Digital System Design Lab F

Ruairi Mullally – 22336002

Introduction:
This report examines the design and implementation of multiple synchronous/sequential Verilog modules. A D-type flip flop (DFF) was tested and then used to implement an 8-bit storage register. A linear feedback shift register (LFSR) was also implemented and tested for maximal length, and later coupled with a clock divider to display the output on the Basys3 LEDs.

Capture the waveform from the functional simulation of DFF and LFSR and the synthesis/implementation results of targeting to board. Capture the block diagram of the LFSR design into the short report with an explanation of your design process and choice of test vectors. Submit the short report in pdf form via the link in Blackboard. Additionally, must submit the following items via Blackboard in a single zip file

Testing of DFF:

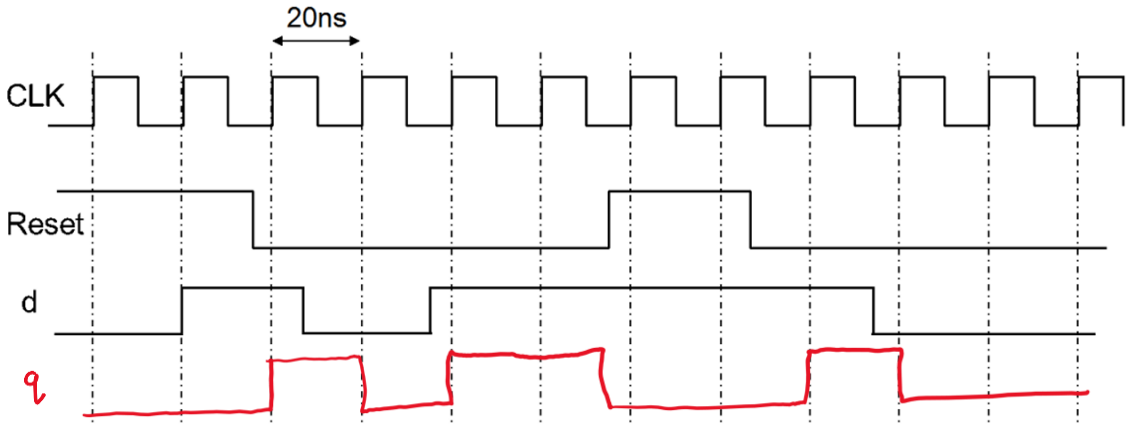

Figure 1. Test setup for DFF, expected output q for asynchronous reset (posedge clock).

Test 1: Rising edge clock trigger, with asynchronous reset.

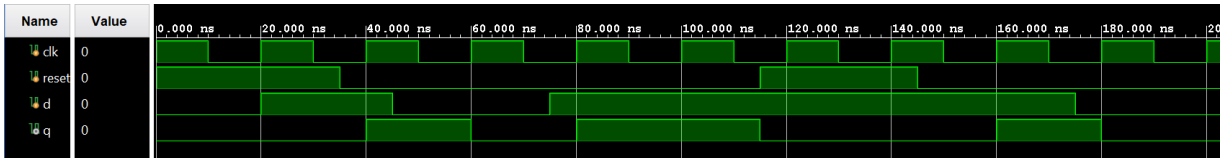

Figure 2. Test results for posedge, async reset.

The output q from the testbench matches the expected output, where d sets q on rising clock edges, but reset can operate asynchronously and reset q to 0 independent of the clock.

Test 2: Falling edge clock trigger, with asynchronous reset.

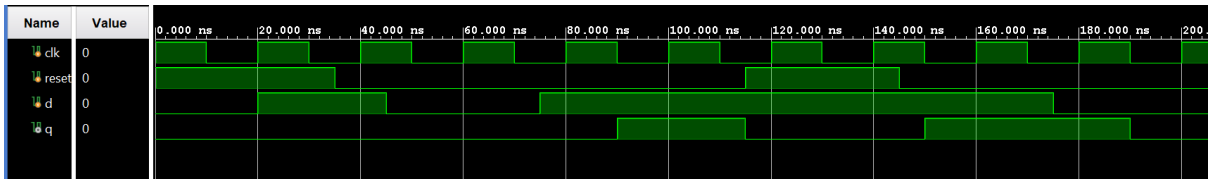

Figure 3. Test results for negedge, async reset.

The DFF behaviour is the same as before except d now sets q on the falling edge of the clock rather than the rising. It can be seen in the waveform that a reset overrides incoming data, which is expected. Reset still functions independently of clock.

Test 3: Falling edge clock trigger, with synchronous reset.

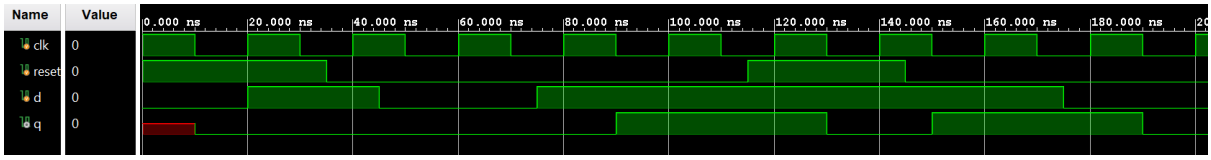

Figure 3. Test results for negedge, synchronous reset.

With reset being synchronous, q can only be set on falling edges of the clock. Reset is no longer independent of the clock. This introduces the state where q is undefined at the beginning until a reset can be completed. The output q is different when comparing the synchronous and asynchronous resets.

Using an 8-bit DFF chain to save a value (Part B):

Using a serries of DFFs a temperature value between 0 - 255 is stored and updated synchronously from button input on the Basys3 board. These buttons are debounced. The saved value is displayed on the built in 7-segment display of the Basys3.

```
//Q_next is the next value that will be stored in 8bit reg
//compute Q_next based off Q and debounced button input
reg[7:0] Q_next;
wire[7:0] Q;//for accessing saved data
always @(*)begin
    if(buttons[0] | buttons[3])begin
        Q_next <= Q + 1;
    end
    else if(buttons[2] | buttons[1]) begin
        Q_next <= Q - 1;
    end
    else if(buttons[4]) begin
        Q_next <= 8'h16;
    end
    else begin
        Q_next <= Q;
    end
end
```

Figure 4. Logic for computing next storage value based on current stored value and button input.

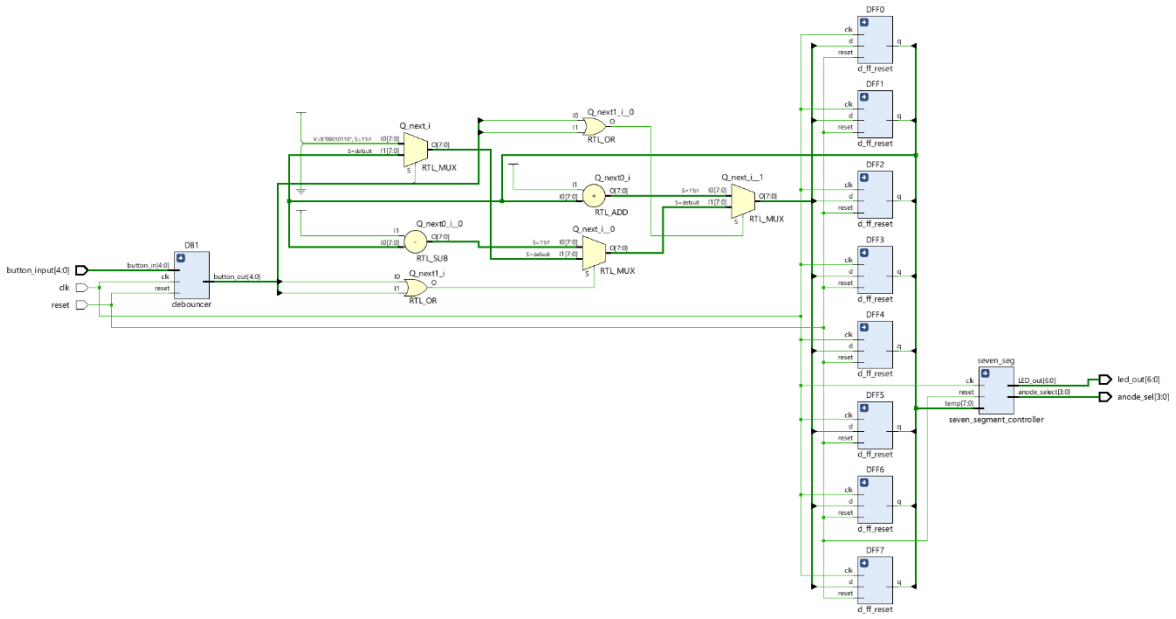

Figure 5. Diagram of temperature counter top module.

The design process began with the testing and verification of the DFF under synchronous and asynchronous conditions with falling and rising edge triggers. These initial tests, shown in figures 1 – 3, confirmed the bahviour of the DFF. Building upon the DFF, an 8 bit storage register was designed for temperature value storage. This was achieved using a chain of DFFs, with additional logic blocks for incrementing and decrementing the value, debouncing button input, and writing to the 7-segment displays. The expected bahviour was empirically verified.

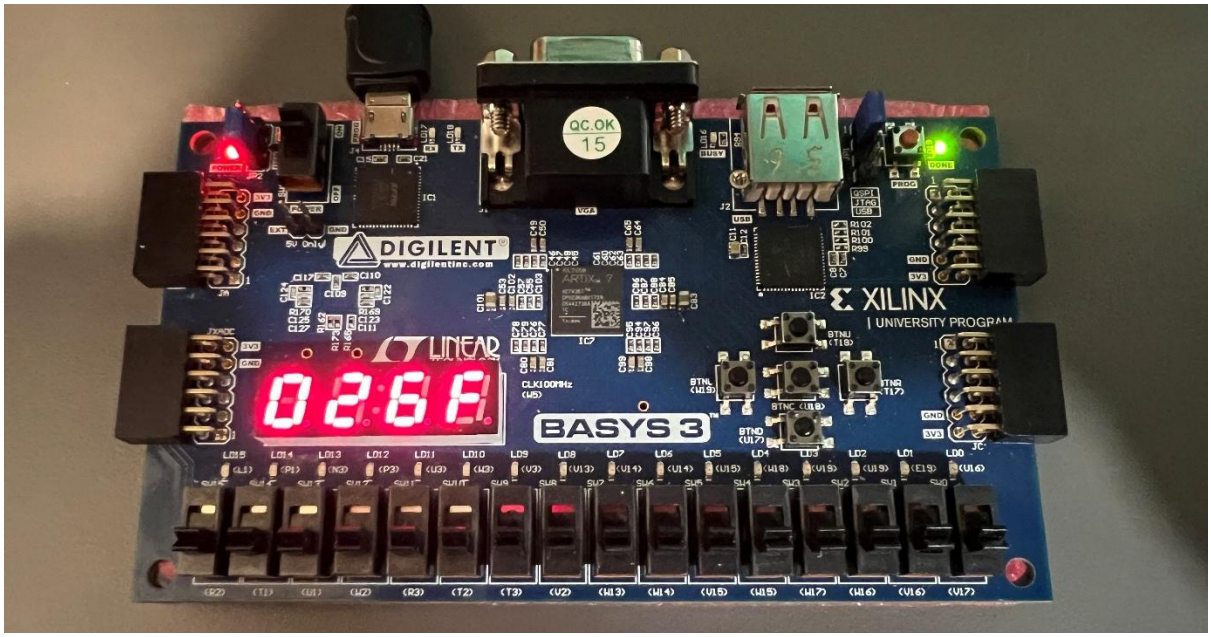

Figure 6. Temperature display and storage on board.

Button input is debounced. If buttons UP (0) or RIGHT (1) are pressed, the stored value is increased by 1. If buttons DOWN (2) or LEFT (3) are pressed, the stored value is decreased by 1. If CENTRE is pressed (4), the stored value is reset to 22. The 7-segment display displays the currently stored value.

The temperature counter operated as expected. The storage was reset when SW0 was flipped, and it reset to 22 when CENTRE was pressed. The counter increased and decreased appropriately when the buttons were pressed. The buttons had to be held for a bit longer than expected to affect change, though this could be due to the reduced polling speed of the buttons with the debouncing module.

Resource	Utilization	Available	Utilization %
LUT	90	20800	0.43
FF	70	41600	0.17
IO	18	106	16.98

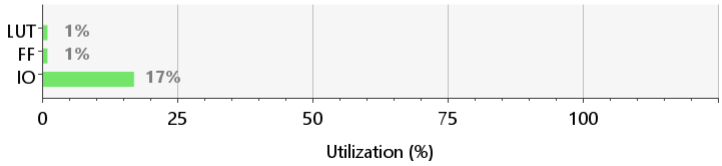

Figure 7. Utilisation report for part B.

LFSR Implementation (Part C):

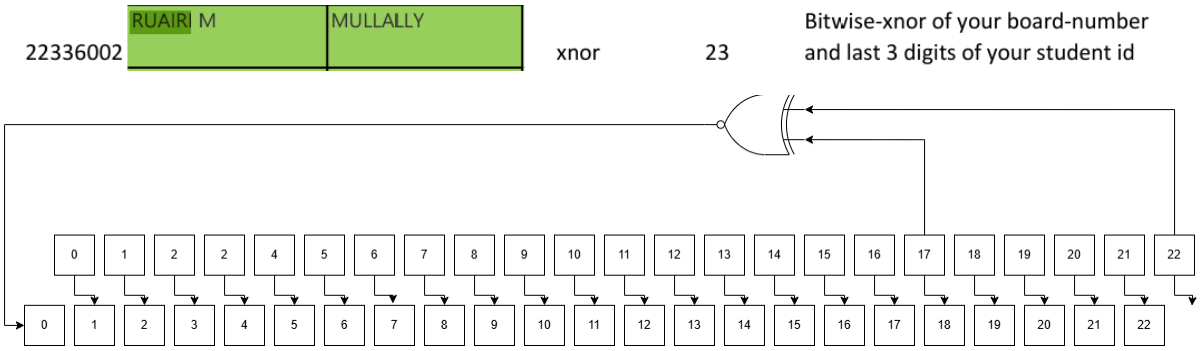

Figure 8. Maximal length tap selection.

To design an efficient LFSR, a maximal length design is needed. For a tap length of 23, LFSR stages 23 and 18 should be XNORed for maximal length. This represents bits 22, and 17. The choice of these taps ensures all 2^{23}-1 states will be reached. Test vectors were chosen to validate several aspects of the LFSR:

- Reset functionality: the LFSR was initialized with a known seed, and also reset mid-cycle to verify functionality (figures 9 and 11).
- Shift enable: The shift enable signal was toggled to confirm the shift register hold its state when disabled.
- Cycle completion: the simulation was ran to capture a full cycle of the LFSR to confirm it return to its seed (figure 10).

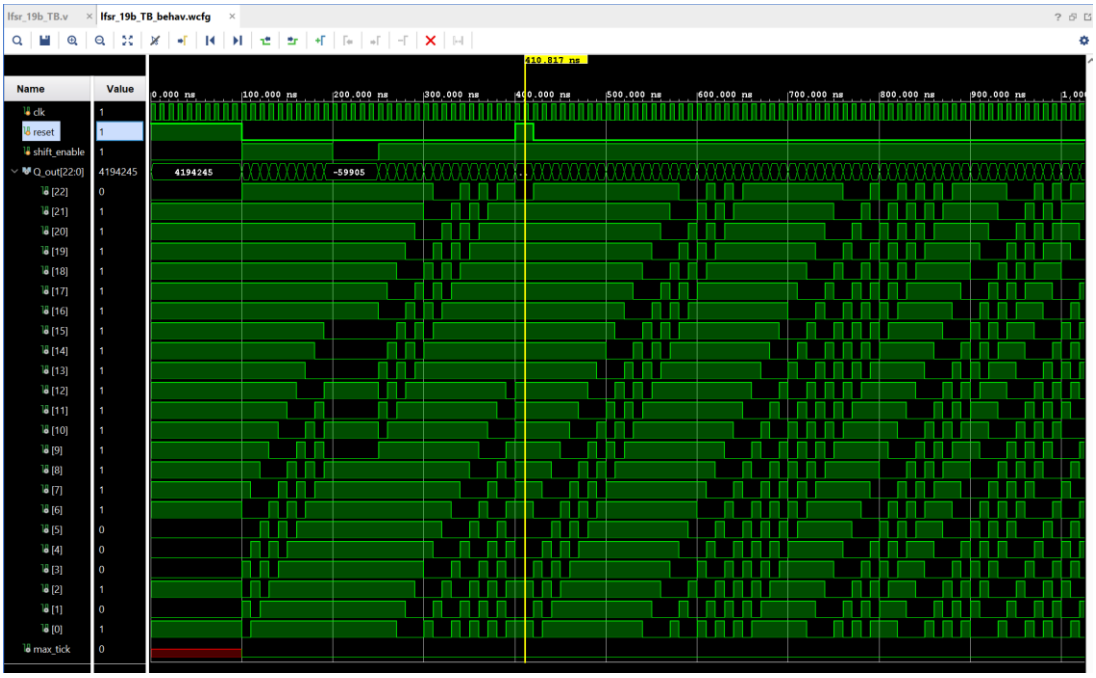

Figure 9. LFSR testbench showing reset, shift enable functionality.

In figure 9, one can see the shifting properties of Q_out as the bits ripple along over time. The correct operation or reset is verified, by using it to set the seed at the beginning and mid-cycle. The correct operation of shift enable can be seen as the bit shifting halts when shift enable is low.

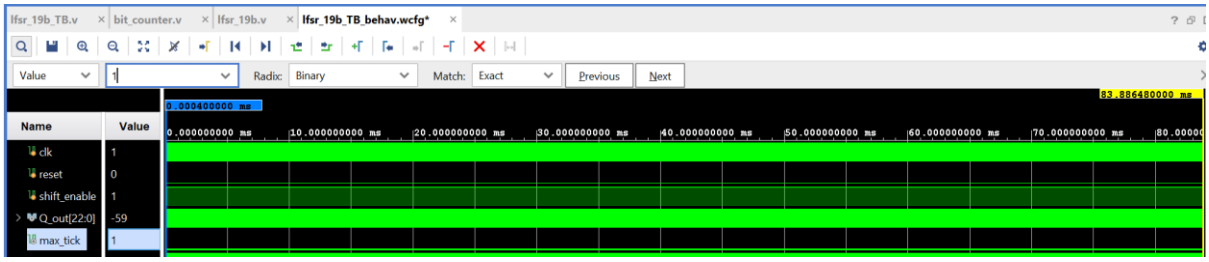

Figure 10. Time to cycle back to seed.

The cycles needed to get back to the seed is 2^{23}-1 = 8,388,607. At a clock speed of 100Mhz, this represents 83,886,070ns, or 83.88607ms. This is verified in figure 10.

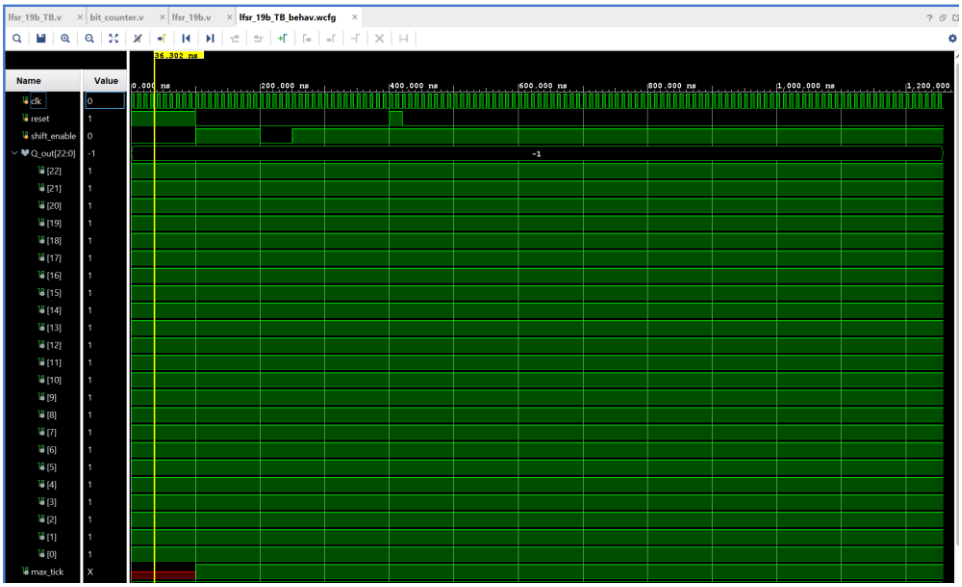

Figure 11. Seed of all 1s locks LFSR.

Forbidden seed values are seeds that would eventually result in all 1s (in the case of XNOR) in the storage register, this would mean that the LFSR would lock up and stop producing new inputs. Because this LFSR has a maximal polynomial, the only value that would result in this case is a seed of all 1s.

Counting:

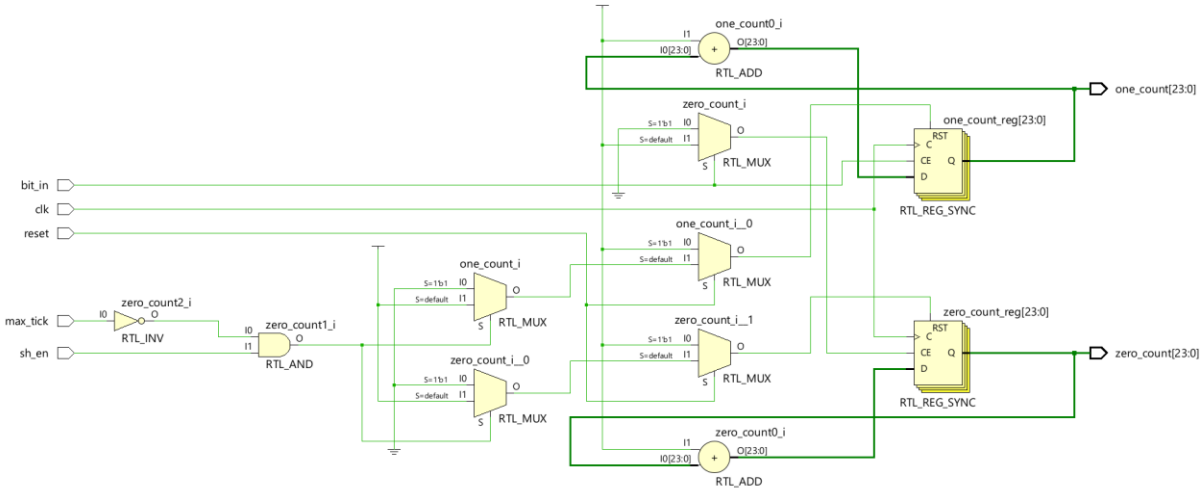

Figure 12. Schematic of counter module.

The counting module needed registers that were large enough to count the total number of 1s and 0s, and hence were set to 24 bits. Additional logic was needed to decide when to increment the counters. They needed to be reset when max tick was reached, as well as when reset was enabled. The counters should also not increment when shift enable was low to stop them from counting the same input bit multiple times. The counter implementation was validated by confirming that for the maximal-length sequence, the number of 1s was one more than the number of 0s (figures 13 – 15). Figure 12 illustrates the implemented design.

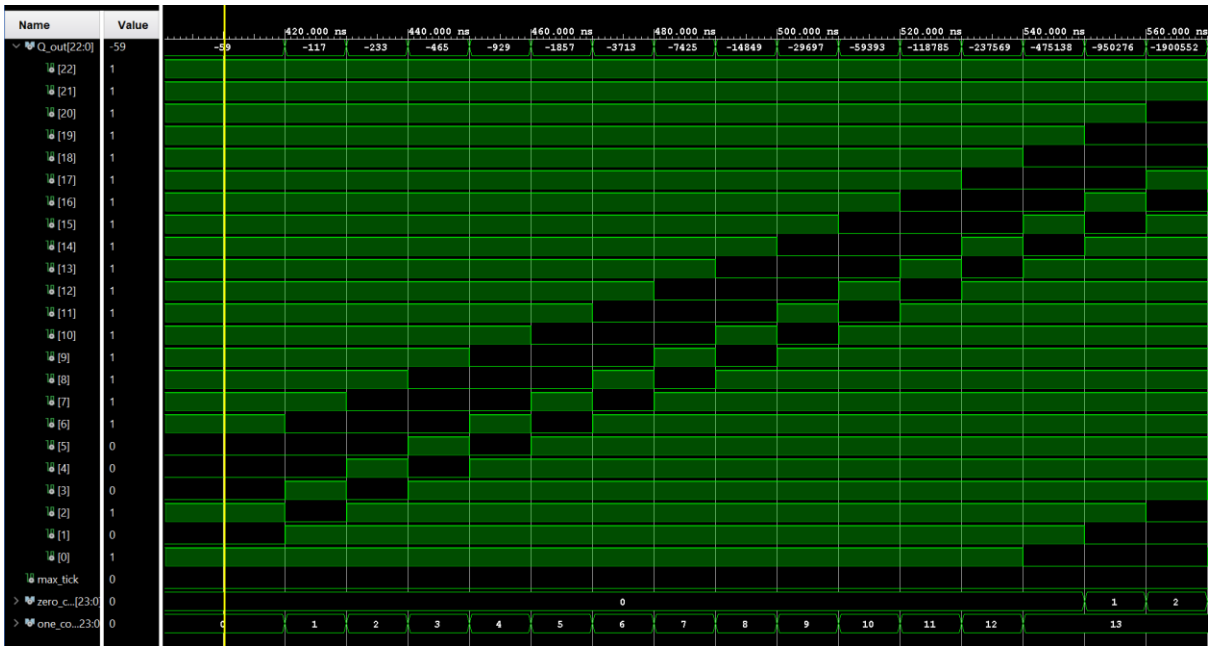

Figure 13. Counter counting the occurance of 1s and 0s.

In figure 13, the counter counting 1s and 0s is demonstrated.

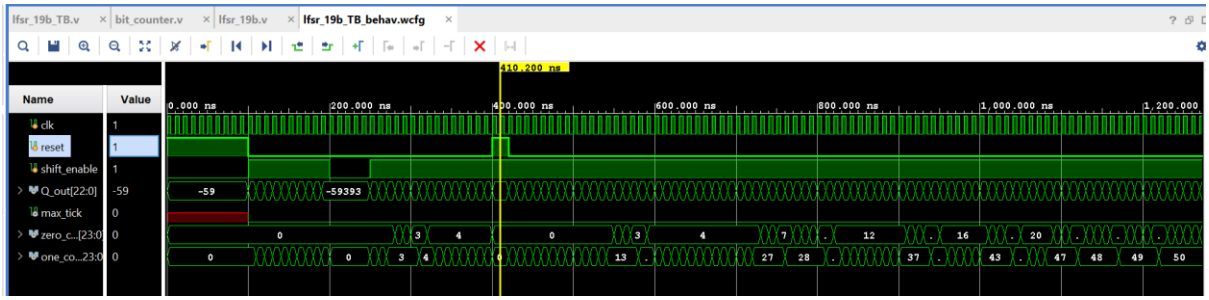

Figure 14. Counting restarting after reset.

In figure 14, the counters restarting after a reset is demonstrated. It can also be seen that the counters do not count when shift enable is low.

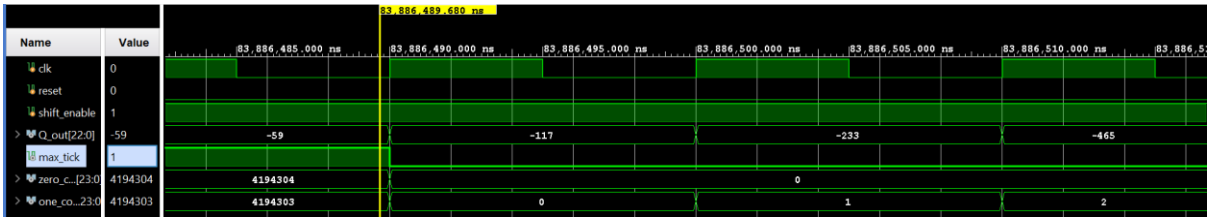

Figure 15. Counting restarting after max tick is reached.

In figure 15 the counters restarting after reaching the max tick is demonstrated. The incidence of 0s is 1 more than 1s, which is the expected behaviour. The sum of their counts is 8,388,607, which is equivalent to the number of cycles to reach the seed, as expected.

Target LFSR to Basys3 (Part D):

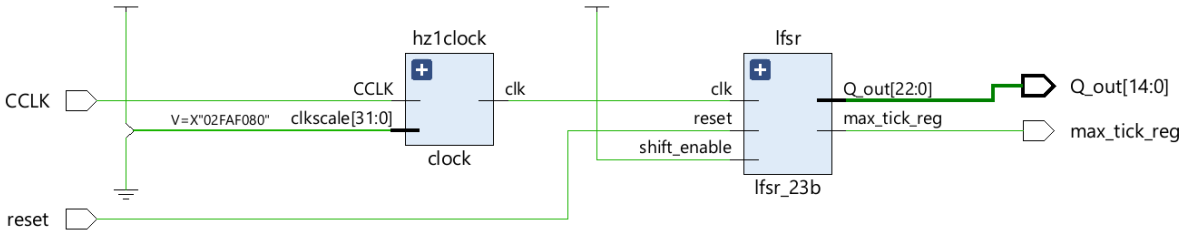

Figure 16. Schematic of clock divider with LFSR.

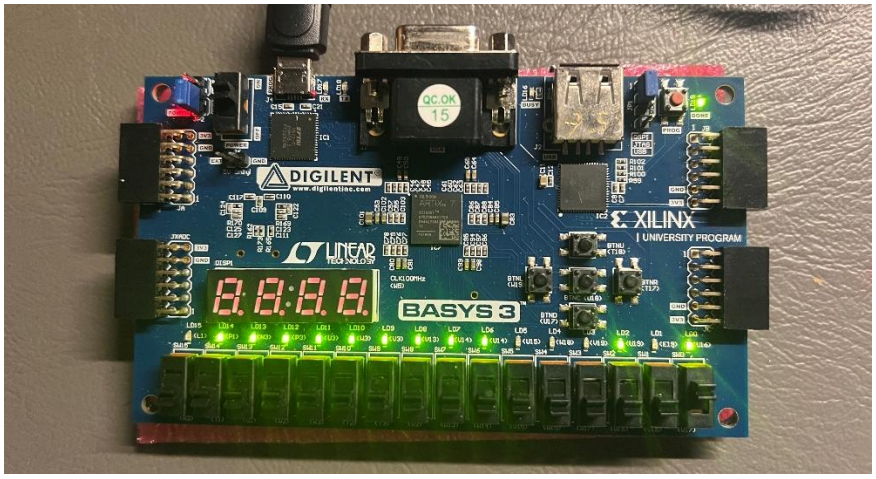

Figure 17. LFSR on board with reset start (displaying seed).

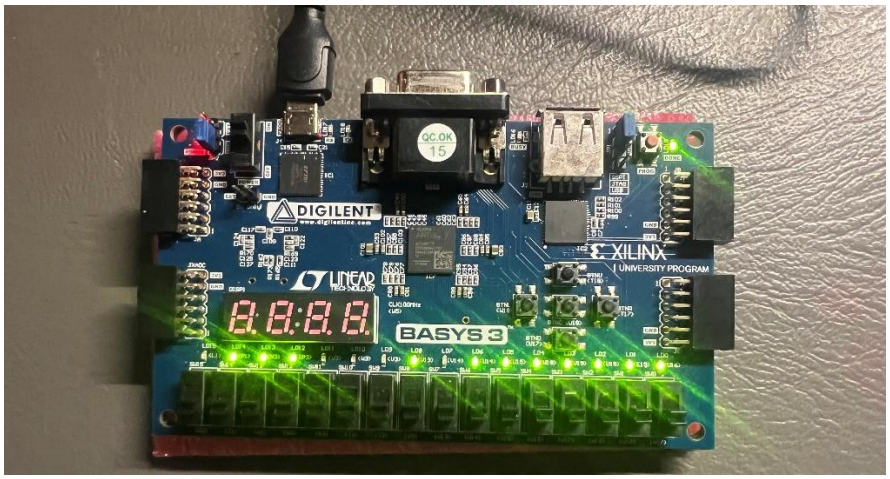

Figure 18. LFSR on board in the process of shifting.

Resource	Utilization	Available	Utilization %
LUT	16	20800	0.08
FF	72	41600	0.17
IO	18	106	16.98

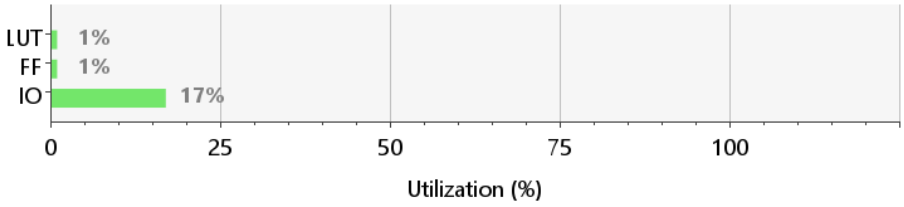

Figure 19. LFSR utilisation report.

Conclusion:

The design process was highly iterative, starting with low level modules, and testing their functionality. In designing modules, features were added one at a time and verified, for example the max tick reset was implemented and validated before the shift enable functionality was added. This approach ensured that the systems (DFF temperature display, LFSR) met the specified performance.