# Digital Stopwatch

Designed in Vivado – Using VHDL – Implemented with Nexys DDR

Jacob Konja, Dawsun Schrum, Zachary Page, Antonio Montagna
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
jacobkonja@oakland.edu, dtschrum@oakland.edu, zpage@oakland.edu, amontagna@oakland.edu

***Abstract*— Using software and hardware, students have worked together to design a functional digital system. The reason for this project is for students to get a better understanding of digital systems as well as prove the knowledge gained throughout this course. Developed in Vivado using VHDL, a successful digital stopwatch has been implemented onto the Nexys DDR. Like any general digital stopwatch, it will feature a reset function, start/stop function (pause), and lap function (read_write).**

## I. Introduction

No matter how advanced the world is or may become, stopwatches will always play a role in society. A digital stopwatch proves extremely helpful in certain situations. For example, running a mile around a track and timing each lap around the track utilizes a stopwatch perfectly and proves its usefulness.

The overall goal is to project a count on a total of 8 seven-segment displays. Now including the expected capabilities of a basic digital stopwatch, there are some extra has some extra features that will be implemented within this project. First is a "reset" button, also known as a "cpu reset" on the Nexys board, its functionality is quite simple and is defined within the name. When pressed, it will restart the count of the stopwatch. Next is the "pause" or "start_stop" function of the project. This is also a simple goal as when pause is enabled our desired outcome is that the count stops and vice versa. When the pause is low, the count continues. Lastly and the most complex is the "lap" function or "read_write" function. At any moment in time we are able to store (read) a desired count and show (write) it on the seven-segment display.

These functions prove the effectiveness of this project and how it may be used in everyday life. Why use your phone as a timer when you can carry your laptop and Nexys board with you?

## II. Methodology

As a group we quickly identified what would be required of a digital stopwatch, and that would be a digital system of course. A digital system will comprise of a Finite State Machine (FSM) as well as a data-path circuit. To put together a circuit such as this one, it requires a great deal of critical thinking. It is very important to break apart the data-path circuit into smaller parts to help explain and achieve a better grasp of each elements functionality and the reason behind it. These three smaller elements of the data-path circuit as well as the FSM(s) will be explained in further detail component by component.

As mentioned previously, we will be using all 8 seven-segment displays of the Nexys DDR. The right most display will show hundredths of seconds while the left most display will show tens of hours. The max count for this digital stopwatch is projected to be 99 hours 59 minutes 59.99 seconds. The count after this will revert back to 0 and recount. In regards to the lap function, it will be controlled using switches. 4 total laps will be able to be saved thus requiring to assign a total of 5 switches. One switch for start/stop and 4 for the lap functionality. Lastly, the cpu_reset button will be enabled to of course reset the timer when desired. Registers, counters, logic gates, and many other important components play important roles and will be utilized within this project to achieve the ultimate end goal. When the following elements are put together, it creates a fully functional and efficient digital stopwatch.

### *A. Counting*

There is a total of 11 counters within the scope of this project. 9 of them will be explained in further in this section. To begin, as stated previously the max count possible is 99 hours 59 minutes 59.99 seconds. Therefore, these counters require a limit. 6 of these counters have a max count of 9 or "1001". These are known as a BCD counter or Decade counter; they have the capability to count 10 total unique digits. 2 of these counters have a max count of 5, a max count of 5 results in a modulo-6 counter.

Notice that once a counter reaches a max count, it effects the other. For all counters except the 10 millisecond (hundredth of a second) counter, the preceding counter must reach a max out; almost like a domino effect. Therefore, each counter must have 3 inputs and 2 outputs. The first input is resetn, an active low input that as stated before will reset the count to 0 when pressed. Second is the clock signal, enforcing that the counter functions on the rising edge of the clock. However, the counter may function on the rising edge of the clock and when the third input is active; enable. When will enable be '1'? Enable will be high when pause is low AND the previous counter output has reached a max count (pulse Z). The final out, Q, is simply the current count of the counter. To summarize: when the pause is low AND the pulse Z is high for the previous counter, that counter is able to count (during transition phase from max to first count) and increase Q.

Now, this element is complete with 1 more addition. The Nexys DDR has a native clock speed of frequency 100MHz. This translates to a clock cycle every 10ns. This is not ideal for a stopwatch that requires a count every 10ms. Therefore, another counter is required to regulate the overall count. count. It is connected to all enables in the counters. It has a resetn, enable (pause), and an output Z that sends a pulse every 10ms (.01s).

The 4-bit Q's for each counter are set into a 32-bit bus and are lead into the next important element of this stopwatch. See *Figure 1* below for the drawn schematic of the counting element.

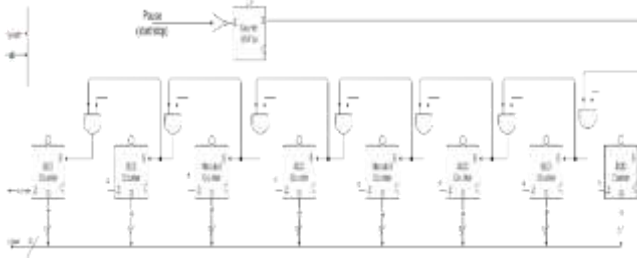

*Figure 1- Counting Element*

*B. Lapping*

This element utilizes registers and tri-state buffers. Effectively, the register will store data when enabled, otherwise it will continue its previous value. The tri-state buffer for that register determines whether the data may be passed and eventually displayed onto the seven-segment display. These 4 registers (for the 4 lap switches) include resetn, a clock signal, lapE (lap enable), a data in (count) and a data out (count). Essentially, when we want the lap function to hold the value that is desired when the switch is high, enforcing the register to hold the data of that desired time. Thus, when a switch is high (to acquire a desired lap time), we want the enable of the register (regE) to be low. Therefore, regE = not(lapE) (when the counter is active). LapE is an active-low enable.

Now, it is impossible to display both the count as well as the elapsed time. Only one path of data is allowed to pass to be displayed. This is where the tri-state buffers come into play. There will be a total of 5 tri-state buffers. One for the original count to pass, and one for each register (4). BuffE (buffer enable) will be 5 bits. The buffer that holds the count (BuffE(4)) will be the default buffer that is enabled. While the stopwatch is counting, BuffE(4) is enabled. The buffer enables are tied with the lapE. When a LapE (switch) is flipped high and the count is paused, the buffer assigned to that switch is prioritized and the desired time captured while the count was running is displayed on the seven-segment display. BuffE, regE and LapE will be controlled via FSM.

All 5 of these 32-bit data paths then feed into an OR gate. This OR gates works well for this situation because, as stated previously, only one tri-state buffer is enabled at a time. Thus the OR gate will always have 1 path of data that contain 1's and all the other paths of data will be '0's. See *Figure 2* below for the full lapping schematic.

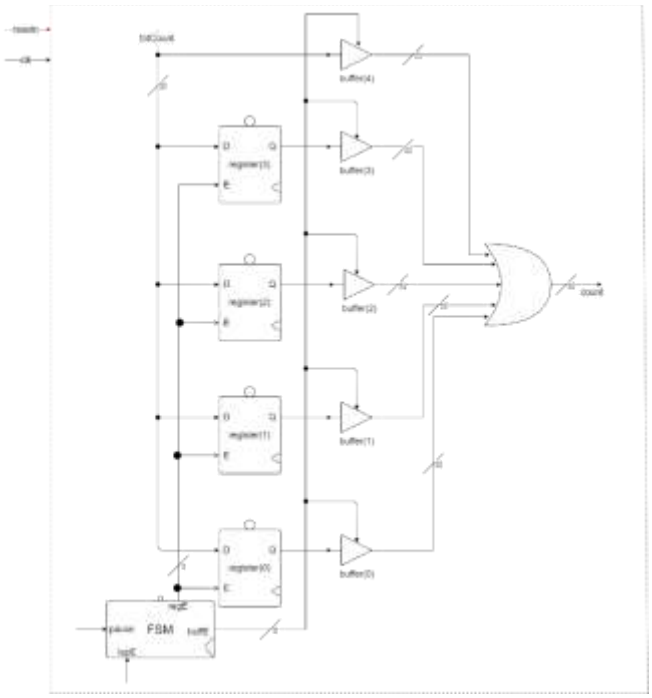


*Figure 2 – Lapping Element*

*C. Finite State Machine for Lap Element*

Now, as previously mentioned, the lapping element includes a FSM. This FSM is intended to control the register enable (regE) and the buffer enable (buffE). It is possible to control these elements by the pause function as well as the lap enable (lapE) functions. There are 3 possible states for this FSM.

The first state is the counting state. It is business per usual, the counter is doing its job and it is being displayed on the FPGA. In this state, buffE(4) or the counting buffer, is the buffer that is enabled; no other buffers pass data. In this state we make the regE = not(lapE), this is allowing us to store a desired lap time. When we flip a lap switch, the regE set to that switch is now low. This effectively stores the desired time we would like to later display.

Now to achieve the second state, it depends on the pause function. While un-paused, we stay in state one. However, if the user decides to pause we now enter state two. This state has all the register enables set to low. This is to allow us to keep the stored data from state one. Now while staying in this paused state, we can display the elapsed times chosen during state one. If any of the switches are now flipped high, it also enables the buffer for that register to eventually be shown on the display. If the user attempts to show two elapsed times, it will display the most significant bit (MSB) of the switches that are enabled. For example, if the user has lapE(3) and LapE(1) flipped high. LapE(3) will be displayed over LapE(1).

If we are in state two and no elapsed times are chosen to be shown, it defaults to the original count being

shown on the seven segment display. To go back to state 1, it is determined by the pause function. See *figure 3* below for the Algorithmic State Machine (ASM) of the Lapping FSM.

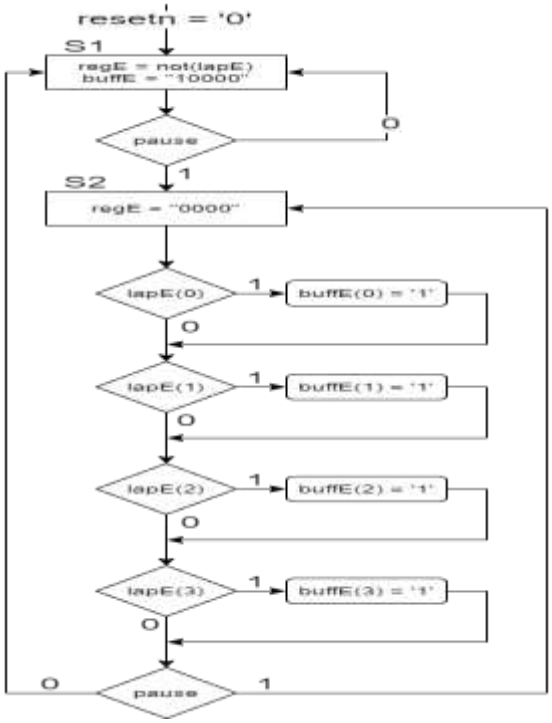


*Figure 3 – ASM of FSM for Lap Element*

*D. Display Element*

Now, the goal is to display the count or elapsed time onto a total of 8 seven-segment displays. However, only one seven-segment display may be used. This is where a seven-segment serializer is utilized and implemented. The serializer illuminates each digit for 1 ms every 8 ms. This action is too quick for the human eye to see and thus works perfectly for the digital stopwatch. This requires a 32-to-4 multiplexer, seven-segment decoder, 3-to-8 decoder, and a FSM to control the selector of the 32-to-4 multiplexer.

The inputs of this multiplexer contain eight 4-bit inputs (totals to 32-bit bus). The FSM utilized within this element is the selector for this multiplexer. Every ms it selects a new 4-bit input. Therefore, all eight 4-bits are illuminated in their respective display every 8 ms. The selector of the FSM is also fed into a 3-to-8 decoder where each display is illuminated every ms as well. Note that each enable input (EN) of the decoder is fed into a NOT gate before entering the display. This is because the seven-segment displays are active low inputs.

Finally, the outputs of the multiplexer are fed into a seven-segment decoder, which is ultimately feeds into each seven-segment display. See the *Figure 4* below of the full display schematic.

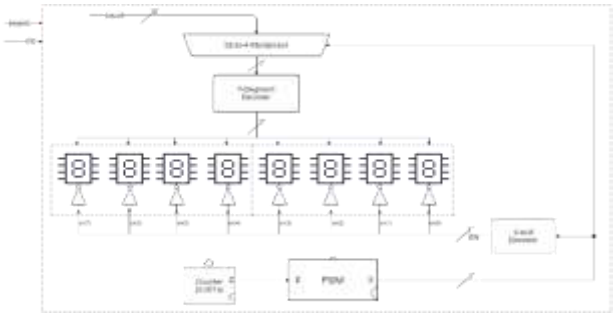

*Figure 4 – Display Element*

*E. Finite State Machine for Display Element*

As previously stated, the FSM acts as the selector of the multiplexer and selects a new input every ms. This making all eight 4-bit inputs being displayed into their respective display. How does the FSM select a new input every ms? The FSM contains an enable; this enable is controlled by a .001 second (1 ms) counter. This counter has an output pulse Z that is generated every ms. This pulse then feeds the into the enable of the FSM. Thus, the FSM is active every ms and is able to select a new input to the multiplexer every ms. See *figure 5* of the ASM for the FSM.

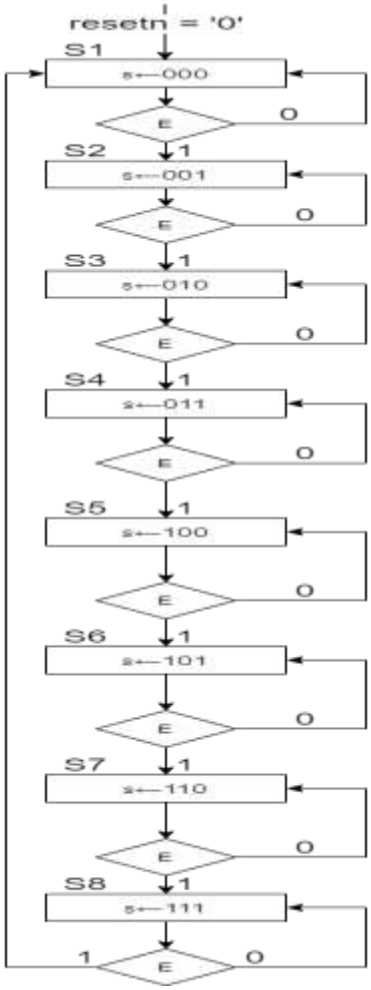


*Figure 5 – ASM of FSM for Display Element*

## III. Experimental Setup

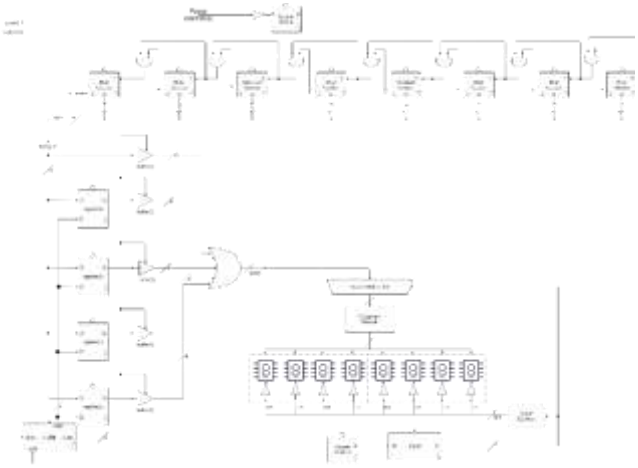

*Figure 6 – Final Schematic*

The figure directly above shows the final schematic that was ultimately designed into Vivado. The three main elements (counting, lapping, and display) were first designed, each having their own top file. Each element was treated as its own mini-project. Once all three main elements were complete, a master top file connected each element together.

In regards to the I/Os for the Nexys board, there were a small amount of constraints used. The cpu_reset was linked to resetn, clock (clk) was added, and the 7-segement display was also added. Switch 15 (SW [15]) was used as the pause function. When the switch was high, the board is in a paused state. When low, the board is in the counting state. Lastly, SW [3] – SW [0], were used as the lap enables.

## IV. Results

There were a multitude of different results encountered when working toward the ultimate end goal. There were complications and different problems that stood out when developing this project.

Some problems shined more than others. One was the test bench. While having a successful project, it was a struggle to get a desired timing simulation to show the functionality of the project. Below is the timing simulation of the project, the counting portion is shown correctly and shows the first 4 bits of the count (count[0] through count [3]). However, it was a struggle to show the lap function in the timing simulation. See figure 7 below for the behavorial simulation of the project.

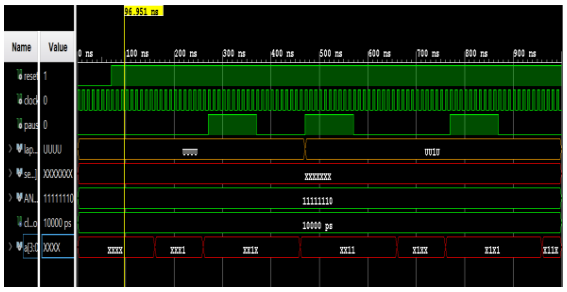


*Figure 7 – Behavioral Simulation*

Ultimately, the desired scope of the project was successful and the stopwatch functioned in accordance to the explained methodology. The desired functions worked properly when implemented into the DDR as well as a successful demo given to the instructor.

## Conclusions

This project helped develop a better understanding of digital systems or digital logic as a whole. There were components used in this project that were learned throughout this entire class; from logic gates to FSM's. Although this was a successful design, there is always room for improvement. There were some considerations an improved design but due to time constraints, knowledge of the subject, and unable to meet with fellow group members (COVID-19) for an improved collaboration these considerations were unable to be put into effect. One consideration is displaying the elapsed time while having the count continue. Essentially, not having to pause the count to display the saved elapsed times. Another consideration is to display the elapsed times on a different display, such as a LCD (Liquid Crystal Display).

Overall, this project was a great success and the knowledge learned will carry over to future classes as well as future careers.

## References


[1] Daniel Llamocca, *Electrical and Computer Engineering Department, Oakland University,* "VHDL(Unit 7): Digital System Design" Fall 2020

[2] Daniel Llamocca, *Electrical and Computer Engineering Department, Oakland University,* "Unit 7 – Introduction to Digital System Design" Fall 2020

[3] Daniel Llamocca, *Electrical and Computer Engineering Department, Oakland University,* "VHDL(Unit 6): Finite State Machines" Fall 2020

[4] Daniel Llamocca, *Electrical and Computer Engineering Department, Oakland University,* "VHDL(Unit 5): Sequential Circuits" Fall 2020

[5] Daniel Llamocca, *Electrical and Computer Engineering Department, Oakland University,* "Unit 6 – Sychronous Sequential Circuits" Fall 2020

[6] Daniel Llamocca, *Electrical and Computer Engineering Department, Oakland University, Reconfigurable Computing Research Laboratory,*

[7] http://www.secs.oakland.edu/~llamocca/index.html

Improved resolution schematics:

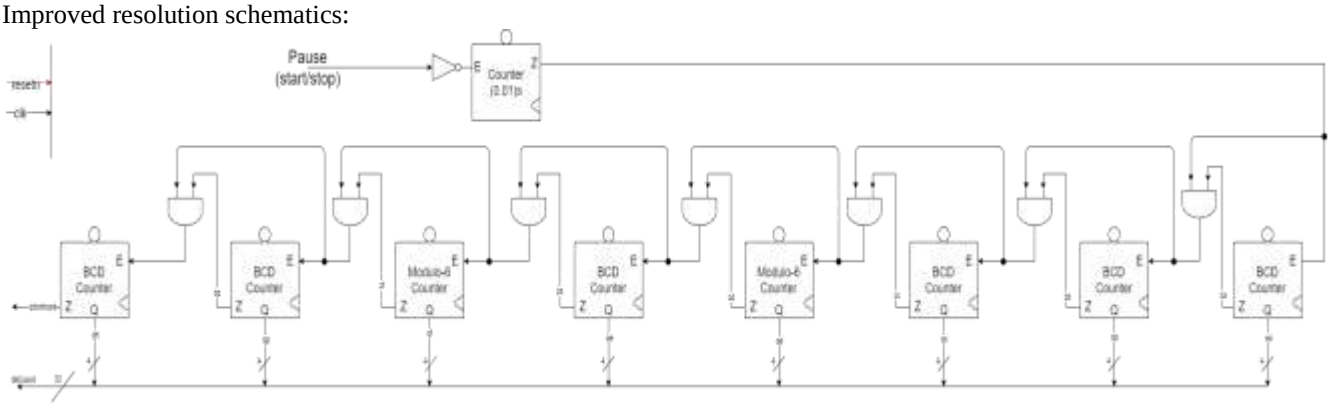


*Figure 1*

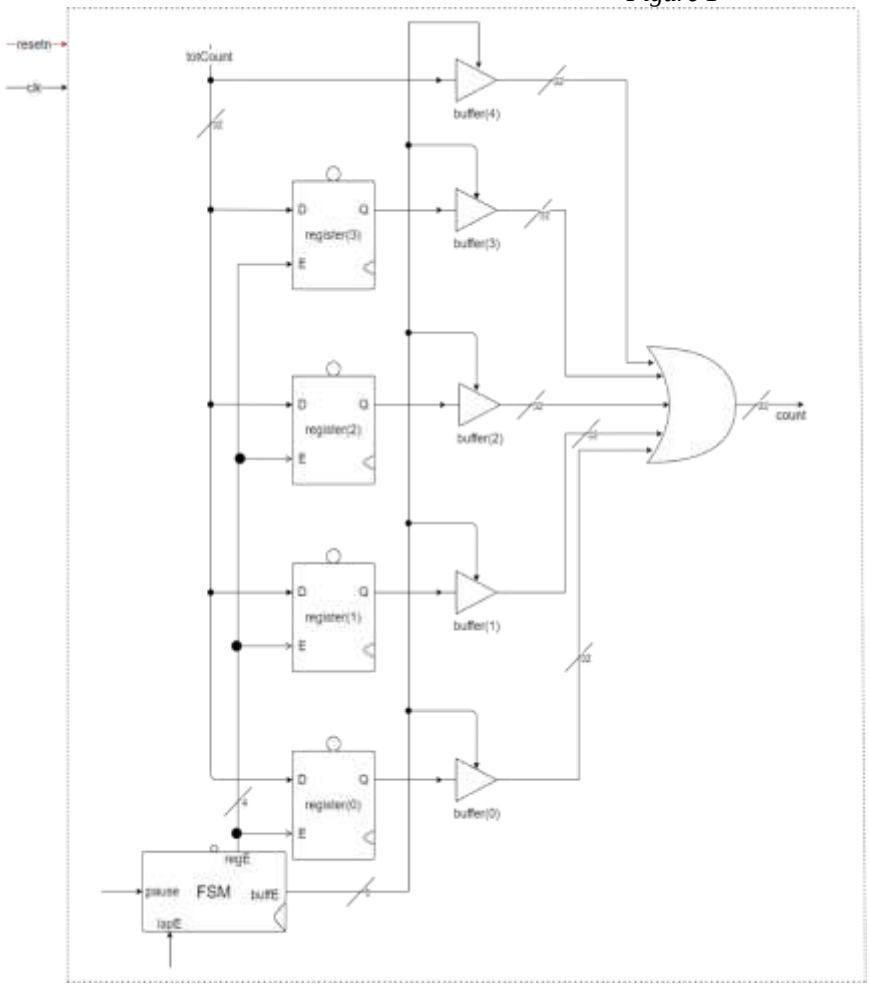


*Figure 2*

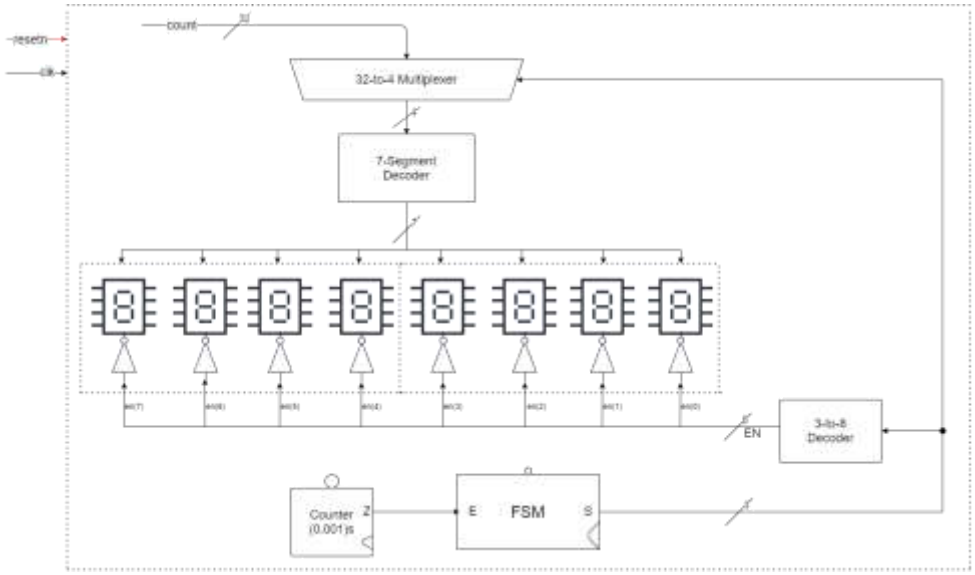


*Figure 4*

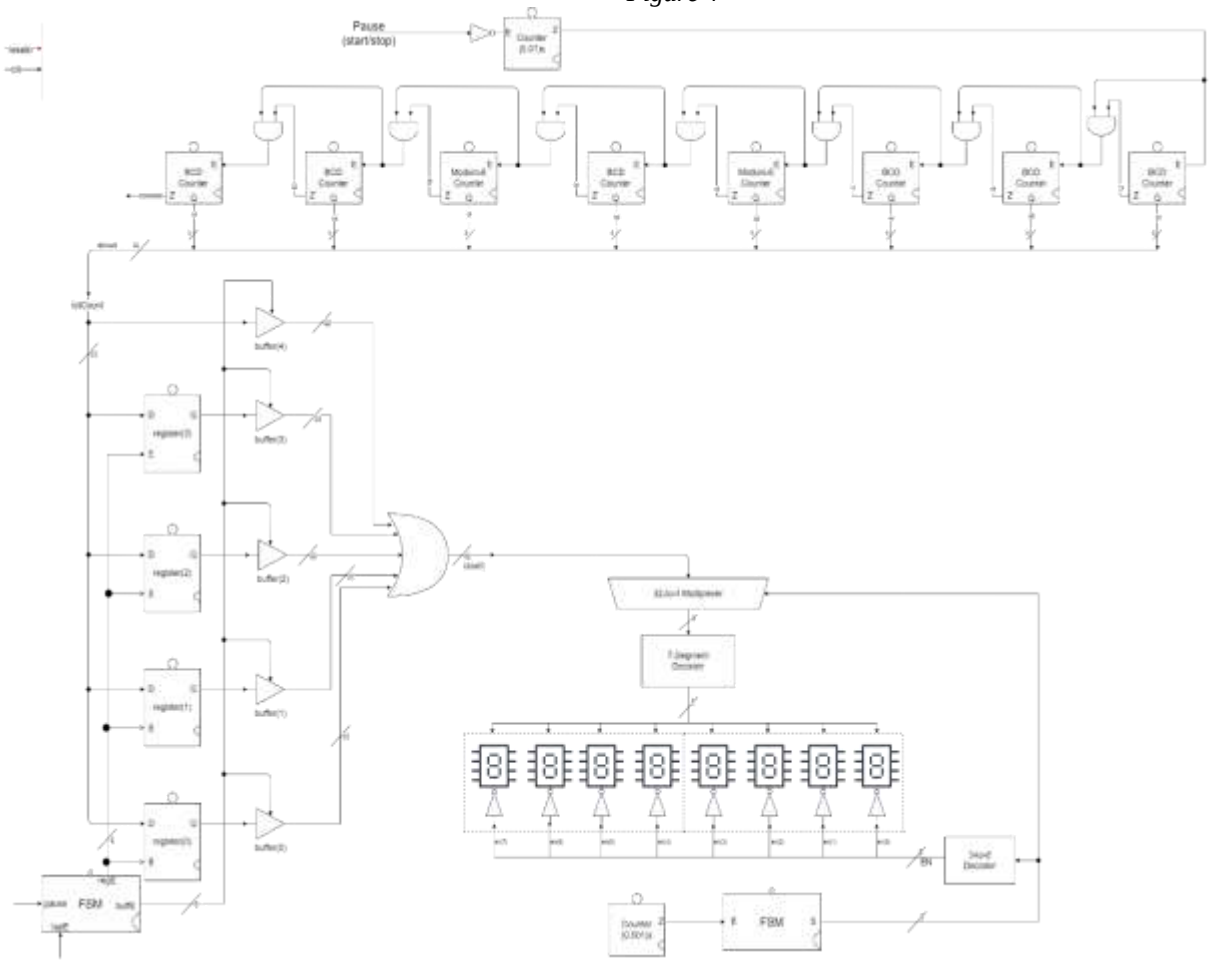


*Figure 6*

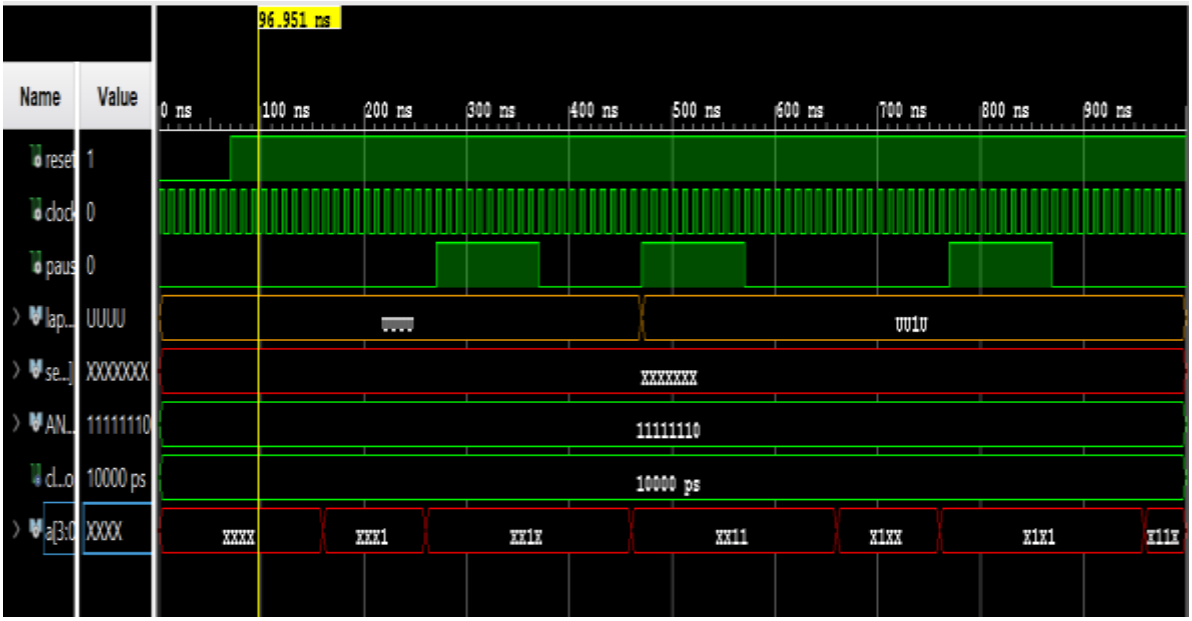


*Figure 7* (also attached with code)