Interrupt Service Routines.
Configuring a timer interrupt.

1 Introduction

This lab is concerned with development of interrupt service routines for the `LPC_2378_STK` development board. It's concerned particularly with how to configure one of the on-chip timers to generate an interrupt and how to write a handler to service that interrupt. Interrupt handling is the foundation of developing operating systems software.

2 In the lab

1. Download the file workspace.zip into a suitable directory either on a pen drive or in your University workspace. I suggest you call the directory `EN572/labs/lab04`. Unzip `workspace.zip`.

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

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

4. Before starting on the work on ISRs, make sure that you understand the solution to `lab03` – it's in the project `lab03S`. Download and debug this project. Test the program. Make sure that you understand how the driver for the buttons has been written (see `buttons.h` and `buttons.c`). Note that you can change the rate at which the light flashes by using the joystick: up for faster, down for slower. The change in the rate of flashing occurs very slowly; be patient.

5. You can now move on to `lab04`. This project uses the liquid crystal display on the LPC2378-STK board. The display needs to be initialised. Add a call to `lcdInit()` at the end of `bspInit()` in `bsp.c`. You will need to include the header file for the lcd, `lcd.h`, at the top of `bsp.c`. Insert the appropriate directive immediately after `#include <buttons.h>`.

6. Now download and debug the project `lab04`. Run it and observe its behaviour. Try pressing button 1 a few times. It's yet another program

to flash the USB link LED. But this time the flashing is generated by an ISR.

7. As usual, the code is organised in two groups: `app` and `bsp`, each of which contain `include` and `src` groups. The `include` group contains the header files for its containing group. The `src` group contains the C source files for the its containing group.

8. Find the definition of `IRQ_Handler` (what's the easy way to do this in the IDE?). Notice that this definition replaces the default definition in `$TOOLKIT_DIR$arm\src\lib\arm\vectortrap.s` which is linked with `$TOOLKIT_DIR$arm\src\lib\arm\cstartup.s` and defines the initial behaviour of the board after reset. Read p.46 in The Insider's Guide to help your understanding of what's going on here.

9. Find the definition of `vicInit` and study how the Vectored Interrupt Controller (VIC) is initialised. Read p.82 in The Insider's Guide for more information about the VIC.

10. Read section 4.3 in The Insider's Guide for information about the timers. We are going to use TIMER0 in match mode, i.e. whenever the value of the timer counter is equal to the value in the match register, an interrupt will be raised and the value of the timer counter will be reset.

11. Find the definition of `initTimer0` and study the code carefully. Make sure that you can explain how the code works, based on your reading in the previous task.

12. Based on your reading of the code so far, explain how it is that `timer0ISR` is called whenever `TIMER0` raises an interrupt.

13. In order to test your understanding of what you've looked at so far. Develop new functions `initTimer1()` and `timer1ISR`, and use them to flash the USB CONNECT LED at the same time as the program is carrying on with its current behaviour. You should configure the timer so that the CONNECT LED flashes at a rate of 1Hz.

14. In `main.c` comment out the statement `__enable_interrupt()`. Rebuild, download and debug the project. Explain the behaviour that you observe.