University of Victoria

Department of Electrical and Computer Engineering

ECE 455
REAL-TIME COMPUTER SYSTEMS DESIGN PROJECTS

Project 2

ECE-455 Project2 Manual

Part 2. Traffic Light System

Introduction

This project is intended to introduce you to middleware and to design a traffic light controller software that uses different features of FreeRTOS including task scheduling, task synchronization and software timers.

P2.1 Marking

This project counts for 45% of your Lab mark. Your mark for this project will be based mainly on your demo and your project documentation. The mark distribution is:

Overall Documentation: 10
Design Documents: 10
Code and Circuit Design and implementation: 10
Correctness: 15

Total: 45

P2.2 Project Report

Using your project report, a person familiar with this field should be able to understand and recreate your approach to the project. Some guidelines on what is expected in your project report are given below.

P2.2.1 Overall Documentation

The project report must be an appropriately formatted engineering report. The report should consist of an introduction, a discussion of the design (i.e., the design document), a discussion of the implementation, a brief summary, and an appendix with the source code. Note that the report will also be marked for presentation, grammar, punctuation, spelling, etc.

P2.2.2 Design Documents

Before you start any coding you must do a thorough design on paper. This should consist of diagrams indicating how the various tasks interact with each other, pseudo code, and so on.

P2.2.3 Code and Circuit Design and Implementation

The code must be implemented in C with good coding style: meaningful variable names (e.g., use a *counter* instead of *ct*), sufficient comments and documentation, modular design, and clean coding style (i.e., avoid *gotos*, etc.). Any problems encountered during implementation (i.e., bugs), testing methodology, and known bugs should also be described. To test and demo the software you need to build the traffic light system circuit.

The circuit is composed of a number of LEDs you need to turn on and off using the GPIO of the Discovery board and a potentiometer that is connected to one of the ADC channels.

P2.2.4 Correctness

This mark will depend on how much of the project work correctly, and if there are any design faults in the implementation.

P2.3 Project Description and Guidelines

In this project, you are designing a *traffic light* system for an intersection. The system is shown in Figure 1.

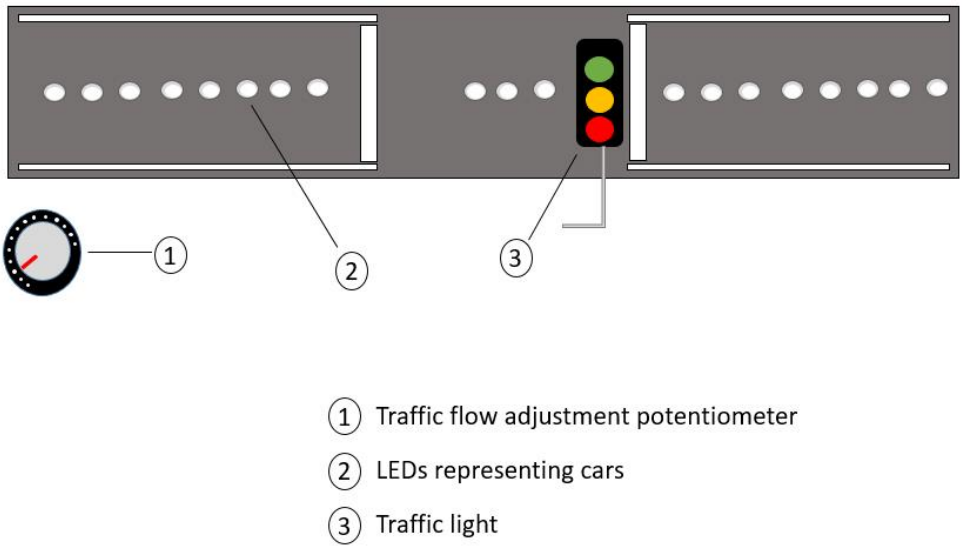

Figure 1. The traffic light system.

The system, as depicted in Figure 1, is composed of three sections as follow:

1. The traffic flow adjustment potentiometer: This potentiometer is used to adjust the flow the traffic entering the intersection. The low value (resistance) corresponds to light traffic and a high value (resistance) corresponds to heavy traffic.

2. The LEDs: These LEDs represent the position of a car at different points in time. If a LED is ON, it means a car presents on the road and the ON LED should be shifted to the left or right to indicate the car is moving. More than one car can exist on the road and their movement should be managed accordingly.

3. The traffic lights: A green, a yellow and a red LED that is used to control the traffic flow at the intersection.

There is a demo video (Traffic_Light_System_Demo.mp4) of the system on the CourseSpaces that shows the *Traffic Light System* in action.

P2.3.1 Traffic Light System Circuit

There are 19 LEDs on the road that you need to connect to the Discovery board. Eight LEDs are placed before the intersection, 3 LEDS at the middle of the intersection and 8 LEDs after the traffic light. Feel free to adjust the number of LEDs based on your taste, but there should be a sufficient number of LEDs on the road that the flow of cars can be tracked.

To save GPIO pins, we suggest you use shift registers (e.g. 74LS164) as a serial to parallel converter and connect the output of the shift registers to the LEDs. You also need to connect a potentiometer to one of the ADC channels of the Discovery board.

You may use the following video about shift registers helpful to get started.
How to use shift registers

P2.3.2 Traffic Light System Software

The software of the system is divided into two sections; the middleware for the ADC and GPIO peripherals and the user tasks to manage different parts of the system.

P2.3.2.1 The Middleware (Firmware)

Middleware is usually the software that mediates between application software and the kernel or device driver software. Middleware is also software that mediates and serves different application software. Specifically, middleware is an abstraction layer generally used on embedded devices with two or more applications in order to provide flexibility, security, portability, connectivity, intercommunication, and/or interoperability

mechanisms between applications [1]. Figure2 shows different layers of an Embedded System model.

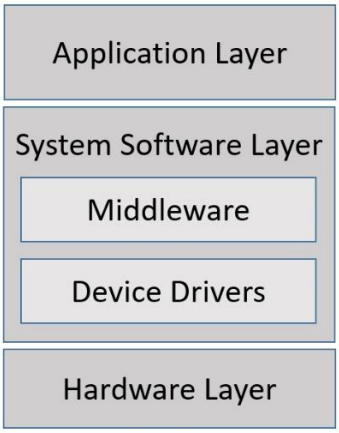

Figure 2. Embedded System Model [1].

For this project, middleware is essentially some functions you have to provide to get hardware into action. These functions include Initialization and Configuration and, also functions related to a specific action (e.g. read ADC). The sample discovery project you downloaded in your first lab project, includes default middleware provided by ST in Atollic TrueStudio. You can find them here:
Project Explorer -> Libraries -> STM32F4xx_StdPeriph_Driver -> src
For ADC, the default middleware file is *stm32f4xx_adc.c*. Note that not all functions and codes in the default file are required by every program. For this project, you need to write your own middleware from scratch.

P2.3.2.1 The User Tasks

You may be able to write a single threaded software for this project, but using FreeRTOS tasks will extremely simplify the software design. We encourage you to use the following structure for your application (Note: a single threaded application is not acceptable for this project!).

You need to create a number of tasks in the system to manage different resources. The recommended tasks are listed below.

1. **Traffic flow adjustment task:** The traffic flow that enters the intersection is set by a potentiometer. This task reads the value of the potentiometer at proper intervals. The low resistance of the potentiometer corresponds to light traffic and a high resistance corresponds to heavy traffic. The reading by this task is sent and used by other tasks.

2. **Traffic creator task:** This task generates random traffic with a rate that is based on the potentiometer value reading. This value is received from the traffic flow adjustment task. The created traffic is sent to the task that displays the flow of the cars on the road.

3. **Traffic light task**: This task controls the timing of the traffic light. This timing is affected by the load of the traffic which is received from the traffic flow adjustment task.

4. **Traffic display task:** This task controls the LEDs that represent the cars at the intersection. It receives the traffic from traffic creator task and displays them on the LEDs. It refreshes the LEDs at a certain interval to emulate the flow of the traffic.

You can add use a different set of tasks. You need to explain why your design is superior to the suggested design above.

You must use the software timers to manage the timing of the traffic light.

P2.4 Notes

This project is extremely difficult if not impossible to 'hack' together. Doing your design of the user tasks, the middleware functions and how the tasks communicate on paper is a must (i.e., detailed pseudo code). You will not save yourself any time by skipping this step and in fact, it will probably take you much longer to complete the project.
Furthermore, you are required to hand in your design document as part of your report.
As a rough estimate, you should expect to spend at least 20 hours designing, implementing, and testing the project.

References:

[1] Tammy Noergaard, "Guide to Embedded Systems Architecture - Part 1: Defining middleware." Online: https://www.eetimes.com/document.asp?doc_id=1276764