## Abstract:

The purpose of this project was to bring previous labs mainly Labs F-G together to develop and test a larger sequential design and then target it to the board. The aim was to detect a certain sequence (using a finite state machine) of numbers coming from the bit stream produced by the LFSR. Then once this sequence was found we should increment our counter by one and show it on the 7-seg display.
In conclusion when my final code was tested on the board it worked flawlessly as with each time the codeword appeared it would add on to the counter and when you reached 2^(n-1) steps the counter reset and started counting again.

## Aims:

- To understand and implement a finite state machine (Mealy or Moore)
- Use tested code from previous labs that have been tested and verified.
- To understand and use the 7 segment display

## Introduction:

In this lab I was started off with my seed value being 11 bits long and had the value of 1110000111. Then the code word that I am looking for has the value of 1110. To try and detect this I used one higher level module called highh.v Then inside the highh.v I then have another 4 modules which are clock, lfsr, FSM, counter and the display module.

The clock module allows you take in the clock signal from pin w5 and then divide it by the clkscale to get your desired value of clock.

The LFSR module generates the bit stream of numbers by outputting bit 10 and then xor bits 10 and 6 to get a value. It then shifts everything to the left by one bit and then stores the value it created in the LSB. The lfsr will also output a high for one clock cycle whenever it has cycled 2^(n-1) times. In my case it was 2047 times.

The FSM reads in the bit stream coming from the LFSR and it uses a Moore finite state machine to detect my code word of 1110. If it detects this word it will then produce a high for 1 clock cycle and then return to 0 until it next detects it. It will then output this signal to the counter.

The counter will then take in the signal from the FSM. It will then add one to the counter. It will then convert this into a 4 digital decimal number. Eg the number cannot go higher than 9999.

Then the display module takes the output from the counter and then displays this on the 7 segment display.

## Experimental Set-up:

In this assignment there was only 2 main components that we needed to set up and use:

1. Vivado on a windows computer.
2. A Basys 3 board and a connecting cable to the computer.

## Building and Testing:

Before I loaded any of my modules on to the board I made sure that I tested each module using a test bench on Vivado.

The first module that I used was the LFSR. In this project I did not have to test this code again as I knew that he had been previously fully tested in a different lab. You can see the result below

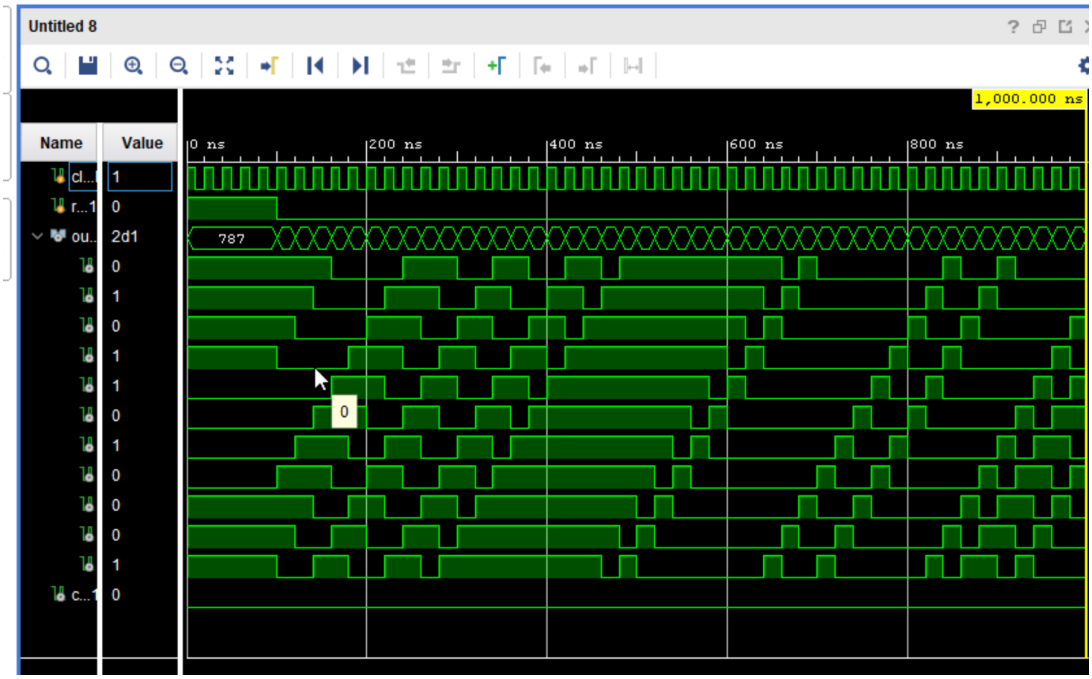


*Figure 1 LFSR testing*

Once I confirmed this was working correctly. I added to the code an output stream(one bit out) and reached counter that would go high whenever it reached 2^(n-1) times (2047). As you can see below.

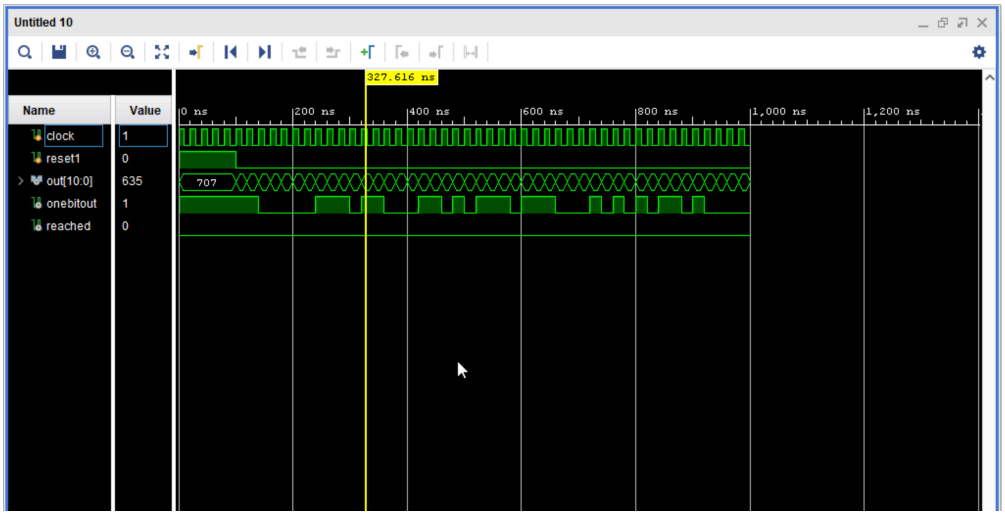


*Figure 2 onebitout output stream*

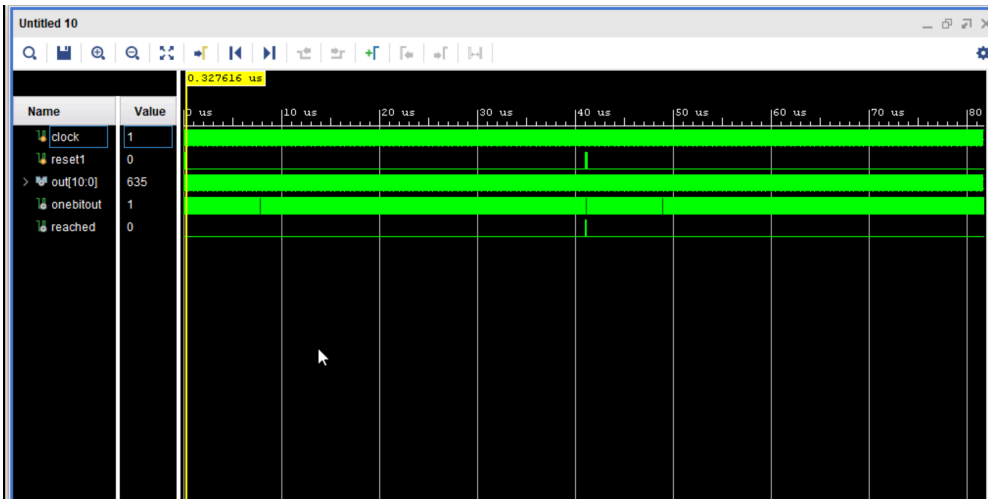


*Figure 3 reached going high*

Figure 2 shows the output stream that the FSM will eventually analyze. Figure 3 then shows reached being activated as LFSR as now reached 2^(n-1) cycles. When reached is equal to 1, it also turns the reset on. I added this feature now as it might be useful in future when it comes to resetting the counter.

The next module that was built was the finite state machine

Mealy Machine

The type of state machine that I chose to use was the Mealy machine. Below you can see how I came up with the minimization:

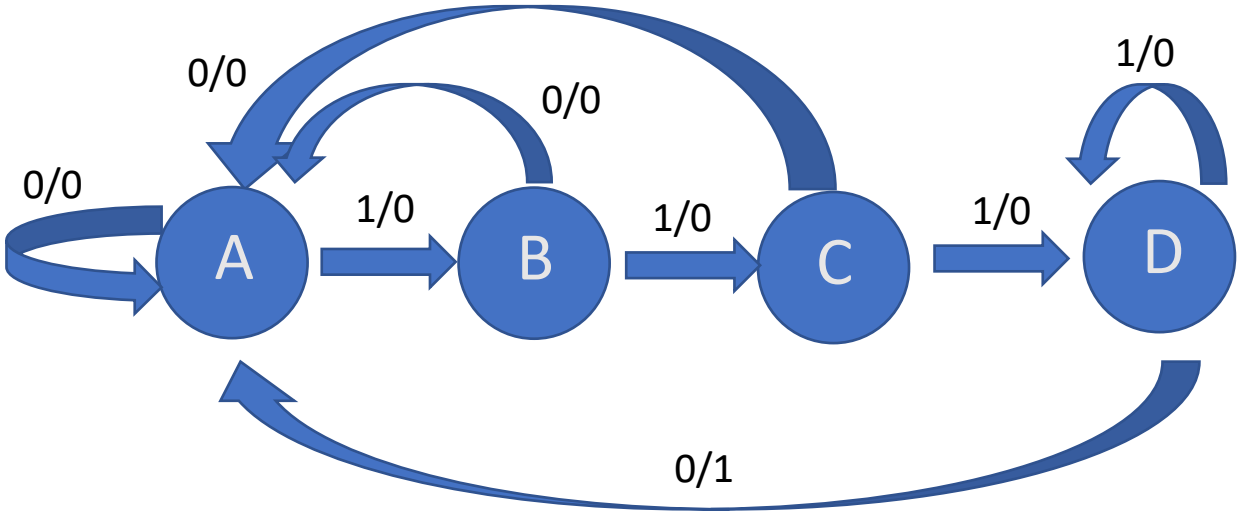


*Figure 4 My original FSM*

Key: input/output

| Current State | Next state | | Output | |
|---|---|---|---|---|
| | Input 0 | Input 1 | Input 0 | Input 1 |
| A | A | B | 0 | 0 |
| B | A | C | 0 | 0 |
| C | A | D | 0 | 0 |
| D | A | D | 0 | 1 |

So looking at this table we can see that no 2 rows are the same so from this table we can not make any minimizations. So we shall confirm if we cannot minimize it by using an implication chart.

Implication chart:

| | | | |
|---|---|---|---|
| B | B=C, | | |
| C | B=D | C=D | |
| D | X | X | X |
| | A | B | C |

In the first chart the bottom rows are all X because in Row D it has an output of 1 so its not the same as anything so has be X'd out.

| | | | |
|---|---|---|---|
| B | B=C, | | |
| C | X | X | |
| D | X | X | X |
| | A | B | C |

In the next table the C row also has to be X'd out because B cannot equal D and C cannot equal D. As if we try and find these comparisons in the table they are already X'd out.

| | | | |
|---|---|---|---|
| B | X | | |
| C | X | X | |
| D | X | X | X |
| | A | B | C |

We now do the same thing to the last row B. It's the same result with it also be X'd out.

With all the spaces being filled in with X's we can say that the above mealy diagram is the most efficient and minimised mealy FSM possibly for my given code word.

Below you can see my testing for my Mealy Finite State Machine:

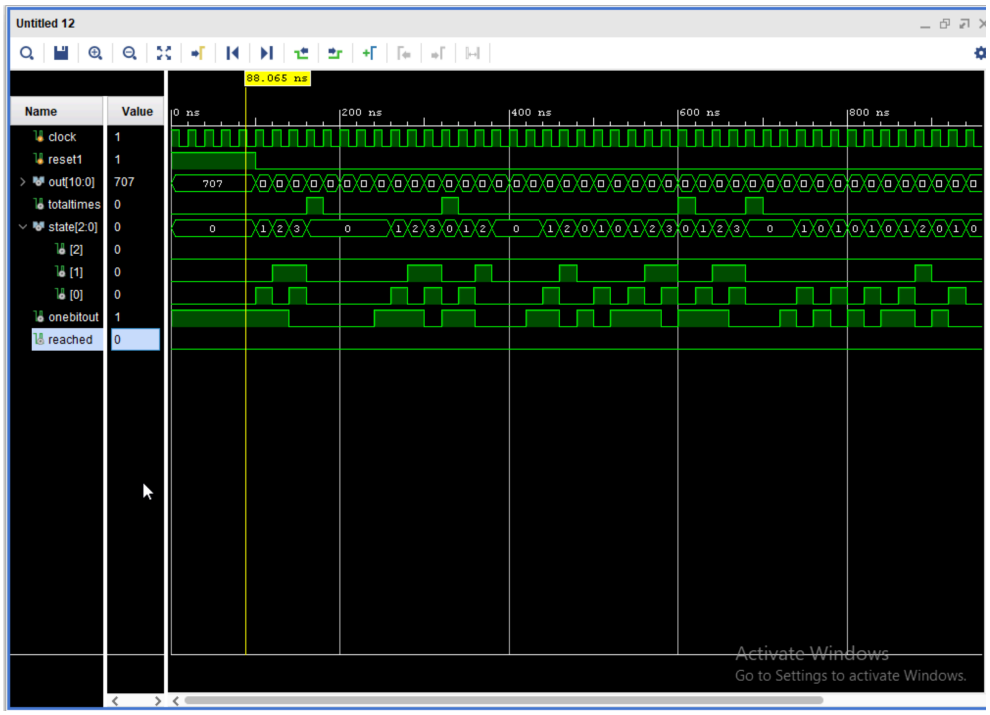


*Figure 5 testing for Mealy State Machine*

In the above picture you can see there has been a couple of variables added, such as state and totaltimes. The state is stage at which the mealy fsm is at. If it is at no.3 then it has detected the number 1110 and will sent a flag to one. This flag is called totaltimes. Above figure 4 you can see that when state is at number 3, totaltimes goes high for one clock cycle. If state is at no 2 then we know it has detected a number 111 and if it's at no 1 then its detected the number 11 and so on.

Counter:

The counter receives the flag coming from the FSM module. It then keeps a record of how many times the flag is flagged. During each clock cycle it updates this value and then converts this number into 4 digit decimal number (e.g max number =9999) to be displayed on the 7 segment display. It outputs 4 wires each for the 4 different 7 segments display. Below you can see the testing for this module:

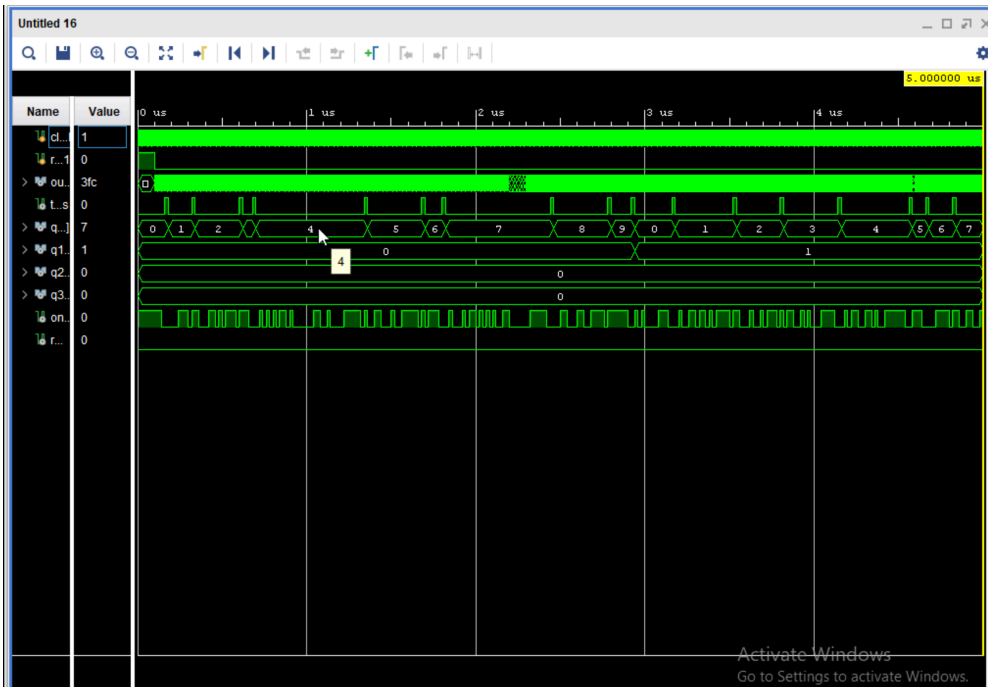


*Figure 6 counter testing*

In figure 5 it clearly shows the numbers counting up and whenever it hits 9 and needs to increment by it resets it to 0 and increments the next digit to 1. This was the last simulation done on Vivado before testing the code on the board.

I also have a module that changes the value of the clock. This is done because the clock signal from W5 is too fast and we wouldn't be able to see results on the board. So in my clock module I have 2 inputs. One is the input from w5 and the other is a clock scale. We divide w5 clock by clock scale and then output this to the rest of the files. We labelled the new clock signal as LD0 which is what will appear in the XDC file.

The next step is to create a top level and then load and test this on the FPGA. The top level file will instantiates all of the given files. The block diagram below shows what the opt level is doing:

Block Diagram (simple version):

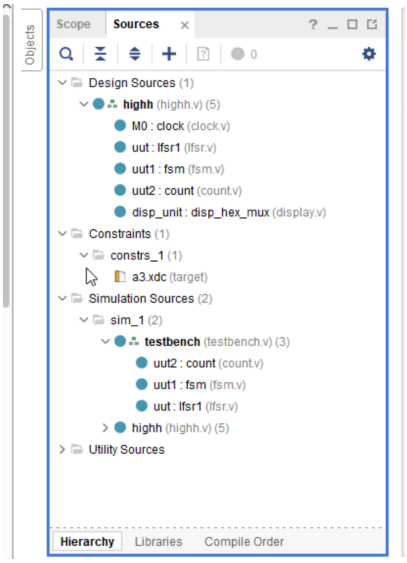


*Figure 7 Hierarchy*

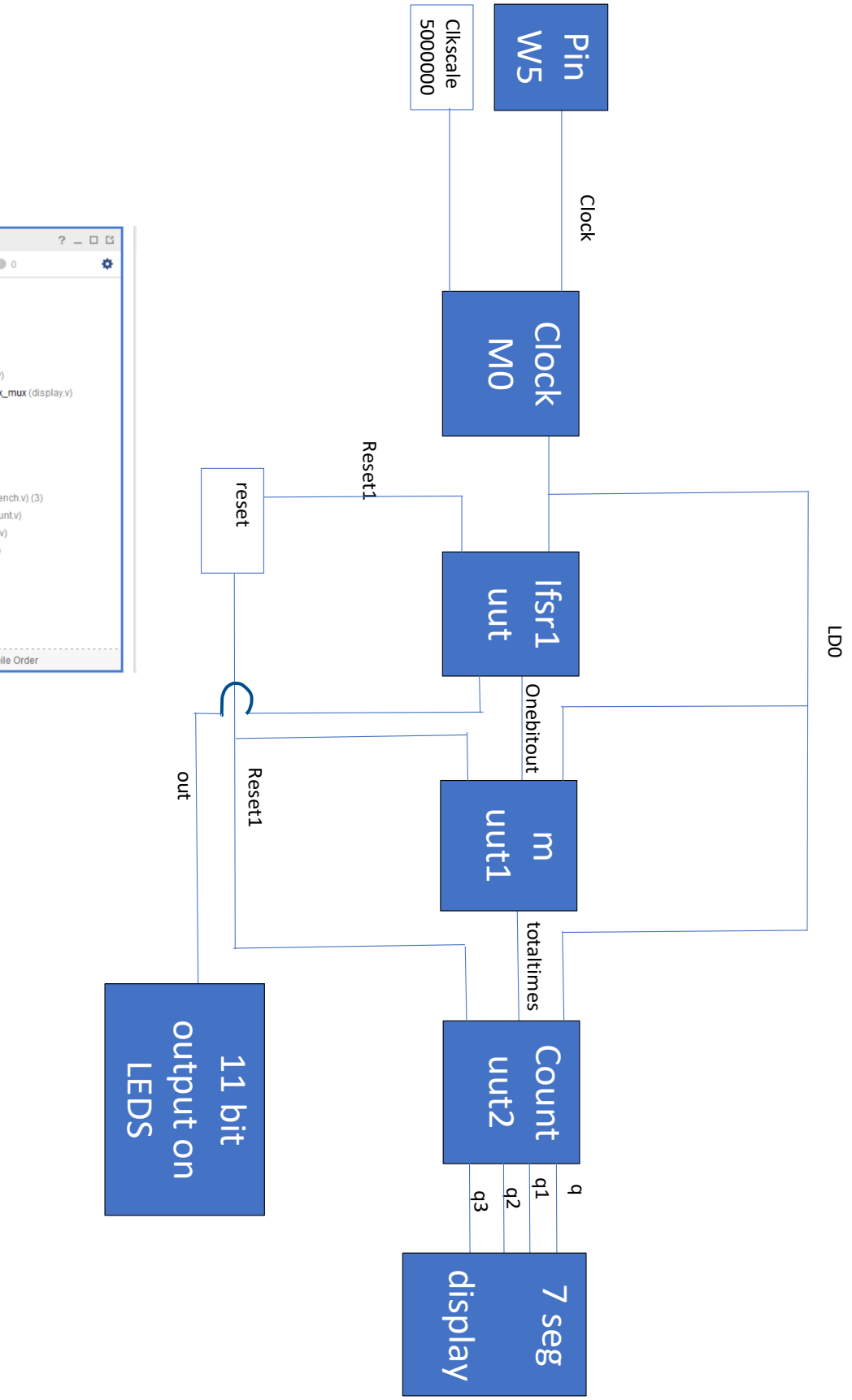

Figure 8 Block Diagram
Demo:

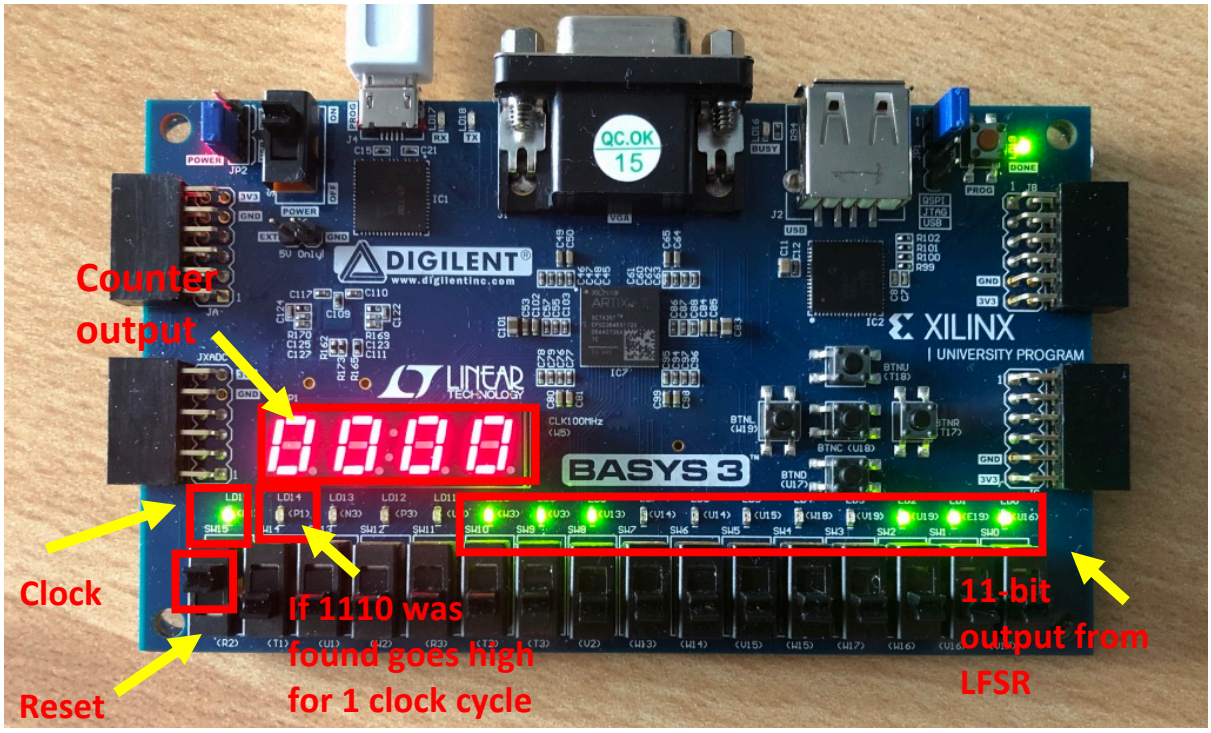


*Figure 9 input and outputs of board*

Clock: L1
Flag: P1
Reset: Switch R2
11 bit number: W3->U16 with LSB= W3 and MSB = U16

The code for this runs by itself, if you need to reset the board at any time just use the switch R2. If you need more instructions on how this works then please watch the video attached to this file.

It should run until number 128 is shown on the counter and then it will reset to zero as it has hit 2^(n-1) cycles. When looking at this in the test bench it also confirms this number to be correct

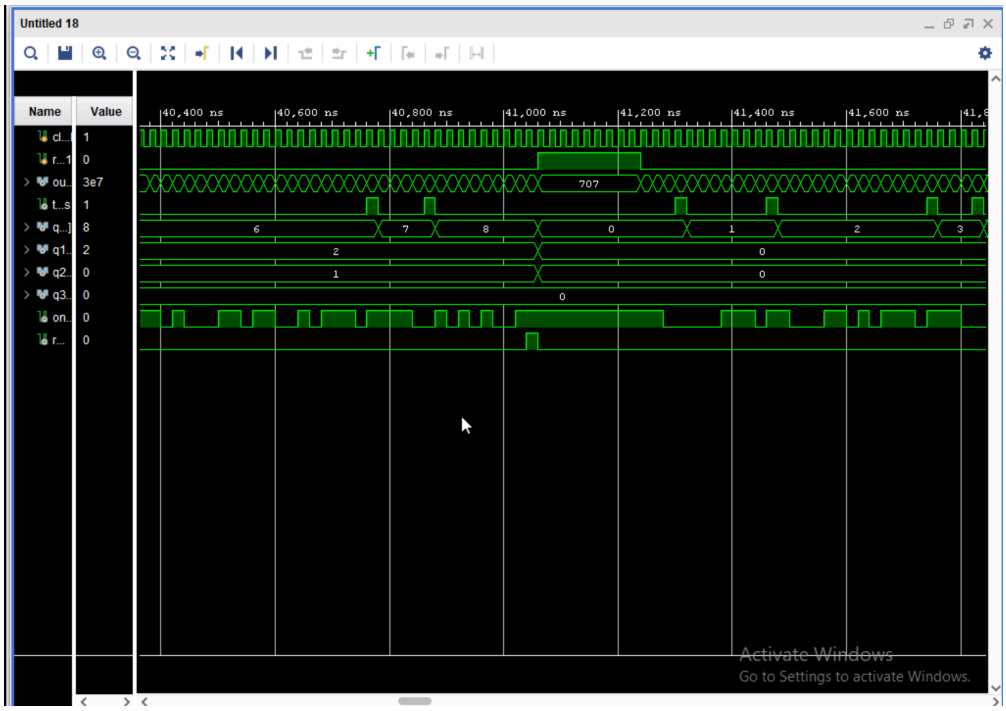


*Figure 10 number of times it see 1110 before resetting*

Utilization Report:

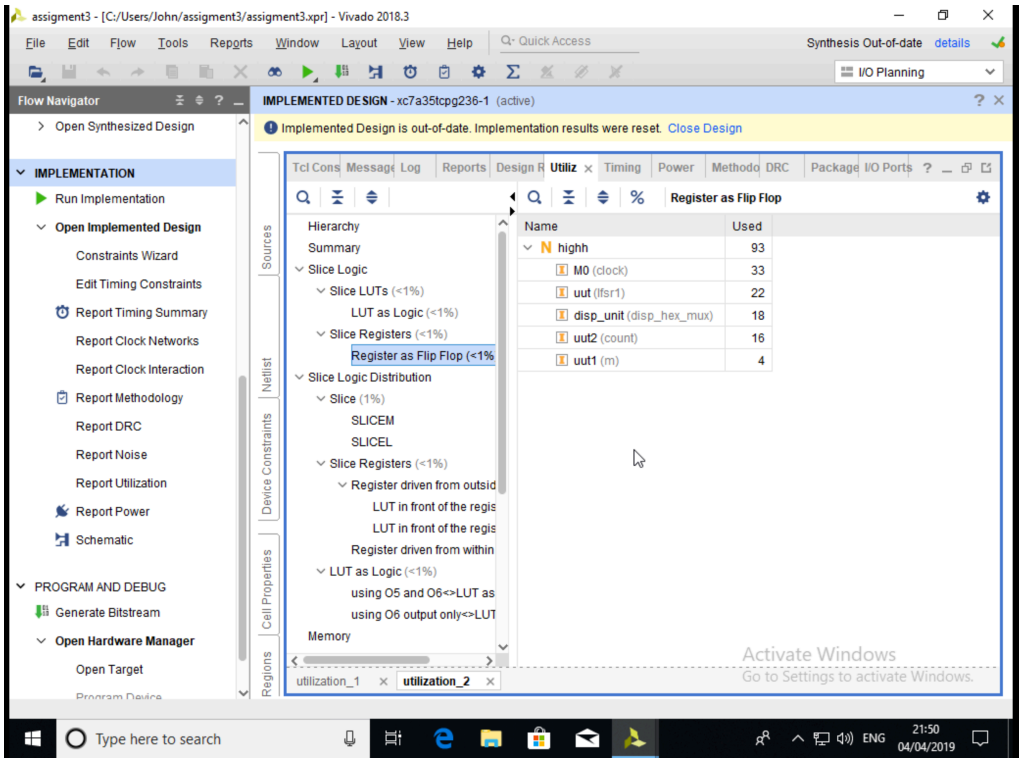


*Figure 11 No of flip flops used*

## Discussion:

The hardest part of the project was timings of the different always blocks in the counter and fsm.v. I originally tried to count inside the fsm however this wouldn't have worked because its value kept getting reset to zero before I could add a new value to it. So I had to replace it with a simple flag and then send that flag to new module which would add number the number of flags.
Currently when I run the code, the flag output is one clock cycle behind the lfsr outputting the code word and then my counter on the 7-segment display is further clock cycle behind that. To change this all that would need to be done would be to change all the _reg to _next for the outputs of the different modules however this is not correct so it will be left as _reg.

## Conclusion:

Overall, I achieved the aims that was set out for this project. As when my code word appears it will add one to the counter and when it reaches 2^(n-1) cycles it will then reset itself back to a count of zero.  It will reach a count of 128 before being reset. Because of my given codeword there was no overlapping occurring, so I did not need to worry about this.