SANTA CLARA UNIVERSITY

Fatemeh Tehranipoor, James Lewis, and Tokunbo Ogunfunmi **ELEN/COEN 21L**

Laboratory #7: Counters

For lab sections Monday-Friday Nov.8 – Nov. 13, 2020

I. OBJECTIVES

In this laboratory you will:

- Learn to use a comparator and a counter in a circuit
- Integrate several smaller designs to build a more complex circuit

PROBLEM STATEMENT

This lab is going to model some functionality that might be employed in a simple game that is intended to test a person's reactions. It will involve two counters that will move in opposite directions; one will count up, the other will count down. We will need to imagine that the numbers are displayed to the person playing the game.

If we had a physical implementation of this game, the player would have a button to control the counters. When the button is pressed, the two counters begin to count. And they continue to count as long as the button remains depressed. When the button is released, the counters stop counting. The objective of the game is to try to stop when both counters show the same value.

Unfortunately, we don't have a button to use here. But we will model this will an input to our design that we will call Stop. Stop will be the inverse of the button being pressed, i.e., when Stop is not asserted, the counters will count, and when Stop is asserted the counters will stop counting and a result will be generated.

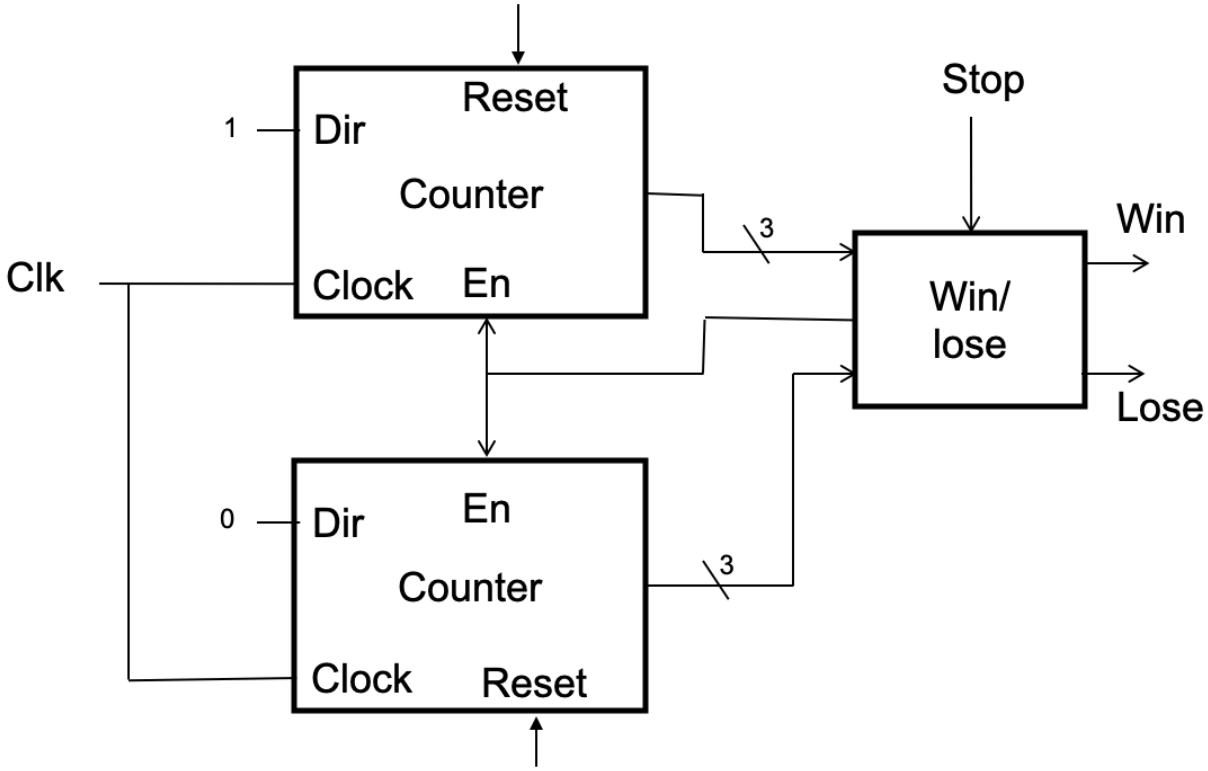
To indicate the results of the game, there will be two output signals:

- A "win" signal is asserted if the two counter values **match** when Stop is asserted.
- A "lose" signal is asserted if the two counter values **do not match** when Stop is asserted.
- While Stop is not asserted, and the counters are counting, neither the "win" nor the "lose" signal is asserted.

You might envision that this design could support multiple speeds. As the speed increases, the game becomes more challenging. But since we'll be doing this in simulation, we will not be concerned with the actual speed at which the circuit is running.

II. PRE-LAB

Here is a block diagram for the functionality we are going to implement:



There are only three inputs to this diagram: the clock, a signal called "Stop", and a reset signal.

1. **Design the counter.** Write a Verilog module for a 3-bit counter. This counter should only produce the values 1,2,3,4,5,6. Note the following
    a. In addition to clock and reset inputs, the module should have both an enable and a direction input
    b. If the enable is not asserted, the counter should hold its value.
    c. The direction input should control whether the counter counts up or counts down. You can define for yourself whether 1 means up or down. Note that, whether counting up or down, the count values should stay within the range of 1 to 6.

    You can design the counter however you like, applying any of the techniques you will have learned in the lecture section.

2. **Design the win/lose block**

Write a Verilog module *WinLose* that takes as inputs the two 3-bit count values as well as the Stop signal. There are three outputs from this module

- a signal to be used as a count enable for your counters
- explicit win and lose signals

The counters should only count when Stop is not asserted. The win and lose signals should only assert while Stop is asserted. And only win or lose should assert, not both. Winning is determined by checking to see if the two count values are the same.

Submit your Verilog code as your pre-lab assignment

III. PROCEDURE

1. **Reconcile your design choices with your lab partner**

   Compare the implementation choices that you made in your pre-lab with what your partner has done. Decide between yourselves which version of code you will use.

2. **Instantiate your components in a top level schematic.**
   a. Create a top level design and add the modules you developed in your prelab.
   b. Your TA will provide a framework for testing your design, which we call a testbench. It will be a means for generating the clock and reset signals, and a framework for controlling the value of the Stop signal over time.
   c. See the material in the References section at the end of this document for details on how to run a simulation in Quartus.

3. **Test your design**
   a. Verify that the counters are NOT counting when Stop is asserted.
   b. Verify that both counters are counting when Stop is not asserted.
   c. Verify that the two counters are cycling thru the correct values, in the opposite order.
   d. Verify that the status (win or lose) outputs work correctly.

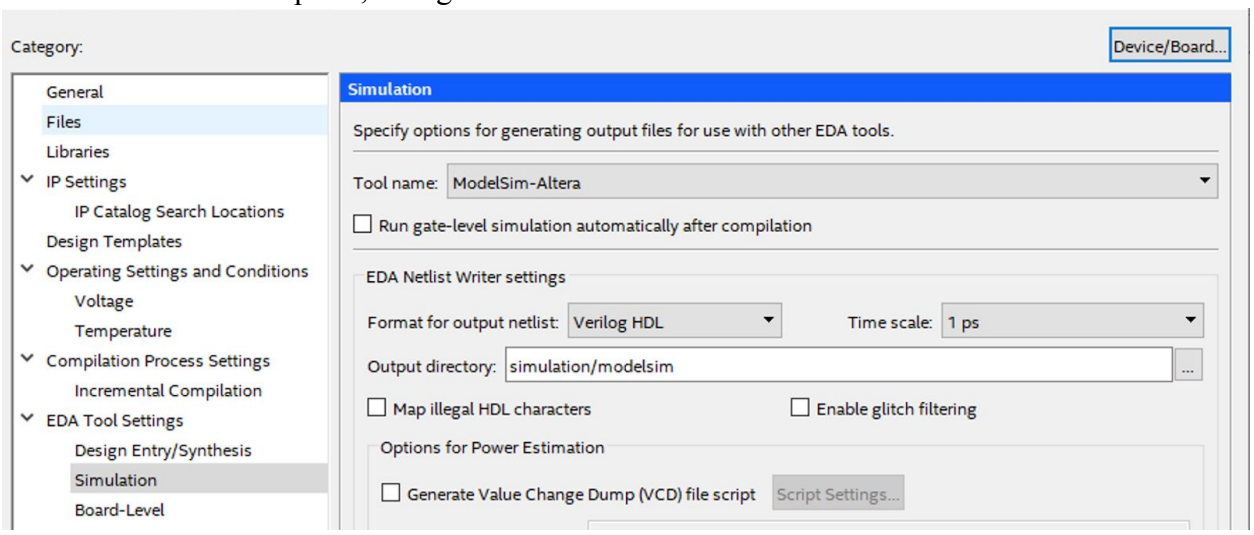
   When you have convinced yourself that it works, demonstrate to your TA.

IV. REPORT

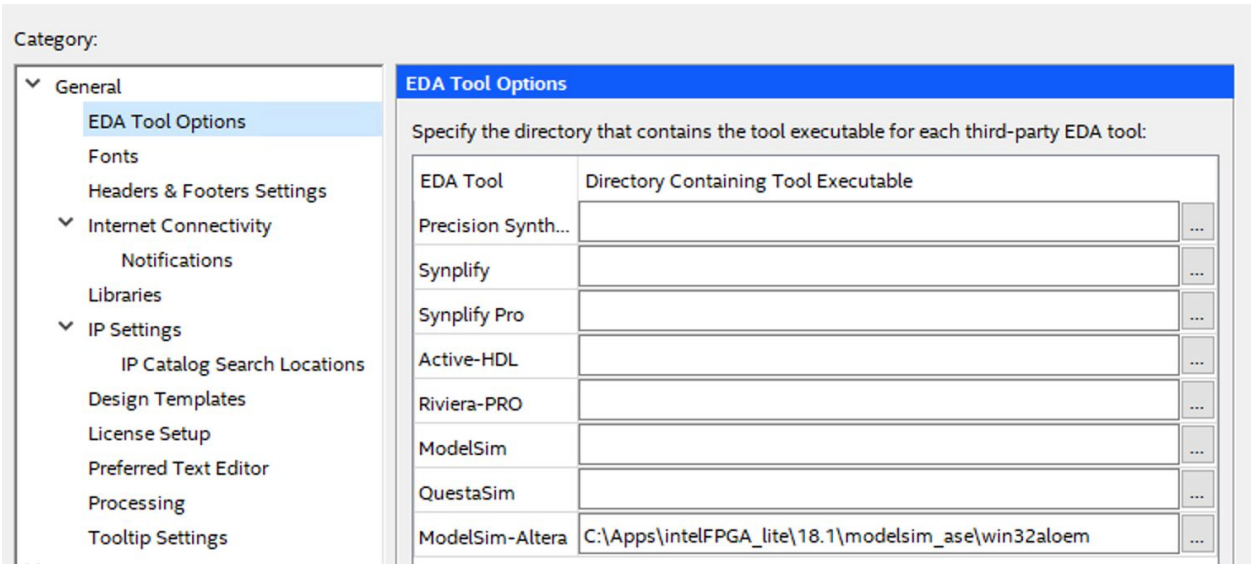
1. Include an introduction, circuit diagrams and Verilog code from the prelab and lab.
2. If you were to change your design to have the counters count from 1 to 8 (or 8 to 1) instead of only 1 to 6, list all the things you would need to change.

.V. REFERENCES

In order to simulate with Verilog code rather than the Waveform editor, we will need to set up our simulator. To this, first go to *Assignment -> Settings -> EDA Tool Settings -> Simulation*. For the "Tool Name" option, change it from "None" to "Modelsim-Altera" and then click OK.



Next, go to *Tools -> Options -> General -> EDA Tool Options*. Under the "Modelsim-Altera" simulator, paste in the following: *"C:\Apps\intelFPGA_lite\18.1\modelsim_ase\win32aloem"*.



Now that our simulation tool is set up, we need to convert our top-level schematic into a Verilog file, as the simulator cannot natively parse schematic files. To do this, go to *File -> Create/Update -> Create HDL Design File For Current File* while your top-level schematic is open. Make sure the file type is a "Verilog HDL" file, and click OK. Next, remove your schematic (BDF) file from the project, **BUT** do not delete the file from your directory (in case you need to change it later). Make the newly-generated Verilog file your top-level entity.

Once you're ready to start simulating, start an "Analysis & Elaboration," **NOT** a synthesis or full compilation. When this is complete, go to *Tools -> Run Simulation Tool -> RTL Simulation*. A window should pop up. This is the simulation tool we are using called Modelsim. On the left-hand side, one of the open panes should say "Library," with the first item in the library listed as "work". Select this library, then go to *Compile -> Compile* and select your top-level file and the test bench file, then click OK. When this is done, you should see your test bench module appear in the "work" library. Double click on it to open it. You should a bunch of signals appear in the "Objects" pane. Select all of them and right click, and choose "Add Wave". The signals should now appear in the waveform viewer on the right-hand side. Finally, to run the simulation, go to *Simulate -> Run -> Run -All* to run the test bench simulation.