Lab #7 Light and Sound Control (PWM)
ECE 266, Summer 2021

Note: This lab is NOT required. The contents will not be covered in the class and final.

Introduction

The goal of this lab is for you to learn the PWM hardware function of the Tiva C. The learning objectives are:

- To gain programming experience with Tiva C's general-purpose timer to generate PWM waveform, using TivaWare library functions for general-purpose timers.
- To learn how to use PWM waveforms to control the brightness of a light and the loudness of a speaker (the buzzer).
- Exercise embedded systems programming with I/O register access (bonus part) – see the notes.

Background

PWM (Pulse Width Modulation) is a commonly used hardware function in embedded systems for controlling other devices. In Tiva C, this hardware function is implemented in on-chip general-purpose timers[1]. In this lab, PWM is used to control the brightness of the on-chip LEDs and the loudness of a buzzer.

Before you start on Part 2, read through the whole lab manual, and then discuss with your partner about the system development approach, identify things to do, and assign work to each person.

Prelab

It is important that you do the prelab. **Do not skip the prelab**.

- Read Textbook, Section 15.3, to understand the concepts of PWM, PWM period, PWM pulse width, and PWM duty cycle.
- Read Tiva C datasheet, pages 704-709 (to the end of Sec. 13.3.2 introduction).
- Read Tiva C datasheet, **Section 11.3.2.5** (PWM Mode) on **pages 716-718**, and **Section 11.4.5** on pages 724-725.
- Read quickly **Ch. 30 "Timer"** of "TivaWare Peripheral Driver Library User Guide".
- Read carefully the descriptions of the following functions in Ch. 30 "Timer" of

[1] There is another, dedicated PWM unit in Tiva C, which should not be confused with this one.

the TivaWare User Guide. You have used some of those questions in Lab 7.
  - TimerConfigure(): To configure the timers to work in the PWM mode.
  - TimerLoadSet(): To set the PWM period.
- TimerMatchSet(): To set the PWM pulse width (if PWM output inversion is enabled).
- TimerEnable(): To enable the timer.
- Read carefully the descriptions of the following functions in Ch. 15 “GPIO” of the TivaWare User Guide. *You will need to use those two functions*.
  - GPIOPinTypeTimer(): To configure a GPIO pin as compatible with timer’s input/output.
  - GPIOPinConfigure(). Note: To route a pin to a timer. There are only limited choices of the routing.

Part 1: Preparation

In your working directory, create a new folder named “Lab7”. Download “lab7_main.c”, “pwmled.c”, “pwmled_asm.asm”, “pwmled.h” from the attachment into the folder. Switch to CCS, create a new CCS project “Lab7” with an external reference to the “Lab7” folder. Remember to add the #include directory paths and the library file paths for TivaWare and Util libraries.

Attach the LaunchPad to the Grove base. Build and debug the project. The on-board LED will start to blink. Its brightness and color will change periodically and gradually.

Study the source code in “pwmled.c”, “pwmled_asm.asm” and “pwmled.h” to understand how the code initializes PWM and set PWM parameters for the three sub-LEDs of Tiva C.

Part 2: Light Control

You may have noticed in previous labs that the on-board LED is very bright, so it is desirable to control its brightness. The starter code has used PWM to vary the brightness. In this part, you will enhance the system so that a user may have another level of control of the brightness.

Attach a rotary angle sensor to Jumper **J5** as you did in Lab 6. Copy your “ras.c”, “ras_asm.asm”, and “ras.h” from Lab6 to Lab7 folder. Revise lab7_main.c to add a new feature as follows:

- The on-board LED should fluctuate in the same pattern as before.
- The user can use the knob to control the maximum brightness the on-board LED.
- If the knob is turned all the way to the left (counterclockwise far end), the on-board LED should be fully dark. If it is turned all the way to the right (clockwise far end), it should be as bright as before.
- The brightness should change gradually when the knob is turned.

Part 3: Sound Control

In this part, you will use PWM to control the loudness of the buzzer.

Attach a buzzer to Jumper **J17** as you did in Lab 4. Create new files named "pwmbuzzer.c", "pwmbuzzer_asm.asm" and "pwmbuzzer.h". Write a function for initializing PWM for the buzzer in C and put it in "pwmbuzzer.c". Write a function to set PWM parameters for the buzzer in assembly and put it in "pwmbuzzer_asm.asm". Put the prototypes of those two functions in pwmbuzzer.h.

Attach a second rotary angle sensor to Jumper **J6** as you did in Lab 6.

Then revise lab7_main.c to add the following features to the system:

- The buzzer keeps buzzing continuously as in Lab 4, but now with revised time parameters. It should be turned on for 200 ms and off for 2800 ms, and the pattern should repeat.
- The user may use the first RAS to control both the LED brightness and the buzzer's volume. If the knob is turned all the way to the left (counterclockwise far end), the buzzer should be fully silent. If it is turned all the way to the right (clockwise far end), it should be loud (you may use 25% maximum duty cycle ratio). The volume should change gradually when the knob is turned.
- The user may use the second RAS to control the pitch of the buzzer sound. If the knob is turned all the way to the left (counterclockwise far end), the buzzer should make a sound of frequency 261.63 Hz (which is music note C4). If the knob is turned all the way to the right, the buzzer should make a sound of frequency 523.25 Hz (which is music note C5). The pitch should change gradually when the knob is turned.
- *Note: If you and your partner work separately, each of you may hot-swap your knob between the two ports to control one of the two features at a time.*

***Application Notes (Read Carefully)*:**

- The PWM duty cycle (ratio) decides the volume of a sound. The duty cycle is the ratio of **PWM pulse width** over **PWM period**. Your program can set those two parameters in clock cycles.
- The PWM frequency decides the pitch of a sound, which is the reciprocal of the **PWM period**. Your program can set the PWM period in clock cycles.

Given the CPU frequency of 50 MHz, the PWM period should be (50,000,000 / *f*) clock cycles to produce a sound of frequency *f*. Example: To play music note C4, the PWM frequency should be 261.63. Therefore, the PWM period should be set as 50,000,000 / 261.63 = 191,110 clock cycles. To play the note at certain volume, the PWM pulse width should be set as a ratio of the PWM period, e.g. 30% * 191,110 = 57,333.

For more information about frequency and pitch level, you may read this Wikipedia link: https://en.wikipedia.org/wiki/Pitch_(music)#Pitch_and_frequency

***Programming Notes*:**

- The “lab4_main.c” in Lab 4 starter code has a callback function to play the buzzer in an on/off pattern. Reuse the callback function in this lab.
- Use “pwmled.c”, “pwmled_asm.asm”, and “pwmled.h” as the template codes for “pwmbuzzer.c”, “pwmbuzzer_asm.asm”, and “pwmbuzzer.h”.
- **Important**: You should use TimerControlLevel() function to enable PWM output inversion.
- Use TimerLoadSet() function to set the PWM period in clock cycles.
- Use TimerMatchSet() to set the PWM pulse width in clock cycles.