New Paltz
STATE UNIVERSITY OF NEW YORK

EGC331-01 Microcontroller System Design

Professor Michael Otis

# Final Report

12/08/2020

Frank Seelmann

## Introduction

For this project, students are tasked with making a traffic light controller. The minimum requirements for the controller are that it can simulate the functionality of a rural traffic light that is capable of pedestrian traffic. The machine must also have some form of human-machine interface. This must be done on the NUCLEO-F446RE in Keil-based C or Assembly language, not an operating system type environment.

The traffic light controller described in this report has two modes of operation, rural and urban, and is able to switch between them via input from a hex keypad as part of the human-machine interface. This can be done without resetting the machine, and it also has pedestrian crossing capabilities. The output half of the human-machine interface is provided via an LCD which displays the current mode of operation and the state of the machine.

## Design

The traffic controller has two operating modes. The primary mode, mode 0, is the rural traffic light mode. In this mode the state of the machine changes based on input and time. A change occurs if there is a car waiting on the road which currently has a red light, or if a pedestrian requests to cross.

The secondary mode, mode 1, is the urban traffic light mode. In this mode the state of the machine changes based on time, or if a pedestrian requests to cross. In this mode the pedestrian crossing can only occur after the last state.

The status of the buttons, which are the input, are read after the light has been in a green state for 5 seconds. Keys 1 through 4 of the hex keypad are the inputs that represent whether a car is present and waiting at a road. Key 1 and 2 are for North and South, respectively, and Key 3 and 4 are for East and West, respectively. Key 13 and 14 are used to set the mode to mode 0 and mode 1, respectively. Key 16 is used to take a pedestrian crosswalk request.

The LCD has 16 pins on it. It displays letters and numbers by being sent ASCII codes for the corresponding character to D0-D7. The pins $V_{DD}$ and $V_{SS}$ provide +5V power and ground, while pin $V_0$ is used to control the contrast. The RS (register select) pin is used to alter the type of input received (i.e. a command like "clear" versus displaying the data). The R/W (read/write) pin determines if data is being written to the LCD controller or read from it. The E (enable) pin is used by the LCD to latch information presented to its data pins. The LED+ and LED- pins control the LED backlight.

The keypad has 8 pins on it, 4 for the rows and 4 for the columns. The program for the keypad first drives all rows low to see if a key was pressed. If no key is pressed, a 0 is returned. If one is pressed, the code will drive one row low at a time and read the columns. When the column is found, the code can return the correct key based on the (x,y) coordinate on the keypad.

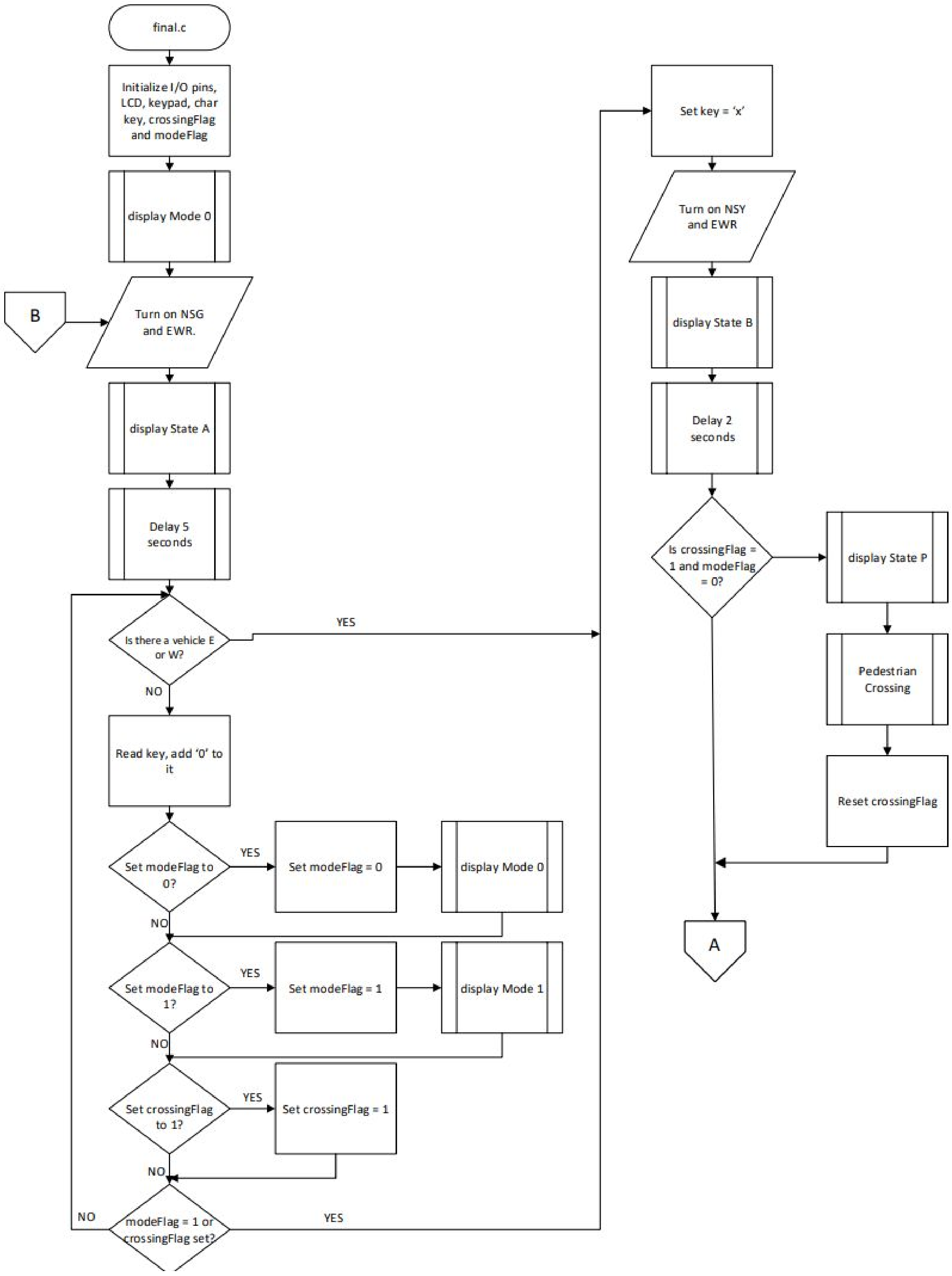


Figure 1 - Flowchart (1/3)

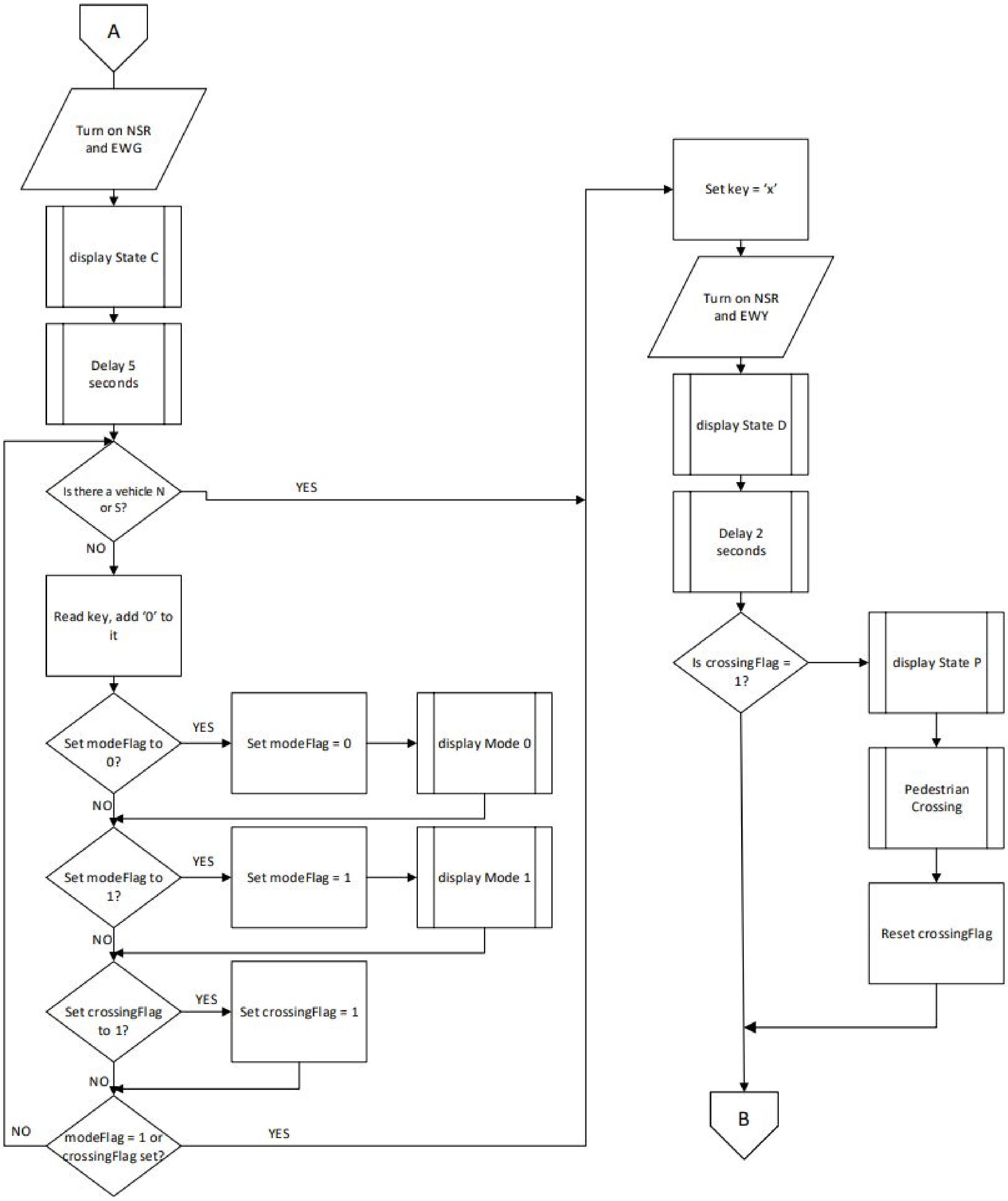


Figure 2 - Flowchart (2/3)

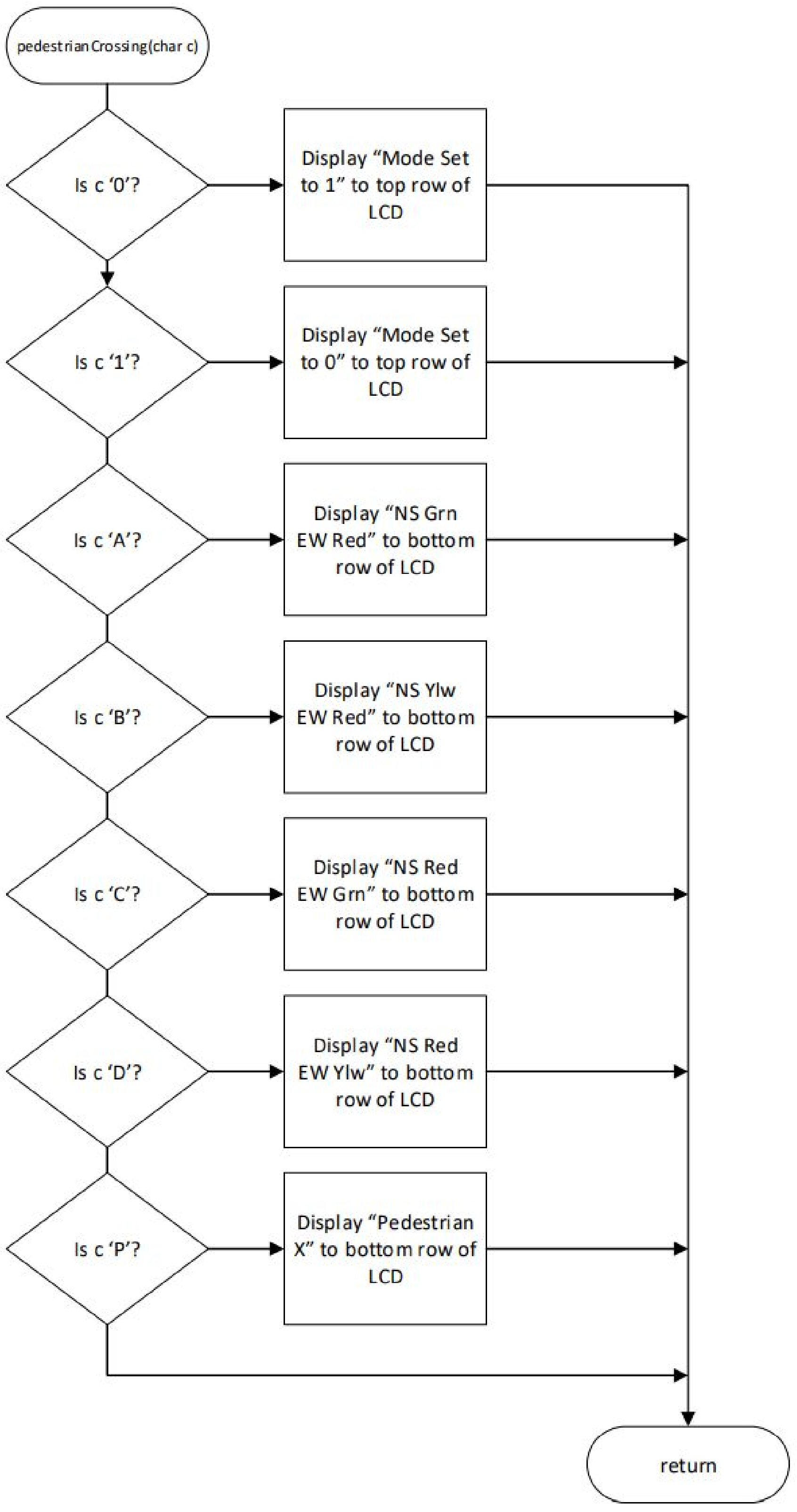

pedestrianCrossing(char c)
Is c '0'?
Display "Mode Set to 1" to top row of LCD
Is c '1'?
Display "Mode Set to 0" to top row of LCD
Is c 'A'?
Display "NS Grn EW Red" to bottom row of LCD
Is c 'B'?
Display "NS Ylw EW Red" to bottom row of LCD
Is c 'C'?
Display "NS Red EW Grn" to bottom row of LCD
Is c 'D'?
Display "NS Red EW Ylw" to bottom row of LCD
Is c 'P'?
Display "Pedestrian X" to bottom row of LCD
return

Figure 3 - Flowchart (3/3)

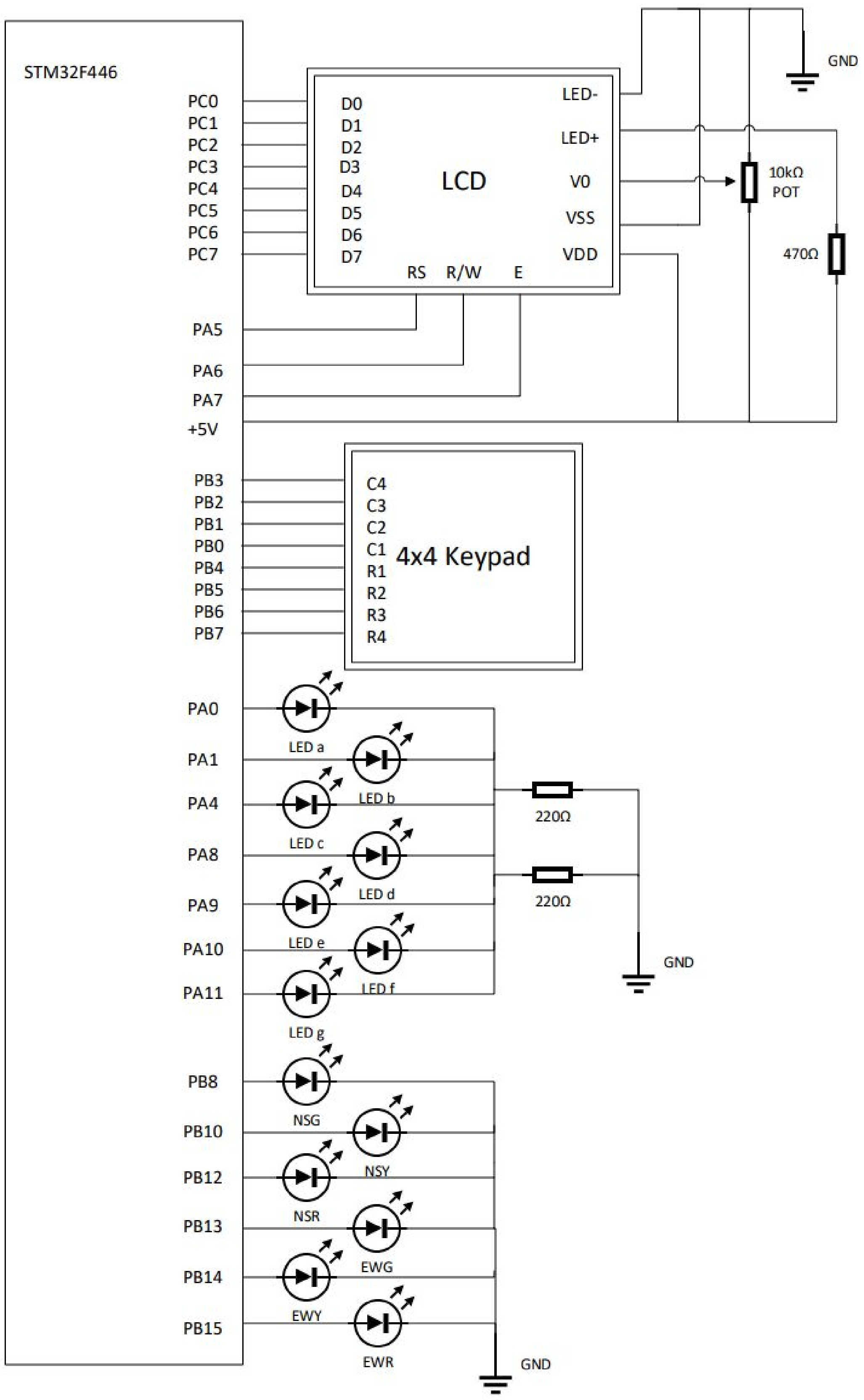


Figure 4 - Block Diagram

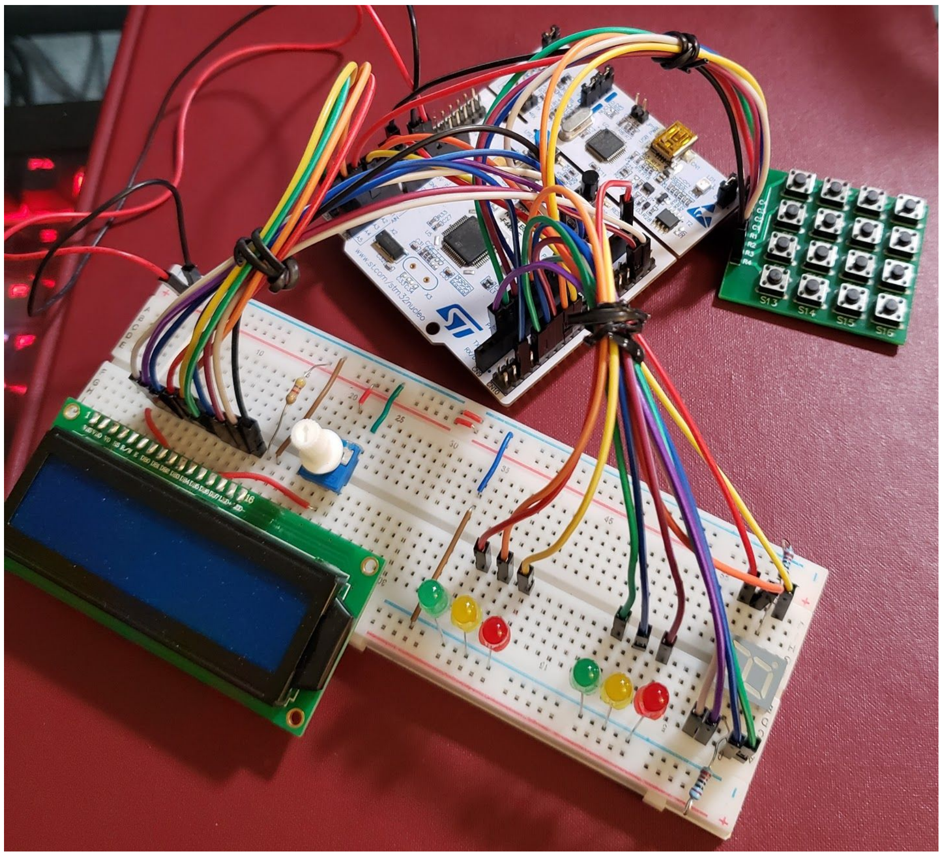


Figure 5 - Physical Setup

## Results & Conclusion

The machine works as anticipated, achieving all of the goals set. This project allows students to show off what they learned in all of the lab assignments for the course, including handling input and output on the NUCLEO board to effect the registers, making output codes for a 7-segment display, outputting to a Liquid Crystal Display, and taking input from a hexadecimal keypad. A video of the functional verification of the machine is linked below, as well as the github link for the project.

Video: youtu.be/3ORF2Jzng9E
Github: github.com/Frank-Seelmann/Microcontollers-Final-TrafficController

## Appendix

### C Program

```
/*
*	final.c
*
*	Frank Seelmann			EGC331-01 - Microcontrollers System Design
*	Prof Michael Otis		12/8/20
*
*	The traffic controller has two operating modes. In the primary mode,
* mode 0, the state of the machine only changes on input and time. This
*	change occurs if there is a car waiting on the road which currently
*	has a red light, or if a pedestrian requests to cross.
*
*	In the secondary mode, mode 1, the state of the machine changes purely
* based on time, or if a pedestrian requests to cross. In this mode pedestrian
* crossing can only occur after the last state.
*
*	The status of the buttons, which are the input, are read after the light
*	has been in a green state for 5 seconds.
*/

#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 keypad_init(void);
char keypad_getkey(void);
void LCD_command(unsigned char command);
void LCD_data(char data);
void LCD_init(void);
void PORTS_init(void);
void pedestrianCrossing(void);
void display(char);

int main(void) {
```

```
unsigned char key = '0';
keypad_init();
LCD_init();
GPIOA->MODER = 0x55555555;
GPIOB->MODER = 0x55555555;    // set pins to output mode
int crossingFlag = 0;                    // flag for pedestrian crosswalk request
int modeFlag = 0;                        // flag for the mode of operation
display('0');                            // display the mode on the LCD
while(1) {

    GPIOB->ODR = 0x8100;  // State A - NSG,EWR output to LEDs
    display('A');                                   // display the state on the LCD
    delayMs(5000);

    while(key != '3' && key != '4')  {
            key = keypad_getkey() + '0';  //read the keypad
                    if (key == '=') {
                            modeFlag = 0;
                            display('0');                   // display the mode on the LCD
                    }
                    else if (key == '>') {
                            modeFlag = 1;
                            display('1');                   // display the mode on the LCD
                    }
                    if (key == '@') {
                            crossingFlag = 1;
                    }

                    if (modeFlag == 1 | key == '@')
                            break;
            }

            key = 'x';                          // reset key
            GPIOB->ODR = 0x8400;  // State B - NSY,EWR to LEDs
            display('B');                   // display the state on the LCD
            delayMs(2000);

            if (crossingFlag && modeFlag == 0) {
                    display('P');           // display the state on the LCD
                    pedestrianCrossing();   // method for down-counting the 7-segment display
```

```
                crossingFlag = 0;
            }

            GPIOB->ODR = 0x3000;  // State C - NSR,EWG to LEDs
            display('C');                                   // display the state on the LCD
            delayMs(5000);

            while(key != '1' && key != '2') {
                key = keypad_getkey() + '0';
                if (key == '=') {
                    modeFlag = 0;
                    display('0');                           // display the mode on the LCD
                }
                else if (key == '>') {
                    modeFlag = 1;
                    display('1');                           // display the mode on the LCD
                }
                if (key == '@')
                    crossingFlag = 1;

                if (modeFlag == 1 | key == '@')
                    break;
            }

            key = 'x';                          // rest key
             GPIOB->ODR = 0x5000;   // State D - NSR,EWY ON to LEDs
            display('D');                       // display the state on the LCD
             delayMs(2000);

            if (crossingFlag) {
                display('P');               //display the state on the LCD
                pedestrianCrossing();  //method for down-counting the 7-segment display
                crossingFlag = 0;
            }
    }
}

/*
* This function outputs a message to the LCD depending
* the parameter it is passed
```

```
*/
void display(char c) {
        if (c == '0') {
            LCD_command(0x80);   // this command puts the cursor the top row
            LCD_data('M');
            LCD_data('o');
            LCD_data('d');
            LCD_data('e');
            LCD_data(' ');
            LCD_data('S');
            LCD_data('e');
            LCD_data('t');
            LCD_data(' ');
            LCD_data('T');
            LCD_data('o');
            LCD_data(' ');
            LCD_data('0');
            LCD_data(' ');
            LCD_data(' ');
            LCD_data(' ');
        }
        else if (c == '1') {
            LCD_command(0x80);
            LCD_data('M');
            LCD_data('o');
            LCD_data('d');
            LCD_data('e');
            LCD_data(' ');
            LCD_data('S');
            LCD_data('e');
            LCD_data('t');
            LCD_data(' ');
            LCD_data('T');
            LCD_data('o');
            LCD_data(' ');
            LCD_data('1');
            LCD_data(' ');
            LCD_data(' ');
            LCD_data(' ');
        }
```

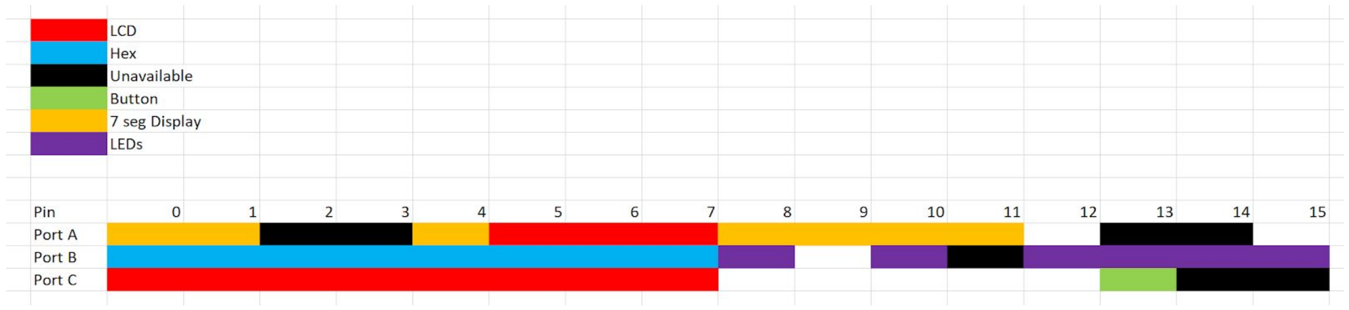
```
else if (c == 'A') {
        LCD_command(0xC0);// this command puts the cursor to the bottom row
        LCD_data('N');
        LCD_data('S');
        LCD_data(' ');
        LCD_data('G');
        LCD_data('r');
        LCD_data('n');
        LCD_data(' ');
        LCD_data(' ');
        LCD_data(' ');
        LCD_data(' ');
        LCD_data('E');
        LCD_data('W');
        LCD_data(' ');
        LCD_data('R');
        LCD_data('e');
        LCD_data('d');
}
else if (c == 'B') {
        LCD_command(0xC0);
        LCD_data('N');
        LCD_data('S');
        LCD_data(' ');
        LCD_data('Y');
        LCD_data('l');
        LCD_data('w');
        LCD_data(' ');
        LCD_data(' ');
        LCD_data(' ');
        LCD_data(' ');
        LCD_data('E');
        LCD_data('W');
        LCD_data(' ');
        LCD_data('R');
        LCD_data('e');
        LCD_data('d');
}
else if (c == 'C') {
        LCD_command(0xC0);
        LCD_data('N');
        LCD_data('S');
        LCD_data(' ');
        LCD_data('R');
        LCD_data('e');
        LCD_data('d');
        LCD_data(' ');
        LCD_data(' ');
        LCD_data(' ');
        LCD_data(' ');
        LCD_data('E');
        LCD_data('W');
        LCD_data(' ');
        LCD_data('G');
        LCD_data('r');
        LCD_data('n');
}
else if (c == 'D') {
        LCD_command(0xC0);
        LCD_data('N');
        LCD_data('S');
        LCD_data(' ');
        LCD_data('R');
        LCD_data('e');
        LCD_data('d');
        LCD_data(' ');
        LCD_data(' ');
        LCD_data(' ');
        LCD_data(' ');
        LCD_data('E');
        LCD_data('W');
        LCD_data(' ');
        LCD_data('Y');
        LCD_data('l');
        LCD_data('w');
}
else if (c == 'P') {
        LCD_command(0xC0);
        LCD_data('P');
        LCD_data('e');
```

```
                              LCD_data('d');
                              LCD_data('e');
                              LCD_data('s');
                              LCD_data('t');
                              LCD_data('r');
                              LCD_data('i');
                              LCD_data('a');
                              LCD_data('n');
                              LCD_data(' ');
                              LCD_data('X');
                              LCD_data(' ');
                              LCD_data(' ');
                              LCD_data(' ');
                              LCD_data(' ');
                        }
}


void pedestrianCrossing(void) {
                        GPIOB->ODR = 0x9000;
                        int arr[12] = {0xC13, 0xF13, 0x013, 0xF11, 0xD11, 0xC12, 0x913, 0xB03,
0x012, 0x713, 0x713, 0x713};

    for(int i = 0; i < 12; i++) { /* loop until last number in the list */
        GPIOA->ODR = arr[i];
        delayMs(750);                                   /* display the mumber */
                                    GPIOA->ODR = 0x0000;
                                    delayMs(250);                                   /* display
nothing */
    }
}

/* this function initializes PC0-3 (column) and PC4-7 (row).
 * The column pins need to have the pull-up resistors enabled.
 */
void keypad_init(void) {
    RCC->AHB1ENR |=  0x17;              /* enable GPIOC clock */
    GPIOB->MODER &= ~0x0000FFFF;    /* clear pin mode to input */
    GPIOB->PUPDR =   0x00000055;    /* enable pull up resistors for column pins */
}
```

```
char keypad_getkey(void) {
   int row, col;
   const int row_mode[] = {0x00000100, 0x00000400, 0x00001000, 0x00004000}; /* one row is
output */
   const int row_low[] =  {0x00100000, 0x00200000, 0x00400000, 0x00800000}; /* one row is
low */
   const int row_high[] = {0x00000010, 0x00000020, 0x00000040, 0x00000080}; /* one row is
high */

   /* check to see any key pressed */
   GPIOB->MODER = 0x55555500;      /* make all row pins output */
   GPIOB->BSRR =  0x00F00000;      /* drive all row pins low */
   delayMs(10);                    /* wait for signals to settle */
   col = GPIOB->IDR & 0x000F;      /* read all column pins */
   GPIOB->MODER &= ~0x0000FF00;    /* disable all row pins drive */
   if (col == 0x000F)              /* if all columns are high */
      return 0;                    /* no key pressed */

   /* 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++) {
      GPIOB->MODER &= ~0x0000FF00;    /* disable all row pins drive */
      GPIOB->MODER |= row_mode[row];  /* enable one row at a time */
      GPIOB->BSRR = row_low[row];     /* drive the active row low */
      delayMs(10);                    /* wait for signal to settle */
      col = GPIOB->IDR & 0x000F;      /* read all columns */
      GPIOB->BSRR = row_high[row];    /* drive the active row high */
      if (col != 0x000F) break;       /* if one of the input is low, some key is pressed. */
   }
   GPIOB->BSRR = 0x000000F0;           /* drive all rows high before disable them */
   GPIOB->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 */
}

/* initialize port pins then initialize LCD controller */
void LCD_init(void) {
    PORTS_init();

    delayMs(30);            /* initialization sequence */
    LCD_command(0x30);
    delayMs(10);
    LCD_command(0x30);
    delayMs(1);
    LCD_command(0x30);

    LCD_command(0x38);      /* set 8-bit data, 2-line, 5x7 font */
    LCD_command(0x06);      /* move cursor right after each char */
    LCD_command(0x01);      /* clear screen, move cursor to home */
    LCD_command(0x0F);      /* turn on display, cursor blinking */
}

void PORTS_init(void) {
    RCC->AHB1ENR |=  0x07;          /* enable GPIOB/C clock */

    /* PB5 for LCD R/S */
    /* PB6 for LCD R/W */
    /* PB7 for LCD EN */
    GPIOA->MODER &= ~0x0000FC00;    /* clear pin mode */
    GPIOA->MODER |=  0x00005400;    /* set pin output mode */
    GPIOA->BSRR = 0x00C00000;       /* turn off EN and R/W */

    /* PC0-PC7 for LCD D0-D7, respectively. */
    GPIOC->MODER &= ~0x0000FFFF;    /* clear pin mode */
    GPIOC->MODER |=  0x00005555;    /* set pin output mode */
}

void LCD_command(unsigned char command) {
    GPIOA->BSRR = (RS | RW) << 16;  /* RS = 0, R/W = 0 */
    GPIOC->ODR = command;           /* put command on data bus */
    GPIOA->BSRR = EN;               /* pulse E high */
    delayMs(0);
    GPIOA->BSRR = EN << 16;         /* clear E */

    if (command < 4)
        delayMs(2);         /* command 1 and 2 needs up to 1.64ms */
    else
        delayMs(1);         /* all others 40 us */
}

void LCD_data(char data) {
    GPIOA->BSRR = RS;               /* RS = 1 */
    GPIOA->BSRR = RW << 16;         /* R/W = 0 */
    GPIOC->ODR = data;              /* put data on data bus */
    GPIOA->BSRR = EN;               /* pulse E high */
    delayMs(0);
    GPIOA->BSRR = EN << 16;         /* clear E */

    delayMs(1);
}

/* 16 MHz SYSCLK */
void delayMs(int n) {
    int i;

    /* Configure SysTick */
    SysTick->LOAD = 16000;  /* reload with number of clocks per millisecond */
    SysTick->VAL = 0;       /* clear current value register */
    SysTick->CTRL = 0x5;    /* Enable the timer */

    for(i = 0; i < n; i++) {
        while((SysTick->CTRL & 0x10000) == 0) /* wait until the COUNTFLAG is set */
            { }
    }
    SysTick->CTRL = 0;      /* Stop the timer (Enable = 0) */
}
```

## <u>Pin Assignments</u>



| | |
|---|---|
| 🟥 | LCD |
| 🟦 | Hex |
| ⬛ | Unavailable |
| 🟩 | Button |
| 🟨 | 7 seg Display |
| 🟪 | LEDs |

| Pin | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Port A | 🟨 | 🟨 | ⬛ | ⬛ | 🟨 | 🟥 | 🟥 | 🟥 | 🟨 | 🟨 | 🟨 | 🟨 | | ⬛ | ⬛ | |
| Port B | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 | 🟪 | | 🟪 | ⬛ | 🟪 | 🟪 | 🟪 | 🟪 |
| Port C | 🟥 | 🟥 | 🟥 | 🟥 | 🟥 | 🟥 | 🟥 | 🟥 | | | | | | 🟩 | ⬛ | ⬛ |