Embedded Systems Lab

CPE 325-02

Clocks and Interrupts on the MSP430

By: David Thornton

Lab Date: September 29, 2020

Lab Due: October 12, 2020

Demonstration Due: October 12, 2020

Introduction

This lab introduces the concept of clocks and interrupts with the MSP430.

Theory

Topic 1: Interrupts and Interrupt Vectors

- “Interrupt is a signal emitted by hardware or software when a process or an event needs immediate attention. It alerts the processor to a high priority process requiring interruption of the current working process. In I/O devices one of the bus control lines is dedicated for this purpose and is called the Interrupt Service Routine (ISR).” - Source

Topic 2: Clock Module in MSP430

- There are 5 clock sources on the MSP430, XT1CLK, VLOCLK, REFOCLK, DCOCLK, and XT2CLK. By changing these values, we can change the frequency of peripheral devices such as how quickly LEDs blink.

Lab Assignment

1. Write an assembly program that interfaces switches, SW1 and SW2, and LEDs, LED1 and LED2, as follows (You should use interrupts for both switches).
    a. Initially, both the LEDs should be turned off.
    b. When SW2 is pressed for the first time, LED2 should be turned on. The next time SW2 is pressed, LED2 is turned off, and so on. Hence each press changes the state of LED2.
    c. When SW1 is pressed, LED1 blinks 3 times at 1Hz and then toggles the state of LED2.
    d. What happens when SW2 is pressed while LED1 is blinking? Does that disrupt the blinking? Does SW2 function correctly? Explain.

2. Write a C program that interfaces switches SW1 and SW2, LEDs 1 and 2, and the clock frequency as follows (You should use interrupts for both switches):
    a. Initially, LED1 is on and LED2 is off and the clock frequency is set to 1MHz.
    b. Both LEDs blink with a 1,000,000-loop iteration delay.
    c. Every time SW1 is pressed, the blinking frequency is increased by doubling the clock frequency and the clock frequency does not exceed 8MHz.

d. Every time SW2 is pressed, the blinking frequency is decreased by halving the clock frequency and the clock frequency does not go below 1MHz.
e. Calculate the LEDs blinking rate for each clock frequency and show your work.

Observations

All code satisfies the requirements.

1d. What happens when SW2 is pressed while LED1 is blinking? Does that disrupt the blinking? Does SW2 function correctly? Explain.

- When SW2 is pressed while LED1 is blinking, it will cause LED1 to blink six times, not three as well as only toggle LED2 once. This is because while the SW1 interrupt is executing, it is disturbed by the SW2 interrupt.

2e. Calculate the LEDs blinking rate for each clock frequency and show your work.

- My work can be found in Appendix 2, specifically lines 119 - 165.

Conclusion

This lab was successful in introducing me to clocks and interrupts.
Demo link

Appendix

Appendix 1: Lab_6_Q1.asm

```
;-------------------------------------------------------------------------------
; File:             Lab_6_Q1.asm
; Description:  S2 toggles GREEN_LED, S1 also toggles GREEN_LED, but after
;                                   blinking RED_LED 3 times at 1Hz.
; Input:            S1 and S2 on MSP-EXP430F5529LP
; Output:                   Blinking LEDs 1 and 2
; Author:                   David Thornton
; Lab Section:  2
; Date:                     October 12, 2020
;-------------------------------------------------------------------------------
                            .cdecls C,LIST,"msp430.h"               ; Include device header file
;-------------------------------------------------------------------------------
                            .def    RESET                   ; Export program entry-point to
                                                                                                    ; make it
known to linker.
```

```
                        .def    S1_ISR
                        .def    S2_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 stack pointer
StopWDT:    mov.w   #WDTPW|WDTHOLD,&WDTCTL      ; Stop watchdog timer
;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
Setup:
                        bis.b   #0x01, &P1DIR                   ; Set P1.0 as output
(RED_LED)
                        bis.b   #0x80, &P4DIR                   ; Set P4.7 as output
(GREEN_LED)
                        bic.b   #0x01, &P1OUT                   ; Turn RED_LED off at
start
                        bic.b   #0x80, &P4OUT                   ; Turn GREEN_LED off at
start

                        ; S2 input and output
                        bic.b   #0x02, &P2DIR                   ; Set P2.1 as input for S1
                        bis.b   #0x02, &P2REN                   ; Enable pull-up resistor at
P2.1
                        bis.b   #0x02, &P2OUT                   ; Required for proper IO
set up

                        ; S1 input and output
                        bic.b   #0x02, &P1DIR                   ; Set P1.1 as input for S2
                        bis.b   #0x02, &P1REN                   ; Enable pull-up resistor at
P1.1
                        bis.b   #0x02, &P1OUT                   ; Required for proper IO
set up

                        bis.w   #GIE,  SR                       ; Enable global interrupts
                        bis.b   #0x02, &P1IE                ; Enable port 1 interrupt from bit 1
                        bis.b   #0x02, &P1IES                   ; Set interrupt for high to
low
                        bic.b   #0x02, &P1IFG                   ; Clear interrupt flag
                        bis.b   #0x02, &P2IE                ; Enable port 2 interrupt from bit 1
                        bis.b   #0x02, &P2IES                   ; Set interrupt for high to
```

```
low
                        bic.b   #0x02, &P2IFG                    ; Clear interrupt flag

Start:          cmp             #1, R5                          ; Check if R5 = 1
                        jne             Red_Press                       ; If R5 != 1, jump to
Red_Press
                        clr             R5                                      ; Clear R5
                        xor.b   #0x01, &P1OUT                   ; Toggle P1.0 (RED_LED)

Red_Press:      cmp             #1, R6                          ; Check if R6 = 1
                        jne     Inf_Loop                        ; If R6 != 1, jump to
Inf_Loop
                        clr             R6                                      ; Clear R6
                        mov             #6, R5                          ; R5 <- 6

Cycle:          mov             #0xFFFF, R7                     ; R7 <- 0xFFFF

Delay:          dec             R7                                      ; Decrement R7
                        nop
                        nop
                        nop
                        nop
                        nop
                        jnz             Delay                           ; If R7 != 0, jump to
Delay
                        xor.b   #0x01, &P1OUT                   ; Toggle P1.0 (RED_LED)
                        dec             R5                                      ; Decrement
R5
                        jnz     Cycle                           ; If R5 != 0, jump to Cycle
                        bit.b   #0x01, &P1OUT                   ; P1 AND 1
                        xor.b   #0x80, &P4OUT                   ; Toggle P4.7
(GREEN_LED)
                        jz              Inf_Loop                        ; If P4.7 changes,
jump to Inf_Loop

Inf_Loop:       jmp             Start                           ; Loop until interrupt occurs


;-------------------------------------------------------------------------
; P1_0 (RED_LED) / P2_1 (S1) ISR
;-------------------------------------------------------------------------
S1_ISR:
                        bic.b   #0x02, &P2IFG                   ; Clear interrupt flag
                        bit.b   #0x02, &P2IN                ; Check if S1 is pressed
                        jnz             Red_Exit                        ; If SW is not
pressed, jump to exit
                        mov.b   #2000, R7                       ; Set to (2000 * 10 cc =
```

```
20,000 cc)

RedDelay:   dec     R7                              ; Decrement R7
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            jnz     RedDelay                        ; Is R7 = 0? (Delay over?)
            bit.b   #0x02, &P2IN                ; Verify S1 is still pressed
            jnz         Red_Exit                        ; If not, wait for S1
press
            mov.b   #1, R6                          ; R6 <- 1

Red_Exit:   reti                                    ; Return from interrupt


;-------------------------------------------------------------------------
; P4_7 (GREEN_LED) / P1_1 (S2) ISR
;-------------------------------------------------------------------------
S2_ISR:
            bic.b   #0x02, &P1IFG                   ; Clear interrupt flag
            bit.b   #0x02, &P1IN                ; Check if S2 is pressed
            jnz         Green_Exit                      ; If SW is not
pressed, jump to exit
            xor.b   #0x80, &P4OUT                   ; Toggle P4.7
(GREEN_LED)
            mov.b   #2000, R7                       ; Set to (2000 * 10 cc =
20,000 cc)

GreenDelay: dec     R7                              ; Decrement R7
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            jnz     GreenDelay                      ; Is R7 = 0? (Delay over?)
            bit.b   #0x02, &P1IN                ; Verify S2 is still pressed
            jnz         Green_Exit                      ; If not, wait for S2
press
            mov.b   #1, R7                          ; R7 <- 1

Green_Exit: 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  S2_ISR
                        .sect   ".int42"                ; PORT1_VECTOR
                        .short  S1_ISR
                        .end
```

Appendix 2: Lab_6_Q2.c

```
/*-------------------------------------------------------------------------
* File:             Lab_6_Q2.c
* Description:  This program varies the clock frequency depending on S1 and S2.
* Input:            S1 and S2 on MSP-EXP430F5529LP
* Output:                   Blinking LEDs 1 and 2 at various frequencies
* Author:                   David Thornton
* Lab Section: 2
* Date:                     October 12, 2020
* *-------------------------------------------------------------------------*/
#include  <msp430.h>

#define S1 ((P2IN & BIT1) == 0)             // Red LED switch
#define S2 ((P1IN & BIT1) == 0)             // Green LED switch
#define REDLED 0x01                                     // LED1 - Mask for BIT0
(0000_0001b)
#define GREENLED 0x80                           // LED2 - Mask for BIT7 (1000_0000b)

void configure_clock_sources();             // Function prototype: Configure clock
inline void _1Mhz();                        // Function prototype: Change CF to 1 Mhz
inline void _2Mhz();                        // Function prototype: Change CF to 2 Mhz
inline void _4Mhz();                        // Function prototype: Change CF to 4 Mhz
inline void _8Mhz();                        // Function prototype: Change CF to 8 Mhz

int status = 1;                                 // Status to check blink frequency
```

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

        P1DIR &= ~BIT1;                             // Set P1.1 as input (S2)
        P1REN |= BIT1;                              // Enable pull-up resistor
        P1OUT |= BIT1;                              // Turn P1 output on

        P2DIR &= ~BIT1;                             // Set P2.1 as input (S1)
        P2REN |= BIT1;                              // Enable pull-up resistor
        P2OUT |= BIT1;                              // Turn P2 output on

        _EINT();                                    // Enable interrupts
        P1IE  |= BIT1;                          // Enable interrupt at P1.1 for S1
        P1IES |= BIT1;                              // Enable hi->lo edge for interrupt
        P1IFG &= ~BIT1;                             // Clear any errornous interrupt
flag

        P2IE  |= BIT1;                          // Enable interrupt at P2.1 for S2
        P2IES |= BIT1;                              // Enable hi->lo edge for interrupt
        P2IFG &= ~BIT1;                             // Clear any errornous interrupt
flag

        configure_clock_sources();          // Configure the clock sources
        _1Mhz();                                    // Set initial blinking to 1 Mhz

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

        P1OUT = P1OUT | REDLED;                 // Turn on REDLED
        P4OUT = P4OUT & ~GREENLED;          // Turn off GREENLED

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


#pragma vector = PORT1_VECTOR
__interrupt void PORT1_ISR(void)    // ISR to handle S2 press (decrease CF)
{
        P1IFG &= ~BIT1;                         // Clear the interrupt flag
        __delay_cycles(25000);
```

```
        if(S2 != 0)                                     // If S2 is not pressed
                return;                                 // Return

        if(status == 8)                         // Are LEDs blinking at 8 Mhz?
        {
                _4Mhz();                                // Decrease to 4Mhz
                status = 4;                             // Adjust status
        }
        else if(status == 4)                    // Are LEDs blinking at 4 Mhz?
        {
                _2Mhz();                                // Decrease to 2Mhz
                status = 2;                             // Adjust status
        }
        else if(status == 2)                    // Are LEDs blinking at 2 Mhz?
        {
                _1Mhz();                                // Decrease to 1Mhz
                status = 1;                             // Adjust status
        }
}

#pragma vector = PORT2_VECTOR
__interrupt void PORT2_ISR(void)    // ISR to handle S1 press (increase CF)
{
        P2IFG &= ~BIT1;                         // Clear the interrupt flag
        __delay_cycles(25000);

        if(S1 != 0)                                     // If S1 is not pressed
                return;                                 // Return

        if(status == 4)                         // Are LEDs blinking at 4 Mhz?
        {
                _8Mhz();                                // Increase to 8Mhz
                status = 8;                             // Adjust status
        }
        else if(status == 2)                    // Are LEDs blinking at 2 Mhz?
        {
                _4Mhz();                                // Increase to 4Mhz
                status = 4;                             // Adjust status
        }
        else if(status == 1)                    // Are LEDs blinking at 1 Mhz?
        {
                _2Mhz();                                // Increase to 2Mhz
                status = 2;                             // Adjust status
        }
}
```

```
void _1Mhz()                                    // Change the clock frequency to 1 Mhz
{
        __bis_SR_register(SCG0);            // Disable the FLL control loop
        UCSCTL1 = DCORSEL_3;                    // Select DCO range _1Mhz operation
        UCSCTL2 = 32;                                   // Set DCO Multiplier for 1Mhz
                                                        // (N + 1) * FLLRef = Fdco
                                                        // (32 + 1) * 32768 = 1Mhz
        __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
}


void _2Mhz()                                    // Change the clock frequency to 2 Mhz
{
        __bis_SR_register(SCG0);            // Disable the FLL control loop
        UCSCTL1 = DCORSEL_4;                    // Select DCO range 2Mhz operation,
        UCSCTL2 = 62;                                   // Set DCO Multiplier for 2Mhz
                                                        // (N + 1) * FLLRef = Fdco
                                                        // (62 + 1) * 32768 = 2Mhz
        __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
}


void _4Mhz()                                    // Change the clock frequency to 4 Mhz
{
        __bis_SR_register(SCG0);            // Disable the FLL control loop
        UCSCTL1 = DCORSEL_4;                    // Select DCO range 4 Mhz operation
        UCSCTL2 = 124;                                  // Set DCO Multiplier for 4 Mhz
                                                        // (N + 1) * FLLRef = Fdco
                                                        // (124 + 1) * 32768 = 4
Mhz
        __bic_SR_register(SCG0);            // Enable the FLL control loop
        __delay_cycles(125000);                 // 32 x 32 x 4 MHz / 32,768 Hz
                                                        // 125000 = MCLK cycles
for DCO to settle
}


void _8Mhz()                                    // Change the clock frequency to 8 Mhz
{
        __bis_SR_register(SCG0);            // Disable the FLL control loop
```

```
        UCSCTL1 = DCORSEL_5;                    // Select DCO range 8 Mhz operation
        UCSCTL2 = 249;                                  // Set DCO Multiplier for 8 Mhz
                                                        // (N + 1) * FLLRef = Fdco
                                                        // (249 + 1) * 32768 = 8
Mhz
        __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, and DCO stabilizes.
        // In this case only DCO has to stabilize.
        do
        {
                UCSCTL7 &= ~(XT2OFFG + XT1LFOFFG + DCOFFG);     // Clear XT2, XT1,
DCO flags
                SFRIFG1 &= ~OFIFG;                                                      //
Clear fault flags
        }
        while (SFRIFG1 & OFIFG);                // Test oscillator fault flag
}
```