TEXAS INSTRUMENTS

MSP430 4xx One Day Workshop 2010

Student Guide

Revision 3.2
January 2010

TTO

Technical Training
Organization

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.

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.

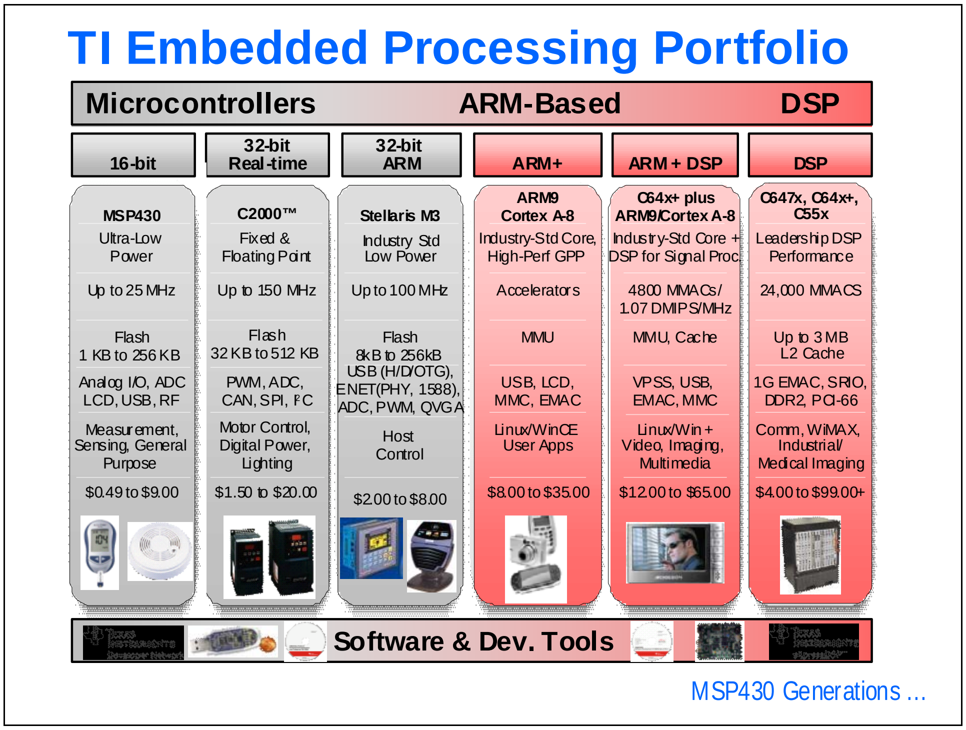
Objectives

- 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

TI Microcontroller Portfolio



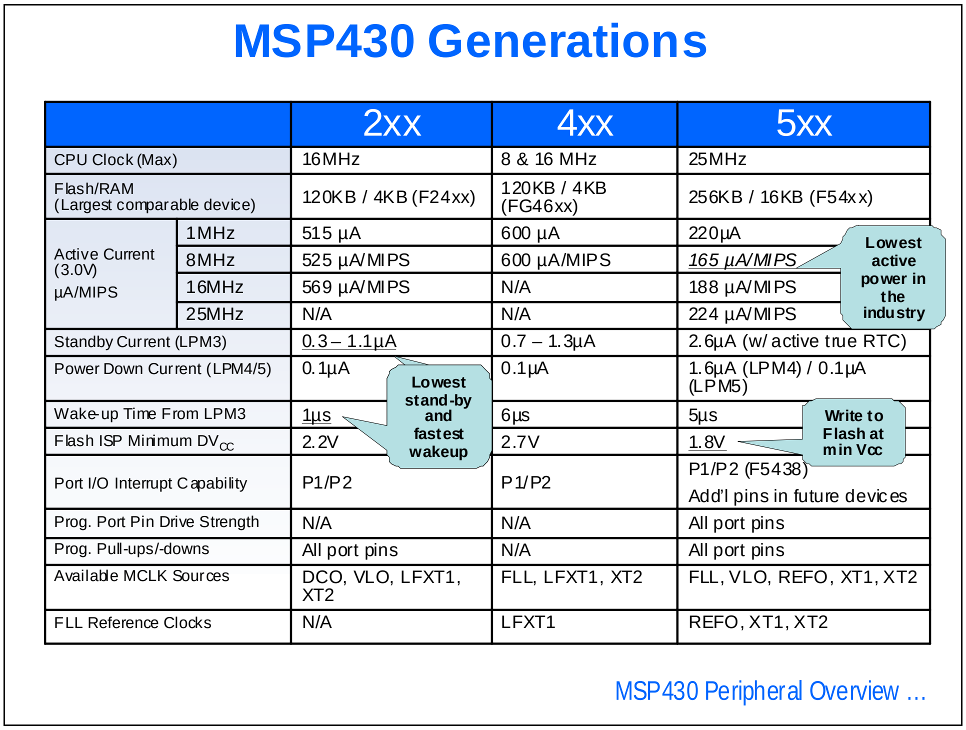
TI Embedded Processing Portfolio

Microcontrollers			ARM-Based		DSP
16-bit	32-bit Real-time	32-bit ARM	ARM+	ARM + DSP	DSP
MSP430	**C2000™**	**Stellaris M3**	**ARM9 Cortex A-8**	**C64x+ plus ARM9/Cortex A-8**	**C647x, C64x+, C55x**
Ultra-Low Power	Fixed & Floating Point	Industry Std Low Power	Industry-Std Core, High-Perf GPP	Industry-Std Core + DSP for Signal Proc.	Leadership DSP Performance
Up to 25 MHz	Up to 150 MHz	Up to 100 MHz	Accelerators	4800 MMACs/ 1.07 DMIPS/MHz	24,000 MMACS
Flash 1 KB to 256 KB	Flash 32 KB to 512 KB	Flash 8kB to 256kB	MMU	MMU, Cache	Up to 3 MB L2 Cache
Analog I/O, ADC LCD, USB, RF	PWM, ADC, CAN, SPI, I²C	USB (H/D/OTG), ENET(PHY, 1588), ADC, PWM, QVGA	USB, LCD, MMC, EMAC	VPSS, USB, EMAC, MMC	1G EMAC, SRIO, DDR2, PCI-66
Measurement, Sensing, General Purpose	Motor Control, Digital Power, Lighting	Host Control	Linux/WinCE User Apps	Linux/Win + Video, Imaging, Multimedia	Comm, WiMAX, Industrial/ Medical Imaging
$0.49 to $9.00	$1.50 to $20.00	$2.00 to $8.00	$8.00 to $35.00	$12.00 to $65.00	$4.00 to $99.00+

Software & Dev. Tools

MSP430 Generations ...

MSP430 Generations



MSP430 Generations

		2xx	4xx	5xx
CPU Clock (Max)		16MHz	8 & 16 MHz	25MHz
Flash/RAM (Largest comparable device)		120KB / 4KB (F24xx)	120KB / 4KB (FG46xx)	256KB / 16KB (F54xx)
Active Current (3.0V) µA/MIPS	1MHz	515 µA	600 µA	220µA
	8MHz	525 µA/MIPS	600 µA/MIPS	*165 µA/MIPS*
	16MHz	569 µA/MIPS	N/A	188 µA/MIPS
	25MHz	N/A	N/A	224 µA/MIPS
Standby Current (LPM3)		0.3 – 1.1µA	0.7 – 1.3µA	2.6µA (w/ active true RTC)
Power Down Current (LPM4/5)		0.1µA	0.1µA	1.6µA (LPM4) / 0.1µA (LPM5)
Wake-up Time From LPM3		1µs	6µs	5µs
Flash ISP Minimum DV$_{CC}$		2.2V	2.7V	1.8V
Port I/O Interrupt Capability		P1/P2	P1/P2	P1/P2 (F5438) Add'l pins in future devices
Prog. Port Pin Drive Strength		N/A	N/A	All port pins
Prog. Pull-ups/-downs		All port pins	N/A	All port pins
Available MCLK Sources		DCO, VLO, LFXT1, XT2	FLL, LFXT1, XT2	FLL, VLO, REFO, XT1, XT2
FLL Reference Clocks		N/A	LFXT1	REFO, XT1, XT2

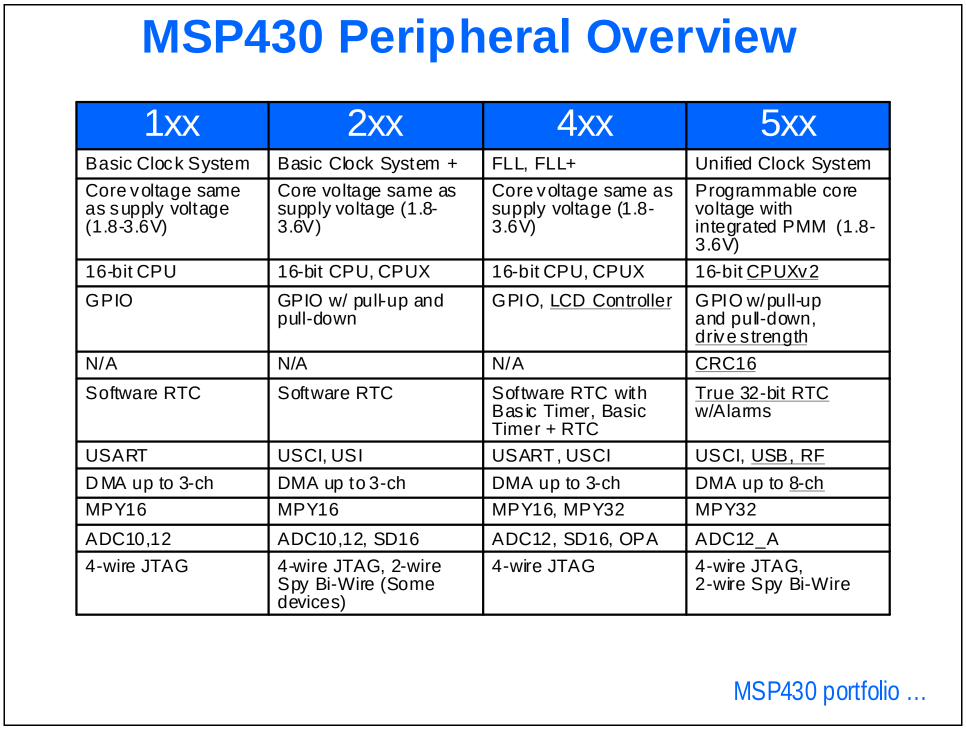
Lowest active power in the industry

Lowest stand-by and fastest wakeup

Write to Flash at min Vcc

MSP430 Peripheral Overview ...

MSP430 Peripheral Overview

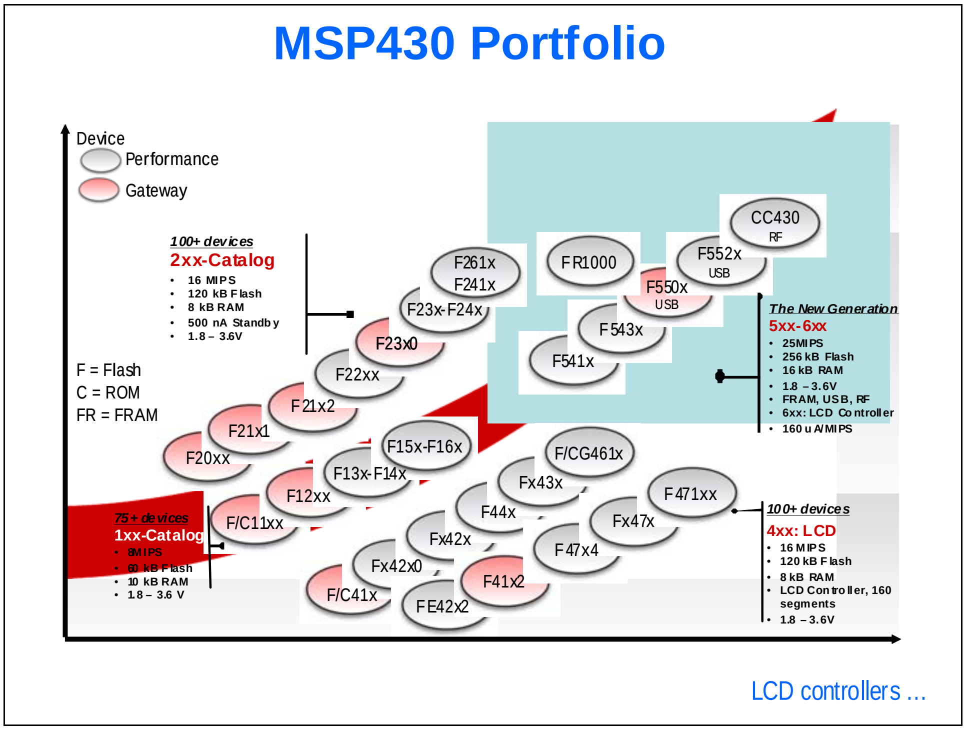


MSP430 Peripheral Overview

1xx	2xx	4xx	5xx
Basic Clock System	Basic Clock System +	FLL, FLL+	Unified Clock System
Core voltage same as supply voltage (1.8-3.6V)	Core voltage same as supply voltage (1.8-3.6V)	Core voltage same as supply voltage (1.8-3.6V)	Programmable core voltage with integrated PMM (1.8-3.6V)
16-bit CPU	16-bit CPU, CPUX	16-bit CPU, CPUX	16-bit CPUXv2
GPIO	GPIO w/ pull-up and pull-down	GPIO, LCD Controller	GPIO w/pull-up and pull-down, drive strength
N/A	N/A	N/A	CRC16
Software RTC	Software RTC	Software RTC with Basic Timer, Basic Timer + RTC	True 32-bit RTC w/Alarms
USART	USCI, USI	USART, USCI	USCI, USB, RF
DMA up to 3-ch	DMA up to 3-ch	DMA up to 3-ch	DMA up to 8-ch
MPY16	MPY16	MPY16, MPY32	MPY32
ADC10,12	ADC10,12, SD16	ADC12, SD16, OPA	ADC12_A
4-wire JTAG	4-wire JTAG, 2-wire Spy Bi-Wire (Some devices)	4-wire JTAG	4-wire JTAG, 2-wire Spy Bi-Wire

MSP430 portfolio ...

MSP430 Portfolio



MSP430 Portfolio

100+ devices

2xx-Catalog

- 16 MIPS
- 120 kB Flash
- 8 kB RAM
- 500 nA Standby
- 1.8 – 3.6V

75+ devices

1xx-Catalog

- 8MIPS
- 60 kB Flash
- 10 kB RAM
- 1.8 – 3.6 V

The New Generation

5xx-6xx

- 25MIPS
- 256 kB Flash
- 16 kB RAM
- 1.8 – 3.6V
- FRAM, USB, RF
- 6xx: LCD Controller
- 160 uA/MIPS

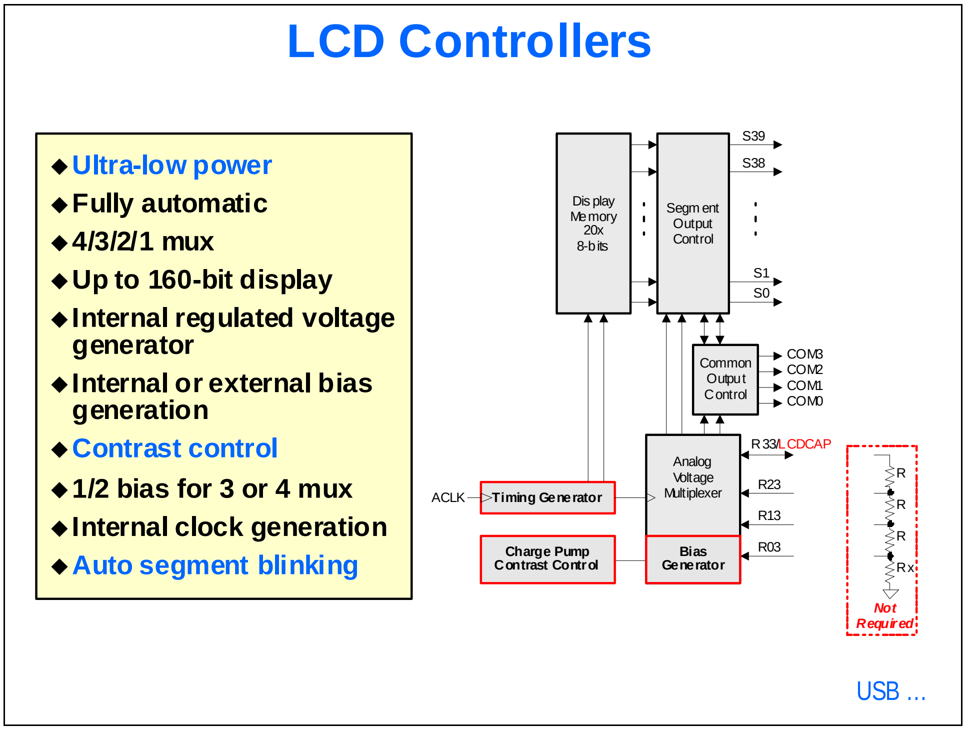
100+ devices

4xx: LCD

- 16 MIPS
- 120 kB Flash
- 8 kB RAM
- LCD Controller, 160 segments
- 1.8 – 3.6V

LCD controllers ...

LCD Controllers

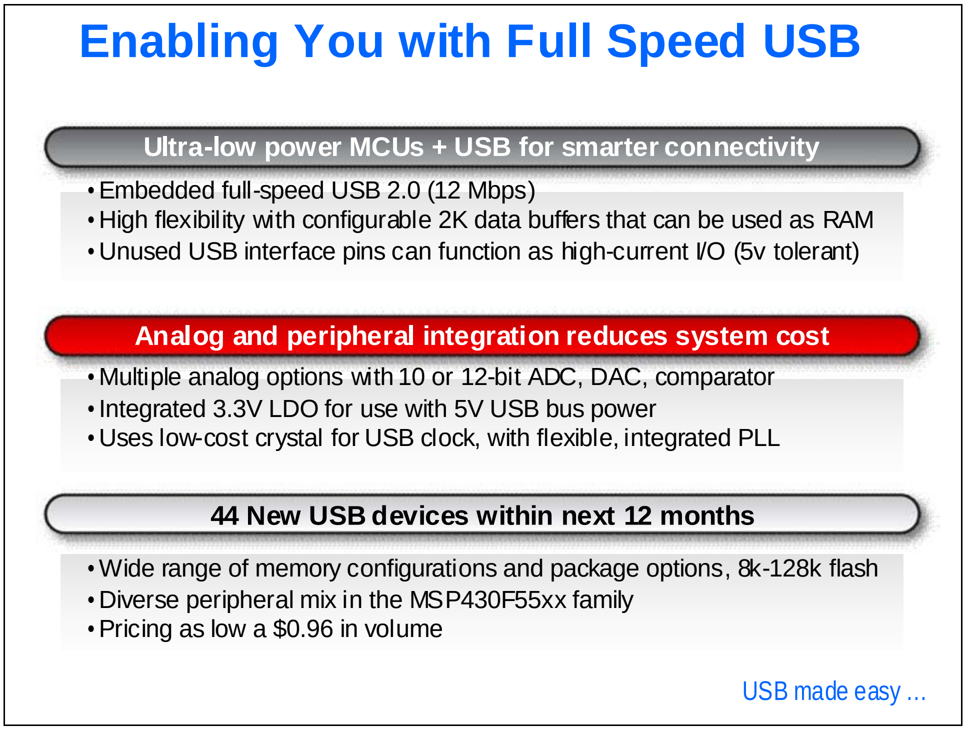


LCD Controllers

- ◆ 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 …

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)

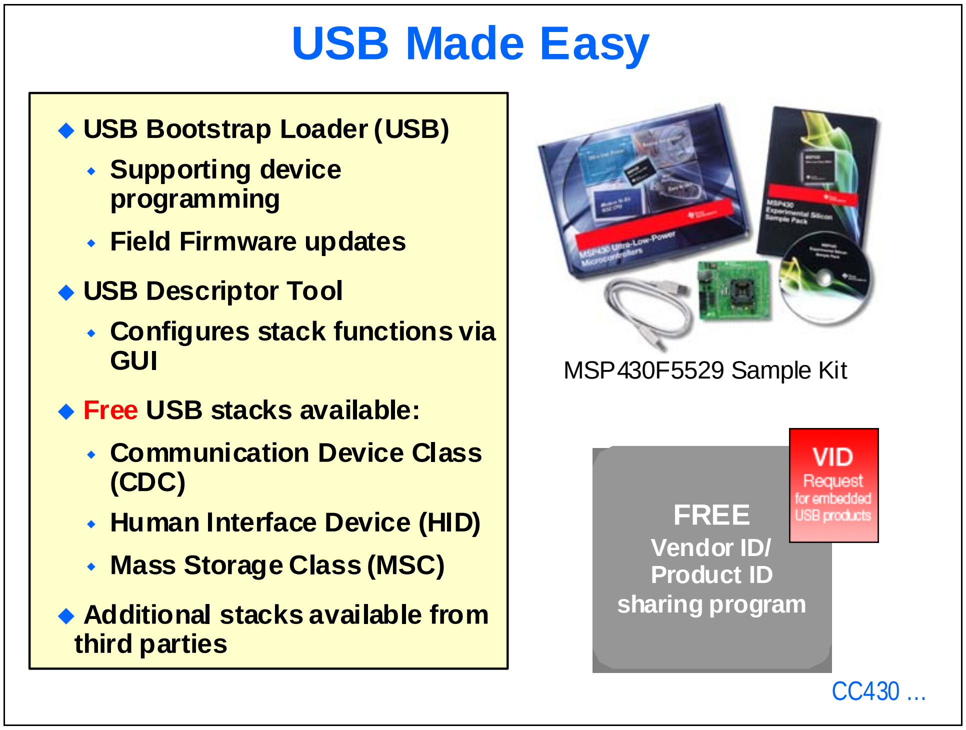
Analog and peripheral integration reduces system cost

- 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**

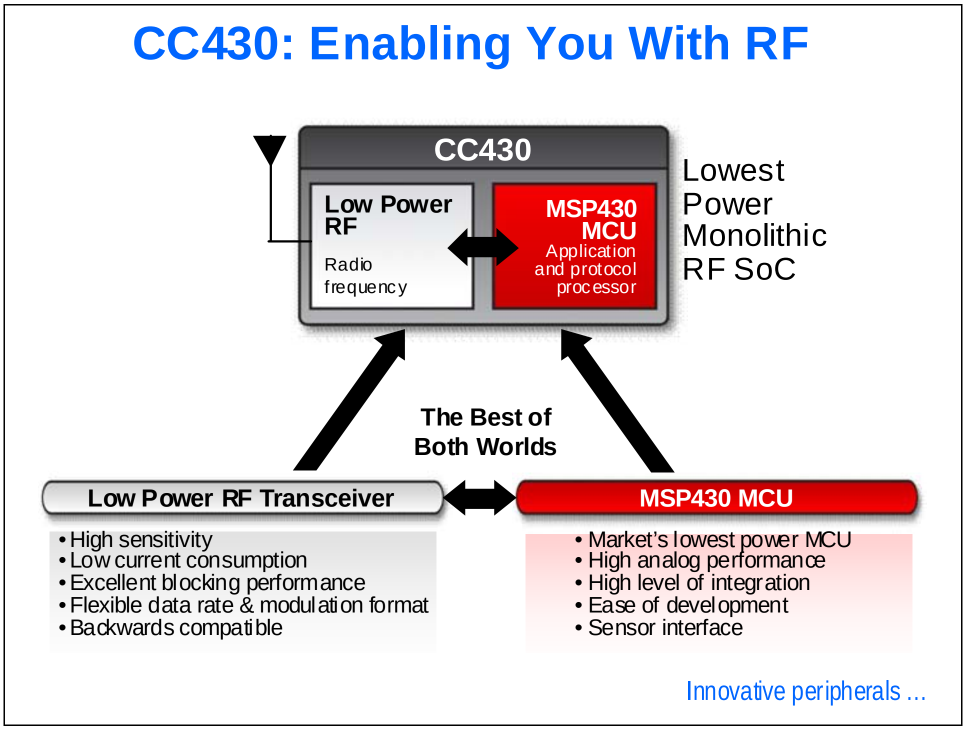
MSP430F5529 Sample Kit

FREE
Vendor ID/ Product ID sharing program

VID
Request for embedded USB products

CC430 …

CC430



CC430: Enabling You With RF

CC430

Low Power RF

Radio frequency

MSP430 MCU

Application and protocol processor

Lowest Power Monolithic RF SoC

The Best of Both Worlds

Low Power RF Transceiver

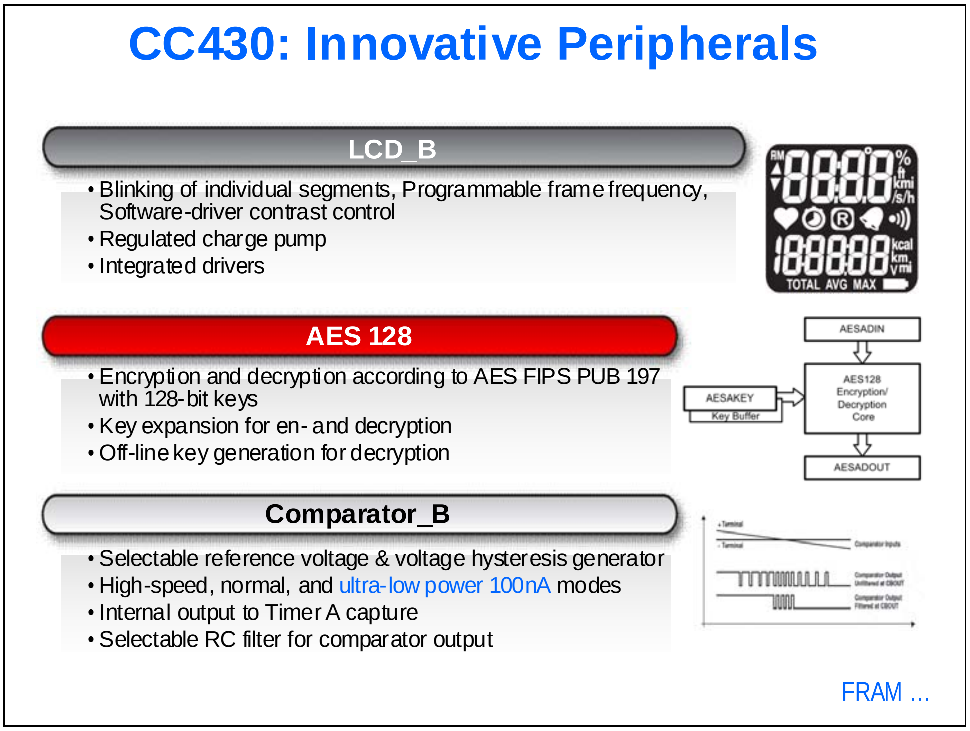
- High sensitivity
- Low current consumption
- Excellent blocking performance
- Flexible data rate & modulation format
- Backwards compatible

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

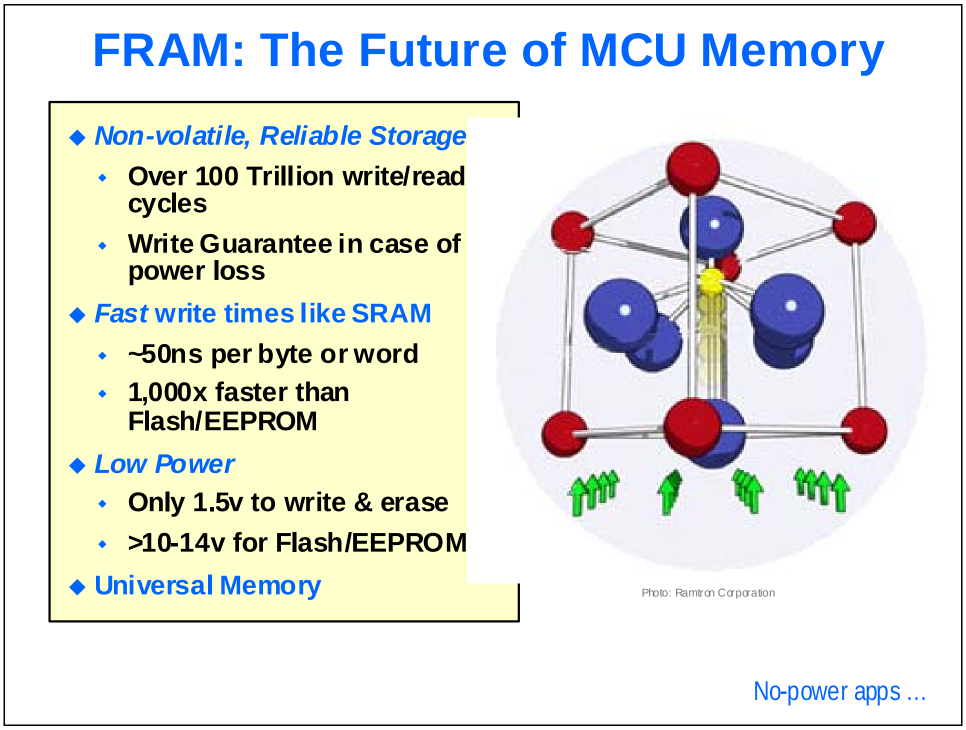
Comparator_B

- 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

+Terminal
- Terminal
Comparator Inputs
Comparator Output Unfiltered at CBOUT
Comparator Output Filtered at CBOUT

FRAM ...

FRAM



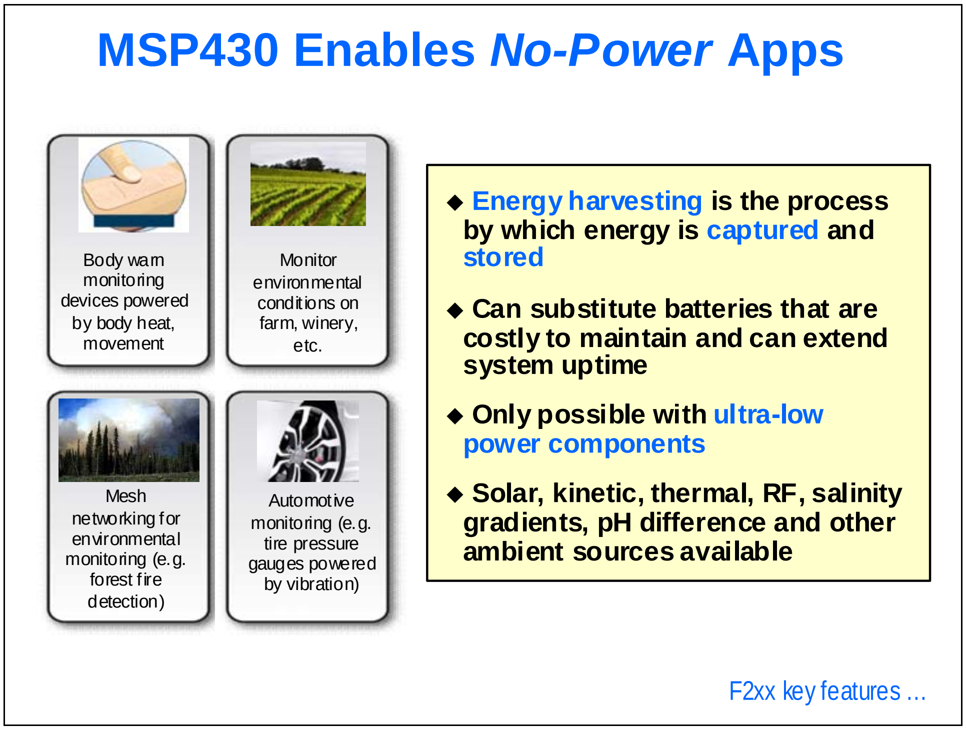
FRAM: The Future of MCU Memory

- *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

Photo: Ramtron Corporation

No-power apps ...

No-Power Apps



MSP430 Enables *No-Power* Apps

Body warn monitoring devices powered by body heat, movement

Monitor environmental conditions on farm, winery, etc.

Mesh networking for environmental monitoring (e.g. forest fire detection)

Automotive monitoring (e.g. tire pressure gauges powered by vibration)

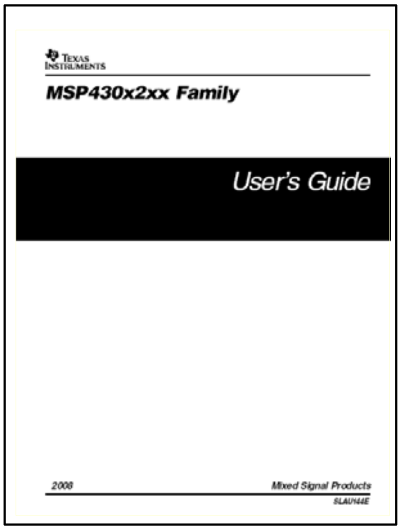
- Energy harvesting is the process by which energy is captured and stored
- Can substitute batteries that are costly to maintain and can extend system uptime
- Only possible with ultra-low power components
- Solar, kinetic, thermal, RF, salinity gradients, pH difference and other ambient sources available

F2xx key features ...

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*



Texas Instruments

MSP430x2xx Family

User's Guide

2008

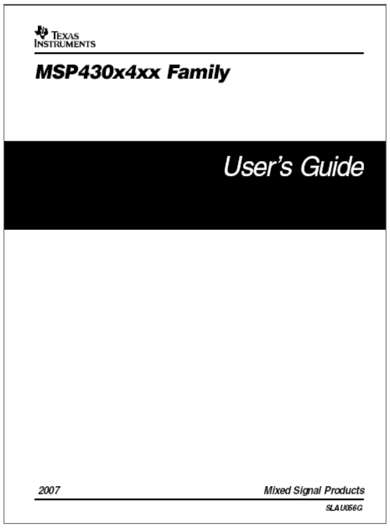
Mixed Signal Products

SLAU144E

F4xx key features ...

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*



Texas Instruments

MSP430x4xx Family

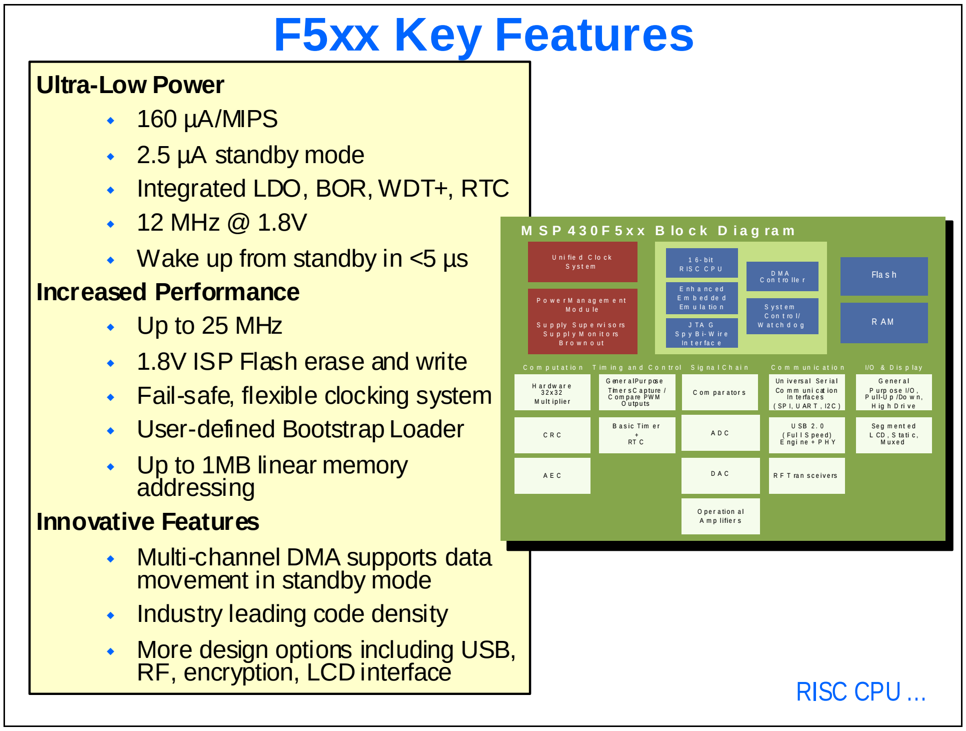
User's Guide

2007

Mixed Signal Products

SLAU056G

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

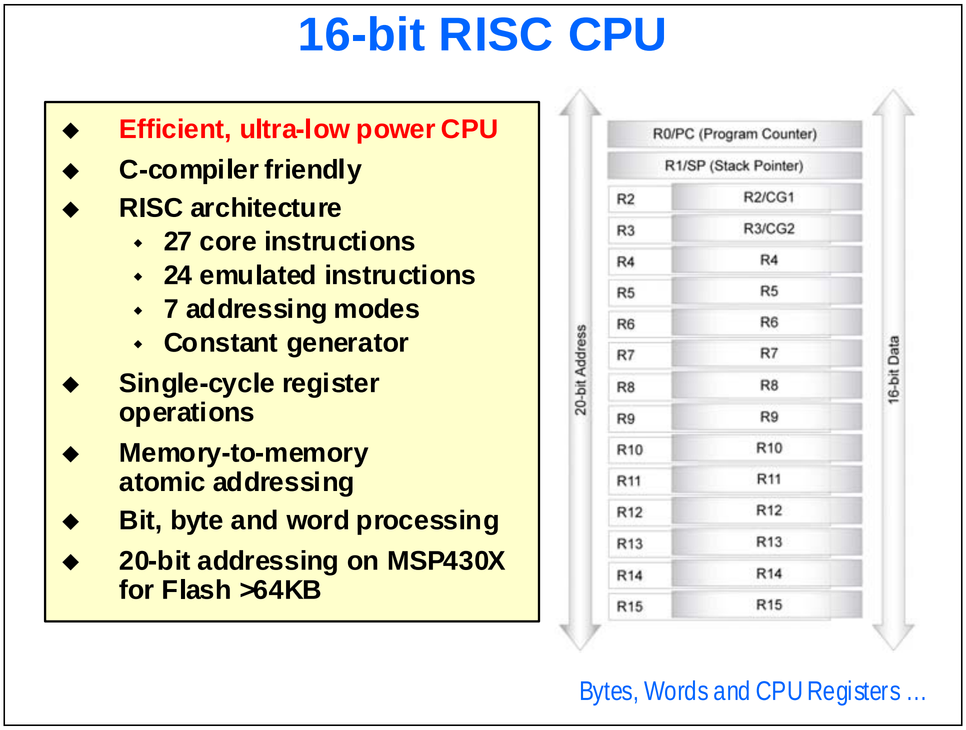
- Up to 25 MHz
- 1.8V ISP Flash erase and write
- Fail-safe, flexible clocking system
- User-defined Bootstrap Loader
- Up to 1MB linear memory addressing

Innovative Features

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

RISC CPU ...

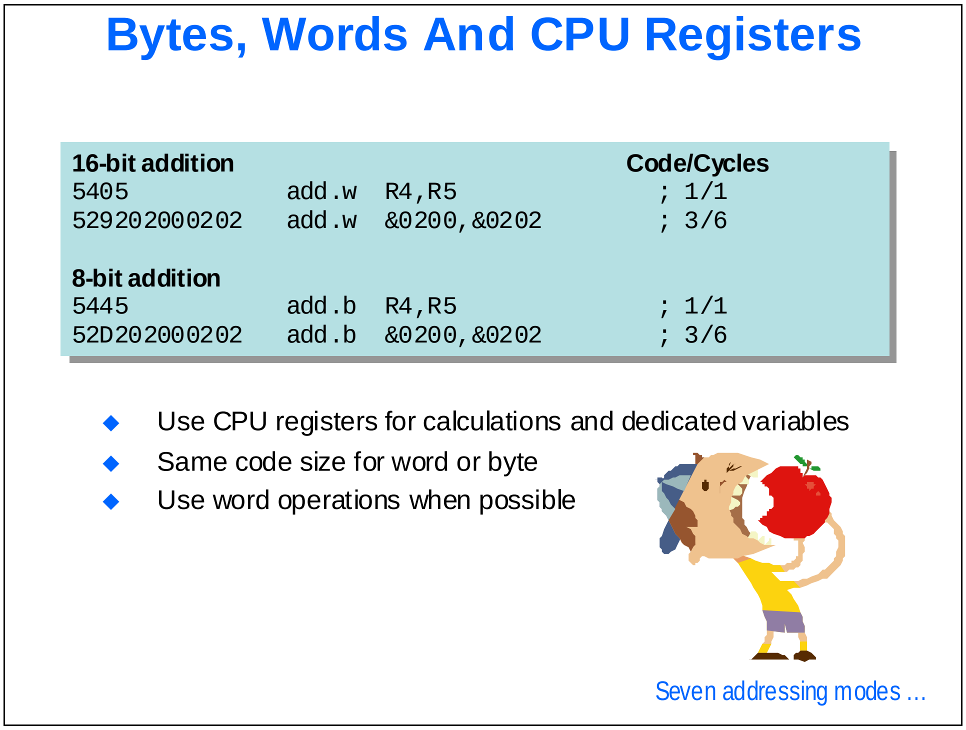
The Nuts and Bolts



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 …

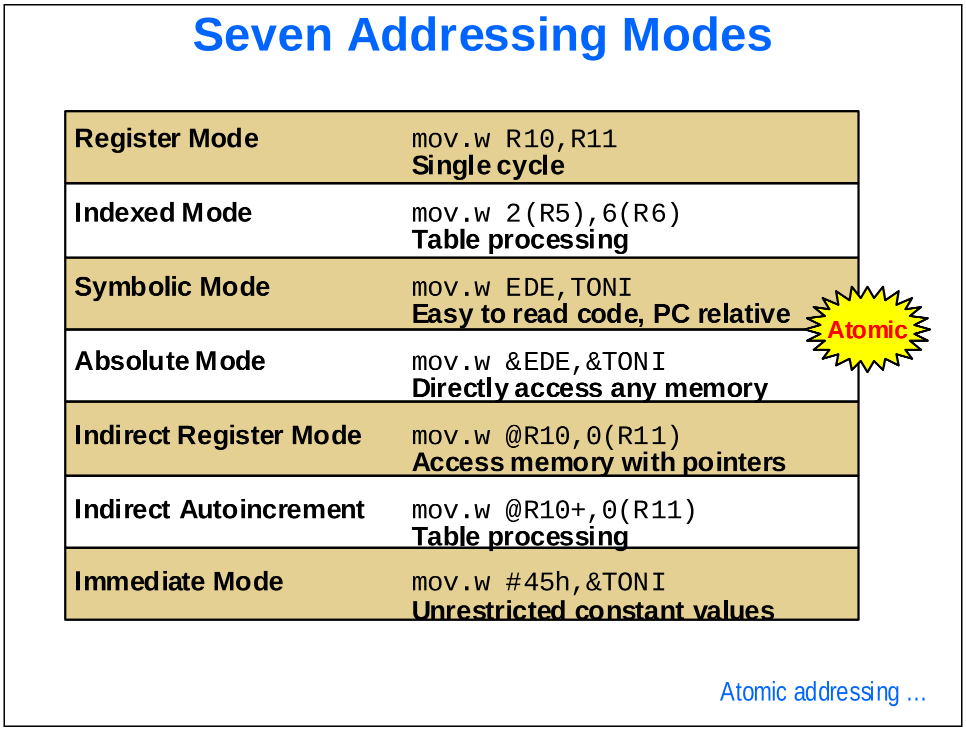


Bytes, Words And CPU Registers

```
16-bit addition                                     Code/Cycles
5405            add.w   R4,R5                        ; 1/1
529202000202    add.w   &0200,&0202                  ; 3/6

8-bit addition
5445            add.b   R4,R5                        ; 1/1
52D202000202    add.b   &0200,&0202                  ; 3/6
```

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

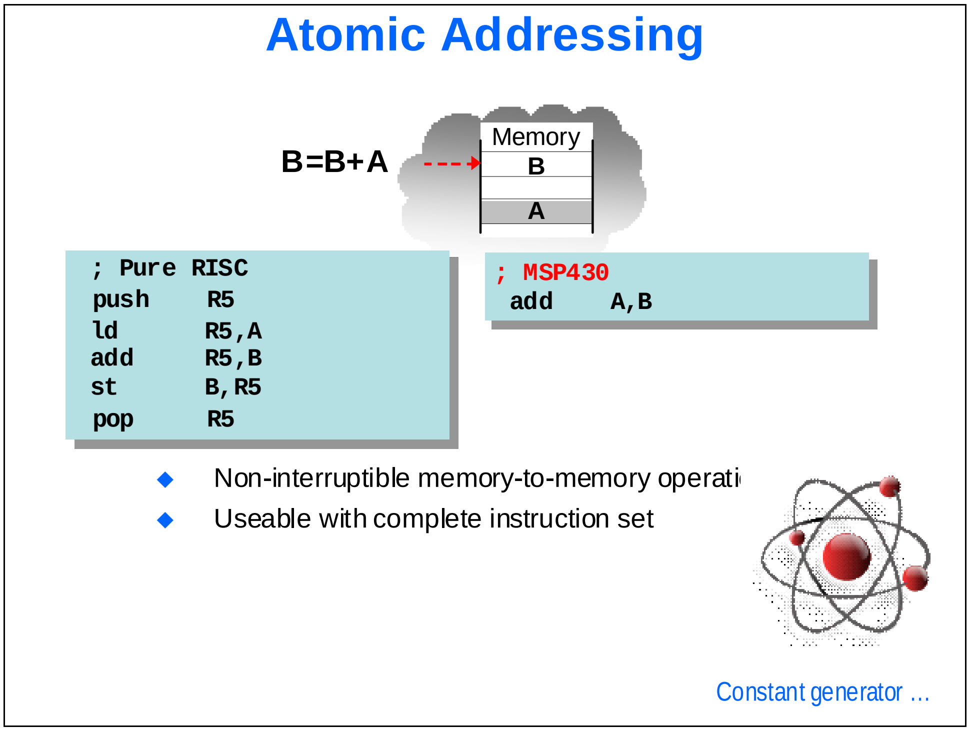
Seven addressing modes …



Seven Addressing Modes

Register Mode	`mov.w R10,R11` **Single cycle**
Indexed Mode	`mov.w 2(R5),6(R6)` **Table processing**
Symbolic Mode	`mov.w EDE,TONI` **Easy to read code, PC relative**
Absolute Mode	`mov.w &EDE,&TONI` **Directly access any memory**
Indirect Register Mode	`mov.w @R10,0(R11)` **Access memory with pointers**
Indirect Autoincrement	`mov.w @R10+,0(R11)` **Table processing**
Immediate Mode	`mov.w #45h,&TONI` **Unrestricted constant values**

Atomic addressing ...



Atomic Addressing

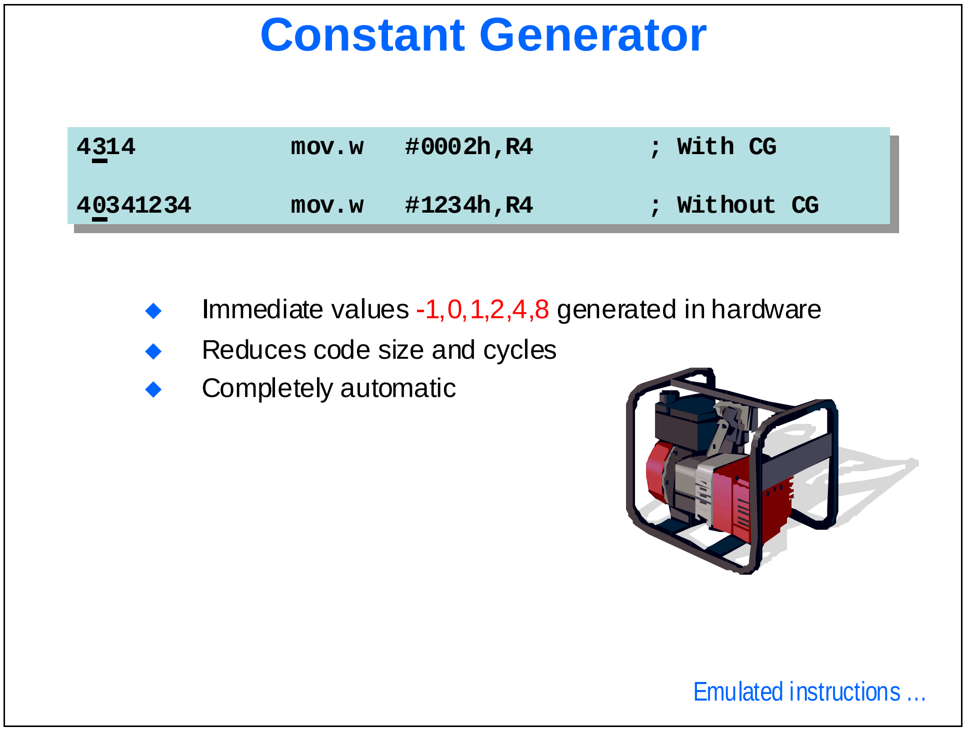
B=B+A

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

```
; MSP430
add     A,B
```

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

Constant generator ...

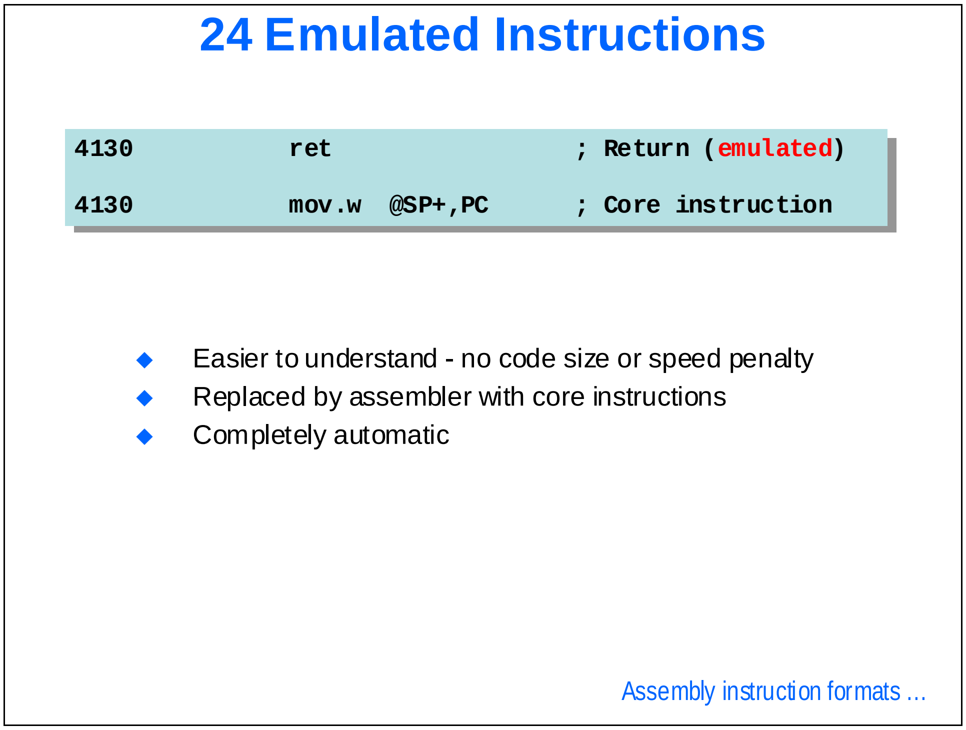


Constant Generator

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

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

Emulated instructions ...

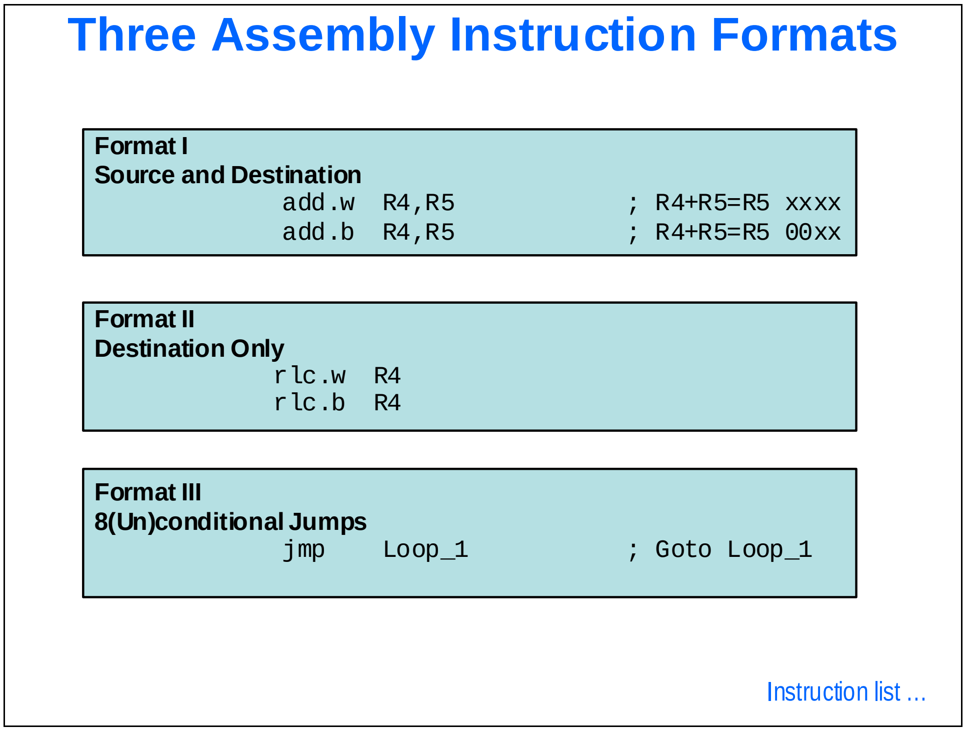


24 Emulated Instructions

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

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

Assembly instruction formats ...



Three Assembly Instruction Formats

Format I
Source and Destination

```
add.w  R4,R5          ; R4+R5=R5 xxxx
add.b  R4,R5          ; R4+R5=R5 00xx
```

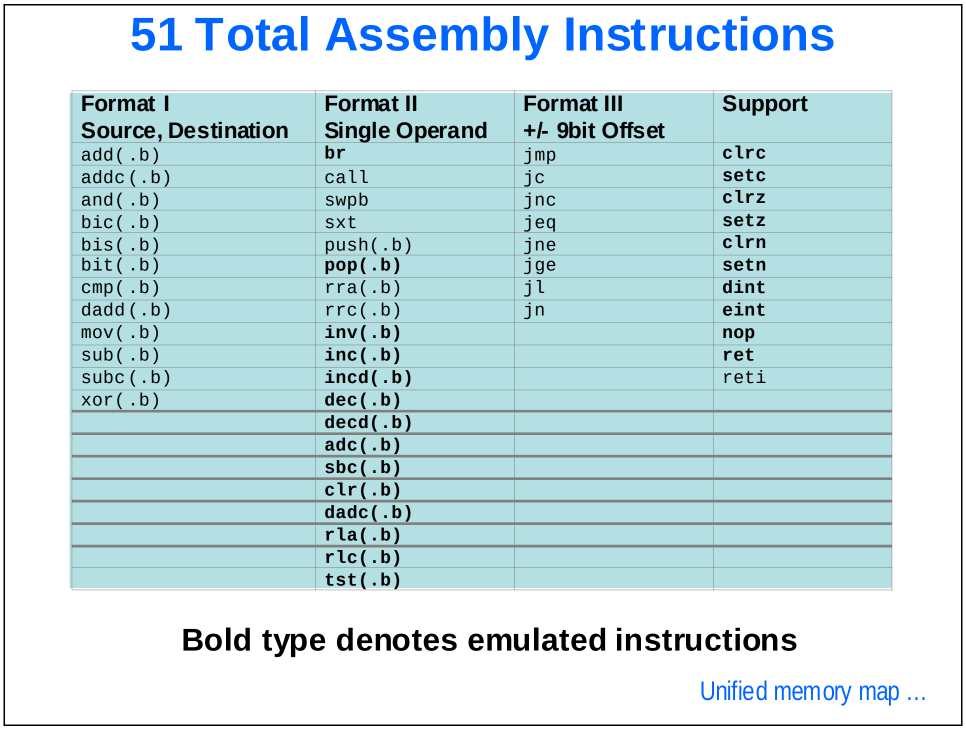
Format II
Destination Only

```
rlc.w  R4
rlc.b  R4
```

Format III
8(Un)conditional Jumps

```
jmp    Loop_1         ; Goto Loop_1
```

Instruction list …



51 Total Assembly Instructions

Format I Source, Destination	Format II Single Operand	Format III +/- 9bit Offset	Support
add(.b)	**br**	jmp	**clrc**
addc(.b)	call	jc	**setc**
and(.b)	swpb	jnc	**clrz**
bic(.b)	sxt	jeq	**setz**
bis(.b)	push(.b)	jne	**clrn**
bit(.b)	**pop(.b)**	jge	**setn**
cmp(.b)	rra(.b)	jl	**dint**
dadd(.b)	rrc(.b)	jn	**eint**
mov(.b)	**inv(.b)**		**nop**
sub(.b)	**inc(.b)**		**ret**
subc(.b)	**incd(.b)**		reti
xor(.b)	**dec(.b)**		
	decd(.b)		
	adc(.b)		
	sbc(.b)		
	clr(.b)		
	dadc(.b)		
	rla(.b)		
	rlc(.b)		
	tst(.b)		

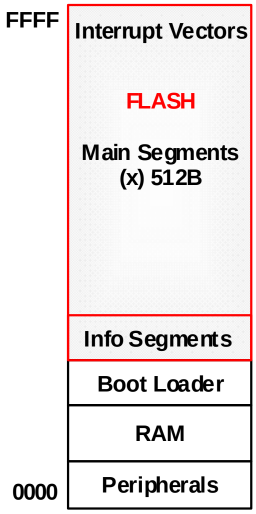
Bold type denotes emulated instructions

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

Embedded Emulation

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

Tools ...

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

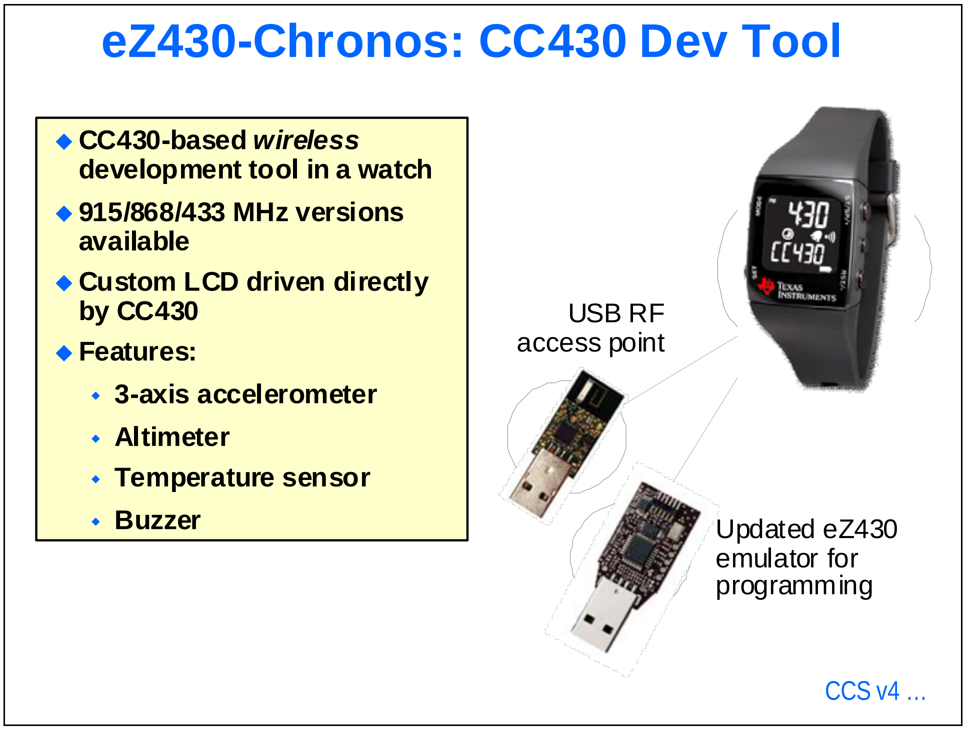
MSP430 Experimenter Boards
- Fully featured prototyping system
- Available for FG4618 & F5438
- Starting at $99

) Tools

plete development
m in USB stick
able for wireless
energy harvesting
ing at $20

Chronos …

eZ430-Chronos



eZ430-Chronos: CC430 Dev Tool

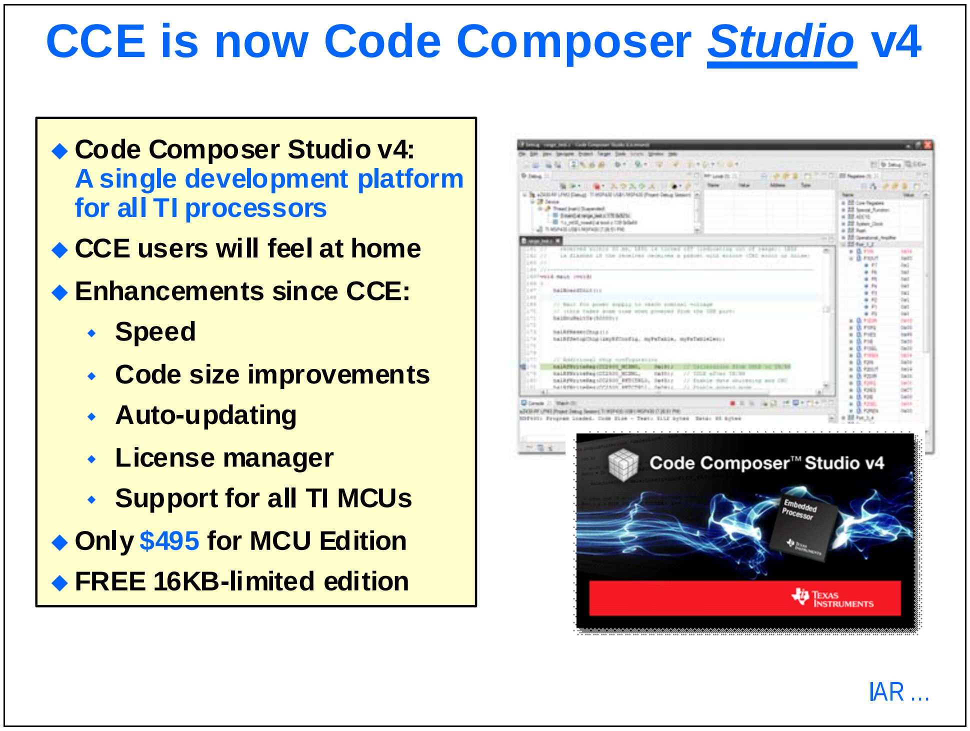
- **CC430-based *wireless* development tool in a watch**
- **915/868/433 MHz versions available**
- **Custom LCD driven directly by CC430**
- **Features:**
  - **3-axis accelerometer**
  - **Altimeter**
  - **Temperature sensor**
  - **Buzzer**

USB RF
access point

Updated eZ430
emulator for
programming

CCS v4 …

Code Composer Studio V4

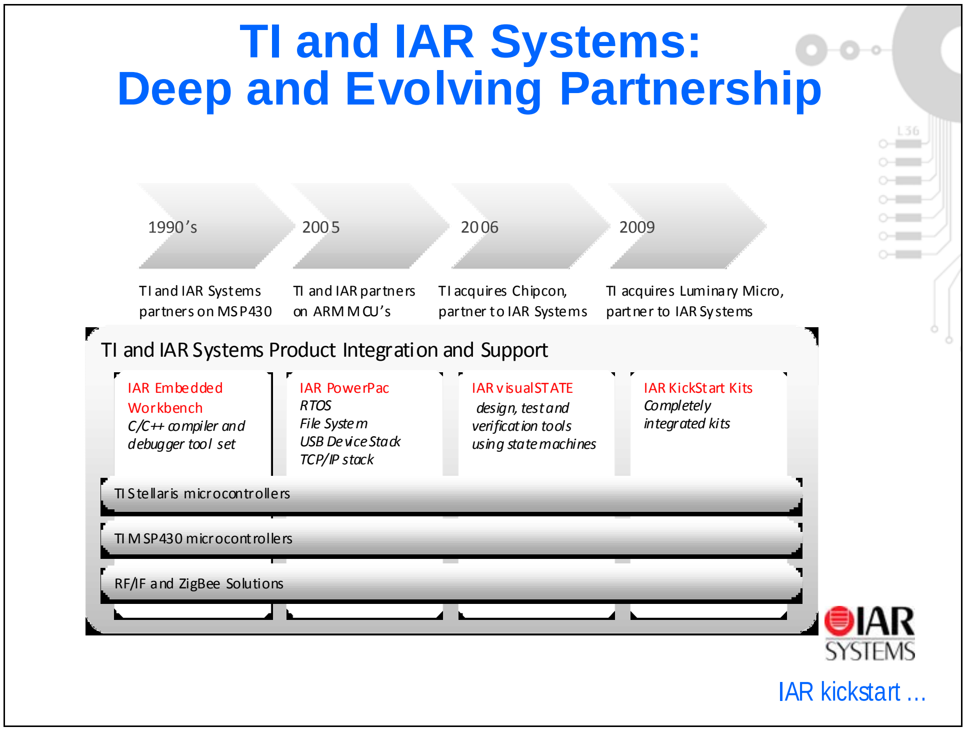


CCE is now Code Composer *Studio* v4

- 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 …

IAR Systems



TI and IAR Systems: Deep and Evolving Partnership

1990's
TI and IAR Systems partners on MSP430

2005
TI and IAR partners on ARM MCU's

2006
TI acquires Chipcon, partner to IAR Systems

2009
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
File System
USB Device Stack
TCP/IP stack

IAR visualSTATE
design, test and verification tools using state machines

IAR KickStart Kits
Completely integrated kits

TI Stellaris microcontrollers

TI MSP430 microcontrollers

RF/IF and ZigBee Solutions

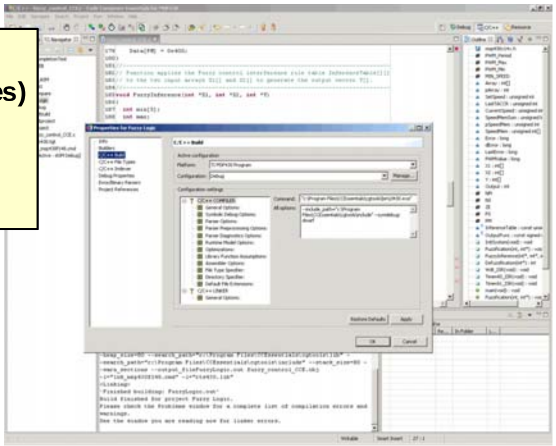
IAR SYSTEMS

IAR kickstart …

IAR Kickstart

IAR Kickstart IDE

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



Third parties ...

Third Party Resources

Third Party Development Resources

- **Rowley CrossWorks**
  - Complete IDE solution
  - High code density
  - Simulator
  - Windows, Linux, Mac

www.rowley.co.uk

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

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

- **MSPGCC Tool Chain**
  - Free
  - Open Source
  - GNU C Compiler, Assembler/ Linker, GDB Debugger
  - Windows, Linux, Unix

http://mspgcc.sourceforge.net

- **Amber Wireless**
  - Drop in wireless modules
  - <1GHZ eZ430-RF target boards
  - CC430 Development boards

- **USB Stacks**
  - IAR
  - HCC

ti.com/msp430 ...

www.ti.com/msp430

www.ti.com/msp430

- User's Guides
- Datasheets
- TI Community Forum
- 100+ Application Reports
- 1000+ Code Examples
- Product Brochure
- MCU Selection Tool
- Latest Tool Software
- 3rd Party Listing
- Silicon Errata

Community support …

Community Support

Extensive Community Support

E2E Community

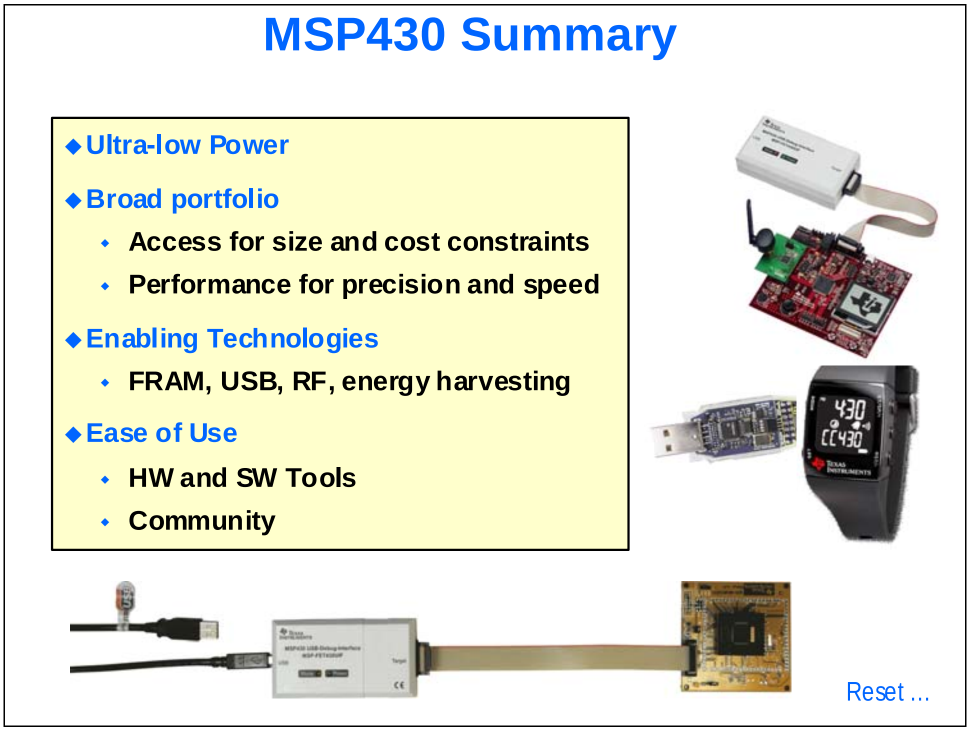
- Videos, Blogs, Forums
- Extensive community support and idea exchange
- Global customer support
- http://e2e.ti.com

Processor Wiki

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

Summary …

MSP430 Summary

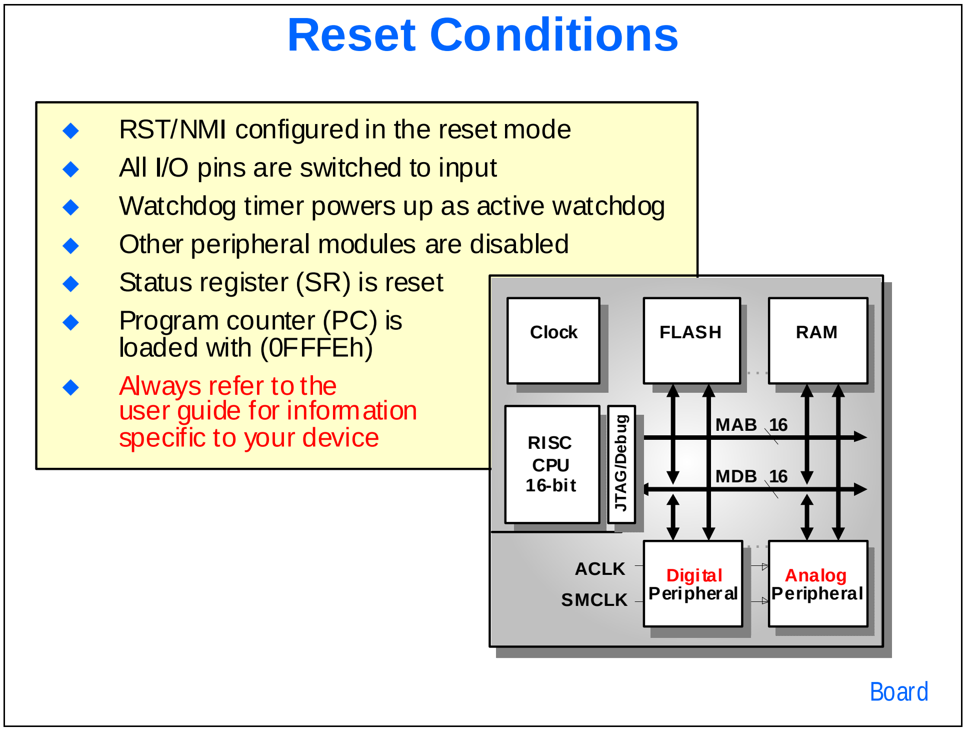


MSP430 Summary

- 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 ...

Reset Conditions



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

Board

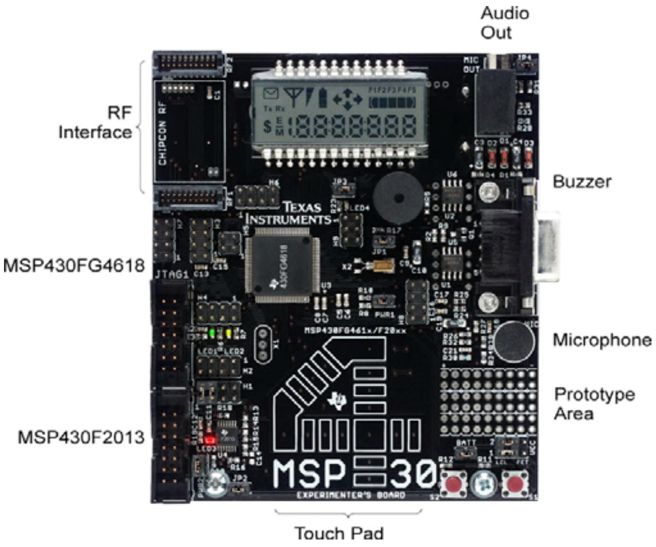
Experimenter’s Board

MSP430FG461x/F20xx Experimenter’s Board

Two MSP430 devices:

- MSP430FG4618 or MSP430FG4619
- MSP430F2013

Interface for ChipCon RF transceiver EMK boards

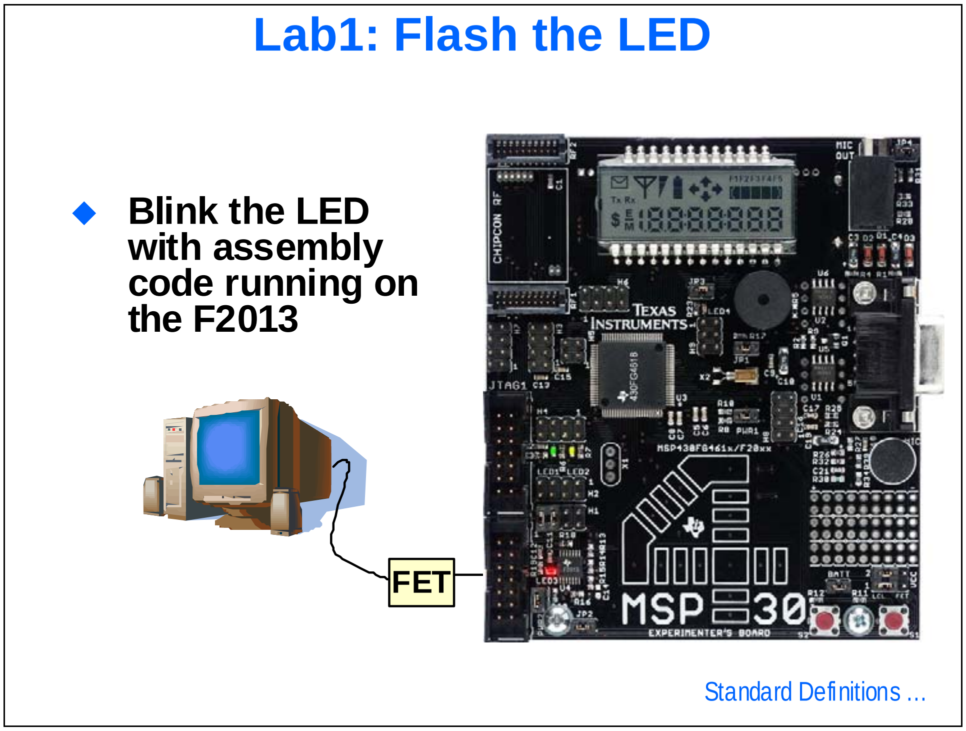


Lab1 …

*** ***

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.



Lab1: Flash the LED

- Blink the LED with assembly code running on the F2013

Standard Definitions ...

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

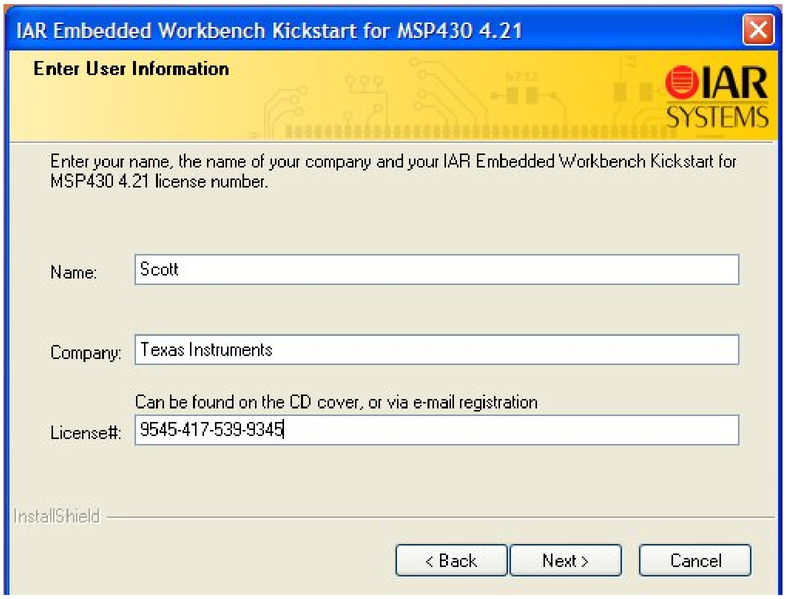
In this lab, you will verify that the hardware/software has been set up properly. We'll also familiarize ourselves with the tools we'll be using for the rest of the workshop via a short program running on the F2013.

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

Enter User Information

Enter your name, the name of your company and your IAR Embedded Workbench Kickstart for MSP430 4.21 license number.

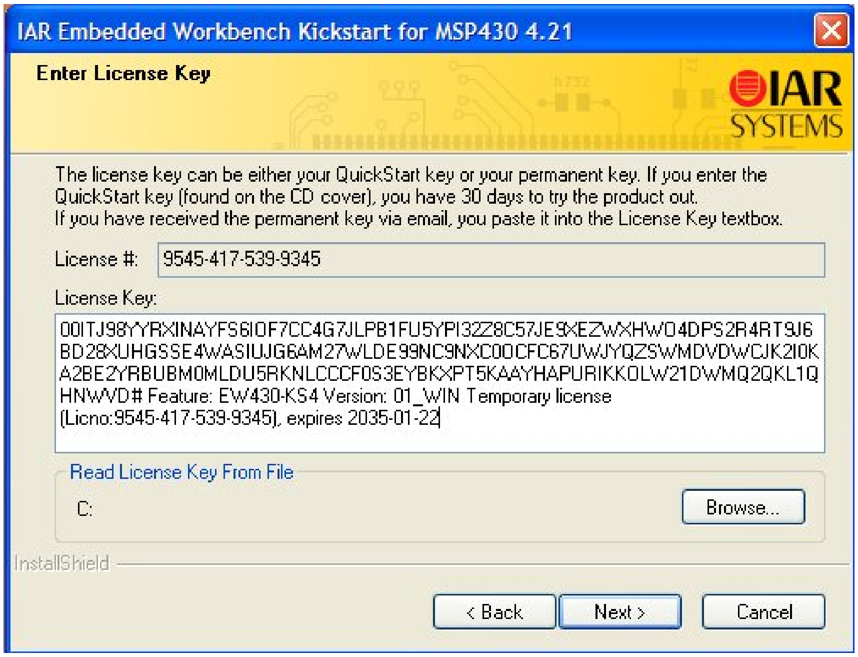
Name: Scott

Company: Texas Instruments

Can be found on the CD cover, or via e-mail registration

License#: 9545-417-539-9345

4. In the same way, **copy/paste** the **License Key** into the next window and click **Next**.



The license key can be either your QuickStart key or your permanent key. If you enter the QuickStart key (found on the CD cover), you have 30 days to try the product out.
If you have received the permanent key via email, you paste it into the License Key textbox.

License #: 9545-417-539-9345

License Key:

```
00ITJ98YYRXINAYFS6IOF7CC4G7JLPB1FU5YPI32Z8C57JE9XEZWXHWO4DPS2R4RT9J6
BD28XUHGSSE4WASIUJG6AM27WLDE99NC9NXC0OCFC67UWJYQZSWMDVDWCJK2I0K
A2BE2YRBUBM0MLDU5RKNLCCCF0S3EYBKXPT5KAAYHAPURIKKOLW21DWMQ2QKL1Q
HNWVD# Feature: EW430-KS4 Version: 01_WIN Temporary license
(Licno:9545-417-539-9345), expires 2035-01-22
```

Read License Key From File

C:

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.

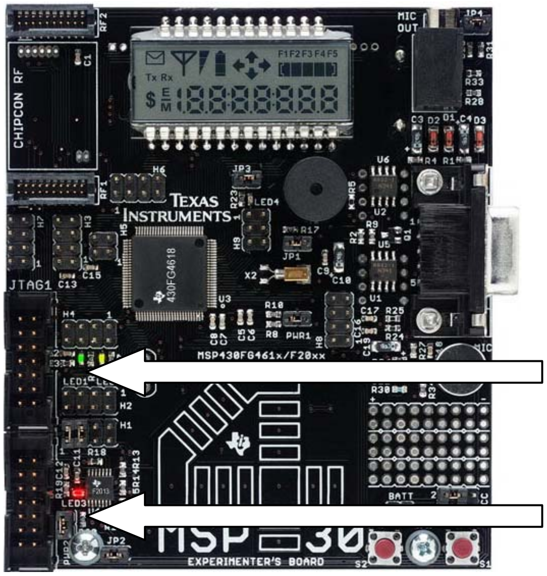
6. **Lab Files Installation**
   Using Windows Explorer, look on the workshop flash drive and double-click on **all_labs.exe**. Leave the unzip directory as **C:\** and click **Unzip**. When the process completes, click **Close**. The labs have been placed in **C:\MSP430ODW**.

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).



MSP430FG4619 JTAG Emulation Port

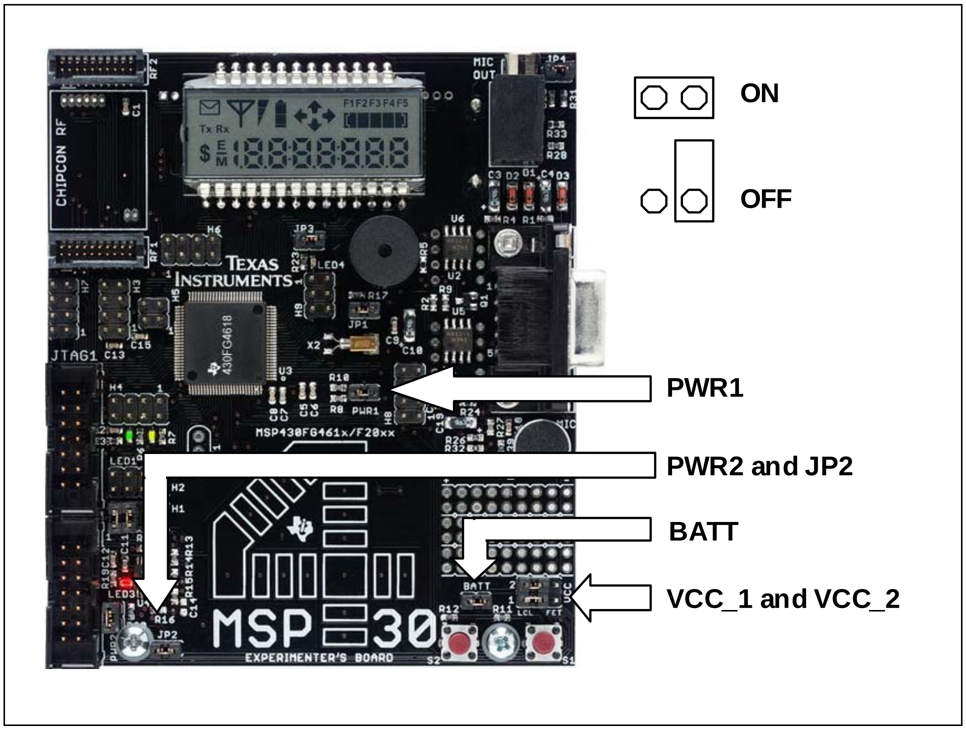
MSP430F2013 JTAG Emulation Port

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:

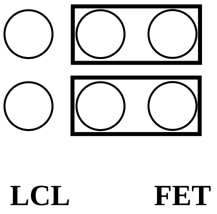
PWR1 controls power to the MSP430FG4619 (**ON**)

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:

IAR Embedded Workbench

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.

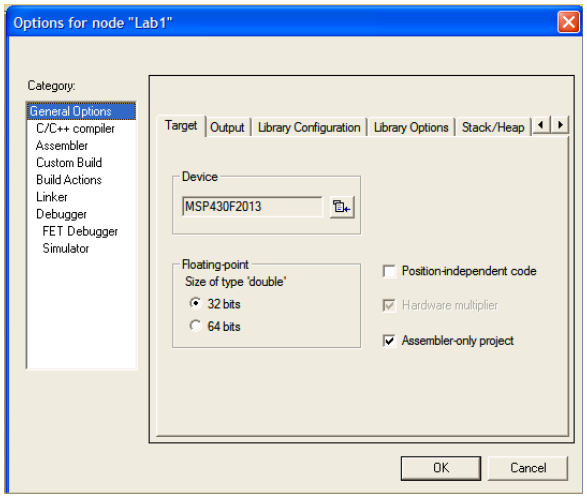
Still under the *Target* tab, click **Assembler-only project**.

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**.



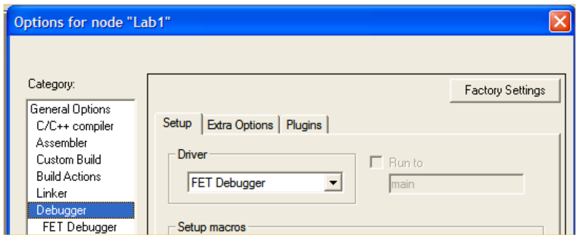
Device: MSP430F2013

Floating-point: Size of type 'double' ● 32 bits ○ 64 bits

□ Position-independent code

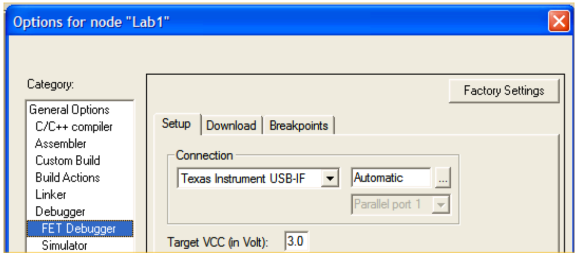
☑ Hardware multiplier

☑ Assembler-only project



Driver: FET Debugger

□ Run to: main



Connection: Texas Instrument USB-IF; Automatic; Parallel port 1

Target VCC (in Volt): 3.0

Create and Add the Source File

8. Create the Source File

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

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

```
#include  "msp430x20x3.h"

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

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

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

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

9. Add the File to the Project

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

Download and Run the Program

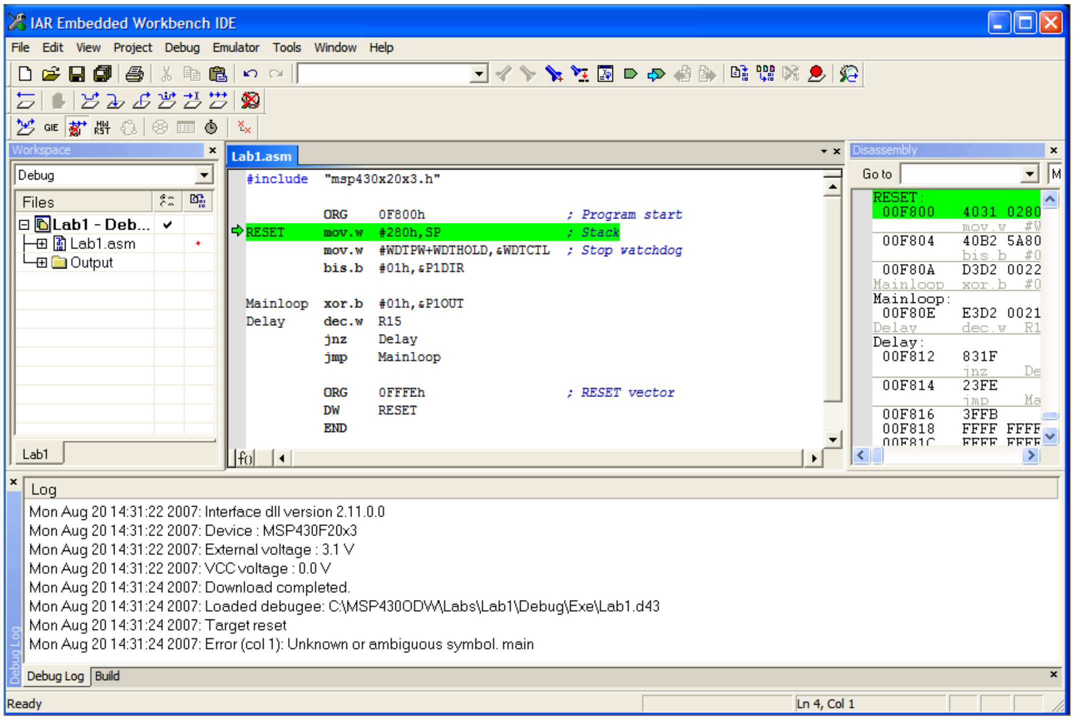
10. Assemble and Download

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

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

11. Run the Program

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



```
#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
```

```
RESET:
 00F800    4031 0280
           mov.w  #0
 00F804    40B2 5A80
           bis.b  #0
 00F80A    D3D2 0022
Mainloop   xor.b  #0
Mainloop:
 00F80E    E3D2 0021
Delay      dec.w  R1
Delay:
 00F812    831F
           jnz    De
 00F814    23FE
           jmp    Ma
 00F816    3FFB
 00F818    FFFF FFFF
```

```
Mon Aug 20 14:31:22 2007: Interface dll version 2.11.0.0
Mon Aug 20 14:31:22 2007: Device : MSP430F20x3
Mon Aug 20 14:31:22 2007: External voltage : 3.1 V
Mon Aug 20 14:31:22 2007: VCC voltage : 0.0 V
Mon Aug 20 14:31:24 2007: Download completed.
Mon Aug 20 14:31:24 2007: Loaded debugee: C:\MSP430ODW\Labs\Lab1\Debug\Exe\Lab1.d43
Mon Aug 20 14:31:24 2007: Target reset
Mon Aug 20 14:31:24 2007: Error (col 1): Unknown or ambiguous symbol. main
```

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

12. Stop Debugging and Close IAR Kickstart

Click the **Stop Debugging** button.

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

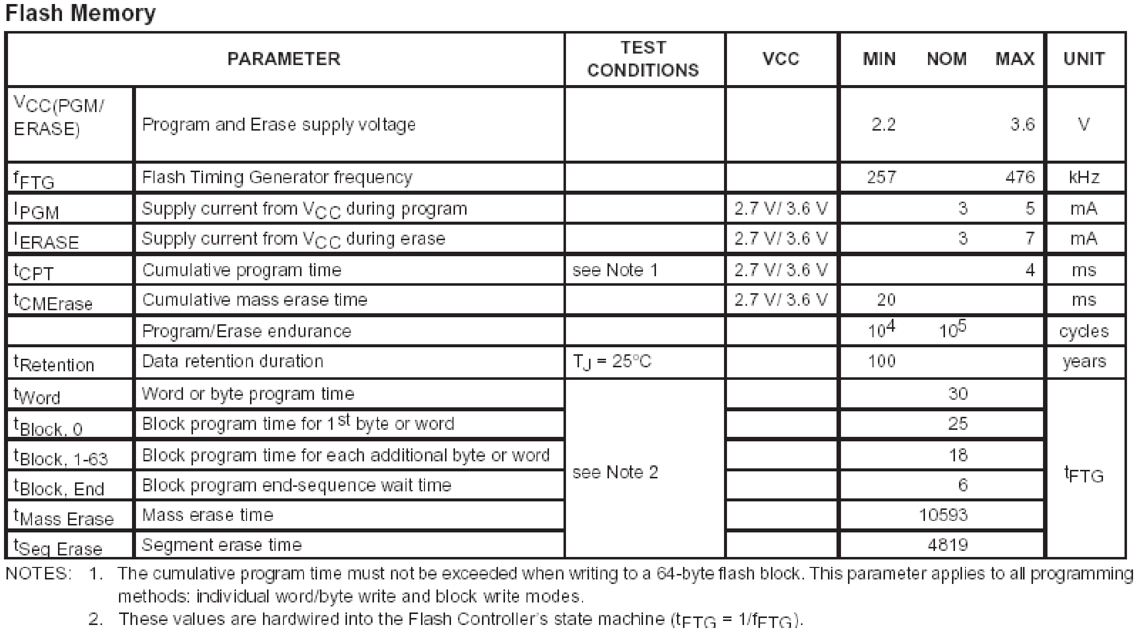
FLASH Programming Exercise

13. Exercise

In the F2xx family, the time to program any bit, byte or word in FLASH is $30/f_{FTG}$ – 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

PARAMETER		TEST CONDITIONS	VCC	MIN	NOM	MAX	UNIT
$V_{CC(PGM/ERASE)}$	Program and Erase supply voltage			2.2		3.6	V
f_{FTG}	Flash Timing Generator frequency			257		476	kHz
I_{PGM}	Supply current from V_{CC} during program		2.7 V/ 3.6 V		3	5	mA
I_{ERASE}	Supply current from V_{CC} during erase		2.7 V/ 3.6 V		3	7	mA
t_{CPT}	Cumulative program time	see Note 1	2.7 V/ 3.6 V			4	ms
$t_{CMErase}$	Cumulative mass erase time		2.7 V/ 3.6 V	20			ms
	Program/Erase endurance			10^4	10^5		cycles
$t_{Retention}$	Data retention duration	T_J = 25°C		100			years
t_{Word}	Word or byte program time	see Note 2			30		t_{FTG}
$t_{Block, 0}$	Block program time for 1st byte or word				25		
$t_{Block, 1-63}$	Block program time for each additional byte or word				18		
$t_{Block, End}$	Block program end-sequence wait time				6		
$t_{Mass Erase}$	Mass erase time				10593		
$t_{Seg Erase}$	Segment erase time				4819		

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.

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

CCS 4.1 Procedure

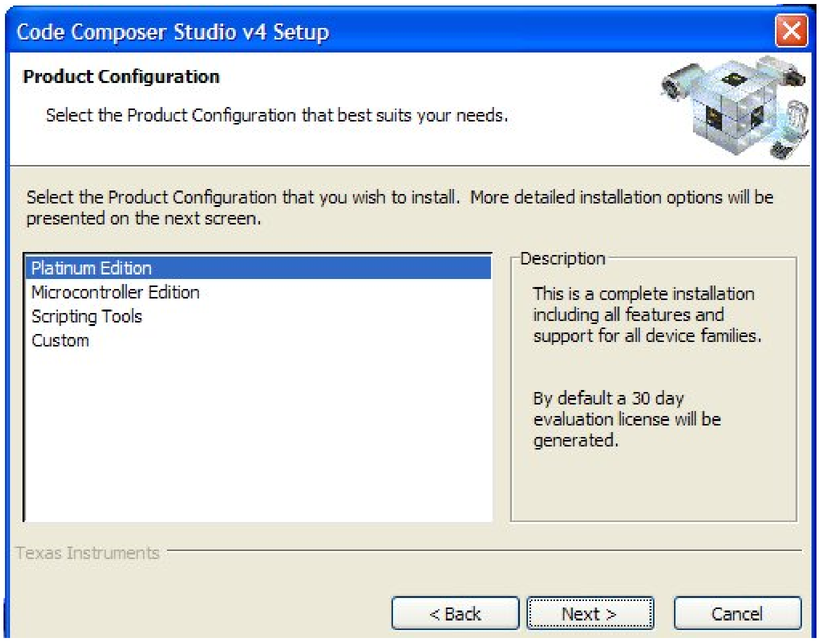
In this lab, you will install Code Composer Studio and verify that the hardware/software has been set up properly. We'll also familiarize ourselves with the tools we'll be using for the rest of the workshop via a short program running on the MSP430F2013.

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**.



Code Composer Studio v4 Setup

Product Configuration

Select the Product Configuration that best suits your needs.

Select the Product Configuration that you wish to install. More detailed installation options will be presented on the next screen.

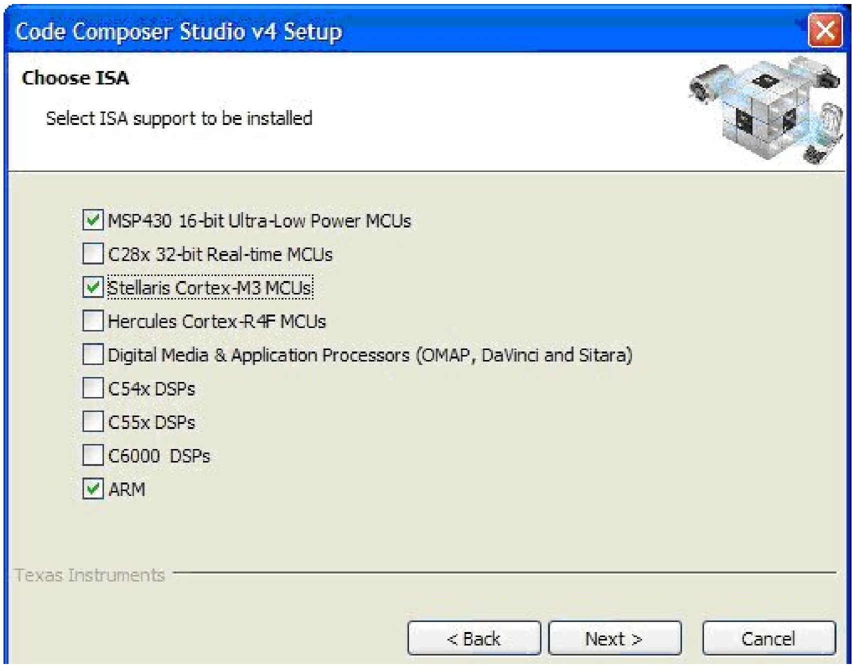
Platinum Edition
Microcontroller Edition
Scripting Tools
Custom

Description

This is a complete installation including all features and support for all device families.

By default a 30 day evaluation license will be generated.

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**.



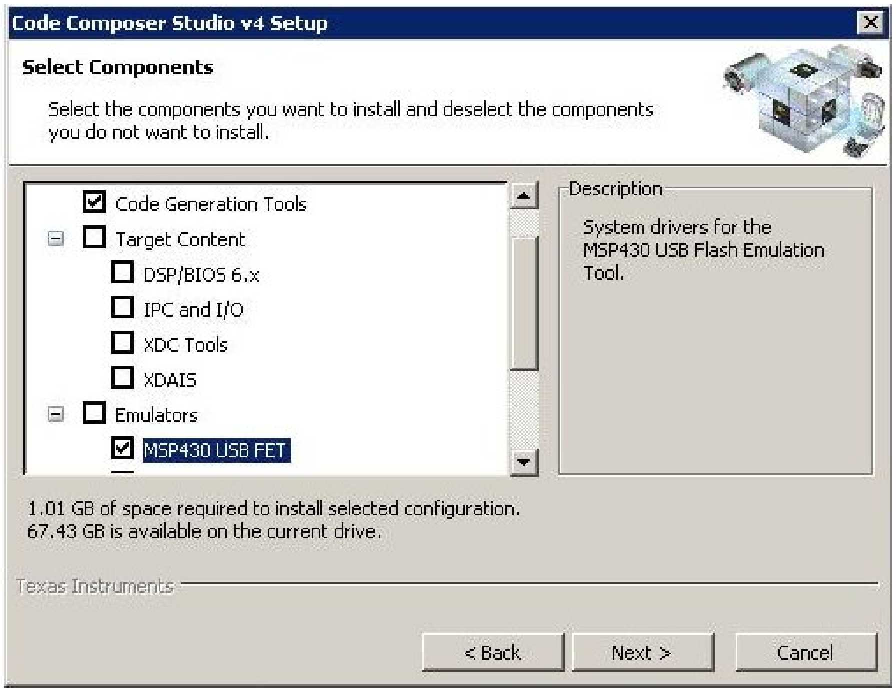
Code Composer Studio v4 Setup

Choose ISA

Select ISA support to be installed

☑ MSP430 16-bit Ultra-Low Power MCUs
□ C28x 32-bit Real-time MCUs
☑ Stellaris Cortex-M3 MCUs
□ Hercules Cortex-R4F MCUs
□ Digital Media & Application Processors (OMAP, DaVinci and Sitara)
□ C54x DSPs
□ C55x DSPs
□ C6000 DSPs
☑ ARM

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.



Code Composer Studio v4 Setup

Select Components

Select the components you want to install and deselect the components you do not want to install.

☑ Code Generation Tools
⊟ □ Target Content
  □ DSP/BIOS 6.x
  □ IPC and I/O
  □ XDC Tools
  □ XDAIS
⊟ □ Emulators
  ☑ MSP430 USB FET

Description

System drivers for the MSP430 USB Flash Emulation Tool.

1.01 GB of space required to install selected configuration.
67.43 GB is available on the current drive.

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**.

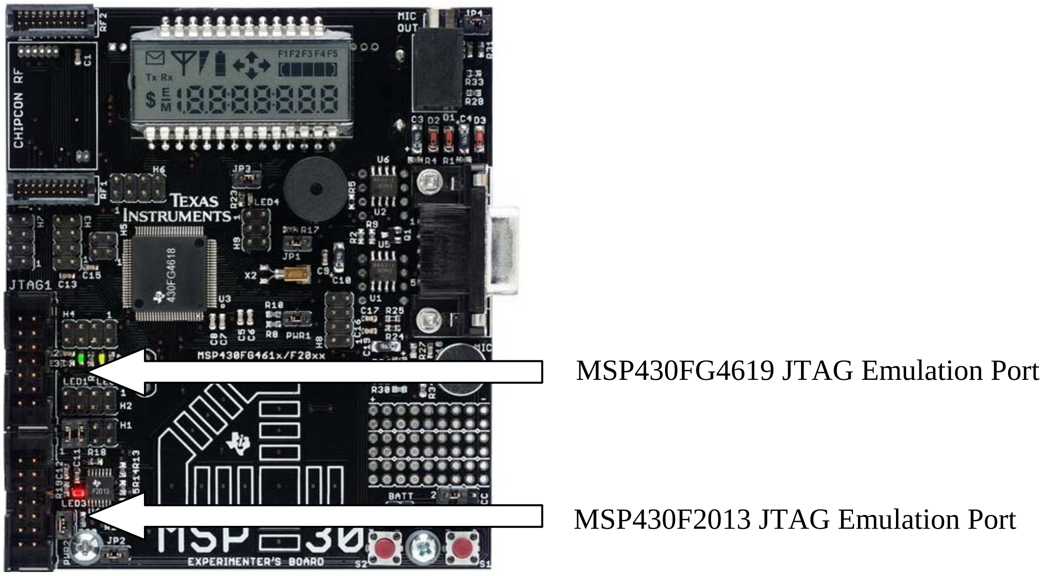
If you've been tasked with installing Code Composer, the drivers and labs only, please stop here and ask your instructor for further directions.

Hardware Verification

1. Check out the hardware

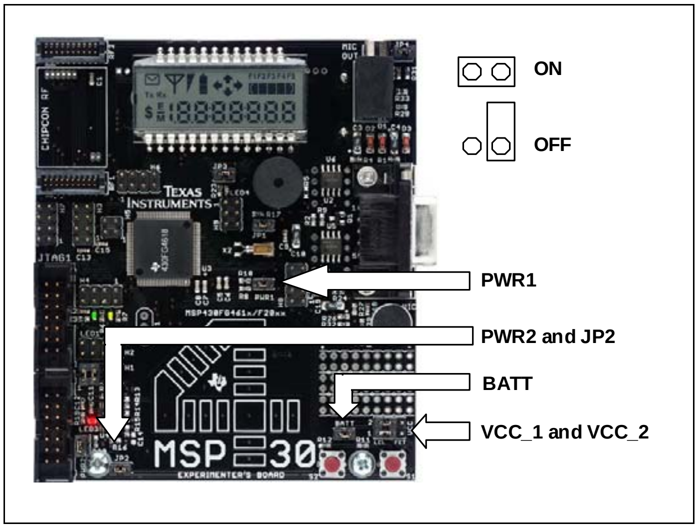
Make sure that the MSP430 USB FET is connected to the USB cable and that the other end of the cable is connected to the PC's USB port.

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).


MSP430FG4619 JTAG Emulation Port
MSP430F2013 JTAG Emulation Port

Power jumpers

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



Make sure the jumpers are set as follows:

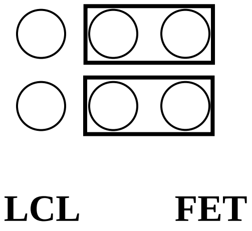
PWR1 controls power to the MSP430FG4619 (**ON**)

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**)

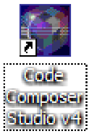
VCC_1 and **VCC_2** control whether the microcontrollers are powered by the emulator (FET) or the batteries (LCL). Since we'll be powering from the board from the emulator, place both jumpers over the rightmost two pins as shown:



CCS 4.1

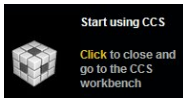
3. Start up the IDE

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


Code Composer Studio v4

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.


Start using CCS

Click to close and go to the CCS workbench

4. Create a New Project

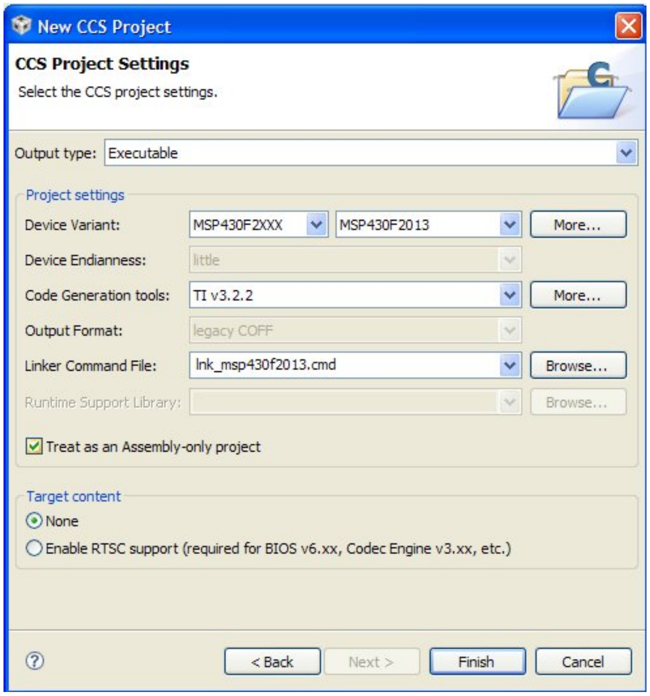
On the menu bar, click **File ⇨ New ⇨ CCS Project**. When the *New Project* dialogue appears, name the project **Lab1** and click **Next**. Note that the location is our Lab1 workspace folder.

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**.


New CCS Project

CCS Project Settings

Select the CCS project settings.

Output type: Executable

Project settings

Device Variant: MSP430F2XXX MSP430F2013

Device Endianness: little

Code Generation tools: TI v3.2.2

Output Format: legacy COFF

Linker Command File: lnk_msp430f2013.cmd

Runtime Support Library:

☑ Treat as an Assembly-only project

Target content

● None

○ Enable RTSC support (required for BIOS v6.xx, Codec Engine v3.xx, etc.)

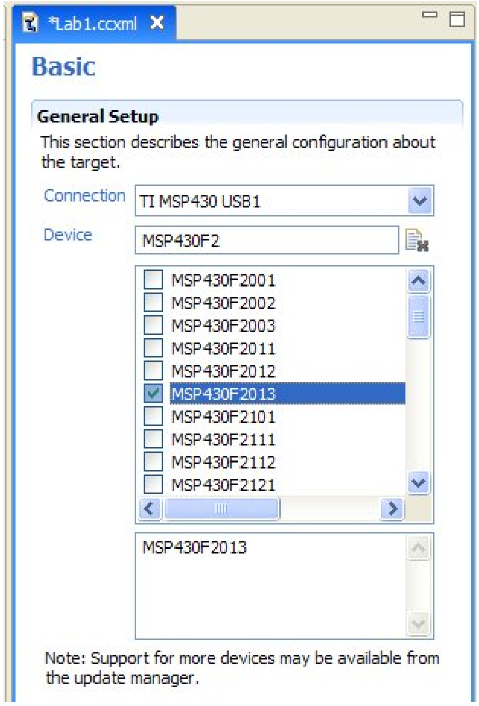
Configuring the Target

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:



*Lab1.ccxml

Basic

General Setup

This section describes the general configuration about the target.

Connection: TI MSP430 USB1

Device: MSP430F2

□ MSP430F2001
□ MSP430F2002
□ MSP430F2003
□ MSP430F2011
□ MSP430F2012
☑ MSP430F2013
□ MSP430F2101
□ MSP430F2111
□ MSP430F2112
□ MSP430F2121

MSP430F2013

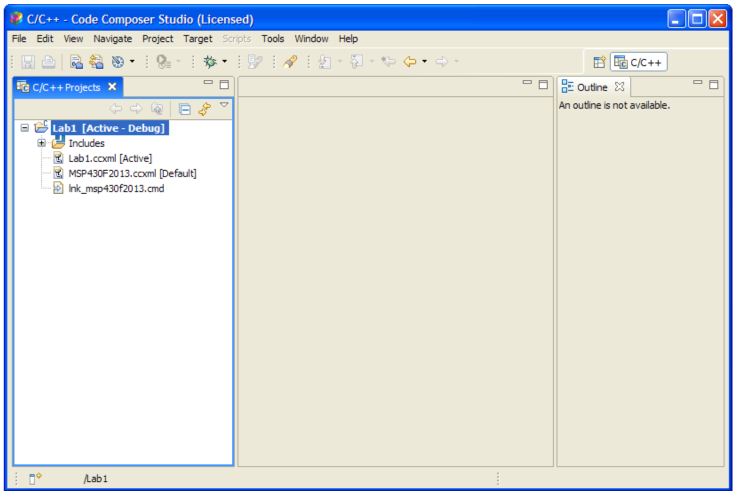
Note: Support for more devices may be available from the update manager.

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:


C/C++ - Code Composer Studio (Licensed)
File Edit View Navigate Project Target Scripts Tools Window Help
C/C++
C/C++ Projects
Lab1 [Active - Debug]
Includes
Lab1.ccxml [Active]
MSP430F2013.ccxml [Default]
lnk_msp430f2013.cmd
Outline
/Lab1

An outline is not available.

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                           ; Progam 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

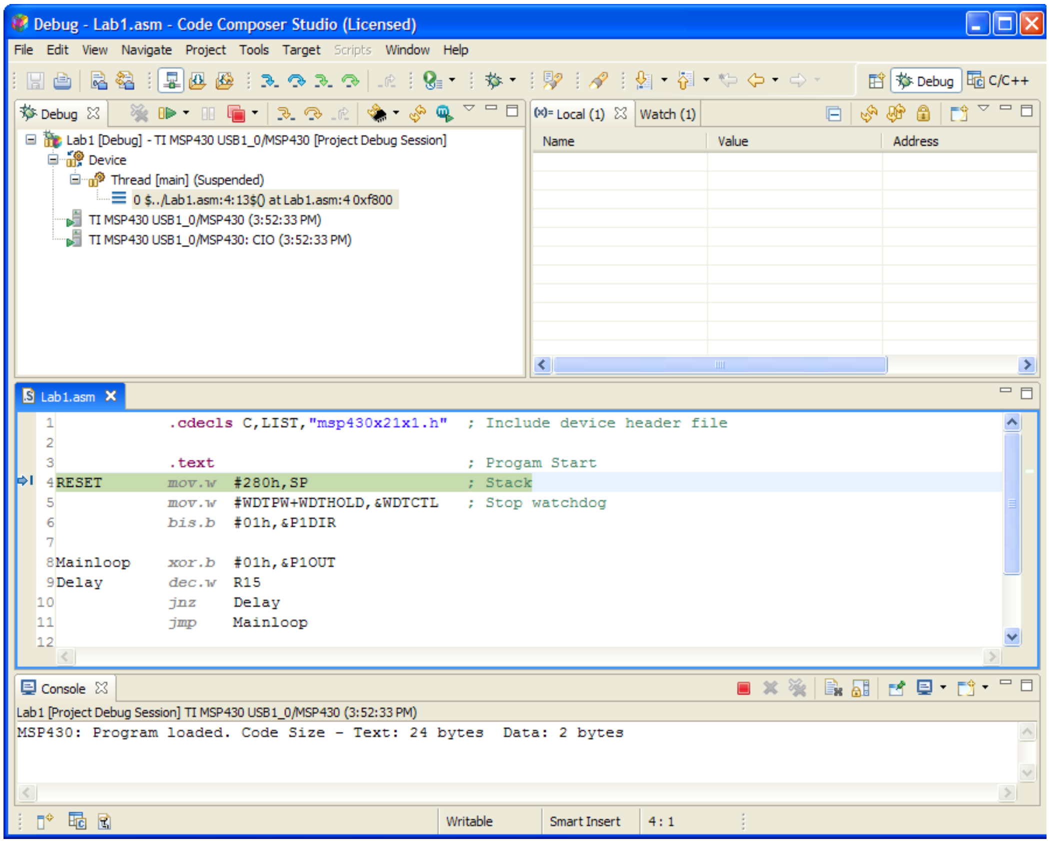
8. Assemble and Download

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

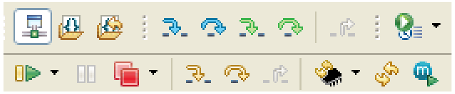
You should be looking at a screen that looks something like this:


Debug - Lab1.asm - Code Composer Studio (Licensed)
File Edit View Navigate Project Tools Target Scripts Window Help
Debug
C/C++
Debug
Lab1 [Debug] - TI MSP430 USB1_0/MSP430 [Project Debug Session]
Device
Thread [main] (Suspended)
0 $../Lab1.asm:4:13$() at Lab1.asm:4 0xf800
TI MSP430 USB1_0/MSP430 (3:52:33 PM)
TI MSP430 USB1_0/MSP430: CIO (3:52:33 PM)
Local (1)
Watch (1)
Name
Value
Address
Lab1.asm
Console
Writable
Smart Insert
4 : 1

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

            .text                           ; Progam 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
```

```
Lab1 [Project Debug Session] TI MSP430 USB1_0/MSP430 (3:52:33 PM)
MSP430: Program loaded. Code Size - Text: 24 bytes  Data: 2 bytes
```

The buttons on the top-left that look like this: control the running of the code. Click on the **Run** button 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 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.

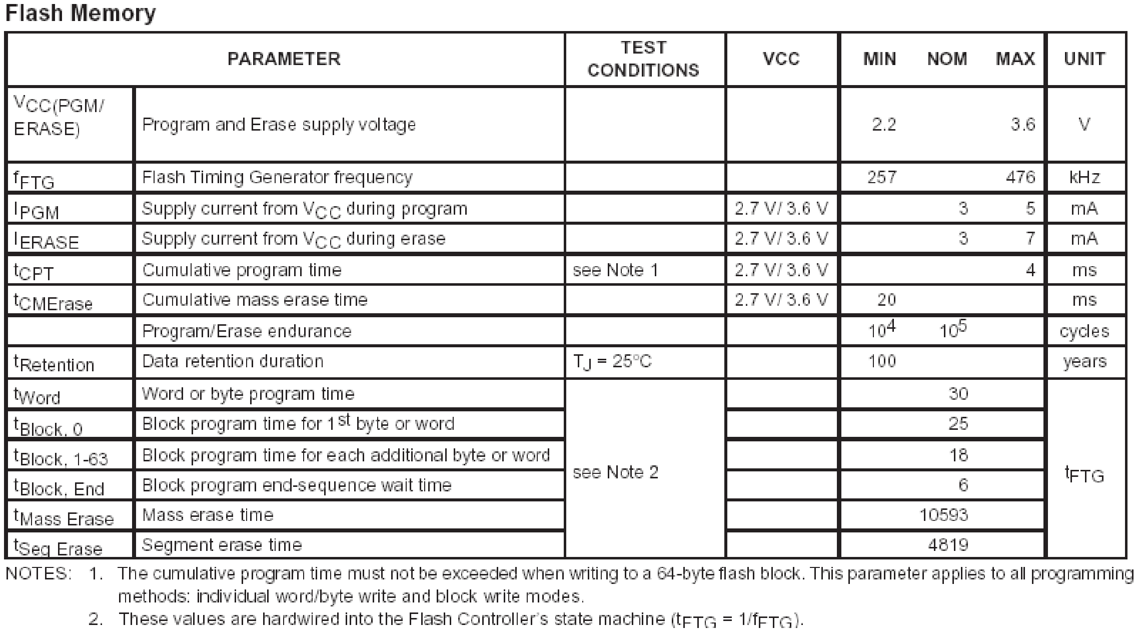
FLASH Programming Exercise

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

PARAMETER		TEST CONDITIONS	VCC	MIN	NOM	MAX	UNIT
$V_{CC(PGM/ERASE)}$	Program and Erase supply voltage			2.2		3.6	V
f_{FTG}	Flash Timing Generator frequency			257		476	kHz
I_{PGM}	Supply current from V_{CC} during program		2.7 V/ 3.6 V		3	5	mA
I_{ERASE}	Supply current from V_{CC} during erase		2.7 V/ 3.6 V		3	7	mA
t_{CPT}	Cumulative program time	see Note 1	2.7 V/ 3.6 V			4	ms
$t_{CMErase}$	Cumulative mass erase time		2.7 V/ 3.6 V	20			ms
	Program/Erase endurance			10^4	10^5		cycles
$t_{Retention}$	Data retention duration	T_J = 25°C		100			years
t_{Word}	Word or byte program time	see Note 2			30		t_{FTG}
$t_{Block, 0}$	Block program time for 1st byte or word				25		
$t_{Block, 1-63}$	Block program time for each additional byte or word				18		
$t_{Block, End}$	Block program end-sequence wait time				6		
$t_{Mass Erase}$	Mass erase time				10593		
$t_{Seg Erase}$	Segment erase time				4819		

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 ____

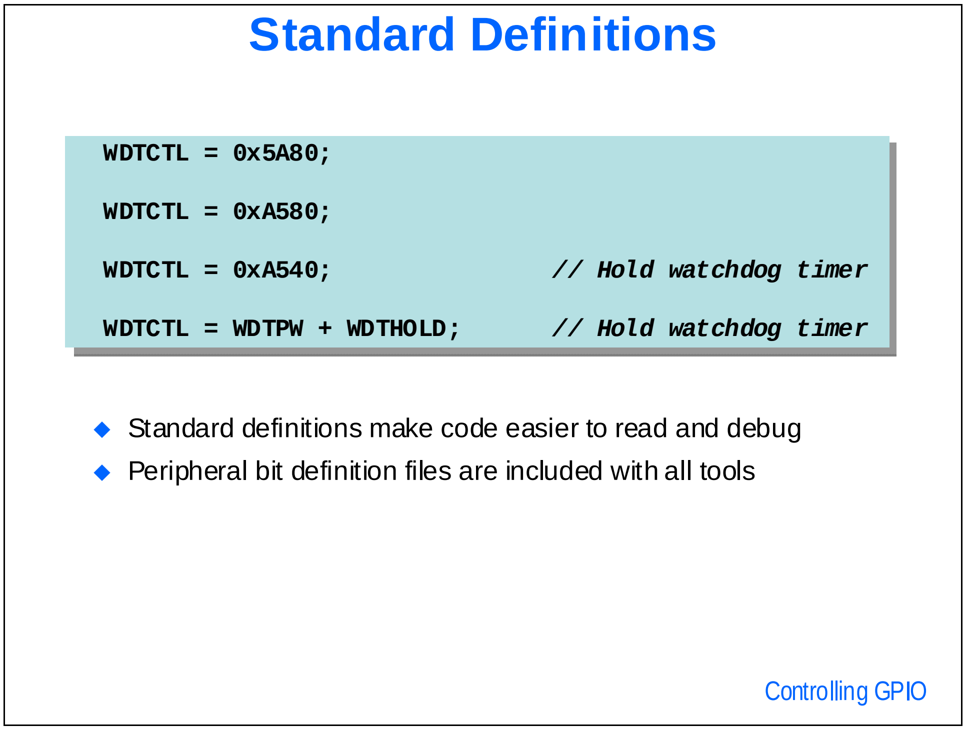
Multiply that by 1024 words _______________________________

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

CCS users … you're done

*** This page left blank by order of the fire marshal ***

Standard Definitions



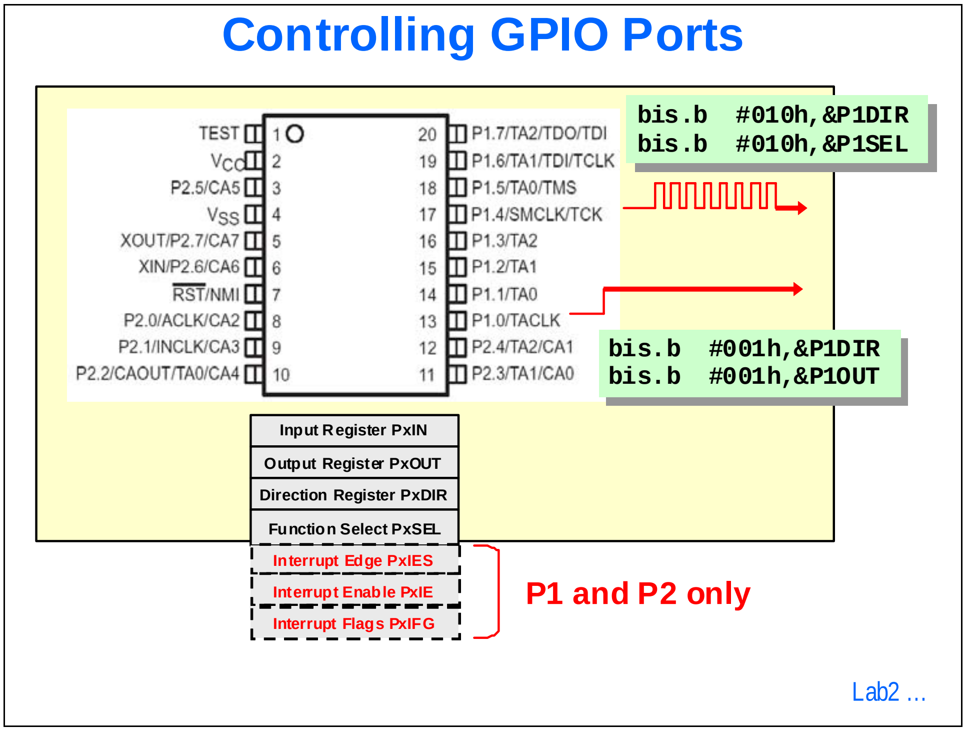
Standard Definitions

```
WDTCTL = 0x5A80;

WDTCTL = 0xA580;

WDTCTL = 0xA540;                // Hold watchdog timer

WDTCTL = WDTPW + WDTHOLD;       // 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



Controlling GPIO Ports

```
bis.b  #010h,&P1DIR
bis.b  #010h,&P1SEL
```

```
bis.b  #001h,&P1DIR
bis.b  #001h,&P1OUT
```

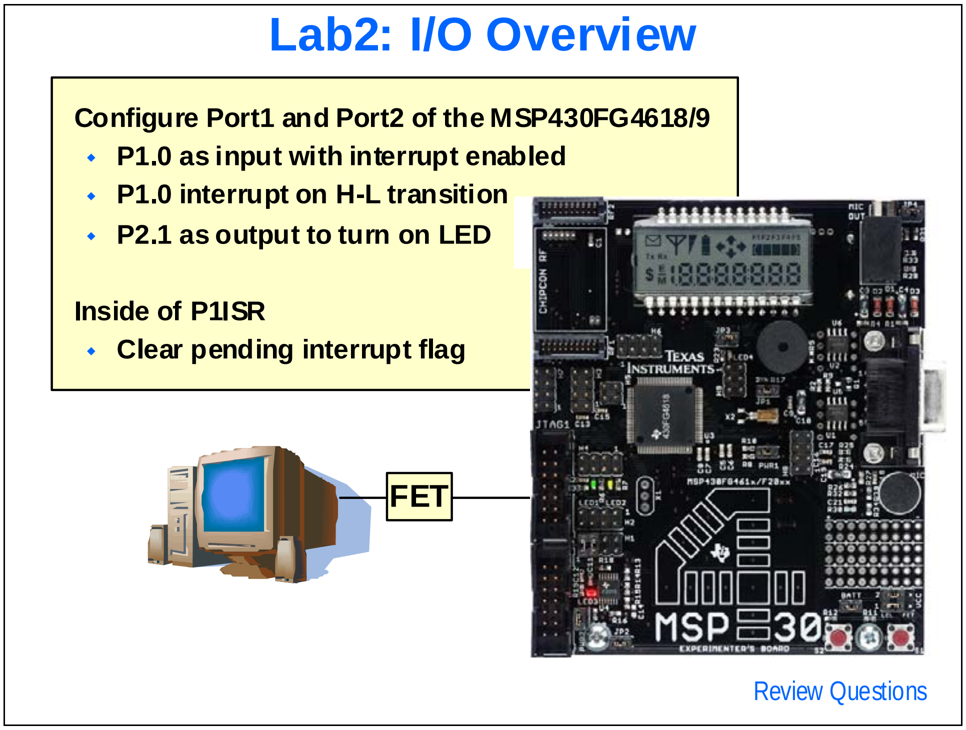
P1 and P2 only

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

FET
MSP430 EXPERIMENTER'S BOARD
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 Experimenter’s Board and **connect** it to the MSP430FG4619 port as shown on page 1-19. The red LED next to the MSP430F2013 emulator port should start blinking again. After all, the program is still in flash memory and you just applied power to the part …

2. Start IAR Kickstart

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

New Workspace and Project

3. New Workspace

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

4. New Project

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

Configure the Project

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

Feel free to write it down here ______________________

5. Configure the Project

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

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

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

Add Source File

6. Add the source file to the project

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

Complete the Code

7. Answer some questions

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

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

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

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

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

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

```
#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.

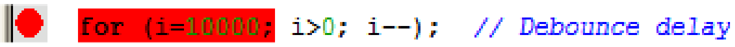
11. Some Debugging Fun

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.

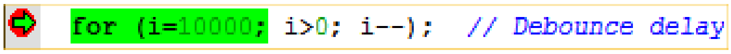
Reset the CPU by clicking the Reset button .

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.

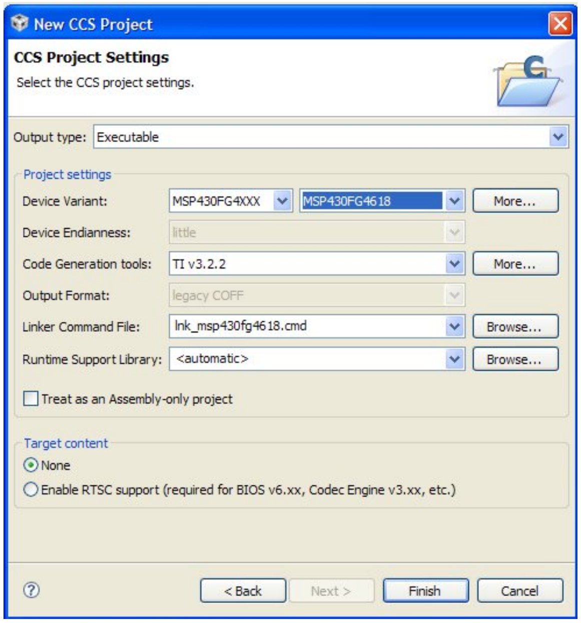
3. New Project

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 <u>not</u> be an assembly project.



New CCS Project

CCS Project Settings

Select the CCS project settings.

Output type: Executable

Project settings

Device Variant: MSP430FG4XXX MSP430FG4618

Device Endianness: little

Code Generation tools: TI v3.2.2

Output Format: legacy COFF

Linker Command File: lnk_msp430fg4618.cmd

Runtime Support Library: <automatic>

□ Treat as an Assembly-only project

Target content

● None

○ Enable RTSC support (required for BIOS v6.xx, Codec Engine v3.xx, etc.)

Add a Source File

4. Add the source file to the project

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

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

Complete the Code

5. Answer some questions

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

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

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

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

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

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

```
#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.

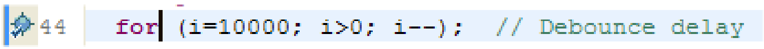
9. Some Debugging Fun

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.

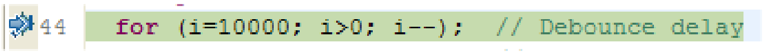
Reset the CPU by clicking the **Reset CPU** button.

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:



```
44    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:



```
44    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

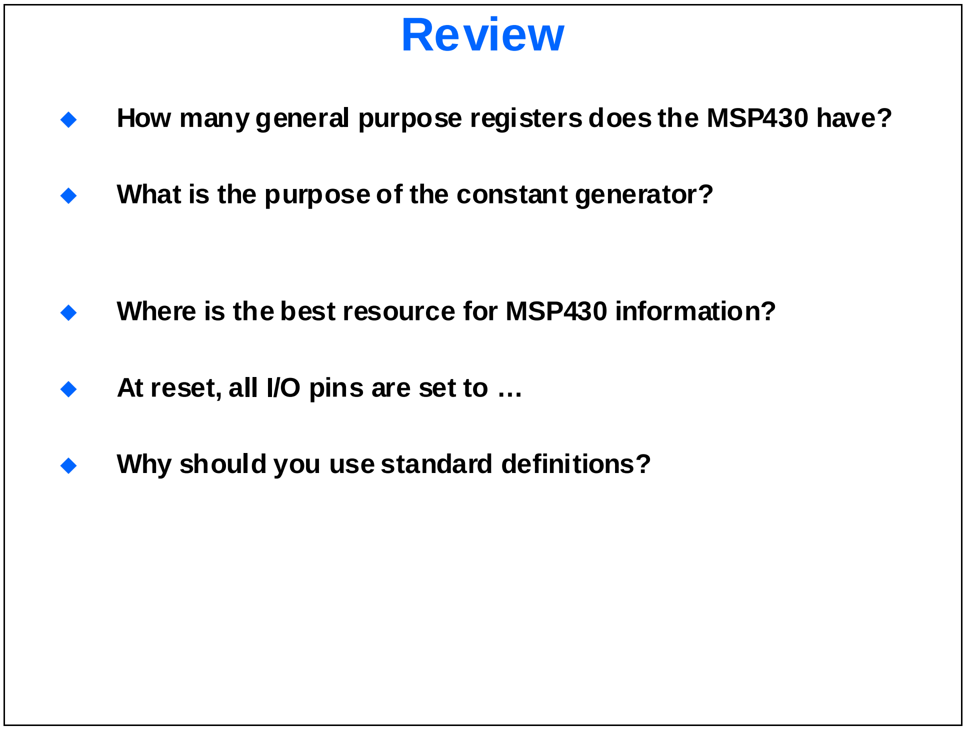
Start the code running again by clicking the **Run** button, then press S1. Unless you continue pressing **S1** when you click **Run**, the LED won't toggle since the IF statement didn't detect **S1** being pressed. Try this a few times, and notice the register values change. You may want to set other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

Ultra-Low Power

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

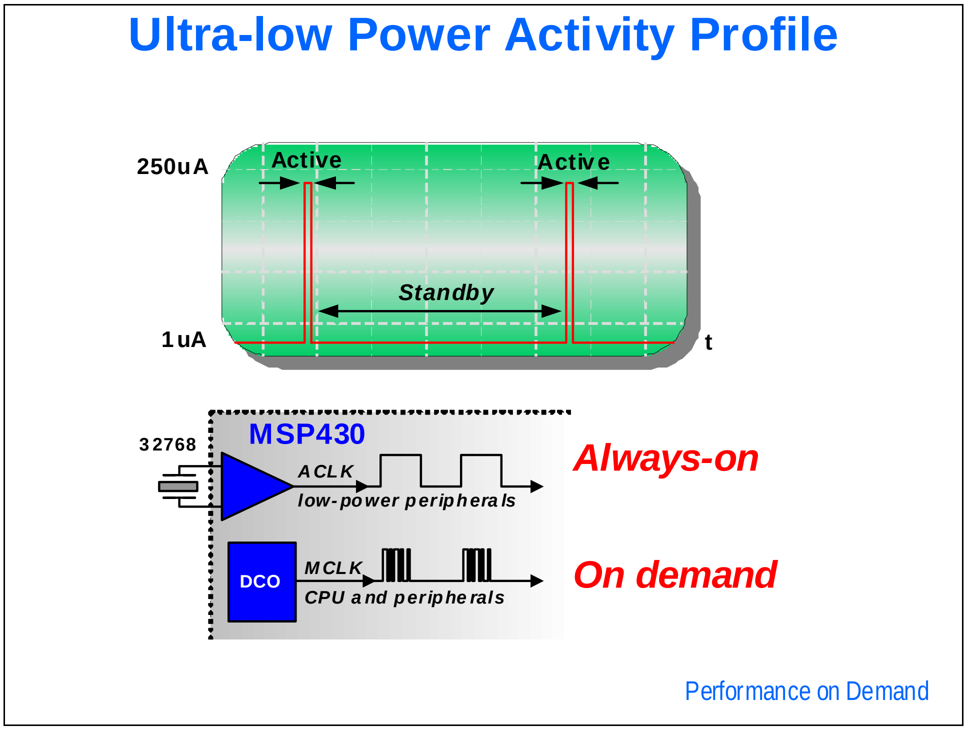
- Principles of ultra-low power applications
- Low power modes
- Oscillators
- Interrupts
- Ultra-low power lab

*** This page isn’t really blank, you know. ***

Module Topics

*** Let this be your doodle area ***

Activity Profile



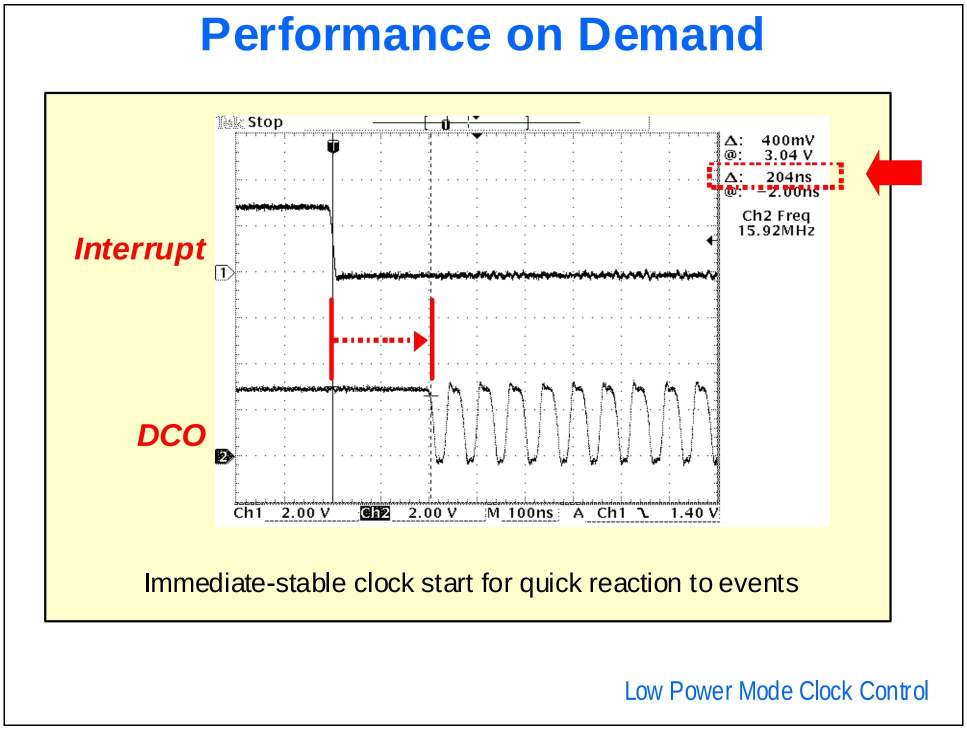
Ultra-low Power Activity Profile

Always-on

On demand

Performance on Demand

Performance on Demand

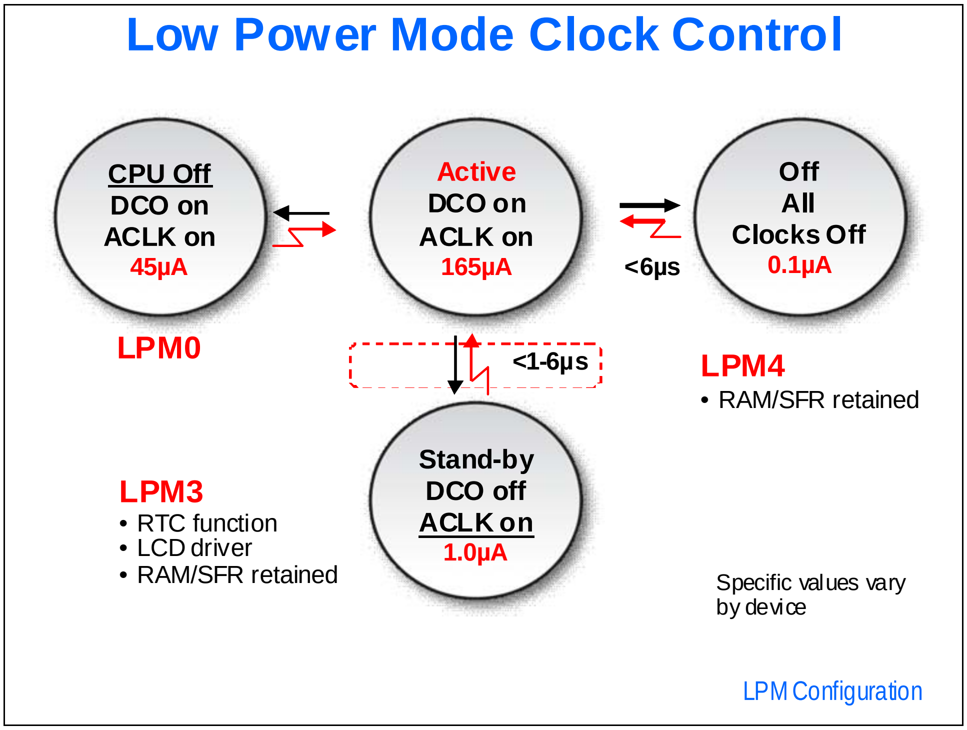


Performance on Demand

Immediate-stable clock start for quick reaction to events

Low Power Mode Clock Control

Low Power Mode Clock Control



Low Power Mode Clock Control

CPU Off
DCO on
ACLK on
45µA

LPM0

Active
DCO on
ACLK on
165µA

<6µs

Off
All
Clocks Off
0.1µA

<1-6µs

LPM4
- RAM/SFR retained

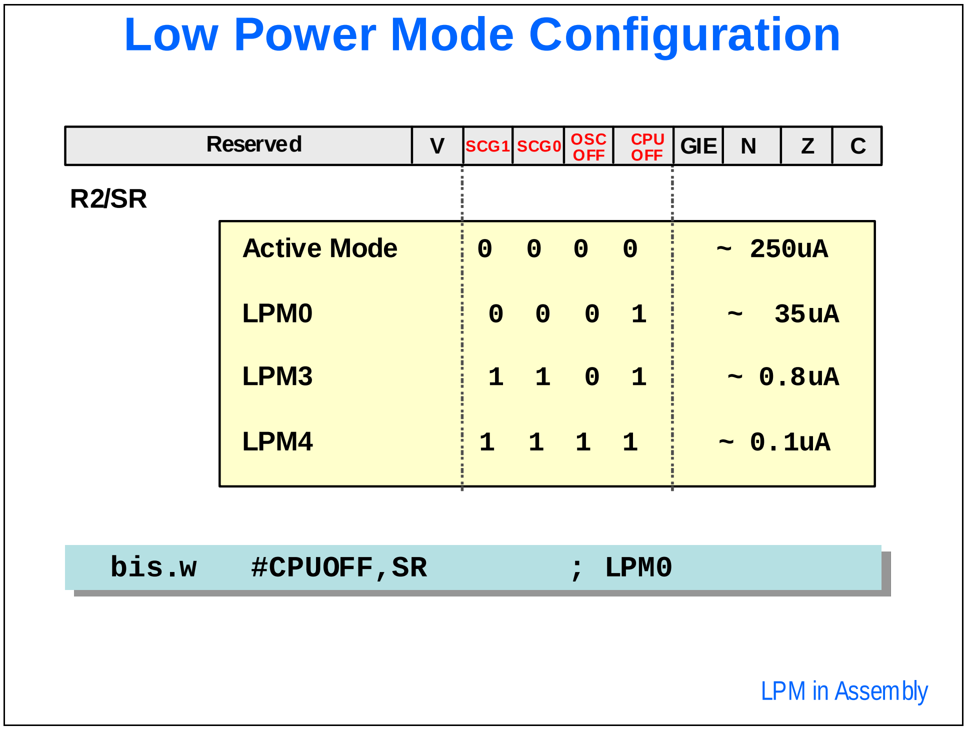
Stand-by
DCO off
ACLK on
1.0µA

LPM3
- RTC function
- LCD driver
- RAM/SFR retained

Specific values vary by device

LPM Configuration

Low Power Mode Configuration



Low Power Mode Configuration

Reserved	V	SCG1	SCG0	OSC OFF	CPU OFF	GIE	N	Z	C

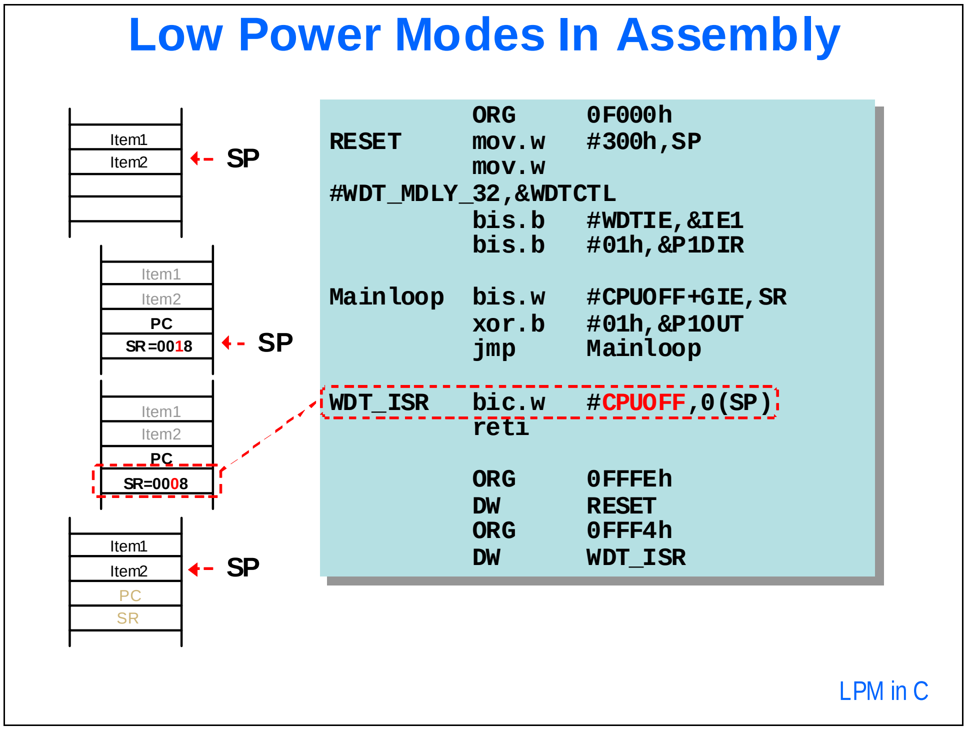
R2/SR

Active Mode	0	0	0	0	~ 250uA
LPM0	0	0	0	1	~ 35uA
LPM3	1	1	0	1	~ 0.8uA
LPM4	1	1	1	1	~ 0.1uA

```
bis.w   #CPUOFF,SR          ; LPM0
```

LPM in Assembly

Low Power Modes in Assembly

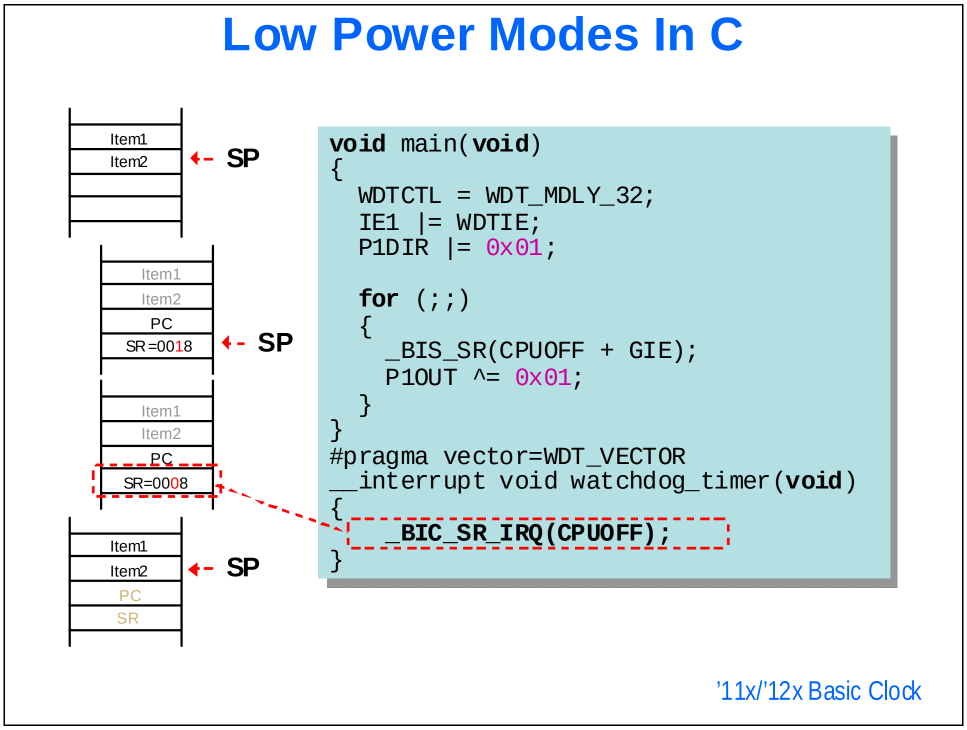


Low Power Modes In Assembly

```
          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
          DW      WDT_ISR
```

LPM in C

Low Power Modes in C



Low Power Modes In 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);
}
```

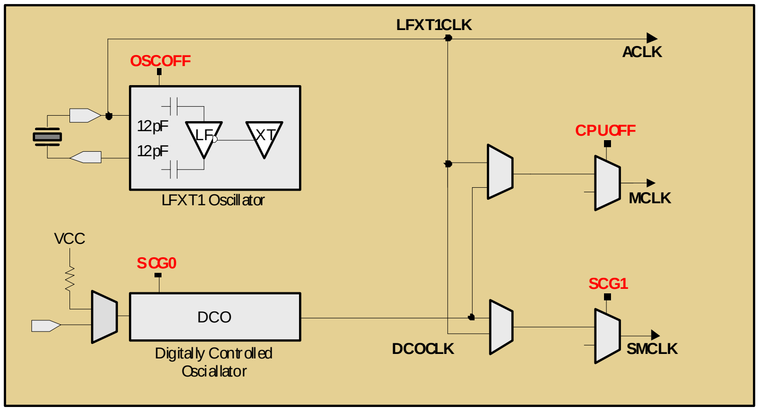
'11x/'12x Basic Clock

11x/12x Basic Clock

MSP430x11x/12x Basic Clock

Reserved	V	SCG1	SCG0	OSC OFF	CPU OFF	GIE	N	Z	C

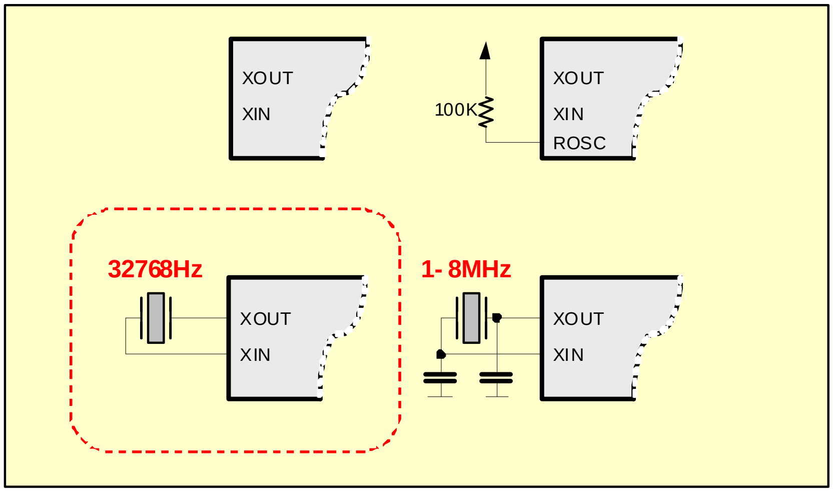
R2/SR



'1xx Basic Clock XTAL options

1xx XTAL Options

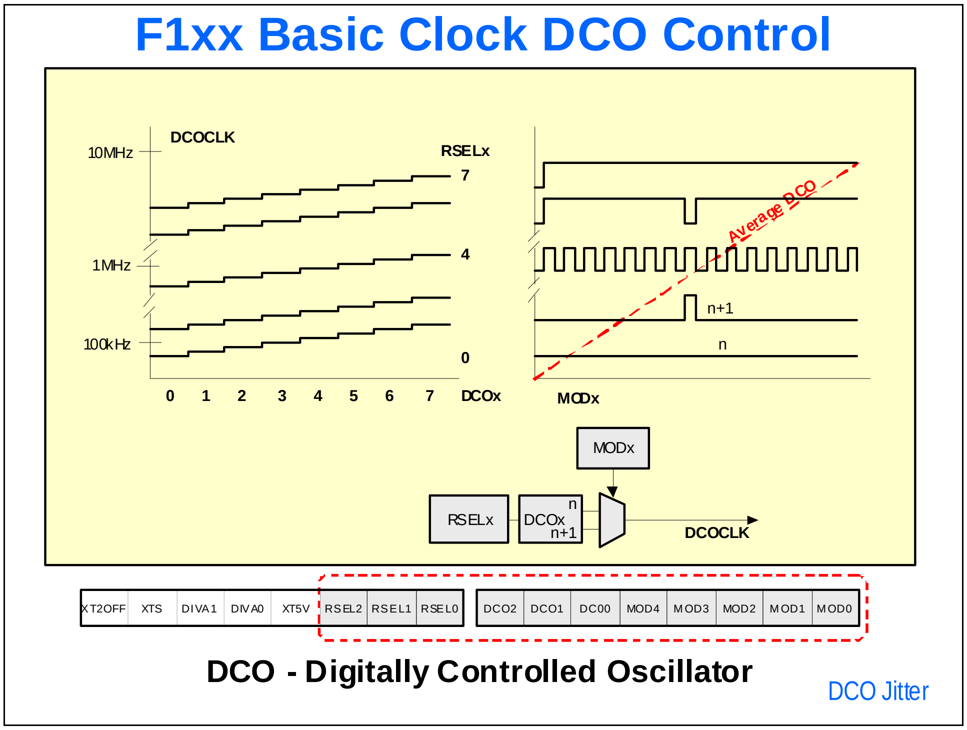
F1xx Basic Clock XTAL Options



Most MSP430 applications use a 32,768Hz crystal

'1xx Basic Clock DCO Control

1xx DCO Control



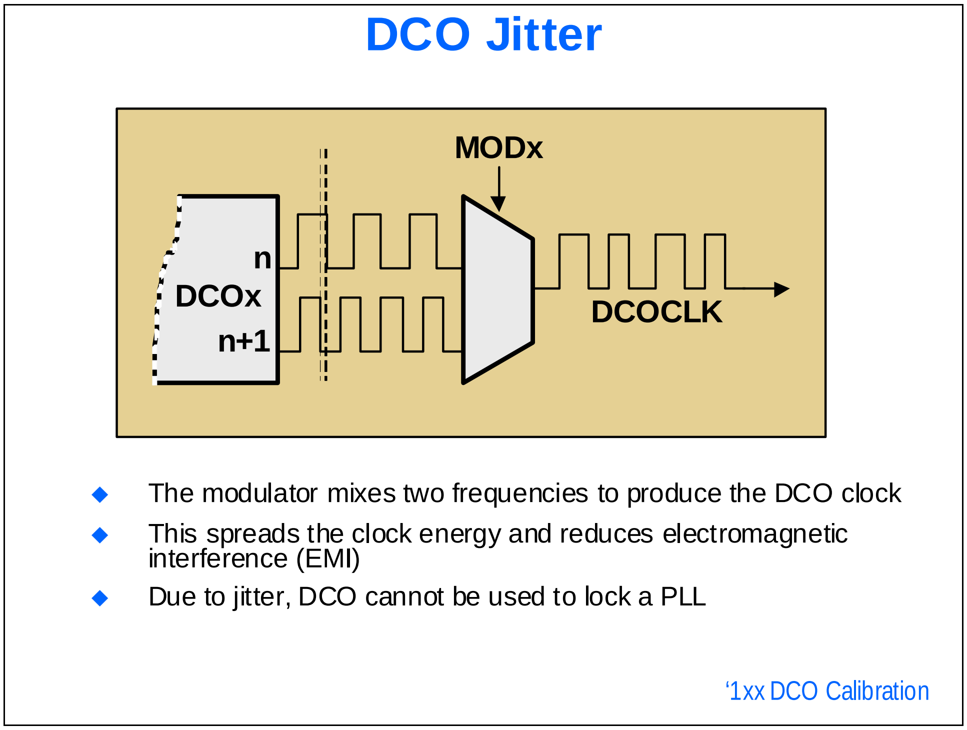
F1xx Basic Clock DCO Control

XT2OFF	XTS	DIVA1	DIVA0	XT5V	RSEL2	RSEL1	RSEL0	DCO2	DCO1	DCO0	MOD4	MOD3	MOD2	MOD1	MOD0

DCO - Digitally Controlled Oscillator

DCO Jitter

DCO Jitter

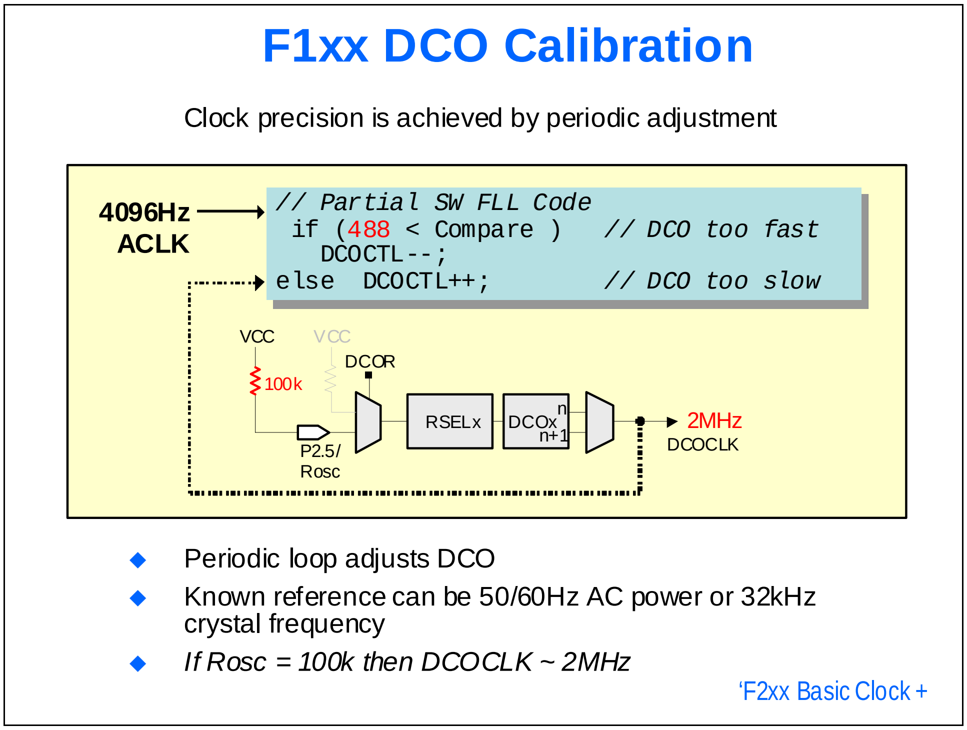


DCO Jitter

- 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

'1xx DCO Calibration

1xx DCO Calibration



F1xx DCO Calibration

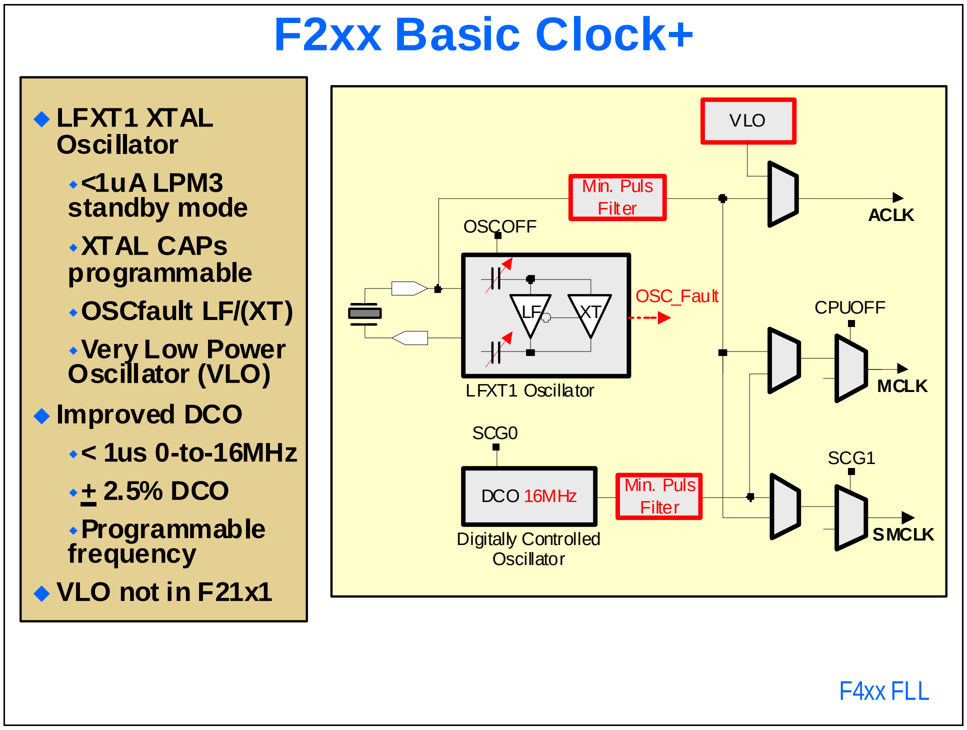
Clock precision is achieved by periodic adjustment

```
// 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*

'F2xx Basic Clock +

2xx Basic Clock

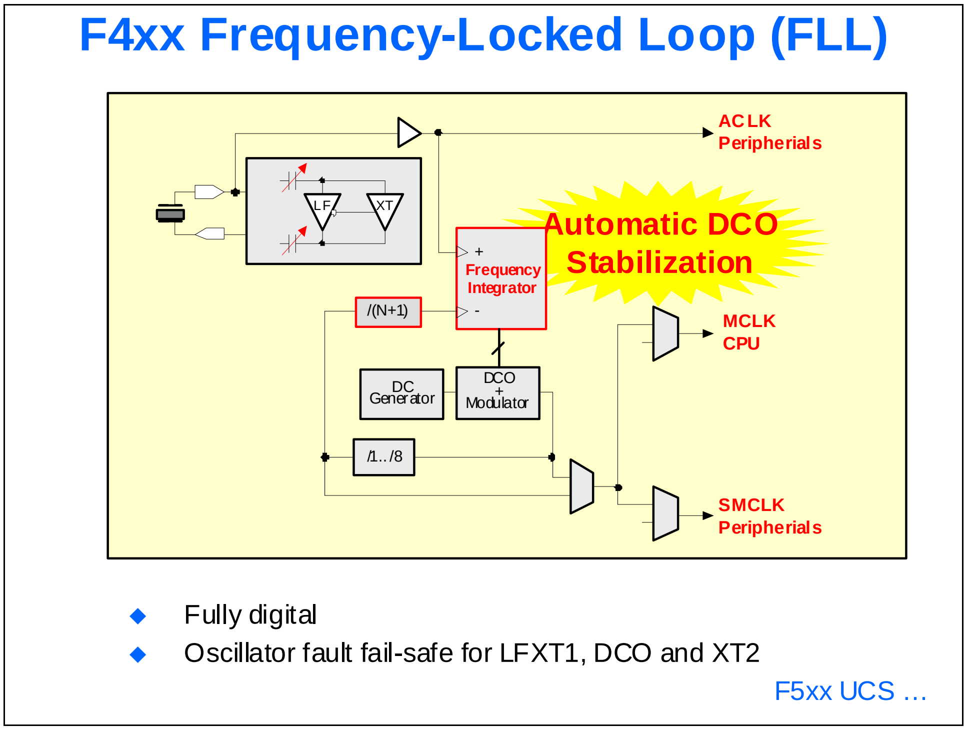


F2xx Basic Clock+

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

F4xx FLL

4xx FLL

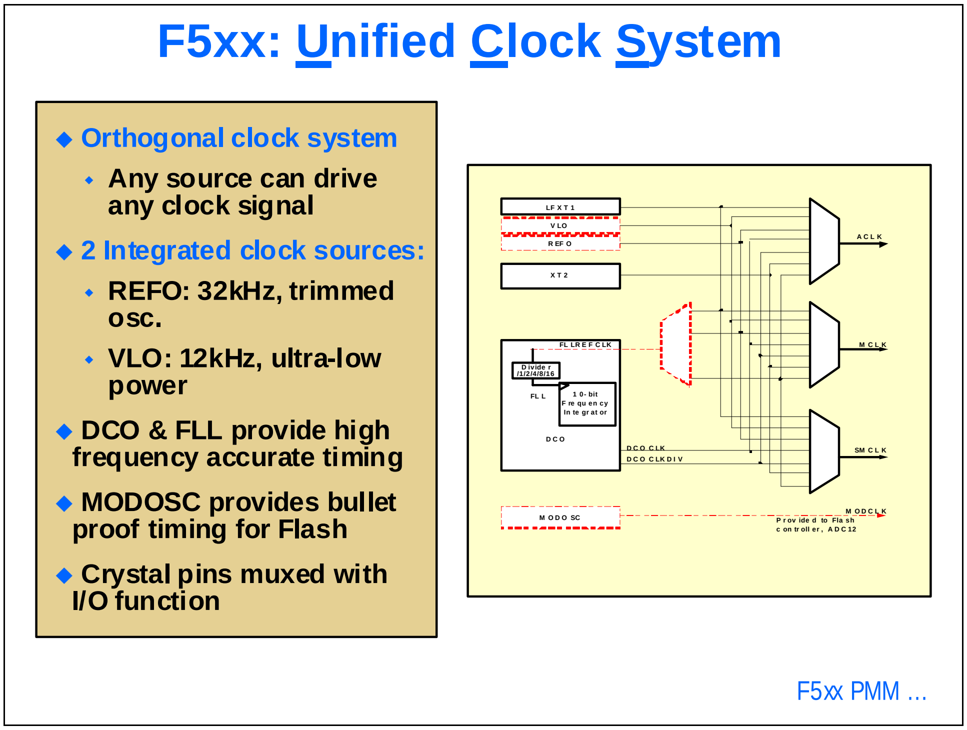


F4xx Frequency-Locked Loop (FLL)

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

F5xx UCS …

5xx Unified Clock System

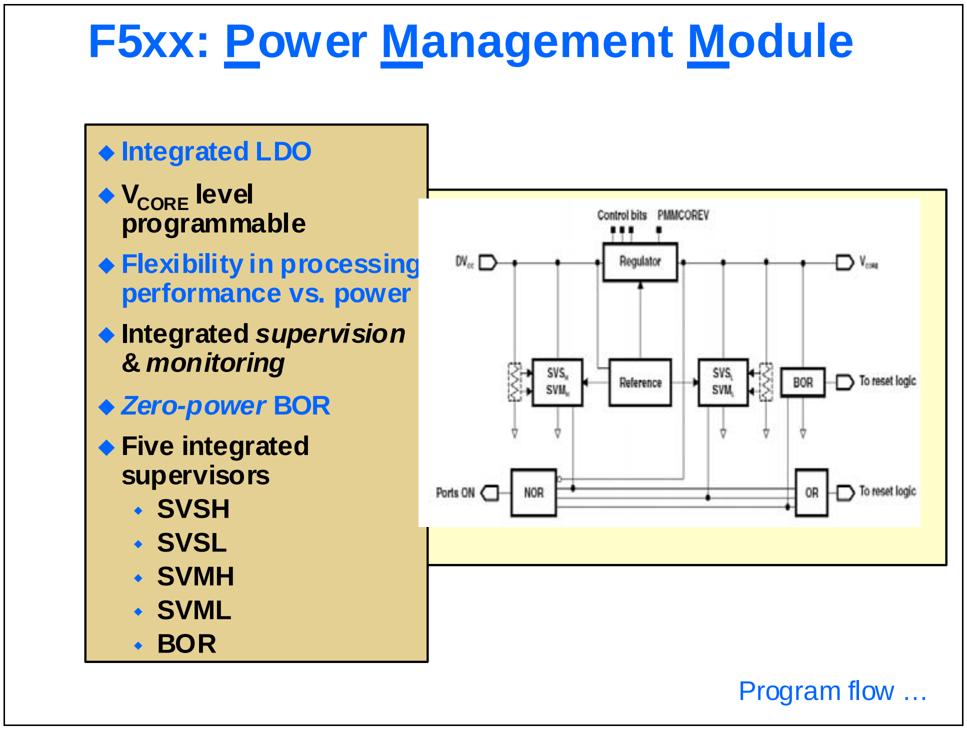


F5xx: Unified Clock System

- 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

F5xx PMM …

5xx Power Management Module

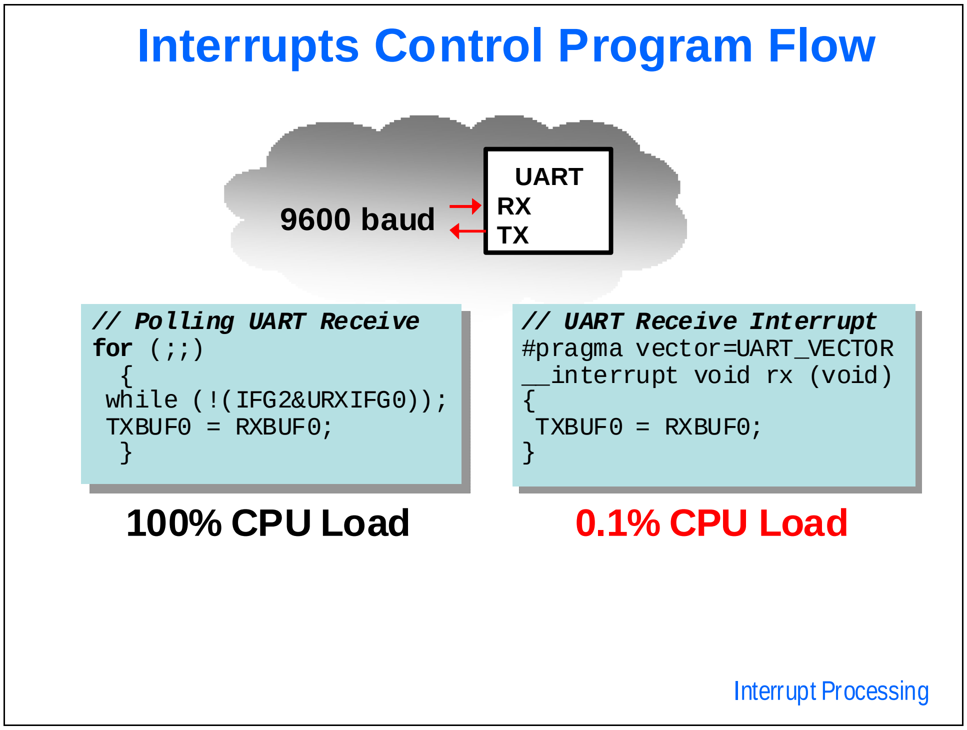


F5xx: Power Management Module

- Integrated LDO
- V_{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 …

Program Flow



Interrupts Control Program Flow

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

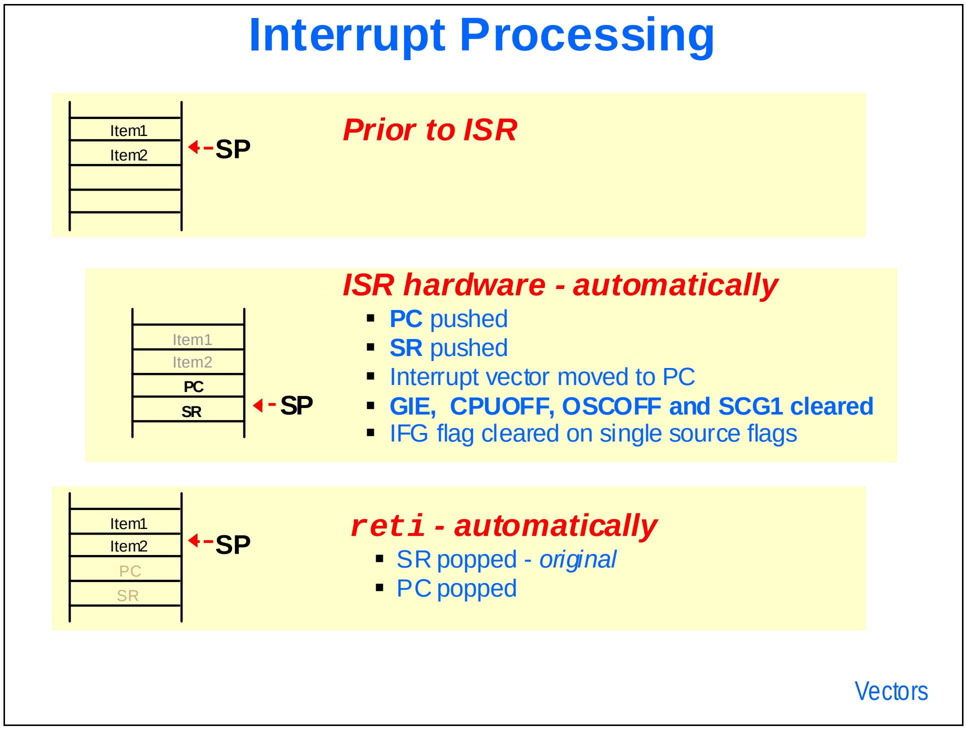
100% CPU Load

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

0.1% CPU Load

Interrupt Processing

Interrupt Processing



Interrupt Processing

Prior to ISR

ISR hardware - automatically

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

`reti` - automatically

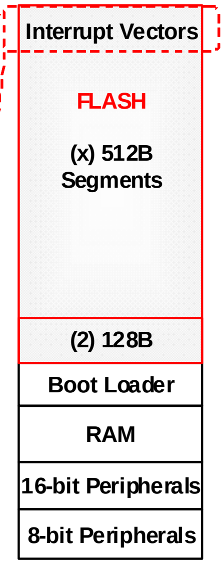
- SR popped - *original*
- PC popped

Vectors

11x1 Interrupt Vectors

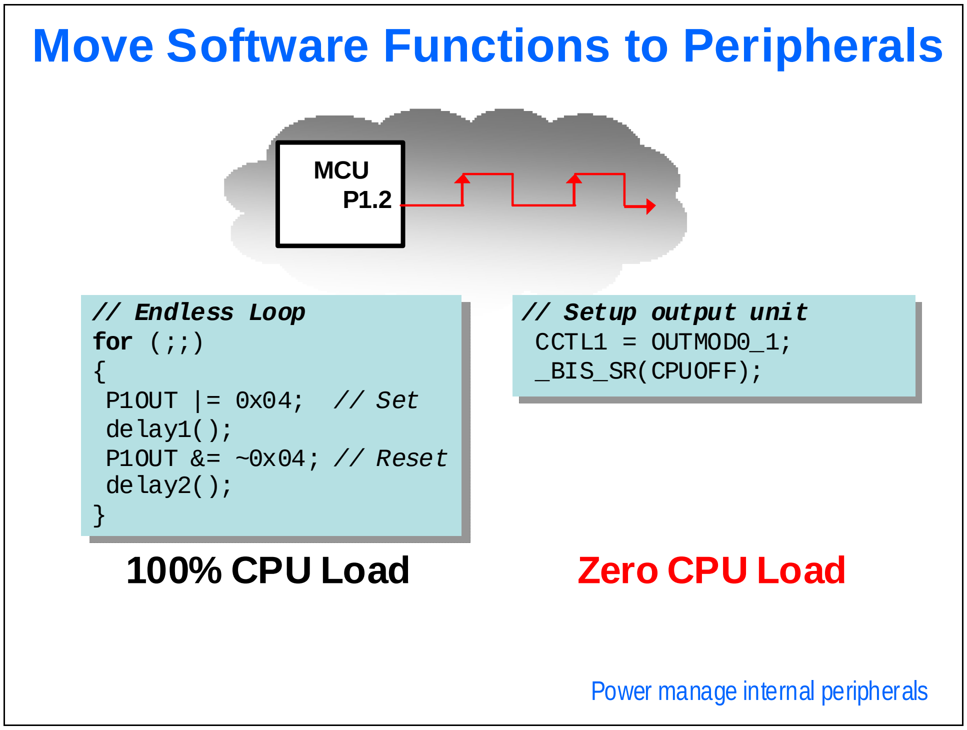
Interrupt Vectors – F11x1

SOURCE	FLAG	INTERRUPT	ADDRESS	PRIORITY
Power-up ext. Reset Watchdog	WDTIFG	Reset	0FFFEh	15, highest
NMI Osc. Fault Flash violation	NMIIFG OFIFG ACCVIFG	(non)-maskable (non)-maskable (non)-maskable	0FFFCh	14
			0FFFAh	13
			0FFF8h	12
Comparator_A	CAIFG	maskable	0FFF6h	11
Watchdog timer	WDTIFG	maskable	0FFF4h	10
Timer_A	CCIFG0	maskable	0FFF2h	9
Timer_A	CCIFGx	maskable	0FFF0h	8
			0FFEEh	7
			0FFECh	6
			0FFEAh	5
			0FFE8h	4
I/O Port P2	P2IFGx	maskable	0FFE6h	3
I/O Port P1	P1IFGx	maskable	0FFE4h	2
			0FFE2h	1
			0FFE0h	0, lowest



Move s/w to peripherals

Move S/W Functions to Peripherals



Move Software Functions to Peripherals

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

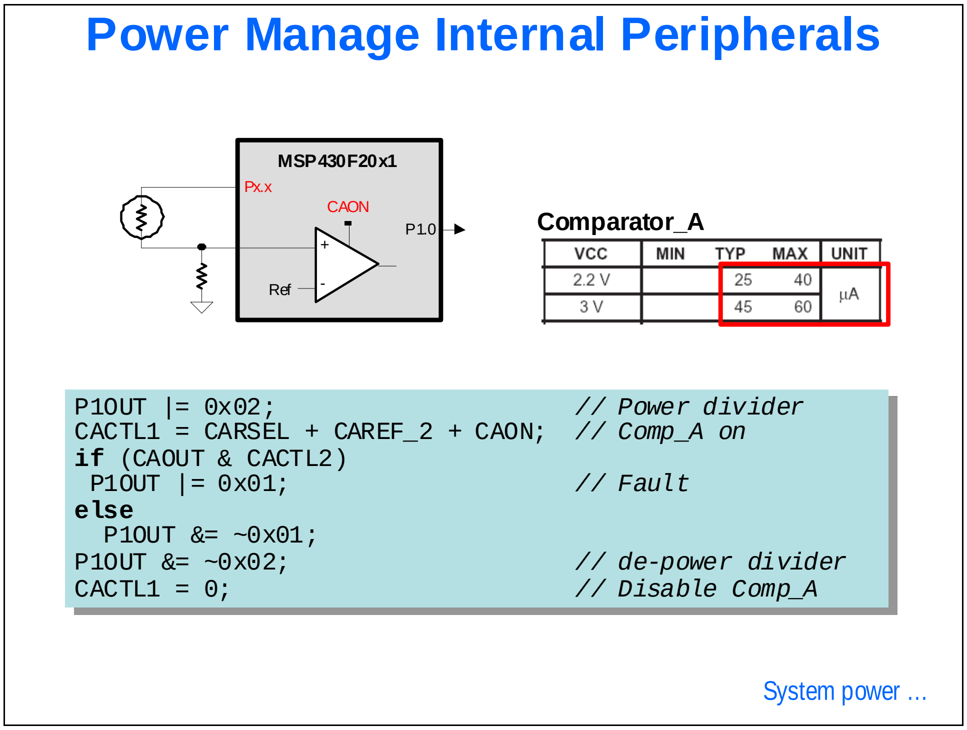
```
// Setup output unit
 CCTL1 = OUTMOD0_1;
 _BIS_SR(CPUOFF);
```

100% CPU Load

Zero CPU Load

Power manage internal peripherals

Power Manage Internal Peripherals



Power Manage Internal Peripherals

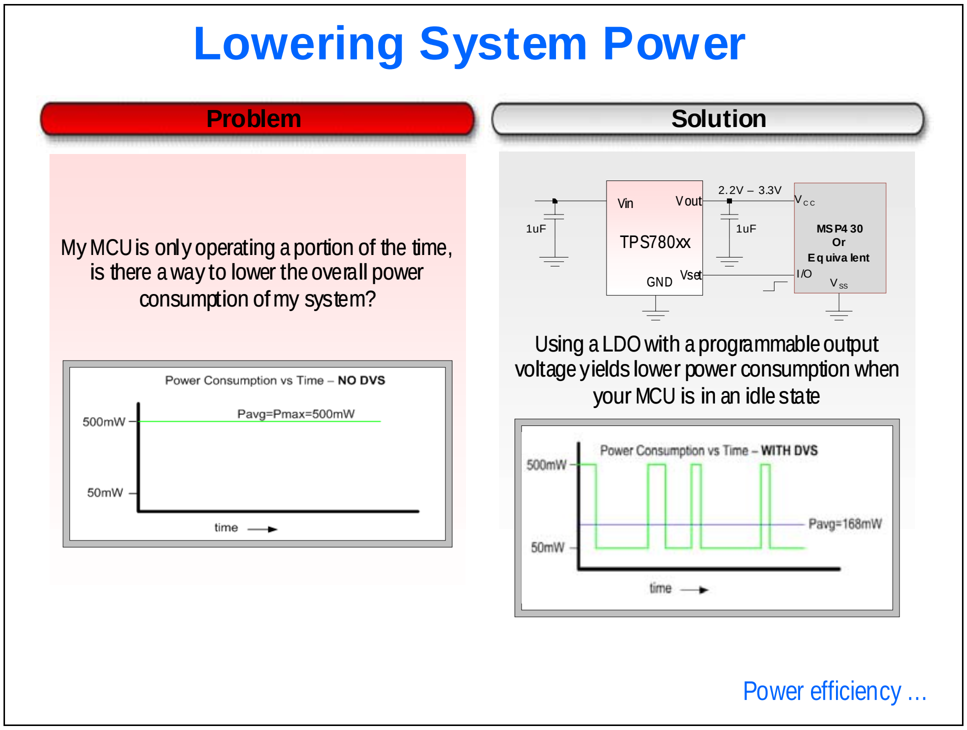
Comparator_A

VCC	MIN	TYP	MAX	UNIT
2.2 V		25	40	µA
3 V		45	60	

```
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
```

System power …

Lowering System Power



Lowering System Power

Problem

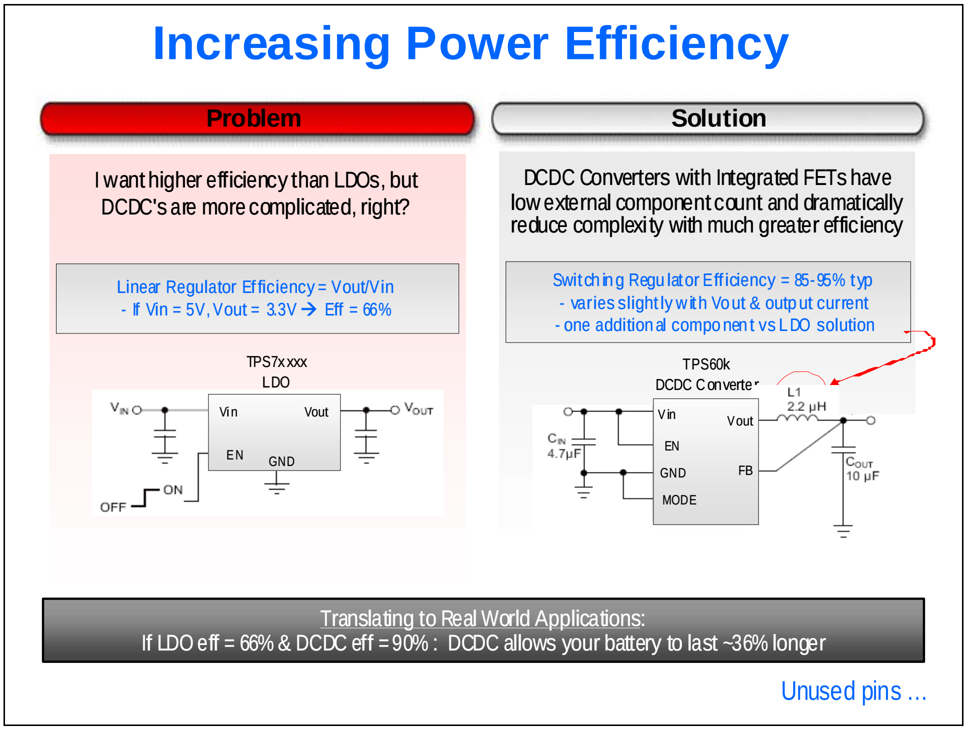
My MCU is only operating a portion of the time, is there a way to lower the overall power consumption of my system?

Solution

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

Power efficiency …

Increasing Power Efficiency



Increasing Power Efficiency

Problem

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

Linear Regulator Efficiency = Vout/Vin
- If Vin = 5V, Vout = 3.3V → Eff = 66%

Solution

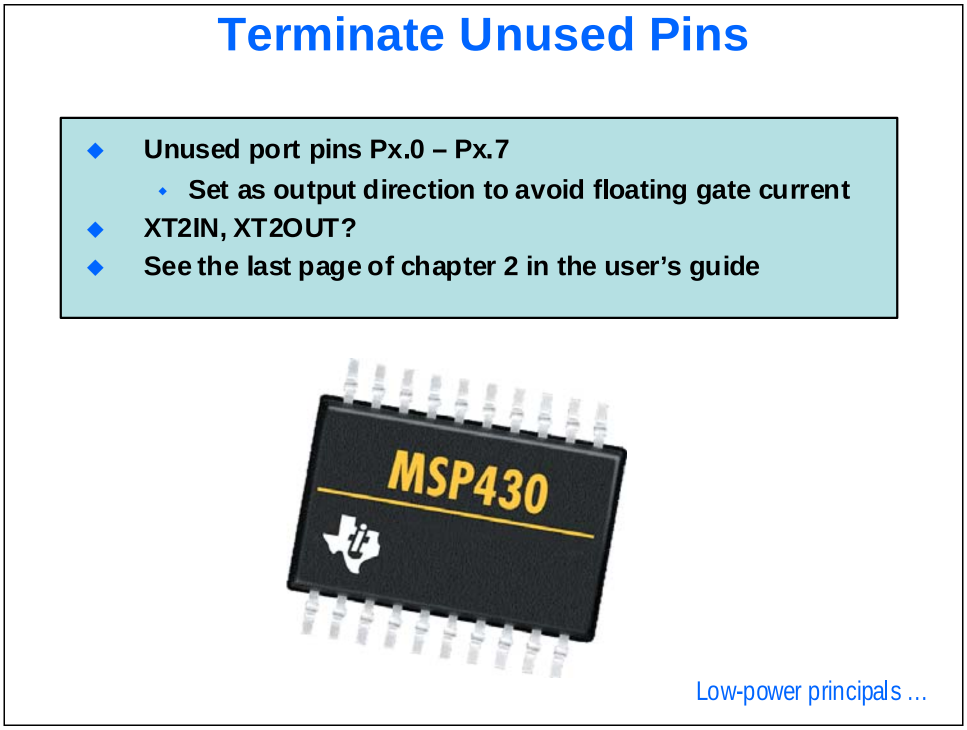
DCDC Converters with Integrated FETs have low external component count and dramatically reduce complexity with much greater efficiency

Switching Regulator Efficiency = 85-95% typ
- varies slightly with Vout & output current
- one additional component vs LDO solution

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

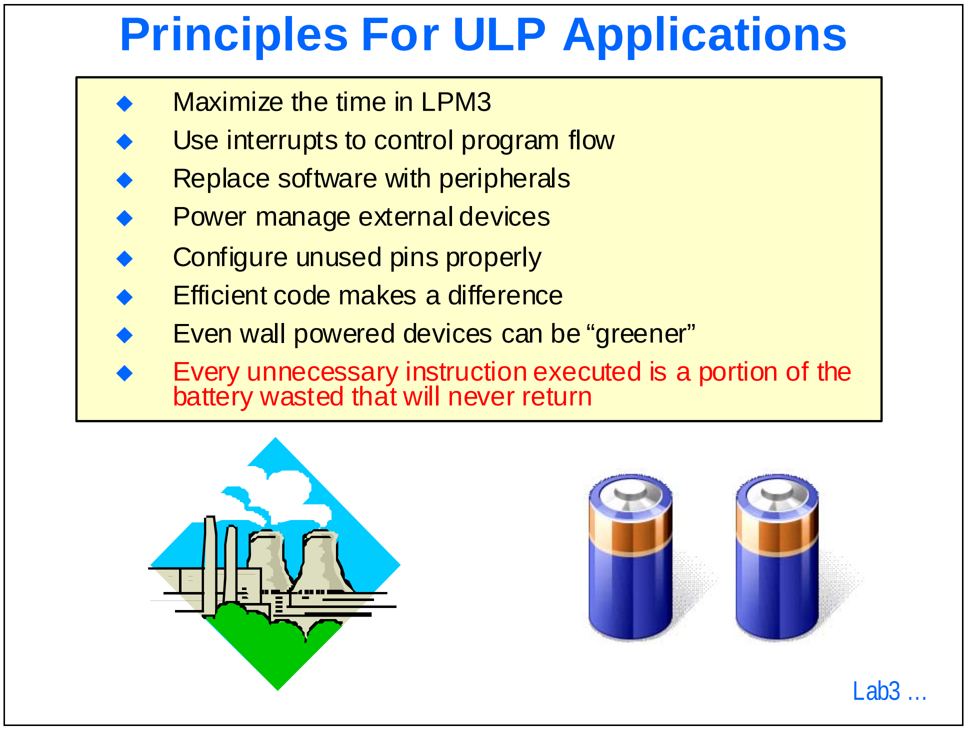


Terminate Unused Pins

- 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

Low-power principals ...

Ultra Low Power Principles



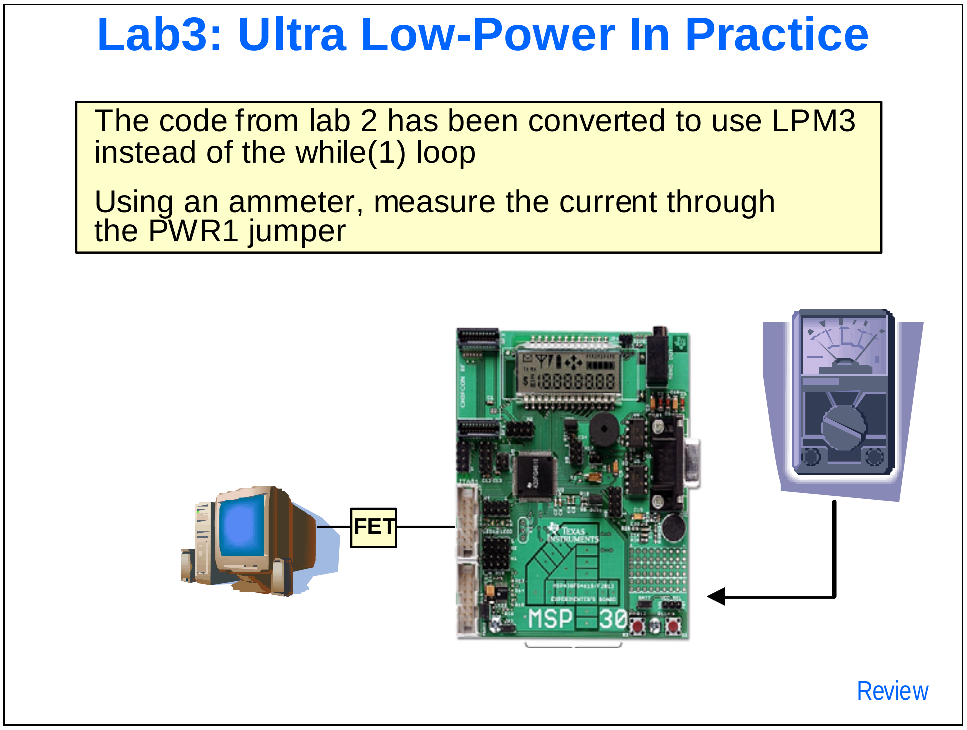
Principles For ULP Applications

- 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

Lab3 ...

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

Review

Hardware list:

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

Software list:

- IAR Kickstart for MSP430 version 4.21B
- 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.

Code used	LED Off (mA)	LED On (mA)
Lab2		
Lab3		

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:

Code used	LED Off (mA)	LED On (mA)
Lab2	0.6	2.7
Lab3	0.0	2.1

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.

Code used	LED Off (mA)	LED On (mA)
Lab2		
Lab3		

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:

Code used	LED Off (mA)	LED On (mA)
Lab2	**0.6**	**2.7**
Lab3	**0.0**	**2.1**

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 *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?

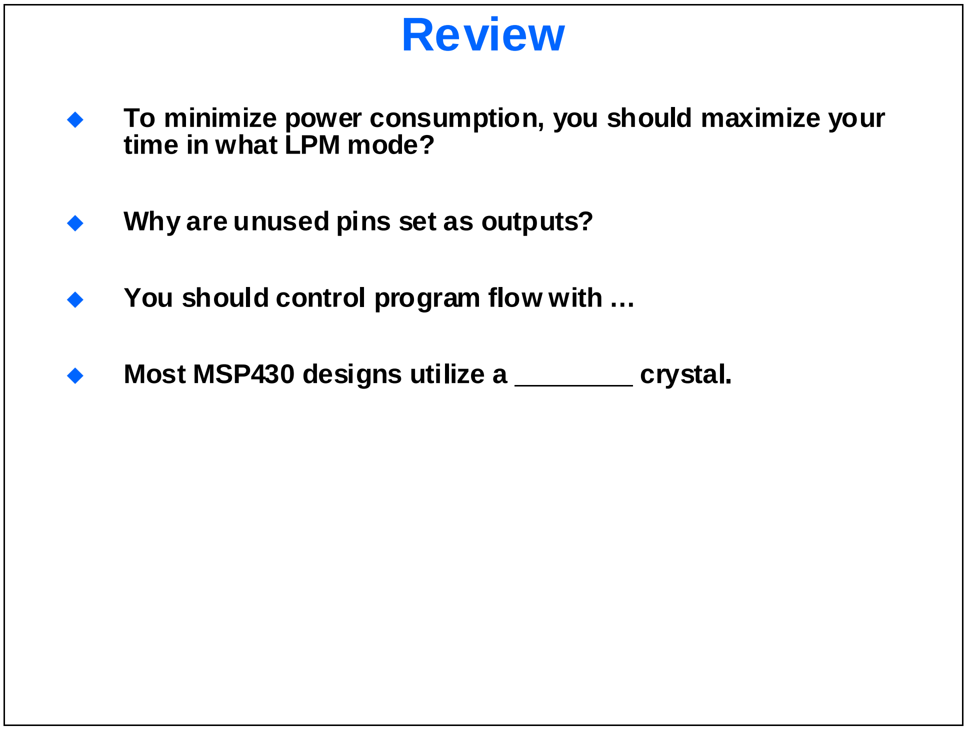
Look in the header file to see how LPM3_bits is defined

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

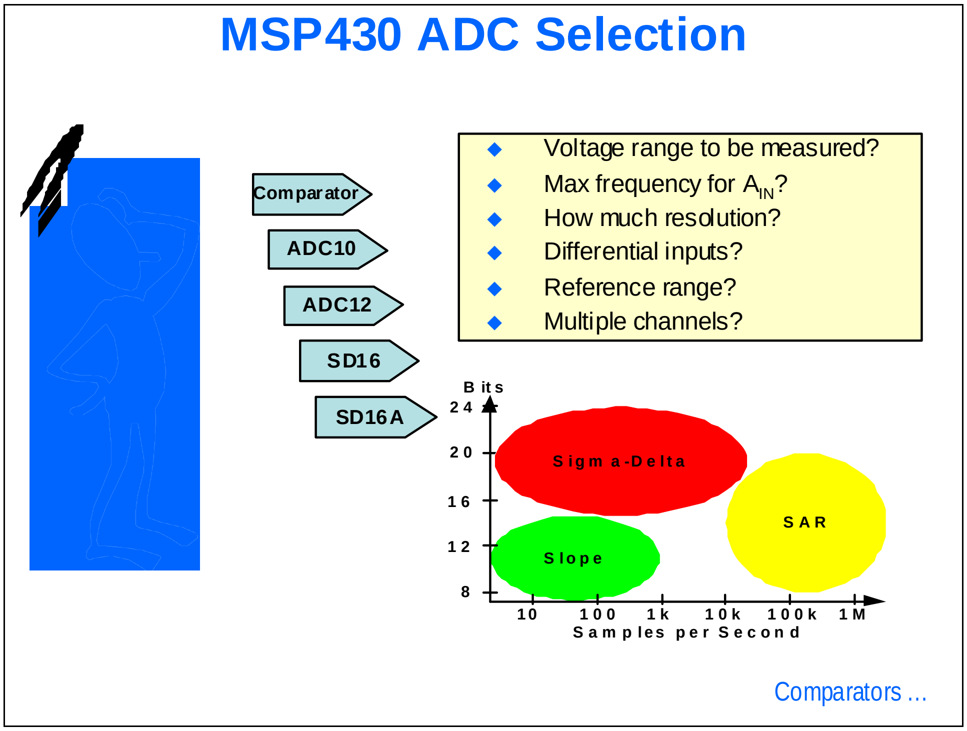
- Comparators
- ADC10 & 12
- SD16 & SD16A
- DAC12
- DTC
- Timer triggers
- Lab

*** This page intentionally left _____ ***

Module Topics

*** Blankety, blank ***

ADC Selection



MSP430 ADC Selection

Comparator

ADC10

ADC12

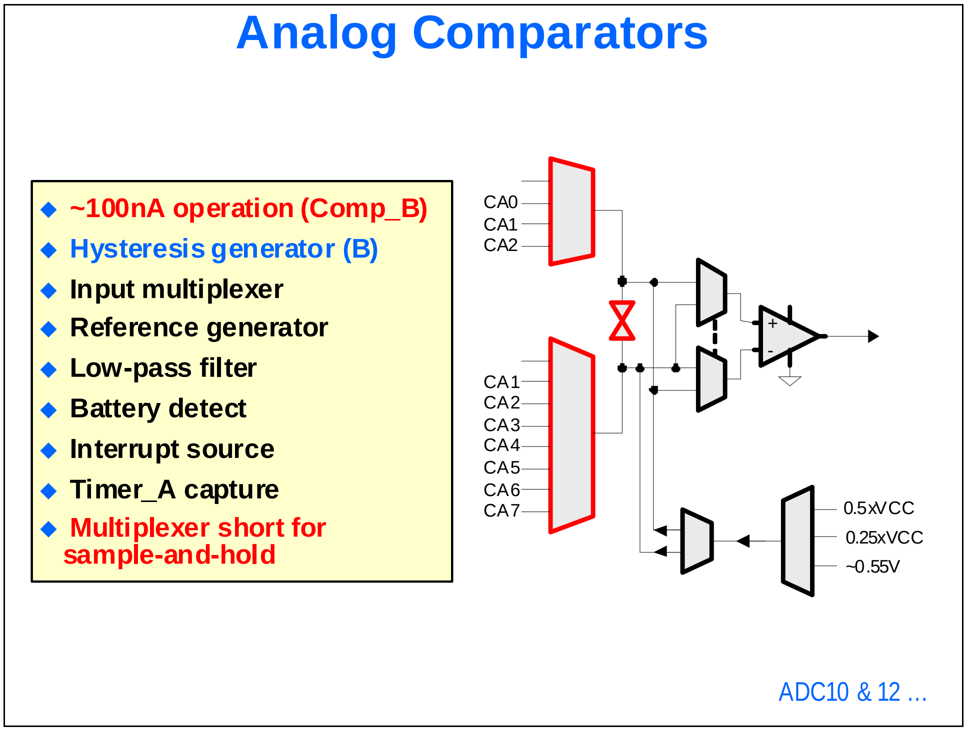
SD16

SD16A

- Voltage range to be measured?
- Max frequency for A_{IN}?
- How much resolution?
- Differential inputs?
- Reference range?
- Multiple channels?

Comparators ...

Comparators

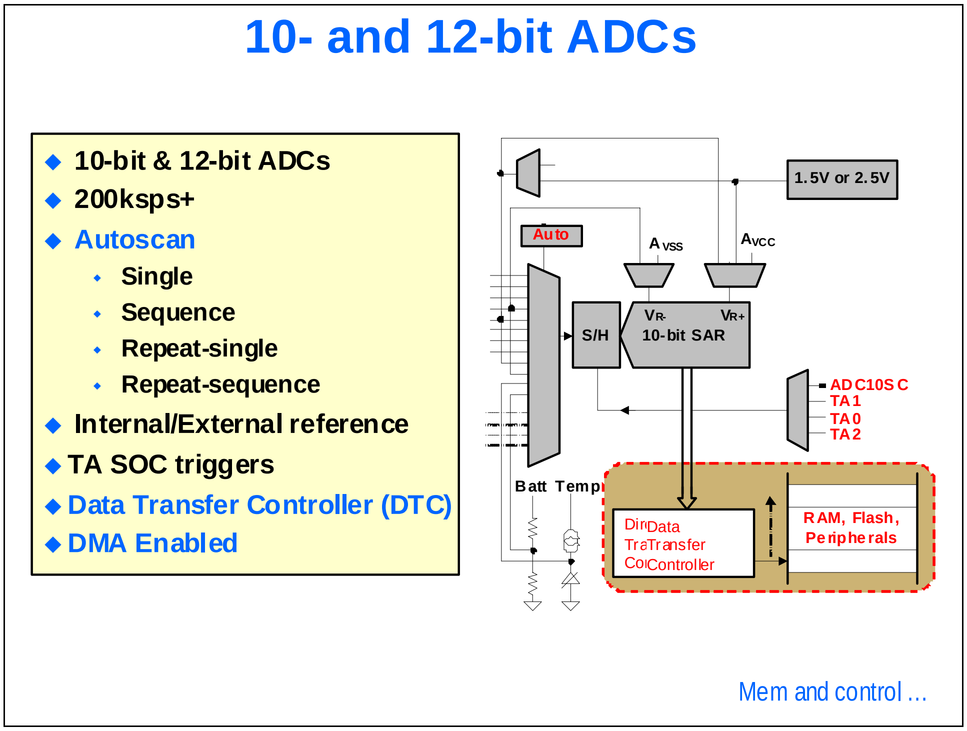


Analog Comparators

- ~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

ADC10 & 12 ...

ADC10 & ADC12

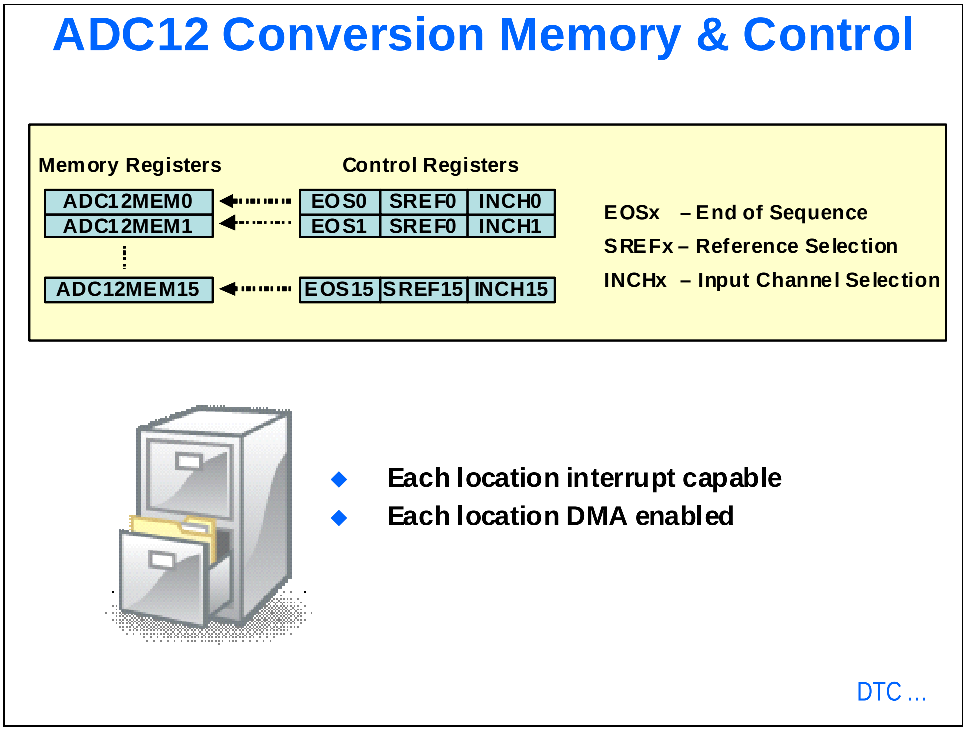


10- and 12-bit ADCs

- 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

Mem and control ...

Conversion Memory and Control



ADC12 Conversion Memory & Control

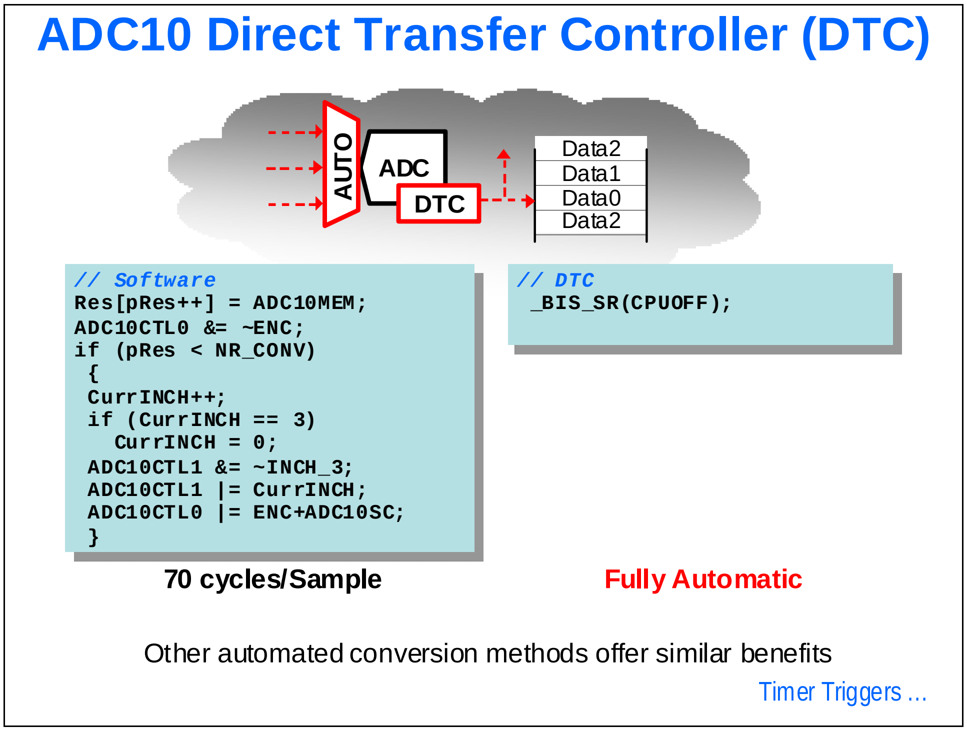
Memory Registers	Control Registers		
ADC12MEM0	EOS0	SREF0	INCH0
ADC12MEM1	EOS1	SREF0	INCH1
⋮			
ADC12MEM15	EOS15	SREF15	INCH15

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)

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

```
// DTC
 _BIS_SR(CPUOFF);
```

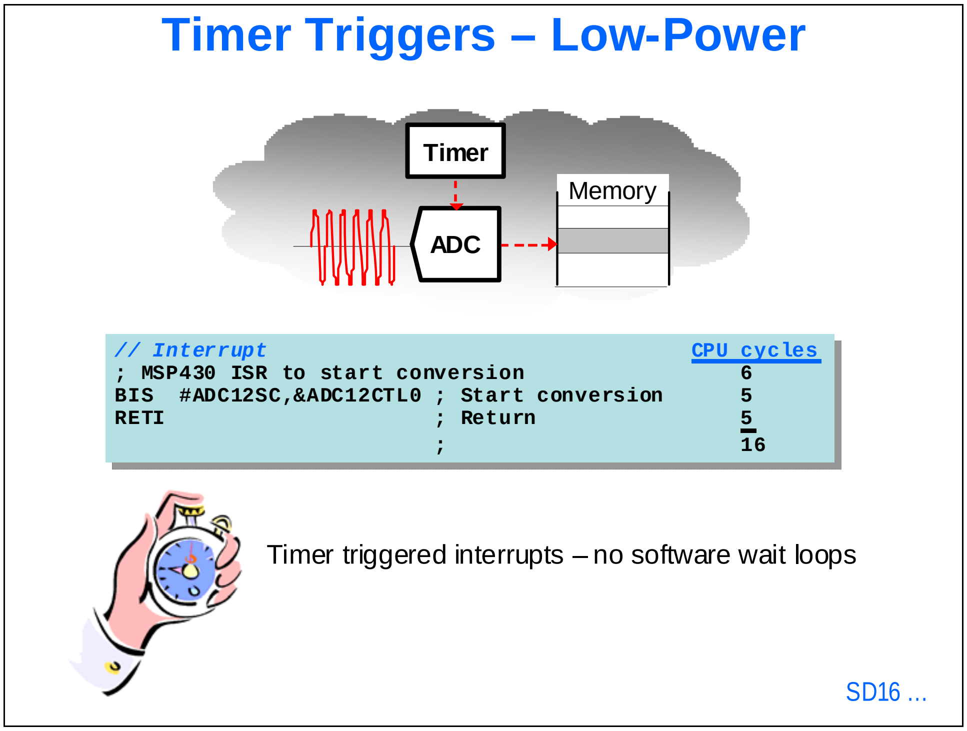
70 cycles/Sample

Fully Automatic

Other automated conversion methods offer similar benefits

Timer Triggers ...

Timer Triggers



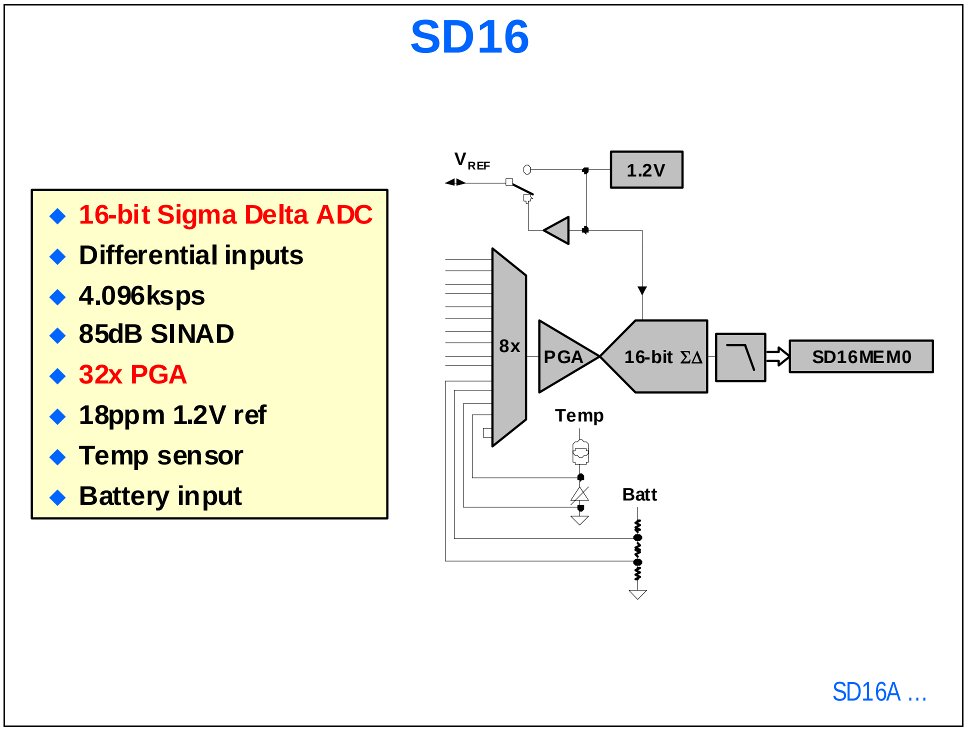
Timer Triggers – Low-Power

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

Timer triggered interrupts – no software wait loops

SD16 ...

SD16

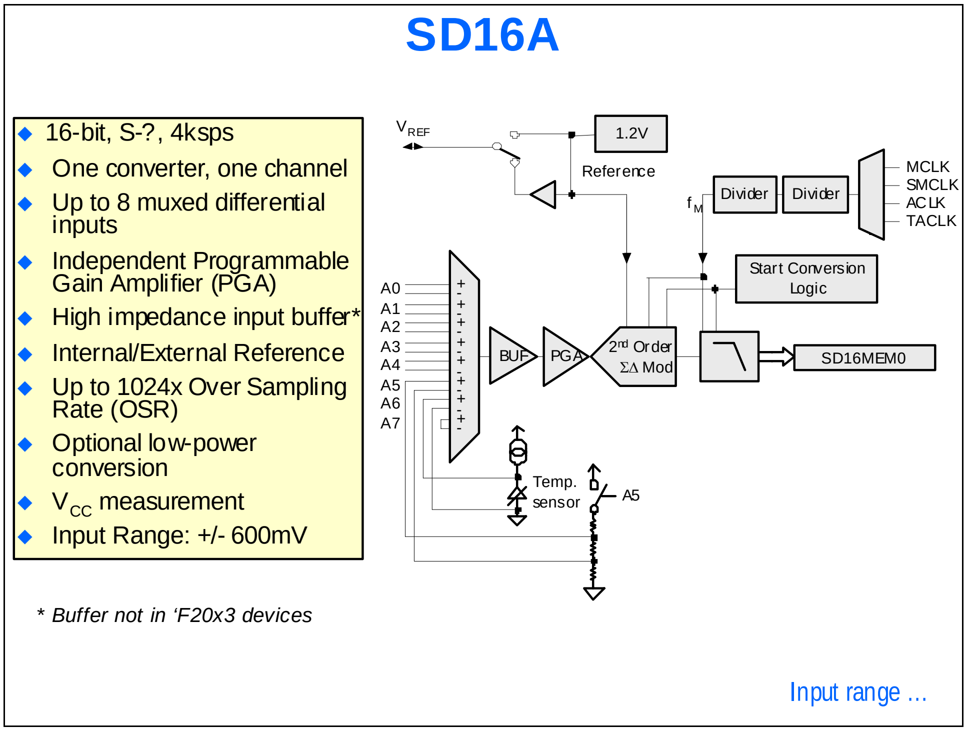


SD16

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

SD16A ...

SD16A



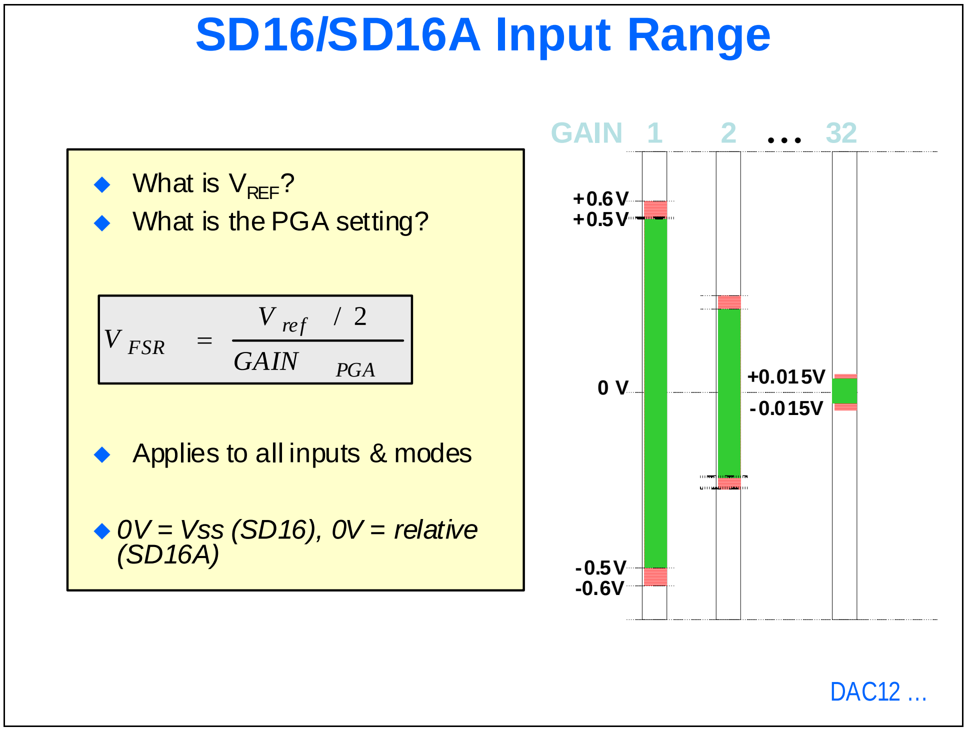
SD16A

- 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

Input range ...

SD16 & SD16A Input Range



SD16/SD16A Input Range

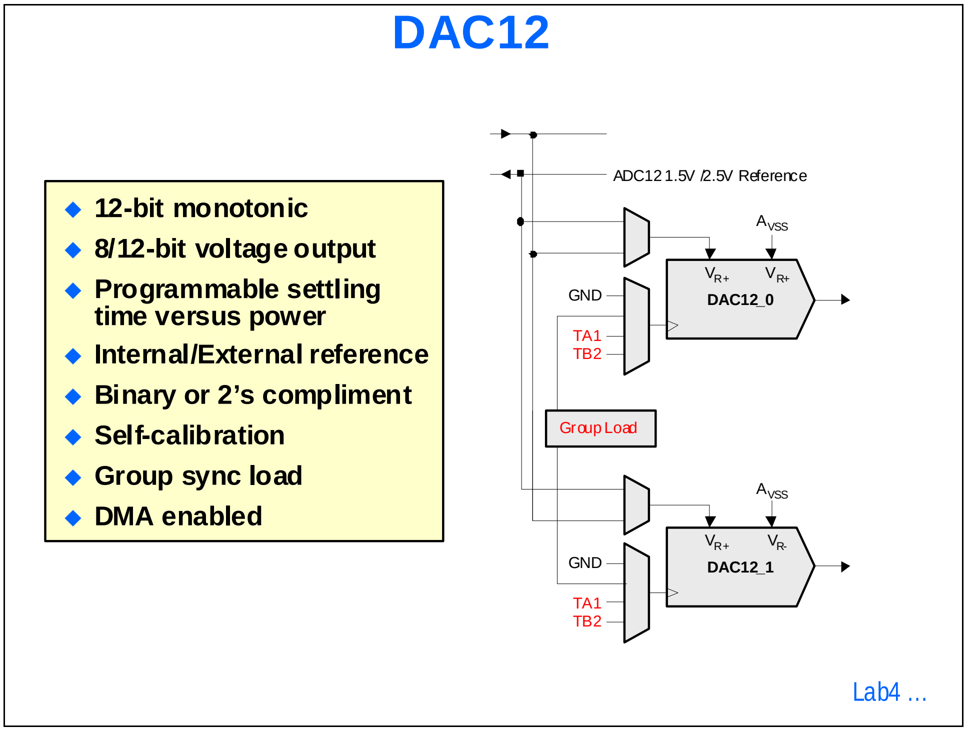
- What is V_{REF}?
- What is the PGA setting?

$$V_{FSR} = \frac{V_{ref} / 2}{GAIN_{PGA}}$$

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

DAC12 ...

DAC12



DAC12

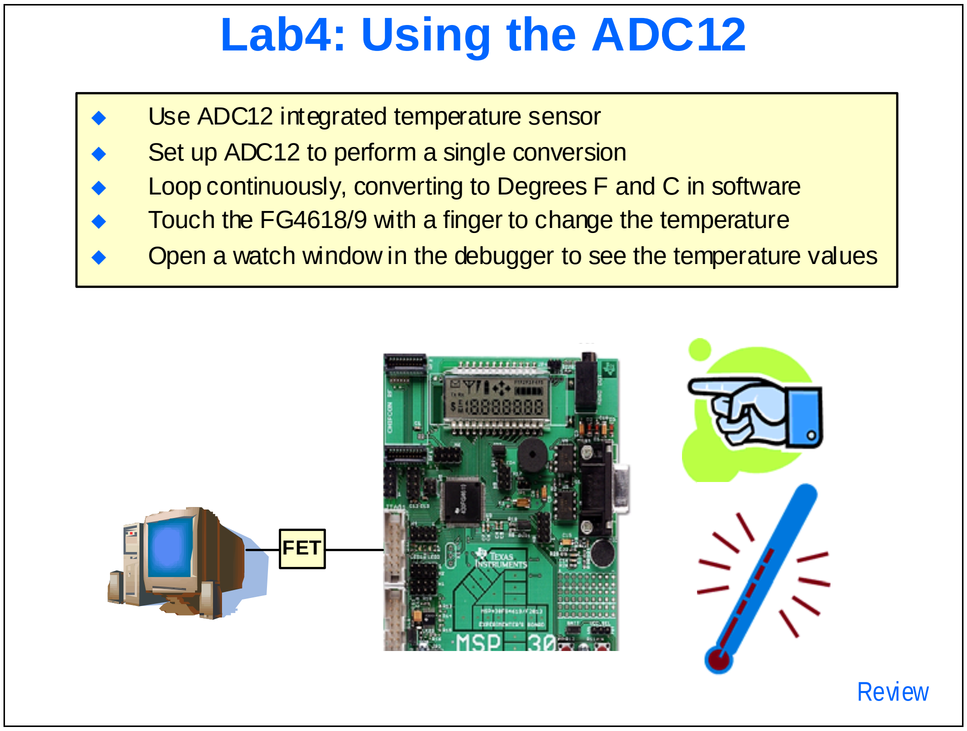
- **12-bit monotonic**
- **8/12-bit voltage output**
- **Programmable settling time versus power**
- **Internal/External reference**
- **Binary or 2's compliment**
- **Self-calibration**
- **Group sync load**
- **DMA enabled**

Lab4 ...

*** 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

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
  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);             // 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
}
```

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:

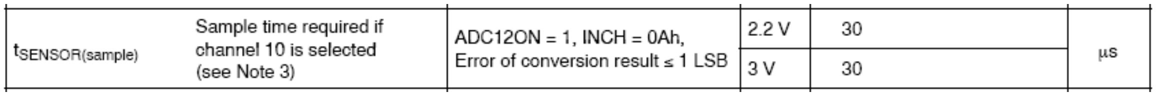
Turn ADC12 on: _________________

Turn ADC12 reference on: _____________________

Set the reference to 2.5V: _______________________

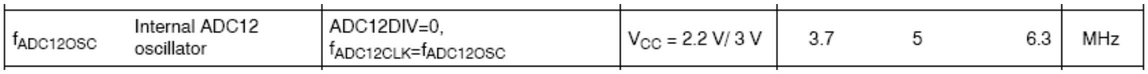
Set the sampling time: _______________________

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:



$t_{SENSOR(sample)}$	Sample time required if channel 10 is selected (see Note 3)	ADC12ON = 1, INCH = 0Ah, Error of conversion result ≤ 1 LSB	2.2 V	30	µs
			3 V	30	

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:



$f_{ADC12OSC}$	Internal ADC12 oscillator	ADC12DIV=0, $f_{ADC12CLK}$=$f_{ADC12OSC}$	V_{CC} = 2.2 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.

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!

```
#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. 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:

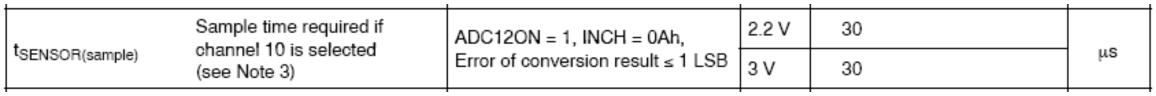
Turn ADC12 on: _________________

Turn ADC12 reference on: _____________________

Set the reference to 2.5V: _______________________

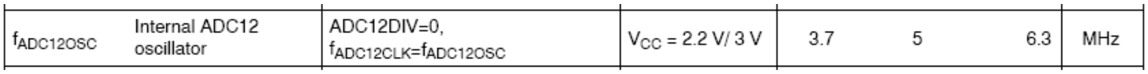
Set the sampling time: _______________________

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_{SENSOR}** and you'll see the following:



$t_{SENSOR(sample)}$	Sample time required if channel 10 is selected (see Note 3)	ADC12ON = 1, INCH = 0Ah, Error of conversion result ≤ 1 LSB	2.2 V	30	μs
			3 V	30	

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



$f_{ADC12OSC}$	Internal ADC12 oscillator	ADC12DIV=0, $f_{ADC12CLK}=f_{ADC12OSC}$	V_{CC} = 2.2 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. 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. 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_{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.

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!

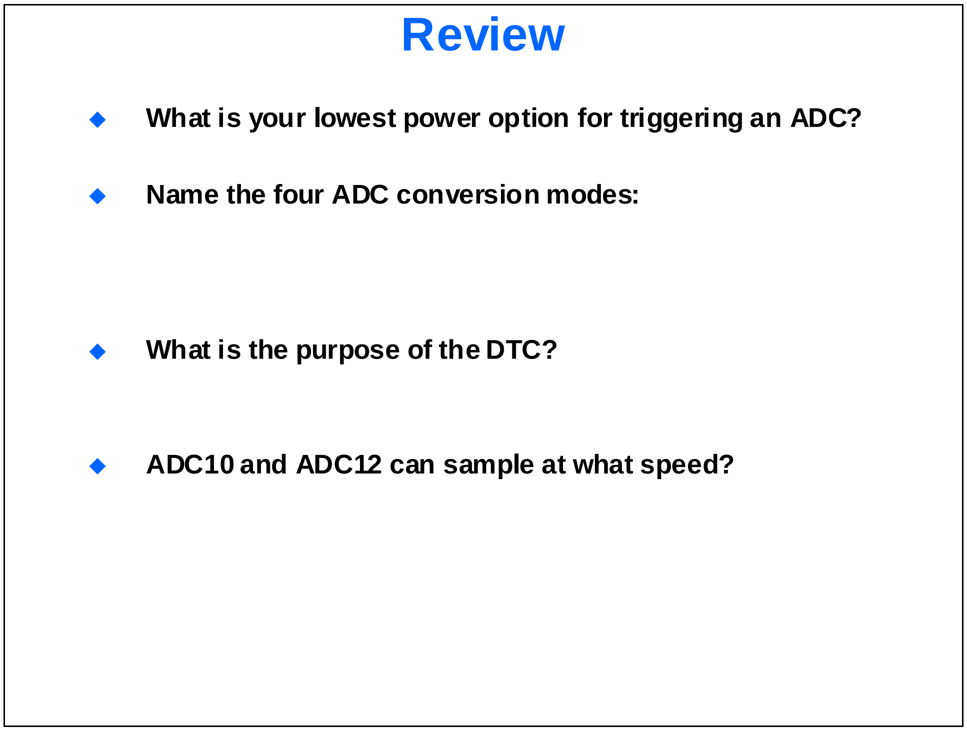
Shut Down

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

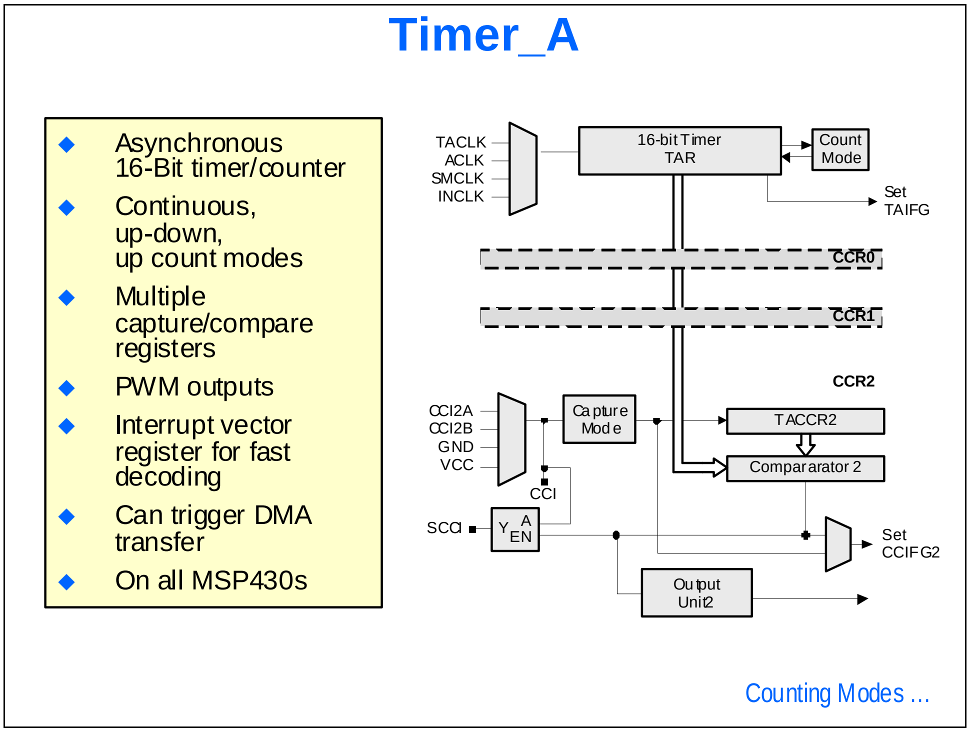
- Timer_A Architecture
- Count modes
- Interrupts
- TAIV
- Timer_B differences
- Timer lab

*** This page left blank with malice aforethought ***

Module Topics

*** Blank! Blank! My kingdom for a blank! ***

Timer_A

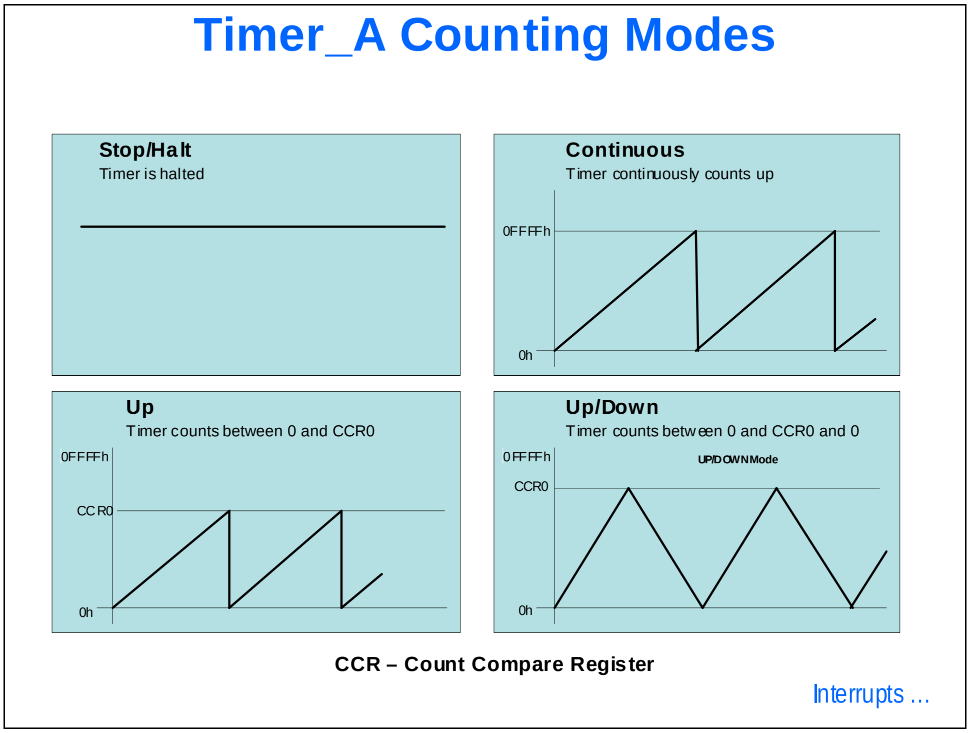


Timer_A

- 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 ...

Counting Modes



Timer_A Counting Modes

Stop/Halt
Timer is halted

Continuous
Timer continuously counts up

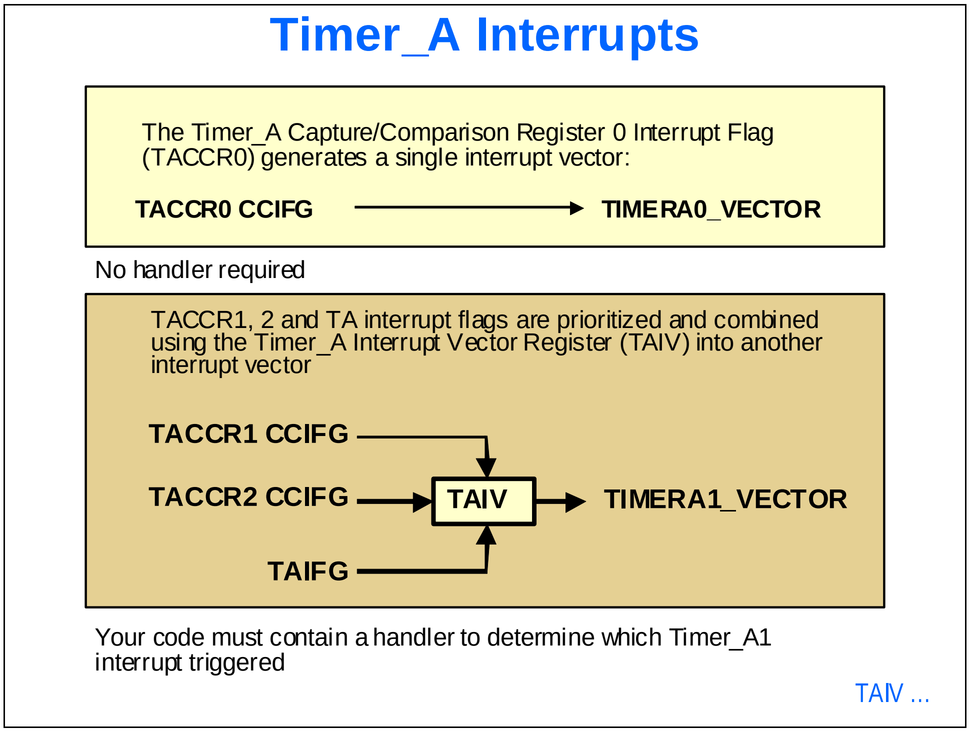
Up
Timer counts between 0 and CCR0

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:

TACCR0 CCIFG → **TIMERA0_VECTOR**

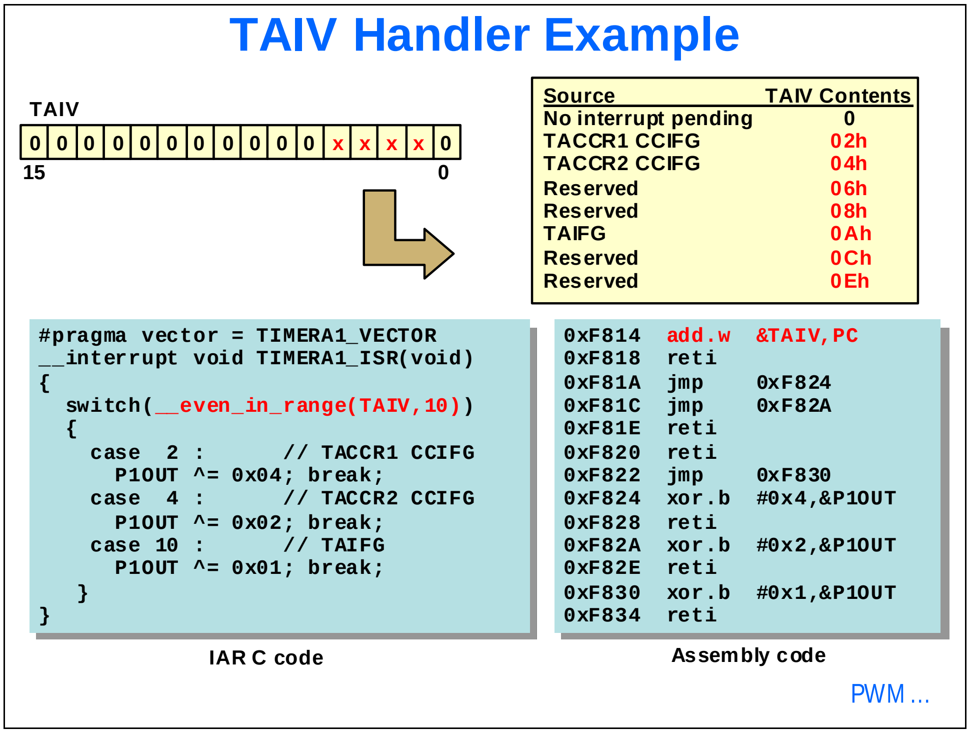
No handler required

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

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

TAIV ...

TAIV Handler



TAIV Handler Example

TAIV

0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	0

15 … 0

Source	TAIV Contents
No interrupt pending	0
TACCR1 CCIFG	02h
TACCR2 CCIFG	04h
Reserved	06h
Reserved	08h
TAIFG	0Ah
Reserved	0Ch
Reserved	0Eh

```
#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;
   }
}
```

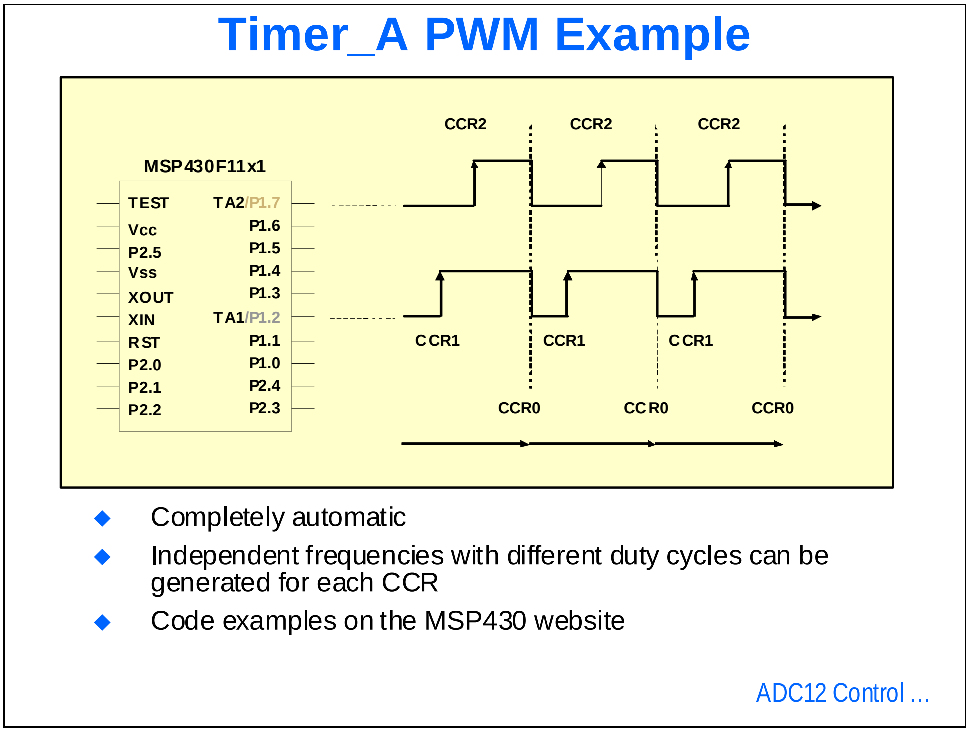
IAR C code

```
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
```

Assembly code

PWM ...

PWM Example

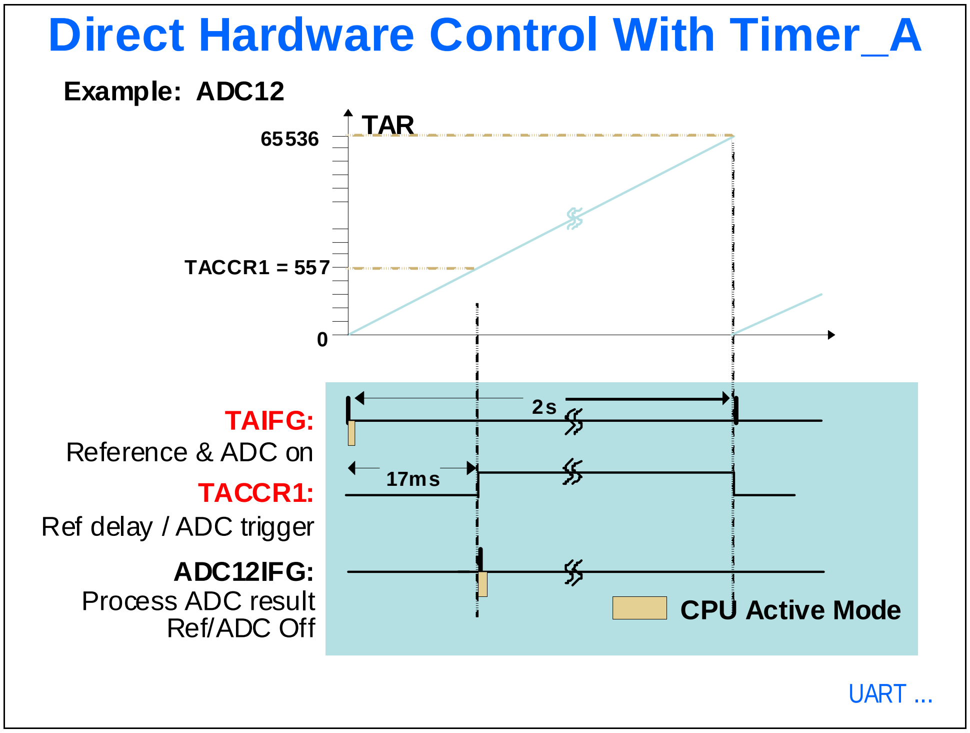


Timer_A PWM Example

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

ADC12 Control ...

Direct Hardware Control

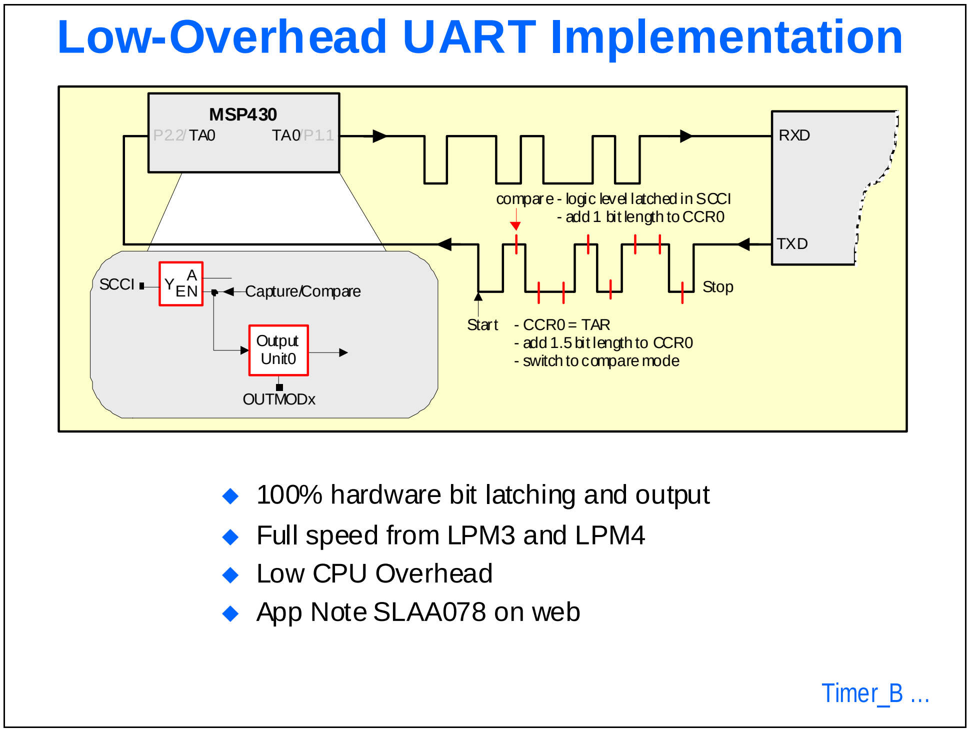


Direct Hardware Control With Timer_A

Example: ADC12

UART ...

UART Implementation

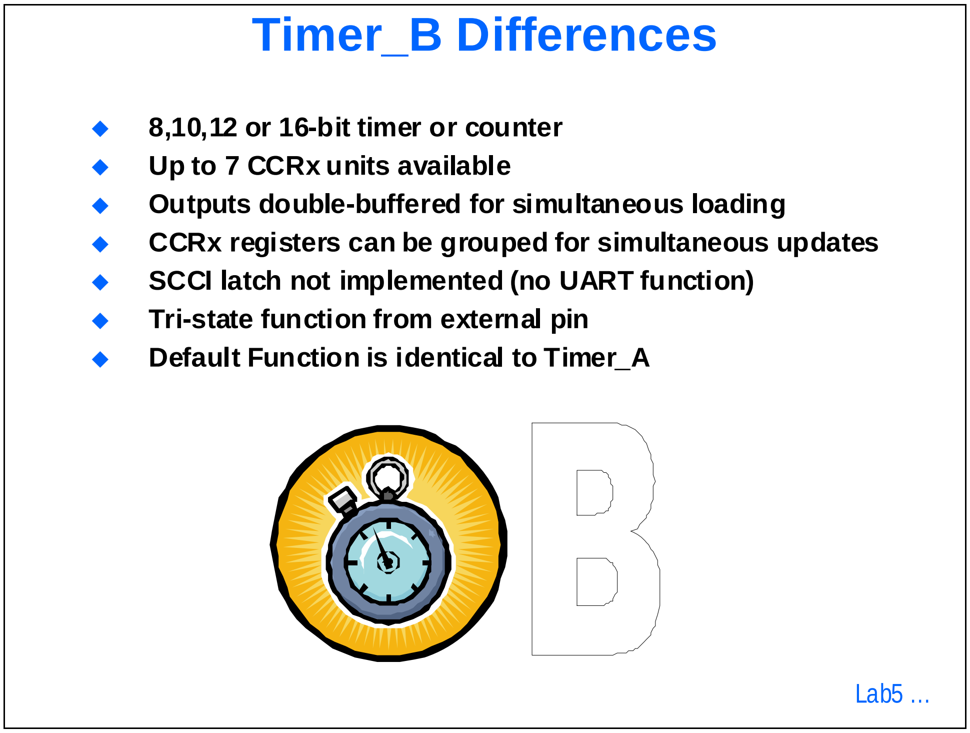


Low-Overhead UART Implementation

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

Timer_B ...

Timer B



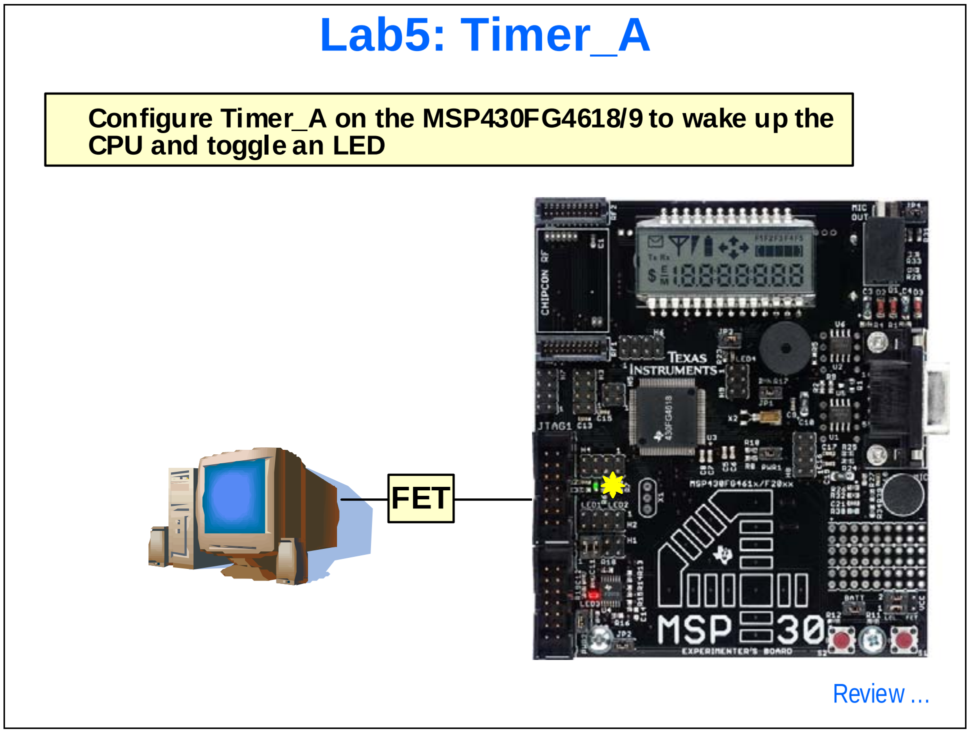
Timer_B Differences

- **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**

Lab5 ...

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.



Lab5: Timer_A

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

Review ...

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.

```
#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** *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.

```
#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?

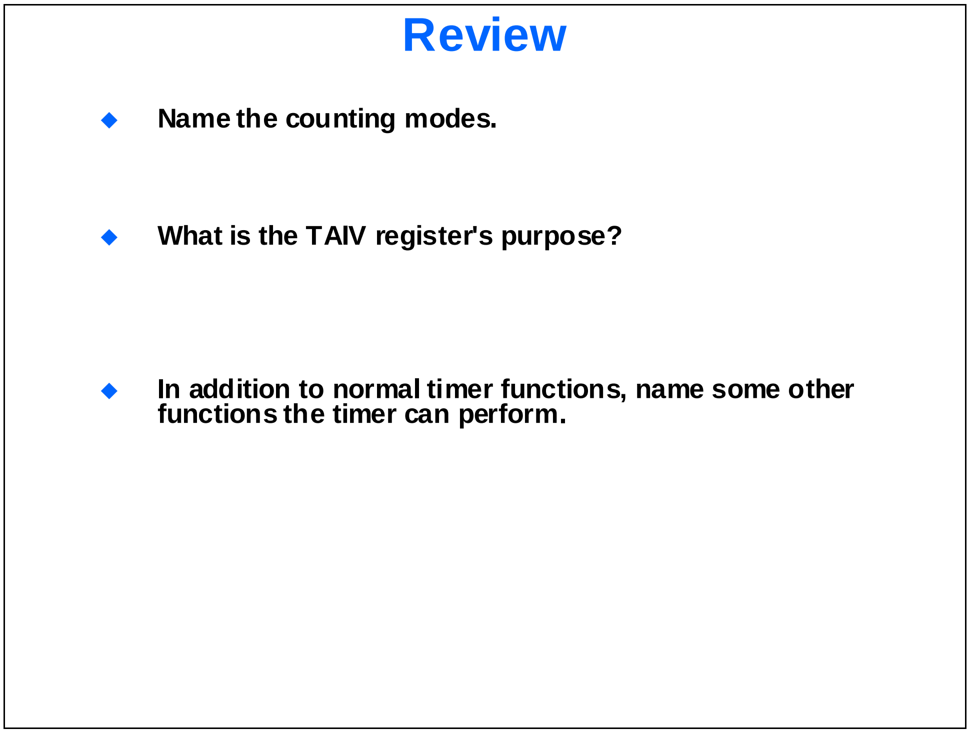
Shut Down

10. Shut down

Halt the debugger and **shut down** *Code Composer Studio*.

CCS Users … you’re done.

Review Questions



Review

- 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.

*** !KCOR s034PSM ***

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

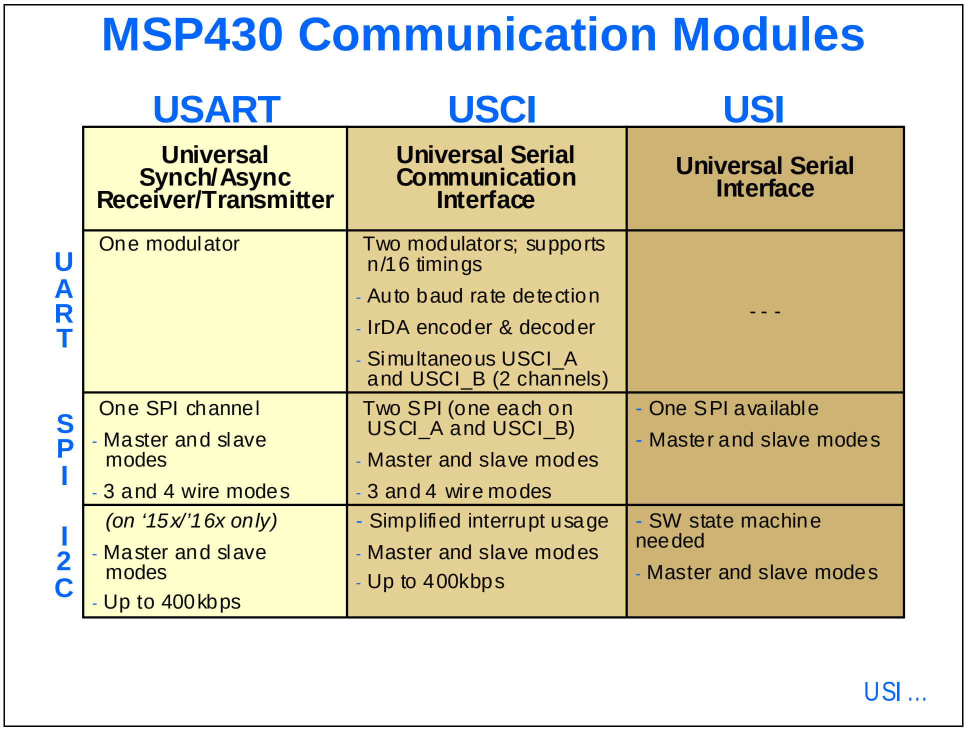
- USART
- USCI
- USI

*** I only insert blank pages when the voices tell me to***

Module Topics

*** For security reasons this page must be left blank ***

MSP430 Communication Modules

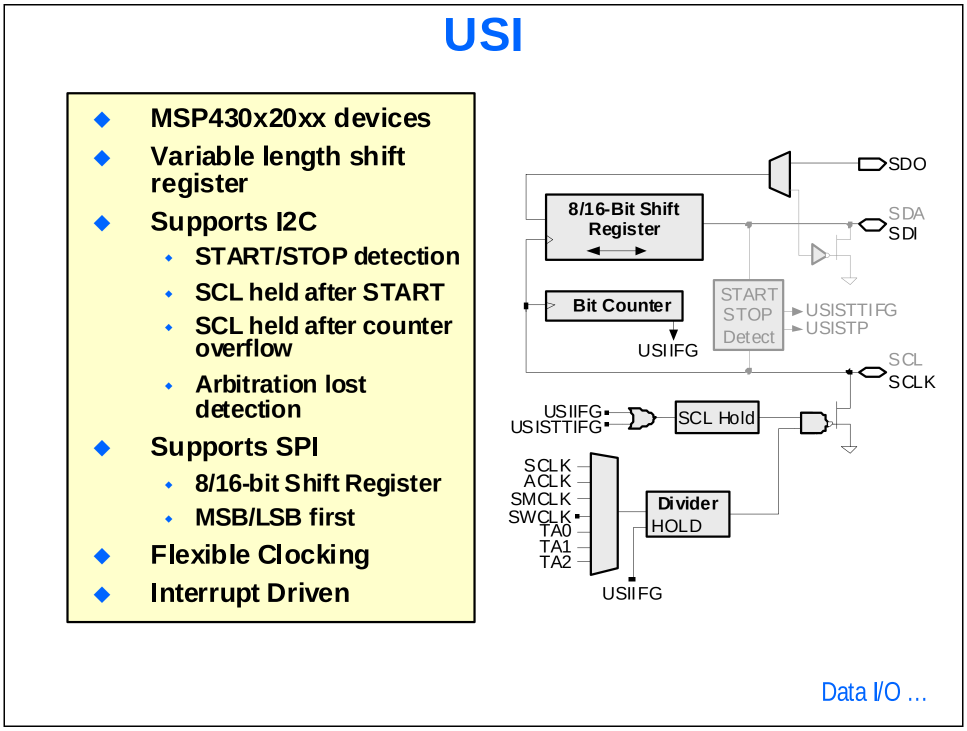


MSP430 Communication Modules

	USART	USCI	USI
	Universal Synch/Async Receiver/Transmitter	Universal Serial Communication Interface	Universal Serial Interface
UART	One modulator	Two modulators; supports n/16 timings - Auto baud rate detection - IrDA encoder & decoder - Simultaneous USCI_A and USCI_B (2 channels)	- - -
SPI	One SPI channel - Master and slave modes - 3 and 4 wire modes	Two SPI (one each on USCI_A and USCI_B) - Master and slave modes - 3 and 4 wire modes	- One SPI available - Master and slave modes
I2C	*(on '15x/'16x only)* - Master and slave modes - Up to 400kbps	- Simplified interrupt usage - Master and slave modes - Up to 400kbps	- SW state machine needed - Master and slave modes

USI ...

USI

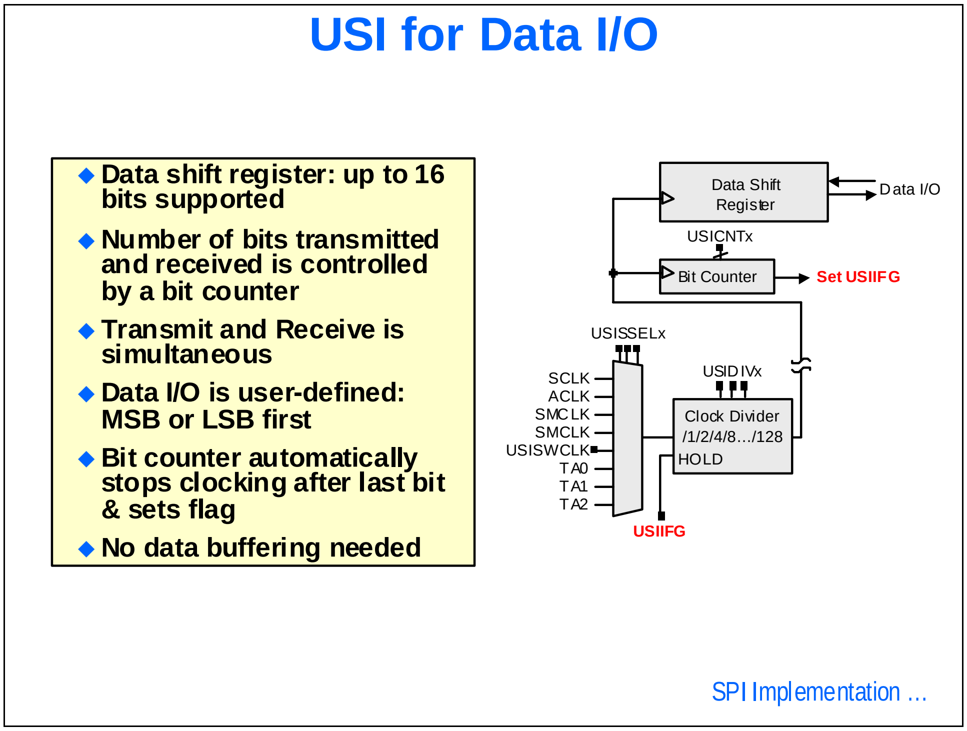


USI

- 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

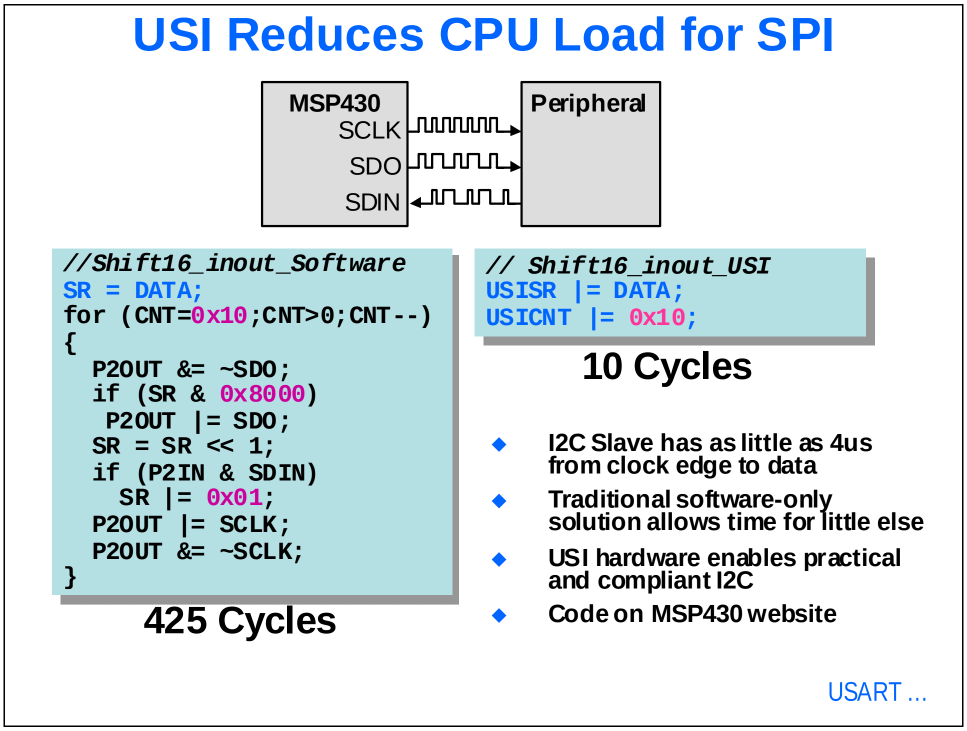


USI for Data I/O

- 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

MSP430
SCLK
SDO
SDIN
Peripheral

```
//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;
}
```

425 Cycles

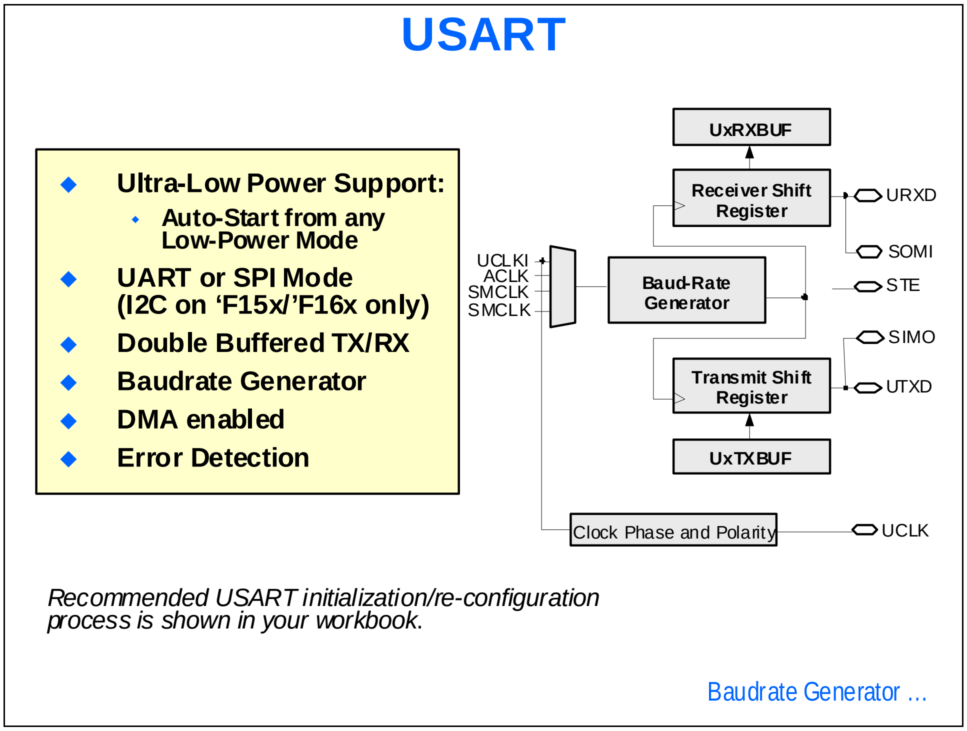
```
// Shift16_inout_USI
USISR  |= DATA;
USICNT  |= 0x10;
```

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

USART …

USART



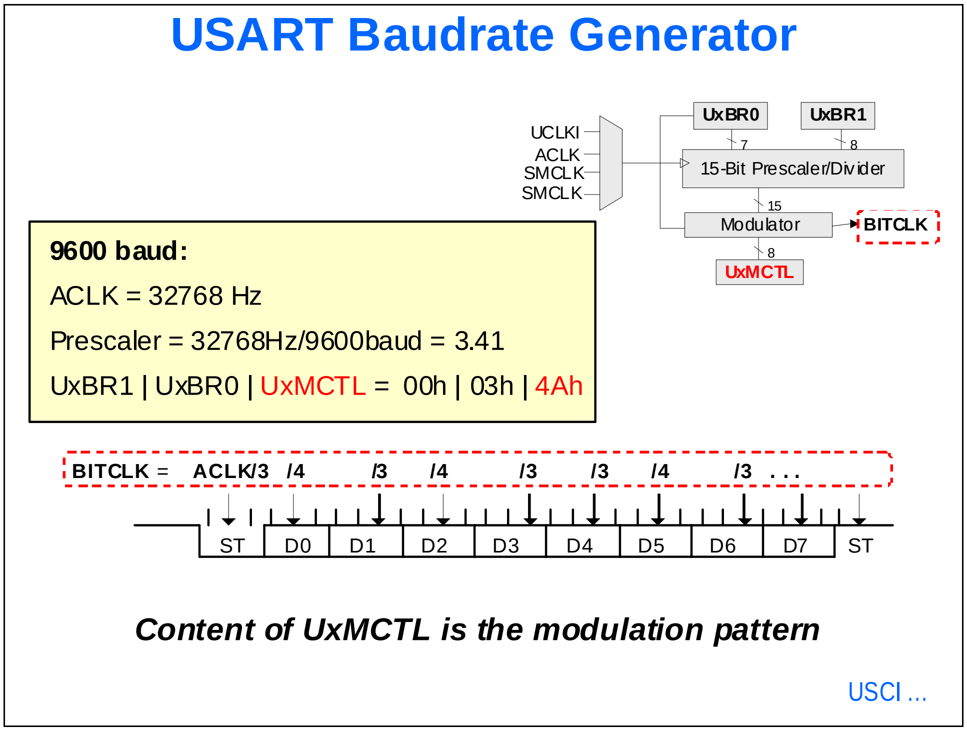
USART

- 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 ...

Baudrate Generator



USART Baudrate Generator

9600 baud:

ACLK = 32768 Hz

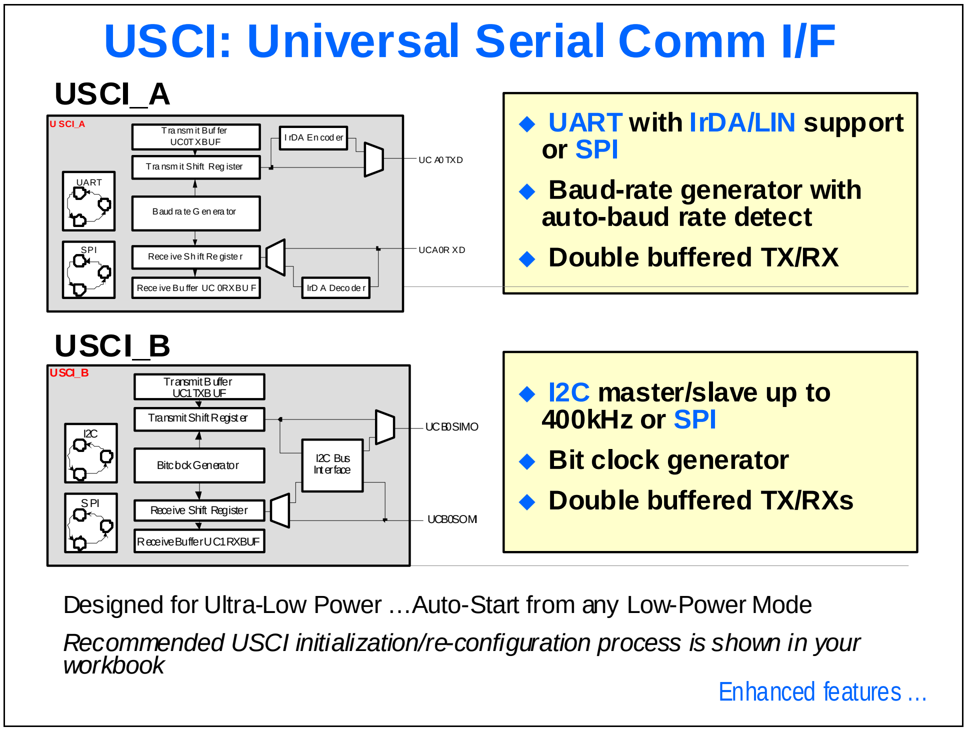
Prescaler = 32768Hz/9600baud = 3.41

UxBR1 | UxBR0 | UxMCTL = 00h | 03h | 4Ah

Content of UxMCTL is the modulation pattern

USCI ...

USCI



USCI: Universal Serial Comm I/F

USCI_A

- UART with IrDA/LIN support or SPI
- Baud-rate generator with auto-baud rate detect
- Double buffered TX/RX

USCI_B

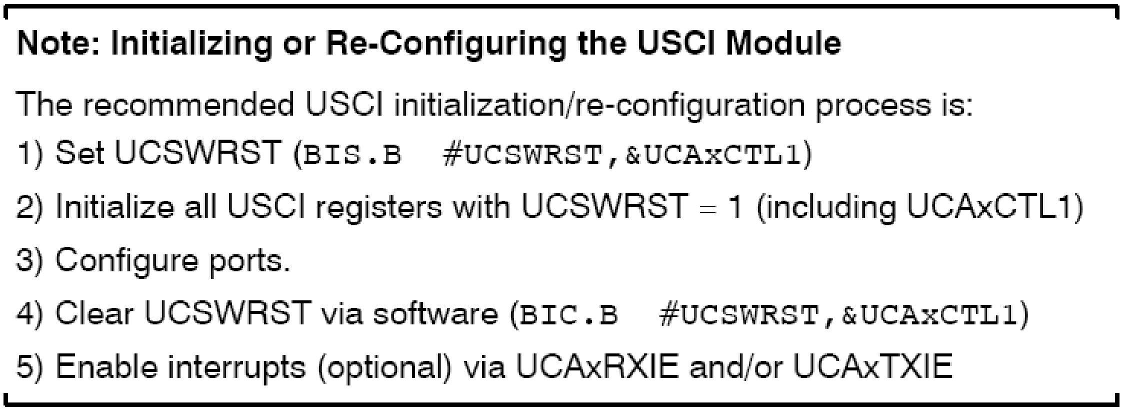
- I2C master/slave up to 400kHz or SPI
- Bit clock generator
- Double buffered TX/RXs

Designed for Ultra-Low Power …Auto-Start from any Low-Power Mode

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

Enhanced features …

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`)

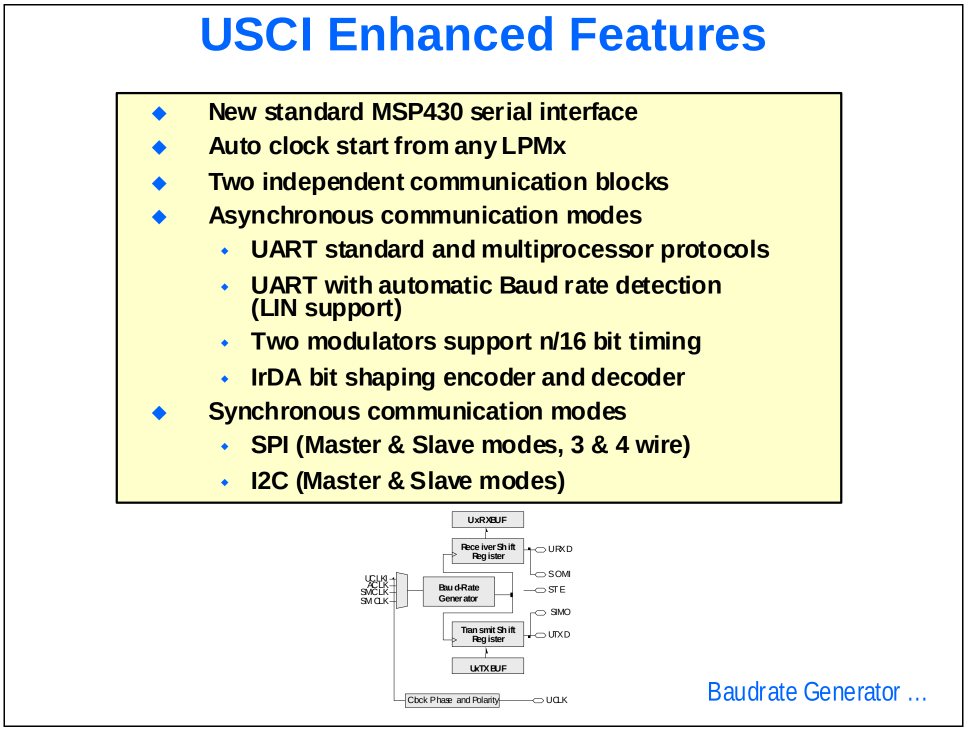
2) Initialize all USCI registers with UCSWRST = 1 (including UCAxCTL1)

3) Configure ports.

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

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

USCI Enhanced Features

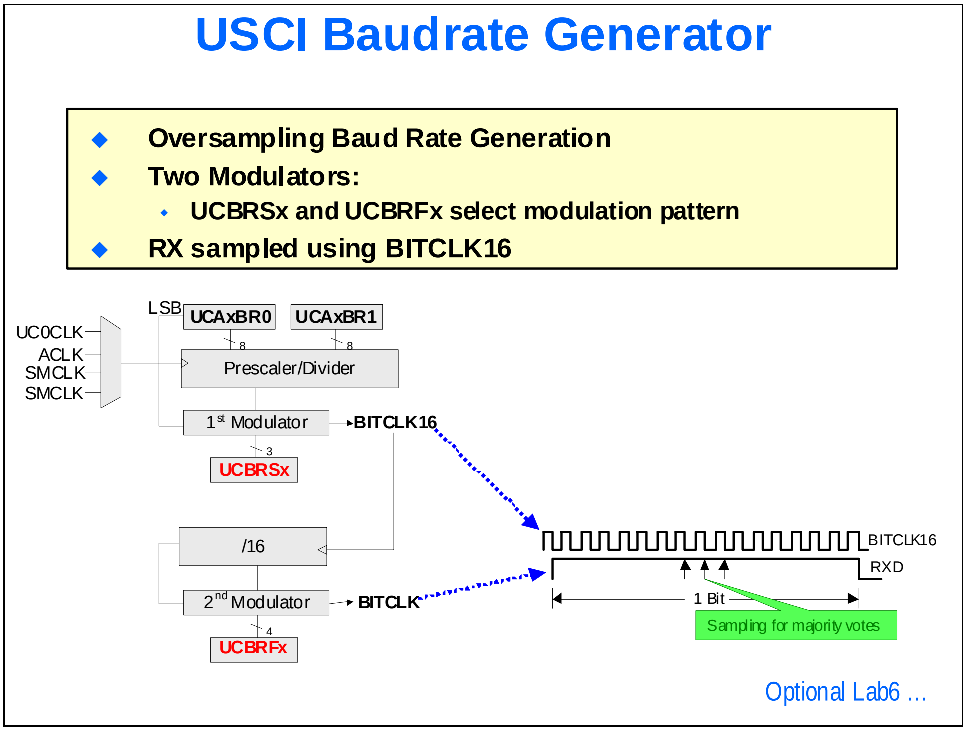


USCI Enhanced Features

- 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)

Baudrate Generator …

USCI Baudrate Generator



USCI Baudrate Generator

- Oversampling Baud Rate Generation
- Two Modulators:
  - UCBRSx and UCBRFx select modulation pattern
- RX sampled using BITCLK16

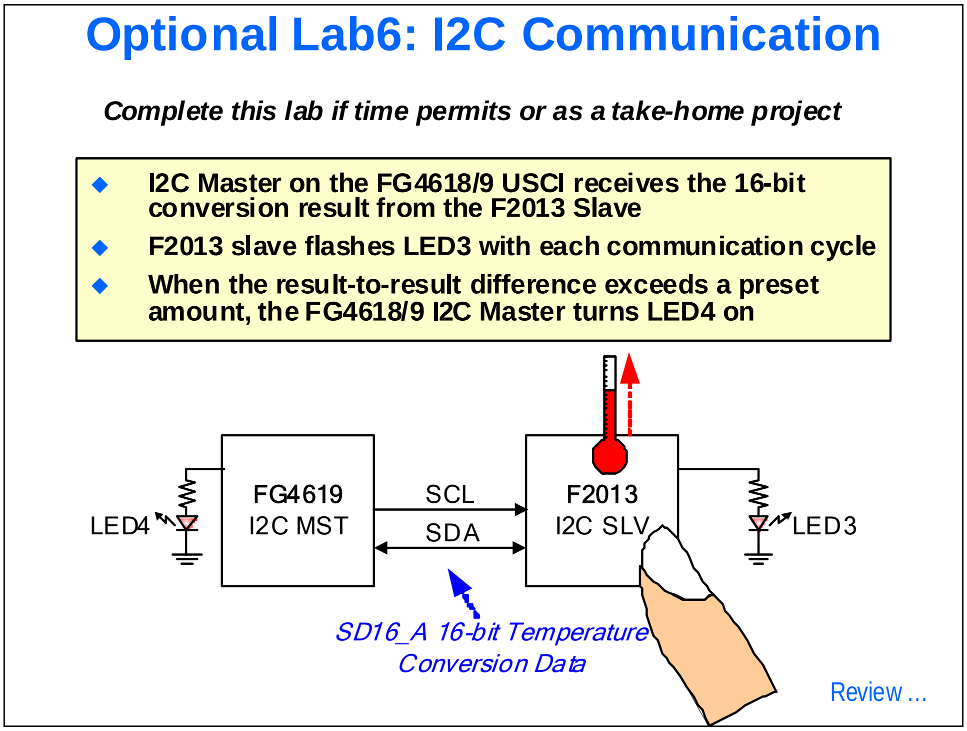
Optional Lab6 …

*** 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 …

```
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 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.

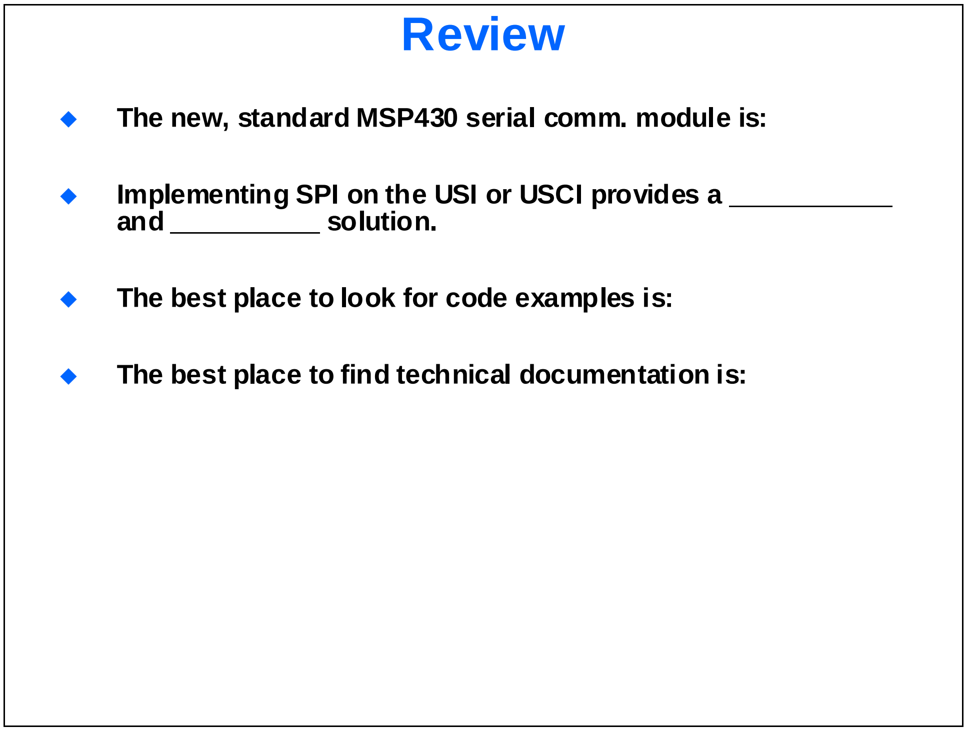
Shut Down

10. Shut down

Shut down *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 ***

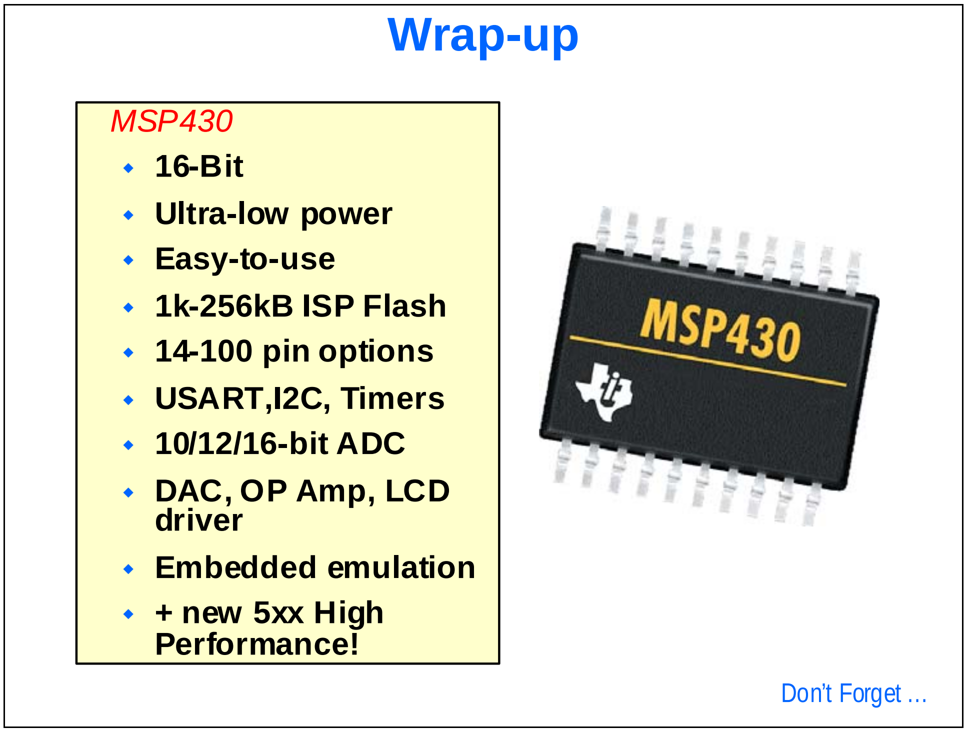
Wrap-Up

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



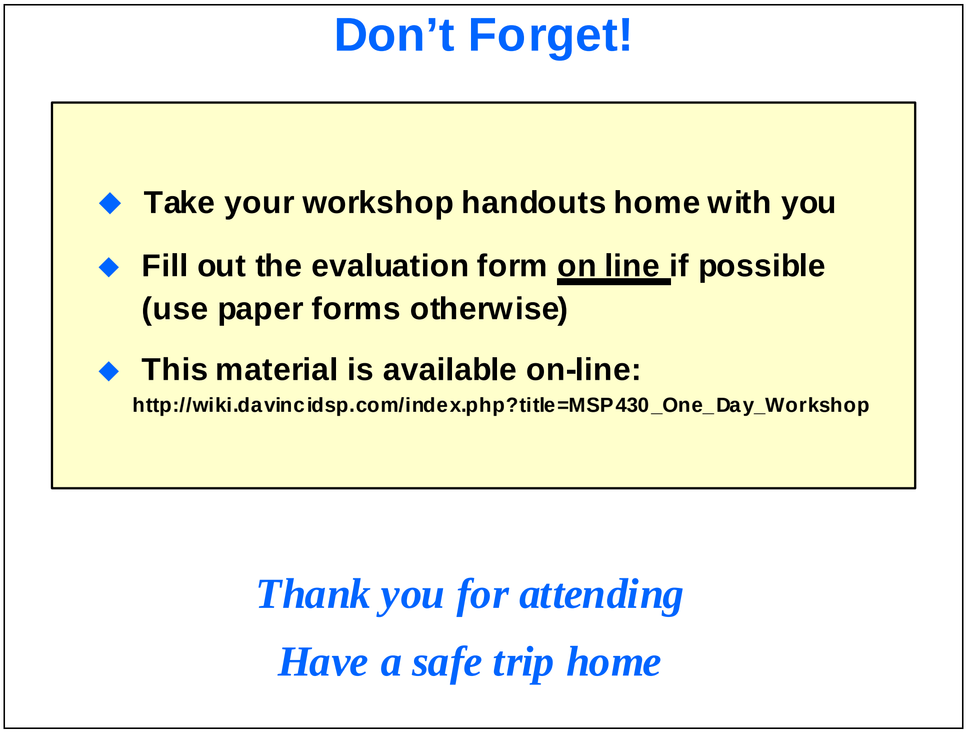
Wrap-up

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 ...

Don't Forget



Don't Forget!

- Take your workshop handouts home with you
- Fill out the evaluation form <u>on line</u> if possible (use paper forms otherwise)
- This material is available on-line:
  http://wiki.davincidsp.com/index.php?title=MSP430_One_Day_Workshop

Thank you for attending

Have a safe trip home

Texas Instruments

Addendum

Introduction

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

Objectives

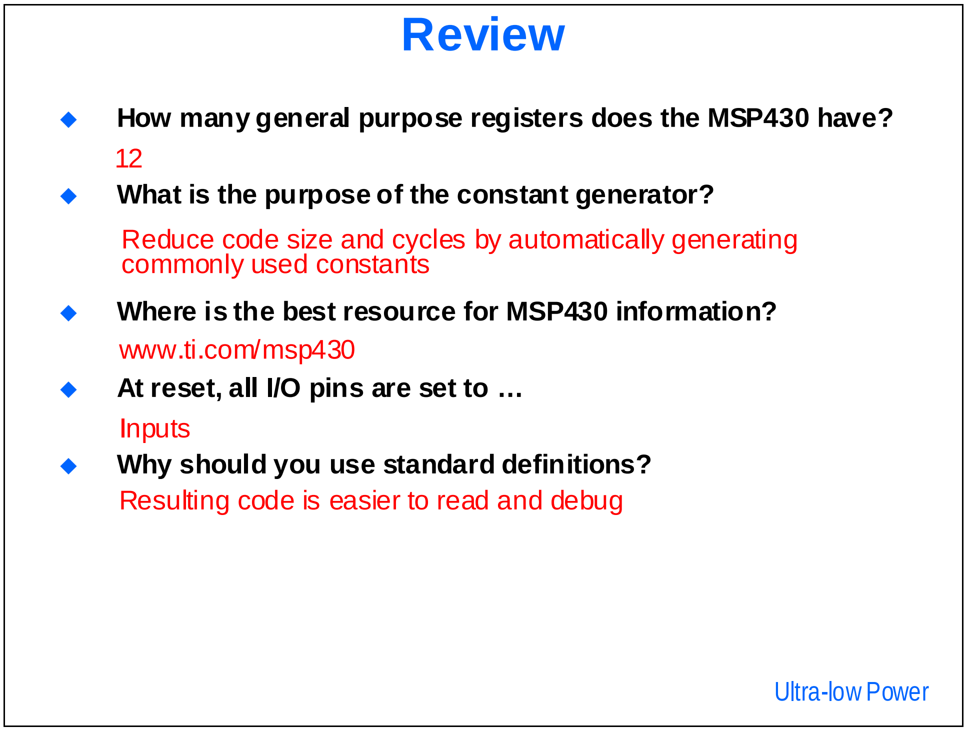
- Module review answers
- Lab Question Answers
- Lab Solutions

*** I was somewhat ambivalent about leaving this page blank. ***

Lab Solutions and Review Answers

*** 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 …

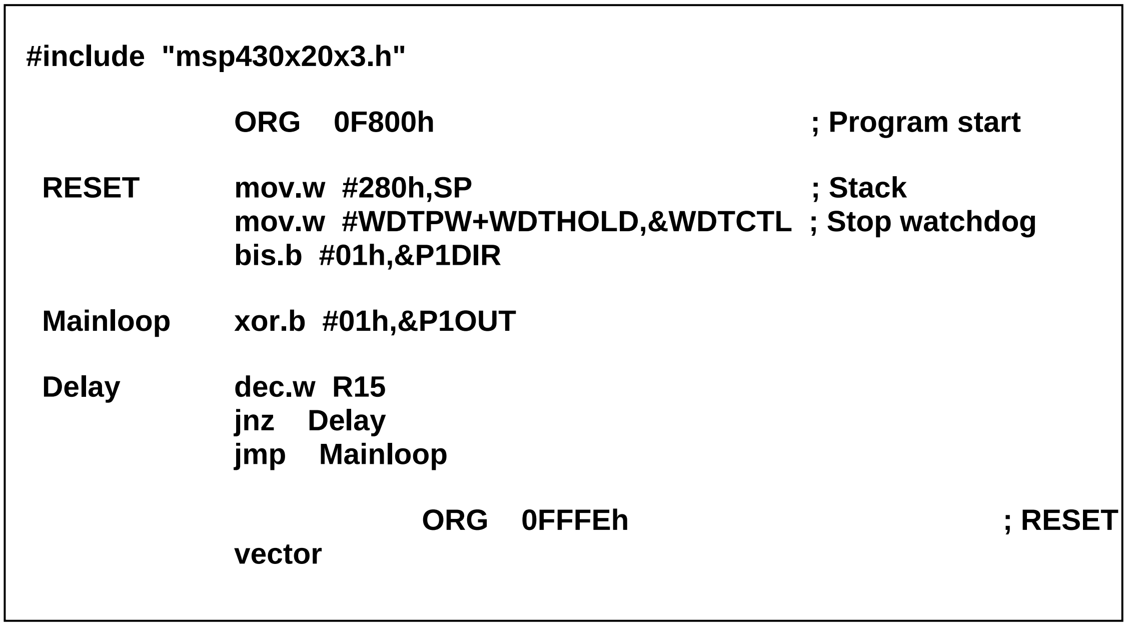
  Inputs

- Why should you use standard definitions?

  Resulting code is easier to read and debug

Ultra-low Power

Lab1 IAR Code



```
#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
```

Lab1 CCS Code

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

            .text                               ; Progam 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 kHz

tWord = 30

Time to program a word or byte = 30/476000 = 63uS

Time to randomly program 1024 words = 1024 * 63uS = 64.5mS

Lab2 IAR Solution

```
#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.

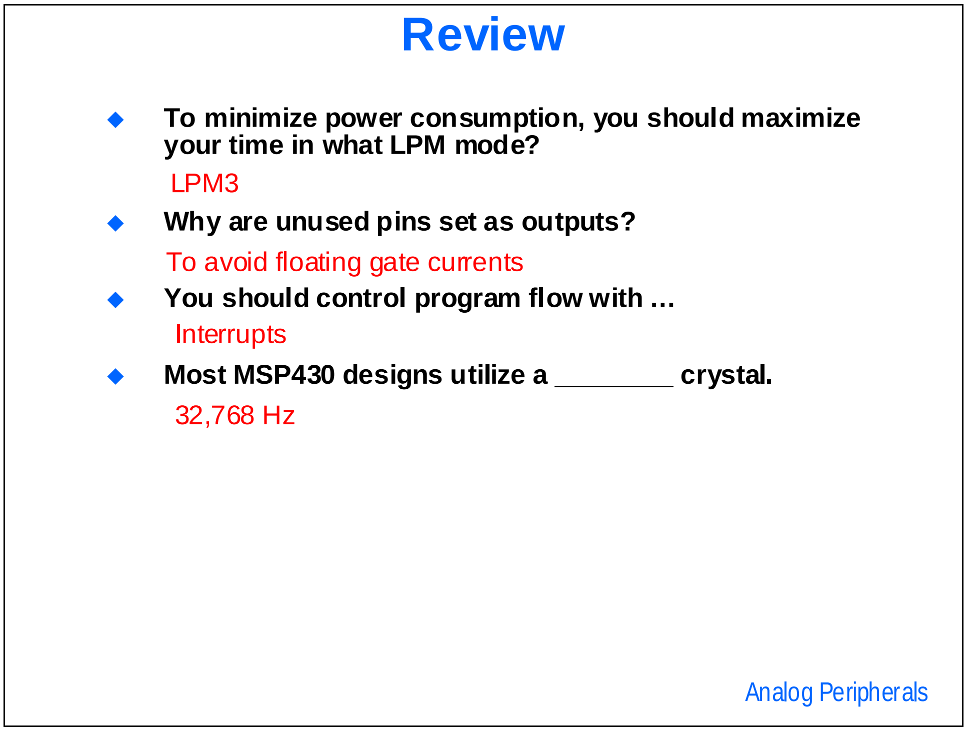
Look in the header file to see how LPM3_bits is defined

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.**

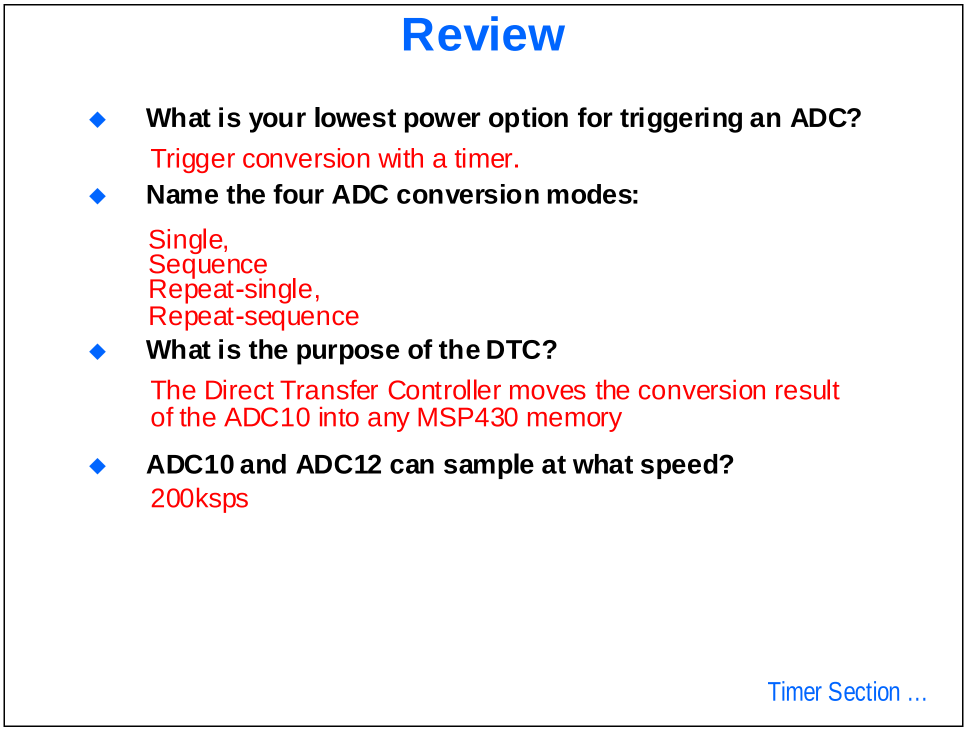
  32,768 Hz

Analog Peripherals

Lab4 IAR Solution

```
#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
}
```

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

Timer Section ...

Lab5 IAR Solution

```
#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=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
    P2OUT ^= 0x02;                   // Toggle P2.1 using exclusive-OR
}
```

Lab5 Step 9 Answers

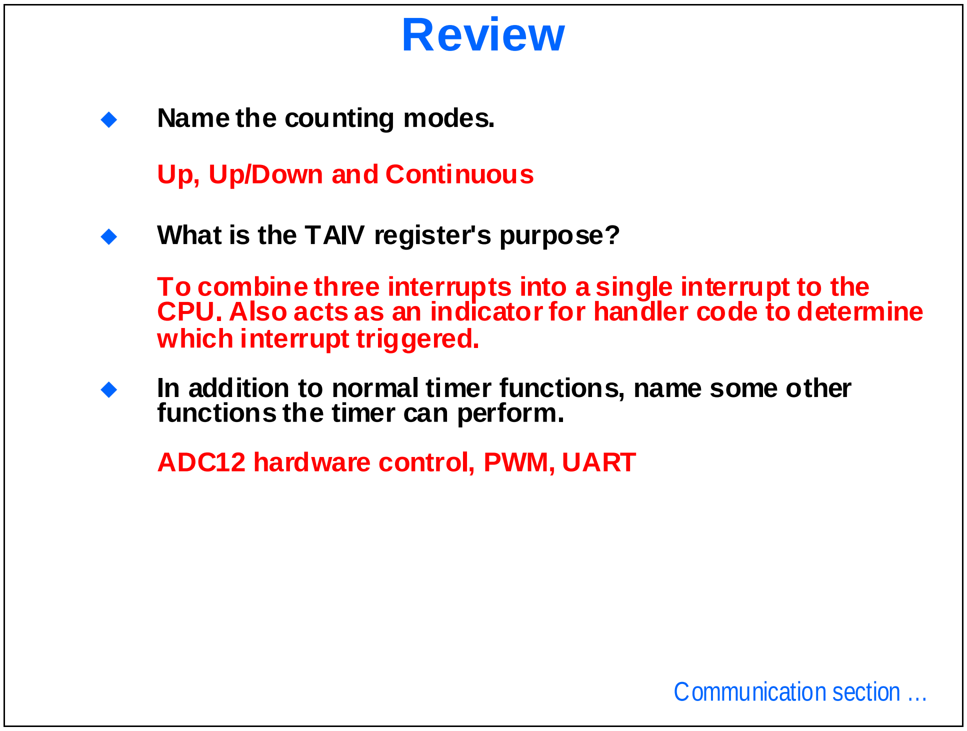
Why was TAIE not set in TACTL?

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 TAIV register's purpose?

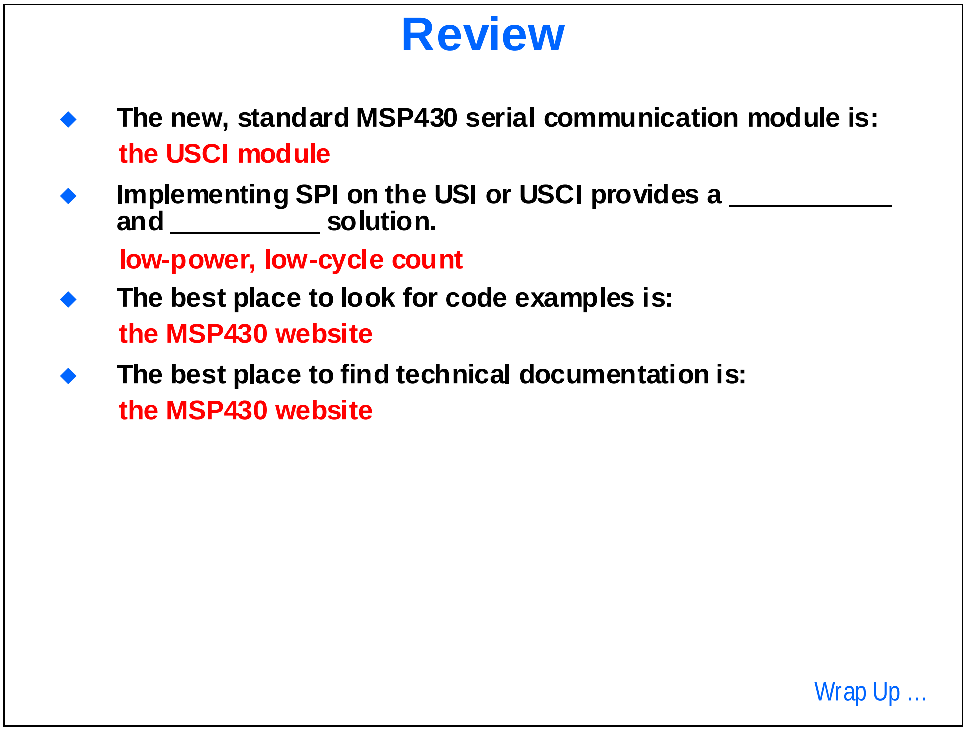
  To combine three interrupts into a single interrupt to the CPU. Also acts as an indicator for handler code to determine which interrupt triggered.

- In addition to normal timer functions, name some other functions the timer can perform.

  ADC12 hardware control, PWM, UART

Communication section ...

Communications Module Review Answers

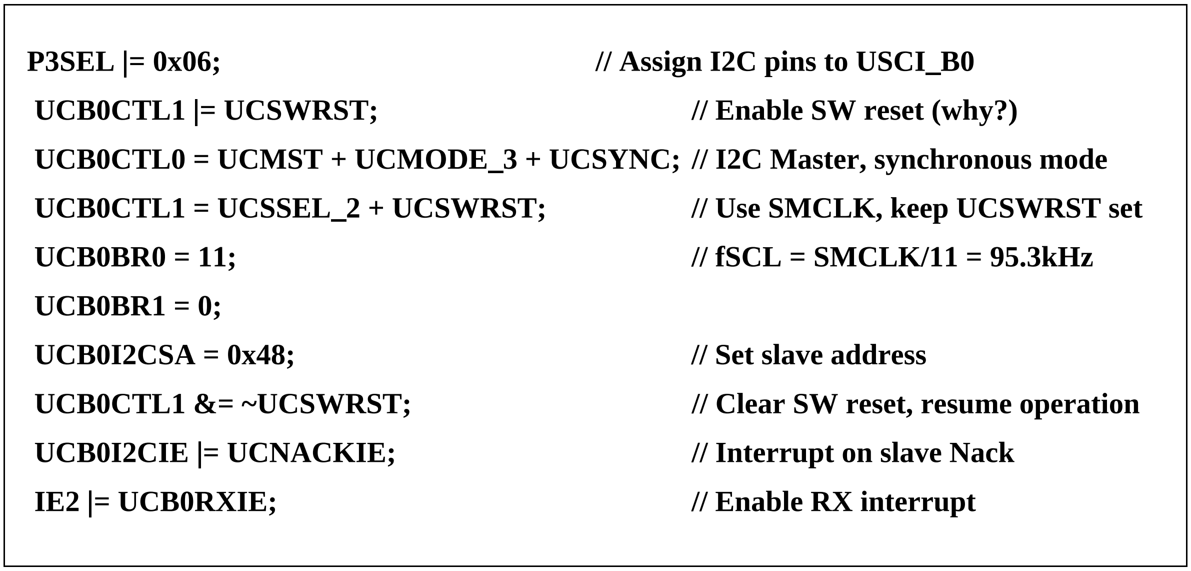


Review

- 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
```