ENGD2103 EMBEDDED SYSTEMS FUNDAMENTALS
Laboratory guidance notes
Week 10: Summative Assignment – Part 4 – Priority Switching and Robust Coding

Author: M A Oliver, J A Gow, T Allidina *v1.0*

Table of Contents

Version record

Version	Notes	Date	By
1.0	Initial version	04/12/22	MAO

PLEASE READ THIS CAREFULLY

This is the fourth and final part of a series of guidance documents to get you started with your Embedded Systems Fundamentals assignment. They are aligned with the key aspects of the summative 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 summative assignment, you will be creating a single project capable of performing multiple tasks concurrently. This document outlines the fourth part: priority scheduling.

Prerequisite: you should have completed the material from Weeks 7, 8 and 9 before embarking on this exercise.

2 Priority Switching

The traffic lights can run in one of three possible modes:-

Equal priority (where both sets of lights have equal green time)

Set 1 priority (where Set 1 has a longer green time than Set 2)

Set 2 priority (where Set 2 has a longer green time than Set 1)

An enumerated type can be defined to represent this:-

```
typedef enum
{
    Equal,
    Set1,
    Set2
} priority_t;
```

A variable called `priority` can now be declared.

```
priority_t  priority;
```

From the main assessment specification, in Mode 5, the traffic lights and accelerometer run concurrently. The priority of the traffic lights needs to be changed on-the-fly by the orientation of the board.

For each of the six orientations, the corresponding value of `Equal`, `Set1` or `Set2` can be assigned to the variable `priority`.

There are several methods for changing the priority. Perhaps the simplest is to examine the priority variable within the traffic lights finite state machine in states 3 and 7. Here the (green) time can be assigned based on the value of `priority`.

3 Rollover-safe Timing

The `millis()` function returns the number of milliseconds the ATmega328P processor has been running since its last reset. The function returns this value as an `unsigned long` value – a 32-bit value capable of storing numbers from 0 to (2^{32} - 1). This means that after 4,294,967,295ms (or 49.7 days), the value will roll-over to zero and issues will occur with the timing.

Care needs to be taken to ensure the timing is "rollover safe". This can be achieved by casting the difference between the current time and the module time to a `long` (as per the Week 10 lecture).

4 Preserving D0 and D1

Suppose you are setting a single bit in Port D, such as:-

```
PORTD |= B01000000;
```

This would pose no issues as, for this example, only bit 6 is being changed. The remainder of the port remains unchanged. If you have taken this approach and are individually turning each LED on and off, you will need to take no further action.

Now, suppose you write to Port D in one go, such as:-

```
PORTD = (B01000000 | B00001000);
```

This would write the value B01001000 to Port D. The required LEDs would illuminate, but D0 and D1 would be overwritten with zeros. This could cause issues if D0 and D1 ever needed to be used.

If an approach like this is being taken, then the lines D0 and D1 should firstly be preserved through `(PORTD & B00000011)`. The operation to switch on the two LEDs would now be:-

```
PORTD = (PORTD & B00000011) | (B01000000 | B00001000);
```

5 Race Conditions on the 7-segment Display

You might be experiencing race conditions on the 7-segment display. This could occur due to the button counter or the accelerometer attempting to write characters to the display and the heartbeat module writing to the dot point. We essentially have two processes competing for the same resource. If care is not taken, each process may overwrite the display.

If you are experiencing this, there are steps that can be taken to counter this.

Firstly, have a ***single*** call to your shift register function throughout the code. This should be the final statement within your `loop()` function.

Have a variable for the counter and accelerometer modules that writes the necessary character to this variable, ***not*** to the shift register.

Have a second variable for the heartbeat module. The module simply sets / clears the dot-point bit (bit 7) in the variable, ***not*** the shift register.

Finally combine (OR) these variables together and write the result to the shift register in the final statement within the `loop()` function.