

**ENGD2103 EMBEDDED SYSTEMS FUNDAMENTALS**  
**Laboratory guidance notes**  
**Week 2: Formative Assignment – Part 1 – Traffic Lights Sequencer**

Author: M A Oliver, J A Gow, T Allidina

v1.0

## Table of Contents

1 Aims and objectives.....	1
2 Overview.....	1
3 The Traffic Lights Sequencer Code.....	2
4 Hardware Abstraction Layer (HAL).....	3
Appendix: LED Blinker Code.....	4

### Version record

Version	Notes	Date	By
1.0	Initial version	07/10/22	MAO

## PLEASE READ THIS CAREFULLY

This is the second of a series of guidance documents to get you started with your Embedded Systems Fundamentals projects. They are aligned with the key aspects of the formative coursework specification, and thus this document should be read in conjunction with the coursework specification document. These documents will start off with some fairly close guidance - however as you develop your skills the notes will provide less and less detailed guidance and become more of an overview.

## 1 Aims and objectives

During the formative assignment, you will be writing three short pieces of code. This document outlines the first part: a traffic lights sequencer.

Prerequisite: you should have completed the assembly of the circuit from Week 1 before commencing this task.

## 2 Overview

















Consider a crossroads controlled by traffic lights. At the crossroads, suppose there is a road running North-South and another road running West-East. One set of traffic lights

(Set 1) will control traffic entering the crossroads from the North-South road. The other set of traffic lights (Set 2) will control traffic entering the junction from the West-East road.

Your task is to write the code to sequence the traffic lights.

### 3 The Traffic Lights Sequencer Code

Your task is to write a traffic lights sequencer that implements the following sequence.

STATE	LIGHTS SET 1	LIGHTS SET 2	TIME (s)
1			1
2			1
3			5
4			1
5			1
6			1
7			3
8			1

In Week 1's lecture an example was shown how to blink an LED. The code is revisited in the appendix.

The `pinMode()` function is used to configure a digital I/O line as an input or output.

The `digitalWrite()` function can be used to set an output pin to logic high (around 5V) to illuminate an LED, or to logic low (around 0V) to extinguish an LED.

The `delay()` function (which will be allowed for the formative task, but prohibited in the summative task) can produce a blocking delay; its argument determines the delay in milliseconds.

These techniques can be extended to implement the traffic lights sequencer.

## 4 Hardware Abstraction Layer (HAL)

In embedded systems projects it is good practice to create a Hardware Abstraction Layer (HAL) module. This contains all the low-level platform / processor-specific code. Having a Hardware Abstraction Layer means that if a project needed to be ported over to a different processor, most of the required changes would involve modifying the code in the HAL.

Hardware Abstraction Layers can be implemented as a module containing two files:-

hal.h	Header (definitions) file
hal.cpp	Source (implementations) file.

Considering the example in the Appendix, the following definitions can potentially reside in the HAL header file:-

```
#define led          13
#define HAL_ledOn    digitalWrite(led, HIGH)
#define HAL_ledOff   digitalWrite(led, LOW)
```

Using this definition alone, the `loop()` function would now become

```
void loop()
{
    HAL_ledOn;           // Turn the LED on
    delay(500);          // Wait for 500 ms
    HAL_ledOff;          // Turn the LED off
    delay(200);          // Wait for 200ms
}
```

The microcontroller specifics for driving the LED have been abstracted from the firmware engineer. Suppose this project needs to be ported to a different microcontroller (or even the LED needs to be moved to another pin), a code developer can simply go into the HAL files and make the necessary changes for the definitions for `HAL_ledOn` and `HAL_ledOff`.

Think about the `loop()` function. Are all the platform-specifics abstracted from the firmware engineer? If not, how would you go about rectifying that?

A template is provided containing three files:-

- `form1.ino` This is a blank Arduino sketch that `#includes` `hal.h`.  
Your traffic lights sequencing code should be written here.
- `hal.h` This is a blank HAL header file containing some preprocessor directives, and `#includes` the `<Arduino.h>` for platform-specific functionality.  
Your platform-specific definitions should be written here.

- `hal.cpp` This is a blank HAL definition file that `#includes` `hal.h`.  
Your platform-specific function implementations should reside here.

## Appendix: LED Blinker Code

```
#define led      13           // Using the built-in LED.

void setup()
{
    pinMode(led, OUTPUT);    // Make the LED pin an output
}

void loop()
{
    digitalWrite(led, HIGH); // Turn the LED on
    delay(500);              // Wait for 500 ms
    digitalWrite(led, LOW);  // Turn the LED off
    delay(200);              // Wait for 200ms
}
```