VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY

HO CHI MINH UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

MICROPROCESSORS - MICROCONTROLLERS

Assignment

STM32 Multi-Mode Traffic Lights

Instructor: Dr. Le Trong Nhan

STT	Full name	Students ID	Class
1	Nguyễn Hưng Thịnh	2313287	L02
2	Nguyễn Thái Hoàng	2311062	L01
5	Dương Khôi Nguyên	2312333	L02

Ho Chi Minh City, 12/2025

Acknowledgement

We would like to express our sincere gratitude to our instructor, **Dr. Le Trong Nhan**, for his dedicated guidance and for providing the fundamental knowledge of Microprocessors and Microcontrollers that made this project possible. His lectures and course materials were instrumental in helping us understand the complex concepts of STM32 programming and controlling systems.

We also wish to thank the Faculty of Computer Science and Engineering at Ho Chi Minh City University of Technology (HCMUT) for providing an excellent academic environment that encourages research and practical application. And lastly, we appreciate the constructive feedback and support from our peers during the development of this project.

Table of Contents

1 Design objectives and Requirements

Traffic congestion is a critical issue in many areas. Effective traffic light control is essential for managing vehicle flow, reducing bottlenecks, and enhancing road safety. This project focuses on simulating a flexible four-way traffic light control system that allows for manual intervention and clear status display via an LCD screen.

The main goal is to design and simulate a microcontroller-based traffic light system. The system must meet the following technical requirements:

- **Microcontroller:** Utilize a standard 8 - bit microcontroller.
- **Display:** Two sets of traffic lights for two directions. A 16x2 LCD module must display the current mode and the countdown timer.
- **Control Interface:** Four buttons are used to control the system with two modes, each has a distinct set of functionalities. In `AUTO` mode:
  - `MODE` Button: Cycle through operating modes (Automatic ↔ Manual).
  - `CONFIG` Button: Entering configuration mode.
  - `UP` Buttons: Increase the Red/Green/Yellow duration.
  - `DOWN` Buttons: Decrease the Red/Green/Yellow duration.

  In `MANUAL` mode:
  - `MODE` Button: Cycle through operating modes (Automatic ↔ Manual).
  - `SWITCH` Button: Changing light signals on each direction.
  - `FLASH` Buttons: Flashing Yellow light (Warning signal).
  - `RED` Buttons: Red light on both directions (Emergency stop).
- **Safety Logic:** Adherence to traffic regulations must be ensured by satisfying the time constraint:

$$T_{Red} = T_{Green} + T_{Yellow} \tag{1}$$

  The system must automatically calculate the Red light duration based on the user-set Green and Yellow times.

2 Summary

This project involved the development of a comprehensive Python-based toolset for the simulation and verification of Petri nets. Through the completion of five distinct implementation tasks, the following milestones were achieved:

- **Parsing and Robustness:** Implementation of a fault-tolerant PNML parser and matrix processing unit capable of handling diverse input constraints.
- **Reachability Analysis:** A dual-approach implementation involving Explicit (BFS/DFS) and Symbolic (BDD) algorithms. The results highlighted a clear dichotomy: explicit methods excel in processing speed for small nets, whereas BDDs are required for scalability in high-complexity environments.
- **Deadlock and Optimization:** The deployment of a hybrid verification strategy that successfully identifies deadlocks and optimal firing sequences while maintaining system stability.

Overall, the application serves as a practical demonstration of formal methods in software engineering. It reinforces the necessity of adapting algorithmic strategies—specifically choosing between explicit state enumeration and symbolic representation—based on the scale and complexity of the concurrent system being modeled.

3 THEORETICAL BACKGROUND

4 The Microcontroller Unit (MCU)

The project utilizes the [Specify MCU, e.g., AT89S52/ATmega32]. This section details its architecture, memory organization, I/O ports, Timers, and Interrupt structure.

5 Interfacing with LCD 16x2

The LCD 16x2 module based on the HD44780 controller is used for information display.

- **Interface Mode:** 4-bit mode is selected to minimize I/O pin usage.
- **Control Pins:** RS (Register Select), RW (Read/Write), and E (Enable) are utilized to send commands and data.

6 Input/Output Peripherals

6.1 LED Indicators

The traffic lights are implemented using high-intensity LEDs. The current limiting resistor (R) is calculated by:

$$R = \frac{V_{CC} - V_{LED}}{I_{LED}} \tag{2}$$

With $V_{CC} = 5V$, $V_{LED} \approx 2V$, and $I_{LED} = 15mA$, a standard $R = 200\Omega$ or 330Ω resistor is selected.

6.2 Pushbuttons and Debouncing

The system uses four pull-up configured pushbuttons. Software debouncing (typically 20ms delay) is implemented to ensure reliable signal readings from the mechanical switches.

7 HARDWARE AND SYSTEM DESIGN

8 System Block Diagram

The system is logically divided into three main blocks as shown below:

Figure 1: System Block Diagram

9 Pin Allocation and Interfacing

- **Port 1 (P1.0 - P1.5):** Outputs for 6 Traffic LEDs (Direction A and B).
- **Port 2 (P2.4 - P2.7 + Control Pins):** 4-bit Data Interface for LCD.

- **Port 3 (P3.0 - P3.3):** Inputs for the 4 Pushbuttons (Input with internal pull-ups enabled).
- **Clock Circuitry:** Crystal Oscillator (11.0592MHz or 12MHz) for MCU timing.

10 Circuit Schematic

The detailed schematic (simulated in Proteus) shows the connections. (A driving circuit, such as a Transistor array like ULN2003 or simple NPN transistors, should be used if the LED current exceeds the MCU's maximum sink/source current per pin.)

Figure 2: Circuit Schematic of the Traffic Light Controller

11 ALGORITHM AND SOFTWARE IMPLEMENTATION

12 Finite State Machine (FSM) Logic

The traffic light sequence follows a standard four-state Finite State Machine model:

1. **State 1:** Direction A: GREEN (T_{Green}) - Direction B: RED (T_{Red}).
2. **State 2:** Direction A: YELLOW (T_{Yellow}) - Direction B: RED (continued).
3. **State 3:** Direction A: RED (T_{Red}) - Direction B: GREEN (T_{Green}).
4. **State 4:** Direction A: RED (continued) - Direction B: YELLOW (T_{Yellow}).

13 Flowchart

The main routine flowchart details the system's initialization, button scanning, mode checking, and function calls.

Figure 3: Main Program Flowchart

14 Timer Calculation

To achieve a precise 1-second delay using Timer 0 in 16-bit mode with an F_{osc} of 12MHz, the required Timer value is calculated as follows:

$$T_{delay} = (65536 - Value) \times \frac{12}{F_{osc}}$$

For a 1ms interrupt (to count up to 1s), $Value = 65536 - \frac{1ms \times 12MHz}{12} = 64536 = 0xFC18$. The final 1-second counter is then built by accumulating 1000 such 1ms interrupts.

15 Software Code Snippet (C Language)

The system is programmed in C using the Keil C IDE.

```c
#include <reg51.h>
// Pin definitions (example)
sbit BTN_MODE = P3^0;
// Global Variables
unsigned int T_Green = 25; // Default 25s
unsigned int T_Yellow = 3; // Default 3s
unsigned int T_Red;        // Calculated: T_Red = T_Green + T_Yellow

unsigned char Current_Mode = 0; // 0: Auto, 1: Manual/Warning, 2: Edit

// Function to enforce the safety logic
void calculate_red_time() {
    T_Red = T_Green + T_Yellow;
}

// Timer Interrupt Service Routine (1ms)
void Timer0_ISR() interrupt 1 {
    // Reload for 1ms delay (adjust based on Fosc)
    TH0 = 0xFC;
```

```
    TL0 = 0x18;

    // 1-second counter logic...
}

void main() {
    System_Init();
    calculate_red_time();
    while(1) {
        Scan_Buttons();
        switch(Current_Mode) {
            case 0: Run_Auto_Mode(); break;
            case 1: Run_Manual_Mode(); break;
            case 2: Run_Edit_Mode(); break;
        }
        Update_LCD_Display();
    }
}
```

1: Main Logic and Variable Definition

16 SIMULATION RESULTS AND EVALUATION

17 Simulation Environment

The hardware design and system functionality were verified using the Proteus 8 Professional simulation tool.

18 Achieved Results

18.1 Automatic Mode

The system successfully executed the standard traffic light sequence. When Direction A is Green (25s), Direction B is Red (28s). The LCD displays the countdown for both directions.

Figure 4: Simulation in Automatic Mode: Direction A is Green

18.2 Editing Mode

The Edit Mode allows the operator to change the T_{Green} and T_{Yellow} values using the UP/DOWN buttons. The system correctly recalculates T_{Red} automatically.

- When T_{Green} is changed from 25s to 30s, T_{Red} automatically updates from 28s to 33s.

18.3 Warning Mode

Activating the Warning mode causes both Yellow lights to flash synchronously at 1Hz, and the LCD displays "WARNING MODE".

19 Evaluation

Strengths:

- The critical safety logic ($T_{Red} = T_{Green} + T_{Yellow}$) is robustly implemented.
- The LCD provides an intuitive and necessary human-machine interface (HMI) for status monitoring.
- The FSM structure ensures clear separation and control of traffic phases.

Limitations:

- The system is based on fixed timing and does not adapt to traffic density.
- Lack of a real-time clock (RTC) for time-of-day programming.

CONCLUSION AND FUTURE WORK

Conclusion: The final project successfully met all design requirements for a flexible four-way traffic light controller. The group gained practical experience in applying

microcontroller programming techniques, including Timer/Interrupt handling, and interfacing with key peripherals like LCDs and pushbuttons.

Future Work: To enhance the system for real-world application, future developments could include:

1. Implementing sensor input (e.g., inductive loops or cameras) to dynamically adjust T_{Green} based on real-time traffic flow (Smart Traffic Light).

2. Integrating a communication module (e.g., UART or ESP8266 Wi-Fi) for remote monitoring and centralized control.

3. Upgrading the MCU to a 32-bit platform (e.g., STM32) for greater processing power and peripheral options.