SYSC3310 Lab 4 – Interrupts using GPIO

Fall 2018

Objectives:

- Programming of first interrupt service routine
- Using the GPIO interrupts
- Developing independence and problem-solving skills through an applied problem.

Equipment:

- MSP432 P401R LaunchPad Development Kit

Submission: Lab4b.c and Lab4c.c.

References and Reading Material

- Demo programs from the lectures: InputOutput.c, Demo-Interrupt-GPIO

Background Preparation

When using interrupts (as well as SYSTICK, later) there are complications with the included header files.

Before:

```
#include <msp.h>
```

Now:

```
#include "../inc/msp432p401r.h"
#include "../inc/CortexM.h"          // And add CortexM.c to your project
```

The relative pathname in the code above **(../inc/)** assumes that you have used this folder structure. It's time that we set up this folder structure, for use in the rest of the term.

Step 1: Create the following folder structure

```
SYSC3310
    Lab1
    Lab2
    …
    inc                    // Short for include. Will be shared by all future labs
```

Step 2: Download Valvano's Board Support Package from CU Learn, unzipping all the files into your new inc folder.

Step 3: [To be repeated each future lab] When any of the files in the **inc** folder are required, with your Keil project(1) use the **relative** pathname for your MSP432-related include files and (2) add the .c files to your project

Part A: Exploring an Interrupt Driven Program

In the lectures notes, you have an example of an interrupt program that uses the GPIO pins of the two Launchpad switches (**Demo-Interrupt-GPIO**). As a start, you should create a project called **Lab4a** and copy-paste this code into its main program, and run the program to see that it works.

- Push on SW1 to toggle the blue portion of the RGB LED
- Push on SW2 to toggle the red LED

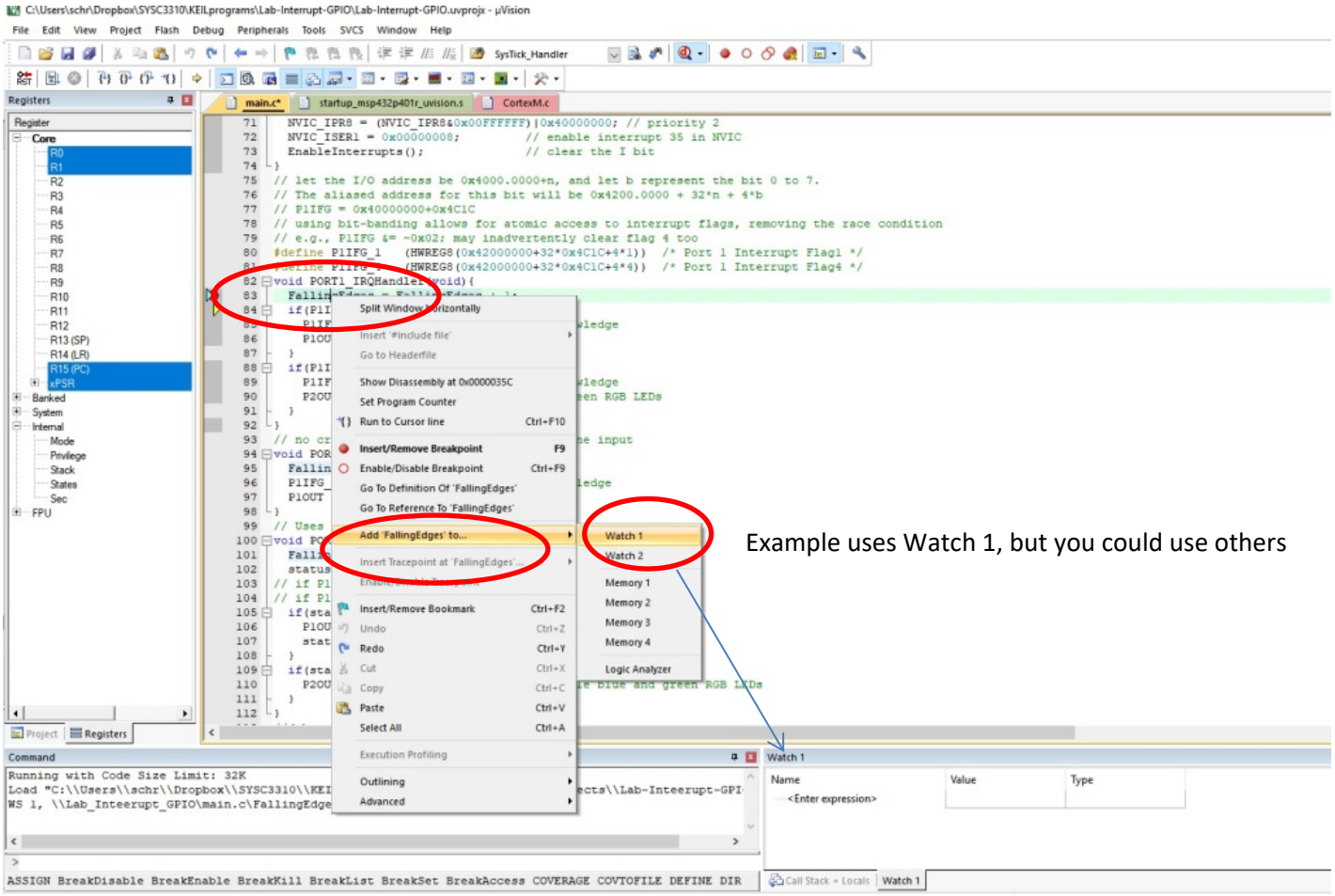
As you execute this demonstration program, it is suggested firstly that you use the debugger's breakpoint to verify the execution of the ISR, because debugging interrupt-driven programs usually involves three kinds of bugs:

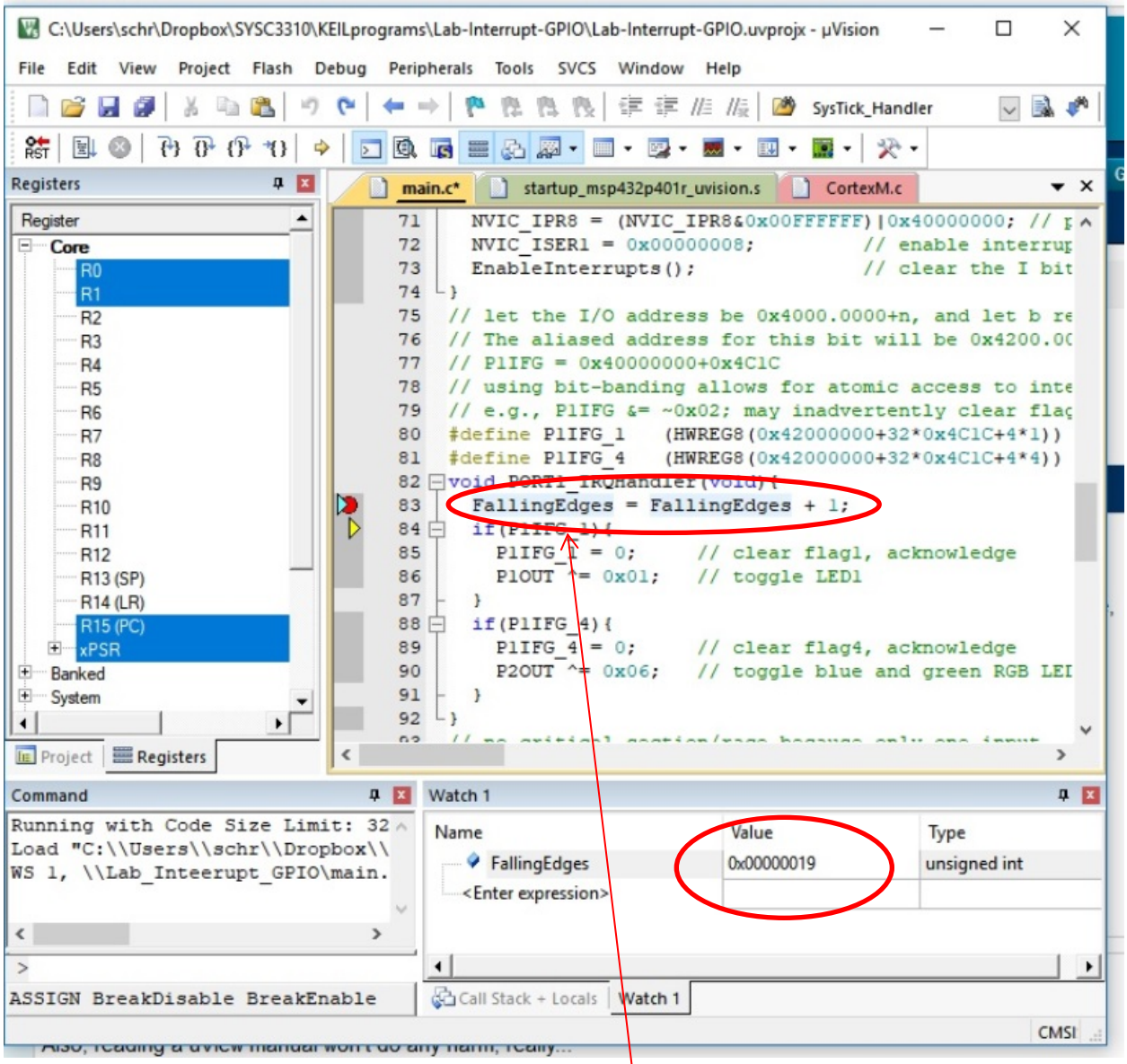
(1) The interrupts don't actually happen because the hardware is not configured properly
(2) The ISR doesn't actually run because the ISR is not installed correctly
(3) Your own kind of regular software bugs.

Consequently as a first step whenever developing an interrupt-driven program, it is common to put a breakpoint on the first instruction of (each of) the ISRs and just see if the program ever gets there. If you never reach the breakpoint, you know that your hardware configuration or your ISR installation is faulty or interrupts are not probably enabled. If you do reach the breakpoint, you know that hardware is working and all you have to concentrate on is your own programming logic.

Secondly, once your program is running, you are advised to further inspect the ISR's global variable called `fallingEdges`. There are special steps needed to view the global variables used so prevalently in ISRS within KEIL (as opposed to local variables). The reason is that a debugger usually shows variables within their scope (e.g. locally, within their functions); yet, global variables are outside of the scope of all functions. The solution is not to use only a breakpoint, but to instead **add a Watch.** Please see the notes below. (Discussion Forum: http://www.keil.com/forum/21557/where-to-see-global-variables/)

Practice these steps now so that you know what to do in later work, when you have troubles.


Example uses Watch 1, but you could use others



WATCHed value will be
updated at each breakpoint

Part B: Writing your first Interrupt-Driven Program

You will create an interrupt-driven version of the demonstration GPIO program that you ran in last week's lab (Posted as **InputOutput.c**) In a nutshell, you have to merge the logic of Lab3's program with the interrupt structure of the demonstration interrupt program that you just ran in **Part A**.

- When SW1 is pressed, the RGB LED should be BLUE
- When SW2 is pressed, the RGB LED should be RED
- The red LED should not be used at all.
- Note: You cannot have a situation when the RGB LED is off, except for upon startup. There is no interrupt to say that a button is NOT pressed.
- Note: You cannot have a situation when "both switches" are pushed, because the human hand cannot make them exactly simultaneous and two separate interrupts will be generated and processed, with the colour of the LED being set to the last interrupt (slowest finger)
- Note: In this exercise, you don't have to account for switch debouncing.

Part C: Problem Solving with an Applied Problem

- *This section is written in the style of a sample exam question.*
- *You are to use a project called **Lab3c** for your solution.*

You have been hired to build a traffic light using the MSP432P401r Launchpad. The traffic light will be implemented using the RGB LED (and optionally the pedestrian walk-light using the red LED). The pedestrian button will be implemented using the SW1 switch and an arrival car sensor (dug into the pavement under the stop line) as the SW2 switch.

The system will begin with the light GREEN (so cars can drive through). It will cycle through a regular schedule of GREEN (10 seconds) – YELLOW (5 seconds) – RED (5 seconds). If a pedestrian presses the pedestrian button while the light is GREEN, the light should immediately turn YELLOW (so that the pedestrian can cross when the light safely turns red) If a car arrives while the light is RED, the light should turn GREEN no more than 2 seconds later (so that the pedestrian has time to finish crossing)

Optionally, the pedestrian light should turn on when the light is red (and yellow), and off when it is green.

There are many small tasks involved in this problem (how to make the RGB LED turn YELLOW, for instance) that are being left for you to work out, but the following paragraph lays out an underlying design for the program.

Your program will consist of two threads-of-control: the main background thread and the interrupt foreground thread. Firstly, this means that the loop in the `main`() function is not empty. Secondly, part

of your solution will lie in the global variables through which the two threads communicate with each other. Typically, one thread will write a global variable that is read by the other thread, thereby effectively sending a message. In the system below, it is suggested that the `timeRemaining` (0 to 10 seconds) be used as one of those global variables. For example, when the pedestrian button is pushed, the `timeRemaining` is set to 2, if it is not already less than two.

Global variables
(add more as needed)

uint8_t colour = GREEN // GREEN, YELLOW, RED

uint32_t timeRemaining= 10

```
int main() {

// Initialisation Ritual

while (1) {

  // Logic for cycling the LED
// through the colour sequence
// according to given schedule
// using software time delays
```

```
void PORT1_IRQHandler () {

// Acknowledge interrupt

 if (???)

   timeRemaining = 2;

else if (???)

   colour = YELLOW
```

For the program in main(), the following is suggested:

Much of the timing in the problem is based on n-second delays. In computers, one second is a long time to delay. It is suggested that you use the following function that delays for approximately 1 millisecond

```
void delay1m (int number) {
      uint32_t i,j;
      for (i=0; i<number;i++) {
            for (j=0;j<900;j++);
      }
}
```

To delay for n-seconds, you would call this function within a loop:

```
// Assuming nSeconds is set to your intended delay
      while (nSeconds > 0) {
            delay1m (1000); // 1000 ms = 1 second
            nSeconds--;
      }
```

Marking Scheme: Total Marks of 15

Part A: No submission needed. For learning only.

Part B:

Demonstration: (0: Not working; 1: Working; 2: Working & ISR logic; 3: Working & PRIMASK)

- RGB LED changes as described; Red LED is not used at all; after startup, RGB LED is never off (after initial push)
    - Each student must be able to explain the logic in the ISR. For full marks, they must be able to explain patterns used to install the ISR (e.g. PRIMASK)

Inspection: 1 mark each, for a total of 3

1. Code implements the requirements regarding the LED
2. All statements must be in DRA mode
3. Overall Style

Part C:

Demonstration: 3 marks

- 0: No working demonstration
- 1: Traffic light cycles as described without user input
- 2: Traffic light cycles and responds to user inputs
- 3: Traffic light cycles, responds to user inputs and also uses the red LED for pedestrian light

Inspection: 1 mark each (0, ½ or 1) Total of 6 marks

1. Structure of program follows described architecture (main loop for cycle, and ISR for sensors)
2. Main loop: Simplified logic for (1) sequence of colours and (2) delays. Zero if any reference to the switches.
3. ISRs: Simple if-structure. Zero if any reference to the red LED (pedestrian).
4. Logic: Reaction to switches must only be taken if relevant to current colour.
5. Readability of a more complex program: Use of some subroutines to organize code (e.g. delays and setting of colours) as well as use of #defines for clarity
6. Overall Style

Overall marks for style (To be used for all labs in this course):

- Comments, indentation, and well-named functions and variables
- Removal of all extra code (no commented out sections, no unused code leftover from some other example)