**11/23/2020**

Joshua Frazer

Lab 6

Advanced Timer Features

**Introduction:**

In this lab, we learned multiple ways to setup multiple clocks with interrupts. Part 1, we set up a one clock for the green led and one for the red led. Each of these clocks set to different time intervals. Part 2, was very similar to part 1, except we would stop both clocks at the same time for 4 seconds and then continue them through another clock. For part 3, we mess with a pulse-width modulation signal to set up the lowest brightness of the red led. Finally, for part 4, we elaborate more on part 3 by adding a clock to change through the brightness level from 0, 5, 10, 15, 20, 25, 30.

**Part 1:**

This part was very straightforward. If anyone from our class were only given the code to this part and saw what was missing, they could easily have filled in the missing pieces. The only part that may demand some brain power was calculating exactly what time was needed for the new clock. All the other code was simple programming to get what was being expected of us.

// 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 = 16384-1; // @ 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;

// 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**() {

P9OUT ^= greenLED; // Toggle the green LED

TA0CCR1 += 16384; // 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 = 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**;

}

**Questions:**

1. Show how the number of cycles for each channel is derived.
   1. **Channel 0: 32768 \* .1 = 3276.8 then round up for 3277**
   2. **Channel 1: 32768 \* .5 = 16384**

**Part 2:**

With this part, we needed three channels. One channel for the red led and one for the green led – these have the same setup as the previous part. The twist to this part was having the third channel take control of both leds by setting them off for four seconds and then resuming their functions for another four seconds consecutively. The trick to this was to have a flag that would let the hardware know whether the lights have been off or on, then do the opposite on interrupt.

// 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>

**#include** <stdbool.h>

**#define** redLED BIT0 // Red at P1.0

**#define** greenLED BIT7 // Green at P9.7

**bool** flag = **false**;

**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; // @ 32 KHz --> 0.1 seconds

TA0CCTL0 |= CCIE;

TA0CCTL0 &= ~CCIFG;

// Configure Channel 1 (write 3 lines similar to above)

TA0CCR1 = 4096-1; // @ 32 KHz --> 0.5 seconds

TA0CCTL1 |= CCIE;

TA0CCTL1 &= ~CCIFG;

// Configure Channel 2 (write 3 lines similar to above)

TA0CCR2 = 32768-1; // @ 32 KHz --> 4 seconds

TA0CCTL2 |= CCIE;

TA0CCTL2 &= ~CCIFG;

// Configure timer (ACLK) (divide by 4) (continuous mode)

TA0CTL = TASSEL\_1 | ID\_2 | MC\_2 | TACLR;

// 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 += 819; // 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**() {

// Detect Channel 1 interrupt

**if**((TA0CCTL1 & CCIFG)!=0)

{

P9OUT ^= greenLED;

TA0CCR1 += 4096;

TA0CCTL1 &= ~CCIFG;

}

// Detect Channel 2 interrupt

**if**((TA0CCTL2 & CCIFG)!=0)

{

**if** (flag == **false**)

{

TA0CCR0 += 32768;

TA0CCR1 += 32768;

flag = !flag;

}

**else**

{

flag = !flag;

}

P1OUT &= ~redLED;

P9OUT &= ~greenLED;

TA0CCR2 += 32768;

TA0CCTL2 &= ~CCIFG;

}

}

**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**;

}

**Questions:**

1. Show how the number of cycles for each channel is derived.
   1. **Channel 0: 32768/4 \* .1 = 819.2 then round down for 819**
   2. **Channel 1: 32768/4 \* .5 = 4096**
   3. **Channel 2: 32768/4 \* 4 = 32768**

**Part 3:**

I do not think much explanation is needed for this part since the majority of this code is given to you. I feel as if this was more of a look at how it is done and understand how it works, than anything else. After looking over the lines that are missing, the solution is intuitive.

// 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**(;;) {}

**return**;

}

**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**;

}

**Questions:**

1. Do larger values of TA0CCR1 results in higher or lower brightness? Explain.
   1. **Larger values of TA0CCR1 results higher brightness. When increasing TA0CCR1, you are increasing the number of cycles so the led has a chance to reach full brightness before it reverts to the off position. When decreasing the TA0CCR1, you shorten the number of cycles, so it does not reach the led’s maximum brightness before it turns back off.**

**Part 4:**

With this part, as with part 1 and part 2, the code is similar to its predecessor. Instead of us manually changing TA0CCR1, we are tasked to automatically alternate from 0, 5, 10, 15, 20, 25, and 30 each second. All this was done in an interrupt of a new timer I needed to add.

// 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 // Green at P9.7

**void** **config\_ACLK\_to\_32KHz\_crystal**();

**void** **main**(**void**) {

WDTCTL = WDTPW | WDTHOLD; // Stop WDT

PM5CTL0 &= ~LOCKLPM5;

P9DIR |= greenLED;

P9OUT &= ~greenLED;

// 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 = 0; // Modify this value between 0 and

// 32 to adjust the brightness level

// Configuring timer 1 with channel 0; period = 1 second

TA1CCR0 = 32768-1; // @ 32 KHz --> 1 second

TA1CCTL0 |= CCIE;

TA1CCTL0 &= ~CCIFG;

// (ACLK @ 32 KHz) (Divide by 1) (Up mode)

TA1CTL = TASSEL\_1 | ID\_0 | MC\_1 | TACLR;

// Engage a low-power mode

\_low\_power\_mode\_3();

**for**(;;) {}

**return**;

}

**#pragma** vector = TIMER1\_A0\_VECTOR

**\_\_interrupt** **void** **T1A0\_ISR**() {

P9OUT ^= greenLED; // Toggle the green LED

TA0CCR1 += 5; // Increase brightness level

**if** (TA0CCR1 == 30)

TA0CCR1 = 0;

TA1CCR0 += 32768; // Schedule the next interrupt

// Hardware clears Channel 0 flag (CCIFG in TA1CCTL0)

}

**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**;

}

**Questions:**

1. Explain how the two timer modules are interacting with each other.
   1. **One module is set for the PWM signal of the red led while the other is set as a regular timer for 1 second at 32 KHz. When the flag for the timer activates, the interrupt is called. At this point, the interrupt increases TA0CCR1 by 5 and then schedules the next interrupt and continues. This same sequence occurs until 30 is encountered, then resets back to 0.**

**Student Q&A:**

1. In the code with three channels, why did we divide ACLK so it becomes 8 KHz?
   1. **We needed a four second interval. This would be difficult to achieve since four seconds corresponds to 131,072 cycles which can’t be stored in a 16-bit register.**
2. In the first part, we configured two periodic interrupts using two channels of the timer. Is this approach scalable? For example, using a TimerA module with five channels, can we configure five periodic interrupts? Explain and mention in what mode the timer would run.
   1. **Yes, this approach is scalable. In order to do so, you would need to be in continuous mode and then configure your channels through TA0CCRx.**
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)
   1. **The next interrupt is scheduled at 16K. This is due to the overflow mechanism. When you add 40K to 40K, 16 bit will only count to 64K then rollback to 0, and then continue counting. Once you count to 64K, you will have a left over of 16K to count up to, from the previous added count, thus giving a 16K for the next interrupt.**

**Conclusion:**

I learned a lot from this lab. The multitasking of technology has always amazed me. I even loved how most of this lab required us to work on much of the interrupts, which made it more fun for me. The significant part in this lab would be a total of two parts, parts 1 and 2. These parts carried much of the information needed to further continue to parts 3 and 4.