State University of New York at New Paltz
Division of Engineering Programs
Department of Electrical and Computer Engineering

EGC332 Microcontroller System Design

Universal Traffic Light Controller

Yiwen Jia, Benjamin Jon, Andrew Sykut

May 2023

ABSTRACT

This experiment presents a versatile traffic control system with a mode selection feature. Upon initialization, the user can select between either the rural or urban traffic light sequence. If the urban traffic light sequence is chosen, it starts a continuous cycle of urban traffic lights until a pedestrian presses the crosswalk button. Once activated, the microcontroller will start a visual countdown with auditory cues on the digital seven-segment display, displaying the correct state allowing the pedestrian to be aware when it is safe to cross. Afterwards, the urban sequence resumes its cycle. If the user instead chooses the rural sequence, the system will transition smoothly from urban to rural. In the rural mode, the system cycles through the traffic light patterns only when traffic is detected in the opposite direction. The Universal Traffic Control Unit will continuously check to see if the corrected value is being displayed. A comprehensive review of this design from both low-level and high-level perspectives showcases a universally adaptable traffic light controller, adept at serving diverse environments.

CONTENTS

INTRODUCTION

This project consists of a traffic light system at an intersection that is designed and implemented through an ARM STM32F446 Nucleo microcontroller board. Based on the traffic flow density present, this system can be set to either the Urban or the Rural mode. In the Urban mode, the traffic is assumed to be heavy and thus the system allows an equal amount of time for the traffic in East-West and North-South directions to flow. In addition, a pedestrian crosswalk button is implemented so that once pressed, pedestrians in both directions can cross the street before the traffic switches directions. On the other hand, in the Rural mode the traffic is assumed to be light, thus the traffic lights of one direction remain unchanged until the system detects traffic from the other direction. To make the system more user friendly, an LCD is used to display the mode of the system and a message reminding the pedestrian to walk. A seven-segment display and an audio device are also implemented to visually and audibly inform the pedestrians the amount of time left to cross the street. Furthermore, a preliminary transition state is incorporated to indicate a shift in mode, informing both the designers and peers of a transition shift. In this state, all traffic lights are on, and the LCD displays asterisks.

The system is simulated through a Moore finite state machine (FSM) that is programmed in the C language using a linked list structure, where each node represents a state and contains pointers to the next state. The comprehensive design and implementation details are elaborated in this report.

THEORY

Traffic light controllers are crucial systems that govern the flow of traffic. They provide guidance for cars to follow, ensuring that traffic moves safely and efficiently. Traffic engineering often focuses on optimal placement of traffic lights, ensuring that the timing and sequence of light changes are as efficient as possible to minimize accidents, and coordinating multiple traffic signals to facilitate smooth traffic flow. The first electric traffic light system was invented by Lester Wire, and it was a two-color, red-and-green light that had to be manually switched from one light to another by a police officer. If the police officer requested a light transition, a buzzer would be played to warn pedestrians ahead of time.

However, there were many limitations. One limitation was that it required the constant presence of a traffic officer to function. It also didn't include a yellow light to provide a cautionary signal to the driver, which could lead to abrupt stops or confusion at intersections, increasing the risk of accidents. There was also a lack of vehicle detection, so in the case of a rural area, the lights would change continuously even if it was unnecessary [1].

Although Wire's primitive design was very limited, it offered a prototype that was automated and modernized to transform into the traffic light system of our time. A universal traffic light controller with mode select is an advanced traffic light controller that supports multiple modes, suitable for both urban and rural locations. The urban traffic light would continuously cycle between red, green, and yellow with a delay standard to meet the demands of a busy urban environment. Moreover, a pedestrian activated button for crosswalks prioritizes pedestrian safety as it explicitly signals when it is safe to cross. For accessibility reasons, a beep is played after the button is pressed and while the timer counts down. In rural locations, the traffic light sequence does not need to keep cycling, especially if one intersection tends to be more busy than the other. In this case, a sensor is placed in both intersections to determine if a car is present. If it is, it will alert the traffic light to be changed. Such advancements not only increase traffic and energy efficiency but also improve traffic flow and make it safer for both drivers and pedestrians crossing the street.

The universal traffic light system is simulated by a Moore FSM as shown in **Figure 1.** The common information contained in every state includes the state index, outputs of traffic lights (6 digits), messages displayed in two lines in the LCD, and the delay time. The states U5 and U6 contain the outputs to the seven-segment display (Countdown) instead of the delay time. These two states are the only states that stop the traffic in both directions and allow pedestrians to cross the street. The FSM contains three parts: the blue and green sections representing the sub-FSMs of the urban and rural models, respectively; and the beige section with input m values, representing the transition state between the two models. The blue and green sections are explained in the previous lab reports. The state S0 is the starting state and the transition state of the system. In this one-second state, all the traffic lights are on, and the first line on the

LCD displays six asterisks. The input m is the mode selector. The system is directed to the urban mode when m is set to 1. Otherwise, the system goes to the rural mode when m is set to 0. Once the system is in one mode, the change of the mode selector directs the system to go to S0, then transit to the other mode. And the mode shift happens only after the NSG/EWR or the NSR/EWG states. The implementation of the FSM is elaborated in the next section.

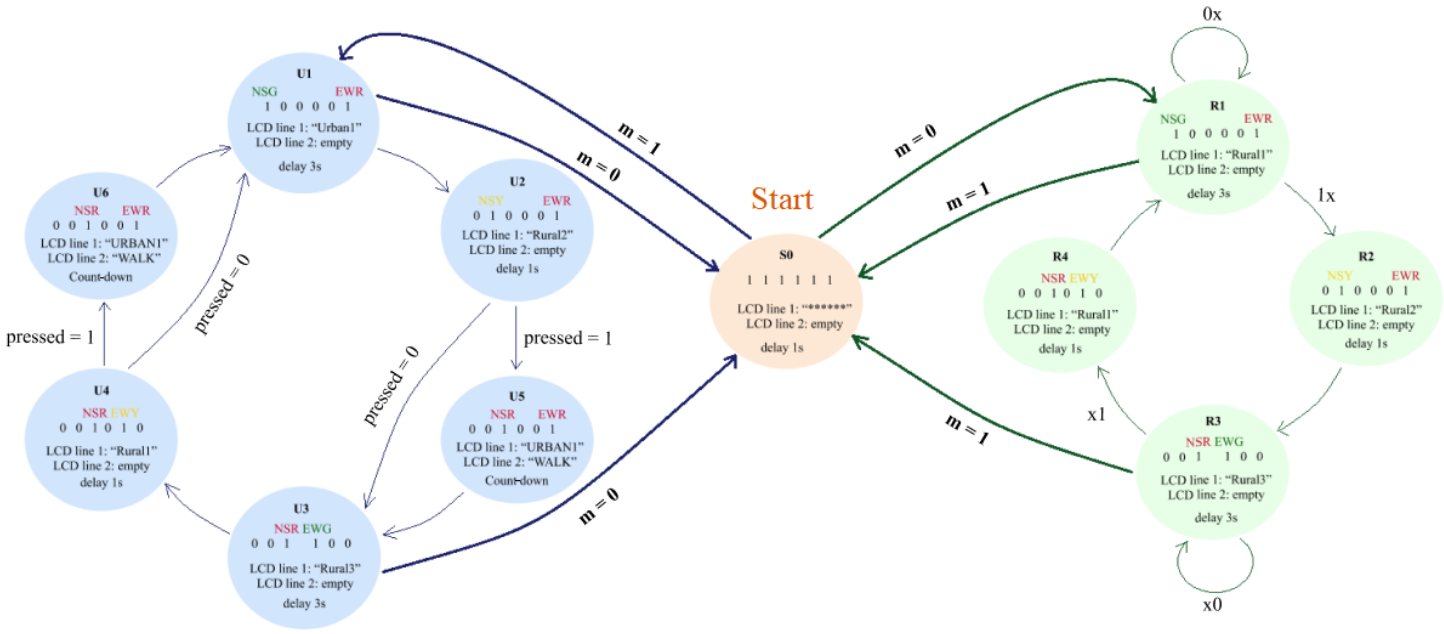


Figure 1. FSM of the universal traffic light system. The blue circles are the Urban mode while the green ones are the Rural mode. The beige circle represents the transition state S0. The input variable, pressed, represents the pedestrian crossing button. pressed = 1 corresponds to the presence of the pedestrians, while pressed = 0 means no pedestrian. And the 2-digit input in the Rural mode correspond to EW|NS traffic sensors, respectively. The value 1 represents traffic occurrance in the respective direction, while 0 represents no traffic in that direction.

DESIGN

Hardware:

The hardware of the Universal Traffic Light Controller is shown in **Figure 2**. The microcontroller's pins establish connections with various output devices such as the 7-segment display (pins A4-A10), traffic lights (pins B0, B2, B4, B6, B8, B9), the LCD (pins B5, B7, and C4-C7), and the speaker (pin A0). To collect the user's input are the interrupt button (pin C13), the traffic sensors (pins C0, C1), and the mode selector (pin C2).

Since the audio device beeps with the countdown process, its pin selection follows the same port as those connected to the 7-segment display. This port selection makes the software implementation easier by effectively enabling one section of the code to control two devices.

The 7-segment display and switches are soldered on a separate board for a better display and

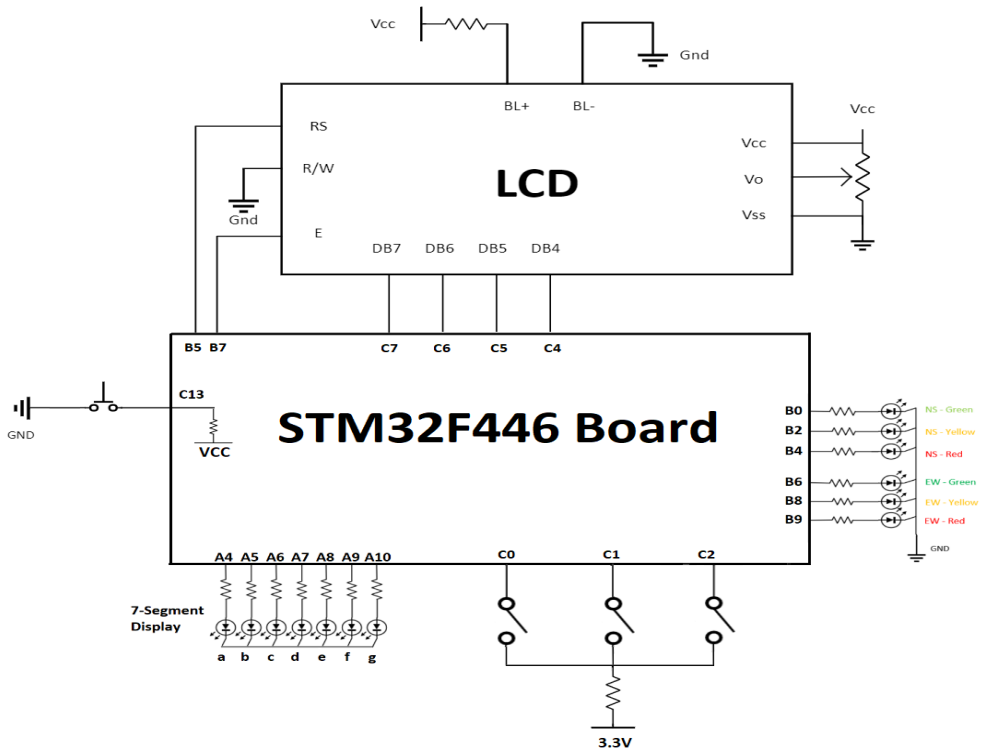


Figure 2: Hardware diagram representing the entire traffic light system

connection. The separate board serves as a convenient place to provide a user interface that is simple and easy to interact with.The input pins, the mode selector and the EW and NS sensors, are designed to be consecutive pins because the binary values represented by the inputs are used as the conditionals in the code.

Software:

The Universal Traffic Light Control Unit is a C program for controlling a traffic light system using an STM32F446RE microcontroller. It consists of a main function, ten user defined functions, and one structure. The UML in **Figure 3** displays the whole structure of the design of software. The main function is responsible for initializing variables, setting up the clock, enabling GPIO ports, and setting the initial state to S0 from the State structure. After the initialization, the program enters an infinite loop that runs the finite state machine, continuously updating what the next state will be based on the current state and inputs.

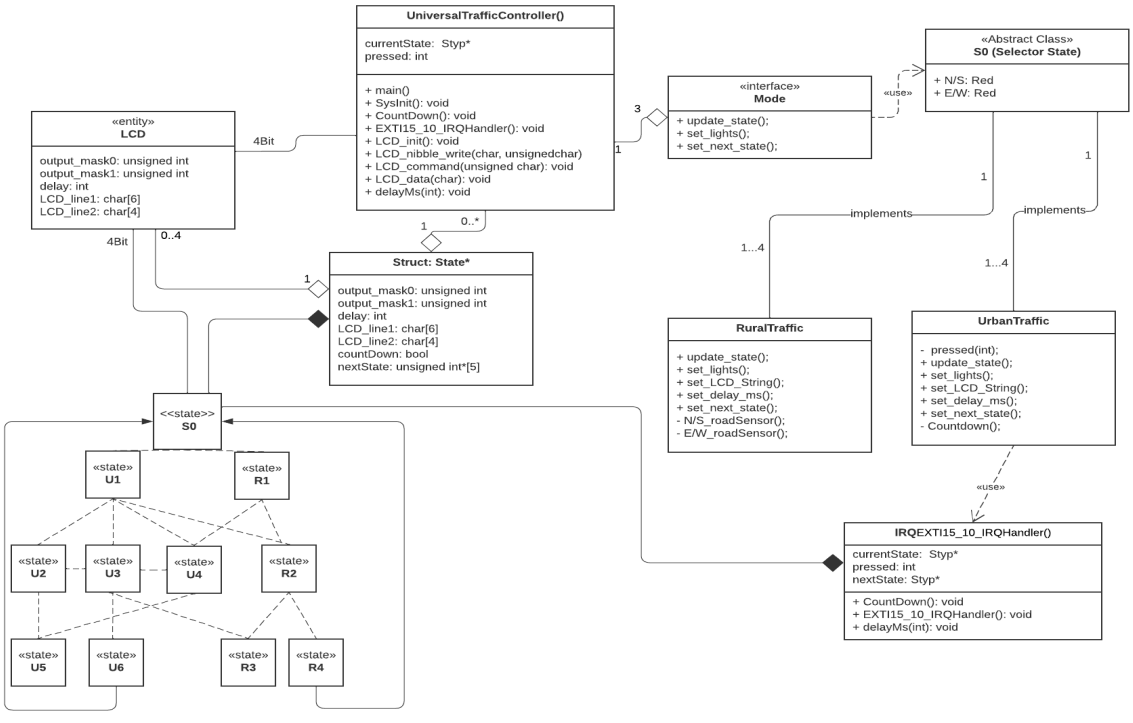


Figure 3: The UML of the software design

The *State* structure contains variables used to control the traffic light system. These variables include output masks, delays, LCD strings, and a boolean variable for invoking a countdown function. Additionally, each state has a next state array of size five, which determines the next state of the traffic light system. The *SysInit* class handles system initialization tasks such as setting up the clock and enabling the required GPIO ports. The LCD class is responsible for initializing and writing to an LCD display. The *EXTI15_10* class serves as an interrupt handler triggered when a button is pressed. The *delayMs* class is used to implement delays in milliseconds. Finally, the *UrbanTraffic* and *RuralTraffic* classes define the behavior of the traffic light system for urban and rural environments, respectively.

In the main function, the program continuously updates the traffic light system based on the current state. This involves setting the output masks of the GPIO pins to control the lights, writing appropriate information to the LCD display, and checking for button presses using the *EXTI15_10* interrupt handler. Based on the next state array defined in the *State* struct, the program transitions to the next state.

By following this structure, the program ensures the traffic light system functions correctly by controlling the GPIO pins, monitoring button presses, and displaying relevant information on the LCD display. The use of classes and a structure helps to organize the code and modularize the functionality for better readability and maintainability.

RESULTS AND VERIFICATION

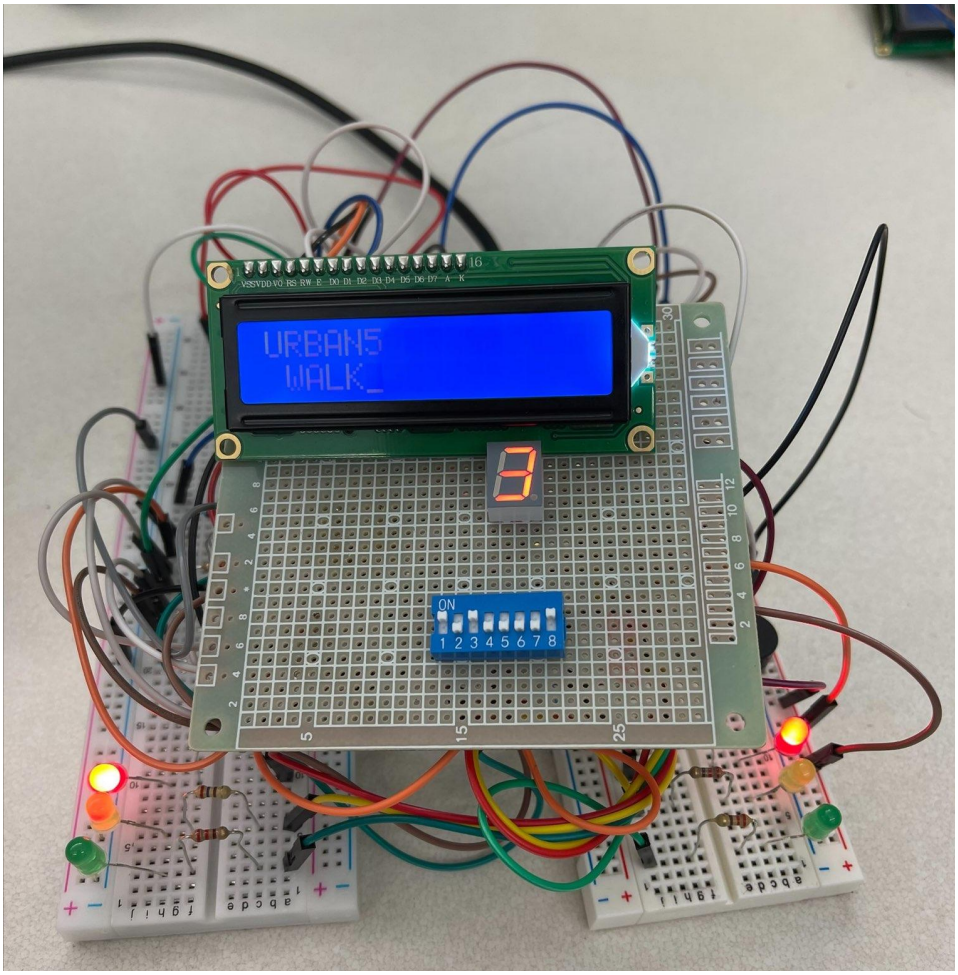
The implemented universal traffic control system was tested in both urban and rural environments, demonstrating its successful operation and adaptability to various traffic conditions. The urban traffic light sequence effectively cycled between the red, green, and yellow lights with appropriate delays for busy urban areas. The pedestrian crosswalk button activated a countdown timer on a digital display, accompanied by auditory cues for enhanced pedestrian awareness.

Smooth transitions were observed from the urban to rural sequences, considering vehicle presence at each intersection. The system promptly switched traffic lights when cars were detected, prioritizing

traffic flow. Continuous monitoring of vehicle presence allowed real-time adjustments of the traffic light state, responding effectively to changing traffic conditions.

The mode selection feature enabled graceful switching between urban and rural traffic light sequences, ensuring smooth transitions without disruptions. In conclusion, the experiment provided evidence of the universal traffic control system's functionality and adaptability. It performed well in both urban and rural settings, accommodating different traffic patterns and priorities. The finite state machine (FSM) design improved modularity, readability, and scalability.

Overall, the universal traffic control system offers a versatile and adaptable solution for managing traffic in diverse environments. It enhances traffic flow, ensures pedestrian safety, and effectively manages different traffic conditions, making it a robust and flexible solution for traffic control.



DISCUSSIONS

Implementing a universal traffic light controller using a large finite state machine (FSM) offers numerous benefits. Initially, the approach may seem tedious; however, it provides modularity, enabling simpler scalability. To incorporate new features into the system, only a few states need to be created and rerouted. Furthermore, the well-defined transitions associated with each state facilitate easier debugging, as the transition between states can be easily traced and analyzed. Structuring the code as an FSM for the traffic light system brings about several advantages. It improves modularity and readability, as each state is encapsulated within a specific state type, leading to a more organized and comprehensible codebase. This modular approach also simplifies maintenance and extensibility, making it straightforward to add or modify states. Additionally, the FSM representation with the next state array in the State struct offers clarity in state transitions, providing a structured method to determine the next state based on the current state and input signals. Consequently, following the flow of the traffic light system becomes more manageable.

Using an FSM also enhances scalability. It allows for easy expansion and modification of the system, as new states can be added or existing states can be changed by updating the corresponding state, functions, or objects and adjusting the next state array accordingly. This flexibility is valuable when system requirements need new features to be implemented. The structured approach ensures that each state's behavior is well-defined and isolated, reducing the likelihood of unintended side effects or bugs. Additionally, FSMs make it easier to handle exceptional cases and error conditions by including specific states to address such scenarios.

FSMs also contribute to improved testability and debugging. With each state being independent, it becomes easier to write unit tests for individual states, ensuring their correctness. Debugging is also simplified since it is easier to isolate and analyze the behavior of each state and its transitions separately. Overall, using an FSM for implementing the traffic light system enhances modularity, readability,

flexibility, reliability, and testability. It provides a structured and organized approach to handle different states and transitions, leading to a more maintainable and robust codebase.

Nonetheless, there is considerable improvement from the first electric traffic light to the universal traffic light controller constructed in this experiment. However, there is more room for advancements that enhance the efficiency and flow of traffic. One particular advancement to the universal traffic light controller can be a more flexible model that not only combines both urban and rural models, but also allows seamless, automatic switching between them based on the traffic density. This allows the traffic light to respond to real-time traffic conditions without needing a technician or operator to manually switch between modes. However, the intermediary stages which signal a switch from Urban to Rural must have a longer delay and more explicit transition indicator to warn drivers and pedestrians so that no accidents occur. In some urban areas, a flashing red light may be implemented to the traffic light controller to communicate that the driver must stop and then proceed when it is safe to do so. An even more advanced implementation could incorporate AI and Machine Learning algorithms that can, with the help of video cameras, radar, and GPS data, determine the traffic density during certain times of the day and display varying signals according to this data.

WORKS CITED

[1] Design Observer. (2019, March 14). Red Light, Green Light: The Invention of the Traffic Signal. Design Observer. Accessed: May 12, 2023. Available: https://designobserver.com/feature/red-light-green-light--the-invention-of--the-traffic-signal/8627

APPENDIX

```
#include "stm32f4xx.h"
#include "stdbool.h"

void CountDown(void);
void UrbanTrafficLight(void);
void RuralTrafficLight(void);
void EXTI15_10_IRQHandler(void);
void delayMs(int n);
void delayMs(int n);
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 SysInit(void);
#define RS 0x20         /* PB5 mask for reg select */
#define EN 0x80         /* PB7 mask for enable */


struct State
{
   unsigned int output_mask0;
   unsigned int output_mask1;
          int delay;
   char LCD_line1[6];
   char LCD_line2[4];
   bool countDown;  /* true: call countDown. false: do nothing */
   unsigned int *nextState[5];
};
typedef const struct State STyp;
#define S0 &FSM[0]
#define U1 &FSM[1]
#define U2 &FSM[2]
#define U3 &FSM[3]
#define U4 &FSM[4]
#define U5 &FSM[5]
#define U6 &FSM[6]
#define R1 &FSM[7]
#define R2 &FSM[8]
#define R3 &FSM[9]
#define R4 &FSM[10]

STyp FSM[11] = {
```

```
/* S0 */                        {0xFFFF, 0x0355, 1000, {'*','*','*','*', '*', '*'}, {' ',' ',' ',' ',' '}, false,{0, 0, 0,
0, 0}},
/* U1 */                        {0xFEAB, 0x0201, 3000, {'U','R','B','A', 'N', '1'}, {' ',' ',' ',' ',' '}, false,
{S0, S0, U2, U2, 0}},
/* U2 */          {0xFDAB, 0x0101, 1000, {'U','R','B','A', 'N', '2'}, {' ',' ',' ',' ',' '}, false, {U3, U5, U3, U5,
0}},
/* U3 */                        {0xFCFA, 0x0050, 3000, {'U','R','B','A', 'N', '3'}, {' ', ' ', ' ', ' ', ' '},
false,{S0, S0, U4, U4, 0}},
/* U4 */                        {0xFCEE, 0x0044, 1000, {'U','R','B','A', 'N', '4'}, {' ',' ',' ',' ',' '},
false,{U1, U6, U1, U6, 0}},
/* U5 */                        {0xFCEB, 0x0041, 1000, {'U','R','B','A', 'N', '5'}, {'W','A','L','K'}, true,
{U3, U3, U3, U3, 0}},
/* U6 */                        {0xFCEB, 0x0041, 1000, {'U','R','B','A', 'N', '6'}, {'W','A','L','K'}, true,
{U1, U1, U1, U1, 0}},
/* R1 */                        {0xFEAB, 0x0201, 3000, {'R','U','R','A', 'L', '1'}, {' ',' ',' ',' ',' '}, false,
{R1, R1, R2, R2, S0}},
/* R2 */                        {0xFDAB, 0x0101, 1000, {'R','U','R','A', 'L', '2'}, {' ',' ',' ',' ',' '}, false,
{R3, R3, R3, R3, R3}},
/* R3 */                        {0xFCFA, 0x0050, 3000, {'R','U','R','A', 'L', '3'}, {' ', ' ', ' ', ' ', ' '}, false,
{R3, R4, R3, R4, S0}},
/* R4 */                        {0xFCEE, 0x0044, 1000, {'R','U','R','A', 'L', '4'}, {' ',' ',' ',' ',' '}, false,
{R1, R1, R1, R1, R1}}
};



int pressed = 0; /*added*/

int main(void) {

   /* initialize LCD controller */
  SysInit();
   LCD_init();

   STyp *currentState;
         currentState = S0;


         while(1) {
                  GPIOB->ODR &= currentState->output_mask0;
                  GPIOB->ODR |= currentState->output_mask1;

                  for(int i = 0; i < 6; i++)
                  {
```

```
                LCD_data(currentState->LCD_line1[i]);
        }
        for(int k = 0; k < 35; k++)
        {
                LCD_data(' ');
        }
        for(int j = 0; j < 4; j++)
        {
                LCD_data(currentState->LCD_line2[j]);
        }
        delayMs(currentState->delay);

        if(currentState->countDown)
        {
                CountDown();
        }
        LCD_command(1);
        delayMs(5);
/*      currentState = currentState->nextState[(GPIOC->IDR) & 0x00000003];*/

        if(currentState == S0)
        {
                if((GPIOC->IDR & 0x0004) == 4)
                {
                        currentState = U1;
                }
                if((GPIOC->IDR & 0x0004) == 0)
                {
                        currentState = R1;
                }
        }
        else if(currentState >= U1 && currentState <= U6)
        {
                int mode = (GPIOC->IDR & 0x0004)/2;
                currentState = currentState->nextState[mode+pressed];
        }
        else if(currentState >= R1 && currentState <= R4)
        {
                int input = GPIOC->IDR & 0x0007;
                if(input > 4)
                {
                        input = 4;
                }
                currentState = currentState->nextState[input];
```

```
                }
        }
}

void SysInit(void){
   __disable_irq();                /* global disable IRQs */
     RCC->AHB1ENR |= 7; /* enable GPIOA & GPIOC clock */
        RCC->APB2ENR |= 0x4000;           /* enable SYSCFG clock */

        GPIOA->MODER &= 0xFF5555FD;           /* clear pin mode */
        GPIOA->MODER |= 0x00555501;                                  /* set port A as Output */
         /* Config PB0,2,4,6,8,9 for output (Traffic lights). PB5 PB7 for LCD RS and EN */
         GPIOB->MODER &= 0xFFF555DD;
         GPIOB->MODER |= 0x00055511;
         /* PC4-PC7 for LCD D4-D7, respectively. PC13 for the interrupt button. PC0 and PC1
for EW/NS sensors. PC2 for mode selection */
        GPIOC->MODER &= 0xF3FF55C0;  /* clear pin mode */
        GPIOC->MODER |= 0x00005500;    /* set pin output mode */
        GPIOB->BSRR = 0x00800000;        /* turn off EN */
        SYSCFG->EXTICR[3] &= 0xFF0F;   /* clear port selection for EXTI13 */
        SYSCFG->EXTICR[3] |= 0x0020;     /* select port C for EXTI13 */
        EXTI->IMR |= 0x2000;                  /* unmask EXTI13 */
        EXTI->FTSR |= 0x2000;                 /* select falling edge trigger */
/*   NVIC->ISER[1] = 0x00000100;    */        /* enable IRQ40 (bit 8 of ISER[1]) */
        NVIC_EnableIRQ(EXTI15_10_IRQn);
        __enable_irq();                   /* global enable IRQs */
}

void CountDown(void)
{
   int sevenSeg[] = {0x001, 0x3f1, 0x001, 0x3f1, 0x001, 0x3f0, 0x061, 0x5b0, 0x4f1, 0x660,
0x6d1, 0x7d0, 0x071, 0x7f0, 0x6f1};
   int i;
   for(i = 14; i>=0; i--)
                {
                        GPIOA->ODR = sevenSeg[i];
                        delayMs(500);
                }
                pressed = 0;
}

/* Interrupt function */
void EXTI15_10_IRQHandler(void) {
   pressed = 1;
```

```
/*    GPIOA->ODR |= 0x00000001;
   delayMs(300);  */
/*    CountDown(); */
   EXTI->PR = 0x2000;  /* reset PR */
/*    GPIOA->ODR &= 0xFFFE;*/
}

/* initialize GPIOB/C then initialize LCD controller */
void LCD_init(void) {

        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 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)
        GPIOB->BSRR = RS;
        else
        GPIOB->BSRR = RS << 16;

        /* pulse E */
        GPIOB->BSRR = EN;
        delayMs(0);
        GPIOB->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(2);
}

/* delay n milliseconds (16 MHz CPU clock) */
void delayMs(int n) {
   int i;
   SysTick->LOAD = 16000;
   SysTick->VAL = 0;
   SysTick->CTRL = 0x5;
   for(i = 0; i < n; i++){
         while((SysTick->CTRL & 0x10000) == 0)
         {}
   }
   SysTick -> CTRL = 0;
}
```