EEL4742C: Embedded Systems

Experiment Three:

Using the Timer

Ryan Koons 9/27/2020

Introduction: This lab will cover how to use “polling” to introduce the notion of timing into embedded systems. The MSP430 has two types of modes that can be used for timing—continuous mode and up mode. In continuous mode the TAR (counter) counts from 0 to (2^16)-1 and then starts over. Whenever, this TAR rolls back (in both modes) to zero a flag is raised. Programmers can take advantage of this flag raising (in both modes) to achieve delays/toggles. In up mode, the top value of the counter can be set using TA0CCRO, this allows the programmer to configure various periods in their code. Each cycle, the TAR is compared with the value of TA0CCR0; TAR won’t roll back until these values are equal. This is called the compare event. The MSP430 has ACLK (low freq~32kHz) and SMCLK (high freq ~1MHz) that can be used for polling. Each one has a different frequency and can also be divided down to create different periods for the code. By selecting various clocks, clock divisions, different counting modes, and taking advantage of flags, programmers can obtain a wide variety of different timing intervals.

3.1 Continuous Mode: The first part of this experiment consisting of configuring the Timer_A0 CTL register in order to toggle the red LED with a 2 second delay. After initializing the LEDs and disabling the watchdog timer, first a function needs to be written that redirects ACLK from the default to the 32kHz crystal on the MSP430 Launchpad. This function first sets PJSEL1.4 to 0 and PJSEL0.4 to 1. The Launchpad User’s guide shows that these pins can be reconfigured for LFXIN and LFXOUT in order to take advantage of the 32kHz crystal soldered to the launchpad. The rest of the function unlocks the register and then waits for the crystal to settle (via global flags).

The timer A is then configured tin the code by using an ‘|’ operator to set all of the configuration bits. “TA0CTL = TASSEL_1 | ID_0 | MC_2 | TACLR” selects ACLK (32Khz crystal) as the clock source, divides the frequency by 1, selects Continuous mode, and TACLR starts at 0. Then the TAIFG is cleared from the start.

The last part of the code is an infinite loop that checks if TAIFG has been raised. If it has, then the LED toggles, and the flag must be cleared by the programmer to trigger successive detections.

3.1 Analysis: Expected delay = 65535 cycles/32000 cycles/s = 2.05 seconds. Very close to 20 seconds on iPhone. ID_0: 2.05s ID_1: 4.096s ID_2: 8.19s ID_3: 16.38s (match expected).

.

3.1 CCS Code: LED toggle with 32kHz Crystal Delay

```
//3.1--Flash the red LED with Timer_A, continuous mode, via polling
#include <msp430.h>
#define redLED BIT0   //red LED at P1.0
```

```
#define greenLED BIT7 //green LED at  P9.7

//Function Prototype for 32Khz ACLK
void convert_32khz();



int main(void)
{
	WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer
	PM5CTL0 &= ~LOCKLPM5;       //clear lock and allow new code re-flash/enable
GPIO Pins

	P1DIR |= redLED;    //direct pin as output
	P9DIR |= greenLED;  //direct pin as output

	P1OUT &= ~redLED;  //turn LED OFF
	P9OUT &= ~greenLED;//turn LED off

	//Configure ACLK to 32kHz
	convert_32khz();

	//Configure Timer_A: ACLK, Divide by 1, Continuous Mode, clear TAR
	TA0CTL = TASSEL_1 | ID_0 | MC_2 | TACLR;
	TA0CTL &= ~TAIFG; //Clear flag at start

	for(;;){
	    //Wait here until TAIFG is raised
	    while( (TA0CTL&TAIFG) ==0 ){}

	    TA0CTL &= ~TAIFG; //clr flag
	    P1OUT ^= redLED; //toggle red LED
	}

	return 0;
}



//Function for Converting ACLK to 32Khz
void convert_32khz()
{
    //By default, ACLK runs on  LFMODCLK at 5Mhz/128= 39 Khz

    //Reroute pins to LFXIN/LFXOUT functionality
    PJSEL1 &= ~BIT4;
    PJSEL0 |= BIT4;

    //wait until oscillator fault flags are cleared
    CSCTL0 = CSKEY;  //unlock CS registers
    do {
        CSCTL5 &= ~LFXTOFFG;     //local fault flag
        SFRIFG1 &= ~OFIFG;       //global fault flag
    } while((CSCTL5 & LFXTOFFG) != 0 );
    CSCTL0_H = 0;   //Lock CS registers
    return;
}
```

3.2 Up Mode: This section of the lab involves creating a precise one second delay using a 32kHz crystal and up mode. After using the function “convert_32kHz()” to select the 32kHz crystal, the value of TACCR0 needs to be defined to create the “compare” event. A value of 32,000-1 will yield a 1 second delay at 32 kHz. The configuration register for timer A also needs to be set. “TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;” selects ACLK, with no clock division, and up mode selected, and it starts at zero. Then the TAIFG is cleared at the very beginning. The rest of the code (polling loop) is the same as the previous section.

3.2 Analysis: Was able to achieve precise one second delay. TACCR0=3,200-1 would achieve .1 second delay. TACCR0=320-1 to obtain .01 second delay. Observation: This value of TACCR0 is too small to actually see LED toggle.

3.2 CCS Code: Precise One Second Delay (LED Toggle)

```
//3.1--Flash the red LED with Timer_A, Up Mode, via polling
#include <msp430.h>
#define redLED BIT0   //red LED at P1.0
#define greenLED BIT7 //green LED at  P9.7

//Function Prototype for 32Khz ACLK
void convert_32khz();



int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;       //clear lock and allow new code re-flash/enable GPIO
Pins

    P1DIR |= redLED;    //direct pin as output
    P9DIR |= greenLED;  //direct pin as output

    P1OUT &= ~redLED;  //turn LED OFF
    P9OUT &= ~greenLED;//turn LED off

    //Configure ACLK to 32kHz
    //convert_32khz();

    //Set timer Period
    TA0CCR0 = (32000-1); // @32Khz, 1 second
    //Configure Timer_A: ACLK, Divide by 1, Up Mode, clear TAR
    TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
    TA0CTL &= ~TAIFG; //Clear flag at start

    for(;;){
        //Wait here until TAIFG is raised
        while( (TA0CTL&TAIFG) ==0 ){}
        TA0CTL &= ~TAIFG; //clr flag
        P1OUT ^= redLED; //toggle red LED
    }

    return 0;
}



//Function for Converting ACLK to 32Khz
void convert_32khz()
{
    //By default, ACLK runs on  LFMODCLK at 5Mhz/128= 39 Khz

    //Reroute pins to LFXIN/LFXOUT functionality
    PJSEL1 &= ~BIT4;
    PJSEL0 |= BIT4;

    //wait until oscillator fault flags are cleared
    CSCTL0 = CSKEY;  //unlock CS registers
    do {
        CSCTL5 &= ~LFXTOFFG;     //local fault flag
        SFRIFG1 &= ~OFIFG;       //global fault flag
    } while((CSCTL5 & LFXTOFFG) != 0 );

    CSCTL0_H = 0;   //Lock CS registers
    return;
}
```

3.3 Independent Design—Up Mode: This program effectively creates a stopwatch with a precise one second toggle in up mode. The stop watch has two set intervals; the second interval is two times the value of the first interval. The first interval is inputted by the programmer. In the example below, the two settings are 6 seconds and 12 seconds, respectively. This stopwatch initially starts at 0 seconds and counts up to the final value. Button one activates the lower setting, and button two activates the higher setting. Pressing and holding the opposite button that set the timing interval can also pause the stop watch until said button is released.

After initializing the code for the buttons and LEDs, convert32kHz() is called to select the 32 kHz crystal for ACLK. TACCR0 was assigned a value of 32,000-1, to create a one second delay at 32 kHz. The TA0CTL register is configured identically to part 2 of the lab. Two stopwatch variables are then initialized; these set the lower and upper settings for the stopwatch. The +1 is necessary to get the count to start at 0. Another integer is designed for being a counter during all of the loops in the code. Lastly, TAIFG is cleared.

When the infinite loop starts, both LEDs are OFF. The code checks to see if only button one is pressed (by using the largest bit mask) and if it is, the lowest setting starts. NOTE: due to the 'bouncing effect' of the buttons, it I advised that the user holds down the button that sets the timing interval for at least one second before releasing it. This ensures that the timer starts correctly and doesn't skip any cycles. A for loop executes that repeats for the amount of times of the lower stopwatch variable. During the first execution of the loop, the LED is off, this ensures that the toggle LED accurately displays the seconds remaining for the timing interval. Within this loop, a conditional statement checks to see if the other button is being pressed; if it is the green LED turns and the count pauses. Once this button is released, the code resumes, when time has elapsed a green LED turns on for a few seconds, before the code starts over again and another timing interval can be selected.

The code for the longer timing duration is identical to the previous (lower duration) code, however all of the buttons and their functionality are switched.

3.3 CCS Code:

```
//3.3--Stop Watch for set interval (2 settings) using Up Mode
#include <msp430.h>
#define redLED BIT0   //red LED at P1.0
#define greenLED BIT7 //green LED at  P9.7
#define BUT1 BIT1     //Button 1 at P1.1
#define BUT2 BIT2     //BUtton 2 at P1.2

//Function Prototype for 32Khz ACLK
void convert_32khz();



int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;       //clear lock and allow new code re-flash/enable GPIO
Pins

    P1DIR |= redLED;       //direct pin as output
    P9DIR |= greenLED;     //direct pin as output

    P1DIR &= ~(BUT1|BUT2);             //direct pins as inputs
    P1REN |= (BUT1 | BUT2);            //enable built-in resistor for P1.1 and P1.2
    P1OUT |= (BUT1 | BUT2);            //set resistor as pull-up to Vcc for P1.1 and
P1.2

    P1OUT &= ~redLED;  //turn LED OFF
    P9OUT &= ~greenLED;//turn LED OFF

    //Configure ACLK to 32kHz
    convert_32khz();

    //Set timer Period
    TA0CCR0 = (32000-1); // @32Khz, 1 second
    //Configure Timer_A: ACLK, Divide by 1, Up Mode, clear TAR
    TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
    TA0CTL &= ~TAIFG; //Clear flag at start
    unsigned int stopwatch_1 = (6)+1; //set stopwatch low duration (+1 to desired
value, takes a second to start the stopwatch)
    unsigned int stopwatch_2 = 2*(stopwatch_1 -1)+1; //set stopwatch high duration
for 2*(low duration)
    unsigned int i=0; //counter for loops


    for(;;){
            P1OUT &= ~redLED;  //turn LED OFF
            P9OUT &= ~greenLED;//turn LED OFF

        //if only switch 1 is pressed, activate low duration
        if((P1IN&(BUT1|BUT2)) == BUT2 )
                {
            P1OUT &= ~redLED;  //turn LED OFF
                for(i=0; i< stopwatch_1; i++)
                    {

                    //wait for flag
                    while( (TA0CTL&TAIFG) ==0 ){}

                    TA0CTL &= ~TAIFG; //clr flag
                    if(i==0)
                    P1OUT &= ~redLED;

                    P1OUT ^= redLED; //toggle red LED

                    if((P1IN&BUT2) ==0 )
                    {
                        P9OUT ^= greenLED;
                        while((P1IN&BUT2)==0){ }
                    }
                           P9OUT &= ~greenLED;
                          // i= i-1;
                    }
                P1OUT &= ~redLED;
                P9OUT |= greenLED;
                for(i=0; i<5; i++){
                    //wait for flag
                  while( (TA0CTL&TAIFG) ==0 ){}
                }

                }
        //Button 2 only is pressed
        if((P1IN&(BUT1|BUT2)) == BUT1 )
                        {
                    P1OUT &= ~redLED;  //turn LED OFF
                        for(i=0; i< stopwatch_2; i++)
                            {

                            //wait for flag
                            while( (TA0CTL&TAIFG) ==0 ){}

                            TA0CTL &= ~TAIFG; //clr flag
                            if(i==0)
                            P1OUT &= ~redLED;

                            P1OUT ^= redLED; //toggle red LED

                            if((P1IN&BUT1) ==0 )
                                                {
                                                    P9OUT ^= greenLED;
                                                    while((P1IN&BUT1)==0){ }
                                                }
                                                       P9OUT &= ~greenLED;

                            }
                        P1OUT &= ~redLED;
                        P9OUT |= greenLED;
                        for(i=0; i<5; i++){
                            //wait for flag
                          while( (TA0CTL&TAIFG) ==0 ){}
                        }

                        }
    }

    return 0;
}


//Function for Converting ACLK to 32Khz
void convert_32khz()
{
    //By default, ACLK runs on  LFMODCLK at 5Mhz/128= 39 Khz

    //Reroute pins to LFXIN/LFXOUT functionality
    PJSEL1 &= ~BIT4;
    PJSEL0 |= BIT4;

    //wait until oscillator fault flags are cleared
    CSCTL0 = CSKEY;  //unlock CS registers
    do {
        CSCTL5 &= ~LFXTOFFG;     //local fault flag
        SFRIFG1 &= ~OFIFG;       //global fault flag
    } while((CSCTL5 & LFXTOFFG) != 0 );

    CSCTL0_H = 0;   //Lock CS registers
    return;
}
```

3.4 Independent Design—Timer using Continuous Mode: This project creates a 2 interval timer (default—10s,20s) using the continuous mode and the SMCLK with clock division. Pressing button 1 turns ON the red LED for 10 seconds (this number is modifiable in the code. Pressing button 2 turns ON the red LED for two times the amount of the lower interval (i.e. 20 seconds).

When there are only 10 seconds left in the timer, the green LED briefly comes on. Additionally, pressing the opposite button of the one that selected the timing interval will add 10 seconds to the timer and flash the green LED to confirm the input. This program was designed to work similarly to a phone's timer app i.e. start at the requested input and count all the way down to zero, and then report that the timer is up 2 cycles later.

After initializing the LEDs and buttons, the configuration for the timer takes place. SMCLK is selected and divided down to yield a frequency of 125 kHZ. This project was done with the assumption that there is no 32kHz crystal on the board. Therefore, an approximate alternative is 2 cycles of the 125kHz in continuous mode; this results in a toggle of 1.04 seconds. Additionally the lower and upper timer interval variables are defined, as well as counters for the loops.

IF button one only is pressed, then the lower interval (10 seconds) is selected and the timer starts (red LED on). A line of code also checks if there are 10 seconds left and flashes the green LED. Within each execution of the for loop, the program "waits" for 2 cycles instead of one to obtain an approximate one second delay. If button one is pressed during this loop then, 20 cycles (10 seconds) is added to the timer variable and a green LED flashes, then the code resumes. Once the timer finishes, the green LED briefly turns on to show that time has elapsed.

The code for the longer duration is identical, except that the responsibilities of button one and button two are switched.

3.4 CCS Code:

```
//3.3--Stop Watch for set interval (2 settings) using Continuous Mode
//Feature: Add two different amounts of time to the   timer (kinda like a reset)
#include <msp430.h>
#define redLED BIT0   //red LED at P1.0
#define greenLED BIT7 //green LED at  P9.7
#define BUT1 BIT1     //Button 1 at P1.1
#define BUT2 BIT2     //BUtton 2 at P1.2


int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;       //clear lock and allow new code re-flash/enable GPIO
Pins

    P1DIR |= redLED;       //direct pin as output
    P9DIR |= greenLED;     //direct pin as output

    P1DIR &= ~(BUT1|BUT2);            //direct pins as inputs
    P1REN |= (BUT1 | BUT2);           //enable built-in resistor for P1.1 and P1.2
    P1OUT |= (BUT1 | BUT2);           //set resistor as pull-up to Vcc for P1.1 and
P1.2

    P1OUT &= ~redLED;  //turn LED OFF
    P9OUT &= ~greenLED;//turn LED OFF

    //Configure Timer_A: SMCLK, Divide by 8, Continuous Mode, clear TAR
    TA0CTL = TASSEL_2 | ID_3 | MC_2 | TACLR;
    TA0CTL &= ~TAIFG; //Clear flag at start
```

```
                    //put desired low value in parantheses for timer_1
    unsigned int timer_1 = 2*(10); //set stopwatch low duration (+1 to desired value,
takes a second to start the stopwatch) (each two toggles is one second)
    unsigned int timer_2 = 2*(timer_1); //set stopwatch high duration for 2*(low
duration)
    unsigned int i=0; //counter for loops
    unsigned int j=0; //counter for wait loops


    for(;;){
            P1OUT &= ~redLED;  //turn LED OFF
            P9OUT &= ~greenLED;//turn LED OFF

        //if only switch 1 is pressed, activate low duration
        if((P1IN&(BUT1|BUT2)) == BUT2 )
                {
            P1OUT |= redLED;  //turn LED ON
                for(i=0; i< timer_1; i++)
                    {
                    //wait for flag, wait 2 cycles.
                    for(j=0; j<2; j++){ while( (TA0CTL&TAIFG) ==0 ){} }

                    TA0CTL &= ~TAIFG; //clr flag
                    if(i==(timer_1)-20) //2cycles
                    {
                        P9OUT ^= greenLED;
                        for(j=0; j<4; j++){ while( (TA0CTL&TAIFG)==0){} }
                        P9OUT ^= greenLED;
                    }

                    if((P1IN&BUT2)==0)
                     {
                      //add 10 seconds to timer
                      timer_1+=20; //2 cycles per second
                      P9OUT ^= greenLED;
                      while((TA0CTL&TAIFG)==0){}
                      P9OUT ^=greenLED;
                     }

                    }
                P1OUT &= ~redLED;
                P9OUT |= greenLED;
                for(i=0; i<5; i++){
                    //wait for flag
                    for(j=0; j<2; j++){ while( (TA0CTL&TAIFG) ==0 ){} }
                }

                }
        //Button 2 only is pressed
        if((P1IN&(BUT1|BUT2)) == BUT1 )
                        {
                    P1OUT |= redLED;  //turn LED ON
                        for(i=0; i< timer_2; i++)
                            {
```

```
                                        //wait for flag, 2 cycles
                                        for(j=0; j<2; j++){ while( (TA0CTL&TAIFG) ==0 ){} }

                                        TA0CTL &= ~TAIFG; //clr flag
                                        if(i==(timer_2)-20) //2cycles
                                          {
                                            P9OUT ^= greenLED;
                                            for(j=0; j<4; j++){ while( (TA0CTL&TAIFG)==0){} }
                                            P9OUT ^= greenLED;
                                           }

                                        if((P1IN&BUT1)==0)
                                           {
                                            //add 10 seconds to timer
                                            timer_2+=20; //2 cycles per second
                                            P9OUT ^= greenLED;
                                            while((TA0CTL&TAIFG)==0){}
                                            P9OUT ^=greenLED;
                                           }

                                        }
                                    P1OUT &= ~redLED;
                                    P9OUT |= greenLED;
                                    for(i=0; i<5; i++){
                                        //wait for flag
                                      for( j=0; j<2; j++){ while( (TA0CTL&TAIFG) ==0 ){} }
                                    }

                                    }
                }

                return 0;
            }
```

Student Q&A:

1) Timer_A provides much more control and accuracy. With a standard delay loop, proprortionality of delays is the only way to create pseudo-instances of time. Whereas, the timer is configurable and runs off of a reliable crystal and is the best way generate delays that can be flexible and accurate.
2) Polling in this lab refers to waiting until a flag is raised and then taking a particular action. This is similar in manner to the ISR functionality of later labs, however GIE is is not enabled therefore the ISR is written directly into the polling loop. For this lab, TAIFG (roll back to zero event) is the interrupt flag that is utilized to signify the end of a period.

3) The polling technique is not suitable since the CPU is always on; Low power mode (through the use of GIE and interrupts) is the best way to conserve battery power.
4) The TAIFG only goes to 1 if the previous value of TAR was 65535-continous mode or TACCR0-1 in Up mode. A roll back event has to take place in order for TAIFG to be 1.
5) In similar fashion to question 2, this lab takes advantage of the TAIFG (interrupt flag) but GIE is not enabled. Therefore, there technically is not an ISR that the Hardware calls via vector table. Interrupts are not being fully utilized in this lab; rather TAIFG and its interrupt functionality is being used in an elementary fashion to achieve polling.
6) Up mode gives more control, since you can change the clock source, input divider, and the value that TAR counts up to. Whereas, in continuous mode the TAR max value is bound to 65536, and the only way to change the period is through the input divider, and the source select. Therefore, Up mode is more versatile since you can be set virtually any 16-bit value for TA0CCR0.

Conclusion: This lab proved to be a great resource for understanding a lot of Timer_A's functionality. It is clear that there is a lot of configuration that goes into using the Timer i.e. TASSEL, MC, ID, etc. All of these various choices provide sufficient means to obtain various timing periods. Continuous mode does not allow you to set the upper bound of TAR (locked at 65535). Whereas, Up mode you can set any (appropriate)value in TACCR0 for a "compare" event which allows for further customization of the timing. For this lab mainly ACLK and SMCLK were used; ACLK was reconfigured to use the 32,000 Hz crystal soldered to the Launchpad. A function was written to obtain this task. Both types of clock sources here were utilized during this lab. Using the higher freq. clock (SMCLK) means in some cases, the Frequency will be too high to achieve a realistic value for TACCR0 in up mode. From the lab, one of the best ways to achieve a precise one second timer is using the ACLK (32kHZ) and Up mode (TAACCR0=32,000-1). This will most likely be a common configuration for future experiments. NOTE: It is important, to always clear the Flag! If this is not done, then only one toggle will show up in the code.

Both modes of timing were showcased during the independent design projects. These projects were both built around one second timing, but achieved it in two different ways. I.e. the timer was obtained with higher frequency clock, with clock divider, and 2 cycles to obtain an approximate one second delay. These projects showcased enhanced functionality of the Timer_A module to create more common uses in embedded systems today (timer, stopwatch). The knowledge gained from these independent results will surely prove useful during future labs.