ECE 3544: Digital Design I
Project 3 (Part A) – Design and Synthesis of a Magnitude Comparator System

Read this specification in its entirety before you begin working on the project!

Honor Code Requirements
You must do this assignment individually. The following represent unauthorized aids, as the term is used to define cheating in the Virginia Tech Honor System Constitution:

- Discussing *any aspect or result* of your design with *any person* other than your instructor and the ECE 3544 GTAs. This includes but is not limited to the implementation of Verilog code, as well as the supporting details for that code – truth tables, state diagrams, state tables, block diagrams, *etc.*
- Using *any design element* – including Verilog code – from *any printed or electronic source* other than your course textbook and those sources posted on the course Scholar site.

It is permissible for you to re-use any of your own original Verilog code.

All code submitted is subject to plagiarism checking by MOSS (theory.stanford.edu/~aiken/moss/). Any copying flagged by MOSS will be treated as Honor Code violations and submitted to the Virginia Tech Undergraduate Honor System.

Project Objective
In this project, you will implement a magnitude comparator system from new and existing design units. Since your design will rely largely upon code that you have already written, you will have the opportunity to test (and possibly improve upon) the synthesizability of your Verilog code.

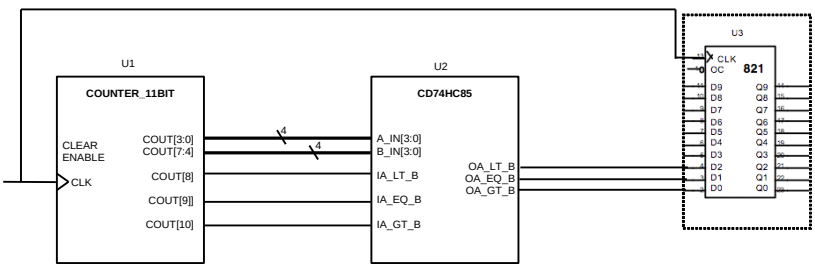
In addition, you will implement your top-level module on the Altera DE1-SoC board, so you will learn how to assign pins of your FPGA to the module's input and output ports, and how to synthesize the module.

Requirements
The DE1-SoC board IS REQUIRED for this project.

You must have the "current version" of ModelSim-Altera Starter Edition (10.3 or higher) and Quartus (Quartus II Web Edition 15.1 or higher) installed on your computer.

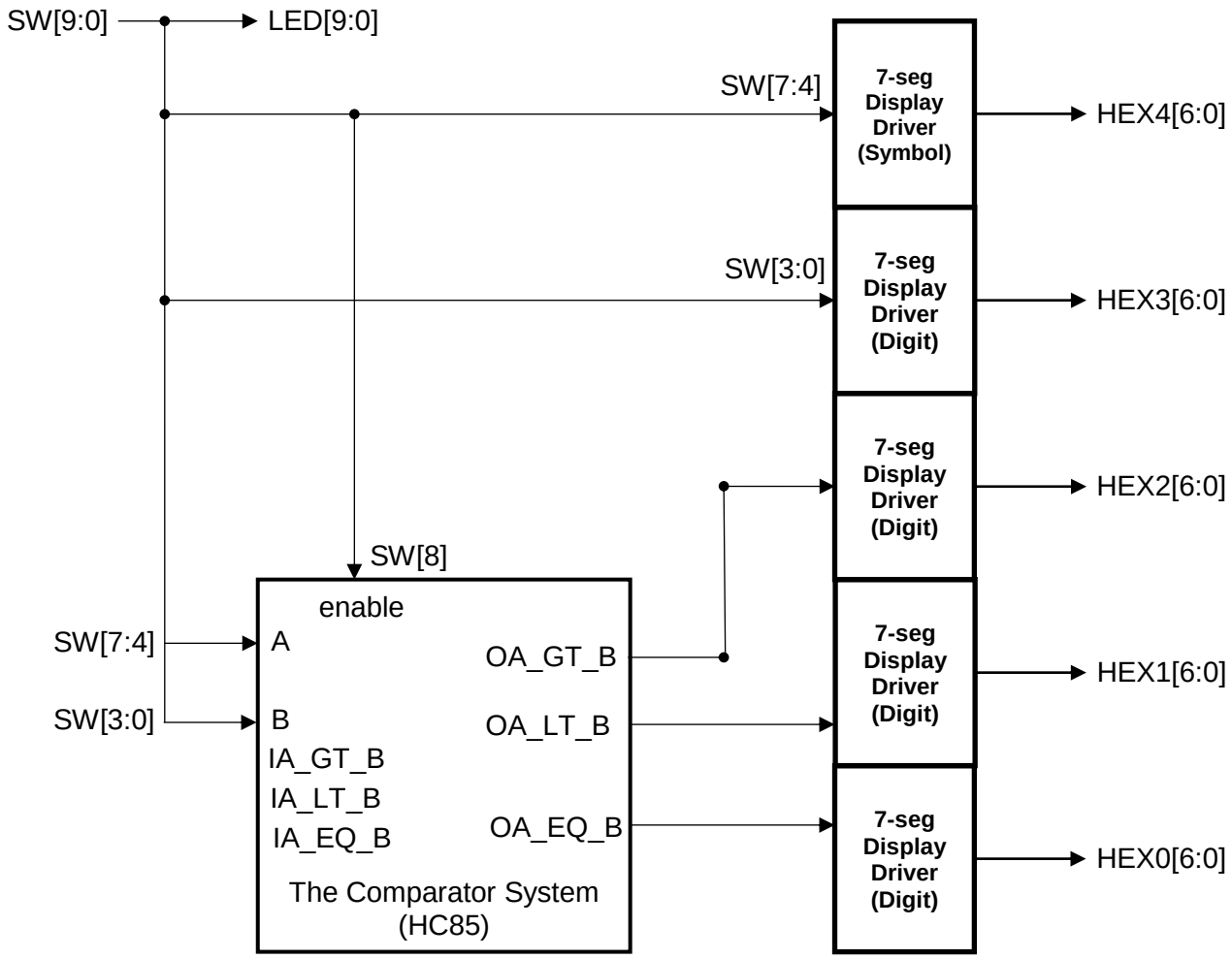
The instructions are done using an appropriate version of ModelSim and Quartus. While the directions may be consistent with other versions of ModelSim and Quartus, you must use the versions indicated above.

Project Description
We did project 2 before. So now our project 3a is more based on it. The project 2 is a digital system shown below:



It has 3 parts, 11 bits counter, CD74HC85, 10 bits register. The 11 bits counter part (U1) is to produce the inputs of U2. And U2 (CD74HC85) compare the inputs data and has the output of A is bigger than B or not. And then the output data will be saved in U3 (10 bits register).

In this assignment, you will design and implement the system represented in the block diagram shown below so that it works on the DE1-SoC board. It mainly based on the HC85 model.



Here we just have 10 switches and 10 LEDs on DE1-SoC. So, we implement 4 switches for input A, 4 switches for input B, one switch for enable. Therefore, you can edit the model HC85 in project 2 to add the enable and set the cascading inputs. When the enable is true or 1, the comparator system will work. And when the enable is false or 0, the comparator system will not work. You can set the cascading inputs to the constant values: IA_GT_B = 0, IA_LT_B = 0 and IA_EQ_B = 1. So, when An = Bn, the system's output will be OA_GT_B = 0, OA_LT_B = 0 and OA_EQ_B = 1.

You will receive a top-level module that defines the input and output interfaces of the FPGA and the DE1-SoC board. The basic top-level module also provides examples of module instantiation. Follow the instructions on pin assignment and programming your board to implement the existing behavior of the top-level module on your DE1-SoC board, then test the board to verify that that existing module performs as expected. *Do not program your board before you have performed the pin assignment.*

After you verify the functionality of the existing module and your board, implement the system described in the block diagram. Create a test-bench to simulate your design. After you verify the functionality of your design in simulation, program your board with the implementation of the final design.

Implementing the Basic Module
Make a working folder for this project and copy the Quartus archive to that folder. You should use the same guidelines for locating and naming your working folder as have been indicated for previous assignments.

Start Quartus. Open the archive file in Quartus by selecting **Project → Restore Archived Project**. You will find an interface for the project. The existing interface does not implement the final design, but it does demonstrate the manner in which board-level inputs and outputs interface with instantiated modules. For now, you don't need to modify any the code in the top-level module or in the modules that have been included for instantiation in the top-level module, but you should review all of the Verilog files to develop an understanding for what the system should do when you implement it on the DE1-SoC board.

Pin Assignment
Before you program the FPGA with the design contained in a Quartus project, you must perform a pin assignment. *This step is **important** – an incomplete or incorrect pin assignment could damage your board.*

The DE1-SoC Board has many peripherals, but we will not use all of them in this assignment. In general, you must assign each bit of the ports in your top-level module (slider switches, pushbuttons, clocks, LEDs, and hex display inputs – all as needed) to a physical pin on your FPGA. The existing top-level module uses the switches, the LEDs, and the hex displays. Your final design will use ten switches and ten LEDs. It will not use any of the hex displays. You may safely ignore the unused outputs, or you may insert code into the top-level module to account for their non-use, but since they are ports of the top-level module, you will still need to account for them in you pin assignment.

To make a pin assignment, you have to know which FPGA pins correspond with the board's peripherals. You can find this information in the DE1-SoC manual, which I have posted for download along with the project files.

Follow the instructions in the Lab Manual section entitled "Pin Assignment for the DE1-SoC" board to assign pins for the top-level module. Remember that you must perform the analysis and elaboration step on an existing top-level module before you can assign pins for that module, and you must re-compile your project each time you make a change to the pin assignment. (You should make a habit of re-compiling your project each time you make a change to any element of your design.)

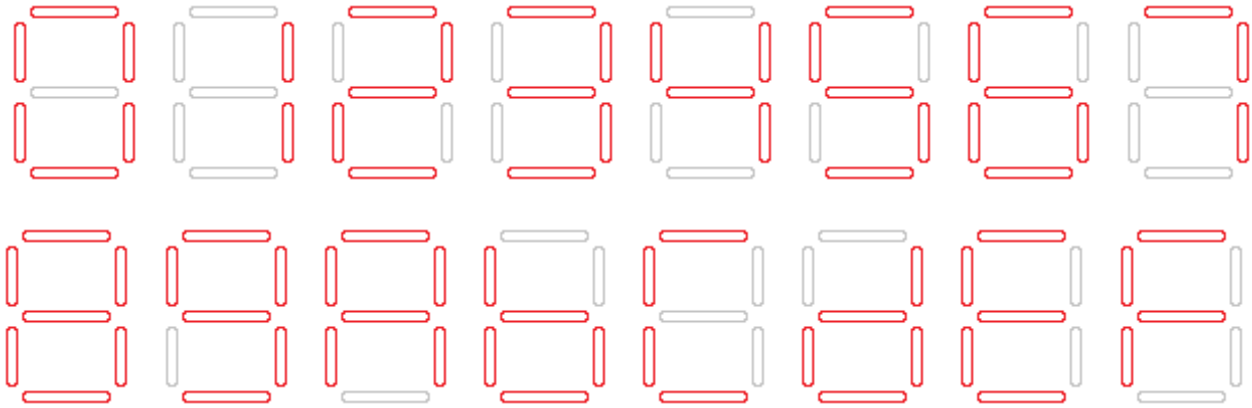
Programming the FPGA with the Basic Module
After you make a complete, correct pin assignment, you can program your board. Follow the instructions in the Lab Manual section entitled "Programming your DE1-SoC Board" to implement the design contained in the top-level module.

After you program your board, test the functionality of the existing design and your board by manipulating the input switches and observing the LEDs and the hex displays. *You should be able to verify that the design and your board are in working order based on your understanding of the behavior of the top-level module and its instantiated components.*

Implementing the Final Design
Make changes to the top-level module as described in the comments of the existing module. Your goal is to implement a system matching the block diagram in the "Project Description" section of this specification.

You have already implemented several of the blocks in the diagram in previous assignments. Use your modified HC85 module from Project 2. *You must use the version of the HC85 module that has no delays in it*. Add any files that you write or carry over from Project 2 to the existing project. I recommend that you copy your files to the working folder that you create for this project. In the **Files** tab of the **Project Navigator**, right-click on the Files folder and choose **Add/Remove Files in Project**. In the window that appears, choose the **...** button to browse for the files you want to add, then click **Add** once you have selected them.

As far as the system implementation is concerned, you have already created models for several of the required elements in previous assignments. Use your seven-segment display module from Homework 3 to implement.



```
module proc7segdecoder(hex_digit, hex_display);
  input  [3:0] hex_digit;
  output [6:0] hex_display;

  // INSERT YOUR CODE HERE

  // END INSERT

endmodule
```

Since our goal this semester has been to write code that will synthesize correctly in addition to simulating correctly in ModelSim, this project will provide you with the opportunity to test the synthesizability of code that you have already written. If Quartus gives synthesis errors for any of your modules, you should determine what elements of your code have caused the synthesis errors and correct them.

To create new modules in Quartus, choose **File** → **New**, and then choosing **Verilog HDL File** under **Design Files**. Write the module as you have written modules in ModelSim previously. After you save the file, you must add it to the project. To compile the module, click **Analysis & Synthesis** under **Compile Design** in the **Tasks** window. (Performing the Analysis & Synthesis task should suffice for any situation where you make changes to a Verilog module and wish to re-compile the Verilog source. If this does not suffice, you may need to re-compile the entire project. Since compilation can take a long time, you should only do it in situations where you have to do it, *e.g.*, when you are ready to program your board with your design.)

When you write your modules (as you did in Project 2), you should write them as though they can take any input that is appropriate for the port declaration you made. But when you instantiate your modules in the top-level module, you are effectively connecting them to the switches and LEDs that you want to

use to provide the I/O to your top-level module. Use the existing top-level module as an example of how to connect inputs and outputs to the instantiated module, then follow this logic when you do the same things in your actual implementation.

Simulation

The project files include the framework for a test bench. In general, you can create a test bench in Quartus by choosing **File** → **New**, and then choosing **Verilog HDL File** under **Design Files**. Remember to add the file to the existing project, and to repeat the Analysis and Synthesis step after you have finished making changes to the file that you add.

Follow the instructions in the document "Simulating Quartus Modules Using ModelSim" to simulate the behavior of the top-level module.

Write your test bench to simulate different operating cases of the comparator system. A sequence similar to the validation sequence (or the validation sequence itself) would be a good start. *The validation test sequences are not nearly exhaustive. While you need not produce an exhaustive set of test cases in your test bench, you should not rely on the validation sequence as the sole test of your system's correctness.*

Programming the FPGA with the Final Design

After you make sure of your comparator's functionality via simulation, you are ready to program the FPGA with your final design. You should not need to assign the pins a second time, but you might wish to open the Pin Planner to verify that your pin assignment remains unchanged. Follow the instructions in the document "Programming your DE1-SoC Board" to implement the final design on your FPGA.

After your program your board, test the functionality of your final design by manipulating the input switches and observing the LEDs and the hex displays. While the validation sheet contains one sequence of input tests that you can try, the validation test is not exhaustive. As you did with your test bench, you should perform an amount of testing that you consider adequate for verifying the working order of your design.

Project Submission

Write a report describing your design and implementation process. Your report should include waveforms showing the correct behavior of your design.

I have included the validation sheet that the GTAs will use to test your design after the submission deadline. *You do not have to go to the CEL to have your project validated.* Instead, you should use the information in the validation sheet as one basis for testing your design before you submit your project.

Submit your report as an electronic document on Scholar. Put your report and source files into a single .zip file. You must submit all elements of the project by the deadline to avoid late penalties; the assignment is only as timely as the last element to be submitted.

Your project submission on Scholar should include the following items:

1. Your project report in PDF. The included cover sheet should be the first page of the project. The validation sheet should be the second page. I have provided these pages as Word documents that you can insert into your project report before you convert it to PDF.

2. A Quartus Archive containing the source files for your top-level module, any modules that the top-level module requires to function, and your test benches for the top-level module.

   To create the Quartus Archive, choose **Project > Archive Project** after you complete your implementation. When prompted for a name for your archive, the default archive name will be the same as the original archive. *Append your Virginia Tech PID to the end of the filename*. Make certain that you submit the archive that you create – the one containing your solution to the project – and not the one that I provided to you – the one that only contains the initial files.

Your top-level module may be tested with our own secret test bench, so it is important that you use the module declaration provided with the archived project. You must also include the source files for every module required for your top-level module. Failure to do so will result in a grade of 0 for that portion of the project.

Grading for your submission will be as described on the cover sheet included with this description