EEL 4742C: Embedded Systems

Christopher Badolato
9/30/2019
EEL4742-0011
LAB 3 Using the Timer

Introduction

In this lab we will configure the Timer_A module on the microcontroller. Using this timer, we can generate timing events. First, we will configure the timer to continuous mode, then in up mode. Using TA0CTL we can configure the timer to the mode we would like. Each bit of the 16-bit TA0CTL is configurable (Clock input divider) for this experiment we will only be changing the MCx and TAIFG fields. Our microcontroller contains a Timer_A register that increments by one each clock cycle. Once this TAR value reaches its maximum value (in our case 65535 for 16-bit), the value will "roll-back" to zero and the TAIFG flag will be raised indicating we have reset the TAR. In up mode it is possible to set the value that the flag will be raised using the TACCR0 register. So, if we know our clock cycle time of the MCU (or we can configure it), we can calculate our expected delay. In continuous and up mode, we must be sure to reset the flag bit (TAIFG) each iteration of the continuous for loop to ensure the loop will continue our timer delay.

Code to configure TA0CTL and reset ONLY the flag bit.

```
TA0CTL = (TASSEL_1|ID_0|MC_2|TACLR); //ID is divisor, MC is MODE
TA0CTL &= ~TAIFG; //Used to reset TAIFG to 0 each iteration.
```

Part 3.1: ...

In this part of the experiment the clock is configured to continuous mode. According to my calculation below, the red led will turn on for 2 seconds, then turn off for 2 seconds. When the TAR register reaches 65535 it will roll back to 0 and the TAIFG flag will be raised.
Write an analysis showing what delay you expect.
ACLK @ 32 KHz

65536 ~~cycles~~	sec	= 2 seconds
	37768 ~~cycles~~	

Measure the observed delay with your phone's stopwatch for at least 20 seconds. The timing should match closely since the crystal is accurate. Does it match?

It was accurate, 00:20:16, might be more human timing error

Try dividing the clock by 2, 4 or 8 using the ID field. What delays do these correspond to? Do they match what you observe?

Time Period (seconds)	ID	Divisor
2	0	/1
4	1	/2
8	2	/4
16	3	/8

```
//Christopher Badolato
//9/23/2019
//LAB 3.1
//EEL 4742L-0011

// Flashing the LED with Timer_A, continuous mode, via polling
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
#define greenLED BIT7 // Green LED at P9.7

void config_ACLK_to_32KHz_crystal();

void main(void) {

    volatile int i, j;
    WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer
    PM5CTL0 &= ~LOCKLPM5; // Enable the 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 the 32 KHz crystal (function call)
    config_ACLK_to_32KHz_crystal();
        // Configure Timer_A
        // Use ACLK, divide by 1, continuous mode, clear TAR
        // Ensure flag is cleared at the start
    TA0CTL = (TASSEL_1|ID_0|MC_2|TACLR);
    TA0CTL &= ~TAIFG;
        // Infinite loop
    for(;;) {
            // Empty while loop; waits here until TAIFG is raised
        while((TA0CTL & TAIFG) == 0) {}
            // Toggle the red LED
            // Clear the flag
        P1OUT ^= redLED;
        TA0CTL &= ~TAIFG;
    }
}


//*********************************
// Configures ACLK to 32 KHz crystal
void config_ACLK_to_32KHz_crystal(){
    // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz
    // Reroute pins to LFXIN/LFXOUT functionality
    PJSEL1 &= ~BIT4;
    PJSEL0 |= BIT4;
    // Wait until the oscillator fault flags remain 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;
}
```

Part 3.2: ...

In this part of the experiment we configure the Timer_A to up mode. Up mode is more configurable because we can control the value in which the TAIFG flag is raised at. Seeing as the value in TAR increments each clock cycle, we can solve for the desired time using the calculations from the previous part of the lab.

With ACLK @ 32 KHz

1 ~~second~~	65536 cycles	32768 cycles
	2 ~~seconds~~	

So, to stop the timer at the desired time (1s) we must set our TA0CCR0 register to (32768-1), this will increment the TAR register then roll back to zero once reached, hence the minus one.

What value of TA0CCR0 did you use?

(32768 – 1) = 32767

Were you able to achieve a precise 1-second timer period? Compare it to your phone's stopwatch for at least 20 seconds, it should match closely.

My measurement was 00:20.07, I'd say the error is once again human error and not computer. It seems to line up almost perfectly!

_ What value of TA0CCR0 achieves a delay of 0.1 seconds? Round up to the nearest integer and test this value.

TA0CCR0 = 3,277 should be the value to achieve the desired delay

_ What value of TA0CCR0 achieves a delay of 0.01 seconds? Round up to the nearest integer and test this value. What do you observe?

TA0CCR0 = 328 should be the value to achieve the desired delay

The light is blinking so fast that you can hardly notice a difference, but if you look close enough you can see it flickering.

```
//Christopher Badolato
//9/23/2019
//LAB 3.2
//EEL 4742L-0011

// Flashing the LED with Timer_A, UP MODE

#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
#define greenLED BIT7 // Green LED at P9.7
```

```
void config_ACLK_to_32KHz_crystal();

void main(void) {


    WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer
    PM5CTL0 &= ~LOCKLPM5; // Enable the 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 the 32 KHz crystal (function call)
    config_ACLK_to_32KHz_crystal();
        // Configure Timer_A
        // Use ACLK, divide by 1, continuous mode, clear TAR
        // Ensure flag is cleared at the start
    TA0CCR0 = (32768-1);
    TA0CTL = (TASSEL_1|ID_0|MC_1|TACLR);
    TA0CTL &= ~TAIFG;
        // Infinite loop
    for(;;) {
            // Empty while loop; waits here until TAIFG is raised
        while((TA0CTL & TAIFG) == 0) {}
            // Toggle the red LED
            // Clear the flag
        P1OUT ^= redLED;
        TA0CTL &= ~TAIFG;
    }
}



//*********************************
// Configures ACLK to 32 KHz crystal
void config_ACLK_to_32KHz_crystal(){
        // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz
        // Reroute pins to LFXIN/LFXOUT functionality
    PJSEL1 &= ~BIT4;
    PJSEL0 |= BIT4;
        // Wait until the oscillator fault flags remain 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;
}
```

Part 3.3.1

In this part of the experiment, using continuous mode, I changed the LED flashing pattern as well as include both LED.
Each iteration of the loop will, turn green on, toggle red twice, then shut both off for 2 seconds. Then, turn green off, toggle red twice again, shut both off for 2 seconds.

65536 ~~cycles~~	sec	= 2 seconds
	37768 ~~cycles~~	

```
//Christopher Badolato
//9/23/2019
//LAB 3.1
//EEL 4742L-0011

// Flashing the LED with Timer_A, continuous mode, via polling
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
#define greenLED BIT7 // Green LED at P9.7

void config_ACLK_to_32KHz_crystal();

void main(void) {

    volatile int i, j;
    WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer
    PM5CTL0 &= ~LOCKLPM5; // Enable the 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 the 32 KHz crystal (function call)
    config_ACLK_to_32KHz_crystal();
        // Configure Timer_A
        // Use ACLK, divide by 1, continuous mode, clear TAR
        // Ensure flag is cleared at the start
    TA0CTL = (TASSEL_1|ID_0|MC_2|TACLR);
    TA0CTL &= ~TAIFG;
        // Infinite loop
    for(;;) {
            //toggle green LED
        P9OUT ^= greenLED;
            // Empty while loop; waits here until TAIFG is raised
        while((TA0CTL & TAIFG) == 0) {}
            // Toggle the red LED
            // Clear the flag
        for(i = 0; i < 4; i++){
            for(j = 0; j <10000; j++){}
            P1OUT ^= redLED;
        }
        TA0CTL &= ~TAIFG;
    }
}
```

```
//*********************************
// Configures ACLK to 32 KHz crystal
void config_ACLK_to_32KHz_crystal(){
    // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz
    // Reroute pins to LFXIN/LFXOUT functionality
    PJSEL1 &= ~BIT4;
    PJSEL0 |= BIT4;
    // Wait until the oscillator fault flags remain 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;
}
```

Part 3.3.2

In this part of the experiment, using up-mode, I changed the LED flashing pattern as well as include both LED
Each iteration of the main for loop will run in up mode for 1 second. First the Red LED will toggle, then the green LED will flash twice. The red LED toggles every 1 second, and the green led will flash twice each second.

1 ~~second~~	65536 cycles	32768 cycles
	2 ~~seconds~~	

```
//Christopher Badolato
//9/23/2019
//LAB 3.2
//EEL 4742L-0011

// Flashing the LED with Timer_A, UP MODE

#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
#define greenLED BIT7 // Green LED at P9.7

void config_ACLK_to_32KHz_crystal();

void main(void) {

    volatile int i, j;
    WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer
    PM5CTL0 &= ~LOCKLPM5; // Enable the 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 the 32 KHz crystal (function call)
    config_ACLK_to_32KHz_crystal();
        // Configure Timer_A
        // Use ACLK, divide by 1, continuous mode, clear TAR
        // Ensure flag is cleared at the start
        //
    TA0CCR0 = (32768-1);
    TA0CTL = (TASSEL_1|ID_0|MC_1|TACLR);
    TA0CTL &= ~TAIFG;
        // Infinite loop
    for(;;) {
            // Empty while loop; waits here until TAIFG is raised
        P1OUT ^= redLED;
        while((TA0CTL & TAIFG) == 0) {}
            // Toggle the red LED
            // Clear the flag
        for(i=0; i<4; i++){
            for(j=0;j<10000;j++){}
            P9OUT ^= greenLED;
        }
        TA0CTL &= ~TAIFG;
    }
}



//*********************************
// Configures ACLK to 32 KHz crystal
void config_ACLK_to_32KHz_crystal(){
        // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz
        // Reroute pins to LFXIN/LFXOUT functionality
    PJSEL1 &= ~BIT4;
    PJSEL0 |= BIT4;
        // Wait until the oscillator fault flags remain 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;
}
```

Student Q&A

1. So far, we have seen two ways of generating delays: using a delay loop and using Timer A. Which approach provides more control and accuracy over the delays? Explain.

In up mode, TACCR0 can be configured to any value we like which provides us with more control and which can be simply calculated. Continuous mode is limited by the size of the TAR register and the clock cycles per second we are running at.

2. Explain the polling technique and how it's used in this lab.

Polling technique is based on reading a flag continuously, when read "set" an action occurs and in our case the flag is reset.

3. Is the polling technique a suitable choice when we care about saving battery power? Explain.

No, it wastes a lot of power by constantly polling checking the flag each TAR iteration. We will later learn to set up interrupts to save power.

4. If we write 0 to TAR using a line code, does the TAIFG go to 1?

Yes, when TAR is 0, TAIFG becomes 1.

5. In this lab, we used TAIFG to time the duration. TAIFG is known as the Timer A Interrupt Flag, which is an interrupt flag. Were we using interrupts in this lab? Explain.

No, we are using polling technique to check the TAIFG flag. An interrupt is triggered when a time period elapses the timer will raise an interrupt which invokes the hardware to run a special piece of code.

6. From what we have seen in this lab, which mode gives us more control over the timing duration: the up mode or the continuous mode?

Up mode gives more control over the timing and the duration of the delay.

Conclusion

In this lab experiment we configured the Timer_A module on the microcontroller using the polling technique. We do this by checking when the TAR register has reset. First, we configured continuous mode which continues incrementing the TAR register until the maximum value is reach, then the flag is raised and the register rolls back to zero. This is not very customizable. Secondly, we configured the timer to up-mode which allows the user to set the number of cycles the TAR register will count until the flag is raised. Finally, I created my own version of the continuous and up mode experiments.