EEL 4742C: Embedded Systems

Christopher Badolato
10/21/2019
EEL4742-0011
Lab 6 Advanced Timer Features

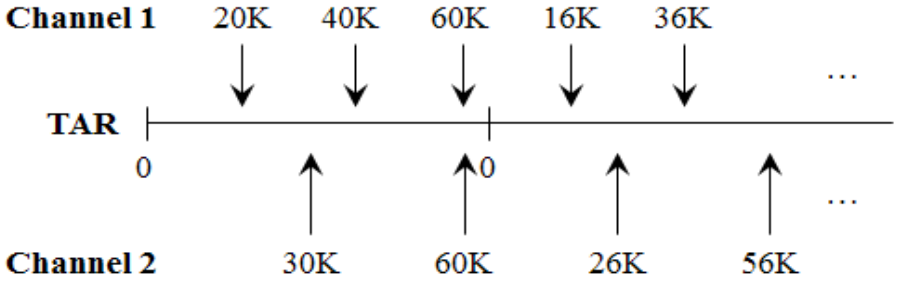
Introduction

In this lab we will be using the timer to control multiple channels so we can generate independent periodic interrupts. In this lab using these channels we will first generate patterns by changing the time of the LED. Then, use output patterns to generate a Pulse Width Modulation signal (PWM) which can be used to control the brightness level of the LED. Finally, we will cycle through different brightness levels.

Part 6.1: ...

In this part of the Lab we will be configuring two channels. Doing so will allow us to toggle our LEDs at different rates that will stay consistent with the Timer interrupts. Independent intervals with multiple channels are created by first setting the clock to continuous mode. TAR counts from 0 to 32768 (we are configuring the clock to 32Khz), and the individual interrupts functions will be triggered according to the values of TA0CCR0 and TA0CCR1 that we set. That is, when TAR = TA0CCR0 or TA0CCR1 the interrupt will trigger. Within each timer interrupt ISR we must schedule the next interrupt by incrementing the values at TA0CCR1 by the number of cycles we want to see before the next interrupt. Below in figure 6.1 we can see how these TA0CC values change with the value of TAR but for 64KHz.

Figure 6.1: Using two channels of Timer A (taken from lab manual)



These values will not exceed the maximum value of TAR. If they do they will be set accordingly:
60k + 20k = 80k then
80k – 64k = 16k
This will set the TACCR values at the correct spot for the next interrupt to occur.

Show how the number of cycles for each channel is derived.

Channel 0		
0.1 second	32768 cycles	= 3277cycles
	Second	

Channel 1		
0.5 second	32768 cycles	= 16385 cycles
	Second	

```
//Christopher Badolato
//10/21/2019
//LAB 6.1
//EEL 4742L-0011

// Using Timer_A with 2 channels
// Using ACLK @ 32 KHz (undivided)
// Channel 0 toggles the red LED every 0.1 seconds
// Channel 1 toggles the green LED every 0.5 seconds

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

void config_ACLK_to_32KHz_crystal();

void main(void) {

    WDTCTL = WDTPW | WDTHOLD; // Stop WDT
    PM5CTL0 &= ~LOCKLPM5; // Enable GPIO pins
    P1DIR |= redLED;

    P9DIR |= greenLED;
    P1OUT &= ~redLED;
    P9OUT &= ~greenLED;

    config_ACLK_to_32KHz_crystal();
        // Configure Channel 0
    TA0CCR0 = 3277-1; // @ 32 KHz --> 0.1 seconds
    TA0CCTL0 |= CCIE;
    TA0CCTL0 &= ~CCIFG;
        // Configure Channel 1 (write 3 lines similar to above)
    TA0CCR1 = 16385-1; // @ 32 KHz --> 0.5 seconds
    TA0CCTL1 |= CCIE;
    TA0CCTL1 &= ~CCIFG;
        // Configure timer (ACLK) (divide by 1) (continuous mode)
    TA0CTL = (TASSEL_1|ID_0|MC_2|TACLR|TAIE);
        // Engage a low-power mode
    _low_power_mode_3();
```

```
    return;

}
    // ISR of Channel 0 (A0 vector)
#pragma vector = TIMER0_A0_VECTOR
__interrupt void T0A0_ISR() {
    P1OUT ^= redLED; // Toggle the red LED
    TA0CCR0 += 3277; // Schedule the next interrupt
        // Hardware clears Channel 0 flag (CCIFG in TA0CCTL0)
}
    // ISR of Channel 1 (A1 vector) ... fill the vector name below
#pragma vector = TIMER0_A1_VECTOR

__interrupt void T0A1_ISR() {
    if((TA0CCTL1 & CCIFG) != 0){
        P9OUT ^= greenLED; // Toggle the Green LED
        TA0CCR1 += 16385; // Schedule the next interrupt
        TA0CCTL1 &= ~CCIFG; // Clear Channel 1 interrupt flag
    }
}


//*********************************
// 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 6.2: ...

This part of the lab is similar to part 6.1 though we will be using 3 channels. Two (Channels 0 and 1) will be configured for toggling of the LEDs and the other will time 4 second intervals between LED flashing.
We will set a status bit, that will toggle when our 4 second (channel 2) timer interrupt occurs letting us know when we can either, turn off the LED toggling for 4 seconds and increment the TA0CCR value or, the LEDs will be toggling according to their own timer interrupts each time the channel 2 interrupt occurs.
If the status bit is 1 the LED's will be toggled
If the status bit is 0, the LEDs will turn off and remain off.
Below we can see the cycles used for each channel.

Show how the number of cycles for each channel is derived.

Channel 0		
4 seconds	8192 cycles	= 32768 cycles
	Second	

Channel 1		
0.5 second	8192 cycles	= 4096 cycles
	Second	

Channel 2		
0.1 second	8192 cycles	= 819 cycles
	Second	

```
//Christopher Badolato
//10/21/2019
//LAB 6.2
//EEL 4742L-0011

// Using Timer_A with 2 channels
// Using ACLK @ 32 KHz (undivided)
// Channel 0 toggles the red LED every 0.1 seconds
// Channel 1 toggles the green LED every 0.5 seconds

#include <msp430fr6989.h>
#define redLED BIT0 // Red at P1.0
#define greenLED BIT7 // Green at P9.7
int status = 1;

void config_ACLK_to_32KHz_crystal();

void main(void) {

    WDTCTL = WDTPW | WDTHOLD; // Stop WDT
    PM5CTL0 &= ~LOCKLPM5; // Enable GPIO pins
    P1DIR |= redLED;
    P9DIR |= greenLED;
    P1OUT &= ~redLED;
    P9OUT &= ~greenLED;

    config_ACLK_to_32KHz_crystal();
        // Configure Channel 0
    TA0CCR0 = (819-1); // @ 8 KHz --> 0.1 seconds
    TA0CCTL0 |= CCIE;
    TA0CCTL0 &= ~CCIFG;

        // Configure Channel 1 (write 3 lines similar to above)
    TA0CCR1 = (4097-1); // @ 8 KHz --> 0.5 seconds
```

```
    TA0CCTL1 |= CCIE;
    TA0CCTL1 &= ~CCIFG;

    // Configure Channel 2
    TA0CCR2 = (32768-1); // @ 8 KHz --> 4 seconds
    TA0CCTL2 |= CCIE;
    TA0CCTL2 &= ~CCIFG;

        // Configure timer (ACLK) (divide by 4) (continuous mode)
    TA0CTL = (TASSEL_1|ID_2|MC_2|TACLR|TAIE);
        // Engage a low-power mode
    _low_power_mode_3();
    return;

}
    // ISR of Channel 0 (A0 vector)
#pragma vector = TIMER0_A0_VECTOR
__interrupt void T0A0_ISR() {
    // Toggle the red LED
     // Schedule the next interrupt

       // Hardware clears Channel 0 flag (CCIFG in TA0CCTL0)
    if(status == 1){
        P1OUT ^= redLED;
    }
    TA0CCR0 += 819;
}


// ISR of Channels 1 and 2 (A1 vector)
#pragma vector = TIMER0_A1_VECTOR
__interrupt void T0A1_ISR() {
    // Detect Channel 1 interrupt
    if((TA0CCTL1 & CCIFG) != 0) {
        if(status == 1){
            P9OUT ^= greenLED;
        }
        TA0CCR1 += 4096;
        TA0CCTL1 &= ~CCIFG; // Clear Channel 1 interrupt flag
    }
    // Detect Channel 2 interrupt 4 second interrupt
    if((TA0CCTL2 & CCIFG) != 0) {
            //stop channels 1 and 2
        status ^= 1;
        P1OUT &= ~redLED;
        P9OUT &= ~greenLED;

        TA0CCR2 += 32768; // Schedule the next interrupt
        TA0CCTL2 &= ~CCIFG; // Clear Channel 2 interrupt flag
    }

}

//*********************************
// Configures ACLK to 32 KHz crystal
```

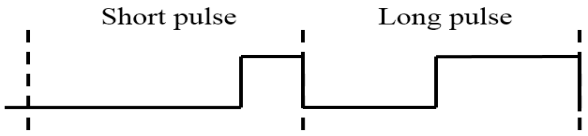
```
void config_ACLK_to_32KHz_crystal(){
        // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz
        // Reroute pins to LFXIN/LFXOUT functionality
    PJSEL1 &= ~BIT4;
    PJSEL0 |= BIT4;
        // Wait until the oscillator fault flags remain cleared
    CSCTL0 = CSKEY; // Unlock CS registers
    do {
        CSCTL5 &= ~LFXTOFFG; // Local fault flag
        SFRIFG1 &= ~OFIFG; // Global fault flag
    } while((CSCTL5 & LFXTOFFG) != 0);
        CSCTL0_H = 0; // Lock CS registers
    return;
}
```

Part 6.3.

In this part of the experiment we will be using the Timer_A module to generate a PWM Pulse-Width modulation to change the brightness of the LED.

Using figure 6.3 we can see an example of a PSW. The ratio of the high duration to the period is called the duty cycle. Higher duty cycle will result in higher brightness, if we make the period too long, the user cannot notice the blinking of the LED. In our experiment we use 1000Hz frequency or (0.001s).

Figure 6.3: Pulse Width Modulation (PWM) Signal



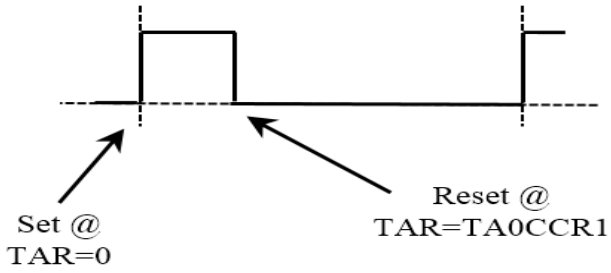
- Short pulse, less bright
- Long pulse, more bright

To set up the PSW to the LED we must first divert our Pin to TA0.1. Before we were controlling the LED with P1OUT. Using the lines below we can set the PINOUT to divert the pin.

```
    P1DIR |= PWM_PIN; // P1DIR bit = 1
    P1SEL1 &= ~PWM_PIN; // P1SEL1 bit = 0
    P1SEL0 |= PWM_PIN; // P1SEL0 bit = 1
```

We must then set our output pattern. The output mode has two actions, in this case, set/reset. One action occurs when TAR = TACCR1 (reset), while the other occurs when TAR rolls back to 0 (TAR = 0). So, we can see that increasing the value of TACCR0 will increase the brightness.

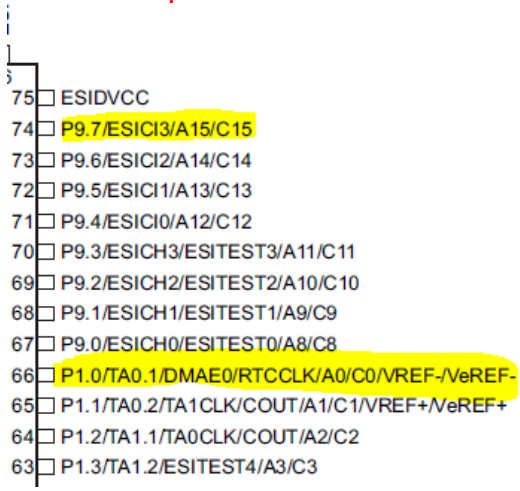
Figure 6.4: Reset/Set Output Pattern (From lab manual)



We can set this modulation with the line of code shown below, it will set to Set/Reset outmode.

```
TA0CCTL1 |= OUTMOD_7;
```

Include the snapshot of the pinout and mention, for each of the LED pins (P1.0 and P9.7), whether it's possible to connect it to a timer driven PWM signal.



We can see above that the P1.0 can be configured to TA0.1 but P9.7 has no timer associated with it.

Vary the value of TA0CCR1 between 1 and 32 and ensure this results in various brightness levels.
Do larger values of TA0CCR1 results in higher or lower brightness? Explain.

If you increase the value of TA0CCR1 the brightness decreases, the duty circle gets shorter the larger the value of TA0CCR1.

```
//Christopher Badolato
//10/21/2019
//LAB 6.3
//EEL 4742L-0011

// Generating a PWM on P1.0 (red LED)
// P1.0 coincides with TA0.1 (Timer0_A Channel 1)
```

```
// Divert P1.0 pin to TA0.1 ---> P1DIR=1, P1SEL1=0, P1SEL0=1
// PWM frequency: 1000 Hz -> 0.001 seconds
#include <msp430fr6989.h>
#define PWM_PIN BIT0

void config_ACLK_to_32KHz_crystal();

void main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop WDT
    PM5CTL0 &= ~LOCKLPM5;

    // Divert pin to TA0.1 functionality (complete last two lines)
    P1DIR |= PWM_PIN; // P1DIR bit = 1
    P1SEL1 &= ~PWM_PIN; // P1SEL1 bit = 0
    P1SEL0 |= PWM_PIN; // P1SEL0 bit = 1

    // Configure ACLK to the 32 KHz crystal (call function)
    config_ACLK_to_32KHz_crystal();

    // Starting the timer in the up mode; period = 0.001 seconds
    // (ACLK @ 32 KHz) (Divide by 1) (Up mode)
    TA0CCR0 = (33-1); // @ 32 KHz --> 0.001 seconds (1000 Hz)
    TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;

    // Configuring Channel 1 for PWM
    TA0CCTL1 |= OUTMOD_7; // Output pattern: Reset/set
    TA0CCR1 = 1; // Modify this value between 0 and
                 // 32 to adjust the brightness level

    for(;;) {}

}

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 6.4.

This part of the lab is similar to part 6.3. We will be using a timer interrupt to increment the value of TA0CCR1 by 5 to slowly increase the brightness of the LED each time the interrupt occurs. Each second (32768 cycles with 3KHz clock) the timer interrupt will occur increasing the brightness, if TA0CCR1 exceeds 32 (where we cannot notice a brightness change) we will reset TA0CCR1 to 0.

Explain how the two timer modules are interacting with each other.
The LED is toggling extremely fast according to the value we set for TA0CCR0, when it is set the LED will be lit and these interrupts occur every (0.001 seconds). The other timer will just be used to change the brightness of the LED each second as well as toggle the greenLED.

```
//Christopher Badolato
//10/21/2019
//LAB 6.4
//EEL 4742L-0011

// Generating a PWM on P1.0 (red LED)
// P1.0 coincides with TA0.1 (Timer0_A Channel 1)
// Divert P1.0 pin to TA0.1 ---> P1DIR=1, P1SEL1=0, P1SEL0=1
// PWM frequency: 1000 Hz -> 0.001 seconds
#include <msp430fr6989.h>
#define PWM_PIN BIT0
#define greenLED BIT7

void config_ACLK_to_32KHz_crystal();

void main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop WDT
    PM5CTL0 &= ~LOCKLPM5;

        // Divert pin to TA0.1 functionality (complete last two lines)
    P1DIR |= PWM_PIN; // P1DIR bit = 1
    P1SEL1 &= ~PWM_PIN; // P1SEL1 bit = 0
    P1SEL0 |= PWM_PIN; // P1SEL0 bit = 1

        // Configure greenLED
    P9DIR |= greenLED;
    P9OUT &= ~greenLED;

        // Configure ACLK to the 32 KHz crystal (call function)
    config_ACLK_to_32KHz_crystal();

        // Starting the timer in the up mode; period = 0.001 seconds
        // (ACLK @ 32 KHz) (Divide by 1) (Up mode)
    TA0CCR0 = (33-1); // @ 32 KHz --> 0.001 seconds (1000 Hz)
    TA0CTL = TASSEL_1 | ID_1 | MC_1 | TACLR;

        // Configuring Channel 1 for PWM
    TA0CCTL1 |= OUTMOD_7; // Output pattern: Reset/set
    TA0CCR1 = 0; // Modify this value between 0 and
                 // 32 to adjust the brightness level

        // Configuring Timer1_A Channel 0
```

```
    TA1CCR0 = 32768;
    TA1CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
    TA1CCTL0 |= CCIE;
    TA1CCTL0 &= ~CCIFG;

        // Engage a low-power mode
    _low_power_mode_3();

}

    // ISR of Timer1_A Channel 0
#pragma vector = TIMER1_A0_VECTOR
__interrupt void T1A0_ISR() {
    P9OUT ^= greenLED; // Toggle the greenLED

    if(TA0CCR1 >= 32)
    {
     TA0CCR1 = 0;
    }
    else
     TA0CCR1 += 5;
}

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. In the code with three channels, why did we divide ACLK so it becomes 8 KHz?

We want a delay of 4 seconds between each toggle of the LEDs. The only way to achieve this is to reduce the clock cycles. If we used continuous mode 4 seconds would be (32K * 4) because 65536 is our cap cycle value (which can only equal 2 seconds at 32Khz), we must decrease our clock cycle to 8Khz

2. In the first part, we configured two periodic interrupts using two channels of the timer. Is

this approach scalable? For example, using a Timer A module with five channels, can we configure five periodic interrupts? Explain and mention in what mode the timer would run.

Yes, this is scalable, you can configure multiple channels to follow the same timer. This timer will run in continuous mode to follow the rollback to 0 as well as keep the math consistent.

3. As an example, Channel 1's interrupt occurs every 40K cycles. The first interrupt is scheduled for when TAR=40K cycles. Explain how the next interrupt is scheduled?

The next interrupt is scheduled by incrementing TA0CCR1 by 40k. If we are using the 64Khz clock the math is as follow 40k + 40k = 80k 80k-64k = 14k 14k+40k = 54k and so on…

Conclusion

In this Lab experiment we first learned to configure multiple timer interrupt channels. We then use three channels to toggle the LEDS for a specific amount of time and shut them off for the same amount of time. Next, we learn to change the brightness of the LEDs using the set/reset mode to create a duty cycle of LED toggling. Finally we cycle through different brightness levels on the red LED.