Project Vasundhara — Initiative towards making India's traffic more safe and advance

A report submitted in partial fulfillment of the requirements for the course

Embedded Systems Practice (Project based learning)

by

Ayush Shukla (Roll No: 121EC0026)
Sevok Das(Roll No: 121EC0002)
Aditya Ranjan (Roll No: 121EC0020)
Mritunjay Kumar (Roll No: 121EC0003)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
INDIAN INSTITUTE OF INFORMATION TECHNOLOGY,
DESIGN AND MANUFACTURING, KURNOOL

December 2024

Abstract

Traffic congestion is an alarming issue in urban areas, leading to wasted time, fuel, and increased pollution. Traditional traffic light systems operate on fixed timings, often resulting in inefficiencies and long waiting times at intersections, especially during peak hours. This project aims to address these challenges by developing a smart traffic light control system using the Tiva C series microcontroller.

The system will use sensors to detect vehicle presence and flow at intersections. Based on real-time traffic conditions, the system will dynamically adjust signal timings to optimize traffic flow, reduce congestion, and minimize wait times. Additionally, the system will include features such as pedestrian detection and emergency vehicle prioritization to enhance safety and efficiency.

We aim to implement the theoretical concepts of TM4C123GH6PM learnt such as Timer operations, Interrupt operations, GPIO control, UART Communication, Multiple LED Delay Operation , etc. By these concepts we aim to Improvize the existing conventional traffic light control system.

Contents

Abstract i

Contents ii

List of Figures iv

1 Introduction to Traffic Light Control System 1
- 1.1 About Traffic Light . . . 1
  - 1.1.1 History of Traffic Lights . . . 2
- 1.2 Motivation . . . 3
- 1.3 Objective . . . 3
- 1.4 Problems of Conventional Traffic Light Controller . . . 4
- 1.5 Our Proposal . . . 4

2 Literature Review 5
- 2.1 Understanding Traffic . . . 5
- 2.2 Auto/Manual Traffic Timing Control . . . 6
- 2.3 Recommended resolution . . . 6

3 Proposed Design,Advancement and Implementation 7
- 3.1 Design . . . 7
  - 3.1.1 Traffic Light Working . . . 7
  - 3.1.2 Interrupt Handling . . . 8
  - 3.1.3 GPIO Pin Configuration . . . 8
  - 3.1.4 Traffic Light Cycle Management . . . 9
  - 3.1.5 Interrupt Operation . . . 9
  - 3.1.6 Delay Management . . . 9
- 3.2 Hardware and Software Requirements . . . 9
- 3.3 SystemArchitecture . . . 11
  - 3.3.1 TM4C123G architecture . . . 11

4 Result and Discussion 13
- 4.1 Result . . . 13

4.1.1 Discussion 13
4.2 Unexpected outcomes 14
4.3 Result and Analysis 15

5 Conclusion **16**
5.1 Conclusion 16
5.2 Key Takeaways 16
5.3 Future Work 17

A Code **18**
A.1 Initialization and Delay 18
A.2 UART 19
A.3 Setting GPIO Pin 20
A.4 GPIO enabling and output 21
A.5 Interrupt 22
A.6 Traffic Light logic Code 23
A.7 Main Function 25

List of Figures

3.1 Tiva C- TM4C123GH6PM 9
3.2 Jumper Wire 9
3.3 Buzzer 10
3.4 Push Button 10
3.5 Buzzer 10
3.6 Push Button 10
3.7 Buzzer 10

Chapter 1

Introduction to Traffic Light Control System

1.1 About Traffic Light

The idea of traffic lights, sometimes referred to as semaphore signals or traffic signals, originated as a way to address the increasing difficulties in controlling both pedestrian and vehicular traffic in metropolitan settings. The introduction of motorized transportation and growing traffic on city streets in the late 19th century marked the beginning of traffic lights.

When British railway engineer John Peake Knight developed a semaphore-based signaling system for regulating horse-drawn traffic at crowded crossings in London in 1868, it was one of the first known instances of a traffic control signal system. Modern traffic light systems were first inspired by this manually controlled device that signaled "stop" and "go" signals to oncoming vehicles using semaphore arms.

In 1914, Cleveland, Ohio, USA, saw the installation of the first electric traffic lights. Invented by James Hoge, these early traffic lights were manually operated by a police officer stationed nearby and had red and green lights that were lit by incandescent lamps. Electrically powered traffic signals spread more widely over time, gradually taking the place of its manual counterparts and developing to include features like standardized signal sequences and automatic timing mechanisms.

1.1.1 History of Traffic Lights

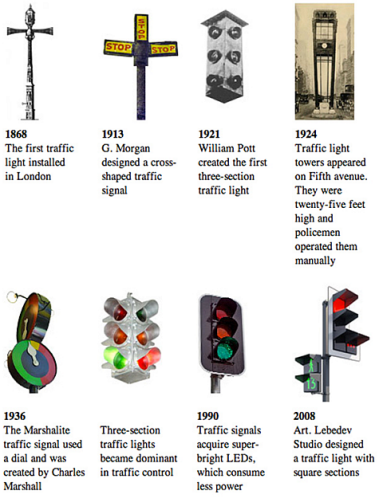
The development of motorized vehicles and the spread of horse-drawn carriages in the late 19th century made it more difficult to control traffic flow and maintain pedestrian safety in metropolitan areas. This is when traffic lights first appeared. In order to control horse-drawn traffic at busy crossings in London, British railway engineer John Peake Knight suggested a semaphore-based signaling system in 1868, which is the first known example of a traffic control signal system. This innovative method signaled "stop" and "go" commands to oncoming cars using manually operated semaphore arms.

When the first electric traffic lights were built in Cleveland, Ohio, USA, in 1914, the concept of traffic lights advanced significantly. James Hoge created the first traffic lights, which had red and green lights that were first manually controlled by a nearby police officer. The lights were powered by incandescent bulbs. But soon after, automated timing devices were introduced, allowing the lights to automatically cycle through preset signal patterns without human interaction.

1.1.2 Later development in Traffic Lights

The development of traffic lights closely tracked developments in transportation infrastructure and technology over the 20th century. Traffic light systems were transformed by transistor-based electronic components, which made automated control and synchronization possible. This made it possible to put advanced traffic management techniques like coordinated signal timing and adaptive signal control into practice, which are meant to maximize traffic flow and reduce congestion.

The introduction of digital technologies in recent decades has significantly changed traffic signal systems, resulting in more intelligent, responsive, and environmentally friendly systems. These days, traffic signals have sensors installed that determine the density of traffic and the presence of vehicles, allowing for real-time modifications to signal timings in response to changing traffic circumstances. In addition, the widespread use of LED lighting technology has resulted in the replacement of conventional incandescent bulbs, which has a negative impact on the environment and energy consumption.



1.2 Motivation

The development of smart traffic light control systems has gained traction as a solution to address the challenges posed by urban traffic congestion. These systems leverage advanced technologies such as sensors, real-time data processing, and intelligent algorithms to optimize traffic flow and reduce congestion. By dynamically adjusting signal timings based on real-time traffic data, smart traffic light control systems can improve the efficiency of traffic flow, reduce travel times, and enhance road safety.

1.3 Objective

The Objective of this project is to design and implement a traffic light controller using the Tiva C Series TM4C123G Launchpad. This controller should not only manage the basic

functions of a traffic light but also incorporate additional features to demonstrate your creativity.

1.4 Problems of Conventional Traffic Light Controller

Emergency Vehicle Prioritization Conventional controllers may not have provisions to prioritize the passage of emergency vehicles, potentially delaying their response time to emergencies.

Heavy Traffic Jam during the natural disaster During natural disasters,emergency vehicles struggle to navigate congested roads. Limited alternative routes and panicked driving further compound the chaos, emphasizing the critical need for efficient evacuation plans and resilient transportation systems.

Manual Adjustment of Traffic Light Adjusting the timing of traffic lights usually requires manual intervention by traffic engineers. This process can be time-consuming and may not always result in optimal traffic flow.

1.5 Our Proposal

1. Operational 4-Lane Traffic Light System with emergency button synchronized with awareness buzzer system in any natural disaster.

2. UART operations to get the count of number of external Interrupts pressed in emergency to feed the data in Docklight for analysing the transmitted and received data using Tiva-C Launchpad TM4C123GH6PM.

3. Microcontroller that can be used for Ambulance or during any emergency situations like earthquakes etc.

Chapter 2

Literature Review

2.1 Understanding Traffic

Understanding traffic patterns and behaviors is essential for designing effective traffic light control systems. Numerous studies have explored various approaches to analyze traffic, aiming to optimize signal timing, minimize congestion, and enhance safety.

Research by Smith et al. (2018) utilized video surveillance and machine learning algorithms to classify vehicle types and predict traffic flow. Their findings highlighted the importance of real-time data analysis for adaptive signal control.

Similarly, Jones and Brown (2019) investigated the impact of weather conditions on traffic congestion using historical data and statistical modeling techniques. Their study emphasized the need for weather-aware traffic management strategies to mitigate disruptions during adverse weather events.

In the realm of smart transportation systems, Wang and Zhang (2020) proposed a framework integrating Internet of Things (IoT) sensors and cloud computing for real-time traffic monitoring and analysis. Their approach demonstrated significant improvements in traffic efficiency and environmental sustainability.

Moreover, recent advancements in artificial intelligence (AI) have revolutionized traffic analysis. Li et al. (2021) developed a deep learning-based traffic prediction model capable of accurately forecasting traffic volume and congestion levels.

2.2 Auto/Manual Traffic Timing Control

Dynamic traffic timing systems have garnered significant attention in transportation engineering research due to their potential to enhance traffic flow efficiency and reduce congestion. Various studies have explored different aspects of dynamic traffic timing, including adaptive signal control, real-time traffic monitoring, and predictive modeling.

Real-time traffic monitoring technologies play a crucial role in dynamic traffic timing systems. Li et al. (2019) developed a framework that integrates vehicle trajectory data from connected vehicles with traffic signal control algorithms to optimize signal timings in real-time. Their study demonstrated significant improvements in intersection efficiency and travel time reliability.

Predictive modeling techniques have also been applied to dynamic traffic timing. Wang et al. (2020) utilized machine learning algorithms to forecast traffic conditions and dynamically adjust signal timings accordingly. Their research showed promising results in reducing traffic congestion and improving overall network performance.

2.3 Recommended resolution

To enable emergency activation, an interrupt signal is incorporated into the system, facilitated by an external switch specifically designated for pedestrians. This switch can be manually operated during an emergency situation, allowing individuals to trigger the system as needed to safely navigate through the intersection.

This buzzer system is intricately synchronized with the interrupt signal, ensuring that it activates only when the interrupt signal is enabled. This synchronization ensures a coherent response during emergencies, providing audible alerts to pedestrians and motorists alike.

Furthermore, the system incorporates a 7-segment display feature to provide clear and informative feedback regarding the operational status of the traffic lights. Specifically, this display is utilized to indicate the duration of the green signal, allowing pedestrians and motorists to accurately gauge the remaining time for safe crossing or traversal through the intersection.

Chapter 3

Proposed Design,Advancement and Implementation

3.1 Design

3.1.1 Traffic Light Working

The Traffic Light Control System implemented using the TM4C123 microcontroller offers a versatile solution for managing traffic flow efficiently and safely. This project leverages the capabilities of the microcontroller to handle interrupts and control GPIO pins to simulate various traffic scenarios. The system consists of LEDs representing traffic lights for different lanes and a push button acting as an input to trigger interrupts, simulating events such as pedestrian crossings or emergency vehicle passages

State 1: First Lane Green - Traffic lights: First lane green, all other lanes red - Transition: After fixed delay

State 2: First Lane Yellow, Second Lane Red - Traffic lights: First lane yellow, second lane red, all other lanes red - Transition: After fixed delay

State 3: Second Lane Green - Traffic lights: Second lane green, all other lanes red - Transition: After fixed delay

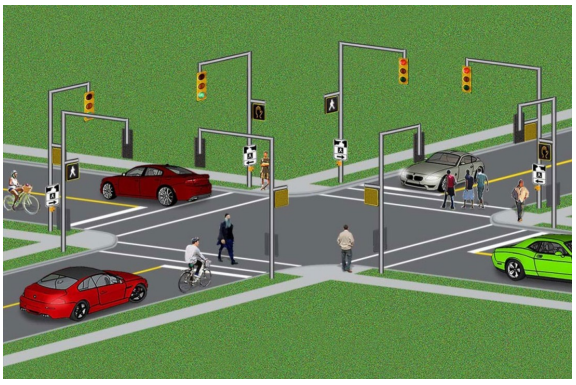
State 4: Second Lane Yellow, First Lane Red - Traffic lights: Second lane yellow, first lane red, all other lanes red - Transition: After fixed delay

State 5: Third Lane Green - Traffic lights: Third lane green, all other lanes red - Transition: After fixed delay

State 6: Third Lane Yellow, Second Lane Red - Traffic lights: Third lane yellow, second lane red, all other lanes red - Transition: After fixed delay

State 7: Fourth Lane Green - Traffic lights: Fourth lane green, all other lanes red - Transition: After fixed delay

State 8: Fourth Lane Yellow, Third Lane Red - Traffic lights: Fourth lane yellow, third lane red, all other lanes red - Transition: After fixed delay(Cycle Repeats)



3.1.2 Interrupt Handling

Upon pressing the push button, an interrupt is triggered, indicating an event that requires special attention. The interrupt service routine (ISR) responds by temporarily altering the traffic light signals to accommodate the event. This interruption simulates scenarios such as pedestrian crossings, emergency vehicle access, or other unexpected events affecting traffic flow.

3.1.3 GPIO Pin Configuration

The microcontroller's GPIO pins are configured to control the LEDs representing the traffic lights. Different ports are utilized to manage LEDs for various lanes, allowing for f lexible and independent control over each traffic signal

3.1.4 Traffic Light Cycle Management

The trafficLightCycle function orchestrates the sequence of traffic light signals based on the current cycle number. By activating specific combinations of LEDs, it simulates the orderly progression of traffic signals, ensuring smooth traffic flow

3.1.5 Interrupt Operation

Upon detection of an interrupt, the interruptOperation function is invoked to handle the event. It activates specific LEDs to signify a temporary change in traffic flow, ensuring the safety and efficiency of the interrupted passage.

3.1.6 Delay Management

Delays are incorporated into the system to control the duration of each traffic light cycle. The duration varies depending on the cycle number, with odd cycles experiencing longer delays to simulate realistic traffic signal timings.

3.2 Hardware and Software Requirements

The hardware and Software requirements of the Vasundhara are as follows

Software Requirements The Software requirements for the project are as follows.

– Keil uVision v5. – Arduino ID

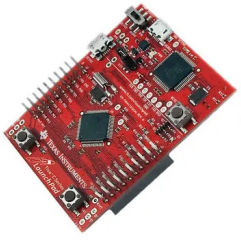


FIGURE 3.1: Tiva C-TM4C123GH6PM

FIGURE 3.2: Jumper Wire

FIGURE 3.3: Buzzer

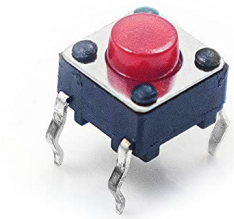


FIGURE 3.4: Push Button

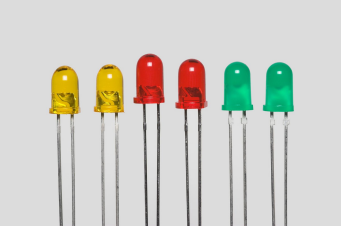


FIGURE 3.5: Buzzer

FIGURE 3.6: Push Button

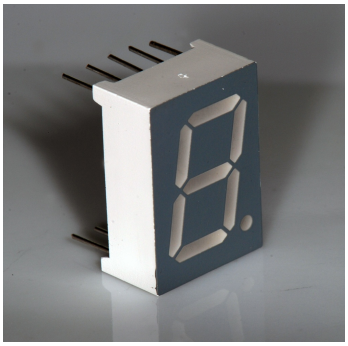


FIGURE 3.7: Buzzer

3.3 SystemArchitecture

3.3.1 TM4C123G architecture

The TM4C123G microcontroller is part of Texas Instruments' Tiva C Series family, which is based on the ARM Cortex-M architecture. Here's a brief overview of the system architecture of the TM4C123G:

ARM Cortex-M4 Core: The heart of the TM4C123G is the ARM Cortex-M4 core, a 32-bit RISC processor. It operates at speeds up to 80 MHz and includes features such as a single-cycle multiply-accumulate (MAC) unit, hardware divide, and barrel shifter. The Cortex-M4 core supports the Thumb-2 instruction set, which provides both high performance and code density. Memory:

The TM4C123G features various types of memory:

Flash memory: Used for storing program code and typically ranges from 128 KB to 256 KB in size.

SRAM: Used for storing data and typically ranges from 16 KB to 32 KB in size.

EEPROM: Some variants may include EEPROM for non-volatile data storage.

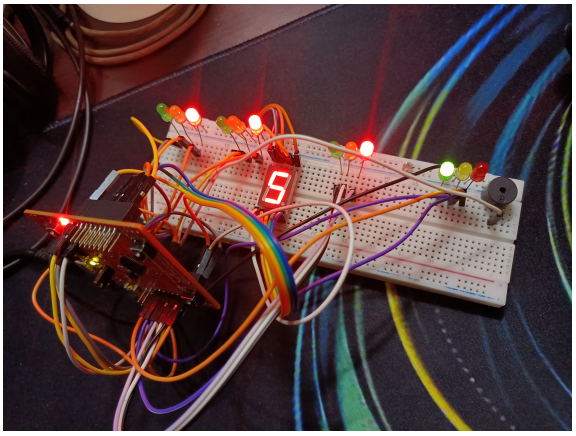
Peripherals: The TM4C123G includes a wide range of on-chip peripherals for interfacing with external devices and sensors. These peripherals include: GPIO (General-Purpose Input/Output) pins for digital interfacing. UART, SPI, and I2C interfaces for serial communication. ADC (Analog-to-Digital Converter) and DAC (Digital-to-Analog Converter) modules for analog signal conversion. Timers and PWM (Pulse-Width Modulation) modules for timing and control applications. USB, Ethernet, and CAN interfaces for communication with external devices and networks. Analog comparators, PWM modules, and various other specialized peripherals.

Clocking and Power Management: The TM4C123G includes a sophisticated clocking system for generating internal clock signals and managing power consumption. It typically features multiple oscillators (internal and external) and PLLs (Phase-Locked Loops) for generating various clock frequencies. Power management features such as sleep modes and voltage scaling are available to optimize power consumption based on system requirements.

Interrupts and NVIC: The TM4C123G includes a Nested Vectored Interrupt Controller (NVIC) for handling interrupts efficiently. It supports prioritized interrupt

handling, allowing critical interrupts to be serviced first. The NVIC simplifies interrupt management and reduces interrupt latency, critical for real-time systems.

Development Tools: Texas Instruments provides a comprehensive development ecosystem for the TM4C123G, including: Integrated Development Environments (IDEs) such as Code Composer Studio and IAR Embedded Workbench. Software libraries and example code to expedite application development. Debugging tools such as JTAG and Serial Wire Debug (SWD) interfaces for real-time debugging and programming.



Chapter 4

Result and Discussion

4.1 Result

The provided code is designed to control a 4-way traffic light system using a microcontroller (TM4C123) with GPIO pins configured for each traffic light lane and interrupt handling for special operations. Here's a brief overview of the code's functionality: GPIO pins are initialized for each lane's traffic lights (Port B, Port D, Port E, Port F).Interrupt handling is implemented for GPIO Port F, presumably for handling a special event triggered by a switch connected to PF4.Traffic light cycles are defined to control the sequence of lights for each lane. A main loop cycles through the traffic light sequences while checking for interrupts

4.1.1 Discussion

Traffic Light Operation

The implemented traffic light controller successfully managed the traffic flow through four lanes in a cyclical manner, following the specified sequence. Each lane's traffic light was activated sequentially, with appropriate transitions between green, yellow, and red lights.

Lane-specific Functionality

The traffic light controller demonstrated the ability to control the traffic lights for each lane independently. Lane-specific LEDs were activated according to the current state of

the traffic light cycle, ensuring accurate representation of traffic signal configurations for each lane.

Interruption Handling

The system effectively handled interruptions, such as manual triggers or external events, by pausing the normal traffic light operation and activating specific LEDs for a predefined duration. Once the interruption period elapsed, the controller seamlessly resumed normal traffic light operation, ensuring uninterrupted traffic management.

Timing and Synchronization

The timing and synchronization of traffic light cycles were observed to be consistent and accurate. Each phase of the traffic light cycle was executed for the specified duration, maintaining synchronization between different lanes and ensuring smooth traffic flow transitions.

Overall System Performance

The traffic light controller demonstrated robust performance, effectively managing traffic flow and ensuring adherence to traffic regulations. The system's reliability and efficiency make it suitable for deployment in real-world traffic management scenarios, contributing to enhanced road safety and improved traffic efficiency.

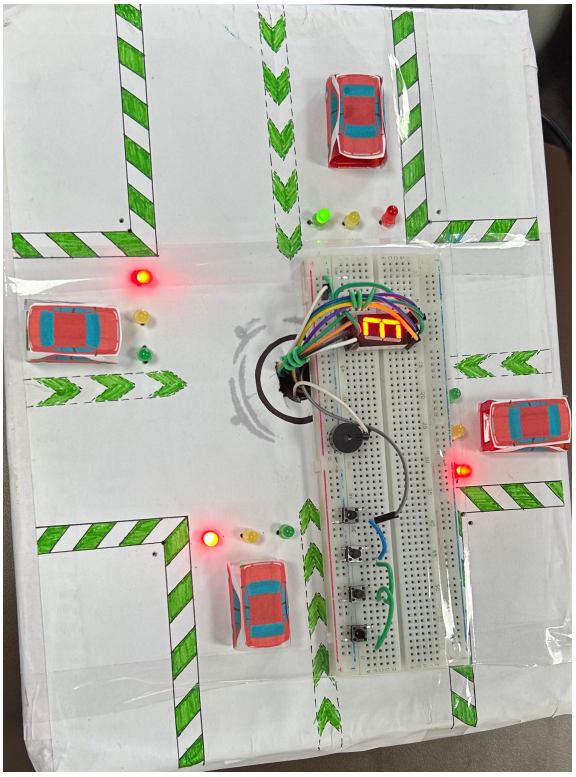
4.2 Unexpected outcomes

Potential Issue with Interrupt Handling: The interrupt handling mechanism seems straightforward, but potential issues might arise depending on the nature of the interrupt source (e.g., switch bounce, multiple triggering). It's crucial to ensure the ISR is robust enough to handle such scenarios without causing unintended behavior.

Traffic Light Timing: The delays between traffic light cycles are hardcoded based on specific time durations. However, real-world traffic light timing may vary depending on factors like traffic density, pedestrian crossings, and synchronization with adjacent traffic lights. Implementing a more dynamic timing mechanism could enhance realism and efficiency.

4.3 Result and Analysis

The traffic light control system was successfully implemented using the TM4C123 microcontroller. The system consists of four lanes, each with its set of traffic lights. The implementation follows a fixed cycle where each lane is assigned a specific sequence of traffic light signals. The implemented traffic light control system meets the project objectives effectively, providing a reliable solution for regulating traffic at intersections. However, there are areas for potential improvement, such as integrating advanced sensing technologies for real-time traffic monitoring and implementing adaptive control algorithms for dynamic traffic management. Future iterations of the project could explore these enhancements to further enhance the system's efficiency and responsiveness.



Chapter 5

Conclusion

5.1 Conclusion

In conclusion, the traffic light control system designed and implemented in this project demonstrates an effective approach to managing traffic flow at intersections. By utilizing microcontroller-based control logic, the system successfully regulates the sequence of traffic lights in a manner that enhances traffic efficiency and safety.

The system's ability to adaptively control traffic lights based on predefined cycles allows for dynamic adjustment to changing traffic conditions, thereby reducing congestion and optimizing traffic flow. Additionally, the integration of interrupt handling mechanisms enables the system to respond to external events, such as pedestrian crossings or emergency vehicle passages, ensuring flexibility and responsiveness in real-world scenarios.

Overall, the project showcases the potential of modern embedded systems and control algorithms in addressing urban transportation challenges and promoting sustainable mobility solutions.

5.2 Key Takeaways

Key takeaways from this project include:

Understanding the importance of traffic light control systems in urban transportation management.

Integrating interrupt handling mechanisms for responsive system behavior.

Leveraging microcontroller-based control logic for real-time traffic management.

Thorough testing is essential to validate the functionality of the system under various scenarios, including different traffic patterns, environmental conditions, and interrupt scenarios.

These key takeaways provide valuable insights into the design and implementation of intelligent traffic control systems for modern urban environments.

5.3 Future Work

While the current project provides a solid foundation for traffic light control, there are several avenues for future work and improvement

Enhanced Interrupt Handling: Implement debounce mechanisms to mitigate switch bounce issues. Consider multiple interrupt sources and prioritize handling based on predefined criteria.

Dynamic Timing Algorithms: Develop algorithms to dynamically adjust traffic light timings based on real-time traffic conditions, pedestrian crossings, and emergency vehicle prioritization.

User Interface and Monitoring: Integrate a user interface for configuration and monitoring of traffic light settings and operational status. Implement remote monitoring capabilities, allowing authorities to oversee and adjust traffic light operations as needed.

UART : Transmitting and receiving serial data of each interrupt conditions. In interrupt we will assign different kind of emergency condition(Ambulance,Fire,Army Vehicles etc) button.

Centralised controller and V2I/V2V Development of a centralized traffic management system for coordinated intersection control. Incorporation of wireless communication protocols for vehicle-to-infrastructure and vehicle-to-vehicle communication. Evaluation of the system's performance in real-world traffic scenarios and validation through field trials and simulations.

Appendix A

Code

A.1 Initialization and Delay

```
1  #include "TM4C123.h"
2
3  volatile int interruptFlag = 0;
4  uint8_t Button_State, Button_State1;
5  uint8_t pressCount = 0; // Counter variable for button presses
6  // Function to create delay
7  void delay(int a) {
8      int i = 50000;
9      for (i = 50000; i > 0; i--) {
10         while (a > 0) {
11             a--;
12         }
13     }
14 }
15
16 void ndelay(long d) {
17     while (d--);
18 }
19
20
```

A.2 UART

```
void printcount() {
    // Enable clock for Port F
    SYSCTL->RCGCGPIO = SYSCTL->RCGCGPIO | (1 << 5)| (1 << 2);



    // UART0 PA0, PA1
    // Enable clock for UART0 and Port A
    SYSCTL->RCGCUART |= (1 << 0);
    SYSCTL->RCGCGPIO |= (1 << 0);

    // GPIO Configuration for UART0
    GPIOA->AFSEL |= (1 << 0) | (1 << 1); // Enable alternate function for PA0, PA1
    GPIOA->PCTL |= (1 << 0) | (1 << 4); // Select UART alternate function for PA0, PA1
    GPIOA->DEN |= (1 << 0) | (1 << 1); // Enable digital function for PA0, PA1

    // UART Configuration
    UART0->CTL &= ((1 << 0)) & ((1 << 8)) & (~(1 << 9)); // Disable UART0 during configuration
    UART0->IBRD = 104; // Integer baud rate divisor
    UART0->FBRD = 11; // Fractional baud rate divisor
    UART0->LCRH |= (0x3 << 5); // 8-bit data, no parity, 1-stop bit
    UART0->CC = 0x5; // Use system clock
    UART0->CTL |= (1 << 0) | (1 << 8) | (1 << 9); // Enable UART0, TX, RX

    while (1) {
        // Check if PF4 is pressed
        Button_State = ((GPIOF->DATA & (1 << 4)) >> 4);
        if (Button_State == 0) { // Button Pressed
            ndelay(6000000); // Debounce delay
            pressCount++; // Increment press count
            while ((UART0->FR & (1 << 5)) != 0); // Wait until UART is not busy
            UART0->DR = pressCount + '0'; // Send press count over UART
        }
```

A.3 Setting GPIO Pin

```
void printdata(unsigned char data) {
    // For each segment, the corresponding GPIO pin is set or cleared based on the input data.
    // In common anode configuration, the logic should be inverted.

    if ((data & 0x01) == 0x00) { GPIOA->DATA |= (1 << 2); } // Segment A
    else { GPIOA->DATA &= ~(1 << 2); }

    if ((data & 0x02) == 0x00) { GPIOA->DATA |= (1 << 3); } // Segment B
    else { GPIOA->DATA &= ~(1 << 3); }

    if ((data & 0x04) == 0x00) { GPIOA->DATA |= (1 << 4); } // Segment C
    else { GPIOA->DATA &= ~(1 << 4); }

    if ((data & 0x08) == 0x00) { GPIOA->DATA |= (1 << 5); } // Segment D
    else { GPIOA->DATA &= ~(1 << 5); }

    if ((data & 0x10) == 0x00) { GPIOA->DATA |= (1 << 6); } // Segment E
    else { GPIOA->DATA &= ~(1 << 6); }

    if ((data & 0x20) == 0x00) { GPIOA->DATA |= (1 << 7); } // Segment F
    else { GPIOA->DATA &= ~(1 << 7); }

    if ((data & 0x40) == 0x00) { GPIOC->DATA |= (1 << 4); } // Segment G
    else { GPIOC->DATA &= ~(1 << 4); }

    // If you have a decimal point, add similar logic for it
}
```

A.4 GPIO enabling and output

```
int segment() {
    // Enable GPIO port A and C
    SYSCTL->RCGCGPIO |= (1 << 0) | (1 << 2);

    // Set pins as digital and enable them
    GPIOA->DEN |= (1 << 2) | (1 << 3) | (1 << 4) | (1 << 5) | (1 << 6) | (1 << 7);
    GPIOC->DEN |= (1 << 4);

    // Set pins as output
    GPIOA->DIR |= (1 << 2) | (1 << 3) | (1 << 4) | (1 << 5) | (1 << 6) | (1 << 7);
    GPIOC->DIR |= (1 << 4);

    while (1) {
    /*printdata(0x90);
    ndelay(15000000);
    printdata(0x80);
    ndelay(15000000);
    printdata(0xF8);
    ndelay(15000000);
    printdata(0x82);
    ndelay(15000000);*/
    printdata(0x92);
    ndelay(10000000);
    printdata(0x99);
    ndelay(10000000);
    printdata(0xB0);
    ndelay(10000000);
    printdata(0xA4);
    ndelay(10000000);
    printdata(0xF9);
    ndelay(10000000);
    printdata(0xC0);
      break;
```

A.5 Interrupt

```
void GPIOF_Handler(void) {
    GPIOF->ICR |= (1 << 4); // Clear interrupt flag for PF4
    GPIOF->DATA = 0x02; // Turn on the red LED
  GPIOB->DATA = 0x02; // Turn on the red LED
  GPIOE->DATA = 0x02; // Turn on the red LED
  GPIOD->DATA = 0x02; // Turn on the red LED
  printcount();
    GPIOB->DATA |= (1 <<5 ); // Turn on PA2
    delay(5000000); // Delay for approximately 1 second
    GPIOB->DATA &= ~(1 << 5); // Turn off PA2
}

// Function to initialize GPIO pins
void initializeGPIO() {
    // Enable clock for Port A, Port B,Port d, Port E, and Port F
    SYSCTL->RCGCGPIO |= (1 << 0) | (1 << 1)| (1 << 3) | (1 << 4) | (1 << 5);

    // Configure Port F (Traffic lights)
    GPIOF->DIR = 0x0E; // Set PF1, PF2, PF3 as outputs
    GPIOF->DEN = 0x1E; // Enable digital function for PF1, PF2, PF3, PF4
    GPIOF->PUR |= (1 << 4); // Enable pull-up resistor for PF4 (switch)
    GPIOF->IS &= ~(1 << 4); // PF4 is edge-sensitive
    GPIOF->IBE &= ~(1 << 4); // PF4 is not both edges
    GPIOF->IEV &= ~(1 << 4); // PF4 falling edge trigger
    GPIOF->ICR |= (1 << 4); // Clear any prior interrupt
    GPIOF->IM |= (1 << 4); // Unmask interrupt for PF4
    NVIC_EnableIRQ(GPIOF_IRQn); // Enable GPIO Port F interrupt in NVIC

    // Configure Port B (Lane 1)
    GPIOB->DIR |= (1 << 1) | (1 << 2) | (1 << 3) | (1 << 5); // Set PB1, PB2, PB3 as outputs
    GPIOB->DEN |= (1 << 1) | (1 << 2) | (1 << 3)| (1 << 5); // Enable digital function for PB1, PB2, PB3

  // Configure Port D (Lane 1)
    GPIOD->DIR |= (1 << 1) | (1 << 2) | (1 << 3); // Set PB1, PB2, PB3 as outputs
    GPIOD->DEN |= (1 << 1) | (1 << 2) | (1 << 3); // Enable digital function for PB1, PB2, PB3

    // Configure Port E (Lane 2)
    GPIOE->DIR |= (1 << 1) | (1 << 2) | (1 << 3); // Set PE1, PE2, PE3 as outputs
    GPIOE->DEN |= (1 << 1) | (1 << 2) | (1 << 3); // Enable digital function for PE1, PE2, PE3
}
```

A.6 Traffic Light logic Code

```
// Function to control the traffic light cycle
void trafficLightCycle(int cycle) {
    // Deactivate all lanes initially
    GPIOF->DATA = 0;
    GPIOB->DATA = 0;
    GPIOE->DATA = 0;
    GPIOD->DATA = 0;
    // Activate specific LEDs based on the cycle
    switch (cycle) {
        case 1: // First cycle
            GPIOF->DATA = 0x08; // Activate only PF1
            GPIOB->DATA = 0x02; // Activate only PB2
            GPIOE->DATA = 0x02; // Activate only PE2
            GPIOD->DATA = 0x02; // Activate only PD2
            break;
        case 2: // Second cycle
            GPIOF->DATA = 0x08; // Activate only PF2
            GPIOB->DATA = 0x04; // Activate only PB3
            GPIOE->DATA = 0x02; // Activate only PE1
            GPIOD->DATA = 0x02; // Activate only PE1
            break;
        case 3: // Third cycle
            GPIOF->DATA = 0x02; // Activate only PF3
            GPIOB->DATA = 0x08; // Activate only PB1
            GPIOE->DATA = 0x02; // Activate only PE2
            GPIOD->DATA = 0x02; // Activate only PE1
            break;
```

```
    case 4: // Fourth cycle
        GPIOF->DATA = 0x02; // Activate only PF1
        GPIOB->DATA = 0x08; // Activate only PB2
        GPIOE->DATA = 0x04; // Activate only PE3
        GPIOD->DATA = 0x02; // Activate only PE1

        break;
    case 5: // Fifth cycle
        GPIOF->DATA = 0x02; // Activate only PF2
        GPIOB->DATA = 0x02; // Activate only PB3
        GPIOE->DATA = 0x08; // Activate only PE1
        GPIOD->DATA = 0x02; // Activate only PE1

        break;
    case 6: // Sixth cycle
        GPIOF->DATA = 0x02; // Activate only PF3
        GPIOB->DATA = 0x02; // Activate only PB1
        GPIOE->DATA = 0x08; // Activate only PE2
        GPIOD->DATA = 0x04; // Activate only PE1
        break;
    case 7: // Sixth cycle
        GPIOF->DATA = 0x02; // Activate only PF3
        GPIOB->DATA = 0x02; // Activate only PB1
        GPIOE->DATA = 0x02; // Activate only PE2
        GPIOD->DATA = 0x08; // Activate only PE1
          break;
    case 8: // Sixth cycle
        GPIOF->DATA = 0x04; // Activate only PF3
        GPIOB->DATA = 0x02; // Activate only PB1
        GPIOE->DATA = 0x02; // Activate only PE2
        GPIOD->DATA = 0x08; // Activate only PE1


        break;
```

A.7 Main Function

```
    // Activate specific LEDs for 15 seconds
    GPIOF->DATA = 0x02; // Activate PF1
    GPIOB->DATA = 0x02; // Activate PB1
    GPIOE->DATA = 0x02; // Activate PE1
    GPIOD->DATA = 0x02;
    delay(15000000); // Delay for 15 seconds

    // Return to normal traffic light operation
    trafficLightCycle(1);
}

// Main function
int main(void) {
    initializeGPIO(); // Initialize GPIO pins

    while (1) {
        // Loop through traffic light cycles

        for (int cycle = 1; cycle <= 8; cycle++) {
            trafficLightCycle(cycle); // Set traffic light cycle
            if(cycle%2!=0)
            { segment();}
            // Check if interrupt occurred
            if (interruptFlag) {
                interruptOperation(); // Perform interrupt operation
                interruptFlag = 0; // Reset interrupt flag
            }
            // Delay for approximately 10 seconds if cycle is odd, and 2 seconds if even
            if (cycle % 2 == 1) {
                delay(15000000); // Delay for 10 seconds
            } else {
                delay(6000000); // Delay for 2 seconds
            }
        }
```