Contents

Project Objectives

Project 3B introduced a simple FSM circuit and called for the addition of additional behavior. The starting circuit incremented a "state_counter" synchronously with a positive clock edge when KEY0 was released. An asynchronous reset button (KEY1) would reset the "state_counter" to 0.

After studying the starter module circuit's behavior, additional functionality was to be added. The circuit was to drive four 7-segment-displays to display hex digits and perform certain operations depending on the values of SW[6:0]. These operations are summarized in the table below.

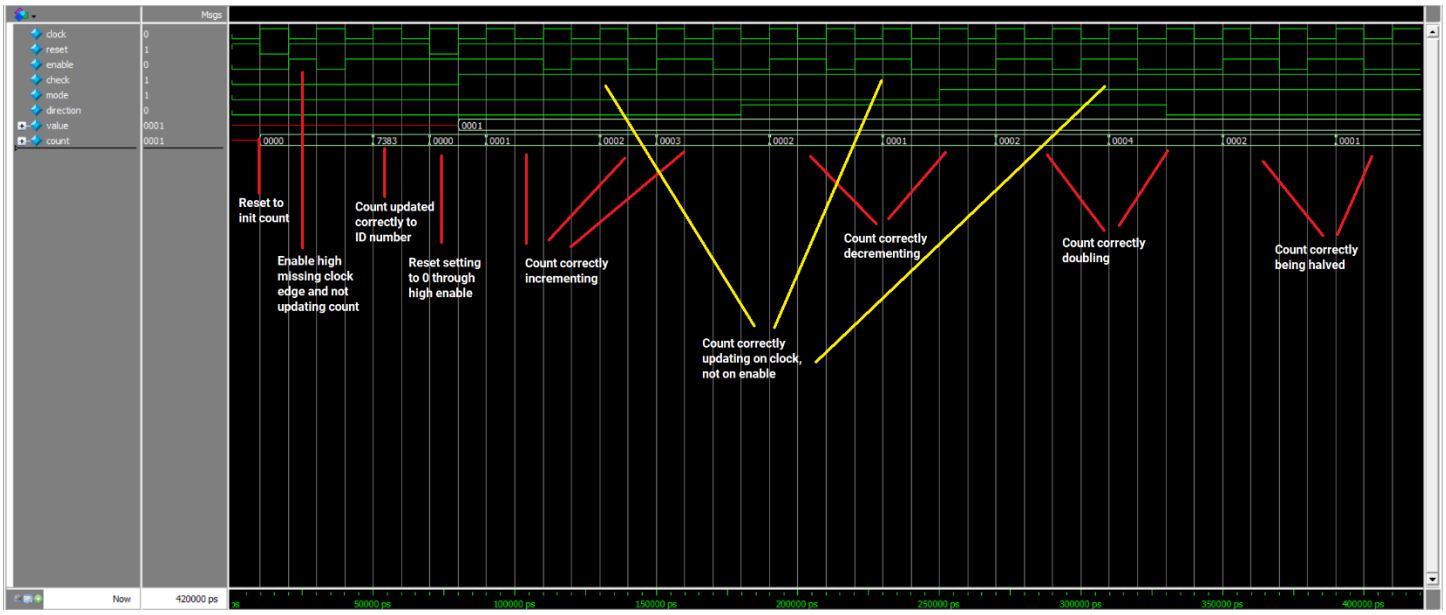
Value	SW[6] "Check"	SW[5] "Mode"	SW[4] "Direction"
0	Display last 4 digits of student ID (ignores Mode and Direction)	Counter mode	Subtract SW[3:0] (value) / Shift right
1	Perform other operations	Circular Shift register	Add SW[3:0] (value) / Shift left

Design and Implementation Process

The first design was a series of nested if statements that effectively acted as 2-to-1 muxes. These if statements would check each bit of the "select" signal (Check, Mode, Direction) in series, and branch depending on the value. After these 3 branching paths, the correct operation would be performed.

While this approach functioned correctly, Quartus didn't detect it as a finite state machine. So the nested if statements were changed into a case structure, where the "select" signal was updated and then depending on the value, would perform the appropriate action. While Quartus didn't this as an FSM either, the code is much more readable. Parameters were added during this process as well to increase the readability and modularity of the code.

While very basic tests were run on the nested if version, a more robust testbench was written after implementing the case structure. The waveforms of this testbench can be found below. After this testbench showed a properly functioning circuit, the model was flashed to the FPGA and more thorough testing was done on the FPGA. Every function was tested and worked correctly.



Testbench Output

Specific Specification Questions

- *Based on the examples of finite state machines presented, how should the procedural blocks representing logic of a synchronous FSM be divided?*

  Procedural blocks for FSMs should be divided into combinational logic to determine the input to the state machine, the state block(s), and output blocks (in the case of a Mealy machine). This is due to the differing sensitivity lists and the nature of those lists (clock edge triggered or not).

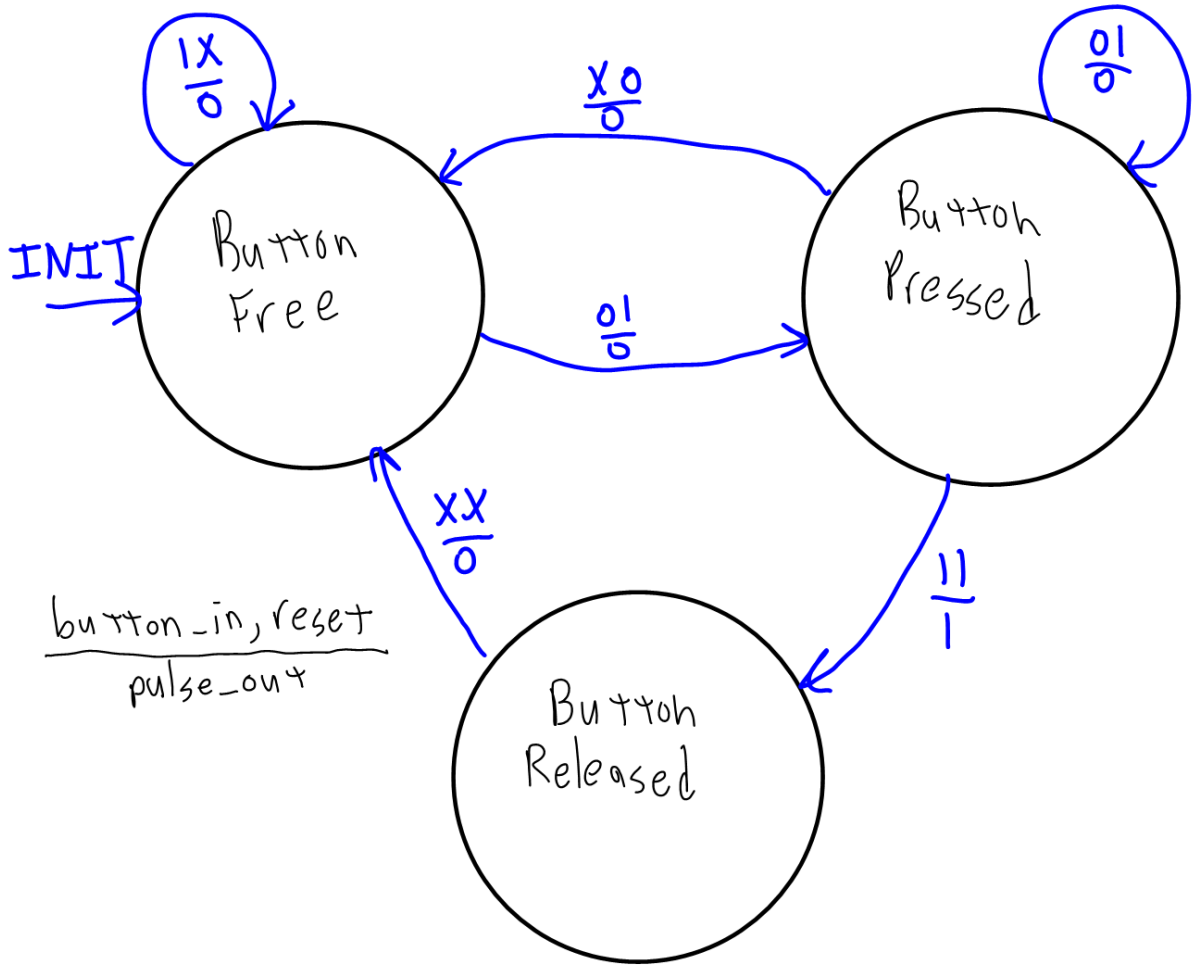
- *The top-level module contains two different synchronous finite state machines: the* buttonpressed *module and the* fsm16bit *module. Do the two modules contain the same number of procedural blocks? If not, how does each module accomplish the functionality of a FSM? If there is more than one procedural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

  The buttonpressed module uses three procedural blocks: one to control being reset or advancing to the next state; one to take in the input of the button status and determining what the next state should be; and changing the output based on the present state. This is due to the fact that the buttonpressed module is a Mealy machine.

  In contrast, the fsm16bit has one always block to determine the output and a continuous assignment to handle the output. The procedural block in fsm16bit behaves similarly to the three in buttonpressed but is wrapped into one block. Every positive edge, if the enable signal from buttonpressed is high, the "select" signal is updated and then the output is determined to an intermediate variable, before being assigned to the output. Upon a reset signal (also part of the sensitivity list of this block), the output is simply set to 0.

- *The buttonpressed module uses KEY0 as its input to obtain the enable pulse. However, the reset does* ***not*** *also use a buttonpressed module to produce its output. In your report, explain why you think this is the case.*

  The reset is asynchronous, so no module is required (nor desired) to sync the reset with a clock edge.



State Diagram of Button Pressed module

Conclusion

This project was a useful introduction to Verilog FSMs. While I felt pretty confident on the theory, in practice FSMs look very different to the clean diagrams we usually draw and discuss. I had to double-check myself on a few occasions when examining buttonpressed and multiple times when writing the case structure (and the nested ifs before that) for fsm16bit. It wasn't showing up as a state machine inside Quartus which had me second-guessing myself, but ultimately had me looking over detail of the circuit and confirming that it was working as intended. The real meat of the project didn't take very long but I ended up researching FSMs a lot over the course of doing this project. It was good finding out that I was weaker on practical FSM design than I realized and can now start to correct it.