Assignment 2 - Report

Stephen Komolafe (21336975)

The objective of this assignment was to develop a system that would detect a given code word in a stream of binary values. The system in question would consist of a linear feedback shift register (LFSR) to generate said stream of binary values each value being the most significant bit (MSB) from the LFSRs output, a Moore model finite state machine (FSM) to detect the code word within the stream of MSB outputs from the LFSR, and a counter to count the number of times the code word was detected in the stream during a full cycle using the FSM output as an input. A plan showcasing the intended hierarchy can be seen in Figure 1 below:

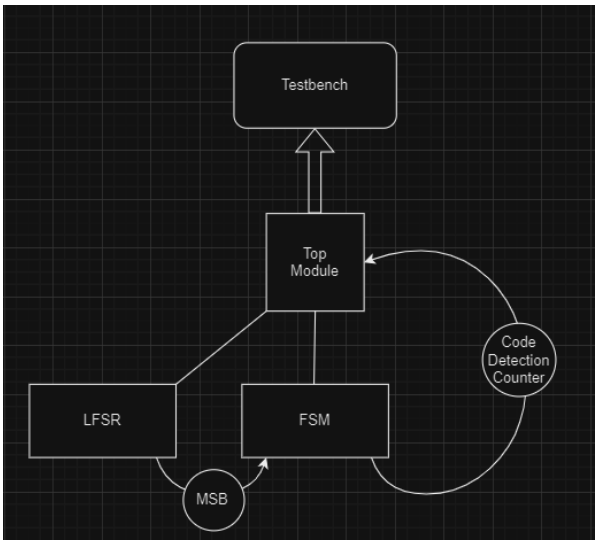


Figure 1.) Block diagram for showing system hierarchy plan

LFSR Development

As a functional maximal length 12-bit LFSR had already been developed in lab F, I was able to reuse the module and update it to fit the specification outlined in the setup document (assign2_lfsr_setup.pdf) which was provided via Blackboard. In it, the necessary requirements of the system specific to each student where defined. In my case, LFSR was required to have a width of 22 bits (therefore taps are 21 and 20) and utilise XNOR feedback logic. With this information and knowledge on how LFSRs work from Lab F a diagram can be made to illustrate how the 22-bit LFSR would function as seen in Figure 2.

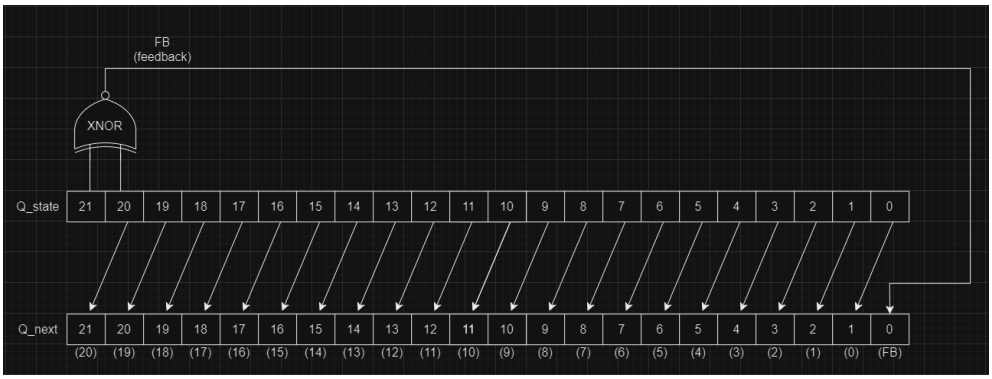


Figure 2.) Block diagram for 22-bit LFSR

Additionally, the document specified the specific seed value to be used that being '0001011001001110011111'. From this I was able to test the updated module using the testbench from lab F, which produced the following waveform:

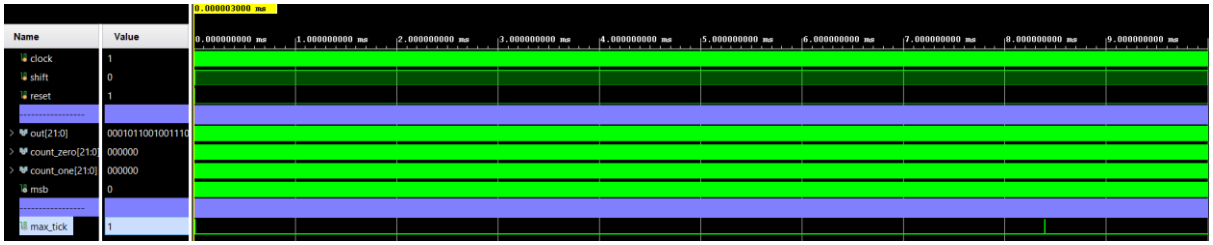


Figure 3.) 22-bit LFSR Waveform (Full)

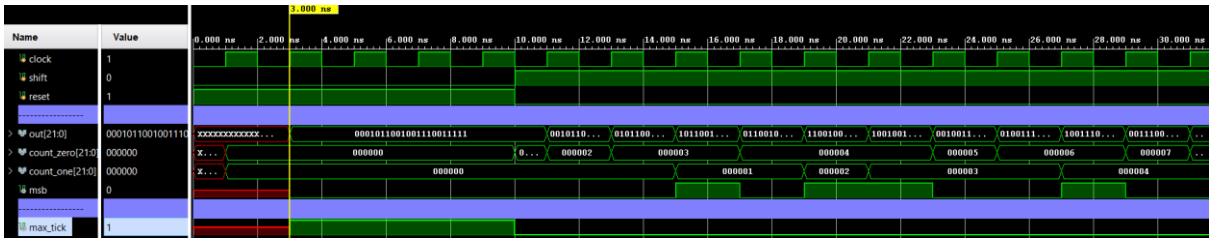


Figure 3.1.) 22-bit LFSR Waveform (Zoomed In at Start)

From these waveforms in figure 3 and 3.1, it is clear that with the bit width of the LFSR increased to 22 the testbench requires a longer simulation runtime to be set, with it being set to 10000000ns, in order to find the LFSR's maximum length as opposed the previous lab where it was 12-bits wide and only needed a simulation runtime of 10000ns.

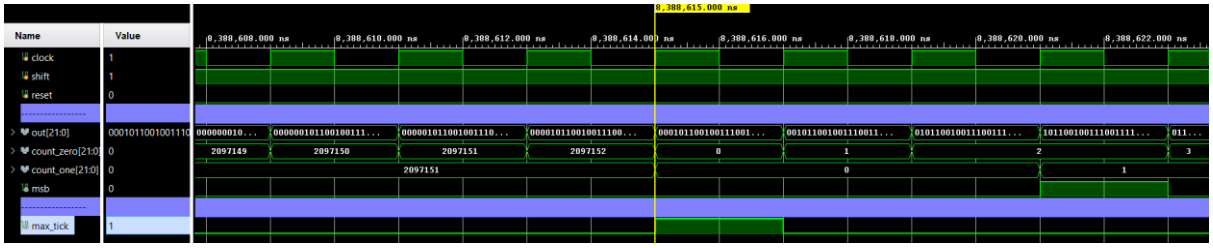


Figure 3.2.) 22-bit LFSR Waveform (Zoomed In at Max Tick)

From observing the waveform closer it can be seen that the number of times MSB of the output was zero is 2097152 and the number of times it is one is 2097151, therefore the maximum number of cycles required for a maximal length 22-bit LFSR to return to its seed value is 4,194,303. This is verified by the general equation for finding the max LFSR cycle that being:

(2^22)-1 cycles = 4,194,303 cycles

FSM Development

The previously mentioned setup document also denoted the student specific codeword to look for in the bit stream which in my case was '11111010101'. Before writing any of the code for the FSM a state machine diagram was made to illustrate the intended procedure in finding the codeword in the

stream MSB's from the LFSR output.

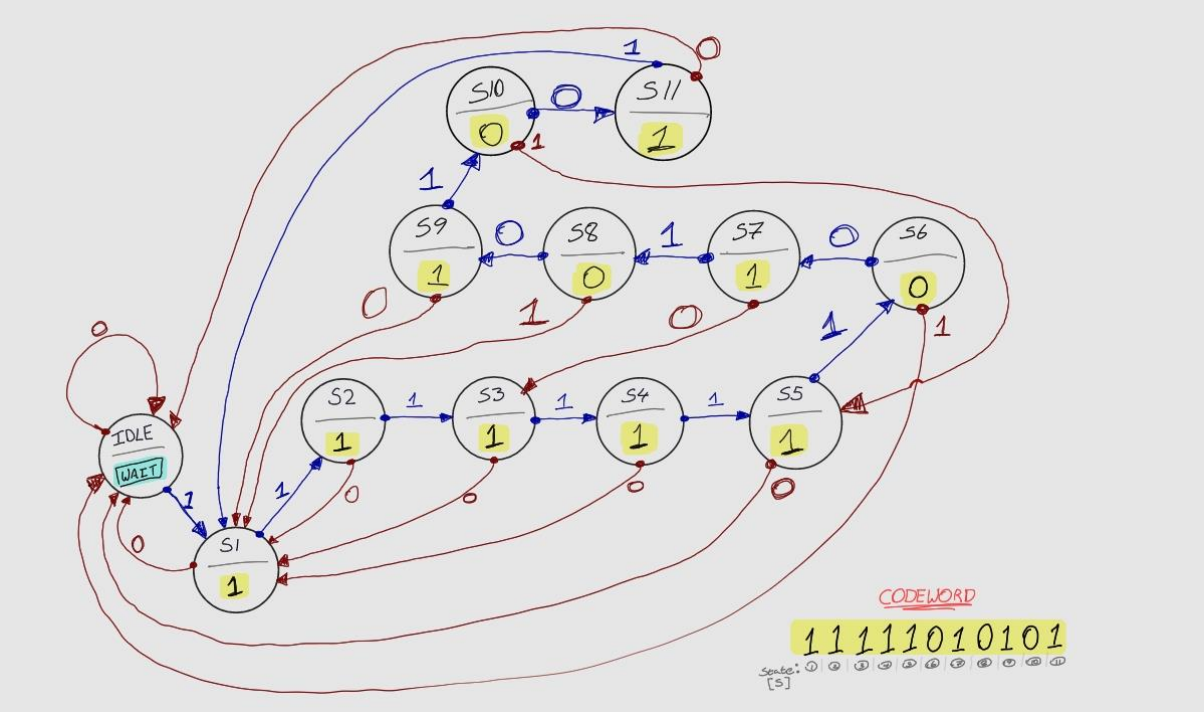


Figure 4.) Finite State Machine Diagram to find codeword "11111010101"

In the FSM a state (S) was given to each of the 11 bits that would make up the codeword going from 'S1' up to 'S11'. A preliminary idle state was also introduced for when there was no expected value of the next bit. As mentioned before the MSB of the LFSRs output (lfsr_output) would be used as an input in the FSM. The signal to detect when the maximum number of cycles has been reached (max_tick_reg) was also used as an input in order to reset the counter after 2^22-1 cycles. As the sequence of MSB values are read into the FSM it undergoes the following process in order to detect the codeword:

Transition from IDLE to S1:

- Initially, the FSM is in the IDLE state (curr_state = IDLE).
- When the first bit from the LFSR (lfsr_output) is detected, the FSM checks if it's a 1.
- If lfsr_output is 1, the FSM transitions to state S1 to start the sequence detection process.
- If lfsr_output is 0, the FSM remains in the IDLE state, waiting for the beginning of the sequence.

Transition from S1 to S2:

- In state S1, the FSM expects to see the next bit of the sequence, which is also a 1.
- If the next bit from the LFSR (lfsr_output) is 1, the FSM transitions to state S2.
- If lfsr_output is 0, indicating an incorrect bit, the FSM resets to the IDLE state.

Transition from S2 to S3:

- In state S2, the FSM expects to see another 1, forming the sequence 11.
- If lfsr_output is 1, the FSM transitions to state S3.
- If lfsr_output is 0, indicating an incorrect bit, the FSM resets to the IDLE state.

Transition from S3 to S4:

- In state S3, the FSM expects to see another 1, forming the sequence 111.
- If lfsr_output is 1, the FSM transitions to state S4.
- If lfsr_output is 0, indicating an incorrect bit, the FSM resets to the IDLE state.

Transition from S4 to S5:

- In state S2, the FSM expects to see another 1, forming the sequence 11.
- If lfsr_output is 1, the FSM transitions to state S3.
- If lfsr_output is 0, indicating an incorrect bit, the FSM resets to the IDLE state.

Transition from S5 to S6:

- In state S5, the FSM expects to see another 1, forming the sequence 11111.
- If lfsr_output is 1, the FSM transitions to state S6.
- If lfsr_output is 0, indicating an incorrect bit, the FSM resets to the IDLE state.

Transition from S6 to S7:

- In state S6, the FSM expects to see a 0, forming the sequence 111110.
- If lfsr_output is 0, the FSM transitions to state S7.
- If lfsr_output is 1, indicating an incorrect bit, the FSM resets to the IDLE state.

Transition from S7 to S8:

- In state S7, the FSM expects to see another 1, forming the sequence 1111101.
- If lfsr_output is 1, the FSM transitions to state S8.
- If lfsr_output is 0, indicating an incorrect bit, the FSM resets to the IDLE state.

Transition from S8 to S9:

- In state S8, the FSM expects to see a 0, forming the sequence 11111010.
- If lfsr_output is 0, the FSM transitions to state S9.
- If lfsr_output is 1, indicating an incorrect bit, the FSM resets to the IDLE state.

Transition from S9 to S10:

- In state S9, the FSM expects to see another 1, forming the sequence 111110101.
- If lfsr_output is 1, the FSM transitions to state S10.
- If lfsr_output is 0, indicating an incorrect bit, the FSM resets to the IDLE state.

Transition from S10 to S11:

- In state S10, the FSM expects to see a 0, forming the complete sequence 1111101010.
- If lfsr_output is 0, the FSM transitions to state S11, indicating the successful detection of the entire codeword.
- If lfsr_output is 1, indicating an incorrect bit, the FSM resets to the IDLE state.

Once the state S11 has been reached, the FSM sets an output signal, ‘CW_detected’ high, indicating the successful detection of the codeword. It then increments an output integer variable, ‘counter’ to track the current number of successful detections. Afterwards, the FSM resets to the IDLE state to begin detecting the sequence again in subsequent MSB input bit streams.

A top module aptly named ‘TopModule.v’ was designed to allow the concurrent operation of the LFSR and FSM modules. A testbench for the top module was also created. The following figure shows the waveform produced from the testbench:

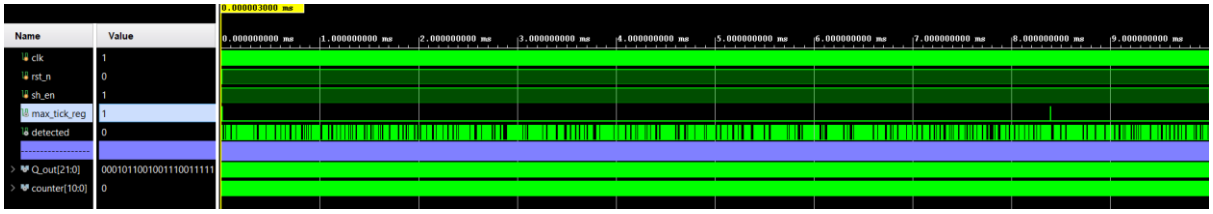


Figure 5.) TopModule Waveform (Full)

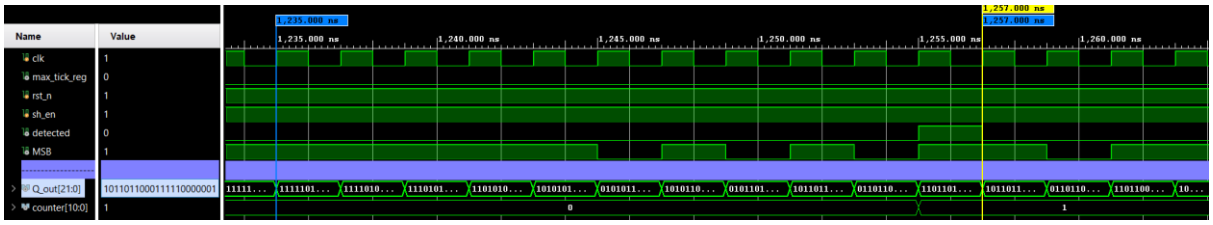


Figure 5.1.) Top Module Waveform (Zoomed In at First Detected Signal)

As can be seen in the Figure 5.1 when each MSB of a sequence of eleven LFSR outputs match the codeword (11111010101) the state of the ‘detected’ signal is briefly set high to indicate that it indeed detects the codeword.

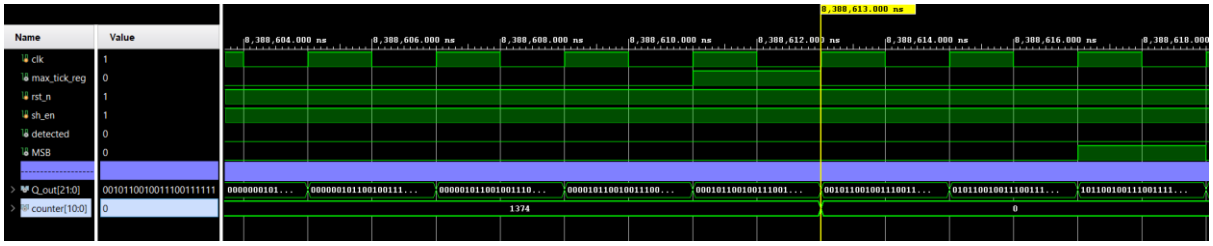


Figure 5.2.) TopModule Waveform (Zoomed In at Max Tick)

Just as expected, when the maximum cycle is reached the counter resets to zero. From this, it can be seen that the codeword is detected a total of 1374 times.

Getting the Counter on the Seven Segment display

With a functional counter now present in the design, there still needed to be a way to display the number of the counter on the board. For this reason, a ‘disp_hex_mux’ module provided via blackboard in a previous lab to initialise functionality of the boards seven segment display outputting a 4-bit ‘an’ and an 8-bit ‘sseg’ array in the process, and ‘Ctr_SSeg’ module to convert the integer counter values into the necessary hex digits needed to be read by the seven-segment display, were added and instantiated in the top module. Figure 6 shows the current state of the module hierarchy with these new additions:

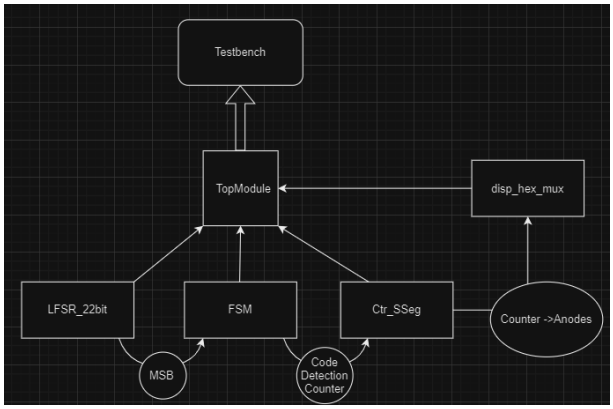


Figure 6) Updated System Hierarchy Plan to Account for Counter and 7-Segment Display

Making a Separate LSFR for MSB Stream

After the synthesis process and while placing the I/O ports to eventually target the board, it became evident that the same approach used for Lab F in targeting the current system to the board would not work as there weren't enough on-board LEDs to display each bit in the 22-bit LFSR output. To remedy this a separate 11-bit LSFR module named 'MSB_lfsr_Board' was designed to display the MSB stream in real-time. Figure 7 illustrates how the two LFSR modules are intended to interoperate:

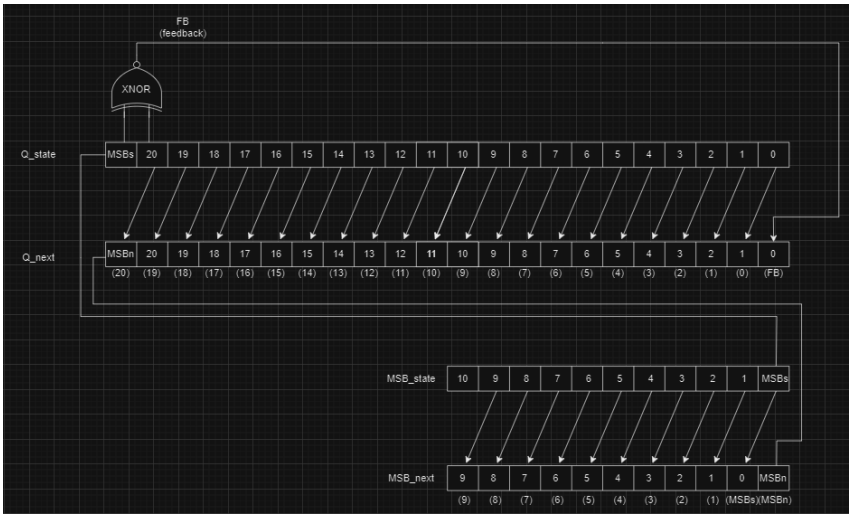


Figure 7.) Block diagram illustrating how 22-bit LFSR and MSB LFSR are to connect

Like the FSM, the MSB of the LFSR output was used as an input and played a similar role to the LFSR as the feedback (FB) bit did, where during each clock pulse, the MSB of the 22-bit LFSR output is taken and registered as the least significant bit in a current array 'MSB_state[10:0]', which in this case is initially set to all zeroes (11b'0). In the subsequent array 'MSB_next[10:0]', also initially set to 11b'0, all bits are shifted one position towards the left, with the leftmost bit being truncated out of the array. The next 'MSB' bit is then placed in the least significant bit position, updating the LFSR array to become the new 'MSB_state[10:0]'. This 'MSB_state[10:0]' is then assigned to an output array of equal length 'MSB_out[10:0]' to be read in the waveformtaken as an output. As it is possible for this cycle to continue indefinitely,'max_tick_reg' is used to signal when to reset 'MSB_state[10:0]' and 'MSB_next[10:0]' to 11b'0 once the maximum cycle has been reached. Figure 8 bellow shows the waveform produced:

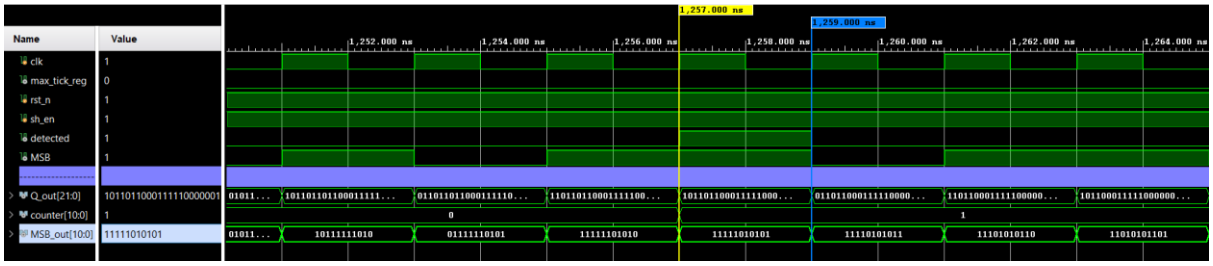


Figure 8) Topmodule Waveform (Zoomed In To Show How MSB_Out Displays Codeword When Detected Is High)

As can be seen when the 'detected' signal goes to a high state, the codeword is displayed by 'MSB_out[10:0]', further validating the functionality of the FSM. Now that a way to see the 'MSB' stream was developed and fully operational, the synthesis and implementation of the design to the Basys3 FPGA board could be commenced.

Synthesis and Implementation

As there were too many output signals to place in unique ports, specifically due to the 22bit LFSR output and the 11bit counter output, a separate top module 'TopModBoard' was created for the specific purpose commenting out these outputs and leaving those that needed to be displayed on the board or were required for the systems functionality. 'TopModBoard' also saw the addition of a clock module provided via blackboard to scale the clock pulses so each pulse can clearly be viewed on the board. The old top module was renamed 'TopModWaveform' and was repurposed as a means to view the waveforms previously shown. The usual steps were taken in preforming the synthesis, leading to following synthesised design being generated:

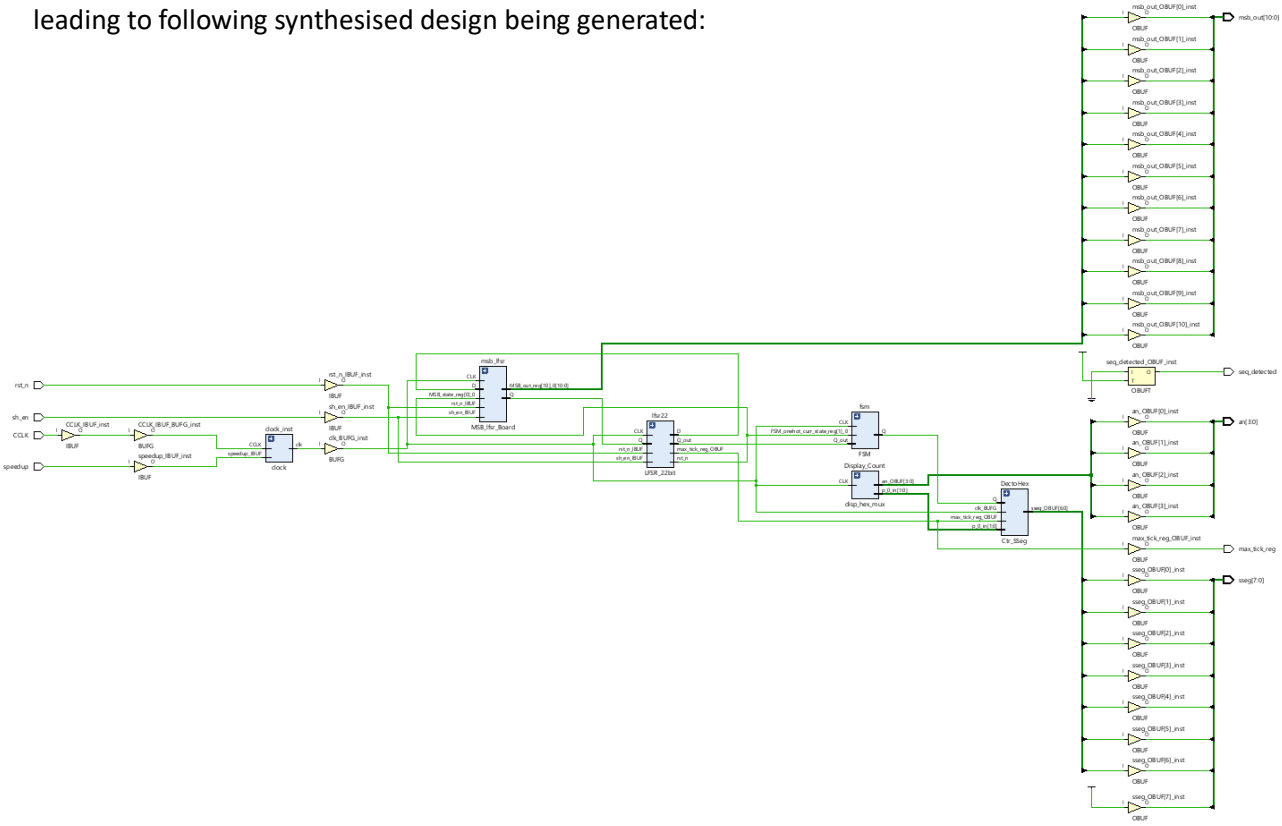


Figure 10.) Synthesised Design of System

After synthesis the ports were manually configured using the I/O port configuration tool. The ports were appropriately placed to bring it in line with the inputs and outputs specified in the design. Following this, the design was implemented in order to generate the bitstream necessary for programming the Basys-3 Board. Once programmed the design can be demonstrated on the board.

Demonstration (Port Setup)

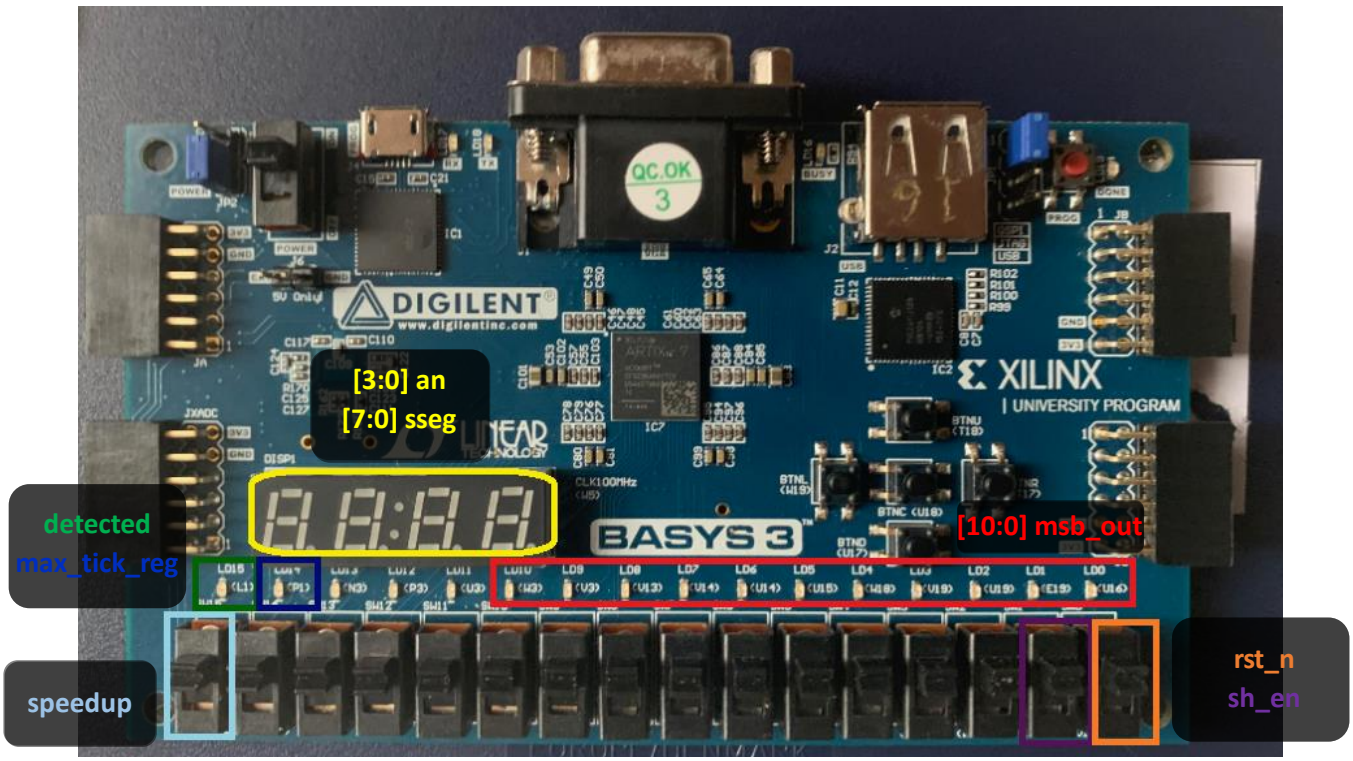


Figure 11.) Board showing specific switches and LEDs used to control/display input and output signals

(Note: this is w.r.t pin values being read from right to left [i.e. (V17 –> V15) = V17, V16, W16, W17, W15, V15])

Switches

The board was successfully programmed and worked as intended however the rate at which the MSB stream was outputting to the board was unsatisfactory. To resolve this, logic was added to change the 32bit 'clkscale' signal from its current value (20000000) to a smaller value (5000000) depending on if a newly added input 'speedup' was set to high. This 'speedup' input was then set to the leftmost switch on the board R2. The two rightmost switches on the board, V17 and V16 controlled the reset (rst_n) and shift enabler(sh_en) inputs respectively.

LEDs

The 11 LEDs on the right (U16 -> W3) were used to visually represent each bit in the 'MSB_LFSR' output (msb_out[n] ~ where n ranges from 0 to 10). These LED would display the bit stream in real-time and the codeword in its entirety once detected. The two leftmost LEDs on the board, L1 and P1 are used to depict the when the codeword has been detected and when the maximum LFSR cycle has been reached, respectively.

Seven Segment Display

As mentioned before a 4-bit ‘an’ and 8-bit ‘seg’ array were to be outputted to the boards seven segment display, therefore each signal was assigned to the corresponding port in order to get the display to show the codeword detection count.

Demonstration (Testing)

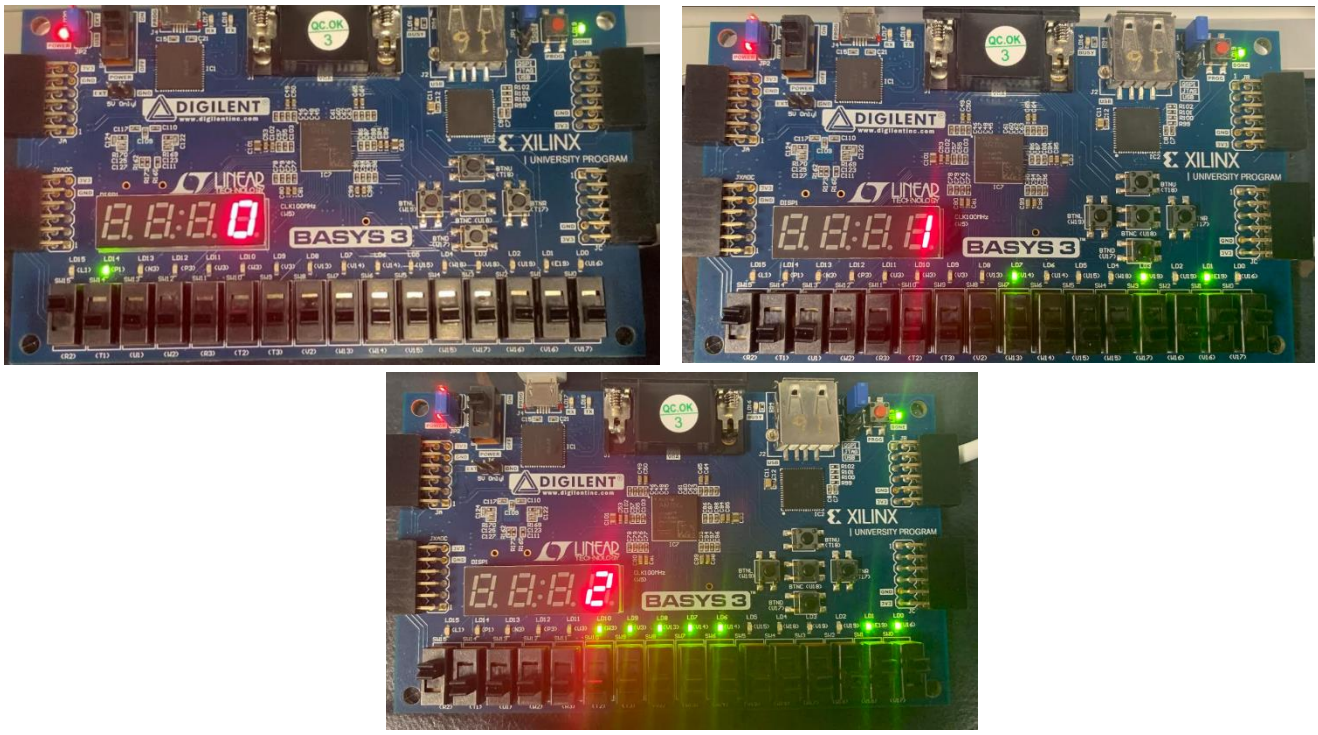


Figure 12. [0-2]) Photos Taken of The Board In Operation

As the process of detecting the codeword in the bitstream using the board was autonomous, only requiring inputs to start the process, reset the process and change the speed, specific test vectors could not be made. However, a test was performed with the system to see how many times the codeword would be detected by the FSM in an hour. In the hour the codeword had only been detected 9 times.

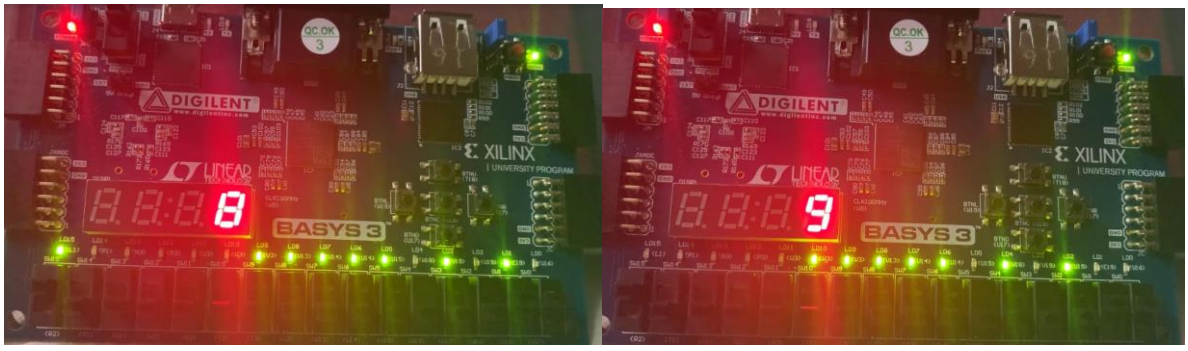


Figure 13.) counter incrementing as the 9th codeword detection has occurred

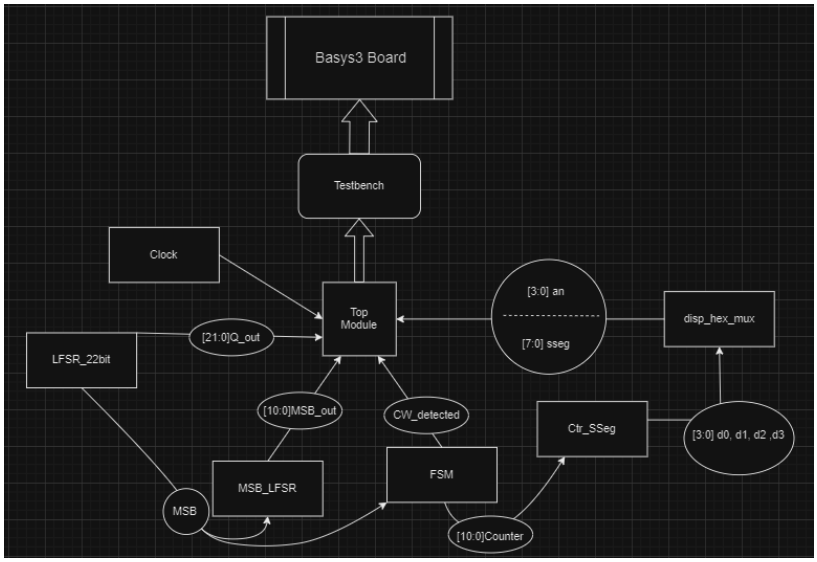
This is less than a fraction of the total amount of detections counted while using the waveforms which was recorded to be 1374 times. By that logic It would take roughly 152 hours (1374/9) until

the maximum number of detections was reached, therefore I concluded that it was infeasible to try and record every detection.

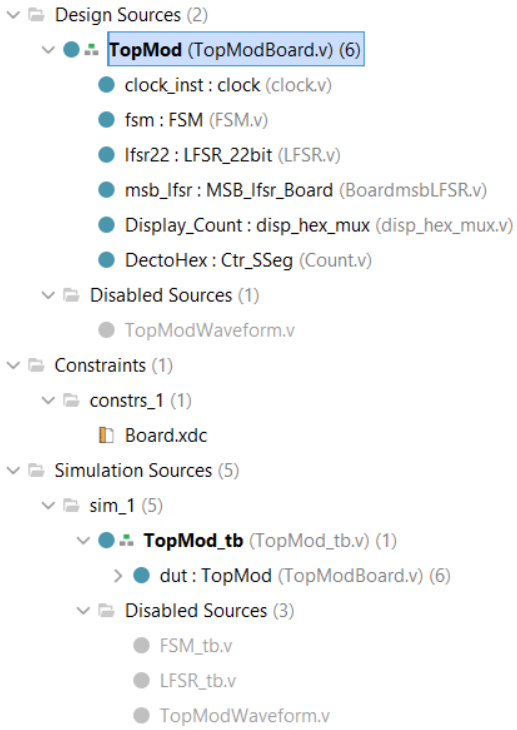
Appendix

- All code mentioned in this report can be found within the submitted zip file.
- A ten-minute recording of the hour-long test capturing the first six detections is included in the submitted zip file. (full video is >8GB so it can't be included)

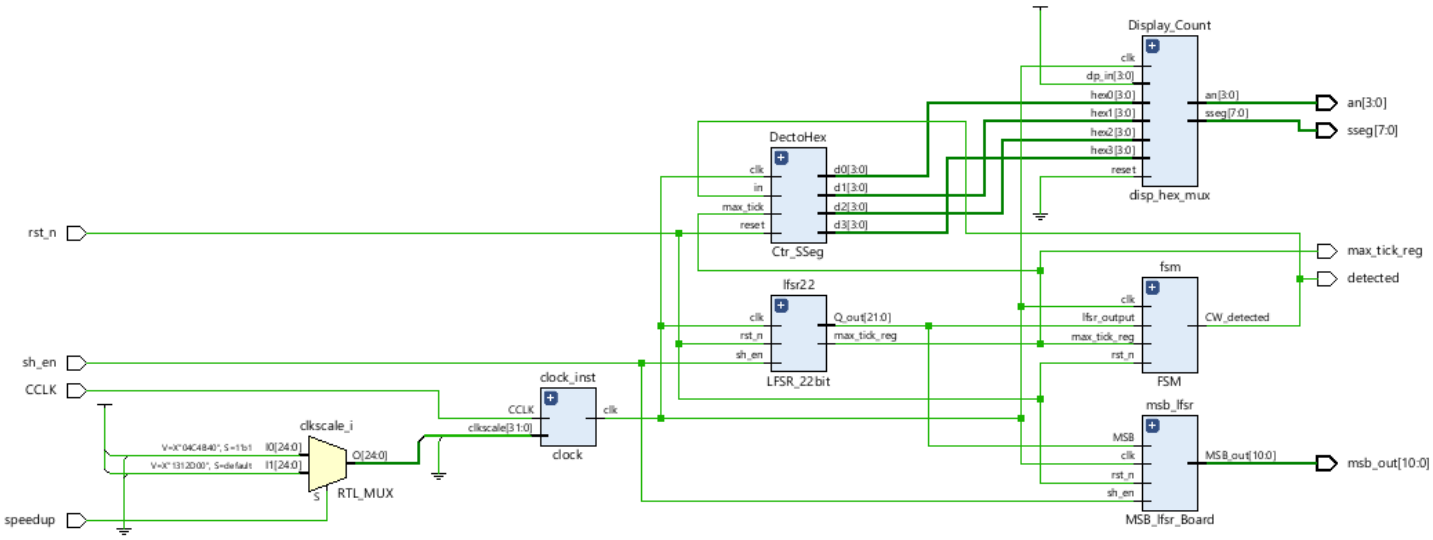
- ## Finalised hierarchy plan block diagram



- ## Vivado Hierarchy

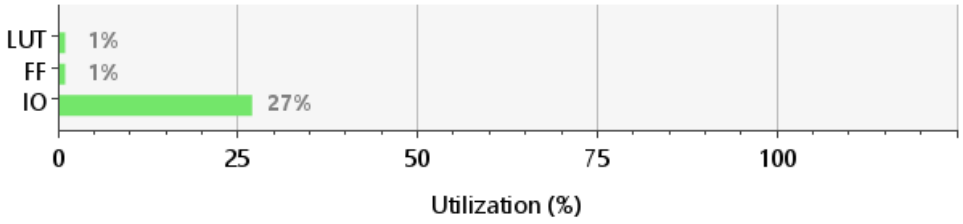


- **Vivado - Elaborate Design**



- **Vivado – Utilisation Report**

Resource	Utilization	Available	Utilization %
LUT	79	20800	0.38
FF	158	41600	0.38
IO	29	106	27.36



Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)	BUFGCTRL (32)
TopMod	79	158	62	79	29	2
clock_inst (clock)	18	33	21	18	0	0
DectoHex (Ctr_SSeg)	25	16	8	25	0	0
Display_Count (disp_hex_mux)	3	18	7	3	0	0
fsm (FSM)	7	13	3	7	0	0
lfsr22 (LFSR_22bit)	21	56	20	21	0	0
msb_lfsr (MSB_lfsr_Board)	5	22	10	5	0	0

- **Vivado - Implemented Package Design**