1. A description of the counter and clock divider circuits. Explain why these two circuits are considered as sequential designs.

4-bit up-counter is used because the circuit must remember the previous count in order to be able to add to the next count. Our circuit also needs a memory storage element which stores the signal values to represent the circuits current state. This circuit is a sequential design because the counter is reliant on the clock divider outputs. Furthermore, each instances of the counter is dependant on the previous instance of the counter. For instance, Counter\_1 will be dependant on Counter\_0. The details of the circuits will be addressed later into the lab report. Another reason why this circuit is considered sequential is because it is dependant on the clock cycles and it depends on the past inputs to generate the current output. The fact that it is dependant on the previous inputs makes this circuit sequential from nature.

The clock divider generates a signal every T cycles. Like the 4-bit counter, its previous inputs are stored in the memory element in order to produce the correct output at the given time. In the case of the clock divider, a down counter is used instead of a up counter. This means that 1 is being subtracted instead of being added to the previous value. The fact that the clock divider and the counter relies on previous inputs qualifies this circuit as a sequential design.

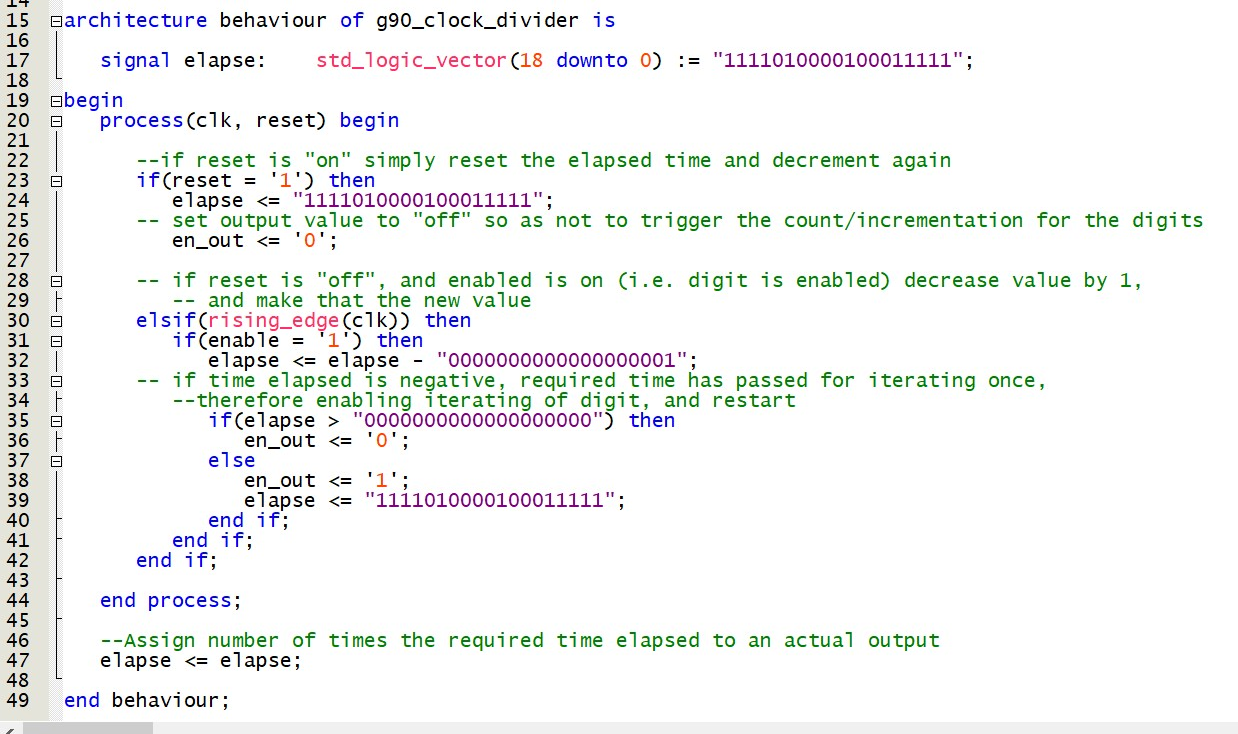
1. Explain why even though we could build a clock divider using an up-counter it is easier to build the divider using a down-counter.

Figure 1: Clock Divider Process Block

Line 17 of figure 1 shows the signal elapse transforms which is the maximum number that the on-board clock can count to (11110100001000111112 = 49999910). From that, we subtract by one at each count since it makes the circuit easier to implement. If we used an up counter, an extra variable would be needed since one would have to keep track of the current number the clock is at and another to compare which if it has exceeded the maximum. The key difference between a down-counter and an up-counter is that a down counter needs to check if the number is equal to zero. However, this is indirectly checked since we are using the unsigned logic library (use ieee.std\_logic\_unsigned.all). The up counter will need to check if the current count is smaller than the maximum allowed count. Furthermore, the down-counter is easier to integrate with the clock input (*clk*). This clock divider is based on the specific clock on the Altera board we are working on. If we used this logic on another board which the frequencies are different, it would not be possible to create an accurate counter. The entire circuitry will need to be changed in order to adapt to a new board.

1. A discussion of how the counter and clock divider circuits were tested, showing representative simulation plots. How do you know that these circuits work correctly?

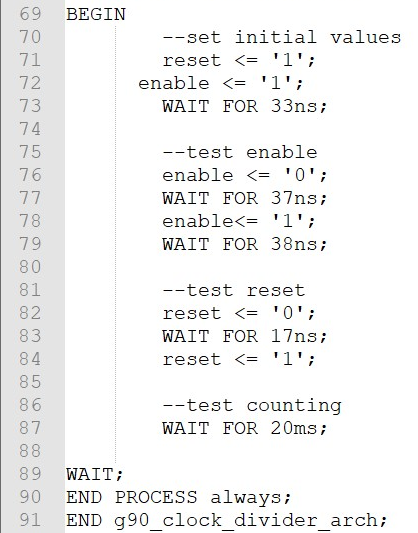


Figure 2: Clock Divider TestBench

CHANGE THE MODELSIM PICTURE

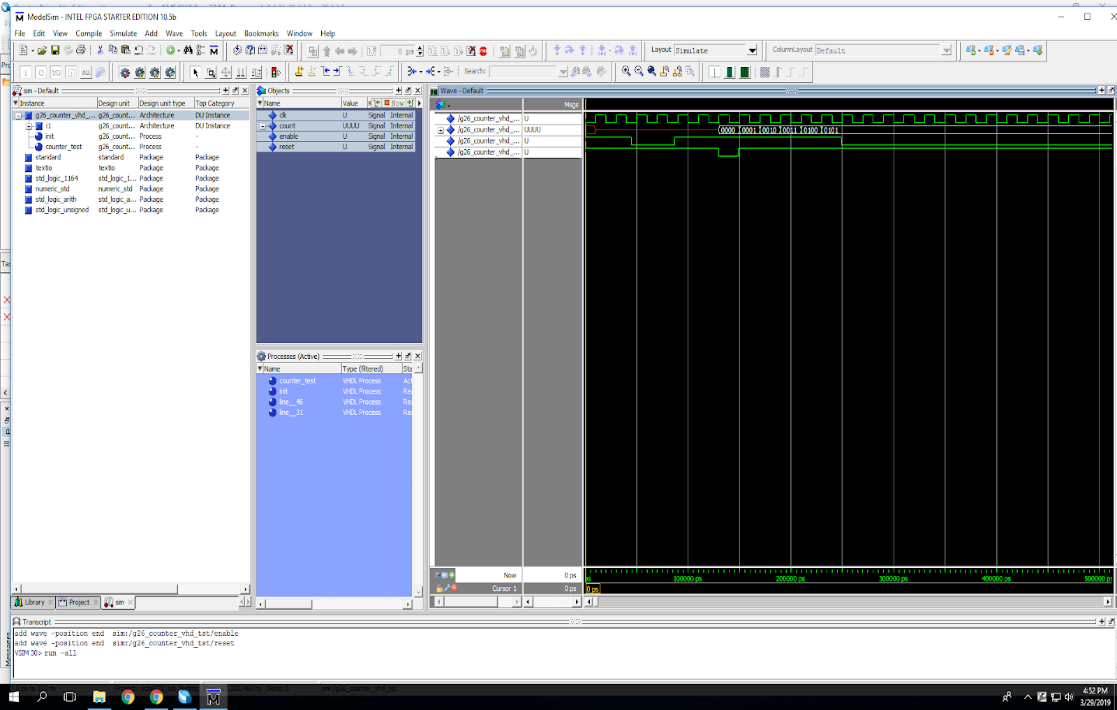


Figure 3: ModelSim Clock tests

\*\* reword this part

From the ModelSim tests on figure 3, it clearly shows that the reset is an asynchronous because the count stores no value and the counter is a red line until the reset is at a logical high, with an active-low value of 0. The clock divider works perfectly since the count value should begin at 0 when the reset is at a logical high. When the reset is at that state, the value is increased at every rising edge cycle of the clock . This also demonstrate that our counter works because the value is being incremented at every rising edge of the clock (clk). When the enable is at a logical low (active high value of 0), the clock stops counting. This demonstrates that the enable function stores the count value at a logical high. When it is a logical low, the enable function no longer stores the value since the value is always incrementing by 1 every cycle.

1. **A description of the stopwatch circuit. Explain why you created six instances of the counter circuit in your design and why?**

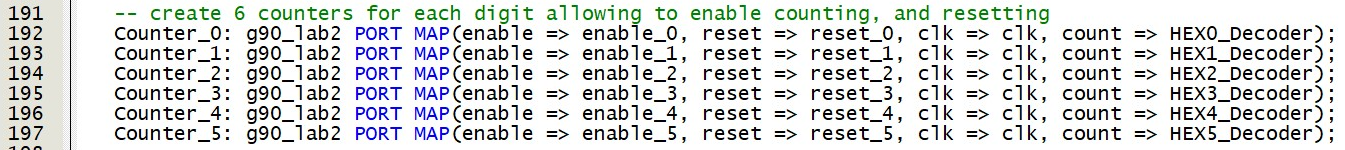


Figure 4: Instances of the counter circuit

In figure 4, we used 6 instances of the counter circuit for each display which are being PORT MAP into the stopwatch circuit. There is a master clock which has the 50 MHz internal clock, but we had to slow it down to around by having it counting every 10ms. The master clock (g90\_clock\_divider) is linked to every instance of the counter (g90\_lab2). For the first counter, it is directly associated to its enable since counter 0 is controlling hex0 which are the milliseconds. Every counter instances is linked through an equals validation gate and a MUX from the previous counter. The equal gate is to reset the previous counter back to 0 and to enable the next one. The MUX acts like an AND gate in this case since it is only a 2 to 1 MUX. Its purpose is to check if the previous counter has completed in order to allow the next counter to start counting. The OR gates are used for the reset buttons since the clock can either be reset with the push button by the user or by the previous clocks. As mentioned before, these 6 counters are in series where counter\_0 as the first one and counter\_5 as the last one. The output of counter is linked to its appropriate decoder.

Basically, each instance of the counter circuit controls a seven-segment display. Then, the counters are linked together in order to allow them to sequentially perform.

1. **A discussion of how the stopwatch circuit was tested.**

I think hta tyou haven’t finish this??

The testing process of the stopwatch was a fairly simple endeavor. Unlike the other VHDL files, a testbench was not needed to determine the viability of the stopwatch code once inputted into the FPGA board. As a result, as instructed, the stopwatch was tested directly, once uploaded onto the board. The metrics used to determine whether our program proved to be satisfactory was to observe the following conditions: Do the individual Start/Stop/Reset buttons work? Are the displays accurately displaying, and incrementing? Finally, is the board incrementing at a useful pace; that is to say, does it accurately reflect time as needed? These tests were executed fairly simply. A simple toggle of each individual button to determine their functionality proved to be successful. Additionally, visually observing the incrementation also demonstrates desirable results. Lastly, to determine whether or not the stopwatch was operating at an appropriate speed, we used an online stopwatch, although any will do, which was started simultaneously, and running concurrently with our custom stopwatch. In doing so, we are able to observe any delays, or leads, and record any deviations from the control, our online stopwatch. We have found that...

* Say that we tested against an actual stopwatch and we haven’t found **major** differences over 20 minutes. Thus we concluded that our stopwatch was accurate enough.
* From the internet: The digital stopwatch also has a **precision** of 1/10 of a second. This is a surprise! After all, the watch has a **resolution** of 1/100 second. But, because of the human reaction time, the hundredths digit is not reliable. If you measured precisely-known elapsed times with this arrangement, you would find the last digit's value to be almost random. There is a spread of about 1/10 of a second in the measured times due to the human factor. So it is *repeatable* to only 1/10 second.

Also say that the onbard clock stability: <https://forums.intel.com/s/question/0D50P00003yyGYZSA2/how-to-find-the-precisionaccuracy-of-system-clock-on-cyclone-iii-starter-board?language=en_US>

Frequency Stability = ±100ppm (parts per million) aka very stable and precise

1. **A summary of the FPGA resource utilization (from the Compilation Report’s Flow Summary) and the RTL schematic diagram for**

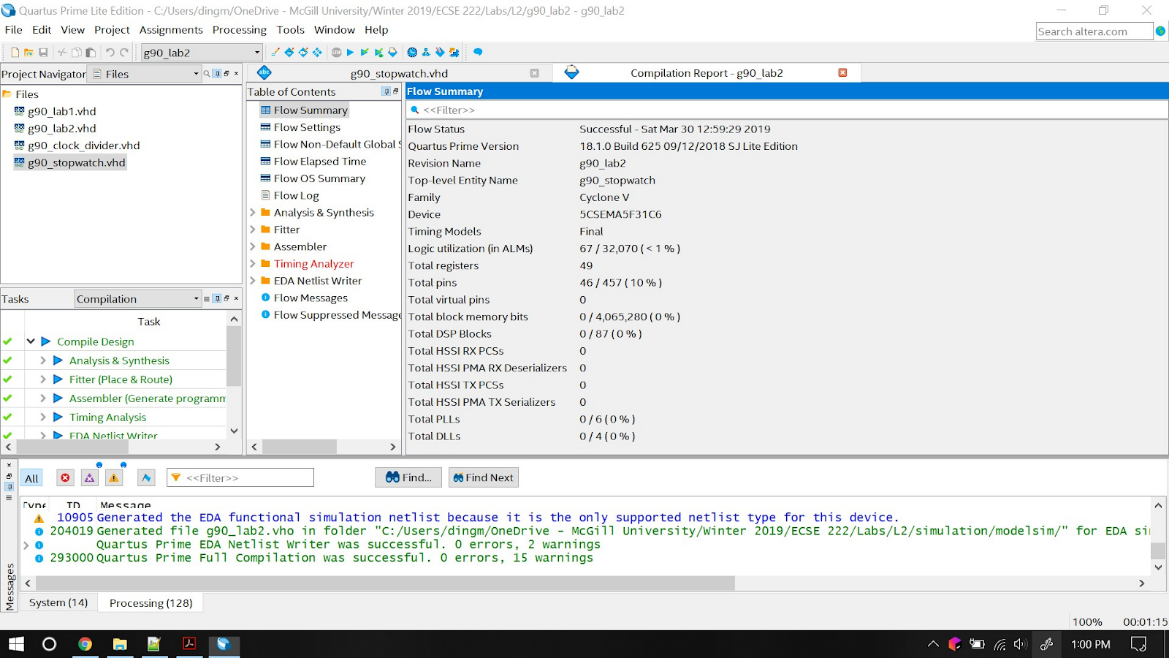


Figure 5: Compilation Report

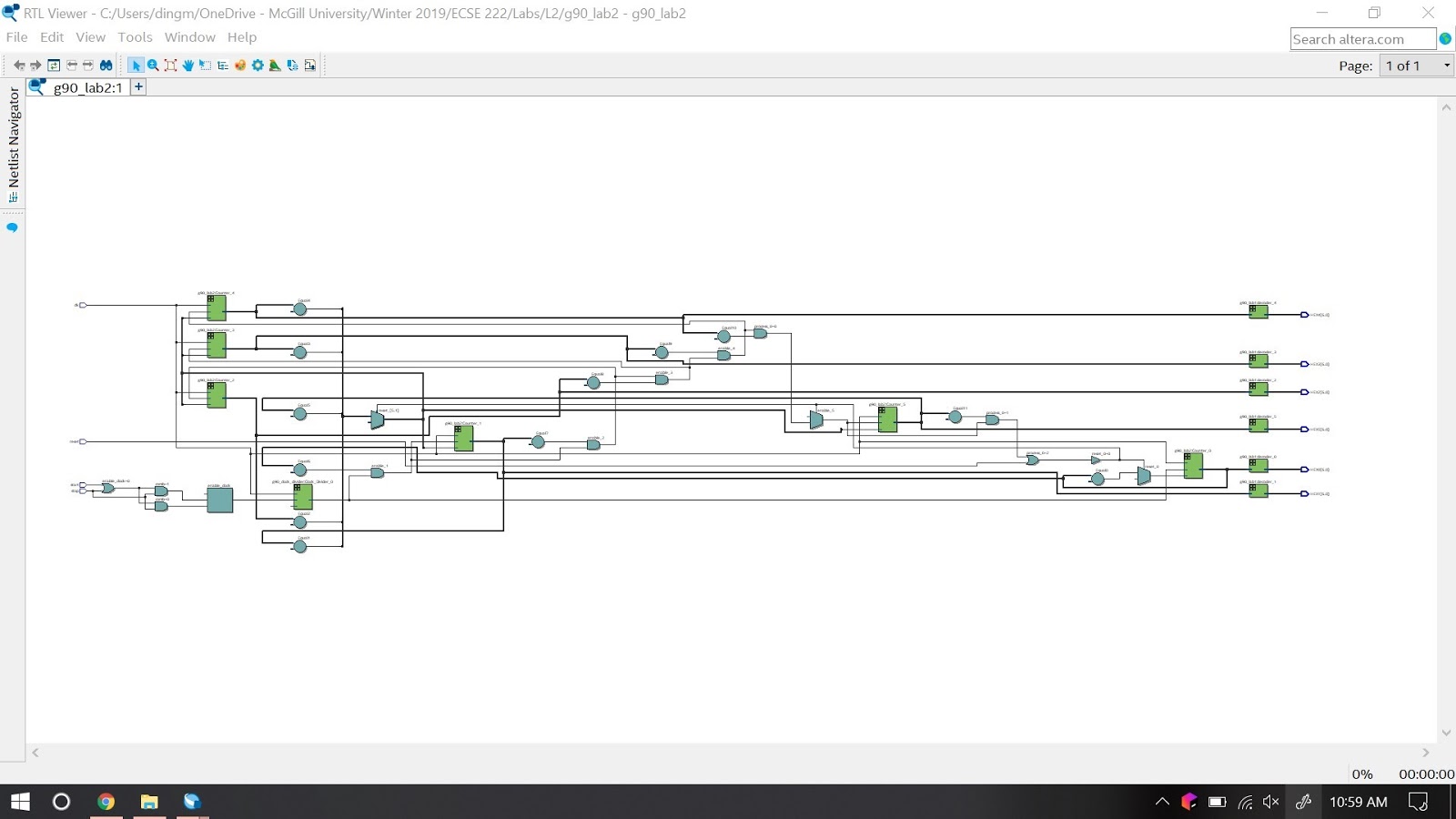
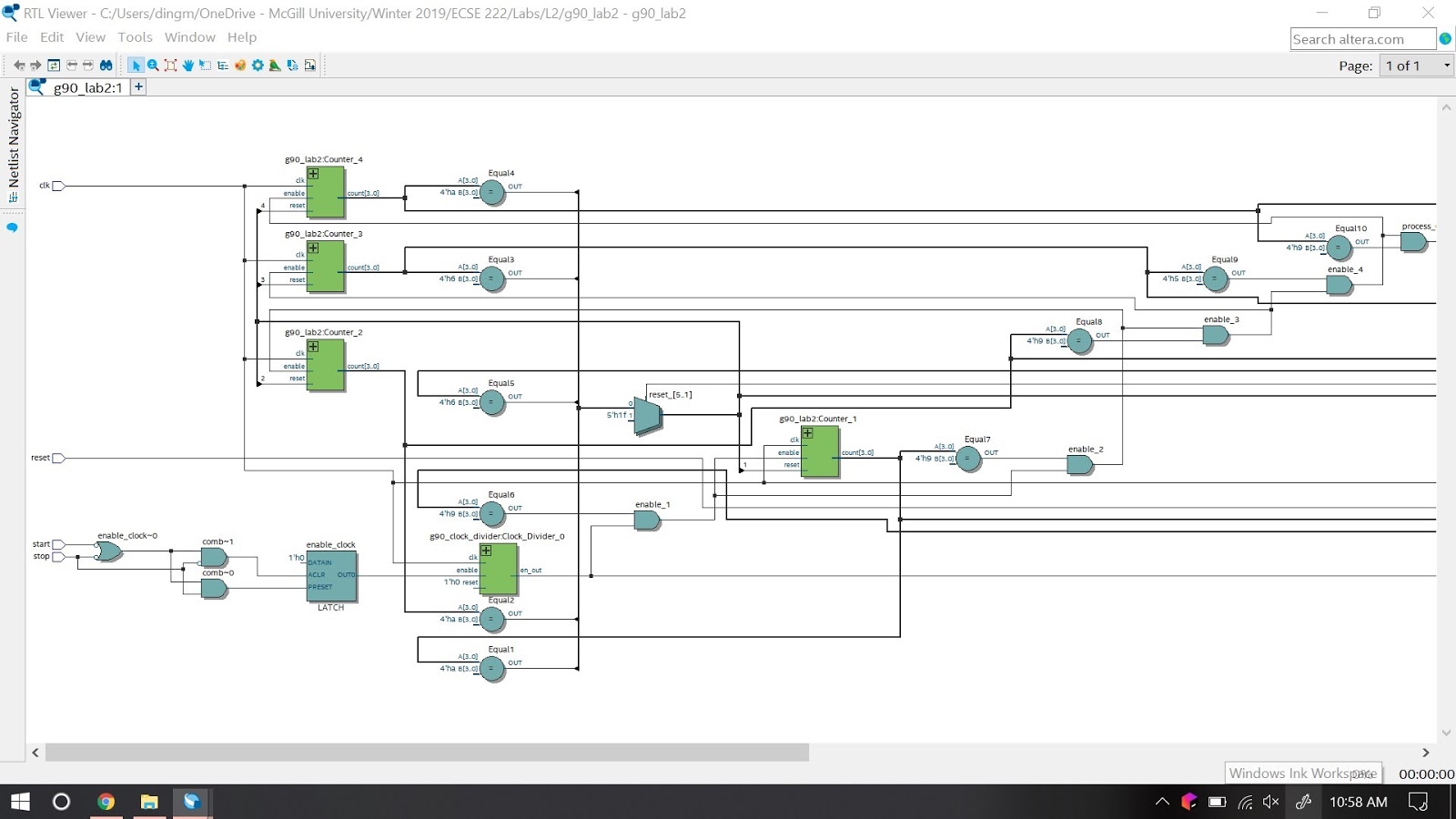


Figure 6: RLT Schematic Diagram Overview



1

2

3

4

5

6

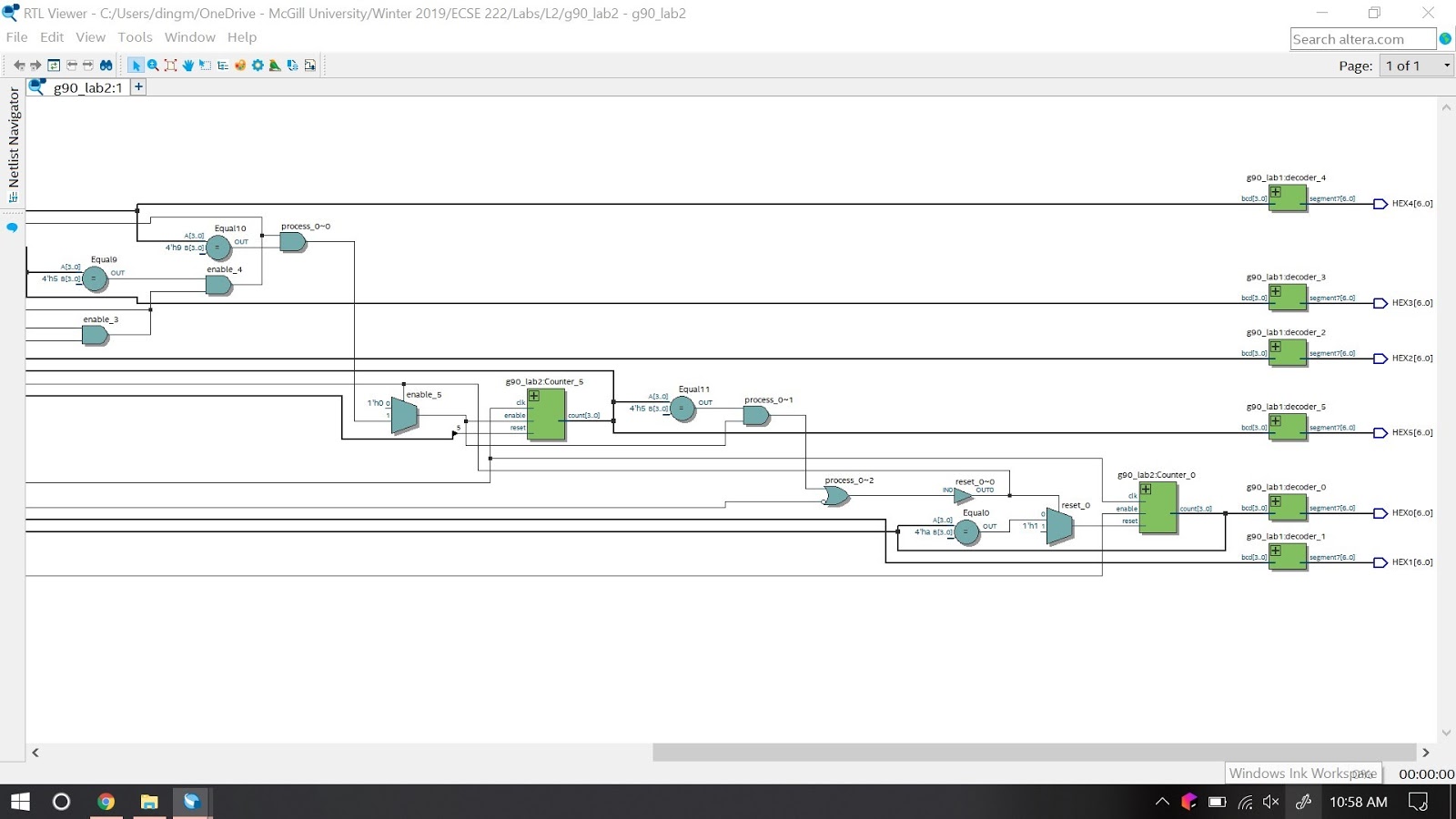
7

8

9

12

Figure 7: RLT Schematic Diagram Zoom-in pt.1



11

10

Figure 8: RLT Schematic Diagram Zoom-in pt.2

1. clk : in std\_logic;
2. reset : in std\_logic;
3. start : in std\_logic;
4. stop : in std\_logic;
5. \*if (HEX0\_DECODED = "1010") then
6. \*signal reset\_5 : std\_logic;
7. signal enable\_clock : std\_logic;
8. Clock\_Divider\_0: g90\_clock\_divider PORT MAP(enable => enable\_clock, reset => reset\_clock, clk => clk, en\_out => enable\_0);
9. \*Counter\_1: g90\_lab2 PORT MAP(enable => enable\_0, reset => reset\_0, clk => clk, count => HEX0\_Decoder);
10. \*decoder\_0: g90\_lab1 PORT MAP(bcd => HEX0\_Decoder, segment7 => HEX0);
11. \*HEX0 : out std\_logic\_vector(6 downto 0);
12. \* signal enable\_1 : std\_logic;

\* means they have more than one instances.

Number 5, 10 to 12 have multiple instances. Since every one of them has 6 instances, it will be extremely redundant to include them all in this lab report. Every counter instances comes with a equal and reset. The equal is defined by #5 allows the next counter to start when the current counter hits 10. #6 allows the reset signal to be sent once the previous counter hits 10 or if the reset button is pressed on the board. This is a 2 to 1 MUX which acts a little bit like an OR gate. Each of these two events allows the reset of the clock. #9 is the counter which start from Counter\_0 to Counter\_5. This is the piece of code that add 1 every determined time. The equals and reset handles overflow and activates the next clock when the previous one has hit 10. #10 is the same as the lab1, it decodes the BCD code into hex in order for #11 to display it on the display. #10 of them have instances from 0 to 5 in order to handle each display separately while #11 has instances from 0 to 6 in order to make use of the 7 segment display. #12 has 5 instances from enable\_0 to enable\_5, those are used to start the other display after the first one has hit 10.