Lab 8: Final Project: Stopwatch

Purpose

In this lab, you will:

- Design and implement a stopwatch device using elements of combinational and sequential logic.
- Aggregate knowledge from throughout the semester including,
  - State Machine Diagrams
  - Registers and Memory
  - Combinational Logic
  - Decoding
  - Binary to BCD Conversion.

Tutorials and links

Here are a few links and tutorials just for easy access. You should understand most of these concepts already.

a) VHDL tutorial
b) Buses and Bus Taps
c) Pattern Wizard
d) State Machine Diagrams
e) Pin Assignments

Background

For this lab, you will be designing a variation of the stopwatch circuit that was presented in the lecture. The comprehensive description is available on the slides of the lecture notes (posted on Blackboard). Once you have read through the design process of the stopwatch, you will implement the data-path in Figure 1 (shown below) using all you have learned. The Counter and Storage Register modules can be taken directly from Xilinx. All other modules will need to be designed and implemented (or reused from previous labs). Specifically, you will need to design the following:

- 1 16-bit, A<B / A=B Comparator.
- 2 16-bit 2-to-1 Multiplexers
- 1 16-bit Binary to BCD decoder
- 1 4-digit BCD to 7-segment decoder
- 1 State Machine Diagram-based Control Module
- Display Logic
Figure 1: Stopwatch Data-Path

(note this is slightly different from the one in the slides given earlier; follow this one)
Pre-Lab Questions
(Done individually)

NOTE: We will be checking the prelabs at the beginning of lab. Any prelab not ready by the beginning of your lab section will not receive credit. All prelabs must be done individually, not in groups.

1. To get started with the comparator, fill in the rest of the truth table below for a two-bit comparator. The K-map generated from this table will help create the comparator block shown above. (Why do we need both A<B and A=B? Think about cascading multiple 2-bit comparators to create a 16-bit comparator.)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>A&lt;B</th>
<th>A=B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Now that you have the table complete, find the logical expression (in terms of A<B and A=B) for each output, based on the inputs. Knowledge of Boolean algebra and/or K-maps may help in reducing the number of terms. Once you have those, you can construct a combinational logic block with both outputs. Using multiple blocks you can build comparators of 4, 8, 16, … bits. (How will the intermediate logic work?)

Expression for A<B: \( F1 = \)

Expression for A=B: \( F2 = \)
Lab Procedure

These steps are given only as a recommendation. You are free to come up with your own approach. At this point, you should be comfortable with doing the labs, Xilinx, modular design, etc. For the lab final, you will have to complete an entire lab individually, so make sure both partners know how to do all the parts, including VHDL, schematic entry, simulation, and implementation.

NOTE: If you complete this lab and demo to the TAs by the end of the second lab period, you do not have to write a lab report. Keep this in mind and plan ahead to ensure you finish demoing on time.

Procedure, Part I – State Machine Diagram

1. Create a new project and give it name is Stop_Watch. Place the project in C:\users\your_name folder.

2. Create a new top level schematic source and call it SW_Toplevel. This schematic will be where you will assemble all the pieces of this lab.

3. Create a state diagram source (using StateCAD) and call this SW_Controller. This will be the module that sends output signals to your modules and receives information from the comparator.

4. The SW_Controller block should have five inputs (ALTB, RST, CSS, START and STOP) and five outputs (LSR, RSTM, ENTM, UPDATE, and DS). Also, be sure that your Reset signal is asynchronous. If you find that the circuit is malfunctioning, consult the diagram from the slides and ensure you have made all connections and conditions appropriately. This diagram is a little complicated, so make sure you understand what is exactly happening in each stage. This may take some time so start early.

5. Generate the VHDL code associated with the diagram using StateCAD and then create a schematic symbol for this code to use as part of your overall design.

6. Verify the functionality of your design.
Procedure, Part II – Comparator

1. Add a new source to the project Stop_Watch of the Schematic type and call it ALTB_Comp2.

2. Using the logical expressions from your pre-lab, construct a two bit comparator for inputs A (1:0) and B (1:0).

3. Verify circuit functionality for all cases, and make a symbol.

4. Create a 16-bit comparator using the 2-bit comparators. This may require multiple steps (eg. Scaling from 2-bit to 4-bit to 8-bit….). Call it ALTB_Comp16 and verify its functionality.

5. Create a schematic symbol for the ALTB_Comp16 module.

Procedure, Part III – Multiplexers

1. Add a new VHDL source to the project Stop_Watch and call it Two_1_16.

2. Program your multiplexer module using VHDL.

3. Create a Test Bench Module to verify the operation of your multiplexer and take a screenshot. Include this in your report. Create a symbol for the completed module.

Procedure, Part IV – Binary to BCD Converter

1. Add a new source to the project Stop_Watch of the Schematic type and call it BTOBCD_16.

2. Using the diagram of figure 2 and the Shift-Add-Three modules created in your combinational multiplier lab, chain together a 16-bit Binary to BCD converter. Making this schematic is a little tedious, so make sure to verify all your input and output bus connections. Also, make the schematic area as large as possible to avoid cluttering. The ten thousands bus is unnecessary because we only have four 7-segment displays. **Pay careful attention to the schematic below. The input bits are not in the same order on the schematic as they were in previous labs.**

3. Create a Test Bench Module to verify the operation (again include a good variety of cases) of your decoder and create a schematic symbol.
**Procedure, Part V – Storage Register**

1. Since in previous labs you have had significant experience with registers, we will allow you to use the FD16CE, which is a functional block provided by Xilinx. If you place the symbol in a schematic and double-click on it, and then click “Symbol Info”, you can see the documentation for the block. It is fairly straightforward. This will serve as your storage register which displays the shortest time recorded time.

**Procedure, Part VI – Clocks**

1. The clock system in this lab is a little more complicated than what we have done before. We are still using the CB16CE modules, but we will be using two clocks in the circuit: one for the switcher circuit and one for the top level.

2. At the top level we will be inserting a Verilog module (a language similar to VHDL) between the sysclk and CB16CE to regulate the sysclk and slow it down. We will provide you with a block called `clkgen.v`, which should be set up as per Figure 3 (on the last page of this document). The inputs to the clkgen block will be the system clock and the reset signal. The output will be a much slower clock (around 100Hz), which then goes into the CB16CE module. Essentially, we are slowing down the system clock (which functions at around 32 MHz) so that the display numbers change at 1 Hz (or multiples of 10 of this value). You can change
the constant value at the top of `clkgen.v` to change the speed of the clock (1 Hz, 10 Hz, 100 Hz etc.).

Aside: If you have time, you may want try going through the Verilog file to understand the logic behind `clkgen.v`. Verilog is much more widely used than VHDL in industry, and it is the language used in CIS 371, so this is particularly useful for computer engineering majors. If you are interested this link gives the details of how Verilog works ([Verilog Link](#)).

3. However, we will still be using the same CB16CE setup as before for the switching circuit for the 7-segment display (since we want the displays to be switching fast enough to seem like they are always on), so retain that setup from previous labs. Keep the second CB16CE in the Switcher module, and feed sysclk into it from the top level (as shown in Figure 3).

**Procedure VII – Decoder and Switcher**

1. This is the same binary to seven-segment decoder you have been using in past labs. Simply copy the VHDL into a new module for this lab.

2. The switcher module is also the same from previous labs, but try to keep the second CB16CE out of the top level so it doesn’t get too cluttered.

**Procedure VIII – Final Multiplexer**

1. As you can see from Figure 3, we have a third multiplexer that takes in the value outputted by the storage register and the value determined by the state machine. This is simply to check the value of the register at all times (mainly for debugging purposes). The input for this multiplexer will be the traditional output and the register output. The select input will be associated with switch 1 of the daughter board, so whenever you turn on that switch, the register output will be displayed. This is a good way of checking whether the state machine diagram is working properly.

**Procedure, Part IX – Top Level**

1. The top level should look similar to Figure 3. Try to make the connections as clean as possible. Feel free to increase the schematic size to accommodate all the macros and connections (you probably had to do this for the Binary to BCD converter).

2. Synthesize the schematic to make sure no design errors have occurred

3. Make sure you have behavioral simulations of all the parts you created for this lab (Comparator, Multiplexer, and 16-bit Binary to BCD decoder). A top-level simulation is not necessary since the only outputs are Cathode and Anode, which
are very hard to interpret visually. Try implementing it on the board instead and see if there are any problems.

4. Assign pins to each of the inputs and outputs. Be careful with your pin assignments: there are a lot of them (cathodes, anodes, input buttons, and register display switch).

5. Download the design onto the board and give a demo to one of the TAs.

6. Before leaving the lab, archive your project and store it on your seas account or on a memory stick. Then, delete your project from the computer.

---

**Lab Report (Due online by the start of third lab session)**

*Again, if you demo by the end of the second lab session, you DO NOT need to write a report. However, if you are unable to demo:*

You have to submit a report that contains the following (See guidelines for reports):

1. Course title, Lab number & title, your names, your group number and date
2. Brief description of the experiment including the goals and theory
   a. Also explain your procedure, even if you generally followed the instructions above
3. Circuit schematics (screenshots) and relevant explanation
4. Behavioral waveform simulations (screenshots) and relevant explanation
5. Discussion of the results, including showing that it worked as intended
6. Conclusion (including problems that arose and how they were solved, and what you learned, as well as what you accomplished)
This is approximately what the final design should look like. The combination of GND and VCC signals going into the first multiplexer simply need to be a very high number so that the initial comparison always picks the newly recorded time. Think of this as the maximum possible stored value. The initial flow chart (Figure 1) recommends \((1001100110011001)_{BCD} = 9999\).