EE529 Embedded Systems

Lab Assignment 2

Student Name : Ayan Garg

Roll Number : B23484

School of Computing and Electrical Engineering (SCEE)
Indian Institute Of Technology Mandi
February 9, 2026

Contents

1 Objective

The objective of this experiment is to design and implement a finite state machine (FSM) capable of recognizing specific input sequences in a synchronous digital system. The FSM is required to detect four consecutive occurrences of logic '1' or logic '0' at its input and assert an output accordingly.

2 Theory

A finite state machine (FSM) is a sequential digital system whose behavior depends not only on the current input values but also on its present state, which represents the history of past inputs. FSMs are widely used in embedded systems, digital controllers, communication protocols, and sequence detection circuits.

2.1 Sequence Detection FSM

In this experiment, the FSM is designed to recognize two specific input sequences:

- Four consecutive logic '1's
- Four consecutive logic '0's

The system has a single input signal w and a single output signal z. The output z is asserted ($z = 1$) whenever the input w remains at logic '1' or logic '0' for four consecutive clock cycles. For all other conditions, the output remains deasserted ($z = 0$).

2.2 Overlapping Sequence Detection

Overlapping sequences are permitted in this FSM. This means that once four consecutive identical inputs are detected, continued occurrences of the same input value will keep the output asserted in subsequent clock cycles. For example, if the input remains at logic '1' for five consecutive clock pulses, the output will be asserted after the fourth and fifth pulses.

This behavior requires the FSM to remain in an output-asserting state as long as the input continues to satisfy the detection condition.

2.3 State Representation and Moore Machine

The FSM is implemented as a **Moore machine**, in which the output depends solely on the current state of the system and not directly on the input. Separate states are used to represent the number of consecutive '1's or '0's observed so far. When the input changes value, the FSM transitions to the appropriate state corresponding to the new input sequence.

The Moore machine approach ensures synchronous and glitch-free output behavior, which is desirable in clocked digital systems.

2.4 VHDL Modeling Approaches

The FSM is described using two VHDL modeling styles:

- **Behavioral modeling**, which uses processes and conditional statements to describe the FSM functionality at a high level.
- **Structural modeling**, which represents the FSM as an interconnection of flip-flops and combinational logic, closely reflecting the actual hardware realization.

Using both approaches highlights the abstraction capabilities of VHDL while also demonstrating the correspondence between high-level FSM descriptions and low-level sequential circuit implementations.

2.5 State Transition Diagram

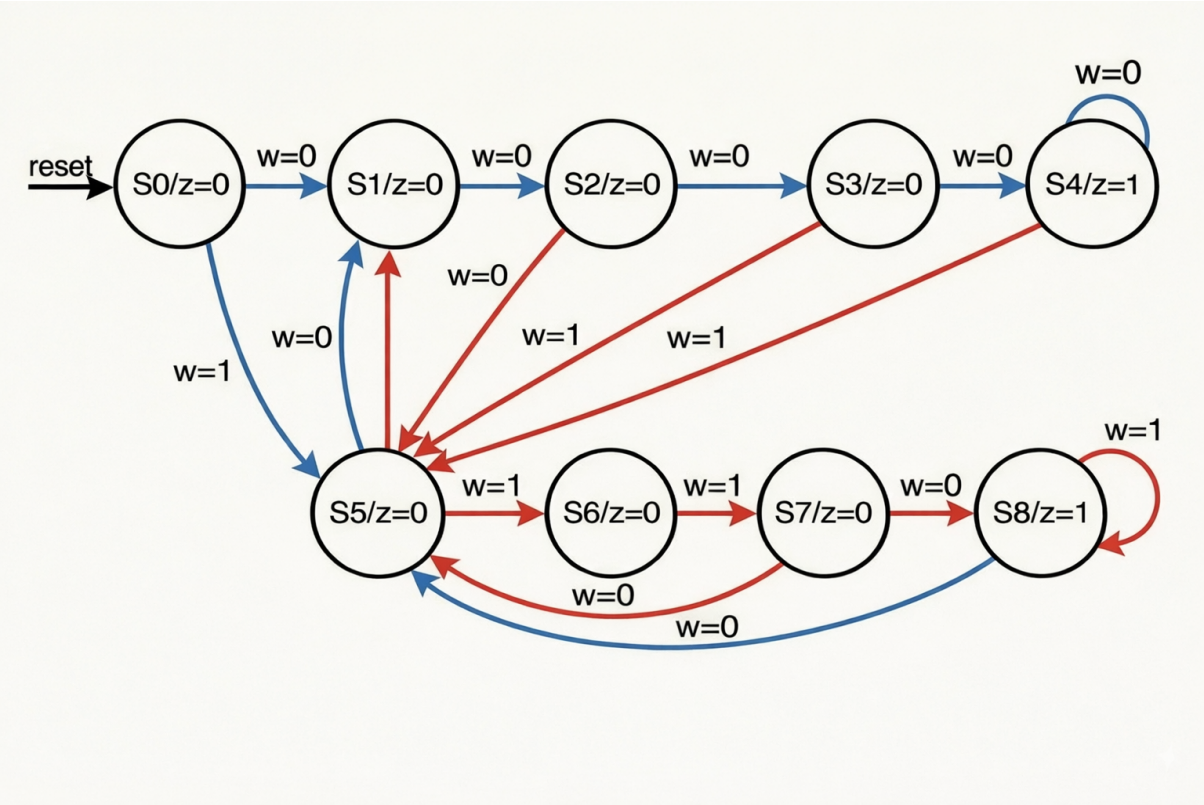

Figure 1: Simulation Result with Expected Output

2.6 Structural Realization of the FSM

The given finite state machine (FSM) has nine states $S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8$, one binary input w, and one binary output z. In the structural implementation, the FSM is realized using a one–hot state encoding and a bank of D flip–flops with asynchronous active–high reset.

State Encoding

A one–hot encoding is used, where each state is represented by a dedicated flip–flop. Let the state vector be $\mathbf{S} = [S_0\ S_1\ S_2\ S_3\ S_4\ S_5\ S_6\ S_7\ S_8]^T$, with $S_i \in \{0, 1\}$ and exactly one $S_i = 1$ at any time.

State	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
S_0	1	0	0	0	0	0	0	0	0
S_1	0	1	0	0	0	0	0	0	0
S_2	0	0	1	0	0	0	0	0	0
S_3	0	0	0	1	0	0	0	0	0
S_4	0	0	0	0	1	0	0	0	0
S_5	0	0	0	0	0	1	0	0	0
S_6	0	0	0	0	0	0	1	0	0
S_7	0	0	0	0	0	0	0	1	0
S_8	0	0	0	0	0	0	0	0	1

Next–State Equations

Let S_i^+ denote the next value of the flip–flop corresponding to state S_i. The reset signal forces the FSM into state S_0. In terms of the one–hot state variables and the input w, the next–state equations are:

$$S_0^+ = \text{reset}$$

$$S_1^+ = S_0\,\overline{w} + S_5\,\overline{w} + S_6\,\overline{w} + S_7\,\overline{w} + S_8\,\overline{w}$$

$$S_2^+ = S_1\,\overline{w}$$

$$S_3^+ = S_2\,\overline{w}$$

$$S_4^+ = S_3\,\overline{w} + S_4\,\overline{w}$$

$$S_5^+ = (S_0 + S_1 + S_2 + S_3 + S_4)\,w$$

$$S_6^+ = S_5\,w$$

$$S_7^+ = S_6\,w$$

$$S_8^+ = S_7\,w + S_8\,w$$

where + denotes logical OR, juxtaposition denotes logical AND, and $\overline{w}$ is the logical complement of w.

Output Equation

The output z is asserted in states S_4 and S_8. Using the one–hot encoding, the output equation is:

$$z = S_4 + S_8$$

Structural Implementation

Each state bit S_i is implemented with a D flip–flop. The input of the i-th flip–flop is driven by the corresponding next–state equation S_i^+, and all flip–flops share the common clock signal. The asynchronous active–high reset clears the state flip–flops and, together with the combinational logic driving S_0^+, initializes the FSM into the reset state S_0.

3 VHDL Implementation

3.1 Behavioral VHDL Description

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity q1_behavioral is
    Port (
        clk   : in  std_logic;
        reset : in  std_logic;
        w     : in  std_logic;
        z     : out std_logic
    );
end q1_behavioral;

architecture behavioral of q1_behavioral is

    type state_type is (
        S0,
        S1, S2, S3, S4,
        S5, S6, S7, S8
    );
    signal current_state, next_state : state_type;

begin

    process(clk, reset)
    begin
        if reset = '1' then
            current_state <= S0;
        elsif rising_edge(clk) then
            current_state <= next_state;
        end if;
    end process;

    process(current_state, w)
    begin
        case current_state is

            when S0 =>
                if w = '0' then
                    next_state <= S1;
                else
                    next_state <= S5;
                end if;

            when S1 =>
                if w = '0' then next_state <= S2;
                else            next_state <= S5;
                end if;

            when S2 =>
                if w = '0' then next_state <= S3;
                else            next_state <= S5;
                end if;

            when S3 =>
                if w = '0' then next_state <= S4;
                else            next_state <= S5;
                end if;

            when S4 =>
                if w = '0' then next_state <= S4;
                else            next_state <= S5;
                end if;

            when S5 =>
                if w = '1' then next_state <= S6;
                else            next_state <= S1;
                end if;

            when S6 =>
                if w = '1' then next_state <= S7;
                else            next_state <= S1;
                end if;

            when S7 =>
                if w = '1' then next_state <= S8;
                else            next_state <= S1;
                end if;
```

```
78
79              when S8 =>
80                  if w = '1' then next_state <= S8;
81                  else            next_state <= S1;
82                  end if;
83
84          end case;
85      end process;
86
87      process(current_state)
88      begin
89          case current_state is
90              when S4 | S8 =>
91                  z <= '1';
92              when others =>
93                  z <= '0';
94          end case;
95      end process;
96
97  end behavioral;
```

Listing 1: VHDL Code for Required Sequence Detector

3.2 Behavioral VHDL TestBench

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity q1_behavioral_tb is
5   end q1_behavioral_tb;
6
7   architecture Behavioral of q1_behavioral_tb is
8       component q1_behavioral
9           Port (
10              clk : in  std_logic;
11              reset : in  std_logic;
12              w   : in  std_logic;
13              z   : out std_logic
14          );
15      end component;
16
17      signal clk : std_logic := '0';
18      signal reset : std_logic := '0';
19      signal w   : std_logic := '0';
20      signal z   : std_logic;
21
22  begin
23
24      uut: q1_behavioral
25          port map (
26              clk => clk,
27              reset => reset,
28              w   => w,
29              z   => z
30          );
31
32      clk_process : process
33      begin
34          while true loop
35              clk <= '0';
36              wait for 5 ns;
37              clk <= '1';
38              wait for 5 ns;
39          end loop;
40      end process;
41
42      rst_process : process
43      begin
44          reset <= '1';
45          wait for 20 ns;
46          reset <= '0';
47          wait;
48      end process;
49
50
51      stim_proc: process
52      begin
53          wait until reset = '0';
54
55          w <= '1'; wait for 10 ns;
56          w <= '1'; wait for 10 ns;
57          w <= '1'; wait for 10 ns;
58          w <= '1'; wait for 10 ns;
59          w <= '1'; wait for 10 ns;
60          w <= '0'; wait for 10 ns;
61          w <= '0'; wait for 10 ns;
62          w <= '1'; wait for 10 ns;
63          w <= '0'; wait for 10 ns;
64          w <= '0'; wait for 10 ns;
65          w <= '0'; wait for 10 ns;
66          w <= '0'; wait for 10 ns;
67
68          assert false
```

```
            report "Simulation completed successfully"
            severity failure;
    end process;

end Behavioral;
```

Listing 2: VHDL Testbench for Required Sequence Detector

3.3 Structural VHDL Description

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity q1_structural is
    port (
        clk   : in  std_logic;
        reset : in  std_logic;
        w     : in  std_logic;
        z     : out std_logic
    );
end q1_structural;

architecture structural of q1_structural is

    signal s0,s1,s2,s3,s4,s5,s6,s7,s8 : std_logic;
    signal d0,d1,d2,d3,d4,d5,d6,d7,d8 : std_logic;

begin

    d0 <= '1' when reset = '1' else '0';

    d1 <= (s0 and not w) or (s5 and not w) or (s6 and not w) or
          (s7 and not w) or (s8 and not w);

    d2 <= s1 and not w;
    d3 <= s2 and not w;
    d4 <= (s3 and not w) or (s4 and not w);

    d5 <= (s0 or s1 or s2 or s3 or s4) and w;
    d6 <= s5 and w;
    d7 <= s6 and w;
    d8 <= (s7 and w) or (s8 and w);

    z <= s4 or s8;

    FF0 : entity work.DFF port map (D => d0, clk => clk, reset => '0',   Q => s0);
    FF1 : entity work.DFF port map (D => d1, clk => clk, reset => reset, Q => s1);
    FF2 : entity work.DFF port map (D => d2, clk => clk, reset => reset, Q => s2);
    FF3 : entity work.DFF port map (D => d3, clk => clk, reset => reset, Q => s3);
    FF4 : entity work.DFF port map (D => d4, clk => clk, reset => reset, Q => s4);
    FF5 : entity work.DFF port map (D => d5, clk => clk, reset => reset, Q => s5);
    FF6 : entity work.DFF port map (D => d6, clk => clk, reset => reset, Q => s6);
    FF7 : entity work.DFF port map (D => d7, clk => clk, reset => reset, Q => s7);
    FF8 : entity work.DFF port map (D => d8, clk => clk, reset => reset, Q => s8);

end structural;
```

Listing 3: VHDL Code for Required Sequence Detector

3.4 Structural VHDL TestBench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity q1_structural_tb is
end q1_structural_tb;

architecture Behavioral of q1_structural_tb is

    component q1_structural
        port (
            clk   : in  std_logic;
            reset : in  std_logic;
            w     : in  std_logic;
            z     : out std_logic
        );
    end component;

    signal clk   : std_logic := '0';
    signal reset : std_logic := '0';
    signal w     : std_logic := '0';
    signal z     : std_logic;

begin

    DUT : q1_structural
        port map (
```

```
27              clk   => clk,
28              reset => reset,
29              w     => w,
30              z     => z
31          );
32
33      clk_process : process
34      begin
35          while true loop
36              clk <= '0';
37              wait for 5 ns;
38              clk <= '1';
39              wait for 5 ns;
40          end loop;
41      end process;
42
43      reset_process : process
44      begin
45          reset <= '1';
46          wait for 20 ns;
47          reset <= '0';
48          wait;
49      end process;
50
51      stim_proc : process
52      begin
53          wait until reset = '0';
54
55          wait until rising_edge(clk); w <= '1';
56          wait until rising_edge(clk); w <= '1';
57          wait until rising_edge(clk); w <= '1';
58          wait until rising_edge(clk); w <= '1';
59          wait until rising_edge(clk); w <= '1';
60          wait until rising_edge(clk); w <= '0';
61          wait until rising_edge(clk); w <= '0';
62          wait until rising_edge(clk); w <= '1';
63          wait until rising_edge(clk); w <= '0';
64          wait until rising_edge(clk); w <= '0';
65          wait until rising_edge(clk); w <= '0';
66          wait until rising_edge(clk); w <= '0';
67          wait until rising_edge(clk); w <= '0';
68          wait until rising_edge(clk); w <= '0';
69
70          assert false
71              report "Simulation completed successfully"
72              severity failure;
73      end process;
74
75  end Behavioral;
```

Listing 4: VHDL Testbench for Required Sequence Detector

4 Results

4.1 Behavioral Coding

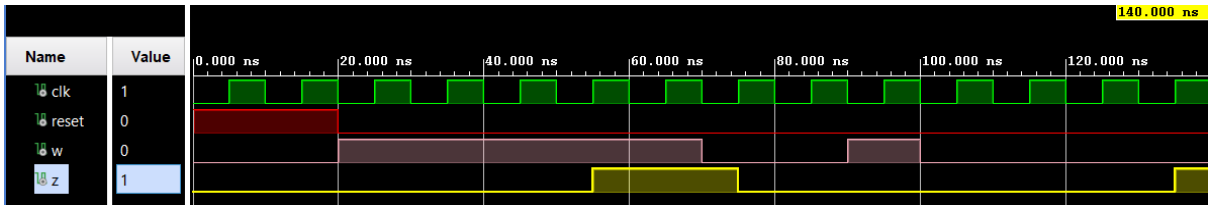

Figure 2: Simulation Result with Expected Output

4.2 Structural Coding

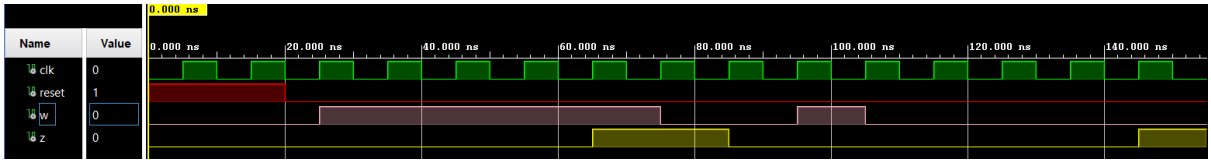

Figure 3: Simulation Result with Expected Output