

Assignment 1: Programming a Traffic-Light Controller

Overview

In this assignment, you will program a traffic-light controller (**Figure 1**). Your program will handle multiple tasks, some of which are executed periodically, and others of which are triggered asynchronously. Periodic tasks are those executed by the program with a fixed period. Asynchronous tasks are those executed by the program in response to an external event signalled by an interrupt.

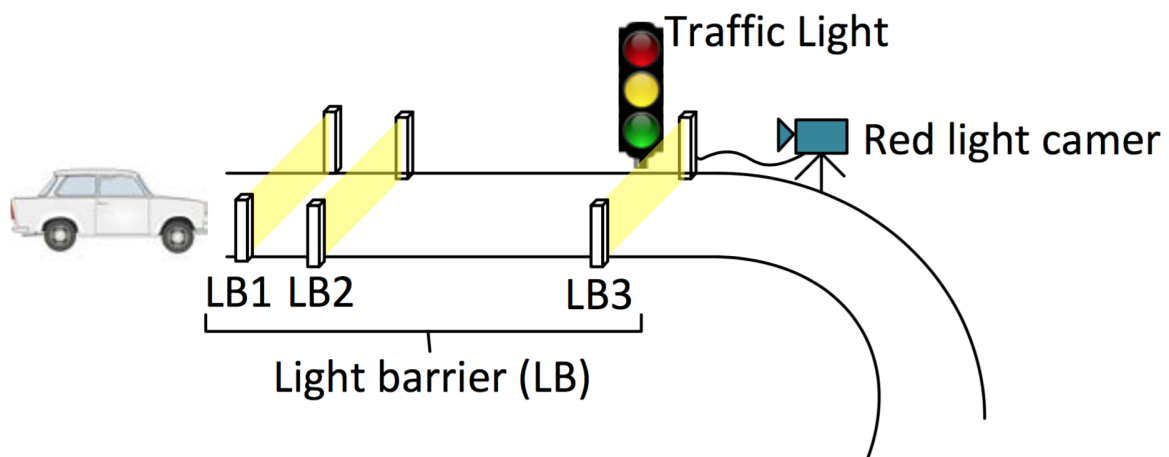


Figure 1: Traffic control system

The system that you will program comprises traffic lights, which control the flow of vehicles on a one-lane entrance to a freeway. The system estimates the speed of vehicles entering the freeway using **light barriers LB1 and LB2**. A **red-light camera** monitors the behaviour of drivers; **the camera is triggered if light barrier LB3 is breached while the traffic light is red**. The system also logs data, **counting the number of vehicles that runs a red light**.

This assignment is to be completed in groups of 2, implemented using low-level C programming language on an Arduino Uno board. Your traffic-light controller will be implemented as four subsystems, each of which will be implemented as a different program (i.e., you will submit four programs written in C).

As with Labs, please read this hand-out carefully before getting started.

Task 1: Basic Traffic Light (Normal Mode)

Implement a basic traffic light system. The traffic light changes state every second. The order of the cycle is as follows: red to green, green to yellow, yellow to red. This cycle should repeat for as long as the system remains on.

The traffic light will be simulated using three light-emitting diodes (LEDs); one red, one yellow, and one green. Each of these LEDs is connected to a different digital IO pin on an Arduino Uno (e.g., see [Figure 2](#)).

Task 2: Configurable Traffic Light (Configuration Mode)

Develop your basic traffic-light system from Task 1 so that it is now configurable. **The configurable system can be set to operate in one of four states.** In State 1 (the default state), the configurable system is like the basic system, changing state every second. There are three additional states; **different states are associated with different periods between each state transition**, as shown in Table 1.

Table 1: Traffic light states

State	Period (seconds)
1 (Default)	1
2	2
3	3
4	4

The traffic lights can only be configured when the system enters “configuration mode”. To enter configuration mode, the user presses a tactile push button (see [Figure 2](#)) at any time during the traffic light cycle, and in any state of operation (1 through 4). When that button is pressed, the system **continues to operate until the next red light**, at which point the system enters configuration mode.

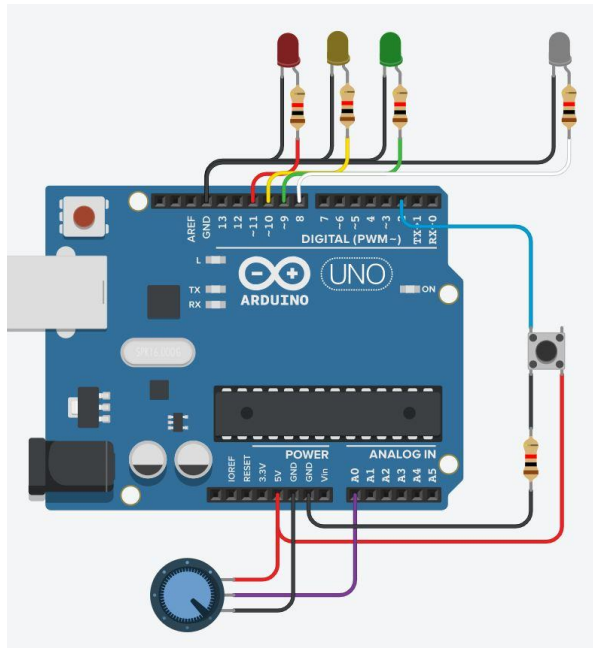


Figure 2: Configurable traffic light control system

In configuration mode:

- The traffic light should stay **red**.
- **A fourth LED** (see white LED at top-right of [Figure 2](#)) should be used to represent the current state of the traffic lights: this LED encodes a state number, x , by **blinking x times in 1 second**, and then **remaining off for 2 seconds**. For example, in State 3, the LED should issue three pulses (each pulse period = 0.33 s; duty cycle = 50%) and then remain off for 2 seconds. In State 4, the LED should issue four pulses (pulse period = 0.25 s; duty cycle = 50%) and then remain off for 2 seconds. This three-second cycle should repeat for as long as the system remains in configuration mode.
- **States can be selected using a potentiometer via ADC input** (see [Figure 2](#)). The full range of ADC outputs should be used when selecting a State. For example, if an 8-bit ADC produces 256 digital words/codes, then words/codes 0 through 63 should be used to select State 1, 64 through 127 to select State 2, and so on.

To exit configuration mode, the user presses the tactile push button again; traffic light operation then resumes in the newly selected state. **When the system is not in configuration mode, the fourth LED should be constantly illuminated.**

Task 3: Speed Monitor

Implement a system that **estimates the speed of vehicles** as they approach the freeway. The speed of vehicles is estimated using the light barriers LB1 and LB2. To estimate the speed of a vehicle, the system first computes the difference between (1) the time LB1 is breached, and (2) the time LB2 is breached (the “elapsed time”). Then, assuming **LB1 and LB2 are separated by 20 metres**, it computes an estimation of speed.

In this task, **we will remove the functionality of the three traffic light LEDs**. Instead, the red and green LEDs will be used to indicate the breaching of the light barriers LB1 and LB2, respectively. The sensing of light barriers LB1 and LB2 are simulated using tactile push buttons, as illustrated in **Figure 3**; **by pressing the left button we simulate the breach of LB1**, and **by pressing the right button we simulate the breach of LB2**. When any car breaches LB1 and LB2 (the push of the two buttons), each LED should output a short blink to indicate that its corresponding light barrier has been breached. You should choose an appropriate blink period.

The speed estimate, in units of km/h, is to be represented using a **pulse-width modulated (PWM) signal** (pulse period = 1 s). You should write this PWM signal to one of the Arduino Uno digital IO pins, and observe the signal using an oscilloscope (**Figure 3**). The encoding of speed is as follows: PWM signal **duty cycle is directly proportional to estimated speed**. For example, if the estimated speed is 10 km/h, the PWM signal duty cycle = 10%. For 20 km/h, then duty cycle = 20%, and so on. If the speed estimate ≥ 100 km/h, then duty cycle = 100%. The PWM signal should persist until another vehicle successfully passes both LB1 and LB2.

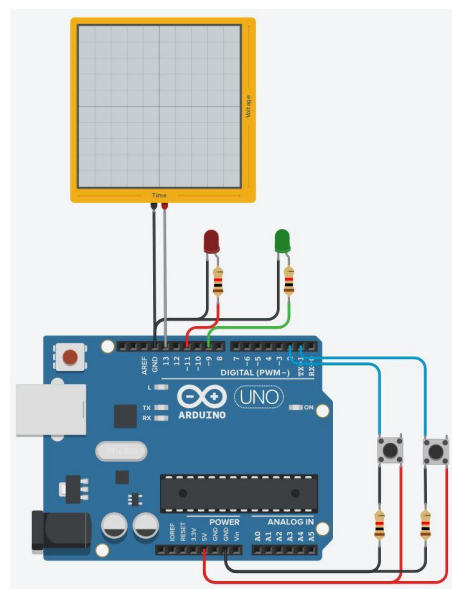


Figure 3: Circuit simulating the speed monitor system

The plot below (Figure 4) illustrates speed versus elapsed time for an ideal speed monitor system. If a vehicle is travelling very fast (e.g., 150 km/h), then small errors in the estimate of elapsed time will cause a relatively large error in the speed estimate. The errors associated with the speed estimates of the system you implement should be **no greater than 1 km/h** for speeds up to 150 km/h. A speed of 150 km/h corresponds to an elapsed time of 0.480 s, and 151 km/h corresponds to an elapsed time of 0.477 s. This 3 ms difference means the response time of your system to breach of LB1 or LB2 must be **less than 1.5 ms**. You will be **required to demonstrate** that your system meets the timing constraint by probing the input signal from each push button (LB1 and LB2) and the output signal from each LED (LB1 and LB2) using an oscilloscope. The difference in time between the rising edges of the input and output signals **must be less than 1.5 ms**.

Furthermore, your system should be robust and operable in various real-world scenarios, such as allowing multiple vehicles to pass (and more).

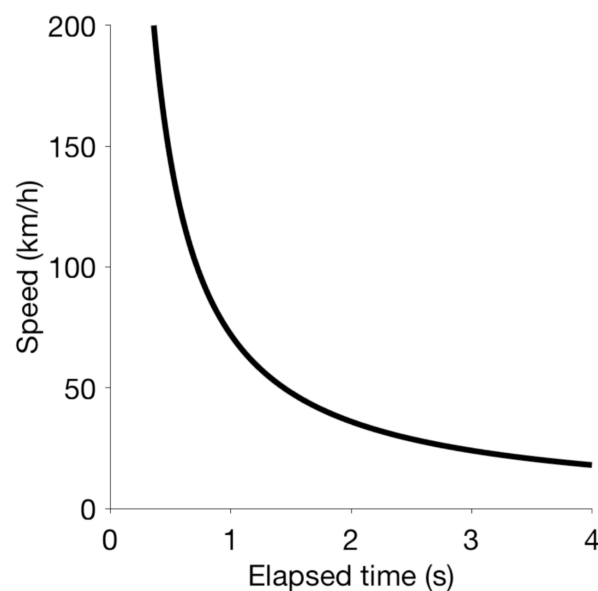


Figure 4: Speed-versus-elapsed time for an ideal speed monitor system. Elapsed time = time of LB2 breach - time of LB1 breach. Because LB1 and LB2 are separated by 20 m, then an elapsed time of 0.72 s, for example, corresponds to a speed of 100 km/h.

Task 4: Red-Light Camera

Reintroduce the traffic light system from Task 1 and cycle in normal mode (i.e., change state every second). Implement a red-light camera. The **camera is triggered if a vehicle breaches light barrier LB3 while the traffic light is red**. The light barrier LB3 is simulated using a tactile push button, as illustrated in [Figure 5](#). By **pressing the button we simulate the breach of LB3**. To indicate that the camera has been triggered, the white LED should be pulsed twice in rapid succession (pulse period = 0.25 s; duty cycle = 50%). The flashing of the LED should not otherwise affect the running of your system.

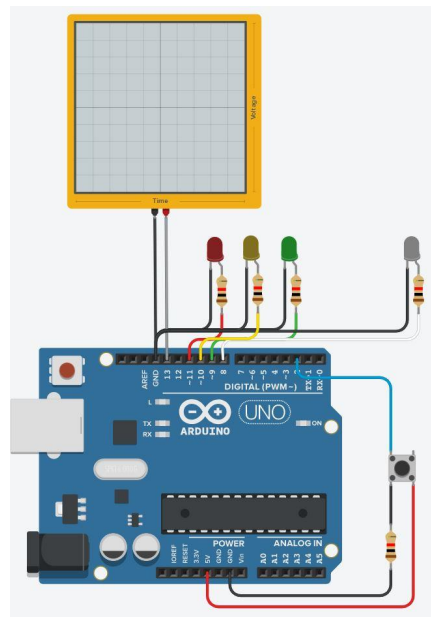


Figure 5: Circuit simulating a red-light camera

When light barrier LB3 is breached, the system must **respond within 10 ms** (enabling an image of the vehicle's number plate to be taken before the vehicle leaves the camera's frame). You will be **required to demonstrate** that your system meets the timing constraint by probing the input signal from the push button and the output signal from the fourth LED using an oscilloscope. The difference in time between the rising edges of the two signals **must be less than 10 ms**.

The system should keep track of the **total number of cars that have triggered the red-light camera**. This count is to be **encoded using a PWM signal** (pulse period = 1 s). You should write this PWM signal to an Arduino Uno digital IO pin, and observe the signal using an oscilloscope ([Figure 5](#)). The encoding of this count is as follows: PWM signal duty cycle is directly proportional to the count. For example, if the count is 10, the PWM signal duty cycle = 10% and so on. If the count ≥ 100 , then duty cycle = 100%. The PWM signal should persist until the count increases.

Marking Criteria

This assignment is worth 20% of your final grade for MECHENG 313.

Demonstration (10%). In labs, you will be asked to demonstrate your system on an Arduino Uno microcontroller board (if you develop code using TinkerCAD, you need to verify that it runs on an actual Arduino Uno microcontroller board prior to your demonstration). TAs will check that your system's behaviour conforms to the above-described specifications.

Software (5%). Your code will be assessed based on its readability (good commenting and clear, consistent style) and your technical proficiency. You are expected to use low-level C, as described in lectures, and avoid using the Arduino application programming interface (API).

NB: Do not share your code with other groups. The group that submits the shortest code will earn up to **2 bonus marks**. To earn these 2 bonus marks, you must get 10/10 for your demonstration. We will quantify the length of your code by counting the number of semicolons you use. You can't earn more than 20 marks in total for this assignment.

Report (5%). Write a report describing your system (12pt font, single-spaced). Your report should comprise the following sections: Introduction, Methods, Results, Discussion. Your Introduction should run to approximately half a page. There, you should give an overview of your system; this overview should be pitched to a non-specialist mostly unfamiliar with microcontrollers and embedded systems (e.g., a Part 3 student enrolled in Civil/Environmental). In Methods, you should describe your system in sufficient detail such that a specialist familiar with the Arduino Uno and the ATmega328P can reproduce your implementation (e.g., pitch your Methods section to a fellow Part 3 Mechatronics student). In Results, describe how your design enabled the system to meet the following two deadlines:

1. Red-light camera: "When light barrier LB3 is breached, the system must respond within 10 ms."
2. Speed monitor: "The errors associated with the speed estimates of the system you implement should be no greater than 1 km/h for speeds up to 150 km/h."

In Discussion, you should describe any nontrivial design decisions and assumptions that you made, making reference to real-time concepts described in lectures (e.g., was programmed IO an appropriate solution to a design problem, or is all IO interrupt-driven?) Your Discussion should run to no more than 1 page. Overall, your report should be no more than 5 pages, including any necessary plots, images, and/or appendices.



Files to Submit

Files are to be submitted on Canvas.

Software submission. One submission per group. Please submit a zip file. This zip file (e.g., “MECHENG313_A1_Group20.zip”) should contain several C source code files (e.g., “Task3.c”), as well as a “README.txt” text file. The README.txt file should list names and UPIs of group members. It should also describe clearly the functionality of each C source code file and its corresponding task.

Report. One submission per group. Please submit a PDF.

Peer Review. One submission per person. Please use the form provided on Canvas.

