Lab Sheet 1: Introduction to Keil µVision

***This lab exercise is not assessed**. However, a practice submission will be available. This is intended to allow you to: a) get feedback b) practice the submission arrangements used.*

There are several videos that introduce different aspects of the practical work: see the QMplus page.

1 Aims

The aim of this lab is to examine and modify an example program to:

- Become familiar with the Keil µVision IDE.
- Start using Git Classroom
- Understand digital output using the GPIO peripheral.

When you have completed this lab you should know how to:

- Clone a git repository, commit and push your work.
- Build a project to the Freescale Freedom board.
- Control a digital output using a GPIO port.
- [**Kit Required**] Use the debugger to view assembly code, set breakpoints and inspect program variables.

Some notes on using the Keil microVision IDE are at the end of this sheet. Further help is available in the IDE, with links to Keil's web page.

2 Activities

Perform the following tasks and answer the questions.

1. Download and run the sample project using a Freescale Freedom board.
2. Use debug mode to inspect the program.
3. Understand how the program works and modify it.
4. *(Advanced) Further modify the program to have multiple tasks.*

2.1 Activity 1: Download, Read, Compile and Run the Sample Project

A sample project is available from the course QMPlus web page using Git Hub. Follow the instructions on QMPlus to set up a repository on Git Hub and then clone the project repository into a suitable directory (not the desktop) locally. The directory structure must be preserved.

The Freedom board includes a multi-colour LED (which is a red, green and blue LED in a single package). The sample program flashes two colours in turn: on for 1 second, off for 1 second.

Start the Keil microVision software. Use the open project (not open file) menu to open the project.

Carefully review the program code:

- main.c contains the code: read the comments carefully
- gpio.h contains some macro definitions
- There are two other files: SysTick.c and SysTick.h: these will be explained later.

Look at the notes at the end of this sheet to:

- Build the project
- [**Lab Kit Required**] Download the project to the Freedom board
- [**Lab Kit Required**] Run the program. Pressing the reset switch also causes it to restart.
- [**Lab Kit Required**] Verify that the program is preserved (in the Flash memory) when the power is removed.

Review
The steps required to run programmes on the KL25Z board are:
1. Create or edit the source code using the ARM Keil MDK on a PC
2. Compile the code **on** the PC but **for** the ARM Cortex-M0+
3. Download the compiled binary to the KL25Z using the debug interface (connected with USB)
4. Press the reset button on the KL25Z board to start the programme

*2.2 [*Lab Kit Required*] Activity 2: Use Debug Mode to Inspect the Program*
Enter debug mode. Complete the following steps:
1. Insert a break point in the program. When the program reaches the break point, step through the program instructions.
2. Look at the assembly (machine) code listing and the instructions corresponding to the C source code.
3. Using a breakpoint (or otherwise), find the memory address (in hex) of the port used to turn the green LED on.

2.3 Activity 3: Modify the Program
Since the 3 LED colours are in the same device, a mixture of colours can be produced. If all three colours are alight together, the LED appears white. Three other colours are possible by lighting two out of the three LEDs.

This would be a good time to review the instructions on the QMPlus page on the Coding Requirements for Embedded Systems, covering both the **Presentation Requirements** and the **Design Guidelines for Cyclic Systems**.

Modify the program so that:

1. The 7 possible colours flash in sequence (with no gaps)
2. The three 'single' colours (red, green, blue) are on for 2 seconds while the four mixtures (including white) are on for 1 sec. The whole cycle should repeat after 10sec and continue for ever.
3. The program should be organised to have 7 states (instead of 4 states as at present).

2.4 Activity 4: Push Changes and Update README File
Your work is submitted by updating the repository on Git Hub. You can do this as often as you like. It is strongly recommended that you try it now.

Try the following steps:

1. Push changes to the Git Hub repository (see instructions on QMPlus page)

2. Review the formatting of code as it appears on the Git Hub
3. Update the Git Hub README file so that it:
    a. Is correctly describes the work you have done (if it says 'starter code' then the markers may assume that it is unchanged).
    b. Includes you name and email.

You can edit the README with any text editor (including ARM Keil MDK). However, you can also edit it on the Git Hub system which can preview the use of markdown. Be careful not to make changes to both repositories (the one on your laptop and the one on Git Hub) at the same time. Use 'push' or 'pull' to synchronise first.

> There are extensive instructions on the use of Git on the QMPlus page (as well as elsewhere). The documentation on the QMPlus page focusses on the minimum you need for this module.

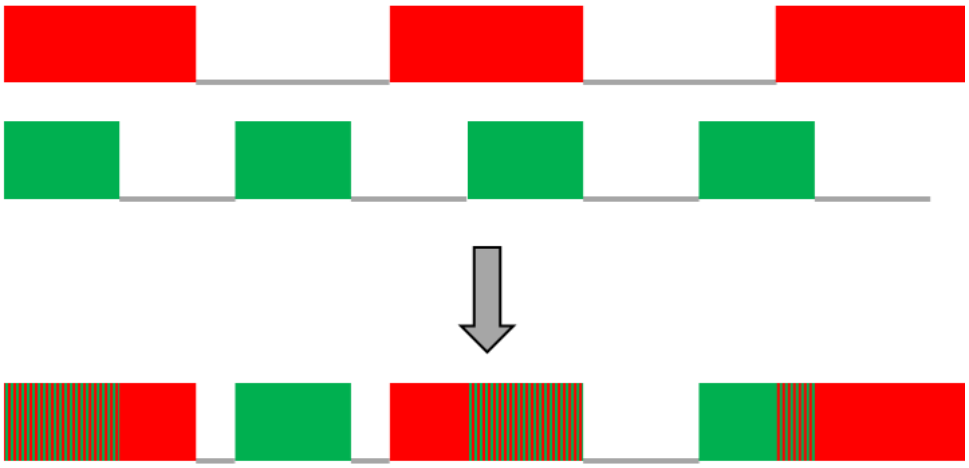
3 [*Advanced*] Multiple Tasks

The purpose of this section is to understand the potential advantages of using more than one task, each with its own state machine.

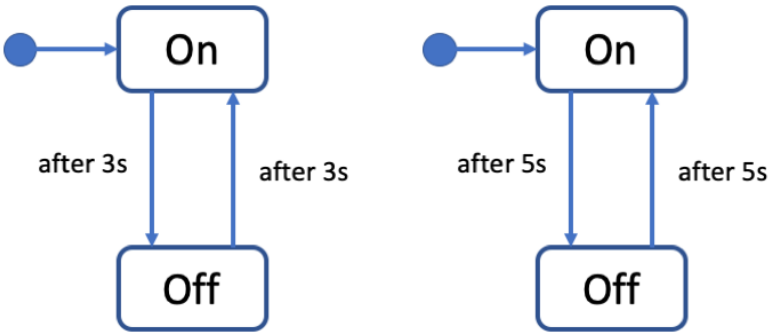
In the lecture (week 2) we looked at the following problem:

> *Two lights flash. The red light is on for 5 seconds and off for 5 seconds. The green light is on for 3 seconds and off for 3 seconds.*

The diagram below shows the initial behaviour: for 3 second both lights are on; then for 2 seconds just red; then neither for 1 second then just green for 3 seconds and so on. How long before the pattern repeats?



The suggested design for this system is to use two state machines. They both look similar:



3.1 Activity 4.1: Implement the Design with Two Tasks / State Machines

To do this, you need to

- Rename the function 'every10ms' and make a copy of it. Suitable names might be 'Task1Red' and 'Task2Green'.
- Give each copy separate state and counter variables: since these variables are global (why?) they need different names.
- Call both new functions from the main loop.

There is some duplication of code; you may consider how to reduce this and whether it is worth doing so.

3.2 Activity 4.2: Compare this with a One Task Design

If the system were implemented using a single task, only 4 states would be needed: Off, Red-Only, Green-Only, Red-Green.

Draw a state transition diagram with 4 states for this system and compare it with the two-task version above. Decide whether you want to implement it – it is obviously possible to do this.

3.3 Activity 4.3: Challenge

In the earlier part of this lab sheet, you have implemented a system that cycles through multiple colours using a single task.

Consider whether there is a similar behaviour than can readily be implemented using separate tasks. Try it out.

Note: you should push the code to the repository on GitHub. You may be wondering '*which version?*'. In general, we will avoid creating multiple divergent systems in a single lab exercise.