SCC.369: Lab Exercise 2

Timers and Interrupts

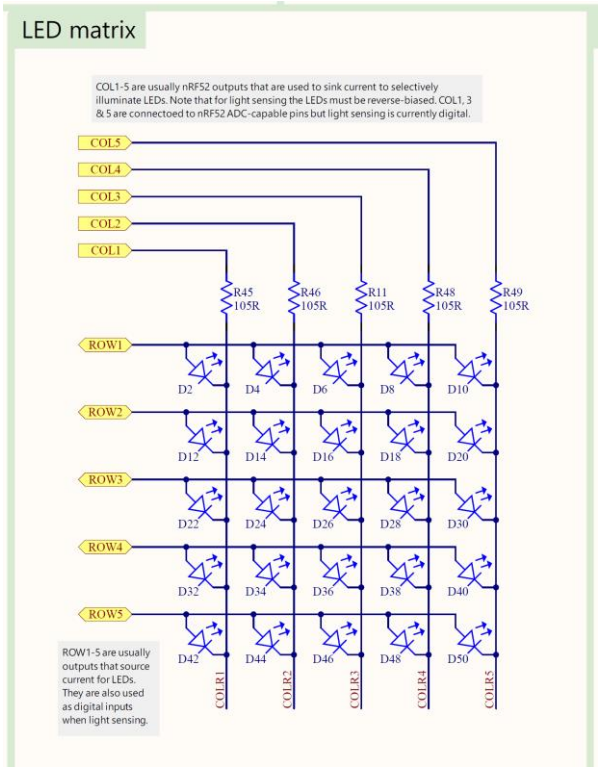
Aim of the Exercise
The aim of this exercise is to introduce you to the hardware Timer peripherals that are typically found on microcontroller systems, and how the use of interrupts can allow you to create power efficient, real-time systems.

Submission: **SCC.369 moodle submission, 16:00 Friday Week 4.**
Weighting: 25%

The Task

The front of the micro:bit contains 25 red LEDs, that are used to form a simple display, capable of displaying letters, numbers and simple graphics (typically emojis!). In this task, you will be controlling this display from first principles (so again, no external libraries). Once more, this exercise is split into a number of subtasks, listed below. Implement each one of them **from first principles using pure C/C++.** This means that you are **NOT** permitted to use any library code or function calls that you have not written yourself. You **are** encouraged to use the constants and types (structs) defined as part of the nrf52 SDK to keep your code clean and readable, and the NVIC calls to support interrupt configuration.

Firstly, review the micro:bit schematic provided on moodle, and you should see that the LED matrix contains the following wiring:



Notice how this looks quite familiar to the LEDs you wired on your breadboard in Task 1 (a resistor placed in series with an LED). Review the nrf52833 part of the schematic, and you will notice that the pins labelled ROW and COL are all just GPIO pins, connected to your microcontroller… so we all know what to do with that!

However, notice that the **anode** (positive side) of the five LEDs in the top row are all connected to the same pin labelled ROW1. Similarly for the other four rows. Notice also that the cathode (negative side) of the leftmost column of LEDs are connected to the same pin labelled COL1, and similarly for the other four columns. Also, each column contains a series resistor to limit current (just like the ones you put on your breadboard last week).

This may seem like a strange arrangement. But, if you think about it, you can now use just 10 GPIO pins to control 25 LEDs. This means we can therefore use a smaller, cheaper processor to solve the problem. This saves us money, size on the PCB and even natural resources!

Matt Oppenheim has kindly created a resource for you that simulates how this LED matrix operated. You'll find it on the moodle web page. I'd strongly suggest you experiment with it!

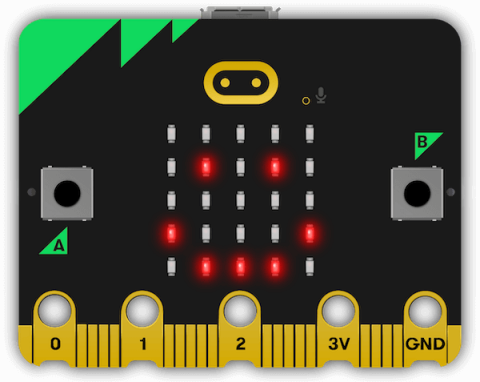
Create a file called **Task2.cpp**, and write functions that meet the following specification. Take care to ensure you use the function name and parameter list specified, as we will be using automated testing as part of the marking of your work.

Subtask 1: Don't Worry…. Be Happy….
Function prototype: `void beHappy();`

The title says it all, and its great advice! Write a function **from first principles**, that displays a smiley emoji on the screen when run… you can be artistic with your emoji. As long as it's happy. Micro:bits are like students - they should always be happy.

Now you might be wondering how to display more than one row at a time… well… **you can't**. You need to cheat by taking advantage of the limitations of human perception. 😊

Humans can only see things that change more slowly than about 50Hz. This is why our mains electricity supply runs at 50Hz AC – that way you can't see your light bulbs flickering. Unlike pigeons (but that's another story!). Hence, you can create the **illusion** of all the LEDs being individually controlled by taking in turns which row you are displaying. Provided that you **update the entire display within 1/50 of a second**, it will look smooth to humans.



Subtask 2: Using a Hardware Timer
Function prototype: `void beVeryHappy();`

Complete the same challenge as Subtask 1, but use a hardware timer to measure the passage of time, rather than a busy loop. We suggest you use the Timer1 peripheral for this task.

Subtask 3: Using Interrupts
Function prototype: `void beHappyAndFree();`

Complete the same challenge as Subtask 2, but also use interrupts to determine when the row/columns values should be updated. Ensure that you don't update the display faster than necessary. Remember, updating the display at 50Hz is fast enough for humans. Updating much faster than this will just waste CPU cycles… which in turn leads to less CPU for other tasks, energy inefficiency, global warming and the death of polar bear babies… so let's not do that.

Subtask 4: Controlling the Content
Function prototype: `void showNumber(int n);`

Take the number **n** provided as a parameter, and display that as a single decimal digital on the LED display. Again, you may be creative with your typeface/font, but the displayed number should be recognizable. You may ignore values outside the range 0..9. However, scrolling larger numbers is just… cool.

Assessment

Marks will be awarded for each of the tasks completed, the quality of the code produced, and the quality of the comments written. This work will be assessed through offline marking and a code inspection of your work.

Marking Scheme

Your work will be marked based on the following four categories. Your final grade will be determined based on a weighted mean of the three weight-bearing grades shown in the table below.

Subtask 1: Don't Worry…. Be Happy….	30%
Subtask 2: Using a Hardware Timer	20%
Subtask 3: Using Interrupts	20%
Subtask 4: Controlling the Content	20%
Code Style and Commenting	10%