Embedded Systems Lab

CPE 325-02

Timers on the MSP430

By: David Thornton

Lab Date: October 8, 2020

Lab Due: October 21, 2020

Demonstration Due: October 21, 2020

Introduction

This lab introduces the concept of timers with the MSP430.

Theory

Topic 1: Watchdog Timer

- During normal operation, the computer regularly resets the watchdog timer to prevent it from elapsing, or "timing out". If, due to a hardware fault or program error, the computer fails to reset the watchdog, the timer will elapse and generate a timeout signal. The timeout signal is used to initiate corrective actions. The corrective actions typically include placing the computer system in a safe state and restoring normal system operation.

Topic 2: Timers

- “A timer is a specialized type of clock which is used to measure time intervals. A timer that counts from zero upwards for measuring time elapsed is often called a stopwatch. It is a device that counts down from a specified time interval and used to generate a time delay, for example, an hourglass is a timer. A counter is a device that stores (and sometimes displays) the number of times a particular event or process occurred, with respect to a clock signal. It is used to count the events happening outside the microcontroller. In electronics, counters can be implemented quite easily using register-type circuits such as a flip-flop.” - Source

Topic 3: Timer Mode Control

Table 1-1. Timer Modes

MC	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TAxCCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh.
11	Up/down	The timer repeatedly counts from zero up to the value of TAxCCR0 and back down to zero.

Source

Lab Assignment

1. Pulse width modulation (PWM) can be used to control power supplied to some devices from the microcontroller. For example, using PWM you can change the brightness of your LEDs. If the frequency of the PWM signal supplied to an LED is high enough, you will not notice any blinking, but will be able to adjust the brightness by changing the duty cycle of the signal. Write a C program that implements the following functionality:
    a. LED1 is initially off. Then it starts gradually turning on until it gains the full brightness. Upon that it should start gradually turning off. Then the cycle repeats. The time between fully on and fully off states should be 3 seconds (full cycle takes 6 seconds).
    b. SW1 should stop the brightness change process. After it is pressed, LED1 should keep its current brightness.
    c. SW2 should start the brightness change process from the point where it was stopped by SW1. You do not have to use interrupts to process switches.
    d. Use TimerA to produce the PWM signal for controlling LED1 brightness. Make it control the LED. Choose appropriate counting and output modes to do that.
    e. Use interrupts from the watchdog timer in the interval mode to change LED1 brightness.
    f. The process of changing brightness should be smooth. That is, the transition between adjacent brightness levels should be barely noticeable.

2. Write a program in C that achieves the following:
    a. Initializes watchdog timer to raise interrupt every 0.5 mili-seconds.
    b. On every 1000th interrupt you raise an interrupt on the pin where SW1 is interfaced.
    c. The ISR for SW1 toggles LED2.

Observations

All code satisfies the lab requirements. For question 2, since we are delaying for 0.5 ms 1000 times, that results in 0.5 seconds per toggle. Thus, LED2 blinks at 1Hz in question 2.

Conclusion

This lab was successful in introducing me to using timers on the MSP430.
Demo link

Appendix

Appendix 1: Lab_07_Q1

```
/*-------------------------------------------------------------
* File:           Lab_7_Q1.c
* Description:    This program increases and decreases the
*                       brightness of LED1 at a frequency of .16 Hz
* Input:          S1 and S2
* Output:         LED1 gradually changes intensity, and can
*                       pause depending on button presses.
* Author:         David Thornton
* Lab Section:    2
* Date:           October 21, 2020
* *-------------------------------------------------------------*/
#include <msp430.h>
#define SW1 ((P2IN&BIT1) == 0)      // Define switch 1
#define SW2 ((P1IN&BIT1) == 0)      // Define switch 2
#define REDLED 0x01                         // LED1 - Mask for BIT0 (0000_0001b)

void main(void)
{
      _EINT();                              // Enable interrupts
      WDTCTL = WDT_MDLY_32;         // Set interval to 32 milliseconds
      SFRIE1 |= WDTIE;

      P1DIR |= REDLED;              // Set LED1 as output
      P1OUT &= ~REDLED;             // Turn LED1 off at start

      P2DIR &= ~BIT1;                       // SW1 direction
      P2REN |= BIT1;                        // Enable the pull-up resistor at P2.1
      P2OUT |= BIT1;                        // Set SW1 output to on

      P1DIR &= ~BIT1;                       // SW2 direction
      P1REN |= BIT1;                        // Enable the pull-up resistor at P1.1
      P1OUT |= BIT1;                        // Set SW2 output to on

      TA0CCTL0 = CCIE;              // TA0 count triggers interrupt
      TA0CCR0 = 96;                         // Set TA0 count value

      TA0CCTL1 = CCIE;              // TA0.1 count triggers interrupt
```

```
	TA0CCR1 = 96;                           // Set TA0.1 count value

	TA0CTL = TASSEL_1 | MC_3;     // ACLK is clock source, UP/DOWN mode

	int i = 0;                              // Used in for loops for debouncing
	while(1)
	{
		if(SW1)
		{
			for(i = 2000; i >= 0; i--); // Debounce
			if (SW1)
			{
				WDTCTL = WDTPW + WDTHOLD; // Stop WDT
			}
		}
		if(SW2)
		{
			for(i = 2000; i >= 0; i--); // Debounce
			if(SW2)
			{
				WDTCTL = WDT_MDLY_32;   // Resume interval
			}
		}
	}
}


#pragma vector = WDT_VECTOR
__interrupt void wdtISR(void)
{
	static int counter = 96;
	static int pwm = 1;
	if(pwm == 0 )
	{
		if(counter == 96)
		{
			pwm = 1;
		}
		counter++;
		TA0CCR1 = counter;
	}
	if(pwm == 1)
```

```
	{
		if(counter == 0)
		{
			pwm = 0;
		}
		counter--;
		TA0CCR1 = counter;
	}
}


#pragma vector = TIMER0_A0_VECTOR
__interrupt void timerISR(void)
{
	P1OUT |= REDLED;
	TA0CCTL0 &= ~CCIFG;
}


#pragma vector = TIMER0_A1_VECTOR
__interrupt void timerISR2(void)
{
	P1OUT &= ~REDLED;
	TA0CCTL1 &= ~CCIFG;
}
```

Appendix 2: Lab_07_Q2

```
/*-------------------------------------------------------------
* File:           Lab_7_Q2.c
* Description:    This program uses the WDT to blink LED2
* Input:          None
* Output:         Blinking LED2
* Author:         David Thornton
* Lab Section:    2
* Date:           October 21, 2020
* *-------------------------------------------------------------*/
#include  <msp430.h>

#define GREENLED 0x80                   // LED2 - Mask for BIT7 (1000_0000b)

void main(void)
```

```
{
      WDTCTL = WDT_MDLY_0_5;              // 0.5 ms interrupt
      P4DIR |= GREENLED;                        // Configure the P4.7 as output.
      _EINT();                                  // Enable interrupts
      SFRIE1 |= WDTIE;                    // Enable WDT interrupt

      P2IE  |= BIT1;                            // Enable interrupt at P2.1 for S2
      P2IES |= BIT1;                            // Enable hi->lo edge for interrupt
      P2IFG &= ~BIT1;                           // Clear any errornous interrupt
flag

      _BIS_SR(LPM0_bits + GIE);           // Enter LPM0 with interrupt
}


// SW1 key press ISR
#pragma vector = PORT2_VECTOR
__interrupt void PORT2_ISR(void)
{
      P2IFG &= ~BIT1;                           // Clear the flag
      P4OUT ^= GREENLED;                        // Toggle LED
}


// WDT ISR
#pragma vector = WDT_VECTOR
__interrupt void watchdog_timer(void)
{
      static int i = 0;
      i++;
      if (i == 1000)
      {
            P2IFG |= BIT1;    // Clear the flag
            i = 0;                  // Reset i
      }
}
```