

SANTA CLARA UNIVERSITY	ELEN 21L Fall 2022
<p style="text-align: center;"><b>Laboratory #7: Counters</b></p> <p style="text-align: center;"><b>For lab sections Monday-Friday November 7- 11, 2022</b></p>	

## 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.

We're going to use **3-bit** counters as the primary building block for this design. A 3-bit counter can express 8 possible values (from 0 to 7), which is an even number of values. If we have an up counter and a down counter running through an even number of values, and running on the same clock, they will always match at a given value every time they run through their sequence. So we are going to modify the behavior of these counters so they only range through an odd number of values, specifically they will stay within the range of 1 to 5.

We will model the button control of the counting with 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.

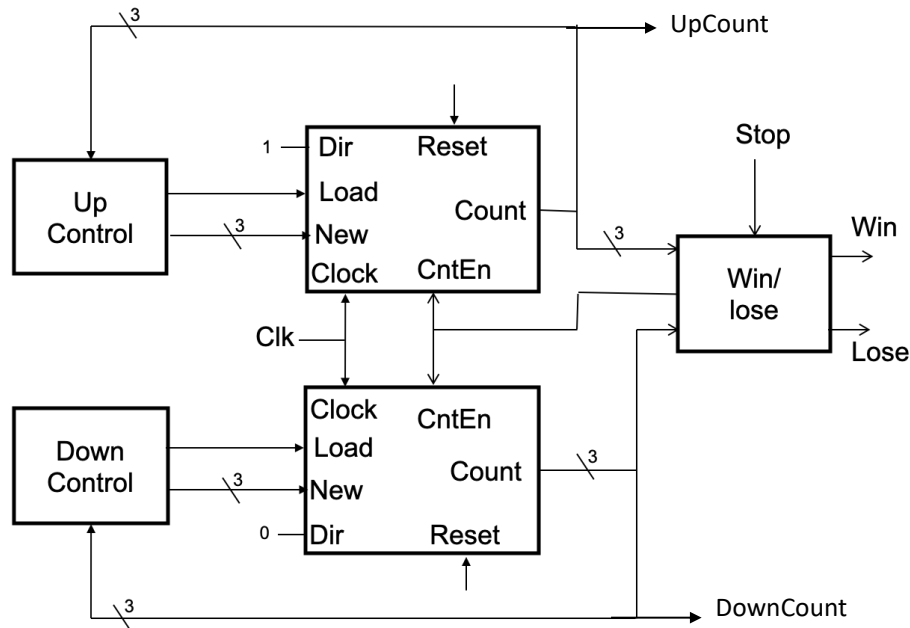
A "lose" signal is asserted if the two counter values **do not match** when Stop is asserted.

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:



You will be given a Counter module (in modules.txt), which is represented by the two blocks in the middle of the diagram above. These counters will be standard 3-bit counters, i.e., they will be designed to count the entire range of 0 to 7. But note that this Counter module has a number of inputs (besides Clock and Reset):

- *CntEn* – which will enable the *Count* value to change on the next rising edge of the clock
- *Dir* – which will determine the direction in which the counter will count. If *Dir*=1 the counter will count up. If *Dir*=0 the counter will count down. Note in the diagram that we're going to be explicitly forcing one of the counters to be an up counter and one to be a down counter.
- *Load* – when asserted, this will cause the *Count* output to change (at the next rising edge of the clock) to the 3-bit value on the *New* input
- *New* – a 3-bit value that will be loaded into the counter (on the next rising edge of the clock) if and only if the *Load* signal is asserted

You will be designing the other three modules (*Up Control*, *Down Control*, and *Win/Lose* – skeletons are given in modules.txt) and connecting everything up in a top-level module. The behaviors of these modules are described below. You will be completing the Verilog for these as your pre-lab submission (see the provided modules.txt file to get started).

### 1. UpControl

This module takes a 3-bit *Count* value as input, and generates a *Load* signal and 3-bit *New* count value as outputs. Since we're aiming for a counter that counts from 1 to 5, this module needs to assert *Load* when it sees the count input is the value 5, and it needs to cause the counter to load the new value 1 (3'b001 in Verilog).

## 2. DownControl

The inputs and outputs for this module are the same as UpControl, but in this case we need a module that will assert *Load* when it sees its count input shows the value 1, and it needs to cause the down counter to load the new value 5 (3'b101 in Verilog).

## 3. Design the win/lose block

Write a Verilog module *WinLose* that takes as inputs the two 3-bit count values (*UpCount* and *DownCount*) 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 (*CntEn*)
- explicit win and lose signals (*Win* and *Lose*)

The counters should only count when *Stop* is not asserted. The *Win* and *Lose* signals should only assert while *Stop* is asserted. And only one of *Win* and *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**


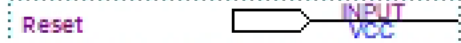





## III. LAB PROCEDURE


### 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.

- Create a top-level design and add the modules you developed in your prelab.
  - Draw a top level schematic (*File* → *New...* → *Design Files* → *Block Diagram/Schematic File*) and save it as lab7.bdf.
  - In order to make the port names consistent with the provided testbench, use the following port names in your schematic:

	Port name	Quartus symbol
Inputs	Clock	
	Reset	
	Stop	
outputs	Win	
	Lose	
	UpCount[2..0]	
	DownCount[2..0]	

- iii. We need to convert the 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.
- iv. Make sure the file type is a “Verilog HDL” file, and click OK.
- v. 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) – use *Project* → *Add/Remove Files in Project...*
- vi. For compilation, use “*Analysis & Elaboration*” ( , NOT a synthesis or full compilation.
- b. Your TA will provide a framework (tb.v) 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<sup>1</sup>.
- c. See the material in the References section at the end of this document for details on how to run a simulation with a testbench in Quartus.

### 3. Test your design

- d. Verify that the counters are NOT counting when Stop is asserted.
- e. Verify that both counters are counting when Stop is not asserted.
- f. Verify that the two counters are cycling through the correct values, in the opposite order.
- g. Verify that the status (win or lose) outputs work correctly.

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

## IV. REPORT

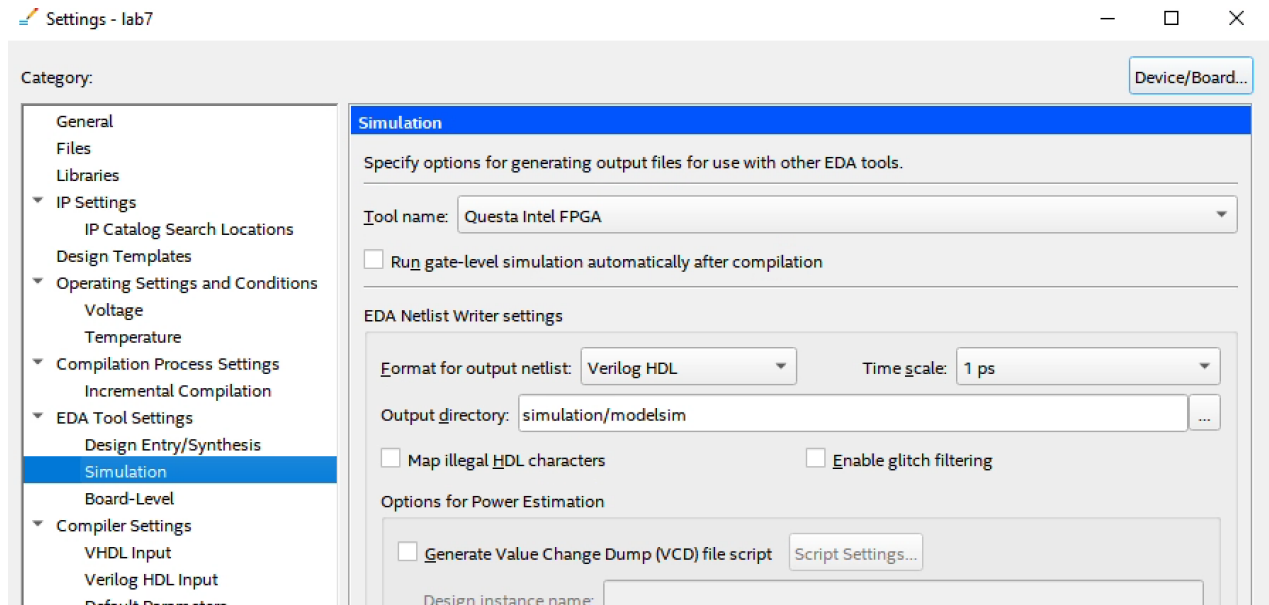
1. Include an introduction, the block diagram, and your Verilog code from the lab.
2. If you were to change your design to have the counters count from 2 to 6 (or 6 to 2) instead of only 1 to 5, list all the things you would need to change.

---

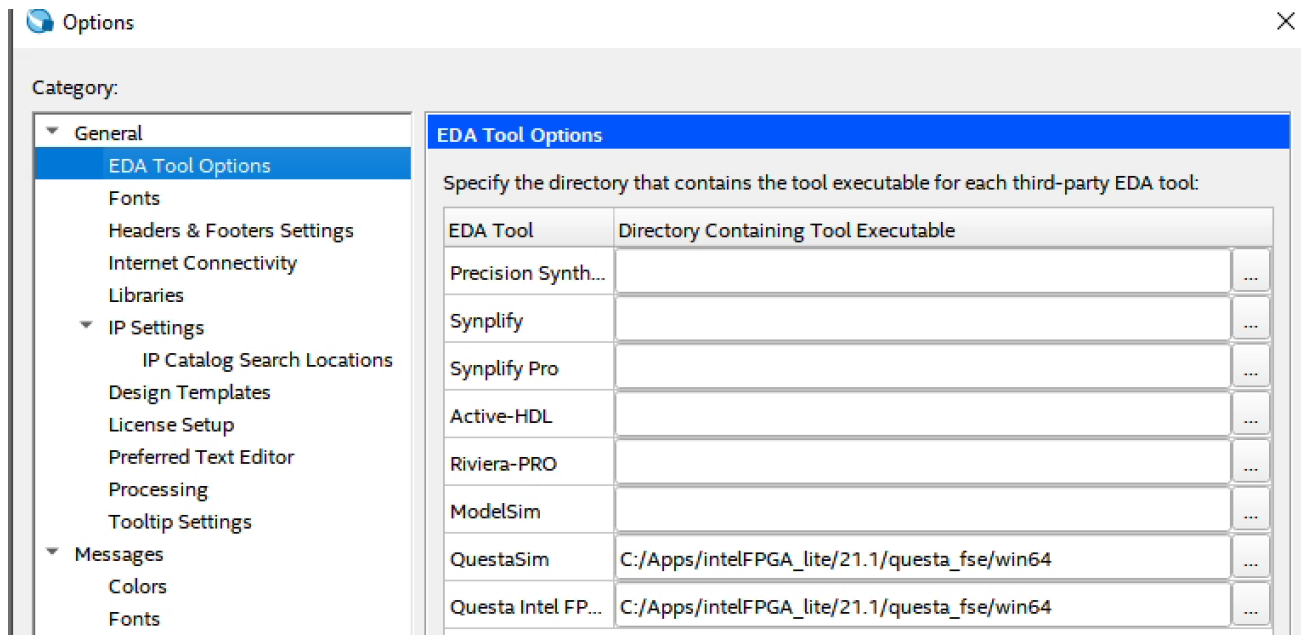
<sup>1</sup> Previously we used a tool (University Program VWF) for defining the input sequence to be simulated. But we can do this directly in a testbench Verilog module. If you are further interested, see pages 44-45 of Verilog Syntax Reference.

## V. REFERENCES

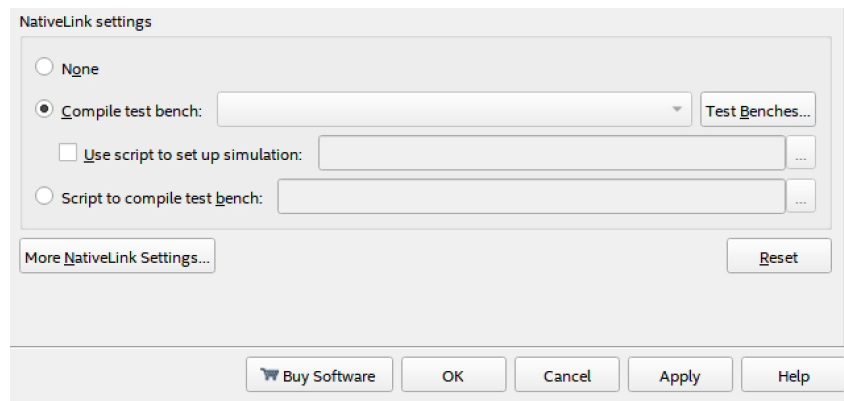
In order to simulate with Verilog code rather than the Waveform editor, make sure that simulation option has been set up correctly. To do this, first go to *Assignment* → *Settings...* → *EDA Tool Settings* → *Simulation*. For the “Tool Name” option, change it to “Questa Intel FPGA”.



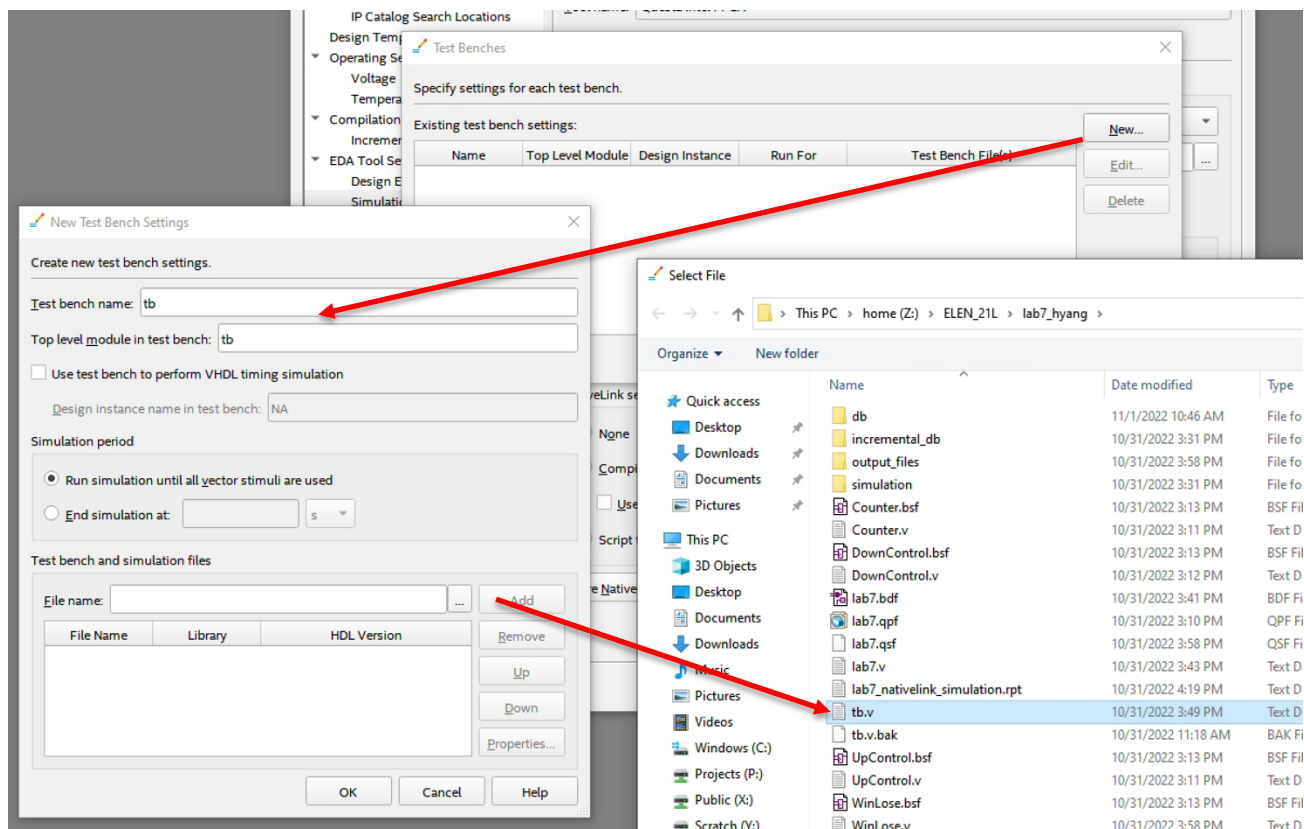
Also, make sure that the simulation tool paths are set correctly: *Tools* → *Options* → *General* → *EDA Tool Options*.



Put the provided testbench file (tb.v) in your project folder. Go to *Assignment* → *Settings...* → *EDA Tool Settings* -> *Simulation* and choose “*Compile test bench*” for the “*NativeLink settings*”. Then, click on “*Test Benches...*”



In the Test Benches panel, click on “*New...*” and set the “*Test bench name*” and “*Top level module in test bench*” to tb, which is the module name of the provided testbench (tb.v). Locate tb.v in your project folder and “*Add*” it in “*Test bench and simulation files*”.



When this is complete, go to *Tools* → *Run Simulation Tool* → *RTL Simulation*. (NOT *Gate Level Simulation...*) A simulation window should pop up after a delay.