1. CPE 325: Laboratory Assignment #5 Digital Input/Output Ports, Experimenter Board Hardware, and Hardware Debugging

Objectives: This tutorial will help you get started with the MSP30 IAR Assembly program development on the TI's MSP430FG4618/F2013 Experimenter Board. You will learn the following topics:

- Hardware development platform MSP430FG4618/F2013 Experimenter Board
- Assembly programming
- Creating an application project using assembly programs
- Program downloading on a real platform (MSP430FG4618/F2013 Experimenter Board).
- Software delays
- Interfacing buttons

Note: It is required that students have completed the tutorial *Getting Starting With MSP430 IAR Embedded Workbench* before starting with this one.

1.1. Digital Input/Output Introduction

A microcontroller interacts in many ways with the system in which it is embedded. It may receive inputs from a human through switches, buttons, sensors, etc. In opposite direction, the microcontroller may control external devices such as, light-emitting diodes (LEDs), seven-segment displays, liquid-crystal displays (LCDs), or some actuators (e.g., motors). The MSP430 microcontrollers can drive these external devices directly if they work from the same voltage and draw a sufficiently small current. Heavier loads require dedicated circuitry to drive them. The most straightforward form of input/output is through the digital input/output ports using binary values (0 or 1). The primary way how the MSP430 interfaces the rest of the world is through parallel ports. The actual number of physical parallel ports varies with device type and can range from two to ten 8-bit parallel ports, typically marked P1-P10. A parallel port encompasses multiple registers, including:

- PxIN – input register, reading it returns the logical values on the pins (determined by the external signals);
- PxOUT – output register, writing it sends the value to the pins;
- PxDIR – direction register, configures pins as inputs or outputs (e.g., P2DIR.BIT1=0 configures bit 1 of port P2 as an input pin; P2DIR.BIT2=1 configures bit 2 of port P2 as an output pin).
- PxSEL – selection register, this allows the user to change the register from the standard digital I/O to a special function. On the hardware diagram, when multiple symbols are seen on a pin, this will select between those functions. The default setting is the port I/O.
- Some ports include additional registers that support interrupt capability.

Our development platform includes several LEDs (LED1-LED4) and switches (SW1 and SW2). The LEDs can be turned on and off by writing digital 1 or 0 to the appropriate output port registers. Therefore, in order to turn a LED on, first the I/O port should be set to output direction, and then either a 0 or a 1 should be written to the output register. In this lab, four LEDs are turned on and off with specific frequencies. Let us develop a program to blink a LED.

1.2. Blink a LED Project Using Assembly Language

This section defines the problem that will be solved by the "Blink a LED" application. Your task is to write an assembly program that will repeatedly blink the LED1 on the TI's MSP430FG4618/F2013 Experimenter Board every second, i.e., the LED1 will be on and off for about 0.5 seconds each.

Step 1: Analyze the assignment.
In order to better understand the assignment we will first study schematics of the MSP430FG4618/F2013 Experimenter Board. This board includes TI's MSP430 microcontrollers (MSP430FG4618 and MSP430F2013), capacitive touch pad, serial RS232 port, 4 leds (LED1-LED4), 2 switches (SW1 and SW2), a microphone, a buzzer and several extension slots that allow an easy access to all microcontroller ports. Detailed schematic of the board is provided in the following document (which can also be accessed on ANGEL):
http://www.ece.uah.edu/~milenka/msp430/TI-MSP430FG4618-EXPB/TI-MSP430FG4618-EXPB_schematic.pdf.
Open the schematic file. In the schematic see how the LED1 and LED2 ports are actually connected to physical LEDs (a diode through a resistor) (Figure 1). What microcontroller port pins are connected to LED1 and LED2? A LED is on when the current is flowing through it and it is off when there is no current. How should we drive the corresponding ports to have the current flow?

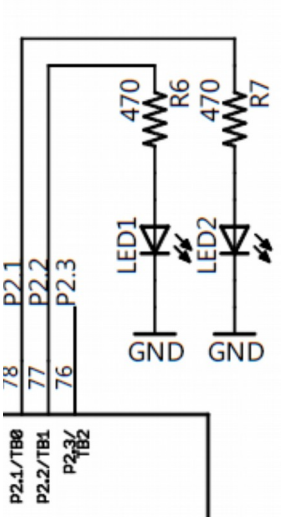

Figure 1. LED1 and LED2 connections on the TI's experimenter's board.

Step 2. Develop a plan.

From the schematic we see that if we want LED1 on, we should provide a logical '1' at the output port of the microcontroller (port P2.2), and a logical '0,' if we want LED1 to be off. We could take several approaches to solving this problem. The simplest one is to toggle the port P2.2 and have 0.5 seconds delay in software.
After initializing the microcontroller, our program will spend all its time in an infinite loop (LED1 and LED2 should be repeatedly blinking alternatively). Inside a loop we will toggle the ports P2.1 and P2.2 and then wait for approximately 0.5 seconds. The port toggling can be done using an XOR operation of the current value of the port (P2OUT) and the constant 0x06, i.e., (P2OUT=P2OUT xor 0x06). Software delay of 0.5 seconds can be implemented using an empty loop with a certain number of iterations.

1.2.1. Assembly Code

Figure 1 shows the assembly code of the blink application (lab3_D1.s43). Here is a short description of the assembly code for this application.
1. The comments in a single line start with a column character (;). Multi-line comments can use C-style /* comment */ notation.

2. `#include <msp430xG46x.h>;` This is a C-style pre-processor directive that specifies a header file to be included in the source. The header file includes all macro definitions, for example, special function register addresses (`WDTCTL`), and control bits (`WDTPW+WDTHOLD`).

3. RSEG is a segment control assembler directive that controls how code and data are located in memory. `RSEG` is used to mark the beginning of a relocatable code or data segment. `CODE` and `DATA` are recognized segment types that are resolved by the linker. The IAR XLINK linker can recognize any other type of segment (e.g., `CSTACK` for code stack).

4. How does my program execute on the MSP430. Upon powering-up the MSP430 control logic always generates a `RESET` interrupt request (it is the highest priority interrupt request). The value stored at the address 0xFFFE (the last word in the 64KB address space) is reserved to keep the starting address of the reset handler (interrupt service routine), and the first thing that the microcontroller does is to fetch the content from this address and put it in the program counter (`PC`). Thus, the starting address of our program should be stored at location 0xFFFE. The IAR linker is responsible for this step. Please notice that the symbolic name `RESET` (the starting address of our program) is used to load the reset vector in the interrupt vector table.

5. First instruction initializes the stack pointer register.

6. The move instruction at the StopWDT label, sets certain control bits of the watchdog timer to disable it. The watchdog timer by default will be active generating interrupt requests periodically. As this functionality is not needed in our program, we simply need to disable it.

7. Parallel ports in MSP430 microcontroller can be configured as input or output. A control register `PxDIR` determines whether the port x is input or output (we can configure each individual port pin). Our program drives the port pins P2.1 and P2.2, so they should be

configured as the output. The second and third bits of the `P2DIR` are set to 1, while other bits are unchanged (0 by default).

8. The main loop starts at Mainloop label. The xor instruction performs toggling operation as explained above.

9. The code starting at label Wait implements software delay. To exactly calculate the software delay we need to know instruction execution time and the clock cycle time. The register R15 is loaded with 50,000. The dec.w instruction takes 1 clock cycle to execute, and jnz L1 takes 2 clock cycles to execute (note: this can be determined by using the Simulator and the Register View CYCLECOUNTER field). The total time per iteration is 3 clock cycles.
Determining clock cycle time requires in-depth understanding of the FLL-Clock module of the MSP430 which is beyond the scope of this tutorial. We note that the processor clock frequency is approximately 1MHz for our configuration. The total delay is thus 50,000*3*1μμs=150,000μμs. Obviously, to meet our requirements we need to increase the number of instructions in the software delay loop (to be 10). We can do it by adding nop instructions.

10. The final program code that satisfies the requirements is shown in Figure 1 (file lab3_D2.s43).

```
;*******************************************************************************
; TI Experimenter board demo, blinking leds LED1 and LED2 (msp430FG4618)
;
; Description: Toggle P2.1 and P2.2 by xoring P2.1 and P2.2 inside a loop.
;              The leds are connected to P2.1 and P2.2 and are on when
;              P2.1=1 and P2.2=1;
;              The LEDs are initialized P2.1 to be off, and P2.2 to be on;
;              Clocks: ACLK = 32.768kHz, MCLK = SMCLK = default DCO (~1 MHz)
;
;                  MSP430xG461x
;               -----------------
;           /|\|                 |
;            | |                 |
;            --|RST              |
;              |             P2.2|-->LED1(GREEN)
;              |             P2.1|-->LED2(YELLOW)
;
; Alex Milenkovich, milenkovic@computer.org
;*******************************************************************************
#include  <msp430xG46x.h>
;-------------------------------------------------------------------------------
            RSEG    CSTACK                  ; Define stack segment
;-------------------------------------------------------------------------------
            RSEG    CODE                    ; Assemble to Flash memory
;-------------------------------------------------------------------------------
RESET       mov.w   #SFE(CSTACK),SP         ; Initialize stack pointer
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT
SetupP2     bis.b   #006h,&P2DIR            ; Set P2.1&P2.2 to output
                                            ; direction (0000_0110)
            bis.b   #002h,&P2OUT            ; Set P2OUT to 0x0000_0010
                                            ;(LED2 is on, LED1 is off)
MainLoop    xor.b   #006h,&P2OUT            ; Toggle P2.1&P2.2 using XOR
Wait        mov.w   #050000,R15             ; Delay to R15
```

```
34. L1          dec.w   R15                     ; Decrement R15
35.             jnz     L1                      ; Delay over?
36.             jmp     MainLoop                ; Again
37.                                             ;
38. ;-------------------------------------------------------------------------
39.             COMMON  INTVEC                  ; Interrupt Vectors
40. ;-------------------------------------------------------------------------
41.             ORG     RESET_VECTOR            ; MSP430 RESET Vector
42.             DW      RESET                   ;
43.             END
44. ;*************************************************************************
```

Figure 1. MSP430 Assembly Code forblinking LED1 and LED2 alternatively on the TI's MSP430FG4618 Experimenters board.

```
1   ;*************************************************************************
2   ;   TI Experimenter board demo, blinking leds LED1 and LED2 (msp430FG4618)
3   ;
4   ;   Description: Toggle P2.1 and P2.2 by xoring P2.1 and P2.2 inside a loop.
5   ;                The leds are connected to P2.1 and P2.2 and are on when
6   ;                P2.1=1 and P2.2=1;
7   ;                The LEDs are initialized P2.1 to be off, and P2.2 to be on;
8   ;                ACLK = 32.768kHz, MCLK = SMCLK = default DCO
9   ;
10  ;                 MSP430xG461x
11  ;             -----------------
12  ;         /|\|                 |
13  ;          | |                 |
14  ;          --|RST              |
15  ;            |             P2.2|-->LED1(GREEN)
16  ;            |             P2.1|-->LED2(YELLOW)
17  ;
18  ;   Alex Milenkovich, milenkovic@computer.org
19  ;*************************************************************************
20  #include  <msp430xG46x.h>
21  ;-------------------------------------------------------------------------
22              RSEG    CSTACK                  ; Define stack segment
23  ;-------------------------------------------------------------------------
24              RSEG    CODE                    ; Assemble to Flash memory
25  ;-------------------------------------------------------------------------
26  RESET       mov.w   #SFE(CSTACK),SP         ; Initialize stack pointer
27  StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT
28  SetupP2     bis.b   #006h,&P2DIR            ; Set P2.1&P2.2 to output
29                                              ; direction (0000_0110)
30              bis.b   #002h,&P2OUT            ; Set P2OUT to 0x0000_0010 (LED2
31                                              ; is on, LED1 is off)
32  MainLoop    xor.b   #006h,&P2OUT            ; Toggle P2.1 & P2.2 using exclusive-OR
33  Wait        mov.w   #050000,R15             ; Delay to R15
34  L1          dec.w   R15                     ; Decrement R15
35              nop
36              nop
37              nop
38              nop
39              nop
40              nop
41              nop
42              jnz     L1                      ; Delay over?
43              jmp     MainLoop                ; Again
44                                              ;
45  ;-------------------------------------------------------------------------
```

```
46              COMMON  INTVEC                  ; Interrupt Vectors
47  ;-------------------------------------------------------------------------------
48              ORG     RESET_VECTOR            ; MSP430 RESET Vector
49              DW      RESET                   ;
50              END
51  ;*******************************************************************************
```

Figure 1. MSP430 Assembly Code for blinking LED1 and LED2 alternatively every 1s on MSP430FG4618 Experimenters board.

1.2.2. Creating an Application Project

Please consult the previous tutorials on how to create a new workspace and a project.

Step 1:- Choose Project>Create New Project. Select Empty project and save it as Lab3_D2 (or Lab3_D1).

Step 2:- Add the source file Lab3_D2.s43 to the project. Choose Project>Add Files.

Step 3:- Choose Project>Options and set all categories as explained in the Getting Starting With MSP430 IAR Embedded Workbench tutorial.

Step 4:- Select the source file Lab3_D2.s43 and compile it. (Choose Project>Compile). Click on the list file Lab3_D2.lst to see a report generated by the compiler.

Step 5:- Setting the linker options. In Project>Options, select Linker category. Check the box for "Override default program entry" and select "Defined by application." This step notifies the linker where to find the beginning of the program. See Figure 1.

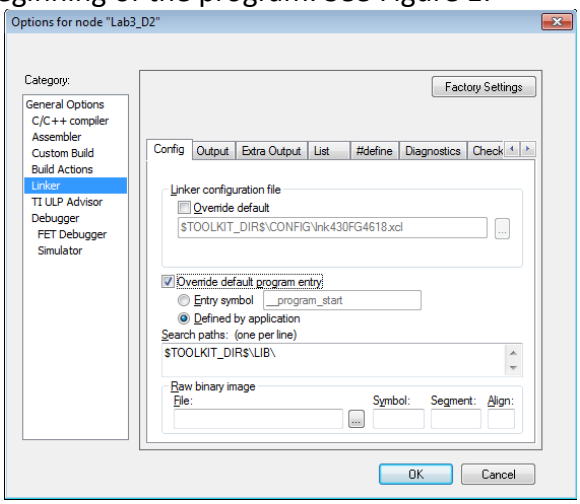

Figure 1. Linker category.

Step 6:- Link the program. Choose Project>Make. The application is now ready for debugging and for download to the TI experimenter's board. Click on blink_asm.map to see report from the linker.

1.2.3. Program Simulation and Debugging

In this section we will discuss how to simulate program execution. ~~will simulate program~~

~~execution.~~

Step 1: Go to Project>Options and select Debugger category. In the Driver box, choose Simulator (Figure 1). Click OK.

Step 2: Choose Project>Debug. Ignore the warning. You will see the EW display as shown in Figure 1.

Step 3: Step through the program. Observe the Disassembly, Register View, and Memory View windows. Answer the following questions.
What is the starting address of the program?
How many clock cycles does each instruction take to execute?
Observe the contents of memory location and registers as you step through the program. What is the content of the memory location at the address 0xFFFE? What are addresses of the special-purpose registers P2DIR and P2OUT? Monitor the contents of these locations as you walk through your program. Set breakpoints to move easier through your program.

Step 4: Choose Stop Debugging.

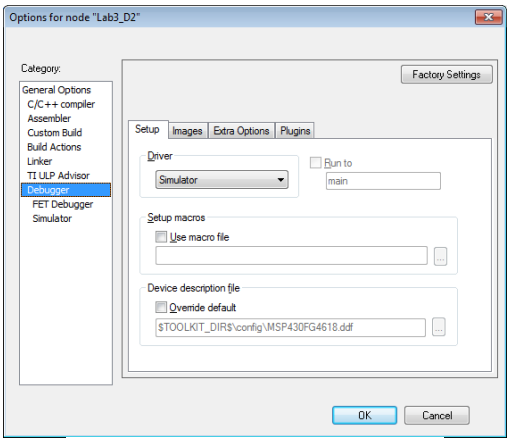

Figure 1. Debugger category.

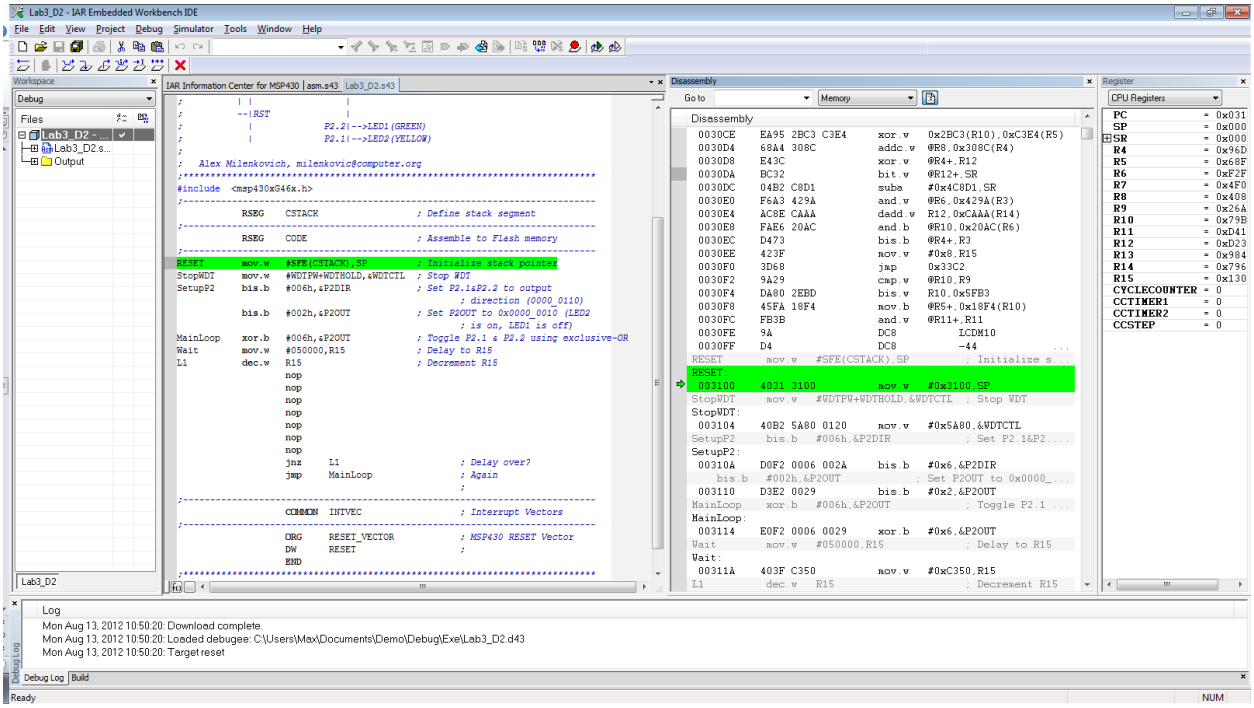

Figure 1. IAR Simultor.

1.2.4. Program Download

Step 1: Choose Project>Options, Debugger category. Select FET Debugger (Figure 1) and click OK.

Step 2:- Start debugger. The executable is downloaded to the board through JTAG programmer.

Step 3:- Choose Debug>Go. Observe the LED1 on the board.

Step 4:- Stop the debugger. Experiment on your own.

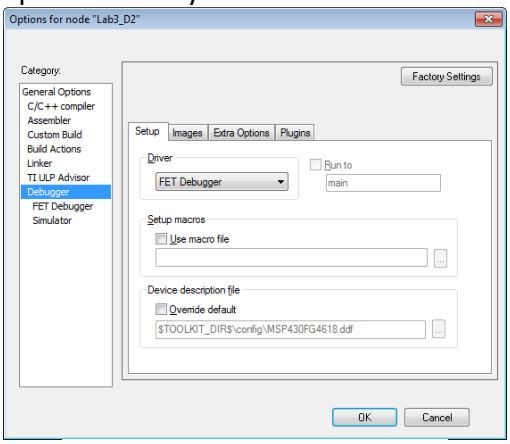

Figure 1. Debugger category.

1.3. Interfacing Buttons (Switches)

Often we would like to trigger a certain task in embedded systems by pressing a button or switch. Here we will learn how to interface a button, how to detect that it is pressed and released. First, let us take a look at the TI Experimenter's board development platform schematic (Figure 8) that illustrates how the buttons are connected to the MSP430. We can see two switches SW1 and SW2. The ports SW1 and SW2 that are connected to the MSP430's port one pins P1.0-P1.1. When the switches are not pressed the inputs SW1 and SW2 are at logic 1 level (VCC) and when they are pressed the inputs will be at logic 0 (GND). This may seem counterintuitive, and it should reinforce the habit of becoming familiar with your hardware datasheet before programming.

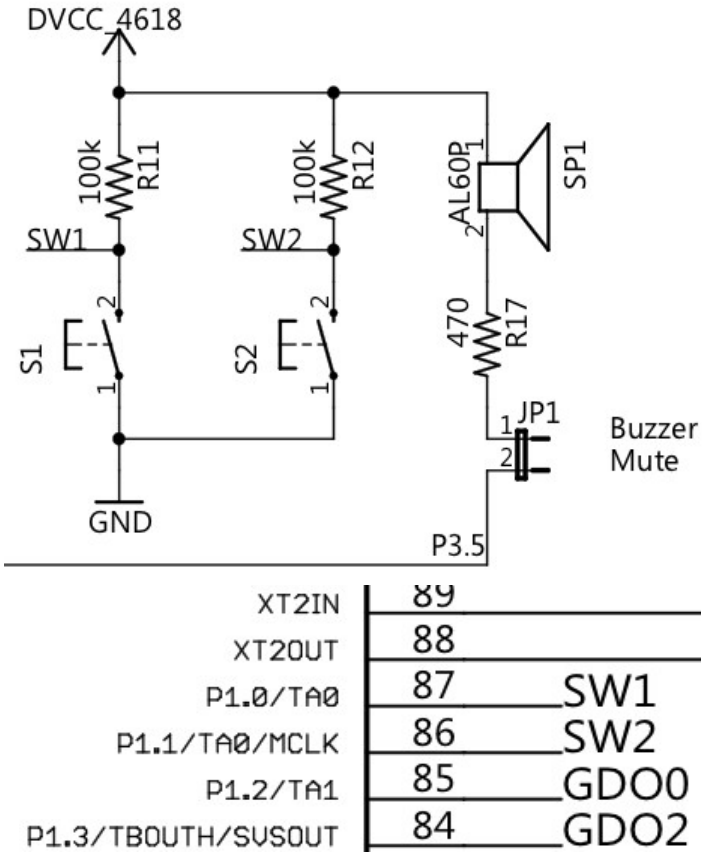

Figure 2. SW1 and SW2 specifics from the hardware schematic

In interfacing buttons we must take care that we properly detect whether a button is pressed. Typically, we need (1) to detect that a button has been pressed, (2) to debounce it (apply software delay to ensure that is indeed pressed, rather than a faulty detection caused by noise), and (3) to detect that it has been released. An example of the noise created by the action of a switch pressing can be seen in figure 9. If not programmed robustly, the hardware could think that each spike was an individual press of the button, which could be detrimental to the functionality of the hardware.

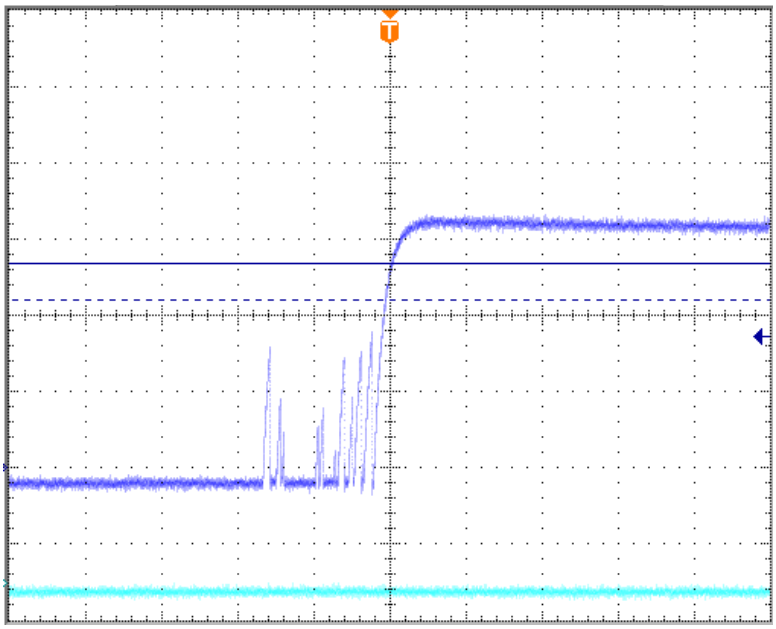

Figure 3. Oscilloscope showing noise from switch depressing action

We can individually examine the P1IN bit 0 status to see if a SW1 has been depressed. If it has indeed been pressed (bit 0 of P1IN = 0), we perform a loop to wait for a short period of time (~20 ms) to avoid faulty detections that may be caused by electrical noise. After the software delay, we validate that the input port pin is still at 0; if yes, that means that the switch is indeed pressed. We may keep the button pressed for a longer period of time, and we often want to ensure that a button is released before we go to process an event that may be triggered by this detection.

```
ChkSw1      bit.b #00000001b,&P1IN        ; Check if SW1 is pressed
            jnz   NotPress                ; If not zero, SW is not pressed
Wait        mov.w #02000,R15              ; Set to (2000 * 10 ms delay)
L1          dec.w R15                     ; Decrement R15
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            jnz   L1                      ; Delay over?
            bit.b #00000001b,&P1IN        ; verify SW1 is still pressed
            jz    Sw1Pressed              ; SW1 is pressed, go to module
NotPress    nop

SW1Pressed  nop
```

Figure 9 shows a program that turns the LED1 when the switch SW1 is pressed and it is kept on as long as the SW1 is pressed. When the button is released, the LED1 is turned off.

```
;-------------------------------------------------------------------------------
;   TI Experimenter board demo, activate LED1 with SW1 (msp430FG4618)
```

```
;
;   Description: The LED1 is initialized off.  If SW1 is pressed
;                then a delay loop waits 20 ms.  The switch is checked
;                again, and if on, the LED1 is turned on until the
;                switch is released.
;                 ACLK = 32.768kHz, MCLK = SMCLK = default DCO
;
;                  MSP430xG461x
;              -----------------
;          /|\|                 |
;           | |                 |
;           --|RST              |
;             |             P2.2|-->LED1(GREEN)
;             |             P1.0|<--SW1
;
;   Micah Harvey
;*******************************************************************************
#include  <msp430xG46x.h>
;-------------------------------------------------------------------------------
        RSEG    CSTACK                  ; Define stack segment
;-------------------------------------------------------------------------------
        RSEG    CODE                    ; Assemble to Flash memory
;-------------------------------------------------------------------------------
RESET:  mov.w   #SFE(CSTACK),SP         ; Initialize stack pointer
StopWDT:
        mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT
SetupP2:
        bis.b   #004h,&P2DIR            ; Set P2.2 to output
                                        ; direction (0000_0100)
        bic.b   #004h,&P2OUT          ; Set P2OUT to 0x0000_0100 (ensure
                                        ; LED1 is off)
ChkSw1:     bit.b #01h,&P1IN          ; Check if SW1 is pressed
                                        ; (0000_0001 on P1IN)
      jnz   ChkSw1            ; If not zero, SW is not pressed
                                        ; loop and check again
Debounce:
      mov.w #02000,R15        ; Set to (2000 * 10 ms delay)
L1:   dec.w R15               ; Decrement R15
      nop
      nop
      nop
      nop
      nop
      nop
      nop
      jnz   L1                ; Delay over?
      bit.b #00000001b,&P1IN  ; verify SW1 is still pressed
      jnz   ChkSw1            ; if not, wait for SW1 press

LEDon:
      bis.b   #004h,&P2OUT            ; turn on LED1
SW1wait:
      bit.b   #001h,&P1IN             ; test SW1
      jz      SW1wait                 ; wait until SW1 is released
      bic.b   #004,&P2OUT             ; turn off LED1
      jmp     ChkSw1                  ; loop to beginning
```

```
;-------------------------------------------------------------------------------
        COMMON  INTVEC                  ; Interrupt Vectors
;-------------------------------------------------------------------------------
        ORG     RESET_VECTOR            ; MSP430 RESET Vector
        DW      RESET                   ;
        END
;*******************************************************************************
```

Figure 9. Assembly program that activates LED1 when SW1 is pressed

With microcontrollers, it is often useful to be able to use interrupts in your program. An interrupt allows an automatic break from the current instruction based on a set of conditions. Some of the I/O ports on the MSP430 have an interrupt that you can configure. When the interrupt condition is reached, an interrupt vector determines a set of code instructions that will be completed, and then the program will go where it left off using a RETI (return from interrupt) command. We will learn more about interrupts in a subsequent lab, but you should understand how interrupt vectors are used and what interrupts do.

To set up an interrupt for an I/O port, you must perform a few tasks:

- Enable global interrupts in the status register
- Enable interrupts to occur for the particular bits on the desired port
- Specify whether the interrupt is called on a falling edge or rising edge
- Initialize the interrupt flag by clearing it

```
;*******************************************************************************
; TI Experimenter board demo, activate LED1 with SW1 using ISR (msp430FG4618)
;
;   Description: The LED1 is initialized off.  Global interrupts are enabled.
;                An endless loop is entered, and if SW1 is pressed
;                then an ISR is called delay loop waits 20 ms.  The switch is
;                checked again, and if on, the LED1 is turned on until the
;                switch is released.  The interrupt is returned from.
;                 ACLK = 32.768kHz, MCLK = SMCLK = default DCO
;
;                  MSP430xG461x
;              -----------------
;          /|\|                 |
;           | |                 |
;           --|RST              |
;             |             P2.2|-->LED1(GREEN)
;             |             P1.0|<--SW1
;
;   Micah Harvey
;*******************************************************************************
#include  <msp430xG46x.h>
;-------------------------------------------------------------------------------
        RSEG    CSTACK                  ; Define stack segment
;-------------------------------------------------------------------------------
        RSEG    CODE                    ; Assemble to Flash memory
;-------------------------------------------------------------------------------
RESET:  mov.w   #SFE(CSTACK),SP         ; Initialize stack pointer
StopWDT:
        mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT
Setup:
```

```
        bis.b   #004h,&P2DIR            ; Set P2.2 to output
                                        ; direction (0000_0100)
        bic.b   #004h,&P2OUT          ; Set P2OUT to 0x0000_0100 (ensure
                                        ; LED1 is off)
        bis.w   #GIE,SR                 ; Enable global interrupts
        bis.b   #001h,&P1IE             ; Enable Port 1 interrupt from bit 0
        bis.b   #001h,&P1IES            ; Set interrupt for hi to low edge
        bic.b   #001h,&P1IFG            ; Clear interrupt flag
InfLoop:
        jmp     $                       ; Loop here until interrupt

;-------------------------------------------------------------------------------
; P1_0 (SW1) interrupt service routine (ISR)
;-------------------------------------------------------------------------------
SW1_ISR:
      bic.b   #001h,&P1IFG            ; Clear interrupt flag

Debounce:
      mov.w #02000,R15          ; Set to (2000 * 10 ms delay)
L1:   dec.w R15                 ; Decrement R15
      nop
      nop
      nop
      nop
      nop
      nop
      nop
      jnz   L1                  ; Delay over?
      bit.b #00000001b,&P1IN    ; verify SW1 is still pressed
      jnz   isr_ret             ; if not, wait for SW1 press

LEDon:
      bis.b   #004h,&P2OUT            ; turn on LED1
SW1wait:
      bit.b   #001h,&P1IN             ; test SW1
      jz      SW1wait                 ; wait until SW1 is released
      bic.b   #004,&P2OUT             ; turn off LED1
isr_ret:
      reti                            ; return from interrupt
;-------------------------------------------------------------------------------
      COMMON  INTVEC                  ; Interrupt Vectors
;-------------------------------------------------------------------------------
      ORG     RESET_VECTOR            ; MSP430 RESET Vector
      DW      RESET                   ;
      ORG     PORT1_VECTOR            ; Port1 interrupt vector
      DW      SW1_ISR                 ; go to SW1_ISR
      END
;*******************************************************************************
```

Figure 10. Assembly program activating LED1 while SW1 is pressed using interrupts

1.4. References

It is crucial that you become familiar with the basics of how digital ports work – how to set their output direction, read from or write to the ports, set interrupts, and set up their special functions. We will be using these features to control hardware and communication between

devices throughout this class. Please reference the following material to gain more insight on the device:

- The MSP430 Experimenter's Board hardware schematic
- Chapter 11 in the MSP430FG4618 user's guide (pages 407-414)
- Chapter 7 in the Davies text

1.5. Assignment

Write an assembly program that turns on LED1 when SW1 is pressed, and turns off LED1 when SW1 is pressed again. If SW2 is pressed, LED2 should blink at a frequency of10 Hz, and it should remain off if the switch is released. You must use an interrupt service routine (ISR) on port 1 to perform the functions when the appropriate switch is pressed.