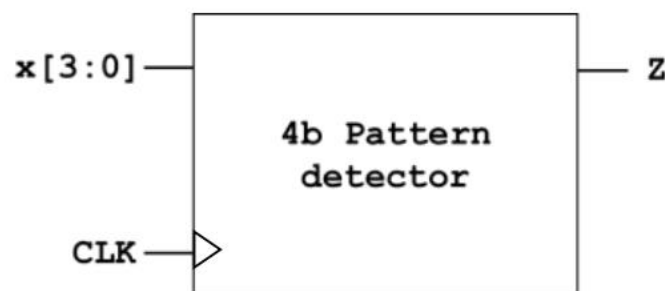
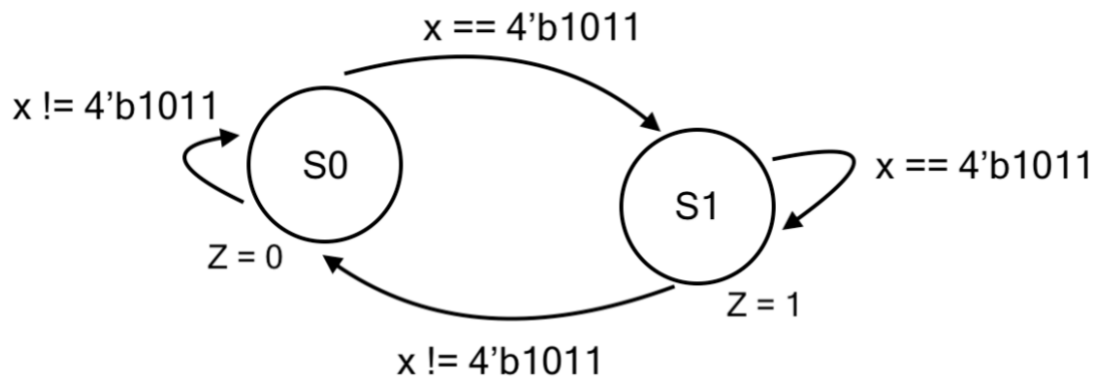


## Lab 7: A Simple Finite State Machine

Today's lab is about building a simple Finite State Machine that will detect 4b patterns as its input from a stream of 4b packets. Here at every positive edge of the clock CLK, it samples a 4b input vector from the input port  $x[3:0]$  into the machine. Before we talk about the detector, we need to understand how a simple finite state machine works. A Finite State Machine are most widely used sequential logic circuit that contains a predefined number of states. The machine can exist in one and only one state at a time, and the circuit makes transition between states based on a triggering event, for example the positive edge of the clock with the values of any inputs of the machine. Using the states and predefined sequence of transitions, the circuit is able to make decisions on the next state to transition based on the history of past states. So in this lab we try to create a simple Finite State Machine that can detect a 4b pattern from its input. First we will try to write Verilog module that has an input vector ' $x[3:0]$ ' and an input 'CLK' and an output 'z' that sets to '1' if it detects the pattern '1011' in each 4b packet it receives at the input 'x'.



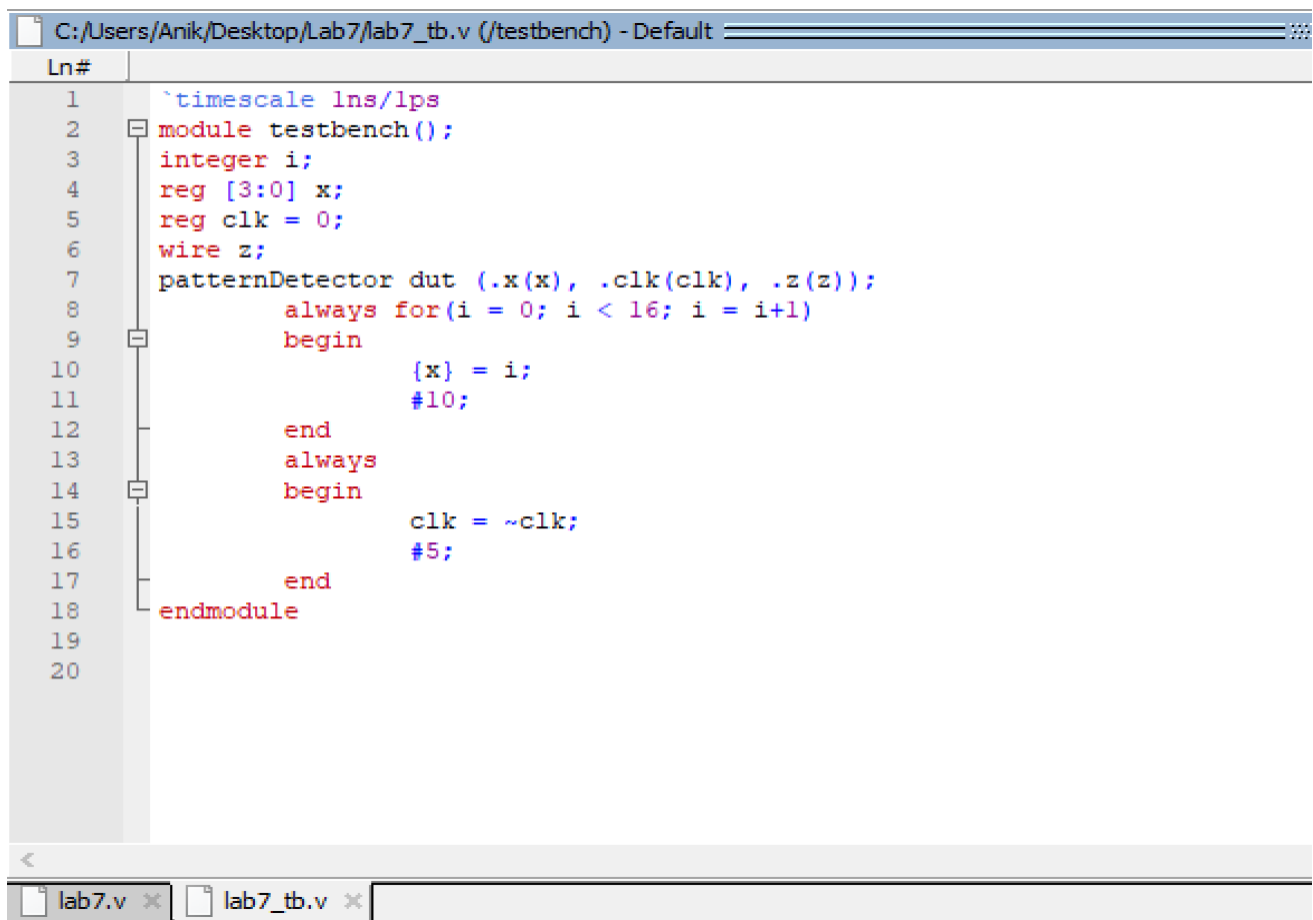
Before we write the module, we need to make a state diagram to understand how this state machine will operate. We need 2 states for this state machine, S0 with  $z = 0$  and S1 with  $z = 1$ .



At every positive edge of the clock, if our input X matches with '1011', we go to state 1 (Z = 1). Until we get a different 4'b pattern, we will stay at State 1 and when it doesn't match we go back to state 0. In state 0 if the next X value is not 1011, we stay at state 0. Now we will try to implement it for the extended version where output 'z' will go to '1' if it detects any of the following 4b packets at its vector input port 'x': '1011', '0110' and '0100'. It's basically same as checking for 1011 but now we are checking for 3 of them together. So on Verilog module - we

Ln#	
1	module patternDetector (input [3:0]x, input clk, output reg z);
2	parameter s0 = 1'b0, s1 = 1'b1;
3	reg [3:0] transition;
4	reg state = s0;
5	always@ (posedge clk)
6	begin
7	transition <= x;
8	end
9	always@ (state)
10	begin
11	if(state == s0)
12	z <= 0;
13	else
14	z <= 1;
15	end
16	always@ (transition)
17	begin
18	if((transition == 4'b1011)    (transition == 4'b0110)    (transition == 4'b0100))
19	state <= s1;
20	else
21	state <= s0;
22	end
23	endmodule

have input X, input CLK and output reg z. We can declare the states as parameter as s0 = 1'b0 and s1 = 1'b1. Next we have a 4 bit transition variable that will work as a temporary variable as the reference to the value of x and we declare a state variable and initialize it to S0 (Z=0). Now using an always block, at every positive edge of the clock, we set transition value to x. Next we have to take care of the output z. So using an always block for state if state is S0, we set z = 0 else we set z = 1. Now at each transition we have to check for the 4b patterns. So we use a always block for transition and using an if-else statement if our transition matches with the pattern we set state = S1, else state = S0. That's our Verilog module to detect the 4'b patterns. Now we need to create a testbench that will drive the 4b packets to input 'x' and we can verify the module's functionality.

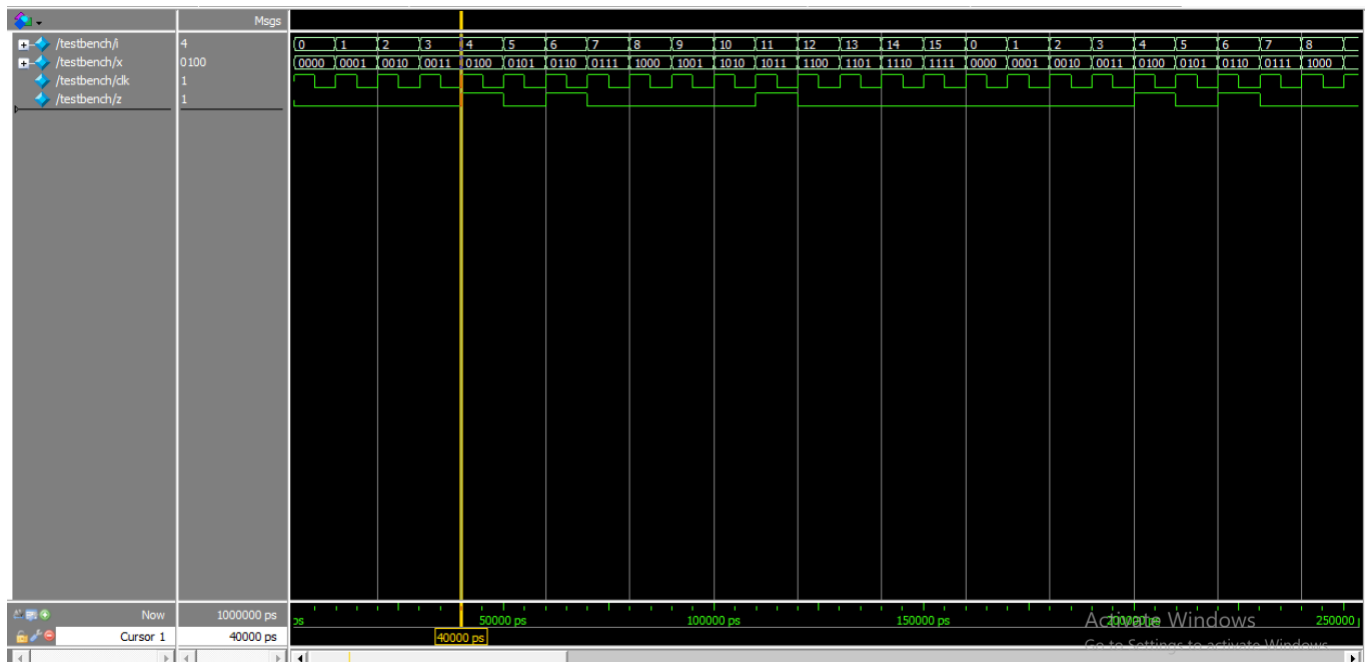


```
C:/Users/Anik/Desktop/Lab7/lab7_tb.v (/testbench) - Default
Ln#
1  `timescale 1ns/1ps
2  module testbench();
3      integer i;
4      reg [3:0] x;
5      reg clk = 0;
6      wire z;
7      patternDetector dut (.x(x), .clk(clk), .z(z));
8      always for(i = 0; i < 16; i = i+1)
9      begin
10         {x} = i;
11         #10;
12     end
13     always
14     begin
15         clk = ~clk;
16         #5;
17     end
18 endmodule
19
20
```

lab7.v x lab7\_tb.v x

For the testbench, we declare a reg x (4 bit), clk set to 0, and wire z which is the output. Now we map them to our patternDetector module following the same input and output sequence. Next using an always block, we have to drive 4b packets to input 'x'. To do that we can use a for loop and since it's a 4 bit, we have  $2^4 = 16$  possible packets that will be our input value for x. For each value of  $x = i$ , we will put a #10 delay. Now using another always block, we have to negate the clk value after every #5 delay. That's our testbench and after simulating it we will get our waveform.

Waveform -



From the waveform, we can verify that our module functionality works. When x is '1011', '0110' and '0100', we can see the our z value goes to 1 otherwise it goes to 0. It happens on the positive edge of the clock and waits till the next positive edge to go to 0 if it sees a different 4'b pattern. So our simple Finite State Machine can detect the specified 4b patterns at its input from a stream of 4b packets.

Extra Credit -

Detect a 16b pattern '0011 1001 1100 0001' from the input port x [15:0]

- For this we need to change our input x that will now take 16 bit number. The reg transition will also be 16 bit and for if-else, we need to check for the 16b pattern '0011 1001 1100 0001' inside the always block.

```
C:\Users\Anik\Desktop\Lab7EC\Lab7EC.v - Default
Ln#
1 module patternDetector (input [15:0]x, input clk, output reg z);
2   parameter s0 = 1'b0, s1 = 1'b1;
3   reg [15:0] transition;
4   reg state = s0;
5       always@ (posedge clk)
6       begin
7           transition <= x;
8       end
9       always@ (state)
10      begin
11          if(state == s0)
12              z <= 0;
13          else
14              z <= 1;
15      end
16      always@ (transition)
17      begin
18          if(transition == 16'b0011100111000001)
19              state <= s1;
20          else
21              state <= s0;
22      end
23  endmodule
```

Now in testbench, we need to change the reg x to 16 bit and the for-loop. Since it's a 16 bit, we have  $2^{16} = 65536$  possible values for x. Other than that everything will be same and we can simulate this to verify it works.

```

C:/Users/Anik/Desktop/Lab7EC/Lab7EC_tb.v (/testbench) - Default
Ln#
1  `timescale 1ns/1ps
2  module testbench();
3      integer i;
4      reg [15:0] x;
5      reg clk = 0;
6      wire z;
7      patternDetector dut (.x(x), .clk(clk), .z(z));
8          always for(i = 0; i < 65536; i = i+1)
9              begin
10                  {x} = i;
11                  #10;
12              end
13          always
14              begin
15                  clk = ~clk;
16                  #5;
17              end
18      endmodule
19
20

```

Waveform - From the waveform we can see that when x is '0011 1001 1100 0001', our z value goes to 1. So this verifies that it can detect a 16b pattern '0011 1001 1100 0001' from the input port x [15:0].

