CSCI 355 - Lab Final Report

Student: Luka Karanovic
Student #: 665778833
Instructor: Ajay Shrestha

Priority Encoder

Verilog:

Filename: PriorityEncoder.v

```
module PriorityEncoder(
    input wire [3:0] W,
    output wire [1:0] Y,
    output wire Z
    );

    reg [1:0] Y_reg;
    reg Z_reg;

    assign Y = Y_reg;
    assign Z = Z_reg;

    always @(W) begin
        Z_reg = 1'b1;
        casex (W)
            4'b0001: Y_reg = 2'b00;
            4'b001x: Y_reg = 2'b01;
            4'b01xx: Y_reg = 2'b10;
            4'b1xxx: Y_reg = 2'b11;
            default: begin // 0000 case
                Z_reg = 1'b0;
                Y_reg = 2'bxx;
            end
        endcase
     end

endmodule
```

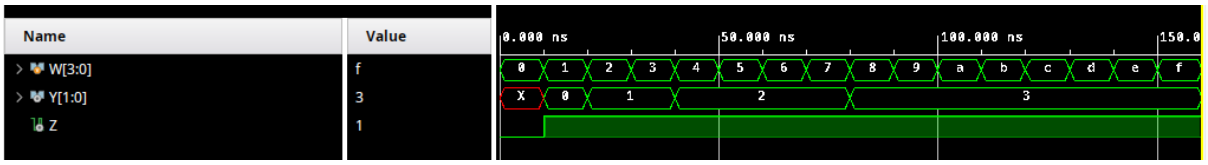
Testbench:

Filename: PriorityEncoder_tb.v

```
module PriorityEncoder_tb();

    reg [3:0] W;
    wire [1:0] Y;
    wire Z;

    PriorityEncoder UT (W, Y, Z);

    initial begin

        W = 4'b0000; #10;
```

```
        W = 4'b0001; #10;
        W = 4'b0010; #10;
        W = 4'b0011; #10;
        W = 4'b0100; #10;
        W = 4'b0101; #10;
        W = 4'b0110; #10;
        W = 4'b0111; #10;
        W = 4'b1000; #10;
        W = 4'b1001; #10;
        W = 4'b1010; #10;
        W = 4'b1011; #10;
        W = 4'b1100; #10;
        W = 4'b1101; #10;
        W = 4'b1110; #10;
        W = 4'b1111; #10;

        $finish;
    end

endmodule
```

Waveform:



Moore Modulo-8 Counter

Verilog:

Filename: MooreModulo.v

```
module MooreModulo(
    input wire w, Clk, Resetn,
    output reg [2:0] z
    );
    reg [2:0] y, Y; // Current and next state
    // States for current value of output.
    parameter A = 3'b000, B = 3'b001, C = 3'b010, D = 3'b011, E = 3'b100, F = 3'b101, G =
3'b110, H = 3'b111;

    always @(w, y) begin
        case (y)
            A: if (w) Y = B;
```

```
            else Y = y;
        B: if (w) Y = C;
            else Y = y;
        C: if (w) Y = D;
            else Y = y;
        D: if (w) Y = E;
            else Y = y;
        E: if (w) Y = F;
            else Y = y;
        F: if (w) Y = G;
            else Y = y;
        G: if (w) Y = H;
            else Y = y;
        H: if (w) Y = A;
            else Y = y;
        default: Y = 3'bxxx; // Don't care

    endcase
    z = y;
  end

  always @(negedge Resetn, posedge Clk) begin
    #10;

    if (Resetn == 0) y <= A;
    else y <= Y;
  end

endmodule
```

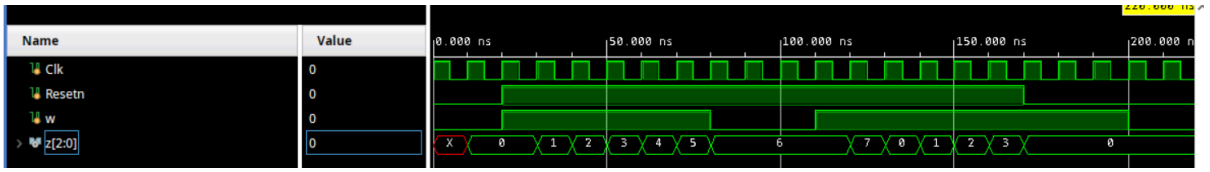
Testbench:

Filename: MooreModulo_tb.v

```
module MooreModulo_tb();

  reg Clk = 0;
  reg Resetn = 0;
  reg w;
  wire [2:0] z;

  MooreModulo UT(w, Clk, Resetn, z);

  always begin
    Clk = ~Clk;
    #5;
  end

  initial begin
    w = 0; #20;
```

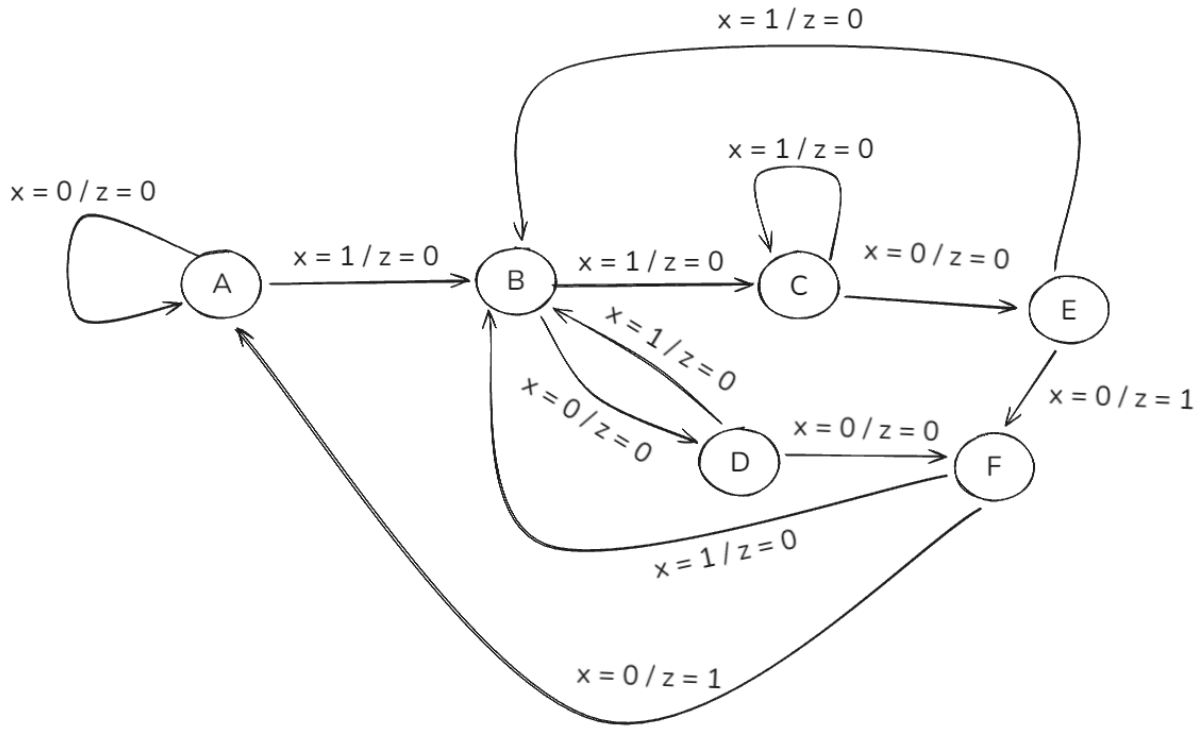
```
        Resetn = 1;
        w = 1; #60;
        w = 0; #30;
        w = 1; #40;
        w = 1; #20;
        Resetn = 0;
        w = 1; #30;
        w = 0; #20;
        $finish;
    end
endmodule
```

Waveform:



Mealy

State Diagram:



Verilog:

Filename: Mealy.v

```
module Mealy(
```

```
input wire Clk, x,
output reg z
);

reg [2:0] y, Y;
parameter A = 3'b000, B = 3'b001, C = 3'b010, D = 3'b011, E = 3'b100, F = 3'b101;

// Determine next state and output z based on input w.
always @(x, y) begin
    case (y)
        A: if (x) begin
            Y = B;
            z = 0;
        end
        else begin
            Y = A;
            z = 0;
        end
        B: if (x) begin
            Y = C;
            z = 0;
        end
        else begin
            Y = D;
            z = 0;
        end
        C: if (x) begin
            Y = C;
            z = 0;
        end
        else begin
            Y = E;
            z = 0;
        end
        D: if (x) begin
            Y = B;
            z = 0;
        end
        else begin
            Y = F;
            z = 0;
        end
        E: if (x) begin
            Y = B;
            z = 0;
        end
        else begin
            Y = F;
```

```
            z = 1;
        end
        F: if (x) begin
            Y = B;
            z = 0;
        end
        else begin
            Y = A;
            z = 1;
        end
        default: Y = A; // If no current state, set next state to inital state
    endcase
end

// Handle clock updates and reset, reset is async and happens when it is set to 0.
always @(posedge Clk) begin
    #10;
    y <= Y;
end

endmodule
```

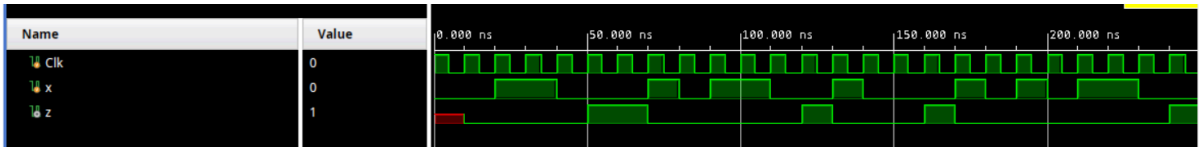
Testbench:

Filename: Mealy_tb.v

```
module Mealy_tb();
    reg Clk = 0;
    reg x;
    wire z;

    Mealy UT(Clk, x, z);

    always begin
        Clk = ~Clk;
        #5;
    end

    initial begin
        x = 0; #10;
        x = 0; #10;
        x = 1; #10;
        x = 1; #10;
        x = 0; #10;
        x = 0; #10;
        x = 0; #10;
```

```
        x = 1; #10;
        x = 0; #10;
        x = 1; #10;
        x = 1; #10;
        x = 0; #10;
        x = 0; #10;
        x = 1; #10;
        x = 0; #10;
        x = 0; #10;
        x = 0; #10;
        x = 1; #10;
        x = 0; #10;
        x = 1; #10;
        x = 0; #10;
        x = 1; #10;
        x = 1; #10;
        x = 0; #10;
        x = 0; #10;
        $finish;
    end

endmodule
```

Waveform:



Serial Adder FSM

Verilog:

Filename: serialAdder.v

```
module shiftReg #(parameter n = 8)
(
    input Clk,                  // Pos-edge clock
    input wire E,               // Enable signal
    input wire L,               // Load
    input wire [n-1:0] pdata,   // Parallel data
    input wire w,               // Serial data
    output reg [n-1:0] Y        // Output
);
    integer i;
    always @(posedge Clk) begin
        if (E) begin
            if (L) begin // Parallel load
                Y <= pdata;
```

```
            end
            else begin // Serial load
                for (i = 0; i < n-1; i = i+1) begin
                    Y[i] <= Y[i+1];
                end
                Y[n-1] <= w;
            end
        end
        // Else: E is 0 and Y holds its value
    end
endmodule

module serialAdder(
    input Clk,              // Pos-edge clock
    input wire Reset,       // Enable signal
    input wire [7:0] A,     // Input data A
    input wire [7:0] B,     // Input data B
    output wire [7:0] S     // Sum
);
    wire a, b;              // Serial output from shift regs
    reg E;                  // Enable for sum shift reg
    reg s;                  // Serial sum input to sum shift reg
    reg Cin = 0, Cout;          // Current and next state for Adder FSM
    reg [3:0] counter = 4'b1000;    // Counter initialized to 1000 like diagram
    parameter G = 1'b0, H = 1'b1;   // Possible states (Cin and Cout values)

    shiftReg shiftA(Clk, 1, Reset, A, 0, a);
    shiftReg shiftB(Clk, 1, Reset, B, 0, b);
    shiftReg shiftS(Clk, E, Reset, 0, s, S);

    // Mealy Adder FSM
    always @(a, b, Cin) begin
        case (Cin)
            G: if (a ^ b) begin // ab = 10 or 01
                s = 1;
                Cout = G;
            end
            else if (a && b) begin // ab = 11
                s = 0;
                Cout = H;
            end
            else begin // ab = 00
                s = 0;
                Cout = G;
            end
            H: if (a ^ b) begin // ab = 10 or 01
                s = 0;
                Cout = H;
```

```
            end
            else if (a && b) begin // ab = 11
                s = 1;
                Cout = H;
            end
            else begin // ab = 00
                s = 1;
                Cout = G;
            end
            default: Cout = G; // If nothing, Cout is 0

        endcase
    end

    always @(posedge Clk, Reset) begin
        #10;
        Cin <= Cout;
        if (|counter) begin // If there is a 1 in counter
            E = 1;
            counter = counter + 1;

        end
        else begin // Low E signal after one n-bit addition.
            E = 0;
        end

        if (Reset) begin
            counter = 4'b1000;
            Cin <= 0;
        end
    end

endmodule
```

Testbench:

Filename: serialAdder_tb.v

```
module serialAdder_tb();

    reg Clk = 0;
    reg Reset = 1;
    reg [7:0] A, B;

    wire [7:0] S;

    serialAdder UT(Clk, Reset, A, B, S);
```

```
    always begin
        Clk = ~Clk;
        #5;
    end

    initial begin
        A = 8'hA4;
        B = 8'h2D;
        #15; // So things can load
        Reset = 0; #120;
        $finish;

    end

endmodule
```

Waveform: