ECE 3544: Digital Design I
Project 3 (Part B) – Design and Synthesis of a Synchronous Finite State Machine

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 – 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 Canvas 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 take an existing synchronous finite state machine module and modify it to add features. You will implement your top-level module on the Altera DE1-SoC board, so you will gain practice with assigning pins of your FPGA to the module's input and output ports, and with synthesizing modules.

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

You must have the current version of ModelSim ALTERA STARTER EDITION and Quartus II Web Edition 16.1 installed on your computer. The instructions are done using these versions of ModelSim and Quartus. While the directions may be consistent with other versions of ModelSim and Quartus, you must use the versions indicated above.

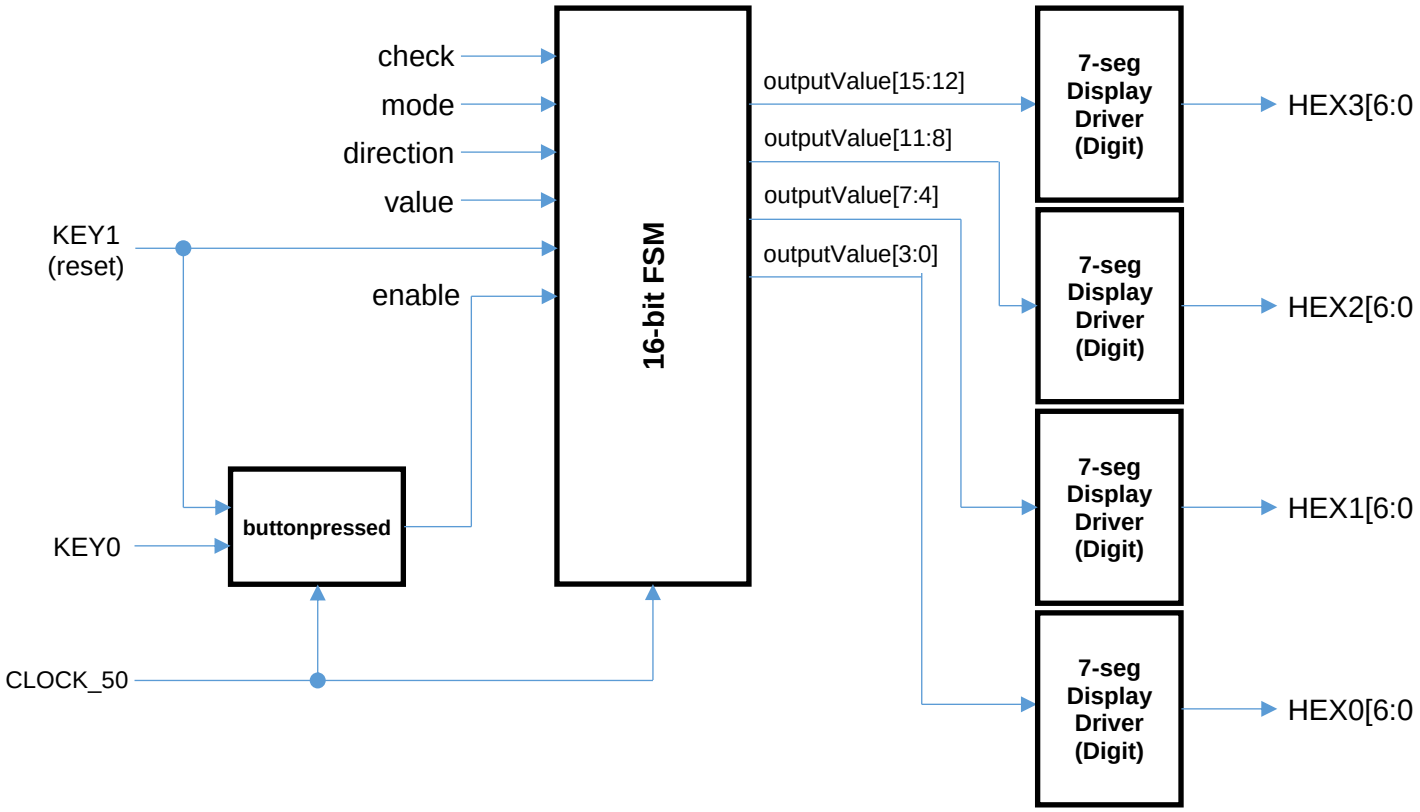
Using the Documentation
I have added extensive comments to the modules that you will use as the basis for completing this assignment. You should use these comments in conjunction with the code structures and general code organization to learn more about how to properly formulate state machines in Verilog. Future projects will require you to correctly represent state machines in Verilog; this project represents a good first example of how to do that.

Pay special attention to comments that appear in ALL CAPS. They are meant to draw your attention to particular aspects of the Verilog code. In general, you should remove these comments as a part of documenting the version of the code that you submit.

Study the `fsm16bit` module carefully to ensure that you understand how it operates before you make any changes to it. *Do not modify the* `buttonpressed` *state machine.* Study it to ensure you understand how it operates. ***Provide a state diagram that models the behavior of the*** **`buttonpressed`** ***module in your report.***

Project Description

In this assignment, you will design and implement modifications to an existing finite state machine and synthesize it so that it works on the DE1-SoC board. The block diagram shown below represents the top-level module.



You have been given the basic module; you should implement the changes described in the specification and create a test-bench to simulate your design. 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.*

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. You will find the interface for the project, which already has a basic 16-bit counter. The top-level module already instantiates the `buttonpressed` and `fsm16bit` modules. It connects the output of the `buttonpressed` module to the `enable` input of the `fsm16bit` module. Both modules use the FPGA clock as a common clock.

Because the counter is meant to be operated using the board's pushbuttons, the system is implemented using two communicating state machines. The `buttonpressed` state machine generates a one clock cycle-wide enable pulse each time the user presses and releases KEY0. The `fsm16bit` state machine generates the count sequence. The counter is active only on clock cycles when its enable is asserted.

The existing counter will function on the board if you provide instantiations of a correctly-working seven-segment decoder. Obtain one of the seven-segment decoder modules that you wrote previously for homework. It should not matter whether you use a version that is modeled using continuous assignment or using procedural assignment. I recommend that you copy it to the working folder for this project. Follow the instructions in the comments of the top-level module to instantiate the seven-segment decoders. Connect the appropriate outputs of the `fsm16bit` module to the inputs of the corresponding seven-segment decoder.

Pin Assignment
Follow the instructions in the section of the Lab Manual 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.

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 section of the Lab Manual 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 appropriate input switches and pushbuttons and observing 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
Modify the `fsm16bit` module so that it functions as follows:

- The FSM operates on the positive edge of the FPGA's 50 MHz clock (CLOCK_50).
- The asynchronous active-low `clear` (KEY1) resets the FSM.
- The `enable` input (KEY0) allows the FSM to perform one operation each time KEY0 is pressed and released. Holding `enable` low does nothing; `enable` must go low and then high again for the FSM to complete an operation.

It should be clear that the enable input is NOT THE FSM CLOCK. We could design and implement a synchronous FSM that only uses the clock to trigger operations, but we would not be able to see distinct values on the hex displays if they were changing at a rate of 50 MHz. Therefore, we use the enable to restrict which clock triggers actually cause display changes. In pseudocode:

On a positive clock edge:
If enable = 1
Perform the FSM operation determined by the slide switches.
Else
Leave the counter state unchanged.

- The `check` input (SW6) determines the value of the FSM state:
  - If `check` = 0, then the 16-bit FSM state should take on the value of the last four digits of your student number, expressed as binary-coded decimal numbers. Your implementation need not calculate these values; you may encode the values as constants.
  - If `check` = 1, the FSM state should appear as determined by the operations specified below.

- The `mode` input (SW5) determines whether the FSM acts as a counter or as a circular rotation:
  - If `mode` = 0, then the FSM is in "counter" mode.
  - If `mode` = 1, then the FSM is in "circular rotation" mode. All rotate operations are logical circular rotation. For example, 1110010111100101 left rotate by 1 bit will generate 1100101111001011; 1110010111100101 right rotate by 1 bit will generate 1111001011110010.
- The `direction` input (SW4) specifies the direction of the count and shift operations:
  - If `direction` = 0, the FSM should decrement (count down) if it is in "counter" mode, or circular rotate the FSM state to the right by 1 bit if it is in "circular rotation" mode.
  - If `direction` = 1, the FSM should increment (count up) if it is in "counter" mode, or circular rotate the FSM state to the left by 1 bit if it is "circular rotation" mode.
- When the FSM is in "counter" mode, the `value` inputs (SW[3:0]) determine the value by which the FSM state is incremented or decremented. The `value` inputs have no function when the FSM is in "circular rotation" mode.

The inputs `check`, `mode`, `direction`, and `value` appear in the top-level module provided to you but are unused. Part of your task is to incorporate them into the existing FSM module to implement the behavior described above. Update the comments as needed to reflect the changes that you make to the existing counter.

Simulation

The project files include the framework for a test bench. Write your test bench to simulate different operating cases of the counter. 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 use your test bench to produce an exhaustive set of test results, you should not rely on the validation sequence as the sole test of your system's correctness.*

Follow the instructions in the section of the Lab Manual entitled "Simulating Quartus Modules Using ModelSim" to simulate the behavior of the top-level module.

You need not use the test bench to verify the operation of the top-level module, as it will be incomplete when you receive it. Instead, you may want to begin by making a new test bench that verifies the existing counter and your understanding of how it works. Then you can proceed to editing the counter and testing your changes. Ensure that your final test bench runs through ALL possible counter operations and input combinations.

Programming the FPGA with the Final Design

After you make sure of your counter'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 section of the Lab Manual entitled "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.

Your report should include a state diagram that models the `buttonpressed` module.

Your report should address your conclusions on the proper way to structure finite state machines in Verilog. Use the following questions to guide your discussion:

- *Into how many procedural blocks should the logic of a synchronous FSM be divided?*
- *The top-level module contains two synchronous FSMs: the* `buttonpressed` *module and the* `fsm16bit` *module. Do the two modules contain the same number of procedural blocks? If not, how does each module accomplish the functionality of a FSM?*
- *If there is more than one procedural block involved, in what ways do the blocks differ? What are the responsibilities of each block?*

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.

Your project submission on Canvas should include the following items:

1. Project report in Word or 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.