µFlight

The Micro-dodging spaceship game to end all programmers' careers.

Hugo Menendez
Aaron Cantu
Fabricio Zuniga

Team #11

EE 3463
22 April 2019

University of Texas at San Antonio

µFlight

Team #11

I. Abstract

µFlight is a 2-Dimenionsal side-scrolling video game involving a spaceship aptly named "~z" that dodges 1's and 0's that fly across the screen.

II. Index Terms

VIVA microcontroller board, UTSA microcontroller board, PIC16F1829, 8-bit microcontroller project, piezo buzzer, 128 x 64 SSD1306 OLED display, 2-D joystick, breadboard, PWM, PFM, I2C.

III. Introduction

Figure 1 – Arcade Machine-inspired interface.

µFlight is a 2-D side-scroller video game that allows the player to move vertically with a joystick in order to dodge 1's and 0's that fly at the spaceship "~z" (named for the ASCII characters that were modified in order to display it on the OLED). A demonstration of the final product can be seen here: https://www.youtube.com/watch?v=3wbYu8I93qg

IV. Project Requirements

This project had the following requirements:

1. The ship had to have vertical movement via joystick (which required A2D conversion).
2. The ship had to be able to fire projectiles.
3. Enemies would be randomly generated and would try to crash into the player.
4. There would be a start menu.
5. There would be a game over screen that triggered when the player was hit by an enemy.
6. Everything would be displayed on a 128 x 64 SSD1306 OLED display and programmed with a PIC16F1829 chip.
7. Music had to be made for the game using a piezo buzzer.
8. Everything would be displayed as an old arcade machine complete with its own custom art for the game.

V. Hardware Design

Figure 2 – Hardware storage.

The group needed to incorporate a 128 x 64 SSD1306 OLED display, which required the use of I2C to write to it. This meant the group had to configure two pins for the SDA and SCL lines for the OLED, which were pins RB4 and RB6 respectively. In addition to this, pins RC2, RC3, and RB7 (the for the push button) had to be configured as analog inputs for the joystick. Both the joystick and OLED were set up on a single Viva Board (Figure 4), while the piezo buzzer was set up on a different one, as pictured in Figure 3 (RC3 was configured for it). The piezo buzzer required the use of PFM to modify the different frequencies that were needed to make a song. Since the group needed to fit all this hardware into an arcade machine, the arcade needed to be designed large to make enough room, as seen in Figure 2. One issue the group considered was the size of the OLED was too small, so the group decided to use a magnifying glass to make it a little easier for the player to see.

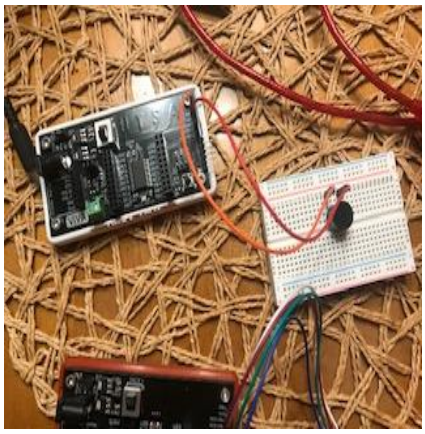
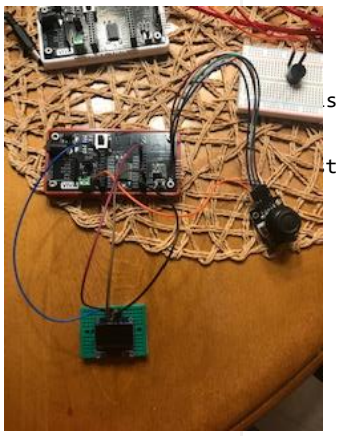

Figure 3 and Figure 4 – Wiring setups for both boards.

VI. Software Design

µFlight was programmed entirely in C using MPLAB X IDE with compiler XC8 version 1.45. The I2C and SSD1306 headers were provided by Dr. Morton and were left unmodified, whereas the main source file was changed for the group's design. A header file was created to store bitmaps for the start menu and game over screen that were made on a program called "Paint.net" and then converted to arrays in C using software called "LCD Assistant." The spaceship, as briefly mentioned before, was made by modifying the ASCII characters "~" and "z" and then displayed with one of the functions provided by Dr. Morton's code (Figure 5).

```
0x3F,0x3C,0xB8,0xB8,0xF8,0xF8,0xFC,0x7E,0x
00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
// ~
0x7E,0x3E,0x3C,0x3C,0x38,0x38,0x18,0x10,0x
00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
// z
```

Figure 5 – Spaceship code.

The group also made a header file for the joystick, whose functions were later called in the main file and then programmed under a switch statement that controlled vertical movement of the ship (Figure 6).

```
switch(x){

        case 0: //start screen. Only time it
s when game is over or time elapsed.
            SSD1306_Out16( 0, 0, "~z", 1 );
tLane();
            break;
        case 1: //This is going to be position

        if(Y_val == 4){
            //display image 2
            x++;
            SSD1306_Out16( 2, 0, "~z", 1 );
            SSD1306_Out16( 2, 0, "ww", 1 );
            //SSD1306_Image( plane2, 1 );
            Y_val = 1;
            //firstLane();
            if(Y_val == 4){
                //display image 3
                x++;
                SSD1306_Out16( 4, 0, "~z", 1
);
                SSD1306_Out16( 4, 0, "ww", 1
);
                //SSD1306_Image( plane3, 1 );
                SSD1306_Out16( 4, 0, "~z", 1
);
                SSD1306_Out16( 4, 0, "ww", 1
);
            }

        }else{
            //if joystick isn't moved stay in
2
            //SSD1306_Image( plane2, 1 );
            SSD1306_Out16( 2, 0, "~z", 1 );
            SSD1306_Out16( 2, 0, "ww",
```

Figure 6 – Sample joystick code.

VII. Testing

An issue that immediately arose when the group started testing was insufficient memory to store bitmaps. Originally, the group had made individual bitmaps that corresponded to individual positions of the spaceship, and the joystick was programmed to cycle between them. However, this wasted most of the program memory of the chip, so that idea was quickly scrapped. Another issue with this plan was that it did not allow for enemies to be programmed to move in conjunction with the movement of the ship. The solution to this was modifying ASCII characters to be able to save memory and display multiple things at once. Other aspects that were tested include:

- Proper movement (including propagation delay) of the ship via the joystick.
- The amount of time each screen appeared for.
- Generating different enemies (which at this point were changed to be 1's and 0's).
- Attempting to run two different functions at once.
- Appropriate sound effects for the piezo buzzer.
- Looping the start menu at the push of a button.

VIII. Results

The arcade machine was built with little-to-no issues and the UI was simple enough for anyone to start playing. The vertical movement of the ship felt fluid and accurate, the start menu was creative and appropriate, the game over screen was functional (although the user had to push a button to display this screen), and the game looped once the game over screen finished displaying. Furthermore, the piezo buzzer worked very well and provided the game with quaint little song that looped infinitely. The one glaring flaw and biggest deviation from the original design was the exclusion of the randomly generated enemies/obstacles. Simply put, the group was not able to find a way to run two functions at once to work this into the final product. This meant there was no "real" game to play, which was unfortunate for those that stepped up to test the game.

IX. Conclusion

The group quickly learned that memory management is extremely important and can drastically shift the direction of a project. Furthermore, the realization that running two functions with this type of chip is nearly impossible (or at least very difficult) altered the way future designs will be planned by the group. This project has vastly improved the group's understanding of microcontrollers and embedded design by focusing on adequate collaboration, in-depth research, and requesting help by those more experienced. In conclusion, the project was a moderate success and a very eye-opening experience for the group, which will undoubtedly affect their individual career interests in a positive way.

X. References

1. Dr. Morton's header files and starter code.
2. https://www.youtube.com/watch?v=cURh2-dTuI0
3. http://ww1.microchip.com/downloads/en/devicedoc/41440a.pdf

APPENDIX A

EE3463 Final Project Proposal Form

Date: 03/08/19

Team #: 11

Team Members (1, 2 or 3 only)

___Hugo Menendez___

___Aaron Cantu___

___Fabricio Zuniga___

Project Title: 2-D Side-Scrolling Game

Project Description:

This will be either an adventure-type side-scroller similar to the likes of Super Mario or one of those games where the screen is constantly moving and you're moving your character (likely a space ship) vertically while shooting. We will use a joystick to control the inputs of movement of our character (along with an A2D module for it), we will code the obstacles/enemies using loops and timer modules where needed, and we will display it all on either an OLED display or LCD Display (perhaps using multiple at once). In either case, it should take us 3 or 4 weeks to complete.

Modules and Devices used:

- OLED Display/ LCD Display
- Timer Modules
- Joystick
- A2D module
- Possibly having Bluetooth incorporated

Mid-Project Bench Mark Described:

We should have at least the animations and game mechanics done by the halfway point. The latter half will consist of us working out the controls and UI. If there is sufficient time, we will incorporate our own custom sounds for the game.