McGill UNIVERSITY

# ECSE 222 Digital Logic Lab #2

Alex Gruenwald, 260783506

Kaustav Das Sharma, 260772982

Group #50

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

The counter circuit is extremely important for this lab because it is what "remembers" the current count of the circuit and then adds 1 in the next cycle. Without this component, the stopwatch would not be increasing in value.

To begin the description of how the counter works, we must first look at the implementation of behavior and process. These are implemented to define the circuit as sequential. In the implementation, two variables are placed within process (Clk and reset) which means they are executed sequentially within the circuit, which will be described in the below explanations.

Now, looking at the specific components of the circuit, there are 4 that ensure the circuit works properly. These components are the **clock** (Clk) which is what determines the period of each cycle, the **enable** (enable) which ensures that the right bit locations are incremented and will depend on the output signal of the clock_divider, the **reset** (reset) which is what resets the circuit to "0000", and finally, we look at the **count** which is a 4 bit value that is remembered by the counter and increases by 1 each positive edge of the clock if enabled (**Figure 1** shows a good representation of what the counter looks like excluding the reset which would act as the parallel load in the image).

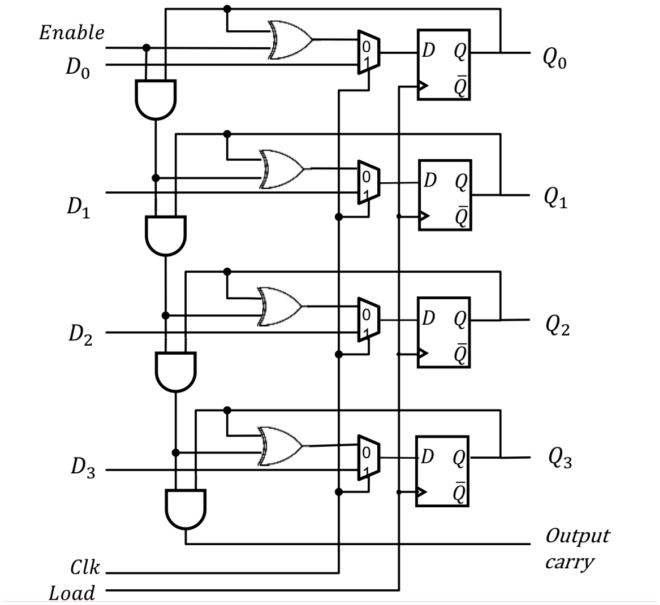


**Figure 1:** Counter representation of a 4-bit value

Looking at each component separately, we start with the clock. The clock is what determines the period of each cycle (how much time is spaced between each increment of the counter = 10ms) and will be explained more in depth in the clock_divider. The clock is very important as all synchronous variables – count – are dependent on the clock's positive edge for this circuit. When the clock is at a positive edge, the change in the circuit is performed. However, what is important to note is that despite the increment being added at the positive edge of the clock in this cycle, the change in the count is processed during the next cycle.

The second component is reset which is usually an asynchronous feature but is not implemented as synchronous and dependent on the clk. The reset component, when activated, resets the entire counter to "$0000_2$". This reset is very important when the counter reaches "$1001_2$" ($9_{10}$) because the clock cannot exceed single digit values and must restart at $0_{10}$ (discussed in the clock_divider). In our implementation, we set up the reset value to an active low and set it up first because it is asynchronous and does not depend on any other component. Being active low, when it reaches "0", the reset action takes place. For the reset, depending on the display location on the Altera board, a reset will be activated '0'. For the 6 displays, when the values are $1001_2$ ($9_{10}$) for displays #0, #1, #2, and #4 (modulo 10) the reset is activated and when the display reached $0101_2$ ($5_{10}$) for the displays #3 and #5 (modulo 6) a reset is activated as well bringing the value back to $0_{10}$ as seen in **Figure 2**. The reset is only activated for that specific display, as each display has its own 7_segment_decoder instance.

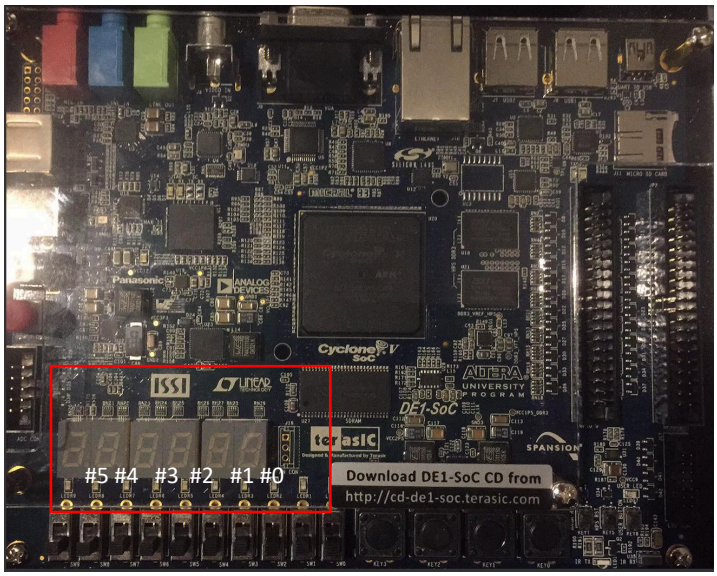


**Figure 2:** Display locations on the Altera board

Next is the enable component which is incredibly important in determining which bit positions for a 4-bit counter – $A_3A_2A_1A_0$ – are incremented at the positive edge of the clock. Because enable is an active high, when it is '1', it will increment the count by 1. If enable is off '0' then the count value is maintained and carried to the next cycle. As seen in **Figure 1**, the enable will also determine how the bits will be incremented within the count (which bit positions should be added and which shouldn't).

Finally, we consider the count vector, the variable that stores the 4-bit value each cycle. In the implementation, we set up 2 separate count values, one temporary count which is what is actually incremented each cycle, and one counter variable which is given the value at the end and will be the output signal to the stopwatch.

To sum up the entire process, we start with a reset '0' to ensure the circuit count starts at zero. If the enable is activated '1' on each positive edge of the clock, then the count will be incremented by 1 otherwise, if the enable is '0', it will not increment on the clock cycle and instead remember the count value for the next cycle. If reset is activated at any moment '0', the entire count will restart at "$0000_2$" but this will be looked at more in depth in the clock_divider.

**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.**

The clock_divider in this project has the sole purpose of determining the number of cycles in a period of the clock for the stopwatch. When the period is complete, the output signal of the clock_divider is an enable '1' which communicates to the stopwatch to increment a value by 1 (which occurs every 10ms). All other cycles result in an output enable signal of '0' which, because the enable is an active high, means the stop watch must remember the previous state instead of incrementing by 1 (as discussed in the counter). Process is used for clock_divider for the same reasons as described above, to ensure certain variables maintain sequential properties in the circuit.

The first question asked in the report was asking how many cycles *T* must be implemented to ensure the clock sends an output signal of '1' every 10ms. Considering the Altera Board clock operates at a frequency of 50MHz, converting this into a period, we get 0.00002ms by the equation seen in **Figure 3**.

$$\boldsymbol{Wave\ Period = \frac{1}{Frequency}}$$

**Figure 3:** Equation to find the period given a frequency

As we are attempting to achieve a period of 10ms, we simply divide 10ms / 0.00002ms to get the number of cycles *T* to be 500 000. With the number of cycles considered, we are now able to begin the explanation of the clock_divider in depth.

Just as explained before in the counter, the clock, enable, and reset are required to ensure the cycles are accurate. To begin, we first declare the number of cycles we must decrement by in order to get approximately 10ms periods from the clock. We set the generic constant value T = 500 000 as it will not be changing throughout the entire application.

In the process of the clock_divider, we first start by setting reset = 0. Since this is an active low, we reset all the variables to their original state. In this case, we reset the temporary variable to 0 and will set the current variable back to T – 1 (representing the number of cycles / period). The current variable is what keeps track of the number of cycles that have been decremented, from the 499 999$^{th}$ cycle to the 0$^{th}$ cycle, where the reset will be activated again. The temporary variable is a temporary enable variable that keeps track of when the en_out should be equal to 0 (hasn't reached 10ms) and when it should be equal to 1 (10ms has passed) where the en_out (the output signal to the stopwatch) will communicate to the stopwatch 10ms has passed.

For the clock_divider, the reset variable is activated when the current reaches 0 meaning that the clock has had 500 000 positive edges (as it is a down counter). When this is reached, the current counter is set to T – 1 again and the temporary enable variable = 1. From here, the en_out = temporary and is sent to the stopwatch as explained above.

Every cycle, the en_out variable sends a signal to the stopwatch about the status of the time that has passed. Whenever a 0 is sent, the stopwatch receives enable value of 0 meaning none of the its variables can increment by 1, changing nothing on the Altera board display or in the counter of the stopwatch. If the cycle hasn't reached 500 000 positive edges (temporary = 0), the temporary value will decrease by 1 and the next cycle will be activated only if the enable for the clock_divider is actually 1. This internal clock_divider enable will be able to be

manipulated by the user directly from the Altera board when they press start and stop, ‘1’ and ‘0’ respectively.

Both the counter and clock_divider were built as sequential circuits which is obvious as process is defined in both. First, to define what a sequential circuit is: a sequential circuit is a circuit whose output depends on the timing and order on the input as well as the history of inputs.

When we look at the counter and clock_divider, both designs are related to having sequential designs because of the implementation of the clock (clk) a timing trigger. This clock determines when a change is allowed to take place (ex. an incrementation up 1 or down 1) on it’s positive edge. In both the counter and the clock_divider, the input and output values are time dependent, an important property of sequential circuits.

The second important observation which is another property of sequential circuits, is the addition of storing and “remembering” previous states and inputs. In both the counter and clock_divider, flip-flops are implemented (as seen by the RTL’s generated) to remember past inputs and outputs that affects the next state of the cycle. When enable = 1, with the up-count and down-count processes, in order to determine what the next value must be, the counters need to be aware of what the temporary variable was on the last cycle to increment properly for the next. If enable = 0, the memory process is used to maintain the same state (value) for the next cycle.

For the clock_divider, a down-counter is used to determine how many clock cycles have passed in the circuit. We use a down-counter in this situation for the purpose of simplicity. The implementation of a down-counter is almost identical to an up-counter except for how we determine if a given number of cycles has passed. With the up-counter, we signify that T – 1 cycles have passed when the current = T – 1 while for a down-counter, when T – 1 = 0, we know T – 1 cycles have passed. For this implementation, considering T – 1 = 499 999, an extremely large value, it is very difficult and costly to check if the current binary bit value is equal to 499 999 to figure out if 500 000 cycles have passed.

$499\ 999_{10} = 1111010000100011111_2$ (19 bits minimum).

With a down-counter, we only have to check if the bit value of current is equal to 0 which can be implemented with a lot fewer bits (check if all bits are '0') which is a lot more cost efficient and faster.

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

To test the counter circuit, we started off by creating a template from Quantus Prime, by setting the counter circuit as the top-level entity. The testbench was then written such that it tests the output for enable, reset and counting. The simulation was then ran on ModelSim Altera, and produces the output wave displayed in **Figure 4.** The values listed for the counter were also converted to hexadecimal, displaying the correct, desired output (ranging from 1 - 16) for this circuit.


Figure 4: ModelSim simulation of counter circuit

A similar process was used for the simulation for the clock divider, except the top level entity was changed to the clock_divider in order to create the initial testbench, and it was written to accommodate for the testing of enable, reset and counting, producing the simulated wave shown in **Figure 5.** The simulation should create a signal that is asserted every period of

the clock cycle, as explained previously, and since the output assertion matches the counter's it is proved to correct.

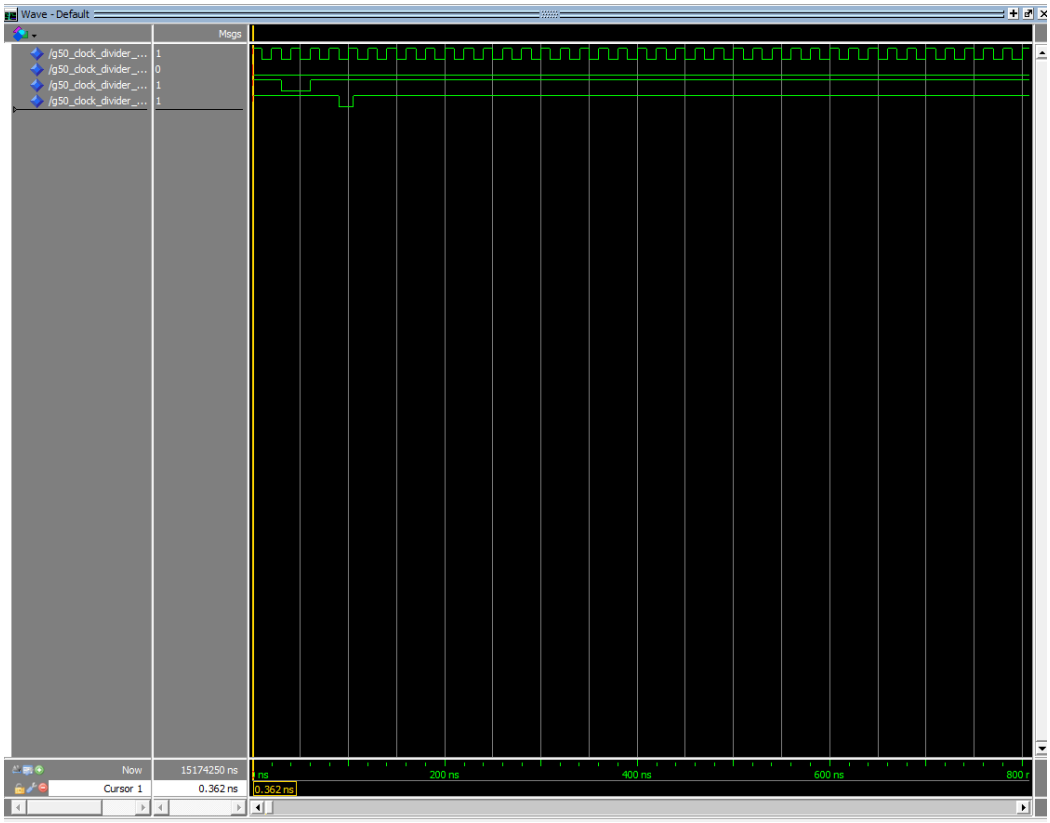


Figure 5: ModelSim simulation of clock_divider circuit

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

The stopwatch is the main circuit that imports other circuits – counter, clock_divider, and 7_segment_decoder. To begin the explanation of the stopwatch, we first must import the three other circuits as we will use them throughout the circuit of the stopwatch. The output of the counter instance will be used to increase the count for each instance of the 7_segment_decoder, that is displayed on the LED display on the Altera board. The output of the clock_divider will determine when the count increment must be activated.

For the stopwatch, the code pertaining to the entity of the circuit was given and is defined as having a start (active low), a stop (active low) and a reset (active low) that will be triggered upon activation from the user on the Altera board. Additionally, because the stopwatch circuit is sequential and depends on timed increments, a clock will also be added. Finally, 6 instances of the display will be created for later use for the 7_segment_decoder – HEX0, HEX1, etc.

After importing the three circuits as described above, we begin with describing the circuit of the stopwatch. In order to store the count for each display on the Altera board, we must create 6 instances - digit_0, digit_1, etc. – where each instance will be assigned to an instance of the counter. These instances all start at "0000" and will increment upon each clock positive edge, which is determined by the clock_divider enable output. We must also create a signal variable for the enable of the stopwatch and a signal variable that takes the value of the output of the clock_divider (starts at '0' and becomes '1' every cycle) which signifies when the enable for the stopwatch should be '1'. For each display variable – digit_0, etc. – there must be a reset variable and a clock variable assigned to it, as the instances have different max values (some being mod 6 while others are mod 10 as described earlier). What is important to note is that we don't have to create a new clock variable for the first display location (#0 as seen in Figure 2) as this clock is the en_cd variable assigned earlier (enabled every 10ms).

Next, we must create 6 instances of the counter and assign all the variables, created before, to the specified counter as well as enable the stopwatch for each instance.
Ex. counter1 has a clk => clk1, a reset => reset1, and the storage space for a 4-bit count => digit_1.

With all the architecture set, we begin the sequential circuit with defining the process to have the start, stop, and reset variables – variables that are asynchronous. In a convenient if, elsif statement, each cycle we check if 'start' is activated ('0' starts the stopwatch), if stop is activated ('0' stops the stopwatch) or if reset is activated ('0' resets the stopwatch). These three variables are all activated upon user request.

The final check in the if elsif statement is the fourth process defined – the en_od– the clock that is enabled every 10ms defining the speed of the stopwatch. On every positive edge of this clk, 10ms has passed and the stopwatch displays are updated. In order to update the correct display, we start at display #0. This display changes every positive edge of the clk and counts up to $1001_2$ ($9_{10}$) before resetting the count to $0000_2$, starting the count again. What is important to note is that not all the counts increment to 9 and instead, some have to reset after 5. This is due to the property of a real clock, where displays #3 and #5 have a mod 6 display. To consider this, we execute a reset after the count = '0101' ($5_{10}$).

When we look at the displays from right to left, fastest to slowest, if a display on the right is reset, it signifies that the next display in the list is incremented by 1, otherwise the first

display is simply incremented and the reset stays '1'. This is captured by the 'end if' statement executed only if a display instance is incremented without a reset (stays '1': off). This ensures the stopwatch is in sync where the displays to the left are only incremented when the display to the right reaches its max value, just like a regular watch.

Finally, when the process ends, all clks are reset and the individual display instances are sent as inputs to the 7_segment_decoder to be converted into a hexadecimal display representation for the Altera board.

The reason why there must be 6 instances of the counter circuit is because each instance of the display must have a different counter instance. This is for two main reasons.

First, not every display has a counter incrementing to $1001_2$ ($9_{10}$) as some displays can only increment to $0101_2$ ($5_{10}$) simply because of the way a stopwatch works. For this reason, the displays need to have different instances and simply cannot use the same counter for two displays.

The second reason is that each counter increments at different speeds. It is true that they are all synchronous and are based off of the 10ms enable variable supplied by the clock_divider, but despite sharing a similar positive edge location, they don't all increment at every edge. Instead, each display increments at the positive edge only if the previous display had a reset.

## A discussion of how the stopwatch circuit was tested.

To test the stopwatch circuit, we used the pin planner in Quantus to program the respective switches and buttons on the FPGA board to the desired start, stop and reset functions. Then, using Quantus, we uploaded the circuits to the board, and tested the output, shown below in Figure 6 & 7. As shown on the figures, we conducted a basic test, stopping the stopwatch at 3 seconds, and then reset, and then conducted a more extensive test, letting the stopwatch run until 66 seconds and then stopping it.

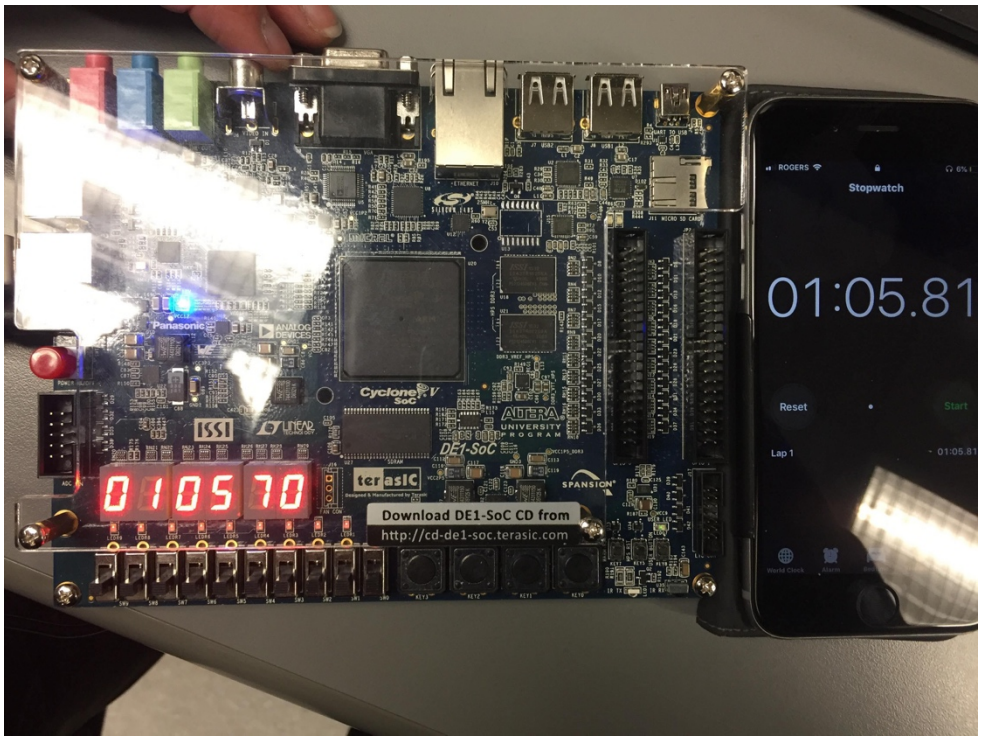


**Figure 6:** Maximum test

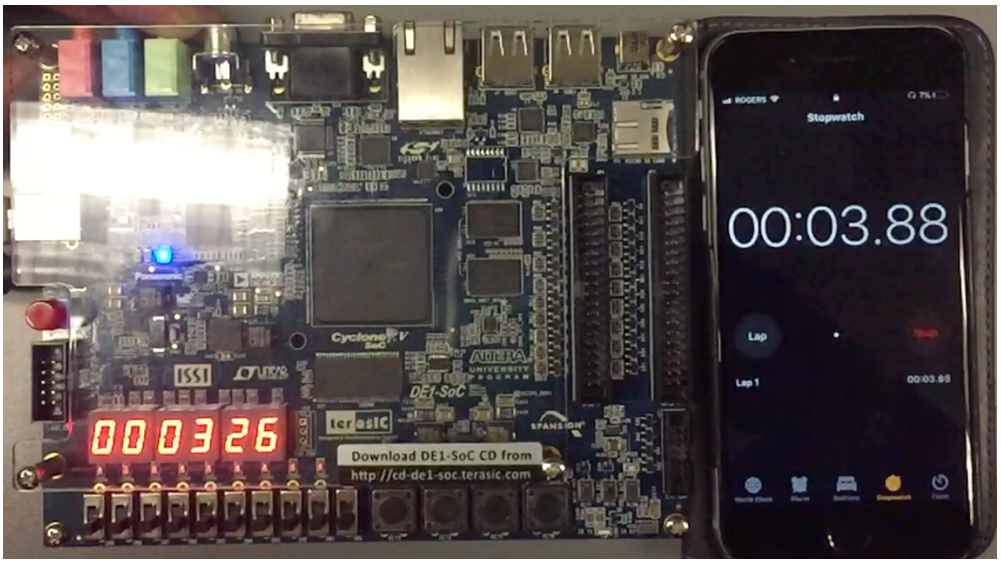


**Figure 7:** Basic test

A summary of the FPGA resource utilization (from the Compilation Report's Flow Summary) and the RTL schematic diagram for the stopwatch circuit. Clearly specify which part of your code maps to which part of the schematic diagram.

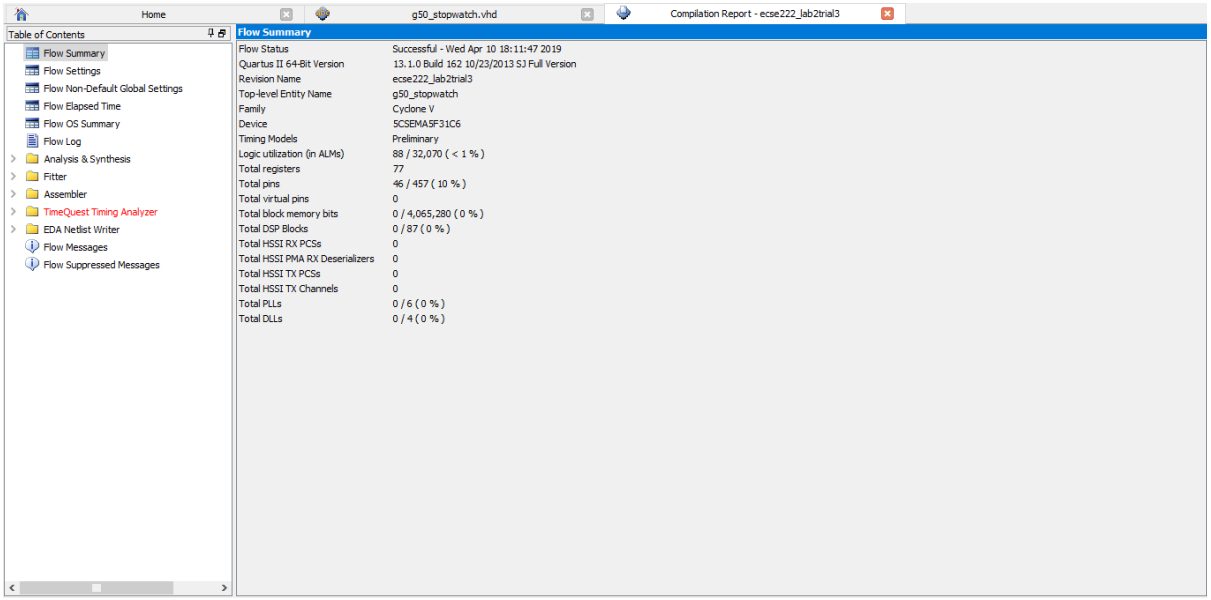


**Figure 8:** Flow Summary Report

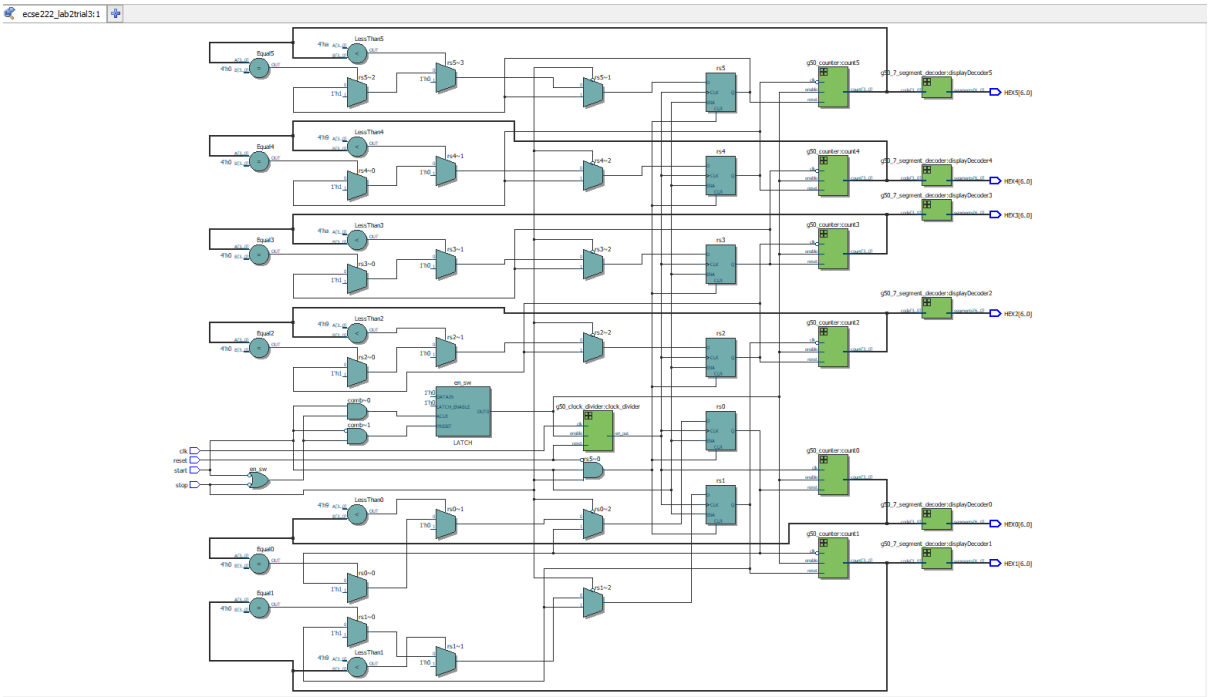


**Figure 9:** RTL schematic diagram

The latch is the stopwatch's enable function. It is connected to every display as it determines if the display should change or not, and thus the latch is dependent on the start and stop button where it changes when something is pressed. The clock defined in the stopwatch circuit is linked directly to the clock_divider circuit where the output (the 10ms period clock) is used as the output and is connected to the 6 flip-flops that store all the

information about the next cycle. The stopwatch input is a mix of both the clock - from the clock divider - and the bit value that it is to be displayed. In order to remember what the previous state was on, the flip-flops storing the present state is connected to the beginning (with all the defined variables) where it then checks with a modulo whether the designated number has reached. Now considering reset, each flip flop (rst) is able to be reset to 0 if the reset button is pressed. Finally, in order to determine whether the next display in the line should change (as it needs to be aware of the previous state) the output of the counter is connected to the next addition to check if it has reached its max value (mod 6 or 10) in which case the next display is able to increment up one.

*****Since there are 6 instances of the display, there has to be 6 resets, 6 controls and 6 temporary storage variables in order to account for the displays.