EE 421/621, Fall 2024

Project 2:
Moore FSM Design

November 26th, 2024

Michael Pittenger

Background

Finite State Machines (FSMs) are foundational tools in digital electronics design and are used for sequential decision making logic. FSMs consist of defined (finite) states in which the next state is determined by the current state and input. The output of a Moore FSM is determined solely by the state the machine is in and the output of a Mealy FSM is determined by both the current state and input of the machine.

Design

The task for this project is to design a Moore FSM that takes an input w and an output z and generates z = “1” when the previous four values of w were “1001”, “1111”, or “1001”, otherwise z = “0”. The machine should allow for overlapping patterns as well. The machine should be implemented on Quartus Prime Software in VHDL and run on the DE1-SoC FPGA board.

The first step to constructing a Moore FSM is to design the states. By taking the desired patterns and incrementing them digit by digit, we are able to find all of the states necessary for the detection of those specific patterns. **Figure 1** shows the development in the states, which comes out to a total of 12 states. Due to 12 being represented as “1100” in binary, we only need 4 DFFs (D Flip Flops) in order to represent and track all of the states.

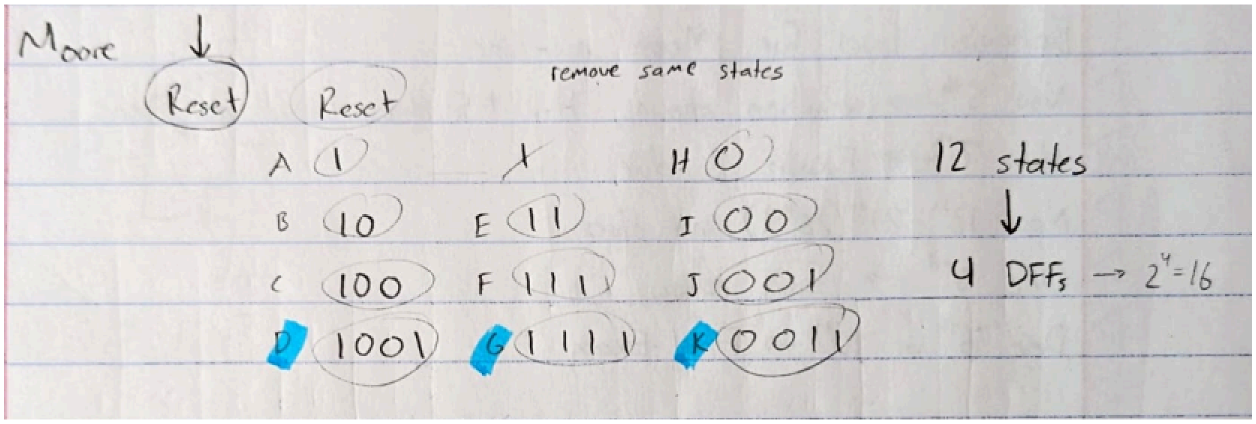


Figure 1: Finding States

Once the states are found, we need to determine the state transition flow of the machine. We do this by starting with a state and drawing the different possible inputs at that state, which is just “0” and “1” in this case. **Figure 2** shows the state transition diagram. The patterns we are tracking are 4 digits long, meaning that we must only track the last 4 inputs at a state when determining what the next state will be. For example, State D with an input of “0” has an input stream of “0010010”, but only the last four “0010” matter for pattern detection. This “0010” aligns with State B, so we can transition State D to State B instead of creating a new state.

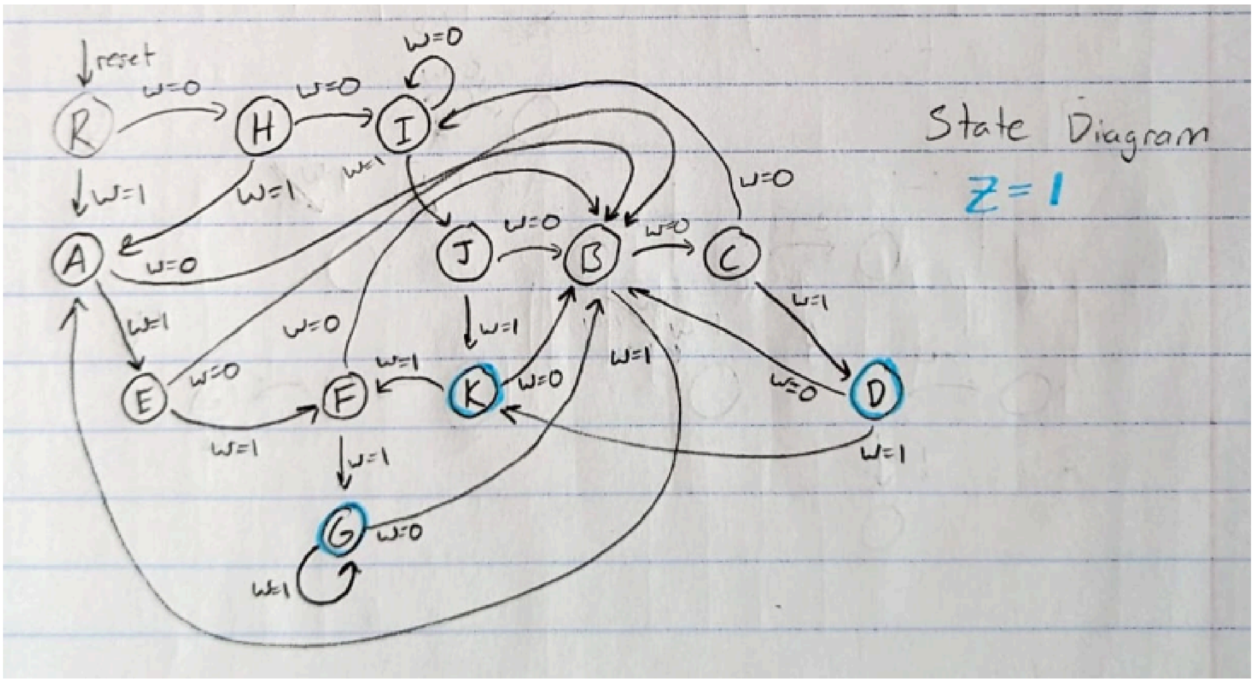


Figure 2: State Transition Diagram

Table 1 shows the State Transition Table, which represents the same information shown in the State Transition Diagram.

Table 1: State Transition Table

Present State	Next State		Output z
	w = 0	w = 1	
Reset	H	A	0
A	B	E	0
B	C	A	0
C	I	D	0
D	B	K	1
E	B	F	0
F	B	G	0
G	B	G	1
H	I	A	0
I	I	J	0

J	B	K	0
K	B	F	1

From here, you can either convert the states to binary values at register locations (such as State 1 = “0001” and State 12 = “1100” and design combination circuits to get your from the previous state values to the current state values, shown in **Figure 3**. Alternatively, you could use behavioral level programming with case statements to allow the software to design the logic for you, shown in **Figure 4**. For this project, the behavioral programming method was used.

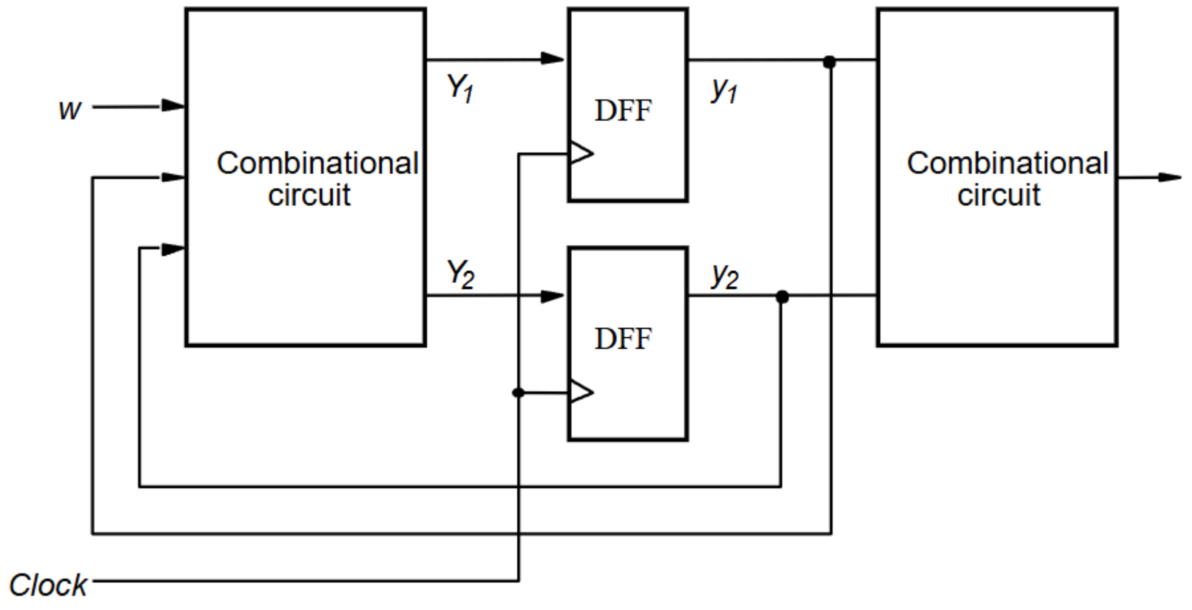
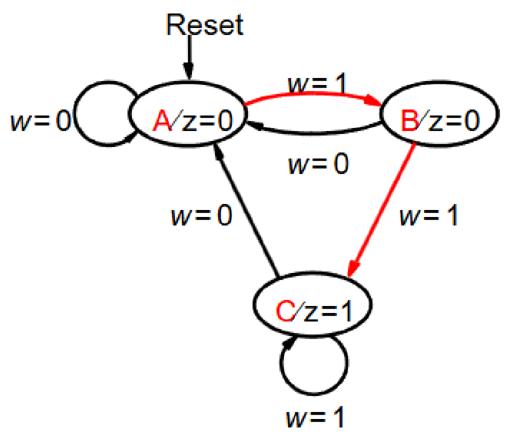


Figure 3: Moore FSM Combination Example

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY simple IS
        PORT (    Clock, Resetn, w  : IN    STD_LOGIC ;
                  z                         : OUT   STD_LOGIC ) ;
END simple ;

ARCHITECTURE Behavior OF simple IS
        TYPE State_type IS (A, B, C) ;
        SIGNAL y : State_type ;
BEGIN
        PROCESS ( Resetn, Clock )
        BEGIN
              IF Resetn = '0' THEN
                      y <= A ;
              ELSIF (Clock'EVENT AND Clock = '1') THEN
                      CASE y IS
                              WHEN A =>
                                      IF w = '0' THEN
                                              y <= A ;
                                      ELSE
                                              y <= B ;
                                      END IF ;
                              WHEN B =>
                                      IF w = '0' THEN
                                              y <= A ;
                                      ELSE
                                              y <= C ;
                                      END IF ;
                              WHEN C =>
                                      IF w = '0' THEN
                                              y <= A ;
                                      ELSE
                                              y <= C ;
                                      END IF ;
                      END CASE ;
              END IF ;
        END PROCESS ;
        z <= '1' WHEN y = C ELSE '0' ;
END Behavior ;
```



Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	A	B	0
B	A	C	0
C	A	C	1

Figure 4: Moore FSM Behavioral Example

A generated Netlist of the FSM design is shown in **Figure 5**. The KEY[3..0] buttons are the inputs and the LEDR[9..0] LEDs are the outputs. The y section represents the state transition logic, shown in **Figure 6**.

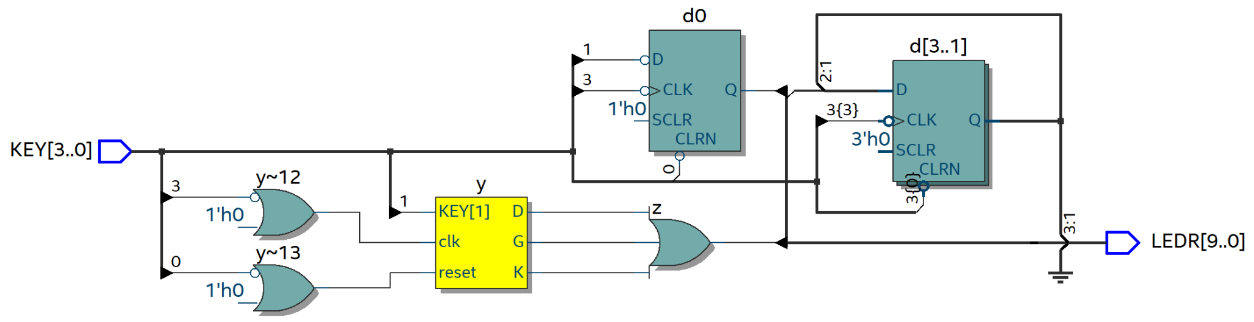


Figure 5: FSM Netlist

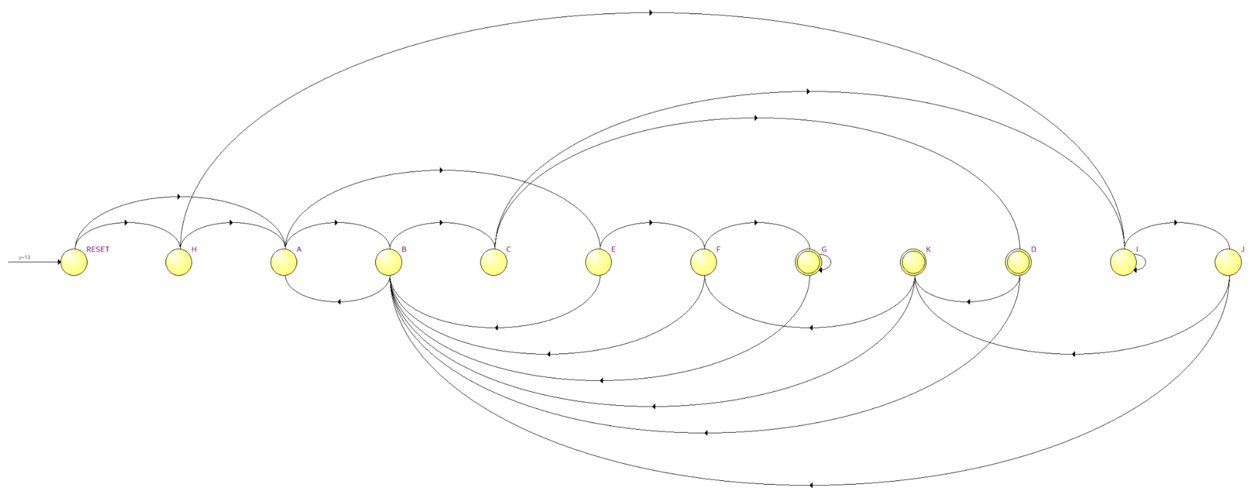


Figure 6: FSM Netlist State Transition Diagram

Conclusion

This project successfully showcases the design and implementation of a Moore FSM for detecting the overlapping patterns of “1001”, “1111”, and “0011”. By systematically creating a state diagram and state transition table, the FSM was efficiently designed to handle the proposed input sequences. The machine was implemented at a behavioral level in VHDL, simulated using Quartus Prime Software, and verified on the DE1-SoC FPGA board. This project highlights one of the use cases of FSMs, which is for pattern detection, and it also reinforced the concepts of sequential logic design.