
# ELEX 7660: Digital System Design

## *Project Report – Spinning Display*

Brayden Aimar

Adam Wells

# Table of Contents

1 Introduction ..... 3
2 Procedure ..... 3
3 Circuit Design ..... 4
3.1 LED Display ..... 4
3.2 Hall Effect Sensor ..... 5
3.3 Power Supply ..... 5
4 Module Design ..... 6
4.1 Display Decode Module ..... 6
4.2 Scan Position Module ..... 7
4.3 Multiplexing Module ..... 7
5 Conclusion ..... 7
6 References ..... 8
7 Appendix: Design Drawings ..... 9
8 Appendix: SystemVerilog Code ..... 18
8.1 Top-Level Module ..... 18
8.2 Display Update Module ..... 20
8.3 Decode Character Module ..... 23
8.4 Scan Position Module ..... 28
8.5 Multiplexer Module ..... 30
9 Appendix: SystemVerilog Testbenches ..... 31
9.1 Top-Level Testbench ..... 31
9.2 Display Update Testbench ..... 32
9.3 Scan Position Testbench ..... 33
10 Appendix: Pin Assignments ..... 34

# Table of Figures

Figure 1: LED Display Circuit ..... 4
Figure 2: LED Display Internal Connections [1] ..... 4
Figure 3: Hall Effect Sensor Circuit ..... 5
Figure 4: FPGA Power Supply Circuit ..... 5
Figure 5: Decode String Simulation Waveform [3] ..... 6
Figure 6: Scan Position Simulation Waveform [3] ..... 7
Figure 7: Top-Level and Multiplexing Simulation Waveform [3] ..... 7

# 1 Introduction

The project we worked on this past semester was a spinning display. With spinning displays in general, a phenomenon known as Persistence of Vision is employed. Persistence of Vision is an optical illusion that encompasses a multitude of discrete images which then form a single image in the brain. Our design involved many different hardware applications separate from the use of the FPGA. We 3D printed a bowl like housing with a lid. Attached to the housing was the arm which held the LED display. Inside the housing was a hall effect sensor as well which was used to locate the arm at any given moment in the code.

# 2 Procedure

Our project was basically to spin an LED. We had to first think about how we could spin an LED while also spinning components involved in switching LED's on and off, i.e. FPGA, hall effect sensor. Using Fusion 360, we went through multiple different designs of the housing. The housing started being a rectangular shape, but later we re-designed it to be a bowl shape; this made sense since it would be more aerodynamic, therefore causing less vibrations and noise as the motor spun the display. One very challenging aspect we came across was how we were going to run power to the FPGA inside the housing. One idea we had was to use a slip ring where we could run power up from the bottom of the housing. Unfortunately, such an idea was too costly. With more thought and engineering, we decided to use two bearings which would sit atop the motor. Inside the bearings we 3D printed a cylinder with four holes on the top, and from inside the housing we slotted four legs which would fit perfectly inside the four holes inside the bearings. This way, as the motor spun inside the bearings, the housing would spin as well. With this design, we passed power through the bearings by supplying the outside if the top bearing with +5V and the bottom bearing with ground. Then, attaching a wire to the inside of the bearing we could power the FPGA because the inside of the bearing spun as the motor spun, and through this the wire spun with it.

# 3 Circuit Design

## 3.1 LED Display

The LED display has 48 LED's internally connected for multiplexing to a total of 25 pins, as seen in Figure 2. We decided to wire the display pins to the FPGA in the way that would reduce the current required from each pin.

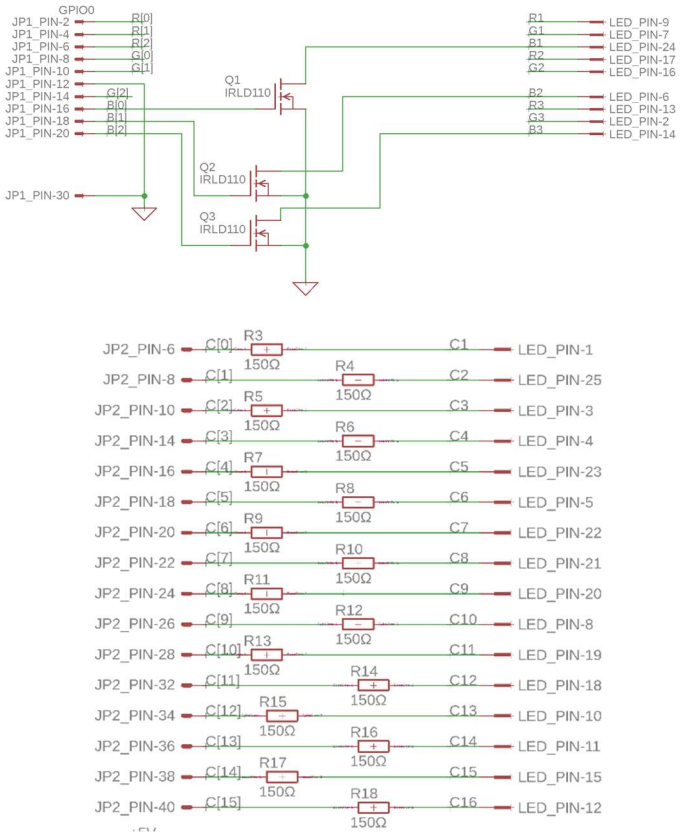



*Figure 1: LED Display Circuit*

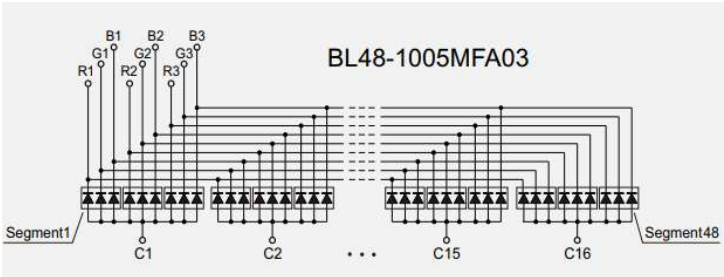



*Figure 2: LED Display Internal Connections [1]*

## 3.2 Hall Effect Sensor

The hall effect sensor requires a minimum supply voltage of 2.5V. [2] We powered the chip with 5V and pulled the open collector output to 3.3V to interface it with the FPGA GPIO pins.

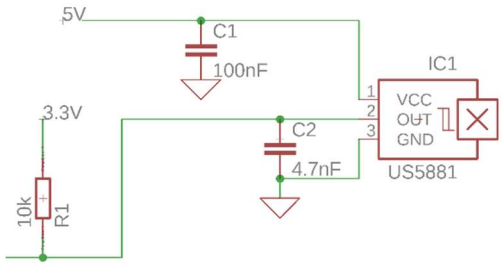



*Figure 3: Hall Effect Sensor Circuit*

## 3.3 Power Supply

To supply the FPGA with power, we passed 5V power through a set of bearings to the spinning unit. This power was then filtered with a large capacitor to remove noise introduced by the bearings.

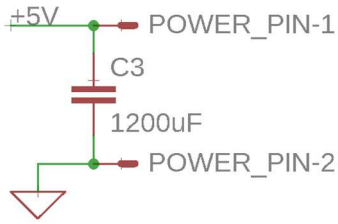



*Figure 4: FPGA Power Supply Circuit*

# 4 Module Design

Our System Verilog code consisted of a single top-level module that calls three sub modules that perform tasks such as, display string decoding, display timing, and display multiplexing.

## 4.1 Display Decode Module

The display decode module receives a string, parses through it, and sends each character to the decode character module and then adds each character data to the top-level display data.

We got all our code working except the string decoding module. The module its self simulates correctly, however, an issue in the timing of communications between the top-level module and the decode module leads to no characters being decoded in practice. To get the display working we made a $48 \times 128$ by hand and assigned it directly to *disp_data* to bypass the use of the decode display module.

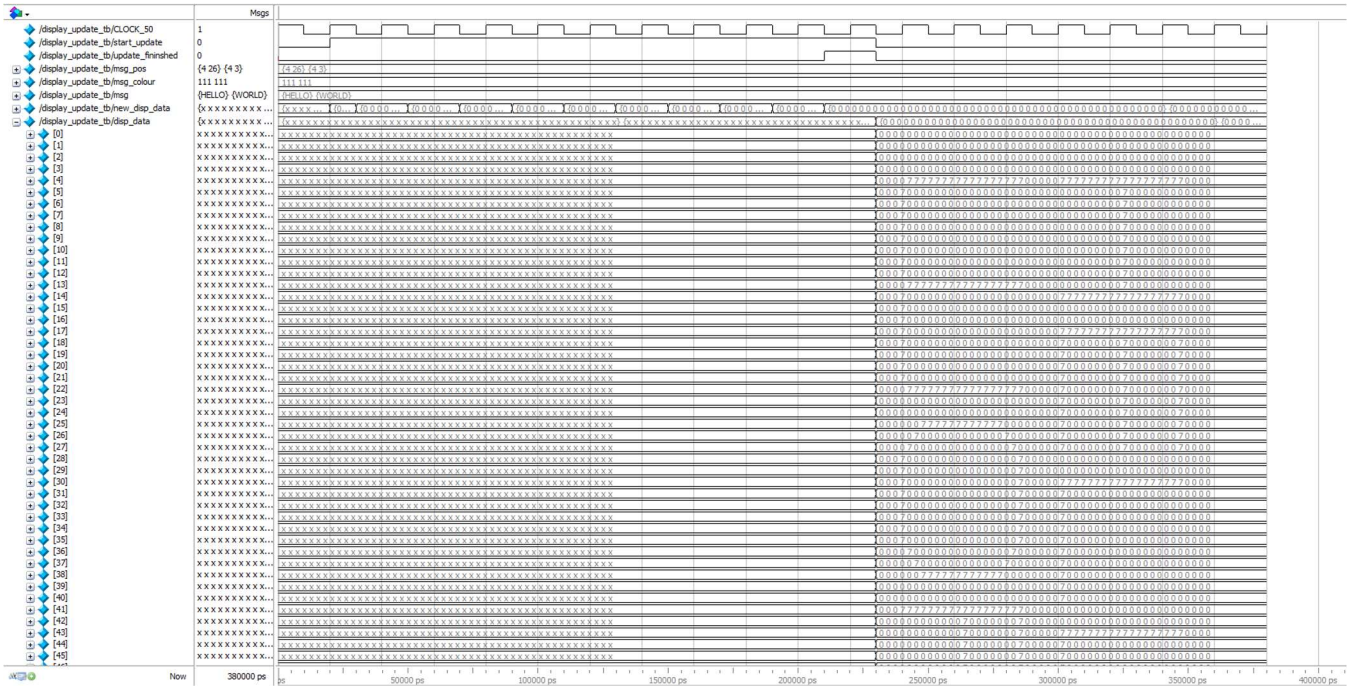



*Figure 5: Decode String Simulation Waveform [3]*

In the simulation of the display decoding module, after the *start_update* signal goes high, the values within *new_disp_data* are being altered as each character of the string is decoded, as seen in Figure 5. Once the decode is complete, the new data gets pushed to the global *disp_data* where it is then read by the multiplexing modules and written to the display.

## 4.2 Scan Position Module

The scan position module takes the hall effect sensor to produce an estimate of where the display arm is at any given point in time. In Figure 6, *bit_period*, the time between each vertical scan, starts at a reasonable guess. This guessed value is then changed based on wether or not the display scanned through too quickly or too slowly.

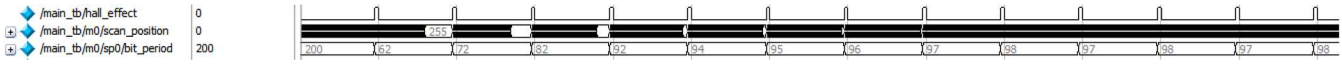



*Figure 6: Scan Position Simulation Waveform [3]*

## 4.3 Multiplexing Module

The multiplexing module controls the on/off state of each LED on the display based on *disp_data* at the current *scan_position*. In Figure 7, the multiplexing module is controlling the values of *B[0:2]* and *C[0:15]* to multiplex the string “HELLO WORLD” to the display.

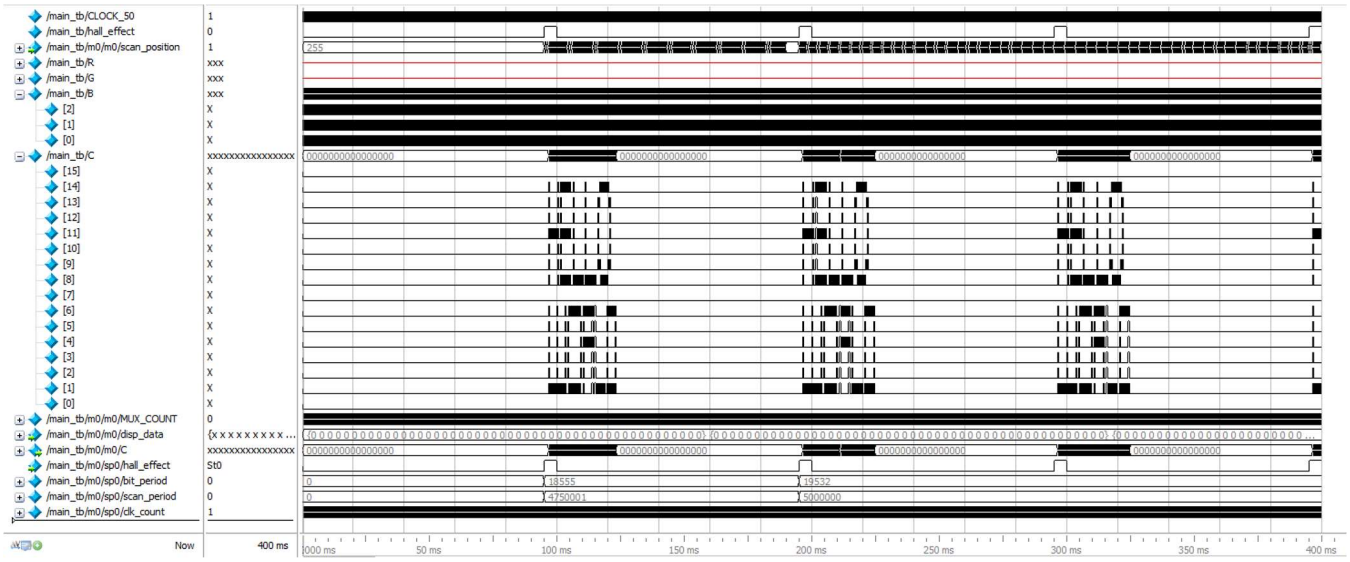



*Figure 7: Top-Level and Multiplexing Simulation Waveform [3]*

# 5 Conclusion

Overall, this project took 50-60 hours to complete. Over the course of this project we learned a lot. Specifically, Verilog and design. A large takeaway which will help us in our Capstone project next year is project management. Projects can go wrong at anytime, including when its time to present. Having backup hardware is always a good idea.

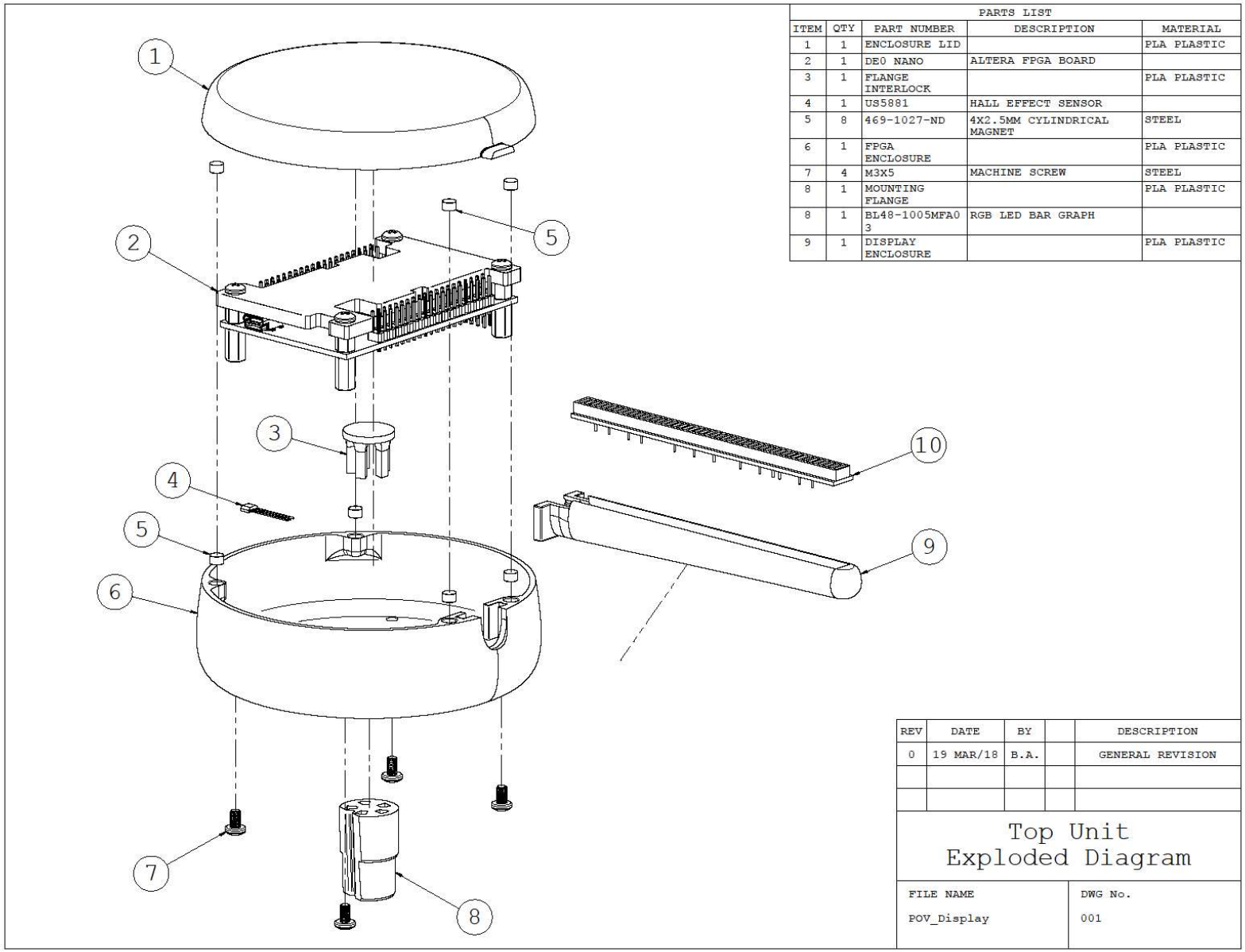
# 6 References

[1] Barmeter Electronics Inc, *BL48-1005MFX03 Datasheet.*

[2] Melexis, *US5881 Datasheet.*

[3] B. Aimar and A. Wells, *ModelSim Simulations.*

# 7 Appendix: Design Drawings



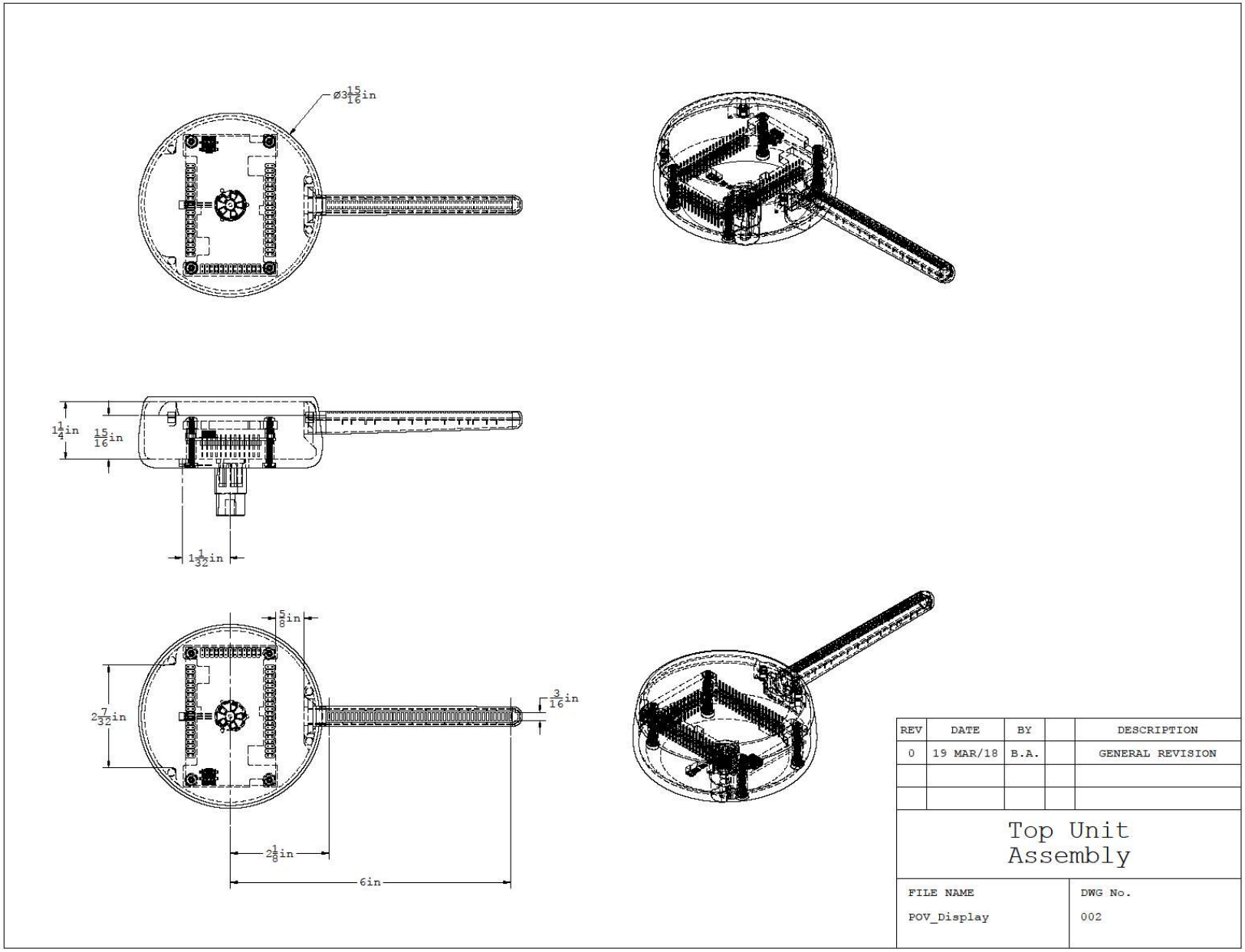

PARTS LIST

| ITEM | QTY | PART NUMBER | DESCRIPTION | MATERIAL |
|---|---|---|---|---|
| 1 | 1 | ENCLOSURE LID | | PLA PLASTIC |
| 2 | 1 | DE0 NANO | ALTERA FPGA BOARD | |
| 3 | 1 | FLANGE INTERLOCK | | PLA PLASTIC |
| 4 | 1 | US5881 | HALL EFFECT SENSOR | |
| 5 | 8 | 469-1027-ND | 4X2.5MM CYLINDRICAL MAGNET | STEEL |
| 6 | 1 | FPGA ENCLOSURE | | PLA PLASTIC |
| 7 | 4 | M3X5 | MACHINE SCREW | STEEL |
| 8 | 1 | MOUNTING FLANGE | | PLA PLASTIC |
| 8 | 1 | BL48-1005MFA03 | RGB LED BAR GRAPH | |
| 9 | 1 | DISPLAY ENCLOSURE | | PLA PLASTIC |

| REV | DATE | BY | | DESCRIPTION |
|---|---|---|---|---|
| 0 | 19 MAR/18 | B.A. | | GENERAL REVISION |
| | | | | |
| | | | | |

Top Unit
Exploded Diagram

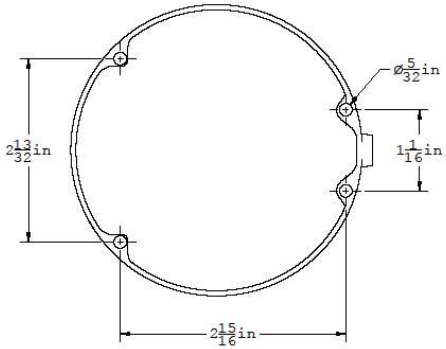
| FILE NAME | DWG No. |
|---|---|
| POV_Display | 001 |

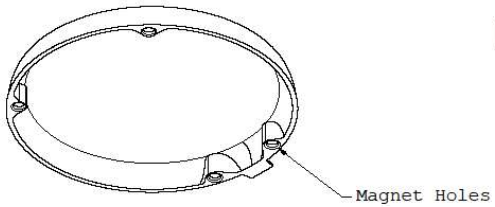


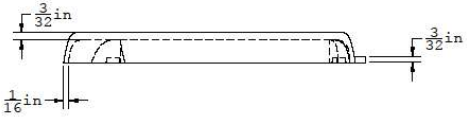

| REV | DATE | BY | | DESCRIPTION |
|---|---|---|---|---|
| 0 | 19 MAR/18 | B.A. | | GENERAL REVISION |
| | | | | |
| | | | | |

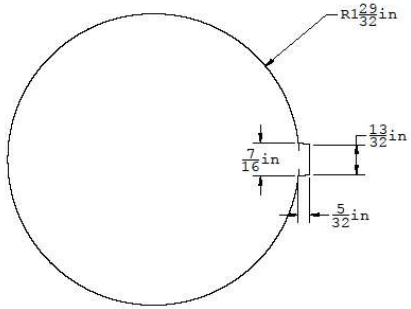
Top Unit Assembly

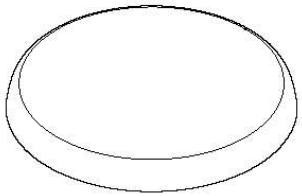
| FILE NAME | DWG No. |
|---|---|
| POV_Display | 002 |



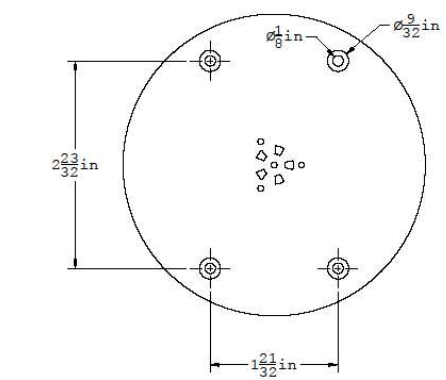



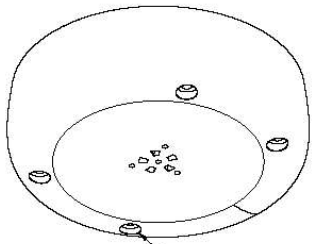



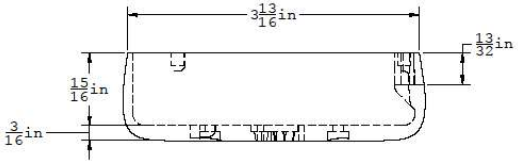
| REV | DATE | BY | | DESCRIPTION |
|---|---|---|---|---|
| 0 | 19 MAR/18 | B.A. | | GENERAL REVISION |
| | | | | |
| | | | | |

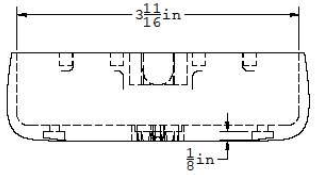
TOP Unit
FPGA Enclosure Lid

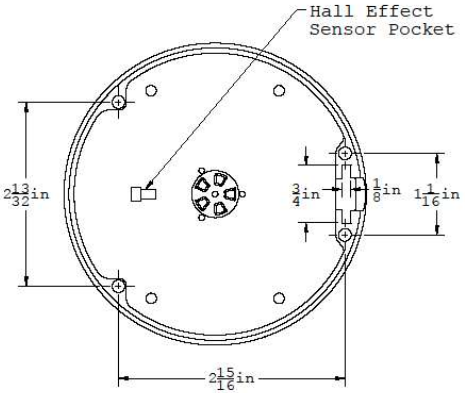
| FILE NAME | DWG No. |
|---|---|
| POV_Display | 003 |

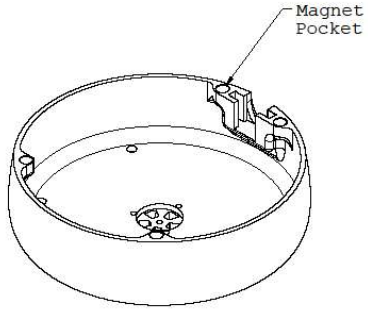


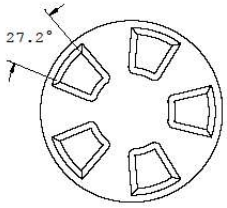

NOTES:
1) FPGA Enclosure is to be 3D printed right-side up.

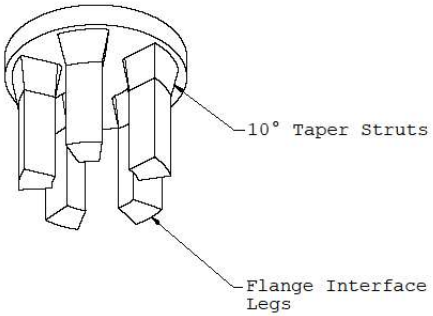
2) Magnets will be intalled with cyanoacrylate adheasive.

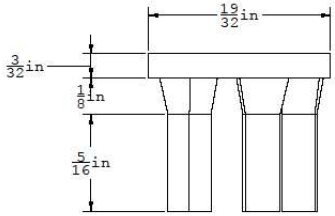
| REV | DATE | BY | | DESCRIPTION |
|---|---|---|---|---|
| 0 | 19 MAR/18 | B.A. | | GENERAL REVISION |
| | | | | |
| | | | | |

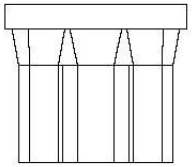
Top Unit
FPGA Enclosure

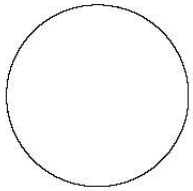
| FILE NAME | DWG No. |
|---|---|
| POV_Display | 004 |

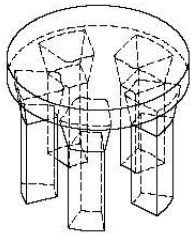






NOTES:
1) This securing stake is to be 3D printed with the legs facing upwards.

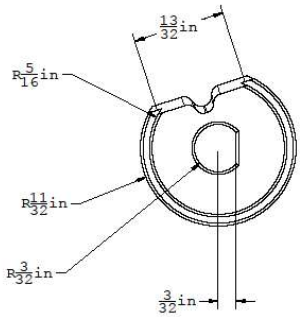
2) This securing stake is to be installed with a liberal application of JB Weld (10min cure time) two-part epoxy on all surfaces except top surface and center hole.

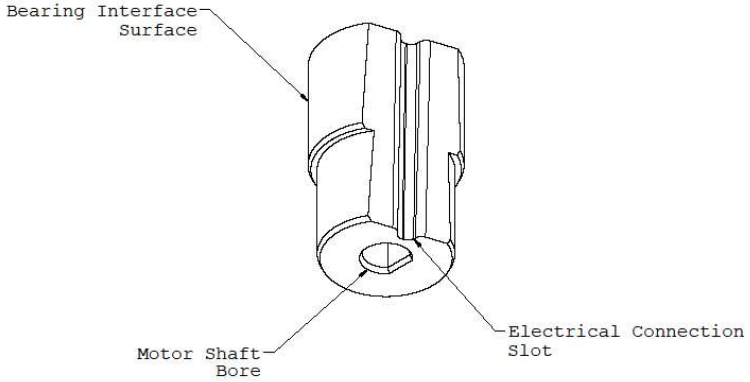
3) This securing stake must be visually instected for surface cracks and x-rayed to detect sub-surface material defects. Any surface cracks or material defects shall prevent the part from being installed in the project.

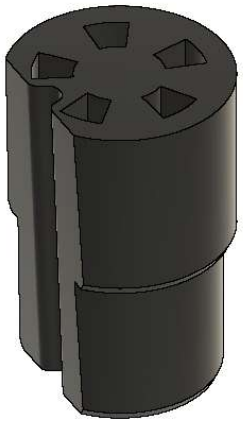
| REV | DATE | BY | | DESCRIPTION |
|---|---|---|---|---|
| 0 | 19 MAR/18 | B.A. | | GENERAL REVISION |
| | | | | |
| | | | | |

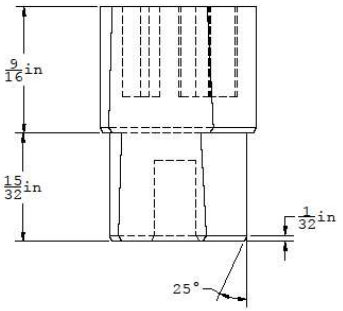
Top Unit
Flange Interlock

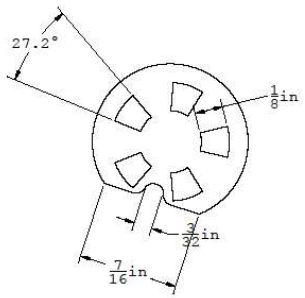
| FILE NAME | DWG No. |
|---|---|
| POV_Display | 005 |

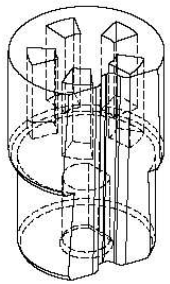




NOTES:
1) This mounting flange is to be 3D printed with the motor shaft bore facing up and finish quality and print accuracy settings at maximum.

2) The ID and OD measurements must be within ±0.005" to ensure proper friction fits to both the motor shaft and bearing bore respectively.

3) Both the ID and OD rims have been chamfered to ease the installation of both the motor shaft and bearing into a friction fit seat.

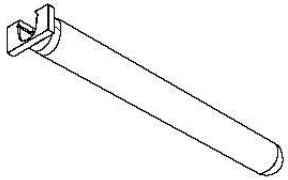
4) Differetial heating to install motor shaft and/or bearing can not be used as doing so would compromise the integrity of this mounting flange.

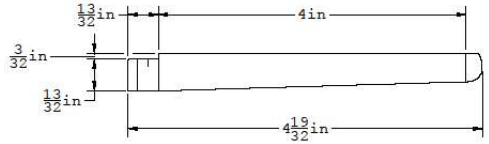
5) Both the motor shaft and bearing should be installed to this mounting flange before it is permanently affixed to the FPGA enclosure with the securing stake and two-part epoxy

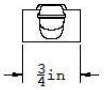
| REV | DATE | BY | | DESCRIPTION |
|---|---|---|---|---|
| 0 | 19 MAR/18 | B.A. | | GENERAL REVISION |
| | | | | |
| | | | | |

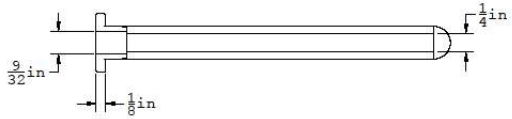
Top Unit
Mounting Flange

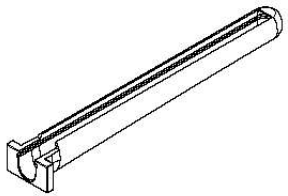
| FILE NAME | DWG No. |
|---|---|
| POV_Display | 006 |



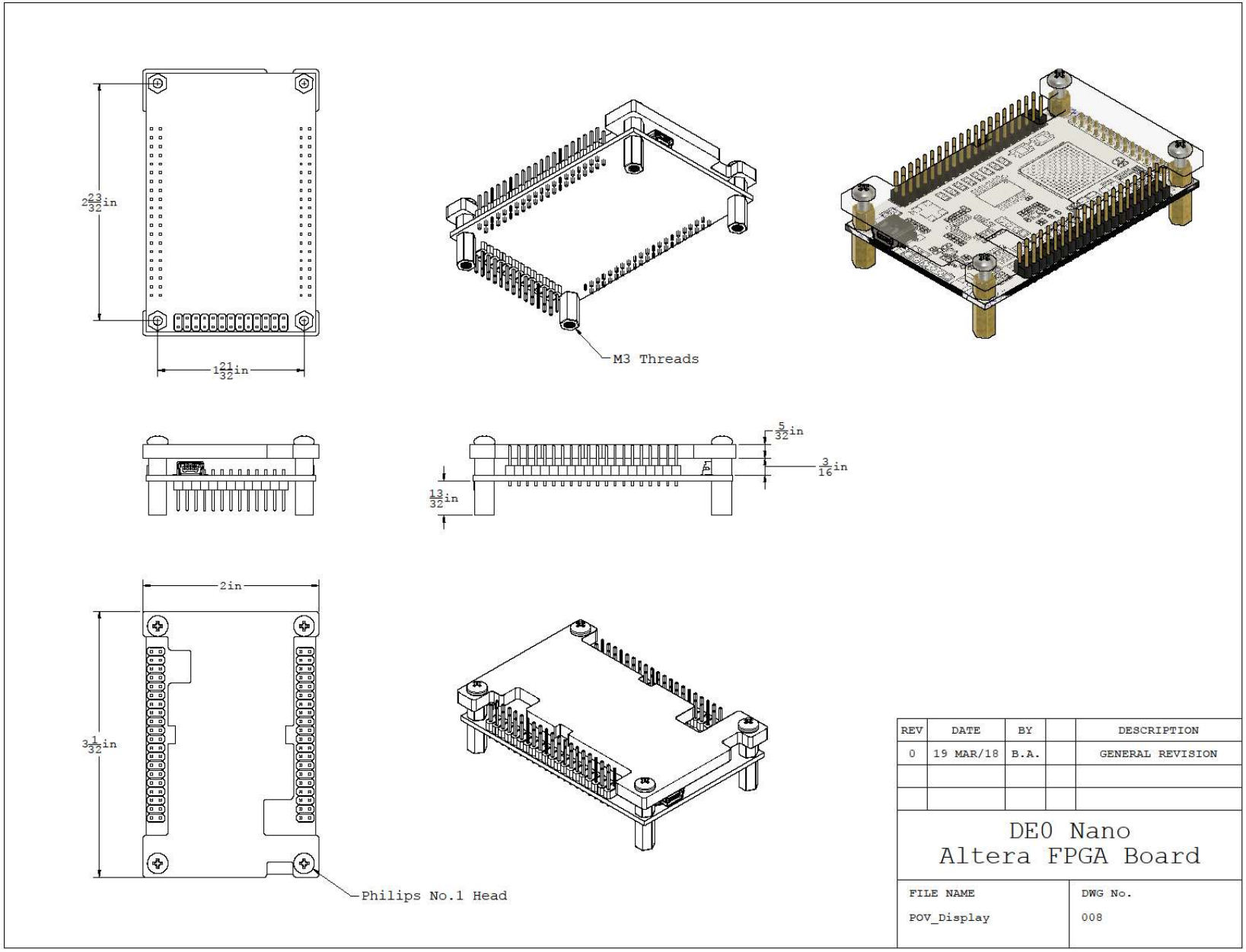



NOTES:
1) This display enclosure is to be 3D printed with the display pocket facing up and using raft supports.

| REV | DATE | BY | | DESCRIPTION |
|---|---|---|---|---|
| 0 | 19 MAR/18 | B.A. | | GENERAL REVISION |
| | | | | |
| | | | | |

Top Unit
Display Enclosure

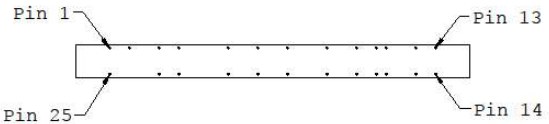
| FILE NAME | DWG No. |
|---|---|
| POV_Display | 007 |

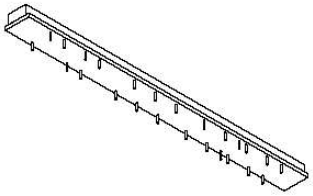


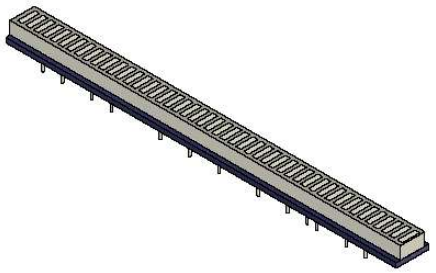

| REV | DATE | BY | | DESCRIPTION |
|---|---|---|---|---|
| 0 | 19 MAR/18 | B.A. | | GENERAL REVISION |
| | | | | |
| | | | | |

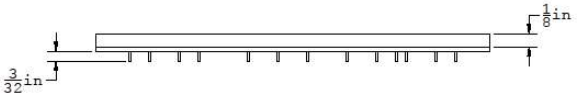
DE0 Nano
Altera FPGA Board

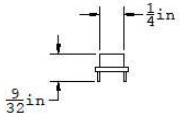
| FILE NAME | DWG No. |
|---|---|
| POV_Display | 008 |

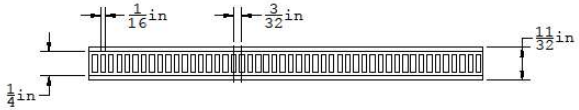


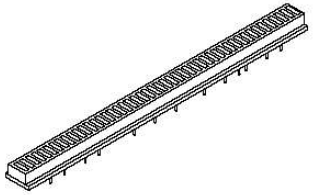







| REV | DATE | BY | | DESCRIPTION |
|---|---|---|---|---|
| 0 | 19 MAR/18 | B.A. | | GENERAL REVISION |
| | | | | |
| | | | | |

BL48-1005MFA03
RGB LED Bar Graph

| FILE NAME | DWG No. |
|---|---|
| POV_Display | 009 |

# 8 Appendix: SystemVerilog Code

## 8.1 Top-Level Module

```
// main.sv - The top level verilog program for the POV display.
// Authors: Adam Wells, Brayden Aimar

module main (
        input logic CLOCK_50,
        input logic hall_effect,
        input logic [1:0] KEY,
        output logic [7:0] LED = '0,
        output logic [2:0] R, G, B,
        output logic [15:0] C
    );

    // Hardware-defined parameters
    localparam  X_AXIS = 0;
    localparam  Y_AXIS = 1;
    localparam  DISP_AXES = 2;      // Display has two (2) axes (horizontal and vertical).
    localparam  DISP_HEIGHT = 48;   // Vertical resolution of the display [pixels].
    localparam  DISP_COLORS = 3;    // Number of colors suported by the display.
    localparam  DISP_ANODES = 16;   // Number of common anode connections on the display.
    localparam  DISP_CATHODES = 3;  // Number of cathodes per color of the display.
    localparam  DISP_LINES = 2;     // Number of lines to show on display.
    localparam  LINE_WIDTH = 5;     // Number of characters on a line.

    // User-defined parameters
    localparam  DISP_WIDTH = 128;   // Horizontal resolution of the display [pixels].
    localparam  CHAR_SPACING = 2;   // Spacing between characters [pixels].
    localparam  SCAN_OFFSET = 0;    // Sets rotational offset of display with respect to
magnet and hall effect sensor positions [pixels].
    logic [DISP_COLORS-1:0] disp_data [DISP_WIDTH][DISP_HEIGHT] = '{ DISP_WIDTH{'{
DISP_HEIGHT{3'b0} }} };
    logic [DISP_COLORS-1:0] new_disp_data [DISP_WIDTH][DISP_HEIGHT];
    int scan_position;

    logic [LINE_WIDTH-1:0][7:0] msg [DISP_LINES] = '{ { 8'h48, 8'h45, 8'h4C, 8'h4C, 8'h4F },
{ 8'h57, 8'h4F, 8'h52, 8'h4C, 8'h44 } };
    logic [DISP_COLORS-1:0] msg_colour [DISP_LINES] = '{ DISP_LINES{3'b111} };
    int msg_pos [DISP_LINES][DISP_AXES] = '{ '{ 0, 26 }, '{ 0, 3 } };
    logic start_update = 0;
    logic update_fininshed;
    logic prev_update_finished = 0;

    display_update #(X_AXIS, Y_AXIS, DISP_AXES, DISP_WIDTH, DISP_HEIGHT, DISP_COLORS,
DISP_LINES, LINE_WIDTH, CHAR_SPACING) du0 (
        .CLOCK_50,
        .disp_data,
        .new_disp_data,
        .msg,
        .msg_colour,
        .msg_pos,
        .start_update,
        .update_fininshed
    );

    initial begin

        start_update = 1;

    end

    always @(posedge CLOCK_50) begin

        LED[0] = hall_effect;
```

```
        // Display update finished
        if (update_fininshed && !prev_update_finished)
            disp_data = new_disp_data;

        prev_update_finished = update_fininshed;

    end

    scan_position #(DISP_WIDTH, SCAN_OFFSET) sp0(
        .CLOCK_50,
        .hall_effect,
        .scan_position
    );

    multiplexer #(DISP_WIDTH, DISP_HEIGHT, DISP_COLORS, DISP_ANODES, DISP_CATHODES) m0(
        .CLOCK_50,
        .disp_data,
        .scan_position,
        .R,
        .G,
        .B,
        .C
    );

endmodule
```

## 8.2 Display Update Module

```
// display_update.sv - Updates the display data for a given msg.
// Authors: Adam Wells, Brayden Aimar

module display_update #(
        parameter  X_AXIS = 0,
        parameter  Y_AXIS = 1,
        parameter  DISP_AXES = 2,
        parameter  DISP_WIDTH = 256,
        parameter  DISP_HEIGHT = 48,
        parameter  DISP_COLORS = 3,
        parameter  DISP_LINES = 2,
        parameter  LINE_WIDTH = 10,
        parameter  CHAR_SPACING = 2
    ) (
        input logic CLOCK_50,
        input logic [DISP_COLORS-1:0] disp_data [DISP_WIDTH][DISP_HEIGHT],
        output logic [DISP_COLORS-1:0] new_disp_data [DISP_WIDTH][DISP_HEIGHT],
        input logic [LINE_WIDTH-1:0][7:0] msg [DISP_LINES],
        input logic [DISP_COLORS-1:0] msg_colour [DISP_LINES],
        input int msg_pos[DISP_LINES][DISP_AXES],
        input logic start_update,
        output logic update_fininshed = 0
    );

    localparam  update_delay = 1000;
    logic update_clk = '1;
    int update_count = 0;

    logic [DISP_COLORS-1:0] new_char_disp_data [DISP_WIDTH][DISP_HEIGHT];
    logic [7:0] disp_char = '0;
    logic [DISP_COLORS-1:0] disp_colour;
    int start_pos [DISP_AXES];
    int end_pos [DISP_AXES];
    logic start_decode = 0;
    logic prev_start_update = 0;
    logic decode_finished;
    logic prev_decode_finished = 0;

    localparam  MAX_CHAR_HEIGHT = 18;
    localparam  MAX_CHAR_WIDTH = 19;
    logic [7:0] char_width;
    logic char_data [MAX_CHAR_WIDTH][MAX_CHAR_HEIGHT];

    decode_char #(X_AXIS, Y_AXIS, DISP_AXES, DISP_WIDTH, DISP_HEIGHT, DISP_COLORS) dc0 (
        .new_disp_data,
        .new_char_disp_data,
        .disp_char,
        .disp_colour,
        .start_pos,
        .end_pos,
        .start_decode,
        .decode_finished
    );

    int line_index = 0;
    int char_index = 0;

    always @(posedge update_clk) begin

        if (start_decode) begin

            start_decode = 0;
            new_disp_data = new_char_disp_data;

            // Add space between characters
```

```
            start_pos[X_AXIS] = end_pos[X_AXIS] + CHAR_SPACING;
            start_pos[Y_AXIS] = end_pos[Y_AXIS];

        end
        else begin

            if (start_update && !prev_start_update) begin

                update_fininshed = 0;
                // Clear existing display data
                new_disp_data = '{ DISP_WIDTH{'{ DISP_HEIGHT{3'b0} }} };

                line_index = 0;
                char_index = LINE_WIDTH - 1;

                disp_colour = msg_colour[line_index];
                start_pos = msg_pos[line_index];

                disp_char = msg[line_index][char_index];

                start_decode = 1;

            end

            // Decode character finished
            if (decode_finished && !prev_decode_finished) begin

                // Display update finished
                if (line_index == DISP_LINES - 1 && !char_index) begin

                    $display("Display update finished.");
                    update_fininshed = 1;

                end
                else begin  // Continue to update display

                    // At end of display line
                    if (!char_index) begin

                        line_index = line_index + 1;
                        char_index = LINE_WIDTH - 1;

                        disp_colour = msg_colour[line_index];
                        start_pos = msg_pos[line_index];

                    end
                    else begin

                        char_index = char_index - 1;

                    end

                    disp_char = msg[line_index][char_index];
                    start_decode = 1;

                end

            end

            prev_start_update = start_update;
            prev_decode_finished = decode_finished;

        end

    end

    always @(posedge CLOCK_50) begin
```

```
        if (update_count >= update_delay) begin

            update_clk = ~update_clk;
            update_count = 0;

        end
        else
            update_count = update_count + 1;

    end

endmodule
```

## 8.3 Decode Character Module

```
// decode_char.sv - Resolves individual characters into display pixels.
// Authors: Adam Wells, Brayden Aimar

module decode_char #(
        parameter  X_AXIS = 0,
        parameter  Y_AXIS = 1,
        parameter  DISP_AXES = 2,
        parameter  DISP_WIDTH = 256,
        parameter  DISP_HEIGHT = 48,
        parameter  DISP_COLORS = 3
    ) (
        input logic [DISP_COLORS-1:0] new_disp_data [DISP_WIDTH][DISP_HEIGHT],
        output logic [DISP_COLORS-1:0] new_char_disp_data [DISP_WIDTH][DISP_HEIGHT],
        input logic [7:0] disp_char,
        input logic [DISP_COLORS-1:0] disp_colour,
        input int start_pos [DISP_AXES],
        output int end_pos [DISP_AXES],
        input logic start_decode,
        output logic decode_finished = 0
    );

    localparam  MAX_CHAR_HEIGHT = 18;
    localparam  MAX_CHAR_WIDTH = 19;
    logic [7:0] char_width;
    logic char_data [MAX_CHAR_WIDTH][MAX_CHAR_HEIGHT];

    always @(posedge start_decode) begin

        decode_finished = 0;

        case (disp_char)
            8'h20: begin  // SPACE
                $display("Decoding a space.");
                char_width = 4;
                char_data = '{
                   // 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17  - Row
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 0-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 1-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 2-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 3-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 4-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 5-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 6-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 7-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 8-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 9-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 10-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 11-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 12-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 13-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 14-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 15-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 16-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 17-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }   // 18-Column
                };
            end
            8'h31: begin  // 1
                $display("Decoding a 1 (one).");
                char_width = 7;
                char_data = '{
                    // 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17  - Row
                    '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 },  // 0-Column
                    '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 },  // 1-Column
                    '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 },  // 2-Column
```

```
            '{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0 },  // 3-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 4-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 5-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 6-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 7-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 8-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 9-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 10-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 11-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 12-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 13-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 14-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 15-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 16-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 17-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }   // 18-Column
        };
    end
    8'h44: begin  // D
        $display("Decoding a D");
        char_width = 10;
        char_data = '{
            // 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17  - Row
            '{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },  // 0-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },  // 1-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },  // 2-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },  // 3-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },  // 4-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },  // 5-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },  // 6-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },  // 7-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },  // 8-Column
            '{ 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0 },  // 9-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 10-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 11-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 12-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 13-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 14-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 15-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 16-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 17-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }   // 18-Column
        };
        end
    8'h45: begin  // E
        $display("Decoding an E");
        char_width = 11;
        char_data = '{
            // 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17  - Row
            '{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },  // 0-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 },  // 1-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 },  // 2-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 },  // 3-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 },  // 4-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 },  // 5-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 },  // 6-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 },  // 7-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 },  // 8-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 },  // 9-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 },  // 10-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 11-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 12-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 13-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 14-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 15-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 16-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 17-Column
```

```
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }   // 18-Column
        };
        end
    8'h48: begin  // H
        $display("Decoding an H");
        char_width = 11;
        char_data = '{
            // 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17  - Row
            '{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },  // 0-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 },  // 1-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 },  // 2-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 },  // 3-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 },  // 4-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 },  // 5-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 },  // 6-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 },  // 7-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 },  // 8-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 },  // 9-Column
            '{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },  // 10-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 11-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 12-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 13-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 14-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 15-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 16-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 17-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }   // 18-Column
        };
    end
    8'h4C: begin  // L
        $display("Decoding an L");
        char_width = 11;
        char_data = '{
            // 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17  - Row
            '{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },  // 0-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 1-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 2-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 3-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 4-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 5-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 6-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 7-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 8-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 9-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 10-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 11-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 12-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 13-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 14-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 15-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 16-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 17-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }   // 18-Column
        };
    end
    8'h4F: begin  // O
        $display("Decoding an O");
        char_width = 14;
        char_data = '{
            // 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17  - Row
            '{ 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0 },  // 0-Column
            '{ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 },  // 1-Column
            '{ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0 },  // 2-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },  // 3-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },  // 4-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },  // 5-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },  // 6-Column
```

```
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },  // 7-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },  // 8-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },  // 9-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },  // 10-Column
            '{ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0 },  // 11-Column
            '{ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 },  // 12-Column
            '{ 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0 },  // 13-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 14-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 15-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 16-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 17-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }   // 18-Column
        };
    end
    8'h52: begin  // R
        $display("Decoding a R");
        char_width = 10;
        char_data = '{
            // 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17  - Row
            '{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },  // 0-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 },  // 1-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 },  // 2-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 },  // 3-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 },  // 4-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 },  // 5-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 },  // 6-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 },  // 7-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 },  // 8-Column
            '{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0 },  // 9-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 10-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 11-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 12-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 13-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 14-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 15-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 16-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 17-Column
            '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }   // 18-Column
        };
        end
    8'h57: begin  // W
        $display("Decoding a W");
        char_width = 19;
        char_data = '{
            // 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17  - Row
            '{ 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },  // 0-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 1-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 2-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 3-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 4-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 5-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 6-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 7-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 8-Column
            '{ 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },  // 9-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 10-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 11-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 12-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 13-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 14-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 15-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 16-Column
            '{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 17-Column
            '{ 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }   // 18-Column
        };
    end
    default: begin
```

```
                $display("Decoding default.");
                char_width = 14;
                char_data = '{
                    // 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17  - Row
                    '{ 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 0-Column
                    '{ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 1-Column
                    '{ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 2-Column
                    '{ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 3-Column
                    '{ 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0 },  // 4-Column
                    '{ 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0 },  // 5-Column
                    '{ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0 },  // 6-Column
                    '{ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0 },  // 7-Column
                    '{ 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0 },  // 8-Column
                    '{ 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0 },  // 9-Column
                    '{ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 10-Column
                    '{ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 11-Column
                    '{ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 12-Column
                    '{ 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 13-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 14-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 15-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 16-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 17-Column
                    '{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }   // 18-Column
                };
            end
        endcase

        new_char_disp_data = new_disp_data;

        // Add char_data to disp_data at start_potition.
        for (int x = 0; x < MAX_CHAR_WIDTH; x++) begin
            for (int y = 0; y < MAX_CHAR_HEIGHT; y++) begin
                new_char_disp_data[start_pos[X_AXIS] + x][start_pos[Y_AXIS] + y] =
(|disp_colour & char_data[x][y]) ? 3'b111 : 3'b000;
            end
        end

        end_pos[X_AXIS] = start_pos[X_AXIS] + char_width;
        end_pos[Y_AXIS] = start_pos[Y_AXIS];

       decode_finished = 1;

    end

endmodule
```

## 8.4 Scan Position Module

```
// scan_position.sv - Resolves the rotational position of display arm given hall effect
sensor signal.
// Authors: Adam Wells, Brayden Aimar

module scan_position #(
        parameter  DISP_WIDTH = 256,
        parameter  SCAN_OFFSET = 0
    ) (
        input logic CLOCK_50,
        input logic hall_effect,
        output int scan_position = 0
    );

    localparam  scan_delay = 1000;

    logic scan_clk = '1;
    int scan_count = 0;
    logic zero_scan_position = '0;
    int bit_clk = 0;
    int bit_period = 200;
    logic prev_hall_effect = '1;

    always @(posedge scan_clk) begin

        if (hall_effect && !prev_hall_effect) begin  // posedge hall_effect

            if (bit_clk < bit_period)
                bit_period = bit_period - (bit_period - (DISP_WIDTH - scan_position) > 0 ?
(DISP_WIDTH - scan_position) : 1 );

            else if (scan_position == DISP_WIDTH - 1)
                bit_period = bit_period + ((bit_clk - bit_period) > 4000 ? 10 : ((bit_clk -
bit_period) > 1000 ? 2 : 1));

            zero_scan_position = '1;

        end

        prev_hall_effect = hall_effect;

        if (zero_scan_position) begin

            scan_position = 0;
            bit_clk = 0;
            zero_scan_position = '0;

        end
        else begin

            bit_clk = bit_clk + 1;

            if (bit_clk >= bit_period && scan_position < DISP_WIDTH - 1) begin

                bit_clk = 0;
                scan_position = scan_position + 1;

            end

        end
    end

    always @(posedge CLOCK_50) begin

        if (scan_count >= scan_delay) begin
```

```
            scan_clk = ~scan_clk;
            scan_count = 0;

        end
        else
            scan_count = scan_count + 1;

    end

endmodule
```

## 8.5 Multiplexer Module

```
// multiplexer.sv - Multiplexes LED display arm based on display data and arm position.
// Authors: Adam Wells, Brayden Aimar

module multiplexer #(
        parameter  DISP_WIDTH = 256,
        parameter  DISP_HEIGHT = 48,
        parameter  DISP_COLORS = 3,
        parameter  DISP_CATHODES = 3,
        parameter  DISP_ANODES = 16
    ) (
        input logic CLOCK_50,
        input logic [DISP_COLORS-1:0] disp_data [DISP_WIDTH][DISP_HEIGHT],
        input int scan_position,
        output logic [2:0] R = 0,
        output logic [2:0] G = 0,
        output logic [2:0] B = 0,
        output logic [15:0] C = 0
    );

    localparam  mux_delay = 1000;
    localparam  MUX_STEPS = 3;

    logic MUX_CLK = '1;
    int mux_count = 0;
    int mux_state = 0;

    always @(posedge MUX_CLK) begin

        mux_state = (mux_state < MUX_STEPS - 1) ? mux_state + 1 : 0;

        R <= 3'b1 << mux_state;
        G <= 3'b1 << mux_state;
        B <= 3'b1 << mux_state;

        C[0] <= (disp_data[scan_position][0 * 3 + mux_state]) ? 1'b1 : 1'b0;
        C[1] <= (disp_data[scan_position][1 * 3 + mux_state]) ? 1'b1 : 1'b0;
        C[2] <= (disp_data[scan_position][2 * 3 + mux_state]) ? 1'b1 : 1'b0;
        C[3] <= (disp_data[scan_position][3 * 3 + mux_state]) ? 1'b1 : 1'b0;
        C[4] <= (disp_data[scan_position][4 * 3 + mux_state]) ? 1'b1 : 1'b0;
        C[5] <= (disp_data[scan_position][5 * 3 + mux_state]) ? 1'b1 : 1'b0;
        C[6] <= (disp_data[scan_position][6 * 3 + mux_state]) ? 1'b1 : 1'b0;
        C[7] <= (disp_data[scan_position][7 * 3 + mux_state]) ? 1'b1 : 1'b0;
        C[8] <= (disp_data[scan_position][8 * 3 + mux_state]) ? 1'b1 : 1'b0;
        C[9] <= (disp_data[scan_position][9 * 3 + mux_state]) ? 1'b1 : 1'b0;
        C[10] <= (disp_data[scan_position][10 * 3 + mux_state]) ? 1'b1 : 1'b0;
        C[11] <= (disp_data[scan_position][11 * 3 + mux_state]) ? 1'b1 : 1'b0;
        C[12] <= (disp_data[scan_position][12 * 3 + mux_state]) ? 1'b1 : 1'b0;
        C[13] <= (disp_data[scan_position][13 * 3 + mux_state]) ? 1'b1 : 1'b0;
        C[14] <= (disp_data[scan_position][14 * 3 + mux_state]) ? 1'b1 : 1'b0;
        C[15] <= (disp_data[scan_position][15 * 3 + mux_state]) ? 1'b1 : 1'b0;

    end

    always @(posedge CLOCK_50) begin;
        if (mux_count >= mux_delay) begin
            MUX_CLK = ~MUX_CLK;
            mux_count = 0;
        end
        else
            mux_count = mux_count + 1;
    end

endmodule
```

# 9 Appendix: SystemVerilog Testbenches

## 9.1 Top-Level Testbench

```
// main_tb.sv - Testbench for the main module.
// Authors: Adam Wells, Brayden Aimar

module main_tb;

    // Hardware-defined parameters
    localparam  DISP_CATHODES = 3;  // Number of colors suported by the display.
    localparam  DISP_ANODES = 16;   // Number of common anode connections on the display.

    // User-defined parameters
    localparam  TEST_ITERATIONS = 20;

    logic CLOCK_50 = '1;
    logic hall_effect = '1;
    logic [1:0] KEY = '0;
    logic [7:0] LED;
    logic [DISP_CATHODES-1:0] R, G, B;
    logic [DISP_ANODES-1:0] C;

    main m0 (
        .CLOCK_50,
        .hall_effect,
        .KEY,
        .LED,
        .R,
        .G,
        .B,
        .C
    );

    initial begin

        $display("120rpm Test");

        for (int i = 0; i < TEST_ITERATIONS; i++) begin  // 120rpm Test
            #475ms hall_effect = 0;
            #25ms hall_effect = 1;
            $display("Rotation #%d", i);
        end

        $stop;

    end

    always
        #10ns CLOCK_50 = ~CLOCK_50;  // 50MHz Clock

endmodule
```

## 9.2 Display Update Testbench

```
// decode_char_tb.sv - Testbench for the display_update module.
// Authors: Adam Wells, Brayden Aimar

module display_update_tb;

    // Hardware-defined parameters
    localparam  X_AXIS = 0;
    localparam  Y_AXIS = 1;
    localparam  DISP_AXES = 2,
    localparam  DISP_HEIGHT = 48;  // Vertical resolution of the display [pixels].
    localparam  CHAR_SPACING = 2;  // Spacing between characters [pixels].
    localparam  DISP_LINES = 2;

    // User-defined parameters
    localparam  DISP_WIDTH = 128;  // Horizontal resolution of the display [pixels].

    logic [DISP_COLORS-1:0] disp_data [DISP_WIDTH][DISP_HEIGHT];
    logic [DISP_COLORS-1:0] new_disp_data [DISP_WIDTH][DISP_HEIGHT];

    logic CLOCK_50 = '1;
    string msg [DISP_LINES] = '{ "HELLO", "WORLD" };
    logic [DISP_COLORS-1:0] msg_colour [DISP_LINES] = '{ DISP_LINES{3'b111} };
    int msg_pos [DISP_LINES][2] = '{ '{ 4, 26 }, '{ 4, 3 } };
    logic start_update = 0;
    logic update_fininshed = 0;

    display_update #(X_AXIS, Y_AXIS, DISP_AXES, DISP_WIDTH, DISP_HEIGHT, DISP_COLORS,
DISP_LINES, CHAR_SPACING) du0 (
        .disp_data,
        .new_disp_data,
        .msg,
        .msg_colour,
        .msg_pos,
        .start_update,
        .update_fininshed,
        .CLOCK_50
    );

    initial begin

        @(posedge CLOCK_50);

        start_update = 1;
        @(posedge update_fininshed);
        @(negedge CLOCK_50);

        start_update = 0;
        @(negedge update_fininshed);

        disp_data = new_disp_data;

        repeat(8) @(posedge CLOCK_50);
        $stop;

    end

    always
        #10ns CLOCK_50 = ~CLOCK_50;  // 50MHz Clock

endmodule
```

## 9.3 Scan Position Testbench

```
// scan_position_tb.sv - Testbench for the display_update module.
// Authors: Adam Wells, Brayden Aimar

module scan_position_tb;

    // Hardware-defined parameters
    localparam  DISP_WIDTH = 128;  // Horizontal resolution of the display [pixels].

    // User-defined parameters
    localparam  SCAN_OFFSET = 0;
    localparam  TEST_ITERATIONS = 4;

    logic CLOCK_50 = '1;
    logic hall_effect = '1;
    int scan_position;

    scan_position #(DISP_WIDTH, SCAN_OFFSET) sp0(
        .CLOCK_50,
        .hall_effect,
        .scan_position
    );

    initial begin

        @(posedge CLOCK_50);

        $display("300rpm Test");

        for (int i = 0; i < TEST_ITERATIONS; i++) begin  // 300rpm Test
            #190ms hall_effect = 0;
            #10ms hall_effect = 1;
        end

        $display("500rpm Test");

        for (int i = 0; i < TEST_ITERATIONS; i++) begin  // 500rpm Test
            #114ms hall_effect = 0;
            #6ms hall_effect = 1;
        end

        $stop;

    end

    always
        #10ns CLOCK_50 = ~CLOCK_50;  // 50MHz Clock

endmodule
```

# 10 Appendix: Pin Assignments

```
set_location_assignment PIN_R8 -to CLOCK_50
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to CLOCK_50

set_location_assignment PIN_A6 -to B[0]
set_location_assignment PIN_D6 -to B[1]
set_location_assignment PIN_C6 -to B[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to B[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to B[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to B[2]

set_location_assignment PIN_B12 -to hall_effect
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to hall_effect

set_location_assignment PIN_F13 -to C[0]
set_location_assignment PIN_T12 -to C[1]
set_location_assignment PIN_T11 -to C[2]
set_location_assignment PIN_R11 -to C[3]
set_location_assignment PIN_P11 -to C[4]
set_location_assignment PIN_P9 -to C[5]
set_location_assignment PIN_N11 -to C[6]
set_location_assignment PIN_K16 -to C[7]
set_location_assignment PIN_L15 -to C[8]
set_location_assignment PIN_P16 -to C[9]
set_location_assignment PIN_N16 -to C[10]
set_location_assignment PIN_P14 -to C[11]
set_location_assignment PIN_N14 -to C[12]
set_location_assignment PIN_L13 -to C[13]
set_location_assignment PIN_K15 -to C[14]
set_location_assignment PIN_J14 -to C[15]

set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to C[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to C[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to C[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to C[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to C[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to C[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to C[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to C[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to C[8]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to C[9]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to C[10]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to C[11]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to C[12]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to C[13]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to C[14]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to C[15]
```