SYSC3310 Lab 3 – GPIO

Fall 2018

Objectives:

- Programming of first peripherals – the LEDs and switches on the Launchpad
- Understanding the difference between GPIO using DRA (Direct Register Access) versus SRA (Structured Register Access)
- Development of a debouncing algorithm for switches

Equipment:

- MSP432 P401R LaunchPad Development Kit

Submission: Lab3a.c and Lab3c.c as well as **CULearn Quiz called Lab 3**

References and Reading Material

- Lab 1 – For reference on how to create a project and configure the debugger
- BlinkLED.c - Valvano's example used in Lab 1
- InputOutput.c

Part A: Exploring the First Program

- *Make a project called Lab3a*

In Lab 1, you ran a project called **BlinkLED** that turned the red LED on the Launchpad on and off. We are going to begin by returning to that example and studying it in depth, now that we've had some time for some lectures. The code is copied below, for easy reference in this lab

```
#include "msp.h"

int main(void)
{
    volatile uint32_t i;

    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;  // Stop watchdog timer

    // The following code toggles P1.0 port
    P1->DIR |= BIT0;                             // Configure P1.0 as output

    while(1)
    {
        P1->OUT ^= BIT0;                         // Toggle P1.0
        for(i=10000; i>0; i--);                  // Delay
    }
}
```

1. There is a CULearn quiz posted on the course webpage, called **Lab3 Quiz**. It has a number of questions about our code. Each partner must complete their own quiz (each will receive a separate mark)

2. There are two different styles of writing code that uses the GPIO.

- **Direct Register Access (DRA**) – Uses the actual names of the registers – and the bits within those registers – that are defined in the Technical Reference Manual (TRM)
    - We will focus on this style because it is consistent with the TRM, and because it will re-inforce your understanding of the underlying hardware that we are using.

    Example : P0SEL0 = 0x00;

- **Structured Register Access (SRA)** - Uses a higher-level abstraction of registers, grouping related registers into a C-struct.
    - It is intended to be more user-friendly as well as portable between multiple versions of MCUs within the same family.

    Example: P0->SEL0 = 0x00;

This topic has been covered in class and you should use those lectures notes. In this lab, you must demonstrate your understanding of the equivalence of the two styles. **Specifically:**

- BlinkLED.c has been written in the SRA style.
- You must re-write BlinkLED.c in the DRA style.

*Tip: Don't lose your work in Lab1. Make a copy of Lab1, called **Lab3a**. Call the file **Lab3a.c***
Tip: Do one line at a time, or at least until you get the hang of it. Re-build the project and run it to see if it still works.

Part B: Exercise in the Debugger

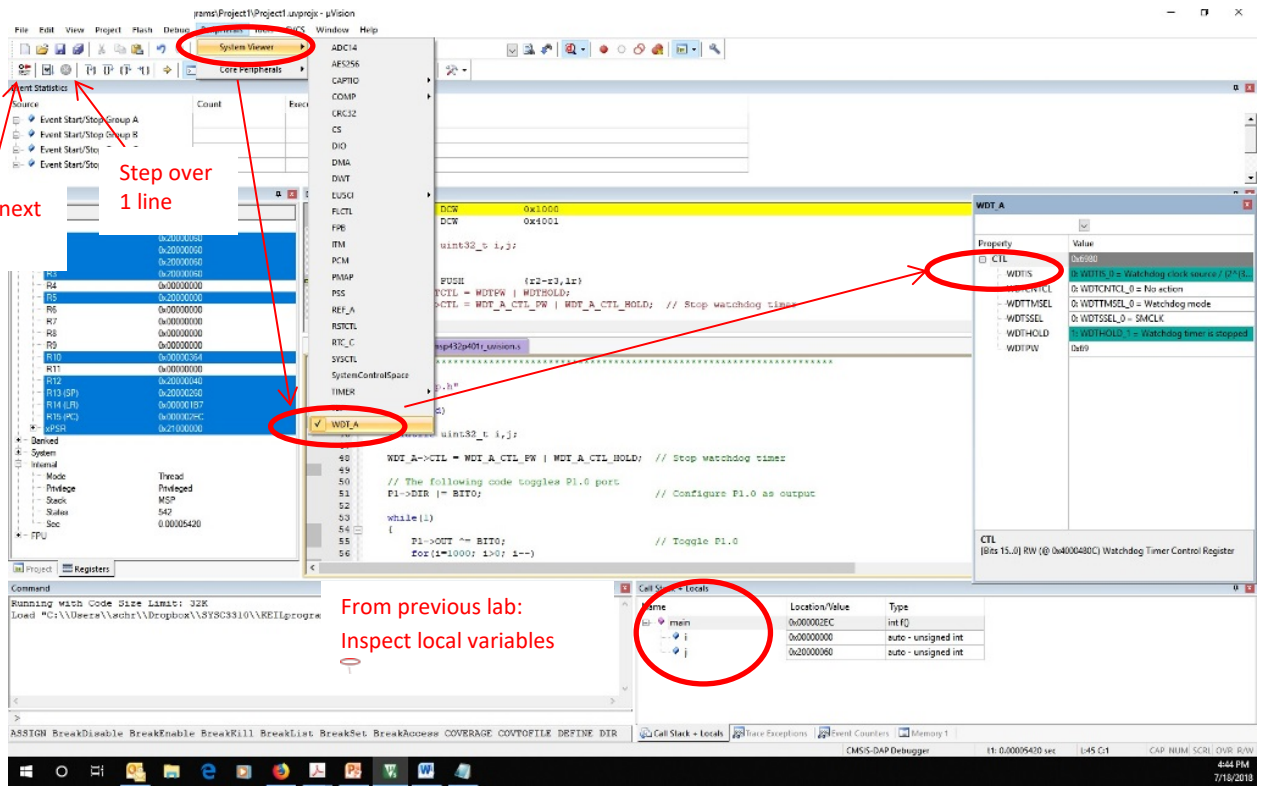
In a previous lab, we used the debugger to inspect the current contents of a **variable** while the program is running. Now, with GPIO, we want to be able to inspect our **port registers**.

1. Open up the BlinkLED project (Either version will work although the instructions use the original version from Lab 1 that uses SRA).
2. Start the debugger but do not run the program
3. Insert breakpoints in the code. Use the two images below to help guide you.

   Tip: A breakpoint at Line n *stops the program after the execution of Line* n-1 *and before the execution of Line* n
   Tip: You set a breakpoint by clicking to the left of the line of code (in the grey area)

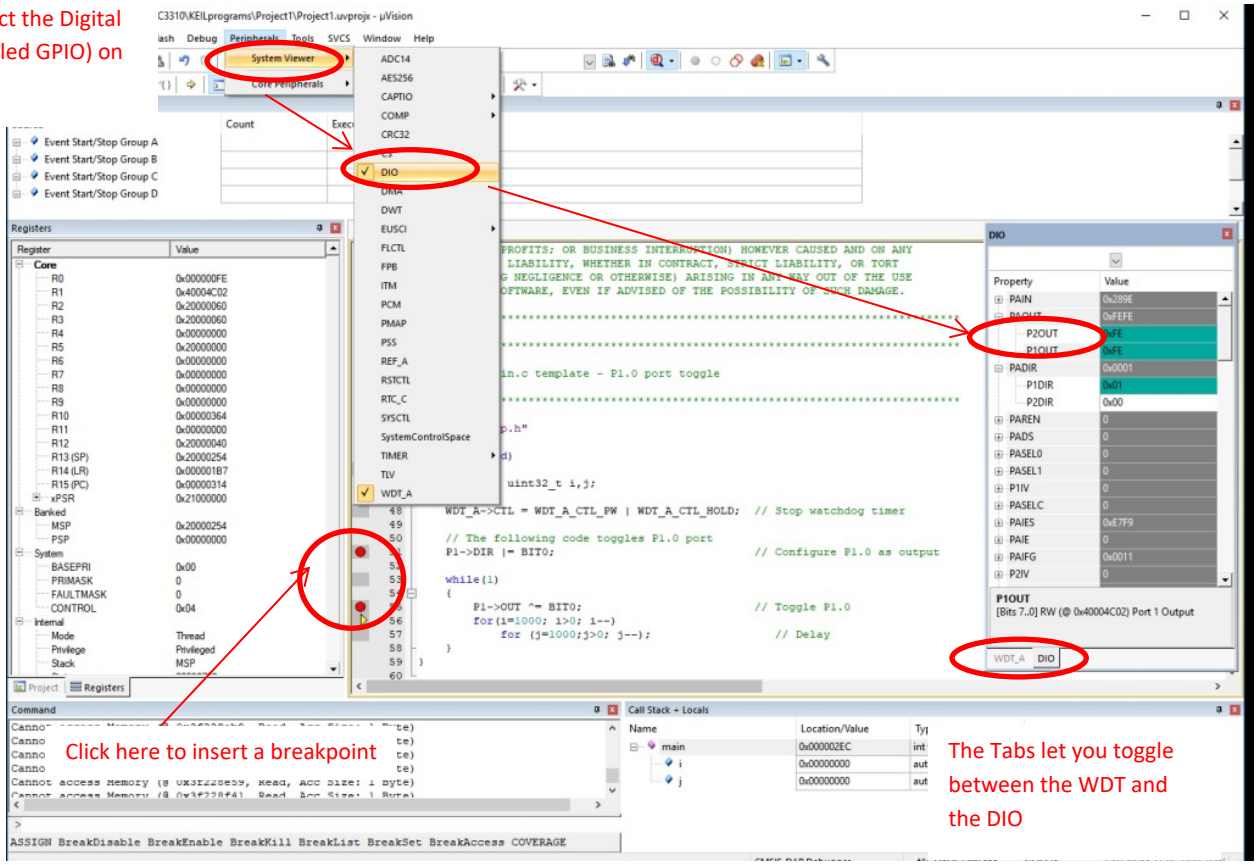
   a. Put a first breakpoint on the line of code: `P1->DIR |= BIT0` (or your new version). It must be AFTER the line of code that disables the watchdog timer.
   b. Put a second breakpoint on the line of code: `P1->OUT ^= BIT 0;`
   c. Select **View->Call Stack Window** (To trace the values of local variables)
   d. Select **Peripherals->System Viewer -> WDT_A.** (To trace the values of watchdog timer)
      - Notice that the Watchdog timer is stopped
   e. Select **Peripheral->System Viewer -> DIO** (To trace the values of Digital I/O pins)
      - **Question**: What two values do you expect P1OUT take on?
   f. RUN the program. It will execute … until it hits the first breakpoint. Your first breakpoint is before **P1->DIR** is set. Observe the value of **P1DIR** before. Step over 1 line. Observe the value of **P1DIR** after. Do you understand the change?
   g. Run the program … until the second breakpoint, i.e. before P1->OUT is set. Observe the value of **P1OUT** before. Step over 1 line. Observe the value of **P1OUT** after.
   h. Repeat the process. You will cycle through the while-loop. You should see the value of **P1OUT** alternate after each breakpoint. Notice also that the actual red LED on the Launchpad is also changing in sync with this port's value (because the LED is connected to this port's pin)
   i. Last exercise: Select **Peripherals->System Viewer -> SYSCTL**
      - What is the FLASH_SIZE? In Hexadecimal? In decimal?

NEW: Inspect the WDT
"inside" the MCU



j.

NEW: Inspect the Digital I/O (also called GPIO) on the MCU



Part C: Switch Debouncing

- *Make a project called Lab3c*

There is a program – discussed in class – that is posted with the Lectures on the course webpage, called **InputOutput.c**. It is a sample program from Valvano's textbook that uses both LEDs and both switches on the LCD. You will begin by running this program as-is and then make some small changes to it.

1. Create a new project called **Lab3c**. Copy-paste the code into this project's .c file (called **Lab3c.c**). Run the program and enjoy playing with it. You may have to read the code to figure out what it's supposed to do.
2. The program uses the switches, but it does not handle any switch bounces. Switches are made up of slender strips of metal that make contact when pressed, allowing current to flow. Because they are so slender, when you lift your finger off the button, the strip of metal may physically bounce – rising up, and then down, making contact again – looking like another button press. You meant to press once, but the program thinks it was pressed twice, or thrice … This, of course, is a bug.

   We will begin by first (trying to) observe bounces. You will add counters to keep track of the number of times that the program detects a switch-press: `nSW1`, `nSW2` and `nBothSws`. Each time the program detects a particular switch being pressed, the respective counter should be incremented. Run the program while you methodically test out the switches.

   Example: First, focus only on **SW1**. Run your program and hit the **SW1** switch 4 times.

   After running your test, inspect the contents of the counters. Do their values match the number of presses that you made?

   Example: Look at the contents of `nSW1`. If you hit **SW1** 4 times, `nSW1` should equal 4. If it is more than 4, your switch bounced. (If it is less, you have a different software bug)

3. Hopefully you observed some switch bounces and you're motivated to de-bouncing your software. The simplest form of de-bouncing is to simply wait a wee-bit-of-time – enough to let the bouncing settle down – before polling the switch again. Add a delay-loop between polls of the port. Start with a tiny delay, and keep increasing it slowly until you don't register any bounces anymore.
   - If your delay is too big, this is also wrong because you might miss an actual button press.

Marking Scheme: (PLUS THE CULEARN QUIZ)

Part A:

Inspection: 1 mark each

1. All statements involving GPIO registers have been converted to DRA mode
2. Overall Style

Part B: No submission required.

Part C:

Demonstration:

- Use of debugger to observe switch bounces
- Ability to identify when a bounces has occurred.

Inspection: 1 mark each

1. Counters have been implemented to observe bounces
2. Switch de-bouncing algorithm has been implemented.
3. 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)