Traffic Light with FreeRTOS

Name: Cheng Chen, Tran Tri Tin, Abbasi Afrooz

Date: 04/23/2025

CONTENTS

01

General Description

➢ Traffic light control necessary

Traffic light Background

- In reality, a traffic light system is not just a timer, but a "real-time" system that needs to respond dynamically according to different times and traffic.
- Using an embedded platform to simulate and implement traffic light control can help you understand the design concept and implementation mechanism of real-time control systems.
- Traffic signals are the core components of intelligent transportation systems, used to regulate traffic flow, ensure traffic order and pedestrian safety.

➢ FreeRTOS in embedded systems

FreeRTOS is a lightweight real-time operating system that is widely used in industrial control, Internet of Things, automotive electronics, and other fields.

It supports key features such as multi-task scheduling, priority management, semaphore, mutex, making the system structure clearer and more responsive.

FreeRTOS in embedded systems

This project implements the traffic light system through FreeRTOS, simulates the real logic of red, yellow and green lights, and demonstrates the design methods of multi-task real-time systems in practical applications,

This project will provide valuable experience for in-depth embedded system design (such as robot scheduling, smart city systems, etc.) in the future

02

System Architecture

Overview of System Architecture

STM32F407 DISC1 development board

Core chip: STM32F407VGT6 (ARM Cortex-M4 @ 168MHz)

Have rich peripheral interfaces (GPIO, USART, ADC, Timer, etc.)

Code download and debugging are achieved through USB connection to the computer

STM32CubeIDE + FreeRTOS

STM32CubeIDE: an integrated development environment based on Eclipse, supporting graphical configuration, code generation, and debugging

Integrate the STM32 HAL library and FreeRTOS kernel to quickly create multitasking systems

LED and 74HC595 Shift Register

74HC595: Serial to Parallel Shift Register for Controlling Multiple LEDs, with only 3 GPIO pins to control 8 outputs

Pin function:

SER (data input)

SRCLK (shift clock)

RCLK (latch clock)

System Functional Design

The switching logic of red, yellow and green lights

Green light → Yellow light → Red light → Green light Periodic switching;
the switching logic is controlled by the task cycle (vTaskDelay), keeping the light state synchronous and regular changes

LED traffic simulation (flowing light effect)

Simulate the vehicle through the green light area by controlling the shift through 74HC595
Under green light: LED chase effect (fast-paced flow)
In yellow light state: the left side flashes slowly
In red light state: All vehicles stop

Press a button to trigger an interrupt

The stm32 comes with buttons that can control the movement of traffic lights

03

Function design

➢ Task Management in FreeRTOS

TrafficLight.c

·Control the switching of red, green, and yellow light states
·Leverage vTaskDelay() for periodic light changes
·Use the mutex xMutexLight to ensure secure access to state variables
·Perform light status changes and notify relevant modules (e.g. traffic display)

TrafficDisplay.c

·Control the 74HC595 output LED flow light effect
·Select modes such as chase lights/flashing lights based on the current traffic light status
·It reflects the control effect of traffic signals on the flow of "vehicles".
·Use the encapsulation interface in the shift function ShiftRegister.c to control the light sequence

TrafficCreator.c ShiftRegister.c

Data Write:
Writes 8 bits of data to the shift register
Each bit is sent sequentially (from MSB to LSB), and the data and clock lines are controlled by HAL_GPIO_WritePin().
When the write is complete, the latching function is called and the output is output to the LED
ShiftRegister_ClearAll ()
Zero the output registers so that all LEDs are off

➢ FreeRTOS

Use vTaskDelay() to control task rhythm

The system uses vTaskDelay() to make each task run in a fixed cycle, such as:

The red light lasts for 5 seconds:
vTaskDelay(pdMS_TO_TICKS(5000));

Traffic flow light interval 100 milliseconds:
vTaskDelay(pdMS_TO_TICKS(100));

This allows precise time control without consuming CPU resources (non-blocking delay).

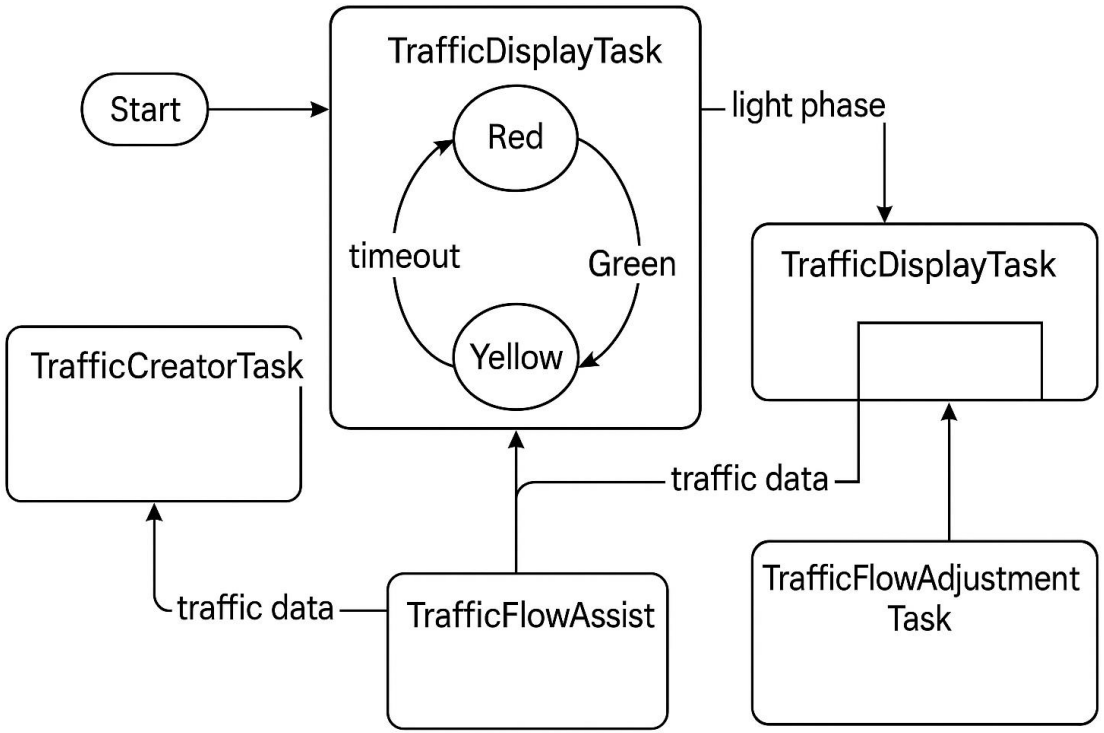
Tasks communicate with each other through shared variables and mutex locks

Multiple tasks share variables such as g_light_phase, g_car_value, g_flowrate. In order to avoid race conditions, the system uses the Semaphore mechanism of FreeRTOS to achieve mutually exclusive access:

```
xSemaphoreTake(xMutexLight, pdMS_TO_TICKS(10));
g_light_phase = 1;
xSemaphoreGive(xMutexLight);
```

This mechanism ensures the read and write security of variables in a concurrent environment.

➢ Performance and Cost Comparison with Other RTOS

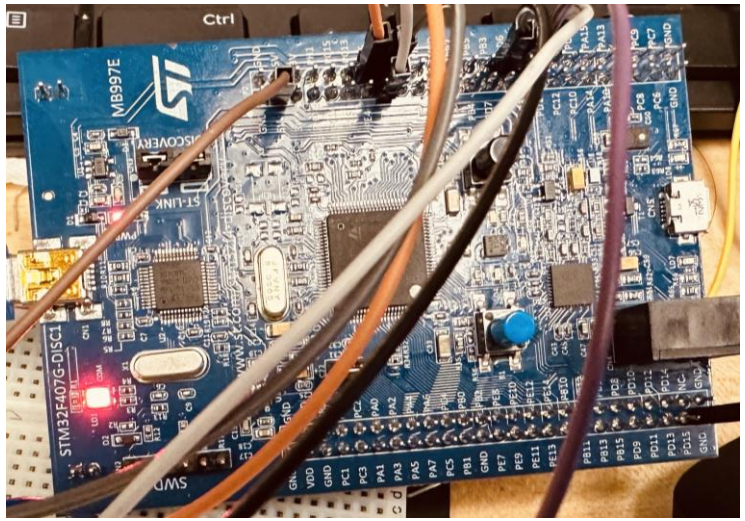


This system implements a multi-task traffic light control system based on STM32F407 and FreeRTOS. The core includes three threads: **TrafficLightTask** is responsible for controlling the switching of red, green and yellow traffic lights at a fixed period and through the shared variable g light phase Broadcast the current light status to other tasks; **TrafficCreatorTask** randomly generates whether there are vehicles passing in a probabilistic manner based on the traffic level and writes it into the g car value. **TrafficDisplayTask** controls the LED display effect driven by the 74HC595 based on the traffic light status and vehicle data, achieving a traffic flow animation where the red light remains constantly on, the green light flows continuously, and the yellow light flashes Four tasks access shared resources through mutually exclusive semaphores to achieve system state synchronization and form a real-time response and visually intuitive intelligent traffic light system

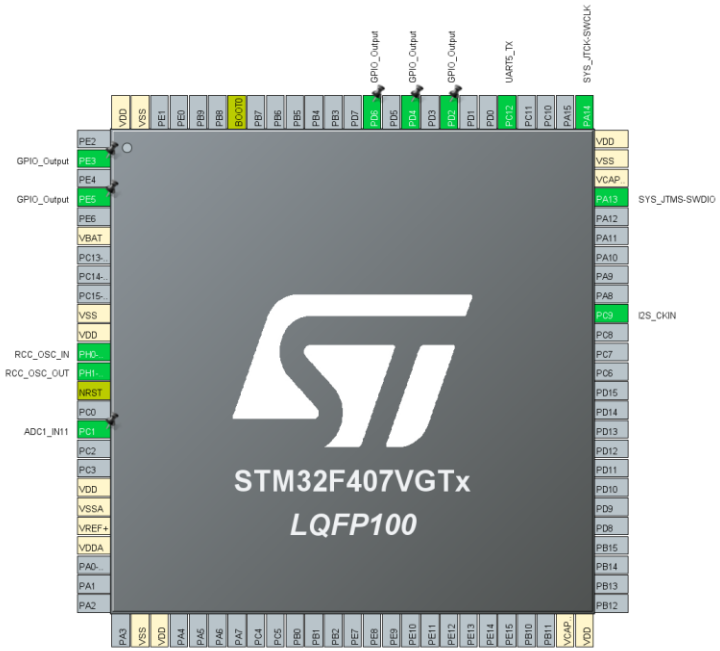
04

Demo of using

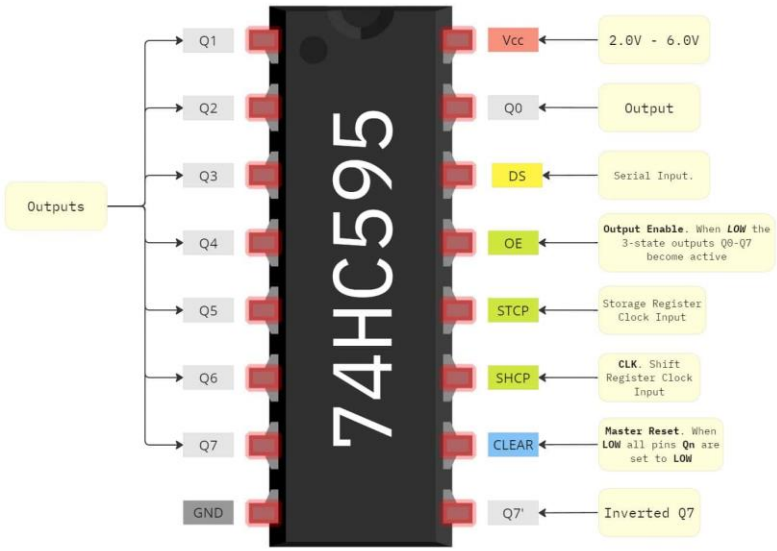
Demo of using



STM32F407 Discover



STM32F407 Discover
in STMCubeIDE



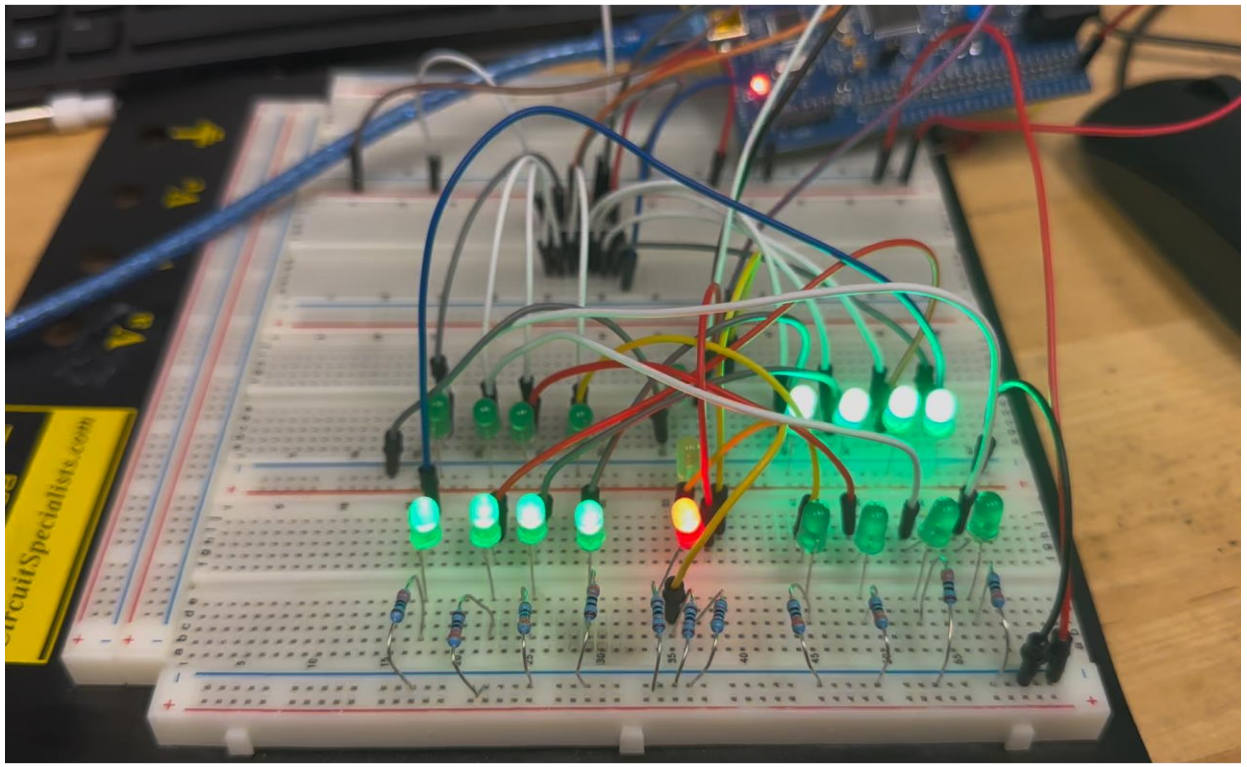
74HC595 chips

➢ Demo of using

🔴 Red light phase (lasting for 5 seconds)
Traffic light display: red light is on, the rest is off
Vehicle LED effect: All LEDs on the left lane are lit, indicating that the vehicle is waiting; all LEDs on the right are turned off, indicating that they are not accessible

🟢 Green light phase (lasting 10 seconds)
Traffic light display: Green light is on, the rest is off
Vehicle LED effect: The flowing light is quickly lit from left to right, and simulate the vehicle passing in sequence

🟡 Yellow light stage (lasting for 2 seconds)
Traffic light display: The yellow light is on, the rest is off
Vehicle LED effect: Four LEDs on the left (Q4–Q7) flash, four LEDs on the right (Q0–Q3) stay off



05

Project Summary

Problems encountered and solutions

1. Multitasking scheduling conflicts with synchronization

- Difficulty:

Multiple tasks need to read and write global variables (such as traffic light status), which is prone to data inconsistencies or abnormal system behavior

- Settle:

Protect shared variables with the FreeRTOS mutex (xSemaphoreCreateMutex).

All tasks that access g_light_phase are synchronously accessed via xMutexLight, avoiding race conditions

2. LED flow light control timing is out of order

- Difficulty:

When controlling multiple LEDs with the 74HC595, there were initial problems such as misaligned lighting and uneven flickering

- Settle:

Encapsulate the ShiftRegister.c control interface to manage SER/SRCLK/RCLK logic in a unified manner

The Latch() function is called after each update to ensure that the output is consistent

Learning & Improvement

Project Completion

1. Successfully built a complete traffic light control system on the STM32F407 development board
2. The main functions such as status switching of red and yellow lights, LED traffic flow simulation and so on are realized
3. A multi-task architecture is built using FreeRTOS, and each functional module runs stably and responds in a timely manner

Harvest and improvement

1. Master FreeRTOS multi-task scheduling, task creation and communication mechanism (semaphores, mutex locks)
2. Learn how to modularize hardware control logic (such as ShiftRegister, ADC, LED, etc.)
3. Understand the practical challenges of real-time, resource competition and task collaboration in embedded systems
4. The GPIO, ADC, interrupt, peripheral driver and other capabilities of the STM32 platform have been significantly improved
5. Lay a good foundation for future projects (such as smart cars and robot systems)

THANKS

Name: Cheng Chen, Tran Tri Tin, Afrooz Abbasi

Date: 04/23/2025