New Paltz
STATE UNIVERSITY OF NEW YORK

Final Project

Universal Traffic Intersection Controller

EGC332– Microcontrollers

April 25, 2021

Christopher Lepore

Abstract

In this project we designed a Universal Traffic Intersection Controller that can operate 7 modes including the following. The urban mode which cycles through the lights with no added features. The rural mode which has the added feature of car sensors. The urban with crosswalk mode functions like the urban mode but has the added feature of crosswalks. The rural with crosswalk mode functions like the rural mode but has the added feature of crosswalks. The flashing yellow EW and red NS mode(YEW_RNS) does as stated and the flashing yellow NS and red ES(YNS_REW) mode does the same in reverse. Last, the flashing NS and EW red(Red_Red) mode features blinking red lights on both lanes. Along with these traffic light modes, there is an LCD menu that shows the current mode and can be used to change the mode, change the time delay, and can be used to change between 2 traffic modes at a certain time of day. In order to use the LCD menu, the controller's kill switch must be on which stops the traffic light cycle and sets it to two solid red lights. This prevents any accidents while changing the controller's settings. These features allow for a versatile and safe traffic light system to protect people on any intersection.

Table of contents

Introduction

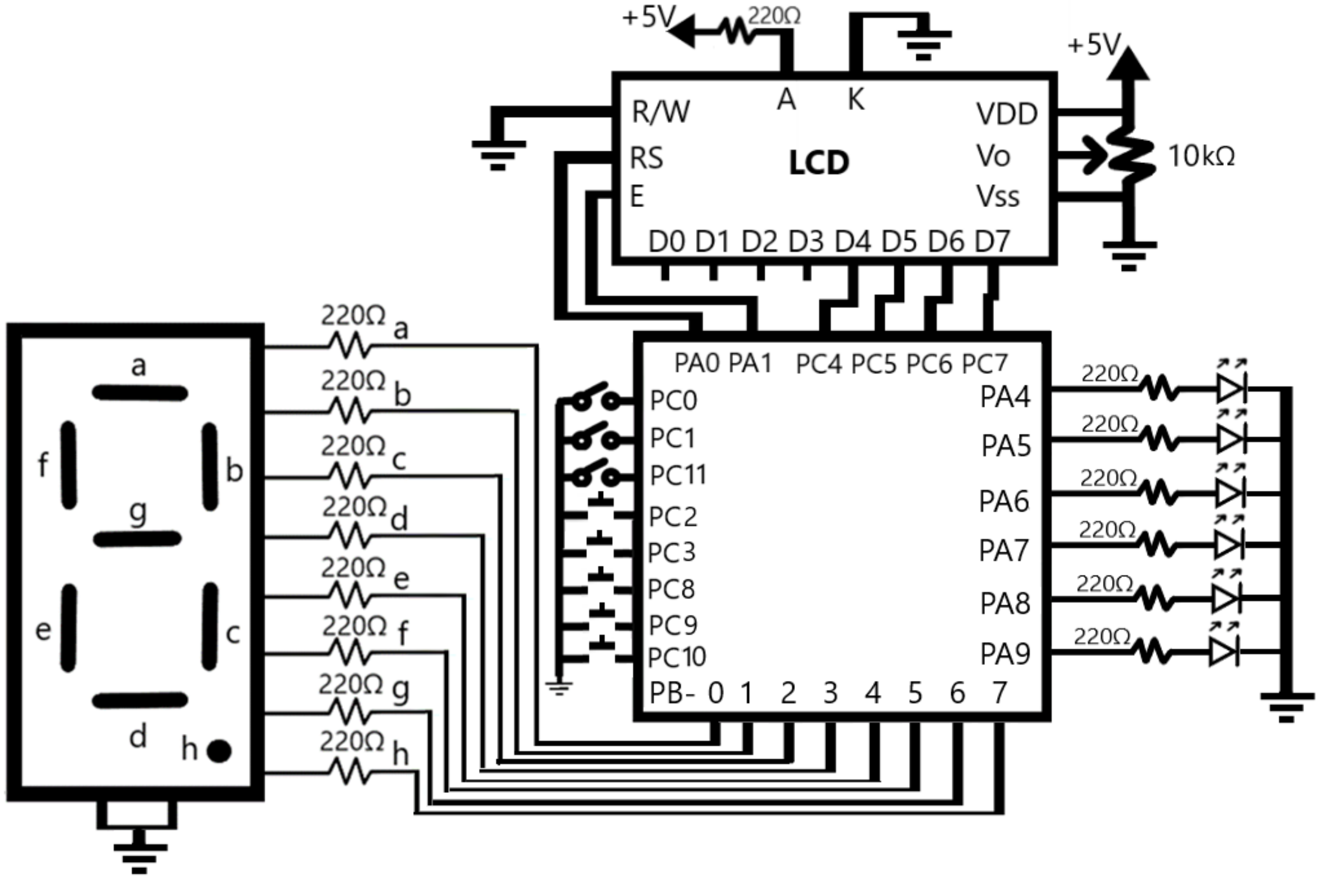
In this project we designed a Universal Traffic Intersection Controller that can operate 7 modes including urban, rural, urban with crosswalk, rural with crosswalk, flashing yellow EW and red NS(YEW_RNS), flashing yellow NS and red ES(YNS_REW), and flashing NS and EW red(Red_Red) mode using a Nucleo-64 microcontroller. This project also includes an LCD interface menu to set the mode, time delay of LEDs, and allow a mode change at a set time-of-day; it also includes a 7-segment display to count down when the crosswalk is active. This project introduced coding with interrupts in C programing. This also introduced sorting and wiring a high number of desired input and output ports of the STM32F446RE circuit board to allow for the use of many devices. The Universal Traffic Intersection Controller determines the mode and time delay of the traffic lights and can shift between them as needed through the LCD menu. By designing this traffic light controller, a deeper comprehension of port sorting and interrupts with the STM32F446RE circuit board was achieved.

Theory

The first electric traffic light was developed and implemented in 1912 by a policeman called Lester Wire and placed on the corner of Euclid Avenue and East 105th Street in Cleveland, Ohio, on August 5, 1914. The attempt to control traffic though was first demonstrated during the Roman Empire. However, the first breakthrough of the electric traffic light dated back to a device installed in London on year 1868 by J.P Knight, a railway signal engineer. This design was powered by gas and featured 2 semaphore arms that were used to signal stop when horizontal and signal caution when at a 45-degree angle. Unfortunately, however, the device unexpectedly exploded as a result of a gas leak resulting in the death of a police officer, and the use of the device was therefore discontinued. When electric traffic lights were first implemented in 1910, they only featured the normal urban traffic light and did not include any turn signals, car sensors, crosswalks, or ability to change the type of traffic mode or delay. These features didn't come until the 1950s and the 1960s, like the car sensors that were used in the form of a pressure plate that was placed at intersections that allowed computers to know that a vehicle was waiting at the red light. This was the time where computers were implemented and caused traffic light efficiency to increase significantly.

Design

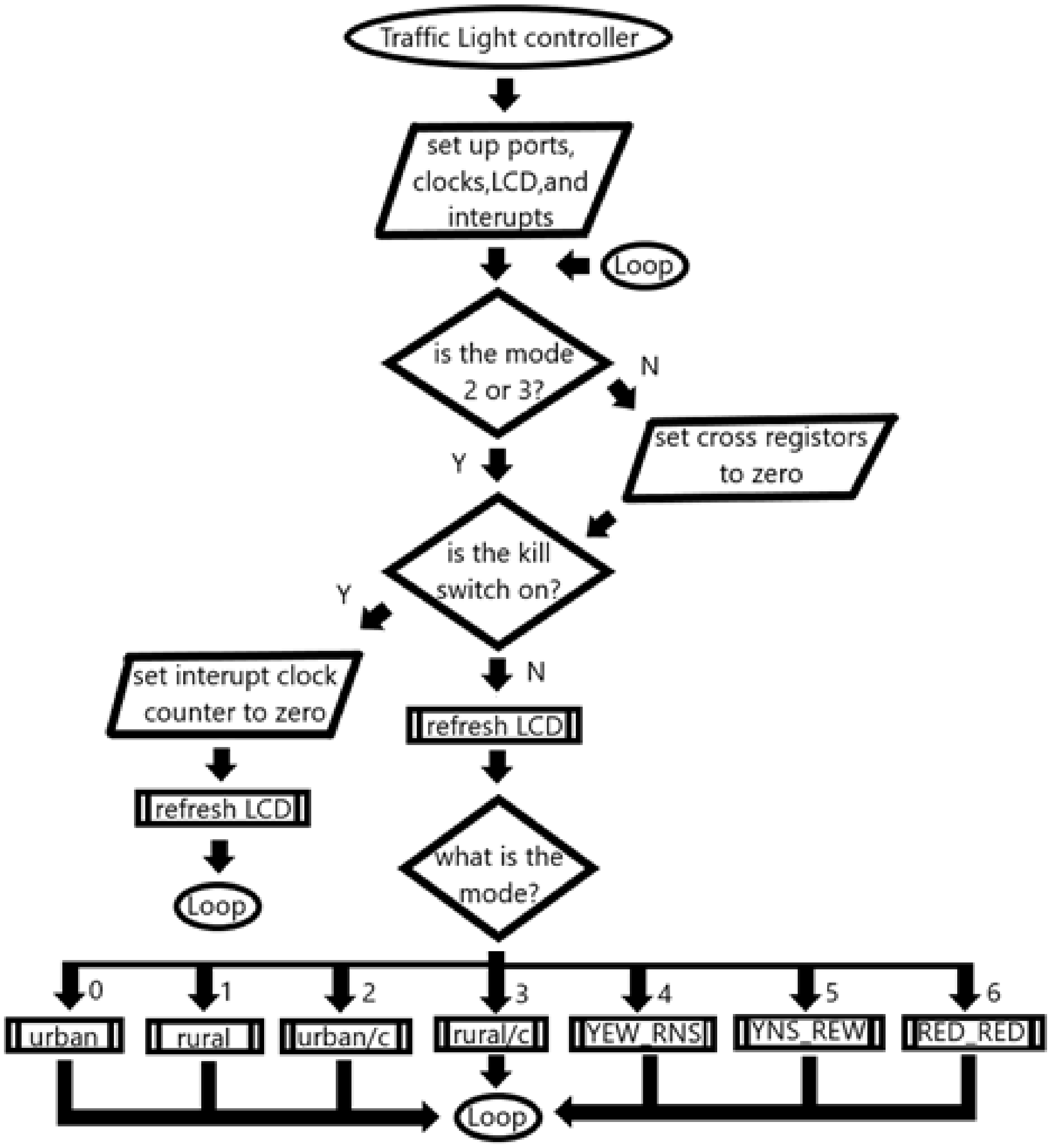
Hardware (Block Diagrams)



For the traffic light controller circuit, port PA4 to PA9 were set to output mode and then each connected to a light emitting diode (LED) to represent the traffic lights. There are 220-ohm resistors in between them to prevent the LEDs from burning out. Port PB0 to PB7 were set to output mode and then each connected to the seven-segment display from a to h accordingly. The common of the display are connected to ground. There are 220-ohm resistors in between them to prevent the LEDs of the display from burning out. Port PC4 to PC7 were set to output mode and then each connected to the LCD data pins from D4 to D7 accordingly. The cathode and anode of the display are connected to ground and 5 V accordingly. Port PA0 is set to register select(RS), PA1 to the enable(E), and the read/write(R/W) pin was grounded. Port PC0-PC3 and PC8-PC11 were set to input mode with pull up resistors. PC 0, 1, and 11 were given switches and others had buttons.

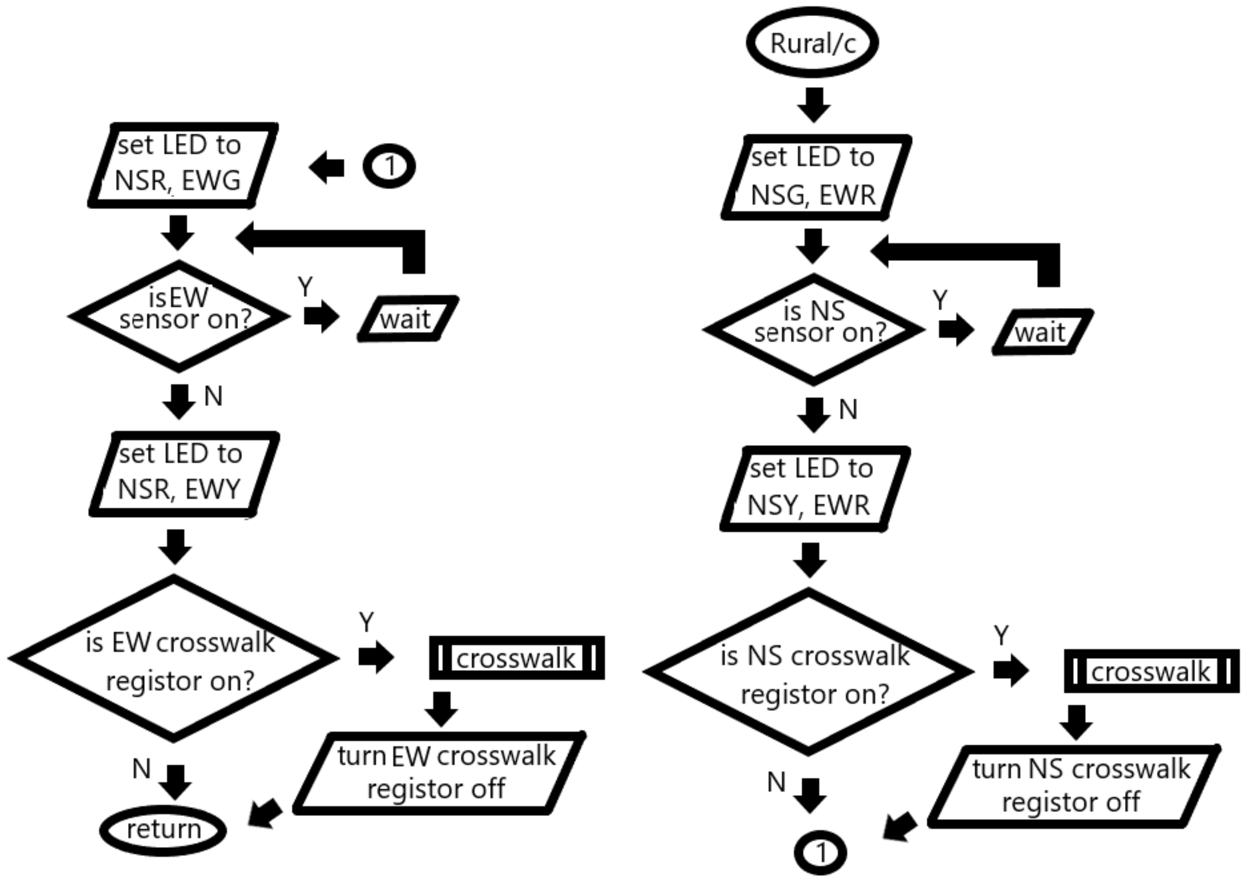
Software (Flow Charts)

Traffic light controller program



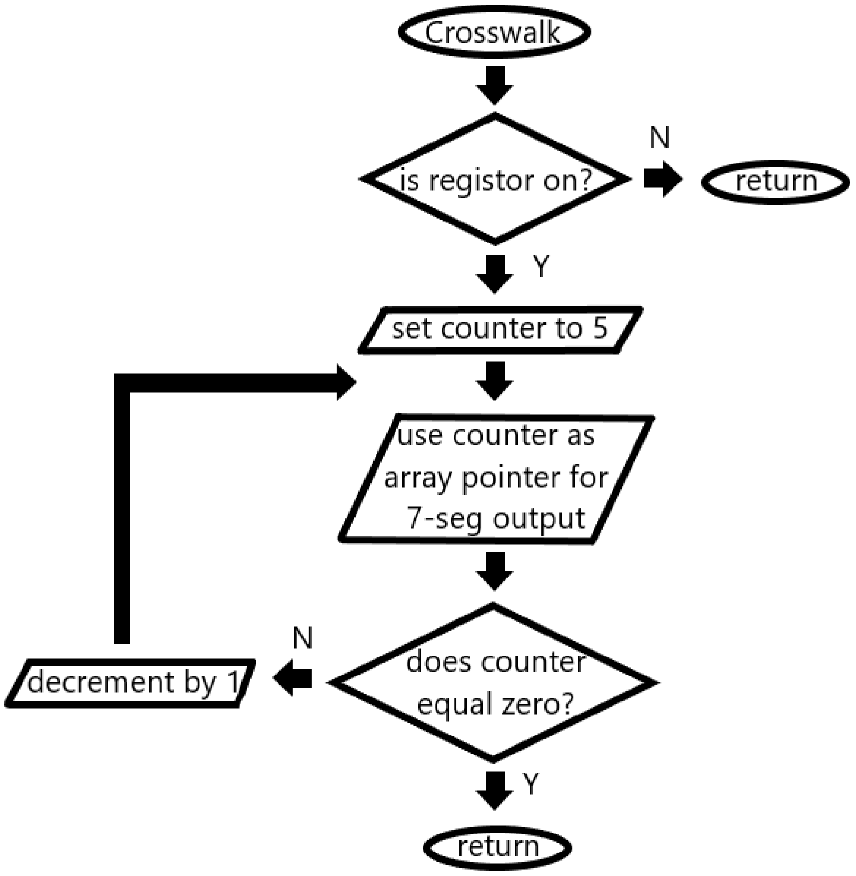
In the traffic light controller program the code will first set up the ports, clocks, LCD and interrupt ports. Then it will check if the mode is 2 or 3, if it is not then the crosswalk register will be set to zero to prevent unwanted activation when switching traffic modes. After, it will check if the kill switch is on (PC11 switch), if it’s on then instead of cycling through the current traffic mode, the lights will show solid NS and EW red LEDs after the traffic light finishes its current traffic cycle. If the kill switch is on the controller will also set the interrupt clock counter to 0 to prevent time interrupts when the traffic light is inactive otherwise the controller will output the current traffic mode. At the end of the loop the LCD will refresh and cycle to the top of the program.

Rural with crosswalk traffic light program



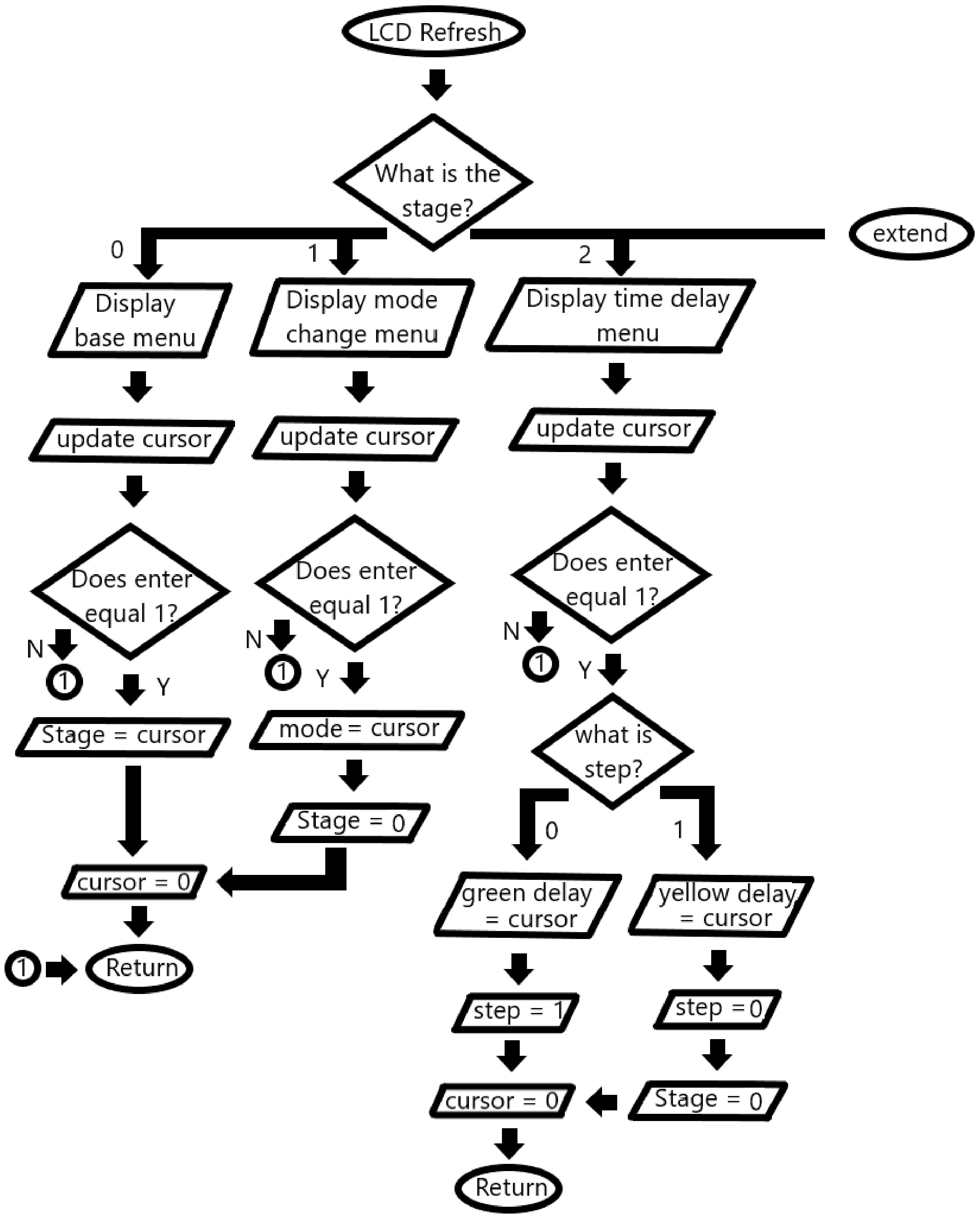
In the Rural with crosswalk program the code will cycle once just like a normal traffic light, but the green LEDs delay and the yellow LEDs delay are determined by its correlating variable in the LCD refresh function. When the NS car sensor(PC0 switch) is logic 1 the NSG and EWR LEDs will stay on till the sensor switches to logic 0. The same goes for the EW car sensor(PC1 switch) and the NSR and EWG LEDs. Just before the NS lights turn red the code will check if the NS crosswalk register is on(PC2 button), if it is the crosswalk function will run and after the register will be set to 0. The same goes for the EW lights with EW crosswalk register controlled by the PC3 button. The urban, rural, and urban with crosswalk modes are similar but exclude parts of this program. The urban mode doesn’t include the car sensors or crosswalks, the rural mode only includes the car sensors, and the urban with crosswalk mode only includes the crosswalks.

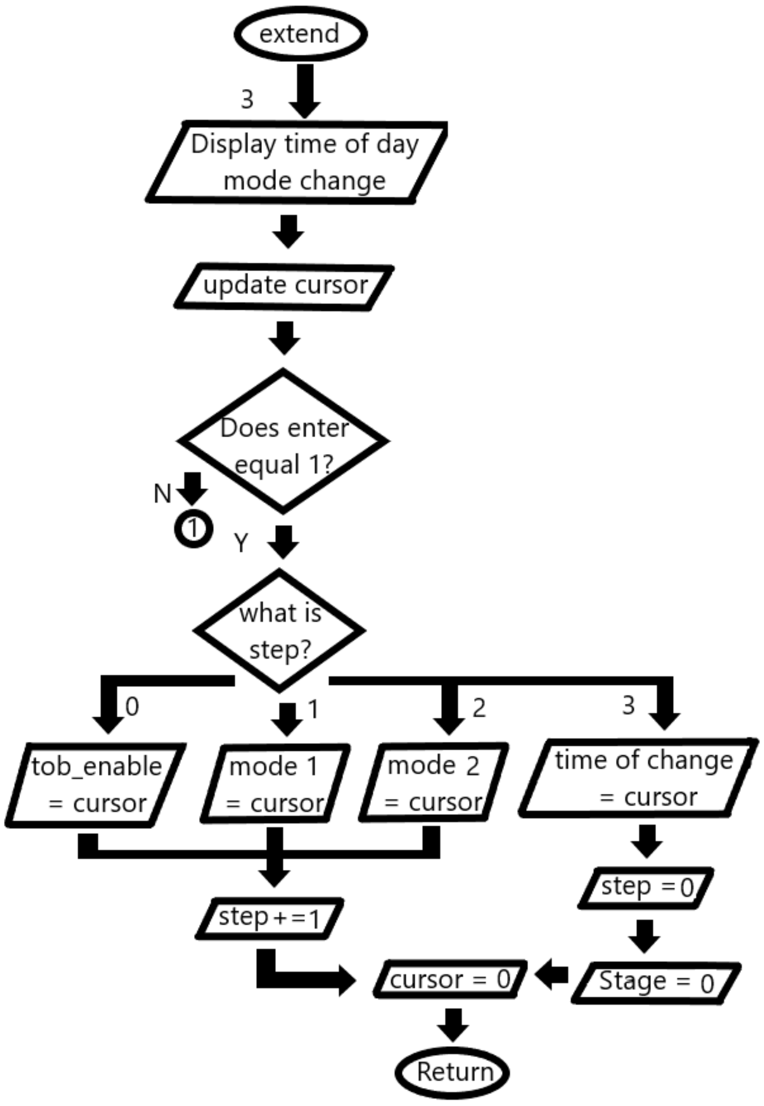
Crosswalk program



In the cross walk program the code will first check if the crosswalk register parameter is on, if not the program will do nothing. Otherwise, the counter will be set to 5 and will cycle through using the counter as an array pointer which will output the preset array value to the 7-segment display and then decreased by 1. This cycle will flash the red LEDs of the traffic light each time the counter decrements until the counter equals 0 and the program returns.

LCD Refresh program

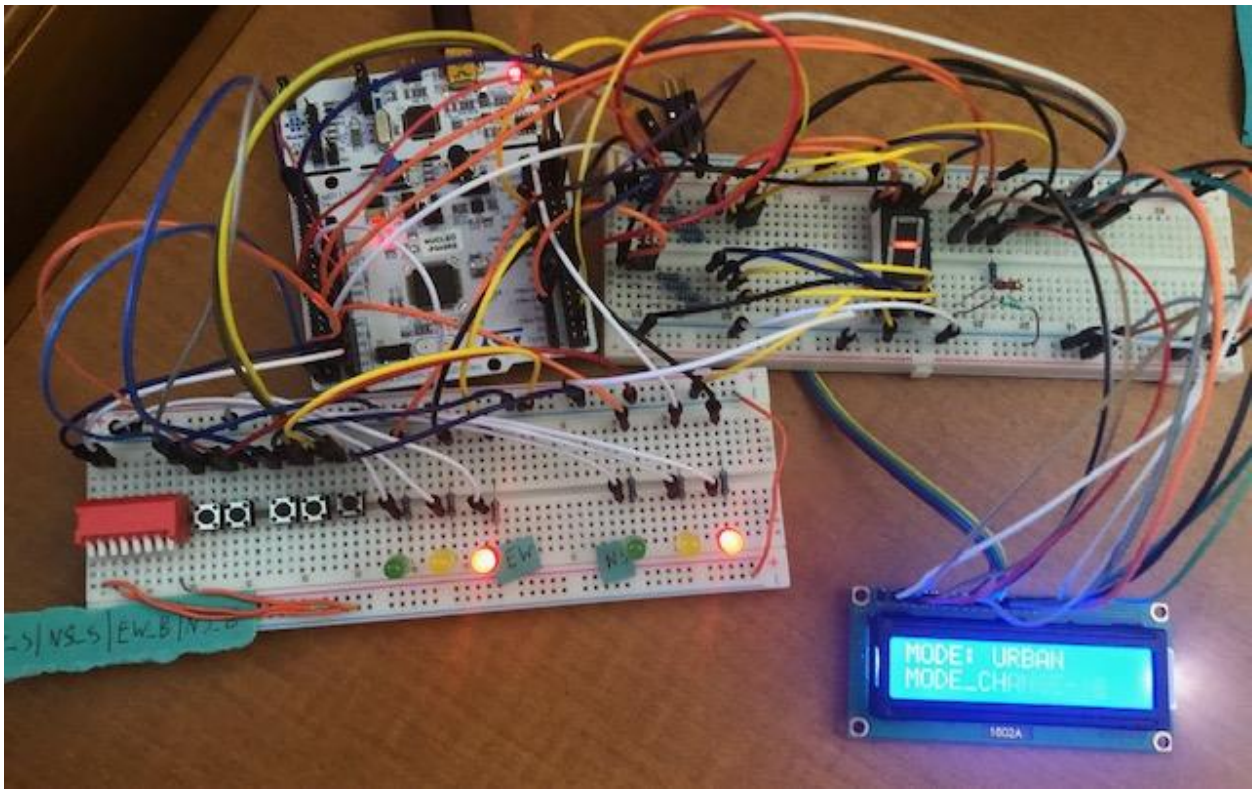




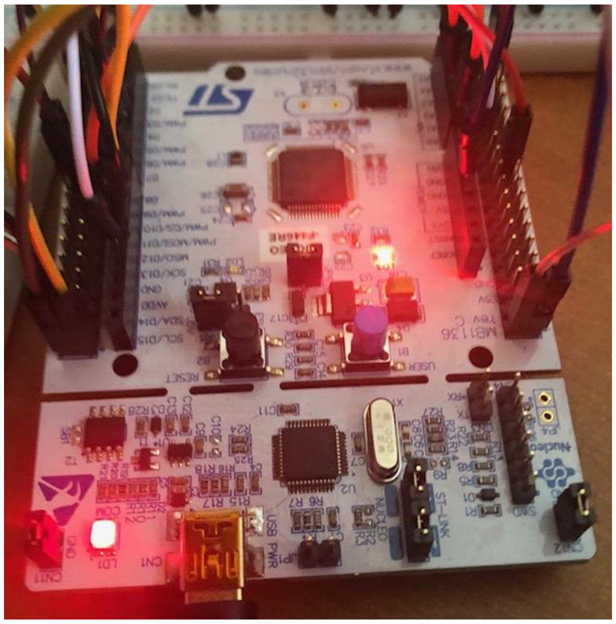
In the LCD refresh program the code will first display the menu based on the stage and step value. For stage 0 the base menu will be displayed, stage 1 displays the mode change menu, stage 2 displays the time delay menu, and stage 3 displays the time-of-day mode change menu. The base menu displays the current mode and is used to select stage 1, 2, or 3. The time delay and time-of-day mode change menu will display differently based on the current step value in order to select the next needed value with the cursor. Then the cursor will update based on the left(PC10) and right(PC9) buttons. If the enter button(PC8) is pressed the cursor value will be assigned to the needed variable and then set to 0. If the menu has multiple steps, then after the enter button is pressed the step will increment. The stage and step will go back to 0 after all the needed values are selected. The LCD prints characters form a 2d array where the first character is used to give the size of the word for a for loop to go through and print each character.

Verification

Nucleo and breadboard circuit



Breadboard wiring



NUCLEO circuit board

Summary and conclusion

In the Universal Traffic Intersection Controller, there are 7 modes that can be chosen including urban, rural, urban with crosswalk, rural with crosswalk, flashing yellow EW and red NS(YEW_RNS), flashing yellow NS and red ES(YNS_REW), and flashing NS and EW red(Red_Red) mode. The urban mode is a normal traffic light cycle, the rural mode includes car sensors that when triggered keeps the correlating lane's green LED on and the opposing lane's red LED on, this is controlled by the switches PC0(NS sensor) and PC1(EW sensor). There are two more urban and rural modes, but they include a crosswalk; the crosswalk is controlled by the buttons PC2(NS crosswalk register) and PC3(EW crosswalk register) and when pressed the main program is interrupted to set the buttons correlating register to 1 before returning to the main program. If the register is 1 for either the NS or EW lanes when its light is about to turn red then the crosswalk will count down from 5 and display the count on the 7-segment display while the red LEDs flash, after the register will be set to 0. If the mode does not include the crosswalk function, then the registers will be set to 0. Before the controller starts the traffic light program it will check if the kill switch(PC11) is on, if it is then the controller will display solid NS and EW red LEDs instead of doing the traffic light cycle for the selected mode. This allows the LCD to refresh more often so the user can interact with the menu.

The LCD menu allows the user to do operations such as change the mode, time delay, and enable and set a mode change at a time-of-day. The menu uses 3 variables to keep track of the menu including the stage which controls the operation, the step which if needed controls which variable of the operation is being changed, and the cursor which allows the user to scroll with the left(PC10) and right(PC9) buttons and when used with the enter button(PC8) it can change the selected variable of the menu or traffic light. For example, if the stage is 2 and step is 1 then the cursor with select a value for the yellow time delay. If the mode change at a time-of-day is enabled, then the program will alternate between two selected modes at a selected time. This time mode change is controlled by an SYSCLK interrupt and will trigger at the time selected by the user to switch mode 1 to mode 2 or vice versa. In the event the interrupt triggers in the middle of a time delay then it will wait till the delay finishes. This is because while the delay is active the SYSCLK interrupt is not enabled, and the interrupt counter continues until the delay finishes and the interrupt is reenabled.

This project introduced coding with interrupts in C programing along with sorting and wiring a high number of desired input or output ports of the

STM32F446RE circuit board to allow for use of many devices. This was done by analyzing the schematic to learn how to make the most use of the ports on the STM32F446RE circuit board and by reading the textbook to learn to use interrupts with the board. Due to the lessons above this project has prepared students to begin designing, coding, and wiring more complex projects with interrupts and the skills used.

Work cited

[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.

[2] M. A. Mazidi, S. Naimi, S. Naimi, and S. Chen, ARM assembly language programming & architecture. Middletown, DE?: www.MicroDigitalEd.com, 2016. 2nd Edition

Appendix (Programs)

```
#include "stm32f4xx.h"

#define RS 0x01     /* PA0 mask for LCD reg select */
#define EN 0x02     /* PA1 mask for LCD enable */

void delayMs(int n);
void urbanTraffic(void);
void ruralTraffic(void);
void urbanTraffic_c(void);
void ruralTraffic_c(void);
void redRed(void);
void yNS_rEW(void);
void yEW_rNS(void);
void crosswalk(int button);

void LCD_HUD(void);
int cursor_update(int max, int c);
void LCD_print(char data[]);
void LCD_nibble_write(char data, unsigned char control);
void LCD_command(unsigned char command);
void LCD_data(char data);
void LCD_init(void);
void PORTS_init(void);
void EXTI2_IRQHandler(void);
void EXTI3_IRQHandler(void);
void TIM2_IRQHandler(void);

int static mode = 0;                    //controls mode of traffic light
int static cursor = 0;                  //controls LCD menu
int static stage = 0;                   // controls menu operation
int static step = 0;                    //controls menu operation step
int static G = 5;                       //green light delay
int static Y = 2;                       //yellow light delay
int static Ns_cross = 0;                //NS crosswalk regestor
int static Ew_cross = 0;                //EW crosswalk regestor
int static tod_enable = 0;              //change at time of day enable
int static mode_1;                      //change at time of day mode 1
int static mode_2;                      //change at time of day mode 2

int main(void) {
   LCD_init();
   GPIOB->ODR = 0x40; //dash

   while(1)
            {
                  if(mode != 2 && mode != 3)             //rest crosswalk register if not in use
                  {Ns_cross = 0; Ew_cross = 0; GPIOB->ODR = 0x40;}
                  if((GPIOC -> IDR >> 11)      & 0x1) //kill switch
                  {
                            switch(mode)
                            {
                                      case 0: LCD_HUD();
                                              urbanTraffic();
                                              break;
                                      case 1: LCD_HUD();
                                              ruralTraffic();
                                              break;
                                      case 2: LCD_HUD();
                                              urbanTraffic_c();
                                              break;
                                      case 3: LCD_HUD();
                                              ruralTraffic_c();
                                              break;
                                      case 4: LCD_HUD();
                                              yEW_rNS();
                                              break;
                                      case 5: LCD_HUD();
                                              yNS_rEW();
```

```
                                break;
                        case 6: LCD_HUD();
                                redRed();
                                break;
                        default: urbanTraffic();
                                break;
                }
            }
            else {LCD_HUD(); GPIOA->ODR = 0x00000240; TIM2->CNT = 0;}
            //sets interupt counter to 0 and LEDs to solid red

            delayMs(500);
            LCD_command(1);
        }
}

void LCD_HUD(void)
{
        char static modec[6] = {"6MODE:"};
        char static p_back[3] = {"3<-"};
   char static p_front[3] = {"3->"};
   char static mc[13] = {"=MODE_CHANGE:"};
   char static td[12] = {"<TIME_DELAY:"};

   char static gy[4][3] = {"2G","2Y","3G:","3Y:"};
   char static operations[3][12] = {"<MODE_CHANGE",";TIME_DELAY","5TOD:"};
   char static modes[7][10] = {"6URBAN","6RURAL","8URBAN/c","8RURAL/c",
                               ":Y_EW/R_NS",":Y_NS/R_EW",":BLINK_RED"};
   char static tod[3][6] = {"4M1:","4M2:","6TIME:"};
   char static tod_en[2][8] = {"7ENABLE","8DISABLE"};
   char static tod_count[8][3] = {"25","310","315","320","325","330","335","340"};

   switch(stage)
   {
        case 0: LCD_print(modec);    //base menu
            LCD_data(' ');
            LCD_print(modes[mode]);
            LCD_command(192); //next line
            if((~(GPIOC -> IDR >> 9)     & 0x1) || (~(GPIOC -> IDR >> 10)    & 0x1))
            {cursor = cursor_update(2, cursor);} //call cursor update
            if(cursor != 0) LCD_print(p_back);
            LCD_print(operations[cursor]);
            if(cursor == 2) LCD_print(tod_en[((~tod_enable)&0x1)]);
            if(cursor != 2) LCD_print(p_front);
            if(~(GPIOC -> IDR >> 8)      & 0x1)
            {stage = cursor+1; cursor = 0;  delayMs(1000);}      //stage 0 menu controller
            break;

case 1: LCD_print(mc);          //mode change menu
            LCD_command(192); //next line
            if((~(GPIOC -> IDR >> 9)     & 0x1) || (~(GPIOC -> IDR >> 10)    & 0x1))
            {cursor = cursor_update(6, cursor);} //call cursor update
            if(cursor != 0) LCD_print(p_back);
            LCD_print(modes[cursor]);
            if(cursor != 6) LCD_print(p_front);
            if(~(GPIOC -> IDR >> 8)      & 0x1)
            {mode = cursor; cursor = 0; stage = 0; delayMs(1000);}  //stage 1 menu controller
            break;

case 2: LCD_print(td); LCD_print(gy[0]); LCD_data(G|0x30); LCD_data(',');
            LCD_print(gy[1]); LCD_data(Y|0x30);                    //time delay change
            LCD_command(192); //next line
            if(step == 0) LCD_print(gy[2]); else LCD_print(gy[3]);
            LCD_data(' ');
            if((~(GPIOC -> IDR >> 9)     & 0x1) || (~(GPIOC -> IDR >> 10)    & 0x1))
            {cursor = cursor_update(9, cursor); }//call cursor update
            if(cursor != 0) LCD_print(p_back);
            LCD_data(cursor + 0x30); LCD_data(' '); LCD_data('S');
            if(cursor != 9) LCD_print(p_front);
            if(step == 0 && (~(GPIOC -> IDR >> 8)      & 0x1))
            {G = cursor; cursor = 0; step = 1; delayMs(1000);}  //stage 2 menu controller
            else if (step == 1 && (~(GPIOC -> IDR >> 8)         & 0x1))
            {Y = cursor; cursor = 0; step = 0; stage = 0; delayMs(1000);}
            break;

case 3: LCD_print(operations[2]);                          //time of day mode change
            LCD_command(192); //next line
            if(step == 1) LCD_print(tod[0]);
            if(step == 2) LCD_print(tod[1]);
            switch(step)
            {
               case 0: if((~(GPIOC -> IDR >> 9) & 0x1) || (~(GPIOC -> IDR >> 10)& 0x1))
                           cursor = cursor_update(1, cursor); //select mode 1 , call cursor update
                           if(cursor != 0) LCD_print(p_back);
                           LCD_print(tod_en[cursor]);
                           if(cursor != 1) LCD_print(p_front);
                           break;
               case 1:
               case 2: if((~(GPIOC -> IDR >> 9) & 0x1) || (~(GPIOC -> IDR >> 10)& 0x1))
                           cursor = cursor_update(6, cursor); //select mode 2 , call cursor update
                           if(cursor != 0) LCD_print(p_back);
                           LCD_print(modes[cursor]);
```

```
                        if(cursor != 6) LCD_print(p_front);
                        break;
                    case 3: LCD_print(tod[2]);
                        if((~(GPIOC -> IDR >> 9) & 0x1) || (~(GPIOC -> IDR >> 10)& 0x1))
                        cursor = cursor_update(7, cursor); //select time , call cursor update
                        if(cursor != 0) LCD_print(p_back);
                        LCD_print(tod_count[cursor]); LCD_data(' '); LCD_data('S');
                        if(cursor != 7) LCD_print(p_front);
                        break;
                } // stage 3 menu controller
                if(step == 0 && (~(GPIOC -> IDR >> 8)      & 0x1) && cursor == 0)
                {cursor = 0; step = 1; tod_enable = 1; delayMs(1000);} //ENABLE time interup
                else if(step == 0 && (~(GPIOC -> IDR >> 8)          & 0x1) && cursor == 1)
                {cursor = 0; tod_enable = 0; stage = 0; delayMs(1000);} //DISABLE time interup
                else if(step == 1 && (~(GPIOC -> IDR >> 8)          & 0x1))
                {mode_1 = cursor; mode = cursor; cursor = 0; step = 2; delayMs(1000);}
                else if(step == 2 && (~(GPIOC -> IDR >> 8)          & 0x1))
                {mode_2 = cursor; cursor = 0; step = 3; delayMs(1000);}
                else if(step == 3 && (~(GPIOC -> IDR >> 8) & 0x1)) {TIM2->ARR =
                (((5*cursor)+5)*1000)-1; cursor = 0; step = 0; stage = 0; delayMs(1000);}
                break;
        }
}

void urbanTraffic(void)
{
        GPIOA->ODR =  0x00000210;  /* turn on NSG, EWR */
        delayMs(G*1000);
        GPIOA->ODR = 0x00000220;  /* turn on NSY, EWR */
        delayMs(Y*1000);
        GPIOA->ODR = 0x000000C0;  /* turn on NSR, EWG */
        delayMs(G*1000);
        GPIOA->ODR = 0x00000140;  /* turn on NSR, EWY */
        delayMs(Y*1000);
}
void ruralTraffic(void)
{
        GPIOA->ODR =  0x00000210;  /* turn on NSG, EWR */
        delayMs(G*1000);

        while(((GPIOC -> IDR)  & 0x1)){}  //NS sensor

        GPIOA->ODR = 0x00000220;  /* turn on NSY, EWR */
        delayMs(Y*1000);
        GPIOA->ODR = 0x000000C0;  /* turn on NSR, EWG */
        delayMs(G*1000);
        while(((GPIOC -> IDR >>     1)        & 0x1)){}  // EW sensor

        GPIOA->ODR = 0x00000140;  /* turn on NSR, EWY */
        delayMs(Y*1000);
}
void urbanTraffic_c(void)
{
        GPIOA->ODR =  0x00000210;  /* turn on NSG, EWR */
        delayMs(G*1000);
        GPIOA->ODR = 0x00000220;  /* turn on NSY, EWR */
        delayMs(Y*1000);

        crosswalk(Ns_cross);  Ns_cross = 0;
        if(Ns_cross == 0 && Ew_cross == 0)
        GPIOB->ODR = 0x40; else GPIOB->ODR = 0xC0;

        GPIOA->ODR = 0x000000C0;  /* turn on NSR, EWG */
        delayMs(G*1000);
        GPIOA->ODR = 0x00000140;  /* turn on NSR, EWY */
        delayMs(Y*1000);

        crosswalk(Ew_cross); Ew_cross = 0;
        if(Ns_cross == 0 && Ew_cross == 0) GPIOB->ODR = 0x40;
        else GPIOB->ODR = 0xC0;
}
void ruralTraffic_c(void)
{
        GPIOA->ODR =  0x00000210;  /* turn on NSG, EWR */
        delayMs(G*1000);

        while(((GPIOC -> IDR)  & 0x1)){}  //NS sensor

        GPIOA->ODR = 0x00000220;  /* turn on NSY, EWR */
        delayMs(Y*1000);

        crosswalk(Ns_cross);  Ns_cross = 0;
        if(Ns_cross == 0 && Ew_cross == 0) GPIOB->ODR = 0x40;
        else GPIOB->ODR = 0xC0;

        GPIOA->ODR = 0x000000C0;  /* turn on NSR, EWG */
        delayMs(G*1000);

        while(((GPIOC -> IDR >>     1)        & 0x1)){}  // EW sensor

        GPIOA->ODR = 0x00000140;  /* turn on NSR, EWY */
        delayMs(Y*1000);

        crosswalk(Ew_cross); Ew_cross = 0;
        if(Ns_cross == 0 && Ew_cross == 0) GPIOB->ODR = 0x40;
        else GPIOB->ODR = 0xC0;
}
void redRed(void)
{
        GPIOA->ODR = 0x00000240;        /* turn on NSR, EWR */
        delayMs(500);
        GPIOA->ODR = 0x00000000; /* turn off lights */
        delayMs(500);
}
void yNS_rEW(void)
{
        GPIOA->ODR = 0x00000220; /* turn on NSY, EWR */
        delayMs(500);
        GPIOA->ODR = 0x00000000; /* turn off lights */
        delayMs(500);
}
void yEW_rNS(void)
{
        GPIOA->ODR = 0x00000140;  /* turn on NSR, EWY */
        delayMs(500);
        GPIOA->ODR = 0x00000000; /* turn off lights */
        delayMs(500);
}
void crosswalk(int button)
{
        unsigned int lookup[16] =
{0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7C,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71};
// 0 - F array
        if(button == 1)
        {
                for(int i=5;i>=0;i--)
                {
                        GPIOB->ODR = lookup[i]; // output counter to 7seg
                        redRed();
                }
        }
}
/* initialize ports then initialize LCD controller */
void LCD_init(void)
{
   PORTS_init();
```

```
    delayMs(20);              /* LCD controller reset sequence */
    LCD_nibble_write(0x30, 0);
    delayMs(5);
    LCD_nibble_write(0x30, 0);
    delayMs(1);
    LCD_nibble_write(0x30, 0);
    delayMs(1);

    LCD_nibble_write(0x20, 0);  /* use 4-bit data mode */
    delayMs(1);
    LCD_command(0x28);        /* set 4-bit data, 2-line, 5x7 font */
    LCD_command(0x06);        /* move cursor right */
    LCD_command(0x01);        /* clear screen, move cursor to home */
    LCD_command(0x0F);        /* turn on display, cursor blinking */
}
void PORTS_init(void)
{
                __disable_irq();                              // global disable IRQs

    RCC->AHB1ENR =  7;           /* enable GPIOA, GPIOB and GPIOC clock */
    RCC->APB2ENR = 0x4000;    /* enable TIM2 clock*/

                GPIOC->PUPDR &= ~0x00FF00FF; /* clear pin mode */
                GPIOC->MODER &= ~0x0000FF00; /* clear pin mode */
                GPIOA->MODER &= ~0x000FFF0F; /* clear pin mode */
                GPIOB->MODER &= ~0x0000FFFF; /* clear pin mode */

                GPIOC->PUPDR =  0x00550055;     /* INPUT: PC0-PC3, PC8-PC11 set to input
                                                for mode sensors, cross buttons, menu
                                                controls, kill switch */
                GPIOC->MODER =  0x00005500;   /* set pin output mode for LCD PC4-PC7 for
                                                D4-D7*/
                GPIOA->MODER =  0x00055505;   /* LED: set pin A4 - A9 to output mode for
                                                lights, PA0-RS, PA1-En for LCD */
                GPIOB->MODER =  0x00005555;     /* 7-seg: set pin to output mode PB 0 - 7 */
                GPIOA->BSRR  =  0x00020000;     /* turn off EN */

                SYSCFG->EXTICR[0] &= ~0xFF00; // clear ports
                SYSCFG->EXTICR[0] |= 0x2200; //select PORTS_init C for EXTI 2 and 3
                EXTI->IMR |= 0xC;             //unmask EXTI 2 and 3
                EXTI->RTSR |= 0xC;            // set rising edge trigger
                NVIC_EnableIRQ(8);            //enable IRQ8 -- EXTI2
                NVIC_EnableIRQ(9);            //enable IRQ9 -- EXTI3

                RCC->APB1ENR |= 1;                    //enables interupt counter
                TIM2->PSC = 16000-1;                  //reduces microcontroller clock speed
```

```
			TIM2->ARR = (10*1000)-1;		// sets the count number for interupt
			TIM2->CR1 = 1;				//enable counter
			TIM2->DIER |= 1;			//enable UIE
			NVIC_EnableIRQ(TIM2_IRQn);	//enable interupt in NVIC

			__enable_irq();   // global disable IRQs
}

void TIM2_IRQHandler(void)
{
		TIM2->SR = 0;
		if(((GPIOC -> IDR >> 11)	& 0x1) && tod_enable == 1)
			// if kill switch is off and time of day change is enabled
		{
			if(mode == mode_1) mode = mode_2; //switch modes
			else mode = mode_1;
		}
		TIM2->CNT = 0;			//set counter to 0 in case needed
}

void EXTI2_IRQHandler(void)
{
		if(mode == 2 || mode == 3) //PC1 Ns_cross
		{
			Ns_cross = 1;
			GPIOB->ODR = 0xC0;				//turn on registor and 7 seg dp LED
		}
		EXTI->PR = 0x4;
}

void EXTI3_IRQHandler(void)
{
		if(mode == 2 || mode == 3) //PC2 Ew_cross
		{
			Ew_cross = 1;
			GPIOB->ODR = 0xC0;				//turn on registor and 7 seg dp LED
		}
		EXTI->PR = 0x8;
}

void LCD_nibble_write(char data, unsigned char control)
{
   /* populate data bits */
   GPIOC->BSRR = 0x00F00000;	/* clear data bits */
   GPIOC->BSRR = data & 0xF0;	/* set data bits */
```

```
    /* set R/S bit */
    if (control & RS)
        GPIOA->BSRR = RS;
    else
        GPIOA->BSRR = RS << 16;

    /* pulse E */
    GPIOA->BSRR = EN;
    delayMs(0);
    GPIOA->BSRR = EN << 16;
}

void LCD_command(unsigned char command)
{
    LCD_nibble_write(command & 0xF0, 0);    /* upper nibble first */
    LCD_nibble_write(command << 4, 0);      /* then lower nibble */

    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)
{
    LCD_nibble_write(data & 0xF0, RS);      /* upper nibble first */
    LCD_nibble_write(data << 4, RS);        /* then lower nibble */

    delayMs(1);
}

void LCD_print(char data[])
{
        int size = data[0] - 0x30;                  //use first character as size
        for(int i=1;i<size;i++)
        {
                if(data[i] != 0)
                {
                        LCD_data(data[i]);                  //print each character
                        delayMs(30);
                }else break;
        }
}

int cursor_update(int max, int c)
{
        if(c == 0 && (~(GPIOC -> IDR >> 10)       & 0x1))
        {
                return max;
        }
        else if(c != 0 && (~(GPIOC -> IDR >> 10)  & 0x1))
        {
                return c-1;
        }
        else if(c != max && (~(GPIOC -> IDR >> 9)          & 0x1))
        {
                return c+1;
        }
        else if(c == max && (~(GPIOC -> IDR >> 9)         & 0x1))
        {
                return 0;
        }
}
/* 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) */
}
```