ELC 2137 Lab 11: FSM: Guessing Game

Aaron Mendoza

November 19, 2020

Summary

The purpose of this lab is to build a guessing game in which the user presses buttons on a Basys3 board that correspond to certain states in order to win.

The first module built during this lab was the debounce module. Although pressing a button is seemingly straightforward, there is actually something called a "bounce" where the output cycles on and off a few times when the button is pressed or a switch is moved. In order to fix this, I built a debounce module, where it changes between four states: zero, wait1, one, wait0. In order to switch from zero to one, the output from zero must remain on for a long enough time during wait1, in which case it will switch to one. If the output from zero does not remain on long enough during wait1, then it returns back to zero. This eliminates the bounce behavior from pressing the pushbuttons on the Basys3 board. This module has three inputs (in, reset, clock) and two outputs (out and tick). Tick corresponds with if the button was pressed, and out corresponds with how long the button was pressed.

The second module built during this lab was the guess-FSM module. This module contains the main states of the game. The guess-FSM module has six states: s0, s1, s2, s3, swin, and slose. Whenever there is no input given, the FSM cycles through s0 to s3. When there is an input given at one of those states, it will either go to swin or slose based on logic. My guess-FSM module has three inputs (clock, enable, and b) and three outputs (y, win, and lose).

The third module built was just a counter module built from a previous lab. This controls the time it takes to switch between the states of my guess-FSM module. In other words, this controls the difficulty of the game. This is made possible by using a mux, where turning a switch on and off changes which counter is used. The only difference between the counters is the parameter N (number of bits) given within the code.

My top level module guessing-game combines all of these previous modules. Each of the buttons on the Basys3 board(btnU, btnL, btnR, and btnD) feed into the input of a debounce module whose outputs go into the "b" input of the my guess-FSM module. The output of the mux whose inputs are from the counters are decided by sw[0] of the board, which goes into the enable input of my guess-FSM to keep the states switching from s0 to s3 at a constant rate. Finally, I made my y output of the guess-FSM module correspond with the states. In order to do this, I made a case statement where the y switches between the top four segments of the right most anode. In other words, the moving target is a dash that moves in a box on the display. This makes it easier for the user to pick the button that matches with the moving target.

Q&A

1. At what time in the simulation the did debounce circuit reach each of the four states (*zero, wait1, one, wait0*)?

   The debounce circuit reached zero at 20 ns, wait1 at 200 ns, one at 245 ns, and wait0 at 600 ns.

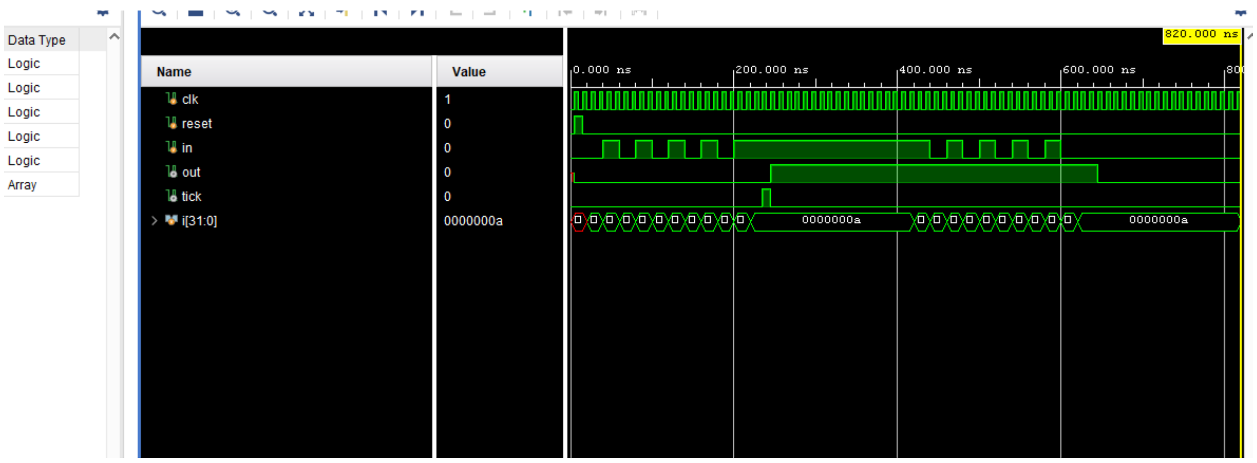
2. Why can this game not be implemented with regular sequential logic?

   This game cannot be built using regular sequential logic because the outputs do not repeat or exhibit a specific pattern. So, in this lab I implemented a finite state machine. An FSM has a limited number of internal ”states” that switch between each other based on the inputs. This is crucial for the game; the states are not just repeating from s0 to s1 to s2 to s3 all the time. Instead, s0 can go to swin, slose, or s1 based on the input, so an FSM is needed in order for this game to work.

3. What types of outputs did you use for your design (Mealy or Moore)? Explain.

   Moore because it is safer than Mealy. This is the safer design because there is less risk for glitches. The output is only dependent on the current state, and if the inputs change the output changes as well. Although Mealy is faster, it is harder to design and puts me more at risk for glitches, so I used Moore.

Results



Data Type	Name	Value
Logic	clk	1
Logic	reset	0
Logic	in	0
Logic	out	0
Logic	tick	0
Array	i[31:0]	0000000a

Figure 1: Debounce Simulation Screenshot

Table 1: Games Played

Game:	1	2	3	4	5	6	7	8	9	10
Easy	W	W	W	W	W	W	W	W	W	W
Hard	W	W	W	W	W	W	W	W	W	W

The win percentage of both my difficulties was 100 percent. The speed of my moving target was not very fast, but that can easily change by modifying the parameters within my code.

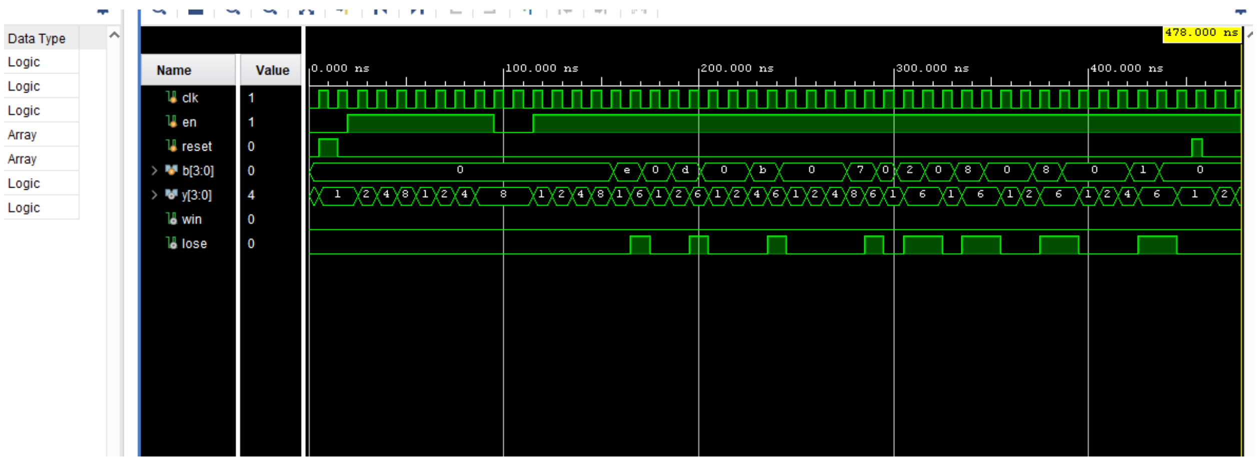


Figure 2: Guess-FSM Simulation Screenshot

Code

Listing 1: Debounce Module Code

```
`timescale 1ns / 1ps
// ELC 2137, John Miller, 2019-11-08

module debounce #(parameter N=21)
   (input clk, reset,
    input  in,
    output reg out,
    output reg tick);

   // define states as local parameters (constants)
   localparam [1:0]
      zero    = 2'b00,
      wait1   = 2'b01,
      one     = 2'b11,
      wait0   = 2'b10;

   // internal signals
   reg [1:0] state, state_next;
   reg [N-1:0] counter, counter_next;

   // state memory (register)
   always_ff @(posedge clk or posedge reset)
      if (reset) begin
         state   <= zero;
         counter <= {N{1'b1}};
      end
      else begin
         state   <= state_next;
         counter <= counter_next;
      end

   // combined next-state and output logic
   always_comb begin
```

```
      // default behavior
      state_next   = state;
      counter_next = counter;
      tick = 0;

      case(state)
         zero: begin
            out = 0;
            counter_next = {N{1'b1}};
            if (in)
               state_next = wait1;
         end

         wait1: begin
            out = 0;      // Moore output
            counter_next = counter - 1;
            if (counter == 0) begin
               tick = 1'b1; // Mealy output
               state_next = one;
            end
            else if (~in)
               state_next = zero;
         end

         one: begin
            out = 1;
            counter_next = {N{1'b1}};
            if (~in)
               state_next   = wait0;
         end

         wait0: begin
            out = 1;
            counter_next = counter - 1;
            if (counter == 0)
               state_next = zero;
            else if (in)
               state_next = one;
         end
      endcase
   end

endmodule // debounce
```

—

Listing 2: Debounce Testbench Code

```
`timescale 1ns / 1ps
// ELC 2137, John Miller, 2019-11-08

module debounce_test();

    reg clk, reset, in;
    wire out, tick;
    integer i;

    debounce #(.N(2)) db (.clk(clk), .reset(reset), .in(in), .out(out),
        .tick(tick));

    always begin
        #5 clk = ~clk;
    end

    initial begin
        clk=0; reset=0; in=0; #5;
        reset=1; #10;
        reset=0; #5;
        // bounce
        for (i=0; i<10; i=i+1) begin
            #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 3: Counter Module Code

```
module counter #(parameter N=20)(
    input clk,
    input rst,
    output out
    );
    reg [N-1:0] Qreg, Qnext;
    always @(posedge clk, posedge rst) begin
        if (rst)
        Qreg <= 0;
        else
        Qreg <= Qnext;
    end
    always @* begin
        Qnext = Qreg+1;
    end
    assign out = &Qreg;

endmodule
```

—

Listing 4: MUX Difficulty Module Code

```
module mux_diff(
    input in0, in1, sel,
    output reg out
    );
    assign out = sel ? in1 : in0;
endmodule
```

—

Listing 5: Guess FSM Module Code

```
module guess_FSM(
    input clk, en, reset,
    input [3:0] b,
    output reg [3:0] y,
    output reg win,
    output reg lose
    );
    localparam[2:0]
        s0 = 3'b001,
        s1 = 3'b010,
        s2 = 3'b011,
        s3 = 3'b100,
        swin = 3'b101,
        slose = 3'b110;

    reg [2:0] state, state_next;

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

    always_comb begin
        y[3:0]  = 0;
        win = 0;
        lose=0;
        state_next = s0;
        case(state)
            s0: begin
            y=4'b0001;
                if (b[1] | b[2] | b[3]) begin
                    state_next = slose;
                    end
                else if (b[0] & ~b[1] & ~b[2] & ~b[3]) begin
                    state_next = swin;
                    end
                else
                    state_next = s1;
                end
            s1: begin
            y=4'b0010;
                if (b[0] | b[2] | b[3]) begin
                    state_next = slose;
                    end
                else if (~b[0] & b[1] & ~b[2] & ~b[3]) begin
                    state_next = swin;
                    end
                else
                    state_next = s2;
                end
            s2: begin
            y=4'b0100;
                if (b[0] | b[1] | b[3]) begin
                    state_next = slose;
                    end
                else if (~b[0] & ~b[1] & b[2] & ~b[3]) begin
                    state_next = swin;
                    end
                else
                    state_next = s3;
                end
            s3: begin
            y=4'b1000;
                if (b[0] | b[1] | b[2]) begin
                    state_next = slose;
                    end
                else if (~b[0] & ~b[1] & ~b[2] & b[3]) begin
                    state_next = swin;
                    end
                else
                    state_next = s0;
                end
            slose: begin
            lose = 1;
            win = 0;
            y = 4'b0000;
                if (b[0] | b[1] | b[2] | b[3]) begin
                    state_next = slose;
                    end
                else
                    state_next = s0;
                end
            swin: begin
            win= 1;
            lose = 0;
            y=4'b1111;
                if (b[0] | b[1] | b[2] | b[3]) begin
                    state_next = swin;
                    end
                else
                    state_next = s0;
                end
          endcase
```

```
     end
endmodule
```

—

Listing 6: Guess FSM Testbench Code

```
module guess_FSM_test();
    reg clk, en, reset;
    reg [3:0] b;
    wire [3:0] y;
    wire win;
    wire lose;
    guess_FSM dut(
        .clk(clk),
        .en(en),
        .reset(reset),
        .b(b),
        .y(y),
        .win(win),
        .lose(lose));
    always begin
        #5 clk = ~clk;
    end

    initial begin
        clk=0; reset=0; en=0; ; b=4'b0000; #5;
        reset=1; #10;
        reset=0; #5;
        en=1; #75; //two cycles through the states
        en=0; #20; //testing hold
        en=1; #41; //goes into first state
        b=4'b1110; #15; //win state for first state
        b=4'b0000; #15; //let go of button, goes back to first state and
           enters second state
        b=4'b1101; #15; //win state for second state
        b=4'b0000; #25;
        b=4'b1011; #15; //win state for third state
        b=4'b0000; #35;
        b=4'b0111; #15; //win state for fourth state
        b=4'b0000; #10;
        b=4'b0010; #15; //lose state for first state
        b=4'b0000; #15;
        b=4'b1000; #15; //lose state for second state
        b=4'b0000; #25;
        b=4'b1000; #15; //lose state for third state
        b=4'b0000; #35;
        b=4'b0001; #15; //lose state for fourth state
        b=4'b0000; #17;
        reset = 1; #5;
        reset = 0; #20; //testing reset
        $finish;
    end
endmodule
```

—

Listing 7: Guessing Game Module Code

```
module guessing_game #(parameter B = 27, R = 21)(
    input btnU, btnD, btnR, btnL, clk, btnC,
    input[15:0] sw,
    output reg [6:0] seg,
    output [3:0] an,
    output reg [15:0] led,
    output dp
    );

    wire[3:0] db_out, dc;
    wire easy_out, hard_out, mux_out;
    reg win_out, lose_out;
    reg[3:0] y_out;

    debounce #(.N(R)) db1 (
        .clk(clk),
        .reset(btnC),
        .in(btnR),
        .out(dc[0]),
        .tick(db_out[0]));
    debounce #(.N(R)) db2 (
        .clk(clk),
        .reset(btnC),
        .in(btnD),
        .out(dc[1]),
        .tick(db_out[1]));
    debounce #(.N(R)) db3 (
        .clk(clk),
        .reset(btnC),
        .in(btnL),
        .out(dc[2]),
        .tick(db_out[2]));
    debounce #(.N(R)) db4 (
        .clk(clk),
        .reset(btnC),
        .in(btnU),
        .out(dc[3]),
        .tick(db_out[3]));

        counter #(.N(B)) easy(
            .clk(clk),
            .rst(btnC),
            .out(easy_out));

        counter #(.N(B-1)) hard(
            .clk(clk),
            .rst(btnC),
            .out(hard_out));

        mux_diff mymux(
            .in0(easy_out),
            .in1(hard_out),
            .sel(sw[0]),
            .out(mux_out));

    guess_FSM myguess_FSM(
        .b(dc),
        .clk(clk),
        .en(mux_out),
        .reset(btnC),
        .y(y_out),
        .win(win_out),
        .lose(lose_out));

    always @* begin
        case(y_out)
            4'b0001: seg = 7'b1111101;
            4'b0010: seg = 7'b0111111;
            4'b0100: seg = 7'b1011111;
            4'b1000: seg = 7'b1111110;
            default: seg = 7'b1111111;
        endcase
    end

    always @* begin
    if (win_out)
        led = 16'b1111111111111111;
    else if (lose_out)
        led = 16'b1010101010101010;
    else
        led = 16'b0000000000000000;
    end

    assign an = 4'b1110;
    assign dp=1'b1;

endmodule
```

—