**ECSE 222: Lab 3**

Presented to: Prof. Psaromiligkos

Authors:

Edward Latulipe-Kang (260746475)

Ding Ma (260871301)

April 12, 2019

**The state diagram of your FSM.**

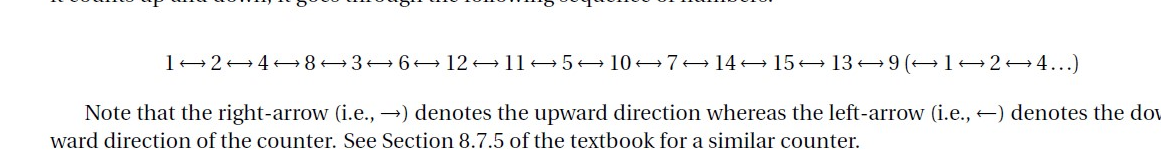


Figure 1: State Machine Possible States

The tests shown in the lab instructions are by figure 1. The each number can be represented by a letter which will be in alphabetical order, by convention. Thus, variable A represents 1, B represents 2 and so for.

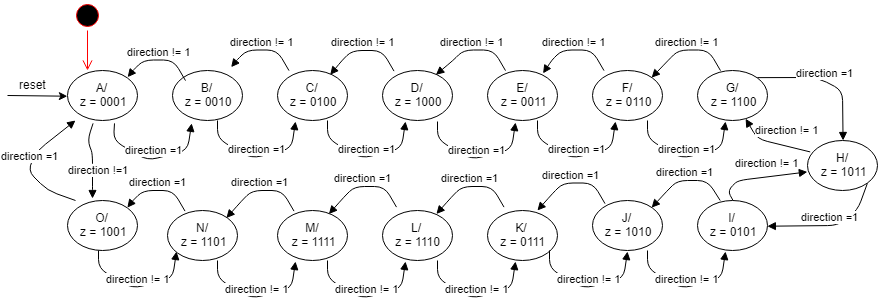


Figure 2: State Diagram

For our state diagram, A is the corresponding state from the instructions. Z is the binary representation of the value in that state. If the direction is 1, the state diagram will change to the next state. Consequently, if it is not 1, it will go to the previous state. For instance, if the current state is A, the next state is B if direction = 1 or it will be O if direction != 1. Also, the starting state (initiation) our state machine is at state A so as the reset.

1. **A description of the FSM circuit.**

The FSM circuit can be broken down into 3 parts.

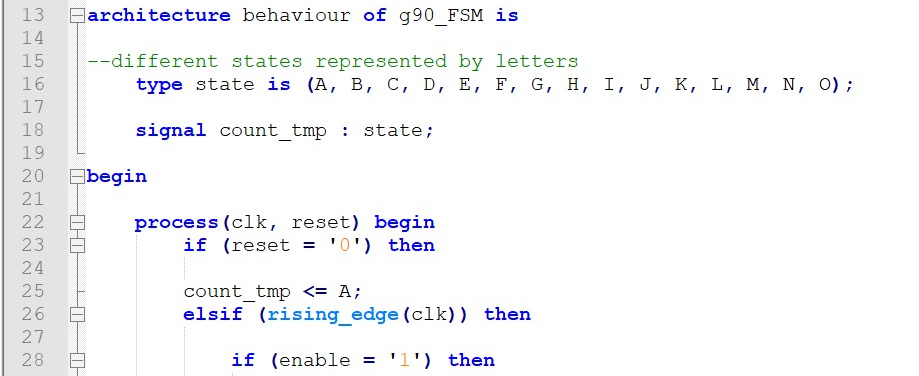


Figure 3: Part 1 of State Machine - Initialization of count and variables

The first part of our FSM is the initiation of the count and checking the variables. For instance, in order to start going through the states, reset = ‘1’. Then, we have to count at every rising edge cycle of the clock. Finally, we have to make sure that the clock is enabled in order to be able to start counting. The count\_temp signal is linked to the state which are from A to O.

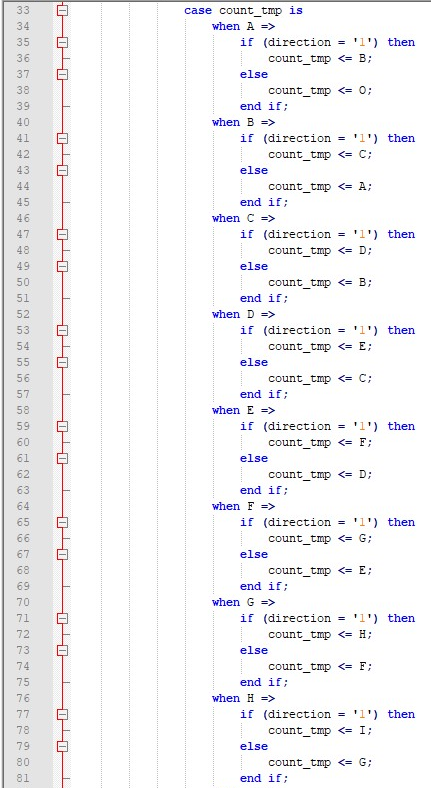
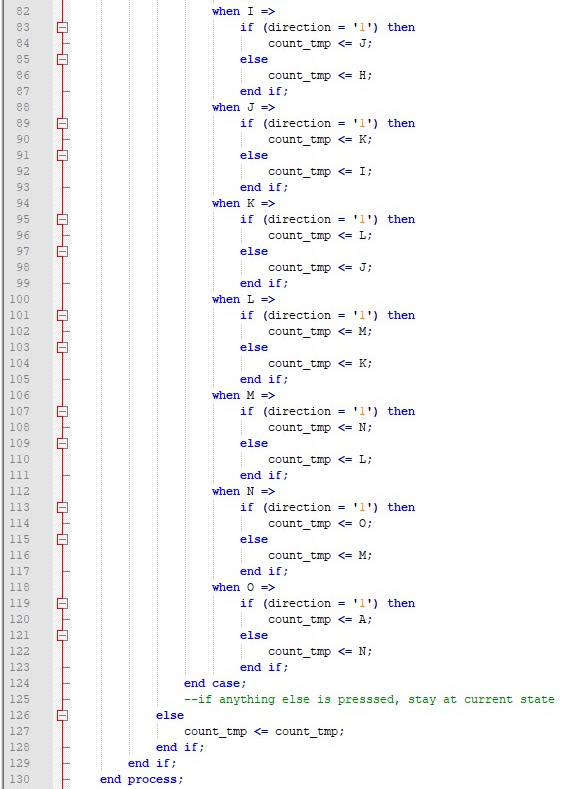
 Figure 4.1 and 4.2 are from the textbook but we had to modify the code for this lab. Basically, it says that when the direction = ‘1’, the clock is counting forward. Else, it is counting backward. That is what the big case statement is done, anoter way of doing this case statement is with a giant if statement but this adds a lot of complexity to the code. Line 127 of figure 4.2 indicates that when the clock is not counting, it should remain at its current state.

Figure 4.2: From State I to State O and other cases

Figure 4.1: From State A to State H

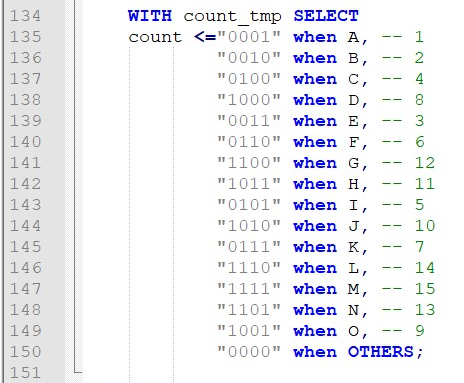


Figure 5: Conversion of States to Binary

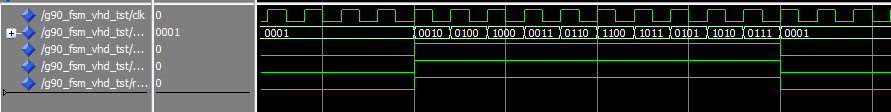
Finally, our state machine needs to convert the letters that we assigned into the actual number for this lab based on figure 1 on page 1. For instance, A is “0001” and it represents 1 in a 4 bit unsigned binary form. Figure 5 is basically the conversion chart between the letters in figure 4.1 and 4.2 and the binary representation of those numbers.

1. **A discussion of how the FSM circuit was tested, showing representative simulation plots. How do you know that these circuits work correctly?**

The FSM circuit was tested by simulating its behavior. Observing both figures, we notice that our testbenches accurately track the clock. Given that this worked in past labs, this isn’t so much a point of interest. Instead, we want to observe the state change, that is to say, the second element, and to determine if it iterates through the states, whether or not it’s moving in the desired direction, and finally what conditions it appropriately functions under.

When observing both files, we realize that the defined state machine properly iterates through all desired states in the desired fashion. To elaborate, the simulation properly illustrates that the iteration only occurs when both enable and reset are active (i.e. rising edge). Given the logic of the written code, this demonstration describes the states only iterating, when the enable is “on”, and reset is “off”. In this regard, our simulation works correctly.

Furthermore, we can observe, that for both test files, our simulation does in fact iterate through its options in the desired fashion outlined in the lab. Depending on what the direction is set to (i.e. 0 or 1), the state machine will iterate either forward, or backward, respectively. Observe the figures below.

 Figure 6.1: FSM simulation with direction set to 0

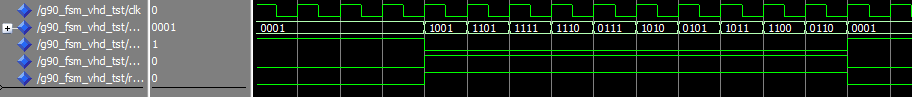


Figure 6.2: FSM simulation with direction set to 1

1. **A description of the multi-mode counter circuit.**
2. **A discussion of how the multi-mode counter circuit was tested on the FPGA board.**

The way that we tested the circuit is straight forward. Once the code is loaded onto the board, we pressed start while keeping the direction = 1. We followed what was displayed on the board with the figure 1 which was provided with the lab report. Then, we flipped the direction = 0; we looked if the FSM would go into the other direction. We had to make sure that when the state hits O it switches to A and same for the inverse. Also, we tested if the stop function to see if the board would remain at its current position. Finally, we made sure that the reset button will return the state machine into its starting state which is A (0001).

To further test our board, we randomly flipped the direction switch in order to test if the clock would go to the previous or the next state.

1. **A summary of the FPGA resource utilization (from the Compilation Report’s Flow Summary) and the RTL schematic diagram for the multi-mode counter circuit. Clearly specify which part of your code maps to which part of the schematic diagram**

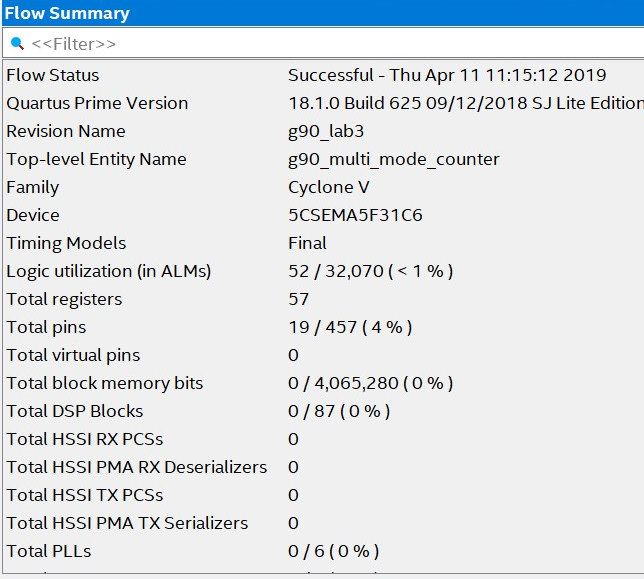
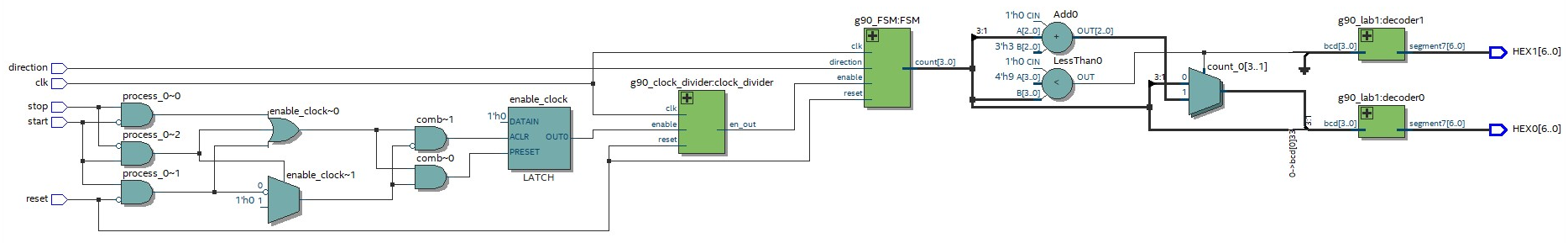


Figure 6: Flow Summary

Our program account for less than 1% of the logic utilization. This is means that our board uses little resources in order to implement our FSM.



6

5

4

3

2

1

Figure 7: RTL Viewer

The RTL schematic indicates what kind of component we have used in order to make our circuit. It is basically a drawing of our circuit. The components in green are those that have been Port map into the multimode counter. If we click on the “+”, we can see an accurate breakdown of that circuit.

1. signal enable\_clock : std\_logic:= '0';

What this is the signal in order to enable our FMS to start counting. By default, it is kept at 0 because we want to start counting when the user presses start. The logic circuit before enable\_clock is to check the inputs such as preserving the state when stop is pushed.

1. clock\_divider : g90\_clock\_divider PORT MAP(enable => enable\_clock, reset => reset, clk => clk, en\_out => reset\_clock);

This imports the logical circuit of our clock divider. See figure 8 for a more detailed representation of this circuit.

1. FSM : g90\_FSM PORT MAP(enable => reset\_clock, direction => direction, reset =>reset ,clk => clk, count => count\_fsm);

This imports the FSM logic into our multi mode counter, same logic as before. See figure 9.

1. if(count\_fsm > "1001")then

count\_0 <= std\_logic\_vector(unsigned(count\_fsm)+"0110");

count\_1 <= "0001";

else

count\_0 <= count\_fsm;

count\_1 <= "0000";

end if;

#4 allows the detection of a binary number that is over 9. This ensures that the binary number is converted into BCD format in order to display it properly on the HEX displays. There is a more detailed picture of this circuit at figure 10.

1. decoder0: g90\_lab1 PORT MAP(bcd => count\_0, segment7 => HEX0);

There are 2 instances of the seven-segment decoder because it has to display numbers from 0 ranging up to 14 in decimal format. Therefore, the binary numbers are needed to be converted into BCD format for the display to use.

1. HEX1 : out std\_logic\_vector(6 downto 0);

HEX0 and HEX1 are the 2 instances of the seven-segment display.

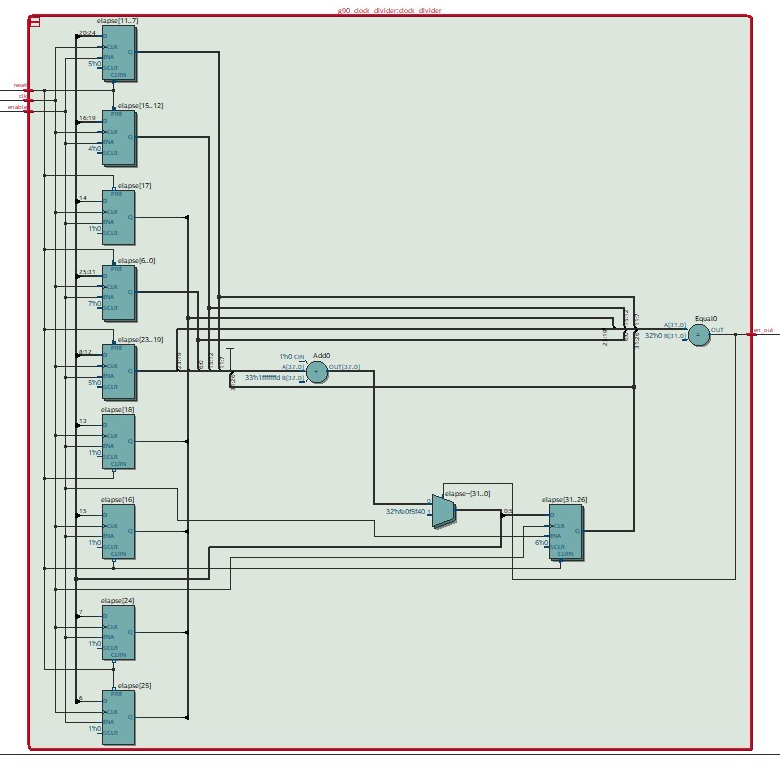


Figure 8: Inside g90\_clock\_divider

Figure 8 is the circuit representation of our clock divider.

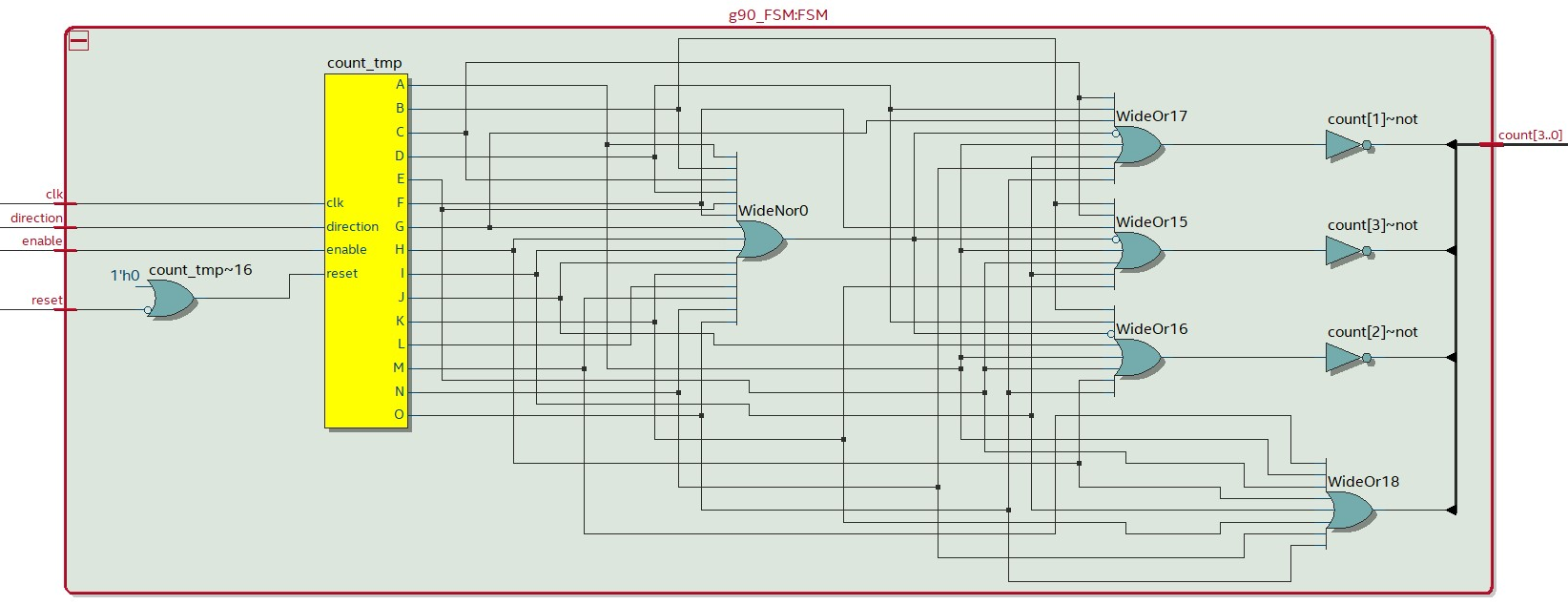


Figure 9: Inside g90\_FSM

Figure 9 is the circuit representation of our FSM code.

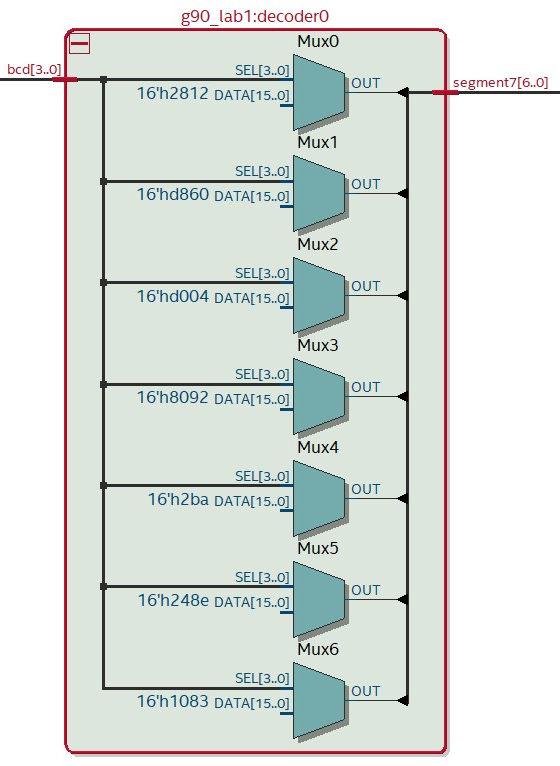


Figure 10: Inside g90\_lab1 (binary to BCD)

Figure 10 is the circuit representation of our binary to seven segment decoders.