Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

3C7 – Lab F

Student Name:	**Aaron Dinesh**
Student ID Number:	**20332661**
Board Number	**8**
Assessment Title:	**Lab F**
Lecturer (s):	**Shreejith Shanker**
Date Submitted	**31/03/23**

Signed: [illegible]

Date: 31/03/23

Introduction

This lab served as the introduction to synchronous design in Verilog. In this lab we had to implement a variety of circuits whose state transitions were synchronised with the use of a clock and made use of flip flops to hold the states between clock cycles.

Experiments and Observations

Lab F was split into 4 parts and this section will be broken down as such. It will detail the steps taken to complete the relevant part and also list any screenshots and block diagrams necessary.

Part A

Part A dealt with the use of d-type flip-flops, the basic building blocks of synchronous designs. On every clock cycle (rising or falling edge depending on the configuration), they can store and output the logic level presented on their D pin. They can also be configured to have a synchronous (on the edge of a clock cycle) reset, or an asynchronous reset (resets whenever the reset pin goes high). The first part of the lab required me to create a clock with a 20ns period. This was done using an initial block to set the initial clock value and then use a forever block to invert the clock and generate the pulse. Once I instantiated the provided flip-flop module (d_type_ff.v), I also created a testbench that would provide the clock, data and reset signals as described in the lab report.

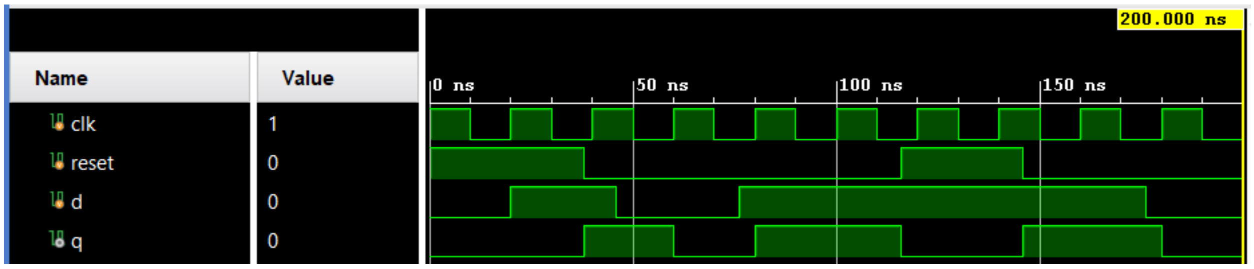

Figure 1: Async Reset, Rising Edge Trigger

Figure 1 shows a rising edge triggered, asynchronous reset, d-type flip-flop. At the beginning when the reset is high, Q does not follow D, but as soon as the reset signal goes low, Q copies D on the rising edge of the clock pulse. This can be seen again during the 50ns to 60ns mark. D transitions from high to low around 47ns but Q does not respond until 60ns when the next clock rising edge occurs. So it is evident then that this waveform is for a rising edge triggered, asynchronous reset d-type flip-flop.

Next part A required that we edit the d_type_ff.v file to make it a falling edge triggered, asynchronous reset flip-flop. This was done by changing the sensitivity list of the always block to read “negedge clk” instead of “posedge clk”. The reset was also changed to be “posedge reset” to ensure that the module would only reset if the reset signal went high. The module was then simulated, and the waveforms can be seen in figure 2. The previous testbench was also used to test this new module.

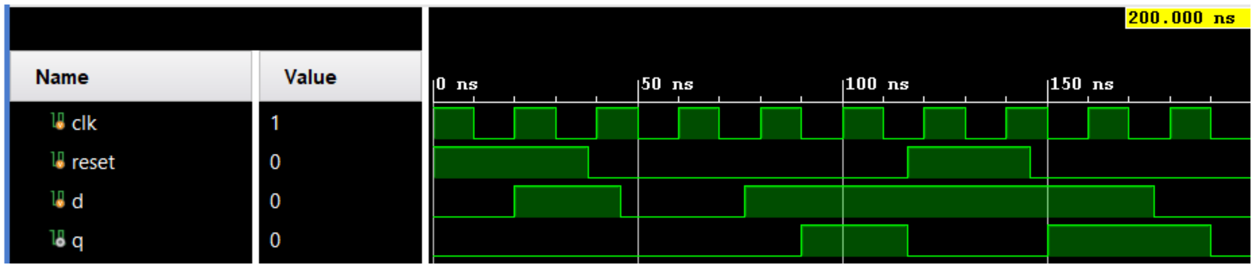

Figure 2: Async Reset, Falling Edge Trigger

Again, for the first 38ns we see the asynchronous reset signal resetting the flip-flop regardless of what the D signal and the CLK signal are doing. Only on the falling edge of the clock at 90ns does the Q output change to follow the D input. This continues till 118ns when the reset signal goes high and the flip-flop resets regardless of whether there was a falling edge of the clock.

For the last part of this section, the module had to be changed once again, but this time we needed to have a synchronous reset signal. Since the trigger wasn’t specified, it was assumed that the entire module would update on the falling edge of the clock. Again, for this module the previous testbench was used. The results of the simulation can be seen in Figure 3.

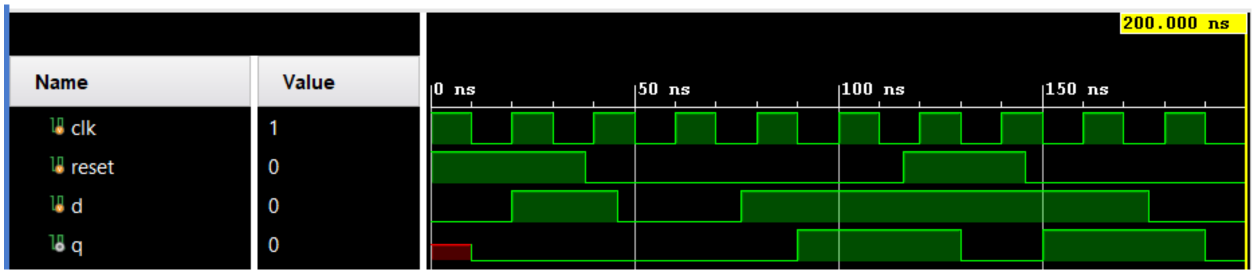

Figure 3: Sync Reset, Falling Edge Trigger

From 0-10ns the output is undefined because there is no falling edge to latch the module to a particular value. But as soon as the falling edge occurs, the flip-flop immediately latches to 0 as it should. The next interesting timestamp is around 115ns, the reset signal goes high in the middle of a clock period. If this was an asynchronous reset module, then the output would immediately fall to 0. However, since output transitions can only happen on the falling edge of the clock, we see that it doesn’t take effect until 130ns. Depending on the use case of this module, this 15ns delay might be acceptable and might not affect the performance of any other module this flip-flop is contained in. But in some cases when we need to reset to a known state immediately when the reset signal goes high, then this 15ns delay might not be acceptable. For the purposes of this lab, this was determined to be an acceptable compromise.

Part B

Part B of the lab tasked us with using the flip-flop for a practical purpose. Mainly these flip-flops are used in conjunction with combinational circuity to allow them to store a previous state and then use that stores state to transition to another state. Using the flip-flop module given in the previous part as well as the debouncer.v and the sevenseg.v module, I created a new module top level module that instantiated them and connected them together as shown below in Figure 4.

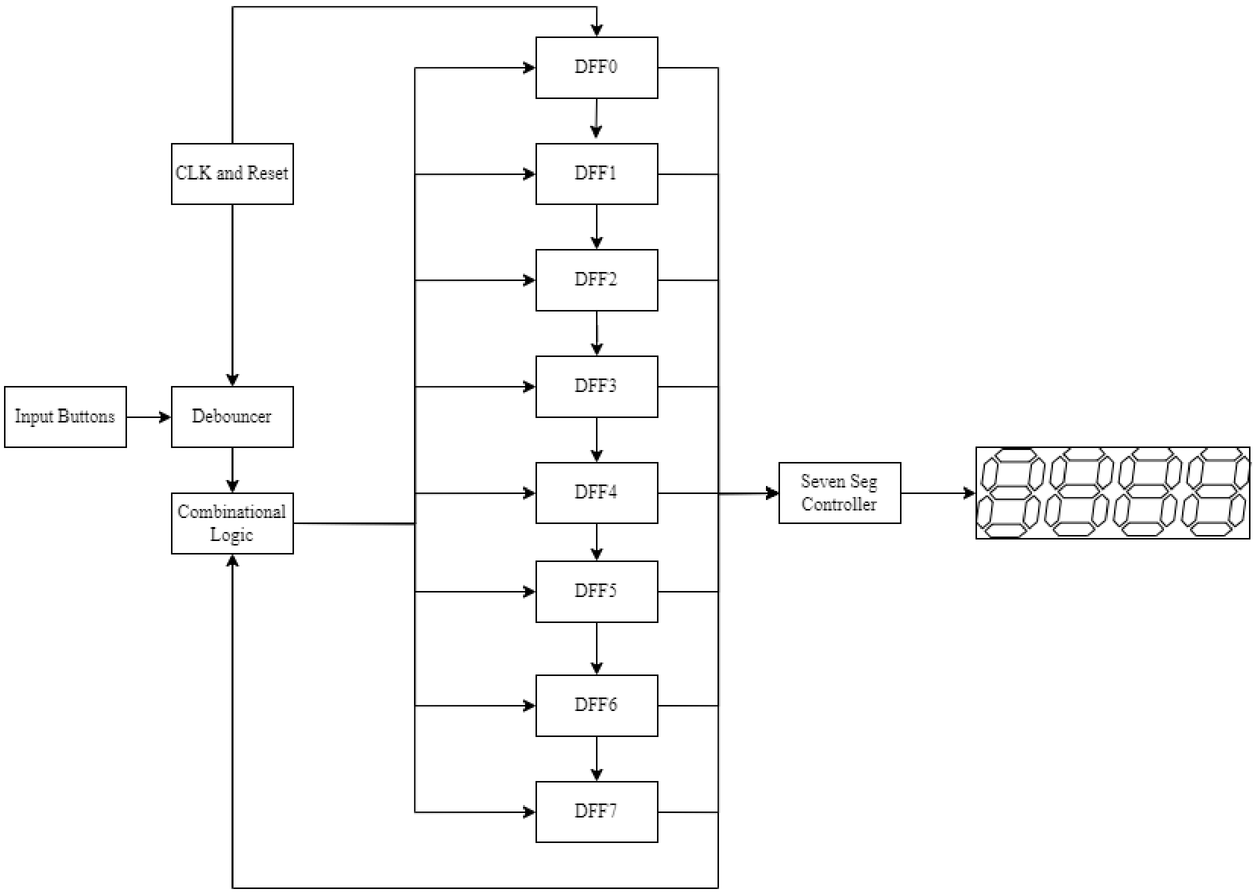

Figure 4: Temp Controller Block Diagram

The above figure shows the working block diagram of a temperature controller built using the internal clock of the BASYS3 FPGA, the 5 push button switches and one of the toggle switches on the board. The input buttons were first passed through the the debouncer module. This purpose of this module was to slow the polling rate of the buttons in-order to prevent the bouncing that occurs when clicking push button switches. After writing the design to the board, I noticed that simultaneous button presses would not update the state of the temp controller. In certain extreme cases when a single button was pressed very fast, the debouncer module would seem to lock up and never output a value, while the button was being pressed. From using the board, it seems to be that a pressing the same button every half second, would still register as different button pressed. The output from the debouncer circuit then passed into the combinational logic part of the design. This would take the current state of the flip-flops and then incriment the value if the UP and RIGHT button were pressed, decriment the value if the DOWN or LEFT button is pressed, and reset the value to $(22)_{10}$ if the CENTER button was pressed. The output of the flip-flops was then passed to the seven segement controller which then displayed the stored value on the seven segment display. The SW0 switch on the board was used as a master reset switch. Toggling this switch high for a second would reset everything to 0.

Part C

This required the implementation of a Linear Feedback Shift Register (LFSR). The primary use case of an LFSR is to provide pseudorandom string of binary digits that can be used in all areas from cryptography to telecommunications. From the sheet provided I determined that my LSFR should be 19 bits long and use an XOR tap scheme. The goal when building an LFSR is to build a maximal length LFSR. This is an LFSR that will cycle through all 2^{n-1} before coming back to the seed value. The choice of taps have a great impact on whether the LFSR will have this property, choosing the wrong taps would result in creating a shorter period non-maximal LFSR. The positions of the taps were determined by looking at the powers in the maximal-length feedback polynomials for a shift length of 19 [1]. These tap positions were determined to be 18, 17, 16 and 13. A block diagram of the circuit can be seen below in Figure 5:

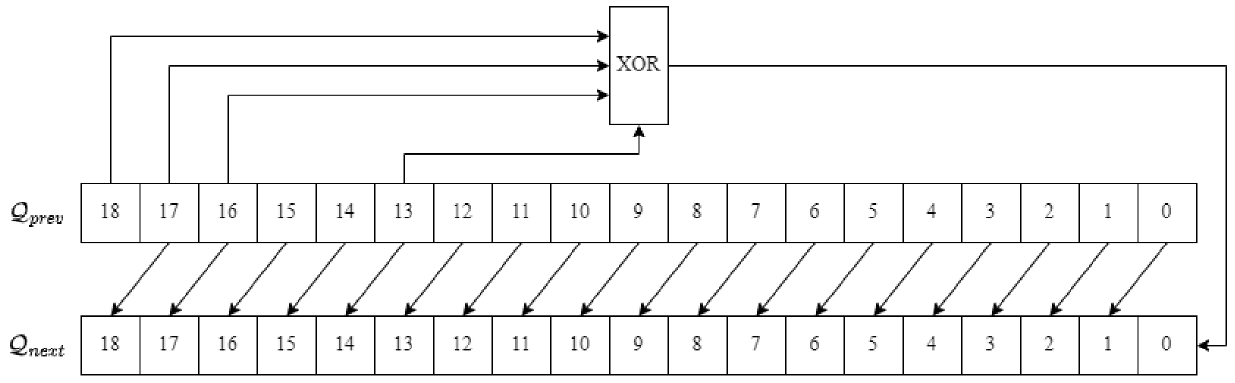

Figure 5: Maximal Length LFSR

On every rising clock edge, the bits in Q_{prev} was shifted one position to the right, while the feedback created from the XOR gate appended to the end of the register. This cycle would then continue ad infinitum, provided that the seed value was not one of the forbidden values. Since this is an XOR LFSR, having a seed value of all 0s would be a forbidden value. If we set the seed value to be 0s then we would never generate any feedback value other than 0 (since 0 xor 0 is 0) and the register would never change. And the circuit demonstrates this behaviour as well as evidenced by the waveform in Figure 6:

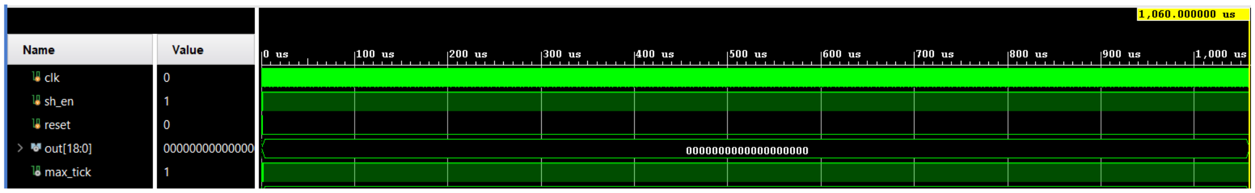

Figure 6: XOR LFSR Seed Value of 0

As we can see from the figure above, the out[18:0] register never changes even though there is a valid clock and shift enable signal (sh_en) and the reset signal is held low. So long as this forbidden value is avoided, the LFSR will be able to cycle through all the 2^{n-1} permutations. To demonstrate the functionality of our LFSR, we were told to set the seed value to be the bitwise XOR of our board number and the last 3 digits of our student id. In my case this seed value turned out to be 0000000001010011101 (board number = 8, last digits of student id = 661). I also made the design choice of choosing a clock period of 2ns. This was done so that the LFSR would be able to quickly cycle through all possible values. I also added a "max_tick" that would only go high once we have gone through all possible permutations and have come back to the seed value. The results of this simulation are gathered in Figure 7:

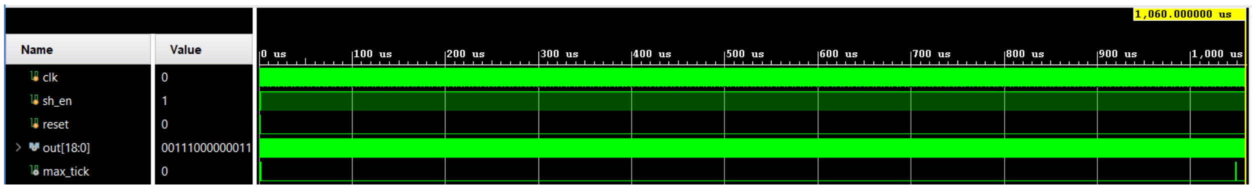

Figure 7: Maximal Length LFSR Operation

Looking at Figure 7a illustrates the operation of the LSFR more clearly and how the out[18:0] register only changes on the positive edge of each clock cycle:

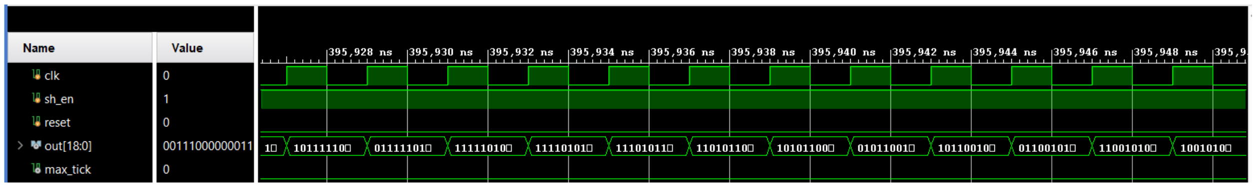

Figure 7a: Maximal Length LFSR Operation

Looking at Figure 7 we can also see that the max_tick signal gets asserted briefly around 1,050us when the output comes back to the seed value.

Part C Advanced

The advanced part of the lab tasked us with developing a module to count the frequency of the 1s and 0s in the MSB of the LFSR output. The concept for the N bit MSB counter was that it would receive an $N - 1$ bit input register, a 1 bit "to_count" wire and a reset signal. The counter would then output an $\left(\left\lceil\frac{N-1}{2}\right\rceil + 1\right)$ bit output register that would count the number of times, it has seen the "to_count" bit in the MSB of the input signal. This is better demonstrated in the block diagram and the finite state machine diagram in Figures 8 and 8a:

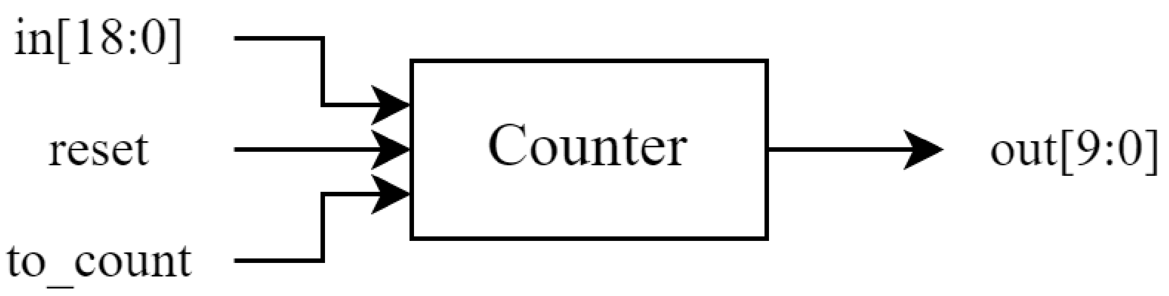

Figure 8: Counter Module with N = 19

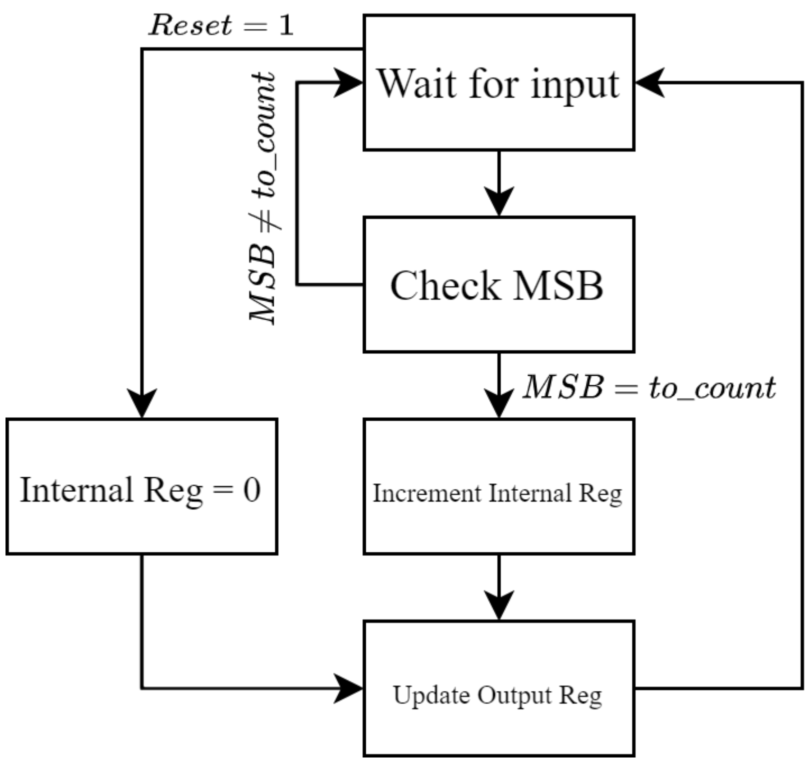

Figure 8a: Finite State Model of Counter

This module was instantiated twice in the LFSR, one to count the 1s and another one to count the 0s. Using the same testbench from before, the new module was simulated, and the output waveforms are shown below in Figure 9:

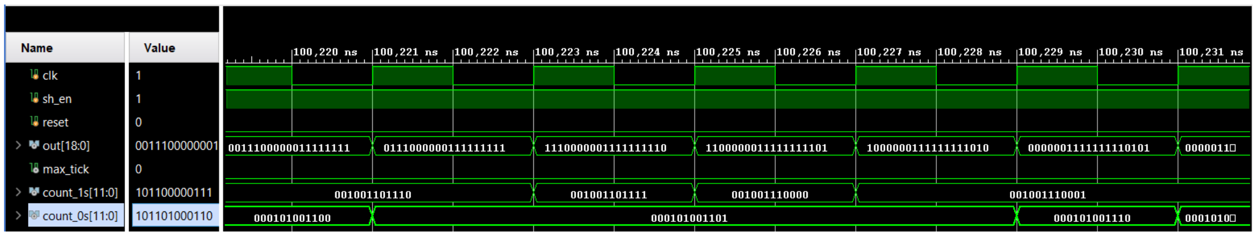

Figure 9: MSB Counters in Operation

The waveform above shows the 1s counter and the 0s counter performing their job and only incrementing on the rising edge of the clock. Notice the time period from 100,221ns to 100,228ns, we have a string of outputs where the MSB is 1. In this case only the count_1s register gets incremented while the count_0s register stays the same. Next we were asked to demonstrate the reset capability of the counters. In order to do this, I modified my test bench to set the reset signal high after a set amount of time. The results of this are shown in Figure 10:

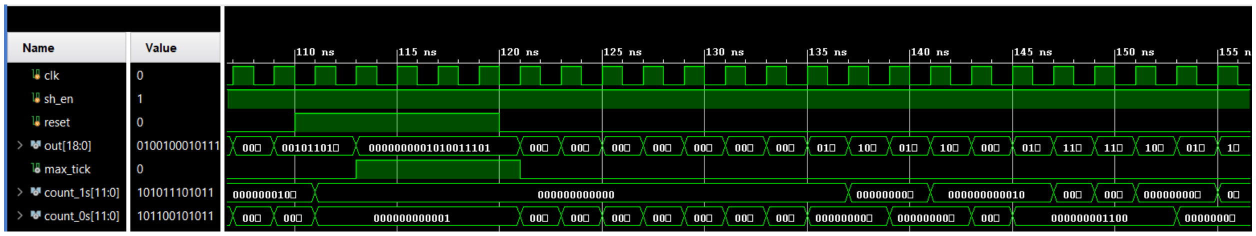

Figure 10: Reset Signal Resetting the Counter

As we can see, the reset signal goes high at 110ns which reset both the counters. They then begin to count up from 0 based on what the input signal was. The counters were also required to reset back to 0 when the max_tick signal was asserted. This can be seen in Figure 11:

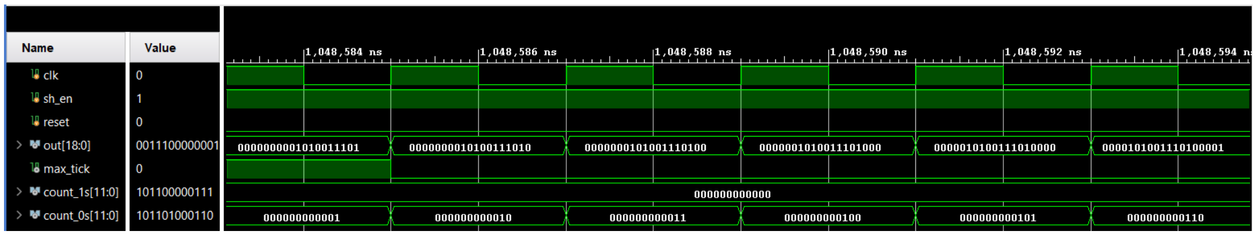

Figure 11: Max_tick Signal Getting Asserted

Once the max_tick signal gets asserted, it means that we are back at the seed value and must set the counters back to 0, then we can begin counting again.

Part D

The final part of this lab required us to implement the LFSR module from part C onto the Basys3 board. To do this a few changes had to be made to the LFSR module and a new clock divider module had to be implemented. The clock module took in the board clock signal as an input and scaled it down to 1Hz. This 1Hz clock would then act as the clock for the rest of the designs. To show the output from the LFSR, the LED row on the board was used. However, due to the limited number of LEDs on the board, the full output couldn't be displayed at the same time. To get around this, I limited the output of the module to be 10 bits instead of the full 19 bits. Then depending on the state of a switch the user could select whether the lower 9 bits or the upper 10 bits were displayed on the LEDs. The reset switch was tied to V17 and was active high, the button to switch which bits was being displayed was tied to V16 and the max_tick signal was tied to L1.

References

[1] Wikipedia. "Example Polynomials for Maximal Length LFSRs." https://www.wikiwand.com/en/Linear-feedback_shift_register#Example_polynomials_for_maximal_LFSRs (accessed 30/03/23, 2023).