CPEN 230L: Introduction to Digital Logic Laboratory
Lab #11: Finite State Machines
Scott Tornquist
12/3/2019

Part 1:

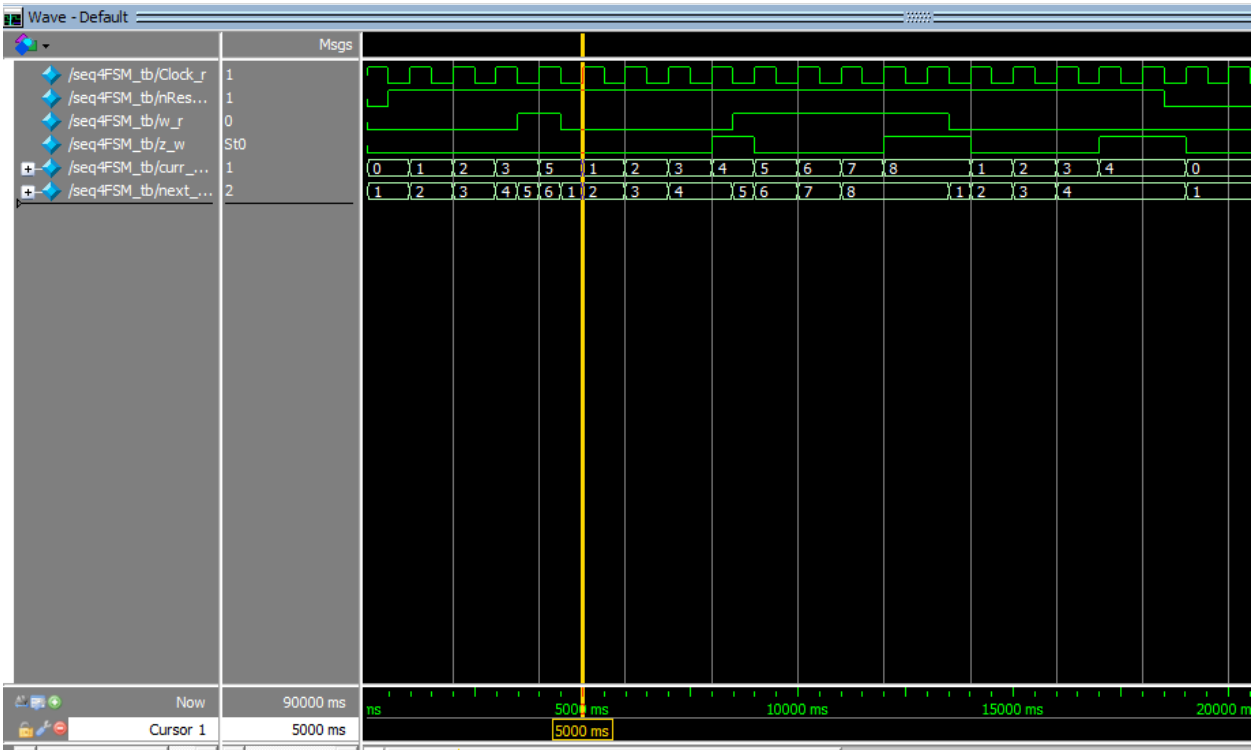
- Pre-lab work:
  - At t=4, w the current state is at 3, but W_r changes to 1 and this causes the next state to be 5 instead of 4, so there are not 4 zeros' in a row, so z is not triggered and remains 0
  - At t=8, the current state becomes 4 because there were 4 consecutive 0's in a row, so z is triggered and becomes 1
  - At t=12, the current state becomes 8 because there were 4 consecutive 1's in a row, so z is triggered and becomes 1
  - nReset is active low because at the beginning of the wave it is set to zero and the current state becomes 0, meaning it was reset by the active low nReset
- Procedure Changes:
  - None
- Results:

Seq4FSM_tb.v (Test Bench)

T:/Lab 11/SRC/seq4FSM_tb.v (/seq4FSM_tb) - Default

```
Ln#
21      $display("time  nReset  w  c_state  n_state  z");
22      $monitor("%4d  %6b  %1b  %7d  %7d  %1b",
23        $time, nReset_r, w_r, curr_state_w, next_state_w, z_w);
24
25                              // Test Procedure
26          Clock_r  = 1'b1;    // @t=0: Clock = 1 so +edges @t = 1, 2, 3...
27          nReset_r = 1'b0;    //       reset
28          w_r      = 1'b0;    //       main FSM input low
29      #0.5 nReset_r = 1'b1;   // @t=0.5 release reset
30
31      #3   w_r      = 1'b1;   // @t=3.5 w goes high, avoid z going high @t=4
32      #1   w_r      = 1'b0;   // @t=4.5 w goes low to start a seq of 4 0s
33                              // @t=8 z should go high, after seq of 4 0s
34      #4   w_r      = 1'b1;   // @t=8.5 w high (start long pulse)
35                              // @t=9 z should go low
36                              // @t=12 z should go high, after seq of 4 1s
37      #5   w_r      = 1'b0;   // @t=13.5 w low (end long pulse)
38                              // @t=14 z should go low
39                              // Next demonstrate the reset.
40      #5   nReset_r = 1'b0;   // @t=18.5 z has gone high, reset on clock -edge
41                              // @t=19 (next clock +edge) z should go low
42      #1.5 $finish;           // @t=20, finish
43    end
44
45    seq4FSM fsm (             // instantiate the DUT
46      .Clock      (Clock_r),
47      .nReset     (nReset_r),
48      .w          (w_r),
49      .z          (z_w),
50      .curr_state (curr_state_w),
51      .next_state (next_state_w) );
52
53  endmodule
54
```

Test Bench Wave Form



Test Bench Table Output

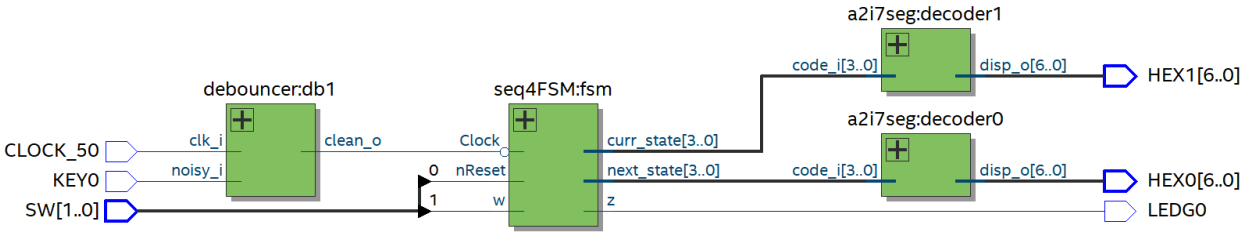
```
run
#     2        1  0        2        3  0
#     3        1  0        3        4  0
#     4        1  1        3        5  0
#     4        1  1        5        6  0
#     5        1  0        5        1  0
#     5        1  0        1        2  0
#     6        1  0        2        3  0
#     7        1  0        3        4  0
#     8        1  0        4        4  1
#     9        1  1        4        5  1
#     9        1  1        5        6  0
#    10        1  1        6        7  0
#    11        1  1        7        8  0
run
#    12        1  1        8        8  1
#    14        1  0        8        1  1
#    14        1  0        1        2  0
#    15        1  0        2        3  0
#    16        1  0        3        4  0
#    17        1  0        4        4  1
#    19        0  0        4        4  1
#    19        0  0        0        1  0
# ** Note: $finish    : T:/Lab 11/SRC/seq4FSM_tb.v(42)
#    Time: 20 sec  Iteration: 0  Instance: /seq4FSM_tb
```

seq4FSM_top.v

seq4FSM_top.v* | seq4FSM.v | debouncer.v | a2i7seg.v | Compilatio

```
1   // CPEN 230L lab 11, Sequence of 4 FSM on the DE2-115 board
2   // Scott Tornquist, 12/3/2019
3
4   module seq4FSM_top (
5     input        CLOCK_50,  // for debouncer
6     input        KEY0,      // Clock (FSM input)
7     input  [1:0] SW,        // w, nReset (FSM inputs)
8     output       LEDG0,     // z (FSM output)
9     output [6:0] HEX1,      // FSM current state (A b C d E F G H I)
10    output [6:0] HEX0 );    // FSM next state
11
12    wire KEY0_clean_w;        // debounced KEY0
13    wire [3:0] curr_state_w;  // current state
14    wire [3:0] next_state_w;  // next state
15
16    // Clean up noisy KEY0 with debounce settling of 1/50e6 s/period *
17    // 2^20 periods ~= 21.0 ms.
18    debouncer #(.cnt_bits (20)) db1 (
19      .clk_i   (CLOCK_50),
20      .noisy_i (KEY0),
21      .clean_o (KEY0_clean_w) );
22
23    seq4FSM fsm (
24      .Clock      (~KEY0_clean_w),  // key press = + edge
25      .nReset     (SW[0]),          // active low synchronous reset
26      .w          (SW[1]),          // main FSM input
27      .z          (LEDG0),
28      .curr_state (curr_state_w),
29      .next_state (next_state_w) );
30
31    a2i7seg decoder1 (      // current state to A..I on HEX1
32      .code_i  (curr_state_w),
33      .disp_o  (HEX1) );
34
35    a2i7seg decoder0 (      // next state to A..I on HEX0
36      .code_i  (next_state_w),
37      .disp_o  (HEX0) );
38
39  endmodule
40
```

RTL Viewer

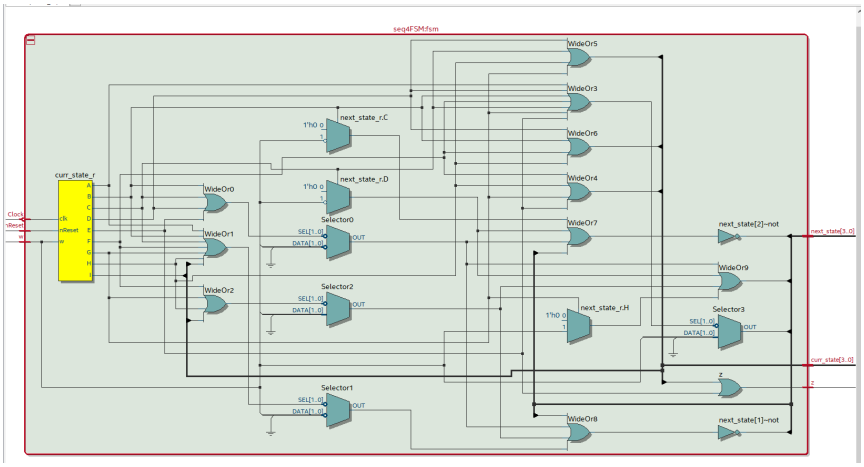


Seq4FSM.v

seq4FSM_top.v | seq4FSM.v | debouncer.v | a2i7seg.v | Compilation Rep

```
1   // CPEN 230L lab 11, Sequence of 4 0s or 1s Finite State Machine
2   // Scott Tornquist 12/3/2019
3
4   module seq4FSM (
5     input        Clock,  // posedge triggered
6     input        nReset, // active low synchronous reset
7     input        w,      // main FSM input
8     output       z,      // main output, 1 indicates a w seq of 4
9     output [3:0] curr_state,    // for demo/diagnostics
10    output [3:0] next_state );  // for demo/diagnostics
11
12    parameter  // state assignment (minimum flip-flops, not one-hot)
13      A = 4'd0, B = 4'd1, C = 4'd2, D = 4'd3, E = 4'd4,
14      F = 4'd5, G = 4'd6, H = 4'd7, I = 4'd8;
15
16    // registers for use in "always" block procedural assignments
17    reg [3:0] curr_state_r;
18    reg [3:0] next_state_r;
19
20    // See textbook Figure 6.29 on p356. Define the *combinational*
21    // circuit that defines next state, using *blocking* assignment "=".
22    always @(curr_state_r, w)
23      case (curr_state_r)
24        A: if (!w) next_state_r = B; else next_state_r = F;
25        B: if (!w) next_state_r = C; else next_state_r = F;
26        C: if (!w) next_state_r = D; else next_state_r = F;
27        D: if (!w) next_state_r = E; else next_state_r = F;
28        E: if (!w) next_state_r = E; else next_state_r = F;
29        F: if (!w) next_state_r = B; else next_state_r = G;
30        G: if (!w) next_state_r = B; else next_state_r = H;
31        H: if (!w) next_state_r = B; else next_state_r = I;
32        I: if (!w) next_state_r = B; else next_state_r = I;
33        // Write this part to complete the state diagram.
34        default:   next_state_r = 4'bxxxx;  // required, not all 16 used
35      endcase
36
37    // Define the *sequential* circuit implemented with flip-flops, using
38    // *non-blocking* assignment "<=".  The reset signal doesn't appear in
39    // the sensitivity list because we want a *synchronous* reset.
40    always @(posedge Clock)
41      // complete the always block
42      begin
43         if (!nReset)
44            curr_state_r <= A;
45         else
46            curr_state_r <= next_state_r;
47      end
48
49    // Define the *combinational* circuit that sets outputs based on the
50    // current state of the FSM.
51    assign z = ((curr_state_r == E) || (curr_state_r == I));
52    assign curr_state = curr_state_r;
53    assign next_state = next_state_r;
54
55  endmodule
```

RTL Viewer

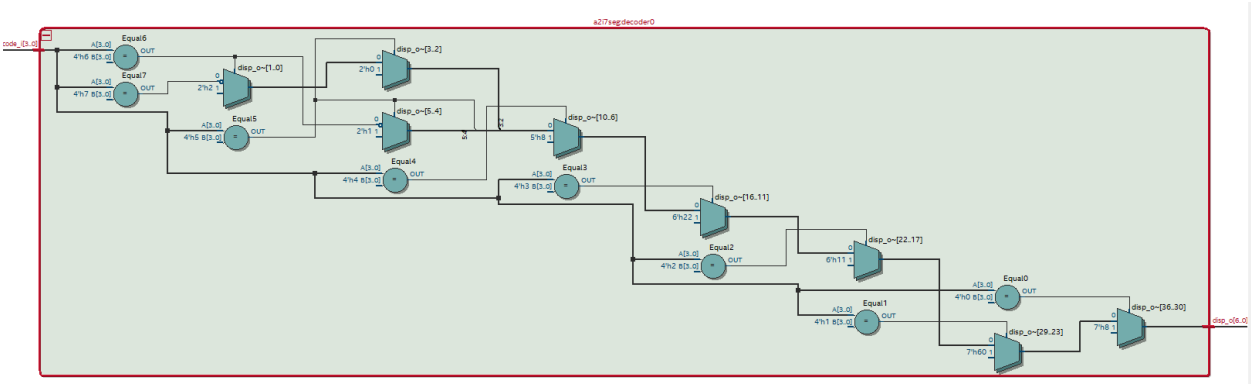


A2i7seg.v

seq4FSM_top.v | seq4FSM.v | debouncer.v | a2i7seg.v | Compilation Re

```
1   // CPEN230L lab 11, convert binary 1..9 to A..I on a 7-segment display
2   // Scott Tornquist, 12/3/2019
3
4   module a2i7seg (
5     input  [3:0] code_i,
6     output [6:0] disp_o );
7
8     assign disp_o = (code_i == 4'd0) ? 7'b0001000 : // A
9                     (code_i == 4'd1) ? 7'b0000011 : // b
10                    (code_i == 4'd2) ? 7'b1000110 : // C
11                    (code_i == 4'd3) ? 7'b0100001 : // d
12                    (code_i == 4'd4) ? 7'b0000110 : // E
13                    (code_i == 4'd5) ? 7'b0001110 : // F
14                    (code_i == 4'd6) ? 7'b0010000 : // G
15                    (code_i == 4'd7) ? 7'b0001001 : // H
16                    (code_i == 4'd8) ? 7'b1111001 : // I (1)
17                                       7'bxxxxxxx;
18  endmodule
```

RTL Viewer



- Discussion:
  - This lab was fairly straight forward and easy to code, we just had to edit 4 files, the important part of the lab was understanding how the FSM, test bench waveform, and Verilog code worked
  - The test bench and the top file just need basic connection to be made
  - The hardest part was figuring out the code for the always blocks
  - For the first block I used the state diagram to figure out which state went to which and followed the template given
  - For the second block I just reasoned out that we needed to reset if the reset signal was zero and otherwise progress to the next state

Summary and Conclusions

- This lab was not too tough but enjoyable
- It was quick, which I liked, but it definitely gave me a better understanding and more practice with finite state machines
- As well as more practice working with the programs and coding, which was good after a 2-week break