Department of Electrical and Computer Engineering

Microcontrollers Lab
EGC332-01

Real World Intersection
Final Project

Date: 12/08/20 Semester: Fall 2020	
Group Members	Department
Kevin Nelson	EE

Lab Instructor: Prof. Mike Otis

Table of Contents

I. Abstract

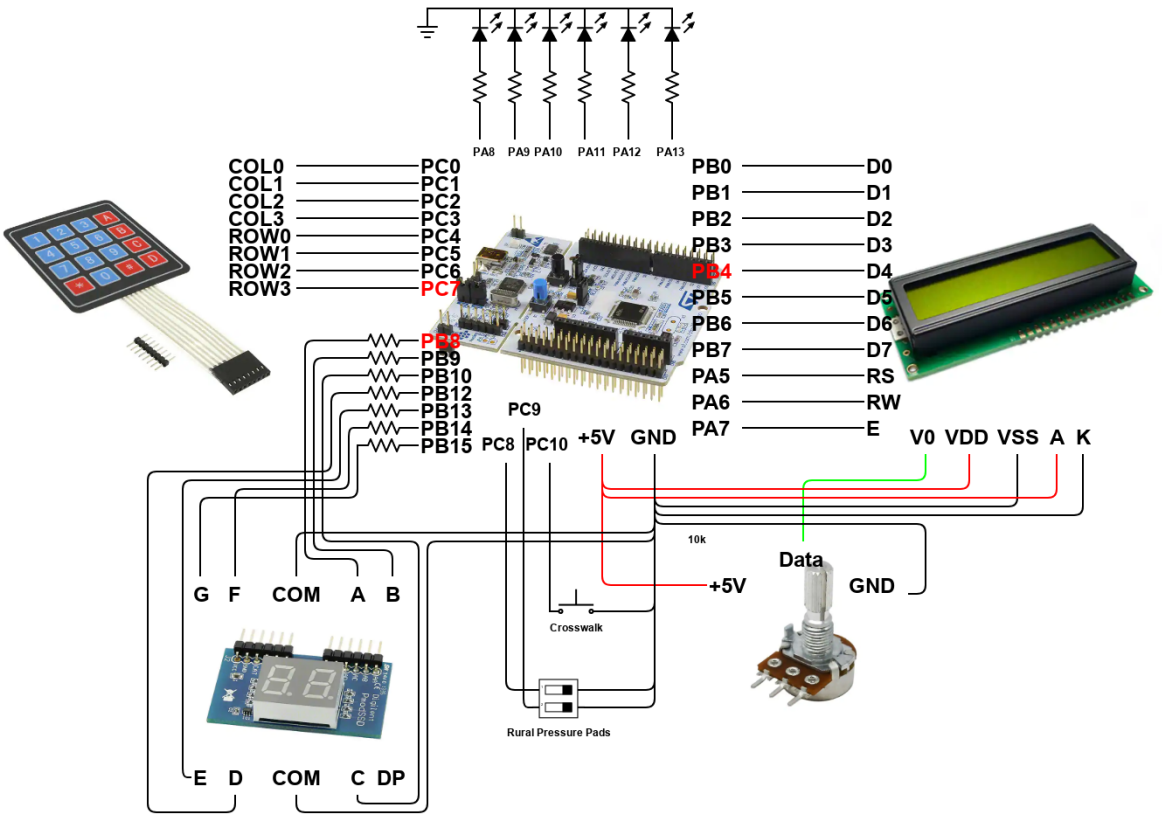
This project is a thorough demonstration on how one can utilize an STM32 board in conjunction with a multitude of peripherals in order to achieve a fully functional real world intersection. It is through the use of coding language C, that the author demonstrates two possible approaches to coding such a device. With real world applications being very complicated, approaching this problem through the use of traditional programming or interrupts, allows one to account for different situations, as one would expect in real world applications. It should be noted that neither method is perfect, and both maintain a number of setbacks, however it will be demonstrated that both can be effective.

This project includes a number of peripheral devices that one might expect to find in the real world, regardless of location. Some of these components are: LCD Display, Hex Keypad, six Light Emitting Diodes(LEDs), seven segment display, and some switches. Each component is crucial, as it allows a user to enable different modes depending on the setting and situation, and will allow pedestrians to enable a crosswalk when needed.

II. Design

Hardware Design

Figure 1.1 - Real World Intersection



Software Design

Figure 1.1 - Urban Traffic Light

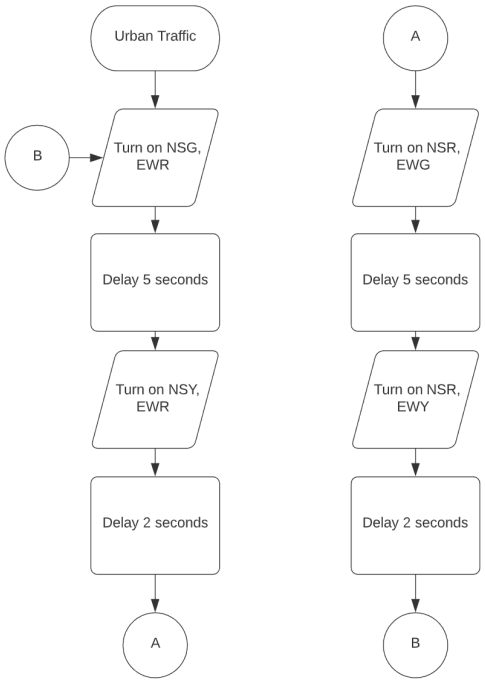
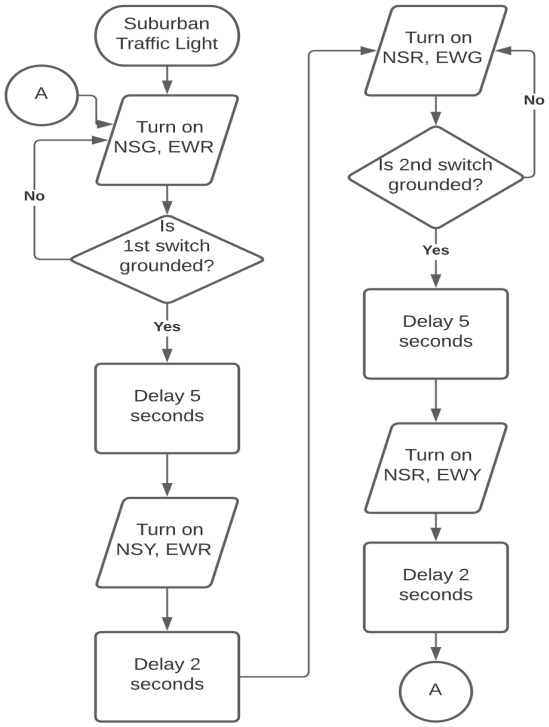


Figure 1.2 - Suburban Traffic Light



Both Figure 1.1 and 1.2 show how through the use of delays and setting individual states, one can create a visual representation of a traffic light. Each parallelogram represents a state turning on and off different bits that correlate to an LED. Additionally, the squares represent delays, this is important to maintain each state for a given period of time. Lastly, a shape unique to Figure 1.2 is the diamond. Each diamond represents a decision that corresponds to a switch being turned on or off. Like a pressure pad on the road, if no car drives by and the pad is ungrounded, the state will remain constant. Inversely, if a car drives by and grounds the pressure pad, the state will change.

Figure 1.3 - Seven Segment Display

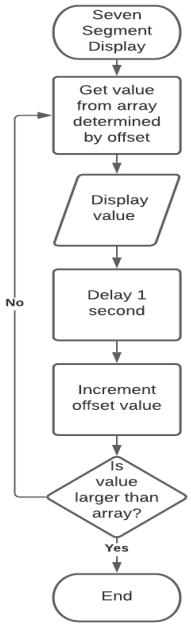


Figure 1.4 - LCD Display

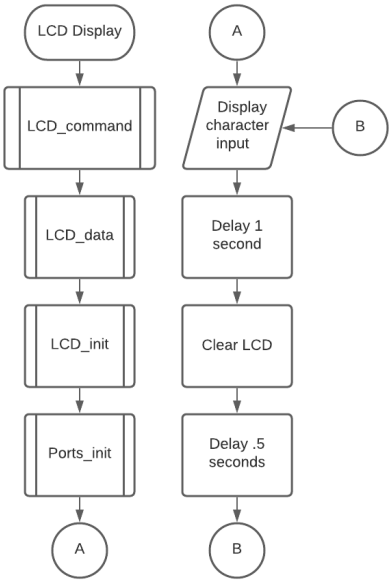


Figure 1.1 shows the process in which one can effectively create a countdown timer by accessing an array. It is important to see that determining which value of the array to get is determined by an offset value, which then passes back to a function that outputs the value to some LEDs. Additionally, one can see that a decision block is used, this is to ensure that the user will no longer try to access the array once the last value has been loaded.

Figure 1.4 shows the necessary steps in which one will have to take in order to turn on and output a character to an LCD display. The squares with two vertical lines represent the subroutines that Hitachi requires in order to turn on the LCD controller and allow it for use. It should be seen that there are no peripheral inputs that determine what is to be displayed, and the characters chosen for display come from the software.

Figure 1.5 - Hex Keypad with LCD Display

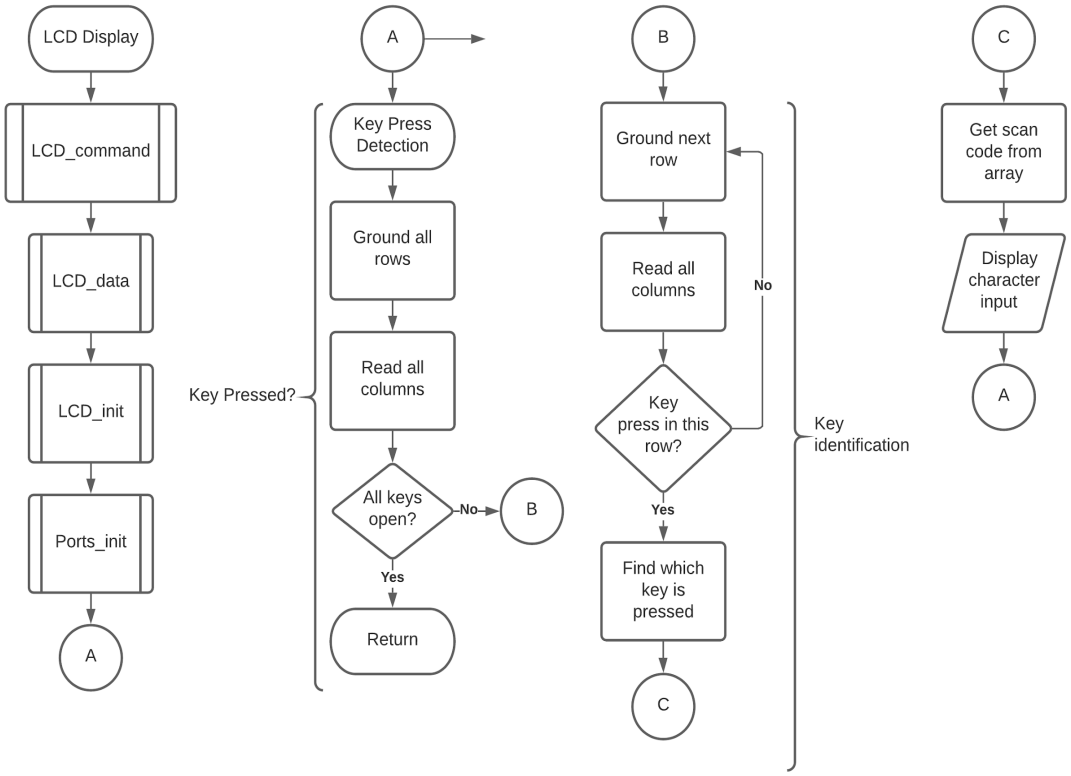
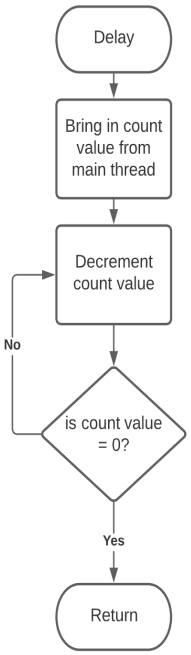


Figure 1.5 is the software flowchart for an LCD display that requires a Hex Keypad for input. Similarly to Figure 1.4, this flowchart requires the same subroutines, however displaying the input is much different. In order to properly read which key is being pressed, the program must ground all the rows and read all the columns. This is done so that when a key is pressed(pull-up resistor is grounded), the program will start looking for which key it is. In order to understand which key represents what character, the program can take the associated key value and use it as an offset when accessing an array.

Figure 1.6 - Delay Function



State Diagram

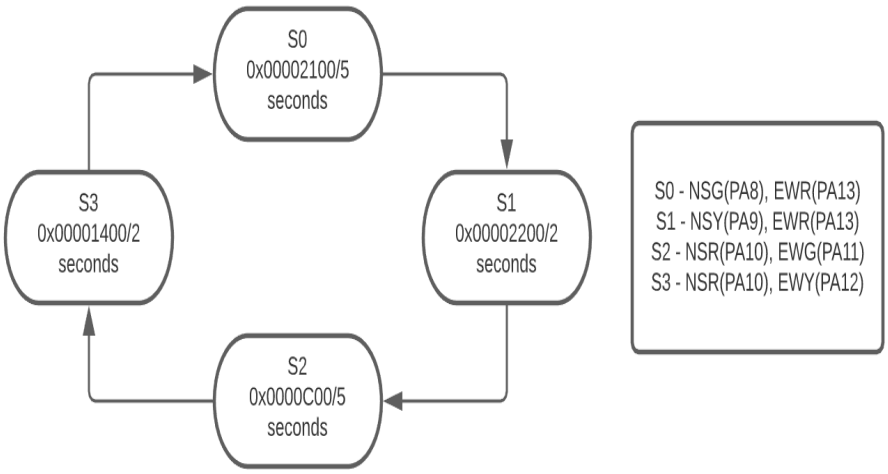


Figure 1.6 represents the steps one must take in creating a delay function. The way it works is by inputting the desired delay(in milliseconds) into the program, the program then passes that input parameter into the delay function, the function decrements the value by 1 until it reaches zero, and then it can return back to the main thread.

III. How It Works

No-Interrupts

Once the board and its components are completely wired up, and the code has been loaded into the board, one can now decide what should happen next. From the start, a message should pop up prompting the user for which mode they would like to use. Upon selection, the corresponding state will be displayed to the LCD and its function(ex. Urban Traffic Light) will begin to output to the LEDs. The states will work indefinitely until the user prompts the system with a change. One such change might be the pedestrian crossing countdown timer. If the user wishes to enable the timer, it is important to understand that the push button that starts the timer must be held down until the end of a traffic light statement(ex. NSR, EWY). This is one of the drawbacks of regular programming, as it does not allow for multithreading and saving of the push button.

Interrupts

This program maintains a number of similarities with the no-interrupt program. Much of the functions remain the same, however the way each function is called and switched between has effectively been rewritten. Once the wiring and code have all been loaded into the board, one might notice the familiar user interface with the LCD screen, as it displays each mode. Pressing a key on the keypad still chooses the mode it will switch to, however when switching between modes it is very different. With interrupts, the user can now press a mode to switch to at any point in time, as the incorporation of the NVIC controller allows a keypress to be saved. However, due to the nature of interrupts and some of its difficulties, the user must first switch over to maintenance mode before changing to another mode. This is important because at the time of writing, each interrupt function does not clear its pending flag, which allows for the program to loop continuously. It is when the user switches over to maintenance mode, the flags will be cleared and the user can select another mode without worrying that another mode might interfere with the current one(if the flags aren't cleared and two modes are selected, the device will switch back and forth between modes). On the plus side, when the user selects the crosswalk interrupt, it will be saved and executed at the end of the traffic light mode as one might expect from a real world intersection.

IV. Discussion & Conclusion

There were many considerations to be made throughout the scope of this project. Ports, hardware devices, types of logic statements and how to implement them. In this project, the author chose to design the system in such a way that PORTC on the STM32 board was utilized for all inputs and ports A & B were used for output. This design choice from the beginning allowed ease of use when inputting a value, as all ports from PORTC0-10 are freed up on the Central Processing Unit(CPU), thus eliminating any need for bit manipulation. Additionally, PORTB and PORTA were both utilized for the LCD, seven segment display, and LEDs. The use of two different ports for the display was key here, as the functions Register Select(RS), Read & Write(RW), and Enable(EN) needed to be separated from the rest of the LCD pins, as to eliminate any need for bit manipulation as well.

From this project's conception, the author chose to create as many functions as possible. This was done to make the code more readable, and to simplify the main thread. Each traffic light function contains the same structure(Figure 4.1), setting up the pins writing to a specific set of LED's, and a delay. When interfacing with a set of LEDs not much changes, like a recipe for baking a cake. Much of the same can be said when writing the functions for each interrupt(Figure 4.2), with a few important things to note. Each interrupt must go through the interrupt handler, which allows the program to break away from the main thread and return back once finished. However, in order to allow the program to dismiss the interrupt and return back to the main thread, the pending register flags must

```
105 // Urban Traffic Light
106 void urban_traffic_light(void) {
107     GPIOA->MODER = 0xF00055FF;            // clear PA8-13
108     GPIOA->MODER = 0x55555555;            // set pins to output mode (PA8-PA13)
109     GPIOC->PUPDR = 0x00150055;            // pull up PC10
110     GPIOA->ODR = 0x00002100;              // turn on NSG(PA8), EWR(PA13)
111       delayMs(5000);
112     GPIOA->ODR = 0x00002200;              // turn on NSY(PA9), EWR(PA13)
113       delayMs(2000);
114     GPIOA->ODR = 0x0000C00;               // turn on NSR(PA10), EWG(PA11)
115       delayMs(5000);
116     GPIOA->ODR = 0x00001400;              // turn on NSR(PA10), EWY(PA12)
117       delayMs(2000);
118 }
119
120 // Rural Traffic Light
121 void rural_traffic_light(void){
122     GPIOA->MODER = 0xF00055FF;            // clear PA8-13
123     GPIOA->MODER = 0x55555555;            // set pins to output mode (PA8-PA13)
124
125     GPIOA->ODR = 0x02100;                 // turn on NSG(PA8), EWR(PA13)
126       delayMs(5000);
127     while (GPIOC->IDR &= 0x0100){}        // pin 8
128     GPIOA->ODR =  0x2200;                 // turn on NSY(PA9), EWR(PA13)
129       delayMs(2000);
130     GPIOA->ODR =  0x0C00;                 // turn on NSR(PA10), EWG(PA11)
131       delayMs(5000);
132     while(GPIOC->IDR  &= 0x0200){}        // pin 9
133     GPIOA->ODR =  0x1400;                 // turn on NSR(PA10), EWY(PA12)
134       delayMs(2000);
135 }
136
137 // Blinking Yellow Light
138 void blinking_yellow(void){
139     GPIOA->MODER = 0xF00055FF;            // clear PA8-13
140     GPIOA->MODER = 0x55555555;            // set pins to output mode (PA8-PA13)
141
142     GPIOA->ODR = 0x1200;                  // Turns on NSY(PA9), EWY(PA12)
143       delayMs(2000);
144     GPIOA->ODR = 0x0000;                  // Turns off NSY(PA9), EWY(PA12)
145       delayMs(2000);
146 }
```

Figure 4.1 - Mode Functions

be cleared. As one can see from Figure 4.3, only one function(Maintenance Mode) contains the necessary statement to clear those flags(Line 89). This was done by the author in order to allow each interrupt to essentially loop, as one would expect a traffic light to do. This is why it is important to first switch to maintenance mode before switching to another, otherwise multiple modes can be called at once and the NVIC controller will switch back and forth between them unless they were given different levels of priority.

```
92 // Countdown Timer for Pedestrian Crosswalk
93 void EXTI15_10_IRQHandler(void){
94   unsigned int myCountdown[] = {0xD700, 0xFF00, 0x4700, 0xF500, 0xD500, 0xC600, 0x9F00, 0xB300, 0x0600, 0x7F00, 0x00, 0x7F00, 0x00, 0x7F00, 0x00};
95
96   GPIOA->ODR = 0x00002400;              // turn on NSR(PA10), EWR(PA13)
97     delayMs(2000);
98   for(int i=0; i<15; i++) {
99     GPIOB->ODR = myCountdown[i];
100    delayMs(1000);
101  }
102  EXTI->PR = 0x0400;                    // clear interrupt pending flag
103 }
```

Figure 4.2 - Interrupt Functions(1)

```
47 // Interrupt for Urban Traffic Light
48 void EXTI0_IRQHandler(void){
49   unsigned char mode_1[11] = "Mode: Urban";
50   LCD_command(0x01);
51   for(int k=0; k<11; k++) {
52     LCD_data(mode_1[k]);                    // Prints Urban mode
53   }
54   urban_traffic_light();                    // Starts Urban Traffic Light
55 }
56
57 // Interrupt for Rural Traffic Light
58 void EXTI1_IRQHandler(void){
59   unsigned char mode_2[11] = "Mode: Rural";
60   LCD_command(0x01);
61   for(int k=0; k<11; k++) {
62     LCD_data(mode_2[k]);                    // Prints Rural mode
63   }
64   rural_traffic_light();                    // Starts Rural Traffic Light
65 }
66
67 // Interrupt for Blinking Yellow
68 void EXTI2_IRQHandler(void){
69   unsigned char mode_3[20] = "Mode:Blinking Yellow";
70   LCD_command(0x01);
71   for(int k=0; k<14; k++) {
72     LCD_data(mode_3[k]);                    // Prints Blinking mode
73   }
74   LCD_command(0xC0);
75   for(int k=14; k<20; k++) {
76     LCD_data(mode_3[k]);
77   }
78   blinking_yellow();
79 }
80
81 // Interrupt for Maintanence Mode
82 void EXTI3_IRQHandler(void){
83   unsigned char mode_4[16] = "Mode:Maintanence";
84   LCD_command(0x01);
85   for(int k=0; k<16; k++) {
86     LCD_data(mode_4[k]);                    // Prints Maintanence mode
87   }
88   maintenance_mode();
89   EXTI->PR = 0x000F;
90 }
```

Figure 4.3 - Interrupt Functions(2)

Figure 4.4 showcases how the author decided to go about the port initializations. It is important to set up both the clocks associated with GPIO as well as SYSCFG when utilizing interrupts(Lines 159 & 160), as without a clock no useful information can be sent from the board. Another important thing to remember when setting up interrupts is to disable global interrupt requests(IRQs) during initialization. This is done so that execution of some critical tasks like manipulating a common pointer shared by multiple threads can be masked. Like setting up the interrupt and traffic light functions, initializing interrupts is like baking a cake. To complete the configuration, one must follow the procedure:

1. Configure the Px.y pins to generate interrupts, using GPIOx_MODER
2. Clear and set the bits of the SYSCFG_EXTICR
3. Enable the mask bit of the interrupt line in EXTI_IMR register
4. Select the trigger by the selection bits of the EXTI_FTSR(Falling Trigger Selection Register)
5. Set the priority of the EXTI line if necessary
6. Enable the interrupt in NVIC registers

This procedure can be seen in Figure 4.4 from lines 167-189, as well as re-enabling global IRQs on line 197.

```
156 // Port Initializations. PA5-R/S, PA6-R/W, PA7-EN, PB0-PB7 for D0-D7, respectively and Port C for Interrupts.
157 void PORTS_init(void) {
158
159     RCC->AHB1ENR = 7;                   // enable GPIOA/B/C clock
160     RCC->APB2ENR = 0x4000;              // enable SYSCFG clock
161
162     __disable_irq();                    // global disable IRQs
163
164     GPIOC->MODER = 0xFF000000;          // clear pin mode to input
165     GPIOC->PUPDR = 0x00150055;          // enable pull up resistors for column pins, crosswalk & pressure pads
166
167     SYSCFG->EXTICR[0] = 0xFFF0;         // clear port selection for EXTI0-3
168     SYSCFG->EXTICR[0] = 0x2222;         // select port C for EXTI0-3
169
170     EXTI->IMR = 0x000F;                 // unmask EXTI0-3
171     EXTI->FTSR = 0x000F;                // select falling edge trigger
172
173     NVIC_SetPriority(EXTI0_IRQn, 1);    // set priority interrupt for push button PC0
174     NVIC_SetPriority(EXTI1_IRQn, 1);    // set priority interrupt for push button PC1
175     NVIC_SetPriority(EXTI2_IRQn, 1);    // set priority interrupt for push button PC2
176     NVIC_SetPriority(EXTI3_IRQn, 1);    // set priority interrupt for push button PC3
177     NVIC_EnableIRQ(EXTI0_IRQn);         // enable interrupt in NVIC
178     NVIC_EnableIRQ(EXTI1_IRQn);         // enable interrupt in NVIC
179     NVIC_EnableIRQ(EXTI2_IRQn);         // enable interrupt in NVIC
180     NVIC_EnableIRQ(EXTI3_IRQn);         // enable interrupt in NVIC
181
182     SYSCFG->EXTICR[2] = 0xF0FF;         // clear port selection for EXTI10
183     SYSCFG->EXTICR[2] = 0x0200;         // select port C for EXTI10
184
185     EXTI->IMR |= 0x0400;                // unmask EXTI10
186     EXTI->FTSR |= 0x0400;               // select falling edge trigger
187
188     NVIC_SetPriority(EXTI15_10_IRQn, 1);  // set priority interrupt for push button
189     NVIC_EnableIRQ(EXTI15_10_IRQn);       // enable interrupt in NVIC
190
191     GPIOA->MODER = 0;                   // clear pin mode
192     GPIOA->MODER = 0x55555555;          // set pin output mode
193     GPIOA->BSRR  = 0x00C00000;          // turn off EN and R/W
194     GPIOB->MODER = 0;                   // clear pin mode
195     GPIOB->MODER = 0x55555555;          // set pin output mode
196
197     __enable_irq();                     // global enable IRQs
198 }
199
```

Figure 4.4 - Port Initializations

Figure 4.5 shows how the programmer must go through the necessary steps of using an LCD, as set by the manufacturer Hitachi. Line 201 shows the function LCD_init, and in this initialization sequence a series of commands are sent to the LCD controller to set the 8-bit data line, move the cursor, clear the screen and turn on the display. Functions LCD_command & LCD_data on lines 216 & 229 are very similar in that they take some parameter they've been given, set the proper RS & RW pins, place the

parameter on the Output Data Register(ODR) and pulse EN high and low so the information can latch in to the LCD.

```
200 // Initialize LCD controller
201 void LCD_init(void) {
202     delayMs(30);            // initialization sequence
203     LCD_command(0x30);
204     delayMs(10);
205     LCD_command(0x30);
206     delayMs(1);
207     LCD_command(0x30);
208
209     LCD_command(0x38);      // set 8-bit data, 2-line, 5x7 font
210     LCD_command(0x06);      // move cursor right after each char
211     LCD_command(0x01);      // clear screen, move cursor to home
212     LCD_command(0x0F);      // turn on display, cursor blinking
213 }
214
215 // Send command to LCD
216 void LCD_command(unsigned char command) {
217     GPIOA->BSRR = (RS | RW) << 16;      // RS = 0, R/W = 0
218     GPIOB->ODR = command;               // put command on data bus
219     GPIOA->BSRR = EN;                   // pulse E high
220     delayMs(0);
221     GPIOA->BSRR = EN << 16;             // clear E
222     if (command < 4)
223         delayMs(2);                     // command 1 and 2 needs up to 1.64ms
224     else
225         delayMs(1);
226 }
227
228 // Write data to the LCD
229 void LCD_data(char datawrite) {
230     GPIOA->BSRR = RS;                   // RS = 1
231     GPIOA->BSRR = RW << 16;             // R/W = 0
232     GPIOB->ODR = datawrite;             // put data on data bus
233     GPIOA->BSRR = EN;                   // pulse E high
234     delayMs(0);
235     GPIOA->BSRR = EN << 16;             // clear E
236     delayMs(1);
237 }
```

Figure 4.5 - LCD Initializations

Figure 4.6 is a very cut and dry way of identifying a keypress from the keypad and associating that key to a value so that the programmer can make it useful. As explained in Figure 1.5, in order to properly read which key is being pressed, the program grounds all the rows and reads all the columns. This is done so that when a key is pressed(pull-up resistor is grounded - shown on line 246), the program will start looking for which key it is(Line 255). In order to understand which key represents what character, the program can take the associated key value(Line 271) and use it as an offset when accessing an array.

```
239 // Get key from keypad
240 char keypad_getkey(void) {
241     int row, col;
242     const int row_mode[] = {0x00000100, 0x00000400, 0x00001000, 0x00004000}; // one row is output
243     const int row_low[] =  {0x00100000, 0x00200000, 0x00400000, 0x00800000}; // one row is low
244     const int row_high[] = {0x00000010, 0x00000020, 0x00000040, 0x00000080}; // one row is high
245
246     // check to see any key pressed
247     GPIOC->MODER = 0x00005500;          // make all row pins output
248     GPIOC->BSRR =  0x00F00000;          // drive all row pins low
249     delayMs(1);                         // wait for signals to settle
250     col = GPIOC->IDR & 0x000F;          // read all column pins
251     GPIOC->MODER &= ~0x0000FF00;        // disable all row pins drive
252     if (col == 0x000F)                  // if all columns are high
253         return 0;                       // no key pressed
254
255 // If a key is pressed, it gets here to find out which key. It activates one row at a time and
256 // read the input to see which column is active.
257     for (row = 0; row < 4; row++) {
258         GPIOC->MODER &= ~0x0000FF00;    // disable all row pins drive
259         GPIOC->MODER |= row_mode[row];  // enable one row at a time
260         GPIOC->BSRR = row_low[row];     // drive the active row low
261         delayMs(1);                     // wait for signal to settle
262         col = GPIOC->IDR & 0x000F;      // read all columns
263         GPIOC->BSRR = row_high[row];    // drive the active row high
264         if (col != 0x000F) break;       // if one of the input is low, some key is pressed.
265     }
266     GPIOC->BSRR = 0x000000F0;           // drive all rows high before disable them
267     GPIOC->MODER &= ~0x0000FF00;        // disable all rows
268     if (row == 4)
269         return 0;                       // if we get here, no key is pressed
270
271     // gets here when one of the rows has key pressed, check which column it is
272     if (col == 0x000E) return row * 4 + 1;    // key in column 0
273     if (col == 0x000D) return row * 4 + 2;    // key in column 1
274     if (col == 0x000B) return row * 4 + 3;    // key in column 2
275     if (col == 0x0007) return row * 4 + 4;    // key in column 3
276
277     return 0;   // just to be safe
278 }
```

Figure 4.6 - Keypad Function

Lastly, Figure 4.7 contains a few functions that maintain a good quality of life for any user. Line 281 starts the function Mode_Question, and in this mode is an array and some for loops utilized to display which key corresponds to which function. In conjunction with this is the function scroll_and_update_cursor located on line 294. When the main thread calls these two functions, it allows the LCD to scroll through the modes and return back to start, since there is only a limited amount of space on the LCD. Line 307 contains the systick delay function, and as explained in Figure 1.6, decrements a count value given by the programmer until it reaches zero.

```
280 // Mode Question
281 void Mode_Question(void){
282   unsigned char mode_question[60] = "Select mode: 1.Urban 2.Rural 3.Blinking Yellow 4.Maintanence";
283 
284   for(int k=0; k<28; k++){
285       LCD_data(mode_question[k]);             // Prints Mode Question
286   }
287   LCD_command(0xC0);
288   for(int k=29; k<60; k++){
289       LCD_data(mode_question[k]);
290   }
291 }
292 
293 // scroll and update cursor
294 int scroll_and_update_cursor(int scroll_cursor) {
295     delayMs(400);
296     LCD_command(0x18);
297     scroll_cursor += 1;
298     if(scroll_cursor == 16) {
299       delayMs(3000);
300       LCD_command(0x02);
301       scroll_cursor = 0;
302     }
303     return scroll_cursor;
304 }
305 
306 // Delay timer
307 void delayMs(int n) {
308     int i;
309 
310     // Configure SysTick
311     SysTick->LOAD = 16000;  // reload with number of clocks per millisecond
312     SysTick->VAL = 0;       // clear current value register
313     SysTick->CTRL = 0x5;    // Enable the timer
314 
315     for(i = 0; i < n; i++) {
316         while((SysTick->CTRL & 0x10000) == 0) // wait until the COUNTFLAG is set
317             { }
318     }
319     SysTick->CTRL = 0;      // Stop the timer (Enable = 0)
320 }
321 
```

Figure 4.7 - Mode Question Function, Scrolling Function, Delay Timer

In summary, most of this project requires a good amount of forethought and planning. Understanding from the beginning which ports one will use for which device, as well as the way each function will work, allows for a project of a larger scope to be simplified and broken down into its elements. This is why a hardware diagram as well as software flowcharts are critical to make from the start, before any code is written. Additionally, choosing which method(interrupts or no-interrupts) requires some design, as they effectively work differently from each other. At the time of writing, the author understands the drawbacks of each method and would like to note that given more time and access to resources, a more well thought out and efficient code could be written, such as one that has an interrupt function return back a value to the main thread and thus could locally clear the pending flags for each interrupt(based on the assumption that it is possible). Lastly, searching for a set of programming that could prove to be even more effective or efficient than using interrupts, like multithreading, could indeed prove itself to be a worthwhile project for future research.

V. Citations

1. M. A. Mazidi, S. Chen, and E. Ghaemi, *STM32 arm programming for embedded systems using C Language with STM32 Nucleo*. Place of publication not identified: Mazidi, 2018.

VI. Appendix

Code for Real World Intersection in C(No Interrupts):

```
// Kevin Nelson
// Final Project
// Microcontrollers - Fall 2020
// Real World Intersection - No Interrupts

#include "stm32f4xx.h"

#define RS 0x20                                                 // PA5 mask for reg select
#define RW 0x40                                                 // PA6 mask for read/write
#define EN 0x80                                                 // PA7 mask for enable

void delayMs(int n);
void LCD_command(unsigned char command);
void LCD_data(char datawrite);
void LCD_init(void);
void PORTS_init(void);
void urban_traffic_light(void);
void rural_traffic_light(void);
void blinking_yellow(void);
void crosswalk_countdown(void);
int  scroll_and_update_cursor(int);
char keypad_getkey(void);

int main(void) {

        int key;
        int scroll_cursor = 0;
        unsigned char mode_question[46] = "Select mode: 1.Urban 2.Rural 3.Blinking Yellow";
        unsigned char mode_1[11] = "Mode: Urban";
        unsigned char mode_2[11] = "Mode: Rural";
        unsigned char mode_3[20] = "Mode:Blinking Yellow";

        RCC->AHB1ENR = 7;                                       // enable GPIOA/B/C clock
        RCC->APB2ENR = 0x4000;                                  // enable SYSCFG clock
        LCD_init();                                             // initialize LCD controller

        for(int k=0; k<28; k++){
                LCD_data(mode_question[k]);                     // Prints Mode Question
        }
        LCD_command(0xC0);
        for(int k=29; k<46; k++){
                LCD_data(mode_question[k]);
        }
        key = 0;
        int updated_key = 0;
        int should_scroll = 1;                                  // variable to see if the LCD should scroll
        int should_print_mode = 1;
        while(1) {
                updated_key = keypad_getkey();
                if ((updated_key == 1 || updated_key == 2 || updated_key == 3) && updated_key != key) {
                        should_print_mode = 1;
                        key = updated_key;
                }
                if (should_scroll) {
                  scroll_cursor = scroll_and_update_cursor(scroll_cursor);
                }
                if (key != 1 && key != 2 && key!=3) {
                        continue;
                }

```

```
61.                         LCD_command(0x01);
62.                         should_scroll = 0;
63.                         if (key == 1)
64.                                 {                                       // if Urban Mode is selected
65.                                 if (should_print_mode) {                // Prints when 1(1 at the start)
66.                                   for(int k=0; k<11; k++) {
67.                                     LCD_data(mode_1[k]);                // Prints Urban mode
68.                             }
69.                                   should_print_mode = 0;                // stops mode_1 from printing
70.                                 }
71.                                 urban_traffic_light();                  // Starts Urban Traffic Light
72.                         }
73.                         else if (key == 2) {                            // if Rural Mode is selected
74.                                 if (should_print_mode) {                // Prints when 1(1 at the start)
75.                                   for(int k=0; k<11; k++) {
76.                                           LCD_data(mode_2[k]);          // Prints Rural mode
77.                     }
78.                                   should_print_mode = 0;                // stops mode_1 from printing
79.                                 }
80.                                 rural_traffic_light();                  // Starts Rural Traffic Light
81.                         }
82.                         else if (key == 3) {                            // if Rural Mode is selected
83.                                 if (should_print_mode) {                // Prints when 1(1 at the start)
84.                                         for(int k=0; k<14; k++) {
85.                                                 LCD_data(mode_3[k]);    // Prints Blinking mode
86.                                         }
87.                                         LCD_command(0xC0);
88.                                         for(int k=14; k<20; k++) {
89.                                                 LCD_data(mode_3[k]);
90.                                         }
91.                                   should_print_mode = 0;                // stops mode_1 from printing
92.                                 }
93.                                 blinking_yellow();                      // Starts Blinking Yellow Light
94.                         }
95.                         if((GPIOC->IDR & 0x0400) != 0x0400){    // If crosswalk push button is pushed
96.                                 crosswalk_countdown();                  // Start displaying countdown
97.                         }
98.                 }
99.   }
100.
101.  // scroll and update cursor
102.  int scroll_and_update_cursor(int scroll_cursor) {
103.                        delayMs(400);
104.                        LCD_command(0x18);
105.                        scroll_cursor += 1;
106.                        if(scroll_cursor == 12) {
107.                                delayMs(3000);
108.                                LCD_command(0x02);
109.                                scroll_cursor = 0;
110.                        }
111.                        return scroll_cursor;
112.  }
113.
114.  // Urban Traffic Light
115.  void urban_traffic_light(void) {
116.                GPIOA->MODER = 0xF00055FF;                              // clear PA8-13
117.                GPIOA->MODER = 0x55555555;                              // set pins to output mode (PA8-PA13)
118.                GPIOC->PUPDR = 0x00150055;                              // pull up PC10
119.                GPIOA->ODR = 0x2100;                                    // turn on NSG(PA8), EWR(PA13)
120.                        delayMs(5000);
121.                if((GPIOC->IDR & 0x0400) != 0x0400){
122.                                GPIOA->ODR = 0x2200;                    // turn on NSY(PA9), EWR(PA13)
123.                                delayMs(2000);
124.                                crosswalk_countdown();
125.                        return;
```

```
126.            }
127.            GPIOA->ODR = 0x2200;                                    // turn on NSY(PA9), EWR(PA13)
128.                    delayMs(2000);
129.            if((GPIOC->IDR & 0x0400) != 0x0400){
130.                    crosswalk_countdown();
131.            }
132.            GPIOA->ODR = 0x0C00;                                    // turn on NSR(PA10), EWG(PA11)
133.                    delayMs(5000);
134.            if((GPIOC->IDR & 0x0400) != 0x0400){
135.                    GPIOA->ODR = 0x1400;                            // turn on NSR(PA10), EWY(PA12)
136.                    delayMs(2000);
137.                    crosswalk_countdown();
138.                    return;
139.            }
140.            GPIOA->ODR = 0x1400;                                    // turn on NSR(PA10), EWY(PA12)
141.                    delayMs(2000);
142.  }
143.
144.
145.  // Rural Traffic Light
146.  void rural_traffic_light(void){
147.            GPIOA->MODER = 0xF00055FF;                      // clear PA8-13
148.            GPIOA->MODER = 0x55555555;                      // set pins to output mode (PA8-PA13)
149.
150.            GPIOA->ODR = 0x2100;                            // turn on NSG(PA8), EWR(PA13)
151.                    delayMs(5000);
152.            if((GPIOC->IDR & 0x0400) != 0x0400){
153.                    GPIOA->ODR = 0x00002200;                // turn on NSY(PA9), EWR(PA13)
154.                    delayMs(2000);
155.                    crosswalk_countdown();
156.                    return;
157.            }
158.            while (GPIOC->IDR   &= 0x0100){                 // pin 8
159.                    if((GPIOC->IDR & 0x0400) != 0x0400){
160.                            GPIOA->ODR = 0x2200;             // turn on NSY(PA9), EWR(PA13)
161.                            delayMs(2000);
162.                            crosswalk_countdown();
163.                    }
164.            GPIOA->ODR = 0x2100;                            // turn on NSG(PA8), EWR(PA13)
165.                    delayMs(5000);
166.            }
167.                    GPIOA->ODR =  0x2200;                           // turn on NSY(PA9), EWR(PA13)
168.                            delayMs(2000);
169.                    if((GPIOC->IDR & 0x0400) != 0x0400){
170.                            crosswalk_countdown();
171.                    }
172.                    GPIOA->ODR =  0x0C00;                   // turn on NSR(PA10), EWG(PA11)
173.                            delayMs(5000);
174.                    while(GPIOC->IDR &= 0x0200){                    // pin 9
175.                            if((GPIOC->IDR & 0x0400) != 0x0400){
176.                                    GPIOA->ODR =  0x1400;            // turn on NSR(PA10), EWY(PA12)
177.                                    delayMs(2000);
178.                                    crosswalk_countdown();
179.                    }
180.                    GPIOA->ODR =  0x0C00;                   // turn on NSR(PA10), EWG(PA11)
181.                            delayMs(5000);
182.                    }
183.                    if((GPIOC->IDR & 0x0400) != 0x0400){
184.                            GPIOA->ODR = 0x1400;                    // turn on NSR(PA10), EWY(PA12)
185.                            delayMs(2000);
186.                            crosswalk_countdown();
187.                            return;
188.                    }
189.                    GPIOA->ODR =  0x1400;                           // turn on NSR(PA10), EWY(PA12)
190.                            delayMs(2000);
```

```
191.  }
192.
193.
194.  // Countdown Timer for Pedestrian Crosswalk
195.  void crosswalk_countdown(void){
196.  unsigned int myCountdown[] = {0xD700, 0xFF00, 0x4700, 0xF500, 0xD500, 0xC600, 0x9F00, 0xB300, 0x0600, 0x7F00,
      0x00, 0x7F00, 0x00, 0x7F00};
197.
198.              GPIOA->ODR = 0x2400;                              // turn on NSR(PA10), EWR(PA13)
199.                        delayMs(2000);
200.              for(int i=0; i<14; i++) {
201.                        GPIOB->ODR = myCountdown[i];
202.                        delayMs(1000);
203.              }
204.  }
205.
206.  // Blinking Yellow Light
207.  void blinking_yellow(void){
208.                        GPIOA->MODER = 0xF00055FF;                // clear PA8-13
209.                        GPIOA->MODER = 0x55555555;                // set pins to output mode (PA8-PA13)
210.
211.                        GPIOA->ODR = 0x1200;                      // Turns on NSY(PA9), EWY(PA12)
212.                                  delayMs(2000);
213.                        GPIOA->ODR = 0x0000;                      // Turns off NSY(PA9), EWY(PA12)
214.                                  delayMs(2000);
215.  }
216.
217.  // initialize port pins then initialize LCD controller
218.  void LCD_init(void) {
219.     PORTS_init();
220.
221.     delayMs(30);                        // initialization sequence
222.     LCD_command(0x30);
223.     delayMs(10);
224.     LCD_command(0x30);
225.     delayMs(1);
226.     LCD_command(0x30);
227.
228.     LCD_command(0x38);          // set 8-bit data, 2-line, 5x7 font
229.     LCD_command(0x06);          // move cursor right after each char
230.     LCD_command(0x01);          // clear screen, move cursor to home
231.     LCD_command(0x0F);          // turn on display, cursor blinking
232.  }
233.
234.  // Port Initializations for LCD. PA5-R/S, PA6-R/W, PA7-EN, PB0-PB7 for D0-D7, respectively.
235.  void PORTS_init(void) {
236.     GPIOA->MODER = 0;                                                    // clear pin mode
237.     GPIOA->MODER = 0x55555555;                                           // set pin output mode
238.     GPIOA->BSRR  = 0x00C00000;                                           // turn off EN and R/W
239.     GPIOB->MODER = 0;                                                    // clear pin mode
240.     GPIOB->MODER = 0x55555555;                                           // set pin output mode
241.                        GPIOC->MODER = 0xFF000000;                        // clear pin mode to input
242.     GPIOC->PUPDR = 0x00150055;          // enable pull up resistors for column pins, crosswalk & pressure pads
243.  }
244.
245.  // Send command to LCD
246.  void LCD_command(unsigned char command) {
247.     GPIOA->BSRR = (RS | RW) << 16;                  // RS = 0, R/W = 0
248.     GPIOB->ODR = command;                           // put command on data bus
249.     GPIOA->BSRR = EN;                               // pulse E high
250.     delayMs(0);
251.     GPIOA->BSRR = EN << 16;                         // clear E
252.     if (command < 4)
253.        delayMs(2);                                  // command 1 and 2 needs up to 1.64ms
254.     else
```

```
255.       delayMs(1);
256. }
257.
258. // Write data to the LCD
259. void LCD_data(char datawrite) {
260.    GPIOA->BSRR = RS;                              // RS = 1
261.    GPIOA->BSRR = RW << 16;                         // R/W = 0
262.    GPIOB->ODR = datawrite;                         // put data on data bus
263.    GPIOA->BSRR = EN;                              // pulse E high
264.    delayMs(0);
265.    GPIOA->BSRR = EN << 16;                        // clear E
266.    delayMs(1);
267. }
268.
269. // Get key from keypad
270. char keypad_getkey(void) {
271.    int row, col;
272.    const int row_mode[] = {0x00000100, 0x00000400, 0x00001000, 0x00004000}; // one row is output
273.    const int row_low[] =  {0x00100000, 0x00200000, 0x00400000, 0x00800000}; // one row is low
274.    const int row_high[] = {0x00000010, 0x00000020, 0x00000040, 0x00000080}; // one row is high
275.
276.    // check to see any key pressed
277.    GPIOC->MODER = 0x00005500;          // make all row pins output
278.    GPIOC->BSRR =  0x00F00000;          // drive all row pins low
279.    delayMs(1);                         // wait for signals to settle
280.    col = GPIOC->IDR & 0x000F;          // read all column pins
281.    GPIOC->MODER &= ~0x0000FF00;        // disable all row pins drive
282.    if (col == 0x000F)                   // if all columns are high
283.       return 0;                        // no key pressed
284.
285. // If a key is pressed, it gets here to find out which key. It activates one row at a time and
286. // read the input to see which column is active.
287.    for (row = 0; row < 4; row++) {
288.       GPIOC->MODER &= ~0x0000FF00;              // disable all row pins drive
289.       GPIOC->MODER |= row_mode[row];            // enable one row at a time
290.       GPIOC->BSRR = row_low[row];              // drive the active row low
291.       delayMs(1);                               // wait for signal to settle
292.       col = GPIOC->IDR & 0x000F;                // read all columns
293.       GPIOC->BSRR = row_high[row];              // drive the active row high
294.       if (col != 0x000F) break;                // if one of the input is low, some key is pressed.
295.    }
296.    GPIOC->BSRR = 0x000000F0;                    // drive all rows high before disable them
297.    GPIOC->MODER &= ~0x0000FF00;                 // disable all rows
298.    if (row == 4)
299.       return 0;                                // if we get here, no key is pressed
300.
301.    // gets here when one of the rows has key pressed, check which column it is
302.    if (col == 0x000E) return row * 4 + 1;       // key in column 0
303.    if (col == 0x000D) return row * 4 + 2;       // key in column 1
304.    if (col == 0x000B) return row * 4 + 3;       // key in column 2
305.    if (col == 0x0007) return row * 4 + 4;       // key in column 3
306.
307.    return 0;                                    // just to be safe
308. }
309.
310. // Delay timer
311. void delayMs(int n) {
312.    int i;
313.
314.    // Configure SysTick
315.    SysTick->LOAD = 16000;                       // reload with number of clocks per millisecond
316.    SysTick->VAL = 0;                            // clear current value register
317.    SysTick->CTRL = 0x5;                          // Enable the timer
318.
319.    for(i = 0; i < n; i++) {
```

```
320.        while((SysTick->CTRL & 0x10000) == 0)          // wait until the COUNTFLAG is set
321.          { }
322.    }
323.    SysTick->CTRL = 0;                                 // Stop the timer (Enable = 0)
324. }
```

Code for Real World Intersection in C(Interrupts):

```
1.    // Kevin Nelson
2.    // Final Project
3.    // Microcontrollers - Fall 2020
4.    // Real World Intersection - Interrupts
5.
6.    #include "stm32f4xx.h"
7.
8.    #define RS 0x20                                                    // PA5 mask for reg select
9.    #define RW 0x40                                                    // PA6 mask for read/write
10.   #define EN 0x80                                                    // PA7 mask for enable
11.
12.   void delayMs(int n);
13.   void LCD_command(unsigned char command);
14.   void LCD_data(char datawrite);
15.   void LCD_init(void);
16.   void PORTS_init(void);
17.   void urban_traffic_light(void);
18.   void rural_traffic_light(void);
19.   void blinking_yellow(void);
20.   void maintenance_mode(void);
21.   void EXTI0_IRQHandler(void);
22.   void EXTI1_IRQHandler(void);
23.   void EXTI2_IRQHandler(void);
24.   void EXTI3_IRQHandler(void);
25.   void EXTI15_10_IRQHandler(void);
26.   void Mode_Question(void);
27.   char keypad_getkey(void);
28.   int  scroll_and_update_cursor(int);
29.
30.   int main(void) {
31.   int scroll_cursor = 0;
32.   int should_scroll = 1;
33.
34.   PORTS_init();                                                      // Port Initialization - Line 156
35.   LCD_init();                                                        // Initialize LCD controller - Line 200
36.   Mode_Question();                                                   // Prints Mode Question - Line 280
37.
38.   while(1) {
39.   keypad_getkey();                                                   // Gets Keypress - Line 238
40.   if (should_scroll) {
41.   scroll_cursor = scroll_and_update_cursor(scroll_cursor);
42.   }
43.   }
44.   }
45.
46.   // Interrupt for Urban Traffic Light
```

```
47.  void EXTI0_IRQHandler(void){
48.  unsigned char mode_1[11] = "Mode: Urban";
49.  LCD_command(0x01);
50.  for(int k=0; k<11; k++) {
51.  LCD_data(mode_1[k]);                                   // Prints Urban mode
52.  }
53.  urban_traffic_light();                                 // Starts Urban Traffic Light
54.  }
55.
56.  // Interrupt for Rural Traffic Light
57.  void EXTI1_IRQHandler(void){
58.  unsigned char mode_2[11] = "Mode: Rural";
59.  LCD_command(0x01);
60.  for(int k=0; k<11; k++) {
61.  LCD_data(mode_2[k]);                                   // Prints Rural mode
62.  }
63.  rural_traffic_light();                                 // Starts Rural Traffic Light
64.  }
65.
66.  // Interrupt for Blinking Yellow
67.  void EXTI2_IRQHandler(void){
68.  unsigned char mode_3[20] = "Mode:Blinking Yellow";
69.  LCD_command(0x01);
70.  for(int k=0; k<14; k++) {
71.  LCD_data(mode_3[k]);                                   // Prints Blinking mode
72.  }
73.  LCD_command(0xC0);
74.  for(int k=14; k<20; k++) {
75.  LCD_data(mode_3[k]);
76.  }
77.  blinking_yellow();
78.  }
79.
80.  // Interrupt for Maintenance Mode
81.  void EXTI3_IRQHandler(void){
82.  unsigned char mode_4[16] = "Mode:Maintenance";
83.  LCD_command(0x01);
84.  for(int k=0; k<16; k++) {
85.  LCD_data(mode_4[k]);                                   // Prints Maintenance mode
86.  }
87.  maintenance_mode();
88.  EXTI->PR = 0x000F;
89.  }
90.
91.  // Countdown Timer for Pedestrian Crosswalk
92.  void EXTI15_10_IRQHandler(void){
93.  unsigned int myCountdown[] = {0xD700, 0xFF00, 0x4700, 0xF500, 0xD500, 0xC600, 0x9F00, 0xB300, 0x0600, 0x7F00,
     0x00, 0x7F00, 0x00, 0x7F00, 0x00};
94.
95.  GPIOA->ODR = 0x00002400;                               // turn on NSR(PA10), EWR(PA13)
96.  delayMs(2000);
97.  for(int i=0; i<15; i++) {
98.  GPIOB->ODR = myCountdown[i];
99.  delayMs(1000);
100. }
101. EXTI->PR = 0x0400;                                     // clear interrupt pending flag
```

```
}

// Urban Traffic Light
void urban_traffic_light(void) {
GPIOA->MODER = 0xF00055FF;                                  // clear PA8-13
GPIOA->MODER = 0x55555555;                                  // set pins to output mode (PA8-PA13)
GPIOC->PUPDR = 0x00150055;                                  // pull up PC10
GPIOA->ODR = 0x00002100;                                    // turn on NSG(PA8), EWR(PA13)
delayMs(5000);
GPIOA->ODR = 0x00002200;                                    // turn on NSY(PA9), EWR(PA13)
delayMs(2000);
GPIOA->ODR = 0x0000C00;                                     // turn on NSR(PA10), EWG(PA11)
delayMs(5000);
GPIOA->ODR = 0x00001400;                                    // turn on NSR(PA10), EWY(PA12)
delayMs(2000);
}

void rural_traffic_light(void){
                GPIOA->MODER = 0xF00055FF;                  // clear PA8-13
                GPIOA->MODER = 0x55555555;                  // set pins to output mode (PA8-PA13)

                GPIOA->ODR = 0x02100;
    // turn on NSG(PA8), EWR(PA13)
                        delayMs(5000);
                while (GPIOC->IDR   &= 0x0100){             // pin 8
                        if((GPIOC->IDR & 0x0400) != 0x0400){
                                GPIOA->ODR = 0x00002200;    // turn on NSY(PA9), EWR(PA13)
                                delayMs(2000);
                                EXTI15_10_IRQHandler();
                                GPIOA->ODR = 0x02100;
                // turn on NSG(PA8), EWR(PA13)
                                delayMs(5000);
                        }
                }
                GPIOA->ODR =  0x2200;                       // turn on NSY(PA9), EWR(PA13)
                        delayMs(2000);
                GPIOA->ODR =  0x0C00;                       // turn on NSR(PA10), EWG(PA11)
                        delayMs(5000);
                while(GPIOC->IDR    &= 0x0200){             // pin 9
                        if((GPIOC->IDR & 0x0400) != 0x0400){
                                GPIOA->ODR =  0x00001400;   // turn on NSR(PA10), EWY(PA12)
                                delayMs(2000);
                                EXTI15_10_IRQHandler();
                                GPIOA->ODR =  0x0C00;       // turn on NSR(PA10), EWG(PA11)
                                delayMs(5000);
                        }
                GPIOA->ODR =  0x1400;                       // turn on NSR(PA10), EWY(PA12)
                        delayMs(2000);
        }
}


// Blinking Yellow Light
void blinking_yellow(void){
GPIOA->MODER = 0xF00055FF;                                  // clear PA8-13
GPIOA->MODER = 0x55555555;                                  // set pins to output mode (PA8-PA13)
```

```
156.  GPIOA->ODR = 0x1200;                                   // Turns on NSY(PA9), EWY(PA12)
157.  delayMs(2000);
158.  GPIOA->ODR = 0x0000;                                   // Turns off NSY(PA9), EWY(PA12)
159.  delayMs(2000);
160.  }
161.
162.  // Maintenance Mode
163.  void maintenance_mode(void){
164.  GPIOA->MODER = 0xF00055FF;                             // clear PA8-13
165.  GPIOA->MODER = 0x55555555;                             // set pins to output mode (PA8-PA13)
166.  GPIOA->ODR = 0x2400;                                   // turn on NSR(PA8), EWR(PA13)
167.  }
168.
169.  // Port Initializations. PA5-R/S, PA6-R/W, PA7-EN, PB0-PB7 for D0-D7, respectively and Port C for Interrupts.
170.  void PORTS_init(void) {
171.
172.  RCC->AHB1ENR = 7;                                      // enable GPIOA/B/C clock
173.  RCC->APB2ENR = 0x4000;                                 // enable SYSCFG clock
174.
175.  __disable_irq();                                       // global disable IRQs
176.
177.  GPIOC->MODER = 0xFF000000;                             // clear pin mode to input
178.  GPIOC->PUPDR = 0x00150055;          // enable pull up resistors for column pins, crosswalk & pressure pads
179.
180.  SYSCFG->EXTICR[0] = 0xFFF0;                            // clear port selection for EXTI0-3
181.  SYSCFG->EXTICR[0] = 0x2222;                            // select port C for EXTI0-3
182.
183.  EXTI->IMR = 0x000F;                                    // unmask EXTI0-3
184.  EXTI->FTSR = 0x000F;                                   // select falling edge trigger
185.  NVIC_SetPriority(EXTI0_IRQn, 1);                       // set priority interrupt for push button PC0
186.  NVIC_SetPriority(EXTI1_IRQn, 1);                       // set priority interrupt for push button PC1
187.  NVIC_SetPriority(EXTI2_IRQn, 1);                       // set priority interrupt for push button PC2
188.  NVIC_SetPriority(EXTI3_IRQn, 1);                       // set priority interrupt for push button PC3
189.
190.  NVIC_EnableIRQ(EXTI0_IRQn);                            // enable interrupt in NVIC
191.  NVIC_EnableIRQ(EXTI1_IRQn);                            // enable interrupt in NVIC
192.  NVIC_EnableIRQ(EXTI2_IRQn);                            // enable interrupt in NVIC
193.  NVIC_EnableIRQ(EXTI3_IRQn);                            // enable interrupt in NVIC
194.
195.  SYSCFG->EXTICR[2] = 0xF0FF;                            // clear port selection for EXTI10
196.  SYSCFG->EXTICR[2] = 0x0200;                            // select port C for EXTI10
197.
198.  EXTI->IMR |= 0x0400;                                   // unmask EXTI10
199.  EXTI->FTSR |= 0x0400;                                  // select falling edge trigger
200.
201.  NVIC_SetPriority(EXTI15_10_IRQn, 1);                   // set priority interrupt for push button
202.  NVIC_EnableIRQ(EXTI15_10_IRQn);                        // enable interrupt in NVIC
203.
204.    GPIOA->MODER = 0;                                    // clear pin mode
205.    GPIOA->MODER = 0x55555555;                           // set pin output mode
206.    GPIOA->BSRR  = 0x00C00000;                           // turn off EN and R/W
207.    GPIOB->MODER = 0;                                    // clear pin mode
208.    GPIOB->MODER = 0x55555555;                           // set pin output mode
209.
210.  __enable_irq();                                        // global enable IRQs
211.  }
```

```
212.
213.  // Initialize LCD controller
214.  void LCD_init(void) {
215.     delayMs(30);                                          // initialization sequence
216.     LCD_command(0x30);
217.     delayMs(10);
218.     LCD_command(0x30);
219.     delayMs(1);
220.     LCD_command(0x30);
221.
222.     LCD_command(0x38);                                    // set 8-bit data, 2-line, 5x7 font
223.     LCD_command(0x06);                                    // move cursor right after each char
224.     LCD_command(0x01);                                    // clear screen, move cursor to home
225.     LCD_command(0x0F);                                    // turn on display, cursor blinking
226.  }
227.
228.  // Send command to LCD
229.  void LCD_command(unsigned char command) {
230.     GPIOA->BSRR = (RS | RW) << 16;                        // RS = 0, R/W = 0
231.     GPIOB->ODR = command;                                 // put command on data bus
232.     GPIOA->BSRR = EN;                                     // pulse E high
233.     delayMs(0);
234.     GPIOA->BSRR = EN << 16;                               // clear E
235.     if (command < 4)
236.        delayMs(2);                                        // command 1 and 2 needs up to 1.64ms
237.     else
238.        delayMs(1);
239.  }
240.
241.  // Write data to the LCD
242.  void LCD_data(char datawrite) {
243.     GPIOA->BSRR = RS;                  // RS = 1
244.     GPIOA->BSRR = RW << 16;            // R/W = 0
245.     GPIOB->ODR = datawrite;            // put data on data bus
246.     GPIOA->BSRR = EN;                  // pulse E high
247.     delayMs(0);
248.     GPIOA->BSRR = EN << 16;            // clear E
249.     delayMs(1);
250.  }
251.
252.  // Get key from keypad
253.  char keypad_getkey(void) {
254.     int row, col;
255.     const int row_mode[] = {0x00000100, 0x00000400, 0x00001000, 0x00004000}; // one row is output
256.     const int row_low[] =  {0x00100000, 0x00200000, 0x00400000, 0x00800000}; // one row is low
257.     const int row_high[] = {0x00000010, 0x00000020, 0x00000040, 0x00000080}; // one row is high
258.
259.     // check to see any key pressed
260.     GPIOC->MODER = 0x00005500;              // make all row pins output
261.     GPIOC->BSRR =  0x00F00000;              // drive all row pins low
262.     delayMs(1);                             // wait for signals to settle
263.     col = GPIOC->IDR & 0x000F;              // read all column pins
264.     GPIOC->MODER &= ~0x0000FF00;            // disable all row pins drive
265.     if (col == 0x000F)                      // if all columns are high
266.        return 0;                            // no key pressed
267.
```

```
// If a key is pressed, it gets here to find out which key. It activates one row at a time and
// read the input to see which column is active.
    for (row = 0; row < 4; row++) {
        GPIOC->MODER &= ~0x0000FF00;              // disable all row pins drive
        GPIOC->MODER |= row_mode[row];             // enable one row at a time
        GPIOC->BSRR = row_low[row];               // drive the active row low
        delayMs(1);                                // wait for signal to settle
        col = GPIOC->IDR & 0x000F;                // read all columns
        GPIOC->BSRR = row_high[row];              // drive the active row high
        if (col != 0x000F) break;                  // if one of the input is low, some key is pressed.
    }
    GPIOC->BSRR = 0x000000F0;                     // drive all rows high before disable them
    GPIOC->MODER &= ~0x0000FF00;                  // disable all rows
    if (row == 4)
        return 0;                                 // if we get here, no key is pressed

    // gets here when one of the rows has key pressed, check which column it is
    if (col == 0x000E) return row * 4 + 1;         // key in column 0
    if (col == 0x000D) return row * 4 + 2;          // key in column 1
    if (col == 0x000B) return row * 4 + 3;         // key in column 2
    if (col == 0x0007) return row * 4 + 4;         // key in column 3

    return 0;   // just to be safe
}

// Mode Question
void Mode_Question(void){
unsigned char mode_question[60] = "Select mode: 1.Urban 2.Rural 3.Blinking Yellow 4.Maintanence";

for(int k=0; k<28; k++){
LCD_data(mode_question[k]);                                  // Prints Mode Question
}
LCD_command(0xC0);
for(int k=29; k<60; k++){
LCD_data(mode_question[k]);
}
}

// scroll and update cursor
int scroll_and_update_cursor(int scroll_cursor) {
delayMs(400);
LCD_command(0x18);
scroll_cursor += 1;
if(scroll_cursor == 16) {
delayMs(3000);
LCD_command(0x02);
scroll_cursor = 0;
}
return scroll_cursor;
}

// Delay timer
void delayMs(int n) {
    int i;

    // Configure SysTick
```

```
324.    SysTick->LOAD = 16000;  // reload with number of clocks per millisecond
325.    SysTick->VAL = 0;       // clear current value register
326.    SysTick->CTRL = 0x5;    // Enable the timer
327.
328.    for(i = 0; i < n; i++) {
329.      while((SysTick->CTRL & 0x10000) == 0) // wait until the COUNTFLAG is set
330.        { }
331.    }
332.    SysTick->CTRL = 0;      // Stop the timer (Enable = 0)
333.  }
```

Link to video