Introduction

In this section we’ll take a look at the MSP430 architecture, instructions, and tools and give you a chance to get some hands-on time with the hardware and software with a lab using the MSP430F2013. We’ll also learn about the I/O and do another lab using the MSP430FG4618/9.

Objectives

- Overview
- TI Embedded Processor Portfolio
- Architecture
- Tools
- Introduction lab
- I/O
- I/O lab
MSP430 One Day Workshop 2009

Introduction ............................................................................................................................................... 1-1

Lab 1 – Flash the LED ............................................................................................................................. 1-17
  Hardware list: ........................................................................................................................................ 1-18
  Software list: ......................................................................................................................................... 1-18

IAR Kickstart Procedure ......................................................................................................................... 1-19
  Hardware Verification .......................................................................................................................... 1-19
  Power jumpers ..................................................................................................................................... 1-20
  IAR Kickstart ..................................................................................................................................... 1-21
  Configuring the Project ....................................................................................................................... 1-21
  Create and Add the Source File ......................................................................................................... 1-22
  Download and Run the Program ......................................................................................................... 1-23
  FLASH Programming Exercise .......................................................................................................... 1-24

CCS 4.0 Procedure .................................................................................................................................. 1-25
  Hardware Verification .......................................................................................................................... 1-25
  Power jumpers ..................................................................................................................................... 1-26
  LCL    FET .......................................................................................................................................... 1-26
  CCS 4.0 ................................................................................................................................................. 1-27
  Configuring the Target ....................................................................................................................... 1-27
  Understanding the IDE Display ........................................................................................................... 1-28
  Create and Add a Source File ............................................................................................................. 1-29
  Download and Run the Program ......................................................................................................... 1-30
  FLASH Programming Exercise .......................................................................................................... 1-31

Standard Definitions ............................................................................................................................... 1-33
  Controlling GPIO Ports ...................................................................................................................... 1-33

Lab 2 – I/O Overview .................................................................................................................................. 1-35
Hardware list: ....................................................................................................................................1-36
Software list: ....................................................................................................................................1-36

IAR Kickstart Procedure ....................................................................................................................1-37
  New Workspace and Project ............................................................................................................1-37
  Configure the Project .......................................................................................................................1-37
  Add Source File ..............................................................................................................................1-37
  Complete the Code ..........................................................................................................................1-38
  Test Your Code ..............................................................................................................................1-38

CCS 4.0 Procedure ............................................................................................................................1-43
  Add a Source File ............................................................................................................................1-44
  Complete the Code ..........................................................................................................................1-44
  Test Your Code ..............................................................................................................................1-46

Review Questions .............................................................................................................................1-48
MSP430 One Day Workshop 2009

TI Microcontroller Portfolio

TI Embedded Processing Portfolio

Microcontrollers

<table>
<thead>
<tr>
<th>16-bit</th>
<th>32-bit ARM</th>
<th>32-bit Real-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSP430</td>
<td>ARM</td>
<td>C2000</td>
</tr>
<tr>
<td>Ultra-low Power</td>
<td>Low Power</td>
<td></td>
</tr>
<tr>
<td>Up to 25 MHz</td>
<td>&lt;100 MHz</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ARM</th>
<th>ARM Cortex A8</th>
<th>ARM Cortex A-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Industry Std</td>
<td>Industry-Std</td>
<td>Industry-Std</td>
</tr>
<tr>
<td>Code</td>
<td>Code</td>
<td>Code</td>
</tr>
<tr>
<td>Fixed &amp; Floating Point</td>
<td>Fixed &amp; Floating Point</td>
<td>Fixed &amp; Floating Point</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Embedded Processors/DSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM+</td>
</tr>
<tr>
<td>ARM Cortex A-8</td>
</tr>
<tr>
<td>Accelerators</td>
</tr>
<tr>
<td>MMU</td>
</tr>
<tr>
<td>Cache</td>
</tr>
<tr>
<td>800 MMACS to 1.07 GMIPS/MHz</td>
</tr>
<tr>
<td>$0.49 to $9.00</td>
</tr>
</tbody>
</table>

Software, Tools & BSPs

MSP430 Generations

MSP430 Generations

<table>
<thead>
<tr>
<th>2xx</th>
<th>4xx</th>
<th>5xx</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Clock (Max)</td>
<td>16MHz</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash/RAM (Largest comparable device)</td>
<td>120KB / 4KB (F24xx)</td>
<td>120KB / 4KB (F246xx)</td>
</tr>
<tr>
<td>Active Current (3.0V)</td>
<td>515µA</td>
<td>600µA</td>
</tr>
<tr>
<td>1MHz</td>
<td>4.2mA</td>
<td>4.8mA</td>
</tr>
<tr>
<td>8MHz</td>
<td>9.1mA</td>
<td>N/A</td>
</tr>
<tr>
<td>16MHz</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>25MHz</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Standby Current (LPM3)</td>
<td>0.9 – 1.1µA</td>
<td>1.1 – 2.5µA</td>
</tr>
<tr>
<td>Power Down Current (LPM3)</td>
<td>0.1µA</td>
<td>0.1µA</td>
</tr>
<tr>
<td>Wake-up Time From LPM3</td>
<td>1µs</td>
<td>8µs</td>
</tr>
<tr>
<td>Flash ISP Minimum DVcc</td>
<td>2.2V</td>
<td>2.7V</td>
</tr>
<tr>
<td>Port I/O Interrupt Capability</td>
<td>P1/P2</td>
<td>P1/P2</td>
</tr>
<tr>
<td>Add1 pins in future devices</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prog. Port Pin Drive Strength</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Prog. Pull-ups/downs</td>
<td>All port pins</td>
<td>N/A</td>
</tr>
<tr>
<td>Available MCLK Sources</td>
<td>DCO, VLO, LFXT1, XT2</td>
<td>FLL, LFXT1, XT2</td>
</tr>
<tr>
<td>FLL Reference Clocks</td>
<td>N/A</td>
<td>LFXT1</td>
</tr>
</tbody>
</table>
Peripheral Overview

MSP430 Peripheral Overview

<table>
<thead>
<tr>
<th>1xx</th>
<th>2xx</th>
<th>4xx</th>
<th>5xx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Clock System</td>
<td>Basic Clock System</td>
<td>FLL, FLL+</td>
<td>Unified Clock System</td>
</tr>
<tr>
<td>Core voltage same as supply voltage (1.8-3.6V)</td>
<td>Core voltage same as supply voltage (1.8-3.6V)</td>
<td>Core voltage same as supply voltage (1.8-3.6V)</td>
<td>Programmable core voltage with integrated PMM (1.8-3.6V)</td>
</tr>
<tr>
<td>16-bit CPU</td>
<td>16-bit CPU, CPU X</td>
<td>16-bit CPU, CPU X</td>
<td>16-bit CPU Xv2</td>
</tr>
<tr>
<td>GPIO</td>
<td>GPIO w/ pull-up and pull-down</td>
<td>GPIO, LCD Controller</td>
<td>GPIO w/ pull-up and pull-down, drive strength</td>
</tr>
<tr>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>CRC16</td>
</tr>
<tr>
<td>Software RTC</td>
<td>Software RTC</td>
<td>Software RTC with Basic Timer, Basic Timer + RTC</td>
<td>True 32-bit RTC w/Alarms</td>
</tr>
<tr>
<td>USART</td>
<td>USART, USCI</td>
<td>USART, USCI</td>
<td>USART, USCI, RF</td>
</tr>
<tr>
<td>DMA up to 3-ch</td>
<td>DMA up to 3-ch</td>
<td>DMA up to 3-ch</td>
<td>DMA up to 8-ch</td>
</tr>
<tr>
<td>MPY16</td>
<td>MPY16</td>
<td>MPY16, MPY32</td>
<td>MPY32</td>
</tr>
<tr>
<td>ADC10, 12</td>
<td>ADC10, 12, SD16</td>
<td>ADC12, SD16, OPA</td>
<td>ADC12_A</td>
</tr>
<tr>
<td>4-wire JTAG</td>
<td>4-wire JTAG, 2-wire Spy Bi-Wire (Some devices)</td>
<td>4-wire JTAG</td>
<td>4-wire JTAG, 2-wire Spy Bi-Wire</td>
</tr>
</tbody>
</table>

MSP430F5xx Architectural Advances

**Ultra-Low Power**
- 160 µA/MIPS
- 2.5 µA standby mode
- Integrated LDO, BOR, WDT+, RTC
- 12 MHz @ 1.8V
- Wake up from standby in <5 µs

**Increased Performance**
- Up to 25 MHz
- 1.8V ISP Flash erase and write
- Fail-safe, flexible clocking system
- User-defined Bootstrap Loader
- Up to 1MB linear memory addressing

**Innovative Features**
- Multi-channel DMA supports data movement in standby mode
- Industry leading code density
- More design options including USB, RF, encryption, LCD interface

MSP430 Features ...
MSP430 Features

Ultra-low Power
- 0.1uA power down
- 0.8uA standby mode
- 250uA / 1MIPS
- <6us clock start-up (most parts <1us)
- <50nA port leakage
- Zero-power brown-out reset (BOR)

Ultra-Flexible
- 1k-128kB In-System Programmable (ISP) Flash
- 14-100 pin options
- UART, SPI, I2C, IrDA
- 10/12/16-bit ADC
- DAC, OP Amp, LCD driver
- Embedded emulation

16-bit RISC CPU

- Deep single-cycle register file
  - 4 special purpose
  - 12 general purpose
  - No accumulator bottleneck
- RISC architecture
  - 27 core instructions
  - 24 emulated instructions
  - 7 address modes
- Atomic memory-to-memory addressing
- Bit, byte and word processing
- Constant generator
### Bytes, Words and CPU Registers

#### 16-bit addition
- **Code/Cycles**
  - `add.w R4,R5` ; 1/1
  - `add.w &0200,&0202` ; 3/6

#### 8-bit addition
- **Code/Cycles**
  - `add.b R4,R5` ; 1/1
  - `add.b &0200,&0202` ; 3/6

- Use CPU registers for calculations and dedicated variables
- Same code size for word or byte
- Use word operations when possible

### Seven Addressing Modes

#### Register Mode
- **mov.w R10,R11**
  - Single cycle

#### Indexed Mode
- **mov.w 2(R5),6(R6)**
  - Table processing

#### Symbolic Mode
- **mov.w EDE,TONI**
  - Easy to read code, PC relative

#### Absolute Mode
- **mov.w &EDE,&TONI**
  - Directly access any memory

#### Indirect Register Mode
- **mov.w @R10,0(R11)**
  - Access memory with pointers

#### Indirect Autoincrement
- **mov.w @R10+,0(R11)**
  - Table processing

#### Immediate Mode
- **mov.w #45h,&TONI**
  - Unrestricted constant values

Atomic addressing ...
Atomic Addressing

Atomic Addressing

\[ B = B + A \]

- Non-interruptible memory-to-memory operations
- Useable with complete instruction set

```
; Pure RISC
push    R5
ld      R5, A
add     R5, B
st      B, R5
pop     R5

; MSP430
add     A, B
```

Constant Generator

Constant Generator

```
4314 mov.w #0002h,R4 ; With CG
4314 mov.w #0002h,R4 ; Without CG
40341234 mov.w #1234h,R4
```

- Immediate values -1,0,1,2,4,8 generated in hardware
- Reduces code size and cycles
- Completely automatic
Emulated Instructions

24 Emulated Instructions

- Easier to understand - no code size or speed penalty
- Replaced by assembler with core instructions
- Completely automatic

4130           ret                 ; Return (emulated)
4130           mov.w  @SP+,PC      ; Core instruction

Assembly Instruction Formats

Three Assembly Instruction Formats

<table>
<thead>
<tr>
<th>Format I</th>
<th>Source and Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>add.w R4, R5</td>
<td>R4+R5=R5 xxxx</td>
</tr>
<tr>
<td>add.b R4, R5</td>
<td>R4+R5=R5 00xx</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Format II</th>
<th>Destination Only</th>
</tr>
</thead>
<tbody>
<tr>
<td>rlc.w R4</td>
<td></td>
</tr>
<tr>
<td>rlc.b R4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Format III</th>
<th>8(Un)conditional Jumps</th>
</tr>
</thead>
<tbody>
<tr>
<td>jmp Loop_1</td>
<td>Goto Loop_1</td>
</tr>
</tbody>
</table>
Assembly Instructions List

51 Total Assembly Instructions

<table>
<thead>
<tr>
<th>Format I</th>
<th>Format II</th>
<th>Format III</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source, Destination</td>
<td>Single Operand</td>
<td>+/- 9bit Offset</td>
<td></td>
</tr>
<tr>
<td>add(.b)</td>
<td>br</td>
<td>jmp</td>
<td>clrc</td>
</tr>
<tr>
<td>adde(.b)</td>
<td>call</td>
<td>jcc</td>
<td>setc</td>
</tr>
<tr>
<td>and(.b)</td>
<td>sulpb</td>
<td>jnc</td>
<td>clrz</td>
</tr>
<tr>
<td>bic(.b)</td>
<td>std</td>
<td>jeg</td>
<td>setz</td>
</tr>
<tr>
<td>bis(.b)</td>
<td>push(.b)</td>
<td>jne</td>
<td>clrn</td>
</tr>
<tr>
<td>bit(.b)</td>
<td>pop(.b)</td>
<td>jge</td>
<td>setn</td>
</tr>
<tr>
<td>cmp(.b)</td>
<td>rra(.b)</td>
<td>jl</td>
<td>dint</td>
</tr>
<tr>
<td>dadd(.b)</td>
<td>rrrc(.b)</td>
<td>jn</td>
<td>eint</td>
</tr>
<tr>
<td>mov(.b)</td>
<td>inv(.b)</td>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>sub(.b)</td>
<td>inc(.b)</td>
<td>ret</td>
<td></td>
</tr>
<tr>
<td>subc(.b)</td>
<td>incd(.b)</td>
<td>reti</td>
<td></td>
</tr>
<tr>
<td>xor(.b)</td>
<td>dec(.b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>addc(.b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sbc(.b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>clr(.b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>dadc(.b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>rla(.b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>rlc(.b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>tst(.b)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bold type denotes emulated instructions

Unified Memory Map

- Absolutely no paging
- Supports code agility
- In System Programmable (ISP) Flash
  - Self programming
  - JTAG
  - Bootloader

// Flash In System Programming
FCTL3 = FWKEY; // Unlock
FCTL1 = FWKEY | WRT; // Enable
*(unsigned int *)0xFC00 = 0x1234;
Embedded Emulation

- Debug real time in system
  - No application resources used
  - Full speed
  - Breakpoint
  - Single step
  - Complex trigger
  - Trace
- Security Fuse

---

Flash Emulation Tool

USB FET Interface:

<table>
<thead>
<tr>
<th>JTAG Interface</th>
</tr>
</thead>
</table>

Target Board

Interface only without target board:

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Product Family</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSP-FET430UIF</td>
<td>MSP430</td>
<td>$99.00</td>
</tr>
</tbody>
</table>

- Third parties offer additional JTAG interfaces
- Many target boards available from TI and third parties
- Spy Bi-Wire (SBW) interface adapters available
**eZ430-F2013**

- Spy Bi-Wire Interface
- All target pins accessible
- USB Powered
- LED
- Removable Target Board
- Emulator

- The world’s smallest complete development tool for only $20
- All hardware and software included

**MSP430 Website**

**www.ti.com/msp430**

- User Guides
- Datasheets
- 100+ Application reports
- 1000+ Code examples
- Complete 3rd party listing
- IDE Links:
  - IAR Embedded workbench
  - TI Code Composer Essentials
  - Rowley, Quadra/ox, Image Craft, GCC, Others
  - Check Third-Party website for complete list
  - Most have 30-day trials
  - Check the Yahoo! user group for recommendations

**Code Examples**
Code Examples

- Reduce development time
- Over 1000 free examples in C & Assembly
- Use standalone or as a template for your next project

Reset Conditions

- RST/NMI configured in the reset mode
- All I/O pins are switched to input
- Watchdog timer powers up as active watchdog
- Other peripheral modules are disabled
- Status register (SR) is reset
- Program counter (PC) is loaded with (0FFFEh)
- Always refer to the user guide for information specific to your device
MSP430FG461x/F28xx Experimenter’s Board

Two MSP430 devices:
- MSP430FG4618 or MSP430FG4619
- MSP430F2013

Interface for ChipCon
RF transceiver EMK boards

IAR Kickstart IDE

- 4kB Compiler
- Assembler/Linker
- Editor
- Debugger

Instructor –
Lead a short tour of Kickstart IDE
Code Composer Studio 4.0

Code Composer Studio 4.0 (Free Version)

- 16kB Compiler
- Assembler/Linker
- Editor
- Debugger
- CCE’s successor

Let’s do a lab …
Lab 1 – Flash the LED

Let’s familiarize ourselves with the lab equipment and then move on to performing a simple task: flashing the LED using the F2013.

There are two sets of instructions for the labs; one using the IAR Kickstart IDE and the other using TI’s Code Composer Studio 4.0. Decide which IDE you’d like to use and then team up with a partner using the same IDE.
Hardware list:

- WinXP PC
- MSP-FET430UIF
- USB cable
- JTAG ribbon cable
- MSP430FG461x/F28xx Experimenter’s Board with batteries
- Digital Multimeter
- Jumpers
- Two AAA Batteries

Software list:

- IAR Kickstart for MSP430 version 4.11B
- Code Composer Studio 4.0
- Labs
- Additional pdf documentation
- Adobe™ Reader
IAR Kickstart Procedure

In this lab, you will verify that the hardware/software has been set up properly. We’ll also familiarize ourselves with the tools we’ll be using for the rest of the workshop via a short program running on the F2013.

Hardware Verification

1. **Check out the hardware**

Make sure that the MSP430 USB FET is connected to the USB cable and that the other end of the cable is connected to the PC’s USB port. The ribbon cable should be connected to the debug interface at one end to the port marked Target and to the lower of the two debug ports on the MSP430FG461x/F28xx Experimenter’s Board (the MSP430F2013 emulation port).

2. **Software driver**

We’ve previously loaded the software driver for the debug interface. In case your setup requires the driver, you can find it in the C:\Program Files\IAR Systems\Embedded Workbench 5.0\430\drivers\TIUSBFET folder.
Power jumpers

3. The board has several jumpers that control power to the board …

Make sure the jumpers are set as follows:

- **PWR1** controls power to the MSP430FG4619 (ON)
- **PWR2** controls power to the MSP430F2013 (ON)
- **JP2** isolates the LED from the touch pad (ON)
- **BATT** controls power from the AAA batteries and can be used to measure current (OFF)
- **VCC_1** and **VCC_2** control whether the microcontrollers are powered by the emulator (FET) or the batteries (LCL). Since we’ll be powering from the board from the emulator, place both jumpers over the rightmost two pins as shown:
IAR Kickstart

4. Start up the IDE

On the desktop of your PC you should see a shortcut that looks like:

Double-click the shortcut to start IAR Kickstart. The *Embedded Workbench Startup* window will appear on top of the IAR tool. Click the Cancel button on the lower right.

5. Create a New Workspace

Click File ⇒ New ⇒ Workspace on the menu bar to create a new workspace.

6. Create a New Project

On the menu bar, click Project ⇒ Create New Project. When the Create New Project dialogue appears, click OK. The Save As dialogue will appear; name your project Lab1 in the C:\MSP430ODW\IAR Labs\Lab1 folder and click Save.

Configuring the Project

7. Set the Project Options

From the IAR Embedded Workbench menu bar, select Project ⇒ Options.

Under the Target tab, note the Device selection box. Click the drop-menu to the right of this box and select *MSP430x2xx Family*, then *MSP430F2013* from the list.

Still under the Target tab, click Assembler-only project.

In the Category list to the left, click Debugger. Under the Setup tab, select FET Debugger from the Driver drop-down menu.

Select the Plugins tab, and uncheck the box next to Stack.

In the Category list to the left, click FET Debugger. Under the Setup tab, select Texas Instrument USB-IF from the Connection drop-down menu.

Click OK.
Create and Add the Source File

8. Create the Source File

From the IAR Embedded Workbench menu bar, select File ⇒ New ⇒ File. In the untitled editor window that appears, type the following code or you can cut/paste it from the Lab1.txt file included in the Lab1 folder.

To cut/paste, select File ⇒ Open ⇒ File from the menu bar. Change the Files of type: to Text Files (*.txt) and select Lab1.txt, then click Open. Cut/Paste to the Untitled1 file in your IAR editor.

```
#include "msp430x20x3.h"

ORG  0F800h       ; Program start
RESET mov.w  #280h,SP  ; Stack
         mov.w  #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog
         bis.b  #01h,&P1DIR

Mainloop xor.b #01h,&P1OUT
Delay dec.w  R15
         jnz  Delay
         jmp  Mainloop

ORG  0FFFEh       ; RESET vector
DW     RESET
END
```

On the menu bar, click the Save button , name the file Lab1.asm and place it in the C:\MSP430ODW\IAR Labs\Lab1 folder. Click the Save button.

9. Add the File to the Project

From the IAR Embedded Workbench menu bar, select Project ⇒ Add Files. You may need to change the Files of type to Assembler Files. Highlight Lab1.asm and click Open.
Download and Run the Program

10. Assemble and Download

Click the **Debug** button 📡. Clicking this button will assemble the source file in your project and download the executable to the flash memory of the MSP430. You may be prompted to save your workspace. Click **Yes**, name the workspace `Lab1.eww`, locate it in the `C:\MSP430ODW\IAR Labs\Lab1` folder and click **Save**.

A Message window will open at the bottom of the IAR tool and will inform you of the status of the build as it runs. Notice the download status as the code is transferred to the MSP430 flash memory. The IAR debugger may ask if you want to update the FET pod firmware; click **OK**.

11. Run the Program

You should be looking at a screen that looks something like this:

![Image of IAR Embedded Workbench IDE](image)

The buttons on the top-left that look like this: 🛡️ control the running of the code. Click on the **Go** button 🛡️ to run the code. You should notice that the red LED near the MSP430F2013 debug port is blinking about twice per second.

12. Stop Debugging and Close IAR Kickstart

Click the **Stop Debugging** button 🛋️.

From the IAR Embedded Workbench menu bar, click **File ➤ Exit**. If you are prompted to save anything, do so.
FLASH Programming Exercise

13. Exercise

In the F2xx family, the time to program any bit, byte or word in FLASH is $30/ft_{FG}$ – where FTG is between 257kHz – 476kHz. This means that the minimum programming time for any random bit, byte or word is 63us.

If FLASH memory is programmed sequentially though, the programming time can be reduced to $18/ft_{FG}$.

We’ve provided you with an excerpt from the F2013 datasheet below. Use it to fill in the blanks provided. Remember that 2KB is equal to 1KW, so it makes sense to program in words to reduce programming time.

Flash Memory

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TEST CONDITIONS</th>
<th>VCC</th>
<th>MIN</th>
<th>NOM</th>
<th>MAX</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC(PGM/ERASE)</td>
<td>Program and Erase supply voltage</td>
<td></td>
<td>2.2</td>
<td>3.6</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>ft_{FG}</td>
<td>Flash Timing Generator frequency</td>
<td>257</td>
<td>478</td>
<td>kHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I{PGM}</td>
<td>Supply current from VCC during program</td>
<td>2.7 V/3.6 V</td>
<td>3</td>
<td>5</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>I{ERASE}</td>
<td>Supply current from VCC during erase</td>
<td>2.7 V/3.6 V</td>
<td>3</td>
<td>7</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>t_CPT</td>
<td>Cumulative program time</td>
<td>see Note 1</td>
<td>2.7 V/3.6 V</td>
<td>4</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>t_CBCErase</td>
<td>Cumulative erase time</td>
<td>2.7 V/3.6 V</td>
<td>20</td>
<td>ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program/Erase endurance</td>
<td>10^4</td>
<td>10^5</td>
<td>cycles</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_Relation</td>
<td>Data retention duration</td>
<td>$t _1 = 25^\circ C$</td>
<td>100</td>
<td>years</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{Word}$</td>
<td>Word or byte program time</td>
<td></td>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{Block, 0}$</td>
<td>Block program time for 1st byte or word</td>
<td>see Note 2</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{Block, 1-53}$</td>
<td>Block program time for each additional byte or word</td>
<td></td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{Block, End}$</td>
<td>Block program end or sequence wait time</td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_MassErase</td>
<td>Mass erase time</td>
<td></td>
<td>1053</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_SegErase</td>
<td>Segment erase time</td>
<td></td>
<td>4019</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTES: 1. The cumulative program time must not be exceeded when writing to a 64-byte flash block. This parameter applies to all programming methods: individual word, byte, write and block write modes.
2. These values are hardwired into the Flash Controller's state machine ($t_{FTG} = 1/ft_{FG}$).

What is $f_{FTG}$? ____________ (pick the highest frequency/shortest period)

What is $t_{word}$? ____________

Calculate the time to program a word or byte ________________________________________

Multiply that by 1024 words ______________________________

We calculated that the time required to program the entire F2013 2KB Flash array as random words is 64.5ms.

IAR Kickstart users … you’re done. Proceed to page 1-33.
CCS 4.0 Procedure

In this lab, you will verify that the hardware/software has been set up properly. We’ll also familiarize ourselves with the tools we’ll be using for the rest of the workshop via a short program running on the MSP430F2013.

Hardware Verification

1. Check out the hardware

Make sure that the MSP430 USB FET is connected to the USB cable and that the other end of the cable is connected to the PC’s USB port. The ribbon cable should be connected to the debug interface at one end to the port marked Target and to the lower of the two debug ports on the MSP430FG461x/F28xx Experimenter’s Board (the MSP430F2013 emulation port).

2. Software driver

We’ve previously loaded the software driver for the debug interface. In case your setup requires the driver, you can find it in the: C:\Program Files\Texas Instruments\CCSv4\emulation\drivers\msp430\USB_FET folder.
3. The board has several jumpers that control power to the board …

Make sure the jumpers are set as follows:

**PWR1** controls power to the MSP430FG4619 (ON)

**PWR2** controls power to the MSP430F2013 (ON)

**JP2** isolates the LED from the touch pad (ON)

**BATT** controls power from the AAA batteries and can be used to measure current (OFF)

**VCC_1** and **VCC_2** control whether the microcontrollers are powered by the emulator (FET) or the batteries (LCL). Since we’ll be powering from the board from the emulator, place both jumpers over the rightmost two pins as shown:
CCS 4.0

4. Start up the IDE

On the desktop of your PC you should see a shortcut that looks like this:

Double-click the shortcut to start Code Composer Studio 4.0. The Workspace Launcher window will appear on top of the CCS tool. In the Workspace window, enter C:\MSP430ODW\CCS Labs\Lab1\workspace and click the OK button on the lower right. This will create a workspace folder in the Lab1 folder.

Ignore the CCS error log that appears; click the X on the right of the Error Log tab.

We won’t need the Welcome screen either, close that one as well.

5. Create a New Project

On the menu bar, click File ⇒ New ⇒ CCS Project. When the New Project dialogue appears, name the project Lab1 and click Next. Note that the location is our Lab1 workspace folder.

In the Select a type of project window, change the project type to MSP430 and click Next.

In the Additional Project Settings window, make no changes and click Next.

In the Project Settings window, change the Device Variant to MSP430F2013.

Check the box marked Treat as an Assembly-only project and click Finish.
Configuring the Target

6. Create a New Target Configuration

From the CCS menu bar, select Target ⇒ New Target Configuration …
Change the File name to Lab1.ccxml and click Finish.

When the Basic window appears, make the change as shown below:

![Lab1.ccxml tab](image)

Close the Lab1.ccxml tab. When prompted, click Yes to save the changes.

Understanding the IDE Display

7. Displayed Windows

CCS 4.0 is a highly customizable tool, but your first view of it should look like below:

![IDE Display](image)

The left hand pane is the Project pane. All of the components; libraries, source files, settings, etc that comprise a project are displayed here. The middle pane is the Workspace pane. When you are editing, the Eclipse editor will be seen here, along with tabs to the files being edited. The Outline pane, on the right displays C/C++ file elements, like structures, etc. Since this project is an assembly project, you can close this pane now by clicking the X in the Outline tab.
Create and Add a Source File

8. Create a Source File

Right-click in the Project pane and select New ⇒ Source File. When the New Source File window appears, name the Source File Lab1.asm and click Finish. In the Project pane you’ll see that Lab1.asm is now added to the project and that the file is open for editing in the Workspace pane.

In the Lab1.asm editor window that appears, type the following code or you can cut/paste it from the Lab1.txt file included in the Lab1 folder.

To cut/paste, select File ⇒ Open File … from the menu bar. Navigate to: C:\MSP430ODW\CCS Labs\Lab1, select Lab1.txt, and then click Open. Cut/Paste to the Lab1.asm editor window.

```asis
.cdecls C, LIST, "msp430x21x1.h" ; Device header file

.text
RESET
.mov.w #280h, SP
.mov.w #WDTPW+WDTHOLD, &WDTCTL
.bis.b #01h, &P1DIR

Mainloop
.xor.b #01h, &P1OUT

Delay
.dec.w R15
.jnz Delay
.jmp Mainloop

.sect ".reset" ; MSP430 RESET Vector
.short RESET

.end
```

On the menu bar, click the Save button .
Download and Run the Program

9. Assemble and Download

Click the **Debug Launch** button (not the Debug perspective button). Clicking this button will assemble the source file in your project and download the executable to the flash memory of the MSP430F2013.

A *Progress Information* window will open and inform you of the status of the assembly and download.

10. Run the Program

You should be looking at a screen that looks something like this:

![Image of CCS interface](image)

The buttons on the top-left that look like this: ![Control buttons](image) control the running of the code. Click on the **Run** button ![Run button](image) to run the code. You should notice that the red LED near the MSP430F2013 debug port is blinking about twice per second.

11. Halt Debugging and Close CCS

Click the **Terminate All** button ![Terminate All button](image) to halt the program, terminate the debugger session and return to the editor view. From the CCS menu bar, click **File ⇫ Exit**. If you are prompted to save anything, do so.
FLASH Programming Exercise

12. Exercise

In the F2xx family, the time to program any bit, byte or word in FLASH is \( 30/\text{f}_{\text{FTG}} \) – where FTG is between 257kHz – 476kHz. This means that the minimum programming time for any random bit, byte or word is 63us.

If FLASH memory is programmed sequentially though, the programming time can be reduced to \( 18/\text{f}_{\text{FTG}} \).

We’ve provided you with an excerpt from the F2013 datasheet below. Use it to fill in the blanks provided. Remember that 2KB is equal to 1KW, so it makes sense to program in words to reduce programming time.

Flash Memory

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TEST CONDITIONS</th>
<th>VCC</th>
<th>MIN</th>
<th>NOM</th>
<th>MAX</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC (PGM/ERASE)</td>
<td>Program and Erase supply voltage</td>
<td></td>
<td>2.2</td>
<td>3.6</td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>f_{FTG}</td>
<td>Flash Timing Generator frequency</td>
<td></td>
<td>257</td>
<td>476</td>
<td></td>
<td>kHz</td>
</tr>
<tr>
<td>I_{PGM}</td>
<td>Supply current from VCC during program</td>
<td>2.7</td>
<td>3.6</td>
<td>3</td>
<td>5</td>
<td>mA</td>
</tr>
<tr>
<td>I_{ERASE}</td>
<td>Supply current from VCC during erase</td>
<td>2.7</td>
<td>3.6</td>
<td>3</td>
<td>7</td>
<td>mA</td>
</tr>
<tr>
<td>T_{CPT}</td>
<td>Cumulative program time</td>
<td>see Note 1</td>
<td>2.7</td>
<td>3.6</td>
<td>4</td>
<td>ms</td>
</tr>
<tr>
<td>T_{CMerase}</td>
<td>Cumulative mass erase time</td>
<td>2.7</td>
<td>3.6</td>
<td>20</td>
<td></td>
<td>ms</td>
</tr>
<tr>
<td>t_{word}</td>
<td>Program/Erase endurance</td>
<td>10^4</td>
<td>10^5</td>
<td></td>
<td></td>
<td>cycles</td>
</tr>
<tr>
<td>T_{ retention}</td>
<td>Data retention duration</td>
<td>T_J = 25°C</td>
<td>100</td>
<td></td>
<td></td>
<td>years</td>
</tr>
<tr>
<td>t_{Word}</td>
<td>Word or byte program time</td>
<td></td>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{block, 0}</td>
<td>Block program time for 1st byte or word</td>
<td></td>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{block, 1-63}</td>
<td>Block program time for each additional byte or word</td>
<td>see Note 2</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{block, End}</td>
<td>Block program end-sequence wait time</td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{Mass Erase}</td>
<td>Mass erase time</td>
<td>10^5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{Scan Erase}</td>
<td>Segment erase time</td>
<td>4810</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTES: 1. The cumulative program time must not be exceeded when writing to a 64-byte flash block. This parameter applies to all programming methods: individual word/byte write and block write modes.
2. These values are hardwired into the Flash Controller's state machine (t_{FTG} = 1/f_{FTG}).

What is \( f_{FTG} \)? ____________ (pick the highest frequency/shortest period)

What is \( t_{word} \)? ____________

Calculate the time to program a word or byte ____________

Multiply that by 1024 words ____________

We calculated that the time required to program the entire F2013 2KB Flash array as random words is 64.5ms.

STOP

CCS users … you’re done
*** This page left blank by order of the fire marshal ***
Standard Definitions

- Standard definitions make code easier to read and debug
- Peripheral bit definition files are included with all tools

Controlling GPIO Ports

- Input Register PxIN
- Output Register PxOUT
- Direction Register PxDIR
- Function Select PxSEL
- Interrupt Edge PxIES
- Interrupt Enable PxIE
- Interrupt Flags PxIFG

P1 and P2 only

Let's do a lab...
*** Yet another senseless waste of resources ***
Lab 2 – I/O Overview

In this lab we’ll configure I/O ports on a FG4618 or FG4619 to recognize an interrupt from a switch and toggle an LED.

### Lab 2: I/O Overview

- **Configure Port1 and Port2 of the MSP430FG4618/9**
  - P1.0 as input with interrupt enabled
  - P1.0 interrupt on H-L transition
  - P2.1 as output to turn on LED
- **Inside of P1ISR**
  - Clear pending interrupt flag

---

**Review Questions**

---

**MSP430 One Day Workshop - Introduction** 1 - 35
Hardware list:

- WinXP PC
- MSP-FET430UIF
- USB cable
- JTAG ribbon cable
- MSP430FG461x/F28xx Experimenter’s Board with batteries
- Digital Multimeter
- Jumpers
- Two AAA Batteries

Software list:

- IAR Kickstart for MSP430 version 4.11B
- Code Composer Studio 4.0
- Labs
- Additional pdf documentation
- Adobe™ Reader
IAR Kickstart Procedure

1. **JTAG**

Remove the JTAG ribbon cable from the MSP430F2013 debug port on the Experimenter’s Board and connect it to the MSP430FG4619 port as shown on page 1-19. The red LED next to the MSP430F2013 emulator port should start blinking again. After all, the program is still in flash memory and you just applied power to the part …

2. **Start IAR Kickstart**

Double-click on the *IAR Kickstart* shortcut on the desktop to start the tool. When the *Embedded Workbench Startup* dialogue appears, click **Cancel**.

**New Workspace and Project**

3. **New Workspace**

Create a new workspace by clicking **File ➔ New ➔ Workspace** on the menu bar. We could have used the previous workspace, but for clarity and practice, let’s make a new one.

4. **New Project**

Create a new project named **Lab2** and save it in the C:\MSP430\IAR Labs\Lab2 folder. If you are unsure how to do this, look back at Lab1.

**Configure the Project**

**NOTE:** The Experimenter’s Board at your workstation may have either a FG4618 or a FG4619 device installed on it. It’s important at this point that you look at the device itself and identify which part you have.

Feel free to write it down here _____________________

5. **Configure the Project**

Click **Project ➔ Options** on the menu bar. Change the target device to the MSP430FG4618 or MSP430FG4619.

In the **Debugger** category, change the Driver to **FET Debugger**.

In the **FET Debugger** category, change the Connection to **Texas Instrument USB-IF**. Click **OK**.

**Add Source File**

6. **Add the source file to the project**

Click **Project ➔ Add Files** on the menu bar. Select **Lab2_exercise.c** from the C:\MSP430\IAR Labs\Lab2 folder and click **Open**.
Complete the Code

7. Answer some questions

Fill in the four blanks in the code on the facing page.

Where will you find the information to complete this task? Start by searching your workstation PC for the MSP430x4xx Family User’s Guide (slau056g.pdf). The Digital I/O section contains some pertinent information. You might also want to open the header file included at the start of the program (msp430xG46x.h), which is also on your PC.

If seeing the schematic will help, try MSP-EXP430FG4618Schematic.pdf.

A couple other files of interest are MSP430FG4618.sfr and .ddf. (or MSP430FG4619.sfr and .ddf). The first file is the peripheral I/O registers and bits definition. The second file is the I/O register description file.

Finally, if you just want to throw up your hands and give up, you can look in the Lab2_solution.c file in the Lab2 folder or see the completed code in the Addendum chapter at the end of the workbook.

Once you have completed the paper exercise, type your answers into the code in Lab2_exercise.c.
```c
#include <msp430xG46x.h>

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;   // Stop WDT
    FLL_CTL0 |= XCAP14PF;       // Configure load caps
    P2DIR = ____;               // Set P2.1 to output direction
    P1IES = ____;               // H-L transition
    P1IE = ____;                // Enable interrupt
    _EINT();                    // Enable interrupts
    while (1);
}

// P1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void P1ISR (void)
{
    unsigned volatile int i;
    for (i=10000; i>0; i--);   // Debounce delay
    P1IFG &= ~____;            // Clear P1IFG
    if ((P1IN & 0x01) == 0)
        P2OUT ^= 0x02;        // Toggle P2.1 using exclusive-OR
}
```
**Test Your Code**

8. **Compile, Download and Debug**

Click the **Debug** button to compile and download your code to the MSP430FG4618/9. When prompted to save your workspace, name it **Lab2** and save it in the **Lab2 folder**. Correct any errors that you may find.

9. **Run Your Code**

Click the **Go** button. If your code works, LED3 (yellow, near the FG4618/9 debug port) should toggle each time you press S1 on the bottom right of the Experimenter’s Board.

10. **Code Explanation**

In case you haven’t already figured it out, the first part of the Lab2 code sets up the ports; one for output and the other as an interrupt input. Execution is then trapped by a while(1) statement until an interrupt occurs. The second part of the code is the interrupt service routine (ISR). When an interrupt occurs, execution of code is vectored to this ISR through the use of the `#pragma` statement.

The mechanical contacts within a pushbutton switch can literally bounce hundreds of times before finally coming to rest, and a microcontroller is fast enough to try to respond to most of them as legitimate key presses. The `for` statement located first in the ISR allows time for the switch contacts to stabilize. The following statement clears the interrupt flag for port1. If you fail to do this, the ISR will only run once! The final `IF` statement detects whether the switch is depressed and toggles the LED port using an XOR. After that, execution is again trapped in the `while(1)` statement.
11. Some Debugging Fun

How can you know if an ISR is running properly? You might be surprised how few students know the right answer. By setting a breakpoint on the first instruction!

If your code is still running, halt it by clicking the Break button on the menu bar.

Reset the CPU by clicking the Reset button.

Double-click to the left of the for statement in the ISR code (in the gray area). This will set a breakpoint just before the instruction executes. It should look like this:

```
for (i=10000; i>0; i--); // Debounce delay
```

Click the Go button. The green arrow and highlight (indicating the position of the Program Counter) over the first instruction in main() should go away. Nothing else should happen until you press S1 … go ahead and press it now. You should see this:

```
for (i=10000; i>0; i--); // Debounce delay
```

Now you can see (by the green arrow) that indeed, the ISR code is about to run for the first time.

At this point it might be nice to check on the status of the port pins. Click View ➔ Register. A window will appear on the right of the IAR Workbench. In the drop-down menu select Port 1/2. Expand P1IN and P2OUT by clicking the + to the left. If you ever get confused about exactly which hardware port/pin you’re dealing with, this is a good way to find out.

P1IN – P0 (Port 1 input pin 0) is the MSP430 input pin reading the status of the pushbutton.

P2OUT – P1 (Port 2 output pin 1) is the MSP430 pin connected to the LED

Start the code running again by clicking the Go button, then press S1. Unless you continue pressing S1 when you click Go, the LED won’t toggle since the IF statement didn’t detect S1 being pressed. Try this a few times, and notice the register values change. You may want to set other breakpoints in the ISR code to better see the values change.

12. Shut Down

When done, click the Stop Debugging button and close IAR Kickstart.

IAR Kickstart users … you’re done. Proceed to the Review Questions at the end of this module.
*** Bottled water … what’s next? Bottled air? ***
CCS 4.0 Procedure

1. JTAG

Remove the JTAG ribbon cable from the MSP430F2013 debug port on the Experimenter’s Board and connect it to the MSP430FG4619 port as shown on page 1-19. The red LED next to the MSP430F2013 emulator port should start blinking again. After all, the program is still in flash memory and you just applied power to the part …

2. Start CCS and Create New Workspace

Double-click on the Code Composer Studio shortcut on the desktop to start the tool. When the Select a Workspace window appears, enter C:\MSP430ODW\CCS Labs\Lab2\workspace in the dialog, and click OK. Close the Error Log and Welcome windows when they appear.

3. New Project

Create a new project named Lab2 and save it in the Lab2 workspace folder. If you are unsure how to do this, or have a short term memory issue, look back at Lab1.

NOTE: The Experimenter’s Board at your workstation may have either a FG4618 or a FG4619 device installed on it. It’s important at this point that you look at the device itself and identify which part you have.

Feel free to write it down here _____________________

When you reach the Project Settings window, make sure to select the correct Device Variant, written above. This project will not be an assembly project.
Add a Source File

4. Add the source file to the project

Right-click in the Project pane and select Add Files to Project. Select Lab2_exercise.c from the C:\MSP430\CCS Labs\Lab2 folder and click Open.

Double-click on Lab2_exercise.c in the Project pane to open the file for editing.

Complete the Code

5. Answer some questions

Fill in the four blanks in the code on the facing page.

Where will you find the information to complete this task? Start by searching your workstation PC for the MSP430x4xx Family User’s Guide (slau056g.pdf). The Digital I/O section contains some pertinent information. You might also want to open the header file included at the start of the program (msp430xG46x.h), which is also on your PC.

If seeing the schematic will help, try MSP-EXP430FG4618Schematic.pdf.

A couple other files of interest are MSP430FG4618.sfr and .ddf (or MSP430FG4619.sfr and .ddf). The first file is the peripheral I/O registers and bits definition. The second file is the I/O register description file.

Finally, if you just want to throw up your hands and give up, you can look in the Lab2_solution.c file in the Lab2 folder or see the completed code in the Addendum chapter at the end of the workbook.

Once you have completed the paper exercise, type your answers into the code in Lab2_exercise.c.

```c
#include <msp430xG46x.h>

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;  // Stop WDT
    FLL_CTL0 |= XCAP14PF;       // Configure load caps
    P2DIR = ____;                // Set P2.1 to output direction
    P1IES = ____;                // H-L transition
    P1IE = ____;                 // Enable interrupt
    _EINT();                     // Enable interrupts
    while (1);
}

// P1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void P1ISR (void)
{
    unsigned volatile int i;
    for (i=10000; i>0; i--);     // Debounce delay
    P1IFG &= ~____;              // Clear P1IFG
    if ((P1IN & 0x01) == 0)
    P2OUT ^= 0x02;             // Toggle P2.1 using exclusive-OR
}
```
Test Your Code

6. Compile, Download and Debug

Click the Debug button to compile and download your code to the MSP430FG4618/9. Correct any errors that you may find.

7. Run Your Code

Click the Run button. If your code works, LED3 (yellow, near the FG4618/9 debug port) should toggle each time you press S1 on the bottom right of the Experimenter’s Board.

8. Code Explanation

In case you haven’t already figured it out, the first part of the Lab2 code sets up the ports; one for output and the other as an interrupt input. Execution is then trapped by a while(1) statement until an interrupt occurs. The second part of the code is the interrupt service routine (ISR). When an interrupt occurs, execution of code is vectored to this ISR through the use of the #pragma statement.

The mechanical contacts within a pushbutton switch can literally bounce hundreds of times before finally coming to rest, and a microcontroller is fast enough to try to respond to most of them as legitimate key presses. The for statement located first in the ISR allows time for the switch contacts to stabilize. The following statement clears the interrupt flag for port1. If you fail to do this, the ISR will only run once! The final IF statement detects whether the switch is depressed and toggles the LED port using an XOR. After that, execution is again trapped in the while(1) statement.
9. Some Debugging Fun

How can you know if an ISR is running properly? You might be surprised how few students know the right answer. By setting a breakpoint on the first instruction!

If your code is still running, halt it by clicking the Halt button on the menu bar.

Reset the CPU by clicking the Reset CPU button.

Double-click to the left of the for statement in the ISR code (in the gray area). This will set a breakpoint just before the instruction executes. It should look like this:

```
for (i=10000; i>0; i--); // Debounce delay
```

Click the Run button. The blue arrow and green highlight (indicating the position of the Program Counter) over the first instruction in main() should go away. Nothing else should happen until you press S1 … go ahead and press it now. You should see this:

```
for (i=10000; i>0; i--); // Debounce delay
```

Now you can see (by the blue arrow) that indeed, the ISR code is about to run for the first time.

At this point it might be nice to check on the status of the port pins. Click View ⇒ Registers. A window will appear on the top-right of the CCS display. Click the + next to Port 1/2. Expand P1IN and P2OUT by clicking the + to the left. Re-arrange the window so that you can see the display clearly. If you ever get confused about exactly which hardware port/pin you’re dealing with, this is a good way to find out.

P1IN – P0 (Port 1 input pin 0) is the MSP430 input pin reading the status of the pushbutton. P2OUT – P1 (Port 2 output pin 1) is the MSP430 pin connected to the LED

Start the code running again by clicking the Run button, then press S1. Unless you continue pressing S1 when you click Run, the LED won’t toggle since the IF statement didn’t detect S1 being pressed. Try this a few times, and notice the register values change. You may want to set other breakpoints in the ISR code to better see the values change.

10. Shut Down

When done, click the Terminate All button and exit Code Composer Studio.

CCS 4.0 users … you’re done.
### Review Questions

<table>
<thead>
<tr>
<th>Review</th>
</tr>
</thead>
<tbody>
<tr>
<td>✦ How many general purpose registers does the MSP430 have?</td>
</tr>
<tr>
<td>✦ What is the purpose of the constant generator?</td>
</tr>
<tr>
<td>✦ Where is the best resource for MSP430 information?</td>
</tr>
<tr>
<td>✦ At reset, all I/O pins are set to …</td>
</tr>
<tr>
<td>✦ Why should you use standard definitions?</td>
</tr>
</tbody>
</table>

You can find the answers to these questions in the Addendum section at the end of this workbook.