ENGR 272 Lab 5: SystemVerilog (Digital Clock)

5.1 Background Information

As digital logic designs get complex, it becomes cumbersome to describe logic with schematics. Simplifying logic by hand can take a long time and is prone to human error. Hardware Description Languages describe digital hardware. As time went on and systems like FPGAs were created, Hardware Description Languages began to be used to directly describe the hardware to be implemented.

Computers can synthesize and simplify a design much faster than a human can. A language like Verilog (Or its newer standard, SystemVerilog) can be used to describe very complicated logic and force a computer to synthesize the logic. This opens the door to bad design processes, such as people implementing hardware they do not understand. Do not forget the previous labs, hardware is still being described, just at a more abstract level. Keep the modules in your design simple, and then connect modules together in a way to create a more complex system.

5.2 Section Overview

Objective: Design and Implement a clock using SystemVerilog

The following modules have been provided for you in Moodle:

1. Counter - Example 5.5
2. Comparator - Example 5.3
3. Synchronizer - Example 4.20

Use the system Verilog 7-segment display that you created for lab 4. In addition to these files, you will need a parser module to be used to create a clock. The clock should have seconds, minutes, and hours displayed on the 7-segment displays.

5.3 Learning Objectives

The objective of this lab is to understand Hardware Description Languages (HDL) / SystemVerilog (SV)

5.4 Materials

1. Quartus Prime Lite Edition V. 18.0
2. DE10-Lite kit with MAX10 10M50DAF484C7G FPGA
3. USB to USB-B cable

4. The ECE 271 textbook, Digital Design and Computer Architecture by Drs. David and Sarah Harris CH. 4 & 5

5.5 Procedure:

There are 6 steps to digital logic design

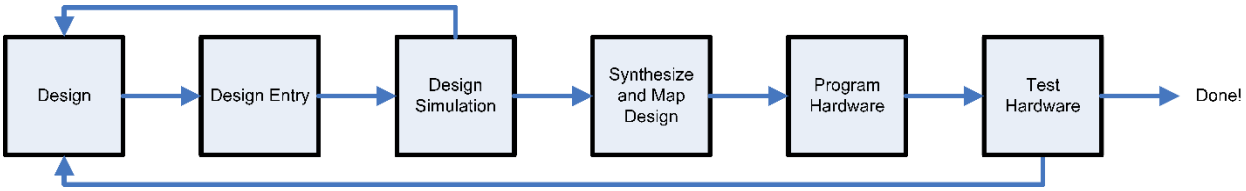

Figure 5.2: Use this process for designing the final project.

1. **Design:** The context of the design is established in this step. The context involves defining the inputs, desired outputs, and all the logic required in-between. In this step, all the minimizations and layout are planned for the design entry process. While this step is not always the longest, it should involve the most thought and effort. This typically requires a complete block diagram showing all the logic blocks and the connections between them, often with written explanations of specific functions.
2. **Design Entry:** The actual drafting of the digital logic design occurs in this step, translating the design from block diagrams and descriptions into the software. This can be accomplished directly by writing HDL code, or graphically by drawing a schematic that a software tool can convert into HDL code.
3. **Design Simulation:** Before committing to hardware, this step tests the design in a controlled computer simulation. If the design does not function as specified in the “Design” step, it is revised.
4. **Synthesize and Map Design:** When the design simulates correctly, the HDL and schematic source files are synthesized into a design file that can be written to the FPGA. This includes assigning the inputs and outputs of the design to IO pins.
5. **Program Hardware:** After the design file is created, it is used to configure the FPGA. Quartus Prime sends a bit stream over the USB-B cable to configure the DE10-Lite FPGA.
6. **Test Hardware:** Verify hardware operation once the FPGA has been programmed. The FPGA should operate exactly as the design predicted, which was verified by the simulation. Synthesis problems, timing violations, or incorrect assumptions about the hardware can require the designer to return to the “Design” step.

5.6 Design

Start by creating a software-level block diagram. This step is critical. Typically, Digital logic design done with an HDL is not simple, and starting with an idea of how information should ‘flow’ through the system is a good way to avoid bad design.

A partial schematic for this section is shown below in Fig. 5.1. This will need to be edited to have the correct values in the counter and comparator. You do not have to make it this way but use Fig. 5.1 as inspiration for your block diagram.

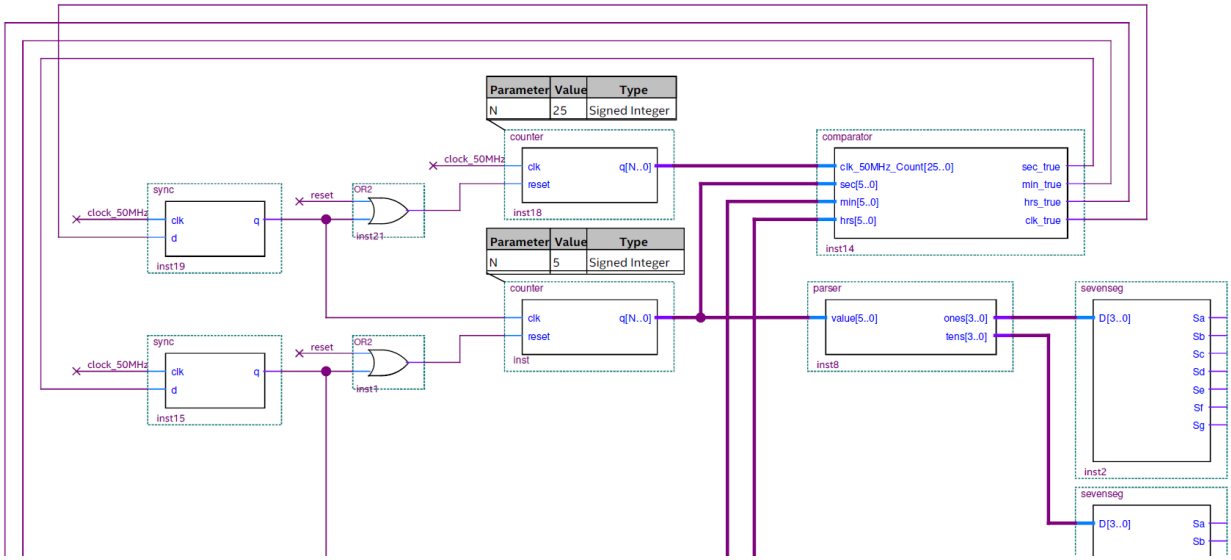

Figure 5.1: Partial schematic for the layout of the clock.

A good software-level block diagram will have system inputs, outputs, and interconnecting logic labeled with a name and bus size. Each module and its ports will also be named.

The only parts the student needs to create from scratch are the parser and the Top-Level Design file.

This parser is a module that takes a large number and splits it up into individual digits. If this operation is done on the Maximum 10-bit number, 1023, you would create 4 (one for each digit) 4-bit (0-15) outputs. The outputs would be 0001, 0000, 0010, and 0011. These four-digit numbers will be fed to the 7-seg decoders.

Since our parser only needs to go to 60 you only need a 6-bit parser for this lab. Create combinational logic (No clock input) for the parser by using math operators (In section 4 of the textbook).

Edit the counter parameters to keep accurate time. Edit the comparator values to trigger reset at 60.

The Top-Level design file is very easy to make if you made a good block diagram. See the last two pages of the System Verilog Starter document for information on instantiating modules at the top level. You can think of instantiating a module a little like calling a function.

5.7 Design Entry.

ENGR 272 is not a SystemVerilog Class, we just use it for defining a few blocks of combinational and sequential logic based on examples from the textbook. Section 4 of the textbook and the ENGR 271 lectures are great resources to learn some fundamental concepts about SystemVerilog.

When you create files to enter the design in, create SystemVerilog HDL files.

Transcribe your block diagrams and logic into SystemVerilog.

Connect all the modules together in the top level.

5.8 Design Simulation

It is required that you use ModelSim to simulate your design. No file conversion is needed, you created your files simulation ready!

5.9 Map Design

Synthesize your design and assign pins to your inputs and outputs using the same process as in Section 1.8.

5.10 Program Hardware

Program the Max10 using the same process as in Section 1.9.

5.11 Test Hardware

Program the device and see if it keeps time. Refer to the Simulation if it does not.

5.12 Checkoff

1. Valid hardware output
2. Valid SV instantiation of textbook modules
3. Valid Simulation

5.13 Study Questions

1. Describe the operation of your Parser?
2. What does the Synchronizer do and why might it be a good idea to use it?
3. How off is your clock? (how much does it differ from a minute on a stopwatch/phone)

5.14 Challenge

Use some buttons to adjust the hours, minutes, and seconds displayed on your clock. Is your user interface intuitive?

4.14 Report Rubric

The reports for the labs are to be formal reports and are to be completed using a word processing application. Please turn in the report to Moodle in a PDF or MS Word format (No Google Docs share links).

Your report should contain at a minimum the following items:

1. **Title**
2. **Date**
3. **Name**
4. **Objectives**
5. **Equipment/Parts/Materials**
6. **Procedure**
    a. **Design**
    b. **Design Entry**
        i. **Block Diagrams**
        ii. **A screen capture of what pin assignments you made in the Assignment Editor in Quartus.**
    c. **Design Simulation**
        i. **ModelSim simulation figures (Screen Captures).**
7. **Complete the study questions in section 5.13**
8. **Challenge - Extra Credit (If this was done)**
9. **Observations**
10. **Conclusion**
11. **References**
12. **Appendix (Source code of all System Verilog files used for your design).**

All images need to have a title and a figure number. For more detailed information about what goes into each of the different sections, see "ENGR 272 Guidelines for Technical Lab Reports" in Moodle.

For an example of what your reports should look like see "ENGR 272 Example Report" in Moodle.

References:

1. Oregon State University ECE 272 Section 5: System Verilog (2019) https://eecs.oregonstate.edu/tekbots/courses/ece272/section5

2. Harris, S. L. & Harris, D. H. (2016) Digital Design and Computer Architecture: ARM Edition 1st Edition, Waltham, MA: Elsevier.