ECE 3544: Digital Design I

Project 4 – Design and Synthesis of a Synchronous Finite State Machine

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

*Read the Assignment page on Scholar where you downloaded the project materials!*

**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 Scholar site.

*It is permissible for you to re-use any of your own original Verilog code. It is also permissible to use the* keypressed *module that was provided to you in Project 3B.*

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 synchronous finite state machine that operates as a dual-function stopwatch. You will gain proficiency in the following:

* using techniques to derive state machine models from natural language specifications
* using state machine models to produce synthesizable Verilog modules
* designing and implementing synchronous counters having various control and output functions
* designing interacting synchronous finite state machines
* implementing larger-scale synchronous systems via synthesis from smaller finite state machines

You will implement your top-level module on the Altera DE1-SoC board, so you will gain greater practice with assigning FPGA pins to the module’s input and output ports, and with using Quartus to synthesize hardware from Verilog models.

**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 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.

**Project Description**

In this assignment you will design and implement a two-function stopwatch so that it works on the DE1-SoC board. The block diagram below is a *rudimentary* representation of the top-level module.

HEX5 (100 ms)

HEX4 (10 ms)

HEX3 (1 ms)

HEX2 (100 us)

HEX1 (10 us)

**Stopwatch**

**System**

KEY[3] (reset)

KEY[2] (mode)

KEY[1] (action)

KEY[0] (stopgo)

CLOCK\_50 (clock)

HEX0 (1 us)

LEDS (mode)

*Submit a detailed block diagram representing your implementation of the system as part of your report.*

*In Quartus, choose* **Tools > Netlist Viewers > RTL Viewers** *to view the RTL schematic that represents your top-level Verilog module. You may submit a screenshot of this schematic if you believe that it is sufficiently detailed, simple to follow, and descriptive of the structure of your implementation. It might interest you to view the structure of the instances in your top-level module; clicking the boxed plus sign on any instance allows you to see the structure of that instance.*

For this project, the basic module that you will receive only represents the inputs and outputs of the DE1-SoC board. The system will not require all the inputs and outputs that the top-level module declaration includes on its port list or that are listed in the port declaration. You may use the unused inputs and outputs for any testing or debugging purpose that you decide is appropriate. You may also remove them from the port list of the module declaration and from the port declaration.

You will be responsible for writing all the Verilog that implements the design elements that comprise your stopwatch system. *You may re-use the* keypressed *modules from Project 3B. You may also re-use any of your own original Verilog code.*

**System Interface**

The system has the following inputs:

* Clock / CLOCK\_50 – The DE1-SoC board has four 50 MHz clock signals. Use CLOCK\_50 as the clock for the sequential elements of your design. Do not use the switches or pushbuttons as clock signals.
* Pushbuttons: The system uses four pushbuttons as follows.
  + Reset / KEY[3] – The Reset button is an active-low system reset. Pressing and releasing the Reset button should return the system to its initial state.
  + Mode / KEY[2] – If the system is outside one of its functions, pressing and releasing the Mode button toggles the mode choice. If the system is inside one of its functions, pressing and releasing the Mode button exits that function and returns to the “mode select” level – specifically, to the mode choice for the mode that the user just exited.
  + Action / KEY[1] – Depending on the level of operation, pressing and releasing the Action button accepts the choice made at a particular level of operation, or performs other functions relating to that level of operation.
  + Stopgo / KEY[0] – When the system is in a mode and the timer is running, pressing and releasing the Stopgo button pauses the timer it is running, and starts the timer if it is stopped or paused. Pausing the timer does not cause it to reset. Restarting the timer should cause it to resume counting at the place where it stopped.

The system has the following outputs:

* Hex displays: The system uses five hex displays as follows:
  + Timer value / HEX5, HEX4. HEX3, HEX2, HEX1, HEX0 – The timer value shows the current value of the timer. Respectively, the four displays show units of 100 ms, 10 ms, 1 ms, 100 us, 10 us, and 1 us. The timer value should appear in a manner that is consistent with a decimal counter.

*The time scale of changes on the lowest-order digits would not normally be visible on a seven-segment display. The time scale of changes for this assignment has been adjusted specifically to facilitate observing these changes in simulation. Note that if the timer had a base time unit of 1 ms, it would take 50,000 clock cycles just to see one change in the base unit. It would take 50,000,000 clock cycles to see one second’s worth of change. While it still might be burdensome to simulate a timer at this time scale, the burden should be lower than it would be with a longer time scale of changes.*

* + Mode Indicator / LEDS –The mode indicator identifies the mode and general level of operation of the system. Use LED[9] to indicate that the timer is “configured” for up-mode. Use LED[8] to indicate that the timer is “configured” for down-mode. Use LED[7] to indicate that a mode has been chosen. Use LED[6] to indicate that the timer is “running” in either up-mode or down-mode. Use LED[5] to indicate that the timer is paused or stopped in either up-mode or down-mode.

*I encourage you to use the remaining LEDs for debugging purposes. Using them to display the state value for some state machine is an excellent usage.*

**Method of Operation**

This diagram shows the general way the pushbuttons change the mode and level of operation. It is not necessarily meant to represent a complete state diagram, but you might consider using it to help make the state diagram that you must make to control your system.

MODE

STOPGO

ACTION

STOPGO

ACTION

MODE

MODE

ACTION

STOPGO

ACTION

STOPGO

ACTION

MODE

MODE

ACTION

RESET

ACTION

* The initial “state” of the system is one where it is waiting to enter the up-counter mode. In this “state,” LED[9] should be on. The counter value should read 000000.
  + Pressing and releasing reset in *any* “state” causes the system to enter this “state.”
  + Pressing and releasing mode in this “state” causes the system to enter a “state” where it is waiting to enter the down-counter mode.
  + Pressing and releasing action in this “state” causes the system to enter the “state” where the up-counter is ready to go.
* In the “state” where the up-counter is ready to go, LED[9] and LED[7] should be on. The counter value should read 000000. Whenever the system returns to this “state”, the counter value should reset and be ready to count up from 000000.
  + Pressing and releasing mode in this “state” causes the system to enter the “state” where it is waiting to enter the up-counter mode.
  + Pressing and releasing stopgo in this “
* In the “state” where the up-counter is running, LED[9], LED[7], and LED[6] should be on. The least significant digit of the timer value should increment every microsecond. The counter is a decimal counter, so a value like 000009 should be followed by 000010 and not by 00000A. When the counter shows 999999, it should roll over to 000000 on the next count.
  + Pressing and releasing mode in this “state” causes the system to enter the “state” where it is waiting to enter the up-counter mode.
  + Pressing and releasing action in this “state” causes the system to enter the “state” where the up-counter is ready to go.t
  + Pressing and releasing stopgo in this “state” pauses the up-counter.
* In the “state” where the up-counter is paused, LED[9], LED[7], and LED[5] should be on. The counter value should stop on the value that was present when the user paused the counter by pressing the stopgo button.
  + Pressing and releasing mode in this “state” causes the system to enter the “state” where it is waiting to enter the up-counter mode.
  + Pressing and releasing action in this “state” causes the system to enter the “state” where the up-counter is ready to go.
  + Pressing and releasing stopgo in this “state” causes the up-counter to run. When it does so, the counter should pick up from the value that it held when the counter was paused. The value should not “skip” any time based on how long it might have been paused.
* In the “state” where the system is waiting to enter the down-count mode, LED[8] should be on. The counter value should read 000000.
  + Pressing and releasing mode in this “state” causes the system to enter a “state” where it is waiting to enter the down-counter mode.
  + Pressing and releasing action in this “state” causes the system to enter the “state” where the down-counter is ready to go.
* In the “state” where the down-counter is ready to go, the user must select a duration for the down-counter. LED[8] and LED[7] should be on. The counter value should initially read 000000. Whenever the system returns to this “state”, the counter value should reset and be ready to have the user set a duration.
  + Pressing and releasing mode in this “state” causes the system to enter the “state” where it is waiting to enter the down-counter mode.
  + Pressing and releasing action in this “state” causes the counter value to increment by one millisecond. This value represents the duration of the down-counter. The value should only increment by one each time the user presses and releases the action button. If the count value is 999000, pressing and releasing the action button should cause the value to roll over to 000000.
  + Pressing and releasing stopgo in this “state” causes the down-counter to run.
* In the mode where the down-count is running, LED[8], LED[7], and LED[6] should be on. The least significant digit should decrement every microsecond. The counter is a decimal counter, so a value like 000010 should be followed by 000009 and not by 00000F.
  + Pressing and releasing mode in this “state” causes the system to enter the “state” where it is waiting to enter the down-counter mode.
  + Pressing and releasing action in this “state” causes the system to enter the “state” where the down-counter is ready to go.
  + Pressing and releasing stopgo in this “state” pauses the down-counter.
  + When the counter shows 000000, it should *stop*. When the counter stops at 0, LED[6] should turn off and LED[5] should turn on.
* In the “state” where the down-counter is paused, LED[8], LED[7], and LED[5] should be on. The counter value should stop on the value that was present when the user paused the counter by pressing the stopgo button. The counter value should not reset.
  + Pressing and releasing mode in this “state” causes the system to enter the “state” where it is waiting to enter the down-counter mode.
  + Pressing and releasing action in this “state” causes the system to enter the “state” where the down-counter is ready to go.
  + Pressing stopgo in this “state” causes the down-counter to run. When it does so, the counter should pick up from the value that it held when the counter was paused. The value should not “skip” any time based on how long it might have been paused.

In general, if the effect of pressing a button in some “state” is not mentioned, then pressing that button should have no effect. This listing of “states” is not exclusive and need not represent the entire set of states of your actual state machine. Add states as you determine is appropriate to implement correct behavior of the system, but do not change any element of the system behavior described here.

**Design Tips**

* Design smaller aspects of this system individually. Implement and test one before moving on the next one. *Don’t try to implement the whole system before testing any of it*.
* Make a state diagram to represent the way button presses change the mode and level of operation. *Include the state diagram for your “system controller” in your report.*
* Decompose the design into communicating finite state machines, e.g., a set of counters that receive direction from your system controller and might also provide information to the system controller, or counters that supply outputs that control the inputs of other counters.

|  |
| --- |
| **This is important!**  A timer is basically a set of interacting counters that run at different frequencies. While this might suggest that they should have different clocks, this is a *very bad idea*! Introducing multiple clock domains into a system is a good way to add clock skew, which in turn detracts from the stability of the system.  Using a counter with an enable allows the designer to use a common clock on all such counters, while allowing the action in the counter to be triggered on particular occurrences. This should remind you of what you did in Project 3b – it only now bears observing that a button press doesn’t *have* to be the trigger for the enable. |

* Make a block diagram of the system before you write any of the module. This will help you settle the way the various state machines should interact. *Include a detailed block diagram representing your implementation of the system as part of your report. You need not break-out the structure of every block in your diagram, but a good guideline is to include a block for every Verilog module you write and instantiate in some other module.*
* Use the keypressed module from Project 3B to generate a one-clock-cycle enable signal when a pushbutton is pressed and released.

**Project Submission**

Write a report describing your design and implementation process.

* Discuss the decisions you made about implementing the elements of your design.
* Include a detailed block diagram of the interacting units in your system.
* Include a state diagram for the system controller, and the state diagrams of other units that might require them.
* Include waveforms showing the correct behavior of your design.
* Devise a simple test (presumably involving repeated trials) to test the accuracy of your timer. Include the results of your test and comment on the accuracy of your timer.
* Document any additions you make for debugging purposes and describe your motivation for each debug operation.

Your project submission on Scholar 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.