# CPE 325: Intro to Embedded Computer System

**Lab06**

**Interfacing Switches and LEDS on the MSPExp430 using Assembly and C programming**

**Submitted by**: Caleb Keller

**Date of Experiment**:_____09/30/2022_____

**Report Deadline**:_____10/02/2022_____

**Demonstration Deadline**: _____10/04/2022_____

# Introduction

*Write a brief discussion on what the lab is about. (Use the tutorial and write in your own words. DO NOT copy text).*

In this lab, I am writing assembly code to interface switches and LEDs on the MSPExp430 board. I am also creating a C program that will interface switches, as well as manipulate the output LEDs to blink at specified periods of time. I am using ISRs in both languages as well.

# Theory

*Write short notes on each topic discussed in lab.*

**Topic 1**: Interrupts and Interrupt Vectors

**a)** **Interrupts:** an interrupt allows an automatic break from the current instruction based on a set of conditions. When an interrupt condition is met, the program execution departs into a service routine that handles the interrupt event.

**b)** **Interrupt Vectors:** An interrupt vector is a piece of code where the interrupt instructions are held. When an interrupt condition is met within the main code, the instructions of the interrupt are performed within the ISR (Interrupt Service Routine).

**Topic 2**: Clock Module in MSP430

a.) **Clock Module:** Within the MSP430 family, there are several clock modules that a user can manipulate to change the processor clock frequency as well as the frequency of other clock signals that are used for peripheral devices. Within the MSP430FGT4618 device, there is an FLL+ clock module. The FFL+ is the "frequency locked loop". This module can be programmed to provide a range of core clock frequencies that are frequency locked to an external crystal. A frequency-locked loop is an electronic control system that generates a signal that is locked to the frequency of a controlled operator.

# Results & Observation

## Program 1:

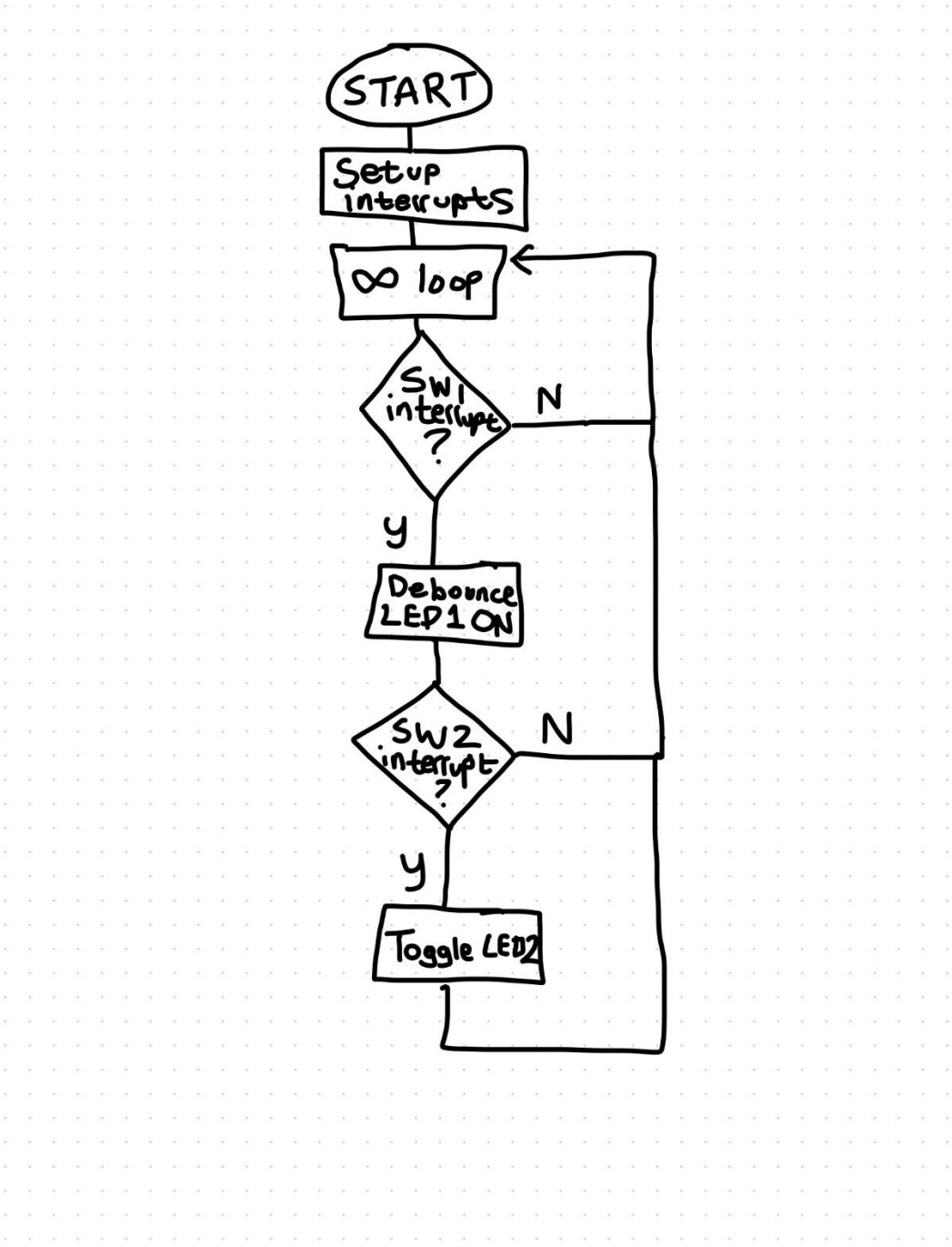
### Program Description:

*Explain your approach in solving the problem.*

To perform the tasks required in the first part of this Lab, I began writing the assembly portion of code that interfaced switches 1 and 2 as well as setup the instructions to perform ISRs to manipulate

LED1 and LED2 based on which of the switches is pressed. Since the LEDs must be initially off, I set the bits to 0. I the followed the tutorial and setup the code to enable interrupts. I then wrote functions to check which switch was pressed, debounced the switches, and what happened to each LED based on the routine.

## Program 1 Flowchart



## Program 2:

### Program Description:

*Explain your approach in solving the problem.*

In this program, I implemented C code to interface switches 1 and 2 and LED1 to perform events based on what happens when a specific switch is pressed. Using an infinite loop within main—I set the clock to 15 (1Hz) to toggle LED1 throughout the entirety of the program. When a switch is pressed, a specific ISR function is then implemented and once the switch is released, the program returns to main to continue the initial blinking state. Switch 1, Switch 2, and a combination of them both, each set the LED1 to blink at different rates.

### Frequency Calculations:

*using the given formula within Lab 06 Tutorial*

**2MHz:** (30.25 + 1) * 65536 = 2048000 -> 1056440/2MHz = 0.528 seconds = 1/0.528 = ~1.89 Hz

**4MHz:** (60 + 1) * 65536 = 3997696 -> 1056440/4MHz = 0.264 seconds = 1/0.264 = ~3.79 Hz

**8MHz:** (121 + 1) * 65536 = 7995392 -> 1056440/8MHz = 0.131 seconds = 1/0.131 = ~7.63 Hz

### Conclusion

While working on this program, I ran into many challenges and issues. My most prevalent challenge was formatting the calls to perform an ISR. I also had minor issues implementing the assembly code to accurately flow through the program. Overall, I now have a much clearer understanding of how ISRs work within assembly and the C programming languages.

## Appendix

*Your first code goes here, if any. Make sure you use a 1X1 table for this.*

*(Note: Make sure the code is readable, have comments. Also reduce spacing between lines to avoid lengthy reports.*

**Program 1 source code**

```
;-------------------------------------------------------------------------------
; MSP430 Assembler Code Template for use with TI Code Composer Studio
; Lab06 Part 1: assembly program to interface switches and LEDs on the MSP430FG4618
board
;               and perform ISRs based on switch press events
; Author: Caleb Keller
; Date: 10/01/2022
; CPE323-01
;-------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"       ; Include device header file

;-------------------------------------------------------------------------------
            .def    RESET                   ; Export program entry-point to
                                            ; make it known to linker.
```

```
             .def   SW1_ISR
;-------------------------------------------------------------------------------
            .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

Setup:      bis.b  #004h, &P2DIR; setting P2.2 to output direction (0000_0100)
                   bis.b  #002h, &P2DIR; setting P2.1 to output direction
                   bic.b  #004h, &P2OUT; set P2OUT  to 0001_0000 (LED1OFF)
                   bic.b  #002h, &P2OUT; set P2OUT to 0x0000_0100 (LED2 OFF)

                   bis.w  #GIE, SR           ; enable global interrupts
                   bis.b  #003h, &P1IE ; enable port 1 interrupt from bit 0
(0000_0001)
                   bis.b  #001h, &P1IES; set interrupt call from hi to low
                   bic.b  #001h, &P1IFG; clear interrpt flag
InfLoop:
                   jmp          $            ; loop here until interrupt
;-------------------------------------------------------------------
; P1.0 (SW1)  interrupt service routine (ISR)
;-------------------------------------------------------------------
SW1_ISR:
                   bic.b  #001h, &P1IFG ; clear interrupt flag
ChkSW:      bit.b  #02h, &P1IN  ; check if SW2 is pressed (0000_0010) on P1IN)
                   jz           Debounce                  ; if not zero, SW2 is
not pressed
                   bit.b  #01h, &P1IN               ; if zero go to ISR
                   jz           Debounce
                   jnz          LExit                     ; if not zero go to end
Debounce:   mov.w  #2000, R15   ; set to (2000 * 10 cc)
SWD20ms2:   dec.w  R15
                   nop
                   nop
                   nop
                   nop
                   nop
                   nop
                   nop
                   jnz          SWD20ms2           ; delay over?
                   bit.b  #00000010b, &P1IN   ; verify sw2 still pressed
                   jnz          LExit         ; if not wait for sw2 press
                   bit.b  #002h, &P2OUT ; output LED2
                   jnz          LED2OFF                   ; turn LED2 off
LED2ON:            bis.b  #002h, &P2OUT; LED2 ON
SW2Wait:    bit.b  #002h, &P1IN ; test SW1
                   jz           SW2Wait       ; wait until SW1 is released
                   jmp          LExit

LED2OFF:    bic.b  #002h, &P2OUT; off
```

```
                    bit.b  #00000010b, &P1IN
                    jz           SW2Wait
Debounce1:    mov.w  #2000, R15    ; set to (2000 * 10 cc)
SWD20ms:      dec.w  R15           ; decremen R15
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    jnz          SWD20ms
                    bit.b  #00000001b, &P1IN   ; verify SW1 is still pressed
                    jnz          LExit1        ; if not, wait for SW1 press
LED1ON:             bis.b  #0x04, &P2OUT; turn on LED1
SW1Wait:      bit.b  #0x01, &P1IN ; test SW1
                    jz           SW1Wait       ; wait until SW1 is released
                    bic.b  #0x04, &P2OUT
                    bis.b  #000h, &P1IES ; signal goes low to high
LExit1:             reti



;Chk_SW1:     bit.b  #01h, &P1IN  ; chck if SW1 is pressed (0000_0001) on P1IN
;
;                   jnz          LExit1 ; if not 0 SW is not pressed lop and check
again


LExit:       reti                              ; return from interrupt


;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------



;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
            .global __STACK_END
            .sect   .stack

;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
            .sect   ".reset"                ; MSP430 RESET Vector
            .short  RESET
            .sect   ".int20"     ;P1.x vector
            .short  SW1_ISR
            .end
```

**Program 2 Source Code**

```
/*
 * Lab06 Part 2: C program to interface switches and LED1 to perform specific ISRs
 * Author: Caleb Keller
 * Date: 10/01/2022
 * CPE323-01
 */


#include <msp430.h>


#define SW1 BIT0&P1IN   // SW1
#define SW2 BIT1&P1IN   // SW2

unsigned char SW1Pressed = 0;
unsigned char SW2Pressed = 0;

void main(void)
{
       WDTCTL = WDTPW+WDTHOLD;    // stop watchdog timer
       FLL_CTL0 |= DCOPLUS + XCAP18PF; // DCO+ set, freq = xta x D x N+1
       SCFI0 |= FN_2; // DCO range control
       SCFQCTL = 15; // setting clock to 0.5 MHz
       _EINT();
       SW1Pressed = 0;
       SW2Pressed = 0;
    // interrupt setup
    P1IE |= BIT0;   // P1.0 Interrupt Enabled sw1
    P1IES |= BIT0;  // P1.0 h2l edge
    P1IFG &= ~BIT0; // P1.0 IFG cleared

    P1IE |= BIT1;   // P1.1 Interrupt Enabled sw2
    P1IES |= BIT1;  // P1.1 h2 edge
    P1IFG &= ~BIT1; // P1.1 IFG cleared
    // LEDs Setup
    P2DIR |= BIT2; // LED1 is ON (0000_000[1])
    P2OUT &= ~BIT2; // LED1 initialized to OFF
    P2OUT = 0x00;

    for(;;) // infinite loop
    {
        //P1IFG &= ~(BIT0|BIT1);
        if((SW1) != 0 && (SW2) != 0)
        {
            SCFQCTL = 15; // setting clock to 0.5 MHz if not pressed
        }
          __delay_cycles(1056440); // setting to 0.5 MHz usng formula from tutorial
        P2OUT ^= BIT2; // TOGGLE LED1 constantly
    }
```

```
}
#pragma vector = PORT1_VECTOR
__interrupt void Por1_ISR (void)
{
    if((SW1) == 0 && (SW2) != 0)
     {
        P2OUT &= ~BIT2; // TOGGLE LED1
        SCFQCTL = 60; // 4 MHz

        /*if(SW1Pressed == 0)
        {
            SW1Pressed = 1;
            P2OUT |= BIT2; //  LED1 ON
            SCFQCTL = 60; // 4 MHz
            P1IFG &= ~BIT0; // clearing
            P1IES |= ~BIT0; // hi 2 low
        }
        else if(SW1Pressed == 1)
        {
            SW1Pressed = 0;
            P2OUT &= ~BIT2; //  LED1 OFF
            SCFQCTL = 15; // 4 MHz
            P1IFG &= ~BIT0; // clearing
            P1IES &= BIT0; // low2hi

        }*/

     }
    if((SW2) == 0 && (SW1) != 0)
    {
        P2OUT &= ~BIT2; // TOGGLE LED1
        SCFQCTL = 30.25; // 2 MHz
        /*if(SW2Pressed == 0)
        {
            SW2Pressed = 1;
            P2OUT |= BIT2; // TOGGLE LED1
            SCFQCTL = 30.25; // 2MHz
            P1IFG &= ~BIT1; // clearing
            P1IES &= ~BIT1; // hi 2 low
        }
        else if(SW2Pressed == 1)
        {
            SW2Pressed = 0;
            P2OUT &= ~BIT2; // TOGGLE LED1
            SCFQCTL = 15; // 4 MHz
            P1IFG &= ~BIT1; // clearing
            P1IES |= BIT1; // low2hi

        }*/
    }
    if((SW1) == 0 && (SW2) == 0)
    {
       P2OUT &= ~BIT2; // TOGGLE LED1
       SCFQCTL = 121;  // (121 + 1) x 32768 x 2 = 7.99 MHz
```

```
    }

    P1IFG &= ~(BIT0|BIT1); // clearing
}
```

**Make sure to submit a single pdf file only**