CPE 325: Intro to Embedded Computer Systems

Lab06

MSP430 Interrupt and Clock Subsystem

Submitted by: Michael Agnew

Date of Experiment: 2/19/2024

Report Deadline: 2/21/2024

Demonstration Deadline: 2/26/2024

Introduction:

The main objective of this lab was to understand several important subjects. The first of which being configuring LEDs and switches in the MSP430's assembly language. The second was understanding how to use the Interrupt Service Routine (ISR) that is also provided by the MSP430. The third important topic that was covered by this tutorial was the clock subsystem that exists in the MSP430 and how to change it with the C programming language. There are various tools that can be used to change the clock frequency of the code; which would be important later on when it was required to be implemented in the given assignment.

Theory:

What are Interrupts?: With a microcontroller such as the MSP430, it is possible and extremely useful to have interrupts in one's program which allow them to break out of a current state and execute a set of instructions based on the interruption. When an interrupt occurs, the main program finishes its current assignment before branching out to its Interrupt Service Routine (ISR) which will handle the interrupt based on the instructions given by the programmer. After completing this routine, the ISR will then return to the point where the interruption broke off from the main code and the program will continue where it left off.

Example: An example of this would be how, in the given assignment, one uses the ISR to branch to LED manipulation based on which switch is being pressed. The program then correctly changes the state of the LED according to the instructions it is given; and then returns back to the main code.

Usefulness: So, with this knowledge being known, one can understand this topic and possibly realize the use of such a routine. The ISR allows one to essentially nest subroutines inside of a

program without having to manually make subroutine calls. In the previous lab, subroutines were learned and implemented with the MSP430's assembly language. The way that these were used was that the programmer was required to pass parameters to the given subroutine and call it manually in order to have it perform the necessary operations. Whereas with interrupts, one can simply enable the interrupt flag based on the switches, and then enter into the ISR based on the status of the switch and perform the necessary operations without any sort of passed parameters or subroutine calls. This is extremely useful and will help any programmer wanting to make more complex code in MSP430 assembly.

Clock Module:

The clock module is a very important part of the MSP430 and is very useful for any programmer. One can change the clock module control registers in order to change the clock frequency of the MSP430's processor. Not only this, but it can also alter the clock signals for other peripheral devices. There are five different clock sources and three different clock signals in the MSP430.

XT1CLK: This is a low or high frequency oscillator which is used for devices such as watch crystals or standard crystals. It can be used as a clock reference in the FLL (Frequency Locked Loop).

VLOCLK: This is an internal oscillator which is very low power and very low frequency which usually sits around 10-kHz.

REFOCLK: This is another low-frequency oscillator which is higher than the previous one at 32768-Hz. This clock can also be used as a reference into the FLL.

DCOCLK: This is another oscillator which can be controlled digitally and is stabilized by the FLL.

XT2CLK: This is a high-frequency oscillator that is capable of use regarding standard crystals or resonators. This is also a clock that can be used as a reference for the FLL.

Clock Signals: Knowing these different clocks in the UCS (Unified Clock System) one should also be aware of the clock signals that are present in the UCS.

ACLK: This is the auxiliary clock. This is selectable as XT1CLK, REFOCLK, VLOCLK, DCOCLK, DCOCLKDIV, and XT2CLK when it is available. This clock signal is selected by individual peripheral modules.

MCLK: This is the master clock of the system. This clock signal is selectable as T1CLK, REFOCLK, VLOCLK, DCOCLK, DCOCLKDIV, and XT2CLK when it is available. This clock signal is made use of by the MSP430's system and CPU.

SMCLK: This is the subsystem master clock. This is a software that is selectable as T1CLK, REFOCLK, VLOCLK, DCOCLK, DCOCLKDIV, and XT2CLK when it is available. This software is made use of by individual peripheral modules just like ACLK.

How to Change Clock Frequency?: The way that one can change the clock frequency of the MSP430 is by taking advantage of the registers UCSCTL1 and UCSCTL2. One will also need to use the function __bis_SR_register(SCG0).

```
__bis_SR_register(SCG0);                // Disable the FLL control loop
UCSCTL1 = DCORSEL_5;                    // Select DCO range 16MHz operation

UCSCTL2 = FLLD_1 + 66;                        // Set DCO Multiplier for 2MHz
                                        // (N + 1) * FLLRef = Fdco
                                        // (66 + 1) * 32768 = 2MHz
__bic_SR_register(SCG0);                // Enable the FLL control loop
```

Figure 1. Code Snippet Changing Clock Frequency.

The given figure illustrates the code needed to change the clock frequency of the processor. This is done by using the function __bis_SR_register(SCG0) in order to pause the FLL control loop. Then, the range of the frequency is set by setting UCSCTL1 equal to DCORSEL_5, which sets the range up to 16Mhz for the operation. The UCS register UCSCTL2 is then changed to be equal to FLLD_1 + 66 in order to change the frequency to 2Mhz. The 66 comes from the formula (N+1)*32768 which is used to calculate the needed number to meet the correct frequency.

Problem 01:

Program Description: The first problem that was assigned for this lab was using the MSP430's assembly language to interface with the board and configure its LEDs and switches using interrupts and interrupt vectors. The way that the interrupts were being treated in this assignment was that the code infinitely loops in the main part of the program but when an "interrupt" is reached, such as one of the switches being pressed, the program will branch to the

Interrupt Service Routine based on which of the switches was pressed. It will then execute the required steps in order to “handle” the interruption, or switch being pressed, that occurred. It will handle this interruption by blinking or toggling the required LEDs in the way that the assignment laid out. For this to be completed, the following truth table for the switches and LEDs needs to be implemented with the code.

SUMMARY	**LED1 (Red LED)**	**LED2 (Green LED)**
Default State (No Press)	OFF	OFF
SW1 PRESSED	OFF	ON
SW1 PRESSED AGAIN	OFF	OFF
SW2 PRESSED	BLINK 4 TIMES 2HZ	AFTER COMPLETION OF LED1: ON

Figure 2. Table Illustrating State of LEDs Based on Switch Status.

Table Analysis: Analyzing this table, one can see the way that these switches and LEDs need to be implemented. When Switch 1 is pressed, the green LED will toggle on, and when Switch 1 is pressed again, the green LED will toggle off. In other words, the action of pressing Switch 1 will change the state of the green LED. As for Switch 2, when it is pressed, the red LED will blink 4 times at 2hz. After the completion of this blinking, the green LED will then change its state to either on or off depending on the previous state.

Program Design: The way that this program was designed, it starts out by initializing all of the needed ports, LEDs, and switches and assigns them to the correct ports and bits on the board. After this, the code enters into an infinite loop which waits for one of the switches to be pressed. When Switch 1 is pressed, the program clears the interrupt flag and debounces the switch

manually before checking to make sure that the switch is still pressed. If it is, then the code checks the previous state of the green LED and turns it either on or off accordingly. After it changes the state of the green LED, it returns from the ISR and jumps back to the infinite loop in the main body of the program. When Switch 2 is pressed, the program initializes a counter register to 4 to serve as the loop counter. It then clears the interrupt flag and debounces switch 2 and makes sure it is still pressed afterwards. If so, then it blinks the red LED 4 times based on the count-controlled loop, at a frequency of 2hz. This delay is implemented by a subroutine that has a 50,000 iteration loop which acts as a half second delay. This subroutine executes and then returns to its place in the blinking loop. Once this loop finishes executing, the program then changes the state of the green LED.

Problem 01 Questions:

1. **What happens when switch 1 is pressed while LED1 is blinking?**

A: For the code that was written for this assignment, pressing switch 1 while LED1 is blinking does not have any effect on the LEDs. The program simply waits for LED1 to stop blinking and then toggles LED2 as normal.

2. **Does Pressing Switch 1 while LED1 is blinking disrupt the blinking?**

A: No, pressing switch 1 while LED1 is blinking does not disrupt the blinking in any way. This is because the ISR executes fully for one of the switches being pressed and does not execute the operation of another switch until the first one is completed.

3. **Does Switch 1 function correctly? Elaborate.**

A: Yes, Switch 1 does function correctly. When it is pressed, it toggles LED2 (the green LED) exactly how it should, and only changes it when switch 1 is pressed again. This enacts the situation that the assignment required where pressing switch 1 changes the state of LED2.

Problem 02:

Program Description: The second assignment that was given is concerning the clock subsystem of the MSP430. The point of this assignment is to once again interface with the switches and LEDs of the MSP430. The objective of this program is to have LEDs blink alternately with a delay of a 50,000 iteration for loop. This is a constant rate and will not be changed. However, what will be changed is the clock speed of the MSP430 itself which will be done by means of the switches being pressed. When Switch 1 is pressed, the clock frequency will be changed to 8Mhz, and when Switch 2 is pressed, the clock frequency will be halved. What this means is that when Switch 1 is pressed, the frequency of the clock is changed to 8Mhz, and when Switch 2 is pressed, it will be changed to 4Mhz, and again 2Mhz, and so forth. This will adjust how fast the LEDs will blink. While the LEDs are blinking at a constant rate relative to the clock frequency, the switches will change the speed of the clock frequency which will change the rate at which the LEDs blink.

Program Design:

The way that this program works is it initializes the clock frequency at a rate of 2Mhz per the assignment's instruction. It then enters into an infinite loop to check if the switches are pressed. If none of the switches are pressed then it simply toggles the red and green LED. This causes the LEDs to blink alternately. If switch 1 is pressed, then the program sets the clock frequency to

8Mhz. Finally, if switch 2 is pressed, the code halves the clock frequency accordingly based on how many times switch 2 has been pressed. Each switch press is properly debounced and the LEDs blink at a rate given by a 50000 iteration for loop.

LED Blink Calculations:

Groundwork: Part of the assignment asks for what the blinking rate of the LEDs will be based on the current frequency of the processor. The assignment required a 50,000 iteration for loop to be used as delay. From previous labs, it is known that a 50,000 iteration for loop will generate a delay of 500,000 clock cycles which is 0.5s. When converting this to hz, one can see that this acts as a delay of 250ms, or, 2hz.

Calculations: Now, for this assignment There are 4 different states: 8Mhz, 4Mhz, 2Mhz, and 1Mhz. If one knows that there are ~1,000,000 clock cycles per 1Mhz, then it can be concluded that there are ~8,000,000 clock cycles for 8Mhz. Knowing that there are 1,000,000 cc (clock cycles) in 1Mhz, and that there is a delay equal to 500,000, one can say that c = #cc. Which is 1,000,000, b = 0.5s for delay, and a = 500,000 cc for delay as well. This means that $a = b$ and that $2a = c$. If one multiplies c by 8 in order to reach 8Mhz, one will need to divide a by 4 in order to preserve the ratio of the equality. This means that $500,000/4 = 125.\ 125/2 = 62.5$ which comes out to around ~19hz blink rate. This can be repeated for 4 and 2 as well. 4Mhz comes out to be $500,000/2 = 250000.\ 250000/2 = 125000$, which is 8hz. And for 2, $500,000/2 = 250000$, which is 4hz.

Conclusion:

Fortunately, there were no issues that were encountered over the course of this lab. Everything that was sought out to be completed was done so in the proper way and according to the assignment's parameters. This lab teaches one to configure ports and LEDs with the MSP430 assembly language and implement them by means of interrupts and using the ISR. As for the second part of the assignment, the lab also teaches one how to make use of changing the clock frequency of the MSP430 in the C programming language. Instead of manually changing the delay of the LEDs, the code makes use of the change in clock frequency of the MSP430 itself. This is a very important topic and should definitely be considered moving forward.

Appendix:

Problem 01:

```
;-------------------------------------------------------------------------------
; File:         Lab6final.asm
; Function:     Handles switches and LEDs with interrupts
; Description:  Chipi Chipi, chapa chapa Dubidubi, dabadaba Mágico mi dubidubi boom,
boom, boom, boom
; Input:        Switches 1 and 2
; Output:       LED1 and LED2
; Author(s):    Michael Agnew, ma0133@uah.edu
; Date:         February 19, 2024
; Revised:      Febryary 21, 2024
;-------------------------------------------------------------------------------
        .cdecls C,LIST,"msp430.h"    ; Include device header file
```

```
;-------------------------------------------------------------------------------
        .def    RESET                   ; Export program entry-point to
                                        ; make it known to linker.
        .def   SW2_INT
        .def   SW1_INT
;-------------------------------------------------------------------------------
        .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 stack pointer
StopWDT:        mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer
;-------------------------------------------------------------------------------
Setup:
        bis.b   #001h, &P1DIR   ; Setting up LED1
        bic.b   #001h, &P1OUT    ; Setting up Output to LED1
        bic.b    #002h, &P1DIR    ; Setting up SW2 Direction
        bis.b    #002h, &P1REN    ; Setting up SW2 Ren
        bis.b    #002h, &P1OUT     ; Setting up SW2 Output

        bis.b    #0x80, &P4DIR    ; Setting up LED2
        bic.b    #0x80, &P4OUT    ; Setting up Output to LED2
        bic.b    #002h, &P2DIR    ; Setting up SW1 Direction
        bis.b    #002h, &P2REN    ; Setting up SW1 Ren
        bis.b    #002h, &P2OUT     ; Setting up SW1 Output

        bis.w    #GIE, SR        ; Enabling Global Interrupts
        bis.b    #002h, &P1IE    ; Enabling Switch 2 Interrupts
        bis.b    #002h, &P1IES
        bic.b    #002h, &P1IFG

        bis.b    #002h, &P2IE    ; Enabling Switch 1 Interrupts
        bis.b    #002h, &P2IES
        bic.b    #002h, &P2IFG

InfLoop:
        jmp     $                ; Loop here until interrupt

;-------------------------------------------------------------------------------
; Interrupt service routine. Caused by one of two switches being pressed
;-------------------------------------------------------------------------------
SW2_INT:
```

```
                mov.w    #4, R10
                bic.b    #002h, &P1IFG     ; clear switch 2 interrupt flag
                bit.b    #02h, &P1IN     ; check if switch 2 is pressed
                jnz     SW2Exit                  ; if switch 2 is not pressed, then exit

                mov.w   #2000, R15              ; Set to (2000 * 10 cc )
SW2Deb:         dec.w   R15                     ; Decrement R15
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        jnz     SW2Deb                  ; Is the delay over?
        bit.b   #00000010b, &P1IN               ; Make sure that SW2 is still pressed
        jnz     SW2Exit                 ; If SW2 is not pressed, exit interrupt

sw2pres:    bis.b    #001h, &P1OUT              ; Turn on the red LED
        call    #Delay                          ; 250ms Delay
        bic.b    #001h, &P1OUT          ; Turn off the red LED
        call    #Delay                   ; 250ms Delay
        dec.w   R10                             ; Decrement the loop counter
        jnz     sw2pres                         ; if the loop counter is not 0, then loop again

LED2on:     xor.b    #0x80, &P4OUT          ; Turn on the green LED

SW2Exit:    reti                                        ; Return from the interrupt

SW1_INT:

        bic.b   #002h, &P2IFG           ; Clear switch 1 interrupt flag
ChkSW1:         bit.b   #02h, &P2IN             ; Check if SW1 is pressed
                                ; (0000_0010 on P1IN)
        jnz     SW1Exit                 ; If not zero, SW is not pressed
                                ; loop and check again
                mov.w   #2000, R15              ; Set to (2000 * 10 cc )
SW1Deb:         dec.w   R15                     ; Decrement R15
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        jnz     SW1Deb                  ; Delay over?
```

```
        bit.b   #00000010b,&P2IN     ; Verify SW2 is still pressed
        jnz     SW1Exit              ; If not, wait for SW2 press

        mov.b    &P4OUT, R6          ; check if LED1 on
                 and.b     #0x80, R6
        jnz      LEDoff
        jmp     LEDon
LEDon:          bis.b   #0x80, &P4OUT         ; Turn on LED1
                 jmp     SW1Exit


Delay:
        mov.w   #50000, R12           ; 50k iteration loop makes 500ms which is a half second.
2hz = 0.5s
loopy_boy:  dec.w   R12                                  ; decrement counter register
        jnz      loopy_boy             ; if counter is not 0, jump back to the top of the loop
                 ret                                              ; return from subroutine

LEDoff:         bic.b  #0x80, &P4OUT         ; Turn off LED1
SW1Exit:        reti                         ; Return from interrupt
;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
        .global __STACK_END
        .sect   .stack

;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
        .sect   ".reset"             ; MSP430 RESET Vector
        .short  RESET
        .sect   ".int42"
        .short    SW1_INT
        .sect    ".int47"
        .short    SW2_INT
        .end
```

Problem 02:

```
/*-------------------------------------------------------------------------------
```

```
 * File:          Lab6CFinal.c
 * Function:    Blinks the LEDs alternately. Clicking the input switches allows one to change the Clock Frequency of the processor
 * Description: This C program turns on the LEDs and blinks them alternately according to the current
 * calibration of the clock frequency. Debouncing delay of 20ms is added for switch presses.
 * Clocks:      ACLK = 32.768kHz, MCLK = SMCLK = default DCO (~1 MHz)
 *              MSP-EXP430F5529LP
 *              -----------------
 *              /|\|                 |
 *              | |                 |
 *              --|RST              |
 *              |          P1.0|-->LED1(RED)
 *              |          P2.1|<--SW1
 *              |          |
 * Input:       Pressing S1 and S2
 * Output:      LED1 and LED2 are blinked alternately.
 * Author(s):   Michael Agnew, ma0133@uah.edu
 * Date:        2/22/2024
 * ------------------------------------------------------------------------*/
#include <msp430.h>

#define S1 P2IN&BIT1                    // switch 1 config
#define S2 P1IN&BIT1                    // switch 2 config
#define REDLED 0x01                     // Mask for BIT0 = 0000_0001b
#define GREENLED 0x80                   // Mask for BIT7 = 1000_0000b

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


        P1DIR |= REDLED; // Set LED1 as output
        P1OUT &= ~REDLED; // Initially turn off LED1

        P4DIR |= GREENLED; // Set LED2 as output
        P4OUT |= GREENLED; // Initially turn on LED2

        P2DIR &= ~BIT1;
        P2REN |= BIT1;
        P2OUT |= BIT1;

        P1DIR &= ~BIT1; // Set P1.1 as input for SW2 Input
        P1REN |= BIT1; // Enable the pull-up resistor at P1.1
        P1OUT |= BIT1; // Required for proper IO
```

```
/*******************I know exactly what all of this does*******************/

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

__bis_SR_register(SCG0);            // Disable the FLL control loop
UCSCTL1 = DCORSEL_5;                      // Select DCO range 16MHz operation

UCSCTL2 = FLLD_1 + 66;                    // Set DCO Multiplier for 8MHz
                        // (N + 1) * FLLRef = Fdco
                        // (66 + 1) * 32768 = 2MHz
__bic_SR_register(SCG0);            // Enable the FLL control loop


unsigned int i = 0;
unsigned int counter = 0;
while(1)                            // Infinite loop
{
for (i = 0; i< 50000; i++);
if ((S1) == 0)              // If S1 is pressed
{
for (i = 2000; i > 0; i--); // Debounce ~20 ms please
P1OUT ^= REDLED;
P4OUT ^= GREENLED;
counter = 0;
__bis_SR_register(SCG0);            // Disable the FLL control loop
UCSCTL1 = DCORSEL_5;                      // Select DCO range 16MHz operation
UCSCTL2 = FLLD_1 + 249;                   // Set DCO Multiplier for 8MHz
                                    // (N + 1) * FLLRef = Fdco
                                    // (249 + 1) * 32768 = 8MHz
__bic_SR_register(SCG0);            // Enable the FLL control loop
}
else if ((S2) == 0)
{ // else if switch 2 is pressed
for (i = 2000; i > 0; i--); // Debounce ~20 ms
P1OUT ^= REDLED;
```

```
        P4OUT ^= GREENLED;
        counter++; // increment counter
        if (counter == 1) // if counter is 1
        {

                __bis_SR_register(SCG0);            // Disable the FLL control loop
                UCSCTL1 = DCORSEL_5;                    // Select DCO range 16MHz
operation

                UCSCTL2 = FLLD_1 + 124;                 // Set DCO Multiplier for 4MHz
                                        // (N + 1) * FLLRef = Fdco
                                        // (124 + 1) * 32768 = 4MHz
                __bic_SR_register(SCG0);            // Enable the FLL control loop
        } // end of counter =1
        else if (counter == 2)
        {

                __bis_SR_register(SCG0);            // Disable the FLL control loop
                UCSCTL1 = DCORSEL_5;                    // Select DCO range 16MHz
operation

                UCSCTL2 = FLLD_1 + 66;                  // Set DCO Multiplier for 4MHz
                                        // (N + 1) * FLLRef = Fdco
                                        // (66 + 1) * 32768 = 4MHz
                __bic_SR_register(SCG0);            // Enable the FLL control loop
        }
        else if (counter == 3)
        {

                __bis_SR_register(SCG0);            // Disable the FLL control loop
                UCSCTL1 = DCORSEL_4;                    // Select DCO range 16MHz
operation

                UCSCTL2 = FLLD_1 + 31;                  // Set DCO Multiplier for 1MHz
                                        // (N + 1) * FLLRef = Fdco
                                        // (31 + 1) * 32768 = 1MHz
                __bic_SR_register(SCG0);            // Enable the FLL control loop
        }
        }
        else
        {

        P1OUT ^= REDLED; // toggle red LED
        P4OUT ^= GREENLED; // toggle green LED
        }
        }
```

```
}
```