
# Lab 5: State Machine Game

T 2pm-5pm

Zichen Huang and Jack Landers

**Introduction**

In this lab we designed a state machine with 16 states for a game where the player could win, lose, or attempt an impossible move depending on their input and the current state of the game. The losing scenario occurs in states where the fox and chicken are left on the same side, or the chicken and seed are left on the same side. Impossible inputs are where the user tries to change the side of the chicken, seeds, or fox, when the farmer is not on the same side.

**Procedure**

For our code we represented each element of the game with a bit, with 1 signifying the element is on the winning side, and a 0 representing the initial side. Our state would change on each clock cycle unless reset is asserted or input is 0. To determine the next state on every negative edge clock cycle we had a case statement for every current state, with a nested case statement for each move. In our cases if a move was impossible the state would remain the current and impossible would be output, communicating to the user that the move was not registered. Win would be asserted when the final state 1111 was reached and lose would be asserted for any of the possible lose states.

**Result**

Our testbench started by asserting reset to enable each of the outputs. We then tested the winning pathway to ensure that this output functioned correctly. After we tested cases where all of the possible states were reached, as well as testing that our impossible and losing outputs would be asserted, to check that these were operating.

What problems did you encounter while testing your steps yourself?

We had issues with our testbench where we were not performing the correct state combinations necessary for our test, getting our pathways confused. There were also mistakes in our cases where we were performing the wrong operations than we meant to.

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

We had a problem with our always block around our case statement where we were running our cases every time move was asserted. We did this because we misconstrued that our outputs should be asserted on the move. This was a problem

which Sunny explained because had our testbench changed move multiple times in one clock cycle, then the state machine would be dysfunctional. We changed this by only running the always block on a negative edge parameter.

**Game Source Code:**

```
module FarmerGame (clk, rst, move, w, l, im);

input clk, rst;
input [2:0]move;		//3-bit input for the move:
					//000-no move
					//001-move farmer
					//010-move farmer with fox
					//011-move farmer with chicken
					//100-move farmer with seeds
output w, l;
output reg im;
reg [3:0]ns, cs;

//Reset when rst is asserted
always @(posedge clk) begin
	if(rst) cs <= 4'b0000;
	else cs <= ns;
end

//State stays the same when w or l is asserted
always @(*) begin
	im = 0;
	case (cs)
		4'b0000: case (move)
				3'b000: ns = 4'b0000;
				3'b001: ns = 4'b1000;
				3'b010: ns = 4'b1100;
				3'b011: ns = 4'b1010;
				3'b100:	ns = 4'b1001;
			endcase
		4'b0001: case (move)
				3'b000: ns = 4'b0001;
				3'b001: ns = 4'b1001;
				3'b010: ns = 4'b1101;
				3'b011: ns = 4'b1011;
```

```
                3'b100: begin im = 1; ns = 4'b0001; end
        endcase
4'b0010: case (move)
                3'b000: ns = 4'b0010;
                3'b001: ns = 4'b1010;
                3'b010: ns = 4'b1110;
                3'b011: begin im = 1; ns = 4'b0010; end
                3'b100:         ns = 4'b1011;
        endcase
4'b0011: case (move)
                3'b000: ns = 4'b0011;
                3'b001: ns = 4'b0011;
                3'b010: ns = 4'b0011;
                3'b011: ns = 4'b0011;
                3'b100:         ns = 4'b0011;
        endcase
4'b0100: case (move)
                3'b000: ns = 4'b0100;
                3'b001: ns = 4'b1100;
                3'b010: begin im = 1; ns = 4'b0100; end
                3'b011: ns = 4'b1110;
                3'b100:         ns = 4'b1101;
        endcase
4'b0101: case (move)
                3'b000: ns = 4'b0101;
                3'b001: ns = 4'b1101;
                3'b010: begin im = 1; ns = 4'b0101; end
                3'b011: ns = 4'b1111;
                3'b100:         begin im = 1; ns = 4'b0101; end
        endcase
4'b0110: case (move)
                3'b000: ns = 4'b0110;
                3'b001: ns = 4'b0110;
                3'b010: ns = 4'b0110;
                3'b011: ns = 4'b0110;
                3'b100:         ns = 4'b0110;
        endcase
4'b0111: case (move)
                3'b000: ns = 4'b0111;
                3'b001: ns = 4'b0111;
```

```
                3'b010: ns = 4'b0111;
                3'b011: ns = 4'b0111;
                3'b100:         ns = 4'b0111;
        endcase
4'b1000: case (move)
                3'b000: ns = 4'b1000;
                3'b001: ns = 4'b1000;
                3'b010: ns = 4'b1000;
                3'b011: ns = 4'b1000;
                3'b100:         ns = 4'b1000;
        endcase
4'b1001: case (move)
                3'b000: ns = 4'b1001;
                3'b001: ns = 4'b1001;
                3'b010: ns = 4'b1001;
                3'b011: ns = 4'b1001;
                3'b100:         ns = 4'b1001;
        endcase
4'b1010: case (move)
                3'b000: ns = 4'b1010;
                3'b001: ns = 4'b0010;
                3'b010: begin im = 1; ns = 4'b1010; end
                3'b011: ns = 4'b0000;
                3'b100:         begin im = 1; ns = 4'b1010; end
        endcase
4'b1011: case (move)
                3'b000: ns = 4'b1011;
                3'b001: ns = 4'b0011;
                3'b010: begin im = 1; ns = 4'b1011; end
                3'b011: ns = 4'b0001;
                3'b100:         ns = 4'b0010;
        endcase
4'b1100: case (move)
                3'b000: ns = 4'b1100;
                3'b001: ns = 4'b1100;
                3'b010: ns = 4'b1100;
                3'b011: ns = 4'b1100;
                3'b100:         ns = 4'b1100;
        endcase
4'b1101: case (move)
```

```
                        3'b000: ns = 4'b1101;
                        3'b001: ns = 4'b0101;
                        3'b010: ns = 4'b0001;
                        3'b011: begin im = 1; ns = 4'b1101; end
                        3'b100:         ns = 4'b0100;
                    endcase
            4'b1110: case (move)
                        3'b000: ns = 4'b1110;
                        3'b001: ns = 4'b0110;
                        3'b010: ns = 4'b0010;
                        3'b011: ns = 4'b0100;
                        3'b100:         begin im = 1; ns = 4'b1110; end
                    endcase
            4'b1111: case (move)
                        3'b000: ns = 4'b1111;
                        3'b001: ns = 4'b1111;
                        3'b010: ns = 4'b1111;
                        3'b011: ns = 4'b1111;
                        3'b100:         ns = 4'b1111;
                    endcase
        endcase
end

//Define the winning and losing states
assign w = (cs == 4'b1111);
assign l = (cs == 4'b1000 || cs == 4'b1100 || cs == 4'b1001 || cs == 4'b0011 || cs ==
4'b0110);

endmodule
```

**Testbench:**

```
module testbench();
reg clk,rst;
reg [2:0] move;
wire win,lose,impossible;

FarmerGame dut(clk, rst, move, win, lose, impossible);

always begin
#5 clk = ~clk;
```

```
$display("move = %d, win = %b, lose = %b, impossible = %b, clk = %b, rst = %b, @
%d", move, win, lose, impossible, clk, rst, $time);
#5 clk = ~clk;
end

initial begin
$display("Start of test");
clk<=0;
rst <=1; move <=0;
#10;
rst <=0; move <=0; //test no move
#10;
rst <=0; move <=3; //test winning case
#10;
rst <=0; move <=1;
#10;
rst <=0; move <=2;
#10;
rst <=0; move <=3;
#10;
rst <=0; move <=4;
#10;
rst <=0; move <=1;
#10;
rst <=0; move <=3; //expect win
#10;
rst <=1; move <=0; //test rst after win
#10;
rst <=0; move <=3;
#10;
rst <=0; move <=0; //test no move
#10;
rst <=0; move <=3;
#10;
rst <=0; move <=1; //test lose
#10;
rst <=1; move <=0; //test rst after lose
#10;
rst <=0; move <=3; //test impossible
#10;
```

```
rst <=0; move <=4;
#10;
rst <=0; move <=1; //test loop
#10;
rst <=0; move <=4;
#10;
rst <=0; move <=3;
#10;
rst <=0; move <=2;
#10;
rst <=0; move <=4;
#10;
rst <=0; move <=3;
#10;
rst <=0; move <=0;
#10;
rst <=0; move <=1;
#10;
rst <=0; move <=0;
#10;
$finish;
end
endmodule
```