CS 3420 / ECE 3140 Final Project

Matthew Daniel (mrd89) Ankush Rayabhari (ar2354), 2019-May-17
Video Demonstration: https://www.youtube.com/watch?v=Zpe5f97Re1w

Project Introduction

Our project is based off of the design of the hard pre-packaged project, namely using a joystick and a 32x32 LED RGB screen to explore a virtual space. Our original idea from the proposal was to have a basic version of Super Mario Bros in which the goal of the player (displayed on the LED screen) is to jump over obstacles and enemies to make it to the other side of the map. The joystick is to move the player right and left, and the button was to jump. We 3d printed a hand-held controller and an apparatus to hold the screen in an upright position.

We spent upwards of two weeks trying to get our 32x32 RGB LED screen to work with the FRDM board. After countless hours, we finally got it to work, but at that point we realized it would take too much time to complete the original Mario game. Instead, we used the prepackaged game where the board displays a player shown on the screen which we can move around using the joystick in all directions. When the player hits the edge of the screen, we turn on the onboard LED as a warning. This final project achieved creating a game using our FRDM K64 board, LED matrix, and a joystick. Interfacing with the board took many hours of hard work and a lot of debugging to get working in time and taught us a lot about different I/O devices and how each has a specific method by which we need to control them.

System Diagram

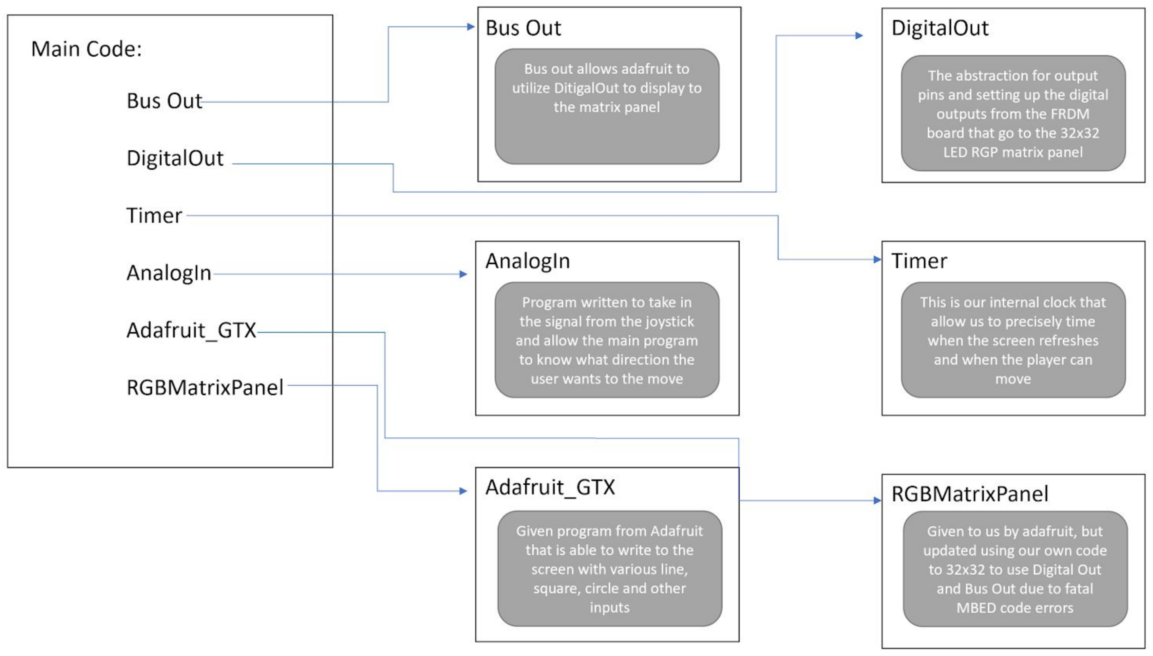
Main Code:
Bus Out
DigitalOut
Timer
AnalogIn
Adafruit_GTX
RGBMatrixPanel
Bus Out
Bus out allows adafruit to utilize DitigalOut to display to the matrix panel
DigitalOut
The abstraction for output pins and setting up the digital outputs from the FRDM board that go to the 32x32 LED RGP matrix panel
AnalogIn
Program written to take in the signal from the joystick and allow the main program to know what direction the user wants to the move
Timer
This is our internal clock that allow us to precisely time when the screen refreshes and when the player can move
Adafruit_GTX
Given program from Adafruit that is able to write to the screen with various line, square, circle and other inputs
RGBMatrixPanel
Given to us by adafruit, but updated using our own code to 32x32 to use Digital Out and Bus Out due to fatal MBED code errors

Figure 1: Block Diagram for Core System Design

Want tight control of when stuff is called. Want to be completely in sync of when update display is called etc.

Hardware Description

We were planning on using a total of 3 I/O devices in our project, but had to limit our scope to just the screen and joystick and use the onboard LED instead. The 32x32 LED panel requires 13 pins, the button requires two pins, and the joystick requires 4 pins. The FRDM board is more than capable of handling the total number of 19 pins. The display requires at most 5V 4A when running all the LEDs on full brightness and rarely uses that much power, so the selected power was fine for our needs for just powering the display. Each of the remaining input components was powered by the board's built in 5V pins and the current supplied from the board itself is more than sufficient to operate them. Here is the final wiring diagram of the system:

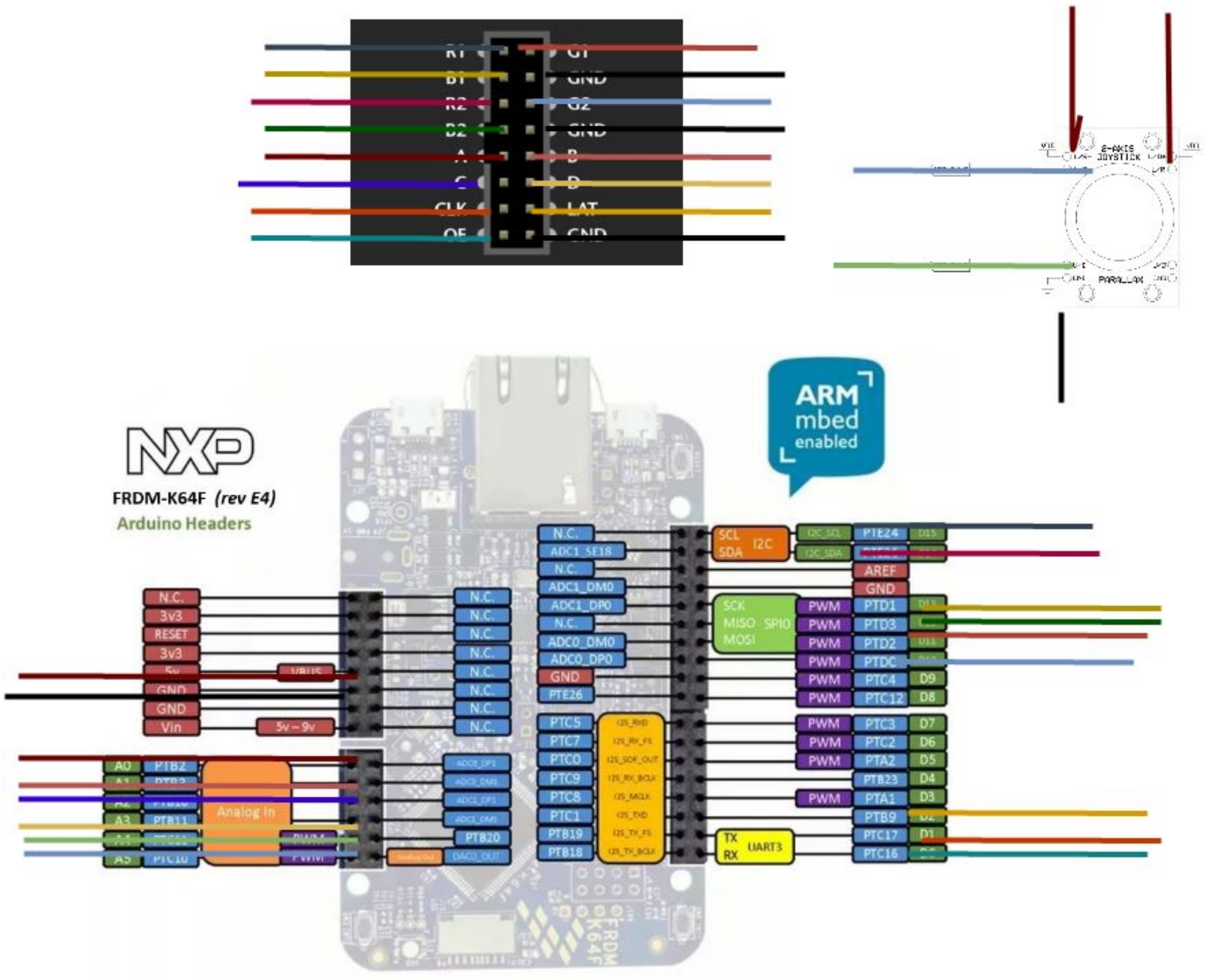

Figure 2: Schematic for Board

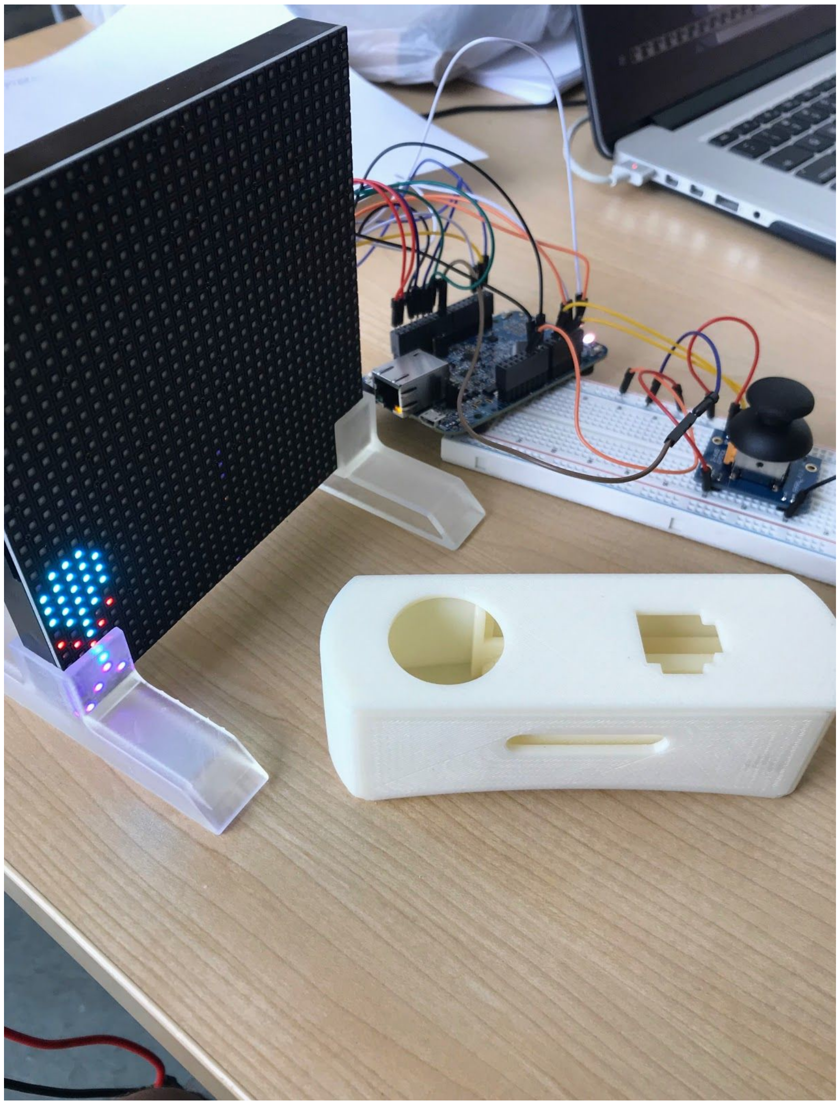

FIgure 3: 3D Printed Controller and Stand for RGB Board

Bill of Materials

Item	**Quantity**	**Price**
32x32 RGB LED matrix panel	1	$45
2-Axis Joy Stick	1	$6.50
3D printing material (clear resin Formlabs) (already acquired)	N/A	0
5v 4A Power Supply	1	$15.00

Wires/Misc Resistors/Breadboard (already acquired)	N/A	0
Freedom K64 board	N/A	0

Software Description

There are a few main classes of the system:

- main.cpp
    - This contains the driver program for running the game and sets up the DigitalOut pins and the RGB matrix, the LED bus for the onboard LED, the AnalogIn pins for the joystick and the Timer for calling updateDisplay at 6 KHz and the game loop every 1/15th of a second.
- Adafruit_GFX
    - This program was downloaded from the MBed repository as a port of the original arduino version. It is the core graphics library for the display providing a common set of graphics such as lines, circles, and other basic graphics. We modified this slightly to remove any reference to any MBed standard libraries as well as fixing any compilation issues that came from that change. On the whole, this was mostly untouched.
- AnalogIn
    - This is a wrapper class around the ADC converter 0. This allows us to read in a 16 bit value from an arbitrary channel. We used this to wrap around the X and Y axis potentiometers of the joystick which output the position based on a signal with a 16 bit number from 0 to 0xFFFF. We used this reference for the K64 to write this: https://os.mbed.com/media/uploads/defrost/frdm-k64_adc_configuration.pdf
- DigitalOut
    - DigitalOut is a wrapper class that generalizes configuring a specific pin on a port as a GPIO output pin. We wrote this class ourselves using the techniques covered in the 4th discussion session. We also overloaded the assignment operator that allows us to directly set the pins value. This was to ensure compatibility with the RGBMatrixPanel library.
- BusOut
    - BusOut is another wrapper class that generalizes writing to multiple GPIO pins at once. This class works by taking a value and setting each bit in the value to the corresponding DigitalOut in the bus. This was also written to ensure compatibility with the RGBMatrixPanel library.
- RGBMatrixPanel
    - This was originally written for the Arduino but the 16x32 version was porten to Mbed. We used the ported version as a starting point, but we made key updates to this program to allow use for our FRDM K64 board. Instead of utilizing Mbed, we substituted the classes it used (BusOut and DigtialOut), to remove the Mbed

dependency. To make the display work for a 32x32 matrix instead of just a 16x32 matrix, we looked at the datasheet for the display and checked what we needed to update. We had to create a D pin select to address the linked rows of the display in rows 17 or higher, which we added to the row bus. We then removed any mention of Mbed in the interface file and then fixed any compilation issues that arose from it.

- Player
    - The player class contains the core functionality of the game. We store a few key variables: x position, y position of the top right corner of the players position. We have an update function that calculates the new position based on the analog inputs. We set it such that the player is allowed to be at the screen's edge, but if it tries to go past the screen, the collision detection gets triggered and the on-board led goes off. The LED goes off when the player is at the edge of the screen as well. Furthermore, we created a draw function that clears the screen and then draws the player in a new position.
- Timer
    - The timer class wraps around the PIT timer that manages running the periodic game loop and display updates. We split up the loop into a major cycle and a minor cycle. On each minor cycle, we update the display as the display functions by only lighting up certain rows at a time. To give the illusion of a continuous image on the screen, we refresh the display at 6 KHz. On each major cycle, we call the game loop update function to process input and update the screen. This class was written generically with generic handlers and an arbitrary major/minor cycle length in mind.
    - We didn't use interrupts but rather polling for this as we wanted the board to loop forever so that the game would not stop. Additionally, we didn't want to give up control to an interrupt at such a high frequency to avoid updating the graphics buffer while it is being written to the display.

Testing

Thoroughly this whole project, we tested tirelessly after every major edit to ensure we would not miss an error. After each test, we uploaded the files to github with comments. This was extremely helpful for both sharing files and going back to previous versions after we made errors in our code. We first worked on creating the program to display to the LED panel since that was the first milestone on our agenda. Once we were able to get the board to display after countless hours, we tested a few of the shapes and colors for the display to ensure stability and correct implementation of all functions. Following this, we set up the joystick program and tested in the debugger. To do this, we set a breakpoint after the program reads a value, and moved the joystick to its max potential and ran again to the breakpoint. Both of these peripheral tests worked well leading to us believing our implementation of both I/O devices is correct.
We tested the timer class by checking the speed at which we updated the screen visually. To test the update position timer, we called the on-board LED to toggle as the timer finished to tune

the frequency at which the game should update player position. On completion of the game, we further tuned the frequency to allow for the optimal player speed. Finally, we tested the game many times by moving the player to every position possible on the screen with no errors or issues. To check that the collision detection system, we again moved the player to each of the edges to ensure it was correct. Finally we called over someone in Duffield to play the game, and he thought it was entertaining and was able to pick up the game immediately!

Results and Challenges

The biggest challenge as detailed in the code and video, was getting our FRDM K64 board to interface correctly with the LED RGB matrix display. Another challenge we had was getting the geometry of our character correct when moving the character around the screen.
After completing this project, we have a greater understanding of how to interface to various peripherals, how ARM K64 GPIO is done, and finally how to create a basic game in C++.

Work Distribution

For this project, we met numerous times over the past two weeks to allow for enough time to complete this project. We met the first time a few days before our project pre-proposal and then decided on a project and wrote the pre-prostal together. For the proposal, we met together again to discuss and write up our proposal which we submitted to CMS. We used a pair programming model to accomplish the lab. Since Ankush has a well-working VM, we used his laptop as the main coding machine. During this time, we were able to collaborate quite well by bouncing ideas off of each other and coming to a consensus on how to proceed. This communication was very helpful for both parties and we believe we made a good team working on this lab.
Outside of the meeting times, Matthew was able to design and 3d print the controller housing and stand for the LED matrix display. Ankush did some extra interfacing debugging and was able to figure out a few issues we were missing when we met together. Most of the project was done while meeting together.

Reference, Software Reuse

Throughout the project we referenced the K64 manual and various Piazza posts for most issues as well as the Mbed libraries for our peripherals. The only software we used from external sources directly was the RGBmatrixPanel and Adafruit_GFX from Mbed, but in each of these we had to modify their implementation a great deal to make it compatible with our processor. We also referenced a manual for how to interface with the ADC converter (linked above), but wrote the actual code ourselves for compatibility. We initially followed Micheal Xiao's confluence article for wiring the board but quickly abandoned it as it didn't work. We instead used the datasheet provided by Adafruit for the display. We wrote everything in C++ as we were more familiar with it and used online references such as learncpp to learn how to do operator overloading.