(ELEN 127 Lab 5: A Bigger State Machine)

(Lab Section: Tuesday 2:15)

(Curtis Robinson, Liam Kelly)

Introduction

In this lab our goal is to create a controller for a game called the farmer game. We are tasked with moving the farmer the fox the chicken and the seed over to the right side of the game without encountering a state where the fox and chicken and the seeds and chicken are on thje same side alone without the farmer. This caused the played to lose the game according to the rules. The rules of the game are that the farmer can only move solo or with 1 other entity at a time and the initial mention that the above entities cannot be left alone. Overall, we were able to implement the state machine and then test it with the testbench shown below.

Procedure

We began with the development of the state machine. We listed out all of the cases for moving each of the entities across to the other side and the moving them back. After this we labeled our win case and our lose cases and the paths to each of them which then would help us with the design of the testbench. Then, we implemented the state machine using a bunch of case statements and choosing the inputs and output signals. After the implementation of that was complete we built our testbench with the notation that we made during the state machine development from earlier. This made the process of the testbench development fast and easy to understand. Finally we tested and corrected any of the errors that we found and then demoed to the TA.

State Machine Implementation

Code

```
module mealy_fsm(
  input wire [2:0] move_item,
  input wire clk,       // Clock input
  input wire reset,     // Reset input
  output reg i,
  output wire win,
  output wire lose
);

  reg [3:0] state, next_state;          // State register

  // Outputs
  assign win = (state == 4'hF);
  assign lose = (state == 4'hC);

  always @(posedge clk or posedge reset) begin
     if (reset) begin
                          state <= 1'b0;
     end else state <= next_state;
  end

  always @(*) begin
     case(state)
       4'h0: begin
         if (move_item <= 3'h2) begin
           next_state <= 4'h0;
           i <= 1'b1;
         end
                          else i <= 1'b0;
                          if (move_item == 3'h3) next_state <= 4'h0;
                          else if (move_item == 3'h4) next_state <= 4'h8;
         else if (move_item == 3'h5) next_state <= 4'hC;
         else if (move_item == 3'h6) next_state <= 4'hA;
         else if (move_item == 3'h7) next_state <= 4'hC;
       end
       4'h1: begin
         if (move_item <= 3'h2 || move_item == 3'h7) begin
           next_state <= 4'h1;
           i <= 1'b1;
         end
                          else i <= 1'b0;
                          if (move_item == 3'h3) next_state <= 4'h1;
         else if (move_item == 3'h4) next_state <= 4'hC;
         else if (move_item == 3'h5) next_state <= 4'hD;
         else if (move_item == 3'h6) next_state <= 4'hB;
       end
       4'h2: begin
         if (move_item <= 3'h2 || move_item == 3'h6) begin
           next_state <= 4'h2;
           i <= 1'b1;
         end
                          else i <= 1'b0;
                           if (move_item == 3'h3) next_state <= 4'h2;
         else if (move_item == 3'h4) next_state <= 4'hA;
         else if (move_item == 3'h5) next_state <= 4'hE;
         else if (move_item == 3'h7) next_state <= 4'hB;
       end
```

```
4'h4: begin
  if (move_item <= 3'h2 || move_item == 3'h5) begin
    next_state <= 4'h4;
    i <= 1'b1;
  end
                        else i <= 1'b0;
                        if (move_item == 3'h3) next_state <= 4'h4;
  else if (move_item == 3'h4) next_state <= 4'hC;
  else if (move_item == 3'h6) next_state <= 4'hE;
  else if (move_item == 3'h7) next_state <= 4'hB;
end
4'h5: begin
  if (move_item <= 3'h2 || move_item == 3'h5 || move_item == 3'h7) begin
    next_state <= 4'h5;
    i <= 1'b1;
  end
                        else i <= 1'b0;
                        if (move_item == 3'h3) next_state <= 4'h5;
  else if (move_item == 3'h4) next_state <= 4'hD;
  else if (move_item == 3'h6) next_state <= 4'hF;
end
4'h8: begin
  if (move_item <= 3'h2 || move_item >= 3'h5) begin
    next_state <= 4'h8;
    i <= 1'b1;
  end
                        else i <= 1'b0;
                        if (move_item == 3'h3) next_state <= 4'h8;
  else if (move_item == 3'h4) next_state <= 4'h0;
end
4'hA: begin
  if (move_item <= 3'h2 || move_item == 3'h5 || move_item == 3'h7) begin
    next_state <= 4'hA;
    i <= 1'b1;
  end
                        else i <= 1'b0;
                         if (move_item == 3'h3) next_state <= 4'hA;
  else if (move_item == 3'h4) next_state <= 4'h2;
  else if (move_item == 3'h6) next_state <= 4'h0;
end
4'hB: begin
  if (move_item <= 3'h2 || move_item == 3'h5) begin
    next_state <= 4'hB;
    i <= 1'b1;
  end
                        else i <= 1'b0;
                        if (move_item == 3'h3) next_state <= 4'hB;
  else if (move_item == 3'h4) next_state <= 4'hC;
  else if (move_item == 3'h6) next_state <= 4'h1;
  else if (move_item == 3'h7) next_state <= 4'h2;
end
4'hD: begin
  if (move_item <= 3'h2 || move_item == 3'h6) begin
    next_state <= 4'hD;
    i <= 1'b1;
  end
                        else i <= 1'b0;
                        if (move_item == 3'h3) next_state <= 4'hD;
  else if (move_item == 3'h4) next_state <= 4'h5;
  else if (move_item == 3'h5) next_state <= 4'h1;
  else if (move_item == 3'h7) next_state <= 4'h4;
end
```

```
      4'hE: begin
        if (move_item <= 3'h2 || move_item == 3'h7) begin
          next_state <= 4'hE;
          i <= 1'b1;
        end
                          else i <= 1'b0;
                          if (move_item == 3'h3) next_state <= 4'hE;
        else if (move_item == 3'h4) next_state <= 4'hC;
        else if (move_item == 3'h5) next_state <= 4'h2;
        else if (move_item == 3'h6) next_state <= 4'h4;
      end
      4'hF: begin
        next_state <= 4'hF;
        if (move_item != 3'h3) i <= 1'b1;
                          else i <= 1'b0;
      end
      4'hC: begin
        next_state <= 4'hC;
        if (move_item != 3'h3) i <= 1'b1;
                          else i <= 1'b0;
      end
    endcase
  end

endmodule
```

Description: Here is our state machine implementation showing our intermediate states a well as our inital and win/lose states.

Testbench Implementation

Code

```
module testbench();
  reg clk;
  reg rst;
  reg [2:0] move_item;
  wire i;
  wire win;
  wire lose;

  mealy_fsm dut (move_item, clk, rst, i ,win ,lose);

  // Clock generation
  always begin
    #5 clk = ~clk;
    $display("Rst: %d State: %h Move: %h Next State: %h Impossible: %b Win: %b Lose: %b %d", rst, dut.state, move_item,
dut.next_state, i, win, lose, $time);
    #5 clk = ~clk;
  end

  // Initialize inputs
  initial begin
    clk = 0;
    rst = 1;
    move_item = 3'b011;

    // Test Case 1: Initial state with no move
    $display("Test Case 1: Initial state with no move");
```

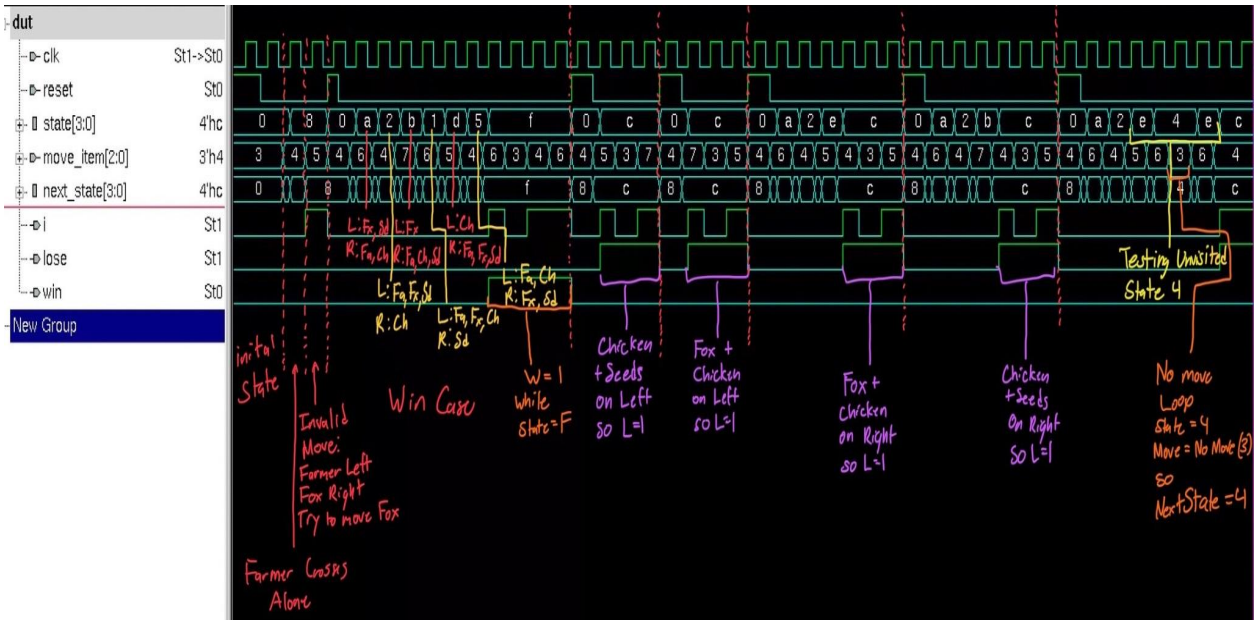
```
#12;
rst = 0;
#10;

// Test Case 2: Valid move - Farmer crosses alone
$display("\nTest Case 2: Valid move - Farmer crosses alone");
move_item = 3'b100;
#10;

// Test Case 3: Invalid move - Farmer and Fox on opposite sides
$display("\nTest Case 3: Invalid move - Farmer + Fox on opposite sides");
move_item = 3'b101;
#10;

// Test Case 4: Win the game
$display("\nTest Case 4: Win the game");
move_item = 3'b100;
        rst = 1;
#5;
        rst = 0;
        #5
move_item = 3'h6;
        #10;
move_item = 3'h4;
        #10;
move_item = 3'h7;
        #10;
move_item = 3'h6;
        #10;
move_item = 3'h5;
        #10;
move_item = 3'h4;
        #10;
move_item = 3'h6;
        #10;
move_item = 3'h3;
        #10;
move_item = 3'h4;
        #10;
move_item = 3'h6;
        #10;

// Test Case 5: Lose: Chicken + Seeds on Left
$display("\nTest Case 5: Lose Chicken Seeds on Left");
move_item = 3'h4; rst=1'b1;
#10;
        rst=0;
move_item = 3'h5;
#10;
move_item = 3'h3;
#10;
move_item = 3'h7;
#10;

// Test Case 6: Lose: Chicken + Fox on Left
$display("\nTest Case 6: Lose Chicken Fox on Left");
move_item = 3'h4; rst=1'b1;
#10;
        rst=0;
move_item = 3'h7;
#10;
```

```
move_item = 3'h3;
#10;
move_item = 3'h5;
#10;

// Test Case 7: Lose: Chicken + Fox on Right
$display("\nTest Case 7: Lose Chicken Seeds on Right");
move_item = 3'h4; rst=1'b1;
#10;
        rst=0;
move_item = 3'h6;
#10;
move_item = 3'h4;
#10;
move_item = 3'h5;
#10;
move_item = 3'h4;
#10;
move_item = 3'h3;
#10;
move_item = 3'h5;
#10;

// Test Case 8: Lose: Chicken + Seeds on Right
$display("\nTest Case 8: Lose Chicken Fox on Right");
move_item = 3'h4; rst=1'b1;
#10;
        rst=0;
move_item = 3'h6;
#10;
move_item = 3'h4;
#10;
move_item = 3'h7;
#10;
move_item = 3'h4;
#10;
move_item = 3'h3;
#10;
move_item = 3'h5;
#10;

// Test Case 9: Testing last unvistited & no move loop
$display("\nTest Case 9: Testing last unvistited & no move loop");
move_item = 3'h4; rst=1'b1;
#10;
        rst=0;
move_item = 3'h6;
#10;
move_item = 3'h4;
#10;
move_item = 3'h5;
#10;
move_item = 3'h6;
#10;
move_item = 3'h3;
#10;
move_item = 3'h6;
#10;
        move_item = 3'h4;
#20;

// Finish simulation
```

```
    $finish;
  end

endmodule
```

Description: Here is our testbench that demonstrates our test cases with the comments.

Waveform



Description: Here is our waveform that shows our transitions between states and the movement to the lose and win cases.

Output

```
Test Case 1: Initial state with no move
Rst: 1 State: 0 Move: 3 Next State: 0 Impossible: 0 Win: 0 Lose: 0                  5
Rst: 0 State: 0 Move: 3 Next State: 0 Impossible: 0 Win: 0 Lose: 0                 15

Test Case 2: Valid move - Farmer crosses alone
Rst: 0 State: 0 Move: 4 Next State: 8 Impossible: 0 Win: 0 Lose: 0                 25

Test Case 3: Invalid move - Farmer + Fox on opposite sides
Rst: 0 State: 8 Move: 5 Next State: 8 Impossible: 1 Win: 0 Lose: 0                 35

Test Case 4: Win the game
Rst: 1 State: 0 Move: 4 Next State: 8 Impossible: 0 Win: 0 Lose: 0                 45
Rst: 0 State: 0 Move: 6 Next State: a Impossible: 0 Win: 0 Lose: 0                 55
Rst: 0 State: a Move: 4 Next State: 2 Impossible: 0 Win: 0 Lose: 0                 65
Rst: 0 State: 2 Move: 7 Next State: b Impossible: 0 Win: 0 Lose: 0                 75
Rst: 0 State: b Move: 6 Next State: 1 Impossible: 0 Win: 0 Lose: 0                 85
Rst: 0 State: 1 Move: 5 Next State: d Impossible: 0 Win: 0 Lose: 0                 95
Rst: 0 State: d Move: 4 Next State: 5 Impossible: 0 Win: 0 Lose: 0                105
Rst: 0 State: 5 Move: 6 Next State: f Impossible: 0 Win: 0 Lose: 0                115
Rst: 0 State: f Move: 3 Next State: f Impossible: 0 Win: 1 Lose: 0                125
Rst: 0 State: f Move: 4 Next State: f Impossible: 1 Win: 1 Lose: 0                135
Rst: 0 State: f Move: 6 Next State: f Impossible: 1 Win: 1 Lose: 0                145

Test Case 5: Lose Chicken Seeds on Left
Rst: 1 State: 0 Move: 4 Next State: 8 Impossible: 0 Win: 0 Lose: 0                155
Rst: 0 State: 0 Move: 5 Next State: c Impossible: 0 Win: 0 Lose: 0                165
Rst: 0 State: c Move: 3 Next State: c Impossible: 0 Win: 0 Lose: 1                175
Rst: 0 State: c Move: 7 Next State: c Impossible: 1 Win: 0 Lose: 1                185

Test Case 6: Lose Chicken Fox on Left
Rst: 1 State: 0 Move: 4 Next State: 8 Impossible: 0 Win: 0 Lose: 0                195
Rst: 0 State: 0 Move: 7 Next State: c Impossible: 0 Win: 0 Lose: 0                205
Rst: 0 State: c Move: 3 Next State: c Impossible: 0 Win: 0 Lose: 1                215
Rst: 0 State: c Move: 5 Next State: c Impossible: 1 Win: 0 Lose: 1                225

Test Case 7: Lose Chicken Seeds on Right
```

```
Test Case 7: Lose Chicken Seeds on Right
Rst: 1 State: 0 Move: 4 Next State: 8 Impossible: 0 Win: 0 Lose: 0                235
Rst: 0 State: 0 Move: 6 Next State: a Impossible: 0 Win: 0 Lose: 0                245
Rst: 0 State: a Move: 4 Next State: 2 Impossible: 0 Win: 0 Lose: 0                255
Rst: 0 State: 2 Move: 5 Next State: e Impossible: 0 Win: 0 Lose: 0                265
Rst: 0 State: e Move: 4 Next State: c Impossible: 0 Win: 0 Lose: 0                275
Rst: 0 State: c Move: 3 Next State: c Impossible: 0 Win: 0 Lose: 1                285
Rst: 0 State: c Move: 5 Next State: c Impossible: 1 Win: 0 Lose: 1                295

Test Case 8: Lose Chicken Fox on Right
Rst: 1 State: 0 Move: 4 Next State: 8 Impossible: 0 Win: 0 Lose: 0                305
Rst: 0 State: 0 Move: 6 Next State: a Impossible: 0 Win: 0 Lose: 0                315
Rst: 0 State: a Move: 4 Next State: 2 Impossible: 0 Win: 0 Lose: 0                325
Rst: 0 State: 2 Move: 7 Next State: b Impossible: 0 Win: 0 Lose: 0                335
Rst: 0 State: b Move: 4 Next State: c Impossible: 0 Win: 0 Lose: 0                345
Rst: 0 State: c Move: 3 Next State: c Impossible: 0 Win: 0 Lose: 1                355
Rst: 0 State: c Move: 5 Next State: c Impossible: 1 Win: 0 Lose: 1                365

Test Case 9: Testing last unvistited & no move loop
Rst: 1 State: 0 Move: 4 Next State: 8 Impossible: 0 Win: 0 Lose: 0                375
Rst: 0 State: 0 Move: 6 Next State: a Impossible: 0 Win: 0 Lose: 0                385
Rst: 0 State: a Move: 4 Next State: 2 Impossible: 0 Win: 0 Lose: 0                395
Rst: 0 State: 2 Move: 5 Next State: e Impossible: 0 Win: 0 Lose: 0                405
Rst: 0 State: e Move: 6 Next State: 4 Impossible: 0 Win: 0 Lose: 0                415
Rst: 0 State: 4 Move: 3 Next State: 4 Impossible: 0 Win: 0 Lose: 0                425
Rst: 0 State: 4 Move: 6 Next State: e Impossible: 0 Win: 0 Lose: 0                435
Rst: 0 State: e Move: 4 Next State: c Impossible: 0 Win: 0 Lose: 0                445
Rst: 0 State: c Move: 4 Next State: c Impossible: 1 Win: 0 Lose: 1                455
$finish called from file "tb.v", line 157.
$finish at simulation time                462
          V C S   S i m u l a t i o n   R e p o r t
Time: 462
CPU Time:      0.180 seconds;       Data structure size:   0.0Mb
Tue Oct 24 15:45:54 2023
[lkelly@linux22405 Lab5]$
```

Description: Here is our output that shows each of our testcases.

Post Lab Questions

• What problems did you encounter while testing your steps yourself?

Throughout the lab we came across an issue with the impossibles being triggered at the wrong time. We did not have the correct flag being set for the output. Once we fixed that we re designed the testbench and then verified our output.

• Did any problems arise when demonstrating for the TA? What were they? Explain your thoughts on how/why these testcases escaped your own testing.

When we were demoing to our TA we did not have a test for no moves and jumping back to a previous state. We also did not realized that we visited all but one state in our initial tests. We added this without any issues in the state transitions and then re-demoed. The only other thing that we came across was our implementation of the mealy state machine and our next state output. We did not realize that this was an issues because next state is being assigned in an always (*) block and not a clocked block. This produces the issue of timings transitions and synchronization which could pose bigger issues in a larger state machine implementation.

Conclusion

Throughout this lab we further developed our ability to pass more complex logic into a module and analyze the states and output to implement a simple game. This taught us to organize our states in sequential orders and how to implement a win lose setup. Overall, the biggest take away was the focus on designing the state machine with the test cases in mind and being able to understand the end goal during our development cycle.

Appendix

testbench:

source code:
https://drive.google.com/file/d/1_h3avf4qOew-C75menaV4gvkkp90m11_/view?usp=sharing

state machine:

source code:
https://drive.google.com/file/d/16eDAy_Z93_PdVA0_s3Bm36oS0apxxvpZ/view?usp=sharing