VLSI System Design

ELE301P

LAB - 6 - Report

Praveen B R

COE19B007

Submission Date: 20/10/2021

INDEX

- Q1) Finite State Machine
  - Objective
  - Overlapping sequences are allowed
    - Theory
    - Code
    - Output/Waveform
  - Overlapping sequences are not allowed
    - Theory
    - Code
    - Output/Waveform
  - Conclusion

- Miscellaneous Questions
  - Question 1
  - Question 2
  - Question 3

Q1) Finite State Machine

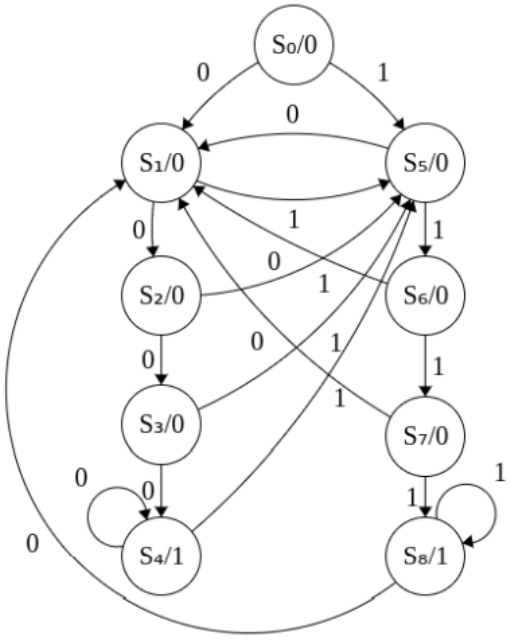
Objective:

To implement a finite state machine (FSM) that recognizes two specific sequences of applied input symbols, namely four consecutive 1s or four consecutive 0s. There is an input w and an output z. Whenever w= 1 or w = 0 for four consecutive clock pulses the value of z has to be 1; otherwise, z = 0.

Overlapping Sequences are allowed:

Theory

Overlapping sequences are allowed, so that if w = 1 for five consecutive clock pulses the output z will be equal to 1 after the fourth and fifth pulses.



Finite state diagram

Code

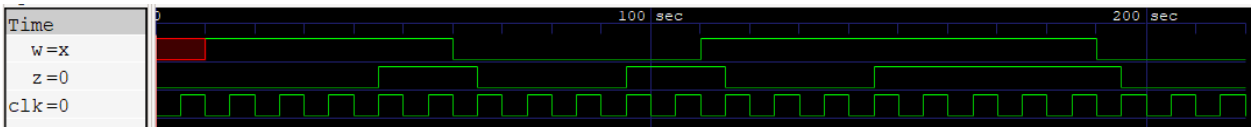
```
module Flip_Flop(next_state,w,clk,rst);
    input clk,rst,w;
    output reg [3:0]next_state;

    reg [3:0]state;
    parameter S0 = 4'b0000 ,
              S1 = 4'b0101 ,
```

```
              S2 = 4'b0110 ,
              S3 = 4'b0111 ,
              S4 = 4'b1000 ,
              S5 = 4'b0001 ,
              S6 = 4'b0010 ,
              S7 = 4'b0011 ,
              S8 = 4'b0100 ;

    always @(posedge clk , negedge rst)
    begin
        if(rst == 1'b0)
        begin
            state      = S0;
            next_state = S0;
        end

        else
        begin
            case(state)
                S0 : if(w) next_state = S1 ; else next_state = S5;
                S5 : if(w) next_state = S1 ; else next_state = S6;
                S6 : if(w) next_state = S1 ; else next_state = S7;
                S7 : if(w) next_state = S1 ; else next_state = S8;
                S8 : if(w) next_state = S1 ; else next_state = S8;
                S1 : if(w) next_state = S2 ; else next_state = S5;
                S2 : if(w) next_state = S3 ; else next_state = S5;
                S3 : if(w) next_state = S4 ; else next_state = S5;
                S4 : if(w) next_state = S4 ; else next_state = S5;
            endcase
            state <= next_state;
        end
    end
endmodule

module Sequence_Detector(z,w,clk,rst);
    output z;
    output [3:0]A;
    input w,clk,rst;

    //wire [3:0]A;
```

```
    wire a3,a2,a1,a0;
    wire p,q;

    Flip_Flop FF1(A,w,clk,rst);

    xor X1(a3,A[3],1'b1);
    xor X2(a2,A[2],1'b1);
    xor X3(a1,A[1],1'b1);
    xor X4(a0,A[0],1'b1);

    and A1(p,A[3],a2,a1,a0);
    and A2(q,a3,A[2],a1,a0);
    or O1(z,p,q);

endmodule

module q1a_tb();
    wire z;
    wire [3:0]A;
    reg w,clk,rst;
    Sequence_Detector S(z,w,clk,rst);

    initial begin
        clk = 1'b0;
        forever #5 clk = ~clk;
    end

    initial begin
            rst = 1'b0 ;
            #10 rst = 1'b1 ; w = 1'b1;
            #10 rst = 1'b1 ; w = 1'b1;
            #10 rst = 1'b1 ; w = 1'b1;
            #10 rst = 1'b1 ; w = 1'b1;
            #10 rst = 1'b1 ; w = 1'b1;
            #10 rst = 1'b1 ; w = 1'b0;
            #10 rst = 1'b1 ; w = 1'b0;
            #10 rst = 1'b1 ; w = 1'b0;
            #10 rst = 1'b1 ; w = 1'b0;
            #10 rst = 1'b1 ; w = 1'b0;
            #10 rst = 1'b1 ; w = 1'b1;
```

```
            #10 rst = 1'b1 ; w = 1'b1;
            #10 rst = 1'b1 ; w = 1'b1;
            #10 rst = 1'b1 ; w = 1'b1;
            #10 rst = 1'b1 ; w = 1'b1;
            #10 rst = 1'b1 ; w = 1'b1;
            #10 rst = 1'b1 ; w = 1'b1;
            #10 rst = 1'b1 ; w = 1'b1;
            #10 rst = 1'b1 ; w = 1'b0;
            #10 rst = 1'b1 ; w = 1'b0;
    end


    initial begin
        forever
            #10 $display("%d :  z = %b , w = %b , clk = %b , rst = %b
",$time,z,w,clk,rst);
    end

    initial begin
        #220 $finish;
    end

    initial
    begin
        $dumpfile("q1a.vcd");
        $dumpvars(0,q1a_tb);
    end
endmodule
```

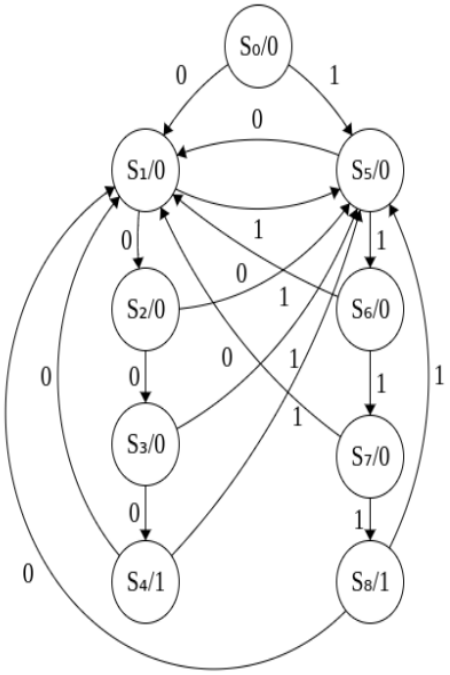
Output/Waveform



Overlapping Sequences are not allowed:

Theory

Overlapping sequences are not allowed, so that if w = 1 for five consecutive clock pulses the output z will be equal to 1 and 0 after the fourth and fifth pulses respectively.



Finite State Diagram

Code

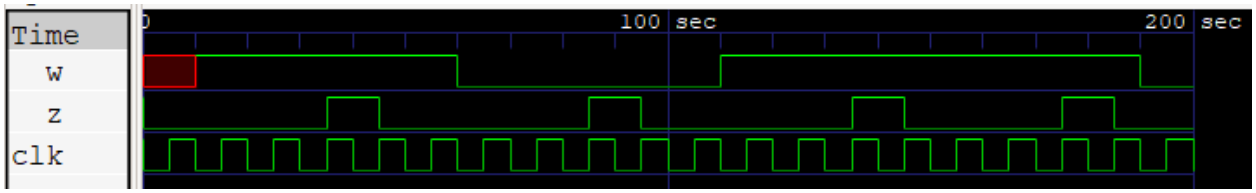
```
module Sequence_Detector(z,w,clk,rst);
    output reg z;
    input w,clk,rst;

    reg[4:0] state,next_state;
    parameter S0 = 4'b0000 ,
              S1 = 4'b0101 ,
              S2 = 4'b0110 ,
              S3 = 4'b0111 ,
```

```
              S4 = 4'b1000 ,
              S5 = 4'b0001 ,
              S6 = 4'b0010 ,
              S7 = 4'b0011 ,
              S8 = 4'b0100 ;

    always @(posedge clk , negedge rst)
    begin
        if(rst == 1'b0)
            state <= S0;
        else
            state <= next_state;
    end

    always @(state,w)
    begin
        case(state)
            S0 : if(w) next_state = S1; else next_state = S5;
            S1 : if(w) next_state = S2; else next_state = S5;
            S2 : if(w) next_state = S3; else next_state = S5;
            S3 : if(w) next_state = S4; else next_state = S5;
            S4 : if(w) next_state = S1; else next_state = S0;
            S5 : if(w) next_state = S1; else next_state = S6;
            S6 : if(w) next_state = S1; else next_state = S7;
            S7 : if(w) next_state = S1; else next_state = S8;
            S8 : if(w) next_state = S1; else next_state = S0;
        endcase
    end

    always @(state,w)
    begin
        case(state)
            S7 : if(w == 1'b0) z = 1'b1; else z = 1'b0;
            S3 : if(w)  z = 1'b1; else z = 1'b0;
            S0,S1,S2,S4,S5,S6,S8 : z = 1'b0;
        endcase
    end
endmodule
```

```
module q1b_tb();
    wire z;
    reg w,clk,rst;
    Sequence_Detector S(z,w,clk,rst);

    initial begin
        clk = 1'b0;
        forever #5 clk = ~clk;
    end

    initial begin
            rst = 1'b0 ;

        #10 rst = 1'b1 ; w = 1'b1;
        #10 rst = 1'b1 ; w = 1'b1;
        #10 rst = 1'b1 ; w = 1'b1;
        #10 rst = 1'b1 ; w = 1'b1;
        #10 rst = 1'b1 ; w = 1'b1;
        #10 rst = 1'b1 ; w = 1'b0;
        #10 rst = 1'b1 ; w = 1'b0;
        #10 rst = 1'b1 ; w = 1'b0;
        #10 rst = 1'b1 ; w = 1'b0;
        #10 rst = 1'b1 ; w = 1'b0;
        #10 rst = 1'b1 ; w = 1'b1;
        #10 rst = 1'b1 ; w = 1'b1;
        #10 rst = 1'b1 ; w = 1'b1;
        #10 rst = 1'b1 ; w = 1'b1;
        #10 rst = 1'b1 ; w = 1'b1;
        #10 rst = 1'b1 ; w = 1'b1;
        #10 rst = 1'b1 ; w = 1'b1;
        #10 rst = 1'b1 ; w = 1'b1;
        #10 rst = 1'b1 ; w = 1'b0;
        #10 rst = 1'b1 ; w = 1'b0;


    end

    initial begin
        forever
```

```
            #10 $display("%d : z = %b , w = %b , clk = %b , rst = %b
",$time,z,w,clk,rst);
    end

    initial begin
        #200 $finish;
    end
    initial
    begin
        $dumpfile("q1b.vcd");
        $dumpvars(0,q1b_tb);
    end
endmodule
```

Output/Waveform



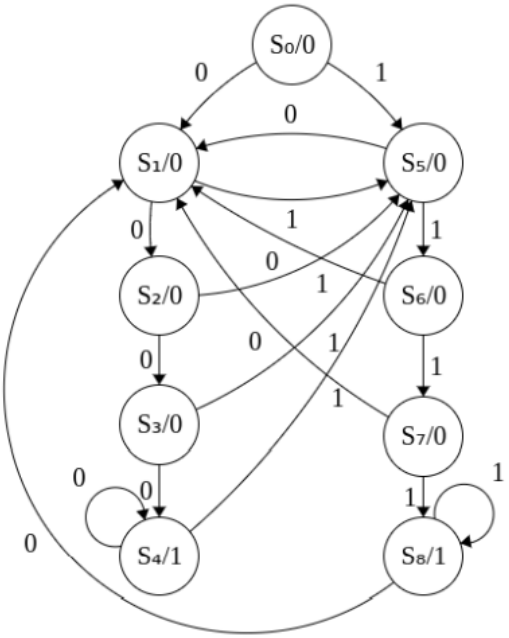
Conclusion:

Thus, Finite State Machines have been successfully implemented in verilog such that it recognizes two specific sequences of applied input symbols, namely four consecutive 1s or four consecutive 0s. There is an input w and an output z. Whenever w= 1 or w = 0 for four consecutive clock pulses the value of z has to be 1;
otherwise, z = 0.

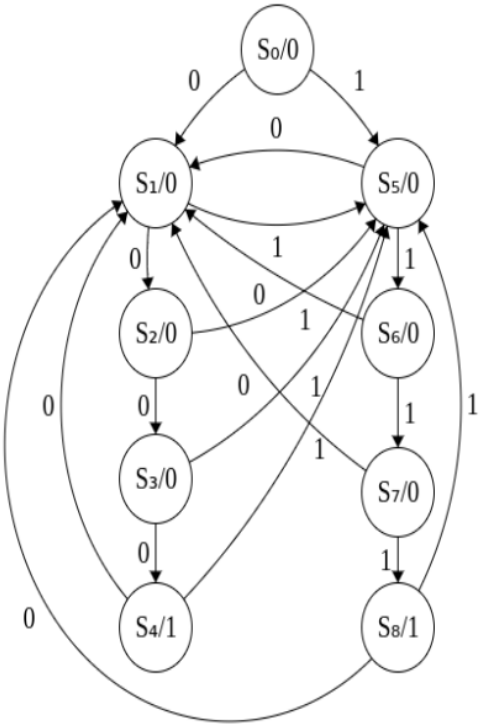
Miscellaneous Questions

Question 1 State Diagram for the above problem statement

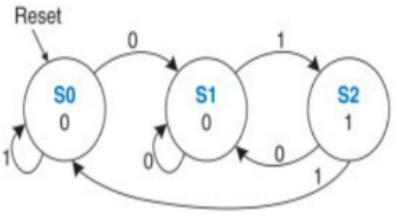
Overlapping:



Non-Overlapping:



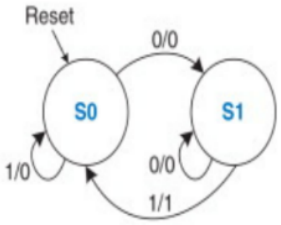
Question 2



a) What is the type of FSM used in the above state diagram?
Moore Type
Reason:
Clearly,outputs of the FSM are given **inside** the states so we can say that it only depends on the current state. So the given FSM is of Moore type.

b) Find the sequence for which we will get outputs.
The given FSM would give high for the input strings **ending with 01**
So, to get a high output, inputs can be 01,001,101,0001,0101,1001,1101,... and so on.
Rest all inputs would give a low output.

Question 3



a) What is the type of FSM used in the following state diagram?
Mealy Type
Reason:
Clearly,outputs of the FSM are given **outside** the states so we can say that it depends on the current state and input. So the given FSM is of Mealy type.

b) Find the sequence for which we will get outputs.
The given FSM would give high for the input strings **ending with 01**
So, to get a high output, inputs can be 01,001,101,0001,0101,1001,1101,... and so on.
Rest all inputs would give a low output.