Important Notice

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI’s standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer’s applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI’s publication of information regarding any third party’s products or services does not constitute TI’s approval, warranty or endorsement thereof.

Copyright © 2005, 2010 Texas Instruments Incorporated

Revision History

Version 2.00  September 2007  TTO release of workshop
Version 2.10  January 2008  Errata
Version 2.20  May 2008  Errata
Version 3.0  March 2009  Include MSP430F5xx and CCS 4.0, general update
Version 3.1  October 2009  Additional 5xx material, general update
Version 3.2  January 2010  Update to CCS4.1

Mailing Address

Texas Instruments
Training Technical Organization
7839 Churchill Way, M/S 3984
Dallas, Texas 75251-1903
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
*** This page intentionally left blank ***
MSP430 4xx One Day Workshop 2010

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

MSP430 4xx One Day Workshop 2010........................................................................................................ 1-3
TI Microcontroller Portfolio ..................................................................................................................... 1-5
MSP430 Generations ................................................................................................................................. 1-5
MSP430 Peripheral Overview .................................................................................................................... 1-6
MSP430 Portfolio ...................................................................................................................................... 1-6
LCD Controllers ....................................................................................................................................... 1-7
USB ......................................................................................................................................................... 1-8
CC430 ..................................................................................................................................................... 1-9
Innovative Peripherals ............................................................................................................................... 1-9
FRAM ......................................................................................................................................................... 1-10
No-Power Apps ...................................................................................................................................... 1-10
Key Family Features ................................................................................................................................. 1-11
The Nuts and Bolts ................................................................................................................................. 1-13
Embedded Emulation .............................................................................................................................. 1-17
Innovative Tools .................................................................................................................................... 1-18
eZ430-Chronos ..................................................................................................................................... 1-18
Code Composer Studio V4 ....................................................................................................................... 1-19
IAR Systems .......................................................................................................................................... 1-19
IAR Kickstart ......................................................................................................................................... 1-20
Third Party Resources ............................................................................................................................ 1-20
www.ti.com/msp430 .............................................................................................................................. 1-21
Community Support ................................................................................................................................. 1-21
MSP430 Summary .................................................................................................................................. 1-22
Reset Conditions ..................................................................................................................................... 1-22
Experimenter’s Board .............................................................................................................................. 1-23

Lab 1 – Flash the LED ............................................................................................................................... 1-25
Hardware list: ......................................................................................................................................... 1-26
Software list: .......................................................................................................................................... 1-26

IAR Kickstart Procedure .......................................................................................................................... 1-27
Install IAR Kickstart ............................................................................................................................... 1-27
Hardware Verification .............................................................................................................................. 1-29
Power jumpers ....................................................................................................................................... 1-30
IAR Kickstart ......................................................................................................................................... 1-31
Configuring the Project .......................................................................................................................... 1-31
Create and Add the Source File ............................................................................................................ 1-32
Download and Run the Program ............................................................................................................ 1-33
FLASH Programming Exercise ............................................................................................................. 1-34

CCS 4.1 Procedure ................................................................................................................................... 1-35
Install Code Composer Studio .................................................................................................................. 1-35
Hardware Verification ............................................................................................................................. 1-38
Power jumpers ....................................................................................................................................... 1-39
CCS 4.1 ....................................................................................................................................................... 1-40
Configuring the Target ............................................................................................................................ 1-41
Understanding the IDE Display ............................................................................................................. 1-42
Create and Add a Source File .................................................................................................................. 1-43
Download and Run the Program ............................................................................................................. 1-44
FLASH Programming Exercise ............................................................................................................. 1-45

Standard Definitions ................................................................................................................................. 1-47
Controlling GPIO Ports ............................................................................................................................ 1-47
Lab 2 – I/O Overview .................................................................................................................. 1-49
  Hardware list: ....................................................................................................................... 1-50
  Software list: .......................................................................................................................... 1-50

IAR Kickstart Procedure ........................................................................................................ 1-51
  New Workspace and Project ................................................................................................. 1-51
  Configure the Project ........................................................................................................... 1-51
  Add Source File ................................................................................................................... 1-51
  Complete the Code ............................................................................................................... 1-52
  Test Your Code ................................................................................................................... 1-54

CCS 4.1 Procedure .................................................................................................................... 1-57
  Add a Source File .................................................................................................................. 1-58
  Complete the Code ............................................................................................................... 1-58
  Test Your Code ................................................................................................................... 1-60

Review Questions ...................................................................................................................... 1-62
**MSP430 Generations**

<table>
<thead>
<tr>
<th></th>
<th>2xx</th>
<th>4xx</th>
<th>5xx</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Clock (Max)</td>
<td>16MHz</td>
<td>8 &amp; 16 MHz</td>
<td>25MHz</td>
</tr>
<tr>
<td>Flash/RAM (Largest comparable device)</td>
<td>120KB/4KB (F24xx)</td>
<td>120KB/4KB (FG4xx)</td>
<td>256KB/16KB (F54xx)</td>
</tr>
<tr>
<td>Active Current (3.0V) (µA/MIPS)</td>
<td>515 µA/1MHz</td>
<td>600 µA/8MHz</td>
<td>465 µA/18MHz</td>
</tr>
<tr>
<td>Power Down Current (LPM4/5)</td>
<td>0.1µA/1MHz</td>
<td>0.1µA</td>
<td>1.6µA (LPM4) / 0.1µA (LPM5)</td>
</tr>
<tr>
<td>Wake-up Time From LPM3</td>
<td>1µs</td>
<td>6µs</td>
<td>5µs</td>
</tr>
<tr>
<td>Flash ISP Minimum UVLO</td>
<td>2.2V</td>
<td>2.7V</td>
<td>1.8V</td>
</tr>
<tr>
<td>Port I/O Interrupt Capability</td>
<td>P1/P2</td>
<td>P1/P2</td>
<td>P1/P2 (F5438)</td>
</tr>
<tr>
<td>Port Pin Drive Strength</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Port Pin Drive Strength</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Available MCLK Sources</td>
<td>DCO, VLO, LFX1, XT2</td>
<td>FLL, LFX1, XT2</td>
<td>FLL, VLO, REFO, XT1, XT2</td>
</tr>
<tr>
<td>FLL Reference Clocks</td>
<td>N/A</td>
<td>LFX1</td>
<td>REFO, XT1, XT2</td>
</tr>
</tbody>
</table>

**Lowest active power in the industry**

**Write to Flash at min UVLO**

---

**MSP430 4xx One Day Workshop 2010**

---

**TI Microcontroller Portfolio**

**TI Embedded Processing Portfolio**

**Microcontrollers**

<table>
<thead>
<tr>
<th>16-bit</th>
<th>32-bit Real-time</th>
<th>32-bit ARM</th>
<th>ARM+</th>
<th>ARM + DSP</th>
<th>DSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSP430</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ultra-Low Power</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Up to 25 MHz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flash 1 KB to 256 KB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analog I/O, ADC, LCD, USB, RF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Measurement, Sensing, General Purpose</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0.49 to $9.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ARM-Based**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C2000™</td>
<td></td>
<td></td>
<td>ARM Cortex A8</td>
<td></td>
</tr>
<tr>
<td>Fixed &amp; Floating Point</td>
<td></td>
<td></td>
<td>Industry Std Core</td>
<td></td>
</tr>
<tr>
<td>Up to 150 MHz</td>
<td></td>
<td></td>
<td>High-Perform GPP</td>
<td></td>
</tr>
<tr>
<td>$1.00 to $20.00</td>
<td></td>
<td></td>
<td>Accelerators</td>
<td></td>
</tr>
</tbody>
</table>

**DSP**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM+ DSP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C647x, C64x+, C55x</td>
<td></td>
<td></td>
<td>Leadership DSP Performance</td>
<td></td>
</tr>
<tr>
<td>32K To 150 MHz</td>
<td></td>
<td></td>
<td>48K MWA/Cs/ 107DMPS/55MHz</td>
<td></td>
</tr>
<tr>
<td>$1.50 to $20.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**MSP430 Generations**

**MSP430 Peripheral Overview ...**
**MSP430 Peripheral Overview**

### Basic Clock System
- Core voltage same as supply voltage (1.8-3.6V)

### Core Voltage
- Core voltage same as supply voltage (1.8-3.6V)

### 16-bit CPU
- 16-bit CPU, CPUX

### GPIO
- GPIO w/ pull-up and pull-down

### 16-bit CPU
- 16-bit CPU, CPUX

### GPIO
- GPIO w/ pull-up and pull-down

### Programmable Core Voltage
- Programmable core voltage with integrated PMM (1.8-3.6V)

### USART
- USART, USCI

### DMA
- DMA up to 3-ch

### MPY
- MPY16

### ADC
- ADC10,12

### JTAG
- 4-wire JTAG, 2-wire Spy Bi-Wire

**MSP430 Portfolio**

### MSP430 1xx Catalog
- 10 MIPS
- 128 kB Flash
- 512 kB RAM
- 500 nA Standby
- 1.8 - 3.6V

### MSP430 2xx Catalog
- 16 MIPS
- 120 kB Flash
- 6 kB RAM
- 2-wire Spy Bi-Wire
- 1.8 - 3.6V

### MSP430 4xx Catalog
- 160 kB Flash
- 8 kB RAM
- 1.8 - 3.6V

### MSP430 5xx Catalog
- 25 MIPS
- 256 kB Flash
- 16 kB RAM
- 1.8 - 3.6V
- FRAM, USB, RF

### MSP430 6xx Catalog
- 160 uA/MIPS
- LCD Controller
- 160 x 4 MIPS

### MSP430 7xx Catalog
- 160 x 4 MIPS
- LCD Controller
- 160 x 4 MIPS

### MSP430 8xx Catalog
- 160 x 4 MIPS
- LCD Controller
- 160 x 4 MIPS

**MSP430 Portfolio ...**
LCD Controllers

- Ultra-low power
- Fully automatic
- 4/3/2/1 mux
- Up to 160-bit display
- Internal regulated voltage generator
- Internal or external bias generation
- Contrast control
- 1/2 bias for 3 or 4 mux
- Internal clock generation
- Auto segment blinking
**USB**

**Enabling You with Full Speed USB**

**Ultra-low power MCUs + USB for smarter connectivity**

- Embedded full-speed USB 2.0 (12 Mbps)
- High flexibility with configurable 2K data buffers that can be used as RAM
- Unused USB interface pins can function as high-current I/O (5V tolerant)

**Analog and peripheral integration reduces system cost**

- Multiple analog options with 10 or 12-bit ADC, DAC, comparator
- Integrated 3.3V LDO for use with 5V USB bus power
- Uses low-cost crystal for USB clock, with flexible, integrated PLL

**44 New USB devices within next 12 months**

- Wide range of memory configurations and package options, 8k-128k flash
- Diverse peripheral mix in the MSP430F55xx family
- Pricing as low a $0.96 in volume

USB made easy ...

**USB Made Easy**

- **USB Bootstrap Loader (USB)**
  - Supporting device programming
  - Field Firmware updates
- **USB Descriptor Tool**
  - Configures stack functions via GUI
- **Free USB stacks available:**
  - Communication Device Class (CDC)
  - Human Interface Device (HID)
  - Mass Storage Class (MSC)
- **Additional stacks available from third parties**

MSP430F5529 Sample Kit

FREE Vendor ID/ Product ID sharing program

FREE Request for enabled products...
CC430

CC430: Enabling You With RF

Lowest Power Monolithic RF SoC

The Best of Both Worlds

Low Power RF Transceiver
- High sensitivity
- Low current consumption
- Excellent blocking performance
- Flexible data rate & modulation format
- Backwards compatible

MSP430 MCU
- Market's lowest power MCU
- High analog performance
- High level of integration
- Ease of development
- Sensor interface

Innovative peripherals ...

Innovative Peripherals

CC430: Innovative Peripherals

LCD_B
- Blinking of individual segments, Programmable frame frequency,
  Software-driver contrast control
- Regulated charge pump
- Integrated drivers

AES 128
- Encryption and decryption according to AES FIPS PUB 197
  with 128-bit keys
- Key expansion for en- and decryption
- Off-line key generation for decryption

Comparator_B
- Selectable reference voltage & voltage hysteresis generator
- High-speed, normal, and ultra-low power 100nA modes
- Internal output to Timer A capture
- Selectable RC filter for comparator output

FRAM ...
FRAM

FRAM: The Future of MCU Memory

- **Non-volatile, Reliable Storage**
  - Over 100 Trillion write/read cycles
  - Write Guarantee in case of power loss
- **Fast write times like SRAM**
  - ~50ns per byte or word
  - 1,000x faster than Flash/EEPROM
- **Low Power**
  - Only 1.5v to write & erase
  - >10-14v for Flash/EEPROM
- **Universal Memory**

No-Power Apps

MSP430 Enables No-Power Apps

- **Energy harvesting** is the process by which energy is captured and stored
- Can substitute batteries that are costly to maintain and can extend system uptime
- Only possible with ultra-low power components
- Solar, kinetic, thermal, RF, salinity gradients, pH difference and other ambient sources available
Key Family Features

F2xx Key Features

- <1µA standby LPM3
- <1µs 0-16MHz
- Zero-power BOR
- Failsafe oscillator
- Enhanced watchdog
- Pull-up / down resistors
- Hack proof boot loader
- 2.2V Flash ISP
- Extended temp 105°C
- Same instruction set architecture

F4xx Key Features

- <1µA standby LPM3
- <1µs 0-16MHz
- 4-120 KB Flash
- Built-in LCD Driver
- Zero-power BOR
- Pull-up / down resistors
- 2.7V Flash ISP
- Same instruction set architecture

F5xx key features
F5xx Key Features

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 docking 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

MSP430F5xx Block Diagram
The Nuts and Bolts

16-bit RISC CPU

- Efficient, ultra-low power CPU
- C-compiler friendly
- RISC architecture
  - 27 core instructions
  - 24 emulated instructions
  - 7 addressing modes
  - Constant generator
- Single-cycle register operations
- Memory-to-memory atomic addressing
- Bit, byte and word processing
- 20-bit addressing on MSP430X for Flash >64KB

Bytes, Words And CPU Registers

16-bit addition

<table>
<thead>
<tr>
<th>Code</th>
<th>Instruction</th>
<th>Address</th>
<th>16-bit addition</th>
<th>Code/Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>5405</td>
<td>add.w R4,R5</td>
<td>&amp;00000000</td>
<td>16-bit addition</td>
<td>1/1</td>
</tr>
<tr>
<td>529202000202</td>
<td>add.w &amp;0200,&amp;0202</td>
<td></td>
<td>Code/Cycles</td>
<td>3/6</td>
</tr>
</tbody>
</table>

8-bit addition

<table>
<thead>
<tr>
<th>Code</th>
<th>Instruction</th>
<th>Address</th>
<th>8-bit addition</th>
<th>Code/Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>5445</td>
<td>add.b R4,R5</td>
<td>&amp;00000000</td>
<td>8-bit addition</td>
<td>1/1</td>
</tr>
<tr>
<td>52D202000202</td>
<td>add.b &amp;0200,&amp;0202</td>
<td></td>
<td>Code/Cycles</td>
<td>3/6</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Mode</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Register Mode</strong></td>
<td><code>mov.w R10,R11</code></td>
<td>Single cycle</td>
</tr>
<tr>
<td><strong>Indexed Mode</strong></td>
<td><code>mov.w 2(R5),6(R6)</code></td>
<td>Table processing</td>
</tr>
<tr>
<td><strong>Symbolic Mode</strong></td>
<td><code>mov.w EDE,TONI</code></td>
<td>Easy to read code, PC relative</td>
</tr>
<tr>
<td><strong>Absolute Mode</strong></td>
<td><code>mov.w &amp;EDE,&amp;TONI</code></td>
<td>Directly access any memory</td>
</tr>
<tr>
<td><strong>Indirect Register Mode</strong></td>
<td><code>mov.w @R10,0(R11)</code></td>
<td>Access memory with pointers</td>
</tr>
<tr>
<td><strong>Indirect Autoincrement</strong></td>
<td><code>mov.w @R10+,0(R11)</code></td>
<td>Table processing</td>
</tr>
<tr>
<td><strong>Immediate Mode</strong></td>
<td><code>mov.w #45h,&amp;TONI</code></td>
<td>Unrestricted constant values</td>
</tr>
</tbody>
</table>

**Atomic Addressing**

B = B + A

- Memory
  - B
  - A

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

; MSP430
add     A,B
```

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

**Atomic addressing...**

**Constant generator...**
Constant Generator

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

24 Emulated Instructions

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

Format I
Source and Destination

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example 1</th>
<th>Example 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>add.w R4,R5</td>
<td>R4+R5=R5 xxxx</td>
<td></td>
</tr>
<tr>
<td>add.b R4,R5</td>
<td>R4+R5=R5 00xx</td>
<td></td>
</tr>
</tbody>
</table>

Format II
Destination Only

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>rlc.w R4</td>
<td></td>
</tr>
<tr>
<td>rlc.b R4</td>
<td></td>
</tr>
</tbody>
</table>

Format III
8(Un)conditional Jumps

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>jmp Loop_1</td>
<td>Goto Loop_1</td>
</tr>
</tbody>
</table>

51 Total Assembly Instructions

<table>
<thead>
<tr>
<th>Format I: Source, Destination</th>
<th>Format II: Single Operand</th>
<th>Format III: +/- 9bit Offset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(.b)</td>
<td>br</td>
<td>jmp</td>
<td>clrc</td>
</tr>
<tr>
<td>addc(.b)</td>
<td>call</td>
<td>jc</td>
<td>setc</td>
</tr>
<tr>
<td>and(.b)</td>
<td>swpb</td>
<td>jnc</td>
<td>clrz</td>
</tr>
<tr>
<td>bis(.b)</td>
<td>sxt</td>
<td>jae</td>
<td>clrn</td>
</tr>
<tr>
<td>bit(.b)</td>
<td>push(.b)</td>
<td>jse</td>
<td>clrn</td>
</tr>
<tr>
<td>cmp(.b)</td>
<td>rra(.b)</td>
<td>j1</td>
<td>dint</td>
</tr>
<tr>
<td>dadd(.b)</td>
<td>rrc(.b)</td>
<td>ja</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>daddc(.b)</td>
<td>daddc(.b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dadd(.b)</td>
<td>dac(.b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dac(.b)</td>
<td>dac(.b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>daddc(.b)</td>
<td>dac(.b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rlc(.b)</td>
<td>rlc(.b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tst(.b)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bold type denotes emulated instructions

Unified memory map...
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

- Real-time, in-system debug
  - No application resources used
  - Full speed execution
  - H/W breakpoints
  - Single stepping
  - Complex triggering
  - Trace capability
- Powerful, easy to use tools
- Spy Bi-Wire
  - 2-wire debug interface
  - No pin function impact
- Only 1 tool required for all devices
Innovative Tools

Easy To Use, Innovative Tools

Flash Emulation Tools
- Compatible with all devices
- Target boards available
- $99 ($149 with target board)
- Target boards available without FET
- Free IDEs included

MSP430 Experimenter Boards
- Fully featured prototyping system
- Available for FG4618 & F5438
- Starting at $99

eZ430 Tools
- Complete development system in USB stick
- Available for wireless and energy harvesting
- Starting at $20

MSP430 Experimenter Boards
- Fully featured prototyping system
- Available for FG4618 & F5438
- Starting at $99

Flash Emulation Tools
- Compatible with all devices
- Target boards available
- $99 ($149 with target board)
- Target boards available without FET
- Free IDEs included

MSP430 Experimenter Boards
- Fully featured prototyping system
- Available for FG4618 & F5438
- Starting at $99

Chronos ...

eZ430-Chronos

eZ430-Chronos: CC430 Dev Tool

- CC430-based wireless development tool in a watch
- 915/868/433 MHz versions available
- Custom LCD driven directly by CC430
- Features:
  - 3-axis accelerometer
  - Altimeter
  - Temperature sensor
  - Buzzer

USB RF access point
Updated eZ430 emulator for programming
CCS v4 ...
Code Composer Studio V4

CCE is now Code Composer Studio v4

- Code Composer Studio v4: A single development platform for all TI processors
- CCE users will feel at home
- Enhancements since CCE:
  - Speed
  - Code size improvements
  - Auto-updating
  - License manager
  - Support for all TI MCUs
- Only $495 for MCU Edition
- FREE 16KB-limited edition

IAR Systems

TI and IAR Systems: Deep and Evolving Partnership

- TI and IAR Systems partners on MSP430
- TI and IAR partners on ARM MCUs
- TI acquires Chipcon, partner to IAR Systems
- TI acquires Luminary Micro, partner to IAR Systems
- TI and IAR Systems Product Integration and Support

* IAR Embedded Workbench C/C++ compiler and debugger tool set
* IAR PowerPac RTOS Firmware USB Device Stack TCP/IP stack
* IAR visualSTATE design, test and verification tools using state machines
* IAR KickStart Kits Completely Integrated Kits

IAR Systems
IAR Kickstart

IAR Kickstart IDE

- 4kB Compiler
  (8kB for >60k Flash devices)
- Assembler/Linker
- Editor
- Debugger

Third Party Resources

Third Party Development Resources

- Rowley CrossWorks
  - Complete IDE solution
  - High code density
  - Simulator
  - Windows, Linux, Mac
  www.rowley.co.uk

- MSPGCC Tool Chain
  - Free
  - Open Source
  - GNU C Compiler, Assembler/Linker, GDB Debugger
  - Windows, Linux, Unix
  http://mspgcc.sourceforge.net

- Elprotronic
  - MSP430, CC Chipcon, C2000 Programmers
  - Fastest download speed
  - Production programmers

- Amber Wireless
  - Drop in wireless modules
  - <1GHz ez430-RF target boards
  - CC430 Development boards

- RTOS Options
  - µC/OS-II™
  - CMX-Tiny™
  - embOS
  - FreeRTOS™
  - IAR PowerPac
  - QP™
  - Salvo™
  - TinyOS

- USB Stacks
  - IAR
  - HCC

www.rowley.co.uk
www.ti.com/msp430

Community Support

Extensive Community Support

E2E Community

- Videos, Blogs, Forums
- Extensive community support and idea exchange
- Global customer support
- http://e2e.ti.com

Processor Wiki

- Growing collection of technical wiki articles
- Tips & tricks, common pitfalls, and design ideas
- http://wiki.msp430.com

Summary...
MSP430 Summary

- Ultra-low Power
- Broad portfolio
  - Access for size and cost constraints
  - Performance for precision and speed
- Enabling Technologies
  - FRAM, USB, RF, energy harvesting
- Ease of Use
  - HW and SW Tools
  - Community

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
Experimenter’s Board

MSP430FG461x/F20xx Experimenter’s Board

Two MSP430 devices:
- MSP430FG4618 or MSP430FG4619
- MSP430F2013

Interface for ChipCon RF transceiver EMK boards
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 4.1. 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
- Jumpers

Software list:

- IAR Kickstart for MSP430 version 4.21B
- Code Composer Studio 4.1
- 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.

Install IAR Kickstart

1. **Disconnect** any evaluation board that you have connected to your PCs USB port(s). **Insert** the Workshop Installation Flash Drive into a free USB port.

2. Using **Windows Explorer**, find and double-click on the file named **EW430-KS-web-4212.exe**.

3. Follow the steps in the IAR installation program. When you reach the **Enter User Information** window, use Windows Explorer to find and open the **IAR License.txt** file on the installation flash drive. **Copy/paste** the license number as shown below and click **Next**.

![IAR Embedded Workbench Kickstart for MSP430 4.21](image)
4. In the same way, **copy/paste** the **License Key** into the next window and click **Next**.

Select a **Complete** installation and click **Next**. Install the tools into the **default folder**, if possible. The installation should take less than 10 minutes to complete.

5. **Driver Installation**
   Using Windows Explorer, look on the workshop flash drive and double-click on `swrc094e setup`. Follow the wizard steps until it completes. Again using Windows Explorer, navigate to `C:\Program Files\Texas Instruments Inc\TUSB3410 Single Driver Installer\DISK1` and double-click on `setup`. Follow the wizard steps until it completes.

6. **Lab Files Installation**
   Using Windows Explorer, look on the workshop flash drive and double-click on `all_labs.exe`. Leave the unzip directory as `C:\` and click **Unzip**. When the process completes, click **Close**. The labs have been placed in `C:\MSP430ODW`.

If you’ve been tasked with installing IAR Kickstart, the drivers and labs only, please stop here and ask your instructor for further directions.
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

If you are prompted to load the driver when you connect the FET to the PC, don’t search the web for the driver and don’t load the driver automatically. You can locate the driver in the C:\Program Files\IAR Systems\Embedded Workbench 5.4 Kickstart\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 *IAR Information Center* window will appear on top of the IAR tool. Click the X in the upper right to close the window.

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.

```asm
#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 ![Debug button](image). 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:

![IAR Embedded Workbench](image)

The buttons on the top-left that look like this: ![Button icons](image) control the running of the code. Click on the **Go** button ![Go button](image) 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 ![Stop Debugging button](image).

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

In the F2xx family, the time to program any bit, byte or word in FLASH is \(30/f_{FTG}\) – 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 \(18/f_{FTG}\).

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></td>
<td>3.6</td>
<td>V</td>
</tr>
<tr>
<td>(f_{FTG})</td>
<td>Flash Timing Generator frequency</td>
<td></td>
<td>257</td>
<td></td>
<td>478</td>
<td>kHz</td>
</tr>
<tr>
<td>(I_{PGM})</td>
<td>Supply current from VCC during program</td>
<td>2.7</td>
<td>3</td>
<td></td>
<td>3.6</td>
<td>mA</td>
</tr>
<tr>
<td>(I_{ERASE})</td>
<td>Supply current from VCC during erase</td>
<td>2.7</td>
<td>3</td>
<td></td>
<td>3.6</td>
<td>mA</td>
</tr>
<tr>
<td>(t_{CE})</td>
<td>Cumulative program time</td>
<td>see Note 1</td>
<td>2.7</td>
<td></td>
<td>3.6</td>
<td>ms</td>
</tr>
<tr>
<td>(t_{CM})</td>
<td>Cumulative erase time</td>
<td>see Note 1</td>
<td>2.7</td>
<td></td>
<td>3.6</td>
<td>ms</td>
</tr>
<tr>
<td>(T_{rel})</td>
<td>Data retention duration</td>
<td>(T_j = 25^\circ)</td>
<td>100</td>
<td></td>
<td></td>
<td>years</td>
</tr>
<tr>
<td>Word</td>
<td>Word or byte program time</td>
<td></td>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>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>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>Block, End</td>
<td>Block program size or erase time</td>
<td>see Note 2</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mass Erase</td>
<td>Mass erase time</td>
<td>see Note 2</td>
<td>10593</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Segm Erase</td>
<td>Segment erase time</td>
<td>see Note 2</td>
<td>4819</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**

IAR Kickstart users … you’re done. Proceed to page 1-41.
CCS 4.1 Procedure

In this lab, you will install Code Composer Studio and 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.

Install Code Composer Studio

7. **Disconnect** any evaluation board that you have connected to your PCs USB port(s).
   **Insert** the Workshop Installation Flash Drive into a free USB port.

8. Using **Windows Explorer**, find the `setup_CCS_n.n.n.n` folder on the Flash drive and double-click on the file named `setup_CCS_n.n.n.n.exe`.

9. Follow the instructions in the Code Composer Studio installation program. Select the **Platinum Edition** for installation when the **Product Configuration** dialog window appears. Click **Next**.
10. In the Choose ISA dialog, if you are attending a Stellaris only workshop, make sure that only the Stellaris Cortex-M3 MCU and ARM checkboxes are selected. If you are also attending an MSP430 workshop, check that checkbox too. Click Next.
11. In the **Select Components** dialog, uncheck the **Target Content** and **Emulators** checkboxes. If you are attending a Stellaris only workshop, click **Next**. If you are attending a MSP430 workshop too, check the **MSP430 USB FET** checkbox and click **Next**. The installation should take less than 10 minutes to complete.

12. **Driver Installation**
   Using Windows Explorer, look on the workshop flash drive and double-click on **swrc094e setup**. Follow the wizard steps until it completes. Again using Windows Explorer, navigate to `C:\Program Files\Texas Instruments Inc\TUSB3410 Single Driver Installer\DISK1` and double-click on **setup**. Follow the wizard steps until it completes.

13. **Lab Files Installation**
   Using Windows Explorer, look on the workshop flash drive and double-click on **all_labs.exe**. Leave the unzip directory as `C:\` and click **Unzip**. When the process completes, click **Close**. The labs have been placed in `C:\MSP430ODW`.

   If you’ve been tasked with installing Code Composer, the drivers and labs only, please stop here and ask your instructor for further directions.
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).
Power jumpers

2. 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:

```
LCL     FET
```
CCS 4.1

3. 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 4.1. The *Workspace Launcher* window will appear. 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.

If the Welcome screen appears, close it by clicking on the CCS emblem in the upper right.

4. 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 **MSP430F2XXX** and select **MSP430F2013**.

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

5. 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 tab appears, make the change as shown below:

![Basic window tab](image)

Close the Lab1.ccxml tab by clicking the X on the tab. When prompted, click Yes to save the changes.
Understanding the IDE Display

6. Displayed Windows

CCS 4.1 is a highly customizable tool, but your first view of it should look like below:

If the Cheat Sheets pane is open on the right, close it by clicking the X on the tab.

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

7. 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.

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

.text ; 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

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

.end
```

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

8. 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.

9. Run the Program

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

![Screen with code and buttons](image)

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

10. Halt Debugging and Close CCS

Click the **Terminate All** button 🔄:stop 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

11. Exercise

In the F2xx family, the time to program any bit, byte or word in FLASH is 30/f_{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/f_{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>2.2</td>
<td>3.6</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f_{FTG}</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_{CMerase}</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>t_{Retention}</td>
<td>Data retention duration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{Word}</td>
<td>Word or byte program time</td>
<td></td>
<td></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></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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{Block, End}</td>
<td>Block program end-sequance wait time</td>
<td></td>
<td></td>
<td></td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>t_{Mass Erase}</td>
<td>Mass erase time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t_{Seg Erase}</td>
<td>Segment erase time</td>
<td></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

$\text{WDTCTL} = 0x5A80;$
$\text{WDTCTL} = 0xA580;$
$\text{WDTCTL} = 0xA540; \quad \text{// Hold watchdog timer}$
$\text{WDTCTL} = \text{WDTPW} + \text{WDTHOLD}; \quad \text{// Hold watchdog timer}$

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

Controlling GPIO

Controlling GPIO Ports

P1 and P2 only

Lab2 ...
*** 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.

**Lab2: 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
**Hardware list:**

- WinXP PC
- MSP-FET430UIF
- USB cable
- JTAG ribbon cable
- MSP430FG461x/F28xx Experimenter’s Board
- Jumpers

**Software list:**

- IAR Kickstart for MSP430 version 4.21B
- Code Composer Studio 4.1
- Labs
- Additional pdf documentation
- Adobe™ Reader
IAR Kickstart Procedure

1. **JTAG**

   Remove the JTAG ribbon cable from the MSP430F2013 debug port on the Experimenters 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 Experimenters 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.
#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 4.1 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 Welcome screen when it appears.

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 _____________________**

   Make sure you select the **Project Type** to be MSP430. 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. 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:

```c
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:

```c
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 4.1 users … you’re done.
Review Questions

Review

- How many general purpose registers does the MSP430 have?
- What is the purpose of the constant generator?
- Where is the best resource for MSP430 information?
- At reset, all I/O pins are set to …
- Why should you use standard definitions?

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

In this section we’ll explore the ultra-low power abilities and architecture of the MSP430. We’ll take a look at its low power modes and unique oscillator arrangement, along with techniques that can be used to minimize power consumption.

Objectives

- Principles of ultra-low power applications
- Low power modes
- Oscillators
- Interrupts
- Ultra-low power lab
*** This page isn’t really blank, you know. ***
Module Topics

Ultra-Low Power

Module Topics.................................................................................................................. 2-3
Activity Profile .................................................................................................................. 2-5
Performance on Demand .................................................................................................. 2-5
Low Power Mode Clock Control ...................................................................................... 2-6
Low Power Mode Configuration ......................................................................................... 2-6
Low Power Modes in Assembly ......................................................................................... 2-7
Low Power Modes in C ....................................................................................................... 2-7
11x/12x Basic Clock ........................................................................................................... 2-8
1xx XTAL Options .............................................................................................................. 2-8
1xx DCO Control ................................................................................................................ 2-9
DCO Jitter ............................................................................................................................ 2-9
1xx DCO Calibration ......................................................................................................... 2-10
2xx Basic Clock .................................................................................................................. 2-10
4xx FLL ............................................................................................................................... 2-11
5xx Unified Clock System .................................................................................................. 2-11
5xx Power Management Module ....................................................................................... 2-12
Program Flow ...................................................................................................................... 2-12
Interrupt Processing .......................................................................................................... 2-13
11x1 Interrupt Vectors ....................................................................................................... 2-13
Move S/W Functions to Peripherals .................................................................................. 2-14
Power Manage Internal Peripherals .............................................................................. 2-14
Lowering System Power ................................................................................................... 2-15
Increasing Power Efficiency ............................................................................................. 2-15
Terminate Unused Pins ....................................................................................................... 2-16
Ultra Low Power Principles ............................................................................................... 2-16

Lab 3 – Ultra-Low Power in Practice .................................................................................. 2-17
Lab3 Baseline ...................................................................................................................... 2-19
Lab3 using LPM3 ............................................................................................................... 2-19
Shut Down .......................................................................................................................... 2-19

IAR Kickstart Procedure .................................................................................................. 2-19
Lab3 Baseline ...................................................................................................................... 2-19
Lab3 using LPM3 ............................................................................................................... 2-19
Shut Down .......................................................................................................................... 2-20

Code Composer Studio 4.1 Procedure .............................................................................. 2-23
Lab3 Baseline ...................................................................................................................... 2-23
Lab3 using LPM3 ............................................................................................................... 2-23
Shut Down .......................................................................................................................... 2-24

Review Questions .............................................................................................................. 2-26
Module Topics

*** Let this be your doodle area ***
Activity Profile

Ultra-low Power Activity Profile

Performance on Demand

Performance on Demand

Interrupt

DCO

Immediate-stable clock start for quick reaction to events
## Low Power Mode Clock Control

### LPM0
- CPU Off
- DCO on
- ACLK on
- 45µA

### LPM3
- Stand-by
- DCO off
- ACLK on
- 1.0µA
  - RTC function
  - LCD driver
  - RAM/SFR retained

### LPM4
- Off
- All
- Clocks Off
- 0.1µA
  - RAM/SFR retained

Specific values vary by device.

## Low Power Mode Configuration

### Active Mode
- 0 0 0 0
- ~ 250µA

### LPM0
- 0 0 0 1
- ~ 35µA

### LPM3
- 1 1 0 1
- ~ 0.8µA

### LPM4
- 1 1 1 1
- ~ 0.1µA

### Assembly Code
- `bis.w #CPUOFF,SR ; LPM0`
Low Power Modes in Assembly

Low Power Modes in Assembly

```assembly
ORG 0F000h
RESET mov.w #300h,SP
        mov.w #WDT_MDLY_32,&WDTCTL
        bis.b #WDTIE,&IE1
        bis.b #01h,&P1DIR
Mainloop bis.w #CPUOFF+GIE,SR
        xor.b #01h,&P1OUT
        jmp Mainloop
WDT_ISR bic.w #CPUOFF,0(SP);
        reti
ORG 0FFFEh
DW RESET
ORG 0FFF4h
```

Low Power Modes in C

Low Power Modes in C

```c
void main(void)
{
    WDTCTL = WDT_MDLY_32;
    IE1 |= WDTIE;
    P1DIR |= 0x01;
    for (;;)
    {
        _BIS_SR(CPUOFF + GIE);
        P1OUT ^= 0x01;
    }
}
#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void)
{
    _BIC_SR_IRQ(CPUOFF);
}
```

'L1x/12x Basic Clock
11x/12x Basic Clock

MSP430x11x/12x Basic Clock

<table>
<thead>
<tr>
<th></th>
<th>V</th>
<th>SCG1</th>
<th>SCG0</th>
<th>OSC OFF</th>
<th>CPU OFF</th>
<th>GIE</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2/SR</td>
<td>LFXT1CLK</td>
<td>OSCOFF</td>
<td>LFXT1 Oscillator</td>
<td>ACLK</td>
<td>MCLK</td>
<td>CPUOFF</td>
<td>SCG1</td>
<td>SMCLK</td>
<td>DCOCLK</td>
</tr>
</tbody>
</table>

1xx XTAL Options

F1xx Basic Clock XTAL Options

32768Hz 1-8MHz

Most MSP430 applications use a 32,768Hz crystal

1xx Basic Clock DCO Control
1xx DCO Control

DCO - Digitally Controlled Oscillator

DCO Jitter

The modulator mixes two frequencies to produce the DCO clock.
This spreads the clock energy and reduces electromagnetic interference (EMI).
Due to jitter, DCO cannot be used to lock a PLL.
Module Topics

1xx DCO Calibration

**F1xx DCO Calibration**

Clock precision is achieved by periodic adjustment

```
// Partial SW FLL Code
if (488 < Compare)  // DCO too fast
    DCOCTL--;
else  DCOCTL++;     // DCO too slow
```

- Periodic loop adjusts DCO
- Known reference can be 50/60Hz AC power or 32kHz crystal frequency
- If Rosc = 100k then DCOCLK ~ 2MHz

2xx Basic Clock

**F2xx Basic Clock**

- LFXT1 XTAL Oscillator
  - <1uA LPM3 standby mode
  - XTAL CAPs programmable
  - OSC fault LF/(XT)
  - Very Low Power Oscillator (VLO)
- Improved DCO
  - < 1us 0-to-16MHz
  - ± 2.5% DCO
  - Programmable frequency
- VLO not in F21x1
**Module Topics**

### 4xx FLL

**F4xx Frequency-Locked Loop (FLL)**

- **Fully digital**
- **Oscillator fault fail-safe for LFXT1, DCO and XT2**

![F4xx FLL Diagram](image)

### 5xx Unified Clock System

**F5xx: Unified Clock System**

- **Orthogonal clock system**
  - Any source can drive any clock signal
- **2 Integrated clock sources**:  
  - REFO: 32kHz, trimmed osc.  
  - VLO: 12kHz, ultra-low power
- **DCO & FLL provide high frequency accurate timing**
- **MODOSC provides bullet proof timing for Flash**
- **Crystal pins muxed with I/O function**
5xx Power Management Module

**F5xx: Power Management Module**

- Integrated LDO
- \( V_{\text{CORE}} \) level programmable
- Flexibility in processing performance vs. power
- Integrated supervision & monitoring
- Zero-power BOR
- Five integrated supervisors
  - SVSH
  - SVSL
  - SVMH
  - SVML
  - BOR

Program Flow

**Interrupts Control Program Flow**

100% CPU Load

```
// Polling UART Receive for (;;)
{ 
    while (!(IFG2&URXIFG0)); 
    TXBUF0 = RXBUF0; 
}
```

0.1% CPU Load

```
// UART Receive Interrupt
#pragma vector=UART_VECTOR
__interrupt void rx (void)
{ 
    TXBUF0 = RXBUF0; 
}
```

Interrupt Processing
Interrupt Processing

### Prior to ISR

- PC pushed
- SR pushed
- Interrupt vector moved to PC
- GIE, CPUOFF, OSCOFF and SCG1 cleared
- IFG flag cleared on single source flags

### reti - automatically

- SR popped - original
- PC popped

---

**Vectors**

### 11x1 Interrupt Vectors

#### Interrupt Vectors – F11x1

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>FLAG</th>
<th>INTERRUPT</th>
<th>ADDRESS</th>
<th>PRIORITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power-up</td>
<td>WDTIFG</td>
<td>Reset</td>
<td>0FFFeH</td>
<td>15, highest</td>
</tr>
<tr>
<td>ext. Reset</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Watchdog</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMI</td>
<td>NMIIFG</td>
<td>(non)-maskable</td>
<td>0FFFCh</td>
<td>14</td>
</tr>
<tr>
<td>Osc. Fault</td>
<td>ORIFG</td>
<td>(non)-maskable</td>
<td>0FFFeH</td>
<td>13</td>
</tr>
<tr>
<td>Flash violation</td>
<td>ACCVIFG</td>
<td>(non)-maskable</td>
<td>0FFFeH</td>
<td>11</td>
</tr>
<tr>
<td>Comparator A</td>
<td>CAIFG</td>
<td>maskable</td>
<td>0FFFeH</td>
<td>10</td>
</tr>
<tr>
<td>Watchdog timer</td>
<td>WDTIFG</td>
<td>maskable</td>
<td>0FFFeH</td>
<td>9</td>
</tr>
<tr>
<td>Timer_A</td>
<td>CCIFG0</td>
<td>maskable</td>
<td>0FFFeH</td>
<td>8</td>
</tr>
<tr>
<td>Timer_A</td>
<td>CCIFGx</td>
<td>maskable</td>
<td>0FFFeH</td>
<td>7</td>
</tr>
<tr>
<td>Timer_A</td>
<td></td>
<td></td>
<td>0FFECh</td>
<td>6</td>
</tr>
<tr>
<td>Timer_A</td>
<td></td>
<td></td>
<td>0FFEh</td>
<td>5</td>
</tr>
<tr>
<td>Timer_A</td>
<td></td>
<td></td>
<td>0FFEh</td>
<td>4</td>
</tr>
<tr>
<td>I/O Port P2</td>
<td>P2IFGx</td>
<td>maskable</td>
<td>0FFEh</td>
<td>3</td>
</tr>
<tr>
<td>I/O Port P1</td>
<td>P1IFGx</td>
<td>maskable</td>
<td>0FFEh</td>
<td>2</td>
</tr>
</tbody>
</table>

---

**FLASH**

(x) 512B Segments

**Boot Loader**

(2) 128B

**RAM**

16-bit Peripherals

8-bit Peripherals

Move sw to peripherals
Move S/W Functions to Peripherals

Move Software Functions to Peripherals

```c
// Endless Loop
for (;;) {
    P1OUT |= 0x04; // Set
delay1();
P1OUT &= ~0x04; // Reset
delay2();
}
```

100% CPU Load

Zero CPU Load

Power Manage Internal Peripherals

```c
P1OUT |= 0x02; // Power divider
CACTL1 = CARSEL + CAREF_2 + CAON; // Comp_A on
if (CAOUT & CACTL2)
    P1OUT |= 0x01; // Fault
else
    P1OUT &= ~0x01;
P1OUT &= ~0x02; // de-power divider
CACTL1 = 0; // Disable Comp_A
```
Lowering System Power

My MCU is only operating a portion of the time, is there a way to lower the overall power consumption of my system?

Using a LDO with a programmable output voltage yields lower power consumption when your MCU is in an idle state.

Using a LDO with a programmable output voltage yields lower power consumption when your MCU is in an idle state.

Increasing Power Efficiency

I want higher efficiency than LDOs, but DCDC’s are more complicated, right?

DCDC Converters with Integrated FETs have a low external component count and dramatically reduce complexity with much greater efficiency.

Translating to Real World Applications:
If LDO eff = 66% & DCDC eff = 90% : DCDC allows your battery to last ~36% longer

Unused pins ...
Terminate Unused Pins

- Unused port pins Px.0 – Px.7
  - Set as output direction to avoid floating gate current
- XT2IN, XT2OUT?
- See the last page of chapter 2 in the user's guide

Ultra Low Power Principles

- Maximize the time in LPM3
- Use interrupts to control program flow
- Replace software with peripherals
- Power manage external devices
- Configure unused pins properly
- Efficient code makes a difference
- Even wall powered devices can be "greener"
- Every unnecessary instruction executed is a portion of the battery wasted that will never return
Lab 3 – Ultra-Low Power in Practice

We’re going to measure the power saving effect of using LPM3 mode.

Lab3: Ultra Low-Power In Practice

The code from lab 2 has been converted to use LPM3 instead of the while(1) loop.

Using an ammeter, measure the current through the PWR1 jumper.
**Module Topics**

**Hardware list:**
- WinXP PC
- MSP-FET430UIF
- USB cable
- JTAG ribbon cable
- MSP430FG461x/F28xx Experimenters Board with batteries
- Digital Multimeter
- Jumpers
- Two AAA Batteries

**Software list:**
- IAR Kickstart for MSP430 version 4.21B
- Code Composer Studio 4.1
- Labs
- Additional pdf documentation
- Adobe™ Reader
IAR Kickstart Procedure

The C code from the previous lab has been modified to use LPM3 mode instead of the while(1) loop. We’ll measure the current draw of both labs.

Lab3 Baseline

1. Set up the Hardware

Remove the PWR1 jumper from the Experimenter’s Board and place it over a nearby pin so you won’t lose it.

Make sure the two AAA batteries are in place and connect the BATT jumper to power the board.

Hook up the positive lead of the multimeter to the right-hand PWR1 pin and the negative lead to the left-hand PWR1 pin. Make sure the leads are connected to the proper jacks on the multimeter. Place the multimeter in the lowest milliamp measurement setting and turn it on.

2. Run the Software

Your Lab2 software should still be loaded in the F4618/9 (as well as the Lab1 code in the F2013). If for some reason the Lab2 code is not running, use the steps in Lab2 to reload and run it. Remove the JTAG cable from the FG4618/9 debug port. You may have to remove and replace the BATT jumper to get the MSP430 to boot properly. Press S1 a couple times to verify that the software is functioning.

3. Measure the current

Fill in the blanks in the chart below for Lab2 with LED3 on and off.

<table>
<thead>
<tr>
<th>Code used</th>
<th>LED Off (mA)</th>
<th>LED On (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lab3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lab3 using LPM3

4. Start up IAR Kickstart

Start up IAR Kickstart. When prompted, load the Lab2 workspace. The Lab2 code is probably visible in the editor window. Close the editor by clicking the tiny, little X in the upper right-hand corner of the editor window (not the one that closes IAR Embedded Workbench).

5. Swap out the Source Files

Right-click on Lab2_exercise.c in the Workspace window and select Remove. When prompted whether or not you are sure, click Yes.

On the menu bar, click Project ➔ Add Files. Navigate to C:\MSP430\IAR Labs\Lab3 and select Lab3_solution.c. Click Open.
6. **Inspect the Modified Code**

Double-click on `Lab3_solution.c` in the Workspace window to open it in the editor. Take a moment to inspect the `Lab3_solution` code. Note the configuration of unused pins in the initialization as well as the use of LPM3 in the `while(1)` loop. The `while(1)` loop itself has been altered somewhat to decrease power. Note also the ISR code changes.

7. **Build, Download and Run**

Replace the JTAG cable in the FG4618/9 debug port. Click the **Debug** button to build and download the code to the Experimenter’s Board. Correct any errors you may find. When you’ve successfully downloaded the code to the board, Click the **Stop Debugging** button in *IAR Embedded Workbench* and remove the JTAG cable from the FG4618/9 debug port. You may have to remove and replace the BATT jumper to get the MSP430 to boot properly. Press S1 a couple times to verify that the software is functioning.

8. **Measure the Lab3 current**

Fill in the remaining cells in the table in step 3.

9. **Analysis**

We made the same measurements, and here’s what we got:

<table>
<thead>
<tr>
<th>Code used</th>
<th>LED Off (mA)</th>
<th>LED On (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab2</td>
<td>0.6</td>
<td>2.7</td>
</tr>
<tr>
<td>Lab3</td>
<td>0.0</td>
<td>2.1</td>
</tr>
</tbody>
</table>

Obviously, the current for Lab3 with the LED off was below the measurement abilities of the meter we were using. Subsequent measurements with a better (more expensive) multimeter showed that the current was 1.5uA. That’s a current reduction of about 97%.

**Shut Down**

10. **Shut Down**

Turn off the multimeter and remove the leads from the PWR1 pins. Replace the PWR1 jumper. Remove the BATT jumper and place it over one pin for safekeeping. Shut down *IAR Embedded Workbench*. When prompted to save the project, click No. Replace the JTAG cable in the FG4618/9 debug port.
11. Some further questions

Why were the I/Os configured as they were?

Why was LPM3 used?

Look in the header file to see how LPM3_bits is defined

What further low-power improvements could be made?

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

IAR Kickstart users ... You’re done.
Proceed to the review questions on page 2-26.
*** Where is my flying car? ***
Code Composer Studio 4.1 Procedure

The C code from the previous lab has been modified to use LPM3 mode instead of the while(1) loop. We’ll measure the current draw of both labs.

Lab3 Baseline

1. Set up the Hardware

Remove the PWR1 jumper from the Experimenter’s Board and place it over a nearby pin so you won’t lose it.

Make sure the two AAA batteries are in place and connect the BATT jumper to power the board.

Hook up the positive lead of the multimeter to the right-hand PWR1 pin and the negative lead to the left-hand PWR1 pin. Make sure the leads are connected to the proper jacks on the multimeter. Place the multimeter in the lowest milliamp measurement setting and turn it on.

2. Run the Software

Your Lab2 software should still be loaded in the F4618/9 (as well as the Lab1 code in the F2013). If for some reason the Lab2 code is not running, use the steps in Lab2 to reload and run it. Remove the JTAG cable from the FG4618/9 debug port. You may have to remove and replace the BATT jumper to get the MSP430 to boot properly. Press S1 a couple times to verify that the software is functioning.

3. Measure the current

Fill in the blanks in the chart below for Lab2 with LED3 on and off.

<table>
<thead>
<tr>
<th>Code used</th>
<th>LED Off (mA)</th>
<th>LED On (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lab3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lab3 using LPM3

4. Start up CCS

Start up CCS. When prompted, select the Lab2 workspace. If CCS opens in the debug perspective, click on C/C++ on the upper right of the menu bar to return to the editing perspective. The Lab2 code is probably visible in the editor window. Close the editor window by clicking the X in the Lab2_exercise.c tab.
5. **Swap out the Source Files**

Right-click on `Lab2_exercise.c` in the Project pane and select **Delete**. When prompted whether or not you are sure, click **Yes**.

On the menu bar, click **Project → Add Files to Active Project …**  Navigate to \C:\MSP430\CCS Labs\Lab3 and select `Lab3_solution.c`. Click **Open**.

6. **Inspect the Modified Code**

Double-click on `Lab3_solution.c` in the Project pane to open it in the editor. Take a moment to inspect the `Lab3_solution` code. Note the configuration of unused pins in the initialization as well as the use of LPM3 in the `while(1)` loop. The `while(1)` loop itself has been altered somewhat to decrease power. Note also the ISR code changes.

7. **Build, Download and Run**

Make sure that the JTAG cable in the FG4618/9 debug port. Click the **Debug Launch** button to build and download the code to the Experimenter’s Board. Correct any errors you may find.

When you’ve successfully downloaded the code to the board, Click the **Terminate All** button in **CCS** and **remove** the JTAG cable from the FG4618/9 debug port. You may have to **remove and replace** the BATT jumper to get the MSP430 to boot properly. **Press S1** a couple times to verify that the software is functioning.

8. **Measure the Lab3 current**

Fill in the remaining cells in the table in step 3.

9. **Analysis**

We made the same measurements, and here’s what we got:

<table>
<thead>
<tr>
<th>Code used</th>
<th>LED Off (mA)</th>
<th>LED On (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab2</td>
<td>0.6</td>
<td>2.7</td>
</tr>
<tr>
<td>Lab3</td>
<td>0.0</td>
<td>2.1</td>
</tr>
</tbody>
</table>

Obviously, the current for Lab3 with the LED off was below the measurement abilities of the meter we were using. Subsequent measurements with a better (more expensive) multimeter showed that the current was 1.5µA. That’s a current reduction of about 97%.

**Shut Down**

10. **Shut Down**

Turn off the multimeter and remove the leads from the PWR1 pins. Replace the PWR1 jumper.

Remove the BATT jumper and place it over one pin for safekeeping.

Shut down Code Composer Studio.

Replace the JTAG cable in the FG4618/9 debug port.
11. Some further questions

Why were the I/Os configured as they were?

Why was LPM3 used?

Look in the header file to see how LPM3_bits is defined

What further low-power improvements could be made?

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

You're done
Review Questions

Review

- To minimize power consumption, you should maximize your time in what LPM mode?
- Why are unused pins set as outputs?
- You should control program flow with ...
- Most MSP430 designs utilize a ________ crystal.

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

Introduction

In this section we’ll take a look at the MSP430 analog peripherals. It’s not possible in this limited amount of time to give you a complete overview of the possible analog inputs, but hopefully this introduction will guide you in the right direction.

Objectives

- Comparators
- ADC10 & 12
- SD16 & SD16A
- DAC12
- DTC
- Timer triggers
- Lab
Module Topics

*** This page intentionally left blank ***
Module Topics

Analog Peripherals

Module Topics
ADC Selection
Comparators
ADC10 & ADC12
Conversion Memory and Control
ADC10 DTC
Timer Triggers
SD16
SD16A
SD16 & SD16A Input Range
DAC12

Lab 4 – Using the ADC12
Hardware list:
Software list:

IAR Kickstart Procedure
Start Up
Add Source File
Complete the Code
Test Your Work
Shut Down

Code Composer Studio 4.1 Procedure
Start Up
Add Source File
Complete the Code
Test Your Work
Shut Down

Review Questions
*** Blankety, blank ***
ADC Selection

MSP430 ADC Selection

- Voltage range to be measured?
- Max frequency for AIN?
- How much resolution?
- Differential inputs?
- Reference range?
- Multiple channels?

Comparators

Analog Comparators

- ~100nA operation (Comp_B)
- Hysteresis generator (B)
- Input multiplexer
- Reference generator
- Low-pass filter
- Battery detect
- Interrupt source
- Timer_A capture
- Multiplexer short for sample-and-hold
**Module Topics**

**ADC10 & ADC12**

**10- and 12-bit ADCs**

- 10-bit & 12-bit ADCs
- 200ksps+
- Autoscan
  - Single
  - Sequence
  - Repeat-single
  - Repeat-sequence
- Internal/External reference
- TA SOC triggers
- Data Transfer Controller (DTC)
- DMA Enabled

**Conversion Memory and Control**

**ADC12 Conversion Memory & Control**

<table>
<thead>
<tr>
<th>Memory Registers</th>
<th>Control Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC12MEM0</td>
<td>EOS0</td>
</tr>
<tr>
<td>ADC12MEM1</td>
<td>EOS1</td>
</tr>
<tr>
<td>ADC12MEM15</td>
<td>EOS15</td>
</tr>
</tbody>
</table>

- EOSx – End of Sequence
- SREFx – Reference Selection
- INCHx – Input Channel Selection
- Each location interrupt capable
- Each location DMA enabled

DTC...
ADC10 DTC

ADC10 Direct Transfer Controller (DTC)

```c
// Software
Res[pRes++] = ADC10MEM;
ADC10CTL0 &= ~ENC;
if (pRes < NR_CONV)
{
    CurrINCH++;
    if (CurrINCH == 3)
        CurrINCH = 0;
    ADC10CTL1 &= ~INCH_3;
    ADC10CTL1 |= CurrINCH;
    ADC10CTL0 |= ENC+ADC10SC;
}
```

70 cycles/Sample

Fully Automatic

Other automated conversion methods offer similar benefits

Timer Triggers ...

Timer Triggers

Timer Triggers – Low-Power

```c
// Interrupt
; MSP430 ISR to start conversion
BIS  #ADC12SC,&ADC12CTL0 ; Start conversion
RETI ; Return
;   6  5  16
```

Timer triggered interrupts – no software wait loops
**SD16**

- 16-bit Sigma Delta ADC
- Differential inputs
- 4.096ksps
- 85dB SINAD
- 32x PGA
- 18ppm 1.2V ref
- Temp sensor
- Battery input

**SD16A**

- 16-bit, S-?, 4ksps
- One converter, one channel
- Up to 8 muxed differential inputs
- Independent Programmable Gain Amplifier (PGA)
- High impedance input buffer*
- Internal/External Reference
- Up to 1024x Over Sampling Rate (OSR)
- Optional low-power conversion
- \( V_{CC} \) measurement
- Input Range: +/- 600mV

*Buffer not in 'F20x3 devices
**SD16 & SD16A Input Range**

**SD16/SD16A Input Range**

- **What is VREF?**
- **What is the PGA setting?**

\[
V_{FSR} = \frac{V_{ref}}{GAIN \cdot PGA}
\]

- Applies to all inputs & modes
- **0V = Vss (SD16), 0V = relative (SD16A)**

**DAC12**

- **12-bit monotonic**
- **8/12-bit voltage output**
- **Programmable settling time versus power**
- **Internal/External reference**
- **Binary or 2's compliment**
- **Self-calibration**
- **Group sync load**
- **DMA enabled**

---

*MSP430 One Day Workshop - Analog Peripherals*
Module Topics

*** www.this-page-intentionally-left-blank.org   no kidding***
Lab 4 – Using the ADC12

In this lab we’ll configure and use the ADC12 analog input in the FG4618/9 to measure the temperature from the internal thermistor.

Lab4: Using the ADC12

- Use ADC12 integrated temperature sensor
- Set up ADC12 to perform a single conversion
- Loop continuously, converting to Degrees F and C in software
- Touch the FG4618/9 with a finger to change the temperature
- Open a watch window in the debugger to see the temperature values

Review
**Module Topics**

---

**Hardware list:**
- WinXP PC
- MSP-FET430UIF
- USB cable
- JTAG ribbon cable
- MSP430FG461x/F28xx Experimenter’s Board
- Jumpers

**Software list:**
- IAR Kickstart for MSP430 version 4.21B
- Code Composer Studio 4.1
- Labs
- Additional pdf documentation
- Adobe™ Reader

---
IAR Kickstart Procedure

In this lab we’ll configure and use the ADC12 analog input in the FG4618/9 to measure the temperature from the internal thermistor. Touching the MSP430 will change the temperature enough to measure it, calculate it and place it in a memory for observation.

Start Up

1. **JTAG**

Assure that the JTAG interface is connected to the FG4618/9 debug port.

2. **New Workspace, New Project**

   **Start up IAR Kickstart** and **create** a new workspace, **Create** a new project named **Lab4** and save it in the **C:\MSP430\IAR Labs\Lab4** folder.

3. **Configure the Project Options**

   **Target device** = **MSP430FG4618** or **MSP430FG4619**

   **Driver** = **FET Debugger**

   **FET Debugger** = **Texas Instrument USB-IF**

Add Source File

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

   Add **Lab4_exercise.c** from the **C:\MSP430\IAR Labs\Lab4** folder.

Complete the Code

The following lab steps will walk you through filling in the blanks in the code as shown on the facing page. You’ll want to **open** the **MSP430x4xx Family User’s Guide (slau056g.pdf)**, as well as the **MSP430FG4618/9 datasheet (msp430fg4618.pdf or msp430fg4619.pdf)**. We’re also going to need to look at the standard definitions in the header file:

**C:\Program Files\IAR Systems\Embedded Workbench 5.0\430\inc\msp430xG46x.h**

**Open** that file in the **IAR Kickstart** editor.

If you want to take the easy way out, you’ll find the completed code in the Addendum chapter at the end of this workbook or you can peruse the solution file in the Lab4 folder.

If you’re a glutton for punishment, ignore the following steps and do it on your own. No one’s forcing you to do it our way!
#include "msp430xG46x.h"

volatile unsigned int i;
volatile unsigned int ADCresult;
volatile unsigned long int DegC, DegF;

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
    ADC12CTL0 = _________________; // Turn ADC on, ref on. Ref = 2.5V,
                                    // Set sampling time
                                    // Channel A10, Vref+ & AVss
    ADC12CTL1 = _________________; // Use sampling timer
    ADC12MCTL0 = ________________;    // Channel A10, Vref+ & AVss
    ADC12IE = 0x01;                                     // Enable ADC12IFG.0
    for (i = 0; i < 0x3600; i++);                      // Delay for reference start-up
    ADC12CTL0 |= ENC;                              // Enable conversions
    __enable_interrupt();                            // Enable interrupts
    while(1)
    {
        ADC12CTL0 |= ________________;                   // Start conversion
        __bis_SR_register(LPM0_bits); // Delay for reference start-up
        while(1)
        {
            ADC12CTL0 |= ________________; // Start conversion
            __bis_SR_register(LPM0_bits); // Enter LPM0
        }
        // DegC = (Vsensor - 986mV)/3.55mV
        // Vsensor = (Vref)(ADCresult)/4095
        // DegC -> ((ADCresult - 1615)*704)/4095
        DegC = ((((long)ADCresult-1615)*704)/4095);        // Calculate DegF
        DegF = ((DegC * 9/5)+32); // Calculate DegF
        __no_operation();                                 // SET BREAKPOINT HERE
    }

#pragma vector=ADC12_VECTOR
__interrupt void ADC12ISR(void)
{
    ADCresult = ADC12MEM0; // Move results, IFG is cleared
    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}
5. ADC12CTL0 = __________________________;

Search slau056f.pdf for ADC12CTL0. Somewhere in there you’ll find the bit field layout for the register. Search the header file for the same thing. Under about the fourth occurrence you’ll see the definitions for the individual bit fields.

Look around and find the following definitions:

**Turn ADC12 on:** ______________

**Turn ADC12 reference on:** ______________

**Set the reference to 2.5V:** ______________

**Set the sampling time:** ______________

This last one is a little harder than the first three. First, we need to find out how fast we can sample the temperature sensor. Search the msp430fg4618.pdf or msp430fg4619.pdf datasheet for \( t_{\text{SENSOR}} \) and you’ll see the following:

<table>
<thead>
<tr>
<th>( t_{\text{SENSOR}} ) (sample)</th>
<th>Sample time required if channel 10 is selected (see Note 3)</th>
<th>ADC12ON = 1, IINCH = 0Ah, Error of conversion result ≤ 1 LSB</th>
<th>( 2.2 \text{ V} )</th>
<th>30</th>
<th>( 3 \text{ V} )</th>
<th>30</th>
<th>( \mu\text{s} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The ADC12 in this lab is set up to use the ADC12OSC as the clock source, so search the datasheet for \( f_{\text{ADC12OSC}} \) and find the following:

<table>
<thead>
<tr>
<th>( f_{\text{ADC12OSC}} )</th>
<th>Internal ADC12 oscillator</th>
<th>( f_{\text{ADC12DIV}=0, f_{\text{ADC12CLK}=f_{\text{ADC12OSC}}} )</th>
<th>( V_{\text{CC}} = 2.2 \text{ V} / 3 \text{ V} )</th>
<th>3.7</th>
<th>5</th>
<th>6.3</th>
<th>MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

 So, we need to make sure that the sampling timer uses enough clock cycles in the sample period to guarantee we meet the 30us sampling time required by the temperature sensor. **Calculate** the clock cycles needed and **select** a sampling time that has at least that many cycles.

Take all those definitions, put + signs in between them and **type** them in the proper blank. By the way, the order doesn’t matter since the definitions are all 16-bits.

6. ADC12CTL1 = ________________;

This one’s easy. Look in the header file for ADC12CTL1 and find the correct mode setting for the sample/hold field. Check the datasheet too, if necessary.

7. ADC12MCTL0 = _________________;

You should have it down by now, but this time search the header file for ADC12CTLx. Look in the definitions for **Input Channel 10**. You also have to select \( V_{\text{REF+}} \) using the SREFx field. A quick look at the mux near the top of the ADC12 block diagram in spau056g.pdf will give you a clue which one to pick.
8. ADC12CTL0 |= ______________;

Last one. You should find it pretty quickly if you look in the header file under ADC12CTL0.

**Test Your Work**

9. **Build and Download**

You know what to do by now. Correct any errors you may find. When prompted to save your workspace, save it in the Lab4 folder as Lab4.eww.

10. **Set a Breakpoint**

Set a breakpoint on the line with the comment //SET BREAKPOINT HERE (wow, that was tough). If you’ve already looked through the code, you’ll see that this line is right after the temperature calculations are complete.

11. **Set a Watch**

Right click on the DegC or DegF variable in the code right before the breakpoint. Select Add to Watch from the drop down menu. Right now the value should be 0.

12. **Run**

Run the code and it will quickly stop at the breakpoint you set. Observe the temperature in the Watch window on the right of the screen. Keep clicking the Go button while you place your finger on the FG4618/9 and watch the temperature rise.

Unfortunately, we didn’t properly calibrate the temperature before we started, so the temperature isn’t very accurate. But it’s close enough to understand the ADC12 functions.

13. **Additional Information**

Did you notice the line in the initialization with the comment //Delay for reference start-up? The ADC12 module has a shortcoming, in that a 17mS delay is required after initializing the ADC in order for the reference to stabilize. A software loop is a terrible waste of cycles, but in this case we thought it would be simpler from a coding perspective.

If you have the time and the motivation, how about eliminating the loop and using Timer_A to delay those 17mS? Let your instructor know that you’re going to give this a try!

**Shut Down**

14. **Shut Down**

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

IAR Kickstart users … you’re done. Proceed to the review questions on page 3-21.
Code Composer Studio 4.1 Procedure

In this lab, we’ll configure and use the ADC12 analog input in the FG4618/9 to measure the temperature from the internal thermistor. Touching the MSP430 will change the temperature enough to measure it, calculate it and place it in a memory for observation.

Start Up

1. JTAG

Assure that the JTAG interface is connected to the FG4618/9 debug port.

2. New Workspace, New Project

Start up CCS and create a new workspace at C:\MSP430ODW\CCS Labs\Lab4\workspace. Create a new project named Lab4 in the newly created workspace folder. Make sure the:

Project Type = MSP430

Device Variant = MSP430FG4618 or MSP430FG4619

Add Source File

3. Add the source file to the project

Add Lab4_exercise.c from the C:\MSP430\CCS Labs\Lab4 folder.

Complete the Code

The following lab steps will walk you through filling in the blanks in the code as shown on the facing page. You’ll want to open the MSP430x4xx Family User’s Guide (slau056g.pdf), as well as the MSP430FG4618/9 datasheet (msp430fg4618.pdf or msp430fg4619.pdf). We’re also going to need to look at the standard definitions in the msp430xG46x.h header file:

Find the line in the code containing #include “msp430xG46x.h”. Right-click on the line and select Show In, then Outline. In the Outline pane (on the right), double-click on msp430xG46x.h to open that file in the editor.

If you want to take the easy way out, you’ll find the completed code in the Addendum chapter at the end of this workbook or you can peruse the solution file in the Lab4 folder.

If you’re a glutton for punishment, ignore the following steps and do it on your own. No one’s forcing you to do it our way!
```c
#include "msp430xG46x.h"

volatile unsigned int i;
volatile unsigned int ADCresult;
volatile unsigned long int DegC, DegF;

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
    ADC12CTL0 = __________________________; // Turn ADC on, ref on. Ref = 2.5V,
    // Set sampling time
    ADC12CTL1 = _________________;           // Use sampling timer
    ADC12MCTL0 = ________________;    // Select channel A10, Vref+
    ADC12IE = 0x01;                                     // Enable ADC12IFG.0
    for (i = 0; i < 0x3600; i++);                      // Delay for reference start-up
    ADC12CTL0 |= ENC;                              // Enable conversions
    __enable_interrupt();                            // Enable interrupts

    while(1)
    {
        ADC12CTL0 |= ________________;                   // Start conversion
        __bis_SR_register(LPM0_bits); // Enter LPM0

        // DegC = (Vsensor - 986mV)/3.55mV
        // Vsensor = (Vref)(ADCresult)/4095
        // DegC -> ((ADCresult - 1615)*704)/4095
        DegC = ((((long)ADCresult-1615)*704)/4095);
        DegF = ((DegC * 9/5)+32);                    // Calculate DegF
        __no_operation();                                 // SET BREAKPOINT HERE
    }
}

#pragma vector=ADC12_VECTOR
__interrupt void ADC12ISR(void)
{
    ADCresult = ADC12MEM0; // Move results, IFG is cleared
    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}
```
4. \( \text{ADC12CTL0} = \) ________________;

Search `slau056f.pdf` for `ADC12CTL0`. Somewhere in there you’ll find the bit field layout for the register. Search the header file for the same thing. Under about the fourth occurrence you’ll see the definitions for the individual bit fields.

Look around and find the following definitions:

**Turn ADC12 on:** ______________

**Turn ADC12 reference on:** ______________

**Set the reference to 2.5V:** ______________

**Set the sampling time:** ______________

This last one is a little harder than the first three. First, we need to find out how fast we can sample the temperature sensor. Search the `msp430fg4618.pdf` or `msp430fg4619.pdf` datasheet for \( t_{\text{SENSOR}} \) and you’ll see the following:

<table>
<thead>
<tr>
<th>( t_{\text{SENSOR}} ) (sample)</th>
<th>Sample time required if channel 10 is selected (see Note 3)</th>
<th>( \text{ADC12ON} = 1, \text{INCH} = 0 \text{Ah} )</th>
<th>Error of conversion result ±1 LSB</th>
<th>2.2 V</th>
<th>30</th>
<th>3 V</th>
<th>30</th>
<th>( \mu \text{s} )</th>
</tr>
</thead>
</table>

The ADC12 in this lab is set up to use the \( \text{ADC12OSC} \) as the clock source, so search the datasheet for \( f_{\text{ADC12OSC}} \) and find the following:

| \( f_{\text{ADC12OSC}} \) | Internal ADC12 oscillator | \( \text{ADC12DIV=0, } f_{\text{ADC12CLK}}=f_{\text{ADC12OSC}} \) | \( V_{\text{CC}} = 2.2 \text{ V/3 V} \) | 3.7 | 5 | 6.3 | MHz |

So, we need to make sure that the sampling timer uses enough clock cycles in the sample period to guarantee we meet the 30us sampling time required by the temperature sensor. **Calculate** the clock cycles needed and select a sampling time that has at least that many cycles.

Take all those definitions, put + signs in between them and type them in the proper blank. By the way, the order doesn’t matter since the definitions are all 16-bits.

5. \( \text{ADC12CTL1} = \) ________________;

This one’s easy. Look in the header file for `ADC12CTL1` and find the correct mode setting for the sample/hold field. Check the datasheet too, if necessary.

6. \( \text{ADC12MCTL0} = \) ________________;

You should have it down by now, but this time search the header file for `ADC12CTLx`. Look in the definitions for **Input Channel 10**. You also have to select \( V_{\text{REFx}} \) using the \( SREFx \) field. A quick look at the mux near the top of the ADC12 block diagram in `spau056g.pdf` will give you a clue which one to pick.
7. ADC12CTL0 |= _______________; 
Last one. You should find it pretty quickly if you look in the header file under ADC12CTL0.

Test Your Work

8. Build and Download

You know what to do by now. Correct any errors you may find.

9. Set a Breakpoint

Set a breakpoint on the line with the comment //SET BREAKPOINT HERE (wow, that was tough). If you’ve already looked through the code, you’ll see that this line is right after the temperature calculations are complete.

10. Set a Watch

Double-click on the DegC or DegF variable in the code right before the breakpoint. Right-click on the selected variable, then select Add Watch Expression from the drop down menu. You should see a Watch tab in the upper right pane in CCS. If the Watch pane isn’t already open, click on the tab now.

11. Run

Run the code and it will quickly stop at the breakpoint you set. Observe the temperature in the Watch pane. Keep clicking the Run button while you place your finger on the FG4618/9 and watch the temperature rise.

Unfortunately, we didn’t properly calibrate the temperature before we started, so the temperature isn’t very accurate. But it’s close enough to understand the ADC12 functions.

12. Additional Information

Did you notice the line in the initialization with the comment //Delay for reference start-up? The ADC12 module has a shortcoming, in that a 17mS delay is required after initializing the ADC in order for the reference to stabilize. A software loop is a terrible waste of cycles, but in this case we thought it would be simpler from a coding perspective.

If you have the time and the motivation, how about eliminating the loop and using Timer_A to delay those 17mS? Let your instructor know that you’re going to give this a try!

Shut Down

13. Shut Down

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

Code Composer Studio users … you’re done
Review Questions

Review

- What is your lowest power option for triggering an ADC?
- Name the four ADC conversion modes:

- What is the purpose of the DTC?

- ADC10 and ADC12 can sample at what speed?

You can find the answers to these questions in the Addendum section at the end of this workbook.
*** Why can’t we do this outside? ***
Timers

Introduction

In many microprocessors, timers are used for determining simple intervals. The MSP430 timers are significantly more capable. They can be used to generate multiple PWM frequencies, control ADC hardware or even implement a UART port. Let’s learn a bit more about then now.

Objectives

• Timer_A Architecture
• Count modes
• Interrupts
• TAI
• Timer_B differences
• Timer lab
*** This page left blank with malice aforethought ***
Module Topics

Timers........................................................................................................................................................................ 4-1

Module Topics................................................................................................................................................................. 4-3
Timer A ............................................................................................................................................................................. 4-5
Counting Modes .............................................................................................................................................................. 4-5
Interrupts .......................................................................................................................................................................... 4-6
TAIV Handler................................................................................................................................................................. 4-6
PWM Example ................................................................................................................................................................. 4-7
Direct Hardware Control .................................................................................................................................................... 4-7
UART Implementation ..................................................................................................................................................... 4-8
Timer B............................................................................................................................................................................. 4-8

Lab 5 – Timer_A ................................................................................................................................................................. 4-9
Hardware list: ................................................................................................................................................................. 4-10
Software list: ................................................................................................................................................................. 4-10

IAR Kickstart Procedure................................................................................................................................................... 4-11
Start-up ........................................................................................................................................................................... 4-11
Add Source File ............................................................................................................................................................... 4-11
Complete the Code .......................................................................................................................................................... 4-11
Shut Down ...................................................................................................................................................................... 4-14

Code Composer Studio 4.1 Procedure ........................................................................................................................................ 4-15
Start-up ........................................................................................................................................................................... 4-15
Add Source File ............................................................................................................................................................... 4-15
Complete the Code .......................................................................................................................................................... 4-15
Shut Down ...................................................................................................................................................................... 4-18

Review Questions ............................................................................................................................................................ 4-19
*** Blank! Blank! My kingdom for a blank! ***
Timer_A

- Asynchronous 16-Bit timer/counter
- Continuous, up-down, up count modes
- Multiple capture/compare registers
- PWM outputs
- Interrupt vector register for fast decoding
- Can trigger DMA transfer
- On all MSP430s

Counting Modes

Timer_A Counting Modes

- **Stop/Halt**
  - Timer is halted
- **Continuous**
  - Timer continuously counts up
- **Up**
  - Timer counts between 0 and CCR0
- **Up/Down**
  - Timer counts between 0 and CCR0 and 0

CCR – Count Compare Register

Interrupts ...
**Interrupts**

**Timer_A Interrupts**

The Timer_A Capture/Comparison Register 0 Interrupt Flag (TACCR0) generates a single interrupt vector:

![Diagram showing the relationship between TACCR0 CCIFG and TIMERA0 VECTOR]

No handler required

TACCR1, 2 and TA interrupt flags are prioritized and combined using the Timer_A Interrupt Vector Register (TAIV) into another interrupt vector.

![Diagram showing the relationship between TACCR1 CCIFG, TACCR2 CCIFG, and TAIFG]

Your code must contain a handler to determine which Timer_A1 interrupt triggered.

**TAIV Handler**

**TAIV Handler Example**

<table>
<thead>
<tr>
<th>TAIV</th>
<th>Source</th>
<th>TAI Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No interrupt pending</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>TACCR1 CCIFG</td>
<td>02h</td>
</tr>
<tr>
<td>0</td>
<td>TACCR2 CCIFG</td>
<td>04h</td>
</tr>
<tr>
<td>0</td>
<td>Reserved</td>
<td>06h</td>
</tr>
<tr>
<td>0</td>
<td>Reserved</td>
<td>08h</td>
</tr>
<tr>
<td>0</td>
<td>TAIFG</td>
<td>0Ah</td>
</tr>
<tr>
<td>0</td>
<td>Reserved</td>
<td>0Ch</td>
</tr>
<tr>
<td>0</td>
<td>Reserved</td>
<td>0Dh</td>
</tr>
</tbody>
</table>

IAR C code

```
#pragma vector = TIMERA1_VECTOR
__interrupt void TIMERA1_ISR(void)
{
    switch(__even_in_range(TAIV,10))
    {
        case  2 :    // TACCR1 CCIFG
            P1OUT ^= 0x04; break;
        case  4 :    // TACCR2 CCIFG
            P1OUT ^= 0x02; break;
        case 10 :    // TAIFG
            P1OUT ^= 0x01; break;
        }
    }
```

Assembly code

```
add.w &TAIV,PC
0xF814 add.w &TAIV,PC
0xF818 reti
0xF81A jmp 0xF824
0xF81C jmp 0xF82A
0xF81E reti
0xF820 reti
0xF822 jmp 0xF830
0xF824 xor.b #0x4,&P1OUT
0xF828 reti
0xF82A xor.b #0x2,&P1OUT
0xF82E reti
0xF830 xor.b #0x1,&P1OUT
0xF834 reti
```
PWM Example

**Timer_A PWM Example**

- Completely automatic
- Independent frequencies with different duty cycles can be generated for each CCR
- Code examples on the MSP430 website

Direct Hardware Control

**Direct Hardware Control With Timer_A**

Example: ADC12

- TAR
- TACCR1 = 557
- TAIFG: Reference & ADC on
- TACCR1: Ref delay / ADC trigger
- ADC12IFG: Process ADC result Ref/ADC Off

- CPU Active Mode

UART ...
Module Topics

UART Implementation

Low-Overhead UART Implementation

- 100% hardware bit latching and output
- Full speed from LPM3 and LPM4
- Low CPU Overhead
- App Note SLAA078 on web

Timer B

Timer_B Differences

- 8,10,12 or 16-bit timer or counter
- Up to 7 CCRx units available
- Outputs double-buffered for simultaneous loading
- CCRx registers can be grouped for simultaneous updates
- SCCI latch not implemented (no UART function)
- Tri-state function from external pin
- Default Function is identical to Timer_A
Lab 5 – Timer_A

Let’s configure a timer to wake the MSP430 from a low power mode and blink an LED. Granted, that’s a pretty simple task, but the idea here is to learn how to program the timer.

Configure Timer_A on the MSP430FG4618/9 to wake up the CPU and toggle an LED
Module Topics

**Hardware list:**
- WinXP PC
- MSP-FET430UIF
- USB cable
- JTAG ribbon cable
- MSP430FG461x/F28xx Experimenter’s Board
- Jumpers

**Software list:**
- IAR Kickstart for MSP430 version 4.21B
- Code Composer Studio 4.1
- Labs
- Additional pdf documentation
- Adobe™ Reader
IAR Kickstart Procedure

Configure a timer to wake up the CPU from a low power mode and blink an LED … pretty straight-forward.

Start-up

1. **Hardware**

Assure that the debug interface is correctly connected to the PC and the FG4618/9 debug port.

2. **Start IAR**

Start *IAR Kickstart*. Create a new workspace and a new project in the Lab5 folder. Configure the project options as shown earlier.

Add Source File

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

Add *Lab5_exercise.c* from the *C:\MSP430\IAR Labs\Lab5* folder to the project. Double-click on the file in the Project pane to open it for editing.

Complete the Code

Like the previous lab, the next steps will lead you though the process of filling in the four blanks in the code. You should already have the process down, though, so we won’t give you nearly the level of detail as you had in the previous lab.

You’ll probably want to open *slau056g.pdf* and the *msp430xG46x.h* header file.

Again, if you’re lazy and want to skip to the solution, you can either look in the Addendum at the back of this workbook or open up the solution file.
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;    // Stop WDT
    FLL_CTL0 |= XCAP14PF;                        // Configure load caps
    P2DIR |= BIT1;                              // Set P2.1 to output direction
    TACTL = ________________;                  // Clock = ACLK (32768), clear
    TACCTL0 = ____;                            // CCR0 interrupt enabled
    TACCR0 = _______;                           // #counts for 1s
    TACTL |= ____;                             // Setting mode bits starts timer

    _BIS_SR(LPM3_bits + GIE);                 // Enter LPM3 w/ interrupt
}

// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
    P2OUT ^= 0x02;                            // Toggle P2.1 using exclusive-OR
}
4. \( TACTL = \_\_\_\_\_\_\_\_\_\_; \)

Clock = ACLK (32768Hz)
To set the clock source select to ACLK you’re going to need to know how the TASSELx field is configured. That’s enough of a hint …

Clear
Finding the counter clear is pretty easy.

5. \( TACCTL0 = \_\_\_\_\_\_\_\_\_; \)

Enable CCR0 interrupt
Enable the capture/compare interrupt. If you’ve ever been geo-caching, this process is analogous. The GPS will get you close, but then you’ve got to hunt around on your hands and knees for the prize.

6. \( TACCR0 = \_\_\_\_\_\_\_\_\_\_\_; \)

Number of counts for one second
This one takes just a little bit of thought. What’s the clock frequency we’re using to drive the timer? (Hint: We selected it in step 4). How many clock cycles would equal one second? Bear in mind that when the timer rolls over to zero, that is also counted as a tick, so to get \( n \) ticks, you put \( n-1 \) in the CCR0 register.

7. \( TACTL |\_\_\_\_\_\_\_\_\_\_\_\_; \)

Check the User’s Guide and make sure which mode you want the timer to operate in, then find the correct symbol in the header file.

8. **Build, Download and Run**

Try out the code and make sure it works properly. Correct any errors you may have. Observe the LED and verify that it blinks at the proper interval. Feel free to play around with the interval period in the code.
9. A Few More Questions

Here’s a great opportunity to show off your ability to search the User’s Guide. The answers are in the Addendum at the back of this workbook.

Why was TAIE not set in TACTL?

Why were the MCx bits not set initially when TACTL was configured?

Shut Down

10. Shut down

Halt the debugger and shut down IAR Kickstart.

IAR Users … you’re done. Proceed to the review questions on page 4-19.
Code Composer Studio 4.1 Procedure

Configure a timer to wake up the CPU from a low power mode and blink an LED … pretty straight-forward.

Start-up

1. Hardware

Assure that the debug interface is correctly connected to the PC and the FG4618/9 debug port.

2. Start CCS

Start CCS. Create a new workspace in C:\MSP430ODW\CCS Labs\Lab5\workspace and a new project in the folder called Lab5. Configure the project settings as shown earlier.

Add Source File

3. Add the source file to the project

Add Lab5_exercise.c from the C:\MSP430\CCS Labs\Lab5 folder to the project. Double-click on the file in the Project pane to open it for editing.

Complete the Code

Like the previous lab, the next steps will lead you though the process of filling in the four blanks in the code. You should already have the process down, though, so we won’t give you nearly the level of detail as you had in the previous lab.

You’ll probably want to open slau056g.pdf and the msp430xG46x.h header file.

Again, if you’re lazy and want to skip to the solution, you can either look in the Addendum at the back of this workbook or open up the solution file.
```c
#include <msp430xG46x.h>

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;    // Stop WDT
    FLL_CTL0 |= XCAP14PF;        // Configure load caps
    P2DIR |= BIT1;               // Set P2.1 to output direction
    TACTL = ________________;    // Clock = ACLK (32768), clear
    TACCTL0 = ____;              // CCR0 interrupt enabled
    TACCR0 = _______;            // #counts for 1s
    TACTL |= ____;               // Setting mode bits starts timer

    _BIS_SR(LPM3_bits + GIE);    // Enter LPM3 w/ interrupt
}

// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
    P2OUT ^= 0x02;               // Toggle P2.1 using exclusive-OR
}
```
4. TACTL = __________________;

Clock = ACLK (32768Hz)

To set the clock source select to ACLK you’re going to need to know how the TASSELx field is configured. That’s enough of a hint …

Clear

Finding the counter clear is pretty easy.

5. TACCTL0 = ______________;

Enable CCR0 interrupt

Enable the capture/compare interrupt. If you’ve ever been geo-caching, this process is analogous. The GPS will get you close, but then you’ve got to hunt around on your hands and knees for the prize.

6. TACCR0 = ______________;

Number of counts for one second

This one takes just a little bit of thought. What’s the clock frequency we’re using to drive the timer? (Hint: We selected it in step 4). How many clock cycles would equal one second? Bear in mind that when the timer rolls over to zero, that is also counted as a tick, so to get n ticks, you put n-1 in the CCR0 register.

7. TACTL |= ______________;

Check the User’s Guide and make sure which mode you want the timer to operate in, then find the correct symbol in the header file.

8. Build, Download and Run

Try out the code and make sure it works properly. Correct any errors you may have. Observe the LED and verify that it blinks at the proper interval. Feel free to play around with the interval period in the code.
9. A Few More Questions

Here’s a great opportunity to show off your ability to search the User’s Guide. The answers are in the Addendum at the back of this workbook.

Why was TAIE not set in TACTL?

Why were the MCx bits not set initially when TACTL was configured?

Shut Down

10. Shut down

Halt the debugger and shut down Code Composer Studio.

CCS Users … you’re done.
Review Questions

- Name the counting modes.
- What is the TAIV register's purpose?
- In addition to normal timer functions, name some other functions the timer can perform.

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

Introduction

In this module we’ll take a look at the MSP430 communications modules and the protocols that can be implemented over them.

Objectives

- USART
- USCI
- USI
*** I only insert blank pages when the voices tell me to***
Module Topics

Communication ......................................................................................................................................... 5-1

Module Topics................................................................................................................................. 5-3
MSP430 Communication Modules....................................................................................................... 5-5
USI.................................................................................................................................................... 5-5
Data I/O via USI .................................................................................................................................. 5-6
SPI via USI ....................................................................................................................................... 5-6
USART ............................................................................................................................................... 5-7
Baudrate Generator ........................................................................................................................... 5-7
USCI.................................................................................................................................................... 5-8
USCI Initialization Sequence ................................................................................................................ 5-8
USCI Enhanced Features .................................................................................................................... 5-9
USCI Baudrate Generator ................................................................................................................... 5-9

Optional Lab 6 – I2C Communications ................................................................................................. 5-11
Hardware list: ..................................................................................................................................... 5-12
Software list: ................................................................................................................................. 5-12

IAR Kickstart Procedure ....................................................................................................................... 5-13
Set up the Hardware ............................................................................................................................ 5-13
Load the MSP430F2013 Software ....................................................................................................... 5-13
Set up for the FG4618/9 ....................................................................................................................... 5-13
Complete the I2C Master Code ........................................................................................................... 5-14
Build/Load/Run/Test ......................................................................................................................... 5-15
Shut Down .......................................................................................................................................... 5-15

Code Composer Studio Procedure ......................................................................................................... 5-17
Set up the Hardware ............................................................................................................................. 5-17
Load the MSP430F2013 Software ....................................................................................................... 5-17
Set up for the FG4618/9 ....................................................................................................................... 5-18
Complete the I2C Master Code ........................................................................................................... 5-18
Build/Load/Run/Test ......................................................................................................................... 5-19
Shut Down .......................................................................................................................................... 5-20

Review Questions ............................................................................................................................... 5-21
*** For security reasons this page must be left blank ***
# MSP430 Communication Modules

## MSP430 Communication Modules

<table>
<thead>
<tr>
<th>UART</th>
<th>USCI</th>
<th>USI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Universal Synch/Async Receiver/Transmitter</td>
<td>Universal Serial Communication Interface</td>
<td>Universal Serial Interface</td>
</tr>
<tr>
<td>One modulator</td>
<td>Two modulators; supports n/16 timings</td>
<td>-</td>
</tr>
<tr>
<td>- Auto baud rate detection</td>
<td>- Master and slave modes</td>
<td></td>
</tr>
<tr>
<td>- IrDA encoder &amp; decoder</td>
<td>- Master and slave modes</td>
<td></td>
</tr>
<tr>
<td>- Simultaneous USCI_A and USCI_B (2 channels)</td>
<td>- Up to 400kbps</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SPI</th>
<th>Two SPI (one each on USCI_A and USCI_B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>One SPI channel</td>
<td>- Master and slave modes</td>
</tr>
<tr>
<td>- Master and slave modes</td>
<td>- 3 and 4 wire modes</td>
</tr>
<tr>
<td>(on '15x/'16x only)</td>
<td>- Simplified interrupt usage</td>
</tr>
<tr>
<td>- Master and slave modes</td>
<td>- Master and slave modes</td>
</tr>
<tr>
<td>- Up to 400kbps</td>
<td>- SW state machine needed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I2C</th>
</tr>
</thead>
<tbody>
<tr>
<td>One modulator</td>
</tr>
<tr>
<td>- Master and slave modes</td>
</tr>
<tr>
<td>- 3 and 4 wire modes</td>
</tr>
</tbody>
</table>

## USI

- MSP430x20xx devices
- Variable length shift register
- Supports I2C
  - START/STOP detection
  - SCL held after START
  - SCL held after counter overflow
  - Arbitration lost detection
- Supports SPI
  - 8/16-bit Shift Register
  - MSB/LSB first
- Flexible Clocking
- Interrupt Driven

---

**Data I/O ...**
Data I/O via USI

**USI for Data I/O**

- Data shift register: up to 16 bits supported
- Number of bits transmitted and received is controlled by a bit counter
- Transmit and Receive is simultaneous
- Data I/O is user-defined: MSB or LSB first
- Bit counter automatically stops clocking after last bit & sets flag
- No data buffering needed

**SPI Implementation ...**

SPI via USI

**USI Reduces CPU Load for SPI**

```
//Shift16_inout_Software
SR = DATA;
for (CNT=0x10;CNT>0;CNT--)
{
    P2OUT &= ~SDO;
    if (SR & 0x8000)
        P2OUT |= SDO;
    SR = SR << 1;
    if (P2IN & SDIN)
        SR |= 0x01;
    P2OUT |= SCLK;
    P2OUT &= ~SCLK;
}
```

```
//Shift16_inout_USI
USISR |= DATA;
USICNT |= 0x10;
```

10 Cycles

- I2C Slave has as little as 4us from clock edge to data
- Traditional software-only solution allows time for little else
- USI hardware enables practical and compliant I2C
- Code on MSP430 website
Module Topics

**USART**

- Ultra-Low Power Support:
  - Auto-Start from any Low-Power Mode
- UART or SPI Mode (I2C on ‘F15x’/‘F16x only)
- Double Buffered TX/RX
- Baudrate Generator
- DMA enabled
- Error Detection

Recommended USART initialization/re-configuration process is shown in your workbook.

**Baudrate Generator**

9600 baud:
ACLK = 32768 Hz
Prescaler = 32768Hz/9600baud = 3.41
UsxBR1 | UsxBR0 | UsxMCTL = 00h | 03h | 4Ah

\[ \text{BITCLK} = \frac{\text{ACLK}}{4} \times \frac{1}{3} \times \frac{1}{4} \times \frac{1}{3} \times \frac{1}{4} \times \frac{1}{3} \ldots \]

Content of UsxMCTL is the modulation pattern
USCI

**USCI Initialization Sequence**

**Note: Initializing or Re-Configuring the USCI Module**

The recommended USCI initialization/re-configuration process is:

1) Set UCSWRST (BIS.B #UCSWRST, &UCAxCTL1)

2) Initialize all USCI registers with UCSWRST = 1 (including UCAxCTL1)

3) Configure ports.

4) Clear UCSWRST via software (BIC.B #UCSWRST, &UCAxCTL1)

5) Enable interrupts (optional) via UCAxRXIE and/or UCAxTXIE
USCI Enhanced Features

- New standard MSP430 serial interface
- Auto clock start from any LPMx
- Two independent communication blocks
- Asynchronous communication modes
  - UART standard and multiprocessor protocols
  - UART with automatic Baud rate detection (LIN support)
  - Two modulators support n/16 bit timing
  - IrDA bit shaping encoder and decoder
- Synchronous communication modes
  - SPI (Master & Slave modes, 3 & 4 wire)
  - I2C (Master & Slave modes)

USCI Baudrate Generator

- Oversampling Baud Rate Generation
- Two Modulators:
  - UCBRSx and UCBRFx select modulation pattern
- RX sampled using BITCLK16
*** War and Peace started this way ***
Optional Lab 6 – I2C Communications

This lab should be attempted if time permits during the class or as a take-home project for the student.

The MSP430F2013 is used to measure the temperature. It then transmits the result to the MSP430FG4618/9 via the I2C connection on the USCI port. The MSP430FG4618/9 will then determine if a preset difference has been reached, at which point it will light LED4. The MSP430F2013 will also flash LED3 each communication cycle. In this I2C implementation, the MSP430F2013 will be the slave and the MSP430FG4618/9 will be the master.

Optional Lab6: I2C Communication

Complete this lab if time permits or as a take-home project

- I2C Master on the FG4618/9 USCI receives the 16-bit conversion result from the F2013 Slave
- F2013 slave flashes LED3 with each communication cycle
- When the result-to-result difference exceeds a preset amount, the FG4618/9 I2C Master turns LED4 on
Hardware list:

- WinXP PC
- MSP-FET430UIF
- USB cable
- JTAG ribbon cable
- MSP430FG461x/F28xx Experimenter’s Board
- Jumpers

Software list:

- IAR Kickstart for MSP430 version 4.21B
- Code Composer Studio 4.1
- Labs
- Additional pdf documentation
- Adobe™ Reader
IAR Kickstart Procedure

In this lab, you will complete an I2C data link between the two MSP430s on the Experimenter’s Board. Our tasks will be to:

- Load ready-to-use USI I2C slave code on the MSP430F2013 (slave address = 0x48)
- Complete partial MSP430FG4618/9 USCI_B I2C Master Receiver code

Set up the Hardware

1. JTAG

The first thing we’re going to do is to load the ready-to-use I2C slave code into the MSP430F2013. **Remove** the JTAG ribbon cable from the MSP430FG4618/9 debug port and place it in the MSP430F2013 debug port.

Load the MSP430F2013 Software

**Note: Please do not load the MSP430F2013 code into the MSP430FG4618/9!**

2. **Load the I2C Software into the MSP430F2013**

Open **IAR Kickstart**, create a new workspace and project called **Lab6** in the **IAR Labs\Lab6** folder. Don’t forget to set the project options with the target device being the **MSP430F2013**.

Add **Lab6_2013_solution.c** to the project and **build/load** it to the **MSP430F2013**. Feel free to open the code in the editor and take a look at it.

Click the **Go** button to start the MSP430F2013 I2C slave code running. You’ll probably have no visual indication that the code is running. **Exit** the debugger by clicking the **Stop Debugging** button. Now you should be looking at the editor window in **IAR Kickstart**.

Set up for the FG4618/9

3. **Set up for the MSP430FG4618/9 I2C Master Code**

Swap the JTAG connector to the MSP430FG4618/9 debug port.

Close the **Lab6_2013_solution.c** code in the editor window (if you still have it open). **Right-click** on the file in the Workspace window and select **Remove**, then click **Yes**.

Add **Lab6_4618_exercise.c** to the project. **Change** the project option target device to the **MSP430FG4618** or **MSP430FG4619**.

Open the source file in the editor and feel free to look around in it.
Complete the I2C Master Code

Let’s fill in the blanks one at the time in the code extract below. Lazy folks can reference the solutions …

```assembly
P3SEL |= 0x06;                           // Assign I2C pins to USCI_B0
UCB0CTL1 |= _______;                     // Enable SW reset (why?)
UCB0CTL0 = _________________________;  // I2C Master, synchronous mode
UCB0CTL1 = __________________;          // Use SMCLK, keep UCSWRST set
UCB0BR0 = 11;                            // fSCL = SMCLK/11 = 95.3kHz
UCB0BR1 = 0;
UCB0I2CSA = 0x48;                        // Set slave address
UCB0CTL1 &= ~______;                     // Clear SW reset, resume operation
UCB0I2CIE |= UCNACKIE;                  // Interrupt on slave Nack
IE2 |= UCB0RXIE;                         // Enable RX interrupt
```

4. UCB0CTL1 |= ______;

It’s pretty easy to find the USCI software reset in the UCB0CTL1 section of the header file. Why do you think the USCI should be in reset while you’re programming its bits? Gee, that’s a tough one …

5. UCB0CTL0 = ________________________;

I2C Master

Look for the master mode select in the UCB0CTL0 section.

Synchronous mode

A quick look at the Initialization and Reset chapter of the USCI/I2C section will tell you that the UCMODEx bits must be set properly to be in I2C mode. In addition, you must select the synchronous mode.

6. UCB0CTL1 = _________________;

Clock source

You must select the appropriate USCI clock source to use SMCLK. In this case, that’s source 2. Verify that in the User’s Guide.

Keep UCSWRST set

Make sure the USCI software reset stays set,
7.  UCB0CTL1 &= ~_______
This one’s easy. Now that everything is all set up, you can clear the USCI software reset.

**Build/Load/Run/Test**

8. **Build/Load/Run**

Compile the code, download it to the MSP430FG4618/9 and run it. LED3 (next to the MSP430F2013 debug port) should be blinking about once every 2 seconds.

9. **Test the Code**

With the code running, place your fingertip on the MSP430F2013 device (the little one next to the MSP430F2013 debug port). After a few seconds, LED4 (underneath the LCD display) should light. Remove your finger and the LED will quickly go off. Look around in the MSP430FG4618/9 code to see what the threshold is to light the LED. Change it if you like.

**Shut Down**

10. **Shut down**

Shut down *IAR Kickstart*. Disconnect the JTAG debug interface from both the Experimenter’s Board and the PC.

IAR Users … you’re done. Proceed to the review questions on page 5-21.
*** It’s about time for a refreshment, I think ***
Code Composer Studio 4.1 Procedure

In this lab, you will complete an I2C data link between the two MSP430s on the Experimenter’s Board. Our tasks will be to:

- Load ready-to-use USI I2C slave code on the MSP430F2013 (slave address = 0x48)
- Complete partial MSP430FG4618/9 USCI_B I2C Master Receiver code

Set up the Hardware

1. JTAG

The first thing we’re going to do is to load the ready-to-use I2C slave code into the MSP430F2013. Remove the JTAG ribbon cable from the MSP430FG4618/9 debug port and place it in the MSP430F2013 debug port.

Load the MSP430F2013 Software

| Note: Please do not load the MSP430F2013 code into the MSP430FG4618/9! |

2. Load the I2C Software into the MSP430F2013

Open CCS, create a new workspace in the CCS Labs\Lab6 folder. Create a new project in that workspace folder called Lab6_2013. Don’t forget to set the project options with the target device being the MSP430F2013.

Add Lab6_2013_solution.c to the project and build/load it to the MSP430F2013. Feel free to open the code in the editor and take a look at it.

Click the Run button to start the MSP430F2013 I2C slave code running. You’ll probably have no visual indication that the code is running. Exit the debugger by clicking the Terminate All button. Now you should be looking at the editor window in Code Composer Studio.
Set up for the FG4618/9

3. Set up for the MSP430FG4618/9 I2C Master Code

Swap the JTAG connector to the MSP430FG4618/9 debug port.

Close the Lab6_2013_solution.c code in the editor window (if you still have it open). We could delete the source file from the project, but …

CAUTION: Eclipse (the editor used here) actually deletes the source file from the workspace folder. In our case, that’s not an issue. When we added our source file, Eclipse made a copy of our source file in the workspace folder. But if you store your original source files in the workspace folder, they will be deleted in this process. Consider yourself warned.

Instead, let’s do something a little more interesting. Create a new project in this workspace called Lab6_4618 (I know, that’s not a very imaginative name). Don’t forget to set the project options with the target device being the MSP430FG4618 (or 19). Check this out … now our workspace has two projects in it. Imagine the possibilities.

Lab6_4618 is now the Active Project (notice the project pane). Add Lab6_4618_exercise.c to the project. We can easily switch between projects by right-clicking on the project and selecting Set as Active Project. But, leave Lab6_4618 as the active project now.

Open the Lab6_4618_exercise.c source file in the editor and feel free to look around in it.

Complete the I2C Master Code

Let’s fill in the blanks one at the time in the code extract below. Lazy folks can reference the solutions …

```
P3SEL |= 0x06;                           // Assign I2C pins to USCI_B0
UCB0CTL1 |= _______;                     // Enable SW reset (why?)
UCB0CTL0 = _________________________;  // I2C Master, synchronous mode
UCB0CTL1 = __________________;          // Use SMCLK, keep UCSWRST set
UCB0BR0 = 11;                            // fSCL = SMCLK/11 = 95.3kHz
UCB0BR1 = 0;
UCB0I2CSA = 0x48;                        // Set slave address
UCB0CTL1 &~ _______;                     // Clear SW reset, resume operation
UCB0I2CIE |= UCNACKIE;                   // Interrupt on slave Nack
IE2 |= UCB0RXIE;                         // Enable RX interrupt
```
4. UCB0CTL1 |= ______;

It's pretty easy to find the USCI software reset in the UCB0CTL1 section of the header file. Why do you think the USCI should be in reset while you're programming its bits? Gee, that's a tough one …

5. UCB0CTL0 = ______________________;

I2C Master

Look for the master mode select in the UCB0CTL0 section.

Synchronous mode

A quick look at the Initialization and Reset chapter of the USCI/I2C section will tell you that the UCMODEEx bits must be set properly to be in I2C mode. In addition, you must select the synchronous mode.

6. UCB0CTL1 = _________________;

Clock source

You must select the appropriate USCI clock source to use SMCLK. In this case, that's source 2. Verify that in the User’s Guide.

Keep UCSWRST set

Make sure the USCI software reset stays set.

7. UCB0CTL1 &= ~_______

This one's easy. Now that everything is all set up, you can clear the USCI software reset.

Build/Load/Run/Test

8. Build/Load/Run

Compile the code, download it to the MSP430FG4618/9 and run it. LED3 (next to the MSP430F2013 debug port) should be blinking about once every 2 seconds.

9. Test the Code

With the code running, place your fingertip on the MSP430F2013 device (the little one next to the MSP430F2013 debug port). After a few seconds, LED4 (underneath the LCD display) should light. Remove your finger and the LED will quickly go off. Look around in the MSP430FG4618/9 code to see what the threshold is to light the LED. Change it if you like.
Shut Down

10. Shut down

Shut down *Code Composer Studio*. Disconnect the JTAG debug interface from both the Experimenter’s Board and the PC.

CCS Users … you’re done.
Review Questions

Review

- The new, standard MSP430 serial comm. module is:

- Implementing SPI on the USI or USCI provides a __________ and __________ solution.

- The best place to look for code examples is:

- The best place to find technical documentation is:

You can find the answers to these questions in the Addendum section at the end of this workbook.
*** Relax, it's almost over ***
Wrap-Up

Introduction

It’s been a long day, or at least it’s felt that way. Here are some things not to forget.
*** Oh, the untapped potential of a page left blank. ***
Wrap-up

MSP430
- 16-Bit
- Ultra-low power
- Easy-to-use
- 1k-256kB ISP Flash
- 14-100 pin options
- USART, I2C, Timers
- 10/12/16-bit ADC
- DAC, OP Amp, LCD driver
- Embedded emulation
- + new 5xx High Performance!

Don’t Forget…

Take your workshop handouts home with you
Fill out the evaluation form on line if possible (use paper forms otherwise)
This material is available on-line:

Thank you for attending
Have a safe trip home
Introduction

Here are the answers to all those pesky questions in the workshop, along with the lab solutions.

Objectives

- Module review answers
- Lab Question Answers
- Lab Solutions
*** I was somewhat ambivalent about leaving this page blank. ***
Lab Solutions and Review Answers

Addendum.................................................................................................................................................. 7-1

Lab Solutions and Review Answers........................................................................................................... 7-3
  Introduction Module Review Answers........................................................................................................ 7-5
  Lab1 IAR Code........................................................................................................................................ 7-5
  Lab1 CCS Code..................................................................................................................................... 7-6
  Flash Programming Exercise.................................................................................................................... 7-6
  Lab2 IAR Solution................................................................................................................................ 7-7
  Lab3 Step 11 Answers.............................................................................................................................. 7-8
  Ultra-Low Power Module Review Answers............................................................................................. 7-9
  Lab4 IAR Solution................................................................................................................................ 7-10
  Analog Peripherals Module Review Answers............................................................................................ 7-11
  Lab5 IAR Solution................................................................................................................................ 7-12
  Lab5 Step 9 Answers............................................................................................................................... 7-13
  Timer Module Review Answers................................................................................................................ 7-14
  Communications Module Review Answers.............................................................................................. 7-15
  Optional Lab6 - USCI/SPI Communications IAR Solution ...................................................................... 7-16
*** This page reluctantly left blank ***
Introduction Module Review Answers

**Review**

- How many general purpose registers does the MSP430 have?
  - 12
- What is the purpose of the constant generator?
  - Reduce code size and cycles by automatically generating commonly used constants
- Where is the best resource for MSP430 information?
  - www.ti.com/msp430
- At reset, all I/O pins are set to ...
  - Inputs
- Why should you use standard definitions?
  - Resulting code is easier to read and debug

Lab1 IAR Code

```c
#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
```

Ultra-low Power
Lab Solutions and Review Answers

Lab1 CCS Code

```
.cdecls C,LIST,"msp430x21x1.h" ; Include device header file

.text                              ; 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

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

DW     RESET
END
```

Flash Programming Exercise

\[ f_{FTG} = 476 \text{ kHz} \]

\[ t_{Word} = 30 \]

Time to program a word or byte \[ = \frac{30}{476000} = 63uS \]

Time to randomly program 1024 words \[ = 1024 \times 63uS = 64.5mS \]
Lab2 IAR Solution

```c
#include <msp430xG46x.h>

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;  // Stop WDT
    FLL_CTL0 |= XCAP14PF;       // Configure load caps
    P2DIR = BIT1;                // Set P2.1 to output direction
    P1IES = BIT0;                // H-L transition
    P1IE = BIT0;                 // 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 &= ~BIT0;              // Clear P1IFG
    if ((P1IN & 0x01) == 0)
        P2OUT ^= 0x02;             // Toggle P2.1 using exclusive-OR
}
```
Lab3 Step 11 Answers

Why were the I/Os configured as they were?

Unused I/O must be configured as outputs, otherwise, floating gate current will occur. The outputs were then set to values so as not to contend with other on-board circuitry.

Why was LPM3 used?

No clocks are needed. LPM3 leaves on the 32768Hz running and shuts down all other clocks.

Look in the header file to see how LPM3_bits is defined

SCG1+SCG0+CPUOFF

What further low-power improvements could be made?

LPM4 could be used. A timer could be employed for the pushbutton debounce.
Ultra-Low Power Module Review Answers

Review

- To minimize power consumption, you should maximize your time in what LPM mode?
  LPM3
- Why are unused pins set as outputs?
  To avoid floating gate currents
- You should control program flow with …
  Interrupts
- Most MSP430 designs utilize a ________ crystal.
  32,768 Hz
Lab Solutions and Review Answers

Lab4 IAR Solution

```c
#include "msp430xG46x.h"

volatile unsigned int i;
volatile unsigned int ADCresult;
volatile unsigned long int DegC, DegF;

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;    // Stop watchdog timer
    ADC12CTL0 = ADC12ON + REFON + REF2_5V + SHT0_7;  // Turn ADC on, ref on. Ref = 2.5V,
                                                        // Set sampling time
    ADC12CTL1 = SHP;              // Use sampling timer
    ADC12MCTL0 = INCH_10 + SREF_1; // Select channel A10, Vref+
    ADC12IE = 0x01;                                     // Enable ADC12IFG.0
    for (i = 0; i < 0x3600; i++);                      // Delay for reference start-up
    ADC12CTL0 |= ENC;                              // Enable conversions
    __enable_interrupt();                            // Enable interrupts

    while(1)
    {
        ADC12CTL0 |= ADC12SC;                     // Start conversion
        __bis_SR_register(LPM0_bits);           // Enter LPM0

        // DegC = (Vsensor - 986mV)/3.55mV
        // Vsensor = (Vref)(ADCresult)/4095
        // DegC -> ((ADCresult - 1615)*704)/4095
        DegC = ((((long)ADCresult-1615)*704)/4095);
        DegF = ((DegC * 9/5)+32);                    // Calculate DegF
        __no_operation();                                 // SET BREAKPOINT HERE
    }
}

#pragma vector=ADC12_VECTOR
__interrupt void ADC12ISR(void)
{
    ADCresult = ADC12MEM0;                          // Move results, IFG is cleared
    __bic_SR_register_on_exit(LPM0_bits);        // Exit LPM0
}
```

7 - 10 MSP430 One Day Workshop - Addendum
Analog Peripherals Module Review Answers

Review

- **What is your lowest power option for triggering an ADC?**
  Trigger conversion with a timer.
- **Name the four ADC conversion modes:**
  Single, Sequence, Repeat-single, Repeat-sequence
- **What is the purpose of the DTC?**
  The Direct Transfer Controller moves the conversion result of the ADC10 into any MSP430 memory.
- **ADC10 and ADC12 can sample at what speed?**
  200ksps
Lab5 IAR Solution

```c
#include <msp430xG46x.h>

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;   // Stop WDT
    FLL_CTL0 |= XCAP14PF;                      // Configure load caps
    P2DIR |= BIT1;                              // Set P2.1 to output direction
    TACTL = TASSEL_1 + TACLR;                 // Clock = ACLK (32768), clear
    TACCTL0 = CCIE;                            // CCR0 interrupt enabled
    TACCR0 = 32768-1;                          // #counts for 1s
    TACTL |= MC_1;                              // Setting mode bits starts timer

    _BIS_SR(LPM3_bits + GIE);                  // Enter LPM3 w/ interrupt
}

// Timer A0 interrupt service routine
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
    P2OUT ^= 0x02;                            // Toggle P2.1 using exclusive-OR
}
```
Lab5 Step 9 Answers

Why was TAIE not set in TACTL?

Actually, we didn’t use the interrupt generated when you enable TAIE. Timer_A has several interrupts. TAIE enables an interrupt to occur on overflow. We could have used it, but instead we used the TACCR0 interrupt, which fires when TAR hits the CCR0 value.

Why were the MCx bits not set initially when TACTL was configured?

Setting the MCx bits starts the timer running, so you want all setup to be completed beforehand.
Timer Module Review Answers

Review

- Name the counting modes.
  - Up, Up/Down and Continuous
- What is the TAV register’s purpose?
  - To combine three interrupts into a single interrupt to the CPU. Also acts as an indicator for handler code to determine which interrupt triggered.
- In addition to normal timer functions, name some other functions the timer can perform.
  - ADC12 hardware control, PWM, UART
Communications Module Review Answers

Review

- The new, standard MSP430 serial communication module is:
  the USCI module
- Implementing SPI on the USI or USCI provides a ________
  and _________ solution.
  low-power, low-cycle count
- The best place to look for code examples is:
  the MSP430 website
- The best place to find technical documentation is:
  the MSP430 website

Wrap Up...
Optional Lab6 - USCI/SPI Communications IAR Solution

P3SEL |= 0x06; // Assign I2C pins to USCI_B0
UCB0CTL1 |= UCSWRST; // Enable SW reset (why?)
UCB0CTL0 = UCMST + UCMODE_3 + UCSYNC; // I2C Master, synchronous mode
UCB0CTL1 = UCSSEL_2 + UCSWRST; // Use SMCLK, keep UCSWRST set
UCB0BR0 = 11; // fSCL = SMCLK/11 = 95.3kHz
UCB0BR1 = 0;
UCB0I2CSA = 0x48; // Set slave address
UCB0CTL1 &!= ~UCSWRST; // Clear SW reset, resume operation
UCB0I2CIE |= UCNACKIE; // Interrupt on slave Nack
IE2 |= UCB0RXIE; // Enable RX interrupt