LED Visual Art

E155 Final Project
December 13, 2019
Reem Alkhamis and Sabrine Griffith

Abstract
For our final project, we were interested in creating some razzle-dazzle visual art with an adorable display. So we chose the 4mm 32x32 RGB LED display. Our goal was to display a static image on the LED panel with moving pixels around the static image controlled by the tilt of the LIS3DH accelerometer using ATSAM4S4B microcontroller and Altera Cyclone IV FPGA. The microcontroller takes in data from the accelerometer as the user tilts it via SPI. The microcontroller is responsible for the animation logic and updating the location of the moving LEDs based on the accelerometer data. The new display frame is sent to the FPGA via SPI. The FPGA then uses this data to drive the LED matrix.

I. Introduction: Motivation, Block Diagram, Overview

The motivation of this project was to merge embedded systems with art. We aimed to incorporate our software, hardware knowledge to create aesthetically pleasant visual art.

The project consists of the ATSAM4S4B microcontroller and the Altera Cyclone IV FPGA. The microcontroller acts as the SPI master for two slaves: the FPGA and the LIS3DH accelerometer. The microcontroller is responsible for receiving data from the accelerometer via SPI and keeping track of the animation state. The microcontroller updates the status of a 32x32 character array which represents the LED matrix display. Then it sends all the rows of this matrix to the FPGA via SPI. The FPGA is responsible for storing the data it receives from the accelerometer into RAM and driving the LED matrix with the correct colors. Figure 1.1 shows the overall block diagram of our system.

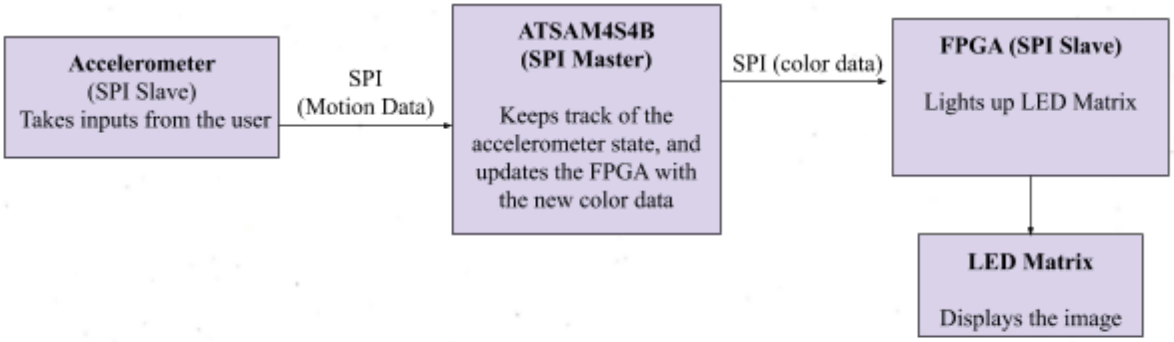

FIG 1.1: Overall Block Diagram of the Animation System

II. New Hardware

Our new piece of hardware is a 32x32 RGB LED Matrix Panel from Adafruit. This matrix consists of 1024 LEDs that are driven by the FPGA. The back of the matrix panel contains a PCB with two IDC connectors and it requires 5V input and 4A, although we found that the matrix will only need up to 2A since it does not light up all LEDs at once. The large power requirement entailed using an external source to power the device. The display also requires 13 digital pins; 7 of which are used for control signals, and 6 are used for bit data. The pin layout for those signals is shown in figure 2.1. It is important to note that the logic level for the FPGA pins are 3.3V. However, there are two 74HC245 octal bus transceiver chips that buffer all the inputs. This allowed us to connect directly to the FPGA 3.3V outputs even though this board is powered by +5V DC.

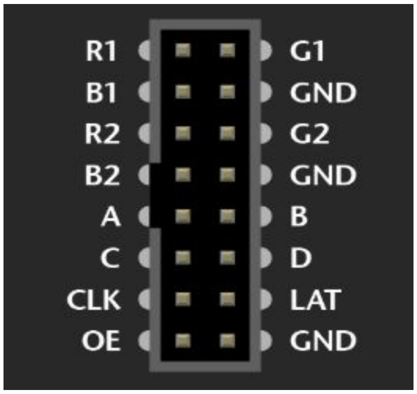

FIG 2.1: Pin layout of 32x32 LED Matrix

Adafruit does not provide any datasheets or timing diagrams for the matrix, which made it really difficult to understand how the matrix worked. In order to use the matrix, we needed to look up online tutorials and the datasheet for the shift registers on the PCB in the back of the matrix [3].

The matrix is divided into two halves, the top and bottom 16 rows. The rows are controlled with a 4-bit register called row, which corresponds to port A, B, C, and D on the matrix, where A is row[0]. The PCB on the matrix has few 74HC138 3-to-8 demultiplexers that use the inputs (A, B, C, and D) and select two of the 32 groups of pixels at a time. The two rows that are driven correspond to row and row+16. For example, to drive rows 1 and 16, row needs to be 0. This display multiplexed with a 1/16 duty cycle, which means that the row changes in response to the frequency at which the display is multiplexed.

For driving the matrix colors, both halves of the display consist of 32 shift registers for each bit. The colors for the top half are controlled by R1, G1, B1, and the colors for the bottom half are controlled by R2, G2, B2. Thus, there are 6 separate 32 shift registers in total. Those shift registers correspond to the columns of the matrix. So each clock cycle color data is shifted into the corresponding column and row on the matrix.

The matrix also has an active high LATCH signal and an active low OE signal (BLANK). We used a 6-bit counter to keep track of the current column, row, LATCH and BLANK signal. Those signals are further discussed in section IV.

III. Schematics

The schematic of our entire system is shown below:

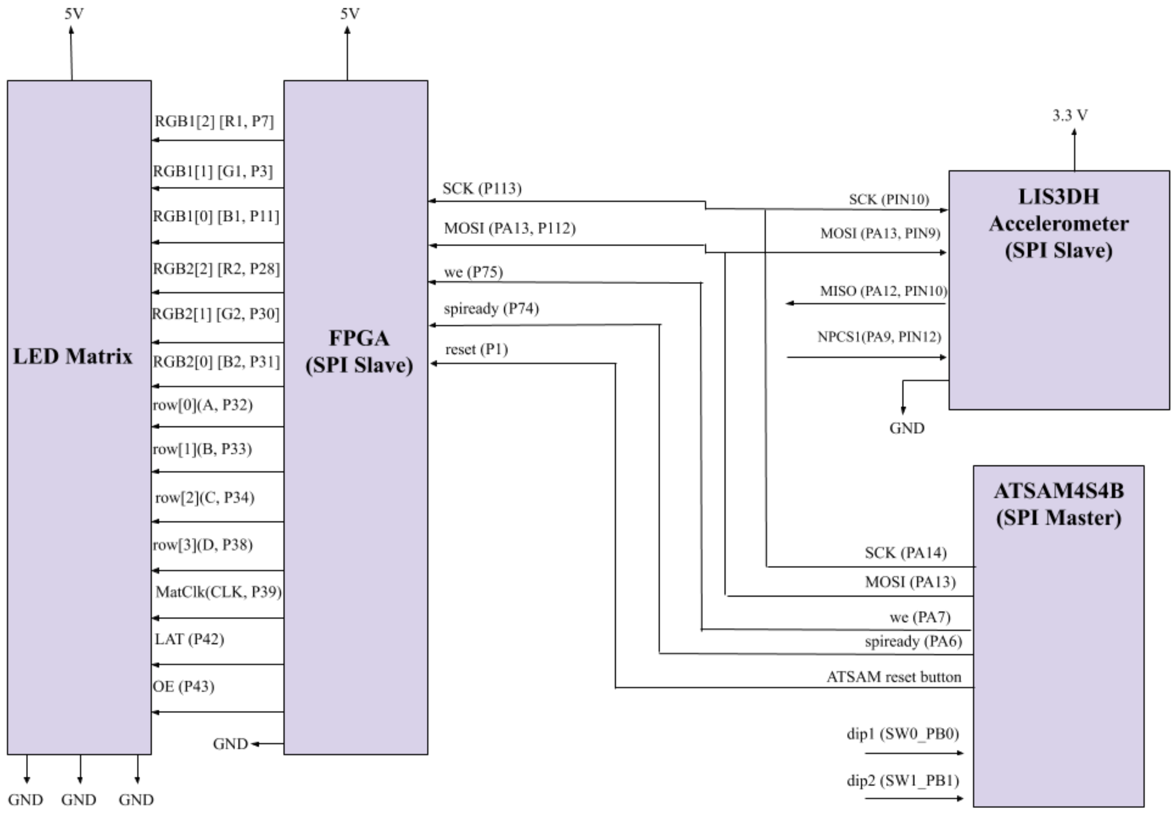

FIG 3.1: Schematic of the LED Visual Art

As stated above, the microcontroller and the FPGA are communicating through SPI. The microcontroller and the accelerometer are connected also via SPI. The LED matrix is driven by the FPGA. For clarity on the matrix signals names written next to the assigned names on the FPGA, refer to figure 2.1.

IV. FPGA Design

The SystemVerilog code is comprised of six primary modules: one module to control the LATCH and BLANK signals going to the matrix, one module to iterate over all addresses in memory, two RAM modules to read two rows to the matrix simultaneously, one module to drive the RGB colors for both halves of the matrix, and one module to receive the rows from ATSAM via SPI. The overall block diagram of the FPGA is shown in figure 4.1.

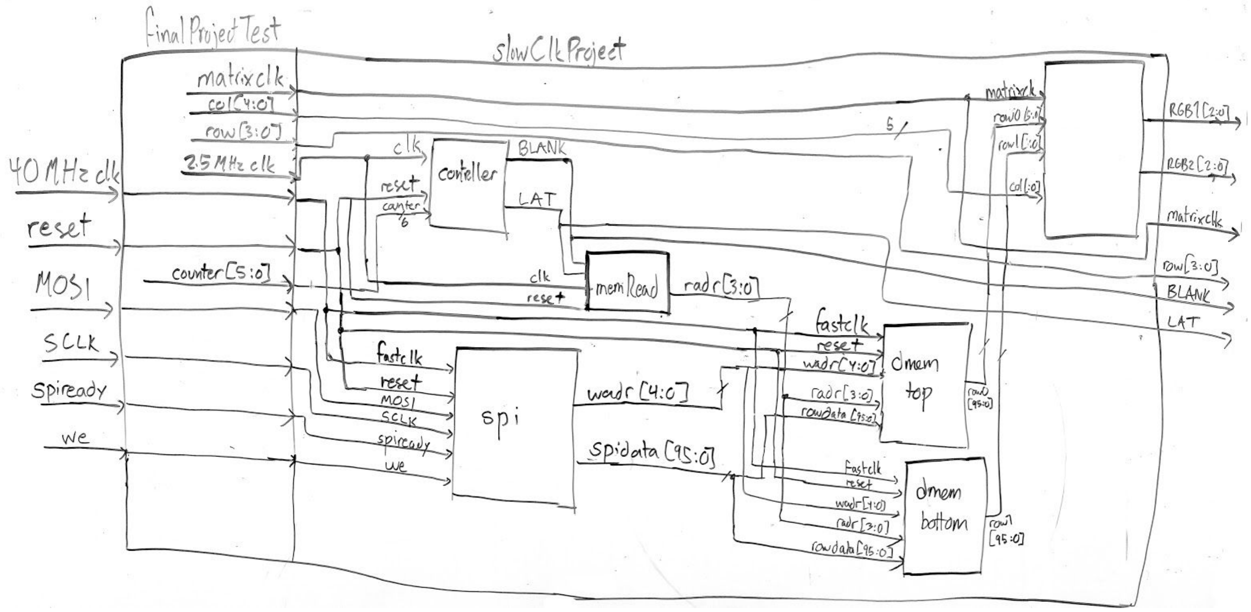

FIG 4.1: Overall FPGA Block Diagram

The “controller” module takes a 6-bit counter input and outputs the LATCH and BLANK signals going to the matrix. Because the matrix is made up of 32 shift registers for each bit of color, it was important to choose a clock that would only be clocking to shift the color data and it pauses during the LATCH and BLANK period. After shifting the 32 RGB colors, the LATCH signal is asserted for one cycle, then the BLANK signal is de-asserted for a fixed number of cycles. This arbitrary number of cycles determines the brightness of the pixels. Figure 4.2 shows the timing diagram for our LATCH and BLANK signals relative to the matrix clock.

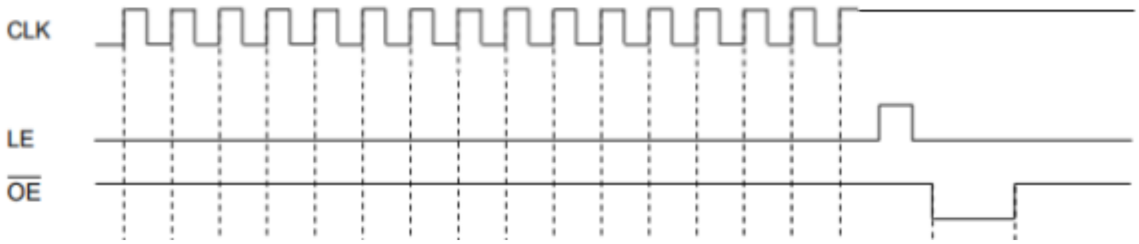

FIG 4.2: Timing Diagram for LATCH and BLANK

The “memRead” module takes the LATCH and BLANK as inputs and it outputs a 4-bit counter that is used as a reading address “radr”. The LATCH and BLANK are used as enable signals to allow a radr counter to increment. This counter then is used by both “dmemtop” and “dmembottom” modules.

Since we decided to store the static image in RAM, this decision added a level of complexity to our design. Since we are using all three bits for each pixel, then each row will be 96-bit long. The dmemtop and dmembottom modules take a writing-enable signal from SPI ‘we”, writing address from SPI “wadr”, 96-bit long data from SPI (row_data), and a reading address from memRead radr. Those two modules map into two distinct two-port 32x96 RAMs. The reason we had two RAM modules was to read two rows simultaneously. The first module reads all the rows for the top half of the matrix and the second module reads all the for the bottom half of the matrix. The dmemtop module uses radr to output row data in the top half of the matrix “row0” and dmembottom module uses radr+16 to output row data in the bottom half of the matrix “row1.”

The “driveColor” module takes a column counter “col”, color data for the current row stored in row0 and row1, and the matrixClk. This module is responsible for shifting out the RGB1 and RGB2 colors for both the top and the bottom halves of the matrix based on the color data in row0 and row1.

The “spi” module is a recieve-only module. It takes 108 bits in a sequence. Those bits consist of 96-bit row data, 5-bit write address and three unused bits. It also takes two signals coming from ATSAM: spiready acts as a chip enable for the transfer of 13 bytes, and we is a write-enable signal to indicate that the transfer of data has ended. Figure 4.3 shows waveforms for SPI signals.

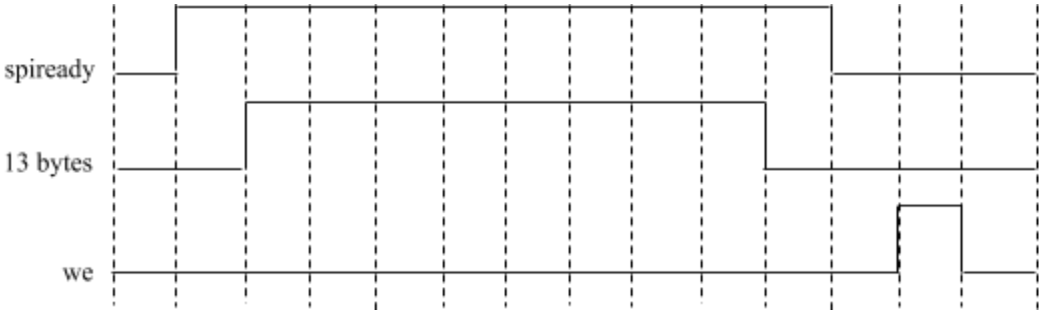

FIG 4.3: waveforms for SPI transfer

V. Microcontroller Design

Our system uses the microcontroller to communicate with the accelerometer as well as control the animation logic for our art. We represent the LED matrix as a 2D 32x32 chary array on the ATSAM. As the user tilts the accelerometer, the matrix would change to present the current state of the animation. Once the array is updated, the ATSAM sends the data of the matrix to the FPGA via SPI.

Besides initializing SPI and the ATSAM peripherals for use, the first step is to load the array that holds the static image meant to be displayed on the LED matrix. A static image is selected using input from the dip switches. Each image is presented in the code by two 32x32 arrays. One array representing the actual static image being display and another array where the static pixels are represented instead by the character ‘S’. This is to later determine which pixels are eligible to move.

Then the accelerometer needs to be set up for use following the guidelines from HMC E85 Lab 8 [1]. It was configured with the highest conversion rate and resolution [2]. The data from the accelerometer is 4 bytes long and is held in “x” and “y” to represent those axes. Based on the received raw acceleration data, we made some simplified velocity calculations for each axis, which then determine the distance and direction for the moving pixels.

Once the distance and direction are configured, the “animationLogic” takes care of all collision logic and updates the current status of the array. The displacements are held in “row_index” or “col_index” for the new x and y values respectively. There are three arrays used in this module: (1) the newArray, which holds the newest frame and includes ‘S’ values, (2) the currentArray which holds the previous frame sent to the FPGA and includes ‘S’ values, and (3) the sendArray which is the currentArray but with colors values in the place of ‘S’, and this last array is what is sent to the FPGA.

There are a number of checks to pass in order for the pixel to be allowed to move. For example, static and empty pixels do not move. The new location has to be within bounds, or in other words between 0 and 31. Otherwise, the pixel does not move. This means pixels will “stick” to the edge of the matrix as intended. The new spot must be empty (‘E’) for a pixel to occupy it, in both the currentArray (as that represents the old image) and newArray (if a pixel hasn’t occupied that spot already). We then account for the cases if the row is in bounds and the column is not, and vice versa.

After the new matrix representation is produced in newArray, it gets loaded into currentArray. The newArray is also loaded into sendArray, but amended to replace the ‘S’ values with the actual colors so that the FPGA can properly drive the LED matrix.

After the currentArray has been updated, the function “spiLogic” sends the rows of currentArray to the FPGA. This process repeats to continually update the LED as the user tilts the accelerometer.

VI. Results

Our hard work paid off and the project was a success! We were even able to display art multiple images and the user can choose between those images. Looking back, the most difficult aspect of our project was getting the LED matrix properly working and setting up the RAM on the FPGA.

For the matrix, since there were no timing diagrams provided by Adafruit, it was a challenge to display anything on the LED, especially a specific pixel. We initially did not know that the OE pin on the matrix was an active low. We also thought that the LATCH signal should be asserted while the OE signal is de-asserted. This meant that we were always latching the data while displaying on the board which caused color bleeding and random colors being displayed in unexpected rows and columns. Once we realized that the OE is an active low signal (BLANK signal), we were able to output the correct colors in the correct rows, but not the correct columns. Since we thought the signal was actually active high, this led us to believe there was an issue with the hardware. We decided to connect it to an Arduino Uno and test the matrix with the code provided by Adafruit. The uploaded code compiled correctly and outputted that images as expected, which proved that the matrix hardware was working. We then decided to read the waveforms of the signals coming out of the Arduino on the oscilloscope to try to replicate the waveforms. However, that also was not very helpful because the way Adafruit implemented their signals were very different. Our last attempt was to find the datasheet for column shift registers on matrix PCB. Only after going over the chips' datasheets were we able to display a box of a specific color in the specified rows and columns.

The second major challenge we faced was designing RAM. Since we are using 3 bits to represent each color, that means that each row has 96 bits. By reading through that FPGA datasheet, we found that the maximum width for M9K is 36bits, so we tried to map each row on the matrix to three addresses on RAM. That design had success potential but it was a little complicated to implement. After discussing this issue with Professor Harris, we found out that we can write a wider RAM and it will map onto multiple M9Ks. Writing and reading 96 bits from RAM solved our issue very quickly. Therefore we ended up using two RAMs, one for the top half of the matrix and the other for the bottom half, and we were able to read two rows of data simultaneously. After those two major challenges, the other issues were minor in comparison and the team made a lot of progress very quickly. Overall, our team was very happy with the results of the project. Our project had a big window for creativity, which we took to our advantage. We implemented three different artistic images with moving pixels. Those images are shown in figure 6.1.

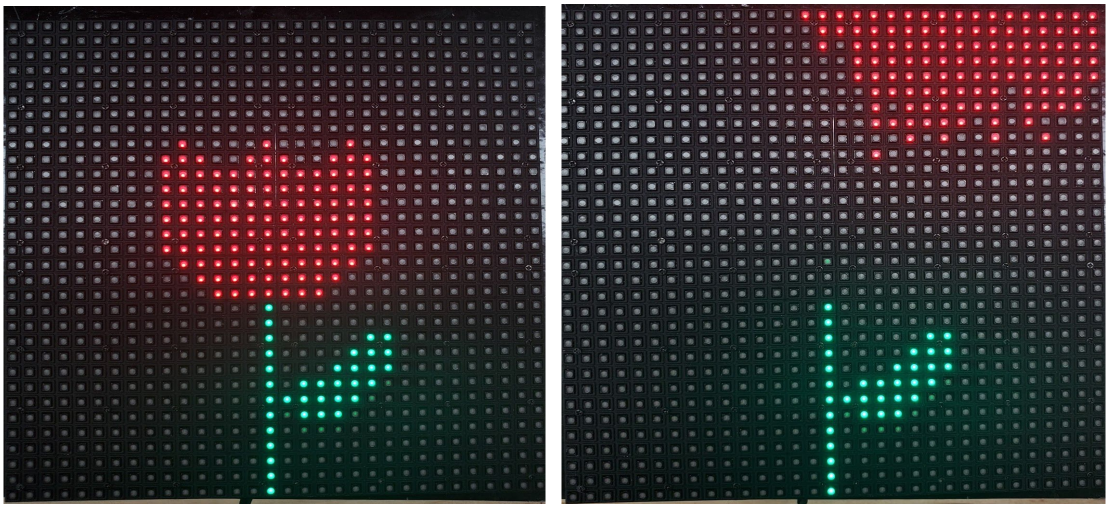

FIG 6.1a: static flower before (left) and after some tilting (right)

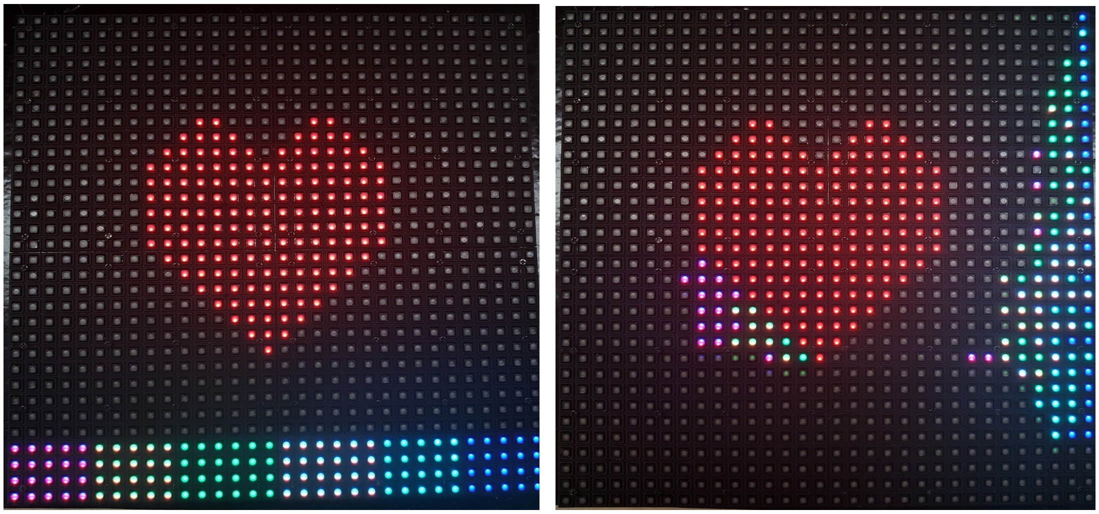

FIG 6.1b: static heart before (left) and after some tilting (right)

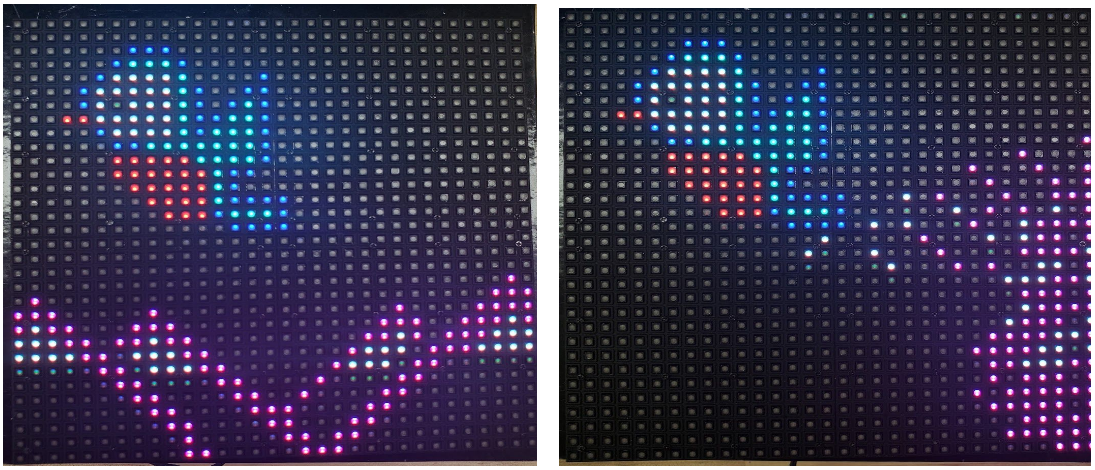

FIG 6.1C: static bird before (left) and after some tilting (right)

VII. References

[1] Harris, David Money. “Lab 8: Digital Level” http://pages.hmc.edu/harris/class/e85/Lab8.pdf

[2] LIS3DH Accelerometer Datasheet. http://pages.hmc.edu/harris/class/e85/LIS3DH.pdf

[3] Macroblock, “16-bit Constant Current LED Sink Driver,” MBI5026 datasheet, 2004.

VIII. Bill of Materials

Part	Source	Product ID #	Price
32 by 32 RGB Board - 4mm pitch	Adafruit	607	\$49.95 + tax + shipping = \$64.86
GenBasic 40 Piece Female to Male Jumper Wires (8 Inch)	Amazon		\$4.99
LIS3DH Motion Sensor	HMC Digital Lab		\$0

IX. Appendix: Code

```c
// E155 Final Project
// Reem Alkhamis  & Sabrine Griffith
/////////////////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "SAM4S4B/SAM4S4B.h"

//////////////////////////////////////////////////////////////////////////////////////////////
// Accelerometer functions

void spiWrite(unsigned char address, unsigned char value){
    spiSendReceive16(address<<8 | value);
}
unsigned char spiRead(unsigned char address){
    return  spiSendReceive16(address<<8 | (1 <<15));
}

// sets the x and y values optianed from the accelerometer to x and y positions
// on the board
int scale(int val) {
     val=val/3000;
   if (val>4){
     return val=4;
   }
   else if (val<-4){
     return val=-4;
   }
   else{
     return val;
  }
}


/////////////////////////////////////////////////////////////////////////////////////////////
// given scale values from the accelerometer:
// the x and y are the pixel position on the pixel grid
// x coordinate (0 to WIDTH-1)
// y coordinate (0 to HEIGHT-1)
////////////////////////////////////////////////////////////////////////////////////////////
// global constants and variables:
#define MATRIX_WIDTH 32
#define MATRIX_HEIGHT 32
#define SPI_READY 6
# define write 7
# define ACCELEN 9
# define dip1 32
# define dip2 33




char currentArray[32][32];
char actualArray[32][32];
char staticArray[32][32];
char newArray[32][32];
char sendArray[32][32];
int row_index;
int col_index;
int v_x;
int v_y;
int ov_x;
int ov_y;






/////////////////////////////////////////////////////////////////////
// different options for images
char staticbirdArray[32][32] = {
```

```c
//    0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16   17   18
19   20   21   22   23   24   25   26   27   28   29   30   31
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 0
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 1
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'S', 'S', 'S', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 2
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'S', 'S', 'S', 'S', 'S', 'K', 'K', 'K', 'K', 'K', 'K'}, // 3
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'K', 'K',
'K', 'K', 'S', 'S', 'S', 'S', 'S', 'S', 'K', 'K', 'K', 'K', 'K'}, // 4
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'S', 'K',
'K', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'K', 'K', 'K', 'K'}, // 5
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'S', 'S',
'K', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'K', 'K', 'K', 'K'},
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'S', 'S',
'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'K', 'K', 'K'}, // 6
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'S', 'S',
'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'K', 'K', 'K', 'K', 'K'}, // 7
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'S', 'S',
'S', 'S', 'S', 'S', 'S', 'S', 'S', 'K', 'K', 'K', 'K', 'K', 'K'}, // 8
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'S', 'S',
'S', 'S', 'S', 'S', 'S', 'S', 'S', 'K', 'K', 'K', 'K', 'K', 'K'}, // 9
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'S',
'S', 'S', 'S', 'S', 'S', 'S', 'S', 'K', 'K', 'K', 'K', 'K', 'K'}, // 10
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S',
'S', 'S', 'S', 'S', 'S', 'S', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 11
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'S', 'S', 'S',
'S', 'S', 'S', 'S', 'S', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 12
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'S', 'S', 'S',
'S', 'S', 'S', 'S', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 13
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'S', 'S', 'S',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 14
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 15
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'},// 16
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 17
    {'K', 'P', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 18
    {'P', 'P', 'P', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'P', 'K'}, // 19
    {'P', 'P', 'P', 'P', 'K', 'K', 'K', 'K', 'P','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'P', 'K', 'K', 'K', 'K', 'K', 'P', 'P', 'P'}, // 20
    {'P', 'W', 'P', 'P', 'P', 'K', 'K', 'P', 'P', 'P', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'P', 'P', 'P', 'K', 'K', 'K', 'P', 'P', 'W', 'P'}, // 21
    {'W', 'W', 'W', 'P', 'P', 'P', 'K', 'P', 'W', 'P', 'P', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'P', 'P', 'W', 'P', 'P', 'K', 'P', 'P', 'W', 'W', 'W'}, // 23
    {'W', 'W', 'W', 'W', 'P', 'P', 'P', 'K', 'W', 'W', 'W', 'P', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'P', 'P', 'W', 'W', 'W', 'K', 'P', 'P', 'W', 'W', 'W', 'W'}, // 24
    {'K', 'K', 'K', 'K', 'K', 'K', 'P', 'P', 'K', 'W', 'W', 'W', 'P', 'K', 'K', 'K', 'K', 'K', 'K',
'P', 'P', 'W', 'W', 'W', 'K', 'P', 'P', 'K', 'K', 'K', 'K', 'K'}, // 25
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'P', 'P', 'K', 'K', 'K', 'K', 'P', 'K', 'K', 'K', 'K', 'P',
'P', 'K', 'K', 'K', 'K', 'P', 'P', 'K', 'K', 'K', 'K', 'K', 'K'}, // 26
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'P', 'P', 'K', 'K', 'K', 'K', 'P', 'K', 'K', 'P', 'P',
'K', 'K', 'K', 'K', 'P', 'P', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 27
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'P', 'P', 'K', 'K', 'K', 'K', 'P', 'P', 'P', 'K',
'K', 'K', 'K', 'P', 'P', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'} , // 28
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'P', 'P', 'K', 'K', 'K', 'P', 'P', 'K', 'K',
'K', 'K', 'P', 'P', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 29
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'P', 'P', 'K', 'P', 'P', 'K', 'K', 'K',
'K', 'K', 'P', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'} , // 30
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'P', 'P', 'P', 'K', 'K', 'K', 'K',
'K', 'P', 'P', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}  // 31

};
char birdArray[32][32] = {
//    0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16   17   18
19   20   21   22   23   24   25   26   27   28   29   30   31
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
```

```c
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 0
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 1
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'B', 'B', 'B', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 2
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'B', 'C', 'C', 'C', 'B', 'K', 'K', 'K', 'K', 'K', 'K'}, // 3
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'B', 'K', 'K', 'K', 'K', 'C', 'W', 'W', 'W', 'W', 'B', 'K', 'K', 'K', 'K', 'K'}, // 4
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'B', 'B', 'K', 'K', 'B', 'C', 'W', 'W', 'W', 'W', 'W', 'B', 'K', 'K', 'K', 'K'}, // 5
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'B', 'C', 'B', 'K', 'B', 'C', 'W', 'W', 'W', 'K', 'W', 'B', 'K', 'K', 'K', 'K'},
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'B', 'C', 'C', 'B', 'B', 'C', 'W', 'W', 'W', 'W', 'W', 'R', 'R', 'K', 'K', 'K'}, // 6
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'B', 'C', 'C', 'C', 'B', 'C', 'W', 'W', 'W', 'W', 'B', 'K', 'K', 'K', 'K', 'K'}, // 7
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'B', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'B', 'B', 'K', 'K', 'K', 'K', 'K', 'K'}, // 8
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'B', 'C', 'C', 'C', 'C', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K'}, // 9
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'B', 'B', 'C', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K'}, // 10
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'B', 'C', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 11
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'B', 'B', 'B', 'B', 'C', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 12
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'B', 'C', 'C', 'C', 'B', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 13
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'B', 'B', 'B', 'B', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 14
        {'B', 'B', 'B', 'B', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 15
        {'B', 'B', 'B', 'B', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'},// 16
        {'B', 'B', 'B', 'B', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 17
        {'B', 'B', 'B', 'B', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 18
        {'B', 'B', 'B', 'B', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 19
        {'W', 'W', 'W', 'W', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 20
        {'W', 'W', 'W', 'W', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 21
        {'W', 'W', 'W', 'W', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 23
        {'W', 'W', 'W', 'W', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 24
        {'W', 'W', 'W', 'W', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 25
        {'G', 'G', 'G', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 26
        {'G', 'G', 'G', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 27
        {'G', 'G', 'G', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'} , // 28
        {'G', 'G', 'G', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 29
        {'G', 'G', 'G', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'} , // 30
        {'G', 'G', 'G', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}  // 31

};


char staticheartArray[32][32] = {
//    0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16   17   18   19   20   21   22   23   24   25   26   27   28   29   30   31
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 0
```

```
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 1
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 2
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 3
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 4
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 5
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'},
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',     'K', 'K', 'K', 'S', 'S', 'K', 'K', 'K', 'K', 'K',
'S', 'S', 'K', 'K', 'K',  'K',      'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 6
    {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',    'K', 'K', 'S', 'S', 'S', 'S', 'K', 'K', 'K', 'S',
'S', 'S', 'S', 'S', 'K',  'K',      'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 7
    {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',    'K', 'S', 'S', 'S', 'S', 'S', 'S', 'K', 'S', 'S',
'S', 'S', 'S', 'S', 'K',  'K',      'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 8
    {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',    'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S',
'S', 'S', 'S', 'S', 'S',  'K',      'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 9
    {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',    'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S',
'S', 'S', 'S', 'S', 'S',  'K',      'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 10
    {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',    'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S',
'S', 'S', 'S', 'S', 'S',  'K',      'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 11
    {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',    'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S',
'S', 'S', 'S', 'S', 'S',  'K',      'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 12
    {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',    'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S',
'S', 'S', 'S', 'S', 'S',  'K',      'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 13
    {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',    'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S',
'S', 'S', 'S', 'S', 'S',  'K',      'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 14
    {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',    'K', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S',
'S', 'S', 'S', 'S', 'K',  'K',      'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 15
    {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',    'K', 'K', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S',
'S', 'S', 'S', 'K', 'K',  'K',      'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 16
    {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',    'K', 'K', 'K', 'S', 'S', 'S', 'S', 'S', 'S', 'S',
'S', 'S', 'K', 'K', 'K',  'K',      'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 17
    {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',    'K', 'K', 'K', 'K', 'S', 'S', 'S', 'S', 'S', 'S',
'S', 'K', 'K', 'K', 'K',  'K',      'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 18
    {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',    'K', 'K', 'K', 'K', 'K', 'S', 'S', 'S', 'S', 'S',
'K', 'K', 'K', 'K', 'K',  'K',      'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 19
    {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',    'K', 'K', 'K', 'K', 'K', 'K', 'S', 'S', 'S', 'K',
'K', 'K', 'K', 'K', 'K',  'K',      'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 20
    {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',    'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'K', 'K',
'K', 'K', 'K', 'K', 'K',  'K',      'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 21

    // 22
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 23
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 24
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 25

    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 26
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',
'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 27
    // BLACK                    // CYAN                    // RED                        //
GREEN                    // YELLOW                  // PURPLE
    {'B', 'B', 'B', 'B', 'B', 'C', 'C', 'C', 'C', 'C', 'W', 'W', 'W', 'W', 'W', 'W', 'G', 'G', 'G',
'G', 'G', 'G', 'Y', 'Y', 'Y', 'Y', 'Y', 'P', 'P', 'P', 'P', 'P'}, // 28
    {'B', 'B', 'B', 'B', 'B', 'C', 'C', 'C', 'C', 'C', 'W', 'W', 'W', 'W', 'W', 'W', 'G', 'G', 'G',
'G', 'G', 'G', 'Y', 'Y', 'Y', 'Y', 'Y', 'P', 'P', 'P', 'P', 'P'}, // 29
    {'B', 'B', 'B', 'B', 'B', 'C', 'C', 'C', 'C', 'C', 'W', 'W', 'W', 'W', 'W', 'W', 'G', 'G', 'G',
'G', 'G', 'G', 'Y', 'Y', 'Y', 'Y', 'Y', 'P', 'P', 'P', 'P', 'P'}, // 30
    {'B', 'B', 'B', 'B', 'B', 'C', 'C', 'C', 'C', 'C', 'W', 'W', 'W', 'W', 'W', 'W', 'G', 'G', 'G',
'G', 'G', 'G', 'Y', 'Y', 'Y', 'Y', 'Y', 'P', 'P', 'P', 'P', 'P'}  // 31

};
char heartArray[32][32] = {
//    0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16   17   18
19   20   21   22   23   24   25   26   27   28   29   30   31
```

```c
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 0
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 1
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 2
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 3
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 4
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 5
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'},
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',     'K', 'K', 'K', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'K', 'K', 'K',  'K',     'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 6
        {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',     'K', 'K', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'K', 'K',  'K',     'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 7
        {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',     'K', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'K',  'K',     'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 8
        {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',     'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R',  'K',     'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 9
        {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',     'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R',  'K',     'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 10
        {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',     'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R',  'K',     'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 11
        {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',     'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R',  'K',     'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 12
        {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',     'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R',  'K',     'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 13
        {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',     'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R',  'K',     'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 14
        {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',     'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K',  'K',     'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 15
        {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',     'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K',  'K',     'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 16
        {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',     'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K',  'K',     'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 17
        {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',     'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K',  'K',     'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 18
        {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',     'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K',  'K',     'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 19
        {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',     'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K',  'K',     'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 20
        {'K','K', 'K', 'K', 'K', 'K', 'K', 'K', 'K',     'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K',  'K',     'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 21

        // 22
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 23
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 24
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 25

        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 26
        {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 27
        // BLACK                    // CYAN                     // RED                         // GREEN                     // YELLOW                   // PURPLE
        {'B', 'B', 'B', 'B', 'B', 'C', 'C', 'C', 'C', 'C', 'W', 'W', 'W', 'W', 'W', 'W', 'G', 'G', 'G', 'G', 'G', 'G', 'Y', 'Y', 'Y', 'Y', 'Y', 'P', 'P', 'P', 'P', 'P'}, // 28
        {'B', 'B', 'B', 'B', 'B', 'C', 'C', 'C', 'C', 'C', 'W', 'W', 'W', 'W', 'W', 'W', 'G', 'G', 'G', 'G', 'G', 'G', 'Y', 'Y', 'Y', 'Y', 'Y', 'P', 'P', 'P', 'P', 'P'}, // 29
        {'B', 'B', 'B', 'B', 'B', 'C', 'C', 'C', 'C', 'C', 'W', 'W', 'W', 'W', 'W', 'W', 'G', 'G', 'G', 'G', 'G', 'G', 'Y', 'Y', 'Y', 'Y', 'Y', 'P', 'P', 'P', 'P', 'P'}, // 30
        {'B', 'B', 'B', 'B', 'B', 'C', 'C', 'C', 'C', 'C', 'W', 'W', 'W', 'W', 'W', 'W', 'G', 'G', 'G', 'G', 'G', 'G', 'Y', 'Y', 'Y', 'Y', 'Y', 'P', 'P', 'P', 'P', 'P'}  // 31

    };

```

```c

char flowerArray[32][32] = {

//     0     1     2     3     4     5     6     7     8     9     10    11    12    13    14    15    16    17    18    19    20    21    22    23    24    25    26    27    28    29    30    31
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 0
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 1
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 2
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 3
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 4
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 5
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'},
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 6
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'K', 'K', 'K', 'K', 'R', 'K', 'K', 'K', 'K', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 7
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'K', 'K', 'R', 'R', 'R', 'K', 'K', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 8
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 9
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 10
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 11
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 12
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 13
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 14
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 15
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'},// 16
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 17
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 18
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 19
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'G', 'G', 'K', 'K', 'K', 'K', 'K', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 20
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'G', 'G', 'G', 'K', 'K', 'K', 'K', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 21
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'G', 'G', 'G', 'G', 'K', 'K', 'K', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 23
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'G', 'G', 'G', 'G', 'G', 'K', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 24
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'G', 'G', 'G', 'G', 'G', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 25
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'G', 'G', 'G', 'K', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 26
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 27
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 28
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 29
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 30
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'G', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}  // 31

};

```

```c
char staticflowerArray[32][32] = {

//    0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16   17   18   19   20   21   22   23   24   25   26   27   28   29   30   31
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 0
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 1
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 2
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 3
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 4
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 5
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'},
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 6
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'K', 'K', 'K', 'K', 'R', 'K', 'K', 'K', 'K', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 7
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'K', 'K', 'R', 'R', 'R', 'K', 'K', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 8
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 9
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 10
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 11
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 12
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 13
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 14
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 15
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'},// 16
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 17
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 18
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 19
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'S', 'K', 'K', 'K', 'K', 'K', 'S', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 20
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'S', 'S', 'K', 'K', 'K', 'K', 'S', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 21
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'S', 'S', 'S', 'K', 'K', 'K', 'S', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 23
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'S', 'S', 'S', 'S', 'K', 'S', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 24
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'S', 'S', 'S', 'S', 'S', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 25
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'S', 'S', 'K', 'S', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 26
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 27
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 28
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 29
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 30
    {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'S', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}  // 31

};

// This function converts character to respective encoding to send to the FPGA.
```

```c
char colorToChar(char color) {
                           //rgb
                if      (color  == 'K') return 0b000;  // black
                else if (color  == 'B') return 0b001;  // blue
                else if (color  == 'G') return 0b010;  // green
                else if (color  == 'C') return 0b011;  // cyan
                else if (color  == 'R') return 0b100;  // red
                else if (color  == 'P') return 0b101;  // purple
                else if (color  == 'Y') return 0b110;  // yellow
                else if (color  == 'W') return 0b111;  // white
                else return 0b0000;

};



// This function assigns arrays to each other.
void assignArray(char giveArray[32][32], char receiveArray[32][32]) {

  for (int i = 0; i < MATRIX_HEIGHT; i++) {
    for (int j = 0; j< MATRIX_WIDTH; j++) {
      receiveArray[i][j] = giveArray[i][j];
  }

}
  }


// This function clears the old array so that new iteration based on animation logic can be loaded.
void clear() {

  for (int i = 0; i < MATRIX_HEIGHT; i++) {
    for (int j = 0; j< MATRIX_WIDTH; j++) {
      if (staticArray[i][j] == 'S'){
        newArray[i][j] = 'S';
      } else {
          newArray[i][j] = 'E';

    }
  }
}
  }
// This function bounds a value to be between 0 and 31, or in other words the row and column slots available on the LED matrix.
int bound (int val){
  if (val<0)
    return 0;
  else if (val>31)
    return 31;
  else
    return val;
}


// migh need to make this take two arguements
bool inBound(int val) {
    bound(val);
   return (val >= 0 && val < 32);
}


// This function controls the animation logic to move pixels on the LED matrix based on data from the accelerometer, obtained through SPI.
void animationLogic() {
      int index_i = row_index;
      int index_j = col_index;

      clear();
      for (int i = 0 ; i < MATRIX_HEIGHT; i++) {
        for (int j = 0; j<MATRIX_WIDTH; j++) {
                int bool_i = inBound(i+index_i);
                int bool_j = inBound(j+index_j);
```

```c
          // If the image is in a "static" spot, it shouldn't move.
          if (currentArray[i][j] == 'S') {
            newArray[i][j] = currentArray[i][j];

          // If the next pixel is "static", and the current spot holds some color,
          // keep the static spot as is and hold the current color in its current position.
          } else if (currentArray[i+index_i][j+index_j] == 'S'&& currentArray[i][j] != 'E') {
            newArray[i+index_i][j+index_j] = 'S';
            newArray[i][j] = currentArray[i][j];
          }
          // If the current pixel is colored, not empty, and not static, then enter the if statements to check bounds.
          else if (currentArray[i][j] != 'K' && currentArray[i][j] != 'S'&& currentArray[i][j] != 'E'){
            // Checks if row of next pixel is in bound
            if (bool_i) { // Row is in bound
                // Column is in bound
                if (bool_j) {
                    if (currentArray[i+index_i][j+index_j] == 'E' && newArray[i+index_i][j+index_j] == 'E') {
                    newArray[i+index_i][j+index_j] = currentArray[i][j];
                    }
                  // Column is in bound but the spot is not empty
                  else if (currentArray[i+index_i][j+index_j] != 'E'  || newArray[i+index_i][j+index_j] != 'E') {
                    newArray[i][j] = currentArray[i][j];
                  // Column is out of bound
                }
              }   // Row is in bound, but column is out of bound, spot is empty
                else if (currentArray[i+index_i][j] == 'E' && newArray[i+index_i][j] == 'E' ) {
                  newArray[i+index_i][j] = currentArray[i][j];
                  // Row is in bound, but column is out of bound, spot is not empty
                } else if (currentArray[i+index_i][j] != 'E' || newArray[i+index_i][j] != 'E') {
                    newArray[i][j] = currentArray[i][j];
                }
              }
              // Row is out of bound, but column is in bound.
             else if (bool_j){
                    // Next spot is empty.
                  if (currentArray[i][j+index_j] == 'E'&& newArray[i+index_i][j+index_j] == 'E') {
                        newArray[i][j+index_j] = currentArray[i][j];
                    // Next spot is not empty.
                  } else if (currentArray[i][j+index_j] != 'E'|| newArray[i+index_i][j+index_j] != 'E') {
                      newArray[i][j] = currentArray[i][j];
                  }
                }
                //If none of above applies, hold position.
              else{
                newArray[i][j] = currentArray[i][j];
          }
      }

    }
  }

      // Assigning the new array to current array, taking into consideraion static pixels.

        for (int  i = 0; i < MATRIX_HEIGHT; i++) {
          for (int j = 0; j< MATRIX_WIDTH; j++) {
            currentArray[i][j] = newArray[i][j];
            if (newArray[i][j] == 'S'){
              sendArray[i][j] = actualArray[i][j];
            } else {
            sendArray[i][j] = newArray[i][j];

            }
          }
        }


}

```

```c

// This function sends the new RGB data for each row of the LED matrix through SPI to the FPGA.
void spiLogic() {
  for (uint16_t i = 0; i < MATRIX_HEIGHT; i++) {
      pioDigitalWrite(SPI_READY, PIO_HIGH);
      pioDigitalWrite(write, PIO_LOW);
      spiSendReceive(i);
      for (int k =0; k < 8; k++) {
            int index = k*4;
            uint16_t rowdata = colorToChar(sendArray[i][index]) << 9 | colorToChar(sendArray[i][index+1]) << 6 |
                                colorToChar(sendArray[i][index+2]) << 3 | colorToChar(sendArray[i][index+3]);
            spiSendReceive(rowdata);
        }
        pioDigitalWrite(SPI_READY, PIO_LOW);
        pioDigitalWrite(write, PIO_HIGH);
        tcDelayMicroseconds(1000);
        pioDigitalWrite(write, PIO_LOW);

  }
}

// This function takes in data from the accelerometer to get "velocities" to use to control the animation.
void speed(volatile short disx, volatile short disy) {
    ov_x = v_x;
    ov_y = v_y;
    v_x = -disx;
    v_y = -disy;

}

// This function determines how much each pixel should increment in the x direction, or in other words the displacement.
void dispX(volatile short v_x, volatile short ov_x ) {

     row_index = (v_x+ov_x)/2;
}


// This function determines how much each pixel should increment in the y direction, or in other words the displacement.
void dispY(volatile short v_y, volatile short ov_y ) {

     col_index = (v_y+ov_y)/2;
}

// Main
int main(void){
  volatile unsigned char debug;
  volatile short disx, disy;
  volatile short x,y, a, b, c, d;

  pioPinMode(ACCELEN, PIO_OUTPUT);
  pioPinMode(write, PIO_OUTPUT);
  pioPinMode(SPI_READY, PIO_OUTPUT);
  pioPinMode(dip1, PIO_INPUT);
  pioPinMode(dip2, PIO_INPUT);

  samInit();
  pioInit();
  // the phase for the SPI clock is 1 and the polarity is 0
  spiInit(MCK_FREQ/244000, 0, 1);
  tcDelayInit();


  // This determines which image to display based on the DIP switch. The default is the heart image.
  if (pioDigitalRead(dip1)){
    assignArray(staticbirdArray, staticArray);
    assignArray(birdArray, actualArray);
    }
```

```c
  else if (pioDigitalRead(dip2)){
    assignArray(staticflowerArray, staticArray);
    assignArray(flowerArray, actualArray);
  }
  else{
    assignArray(staticheartArray, staticArray);
    assignArray(heartArray, actualArray);
    }
  // Load the chosen image that will be iterated on later with animation logic.
  assignArray(staticArray, currentArray);

  // The following code initializes the accelerometer.
  pioDigitalWrite(ACCELEN, 0);
  spiWrite(0x20, 0x77); // highest conversion rate
  pioDigitalWrite(ACCELEN, 1);
  pioDigitalWrite(ACCELEN, 0);
  spiWrite(0x23, 0x88); // block update and high resolution
  pioDigitalWrite(ACCELEN, 1);

  //Read from WHO_AM_I register, should be 0x33.
  pioDigitalWrite(ACCELEN, 0);
  debug = spiRead(0x0F);
  pioDigitalWrite(ACCELEN, 1);

// Enter the while loop to begin iterating.
  while(1){
    // The following commands read tilt information from the accelerometer.
        pioDigitalWrite(ACCELEN, 0);
        a = spiRead(0x28);
        pioDigitalWrite(ACCELEN, 1);
        pioDigitalWrite(ACCELEN, 0);

        b= spiRead(0x29);
        pioDigitalWrite(ACCELEN, 1);
        pioDigitalWrite(ACCELEN, 0);

        c= spiRead(0x2A);
                pioDigitalWrite(ACCELEN, 1);
       pioDigitalWrite(ACCELEN, 0);

       d= spiRead(0x2B);
        pioDigitalWrite(ACCELEN, 1);


         x = a | (b <<8);
         y = c | (d <<8);


    // disx and disy represent scaled versions of the x and y values, as and and y could range from -16000 to 16000.
       disx = scale(x);
       disy = scale(y);

       // Need to set number of bits to send over SPI to 12 to send data to the FPGA using SPI.
       SPI->SPI_CSR0.BITS = 4;
       // Calculate velocities.
       speed(disx, disy);
       // Use those velocities to calculate displacement.
       dispX(v_x, ov_x);
       dispY(v_y, ov_y);
       // Use those displacements to drive animation.
       animationLogic();
       // Send new, iterated matrix to FPGA to drive into LED matrix.
       spiLogic();
       // Set number of bits to send over SPI back to the default of 16 bits in order to communicate using SPI with the accelerometer.
       SPI->SPI_CSR0.BITS = 0;



}

```

```c


}


//// plai black array if needed

//char birdArray[32][32] = {
////    0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16   17   18   19   20   21   22   23   24   25   26   27   28   29   30   31
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 0
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 1
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 2
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 3
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 4
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 5
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'},
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 6
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 7
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 8
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 9
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 10
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 11
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 12
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 13
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 14
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 15
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'},// 16
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 17
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 18
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 19
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 20
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 21
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 23
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 24
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 25
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 26
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 27
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 28
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 29
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}, // 30
```

```
//     {'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K', 'K'}  // 31
//
//};

```

```
// E155 final project
// Reem Alkhamis & Sabrine Griffith
module finalProjectTest(input logic clk,
        input logic reset,               // active low reset
        input logic mosi,                // slave input
        input logic sclk,                // serial clock
        input logic we, spiready,        // write enable and chip enable

         output logic [2:0] RGB1, RGB2, // to the matrix
         output logic BLANK,            // OE: output enable (blanking singal, erases the board)
         output logic LAT,             // LAT: takes data from shift reg to the output register
         output logic [3:0] row,
         output logic matrixclk);

 logic [3:0] slowClk1;

 always_ff@(posedge clk, negedge reset)
  if (!reset) slowClk1 <=0;
  else
   slowClk1 <= slowClk1 + 1'b1;



 slowClkProject project(clk, slowClk1[3], reset, we, spiready, mosi, sclk, RGB1, RGB2, BLANK, LAT, row, matrixclk);

endmodule


module slowClkProject(input logic fastClk,
                     input logic clk,
                      input logic reset,
                      input logic we, spiready,
                     input logic mosi,
                     input logic sclk,
                     output logic [2:0] RGB1, RGB2,
                     output logic BLANK,
                     output logic LAT,
                     output logic [3:0] row,
                     output logic matrixclk);




// pin assignments
// RGB1[2:0]: (RED) R1=RGB1[2], (GREEN) G1=RGB1[1], (BLUE) B1=RBG1[0]
// Rows 31:16
// RGB2[2:0]: (RED) R2=RGB2[2], (GREEN) G2=RGB2[1], (BLUE) B1=2=RBG2[0]
// row[3:0]: D = row[3], C=row[2], B=row[1], A=row[0]
// LAT   -> PIN_42
// OE    -> PIN_43
// D     -> PIN_38
// C     -> PIN_34
// B     -> PIN_33
// A     -> PIN_32
// reset -> PIN_1
// R1    -> PIN_7
// B1    -> PIN_11
// G1    -> PIN_10 // changed to PIN_3
// R2    -> PIN_28
// B2    -> PIN_31
// G2    -> PIN_30
// MarixClk -> PIN_39
/////////////////////////////////////////////////
 logic [5:0] counter;
 logic [95:0] row0, row1;
 logic [95:0] spidata;
 logic [4:0] col;
 logic [3:0] radr;
 logic [4:0] wadr;
 logic hold;


    always_ff@(posedge clk, negedge reset)
```

```
        if (!reset)  begin
         counter<=0;
         col<=0;
         hold <=1;
      end
        else if (counter == 60 || hold==1)
         begin
         col <= 0;
         counter <=0;
         hold<=0;
        end
       else if (counter>=31 && counter<60) begin
        col <=col;
        counter <= counter+ 1'b1;
       end

        else if (hold==0) begin
        counter <= counter + 1'b1;
        col <= col+1'b1;

       end


 //////////
    always_ff@(posedge clk, negedge reset)
      if (!reset)  begin
         row<=0;
    end
      else begin
        if (counter==60 )
            row <= row + 1'b1;
     end


  controller controller(clk, reset, counter, BLANK, LAT);
  memRead memRead(clk, reset, LAT, BLANK, radr);
  // We are using two RAMs in order to display two rows simultaneously.
  dmemtop dmemt(fastClk, reset, we, wadr, radr, spidata,  row0);
  dmembottom dmemb(fastClk, reset, we, wadr, radr, spidata,  row1);

  driveColor color(matrixclk, row0, row1, col, RGB1, RGB2);
  spi spi(fastClk, sclk, reset, we, spiready, mosi, wadr, spidata);




   always_comb
       if (counter <32)
            matrixclk = clk;
       else if (counter==60)
            matrixclk=0;
       else
            matrixclk =1;

endmodule

module controller(input logic clk, reset, // 2.5MHz
                 input logic [5:0] counter,
                 output logic BLANK,
                 output logic LAT);


 always_ff@(posedge clk)
       begin

        if (counter==32) begin
          LAT <= 1;
          BLANK<=1; end
         else if (counter==33) begin
          LAT <= 0;
          BLANK<=0;
          end
        else if (counter>=34 && counter <=59) begin
          LAT <= 0;
          BLANK<=0;
          end
```

```

        else begin
         BLANK <= 1;
         LAT <=0; end
         end

endmodule


module driveColor(input logic matrixclk,
                  input logic [95:0] row0, row1, // read data from RAM
                  input logic [4:0] col,
                  output logic [2:0] RGB1, RGB2);

   always_ff@(negedge matrixclk) begin
       RGB1[0] <= row0[3*col];
       RGB1[1] <= row0[3*col+1];
       RGB1[2] <= row0[3*col+2];

       RGB2[0] <= row1[3*col];
       RGB2[1] <= row1[3*col+1];
       RGB2[2] <= row1[3*col+2];
       end



endmodule
/// we are sending through spi 108 bits every time
// 96 bits of data, 5 bits of address and 7 unused bit
// initilizing memory with the static image in a text file


module memRead(input logic clk,
               input logic reset,
               input logic LAT,
               input logic BLANK,
               output logic [3:0] radr);

      always_ff@(posedge clk, negedge reset) begin // 2.5MHz clk
         if (!reset)
            radr<=0;

         else if (LAT && BLANK)
            radr <= radr+1;



            end

endmodule


module spi(input logic fastclk, sclk, reset,
           input logic we, spiready,
           input logic mosi,
           output logic [4:0] wadr,
           output logic [95:0] spidata);

   logic [107:0] q;

         always_ff@(posedge sclk, negedge reset) begin
            if (!reset) q <=0;
            else if (spiready)
                q <= {q[106:0], mosi};
         end

         always_ff@(posedge fastclk) begin  // writing on the 40MHz clk
            if (we) begin
               wadr<= q[100:96]; // unused 7 bits
               spidata <= q[95:0];
               end
         end

endmodule


```

```
module dmemtop(input logic fastclk, reset, // 40MHz clk
            input logic we,          // write enable
            input logic [4:0] wadr, // write address
            input logic [3:0] radr, // read address
            input logic [95:0] row_data, // write data
            output logic [95:0] row0); // read data


   logic [95:0] RAM[31:0];
   initial
      $readmemb("heart.dat",RAM);

      always_ff@(posedge fastclk)
         row0 <= RAM[radr];



      always_ff@(posedge fastclk)
         if (we)
            RAM[wadr] <= row_data;

endmodule

module dmembottom(input logic fastclk, reset, // 40MHz clk
            input logic we,          // write enable
            input logic [4:0] wadr, // write address
            input logic [3:0] radr, // read address
            input logic [95:0] row_data, // write data
            output logic [95:0]row1); // read data


   logic [95:0] RAM[31:0];
   initial
      $readmemb("heart.dat",RAM);

      always_ff@(posedge fastclk)
         row1 <= RAM[radr+16];



      always_ff@(posedge fastclk)
         if (we)
            RAM[wadr] <= row_data;

endmodule
```