Ministry of Education and Research of the Republic of Moldova
Technical University of Moldova
Department of Software and Automation Engineering

RAPORT

Laboratory work No. 6.2

Disciplină: Sisteme Incorporate

Elaborat: Berco Andrei, FAF - 221

Verificat: asist. univ., Martiniuc A.

Chișinău 2025

1. OBJECTIVES

Main purpose of the work: Developing a modular application for a microcontroller (MCU) that implements a smart traffic light system for an intersection with two traffic directions (East-West and North-South), using a finite state machine and the FreeRTOS real-time operating system. The system controls traffic light states based on requests coming from sensors (simulated with buttons).

The objectives of the work:

- Understanding the implementation of finite state machines in the context of behavioral control
- Implementing traffic light control for two directions of traffic using FreeRTOS
- Synchronizing state transitions through FreeRTOS mechanisms (tasks, delays)
- Handling reactions to requests from digital sensors
- Displaying the current state of the traffic light through the serial interface
- Implementing a night mode with flashing yellow lights as an additional behavior

Problem Definition: The task requires designing and implementing an application that controls a smart traffic light at a cross intersection with two traffic directions:

- East-West direction
- North-South direction

The traffic light must exhibit the following characteristics:

- Transition from one green direction to the other based on an active request (simulated with a button)
- If there's no request for the North-South direction, the East-West direction has priority and remains green
- Transitions must follow traffic signaling rules: green → yellow → red
- Display time for each state (green, yellow, red) is controlled through FreeRTOS delays
- Implementation of a finite state machine for controlling each direction
- Current state of the traffic light must be displayed through STDIO
- Implementation of a night mode with flashing yellow lights as an additional behavior

2. DOMAIN ANALYSIS

1. Technology stack

In order to complete this laboratory work, I used the following hardware components:

- Microcontroller: Arduino Uno
- 6 LEDs (2 green, 2 yellow, 2 red) for both traffic directions
- 2 Pushbuttons (for request simulation and night mode toggling)
- Resistors (220Ω for LEDs, 10kΩ for buttons)
- Breadboard and jumper wires
- USB connection for programming and serial communication

For the software layer of the work, I used:

- Visual Studio Code with PlatformIO extension for embedded development

- FreeRTOS real-time operating system (Arduino_FreeRTOS library)
- Standard I/O library for serial communication
- Arduino core libraries for basic I/O operations

2. Use cases

Smart traffic light systems have numerous practical applications in modern urban infrastructure:

Adaptive Traffic Control: In real-world implementations, smart traffic lights can adapt to traffic patterns by giving green signals to directions with higher vehicle density, reducing overall waiting times and congestion. This is particularly useful at intersections with uneven traffic distribution.

Energy Conservation: By remaining in a default state (East-West green in our implementation) until needed elsewhere, the system conserves energy by minimizing unnecessary light changes, which can lead to reduced electricity consumption and maintenance costs.

Emergency Vehicle Priority: While not implemented in our basic model, real-world extensions of this system could include priority overrides for emergency vehicles, automatically changing lights to green along their route, potentially saving critical minutes during emergencies.

Pedestrian Safety: Smart traffic lights integrate pedestrian crossing buttons that, when pressed, safely transition traffic flow to allow pedestrian crossing, improving safety at busy intersections.

Night Mode Operation: Our implementation includes a night mode with flashing yellow lights, which is commonly used during low-traffic hours to indicate caution while not requiring vehicles to make full stops at empty intersections.

3. PROJECT DESIGN

3.1 Architecture schema

The following diagram (Figure 3.1) represents the architectural schema of the smart traffic light system. It shows the flow of control signals and the interaction between different components.

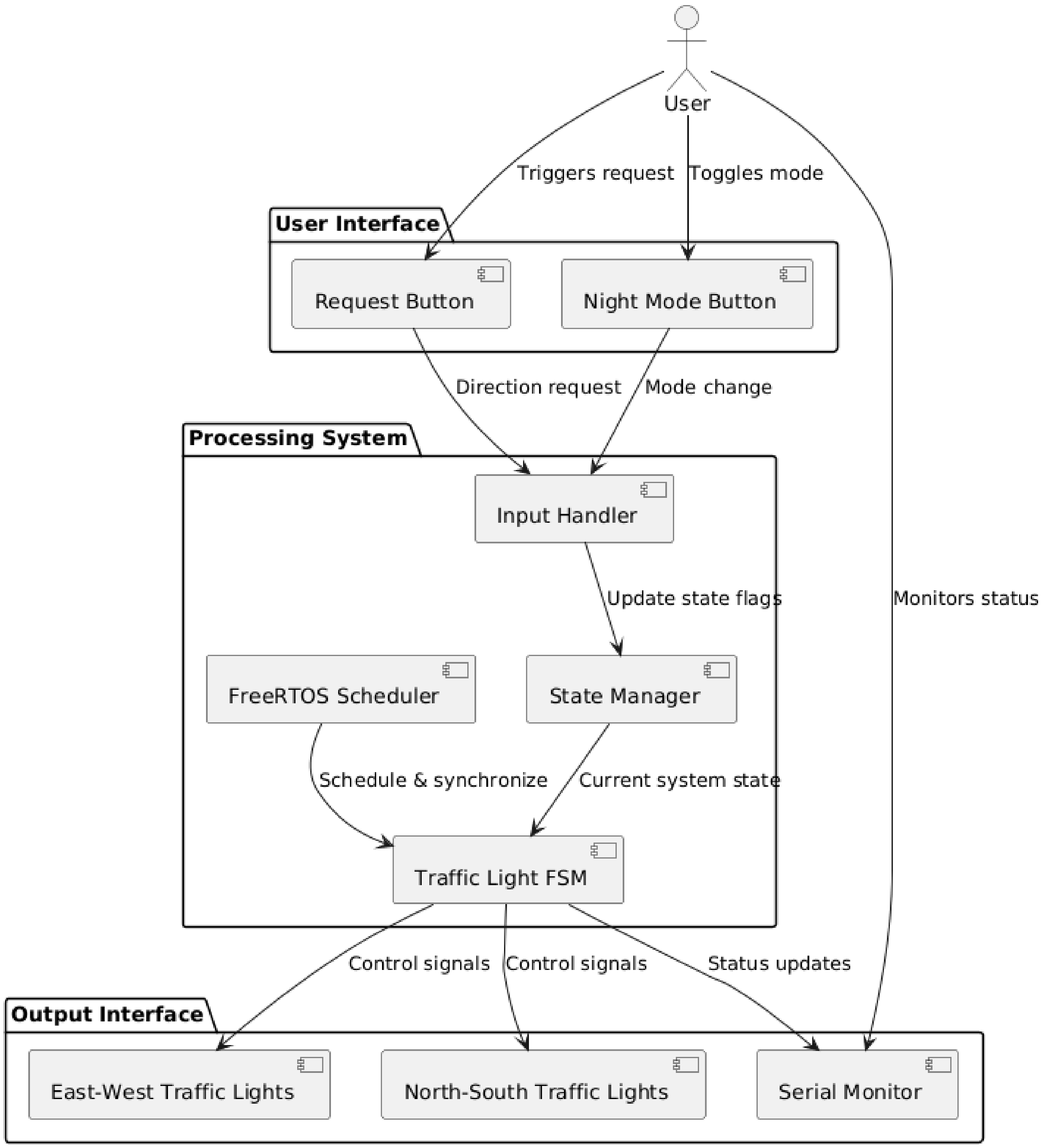


Figure 3.1 System Architecture Schema

The system architecture consists of three main components:

1. **User Interface**: This includes the physical buttons for triggering requests and toggling night mode, as well as the Serial Monitor for displaying system status.

2. **MCU - Arduino**: This central component contains the FreeRTOS kernel which handles task scheduling, the Traffic Light Finite State Machine (FSM) that manages the traffic light states, and handlers for processing inputs and outputs.
3. **Visual Indicators**: These are the physical LEDs that represent the traffic lights for both the East-West and North-South directions.

The next diagram (Figure 3.2) illustrates the software layered architecture for the traffic light control system:

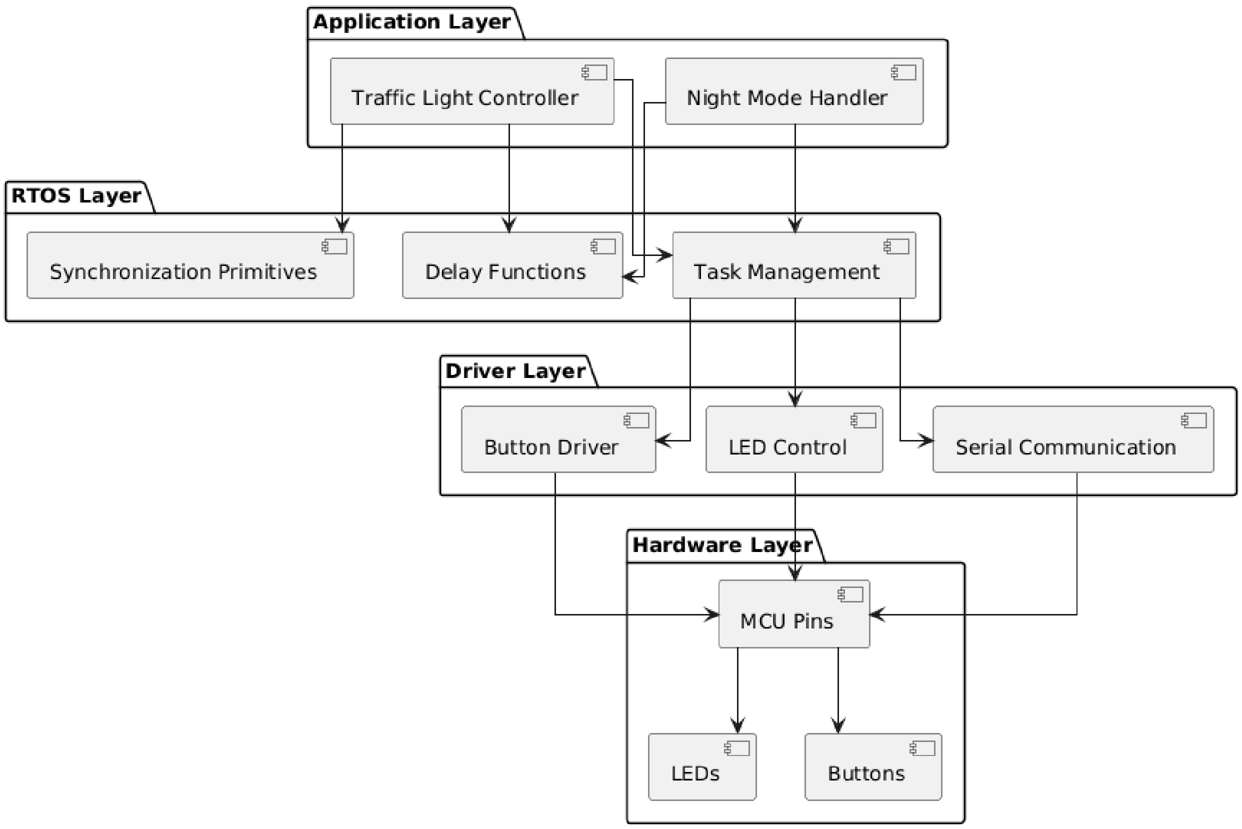


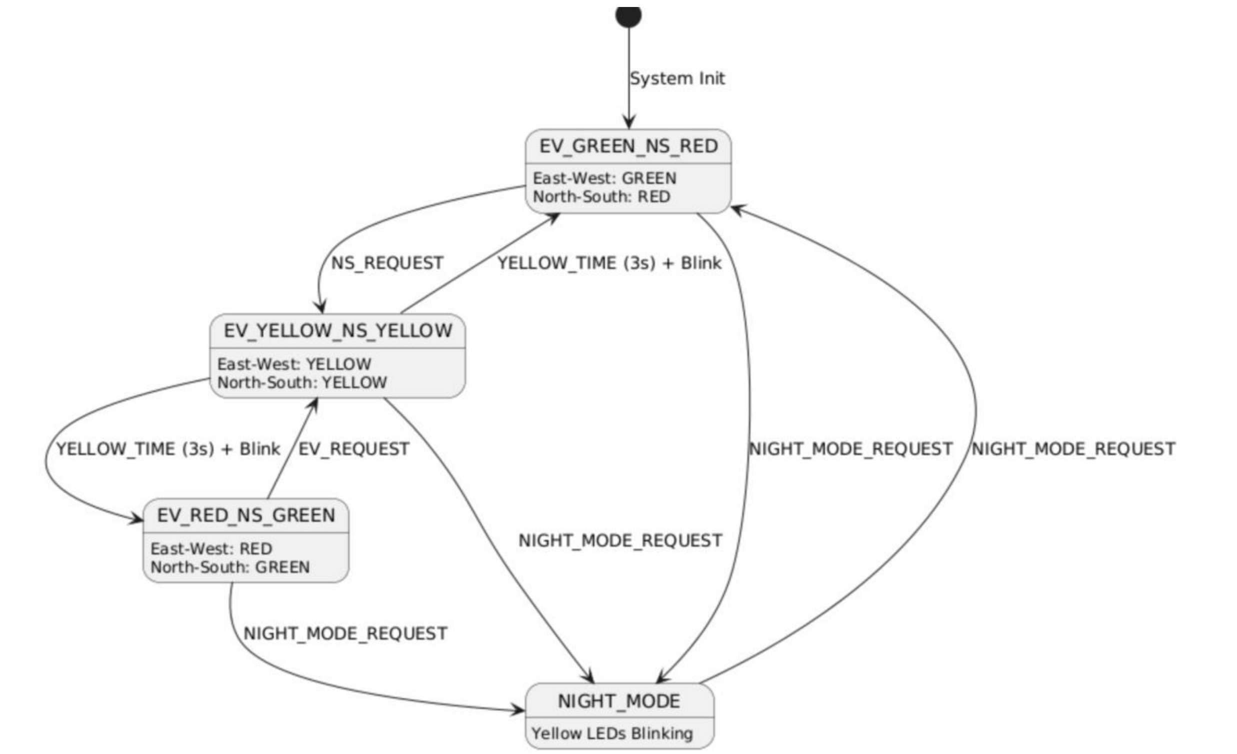
Figure 3.2 Software Layered Architecture

The software architecture consists of four distinct layers:

1. **Application Layer**: Contains the high-level functionality including the Traffic Light Controller which implements the FSM and the Night Mode Handler for the special operating mode.
2. **RTOS Layer**: Provides real-time operating system functionality including task management, delay functions for timing, and synchronization primitives for coordinating different parts of the system.
3. **Driver Layer**: Abstracts the hardware-specific operations into software interfaces for buttons, LEDs, and serial communication.
4. **Hardware Layer**: Represents the physical components including the MCU pins and the connected devices (LEDs and buttons).

3.2 Flowcharts

FSM


System Init
EV_GREEN_NS_RED
East-West: GREEN
North-South: RED
NS_REQUEST
YELLOW_TIME (3s) + Blink
EV_YELLOW_NS_YELLOW
East-West: YELLOW
North-South: YELLOW
YELLOW_TIME (3s) + Blink
EV_REQUEST
NIGHT_MODE_REQUEST
NIGHT_MODE_REQUEST
EV_RED_NS_GREEN
East-West: RED
North-South: GREEN
NIGHT_MODE_REQUEST
NIGHT_MODE_REQUEST
NIGHT_MODE
Yellow LEDs Blinking

State table

State	Description	LED States	Possible Transitions
EV_GREEN_NS_RED	East-West green, North-South red	EV_GREEN: ON, NS_RED: ON	NS_REQUEST → EV_YELLOW_NS_YELLOW NIGHT_MODE_REQUEST → NIGHT_MODE
EV_YELLOW_NS_YELLOW	Both directions yellow (transition)	EV_YELLOW: ON, NS_YELLOW: ON	YELLOW_TIME (3s) + Blink → EV_RED_NS_GREEN (from NS_REQUEST) YELLOW_TIME (3s) + Blink → EV_GREEN_NS_RED (from EV_REQUEST) NIGHT_MODE_REQUEST → NIGHT_MODE
EV_RED_NS_GREEN	East-West red, North-South green	EV_RED: ON, NS_GREEN: ON	EV_REQUEST → EV_YELLOW_NS_YELLOW NIGHT_MODE_REQUEST → NIGHT_MODE
NIGHT_MODE	Yellow LEDs blinking (night operation)	EV_YELLOW: BLINK, NS_YELLOW: BLINK	NIGHT_MODE_REQUEST → EV_GREEN_NS_RED

The following flowchart (Figure 3.3) illustrates the operation of the traffic light state machine:

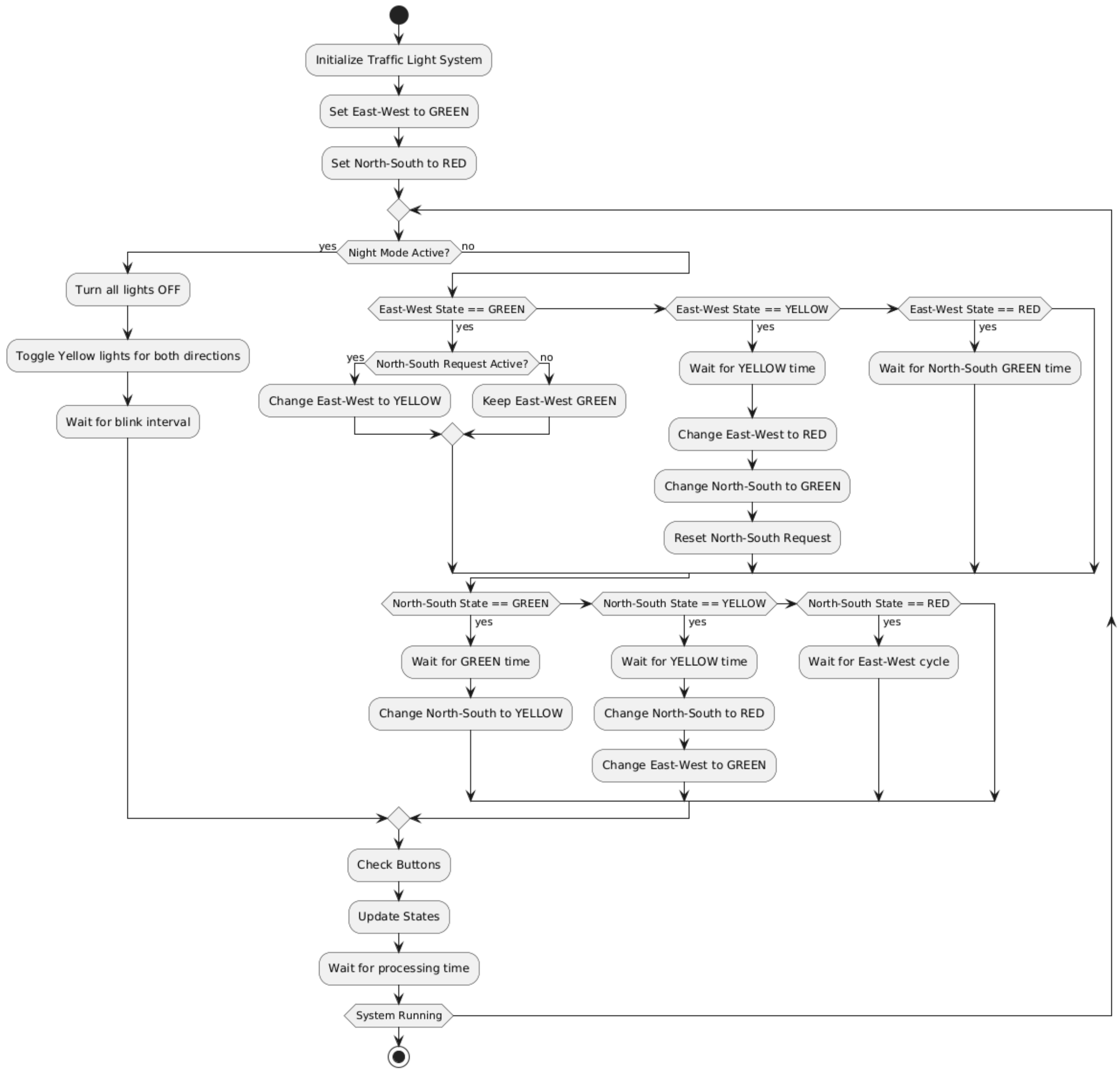


Figure 3.3 Traffic Light State Machine Flowchart

The flowchart above shows the main control flow of the traffic light system. The system begins by initializing the traffic lights with East-West direction set to GREEN and North-South direction set to RED. It then enters a continuous loop that:

1. Checks if night mode is active
2. If night mode is active, it toggles the yellow lights for both directions
3. If night mode is not active, it processes the normal traffic light sequence based on the current states and any active requests
4. Regularly checks button states for new requests or mode changes
5. Updates the physical lights to match the current states

The next flowchart (Figure 3.4) focuses on the button handling logic:

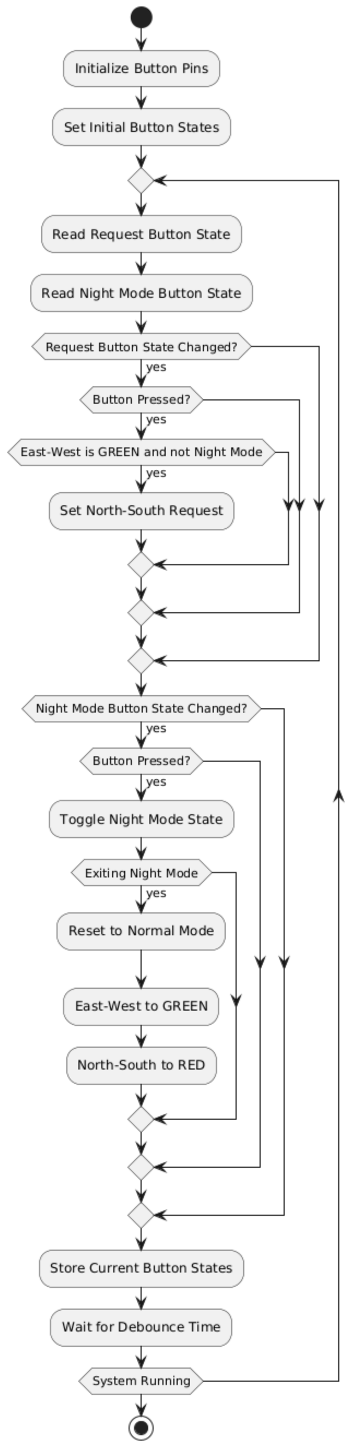


Figure 3.4 Button Handling Flowchart

This flowchart details how the system processes button inputs:

1. It continuously monitors the states of both the request button and the night mode button
2. When the request button is pressed (and changes from HIGH to LOW), it sets a request flag for the North-South direction if the current state allows it

3. When the night mode button is pressed, it toggles the night mode state
4. If exiting night mode, it resets the system to the default state with East-West GREEN and North-South RED
5. It includes debounce handling to prevent false readings from button noise

3.3 Schematic view of the circuit

For the electrical scheme, I used the following components:

- Arduino Uno MCU
- 6 LEDs (2 green, 2 yellow, 2 red)
- 2 pushbuttons
- Resistors (220Ω for LEDs, 10kΩ for buttons)

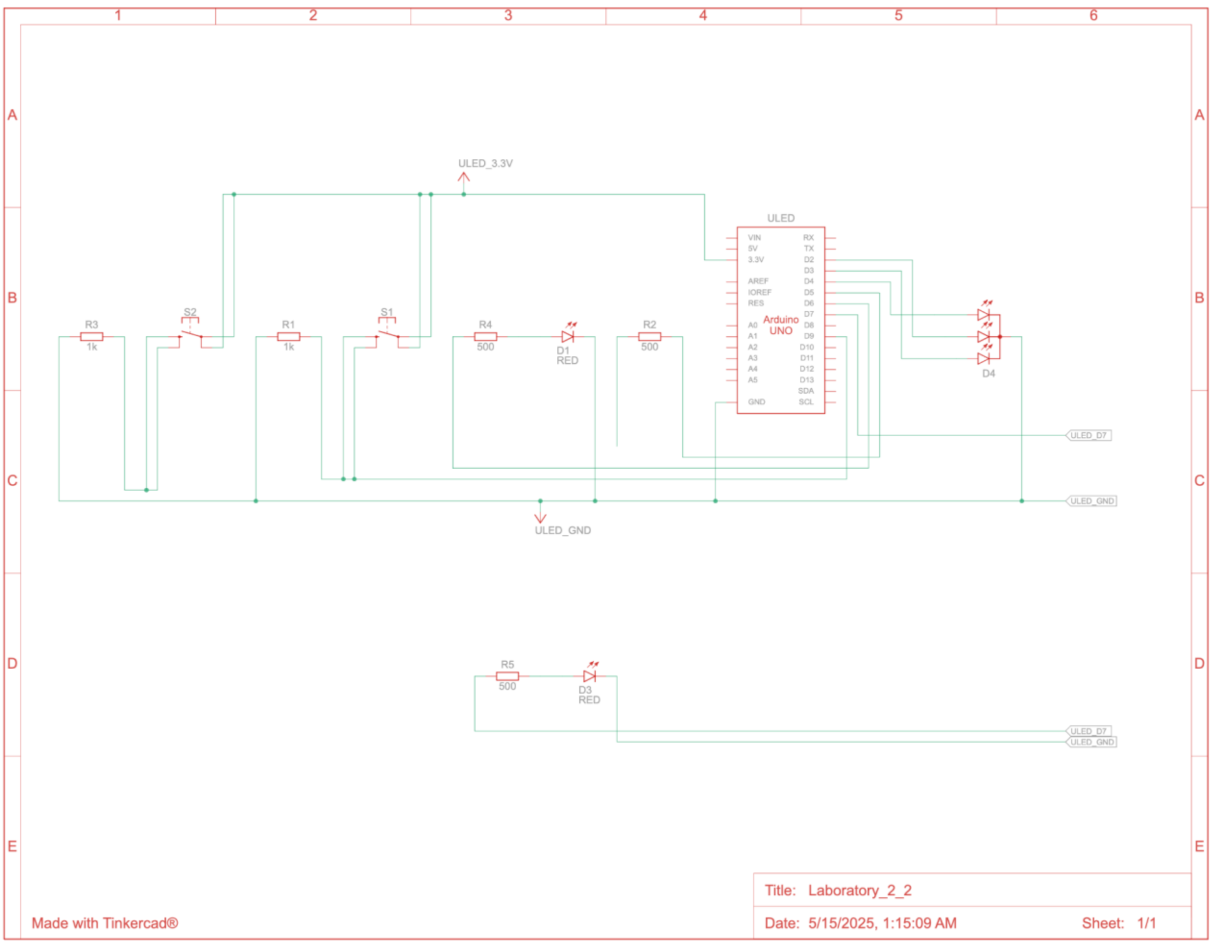


Figure 3.5 Electrical Diagram

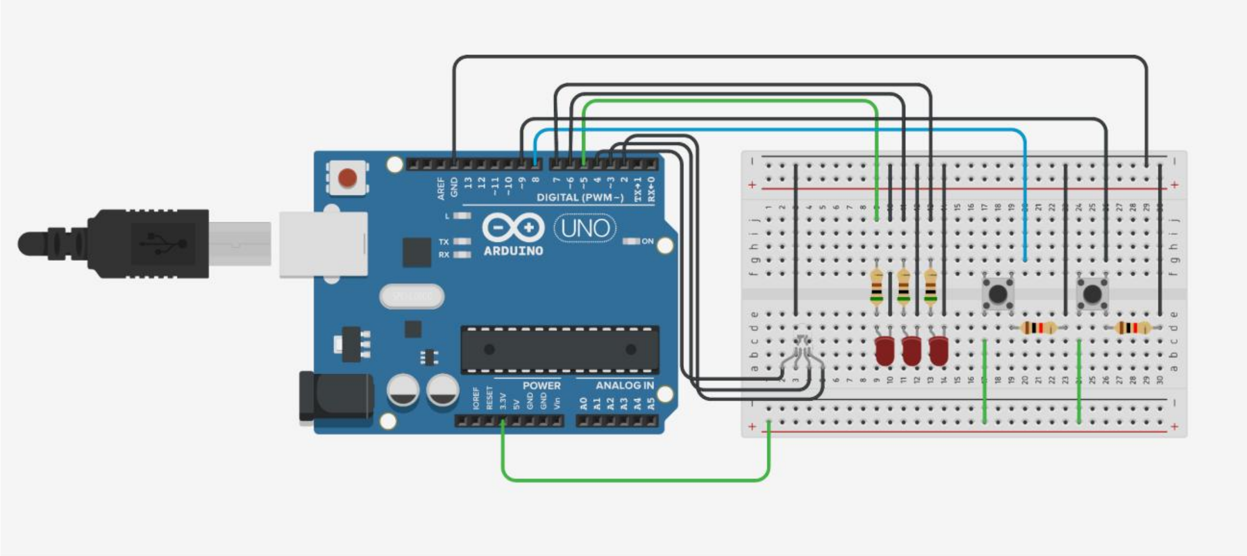


Figure 3.6 Circuit Diagram

The schematic shows the connections between the Arduino Uno and the external components:

1. Six LEDs represent the traffic lights for both directions, with each LED connected to a digital output pin through a 220Ω current-limiting resistor.
2. Two pushbuttons are connected to digital input pins (2 and 3) with 10kΩ pull-up resistors to provide clean input signals.
3. The Arduino Uno provides the processing power and interfaces with all components.

3.4 Structure of the project

The project follows a modular organization to ensure maintainability and separation of concerns:

```
laboratory_6_2/
│
├── include/
│   ├── traffic_light.h        # Traffic light control interface
│   └── uart_helpers.h         # UART communication helpers
│
├── src/
│   ├── main.cpp               # Application entry point
│   ├── traffic_light.cpp      # Traffic light FSM implementation
│   └── uart_helpers.cpp       # UART stdio redirection
```

```
│
├── platformio.ini              # PlatformIO configuration
└── README.md                   # Project documentation
```

This structure separates the interface definitions (in the include directory) from their implementations (in the src directory), following good practice for C++ projects. The main.cpp file contains the entry point of the application, while traffic_light.cpp implements the finite state machine logic for the traffic light control.

4. RESULTS

The smart traffic light control system was successfully implemented and tested according to the requirements. The system effectively controls the traffic lights for both East-West and North-South directions based on requests, with FreeRTOS handling the timing and synchronization.

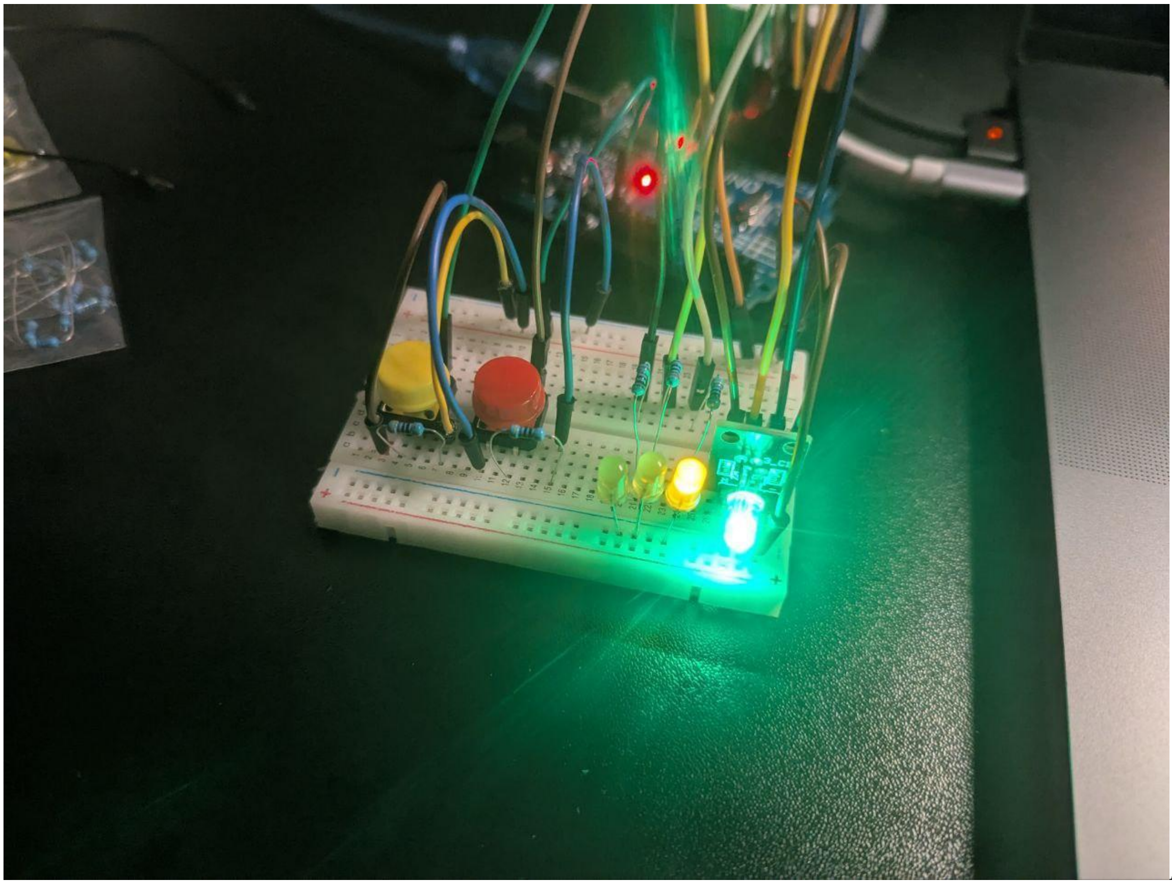


Figure 4.1 Initial state with East-West green and North-South red

When the system starts, it initializes with the East-West direction having a green light and the North-South direction having a red light. The serial monitor displays the current states, as shown in Figure 4.1.

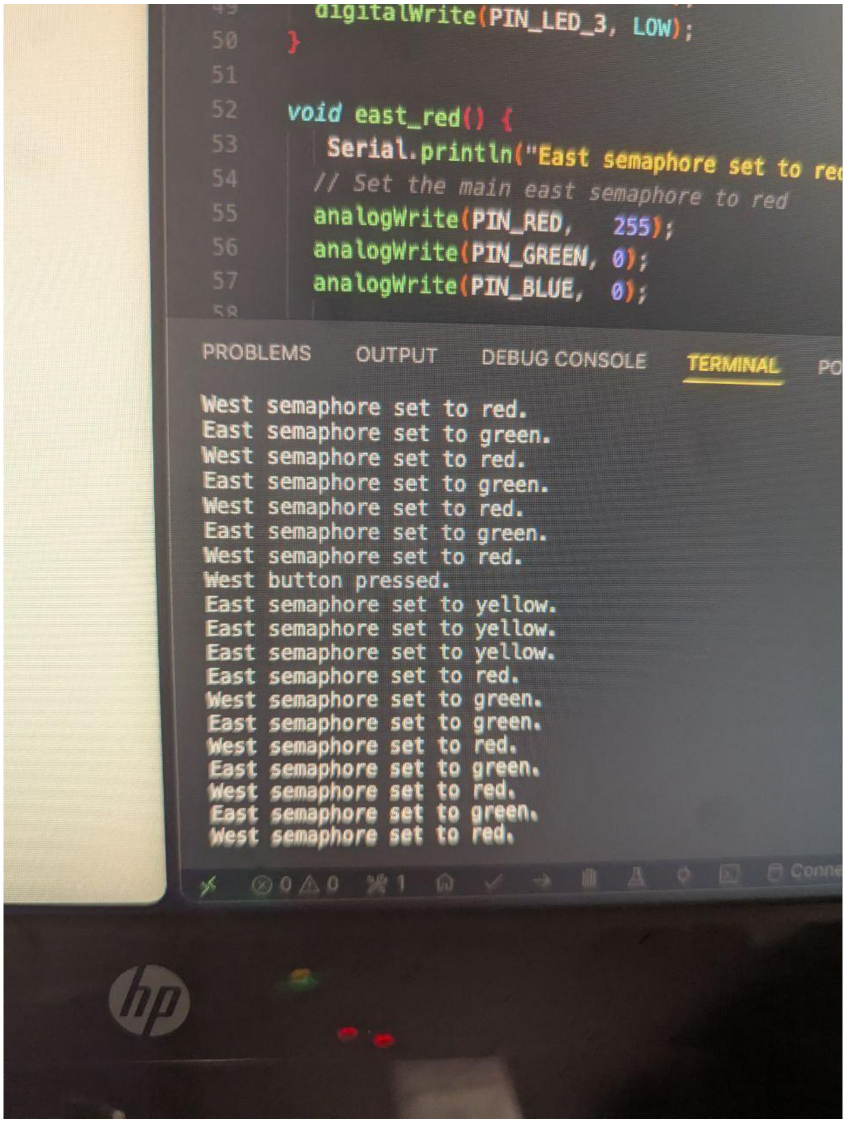


Figure 4.2 System processing a North-South request

Figure 4.2 shows the system's response when a request for the North-South direction is received. The East-West direction transitions from green to yellow as part of the normal traffic light sequence.

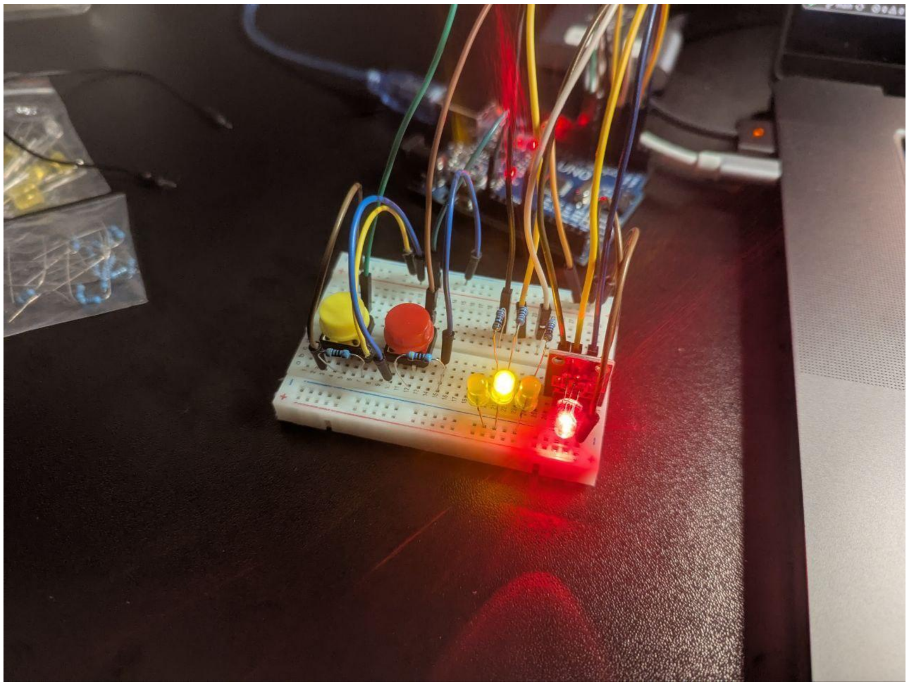


Figure 4.3 North-South direction now has green light

After the appropriate delays, the East-West direction changes to red and the North-South direction changes to green, as shown in Figure 4.3. This demonstrates the proper sequence of traffic light transitions.

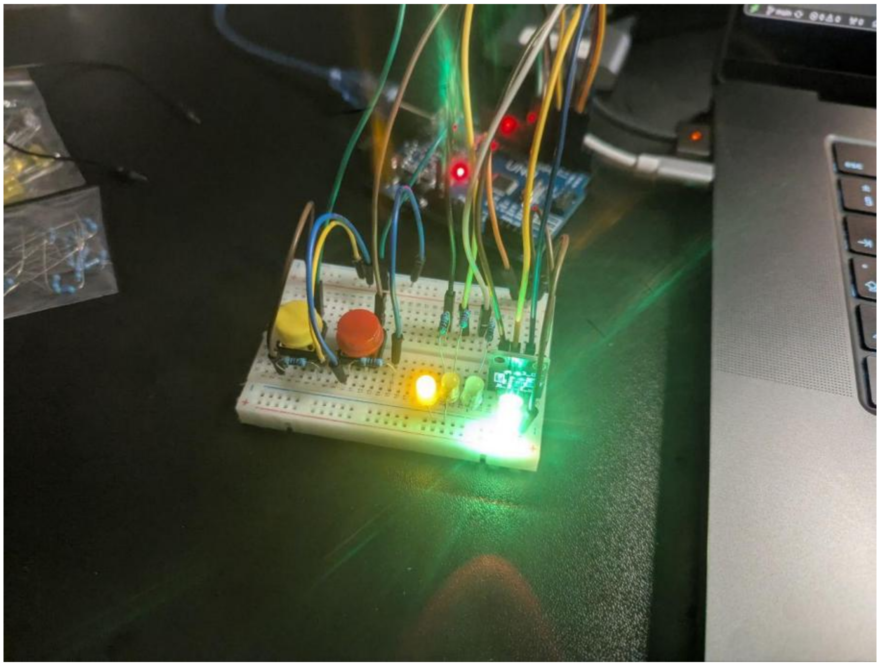


Figure 4.4 Night mode with flashing yellow lights

Figure 4.4 illustrates the night mode operation, where both directions have flashing yellow lights. This special mode is activated by pressing the night mode button and provides a different behavior for low-traffic periods.

The system successfully meets all the requirements specified in the problem definition:

1. It implements a modular application for controlling traffic lights at an intersection with two directions
2. It uses a finite state machine approach for state management
3. It utilizes FreeRTOS for timing and task management
4. It responds to requests through button inputs
5. It displays the current state through the serial interface
6. It implements an additional night mode behavior

5. CONCLUSIONS

In this laboratory work, I successfully designed and implemented a smart traffic light control system using finite state machines and the FreeRTOS real-time operating system. The system demonstrates the application of embedded systems concepts in a realistic traffic control scenario.

The key accomplishments include:

1. Implementation of a state machine for controlling traffic light sequences according to standard traffic rules
2. Utilization of FreeRTOS for managing timing, delays, and task synchronization
3. Development of a modular system with well-defined interfaces for easy maintenance and extension
4. Implementation of an intelligent request-based system that responds to traffic demands
5. Addition of a night mode feature as an enhanced behavior

Throughout the development process, I gained valuable experience in:

- Designing finite state machines for real-world applications
- Using FreeRTOS mechanisms for task coordination
- Implementing event-driven systems with external inputs
- Optimizing code for memory-constrained environments

The challenges faced during implementation included:

- Managing memory constraints on the Arduino platform
- Ensuring proper synchronization between state transitions
- Implementing debounce handling for physical buttons
- Coordinating the timing between different traffic light states

The system could be further improved by:

- Adding more sophisticated traffic sensing capabilities
- Implementing adaptive timing based on traffic flow
- Adding pedestrian crossing functionality
- Incorporating wireless communication for coordinated traffic management across multiple intersections

This laboratory work provided practical experience in applying theoretical concepts of finite state machines and real-time operating systems to solve a common real-world problem, demonstrating how embedded systems can be used to create intelligent infrastructure solutions.

6. BIBLIOGRAPHY

[1] FreeRTOS - Market leading RTOS. URL: https://www.freertos.org/

[2] PlatformIO: Professional collaborative platform for embedded development. URL: https://docs.platformio.org/

[3] Arduino - Reference. URL: https://www.arduino.cc/reference/en/

[4] Finite State Machines for Embedded Systems. URL: https://state-machine.com/fsm

[5] feilipu/FreeRTOS Arduino Library. URL: https://github.com/feilipu/Arduino_FreeRTOS_Library

[6] Traffic Light Control Systems: A Review. URL: https://www.researchgate.net/publication/308794960_Traffic_Light_Control_Systems_A_Review

7. APPENDIX

traffic_light.h

```
#ifndef TRAFFIC_LIGHT_H

#define TRAFFIC_LIGHT_H


#include <Arduino.h>

#include <Arduino_FreeRTOS.h>


// Traffic light states

typedef enum {

    TRAFFIC_GREEN,

    TRAFFIC_YELLOW,

    TRAFFIC_RED,

    TRAFFIC_NIGHT_MODE
```

```c
} TrafficLightState;


// Traffic light directions

typedef enum {

    DIRECTION_EAST_WEST,

    DIRECTION_NORTH_SOUTH

} TrafficDirection;


// State timing configuration (in milliseconds)

#define GREEN_TIME 5000

#define YELLOW_TIME 2000

#define RED_TIME 7000  // Equal to GREEN_TIME + YELLOW_TIME of other direction

#define NIGHT_MODE_BLINK_INTERVAL 500


// Button pins

#define REQUEST_BUTTON_PIN 2

#define NIGHT_MODE_BUTTON_PIN 3


// Initialize traffic light system

void initTrafficLights();


// Get state name as string for display

const char* getStateName(TrafficLightState state);


// Get direction name as string for display

const char* getDirectionName(TrafficDirection direction);
```

```
// Create tasks for the traffic light system
void createTrafficLightTasks();

// External variable for night mode state
extern volatile bool nightModeActive;

#endif // TRAFFIC_LIGHT_H
```

uart_helpers.h

```
#ifndef UART_HELPERS_H
#define UART_HELPERS_H

#include <Arduino.h>
#include <stdio.h>

// Setup printf to use Arduino Serial
#ifndef PRINTF_BUF_SIZE
#define PRINTF_BUF_SIZE 64
#endif

// Function to redirect printf to Serial
int uart_putchar(char c, FILE *stream);

// Function to redirect stdin to Serial
int uart_getchar(FILE *stream);
```

```
// Initialize stdio for uart
void setup_uart_stdio();

extern FILE uart_output;
extern FILE uart_input;

#endif // UART_HELPERS_H
```

main.cpp

```
#include <Arduino.h>
#include <Arduino_FreeRTOS.h>
#include <stdio.h>
#include "uart_helpers.h"
#include "traffic_light.h"

void setup() {
  // Initialize serial communication
  Serial.begin(9600);

  // Initialize stdio for printf functionality
  setup_uart_stdio();

  // Print welcome message
  printf("\n\n==============================\n");
  printf("Smart Traffic Light Controller\n");
  printf("==============================\n\n");
```

```
  // Initialize traffic light system
  initTrafficLights();

  // Create traffic light tasks
  createTrafficLightTasks();

  // No need for a standard loop function as FreeRTOS will manage the
tasks
  printf("System startup complete. FreeRTOS scheduler starting...\n\n");
}

void loop() {
  // Empty - all work is done in FreeRTOS tasks
  // This function is required by Arduino but not used
}
```

traffic_light.cpp

```
#include "traffic_light.h"
#include <stdio.h>
#include <task.h>

// Pin definitions for traffic lights
// East-West traffic light
#define EW_GREEN_PIN 5
#define EW_YELLOW_PIN 6
#define EW_RED_PIN 7
```

```
// North-South traffic light

#define NS_GREEN_PIN 8

#define NS_YELLOW_PIN 9

#define NS_RED_PIN 10


// Task handles

TaskHandle_t xTrafficControlTaskHandle = NULL;


// Global state for night mode

volatile bool nightModeActive = false;

volatile bool northSouthRequest = false;


// Current state of both traffic lights

TrafficLightState eastWestState = TRAFFIC_GREEN;

TrafficLightState northSouthState = TRAFFIC_RED;


// Task function prototypes

void vTrafficControlTask(void *pvParameters);


// Helper functions

void setTrafficLightState(TrafficDirection direction, TrafficLightState
state);

void turnOffAllLights(TrafficDirection direction);

void handleNightMode();


// Initialize traffic light system

void initTrafficLights() {
```

```
    // Setup pins for East-West traffic light
    pinMode(EW_GREEN_PIN, OUTPUT);
    pinMode(EW_YELLOW_PIN, OUTPUT);
    pinMode(EW_RED_PIN, OUTPUT);

    // Setup pins for North-South traffic light
    pinMode(NS_GREEN_PIN, OUTPUT);
    pinMode(NS_YELLOW_PIN, OUTPUT);
    pinMode(NS_RED_PIN, OUTPUT);

    // Setup button pins
    pinMode(REQUEST_BUTTON_PIN, INPUT_PULLUP);
    pinMode(NIGHT_MODE_BUTTON_PIN, INPUT_PULLUP);

    // Initial state: East-West is green, North-South is red
    setTrafficLightState(DIRECTION_EAST_WEST, TRAFFIC_GREEN);
    setTrafficLightState(DIRECTION_NORTH_SOUTH, TRAFFIC_RED);

    printf("System initialized\n");
}

// Create and start traffic light tasks
void createTrafficLightTasks() {
    // Create a single task to control everything
    xTaskCreate(
        vTrafficControlTask,
        "TrafficControl",
```

```
        96,  // Reduced stack size
        NULL,
        1,
        &xTrafficControlTaskHandle
    );

    printf("Task created\n");
}

// Combined traffic control task
void vTrafficControlTask(void *pvParameters) {
    uint8_t lastRequestButtonState = HIGH;
    uint8_t lastNightModeButtonState = HIGH;
    TickType_t lastStateChangeTime = 0;
    TickType_t lastButtonCheckTime = 0;

    for (;;) {
        TickType_t currentTime = xTaskGetTickCount();

        // Check buttons every 50ms
        if (currentTime - lastButtonCheckTime >= pdMS_TO_TICKS(50)) {
            lastButtonCheckTime = currentTime;

            // Read button states
            uint8_t requestButtonState = digitalRead(REQUEST_BUTTON_PIN);
            uint8_t nightModeButtonState =
digitalRead(NIGHT_MODE_BUTTON_PIN);
```

```
            // Handle request button
            if (requestButtonState != lastRequestButtonState) {
                if (requestButtonState == LOW && !northSouthRequest &&
                    eastWestState == TRAFFIC_GREEN && !nightModeActive) {
                    northSouthRequest = true;
                }
                lastRequestButtonState = requestButtonState;
            }

            // Handle night mode button
            if (nightModeButtonState != lastNightModeButtonState) {
                if (nightModeButtonState == LOW) {
                    nightModeActive = !nightModeActive;

                    if (!nightModeActive) {
                        // Return to normal mode
                        eastWestState = TRAFFIC_GREEN;
                        northSouthState = TRAFFIC_RED;
                        setTrafficLightState(DIRECTION_EAST_WEST,
TRAFFIC_GREEN);
                        setTrafficLightState(DIRECTION_NORTH_SOUTH,
TRAFFIC_RED);
                        lastStateChangeTime = currentTime;
                    }
                }
                lastNightModeButtonState = nightModeButtonState;
            }
        }
```

```
        // Night mode handling
        if (nightModeActive) {
            static bool yellowOn = false;
            static TickType_t lastBlinkTime = 0;

            if (currentTime - lastBlinkTime >=
pdMS_TO_TICKS(NIGHT_MODE_BLINK_INTERVAL)) {
                lastBlinkTime = currentTime;
                yellowOn = !yellowOn;

                // East-West yellow
                digitalWrite(EW_GREEN_PIN, LOW);
                digitalWrite(EW_YELLOW_PIN, yellowOn);
                digitalWrite(EW_RED_PIN, LOW);

                // North-South yellow
                digitalWrite(NS_GREEN_PIN, LOW);
                digitalWrite(NS_YELLOW_PIN, yellowOn);
                digitalWrite(NS_RED_PIN, LOW);
            }

            vTaskDelay(pdMS_TO_TICKS(20));
            continue;
        }

        // Normal traffic light control
        switch (eastWestState) {
```

```
            case TRAFFIC_GREEN:

                if (northSouthRequest && (currentTime -
lastStateChangeTime >= pdMS_TO_TICKS(500))) {

                    eastWestState = TRAFFIC_YELLOW;

                    setTrafficLightState(DIRECTION_EAST_WEST,
TRAFFIC_YELLOW);

                    lastStateChangeTime = currentTime;

                }

                break;



            case TRAFFIC_YELLOW:

                if (currentTime - lastStateChangeTime >=
pdMS_TO_TICKS(YELLOW_TIME)) {

                    eastWestState = TRAFFIC_RED;

                    northSouthState = TRAFFIC_GREEN;

                    setTrafficLightState(DIRECTION_EAST_WEST,
TRAFFIC_RED);

                    setTrafficLightState(DIRECTION_NORTH_SOUTH,
TRAFFIC_GREEN);

                    northSouthRequest = false;

                    lastStateChangeTime = currentTime;

                }

                break;



            case TRAFFIC_RED:

                // Controlled by North-South state

                break;

        }



        switch (northSouthState) {
```

```
            case TRAFFIC_GREEN:
                if (currentTime - lastStateChangeTime >=
pdMS_TO_TICKS(GREEN_TIME)) {
                    northSouthState = TRAFFIC_YELLOW;
                    setTrafficLightState(DIRECTION_NORTH_SOUTH,
TRAFFIC_YELLOW);
                    lastStateChangeTime = currentTime;
                }
                break;

            case TRAFFIC_YELLOW:
                if (currentTime - lastStateChangeTime >=
pdMS_TO_TICKS(YELLOW_TIME)) {
                    northSouthState = TRAFFIC_RED;
                    eastWestState = TRAFFIC_GREEN;
                    setTrafficLightState(DIRECTION_NORTH_SOUTH,
TRAFFIC_RED);
                    setTrafficLightState(DIRECTION_EAST_WEST,
TRAFFIC_GREEN);
                    lastStateChangeTime = currentTime;
                }
                break;

            case TRAFFIC_RED:
                // Controlled by East-West state
                break;
        }

        vTaskDelay(pdMS_TO_TICKS(20));
    }
```

```
}


// Set traffic light to specific state

void setTrafficLightState(TrafficDirection direction, TrafficLightState
state) {

    // Get the pins for the specified direction

    uint8_t greenPin, yellowPin, redPin;


    switch (direction) {

        case DIRECTION_EAST_WEST:

            greenPin = EW_GREEN_PIN;

            yellowPin = EW_YELLOW_PIN;

            redPin = EW_RED_PIN;

            break;


        case DIRECTION_NORTH_SOUTH:

            greenPin = NS_GREEN_PIN;

            yellowPin = NS_YELLOW_PIN;

            redPin = NS_RED_PIN;

            break;


        default:

            return; // Invalid direction

    }


    // Turn off all lights first

    digitalWrite(greenPin, LOW);

    digitalWrite(yellowPin, LOW);
```

```
    digitalWrite(redPin, LOW);

    // Set the light according to the state
    switch (state) {
        case TRAFFIC_GREEN:
            digitalWrite(greenPin, HIGH);
            break;

        case TRAFFIC_YELLOW:
            digitalWrite(yellowPin, HIGH);
            break;

        case TRAFFIC_RED:
            digitalWrite(redPin, HIGH);
            break;

        case TRAFFIC_NIGHT_MODE:
            // Night mode is handled separately
            break;
    }

    // Print the state change
    printf("%s: %s\n", getDirectionName(direction), getStateName(state));
}

// Get state name as string for display
const char* getStateName(TrafficLightState state) {
```

```
    switch (state) {
        case TRAFFIC_GREEN:
            return "GREEN";
        case TRAFFIC_YELLOW:
            return "YELLOW";
        case TRAFFIC_RED:
            return "RED";
        case TRAFFIC_NIGHT_MODE:
            return "NIGHT";
        default:
            return "UNK";
    }
}

// Get direction name as string for display
const char* getDirectionName(TrafficDirection direction) {
    switch (direction) {
        case DIRECTION_EAST_WEST:
            return "EW";
        case DIRECTION_NORTH_SOUTH:
            return "NS";
        default:
            return "UNK";
    }
}
```

uart_helpers.cpp

```
#include "uart_helpers.h"

// Setup FILE streams for stdin and stdout
FILE uart_output;
FILE uart_input;

// Function to redirect printf to Serial
int uart_putchar(char c, FILE *stream) {
  Serial.write(c);
  return 0;
}

// Function to redirect stdin to Serial
int uart_getchar(FILE *stream) {
  while (!Serial.available());
  return Serial.read();
}

void setup_uart_stdio() {
  // Initialize serial communication
  Serial.begin(9600);
  while (!Serial); // Wait for serial port to connect

  // Initialize stdio
  fdev_setup_stream(&uart_output, uart_putchar, NULL, _FDEV_SETUP_WRITE);
  fdev_setup_stream(&uart_input, NULL, uart_getchar, _FDEV_SETUP_READ);
  stdout = &uart_output;
```

```
  stdin = &uart_input;
}
```