ECE 270 Lab Verification / Evaluation Form

Experiment 9

Evaluation:

IMPORTANT! You must complete this experiment during your scheduled lab period. All work for this experiment must be demonstrated to and verified by your lab instructor *before the end* of your scheduled lab period.

STEP	DESCRIPTION	MAX	SCORE
Step 1	Original source file functionality verification	1	
Step 2	Verilog module `up_down_counter`	5	
Step 3	Bouncy switch investigation using `up_down_counter`	2	
Step 4	Independently clocked `up_down_counter` pair	2	
Step 5	Verilog module `bintohex` and display mode control	6	
Step 6	Verilog module `ring_counter`	5	
Step 7	Multi-mode light sequencer instantiation and verification	2	
Step 11	Thought Questions	2	
	TOTAL	25	

Signature of Evaluator: _____

Academic Honesty Statement:

"In signing this statement, I hereby certify that the work on this experiment is my own and that I have not copied the work of any other student (past or present) while completing this experiment. I understand that if I fail to honor this agreement, I will receive a score of ZERO for this experiment and be subject to possible disciplinary action."

Last Name (Printed): _____ **Lab Div:** ____ **Date:** _____

E-mail: _____@purdue.edu **Signature:** _____

Introduction to ispMACH 4256ZE Development Board

Instructional Objectives:

- To learn the basic features of a CPLD-based development board
- To learn how to realize state machines on a CPLD

Prelab Preparation:

- Read this document in its entirety
- Review the referenced Module 3 lecture material

See **Appendix A** of this document for instructions on how to program the CPLD board at your station using either **ispVM System** (integrated into ispLever) or **Diamond Programmer** (separate Lattice program).

Lecture/Demonstration:

Your lab instructor will give a brief presentation that includes the following:

- An overview of the ispMACH 4256ZE
- An overview of ispLEVER procedures for loading and testing compiled HDL programs

Experiment Description:

In this experiment, you will implement and test some basic Verilog modules that illustrate how the switches and LEDs on the ispMACH 4256ZE development board are interfaced to the CPLD. A skeleton file containing the I/O pin declarations ("top template") is provided on the course website, along with an outline of the code modules that need to be written.

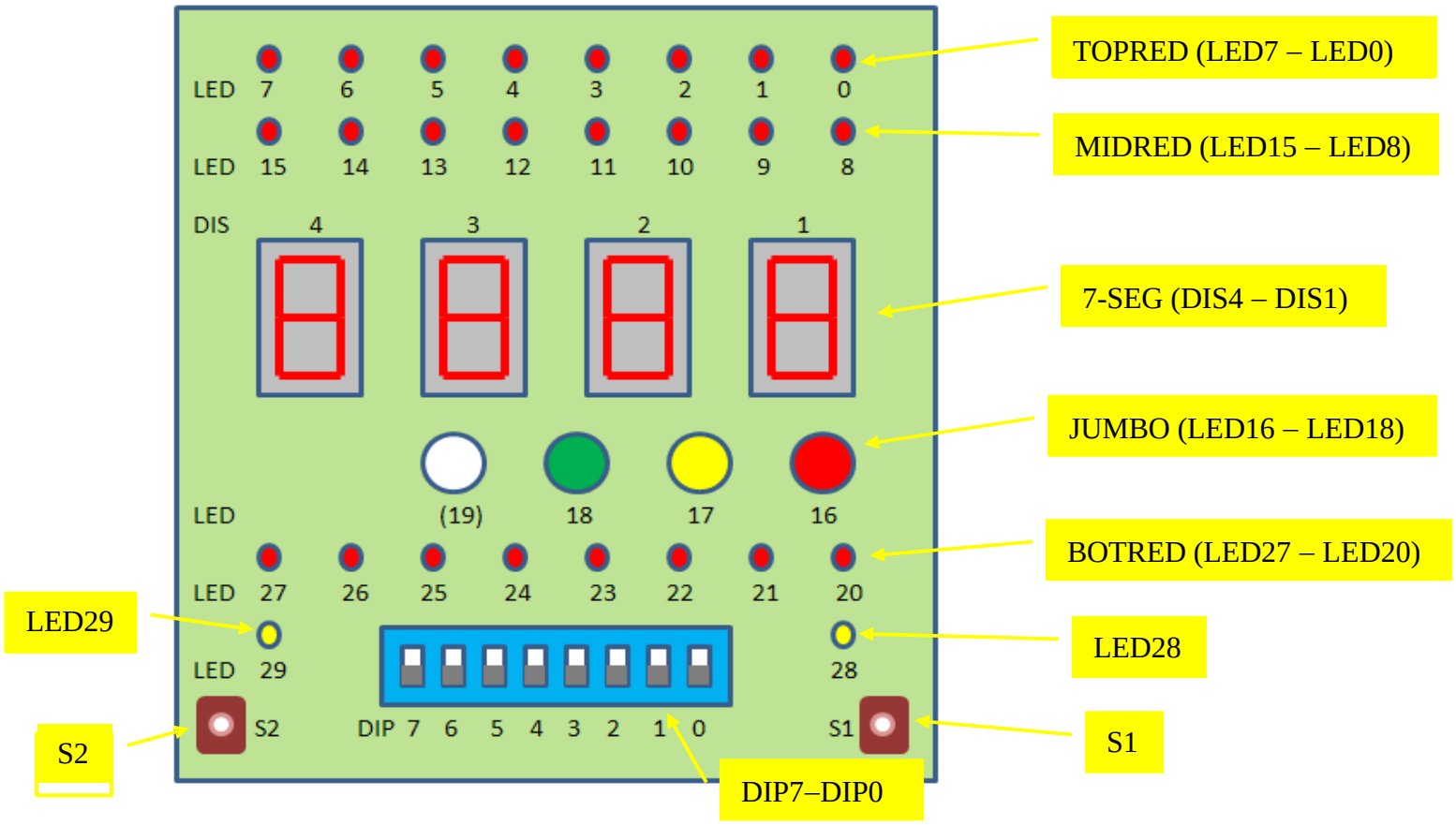


Figure 1. Lattice 4256ZE CPLD Development Board.

Step (1):

Carefully examine the **lab9_top_template.v** file provided for this experiment on the course website. Rename this file *myuserid*_**lab9.v** (where *myuserid* is your 8-character login name) and insert your identifying information where indicated. Next, open ispLever and create a new Verilog project named **Lab9**, select **LC4256ZE-5TN144C** as the device to use, and import your *myuserid*_**lab9.v** source file. Click on "Fit Design" to compile the program, then follow the instructions in Appendix A (at the end of this document) to load the JEDEC file created by ispLever into the CPLD. Verify that `DIP[7]` enables/disables the display of **GoPU** on the four 7-segment displays, and that pressing either pushbutton lights the yellow LED next to it. Also confirm that the state of the individual DIP switches is indicated on the bottom row of red LEDs.

Step (2):

First, comment out the "always" block used in Step 1. Then, write a Verilog module that realizes a 4-bit binary up/down counter. Instantiate two versions of your counter in the top-level module: one with its output routed to `countL` clocked by the internal oscillator output, `tmr_out`; and the other with its output routed to `countR` clocked by the divided clock output, `tim_div`. Set the "count direction" for both counters to "up" (`dir` = 1) and use `DIP[0]` as the asynchronous reset (`rst`) for both counters. Verify that both counters work as expected, and that `countL` increments at twice the rate of `countR`. Also confirm that `DIP[1]` functions as the enable for the on-chip oscillator.

Step (3):

First, comment out the counter instantiations used for Step 2. Then, instantiate two new versions of your up/down counter in the top-level module: one with its output routed to `countL` clocked by the right pushbutton de-bounced output, `S1BC`; and the other with its output routed to `countR` clocked by the "raw" (inherently bouncy) switch contact, `S1_NO`. Set the "count direction" for both counters to "up" (`dir` = 1) and use `DIP[0]` as the asynchronous reset (`rst`) for both counters. Note that the counter outputs (`countL` and `countR`) are routed to the top row of red LEDs (`TOPRED`). Verify that both counters work as expected, and observe the bouncy behavior of the raw switch contact relative to the one that has been de-bounced. Also, note what happens if the asynchronous reset (`DIP[0]`) is asserted while the counters are being clocked.

Checkpoint: Demonstrate Step 3 to your Lab Instructor, including the "bouncy" switch behavior compared with the "de-bounced" behavior.

Step (4):

First, comment out the counter instantiations used for Step 3. Then, instantiate two new versions of your up/down counter in the top-level module:

- one configured as an up/down counter, with its count direction controlled by DIP[6] and its output routed to countL, clocked by the de-bounced left pushbutton (S2BC)
- one configured as an up (only) counter, with its output routed to countR and clocked by the de-bounced right pushbutton (S1BC)

Use `DIP[0]` as the asynchronous reset (`rst`) for both counters. Confirm that both 4-bit binary counters can be clocked independently using the left (S2) and right (S1) pushbuttons, respectively. These up/down counter instantiations will remain in place for all subsequent steps of this experiment.

Step (5):

Write a Verilog module that performs a conversion between 4-bit hexadecimal values and their corresponding 7-segment display code. This module (`bintohex`) should input a 4-bit binary value and convert it to the corresponding 7-segment display code. Next, instantiate two versions of your `bintohex` module: one that converts `countL` to `hexL`, and another that converts `countR` to `hexR`. Finally, write a procedural block that either displays **GoPU** on the four segment displays (when `DIP[7]` = 1); or, when `DIP[7]=0`, displays `hexL` on 7-segment display `DIS4` and `hexR` on `DIS1` (`DIS2` and `DIS3` should be blank when `DIP[7]`=0). Verify that your `bintohex` module and supporting control code operate correctly.

Checkpoint: Demonstrate Steps 4 and 5 to your Lab Instructor.

Step (6):

Write a Verilog module `ring_counter` that realizes a self-correcting 8-bit ring counter and route its outputs to the middle row of red LEDs (`MIDRED`). The behavior of a self-correcting ring counter is described on page 28 of the Lecture Summary notes. Use `tim_div` as the clocking source and `DIP[0]` as the asynchronous reset for this module. Verify that your ring counter functions as expected.

Step (7):

Finally, instantiate the `moorlsa_sd` Verilog module for the 4-mode light sequencer described in the class notes (included in the source file provided) and route the output variables to the `JUMBO` (green/yellow/red) LEDs. Use `tim_div` as the clocking source and `DIP[0]` as the asynchronous reset for this state machine. Use `DIP[3]` and `DIP[2]` to provide the "mode control" inputs `M1` and `M0`, respectively. Confirm that the light sequencer works as anticipated, and that all four state machines (up/down counter clocked using left pushbutton, up-only counter clocked using right pushbutton, ring counter clocked using internal oscillator, and light sequencer clocked using internal oscillator) coexist "peacefully" and operate independently.

Checkpoint: Demonstrate Steps 6 and 7 to your Lab Instructor.

Step (8): Write your answers to the following Thought Questions in the space provided**.**

1. Based on observed counter behavior in response to the "bouncy" switch contact in Step 3, estimate the average number of times the pushbuttons used on the demo board bounce each time they are pressed. Describe any factors that appear to influence the "bounciness" of the switch contacts.

   _____

   _____

2. What change would you make if you wanted to clock your state machine(s) even slower than the "divided" clock (derived from the internal oscillator) utilized in this experiment?

   _____

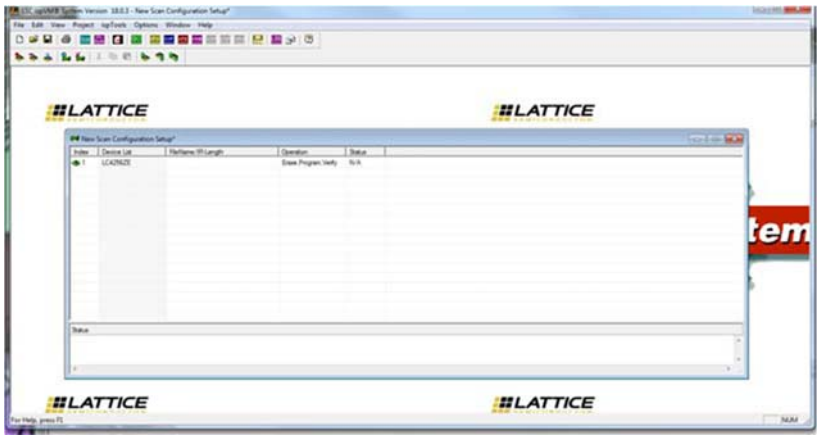
   _____

Appendix A: Programming the CPLD Board

There are two different tools that can be used to program the CPLD board: **ispVM System** (tool within ispLever) and **Diamond Programmer** (separate Lattice program).

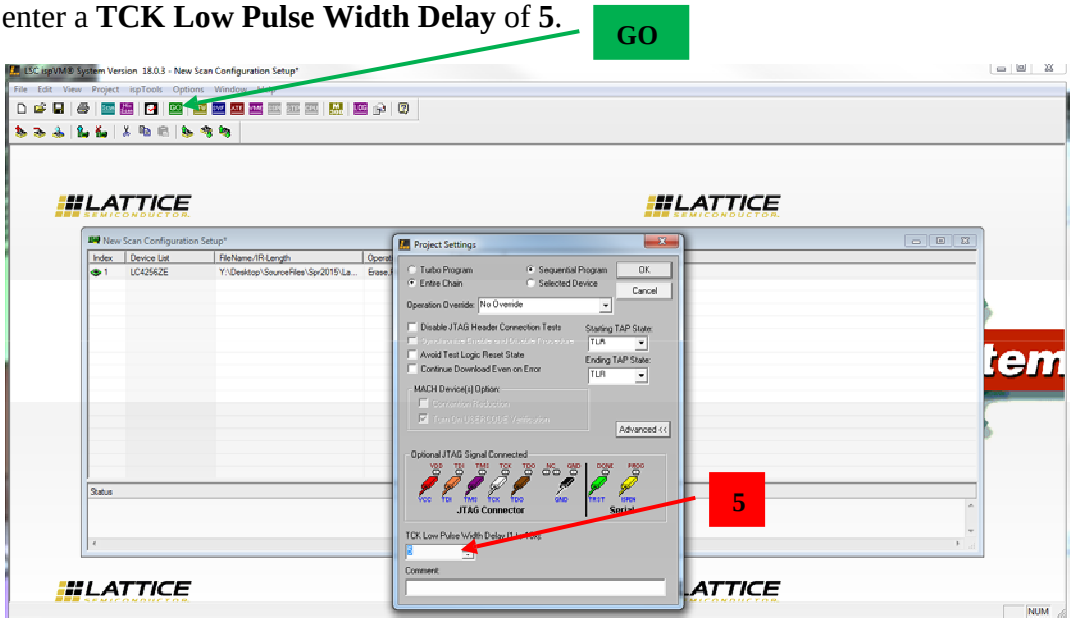
A. Using ispVM System

1. From the **ispLever** toolbar, select **Tools → ispVM System**.
2. Click on the **Scan** icon; **ispVM System** should find the CPLD.



NOTE: If a "device not found" error occurs when you click on the **Scan** icon, go to the **Options** pull-down menu and select **Cable and I/O port setup**. In the dialog box that appears, select **USB2** as the **Cable Type**, then click **OK**.

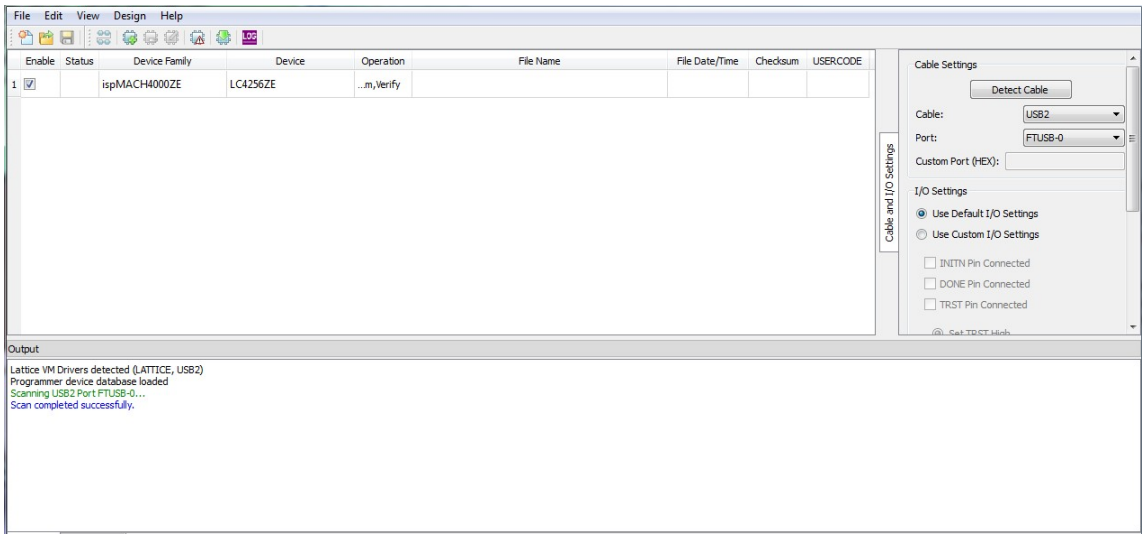
3. Click on the first line of the **New Scan Configuration Setup** box (that lists the LC4256ZE) to bring up the **Device Information** dialog box; browse for the JEDEC **(.jed)** file created by **ispLever** when you synthesized your Verilog module. Upon clicking OK, the filename will be filled in.
4. On the **ispVM System** toolbar, select **Project → Project Settings**; in the **Project Settings** dialog box, click on **Advanced**. In the extension to this dialog box that appears, enter a **TCK Low Pulse Width Delay** of **5**.



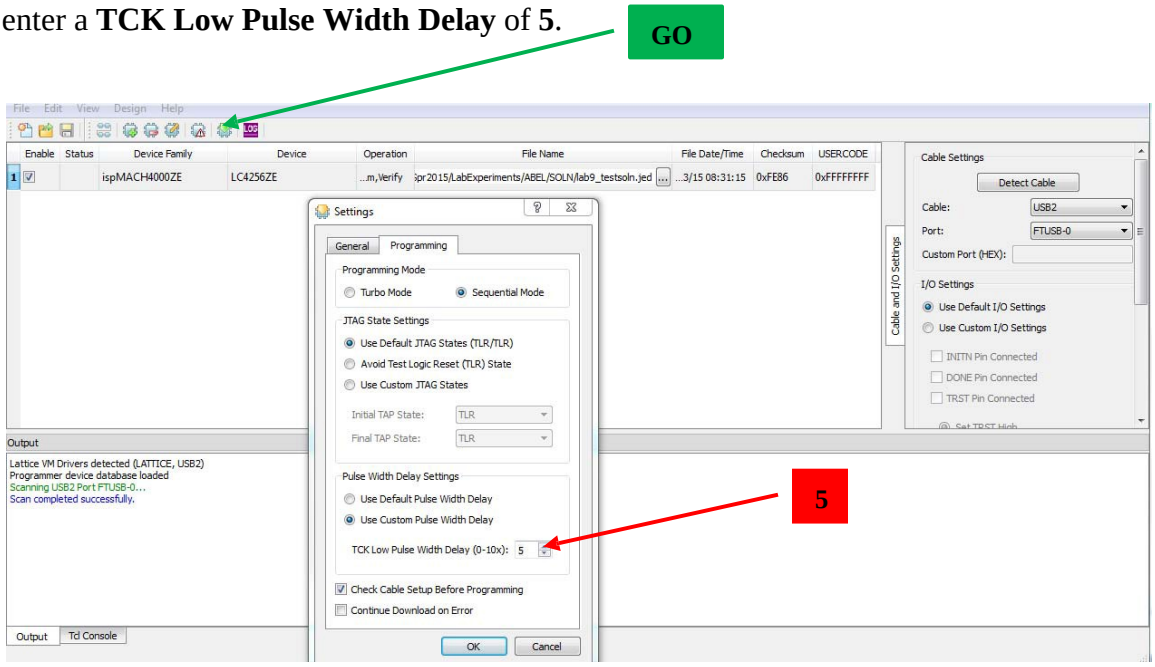
5. After clicking OK, you are now ready to program the CPLD, which can be accomplished by clicking on the **GO** icon (shown above).

B. Using Lattice Diamond Programmer

1. From the **Windows Program** menu, select **Lattice Diamond Programmer.**
2. On the **Getting Started** dialog box, select **Create a new Project from a Scan** then click on the **Detect Cable** button; after scanning, Diamond Programmer should find the CPLD board at your station.



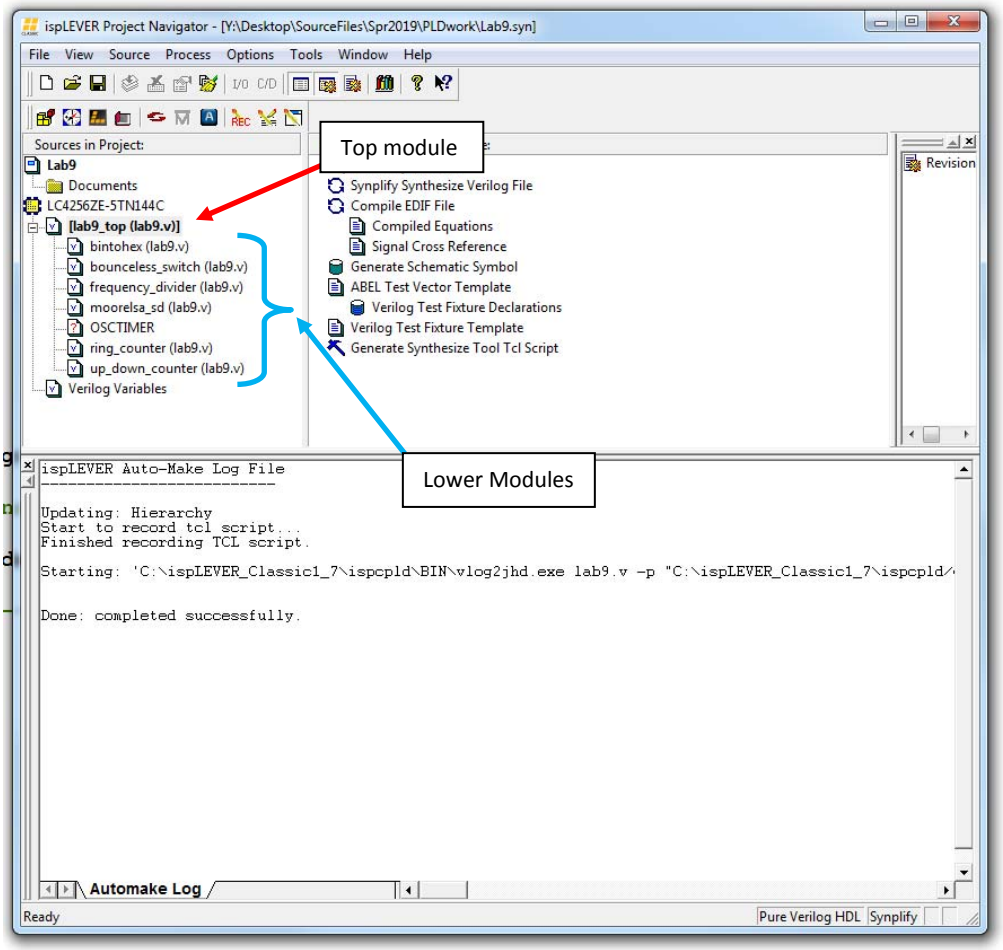
3. Next, click on the box under the **File Name** heading and browse for the JEDEC **(.jed)** file created by **isp_Lever** when you synthesized your Verilog module.
4. Next, go to **Edit → Settings** and select the **Programming** tab; under the heading **Pulse Width Delay Settings**, click on the **Use Custom Pulse Width Delay** radio button and enter a **TCK Low Pulse Width Delay** of **5**.



5. After clicking OK, you are now ready to program the CPLD, which can be accomplished by clicking on the **GO** icon (shown above).

C. Hierarchical Design and Adding Multiple Verilog Modules

1. The best way to describe hardware is by building your TOP level module with smaller lower level modules. This is referred to as *hierarchical design*.
2. Your Verilog description can have multiple lower level modules but should have only one TOP level module. All lower modules should be instantiated in the TOP module.
3. Modules and use of hierarchical design help in keeping your code organized and concise.
4. Your module hierarchy should appear as shown below when you are finished with this experiment.



NOTE: When large digital systems are designed, it is generally preferable to put the lower-level modules in separate files. For this course, however, place all of your lower-level modules in the **same** Verilog source file as your top-level module. This will facilitate on-line submission of the code you write for each experiment.

