MSP430 LaunchPad

许岳兵

Email: yuebingxu@126.com

Tel: 15200584128

Agenda

1、Value Line 简介

2、Code Composer Studio

3、CPU及基本时钟系统

4、中断及通用IO

5、定时器及增强型WDT

6、低功耗优化设计

7、ADC10 及 Comparator_A+

8、串行通信

9、Grace

10、电容式触摸按键解决方案

1.1 MSP430系列MCU

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:
- 16-bit timers
- Watchdog Timer
- Internal Digitally Controlled Oscillator
- External 32-kHz crystal support
- <50 nA pin leakage
- <6 µs wakeup

FRAM
Speed 24Mhz
FRAM 4-16kB
GPIO 14-28
Non-volatile memory
PMM, SVS, SVM, LDO, AES, ADC10 (A), USCI, DMA

L092
0.9V-1.65V
Speed 4Mhz
ROM to 2kB
RAM to 2kB
GPIO 11
BOR, DAC8, Comp, SVS, BOR, WDT, A-POOL, ADC8

F1xx
Speed 8Mhz
Flash 1-60kB
RAM to 10kB
GPIO 14-48
BOR, ADC10.12, Comp_A, DAC12, DMA, MPY, SVS, USART

G2xx
Speed 16Mhz
Flash 0.5-16kB
RAM to 256kB
GPIO 10-16
BOR, ADC10, Comp_A+, Temp, USI, Cap Sense I/Os

F2xx
Speed 16Mhz
Flash 1-120kB
RAM to 8kB
GPIO 10-64
BOR, ADC10,12, SD16_A, Comp_A+, DAC12, DMA, MPY, OpAmp, SVS, USCI, USI

F4xx
Speed 8/16Mhz
Flash 4-120kB
RAM to 8k
GPIO 14-80
BOR, LCD, ADC10,12, SD16(_A), Comp_A, DAC12, DMA, MPY, OpAmp, SVS, USART, USCI, ESP430, SCAN_IF, Basic Timer, WDT+, RTC_C

F5xx
Speed 25Mhz
Flash 8-256kB
512kB coming soon.
RAM to 18kB
GPIO 32-83
PMM, BOR, SVS, SVM, LDO, MPY, USCI, DMA, EDI, USB, ADC10,12 (A), Comp_B, RTC_A/B, WDT, Cap Sense I/Os

CC430
Speed 20Mhz
Flash 8-32kB
RAM to 4kB
GPIO 40
PMM, BOR, SVS, SVM, LDO, MPY, USCI, DMA, Sub 1GHz RF, AES, ADC12 (A), Comp_B, RTC_A/B, LCD

MSP430 MCUs
An Introduction

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

丰富齐全的产品线，低成本

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！

1.2 MSP430支持的应用

公用事业计量

电能表
燃气表
流量表
智能计量仪器

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。

便携式医疗

血糖计
温度计
心率监测计
可植入装置

无线应用

远程传感器
通讯控制器
RFID

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

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

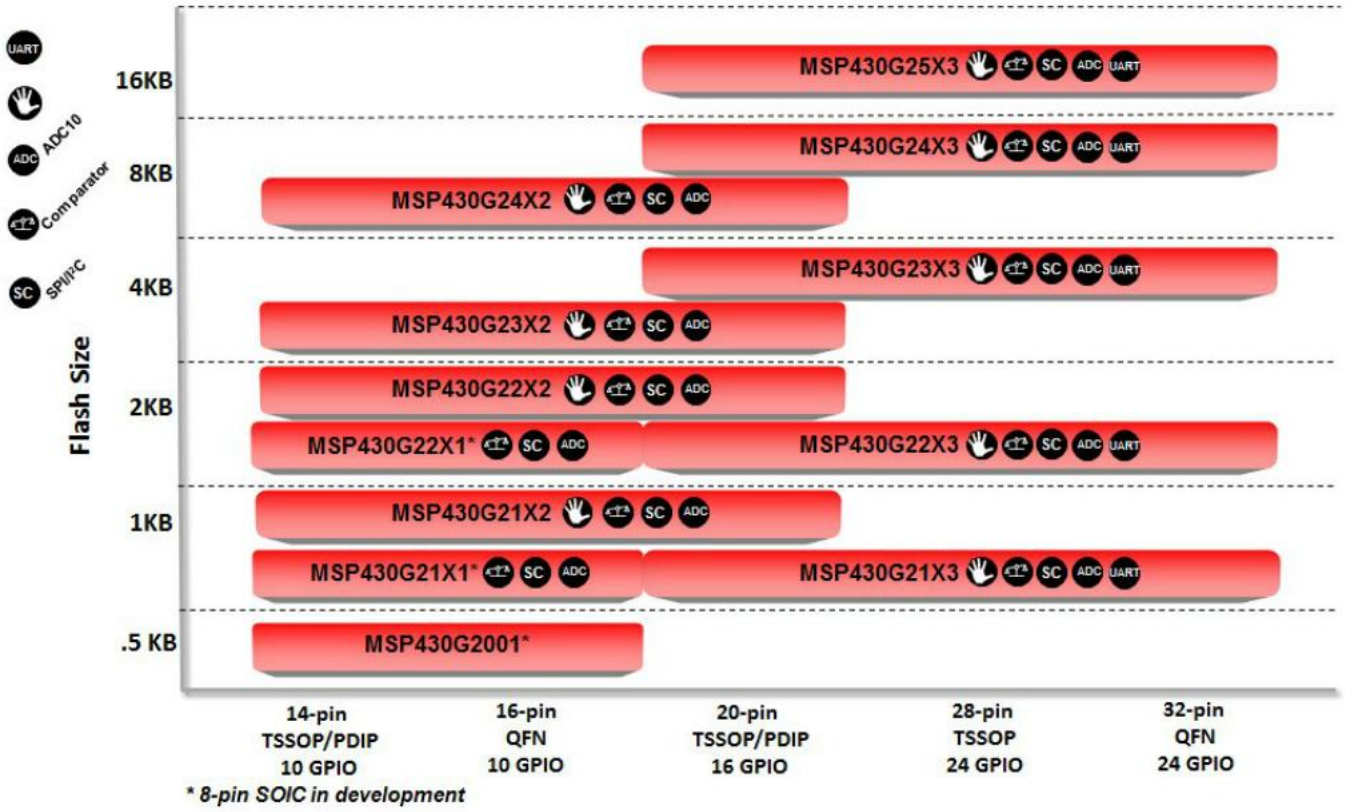
可再生能源
无电池设备
太阳能、热能、振动能等

个人健康与健身

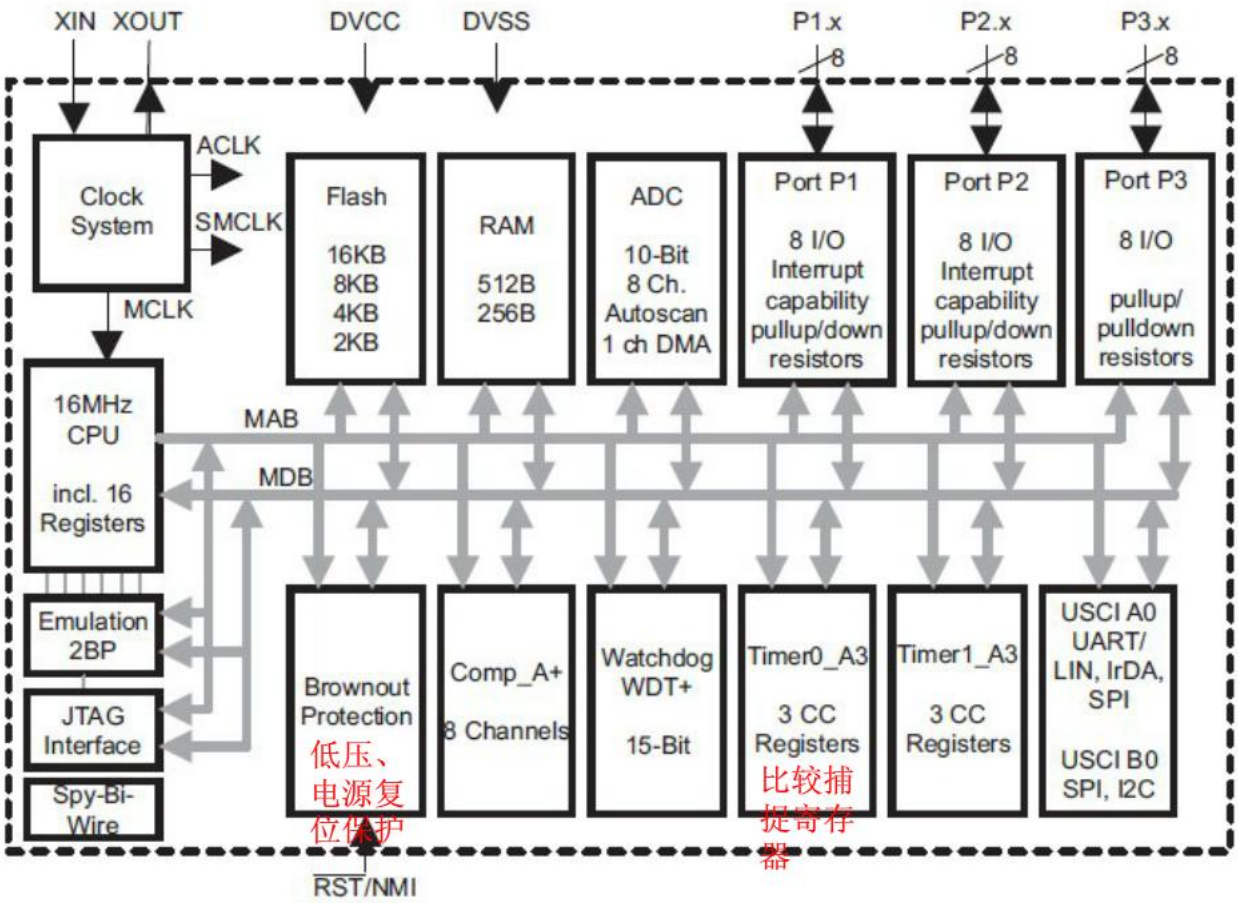
运动手表
计步器
热量计
潜水手表

1.3 Value Line系列

➢ 16-bit performance, 8-bit price



1.4 Value Line功能框图



1.5 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

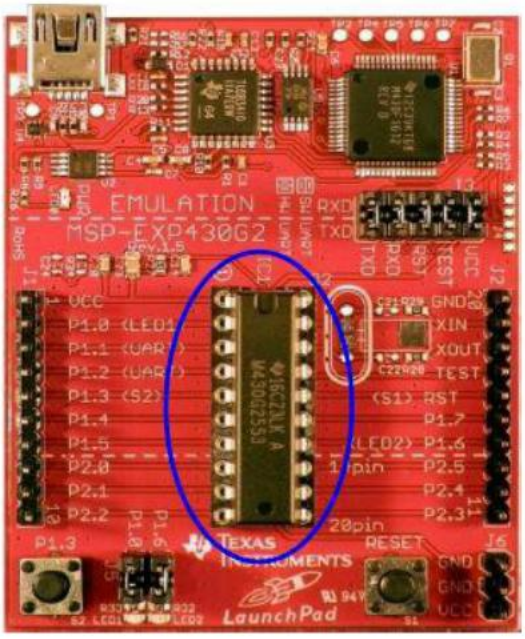
1.5 Value Line外设

- 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
- 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

1.5 Value Line外设

- **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

1.6 MSP-EXP430G2 LaunchPad



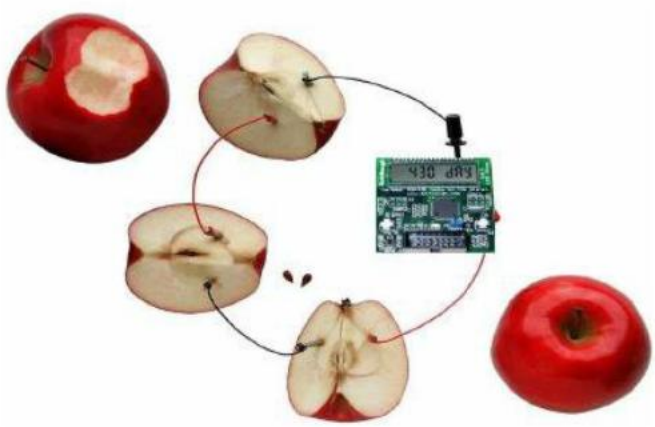
MSP430G2553

Value Line

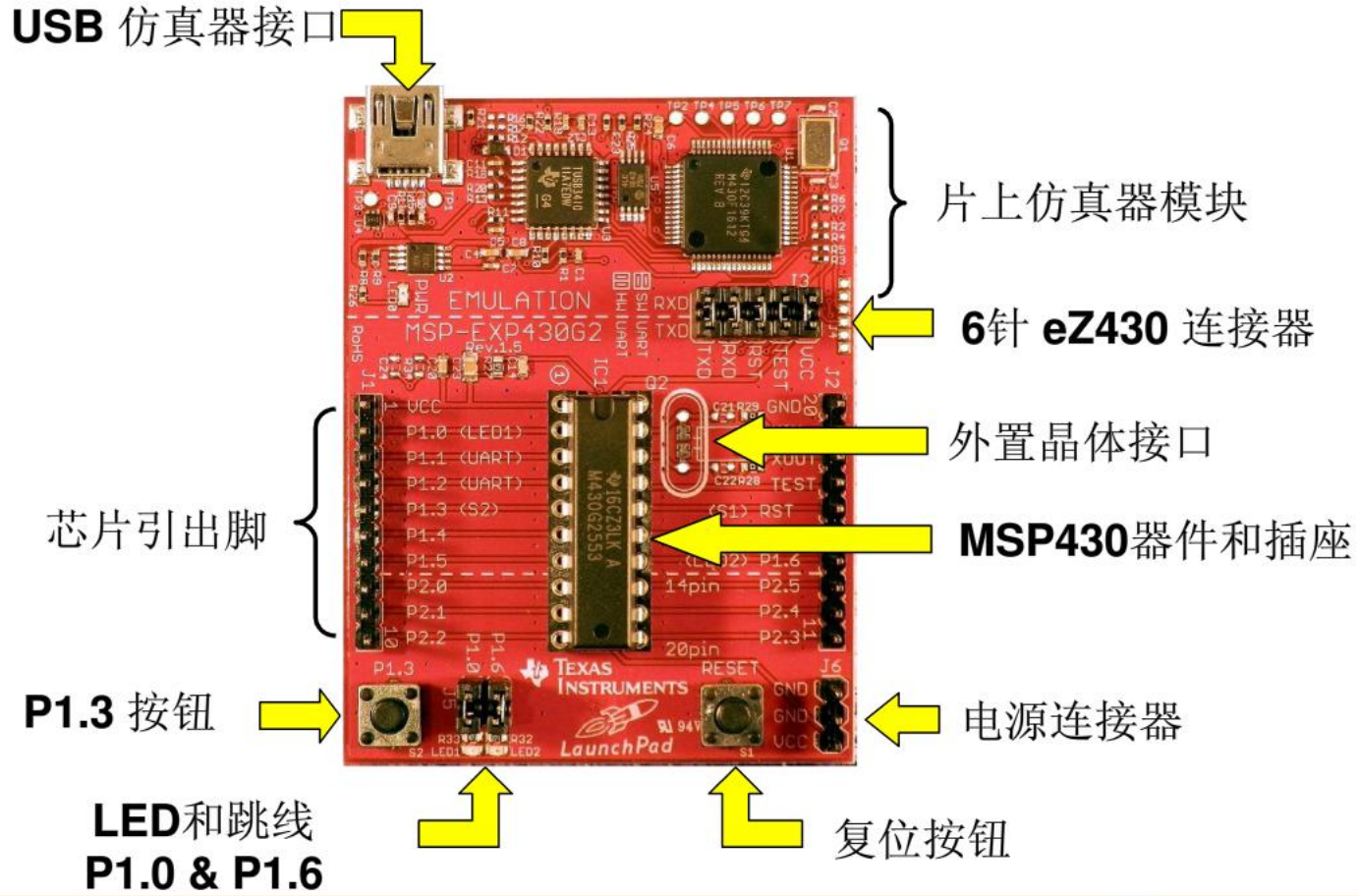
16-bit performance@ 8-bit price

Mixed Signal Processor

Ultra low power



LaunchPad Development Board



Agenda

1、Value Line 简介

2、Code Composer Studio

3、CPU及基本时钟系统

4、中断及通用IO

5、定时器及增强型WDT

6、低功耗优化设计

7、ADC10 及 Comparator_A+

8、串行通信

9、Grace

10、电容式触摸按键解决方案

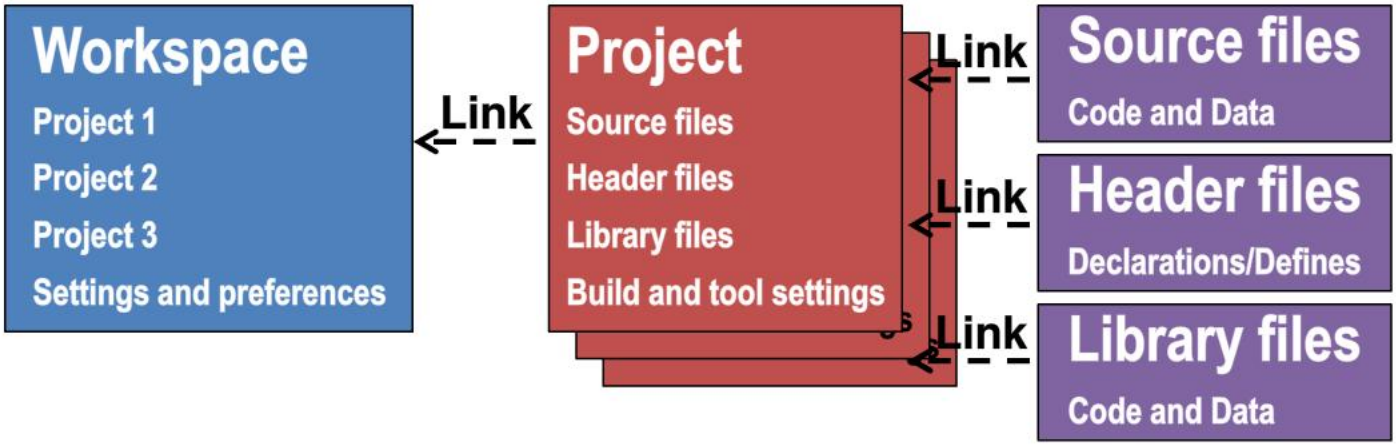
2.1 Code Composer Studio简介

- 用于**TI**嵌入式处理器的集成型开发环境（**IDE**）
  - ✓ 包括 **debugger, compiler, editor, simulator, OS…**
  - ✓ 该环境基于**Eclipse** 开源软件框架
  - ✓ **TI**对其扩展，支持全系列的**TI**嵌入式控制器（**430**，**DSP**，**ARM**，**OMAP……**
- 目前软件版本：**CCS v5**
- 集成更多的工具
  - ✓ 操作系统应用程序开发**(Linux, Android…)**
  - ✓ 代码分析，源控制**…**
- 即将支持**Linux**

2.2 常用任务

- 创建新**Projects**
  - ✓ 提供创建新 **project**的模板，简单易用
- **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
- **Debug**调试
  - ✓ 提供断点，单步等调试工具

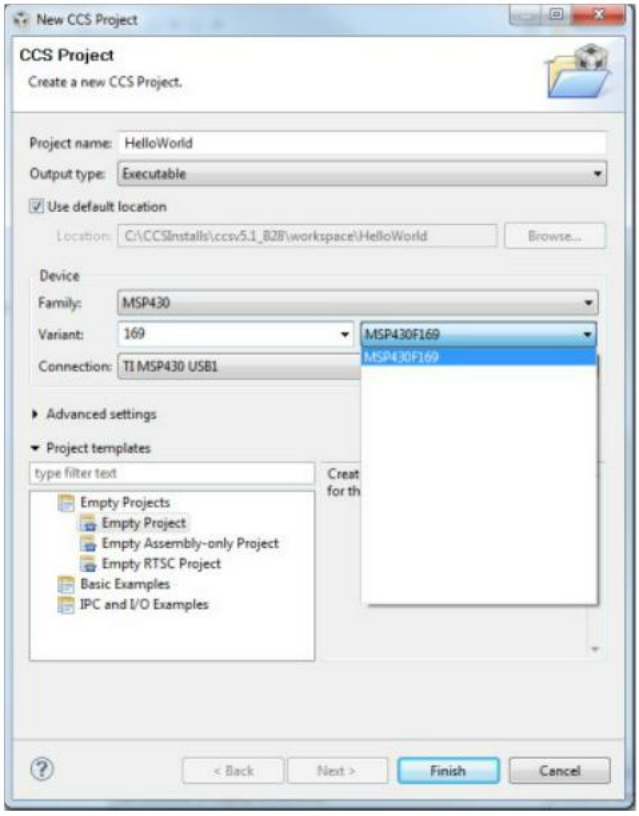
2.3 工作空间与方案



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，而并非设计文件

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，而并非设计文件

2.4 项目向导

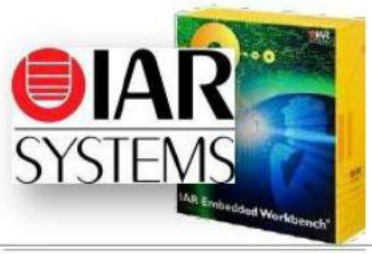


- 单页向导满足大多数情况的使用要求
  - ✓ “下一个”按钮将在某个模板需要附加设置显示
- 包含调试器设置
  - ✓ 假如选取了某款特定的器件，用户还可选择其调试器，并生成 **ccxml** 文件
- 采用默认设置使其简单易用
  - ✓ 一些高级配置编译器版本、字节存储顺序等在 **Advanced Settings** 下

2.5 各种不同的 IDE 选项

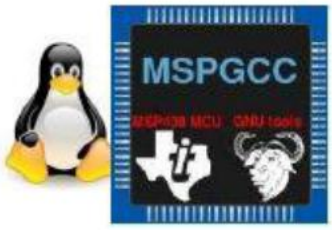
Code Composer Studio

- 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

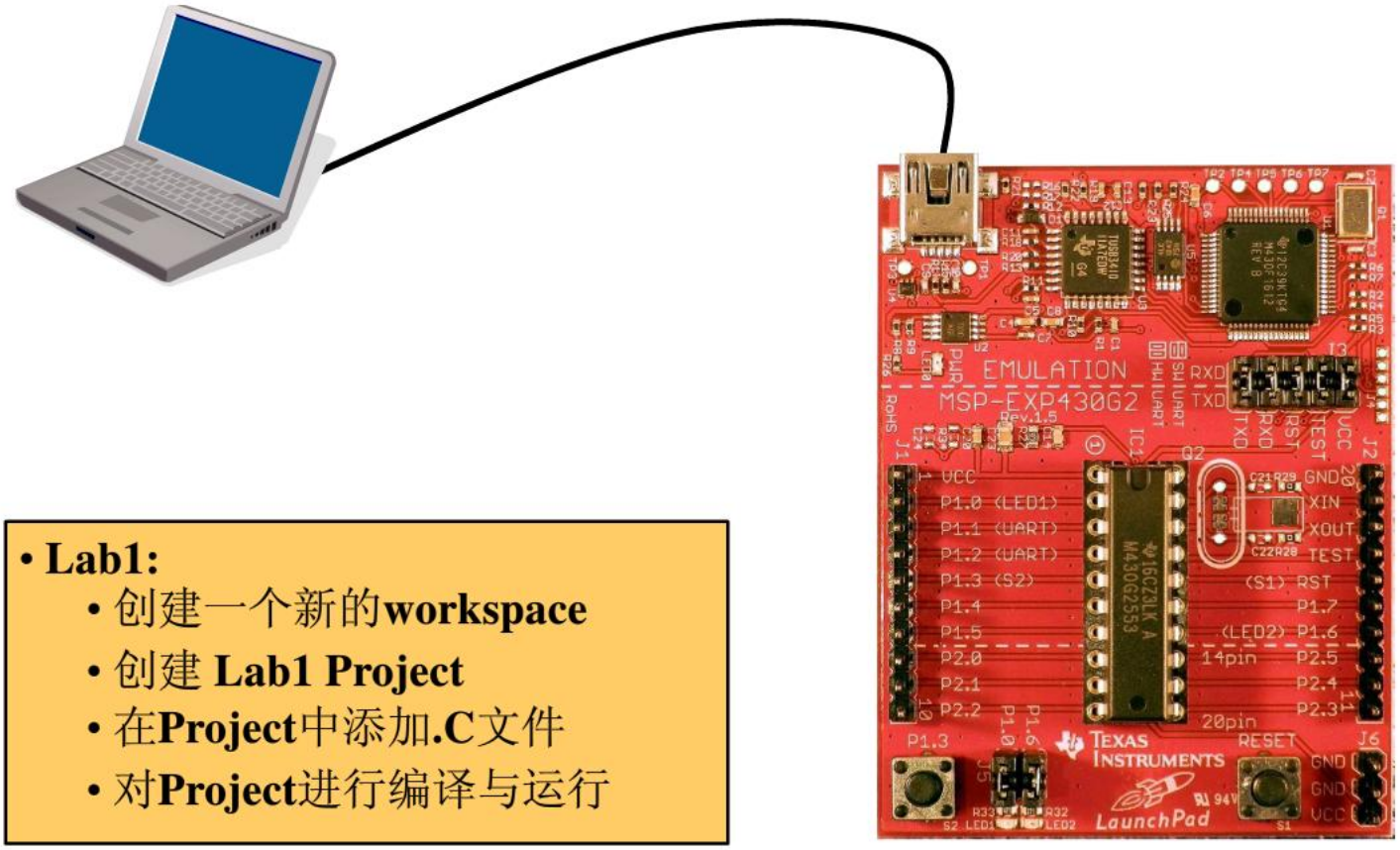


MSPGCC

- 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/

Other MSP430 IDE options are available! Learn more @ www.ti.com/msp430tools

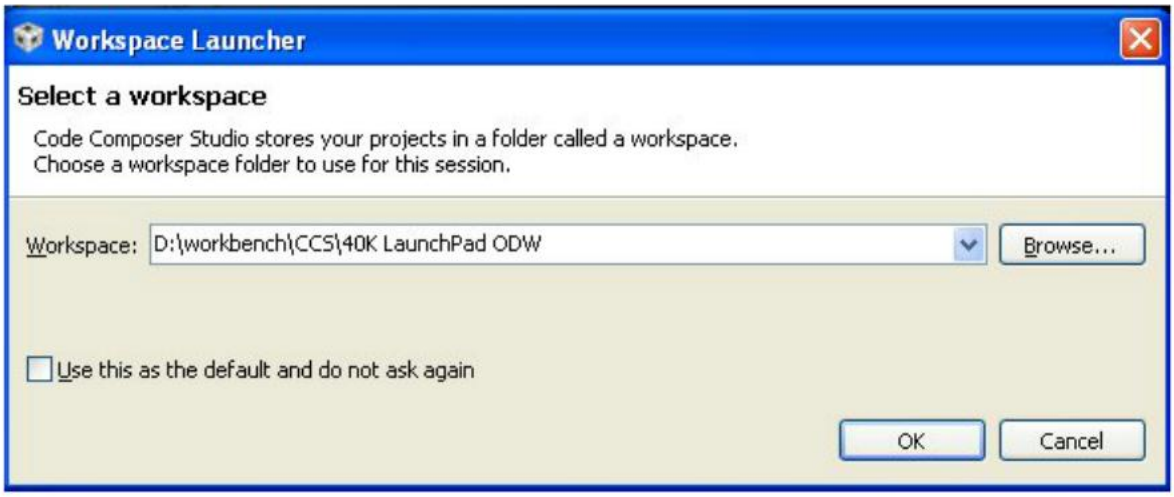
Lab1: Code Composer Studio



- **Lab1:**
  - 创建一个新的**workspace**
  - 创建 **Lab1 Project**
  - 在**Project**中添加**.C**文件
  - 对**Project**进行编译与运行

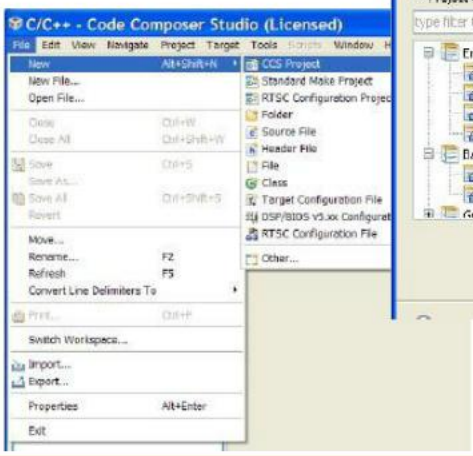
Step 1: 新建 CCS workspace

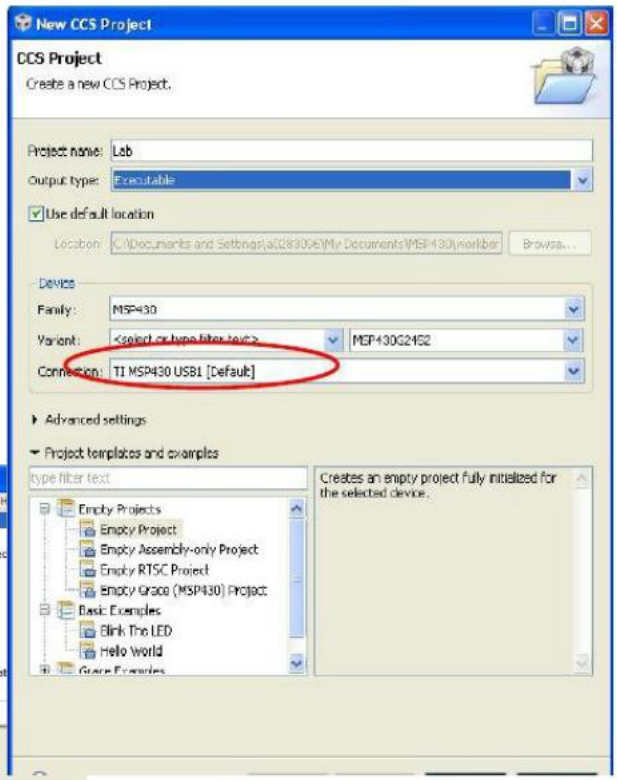
- 双击图标，启动 **CCS v5**
- 若未选择默认路径，设置 **—Workspace‖** 路径



Step 2: 创建 a CCS Project

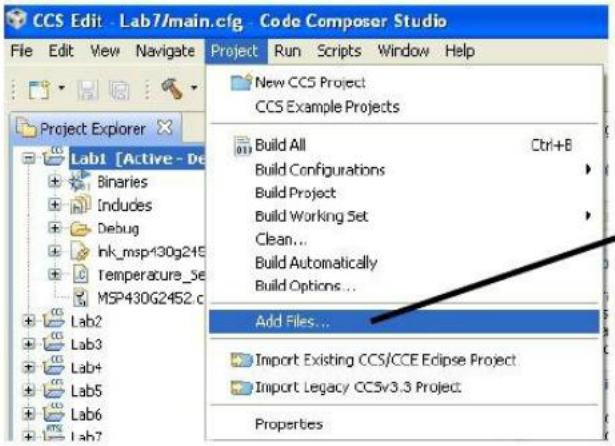
- File > New > CCS Project
- Project 名称: Lab1
- Device>Family: MSP430
- Variant: MSP430G2553
- Project templates and examples
- : Empty Project

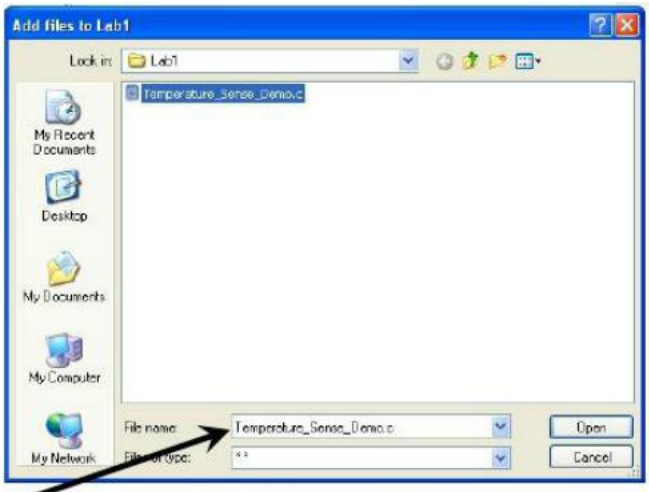


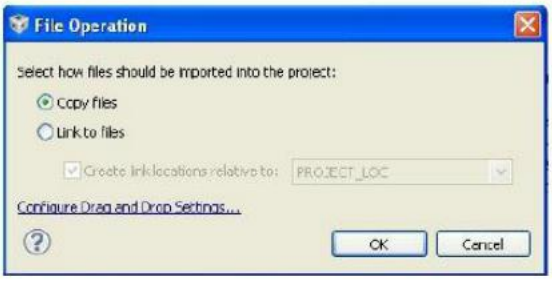


Step 3: 在CCS Project中添加文件

- **Project > Add Files**
- **Navigate to Lab source folder**
- **And select :**
- **Temperature_Sense_Demo.c**

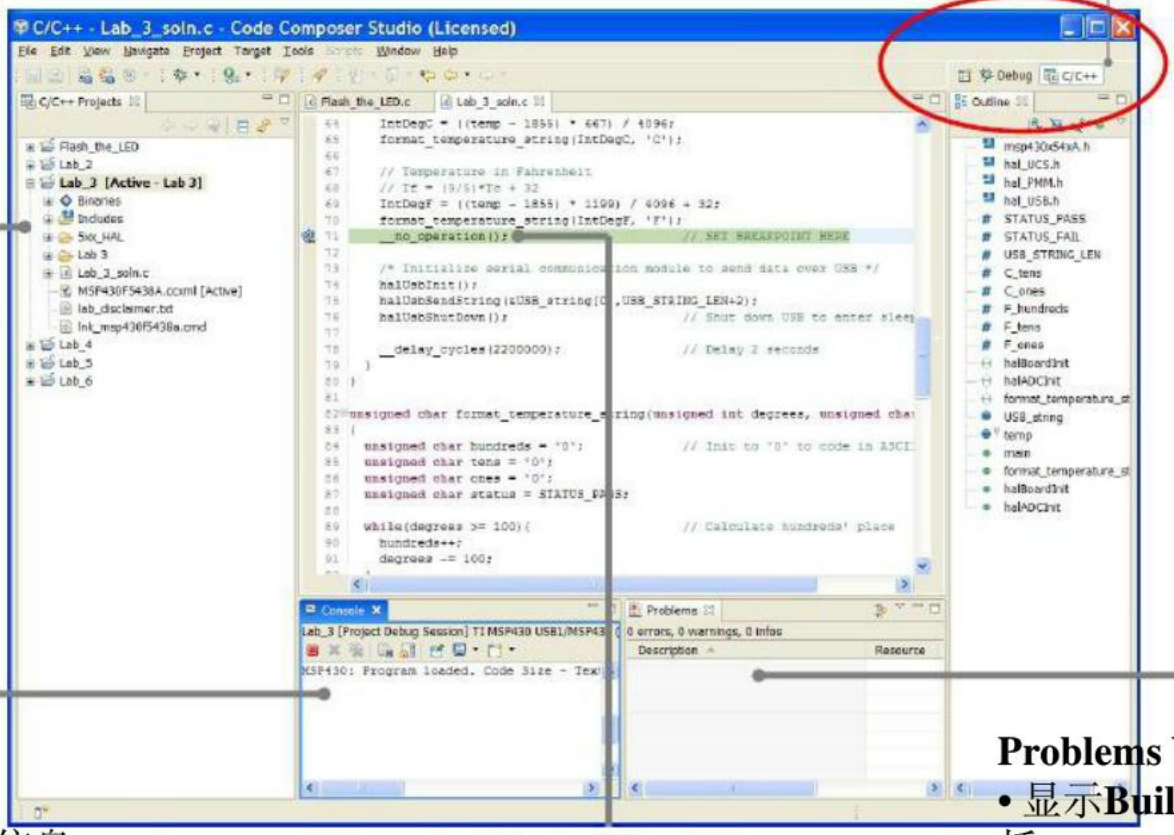






CCS 界面 – C/C++ Perspective Overview

C/C++ 编辑界面



显示当前所有**Projects**

Console

- 显示**Build**信息

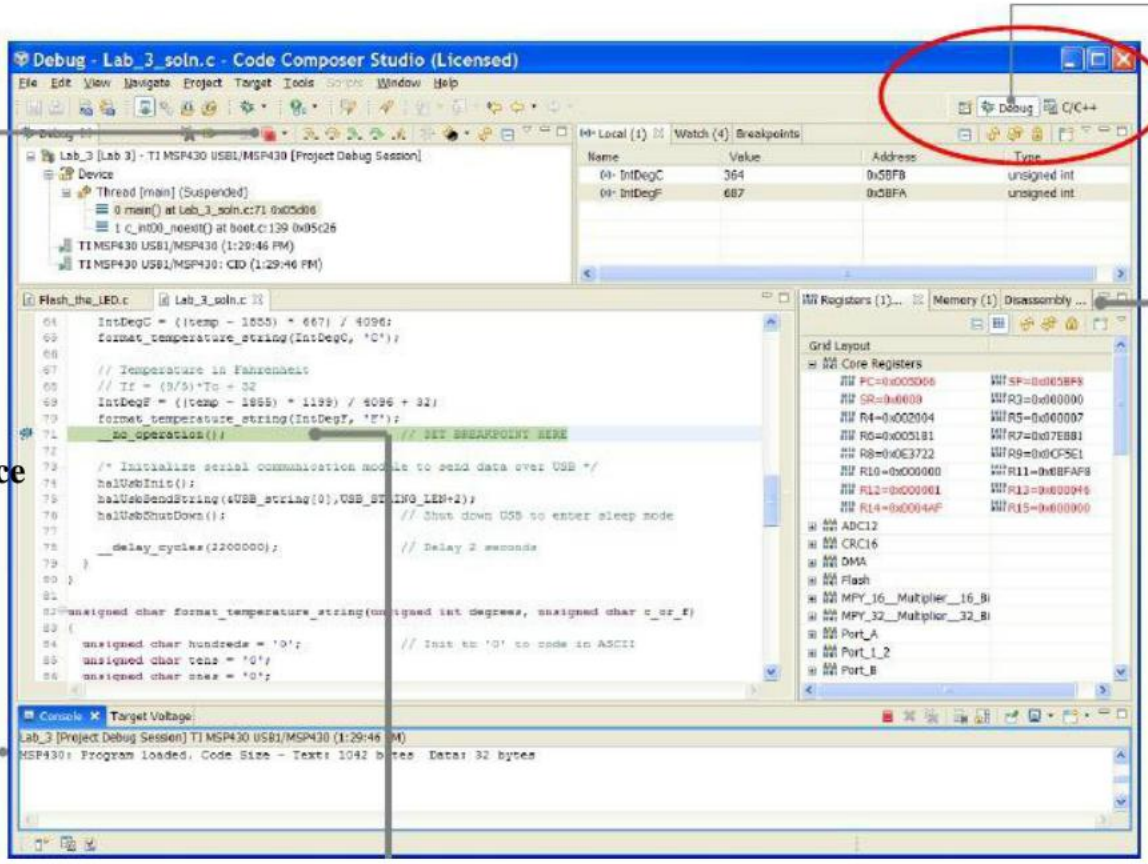
Code Window

Problems View

- 显示**Build**结果，包括**error**，**warning**等

CCS 界面 – Debug Perspective Overview

Debug界面



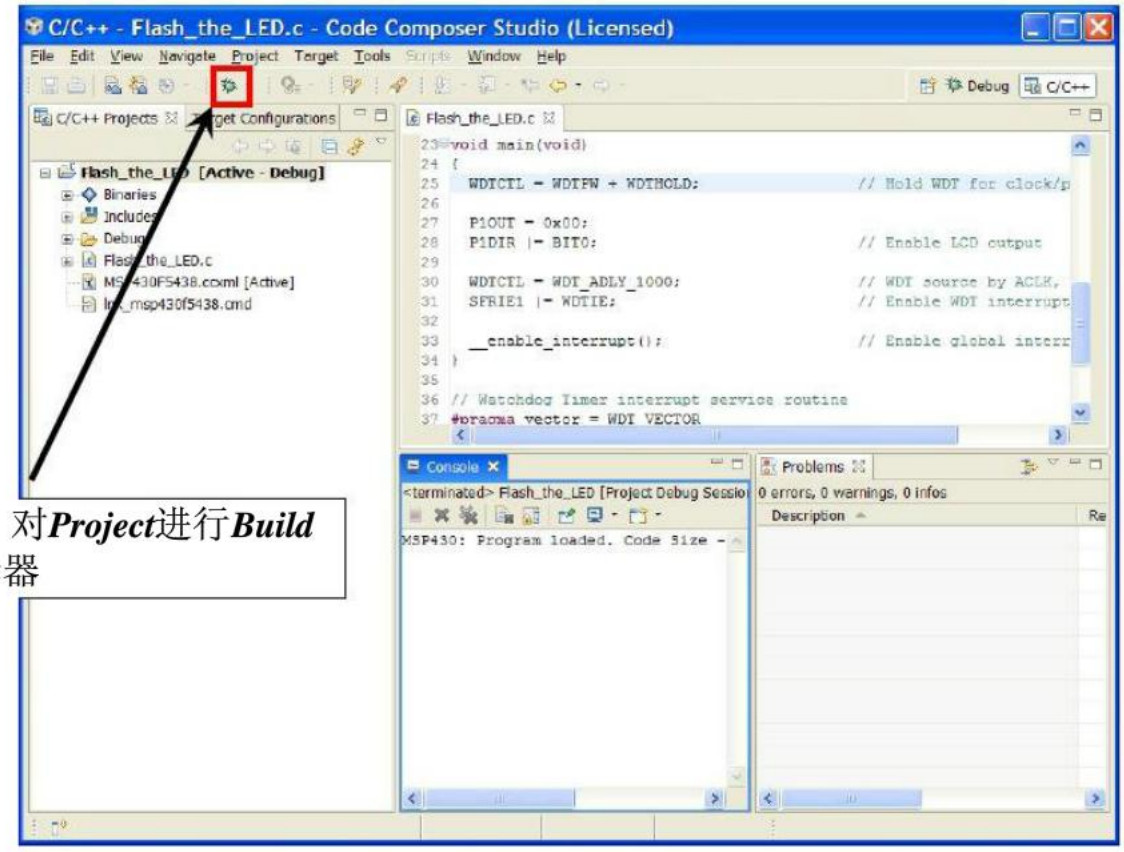
Target 控制
- **Start**
- **Stop**
- **Halt**
- **Stepping**
- **Stack Trace**

实时**430**信息显示
- 寄存器值
- **FLash, RAM, Info, segment**
- 汇编代码显示

程序大小信息**CodeSize**

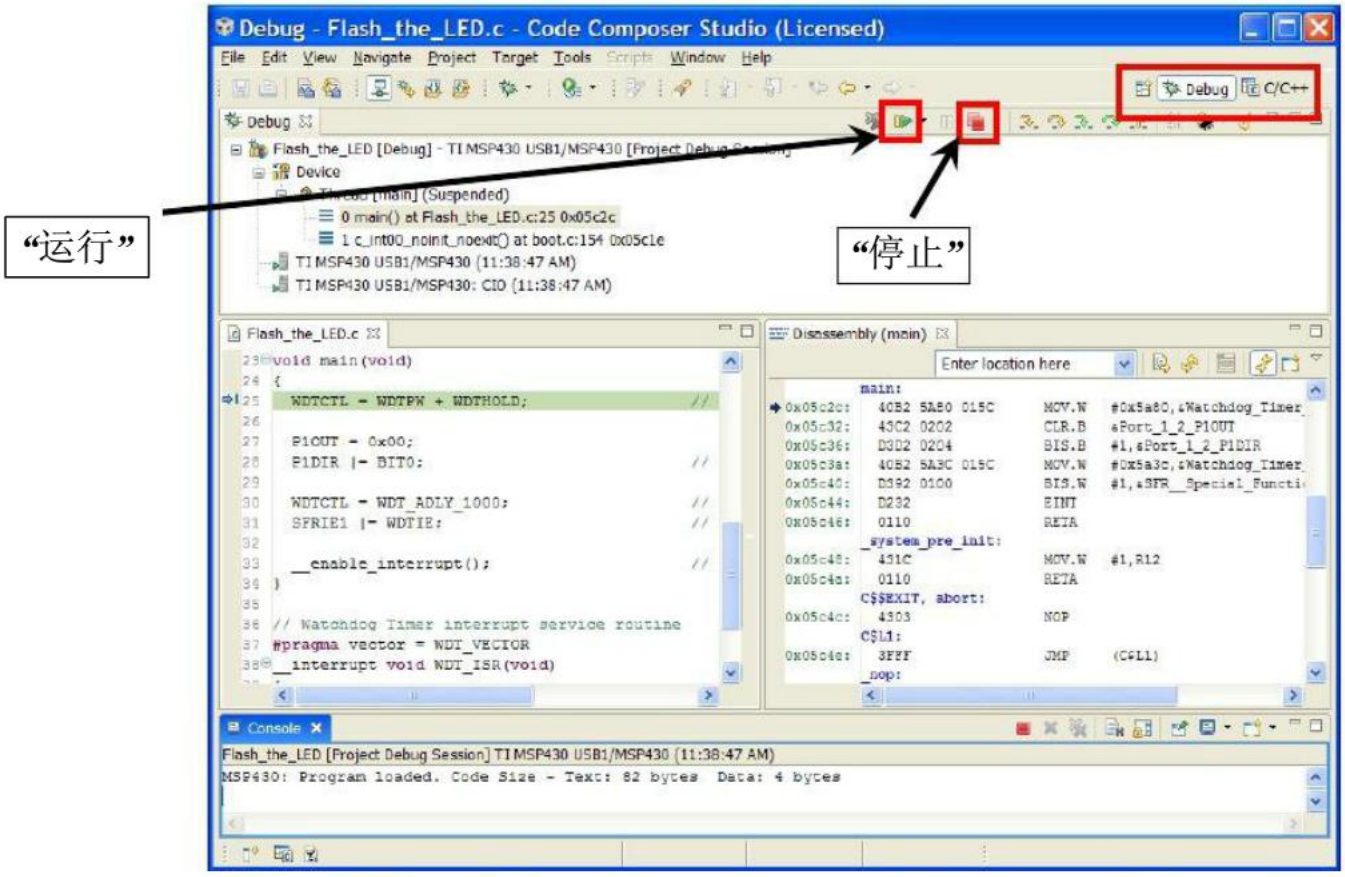
Code Window
- 实时断点，运行状况显示

Step 4: Build & Debug a CCS Project

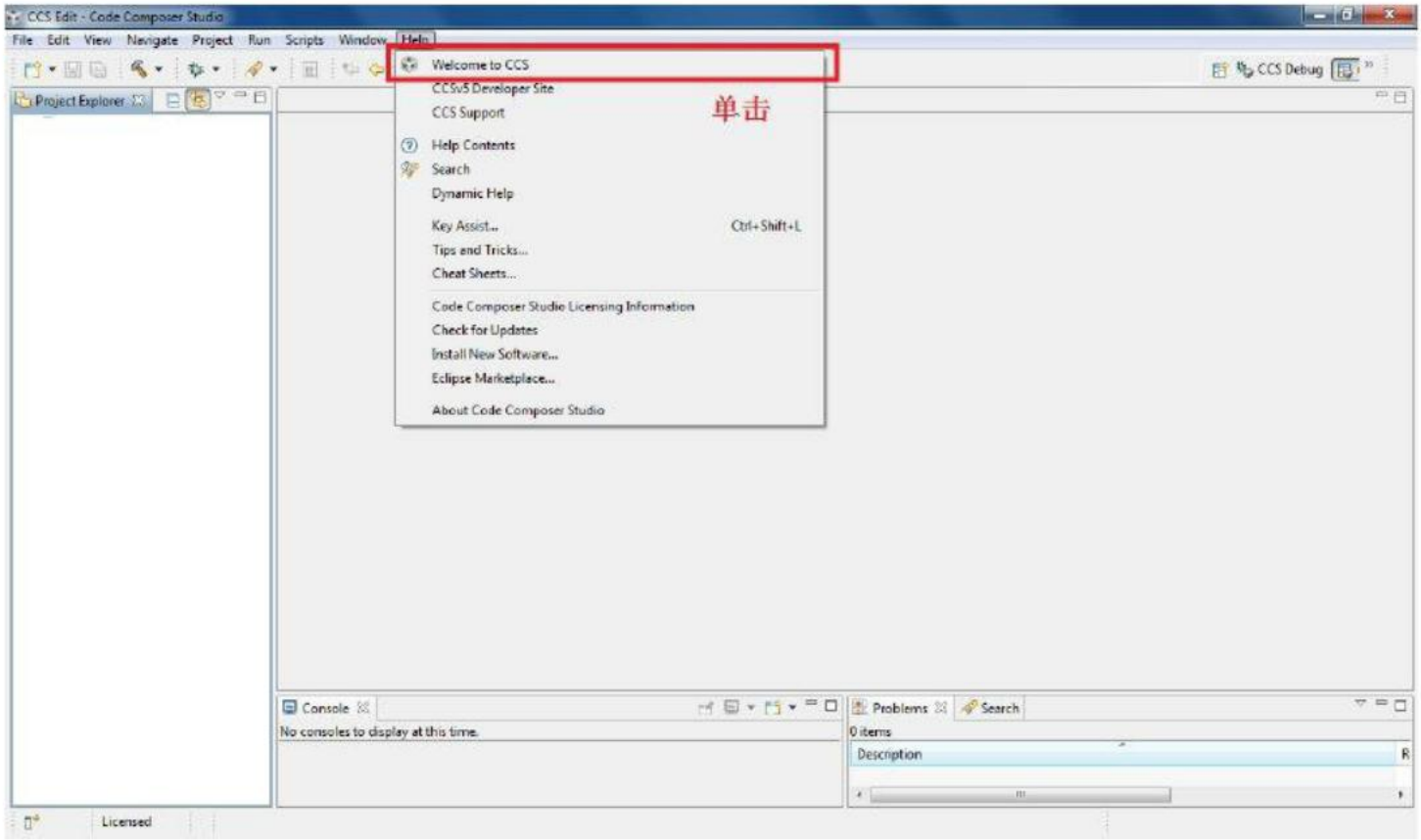


单击***"BUG"***，对***Project***进行***Build***并连接到编译器

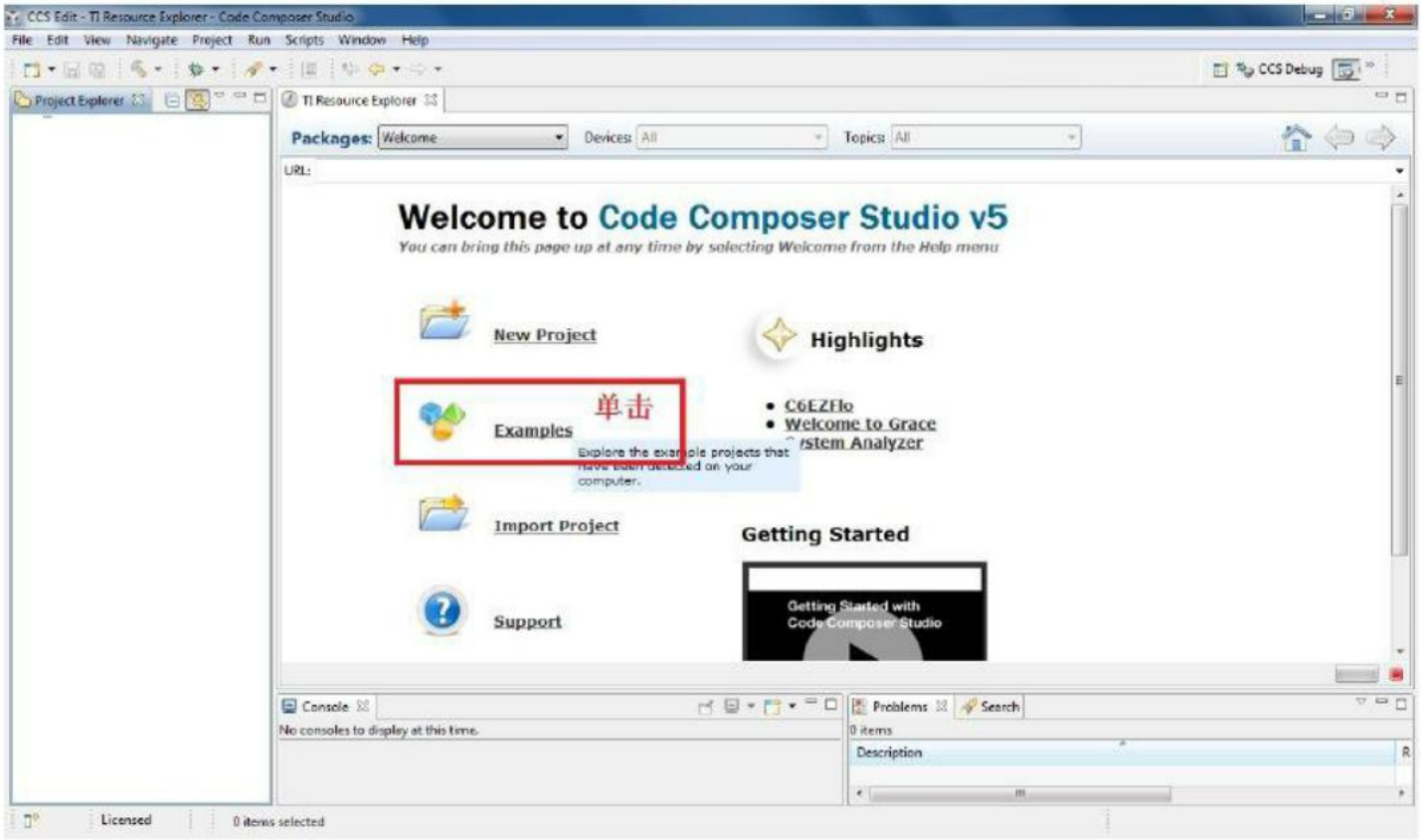
Step 5: 运行，终止 CCS Project



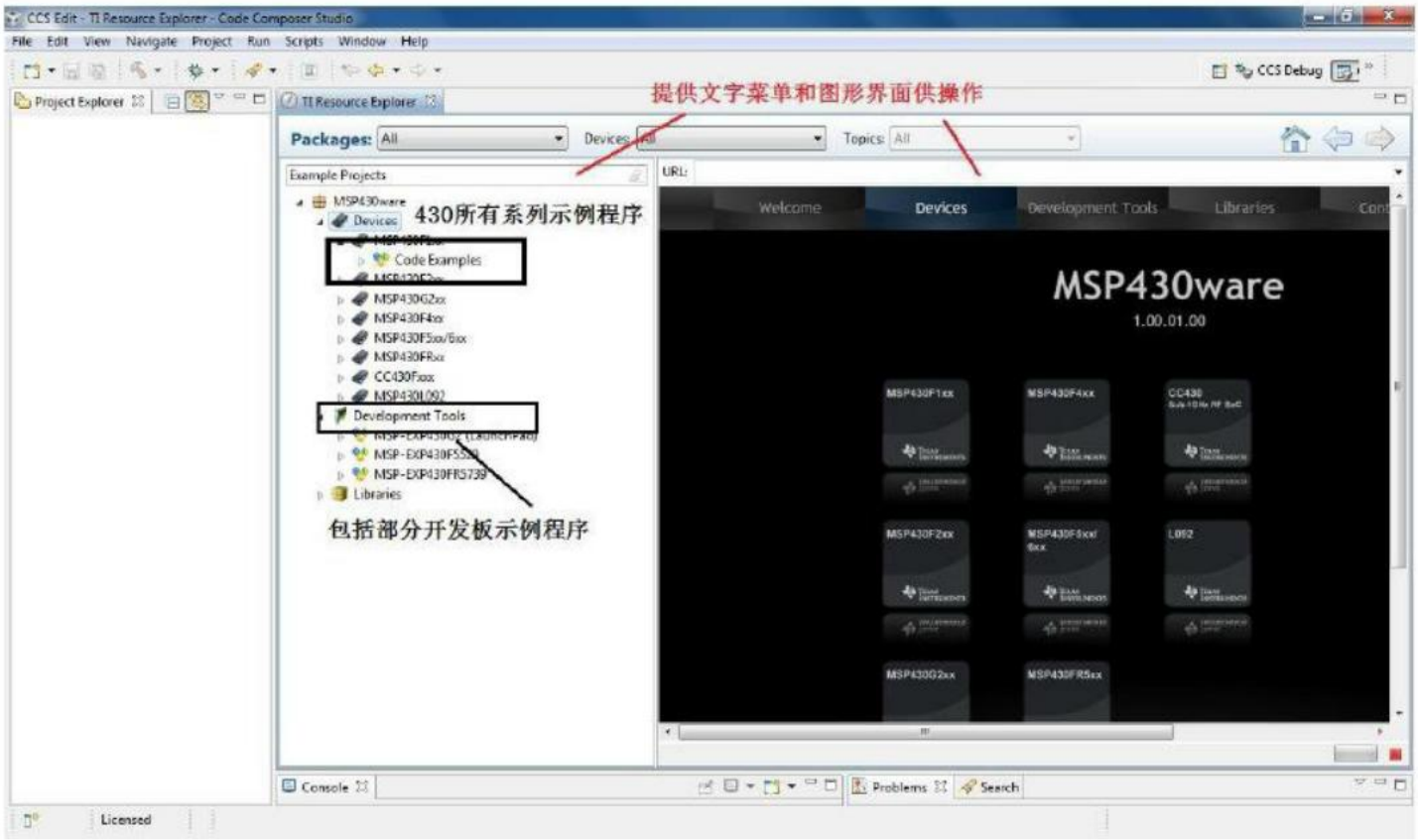
430Ware—快速程序开发工具



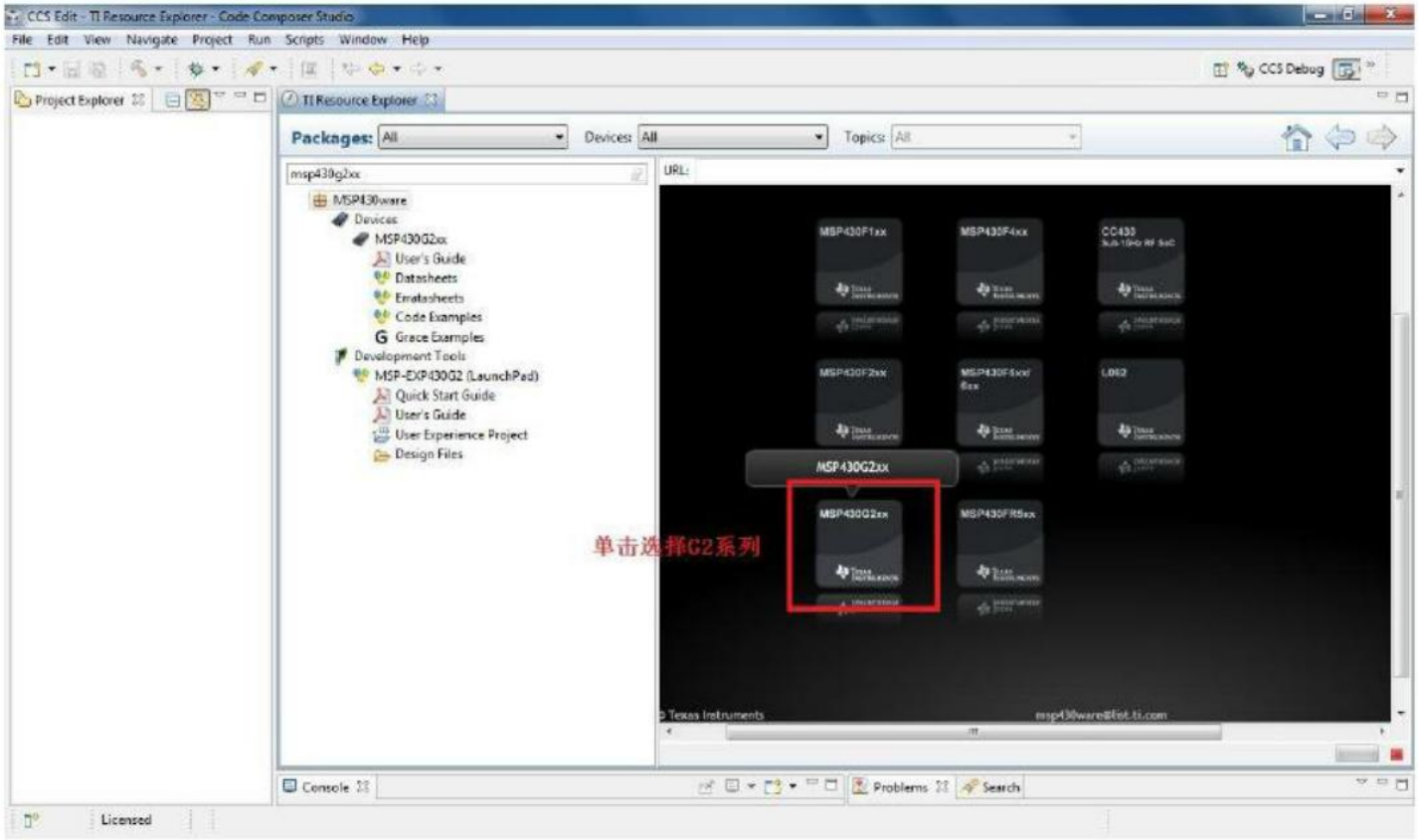
430Ware—快速程序开发工具



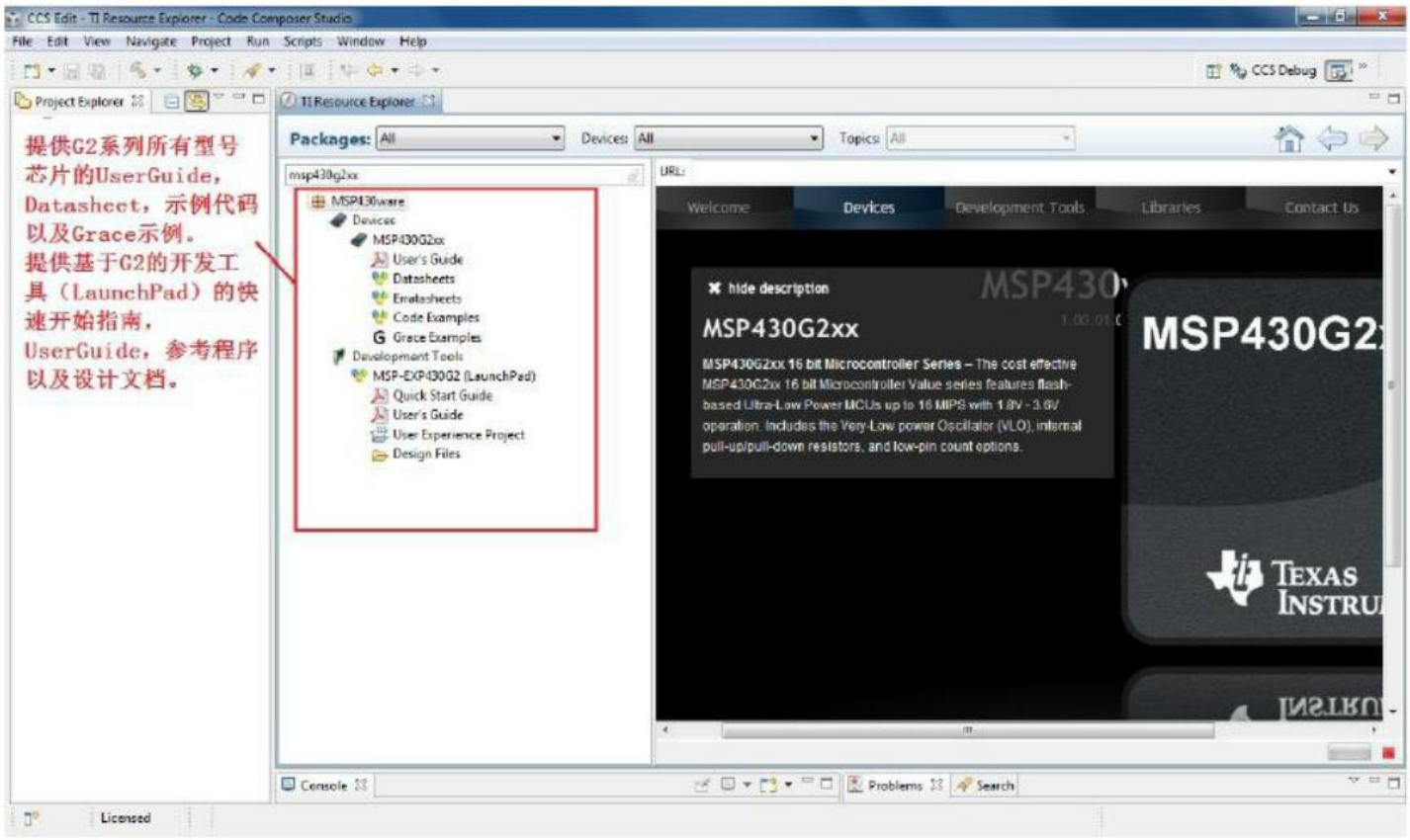
430Ware—快速程序开发工具



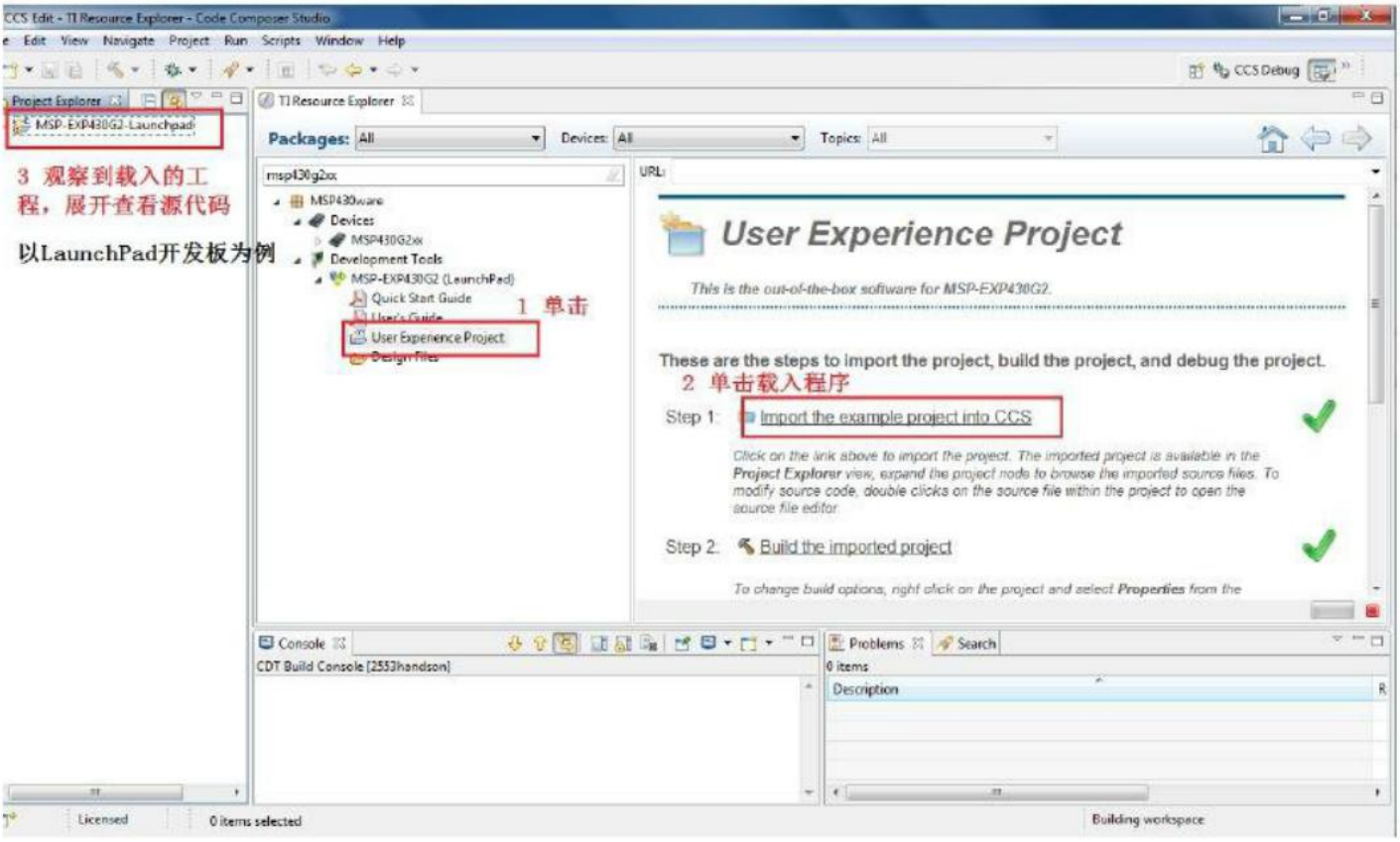
430Ware—快速程序开发工具



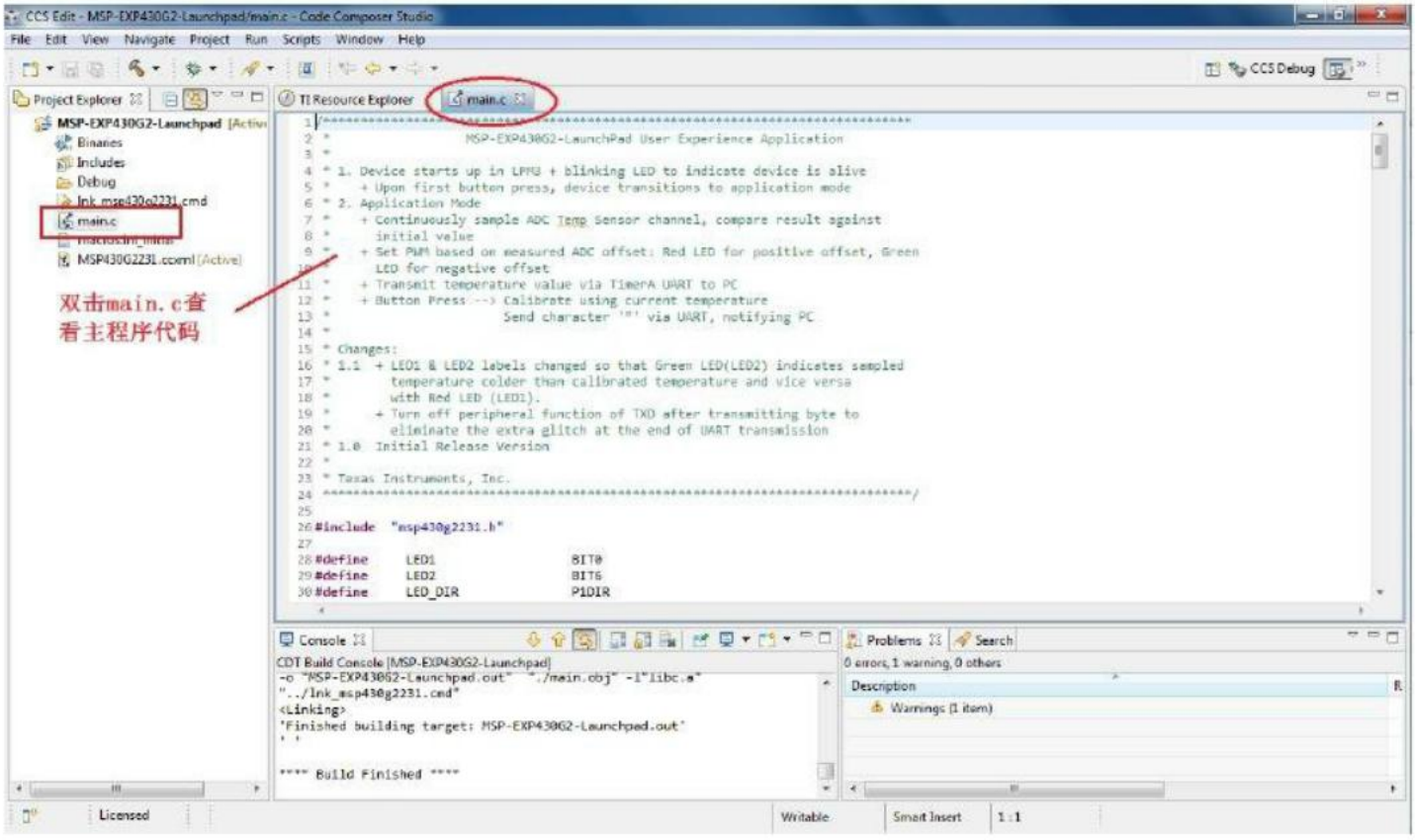
430Ware—快速程序开发工具



430Ware—快速程序开发工具



430Ware—快速程序开发工具



Agenda

1、**Value Line** 简介

2、**Code Composer Studio**

3、**CPU**及基本时钟系统

4、中断及通用**IO**

5、定时器及增强型**WDT**

6、低功耗优化设计

7、**ADC10** 及 **Comparator_A+**

8、串行通信

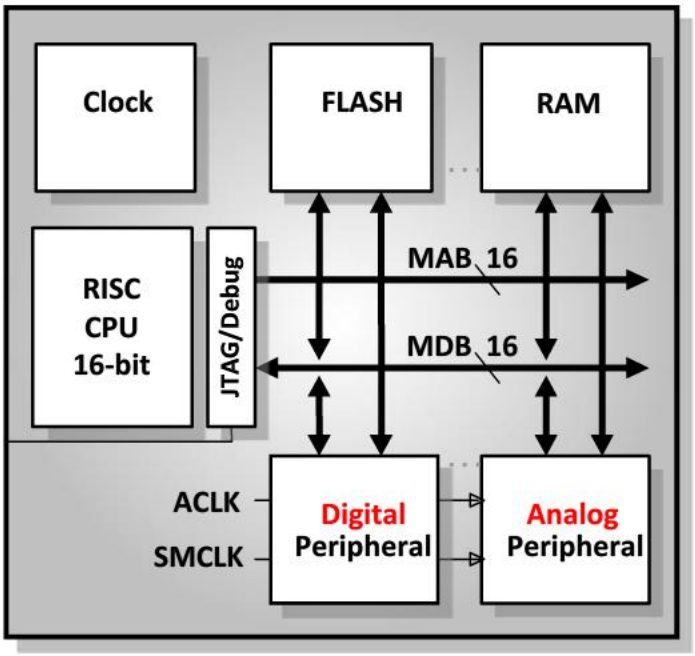
9、**Grace**

10、电容式触摸按键解决方案

3.1 MSP430G2xx 结构

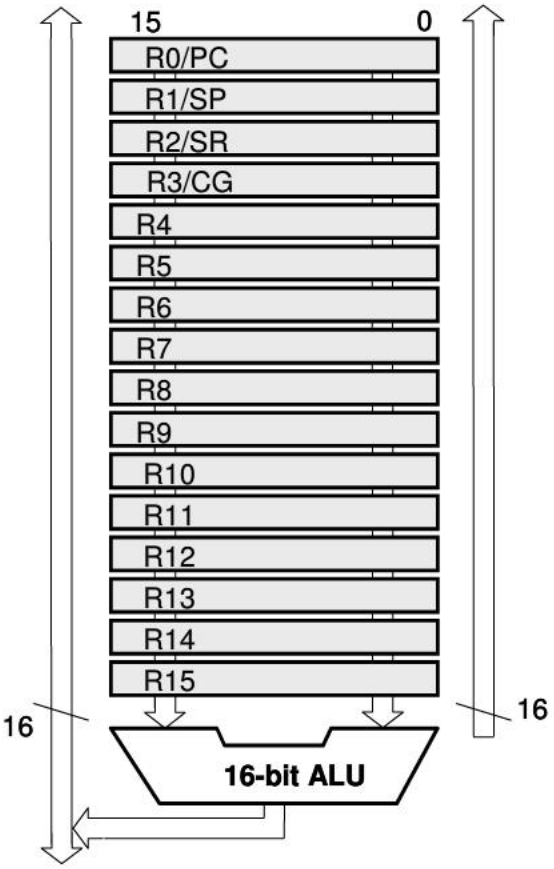
超低功耗
- 0.1uA 断电模式
- 0.8uA 待机模式
- 220uA / 1MIPS
- <1us 时钟唤醒
- <50nA 端口漏电流
- 零功耗欠压复位(BOR)

超灵活
- 0.5k-16kB 系统内可编程 (ISP) Flash
- 16-bit Timer
- SPI, I2C， UART
- 10bit ADC
- 嵌入式仿真



3.2 16-位 RISC CPU

- 单周期寻址寄存器文件
  - 4 专用寄存器
  - 12 通用寄存器
  - 无累加器瓶颈
- **RISC** 架构
  - 27 核心指令
  - 24 仿真指令
  - 7 寻址模式
- **Atomic** 内存至内存寻址
- **Bit, byte**及**word** 处理
- 常数发生器（**CG**，不占用程序空间）



- 程序计数器**PC**
  - ✓ 存放下一条将要从程序存储器中取出的指令的地址
  - ✓ **16**位的计数器，可直接寻址**64K**存储空间
- 堆栈指针**SP**
  - ✓ 存放堆栈栈顶的地址
  - ✓ 数据压栈时，**SP-2**，压入数据；数据出栈时，弹出数据，，**SP+2**
- 状态寄存器**SR**
  - ✓ 状态标志：进位、零标志、负标志、溢出标志
  - ✓ 控制标志：中断标志……

状态寄存器SR

15~9	8	7	6	5	4	3	2	1	0
保留	V	SCG1	SCG0	OscOFF	CPUOFF	GIE	N	Z	C

类型	名称	含义
状态标志	C 进位标志	当运算结果产生进位时 C 置位，否则 C 复位
	Z 零标志	当运算结果为零时 Z 置位，否则 Z 复位
	N 负标志	当运算结果为负时 N 置位，否则 N 复位
	V 溢出标志	当运算结果超出有符号数范围时 V 置位，溢出情况如下： 正数＋正数＝负数　　正数－负数＝负数 负数＋负数＝正数　　负数－正数＝正数
控制标志	GIE 中断标志位	控制可屏蔽中断 GIE 置位 CPU 可响应可屏蔽中断 GIE 复位 CPU 不响应可屏蔽中断
	CPUOFF CPU 控制位	置位 CPUOFF 位可使 CPU 进入关闭模式，可用所有允许的中断将 CPU 唤醒
	OscOFF 晶振控制位	置位 OscOFF 使晶体振荡器处于停止状态，置位 OscOFF 同时 CPUOFF 位也需置位。可用 NMI 或者外部中断（系统当前中断允许）将 CPU 唤醒
	SCG0，SCG1 时钟控制位	SCG1 置位关闭 SMCLK SCG0 置位关闭 DCO 发生器

➢ 常数发生器CG

- ✓ 经常使用的常数可以用常数发生器产生
- ✓ 所用常数的数值由寻址位**As**来定义，硬件完全可以自动地产生数字：**−1、0、1、2、4、8**

寄存器	As	常　数	说　明
R2	00	—	寄存器模式
R2	01	(0)	绝对寻址模式
R2	10	00004H	+4 位处理
R2	11	00008H	+8 位处理
R3	00	00000H	0 字处理
R3	01	00001H	+1
R3	10	00002H	+2 位处理
R3	11	0FFFFH	−1 字处理

使用这种方法产生常数的优点如下：

- 不需要特殊的指令。
- 对 6 种最常用的常数不需要额外的字操作数。
- 不用访问数据存储器，缩短指令周期。
- 当 6 种常数之一被用作立即寻址模式的源操作数时，汇编程序会自动转为利用 R2 或 R3 的方式。状态寄存器 SR/R2(用作源或目的寄存器)只能用于寄存器模式。其他的寻址位 As 组合支持绝对寻址和位处理，而不必增加代码。R2 与 R3 用于常数发生器时不能进行显式寻址，它们只能作为一个寄存器数据源。
- 常数发生器使得某些指令可用别的指令来模拟。用这种方法使 CPU 变得异常简单，整个指令组只需要 27 条指令。例如操作数指令：

```
CLR dst
```

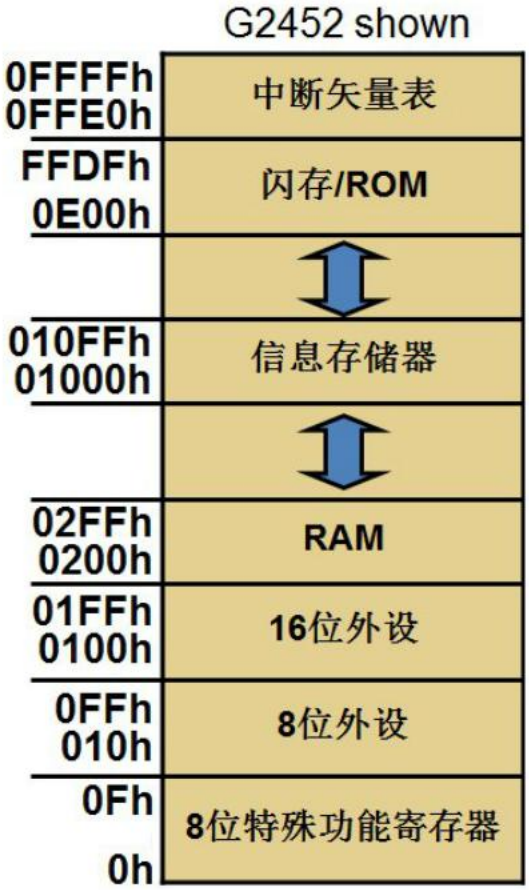
或等价的有：

```
MOV   #0,dst
```

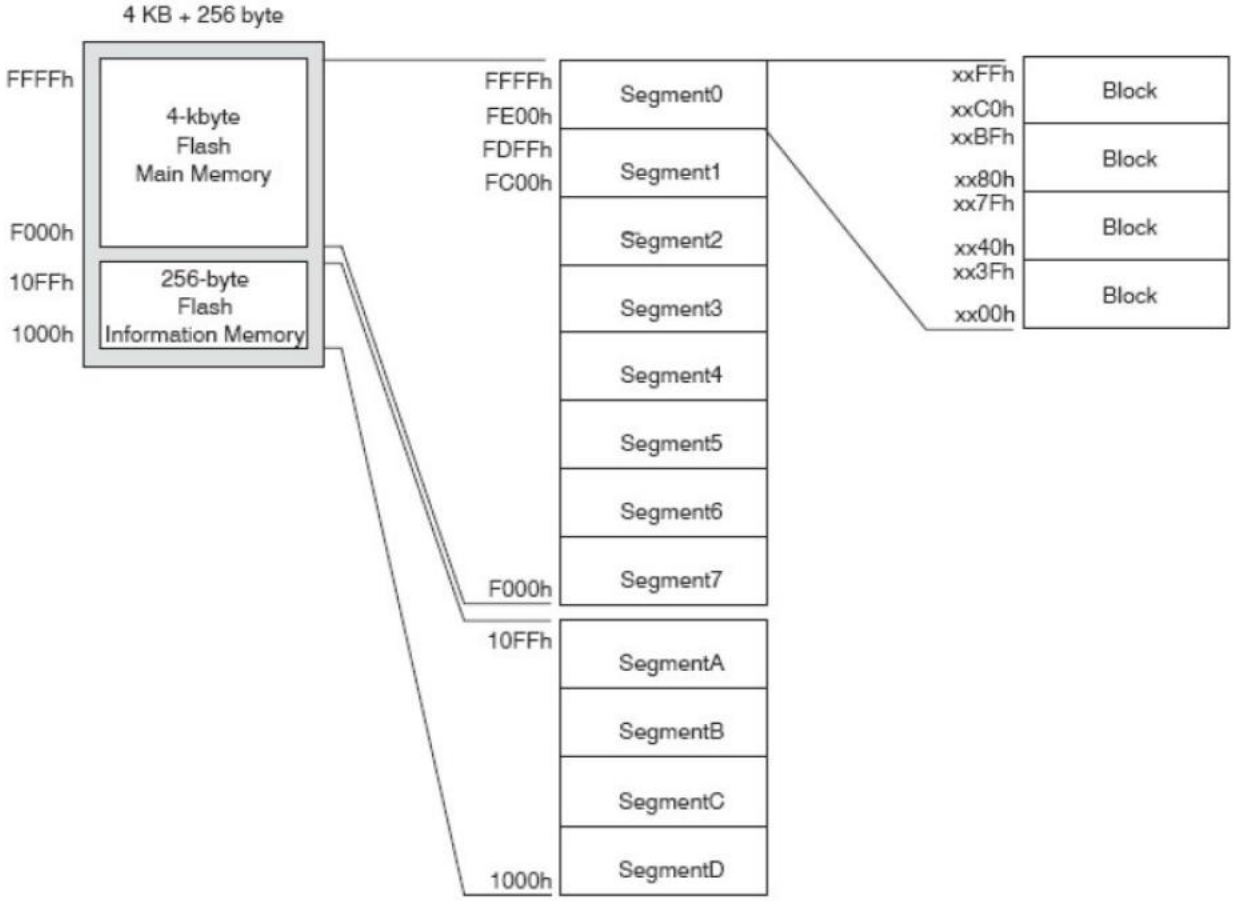
其中 #0 由汇编程序以 As=00 的 R3 来取代。

3.3 内存映射

- 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



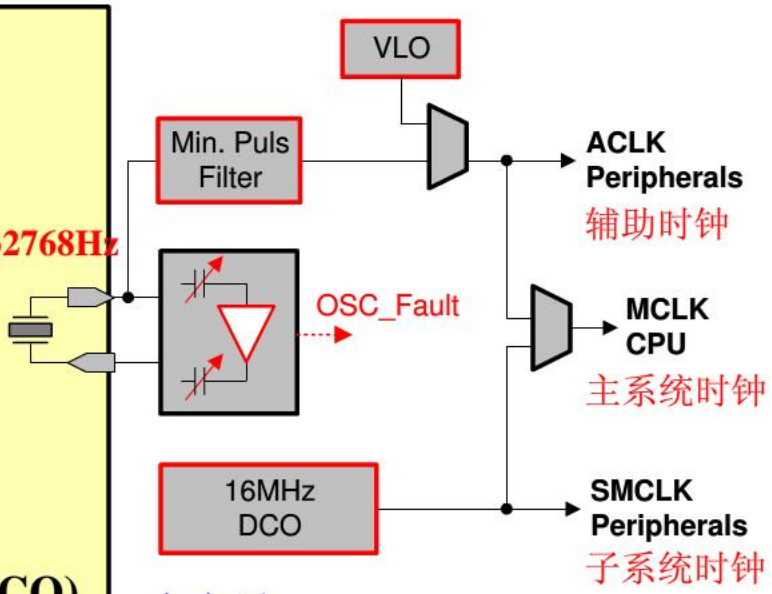
Flash存储器分段


4 KB + 256 byte
FFFFh
4-kbyte
Flash
Main Memory
F000h
10FFh
256-byte
Flash
Information Memory
1000h
FFFFh
FE00h
FDFFh
FC00h
Segment0
Segment1
Segment2
Segment3
Segment4
Segment5
Segment6
Segment7
F000h
10FFh
SegmentA
SegmentB
SegmentC
SegmentD
1000h
xxFFh
xxC0h
xxBFh
xx80h
xx7Fh
xx40h
xx3Fh
xx00h
Block
Block
Block
Block

- **Segment A**
  - ✓ 可以通过**LOCKA**位的设来进行锁定
    - **LOCKA=1**，**Segment A**不能写入/擦除，其他的信息存储器在整个模块擦除时被保护起来
    - **LOCKA=0**，可以像其他**Flash**一样进行擦除和写入的操作。
- **Segment B**、**C**、**D**在段擦除时均能被擦除，与**LOCKA**位的状态无关。

3.4 时钟系统

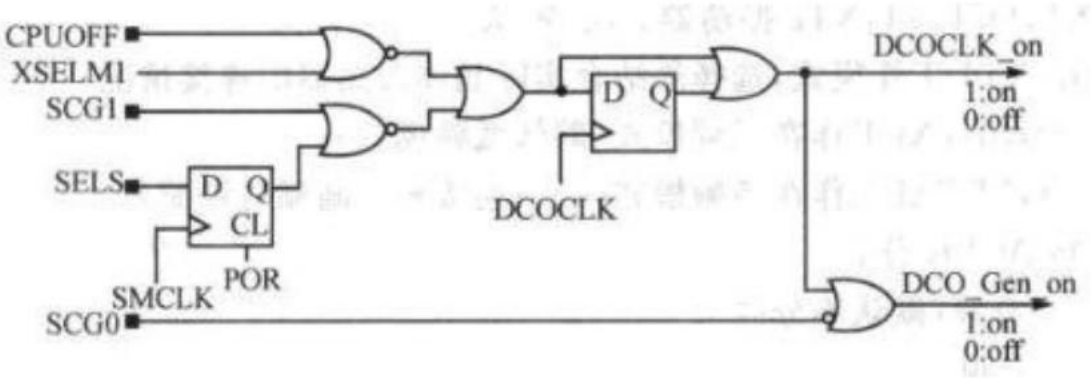
- 超低功耗/低频振荡器 **(VLO)**
  - 4 – 20kHz (典型值12kHz)
  - 500nA 待机流耗
  - 0.5%/° C and 4%/V 漂移
- 外接晶体振荡器**(LFXT1)**
  - 片内可编程负载电容
  - 故障保护 OSC_Fault
  - 脉冲滤波器
- **Digitally Controlled Oscillator (DCO)**
  - 0-to-16MHz
  - ± 3% 容差
  - 出厂校准（Flash I.M.）



上电后：
MCLK 和 **SMCLK** 由**DCOCLK** 提供（**~1.1 MHz**）
ACLK 由 **LFXT1CLK** 提供（**LF** 模式，**6pF**内部负载电容）

3.5 DCO校准

- **Digitally Controlled Oscillator**，可数字控制的**RC**振荡器
- **DCO**的频率随供电电压、环境温度变化而具有一定的不稳定性，可以通过操作寄存器软件调节来增强振荡频率的稳定性。
- 控制逻辑



时钟模块的控制寄存器

DCOCTL DCO 控制寄存器

7	6	5	4	3	2	1	0
DCO. 2	DCO. 1	DCO. 0	MOD. 4	MOD. 3	MOD. 2	MOD. 1	MOD. 0

DCO.0-DCO.4 定义 8 种频率之一，可以分段调节 DCOCLK 频率，相邻两种频率相差 10%。而频率由注入直流发生器的电流定义。

MOD.0-MOD.4 定义在 32 个 DCO 周期中插入的 Fdco+1 周期个数，而在下的 DCO 周期中为 Fdco 周期，控制改换 DCO 和 DCO+1 选择的两种频率。如果 DCO 常数为 7，表示已经选择最高频率，此时不能利用 MOD.0-MOD.4 进行频率调整。

BCSCTL1 基本时钟系统控制寄存器 1

7	6	5	4	3	2	1	0
XT2OFF	TXS	DIVA.1	DIVA.0	XT5V	Rsel.2	Resl.1	Resl.0

XT2OFF 控制 XT2 振荡器的开启与关闭。

TX2OFF=0，XT2 振荡器开启。

TX2OFF=1，TX2 振荡器关闭（默认为 TX2 关闭）

XTS 控制 LFXT1 工作模式，选择需结合实际晶体振荡器连接情况。

XTS=0，LFXT1 工作在低频模式（默认）。

XTS=1，LFXT1 工作在高频模式（必须连接有高频相应的高频时钟源）。

DIVA.0 DIVA.1 控制 ACLK 分频。

0 不分频（默认）

1 2 分频

2 4 分频

3 8 分频

XT5V 此位设置为 0。

Resl1.0，Resl1.1，Resl1.2 三位控制某个内部电阻以决定标称频率。

Resl=0，选择最低的标称频率。

……..

Resl=7，选择最高的标称频率。

BCSCTL2 基本时钟系统控制寄存器 2

7	6	5	4	3	2	1	0
SELM.1	SELM.0	DIVM.1	DIVM.0	SELS	DIVS.1	DIVS.0	DCOR

SELM.1 SELM.0 选择 MCLK 时钟源

0 时钟源为 DCOCLK（默认）

1 时钟源为 DCOCLK

2 时钟源为 LFXT1CLK（对于 MSP430F11/12X），时钟源为 XT2CLK（对于 MSP430F13/14/15/16X）；

3 时钟源为 LFTXTICLK。

DIVM.1 DIVM.0 选择 MCLK 分频

0 1 分频（默认）

1 2 分频

2 4 分频

3 8 分频

SELS 选择 SMCLK 时钟源

0 时钟源为 DCOCLK（默认）

1 时钟源为 LFXT1CLK（对于 MSP430F11/12X），时钟源为 XT2CLK（对于 MSP430F13/14/15/16X）。

DIVS.1 DIVS.0 选择 SMCLK 分频。

0 1 分频
1 2 分频
2 4 分频
4 8 分频

DCOR 选择 DCO 电阻

0 内部电阻
1 外部电阻

Power Up Clear，复位信号

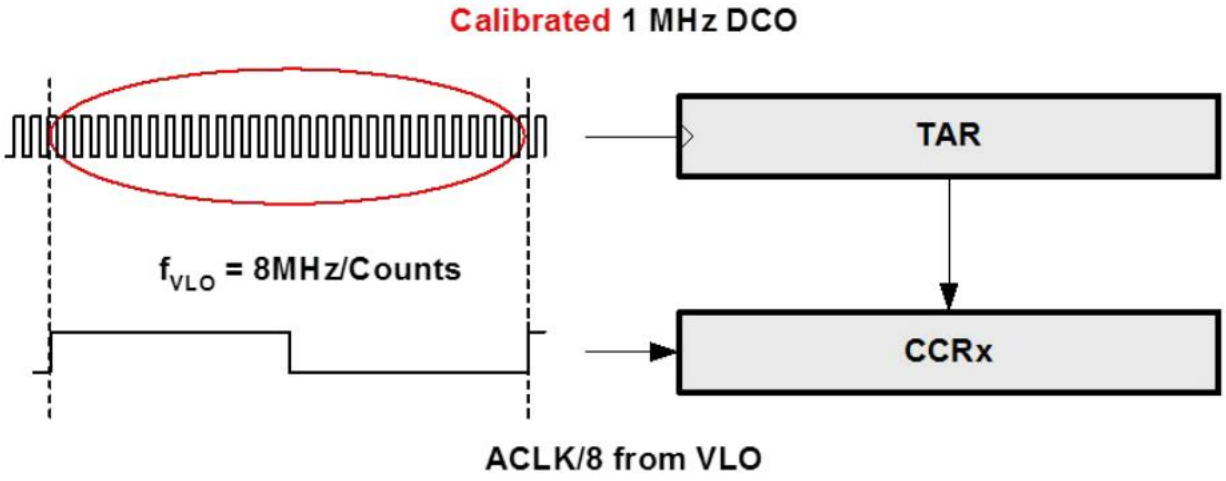
PUC 信号之后，DCOCLK 被自动选择 MCLK 时钟信号，根据需要，MCLK 的时钟源可以另外设置为 LFXT1 或者 XT2。设置顺序如下：

[1] 复位 OscOff
[2] 清除 OFIFG
[3] 延时等待至少 50us
[4] 再次检查 OFIFG，如果仍然置位，则重复[3]、[4]步骤，直到 OFIFG=0 为止。

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

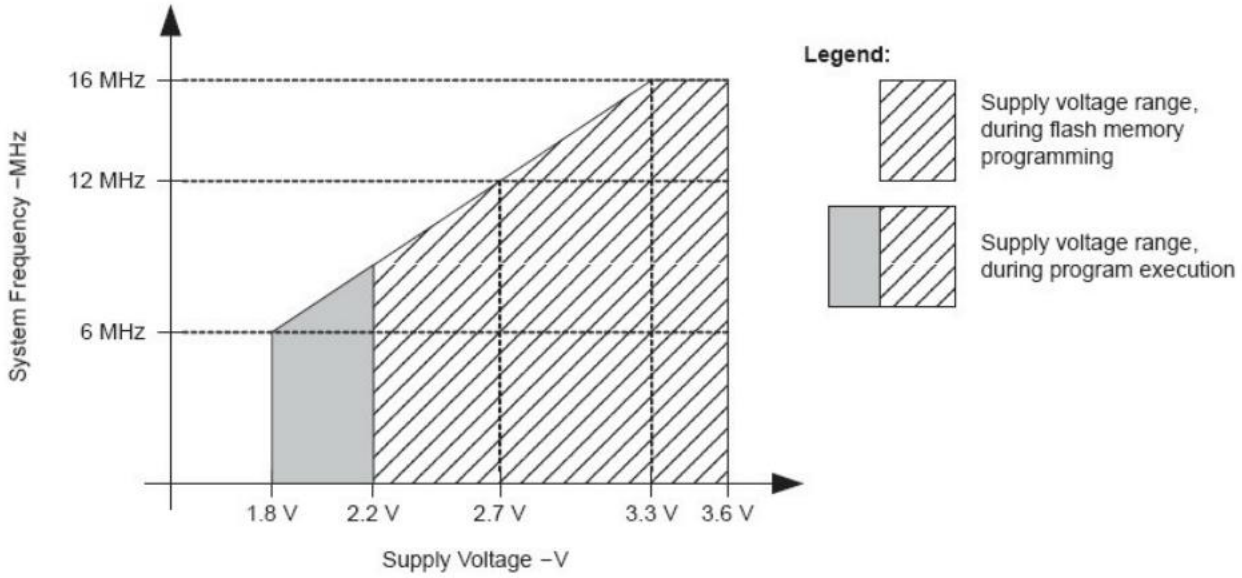
```
 // Setting the DCO to 1MHz
 if (CALBC1_1MHZ ==0xFF || CALDCO_1MHZ == 0xFF)
   while(1);                // Erased calibration data? Trap!
BCSCTL1 = CALBC1_1MHZ;      // Set range
DCOCTL  = CALDCO_1MHZ;      // Set DCO step + modulation
```

3.6 VLO校准



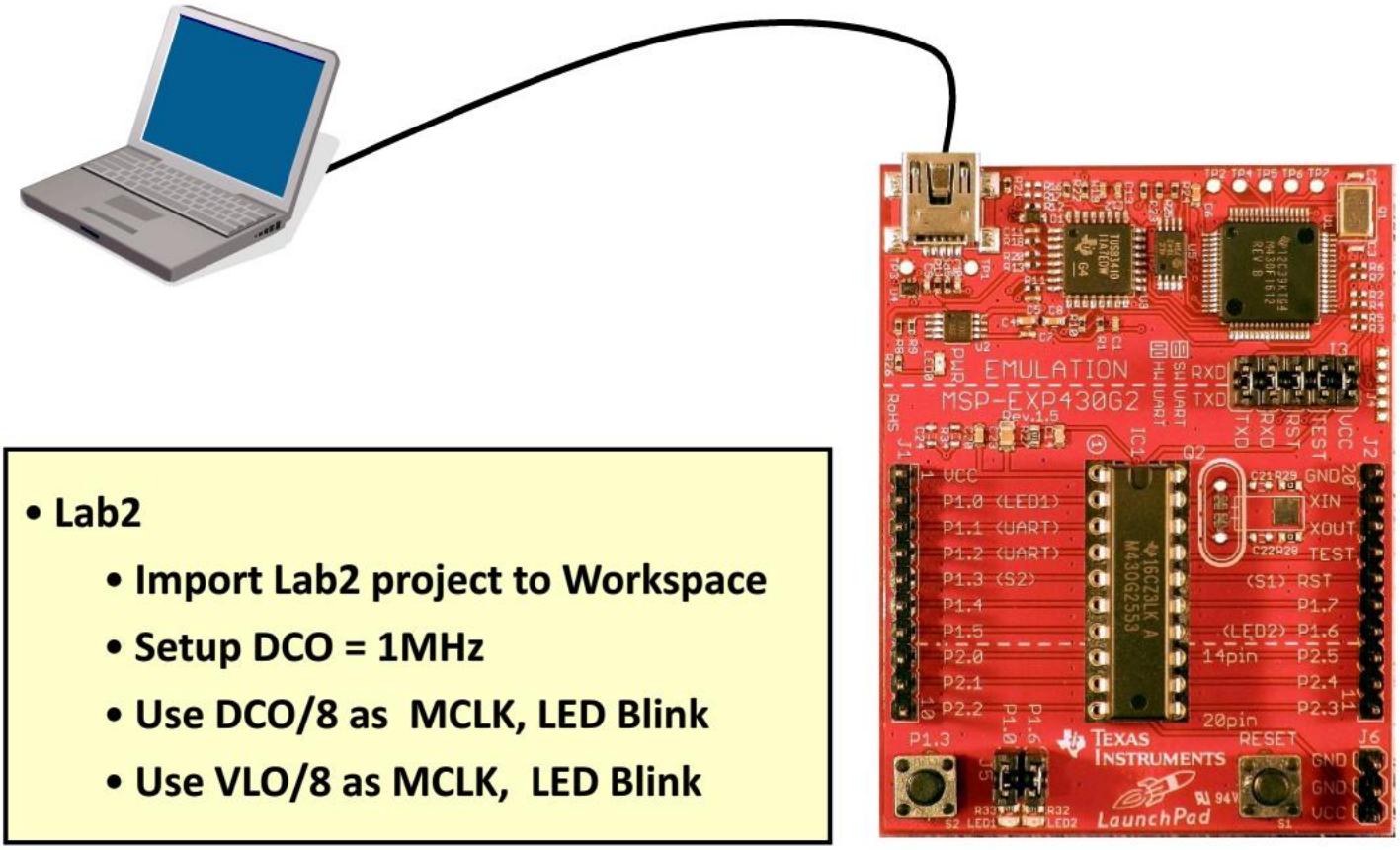
- 在运行期间对VLO进行校准
- 定时器 Timer_A 时钟源采用校准的1MHz DCO
- 采用由VLO提供的ACLK/8进行上升沿捕获
- fVLO = 8MHz/Counts
- 参考例程 (SLAA340)

系统 MCLK & Vcc



- 时钟速度的选取必须与**VCC**相匹配
- 外部 **LDO**稳压器
- 当**VCC**低于所选频率的最小工作电压时，系统工作状态不稳定

Lab2: 基本时钟配置



- **Lab2**
  - **Import Lab2 project to Workspace**
  - **Setup DCO = 1MHz**
  - **Use DCO/8 as MCLK, LED Blink**
  - **Use VLO/8 as MCLK, LED Blink**

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

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

- Reference User's Guide, Datasheet & Schematic

基本时钟控制寄存器

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	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

```
// Configure Basic Clock
BCSCTL1 = CALBC1_1MHZ;  // Set range
DCOCTL = CALDCO_1MHZ;   // Set DCO step + modulation
BCSCTL3 |= LFXT1S_2;// Set LFXT1
```

```
// Configure MCLK
BCSCTL2 |= SELM_1 + DIVM_3; // Set MCLK
```

对**MCLK**，即指令运行的时钟源进行分频

DIVM_0: MCLK=1M

DIVM_1: MCLK=1M/2

DIVM_2: MCLK=1M/4

DIVM_3: MCLK=1M/8

➢ 在CCS中可以对头文件定义进行查看

Temperature_Sense_Demo.c | Lab2.c | Lab3.c | Lab4.c | Lab4_1.c | msp430g2452.h

```
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
```

Agenda

1、**Value Line** 简介

2、**Code Composer Studio**

3、**CPU**及基本时钟系统

4、中断及通用**IO**

5、定时器及增强型**WDT**

6、低功耗优化设计

7、**ADC10** 及 **Comparator_A+**

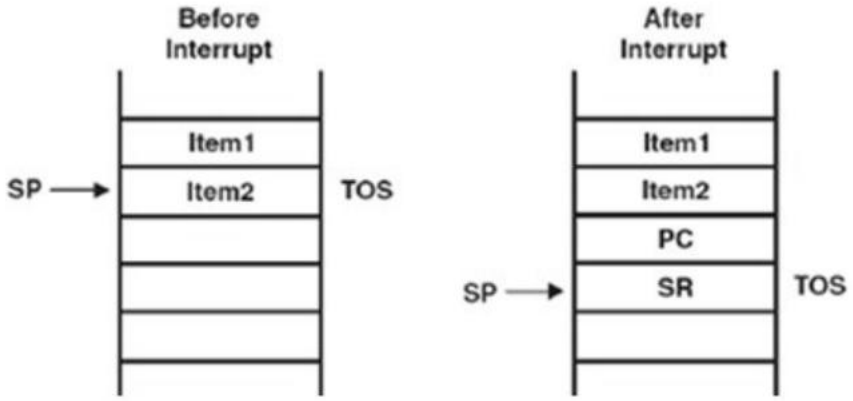
8、串行通信

9、**Grace**

10、电容式触摸按键解决方案

4.1 中断和堆栈

- 进入中断服务程序
  - 完成当前正在执行的指令；指向下一条指令的PC压栈
  - 状态寄存器 SR压栈；选择高优先级别的中断
  - 中断请求标志复位
    - 若为单中断源，则自动复位
    - 若为多中断源，则保持置位状态，由软件控制
  - SR清零：终止任何低功耗模式；由于GIE清零，禁止执行更多的中断
  - PC指向中断服务子程序，执行相应的子程序



4.2 Vector Table

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-Up External Reset Watchdog Timer+ Flash key violation PC out-of-range(1)	PORIFG RSTIFG WDTIFG KEYV(2)	Reset	0FFFEh	31, highest
NMI Oscillator fault Flash memory access violation	NMIIFG OFIFG ACCVIFG(2)(3)	(non)-maskable (non)-maskable (non)-maskable	0FFFCh	30
Timer1_A3	TA1CCR0 CCIFG(4)	maskable	0FFFAh	29
Timer1_A3	TA1CCR2 TA1CCR1 CCIFG, TAIFG(2)(4)	maskable	0FFF8h	28
Comparator_A+	CAIFG(4)	maskable	0FFF6h	27
Watchdog Timer+	WDTIFG	maskable	0FFF4h	26
Timer0_A3	TA0CCR0 CCIFG(4)	maskable	0FFF2h	25
Timer0_A3	TA0CCR2 TA0CCR1 CCIFG, TAIFG (5)(4)	maskable	0FFF0h	24
USCI_A0/USCI_B0 receive USCI_B0 I2C status	UCA0RXIFG, UCB0RXIFG(2)(5)	maskable	0FFEEh	23
USCI_A0/USCI_B0 transmit USCI_B0 I2C receive/transmit	UCA0TXIFG, UCB0TXIFG(2)(6)	maskable	0FFECh	22
ADC10 (MSP430G2x53 only)	ADC10IFG(4)	maskable	0FFEAh	21
			0FFE8h	20
I/O Port P2 (up to eight flags)	P2IFG.0 to P2IFG.7(2)(4)	maskable	0FFE6h	19
I/O Port P1 (up to eight flags)	P1IFG.0 to P1IFG.7(2)(4)	maskable	0FFE4h	18
			0FFE2h	17
			0FFE0h	16
See (7)			0FFDEh	15
See (8)			0FFDEh to 0FFC0h	14 to 0, lowest

4.3 中断处理函数编程

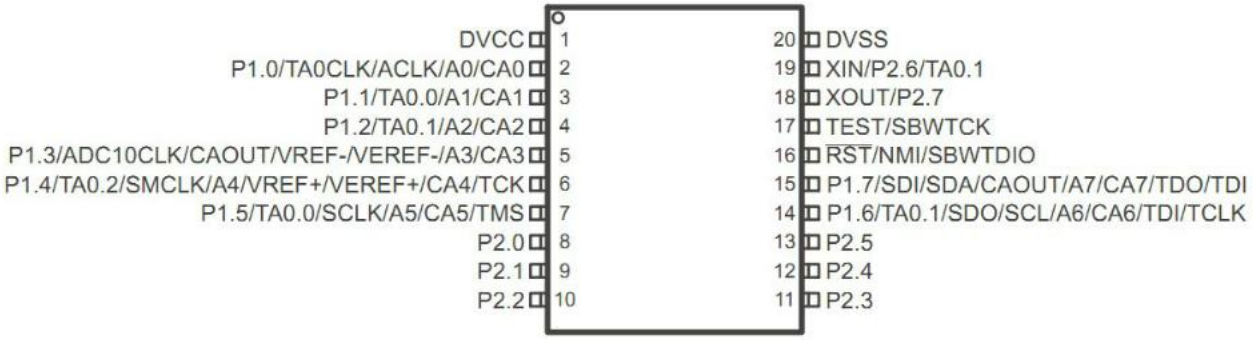
```
#pragma vector=WDT_VECTOR
__interrupt void WDT_ISR(void)
{
    IE1 &= ~WDTIE;                //disable interrupt
    IFG1 &= ~WDTIFG;              //clear interrupt flag
    WDTCTL = WDTPW + WDTHOLD; //put WDT back in hold state
    BUTTON_IE |= BUTTON;          //Debouncing complete
}
```

#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

4.4 GPIO端口



GPIO Registera

- Input Register PxIN
- Output Register PxOUT
- Direction Register PxDIR
- Function Select PxREN
- Function Select PxSEL
- Function Select PxSEL2
- Interrupt Edge PxIES
- Interrupt Enable PxIE
- Interrupt Flags PxIFG

For GPIO Int

GPIO Code Example

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

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

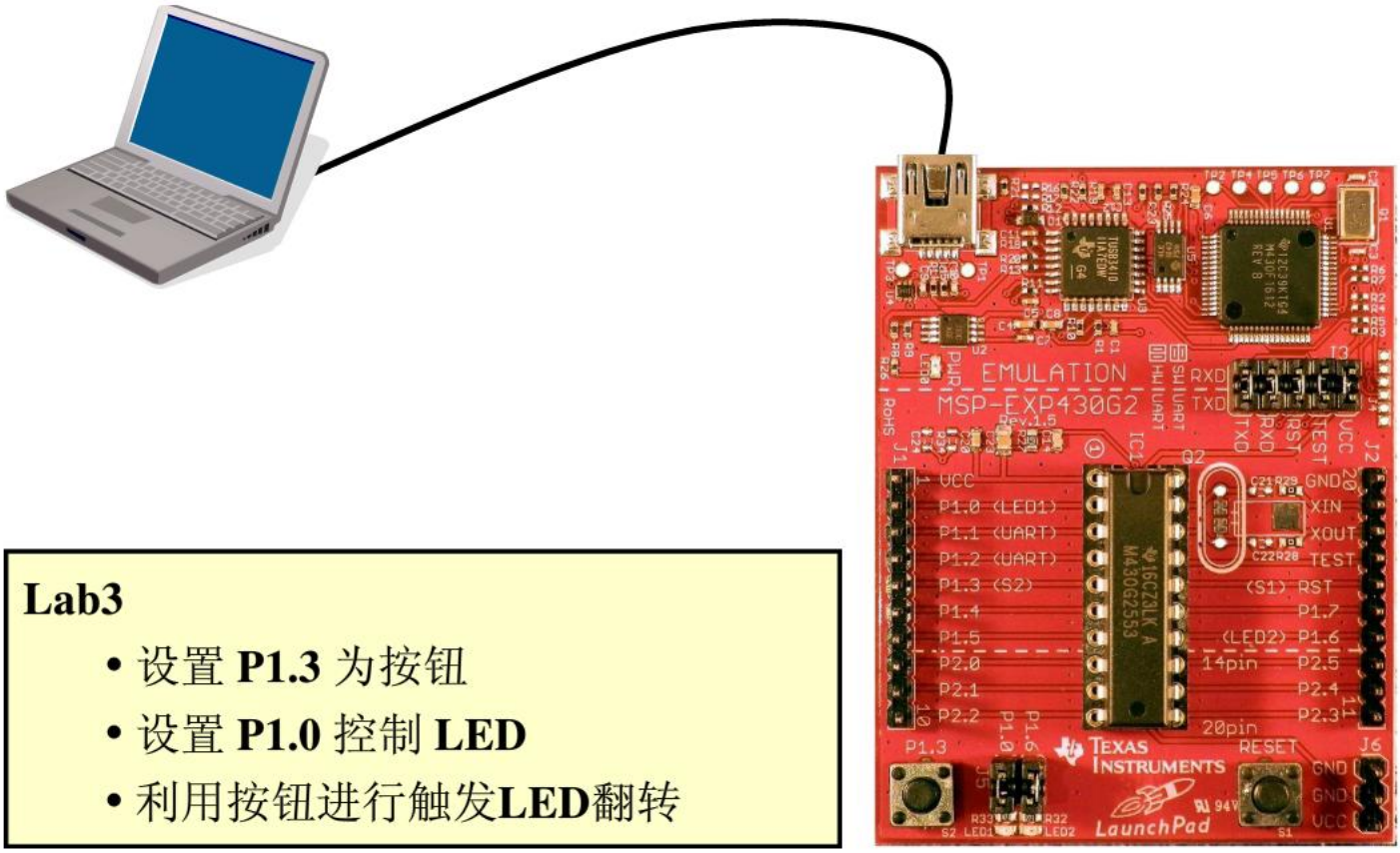
4.5 引脚复用

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/	4	P1.x (I/O)	I: 0; O: 1	0	0	0	0	0
SMCLK/		SMCLK	1	1	0	0	0	0
TA0.2/		TA0.2	1	1	1	0	0	0
		TA0.CCI2A	0	1	1	0	0	0
VREF+[2]/		VREF+	X	X	X	1	0	0
VEREF+[2]/		VEREF+	X	X	X	1	0	0
A4[2]/		A4	X	X	X	1 (y = 4)	0	0
CA4/		CA4	X	X	X	0	0	1 (y = 4)
TCK/		TCK	X	X	X	0	1	0
Pin Osc		Capacitive sensing	X	0	1	0	0	0

- 每个引脚均有多项功能
- 通过**sel**和**sel2**组合设置功能引脚
- 参考**datasheet**

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

```
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
```

```
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
```

```
P1DIR |= BIT0; // Set P1.0 to output direction
P1IES |= BIT3; // P1.3 Hi/lo edge
P1IFG &= ~BIT3; // P1.3 IFG cleared
P1IE  |= BIT3; // P1.3 interrupt
```

```
 // Port1 interrupt service routine
#pragma vector = PORT1_VECTOR
__interrupt void Port_1(void)
```

```
 // Port1 interrupt service routine
P1OUT ^= BIT0;    // P1.0 = toggle
P1IFG &= ~BIT3; // P1.3 IFG cleared
```

Agenda

1、**Value Line** 简介

2、**Code Composer Studio**

3、**CPU**及基本时钟系统

4、中断及通用**IO**

5、定时器及增强型**WDT**

6、低功耗优化设计

7、**ADC10** 及 **Comparator_A+**

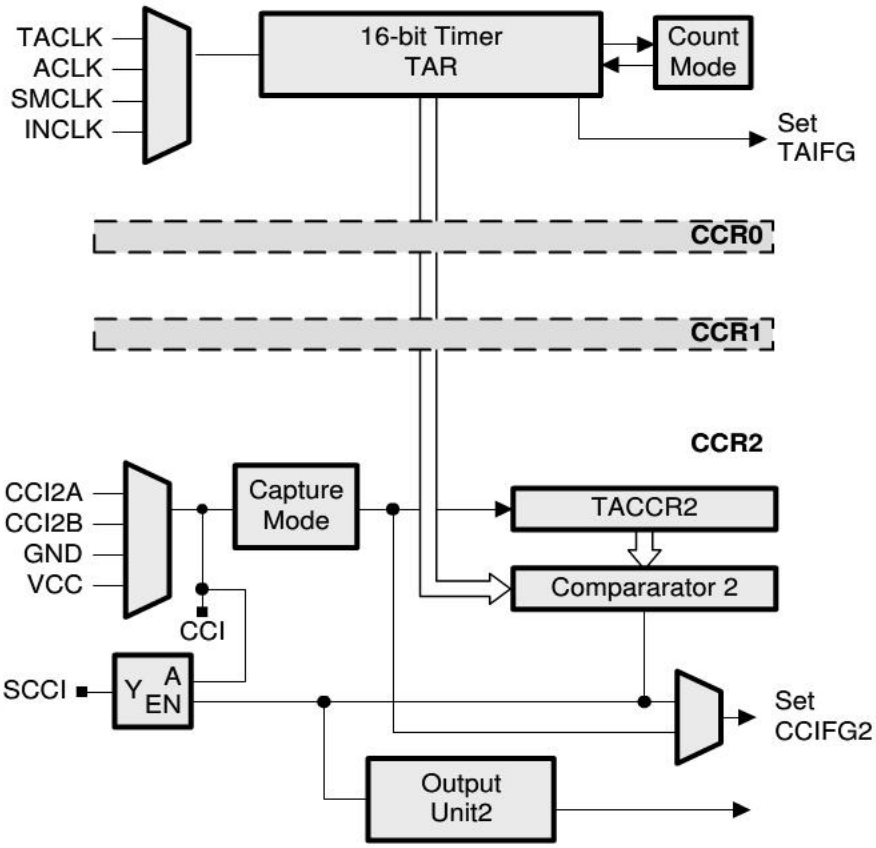
8、串行通信

9、**Grace**

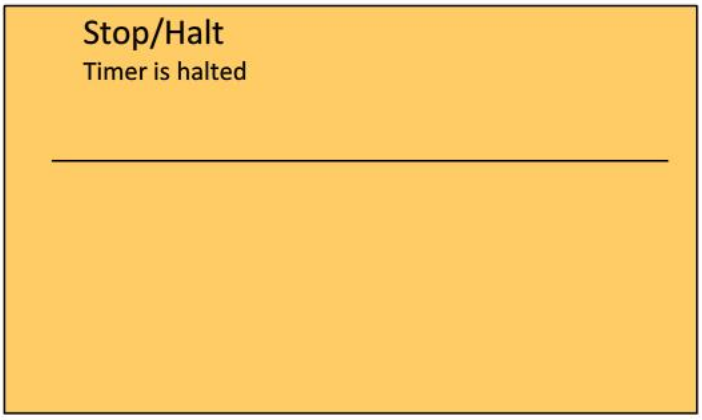
10、电容式触摸按键解决方案

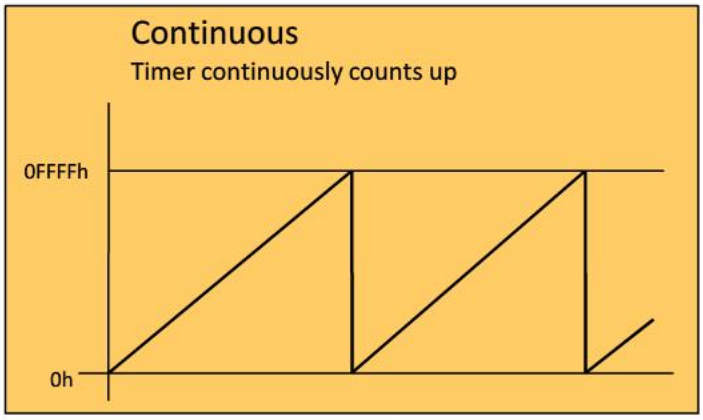
5.1 Timer_A

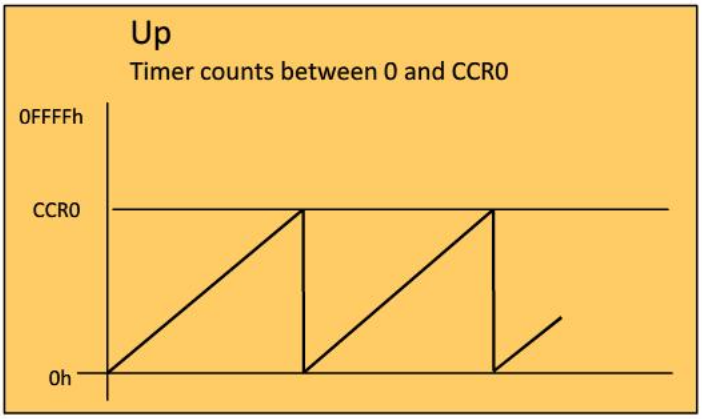
- 异步16位定时器/计数器
- 连续、递增-递减、递增计数模式
- 3个捕获/比较寄存器
- PWM 输出
- 中断向量寄存器，实现快速中断响应
- 可触发DMA
- 多个时钟源可选
- 所有430均有Timer_A

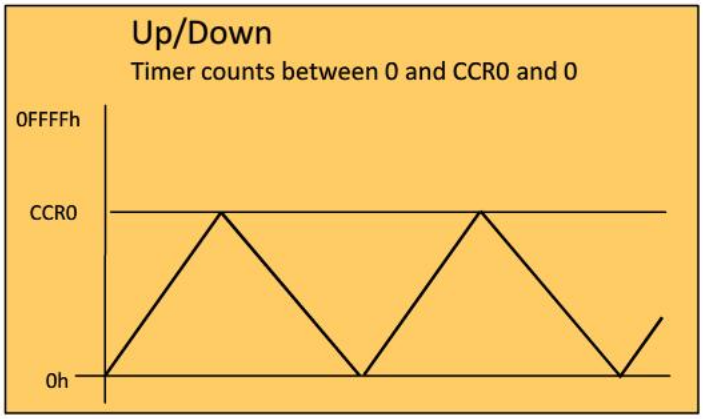


5.2 Timer_A 计数模式









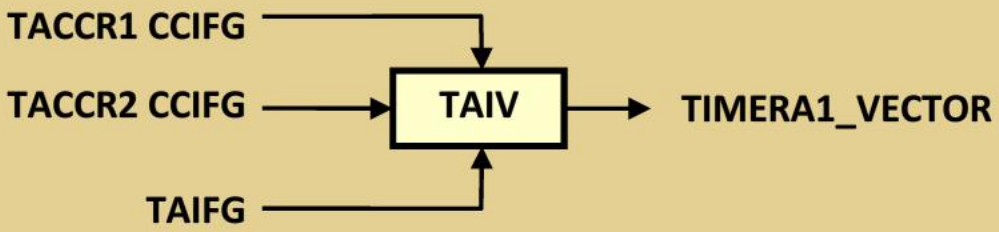
5.3 Timer_A中断

Timer_A 捕获/比较寄存器**TACCR0** 能够产生一个单独的中断向量

TACCR0 CCIFG → **TIMERA0_VECTOR**

无需额外的处理程序

TACCR1, TACCR2 以及 **TA** 中断标志进行优先级处理，组成**TAIV**中断向量。



需要程序判断中断的触发源

5.4 TAIV处理程序示例

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

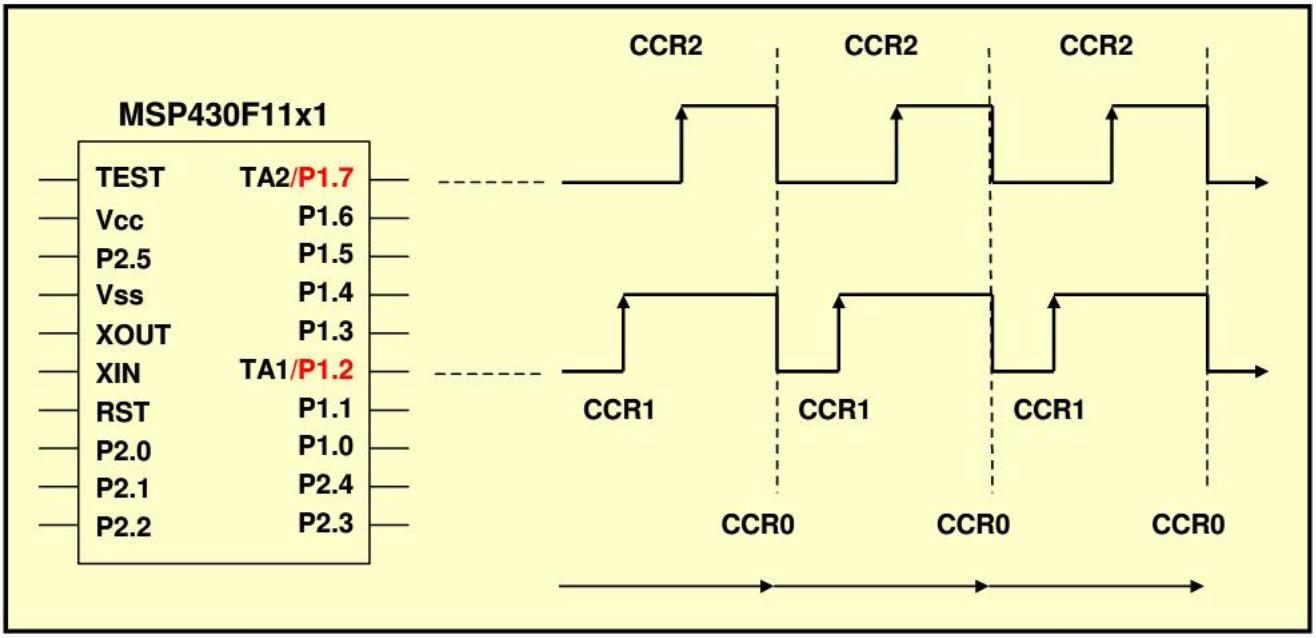
```
#pragma vector = TIMERA1_VECTOR
__interrupt void TIMERA1_ISR(void)
{
  switch(__even_in_range(TAIV,10))
  {
    case  2 :       // TACCR1 CCIFG
      P1OUT ^= 0x04; break;
    case  4 :       // TACCR2 CCIFG
      P1OUT ^= 0x02; break;
    case 10 :       // TAIFG
      P1OUT ^= 0x01; break;
   }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

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

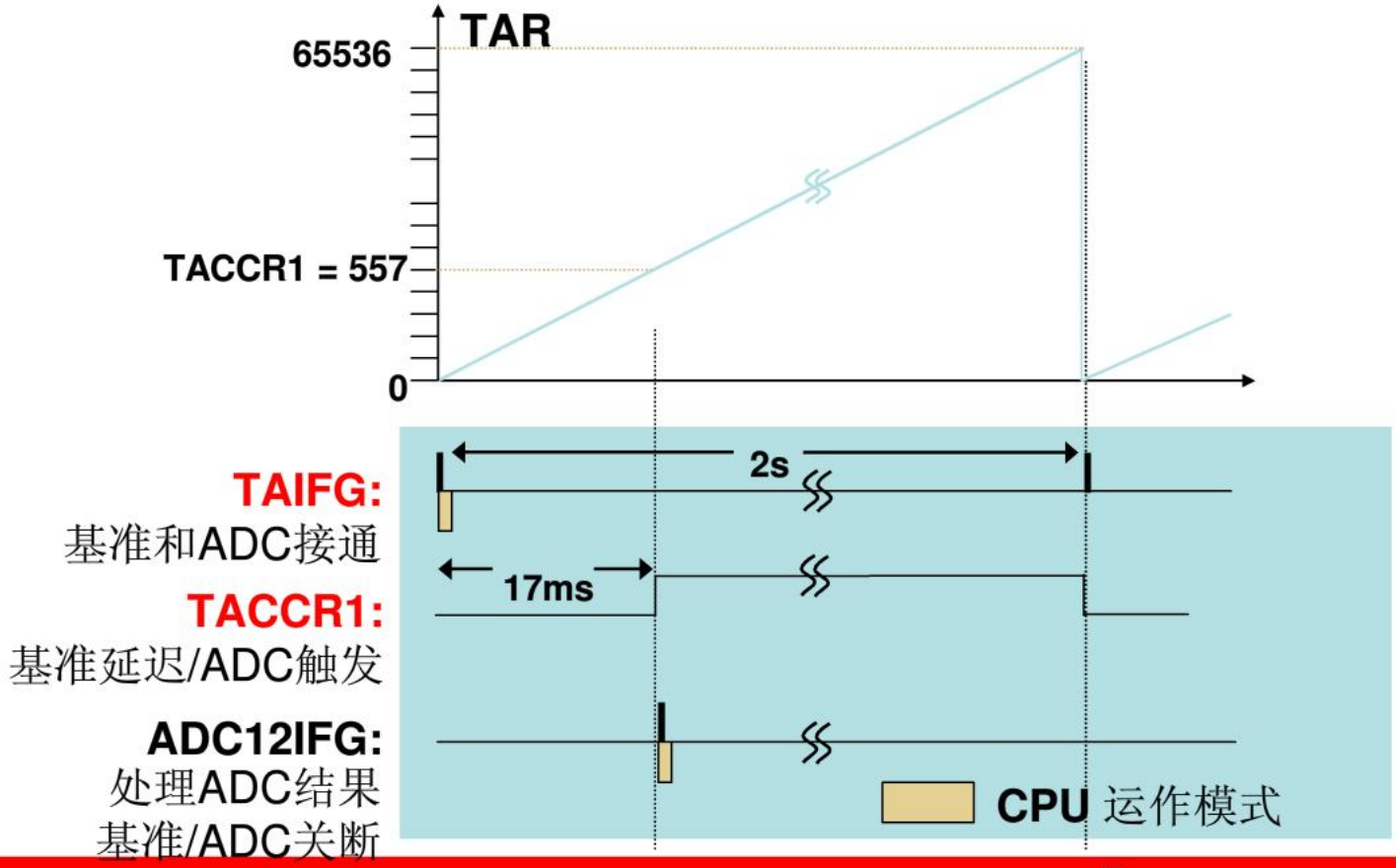
5.5 Timer_A产生PWM



- **PWM**完全自动产生
- 通过对每个**CCR**寄存器参数单独配置，可以产生不同占空比的**PWM**
- 可以在网上找到参考程序

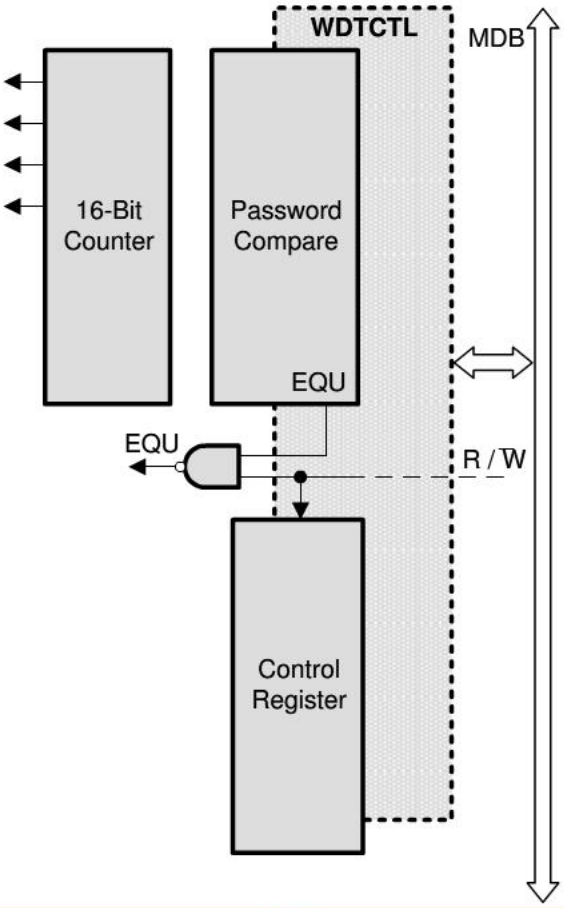
5.6 采用Timer_A直接进行硬件控制

Example: ADC12



5.7 看门狗定时器故障保险

- 所有 **MSP430** 器件均有**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

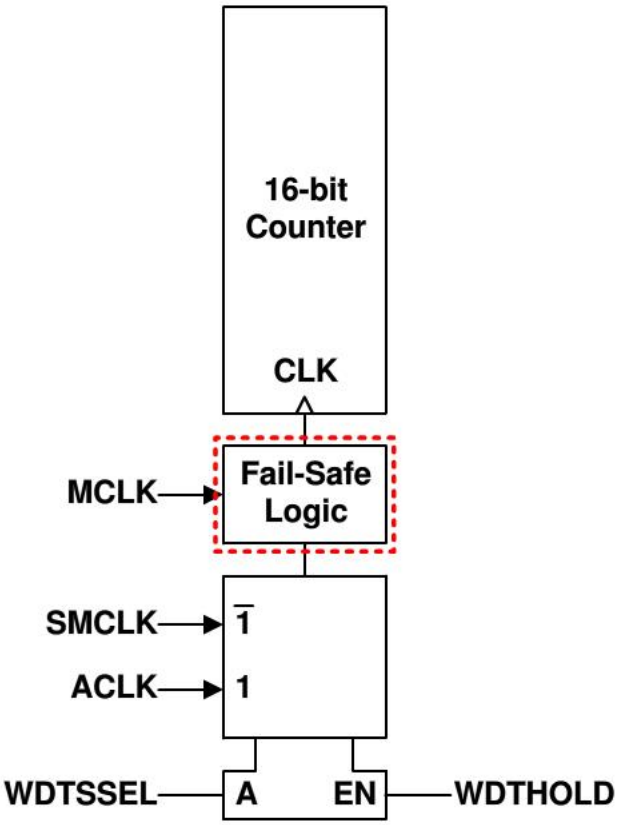


- 倘若 **ACLK/SMCLK** 发生故障，则时钟脉冲源 **= MCLK**

（***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

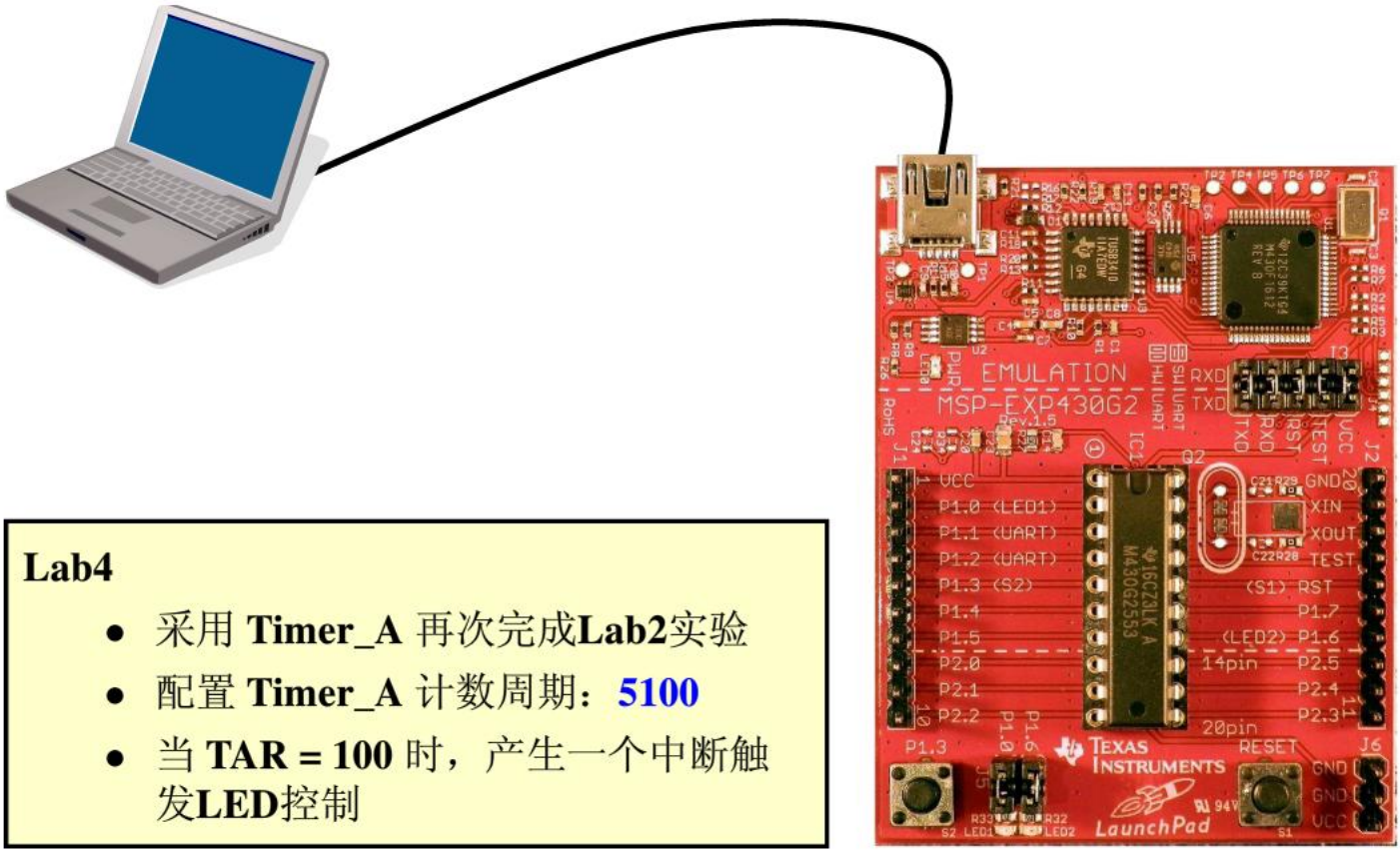
- 程序保持自身的复位!
- 程序动作反常—执行是否达到清除**WDT**的位置?
  - ✓ 设置一个靠近main() 起点的中断，以查看代码是否重新启动
- **CPU**甚至在到达第一条指令之前似乎就出现了冻结现象
  - ✓ C程序是不是具有大量初始化数据?
  - ✓ 通常只会在拥有超大RAM空间器件上出现
  - ✓ 解决方案: 在 __low_level_init() 函数中关闭看门狗

```
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

```
// 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)
```

```
// Configure TimerA
TACTL =TASSEL_1+MC_1+ TAIE;//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 = TIMER_A0_VECTOR
__interrupt void Timer_A0(void)
```

```
 // Timer A1 interrupt service routine
#pragma vector = TIMER_A1_VECTOR
__interrupt void Timer_A1(void)
```