Lab #5 Motion Detection Alarm (Interrupt Processing)
Summer 2021

Lab Time: Tuesday 7/20, 8:00am – 10:30am

Demo Due: The beginning of lab on Tuesday 7/27

Submission Due: Tuesday 7/27, 11:59pm

Introduction

The goal of this lab is for you to exercise programming the interrupt system of Tiva C. The learning objectives are:

- **Gain programming experience with the interrupt system of Tiva C**
- Learn how to write code to configure interrupt processing
- Learn how to write code for interrupt service routine (ISR)
- Understand the power saving features of Tiva C and how interrupt may wake up a microcontroller in an energy saving mode

Notes:

- We will study the interrupt system of Cortex-M4 later in the class.

Prelab

- Read again **Ch. 15** "GPIO" of the TivaWare User Guide. *Pay attention to the library functions related to interrupts*.
- Read **Section 2.7** "Power Management", **Section 3.1.2** "Nested Vectored Interrupt Controller (NVIC)", and **Section 5.2.6** "System Control" of the Tiva C Datasheet.
- Read **Ch. 18** "Interrupt Controller (NVIC)" of the TivaWare User Guide.

Background

In Lab 4 "Motion Detection Alarm", you have used a programming technique called *polling I/O* (or *I/O polling*) to poll the status of the push buttons and the PIR motion sensor.

In this lab, you will use another technique called *interrupt-driven I/O*. This change will significantly improve the execution efficiency of the system, which leads to better energy efficiency and potentially better system response time.

Recall the three software tasks we have developed in Lab 4:

- **Task 1: Play the buzzer**. Turn on and off the buzzer sound to make the alarm sound.
- **Task 2: Check the pushbuttons**. Check the pushbuttons every 10 ms for user inputs (with software debouncing).
- **Task 3: Check the PIR motion sensor**. Check the motion every periodically to detect human motion.

All three tasks were implemented as callback functions. **In this lab, Tasks 2 and Task 3 will be implemented as ISRs** (*Interrupt Service Routine*s, also called *Interrupt Handlers*)**.**

Part 1:

One member (with the help of partner): In your working directory, create a new folder named "Lab5". Download "lab5_main.c" from Lab 5 attachments to this folder. Copy "buzzer_asm.asm", "buzzer.c", "buzzer.h", "motion_asm.asm", "motion.c" and "motion.h" from your Lab 4 folder to this folder. Switch to CCS, create a new CCS project "Lab5" with an external reference to the "Lab5" folder. Add the #include directory paths and the library file paths for TivaWare and Util libraries. Switch to GitHub Desktop, commit the changes to your local repository, then push the changes to your group's GitHub repository. Check on https://github.com to make sure the changes have been pushed into your group's repository.

Another member: Run GitHub Desktop. Pull the new changes from GitHub to your local repository. Check that Lab5/ and the new files appear in your working directory. Switch to CCS, create a new CCS project "Lab5" with an external reference to the "Lab5" folder. Remember to add the #include directory paths and the library file paths for TivaWare and Util libraries.

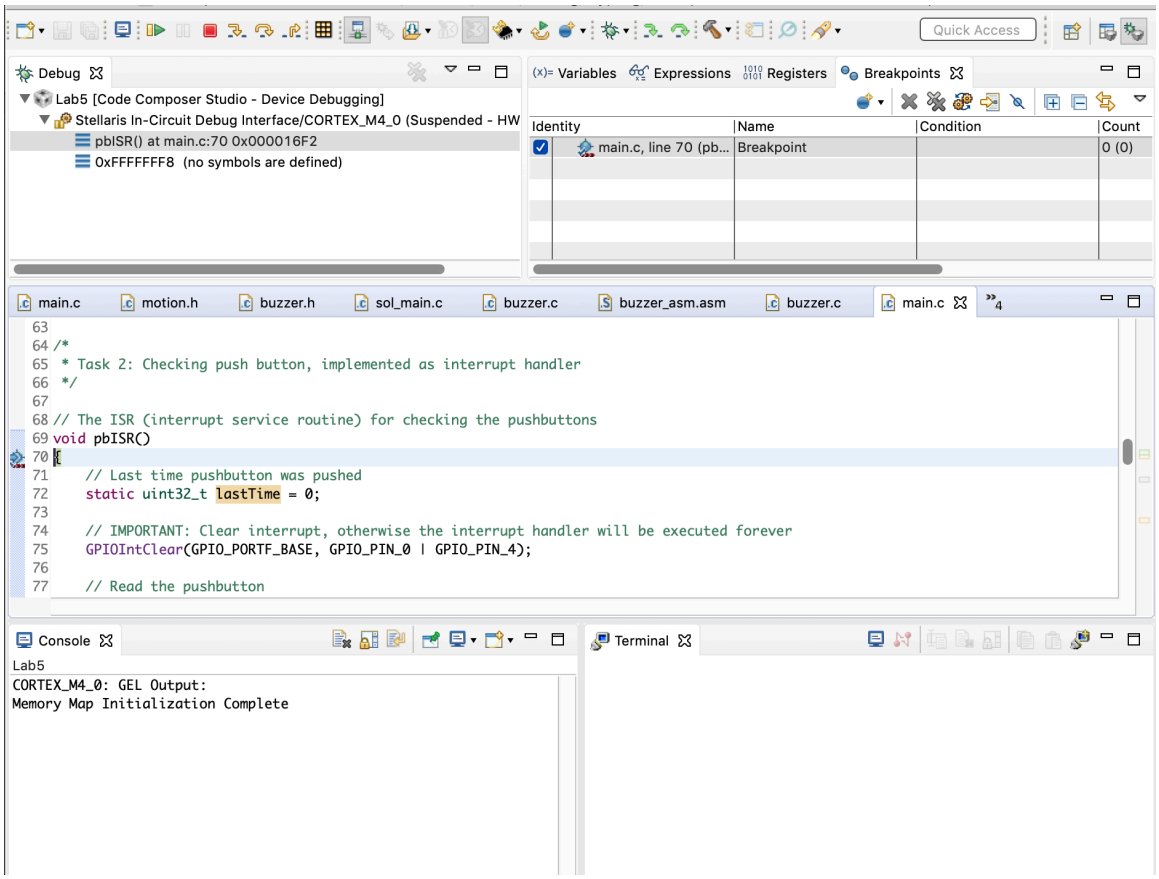
Attach the LaunchPad with the Grove base boosterpack, attach the buzzer to jumper **J17** and attach the PIR motion sensor to jumper **J16**, as you did in Lab 4. Double check that you have used the correct jumpers.

Build and debug the program. The program will behave as follows. It works exactly as the starter code in Lab 4, but the code structure is very different.

- Initially, the LED is turned on in green.
- Push the SW1 button, the buzzer will start beeping periodically, and the LED is turned on in red.
- Push the SW2 button, the buzzer will stop beeping, and the LED is turned on in green.

Important: Set a breakpoint on the entry of the ISR for the pushbuttons, as shown in the following figure. Push a pushbutton, the ISR will be executed, and the debugger will stop the program right there. You may have to keep holding the pushbutton until the pbRead() function is executed. Trace the program execution step by step until it exits the interrupt

handler. In this lab, you may need to use this debugging technique to help debug your program.



Part 2: Interrupts from Motion Detection

First, read through the provided "lab5_main.c" and understand the function **pbISR**() and another function **intrInit**(). You will notice that Task 2, checking the pushbuttons, is now implemented as an ISR. The general idea is to enable interrupt on the I/O port and pins used by SW1 and SW2 (Port C, pins 0 and 4). When a user pushes SW1 or SW2, an interrupt signal will be sent to Tiva C. The ISR is then executed to check which pushbutton has been pushed, and then it activates or de-activates the alarming system.

Note that in the while-loop in the main() function, the program executes an "WFI" (wait for interrupt) assembly instruction. This instruction makes the microcontroller enter a sleep mode and stay in the mode until an interrupt happens. Read "ARMv7-M Architecture Reference Manual", page A7-561, for a detailed explanation of the instruction.

Revise "main.c" such that the microcontroller will also wake up on motion detection. You will need to 1) enable interrupt on the I/O port and pin used by the motion sensor, and 2) add another interrupt handler for the motion sensor. Your system should work as follows:

- If any human motion is detected, the program should turn on the on-board LED in red color.
- If there is no human motion, the program should turn on the on-board LED in green color.

In this part, the LED is not affected by the pushbuttons. The purpose of this part is to make sure the ISR for the PIR motion sensor works.

Important: The C "**`volatile`**" qualifier should be used in the declaration of any global variables whose value will be changed by an ISR. For example:

```
volatile OnOff_t sysState = On;
```

Important: Do NOT call the uprintf() function inside an ISR. A general rule of writing an ISR is NOT to include any time-consuming code in it. A call to the uprintf() function can be time-consuming.

Part 3: Motion Detection Alarm by Interrupts

Revise the code in main.c such that your system works as follows (exactly as in Lab 4, Part 3):

1. Initially, the system is deactivated (sysState = Off).
2. If a user pushes SW1, the system should be activated (sysState = On). If a user pushes the SW2 button, the system will be de-activated.
3. When the system is deactivated, the LED should be turned off, and the buzzer should not make any sound.
4. When the system is activated and there is NO human motion around, the LED should be turned on and stay in the **green** color. The buzzer should not make any sound.
5. When the system is activated and there is any human motion around, the LED

should be turned on in the **red** color and the buzzer should beep (as in Part 1).

6. ***Optional***: You may notice that the alarm is on and off, even if there is consistent human motion. That is caused by the BISS0001 IC output pattern (see the BISS0001 datasheet on BlackBoard). Revise your program so that the alarm stays on whenever there is any human motion detected in the past three seconds. Hint: Your code may memorize the last time that human motion is detected and compare that time with the current time to decide if the alarm should be turned on or off.

Demo your system to your TA. Then, show your source code to your TA and explain how it works.

Note: Each of you may have to answer questions from your TA to test your understanding of the working details. Your answer may affect your lab grade (not your partner's).

If you can demo Part 3, you don't have to demo Part 2.

Lab Submission

Your group should write a lab report that includes the following.

1. Describe the responsibility of each person in your group, and estimate the breakdown of contribution in percentage (e.g. 50% and 50% for a two-person group). Note: If one contributes 40% or higher in a two-person group, he or she may receive the same grade as the group gets.
2. Describe the changes you made in "main.c" to make Part 3 work.
3. How much flash memory does your program take? How much SRAM memory?
4. Estimate how many hours in total you and your partners have worked on this lab.

Make sure that you have pushed the latest version of your code into GitHub. **This step is required for you to get the grade.**

Submit the following items on BlackBoard under Lab 5 assignment:

- The revised main.c
- The lab report