**9/27/2020**

Joshua Frazer

Lab 3

Using the Timer

**Introduction:**

This lab was exclusively about how to configure the timer for the LEDs. In part 1, we learn how to operate the timer in continuous mode, which counts from 0 to 65535 then restarts. For part 2, we mastered up mode, which gave us more control over how many counts the timer should achieve. Finally the last part had us do our own design. For my design. I had both LEDs alternate through each second counting through the timer in continuous mode and in up mode.

**Part 1:**

For this part, as with our previous labs, we gained much information on the timers and their configurations in order to proceed for the following parts of this lab. With continuous mode being the sole part of this part, not much coding needed to be done once we have configured the timer to the 32 kHz crystal. Once we have done so, we just need to make sure we clear the flag after the flag is raised and toggle the LED.

// 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**) {

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

TA0CTL = TASSEL\_1 | ID\_0 | MC\_2 | TACLR;

// Ensure flag is cleared at the start

TA0CTL &= ~TAIFG;

// Infinite loop

**for**(;;) {

// Empty while loop; waits here until TAIFG is raised

**while**(!(TA0CTL & TAIFG)) {}

P1OUT ^= redLED; // Toggle the red LED

TA0CTL &= ~TAIFG; // Clear the 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**;

}

**Questions:**

* Write an analysis showing what delay you expect.
* 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?
  + **Yes**
* 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?

|  |  |
| --- | --- |
| **Period** | **ID** |
| **2 Seconds** | **0 /1** |
| **4 Seconds** | **1 /2** |
| **8 Seconds** | **2 /4** |
| **16 Seconds** | **3 /8** |

**Yes, these match with what I observed.**

**Part 2:**

This part is almost identical to part 1 except you must change the mode from continuous mode to up mode and add an end count. One rule that I had to keep in mind from lectures is to make sure I initialize the TACCR0 before the configuration of the timer. This eliminated and glitches and mess up relating to count at start up.

// Flashing the red LED with Timer\_A, up 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**) {

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

// Set timer period

TA0CCR0 = 32768;

// Timer\_A: ACLK, div by 1, up mode, clear TAR

TA0CTL = TASSEL\_1 | ID\_0 | MC\_1 | TACLR;

// Ensure flag is cleared at the start

TA0CTL &= ~TAIFG;

// Infinite loop

**for**(;;) {

// Empty while loop; waits here until TAIFG is raised

**while**(!(TA0CTL & TAIFG)) {}

P1OUT ^= redLED; // Toggle the red LED

TA0CTL &= ~TAIFG; // Clear the 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**;

}

**Questions:**

* What value of TA0CCR0 did you use?
  + **32768**
* 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.
  + **Yes**
* What value of TA0CCR0 achieves a delay of 0.1 seconds? Round up to the nearest integer and test this value.
* 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?
  + **I observe that the light ‘seems’ to stay on constantly.**

**Part 3:**

Part 3 gave me the opportunity to create my own design. My design was simple, instead of having one led to keep track of the timer I used two. I had it to where the red and green LEDs would alter which with the same TACCR0 as part 2 made it seem as if they were alternating as 1 second. I have two codes one in up mode and the other in continuous mode where I set TA0R as the start count of the value of TACCR0.

// CUSTOMIZE YOUR OWN - 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();

// Set timer period

TA0CCR0 = 32767;

// Configure Timer\_A

// Timer\_A: ACLK, div by 1, up mode, clear TAR

TA0CTL = TASSEL\_1 | ID\_0 | MC\_1 | TACLR;

// Ensure flag is cleared at the start

TA0CTL &= ~TAIFG;

// Infinite loop

**for**(;;) {

// Empty while loop; waits here until TAIFG is raised

**while**(!(TA0CTL & TAIFG)) {}

P9OUT ^= greenLED; // Toggle the green LED

P1OUT ^= redLED; // Toggle the red LED

TA0CTL &= ~TAIFG; // Clear the 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**;

}

// CUSTOMIZE YOUR OWN - Continuous 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();

// Set the initial start of the count for TAR

TA0R |= 0x8000;

// Configure Timer\_A

// Timer\_A: ACLK, div by 1, continuous mode, clear TAR

TA0CTL = TASSEL\_1 | ID\_0 | MC\_2 | TACLR;

// Ensure flag is cleared at the start

TA0CTL &= ~TAIFG;

// Infinite loop

**for**(;;) {

// Empty while loop; waits here until TAIFG is raised

**while**(!(TA0CTL & TAIFG)) {}

P9OUT ^= greenLED; // Toggle the green LED

P1OUT ^= redLED; // Toggle the red LED

TA0CTL &= ~TAIFG; // Clear the flag

TA0R |= 0x8000; // Reset the TAR to custom value

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 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**;

}

**Questions:**

No questions were delivered.

**Student Q&A:**

1. So far, we have seen two ways of generating delays: using a delay loop and using TimerA. Which approach provides more control and accuracy over the delays? Explain.
   1. **Using TimerA gives us more control and accuracy due to knowing the number of cycles and the frequency in which we are using. Through simple calculations we can get nearly any delay we would like. As with delay loops, they are more towards an estimation and bound to hardware specs which won’t give you the precision you would need.**
2. Explain the polling technique and how it’s used in this lab.
   1. **The polling technique we use in this lab is checking the active count of TAR. Once this TAR reaches it’s final value, we get the flag TAIFG which is what we are polling constantly to monitor if it is 1 or not. When it reaches 1, we then clear the flag and start over.**
3. Is the polling technique a suitable choice when we care about saving battery power? Explain.
   1. **This polling technique is not a suitable choice when we care about saving battery power. Reason being your CPU is constantly on checking a flag and counting.**
4. If we write 0 to TAR using a line code, does the TAIFG go to 1?
   1. **The TAIFG will not go to 1 because it didn’t roll back to 0 on its own.**
5. In this lab, we used TAIFG to time the duration. TAIFG is known as the TimerA Interrupt Flag, which is an interrupt flag. Were we using interrupts in this lab? Explain.
   1. **Even though we used the interrupt flag TAIFG we did not use any interrupts. An interrupt is a whole different function in which we would have within our code that would “interrupt” our code when a certain event happens. This interrupt flag does not “interrupt” our 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
   1. **The up mode gives us much more control over the timing duration.**

**Conclusion:**

This lab, for me, was a learning process. I would have to say that Part 2 became the most significant part of the lab. Without this part, I would not have been able to fully grasp how to calculate the timer to a specific delay like we learned in class. As for this, I am extremely thankful for this part. Once I completed this part, everything else became elementary.