CPE 325: Embedded Systems Laboratory
Laboratory #7 Tutorial
MSP430 Timers, Watchdog Timer, Timers A and B

Aleksandar Milenković
Email: milenka@uah.edu
Web: http://www.ece.uah.edu/~milenka

Objective

This tutorial will introduce the watchdog timer (WDT) and the MSP430's TimerB module. You will learn the following topics:

Watchdog timer and its interval mode
Configuration of the peripheral device Timer_B
How to utilize Timer_B operating modes to solve real-world problems

Notes

All previous tutorials are required for successful completion of this lab. Especially, the tutorials introducing the TI Experimenter's Board and the Code Composer Studio software development environment.

Contents

1 Watchdog Timer: Toggling a LED Using Interval Mode ISR

Embedded computer systems usually have at least one timer peripheral device. You can think of timers as simple digital counters that in active mode increment or decrement their value at a specified clock frequency. Before using a timer in our application we need to initialize it by setting its control registers. During initialization we need to specify timer's operating mode (whether they increment or decrement their value on each clock, pause, etc.), the clock frequency, and whether it will raise an interrupt request once the counter reaches zero (or a predetermined value set by software). Timers may have comparison logic to compare the timer value against a specific value set by software. When the values match, the timer may take certain actions, e.g., rollover back to zero, toggle its output signal, to name just a few possibilities. This might be used, for example to generate pulse width modulated waveforms used to control the speed of motors. Similarly, timers can be configured to capture the current value of the counter when a certain event occurs (e.g., the input signal changes from logic zero to logic one). Timers can also be used to trigger execution of the corresponding interrupt service routine. The MSP430 family supports several types of timer peripheral devices, namely the Watchdog Timer, Basic Timer 1, Real Time Clock, Timer A, and Timer B. Here we will learn more about the watchdog timer and how it can be used to periodically blink a LED.

The primary function of the watchdog-timer module (WDT) is to perform a controlled-system restart after a software problem occurs. If the selected time interval expires, a system reset is generated. If the watchdog function is not needed in an application, the module can work as an interval timer, to generate an interrupt after the selected time interval. Figure 1 illustrates a block diagram of the watchdog timer peripheral. It features a 16-bit control register, WDTCTL, and a 16-bit counter, WDTCNT. The watchdog timer counter (WDTCNT) is a 16-bit up-counter that is not directly accessible by software. The WDTCNT is controlled and time intervals selected through the watchdog timer control register WDTCTL. The WDTCNT can be sourced from either ACLK or SMCLK. The clock source is selected with the WDTSSEL bit.

Setting the WDTTMSEL bit to 1 selects the interval timer mode. This mode can be used to provide periodic interrupts. In interval timer mode, the WDTIFG flag is set at the expiration of the selected time interval. A PUC is not generated in interval timer mode at expiration of the selected timer interval and the WDTIFG enable bit WDTIE remains unchanged.
When the WDTIE bit and the GIE bit are set, the WDTIFG flag requests an interrupt. The WDTIFG interrupt flag is automatically reset when its interrupt request is serviced, or may be reset by software. The interrupt vector address in interval timer mode is different from that in watchdog mode. Our goal is to configure the watchdog timer in the interval timer mode and use its interrupt service routine for blinking the LED. Let us assume that we want to have the LED on for 1 sec and off for 1 second (the period is 2 seconds of 0.5 Hz).

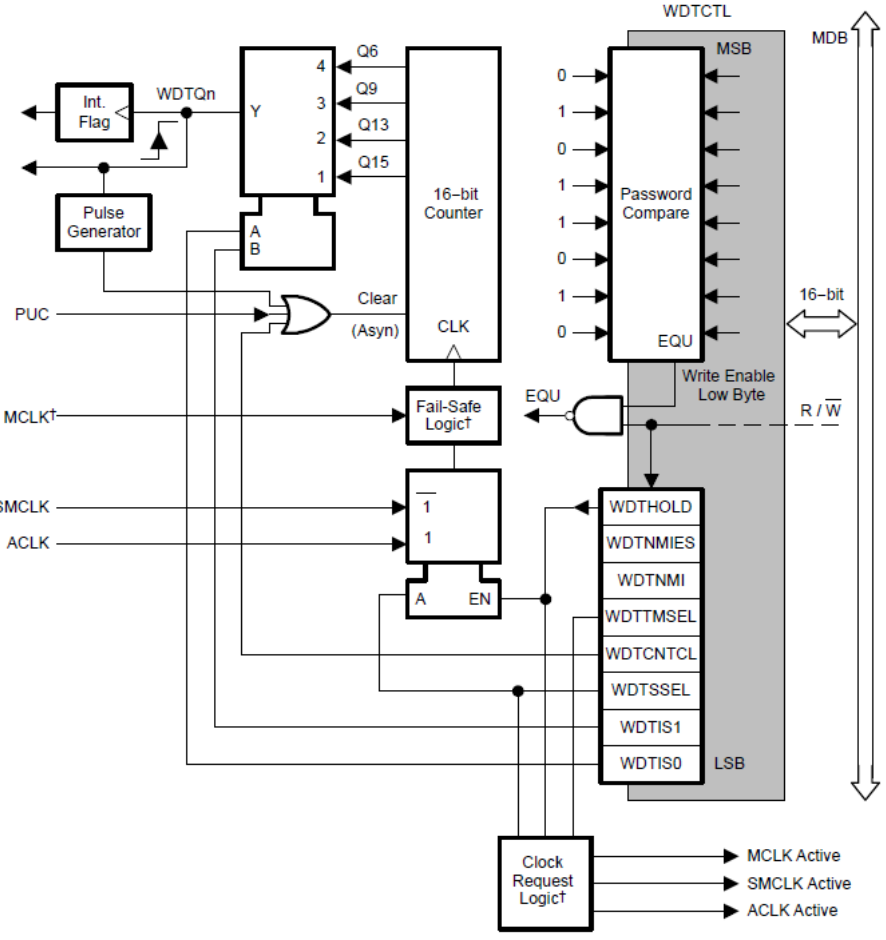

Figure 1. Block Diagram of the Watchdog Timer

Let us first consider how to specify time interval for the watchdog timer. If we select the ACLK clock for the clock source (SSEL = 0: WDTCNT is clocked by SMCLK; SSEL = 1: WDTCNT is clocked by ACLK), the timer clock is ~1 MHz.

Figure 2 shows a program that toggles a LED in the watchdog timer interrupt service routine every second. To generate an interrupt request every second, we configure the WDT as follows: select the ACLK as the clock source, ACLK=32,768 Hz (or 2^{15} Hz), and select the tap to be 2^{15}; the WDT interval time will be exactly 1 second. The WDT control word will look like this: WDTMSEL selects interval mode, WDTSSEL selects ACLK, and WTTCNTCL clears the WDTCNT. Analyze the header file msp430xG46x.h to locate pre-defined command words for the control register (e.g., WDT_ADLY_1000, WDT_ADLY_250, …).

```
/*------------------------------------------------------------------------------
 * File:        Lab7_D1.c (CPE 325 Lab7 Demo code)
 * Function:    Toggling LED1 using WDT ISR (MPS430FG4618)
 * Description: This C program configures the WDT in interval timer mode and
 *              it is clocked with ACLK. The WDT is configured to give an
 *              interrupt for every 1s. LED1 is toggled in the WDT ISR
 *               by xoring P2.2. The blinking frequency of LED1 is 0.5Hz.
 * Clocks:       ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO = default (~1MHz)
 *               An external watch crystal between XIN & XOUT is required for ACLK
 *
 *                        MSP430xG461x
 *                     -----------------
 *                 /|\|              XIN|-
 *                  | |                 | 32kHz
 *                  --|RST          XOUT|-
 *                    |                 |
 *                    |             P2.2|-->LED1(GREEN)
 *                    |                 |
 * Input:        None
 * Output:       LED1 blinks at 0.5Hz frequency
 * Author:       Aleksandar Milenkovic, milenkovic@computer.org
 * Date:         December 2008
 *------------------------------------------------------------------------------*/
#include <msp430xG46x.h>

void main(void) {
    WDTCTL = WDT_ADLY_1000;           // 1 s interval timer
    P2DIR |= BIT2;                    // Set P2.2 to output direction
    IE1 |= WDTIE;                     // Enable WDT interrupt
    _BIS_SR(LPM0_bits + GIE);         // Enter LPM0 w/ interrupt
}

// Watchdog Timer Interrupt Service Routine
#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void) {
    P2OUT ^= BIT2;                    // Toggle P2.2 using exclusive-OR
}

```

Figure 2. Toggling a LED using WDT_ISR

Figure 3 shows the program that also toggles the LED1 every second in the WDT ISR. However, the watchdog timer uses the SMCLK as the clock source and the maximum tap of 32,768 (2^{15}). The WDT generates an interrupt request every 32 ms. To toggle the LED1 every second we need to use a static local variable that is incremented every time we enter the ISR. When we collect 32 periods of 32 ms we have approximately 1 second of elapsed time, and we can then toggle LED1. Analyze the header file msp430xG46x.h to locate pre-defined command word WDT_MDLY_32. What bits of the control register are set with WDT_MDLY_32? What is the purpose of using static variable in the ISR? What happens if we use normal variable?

```
/*------------------------------------------------------------------------------
 * File:         Lab7_D2.c (CPE 325 Lab7 Demo code)
 * Function:     Toggling LED1 using WDT ISR (MPS430FG4618)
 * Description: This C program configures the WDT in interval timer mode and
 *               it is clocked with SMCLK. The WDT is configured to give an
 *               interrupt for every 32ms. The WDT ISR is counted for 32 times
 *               (32*32ms = 1sec) before toggling LED1 to get 1 s on/off.
 *               The blinking frequency of LED1 is 0.5Hz.
 * Clocks:       ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO = default (~1MHz)
 *              An external watch crystal between XIN & XOUT is required for ACLK
 *
 *                          MSP430xG461x
 *                       -----------------
 *                    /|\|             XIN|-
 *                     | |                | 32kHz
 *                     --|RST         XOUT|-
 *                       |                |
 *                       |            P2.2|-->LED1(GREEN)
 *                       |                |
 * Input:       None
 * Output:      LED1 blinks at 0.5Hz frequency
 * Author:      Aleksandar Milenkovic, milenkovic@computer.org
 * Date:        December 2008
 *------------------------------------------------------------------------------*/
#include <msp430xG46x.h>

void main(void) {
    WDTCTL = WDT_MDLY_32;           // 32ms interval (default)
    P2DIR |= BIT2;                  // Set P2.2 to output direction
    IE1 |= WDTIE;                   // Enable WDT interrupt

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

// Watchdog Timer Interrupt Service Routine
#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void) {
    static int i = 0;
    i++;
    if (i == 32) {                  // 31.25 * 32 ms = 1s
        P2OUT ^= BIT2;              // Toggle P2.2 using exclusive-OR
                                    // 1s on, 1s off; period = 2s, f = 1/2s = 0.5Hz
        i = 0;
    }
}

```

Figure 3. Toggling the LED1 using WDT_ISR

Figure 4 shows the program that also toggles LED1 every second. The WDT is still configured in the interval mode and sets the WDTIFG every 1s. The program however does not use the interrupt service routine (the interrupt from WDT remains disabled). Instead, the main program polls repeatedly the status of the WDTIFG. If it is set, LED1 is toggled and the WDTIFG is cleared. Otherwise, the program checks the WDTIFG status again. The program spends majority of time waiting for the flag to be set and this approach is known as software polling. It is inferior to using interrupt service routines, but sometimes can be used to interface various peripherals. What are the advantages of using interrupts over software polling?

```
/*------------------------------------------------------------------------------
```

```
 * File:         Lab7_D3.c (CPE 325 Lab7 Demo code)
 * Function:     Toggling LED1 using software polling.
 * Description: This C program configures the WDT in interval timer mode and
 *              it is clocked with ACLK. The WDT sets the interrupt flag (WDTIFG)
 *              every 1 s. LED1 is toggled by verifying whether this flag
 *              is set or not. After it is detected as set, the WDTIFG is cleared.
 * Clocks:      ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO = default (~1MHz)
 *              An external watch crystal between XIN & XOUT is required for ACLK
 *
 *                          MSP430xG461x
 *                     -----------------
 *                 /|\|              XIN|-
 *                  | |                 | 32kHz
 *                  --|RST          XOUT|-
 *                    |                 |
 *                    |             P2.2|-->LED1(GREEN)
 *                    |                 |
 * Input:       None
 * Output:      LED1 blinks at 0.5Hz frequency
 * Author:      Aleksandar Milenkovic, milenkovic@computer.org
 *------------------------------------------------------------------------------*/
#include <msp430xG46x.h>

void main(void) {
    WDTCTL = WDT_ADLY_1000;           // 1 s interval timer
    P2DIR |= BIT2;                    // Set P2.2 to output direction

    for (;;) {
        // Use software polling
        if ((IFG1 & WDTIFG) == 1) {
            P2OUT ^= BIT2;
            IFG1 &= ~WDTIFG;          // Clear bit WDTIFG in IFG1
        }
    }
}

```

Figure 4. Toggling LED1 using WDT and Software Polling on WDTIFG

2 Timers (A and B)

MSP430 family supports several types of timer peripheral devices, namely the Watchdog Timer, Basic Timer 1, Real Time Clock, Timer A, and Timer B. In this part of the tutorial we will be studying Timer A and B.

Among the several ways to perform periodic tasks, one is to have a software delay. Using a software delay keeps the processor running while it is not needed, which leads to wasted energy. Further, counting clock cycles and instructions is quite cumbersome. MSP430s have peripheral devices called timers that can raise interrupt requests regularly. It is possible to use these timers to perform periodic tasks. Since a timer can use a different clock signal than the processor, it is possible either to turn off the processor or to work on other computations while the timer is counting. This saves energy and reduces code complexity. Note that a MSP430 can have multiple timers and each timer can be utilized independently from the other timers. Furthermore, each timer has several modes of counting. Though in this lab we will be using Timer B to blink a LED at regular interval of time, keeping time is not the only use of timers.

Figure 5 shows a block diagram of Timer B peripheral. It consists of a timer block (TBR) and seven capture and compare blocks (CCR0 – CCR6). The timer block can be configured to act as a 8-bit, 10-bit, 12-bit, or 16-bit counter and supports 4 counting modes: STOP (MCx=00), UP (MCx=01), Continuous (MCx=10), and UP/DOWN (MCx=11). The source clock can be selected among multiple options (TBBSELx bits), and the selected clock can be further divided by 1 (IDx=00), 2 (IDx=01), 4 (IDx=10), and 8 (IDx=11). In the UP and UP/DOWN modes counter counts up to the value that is specified by the TBCCR0 register. The timer block is configured using Timer B control register, TBCTL, which contains control bits TBBSELx, IDx, CNTLx, and others. Please examine the format of this register.

Each Capture and Compare block contains a 16-bit latch register TBCCRx and corresponding control logic enabling two type of operations CAPTURE and COMPARE. Each capture and compare block is controlled by its own control register, TBCCTLx.

Capture refers to an operation where the value of the running counter (TBR) is captured in the TBCCRx on a hardware event (e.g., external input changes its state from a logic 1 to a logic 0 or from a logic 0 to a logic 1) or on a software trigger (e.g., setting some control bits). This operation is very useful when we want to timestamp certain events. Imagine you are designing a system that needs to log an exact moment when a car enters a parking lot. A sensor can trigger a change of state of an input that further triggers the capture operation on Timer B. This way we can precisely timestamp the event down to a single clock cycle precision with no software-induced delay. The value of the running timer is captured by the TBCCRx that can then be read and processed in software.

Compare refers to an operation where actions are triggered at specific moments in time. This operation is crucial for generating Pulse-Width-Modulated signals (PWMs) – periodic signals where duty cycles is fully controlled: the period the signal is on over the entire period. The PWM type of signals are often used in robotics to control motors. In compare mode of operation the corresponding latch register, TBCCRx, is initialized to a certain value. When the value in the running counter (TBR) reaches the value in TBCCRx, an output signal can change its state (set, reset, or toggle).

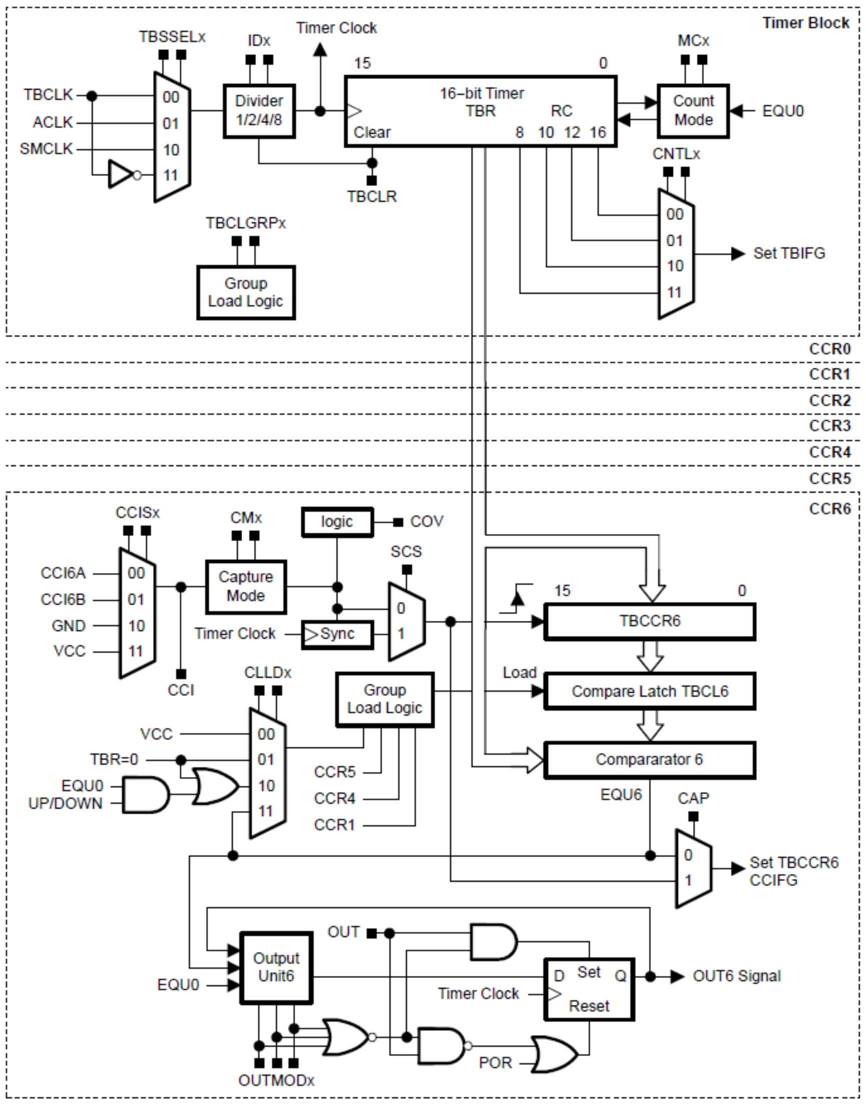

Figure 5. TimerB Block Diagram

2.1 Toggle a LED Using Timer B

Let us first consider an example where we utilize a Timer B device to toggle LED2 on the TI Experimenter's board (LED2 is connected to P2.1). LED2 should be periodically turned on for 0.065 seconds and then turned off for 0.065 seconds (one period is ~0.13 seconds, or toggling rate is ~7.6 Hz). We have learned how to execute the toggling using a software delay or by using the watchdog timer. Now, we would like to utilize the MSP430's Timer B peripheral device.

Figure 7 shows a program that toggles LED2 as specified. We can see that the port 2.1 is multiplexed with TB0 special function, which is the output from the capture and compare block 0 (TB0CC0). Note: how do we know this? Find a document where this information is available? How to configure Port 2.1 to output TB0?

The capture and compare block 0 can be configured to set/reset or toggle the output signal TB0 when the value in the running counter reaches the value in the capture and control register TBCCR0. Thus, when a value in the Timer B counter is equal to the value in TBCCR0, we can configure the Timer B to toggle its output, TB0, automatically (EQU0 control signal will be activated). The default value in TBCCR0 is 0, thus, the output will be toggled every time the counter rolls over to 0x0000. However, before we can use TB0 as an output on P2.1, we need to configure the port 2 selection register, P2SEL, pin 1 to its special I/O function instead of its common digital I/O function (P2SEL |= BIT1;). This way we ensure that the P2.0 mirrors the behavior of the TB0 signal.

The next step is to configure clock sources. The MSP430 clocks MCLK, SMCLK, and ACLK have default frequencies as follows: MCLK = SMCLK ~ 1MHz and ACLK = 32 KHz. Timer B is configured to use the SMCLK as its clock input and to operate in the continuous mode. Timer B's counter will count from 0x0000 to 0xFFFF. When the counter value reaches 0x0000, the EQU0 will be asserted indicating that the counter has the same value as the TBCCR0 (here it is not set because by default it is cleared). We can select the output mode 4 (toggle) that will toggle the output every time EQU0 is asserted. This way we can determine the time period when the TB0 is reset and set. The TB0 will be set for 65,536*1/2^20 = 0.0625 seconds and will be reset for 65,536*1/2^20 = 0.0625 seconds. Please note that we do not need to use an interrupt service routine to toggle the LED2 in this case. The TimerB will toggle the LED2 independently, and we can go into a low power mode and remain there for the rest of the application lifetime. What are the different low power modes available in MSP430FG4618? How do they differ from each other? Explain your answers.

```
/*------------------------------------------------------------------------------
 * File:        Lab7_D4.c (CPE 325 Lab7 Demo code)
 * Function:    Blinking LED2 using Timer_B in continuous mode (MPS430FG4618)
 * Description: In this C program, Timer_B is configured for continuous mode. In
 *              this mode, the timer TB counts from 0 up to 0xFFFF (default 2^16).
 *              So, the counter period is 65,536*1/2^20 = 62.5 ms. The TB0 output
 *              signal is configured to toggle every time the counter reaches the
 *              maximum value, which corresponds to 65 ms. TB0 is multiplexed with
 *              the P2.1, and LED2 is luckily connected to this output. Thus the
 *              output frequency on P2.1 will be f = SMCLK/(2*65536) ~ 8 Hz.
 *              Please note that once configured, the Timer_B toggles the LED2
 *              automatically even when the CPU is in sleep mode.
 * Clocks:      ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO = default (2^20 Hz)
 *              An external watch crystal between XIN & XOUT is required for ACLK
 *
 *                        MSP430xG461x
 *                     -----------------
 *                 /|\|              XIN|-
 *                  | |                 | 32kHz
 *                  --|RST          XOUT|-
 *                    |                 |
 *                    |         P2.1/TB0|-->65536/SMCLK (LED2(YELLOW))
 *                    |                 |
 * Input:       None
 * Output:      LED2 blinks at 8Hz frequency using hardware TB0
 * Author:      Aleksandar Milenkovic, milenkovic@computer.org
 *------------------------------------------------------------------------------*/
#include <msp430xG46x.h>

void main(void)
{
    WDTCTL = WDTPW +WDTHOLD;  // Stop WDT
    P2DIR |= BIT1;            // P2.1 output
    P2SEL |= BIT1;            // P2.1 special function (TB0 output)
    TB0CCTL0 = OUTMOD_4;      // TB0 output is in toggle mode
    TB0CTL = TBSSEL_2 + MC_2; // SMCLK is clock source, Continuous mode
    _BIS_SR(LPM0_bits + GIE); // Enter Low Power Mode 0
}
```

Figure 6. C Program for Toggling LED2 Using TimerB, Continuous Mode

Try to modify the code from Figure 6 by selecting a different source clock for Timer B. What happens if we use the following command: TB0CTL = TBSSEL_1 + MC_2? What is the period of toggling the LED2? Explain your answer. Try using divider for the clock source.

The given example may not be suitable if you want to control the period of toggling since the counter in the continuous mode always counts from 0x0000 to 0xFFFF. This problem can be solved by opting for the UP counter mode. The counter will count from 0x000 up to the value specified in the TBCCR0. This way we can control the time period. Let us consider an example where we want LED2 to be 1 second on and 1 second off (toggling rate is 0.5 Hz).

Figure 7 shows the C code for this example. Note the changes. How do we specify UP mode? How do we select the ACLK clock as the TimerB source clock? What output mode do we use? Is it better to use ACLK instead of SMCLK in this example? Explain your answers.

```
/*------------------------------------------------------------------------------
 * File:        Lab7_D5.c (CPE 325 Lab7 Demo code)
 * Function:    Blinking LED2 using Timer_B in up mode (MPS430FG4618)
 * Description: In this C program, Timer_B is configured for up mode. In this
 *              mode, the timer TB counts from 0 up to value stored in TB0CCR0.
 *              So, the counter period is CCR0*1us. The TB0 output signal is
 *              configured to toggle every time the counter reaches the value
 *              in TB0CCR0. TB0 is multiplexed with the P2.1, and LED2 is luckily
 *              connected to this output. Thus, the output frequency on P2.1 will
 *              be f = ACLK/(2*CCR0) = 0.5Hz. Please note that once configured,
 *              the Timer_B toggles the LED2 automatically even when the CPU is
 *              in sleep mode.
 * Clocks:      ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO = default ((2^20 Hz)
 *              An external watch crystal between XIN & XOUT is required for ACLK
 *
 *                          MSP430xG461x
 *                       -----------------
 *                   /|\|              XIN|-
 *                    | |                 | 32kHz
 *                    --|RST          XOUT|-
 *                      |                 |
 *                      |         P2.1/TB0|--> 32768/ACLK (LED2(YELLOW))
 *                      |                  |
 * Input:       None
 * Output:      LED2 blinks at 0.5Hz frequency using hardware TB0
 * Author:      Aleksandar Milenkovic, milenkovic@computer.org
 *------------------------------------------------------------------------------*/
#include <msp430xG46x.h>

void main(void) {
    WDTCTL = WDTPW +WDTHOLD;    // Stop WDT
    P2DIR |= BIT1;              // P2.1 output
    P2SEL |= BIT1;              // P2.1 special function (TB0 output)
    TB0CCTL0 = OUTMOD_4;        // TB0 output is in toggle mode
    TB0CTL = TBSSEL_1 + MC_1;   // ACLK is clock source, UP mode
    TB0CCR0 = 32767;
    _BIS_SR(LPM3_bits + GIE);   // Enter Low Power Mode 3
}
```

Figure 7. C Program for Toggling LED2 Using TIMER_B (UP MODE)

2.2 Additional Timer_B Functionality

As already seen, Timer B is quite powerful due to the selectable clocks, automated outputs, and adjustable maximum count value. The Timer B peripheral has additional features which greatly expand its functionality and versatility. These features include:

- Multiple capture/compare modules
- Multiple output control modes
- Ability to call multiple interrupts at different count values
- Ability to select from multiple counting modes

Often times, it will be necessary to perform multiple tasks with a single timer peripheral. Fortunately, the Timer B system has multiple channels that can be set up to perform their tasks at designated count values. The MSP430FG4618 has 7 of these channels (the 0 channel, which is basically the master channel, and 6 additional channels). Each channel normally has a shared output pin similar to what we saw with the TB0 pin in the examples above. In Figure 8 below, note that LED1 is on the TB1 output. This means that capture/compare channel 1 can control it automatically just as we saw LED2 being controlled earlier.

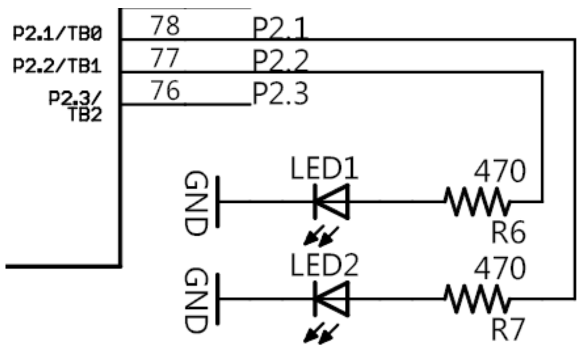

Figure 8. LED1 on TB1 and LED2 on TB0

If you further examine the MSP430 experimenter board schematic, you can find where the other 6 channel output pins are located. In order to use channels 1 – 6, channel 0 must still be set up. Channels 1 – 6 are set up the same as channel 0, all of them have their own configuration register and their own count value register. Each channel can be configured to use its output pin, but they can also be used to call interrupt service routines. An interrupt vector is dedicated to channel 0, but channels 1-6 can be configured to call a separate ISR.

It is important to think about how the counting methods affect the interrupt calls from the different capture/compare channels. The user's guide contains definitive information about the Timer B including examples that demonstrate how the various functions work. In general, the counting modes work as follows:
Counting mode 0 – Stop mode – The timer is inactive
Counting mode 1 – Up mode – The timer counts up to the value for channel 0. An interrupt for each channel set up is called at the corresponding count value on the way up. At the maximum value an interrupt for channel 0 is called, and at the next timer count a general interrupt is generated. Remember that these interrupts may correspond to output pin control or interrupt service routine vectoring depending on the channel configuration.
Counting mode 2 – Continuous mode – The timer counts up to its maximum value (65535 for 16 bit mode). Along the way, corresponding interrupts are called at each channel's count value register. At 0 a general Timer B interrupt is set.
Counting mode 3 – Up/Down mode – The timer counts up to the value in the channel 0 register, and then it counts back down to 0 again. The channel interrupts are called when the value is reach on the up count and the down count. The general Timer B interrupt is called when 0 is reached.

In Figure 9 below, channels 0 and 1 are used to call two separate ISRs. Since the timer is in up/down mode, the channel 0 ISR is only called once per counting cycle (on the max value set by the CCR0 register) while the channel 1 ISR is called twice per counting cycle if its CCR1 value is less than CCR0. It is called on the up count and the down count. This mode is especially useful when creating PWM signals since the count register value determines the duty cycle of the output signal.

```
/*------------------------------------------------------------------------------
 * File:        Lab7_D6.c (CPE 325 Lab7 Demo code)
 * Function:    Blinking LED1 & LED2 using Timer_B with interrupts (MPS430FG4618)
 * Description: In this C program, Timer_B is configured for up/down mode with
 *              ACLK source and interrupts for channel 0 and channel 1 are
 *              enabled. In up/down mode timer TB counts the value from 0 up to
 *              value stored in TB0CCR0 and then counts back to 0. The interrupt
 *              for TB0 is generated when the counter reaches value in TB0CCR0.
 *              The interrupt TB1 is generated whenever the counter reaches value
 *              in TB0CCR1. Thus, TB1 gets two interrupts while counting upwards
 *              and counting downwards. This simulates a PWM control - adjusting
 *              the TB1 and TB0 CCR register values adjusts the duty cycle of the
 *              PWM signal.
 * Clocks:      ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO = default (2^20Hz)
 *              An external watch crystal between XIN & XOUT is required for ACLK
 *
 *                          MSP430xG461x
 *                    -----------------
 *                 /|\|              XIN|-
 *                  | |                 | 32kHz
 *                  --|RST          XOUT|-
 *                    |                 |
 *                    |             P2.1|--> LED2(YELLOW)
 *                    |             P2.2|--> LED1(GREEN)
 * Input:       None
 * Output:      LED1 blinks at 1.64Hz with 20-80% duty cycle and LED2 blinks at
 *              0.82 Hz with 50-50% duty cycle.
 * Author:      Aleksandar Milenkovic, milenkovic@computer.org
 *------------------------------------------------------------------------------*/
#include <msp430xG46x.h>

void main(void)
{
    WDTCTL = WDTPW +WDTHOLD;      // Stop WDT
    _EINT();                      // Enable interrupts
    P2DIR |= BIT1 + BIT2;         // P2.1 and P2.2 set up as output
    P2OUT &= ~(BIT1 + BIT2);      // ensure LED1 and LED2 are off
    TB0CCTL0 = CCIE;              // TB0 count triggers interrupt
    TB0CCR0 = 10000;              // Set TB0 (and maximum) count value
    TB0CCTL1 = CCIE;              // TB1 count triggers interrupt
    TB0CCR1 = 2000;               // Set TB1 count value
    TB0CTL = TBSSEL_1 | MC_3;     // ACLK is clock source, UP/DOWN mode

    _BIS_SR(LPM3);                // Enter Low Power Mode 3
}

#pragma vector = TIMERB0_VECTOR
__interrupt void timerISR(void) {
    P2OUT ^= BIT1;                // Toggle LED2
}

#pragma vector = TIMERB1_VECTOR
__interrupt void timerISR2(void) {
    P2OUT ^= BIT2;                // Toggle LED1
    TBCCTL1 &= ~CCIFG;            // Clear interrupt flag
}
```

Figure 9. Code Using Multiple Timer B CC Channels and ISRs to Toggle LEDs

3 References

Getting started with the timers can be confusing at first. It is important to understand their functions and different operating modes. The following texts can help you understand the timers operation better:

- Chapter 8 in the Davies Text (page 275 – 368)
  - Watchdog Timer – page 276 – 281
  - Timer A – page 287 – 300

  - Timer B – page 353 – 356
- The user’s guide
  - Watchdog Timer – Chapter 12 (page 415 – 424)
  - Timer A – Chapter 15 (page 449 – 472)
- Timer B – Chapter 16 (page 473 – 498)