CPE 325: Intro to Embedded Computer Systems

Lab07

MSP430 Timers, Watchdog Timer, Timers A and B

Submitted by: Michael Agnew

Date of Experiment: 2/28/2024

Report Deadline: 2/29/2024

Demonstration Deadline: 3/4/2024

Introduction:

For this lab, the main focus was the timers that are available for use in the MSP430. There were two different codes that were written that make use of three different timers in different ways. The first code made use of timer A, while the second code made use of timer B as well as the watchdog timer. This code was written in the C programming language and was compiled in Code Composer Studio. Some wires were also used in order to properly configure the timers to the correct output ports.

Theory:

General Timer Info: Timers can be thought of in a way related to a counter which increments or decreases its value according to the given clock frequency of the processor. These timers have various control registers that need to be configured in order for said timer to properly work. During this initialization, the timer's operating mode, clock frequency and interrupt status must be set. These timers also have comparison logic which compares the timer to a preset value in order for the timer to determine when it needs to rollover to zero, or perhaps toggle its output signal.

Watchdog Timer: The main function of the watchdog timer, which is typically referred to as (WDT) is to conduct a system restart when a problem occurs in the currently running software. When the specified interval of time is expired, the WDT executes a system reset. This is something that can be taken advantage of: for example, in the second part of this lab, the watchdog timer was used in order to generate arbitrary interrupts which would cause the buzzer and LED to execute at the desired time interval.

Timers A & B: Timer A and B are two very useful pieces of hardware that are located in the MSP430. One of the places where these timers are very useful is generating interrupts and not wasting hardware capacity on software delay. Using software delay causes the processor to waste energy as it is simply waiting for when it can execute the desired actions; whereas with timers, one can set a timer to generate an interrupt on a certain timer interval and keep the processor in low power mode until then to conserve energy. These timers run at a different clock signal than the processor which means that said timers will still count and run while the processor is off. So, not only are these timers extremely useful for saving power and energy for the processor, but they can also be configured for many things such as toggling LEDs, sounding buzzers, and more.

Problem 01:

Description: Problem one revolves around the idea of using Timer A to dim the red LED on the MSP430. The idea behind this is that timer A will be initialized to a certain value, 1000 for example, and will then be set to start its counting from 500, which is 50% of 1000, thus reducing the duty cycle of the LED to 50%, which will dim it. These numbers were chosen as they were nicely rounded and made it easy to calculate the needed duty cycle. Activating switch 1 will then increase the brightness of the LED while switch 2 will decrease the brightness of the LED by 25%. Therefore, the LED can go all the way down to 0%, which is off, or 100% which is the brightest it can be.

Program Design: This is all done by setting up timer A and changing its duty cycle depending on which switch is being pressed. The last part of this program is making sure that the program is in low power mode when none of the switches are being pressed. This ensures the maximum

amount of power is saved and there is no waste of energy. Low power mode is exited when a switch is pressed because the switches were configured with an interrupt service routine.

Flowchart: A flowchart was also required to be made for this problem and is as follows:

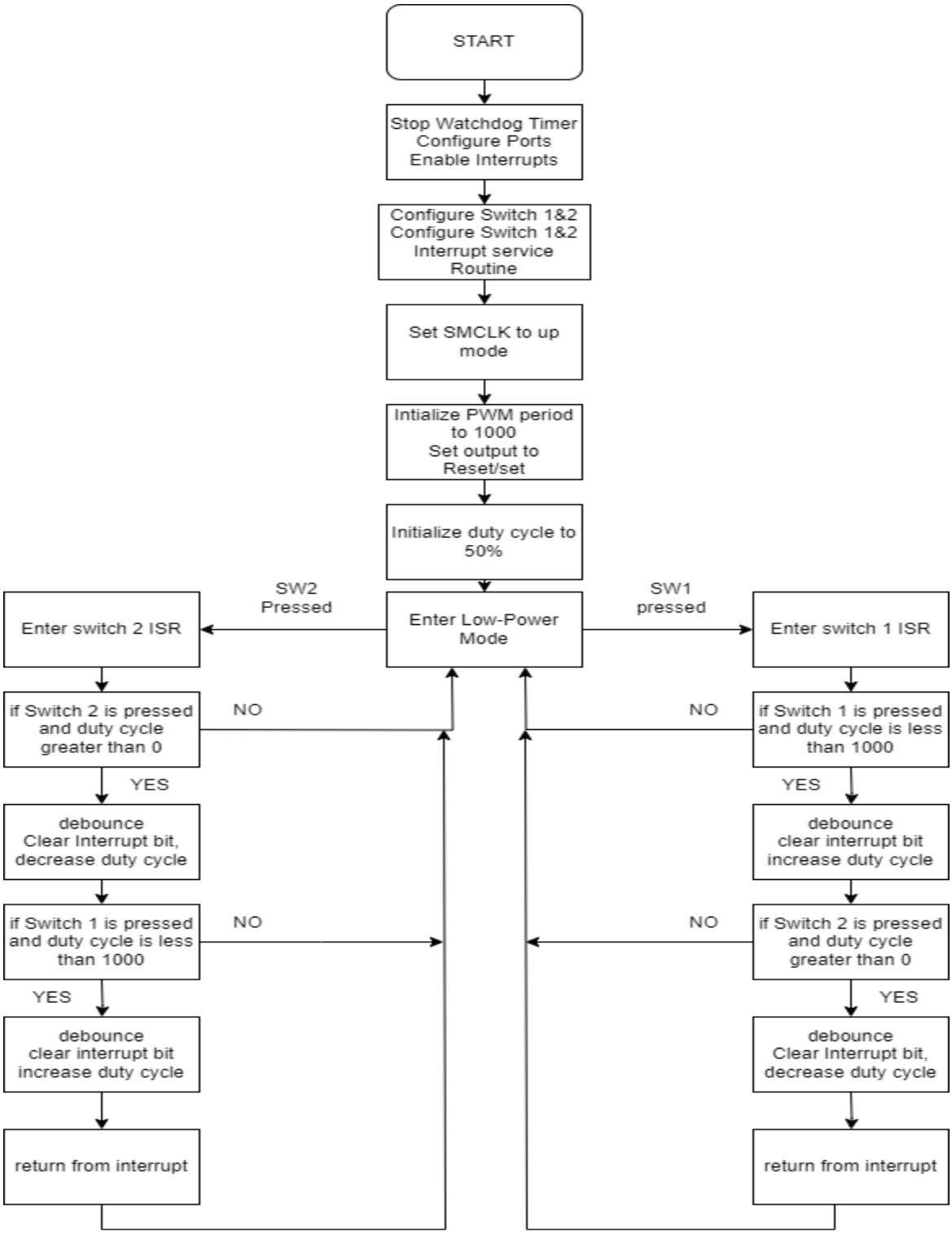

Figure 1. Flowchart for Lab 7 Part 1.

Problem 02:

Description: Problem two was based on the idea of using the watchdog timer to generate interrupts in order to execute certain actions on a set time interval. The interval that was chosen for this problem was 1s. Basis for this program is to have the program sit idly in low power mode, but every second it toggles the LED and sounds the buzzer for a second. This means that the program will spend 1s off, and 1s with the red LED and buzzer on. This will repeat infinitely until the program stops.

Program Design: This program was put into effect by first, enabling the interrupts to the watchdog timer, and then setting it up to set off an interrupt every second. The board then enters into its default state which is low power mode. Then, the interrupts were set up to configure the necessary bits for connecting the buzzer to the board, the chosen port being 2.2. Timer B was then configured to have SMCLK in up mode, with an output of reset/set mode. The buzzer was then set to 1khz which was calculated by SMCLK FREQ = 1,048,576Hz. 1,048,576/1000 = 1048. 1048 was then chosen as the time interval for timer B. After all of this, a simple if-else condition was set up to turn on the buzzer and LED for a second, and turn them off for the next.

Conclusion:

Fortunately, there were no issues encountered with this lab. Everything that was laid out to be completed was, and in the way that it was meant to be. There was much to be learned from this lab, namely the watchdog timer and timers A and B. These are very useful parts of the MSP430 and are extremely helpful for reducing energy consumption as well as code complexity. The ability to trigger interrupts at any time with a simple timer could help anyone making a program where such a thing was needed.

Appendix:

Problem 01:

```
/*-------------------------------------------------------------------------------
 * File:          Lab7Part1Code.c
 * Function:    Increases and Decreases the duty cycle of an LED based on user input.
 * Description: This program initializes LED1 at 50% brightness and allows the user to press
SW1 and SW2 to change the brightness.
 * Input:         Pressing S1 and S2
 * Output:       LED1 (At different variants of brightness)
 * Author(s):   Michael Agnew, ma0133@uah.edu
 * Date:          2/26/2024
 * ------------------------------------------------------------------------------*/

#include <msp430.h>

#define SW1 (P2IN & BIT1) // defining switch 1
#define SW2 (P1IN & BIT1) // defining switch 2

void main(void)
{ // start of main
        WDTCTL = WDTPW + WDTHOLD; // Stop dat boy the watchdog
        _EINT();                  // Enable interrupts

        // Configuring Ports and Bits
        P1DIR |= BIT2; // setting up p1.2 as direction
        P1SEL |= BIT2; // stuff dont worry about it


  // Configuring switch 1
        P2DIR &= ~BIT1;
        P2REN |= BIT1;
        P2OUT |= BIT1;

        // switch 1 ISR
        P2IE |= BIT1;                   // P1IE.BIT1 interrupt enabled
        P2IES |= BIT1;                  // P1IES.BIT1 hi/low edge
        P2IFG &= ~BIT1;                 // P1IFG.BIT1 is cleared

  // Configuring switch 2
```

```
        P1DIR &= ~BIT1;
        P1REN |= BIT1;
        P1OUT |= BIT1;

        // switch 2 ISR
        P1IE |= BIT1;                   // P1IE.BIT1 interrupt enabled
        P1IES |= BIT1;                  // P1IES.BIT1 hi/low edge
        P1IFG &= ~BIT1;                 // P1IFG.BIT1 is cleared


        TA0CTL = TASSEL_2 + MC_1; // Setting SMCLK to up mode
        TA0CCR0 = 1000; // Initializing the PWM period
        TA0CCTL1 = OUTMOD_7; // Setting the output to reset/set
        TA0CCR1 = 500; // initializing the duty cycle to 50%

        _BIS_SR(LPM0_bits + GIE);           // Enter LPM0(CPU is off); Enable interrupts


} // end of main


#pragma vector = PORT1_VECTOR // switch 2 interrupt vector
__interrupt void Port1_ISR (void)
{
        unsigned int i = 0;
        if ((SW2 == 0) && (TA0CCR1 > 0))
        {
        for (i = 0; i < 20000; i++); // debounce
        P1IFG &= ~BIT1; // P1IFG.BIT0 is cleared
        TA0CCR1 -= 250; // decrease the duty cycle by 25%
        for (i = 0; i < 20000; i++); // debounce

        }
        else if ((SW1 == 0) && (TA0CCR1 < 1000))
        {
        for (i = 0; i < 20000; i++); // debounce
        P1IFG &= ~BIT1; // P1IFG.BIT0 is cleared
        TA0CCR1 += 250; // increase the duty cycle by 25%
        for (i = 0; i < 20000; i++); // debounce

        }
        return;}

#pragma vector = PORT2_VECTOR // switch 1 interrupt vector
__interrupt void Port2_ISR (void)
{
```

```
        unsigned int i = 0; // loop counter
        if ((SW1 == 0) && (TA0CCR1 < 1000))
        {
        for (i = 0; i < 20000; i++); // debounce
        P1IFG &= ~BIT1; // P1IFG.BIT0 is cleared
        TA0CCR1 += 250; // increase the duty cycle by 25%
        for (i = 0; i < 20000; i++); // debounce

        }
        else if ((SW2 == 0) && (TA0CCR1 > 0))
        {
        for (i = 0; i < 20000; i++); // debounce
        P1IFG &= ~BIT1; // P1IFG.BIT0 is cleared
        TA0CCR1 -= 250; // decrease the duty cycle by 25%
        for (i = 0; i < 20000; i++); // debounce

        }
        return;}
```

Problem 02:

```
/*-------------------------------------------------------------------------
 * File:          Lab7Buzzer.c
 * Function:    Flashes an LED and sounds a buzzer in a 1s interval
 * Description: This program sounds a buzzer at 1khz and flashes an LED on a 1s interval
 * Input:         None
 * Output:       LED1 & Piezo Buzzer
 * Author(s):   Michael Agnew, ma0133@uah.edu
 * Date:          2/28/2024
 * -------------------------------------------------------------------------*/


#include <msp430.h>

#define REDLED 0x01                    // Mask for BIT0 = 0000_0001b

void main(void)
{
        WDTCTL = WDT_ADLY_1000; // Set watchdog timer to interval mode (1 second)
        SFRIE1 |= WDTIE; // Enable watchdog timer interrupt
```

```
        _BIS_SR(LPM0_bits + GIE); // Enter low-power mode 0 and enable global interrupts
}

#pragma vector = WDT_VECTOR
__interrupt void watchdog_timer(void)
{
        static unsigned int buzz = 0;

        if (buzz == 0)
        {
        P2DIR |= BIT2; // buzzer configuration at 2.4
        P2SEL |= BIT2; // more buzzer configuration at 2.4


        P1DIR |= REDLED; // Set LED1 as output
        P1REN |= BIT1; // Enable the pull-up resistor at P1.1
        P1OUT |= REDLED; // turn on Red LED

        TBCCR0 = 1048; // Set the period for the buzzer (1KHz)
        TBCCR1 = 524; // Set the duty cycle for the buzzer (50%)

        TBCCTL1 = OUTMOD_7; // reset/set mode
        TBCTL = TBSSEL_2 + MC_1 + ID_0; // SMCLK, up mode, no division

        buzz = 1; // change the state of the buzzer after the timer
        }
        else
        {
        P2SEL &= ~BIT2; // Turn off the buzzer
        P1OUT &= ~REDLED; // turn off red LED

        buzz = 0;
        }
}
```