Lab 2 Stopwatch

Deadlines & Grading

- Total 100 points, work in groups of 2 **except Prelab 2A**
- Prelab 2A: 25 points, **work individually**. **Submit to CMSX by Wednesday, Feb 16th 11:59 pm**
- Lab 2A: 20 points, work in groups of 2. **Check off before Friday, Mar 4th 11:59 pm**
- Prelab 2B: 15 points, work in groups of 2. **Submit to CMSX by Wednesday, Mar 9th 11:59 pm**
- Lab 2B: 40 points, work in groups of 2**. Check off before Monday, Mar 14th 11:59 pm**
- **Note that students must rotate partners in Lab 3**

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 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]

Figure 2. Seiko Photo Beam.[5]

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 2A

A. Verilog Basics

As you saw in Lab 1, even a small-scale project 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. If you haven't already, install Quartus on your computer following ***Tutorial A: Installing Quartus II***. You will now get familiar with using Quartus, and will start learning 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. Because of this, it is usually termed a 'hardware description language.' This means that **operations that you write in Verilog, no matter where they are in the file, are *usually* designed to occur in parallel as separate hardware modules**.

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

1. Download from CMSX the Tutorial, as well as **lab2.zip**. Unzip the contents of the ZIP file to a folder named **lab2** on your computer. Read **NOTE 1** to make sure you have properly unzipped the file.
2. Use the **lab2** 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 C: Altera DE0-CV Board***.

NOTE 1 For our labs, you should not simply open a compressed folder (a ZIP file) without unzipping it. Quartus (and most software) cannot see inside the ZIP file. In Windows/Ubuntu, 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 2A Deliverables

DELIVERABLE 1 ***due Feb 16th, 11:59 pm***

a. 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 **prelab2a.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. **Submit this file to CMSX, one submission per person.**

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.

Section IV: Lab 2A: Counter

Next, you need to build a counter using the four-bit register (the **treg4bit** module from ***Tutorial B***). Before working on this part, make sure that registers are working properly. The **lab2** folder you created for Assignment 1 should contain all your Verilog files (**.v**) from your prelab submission, as well as **lab2.qpf**, and **lab2.qsf**.

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.

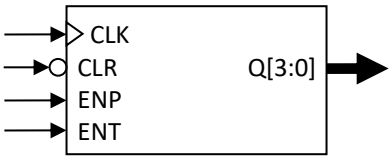

Figure 4. Counter block diagram.

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 CLR 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** is 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.

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 B*** 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 changes when it is supposed to, but also that it only changes 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 change.

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

1. Open the project file **lab2.qpf** (in the **lab2** folder from **Assignment 1**) inside Quartus.
2. Open the *tcounter.v* file.
3. Write up the counter module, using the code blocks in *Tutorial B* 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.
4. 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 **lab2** folder, you will find a file called **lab2.v**. This is what we will be running on the DE0-CV for this lab. Therefore, if you want to test your **tcounter** module, you must create an instance of it inside **lab2.v** and connect it to the inputs and outputs inside there.

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

- The **lab2** input **RESET** should be ***inverted***, and then connected to **CLR** on **tcounter**.
- The **lab2** input **ENABLE** should be connected to **ENP** on **tcounter**.
- Connect **ENT** on **tcounter** to the hard-wired value 1'b1 for now.
- The **lab2** input **CLK** should be connected to **CLK** on **tcounter**.
- The **lab2** output **CENTISEC** should be connected to **Q** on **tcounter**.
- You can leave all other inputs and outputs unconnected for now.

ASSIGNMENT 4 Add your **tcounter** module to the **lab2** module.

1. Click *File* → *Open File*. Open **lab2.v**.
2. Create an instance of **tcounter** inside **lab2.v**, and wire it up as described above.
3. Save the file, compile your project, and test the circuit using the test bench **lab2_test.v**. Verify that your **tcounter** module is working as expected by examining the output shown in ModelSim. Note that we are using the same testbench file for both Lab 2A and 2B, so for now, examine only the output relevant to Lab 2A.

C. Preparing Your Counter for the DE0-CV FPGA Board

Before you download the design on to the FPGA board, you must first setup a few parameters for compiling a design.

For this lab, the input pins have already been assigned for you. They are mapped as follows:
- The **RESET** input uses button **FPGA_RESET**.
- The **CLK_SEL** input uses toggle switches **SW9** (MSB) through **SW7**. 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 **SW0**.

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

1. Open the **lab2.qpf** file (the project).
2. Go to *Assignments* → *Devices…* and select the Cyclone V family. Under *Available Devices*, select *Specific device selected*, and choose the *5CEBA4F23C7*. Click *OK*.
3. For this lab, the pins should have **already** been set for you. Normally you can check pin assignments by going to *Assignment* → *Assignment Editor*. This will show you a list of all pins.
4. Compile your design. This will create the file **lab2.sof**, which we use to program the FPGA.

D. Testing Your Design

1. Take the DE0-CV board and connect one end of the usb cable to the leftmost port (USB Blaster Port) of the DE0-CV board and other end to your system. 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.
2. When you plug in the cable, if you are on Windows, it should tell you that the **Altera USB-Blaster** device was installed properly (see **Tutorial A** for Driver Installation on Windows). If you are running Quartus on a VM, you need to go the Devices → USB Devices in the Virtual-Box menu and select Altera USB-Blaster device.
3. Turn the **RUN/PROG** switch on the left of the board to the **RUN** position.
4. Inside Quartus, go to *Tools* → *Programmer*. Under *Hardware Setup*, select *USB-Blaster*.
5. Inside the programmer tool window, you will see your **lab2.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.
6. We provided with you a simple list of test cases on CMSX, named *Lab2A Self Check.pdf*. Try to think of the expected behavior of your design under each test case and make sure your design behaves as expected before checking off.

E. Check-off

During the lab check-off, you will demonstrate a working counter on the DE0-CV board. The TAs will ask you questions based on the counter you have designed using **treg4bit**.

Section V: Prelab 2B

The counter that you created in Section IV 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 of 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 Section IV so that it can properly represent each of the digits. For this 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 back to 0.

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: ***due Mar 9th, 11:59 pm***

a. **Submit answers for Assignments 5-7** in PDF format to **CMSX**. 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.

Section VI: Lab 2B: Stopwatch

You will use the code developed in Section IV to build a complete stopwatch. The work that you did in **Assignments 5-7** should allow you to determine what logic you need when connecting multiple **tcounter** instances together.

A. Top-Level Assembly & Testing

In this part, you need to connect **tcounter** instances to implement a stopwatch, and test it out. Inside the **lab2** folder, you will find a file called **lab2.v**. This is what we will be running on the DE0-CV for this lab. Therefore, you need to create your **tcounter** instances in this file, and also add necessary logic to control when the counters need to be incremented or cleared. You also need to connect the **tcounter** instances to the inputs and outputs in **lab2.v**.

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

- The **lab2** 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 **lab2** 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 **lab2** input **CLK** should be connected to **CLK** on *all* **tcounter** instances.
- The **lab2** output **CENTISEC** should be connected to **Q** on the **tcounter** instance for digit $\mathbf{h_0}$.
- The **lab2** output **DECISEC** should be connected to **Q** on the **tcounter** instance for digit $\mathbf{h_1}$.
- The **lab2** output **SEC** should be connected to **Q** on the **tcounter** instance for digit $\mathbf{S_0}$.
- The **lab2** output **TENSEC** should be connected to **Q** on the **tcounter** instance for digit $\mathbf{S_1}$.
- The **lab2** output **MIN** should be connected to **Q** on the **tcounter** instance for digit **M**.

ASSIGNMENT 8

Modify the **lab2.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. Note that we are using the same testbench file for both Lab 2A and 2B, so for now, examine only the output relevant to Lab 2B.

B. Preparing Your Stopwatch for the DE0-CV Board

We must first set up the following parameters before compiling a design. We will do so in a file called **lab2_top.v**:

- The **RESET** input uses button **FPGA_RESET (**signal name **RESET_N)**.
- The **CLK_SEL** input uses toggle switches **SW9** (MSB) through **SW7**. 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 **SW0**.

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 DE0-CV board.

C. Testing Your Design

1. Program your FPGA in the same way as you did in Lab 2A.
2. We provided with you a simple list of test cases on CMSX, named *Lab2B Self Check.pdf*. Try to think of the expected behavior of your design under each test case and make sure your design behaves as expected before checking off.

D. Check-off

During the lab check-off, you will need to demonstrate a complete stopwatch on the DE0-CV FPGA board. The TAs will also ask you questions on the lab and ask you to demonstrate debugging using ModelSim and test benches.

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