CPE 325: Intro to Embedded Computer System

Lab 06

MSP430 Timers (Watchdog Timer, Timer A, and Timer B)

Submitted by: Hannah Kelley

Date of Experiment: October 6, 2025

Report Deadline: October 13, 2025

Demonstration Deadline: October 13, 2025

Theory

Topic 1: Watchdog Timer

The Watchdog Timer (WDT) is a hardware module with two modes: watchdog mode and interval mode. In watchdog mode, the WDT periodically expects to be cleared by the program and will reset the entire system if it is not serviced in time. This mode is useful in safety-critical or remote systems where the device must automatically recover from software crashes — for example, in an environmental sensor node deployed outdoors.

In interval mode, the WDT behaves like a regular timer, generating periodic interrupts without causing resets. This mode can be used for scheduled actions such as blinking an LED once per second, toggling a buzzer on and off, or waking the CPU periodically for sensor polling.

Topic 2: Timers

Timer A and Timer B are built-in hardware timers on the MSP430 used for time-related operations such as measuring intervals, generating periodic events, and producing PWM signals. They operate independently of the CPU and can continue running even when the processor is in sleep mode, which makes them useful in both timing and low-power applications.

Both timers support several operating modes. In stop mode, the timer is halted completely and consumes no energy until needed again. In up mode, the timer repeatedly counts from zero to a specified value and then resets, which is commonly used for generating periodic interrupts or producing a fixed-frequency PWM signal. Continuous mode allows the timer to count continuously from zero to its maximum value (0xFFFF), making it useful for measuring time intervals or capturing the length of incoming pulses. Up/down mode causes the timer to count up to a set value and then back down to zero, which is typically used for generating symmetric PWM waveforms.

Both timers include capture/compare registers that allow them to trigger events or measure input signals at precise times. In practice, they are commonly used to control LED brightness through PWM, toggle GPIO pins at fixed intervals such as for blinking LEDs, or measure pulse widths from sensors. Timer A usually offers fewer channels, while Timer B often provides more compare outputs and higher resolution, making it more suitable when multiple PWM outputs or synchronized events are required.

Results & Observation

Program 1:

Program Description:

This program implements adjustable LED brightness control on an MSP430 microcontroller using Pulse Width Modulation (PWM) generated by Timer A. The PWM frequency is configured high enough to eliminate visible flicker, enabling smooth dimming across seven discrete brightness levels: 0%, 16.67%, 33.33%, 50% (default), 66.67%, 83.34%, and 100%. Brightness is adjusted using pushbuttons SW1 and SW2, where SW2 increments and SW1 decrements the level, with both constrained at the upper and lower bounds to prevent overflow. Button input is debounced in software to avoid unintended skipping between levels. In normal operation, the microcontroller remains in low-power

sleep mode until a button press occurs. Additionally, when both SW1 and SW2 are held simultaneously, the program switches to a special mode where LED1 blinks at approximately 0.17 Hz (3 seconds on, 3 seconds off) using the watchdog timer, while maintaining the current PWM brightness. Releasing either button returns the system to standard brightness control mode.

Program Flowchart:

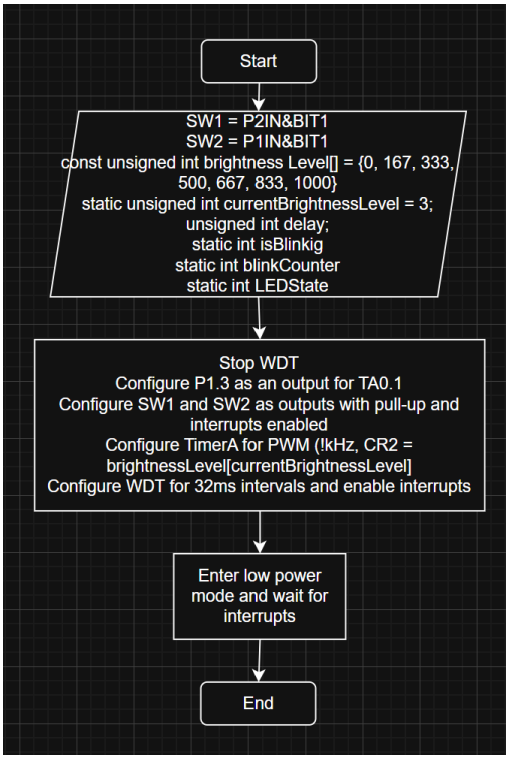

Figure 01: Program 1 Flowchart

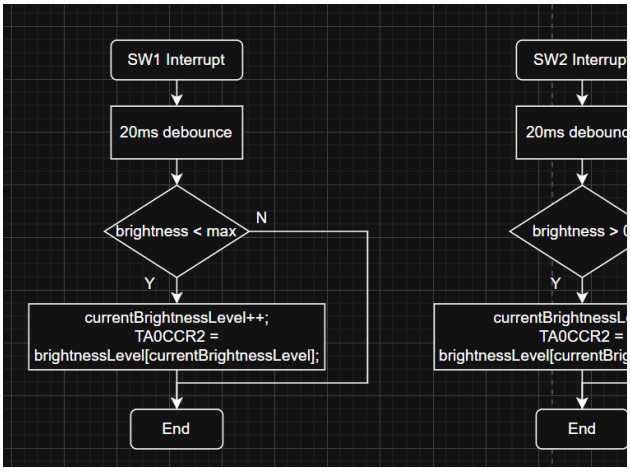

Figure 02: Switch Interrupt Flowcharts

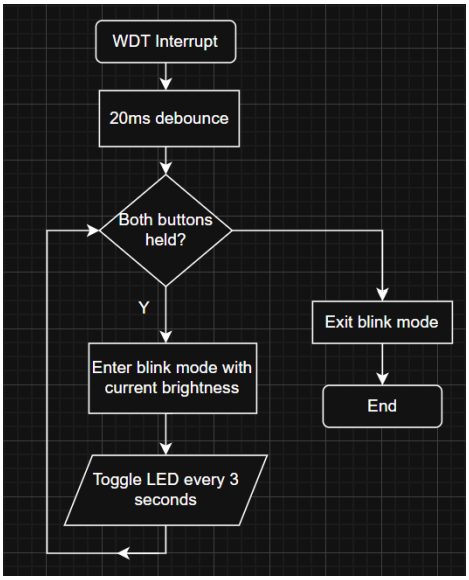

Figure 03: WDT Interrupt Flowchart

Program 2:

Program Description:

This program uses Timer B to drive a buzzer at approximately 1 kHz. The watchdog timer is configured to trigger once per second, and inside its interrupt routine the buzzer is alternated between on and off states. This creates a repeating pattern of one second of sound followed by one second of silence. When no change is required, the microcontroller remains in sleep mode to conserve power.

Report Questions:

1. **Calculation for producing 1 KHz sound**

Timer B is used to drive the buzzer frequency. The period of the timer can be calculated by dividing the clock frequency by the desired frequency. In this case, the 1MHz clock frequency is divided by the 1 kHz desired frequency to get a timer frequency of 1 kHz. In the code I use 1048 Hz since the actual clock frequency is 1.048 MHz.

Appendix

Table 01: Program 1 Source Code

```
/*-------------------------------------------------------------------------
 * File:        Lab03_P1.c
 * Description:
 *
 * Board:       5529
 * Input:       SW1 and SW2
 * Output:      LED1
 * Author:      Hannah Kelley
 * Date:        October 6, 2025
 *-------------------------------------------------------------------------*/
#include <msp430.h>
#include <stdio.h>

#define SW1 P2IN&BIT1
#define SW2 P1IN&BIT1

const unsigned int brightnessLevel[] = {0, 167, 333, 500, 667, 833, 1000};
static unsigned int currentBrightnessLevel = 3; // init brightness = 50%
unsigned int delay = 0;

static int isBlinking = 0;
static int blinkCounter = 0;
static int LEDState = 1;

void updateBrightness(); // function prototype

int main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop WDT
    // config PWM output
    P1SEL |= BIT3; // select TA0.1 for PWM output
    P1DIR |= BIT3; // set P1.2 to output
    // config SW1 (increase)
    P2DIR &= ~BIT1; // set SW2 as input
    P2REN |= BIT1;  // enable pullup reg
    P2OUT |= BIT1;  // set as output
    P2IE |= BIT1;   // enable interupt for SW2
    P2IES |= BIT1;  // set interrupt on falling edge
    P2IFG &= ~BIT1; // clear interrupt flag
    // config SW@ (decrease)
    P1DIR &= ~BIT1; // set SW1 as input
    P1REN |= BIT1;  // enable pullup reg
    P1OUT |= BIT1;  // set as output
    P1IE |= BIT1;   // enable interupt for SW1
    P1IES |= BIT1;  // set interrupt on falling edge
    P1IFG &= ~BIT1; // clear interrupt flag
    WDTCTL = WDT_MDLY_32; // config WDT for 32ms
    SFRIE1 |= WDTIE;      // enable WDT interrupt
    // config timer A for PWM
    TA0CCR0 = 1000 - 1;         // set period to 1kHz
    TA0CCTL2 = OUTMOD_7;        // set/reset mode
    TA0CCR2 = brightnessLevel[currentBrightnessLevel];
    TA0CTL = TASSEL_2 | MC_1;   // use SMCLK in up mode
    __EINT(); // Enable global interrupts
```

```
    __bis_SR_register(LPM0_bits + GIE);
    return 0; // end of program
}
void updateBrightness() {
    TA0CCR2 = brightnessLevel[currentBrightnessLevel];
    return;
}
// SW1 ISR (increase brightness level)
#pragma vector = PORT2_VECTOR
__interrupt void PORT2_ISR(void) {
    if (P2IFG & BIT1) {                             // if SW1 pressed
        for (delay = 20000; delay > 0; delay--);    // debounce
        if (P2IFG & BIT1) {      // if still pressed it is a valid press
            if (currentBrightnessLevel < 6) {  // if level can be increased
                currentBrightnessLevel++;           // increase level
                updateBrightness();                 // update LED
            }
        }
    }
    P2IFG &= ~BIT1;                                 // clear interrupt flag
}
// SW2 ISR (increase brightness level)
#pragma vector = PORT1_VECTOR
__interrupt void PORT1_ISR(void) {
    if (P1IFG & BIT1) {                             // if SW2 pressed
        for (delay = 20000; delay > 0; delay--);    // debounce
        if (P1IFG & BIT1) {         // if still pressed it is a valid press
            if (currentBrightnessLevel > 0) {  // if level can be increased
                currentBrightnessLevel--;           // decrease level
                updateBrightness();                 // update LED
            }
        }
    }
    P1IFG &= ~BIT1;                                 // clear interrupt flag
}
// WDT ISR
#pragma vector = WDT_VECTOR
__interrupt void WATCHDOG_TIMER(void) {
    if(((SW1) == 0) && ((SW2) == 0)) {
        if (isBlinking == 0) {
            isBlinking = 1;
            blinkCounter = 0;
            LEDState = 1;
        }
        blinkCounter++;
        if (blinkCounter >= 94) {
            LEDState = !LEDState;
            if (!LEDState) {
                TA0CCR2 = brightnessLevel[currentBrightnessLevel];
            }
            else {
                TA0CCR2 = 0;
            }
            blinkCounter = 0;
        }
    }
    else {
```

```
        if (isBlinking == 1) {
            isBlinking = 0;
            blinkCounter = 0;
            LEDState = 1;
            TA0CCR2 = brightnessLevel[currentBrightnessLevel];
        }
    }
```

Table 02: Program 2 Source code

```
/*-------------------------------------------------------------------------
 * File:        Lab06_P2.c
 * Description: Generates a PWM tone on a buzzer and toggles it on and off
at
 *              timed intervals using the watchdog timer.
 * Board:       5529
 * Input:       Timer B and WDT
 * Output:      buzzer turning on and off at a 1HZ frequency
 * Author:      Hannah Kelley
 * Date:        October 6, 2025
 *-------------------------------------------------------------------------*/
#include <msp430.h>
#include <stdio.h>

static int i = 0;

int main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop WDT
    // config WDT
    WDTCTL = WDT_MDLY_32;   // set WDT with 32ms intervals
    SFRIE1 |= WDTIE;      // enable WDT interrupts
    //config buzzer
    P7DIR |= BIT4;          // set P7.4 to output
    P7SEL |= BIT4;          // select Timer B periphreal function for P4.7
    // config Timer B for PWM
    TB0CCTL2 = OUTMOD_7;
    TB0CTL = TBSSEL_2 | MC_1;
    TB0CCR0 = 1048 - 1; // period = 1Mhz/1048 ~= 1000 Hz
    TB0CCR2 = 524;      // 50% duty cycle
    // enter low power mode 0 w/ GIE enabled
    __bis_SR_register(LPM0_bits + GIE);
    return 0; // end of program
}
// WDT ISR
#pragma vector = WDT_VECTOR
__interrupt void WATCHDOG_TIMER(void) {
    i++;
    if (i == 32) {
        TB0CCTL2 ^= OUTMOD_7;
        i = 0;
    }
}
```