# CPE 325: Intro to Embedded Computer System

**Lab06**

**Debouncing, Interrupts, Switch / LED Interfacing, Clock Subsystem.**

**Submitted by**: Nolan Anderson

**Date of Experiment**: 10/11/2020

**Report Deadline**: 10/12/2020

## Introduction

This lab covers interrupt service routines in order to interface with switches and LED's on the MSP430. We use interrupts in C and assembly and use the clock subsystem and configuration to add delays to our program.

## Theory

**Interrupts**: Interrupts allow us to automatically break from the program flow when a certain set of conditions is met. When the interrupt we have written is finished, it returns to the line that it left from in the code. Interrupts are extremely useful for switching so that we can always do something when they are pressed.

**Clock Module in MSP430**: The clock module in MSP430 comes with 5 clock sources in the Unified Clock System (UCS) that we can modify. Modifying these values allows us to have full control over the clock frequency, changing the content of relevant clock module control registers, and having control over the frequency of other clock signals in peripheral devices. These 5 clock sources are XT1CLK, VLOCLK, REFOCLK, DCOCLK, and XT2CLK. Changing the clock sources looks a little something like this (from #2):

```
void configure_clock_sources()
{
    UCSCTL3 = SELREF_2;                 // Set DCO FLL reference = REFO
    UCSCTL4 |= SELA_2;                  // Set ACLK = REFO
    UCSCTL0 = 0x0000;                   // Set lowest possible DCOx, MODx
    // Loop until XT1,XT2 & DCO stabilizes - In this case only DCO has to stabilize
    do
    {
    UCSCTL7 &= ~(XT2OFFG + XT1LFOFFG + DCOFFG);   // Clear XT2,XT1,DCO fault flags
    SFRIFG1 &= ~OFIFG;                            // Clear fault flags
    } while (SFRIFG1&OFIFG);                      // Test oscillator fault flag
}
```

**Results & Observation**

Answering both 1d. and #1 on the assignment below. They are essentially the same question and I answered both below.

**d. What happens when SW2 is pressed while LED1 is blinking? Does that disrupt the blinking? Does SW2 function correctly? Explain.**

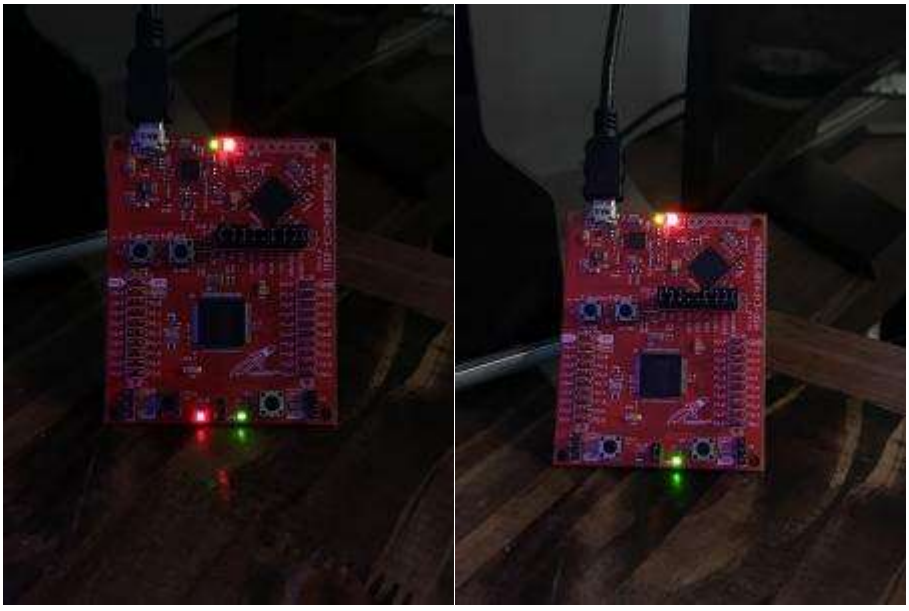
**1. For Q1, what do you observe when you press SW1 and immediately press SW1? Does the operation of SW1 affect the operation of SW2 or vice versa? Why/Why not?**

If you press switch one twice, it blinks LED 1 6 times and toggles LED2 twice, but only twice. While the switch one operation (interrupt) is operating, it does not affect the second switch. The reason for this is that the interrupt has already returned when the first switch is pressed so it does not affect the operation of the second switch. The code just keeps running and also searches for the second switch.

**2. Show the operation of Q2 by first pressing SW1 5-times successively. Then, press SW2 5-times successively.**

When switch 1 is pressed 5 times, it doubles for the first 3 clicks and then stops at 8Mhz. When switch 2 is pressed 5 times, the first 3 clicks halve the frequency until it stops at 1Mhz.

**SW1 5 times:** **SW2 5 times:**



As you can see, the SW1 5 times is blinking too fast for me to photograph that one is off and one is on.

## e. Calculate the LEDs blinking rate for each clock frequency and show your work.

Overall while loop delay time is: `__delay_cycles(500000);          // Delay of 250ms`

**1Mhz:**

```
__delay_cycles(33792);                    // 32 x 32 x 1 MHz / 32,768 Hz = 33792 =
MCLK cycles for DCO to settle
```

**2Mhz:**

```
__delay_cycles(62500);                    // 32 x 32 x 2 MHz / 32,768 Hz = 62500 =
MCLK cycles for DCO to settle
```

**4Mhz:**

```
__delay_cycles(125000);                   // 32 x 32 x 4 MHz / 32,768 Hz = 1255000 =
MCLK cycles for DCO to settle
```

**8Mhz:**

```
__delay_cycles(250000);                   // 32 x 32 x 8 MHz / 32,768 Hz = 250000 =
MCLK cycles for DCO to settle
```

## Observations:

Interrupts are very useful and will most likey continue to use them in projects outside of school and work. Unlike a function, you don't have to necessarily call anything in a loop. Once conditions are met, things just happen. It makes the code a lot nicer, cleaner, and I would suspect more efficient as well.

**Conclusion**

In this lab I better understood how to use debouncing and how to implement software delay's and interrupts. Similar to the observation section, interrupts prove to be very useful.

**Folder Link:**

https://drive.google.com/drive/folders/1_Y3ABMDhCUxc9phtQ8JKHCM8LDOK7k7J?usp=sharing

**Video Link:**

https://drive.google.com/file/d/1wL14zOPqKl9KIAa6fvwnq6VCC_G0owvK/view?usp=sharing

# Appendix

## Appendix 1

```
;-------------------------------------------------------------------------------
; Student:      Nolan Anderson
; Program:      main.asm
; Date:         Aug 20, 2020
; Input:        Switch one and 2 on the board.
; Output:       The red LED blinks three times and toggles the second LED, and the LED2 simply toggles
; Description:  When switch one is pressed, LED1 simply blinks three times at 1Hz and then
;               toggles LED2. When switch 2 is pressed, LED 2 simply toggles off and on.
;-------------------------------------------------------------------------------*/
            .cdecls C,LIST,"msp430.h"       ; Include device header file
;-------------------------------------------------------------------------------
            .def    RESET                   ; Export program entry-point to
                                            ; make it known to linker.
            .def SW1_ISR                    ; Define the SW1_ISR function.
            .def SW2_ISR                    ; Define the SW2_ISR function.
;-------------------------------------------------------------------------------
            .text                           ; Assemble into program memory.
            .retain                         ; Override ELF conditional linking
                                            ; and retain current section.
            .retainrefs                     ; And retain any sections that have
                                            ; references to current section.
;-------------------------------------------------------------------------------
RESET:       mov.w   #__STACK_END,SP         ; Initialize stackpointer
StopWDT:     mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer
;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
; bic: bit clear, and bis, bit set.
; P1.0 is Red LED, P4.7 is Green LED
; P1.1 is switch 2, P2.1 is switch 1
SETUP:
        bis.b   #0x01, &P1DIR           ; Set P1.0 as output, 0'b0000 0001
        bis.b   #0x80, &P4DIR           ; Set P4.7 as output, 0'b1000 0000
        bic.b   #0x01, &P1OUT           ; Turn P1.0 off.
        bic.b   #0x80, &P4OUT           ; Turn P4.7 off.
        ; Setting the Switch 2's data (i/o).
        bic.b   #0x02, &P1DIR           ; Set P1.1 as input for SW2
        bis.b   #0x02, &P1REN           ; Enable Pull-Up resister at P1.1
        bis.b   #0x02, &P1OUT           ; required for proper IO set up
        ; Setting the Switch 1's data (i/o).
        bic.b   #0x02, &P2DIR           ; Set P2.1 as input for SW1
        bis.b   #0x02, &P2REN           ; Enable Pull-up resistor at P2.1
        bis.b   #0x02, &P2OUT           ; Required for proper IO setup.
```

```
        ; Declaring interrupts and bits.
        bis.w   #GIE, SR                ; Enable Global Interrupts
        bis.b   #0x02, &P1IE            ; Enable Port 1 interrupt from bit 1
        bis.b   #0x02, &P1IES           ; Set interrupt to call from hi to low
        bis.b   #0x02, &P2IE            ; Enable Port 2 interrupt from bit 1
        bis.b   #0x02, &P2IES           ; Set interrupt to call from hi to low
        bic.b   #0x02, &P1IFG           ; Clear interrupt flag
        bic.b   #0x02, &P2IFG           ; Clear interrupt flag


Start:  cmp     #1, R5                  ; Compare 1 to R5
        jne     RLED                    ; If it is not one, jump to the Red LED function.
        clr     R5                      ; Clear the status of R5
        xor.b   #0x01, &P1OUT           ; Toggles the Red LED
RLED:   cmp     #1, R6                  ; Compare 1 to R6
        jne     Loop                    ; If it is one, then switch to the infinite loop.
        clr     R6                      ; Clear the status of R6
        mov     #6, R5                  ; Move 6 into R5
Cycle:  mov     #0xFFFF, R7             ; Move FFFF into R7, upper limit of a number.

Delay:  dec     R7                                      ; Decrement R7.
        nop
        nop
        nop
        nop
        nop
        jnz     Delay                   ; If R7 is not zero, jump back to delay.
        xor.b   #0x01, &P1OUT           ; Toggle the Red LED
        dec     R5                      ; Decrement R5
        jnz     Cycle                   ; If R5 is not zero, reset the R7 value and do this again.
        bit.b   #0x01, &P1OUT           ; And 0x01 and P1OUT
        xor.b   #0x80, &P4OUT           ; Toggle the green LED.
        jz      Loop                    ; If the status bit is zero, jump to loop to restart check.
Loop:   jmp     Start                   ; Loop here until interrupt
;-------------------------------------------------------------------------------
; P1_0 (Red) / P2_1 (SW1) interrupt service routine (ISR)
;-------------------------------------------------------------------------------
SW1_ISR:bic.b   #0x02, &P2IFG           ; Clear interrupt flag
        bit.b   #00000010b, &P2IN       ; Check if SW1 is pressed; (0000_0010 on P2IN)
        jnz     Exit1                   ; If not zero, SW is not pressed; loop and check again

Debounce_:      mov.b   #2000, R7       ; Set to (2000 * 10 cc )
SWD20ms_:       dec     R7              ; Decrement R7
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                jnz     SWD20ms_         ; If R7 is 0, then the loop will break and move on.
                bit.b   #0x02, &P2IN     ; Verify SW1 is still pressed
                jnz     Exit1            ; If not, wait for S2 press
                mov.b   #1, R6           ; Move 1 into R6
Exit1:          reti                    ; Return from interrupt
;-------------------------------------------------------------------------------
; P4_7 (Green) / P1_1 (SW2) interrupt service routine (ISR)
;-------------------------------------------------------------------------------
SW2_ISR:
                bic.b   #0x02, &P1IFG           ; Clear interrupt flag
                bit.b   #00000010b, &P1IN       ; Check if S2 is pressed; (0000_0010 on P1IN)
                jnz     Exit2                   ; If not zero, SW is not pressed; loop and check again
                xor.b   #0x80, &P4OUT           ; Toggle P4.7
Debounce:       mov.b   #2000, R7               ; Set to (2000 * 10 cc )
SWD20ms:        dec.w   R7                      ; Decrement R15
                nop
                nop
                nop
                nop
```

```
                nop
                nop
                nop
                jnz     SWD20ms         ; If R7 is 0, then the loop will break and move on.
                bit.b   #0x02, &P1IN    ; Verify S2 is still pressed
                jnz     Exit2           ; If not, wait for S2 press
                mov.b   #1, R7          ; Move 1 into R7
Exit2:          reti                    ; Return from interrupt
;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
            .global __STACK_END
            .sect   .stack
;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
            .sect   ".reset"        ; MSP430 RESET Vector
            .short  RESET
            .sect   ".int47"        ; PORT2_VECTOR,
            .short  SW2_ISR
            .sect   ".int42"        ; PORT1_VECTOR,
            .short  SW1_ISR
            .end
```

## Appendix 2

```
/*-------------------------------------------------------------------------------
* Student:      Nolan Anderson
* Program:      main.asm
* Date:         Aug 20, 2020
* Input:        Switch 1 and 2 on the board
* Output:       The LED's blink back and forth
* Description:  This code blinks the 2 LEDs back and forth on the MSP430 and uses debouncing
*               to delay the time and blink at different rates. The code essentially uses switch
*               one to double the hz rate (no higher that 8Mhz) and switch two to halve the
*               blinking rate. (No Lower than 1Mhz)
*-------------------------------------------------------------------------------*/
#include  <msp430.h>

void configure_clock_sources();
inline void Mhz1();                     // Change cf to 1 Mhz1
inline void Mhz2();                     // Change cf to 2 Mhz2
inline void Mhz3();                     // Change cf to 4 Mhz
inline void Mhz4();                     // Change cf to 8 Mhz

#define REDLED 0x01;
#define GREENLED 0x80;

int counter = 1;                        // Counter to check for what Mhz rate to blink at.

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;           // Stopping the watchdog timer

    P1DIR &= ~BIT1;                     // Set P1.1 as input (SW2)
    P1REN |= BIT1;                      // enable pull-up resistor
    P1OUT |= BIT1;
    P2DIR &= ~BIT1;                     // set P2.1 as input (SW1)
    P2REN |= BIT1;                      // enable pull-up resistor
    P2OUT |= BIT1;

    _EINT();                            // enable interrupts
    P1IE  |= BIT1;                      // Enable interrupt at P1.1 for Switch 1
```

```
    P1IES |= BIT1;                          // Enable hi->lo edge for interrupt
    P1IFG &= ~BIT1;                         // Clear any errornous interrupt flag

    P2IE  |= BIT1;                          // Enable interrupt at P2.1 for Switch 2
    P2IES |= BIT1;                          // enable hi->lo edge for interrupt
    P2IFG &= ~BIT1;                         // clear any errornous interrupt flag

    configure_clock_sources();              // Configure the clock sources
    Mhz1();                                 // Set initial blinking to 1 Mhz

    P1DIR |= REDLED;                        // Configure the P1.0 as output.
    P4DIR |= GREENLED;                      // Configure the P4.7 as output.

    P1OUT = P1OUT | REDLED;                 // Turn on LED 1.
    P4OUT = P4OUT & ~GREENLED;              // Turn off LED 2.

    while(1)
    {
        P1OUT ^= REDLED;                    // Toggle P1.0
        P4OUT ^= GREENLED;                  // Toggle P4.7
        __delay_cycles(500000);             // Delay of 250ms
    }
}


// this ISR handles the SW2 key press
#pragma vector = PORT1_VECTOR
__interrupt void PORT1_ISR(void)
{
    // let us clear the flag

    P1IFG &= ~BIT1;

    //debouncing section
    __delay_cycles(25000);

    // if SW1 is not pressed, return
    if((P1IN&BIT1)!=0x00)
        return;

    if(counter == 8)        // Are we blinking at 8Mhz?
    {
        Mhz4();             // Switch to 4Mhz blinking
        counter = 4;
    }
    else if(counter == 4)   // Are we blinking at 4Mhz?
    {
        Mhz2();             // Switch to 2Mhz blinking.
        counter = 2;
    }
    else if(counter == 2)   // Are we blinking at 2 Mhz?
    {
        Mhz1();             // Switch to 4Mhz blinking.
        counter = 1;        // Switch the counter equal to 1.
    }
    else if(counter == 1)    // Are we blinking at 1 Mhz?
    {
        Mhz1();             // Switch to 1Mhz blinking.
        counter = 1;        // Keep the counter at 1.
    }
}

// this ISR handles the SW1 key press
#pragma vector = PORT2_VECTOR
__interrupt void PORT2_ISR(void)
{
    // let us clear the flag
    P2IFG &= ~BIT1;
```

```
    //debouncing section
    __delay_cycles(25000);

    // if SW1 is not pressed, return
    if((P2IN&BIT1)!=0x00)
        return;

    if(counter == 8)        // Are we blinking at 8Mhz?
    {
        Mhz8();             // Switch to 8Mhz blinking.
        counter = 8;        // Keep the counter at 8.
    }
    else if(counter == 4)   // Are we blinking at 4Mhz?
    {
        Mhz8();             // Switch to 8Mhz blinking.
        counter = 8;        // Set the counter to 8.
    }
    else if(counter == 2)   // Are we blinking at 2 Mhz?
    {
        Mhz4();             // Switch to 4Mhz blinking.
        counter = 4;        // Set the counter equal to 4.
    }
    else if(counter == 1)   // Are we blinking at 1 Mhz?
    {
        Mhz2();             // Switch to 2Mhz blinking.
        counter = 2;        // Set the counter equal to 2.
    }
}


// ***** CHANGING THE CLOCK FREQUENCY TO 1 MHZ ***** //
void Mhz1()
{
    __bis_SR_register(SCG0);                // Disable the FLL control loop
    UCSCTL1 = DCORSEL_3;                    // Select DCO range Mhz1 operation
    UCSCTL2 = 32;                           // Set DCO Multiplier for Mhz1
                                            // (N + 1) * FLLRef = Fdco
                                            // (32 + 1) * 32768 = Mhz1
    __bic_SR_register(SCG0);                // Enable the FLL control loop
    __delay_cycles(33792);                  // 32 x 32 x 1 MHz / 32,768 Hz = 33792 = MCLK cycles for
DCO to settle
}

// ***** CHANGING THE CLOCK FREQUENCY TO 2 MHZ ***** //
void Mhz2()
{
    __bis_SR_register(SCG0);                // Disable the FLL control loop
    UCSCTL1 = DCORSEL_4;                    // Select DCO range Mhz2 operation,
    UCSCTL2 = 62;                           // Set DCO Multiplier for Mhz1
                                            // (N + 1) * FLLRef = Fdco
                                            // (62 + 1) * 32768 = Mhz2
    __bic_SR_register(SCG0);                // Enable the FLL control loop
    __delay_cycles(62500);                  // 32 x 32 x 2 MHz / 32,768 Hz = 62500 = MCLK cycles for
DCO to settle
}

// ***** CHANGING THE CLOCK FREQUENCY TO 4 MHZ ***** //
void Mhz4()
{
    __bis_SR_register(SCG0);                // Disable the FLL control loop
    UCSCTL1 = DCORSEL_4;                    // Select DCO range Mhz4 operation
    UCSCTL2 = 124;                          // Set DCO Multiplier for Mhz1
                                            // (N + 1) * FLLRef = Fdco
                                            // (124 + 1) * 32768 = Mhz4
    __bic_SR_register(SCG0);                // Enable the FLL control loop
    __delay_cycles(125000);                 // 32 x 32 x 4 MHz / 32,768 Hz = 1255000 = MCLK cycles for
DCO to settle
}
```

```
// ***** CHANGING THE CLOCK FREQUENCY TO 8 MHZ ***** //
void Mhz8()
{
    __bis_SR_register(SCG0);                // Disable the FLL control loop
    UCSCTL1 = DCORSEL_5;                    // Select DCO range Mhz8 operation
    UCSCTL2 = 249;                          // Set DCO Multiplier for Mhz8
                                            // (N + 1) * FLLRef = Fdco
                                            // (249 + 1) * 32768 = Mhz8
    __bic_SR_register(SCG0);                // Enable the FLL control loop
    __delay_cycles(250000);                 // 32 x 32 x 8 MHz / 32,768 Hz = 250000 = MCLK cycles for
DCO to settle
}

void configure_clock_sources()
{
    UCSCTL3 = SELREF_2;                   // Set DCO FLL reference = REFO
    UCSCTL4 |= SELA_2;                    // Set ACLK = REFO
    UCSCTL0 = 0x0000;                     // Set lowest possible DCOx, MODx
    // Loop until XT1,XT2 & DCO stabilizes - In this case only DCO has to stabilize
    do
    {
    UCSCTL7 &= ~(XT2OFFG + XT1LFOFFG + DCOFFG);   // Clear XT2,XT1,DCO fault flags
    SFRIFG1 &= ~OFIFG;                            // Clear fault flags
    } while (SFRIFG1&OFIFG);                      // Test oscillator fault flag
}
```