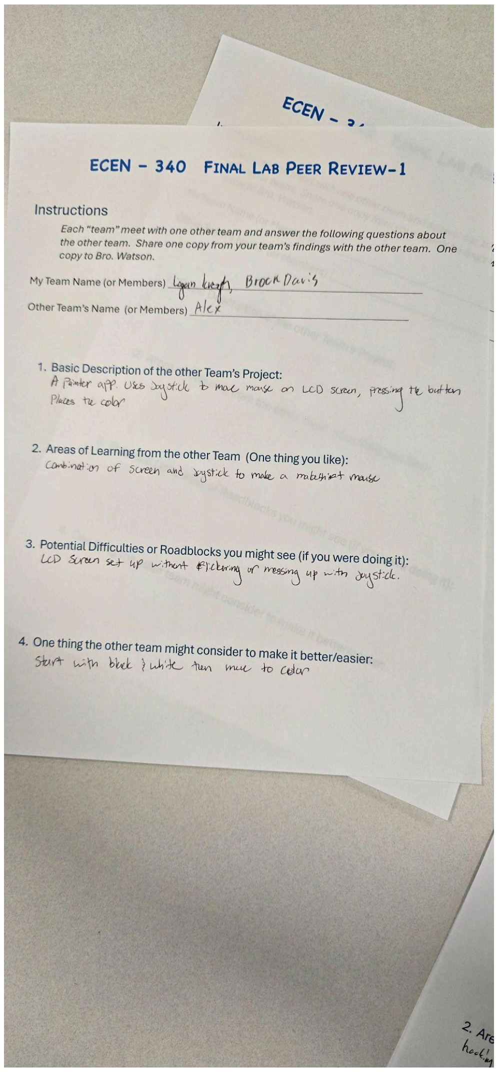
This is a test First and second peer review notes. Include the names of those present and also document the comments that were made.



ECEN - 340 FINAL LAB PEER REVIEW-1

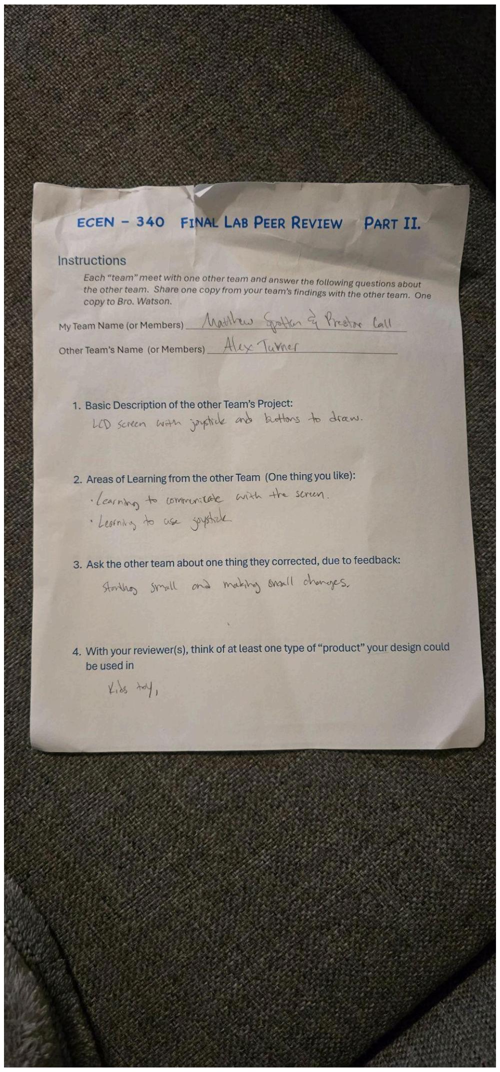
Instructions

Each "team" meet with one other team and answer the following questions about the other team. Share one copy from your team's findings with the other team. One copy to Bro. Watson.

My Team Name (or Members) Logan Knapp, Brock Davis

Other Team's Name (or Members) Alex

1. Basic Description of the other Team's Project:
A Painter app. Uses Joystick to move mouse on LCD screen, pressing the button Places the color
2. Areas of Learning from the other Team (One thing you like):
Combination of Screen and Joystick to make a [illegible] mouse
3. Potential Difficulties or Roadblocks you might see (if you were doing it):
LCD Screen set up without flickering or messing up with Joystick.
4. One thing the other team might consider to make it better/easier:
Start with black & white then move to color



ECEN - 340 FINAL LAB PEER REVIEW PART II.

Instructions

Each "team" meet with one other team and answer the following questions about the other team. Share one copy from your team's findings with the other team. One copy to Bro. Watson.

My Team Name (or Members) Matthew Spotten & Preston Call

Other Team's Name (or Members) Alex Turner

1. Basic Description of the other Team's Project:
LCD screen with joystick and buttons to draw.
2. Areas of Learning from the other Team (One thing you like):
   - Learning to communicate with the screen.
   - Learning to use joystick
3. Ask the other team about one thing they corrected, due to feedback:
Starting small and making small changes.
4. With your reviewer(s), think of at least one type of "product" your design could be used in
Kids toy,

These peer reviews were conducted based on my original project. Due to hardware failure I had to pivot to a new project. Since we were low on time I was unable to redo these peer reviews. That being said I did my best to act on the feedback I did receive. I started small and got more complex and I made sure that the display worked and did not glitch or flicker in any way.

A description of the project. Include diagrams of the project if it includes external components.

As I am sure you know I originally was going to make a drawing pad. However I was having a lot of trouble with the LCD and SPI connection. I was able to get something to display but then I bricked my display and decided I wasn't worth breaking more displays so I went with a different project.

My project is like a bob-it game. You have until the timer runs out to press the correct button. You have 4 options left, right, up and down. If you get the button correct it increases your score which the on board leds show. If you get the button wrong it decreases the time by 2 seconds. There is a menu which scrolls welcoming the user to the gauntlet and explaining the user can press up to start or right to see the high score. Once you press right you can press right again to get back to the main menu. Once the time runs out there are two ending messages. One is if the user got a new high score the other is just a regular one. Both messages inform the user they can play again if they press up.

Verilog code.

Final_Project.v:

```
`timescale 1ns / 1ps


module Final_Project(

   input clk,

   input btnC, input btnR, input btnL, input btnU, input btnD,

   output [15:0] led,

   output [6:0] seg,

   output [3:0] an

);


   wire left, right, up, down, center;

   wire mem_we;

   wire [15:0] score_read, score_write, score;
   wire [1:0] dir;
   wire [6:0] count_down;

   // UPDATED: 4 bits for debug_state
   wire [3:0] debug_state;
   wire new_record;

   debounce_buttons U_Debounce(
       .clk(clk),
       .btnL(btnL), .btnR(btnR), .btnU(btnU), .btnD(btnD), .btnC(btnC),
       .left(left), .right(right), .up(up), .down(down), .center(center)
   );

   game_logic U_Game(
       .clk(clk),
       .rst(center),
       .btnL(left), .btnR(right), .btnU(up), .btnD(down),
       .dir_out(dir),
       .count_down(count_down),
       .score(score),
       .score_read(score_read),
       .score_write(score_write),
       .mem_we(mem_we),
       .debug_state(debug_state),
```

```verilog
    .new_record(new_record)
);

memory U_Memory(
    .we(mem_we), .clk(clk), .addr(4'b0),
    .data_in(score_write), .data_out(score_read)
);

segdisplay_driver U_SegDisplay(
    .clk(clk),
    .rst(center),
    .count_down(count_down),
    .dir(dir),
    .score(score),
    .high_score_val(score_read), // CONNECTED: Pass read score to display
    .game_state(debug_state),
    .is_new_record(new_record),
    .seg(seg),
    .an(an)
);

// LEDs
assign led[3:0] = debug_state; // Shows 4-bit state
assign led[4] = up;
```

```
    assign led[5] = down;

    assign led[6] = left;

    assign led[7] = right;

    assign led[15:8] = score[7:0];


endmodule
```

debounce_buttons.v:

```
`timescale 1ns / 1ps


module debounce_buttons (

     input clk,

     input btnL, btnC, btnR, btnU, btnD,

     output left, center, right, up, down

     );

     wire clk_deb;


    debounce_div div1  (.clk(clk), .clk_deb(clk_deb));

    btn_debounce u_btn (.clk(clk_deb), .btn_in(btnU), .btn_status(up));

    btn_debounce d_btn (.clk(clk_deb), .btn_in(btnD), .btn_status(down));

    btn_debounce l_btn (.clk(clk_deb), .btn_in(btnL), .btn_status(left));

    btn_debounce r_btn (.clk(clk_deb), .btn_in(btnR), .btn_status(right));

    btn_debounce c_btn (.clk(clk_deb), .btn_in(btnC), .btn_status(center));

endmodule
```

debounce_div.v:

```
`timescale 1ns / 1ps

module debounce_div(
    input clk,
    output clk_deb
    );
    reg [15:0] cnt;



    assign clk_deb = cnt[15];

    initial cnt = 0;
    always @(posedge clk)
        cnt <= cnt + 1;

endmodule
```

btn_debounce.v:

```
`timescale 1ns / 1ps

module btn_debounce(
    input clk,
    input btn_in,
    output wire btn_status
    );
    reg [19:0] btn_shift;

    always @ (posedge clk)
        btn_shift <= {btn_shift[18:0], btn_in};

    assign btn_status = &btn_shift;
endmodule
```

game_logic.v:

```
`timescale 1ns / 1ps

module game_logic(
    input clk,
    input rst,
    input btnL, btnR, btnU, btnD,
    output reg [1:0] dir_out,
    output reg [6:0] count_down,
    output reg [15:0] score,
    input [15:0] score_read,
    output reg [15:0] score_write,
    output reg mem_we,
    output [3:0] debug_state,
    output reg new_record
);

    // STATE DEFINITIONS
    localparam IDLE           = 4'd0;
    localparam NEW_ROUND      = 4'd1;
    localparam WAIT_INPUT     = 4'd2;
    localparam CHECK_FAIL     = 4'd3;
    localparam SAVE_SCORE     = 4'd4;
    localparam GAME_OVER      = 4'd5;
    localparam WAIT_REL       = 4'd6;
    localparam WAIT_REL_RETRY = 4'd7;
    localparam WAIT_TO_HS     = 4'd8;
    localparam VIEW_HS        = 4'd9;
    localparam WAIT_TO_IDLE   = 4'd10;
    localparam WAIT_FROM_GO   = 4'd11;

    reg [3:0] state = IDLE;
    reg [31:0] timer_count;
    reg [1:0] dir;

    // LFSR Random Generator linear feedback shift register
    reg [3:0] lfsr = 4'b1011;
```

```
always @(posedge clk) begin
    lfsr <= {lfsr[2:0], lfsr[3] ^ lfsr[2]};
end


// Timer Pulse
wire pulse_1sec;
clk_gen #(.COUNTER_MAX(100_000_000)) timer_pulse_inst (
    .clk(clk), .rst(rst), .en_10hz(pulse_1sec)
);


assign debug_state = state;


initial begin
    score = 0;
    state = IDLE;
    count_down = 30;
    new_record = 0;
end


always @(posedge clk) begin
    if (rst) begin
        state <= IDLE;
        score <= 0;
        mem_we <= 0;
        count_down <= 0;
        dir_out <= 2'd2;
        new_record <= 0;
    end else begin
        mem_we <= 0;
        count_down <= timer_count[6:0];

        case (state)
            // --- MAIN MENU ---
            IDLE: begin
                count_down <= 30;
                dir_out <= 2'd2;
                new_record <= 0;

                if (btnU) begin
                    score <= 0;
                    timer_count <= 30;
                    state <= WAIT_REL;
                end
                else if (btnR) begin
                    state <= WAIT_TO_HS;
                end
            end
```

```
// --- MENU NAVIGATION ---

WAIT_TO_HS: begin

    if (!btnR) state <= VIEW_HS;

end

VIEW_HS: begin

    if (btnR) state <= WAIT_TO_IDLE;

end

WAIT_TO_IDLE: begin

    if (!btnR) state <= IDLE;

end

WAIT_FROM_GO: begin

    if (!btnU) state <= IDLE; // Return to menu only when button released

end


// --- GAMEPLAY ---

WAIT_REL: begin

    if (!btnU && !btnD && !btnL && !btnR) state <= NEW_ROUND;

end

WAIT_REL_RETRY: begin

    if (!btnU && !btnD && !btnL && !btnR) state <= WAIT_INPUT;

end


NEW_ROUND: begin

    dir <= lfsr[1:0];
```

```
            dir_out <= lfsr[1:0];
            state <= WAIT_INPUT;
        end


        WAIT_INPUT: begin
            if (pulse_1sec) begin
                if (timer_count > 0) timer_count <= timer_count - 1;
                else state <= CHECK_FAIL;
            end


            if ((dir != 0 && btnL) || (dir != 1 && btnR) || (dir != 2 && btnU) || (dir != 3 && btnD))
begin
                if (timer_count > 2) begin
                    timer_count <= timer_count - 2;
                    state <= WAIT_REL_RETRY;
                end else begin
                    timer_count <= 0;
                    state <= CHECK_FAIL;
                end
            end
            else if ((dir == 0 && btnL) || (dir == 1 && btnR) || (dir == 2 && btnU) || (dir == 3 &&
btnD)) begin
                score <= score + 1;
                state <= WAIT_REL;
            end
        end

        CHECK_FAIL: begin
            if (score > score_read) begin
                score_write <= score;
                mem_we <= 1;
                new_record <= 1;
                state <= SAVE_SCORE;
            end else begin
                state <= GAME_OVER;
            end
        end

        SAVE_SCORE: begin
            mem_we <= 0;
            state <= GAME_OVER;
        end

        GAME_OVER: begin
            count_down <= 0;
            if (btnU) state <= WAIT_FROM_GO; // NEW: Press U to restart
        end
    endcase
end
```

```
    end
endmodule
```

clk_gen.v:

```
`timescale 1ns / 1ps

module clk_gen #(
    // Default: 100MHz clock / 10Hz target = 10,000,000 cycles
    // If you want it SLOWER (e.g., 1 count per second), change this to 100,000,000
    parameter COUNTER_MAX = 10_000_000
)(
    input clk,
    input rst,
    output reg en_10hz
);

    reg [26:0] count;

    always @ (posedge clk) begin
        if (rst) begin
            count <= 0;
            en_10hz <= 0;
        end else begin
            if (count >= COUNTER_MAX - 1) begin
```

```
                count <= 0;

                en_10hz <= 1; // 1-cycle enable pulse

            end else begin

                count <= count + 1;

                en_10hz <= 0;

            end

        end

    end

endmodule
```

memory.v:

```
`timescale 1ns / 1ps


module memory(

     input wire clk,

     input wire we,

     input wire [3:0] addr,

     input wire [15:0] data_in,

     output reg [15:0] data_out

);

    reg [15:0] mem [15:0];


    initial begin

        mem[0] = 0;
        mem[1] = 0;
        mem[2] = 0;
        mem[3] = 0;
    end

    always@(posedge clk) begin
        if (we)
            mem[addr] <= data_in;
        data_out <= mem[addr];
    end
endmodule
```

segdisplay_driver.v:

```
`timescale 1ns / 1ps

module segdisplay_driver(
    input clk,
    input rst,
    input [6:0] count_down,
    input [1:0] dir,
    input [15:0] score,
    input [15:0] high_score_val,
    input [3:0] game_state,
    input is_new_record,
    output reg [6:0] seg,

    output reg [3:0] an

);


    reg [19:0] refresh_counter;

    always @(posedge clk or posedge rst) begin

        if (rst) refresh_counter <= 0;

        else refresh_counter <= refresh_counter + 1;

    end

    wire [1:0] digit_select = refresh_counter[19:18];


    reg [24:0] scroll_timer;

    always @(posedge clk or posedge rst) begin

        if (rst) scroll_timer <= 0;

        else scroll_timer <= scroll_timer + 1;

    end

    wire scroll_tick = (scroll_timer == 0);


    reg [6:0] scroll_ptr;

    always @(posedge clk or posedge rst) begin

        if (rst) scroll_ptr <= 0;

        else if (scroll_tick) scroll_ptr <= scroll_ptr + 1;

    end
```

```
reg [5:0] char_index;


// Character Constants

localparam C_0=0, C_1=1, C_2=2, C_3=3, C_4=4, C_5=5, C_6=6, C_7=7, C_8=8, C_9=9;

localparam C_L=10, C_r=11, C_U=12, C_d=13, C_SPC=14, C_DASH=15;

localparam C_A=16, C_b=17, C_C=18, C_E=19, C_F=20, C_G=21, C_H=22, C_J=23;

localparam C_n=24, C_o=25, C_P=26, C_S=27, C_t=28, C_y=29, C_I=30, C_M=31;


always @(*) begin

  // DEFAULT ANODES

  case(digit_select)

    2'b00: an = 4'b1110;

    2'b01: an = 4'b1101;

    2'b10: an = 4'b1011;

    2'b11: an = 4'b0111;

  endcase


  // --- STATE DISPLAY LOGIC ---


  // 1. GAME OVER (State 5) or WAIT FROM GO (State 11)

  if (game_state == 4'd5 || game_state == 4'd11) begin

    if (is_new_record) begin

      case(digit_select)

        2'b00: char_index = get_newrecord_char(scroll_ptr + 3);
```

```
            2'b01: char_index = get_newrecord_char(scroll_ptr + 2);
            2'b10: char_index = get_newrecord_char(scroll_ptr + 1);
            2'b11: char_index = get_newrecord_char(scroll_ptr);
        endcase
    end else begin
         case(digit_select)
            2'b00: char_index = get_gameover_char(scroll_ptr + 3);
            2'b01: char_index = get_gameover_char(scroll_ptr + 2);
            2'b10: char_index = get_gameover_char(scroll_ptr + 1);
            2'b11: char_index = get_gameover_char(scroll_ptr);
        endcase
    end
end

// 2. IDLE / MENU (State 0)
else if (game_state == 4'd0 || game_state == 4'd10) begin
    case(digit_select)
        2'b00: char_index = get_welcome_char(scroll_ptr + 3);
        2'b01: char_index = get_welcome_char(scroll_ptr + 2);
        2'b10: char_index = get_welcome_char(scroll_ptr + 1);
        2'b11: char_index = get_welcome_char(scroll_ptr);
    endcase
end
```

```
// 3. VIEW HIGH SCORE (State 9) or WAIT TO HS (State 8)
else if (game_state == 4'd9 || game_state == 4'd8) begin
    case(digit_select)
        2'b00: char_index = get_highscore_char(scroll_ptr + 3);
        2'b01: char_index = get_highscore_char(scroll_ptr + 2);
        2'b10: char_index = get_highscore_char(scroll_ptr + 1);
        2'b11: char_index = get_highscore_char(scroll_ptr);
    endcase
end

// 4. PLAYING
else begin
     case(digit_select)
        2'b00: begin // Direction
            case(dir)
                2'd0: char_index = C_L;
                2'd1: char_index = C_r;
                2'd2: char_index = C_U;
                2'd3: char_index = C_d;
            endcase
        end
        2'b01: char_index = C_SPC; // Blank
        2'b10: char_index = (count_down % 10); // Timer
        2'b11: char_index = (count_down / 10); // Timer
```

```
            endcase
        end
    end

    // --- TEXT FUNCTIONS ---

    // IDLE: "EnTER THE GAUntLEt   PrESS U TO PLAY   PrESS r FOR SCorE      " (Length 58)
    function [5:0] get_welcome_char;
        input [6:0] pos;
        begin
            case(pos % 58)
                0: get_welcome_char = C_E; 1: get_welcome_char = C_n; 2: get_welcome_char =
C_t;
                3: get_welcome_char = C_E; 4: get_welcome_char = C_r; 5: get_welcome_char =
C_SPC;
                6: get_welcome_char = C_t; 7: get_welcome_char = C_H; 8: get_welcome_char =
C_E;
                9: get_welcome_char = C_SPC; 10: get_welcome_char = C_G; 11:
get_welcome_char = C_A;
                12: get_welcome_char = C_U; 13: get_welcome_char = C_n; 14: get_welcome_char
= C_t;
                15: get_welcome_char = C_L; 16: get_welcome_char = C_E; 17: get_welcome_char
= C_t;
                18: get_welcome_char = C_SPC; 19: get_welcome_char = C_SPC; 20:
get_welcome_char = C_SPC;
                21: get_welcome_char = C_P; 22: get_welcome_char = C_r; 23: get_welcome_char =
C_E;
                24: get_welcome_char = C_S; 25: get_welcome_char = C_S; 26: get_welcome_char
= C_SPC;
```

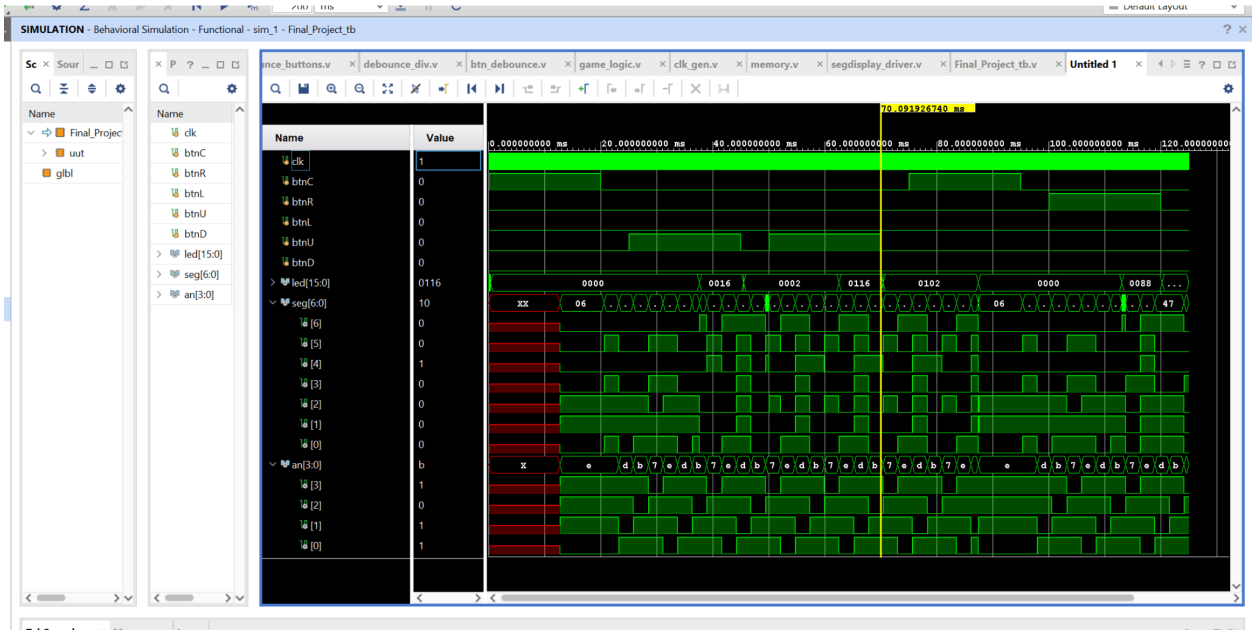
```
            27: get_welcome_char = C_U; 28: get_welcome_char = C_SPC; 29:
get_welcome_char = C_t;

            30: get_welcome_char = C_o; 31: get_welcome_char = C_SPC; 32:
get_welcome_char = C_P;

            33: get_welcome_char = C_L; 34: get_welcome_char = C_A; 35: get_welcome_char
= C_y;

            36: get_welcome_char = C_SPC; 37: get_welcome_char = C_SPC; 38:
get_welcome_char = C_SPC;

            39: get_welcome_char = C_P; 40: get_welcome_char = C_r; 41: get_welcome_char =
C_E;

            42: get_welcome_char = C_S; 43: get_welcome_char = C_S; 44: get_welcome_char
= C_SPC;

            45: get_welcome_char = C_r; 46: get_welcome_char = C_SPC; 47:
get_welcome_char = C_F;

            48: get_welcome_char = C_o; 49: get_welcome_char = C_r; 50: get_welcome_char =
C_SPC;

            51: get_welcome_char = C_S; 52: get_welcome_char = C_C; 53: get_welcome_char
= C_o;

            54: get_welcome_char = C_r; 55: get_welcome_char = C_E;

            default: get_welcome_char = C_SPC;

         endcase

      end

   endfunction



   // HIGH SCORE: "HALL OF FAME   SCorE XX   PrESS r TO bACK      " (Length 46)

   function [5:0] get_highscore_char;

      input [6:0] pos;

      begin

         case(pos % 46)
```

```
            0: get_highscore_char = C_H; 1: get_highscore_char = C_A; 2: get_highscore_char =
C_L;

            3: get_highscore_char = C_L; 4: get_highscore_char = C_SPC; 5:
get_highscore_char = C_o;

            6: get_highscore_char = C_F; 7: get_highscore_char = C_SPC; 8:
get_highscore_char = C_F;

            9: get_highscore_char = C_A; 10: get_highscore_char = C_M; 11:
get_highscore_char = C_E;

            12: get_highscore_char = C_SPC; 13: get_highscore_char = C_SPC; 14:
get_highscore_char = C_SPC;


            15: get_highscore_char = C_S; 16: get_highscore_char = C_C; 17:
get_highscore_char = C_o;

            18: get_highscore_char = C_r; 19: get_highscore_char = C_E; 20:
get_highscore_char = C_SPC;

            21: get_highscore_char = (high_score_val / 10) % 10;

            22: get_highscore_char = (high_score_val % 10);


            23: get_highscore_char = C_SPC; 24: get_highscore_char = C_SPC; 25:
get_highscore_char = C_SPC;

            26: get_highscore_char = C_P; 27: get_highscore_char = C_r; 28:
get_highscore_char = C_E;

            29: get_highscore_char = C_S; 30: get_highscore_char = C_S; 31:
get_highscore_char = C_SPC;

            32: get_highscore_char = C_r; 33: get_highscore_char = C_SPC; 34:
get_highscore_char = C_t;

            35: get_highscore_char = C_o; 36: get_highscore_char = C_SPC; 37:
get_highscore_char = C_b;

            38: get_highscore_char = C_A; 39: get_highscore_char = C_C; 40:
get_highscore_char = C_F;
```

```
            default: get_highscore_char = C_SPC;
         endcase
      end
   endfunction

   // GAME OVER: "LEGEnD FALLEn   SCorE XX   PrESS U TO MEnU      " (Length 47)
   function [5:0] get_gameover_char;
      input [6:0] pos;
      begin
          case(pos % 47)
            0: get_gameover_char = C_L; 1: get_gameover_char = C_E; 2: get_gameover_char =
C_G;
            3: get_gameover_char = C_E; 4: get_gameover_char = C_n; 5: get_gameover_char =
C_d;
            6: get_gameover_char = C_SPC; 7: get_gameover_char = C_F; 8:
get_gameover_char = C_A;
            9: get_gameover_char = C_L; 10: get_gameover_char = C_L; 11: get_gameover_char
= C_E;
            12: get_gameover_char = C_n; 13: get_gameover_char = C_SPC; 14:
get_gameover_char = C_SPC;


            15: get_gameover_char = C_S; 16: get_gameover_char = C_C; 17:
get_gameover_char = C_o;
            18: get_gameover_char = C_r; 19: get_gameover_char = C_E; 20:
get_gameover_char = C_SPC;
            21: get_gameover_char = (score / 10) % 10;
            22: get_gameover_char = (score % 10);
            23: get_gameover_char = C_SPC; 24: get_gameover_char = C_SPC; 25:
get_gameover_char = C_SPC;

            26: get_gameover_char = C_P; 27: get_gameover_char = C_r; 28:
get_gameover_char = C_E;

            29: get_gameover_char = C_S; 30: get_gameover_char = C_S; 31:
get_gameover_char = C_SPC;

            32: get_gameover_char = C_U; 33: get_gameover_char = C_SPC; 34:
get_gameover_char = C_t;

            35: get_gameover_char = C_o; 36: get_gameover_char = C_SPC; 37:
get_gameover_char = C_M;

            38: get_gameover_char = C_E; 39: get_gameover_char = C_n; 40:
get_gameover_char = C_U;

            default: get_gameover_char = C_SPC;

         endcase

      end

   endfunction



   // NEW RECORD: "A nEU LEgEnd RIsES   SCorE XX   PrESS U TO MEnU      " (Length 52)

   function [5:0] get_newrecord_char;

      input [6:0] pos;

      begin

          case(pos % 52)

            0: get_newrecord_char = C_A; 1: get_newrecord_char = C_SPC;

            2: get_newrecord_char = C_n; 3: get_newrecord_char = C_E; 4: get_newrecord_char
= C_U;

            5: get_newrecord_char = C_SPC; 6: get_newrecord_char = C_L; 7:
get_newrecord_char = C_E;

            8: get_newrecord_char = C_G; 9: get_newrecord_char = C_E; 10:
get_newrecord_char = C_n;
```

```
            11: get_newrecord_char = C_d; 12: get_newrecord_char = C_SPC; 13: get_newrecord_char = C_r;

            14: get_newrecord_char = C_I; 15: get_newrecord_char = C_S; 16: get_newrecord_char = C_E;

            17: get_newrecord_char = C_S; 18: get_newrecord_char = C_SPC; 19: get_newrecord_char = C_SPC;



            20: get_newrecord_char = C_S; 21: get_newrecord_char = C_C; 22: get_newrecord_char = C_o;

            23: get_newrecord_char = C_r; 24: get_newrecord_char = C_E; 25: get_newrecord_char = C_SPC;

            26: get_newrecord_char = (score / 10) % 10;

            27: get_newrecord_char = (score % 10);



            28: get_newrecord_char = C_SPC; 29: get_newrecord_char = C_SPC; 30: get_newrecord_char = C_SPC;

            31: get_newrecord_char = C_P; 32: get_newrecord_char = C_r; 33: get_newrecord_char = C_E;

            34: get_newrecord_char = C_S; 35: get_newrecord_char = C_S; 36: get_newrecord_char = C_SPC;

            37: get_newrecord_char = C_U; 38: get_newrecord_char = C_SPC; 39: get_newrecord_char = C_t;

            40: get_newrecord_char = C_o; 41: get_newrecord_char = C_SPC; 42: get_newrecord_char = C_M;

            43: get_newrecord_char = C_E; 44: get_newrecord_char = C_n; 45: get_newrecord_char = C_U;

            default: get_newrecord_char = C_SPC;

        endcase

    end
    endfunction


    // DECODER

    always @(*) begin

        case(char_index)

            C_0: seg = 7'b1000000; C_1: seg = 7'b1111001; C_2: seg = 7'b0100100; C_3: seg =
7'b0110000;

            C_4: seg = 7'b0011001; C_5: seg = 7'b0010010; C_6: seg = 7'b0000010; C_7: seg =
7'b1111000;

            C_8: seg = 7'b0000000; C_9: seg = 7'b0010000;

            C_L: seg = 7'b1000111; C_r: seg = 7'b0101111; C_U: seg = 7'b1000001; C_d: seg =
7'b0100001;

            C_SPC: seg = 7'b1111111; C_DASH: seg = 7'b0111111;

            C_A: seg = 7'b0001000; C_b: seg = 7'b0000011; C_C: seg = 7'b1000110; C_E: seg =
7'b0000110;

            C_F: seg = 7'b0001110; C_G: seg = 7'b0000010; C_H: seg = 7'b0001001; C_J: seg =
7'b1110001;

            C_n: seg = 7'b0101011; C_o: seg = 7'b0100011; C_P: seg = 7'b0001100; C_S: seg =
7'b0010010;

            C_t: seg = 7'b0000111; C_y: seg = 7'b0010001; C_I: seg = 7'b1111001;

            C_M: seg = 7'b0101010; // M looks kinda like n with extra line? or two n's? Using
pseudo-M

            default: seg = 7'b1111111;

        endcase

    end


endmodule
```

Figures and graphs necessary to describe the project, including simulation results.



SIMULATION - Behavioral Simulation - Functional - sim_1 - Final_Project_tb

Name	Value
clk	1
btnC	0
btnR	0
btnL	0
btnU	0
btnD	0
led[15:0]	0116
seg[6:0]	10
[6]	0
[5]	0
[4]	1
[3]	0
[2]	0
[1]	0
[0]	0
an[3:0]	b
[3]	1
[2]	0
[1]	1
[0]	1

Figure 1: Test bench of my final project

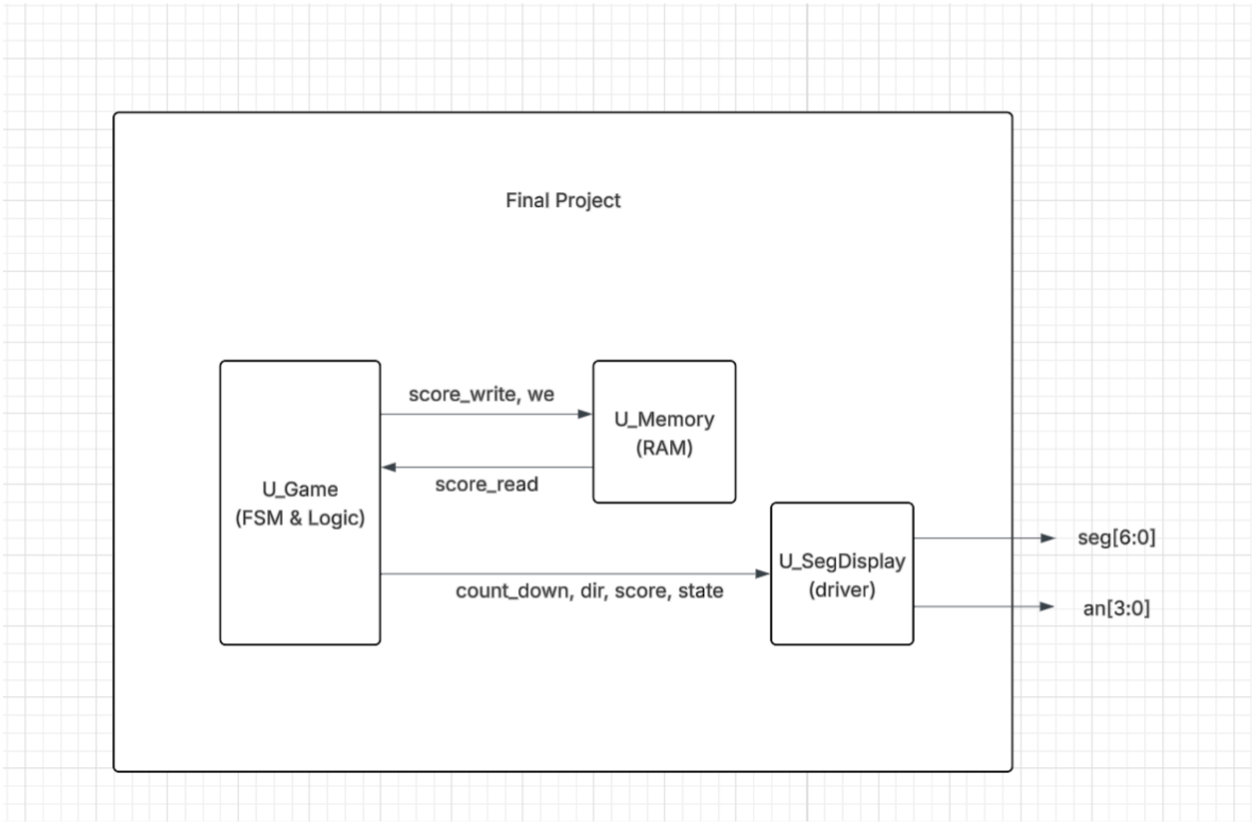


Figure 2: Diagram of my final project

A conclusion statement. As part of the conclusion statement, the level of functionality should be discussed.

My first project failed but my other project was a complete success able to complete every aspect of the project as described in the project descriptions section. I was able to learn how to get scrolling text on the 7 segment display. I also learned how to make a game using tons of different states allowing me to get user input, display a timer and display a score. Finally I learned resilience. I tried as long as I could to get my original project to work but when I had to cut ties I did to ensure that I would have enough time to make a new project. This taught me that even when things don't work out to press on until something does.