Brodric Young

ECEN 340

Lab 7 – State Machines

Introduction

This lab focuses on designing and implementing a Morse Code Digit Decoder using Verilog and an FPGA. The objective is to convert Morse code representations of numbers (0-9) into their corresponding decimal values and display them on a seven-segment display. The project incorporates state machine logic to cycle through numbers sequentially with each clock pulse, unless a specific input is provided via switches. Additionally, an asynchronous reset function is implemented to allow for manual resetting of the system.

To achieve this, the design uses a clock divider module to slow down the clock signal, ensuring the seven-segment display updates at a human-readable rate. The core logic consists of a finite state machine that translates Morse code inputs into numerical values, which are then passed to a seven-segment decoder module for display. The system operates in two modes: automatic sequencing, where numbers increment sequentially with the clock, and manual mode, where the user can directly input Morse code using switches. This lab provides hands-on experience with digital logic design, state machine implementation, and hardware description languages like Verilog, reinforcing fundamental concepts in embedded systems and FPGA programming.

Verilog Code Functionality

The Verilog code for this project consists of several key modules that work together to decode Morse code representations and display the corresponding decimal digits. The main module (morse_code_decoder) takes clock input, user switches, and a reset button as inputs. The clock divider module (clk_div) generates a slower clock signal to drive the state transitions at a reasonable speed. The seven-segment decoder module (decoder_7seg) converts the decoded digit into the correct segment pattern for display.

The finite state machine is responsible for determining the correct digit based on the Morse code input. If the most significant switch (sw[5]) is high, the system enters manual mode, where the user can directly input a 5-bit Morse code representation using the lower switches (sw[4:0]). Otherwise, the system follows an automatic sequence, cycling through numbers 0-9 with each clock pulse. A reset button (btnD) allows the user to reset the system back to zero at any time.

Code

Top module

morse_code_decoder.v

uments/- Hardware Labs/digital_systems-verilog/ecen340/Lab7_State_Machines/morse_code_decoder/morse_code_decoder.srcs/sources_1/new/morse

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date: 03/04/2025 05:43:07 PM
7  // Design Name:
8  // Module Name: morse_code_decoder
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21
22
23 module morse_code_decoder(
24     input clk,
25     input [5:0] sw,
26     input btnD, //Reset
27     output [6:0] seg,
28     output [4:0] led,
29     output dp,
30     output [3:0]an
31     );
32
33     assign an = 4'b1110;        // only the right-most digit display
34
35     wire clkdiv;
36     reg [4:0] digit_code;       // morse code for the digit to display
37     wire [4:0] digit_code_mux;  // Mux between switch value and state machine
38     reg [3:0] decoded_digit;    // digit of the morse code
39     reg [4:0] next_state;       // counts up from 0-9
40
41     //modules
42     clk_div #(.count(25)) T1 (.clk(clk), .rst(btnD), .clkd(clkdiv));
43     decoder_7seg D1 (.sw(decoded_digit), .seg(seg), .dp(dp));
44
45     // Instantly take switch value when sw[5] is on, otherwise go
46     // with the digit_code updated at the clk
47     assign digit_code_mux = sw[5] ? sw[4:0] : digit_code;
48
49     always @(posedge clkdiv or posedge btnD) begin
50         if (btnD) digit_code <= 5'b11111;          // Reset the digit to '0'
51         else if (!sw[5]) digit_code <= next_state;  // if switch 5 is down, go to the next state
52         else if (sw[5]) digit_code <= sw[4:0];      // if switch 5 is up, next state is value of the input switches
53     end
```

// module continues on next page

```
    // decode the morse code into the digit to display and
    // update the next state based on current state of digit_code
    always @* begin
        case (digit_code_mux)
            5'b11111: begin             // '-----' -> 0
                decoded_digit <= 0;
                if (!sw[5]) next_state = 5'b01111;
                end

            5'b01111: begin             // '.----' -> 1
                decoded_digit <= 1;
                if (!sw[5]) next_state = 5'b00111;
                end

            5'b00111: begin             // '..---' -> 2
                decoded_digit <= 2;
                if (!sw[5]) next_state = 5'b00011;
                end

            5'b00011: begin             // '...--' -> 3
                decoded_digit <= 3;
                if (!sw[5]) next_state = 5'b00001;
                end

            5'b00001: begin             // '....-' -> 4
                decoded_digit <= 4;
                if (!sw[5]) next_state = 5'b00000;
                end

            5'b00000: begin             // '.....' -> 5
                decoded_digit <= 5;
                if (!sw[5]) next_state = 5'b10000;
                end

            5'b10000: begin             // '-....' -> 6
                decoded_digit <= 6;
                if (!sw[5]) next_state = 5'b11000;
                end

            5'b11000: begin             // '--...' -> 7
                decoded_digit <= 7;
                if (!sw[5]) next_state = 5'b11100;
                end

            5'b11100: begin             // '---..' -> 8
                decoded_digit <= 8;
                if (!sw[5]) next_state = 5'b11110;
                end

            5'b11110: begin             // '----.' -> 9
                decoded_digit <= 9;
                if (!sw[5]) next_state = 5'b11111;
                end

            default: decoded_digit <= 0; // Default to 0 if invalid sequence
        endcase
    end

    assign led = digit_code_mux; // display the morse code on the leds (on for dash, off for dot)
endmodule
```

Clock divider

clk_div.v

nents/- Hardware Labs/digital_systems-verilog/ecen340/Lab7_State_Machines/morse_code_decoder/morse_code_decoder.srcs/sources_1/new/clk_div.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer: Brodric Young & Karl Richards
5  //
6  // Create Date: 02/14/2025 03:16:20 PM
7  // Design Name:
8  // Module Name: clk_div
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description: divide the clock frequency giving a shorter frequency
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21
22 // module to divide the clock
23 module clk_div #(
24     parameter count = 17 // default of 17 is 381.5 Hz
25 )(
26     input clk,  // 100MHz
27     input rst,
28     output clkd
29 );
30
31     // 26-bit counter
32     reg [25:0] counter;
33     reg clkdiv;
34
35     always @(posedge clk or posedge rst) begin
36         if (rst)
37             counter <= 26'b0; // Reset the counter
38         else
39             counter <= counter + 1'b1; // Increment the counter
40     end
41
42     // Output the counter value
43     always  @ (posedge clk or posedge rst)
44         begin
45             if(rst)
46                 clkdiv <= 1'b0;
47             else
48                 clkdiv <= counter [count];
49         end
50
51     assign clkd = clkdiv;
52 endmodule
53
54
```

7 segment display decoder

decoder_7seg.v

-lardware Labs/digital_systems-verilog/ecen340/Lab7_State_Machines/morse_code_decoder/morse_code_decoder.srcs/sources_1/new/decoder_7seg.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer: Brodric Young & Karl Richards
5  //
6  // Create Date: 02/14/2025 03:30:59 PM
7  // Design Name:
8  // Module Name: decoder_7seg
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description: decode what hexadecimal number to display based on input switch combination
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21
22
23 module decoder_7seg #(
24     parameter n = 4 // default 4-bit input for digit
25 )(
26     input [n-1:0] sw,
27     output reg [6:0] seg,    // 7-segment display output
28     output dp              // Decimal point (set to always off)
29 );
30
31     assign dp = 1'b1;         // Decimal point off
32
33
34     always @ (sw)
35         case (sw)
36             4'b0000: seg = 7'b1000000;  // 0
37             4'b0001: seg = 7'b1111001;  // 1
38             4'b0010: seg = 7'b0100100;  // 2
39             4'b0011: seg = 7'b0110000;  // 3
40             4'b0100: seg = 7'b0011001;  // 4
41             4'b0101: seg = 7'b0010010;  // 5
42             4'b0110: seg = 7'b0000010;  // 6
43             4'b0111: seg = 7'b1111000;  // 7
44             4'b1000: seg = 7'b0000000;  // 8
45             4'b1001: seg = 7'b0011000;  // 9
46             4'b1010: seg = 7'b0001000;  // A
47             4'b1011: seg = 7'b0000011;  // b
48             4'b1100: seg = 7'b1000110;  // C
49             4'b1101: seg = 7'b0100001;  // d
50             4'b1110: seg = 7'b0000110;  // E
51             4'b1111: seg = 7'b0001110;  // F
52         endcase
53 endmodule
54
55
```

Conclusion

By implementing this design, the lab demonstrates how state machines, clock division, and combinational logic can be used to interpret and display coded input data. This project strengthens understanding of hardware-based state machine design and its practical applications in embedded systems. We ran into several errors which we were eventually able to debug and correct leaving us with the this completely functional result.