ELC 2137 Lab 11: FSM: Guessing Game

Maddie Vorhies

November 23, 2020

Summary

The goal of this lab is to use what we have learned in previous labs to create a guessing game. In order to do this, I first have to create a debounce circuit. A debounce ensures that the value of the input is held for a certain period of time. In this lab we were given the debounce module and the test, so I ran the test just to make sure that I recieved the correct output. In the game, there are two different settings: a fast setting and a slow setting. In order to implement this, I created two seperate counters. Both were very similar, but they have different paramters to distinguish between fast and slow. The larger the parameter the slower the counter so I set my parameter in the fast counter to N = 24 and the slow counter to N = 26. Then created a mux so that you can choose which setting you ant to play using switch 0. Next, I created a decoder so that the board would display on the upper four segments of the first 7-segment display digit. Lastly I created the guessFSM module which was the guessing game itself. This modules cycles through different states that correspond to different buttons being pressed. If the correct button is pressed at the correct time, you will go to the in state, and if any other button is pressed then you will go to the lose state. After this module was built, I created a testbench to test it. Then I created my overall guessing game module that put all of theses pieces together. I then created a guessing game testbench. Once it was successful I was able to play my game.

Q&A

(a) At what time in the simulation did the debounce circuit reach each of the four states (zero, wait1, one, wait0)?
The simulation starts out in state zero and then reaches wait1 at 200.0ns, then reaches one at about 245.121ns, then reaches wait0 at 600.0ns, and then reaches state zero again at about 646.327ns.

(b) Why can this game not be implemented with regular sequential logic?
We can't use regular sequential logic because this game does not have a regular/repeating pattern. In this game, I am using a finite state machine (FSM) which handles irregular, non-repeating conditions. The pattern is non-repeating because we don't know what buttons the user is going to guess and we don't know when they are going to win or lose. They could win twice in a row, then lose one, or lose three times in a row, etc. It is an irregular pattern which means we can't use regular sequential logic.

(c) What type of outputs did you use for your design (Mealy or Moore)? Explain
For my design, I used Moore because it's a much safer design than Mealy. Mealy is a machine

whose ouputs are determined by both its current state and the current inputs. Whereas a Moore machine whose output are only determined by its current state. In a Moore, the outputs only change when the state changes which means that it is synchronized with the clock. In a Mealy, the ouput can change from other inputs as well which means its no longer synchronized with the clock and could cause a glitch. I created a case statement that is only based on the specific states which means that its not based on any other input so it is a Moore machine.

Results

Figure 1: Simulation Waveform for Debounce

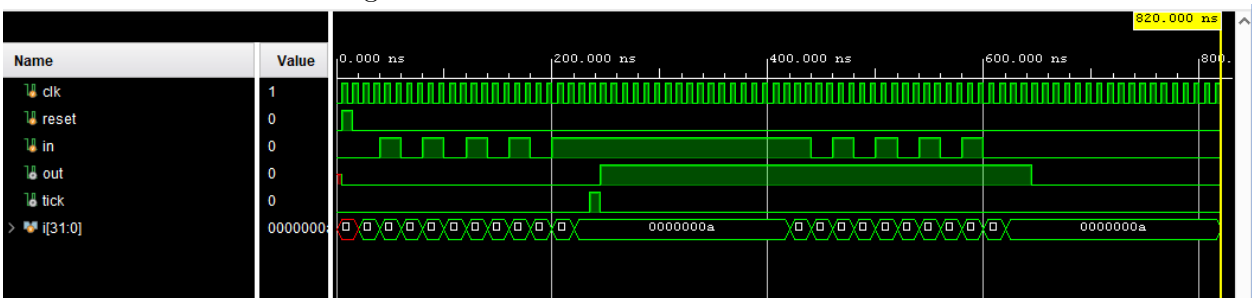

Figure 2: Simulation Waveform for Guess FSM

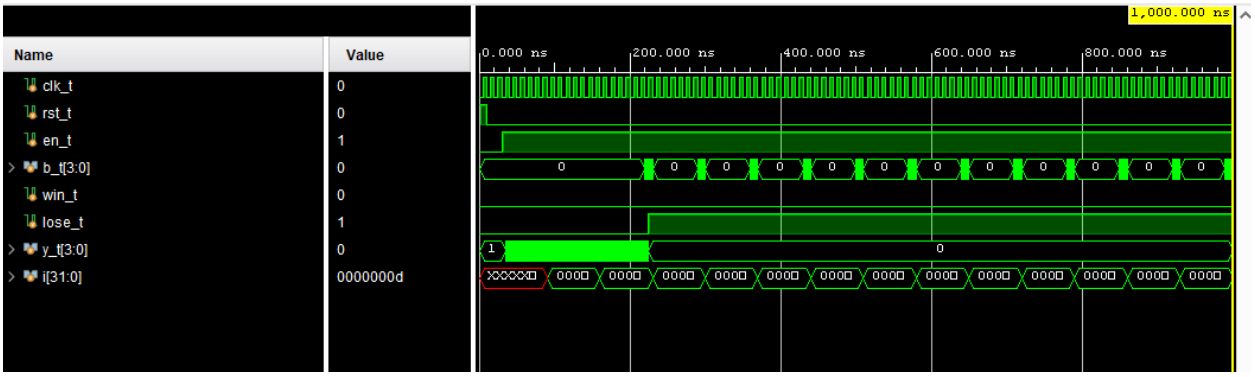

Figure 3: Simulation Waveform for Guessing Game

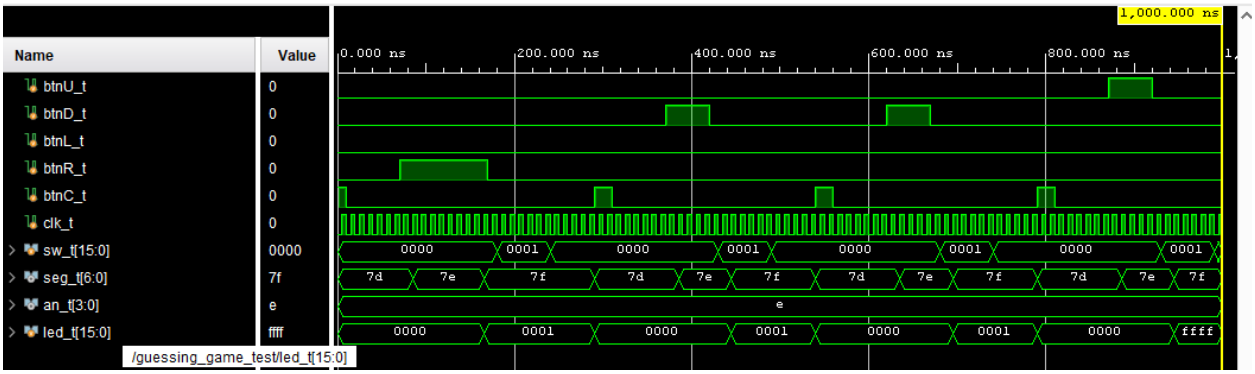

Table 1: 10 games on slow setting

Game:	1	2	3	4	5	6	7	8	9	10
win/lose	win	win	win	lose	win	win	win	win	win	win
segment	right	right	up	down	down	left	left	up	right	down
state	s0	s0	s1	s3	s3	s2	s2	s1	s0	s3
winning percentage	90									

Table 2: 10 games on fast setting

Game:	1	2	3	4	5	6	7	8	9	10
win/lose	win	win	win	win	win	lose	lose	win	win	win
segment	left	right	left	up	up	right	down	down	up	left
state	s2	s0	s2	s1	s1	s0	s3	s3	s1	s2
winning percentage	80									

Code

```
module guess_FSM(
    input clk, rst, en,
    input  [3:0] b,
    output reg [15:0] led,
    output reg win, lose,
    output reg [3:0] y
);

    localparam s0    = 3'b000;
    localparam s1    = 3'b001;
    localparam s2    = 3'b010;
    localparam s3    = 3'b011;
    localparam swin  = 3'b100;
    localparam slose = 3'b101;

    reg [2:0] state, state_next;

    always @(posedge clk, posedge rst)
        if (rst) begin
            state   <= s0;
    end
        else if (en)
          state   <= state_next;

    always @*

        case(state)
```

```
        s0: begin
            if (b == 4'b0000)
                state_next = s1;
            else if (b[3] == 1 || b[2] == 1 || b[1] == 1)
                state_next = slose;
            else
                state_next = swin;
        end

        s1: begin
            if (b == 4'b0000)
                state_next = s2;
            else if (b[3] == 1 || b[2] == 1 || b[0] == 1)
                state_next = slose;
            else
                state_next = swin;
        end

        s2: begin
            if (b == 4'b0000)
                state_next = s3;
            else if (b[3] == 1 || b[1] == 1 || b[0] == 1)
                state_next = slose;
            else
                state_next = swin;
        end

        s3: begin
            if (b == 4'b0000)
                state_next = s0;
            else if (b[2] == 1 || b[1] == 1 || b[0] == 1)
                state_next = slose;
            else
                state_next = swin;
        end

        swin:
            state_next = swin;

        slose:
            state_next = slose;

    endcase

always @* begin
    win = 1'b0;
    lose = 1'b0;
    led[15:0] = 16'b0;
```

```
        case(state)
           s0:
              begin
              y = 4'b0001;
                          win = 1'b0;
                          lose= 1'b0;
                          end
                       s1:
                          begin
                          y = 4'b0010;
                          win = 1'b0;
                          lose= 1'b0;
                          end
                       s2:
                          begin
                          y = 4'b0100;
                          win = 1'b0;
                          lose= 1'b0;
                          end
                       s3:
                          begin
                          y = 4'b1000;
                          win = 1'b0;
                          lose= 1'b0;
                          end

                       swin: begin
                          win = 1'b1;
                          lose = 1'b0;
                          y = 4'b1111;
                          led[15:0] = 16'b1111111111111111;
                       end

                       slose: begin
                          lose = 1'b1;
                          win = 1'b0;
                          y = 4'b0000;
                          led[0] = 1'b1;
                       end
                endcase
        end

endmodule


module guess_FSM_test();
        reg clk_t, rst_t, en_t;
        reg  [3:0] b_t;
```

```
reg win_t, lose_t;
reg [3:0] y_t;
integer i;

guess_FSM #(.N(21)) dut (
        .clk(clk_t),
        .rst(rst_t),
        .en(en_t),
        .b(b_t),
        .win(win_t),
        .lose(lose_t),
        .y(y_t)
);

always begin
        clk_t = ~clk_t; #5;
end

initial begin
        clk_t = 0; rst_t = 1; en_t = 0; b_t = 4'b0000; #10
        rst_t = 0; #20
        en_t = 1; #60

        for (i = 0; i <= 4'b1111; i = i + 1) begin
                b_t = 4'b0000;
                #60;
                b_t = i;
                #10;
        end

        b_t = 4'b0000;
        en_t = 0; #50;
        en_t = 1; #60;
        rst_t = 1; #10;
        rst_t = 0;

        for (i = 0; i <= 4'b1111; i = i + 1) begin
                b_t = 4'b0000;
                #50;
                b_t = i;
                #10;
        end

        b_t = 4'b0000;
        en_t = 0; #50;
        en_t = 1; #60;
        rst_t = 1; #10;
        rst_t = 0;
```

```
                for (i = 0; i <= 4'b1111; i = i + 1) begin
                        b_t = 4'b0000;
                        #70;
                        b_t = i;
                        #10;
                end
                $finish;
        end

endmodule


module guessing_game #(parameter N = 26, D = 21) (
        input btnU, btnD, btnL, btnR, btnC,
        input clk,
        input [15:0] sw,
        output [6:0] seg,
        output [3:0] an,
        output reg [15:0] led
);

        wire deb1_out, deb2_out, deb3_out, deb4_out;
        wire deb1_tick, deb2_tick, deb3_tick, deb4_tick;
        wire fast_count;
        wire slow_count;
        wire mux2_out;
        wire [3:0] guess_y;
        wire guess_win;
        wire guess_lose;

        debounce #(.N(D))deb1(
                .clk(clk),
                .reset(btnC),
                .in(btnR),
                .out(deb1_out),
                .tick(deb1_tick)
        );

        debounce #(.N(D))deb2(
                .clk(clk),
                .reset(btnC),
                .in(btnU),
                .out(deb2_out),
                .tick(deb2_tick)
        );

        debounce #(.N(D))deb3(
```

```
        .clk(clk),
        .reset(btnC),
        .in(btnL),
        .out(deb3_out),
        .tick(deb3_tick)
);

debounce #(.N(D))deb4(
        .clk(clk),
        .reset(btnC),
        .in(btnD),
        .out(deb4_out),
        .tick(deb4_tick)
);

counter #(.N(N)) slow(
        .clk(clk),
        .rst(btnC),
        .tick(slow_count)
);

counter #(.N(N-2)) fast(
        .clk(clk),
        .rst(btnC),
        .tick(fast_count)
);

mux2 #(.BITS(1)) mux_guess(
        .in0(slow_count),
        .in1(fast_count),
        .sel(sw[0]),
        .out(mux2_out)
);

guess_FSM guessing_game(
        .b({deb4_out, deb3_out, deb2_out, deb1_out}),
        .en(mux2_out),
        .clk(clk),
        .rst(btnC),
        .y(guess_y),
        .win(guess_win),
        .lose(guess_lose),
        .led(led)
);

y_decoder decoder(
        .in(guess_y),
        .out(seg)
```

```
        );

        assign an = 4'b1110;

endmodule


module guessing_game_test();
        reg btnU_t, btnD_t, btnL_t, btnR_t, btnC_t;
        reg clk_t;
        reg [15:0] sw_t;
        wire [6:0] seg_t;
        wire [3:0] an_t;
        reg [15:0] led_t;

        guessing_game #(.N(3), .D(1)) dut(
                .btnU(btnU_t),
                .btnD(btnD_t),
                .btnL(btnL_t),
                .btnR(btnR_t),
                .btnC(btnC_t),
                .clk(clk_t),
                .sw(sw_t),
                .seg(seg_t),
                .an(an_t),
                .led(led_t)
        );

        always begin
                clk_t = ~clk_t; #5;
        end

        initial begin
                clk_t = 0; btnC_t = 1; btnR_t = 0; btnD_t = 0;
                btnU_t = 0; btnL_t = 0; sw_t = 0; #10;
                btnC_t = 0; #20;
                sw_t = 0; #40;

                btnR_t = 1; btnL_t = 0;
                btnD_t = 0; btnU_t = 0; #100;

                btnR_t = 0; btnL_t = 0;
                btnD_t = 0; btnU_t = 0; #10;
                sw_t = 1; #60;
                sw_t = 0; #50;
                btnC_t = 1; #20;
                btnC_t = 0; #60;
```

```
                btnR_t = 0; btnL_t = 0;
                btnD_t = 1; btnU_t = 0; #50;

                btnR_t = 0; btnL_t = 0;
                btnD_t = 0; btnU_t = 0; #10;
                sw_t = 1; #60;
                sw_t = 0; #50;
                btnC_t = 1; #20;
                btnC_t = 0; #60;

                btnR_t = 0; btnL_t = 0;
                btnD_t = 1; btnU_t = 0; #50;

                btnR_t = 0; btnL_t = 0;
                btnD_t = 0; btnU_t = 0; #10;
                sw_t = 1; #60;
                sw_t = 0; #50;
                btnC_t = 1; #20;
                btnC_t = 0; #60;

                btnR_t = 0; btnL_t = 0;
                btnD_t = 0; btnU_t = 1; #50;

                btnR_t = 0; btnL_t = 0;
                btnD_t = 0; btnU_t = 0; #10;
                sw_t = 1; #60;
                sw_t = 0; #50;
                btnC_t = 1; #20;
                btnC_t = 0; #60;
                $finish;

        end

endmodule
```