CPE 325: Intro to Embedded Computer System

Lab06

Interrupts and Universal Clock Subsytem

Submitted by: Gianna Foti

Date of Experiment:______29 September 2024_______________

Report Deadline:_______2 October 2024________________

Lab Section: ______CPE353-02______

Demonstration Deadline: _____2 October 2024___________

Introduction

In this lab, we had two main tasks to complete. The first task required us to write an Assembly program to interface two switches (SW1 and SW2) and two LEDs (LED1 and LED2) using interrupts. Initially, both LEDs were turned on, and the state of LED2 would toggle with each press of SW1. Additionally, pressing SW2 would cause LED1 to blink four times at 1 Hz and then toggle the state of LED2. The second task involved modifying a C program that toggles both LEDs. We were required to update the program so that each time SW2 is pressed, the clock frequency is set to 8 MHz. Similarly, pressing SW1 would reduce the clock frequency by half, ensuring that it does not drop below 1 MHz. This lab focused on utilizing interrupts and controlling clock frequencies, which are critical components in embedded systems programming.

Theory Topics

1. Interrupts:
    a. Interrupts are signals that temporarily halt the execution of the main program to allow the microcontroller to respond to specific events, such as pressing a button or receiving data from a sensor. When an interrupt occurs, the processor stops its current task, saves the state of the program, and jumps to a special function called an Interrupt Service Routine (ISR) to handle the event. Once the ISR finishes, the processor resumes the main program where it left off. Interrupts are essential for real-time applications as they allow immediate attention to important events without constantly polling for them in the main program. They can be used for handling tasks like button presses, timers, communication events, and sensor data handling in embedded systems.
2. Interrupt Vectors:
    a. Interrupt vectors are specific memory addresses that store the locations of ISR functions. Each interrupt source (e.g., a specific button press or a timer overflow) is assigned a

unique interrupt vector. When an interrupt occurs, the microcontroller looks at the corresponding interrupt vector to determine which ISR to execute. These vectors provide the link between an interrupt source and its handling code, allowing the microcontroller to manage multiple types of interrupts. Interrupt vectors are used in microcontrollers to handle hardware and software events without needing to constantly check the state of peripherals. For example, in the MSP430, each peripheral that can trigger an interrupt has a dedicated vector that leads to its ISR.

3. Clock Module in MSP430:
    a. The Clock Module in the MSP430 is a subsystem that generates clock signals used by the CPU and peripherals to time their operations. The MSP430 has multiple clock sources, including the DCO (Digitally Controlled Oscillator) and external crystal oscillators. The clock module allows you to configure these sources to achieve different operating frequencies, enabling you to control power consumption and performance. You can change clock frequencies by selecting different clock sources (such as DCO for internal clocks or XT1/XT2 for external crystals) and adjusting clock dividers in the UCS (Unified Clock System) or BCS (Basic Clock System). In programs, functions can be written to modify the clock frequency dynamically, as demonstrated in our lab where pressing switches allowed us to change the clock from 8 MHz down to 1 MHz, optimizing power use based on system needs.

Results & Observation

Part 1:

Program Description:

This program interfaces with two switches and two LEDs. Initially, both LEDs are turned on. When SW1 is pressed, it toggles the state of LED2 (green) on and off. When SW2 is pressed, it makes LED1 (red) blink 4 times at 1 Hz, then toggles LED2. The program uses interrupts to detect switch presses and includes debouncing to ensure stable input handling.

Process:

1. The first thing that I did was initialize the LEDs and switches.
    a. Set up the appropriate pins for LED1 (P1.0) and LED2 (P4.7) as outputs, and ensure both LEDs are initially turned on.
    b. Configure the pins for SW1 (P2.1) and SW2 (P1.1) as inputs, and enable pull-up resistors.
2. Next, I configured interrupts.
    a. Enable interrupts for SW1 and SW2 to detect when the switches are pressed.
    b. Set the interrupt edge to trigger on a high-to-low transition (indicating a button press).
    c. Clear any existing interrupt flags to avoid false triggers.
3. Next, I implemented debugging.
    a. In each switch's Interrupt Service Routine (ISR), implement a debounce delay to ensure stable switch presses.
    b. After the delay, check the switch state again to confirm it is still pressed before proceeding.
4. Lastly, I toggled LEDs based on the switch press
    a. In SW1's ISR, toggle LED2 (green) each time the switch is pressed.
    b. In SW2's ISR, toggle LED1 (red) four times with a delay for blinking, and toggle LED2 once after the blink sequence.

Part 2:

Program 2 Description:

This program controls the clock frequency and LEDs of the MSP430 microcontroller based on user input through two switches, SW1 and SW2. Initially, the program sets up the LEDs and switches, then adjusts the clock speed and LED behavior dynamically. The main functionality is to ensure that the clock frequency is either set to 8 MHz or reduced by half down to a minimum of 1 MHz, depending on which switch is pressed.

Process:

1. First I did my Initialization:
   a. Stop the watchdog timer.
   b. Configure the LEDs and switches (set directions and enable pull-up resistors).
   c. Set up interrupts for the switches.
2. Next, I implemented the main loop (mainymain):
   a. Enter an infinite loop where the program checks for switch presses.
   b. If neither switch is pressed, toggle both LEDs with a delay.
3. Next, I did switch interrupt handling:
   a. For SW1 (P2.1): Reduce the clock frequency by half each time it is pressed, ensuring it doesn't go below 1 MHz.
   b. For SW2 (P1.1): Set the clock frequency to 8 MHz when pressed.

Calculations:

1. (N + 1) * FLLRef = Fdco
   a. DOC Multiplier = (8,388,608 / 32,768) - 1 = 225 ← 8 MHz
   b. DOC Multiplier = (1,048,576 / 32,768) - 1 = 31 ← 1 MHz

Flowchart:

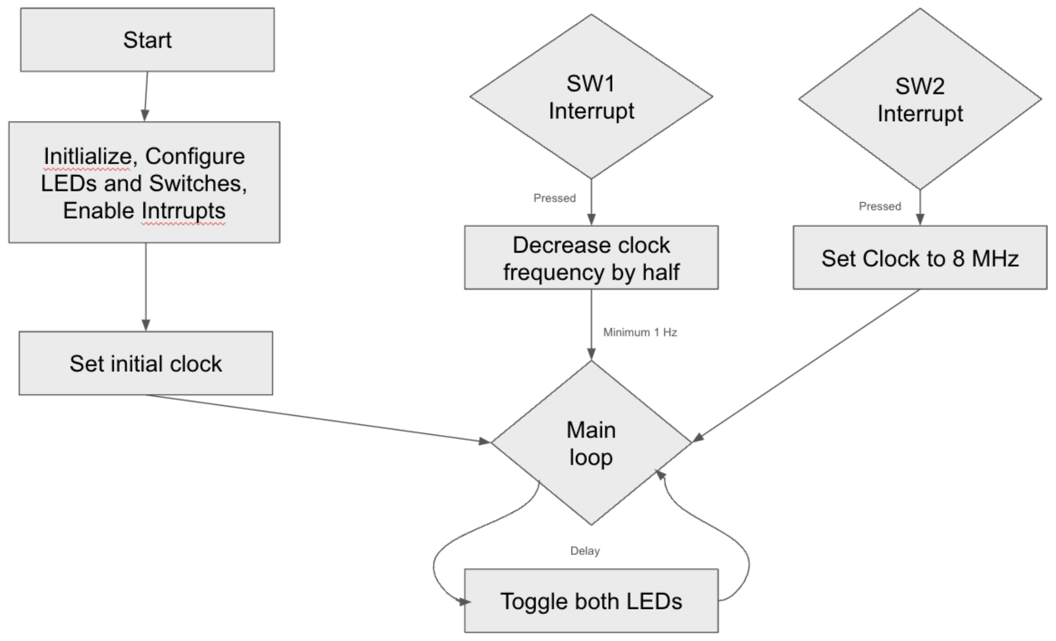

Figure 1: Program 2 Flowchart

Conclusion

In conclusion, this laboratory exercise introduced us to the fundamentals of interfacing switches and LEDs with the MSP430 using Code Composer Studio. In the first program, I learned how to control two LEDs—turning both on initially, then toggling LED2 when SW1 is pressed, and making LED1 blink four times at 1 Hz when SW2 is pressed before toggling LED2. This helped reinforce my understanding of interrupts and debouncing techniques for stable input handling. In the second program, I explored clock frequency adjustments, where pressing SW1 reduced the clock speed by half (down to a minimum of 1 MHz), and pressing SW2 set the clock speed back to 8 MHz, all while toggling the LEDs if neither switch was pressed. Writing these programs enhanced my skills in clock management and hardware interfacing with the MSP430. Overall, this lab provided valuable insights into how to configure inputs, outputs, and clocks for real-time embedded systems.

Appendix

Table 1: Program 1 source code

```
; ---------------------------------------------------------------------------------------------------------------
; file:        Lab06_P1.asm
; description: assembly program interfaces w/ switches 1 & 2, and leds 1 & 2 aka red and green
;              1. turn on both leds
;              2. toggle led2 when sw1 is pressed
;              3. when sw2 is pressed, toggle led1 4x & toggle led2 1x at 1 Hz
; input:       p2.1 - sw1 | p1.1 - sw2
; output:      p1.0 - red led | p4.7 - green led
; author:      gianna foti
; ---------------------------------------------------------------------------------------------------------------
             .cdecls C, LIST, "msp430.h"          ; include device header file
; ---------------------------------------------------------------------------------------------------------------
             .def   RESET                         ; export program entry-point to make it known to linker
             .def   SW2_ISR                       ; define sw2 interrupt
             .def   SW1_ISR                       ; define sw1 interrupt
; ---------------------------------------------------------------------------------------------------------------
             .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 da stack pointer
             mov.w  #WDTPW|WDTHOLD, &WDTCTL                ; stoppy stop da watchdog timer
; ---------------------------------------------------------------------------------------------------------------
; all the boring stuff, inititialization or whatever
MainyMain:
             bis.b  #BIT0, &P1DIR                 ; set p1.0 as output (red LED)
             bis.b  #BIT0, &P1OUT                 ; turn on led1 (p1.0)
             bis.b  #BIT7, &P4DIR                 ; set p4.7 as output (green LED)
             bis.b  #BIT7, &P4OUT                 ; turn on led2 (p4.7)
             bic.b  #BIT1, &P1DIR                 ; set p1.1 as input for sw2
             bis.b  #BIT1, &P1REN                 ; enable pull-up resistor at p1.1
             bis.b  #BIT1, &P1OUT                 ; configure pull-up
             bic.b  #BIT1, &P2DIR                 ; set p2.1 as input for sw1
             bis.b  #BIT1, &P2REN                 ; enable pull-up resistor at p2.1
             bis.b  #BIT1, &P2OUT                 ; configure pull-up
             bis.w  #GIE, SR                      ; enable global interrupts
             bis.b  #BIT1, &P1IE                  ; enable interrupt for p1.1 (sw2)
             bis.b  #BIT1, &P1IES                 ; set interrupt to trigger on high-to-low transition
             bic.b  #BIT1, &P1IFG                 ; clear interrupt flag
             bis.b  #BIT1, &P2IE                  ; enable interrupt for p2.1 (sw1)
             bis.b  #BIT1, &P2IES                 ; set interrupt to trigger on high-to-low transition
             bic.b  #BIT1, &P2IFG                 ; clear interrupt flag
InfinityInfiniteLoop:
             jmp    $                             ; infinite loop, to infinity and beyondddd
; ---------------------------------------------------------------------------------------------------------------
; SW2 Interrupt Service Routine (P1.1)
; Handles toggling of LED1 (red) and LED2 (green) when switch 2 is pressed.
; ---------------------------------------------------------------------------------------------------------------
SW2_ISR:
             bic.b  #BIT1, &P1IFG                 ; clear interrupt flag, bye bye
CheckyCheckSW2:
             bit.b  #BIT1, &P1IN                  ; check if sw2 is still pressed
             jnz    EndyEndSW2ISR                ; exit ISR if sw2 is not pressed
DebounceyDebounceSW2:
             mov.w  #2000, R15                    ; debounce delay count
DebouncyLoopSW2:
```

```
            dec.w   R15
            nop
            nop
            nop
            nop
            jnz     DebouncyLoopSW2          ; wait for debounce period
            bit.b   #BIT1, &P1IN             ; check if sw2 is still pressed
            jnz     EndyEndSW2ISR           ; exit ISR if sw2 is not pressed
            mov.w   #8, R4                   ; initialize counter for 8 toggles (4 blinks)
ToggleyToggleLED4:
            xor.b   #BIT0, &P1OUT            ; toggle LED1 (red)
            mov.w   #50000, R15              ; delay count for ~500 ms
Delay_500ms_SW2:
            dec.w   R15
            nop
            nop
            nop
            nop
            jnz     Delay_500ms_SW2          ; wait for 500 ms
            dec.w   R4                       ; decrement toggle counter
            jnz     ToggleyToggleLED4       ; repeat toggling 4 times
            xor.b   #BIT7, &P4OUT            ; toggle LED2 (green)
WaityWaitSW2:
            bit.b   #BIT1, &P1IN             ; check if sw2 is released
            jz      WaityWaitSW2             ; wait until sw2 is released
EndyEndSW2ISR:
            reti                             ; return from interrupt
; ------------------------------------------------------------------------------------------------------------------
; SW1 Interrupt Service Routine (P2.1)
; Handles toggling of LED2 (green) when switch 1 is pressed.
; ------------------------------------------------------------------------------------------------------------------
SW1_ISR:
            bic.b   #BIT1, &P2IFG            ; clear interrupt flag
CheckyCheckSW1:
            bit.b   #BIT1, &P2IN             ; check if sw1 is still pressed
            jnz     EndyEndSW1ISR           ; exit ISR if sw1 is not pressed
DebouncyBounceSW1:
            mov.w   #2000, R15               ; debounce da delay count
DebouncyLoopySW1:
            dec.w   R15
            nop
            nop
            nop
            nop
            jnz     DebouncyLoopySW1         ; wait for da debounce period
            bit.b   #BIT1, &P2IN             ; check if sw1 is still pressed
            jnz     EndyEndSW1ISR            ; exit ISR if sw1 is not pressed
ToggleyToggleLED2:
            xor.b   #BIT7, &P4OUT            ; toggle LED2 (green)
WaityReleaseSW1:
            bit.b   #BIT1, &P2IN             ; check if sw1 is released
            jz      WaityReleaseSW1          ; wait until sw1 is released
EndyEndSW1ISR:
            reti                             ; return from interrupt
; ------------------------------------------------------------------------------------------------------------------
; Stack Pointer Definition
; ------------------------------------------------------------------------------------------------------------------
            .global __STACK_END
            .sect   .stack
; ------------------------------------------------------------------------------------------------------------------
; Interrupt Vectors
; -------------------------------------------------------------------------------------------------------------------
            .sect   ".reset"                    ; MSP430 reset vector
            .short  RESET
            .sect   ".int47"                    ; Port1 vector
            .short  SW2_ISR                     ; ISR for SW2
            .sect   ".int42"                    ; Port2 vector
            .short  SW1_ISR                     ; ISR for SW1
            .end
```

Table 2: Program 2 source code

```
/*
; -------------------------------------------------------------------------------------------------------------------
; file:       Lab06_P2.c
; description: c program interfaces with switches 1 & 2, and leds 1 & 2 (red and green).
;             1. both leds are initially off.
;             2. when sw1 is pressed, decrease clock frequency by half (minimum 1 mhz).
;             3. when sw2 is pressed, set clock frequency to 8 mhz.
;             4. if neither switch is pressed, toggle both leds.
; input:      p2.1 - sw1 | p1.1 - sw2
; output:     p1.0 - red led | p4.7 - green led
; author:     gianna foti
;--------------------------------------------------------------------------------------------------------------------
*/
#include <msp430.h>
#define SWITCH1 (!(P2IN & BIT1))            // sw1 is pressed when p2.1 is 0
#define SWITCH2 (!(P1IN & BIT1))            // sw2 is pressed when p1.1 is 0
#define REDLED BIT0                         // red led connected to p1.0
#define GREENLED BIT7                       // green led connected to p4.7
void ClockyClock();                         // function prototype
void ClockyClock8MHz();                     // function prototype
int ClockyClock1MHz(int);                   // function prototype
volatile int clockDivider = 32;             // global variable for clock division
void main(void)
{
   WDTCTL = WDTPW + WDTHOLD;                // stop watchdog timer
   // configure leds
   P1DIR |= BIT0;                    // set led1 (red led) as output
   P1OUT = 0x00;                     // clear led1 status (turn off)
   P4DIR |= BIT7;                    // set led2 (green led) as output
   P4OUT = 0x80;                     // set led2 status to 0x80 (turn off)
   // configure switches
   P1DIR &= ~BIT1;                   // set p1.1 as input for sw2
   P1REN |= BIT1;                    // enable pull-up resistor at p1.1
   P1OUT |= BIT1;                    // set pull-up for sw2 to 1
   P2DIR &= ~BIT1;                   // set p2.1 as input for sw1
   P2REN |= BIT1;                    // enable pull-up resistor at p2.1
   P2OUT |= BIT1;                    // set pull-up for sw1 to 1
   // configure interrupts for switches in C style
   P1IE |= BIT1;                     // enable interrupt for p1.1 (sw2)
   P1IES |= BIT1;                    // set interrupt to trigger on high-to-low transition
   P1IFG &= ~BIT1;                   // clear interrupt flag for p1.1
   P2IE |= BIT1;                     // enable interrupt for p2.1 (sw1)
   P2IES |= BIT1;                    // set interrupt to trigger on high-to-low transition
   P2IFG &= ~BIT1;                   // clear interrupt flag for p2.1
   __bis_SR_register(GIE);           // enable global interrupts
   ClockyClock();  // set initial clock
   while (1)
   {
      __delay_cycles(500000); // delay for toggling leds
      // if neither switch is pressed, toggle both leds
      P1OUT ^= REDLED;     // toggle red led
      P4OUT ^= GREENLED;   // toggle green led
   }
}
void ClockyClock()
{
   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 stabilize
   do
   {
      UCSCTL7 &= ~(XT2OFFG + XT1LFOFFG + DCOFFG); // clear xt2, xt1, dco fault flags
      SFRIFG1 &= ~OFIFG;      // clear oscillator fault flag
   }
   while (SFRIFG1 & OFIFG);  // test oscillator fault flag
}
void ClockyClock8MHz()
{
   __bis_SR_register(SCG0);    // disable the fll control loop
   UCSCTL1 = DCORSEL_5;        // select dco range for 8mhz operation
   UCSCTL2 = 249;              // set dco multiplier for 8mhz (249 + 1) * 32768 = 8mhz
   __bic_SR_register(SCG0);    // enable the fll control loop
   __delay_cycles(250000);     // delay for dco to settle
}
int ClockyClock1MHz(int i)
{
   __bis_SR_register(SCG0);    // disable the fll control loop
   UCSCTL1 = DCORSEL_3;        // select dco range for 1mhz operation
   if (i >= 32)
   {
      UCSCTL2 = i / 2;        // set dco multiplier for the current clock
   }
   __bic_SR_register(SCG0);    // enable the fll control loop
   __delay_cycles(33792);      // delay for dco to settle
   return (i >= 32) ? i / 2 : i;  // ensure i does not go below the threshold
}
// interrupt service routine for port 1 (sw2 pressed)
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
   ClockyClock8MHz();  // set clock to 8 mhz
   clockDivider = 250; // reset clock divider for further frequency changes
   P1IFG &= ~BIT1;     // clear interrupt flag for p1.1
}
// interrupt service routine for port 2 (sw1 pressed)
#pragma vector=PORT2_VECTOR
__interrupt void Port_2(void)
{
   clockDivider = ClockyClock1MHz(clockDivider);  // decrease clock frequency by half
   P2IFG &= ~BIT1;     // clear interrupt flag for p2.1
}
```