CPE 325: Intro to Embedded Computer System

Lab07

PWM and LED Hardware Brightness Lab

Submitted by: Gianna Foti

Date of Experiment:_____14 October 2024_____

Report Deadline:_____15 October 2024_____

Lab Section: _____CPE353-02_____

Demonstration Deadline: _____15 October 2024_____

Introduction

In this lab, we focused on using Pulse Width Modulation (PWM) to control the brightness of an LED and the Watchdog Timer to manage periodic tasks. The main task involved configuring Timer A to generate a PWM signal that adjusts LED2 brightness across nine discrete levels (0% to 100%) with no noticeable flickering. The program starts with the LED at 50% brightness, with switches SW1 and SW2 used to incrementally increase or decrease the brightness. To ensure smooth transitions, debounce handling was implemented for both switches.In addition, we configured Timer B to produce a 2 kHz sound using a buzzer. The sound toggled on and off every second, controlled by the Watchdog Timer's ISR, while the microcontroller remained in sleep mode between actions to conserve power. For an optional bonus task, we modified the program so that pressing both SW1 and SW2 simultaneously triggered LED1 to blink at a frequency of approximately 0.17 Hz. This blink pattern was maintained until the buttons were released, after which the system returned to the initial brightness control mode. This lab emphasized using PWM for precise LED control and implementing sleep modes for power-efficient microcontroller operation.

Theory Topics

1. Watchdog Timer: what is it, how can it be used, different modes, etc.). Give an Example of where each mode could be utilized
    a. The Watchdog Timer (WDT) in the MSP430 microcontroller is a safeguard designed to reset the system if a software failure occurs. It can prevent the system from hanging by resetting the microcontroller when the software fails to reset the timer within a specified interval. Alternatively, the WDT can be configured in interval mode, where instead of resetting the system, it generates periodic interrupts at specified intervals. This mode is commonly used for regular tasks, such as blinking an LED or executing periodic sensor readings, allowing the system to perform small operations without needing constant CPU intervention. In this configuration, the WDT triggers an Interrupt Service Routine (ISR) at regular intervals, enabling timed actions like toggling an output pin.

2. Timers:
    a. The Timers A and B in the MSP430 are flexible peripherals used for various time-based operations such as generating Pulse Width Modulation (PWM) signals, counting events, and measuring time intervals.
        i. Timer A is typically used for general timing tasks and PWM generation, making it suitable for most applications that require periodic signals or time delays.
        ii. Timer B offers higher precision and additional advanced features, making it ideal for more complex tasks that demand finer control, such as precise frequency generation or managing multiple PWM outputs simultaneously.
    b. Both timers allow the microcontroller to perform time-sensitive tasks without constantly engaging the CPU, which enables the system to remain in low-power modes (LPM) while still handling these operations efficiently. This capability is crucial in applications where power conservation is important, such as battery-operated devices.

Results & Observation

Part 1:

Program Description:

This program controls the brightness of an LED using Pulse Width Modulation (PWM) generated by Timer A on the microcontroller. The program starts with the LED at 50% brightness, and provides nine adjustable levels of brightness, ranging from 0% to 100% . Pressing SW1 increases the brightness to the next level, up to a maximum of 100%. Pressing SW2 decreases the brightness to the previous level, down to a minimum of 0%. Additionally, the bonus assignment was completed, enabling LED1 to blink every 3 seconds while maintaining its current brightness level when both SW1 and SW2 are pressed simultaneously.

Process:

1. The first thing I did was initialize LEDs, Switches, and PWM:
    a. Set up the LED pin (P1.2) as output for PWM control and configure it to start at 50% brightness.
    b. Configure SW1 (P2.1) and SW2 (P1.1) as inputs with pull-up resistors enabled for detecting button presses.
2. Next, I had to configure the timer and PWM signal:
    a. Use Timer A to generate a PWM signal on the LED pin, setting the frequency high enough to prevent visible flickering.
    b. Define 9 brightness levels and set the initial duty cycle to correspond with 50% brightness.
3. Next, I set up the button interrupts:
    a. Enable interrupts for SW1 and SW2 to detect button presses that adjust brightness.
    b. Set the interrupt edge to trigger on a high-to-low transition, indicating a button press, and clear any existing interrupt flags.
4. Then, I had to implement debouncing in ISRs:
    a. In each button's Interrupt Service Routine (ISR), add a debounce delay to ensure stable detection of button presses.
    b. After debouncing, check the switch state again before adjusting the brightness level.
5. Finally, I had to add brightness adjustment and blinking mode logic:
    a. In SW1's ISR, increase brightness if not already at the maximum; in SW2's ISR, decrease brightness if not at the minimum.
    b. BONUS: In the Watchdog Timer ISR, detect simultaneous button presses to toggle a blinking mode, where the LED blinks on and off every 3 seconds at the current brightness level.

Flowchart:

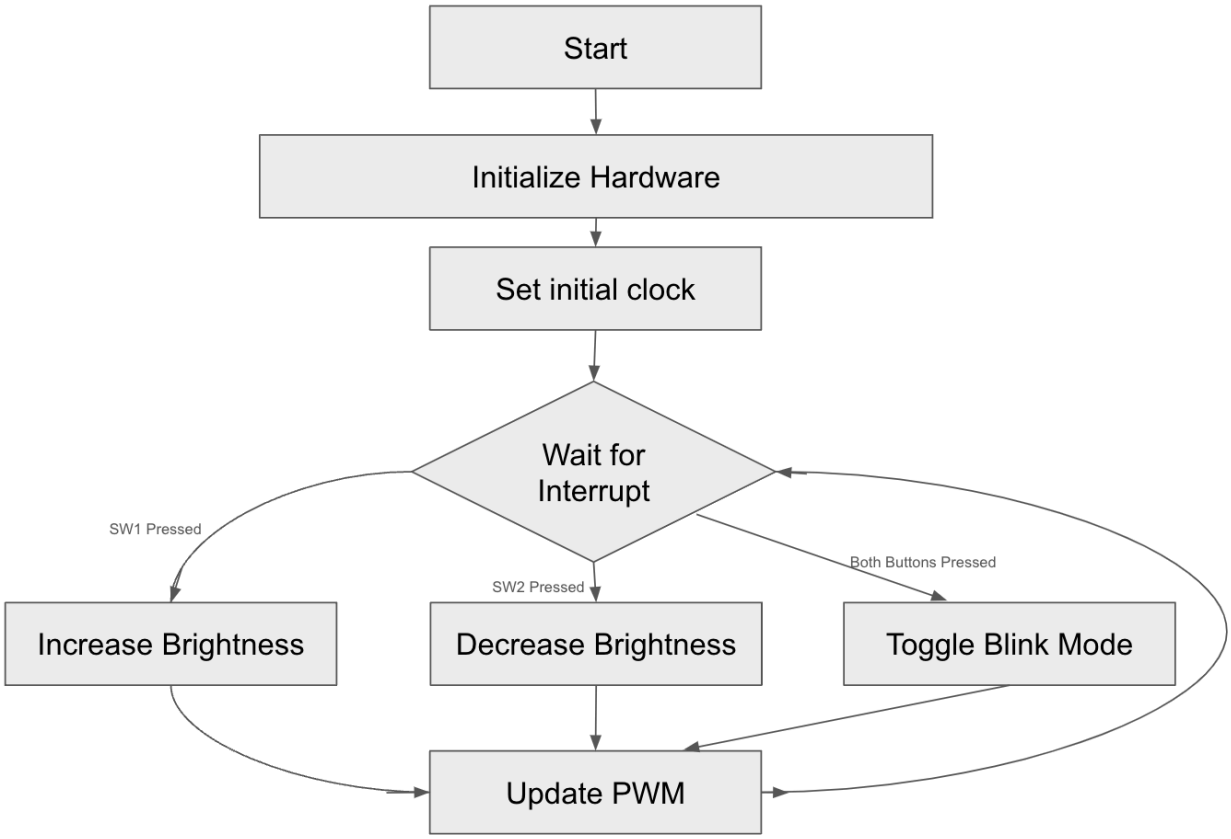

Figure 1: Program 2 Flowchart

Part 2:

Program 2 Description:

This program controls a buzzer connected to P7.4 using Pulse Width Modulation (PWM) via Timer B at a frequency of 2 kHz. The buzzer sound is toggled on and off every second by using the Watchdog Timer (WDT), which generates interrupts every 32 milliseconds.

Process:

1. First, I did my initialization:
    a. Stopped the Watchdog Timer to prevent unexpected resets.
    b. Configured the Watchdog Timer to generate interrupts every 32 ms.

c. Set up Timer B to produce a 2 kHz PWM signal on P7.4 with a 50% duty cycle.

d. Configured P7.4 as the output pin for the buzzer and enabled global interrupts.

2. Next, I implemented the main loop:

    a. Entered Low Power Mode 0 (LPM0) to save energy, allowing the microcontroller to sleep until an interrupt occurs.

3. Next, I did Watchdog Timer interrupt handling:

    a. In the ISR, I counted 32 interrupts (~1 second).

    b. After 1 second, I toggled the buzzer's PWM signal on or off by changing the output mode of Timer B.

4. Finally, I configured the PWM:

    a. Set Timer B to maintain a 2 kHz PWM signal with TB0CCR0 = 250 - 1 (period) and TB0CCR2 = 125 (50% duty cycle), ensuring the buzzer sounds with the desired frequency and toggles every second.

Calculations:

1. f = SMCLK/(2(TBCCR0)) → 2 kHz = 1 MHz/(2(TBCCR0)) → 2000 =1000000/(2(TBCCR0)) → 1000000/4000 = 250

Conclusion

In this lab, I explored the use of Pulse Width Modulation (PWM) and timers to control peripheral devices such as LEDs and a buzzer using the MSP430 microcontroller. Through the implementation of Timer A, I successfully adjusted the brightness of LED2 with 9 distinct brightness levels, ranging from 0% to 100%. The brightness could be increased or decreased with the use of SW1 and SW2 buttons, while the system efficiently entered sleep mode when no actions were required, demonstrating effective power management. I also implemented Timer B and used the Watchdog Timer (WDT) to control the buzzer, producing a 2 kHz sound and toggling it on and off every second. The combination of hardware timers and interrupts allowed for efficient real-time control of both the LED and buzzer, all while keeping the microcontroller in low-power mode. The lab provided hands-on experience with PWM generation,

timer configuration, and interrupt handling, which are essential skills in embedded systems development. Furthermore, the bonus assignment taught the value of simultaneous button presses for controlling additional functionality, such as blinking the LED at a low frequency, utilizing the watchdog timer effectively. I did not have too many issues with the lab and completed both parts along with the bonus.

Appendix

Table 1: Program 1 with Bonus source code

```
/*-------------------------------------------------------------------------
File:        Lab07_P1.c
Description:  Changes the brightness of LED1 using timer A as the pwm signal source.
          The  Initial brightness is set at 50%. It can be increased or decreased by buttons.
Input:       P2.1 - SW1 (Increase) | P1.1 - SW2 (Decrease)
Output:     P1.0 - RED LED | P4.7 - GREEN LED
Author:       Gianna Foti
* -------------------------------------------------------------------------*/
#include <msp430.h>
#define BUTTON_INCREASE (P2IN & BIT1)  // STATUS OF SW1 (P2.1)
#define BUTTON_DECREASE (P1IN & BIT1)  // STATUS OF SW2 (P1.1)
// PWM BRIGHTNESS LEVELS: 0%, 12.5%, 25%, 37.5%, 50%, 62.5%, 75%, 87.5%, 100%
const unsigned int brightyBrightLevelyLevel[] = {0, 125, 250, 375, 500, 625, 750, 875, 1000};
static unsigned int currentBrightnessLevel = 4; // INITIAL BRIGHTNESS AT 50%
unsigned int debounceDelay = 0;
// determines da blinking state ya feel
static int isBlinking = 0;
static int blinkCounter = 0;
static int ledState = 1;
//lil function protypes
void updateyDateBrightyBright();
int main(void)
{
   WDTCTL = WDTPW | WDTHOLD; // STOP WATCHDOG TIMER
   // CONFIGURE PWM OUTPUT
   P1DIR |= BIT3;  // Set P1.3 as output
   P1SEL |= BIT3;  // Select TA0.1 for PWM output
// confingure buttons yuh
   // CONFIGURE BUTTON SW1 (Increase)
   P2DIR &= ~BIT1; // Set P2.1 as input
   P2REN |= BIT1;  // Enable pull-up/pull-down resistor
   P2OUT |= BIT1;  // Set as pull-up
   P2IE |= BIT1;   // Enable interrupt for SW1
   P2IES |= BIT1;  // Set interrupt on falling edge
   P2IFG &= ~BIT1; // Clear interrupt flag
   // CONFIGURE BUTTON SW2 (Decrease)
   P1DIR &= ~BIT1; // Set P1.1 as input
   P1REN |= BIT1;  // Enable pull-up/pull-down resistor
   P1OUT |= BIT1;  // Set as pull-up
   P1IE |= BIT1;   // Enable interrupt for SW2
   P1IES |= BIT1;  // Set interrupt on falling edge
   P1IFG &= ~BIT1; // Clear interrupt flag
   WDTCTL = WDT_MDLY_32; // Configure watchdog timer for 32 ms
   SFRIE1 |= WDTIE; // Enable watchdog timer interrupt
   // CONFIGURE TIMER A FOR PWM
  TA0CCR0 = 1000 - 1; // Set period for 1 kHz PWM frequency
  TA0CCTL2 = OUTMOD_7; // Set/reset mode
  TA0CCR2 = brightyBrightLevelyLevel[currentBrightnessLevel]; // Initialize at 50% brightness
  TA0CTL = TASSEL_2 | MC_1; // Use SMCLK, up mode
   _EINT(); // Enable global interrupts
```

```
    // Enter low power mode
    __bis_SR_register(LPM0_bits + GIE);
}
void updateyDateBrightyBright()
{
   // Update PWM duty cycle for brightness
    TA0CCR2 = brightyBrightLevelyLevel[currentBrightnessLevel];
}
// Interrupt Service Routine for SW2 (Decrease Brightness)
#pragma vector = PORT1_VECTOR
__interrupt void PORT1_ISR(void)
{
    if (P1IFG & BIT1) // Check if SW2 was pressed
    {
        for(debounceDelay = 25000; debounceDelay > 0; debounceDelay--); // Debounce
        if (currentBrightnessLevel > 0) // Check if level can be decreased
        {
            currentBrightnessLevel--; // Decrease brightness level
            updateyDateBrightyBright(); // Update PWM
        }
    }
    P1IFG &= ~BIT1; // Clear interrupt flag
}
// Interrupt Service Routine for SW1 (Increase Brightness)
#pragma vector = PORT2_VECTOR
__interrupt void PORT2_ISR(void)
{
    if (P2IFG & BIT1) // Check if SW1 was pressed
    {
        for(debounceDelay = 25000; debounceDelay > 0; debounceDelay--); // Debounce
        if (currentBrightnessLevel < 8) // Check if level can be increased
        {
            currentBrightnessLevel++; // Increase brightness level
            updateyDateBrightyBright(); // Update PWM
        }
    }
    P2IFG &= ~BIT1; // Clear interrupt flag
}
// Watchdog Timer Interrupt Service Routine
#pragma vector = WDT_VECTOR
__interrupt void WATCHDOG_TIMER(void)
{
    if ((BUTTON_INCREASE == 0) && (BUTTON_DECREASE == 0)) // Both buttons pressed
    {
        if(isBlinking == 0) // If not already in blinking mode
        {
            isBlinking = 1; // Enable blinking
            blinkCounter = 0; // Reset counter
            ledState = 1; // Set LED state to on
        }
        blinkCounter++; // Increment blink counter
        if (blinkCounter >= 94) // Approximately every 3 seconds
        {
```

```
            ledState = !ledState; // Toggle LED state
            if (!ledState) // If LED is off
            {
               TA0CCR2 = brightyBrightLevelyLevel[currentBrightnessLevel]; // Set to current brightness
            }
            else // If LED is on
            {
               TA0CCR2 = 0; // Turn off LED
            }
            blinkCounter = 0; // Reset blink counter
         }
      }
      else // If buttons are not pressed
      {
         if(isBlinking == 1) // If in blinking mode
         {
            isBlinking = 0; // Disable blinking
            blinkCounter = 0; // Reset counter
            ledState = 1; // Turn LED back on
            TA0CCR2 = brightyBrightLevelyLevel[currentBrightnessLevel]; // Restore brightness
         }
      }
   }
```

Table 2: Program 2 source code

```
/*-------------------------------------------------------------------------------
File:         Lab07_P2.c
Description:  Controls a buzzer with PWM using Timer B at 2kHz and toggles the
              sound on and off every second using the Watchdog Timer.
Input:        P2.1 - SW1 (Increase) | P1.1 - SW2 (Decrease)
Output:       P1.0 - RED LED | P4.7 - GREEN LED | P7.4 - BUZZER
Author:       Gianna Foti
* -------------------------------------------------------------------------------*/
#include <msp430.h>
static int i = 0;
void main(void)
{
   // Disable Watchdog Timer to prevent system reset
   WDTCTL = WDTPW | WDTHOLD;
   // Configure Watchdog Timer to generate interrupts every 32ms
   WDTCTL = WDT_MDLY_32;    // Set Watchdog Timer mode to 32ms interval
   SFRIE1 |= WDTIE;         // Enable Watchdog Timer interrupts
   // Configure P7.4 as output for the buzzer
   P7DIR |= BIT4;           // Set P7.4 as output pin
   P7SEL |= BIT4;           // Select peripheral function (Timer B) for P7.4
   // Set up Timer B to generate a 2kHz PWM signal on P7.4
   TB0CCTL2 = OUTMOD_7;     // Set output mode to Reset/Set (PWM mode)
   TB0CTL = TBSSEL_2 | MC_1; // Use SMCLK as the clock source, set Timer B in up mode
   TB0CCR0 = 250 - 1;       // Set Timer B period to achieve 2kHz (assuming SMCLK is 1MHz)
   TB0CCR2 = 125;           // Set duty cycle to 50% for 2kHz signal
```

```
    // Enter Low Power Mode 0 (LPM0) with global interrupts enabled
    __bis_SR_register(LPM0_bits + GIE);
}
// Watchdog Timer interrupt service routine (ISR)
#pragma vector = WDT_VECTOR
__interrupt void WATCHDOG_TIMER(void)
{
    i++;                    // Increment counter every 32ms
    if (i == 32)              // After 32 interrupts (~1 second has passed)
    {
        TB0CCTL2 ^= OUTMOD_7; // Toggle the buzzer on/off by modifying output mode
        i = 0;                // Reset counter after toggling
    }
}
```