Implementing a time-triggered scheduler

1 Introduction

This lab is concerned with development of a time-triggered scheduler for the `LPC_2378_STK` development board. It uses what you've learned in `lab01` about interrupt handlers and timers to create a system based on a single interrupt – generated by each timer tick.

2 In the lab

1. Clone the repository `cm0605_lab01.git` and checkout the solution to last week's lab.

   ```
   $ git clone https://github.com/DavidKendall/cm0605_lab01.git
   $ git checkout P13
   ```

2. Start up EWARM and load the workspace `cm0605_lab01/workspace.eww`.

3. Connect a `LPC-2378-STK` board to a USB port on your computer.

4. Download and debug the project `lab01`. Run it and observe its behaviour. This is a solution to the lab exercise given in week 1. Notice that it provides a new enumeration type `timerIdentifier_t` and uses its values to identify which timer the `initTimer`, `startTimer`, and `stopTimer` functions will affect. See `timers.h` and `timers.c`. When you are convinced that you understand the interrupt service routine and timer configuration, move on. Ask your tutor if there's anything you're not clear about.

5. Now clone the repository `cm0605_lab03.git`. This provides a framework for a solution to the same problem as `lab01` but requires you to build a time-triggered implementation. The only files that you need to change in `lab03` are `main.c`, `scheduler.c` and `ttSchedConfig.h` Each file contains detailed comments explaining how the implementation should be developed. Add the code required to build a fully working system. Test it thoroughly.

6. Add a third task, `buttonsTask`, to your system (remember to modify `ttSchedConfig.h`). The `buttonsTask` should read the state of the buttons, and the flashing of the lights should be controlled by button presses. Initially, the lights should be off. Press `BUT_1` to start flashing; press `BUT_2` to stop flashing.

7. When you build your project, look carefully at the `Debug Log` and make a note of the number of bytes that are downloaded to flash. Now go to `Project->Options->C/C++ Compiler->Code` and change the processor mode to `Thumb`. Make sure that you select the option to `Generate interwork code`. Rebuild the project and note the number of bytes that are downloaded to flash this time. What is the percentage saving in image size of `Thumb` mode over `Arm` mode in this case? Go back and deselect the option to `Generate interwork code`. Rebuild the project. What happens this time? Why?