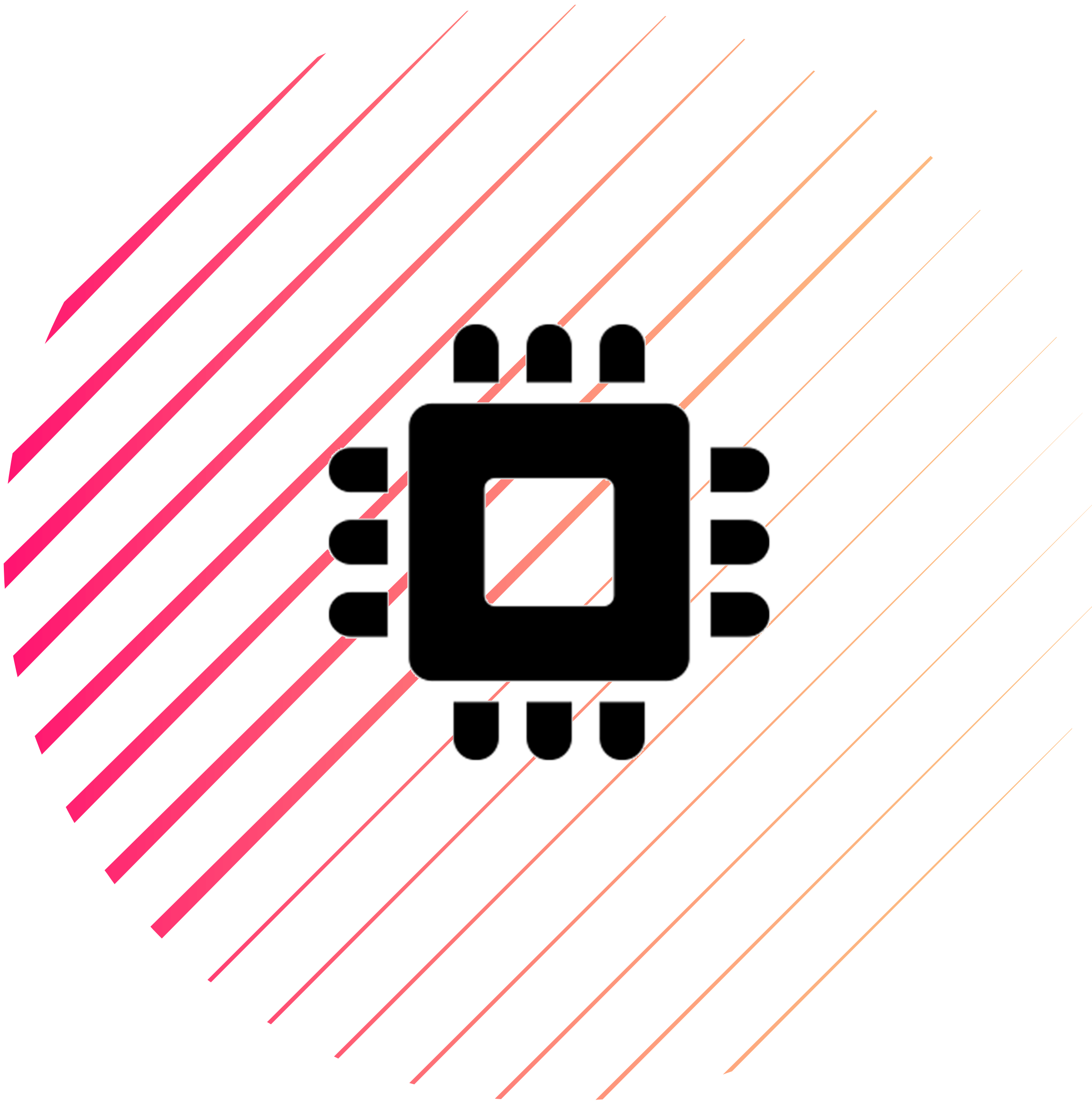
Finite State Machines



Prac 4 Report
CSSE4010

Thulith W Mallawa
S44280422

1. Introduction

1.1 Design Brief and Aims

The aim of this project is to understand and design interacting finite state machines (Mealy/Moore). Specifically, a synchronous sequence detector that will detect the sequence '11001' and hold a detection flag high for 2 clock cycles on the same clock cycle when the last digit is detected.

2. Design Description

2.1 Top Level Design

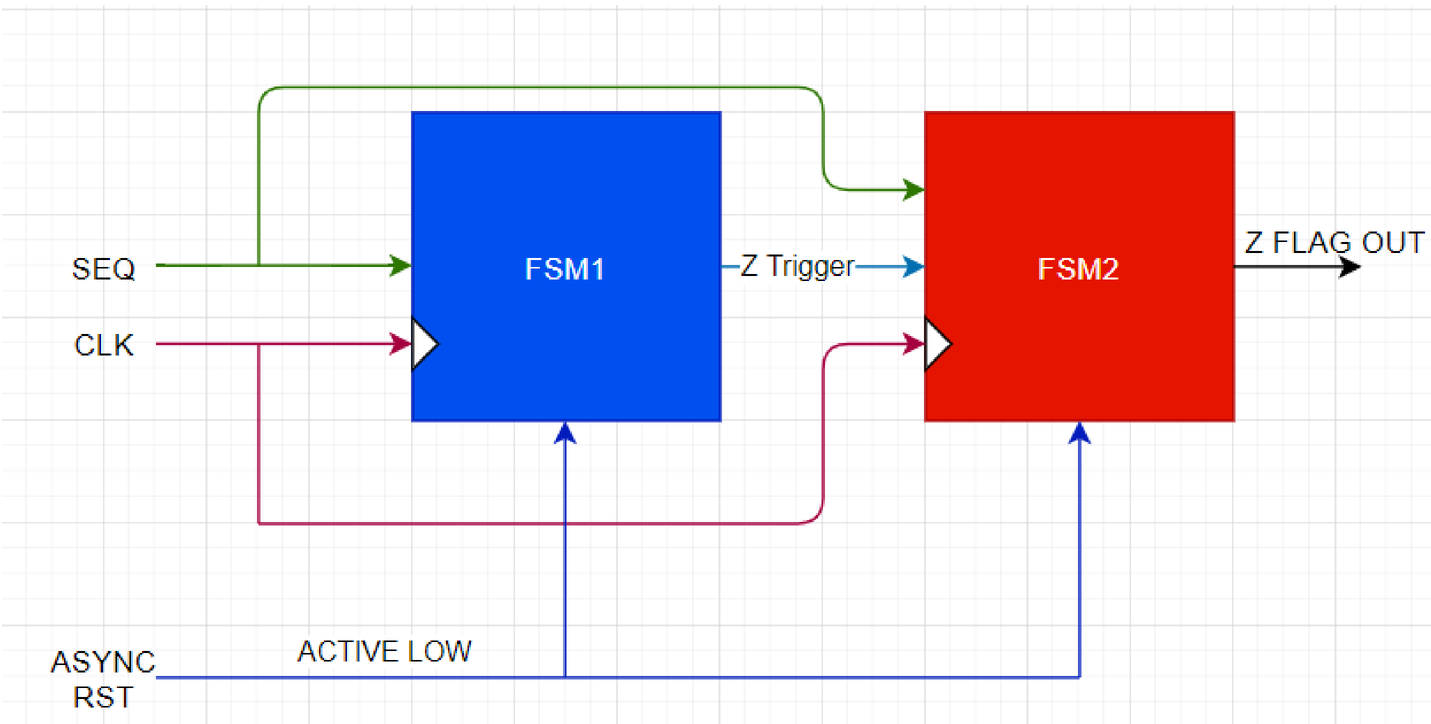


Figure 1 Top Level Sequence Detector

2.2 Design Assumptions

I. The sequence is read right to left, that is, SEQ <= '1', '1', '0', '0', '1' is valid.

II. Given that RST is asynchronous, when it is pressed **(in or outside of clock event),** the state machines will reset back to **the original state on the next clock cycle** (as they are clocked units). **Hence why a 'reset' state is omitted from the following diagrams.**

This design was implemented using two interacting FSMs with a synchronous clock and an active low asynchronous reset (reset occurs outside of CLK Events).

2.3 Design Approach

Design Approach1: Single FSM Mealy

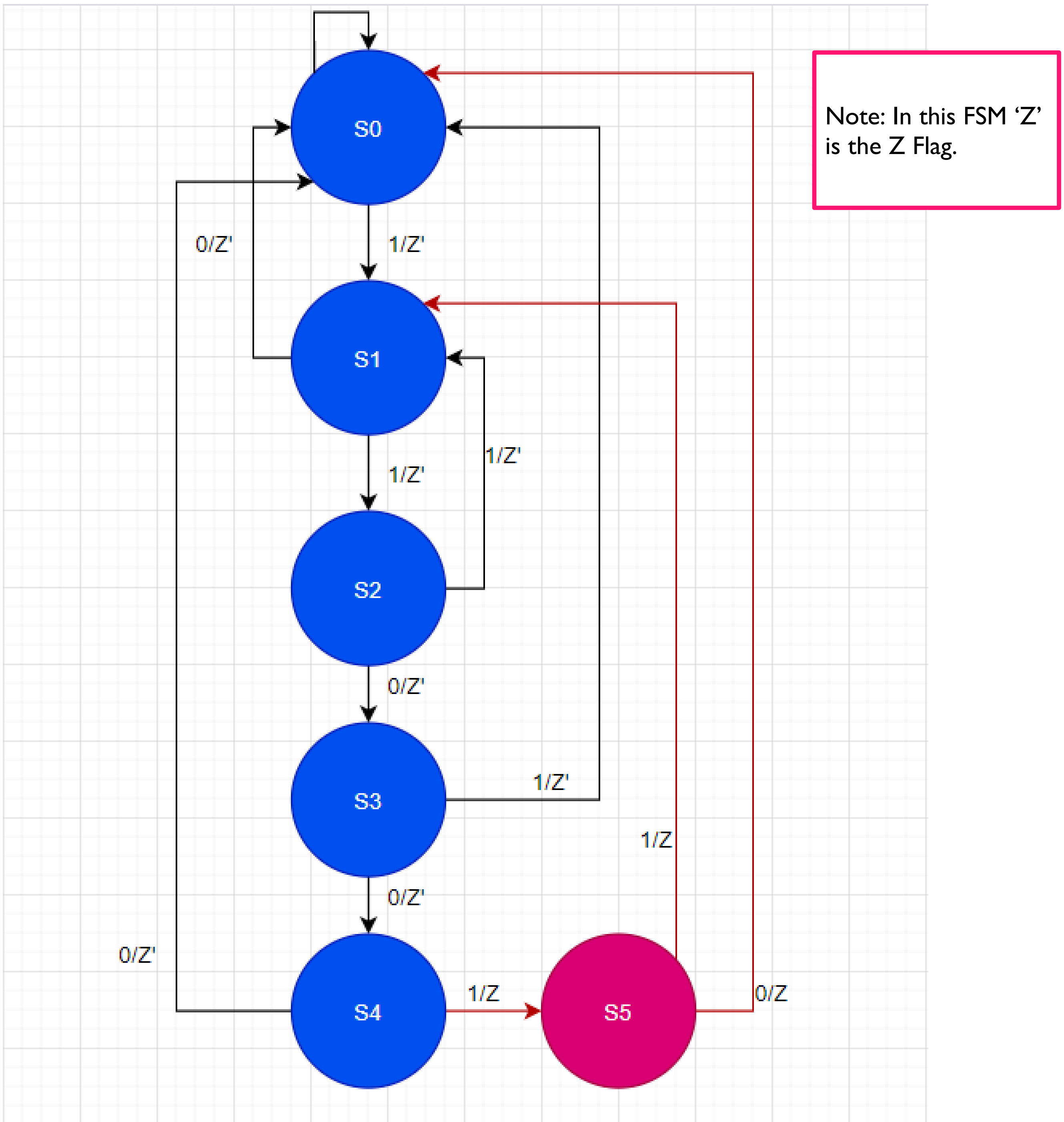
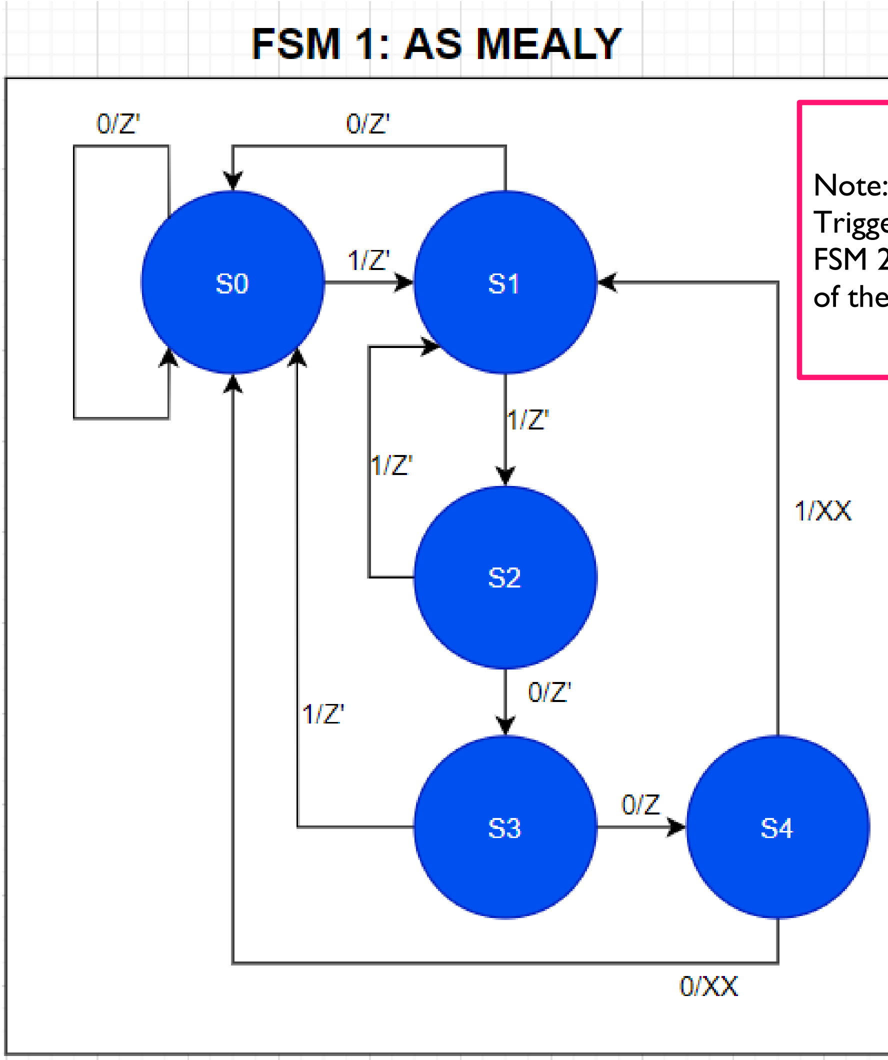


Figure 2 Single FSM State Diagram Mealy Design

Design Approach 2: Two Interacting FSMs



Note: In FSM1 'Z' is the Z Trigger flag that will enable FSM 2 to check the last bit of the sequence

Figure 3 Interacting FSMs Mealy: FSM 1

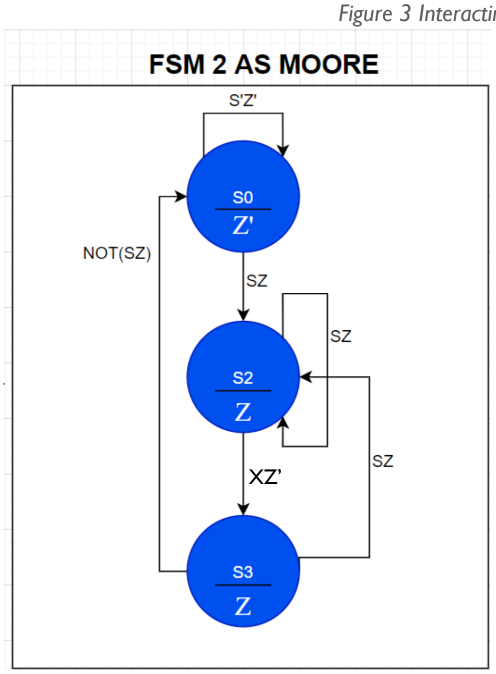
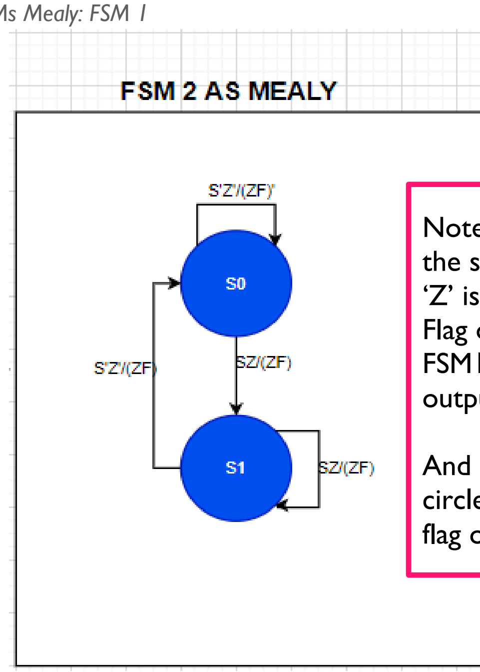


Figure 4 FSM 2 As Moore



Note: In FSM 2, 'S' is the sequence bit and 'Z' is the Z Trigger Flag output from FSM1. 'ZF' is the output Z flag for Mealy

And Z within the state circle is the output Z flag of the Moore

Figure 5 FSM 2 As Mealy (USED)

2.4 Design Approach Comparison

I. Comparing the interacting FSM design to the more primitive single FSM design, the main difference seen is that the interacting design is much more modular and will scale appropriately for different sequences and the amount of clock cycle a Z flag needs to be held, as these can be adjusted at FSM1 and FSM2.

II. Using multiple FSM allows for component level testing as opposed to having to test a single state machine composed of rather complex logic, therefore making it more desirable (specifically for a top down test-driven design approach).

III. With the interacting design, if the Z flag is to be held longer, an extra state can be added as FSM2 only depends on a trigger flag sent from FSM1 where it then checks the last digit within FSM2 to ensure that the **flag is set without any clock delays** (on the same cycle as the input sequence bit). See *figure 1*.

Implemented FSM Level Design:

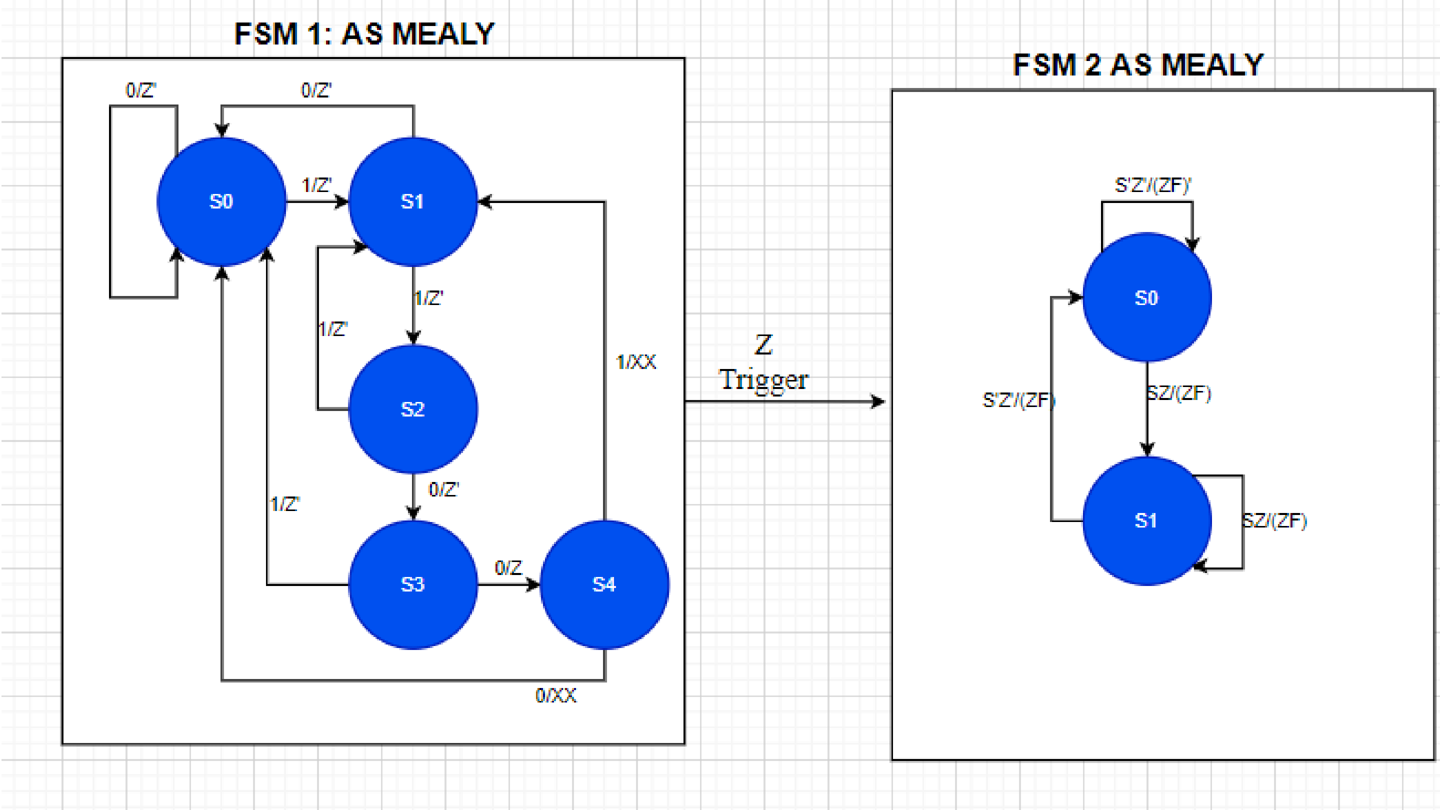


Figure 6 Implemented Design

2.5 Design Logic Flow:

The design implemented in *figure 6,* uses FSM 1 to check the first four bits of the sequence **11001**, if this sequence is detected (S3 -> S4), it sets Z trigger (Z) high which acts as a trigger flag/enable for FSM 2. To ensure that the output Z flag is set on the **same clock event** as the when the last sequence bit_comes through, FSM 2 will check the last sequence bit (denoted by S), if a Z trigger flag is set and sequence bit is **1**, then a Z flag is held **for 2 clock cycle**

3. Simulation and Results

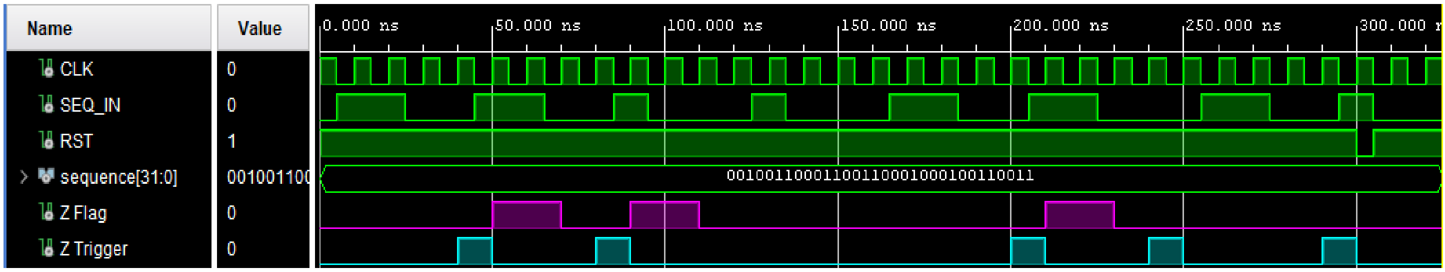


Figure 7 Overall Simulation Results

Figure 7¸ shows the overall simulation results for multiple cases, including an over-lapping sequence, a non-overlapping and a reset signal on clock event. **Sequence** [31:0] holds a 32bit long sequence which is iterated through and applied to SEQ_IN on every falling edge (within the test bed). *Figures* below demonstrate more specifics.

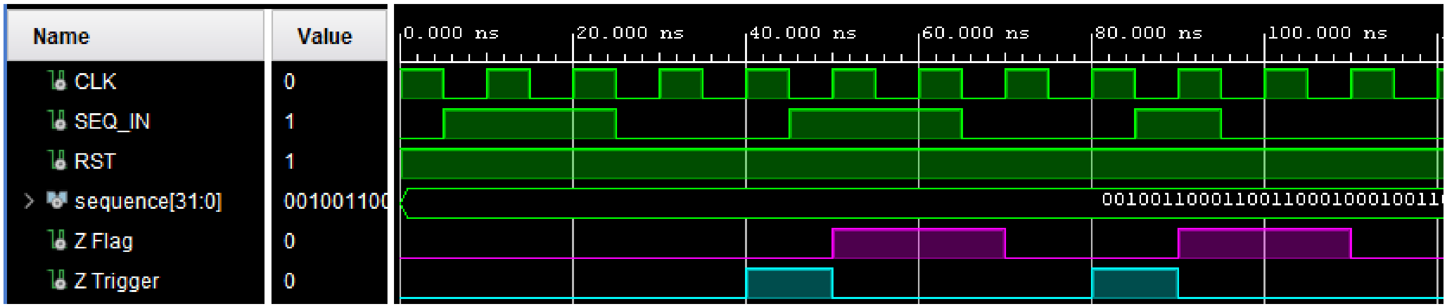


Figure 8 Overlapping Sequence Detection

Figure 8 shows the overlapping sequence **110011001** starting at the clock at 10ns. Notice that the Z Flag is set on the clock event as when the last bit of the correct sequence is detected, for instance at **50ns and is held high for 2 cycles**.

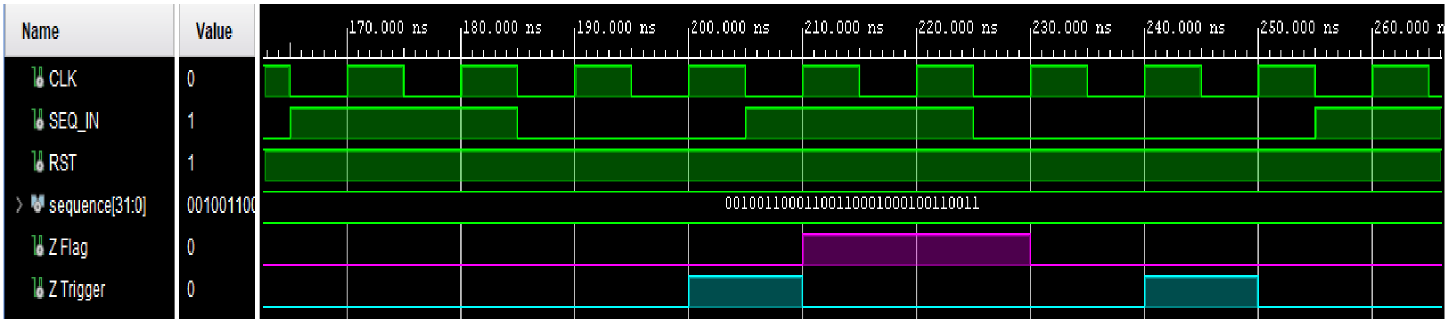


Figure 9 A Non-Overlapping Sequence Detection

Figure 9 shows a non-overlapping sequence detection, where the sequence starts at the clock event on 170ns. See that the Z flag is set at 210ns on same clock event as the last sequence bit without any delays and is held high for 2 cycles. Additionally, notice that at 240ms a Z Trigger flag from FSM1 is set, however the last bit does not match the sequence, therefore FSM 2 does not set a Z Flag at 250ms.

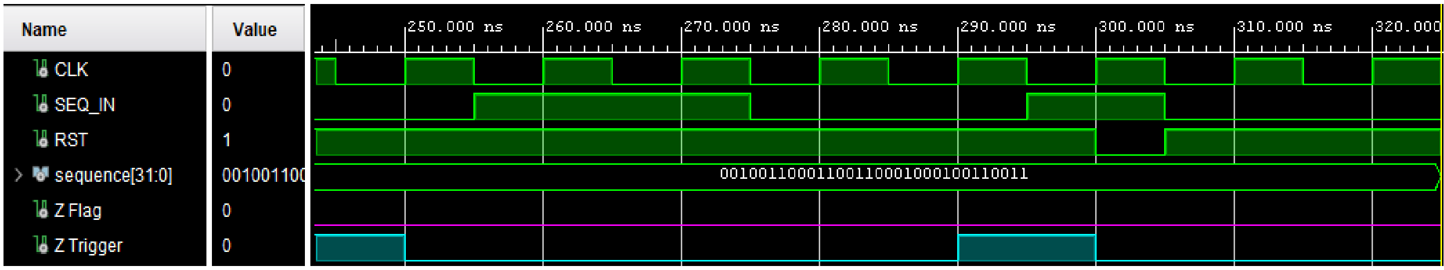


Figure 10 Simulated Active Low Reset

Figure 10 shows the active low resetting of FSM1 and FSM2, notice that a valid sequence starts at 260ns, and a Z Trigger flag is set from FSM1, however, FSM2 will not set the Z flag even if the last sequence bit is correct (300ns). This is a result of both FSMs being **reset** to the initial state.

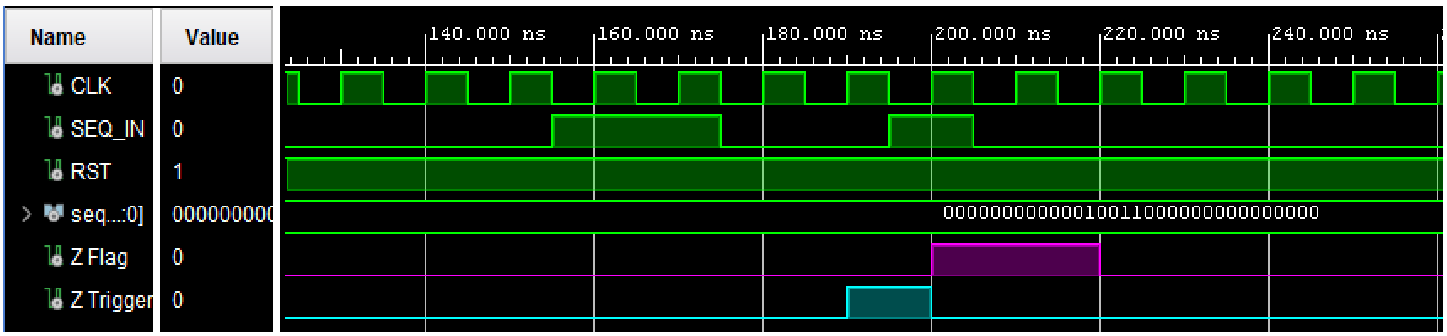


Figure 11 Single Sequence Detection

Figure 11 shows a **single sequence detection**.

4. Schematics

4.1 RTL Schematic

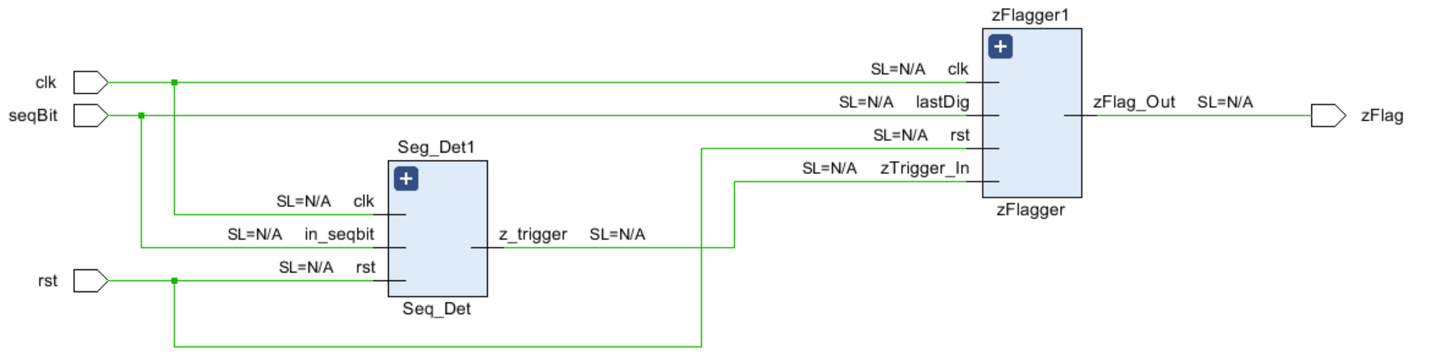


Figure 12 RTL Schematic

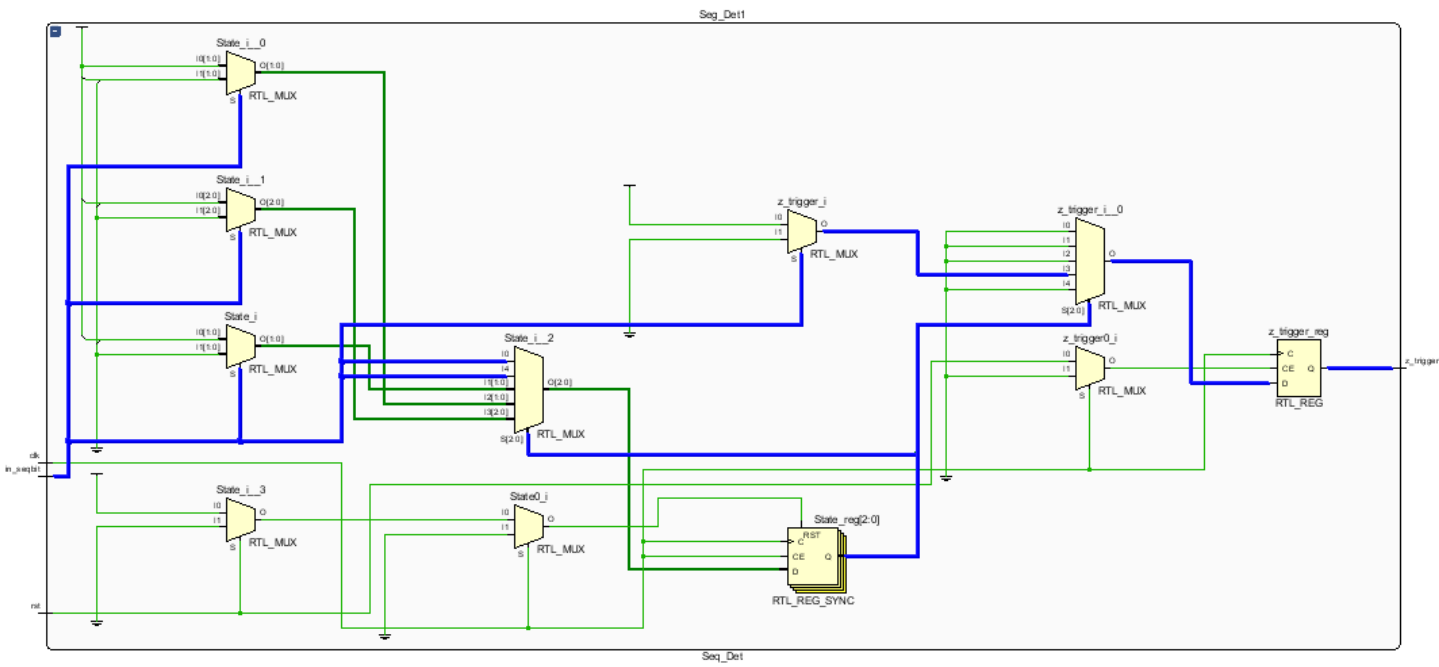


Figure 13 Looking Inside Seq_Det

- Looking inside the Seq_Det block (FSM1), it is seen that that states are stored in a state register (State_reg, RTL_REG_SYNC).
- State selection is done through State_i_2 MUX and this is the logic that parses the sequence bit.
- The state register output is used in the RTL_MUX – z_trigger_i_0 as the data select and is used to set the z_trigger output of FSM1.

- State signal definition:

```
type seq_state is (first, second, third, fourth, fifth);
-- SEQ TO MATCH 11001 left to right
signal State : seq_state;
```

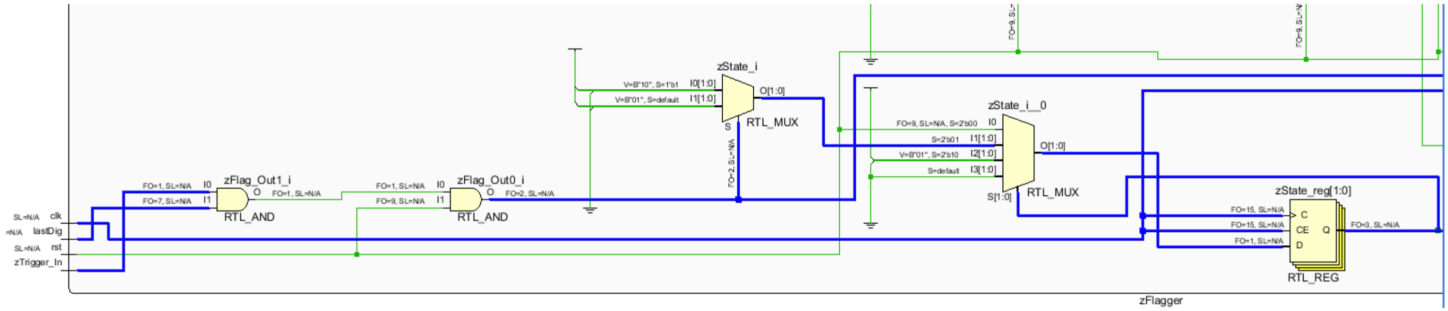


Figure 14 Looking inside zFlagger FSM2

- As expected, this FSM at RTL follows the same general pattern as seen in FSM1.
- The inputs are parsed and is used to set the zState_Reg (zState is defined as below)

```
type z_state is (rstl, first, second);
-- SEQ TO MATCH 11001 left to right
signal zState : z_state;
```

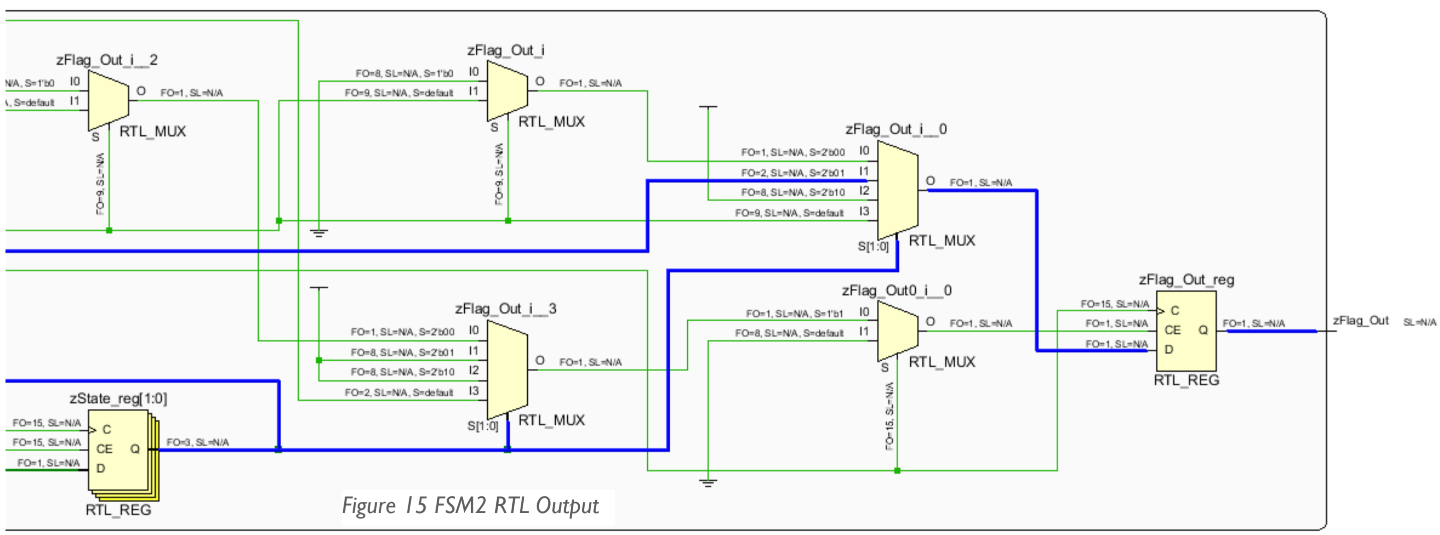


Figure 15 FSM2 RTL Output

- The output of FSM2 is determined similarly to FSM1, where the state register output is used to select the output based on already calculated input logic.

5. Synthesis Schematics

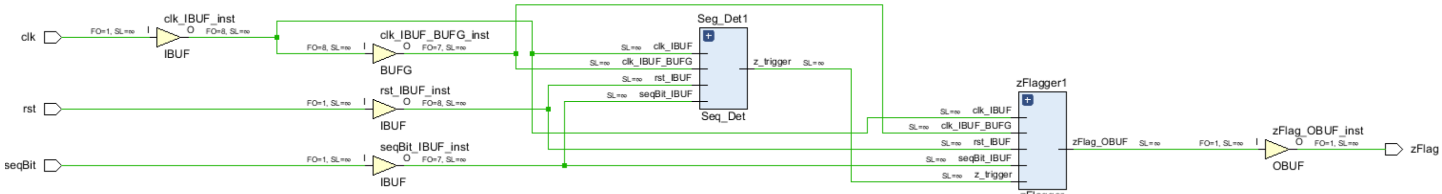


Figure 16 Synthesis Schematic

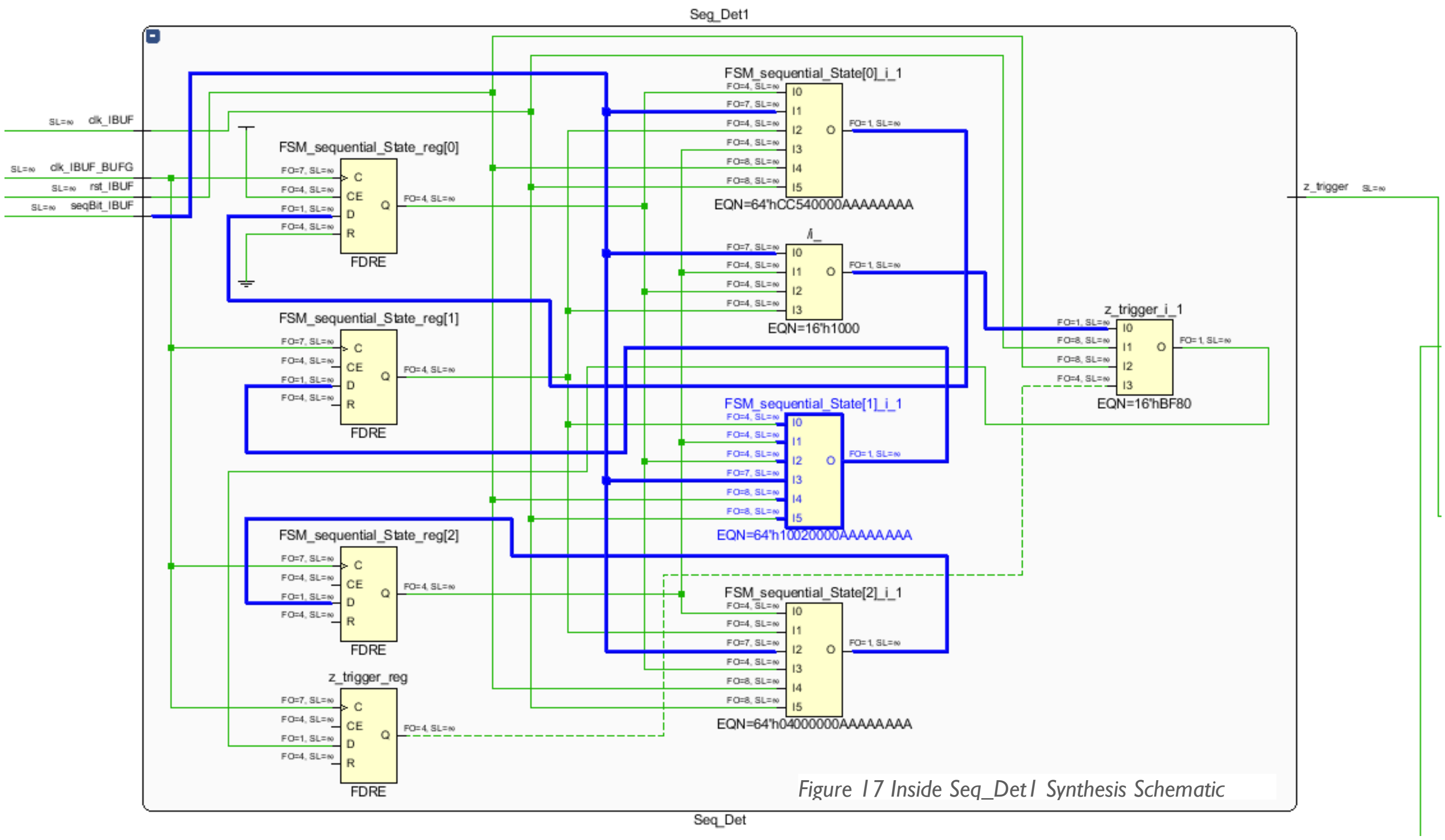


Figure 17 Inside Seq_Det1 Synthesis Schematic

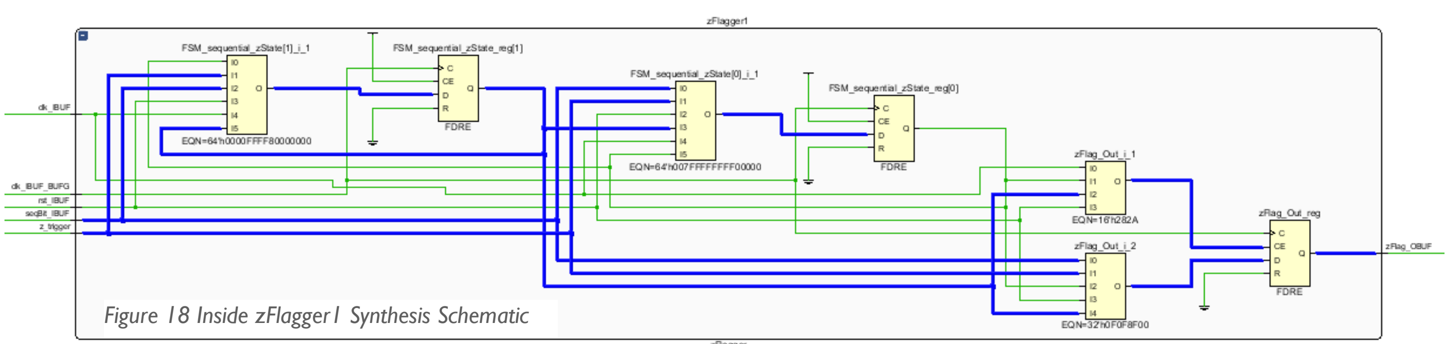


Figure 18 Inside zFlagger1 Synthesis Schematic

The implemented design seen in *figure 19*, is rather difficult to follow at the low level, what seem to be the state transitions can be seen through the highlighted trails. However, *figure 18* is the high-level component level implementation which is exactly as the implemented design in *figure 1* with the addition of signal buffers.

6. FPGA Resource Consumption

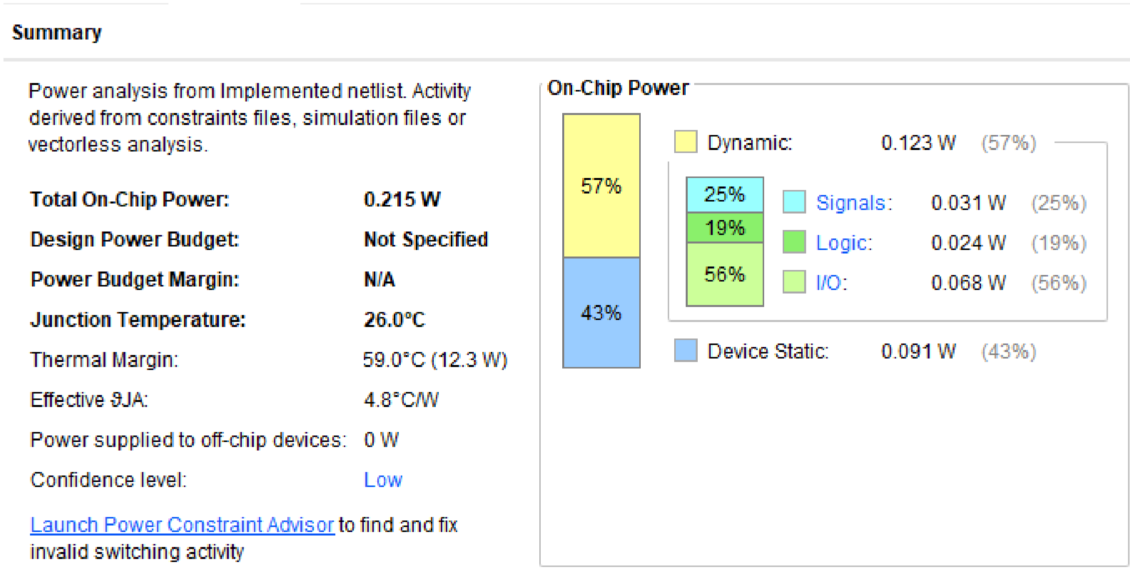


Figure 19 FPGA On Chip Power Consumption

The power usage is based on **default Vivado config** settings for a power supply, as device specifications are not listed in the spec.

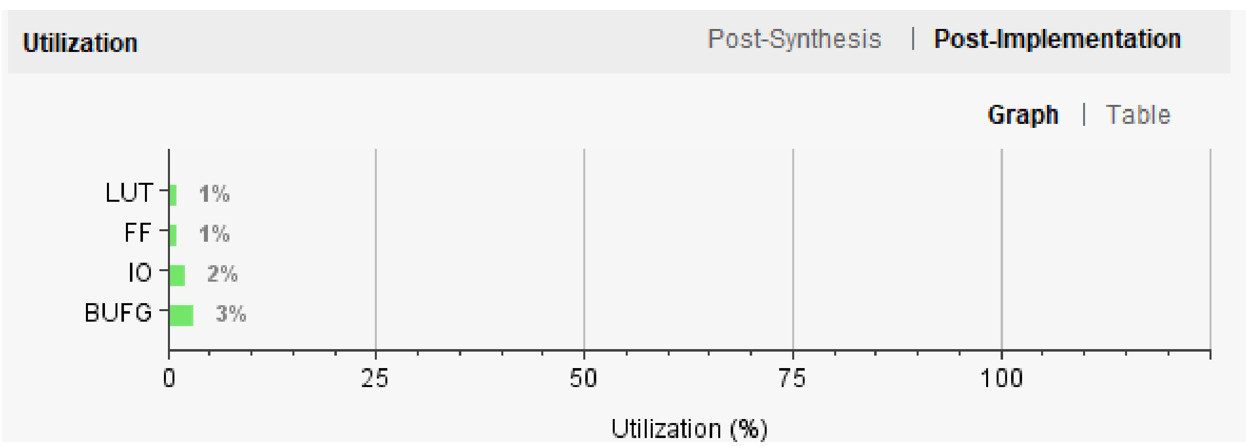


Figure 21 FBGA Component Consumption

Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)	BUFGCTRL (32)
seqDet_Top	9	7	3	9	4	1
Seg_Det1 (Seq_Det)	5	4	2	5	0	0
zFlagger1 (zFlagger)	4	3	2	4	0	0

Figure 20 On Chip Component

7. Summary

Aims of this practical were to understand finites state machines and state diagrams learning to alternate between Mealy and Moore designs. Additionally, to implement two interacting FSM as a sequence detector that will detect "11001" and set an output flag high for two clock cycles. These aims were met.

The design process consisted of first drawing a rough state diagram for the FSMs, then implementing the logic in VHDL and testing exhaustively until the correct functionality reached.

The main difficulty of this project and the last projects so far has been attempting to understand how some of the low-level synthesis schematic implementations function for instance *Figure 13*.

An alternative design approach would be to do the same design for the state machines seen in *figure 6* after **manual state minimization** to simplify design and thus resource utilization and comparing this to the optimizations made by the Vivado synthesis tool.

8. Marking Criteria

Marks	**Criteria**
Simulation	
0 1 2 3 4	Simulation not attempted or does not work Simulation works only for non-overlapping sequence detection Simulation works only for overlapping sequence detection Simulation fully works for both overlapping and non-overlapping cases Simulation fully works for both overlapping and non-overlapping cases with output retained for two clock periods
Report	
0 1 2 3 4	No evidence of content or work Some content, insufficient explanation of circuit Reasonable content, some explanation of circuit (both approaches are explained) Good content, reasonable explanation of circuit (both approaches are explained) Excellent content, good explanation of circuit (both approaches are explained)
Oral assessment	
0 1 2 3 4	No knowledge of the design Very little knowledge of the design Reasonable knowledge of the design Good knowledge of the design. Excellent knowledge of the design.
Total (12):	**Marker Initials:** **Date:**