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

3C7 Assignment 2

Clovis Hanbury-Tenison

20331969

3C7 ASSIGNMENT 2

Student Name:	Clovis Hanbury-Tenison
Student ID Number:	20331969
Assessment Title:	Assignment 2
Lecturer (s):	SHREEJITH SHANKER
Date Submitted	14/04/2023

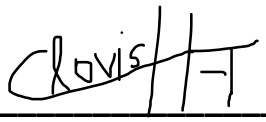
I hereby declare that this assessment submission is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I hereby declare that I have not shared any part of this submission with any other student or person.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at: http://www.tcd.ie/calendar

I have also completed the Online Tutorial on avoiding plagiarism 'Ready, Steady, Write', located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write

I am aware that the module coordinator reserves the right to submit my exam to Turnitin and may follow up with further actions if required should I be found to have breached College policy on plagiarism

Signed: 

Date: 14/04/2023

ABSTRACT

For Assignment 2 we were to implement a FSM, finite state machine to our LFSR from Lab F. A Moore counter was to be implemented in our FSM so that we could check for iterations of the binary pattern 10011110011. Our LFSR consisted of 17 total bits. The first bit was the feedback bit which was an XOR output of our 17^{th} and 14^{th} bit, as instructed in the Xilinx documentation.

As the pattern we were looking for was 11 bits and our LFSR consisted of 17 bits. We used 6 Moore counter machines to count all the iterations through the LFSR. Our Moore counter machine consisted of 11 states, corresponding to each bit of our pattern. Each state has two next states which would be called depending on our input bit. This meant we would be able to minimize the operation by not returning to state0 every time the next bit was unideal for our pattern.

The pattern counter was reset every LFSR cycle as our aim was to count the number of patterns in one cycle. The FSM was then implemented onto our Basys board along with our LFSR, and the count was displayed on our seven segment display and count of our pattern in our LFSR was found to be 312 appearances.

INTRODUCTION

A Moore counter Finite State Machine was create for Assignment 2. An FSM is a system that is able to simulate sequential logic. The FSM we create had 11 states, and each had two next states which can be seen in the diagram below. The FSM was created to find a pattern in our LFSR. This was then counted so that we could display the amount of times the pattern appeared in one cycle of our LFSR. I n a practical implementation of an FSM, an operation could happen whenever this pattern is detected.

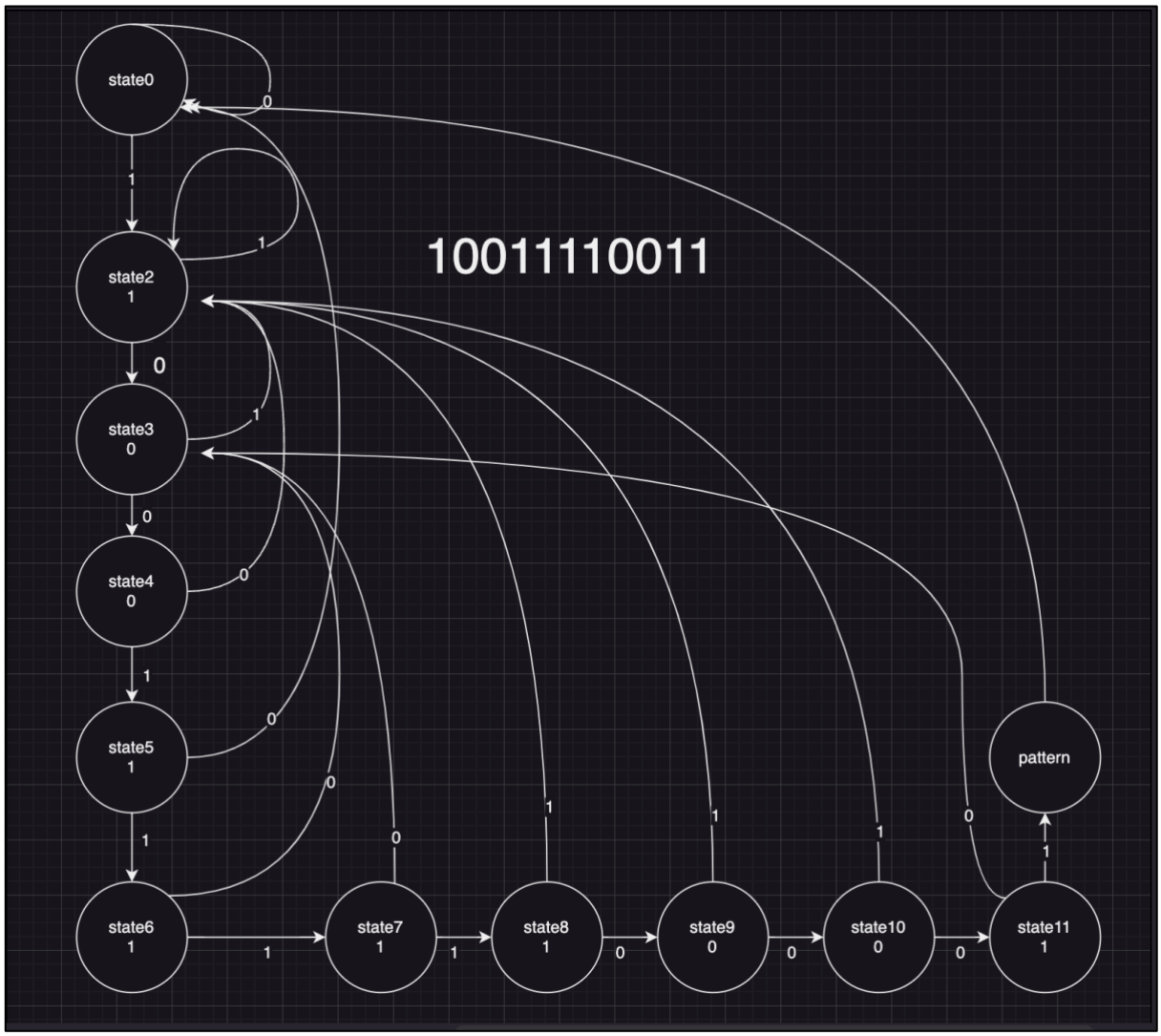


Figure 0 (FSM sequential states)

BODY

For our Top Module, we used our LFSR submodule and we used our 6 Moore Machines modules which took in bits 0, 1, 2, 3, 4, 5 of our LFSR so that we could account for. all of our pattern appearances. We also implemented the seven segment display submodule and the CLK module, so that we would be able to run the programme at our desired speed using the Basys boards Clock and display our pattern count on the seven segment of the Basys board.

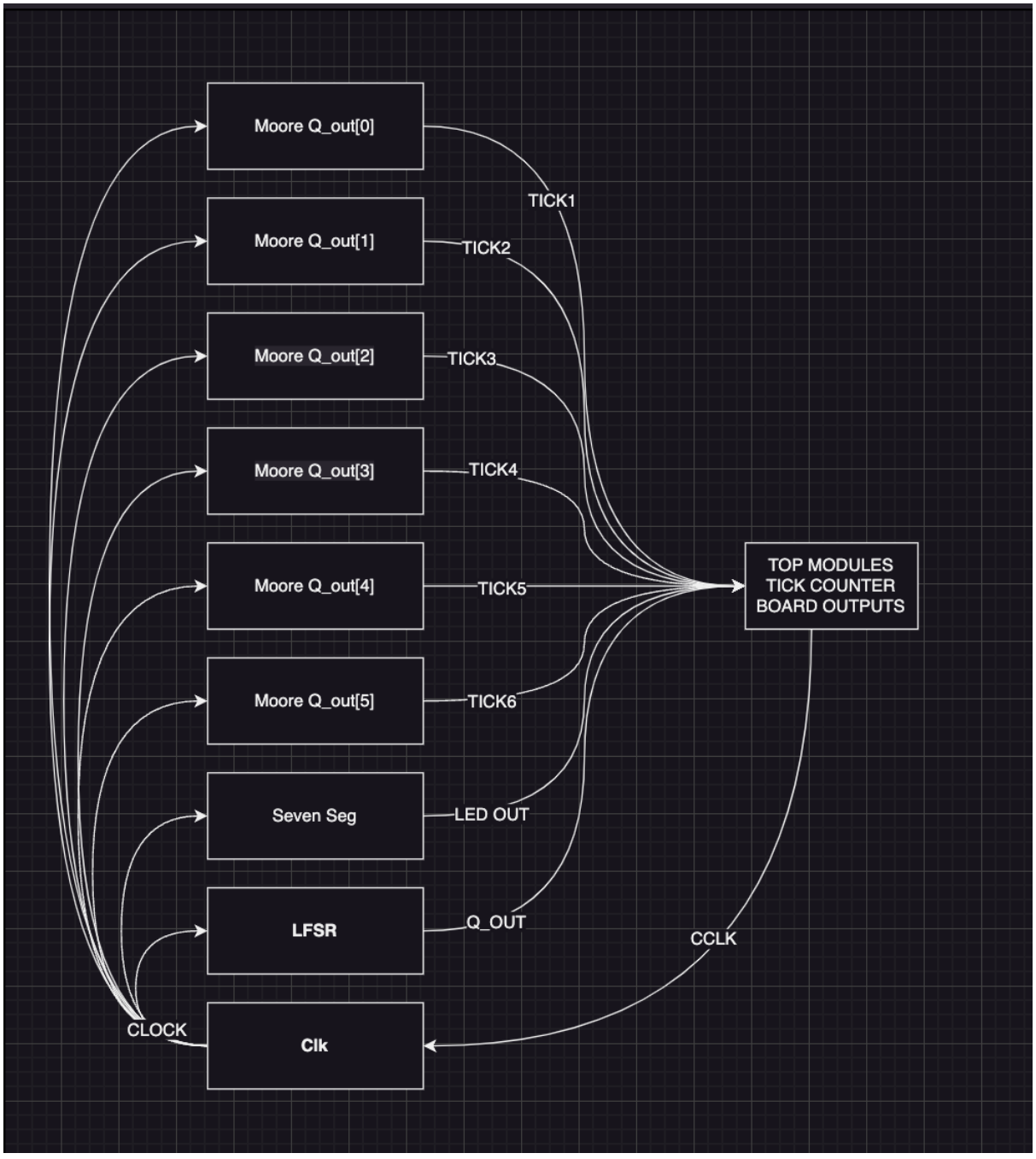


Figure 1.0 (Block Diagram of System)

Sources/Constraints/simulation Hierarchy

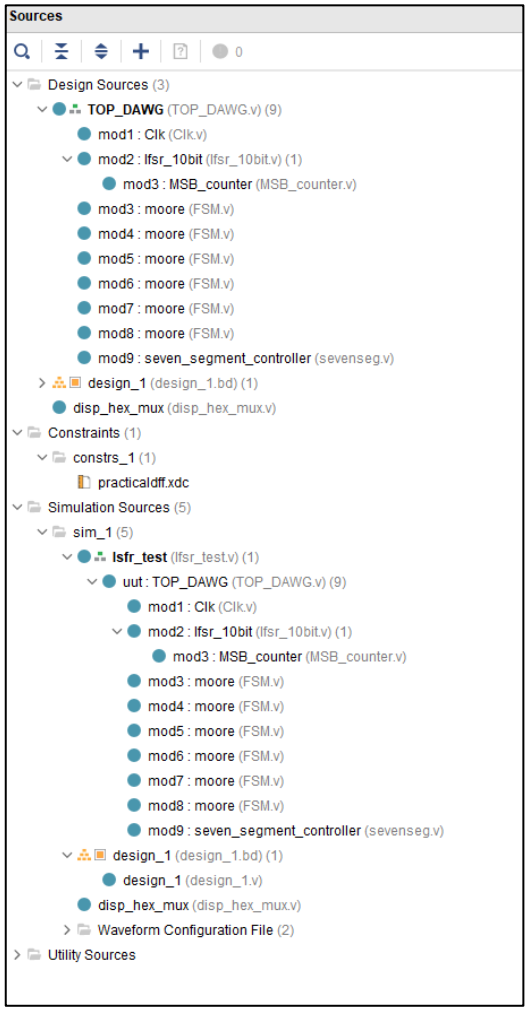


Figure 1.1 (List of modules, test case and constraints file hierarchy)

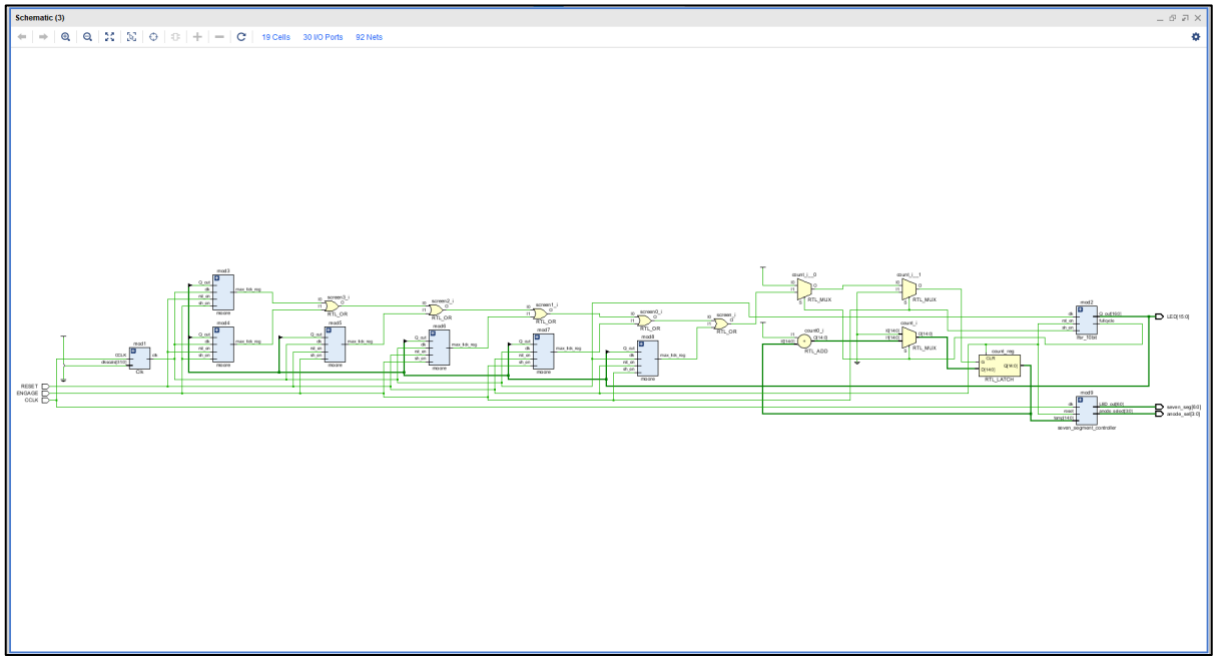


Figure 1.2 (Diagram of the complete system)

Our FSM was created using 11 states in an always block. Each state checked if the desired input LFSR bit matched with the next state and if so incremented to the next state and if not went back to the last state which would keep the pattern sequence going. This was done for each of the 11 states and in the last case, if the pattern was found, a max_tick_reg was set to high while also restating the state back to the 1st.

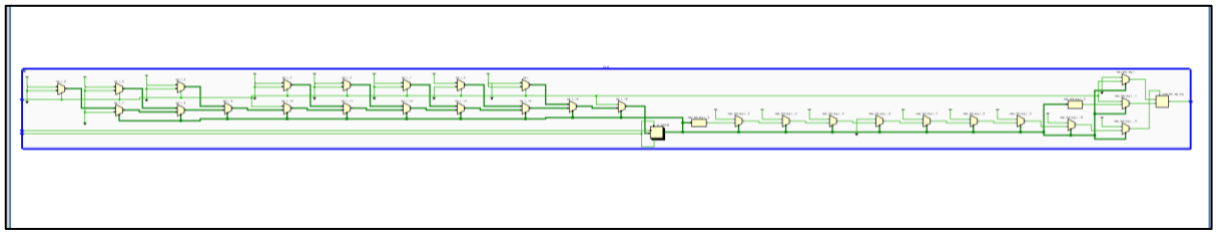


Figure 1.3 (Schematic of the FSM /Moore Machines)

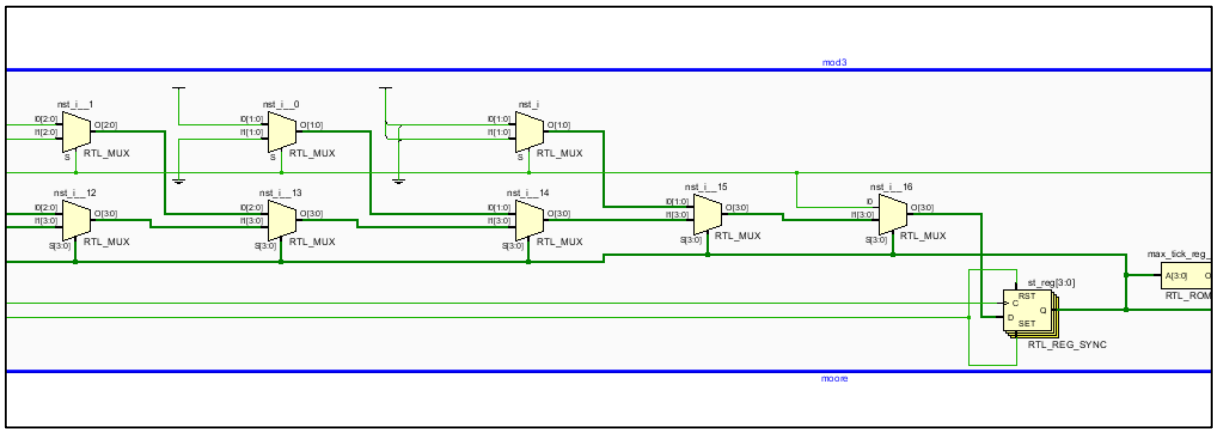


Figure 1.4 (ZOOM of Schematic of the FSM /Moore Machines)

The pattern tick outputs of each of our Moore counter machines s1, s2, s3, s4, s5 and s6 were def through OR logic so that we could assign them to a single wire called screen, we then used this to increment our counter. This was done in our top module, extra conditions to reset the count back to 0 were set for reset and when our LFSR cycle was over.
The seven segment display was taken from the provide code in Lab F and altered slightly, with each display configured to display the value using remainders and division, as Verilog and binary do not accept decimals, this would automatically take our value and reduce it to the desired number as can be seen in the module file.

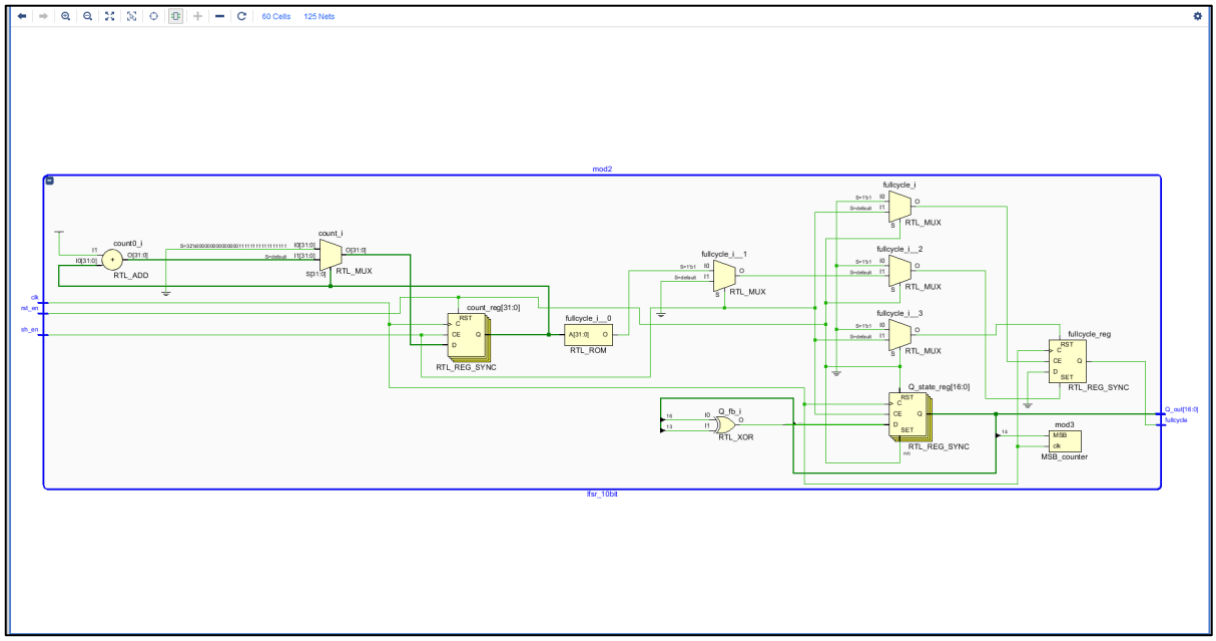


Figure 1.5 (ZOOM of Schematic of the LFSR)

The test was ran and the waveforms were found for our output, clock, tick and counter. We ran the test bench initially just taking the 1st bit of the LFSR into a Moore Machine and then tested again when we had implemented multiple Moore machines for each of the first six bits. Once the desired waveforms were found and we confirmed that our FSMs and counter was working correctly, we created a constraints file. The constraints file mapped 16 of our 17 bits to the boards LED's as we could not fit all 17. The V17 switch was set to ENGAGE and the V16 set to RESET, allowing us to halt and reset our LFSR. The Seven segment constraints were then added similarly to the constraints for seven segment provided in LAB F.

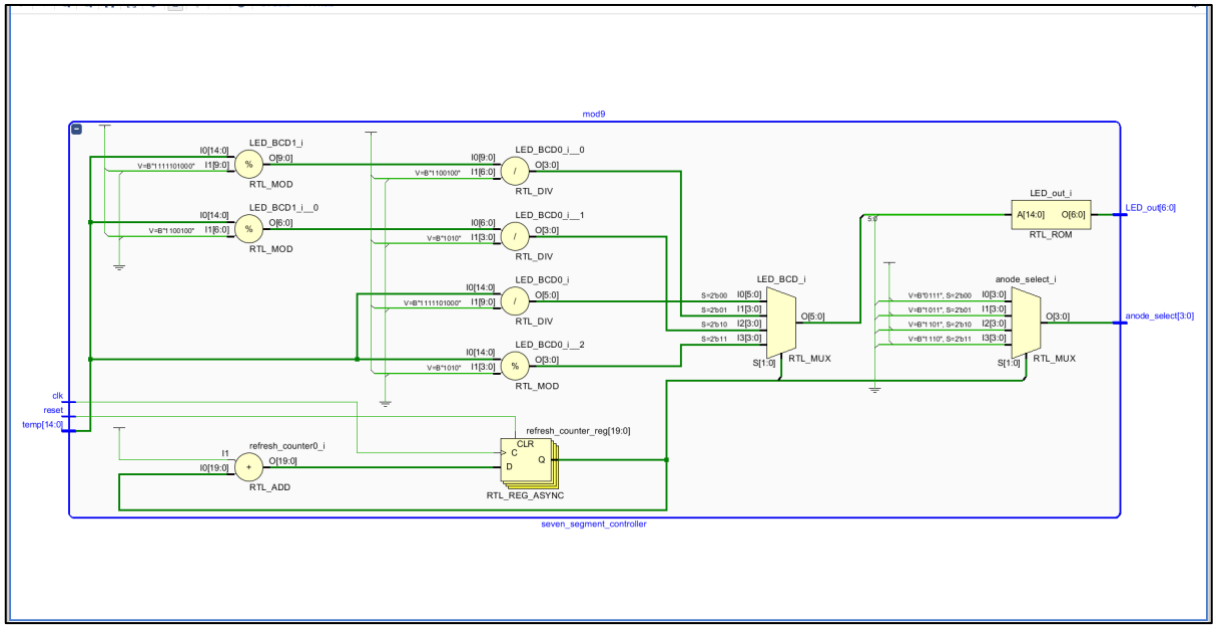


Figure 1.6 (Schematic of seven segment display module)

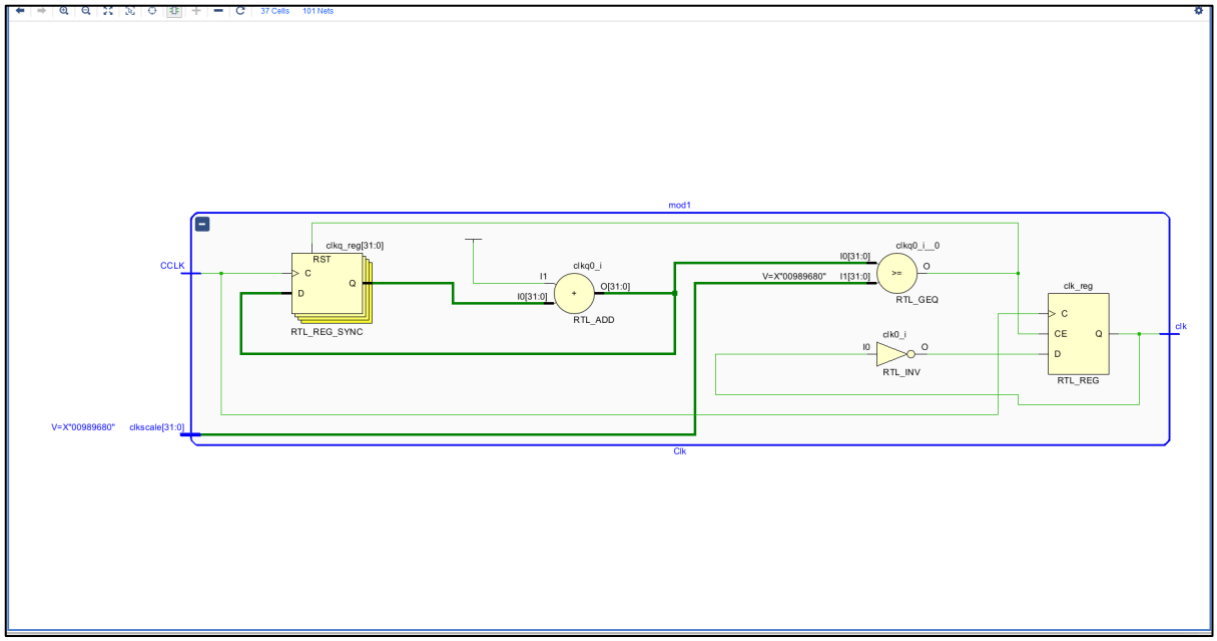


Figure 1.7 (Schematic of seven clock module)

This allowed us to generate a bitstream file and program the Basys board with our code. The board was tested using the RESET and ENGAGE switches. The shifting and counter were confirmed using our test cases and all the desired functions were confirmed such as resetting on RESET and once a full LFSR cycle had been completed. A video was taken to show our implementation working. As we wanted to demonstrate that all our functions worked, we sped up the clock speed so that the LFSR cycle wouldn't take too long.

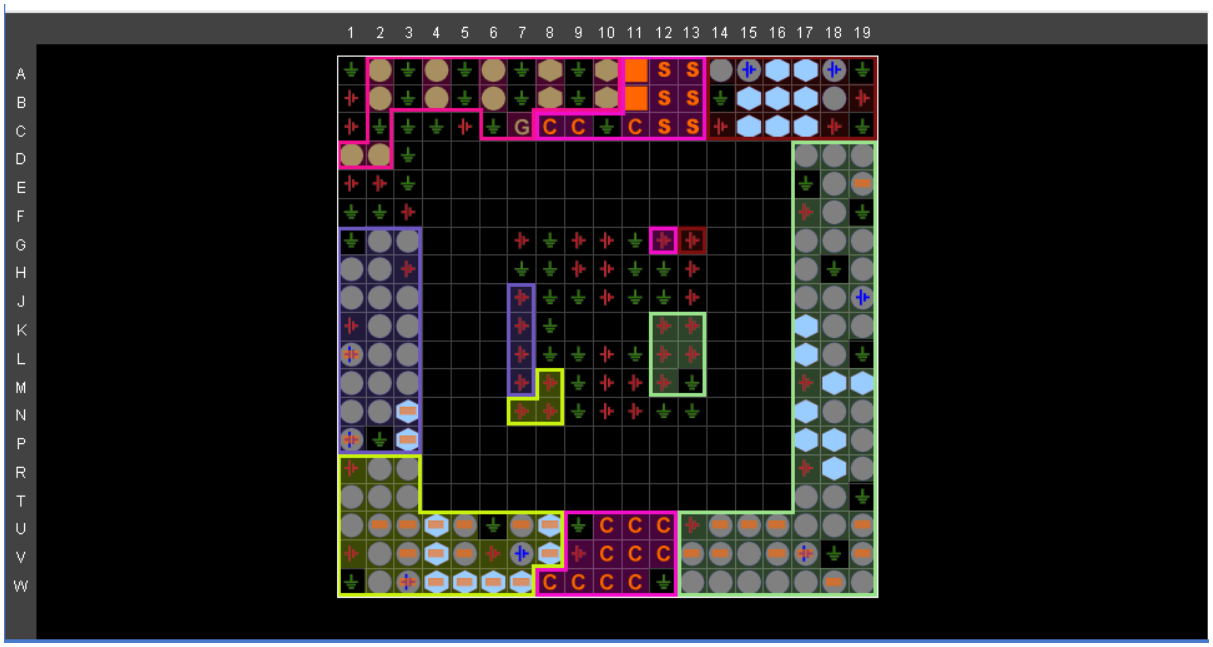
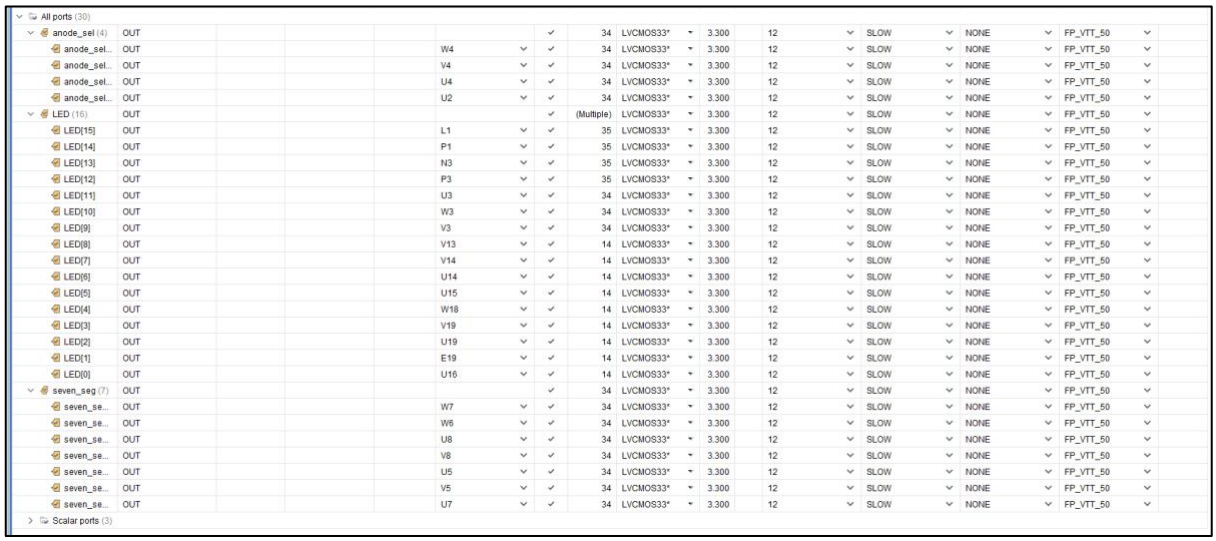


Figure 1.9 (IO view of pin assignments)

```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports CCLK]
        set_property IOSTANDARD LVCMOS33 [get_ports CCLK]


set_property PACKAGE_PIN U16 [get_ports {LED[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED[3]}]
set_property PACKAGE_PIN E19 [get_ports {LED[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED[4]}]
set_property PACKAGE_PIN U19 [get_ports {LED[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED[5]}]
set_property PACKAGE_PIN V19 [get_ports {LED[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED[6]}]
set_property PACKAGE_PIN W18 [get_ports {LED[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED[7]}]
set_property PACKAGE_PIN U15 [get_ports {LED[8]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED[8]}]
set_property PACKAGE_PIN U14 [get_ports {LED[9]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED[9]}]
set_property PACKAGE_PIN V14 [get_ports {LED[10]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED[10]}]
set_property PACKAGE_PIN V13 [get_ports {LED[11]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED[11]}]
set_property PACKAGE_PIN V3 [get_ports {LED[12]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED[12]}]
set_property PACKAGE_PIN W3 [get_ports {LED[13]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED[13]}]
set_property PACKAGE_PIN U3 [get_ports {LED[14]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED[14]}]
    set_property PACKAGE_PIN P3 [get_ports {LED[15]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED[15]}]
set_property PACKAGE_PIN N3 [get_ports {LED[16]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {LED[16]}]

set_property PACKAGE_PIN L1 [get_ports {screen}]
    set_property IOSTANDARD LVCMOS33 [get_ports {screen}]

##Buttons

set_property PACKAGE_PIN V17 [get_ports {ENGAGE}]
        set_property IOSTANDARD LVCMOS33 [get_ports {ENGAGE}]
set_property PACKAGE_PIN V16 [get_ports {RESET}]
        set_property IOSTANDARD LVCMOS33 [get_ports {RESET}]

##7 segment display - Cathode pins
set_property PACKAGE_PIN W7 [get_ports {led_out[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led_out[6]}]
set_property PACKAGE_PIN W6 [get_ports {led_out[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led_out[5]}]
set_property PACKAGE_PIN U8 [get_ports {led_out[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led_out[4]}]
set_property PACKAGE_PIN V8 [get_ports {led_out[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led_out[3]}]
set_property PACKAGE_PIN U5 [get_ports {led_out[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led_out[2]}]
set_property PACKAGE_PIN V5 [get_ports {led_out[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led_out[1]}]
set_property PACKAGE_PIN U7 [get_ports {led_out[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led_out[0]}]


#Anode Selects
set_property PACKAGE_PIN U2 [get_ports {anode_sel[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {anode_sel[0]}]
set_property PACKAGE_PIN U4 [get_ports {anode_sel[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {anode_sel[1]}]
set_property PACKAGE_PIN V4 [get_ports {anode_sel[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {anode_sel[2]}]
set_property PACKAGE_PIN W4 [get_ports {anode_sel[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {anode_sel[3]}]
```

Figure 1.8 (constraints file for implementation)



All ports (30)											
anode_sel (4)	OUT			✓	34	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
anode_sel...	OUT		W4	✓	34	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
anode_sel...	OUT		V4	✓	34	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
anode_sel...	OUT		U4	✓	34	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
anode_sel...	OUT		U2	✓	34	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
LED (16)	OUT			✓	(Multiple)	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
LED[15]	OUT		L1	✓	35	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
LED[14]	OUT		P1	✓	35	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
LED[13]	OUT		N3	✓	35	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
LED[12]	OUT		P3	✓	35	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
LED[11]	OUT		U3	✓	34	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
LED[10]	OUT		W3	✓	34	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
LED[9]	OUT		V3	✓	34	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
LED[8]	OUT		V13	✓	14	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
LED[7]	OUT		V14	✓	14	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
LED[6]	OUT		U14	✓	14	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
LED[5]	OUT		U15	✓	14	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
LED[4]	OUT		W18	✓	14	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
LED[3]	OUT		V19	✓	14	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
LED[2]	OUT		U19	✓	14	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
LED[1]	OUT		E19	✓	14	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
LED[0]	OUT		U16	✓	14	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
seven_seg (7)	OUT			✓	34	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
seven_se...	OUT		W7	✓	34	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
seven_se...	OUT		W6	✓	34	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
seven_se...	OUT		U8	✓	34	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
seven_se...	OUT		V8	✓	34	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
seven_se...	OUT		U5	✓	34	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
seven_se...	OUT		V5	✓	34	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
seven_se...	OUT		U7	✓	34	LVCMOS33*	3.300	12	SLOW	NONE	FP_VTT_50
Scalar ports (3)											

Figure 1.10 (list view of pin assignments)

DEMO

Steps to demonstrate bitstream on Basys3 board.

1. Generate the bitstream from the modules and upload to Basys board.
2. Apply the starting seed using RESET by engaging the v16 switch once.
3. Engage the v17 shift ENGAGE pin and watch as the bits shift left.
4. See that the pattern increments in 6 consecutive counts when it does, as the pattern stays in our LFSR for 6 clock cycles when found.
5. Check that the counter resets after 312, this is because the LFSR has reached a full cycle and 312 is the final count of each cycle of our LFSR. If the RESET is engaged while shifting, the LEDs will reset back to seed and count back to 0.

Note:

This number may change as the seed length used in our demo was the seed from Lab F. However the seed has been changed in the file so it will give the correct count if implemented again.

RESULTS/DISCUSSION

TEST CASE SCENARIOS

Our test wave forms can be found in the screenshots below. In our first test case we tested our counter and pattern ticks while using only one Moore machine on the 1st bit of our LSFR. This test did not use minimalization, and every state went back to state 0 regardless if the pattern could be continued. By running this test case, we were able to verify that our top module tick and tick counter were both working and that we could continue to implement the rest of our system.

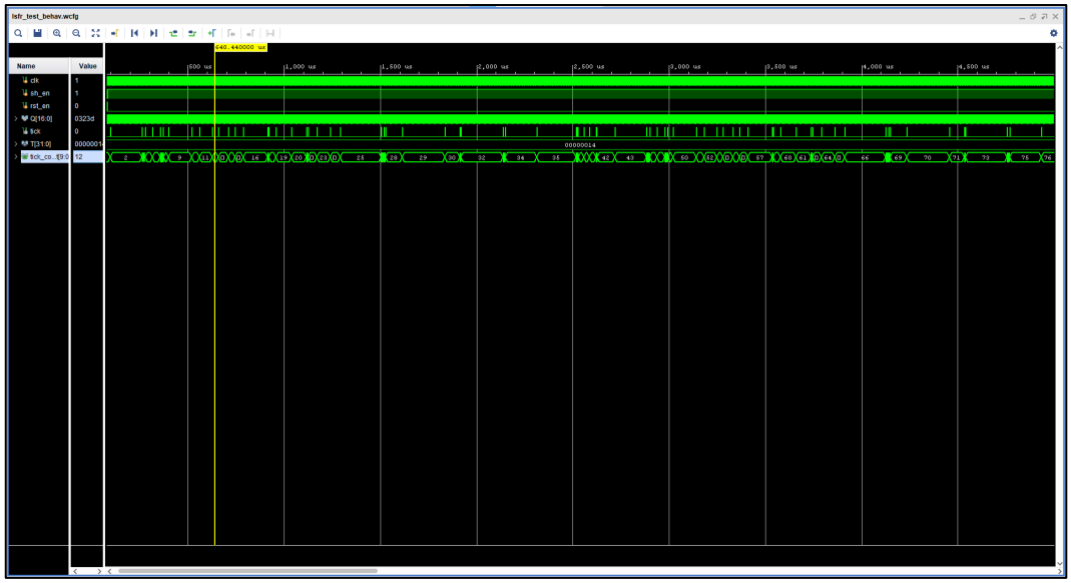


Figure 2.1 (Implementation Wave forms from one Moore FSM)

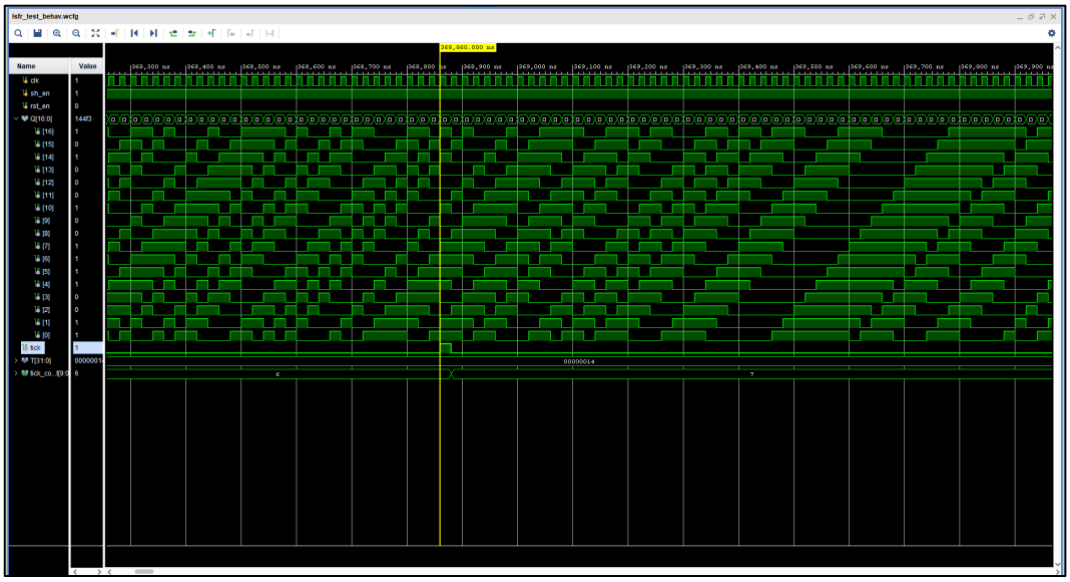


Figure 2.2 (Zoomed in implementation wave that shows the tick going high once and counter incrementing)

Our second and final test case was to check that the Top Module was working when we were implementing six Moore Machines. From here we tested initially, however we were unable to take six inputs to the same screen/tick wire in top module. So Logic had to be implemented to OR each tick, s1, s2, s3, s4, s5 and s6 from the Moore machines. This allowed us to display all the ticks and increment the counter for all.

As we can see in the zoomed in view. The counter increments six times when the tick is high. As the tick is staying in high state for six clock cycles. We are also able to see in our new wave forms that the counter is resetting when the LFSR cycle register goes high.

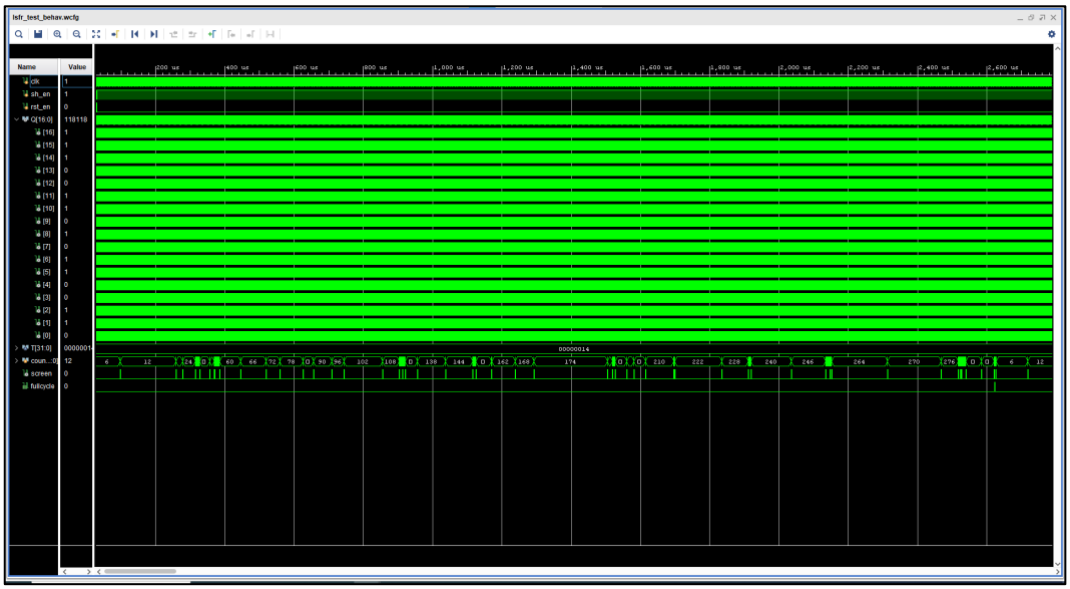


Figure 2.3 (Implementation Wave forms from Six Moore Machine system)

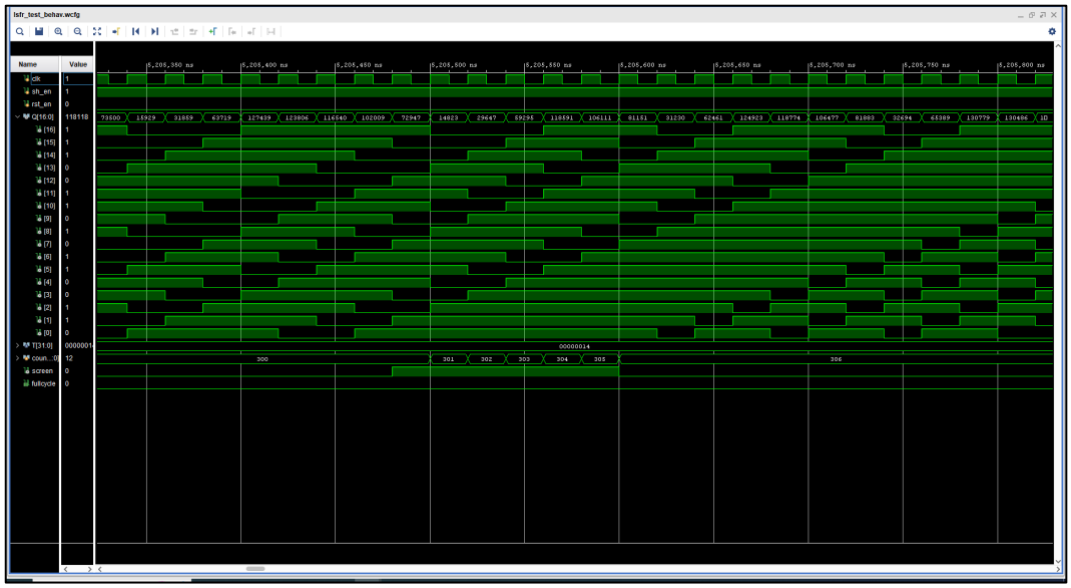


Figure 2.4 (Zoom of Implementation Wave forms from Six Moore Machine system showing six increments to previous one)

UTILIZATION REPORT/FLIP FLOP COUNT

The utilization report obtained from our implementation can be seen below. It gives the user an idea of how much of the board they are targeting and how much more operations the board could handle. As we can see we do not use very much of our Basys board and the board would be capable of much more, as would be expected of a board at this price.

In part 1 of the utilization report, we are provided with a number for the amount of flip flops that are in use in our system. In our system we use 158 of 41600 registers available on the Basys board as flip flops, coming to 0.38% of what we would be able to use if we were to want to. This number really gives us a good idea of how much can done on a FPGA board and the huge possibility of complicated and large operations and systems that could be run.

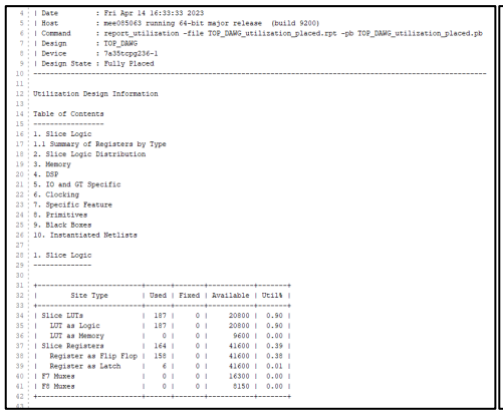


Figure 2.5 (Utilization report part1)

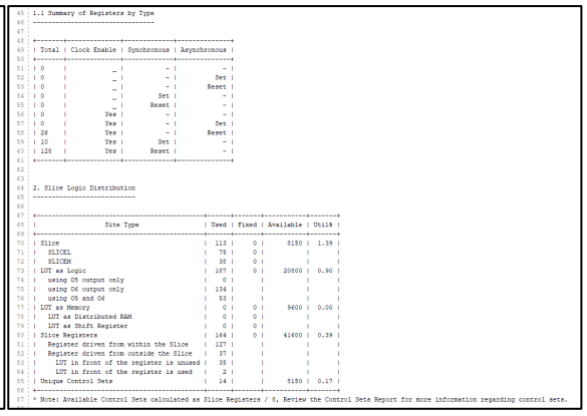
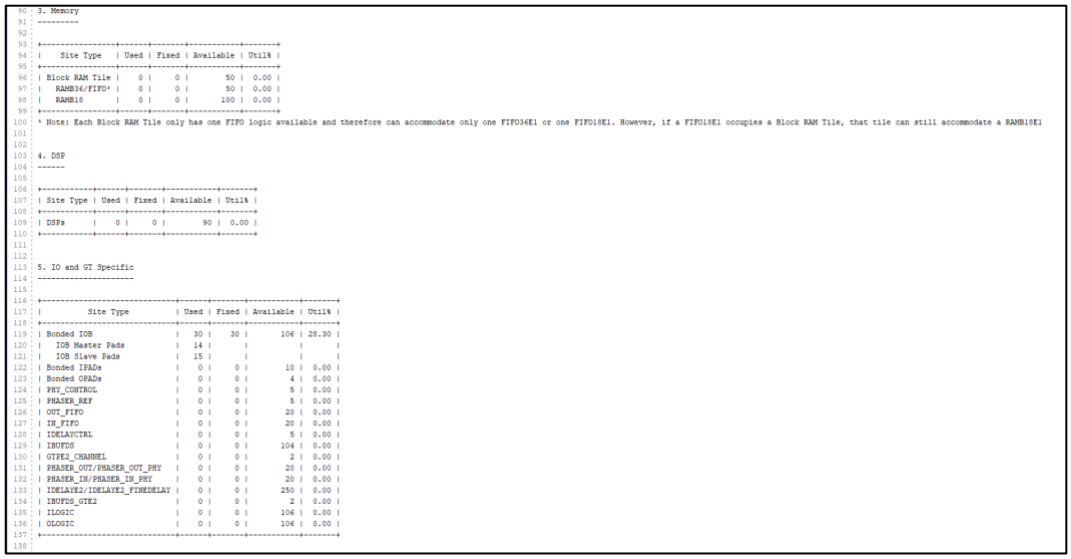
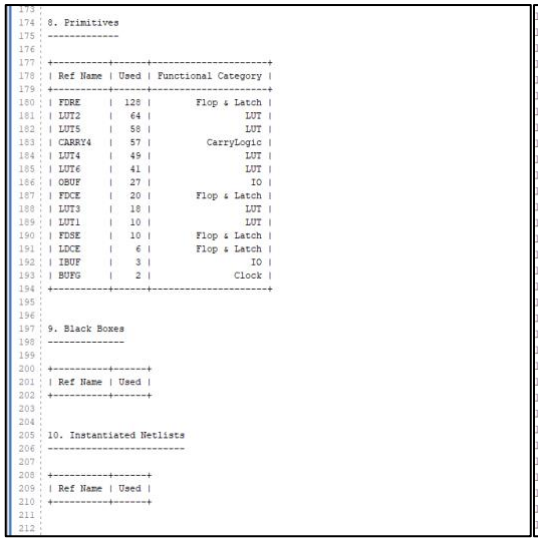


Figure 2.6 (Utilization report part2)



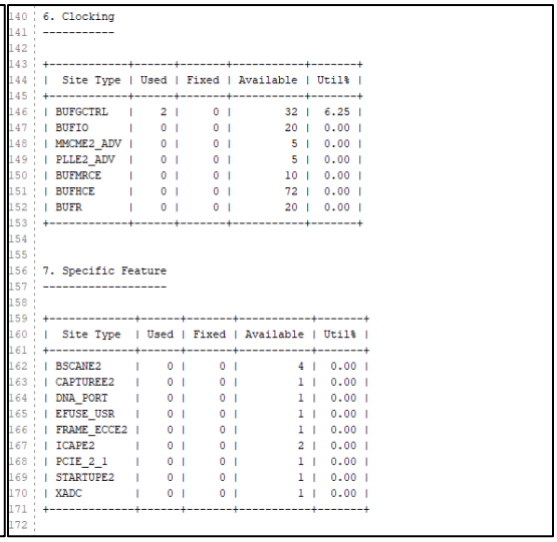
```
3. Memory
---------

+----------------+------+-------+-----------+-------+
|    Site Type   | Used | Fixed | Available | Util% |
+----------------+------+-------+-----------+-------+
| Block RAM Tile |    0 |     0 |        50 |  0.00 |
|   RAMB36/FIFO* |    0 |     0 |        50 |  0.00 |
|   RAMB18       |    0 |     0 |       100 |  0.00 |
+----------------+------+-------+-----------+-------+
* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies a Block RAM Tile, that tile can still accommodate a RAMB18E1


4. DSP
------

+-----------+------+-------+-----------+-------+
| Site Type | Used | Fixed | Available | Util% |
+-----------+------+-------+-----------+-------+
| DSPs      |    0 |     0 |        90 |  0.00 |
+-----------+------+-------+-----------+-------+


5. IO and GT Specific
---------------------

+----------------------------+------+-------+-----------+--------+
|          Site Type         | Used | Fixed | Available | Util%  |
+----------------------------+------+-------+-----------+--------+
| Bonded IOB                 |   30 |    30 |       106 |  28.30 |
|   IOB Master Pads          |   14 |       |           |        |
|   IOB Slave Pads           |   15 |       |           |        |
| Bonded IPADs               |    0 |     0 |        10 |   0.00 |
| Bonded OPADs               |    0 |     0 |         4 |   0.00 |
| PHY_CONTROL                |    0 |     0 |         5 |   0.00 |
| PHASER_REF                 |    0 |     0 |         5 |   0.00 |
| OUT_FIFO                   |    0 |     0 |        20 |   0.00 |
| IN_FIFO                    |    0 |     0 |        20 |   0.00 |
| IDELAYCTRL                 |    0 |     0 |         5 |   0.00 |
| IBUFDS                     |    0 |     0 |       104 |   0.00 |
| GTPE2_CHANNEL              |    0 |     0 |         2 |   0.00 |
| PHASER_OUT/PHASER_OUT_PHY  |    0 |     0 |        20 |   0.00 |
| PHASER_IN/PHASER_IN_PHY    |    0 |     0 |        20 |   0.00 |
| IDELAYE2/IDELAYE2_FINEDELAY|    0 |     0 |       250 |   0.00 |
| IBUFDS_GTE2                |    0 |     0 |         2 |   0.00 |
| ILOGIC                     |    0 |     0 |       106 |   0.00 |
| OLOGIC                     |    0 |     0 |       106 |   0.00 |
+----------------------------+------+-------+-----------+--------+
```

Figure 2.7 (Utilization report part3)



```
8. Primitives
-------------

+----------+------+---------------------+
| Ref Name | Used | Functional Category |
+----------+------+---------------------+
| FDRE     |  128 |        Flop & Latch |
| LUT2     |   64 |                 LUT |
| LUT5     |   58 |                 LUT |
| CARRY4   |   57 |          CarryLogic |
| LUT4     |   49 |                 LUT |
| LUT6     |   41 |                 LUT |
| OBUF     |   27 |                  IO |
| FDCE     |   20 |        Flop & Latch |
| LUT3     |   18 |                 LUT |
| LUT1     |   10 |                 LUT |
| FDSE     |   10 |        Flop & Latch |
| LDCE     |    6 |        Flop & Latch |
| IBUF     |    3 |                  IO |
| BUFG     |    2 |               Clock |
+----------+------+---------------------+


9. Black Boxes
--------------

+----------+------+
| Ref Name | Used |
+----------+------+


10. Instantiated Netlists
-------------------------

+----------+------+
| Ref Name | Used |
+----------+------+
```

Figure 2.8 (Utilization report part5)

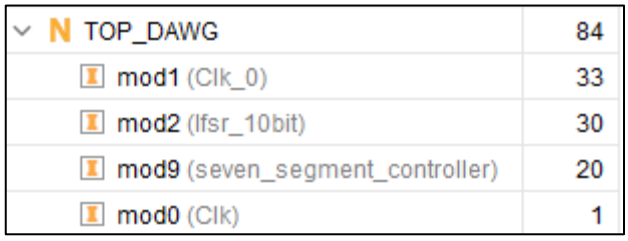


```
6. Clocking
-----------

+------------+------+-------+-----------+-------+
|  Site Type | Used | Fixed | Available | Util% |
+------------+------+-------+-----------+-------+
| BUFGCTRL   |    2 |     0 |        32 |  6.25 |
| BUFIO      |    0 |     0 |        20 |  0.00 |
| MMCME2_ADV |    0 |     0 |         5 |  0.00 |
| PLLE2_ADV  |    0 |     0 |         5 |  0.00 |
| BUFMRCE    |    0 |     0 |        10 |  0.00 |
| BUFHCE     |    0 |     0 |        72 |  0.00 |
| BUFR       |    0 |     0 |        20 |  0.00 |
+------------+------+-------+-----------+-------+


7. Specific Feature
-------------------

+-------------+------+-------+-----------+-------+
|  Site Type  | Used | Fixed | Available | Util% |
+-------------+------+-------+-----------+-------+
| BSCANE2     |    0 |     0 |         4 |  0.00 |
| CAPTUREE2   |    0 |     0 |         1 |  0.00 |
| DNA_PORT    |    0 |     0 |         1 |  0.00 |
| EFUSE_USR   |    0 |     0 |         1 |  0.00 |
| FRAME_ECCE2 |    0 |     0 |         1 |  0.00 |
| ICAPE2      |    0 |     0 |         2 |  0.00 |
| PCIE_2_1    |    0 |     0 |         1 |  0.00 |
| STARTUPE2   |    0 |     0 |         1 |  0.00 |
| XADC        |    0 |     0 |         1 |  0.00 |
+-------------+------+-------+-----------+-------+
```

Figure 2.9 (Utilization report part5)

TIMING REPORT

A Timing Report was also run, and as expected, the higher our module in our hierarchy or the more a module was called, the worse the timings were according to Vivado.



TOP_DAWG	84
mod1 (Clk_0)	33
mod2 (lfsr_10bit)	30
mod9 (seven_segment_controller)	20
mod0 (Clk)	1

Figure 2.10 (Timing report given)

BASYS BOARD IMPLEMENTATION

The results of our BASYS board implementation can be seen below. However the system is hard to demonstrate using images, so a video is included in the file submission on blackboard. Which will demonstrate the system working at a very fast clock speed. Looking at the photos below, we are able to see that the seven segment is displaying a number of multiple six, showing us that the BASYS is indeed counting correctly. We can also see the bits shown in different sequences on our LED while the ENGAGE switch is enabled. The last case of RESET enabled also shows a correct result of our count returning to 0.

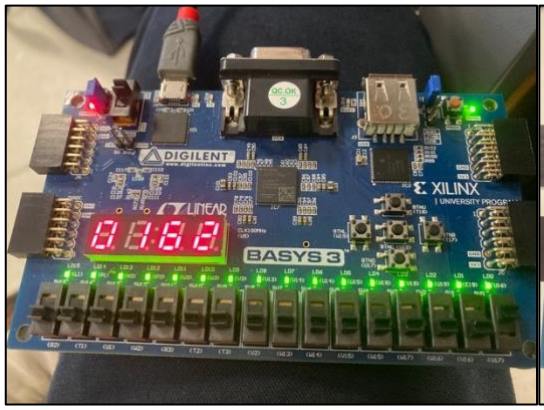


Figure 2.11 (System working at fast clock)

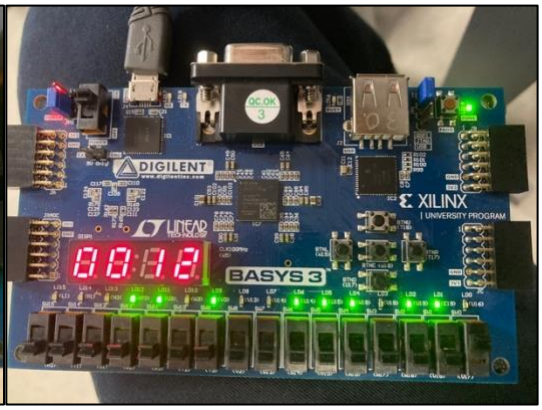


Figure 2.11 (System working normally)

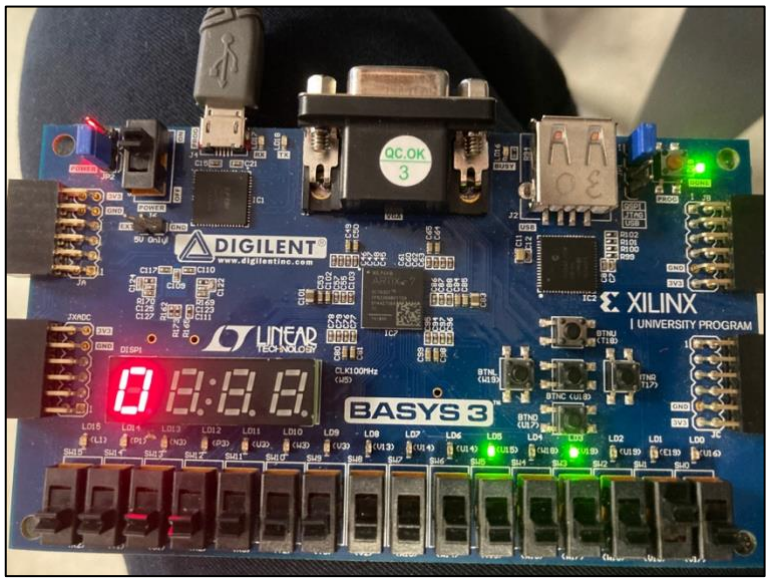


Figure 2.8 (Reset engaged, count rest to 0 and binary number retuned to seed)

CONCLUSION

In this lab we were able to create a Moore Machine FSM to sequentially count the occurrence of a pattern appearing. By incorporating the knowledge acquired from our past Lab and assignments. We were able to create new modules such as the FSM, reuse old modules such as the LFSR, seven seg and clock modules and implement test benches and constraints files to accurately program our FPGA Basys board with our system. The system counted that we found the pattern 10011110011 312 times before the LFSR cycle was done. This was done with six bits, so if we were to have only implemented the system using the 1st bit, we would have expected 51 occurrences of our pattern.

This Assignment showed how a Basys board could be used to implement different operations and showed us how a Finite State Machine can conduct operation using sequential states, and how states can overlap and not always have to go back to initial state.

REFERENCES

Code from Lab F both created and provided was modified and utilised in Assignment 2.

1. The seven segment display was used and altered to display a 15 bit number. The altered coded can be found below.

```
    reg [3:0] LED_BCD;
    reg [19:0] refresh_counter; // 20-bit for creating 10.5ms refresh period or 380Hz refresh rate
             // the first 2 MSB bits for creating 4 LED-activating signals with 2.6ms digit period

    wire [1:0] LED_activating_counter;
                 // count     0    ->  1  ->  2  ->  3
              // activates    LED1    LED2   LED3   LED4
             // and repeat

    always @(posedge clk or posedge reset)
    begin
        if(reset==1)
            refresh_counter <= 0;
        else
            refresh_counter <= refresh_counter + 1;
    end
    assign LED_activating_counter = refresh_counter[19:18];
    // anode activating signals for 4 LEDs, digit period of 2.6ms
    // decoder to generate anode signals

    always @(*)
    begin
        case(LED_activating_counter)
        2'b00: begin
            anode_select = 4'b0111;
            // activate LED1 and Deactivate LED2, LED3, LED4
            LED_BCD = temp/1000;
             // the first digit of the 8-bit temperature value
            end
        2'b01: begin
            anode_select = 4'b1011;
            // activate LED2 and Deactivate LED1, LED3, LED4
            LED_BCD = (temp%1000)/100;
            // the second digit of the 8-bit temperature value
            end
        2'b10: begin
            anode_select = 4'b1101;
            // activate LED3 and Deactivate LED2, LED1, LED4
            LED_BCD = ((temp%1000)%100)/10;
            // the last digit of the 8-bit temperature value
            end
        2'b11: begin
            anode_select = 4'b1110;
            // activate LED4 and Deactivate LED2, LED3, LED1
            LED_BCD = ((temp %1000)%100)%10;
            // F symbol to indicate Fahrenheit
            end
        endcase
    end
    // Cathode patterns of the 7-segment LED display
    always @(*)
    begin
        case(LED_BCD)
        4'd0: LED_out = 7'b0000001; // "0"
        4'd1: LED_out = 7'b1001111; // "1"
        4'd2: LED_out = 7'b0010010; // "2"
        4'd3: LED_out = 7'b0000110; // "3"
        4'd4: LED_out = 7'b1001100; // "4"
        4'd5: LED_out = 7'b0100100; // "5"
        4'd6: LED_out = 7'b0100000; // "6"
        4'd7: LED_out = 7'b0001111; // "7"
        4'd8: LED_out = 7'b0000000; // "8"
        4'd9: LED_out = 7'b0000100; // "9"
        4'd10: LED_out = 7'b0111000; // "F"
        default: LED_out = 7'b0000001; // "0"
        endcase
    end
endmodule
```

2. The LFSR was reused from lab F and the max_tick_reg was negated for the assignment. The code is below.

```
module lfsr_10bit
     ( input wire  clk, sh_en, rst_en,
     output wire  [16:0] Q_out,
     output reg fullcycle = 1'b0
     );
     localparam seed = 17'h3DDF;
     reg [16:0] Q_state = 17'h0;
     wire [16:0] Q_ns;
     reg [31:0] count = 32'b0;
     wire [2:0] garbage;

     MSB_counter mod3(.clk(clk), .MSB(Q_ns[15]));

     //Clk mod1(.CCLK(clk), .clkscale(17'd131071), .clk(tick_high), .clkg(tick_low));
          //next state logic
     assign Q_fb = Q_state[16] ^ Q_state[13];
     assign Q_ns = {Q_state[15:0], Q_fb};
     //output logic



     always @ (posedge clk) begin

        if (rst_en)begin
            Q_state <= seed;
            count <= 32'b0;
            end
        else if (sh_en) begin
            count <= count + 1;
            Q_state <= Q_ns;
            if (count >= 131071)begin
                count <= 32'b0;
                fullcycle <= 1'b1;
                end
            else begin
                fullcycle <= 1'b0;
                end
            end
        else begin
            Q_state <= Q_state;
            end
        end

         assign Q_out = Q_state;


endmodule
```

3. The clock file can be also seen below, this was not changed from Lab F.

```
module Clk(
    input CCLK,
    input [31:0] clkscale,
    output reg clk
    );

    reg [31:0] clkg = 0;


    always@(posedge CCLK)
        begin
            clkg = clkg + 1;

            if (clkg >= clkscale)
                begin
                    clk =~clk;
                    clkg = 0;
                end
            end

endmodule
```

4. The test bench was slightly modified from Lab F to test our top bench.

```
`timescale 1 ns/10 ps

module lsfr_test();
   // declaring our wires for input and ouput
   localparam T = 20;
   reg clk, sh_en, rst_en;
   wire [16:0] Q;
   wire tick;
   wire [9:0] count;
   //wire [16:0] tick_high, tick_low;

   // initialising our test variables with variables for display
   TOP_DAWG uut(.CCLK(clk), .RESET(rst_en), .ENGAGE(sh_en), .LED(Q));

   always
        begin
            clk = 1'b1;
            #(T/2);
            clk = 1'b0;
            #(T/2);

        end

    initial
        begin
            rst_en = 1'b1;
            #(T*10) ;
            rst_en = 1'b0;
        end


    initial
        begin
            sh_en = 1'b0;
            #(T*10);
            sh_en = 1'b1;
            #(T);

        end


endmodule
```