Ethan Spradlin, Leo Zaydes

Professor Sarrafzadeh

CS 152B

9 June 2024

Final Lab Report

Introduction

For our capstone, our first approach was to create a driver for the Pmod OLED 2 screen, and we put a lot of research into it. We wrote an organized C file for the driver and tried to run it to turn the screen on, but the screen never powered on. Later we found that the board only outputs 3.3V to its Pmod ports, while the screen required 5V.

On Monday of the week of the demo we scrapped the driver idea and went with a new idea: an Etch a Sketch. This github link provides the source project and code for both the OLED2 and the Etch a Sketch.

Design:

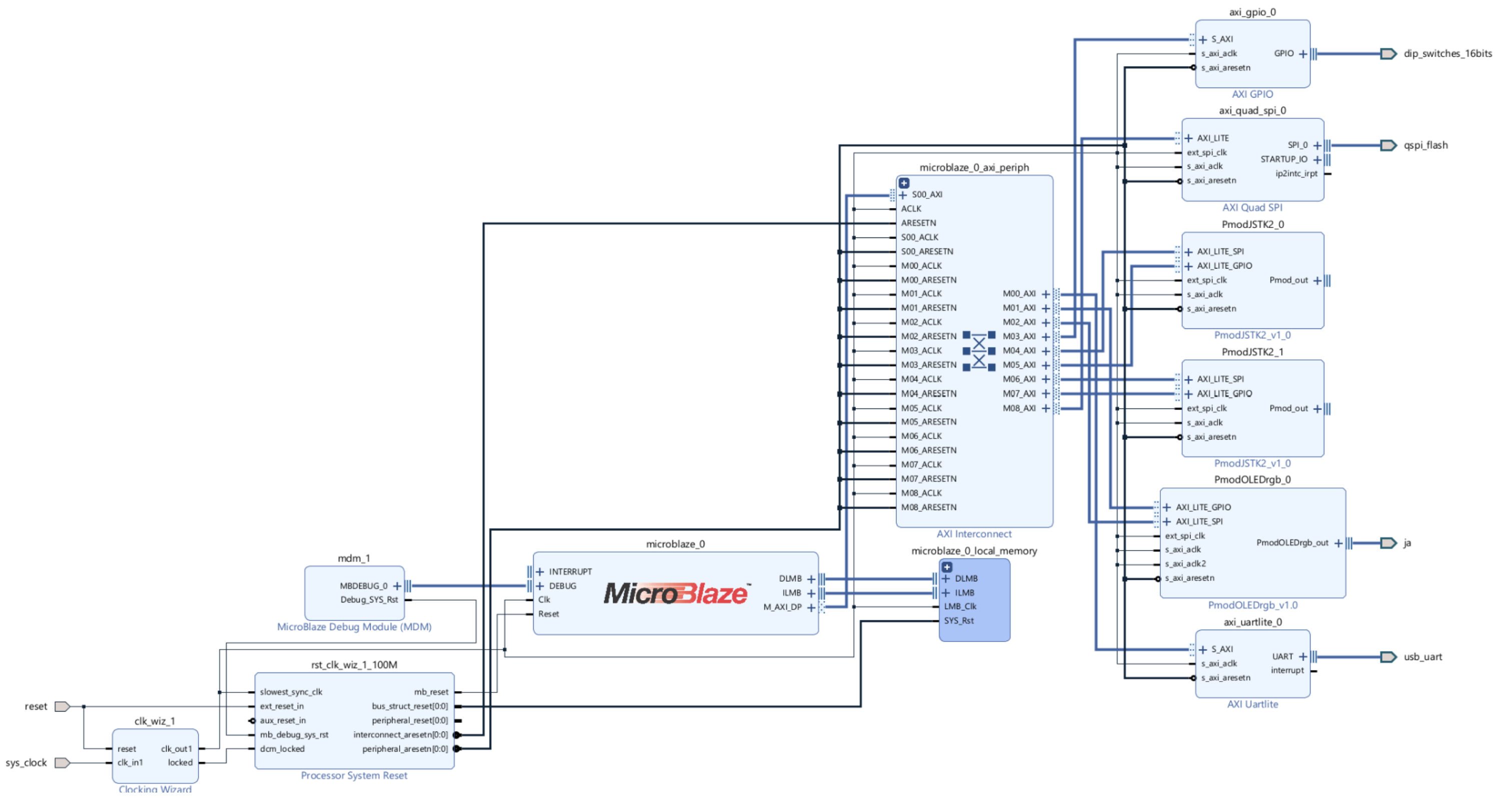
i **Project Introduction**

We recreated the “Etch a Sketch” children’s toy/game with our FPGA board with two joystick PMods and an OLEDrgb screen. Essentially, the screen starts blank and the user can use the right joystick to “draw” on the screen and the left joystick to “erase” parts of their drawing on the screen. We also have a feature to reset the screen (to blank) as well as changing the “light mode” which changes it from a white background and black drawing to a black background and white drawing (and vice versa). The reset and light mode are controlled by switches on the FPGA board. We chose this because it was a fun, entertaining application for users.

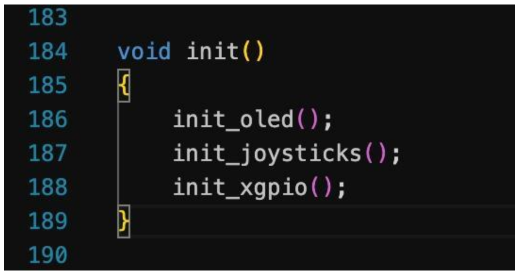
ii. **System Description**

The system programmed to the FPGA made up of the following:

- Microblaze CPU and standard Microblaze modules
- PmodJSTK2 - Port jc (left joystick)
- PmodJSTK2 - Port jxadc (right joystick)
- Pmod OLED rgb - Port ja
- AXI quad SPI (for SPI protocol in joysticks)
- AXI GPIO (for reset, light mode switches)
- UARTlite (for debugging)

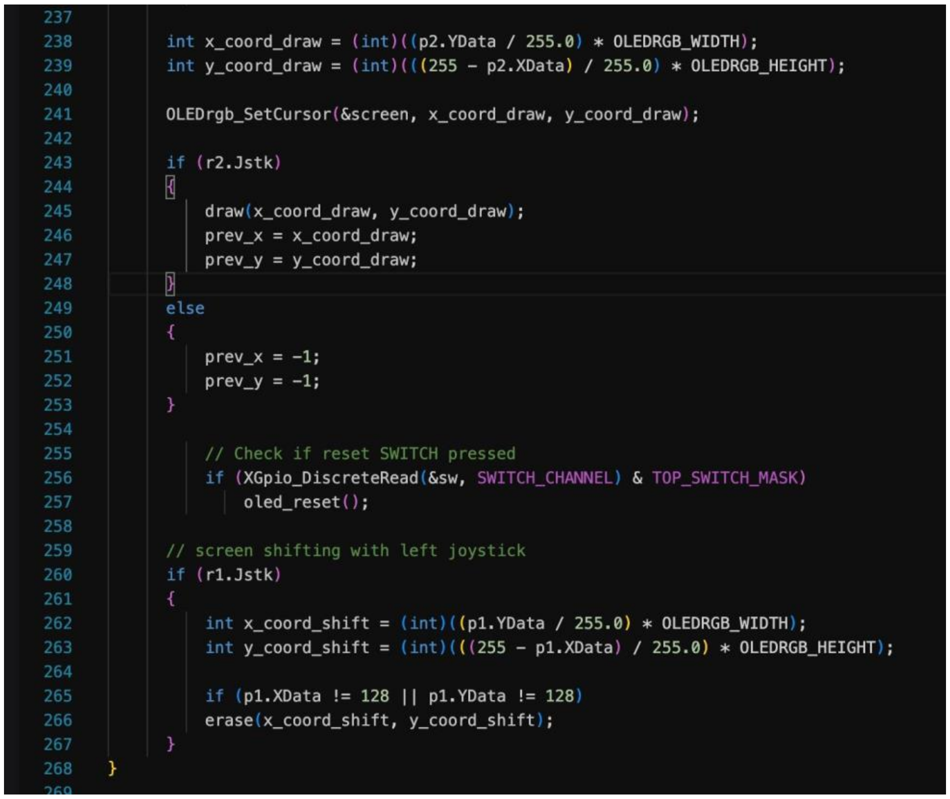


iii. **Algorithms/Code Snippets**



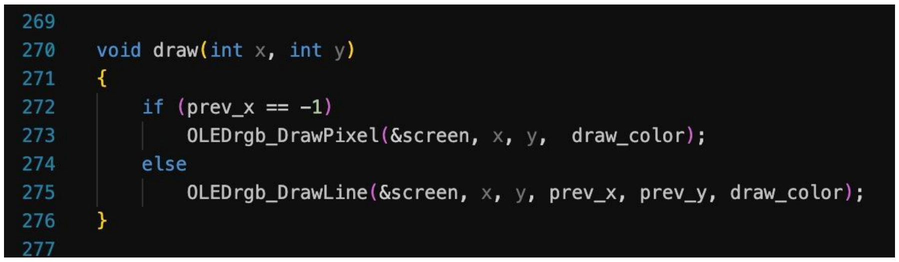
```
void init()
{
    init_oled();
    init_joysticks();
    init_xgpio();
}
```

This code initializes all the PMODs and controllers before the rest of the logic. It uses 1 function to encapsulate all the initializations (OLED, joysticks, and on-board GPIO).



```
        int x_coord_draw = (int)((p2.YData / 255.0) * OLEDRGB_WIDTH);
        int y_coord_draw = (int)(((255 - p2.XData) / 255.0) * OLEDRGB_HEIGHT);

        OLEDrgb_SetCursor(&screen, x_coord_draw, y_coord_draw);

        if (r2.Jstk)
        {
            draw(x_coord_draw, y_coord_draw);
            prev_x = x_coord_draw;
            prev_y = y_coord_draw;
        }
        else
        {
            prev_x = -1;
            prev_y = -1;
        }

            // Check if reset SWITCH pressed
            if (XGpio_DiscreteRead(&sw, SWITCH_CHANNEL) & TOP_SWITCH_MASK)
                oled_reset();

        // screen shifting with left joystick
        if (r1.Jstk)
        {
            int x_coord_shift = (int)((p1.YData / 255.0) * OLEDRGB_WIDTH);
            int y_coord_shift = (int)(((255 - p1.XData) / 255.0) * OLEDRGB_HEIGHT);

            if (p1.XData != 128 || p1.YData != 128)
            erase(x_coord_shift, y_coord_shift);
        }
    }
```

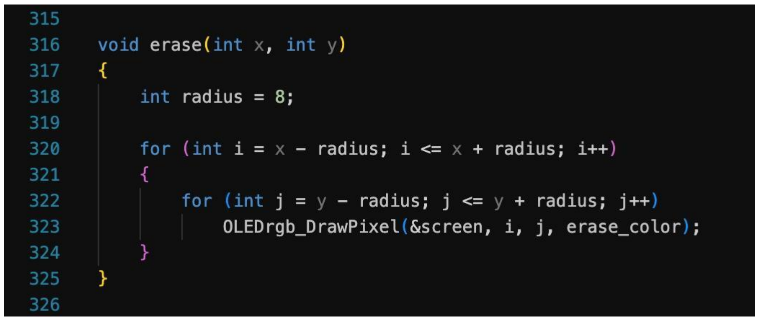
This code reads the joystick data. The joystick coordinates are automatically given in a range from 0 to 255 while the screen is 64 x 96, so we have to normalize the coordinates to the length and width of the screen. In order to make sure the drawing doesn't always start from the center of the screen, we have the drawing only occur when the right joystick is pressed down (we do the same for erasing on the left joystick). We also swap the X and Y coordinates of the joysticks since they are attached sideways to the FPGA board.



```
void draw(int x, int y)
{
    if (prev_x == -1)
        OLEDrgb_DrawPixel(&screen, x, y,  draw_color);
    else
        OLEDrgb_DrawLine(&screen, x, y, prev_x, prev_y, draw_color);
}
```

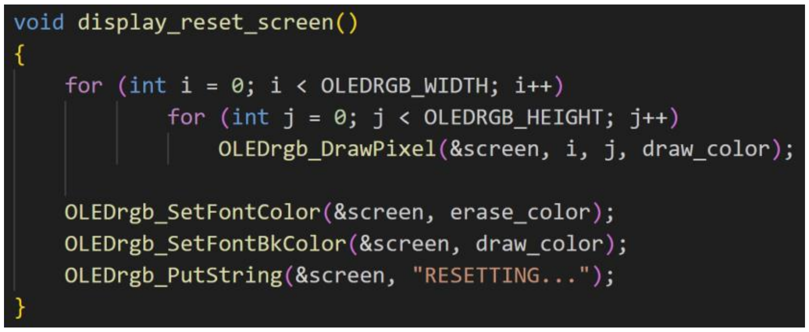
The draw function draws a pixel at the coordinated defined by its inputs x and y. A line is also drawn between (x, y) and the previous pixel drawn. If the OLEDrgb_DrawPixel function is always used instead of OLEDrgb_DrawLine, the drawing looks like a series of dots and is not viable. Thus drawing a line is absolutely necessary.

In the case where the pixel being drawn is the first pixel since the joystick was pressed down, there is no previous pixel to connect a line to and so only the pixel at (x, y) is filled.



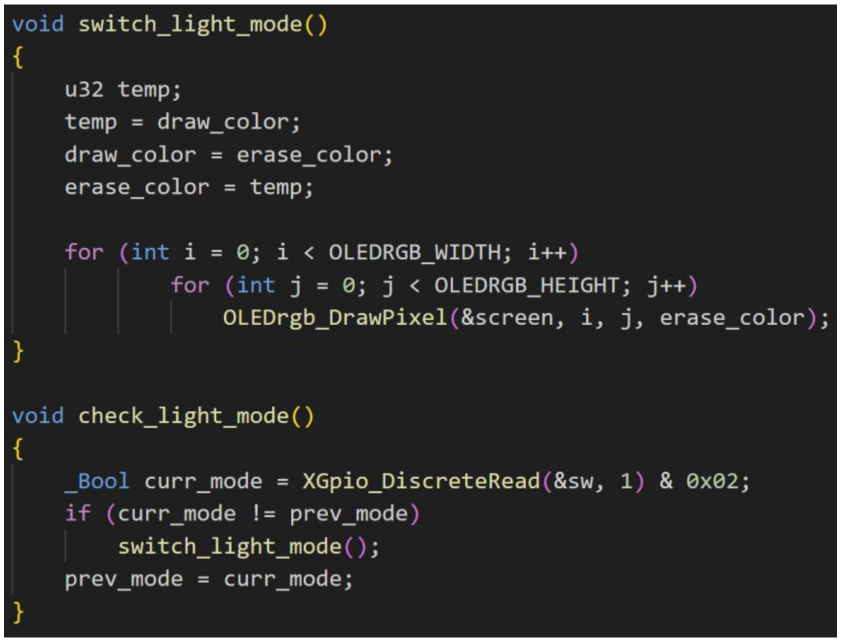
```
void erase(int x, int y)
{
    int radius = 8;

    for (int i = x - radius; i <= x + radius; i++)
    {
        for (int j = y - radius; j <= y + radius; j++)
            OLEDrgb_DrawPixel(&screen, i, j, erase_color);
    }
}
```

The erase function defines a radius of 8 pixels around the cursor's location and, if the left joystick is held down, erases all pixels within the radius by coloring them the background color (erase_color).



```c
void display_reset_screen()
{
    for (int i = 0; i < OLEDRGB_WIDTH; i++)
            for (int j = 0; j < OLEDRGB_HEIGHT; j++)
                OLEDrgb_DrawPixel(&screen, i, j, draw_color);

    OLEDrgb_SetFontColor(&screen, erase_color);
    OLEDrgb_SetFontBkColor(&screen, draw_color);
    OLEDrgb_PutString(&screen, "RESETTING...");
}
```

When reset is toggled on, it displays a blank screen (color determined by the current light mode) with the message “RESETTING…”. After the reset switch is switched back, the screen is set to blank and the user can freely draw.



```c
void switch_light_mode()
{
    u32 temp;
    temp = draw_color;
    draw_color = erase_color;
    erase_color = temp;

    for (int i = 0; i < OLEDRGB_WIDTH; i++)
            for (int j = 0; j < OLEDRGB_HEIGHT; j++)
                OLEDrgb_DrawPixel(&screen, i, j, erase_color);
}

void check_light_mode()
{
    _Bool curr_mode = XGpio_DiscreteRead(&sw, 1) & 0x02;
    if (curr_mode != prev_mode)
        switch_light_mode();
    prev_mode = curr_mode;
}
```

Our function that checks the light mode checks if the mode has been switched due to the switch on the board being toggled. If it has, then it calls the function that switches the light mode. The function that switches the light mode swaps the drawing and erasing colors as well as resets the screen to the erase color.

iv. **Expected vs. Real Output**

Our output that we achieved did reach our expectations, and even exceeded them. However, at first, we wanted to use the large OLED screen, but it had little to no documentation as well as no driver. We started to write the driver code for the big OLED screen, but did not end up successful. We shifted to the small OLEDrgb screen and achieved our goals, as well as implementing additional features beyond our expectations. Our output achieved was a completely successful etch a sketch game, with the abilities to draw, erase, reset screen, and change light mode.

Participation:

Both of us participated in the Verilog code as well as the report.