Assignment 2

Name - Gwen McArdle

Student number – 18322248

Name of assignment – Mini Arithmetic Logic Unit

Date of submission – 18/12/2020

"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."

Signed – Gwen McArdle

Abstract

A system was designed that comprises an LFSR, a sequence detecting Finite State Machine (FSM), and a counter, in order to count the number of times a certain codeword is issued in the stream of bites generated by the LFSR in a full cycle of that LFSR.

Aim

- Practice writing Verilog modules.
- Practice debugging Verilog.
- Gain experience developing and testing a larger sequential design.
- Improve understanding of FSMs.

Method - Overview

1. A sequence detecting FSM, sequence_detector, was written. Sequence_detector searches for the pattern 10011. This module takes in a bit at a time from a bitstream, if the bits that are inputted match the pattern, the output wire 'detected' goes high.
2. The counter module, stop_watch_if, from Lab G, was adjusted to count the amount of times the sequence is detected.
3. A testbench was written to test the sequence detector and counter. The LFSR was used to generate the bitstream.
4. A top module was written that implements sequence_detector and stop_watch_if as well as some of the modules used in Lab F and G; clock, lfsr_19bit and disp_hex_mux.
5. The constraint file from Lab G was adjusted to include an LED that lights when 2^{N-1} cycles have occurred.
6. The design was implemented on the board.

Method – Clock

The clock is controlled by the module 'clock'. This module takes the clock signal generated by the FPGA, CCLK, as an input. It also takes in a signal 'clkscale' and outputs a slower clock signal called 'clk'. I chose 100000 to be my clkscale. This module inverts clk after every 100000 times CCLK inverts. I chose this clkscale because it is fast enough that the system does not take a long time to reach 2^{N-1} cycles, but slow enough that the user can comprehend the output on the board without difficulty. The original signal CCLK is used in the module disp_hex_mux as the cycle between anodes must be much quicker than the other modules need.

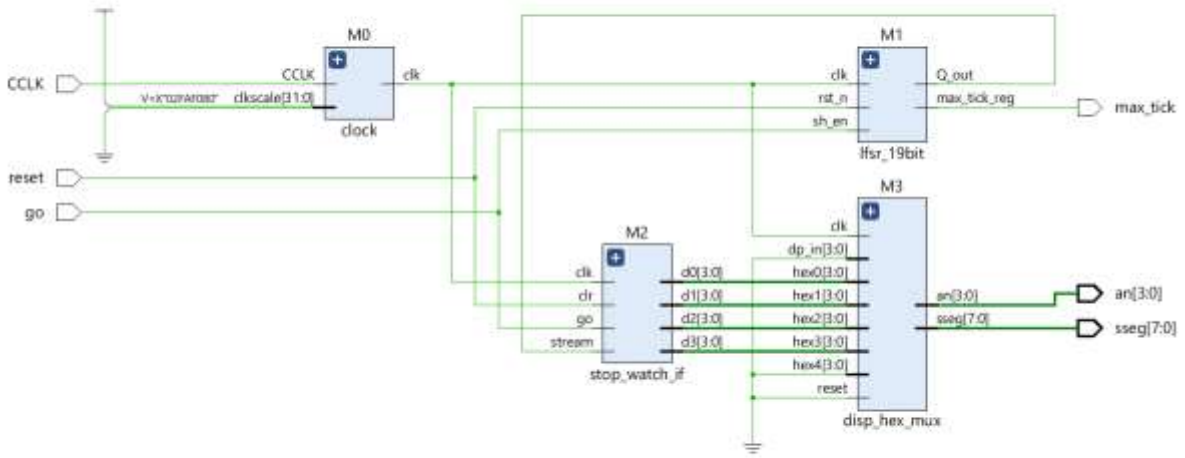
The constraints file assigns the clock generated by the FPGA to the wire named CCLK. It also includes a create-clock constraint which specifies the period and duty cycle of the clock.

The following screenshot shows the section of the XDC file that controls the clock.

```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports CCLK]
    set_property IOSTANDARD LVCMOS33 [get_ports CCLK]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports CCLK]
```

Design

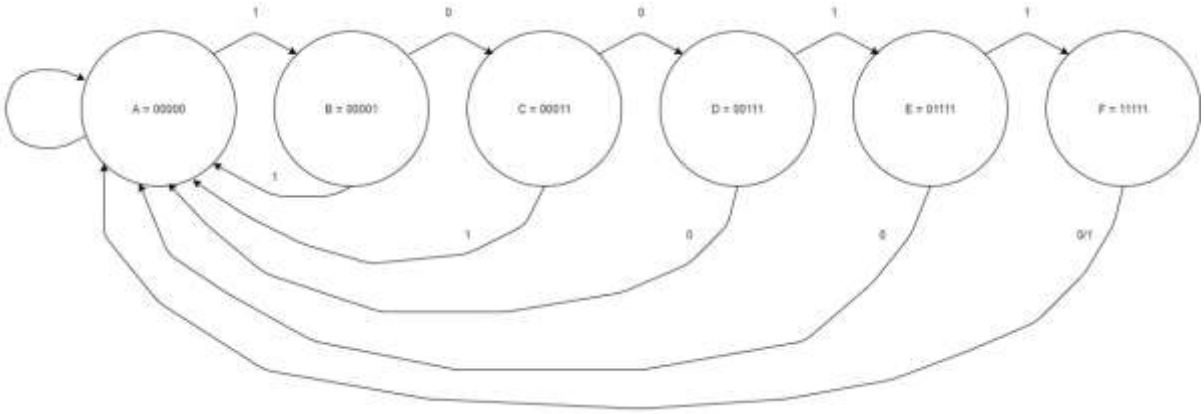
- The below functional diagram shows the system that was designed and implemented on the Basys 3 board.



- The FSM follows this table.

Current State	Input Bit	Next State	Output
00000	0	00000	0
	1	00001	0
00001	0	00011	0
	1	00000	0
00011	0	00111	0
	1	00000	0
00111	0	00000	0
	1	01111	0
01111	0	00000	0
	1	11111	0
11111	0	00000	1
	1	00000	1

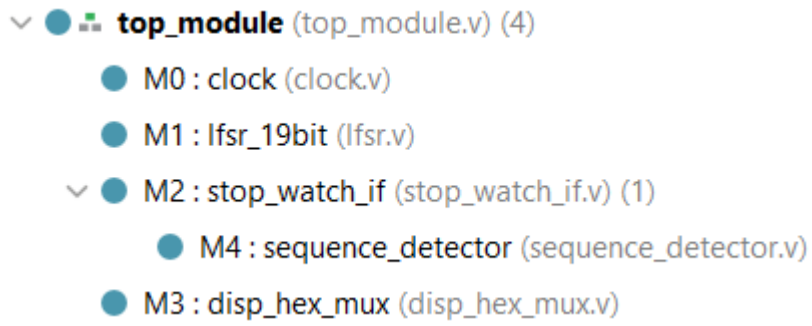
- The state machine diagram shows how the sequence_detector works.



The FSM starts in State A until a 1 is inputted as this is the first bit in the sequence I'm searching for. When this happens, it moves on to State B. The next bit we are looking for is 0, if the next bit inputted into the FSM is a 0, it moves on to State C. If it isn't a zero it returns to State A as this sequence doesn't match the pattern, and looks for the first bit of the sequence again.

If the system reaches State C, then it checks if the next bit is a 0, if it is, it continues on to State D, if not it returns to State A. This pattern is repeated until the system is in State F. When State F is reached, the output wire 'detected' goes high for one clock cycle, and the system returns to State A.

- The project hierarchy can be seen from the following screenshot.



Test Plan

The LFSR was previously tested in Lab F and no changes have been made.

- First, I have tested inputs that include the sequence we are searching for:

Stream1 checks if the sequence is detected if it's the first five bits.

Stream2 checks if the sequence is detected if it starts on the second bit.

Stream3 checks if the sequence is detected twice in a row.

Stream4 checks if the sequence is detected twice if it occurs twice, but there is overlap between the two occurrences.

Stream5 checks if the sequence is detected twice with a gap in between.

```
module testbench();

    reg stream1, stream2, stream3, stream4, stream5, stream6;
    wire [19:0] count1, count2, count3, count4, count5, count6;
    reg clk, reset, sh_en;

    stop_watch_if M0 (.stream(stream1), .clk(clk), .go(sh_en), .clr(reset), .count(count1));
    stop_watch_if M1 (.stream(stream2), .clk(clk), .go(sh_en), .clr(reset), .count(count2));
    stop_watch_if M2 (.stream(stream3), .clk(clk), .go(sh_en), .clr(reset), .count(count3));
    stop_watch_if M3 (.stream(stream4), .clk(clk), .go(sh_en), .clr(reset), .count(count4));
    stop_watch_if M5 (.stream(stream5), .clk(clk), .go(sh_en), .clr(reset), .count(count5));
    stop_watch_if M6 (.stream(stream6), .clk(clk), .go(sh_en), .clr(reset), .count(count6));

    initial begin
        clk = 1'b1;
        forever begin
            #0.05 clk = !clk;          //Therefore, clock cycle is 0.0000ns
        end
    end

    initial begin
        reset = 1'b1;
        sh_en = 1'b0;
        #1                             //10 clock cycles
        reset = 1'b0;
        sh_en = 1'b1;
        #52600
        sh_en = 1'b0;
    end

    initial begin
        #1
        stream1 = 1'b1;
        stream2 = 1'b0;
        stream3 = 1'b1;
        stream4 = 1'b1;
        stream5 = 1'b1;
        stream6 = 1'b0;
        #0.1
        stream1 = 1'b0;
        stream2 = 1'b1;
        stream3 = 1'b0;
        stream4 = 1'b0;
        stream5 = 1'b0;
        stream6 = 1'b0;
        #0.1
        stream1 = 1'b0;
        stream2 = 1'b0;
        stream3 = 1'b0;
        stream4 = 1'b0;
        stream5 = 1'b0;
        stream6 = 1'b0;
        #0.1
        stream1 = 1'b1;
        stream2 = 1'b0;
        stream3 = 1'b1;
        stream4 = 1'b1;
        stream5 = 1'b1;
        stream6 = 1'b0;
        #0.1
        stream1 = 1'b1;
        stream2 = 1'b1;
        stream3 = 1'b1;
        stream4 = 1'b1;
        stream5 = 1'b1;
        stream6 = 1'b0;
        #0.1
        stream1 = 1'b0;
        stream2 = 1'b1;
        stream3 = 1'b1;
        stream4 = 1'b0;
        stream5 = 1'b0;
        stream6 = 1'b0;
        #0.1
```

```
        stream1 = 1'b0;
        stream2 = 1'b1;
        stream3 = 1'b1;
        stream4 = 1'b0;
        stream5 = 1'b0;
        stream6 = 1'b0;
        #0.1
        stream1 = 1'b0;
        stream2 = 1'b0;
        stream3 = 1'b0;
        stream4 = 1'b0;
        stream5 = 1'b1;
        stream6 = 1'b0;
        #0.1
        stream1 = 1'b0;
        stream2 = 1'b0;
        stream3 = 1'b0;
        stream4 = 1'b1;
        stream5 = 1'b0;
        stream6 = 1'b0;
        #0.1
        stream1 = 1'b0;
        stream2 = 1'b0;
        stream3 = 1'b1;
        stream4 = 1'b1;
        stream5 = 1'b0;
        stream6 = 1'b0;
        #0.1
        stream1 = 1'b0;
        stream2 = 1'b0;
        stream3 = 1'b1;
        stream4 = 1'b0;
        stream5 = 1'b1;
        stream6 = 1'b0;
        #0.1
        stream3 = 1'b0;
        stream5 = 1'b1;
        #0.1
        stream5 = 1'b0;
    end

endmodule
```

The following waveform was generated:

Stream1, stream2 and stream5 give the expected result. This means that the FSM does detect when the stream includes 10011. However, stream3 and stream4 only detect one occurrence of the sequence. This means there is a flaw in my design.

- Second, I have tested inputs that don't include the sequence we are searching for:

Stream1 checks if the sequence is detected if the first bit isn't found.

Stream2 checks if the sequence is detected if the second bit isn't found.

Stream3 checks if the sequence is detected if the third bit isn't found.

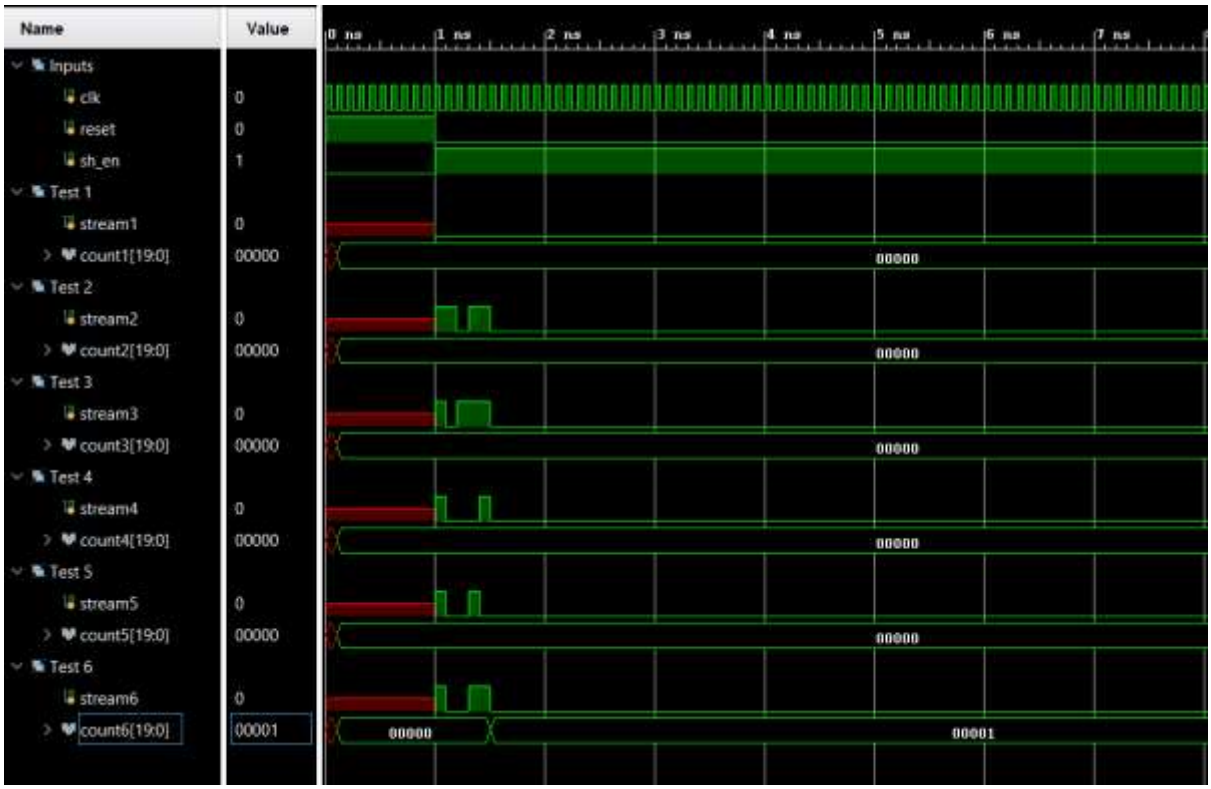
Stream4 checks if the sequence is detected if the fourth bit isn't found.

Stream5 checks if the sequence is detected if the fifth bit isn't found.

Stream6 is a control, to verify that the FSM is still operational.

```
module testbench();

    reg stream1, stream2, stream3, stream4, stream5, stream6;
    wire [19:0] count1, count2, count3, count4, count5, count6;
    reg clk, reset, sh_en;

    stop_watch_if M0 (.stream(stream1), .clk(clk), .go(sh_en), .clr(reset), .count(count1));
    stop_watch_if M1 (.stream(stream2), .clk(clk), .go(sh_en), .clr(reset), .count(count2));
    stop_watch_if M2 (.stream(stream3), .clk(clk), .go(sh_en), .clr(reset), .count(count3));
    stop_watch_if M3 (.stream(stream4), .clk(clk), .go(sh_en), .clr(reset), .count(count4));
    stop_watch_if M5 (.stream(stream5), .clk(clk), .go(sh_en), .clr(reset), .count(count5));
    stop_watch_if M6 (.stream(stream6), .clk(clk), .go(sh_en), .clr(reset), .count(count6));

    initial begin
        clk = 1'b1;
        forever begin
            #0.05 clk = !clk;          //Therefore, clock cycle is 0.0000ns
        end
    end

    initial begin
        reset = 1'b1;
        sh_en = 1'b0;
        #1                             //10 clock cycles
        reset = 1'b0;
        sh_en = 1'b1;
        #52600
        sh_en = 1'b0;
    end

   initial begin
       #1
       stream1 = 1'b0;
       stream2 = 1'b1;
       stream3 = 1'b1;
       stream4 = 1'b1;
       stream5 = 1'b1;
       stream6 = 1'b1;
       #0.1
       stream1 = 1'b0;
       stream2 = 1'b1;
       stream3 = 1'b0;
       stream4 = 1'b0;
       stream5 = 1'b0;
       stream6 = 1'b0;
       #0.1
       stream1 = 1'b0;
       stream2 = 1'b0;
       stream3 = 1'b1;
       stream4 = 1'b0;
       stream5 = 1'b0;
       stream6 = 1'b0;
       #0.1
       stream1 = 1'b0;
       stream2 = 1'b1;
       stream3 = 1'b1;
       stream4 = 1'b0;
       stream5 = 1'b1;
       stream6 = 1'b1;
       #0.1
       stream1 = 1'b0;
       stream2 = 1'b1;
       stream3 = 1'b1;
       stream4 = 1'b1;
       stream5 = 1'b0;
       stream6 = 1'b1;
       #0.1
       stream1 = 1'b0;
       stream2 = 1'b0;
       stream3 = 1'b0;
       stream4 = 1'b0;
       stream5 = 1'b0;
       stream6 = 1'b0;

   end

endmodule
```

The following waveform was generated:



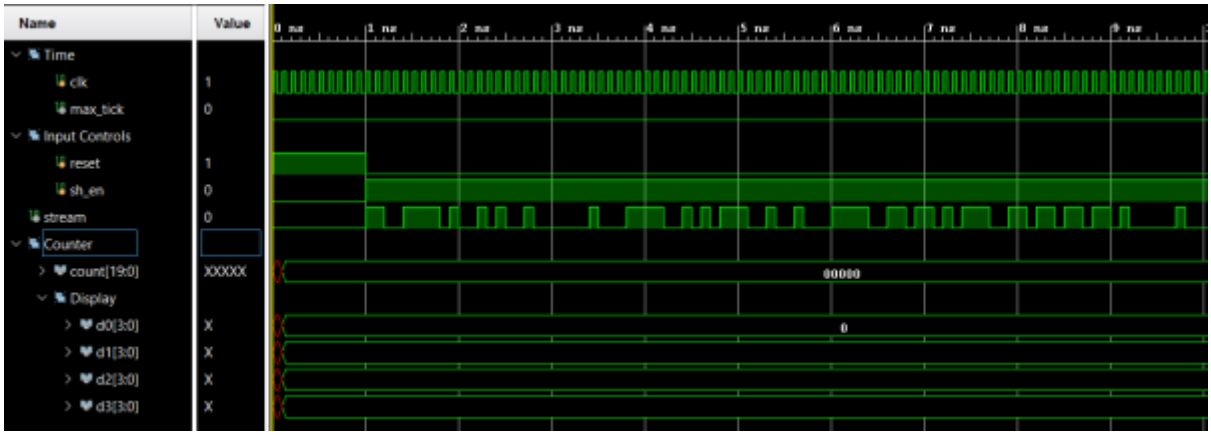
Each stream gives the expected result.

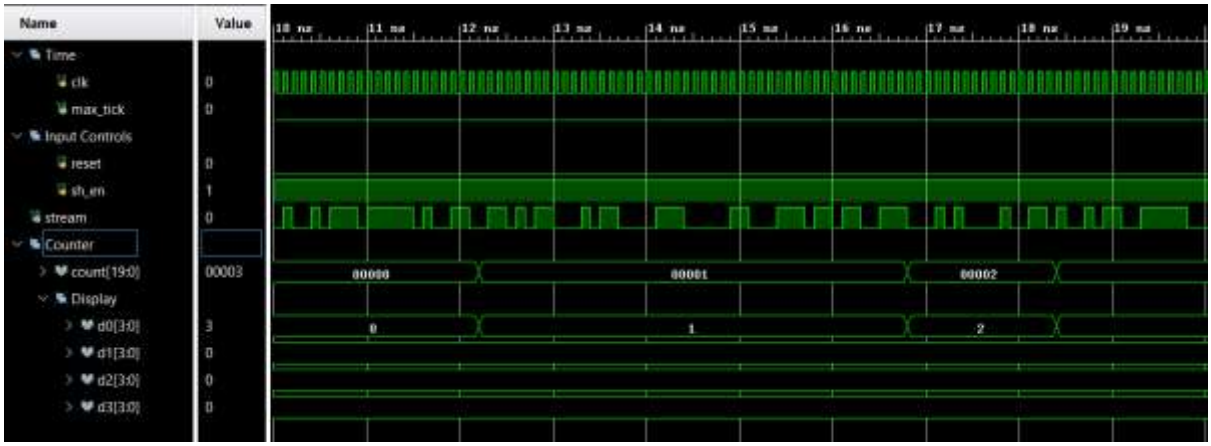
Results

This is a screenshot of a testbench that uses the LFSR module to generate the bitstream and uses stop_watch_if to count the occurrences of the sequence 10011. (sequence_detector is called on within stop_watch_if.)

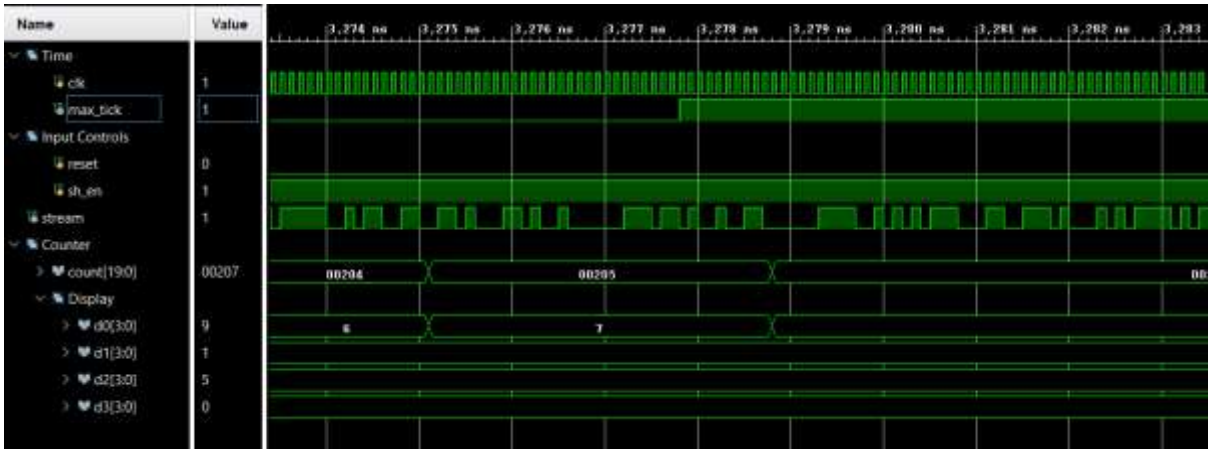
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: TCD
// Engineer: Oven Mokrdle
// Create Date: 13.12.2020 22:11:46
// Module Name: testbench
// Project Name: Assignment 2
//////////////////////////////////////////////////////////////////////////////////

module testbench();

    wire max_tick, stream;
    wire [19:0] count;
    reg clk, reset, sh_en;
    wire [3:0] d0, d1, d2, d3;

    lfsr_19bit M0 (.clk(clk), .sh_en(sh_en), .rst_n(reset), .max_tick_reg(max_tick), .Q_out(stream));
    stop_watch_if M1 (.stream(stream), .clk(clk), .go(sh_en), .clr(reset), .d3(d3), .d2(d2), .d1(d1), .d0(d0), .count(count));

    initial begin
        clk = 1'b1;
        forever begin
            #0.05 clk = !clk;          //Therefore, clock cycle is 0.00002ns
        end
    end

    initial begin
        reset = 1'b1;
        sh_en = 1'b0;
        #1                             //10 clock cycles
        reset = 1'b0;
        sh_en = 1'b1;
        #52600                         //(2^19) - 1 = 524287 clock cycles = 52428.7 ns. 52600 gives time for the cycle to start again
        sh_en = 1'b0;
    end


endmodule
```

The following two waveforms show the beginning of the waveform the testbench. It shows the count increments, as does d0, when the desired sequence occurs.





The following waveform shows the part of the waveform where max_tick goes high.



The counter is at 205 in hex, which is 517 in decimal, after 2^{N-1} cycles.

The following screenshot shows the “Register as Flip Flop” section of the Utilization Report.

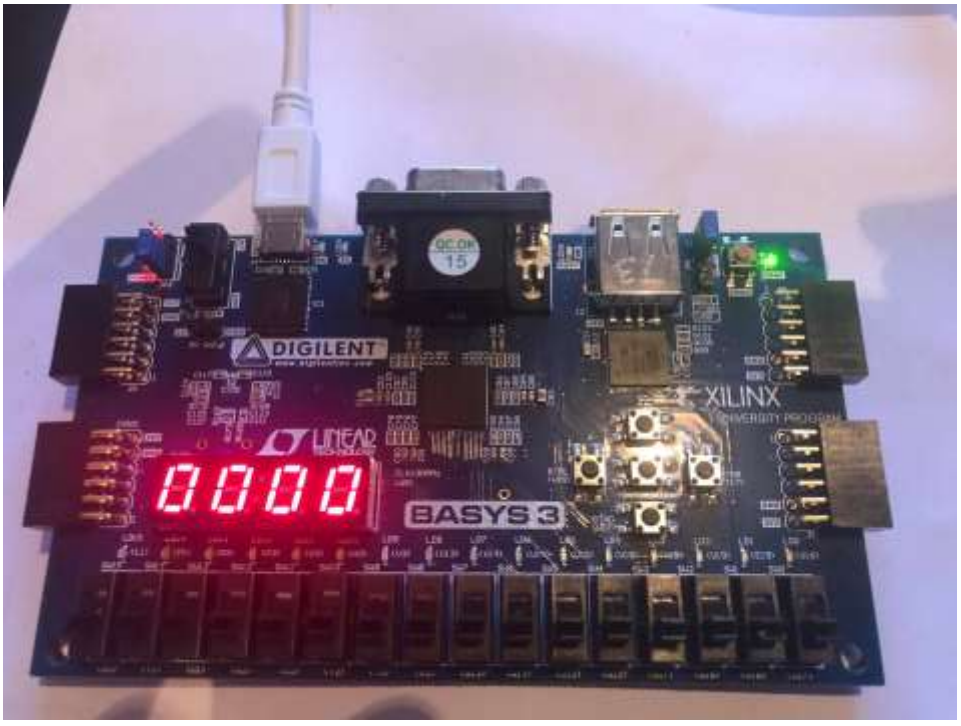
1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs	47	0	20800	0.23
LUT as Logic	47	0	20800	0.23
LUT as Memory	0	0	9600	0.00
Slice Registers	128	0	41600	0.31
Register as Flip Flop	128	0	41600	0.31
Register as Latch	0	0	41600	0.00
F7 Muxes	0	0	16300	0.00
F8 Muxes	0	0	8150	0.00

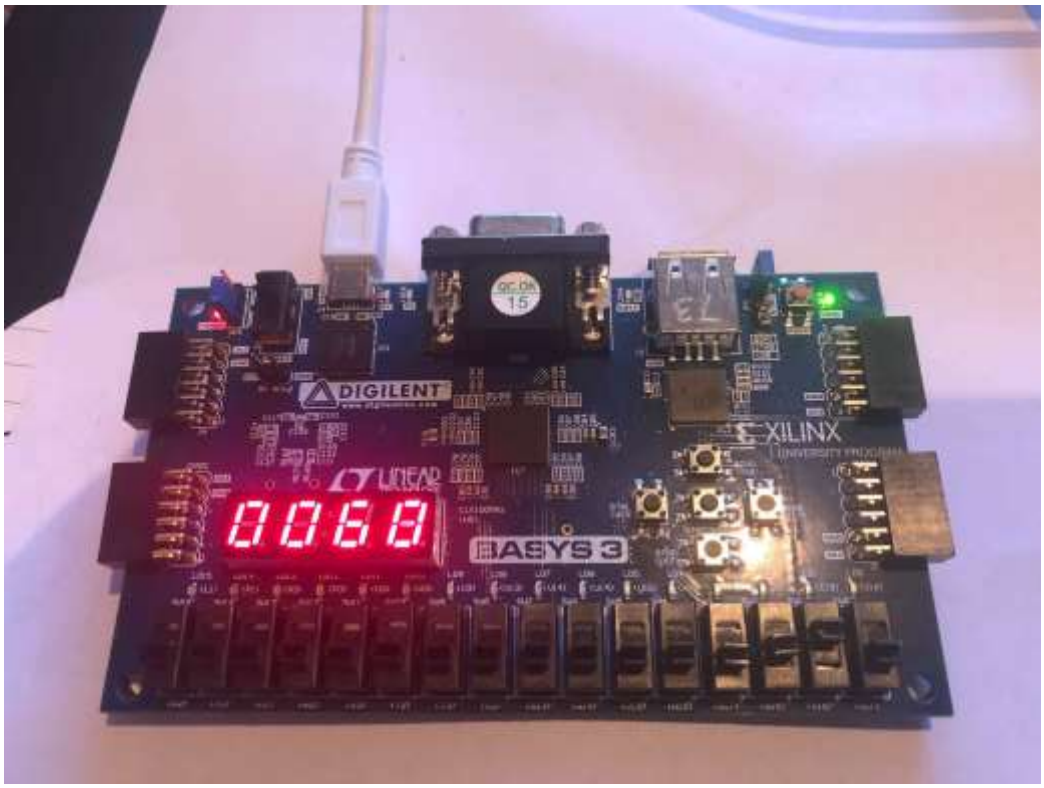
Demo

The only external controls are Switches 0 and 1. Switch 0 resets the counter and Switch 1 enables the counter.

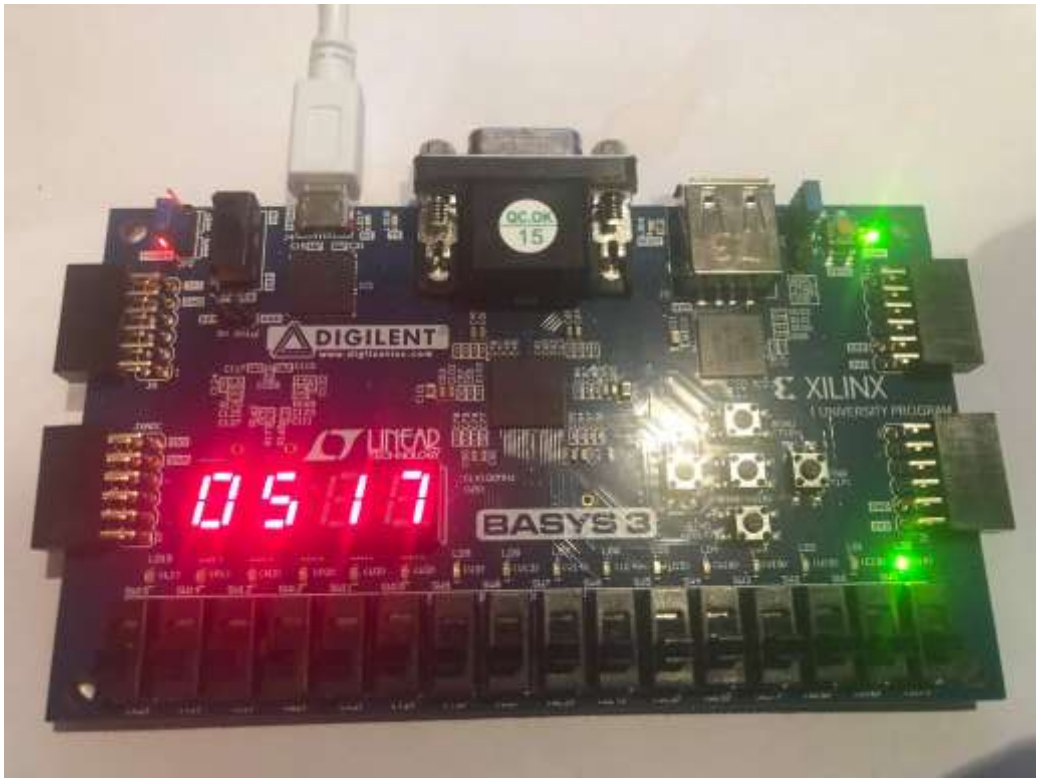
The board starts at zero.



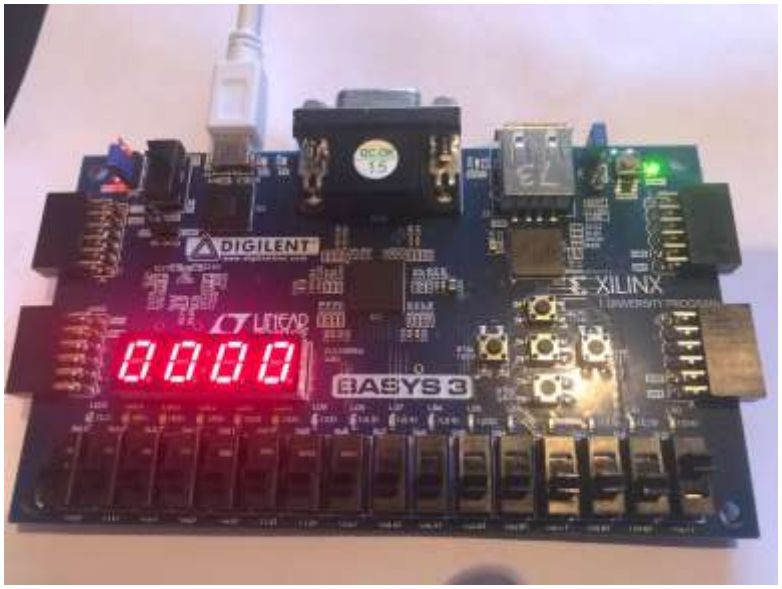
When switch 1 is on, the counter increases at a varying rate, depending on the sequence detection.



When 2^{N-1} cycles have occurred, LED 0 lights. The sequence 10011 was found 517 times in 2^{N-1} cycles.

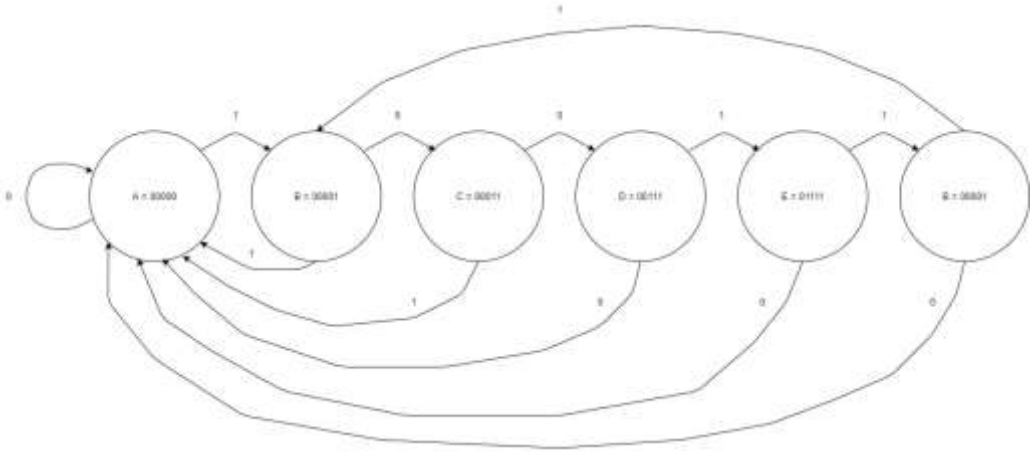


The display resets to zero when switch 0 is turned on.



Conclusions

The system has strengths and weakness. Due to a methodical testing approach, I discovered that I made a mistake in the design of the system. The FSM does not detect a sequence if it overlaps with the previous sequence. If I were to repeat the assignment, I would design an FSM that operates like the following diagram.



However, the top-level module, LFSR, and display all work very well. The FSM works for the majority of cases, but could be improved to catch overlapping cases, if the above change were made.

Appendices

Lab F has been previously submitted.