Digital Stopwatch

ECE 2700 Final Project-Implemented in Vivado using VHDL

George Abouzeid, Lislie McHugh, Matthew Schodowski, Marissa Toma
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
gabouzeid@oakland.edu, lmchugh@oakland.edu, schodowski@oakland.edu, marissatoma@oakland.edu

***Abstract*—The objective behind this project is to design a fully functioning and effective digital stopwatch onto a FPGA board using every seven-segment display and featuring key specifications. Each seven-segment display is operated by a designated counter. This counter's count, or output bits every increment, is converted into a configuration which will light up the LEDs on the display to visualize that desired number. A random access memory emulator is used to control the lap feature. Through the use of registers and a Finite State Machine, each component of the design can piece together to complete the desired count. Each of the fundamental components worked perfectly to combine different aspects of digital logic design in order to implement the stopwatch. It was observed that the use of a random access memory(RAM) emulator proves more useful than initially expected. The project could have been improved by adding more laps, given additional registers were added, or researching a more efficient method for creating the laps.**

I. Introduction

This project details an extensive design and implementation of a digital stopwatch with provided lap functionality. Given the process of designing this project, the required components, various results, and decisive conclusions will be explained or demonstrated.

The purpose of this project is to create a digital stopwatch with various features using VHDL. The stopwatch will have start, stop, reset and lap features. Eight 7-segment displays will be utilized on a Nexys-4 field-programmable gate array (FPGA).

The desire behind this project is to get more depth into how the clock functionality is implemented using FPGAs and to observe how registers can store and eventually display desired outputs. The stopwatch will utilize Finite State Machines (FSM), counters, registers, and a seven-segment serializer, which involves a variety of sub-components as well. A basic background of VHDL along with port mapping and generic mapping can be used to complete most of the applications required in this stopwatch.

A stopwatch can be used in everyday activities for tracking the count or certain timing of an action. Stopwatches are mainly found in phones, wristwatches, or specific training type clocks found in some sports training. Some of the challenges posed that will be encountered during this project include implementing a proper method for storing and displaying required laps, and creating a stopwatch adequately visible, time-wise, given a common anode setup for the displays within the board.

II. Methodology

A. Top File

For the purposes of this design, a datapath circuit and a FSM can be used to create the entirety of the stopwatch. The top file can be broken down into three main components, which can be utilized of create a top file implementation. The inputs consisted of a pause switch, a reset button, two address switches, a store lap switch, and a show lap switch. Each register, counter, and FSM in this design is controlled by the same clock and resetn signal. Those on-board switched corresponded to a few substructures within the design. The pause switch is vital for being able to control the overall count on the visible display, while the address switched are able to choose which register the current lap is stored to. The store and show lap inputs toggle a current lap into a register and displays the lap, respectively. Overall outputs of the stopwatch simply consisted of the common anode control on the seven-segment displays as well as the converted data fed into each display accordingly. The three components, counting element, lapping, and serializer, each serve to maintain a continuous flow of the count or enable the ability to store a lap. The top level block diagram is shown below in **Figure 1.**

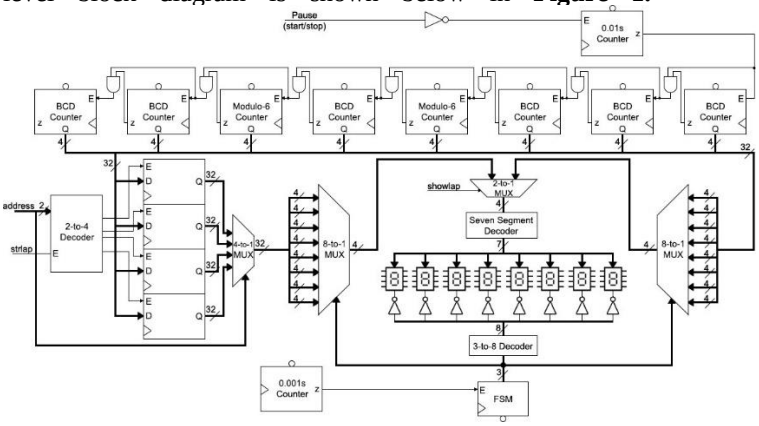

Figure 1 – Complete Block Diagram

B. Counting

While this digital stopwatch utilizes a total of ten counters, the counting function itself is implemented through the use of nine modulus counters. With a total of eight seven-segment displays, the maximum time that can be displayed is 99 hours 59 minutes 59 seconds and 99 hundredths of a second. One modulus counter will be directly responsible for producing the

increments of the stopwatch. Since the Nexys A7 has a built-in clock of 100MHz – or a 10ns period – this counter is used to adjust the 10ns to the correct stopwatch increment of one-hundredth of a second (0.01s) [1].

To effectively account for the maximum digits that can be counted, two types of modulus counters are utilized: six BCD counters and two modulo-6 counters. The BCD counters will serve to count from zero to nine. On the contrary, the modulo-6 counters will serve to count from zero to five. This is due to the nature of time, since a stopwatch increases from 59.99 seconds to a minute, and similarly at the hour mark. In other words, when a specific counter reaches its maximum count, the next counter is directly affected. Therefore, the tens place for minutes and seconds will utilize a modulo-6 counter, while the rest will utilize a BCD counter. The given input for this part of the stopwatch will be a designated start/stop switch. For simplicity, this input can be seen as the pause input. For each counter to increment correctly, the initial counter will control the enable of the smallest countable increment. In this case, the smallest count is 0.01 second, so the controlling modulus counter will thus have a count up to that of 10^6 to achieve this desired timing.

Whenever the pause switch is activated or switched to high, the controller counter must cease counting to stop the entire stopwatch at that exact time. To obtain this desired effect, a not gate is inserted before the enable of the controlling counter so that when the pause switch is active high, each counter ceases its count. An output from the controller counter, called z for most generic counters, is connected to the system's first BCD counter as its enable. This z signal coming from the 0.01 second counter will contain a very short pulse every time the counter reaches its highest count. So the first counter after the controller will only increment with the count of the controller. Every other moment, its enable will be low. This concept is how every counter increments properly based on which segment of the stopwatch it should be counting for.

Naturally, it would seem as though the output of the first stopwatch counter will feed into the enable of the next counter, or the counter holding .10 place's value. This is partially true because each subsequent counter requires the outputs of the previous two counters to know if it has reached a point to increment the count. Each subsequent counter will have its enable controlled by the AND operation between the previous counter's z pulse and its enable. This will maintain the proper counting design and ensure the stopwatch freezes the count when the stop switch eventually goes high. In summary, the following counter's enable is controlled by the previous counter's enable ANDed with the previous z pulse. With each of the counters working and incrementing properly, their separate outputs can be fed into the remaining components to combine the entire design. A separate 32 bit bus, combining all the output bits from each counter, will head to the RAM emulator. The counting component is seen in **Figure 2.**

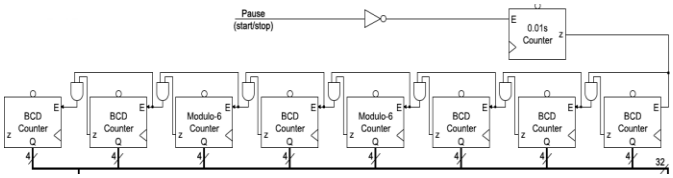

Figure 2 – Diagram of Counting Component

C. Lap Function with Random Access Memory Emulator

A unique feature of this stopwatch will be the lap functionality. The idea behind this is that, for however many laps, a certain elapsed time will be held in the "memory" until requested or used at a later time. This concept will be designed into the stopwatch using a few ideas learned throughout the course. For this stopwatch, four different lap inputs will be available to be stored and displayed as needed.

To implement this function, a general concept of a random access memory emulator is used [2]. In this design, four 32 bit registers are used to each hold a single lap. This provides a total of four laps during the use of the stopwatch. The input data into each register originates from the 32 bits which are combined from the eight counters in the previous section.

The RAM emulator begins with a decoder containing the address inputs as the main data and the store lap input as the enable of the decoder. The address is able to be controlled by switch 2 and 3 on the board, which corresponds to two bits, since two bits can represent the four different registers. A decoder works to create a single high bit, depending on the input data, in a respective bit position on the output into which it is feeding. For example, when the address is set to 01, the decoder output 0010 since the desired bit position in this case is one. Each register enable is controlled by this decoder to achieve the desired store behavior into whichever register is chosen. The address bits essentially selects which register the lap will be stored to through the use of this decoder. However, when the store lap switch is low, the enable of the decoder will be low and output zeroes to each enable of the registers.

In order to write the 32 bit data into the register, the store lap switch must be switched high then low. When the store lap is high, the data is constantly flowing into whichever register is currently selected and the lap count is not yet captured by the register. It is not until the store lap switch is back to low until the register writes and maintains this lap until either reset or given a new data.

The outputs of the four registers are fed into a four to one multiplexor. The select line of this multiplexor is controlled by the same two address bits that feed into the decoder. This is so that, when a register is chosen based on its address, the output of that register will also be selected by the multiplexor and serve as the output of the RAM emulator. While the RAM component usually has an enabled controlled multiplexor, this can be avoided since the serializer will be able to choose if the data coming from the RAM is purposeful or not. The decoder, registers, and multiplexor all create the RAM which controls the lap function at all times. The RAM is shown in **Figure 3.**

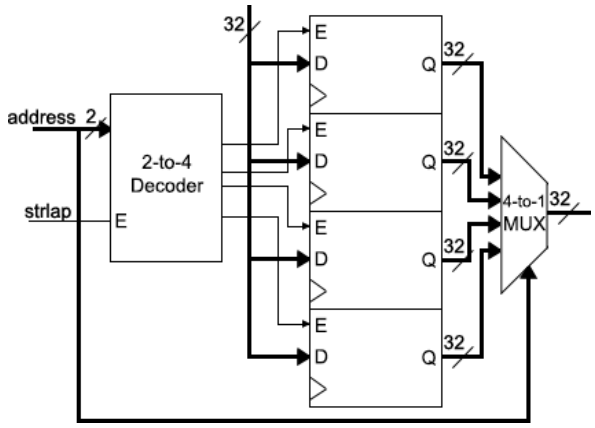

Figure 3 – Random Access Memory Emulator Diagram

D. FPGA Displays

All the seven-segment displays on board the Nexys A7-50T FPGA board will be used during this project to display the stopwatch timing. Due to the connections made on FPGA, each of the seven-segment displays cannot be turned on individually using a common anode method for each display. Instead, they are all connected using the same input, and if wanting to display different data on each display, an alternative method must be used. A trick to solve this problem would be to turn on the display individually, with all the others disabled, for a very short period with the desired data feeding into that display. It happens that the consensus for this amount of time should be one millisecond [1]. Because the display refreshes at such short time, to the human eye, it will have the same effect of all the displays being on simultaneously and show the expected outcome on all of the seven-segment displays. It is now a matter of figuring out how to turn on each of the displays at the right time, and feeding the right input into those displays in the same time interval.

E. 7-Segment Serializer

Given the problem encounter in the previous section, a seven-segment serializer does exactly what is required, with the components contained within it, to solve the issue [3]. The serializer is composed of a multiplexor, a counter, a Finite State Machine, and two decoders. Using the anode input of each display, we can control which one is enabled while the rest are disable. An eight-to-one multiplexor can usually be used to decide which four bits of data will be displayed. However, this is for the case with a stopwatch without the laps. Instead, two eight-to-one multiplexors can be implemented, alongside a separate two-to-one multiplexor.

The bits that originate from the counters at the start of this design, with their output of four bits, feed into one of the eight-to one MUX inputs. This MUX will feed directly into one as one of the inputs for the two-to-one MUX. For the other eight-to-one MUX, the inputs in this case will coincide with the 32 output bits from the RAM lapping component. Here, the 32 bits are broken apart into 4 bit components. The four least significant bits act as the first input into the MUX and this dividing of bits continues up until the last input of the MUX. The 32 bit bus is broken apart into eight four bit buses so that the MUX can select between the counter bits just as the other eight-to-one MUX. This output feeds into the other input of the two-to-one MUX. The select of both of the eight-to one- MUX will be controlled by the finite state machine to properly decide which of the eight data connections are allowed through.

After feeding into the two-to-one MUX, the show lap switch will act as the select in this case before encountering the main conversions of the serializer. When the show lap switch is low, the two-to-one MUX will continue to display the data coming from the main counters, and the stopwatch will run smoothly the entire time this switch is low. When changed to high, the MUX will instead allow the data coming from the RAM to pass into the serializer. So only when the show lap switch is high, the laps for any of the registers can be seen, as the stopwatch displays the lap time. If the stopwatch is not paused, it will continue to count in the background, regardless how long the lap is shown. The output of the two-to-one MUX will then feed into a BCD to seven-segment display decoder in order to convert the bits into a desired active low configuration for each distinct LED on the display to light up the right number.

To separate enables between the displays, a counter timing for one millisecond is needed to at least control the enable of the finite state machine. This counter will contain a count of 10^5 to achieve this timing value. The output of the FSM will connect to each of the eight-to-one multiplexors, and into a three-to-eight decoder, which will in turn connect with the anode of the displays. This is required to turn on the displays with their respective input. For example, if the select is 101, the 5th input into the MUX will be allowed through into the displays, but those bits should be displayed on the 5th display only as well. The decoder will ensure that all the other displays are off except for the 5th, and those bits will end up flashing their corresponding number for 1ms until the next pulse arrives. The diagram for the entire serialize is found in **Figure 4.**

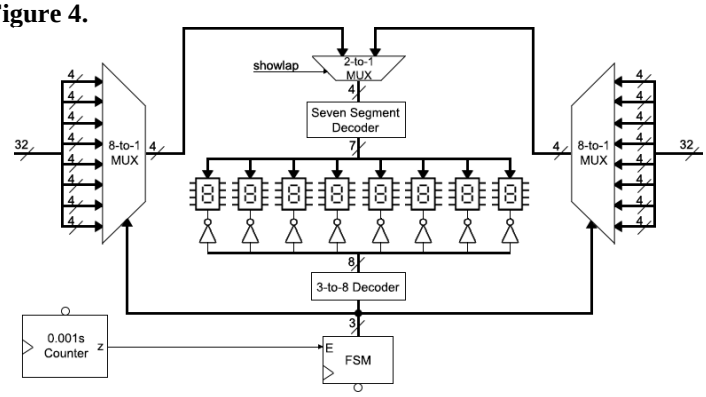

Figure 4 – 7-Segment Serializer Diagram

F. *Finite State Machine for Serializer*

As mentioned in the previous section, a FSM will be used to control the selects of a few components. The input of this FSM will be controlled by the output pulse z of the counter inside the serializer. The pulse z from the counter will feed directly as the input enable for the FSM so that when the counter reaches one millisecond, the machine will move to the next state and change the select output. With eight different inputs, there will be a three-bit select to choose between all those inputs, and therefore eight different states in the FSM [4]. Each state will hold a constant select value and wait for the next enable pulse to move to the next state. The Algorithmic State Machine (ASM) of this FSM is shown here.

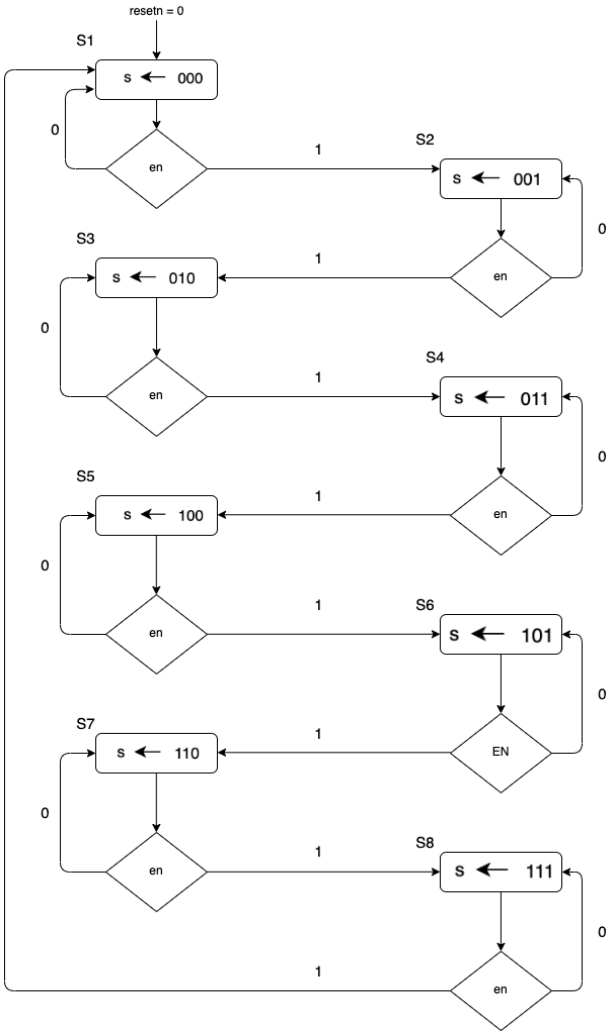

Figure 5 – ASM for the FSM within the Serializer

III. Experimental Setup

In order to test the functionality of the project, a VHDL testbench was created for the top file using the software Vivado. Additionally, an external interface test was used on the Nexys FPGA board to visualize the stopwatch count.

The testbench simulation was focused on showing the main count of the design. To efficiently see the results, the design itself can be minimized so that very long times are not required to be simulated in the waveform window. The alternative is to just have long run times, and this is due to the one millisecond counter controlling the FSM. The simulated waveform can be seen in **Figure 6.**

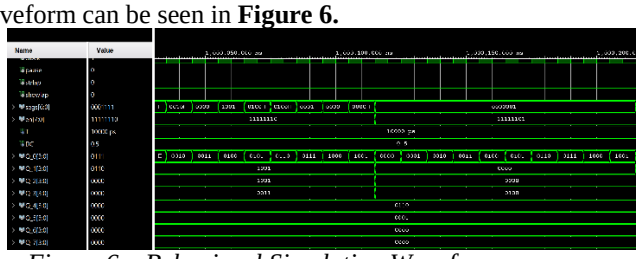

Figure 6 – Behavioral Simulation Waveform

With the external interface, the Nexys 50T FPGA was required due to the project and XDC file being create for such a board. Once the board was plugged into the computer via USB and on, the hardware manager in Vivado was able to connect with the board. After the bitstream was generated, the stopwatch began to run without any issues. Each switch function was tested, and a lap was written and read to each register to ensure the lap was working properly. The design was expected to contain the stopwatch continuously counting until obstructed, and this was shown in both methods of testing.

IV. Results

The functionality of the stopwatch worked as expected and the obtained results were enough to prove that the design works as intended. Every time the stopwatch design programmed onto the FPGA board and tested, it was consistent and unfaltering it the performance. Each designated switch indicated as an input served its purpose. There was some initial uncertainty with the RAM lap function and the inputs since there were some adjustments required to have the RAM work with the counting feature of the stopwatch.

Some of the adjustments made in the components can be confirmed in their usage. The serializer in this stopwatch varies from the common serializer utilized in most cases. Adding the various multiplexors into the design proposed a new challenge as to whether the stopwatch and the laps would show at the right time. Using the RAM as the lap function proved very useful as well. If desired, many more registers can be added in order to add more laps into the design. The only limitation would be the input limited on the board interface used.

When attempting to simulate this design using the testbench, it was difficult to create a waveform where all the desired signals are visible and correct. Some adjustments were required to get this outcome, but only due to minimization in the circuit itself.

The FSM of the stopwatch worked perfectly to control the flow of the datapath circuit. It was able to control both input multiplexors of the serializer and continue to have the displays turn on at the same time. While the design performed perfectly in this case, the FSM may be able to adapt to a different design of stopwatch and have more control over the overall datapath circuit.

CONCLUSIONS

This project adequately counts up until the maximum count and can be expanded further using a board which contains more displays. When using the Nexys 50T FPGA for implementation, the design performed as expected and without any issues.

For the purpose of storing data, a random access memory emulator is incredibly useful and can be used in any case where small amounts of data is required to be kept for some time. Eventually, the RAM will become too cumbersome with more addresses and a different method of implementation can be used instead. Similarly, the serializer can be changed to fit the needs of a design feeding many component outputs to the displays. With more components, the design can be inefficient, so the serializer can be made to simply use one multiplexor and instead change the component itself. In this case, one single multiplexor may be used to simply feed both the count and the laps into one single connection. This would require another FSM to control the lap inputs.

When testing a design with larger increments of time, design minimization seems like a requirement and can be difficult to obtain desired results. This is a remaining issue to be solved. Additionally, having a simulation which displays the laps as well as the count proved difficult as well, and an adequate solutions will have to be found to avoid this problem in the future.

This project can be improved by trying to condense the lapping function into a more efficient subcomponent overall. When considering the exponential increase of the RAM, the design may become too repetitive and inefficient for continued implementation. With more desired addressed, the number of registers needed continues to increase. In this design, a few registers with a FSM control can replace the component.

With that being said, the project still performed in an outstanding fashion with a unique design. The digital logic design knowledge obtained throughout the course was vital in completion of the project. This project can further push to test the horizon of hardware design overall in further studies.

REFERENCES

[1] D. Llamocca, "VHDL Coding for FPGAs Unit 7," *Reconfigurable Computing Research Laboratory (RECRLab), Electrical and Computer Engineering Department, Oakland University*.[Online].Available:http://www.secs.oakland.edu/~llamocca/Tutorials/VHDLFPGA/Unit%207.pdf.

[2] D. Llamocca, "Unit 6 – Synchronous Sequential Circuits," *Reconfigurable Computing Research Laboratory (RECRLab), Electrical and Computer Engineering Department, Oakland University*.[Online].Available:http://www.secs.oakland.edu/~llamocca/Winter2021_ece2700.html.

[3] D. Llamocca, "Unit 7 – Introduction to Digital System Design," *Reconfigurable Computing Research Laboratory (RECRLab), Electrical and Computer Engineering Department, Oakland University*. [Online]. Available: http://www.secs.oakland.edu/~llamocca/Winter2021_ece2700.html.

[4] D. Llamocca, "VHDL Coding for FPGAs Unit 6," *Reconfigurable Computing Research Laboratory (RECRLab), Electrical and Computer Engineering Department, Oakland University*.[Online].Available:http://www.secs.oakland.edu/~llamocca/Tutorials/VHDLFPGA/Unit%207.pdf.

[5] D. Llamocca, *VHDL Coding for FPGAs*. [Online]. Available: http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html.

APPENDIX

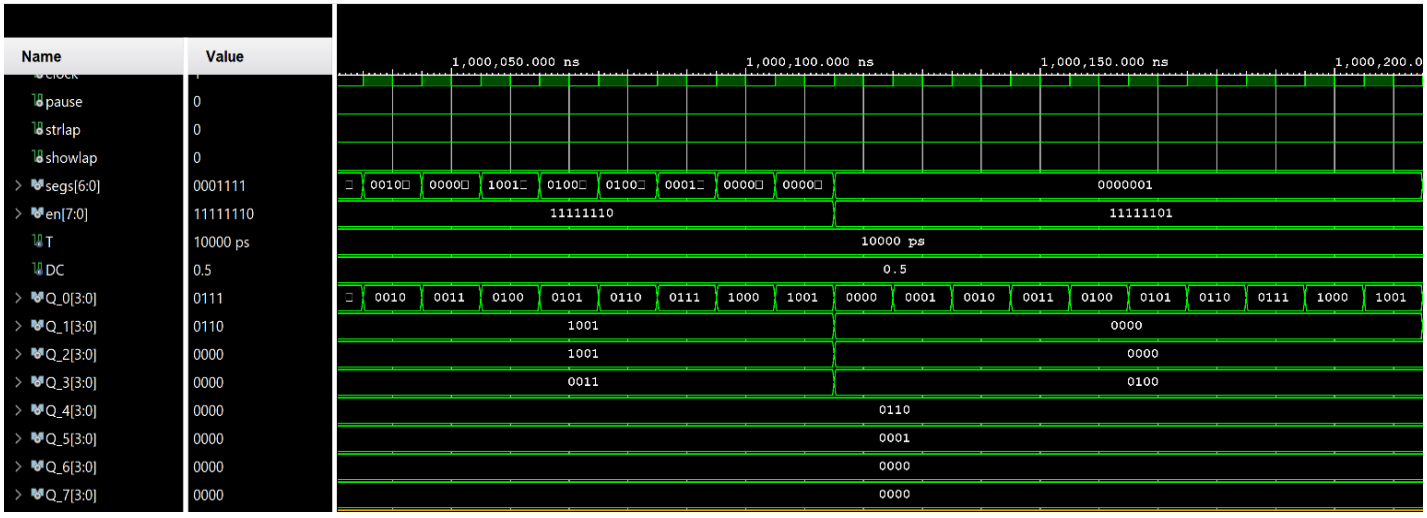

Figure 7 –Enlarged Behavioral Simulation Waveform

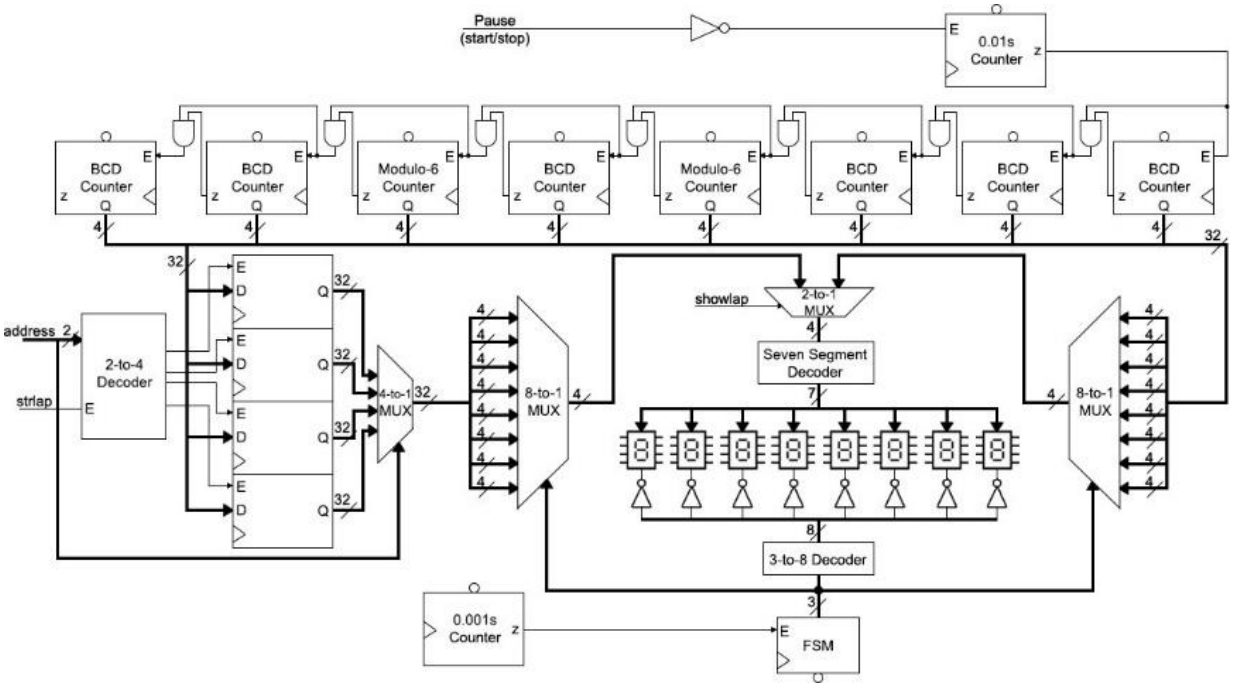

Figure 8 –Enlarged Top File Diagram

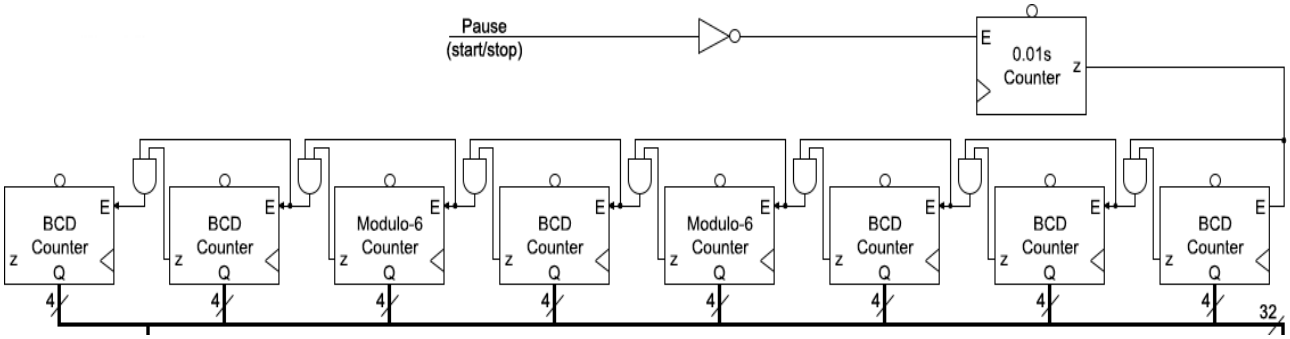

Figure 9–Enlarged Counting Component

Demo URL: https://www.youtube.com/watch?v=NXjo55t8gvE&ab_channel=MarissaToma