McGill UNIVERSITY

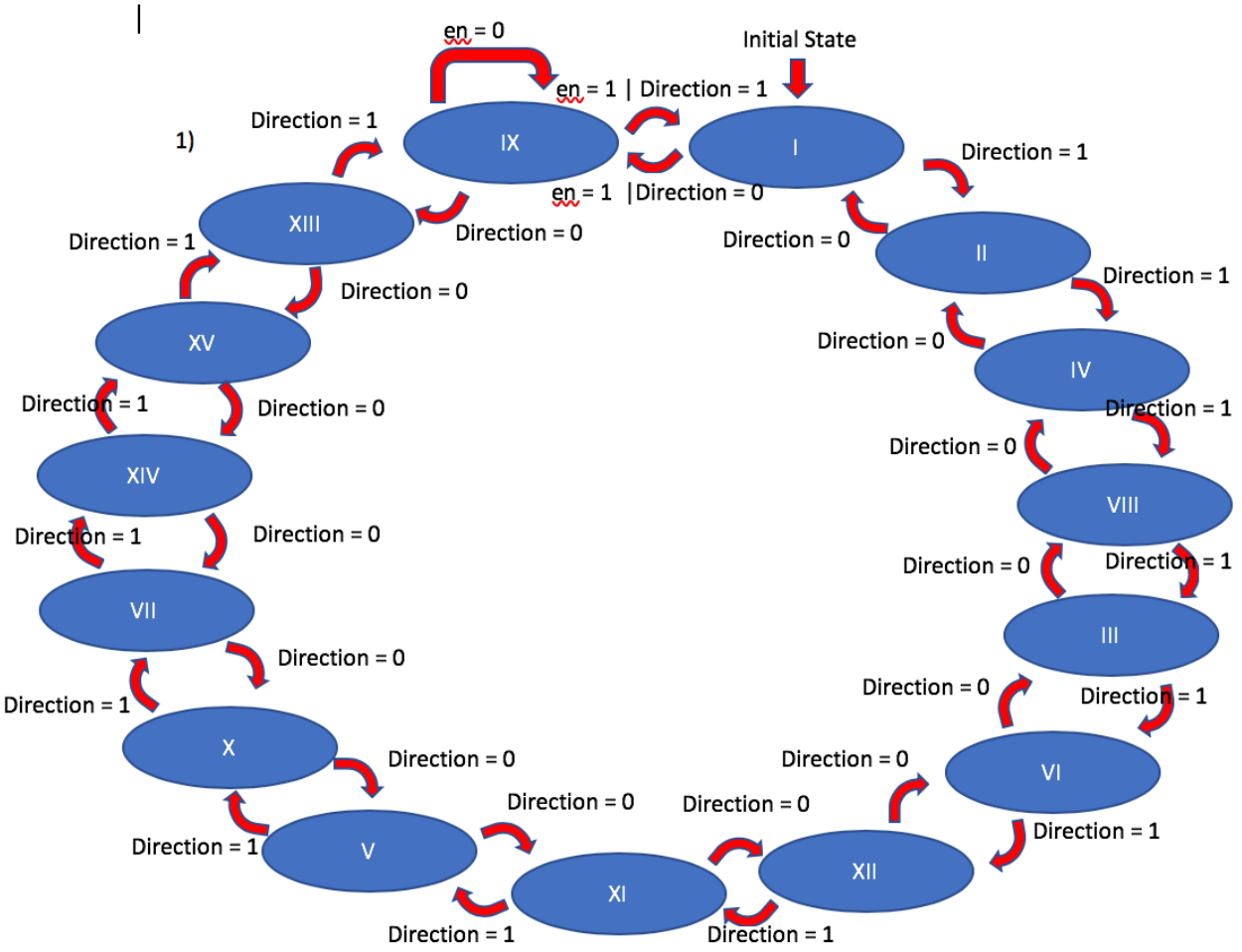
ECSE 222 Digital Logic Lab #3

Alex Gruenwald, 260783506

Kaustav Das Sharma, 260772982

Group #50

The state diagram of your FSM.



Important to note: This entire FSM relies on en (enable). If at any point stop is pressed, the enable is set to ‘0’ which means on each clock cycle, the same state is maintained as seen in the state “XI” (this is the same for each state however it was not included due to a lack of space). When we consider the next state, enable must be activated to a ‘1’. If it is activated, the single bit direction variable is considered to see whether the state is moving in a forward or backward direction. If the direction is 1, the FSM will move in the ‘forward’ direction (clockwise) while a direction of 0 will result in a backward direction (counter clockwise). For our FSM, we did not include information about reset. Reset is dependent on the direction variable and will either set the state to the initial position ('I') or ('IX').

A description of the FSM circuit

The FSM circuit is used to keep track of as well as determine the next state for each cycle of the period. In order to do so, 4 important variables must be inserted into the circuit from the main circuit – multi mode counter. These defined variables are very similar to what was used for the counter with the exception of the addition of the direction variable that takes into consideration the direction of which the cycle shifts (which state is next). The clk (clock) is linked to the clock_divider circuit that has been defined previously, with a 1 second cycle period. The enable works with the direction variable where when the positive edge of the clock is hit, something will happen with the state on the next cycle. This change depends on the two variables – direction and enable – which are both defined by the user as they interact with the Altera Board. Enable is based on whether start or stop has been activated. If stop has been pressed, the enable will be 0 (active high) meaning at the clock edge, no change will occur, and the state will be maintained. If enable is activated (1) then a change will occur. This next state will depend on the direction variable. If a 0 is activated, the next state will be to the state previous to the current. If direction is 1, the next state will be the state directly after the current state. And finally, we look at the reset, which resets the clock to the beginning ('I') if the direction is 1, while a direction of 0 will cause it to reset at to 'IX' (the end of the enumeration).

To conclude the discussion on the entity, the count is the output of the FSM circuit which stores the value in binary of the state that the circuit is currently on based off of the temporary value state. This count will be manipulated and be displayed onto the Altera board display.

To begin our discussion of the actual circuit, we first define the enumeration of all the states. We decided to use roman numerals for our state representation because both participants are familiar with the symbols (also easy and short). These numerical representations represent each state of the circuit as seen in figure 1.

```
type state is (I, II, IV, VIII, III, VI, XII, XI, V, X, VII, XIV, XV, XIII, IX);
```

Figure 1: state representation in roman numerals

Once the enumeration of all the states are made, we must declare what the initial state is for the circuit is ('I'). By default, the initial state is always set to the first state 1 ('I'). This initial state is stored in the variable defined as temporary and changes throughout the cycle of the circuit. The output count is based off the temporary value at the end of the cycle.

Describing the process (we need a sequential circuit), we set the asynchronous variables direction and reset (user manipulated inputs) as well as the clk which defines the cycle period. Once these are set, we can begin the process of figuring out what the next state will be.

Before checking the state however, we must consider the asynchronous features first – reset and enable. These will define if the state needs to change or not. If reset is 0 (active low) the result is a reset of the states based on the direction set at that moment. If the direction was set to a 1 ('I'), the begin of the enumeration will be the new state, whereas if the direction is a 0, the initial state set will be the end of the enumeration 9 ('IX').

If reset is not activated, then we check if a rising edge of the clock is active. If not, then the temporary state maintains its state and waits for the next check. If a positive edge was reached, then we must begin our long check of what the next state will be. First, we must check if enable was activated to see if the start button or stop button had been pressed (enable = 0 or 1). If the stop is activated, then the enable is 0 and nothing should happen. If however, start was active (enable = 1) then a change in the state will occur depending on the direction variable.

For our long state check, we traverse through each case of the given states (In roman numerals) to find which state is currently held. Once the current state case is found, it checks the direction to find out whether the next state should be the next in the enumeration list or the previous state shown in figure 1. Once this is determined, the temporary value is set to the new state (a state defined by the roman numerals).

Once this cycle is complete and a new state is chosen (or maintained) the temporary value must be sent to the multi-mode counter in order to be processed and sent to the 7_segment_decoder to be displayed on the Altera Board. In order to assign the chosen roman numeral states with actual bit values, we set up a selected signal assignment check. If the current state is found, we set the output value (count) the value that corresponds to the temporary roman numeral state. This is sent back to the main circuit – mmc.

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

To test the FSM circuit, we started off by creating a template from Quantus Prime, by setting the FSM circuit as the top level entity. The testbench was then written such that it tests the output for the asynchronous (start/stop) enable and the direction. The simulation was then ran on ModelSim Altera, and produces the output wave displayed in **Figure 2.** The values listed for the FSM were also converted to hexadecimal. The FSM circuit should be going through and outputting the values of the different states, and as seen in **Figure 2,** our simulation produces exactly that.

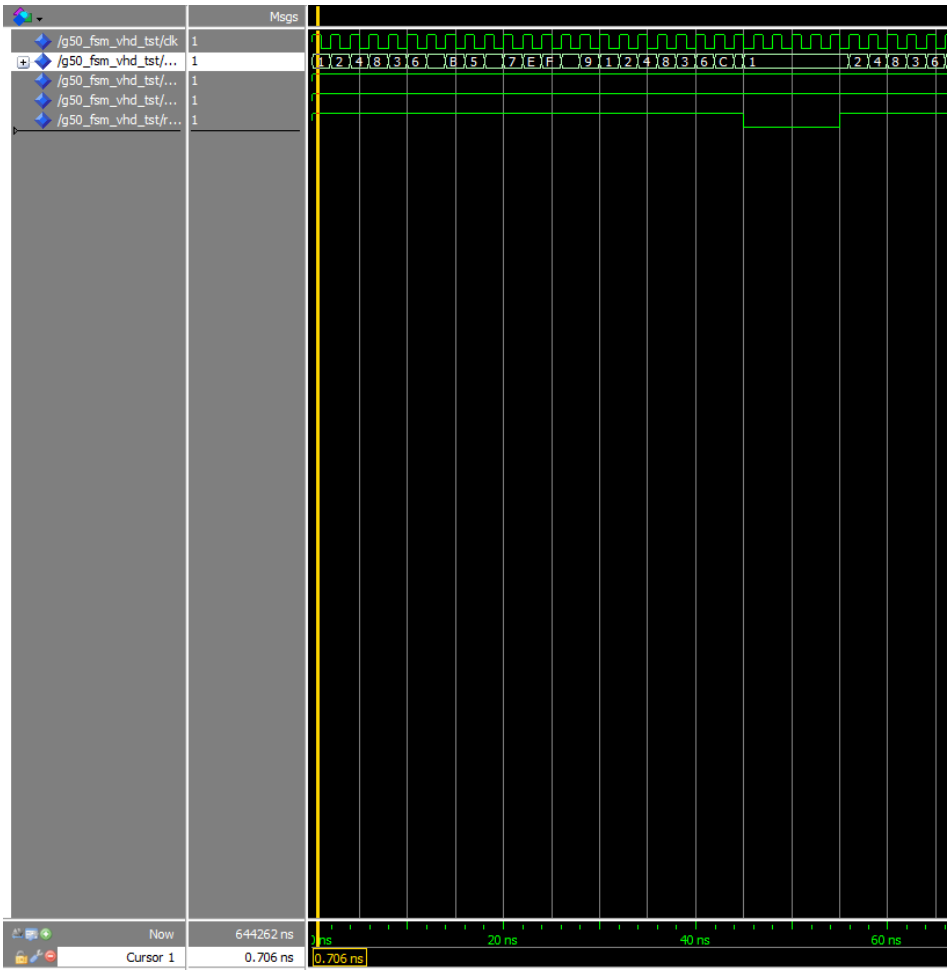


Figure 2: FSM simulation plot

A description of the multi-mode counter circuit.

To discuss the overview, the MMC circuit is the head master for all the operations. The Altera board directly communicates to the circuit where the MMC then communicates to the imported circuits – clock divider, FSM, and the 7_segement_decoder. When all operations are successfully, a fully working state counter should be displayed on the board.

To begin our discussion of the code, we discuss the entity provided in the lab outline. The important inputs into this circuit are the start, stop, reset, and direction which are all asynchronous and can be toggled by the user. None of these variables rely on the clock cycle and can be activated at any point. In the case of start, stop, and reset, the response can be seen immediately whereas the direction switch will only be read at the next clock cycle. The final two variables are the outputs of the MMC circuit which are used for the display – HEX0 and HEX1 (figure 3 shows their locations).

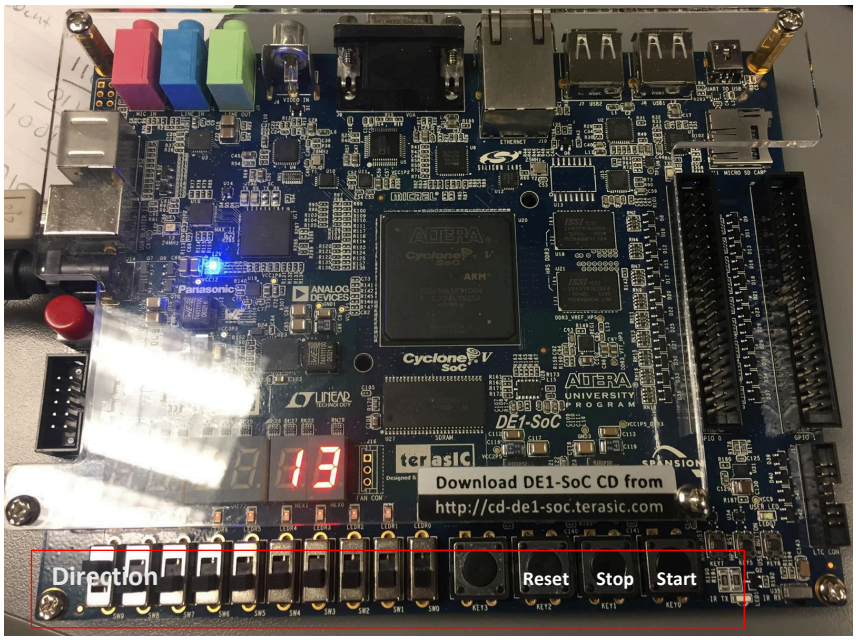


Figure 3: Altera Board variable assignments

For the architecture of the MMC circuit, we have to define all of the circuit imports required for the circuit to function properly. The imports needed are the 7_segment_decoder, the FSM and the clock_divider (these are explained in previous labs). Before we end the architecture and begin our process, we must also define a few variables to be used as imports for the circuits imported.

Most importantly, we define the mmc_enable. This variable is directly responsive to the board where a push on the start button sets a 1 for the variable (start is active low '0') and a stop sets a 0. We must also consider other import variables, such as the cd_enable which tells the MMC circuit when the appropriate 1 second has passed to then check for a change in the FSM. We must also set a temporary 4-bit variable that stores the state in the FSM. Finally, we initialize two very important variables, that will send the appropriate value to the 7_segment_decoder to be displayed (these variables store the addition of the value taken from FSM and adds 0110_2 6_{10} to ensure no letters are displayed).

After the architecture is set we send the variables defined in MMC to their appropriate imported circuit, so they respond to the same events as the main MMC circuit. Ex. mmc_enable is sent to the clock_divider to tell the circuit that the stop button has been pressed, halting the clock.

Once the set-up is complete, we start our process by checking the asynchronous variables start and stop which will determine if the rest of the circuit can operate or not. If start is 0, the mmc_enable is set to 1 (active high). If stop is 0, the mmc_enable is instead set to 0 (inactive).

After the asynchronous check, we then have to set the appropriate values to the two displays. The two variables (display0_BCD and display1_BCD) are sent to the 7_segment_decoder to be displayed. In order to ensure the MMC circuit only displays numerical digits opposed to letters, we have to implement BCD's. By doing so, we avoid the issue of having a value greater than 9 (A, B,..). If the output of the FSM (temporary) is greater than 1001_2 (9), then we add 0110 (6) to ensure we avoid receiving a letter display. We set the first display to the new right most value and set the second display to a 1 (if the value is greater than 1001_2, then the value must have a 1 in the second position of the number). If the value is smaller than 1001_2, the display variable is directly set to the temporary value from the FSM (second display is 0).

With the appropriate values set, the variables are sent to the 7_segment_decoder and is then displayed on the Altera Board.

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

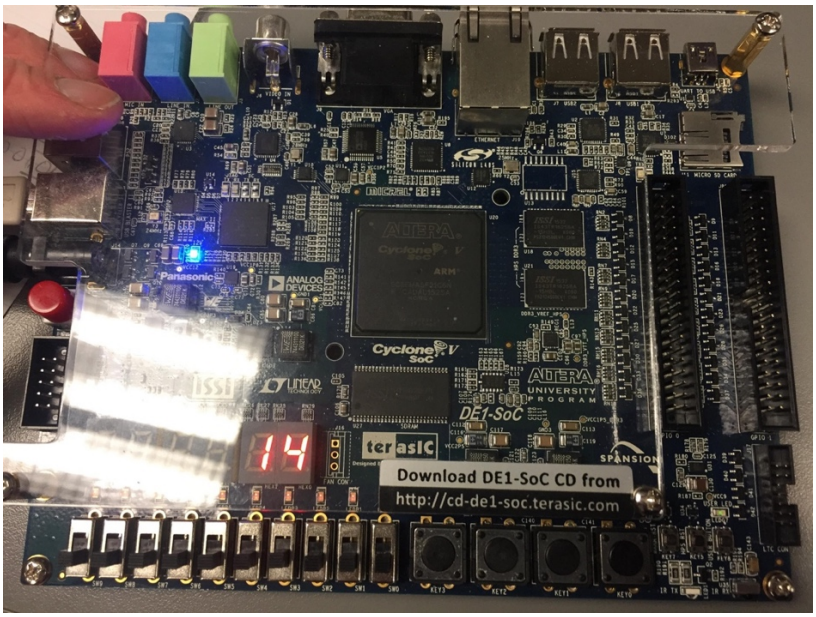


Figure 4: End state

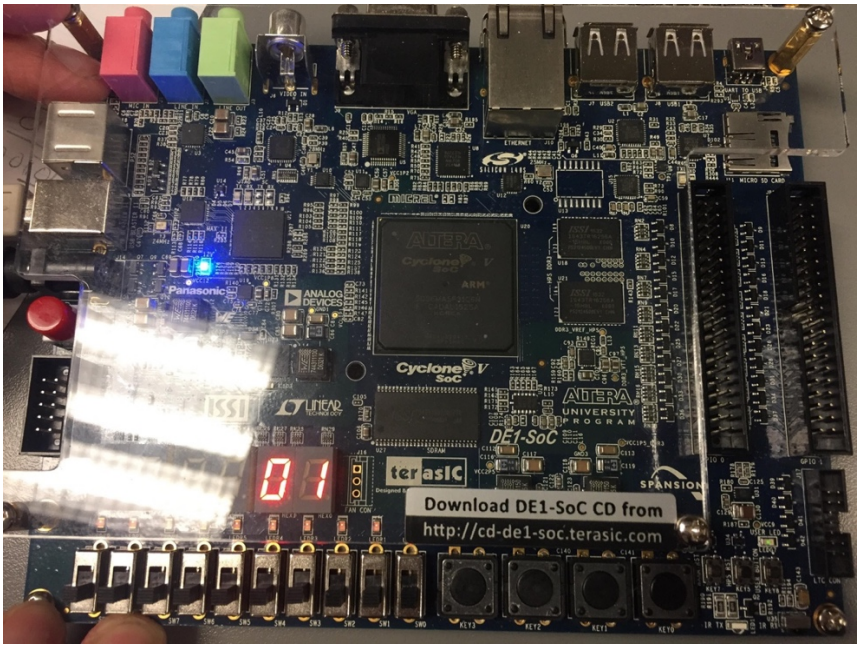


Figure 5: Starting state

To test the multi-mode counter circuit, we used the pin planner in Quantus to program the respective switches and buttons on the FPGA board to the desired start, stop, reset and direction functions. Then, using Quantus, we uploaded the circuits to the board, and tested the output, shown above in **Figure 4 & 5.** As presented on the figures, we ran through an entire sequence of the states, checking if every value was present, and then swapped the direction to test if it would

do the same. In between the state checks, we also tested the stop functions, and then conducted a final reset check to see if everything was in order.

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.

After setting the multi-mode counter as the top level entity, and compiling, the output Flow Summary is presented below in **Figure 6.**

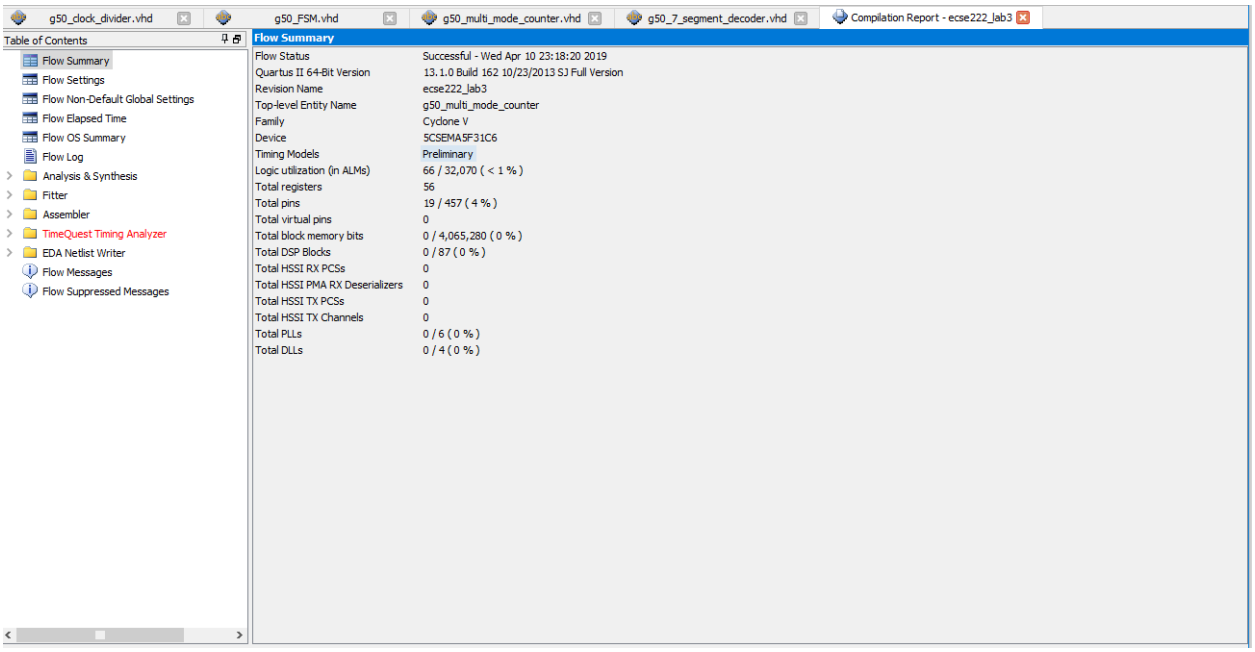


Figure 6: Flow Summary Report

The outputted RTL schematic is also presented below in **Figure 7.**

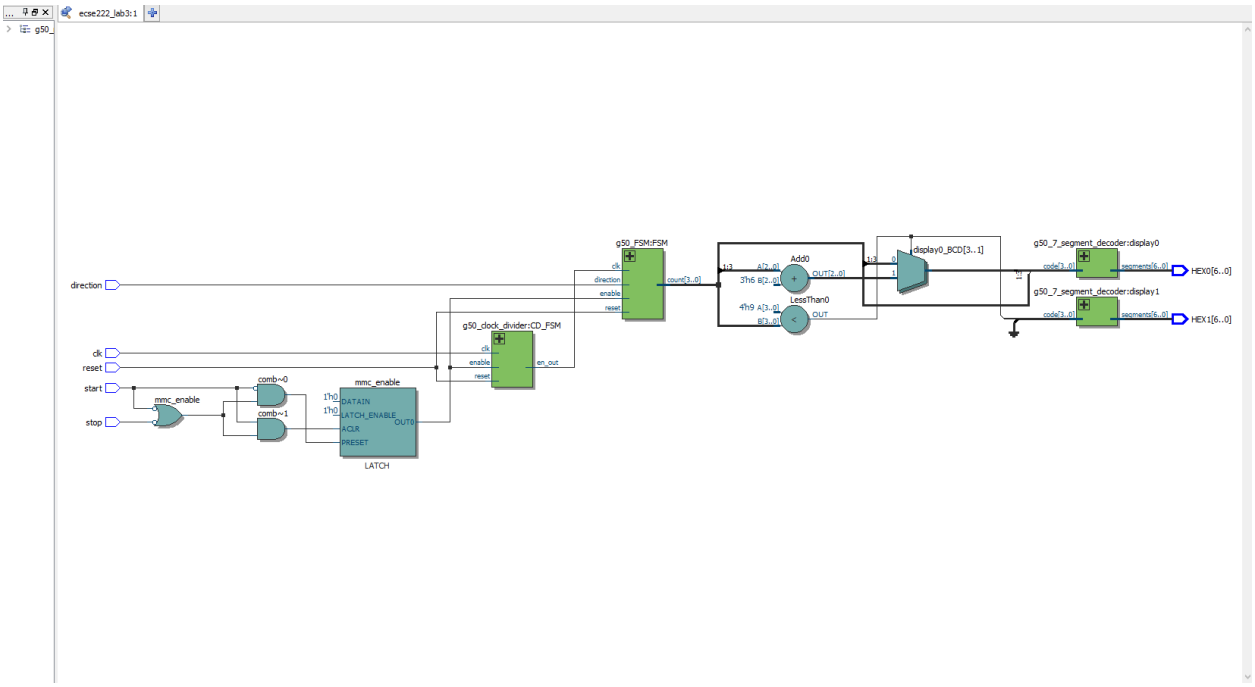


Figure 7: RTL Schematic

As the logic network shows, the start, stop and reset signals go through the clock divider and produces an output to the finite state machine circuit which then controls the ordering of the output depending on the direction signal. This output then needs to be parsed into the display, which is the responsibility of the 7-segment-decoder which performs the necessary parsing described in Lab 1, and then outputs it to the FPGA board. Important to note, because we are not using letters (hexadecimal) representation, a MUX is implemented to check if a 6 must be added or not. This is decided in the previous step with if checks.