CPE 325: Intro to Embedded Computer System

Lab07

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

Submitted by: Anshika Sinha

Date of Experiment:______February 19, 2025________________

Report Deadline:_________February 28, 2025________________

Demonstration Deadline: __March 3, 2025__________________________

Theory

Topic 1: Watchdog Timer

a) A Watchdog Timer (WDT) is a hardware timer used to reset the microcontroller if the system malfunctions or becomes unresponsive. In the MSP430, the WDT is a 16-bit timer designed to enhance system reliability by recovering from software faults, such as infinite loops or memory corruption.

b) In Watchdog Mode, the watchdog timer resets the system if the counter is not cleared within a specified time interval. This mode is used to recover from software hangs or faults. In contrast, Interval Timer Mode generates periodic interrupts without resetting the system, making it useful for timing or software polling.

c) In Watchdog Mode, the software must periodically clear the watchdog timer to prevent a system reset. If the system malfunctions and fails to do so, the watchdog timer automatically resets the microcontroller. This is ideal for safety-critical applications like automotive systems, where software faults could lead to unsafe conditions.

d) Interval Timer Mode allows the watchdog timer to generate interrupts at regular intervals, which can be used to trigger tasks like sensor polling or updating a display. In this mode, the WDT does not reset the system, allowing it to continue running even if an interrupt is missed. This is useful in applications like data logging, where periodic readings are needed.

e) The WDT in MSP430 can use various clock sources, including ACLK, SMCLK, VLOCLK, and DCO.

Topic 2: Timers

a) The MSP430F5529 microcontroller includes two hardware timers, Timer_A and Timer_B, which are used for precise timing, event counting, and waveform generation. These timers are 16-bit and highly configurable.
b) Timer_A is commonly used for timing operations, PWM generation, and input capture. It has multiple capture/compare registers that allow precise control over timing events. Timer_A can be clocked using the sources ACLK, SMCLK, and VLOCLK.
c) Timer_B is similar to Timer_A, but it includes more capture/compare registers and advanced capabilities. It supports up to seven capture/compare channels, allowing numerous PWM signals to be generated at the same time. Timer_B is also ideal for applications that require better resolution or multiple timing events, such as advanced motor control or complicated waveform creation.
d) Both Timer_A and Timer_B operate in several modes, including Up Mode, Continuous Mode, and Up/Down Mode. In Up Mode, the timer counts from zero to a specified value and resets, generating an interrupt at the end of each cycle.
e) In Continuous Mode, the timer counts from zero to its maximum value (0xFFFF) and rolls over, continuously generating interrupts. This mode is typically used for input capture applications, where the exact timing of an external event needs to be recorded. Up/Down Mode allows the

timer to count up to a specified value and then down to zero, useful for generating symmetrical PWM waveforms for motor control.

Results & Observation

Program 1:

Program Description:

This C program controls the brightness of LED2 using a PWM signal generated by Timer A. It offers six brightness levels (0%, 20%, 40%, 60%, 80%, and 100%), starting at 20% by default. The brightness can be adjusted using two buttons. SW1 (P2.1) decreases the brightness, while SW2 (P1.1) increases it. If both buttons are pressed simultaneously, LED1 blinks at approximately 0.17 Hz (3 seconds on and 3 seconds off) while maintaining the current brightness level. The program efficiently handles button debouncing to ensure smooth level transitions and uses the Watchdog Timer to manage the blinking cycle. When no actions are required, the microcontroller enters a low-power mode, conserving energy while maintaining responsiveness.

Program Flowchart:

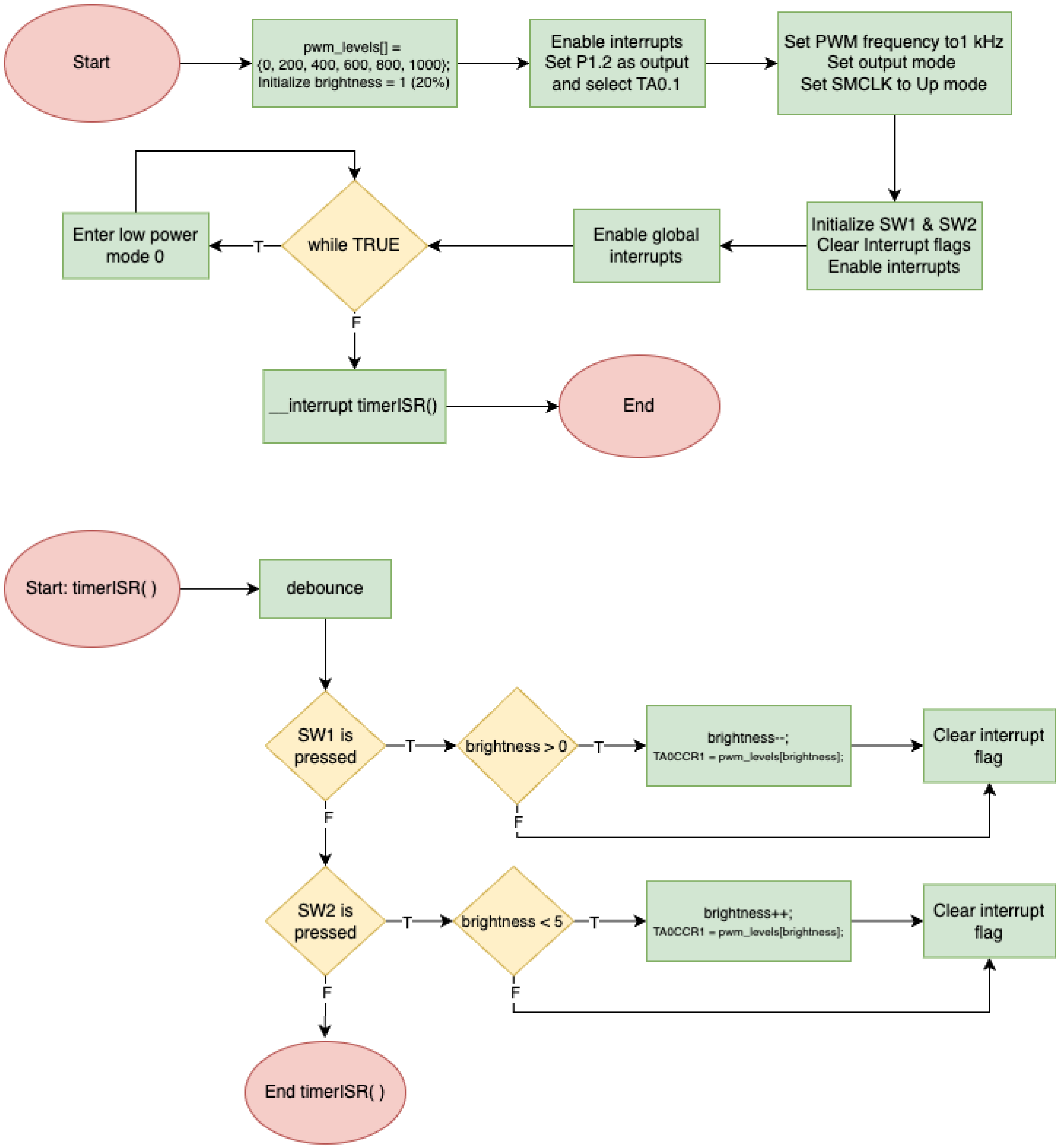

Figure 02: Program 1 Flowchart

Program 2:

Program Description:

This program for the MSP430F5529 microcontroller generates a 2 kHz sound using Timer B and a connected buzzer, toggling the sound on and off every second. The sound is produced through a Pulse Width Modulation (PWM) signal with a 50% duty cycle, achieved by setting Timer B in Up Mode. The Watchdog Timer (WDT) is configured to trigger an interrupt every second, which toggles Timer B between active and stopped modes, controlling the sound. To conserve power, the microcontroller enters Low Power Mode 3 (LPM3) when no processing is required, keeping ACLK active while disabling MCLK and SMCLK.

Calculations:

Period (T) for 2 kHz sound = $\frac{1}{2000} = 0.0005\ s = 500\ \mu s$

Since the the output toggles per cycles between high and low,

$Timer = \frac{500\ \mu s}{2} = 250\ \mu s$

Appendix

Table 01: Program 1 source code

```
/*---------------------------------------------------------------------------
* File:        Lab07_P1.c
* Function:    Changing brightness level of LED2 using PWM (MPS430F5529)
* Description: This C program controls the brightness of LED2 using a PWM
signal
*              generated by Timer A. It offers six brightness levels (0%, 20%,
40%, 60%, 80%, and 100%),
*              starting at 20% by default.
*
* Clocks:      ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO = default (2^20 Hz)
*              An external watch crystal between XIN & XOUT is required for
ACLK
*
* Input:       Press SW2 to increase brightness and SW1 to decrease brightness
* Output:      LED2 is on at 0-100% duty cycle at different brightnesses
*
* Author:      Anshika Sinha
*--------------------------------------------------------------------------*/
#include <msp430F5529.h>
#include "intrinsics.h"
#define SW1 P2IN&BIT1                           // Switch 1 at P2.1
#define SW2 P1IN&BIT1                           // Switch 2 at P1.1

// Define PWM levels for 0%, 20%, 40%, 60%, 80%, and 100%
unsigned int dutyCycle = 2;             // Start at 20%
unsigned int clock = 1;

void main(void) {
   WDTCTL = WDTPW | WDTHOLD;           // Stop watchdog timer

   // Configure LED
   P1DIR |= 0x01;                    // Set LED1 (P1.0) as output
   P1OUT |= 0x01;                   // Start LED1 as ON
```

```c
    // Configure switches as inputs
    P2DIR &= ~BIT1;                 // Set P2.1 as input for SW1 input
    P2REN |= BIT1;                  // Enable pull-up register at P2.1
    P2OUT |= BIT1;

    P1DIR &= ~BIT1;                 // Set P1.1 as input for SW2 input
    P1REN |= BIT1;                  // Enable pull-up register at P1.1
    P1OUT |= BIT1;

    _EINT();                        // Enable interrupts
    // Configure switches for interrupts
    P2IE |= BIT1;                   // Enable interrupt for SW1 (P2.1)
    P2IES |= BIT1;                  // Set interrupt from high to low
    P2IFG &= ~BIT1;                 // Clear interrupt flag

    P1IE |= BIT1;                   // Enable interrupt for SW2 (P1.1)
    P1IES |= BIT1;                  // Set interrupt from high to low
    P1IFG &= ~BIT1;                 // Clear interrupt flag

    // Clock, ACLK continuous
    TA0CCTL0 = CCIE;                // Set output mode
    TA0CCR0 = 1;                    // Start at 20%
    TA0CTL = TASSEL_1 + MC_1;           // Set SMCLK to Up mode

    _BIS_SR(LPM3 + GIE);       // Enter low power mode 0
}

// Switch 1 ISR
#pragma vector=PORT2_VECTOR
__interrupt void Port2_ISR(void) {
    __delay_cycles(40000);              // debounce

    P2IE &= ~BIT1;                      // Disable interrupt
    if ((SW1) == 1)
        return;                         // If SW1 is pressed
    if (dutyCycle > 0) {
        dutyCycle -= 2;                 // decrease brightness
```

```
    }

    P2IFG &= ~BIT1;                        // Clear interrupt flag
    P2IE |= BIT1;                          // Enable interrupt
}

// Switch 2 ISR
#pragma vector=PORT1_VECTOR
__interrupt void Port1_ISR(void) {
    __delay_cycles(40000);                 // debounce

    P1IE &= ~BIT1;                         // Disable interrupt
    if ((SW2) == 1)
        return;                            // If SW2 is pressed
    if (dutyCycle < 10) {
        dutyCycle += 2;                    // increase brightness
    }

    P1IFG &= ~BIT1;                        // Clear interrupt flag
    P1IE |= BIT1;                          // Enable interrupt
}

#pragma vector=TIMER0_A0_VECTOR
__interrupt void timerISR(void) {
    clock++;
    if (clock > 10)
        clock = 1;
    if (dutyCycle >= clock)
        P1OUT |= BIT0;
    else
        P1OUT &= ~BIT0;
}
```

Table 02: Program 2 source code

```c
/*--------------------------------------------------------------------------*
File:        Lab7_P2.c
*
* Function:    Toggle buzzer using Timer_B (MPS430F5529)
*
* Description: This C program for the MSP430F5529 microcontroller generates a
2 kHz sound
*              using Timer B and a connected buzzer, toggling the sound on and
off every second.
*
* Clocks:      ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO = default (2^20 Hz)
*              An external watch crystal between XIN & XOUT is required for
ACLK
* Input:       None
* Output:      Produce buzzer sound at 2 MHz and toggle every 1 second
* Author:      Anshika Sinha
*------------------------------------------------------------------------*/
#include <msp430F5529.h>
#include <assert.h>
#include <stdbool.h>
#include <stdint.h>
#include <intrinsics.h>

volatile int buzzer = 0;

void main(void) {
   WDTCTL = WDTPW + WDTHOLD;     // Stop WDT
   SFRIE1 |= WDTIE;              // Enable Watchdog timer Interrupt

   P7DIR |= BIT4;               // P7.4 output
   P7SEL |= BIT4;               // P7.4 special function (TB0.2 output)



   TBCCR0 |= 500;               // Value to count upto for Up mode
   TB0CTL = TBSSEL_2 + MC_1;    // ACLK is clock source, UP mode

   WDTCTL = WDT_ADLY_1000;       // 2s interval
```

```
    _BIS_SR(LPM0_bits + GIE);    // Enter Low Power Mode 3
}

// Watchdog timer ISR
#pragma vector = WDT_VECTOR
__interrupt void WDT_ISR(void) {
    if (buzzer == 1) {
        TB0CCTL2 = OUTMOD_0;             // Start timer in up mode
        buzzer = 0;
    } else {
        TB0CCTL2 = OUTMOD_4;            // Stop Timer B
        buzzer = 1;
    }
}
```

Make sure to submit a single pdf file only