ENGD2103 EMBEDDED SYSTEMS FUNDAMENTALS
Laboratory guidance notes
Week 8: Summative Assignment – Part 2 – Debouncers, Button Counter and Heartbeat

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

Table of Contents

Version record

Version	Notes	Date	By
1.0	Initial version	20/11/22	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 first part: debounced switches, counter and heartbeat.

Prerequisite: you should have constructed the breadboard layout, completed all the formative tasks, and completed the Week 7 lab sheet before commencing this task.

2 Overview

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

- Understanding non-blocking debouncing
- Implementing a non-blocking button counter using non-blocking debounced switches.
- Implement a non-blocking heartbeat module.

3 Task Requirements

In this task, you will be required to study some code to implement a non-blocking switch debouncer. You will then extend this to debounce the remaining switch.

Using this project as a start-point, you can then add modules to perform button counting and heartbeat.

You are encouraged to use encapsulation for your code modules (see Lectures 5 & 6) to improve the structure of your project

4 Non-blocking debounced switches

Study the code for `debounce.ino` provided. Some code for driving switch SW1 on pin A0 is provided. This should ultimately be converted to bare metal and moved into your HAL module.

This implements a non-blocking switch debouncer for the switch on pin A0.

This is based on a timing module firing every 15ms and a finite state machine (FSM) containing 3 states. These states could have been labelled 0, 1 and 2. To make the code more readable, an enumerated type called `switch_state_t` has been defined containing 3 possibilities:-

• `NOT_PRESSED`	The compiler treats this as the integer value 0
• `PARTIAL_PRESS`	The compiler treats this as the integer value 1
• `DEBOUNCED_PRESS`	The compiler treats this as the integer value 2.

By examining the code, the finite state machine does the following:

In the `NOT_PRESSED` state: if the button is released, the FSM stays in the `NOT_PRESSED` state. If the button is pressed, the variable `debounce_time` acquires the current time using the `millis()` function, then jumps to the `PARTIAL_PRESS` state. For reference, `millis()` returns the number of milliseconds since the microcontroller was last reset.

In the `PARTIAL_PRESS` state: if the button is released, the FSM returns to the `NOT_PRESSED` state. If the button is pressed, either the finite state machine remains in the

`PARTIAL_PRESS` state if the difference between the current time and the `debounce_time` is less than the value of `debounce`, or the FSM jumps to the `DEBOUNCED_PRESS` state if the switch has been continually pressed for `debounce` milliseconds or more.

In the `DEBOUNCED_PRESS` state: if the button is released the FSM will return to the `NOT_PRESSED` state, otherwise the FSM remains in the `DEBOUNCED_PRESS` state.

The current state of the FSM is stored in the variable `B1_state`.

Try running this code to understand this behaviour.

Now, try extending this code to incorporate the second switch on pin A1. You will need a second FSM to deal with this switch as A1 is independent of A0. Will you decide to use a different timing routine, or will you associate both FSMs with a single timing routine?

Once you have done this, try encapsulating the debounce code. Will you have two debouncer modules (one per switch), or will you combine both switches into a single module? If you are attempting the full encapsulation, you can avoid having `B1_state` and `B2_state` in your encapsulations by implementing accessor functions to simply return these states.

5 Button Counter

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

This module should run fairly quickly. A `module_delay` in the region of 20ms to 40ms should suffice.

Before the FSM itself, the value of `count` should be set up for display on the seven-segment display. The required bit pattern could, at this stage, be stored in a variable.

In the FSM region, the `count` should be incremented on the detection of a debounced switch press.

Some care needs to be taken here. Suppose the `count` is incremented whenever the switch debouncer is seen to be in the `DEBOUNCED_PRESS` state, then the value of `count` will continually and quickly increment whenever the switch is debounced and held.

To deal with this, you could potentially have two states in your FSM.

- In the first state, the FSM should remain in this state if the debouncer is not in the `DEBOUNCED_PRESS` state. If the debouncer is now in the `DEBOUNCED_PRESS` state then the FSM should transition to the second state; at this stage the count should be incremented.

- In the second state, the FSM should remain here whilst the debouncer is in the `DEBOUNCED_PRESS` state. It should only revert back to the first state when the debouncer is not in the `DEBOUNCED_PRESS` state.

You might want to write your bit pattern to the shift register / 7-segment display at the end of the `loop()` function.

Don't forget to convert your shift register driver code to bare metal. Also try and encapsulate this module.

6 Heartbeat module

The intention of having a heartbeat module is to provide a visual indication that the system is up-and-running by blinking the dot-point on the 7-segment display on and off.

You can create a new module for the heartbeat. This should provide concurrent timing, and should allow for a FSM. You could use an LED blinker example module as a template.

The required `module_delay` for the heartbeat module can be found in the main specification.

The FSM should have two states.
- In the first state, the dot-point should be illuminated before jumping to the second state.
- In the second state, the dot-point should be extinguished before jumping back to the first state.

Again, a degree of care needs to be undertaken. The button counter module writes to the 7-segment display. The heartbeat module also writes to the 7-segment display. If sufficient care is not taken, a “race condition” will occur causing undesirable output on the display. One way of dealing with this ***starts*** by updating the display as the last statement of your `loop()` function.

7 Traffic Lights Sequencer

You should now be able to incorporate your traffic lights sequencer module from last week into your project for this week. If all goes well, you should be able to get your traffic lights and button counter running concurrently with the heartbeat and debouncer modules.

Next week we will look at scheduling and how individual modules can be turned on and off.