CPE316

John Oliver

F2023: Final Project

Srinivas Sundararaman

Jack Krammer

15 December 2023

Description

doodlestick™ is an STM32L4 embedded application that allows the manipulation of contents on an LED dot matrix display in a multitude of ways:

- Cursor mode (CS)
- Free-draw mode (FD)
- Rect-draw mode (RD/RF)
- Fill mode (FL/FL2)
- Stroke color
- Stroke sensitivity

doodlestick uses a joy-stick to change the current display position, and a keypad for mode selection.

Cursor Mode

CS allows the user to move the current position on the display without drawing over coordinates that the cursor passes through.

Free-Draw Mode

FD allows the user to move the current position on the display *while* drawing over coordinates that the cursor passes through based on the stroke color set. To not draw over coordinates, use CS.

Rect-Draw Mode

RD allows the user to draw rectangles. It initially acts as CS, allowing you to move to any coordinate. By pressing down the joy-stick at two coordinates, a rectangle with a diagonal at said two LED coordinates is rasterized based on the stroke color.
RF is similar to RD, except the rasterized shape is also filled with the stroke color.

Fill Mode

FL allows the user to completely fill the display with a color according to the following keystroke. FL2 is functionally equivalent to CS. However, its implementation is technically more similar to FL. It is recommended to use FL for filling the screen and CS to move the cursor performance-wise, and FL2 for the curious.

Stroke Color

Allows the user to change the current stroke color. It does not affect the previous stroke.

Stroke Sensitivity

Allows the user to change the sensitivity of the joy-stick. In other words, the joy-stick input is read at a faster or slower rate.
Note: As a visual aid, the rate of the blinking cursor is proportional to the stroke sensitivity

Mode Selection

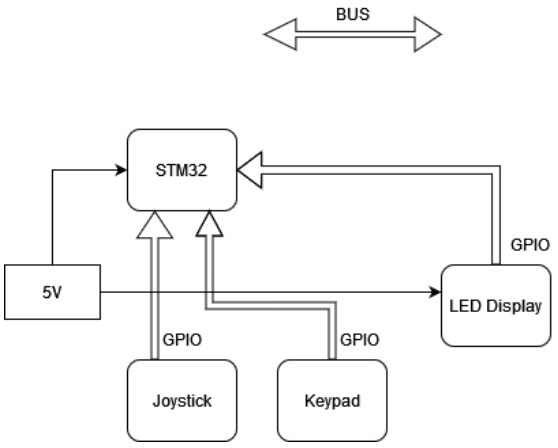
Allows the user to change between the menu types listed above. By having multiple option menus, the amount of user input available is no longer limited to 12 options (one for each button). Instead, each of the four modes uniquely utilize the first eight buttons to achieve a total of thirty-two unique options to select from. As there are many available, some of the options associated with these menus provide a couple matrix presets as easter eggs.

System Specification

Voltage Supply	3.3V via mini-USB
Voltage Operating Range	0 to 3.3 V to 5V (LED display)
System Clock Frequency	32 MHz
Max Current Regulation	4 A
Max Nominal Current Draw	2.5 A
Physical Size (LxWxH)	10x20x6 cm^3

Tb1. System Specs

System Architecture



Fg1. System Specs

This is the System Architecture for doodlestick. The LED matrix display needs more power than the STM32 can provide, so a 5V adapter was used to interface the peripheral. The Joystick and LED Display are both new peripherals.

System Schematics

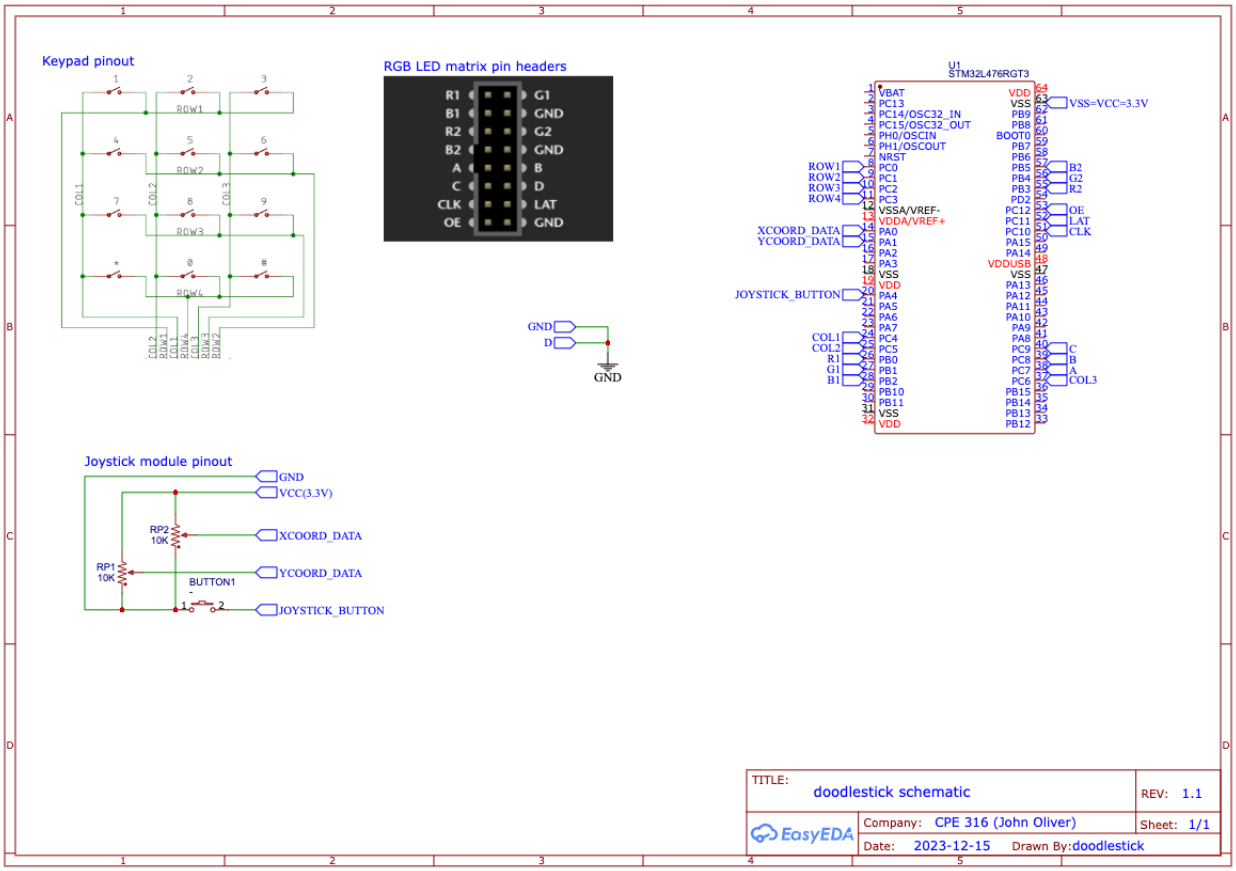


Fig2. doodlestick schematic

This is the schematic for the doodlestick system. The netport labels indicate where the pins are connected. For all the row and col pins of the keypad, standard GPIO input and output modes were used. For the joystick module, separate ADCs were configured on the X and Y coordinate data pins. To keep the voltages inputted to the ADCs within the proper range, the voltage source pin of the joystick was connected to the 3.3V pin. The joystick button was configured as a GPIO input pin with a pull up resistor. For the RGB module, the control pins were configured as standard GPIO outputs and connected as shown above. The power to this module came from a separate 5V 4A DC power source.

Software Architecture

Option 1 (1)	Option 2 (2)	Option 3 (3)
Option 4 (4)	Option 5 (5)	Option 6 (6)
Option 7 (7)	Option 8 (8)	Speed Select Mode (9)
Color Select Mode (*)	Fill Select Mode (0)	Draw Select Mode (#)

Tb2. Keypad button selections

This table represents the keypad layout and the corresponding mode or option that is selected upon that key press. In parenthesis are the values shown on the keypad itself. The table below describes the options available in each mode.

Mode	Color Select (*)	Fill Select (0)	Draw Select (#)	Speed Select (9)
Option 1 (1)	Red	Red	Cursor Mode	Timer PSC = 2
Option 2 (2)	Blue	Blue	Free Draw Mode	Timer PSC = 3
Option 3 (3)	Green	Green	N/A	Timer PSC = 4
Option 4 (4)	Yellow	Yellow	Rect Draw Mode	Timer PSC = 5
Option 5 (5)	Cyan	Cyan	N/A	Timer PSC = 6
Option 6 (6)	Purple	Purple	Smiley Face	Timer PSC = 7
Option 7 (7)	White	White	Hi	Timer PSC = 8
Option 8 (8)	Black (erase)	Black (clear)	Constant Clear	Timer PSC = 9

Tb3. Mode options

This table indicates the option available at each option key for each mode available. This program was designed for one key to be pressed at a time. In order to select the desired option, the mode key must first be pressed, then the desired option key. If the desired option is already within the mode of the previous mode key press, then only the option key press is required to choose that option. The output due to an option selection will not be changed until the next option key press. Pressing one of the mode select keys won't change the current output.

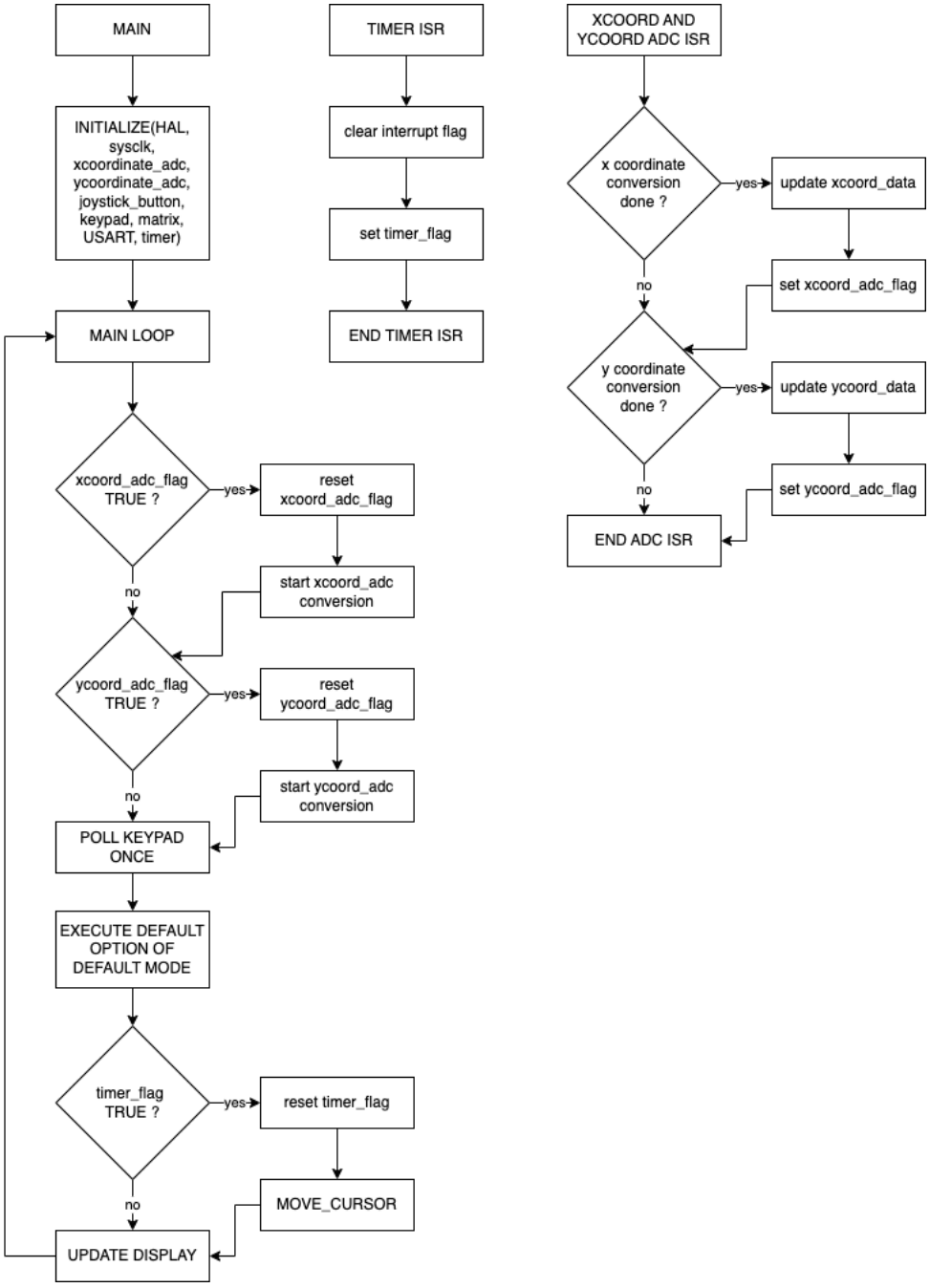


Fig3. Main and ISR software flowcharts

These flowcharts rely on the global variables matrix_buffer, timer_flag, cursor_position, xcoord_adc_flag, xcoord_data, ycoord_adc_flag, ycoord_data. The execute option function block represents the case statement that decides the action performed on the matrix_buffer. Later in the main function, the update display function writes the matrix_buffer to the LED matrix. The timer_flag variable is used to indicate when the cursor position should be updated. The x and y coordinate flags and associated data indicate when another ADC conversion should be started and stores the latest conversion result to be used when moving the cursor.

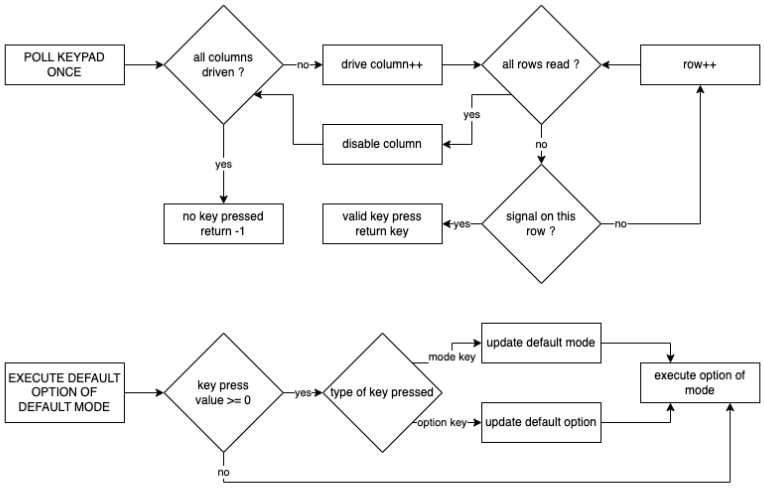


Fig4. Keypad and command execution software flowcharts

The keypad is used to get the user's desired option selection. The return value of -1 when a key isn't pressed indicates that the current selection shouldn't change. Otherwise, when a key is pressed, the associated key value is used to update the mode or option selected. These updated values are then stored and treated as the new defaults until another key is pressed.

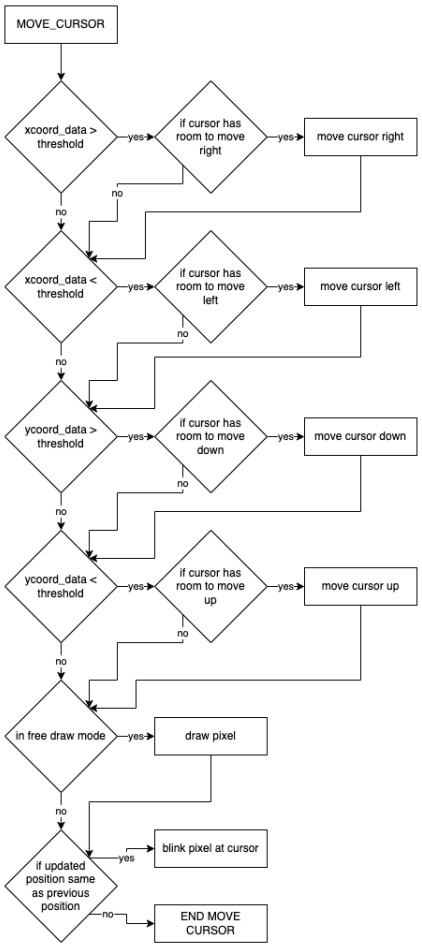


FIg5. Move cursor software flowchart

This function is called every time the timer_flag is set. Due to some variability in the values representing the neutral x and y positions, there is a threshold the user must move the joystick past in order to enact a change in position. This function separately checks the x and y coordinates, enabling diagonal movement. This function also separately checks both directions of both axis directions in order to have the cursor position be as responsive as possible. This function also controls the blinking of the cursor when the cursor position does not change. A static variable is incremented each time this function is accessed and changes the blink state every multiple of the blink threshold defined in main. This ensures the blink can happen at a rate slow enough for the eye to see.

Bill of Materials

Keypad 12-Button COM-14662	4.95
STM32L476RG	24.50
Joystick	2.25
Jumper Wires	4.00
LED Display Matrix	24.95
5V 4A Switching Power Supply	14.95
Total	**75.60**

Tb4. BOM

The bill of materials indicates the cost of each item used in this project.

Ethical Implications

One ethical implication needed to be considered is the extra current that the rather large LED dot matrix needs in order to keep color values on the screen, or else the LEDs won't get driven. This means there's extra power consumption to consider, and whether this application is worth enough to outweigh the miniscule but non-trivial increase in the user's footprint. Another consideration is the tradeoff between ease of programming and testing over efficiency and less power usage. For example, this sys ended up using polling for a vast majority of the I/O in order to function. However, extra checks, reads and writes that are unnecessary also increases the energy consumption, which has the same problem as the last ethical implication.

Appendix

main.c

```
/*
* JACK and SRINI
*
* CPE 316 Final Project
*           John Oliver
*
*
* DOODLESTICK
*                   doodle with a joystick
*                   draws on a LED matrix
*
*
*                   DEPENDENCIES
*
*
*/
// includes
#include "main.h"
#include <stdio.h>
#include "joystick.h"
#include "uart.h"
#include "keypad_12.h"
#include "rgb_matrix.h"
#include "timer2.h"
// defines
#define X_NEUTRAL 2050
#define Y_NEUTRAL 1950
#define THRESHOLD  900
#define BLINK_THRESHOLD 5
#define BUFF_SIZE 16
// typedefs
typedef enum KP_MODE {
                    COLOR  = 0xA,
                    FILL     = 0x0,
                    DRAW  = 0xB,
                    SPEED   = 0x9  // do we need thickness ? speed instead
} KP_MODE;
typedef struct point{
          int8_t x, y;
} point;
// function declarations
void TIM2_IRQHandler(void); // interrupt handler for TIM2
void move_cursor(); // moves the cursor in the direction indicated by the joystick
uint8_t check_color(point pt, color c); // returns 1 if color at point in matrix is same as input, returns 0 if not
void int_to_str(int num, char* buff); // converts and int and returns a string of length BUFF_SIZE
void USART_print_int(int num); // prints an int to USART
```

```
void reset_shapes(point line[2], point square[2], point triangle[3]); // resets the shapes to their defaults, coordinates = -1
void reset_this_shape(point* shape, int size); // resets the vars of this shape
void add_pt_to_shape(point* shape, uint8_t size); // adds the current cursor position to the shape
int8_t get_shape_index(point* shape, int size); // returns -1 or the index of the shape that isn't set yet
uint8_t get_max(uint8_t a, uint8_t b); // gets the minimum of two uint8_t values
uint8_t get_min(uint8_t a, uint8_t b); // gets the maximum of two uint8_t values
void draw_horizontal_line(uint8_t xmin, uint8_t xmax, uint8_t yconst); // draws a horizontal line given the xmin, xmax and y values
void draw_vertical_line(uint8_t ymin, uint8_t ymax, uint8_t xconst); // draws a vertical line given ymin, ymax, and x values
void draw_rect(point p1, point p2); // draws a rectangle with the two points as opposite corners
void draw_shape(point* shape, int size); // draws the given shape
uint8_t same_point(point p1, point p2); // returns 1 if the points have the same coordinates and 0 if they don't
uint8_t pt_inbounds(point pt); // returns 1 if the point is within the bounds of the matrix and 0 if not
// colors
const color RED   = {.r = 1, .g = 0, .b = 0};
const color GREEN         = {.r = 0, .g = 1, .b = 0};
const color BLUE          = {.r = 0, .g = 0, .b = 1};
const color WHITE   = {.r = 1, .g = 1, .b = 1};
const color BLACK         = {.r = 0, .g = 0, .b = 0};
const color PURPLE        = {.r = 1, .g = 0, .b = 1};
const color YELLOW        = {.r = 1, .g = 1, .b = 0};
const color CYAN          = {.r = 0, .g = 1, .b = 1};
//const color NONE        = {.r = 2, .g = 2, .b = 2};
/*
* matrix buffer is indexed as [x][y]
*
* convention (x,y):
*         top left is (0,0)
*         bottom right is (31,15)
*
*/
color matrix_buffer[NUM_COLS][NUM_ROWS];
// cursor variables
point     cursor_pos                    = {.x = 0, .y = 0}; // cursor starts at top lef corner
point     prev_pos              = {.x = 0, .y = 0};         // holds the previous position of the cursor
color     prev_color                    = RED;                    // holds the previous color when blinking
uint8_t cursor_thickness    = 1;                              // default cursor thickness is radius of 1
color     draw_color                    = RED;                          // need to check whether the color is NONE or not before drawing anything
uint8_t drawing                         = 0;                            // 1 = trace mode, 0 = don't trace joystick movement
// more variables
volatile uint16_t  xcoord_data          = X_NEUTRAL;
volatile uint16_t  ycoord_data          = Y_NEUTRAL;
volatile uint8_t   xcoord_adc_flag  = 1;
volatile uint8_t   ycoord_adc_flag  = 1;
//uint8_t enable_serial = 0;            // if 1 then send updates over UART, essentially is debugging
volatile uint8_t   timer_flag                   = 1;
```

```
                uint8_t  button_flag                    = 0;
int main()
{
        /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
        HAL_Init();
        // initialize the joystick
        SystemClock_Config();       // sets system clock to 32MHz
        xcoord_adc_init();          // initializes the ADC for the joystick x data
        ycoord_adc_init();          // initializes the ADC for the joystick y data
        joystick_button_init();     // initializes the button on the joystick
        // initializes the keypad
        keypad_init();
        // initializes the matrix
        matrix_init();                  // initializes the matrix pins
        matrix_begin();                 // sets the initial matrix pin values
        // sets the initial display
        make_smiley(CYAN); // make_hi(PURPLE);
        // initializes the USART serial output
        USART_init();
        // sets the initial serial output
        USART_Print("doodlestick\n\r");
        USART_Print("    by Jack and Srini\n\n\r");
        // initialize the speed timer
        TIM2_init(320000, 9, 0xFFFFFFFF); // initializes timer with values (arr, psc, ccr1)
        // variables
        KP_MODE kp_mode = DRAW;
        int8_t kp_select = 2;
        int8_t kp_ret = -1;
        uint8_t button_ret = 0;
        color colors[8] = { RED,    GREEN,          BLUE,
                                                    YELLOW, CYAN,           PURPLE,
                                                    WHITE,          BLACK };
        // shape variables
        point line[2]                   = { {.x = -1, .y = -1}, {.x = -1, .y = -1} };
        point square[2]     = { {.x = -1, .y = -1}, {.x = -1, .y = -1} };
        point triangle[3]   = { {.x = -1, .y = -1}, {.x = -1, .y = -1} , {.x = -1, .y = -1} };
        while(1)
        {
                // update joystick values, this could be more efficient if does this via a timer
                // update x coordinate joystick value
                if(xcoord_adc_flag)
                {
                        // reset flag
                        xcoord_adc_flag = 0;
                        // start another conversion
                        xcoord_start_conversion();
                }
                // update y coordinate joystick value
                if(ycoord_adc_flag)
```

```
{
        // reset flag
        ycoord_adc_flag = 0;
        // start another conversion
        ycoord_start_conversion();
}
// poll keypad, store as value
kp_ret = loop_keypad_once(); // gets index of keypad
if(kp_ret >= 0) // valid command
{
        // get the value on the button instead of the index
        kp_ret = keypad_vals[kp_ret];
        // check if keypad press was a MODE configuration
        if(kp_ret > 8 || kp_ret == 0)
        {
                switch(kp_ret)
                {
                case 0xA:               // * = COLOR SELECT MODE
                        kp_mode = COLOR;
                        kp_select = -1; // default is no selection
                        break;
                case 0x0:               // 0 = FILL SELECT MODE
                        kp_mode = FILL;
                        kp_select = -1; // default is no selection
                        break;
                case 0xB:               // # = DRAW SELECT MODE
                        kp_mode = DRAW;
                        reset_shapes(line, square, triangle);
                        kp_select = -1; // default is no selection
                        break;
                case 0x9:               // 9 = SPEED SELECT MODE
                        kp_mode = SPEED;
                        kp_select = -1; // default is no selection
                        break;
                default:  // else error, don't change anything
                        break;
                }
        }
        else // keypad press changes the kp_select option
        {
                kp_select = kp_ret;
        }
}
// execute command chosen
switch(kp_mode)
{
case COLOR:                     // COLOR mode = changes color of draw tool
        if(kp_select >= 0 && kp_select < 9)
        {
```

```
                draw_color = colors[kp_select - 1];
            }
            break;
        case FILL:                  // FILL mode       = allows a decision on what color to fill the board with
            if(kp_select >= 0 && kp_select < 9)
            {
                fill_matrix(colors[kp_select - 1]);
            }
            kp_select = -1;
            break;
        case DRAW:                  // DRAW mode    = changes type of draw tool
            switch(kp_select)
            {
            case 1:    // free     = don't change matrix
                drawing = 0;
                break;
            case 2: // trace       = draw where moving
                drawing = 1;
                break;
            case 3: // line                 = can click twice and draw a line between the two spots clicked
                drawing = 0;
                if(button_flag)
                {
                    add_pt_to_shape(line, 2);
                    // reset button flag
                    button_flag = 0;
                }
                break;
            case 4: // square    = can click twice and draw a square with two spots clicked as opposite
corners
                drawing = 0;
                if(button_flag)
                {
                    add_pt_to_shape(line, 4);
                    // reset button flag
                    button_flag = 0;
                }
                break;
            case 5: // triangle = can click three times and draw triangle with 3 spots clicked as vertices
                drawing = 0;
                if(button_flag)
                {
                    add_pt_to_shape(line, 3);
                    // reset button flag
                    button_flag = 0;
                }
                break;
            case 6: // thick      = cursor now 1 dot // now is going to be the cyan w smiley relief
                make_smiley(CYAN); // turn into logo
```

```
                                        break;
                                case 7: // thiick     = cursor now 2 dots radius thickness // now is purple hi or cyan smiley
                                        if(xcoord_data % 2) make_hi(PURPLE);
                                        else make_smiley(CYAN);
                                        kp_select = -1;
                                        break;
                                case 8: // thiiick    = cursor now 3 dots radius thickness // now is going to be a continous fill
of blank
                                        fill_matrix(colors[kp_select - 1]);
                                        break;
                                default:  // error in option selected, do nothing
                                        break;
                                }
                                break;
                        case SPEED:                     // SPEED mode    = changes speed of cursor movement
                                TIM2->PSC = kp_select + 1; // larger number = slower speed
                                break;
                        default:                // invalid mode, do nothing
                                break;
                        }
                        // move cursor
                        if(timer_flag)
                        {
                                move_cursor();
                                timer_flag = 0;
//                              // print status
//                              char buff[BUFF_SIZE];
//                              USART_Print("    mode = ");
//                              USART_print_int(kp_mode);
//                              USART_Print("    select = ");
//                              USART_print_int(kp_select);
//                              USART_Print("\n\r");
                        }
                        // update the display
                        update_display();
                }
                return 0;
}
/* ------------------ INTERRUPT FUNCTIONS ------------------ */
// interrupt handler for TIM2
void TIM2_IRQHandler(void)
{
        // if from TIM2 update event
        if(TIM2->SR & TIM_SR_UIF) // from ARR
        {
                TIM2->SR &= ~TIM_SR_UIF;        // reset interrupt flag
                timer_flag = 1;                                 // set global timer flag
        }
}
```

```
// clear flag and set globals
void ADC1_2_IRQHandler()
{
	// Vref for X is the input to ADC1
	if (ADC1->ISR & ADC_ISR_EOC)
	{
		xcoord_data = ADC1->DR;
		xcoord_adc_flag = 1;
	}
	// Vref for Y is the input to ADC2
	if(ADC2->ISR & ADC_ISR_EOC)
	{
		ycoord_data = ADC2->DR;
		ycoord_adc_flag = 1;
	}
}
/* ------------------- SERIAL FUNCTIONS ------------------- */
// prints an int to USART
void USART_print_int(int num)
{
	char buff[BUFF_SIZE];
	int_to_str(num, buff);
	USART_Print(buff);
}
// converts and int and returns a string of length BUFF_SIZE
void int_to_str(int num, char* buff)
{
	snprintf(buff, /*sizeof(buff)*/BUFF_SIZE, "%d", num);
}
/* ------------------ JOYSTICK FUNCTIONS ------------------ */
// moves the cursor in the direction indicated by the joystick
void move_cursor()
{
	// update the previous cursor position
	prev_pos = cursor_pos;
	// check x+ (right) direction
	if(xcoord_data > X_NEUTRAL + THRESHOLD) // joystick wants to move right
	{
		if(cursor_pos.x < NUM_COLS - 1)   // room to move right
		{
			cursor_pos.x++;
		}
	}
	// check x- (left) direction
	if(xcoord_data < X_NEUTRAL - THRESHOLD) // joystick wants to move left
	{
		if(cursor_pos.x > 0)                                    // room to move left
		{
			cursor_pos.x--;
```

```
		}
	}
	// check y+ (down) direction
	if(ycoord_data > Y_NEUTRAL + THRESHOLD) // joystick wants to move down
	{
		if(cursor_pos.y < NUM_ROWS - 1)  // room to move down
		{
			cursor_pos.y++;
		}
	}
	// check y- (up) direction
	if(ycoord_data < Y_NEUTRAL - THRESHOLD) // joystick wants to move up
	{
		if(cursor_pos.y > 0)
		{
			cursor_pos.y--;
		}
	}
	// create color at the location if drawing
	if(drawing)
	{
		draw_pixel(cursor_pos.x, cursor_pos.y, draw_color);
	}
	static uint8_t state = 0;
	// check if the cursor is in the previous position
	if(cursor_pos.x == prev_pos.x && cursor_pos.y == prev_pos.y) // cursor is at the previous position, blink cursor
	{
		static uint8_t blink_count = 0;
		if(blink_count == BLINK_THRESHOLD)
		{
			if(state) //check_color(cursor_pos, WHITE)) // matrix color at cursor is WHITE
			{
				draw_pixel(cursor_pos.x, cursor_pos.y, prev_color); // BLACK ? // prev_color
				state = 0;
			}
			else
			{
				draw_pixel(cursor_pos.x, cursor_pos.y, check_color(cursor_pos, BLACK) ?
WHITE : BLACK); // WHITE
				state = 1;
			}
			// reset blink count
			blink_count = 0;
			// check the button status
			button_flag = get_joystick_button();
		}
		else
		{
			// increment blink count
```

```
                        blink_count++;
                }
        }
        else // need to replace the color at the previous position, and update prev_color for next time
        {
                draw_pixel(prev_pos.x, prev_pos.y, prev_color);
                prev_color = matrix_buffer[cursor_pos.x][cursor_pos.y];
                state = 0;
        }
}
/* ------------------- MATRIX FUNCTIONS ------------------- */
// adds the current cursor position to the shape
void add_pt_to_shape(point* shape, uint8_t size)
{
        // variables
        uint8_t size_to_use = size == 4 ? 2 : size;
//      uint8_t reset_shape = 1;
        int8_t pt_index = get_shape_index(shape, size_to_use);
        // resets shape if need be
        if(pt_index < 0)
        {
                reset_this_shape(shape, size_to_use);
                pt_index = 0;
        }
        // adds current cursor point to shape
        shape[pt_index].x = cursor_pos.x;
        shape[pt_index].y = cursor_pos.y;
        // check if shape is full now
        pt_index = get_shape_index(shape, size_to_use);
        // if is full, draw shape and reset shape
        if(pt_index < 0)
        {
                draw_shape(shape, size); // keep as original size
                reset_this_shape(shape, size_to_use);
        }
}
// draws the given shape
void draw_shape(point* shape, int size)
{
        // check to see which shape this is
        switch(size)
        {
        case 2: // line
                break;
        case 3: // triangle
                break;
        case 4: // square
                draw_rect(shape[0], shape[1]);
                break;
```

```
		default:
			break;
		}
}
// draws a line between the two points
void draw_line(point p1, point p2)
{
		// variables
		point next; // where starting from
		point dest; // where point going to
		// going from left to right, organize points as such
		if(p1.x <= p2.x) // p1 is on left and p2 is on right
		{
			next.x = p1.x;
			next.y = p1.y;
			dest.x = p2.x;
			dest.y = p2.y;
		}
		else				// p2 is on left and p1 is on right
		{
			next.x = p2.x;
			next.y = p2.y;
			dest.x = p1.x;
			dest.y = p1.y;
		}
		// draw the endpoints
		draw_pixel(p1.x, p1.y, draw_color);
		draw_pixel(p2.x, p2.y, draw_color);
		// draw everything between
		while(!(same_point(next, dest)) && pt_inbounds(next))
		{
			// test for case where slope would be undefined
			if(next.x == dest.x)
			{
			}
			int slope = get_slope(p1,p2); // gets the slope * 1,000,000
		}
}
// draws a rectangle with the two points as opposite corners
void draw_rect(point p1, point p2)
{
		// variables
		uint8_t xmin = get_min(p1.x, p2.x);
		uint8_t xmax = get_max(p1.x, p2.x);
		uint8_t ymin = get_min(p1.y, p2.y);
		uint8_t ymax = get_max(p1.y, p2.y);
		// draw left vertical line
		draw_vertical_line(ymin, ymax, xmin);
		// draw right vertical line
```

```
		draw_vertical_line(ymin, ymax, xmax);
		// draw top line
		draw_horizontal_line(xmin, xmax, ymin);
		// draw bottom line
		draw_horizontal_line(xmin, xmax, ymax);
}
// draws a vertical line given ymin, ymax, and x values
void draw_vertical_line(uint8_t ymin, uint8_t ymax, uint8_t xconst)
{
		for(int y = ymin; y <= ymax; y++)
		{
			draw_pixel(xconst, y, draw_color);
		}
}
// draws a horizontal line given the xmin, xmax and y values
void draw_horizontal_line(uint8_t xmin, uint8_t xmax, uint8_t yconst)
{
		for(int x = xmin; x <= xmax; x++)
		{
			draw_pixel(x, yconst, draw_color);
		}
}
// gets the maximum of two uint8_t values
uint8_t get_min(uint8_t a, uint8_t b)
{
		if(a < b) return a;
		else return b;
}
// gets the minimum of two uint8_t values
uint8_t get_max(uint8_t a, uint8_t b)
{
		if(a > b) return a;
		else return b;
}
// returns 1 if the point is within the bounds of the matrix and 0 if not
uint8_t pt_inbounds(point pt)
{
		return pt.x >= 0 && pt.x < NUM_COLS && pt.y >= 0 && pt.y < NUM_ROWS;
}
// returns 1 if the points have the same coordinates and 0 if they don't
uint8_t same_point(point p1, point p2)
{
		return p1.x == p2.x && p1.y == p2.y;
}
// gets the slope times 1 million, to avoid using floats
int get_slope(point p1, point p2)
{
		int numerator   = p1.y - p2.y * 1000000;
		int denominator = p1.x - p2.x;
```

```
        return numerator / denominator;
}
// returns -1 or the index of the shape that isn't set yet
int8_t get_shape_index(point* shape, int size)
{
        uint8_t pt_index = -1;
        for(int i = 0; i < size; i++)
        {
                if(shape[i].x < 0 || shape[i].y < 0)
                {
                        pt_index = i;
                }
        }
        return pt_index;
}
// resets the shapes to their defaults, coordinates = -1
void reset_shapes(point line[2], point square[2], point triangle[3])
{
        line[0].x = -1;
        line[0].y = -1;
        line[1].x = -1;
        line[1].y = -1;
        square[0].x = -1;
        square[0].y = -1;
        square[1].x = -1;
        square[1].y = -1;
        triangle[0].x = -1;
        triangle[0].y = -1;
        triangle[1].x = -1;
        triangle[1].x = -1;
        triangle[2].x = -1;
        triangle[2].x = -1;
}
// resets the vars of this shape
void reset_this_shape(point* shape, int size)
{
        for(int i = 0; i < size; i++)
        {
                shape[i].x = -1;
                shape[i].y = -1;
        }
}
// returns 1 if color at point in matrix is same as input, returns 0 if not
uint8_t check_color(point pt, color c)
{
        // check if point is inside the matrix
        if(pt.x >= 0 && pt.x < NUM_COLS && pt.y >= 0 && pt.y < NUM_ROWS)
        {
                color mx_color = matrix_buffer[pt.x][pt.y];
```

```
                return mx_color.r == c.r && mx_color.g == c.g && mx_color.b == c.b;
        }
        return 0;
}
// removes the pixel at the input coordinates from the buffer then updates the display
void clear_pixel(uint8_t x, uint8_t y)
{
        matrix_buffer[x][y] = BLACK;
        update_display();
}
// draws the selected color at the input coordinate then updates display
void draw_pixel(uint8_t x, uint8_t y, color c)
{
        matrix_buffer[x][y] = c;
        update_display();
}
// fills the matrix with the selected color
void fill_matrix(color c)
{
        // variables
        uint8_t row, col;
        // loop through matrix
        for(col = 0; col < NUM_COLS; col++)
        {
                for(row = 0; row < NUM_ROWS; row++)
                {
                        matrix_buffer[col][row] = c;
                }
        }
}
// clears the matrix buffer and updates the display
void clear_matrix()
{
        // variables
        uint8_t row, col;
        // loop through matrix
        for(col = 0; col < NUM_COLS; col++)
        {
                for(row = 0; row < NUM_ROWS; row++)
                {
                        matrix_buffer[col][row] = BLACK;
                }
        }
        // updates the display
        update_display();
}
// updates the display with the matrix buffer
void update_display()
{
```

```
	// variables
	uint8_t row, col;
	// initialize matrix control variables
	set_OE(HIGH); // disable output
	set_LAT(HIGH); // latch current data
	// loop through the row sections
	for(row = 0; row < NUM_ROWS / 2; row++)
	{
		// get ready to clock in a section (2 rows) of data
		set_OE(LOW); // enables output
		set_LAT(LOW); // removes latch from previous data
		// loop through the columns of the row
		for(col = 0; col < NUM_COLS; col++)
		{
			// sets the desired upper, lower color
			set_RGB_val(matrix_buffer[col][row], matrix_buffer[col][row + (NUM_ROWS / 2)]);
			// drive the clock
			drive_matrix_clk();
			// clear the previous values in the ODR
			clear_RGB_val(); // if isn't cleared, will fill entire row
		}
		// allow matrix to hold data
		set_OE(HIGH); // disable output, moving to another row
		set_LAT(HIGH); // latch current data, moving to another row
		// set row selection
		set_matrix_section(row);
	}
}
```

joystick.h

```
/*
* joystick.h
*
*  Created on: Dec 7, 2023
*      Author: jackkrammer
*
*
*  HEADER FILE FOR THE ARDUINO JOYSTICK MODULE
*
*		ADC PER COORDINATE (X AND Y)
*			the joystick has both a x Vref pin (PA0) and a y Vref pin (PA1)
*			this header file uses 2 ADC's (pin PA0 for ADC1 and pin PA1 for ADC2)
*
*		CALIBRATION STEPS
*				1)	need to calibrate with a reliable DC voltage source
*				2)	graph the voltage input to the ADC vs the data received
*					to get the calibration equation directly from the trendline equation
*				3)	graph should have y-axis = Vin, x-axis = data
```

```
*                                       so that can input data to equation and get the voltage
*
*                   USART COMPATIBILITY
*                           1)      if using the USART in other sections of the code
*                                       need to replace the FLCK value with the one here
*                                       this ensures a faster conversion
*
*                   CLOCK FREQUENCY
*                           1)      sets the clock to 32MHz for quicker ADC conversions
*                                       check that doesn't interfere with USART
*                                       comment out the FCLK declaration in USART header
*
*                   INTERRUPTS
*                           1)      uses interrupts to trigger each new ADC conversion
*
*
*                   RANGE OF VALUES
*                           the input voltage to the joystick should be 3.3V to work properly
*
*
*                   IMPLEMENTATIONS
*                           1)      the user of this file must include the IRQ for the ADC's
*                                       the IRQ must check for the ADC1 (xcoord) and ADC2 (ycoord)
individually
*                                       example code commented out in this
file-----------------------------------------------------------
*                           2)      the ADC for the X coordinate must be initialized before the ADC for the Y
coordinate
*
*                                       maybe doesn't matter, need to check with ADC123 clock initialization
*
*
*/
#ifndef SRC_JOYSTICK_H_
#define SRC_JOYSTICK_H_
// function declarations
void SystemClock_Config(void); // configures the system clock to 32MHz
void Error_Handler(void); // error handler for system clock configuration
void init_ADCs(); // initializes the system clock to 32MHz and ADC for x coordinate and ADC for y coordinate
void xcoord_adc_init(); // initializes ADC1 for reading the X coordinate on PA0
void ycoord_adc_init(); // initializes ADC2 for reading the Y coordinate on PA1
void xcoord_start_conversion(); // starts the ADC conversion for the x coordinate
void ycoord_start_conversion(); // starts the ADC conversion for the y coordinate
void joystick_button_init(); // initializes the GPIO pin PA4 for the input from the button (as interrupt ? )
uint8_t get_joystick_button(); // gets the value of the button where 1 is being pressed and 0 is not pressed
// THIS FUNCTION DECLARATION INDICATES THAT NEED TO IMPLEMENT IN THE MAIN.C FILE
// clear flag and set globals
void ADC1_2_IRQHandler();/*
{
```

```
		// Vref for X is the input to ADC1
		if (ADC1->ISR & ADC_ISR_EOC)
		{
			xcoord_data = ADC1->DR;
			xcoord_adc_flag = 1;
		}
		// Vref for Y is the input to ADC2
		if(ADC2->ISR & ADC_ISR_EOC)
		{
			ycoord_data = ADC2->DR;
			ycoord_adc_flag = 1;
		}
}*/
// gets the value of the button where 1 is being pressed and 0 is not pressed
uint8_t get_joystick_button()
{
		return !(GPIOA->IDR & GPIO_IDR_ID4);
}
// initializes the GPIO pin PA4 as the input from the active low joystick button
void joystick_button_init()
{
		// starts the GPIOA peripheral clock
		RCC->AHB2ENR |= (RCC_AHB2ENR_GPIOAEN);
		// input mode
		GPIOA->MODER &= ~(GPIO_MODER_MODE4);
		// pull up resistor
		GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPD4);
		GPIOA->PUPDR |=  (GPIO_PUPDR_PUPD4_0);
}
// starts the ADC conversion for the y coordinate
void ycoord_start_conversion()
{
		// Vref for y is the input to ADC2
		ADC2->CR |= ADC_CR_ADSTART;
}
// starts the ADC conversion for the x coordinate
void xcoord_start_conversion()
{
		// Vref for x is the input to ADC1
		ADC1->CR |= ADC_CR_ADSTART;
}
// initializes ADC2 for reading the Y coordinate on PA1
void ycoord_adc_init()
{
		// turns on the clock for ADC
		RCC->AHB2ENR |= RCC_AHB2ENR_ADCEN;
		ADC123_COMMON->CCR = ((ADC123_COMMON->CCR & ~(ADC_CCR_CKMODE)) |
ADC_CCR_CKMODE_0);
		// make sure conversion isn't started
```

```
	ADC2->CR &= ~(ADC_CR_ADSTART);
	// take the ADC out of deep power down mode
	ADC2->CR &= ~(ADC_CR_DEEPPWD);
	// enable to voltage regulator guards the voltage
	ADC2->CR |= (ADC_CR_ADVREGEN);
	// delay at least 20 microseconds (to power up) before calibration
	for(uint16_t i =0; i<1000; i++)
		for(uint16_t j = 0;  j<100; j++); //delay at least 20us
	// calibrate, you need to digitally calibrate
	ADC2->CR &= ~(ADC_CR_ADEN | ADC_CR_ADCALDIF); //ensure ADC is not enabled, also choose single ended calibration
	ADC2->CR |= ADC_CR_ADCAL;	// start calibration
	while(ADC2->CR & ADC_CR_ADCAL); // wait for calibration
	// configure single ended mode before enabling ADC
	ADC2->DIFSEL &= ~(ADC_DIFSEL_DIFSEL_6); // PA1 is ADC1_IN6 (found via nucleo_board.pdf), single ended mode, DIFFERENT PER CHANNEL
	ADC2->SMPR1 |= 0x7 << 18; // configures sample period, DIFFERENT FOR EACH CHANNEL
	// enable ADC
	ADC2->ISR |= (ADC_ISR_ADRDY); // tells hardware that ADC is ready for conversion
	ADC2->CR |= ADC_CR_ADEN; // enables the ADC
	while(!(ADC2->ISR & ADC_ISR_ADRDY)); // waits until the ADC sets this flag low
	ADC2->ISR |= (ADC_ISR_ADRDY); // sets the flag high again
	// configure ADC
	ADC2->SQR1 = (ADC2->SQR1 & ~(ADC_SQR1_SQ1_Msk | ADC_SQR1_L_Msk)) |(6 << ADC_SQR1_SQ1_Pos); // DIFFERENT PER CHANNEL
	ADC2->ISR &= ~(ADC_ISR_EOC); // clears end of conversion flag
	// enables interrupts
	ADC2->IER |= (ADC_IER_EOC); // enables ADC interrupt
	NVIC->ISER[0] |= (1 << (ADC1_2_IRQn & 0x1F)); // enables NVIC interrupt
	__enable_irq(); // enables ARM interrupts
	//configure GPIO pin PA0
	RCC->AHB2ENR |= (RCC_AHB2ENR_GPIOAEN);		// enables the GPIOA peripheral clock
	GPIOA->MODER |= (GPIO_MODER_MODE1);	// analog mode for PA1
	GPIOA->ASCR |= GPIO_ASCR_ASC1;				// set PA1 to analog
}
// initializes ADC1 for reading the X coordinate on PA0
void xcoord_adc_init()
{
//	// configures the system clock to be 32MHz for ADC
//	SystemClock_Config();
	// turns on the clock for ADC
	RCC->AHB2ENR |= RCC_AHB2ENR_ADCEN;
	ADC123_COMMON->CCR = ((ADC123_COMMON->CCR & ~(ADC_CCR_CKMODE)) | ADC_CCR_CKMODE_0);
	// make sure conversion isn't started
	ADC1->CR &= ~(ADC_CR_ADSTART);
	// take the ADC out of deep power down mode
	ADC1->CR &= ~(ADC_CR_DEEPPWD);
	// enable to voltage regulator guards the voltage
```

```
	ADC1->CR |= (ADC_CR_ADVREGEN);
	// delay at least 20 microseconds (to power up) before calibration
	for(uint16_t i =0; i<1000; i++)
		for(uint16_t j = 0;  j<100; j++); //delay at least 20us
	// calibrate, you need to digitally calibrate
	ADC1->CR &= ~(ADC_CR_ADEN | ADC_CR_ADCALDIF); //ensure ADC is not enabled, also choose single ended calibration
	ADC1->CR |= ADC_CR_ADCAL;	// start calibration
	while(ADC1->CR & ADC_CR_ADCAL); // wait for calibration
	// configure single ended mode before enabling ADC
	ADC1->DIFSEL &= ~(ADC_DIFSEL_DIFSEL_5); // PA0 is ADC1_IN5 (found via nucleo_board.pdf), single ended mode
	ADC1->SMPR1 |= 0x7 << 15; // configures sample period, DIFFERENT FOR EACH CHANNEL
	// enable ADC
	ADC1->ISR |= (ADC_ISR_ADRDY); // tells hardware that ADC is ready for conversion
	ADC1->CR |= ADC_CR_ADEN; // enables the ADC
	while(!(ADC1->ISR & ADC_ISR_ADRDY)); // waits until the ADC sets this flag low
	ADC1->ISR |= (ADC_ISR_ADRDY); // sets the flag high again
	// configure ADC
	ADC1->SQR1 = (ADC1->SQR1 & ~(ADC_SQR1_SQ1_Msk | ADC_SQR1_L_Msk)) |(5 << ADC_SQR1_SQ1_Pos); // DIFFERENT PER CHANNEL
	ADC1->ISR &= ~(ADC_ISR_EOC); // clears end of conversion flag
	// enables interrupts
	ADC1->IER |= (ADC_IER_EOC); // enables ADC interrupt
	NVIC->ISER[0] |= (1 << (ADC1_2_IRQn & 0x1F)); // enables NVIC interrupt
	__enable_irq(); // enables ARM interrupts
	//configure GPIO pin PA0
	RCC->AHB2ENR |= (RCC_AHB2ENR_GPIOAEN);		// enables the GPIOA peripheral clock
	GPIOA->MODER |= (GPIO_MODER_MODE0);	// analog mode for PA0
	GPIOA->ASCR |= GPIO_ASCR_ASC0;				// set PA0 to analog
//	// start conversion
//	ADC1->CR |= ADC_CR_ADSTART;
}
// initializes the system clock to 32MHz and ADC for x coordinate and ADC for y coordinate
void init_ADCs()
{
	SystemClock_Config();
	xcoord_adc_init();
	ycoord_adc_init();
}
// error handler for system clock configuration
void Error_Handler(void)
{
/* USER CODE BEGIN Error_Handler_Debug */
/* User can add his own implementation to report the HAL error return state */
__disable_irq();
while (1)
{
}
```

```
/* USER CODE END Error_Handler_Debug */
}
// configures the system clock to 32MHz
void SystemClock_Config(void)
{
	RCC_OscInitTypeDef RCC_OscInitStruct = {0};
	RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
	/** Configure the main internal regulator output voltage
	*/
	if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
	{
		Error_Handler();
	}
	/** Initializes the RCC Oscillators according to the specified parameters
	* in the RCC_OscInitTypeDef structure.
	*/
	RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
	//RCC_OscInitStruct.MSIState = RCC_MSI_ON;  //datasheet says NOT to turn on the MSI then change the
frequency.
	RCC_OscInitStruct.MSICalibrationValue = 0;
	RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_10;
	/* from stm32l4xx_hal_rcc.h
	#define RCC_MSIRANGE_0          MSI = 100 KHz
	#define RCC_MSIRANGE_1          MSI = 200 KHz
	#define RCC_MSIRANGE_2          MSI = 400 KHz
	#define RCC_MSIRANGE_3          MSI = 800 KHz
	#define RCC_MSIRANGE_4          MSI = 1 MHz
	#define RCC_MSIRANGE_5          MSI = 2 MHz
	#define RCC_MSIRANGE_6          MSI = 4 MHz
	#define RCC_MSIRANGE_7          MSI = 8 MHz
	#define RCC_MSIRANGE_8          MSI = 16 MHz
	#define RCC_MSIRANGE_9          MSI = 24 MHz
	#define RCC_MSIRANGE_10          MSI = 32 MHz
	#define RCC_MSIRANGE_11          MSI = 48 MHz  dont use this one*/
	RCC_OscInitStruct.MSIState = RCC_MSI_ON;  //datasheet says NOT to turn on the MSI then change the
frequency.
	RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
	if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
	{
		Error_Handler();
	}
	/** Initializes the CPU, AHB and APB buses clocks
	*/
	RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                                  |RCC_CLOCKTYPE_PCLK1|
RCC_CLOCKTYPE_PCLK2;
	RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
	RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
	RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
```

```
	RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
	if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
	{
		Error_Handler();
	}
}
#endif /* SRC_JOYSTICK_H_ */
```

uart.h

```
/*
* uart.h
*
*  Created on: Dec 5, 2023
*      Author: jackkrammer
*      as of 4:54pm 20231207
*/
#ifndef UART_H_
#define UART_H_
//#define F_CLK 4000000  // bus clock is 4 MHz
#define F_CLK 32000000 // clock for ADC is 32MHz
/* Private function prototypes -----------------------------------------------*/
void USART_Print(const char* message);
void USART_Escape_Code(const char* msg);
void USART_init();
void USART_init()
{
	// configure GPIO pins for USART2 (PA2, PA3) follow order of configuring registers
	// AFR, OTYPER, PUPDR, OSPEEDR, MODDER otherwise a glitch is created on the output pin
	RCC->AHB2ENR |= (RCC_AHB2ENR_GPIOAEN);
	GPIOA->AFR[0] &= ~(GPIO_AFRL_AFSEL2 | GPIO_AFRL_AFSEL3);                    // mask AF
selection
	GPIOA->AFR[0] |= ((7 << GPIO_AFRL_AFSEL2_Pos ) |                            // select USART2 (AF7)
	                  (7 << GPIO_AFRL_AFSEL3_Pos));
	GPIOA->MODER &= ~(GPIO_MODER_MODE2 | GPIO_MODER_MODE3);// enable alternate function
	GPIOA->MODER |= (GPIO_MODER_MODE2_1 | GPIO_MODER_MODE3_1);   // for PA2 and PA3
	// Configure USART2 connected to the debugger virtual COM port
	RCC->APB1ENR1 |= RCC_APB1ENR1_USART2EN;          // enable USART by turning on system clock
	USART2->CR1 &= ~(USART_CR1_M1 | USART_CR1_M0); // set data to 8 bits
	USART2->BRR = F_CLK / 115200;
	USART2->CR1 |= (USART_CR1_TE | USART_CR1_RE);                // enable transmit and receive for
USART
	// enable interrupts for USART2 receive
	USART2->CR1 |= USART_CR1_RXNEIE;                    // enable RXNE interrupt on USART2
	USART2->ISR &= ~(USART_ISR_RXNE);                   // clear interrupt flag while (message[i] != 0)
	NVIC->ISER[1] = (1 << (USART2_IRQn & 0x1F));        // enable USART2 ISR
	__enable_irq();
	USART2->CR1 |= USART_CR1_UE;                        // enable USART
	// clear screen
```

```
	USART_Escape_Code("[2J");
	// top left cursor
	USART_Escape_Code("[H");
}
// use a for loop to output one byte at a time to TDR
void USART_Print(const char* message)
{
	uint8_t i;
	for(i=0; message[i] != 0; i++){			// check for terminating NULL character
		while(!(USART2->ISR & USART_ISR_TXE));	// wait for transmit buffer to be empty
		USART2->TDR = message[i];			// transmit character to USART
	}
}
// add /ESC to TDR before the actual escape code
void USART_Escape_Code(const char* msg)
{
	while(!(USART2->ISR & USART_ISR_TXE));	// wait for transmit buffer to be empty
	USART2->TDR = 0x1B;
	USART_Print(msg);
}
// uses the corresponding color escape code or just prints out character
// enter adds \n after the \r it defaults to
void USART2_IRQHandler(void)
{
	if (USART2->ISR & USART_ISR_RXNE)
	{
//		// writes keyboard input to the serial display
//		while(!(USART2->ISR & USART_ISR_TXE));
//			USART2->TDR = USART2->RDR;
		// clears the flag
		USART2->ISR &= ~(USART_ISR_RXNE);
	}
}
#endif /* UART_H_ */
```

keypad_12.h

```
/*
* keypad_12.h
*
*  Created on: Oct 17, 2023
*      Author: jackkrammer
*
*  header file for the COM-14662 keypad
*/
#ifndef SRC_KEYPAD_12_H_
#define SRC_KEYPAD_12_H_
#define NUM_COL		3
#define NUM_ROW		4
```

```
const char keypad_chars[NUM_ROW * NUM_COL] = {
        '1', '2', '3',
        '4', '5', '6',
        '7', '8', '9',
        '*', '0', '#'
};
const uint8_t keypad_vals[NUM_ROW * NUM_COL] = {
        0x1, 0x2, 0x3,
        0x4, 0x5, 0x6,
        0x7, 0x8, 0x9,
        0xA, 0x0, 0xB
};
void keypad_init(); // initializes keypad
int8_t loop_keypad_once(); // returns index of button pressed or -1 if nothing is pressed
int8_t multiplex_keypad(); // returns index of button pressed by multiplexing keypad until button is pressed
char get_keypad_char(); // returns the char of the button pressed, null if invalid or error
int8_t get_keypad_value(); // returns the value of the button pressed, -1 if invalid or error
// returns the value of the button pressed, -1 if invalid or error
int8_t get_keypad_value()
{
        int8_t index = multiplex_keypad(); // get the index
        if(index != -1)
        {
                return keypad_vals[index];
        }
        return -1;
}
// returns the char of the button pressed
char get_keypad_char()
{
        int8_t index = multiplex_keypad(); // get the index
        if(index != -1) // there is a valid index
        {
                return keypad_chars[index];
        }
        return '\0';
}
// returns index of button pressed or -1 if nothing is pressed
int8_t loop_keypad_once()
{
        uint8_t col, row = 0;
        // clear column ODR
        GPIOC->ODR &= ~(GPIO_ODR_OD4 | GPIO_ODR_OD5 | GPIO_ODR_OD6);
        // drive each column once
        for(col = 0; col < NUM_COL; col++)
        {
                // drive column
                GPIOC->BSRR = (1 << NUM_ROW) << col;
                // check rows
```

```
                for(row = 0; row < NUM_ROW; row++)
                {
                        if(GPIOC->IDR & (1 << row))
                        {
                                return NUM_COL * row + col;
                        }
                }
                // disable column
                GPIOC->BSRR = ((1 << GPIO_BSRR_BR0_Pos) << NUM_ROW) << col;
        }
        return -1;
}
// returns index of button pressed by multiplexing keypad until button is pressed
int8_t multiplex_keypad()
{
        uint8_t col, row = 0;
        // clear column ODR
        GPIOC->ODR &= ~(GPIO_ODR_OD4 | GPIO_ODR_OD5 | GPIO_ODR_OD6);
        // multiplex columns
        while(1)
        {
                // drive column
                GPIOC->BSRR = (1 << NUM_ROW) << col;
                // check rows
                for(row = 0; row < NUM_ROW; row++)
                {
                        if(GPIOC->IDR & (1 << row))
                        {
                                return NUM_COL * row + col;
                        }
                }
                // disable column
                GPIOC->BSRR = ((1 << GPIO_BSRR_BR0_Pos) << NUM_ROW) << col;
                // go to next column, loops through
                col = (col + 1) % NUM_COL;
        }
        return -1;
}
void keypad_init()
{
        /*
         * initializes pins to drive(output) columns and read(input) from rows
         * columns are on pins PC4(col 1) PC5(col 2) PC6(col 3)
         * rows are on pins PC0(row 1) PC1(row 2) PC2(row 3) PC3(row 4)
         * sets pull down resistors on row pins for better reads
         * */
        // enable clock for port C
        RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN;
        // drive columns on pins PC4(col 1) PC5(col 2) PC6(col 3)
```

```
	GPIOC->MODER &= ~(GPIO_MODER_MODE4 | GPIO_MODER_MODE5 |
GPIO_MODER_MODE6); // clear pins
	GPIOC->MODER |= ( (1 << GPIO_MODER_MODE4_Pos) |
                                                (1 << GPIO_MODER_MODE5_Pos) |
                                                (1 << GPIO_MODER_MODE6_Pos) ); // set pins to output
	// read from rows on pins PC0(row 1) PC1(row 2) PC2(row 3) PC3(row 4)
	GPIOC->MODER &= ~(GPIO_MODER_MODE0 | GPIO_MODER_MODE1 |
                                                GPIO_MODER_MODE2 | GPIO_MODER_MODE3); //
clear pins, sets them to input
	// set pull-down resistors for rows to prevent floating voltages (better for reading)
	GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPD0 | GPIO_PUPDR_PUPD1 |
                                                GPIO_PUPDR_PUPD2 | GPIO_PUPDR_PUPD3); // clear
pins
	GPIOC->PUPDR |= ( (2 << GPIO_PUPDR_PUPD0_Pos) |
                                                (2 << GPIO_PUPDR_PUPD1_Pos) |
                                                (2 << GPIO_PUPDR_PUPD2_Pos) |
                                                (2 << GPIO_PUPDR_PUPD3_Pos) ); // pull pins low
}
#endif /* SRC_KEYPAD_12_H_ */
```

rgb_matrix.h

```
/*
* rgb_matrix.h
*
*  Created on: Dec 7, 2023
*      Author: jackkrammer
*/
#ifndef INC_RGB_MATRIX_H_
#define INC_RGB_MATRIX_H_
// defines
#define NUM_ROWS 16 // changed for testing, trying to only have the 1st and 9th row be on //16
#define NUM_COLS 32 // changed for testing, trying to only have the left two cols be on //32
#define HIGH 1
#define LOW 0
// typedefs
typedef struct color
{
	uint8_t r,g,b;
} color;
// function declarations
void matrix_init(); // initializes the Adafruit 32x16 RGB LED matrix
void matrix_begin(); // initializes the matrix values to not latch data and disable output
void update_display(); // updates the display with the matrix buffer
void clear_matrix(); // clears the matrix buffer and updates the display
void fill_matrix(color c); // fills the matrix with the selected color
void draw_pixel(uint8_t x, uint8_t y, color c); // draws the selected color at the input coordinate then updates display
void clear_pixel(uint8_t x, uint8_t y); // removes the pixel at the input coordinates from the buffer then updates the
display
```

```
void make_smiley(color c); // makes a 'relief' smiley face with the background color as the input
void make_hi(color c); // makes a relief hi
void make_logo(color c); // makes the doodlestick name with logo
void set_matrix_section(uint8_t row); // sets the row select bits of the matrix to the desired value
void set_LAT(uint8_t val); // sets the latch pin HIGH if val > 0 and LOW if val == 0
void set_OE(uint8_t val); // sets the output enable pin HIGH if val > 0 and LOW if val == 0
void drive_matrix_clk(); // drives the matrix clock high then low, for a idle-low rising-edge clock
void set_RGB_val(color upper, color lower); // sets the RGB values according to the input colors
void clear_RGB_val(); // clears the values in the RGB pins' ODR
// makes the doodlestick name with logo
void make_logo(color c)
{
        // draw doodle
        // d
        draw_pixel(3,1,c);
        draw_pixel(3,2,c);
        draw_pixel(3,3,c);
        draw_pixel(3,4,c);
        draw_pixel(3,5,c);
        draw_pixel(2,5,c);
        draw_pixel(1,5,c);
        draw_pixel(1,4,c);
        draw_pixel(1,3,c);
        draw_pixel(2,4,c);
        // o
        draw_pixel(5,5,c);
        draw_pixel(5,4,c);
        draw_pixel(5,3,c);
        draw_pixel(6,3,c);
        draw_pixel(7,3,c);
        draw_pixel(7,4,c);
        draw_pixel(7,5,c);
        draw_pixel(6,5,c);
        // draw stick
        // draw logo
}
// makes a relief hi
void make_hi(color c)
{
        fill_matrix(c);
        clear_pixel(1,1);
        clear_pixel(1,2);
        clear_pixel(1,3);
        clear_pixel(1,4);
        clear_pixel(1,5);
        clear_pixel(2,3);
        clear_pixel(3,1);
        clear_pixel(3,2);
        clear_pixel(3,3);
```

```
	clear_pixel(3,4);
	clear_pixel(3,5);
	clear_pixel(5,2);
	clear_pixel(5,4);
	clear_pixel(5,5);
}
// makes a 'relief' smiley face with the background color as the input
void make_smiley(color c)
{
	fill_matrix(c);
	clear_pixel(3,2);
	clear_pixel(5,2);
	clear_pixel(2,4);
	clear_pixel(3,5);
	clear_pixel(4,5);
	clear_pixel(5,5);
	clear_pixel(6,4);
}
// sets the row select bits of the matrix to the desired value
void set_matrix_section(uint8_t row)
{
	/*
	 * sets the appropriate value to the row select pins below
	 *			A(LSB)  B		C(MSB)
	 *			PC7	PC8  PC9
	 */
	// variables
//	uint8_t a = row & 0x01; // LSB of row select
//	uint8_t b = row & 0x02; //
//	uint8_t c = row & 0x04; // MSB of row select
	// clears pins
	GPIOC->ODR &= ~((1 << 7) | (1 << 8) | (1 << 9));
	// sets pins to value
//	GPIOC->ODR |= (((!!c) << 9) | ((!!b) << 8) | ((!!a) << 7)); // works
//	GPIOC->ODR |= (((row & 0x04) << 9) | ((row & 0x02) << 8) | ((row & 0x01) << 7)); // doesn't work
	if(row & 0x01) GPIOC->ODR |= (1 << 7); // set LSB of ADDR bits
	if(row & 0x02) GPIOC->ODR |= (1 << 8);
	if(row & 0x04) GPIOC->ODR |= (1 << 9); // set MSB of ADDR bits
}
// sets the latch pin HIGH if val > 0 and LOW if val == 0
void set_LAT(uint8_t val)
{
	/*
	 * sets the chosen latch value to the ODR
	 *			LAT
	 *			PC11
	 */
	// sets the pin low
	GPIOC->ODR &= ~(1 << 11);
```

```
	// sets the pin high if input is > 0
	if(val > 0)
		GPIOC->ODR |= (1 << 11);
}
// sets the output enable pin HIGH if val > 0 and LOW if val == 0
void set_OE(uint8_t val)
{
	/*
	 * sets the chosen output enable value to the ODR
	 *			OE
	 *			PC12
	 */
	// sets the pin low
	GPIOC->ODR &= ~(1 << 12);
	// sets the pin high if input is > 0
	if(val > 0)
		GPIOC->ODR |= (1 << 12);
}
// drives the matrix clock high then low, for a idle-low rising-edge clock
void drive_matrix_clk()
{
	GPIOC->BSRR |= (1 << 10); // sets clock HIGH
	GPIOC->BRR |= (1 << 10);  // sets clock LOW
}
// sets the RGB values according to the input colors
void set_RGB_val(color upper, color lower)
{
	/*
	 * RGB color pins
	 *			|---upper---|  |---lower---|
	 *			R1  G1  B1  R2  G2  B2
	 *			PB0 PB1 PB2 PB3 PB4 PB5
	 */
	// clear the previous values in the ODR
//	GPIOB->ODR &= ~((1 << 0) | (1 << 1) | (1 << 2) | (1 << 3) | (1 << 4) | (1 << 5));
	// sets the upper and lower rgb values at the same time
	GPIOB->ODR |= ((upper.r << 0) | (upper.g << 1) | (upper.b << 2)
					| (lower.r << 3) | (lower.g << 4) | (lower.b << 5));
}
// clears the values in the RGB pins' ODR
void clear_RGB_val()
{
	/*
	 * RGB color pins
	 *			|---upper---|  |---lower---|
	 *			R1  G1  B1  R2  G2  B2
	 *			PB0 PB1 PB2 PB3 PB4 PB5
	 */
	// clear the previous values in the ODR
```

```
	GPIOB->ODR &= ~((1 << 0) | (1 << 1) | (1 << 2) | (1 << 3) | (1 << 4) | (1 << 5));
}
// initializes the matrix values to not latch data and disable output
void matrix_begin()
{
	/*
	 * RGB pins to be zero
	 * ADDR pins to be zero
	 * CLK pin to be idle low
	 * LAT pin to be low b/c not latching any data
	 * OE pin to be high to disable output
	 */
	// RGB pins
	GPIOB->ODR &= ~((1 << 0) | (1 << 1) | (1 << 2) | (1 << 3) | (1 << 4) | (1 << 5));
	// ADDR pins, CLK pin, LAT pin
	GPIOC->ODR &= ~((1 << 7) | (1 << 8) | (1 << 9) | (1 << 10) | (1 << 11));
	// OE pin
	GPIOC->ODR |= (1 << 12);
	// addr pins
//	GPIOC->ODR |= ((0 << 9) | (0 << 8) | (1 << 7)); // sets address to 1
	// initializes the buffer to be all dark
	clear_matrix();
}
// initializes the Adafruit 32x16 RGB LED matrix
void matrix_init()
{
	/*
	 * initializes the matrix input pins
	 * color pins are the data pins that control the color of the current matrix section
	 *			|---upper---|  |---lower---|
	 *			R1  G1  B1  R2  G2  B2
	 *			PB0 PB1 PB2 PB3 PB4 PB5
	 * row select pins are are the pins that control the current section of the matrix
	 *			A(LSB) B		C(MSB)
	 *			PC7    PC8 PC9
	 * clock pin is the input clock to the matrix
	 *			CLK
	 *			PC10
	 * latch pin is the pin to signal the end of a row of data
	 *			LAT
	 *			PC11
	 * output enable pin is the pin to turn the LEDs off when moving from one row to next
	 *			OE
	 *			PC12
	 */
	// enable clock for port B and C
	RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
	RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN;
	// set color pins as outputs
```

```
	GPIOB->MODER &= ~(GPIO_MODER_MODE0
			| GPIO_MODER_MODE1
			| GPIO_MODER_MODE2
			| GPIO_MODER_MODE3
			| GPIO_MODER_MODE4
			| GPIO_MODER_MODE5);
	GPIOB->MODER |= (GPIO_MODER_MODE0_0
			| GPIO_MODER_MODE1_0
			| GPIO_MODER_MODE2_0
			| GPIO_MODER_MODE3_0
			| GPIO_MODER_MODE4_0
			| GPIO_MODER_MODE5_0);
	// set row select pins as outputs
	GPIOC->MODER &= ~(GPIO_MODER_MODE7
				| GPIO_MODER_MODE8
				| GPIO_MODER_MODE9);
	GPIOC->MODER |= (GPIO_MODER_MODE7_0
				| GPIO_MODER_MODE8_0
				| GPIO_MODER_MODE9_0);
	// set clock, latch, output enable pins as output
	GPIOC->MODER &= ~(GPIO_MODER_MODE10
				| GPIO_MODER_MODE11
				| GPIO_MODER_MODE12);
	GPIOC->MODER |= (GPIO_MODER_MODE10_0
				| GPIO_MODER_MODE11_0
				| GPIO_MODER_MODE12_0);
}
#endif /* INC_RGB_MATRIX_H_ */
```

timer2.h

```
/*
* timer2.h
*
*  Created on: Oct 18, 2023
*      Author: jackkrammer
*
*  a header file to make the initialization of TIM2 easier
*/
#ifndef SRC_TIMER2_H_
#define SRC_TIMER2_H_
void TIM2_init(uint32_t arr, uint32_t psc, uint32_t ccr1); // initializes TIM2 with the provided ARR PSC CCR1 values
// initializes TIM2 with the provided ARR PSC CCR1 values
void TIM2_init(uint32_t arr, uint32_t psc, uint32_t ccr1)
{
	// enable the clock for TIM2
	RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN; // TIM2 clock
	// configure timer count settings
	TIM2->CR1 &= ~TIM_CR1_CMS; // sets count to be one directional
```

```
        TIM2->CR1 &= ~TIM_CR1_DIR; // sets timer to count up
        TIM2->PSC = psc; // divides timer clock by PSC + 1
        TIM2->ARR = arr;//804 // timer counts to ARR + 1 // 399 for 50% (w/o CCR1) // 799 when w CCR1
        //TIM2->CCR1 = ccr1;//594 // compares count to CCR1 // 599 for 25% duty b/c 599 ~= 0.75*799
        // enable interrupts
        __enable_irq(); // enables ARM interrupts
        NVIC->ISER[0] = (1 << (TIM2_IRQn & 0x1F)); // enables NVIC TIM2 interrupt
        TIM2->SR &= ~TIM_SR_UIF; // resets TIM2 update interrupt flag
        //TIM2->SR &= ~TIM_SR_CC1IF; // resets TIM2 CC1 interrupt flag
        TIM2->DIER |= TIM_DIER_UIE; // enable TIM2 update interrupt
        //TIM2->DIER |= TIM_DIER_CC1IE; // enable TIM2 CC1 interrupt
        // start timer
        TIM2->CR1 |= TIM_CR1_CEN; // enables the counter
}
#endif /* SRC_TIMER2_H_ */
```

References

STM32L476RG TRM

STM32 Nucleo-64 Boards User Manual UM1724

Adafruit 32x16 RGB LED Matrix Reference Manual

COM-14662 Keypad Pin-out

EasyEDA – Electrical Schematic

draw.io – System Architecture and Flowchart