for next time the course is taught.
2. Completed Table 1.
3. SystemVerilog modules: `sevenseg.sv`, `sevenseg_wrapper.sv`
4. Testbench module and test vector files (`testbench_sevenseg.sv` and `sevenseg.txt`).
5. Simulation waveforms showing the inputs (data) and outputs (segments) in that order, from top to bottom, and in hexadecimal. Do they pass your testbench and match expectations?
6. RTL Viewer schematics of your `sevenseg_wrapper.sv` and `sevenseg.sv` modules (two schematics). Do they match your expectation?

Please indicate any bugs you found in this lab manual, or any suggestions you would have to improve the lab.