ENGD2103 EMBEDDED SYSTEMS FUNDAMENTALS
Laboratory guidance notes
Week 9: Summative Assignment – Part 3 – Orientation Detection and Scheduler

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

Table of Contents

Version record

Version	Notes	Date	By
1.0	Initial version	25/11/22	MAO

PLEASE READ THIS CAREFULLY

This is the first 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 third part: orientation detection and scheduler.

Prerequisite: as a bare minimum you should have completed Formative Task 3 before commencing the Orientation Detection. You should have at least one other main module (traffic lights or button counter) from Weeks 7 and 8 respectively completed.

2 Overview

The purpose of this task is to give you experience of:-

- Getting multiple code modules running together.
- Being able to schedule which modules to run and which ones to hold.

3 Task Requirements

In this task, you will be required to implement orientation detection in a non-blocking fashion and to implement a scheduler.

4 Orientation detection

In your HAL.h you could define the six possible orientations:-

```
#define ORIENTATION_FLAT                    0
#define ORIENTATION_BASEUP                  1
#define ORIENTATION_LANDSCAPE               2
#define ORIENTATION_UPSIDEDOWN_LANDSCAPE    3
#define ORIENTATION_LEFTPORTAIT             4
#define ORIENTATION_RIGHTPORTRAIT           5
```

In HAL.h you could define a function prototype for determining the orientation:-

```
int HAL_orientation();
```

In HAL.cpp you could implement the function `HAL_orientation()`.

The skeleton for this function would be:-

```
int HAL_orientation()
{
  static int orientation = ORIENTATION_FLAT;   // default

  // Read the accelerometer

  // Determine the board orientation and store the result
  // in the variable 'orientation'

  // Return the orientation
  return orientation;
}
```

You would need to port-over your code from Formative Task 3 that reads the accelerometer.

You would also need to port-over your code that determines the board orientation. You will be able to assign one of the orientation definitions to the variable `orientation` for each of the six possible orientations. (The variable `orientation` is declared as `static` such that its previous value is retained. This is necessary in instances where an orientation cannot be determined.)

5 Non-blocking orientation detection

During Weeks 7 and 8 you will have gained experience of creating non-blocking code modules for concurrent operation.

Now, create a code module for the orientation detection. This should have a module delay of 333ms.

Every 333ms, the module needs to carry out its tasks and should do the following:-

- Determine the orientation using your `HAL_orientation()` function.
- Take the steps to display the orientation on the seven-segment display. Care needs to be taken to prevent race conditions on the display.

Once working, you are encouraged to encapsulate this. If attempting the full encapsulation, it might be worth declaring a `private` member variable called `orientation`, and write a single accessor function to read it.

6 Scheduler

Now, create a new module for the scheduler. This should provide concurrent timing, and should allow for a FSM. You could use an existing module as a template.

This module will need to determine which of the three main modules (traffic lights, button counter and orientation detection) will run and which will be held. The scheduler must run fairly quickly; the module delay should be shorter than those of the three main modules.

A finite state machine will be needed to cycle through the modes. A debounced press of SW2 should move the system into the next mode. There are 5 modes to implement.

Mode 1: traffic lights are running, button counter and orientation detection are held.
Mode 2: button counter is running, traffic lights and orientation detection are held.
Mode 3: orientation detection is running, traffic lights and button counter are held.
Mode 4: traffic lights and button counter are running, orientation detection is held.
Mode 5: traffic lights and orientation detection are running, button counter is held.

Will five states be enough to implement this? Or will you need more states to deal with switch releases?

Try incorporating all of your main modules into your project and see if you can successfully change modes.

7 Next week

Next week we will look at the priority selection for the traffic lights system, and some final tidying.