ELC 2137 Lab 11: FSM Guessing Game

My Nguyen

December 1, 2020

Summary

This lab's purpose is to get a better understanding of the role of finite state machines (FSM), the difference between a Mealy output and a Moore output, how to implement state in Verilog. First part of the lab is to implement guess_FSM module to the given guessing game state diagram. Next, board module is implemented to convert guess_FSM module output to the seven-segment display output. Then, the counter module is modified to output only a tick instead of the two most significant numbers. Finally, guessing_game module is implemented using debounce module for the buttons, counter modules to adjust the difficulty of the game, guess_FSM as logic, and board module to output to the board. Afterward, create test file for guess_FSM, and guessing_game, then test the game using the Basys3 board.

Simulation Waveform

Code

Listing 1: debounce Test Bench

```
            #20 in=~in;
        end
        // hold input = 1 for a while
        in = 1; #200;
        // bounce
        for (i=0; i<10; i=i+1) begin
            #20 in=~in;
        end
        // hold input = 0 for a while
        in = 0; #200;
        $finish;
    end
endmodule // debounce_test
```

Listing 2: guess_FSM Implementation

```
module guess_FSM(
    input clk, rst, en,
    input [3:0]in, //buttons
    output reg [3:0] y,// led[3:0]
    output reg win, lose // indicator of win and lose
    );
    localparam [2:0]
        s0 = 3'b000,
```

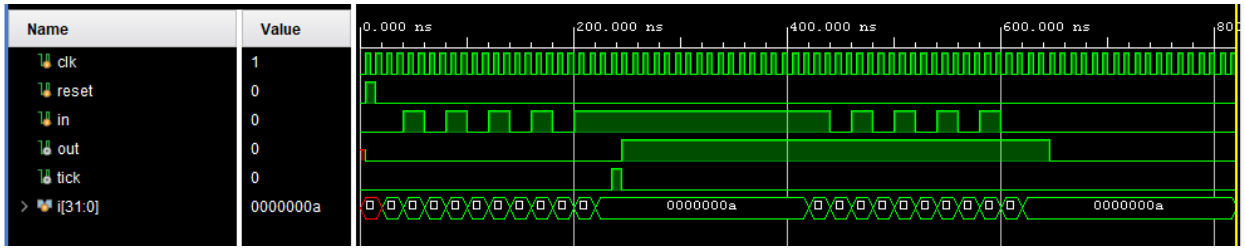

Figure 1: Debouncing ERT

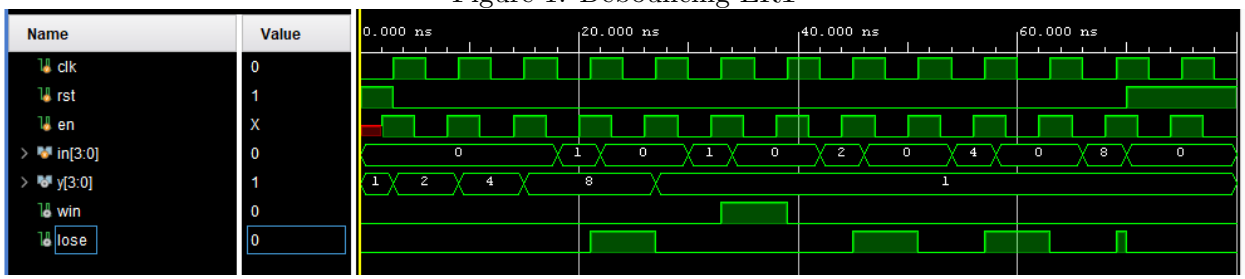

Figure 2: guess_FSM ERT

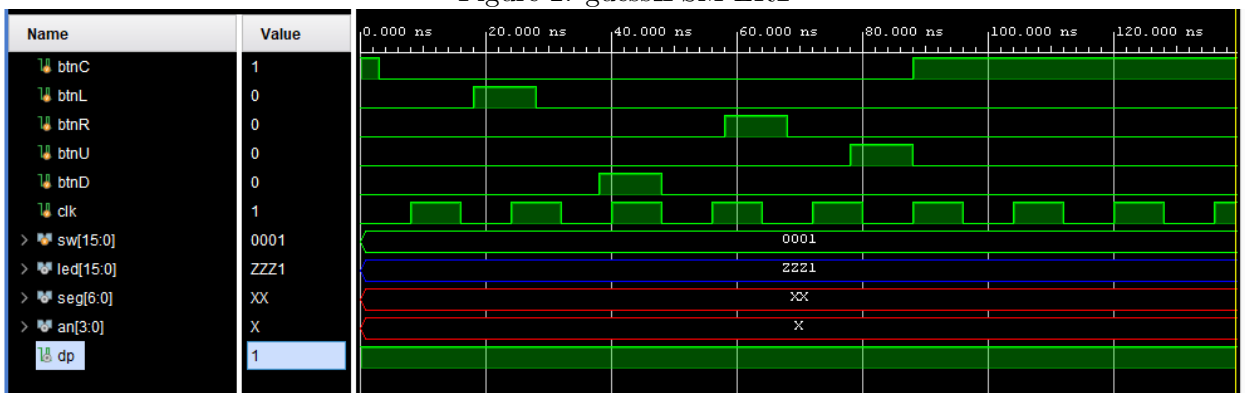

Figure 3: guessing_game ERT

```
        s1 = 3'b001,
        s2 = 3'b011,
        s3 = 3'b010,
        swin  = 3'b110,
        slose = 3'b111;

    // internal signals
    reg [2:0] state, state_next;

    // state memory (register)
    always @(posedge clk, posedge rst)
        if (rst)
            state <= s0;
        else if (en) // need an en to modulate the ticks
            state <= state_next;

    // combined next-state and output logic
    always @* begin
```

```
    // default behavior
    state_next = state;
    win = 1'b0;
    lose = 1'b0;

    case(state)
       //s0 state
       s0: begin
          y = 4'b0001;
          if (in == y)
              state_next = swin;
          else if (in == 4'b0000)
              state_next = s1;
          else
              state_next = slose;
       end
       //s1 state
       s1: begin
          y = 4'b0010;
          if (in == y)
              state_next = swin;
          else if (in == 4'b0000)
              state_next = s2;
          else
              state_next = slose;
       end
       //s2 state
       s2: begin
          y = 4'b0100;
          if (in == y)
              state_next = swin;
          else if (in == 4'b0000)
              state_next = s3;
          else
              state_next = slose;
       end
       //s3 state
       s3: begin
          y = 4'b1000;
          if (in == y)
              state_next = swin;
          else if (in == 4'b0000)
              state_next = s0;
          else
              state_next = slose;
       end
       //win state
       swin: begin
          win = 1'b1; // set win state to 1
          if (in == 4'b0000)
              state_next = s0;
          else
              state_next = swin;
       end
```

```
         //lose state
         slose: begin
            lose = 1'b1; //set lose state to 1
            if (in == 4'b0000)
                state_next = s0;
            else
                state_next = slose;
         end
      endcase
   end
endmodule
```

Listing 3: guess_FSM Test Bench

```
module test_guess_FSM();
    reg clk, rst, en;
    reg [3:0] in; //buttons
    wire [3:0] y;// led[3:0]
    wire win, lose;// led[15:14] // counter for now many time win/lose

    guess_FSM dut(
        .clk(clk),
        .rst(rst),
        .en(en),
        .in(in),
        .y(y),
        .win(win),
        .lose(lose)
        );

   //clk and en module
   always begin
        clk = ~clk ; #2;
        en = clk - 1; #1;
   end

   initial begin
        clk = 0; rst = 1; in = 4'b0000; #3;
        rst = 0; #15; //set up the initial condition
        in = 4'b0001; #4; // pressing the first button which loses
        in = 4'b0000; #8; // wait for the next guess
        in = 4'b0001; #4; // pressing the first button which wins
        in = 4'b0000; #8; // wait for the next guess
        in = 4'b0010; #4; // pressing the second button which loses
        in = 4'b0000; #8; // wait for the next guess
        in = 4'b0100; #4; // pressing the third button which loses
        in = 4'b0000; #8; // wait for the next guess
        in = 4'b1000; #4; // pressing the fourth button which loses
        in = 4'b0000; rst = 1; #10; //rest everything
        $finish;
   end
endmodule
```

Listing 4: guessing_game Implementation

```
module guessing_game(
    input btnC, btnL, btnR, btnU, btnD, clk,
    input [15:0] sw,
    output [15:0] led,
    output [6:0] seg,
    output [3:0] an,
    output dp
    );

    //debouncing the buttons
    wire [3:0] tmp;
    wire [3:0] debounced_buttons;
    debounce #(.N(21)) debounce0(
        .clk(clk),
        .reset(btnC),
        .in(btnL),
        .out(debounced_buttons[0]),
        .tick(tmp[0])
        );

    debounce #(.N(21)) debounce1(
        .clk(clk),
        .reset(btnC),
        .in(btnD),
        .out(debounced_buttons[1]),
        .tick(tmp[1])
        );

    debounce #(.N(21)) debounce2(
        .clk(clk),
        .reset(btnC),
        .in(btnR),
        .out(debounced_buttons[2]),
        .tick(tmp[2])
        );

    debounce #(.N(21)) debounce3(
        .clk(clk),
        .reset(btnC),
        .in(btnU),
        .out(debounced_buttons[3]),
        .tick(tmp[3])
        );

    //creating two counter of different size for mux2 element
    wire counter_in0;
    count #(.N(25)) c0(
        .clk(clk),
        .en(1'b1),
        .rst(btnC),
        .tick(counter_in0)
        );
```

```
    wire counter_in1;
    count #(.N(23)) c1(
        .clk(clk),
        .en(1'b1),
        .rst(btnC),
        .tick(counter_in1)
        );

    //mux2 element sending out signal for guess_FSM
    wire mux_guess;
    mux2 #(.BITS(1)) mux0(
        .sel(sw[0]), //use to adjust difficulty
        .in1(counter_in1),
        .in0(counter_in0),
        .out(mux_guess)
        );

    wire win, lose;
    guess_FSM main(
        .clk(clk),
        .rst(btnC),
        .en(mux_guess),
        .in(debounced_buttons),
        .y(led[3:0]),
        .win(win),
        .lose(lose)
        );

    board b(
        .win(win),
        .lose(lose),
        .clk(clk),
        .en(mux_guess),
        .rst(btnC),
        .seg(seg),
        .an(an),
        .dp(dp)
        );
endmodule
```

Listing 5: guessing_game Test Bench

```
module test_guessing_game();
    reg btnC, btnL, btnR, btnU, btnD, clk;
    reg [15:0] sw;
    wire [15:0] led;
    wire [6:0] seg;
    wire [3:0] an;
    wire dp;

    guessing_game dut(
        .btnC(btnC),
        .btnL(btnL),
        .btnR(btnR),
```

```
        .btnU(btnU),
        .btnD(btnD),
        .clk(clk),
        .sw(sw),
        .led(led),
        .seg(seg),
        .an(an),
        .dp(dp)
        );

    //clk and en module
    always begin
        clk = ~clk ; #8;
    end
    initial begin
        clk = 0; sw = 16'h0001; btnC = 1'b1; btnL = 1'b0; btnD = 1'b0;
           btnR = 1'b0; btnU = 1'b0;#3;
        btnC = 0; #15; //set up the initial condition
        btnL = 1'b1; btnD = 1'b0; btnR = 1'b0; btnU = 1'b0; #10;
        btnL = 1'b0; btnD = 1'b0; btnR = 1'b0; btnU = 1'b0; #10;
        btnL = 1'b0; btnD = 1'b1; btnR = 1'b0; btnU = 1'b0; #10;
        btnL = 1'b0; btnD = 1'b0; btnR = 1'b0; btnU = 1'b0; #10;
        btnL = 1'b0; btnD = 1'b0; btnR = 1'b1; btnU = 1'b0; #10;
        btnL = 1'b0; btnD = 1'b0; btnR = 1'b0; btnU = 1'b0; #10;
        btnL = 1'b0; btnD = 1'b0; btnR = 1'b0; btnU = 1'b1; #10;
        btnC = 1'b1; btnL = 1'b0; btnD = 1'b0; btnR = 1'b0; btnU = 1'b0
           ;#10;
    end
endmodule
```

Data List

Table 1: Lists of Games Played

Game	N = 25	N = 23
1	win	win
2	lose	win
3	win	lose
4	lose	win
5	lose	lose
6	lose	lose
7	win	win
8	lose	win
9	lose	win
10	lose	lose
Winrate	30%	60%

Q&A

1. In the simulation the debounce circuit reach each of the four states around 621ns

2. This game can't be implemented using sequential logic because the game state must run independently from the the buttons and the output, if not, the game will be very glitchy.

3. I used both Mealy and Moore output for my design. I used Moore output to control the led indicating which state the program was in, because there is no further condition needed for it to be true. For everything else like determining the next state, and win/lose condition, I used Mealy output. I did this because there are multiple variable that needed to consider when determining next state for guess_FSM, because a button could reset the state, get the correct guess or the wrong guess. Meanwhile, for my "board" module, Moore output is also used to determine next state since it need win/lose flag to do so.