LAB REPORT
******On Numbers and Display******

Author 洪晨辉 3220101111*

****Date 2024.12.11****

1. Problem Description

1）Implement a finite state machine (FSM) that recognizes four consecutive 1s or four consecutive 0s using one-hot code state machine and pure VHDL logical expression.

2）Change it into a more VHDL state machine expression-like one making advantage of TYPE reserved word and CASE operand.

2. Design Formulation

Markov chain for both assignments are prescribed as follows.

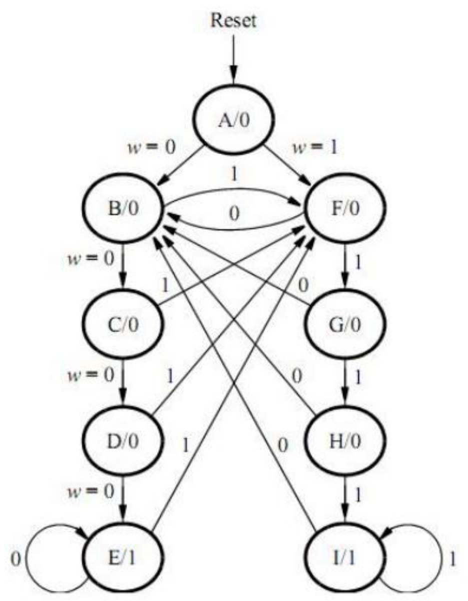


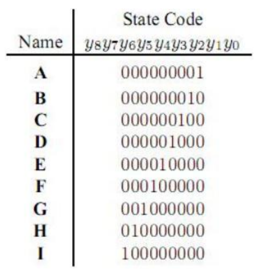
Fig.1 State Flow

Problem 1 treat it more like a logical operation by determine the next stage Q' by the current Q and w. One typical form such as D could be expressed in this form,

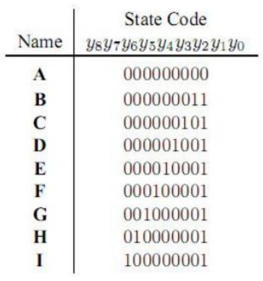
Q'(3) <= (Q(0) and not w)

Indicating the state would only change into D if it was C and w was 0 beforehand.

Two types of one-hot expression could be implemented regarding whether the A state flip-flop is separated from the rest or integrated. The latter one is purely for digital circuit convenience.

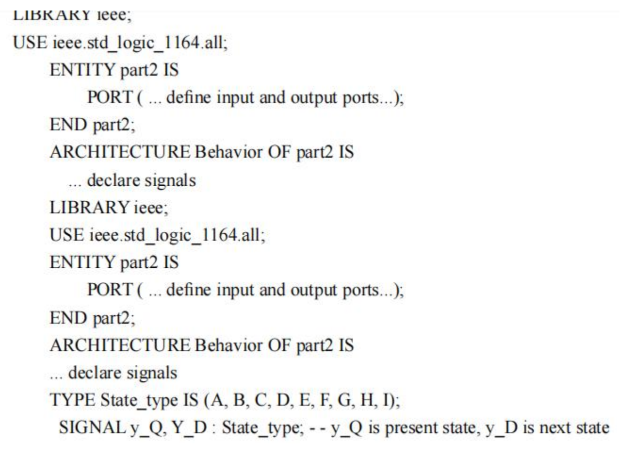


Name	State Code $y_8y_7y_6y_5y_4y_3y_2y_1y_0$
A	000000001
B	000000010
C	000000100
D	000001000
E	000010000
F	000100000
G	001000000
H	010000000
I	100000000



Name	State Code $y_8y_7y_6y_5y_4y_3y_2y_1y_0$
A	000000000
B	000000011
C	000000101
D	000001001
E	000010001
F	000100001
G	001000001
H	010000001
I	100000001

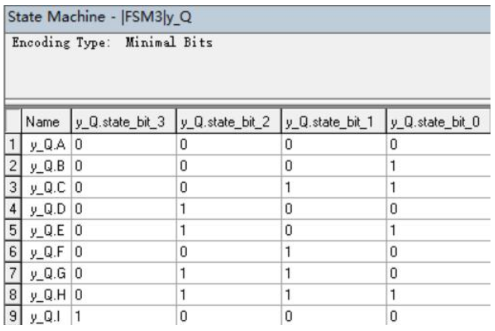
Problem 2 obtained a more state-minded approach on implementing such matter, with structure similar to the following code,



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
    ENTITY part2 IS
        PORT ( ... define input and output ports...);
    END part2;
    ARCHITECTURE Behavior OF part2 IS
      ... declare signals
    LIBRARY ieee;
    USE ieee.std_logic_1164.all;
    ENTITY part2 IS
        PORT ( ... define input and output ports...);
    END part2;
    ARCHITECTURE Behavior OF part2 IS
    ... declare signals
    TYPE State_type IS (A, B, C, D, E, F, G, H, I);
     SIGNAL y_Q, Y_D : State_type; - - y_Q is present state, y_D is next state
```

```
BEGIN
...
PROCESS (w, y_Q) - - state table
BEGIN
    case y_Q IS
    WHEN A IF (w = '0') THEN Y_D <= B;
    ELSE Y_D <= F;
    END IF;
    ... other states
    END CASE;
END PROCESS; - - state table
PROCESS (Clock) - - state flip-flops
    BEGIN
...
END PROCESS;
... assignments for output z and the LEDs
```

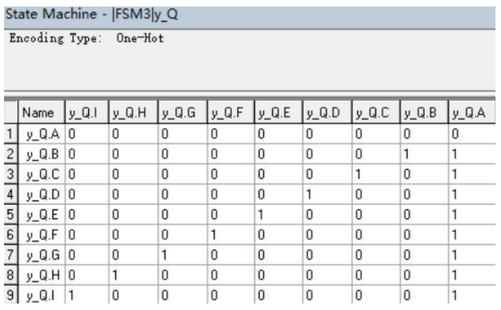
It is worth noting that this mod is more convenient in changing the state machine code. For instance, by clicking the State Machine Processing embedded in the Setting, one could change the code from minimal bits,


State Machine - |FSM3|y_Q

Encoding Type: Minimal Bits

	Name	y_Q.state_bit_3	y_Q.state_bit_2	y_Q.state_bit_1	y_Q.state_bit_0
1	y_Q.A	0	0	0	0
2	y_Q.B	0	0	0	1
3	y_Q.C	0	0	1	1
4	y_Q.D	0	1	0	0
5	y_Q.E	0	1	0	1
6	y_Q.F	0	0	1	0
7	y_Q.G	0	1	1	0
8	y_Q.H	0	1	1	1
9	y_Q.I	1	0	0	0

...to one-hot.


State Machine - |FSM3|y_Q

Encoding Type: One-Hot

	Name	y_Q.I	y_Q.H	y_Q.G	y_Q.F	y_Q.E	y_Q.D	y_Q.C	y_Q.B	y_Q.A
1	y_Q.A	0	0	0	0	0	0	0	0	0
2	y_Q.B	0	0	0	0	0	0	0	1	1
3	y_Q.C	0	0	0	0	0	0	1	0	1
4	y_Q.D	0	0	0	0	0	1	0	0	1
5	y_Q.E	0	0	0	0	1	0	0	0	1
6	y_Q.F	0	0	0	1	0	0	0	0	1
7	y_Q.G	0	0	1	0	0	0	0	0	1
8	y_Q.H	0	1	0	0	0	0	0	0	1
9	y_Q.I	1	0	0	0	0	0	0	0	1

3. Design Entry

Part I

One-hot

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

entity FSM is
    Port (
        clk : in STD_LOGIC;         -- Clock input
        reset : in STD_LOGIC;       -- Active-low reset
        w : in STD_LOGIC;           -- Input signal
        z : out STD_LOGIC;          -- Output signal
        y : out STD_LOGIC_VECTOR(8 downto 0) -- State outputs for debugging
    );
end FSM;

architecture Behavioral of FSM is
    signal state : STD_LOGIC_VECTOR(8 downto 0) := "000000001"; -- One-hot encoding

begin

    process (clk, reset)
    begin
  if rising_edge(clk) then
   if reset = '1' then
    state <= "000000001"; -- Reset to state A
   else
    -- Logic expressions for state transitions
    state(0) <= '0'; -- State A
    state(1) <= (state(0) and not w) or (state(5) and not w) or (state(6) and not w) or (state(7) and not w) or (state(8) and not w);      -- State B
    state(2) <= (state(1) and not w);                -- State C
    state(3) <= (state(2) and not w);                -- State D
    state(4) <= (state(3) and not w) or (state(4) and not w);           -- State E (z = 1)
    state(5) <= (state(0) and w) or (state(1) and w) or (state(2) and w) or (state(3) and w) or (state(4) and w);                    -- State F
    state(6) <= (state(5) and w);                          -- State G
    state(7) <= (state(6) and w);                          -- State H
    state(8) <= (state(7) and w) or (state(8) and w);      -- State I (z = 1)
        end if;
      end if;
    end process;

    -- Assign outputs
    y <= state; -- Debugging output for current state

    -- Output z logic
    z <= state(4) or state(8); -- z = 1 when in state E or I

end Behavioral;
```

One-hot modification just changed the logical part into,

```
state(0) <= '1'; -- State A
state(1) <= (not state(0) and not w) or (state(5) and not w) or (state(6) and not w) or (state(7) and not w) or (state(8) and not w);      -- State B
state(2) <= (state(1) and not w);
             -- State C
state(3) <= (state(2) and not w);
             -- State D
state(4) <= (state(3) and not w) or (state(4) and not w);                    -- State E (z = 1)
state(5) <= (not state(0) and w) or (state(1) and w) or (state(2) and w) or (state(3) and w) or (state(4) and w);
                -- State F
state(6) <= (state(5) and w);
                 -- State G
state(7) <= (state(6) and w);
                 -- State H
state(8) <= (state(7) and w) or (state(8) and w);                    -- State I (z = 1)
```

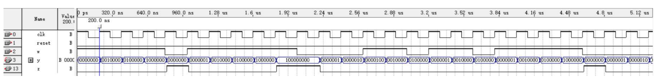
Part II

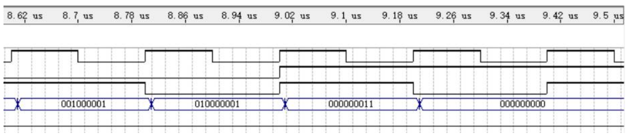
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY FSM3 IS
   PORT (
       clk : IN STD_LOGIC;                 -- Clock input
       reset : IN STD_LOGIC;               -- Active-low reset
       w : IN STD_LOGIC;                   -- Input signal
       z : OUT STD_LOGIC;                  -- Output signal
        leds : OUT STD_LOGIC_VECTOR(8 DOWNTO 0) -- State display on LEDs
    );
END FSM3;

ARCHITECTURE Behavior OF FSM3 IS
    -- Declare state type
    TYPE State_type IS (A, B, C, D, E, F, G, H, I);
    SIGNAL y_Q, y_D : State_type; -- y_Q: Present state, y_D: Next state
BEGIN

    -- State transition logic (State table)
    PROCESS (w, y_Q)
    BEGIN
        CASE y_Q IS
            WHEN A =>
                IF w = '0' THEN
                    y_D <= B;
                ELSE
                    y_D <= F;
                END IF;
            WHEN B =>
                IF w = '0' THEN
                    y_D <= C;
```

```vhdl
                ELSE
                    y_D <= F;
                END IF;
            WHEN C =>
                IF w = '0' THEN
                    y_D <= D;
                ELSE
                    y_D <= F;
                END IF;
            WHEN D =>
                IF w = '0' THEN
                    y_D <= E;
                ELSE
                    y_D <= F;
                END IF;
            WHEN E =>
                IF w = '0' THEN
                    y_D <= E;
                ELSE
                    y_D <= F;
                END IF;
            WHEN F =>
                IF w = '1' THEN
                    y_D <= G;
                ELSE
                    y_D <= B;
                END IF;
            WHEN G =>
                IF w = '1' THEN
                    y_D <= H;
                ELSE
                    y_D <= B;
                END IF;
            WHEN H =>
                IF w = '1' THEN
                    y_D <= I;
                ELSE
                    y_D <= B;
                END IF;
            WHEN I =>
                IF w = '1' THEN
                    y_D <= I;
                ELSE
                    y_D <= B;
                END IF;
            WHEN OTHERS =>
                y_D <= A; -- Default state
        END CASE;
    END PROCESS;

    -- State flip-flops (Sequential logic)
    PROCESS (clk, reset)
    BEGIN
        IF rising_edge(clk) THEN
            IF reset = '1' THEN
    y_Q <= A; -- Reset to initial state
            ELSE
    y_Q <= y_D; -- Update to next state
   END IF;
        END IF;
    END PROCESS;

    -- Output logic for z
    z <= '1' WHEN (y_Q = E OR y_Q = I) ELSE '0';

    -- Assign LEDs to display the current state
    leds <= "000000001" WHEN y_Q = A ELSE
            "000000010" WHEN y_Q = B ELSE
            "000000100" WHEN y_Q = C ELSE
            "000001000" WHEN y_Q = D ELSE
            "000010000" WHEN y_Q = E ELSE
            "000100000" WHEN y_Q = F ELSE
            "001000000" WHEN y_Q = G ELSE
            "010000000" WHEN y_Q = H ELSE
            "100000000";

END Behavior;

```

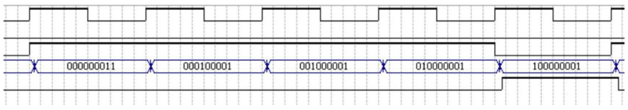
4. Simulation and Synthesis Results

Part I


No reset



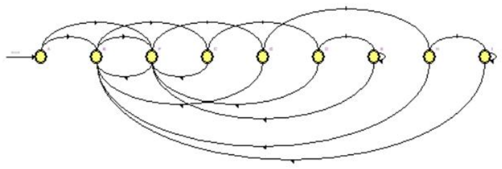
With reset

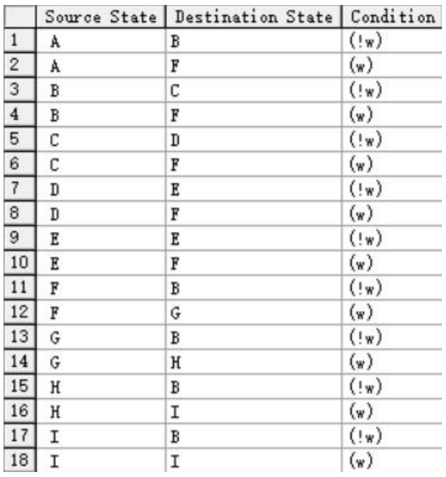


Modified

Part II

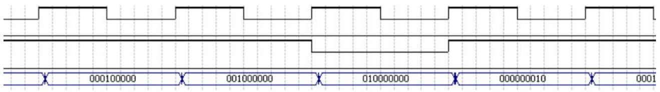
State Machine



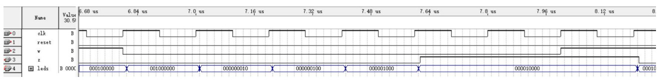


	Source State	Destination State	Condition
1	A	B	(!w)
2	A	F	(w)
3	B	C	(!w)
4	B	F	(w)
5	C	D	(!w)
6	C	F	(w)
7	D	E	(!w)
8	D	F	(w)
9	E	E	(!w)
10	E	F	(w)
11	F	B	(!w)
12	F	G	(w)
13	G	B	(!w)
14	G	H	(w)
15	H	B	(!w)
16	H	I	(w)
17	I	B	(!w)
18	I	I	(w)

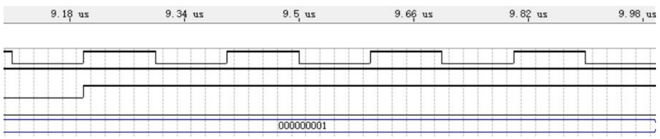
Vector Waveform



Not-activated with no reset



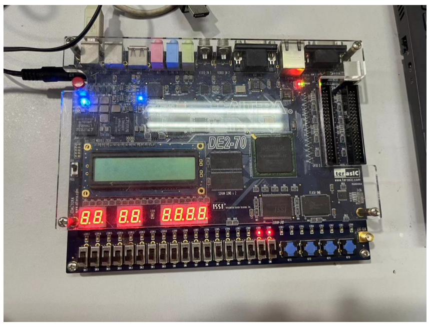
Typical activation

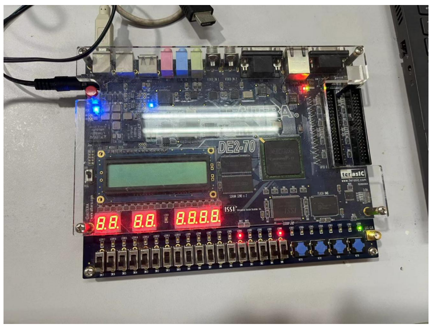


Upon reset

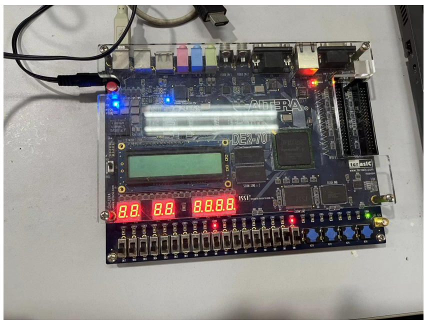
5. Experimental Results

For any of the FSM, whether implemented through logic statements or state machine, these result elaborate,

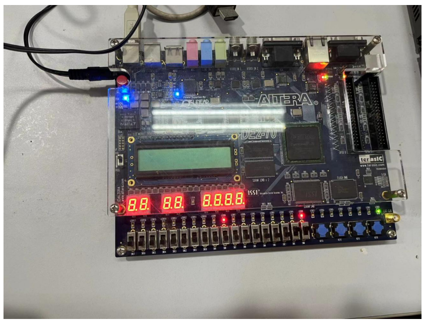


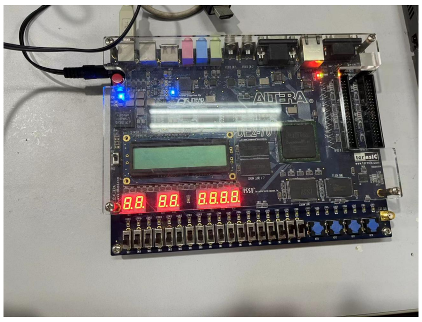


Left lane in the state machine result after 4 cycle



Right lane





Reset works in a synchronous manner, which resets the machine only in the very next cycle.

6. Discussion and Conclusion

In the initial design, the reset signal for the FSM was implemented as asynchronous, meaning that as soon as the reset signal was asserted, the machine’s present state (y_Q) was immediately forced into the reset state (typically state A), regardless of the clock. However, this approach introduced an inconsistency because the FSM outputs and state transitions are inherently synchronous, relying on the rising edge of the clock to propagate changes. As a result, while the reset forced the state immediately, the actual behavior and outputs of the FSM were only updated on the next clock cycle. This mismatch created ambiguity, especially in how the FSM responded when the reset signal was de-asserted close to a clock edge.

To resolve this issue, the FSM reset was changed to a synchronous reset, where the reset signal is sampled and takes effect only on the rising edge of the clock. This ensures that both the state transitions and outputs remain synchronized with the clock signal, leading to more predictable and stable behavior. By aligning the reset with the clock cycle, the FSM avoids potential glitches, eliminates timing hazards, and ensures that outputs such as z and LEDs are consistent with the state transitions. The synchronous reset provides a cleaner and more reliable implementation, particularly when the FSM is integrated into clock-driven digital systems like FPGAs.