Project Repot

Erik Lamp ebl48
Kartikay Jain kj295

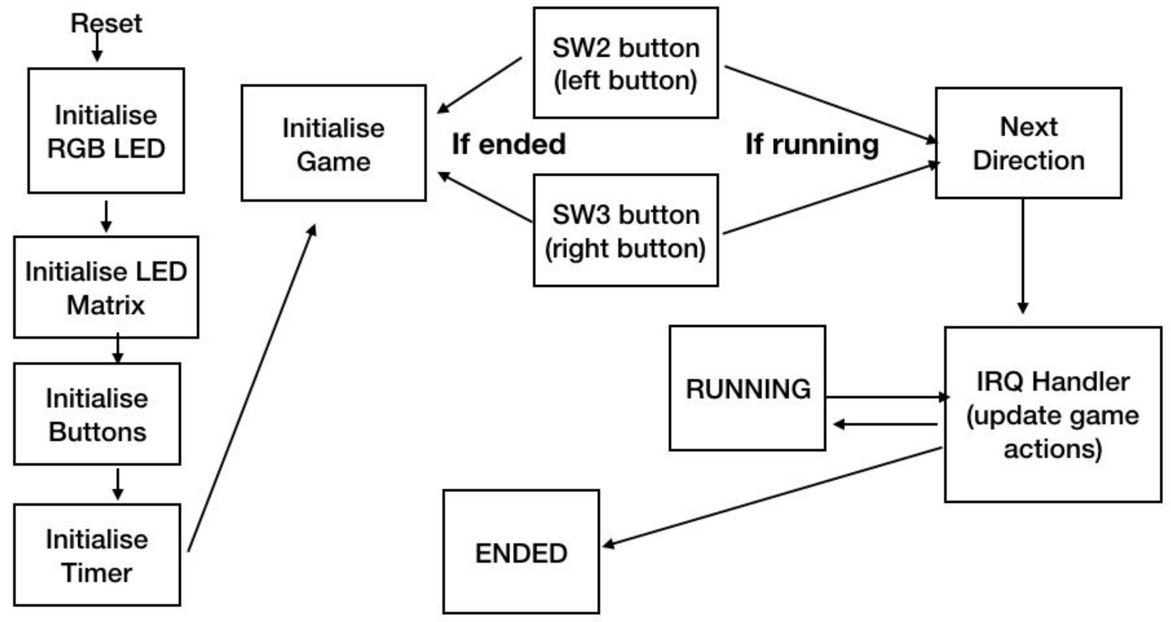
Video Link:

https://youtu.be/9KVBhZo6K5U

1. Introduction

Our project involves making the snake game through the use of an LED matrix and the FRDM board. As mentioned in the proposal the player is a snake and his goal is to eat as many apples as possible. However, each apple he eats causes him to grow one longer and move faster. If he hits the wall or eat his tail then you die and the game ends. Thus, you must try your best to obtain the highest score possible.

2. System diagram

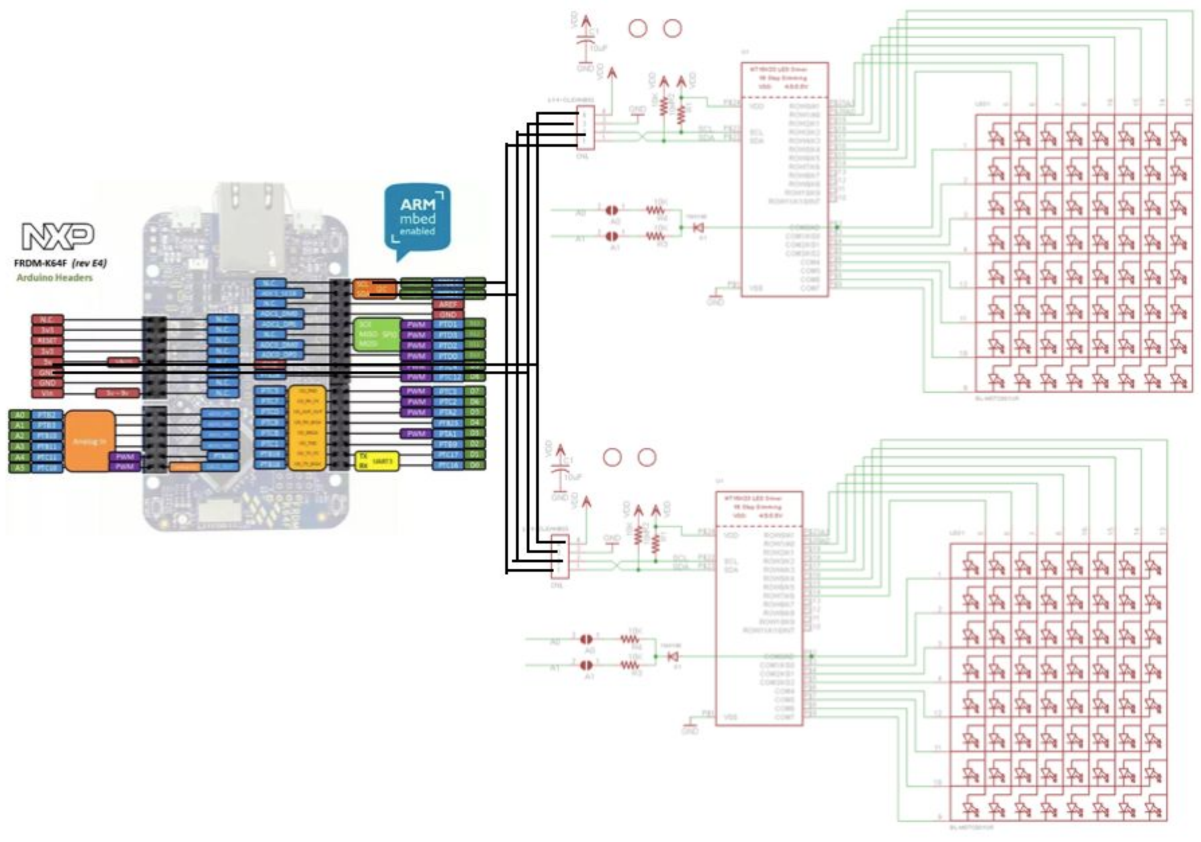


The initialization functions are called in the main function of the software program on micro vision. These help start up the game on the LED matrix. The 2 buttons on the FRDM board control the movement of the snake on the next game clock tick and what happens on each cycle is handled by the interrupt handler. All of the logic for the game occurs in the interrupts. This was decided because we wanted to only draw the screen every time there's an update, and because there is no need for any logic that occurs between game ticks. This also means that our implementation runs in real time, which is beneficial for a reaction game.

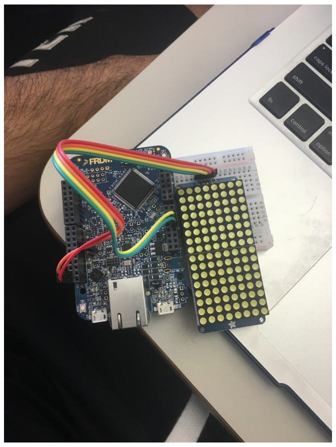
3. Hardware description

ADA2038 (led matrix): Quantity = 1 Pricing = 26.48$

ABK-01 (breadboard and cables): Quantity =1 Pricing = 11.99$
935326293598 (FRDM K64F Board): Quantity = 1 Provided by class



The FRDM board consists of multiple pins that can be used to connect to peripheral devices. We used an 8x16 LED matrix of Adafruit that required 4 connections to the FRDM board. As can be seen in the FRDM board schematic provided by Mbed there is an SCL and SDA pin which are both required to be connected to the corresponding pins of the Matrix to allow I2C communication and data transfer to take place. In addition to this the 5V and Ground pins of the FRDM board get connected to the respective pins of the matrix to provide power. These hardwiring descriptions are shown in the image below:



As shown above to actually connect all the hardware together we did the following steps. When we first bought the materials from amazon the LED matrix came with a backpack. So we soldered the matrix onto the back in the CMC lab. We then clipped the matrix leads short after soldering the required pins. We then soldered a provided 4 pin header onto the board. We then used this header to plug the matrix into a breadboard. 4 wires were connected as described above to their respective pins and the breadboard to ensure proper connection, allowing data transfer to take place between the matrix and the board.

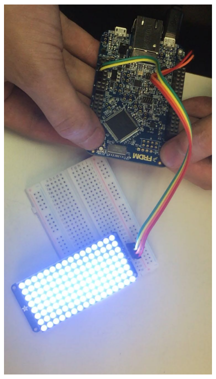
4. Detailed software description

Setting up I2C protocol:

This was one of the trickiest parts. We followed the confluence tutorial to help us set up the protocol. So we first enabled the I2C clock. Then to set the output pins for I2C we set the mux bits of pin 24 (clock line) and 25 (data pin) corresponding to SCL and SDA to 101 to allow them to serve data transfer. Then we enabled I2C and interrupts and initialized its configuration by calling I2C_MasterInit. Then we set up 3 I2C_MasterTransferBlocking calls: one to initialize oscillation, one to initialize blink rate and the last one to initialize the brightness. Finally, to draw to the board, we generated a i2c_master_transfer_t object called redraw_transfer and set its fields accordingly such that we would be able to send a buffer array which represented the led matrix where 0s corresponded to the off LEDs whereas 1s corresponded to the LEDs that were supposed to be on. So for example:

```
 uint8_t volatile buffer[16] = {
  0b11111111, 0b11111111,
  0b11111111, 0b11111111,
  0b11111111, 0b11111111,
  0b11111111, 0b11111111,
  0b11111111, 0b11111111,
  0b11111111, 0b11111111,
  0b11111111, 0b11111111,
  0b11111111, 0b11111111
};
```

The above buffer when sent as data through I2C would make all the LED lights turn on as shown in the photo below:



Before making the 3 calls we had to change the data to be sent before each call according to a tutorial on confluence. Then we checked the status of the master transfer blocking calls: if they were successful the Green Led on the board would turn however if data was not transferred the red led would turn on making debugging our code easier. Since the LEDs on the matrix turned on as well as the green led on the board we knew the I2C protocol was working. These LED debug information was removed when we started implementing the game.

Implementing the Snake Game:

To implement the game itself there were 4 main files involved:

Util.h - This file consisted of function declarations responsible for providing utility functions to make setting up the entire game easier, such as initializing the LEDs on the board as well as the matrix, redrawing everything to the board, and initializing the game timer etc.

Util.c - Implemented the functions defined in Util.h

random.h - Random number generator for the position of the apple on the matrix

main.c - Handled the game logic

In main.c we had an addApple function that was responsible for displaying the apple on random positions on the matrix. In addition we had an init_game function that set up the game and started it. The game starts with the following initial values: snake length as 2 so it appears as 2 pixels on the matrix, the direction of the snake as moving right, and the addApple function was called once to display an apple at a random position. Then we had 2 interrupt handlers that served for registering the 2 player buttons on the FRDM board responsible for turning the snake either right or left. PORTA_IRQHandler was responsible for the button on the board that turned the snake's head right by 90 degrees and PORTC_IRQHandler was responsible for the button on the board that turned the snake's head left by 90 degrees. The first function toggled the green LED to indicate the button press was read and to turn the snake right based off its current heading. The second function does the same with the red LED to signal the left button being pressed hence providing user feedback. There was a conditional statement in both functions that checked if both buttons were pressed simultaneously which would trigger the game to restart if its state was STATE_ENDED.

We then implemented the interrupt handler itself i.e PIT0_IRQHandler which was responsible for handling the game logic. It would increment steps (to help generate a new seed for a new random positon for the apple) as well as decrease the gametick each time the timer expires. When the gametick hits zero, it also does everything necessary for the game logic. It first checks if the snake hit the edges of the matrix or itself, which would trigger all LEDs on the board to turn white and change game state to STATE_ENDED to indicate to the user that game is over. It then checks if the snake's head reached the same position as the apple then it speeds up the game frequency, increases the snake length by a unit, and creates a new random apple on the matrix. Finally, if none of the above happen it just moves the snake as it normally should on the next game tick. In addition to this, the other functions called in the main function are located in Util.c. We have the enable and disable interrupt functions as usual as well as code from lab2 that was responsible for toggling each LED color and initializing them. Then we have the init_LED_matrix function which does all the I2C setup described earlier thus setting up the LED matrix such that data can then be sent from the board to the matrix. We also created an init_buttons which set up the respective top two buttons ready to be used as GPIO so that player input can be registered for turning the snake (similar to setting a GPIO pin in lab2). We had init_timer that set up the timer for the game as well as enabled interrupts. The redraw_matrix function we created is responsible for pushing all changes in a data buffer atomically to the LED matrix to be displayed. Redraw_board is responsible for transferring the changes from the game state to the matrix's data buffer, then triggering a redraw_matrix which is a useful helper function. LED_matrix is what modifies the matrix data buffer as a variable depending on the state as well as the x and y coordinates provided to it. This does the transformation from our game coordinates to the correct LED matrix coordinates. Random.h codes a random int generator using a seed which is based on the steps field in the game and the code used is referenced as being borrowed from the website shown in comments. Finally, as mentioned before Util.h is just the function declarations for the functions implemented in Util.c.

5. Testing

For testing, when we started implementing I2C we used the RGB LEDs to figure out which portions of our code wasn't working. After setting up I2C, we just directly saw how the LED matrix lit up and accordingly changed our code. We also sometimes used watches and debug mode to figure out the issues within the snake program on top of using the LEDs to provide direct visual feedback.

6. Results and challenges

We definitely met everything we proposed to do for our proposal. The most complicated part of our design was actually setting up I2C protocol as there was no proper guide to it for our LED Matrix to Board connection. We also had to set up a small state engine in order to get our game to function properly.

7. Work distribution

We communicated with each other through text messages and email as it was fast and efficient to set up meeting times. We used google docs to write the report and Erik's laptop for the coding. While working on the coding and documentation, both of us had active roles in the work due to the pair programming and google docs.

8. References, software reuse

Reference materials for connecting and setting up the LED matrix comes from here:
https://confluence.cornell.edu/display/ece3140/LED+Matrix+Arcade+Game+Suite
Reference material for setting up I2C protocol on the K64F FRDM board:
https://confluence.cornell.edu/display/ece3140/I2C+for+the+K64F
Documentation for controlling the LED matrix comes from here:
https://www.partsnotincluded.com/electronics/controlling-led-matrix-with-the-ht16k33/
https://confluence.cornell.edu/display/ece3140/Adafruit+8x8+LED+Backpack
Random.h implementation comes from here:
https://stackoverflow.com/questions/4768180/rand-implementation