

采用 MSP430 LaunchPad 启动开发工作



议程

- 介绍 **Value Line** 系列

- **Code Composer Studio**
- **CPU** 与基本时钟模块
- 中断与 **GPIO**
- **Timer_A** 与 **WDT+**
- **MSP430**低功耗设计
- **ADC10** 和 **Comparator_A+**
- 串行通信模块
- **Grace**
- 电容式触摸按键解决方案

MSP430 系列MCU产品

MSP430 | Ultra-Low Power is in our DNA

MSP430 Portfolio at a glance

300+ Ultra-Low Power Devices Starting @ \$0.25USD

Featuring: Up to 256kB Flash, 18kB RAM, 25+ Package Options, Up to 113 pins, High integration

Ultra-Low Power Performance		Analog Integration		Easy-to-Use		
MSP430 16-bit RISC CPU All devices feature: <ul style="list-style-type: none"> • 16-bit timers • Watchdog Timer • Internal Digitally Controlled Oscillator • External 32-kHz crystal support • <50 nA pin leakage • <6 μs wakeup 	L092 0.9V-1.65V Speed 4Mhz ROM to 2kB RAM to 2kB GPIO 11	G2xx Speed 16Mhz Flash 0.5-16kB RAM to 256kB GPIO 10-16	F4xx Speed 8/16Mhz Flash 4-120kB RAM to 8k GPIO 14-80	F5xx Speed 25Mhz Flash 8-256kB 512kB coming soon. RAM to 18kB GPIO 32-83	CC430 Speed 20Mhz Flash 8-32kB RAM to 4kB GPIO 40	
	F1xx Speed 8Mhz Flash 1-60kB RAM to 10kB GPIO 14-48	F2xx Speed 16Mhz Flash 1-120kB RAM to 8kB GPIO 10-64	FRAM Speed 24Mhz FRAM 4-16kB GPIO 14-28 Non-volatile memory			
		BOR DACS Comp SVS BOR WDT A-POOL ADCS	BOR ADC10 Comp_A+ Temp USI Cap Sense I/Os BOR ADC10,12 SD16_A Comp_A+ DAC12 DMA MPY OpAmp SVS USCI USI	BOR LCD ADC10,12 SD16_(A) Comp_A DAC12 DMA MPY OpAmp SVS USART USCI ESP430 SCAN_IF Basic Timer WDT+ RTC_C	BOR SVS SVM LDO MPY USCI DMA EDI USB ADC10,12 (A) Comp_B RTC_A/B WDT Cap Sense I/Os	BOR SVS SVM LDO MPY USCI DMA SUB1GHz RE AES ADC12 (A) Comp_B RTC_A/B LCD

All Devices Some Devices

MSP430 MCU 介绍



超低功耗

业内功耗最低的 MCU

- 超低功耗运行模式
- 7 种低功耗模式
- 即时唤醒
- 所有的 MSP430 器件均具有超低功耗特性

集成

智能型模拟与数字外设

- 外设工作于低功耗模式
- 减少外部分立器件与物料成本
- 具有 FRAM、USB、RF、电容式触摸 I/O、计量模块、LCD、ADC、DAC 等等

丰富齐全的产品线，低成本可供选择

找到适合您需要的理想 MCU

- 400 多款器件
- 容量高达 256kB 闪存，18kB RAM，超过 25 种封装可供选择
- Value Line 系列器件起售价仅 0.25 美元
- 不同性能与片上集成度的器件可供选择

易于启动开发工作

低成本与简单入手

- 完整套件起售价仅 4.30 美元
- 可提供基于 GUI 的编码及调试工具
- MSP430Ware 软件与资源库
-包括代码范例、数据手册、用户指南等等!

MSP430 支持的应用

公用事业计量

电能表
燃气表
流量表
智能计量仪表



MSP430 MCU 可支持数以千计的应用

凭借 MSP430 MCU 的超低功耗性能、高集成度模拟与数字外设、以及易用的工具，客户可方便地实现其产品的差异化

无线应用

远程传感器
通讯控制器
RFID



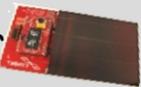
消费类电子

便携式电子产品
遥控器
个人保健
PC 外设



能量收集

可再生能源
无电池设备
太阳能、热能、
振动能，等等



便携式医疗

血糖计
温度计
心率监测计
可植入装置



传感器与安全

烟雾探测器
运动探测器
振动检测器
智能传感器



个人健康与健身

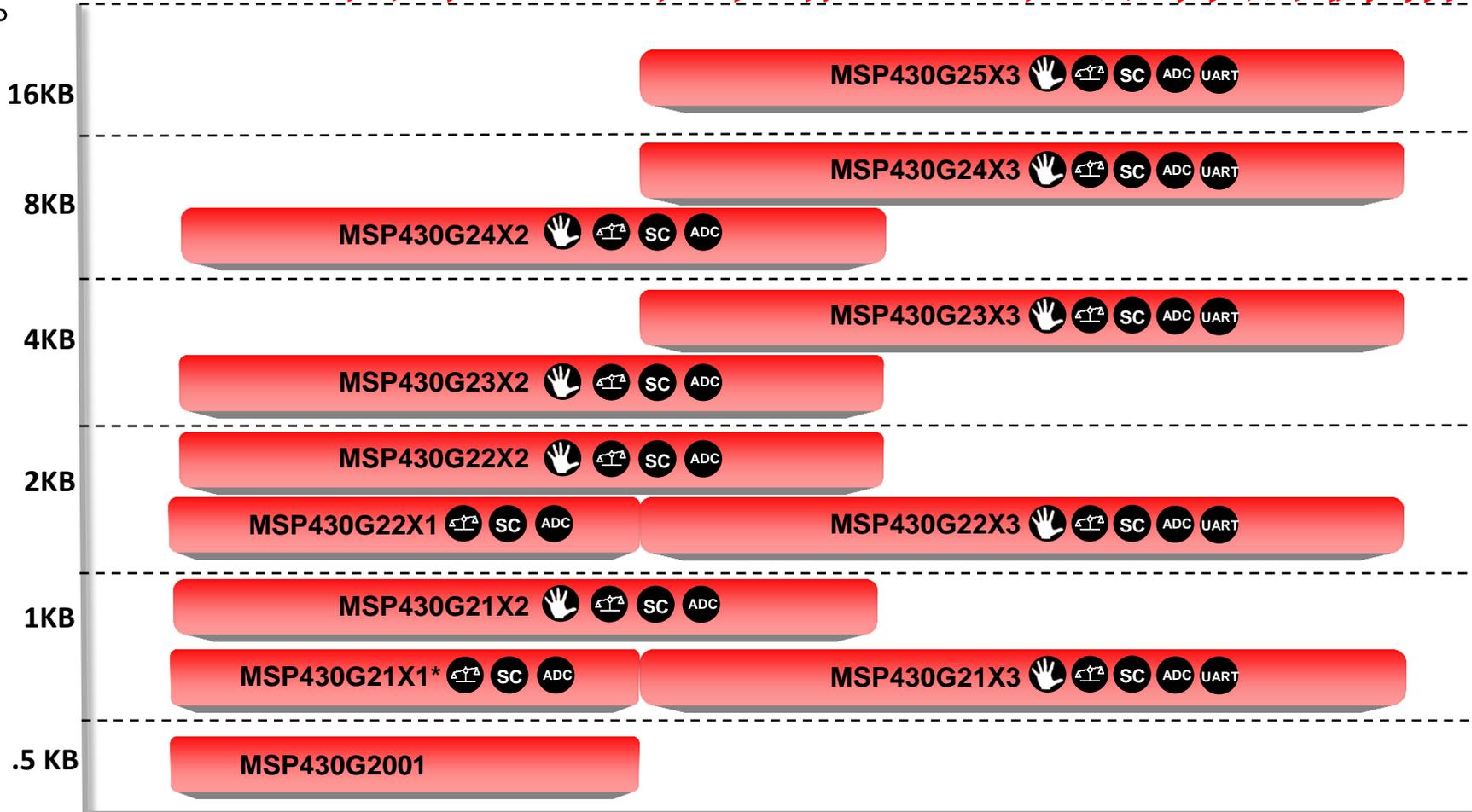
运动手表
计步器
热量计
潜水手表



Value Line 系列：16 位性能，8 位器件的价格

- UART
- 电容式触摸 I/O
- ADC
- ADC10
- 比较器
- SC
- SPI/I2C

闪存容量



14 引脚
TSSOP/PDIP 封装
10 个 GPIO

16 引脚
QFN 封装
10 个 GPIO

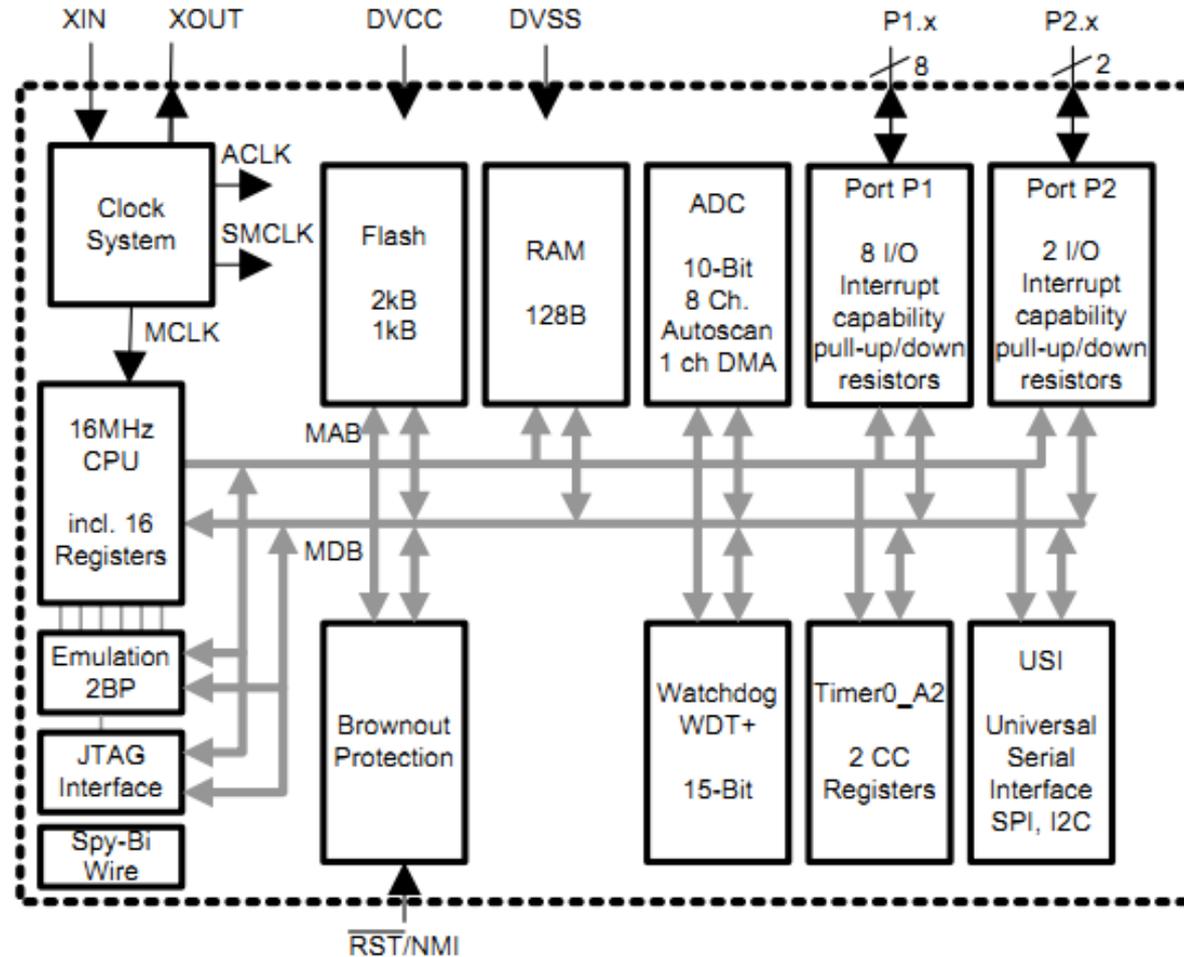
20 引脚
TSSOP/PDIP 封装
16 个 GPIO

28 引脚
TSSOP 封装
24 个 GPIO

32 引脚
QFN 封装
24 个 GPIO

Value Line 功能框图

functional block diagram, MSP430G2x31



Value Line 外设

◆ 通用 I/O

- 可独立编程
- 可提供输入、输出与中断（边沿可选）的任意组合
- 所有寻址指令可对端口控制寄存器进行读/写访问
- 每个 I/O 具有一个可独立编程的上拉/下拉电阻
- 某些器件/引脚具有触摸按键模块 (PinOsc)

◆ 16 位 Timer_A2

- 2 个捕获/比较寄存器
- 丰富的中断功能

◆ WDT+ 看门狗定时器

- 也可用作一个普通定时器

◆ 欠压复位

- 可在上电和断电期间提供正确的复位信号
- 功耗包含于MCU最低功耗时（LPM4）所消耗电流之中

Value Line 外设

◆ 串行通信

- 支持 I2C 和 SPI 的 USI
- 支持 I2C、SPI 以及 UART 的 USCI

◆ Comparator_A+

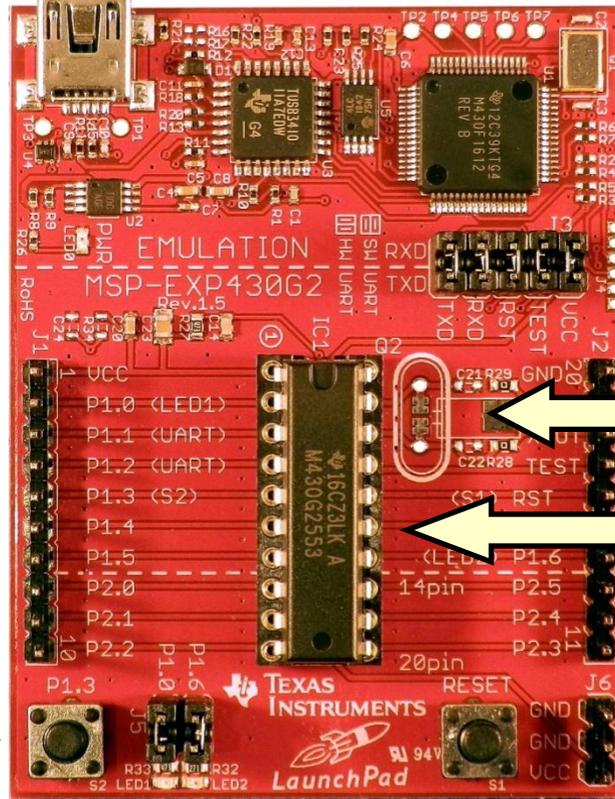
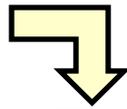
- 可设定反相和同相输入
- 可选的 RC 输出滤波器
- 可直接输出至 Timer_A2 捕获输入
- 具有中断能力

◆ 8 通道/10 位 200 ksps SAR ADC

- 8 个外部通道（取决于器件）
- 内置电压和温度传感器
- 可编程的参考电压
- DTC可在无需 CPU 干预的情况下将结果发送至存储器
- 具有中断能力

LaunchPad 开发板

USB 仿真器接口



片上仿真器模块

6 针 eZ430 连接器

外置晶体接口

MSP430 器件和插座

电源连接器

芯片引出脚

P1.3 按钮

LED 和跳线

P1.0 和 P1.6

复位按钮

议程

- 介绍 Value Line 系列
- **Code Composer Studio**
- CPU 与基本时钟模块
- 中断与 GPIO
- Timer_A 与 WDT+
- MSP43P低功耗设计
- ADC10 和 Comparator_A+
- 串行通信模块
- Grace
- 电容式触摸按键解决方案

Code Composer Studio 简介

- ◆ 用于 TI 嵌入式处理器的集成型开发环境 (IDE)
 - 包括调试器、编译器、编辑器、仿真器、操作系统 (OS) ...
 - 该 IDE 基于 **Eclipse** 开源软件框架
 - 由 TI 对其进行扩展以支持 TI 嵌入式控制器
- ◆ **CCSv5** 基于成熟的 **Eclipse** (在 **CCS 5.1** 中采用的是 **version 3.7**)
 - 未来的 **CCS** 版本将使用 **Eclipse** 的最新版本
 - 由其他供应商提供的全兼容型 **Eclipse** 插件或使用 TI 工具并将它们置于一种现有的 **Eclipse** 环境之中
 - 用户能充分利用 **Eclipse** 的最新特性
- ◆ 集成更多的工具
 - **OS** 应用程序开发工具 (**Linux**、**Android**...)
 - 代码分析、源控制...
- ◆ 很快支持 **Linux**
- ◆ 低成本! **445** 或 **495** 美元

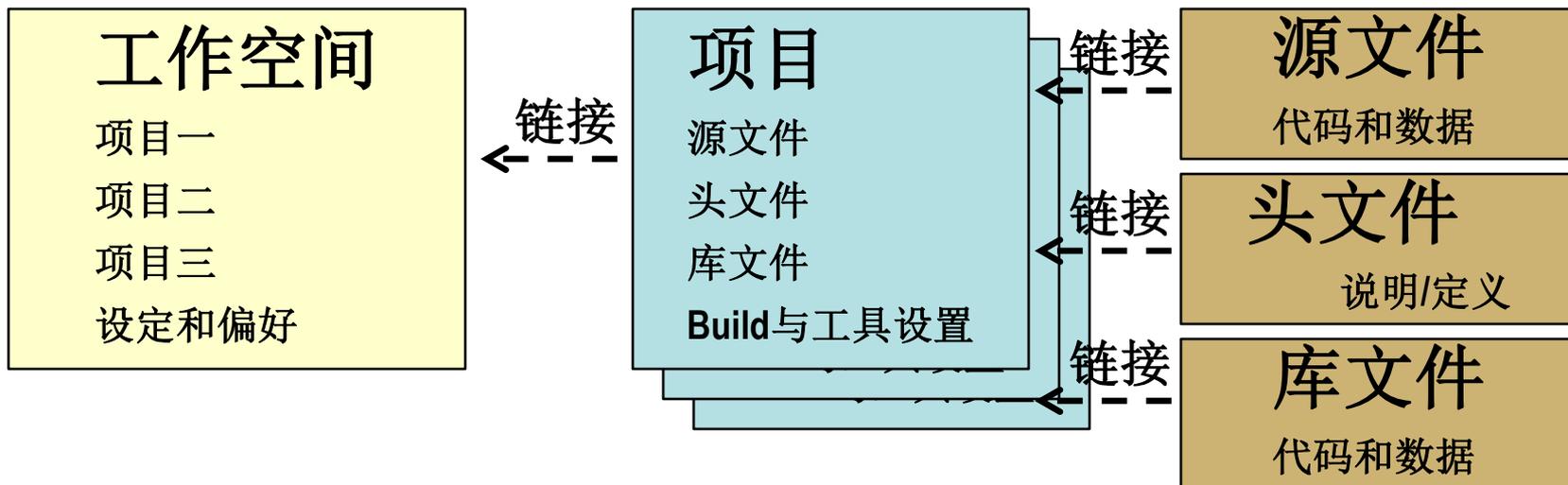


常见任务

- ◆ 创建新项目
 - 采用模板即可非常简单地创建针对某款器件的新项目
- ◆ **Build**选项
 - 用户可使用**Build**选项对项目进行编译配置
 - 选项的更新通过编译器的发布来提供，而不依赖于 **CCS** 的更新升级
- ◆ 共享项目
 - 用户可非常方便共享项目，并包括项目的版本控制等
 - 简化操作以共享链接资源



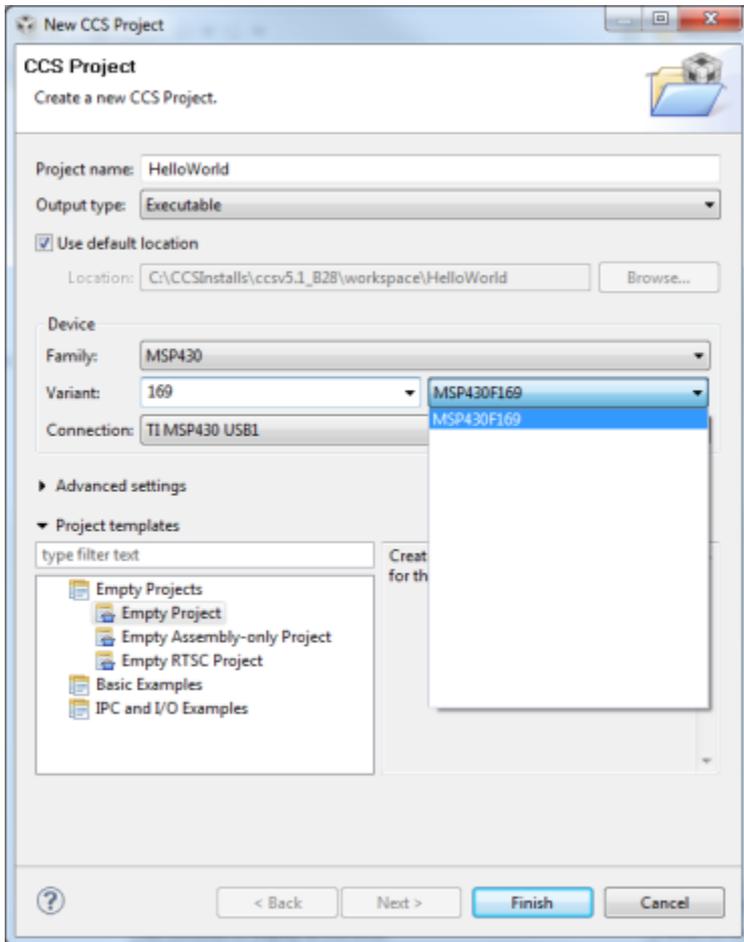
工作空间与方案



工作空间包含您的设置与偏好，以及至您的项目的链接。
将项目从工作空间中删除只是删除了链接，而非设计文件

一个工程项目包含了您的Build选项与工具设置，以及至您的输入文件的链接。
将文件从工作空间中删除只是删除了链接，而非设计文件

项目向导



- ◆ 单页向导满足大多数情况的使用要求
 - “下一个”按钮将在某个模板需要附加设置显示
- ◆ 包含调试器设置
 - 假如选取了某款特定的器件，用户还可选择其调试器，并生成 ccxml 文件
- ◆ 采用默认设置使其简单易用
 - 一些高级配置编译器版本、字节存储顺序等在 **Advanced Settings** 下



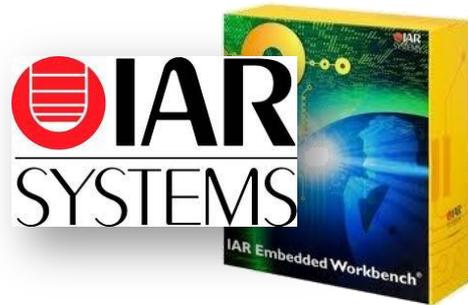
各种不同的 IDE 选项

可提供免费的集成型开发环境 (IDE)



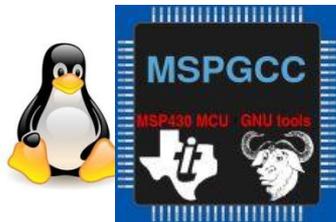
Code Composer Studio

- 基于 Eclipse 的 IDE（编译器、调试器、链接器等），适用于所有的 TI 嵌入式处理器
- 无限制版本售价 495 美元
- 可提供免费版本！
 - 免费的 16kB 代码空间限制版本可供下载
 - 可提供免费、全功能、120 天试用期限版本



IAR Embedded Workbench

- 功能强大的第三方 IDE 产品，配有项目管理工具和编辑器。包括用于所有 MSP430 器件的配置文件。
- 可提供免费版本！
 - 免费的 4/8/16kB 代码空间限制 (Kickstart) 版本可供下载
 - 可提供免费、全功能、30 天试用期限版本



MSPGCC

- 用于 MSP430 的免费、开源、GCC 工具链
- 包含 GNU C 编译器 (GCC)、汇编器和链接器 (binutils)、调试器 (GDB)
- 工具可在 Windows、Linux、BSD 及其他大多数 Unix 版本的操作系统上使用
- 更多详情敬请访问：<http://mspgcc.sourceforge.net/>

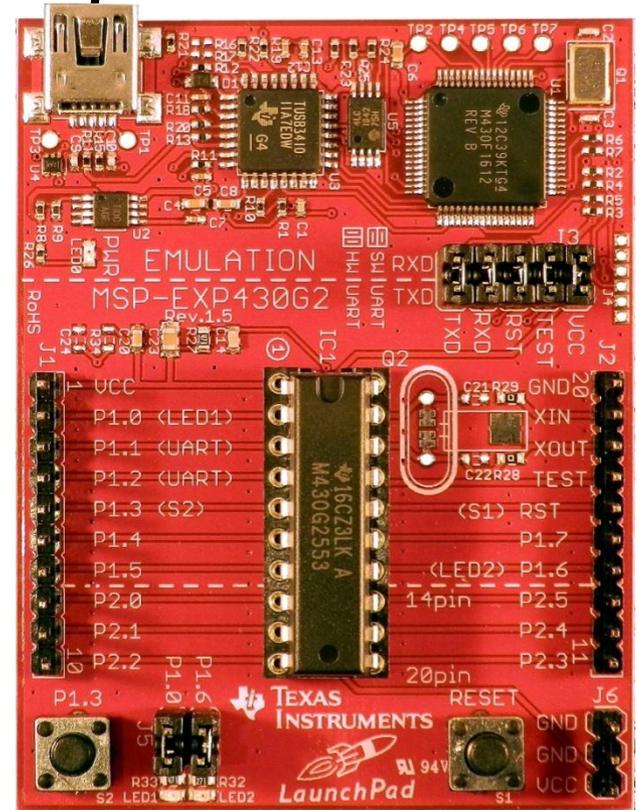
可提供其他的 MSP430 IDE 选项！更多详情敬请访问 www.ti.com/msp430tools

Lab1: Code Composer Studio



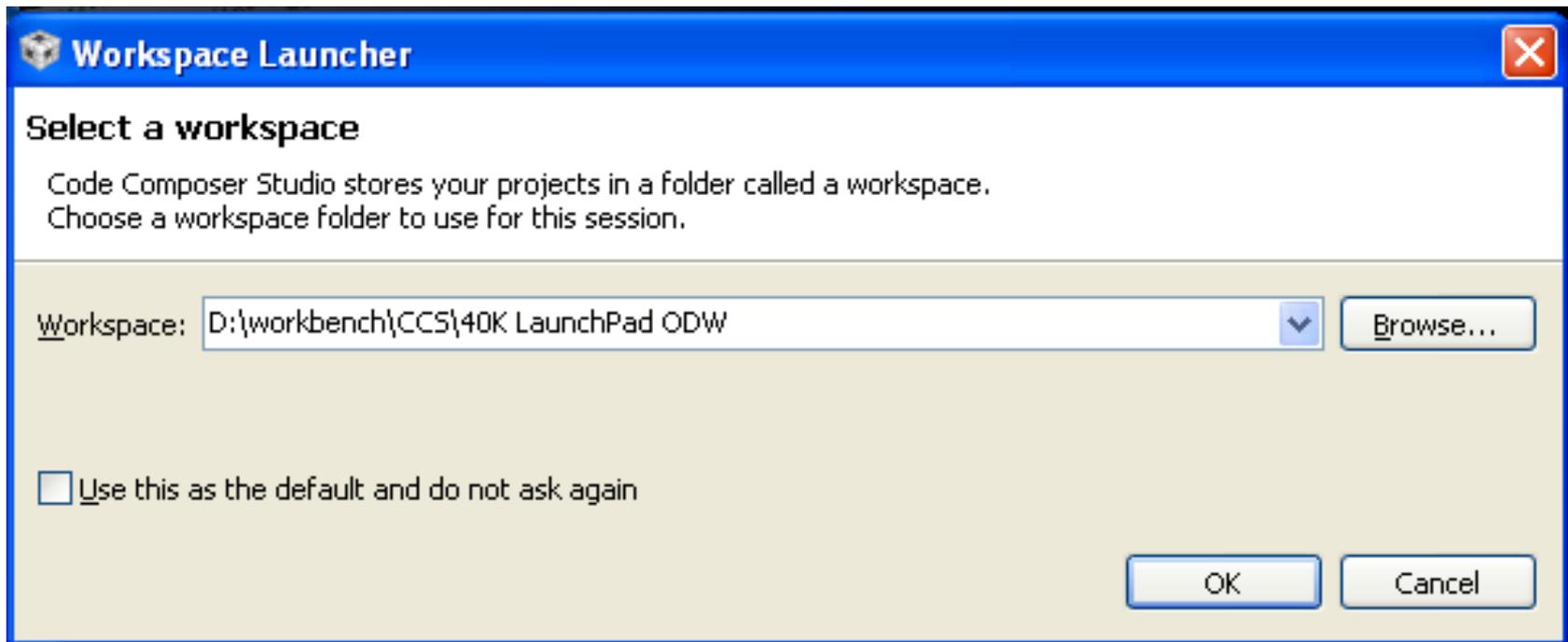
• Lab1:

- 创建一个新的工作空间
- 创建 **Lab1** 项目
- 加进 **temperature sense demo** 代码
- 编译并运行



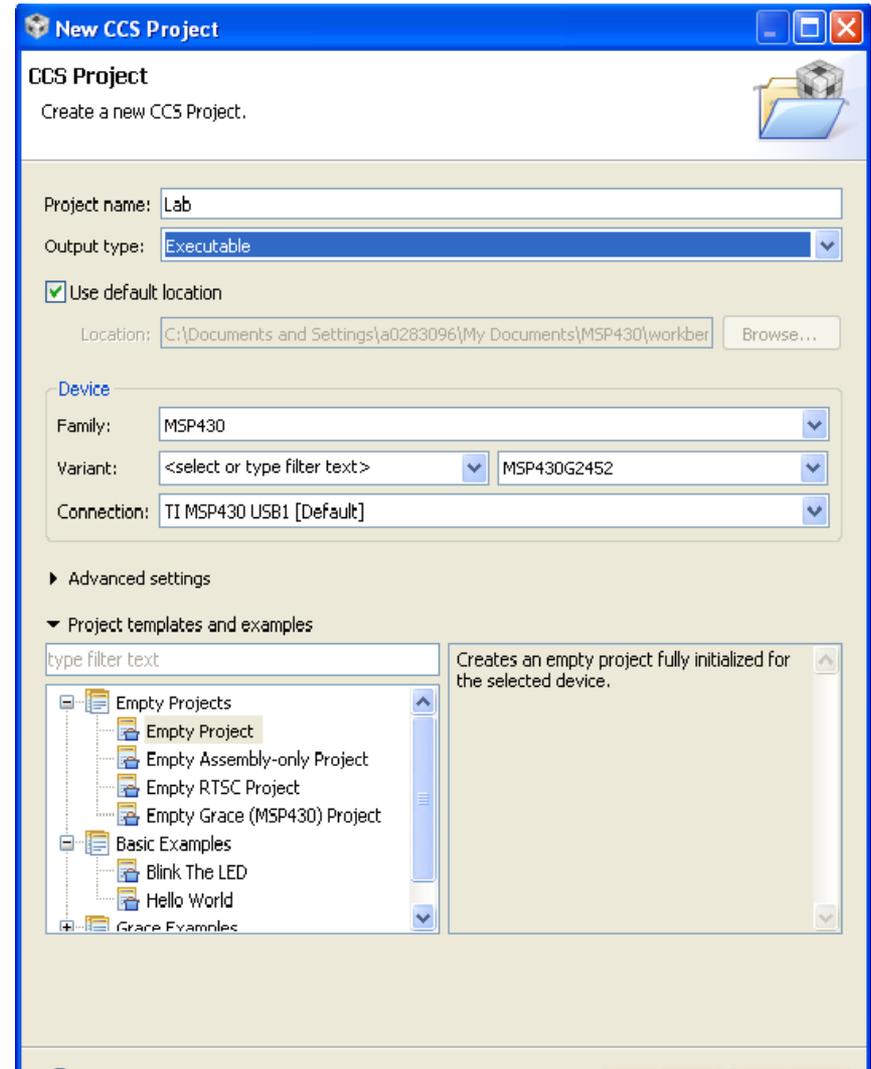
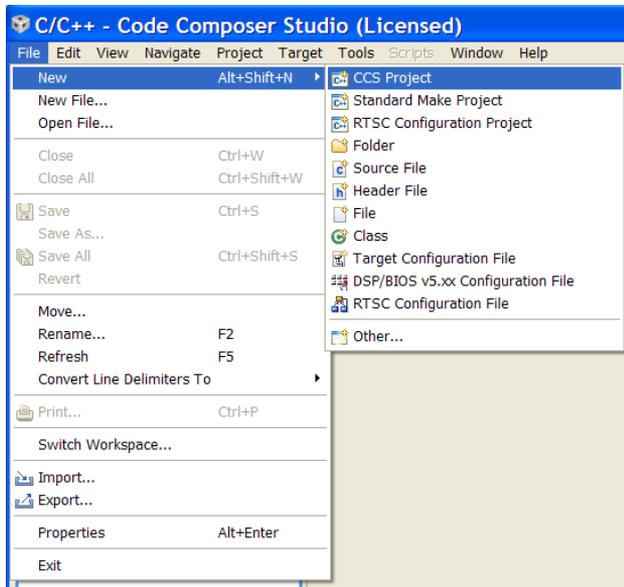
第一步：创建 CCS 工作空间

- 将 Lab 文件置于您的电脑上
- 启动 CCS v5
- 选择一个“工作空间”位置



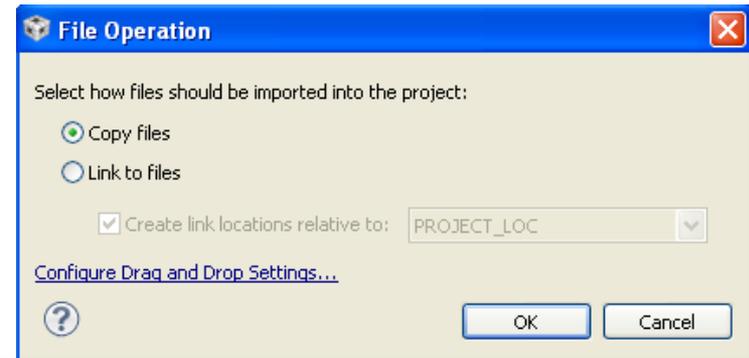
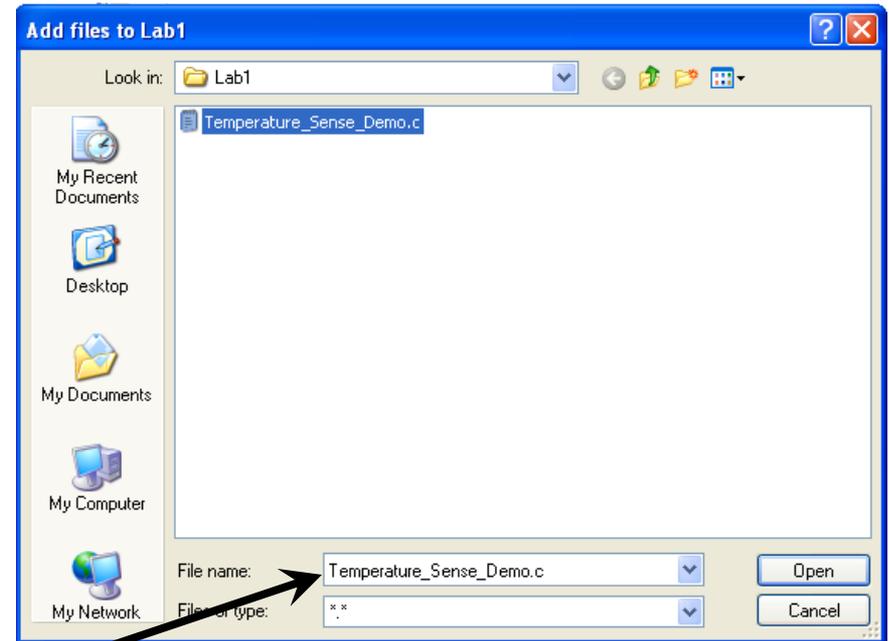
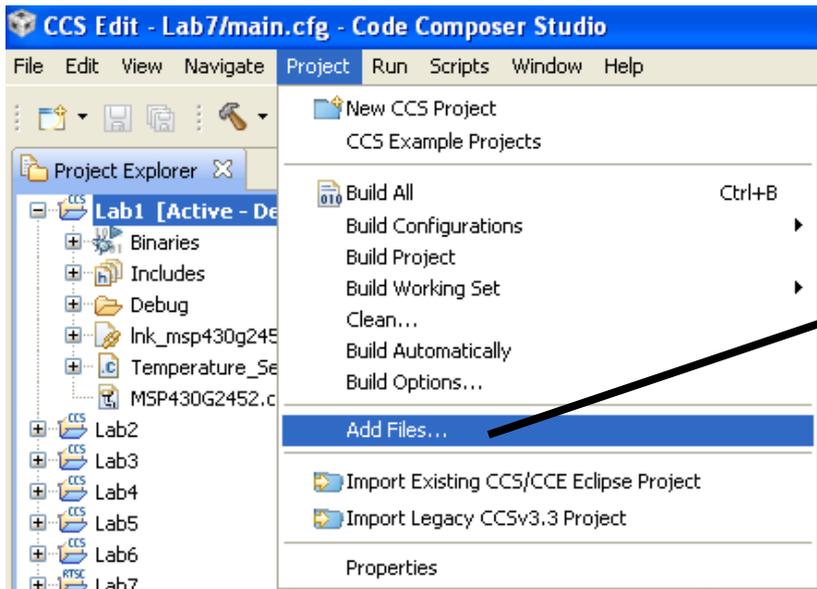
第二步：创建一个 CCS 项目

- File > New > CCS Project
- Project name: Lab1
- Device>Family: MSP430
- Variant: MSP430G2231
- Project templates and examples:
Empty Project



第三步：在 CCS 项目中加进一个文件

- Project > Add Files
- 选择 Lab 所在的文件夹
- 并选择：
Temperature_Sense_Demo.c



CCS 窗口 - C/C++ 透视概览

一键式项目调试

独立的调试与 C/C++ 项目透视图

项目视图

- 所有项目的列表

项目概要

- 快捷了解项目组成部分

控制台

- 构建信息

代码窗口

- 实时断点, 语法高亮显示

问题观察

- 信息、警告、差错

```
64  IntDegC = ((temp - 1855) * 667) / 4096;
65  format_temperature_string(IntDegC, 'C');
66
67
68  // Temperature in Fahrenheit
69  // If = (9/5)*Tc + 32
70  IntDegF = ((temp - 1855) * 1199) / 4096 + 32;
71  format_temperature_string(IntDegF, 'F');
72
73  // SET BREAKPOINT HERE
74  _no_operation();
75
76  /* Initialize serial communication module to send data over USB */
77  halUsbInit();
78  halUsbSendString(&USB_string[0], USB_STRING_LEN+2);
79  halUsbShutDown(); // Shut down USB to enter sleep
80
81  __delay_cycles(2200000); // Delay 2 seconds
82
83 }
84
85 unsigned char format_temperature_string(unsigned int degrees, unsigned char status)
86 {
87     unsigned char hundreds = '0'; // Init to '0' to code in ASCII
88     unsigned char tens = '0';
89     unsigned char ones = '0';
90     unsigned char status = STATUS_PASS;
91
92     while(degrees >= 100) { // Calculate hundreds' place
93         hundreds++;
94         degrees -= 100;
95     }
96 }
```

CCS 窗口 - 调试透视概览

一键式项目调试

独立的调试与 C/C++ 项目透视图

高度可配置的窗口布局

- 用户偏好
- 插件支持

目标控制

- 起动
- 停止
- 暂停
- 步进
- 堆栈跟踪

实时、系统内 MSP430 信息

- 寄存器访问
- 闪存、RAM、信息段访问
- 反汇编视图

程序长度信息

代码窗口

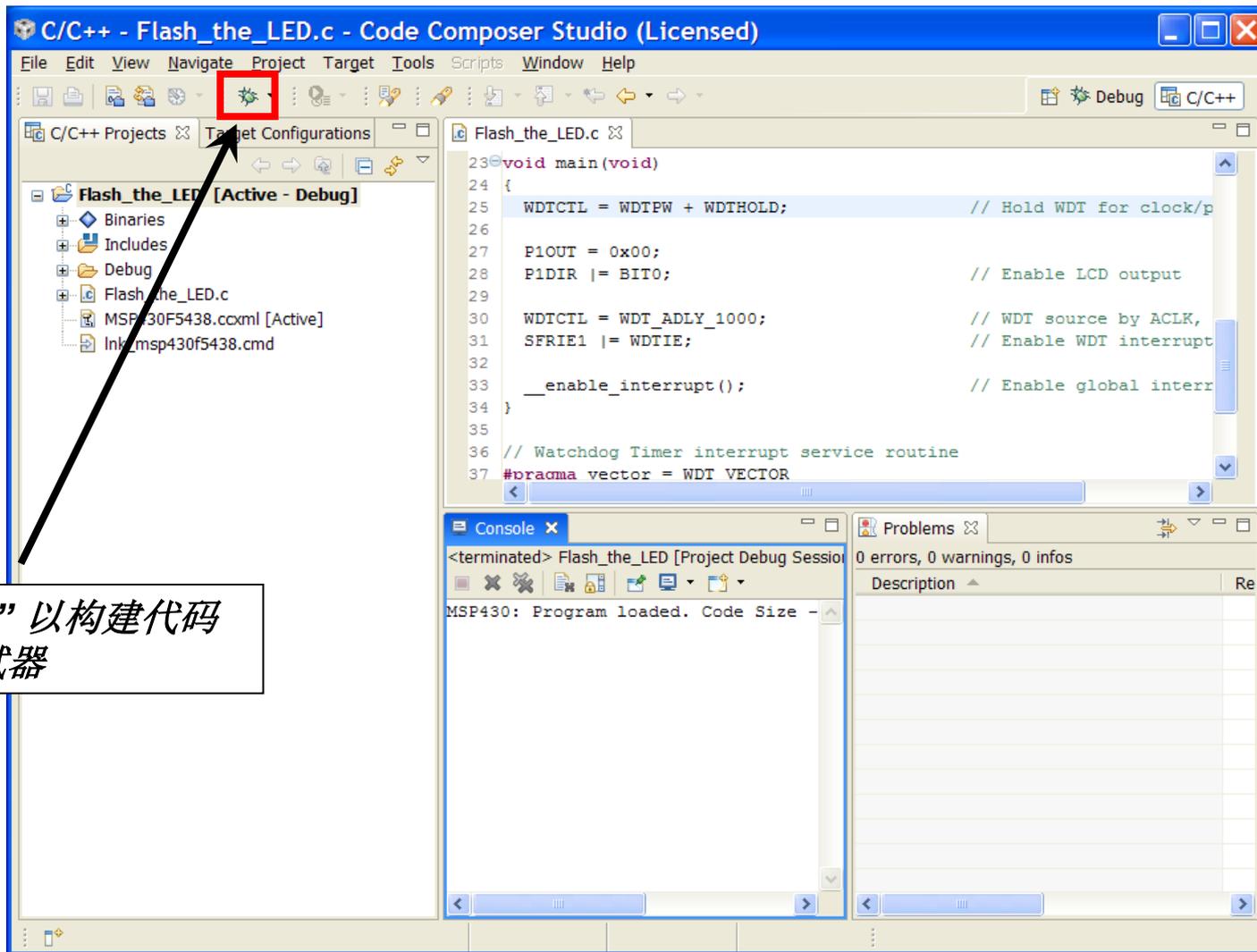
- 实时断点, 语法高亮显示

Name	Value	Address	Type
IntDegC	364	0x5BF8	unsigned int
IntDegF	687	0x5BFA	unsigned int

Register	Value	Register	Value
PC	0x005D06	SP	0x005BF8
SR	0x0000	R3	0x000000
R4	0x002004	R5	0x000007
R6	0x0051B1	R7	0x07E881
R8	0x0E3722	R9	0x0CF5E1
R10	0x000000	R11	0x0BFAF8
R12	0x000001	R13	0x000046
R14	0x0004AF	R15	0x000000

```
64 IntDegC = ((temp - 1855) * 667) / 4096;
65 format_temperature_string(IntDegC, 'C');
66
67 // Temperature in Fahrenheit
68 // Tf = (9/5)*Tc + 32
69 IntDegF = ((temp - 1855) * 1199) / 4096 + 32;
70 format_temperature_string(IntDegF, 'F');
71 _no_operation(); // SET BREAKPOINT HERE
72
73 /* Initialize serial communication module to send data over USB */
74 halUsbInit();
75 halUsbSendString(&USB_string[0], USB_STRING_LEN+2);
76 halUsbShutDown(); // Shut down USB to enter sleep mode
77
78 __delay_cycles(2200000); // Delay 2 seconds
79 }
80 }
81
82 unsigned char format_temperature_string(unsigned int degrees, unsigned char c_or_f)
83 {
84     unsigned char hundreds = '0'; // Init to '0' to code in ASCII
85     unsigned char tens = '0';
86     unsigned char ones = '0';
```

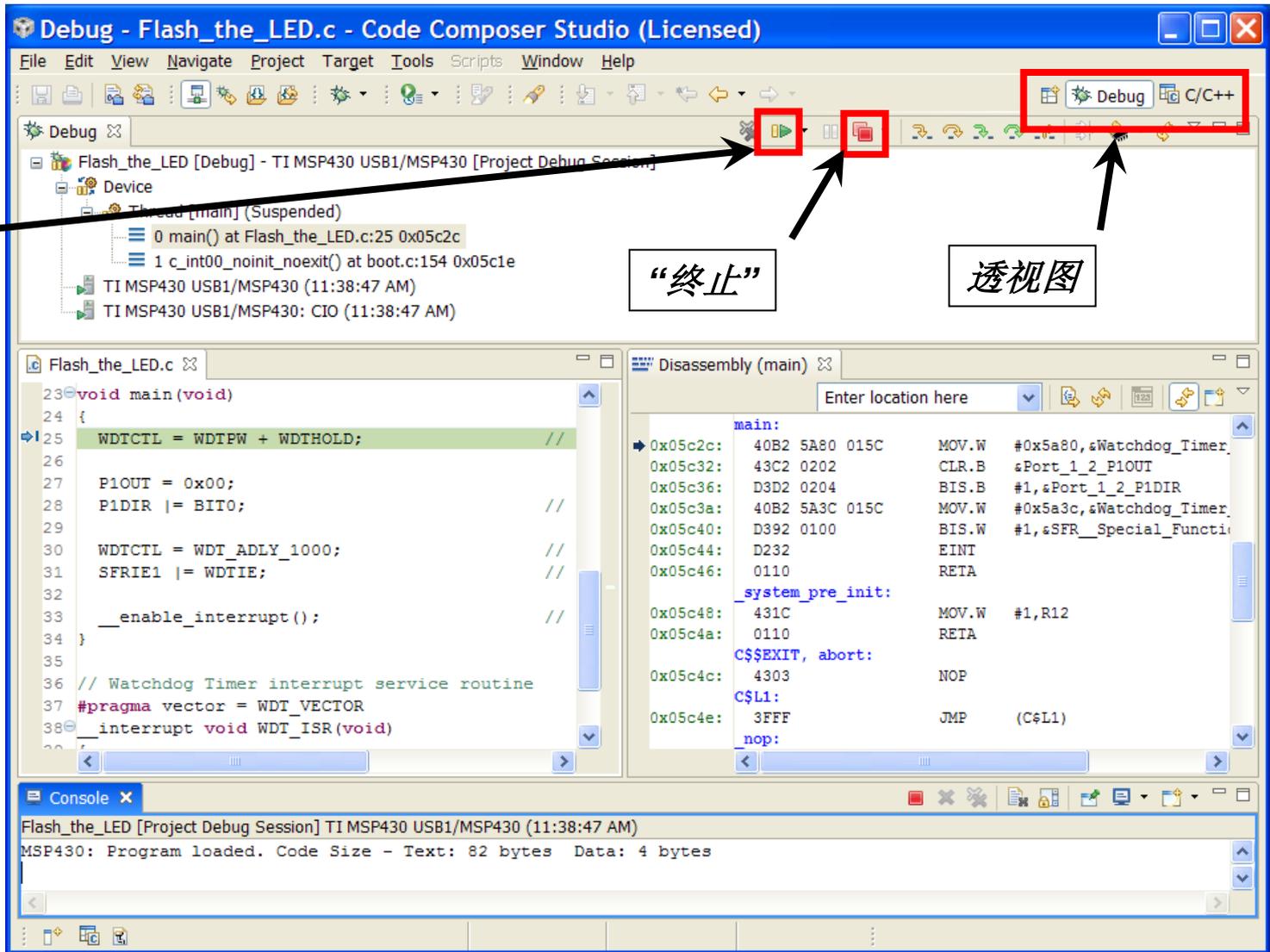
第四步：Build及调试一个 CCS 项目



点击“DEBUG”以构建代码
并启动调试器

第五步：运行、终止一个 CCS 项目

“运行”



“终止”

透视图

议程

- 介绍 Value Line 系列
- Code Composer Studio
- **CPU 与基本时钟模块**
- 中断与 GPIO
- Timer_A 与 WDT+
- MSP430低功耗设计
- ADC10 和 Comparator_A+
- 串行通信模块
- Grace
- 电容式触摸按键解决方案

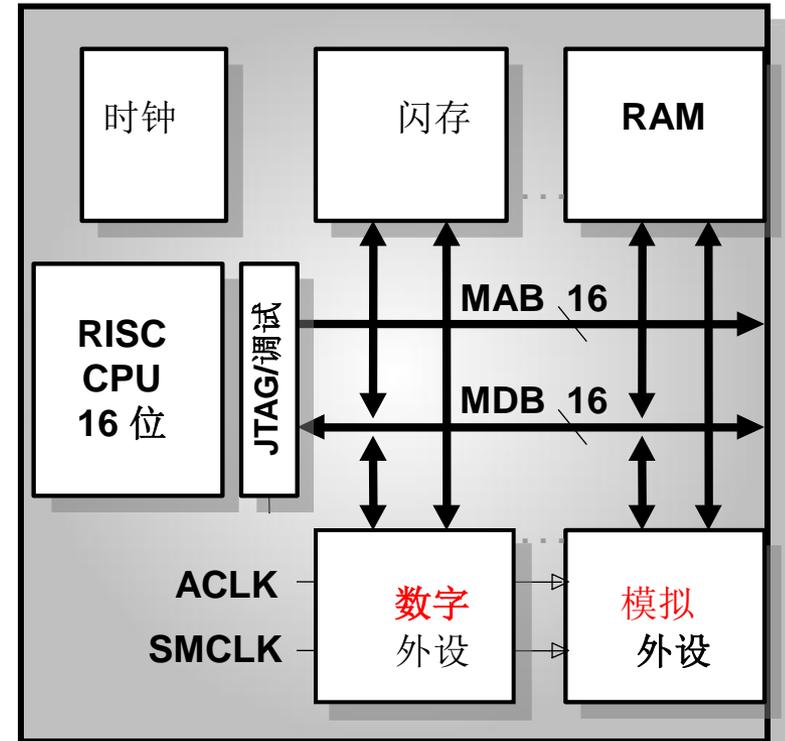
MSP430G2xx 结构

超低功耗

- ◆ 0.1uA 断电模式流耗
- ◆ 0.8uA 待机模式流耗
- ◆ 220uA / 1MIPS
- ◆ <1us 时钟启动时间
- ◆ <50nA 端口漏电流
- ◆ 零功耗欠压复位 (BOR)

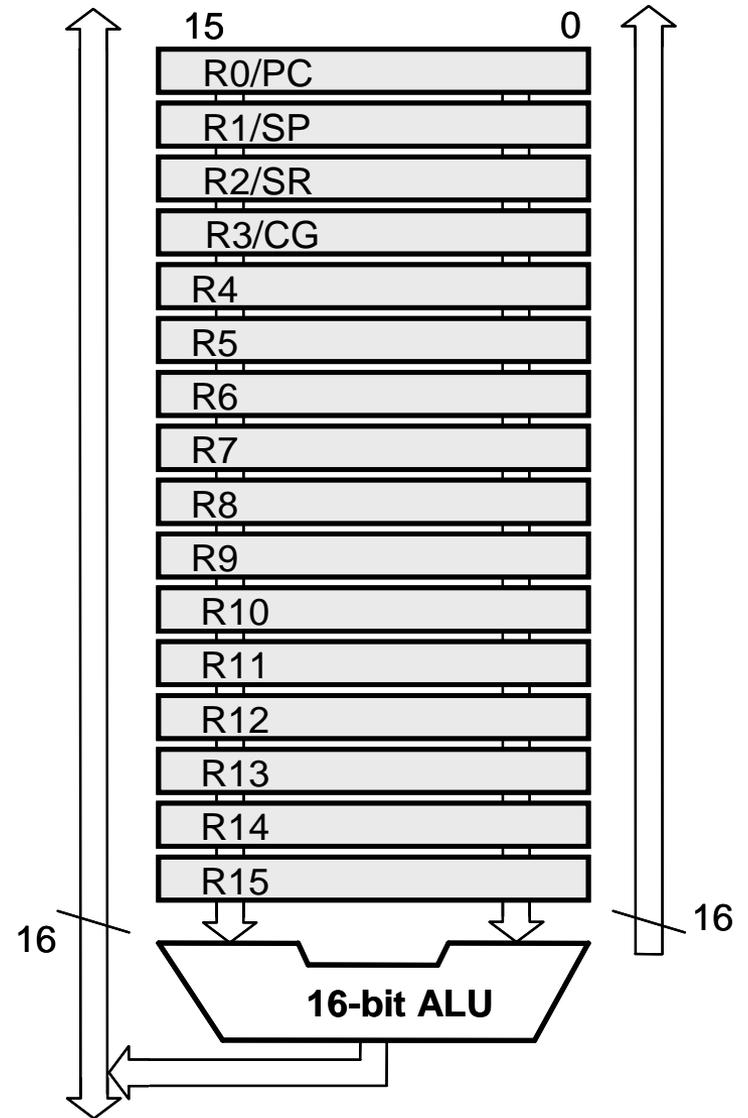
超灵活

- ◆ 0.5k 至 16kB 系统内可编程 (ISP) 闪存
- ◆ 16 位定时器
- ◆ SPI、I2C
- ◆ 10 位 ADC
- ◆ 嵌入式仿真



16 位 RISC CPU

- ◆ **单周期寻址寄存器文件**
 - ◆ 4 种专用型
 - ◆ 12 种通用型
 - ◆ 无累加器瓶颈
- ◆ **RISC 架构**
 - ◆ 27 条核心指令
 - ◆ 24 条仿真指令
 - ◆ 7 种地址模式
- ◆ Atomic内存至内存寻址
- ◆ 位、字节和字处理
- ◆ 常数发生器



存储器映射

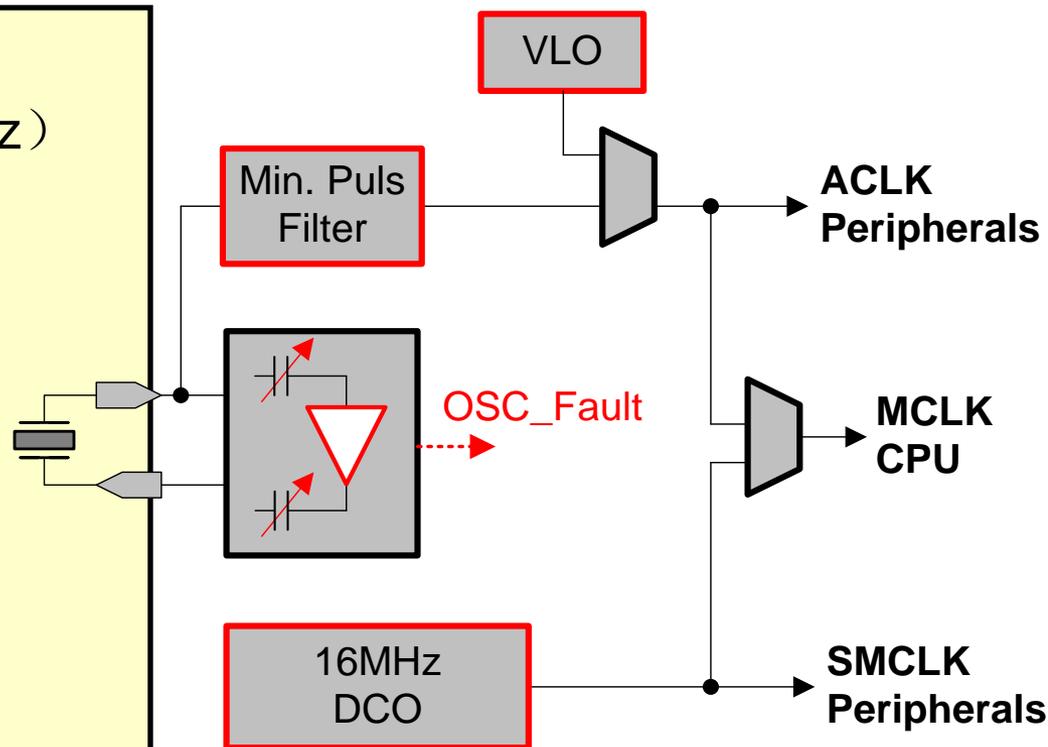
示出的是 G2231

- ◆ 闪存可编程：通过 **JTAG** 或系统内 (**ISP**)
- ◆ 编程电压低至 **2.2V**，单字节或单字编程
- ◆ 主存储器：每段**512** 字节 (**0-n**). 可分段或全部擦除
- ◆ 信息存储器：每段**64** 字节 (**A-D**)
 - **Section A** 包含器件专用的校准数据，并可锁定
- ◆ 可编程闪存定时发生器

0FFFh	中断矢量表
0FFC0h	
FFDFh	闪存/ROM
F800h	
	
010FFh	信息存储器
01000h	
	
027Fh	RAM
0200h	
01FFh	16 位外设
0100h	
0FFh	8 位外设
010h	
0Fh	8 位特殊功能寄存器
0h	

时钟系统

- ◆ **超低功耗/低频振荡器 (VLO)**
 - ◆ 4 – 20kHz (典型值为 12kHz)
 - ◆ 500nA 待机流耗
 - ◆ 0.5%/° C 和 4%/V 漂移
- ◆ **晶体振荡器 (LFXT1)**
 - ◆ 片内集成可编程负载电容
 - ◆ 故障保险 OSC_Fault
 - ◆ 脉冲滤波器
- ◆ **数字控制振荡器 (DCO)**
 - ◆ 0 至 16MHz
 - ◆ ± 3% 容差
 - ◆ 出厂校准 (保存在闪存中)



MCU PUC后, MCLK 和 SMCLK 由 DCOCLK 提供 (约 1.1 MHz)。在 LF 模式中, ACLK 由 LFXT1CLK 采用一个 6pF 内部负载电容提供。

G2xxx –DCO校正

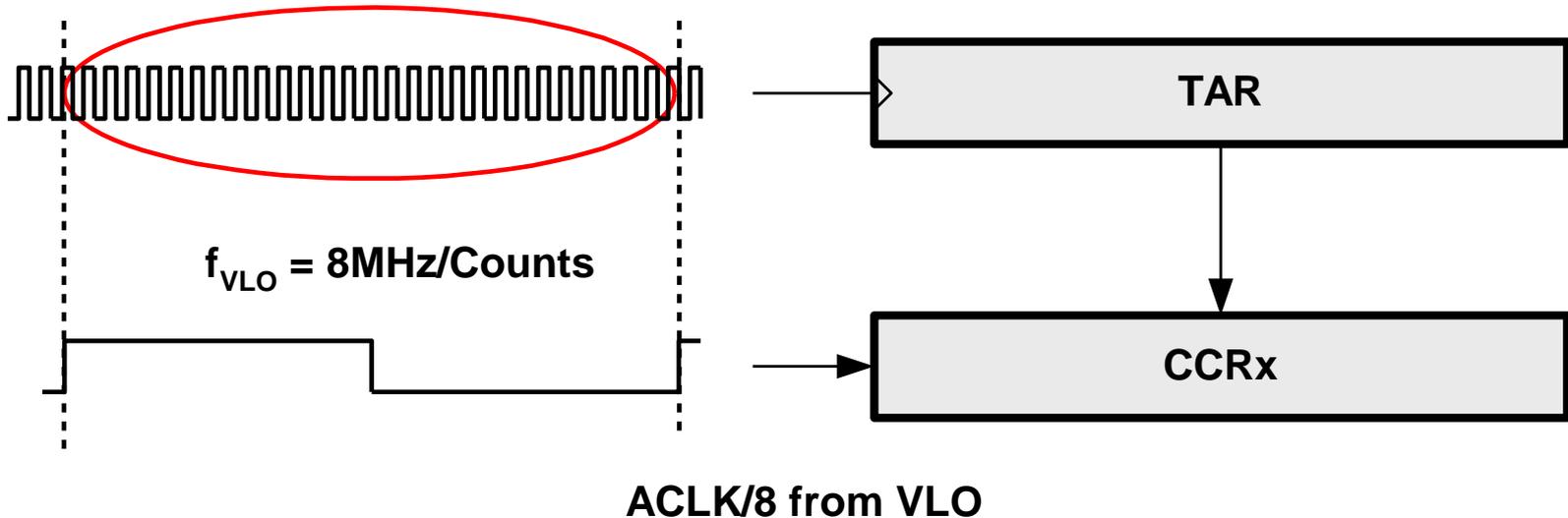
DCO Calibration Data (provided from factory in flash info memory segment A)			
DCO Frequency	Calibration Register	Size	Address
1 MHz	CALBC1_1MHz	byte	010FFh
	CALDCO_1MHz	byte	010FEh
8 MHz	CALBC1_8MHz	byte	010FDh
	CALDCO_8MHz	byte	010FCh
12 MHz	CALBC1_12MHz	byte	010FBh
	CALDCO_12MHz	byte	010FAh
16 MHz	CALBC1_16MHz	byte	010F9h
	CALDCO_16MHz	byte	010F8h

```
// Setting the DCO to 1MHz
if (CALBC1_1MHZ == 0xFF || CALDCO_1MHZ == 0xFF)
    while(1); // Erased calibration data? Trap!
BCSCTL1 = CALBC1_1MHZ; // Set range
DCOCTL = CALDCO_1MHZ; // Set DCO step + modulation
```

- ◆ G2xx1 器件只具有 1MHz DCO 校正参数。若需要较高的频率，客户必须自行校准。
- ◆ G2xx2 和 G2xx3 具有所有 4 个DCO校正参数校准值。

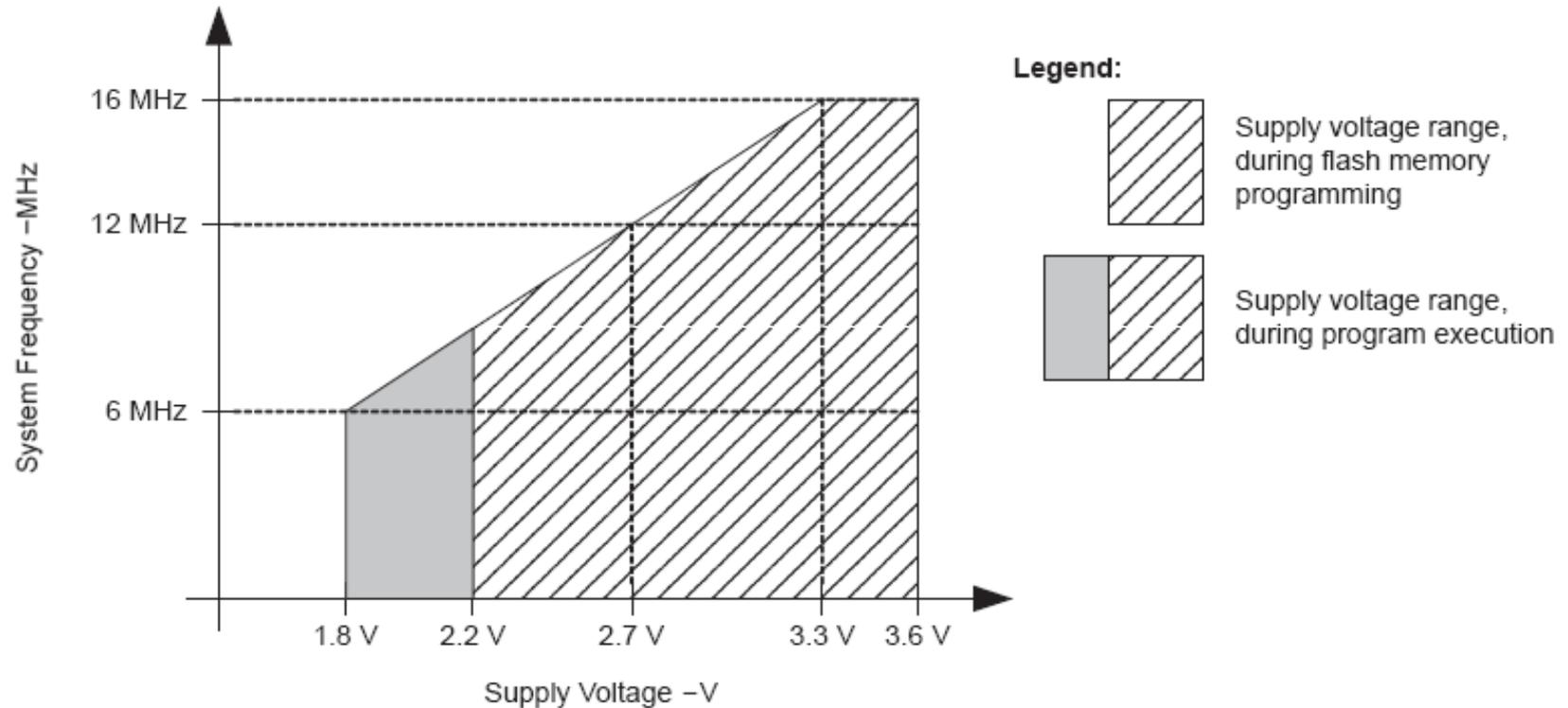
VLO 的校准

Calibrated 1 MHz DCO



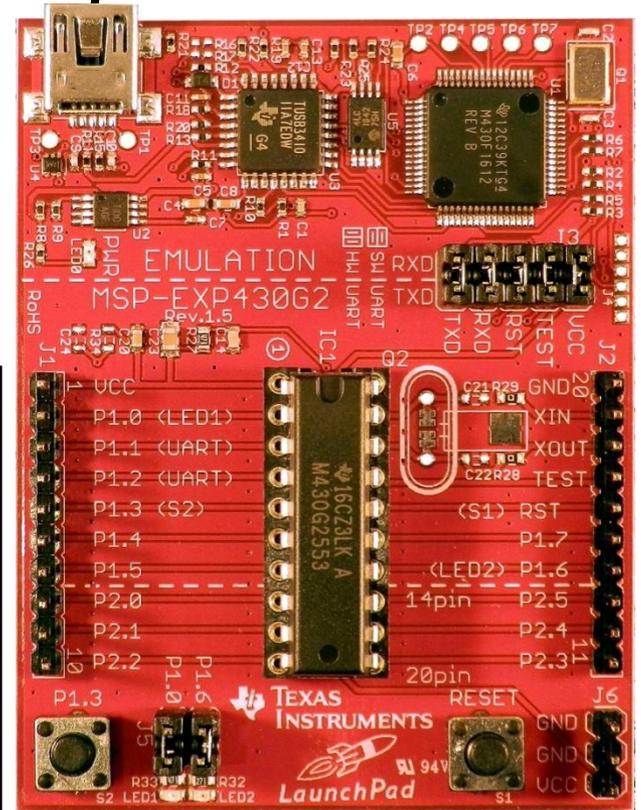
- ◆ 在运行期间可对 VLO 进行校准
- ◆ 采用校准的 1MHz DCO 为 Timer_A 提供时钟
- ◆ 利用 VLO 提供的 ACLK/8，捕获其上升沿
- ◆ 经运算 $f_{VLO} = 8\text{MHz/计数}$
- ◆ 可在网上查询代码库 (SLAA340)

系统 MCLK 和 Vcc



- ◆ 时钟速度与所需的 Vcc 必需匹配
- ◆ 需要外部 LDO 稳压器
- ◆ 如果 $V_{cc} <$ 选定频率所要求的最小电压值，则会导致程序运行不确定性
- ◆ 所有 G2xxx 器件的工作频率均可高达 16MHz

Lab2: 基本时钟配置



• Lab2

- 将 Lab2 项目导入至工作空间
- 设定 DCO = 1MHz
- 将 DCO/8 用作 MCLK, 观察LED 闪烁
- 将 VLO/8 用作 MCLK, 观察LED 闪烁

Lab 2:

```
// Configure Basic Clock  
BCSCTL1 = _____; // Set range  
DCOCTL = _____; // Set DCO step + modulation  
BCSCTL3 |= LFXT1S_2; // Set LFXT1
```

```
// Configure MCLK  
BCSCTL2 |= _____ + DIVM_3; // Set MCLK
```

- 参考用户指南、数据手册及原理图

Lab 2: 2xx 用户指南中的 BCSCTL2

5.3.3 BCSCTL2, Basic Clock System Control Register 2

7	6	5	4	3	2	1	0
SELMx		DIVMx		SELS	DIVSx		DCOR⁽¹⁾⁽²⁾
rw-0		rw-0		rw-0	rw-0		rw-0
SELMx	Bits 7-6	Select MCLK. These bits select the MCLK source.					
		00	DCOCLK				
		01	DCOCLK				
		10	XT2CLK when XT2 oscillator present on-chip. LFXT1CLK or VLOCLK when XT2 oscillator not present on-chip.				
		11	LFXT1CLK or VLOCLK				
DIVMx	Bits 5-4	Divider for MCLK					
		00	/1				
		01	/2				
		10	/4				
		11	/8				
SELS	Bit 3	Select SMCLK. This bit selects the SMCLK source.					
		0	DCOCLK				
		1	XT2CLK when XT2 oscillator present. LFXT1CLK or VLOCLK when XT2 oscillator not present				
DIVSx	Bits 2-1	Divider for SMCLK					
		00	/1				
		01	/2				
		10	/4				
		11	/8				
DCOR	Bit 0	DCO resistor select. Not available in all devices. See the device-specific data sheet.					
		0	Internal resistor				
		1	External resistor				

Lab 2: MSP430G2231 标头文件中的 BCSCCTL2

```
262 #define DIVA_2          (0x20)          /* ACLK Divider 2: /4 */
263 #define DIVA_3          (0x30)          /* ACLK Divider 3: /8 */
264
265 #define DIVS0           (0x02)          /* SMCLK Divider 0 */
266 #define DIVS1           (0x04)          /* SMCLK Divider 1 */
267 #define SELS            (0x08)          /* SMCLK Source Select 0:DCOCLK / 1:XT2CLK/LFXTCLK */
268 #define DIVM0           (0x10)          /* MCLK Divider 0 */
269 #define DIVM1           (0x20)          /* MCLK Divider 1 */
270 #define SELM0           (0x40)          /* MCLK Source Select 0 */
271 #define SELM1           (0x80)          /* MCLK Source Select 1 */
272
273 #define DIVS_0          (0x00)          /* SMCLK Divider 0: /1 */
274 #define DIVS_1          (0x02)          /* SMCLK Divider 1: /2 */
275 #define DIVS_2          (0x04)          /* SMCLK Divider 2: /4 */
276 #define DIVS_3          (0x06)          /* SMCLK Divider 3: /8 */
277
278 #define DIVM_0          (0x00)          /* MCLK Divider 0: /1 */
279 #define DIVM_1          (0x10)          /* MCLK Divider 1: /2 */
280 #define DIVM_2          (0x20)          /* MCLK Divider 2: /4 */
281 #define DIVM_3          (0x30)          /* MCLK Divider 3: /8 */
282
283 #define SELM_0          (0x00)          /* MCLK Source Select 0: DCOCLK */
284 #define SELM_1          (0x40)          /* MCLK Source Select 1: DCOCLK */
285 #define SELM_2          (0x80)          /* MCLK Source Select 2: XT2CLK/LFXTCLK */
286 #define SELM_3          (0xC0)          /* MCLK Source Select 3: LFXTCLK */
287
```

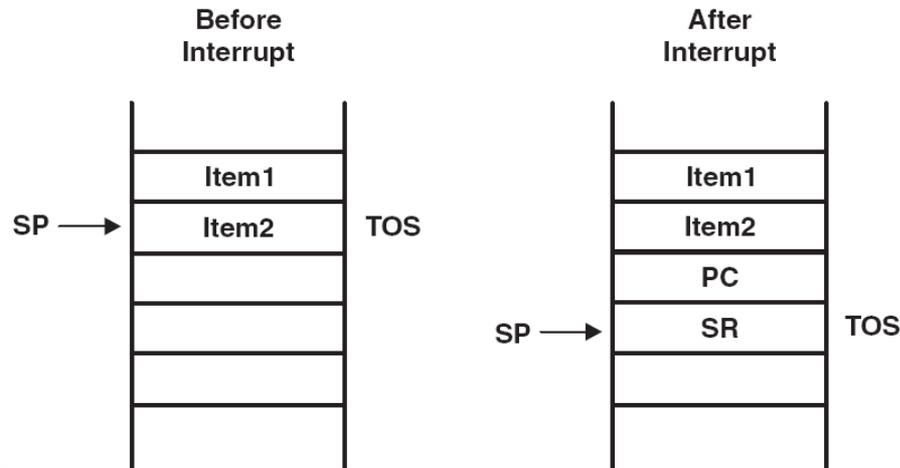
议程

- 介绍 Value Line 系列
- Code Composer Studio
- CPU 与基本时钟模块
- **中断与 GPIO**
- Timer_A 与 WDT+
- MSP430低功耗设计
- ADC10 和 Comparator_A+
- 串行通信模块
- Grace
- 电容式触摸按键解决方案

中断和堆栈

进入中断服务程序

- ◆ 当前执行的指令已完成
- ◆ 指向下一条指令的 **PC** 被推送至堆栈上
- ◆ **SR** 被推送至堆栈上
- ◆ 选择了具有最高优先级的中断
- ◆ 中断请求标志为单源中断标志时自动复位；若为多源中断标志则保持于设定状态，由软件控制
- ◆ **SR** 被清零；这将终止任何低功耗模式；由于 **GIE** 位被清零，因此将禁止执行更多的中断
- ◆ 中断向量的内容被装入 **PC**；程序将利用位于该地址的中断服务例程继续执行



Vector Table – G2231

Interrupt Source	Interrupt Flag	System Interrupt	Word Address	Priority
Power-up External Reset Watchdog Timer+ Flash key violation PC out-of-range	PORIFG RSTIFG WDTIFG KEYV	Reset	0FFFEh	31 (highest)
NMI Oscillator Fault Flash memory access violation	NMIIFG OFIFG ACCVIFG	Non-maskable Non-maskable Non-maskable	0FFFCh	30
			0FFFAh	29
			0FFF8h	28
			0FFF6h	27
Watchdog Timer+	WDTIFG	maskable	0FFF4h	26
Timer_A2	TACCR0 CCIFG	maskable	0FFF2h	25
Timer_A2	TACCR1 CCIFG TAIFG	maskable	0FFF0h	24
			0FFEEh	23
			0FFEC h	22
ADC10	ADC10IFG	maskable	0FFEAh	21
USI	USIIFG USISTTIFG	maskable	0FFE8h	20
I/O Port P2 (2)	P2IFG.6 P2IFG.7	maskable	0FFE6h	19
I/O Port P1 (8)	P1IFG.0 to P1IFG.7	maskable	0FFE4h	18
			0FFE2h	17
			0FFE0h	16
Unused			0FFDEh to 0FFCDh	15 - 0

中断处理函数编程

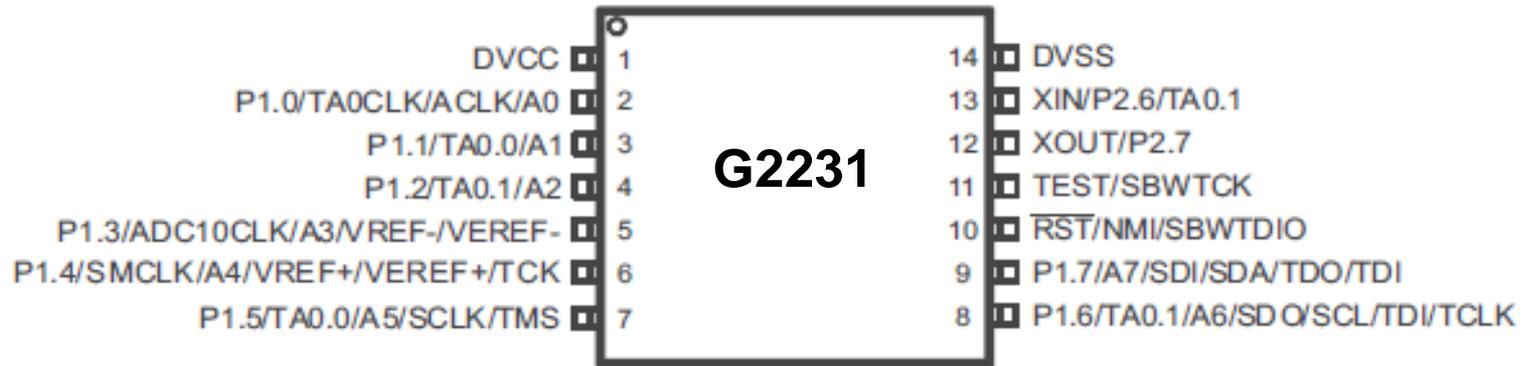
```
#pragma vector=WDT_VECTOR
__interrupt void WDT_ISR(void)
{
    IE1  &= ~WDTIE;           // disable interrupt
    IFG1 &= ~WDTIFG;         // clear interrupt flag
    WDTCTL = WDTPW + WDTTHOLD; // put WDT back in hold state
    BUTTON_IE |= BUTTON;     // Debouncing complete
}
```

#pragma vector — 下面的函数是一个用于所列矢量的 ISR

__interrupt void — 识别 ISR 名称

无特别需要的返回值

GPIO 端口



GPIO 寄存器

输入寄存器 PxIN
输出寄存器 PxOUT
方向寄存器 PxDIR
电阻启用 PxREN
功能选择 PxSEL
功能选择2 PxSEL2
中断边缘 PxIES
中断启用 PxIE
中断标记 PxIFG

用于 GPIO 中断

GPIO 代码范例

```
P1DIR |= BIT4;  
P1SEL |= BIT4;
```



```
P1DIR |= BIT0;  
P1OUT |= BIT0;
```



引脚复用

Table 16. Port P1 (P1.4) Pin Functions

PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS / SIGNALS ⁽¹⁾					
			P1DIR.x	P1SEL.x	P1SEL2.x	ADC10AE.x (INCH.x=1) ⁽²⁾	JTAG Mode	CAPD.y
P1.4/ SMCLK/ TA0.2/ VREF+ ⁽²⁾ / VEREF+ ⁽²⁾ / A4 ⁽²⁾ / CA4/ TCK/ Pin Osc	4	P1.x (I/O)	I: 0; O: 1	0	0	0	0	0
SMCLK		1	1	0	0	0	0	
TA0.2		1	1	1	0	0	0	
TA0.CCI2A		0	1	1	0	0	0	
VREF+		X	X	X	1	0	0	
VEREF+		X	X	X	1	0	0	
A4		X	X	X	1 (y = 4)	0	0	
CA4		X	X	X	0	0	1 (y = 4)	
TCK		X	X	X	0	1	0	
Capacitive sensing		X		0	1	0	0	

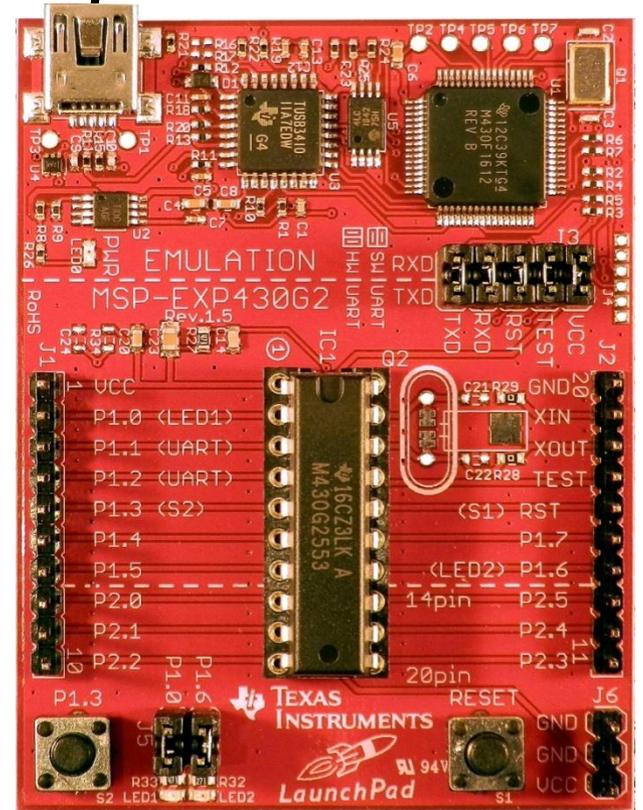
- ◆ 每个引脚具有多项功能
- ◆ 在对应的寄存器选择相应的引脚功能
- ◆ 具体详见各器件的数据手册

Lab3: GPIO



Lab3

- 设置 P1.3 为按钮
- 设置 P1.0 控制 LED
- 利用按钮进行触发LED翻转



Lab 3:

```
P1DIR |= BIT0; // Set P1.0 to output direction
P1IES |= BIT3; // P1.3 Hi/lo edge
_____ &= ~BIT3; // P1.3 IFG cleared
_____ |= BIT3; // P1.3 interrupt
```

```
// Port1 interrupt service routine
#pragma vector = _____
__interrupt void Port_1(void)
```

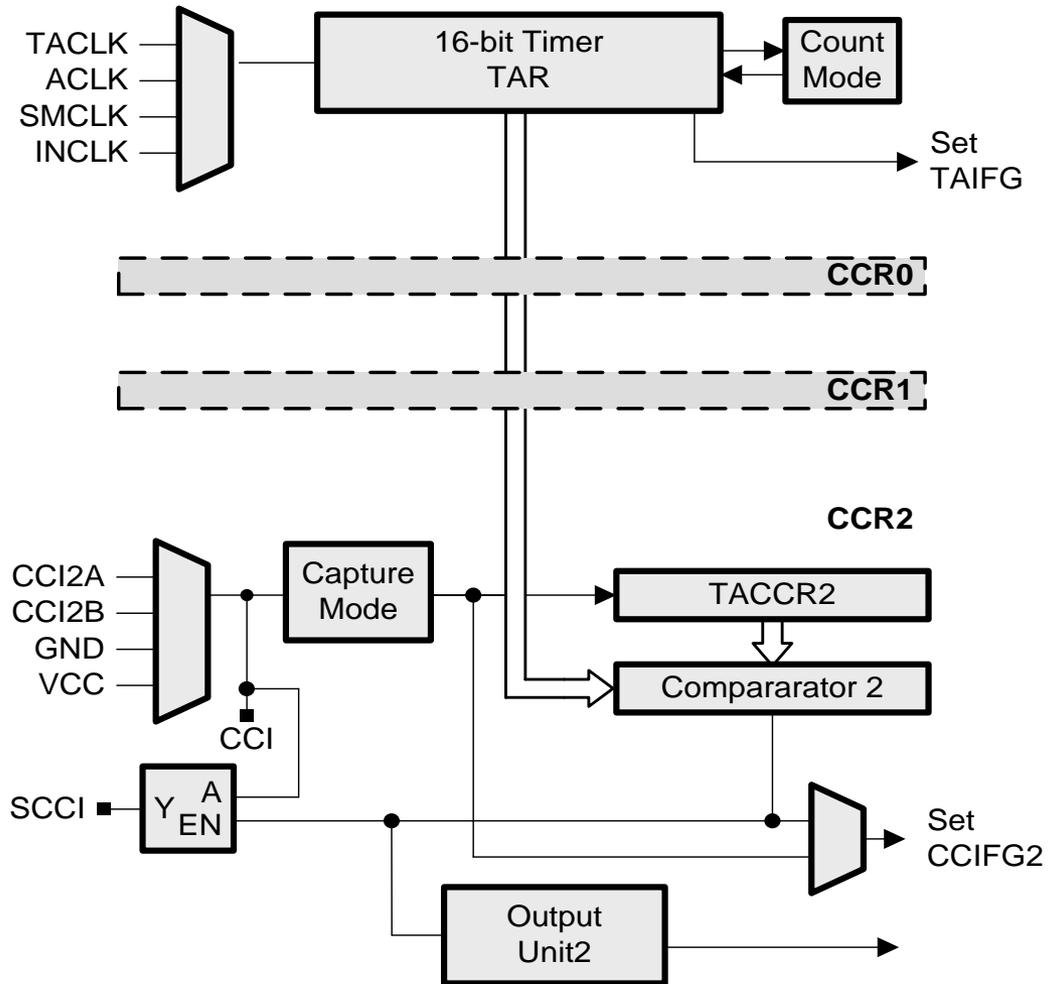
```
// Port1 interrupt service routine
P1OUT ^= BIT0; // P1.0 = toggle
_____ &= ~BIT3; // P1.3 IFG cleared
```

议程

- 介绍 Value Line 系列
- Code Composer Studio
- CPU 与基本时钟模块
- 中断与 GPIO
- **Timer_A 与 WDT+**
- MSP430低功耗设计
- ADC10 和 Comparator_A+
- 串行通信模块
- Grace
- 电容式触摸按键解决方案

Timer_A

- ◆ 异步 16 位定时器/计数器
- ◆ 连续、递增-递减、递增计数模式
- ◆ 多个捕获/比较寄存器
- ◆ PWM 输出
- ◆ 中断向量寄存器用于实现中断快速响应
- ◆ 能触发 DMA 传输
- ◆ 所有 MSP430 上均有 Timer_A 模块



Timer_A 计数模式

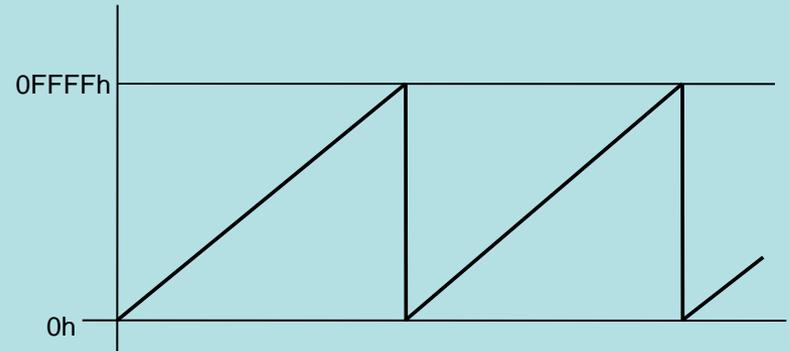
停止/暂停

定时器计数停止/暂停



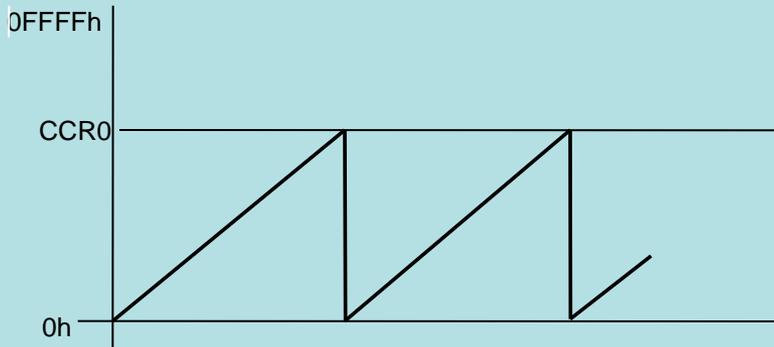
连续

定时器连续递增计数



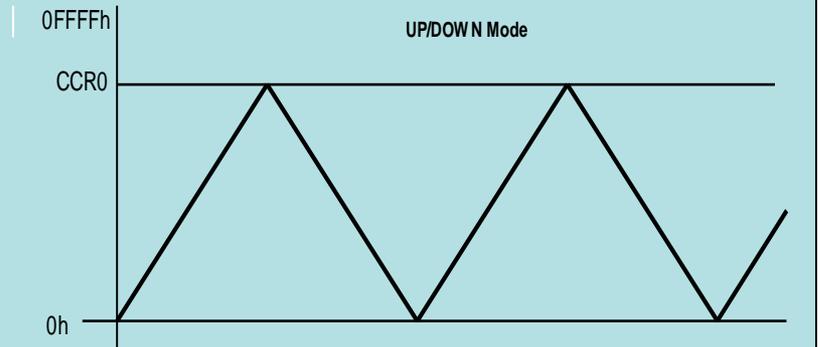
递增

定时器在 0 和 CCR0 之间计数



递增/递减

定时器在 0 - CCR0 - 0 之间计数



CCR - 计数比较寄存器

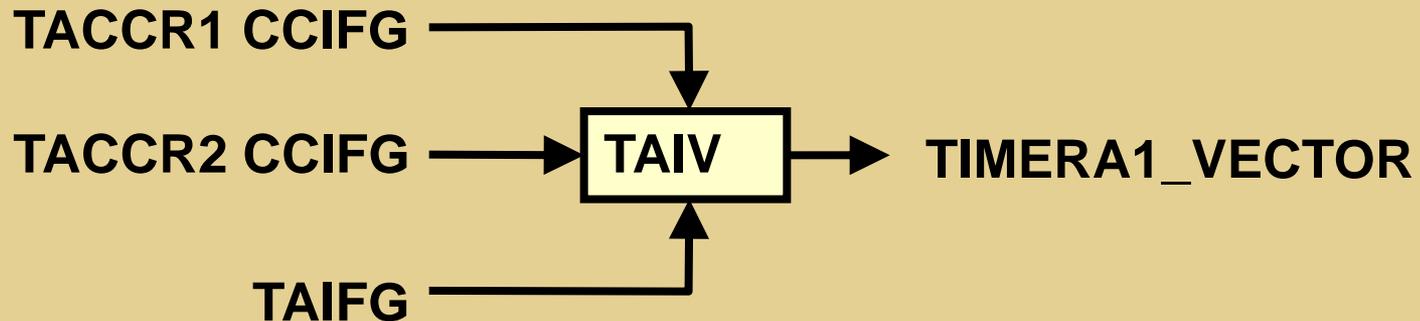
Timer_A 中断

Timer_A 捕获/比较寄存器 0 中断标记 (TACCR0) 生成单个中断向量:

TACCR0 CCIFG → **TIMERA0_VECTOR**

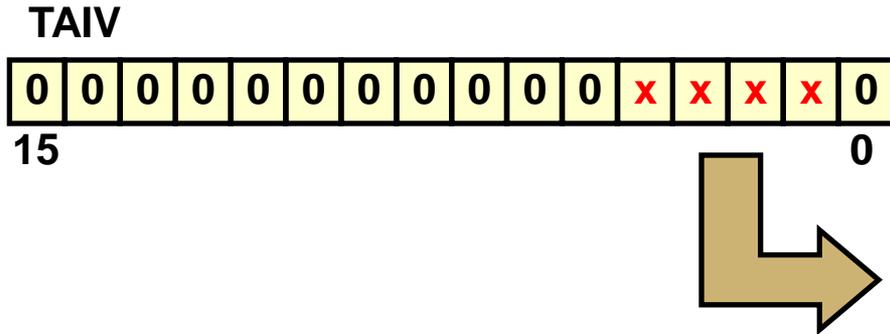
无需处理程序

对 TACCR1、2 和 TA 中断标记进行优先级处理，并采用 Timer_A 中断向量寄存器 (TAIV) 将之组合成为另一个中断向量。



代码必须包含一个处理程序，以确定触发的是哪一个 Timer_A1 中断

TAIV 处理程序范例



源	TAIV 内容
没有即将发生的中断	0
TACCR1 CCIFG	02h
TACCR2 CCIFG	04h
保留	06h
保留	08h
TAIFG	0Ah
保留	0Ch
保留	0Eh

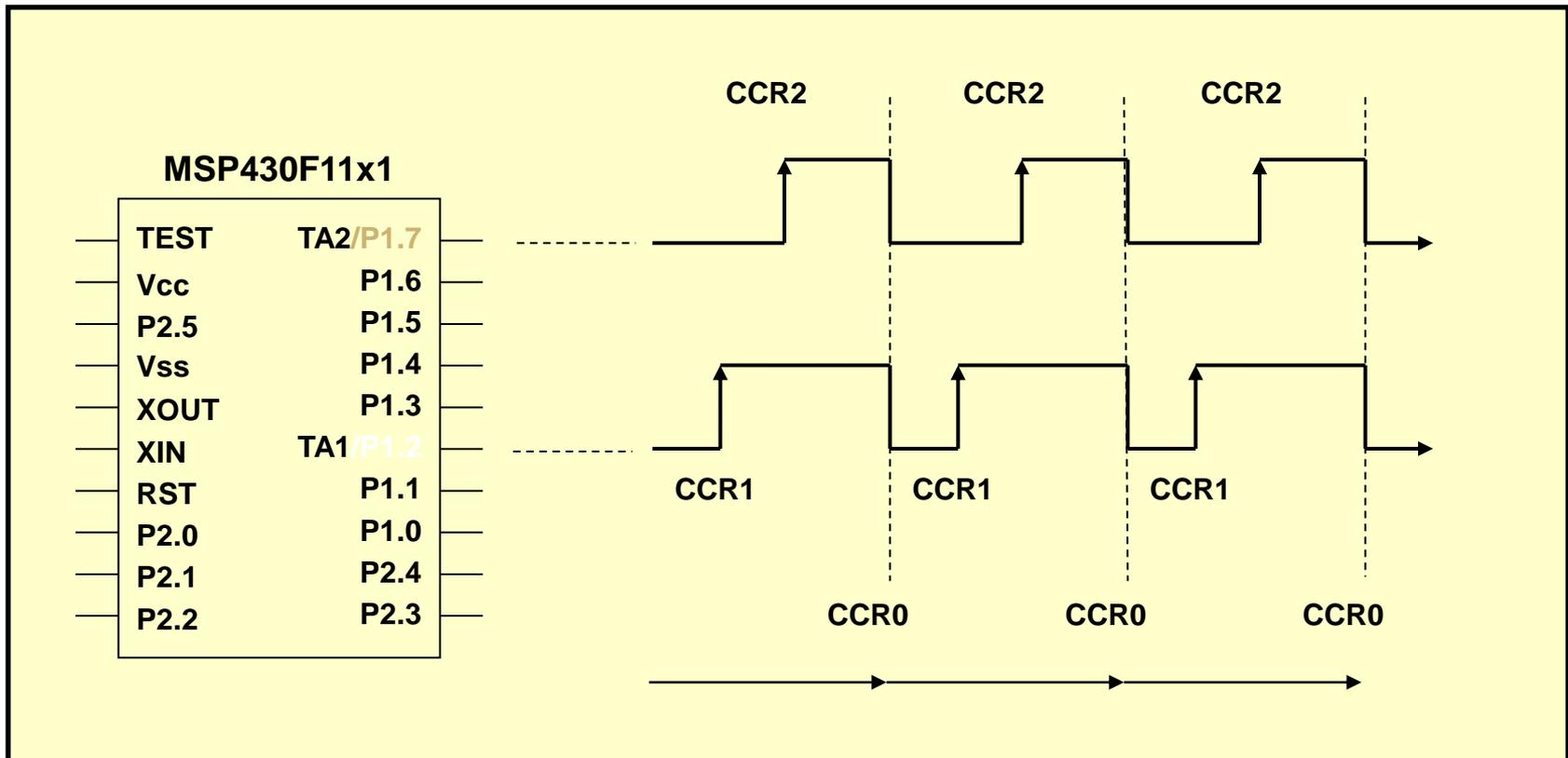
```
#pragma vector = TIMERA1_VECTOR
__interrupt void TIMERA1_ISR(void)
{
    switch(__even_in_range(TAIV,10))
    {
        case 2 :          // TACCR1 CCIFG
            P1OUT ^= 0x04; break;
        case 4 :          // TACCR2 CCIFG
            P1OUT ^= 0x02; break;
        case 10 :         // TAIFG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

C 代码

```
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
```

汇编代码

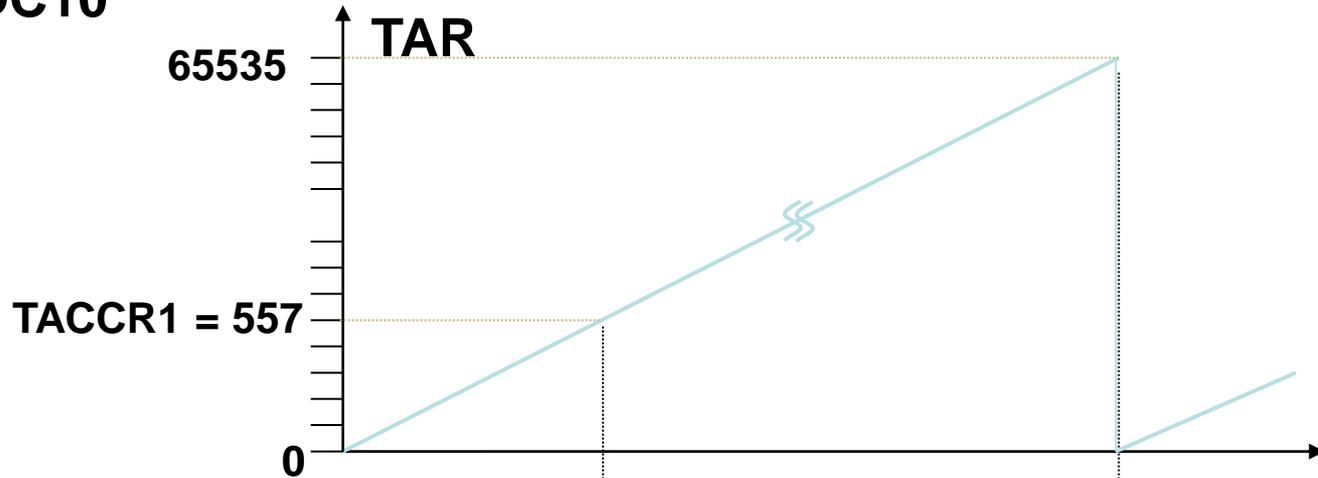
Timer_A PWM 范例



- ◆ PWM完全自动发送
- ◆ 可通过对CCR的配置，生成多路相同频率和不同占空比的PWM
- ◆ 请浏览 MSP430 的相关网址的代码范例

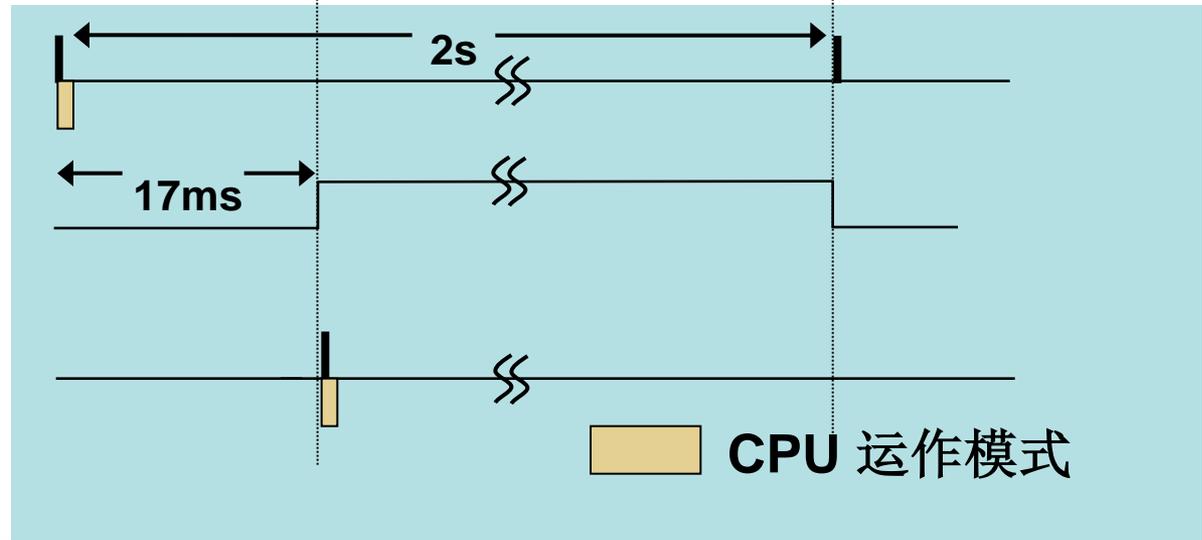
采用 Timer_A 的直接硬件控制

范例: ADC10



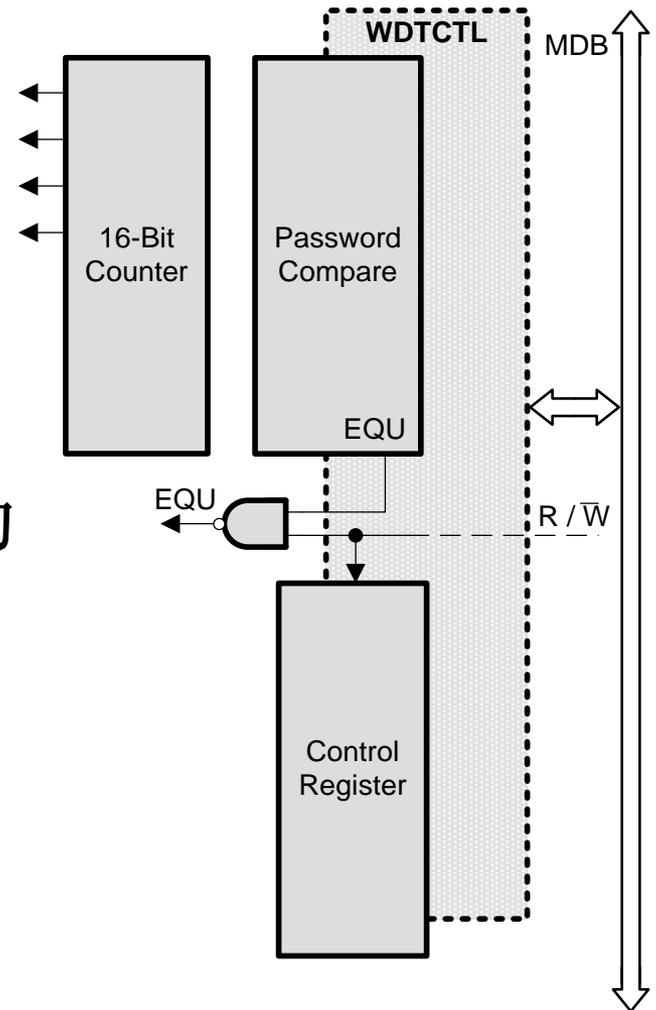
TAIFG:
基准和 ADC 接通

TACCR1:
基准延迟 / ADC 触发
ADC10IFG:
处理 ADC 结果
基准 / ADC 关断



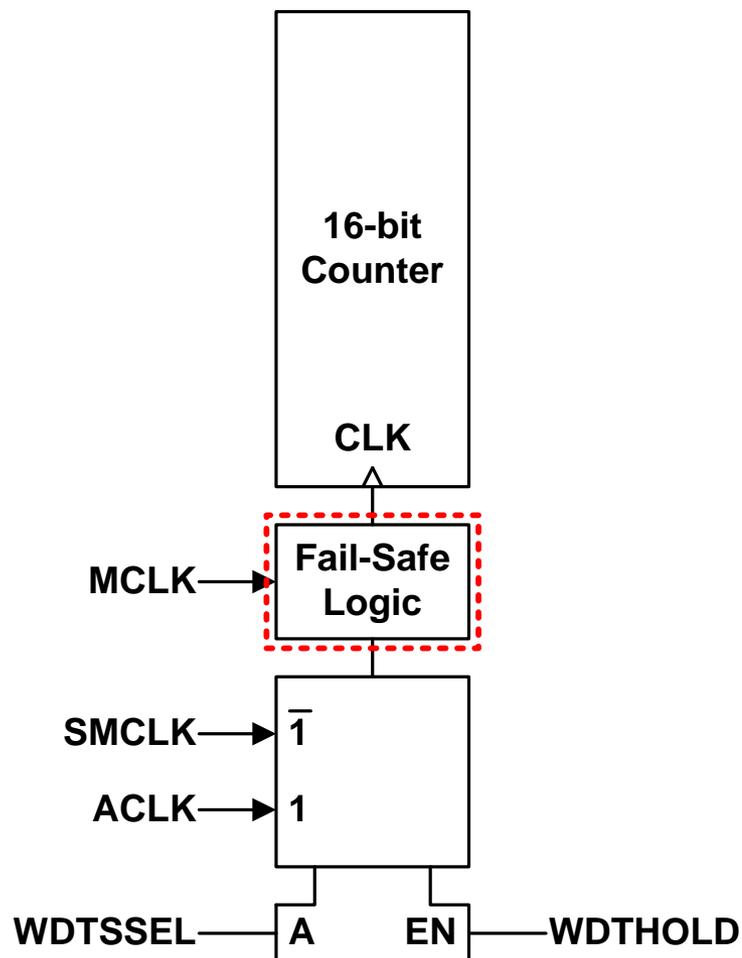
WDT+ 模块：概要

- 在所有 **MSP430** 器件上均有**WDT**
- 两种模式
 - 看门狗
 - 间隔定时器
- 访问受密码保护
- 看门狗模式和定时器模式拥有不同的单独中断向量
- **Clock**可由 **ACLK** 或 **SMCLK** 提供
- 控制 **RST/NMI** 引脚模式
- **WDT+** 添加了故障保护时钟



看门狗定时器故障保险

- ◆ 倘若 **ACLK/SMCLK** 发生故障，则时钟脉冲源 = **MCLK**
(**WDT+** 故障保护特性)
- ◆ 假如 **MCLK** 由一个晶振 (**XTAL**) 提供，且晶振发生故障，则 **MCLK = DCO**
(**XTAL** 故障保护特性)



WDT: 常见设计问题

- 程序保持对其自身的复位
- 程序动作反常 – 执行是否到达清除WDT的位置？
 - 设置一个靠近 main() 起点的中断，以查看代码是否重新启动
- **CPU 甚至在到达第一条指令之前似乎就出现了冻结现象**
 - C 程序是不是具有大量初始化数据？
 - 通常只会在拥有超大RAM空间器件上出现
 - 解决方案：在 __low_level_init() 函数中关闭看门狗

```
void main(void)
{
    WDTCTL = WDTPW+WDTHOLD;           // Stop the dog
    .
    .
}
```

WDT: 间隔定时器功能

- 当定时到达时没有 **PUC** 产生
- 如果 **WDTIE** 和 **GIE** 在到达间隔时被设定，则生成一个 **WDT** 间隔中断（而不是复位中断）
- 定时间隔可编程选择

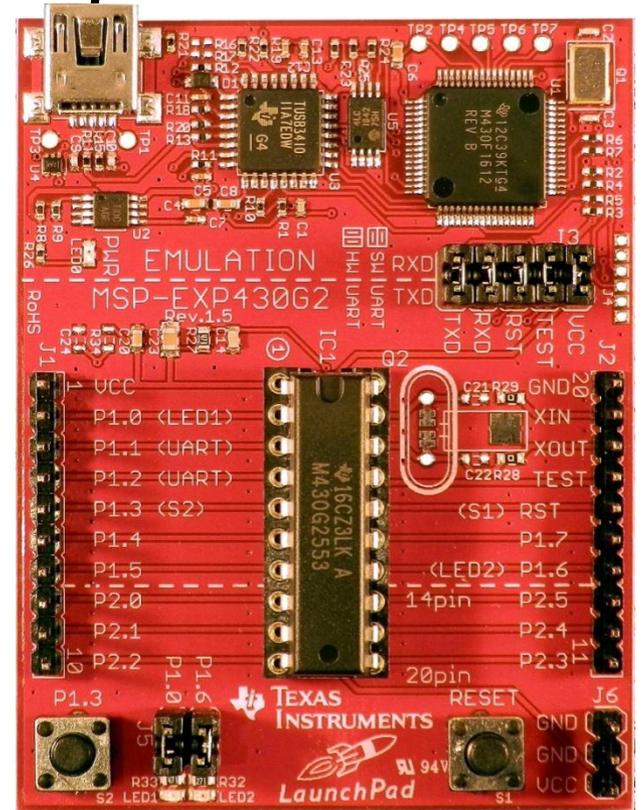


Lab4: 定时器和中断



Lab4

- 采用 `Timer_A` 再次完成Lab2实验
- 配置 `Timer_A` 计数周期: **5100**
- 当 `TAR = 100` 时, 产生一个中断触发LED控制



Lab 4 源代码

```
// Configure TimerA  
→ TACTL = _____; // Source: ACLK, UP mode  
CCR0 = 5100; //Timer count 5100  
CCR1 = 100; //Timer count 100  
CCTL0 = CCIE; //CCR0 interrupt enabled  
CCTL1 = CCIE; //CCR1 interrupt enabled
```

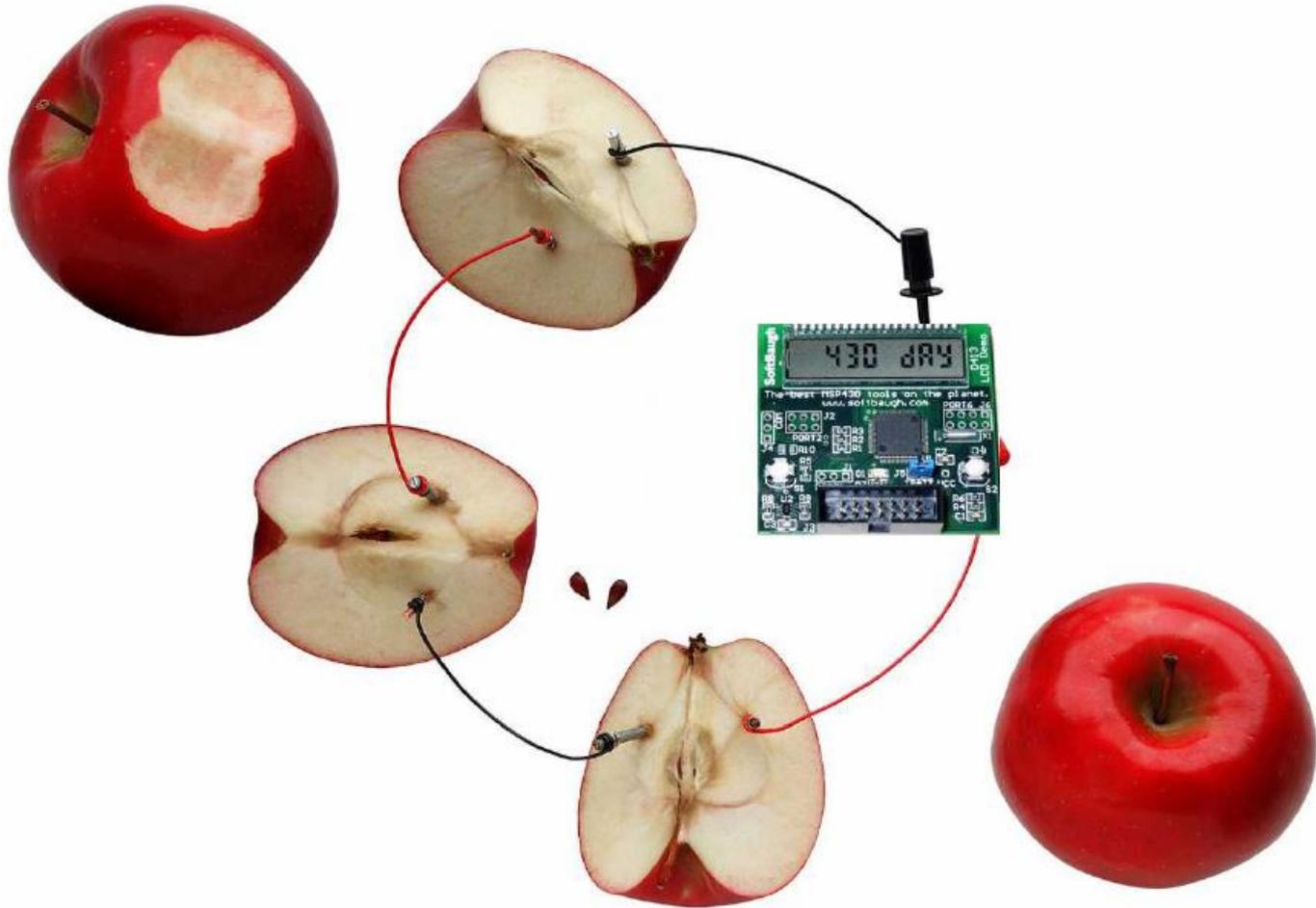
```
// Timer A0 interrupt service routine  
→ #pragma vector = _____  
__interrupt void Timer_A0(void)
```

```
// Timer A1 interrupt service routine  
→ #pragma vector = _____  
__interrupt void Timer_A1(void)
```

议程

- 介绍 Value Line 系列
- Code Composer Studio
- CPU 与基本时钟模块
- 中断与 GPIO
- Timer_A 与 WDT+
- **MSP430低功耗设计**
- ADC10 和 Comparator_A+
- 串行通信模块
- Grace
- 电容式触摸按键解决方案

超低功耗特性



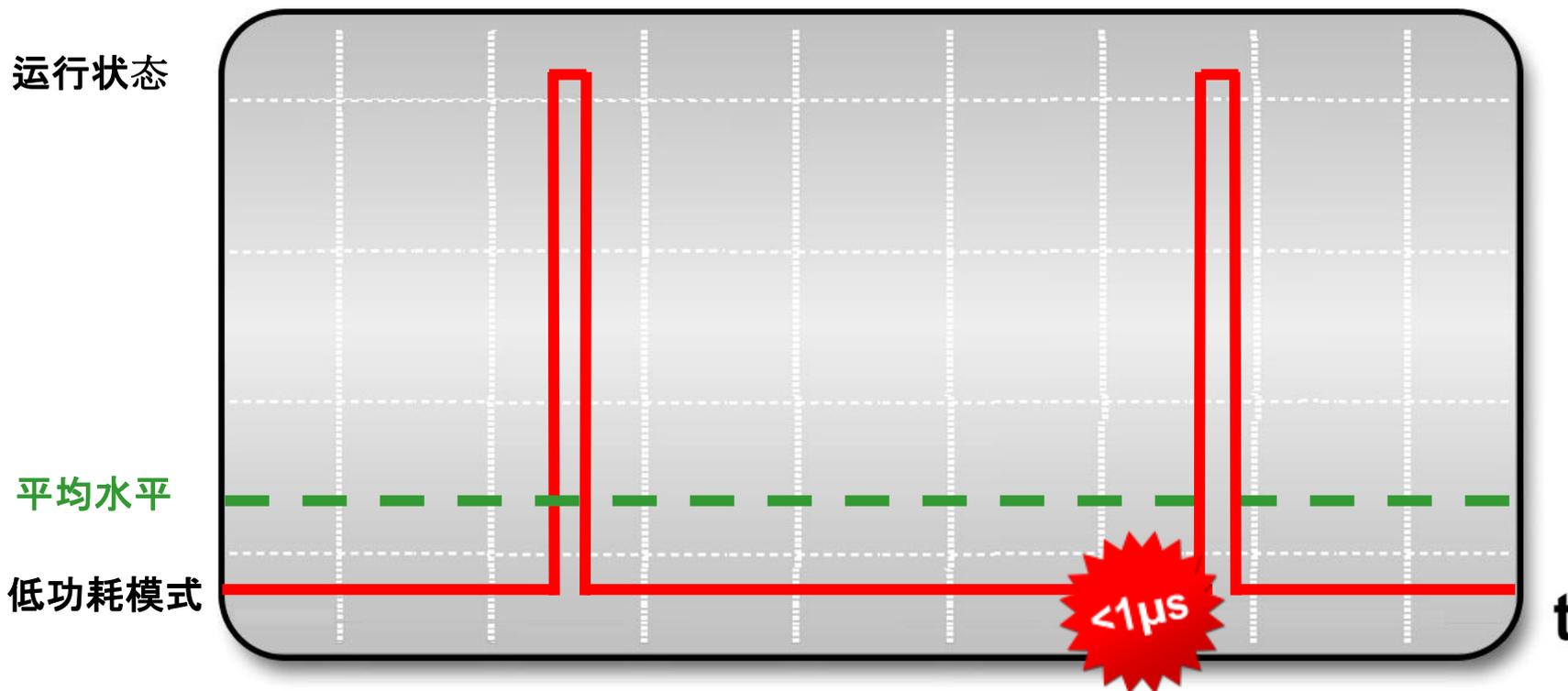
超低功耗是MSP430的NDA

- ◆ **MSP430** 从一开始就是专为超低功耗 (ULP) 而设计的
- ◆ 外设专为减少功耗和最大限度地降低 **CPU** 占用率而优化
- ◆ 智能型低功耗外设能独立于 **CPU** 而工作，并让系统在更长的时间里处于较低功耗模式

www.ti.com/ulp

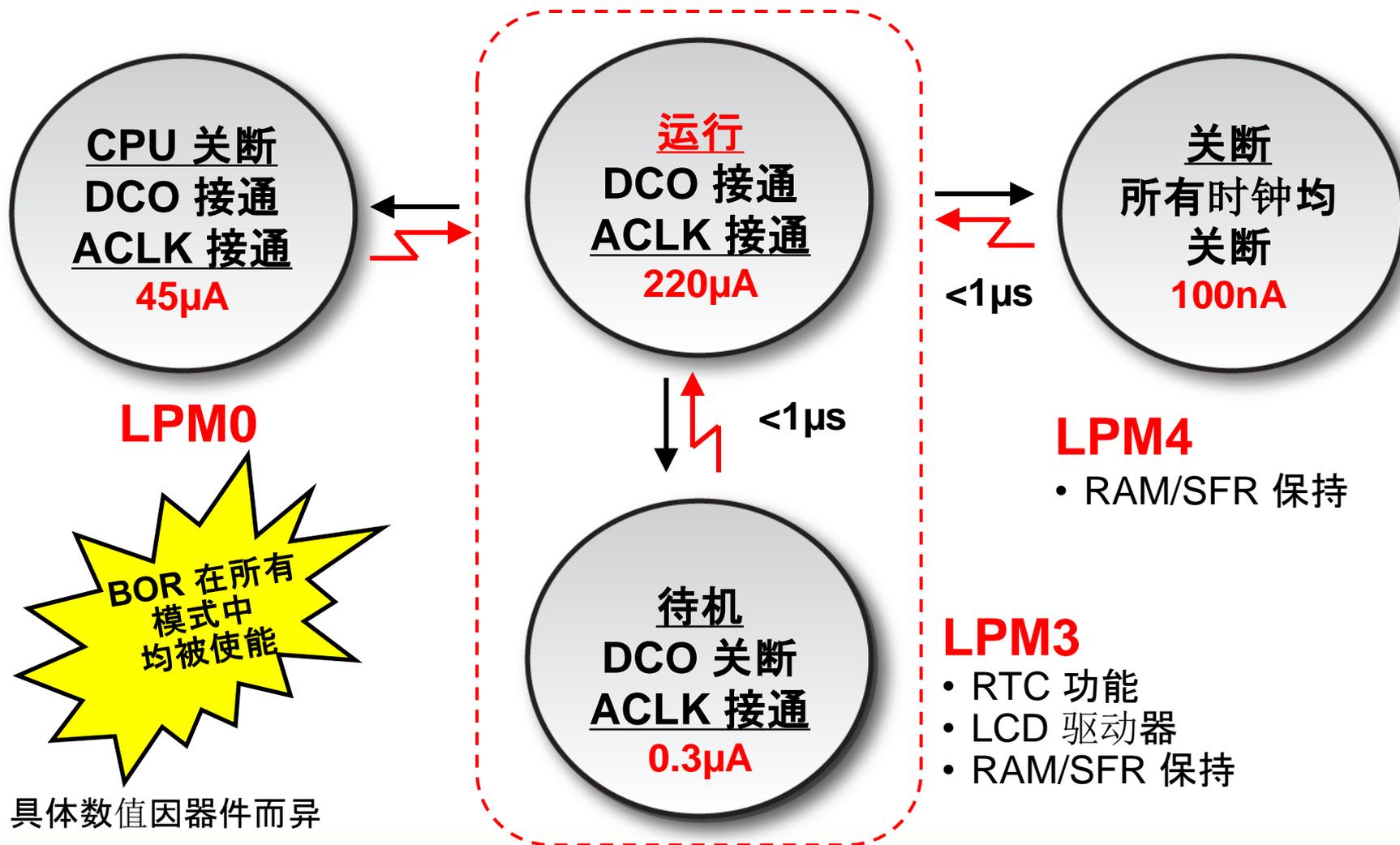
- ✓ 多种操作模式
 - ◆ 100 nA 断电 (RAM 保持)
 - ◆ 0.3 μ A 待机
 - ◆ 110 μ A / MIPS (采用 RAM)
 - ◆ 220 μ A / MIPS (采用闪存)
- ✓ 即时可**稳定**工作的高速时钟
- ✓ 1.8 至 3.6V **单电源**操作
- ✓ **零功率、始终工作的 BOR**
- ✓ **<50nA** 的引脚漏电流
- ✓ 可最大限度地减少每项任务的执行周期的 **CPU**
- ✓ 低功耗智能外设
 - ◆ 自动传输数据的 **ADC**
 - ◆ 功耗微乎其微的定时器
 - ◆ **100 nA** 模拟比较器
- ✓ 可保证所需工作条件下的性能

超低功耗工作模式



- ◆ 最大限度地减少运行时间
- ◆ 最大限度地增加系统处于**低功耗模式**
- ◆ 按功能需求的中断型程序设计（具有 $<1\mu\text{s}$ 的唤醒时间）
- ◆ 始终工作的零功耗**欠压复位 (BOR)** 功能电路

MSP430 低功耗模式



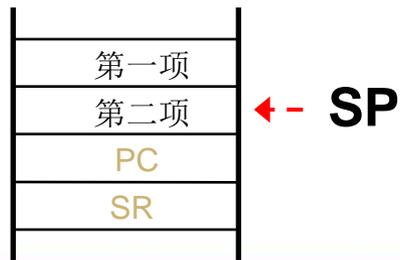
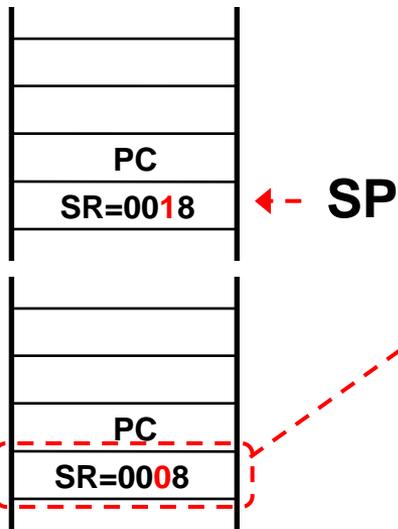
低功耗模式配置

保留	V	SCG1	SCG0	OSC OFF	CPU OFF	GIE	N	Z	C
R2/SR									
运行模式		0	0	0	0				约 250uA
LPM0		0	0	0	1				约 35uA
LPM3		1	1	0	1				约 0.8uA
LPM4		1	1	1	1				约 0.1uA

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

采用汇编程序的 LPM

低功耗模式的堆栈操作



```
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
```

ULP 简单易用!

使用我们的低功耗模式很容易

```
void main(void)
{
    WDT_init(); // initialize Watchdog Timer
    while(1)
    {
        __bis_SR_register(LPM3_bits + GIE); // 进入 LPM3, 启用中断
        activeMode();                       // 运行的代码
    }
}

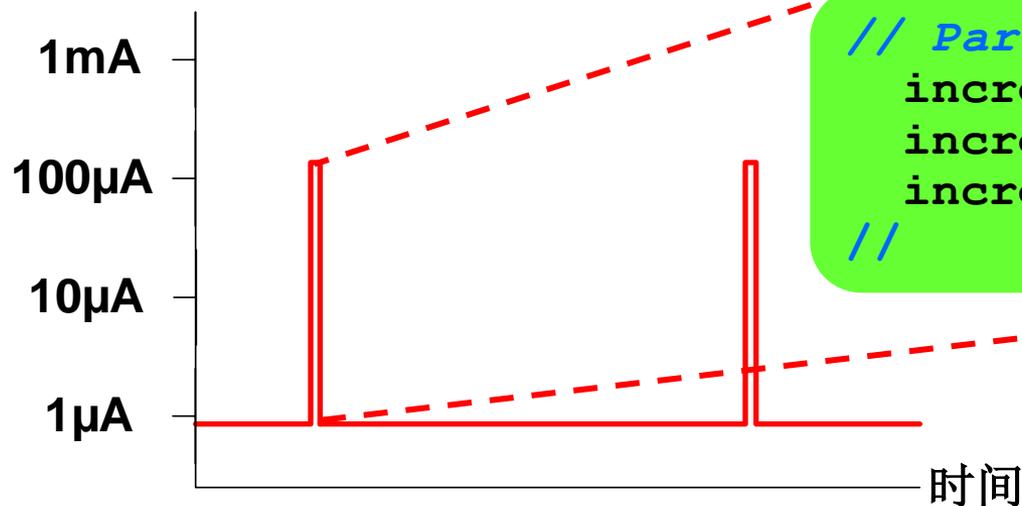
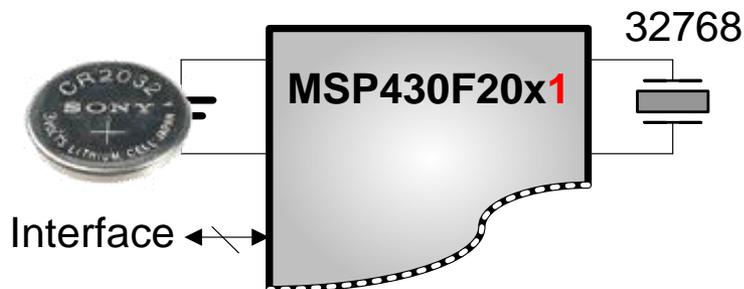
#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer (void)
{
    __bic_SR_register_on_exit(LPM3_bits); // 从 0(SR) 清除 LPM3 位, 退出 LPM3, 进入运行模式
}
```

可连续工作 10 年的嵌入式实时时钟

= LPM3 + RTC_Function

$$0.80\mu\text{A} + 250\mu\text{A} * \frac{100\mu\text{s}}{1000000\mu\text{s}}$$

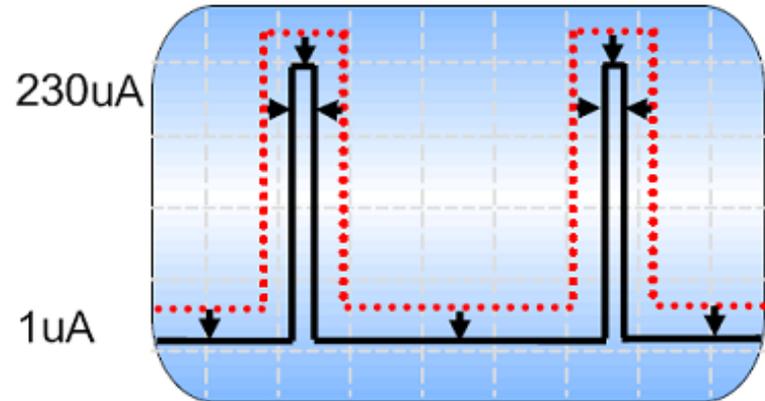
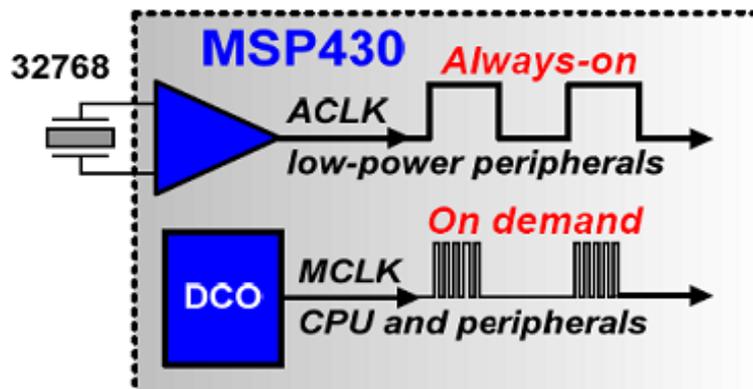
$$0.80\mu\text{A} + 0.030\mu\text{A} = 0.83\mu\text{A}$$



```
// Partial RTC_Function  
incrementseconds();  
incrementminutes();  
incrementhours();  
//
```

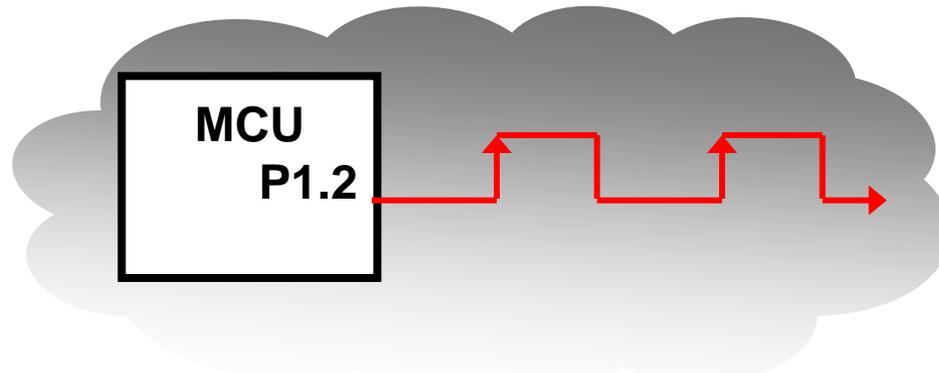
低功耗操作

- ◆ 低功耗MSP430 应用：
 - ◆ 可最大限度地减小瞬间吸收电流
 - ◆ 能最大限度地增加系统处于低功耗模式的时间
- ◆ MSP430 具备固有的低功耗特性，但是您的设计对于系统功耗也有很大的影响
- ◆ 正确的低功耗设计方法能起到改进的作用



即时稳定时钟

将软件功能移至外设来完成



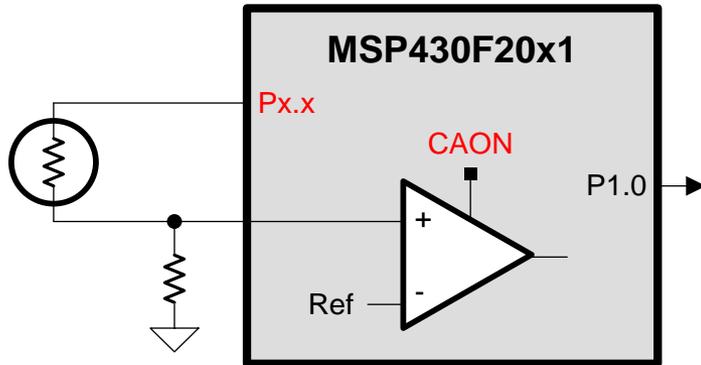
```
// Endless Loop  
for (;;)   
{  
    P1OUT |= 0x04; // Set  
    delay1();  
    P1OUT &= ~0x04; // Reset  
    delay2();  
}
```

100% CPU 负荷

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

零 CPU 负荷

片上模拟外设的电源管理

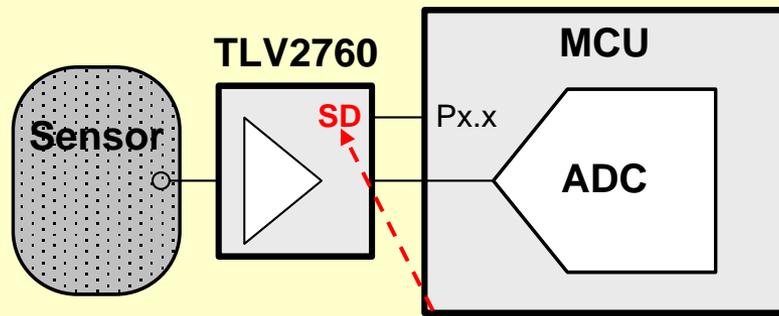


Comparator_A

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
```

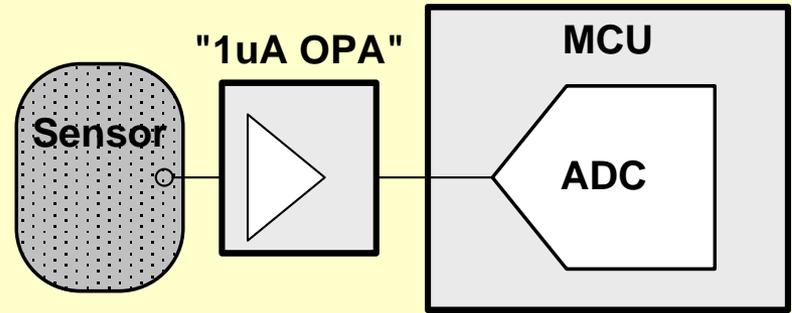
外部设备的电源管理



0.01uA = 关断模式

20uA = 运行模式

0.06uA = 平均值



1uA = 静态模式

1uA = 运行模式

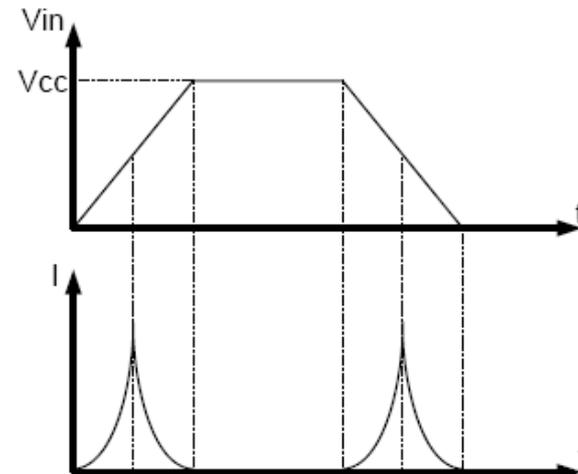
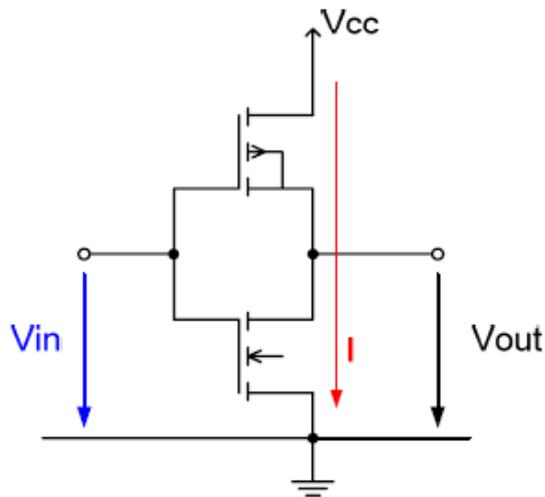
1uA = 平均值

具有关断模式的运算放大器其总功耗可降低 20 倍

未用引脚接口

- ◆ 数字输入引脚需防止遭受击穿电流的影响
 - ◆ 如果引脚设为输入且浮动，则当输入电压在 V_{IL} 与 V_{IH} 之间时会产生击穿电流。
- ◆ 未用的 **GPIO** 应当：
 - ◆ 设为输出
 - ◆ 可由一个外围线路连接至 V_{CC} 或 GND
 - ◆ 或可接上一个上拉/下拉电阻

(Digital) CMOS Inverter



Lab 5:

仅采用一行代码即可进入低功耗模式！

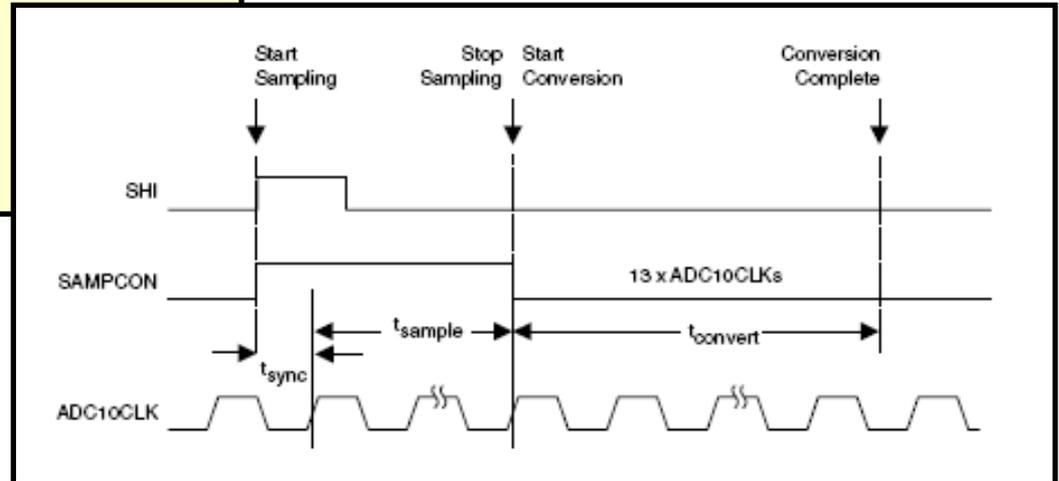
→ `_BIS_SR(_____) //Enter Low Power Mode;`

议程

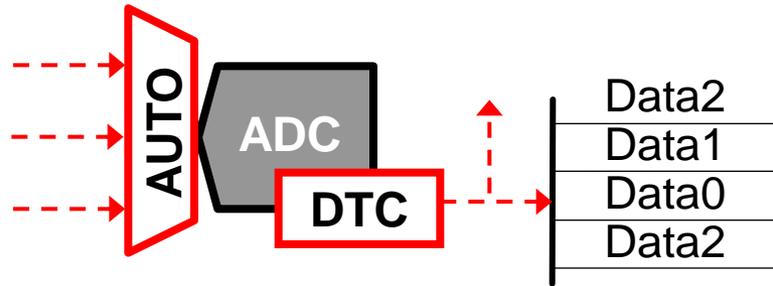
- 介绍 Value Line 系列
- Code Composer Studio
- CPU 与基本时钟模块
- 中断与 GPIO
- Timer_A 与 WDT+
- MSP430低功耗设计
- **ADC10 和 Comparator_A+**
- 串行通信模块
- Grace
- 电容式触摸按键解决方案

采样定时

- ◆ 参考电压建立时间最大为**30uS**
- ◆ 可选的采样保持时间
- ◆ 转换需**13**个时钟周期
- ◆ 可选的时钟脉冲源
 - **ADC10OSC** (约 **5MHz**)
 - **ACLK**
 - **MCLK**
 - **SMCLK**



自动扫描 + DTC 可实现性能提升



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

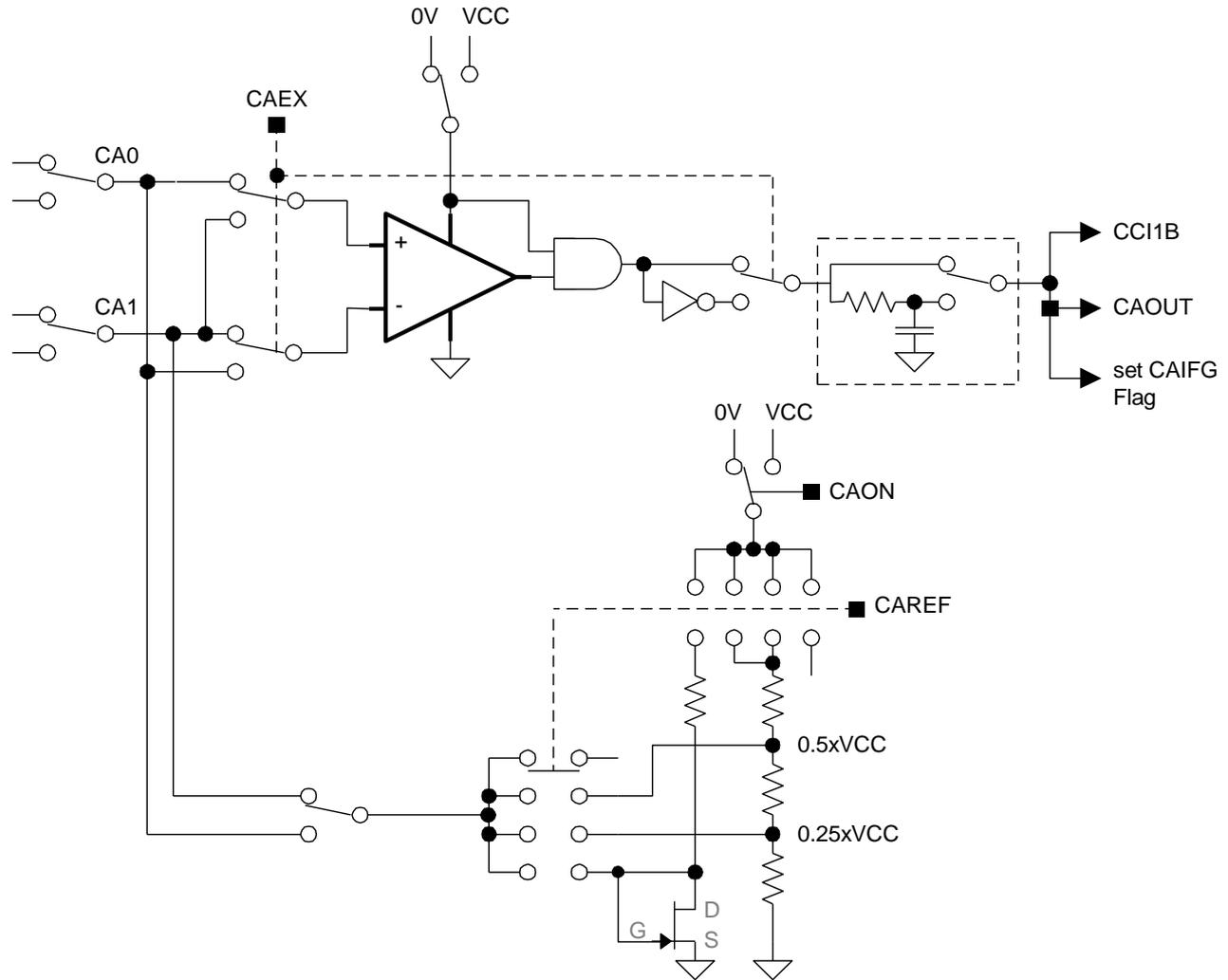
```
// Autoscan + DTC
_BIS_SR(CPUOFF);
```

全自动

70 个周期/样本

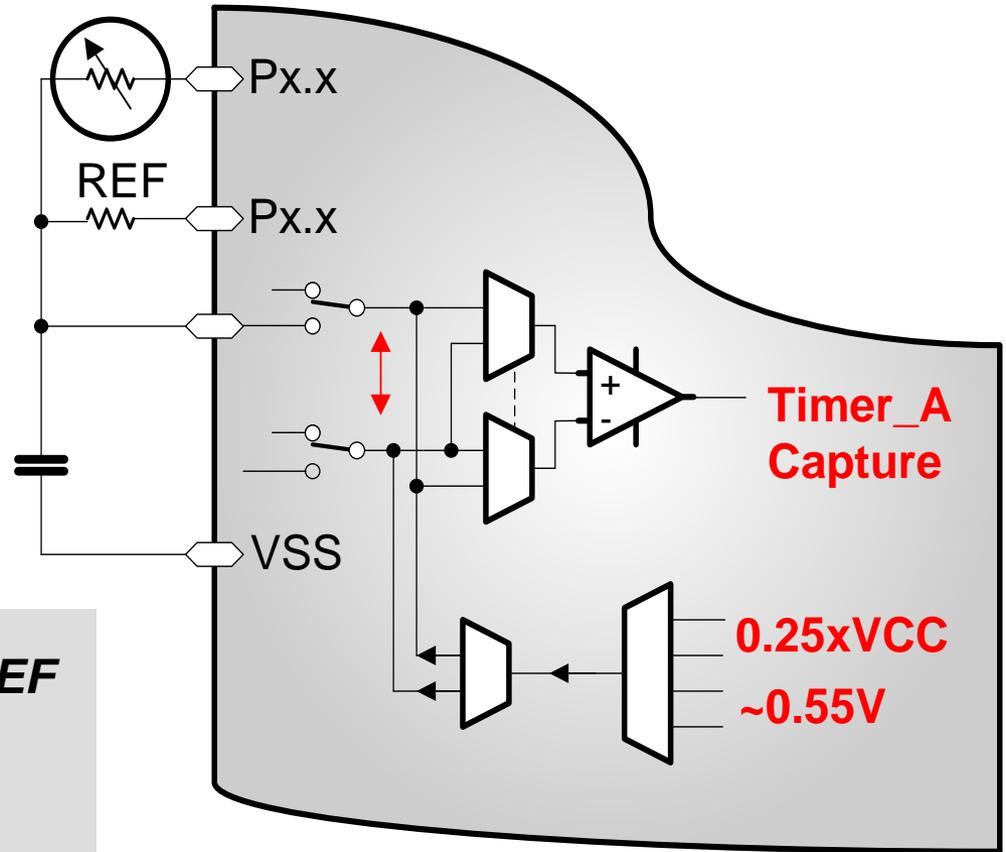
Comparator_A

- ◆ 可用内部电压参考，也可从外部接入
- ◆ 内建低通滤波器可利用软件来选择开或关
- ◆ 多路输入选择开关
- ◆ 具有中断功能



基于比较器的Slope ADC

- ◆ 10 位以上的精度
- ◆ 非常适合电阻型传感器
- ◆ 超低成本应用
- ◆ 应用笔记 SLAA038

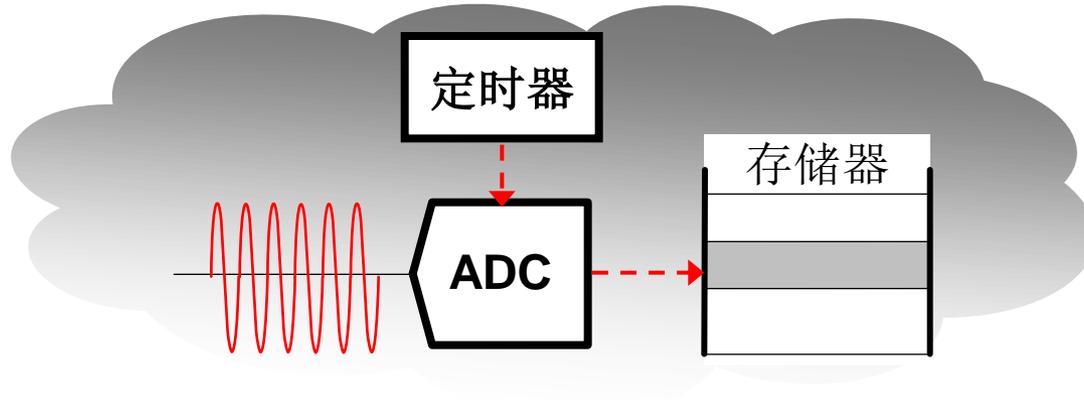


$$t_x = R_x \times C \times \ln \frac{V_{CAREF}}{V_{CC}}$$

...

$$R_{NTC} = 10k \times \frac{t_{NTC}}{t_{10k}}$$

定时器触发采样



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

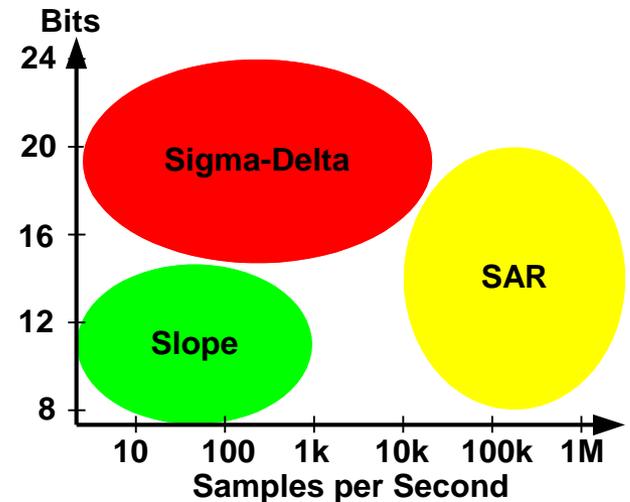


定时器触发的中断 – 无软件等待循环

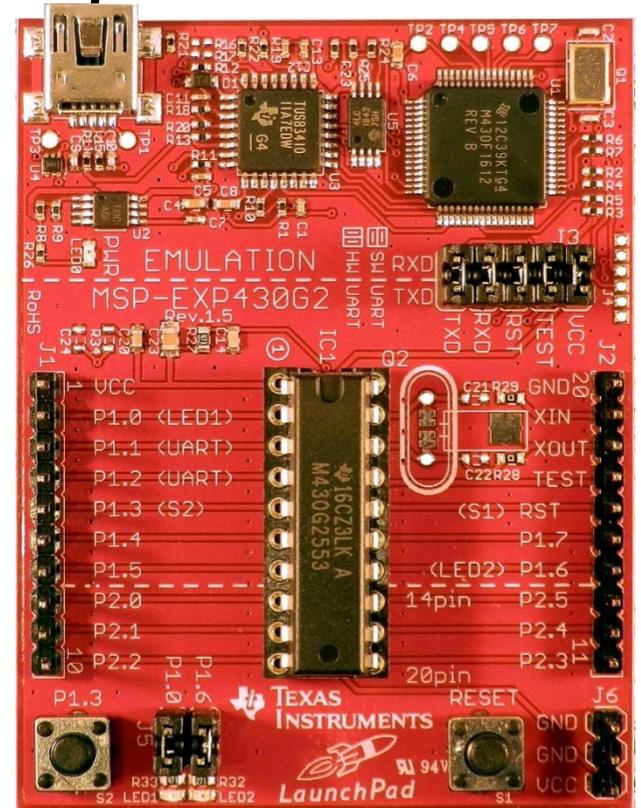
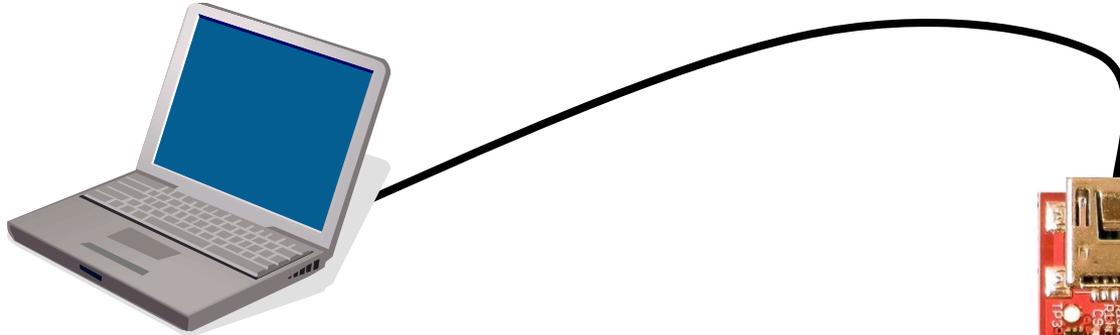
选择一款 MSP430 ADC

	通道数	f_{SAMPLE} (kps)		分辨率	SINAD (典型值)	A_{IN}	基准			触发	增益	特性
		最小值	最大值				Ref _{IN}	Ref _{OUT}	Ref _{I_OUT}			
ADC10	8	34	200+	10	57	V _{SS} 至 V _{REF}	1.4-3.6	1.5/2.5V	+/-1mA	SW/定时器/计数器	N/A	DTC
ADC12	12	34	200+	12	68	V _{SS} 至 V _{REF}	1.4-3.6	1.5/2.5V	+/-1mA	SW/定时器/计数器	N/A	转换存储器
SD16	3个独立	约 4		16	85	+/-600mV	1.0-1.5	1.2V	+/-1mA	SW/计数器	至32倍	预加载
SD16 A	4个复用	约 0.03	约 5	16	85	+/-600mV	1.0-1.5	1.2V	+/-1mA	SW/计数器	至32倍	缓冲输入

- ◆ 需要测量的电压范围是多少？
- ◆ 针对 A_{IN} 的最大频率是多少？
- ◆ 分辨率是多少？
- ◆ 是否采用差分输入？
- ◆ 基准范围是多少？
- ◆ 是否具有多个通道？



Lab6: 应用ADC10



Lab6:

- 使用ADC10内部温度传感器测量温度
- CCS 新特性

Lab 6:

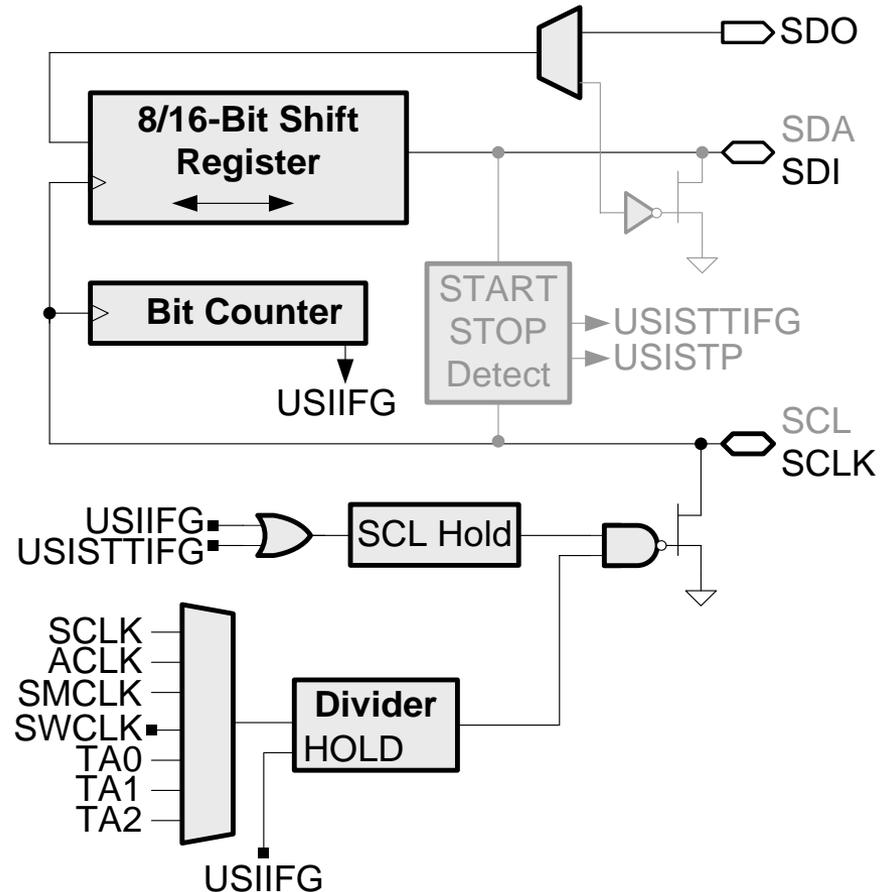
```
//Configure ADC10
// Choose ADC Channel as Temp Sensor
→ ADC10CTL1 = _____ + ADC10DIV_3;
//Choose ADC Ref sourceCCTL1
→ ADC10CTL0 = _____ + ADC10SHT_3 + REFON + ADC10ON + ADC10IE;
```

议程

- 介绍 Value Line 系列
- Code Composer Studio
- CPU 与基本时钟模块
- 中断与 GPIO
- Timer_A 与 WDT+
- MSP430低功耗设计
- ADC10 和 Comparator_A+
- 串行通信模块
- Grace
- 电容式触摸按键解决方案

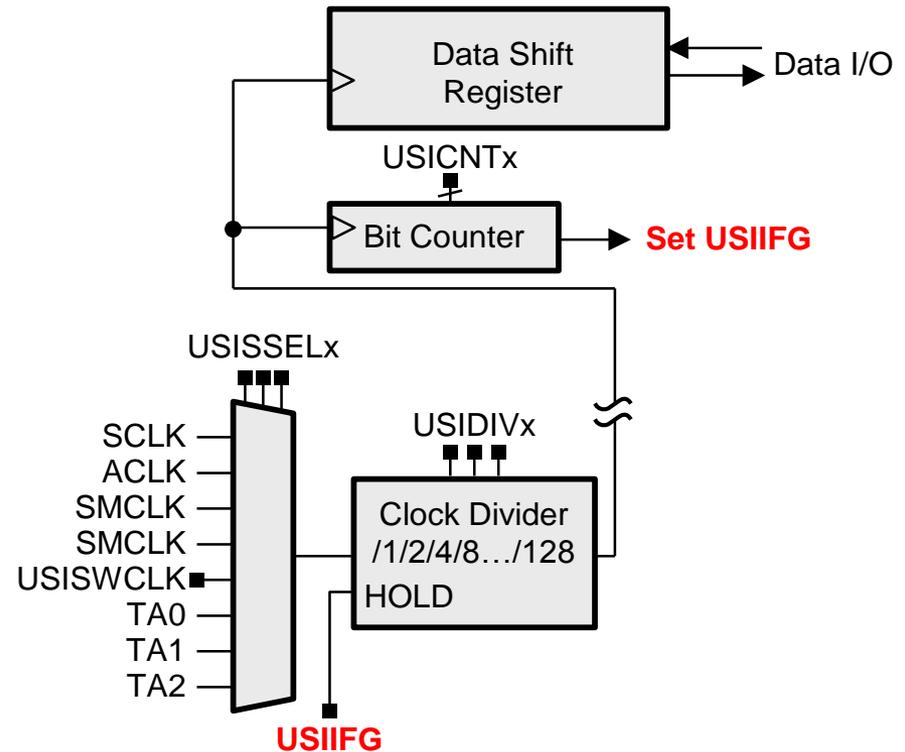
USI

- ◆ **MSP430G2xx1/2 器件**
- ◆ **可变长度移位寄存器**
- ◆ **支持 I2C**
 - ◆ **START/STOP 检测**
 - ◆ **SCL 在 START 之后保持**
 - ◆ **SCL 在计数器溢出之后保持**
 - ◆ **丢失检测仲裁机制**
- ◆ **支持 SPI**
 - ◆ **8/16 位移位寄存器**
 - ◆ **MSB或LSB 优先可选**
- ◆ **灵活的定时**
- ◆ **中断驱动**

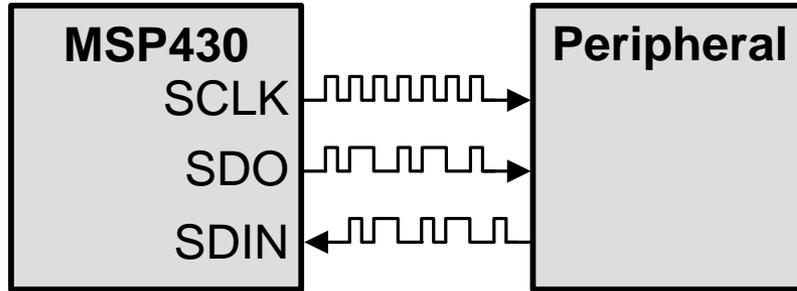


USI的data IO

- ◆ 数据移位寄存器：可支持多达**16**位
- ◆ 发送与接收的位数受控于一个位计数器
- ◆ 发送与接收是同时进行的
- ◆ 数据 I/O 由用户定义：**MSB** 或 **LSB** 优先
- ◆ 位计数器在最后一个位之后自动停止计时并设定标记
- ◆ 无需数据缓冲



USI 減低了 SPI 的 CPU 负荷



```
//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 个周期

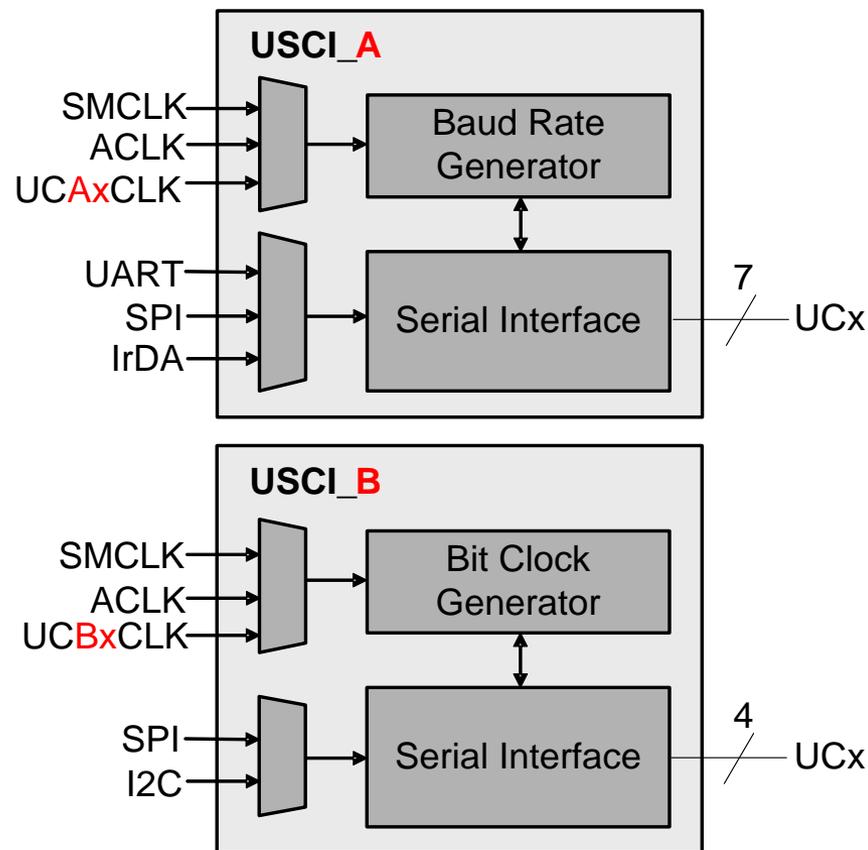
```
// Shift16_inout_USI
USISR |= DATA;
USICNT |= 0x10;
```

10 个周期

- ◆ **I2C Slave**从时钟边缘至数据寄存器的时间短至 **4us**
- ◆ 若使用纯软件解决方案，**MCU**几乎无法完成其他操作
- ◆ **USI** 硬件可实现实用和兼容的 **I2C**
- ◆ 代码可通过 **MSP430** 的相关网址获得

USCI

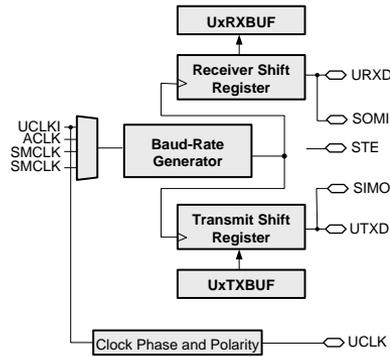
- ◆ 专为超低功耗而精心设计：
 - ◆ 可从任意低功耗模式自动启动
- ◆ 两个专用模块：
 - ◆ **USCI_A:**
UART 或 SPI
 - ◆ **USCI_B:**
SPI 或 I2C
- ◆ 双缓冲发送 (TX) / 接收 (RX)
- ◆ 波特率/位时钟发生器：
 - ◆ 自动波特率检测
 - ◆ 灵活的时钟脉冲源
- ◆ 接收 (RX) 干扰抑制
- ◆ DMA 被启用
- ◆ 误差检测



推荐的 USCI 初始化/重构过程在您的工作手册中进行了说明。

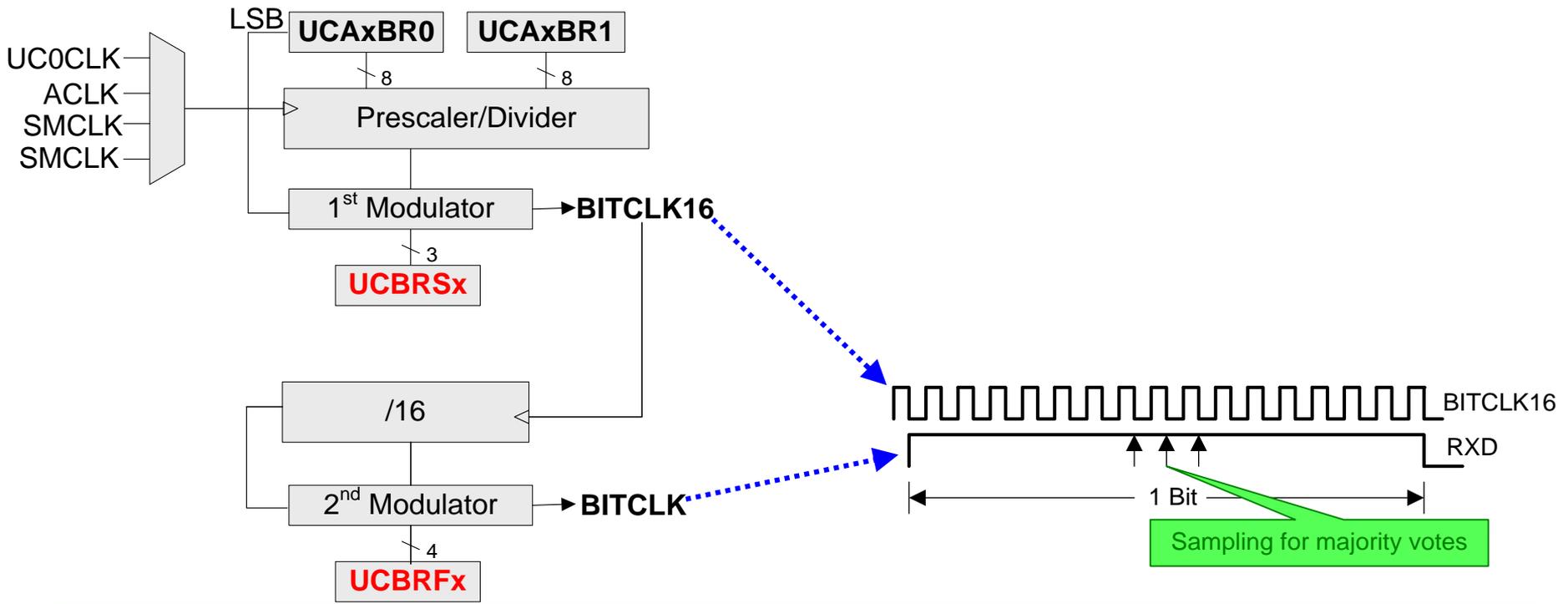
USCI 增强型特性

- ◆ 最新标准 **MSP430** 串行接口
- ◆ 可从任意低功耗模式 (**LPM**) 实现自动时钟启动
- ◆ 两个独立的通信模块
- ◆ 异步通信模式
 - ◆ **UART** 标准和多处理器协议
 - ◆ 具有自动波特率检测功能的 **UART** (支持 **LIN**)
 - ◆ 两个调制器支持 **n/16** 位定时
 - ◆ **IrDA** 位整形编码器与解码器
- ◆ 同步通信模式
 - ◆ **SPI** (主控器和从动器模式, 三线式和四线式)
 - ◆ **I2C** (主控器和从动器模式)



USCI 波特率发生器

- ◆ 过采样波特率发生
- ◆ 两个调制器
 - ◆ **UCBRSx** 和 **UCBRFx**
- ◆ 采用 **BITCLK16** 进行 RX 采样



Value Line 通信模块

USCI

USI

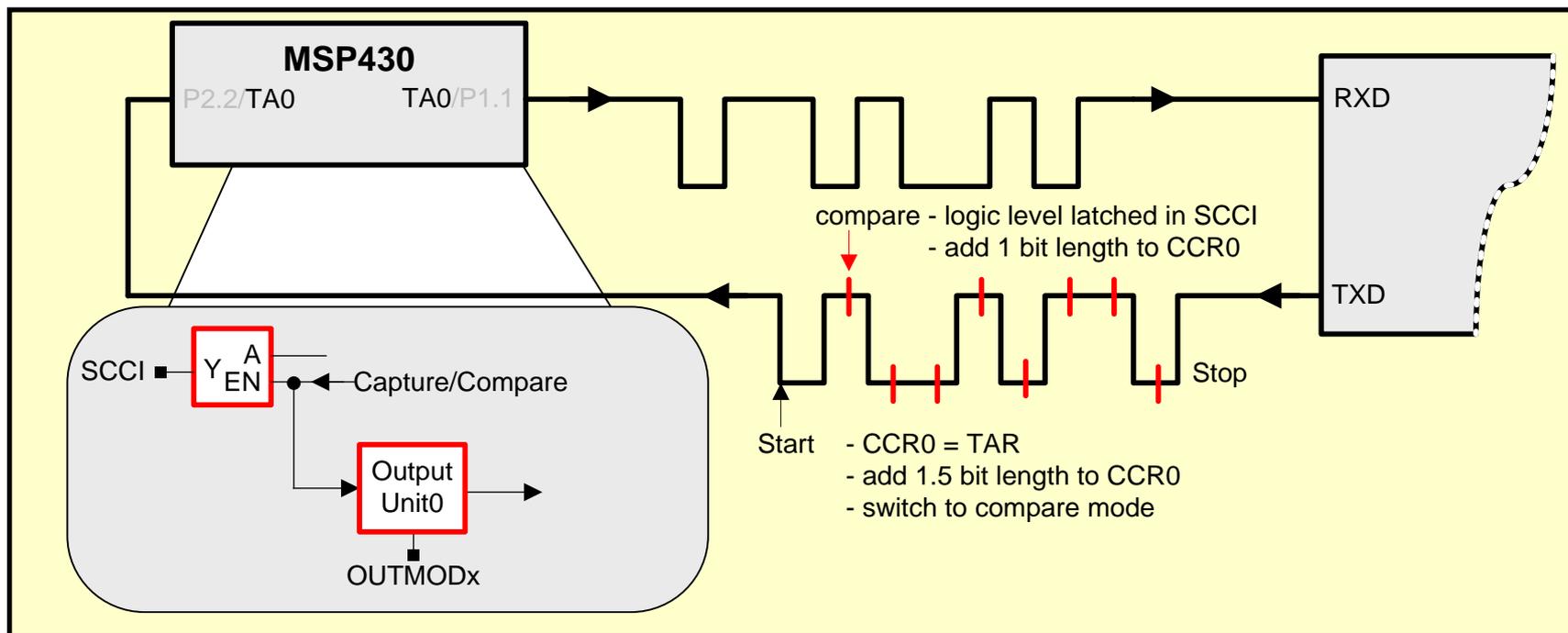
UART

SPI

I2C

	通用串行通信接口 G2xx3	通用串行接口 G2xx1/2
UART	两个调制器；支持 n/16 定时 - 自动波特率检测 - IrDA 编码器与解码器 - 同时 USCI_A 和 USCI_B（两通道）	---
SPI	两个 SPI（USCI_A 和 USCI_B 上各一个） - 支持 Master 和 Slave 模式 - 三线和四线模式	- 可提供一个 SPI - 支持 Master 和 Slave 模式
I2C	- 简化的中断用法 - 支持 Master 和 Slave 模式 - 高达 400kbps	- 需要软件 (SW) 状态机 - 支持 Master 和 Slave 模式

低开销 UART 实现方案

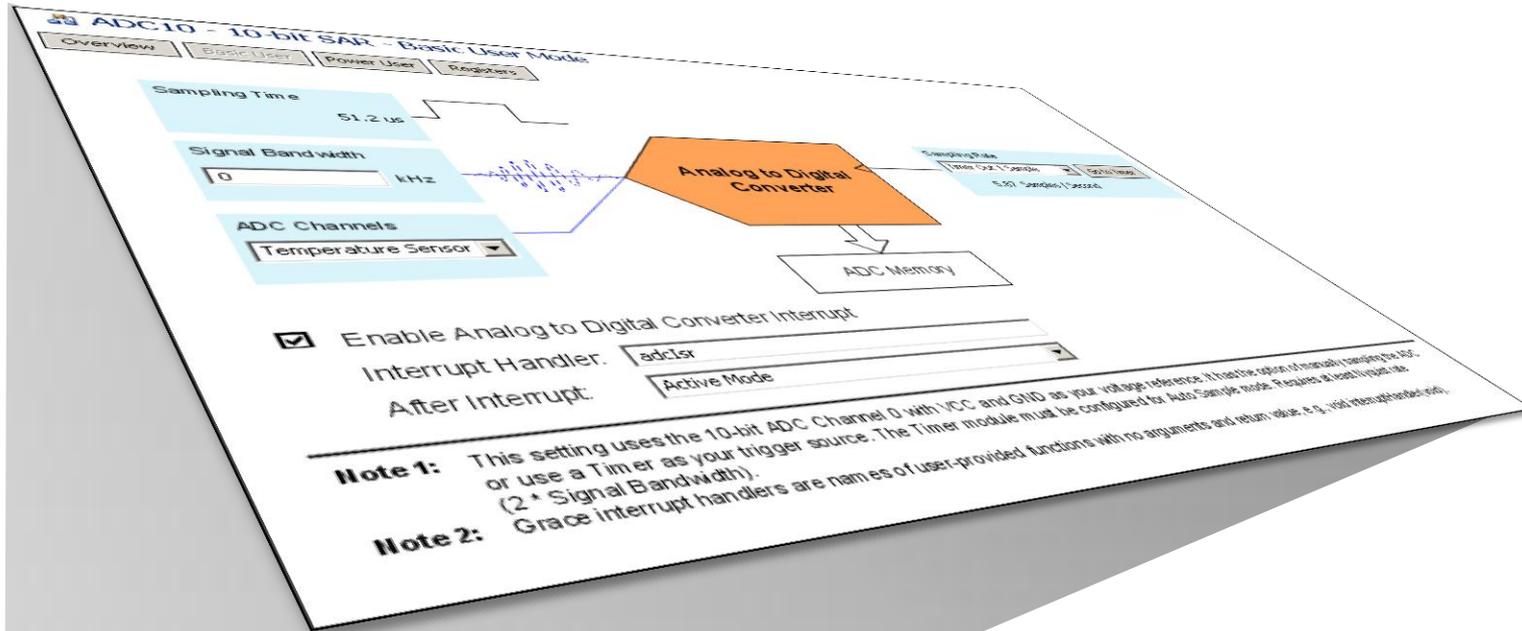


- ◆ 100% 硬件实现锁存与输出
- ◆ 依靠 LPM3 和 LPM4 实现了全速性能
- ◆ 低 CPU 开销
- ◆ 应用笔记 SLAA078 可在网上查阅

议程

- 介绍 Value Line 系列
- Code Composer Studio
- CPU 与基本时钟模块
- 中断与 GPIO
- Timer_A 与 WDT+
- MSP430低功耗设计
- ADC10 和 Comparator_A+
- 串行通信模块
- **Grace**
- 电容式触摸按键解决方案

Grace™



Grace™

一款免费的图形用户界面，可生成源代码，免除手动配置外设

Grace 是什么？

Grace – Graphical Code Engine（图形代码引擎）

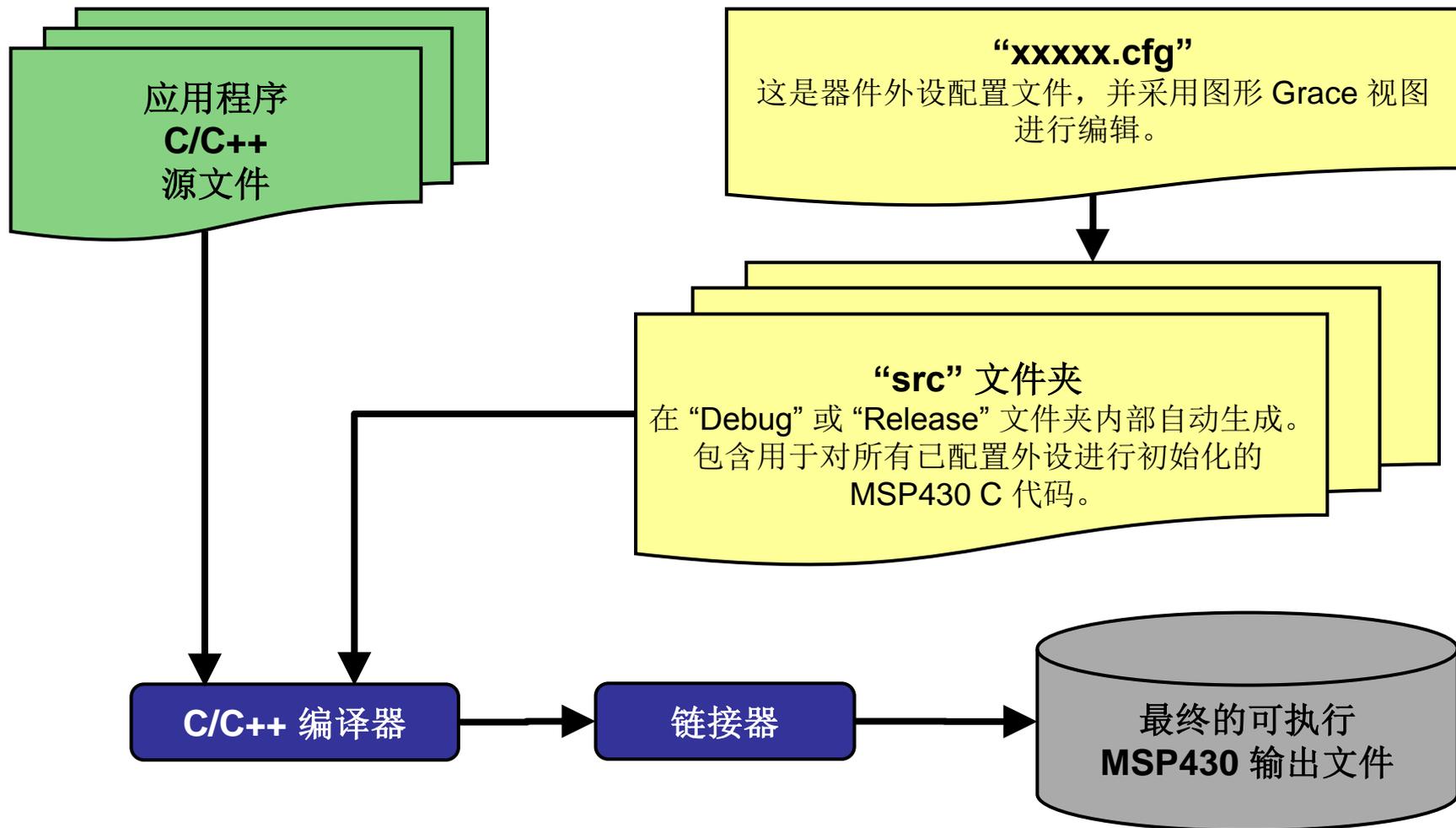
是：

- 可利用 CCS 生成 MSP430 外设初始化代码
- 使新用户能在 15 分钟内运行程序
- 着重专注于易用性
- 将进一步扩展以覆盖所有的 MCU 器件

不是：

- 图形应用程序编制器

项目结构与构建流程



用户代码框架示例

```
/*
 * ===== Standard MSP430 includes =====
 */
#include <msp430.h>

/*
 * ===== Grace related includes =====
 */
#include <ti/mcu/msp430/csl/CSL.h>

/*
 * ===== main =====
 */
int main(int argc, char *argv[])
{
    // Activate Grace-generated configuration
    CSL_init();

    __enable_interrupt(); // Set GIE
    // >>>> Fill-in user code here <<<<<

    return (0);
}
```

430通用头文件

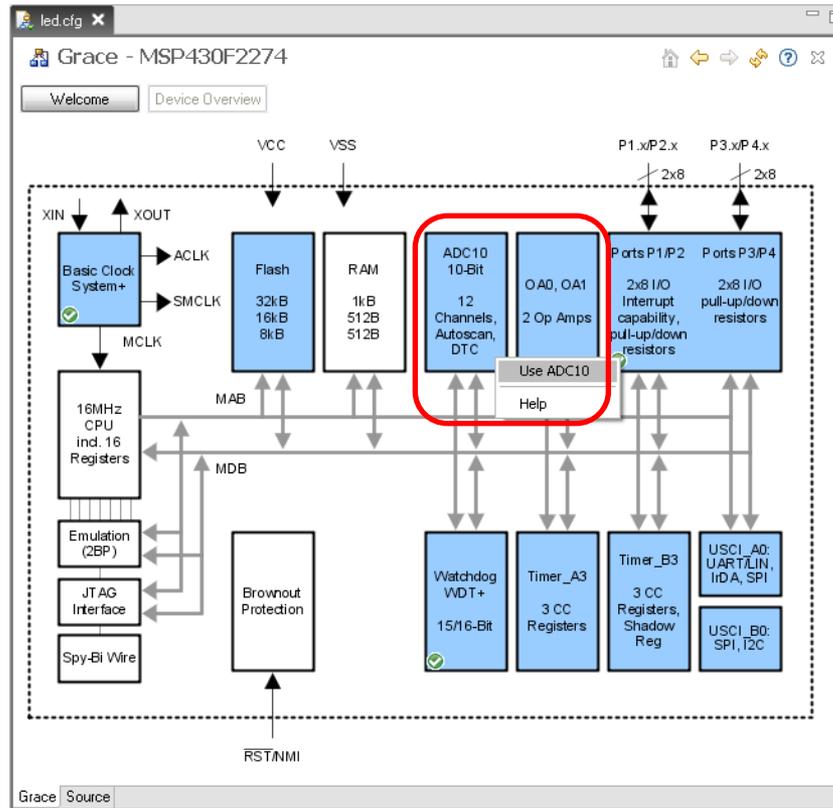
主控器包括用于所有 Grace
关联内容的文件

执行所有由 Grace 配置的外
设设置

用户代码从这里开始...

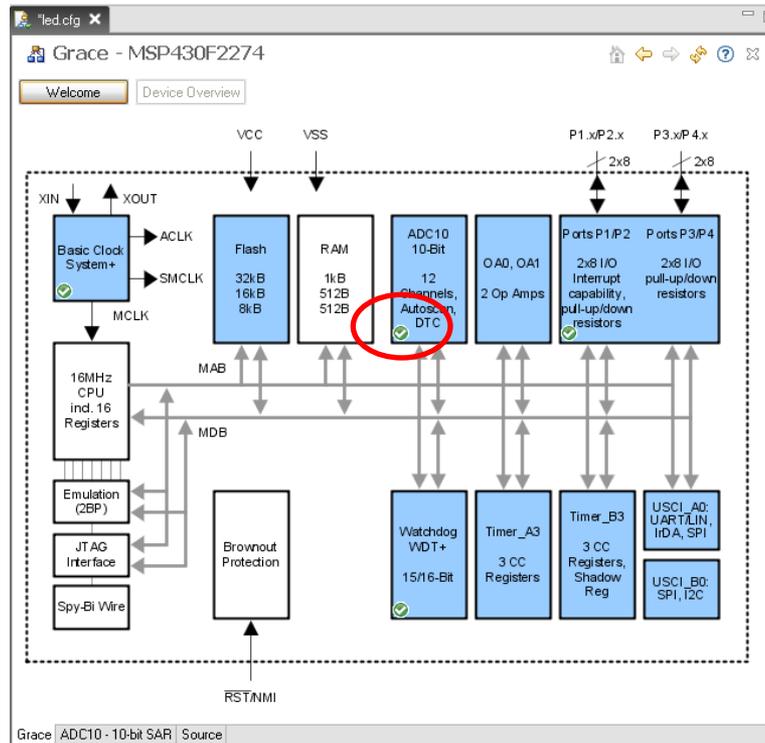
Grace – 添加一个外设

- 在该外设上单击右键并选择“使用 (Use)”
- 所有着蓝色的模块均可配置



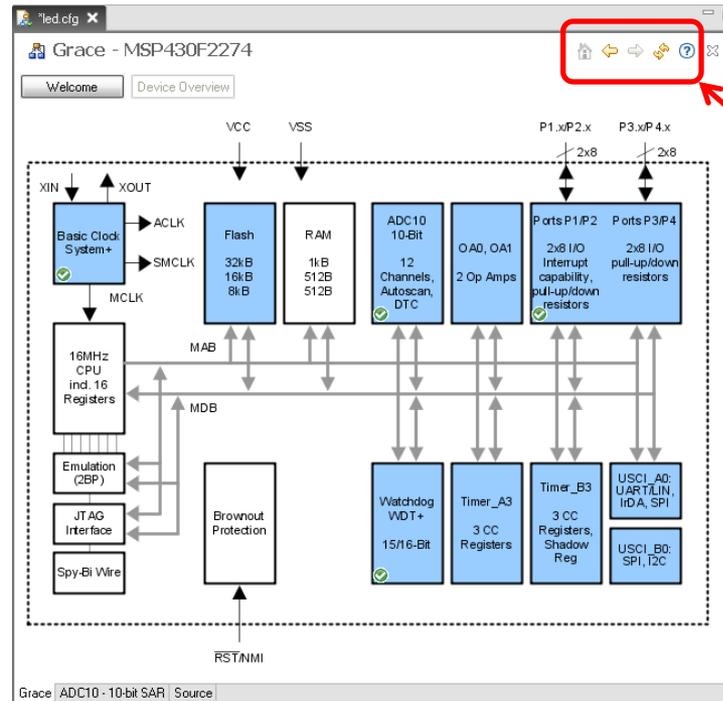
怎样告知添加了一个外设？

- 在 CSL 视图上观察该外设的左下角，如果此外设被初始化，则将显示一个绿色的钩型标记。



Grace – 导航

- 在某个外设上单击左键以导航至其细节视图
- 使用主画面按钮以返回至顶层器件视图
- 还可使用前进/后退按钮



Grace – 配置一个外设

- 每个外设具有 4 种不同的表示：

“Overview”（概要）

“Basic User”（基本用户）

“Power User”（高级用户）

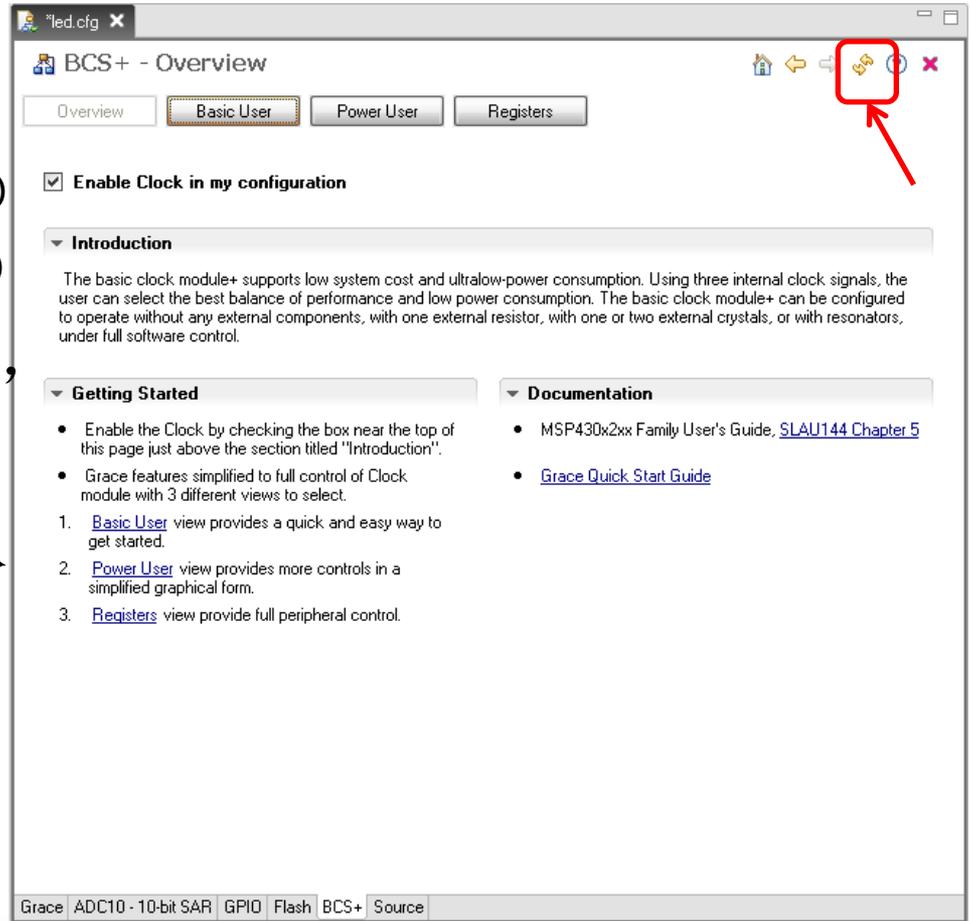
“Registers”（寄存器设置）

- 您可以编辑其中的任一个，

它们是完全连通的

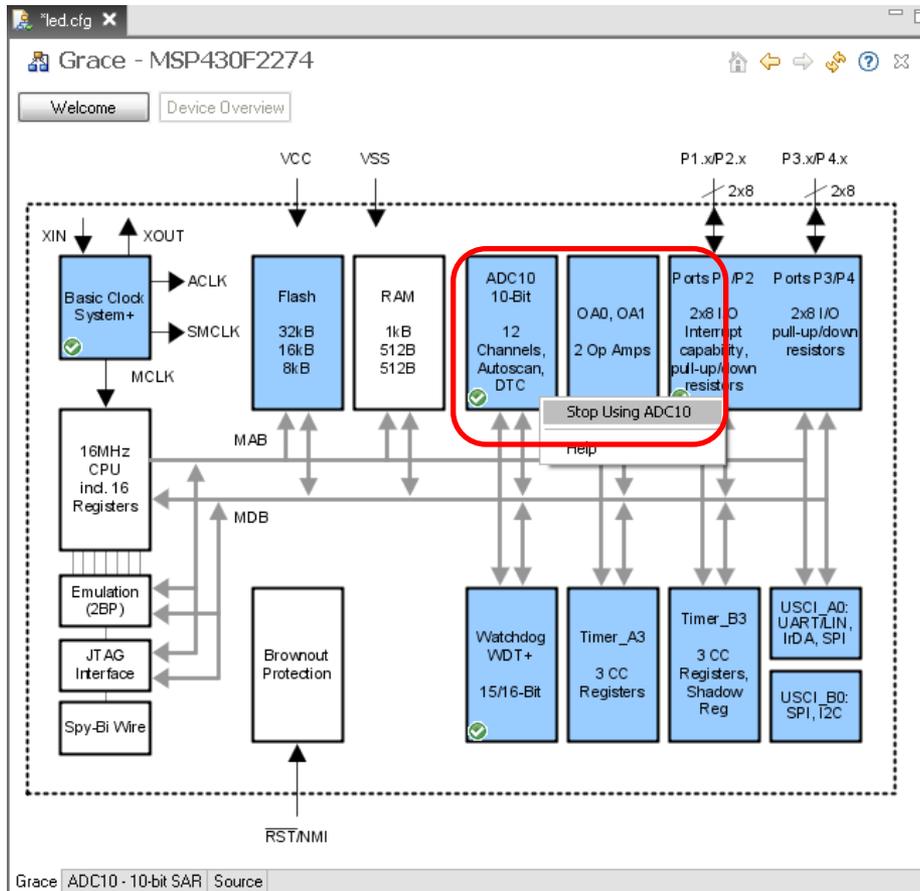
- 通过点击“Refresh”来确认

当前的配置

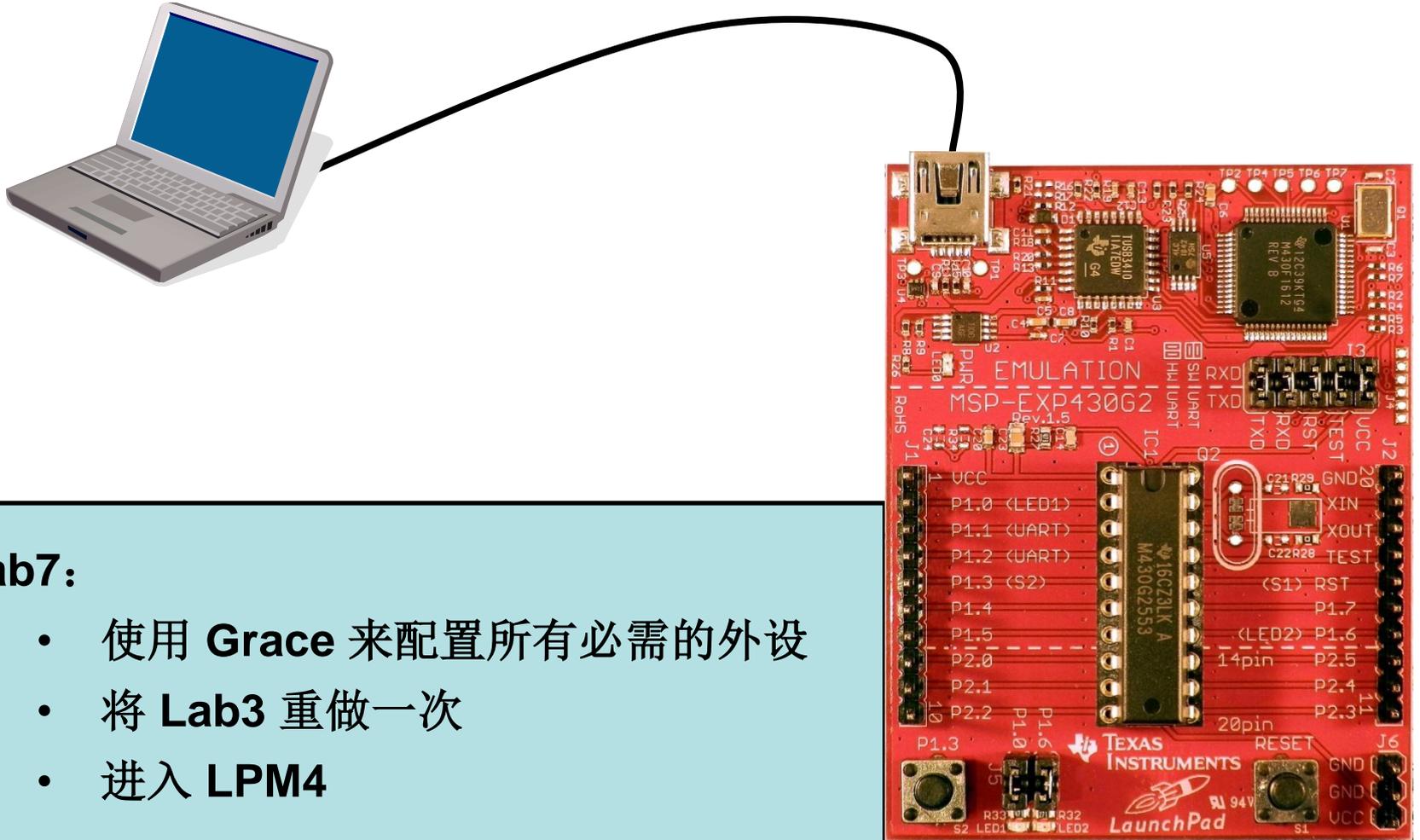


Grace – 移除一个外设

- 在该外设上单击右键并选择“Stop Using”（停止使用）



Lab7: Grace



Lab7:

- 使用 **Grace** 来配置所有必需的外设
- 将 **Lab3** 重做一次
- 进入 **LPM4**

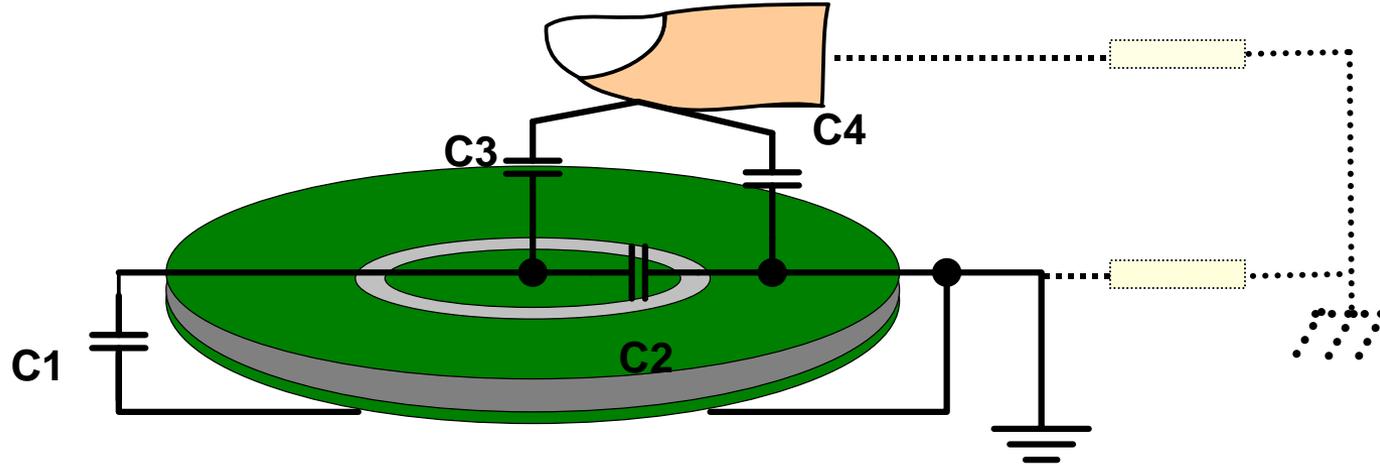
Lab 7: 按步操作

- 停用看门狗定时器
- 配置 DCO 以依靠预校准的内部 1MHz 运行
- 设置 LaunchPad 用于中断操作的 S2(P1.3) 按钮
- 进入 main() 功能中的 LPM4
- 提供一个按钮中断处理程序，用于清除 IFG 并在恢复时立即唤醒 MCU
- 红色 LED(P1.0)在 main() 中的开或者关
- 完成操作后在 main() 中再次进入LPM4

议程

- 介绍 Value Line 系列
- Code Composer Studio
- CPU 与基本时钟模块
- 中断与 GPIO
- Timer_A 与 WDT+
- MSP430低功耗设计
- ADC10 和 Comparator_A+
- 串行通信模块
- Grace
- 电容式触摸按键解决方案

什么是电容式触摸？



电容的变化...

- ◆ 当存在一个导电元件时 – 手指或触针
 - 将增加 C3 和 C4，从而导致电容 $C1 + C2 + C3 \parallel C4$ 增大
 - 这变成了至大地的自由空间耦合通路的一部分

MSP430 电容式触摸检测方法

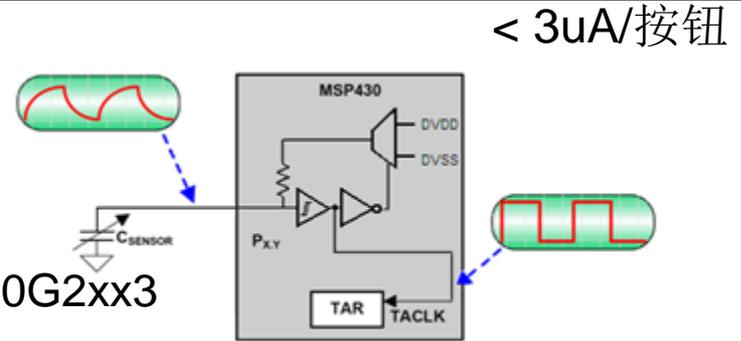
Pin oscillator

(带内部 RO 的 PinOsc)

无需外部组件

使用内部定时器

目前采用此方法的器件是 MSP430G2xx2 和 MSP430G2xx3

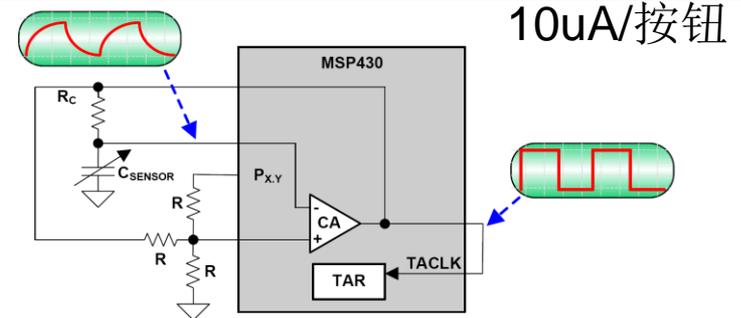


RO

抗干扰能力最强

使用内部定时器、比较器

具有比较器的 MSP430 器件均支持



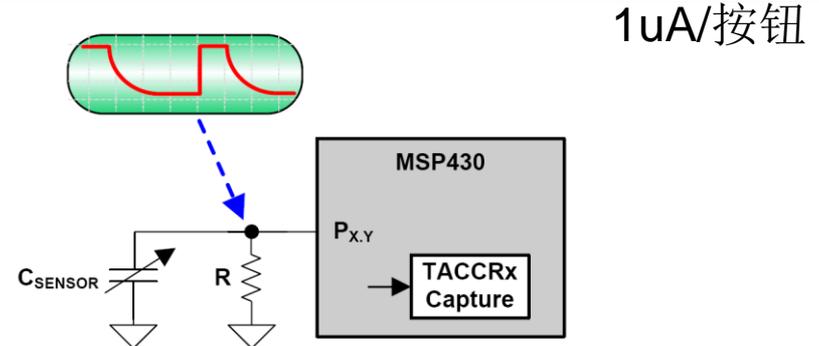
RC

功耗最低的方法

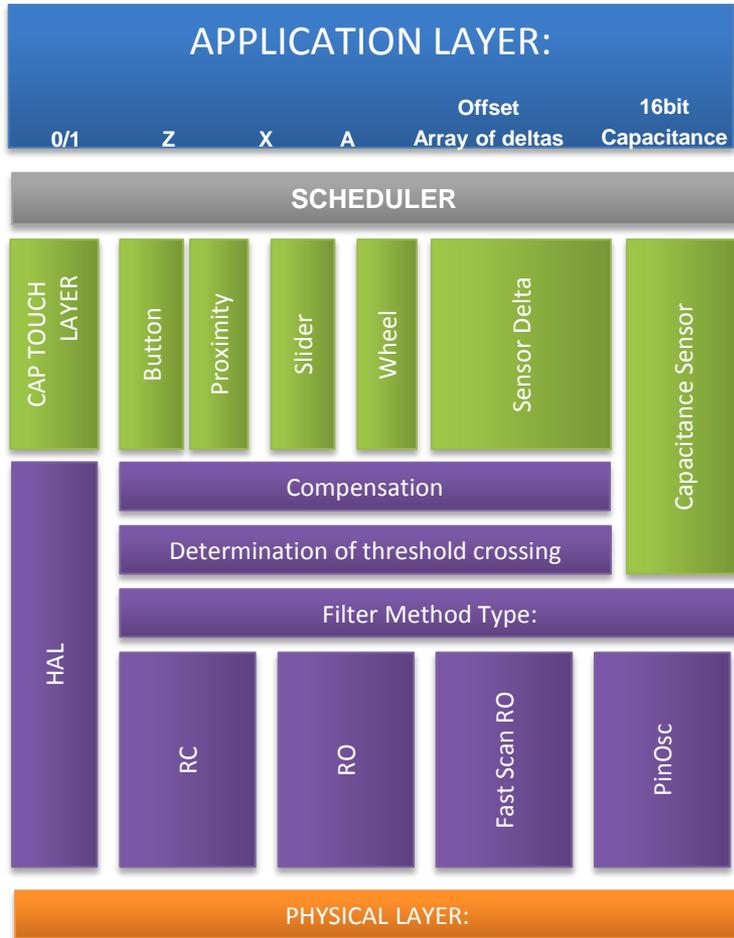
最多可支持 16 个按键

使用 GPIO 和定时器

任何 MSP430 器件均支持



触摸库结构



配置

Schedule:

**Sensor
Peripherals
Period definition**

Sensor:

**Electrodes
Reference
Sensor Type
Measurement Method
Peripherals
Peripheral settings
Measurement Parameters**

Element:

**Port I/O
definitions**

电容式触摸 BoosterPack

器件型号: 430BOOST-SENSE1

- 用于 LaunchPad 的电容式触摸插件
- 触摸按钮、滚轮和接近感应
- 包括一颗具有电容式触摸 I/O 模块的 MSP430G2xx2 芯片
- 有针对滚轮及接近感应的设计范例
- 全面支持电容式触摸感应代码库
- 售价仅 10 美元
- www.ti.com/capacitivetouch

