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

3C7 LAB F

Clovis Hanbury-Tenison

20331969

INSERT LAB

Student Name:	Clovis Hanbury-Tenison
Student ID Number:	20331969
Assessment Title:	Lab F
Lecturer (s):	SHREEJITH SHANKER
Date Submitted	31/03/2023

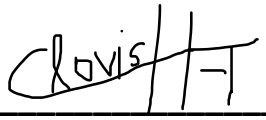
I hereby declare that this assessment submission is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

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: 31/03/2023

Introduction

In Lab F we learned the basics of LFSR's starting with D flipflop and working are way up to LFSR's and implementation of them on our breadboard. We also learnt how to use a clock to trigger operations on different edges synchronous and asynchronously. This allowed us to create the basic always loops which allowed our LFSR to work.

Experiment/Observations

Part A: Clock and Reset

In part A we were given a D-flip-flop module and were instructed to create a test bench with a clock, reset and , d point. The test bench we created can be seen at the bottom of Part A. We use an always block to run a clock at 20ns period.

We then observed how and where we triggered our inputs would effect where the outputs would trigger. We configured our dd_ff_reset module to trigger on positive and negative edges both asynchronously and synchronously.

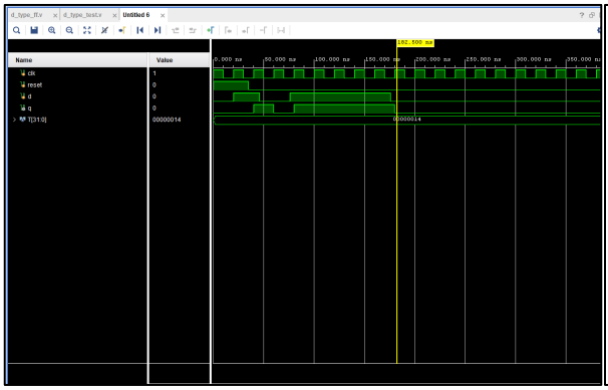


Figure 1.1 (Asynchronous positive edge)

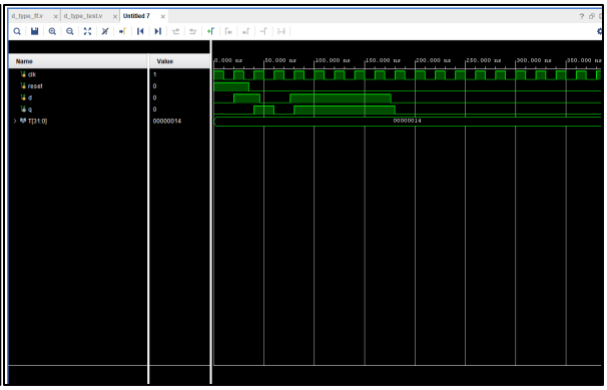


Figure 1.2 (Synchronous positive edge)

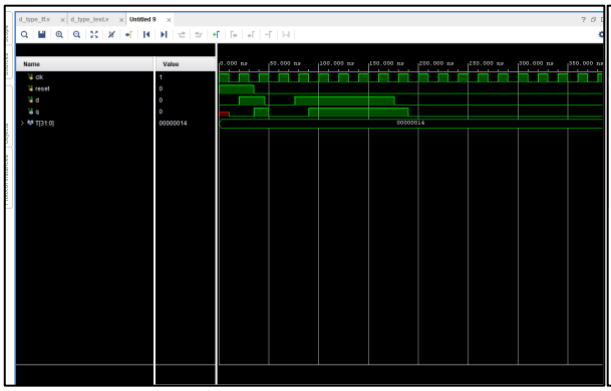


Figure 2.3 (Asynchronous negative edge)

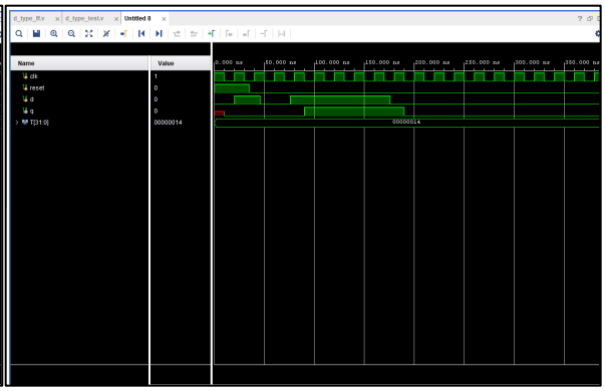


Figure 1.4 (Synchronous negative edge)

The Resulting wave forms show that the asynchronous and synchronous only effected the negative edge trigger in our test case. The synchronous test was unable to trigger the output while reset dropped off and d still stayed high whereas the asynchronous was able to.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// The 'timescale directive specifies that
// the sim~rlation tinre unit is I trs and // the simulator tinrestep is 10 ps
module d_type_test();
// declaration
    localparam T=20;
    reg clk, reset;
    reg d;
    wire q;
// clock

    d_ff_reset uut (.clk(clk), .reset(reset), .d(d), .q(q));


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

end

initial
begin
    reset = 1'b1;
    #(T/2) ;
    reset = 1'b0;
end

initial
begin
    d = 1'b0;
    #(T) ;
    d = 1'b1;
    #(T + T/2) ;
    d = 1'b0;
    #(T) ;
    d = 1'b0;
    #(5*T) ;
end

endmodule
```

(Code 1.1 Test Bench for DFF)

Part B: Practical use of DFF

In Part B we wanted to test clock use and how we could incorporate this into a module along with DFF's and create a counter that relied on button presses. We create the Top_File seen below, which called d_ff_reset 8 times for each of our 8 bit number. We then used the seven segments module and debouncer to display on the Basys board. The .xdc file was provided in the lab and implementation results can be seen below.

```
module Top_File(
    input clk,
    input reset,
    output [3:0] anode_sel,
    output [6:0] led_out,
    input [4:0] button_input
    );

    wire [7:0] Q;
    reg [7:0] Q_next;
    wire [4:0] buttons;

    d_ff_reset DFF0(.clk(clk), .reset(reset), .d(Q_next[0]), .q(Q[0]));
    d_ff_reset DFF1(.clk(clk), .reset(reset), .d(Q_next[1]), .q(Q[1]));
    d_ff_reset DFF2(.clk(clk), .reset(reset), .d(Q_next[2]), .q(Q[2]));
    d_ff_reset DFF3(.clk(clk), .reset(reset), .d(Q_next[3]), .q(Q[3]));
    d_ff_reset DFF4(.clk(clk), .reset(reset), .d(Q_next[4]), .q(Q[4]));
    d_ff_reset DFF5(.clk(clk), .reset(reset), .d(Q_next[5]), .q(Q[5]));
    d_ff_reset DFF6(.clk(clk), .reset(reset), .d(Q_next[6]), .q(Q[6]));
    d_ff_reset DFF7(.clk(clk), .reset(reset), .d(Q_next[7]), .q(Q[7]));


    seven_segment_controller mod1(.clk(clk), .reset(reset), .temp(Q), .anode_select(anode_sel), .LED_out(led_out));

    debouncer mod2 (.clk(clk), .reset(reset), .button_in(button_input), .button_out(buttons));

    always @(posedge clk) begin
        if (buttons[0] | buttons[3])
            Q_next = Q + 1'b1;
        else if (buttons[2] | buttons[1])
            Q_next = Q - 1'b1;
        else if (buttons[4])
            Q_next = 5'b10110;
        else
            Q_next = Q;
        end

endmodule
```

(Code 2.1 Top File for our temperature Clock counter)

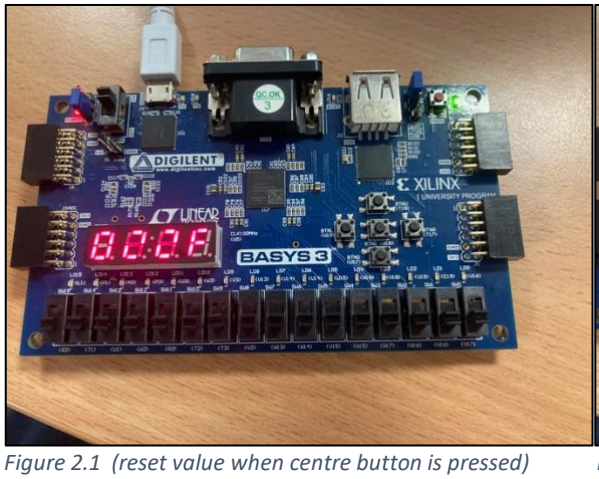


Figure 2.1 (reset value when centre button is pressed)

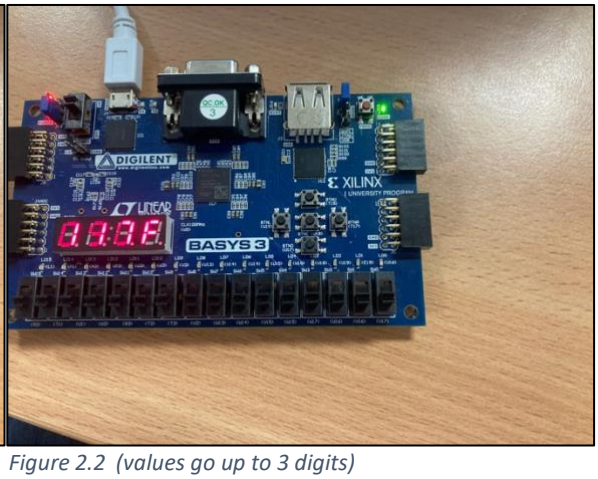


Figure 2.2 (values go up to 3 digits)

Part C: LFSR Implementation

For part C each student was given their individual tap length and seed value seen below. The seed value was a XNOR output of the last 3 digits of my student number 969 and my board number 30. This gave us a result of decimal 40 or binary 101000.

20331969	HANBURY-TENIS	CLOVIS	xor	17	Bitwise-xnor of your board-number and last 3 digits of your student id

The tap length was used to find our tap values in the Xilinx data sheet and were found to be Bit 14 and bit 17, using the logic feedback we were provided, these two bits were fed into a XOR gate and the output feedback was fed into bit 1. This provided our logic for the bit shifter. The module we created lfsr_10bit used the clock to shift the bits every cycle and can be seen below.

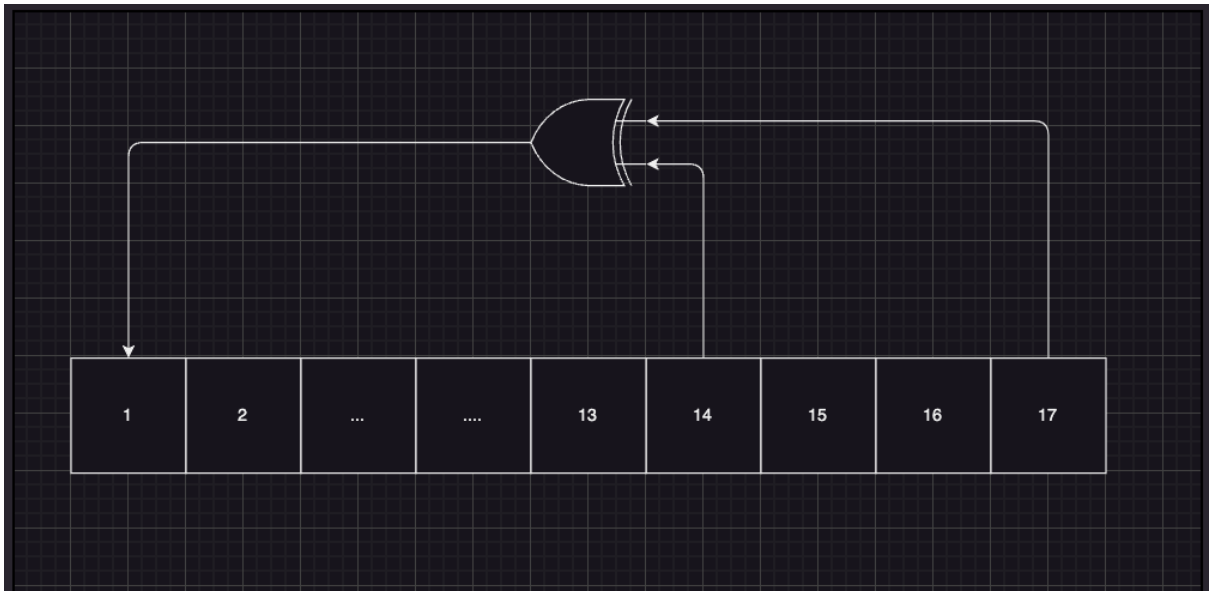


Fig 3.1 (Diagram of our N0 of bits and the logic to create our feedback)

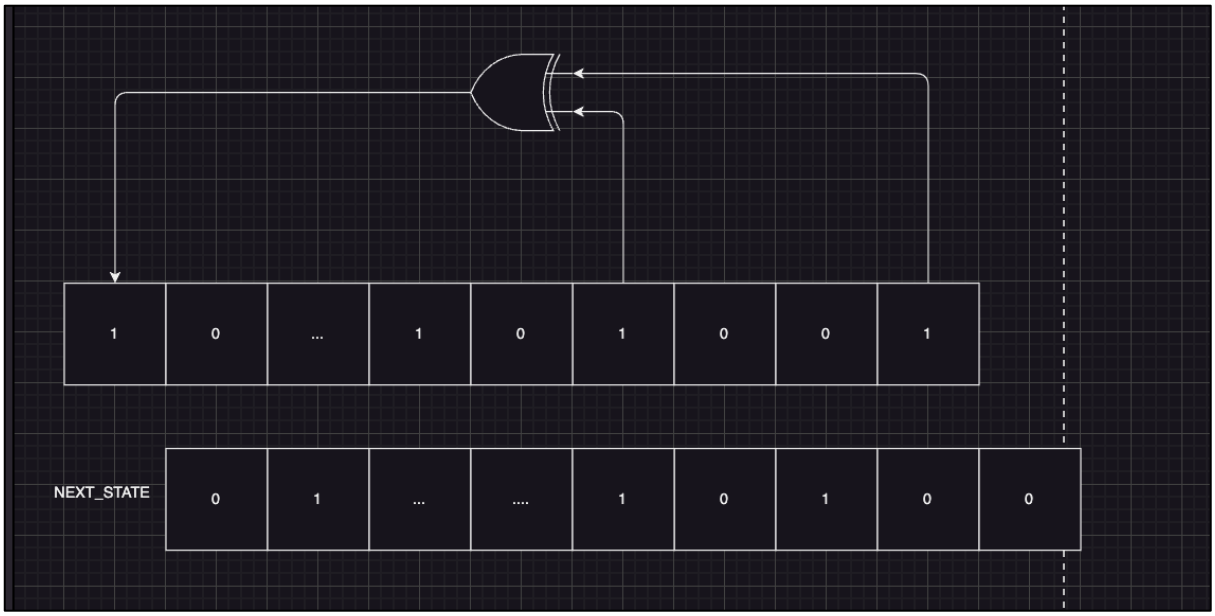


Fig 3.2 (Representation of our feedback loop working while shifting bits)

```
module lfsr_10bit
     ( input wire  clk, sh_en, rst_en,
     output wire  [16:0] Q_out,
     output reg  max_tick_reg=1'b0
     );
     localparam seed = 17'h28;
     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;
            max_tick_reg <= 1'b0;
            count <= 32'b0;
            end
        else if (sh_en) begin
            count <= count + 1;
            Q_state <= Q_ns;
            if (count == 131071)begin
                max_tick_reg <= 1'b1;
                count <= 32'b0;
                end
            else begin
                max_tick_reg <= 1'b0;
                end
            end
        else begin
            Q_state <= Q_state;
            end
        end

         assign Q_out = Q_state;


endmodule
```

(code 3.1 LFSR module code)

```
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
```

(Code 3.2 Clk module for LFSR)

```
module TOP_DAWG(
    input CCLK,
    input RESET, ENGAGE,
    output [16:3] LED,
    output screen

    );

    wire cl;
    Clk mod1(.CCLK(CCLK), .clkscale(50000000), .clk(cl));
    lfsr_10bit mod2(.clk(cl), .sh_en(ENGAGE), .rst_en(RESET), .Q_out(LED), .max_tick_reg(screen));


endmodule
```

(Code 3.3 Top module for LFSR implementation)

The Clk module was used to create a clk for the other submodules and set the clock to have a rising edge every 1s. The module was called in our TOP_DAWG module and the clock output was fed into the lfsr module which shifted our bits using a rising edge trigger and an always loop.

We used a test bench seen below to test our module and design. The test bench simulated a clock with 20ns cycles and set our reset high for 10 cycles and then set our engage to high after. This resulted in the wave form seen below. We let the test run for 1s to better evaluate if our design was working properly. We can see that our max_tick_reg is being reached every so often and the count is resetting as this is around 2*17 -1.

```
`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_out;
   wire tick;
   //wire [16:0] tick_high, tick_low;

   // initialising our test variables with variables for display
   lfsr_10bit uut(.clk(clk), .sh_en(sh_en), .max_tick_reg(tick), .rst_en(rst_en), .Q_out(Q_out));

   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
```

(Code 3.4 Testbench for LFSR implementation)

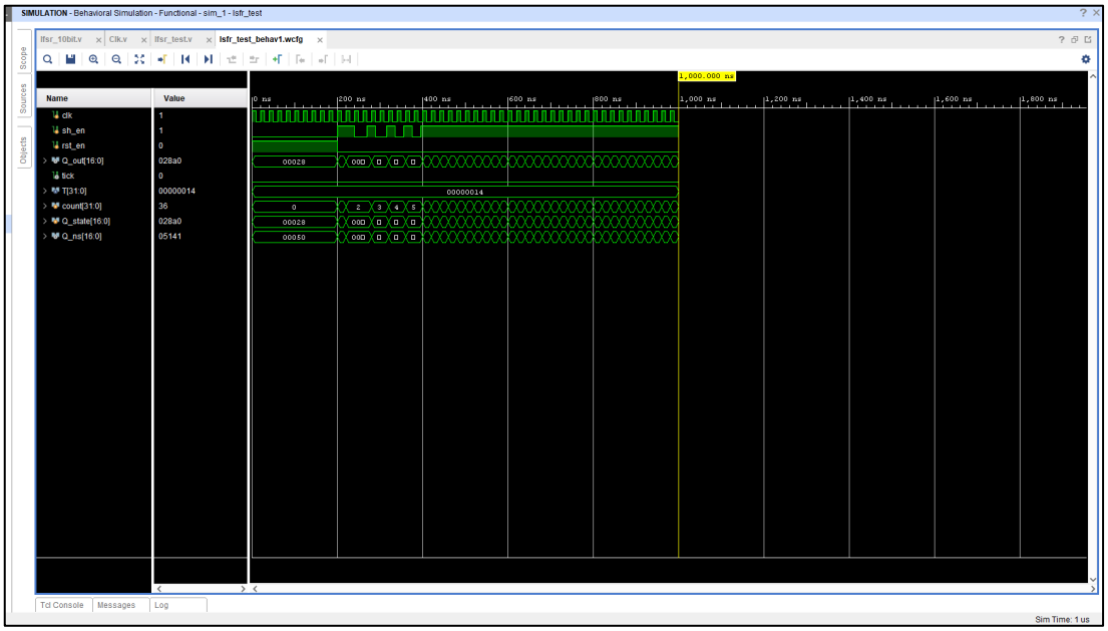


Figure 3.3 ((Waveform showing outputs of our top modules running for test case time)

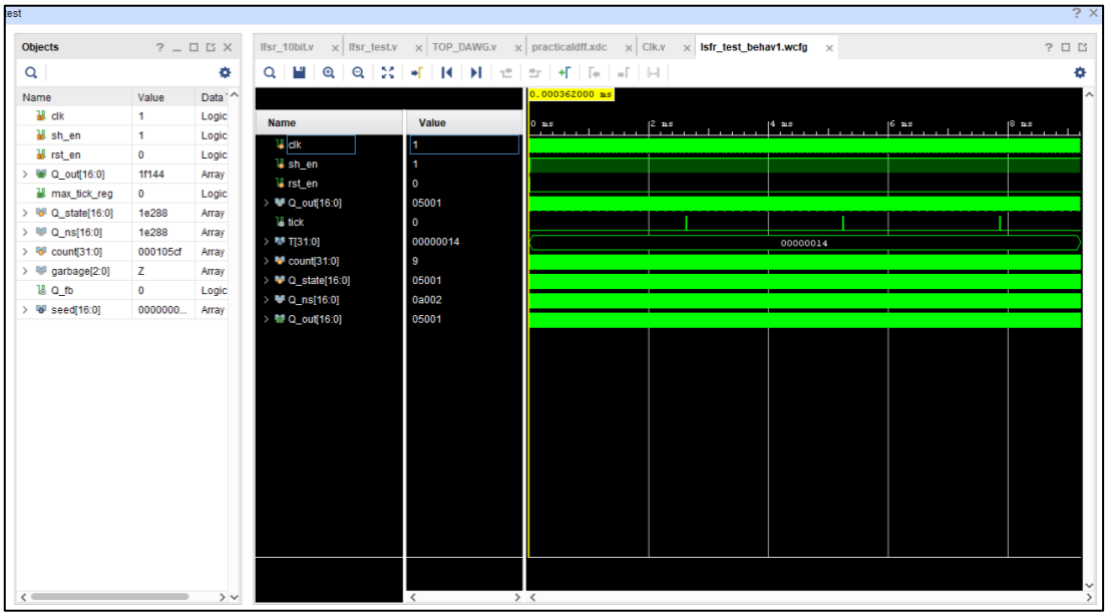


Figure 3.4 (Waveform showing outputs of our top modules running for 9 ms)

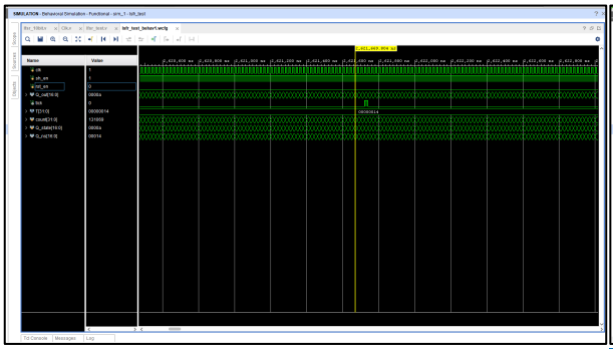


Figure 3.5 (zoomed in wave forms))

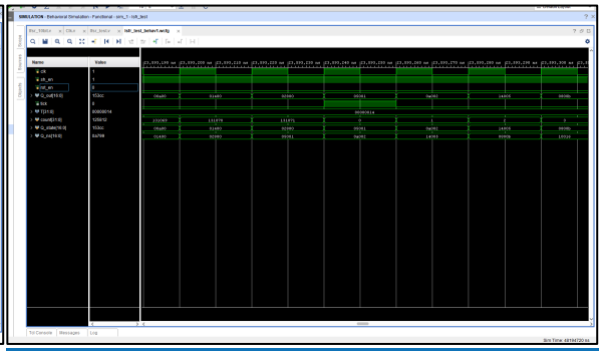


Figure 3.6 (zoomed in again))

Part C: Advanced

```
module MSB_counter(
    input wire clk,
    input wire MSB
    );

    reg [16:0] countzero = 17'b0;
    reg [16:0] countone = 17'b0;

    wire [16:0] garbage1, garbage2;

    assign garbage1 = countzero;
    assign garbage2 = countone;

     always @ (posedge clk) begin
        if (MSB == 1'b0)
            countzero = countzero + 1'b1;
        else if (MSB == 1'b1)
            countone = countone + 1'b1;
        else begin
            countzero = countzero;
            countone = countone;
            end
        end

endmodule
```

(Code 3.5 MSB counter for LFSR implementation)

For the advanced section we implemented a MSB counter, to display the amount of times our MSB bit 16 was high and how many times it was low. We ran the same test case and were able to see the counters working digitally and in wave form. In a 1000ns run, this resulted in our MSB being high for 8 cycles while it was low for 42cycles. This is the expected amount of highs and lows and proved our counters were working correctly.

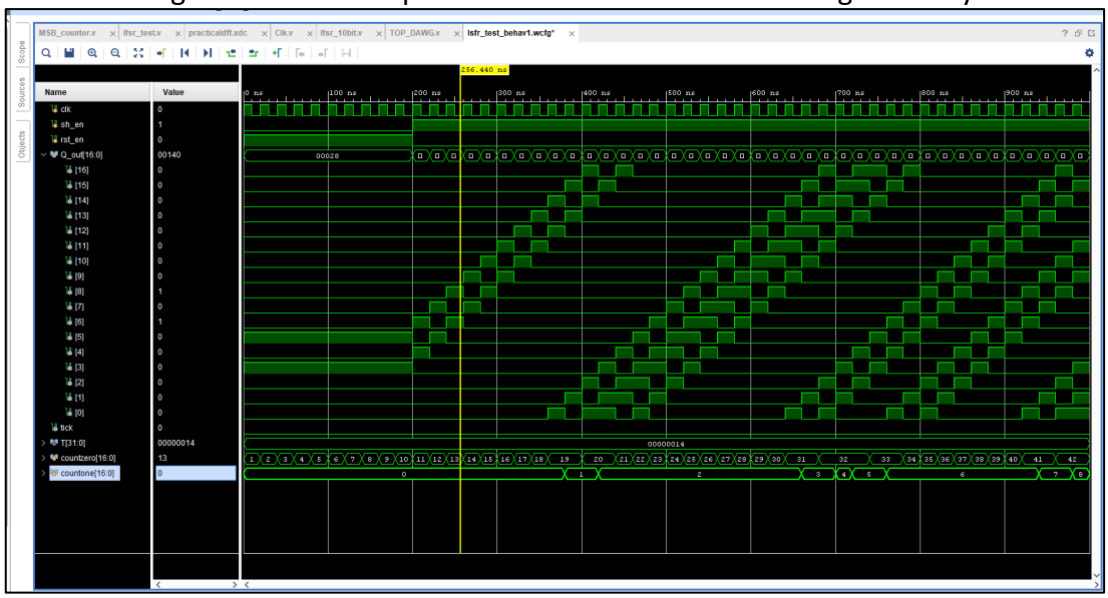


Figure 3.7 (Waveform showing counting of 0's and 1's in our most significant bit in decimal)

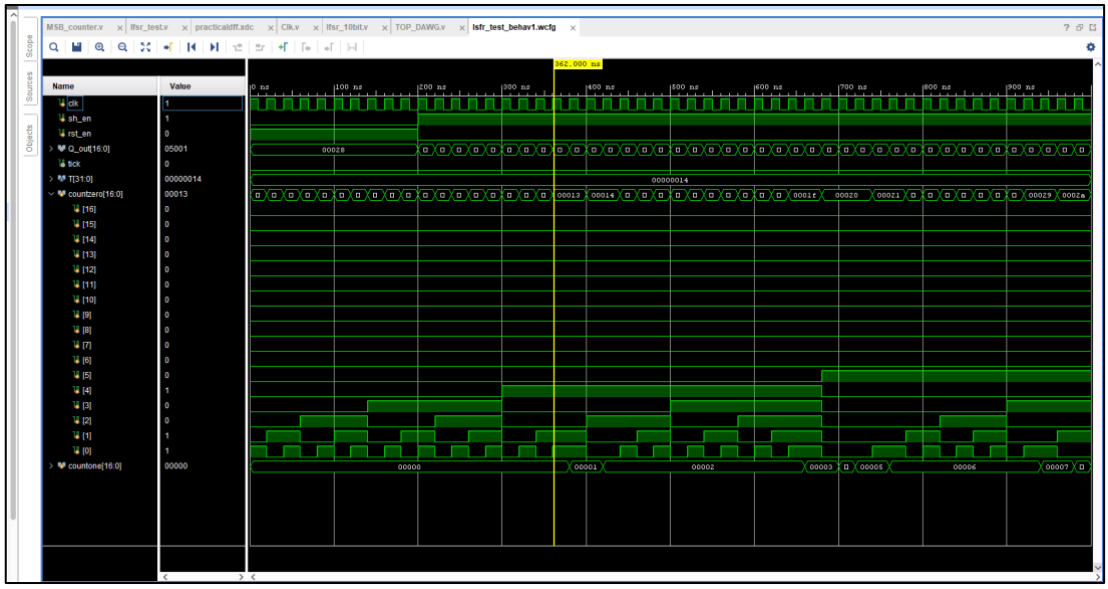


Figure 3.8 (Waveform showing counting of 0's and 1's in our most significant bit in wave form)

Part D: LFSR - Target to Board

The final part of the Lab was to implement our LFSR onto the Basys board. The .xdc file provided was altered so that switch v16 was set to trigger the reset and v17 was set to trigger the engage. As our design required 17 bits and the Basys board only has 16 LED pins, we were forced to only display the first 14 bits of the design as we needed one bit to display the max_tick_reg going high and one to separate the two components of our design.

W generated the bit stream using the .xdc file and programmed the Basys board. The design worked as intended with the reset, setting the LED's to our seed value to binary 101000. If reset was disengaged the board would hold state till Engage was triggered and then the bits would start shifting. We were unable to catch the max_tick_reg being triggered as this would have required us to wait for a long time. However the test case show that this would happen on the board as intended.

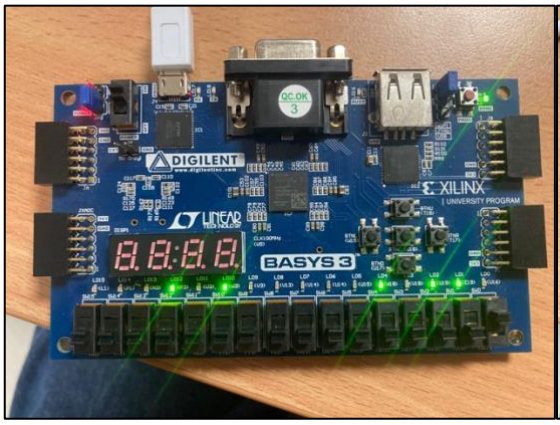


Figure 4.1 (zoomed in wave forms))

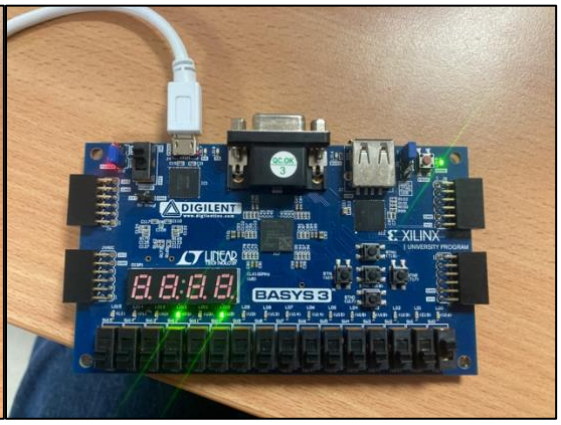


Figure 4.2 (zoomed in again))

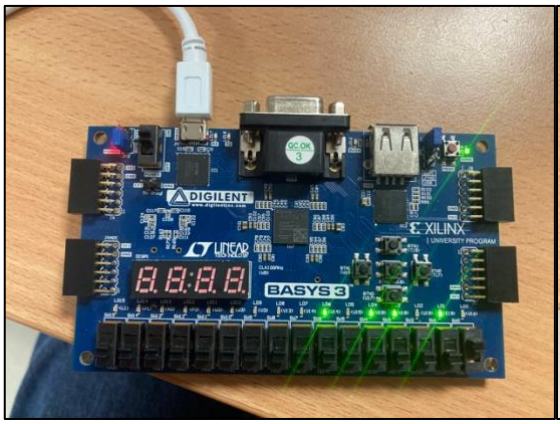


Figure 4.3 (zoomed in wave forms))

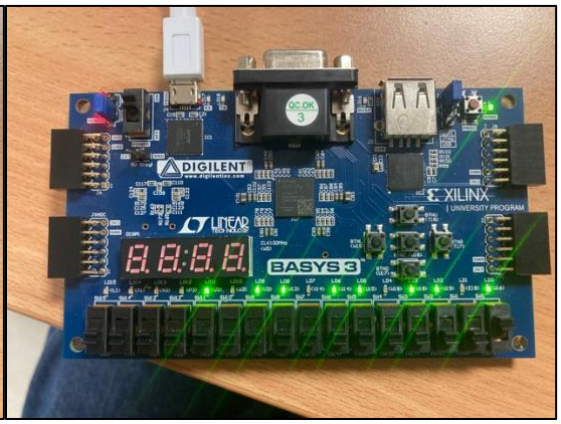


Figure 4.4 (zoomed in again))

Conclusions/Review

The Lab contained 3 parts overall, and we were able to create a working design for each. We started with flip-flops and gradually worked our way to implementing LFSR, while also gaining an understanding of how clocks are created and used in FPGA design. The Lab allowed us to gain a full understanding of each component in order to implement successfully on our Basys boards and give us the insight into how these components each are used to create desired outputs for FPGA.