EEL 4742C: Embedded Systems

Christopher Badolato
10/28/2019
EEL4742-0011
Lab 7 Concurrency via Interrupts

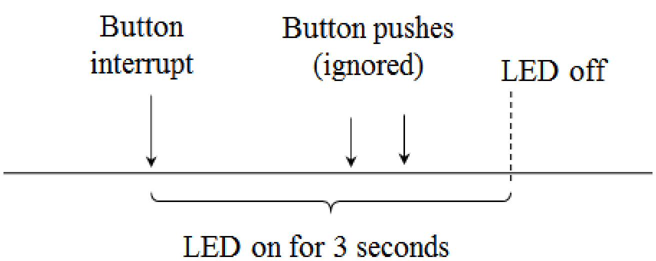
Introduction

In this lab we will be using multiple interrupts to control the LED toggling. First, we will configure a button interrupt to turn on the red LED for exactly 3 seconds, button interrupts will be shut off after first press. Then, we will configure the button to turn the LED on for 3 seconds, but this time, the LED will stay on for 3 seconds after the button is pressed again even if LED is already turned on. Finally, we will disable the "Button Bouncing" by implementing a debouncing algorithm, this algorithm takes two samples of the button separated by the maximum bounce duration.

Part 7.1: (Non-renewing Interval)

Our program for this part of the lab will turn on the LED for 3 seconds. This part will be non-renewing meaning, when the button is pressed the light will remain on for 3 seconds and then shut off is guaranteed. If the button is pressed within the 3 second timer, it will be ignored, we can see in the figure below an example of the timeline. (Taken from Abichar's lab manual)

Figure 7.1: LED turns on for three seconds when the button is pushed.



So, when s1 is pressed the Port1 ISR is triggered, we can follow below.

Looking at the figure, write a list of items to be done when the button raises an interrupt.

1. Disable the button interrupt (So it cannot be pressed until after the timer is complete)
2. Configure timer_A to 3 seconds (TACCR0 = 24576 -1)
3. Enable the timer_A interrupt
4. Configure the clock control
5. Make sure clock interrupt flag is clear.
6. Turn on the LED.

Write a list of items to be done when the timer raises an interrupt at the end of the three second interval.

1. Clear the timer interrupt

2. Enable the button interrupt for next press
3. Turn the LED back off
4. Disable the button flag for next press

Explain your configuration: clock signal and divider, timer mode, channel used
Which low-power mode did you use?

Using timer_A (32KHz)
Divided by 4 (8KHz)
Up Mode
Using Channel 0
we can get 3 seconds by:

3 seconds	8192 cycles	24576 cycles
	second	

Low power mode 3 because we are only using timer A.

```
//Christopher Badolato
//10/28/2019
//LAB 7.1
//EEL 4742L-0011


// Timer_A continuous mode, with interrupt, flashes LEDs
#include <msp430fr6989.h>


#define redLED BIT0 // Red LED at P1.0
#define greenLED BIT7 // Green LED at P9.7
#define BUT1 BIT1 // Button S1 at Port 1.1

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
    // Configuring buttons with interrupt
    P1DIR &= ~(BUT1); // 0: input
    P1REN |= (BUT1); // 1: enable built-in resistors
    P1OUT |= (BUT1); // 1: built-in resistor is pulled up to Vcc
    P1IE |= (BUT1); // 1: enable interrupts
    P1IES |= (BUT1);  // 1: interrupt on falling edge
    P1IFG &= ~(BUT1);  // 0: clear the interrupt flags
        // Enable the global interrupt bit (call an intrinsic function)
    config_ACLK_to_32KHz_crystal();

    _low_power_mode_3();
}
```

```
#pragma vector = TIMER0_A0_VECTOR
__interrupt void T0A0_ISR() {
        //clear timer interrupt
    TA0CCTL0 &= ~CCIE;
        //Enable button interrupt
    P1IE |= BUT1;
        //turn off LED
    P1OUT &= ~redLED;
        //Disable button flag
    P1IFG &= ~BUT1;
}

#pragma vector = PORT1_VECTOR // Write the vector name
__interrupt void Port1_ISR() {
            // Detect button 1 (BUT1 in P1IFG is 1)
    if((P1IFG & BUT1) == BUT1){
        P1IE &= ~BUT1; //disable button interrupt
            //configure timer_A
        TA0CCR0 = (24576-1);   //8khz --> 3 seconds
            //enable timer_A interrupt
        TA0CCTL0 |= CCIE;
            //config clock /4 up mode
        TA0CTL = (TASSEL_1|ID_2|MC_1|TACLR);
            //clear interrupt flag
        TA0CCTL0 &= ~CCIFG;
            //turn on red LED
        P1OUT |= redLED;
    }
}

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 7.2: (Renewing Interval)

Similar to above, in this part we will now use a renewing cycle meaning, when the button is pressed, the timer is reset each interval. If we are in the middle of a 3 second cycle, the timer will reset, and the LED will remain on. To do this we can just make sure each iteration that we keep the button interrupt enabled, that way each time it is pressed the timer restarts at 0. Below describes how each ISR code works.

Write a list of items to be done when the button raises an interrupt.

We need to start the timer and turn on the LED

1. Configure timer_A to 3 seconds (TACCR0 = 24576 -1)
2. Enable timer interrupts
3. Configure clock
4. Make sure interrupt flag is cleared
5. Turn on the redLED
6. Reset the button flag

Also, write a list of items to be done when the timer raises an interrupt at the end of the three-second interval.

All we need to do at the end of the timer interrupt is

1. Clear the timer interrupt flag
2. Shut off the redLED

```
//Christopher Badolato
//10/282019
//LAB 7.2
//EEL 4742L-0011


// Timer_A continuous mode, with interrupt, flashes LEDs
#include <msp430fr6989.h>


#define redLED BIT0 // Red LED at P1.0
#define greenLED BIT7 // Green LED at P9.7
#define BUT1 BIT1 // Button S1 at Port 1.1
#define BUT2 BIT2 // Button S2 at Port 1.2
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
        // Configuring buttons with interrupt
    P1DIR &= ~(BUT1); // 0: input
```

```
    P1REN |= (BUT1); // 1: enable built-in resistors
    P1OUT |= (BUT1); // 1: built-in resistor is pulled up to Vcc
    P1IE |= (BUT1); // 1: enable interrupts
    P1IES |= (BUT1);  // 1: interrupt on falling edge
    P1IFG &= ~(BUT1);  // 0: clear the interrupt flags
        // Enable the global interrupt bit (call an intrinsic function)
    config_ACLK_to_32KHz_crystal();

    _low_power_mode_3();
}


#pragma vector = TIMER0_A0_VECTOR
__interrupt void T0A0_ISR() {
        //clear timer interrupt
    TA0CCTL0 &= ~CCIE;
        //turn off LED
    P1OUT &= ~redLED;
        //Hardware resets interrupt flag
}

#pragma vector = PORT1_VECTOR // Write the vector name
__interrupt void Port1_ISR() {
        if((P1IFG & BUT1) == BUT1){
                //configure timer_A
            TA0CCR0 = (24576-1);   //8khz --> 3 seconds
                //enable timer_A interrupt
            TA0CCTL0 |= CCIE;
                //config clock /4 up mode
            TA0CTL = (TASSEL_1|ID_2|MC_1|TACLR);
                //clear interrupt flag
            TA0CCTL0 &= ~CCIFG;
                //turn on red LED
            P1OUT |= redLED;
                //reset button flag.
            P1IFG &= ~BUT1;
        }
}

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 7.3 (Button Debouncing)

In this part we will implement a debouncing algorithm. This algorithm takes multiple samples of the button separated by a bounce delay. We will wait out all the bouncing. During this interval, we will disable the button interrupt to avoid accidental interrupts this means we will ignore the button edges during the duration of the timer interrupt.

Write a list of items to be done when the button raises an interrupt.

1. If our P1 interrupt flag for button 1 is raised
2. Disable button interrupt
3. Reset TAR to 0
4. Enable the timer interrupt
5. Reset the button flag

Write a list of items to be done when the timer raises an interrupt.

1. If P1IN is 0. (Active low)
2. Toggle the LED
3. Turn off the timer interrupt
4. Enable the button interrupts
5. Reset button interrupt flag.

What value of the maximum bounce duration did you come up with?

0.02 seconds	8192 cycles	164 cycles
	second	

Explain your choices on the following: clock signal used by the timer, timer mode, channel used, low-power mode.

Using timer_A (32KHz)
Divided by 4 (8KHz) to get .02 seconds with 164 cycles
Up Mode to set our TACCR0 value (164-1).
Using Channel 0
Low Power Mode 3 (using ACLK)

```
//Christopher Badolato
//10/28/2019
//LAB 7.3
//EEL 4742L-0011


// Timer_A continuous mode, with interrupt, flashes LEDs
#include <msp430fr6989.h>


#define redLED BIT0 // Red LED at P1.0
#define greenLED BIT7 // Green LED at P9.7
#define BUT1 BIT1 // Button S1 at Port 1.1
#define BUT2 BIT2 // Button S2 at Port 1.2
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
        // Configuring buttons with interrupt
    P1DIR &= ~(BUT1); // 0: input
    P1REN |= (BUT1); // 1: enable built-in resistors
    P1OUT |= (BUT1); // 1: built-in resistor is pulled up to Vcc
    P1IE |= (BUT1); // 1: enable interrupts
    P1IES |= (BUT1);  // 1: interrupt on falling edge
    P1IFG &= ~(BUT1);  // 0: clear the interrupt flags

    config_ACLK_to_32KHz_crystal();

    TA0CCR0 = (164-1);   //8khz --> 0.02 seconds
        //enable timer_A interrupt
    TA0CCTL0 |= CCIE;
        //config clock /4 up mode
    TA0CTL = (TASSEL_1|ID_2|MC_1|TACLR);
        //clear interrupt flag
    TA0CCTL0 &= ~CCIFG;

    _low_power_mode_3();
}


#pragma vector = TIMER0_A0_VECTOR
__interrupt void T0A0_ISR() {
    if((P1IN & BUT1) == 0){
        P1OUT ^= redLED;
        TA0CCTL0 &= ~CCIE;
        P1IE |= BUT1;
        P1IFG |= ~BUT1;
    }
        //Hardware resets interrupt flag
}

#pragma vector = PORT1_VECTOR // Write the vector name
__interrupt void Port1_ISR() {
    if((P1IFG & BUT1) == BUT1){
        P1IE &= ~BUT1;
        TA0R = 0;
        TA0CCTL0 |= CCIE;
        P1IFG &= ~BUT1;
    }
}

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. For the debouncing algorithm we implemented, is it possible the LED will be toggled when the button is released? Explain.

Yes, because we have a check if P1IN at the corresponding BIT of the button is 0, it is possible this value bounces to 1 creating an error.

2. If two random pulses occur on the push button line due to noise and these pulses are separated by the maximum bounce duration, will our algorithm fail? Explain.

No, because the algorithm we implemented is disabling, for .02 seconds, our button press interrupt.

Conclusion

To conclude this experiment we learned to first, enabled concurrency via interrupts to create a long pulse turn on of the LED (Non-renewing interval). It will remain on for 3 seconds with each button press. Then, we changed the code to allow the LED to stay on for 3 seconds anytime the button is pressed (Renewing interval). Finally, we implemented a debouncing algorithm for each button press to make sure our button presses are accurate.