EEL 4742C: Embedded Systems

Name: Hamzah Ullah

Lab 6: Advanced Timer Features

Introduction:

This lab introduces us to multiple channels in the timer, using multiple channels and the PWM or pulse width modulator. We will use multiple timers to replicate behaviors with the LED while keeping track of the TACCR and using intervals to time them. When the flag raises, the difference in the value will not affect the rate. A pulse width modulator allows us to create 'duty cycles' or length of time the clock is triggered up vs low. For instance, a 25% duty cycle is down 3 times as long as it is up(25% up/75% down.)

Part 1: Using the timer with Two Channels

This part of the lab introduces us to using multiple channels in up mode, while keeping track of the interval. Keeping track of timer channels, we also have to keep note of which channel automatically clears the flag and which channels require us to clear the flag ourselves. The intervals are not affected by the carry over zero because if take the difference of the max interval of a 16 bit number (64k) and subtract it from the cycle position + interval if the sum is greater than 64k, we can find the next position of the interval. For example, the book gives us a 20k interval and a 30k interval. At some point both will exceed 64k. What we do is we take the sum of 20k + 60k = 80k which is greater than the 64k and take the difference between those numbers. 80k-64k = 16k which will the position of the next interval when the action will occur, and it will not affect the output. Our labs use 3.2khz and 16khz which conveniently look around nicely, so we will not face that issue.

We are to use 2 channels to create a .5second interval and a .1 second interval. Using a 32khz clock, we can obtain a .5 second interval by setting TACCR to 16384-1 which will trigger every half second, and the second timers TACCR to 3277-1 to trigger every .1 seconds. The analysis behind it is if the timer triggers every second at 32khz, we simply set the flag to raise in half that time and 1/10 that time for .5 and .1 seconds respectively. Below is the code that uses a timer on 2 channels. The durations do appear to be correct

Using 2 channels to blink each light on its own channels at different intervals

redLED @ .1 seconds, and greenLED at .5 Seconds

```c
//Using Timer_A with 2 channels
// Using ACLK @ 32KHz (undivided)
//Channel 0 toggles the red LED every .1 seconds
//Channel 1 toggles the green LED ever .5 Seconds
#include <msp430fr6989.h>
#define redLED BIT0    //red at p.1.0
#define greenLED BIT7  //green at p9.7

/**
 * main.c
 */
void main(void)
{
	WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer
	PM5CTL0 &= ~LOCKLPM5;       //Enable GPIO Pins

	P1DIR |= redLED;
	P9DIR |= greenLED;
	P1OUT &= ~redLED;
	P9OUT &= ~greenLED;

	//32KhZ crystal ACLK
	config_ACLK_to_32KHz_crystal();

	//configure channel 0
```

```
        TA0CCR0 = (3277-1);       //@32KHz --> .1seconds
        TA0CCTL0 |= CCIE;
        TA0CCTL0 &= ~CCIFG;

        //configure channel 1
            TA0CCR1 = (16384-1);       //@32KHz --> .5seconds
            TA0CCTL1 |= CCIE;
            TA0CCTL1 &= ~CCIFG;

     //Configure Timer(ACLK) (Divide by 1) (Continuous mode)
         TA0CTL = TASSEL_1 | ID_0 | MC_2 | TACLR;

         //Engage 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
    //HW clears channel 0 flag(CCIFG in TA0CCTL0)
}
#pragma vector = TIMER0_A1_VECTOR
__interrupt void T0A1_ISR()
{
    P9OUT ^= greenLED;  //toggle the greenLED
    TA0CCR1 += 16324;   //Schedule the next interrupt
    TA0CCTL1 &= ~CCIFG;  //Clear Channel 1 interrupt flag
}
void config_ACLK_to_32KHz_crystal()
    {
        //By default, ACLK runs on LFMODCLK at 5MHz/128 = 39kHz

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

        //Wait until the oscillator fault flags remain cleared
        CSCTL0 = CSKEY;
        do
        {
            CSCTL5 &= ~LFXTOFFG;    //local fault flag
            SFRIFG1 &= ~OFIFG;      //Global fault flag
        }
        while((CSCTL5 & LFXTOFFG) != 0);

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

Part 2: Using three channels

For this part of the lab, we go further by creating a larger interrupt that is kept track of by another clock channel and trigger events based off of that. We are to use 3 channels to create a 4 second interrupt between the lights flashing. We know the inputs for the LEDs from part 1, so that code carries over. To do this we need to divide our new channel we will use to create the delay by the time delay we want. Since we want a 4 second delay, we will divide by 4, ID_2. This means that a 1 second delay is now 8khz, so we need the interrupt to toggle a delay at 4 times that much or 32khz equivalent (32768-1 for TACCR). The second part of this lab is to determine when the LED is flashing or not. To do this we will use a status variable called flashing, to designate when it is flashing and toggle the value of it when it is or isn't flashing. Than when we determine if its flashing, we will start disabling the interrupts for the flashing lights and set the status to not flashing. If it is not flashing, we will reenable the interrupt for flashing and continue at the prior intervals. Since we are using 8khz equivalent, the previous codes TACCR for the redLED and greenLED are divided by 4. (820 and 4096 respectively). This code will toggle every 4 seconds between flashing 0 and flashing 1. Below is the following code. The intervals are derived similar to above, but not using an 8khz clock. So for .5 seconds we will use 8khz/2 which is 4096, and .1 seconds is 8khz/10 which is 819, but always rounds back up to 820.

Using 3 channels to create a 4 second interrupt

```
//Using Timer_A with 2 channels
// Using ACLK @ 32KHz (undivided)
//Channel 0 toggles the red LED every .1 seconds
//Channel 1 toggles the green LED ever .5 Seconds
#include <msp430fr6989.h>
#define redLED BIT0     //red at p.1.0
#define greenLED BIT7   //green at p9.7

/**
 * main.c
 */
void main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;       //Enable GPIO Pins

    P1DIR |= redLED;
    P9DIR |= greenLED;
    P1OUT &= ~redLED;
    P9OUT &= ~greenLED;

    //32KhZ crystal ACLK
    config_ACLK_to_32KHz_crystal();

    //configure channel 0
    TA0CCR0 = (820-1);        //@32KHz --> .1seconds
    TA0CCTL0 |= CCIE;
    TA0CCTL0 &= ~CCIFG;

    //configure channel 1
        TA0CCR1 = (4096-1);       //@32KHz --> .5seconds
        TA0CCTL1 |= CCIE;
        TA0CCTL1 &= ~CCIFG;

    //configure channel 2
        TA0CCR2 = (32768-1);
        TA0CCTL2 |= CCIE;
        TA0CCTL1 &= ~CCIFG;

    //Configure Timer(ACLK) (Divide by 4) (Continuous mode)
     TA0CTL = TASSEL_1 | ID_2 | MC_2 | TACLR;

     //Engage 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 += 820;    //schedule the next interrupt
    //HW clears channel 0 flag(CCIFG in TA0CCTL0)
}
#pragma vector = TIMER0_A1_VECTOR
__interrupt void T0A1_ISR()
{
    if((TA0CCTL1 & CCIFG)!=0)
    {
    P9OUT ^= greenLED;  //toggle the greenLED
    TA0CCR1 += 4096;   //Schedule the next interrupt
    TA0CCTL1 &= ~CCIFG;  //Clear Channel 1 interrupt flag
    }
    static int flashing = 1;
    if((TA0CCTL2 & CCIFG)!=0)
    {
        if(flashing == 1)
        {
            TA0CCTL1 &= ~CCIE;
            TA0CCTL0 &= ~CCIE;
            P1OUT &= ~redLED;
            P9OUT &= ~greenLED;
            flashing = 0;
        } else if (flashing == 0)
        {
            TA0CCR1 = 4096;
            TA0CCTL1 &= ~CCIFG;
            TA0CCTL1 |= CCIE;
            TA0CCR0 = 820;
            TA0CCTL0 &= ~CCIFG;
            TA0CCTL0 |= CCIE;
            flashing = 1;
        }
        TA0CCR2 += 32768;
        TA0CCTL2 &= ~CCIFG;
    }
```

```
}
void config_ACLK_to_32KHz_crystal()
    {
        //By default, ACLK runs on LFMODCLK at 5MHz/128 = 39kHz

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

        //Wait until the oscillator fault flags remain cleared
        CSCTL0 = CSKEY;
        do
        {
            CSCTL5 &= ~LFXTOFFG;    //Local fault flag
            SFRIFG1 &= ~OFIFG;      //Global fault flag
        }
        while((CSCTL5 & LFXTOFFG) != 0);

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

Part 3: Generating a PWM signal with Timer_A

For this part of the lab, we will create a power width modulation by directing the pins with he LED to a timer. When it is directed with the timer we can use the timer to manipulate the light intensity. Looking at the Data sheet, we can find which pins are designated to which function. When we do find it, we can redirect those pins. We look at the datasheet for our board an see if its even possible by checking the Timerx.y which is the timer number and channel. Below is the pins for the red and greenLED.

66☐ P1.0/TA0.1/DMAE0/RTCCLK/A0/C0/VREF-/VeREF-
74☐ P9.7/ESICI3/A15/C15

We can see that the green LED cannot be mapped with the PWM because it has no timer channel. So we map the channel select and direction by looking at the table on the data sheet tables.

Questions:

Changing TA0CCR1 does not result in higher or lower brightness, a higher number simply makes it blink based on the rate and lower just makes the light extremely dim. This is because of the rate at which the flag is cleared. If the flag is cleared right away, the light does not have enough time to shine as opposed to slower clearing of the flag results in a brighter light to a limit. Since we are using 1000hz and 33 cycles for TACCR. This will give us many levels to cycle through later in the lab.

Generating a PWM with a .001second period in up mode. Can change the light intensity.

```
//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, P1SEL=0, P1SEL0=1
//PWM frequency 1000hz --> .001 seconds
#include <msp430fr6989.h>
#define PWM_PIN BIT0

/**
 * main.c
 */
void main(void)
{
      WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer
```

```
        PM5CTL0 &= ~LOCKLPM5;

        //Divert Pin to TA0.1 Functionality (complete the last 2 lines)
        P1DIR |= PWM_PIN;       //P1DIR bit = 1
        P1SEL1 &= ~BIT7;                  //P1SEL1 bit = 0
        P1SEL0 |= PWM_PIN;                //P1SEL0 bit = 1

        //32 KHZ clock
        config_ACLK_to_32KHz_crystal();

        //Starting the timer in up mode; period = .001 seconds
        //ACLK @ 32khz, DIV by 1, up mode
        TA0CCR0 = (33-1);
        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
                                //32 to adjust brightness level

        for(;;){}
}
void config_ACLK_to_32KHz_crystal()
    {
        //By default, ACLK runs on LFMODCLK at 5MHz/128 = 39kHz

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

        //Wait until the oscillator fault flags remain cleared
        CSCTL0 = CSKEY;
        do
        {
            CSCTL5 &= ~LFXTOFFG;    //Local fault flag
            SFRIFG1 &= ~OFIFG;      //Global fault flag
        }
        while((CSCTL5 & LFXTOFFG) != 0);

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

Part 4: Cycling through Brightness Levels

This part of the lab simply adds an interrupt to change the TA0CCR1 levels band cycle through brightness levels from 0 meaning off to 30 being the brightest before we roll back around. To do this I set a interrupt to increment by 5 from an initial TACCR of 0. When the TACCR was over thirty, it would manually set the value to 0 and start from there. So in my code, there is no rolling back to zero, but rather staying within a certain range between 0-30 and reset to zero when we are over 30. Below is the code. The two timers are not really interacting with each other but rather performing their independent functions and changing their output on the interrupt.

This increments the LED between 0 5 10 15 20 25 30 brightness and resets back to zero. Does so every time the interrupt is triggered.

```
//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, P1SEL=0, P1SEL0=1
//PWM frequency 1000hz --> .001 seconds
#include <msp430fr6989.h>
#define PWM_PIN BIT0
#define greenLED BIT7

/**
 * main.c
 */
void main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;

    //Divert Pin to TA0.1 Functionality (complete the last 2 lines)
    P1DIR |= PWM_PIN;       //P1DIR bit = 1
    P1SEL1 &= ~BIT7;                  //P1SEL1 bit = 0
    P1SEL0 |= PWM_PIN;                //P1SEL0 bit = 1

    P9DIR |= greenLED;
    P9OUT &= ~greenLED;
    //32 KHZ clock
    config_ACLK_to_32KHz_crystal();

    //Starting the timer in up mode; period = .001 seconds
    //ACLK @ 32khz, DIV by 1, up mode
    TA0CCR0 = (33-1);
    TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
    TA1CTL = TASSEL_1 | ID_1 | MC_1 | TACLR;

    TA1CCR0 = (16384-1);
    TA1CCTL0 |= CCIE;
    TA0CCTL0 &= ~CCIFG;
    //Configuring Channel 1 for PWM
    TA0CCTL1 |= OUTMOD_7;   //Output pattern: Reset/Set
    TA0CCR1 = 0;            //Modify this value between 0
                            //32 to adjust brightness level

    _low_power_mode_3();
}
#pragma vector = TIMER1_A0_VECTOR //Link the ISR to the vector
    __interrupt void T0A1_ISR()
    {

        //Toggle both LEDs
        P9OUT ^= greenLED;
        TA0CCR1 += 5;

        if(TA0CCR1 > 30)
            TA0CCR1 = 0;
        //clear the TAIFG flag
        TA1CTL &= ~TAIFG;
    }

void config_ACLK_to_32KHz_crystal()
    {
        //By default, ACLK runs on LFMODCLK at 5MHz/128 = 39kHz

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

        //Wait until the oscillator fault flags remain cleared
        CSCTL0 = CSKEY;
        do
        {
            CSCTL5 &= ~LFXTOFFG;    //local fault flag
            SFRIFG1 &= ~OFIFG;      //Global fault flag
        }
        while((CSCTL5 & LFXTOFFG) != 0);

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

QUESTIONS:

1. **In the code with three channels, why did we divide ACLK so it becomes 8KHz?**
   We divided it so that we could create a 4 second delay between interrupts. We understand that 8khz is 1s, and the max clockrate is 32khz or a TACCR of 32768. So we used 8Khz so that we were able to achieve the delay by increasing the time between the interrupts allowing us to create a 4 second delay between blink on and blink off.

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.**
   It is scalable and we can use 5 channels. To do so we would need to be in up mode to be able to manipulate the TACCR of each interrupt we wanted to occur. As for the channels, the interrupts would be based on parameters we set for them, when they trigger and when they detect other values triggering. We can use a status variable to detect the action by checking the flag, or disabling other interrupts for them to do the action we wanted. Up mode, allows us to change the TACCR0 which makes interrupts scalable with the timer channels.

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? (hint: explain the overflow mechanism and why it results in a correct value)**
   The next interrupt is scheduled by taking the difference. If we are using a 64khz clock, we see that the current position of tar is 40k and next position is += 40k or 80k. We would now take the difference between these two numbers; 80k-64k = 16k. This keeps the timer in sync and would not affect the rate at which the actions occur.