ECE 3544: Digital Design I

Project 3 (Part A) – Design and Synthesis of a Binary Converter 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 binary converter 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 14.1 or higher) installed on your computer. When I installed Quartus 14.1, it updated version 10.1d of ModelSim ASE to version 10.3c, but the first time I started ModelSim ASE 10.3 it picked up from the same point that I had left ModelSim ASE 10.1d the last time that I had run it.

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**

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.

**7-seg**

**Display**

**Driver**

**(Digit)**

HEX1[6:0]

**7-seg**

**Display**

**Driver**

**(Digit)**

HEX2[6:0]

**7-seg**

**Display**

**Driver**

**(Digit)**

HEX3[6:0]

**7-seg**

**Display**

**Driver**

**(Symbol)**

HEX4[6:0]

**7-seg**

**Display**

**Driver**

**(Digit)**

HEX0[6:0]

Dual Converter

SW[9:0]

{2’b0, SW[5:4]}

SW[3:0]

LED[9:0]

SW[6]

SW[6]

SW[5:0]

conv\_in

conv\_sel

conv\_out[5:4]

conv\_out[3:0]

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 I have 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 all ten switches (but three of them will only have the job of driving LEDs), all ten LEDs, and five 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 document “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 document “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.

The system is such that the values on SW[5:0] represent a six-bit number. The bits might represent a binary number, or they might represent the lower six bits of a binary-coded decimal number whose two MSBs are assumed to be 0. The value of SW[6] indicates which of the two possibilities is the case: if SW[6] = 0, SW[5:0] represents a binary number, while if SW[6] = 1, SW[5:0] represents a BCD number. *You may assume that the value applied on SW[5:0] will always represent a valid input value that corresponds to an output value that can be displayed in the other format.* (As a practical matter, this means that the input values will be valid representations of the numbers from 0 to 39, inclusive. This should be consistent with your work in Project 2.)

As you might expect, you will need to use the 74184 and 74185 models from Project 2 to convert the input into the corresponding value for the **other format** on the output. But as you might now be thinking, this is not as simple as just instantiating the two modules – you will have to use the value of SW[6] to determine which of the converters represents the desired output. (As a reminder: **Conditional instantiation is not a valid modeling approach.**)

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 drivers from Homework 3 to implement the blocks called “7-seg Display Driver (Digit)” in the block diagram.
* You will need to write a module to implement the “7-seg Display Driver (Symbol)” block. This model is similar to the display drivers you have already written, except that this driver uses a single input bit to cause one of two symbols to be displayed:

|  |  |
| --- | --- |
| C:\Users\Jason Thweatt\Desktop\ECE 2504\Projects\Validation Pictures for Project 1\b.bmp | C:\Users\Jason Thweatt\Desktop\ECE 2504\Projects\Validation Pictures for Project 1\d.bmp |
| SW[6] = 0  (Since the input represents a *binary* number) | SW[6] = 1  (Since the input represents a *BCD* number) |

module symbol\_driver\_YOURPID(select, hex\_driver[6:0]);

input select;

output [6:0] hex\_driver;

* You will need to implement the “Dual Converter” block. However, instead of writing it completely from scratch, you should instantiate instances of your 74184 and 74185 modules, and then write the logic that applies the correct output from one of the converters as the output of the dual converter. *You must use the versions of the 74184 and 74185 modules that have no delays in them.*

module dual\_converter\_YOURPID(conv\_sel, conv\_in, conv\_out);

input conv\_sel;

input [5:0] conv\_in;

output [5:0] conv\_out;

You do **not** need to write a special 7-segment display driver that only takes in the top two bits of the converter output to display a digit. Take a clue from the block diagram as to what you can do when you instantiate the dual converter module into your top-level module.

When incorporating existing files into the current project. I recommend that you copy them to the working folder that you create for this project. Then, 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.

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 either of these modules, you should determine what elements of your code have caused the synthesis error and correct them. All of the modules in this project are combinational circuits, so *if you choose to write a procedural model* *to describe anything*, you should ensure that you use proper synthesizability principles for describing it.

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.)

**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. 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