# Lab 3 Stopwatch

## Deadlines & Grading

- **Prelab**
  - **Part A:** Friday, February 28, 2014 at 11:59 PM – 12 points, **to be done individually**
  - **Part B:** Thursday, March 6, 2014 at 11:59 PM – 8 points, **in groups of 2**
- **In-Lab Exercises**
  - **Part A:** Monday, March 3 / Tuesday, March 4 – 25 points, **in groups of 2**
  - **Part B:** Monday, March 10 / Tuesday, March 11 – 25 points, **in groups of 2**
- **Report**: Monday, March 17 / Tuesday, March 18 at 11:59 PM – 30 points, **in groups of 2**

## Section I: Overview

So far, the logic you have worked with in lab is known as *combinational logic*, where the outputs only depend on the current inputs. However, much of the power of digital design is in its ability to retain *state*, some form of memory. By having state, we can design circuits that can perform much more complex operations, since we can break down these operations into steps and create the circuit such that it remembers what step it is currently in for the operation. We call this *sequential logic*, since we're following a sequence of steps. One simple example of sequential logic is a counter, which just keeps tally of which step it is currently in. In this lab, we will be building a stopwatch, which is in effect nothing more than a counter that counts time.

## Section II: Background

Digital clocks and stopwatches have become omnipresent. Timers and clocks play a critical role in a wide variety of areas, from watches to microwaves, and even to nuclear reactors, and are ubiquitous enough that we often take them for granted. A large number of athletic records would effectively not be trackable if it were not for stopwatches. Patented in 1869,[1] stopwatches (which at the time were analog devices with a dial) have undergone massive changes. Presently, the majority of stopwatches are digital, like the one shown in Figure 1, and are expected to come with once-luxurious features such as time and date displays.


**Figure 1.** Digital stopwatch.[2]

The sports world has benefitted immensely from better timing technology. The practice of timing races started as early as 1860.[1] The Stockholm Olympics in 1912 were the first popular event to use a hand-cranked camera to

**Figure 2.** Seiko Photo Beam.[5]

time athletes crossing the finishing line. [1] Even so, the precision of the results was not high enough. It was only in 1948, during the London Summer Games, that a photo finish camera was used to time the men's 100m finals, recording times with a precision of up to 0.01 seconds.[3]

Advanced stopwatches are no longer constrained to being started by the press of a button; upon detecting a trigger, they can start counting automatically. One such trigger is an audio signal,[4] which can be used to signal athletes at the same time. A more sophisticated design is the Seiko Photo Beam Unit (Figure 2), which records the precise instant an athlete crosses a certain point by observing when an infrared beam is broken.[5]

New timing technology has significantly altered training in sports. Athletic training can be improved by taking into account the robust nature of the human body, stemming from the adaptation of cells to strain and relaxation patterns.[6] For optimal performance, a training ratio (the ratio of load to recovery)[6] is calculated with the use of timers and knowledge of an athlete's body composition, in order to guide the training process. Set Starter introduced the world's smallest timer in 2012, shown fitting around a finger in Figure 3. They utilize touch-sensitive controls, along with features ranging from LEDs to alarms,[7] to enable athletes to focus more on their training and less on their monitoring equipment.


**Figure 3.** Set Starter.[8]

# Section III: Prelab, Part A

## A. Verilog Basics

As you saw in Lab 2, even a smaller-scale project such as the seven segment display circuit can be extremely time consuming to build using real chips and wires. For the remaining labs this semester, we will use Quartus to help make building these circuits much simpler. By now, you should have completed *Tutorial B: Installing Quartus II*. You will now get familiar with how to use Quartus, and will start learning how to use *Verilog*. Verilog is a programming language, but unlike typical languages (such as C, Python, or Java), which have you list a series of steps to perform in a fixed order, Verilog instead describes logic gates and wires. This means that **operations that you write in Verilog, no matter where they are in the file, are *usually* designed to occur at the same time**.

**ASSIGNMENT 1** Read and complete *Tutorial C: Introduction to Verilog and Quartus II*.

1. Download the tutorial, as well as **lab3.zip**, from CMS. Unzip the contents of the ZIP file to a folder named **lab3** on your computer. Read

**NOTE 1** to make sure you have properly unzipped the file.

2. Use the **lab3** folder to complete the tutorial. **Do not delete this folder when you are finished – you will need it for the rest of this lab.**

**ASSIGNMENT 2** Read *Tutorial D: Altera DE2 Board*.

**NOTE 1** For our labs, you cannot simply open a compressed folder (a ZIP file) without unzipping it. Quartus (and most software) cannot see inside the ZIP file. In Windows, you can unzip by opening the ZIP file and clicking on *Extract all files*, which will allow you to create a new folder.

## B. Prelab Deliverables

**DELIVERABLE 1: prelab3.zip** *due February 28*

Submit *all* of your Verilog files, including **sega.v**, **tffp.v**, and **treg4bit.v**, from **Assignment 1**, and a text file **readme.txt**, all as a ZIP file named **prelab3.zip**. The **readme.txt** file should include your name, your NetID, and any pertinent information for the graders. Your Verilog must compile and contain no inferred latches for full credit.

**NOTE 2** If you have created any sub-modules that you are using inside any of the required files, please remember to include the **.v** files for those as well. When in doubt, just submit all **.v** files inside your directory. You will get zero credit for missing files.

Make sure to submit the above file to CMS (http://cms.csuglab.cornell.edu).

# Section IV: Lab Exercises, Part A

In the lab, you will take your four-bit register (the **treg4bit** module from *Tutorial C*), and use that to create a counter. Make sure you bring a digital copy of the **lab3** folder you created for Assignment 1 (**updated with corrections based on our feedback**) to lab. At the very least, the folder should contain all your Verilog files (**.v**) from your prelab submission, **lab3.qpf**, and **lab3.qsf**.You will need these updated files to complete the lab exercise. When you come into lab, find a partner (who cannot be your partner from Lab 1 or Lab 2), and compare your prelab to theirs. Pick the version that looks more correct, and use that for this lab.

## A. Building a Counter

A counter does more than a simple register. While a register just stores data that is fed into it, a counter must be able to *increment* the value stored inside it. The counter must also have a way of incrementing the value only when it is *enabled*. Lastly, there must be some way to *reset* this counter back to zero.

Figure 4 shows you a block-level diagram for the counter, which you will create in the file **tcounter.v**. The one-bit input **CLK** is used to tell the counter when to increment. The one-bit *active low* input signal **CLR** tells the counter to reset **when the input is a zero**. The counter also has two one-bit enable inputs, **ENP** and **ENT**. When *both* **ENP** and **ENT** are high, the counter will increment whenever it sees a *rising edge* (a transition from low to high) on **CLK**. Whenever either **ENP** or **ENT** are low, the counter should *hold* the value currently stored within its register. When **CLR** is low *and* both **ENP** and **ENT** are high, the counter should *still* reset. The counter also has a four-bit output **Q**, which shows the current value stored inside the flip flops.

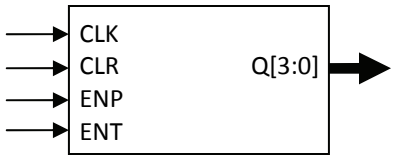

**Figure 4.** Counter block diagram.

When you create the counter, you should use your **treg4bit** register to store the data. The **tcounter** module should have an *instance* of **treg4bit**, and you must build logic inside **tcounter** that determines what inputs to send to the **treg4bit** instance. Refer to *Tutorial C* on what instances are and how to create them. Remember – for a T flip-flop, the value stored inside the register changes when the input is high. Therefore, you must make sure that each bit not only increments when it is supposed to (see slides from Lecture 7 for more details), but that it only increments when **ENP** and **ENT** are high. Also keep in mind that each bit of the **treg4bit** module will have different logic for when it should increment.

**ASSIGNMENT 3** Build a module called **tcounter** that implements a counter.

1. Open the project file **lab3.qpf** (in the **lab3** folder from **Assignment 1**) inside Quartus.
2. Click *File* → *New*. In the window that pops up, select *Verilog HDL File* under *Design File* and click OK.
3. Save this file as **tcounter.v**, making sure that it is also inside the **lab3** folder.
4. Write up the counter module, using the code blocks in *Tutorial C* as examples. Remember that your input and output names should look identical to the names we described above, and are **case sensitive**. Feel free to draw out your logic on scrap paper before you start writing Verilog.
5. Save the file, and then compile your project. Correct any syntax errors.

## B. Top-Level Assembly & ModelSim Testing

Once you have completed the **tcounter** module, you must now hook it up so we can test it out. Inside the **lab3** folder, you will find a file called **lab3.v**. This is what we will be running on the DE2 for this lab. Therefore, if you want to test your **tcounter** module, you must create an instance of it inside **lab3.v** and connect it to the inputs and outputs inside there.

For this part, connect your **tcounter** instance as follows:
- The **lab3** input **RESET** should be *inverted*, and then connected to **CLR** on **tcounter**.
- The **lab3** input **ENABLE** should be connected to **ENP** on **tcounter**.

- Connect **ENT** on **tcounter** to the hard-wired value 1'b1 for now.
- The internal **lab3** wire **CLK** should be connected to **CLK** on **tcounter**.
- The **lab3** output **CENTISEC** should be connected to **Q** on **tcounter**.
- You can leave all other inputs and outputs unconnected for now.

**ASSIGNMENT 4** Connect your **tcounter** module to the top-level **lab3** module.

1. Click *File* → *Open File*. Open **lab3.v**.
2. Create an instance of **tcounter** inside **lab3.v**, and wire it up as described above.
3. Save the file, compile your project, and test the circuit using the test bench **lab3_test.v**. Verify that your **tcounter** module is working as expected by examining the output shown in ModelSim.

## C. Preparing Your Counter for the DE2 Board

Before we download the design, we must first set up a few parameters before compiling a design. You will need a DE2 board, a plug, and a USB cable, all of which will be provided in lab.

For this lab, the input pins have already been assigned for you. They are mapped as follows:
- The **RESET** input uses toggle switch **SW0**.
- The **CLK_SEL** input uses toggle switches **SW3** (MSB) through **SW1**. Set this to $000_2$ to execute your logic at 1 Hz (allowing us to observe one change per second).
- Input **ENABLE** (which is connected to **ENP** on the counter) uses toggle switch **SW4**.

Circuit outputs are assigned as follows:
- Output **CENTISEC** is shown on **HEX0**.

1. Open the **lab3.qpf** file.
2. Go to *Assignments* → *Devices…* and select the Cyclone II family. Under *Available Devices*, select *Specific device selected*, and choose the *EP2C35F672C6*. Click *OK*.
3. Check pin assignments by going to *Assignment* → *Assignment Editor*. This will show you a list of all pins. For this lab, the pins should have already been set for you.
4. Compile your design. This will create the file **lab3.sof**, which we use to program the FPGA.

## D. Testing Your Design

1. Take the DE2 board and plug the power cable into the power port on the top left. Make sure that the red power switch on the left is pushed in to turn the board on. You will see the board light up, and the LCD screen will say "Welcome to the Altera DE2 Board."
2. Connect a USB cable to the leftmost USB port on the DE2 board (the **USB Blaster Port**). Connect the other end to the front of your PC.
3. When you plug in the cable, Windows should tell you that the **Altera USB-Blaster** device was installed properly. If it wasn't, please let a TA know.
4. Turn the **RUN/PROG** switch on the left of the board to the **RUN** position.

5. Inside Quartus, go to *Tools* → *Programmer*. Under *Hardware Setup*, select *USB-Blaster*.
6. Inside the programmer tool window, you will see your **lab3.sof** file listed. Make sure that the check box under *Program/Configure* is checked, and click *Start*. Once this completes, your design will now be downloaded to your FPGA, and you can start testing your circuit.

Once you have verified that your circuit is working properly, request to be checked off by a TA. After you have successfully programmed your FPGA, we may ask questions related to the lab, which will count towards your lab exercise grade. **Remember to save your folder somewhere when you have finished – you will need these files to complete the second part of the lab!**

Before leaving the lab:
- Make sure a TA has checked off all parts of your lab, including parts that aren't fully working.
- If you have used the lab computer, make sure to **save all files remotely** and then log off. Upon logging off, the computer automatically deletes any files that you created, so be careful!
- Clean your bench.

## Section V: Prelab, Part B

The counter that you created in Part A will allow you to count from 0 to 15, since it contains four bits. After it reaches 15, it will increment again, but since it cannot carry the one over (since there is no bit to the left), the counter will reset to 0, and will start counting from there again.

Obviously, while all this is nice, it is of little use for a real stopwatch. After all, a stopwatch has digits that only count from 0 through 9. In fact, not all digits even go all the way up to 9 (since there are only 60 seconds in a minute). We are going to build a stopwatch that can count minutes (**M**), seconds (**S**), and hundredths of seconds (**h**), which will be displayed in the five-digit format $\mathbf{M}:\mathbf{S_1S_0}:\mathbf{h_1h_0}$. To do this, we must understand how to adapt our **tcounter** module from Part A so that it can properly represent each of the digits. For the prelab, you must figure out the range for each digit, as well as what conditions must be met to *(a)* increment and *(b)* reset each digit.

**ASSIGNMENT 5** For each digit of the stopwatch, write down the range that each digit can take. Make sure you label each digit clearly.

**ASSIGNMENT 6** For each of the digits, write down when the digit should increment. (Hint: you can use the values of other digits.) For example, if a two-digit counter is going from 09 to 10, the upper digit should know to increment when the lower digit is 9, not 0, since increments occur in lockstep at the rising clock edge. Inside an `always` block, keep in mind that all flip-flop values on the right hand side of the assignment operator (`<=`) are the values *just before the rising edge*.

**ASSIGNMENT 7** Write down what the reset conditions are for each of the digits.

**DELIVERABLE 2: prelab3.pdf** *due March 6*

Submit answers for **Assignments 5-7** in PDF format. This file must contain the range, increment conditions, and reset conditions for each digit, with the digit clearly labeled. Make sure that the PDF includes the name and NetID of all group members.

Make sure to submit the above file to CMS (http://cms.csuglab.cornell.edu). Do not forget to add and approve your group members on CMS **before uploading your assignment**.

# Section VI: Lab Exercises, Part B

For the second part in lab, you will take your code from Part A, and use that to build a complete stopwatch. The work that you did in **Assignments 5-7** should allow you to determine what logic you need to connect multiple **tcounter** instances together. Make sure you bring a digital copy of the **lab3** folder from the in-lab work for Part A. If you did not successfully complete Part A in lab, **your group must get it working *before* coming into lab to do Part B**.

## A. Top-Level Assembly & Testing

Once you have completed the **tcounter** module, you must now hook it up so we can test it out. Inside the **lab3** folder, you will find a file called **lab3.v**. This is what we will be running on the DE2 for this lab. Therefore, if you want to test your **tcounter** module, you must create an instance of it inside **lab3.v** and connect it to the inputs and outputs inside there.

For this part, connect your **tcounter** instance as follows:

- The **lab3** input **RESET** should be connected, along with the reset logic for each **tcounter** digit (you can use Boolean logic gates to connect these together), to the **CLR** input of each **tcounter** instance. Don't forget that **CLR** is *active low*, so you may need to invert your final input.
- The **lab3** input **ENABLE** should be connected to **ENP** on *all* **tcounter** instances.
- Use the **ENT** input on each **tcounter** instance to control when incrementing should occur.
- The internal **lab3** wire **CLK** should be connected to **CLK** on *all* **tcounter** instances.
- The **lab3** output **CENTISEC** should be connected to **Q** on the **tcounter** instance for digit $\mathbf{h_0}$.
- The **lab3** output **DECISEC** should be connected to **Q** on the **tcounter** instance for digit $\mathbf{h_1}$.
- The **lab3** output **SEC** should be connected to **Q** on the **tcounter** instance for digit $\mathbf{S_0}$.
- The **lab3** output **TENSEC** should be connected to **Q** on the **tcounter** instance for digit $\mathbf{S_1}$.
- The **lab3** output **MIN** should be connected to **Q** on the **tcounter** instance for digit **M**.

**ASSIGNMENT 8**

Modify the **lab3.v** file to connect several **tcounter** instances together, using your work from **Assignments 5-7** to determine what logic needs to be added. Remember to compile and test your design using ModelSim. Inside **lab3_test.v**, you will need to delete the two lines in the test bench that say "FOR PART B, DELETE THIS LINE ONLY" before starting your testing.

## C. Preparing Your Counter for the DE2 Board

Before we download the design, we must first set up a few parameters before compiling a design. You will need a DE2 board, a plug, and a USB cable, all of which will be provided in lab.

For this lab, the input pins have already been assigned for you. They are mapped as follows:
- The **RESET** input uses toggle switch **SW0**.
- The **CLK_SEL** input uses toggle switches **SW3** (MSB) through **SW1**. For this part of the lab, set **CLK_SEL** to $010_2$ to execute your logic at 100 Hz (allowing the stopwatch to operate at its intended speed).
- Input **ENABLE** (which is connected to **ENP** on the counter) uses toggle switch **SW4**.

Circuit outputs are assigned as follows:
- Output **CENTISEC** is shown on **HEX0**.
- Output **DECISEC** is shown on **HEX1**.
- Output **SEC** is shown on **HEX2**.
- Output **TENSEC** is shown on **HEX3**.
- Output **MIN** is shown on **HEX4**.

Use the instructions in Sections IV-C and IV-D to program your DE2 board.

Once you have verified that your circuit is working properly, request to be checked off by a TA. After you have successfully programmed your FPGA, we may ask questions related to the lab, which will count towards your lab exercise grade.

Before leaving the lab:
- Make sure that you have had a TA check off all parts of your lab, including parts that aren't fully working.
- If you have used the lab computer, make sure to **save all files remotely** and then log off. Upon logging off, the computer automatically deletes any files that you created, so be careful!
- Clean your bench.

# Section VII: Lab Report

**DELIVERABLE 3: lab3report.pdf** *due **seven days after lab***

You will submit a single report for your group. **Each partner must share in the writing of the report.** Please refer to the Lab Report Guidelines on Blackboard for general information on what to include. The report must be a PDF file, named **lab3report.pdf**, and should be submitted through CMS (http://cms.csuglab.cornell.edu).

Groups in the Monday lab sessions must submit their report by **Monday, March 17 at 11:59 PM**. Groups in the Tuesday evening lab session have until **Tuesday, March 18 at 11:59 PM**.

In general, lab reports for ENGRD 2300 should do the following:

- Assume that the reader has a general ECE knowledge, but has never seen this handout (and does not have access to it). This means you must summarize the purpose of this project, as well as the tasks you have been asked to complete. This also means you cannot refer to any portion of this document in your write-up.
- Include some sort of introduction.
- Use section headings to organize the document.
- Provide enough detail so we can reproduce your lab *exactly* as you designed it without looking at your submission.
- Include a section on the distribution of work for the jointly-done parts of the lab.

For this particular lab report, you should remember to discuss the following:

- Talk about how the T flip-flop allows you to count upwards.
- Discuss whether you could implement the counter without using sequential logic.
- Compare your prelab design with your partner's. If they differ significantly, please explain why, as well as the reasons you selected one over the other for implementation.

# References

[1] Ross, S. (2008). *Higher, Further, Faster: Is Technology Improving Sport?* (p. 271). Chichester, England: Wiley/Dana Centre.

[2] *Stoppuhr digital* (2008, February 2). Wikimedia Commons. Retrieved February 13, 2014, from https://commons.wikimedia.org/wiki/File:Stoppuhr_digital.jpg

[3] *A History of Technology in Sports* (2012, August 10). +Republic. Retrieved October 19, 2012, from http://plusrepublic.com/a-history-of-technology-in-sports/

[4] Dabnichki, P. (2008). "Motion Analysis in Water Sports." *Computers in Sport* (p. 235). Southampton: WIT Press.

[5] *Harmony with SEIKO – Athletics: Track*. Seiko Holdings Corp. Retrieved October 19, 2012, from http://www.seiko.co.jp/en/harmony/sports_timing/timing_system/track.php

[6] *Training Theory*. International Association of Athletics Federations. Retrieved October 19, 2012, from http://www.coachr.org/training_theory.htm

[7] *Tiny Timer Helps Athletes Monitor Rest Between Exercise Sets to Improve Fitness* (2012, August 31). Yahoo! News. Retrieved October 19, 2012, from http://news.yahoo.com/tiny-timer-helps-athletes-monitor-rest-between-exercise-020310056.html

[8] *Set Starter Interval Training Timer Fits on Your Thumb* (2012, July 10). Set Starter. LLC. Retrieved on February 14, 2014, from http://www.prweb.com/releases/setstarter/intervaltraining/prweb9675425.htm