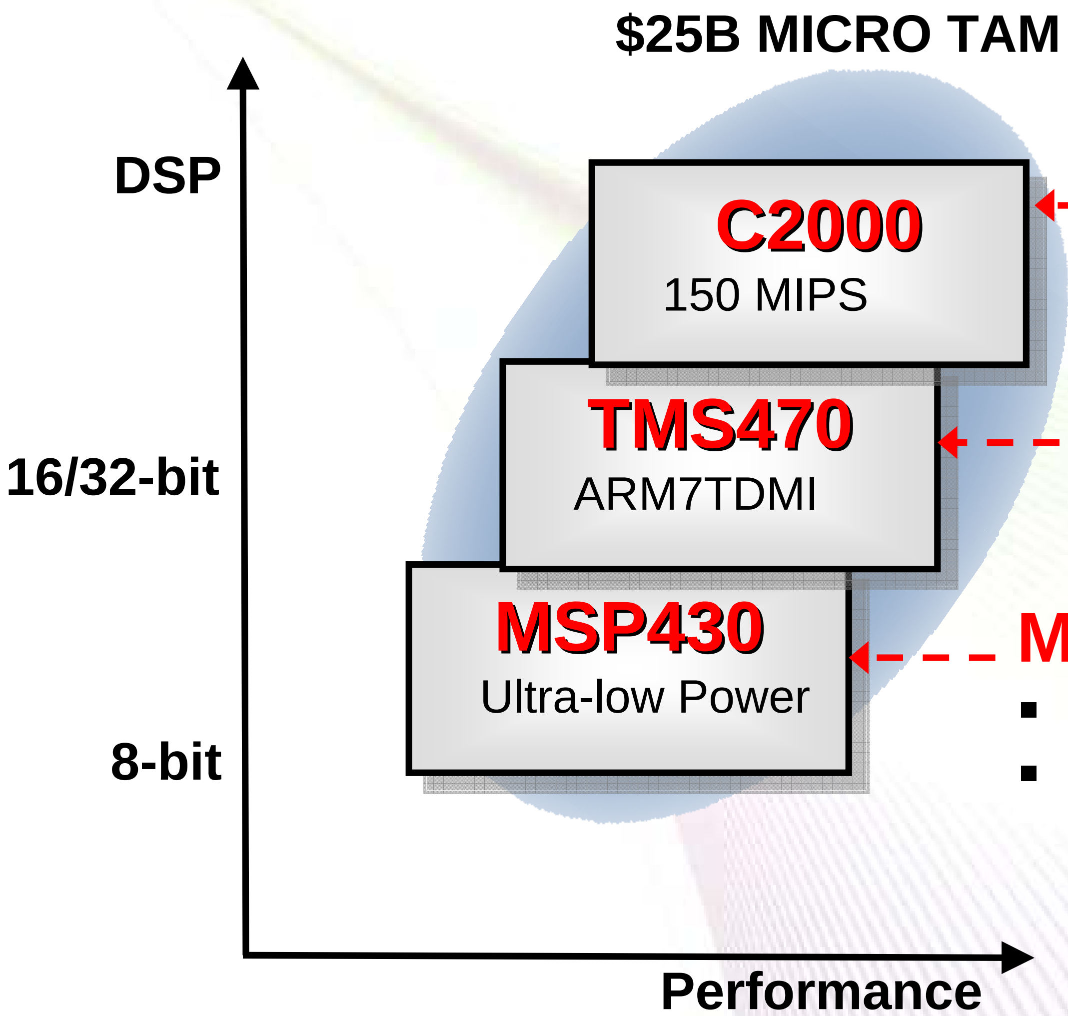
Intro: Getting Started with MSP430

Mike Mitchell
MSP430 Applications Engineer
Texas Instruments

Agenda

- Intro to MSP430 Architecture
- Intro to Tools and I/O
- Intro to Low-Power with the MSP430

TI Microcontroller Portfolio

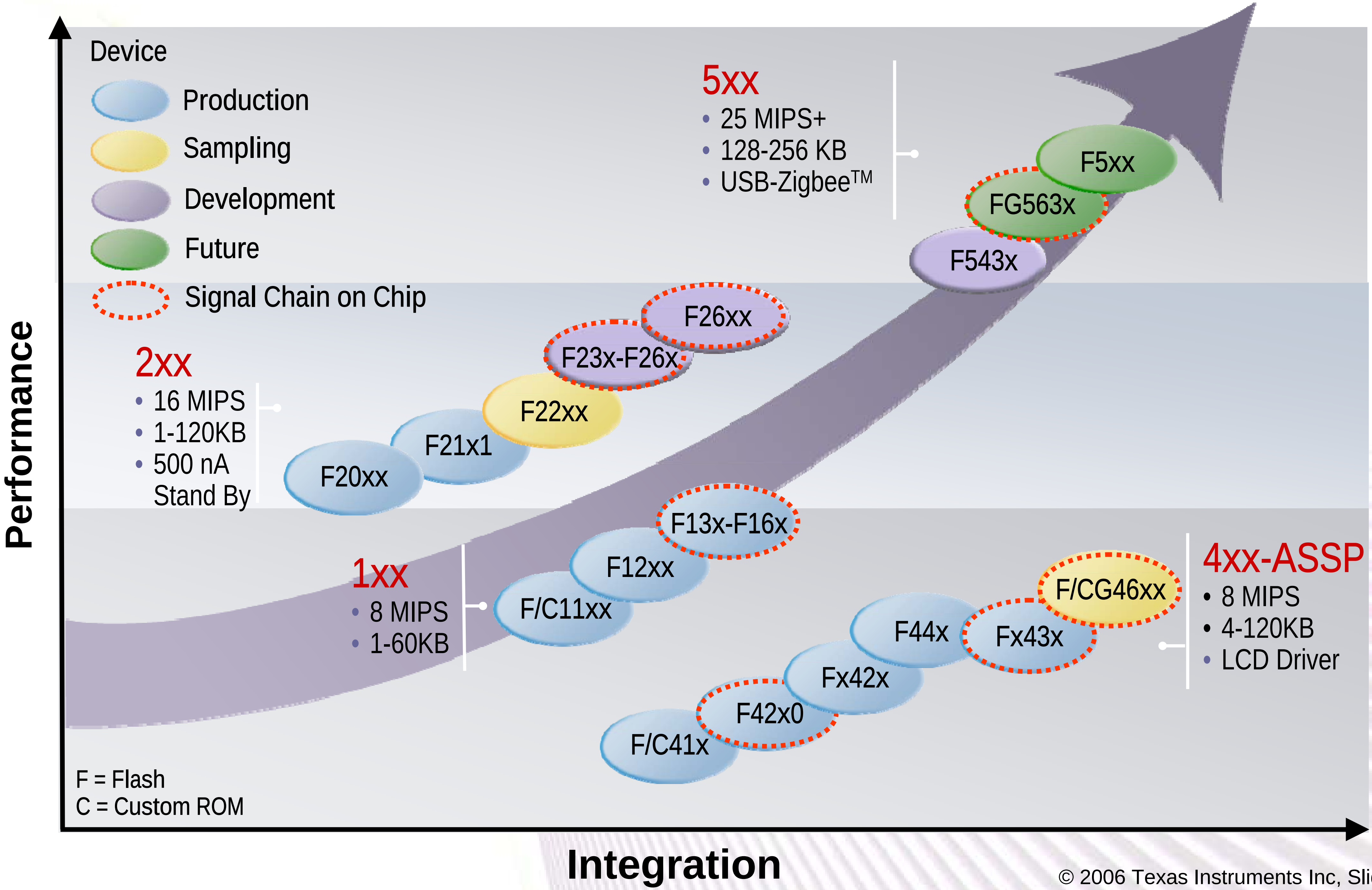


High-Performance
- Motor Control
- Digital Power Supply

Industry Standard
- Automotive
- General Purpose

Measurement
- Utility Metering
- Portable Instrumentation

MSP430 Products



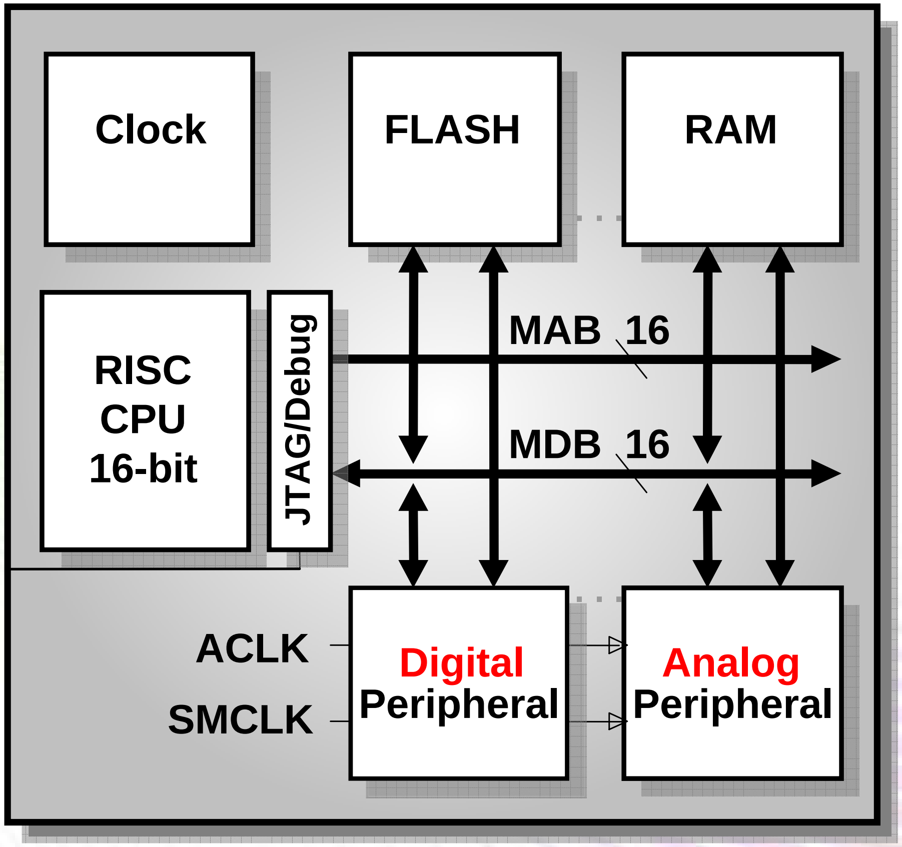
What Is The MSP430?

Ultra-low Power

- 0.1uA power down
- 0.8uA standby mode
- 250uA / 1MIPS
- <1us clock start-up
- <50nA port leakage
- **Zero-power** BOR

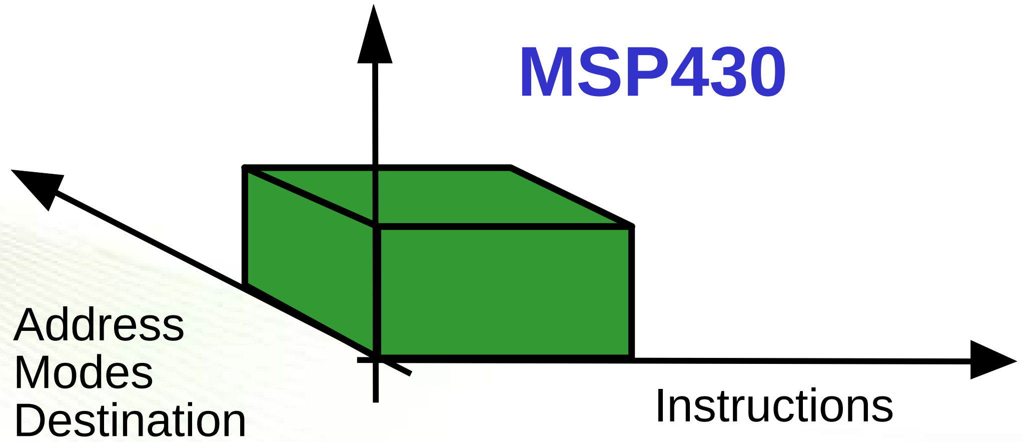
Ultra-Flexible

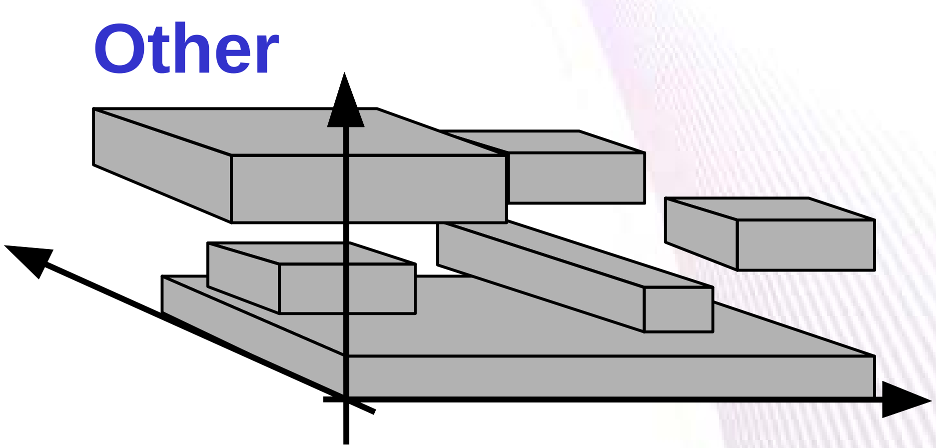
- 1k-128kB ISP Flash
- 14-100 pin options
- USART,I2C, Timers
- 10/12/16-bit ADC
- DAC, OP Amp, LCD driver
- **Embedded emulation**



Orthogonal Architecture

- Clear and consistent with no exceptions
- Compiler efficient

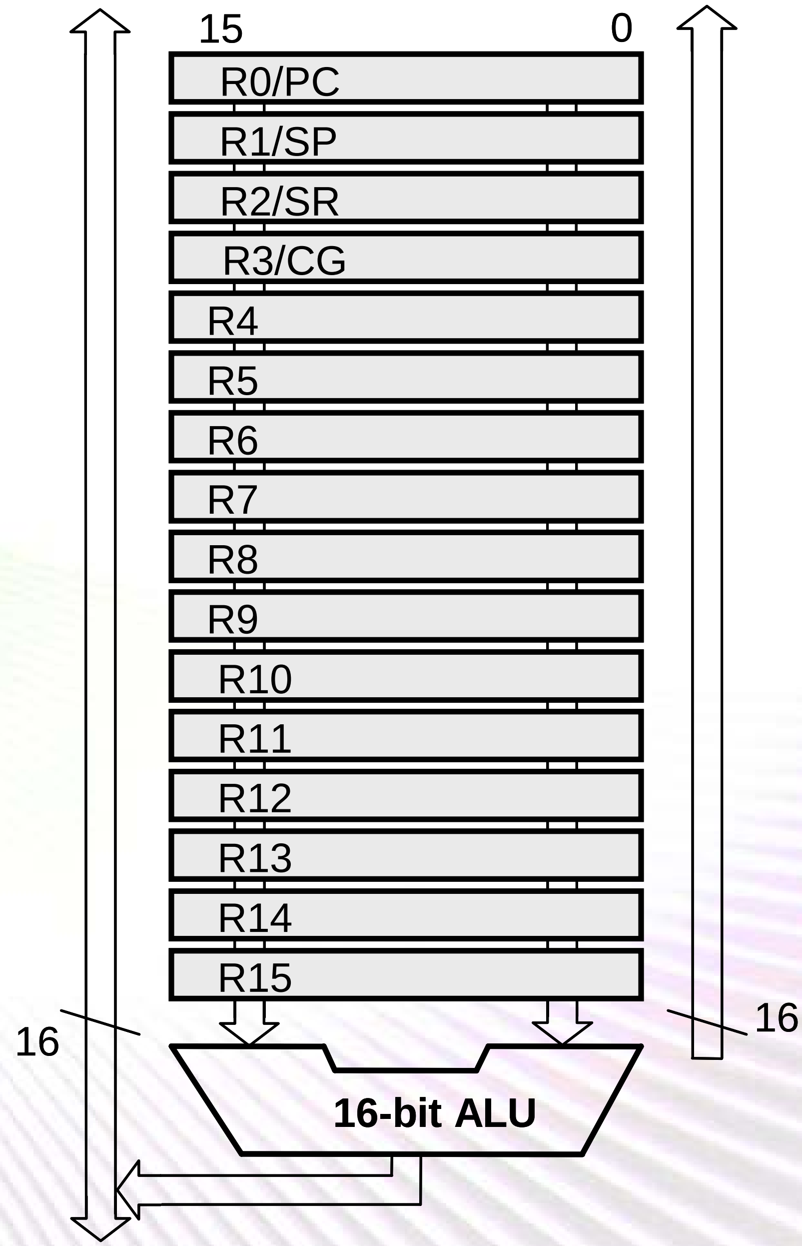




- Special instructions to learn
- Complicated and inefficient

16-bit RISC CPU

- Single-cycle register file
  - 4 special purpose
  - 12 general purpose
  - No accumulator bottleneck
- RISC architecture
  - 27 core instructions
  - 24 emulated instructions
  - 7 addressing modes
- Memory-to-memory atomic addressing
- Bit, byte and word processing



Deep Register File RISC Advantage

```
;-----------------------------------------------------------------
BIN2BCD; Binary Number (R12) -> Packed BCD (R14|R13)
;-----------------------------------------------------------------
        mov     #16,R15             ; Loop Counter
        clr     R14                 ; 0 -> RESULT MSD
        clr     R13                 ; 0 -> RESULT LSD
BIN1    rla     R12                 ; Binary MSB to carry
        dadd    R13,R13             ; RESULT x2 LSD
        dadd    R14,R14             ; MSD
        dec     R15                 ; Through?
        jnz     BIN1                ; Not through
        ret
```

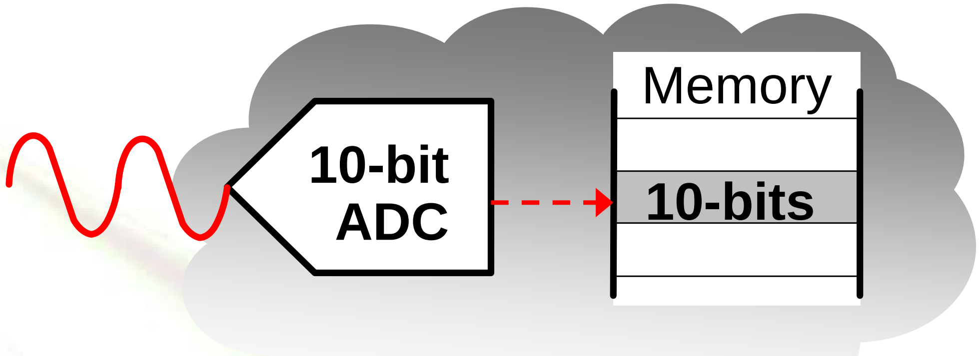
- Access to many single-cycle registers
- Useful for calculation intensive functions

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**

Do You Use A 10+ Bit ADC?




```
; Other MCU
movf     ADCRESH,W
movwf    RAMH
bsf      STATUS,0x20
movf     ADCRESL,W
bcf      STATUS,0x20
movwf    RAML
```

84 bits / 24 cycles

```
; MSP430
mov      ADC10MEM,RAM
```

48 bits / 6 cycles

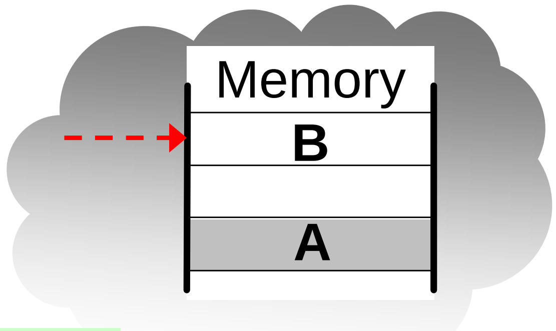
Seven Addressing Modes

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 location**
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

Atomic Addressing

B=B+A




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

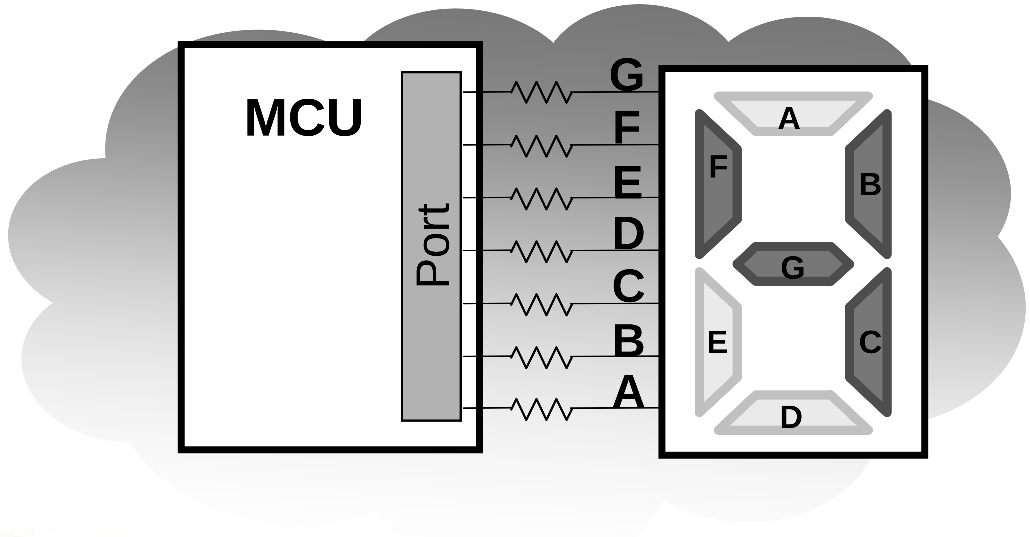
```
; MSP430
add     A,B
```

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

Write Less Code

```
; Other MCU
;    movlw   HIGH Tab
     movwf   PCLATH
     movwf   DispVal,W
     call    Tab
     movwf   PORTB
     goto    Continue
Tab  addwf   PCL,F
     retlw   B'00111111'
     retlw   B'00000110'
     retlw   B'01011011'
     retlw   B'01001111'
     retlw   B'01100110'
     retlw   B'01101101'
     retlw   B'01111101'
     retlw   B'00000111'
     retlw   B'01111111'
     retlw   B'01101111'
Continue
```

238 bits / 48 cycles




```
;    MSP430
;    mov.b Tab(DispVal),P1OUT
Tab  DW     0063Fh
     DW     04F5Bh
     DW     06E66h
     DW     0077Ch
     DW     0677Fh
```

128 bits / 6 cycles

Effect Of The 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!

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!

Three Instruction Formats

Op-Code	S-Register	Ad	B/W	As	D-Register

```
; Format I Source and Destination
5405          add.w  R4,R5            ; R4+R5=R5 xxxx
5445          add.b  R4,R5            ; R4+R5=R5 00xx
```

Op-Code	B/W	As	D/S-Register

```
; Format II Destination Only
6404          rlc.w  R4
6444          rlc.b  R4
```

Op-Code	Condition	10-bit PC offset

```
; Format III There are 8(Un)conditional Jumps
3C28          jmp    Loop_1           ; Goto Loop_1
```

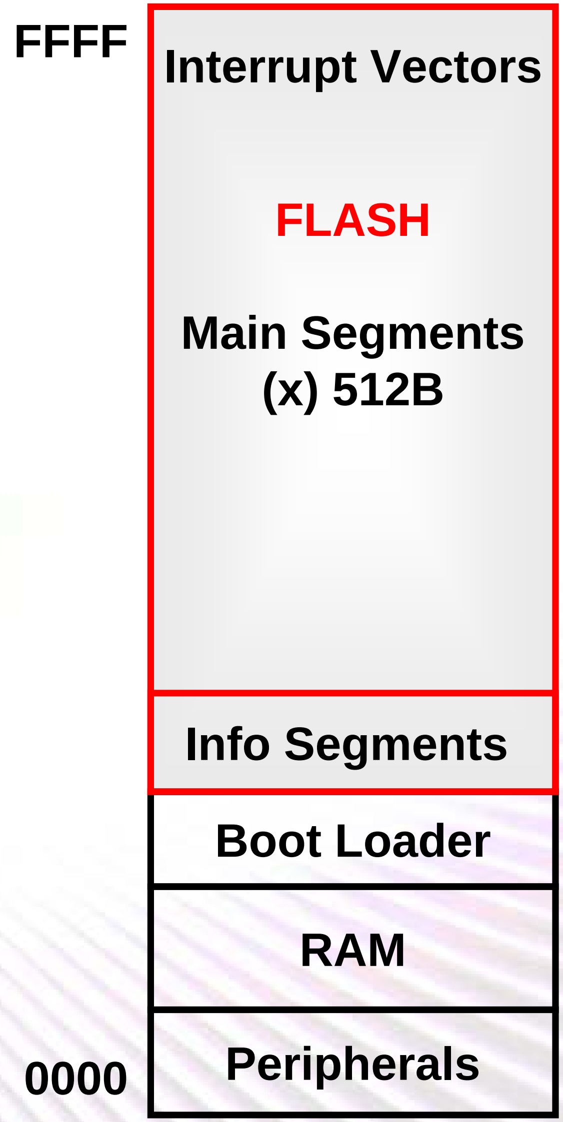
51 Total Instructions

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)		

Unified Memory Map

- **Absolutely no paging**
- **Supports code agility**
- **ISP Flash**
  - Self programming
  - JTAG
  - Bootloader

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



Programming 'F2131 8KB Flash?

- f_{FTG} = __________?
- t_{WORD} = _______?
- Program word or byte = ________________?

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		
$t_{Block,\ 1\text{-}63}$	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 ($t_{FTG} = 1/f_{FTG}$).

Reset Conditions

- RST/NMI configured in the reset mode
- 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)

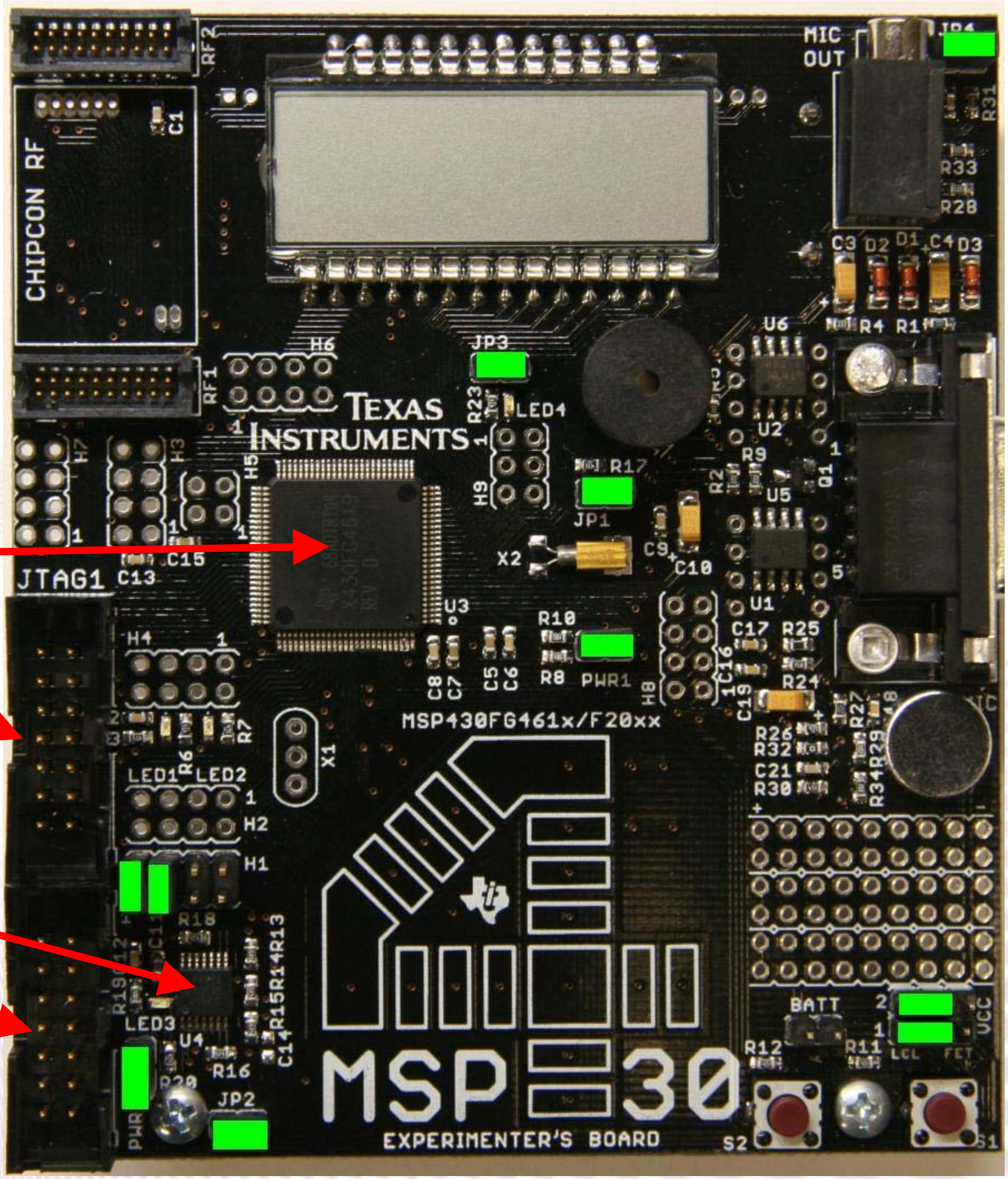
Always refer to Family users guide

ATC Board

- **Default jumper settings**

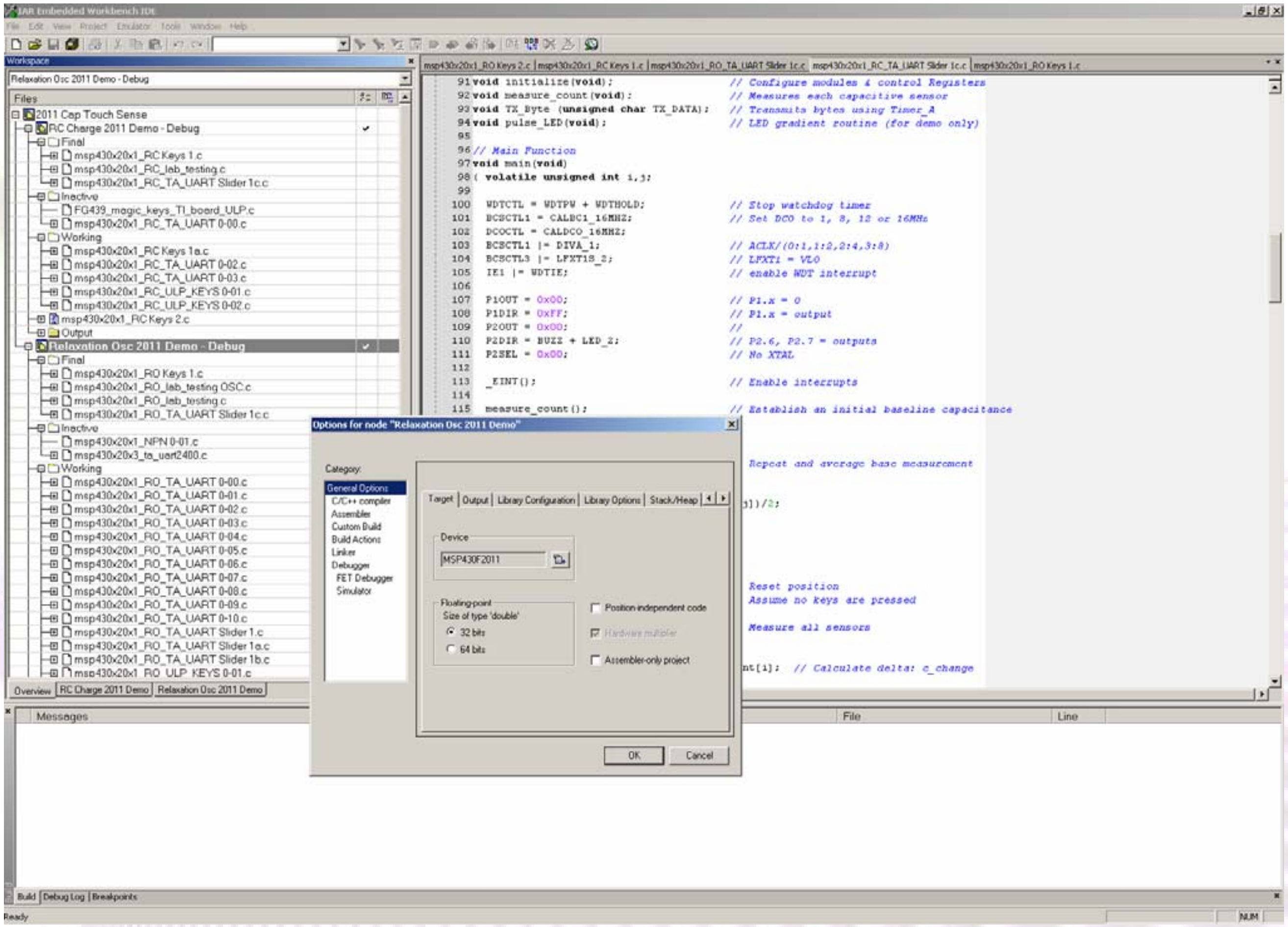
- **MSP430FG4619**
  - '4619 JTAG

- **MSP430F2013**
  - '2013 JTAG



IAR Kickstart Tour

- **Instructor-led tour of Kickstart**



Getting Started Lab1: Blinky

- **Blink the LED with assembly code**

Lab1: Flash The LED First Program

```
#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
```

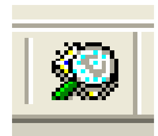
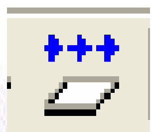
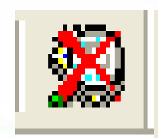
Lab1: Step-by-Step

- **Connect the FET interface to the PC USB port**
- **Connect the JTAG cable to the ‘2013 JTAG port on the board**
- **Connect the BATT jumper on your board**
- **Launch IAR**
- **Create new workspace**
  - File → New → Workspace
- **Create new project**
  - Project → Create new project → Click “OK” on dialogue pop-up
  - Name the project (Lab1) and click “Save”

Lab1: Step-by-Step (cont.)

- **Configure the project**
  - Project → Options
    - Select MSP430F2013 from “Device” drop down menu (DDM)
    - Click “Assembler Only” project
    - Highlight “Debugger” in the Category list
    - Select “FET Debugger” from the “Driver” DDM
    - Highlight “FET Debugger” in the Category list
    - Select the “TI USB” in the “Connection” section
    - Click OK
- **Create the source file**
  - File → New → File
  - Type source from slide
  - Click save, name it, and save it

Lab1: Step-by-Step (cont.)

- **Add source file to the project**
  - Project → Add Files
  - Select “Assembler Files” from “Files Of Type” DDM
  - Select your file and click Open
- **Click the “Debug” button in IAR:** 
- **Name and save workspace when prompted**
- **Click “OK” about the Stack Plug-in**
- **Click “Go” in IAR:** 
- **Your done! Click “Stop Debugging” button:** 
- **Pull the power jumper**

Lab1: For Future Reference

2.2.2 Creating a Project from Scratch

The following section presents step-by-step instructions to create an assembler or C project from scratch, and to download and run the application on the MSP430. Refer to Project Settings above. Also, the MSP430 IAR Embedded Workbench IDE User Guide presents a more comprehensive overview of the process.

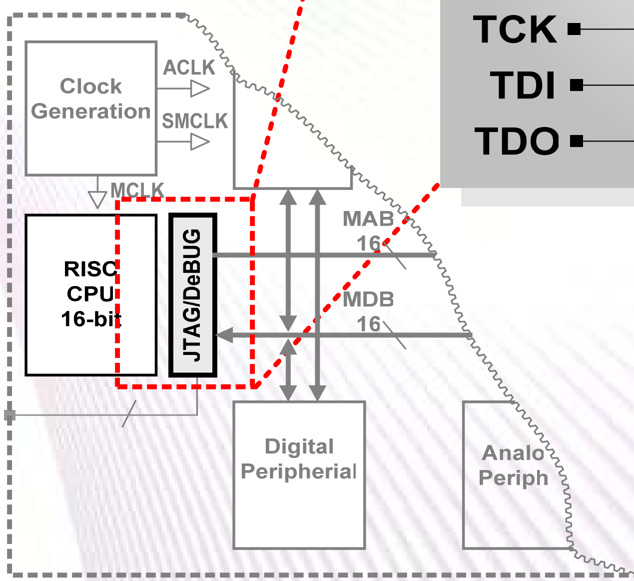
- **Previous instructions summarized section 2.2.2 of FET User's Guide (navigate from START menu)**

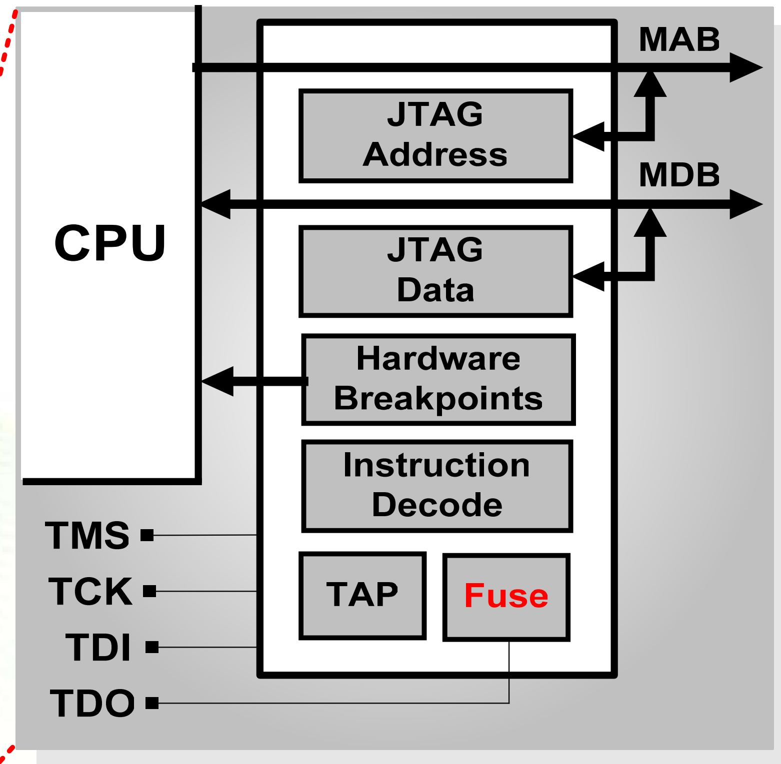
Agenda

- Intro to MSP430 Architecture
- Intro to Tools and I/O
- Intro to Low-Power with the MSP430

Embedded Emulation

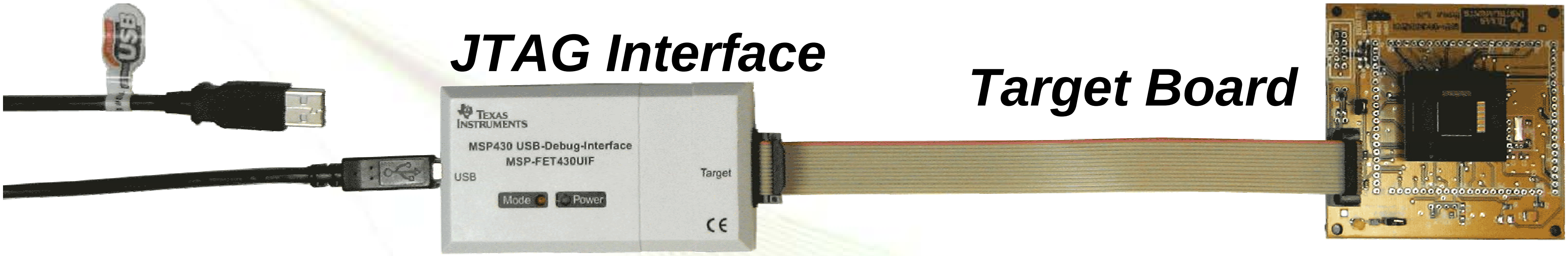
- Debug real time in system
  - No application resources used
  - Full speed
  - Breakpoint
  - Single step
  - Complex trigger
  - Trace
- Security Fuse





MSP-FET430

USB FET Interface:



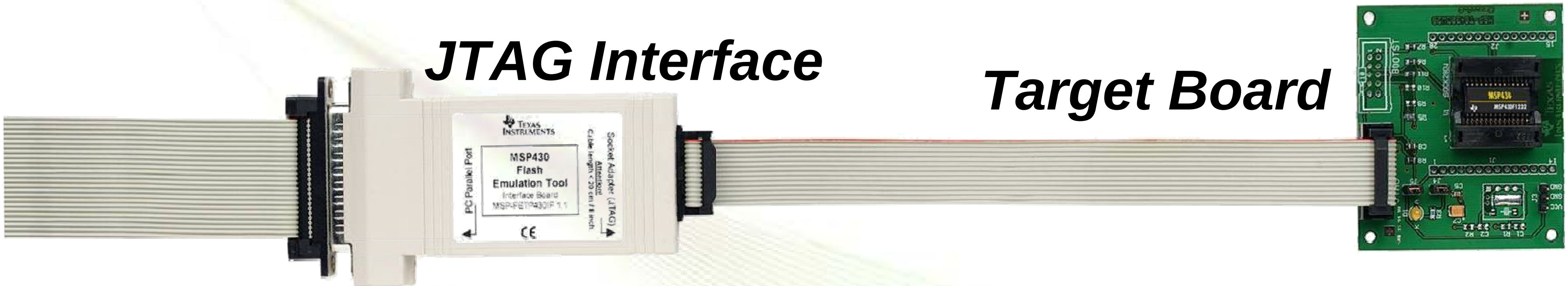
Part Number	Product Family	Price
MSP-FET430U28	**MSP430x11x1A, MSP430x12x/x1xx2**	**$149.00**
MSP-FET430U64	**MSP430x13x/x14x/x15x/x16x**	**$149.00**
MSP-FET430U80	**80-pin MSP430x43x/MSP430x44x**	**$149.00**
MSP-FET430U100	**MSP430FG43x, MSP430x43x**	**$149.00**

Interface only without target board:

Part Number	Product Family	Price
MSP-FET430UIF	**MSP430**	**$99.00**

MSP-FET430

Parallel FET Interface:

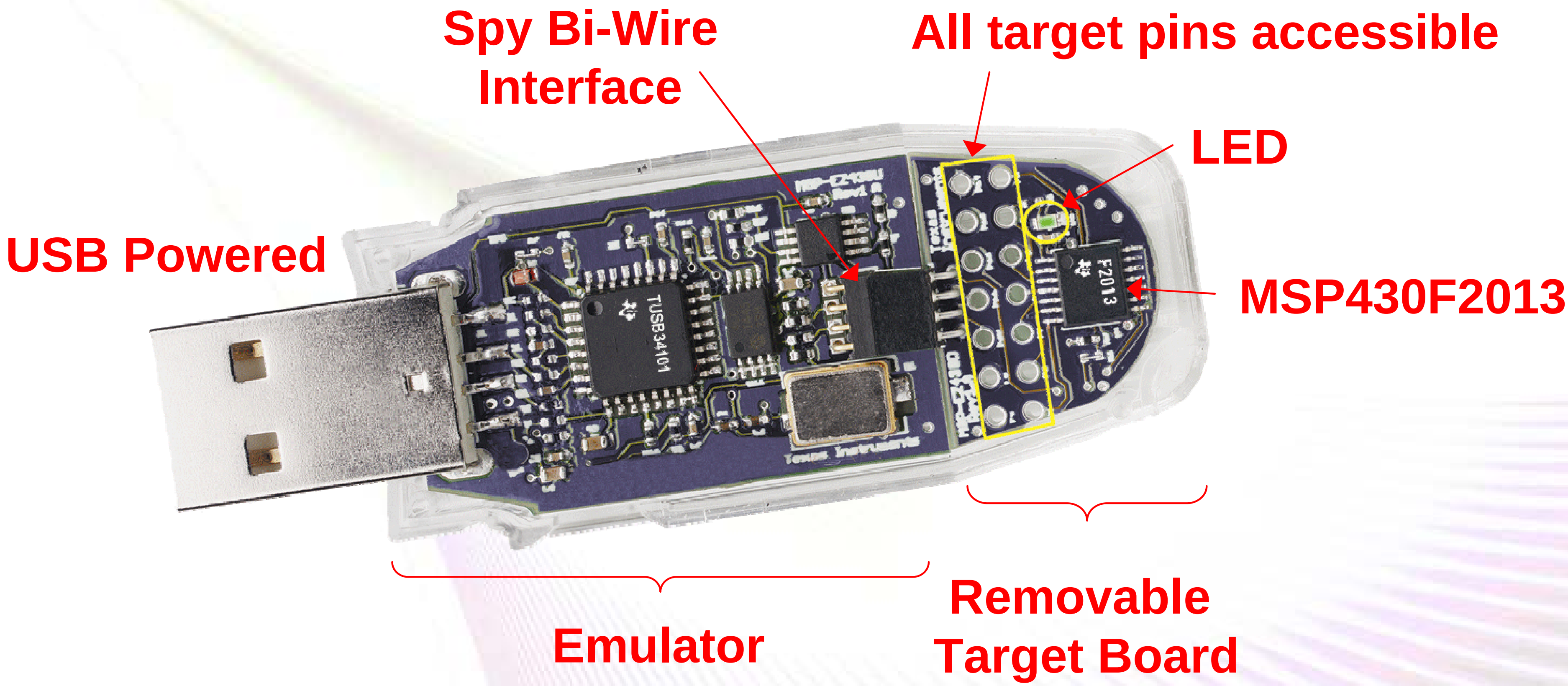


Part Number	Product Family	Price
MSP-FET430P120	**MSP430x11x1A, MSP430x12x/x1xx2**	**$99.00**
MSP-FET430P140	**MSP430x13x/x14x/x15x/x16x**	**$99.00**
MSP-FET430P410	**MSP430x41x, MSP430FE42x, MSP430FW42x**	**$99.00**
MSP-FET430P430	**MSP430FG43x, MSP430x43x**	**$99.00**
MSP-FET430P440	**MSP430F43x/44x**	**$99.00**

Interface only without target board:

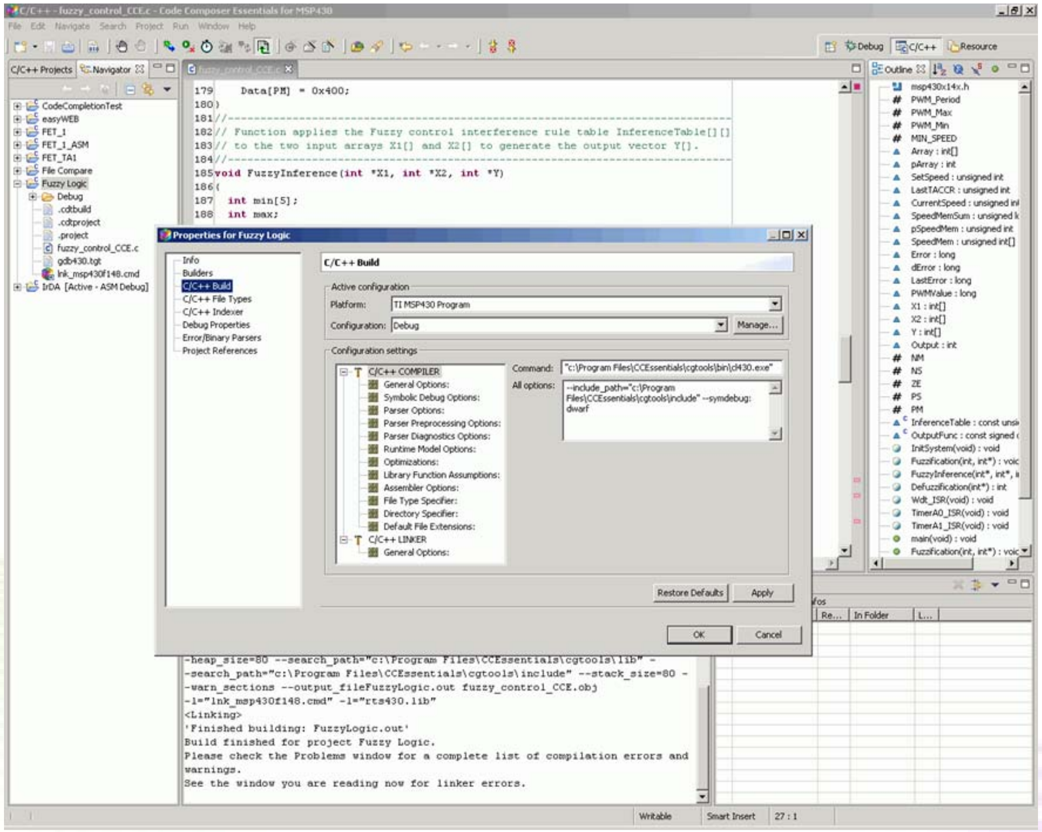
Part Number	Product Family	Price
MSP-FET430PIF	**MSP430**	**$49.00**

eZ430-F2013


Spy Bi-Wire Interface
All target pins accessible
LED
USB Powered
MSP430F2013
Emulator
Removable Target Board

IAR Kickstart IDE

- **MSP-FET430 IDE**
  - **Assembler/linker**
  - **4kB IAR compiler**
  - **Parallel or USB Interface**



Which IDE Do I Use?

- IAR Embedded workbench
- TI Code Composer Essentials
- Rowley, Quadravox, Image Craft, GCC, Others

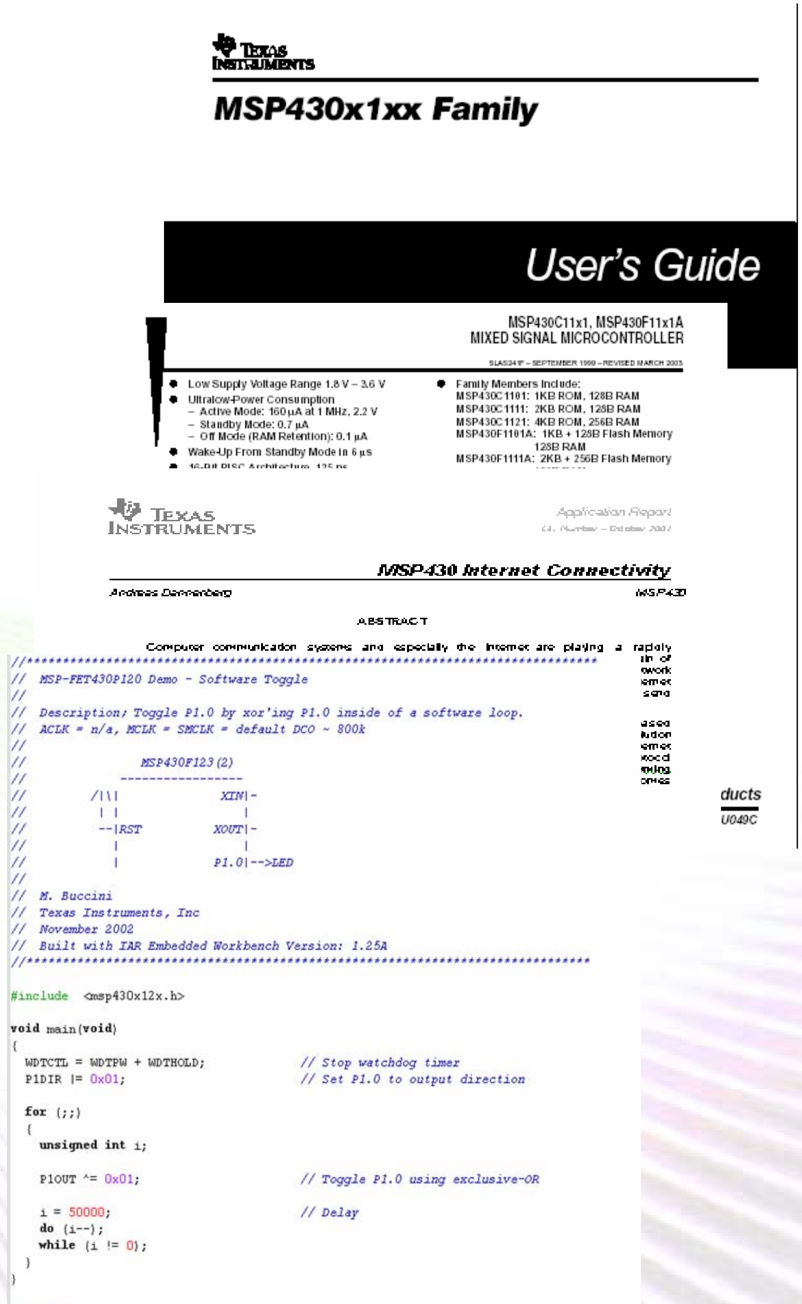
Check Third-Party website for complete list

- Most have 30-day trials

Check the Yahoo! user group for recommendations

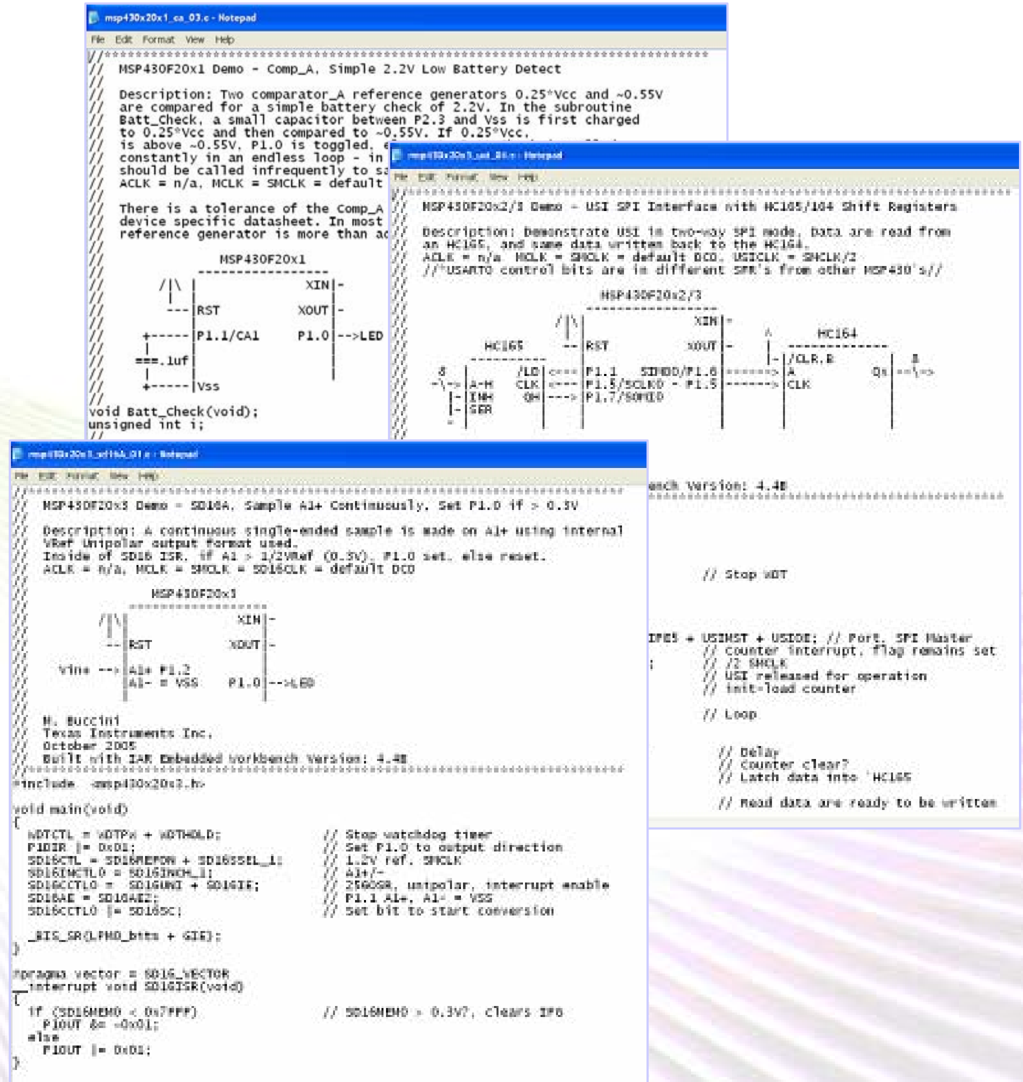
www.ti.com/msp430

- User's Guide
- Datasheets
- 100+ Application reports
- 500+ Code examples
- Complete 3rd party listing
- Errata



Code Examples!

- Reduce development time
- Over 1000 free examples
- Provided in C / assembler
- Use standalone
- Use as a template for your next project



Why Use Standard Definitions?

```
WDTCTL = 0x5A80;

WDTCTL = 0xA580;

WDTCTL = 0xA540;                // Hold watchdog

WDTCTL = WDTPW + WDTHOLD;       // Hold watchdog
```

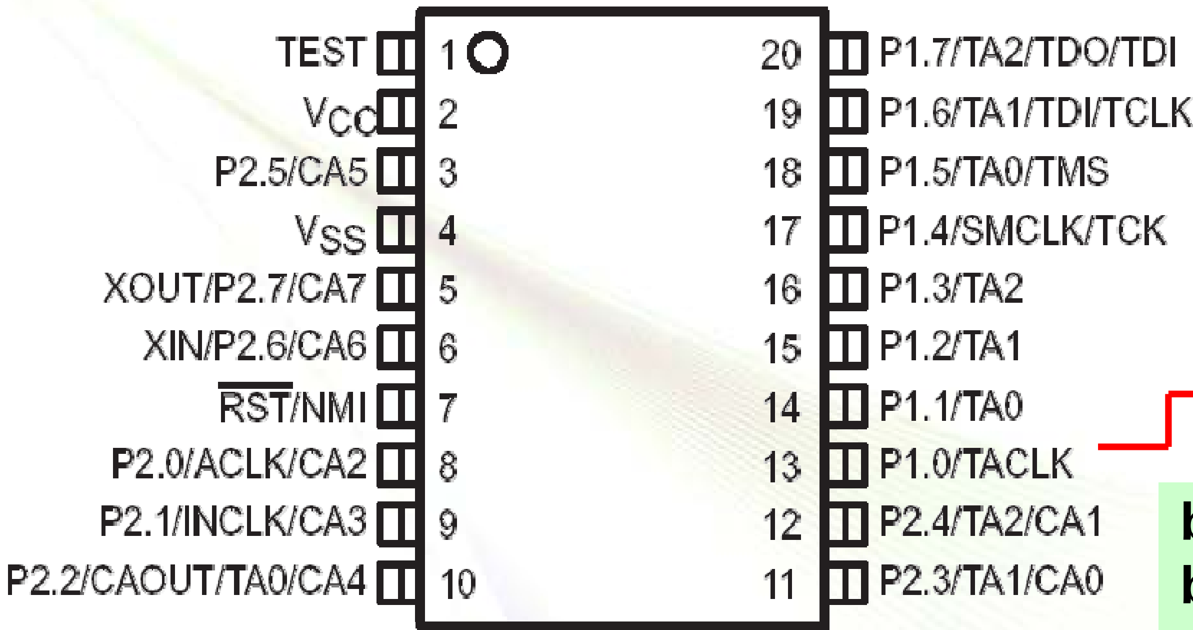
- **Which code line holds the watchdog?**

Getting Started Lab2: I/O Overview

- **Configure Port1 and Port2 of the MSP430FG4619**
  - 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

Port GPIO




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

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

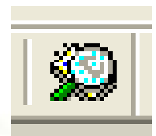
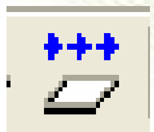
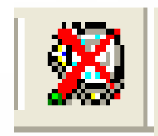
Input Register PxIN
Output Register PxOUT
Direction Register PxDIR
Function Select PxSEL
Interrupt Edge PxIES
Interrupt Enable PxIE
Interrupt Flags PxIFG

P1 and P2 only

Lab2: Step-by-Step

- **Move JTAG cable to ‘4619 and connect BATT jumper**
- **Launch IAR**
- **Open previous workspace when prompted**
- **Create new project as before**
  - Project → Create new project → Click “OK” on dialogue pop-up
  - Name the project (Lab2) and click “Save”
- **Configure the project**
  - Project → Options
    - Select MSP430FG4619 from “Device” drop down menu (DDM)
    - Highlight “Debugger” in the Category list
    - Select “FET Debugger” from the “Driver” DDM
    - Highlight “FET Debugger” in the Category list
    - Select the “TI USB” in the “Connection” section
    - Click OK

Lab2: Step-by-Step (cont.)

- **Add source file to the project**
  - Project → Add Files
  - Select “Getting_Started_Lab2.c” and click Open
- **Complete the code**
- **Click the “Debug” button in IAR:** 
- **Click “Go” in IAR:** 
- **Test and debug your code**
- **When done, click “Stop Debugging” button:** 
- **Pull the power jumper**

Getting Started Lab2: I/O

```
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 intererupt
  _EINT();                  // Enable interrupts
  while (1);
}

// P1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void P1ISR (void)
{
    …………
    P1IFG &= ~____;         // Clear P1IFG
}
```

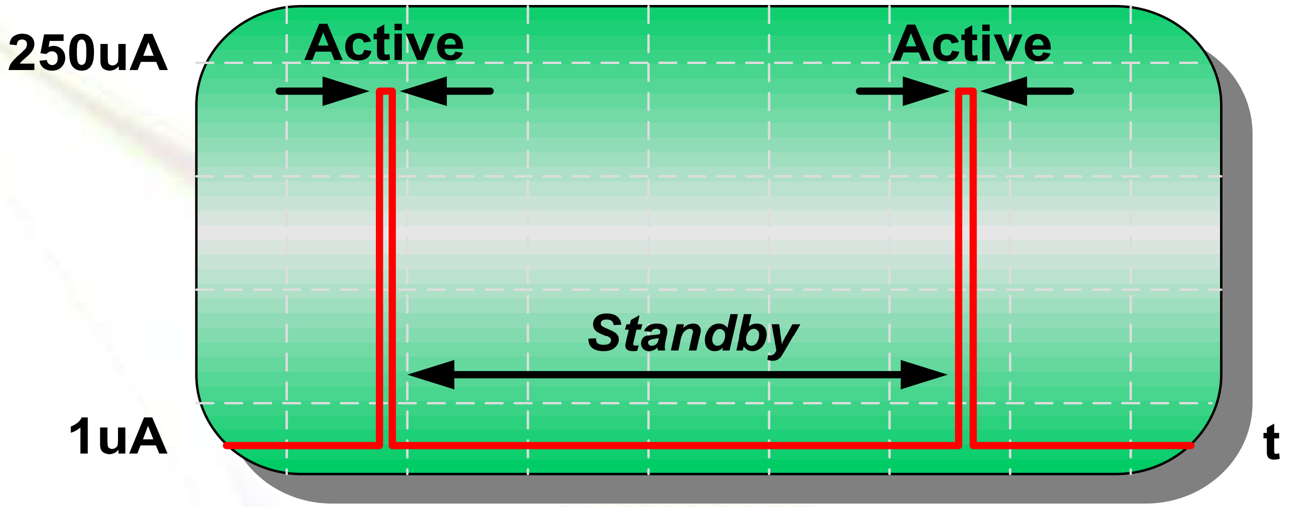
Getting Started Lab2 – Solution

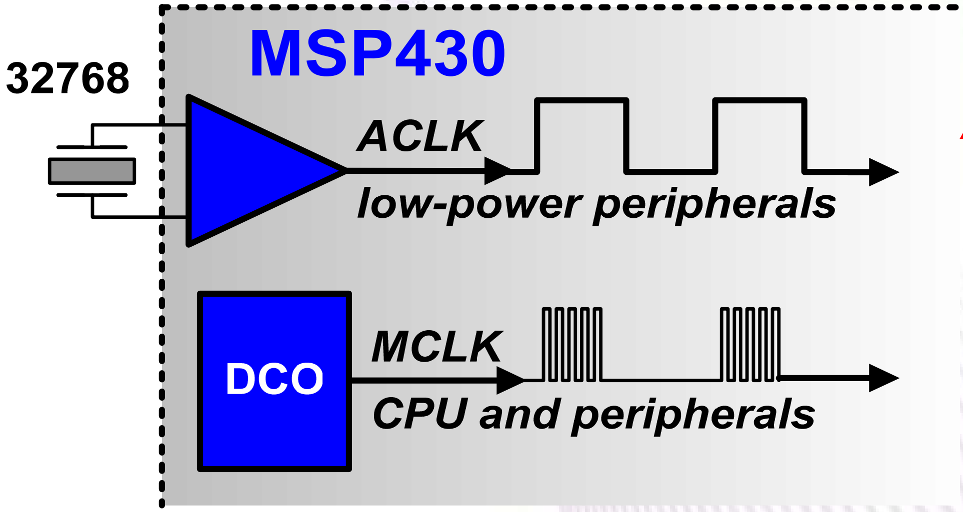
```
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD; // Stop WDT
  FLL_CTL0 |= XCAP14PF;     // Configure load caps
  P2DIR = BIT1;             // Set P2.1 to output direction
  P1IES = BIT0;             // H-L transition
  P1IE = BIT0;              // Enable interrupt
  _EINT();                  // Enable interrupts
  while (1);
}

// P1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void P1ISR (void)
{
    …………
    P1IFG &= ~BIT0;         // Clear P1IFG
}
```

Agenda

- Intro to MSP430 Architecture
- Intro to Tools and I/O
- Intro to Low-Power with the MSP430

Ultra-low Power Activity Profile

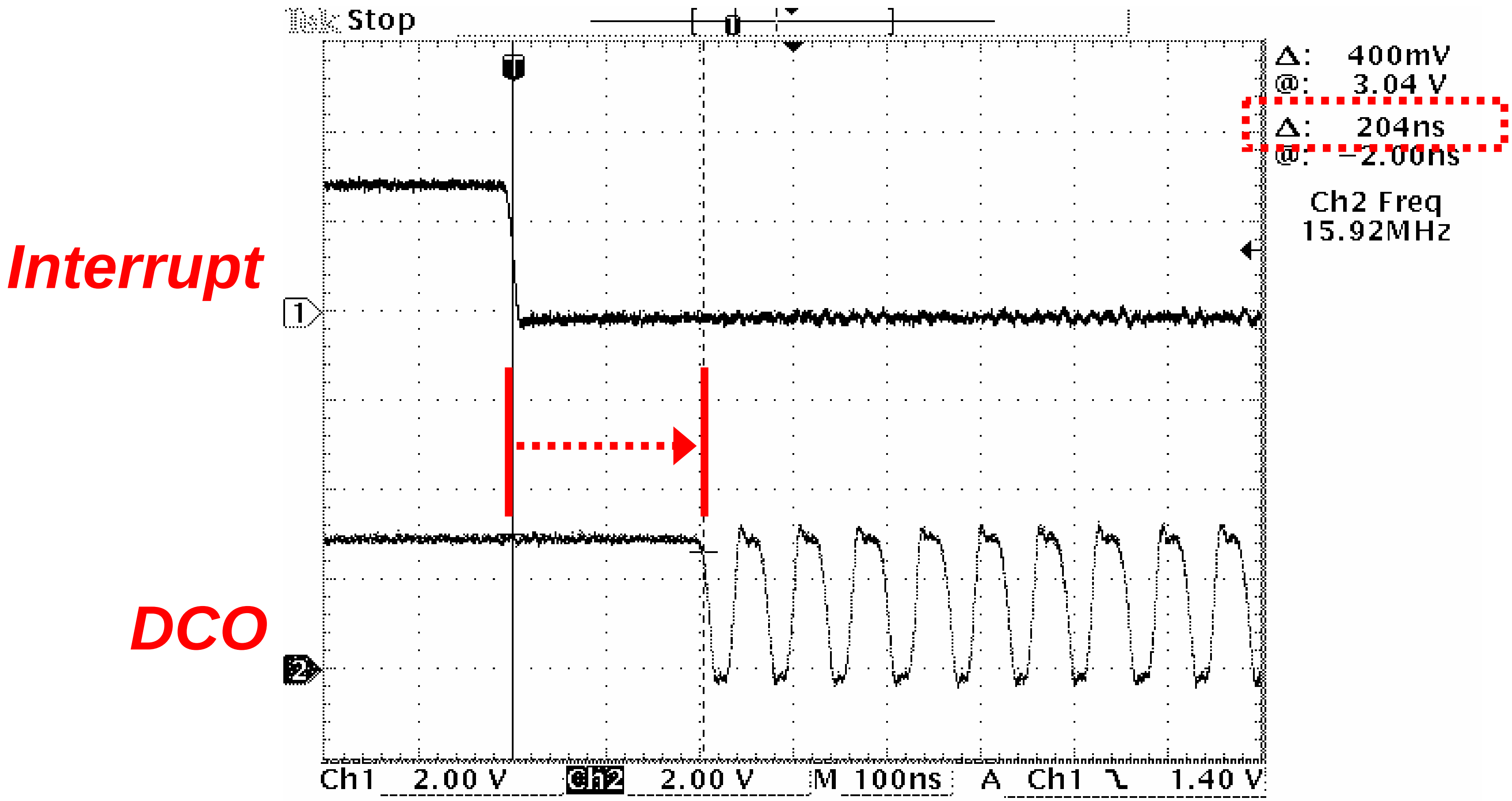




Always-on

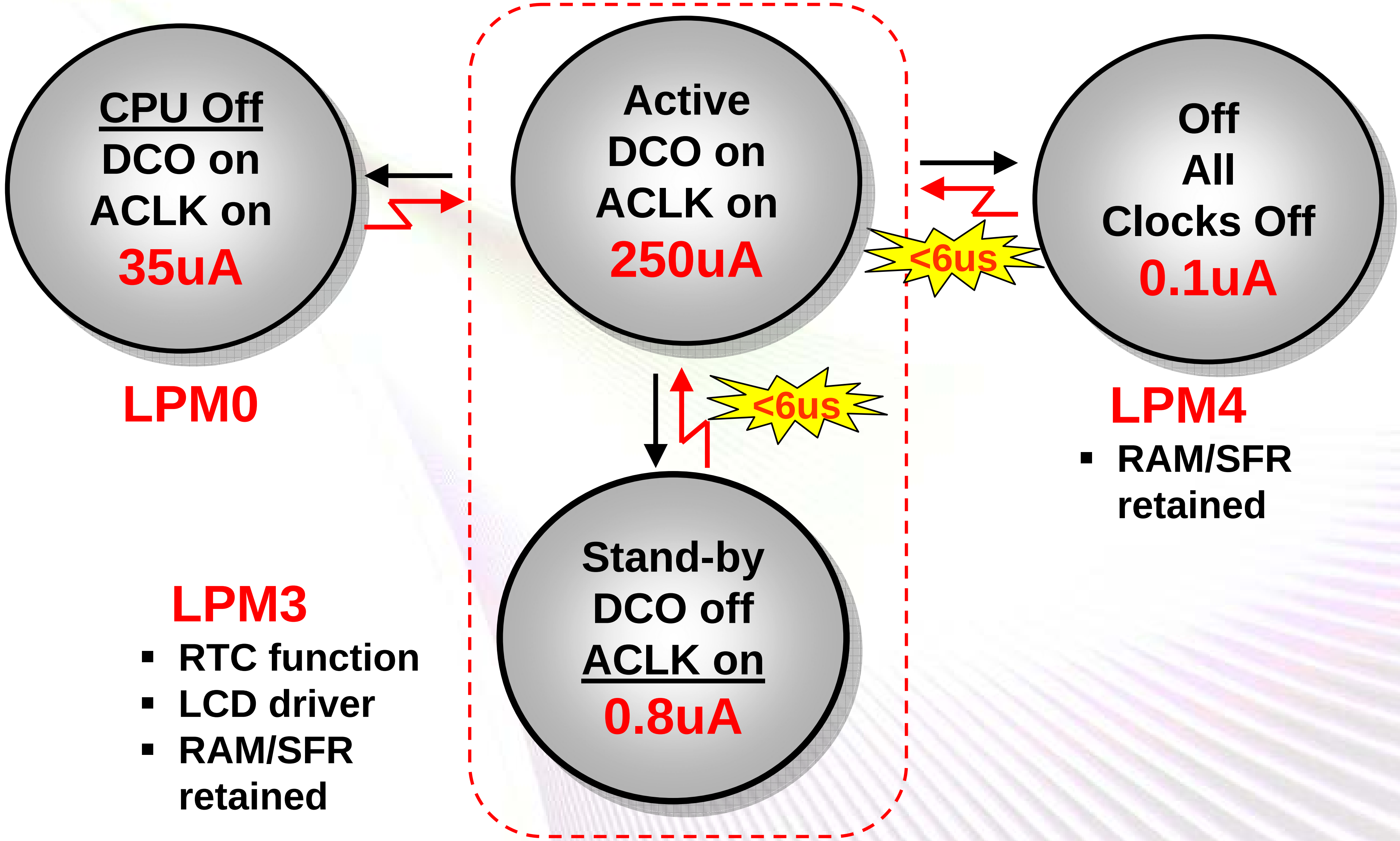
On demand

Performance on Demand



Immediate-stable clock start for reaction to events

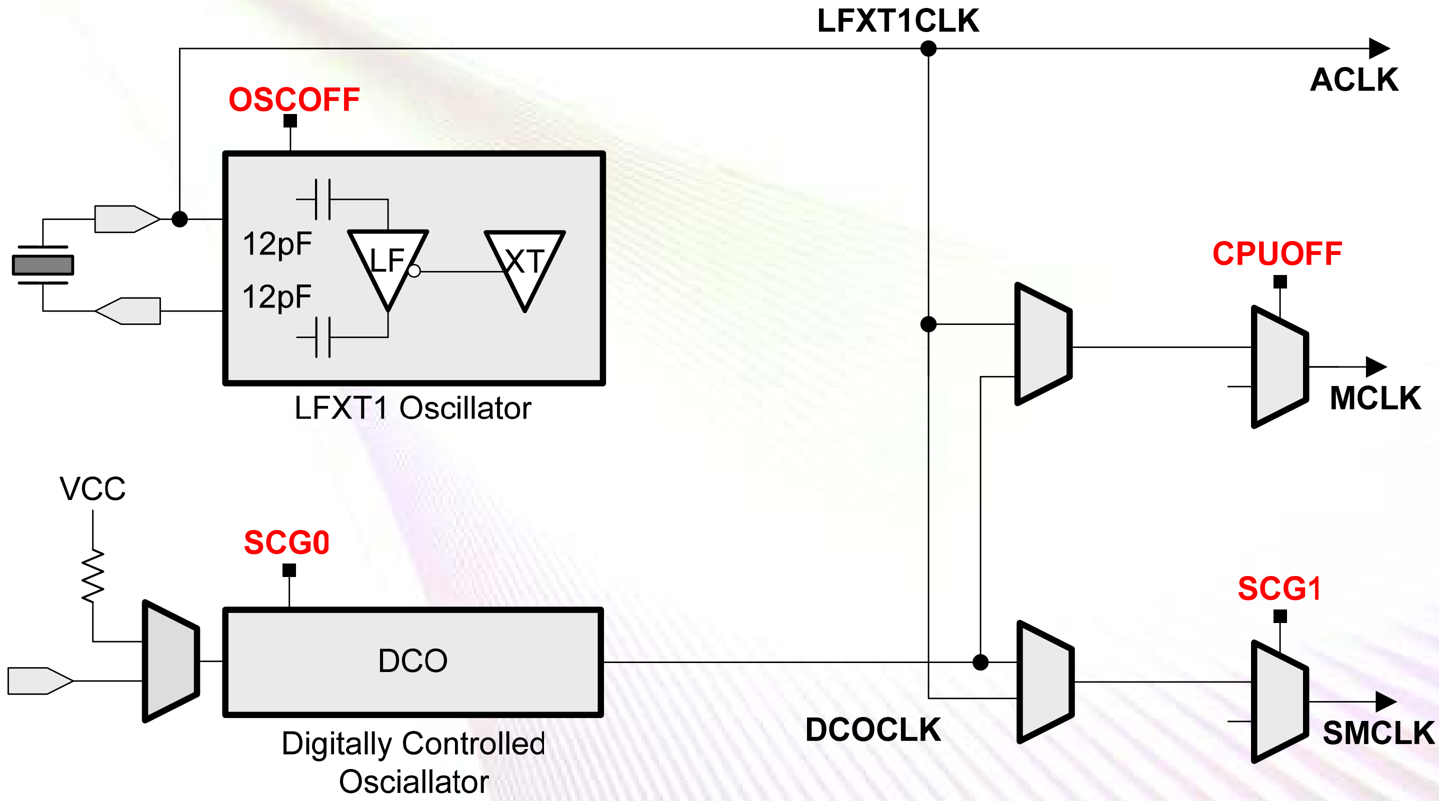
Ultra-low Power Clock Control



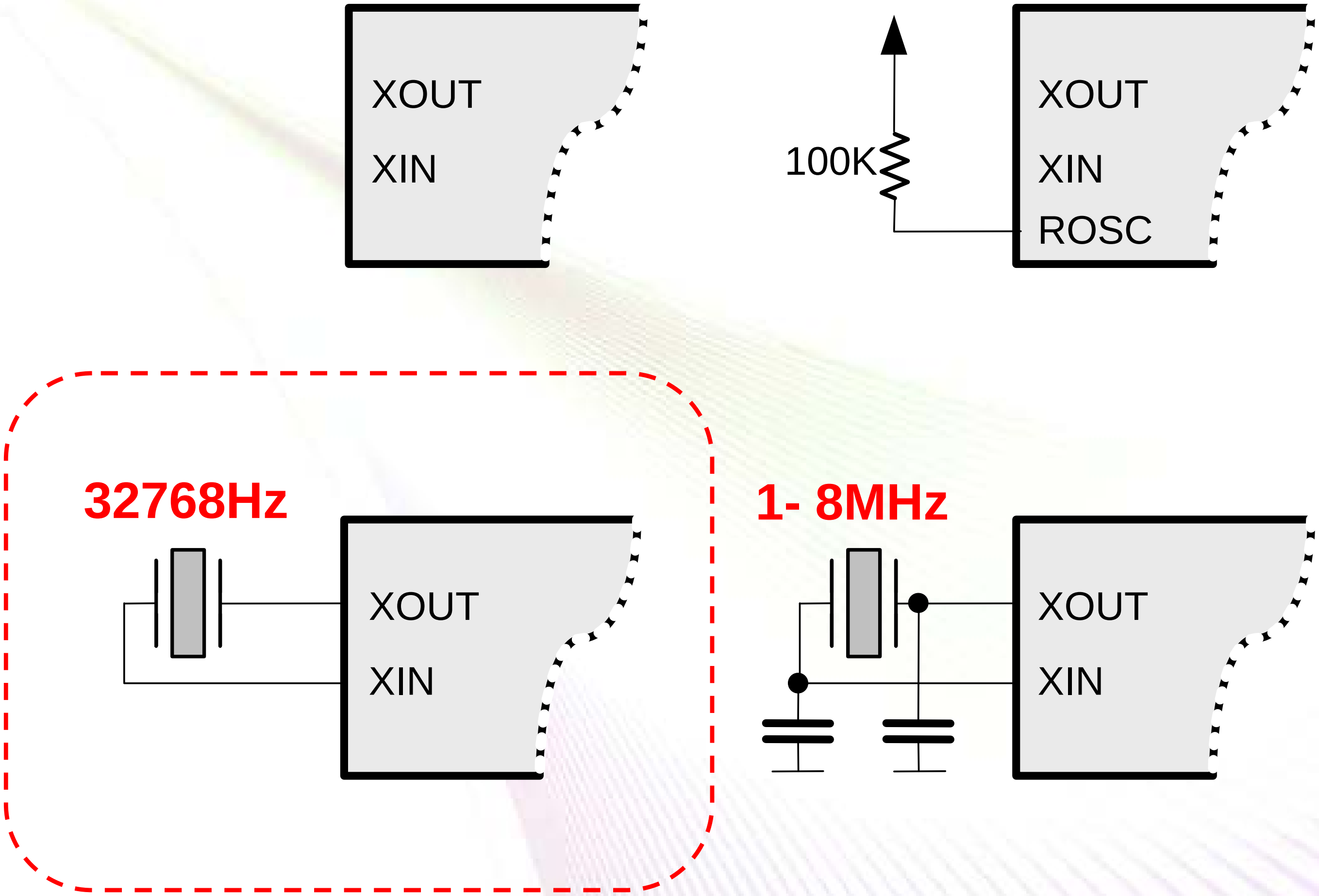
MSP430x11x/12x Basic Clock

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

R2/SR

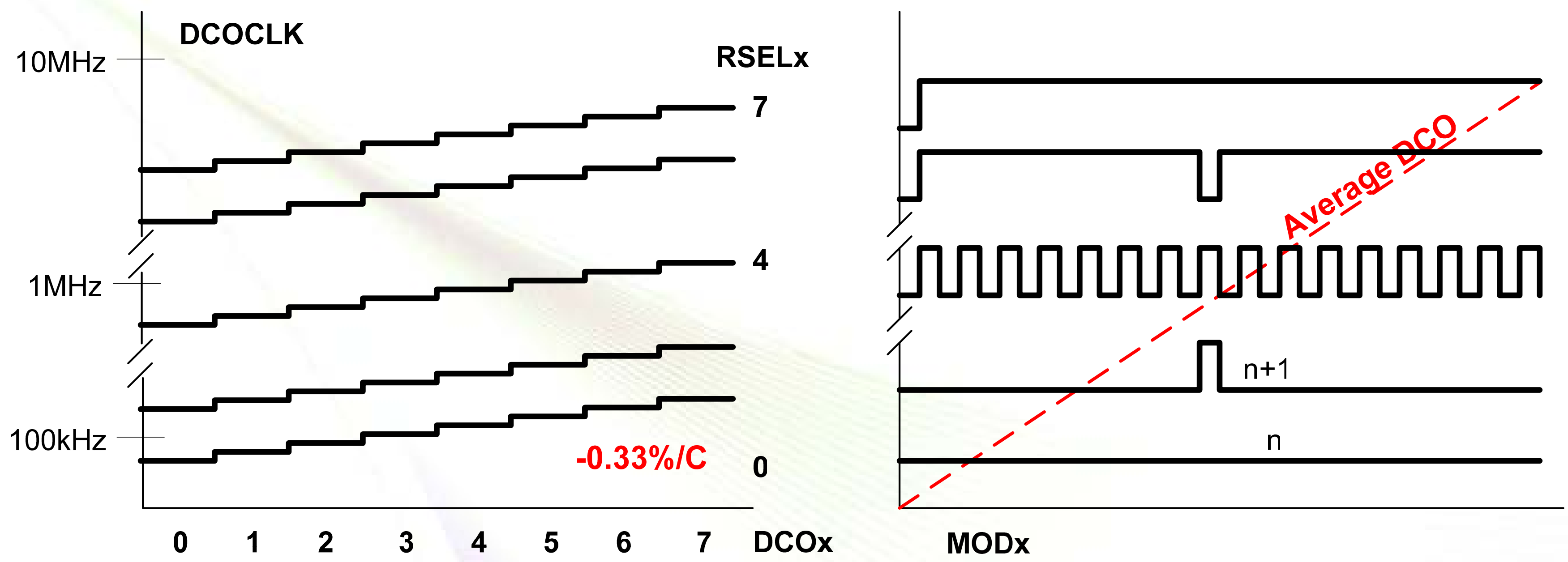


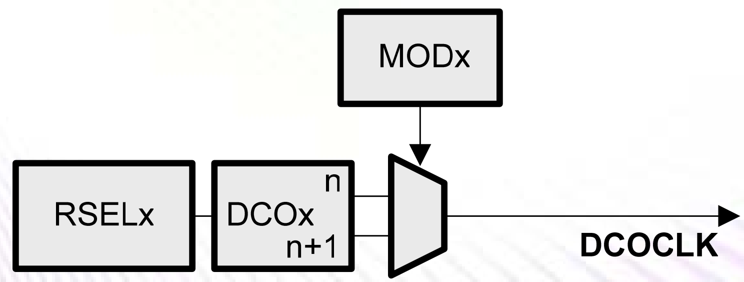
‘1xx Basic Clock XTAL Options



- Most MSP430 applications use a 32768 XTAL

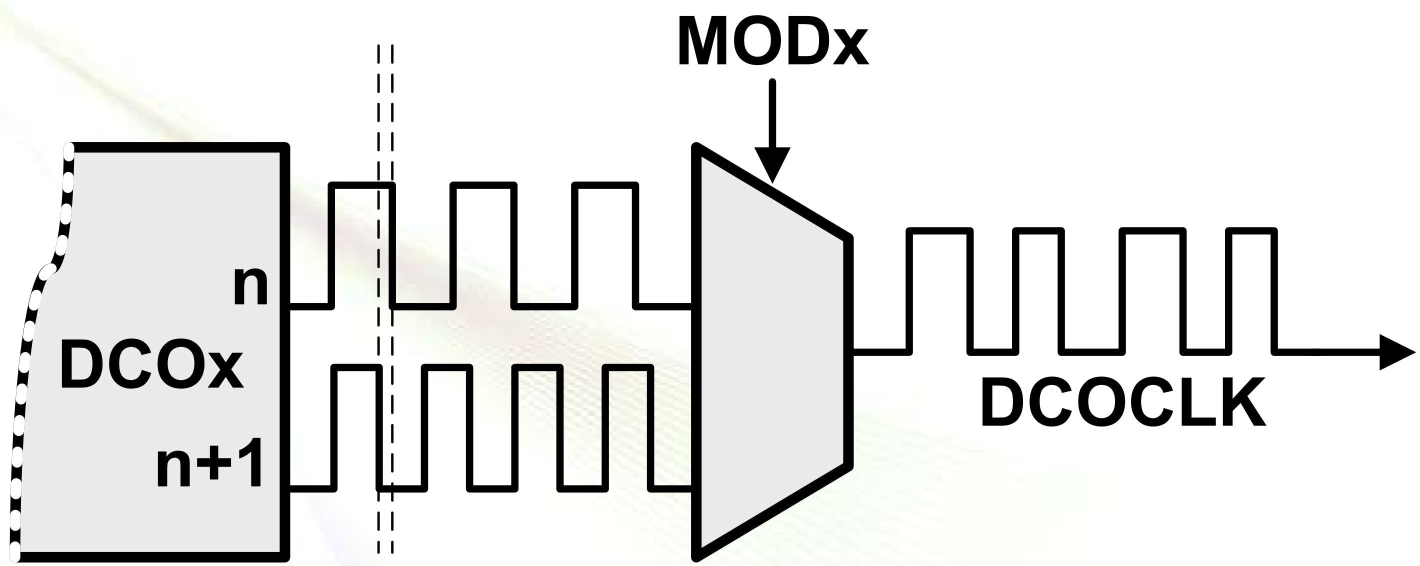
‘1xx Basic Clock DCO Control





XT2OFF	XTS	DIVA1	DIVA0	XT5V	RSEL2	RSEL1	RSEL0	DCO2	DCO1	DC00	MOD4	MOD3	MOD2	MOD1	MOD0

Does the DCO Have Jitter?

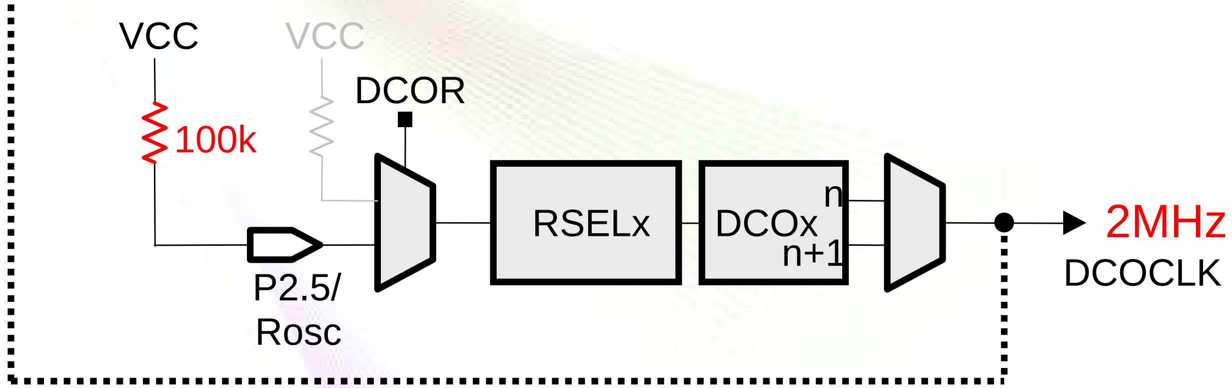


- What are the benefits of mixing two frequencies?

‘1xx DCO Calibration

4096Hz
ACLK

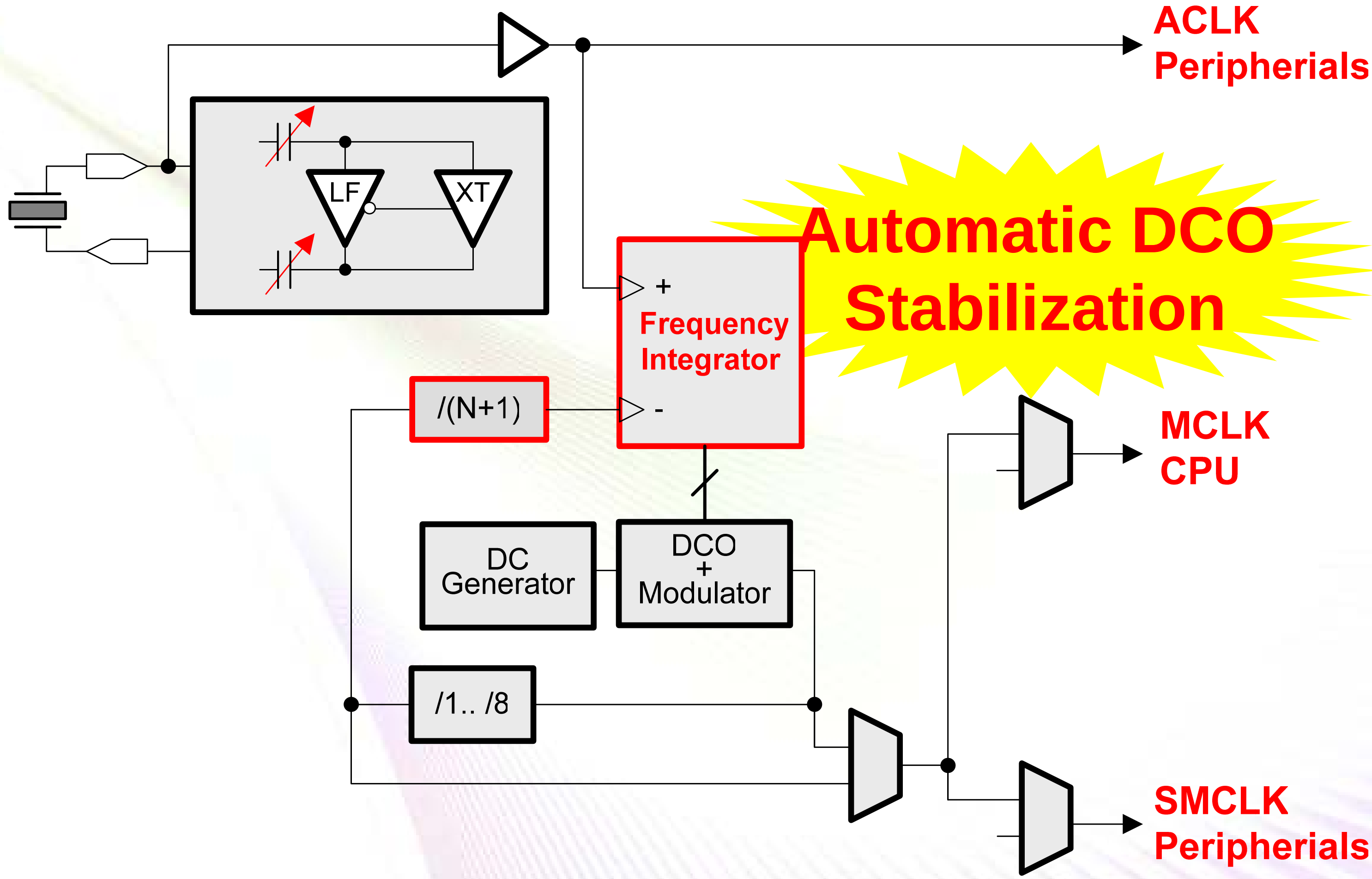
```
// Partial SW FLL Code
 if (488 < Compare )    // DCO too fast
   DCOCTL--;
else  DCOCTL++;            // DCO too slow
```



- Periodic loop can adjust DCO

If Rosc = 100k then DCOCLK ~ 2MHz

‘4xx FLL



- OSCfault fail-safe for LFXT1, DCO and XT2

Low Power Mode Configuration

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

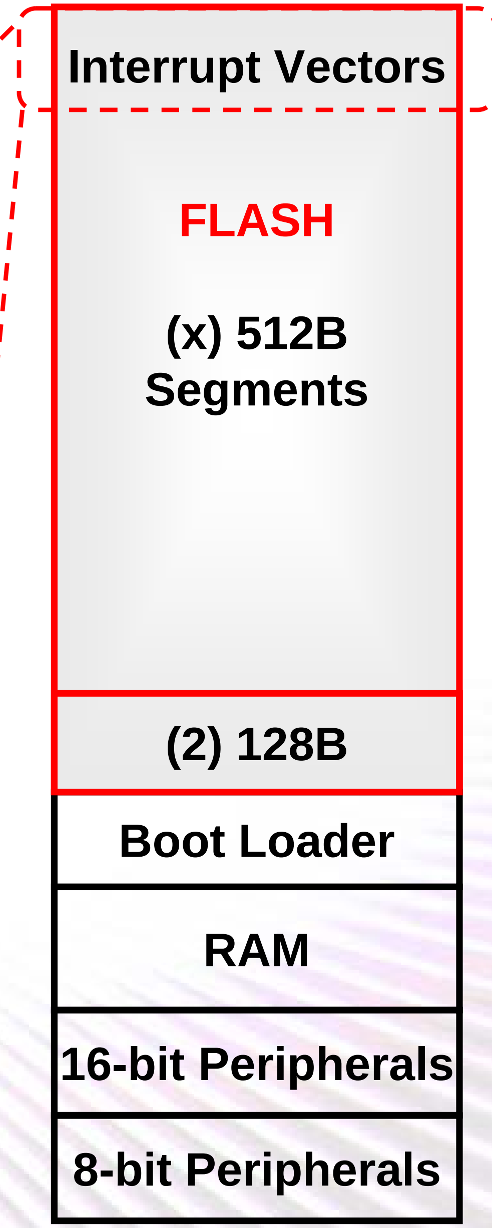
R2/SR

	SCG1	SCG0	OSC OFF	CPU OFF	
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

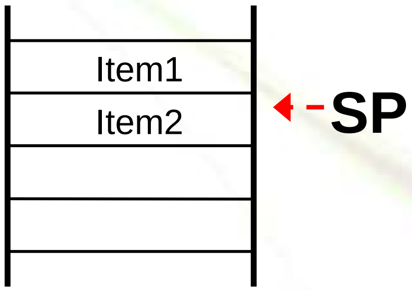
```
bis.w   #CPUOFF,SR          ; LPM0
```

Interrupt Vectors – ‘F11x1

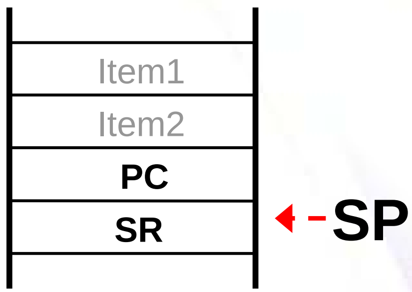
SOURCE	FLAG	INTERRUPT	ADDRESS	PRIORITY
Power-up ext. Reset Watchdog	WDTIFG	Reset	0FFFEh	15, highest
NMI Osc. Fault Flash violation	NMIIFG OFIFG ACCVIFG	(non)-maskable (non)-maskable (non)-maskable	0FFFCh	14
			0FFFAh	13
			0FFF8h	12
Comparator_A	CAIFG	maskable	0FFF6h	11
Watchdog timer	WDTIFG	maskable	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



Interrupt Processing

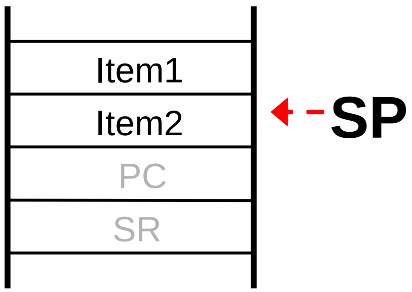


Prior to ISR



ISR hardware - automatically

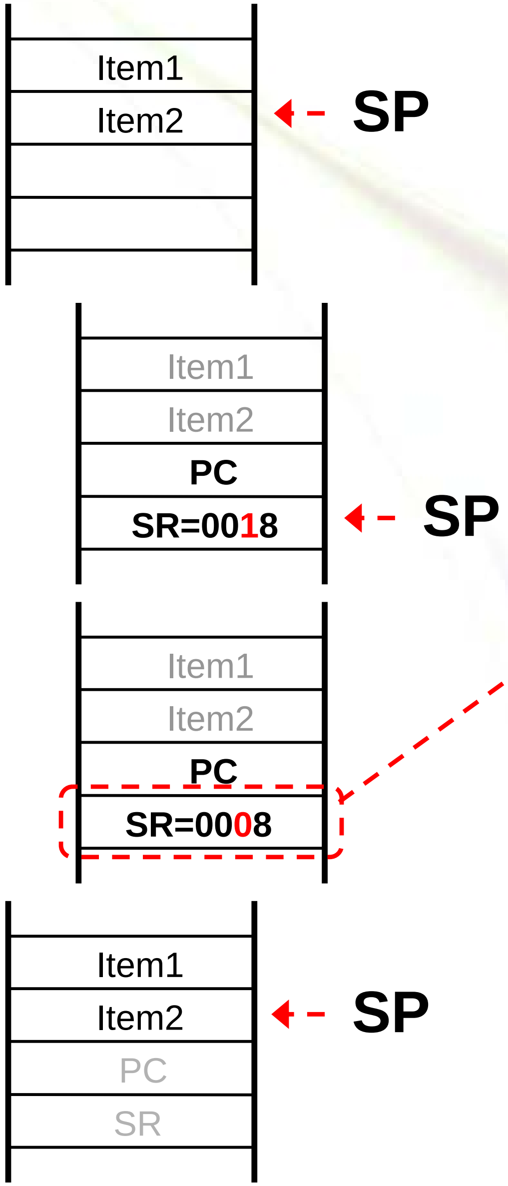
- **PC** pushed
- **SR** pushed
- Interrupt vector moved to PC
- **GIE, CPUOFF, OSCOFF and SCG1 cleared**
- IFG flag cleared on single source flags



`reti` - automatically

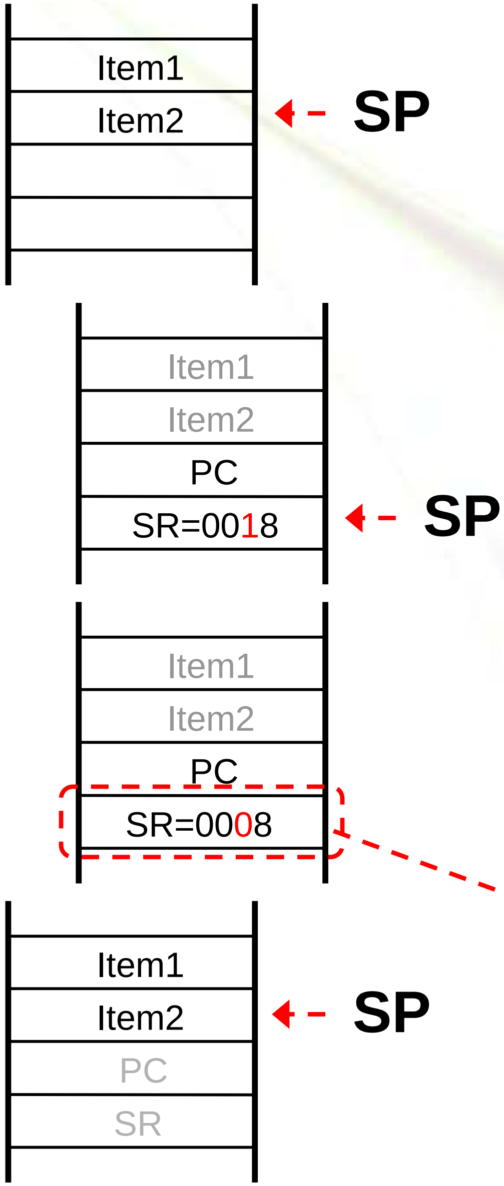
- SR popped - *original*
- PC popped

Low Power Modes In Assembler



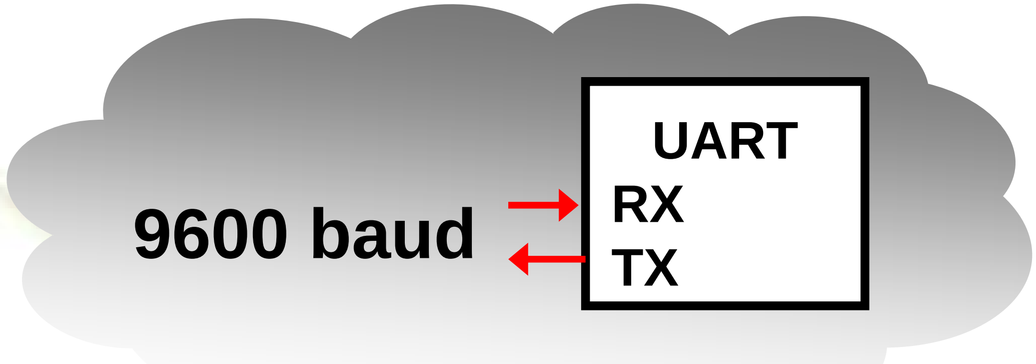

```
          ORG      0F000h
RESET     mov.w    #300h,SP
          mov.w    #WDT_MDLY_32,&WDTCTL
          bis.b    #WDTIE,&IE1
          bis.b    #01h,&P1DIR

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
```

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);
}
```

Interrupts Control Program Flow



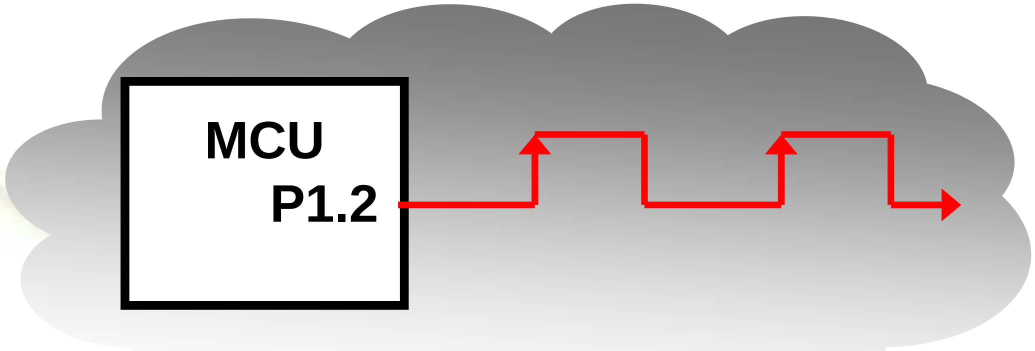

```
// Polling UART Receive
for (;;)
  {
 while (!(IFG2&URXIFG0));
 TXBUF0 = RXBUF0;
  }
```

100% CPU Load

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

0.1% CPU Load

Software Functions >> Peripherals



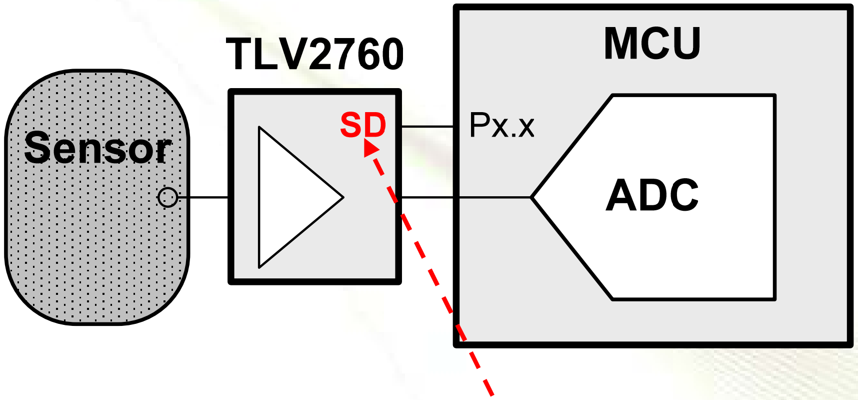

```
// Endless Loop
for (;;)
{
 P1OUT |= 0x04;  // Set
 delay1();
 P1OUT &= ~0x04; // Reset
 delay2();
}
```

100% CPU Load

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

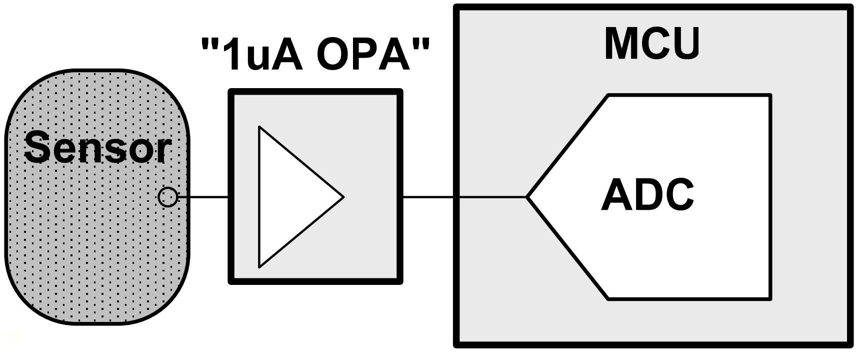
Zero CPU Load

Power Manage External Devices



0.01uA = Shutdown
20uA = Active

0.06uA = Average

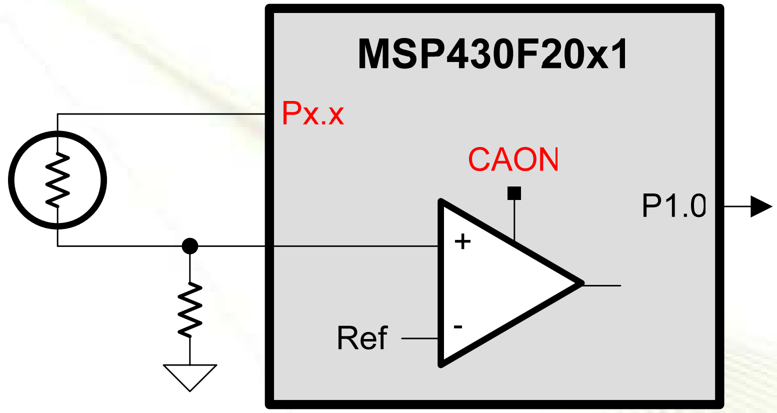


1uA = Quiescent
1uA = Active

1uA = Average

OPA with shutdown can be 20x lower total power

Power Manage Internal Peripherals



Comparator_A

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

```
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
```

How To Terminate Unused Pins?

- **Unused port pins Px.0 – Px.7?**
  - Set as output direction avoids floating gate current.

- **XT2IN, XT2OUT?**

- **Please see last page of chapter 2 in user's guide.**

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
- Every unnecessary instruction executed is a portion of the battery wasted that will never return.

Getting Started Lab3: Low-Power

- Lab2 has been converted to use LPM3 instead of the while(1) wait loop
- Open IAR and create a new project as before
- Add the file “Getting_Started_Lab3.c” to the project
- Download the code as before
- Disconnect the JTAG interface
- Measure the current through the PWR1 jumper

Getting Started Lab3 – Solution

```
 while(1)
  {
    _BIS_SR(LPM3_bits);       // Enter LPM3
    if ((P1IN & 0x01) == 0)
      P2OUT ^= 0x02;          // Toggle P2.1 using exclusive-OR
  }

// 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
  _BIC_SR_IRQ(LPM3_bits);   // Clear LPM3 bits from 0(SR)
}
```

Lab3: Low-Power Questions

- What is the current through the PWR1 jumper?
- 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?

Lab3: Low-Power Answers

- **What is the current through the PWR1 jumper?**
  - About 1.4 uA
- **Why were the I/Os configured as they were?**
  - Unused I/O must be configured as outputs, otherwise, floating gate current will be seen. 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 debounce.

Lab3: Going Further

- **Convert the code to use LPM4 and re-measure the current. What is it now?**
  - Should be about .3 uA

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
Low Power Wireless	www.ti.com/lpw	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265