Department of Electrical Engineering and Technology

College of Engineering and Technology

Mindanao State University – Iligan Institute of Technology

.

Laboratory Report 3 - Pattern Identifier

In partial fulfillment for the course

ECE 134.1 —Introduction to Digital VLSI Design Laboratory

Submitted to

Prof. Kevin O. Maglinte

Faculty, DEET

COET, MSU – IIT

Submitted by

Bryan Jeff C. Saycon

2018-0807

BS ECE – IV

INTRODUCTION

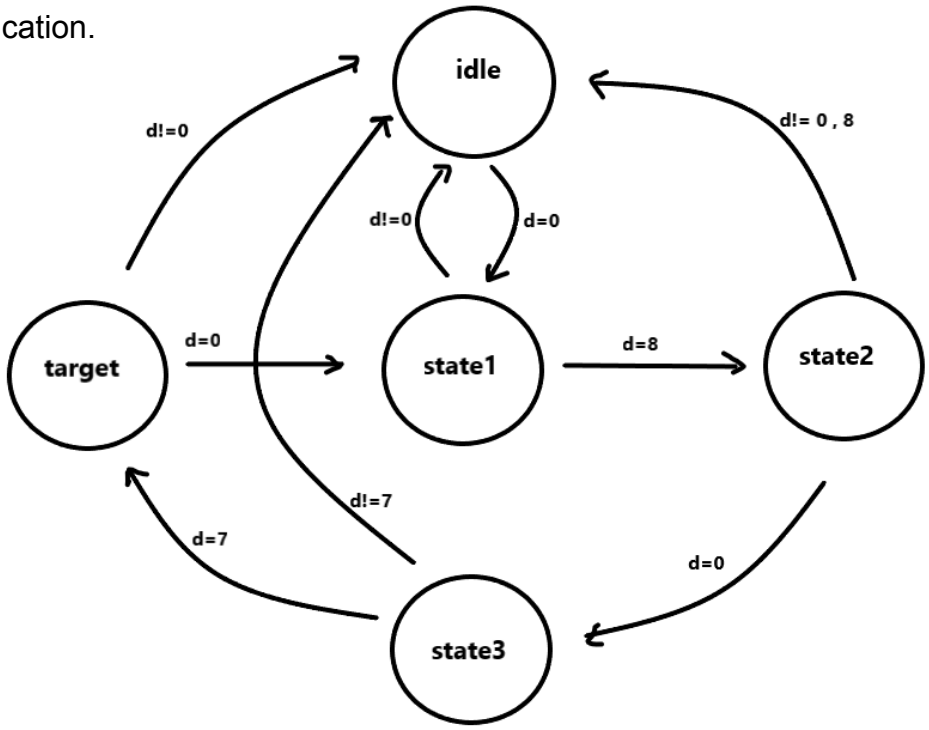
Verilog is a Hardware Description Language (HDL). It is a language used for describing a digital system like a network switch or a microprocessor or a memory or a flip−flop. It means, by using a HDL we can describe any digital hardware at any level. Designs, which are described in HDL are independent of technology, very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits. In this particular activity, we design a Pattern Identifier uniquely patterned to the last 4 digits of the ID Number using Quartus Prime Lite Edition using the lessons discussed in the Laboratory.

OBJECTIVES

Use Quartus Prime Lite Edition and ModelSim in coding (in Verilog HDL format), compiling, and simulation of a system.
Program a Pattern Identifier unique to the last 4 digits of ID Number.
Discuss the simulation process using screen record and voice-over.

FSM DIAGRAM

In this FSM diagram *d* variable is the *data_in* component in verilog code, for simplification.



PATTERN IDENTIFIER MAIN MODULE CODE

pattern_identifier.v

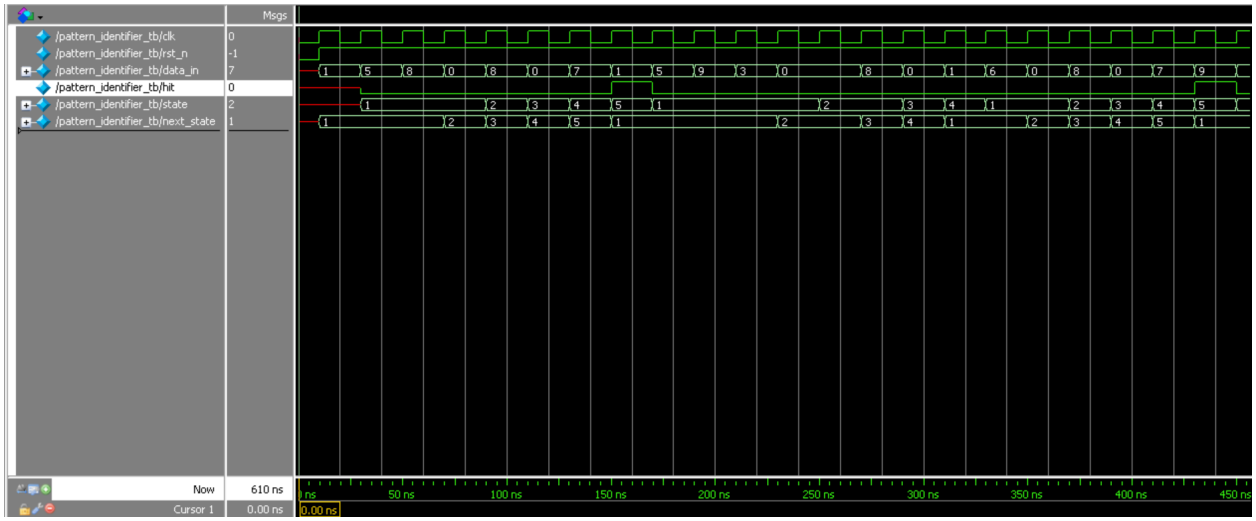
```
module pattern_identifier (clk,
                           state,
                           data_in,
                           rst_n,
                           hit,
                           next_state);
input clk;
input data_in;
input rst_n;

output hit;
output [4:0] state;
output [4:0] next_state;

wire clk;
wire rst_n;
wire [4:0] data_in;

reg [4:0] next_state;
reg [4:0] state;
reg hit;

parameter [4:0] idle = 4'd1;
parameter [4:0] state1 = 4'd2;
parameter [4:0] state2 = 4'd3;
parameter [4:0] state3 = 4'd4;
parameter [4:0] target = 4'd5;

always @(*)
begin
case (state)
    idle: if (data_in == 0)
                begin next_state = state1;
                end
            else
                begin next_state = idle;
                end
    state1: if (data_in == 8)
                begin next_state = state2;
                end
            else if (data_in == 0)
                        begin next_state = state1;
                        end
            else
                begin next_state = idle;
                end
    state2: if (data_in == 0)
                begin next_state = state3;
                end
            else
                begin next_state = idle;
                end
    state3: if (data_in == 7)
                begin next_state = target;
                end
            else if (data_in == 0)
                begin next_state = state1;
                end
                else
                begin next_state = idle;
                end
    target: if (data_in == 0)
                begin next_state = state1;
                end
            else
                begin next_state = idle;
                end
    default: next_state = idle;

endcase
end

always @(posedge clk)
begin
    if(!rst_n)
        state <= idle;
    else
        state <= next_state;
end

always @ (*)
begin
    case (state)
        idle: hit = 1'h0;
        state1: hit = 1'h0;
        state2: hit = 1'h0;
        state3: hit = 1'h0;
        target: hit = 1'h1;
        default: hit = 1'h0;
    endcase
end

endmodule
```

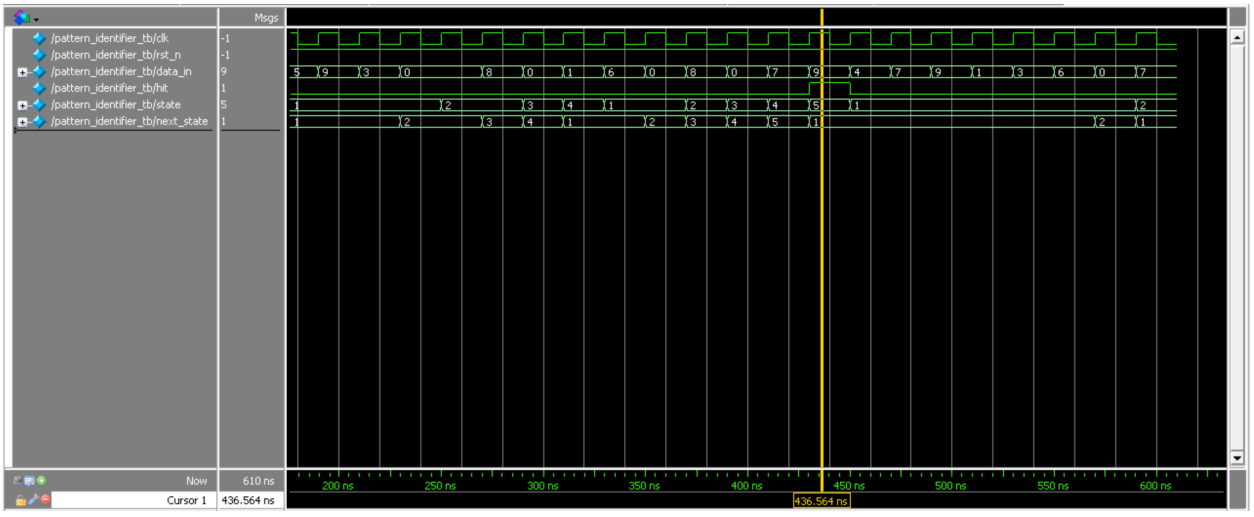
TEST BENCH CODE

pattern_identifier_tb.v

```
1   `timescale 1ns/1ps
2
3   module pattern_identifier_tb();
4
5   reg clk;
6   reg rst_n;
7   reg [4:0] data_in;
8
9   wire hit;
10  wire [4:0] state;
11  wire [4:0] next_state;
12
13  initial
14  begin
15      clk = 0;
16      rst_n = 0;
17
18  #10 data_in = 1;
19      rst_n = 1;
20
21  #20 data_in = 5;
22  #20 data_in = 8;
23  #20 data_in = 0;
24  #20 data_in = 8;
25  #20 data_in = 0;
26  #20 data_in = 7;
27  #20 data_in = 1;
28  #20 data_in = 5;
29  #20 data_in = 9;
30  #20 data_in = 3;
31  #20 data_in = 0;
32  #20 data_in = 0;
33  #20 data_in = 8;
34  #20 data_in = 0;
35  #20 data_in = 1;
36  #20 data_in = 6;
37  #20 data_in = 0;
38  #20 data_in = 8;
39  #20 data_in = 0;
40  #20 data_in = 7;
41  #20 data_in = 9;
42  #20 data_in = 4;
43  #20 data_in = 7;
44  #20 data_in = 9;
45  #20 data_in = 1;
46  #20 data_in = 3;
47  #20 data_in = 6;
48  #20 data_in = 0;
49  #20 data_in = 7;
50  #20 $finish;
51  end
52
53  always #10 clk = ~clk;
54  pattern_identifier  dut(.clk(clk),
55                          .state(state),
56                          .data_in(data_in),
57                          .hit(hit),
58                          .rst_n(rst_n),
59                          .next_state(next_state));
60  endmodule
```

RESULTS


Msgs
/pattern_identifier_tb/clk
/pattern_identifier_tb/rst_n
/pattern_identifier_tb/data_in
/pattern_identifier_tb/hit
/pattern_identifier_tb/state
/pattern_identifier_tb/next_state
Now
610 ns
Cursor 1
0.00 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns
350 ns
400 ns
450 ns


Msgs
/pattern_identifier_tb/clk
/pattern_identifier_tb/rst_n
/pattern_identifier_tb/data_in
/pattern_identifier_tb/hit
/pattern_identifier_tb/state
/pattern_identifier_tb/next_state
Now
610 ns
Cursor 1
436.564 ns
200 ns
250 ns
300 ns
350 ns
400 ns
450 ns
500 ns
550 ns
600 ns

CONCLUSION

The program might seem a bit complicated considering it aims to detect a 4 digit pattern in consecutive order. Its algorithm is actually simpler than I thought, the system doesn't actually care about the previous digits no matter how many digits it actually ran to, it only emphasizes the current state and what state to transition to with the data entry the user inputs. With the conditions we set to identify each state in relay to the target hit, the algorithm only cares about the data entry and how the state reacts to the data inputted by the user.

LABORATORY QUESTION

What are the three main parts of a Finite State Machine code?

1. State
2. Actions
3. Transitions