MSP430 教程 1：MSP430 单片机系列简介

1、MSP430 单片机的发展

MSP430 系列是一个 16 位的、具有精简指令集的、超低功耗的混合型单片机，在 1996 年问世，由于它具有极低的功耗、丰富的片内外设和方便灵活的开发手段，已成为众多单片机系列中一颗耀眼的新星。回忆 MSP430 系列单片机的发展过程，可以看出有这样三个阶段：

开始阶段 从 1996 年推出 MSP430 系列开始到 2000 年初，这个阶段首先推出有 33X 、 32X 、 31X 等几个系列，而后于 2000 年初又推出了 11X 、 11X1 系列。

MSP430 的 33X 、 32X 、 31X 等系列具有 LCD 驱动模块，对提高系统的集成度较有利。每一系列有 ROM 型（ C ）、 OTP 型（ P ）、和 EPROM 型（ E ）等芯片。EPROM 型的价格昂贵，运行环境温度范围窄，主要用于样机开发。这也表明了这几个系列的开发模式，即：用户可以用 EPROM 型开发样机；用 OTP 型进行小批量生产；而 ROM 型适应大批量生产的产品。

2000 年推出了 11X/11X1 系列。这个系列采用 20 脚封装，内存容量、片上功能和 I/O 引脚数比较少，但是价格比较低廉。

这个时期的 MSP430 已经显露出了它的特低功耗等的一系列技术特点，但也有不尽如人意之处。它的许多重要特性，如：片内串行通信接口、硬件乘法器、足够的 I/O 引脚等，只有 33X 系列才具备。 33X 系列价格较高，比较适合于较为复杂的应用系统。当用户设计需要更多考虑成本时， 33X 并不一定是最适合的。而片内高精度 A/D 转换器又只有 32X 系列才有。

寻找突破，引入 Flash 技术 随着 Flash 技术的迅速发展， TI 公司也将这一技术引入 MSP430 系列中。在 2000 年 7 月推出 F13X/F14X 系列，在 2001 年 7 月到 2002 年又相继推出 F41X 、 F43X 、 F44X 这些全部是 Flash 型单片机。

F41X 单片机是目前应用比较广的单片机，它有 48 个 I/O 口， 96 段 LCD 驱动。 F43X 、 F44X 系列是在 13X 、 14X 的基础上，增加了液晶驱动器，将驱动 LCD 的段数由 3XX 系列的最多 120 段增加到 160 段。并且相应地调整了显示存储器在存储区内的地址，为以后的发展拓展了空间。

MSP430 系列由于具有 Flash 存储器，在系统设计、开发调试及实际应用上都表现出较明显的优点。这是 TI 公司推出具有 Flash 型存储器及 JTAG 边界扫描技术的廉价开发工具 MSP-FET430X110 ，将国际上先进的 JTAG 技术和 Flash 在线编程技术引入 MSP430 。

这种以 Flash 技术与 FET 开发工具组合的开发方式，具有方便、廉价、实用等优点，给用户提供了一个较为理想的样机开发方式。

另外， 2001 年 TI 公司又公布了 BOOTSTRAP 技术，利用它可在烧断熔丝以后只要几根线就可更改并运行内部的程序。这为系统软件的升级提供了又一方便的手段。 BOOTSTRAP 具有很高的保密性，口令可达到 32 个字节的长度。

蓬勃发展阶段 在前一阶段，引进新技术和内部进行调整之后，为 MSP430 的功能扩展打下了良好的基础。于是 TI 公司在 2002 年底和 2003 年期间又陆续推出了 F15X 和 F16X 系列的产品。

在这一新的系列中，有了两个方面的发展。一是从存储器方面来说，将 RAM 容量大大增加，如 F1611 的 RAM 容量增加到了 10KB 。这样一来，希望将实时操作系统（ RTOS ）引入 MSP430 的，就不会因 RAM 不够而发愁了。二是从外围模块来说，增加了 I 2 C 、 DMA 、 DAC12 和 SVS 等模块。

在 2003 年中， TI 公司还推出了专门用于电量计量的 MSP430FE42X 和用于水表、气表、热表上的具有无磁传感模块的 MSP430FW42X 单片机。我们相信由于 MSP430 的开放性的基本架构和新技术的应用，新的 MSP430 的产品品种必将会不断出现。

2、 MSP430 单片机的特点

MSP430 系列单片机的迅速发展和应用范围的不断扩大，主要取决于以下的特点。

强大的处理能力 MSP430 系列单片机是一个 16 位的单片机，采用了精简指令集（ RISC ）结构，具有丰富的寻址方式（ 7 种源操作数寻址、 4 种目的操作数寻址）、简洁的 27 条内核指令以及大量的模拟指令；大量的寄存器以及片内数据存储器都可参加多种运算；还有高效的查表处理指令；有较高的处理速度，在 8MHz 晶体驱动下指令周期为 125 ns 。这些特点保证了可编制出高效率的源程序。

在运算速度方面， MSP430 系列单片机能在 8MHz 晶体的驱动下，实现 125ns 的指令周期。 16 位的数据宽度、 125ns 的指令周期以及多功能的硬件乘法器（能实现乘加）相配合，能实现数字信号处理的某些算法（如 FFT 等）。

MSP430 系列单片机的中断源较多，并且可以任意嵌套，使用时灵活方便。当系统处于省电的备用状态时，用中断请求将它唤醒只用 6us 。

超低功耗 MSP430 单片机之所以有超低的功耗，是因为其在降低芯片的电源电压及灵活而可控的运行时钟方面都有其独到之处。

首先， MSP430 系列单片机的电源电压采用的是 1.8~3.6V 电压。因而可使其在 1MHz 的时钟条件下运行时， 芯片的电流会在 200~400uA 左右，时钟关断模式的最低功耗只有 0.1uA 。

其次，独特的时钟系统设计。在 MSP430 系列中有两个不同的系统时钟系统：基本时钟系统和锁频环（ FLL 和 FLL ）时钟系统或 DCO 数字振荡器时钟系统。有的使用一个晶体振荡器（ 32768Hz ） , 有的使用两个晶体振荡器）。由系统时钟系统产生 CPU 和各功能所需的时钟。并且这些时钟可以在指令的控制下，打开和关闭，从而实现对总体功耗的控制。

由于系统运行时打开的功能模块不同，即采用不同的工作模式，芯片的功耗有着显著的不同。在系统中共有一种活动模式（ AM ）和五种低功耗模式（ LPM0~LPM4 ）。在等待方式下，耗电为 0.7uA ，在节电方式下，最低可达 0.1uA 。

系统工作稳定 上电复位后，首先由 DCOCLK 启动 CPU ，以保证程序从正确的位置开始执行，保证晶体振荡器有足够的起振及稳定时间。然后软件可设置适当的寄存器的控制位来确定最后的系统时钟频率。如果晶体振荡器在用做 CPU 时钟 MCLK 时发生故障， DCO 会自动启动，以保证系统正常工作；如果程序跑飞，可用看门狗将其复位。

丰富的片上外围模块 MSP430 系列单片机的各成员都集成了较丰富的片内外设。它们分别是看门狗（ WDT ）、模拟比较器 A 、定时器 A （ Timer_A ）、定时器 B （ Timer_B ）、串口 0 、 1 （ USART0 、 1 ）、硬件乘法器、液晶驱动器、 10 位 /12 位 ADC 、 I 2 C 总线直接数据存取（ DMA ）、端口 O （ P0 ）、端口 1~6 （ P1~P6 ）、基本定时器（ Basic Timer ）等的一些外围模块的不同组合。其中，看门狗可以使程序失控时迅速复位；模拟比较器进行模拟电压的比较，配合定时器，可设计出 A/D 转换器； 16 位定时器（ Timer_A 和 Timer_B ）具有捕获 / 比较功能，大量的捕获 / 比较寄存器，可用于事件计数、时序发生、 PWM 等；有的器件更具有可实现异步、同步及多址访问串

行通信接口可方便的实现多机通信等应用；具有较多的 I/O 端口，最多达 6*8 条 I/O 口线； P0 、 P1 、 P2 端口能够接收外部上升沿或下降沿的中断输入； 12/14 位硬件 A/D 转换器有较高的转换速率，最高可达 200kbps ，能够满足大多数数据采集应用；能直接驱动液晶多达 160 段；实现两路的 12 位 D/A 转换；硬件 I 2 C 串行总线接口实现存储器串行扩展；以及为了增加数据传输速度，而采用直接数据传输（ DMA ）模块。 MSP430 系列单片机的这些片内外设为系统的单片解决方案提供了极大的方便。

方便高效的开发环境 目前 MSP430 系列有 OPT 型、 FLASH 型和 ROM 型三种类型的器件，这些器件的开发手段不同。对于 OPT 型和 ROM 型的器件是使用仿真器开发成功之后在烧写或掩膜芯片；对于 FLASH 型则有十分方便的开发调试环境，因为器件片内有 JTAG 调试接口，还有可电擦写的 FLASH 存储器，因此采用先下载程序到 FLASH 内，再在器件内通过软件控制程序的运行，由 JTAG 接口读取片内信息供设计者调试使用的方法进行开发。这种方式只需要一台 PC 机和一个 JTAG 调试器，而不需要仿真器和编程器。开发语言有汇编语言和 C 语言。

MSP430 单片机目前主要以 FLASH 型为主。

适应工业级运行环境 MSP430 系列器件均为工业级的，运行环境温度为 -40~ 85 摄氏度 ，所设计的产品适合用于工业环境下。

3. MAP430 系列与 89C 51 系列的比较

我国的多数读者对 89C 51 系列的单片机是很熟悉的，为了加深对 MSP430 系列单片机的认识，我们不妨将两者进行一下比较。

首先， 89C 51 单片机是 8 位单片机。其指令是采用的被称为“ CISC ”的复杂指令集，共具有 111 条指令。而 MSP430 单片机是 16 位的单片机，采用了精简指令集（ RISC ）结构，只有简洁的 27 条指令，大量的指令则是模拟指令，众多的寄存器以及片内数据存储器都可参加多种运算。这些内核指令均为单周期指令，功能强，运行的速度快。

其次， 89C 51 单片机本身的电源电压是 5 伏，有两种低功耗方式：待机方式和掉电方式。正常情况下消耗的电流为 24mA ，在掉电状态下，其耗电电流仍为 3mA ；即使在掉电方式下，电源电压可以下降到 2V ，但是为了保存内部 RAM 中的数据，还需要提供约 50uA 的电流。而 MSP430 系列单片机在低功耗方面的优越之处，则是 89C 51 系列不可比拟的。正因为如此， MSP430 更适合应用于使用电池供电的仪器、仪表类产品中。

再者， 89C 51 系列单片机由于其内部总线是 8 位的，其内部功能模块基本上都是 8 位的虽然经过各种努力其内部功能模块有了显著增加，但是受其结构本身的限制很大，尤其模拟功能部件的增加更显困难。 MSP430 系列其基本架构是 16 位的，同时在其内部的数据总线经过转换还存在 8 位的总线，在加上本身就是混合型的结构，因而对它这样的开放型的架构来说，无论扩展 8 位的功能模块，还是 16 位的功能模块，即使扩展模 / 数转换或数 / 模转换这类的功能模块也是很方便的。这也就是为什么 MSP430 系列产品和其中功能部件迅速增加的原因。

最后，就是在开发工具上面。对于 89C51 来说，由于它是最早进入中国的单片机，人们对它在熟悉不过了，再加上我国各方人士的努力，创造了不少适合我们使用的开发工具。但是如何实现在线编程还是一个很大的问题。对于 MSP430 系列而言，由于引进了 Flash 型程序存储器和 JTAG 技术，不仅使开发工具变得简便，而且价格也相对低廉，并且还可以实现在线编程。

4.MSP430 系列的内部结构概述

MSP430 系列器件包含 CPU 、程序存储器 (ROM 、 ROM 和 Flash ROM) 、数据存储器 (RAM) 、运行控制、外围模块和振荡器和倍频器等主要功能模块。其基本结构如 图 1 所示。可以看出， MSP430 内部包含了计算机所有部件，是一个真正的单片机 (微控制器 MCU) 。 图 2 所示。在 16 个寄存器中，程序计数器 PC 、堆栈指针 SP 、状态寄存器 SR 和常数发生器 CGl 、 CG2 这 4 个寄存器有特殊用途。除了 R3 / CG2 和 R2 / CGl 外，所有寄存器都可作为通用寄存器来用于所有指令操作。常数发生器是为指令执行时提供常数的，而不是用于存储数据的。对 CGl 、 CG2 访问的寻址模式可以区分常数的数据。

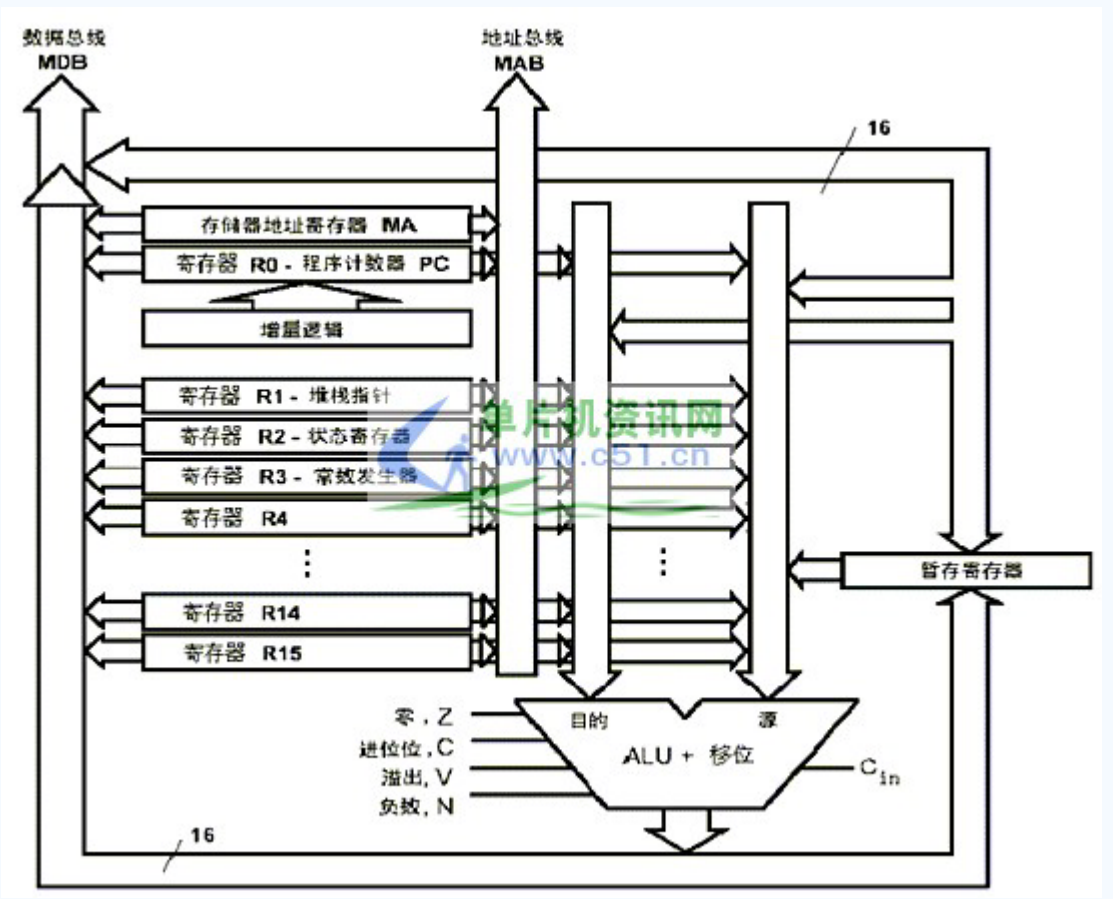


图 **2** 图 3 所示。 表 1 中。 图 3）。 图 4 所示为外围模块的连接总线示意图。从图中可以看出，外围模块可分为字（ 16 位）模块和字节（ 8 位）模块两种。对大多数外围模块， MAB 通常是 16 位， MDB 是 8 位或 16 位。

CPU CPU 由一个 16 位的 ALU 、 16 个寄存器和一套指令控制逻辑组成，其逻辑简图如

在 CPU 内部有一组 16 位数据总线和 16 位的地址总线； CPU 运行正交设计、对模块高度透明的精简指令集； PC 、 SR 和 SP 配合精简指令组所实现的控制，使应用开发可实现复杂的寻址模式和软件算法。

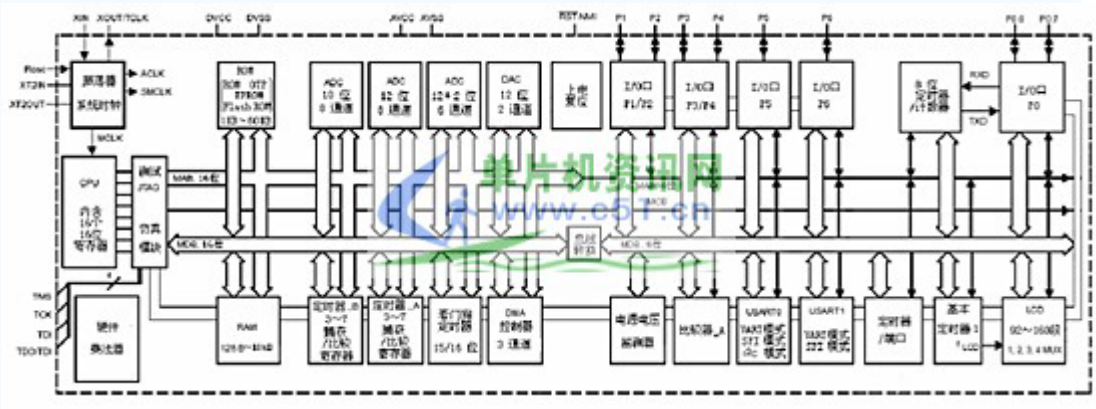


图 1

存储器 MSP430 系列采用“冯—纽曼结构”。因此， RAM 、 ROM 和全部外围模块都位于同一个地址空间内，即用一个公共的空间对全部功能模块进行寻址。支持外部扩展存储器是将来性能增强的目标。特殊功能寄存器及外围模块安排在 000H ～ 1FFH 区域； RAM 和 ROM 共享 0200H ～ FFFFH 区域，数据存储器（ RAM ）的起始地址是 0200H 。

存储器与 CPU 及存储器数据总线 (MDB) 、存储器地址总线 (MAB) 的连接关系如

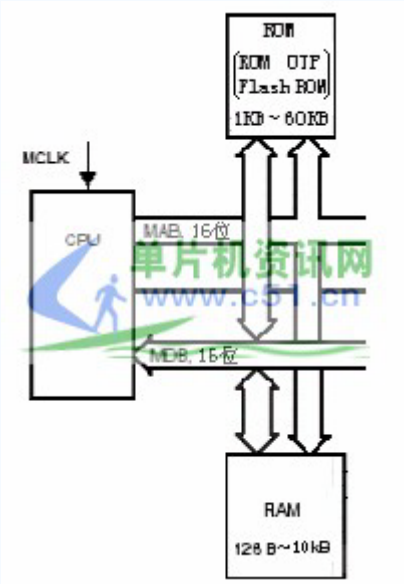


图 3

（ 1 ）程序存储器 MSP430 系列程序存储器的类型有 ROM 、 OTP 和 Flash ROM 三种，存储器的类型和容量示于本刊网站的

ROM 的容量在 1KB ～ 60KB 之间；对于 Flash 型的芯片，内部还集成有两段 128B （共 256B ）的信息存储器以及 1KB 存放自举程序的自举存储器（ BOOT ROM ）；对代码存储器的访问总是以字形式取得代码，而对数据可以用字或字节方式访问。每次访问需要 16 条数据总线（ MDB ）和访问当前存储器模块所需的地址总线（ MAB ）；存储器模块由模块允许信号自动选中。最低的 64KB 空间的顶部 16 个字，即 0FFFFH ～ 0FFE0H ，保留存放复位和中断的向量；在程序存储器中还可以存放表格数据，以实现查表处理等应用；程序对程序存储器可以任意读取，但不能写入。

（ 2 ） 数据存储器 数据存储器 (RAM) 经两条总线与 CPU 相连，即存储器地址总线 MAB 和存储器数据总线 MDB （见

数据存储器可以以字或字节宽度集成在片内，其容量在 128B ～ 10KB 之间；所有指令可以对字节或字进行操作。但是对堆栈和 PC 的操作是按字宽度进行的，寻址时必须对准偶地址。

运行控制 MSP430 系列微控制器的运行主要受控于存储在特殊寄存器 (SFR) 中的信息。不同 SFR 中的位可以允许中断，以支持取决于中断标志状态的软件以及定义外围模块的工作模式。

禁止外围模块，停止它的功能，可以减少电流消耗，而所有存储在模块寄存器中的数据仍被保留。外围模块的工作模式可以用 SFR 的特定位置来标明。

外围模块 外围模块包括基本定时器（ Basic Timer ）、 16 位定时器 (Timer_A 及 Timer_B) 、 ADC 转换器、 I/O 端口、异步及同步串行通讯口（ USART ）以及液晶显示驱动模块等。

外围模块经 MAB 、 MDB 与 CPU 相连。

字节（ 8 位）模块的数据总线是 8 位的，需经总线转换电路与 16 位的 CPU 相连。这些模块的数据交换毫无例外地要用字节指令处理；对字（ 16 位）模块，其数据总线是 16 位的，无需经过转换而直接与 CPU 的 16 位数据总线相连。模块的操作指令就没有任何限制。 MSP430 系列所包含的字节（ 8 位）模块和字（ 16 位）模块，请参看本刊网站上的表 2 和表 3 。

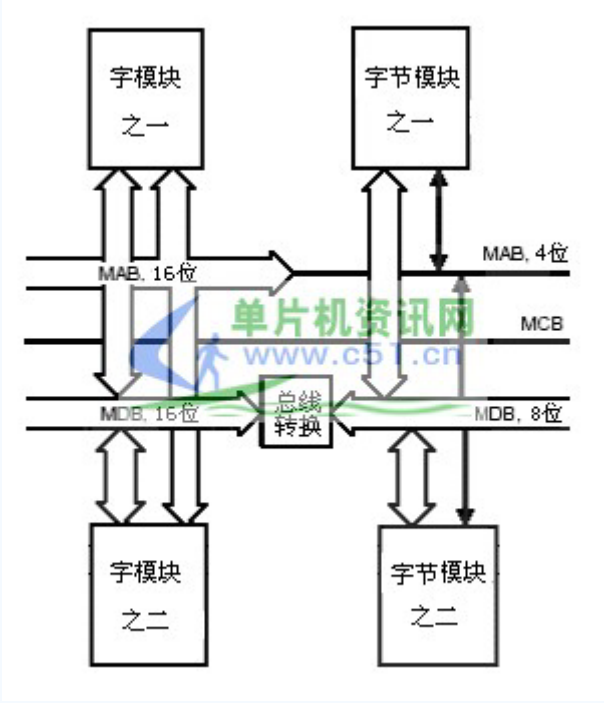


图 4

振荡器和时钟发生器 振荡器 LFXT1 （ LF ）是专门为通用的低功耗 32768 Hz 时钟晶振设计的。除了晶体外接外，所有的模拟元件都集成在片内。但是也可以用一个高速的晶振工作，这时需要外接负载电容。

对于 F13X 、 F14X 、 F15X 和 F16X 以及 F4XX 系列，片内还有一个可接入高速晶振的 XT2 振荡器。除了晶体振荡器之外， F13X 、 F14X 、 F15X 和 F16X 系列都有一

个数字控制 RC 振荡器（ DCO ），用它实现对振荡器的数字控制和频率调节；对于 F4XX 系列，将晶振频率用一个锁频环电路 (FLL 或 FLL ＋) 进行倍频。 FLL 或 FLL ＋在上电后以最低频率开始工作，并通过控制一个数控振荡器 (DCO) 来调整到适当的频率。供处理器工作的时钟发生器的频率固定在晶振的倍频上，并提供时钟信号 MCLK 。

外围模块及 CPU 的时钟源选择非常灵活。可以用以实现各种低功耗模式下的运行。

表 1 MSP430 中的存储器

		类 型			容 量	
	型号	OTP	ROM	Flash ROM	ROM(kB)	RAM(字节)
1	X1101		MSP 430C 11101	MSP 430F 11101A	1	128
2	X1111		MSP 430C 1111	MSP430F1111A	2	128
3	X112	MSP430P112			4	256
4	X1121			MSP 430F 1121A	4	256
5	X1122			MSP430F1122	4	256
6	X1132			MSP430F1132	8	256

7	X122			MSP430F122	4	256
8	X 122 2			MSP430F1222	4	256
9	X123			MSP430F123	8	256
10	X1232			MSP430F1232	8	256
11	X133			MSP 430F 133	8	256
12	X1331		MSP430C1331		8	256
13	X135			MSP 430F 135	16	512
14	X1351		MSP430C1351		16	512
15	X147			MSP 430F 147	32	1 K
16	X1471			MSP 430F 1471	32	1 K
17	X148			MSP 430F 148	48	2 K
18	X1481			MSP 430F 1481	48	2 K

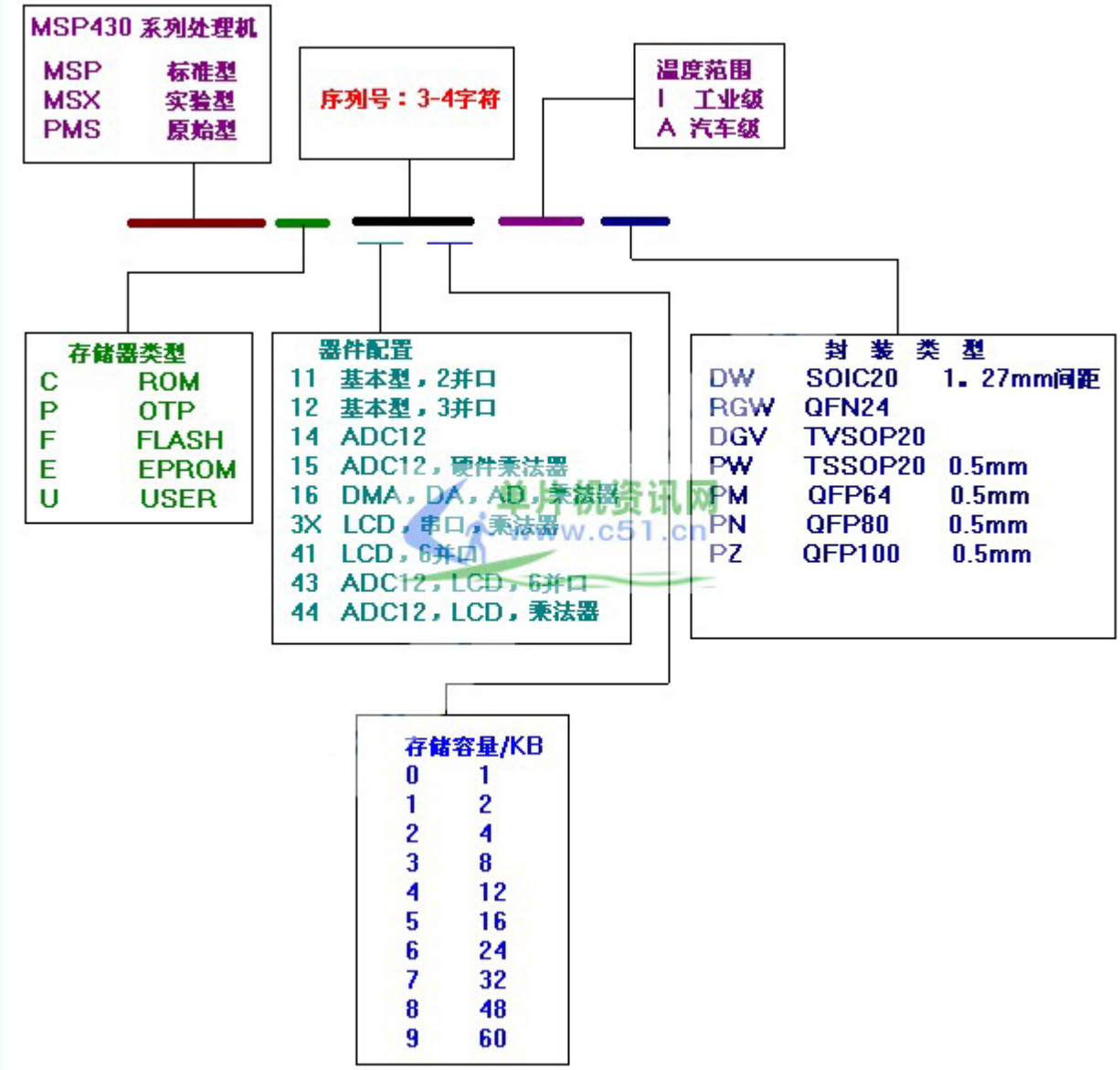
19	X149			MSP 430F 149	60	2 K
20	X1491			MSP 430F 1491	60	2 K
21	X155			MSP 430F 155	16	512
22	X156			MSP 430F 156	24	1 K
23	X157			MSP 430F 157	32	1 K
24	X167			MSP 430F 167	32	1 K
25	X168			MSP 430F 168	48	2 K
26	X169			MSP 430F 169	60	2 K
27	X1610			MSP 430F 1610	32	5 K
28	X1611			MSP 430F 1611	48	10 K
29	X311	MSP 430C 311			2	128

30	X 312		MSP 430C 312		4	256
31	X 313	MSP430P313	MSP 430C 313		8	256
32	X 314		MSP 430C 314		12	512
33	X 315	MSP430P315	MSP 430C 315		16	512
34	X323		MSP 430C 323		8	256
35	X325	MSP430P325	MSP 430C 325		16	512
36	X336		MSP 430C 336		24	1 K
37	X337	MSP430P337	MSP 430C 337		32	1 K
38	X412		MSP 430C 412	MSP 430F 412	4	256
39	X413		MSP 430C 413	MSP 430F 413	8	256
40	X423		MSP430FW423	MSP430FE423	8	256

41	X425		MSP430FW425	MSP430FE425	16	512
42	X427		MSP430FW427	MSP430FE427	32	1 K
43	X435			MSP 430F 435	16	512
44	X436			MSP 430F 436	24	1 K
45	X437			MSP 430F 437	32	2 K
46	X447			MSP 430F 447	32	1 K
47	X448			MSP 430F 448	48	2 K
48	X449			MSP 430F 449	60	2 K

MSP430 教程 2：MSP430 单片机命名规则

MSP430 单片机命名规则



MSP430 教程 3：MSP430 单片机硬件知识

MSP430 单片机是 TI 公司 1996 年开始推向市场的超低功耗微处理器，另外他还集成了很多模块功能，从而使得用一片 MSP430 芯片可以完成多片芯片才能完成的功能，大大缩小了产品的体积与成本。如今，MSP430 单片机已经用于各个领域，尤其是仪器仪表、监测、医疗器械以及汽车电子等领域。

下面来说一下它的主要特点：（1）低电源电压范围，1.8～3.6V。（2）超低功耗，拥有 5 种低功耗模式(以后会详细介绍)。（3）灵活的时钟使用模式。（4）高速的运算能力，16 位 RISC 架构，125ns 指令周期。（5）丰富的功能模块，这些功能模块包括：A：多通道 10－14 位 AD 转换器；B：双路 12 位 DA 转换器；C：比较器；D：液晶驱动器；E：电源电压检测；F：串行口 USART(UART/SPI）；G：硬件乘法器；H：看门狗定时器，多个 16 位、8 位定时器（可进行捕获，比较，PWM 输出）；I：DMA 控制器。（6）FLASH 存储器，不需要额外的高电压就在运行种由程序控制写擦欧哦和段的擦除；（7）MSP430 芯片上包括 JTAG 接口，仿真调试通过一个简单的 JTAG 接口转换器就可以方便的实现如设置断点、单步执行、读写寄存器等调试；（8）快速灵活的变成方式，可通过 JTAG 和 BSL 两种方式向 CPU 内装在程序。

关于他的内存器结构，在匠人的博客里已有详细的介绍，大家去看就是了。在这里我主要说说 MSP430 单片机的复位吧。

MSP430 的复位信号有 2 种：上电复位信号（POR）、上电清除信号（PUC）。还有能够触发 POR 和 PUC 的信号：5 种来在看门狗，1 种来自复位管脚，1 种来自写 FLASH 键值出现错误所产生的信号。

POR 信号只在 2 种情况下发生：（1）微处理上电；（2）RST/NMI 管脚上产生低电平时系统复位。

PUC 信号产生的条件：（1）POR 信号产生；（2）看门狗有效时，看门狗定时器溢出；（3）写看门狗定时器安全键值出现错误；（4）写 FLASH 存储器安全键值出现错误。

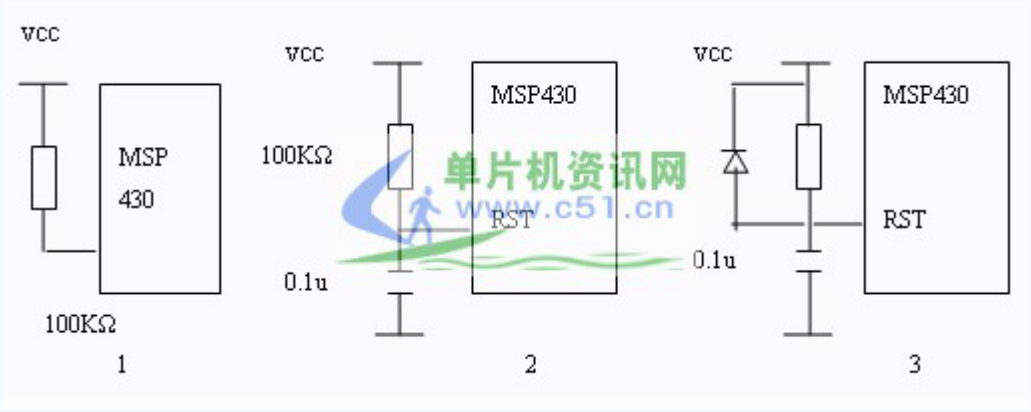
POR 和 PUC 两者的关系：POR 信号的产生会导致系统复位并产生 PUC 信号。而 PUC 信号不会引起 POR 信号的产生。

无论是 POR 信号还是 PUC 信号触发的复位，都会使 MSP430 从地址 0xFFFE 处读取复位中断向量，程序从中断向量所指的地址处开始执行。触发 PUC 信号的条件中，除了 POR 产生触发 PUC 信号外，其他的豆科一通过读取相应的中断向量来判断是何种原因引起的 PUC 信号，以便作出相应的处理。

系统复位（指 POR）后的状态为：（1）RST/NMI 管脚功能被设置为复位功能；（2）所有 I/O 管脚被设置为输入；（3）外围模块被初始化，其寄存器值为相关手册上的默认值；（4）状态寄存器 SR 复位；（5）看门狗激活，进入工作模式；（6）程序计数器 PC 载入 0xFFFE 处的地址，微处理器从此地址开始执行程序。

典型的复位电路有一下 3 种：

（1） 在 RST/NMI 管脚上接 100K 欧的上拉电阻。（2）在（1）的基础上再接 0.1uf 的电容，电容的一端接地，可以使复位更加可靠。（3）再（2）的基础上，再在电阻上并接一个型号为 IN4008 的二极管，可以可靠的实现系统断电后立即上电。



MSP430 单片机的时钟系统

MSP430 根据型号的不同最多可以选择使用 3 个振荡器。我们可以根据需要选择合适的振荡频率，并可以在不需要时随时关闭振荡器，以节省功耗。这 3 个振荡器分别为：

（1）DCO 数控 RC 振荡器。它在芯片内部，不用时可以关闭。DCO 的振荡频率会受周围环境温度和 MSP430 工作电压的影响，且同一型号的芯片所产生的频率也不相同。但 DCO 的调节功能可以改善它的性能，他的调节分为以下 3 步：a：选择 BCSCTL1.RSELx 确定时钟的标称频率；b：选择 DCOCTL.DCOx 在标称频率基础上分段粗调；c：选择 DCOCTL.MODx 的值进行细调。

（2）LFXT1 接低频振荡器。典型为接 32768HZ 的时钟振荡器，此时振荡器不需要接负载电容。也可以接 450KHZ~8MHZ 的标准晶体振荡器，此时需要接负载电容。

（3）XT2 接 450KHZ~8MHZ 的标准晶体振荡器。此时需要接负载电容，不用时可以关闭。

低频振荡器主要用来降低能量消耗，如使用电池供电的系统，高频振荡器用来对事件做出快速反应或者供 CPU 进行大量运算。

MSP430 的 3 种时钟信号：MCLK 系统主时钟；SMCLK 系统子时钟；ACLK 辅助时钟。

（1）MCLK 系统主时钟。除了 CPU 运算使用此时钟以外，外围模块也可以使用。MCLK 可以选择任何一个振荡器所产生的时钟信号并进行 1、2、4、8 分频作为其信号源。

（2）SMCLK 系统子时钟。供外围模块使用。并在使用前可以通过各模块的寄存器实现分频。SMCLK 可以选择任何一个振荡器所产生的时钟信号并进行 1、2、4、8 分频作为其信号源。

（3）ACLK 辅助时钟。供外围模块使用。并在使用前可以通过各模块的寄存器实现分频。但 ACLK 只能由 LFXT1 进行 1、2、4、8 分频作为信号源。

PUC 复位后，MCLK 和 SMCLK 的信号源为 DCO，DCO 的振荡频率为 800KHZ。ACLK 的信号源为 LFXT1。

MSP430 内部含有晶体振荡器失效监测电路，监测 LFXT1（工作在高频模式）和 XT2 输出的时钟信号。当时钟信号丢失 50us 时，监测电路捕捉到振荡器失效。如果 MCLK 信号来自 LFXT1 或者 XT2，那么 MSP430 自动把 MCLK 的信号切换为 DCO，这样可以保证程序继续运行。但 MSP430 不对工作在低频模式的 LFXT1 进行监测。

5 种低功耗模式

5 种低功耗模式分别为 LPM0~LPM4(LOW POWER MODE)，CPU 的活动状态称为 AM(ACTVE MODE)模式。其中 AM 耗电最大，LPM4 耗电最省，仅为 0.1uA。另外工作电压对功耗的影响：电压越低功耗也越低。

系统 PUC 复位后，MSP430 进入 AM 状态。在 AM 状态，程序可以选择进入任何一种低功耗模式，然后在适当的条件下，由外围模块的中断使 CPU 退出低功耗模式，返回 AM 模式，再由 AM 模式选择进入相应的低功耗模式，如此类推。

工作模式的选择由状态寄存器 SR 中的 SCG1、SCG0、OSCOFF、CPUOFF 位控制。由于在 CPU 的头文件中对 CPU 内的各寄存器和模块的各种工作模式都作了详尽的定义，所以编程时尽可能的利用就是了。如：要进入低功耗模式 0，可在程序中直接写：LPM0; 。进入低功耗模式 4，可以写：LMP4;就可以了。退出低功耗模式如下：

LPM0_EXIT; //退出低功耗模式 0

LPM4_EXIT; //退出低功耗模式 4

中断

中断是 MSP430 微处理器的一大特色，有效地利用中断可以简化程序和提高执行效率。MSP430 的几乎每个外围模块都能够产生中断，为 MSP430 针对事件（即外围模块产生的中断）进行的编程打下基础。MSP430 在没有事件发生时进入低功耗模式，事件发生时，通过中断唤醒 CPU，事件处理完毕后，CPU 再次进入低功耗状态。由于 CPU 的运算速度和退出低功耗的速度很快，所以在应用中，CPU 大部分时间都处于低功耗状态。

MSP430 的中断分为 3 种：系统复位、不可屏蔽中断、可屏蔽中断。

（1）系统复位的中断向量为 0xFFFE。

（2)不可屏蔽中断的中断向量为 0xFFFC。响应不可屏蔽中断时，硬件自动将 OFIE、NMIE、ACCVIE 复位。软件首先判断中断源并复位中断标志，接着执行用户代码。退出中断之前需要置位 OFIE、NMIE、ACCVIE，以便能够再次响应中断。需要特别注意点：置位 OFIE、NMIE、ACCVIE 后，必须立即退出中断相应程序，否则会再次触发中断，导致中断嵌套，从而导致堆栈溢出，致使程序执行结果的无法预料。

（3）可屏蔽中断的中断来源于具有中断能力的外围模块，包括看门狗定时器工作在定时器模式时溢出产生的中断。每一个中断都可以被自己的中断控制位屏蔽，也可以由全局中断控制位屏蔽。

多个中断请求发生时，响应最高优先级中断。响应中断时，MSP430 会将不可屏蔽中断控制位 SR.GIE 复位。因此，一旦响应了中断，即使有优先级更高的可屏蔽中断出现，也不会中断当前正在响应的中断，去响应另外的中断。但 SR.GIE 复位不影响不可屏蔽中断，所以仍可以接受不可屏蔽中断的中断请求。

中断响应的过程：（1）如果 CPU 处于活动状态，则完成当前指令；（2）若 CPU 处于低功耗状态，则退出低功耗状态；（3）将下一条指令的 PC 值压入堆栈；（4）将状态寄存器 SR 压入堆栈；（5）若有多个中断请求，响应最高优先级中断；（6）单中断源的中断请求标志位自动复位，多中断源的标志位不变，等待软件复位；（7）总中断允许位 SR.GIE 复位。SR 状态寄存器中的 CPUOFF、OSCOFF、SCG1、V、N、Z、C 位复位；（8）相应的中断向量值装入 PC 寄存器，程序从此地址开始执行。

中断返回的过程：（1）从堆栈中恢复 PC 值，若响应中断前 CPU 处于低功耗模式，则可屏蔽中断仍然恢复低功耗模式；（2）从堆栈中恢复 PC 值，若响应中断前 CPU 不处于低功耗模式，则从此地址继续执行程序。

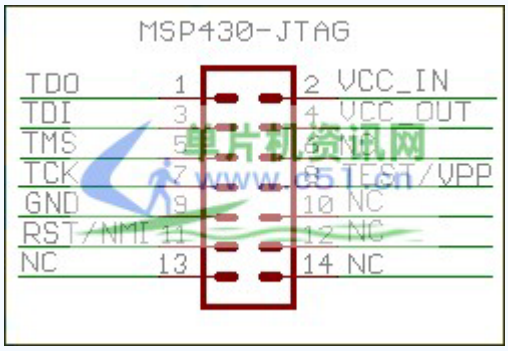
MSP430 各系列的中断向量表请查阅相关资料。

MSP430 教程 4：MSP430 单片机调试接口和 JTAG 仿真器原理图

MSP430 单片机调试接口简介

MSP430F1、F2、F4 系列产品中，采用的是 4 线 JTAG 接口。也即 TMS(模式选择)、TCK(JTAG 时钟信号)、TDO(数据输出)、TDI(数据输入)。

在 4 线制的 JTAG 接口中，TI 公司有定义一个常规的 14pin 接口方式，如下图：

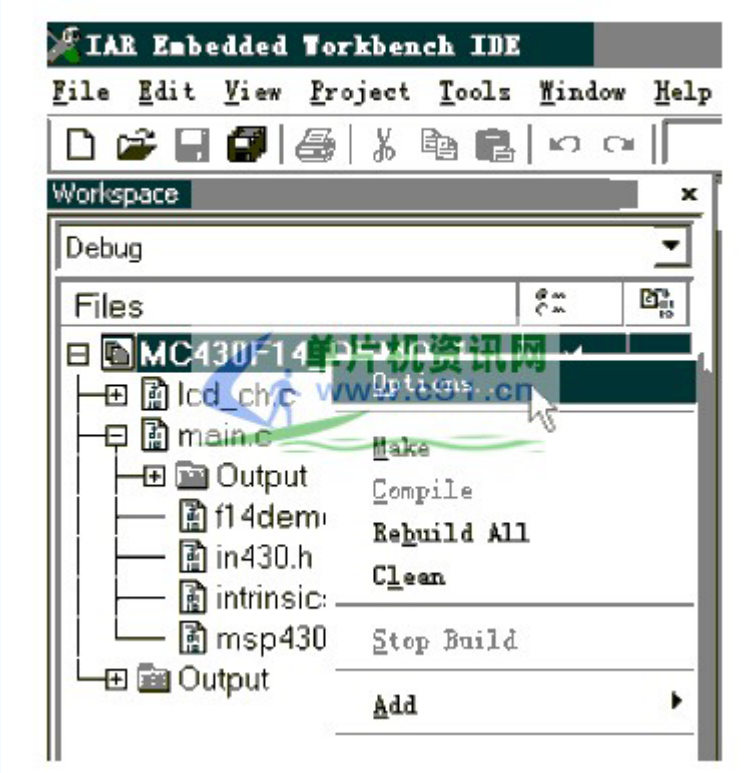


而在 MSP430F2 系列产品中，包含了两种 JATG 接口界面，一种是上述所说的 4 线制 JTAG。另一种是 MSP430F20xx 系列产品中名为“Spy Bi-Wire”的调试接口，此接口方式采用是 2 线制。分别为 SBWTCK(时钟)、SBWTDO(数据线)，加上 GND、VCC 两引脚此接口只需 4 根引线。目前支持 2 线制接口的仿真器有 TI eZSP430 USB 接口仿真器。

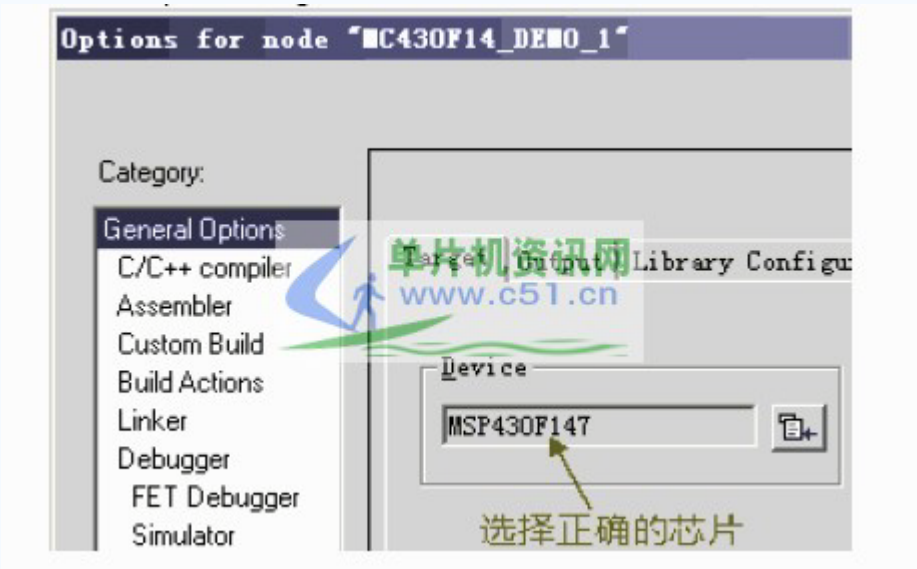
MSP430 JTAG 仿真器原理图

MSP430 教程 5：MSP430 单片机 IAR WE430 使用指南

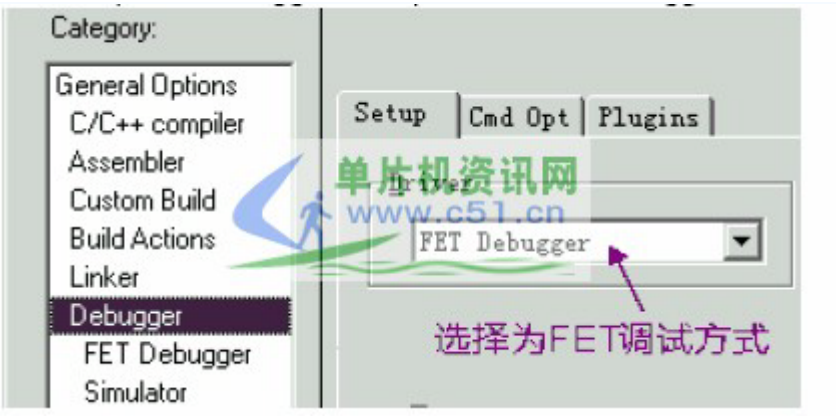
1- 建立工程,从时工程选项. Project/Options



2- Project/Options/General Options / target /Device=MSP 430F147



3- Project/Options/General Options / Debugger/Setup/Driver=FET Debugger



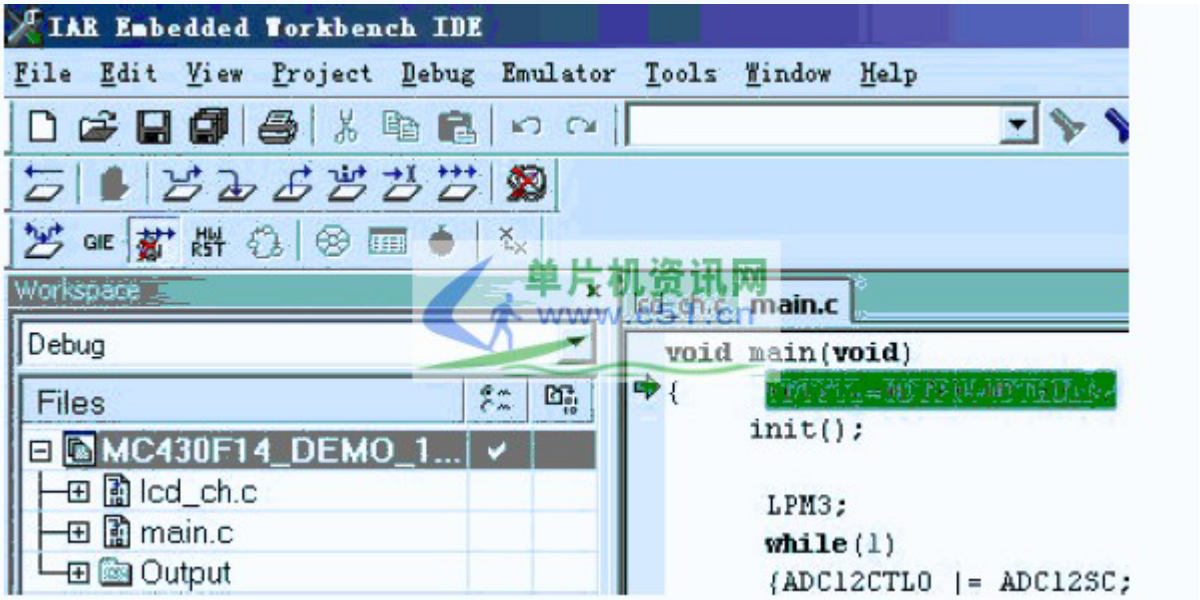
4- Project/Options/General Options / FET Debugger/Setup/Setup/ Connec on/Lpt=LPT1

如果使用是并口则选用 LPT,如果是冰河 USB 接口仿真器则选 J-link，如果是 TI USB 仿真器则选择 TI USB FET。

到此，IAR WE430 三个必要选项已设置正确。其它的暂时不必去选择。

5- 先对工程进行编译，然后可以将程序进行下载到目标板上。

6- 开始仿真或直接运行...



MSP430 教程 6：MSP430 寄存器中文注释

MSP430 寄存器中文注释---P1/2 口（带中断功能）

```
/************************************************************
* DIGITAL I/O Port1/2 寄存器定义 有中断功能
************************************************************/
#define P1IN_ 0x0020 /* P1 输入寄存器 */
const sfrb P1IN = P1IN_;
#define P1OUT_ 0x0021 /* P1 输出寄存器 */
sfrb P1OUT = P1OUT_;
#define P1DIR_ 0x0022 /* P1 方向选择寄存器 */
sfrb P1DIR = P1DIR_;
#define P1IFG_ 0x0023 /* P1 中断标志寄存器*/
sfrb P1IFG = P1IFG_;
#define P1IES_ 0x0024 /* P1 中断边沿选择寄存器*/
sfrb P1IES = P1IES_;
#define P1IE_ 0x0025 /* P1 中断使能寄存器 */
sfrb P1IE = P1IE_;
#define P1SEL_ 0x0026 /* P1 功能选择寄存器*/
sfrb P1SEL = P1SEL_;
#define P2IN_ 0x0028 /* P2 输入寄存器 */
const sfrb P2IN = P2IN_;
#define P2OUT_ 0x0029 /* P2 输出寄存器 */
sfrb P2OUT = P2OUT_;
#define P2DIR_ 0x002A /* P2 方向选择寄存器 */
sfrb P2DIR = P2DIR_;
#define P2IFG_ 0x002B /* P2 中断标志寄存器 */
sfrb P2IFG = P2IFG_;
#define P2IES_ 0x002C /* P2 中断边沿选择寄存器 */
sfrb P2IES = P2IES_;
#define P2IE_ 0x002D /* P2 中断使能寄存器 */
sfrb P2IE = P2IE_;
#define P2SEL_ 0x002E /* P2 功能选择寄存器 */
sfrb P2SEL = P2SEL_;
```

MSP430 寄存器中文注释---P3/4 口 （无中断功能）

```
  /************************************************************
* DIGITAL I/O Port3/4 寄存器定义 无中断功能
************************************************************/
  #define P3IN_ 0x0018 /* P3 输入寄存器 */
const sfrb P3IN = P3IN_;
#define P3OUT_ 0x0019 /* P3 输出寄存器 */
sfrb P3OUT = P3OUT_;
#define P3DIR_ 0x001A /* P3 方向选择寄存器 */
sfrb P3DIR = P3DIR_;
#define P3SEL_ 0x001B /* P3 功能选择寄存器*/
sfrb P3SEL = P3SEL_;
  #define P4IN_ 0x001C /* P4 输入寄存器 */
const sfrb P4IN = P4IN_;
#define P4OUT_ 0x001D /* P4 输出寄存器 */
sfrb P4OUT = P4OUT_;
#define P4DIR_ 0x001E /* P4 方向选择寄存器 */
sfrb P4DIR = P4DIR_;
#define P4SEL_ 0x001F /* P4 功能选择寄存器 */
sfrb P4SEL = P4SEL_;
  /************************************************************
* DIGITAL I/O Port5/6 I/O 口寄存器定义 PORT5 和 6 无中断功能
  ************************************************************/
  #define P5IN_ 0x0030 /* P5 输入寄存器 */
const sfrb P5IN = P5IN_;
#define P5OUT_ 0x0031 /* P5 输出寄存器*/
sfrb P5OUT = P5OUT_;
#define P5DIR_ 0x0032 /* P5 方向选择寄存器*/
sfrb P5DIR = P5DIR_;
#define P5SEL_ 0x0033 /* P5 功能选择寄存器*/
sfrb P5SEL = P5SEL_;
  #define P6IN_ 0x0034 /* P6 输入寄存器 */
const sfrb P6IN = P6IN_;
#define P6OUT_ 0x0035 /* P6 输出寄存器*/
sfrb P6OUT = P6OUT_;
#define P6DIR_ 0x0036 /* P6 方向选择寄存器*/
sfrb P6DIR = P6DIR_;
```

```
#define P6SEL_ 0x0037 /* P6 功能选择寄存器*/
sfrb P6SEL = P6SEL_;
```

MSP430 寄存器中文注释--- 硬件乘法器

```
/************************************************************
硬件乘法器的寄存器定义
************************************************************/
#define MPY_ 0x0130 /* 无符号乘法 */
sfrw MPY = MPY_;
#define MPYS_ 0x0132 /* 有符号乘法*/
sfrw MPYS = MPYS_;
#define MAC_ 0x0134 /* 无符号乘加 */
sfrw MAC = MAC_;
#define MACS_ 0x0136 /* 有符号乘加 */
sfrw MACS = MACS_;
#define OP2_ 0x0138 /* 第二乘数 */
sfrw OP2 = OP2_;
#define RESLO_ 0x013A /* 低 6 位结果寄存器 */
sfrw RESLO = RESLO_;
#define RESHI_ 0x013C /* 高 6 位结果寄存器 */
sfrw RESHI = RESHI_;
#define SUMEXT_ 0x013E /*结果扩展寄存器 */
const sfrw SUMEXT = SUMEXT_;
```

MSP430 寄存器中文注释---看门狗和定时器

```
/************************************************************
* 看门狗定时器的寄存器定义
************************************************************/
#define WDTCTL_ 0x0120
sfrw WDTCTL = WDTCTL_;
#define WDTIS0 0x0001 /*选择 WDTCNT 的四个输出端之一*/
#define WDTIS1 0x0002 /*选择 WDTCNT 的四个输出端之一*/
#define WDTSSEL 0x0004 /*选择 WDTCNT 的时钟源*/
#define WDTCNTCL 0x0008 /*清除 WDTCNT 端: 为 1 时 从 0 开始计数*/
#define WDTTMSEL 0x0010 /*选择模式 0: 看门狗模式; 1: 定时器模式*/
#define WDTNMI 0x0020 /*选择 NMI/RST 引脚功能 0:为 RST; 1:为 NMI*/
#define WDTNMIES 0x0040 /*WDTNMI=1 时.选择触发延 0:为上升延 1:为下降延*/
#define WDTHOLD 0x0080 /*停止看门狗定时器工作 0:启动;1:停止*/
#define WDTPW 0x5A00 /* 写密码:高八位*/
```

```
/* SMCLK= 1MHz 定时器模式 */
#define WDT_MDLY_32 WDTPW WDTTMSEL WDTCNTCL /* TSMCLK*2POWER15=32ms 复位状态 */
#define WDT_MDLY_8 WDTPW WDTTMSEL WDTCNTCL WDTIS0 /* TSMCLK*2POWER13=8.192ms " */
#define WDT_MDLY_0_5 WDTPW WDTTMSEL WDTCNTCL WDTIS1 /* TSMCLK*2POWER9=0.512ms " */
#define WDT_MDLY_0_064 WDTPW WDTTMSEL WDTCNTCL WDTIS1 WDTIS0 /* TSMCLK*2POWER6=0.512ms " */
/* ACLK=32.768KHz 定时器模式*/
#define WDT_ADLY_1000 WDTPW WDTTMSEL WDTCNTCL WDTSSEL /* TACLK*2POWER15=1000ms " */
#define WDT_ADLY_250 WDTPW WDTTMSEL WDTCNTCL WDTSSEL WDTIS0 /* TACLK*2POWER13=250ms " */
#define WDT_ADLY_16 WDTPW WDTTMSEL WDTCNTCL WDTSSEL WDTIS1 /* TACLK*2POWER9=16ms " */
#define WDT_ADLY_1_9 WDTPW WDTTMSEL WDTCNTCL WDTSSEL WDTIS1 WDTIS0 /* TACLK*2POWER6=1.9ms " */
/* SMCLK=1MHz 看门狗模式 */
#define WDT_MRST_32 WDTPW WDTCNTCL /* TSMCLK*2POWER15=32ms 复位状态 */
#define WDT_MRST_8 WDTPW WDTCNTCL WDTIS0 /* TSMCLK*2POWER13=8.192ms " */
#define WDT_MRST_0_5 WDTPW WDTCNTCL WDTIS1 /* TSMCLK*2POWER9=0.512ms " */
#define WDT_MRST_0_064 WDTPW WDTCNTCL WDTIS1 WDTIS0 /* TSMCLK*2POWER6=0.512ms " */
/* ACLK=32KHz 看门狗模式 */
#define WDT_ARST_1000 WDTPW WDTCNTCL WDTSSEL /* TACLK*2POWER15=1000ms " */
#define WDT_ARST_250 WDTPW WDTCNTCL WDTSSEL WDTIS0 /* TACLK*2POWER13=250ms " */
#define WDT_ARST_16 WDTPW WDTCNTCL WDTSSEL WDTIS1 /* TACLK*2POWER9=16ms " */
#define WDT_ARST_1_9 WDTPW WDTCNTCL WDTSSEL WDTIS1 WDTIS0 /* TACLK*2POWER6=1.9ms " */
```

MSP430 寄存器中文注释---A/D 采样寄存器定义

```
/************************************************************
* ADC12 A/D 采样寄存器定义
************************************************************/
/*ADC12 转换控制类寄存器*/
#define ADC12CTL0_ 0x0;' /* ADC12 Control 0 */
sfrw ADC12CTL0 = ADC12CTL0_;
#define ADC12CTL1_ 0x01A2 /* ADC12 Control 1 */
sfrw ADC12CTL1 = ADC12CTL1_;
/*ADC12 中断控制类寄存器*/
#define ADC12IFG_ 0x01A4 /* ADC12 Interrupt Flag */
sfrw ADC12IFG = ADC12IFG_;
#define ADC12IE_ 0x01A6 /* ADC12 Interrupt Enable */
sfrw ADC12IE = ADC12IE_;
#define ADC12IV_ 0x01A8 /* ADC12 Interrupt Vector Word */
sfrw ADC12IV = ADC12IV_;
/*ADC12 存贮器类寄存器*/
#define ADC12MEM_ 0x0140 /* ADC12 Conversion Memory */
#ifndef __IAR_SYSTEMS_ICC
#define ADC12MEM ADC12MEM_ /* ADC12 Conversion Memory (for assembler) */
#else
#define ADC12MEM ((int*) ADC12MEM_) /* ADC12 Conversion Memory (for C) */
#endif
#define ADC12MEM0_ ADC12MEM_ /* ADC12 Conversion Memory 0 */
sfrw ADC12MEM0 = ADC12MEM0_;
#define ADC12MEM1_ 0x0142 /* ADC12 Conversion Memory 1 */
sfrw ADC12MEM1 = ADC12MEM1_;
#define ADC12MEM2_ 0x0144 /* ADC12 Conversion Memory 2 */
sfrw ADC12MEM2 = ADC12MEM2_;
#define ADC12MEM3_ 0x0146 /* ADC12 Conversion Memory 3 */
sfrw ADC12MEM3 = ADC12MEM3_;
#define ADC12MEM4_ 0x0148 /* ADC12 Conversion Memory 4 */
sfrw ADC12MEM4 = ADC12MEM4_;
#define ADC12MEM5_ 0x014A /* ADC12 Conversion Memory 5 */
sfrw ADC12MEM5 = ADC12MEM5_;
#define ADC12MEM6_ 0x014C /* ADC12 Conversion Memory 6 */
sfrw ADC12MEM6 = ADC12MEM6_;
```

```
#define ADC12MEM7_ 0x014E /* ADC12 Conversion Memory 7 */
sfrw ADC12MEM7 = ADC12MEM7_;
#define ADC12MEM8_ 0x0150 /* ADC12 Conversion Memory 8 */
sfrw ADC12MEM8 = ADC12MEM8_;
#define ADC12MEM9_ 0x0152 /* ADC12 Conversion Memory 9 */
sfrw ADC12MEM9 = ADC12MEM9_;
#define ADC12MEM10_ 0x0154 /* ADC12 Conversion Memory 10 */
sfrw ADC12MEM10 = ADC12MEM10_;
#define ADC12MEM11_ 0x0156 /* ADC12 Conversion Memory 11 */
sfrw ADC12MEM11 = ADC12MEM11_;
#define ADC12MEM12_ 0x0158 /* ADC12 Conversion Memory 12 */
sfrw ADC12MEM12 = ADC12MEM12_;
#define ADC12MEM13_ 0x015A /* ADC12 Conversion Memory 13 */
sfrw ADC12MEM13 = ADC12MEM13_;
#define ADC12MEM14_ 0x015C /* ADC12 Conversion Memory 14 */
sfrw ADC12MEM14 = ADC12MEM14_;
#define ADC12MEM15_ 0x015E /* ADC12 Conversion Memory 15 */
sfrw ADC12MEM15 = ADC12MEM15_;
  /*ADC12 存贮控制类寄存器*/
#define ADC12MCTL_ 0x0080 /* ADC12 Memory Control */
#ifndef __IAR_SYSTEMS_ICC
#define ADC12MCTL ADC12MCTL_ /* ADC12 Memory Control (for assembler) */
#else
#define ADC12MCTL ((char*) ADC12MCTL_) /* ADC12 Memory Control (for C) */
#endif
#define ADC12MCTL0_ ADC12MCTL_ /* ADC12 Memory Control 0 */
sfrb ADC12MCTL0 = ADC12MCTL0_;
#define ADC12MCTL1_ 0x0081 /* ADC12 Memory Control 1 */
sfrb ADC12MCTL1 = ADC12MCTL1_;
#define ADC12MCTL2_ 0x0082 /* ADC12 Memory Control 2 */
sfrb ADC12MCTL2 = ADC12MCTL2_;
#define ADC12MCTL3_ 0x0083 /* ADC12 Memory Control 3 */
sfrb ADC12MCTL3 = ADC12MCTL3_;
#define ADC12MCTL4_ 0x0084 /* ADC12 Memory Control 4 */
sfrb ADC12MCTL4 = ADC12MCTL4_;
#define ADC12MCTL5_ 0x0085 /* ADC12 Memory Control 5 */
sfrb ADC12MCTL5 = ADC12MCTL5_;
```

```
#define ADC12MCTL6_ 0x0086 /* ADC12 Memory Control 6 */
sfrb ADC12MCTL6 = ADC12MCTL6_;
#define ADC12MCTL7_ 0x0087 /* ADC12 Memory Control 7 */
sfrb ADC12MCTL7 = ADC12MCTL7_;
#define ADC12MCTL8_ 0x0088 /* ADC12 Memory Control 8 */
sfrb ADC12MCTL8 = ADC12MCTL8_;
#define ADC12MCTL9_ 0x0089 /* ADC12 Memory Control 9 */
sfrb ADC12MCTL9 = ADC12MCTL9_;
#define ADC12MCTL10_ 0x008A /* ADC12 Memory Control 10 */
sfrb ADC12MCTL10 = ADC12MCTL10_;
#define ADC12MCTL11_ 0x008B /* ADC12 Memory Control 11 */
sfrb ADC12MCTL11 = ADC12MCTL11_;
#define ADC12MCTL12_ 0x008C /* ADC12 Memory Control 12 */
sfrb ADC12MCTL12 = ADC12MCTL12_;
#define ADC12MCTL13_ 0x008D /* ADC12 Memory Control 13 */
sfrb ADC12MCTL13 = ADC12MCTL13_;
#define ADC12MCTL14_ 0x008E /* ADC12 Memory Control 14 */
sfrb ADC12MCTL14 = ADC12MCTL14_;
#define ADC12MCTL15_ 0x008F /* ADC12 Memory Control 15 */
sfrb ADC12MCTL15 = ADC12MCTL15_;
  /* ADC12CTL0 内 8 位控制寄存器位*/
#define ADC12SC 0x001 /*采样/转换控制位*/
#define ENC 0x002 /* 转换允许位*/
#define ADC12TOVIE 0x004 /*转换时间溢出中断允许位*/
#define ADC12OVIE 0x008 /*溢出中断允许位*/
#define ADC12ON 0x010 /*ADC12 内核控制位*/
#define REFON 0x020 /*参考电压控制位*/
#define REF2_5V 0x040 /*内部参考电压的电压值选择位 '0'为 1.5V; '1'为 2.5V*/
#define MSH 0x080 /*多次采样/转换位*/
#define MSC 0x080 /*多次采样/转换位*/
/*SHT0 采样保持定时器 0 控制 ADC12 的结果存贮器 MEM0~MEM7 的采样周期*/
#define SHT0_0 0*0x100 /*采样周期=TADC12CLK*4 */
#define SHT0_1 1*0x100 /*采样周期=TADC12CLK*8 */
#define SHT0_2 2*0x100 /*采样周期=TADC12CLK*16 */
#define SHT0_3 3*0x100 /*采样周期=TADC12CLK*32 */
#define SHT0_4 4*0x100 /*采样周期=TADC12CLK*64 */
#define SHT0_5 5*0x100 /*采样周期=TADC12CLK*96 */
```

```
#define  SHT0_6  6*0x100  /*采样周期=TADC12CLK*128  */
#define  SHT0_7  7*0x100  /*采样周期=TADC12CLK*192  */
#define  SHT0_8  8*0x100  /*采样周期=TADC12CLK*256  */
#define  SHT0_9  9*0x100  /*采样周期=TADC12CLK*384  */
#define  SHT0_10  10*0x100  /*采样周期=TADC12CLK*512  */
#define  SHT0_11  11*0x100  /*采样周期=TADC12CLK*768  */
#define  SHT0_12  12*0x100  /*采样周期=TADC12CLK*1024  */
#define  SHT0_13  13*0x100  /*采样周期=TADC12CLK*1024  */
#define  SHT0_14  14*0x100  /*采样周期=TADC12CLK*1024  */
#define  SHT0_15  15*0x100  /*采样周期=TADC12CLK*1024  */
/*SHT1  采样保持定时器 1  控制 ADC12 的结果存贮器 MEM8~MEM15 的采样周期*/
#define  SHT1_0  0*0x100  /*采样周期=TADC12CLK*4  */
#define  SHT1_1  1*0x100  /*采样周期=TADC12CLK*8  */
#define  SHT1_2  2*0x100  /*采样周期=TADC12CLK*16  */
#define  SHT1_3  3*0x100  /*采样周期=TADC12CLK*32  */
#define  SHT1_4  4*0x100  /*采样周期=TADC12CLK*64  */
#define  SHT1_5  5*0x100  /*采样周期=TADC12CLK*96  */
#define  SHT1_6  6*0x100  /*采样周期=TADC12CLK*128  */
#define  SHT1_7  7*0x100  /*采样周期=TADC12CLK*192  */
#define  SHT1_8  8*0x100  /*采样周期=TADC12CLK*256  */
#define  SHT1_9  9*0x100  /*采样周期=TADC12CLK*384  */
#define  SHT1_10  10*0x100  /*采样周期=TADC12CLK*512  */
#define  SHT1_11  11*0x100  /*采样周期=TADC12CLK*768  */
#define  SHT1_12  12*0x100  /*采样周期=TADC12CLK*1024  */
#define  SHT1_13  13*0x100  /*采样周期=TADC12CLK*1024  */
#define  SHT1_14  14*0x100  /*采样周期=TADC12CLK*1024  */
#define  SHT1_15  15*0x100  /*采样周期=TADC12CLK*1024  */

/*  ADC12CTL1  内 8 位控制寄存器位*/
#define  ADC12BUSY  0x0001  /*ADC12 忙标志位*/
#define  CONSEQ_0  0*2  /*单通道单次转换*/
#define  CONSEQ_1  1*2  /*序列通道单次转换*/
#define  CONSEQ_2  2*2  /*单通道多次转换*/
#define  CONSEQ_3  3*2  /*序列通道多次转换*/
#define  ADC12SSEL_0  0*8  /*ADC12 内部时钟源*/
#define  ADC12SSEL_1  1*8  /*ACLK*/
#define  ADC12SSEL_2  2*8  /*MCLK*/
```

```
#define  ADC12SSEL_3  3*8  /*SCLK*/
#define  ADC12DIV_0  0*0x20  /*1 分频*/
#define  ADC12DIV_1  1*0x20  /*2 分频*/
#define  ADC12DIV_2  2*0x20  /*3 分频*/
#define  ADC12DIV_3  3*0x20  /*4 分频*/
#define  ADC12DIV_4  4*0x20  /*5 分频*/
#define  ADC12DIV_5  5*0x20  /*6 分频*/
#define  ADC12DIV_6  6*0x20  /*7 分频*/
#define  ADC12DIV_7  7*0x20  /*8 分频*/
#define  ISSH  0x0100  /*采样输入信号反向与否控制位*/
#define  SHP  0x0200  /*采样信号(SAMPCON)选择控制位*/
#define  SHS_0  0*0x400  /*采样信号输入源选择控制位  ADC12SC*/
#define  SHS_1  1*0x400  /*采样信号输入源选择控制位  TIMER_A.OUT1*/
#define  SHS_2  2*0x400  /*采样信号输入源选择控制位  TIMER_B.OUT0*/
#define  SHS_3  3*0x400  /*采样信号输入源选择控制位  TIMER_B.OUT1*/
/*转换存贮器地址定义位*/
#define  CSTARTADD_0  0*0x1000  /*选择 MEM0 首地址*/
#define  CSTARTADD_1  1*0x1000  /*选择 MEM1 首地址*/
#define  CSTARTADD_2  2*0x1000  /*选择 MEM2 首地址*/
#define  CSTARTADD_3  3*0x1000  /*选择 MEM3 首地址*/
#define  CSTARTADD_4  4*0x1000  /*选择 MEM4 首地址*/
#define  CSTARTADD_5  5*0x1000  /*选择 MEM5 首地址*/
#define  CSTARTADD_6  6*0x1000  /*选择 MEM6 首地址*/
#define  CSTARTADD_7  7*0x1000  /*选择 MEM7 首地址*/
#define  CSTARTADD_8  8*0x1000  /*选择 MEM8 首地址*/
#define  CSTARTADD_9  9*0x1000  /*选择 MEM9 首地址*/
#define  CSTARTADD_10  10*0x1000  /*选择 MEM10 首地址*/
#define  CSTARTADD_11  11*0x1000  /*选择 MEM11 首地址*/
#define  CSTARTADD_12  12*0x1000  /*选择 MEM12 首地址*/
#define  CSTARTADD_13  13*0x1000  /*选择 MEM13 首地址*/
#define  CSTARTADD_14  14*0x1000  /*选择 MEM14 首地址*/
#define  CSTARTADD_15  15*0x1000  /*选择 MEM15 首地址*/

/*  ADC12MCTLx  */
#define  INCH_0  0  /*选择模拟量通道 0  A0  */
#define  INCH_1  1  /*选择模拟量通道 0  A1*/
#define  INCH_2  2  /*选择模拟量通道 0  A2*/
```

```
#define INCH_3 3 /*选择模拟量通道 0 A3*/
#define INCH_4 4 /*选择模拟量通道 0 A4*/
#define INCH_5 5 /*选择模拟量通道 0 A5*/
#define INCH_6 6 /*选择模拟量通道 0 A6*/
#define INCH_7 7 /*选择模拟量通道 0 A7*/
#define INCH_8 8 /*VEREF */
#define INCH_9 9 /*VEREF-*/
#define INCH_10 10 /*片内温度传感器的输出*/
#define INCH_11 11 /*(AVCC-AVSS)/2*/
#define INCH_12 12 /*(AVCC-AVSS)/2*/
#define INCH_13 13 /*(AVCC-AVSS)/2*/
#define INCH_14 14 /*(AVCC-AVSS)/2*/
#define INCH_15 15 /*(AVCC-AVSS)/2*/
/*参考电压源选择位*/
#define SREF_0 0*0x10 /*VR = AVCC; VR- = AVSS*/
#define SREF_1 1*0x10 /*VR = VREF ; VR- = AVSS*/
#define SREF_2 2*0x10 /*VR = VEREF ; VR- = AVSS*/
#define SREF_3 3*0x10 /*VR = VEREF ; VR- = AVSS*/
#define SREF_4 4*0x10 /*VR = AVCC; VR- = VREF-*/
#define SREF_5 5*0x10 /*VR = VREF ; VR- = VREF-*/
#define SREF_6 6*0x10 /*VR = VEREF ; VR- = VREF-*/
#define SREF_7 7*0x10 /*VR = VEREF ; VR- = VREF-*/
#define EOS 0x80 /*序列结束选择位*/
```

MSP430 寄存器中文注释----串口寄存器

```
/************************************************************
* USART 串口寄存器"UCTL","UTCTL","URCTL"定义的各个位可串口 1 串口 2 公用
************************************************************/
/* UCTL 串口控制寄存器*/
#define PENA 0x80 /*校验允许位
#define PEV 0x40 /*偶校验 为 0 时为奇校验*/
#define SPB 0x20 /*停止位为 2 为 0 时停止位为 1*/
#define CHAR 0x10 /*数据位为 8 位为 0 时数据位为 7 位*/
#define LISTEN 0x08 /*自环模式(发数据同时在把发的数据接收回来)*/
#define SYNC 0x04 /*同步模式 为 0 异步模式*/
#define MM 0x02 /*为 1 时地址位多机协议(异步) 主机模式(同步);为 0 时线路空闲多机协
议(异步) 从机模式(同步)*/
#define SWRST 0x01 /*控制位*/
```

```
   /* UTCTL 串口发送控制寄存器*/
#define CKPH 0x80 /*时钟相位控制位(只同步方式用)为 1 时时钟 UCLK 延时半个周期*/
#define CKPL 0x40 /*时钟极性控制位 为 1 时异步与 UCLK 相反;同步下降延有效*/
#define SSEL1 0x20 /*时钟源选择位:与 SSEL0 组合为 0,1,2,3 四种方式*/
#define SSEL0 0x10 /*"0"选择外部时钟,"1"选择辅助时钟,"2","3"选择系统子时钟 */
#define URXSE 0x08 /*接收触发延控制位(只在异步方式下用)*/
#define TXWAKE 0x04 /*多处理器通信传送控制位(只在异步方式下用)*/
#define STC 0x02 /*外部引脚 STE 选择位为 0 时为 4 线模式 为 1 时为 3 线模式*/
#define TXEPT 0x01 /*发送器空标志*/
   /* URCTL 串口接收控制寄存器 同步模式下只用两位:FE 和 OE*/
#define FE 0x80 /*帧错标志*/
#define PE 0x40 /*校验错标志位*/
#define OE 0x20 /*溢出标志位*/
#define BRK 0x10 /*打断检测位*/
#define URXEIE 0x08 /*接收出错中断允许位*/
#define URXWIE 0x04 /*接收唤醒中断允许位*/
#define RXWAKE 0x02 /*接收唤醒检测位*/
#define RXERR 0x01 /*接收错误标志位*/
   /************************************************************
* USART 0 串口 0 寄存器定义
************************************************************/
   #define U0CTL_ 0x0070 /* UART 0 Control */
sfrb U0CTL = U0CTL_;
#define U0TCTL_ 0x0071 /* UART 0 Transmit Control */
sfrb U0TCTL = U0TCTL_;
#define U0RCTL_ 0x0072 /* UART 0 Receive Control */
sfrb U0RCTL = U0RCTL_;
#define U0MCTL_ 0x0073 /* UART 0 Modulation Control */
sfrb U0MCTL = U0MCTL_;
#define U0BR0_ 0x0074 /* UART 0 Baud Rate 0 */
sfrb U0BR0 = U0BR0_;
#define U0BR1_ 0x0075 /* UART 0 Baud Rate 1 */
sfrb U0BR1 = U0BR1_;
#define U0RXBUF_ 0x0076 /* UART 0 Receive Buffer */
const sfrb U0RXBUF = U0RXBUF_;
#define U0TXBUF_ 0x0077 /* UART 0 Transmit Buffer */
sfrb U0TXBUF = U0TXBUF_;
```

```
  /* Alternate register names */
  #define UCTL0_ 0x0070 /* UART 0 Control */
sfrb UCTL0 = UCTL0_;
#define UTCTL0_ 0x0071 /* UART 0 Transmit Control */
sfrb UTCTL0 = UTCTL0_;
#define URCTL0_ 0x0072 /* UART 0 Receive Control */
sfrb URCTL0 = URCTL0_;
#define UMCTL0_ 0x0073 /* UART 0 Modulation Control */
sfrb UMCTL0 = UMCTL0_;
#define UBR00_ 0x0074 /* UART 0 Baud Rate 0 */
sfrb UBR00 = UBR00_;
#define UBR10_ 0x0075 /* UART 0 Baud Rate 1 */
sfrb UBR10 = UBR10_;
#define RXBUF0_ 0x0076 /* UART 0 Receive Buffer */
const sfrb RXBUF0 = RXBUF0_;
#define TXBUF0_ 0x0077 /* UART 0 Transmit Buffer */
sfrb TXBUF0 = TXBUF0_;
  #define UCTL_0_ 0x0070 /* UART 0 Control */
sfrb UCTL_0 = UCTL_0_;
#define UTCTL_0_ 0x0071 /* UART 0 Transmit Control */
sfrb UTCTL_0 = UTCTL_0_;
#define URCTL_0_ 0x0072 /* UART 0 Receive Control */
sfrb URCTL_0 = URCTL_0_;
#define UMCTL_0_ 0x0073 /* UART 0 Modulation Control */
sfrb UMCTL_0 = UMCTL_0_;
#define UBR0_0_ 0x0074 /* UART 0 Baud Rate 0 */
sfrb UBR0_0 = UBR0_0_;
#define UBR1_0_ 0x0075 /* UART 0 Baud Rate 1 */
sfrb UBR1_0 = UBR1_0_;
#define RXBUF_0_ 0x0076 /* UART 0 Receive Buffer */
const sfrb RXBUF_0 = RXBUF_0_;
#define TXBUF_0_ 0x0077 /* UART 0 Transmit Buffer */
sfrb TXBUF_0 = TXBUF_0_;
  /************************************************************
* USART 1 串口 1 寄存器定义
************************************************************/
```

```
  #define U1CTL_ 0x0078 /* UART 1 Control */
sfrb U1CTL = U1CTL_;
#define U1TCTL_ 0x0079 /* UART 1 Transmit Control */
sfrb U1TCTL = U1TCTL_;
#define U1RCTL_ 0x007A /* UART 1 Receive Control */
sfrb U1RCTL = U1RCTL_;
#define U1MCTL_ 0x007B /* UART 1 Modulation Control */
sfrb U1MCTL = U1MCTL_;
#define U1BR0_ 0x007C /* UART 1 Baud Rate 0 */
sfrb U1BR0 = U1BR0_;
#define U1BR1_ 0x007D /* UART 1 Baud Rate 1 */
sfrb U1BR1 = U1BR1_;
#define U1RXBUF_ 0x007E /* UART 1 Receive Buffer */
const sfrb U1RXBUF = U1RXBUF_;
#define U1TXBUF_ 0x007F /* UART 1 Transmit Buffer */
sfrb U1TXBUF = U1TXBUF_;
  #define UCTL1_ 0x0078 /* UART 1 Control */
sfrb UCTL1 = UCTL1_;
#define UTCTL1_ 0x0079 /* UART 1 Transmit Control */
sfrb UTCTL1 = UTCTL1_;
#define URCTL1_ 0x007A /* UART 1 Receive Control */
sfrb URCTL1 = URCTL1_;
#define UMCTL1_ 0x007B /* UART 1 Modulation Control */
sfrb UMCTL1 = UMCTL1_;
#define UBR01_ 0x007C /* UART 1 Baud Rate 0 */
sfrb UBR01 = UBR01_;
#define UBR11_ 0x007D /* UART 1 Baud Rate 1 */
sfrb UBR11 = UBR11_;
#define RXBUF1_ 0x007E /* UART 1 Receive Buffer */
const sfrb RXBUF1 = RXBUF1_;
#define TXBUF1_ 0x007F /* UART 1 Transmit Buffer */
sfrb TXBUF1 = TXBUF1_;
  #define UCTL_1_ 0x0078 /* UART 1 Control */
sfrb UCTL_1 = UCTL_1_;
#define UTCTL_1_ 0x0079 /* UART 1 Transmit Control */
sfrb UTCTL_1 = UTCTL_1_;
#define URCTL_1_ 0x007A /* UART 1 Receive Control */
```

```
sfrb URCTL_1 = URCTL_1_;
#define UMCTL_1_ 0x007B /* UART 1 Modulation Control */
sfrb UMCTL_1 = UMCTL_1_;
#define UBR0_1_ 0x007C /* UART 1 Baud Rate 0 */
sfrb UBR0_1 = UBR0_1_;
#define UBR1_1_ 0x007D /* UART 1 Baud Rate 1 */
sfrb UBR1_1 = UBR1_1_;
#define RXBUF_1_ 0x007E /* UART 1 Receive Buffer */
const sfrb RXBUF_1 = RXBUF_1_;
#define TXBUF_1_ 0x007F /* UART 1 Transmit Buffer */
sfrb TXBUF_1 = TXBUF_1_;
```

MSP430 教程 7：MSP430 单片机的端口介绍

MSP430 的端口有 P1、P2、P3、P4、P5、P6、S 和 COM（型号不同，包含的端口也不仅相同，如 MSP430X11X 系列只有 P1,P2 端口，而 MSP430X4XX 系列则包含全部上述端口），它们都可以直接用于输入/输出。MSP430 系统中没有专门的输入/输出指令，输入/输出操作通过传送指令来实现。端口 P1`P6 的每一位都可以独立用于输入/输出，即具有位寻址功能。常见的键盘接口可以直接用端口进行模拟，用查询或者中断方式控制。由于 MSP430 的端口只有数据口，没有状态口或控制口，在实际应用中，如在查询式输入/输出传送时，可以用端口的某一位或者几位来传送状态信息，通过查询对应位的状态来确定外设是否处于“准备好”状态。

端口的功能。（1）P1,P2 端口： I/O,中断功能,其他片内外设功能如定时器、比较器；（2）P3,P4P5P6 端口：I/O,其他片内外设功能如 SPI、UART 模式，A/D 转换等；（3）S,COM 端口：I/O,驱动液晶。

MSP430 各端口具有丰富的控制寄存器供用户实现相应的操作。其中 P1,P2 具有 7 个寄存器，P3~P6 具有 4 个寄存器。通过设置寄存器我们可以实现：（1）每个 I/O 位独立编程；（2）任意组合输入，输出和中断；（3）P1,P2 所有 8 个位全部可以用作外部中断处理；（4）可以使用所以指令对寄存器操作；（5）可以按字节输入、输出，也可按位进行操作。

端口 P1,P2 的功能可以通过它们的 7 个控制寄存器来实现。这里，Px 代表 P1 或 P2。

（1）PxDIR：输入/输出方向寄存器。 8 位相互独立，可以分别定义 8 个引脚的输入/输出方向。8 位再 PUC 后都被复位。使用输入/输出功能时，应该先定义端口的方向 。作为输入时只能读，作为输出时，可读可写。0：输入模式；1：输出模式。如：P1DIR|=BIT4; //P1.4 输出 ， P2DIR=0XF0; //高 4 位输出，低 4 位输入。

（2）PXIN：输入寄存器，为只读寄存器。用户不能对它进行写入，只能通过读取其寄存器的内容来知道 I/O 口的输入信号。所以其引脚的方向要选为输入。如再键盘键盘扫描程序中经常要读取行线或者列线的端口寄存器值来判断案件情况。例如：unsigned char key;

P1DIR&=~BIT4; //P1.4 输入

……

key=P1IN&0X10; //输出端口 P1.4 的值

……

（3）PXOUT：输出寄存器。该寄存器为 I/O 端口的输出缓冲寄存器，再读取时输出缓存的内容与引脚方向定义无关。改变方向寄存器的内容，输出缓存的内容不受影响。如：PIOUT|=0X01; //P1.0 输出 1 ， PIOUT&=~0X01; //P1.0 输出 0 。

（4）PXIFG：中断标志寄存器。他的 8 个标志位标志相应引脚是否有中断请求有待处理。0：无中断请求， 1：有中断请求。其中断标志分别为 PXIFG.0~PXIFG.7。应该注意的是：PXIFG.0~PXIFG.7 共用一个中断向量，为多源中断。当任一事件引起的中断进行处理时，PXIFG.0~PXIFG.7 不会自动复位，必须由软件来判断是对哪一个事件，并将相应的标志复

位。另外，外部中断事件的时间必须保持不低于 1.5 倍的 MCLK 时间，以保证中断请求被接受，且使相应中断标志位置位。

（5）PXIES：中断触发沿选择寄存器。如果允许 PX 口的某个引脚中断，还需定义该引脚的中断触发方式。0：上升沿触发使相应标志置位，1：下降沿触发相应标志置位。如：MOV.B #07H, &P1IES ;p1 低 3 位下降沿触发中断。

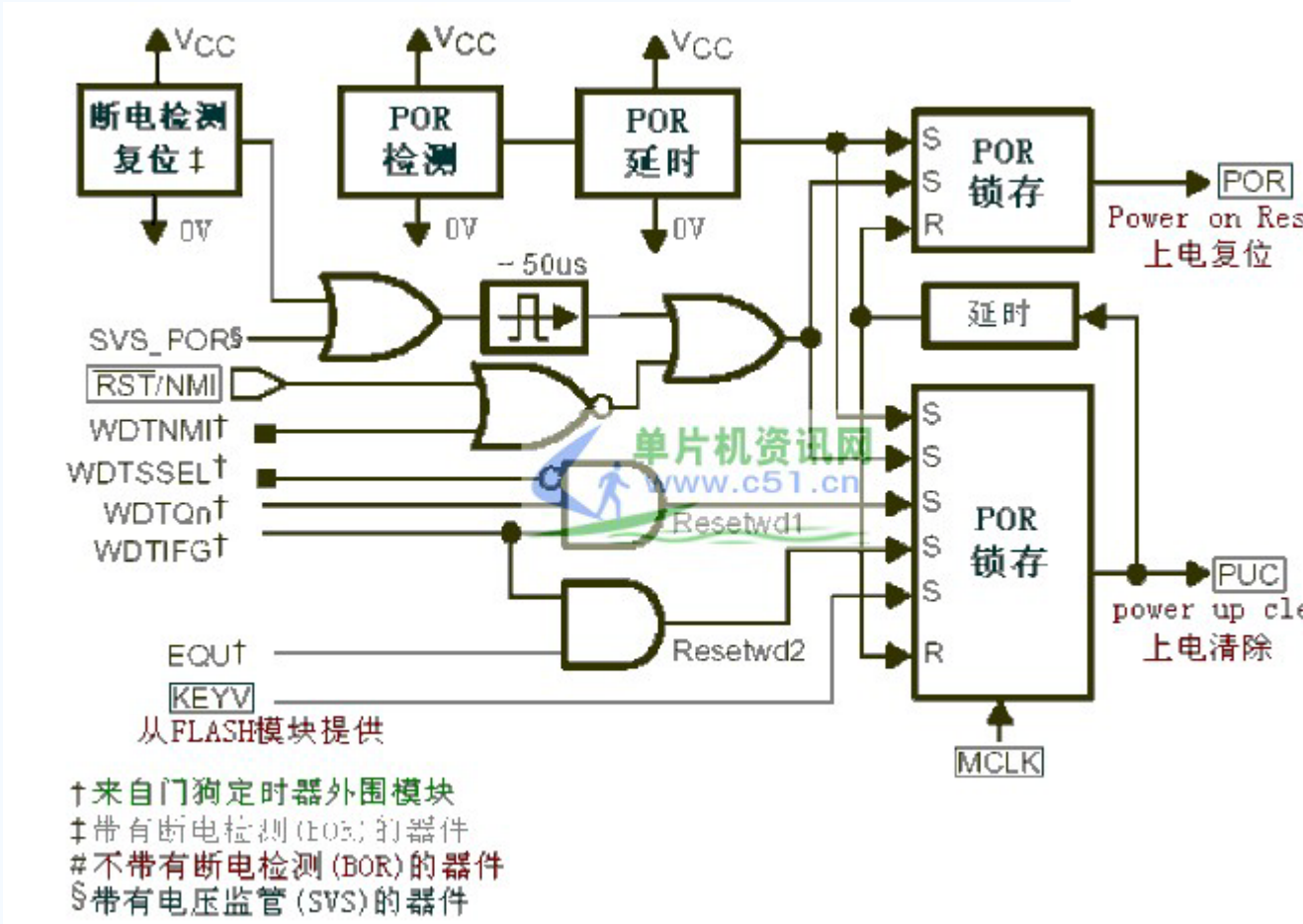
（6）PXIE：中断使能寄存器。PX 口的每一个引脚都有一位用以控制该引脚是否允许中断。0：禁止中断 ，1：允许中断。MOV.B #0E0H, &P2IE ;P2 高 3 位允许中断。

（7）PXSEL：功能选择寄存器。P1,P2 两端口还具有其他片内外设功能，将这些功能与芯片外的联系通过复用 P1,P2 引脚的方式来实现。PXSEL 用来选择引脚的 I/O 端口功能与外围模块功能。0：选择引脚为 I/O 端口，1：选择引脚为外围模块功能。如：P1SEL|=0X10; //P1.4 为外围模块功能。

端口 P3、P4、P5、P6 没有中断能力，其余功能同 PI,P2。除掉端口 P1,P2 与中断相关的 3 个寄存器，端口 P3,P4,P5,P6 的 4 个寄存器（用法同 P1，P2）分别为 PXDIR，PXIN，PXOUT，PXSEL 可供用户使用。

端口 COM 和 S，他们实现与液晶片的直接接口。COM 为液晶片的公共端，S 为液晶片的段码端。液晶片输出端也可经软件配置为数字输出端口。

MSP430 教程 8：MSP430 单片机复位电路



MSP430 单片机系统复位电路

从上 MSP430 系统复位电路功能模块图中可以看到了两个复位信号，一个是上电复位信号 POR(Power On Reset)和上电清除信号 PUC(Power Up Clear)。

POR 信号是器件的复位信号，此信号只有在以下的事件发生时才会产生：
器件上电时。

RST/NMI 引脚配置为复位模式，当 RST/NMI 引脚生产低电平时。

当 POR 信号产生时，必然会产生 PUC 信号；而 PUC 信号的产生时不会产生 POR 信号。会引起产生 PUC 信号的事件：
POR 信号发生时。

启动看门狗时，看门狗定时器计满时。

向看门狗写入错误的安全参数值时。

向片内 FLASH 写入错误的安全参数值时。

MSP430 单片机系统复位后器件的初始

当 POR 信号或 PUC 信号发生时引起器件复位后，器件的初始化状态为：

RST/NMI 引脚配置为复位模式。

I/O 引脚为输入模式。

装态寄存器复位。

程序计数器(PC)装入复位向量地址 0xFFFE,CPU 从此地址开发始执行。

其它模块的寄存器初始化，详情请查器件手册。

MSP430 教程 9：MSP430 单片机时钟模块

MSP430 的时钟模块由低速晶体振荡器 LFXT1、高速晶体振荡器 XT2（MSP430X11X,MSP430X12X 没有）、数字控制振荡器 DCO、琐相环 FLL（MSP430X16X 以上包括）和增强型琐相环 FLL 等部件组成。

MSP430X1XX 基本时钟模块有三个时钟输入源 LFXT1CLK(低速 32768Hz,高速 450Hz 到 8MHz)、XT2CLK(450Hz 到 8MHz)、DCOCLK，提供以下三种时钟信号

1.ACLK 辅助时钟：由 LFXT1CLK 信号经 1、2、4、8 分频后得到，可以由软件选作各个外围模块的时钟信号，一般用于低速外设。

2.MCLK 系统主时钟：MCLK 可由软件选择来自 LFXT1CLK、XT2CLK、DCOCLK 三者之一，然后经 1、2、4、8 分频得到，MCLK 主要用于 CPU 和系统。

3.SMCLK 子系统时钟：可由软件选自 LFXT1CLK 和 DCOCLK（MSP430X11X、MSP430X12X 系列，因其不含 XT2），或 XT2CLK 和 DCOCLK，然后经 1、2、4、8 分频得到。SMCLK 主要用于高速外围模块。

系统频率与系统的工作电压密切相关（MSP430 工作电压 1.8V~3.6V,编程电压 2.7V~3.6V）,所以不同的工作电压，需要选择不同的系统时钟。当两个外部振荡器失效时，DCO 振荡器会自动被选作 MCLK 的时钟源。PUC 信号之后，DCOCLK 被自动选作 MCLK 和 SMCLK 的时钟信号，LFXT1CLK 被选作 ACLK 的时钟信号，根据需要 MCLK 和 SMCLK 的时钟源可以另外设置。

控制时钟模块的三个寄存器为 DCO 控制寄存器 DCOCTL、基本时钟系统控制寄存器 1BCSCTL1、基本时钟控制寄存器 2BCSCTL2

1.DCOCTL

7 6 5 4 3 2 1 0

DCO2 DCO1 DCO0 MOD4 MOD3 MOD2 MOD1 MOD0

DCO.0~DCO.2 定义 8 种频率之一（DCO=0~DCO=8），可分段调节 DCOCLK 频率，相领两种频率相差 10%。

MOD.0~MOD.4 定义在 32 个 DCO 周期中插入的 fdco 1 周期个数，而余下的周期为 fdco 周期，控制切换 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。

DCOCTL POR 后初始值为 60H。

2.BCSCTL1

7 6 5 4 3 2 1 0

XT2OFF XTS DIVA1 DIVA0 XT5V RSEL2 RSEL1 RSEL0

BCSCTL1 初始值为 84H

XT2OFF 控制 XT2 的开启和关闭

0 XT2 振荡器开启

1 XT2 振荡器关闭（默认）

XTS 控制 LFXT1 工作模式

0 低频模式（默认）

1 高频模式

DIVA1、DIVA0 控制 ACLK 分频

0 0 不分频（默认）

0 1 2 分频

1 0 4 分频

1 1 8 分频

XT5V 此位设置为 0

RSEL0~RSEL2 三位控制内部电阻以决定标称频率

0 选择最低标称频率

... .

7 选择最高标称频率

3.BCSCLK2

7 6 5 4 3 2 1 0

SELM1 SELM0 DIVM1 DIVM0 SELS DIVS1 DIVS0 DCOR

SELM1 SELM0 选择 MCLK 时钟源

0 0 DCOCLK（默认）

0 1 DCOCLK

1 0 LFXT1CLK 对于 MSP430F11/12X，XT2CLK 对于 MSP430F13/14/15/16X

1 1 LFXT1CLK

DIVM1 DIVM0 选择 MCLK 分频

0 0 不分频

0 1 2 分频

1 0 4 分频

1 1 8 分频

SELS 选择 SMCLK 时钟源

0 DCLK（默认）

1 LFXT1CLK 对于 MSP430F11/12X，XT2CLK 对于 MSP430F13/14/15/16X

DIVS1 DIVS0 选择 SMCLK 分频

0 0 不分频

0 1 2 分频

1 0 4 分频

1 1 8 分频

DCOR 选择 DCO 电阻

0 内部电阻

1 外部电阻

时钟模块的应用

一、设置 MCLK=XT2，SMCLK=DCOCLK，将 MCLK 由 P5.4 输出(MSP430X14X 中引脚 P5.4 和 MCLK 复用)

```
#include "msp430x14x.h"
void main(void)
{
unsigned int i;
WDTCTL= WDTPW WDTHOLD; //Stop watchdog to prevent to overflow
P5DIR | = 0X10; //Set P5.4 to output
P5SEL | = 0X10; //Set P5.4 to MCLK mode
BCSCTL1 &= ~XT2OFF; //Enable XT2

do{
IFG1 & = ~OFIFG; //Clear OFIFG
for(i=0xff;i>0;i--); //Set a delay
}while(IFG1&OFIFG); //check-up the OFIFG

BCSCTL2 | = SELM_2; //Set XT2CLK the clock of MCLK

for(;;);
}
```

二、 设置 ACLK=MCLK=LFXT1=HF，将 ACLK 用 P2.0（复用）输出。

```
#include "msp430x14x.h"
void main(void)
{
unsigned int i,j;
WDTCTL = WDTPW WDTHOLD;
P1DIR =0x02;
P2DIR =0x01;
P2SEL=0x01;
BCSCTL1 | = XTS;

do{
IFG1 & = ~OFIFG;
for(i=0xff;i>0;i--);
```

```
}while(IFG1 & OFIFG);

BCSCTL2 | = SELM_3;

for(;;);
}
```

MSP430 教程 10：MSP430 单片机 WDT 看门狗定时器

看门狗定时器用来防止程序因供电电源、空间电磁干扰或其它原因引起的强烈干扰噪声而跑飞的事故。程序中设置看门狗清零指令 WDTCTL=WDTPW WDTCNTCL，当程序跑飞不能及时清零看门狗，导致看门狗溢出复位，这样程序可以恢复正常运行状态。

一、WDT 寄存器包括 WDTCNT 和 WDTCTL，两个寄存器在上电和系统复位内容全部清零

1.记数单元 WDTCNT：WDTCNT 是 16 位增记数器，由 MSP430 选定的时钟电路产生的固定周期脉冲信号对记数器进行加法记数。WDTCNT 不能直接软件存取，必须通过看门狗定时器的控制寄存器 WDTCTL 来控制。

2.控制寄存器 WDTCTL：WDTCTL 由两部分组成，高 8 位用作口令，即 5AH(头文件中定义为 WDTPW)，低 8 位是对 WDT 操作的控制命令。写入 WDT 控制命令时先写入口令 WDTPW，口令写错将导致系统复位。读 WDTCTL 时不需口令，低字节 WDTCTL 的值，高字节读出始终为 69H。

bit 15-8 7 6 5 4 3 2 1 0

口令 HOLD NMIES NMI TMSEL CNTCL SSEL IS1 IS0

IS1 SI0 选择看门狗定时器的定时输出，T 为 WDTCNT 的输入时钟源周期。TMSEL WDT 工作模式选择

0 0 T*2 的 15 次方 0 看门狗模式

0 1 T*2 的 13 次方 1 定时器模式

1 0 T*2 的 9 次方 NMI 选择 RST/NMI 引脚功能

1 1 T*2 的 6 次方 0 RST/NMI 为复位端

SSEL 选择 WDTCNT 的时钟源 1 RST/NMI 为非屏蔽中断输入

0 SMCLK

1 ACLK

NMIES 选择 NMI 中断的边沿触发方式 HOLD 停止看门狗定时器工作

0 上升沿触发 NMI 中断 0 看门狗功能激活

1 下降沿触发 NMI 中断 1 时钟禁止输入，记数停止

二、WDT 的操作

1.用户通过设置 WDTCTL 中的 TMSEL 和 HOLD 控制位使 WDT 工作在看门狗模式、定时器模式和低功耗模式三种模式。

a.看门狗模式 (TMSEL=0 ,HOLD=0) 如果记数时间到，就会产生复位和激活系统上电清除信号，系统从上电复位的地址重启动 **中断向量为 RESET_VECTOR**

b.定时器模式(TMSEL=1，HOLD=0) 这一模式产生选定时间的周期性中断 中断标志位为 WDTIFG **中断向量为 WDT_VECTOR**

c.低功耗模式(TMSEL=X，HOLD=1) WDTCTL=WDTPW WDTHOLD;

2.WDT 通过 SSEL 和 IS0 IS1 3 位可以确定与 8 种时钟源相关的时间（ACLK=32768Hz，SMCLK=1MHz）

WDT_MDLY_32 WDT_MRST_32

WDT_MDLY_8 WDT_MRST_8

WDT_MDLY_0_5 WDT_MRST_0_5

WDT_MDLY_0_064 WDT_MRST_0_064

WDT_ADLY_1000 WDT_ARST_1000

WDT_ADLY_250 WDT_ARST_250

WDT_ADLY_16 WDT_ARST_16

WDT_ADLY_1_9 WDT_ARST_1_9

三、看门狗应用

使用看门狗定时器产生一个方波（周期性取反 P1.0）

```
#include "msp430x201x.h"
void main(void)
{
WDT=WDT_MDLY_32;
IE1 |=WDTIE;
P1DIR |=0x01;
_EINT();
for(;;)
{
_BIS_SR(CPUOFF);
_NOP();
}
```

```
}

interrupt[WDT_VECTOR] void watchdog_timer(void)
{
P1OUT^=0x01;
}
```

MSP430 教程 11：MSP430 单片机低功耗结构

当系统时钟发生器基本功能建立之后，CPU 内状态寄存器 SR 的 SCG1，SCG0，CPUOFF，OSCOFF 位是重要的低功耗控制位。只要任意中断被响应，上述控制位就被压入堆栈保存，中断处理之后，又可恢复先前的工作方式。在中断处理子程序执行期间，通过间接访问堆栈数据，可以操作这些控制位；这样允许程序在中断返回(RETI) 后，以另一种功耗方式继续运行。

各控制位的作用如下：

SCG1： 复位，使能 SMCLK；置位，禁止 SMCLK。

SCG0： 复位，激活直流发生器，只有 SCG0 置位，并且 DCOCLK 没有被用作 MCLK 或 SMCLK 时，直流发生器才能被禁止。

OSCOFF：复位，激活 LFXT1，只有当 OSCOFF 被置位并且 LFXT1CLK 不用于 MCLK 或 SMCLK 时，FLXT1 才能被禁止；当使用晶体振荡器关闭选项 OSCOFF 时，需要考虑晶体振荡器的启动设置时间

CPUOFF：复位，激活 MCLK；置位，关闭 MCLK。

控制位 SCG1、SCG0、CPUOFF、OSCOFF 可由软件配制成六种不同的工作模式：

工作模式 控制位 CPU 状态、振荡器及时钟

SCG1=0 CPU 活动

SCG0=0 MCLK 活动

AM CPUOFF=0 SMCLK 活动

OSCOFF=0 ACLK 活动

SCG1=0 CPU 禁止

LPM0 SCG0=0 MCLK 禁止

OSCOFF=0 SMCLK 活动

CPUOFF=1 ACLK 活动

SCG1=0 CPU 禁止

MCLK 禁止

LPM1 SCG0=1 如果 DCOCLK 位用作 MCLK 或 SMCLK，则直流发生器禁止，

否则，仍然活动

OSCOFF=0 SMCLK 活动

CPUOFF=1 ACLK 活动

SCG1=1 CPU 禁止

如果 DCO 未被用作 MCLK 或 SMCLK，自动禁止

SCG0=0 MCLK 禁止

LPM2 OSCOFF=0 SMCLK 禁止

CPUOFF=1 ACLK 活动

SCG1=1 CPU 禁止
DCO 被禁止，直流发生器被禁止
SCG0=1 MCLK 禁止
LPM3 OSCOFF=0 SMCLK 禁止
CPUOFF=1 ACLK 活动
SCG1=1 CPU 禁止
SCG0=1 DCO 被禁止，直流发生器被禁止
LPM4 OSCOFF=1 所有振荡器停止工作
MCLK、SMCLK 禁止
CPUOFF=1 ACLK 禁止
低功耗的设计技巧问题

1.LPM4:在振荡器关闭模式期间，处理机的所有部件工作停止，此时的电流消耗最小。此时只有在系统上电电路检测到低点电平或任一请求异步响应中断的外部中断事件时才会从新工作。因此在设计应含有可能需要用到的外部中断才采用这种模式，否则发生不可预料的结果。

2.LPM3:在 DC 发生关闭期间，只有晶振是活动的。但此时设置基本时序条件的 DC 发生器的 DC 电流被关闭。由于此电路的高阻设计，使功耗被抑制。当从 DC 关闭到启动 DC 需要一段时间（ns~us）

3.LPM2:在此期间晶振和 DC 发生器是工作的，所以可以实现快速启动

4.LPM1：在此期间振荡器已经工作，所以不存在启动延时问题

_BIS_SR(LPM3_bits) _BIC_SR_IRQ(LPM3_bits)
LPM3 LPM3_EXIT

系统响应中断过程：

1.硬件自动中断服务
a.PC 入栈
b.SR 入栈
c.中断向量赋给 PC
d.GIE、SCG1、CPOOFF 和 OSCOFF 清楚

e.IFG 标志位清除（单源中断标志比如 **WDTIFG**）

2.执行中断处理子程序

3.执行 RETI 指令

4.SR 出栈

5.PC 出栈

低功耗应用

```
void main(void)
{
WDTCTL=WDT_ADLY_1000;
IE1 | = WDTIE;
P1DIR | = 0X01;
_EINT();
for(;;)
{
LPM3;
_NOP();
}
}

interrupt[WDT_VECTOR] watchdog_timer(void)
{
P1OUT ^ = 0X01;
}
```

MSP430 教程 12：MSP430 单片机 MSP430 定时器

在 MSP430 系列单片机中带有功能强大的定时器资源，这定时器在单片机应用系统中起到重要的作用。利用 MSP430（以下称为 430）单片机的定时器可以用来实现计时，延时，信号频率测量，信号触发检测，脉冲脉宽信号测量，PWM 信号发生。另外通过软件编写可以用作串口的波特率发生器。后面我们将用定时器 A 作为一个波特率发生器，来编写一个串口例程给初学者参考。以加强初学者对定时器 A 的理解和应用。

在 430 的大系列产品中，不同的子系列产品定时器资源有所不同；在 F11X，F11X1 中是不带定时器 B 资源的。430 的定时器主要分为 3 部分模块：看门狗定时器，定时器 A，定时器 B。定时器 A 主要资源特点有 16 位定时计数器，其计数模式有 4 种。多种计数时钟信号供选择。3 个可配置输入的捕获/比较功能寄存器和 8 种输出模式的 3 个可配置输出单片。以上各块定时器资源可作多种组合使用，以实现强大的功能。

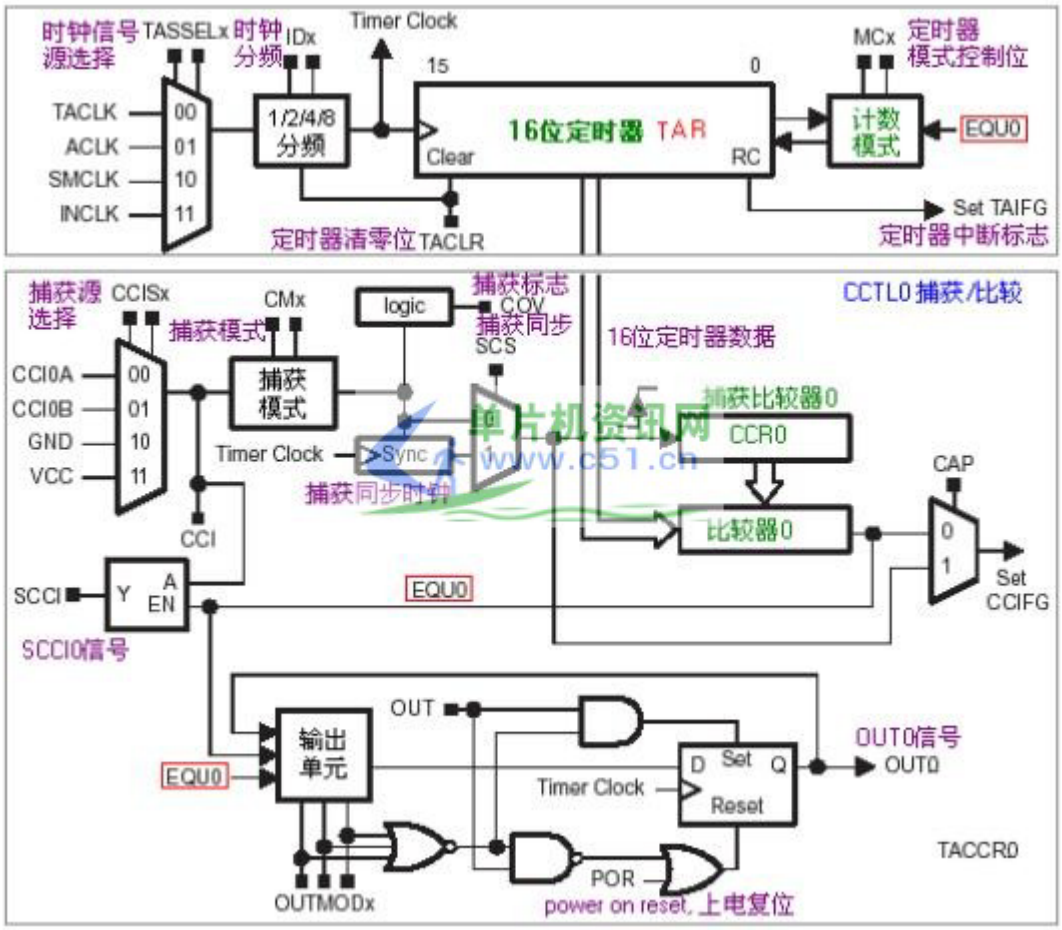
定时器资源功能说明

(1)看门狗定时器(WDT)：主要用于程序在生错误时用作单片机系统复位重起的。另外，也可作为一个基本定时器使用。

(2)定时器 A：作基本定时器使用，结合捕获/比较功能模块可实现时序控制，可编程波形信号发生输出。可作串口波特率发生器使用。

(3)定时器 B：作基本定时器使用，与定时器 A 基本相同,但是功能方面有某些功能会比 A 增强些。详情请看关于定时器 B 应用范例。

3-定时器 A 模块结构



4-定时器 A--基础应用例程(1)

//例程描述：利用定时器定时功能，实现 P1.0 方波输出。

#include <msp430x14x.h>

{

WDTCTL = WDTPW WDTHOLD; //停止看门狗 WDT，不使用内部看门狗定时器。

P1DIR |= 0x01; //设置 P1.0 口方向为输出。

CCTL0 = CCIE; //设置捕获/比较控制寄存器中 CCIE 位为 1，CCR0 捕获/比较功能中断为允许。

CCR0 = 50000; //捕获/比较控制寄存器 CCR0 初值为 5000。

TACTL = TASSEL_2 MC_2; //设置定时器 A 控制寄存器 TACTL，使时钟源选择为 SMCLK 辅助时钟。

```
_BIS_SR(LPM0_bits GIE); //进入低功耗模式 LPM0 和开中断
}

   //定时器 A  中断服务程序区

#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
P1OUT ^= 0x01; //P1.0 取反输出
CCR0 = 50000; //重新载入 CCR0 捕获/比较数据寄存器数据
}

//例程 1 结束----------------------------------------------------------------------------
```

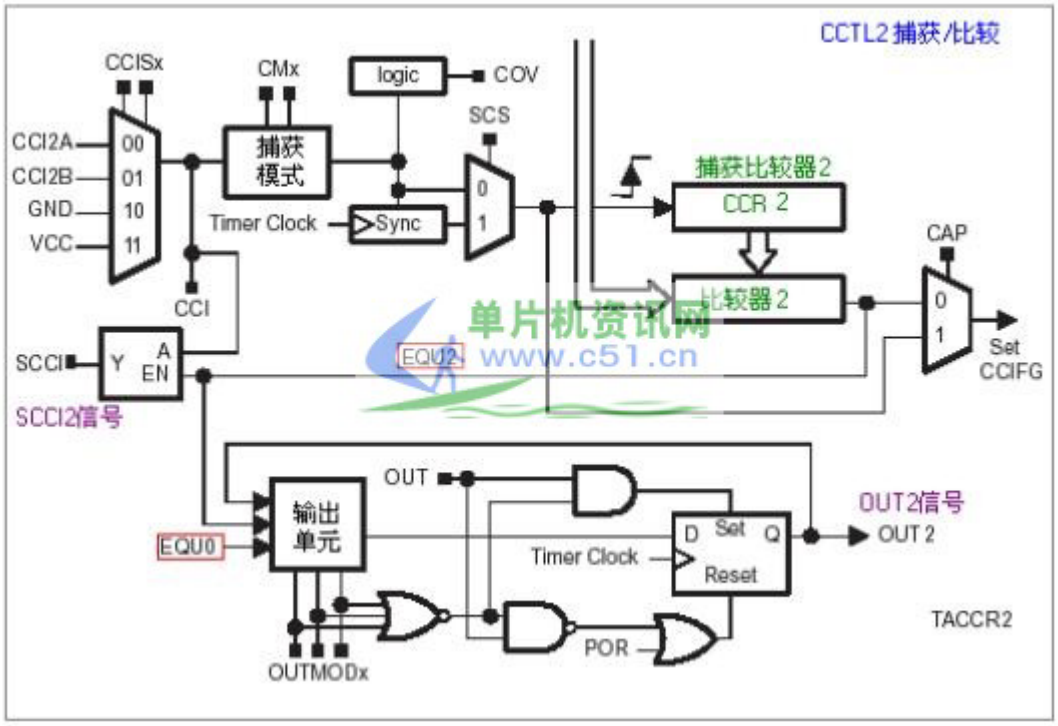
基础应用例程(2)

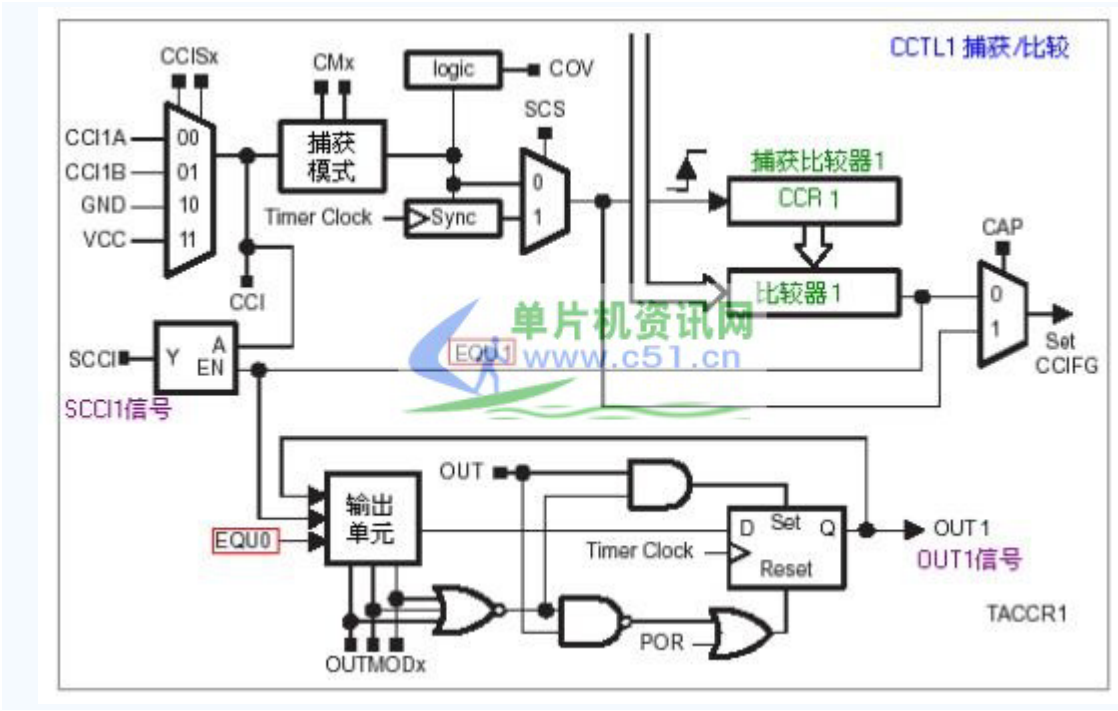
```
//例程描述：利用定时器定时功能，实现 P1.0 方波输出。
//  需要注意的是定时器中断程序,采用向量查询方式。

#include <msp430x14x.h>

   void main(void)
{
WDTCTL = WDTPW WDTHOLD; //  停止看门狗 WDT
P1DIR |= 0x01; //  设置 P1.0 口方向为输出。
TACTL = TASSEL_2 MC_2 TAIE; //  时钟源选择为 SMCLK,选择计数模式,定时器中断开

   _BIS_SR(LPM0_bits GIE); //进入低功耗模式 LPM0 和开中断
}

// Timer_A3  中断向量(TAIV)处理
#pragma vector=TIMERA1_VECTOR

__interrupt void Timer_A(void)
{
switch( TAIV )
{
case 2: break; //CCR1 不使用
```

```
case 4: break; //CCR2 不使用
case 10: P1OUT ^= 0x01; //溢出
break;
}
}
```

Timer_A 的寄存器

寄存器	缩写	读定类型	地址	初态
Timer_A 控制寄存器	TACTL	R/W	160H	POR 复位
Timer_A 计数器	TAR	R/W	170H	POR 复位
捕捞/比较控制寄存器 0	CCTL0	R/W	162H	POR 复位
捕捞/比较寄存器 0	CCR0	R/W	172H	POR 复位
捕捞/比较控制寄存器 1	CCTL1	R/W	164H	POR 复位
捕捞/比较寄存器 1	CCR1	R/W	174H	POR 复位
捕捞/比较控制寄存器 2	CCTL2	R/W	166H	POR 复位
捕捞/比较寄存器 2	CCR2	R/W	176H	POR 复位
中断向量寄存器	TAIV	R/W	12EH	POR 复位

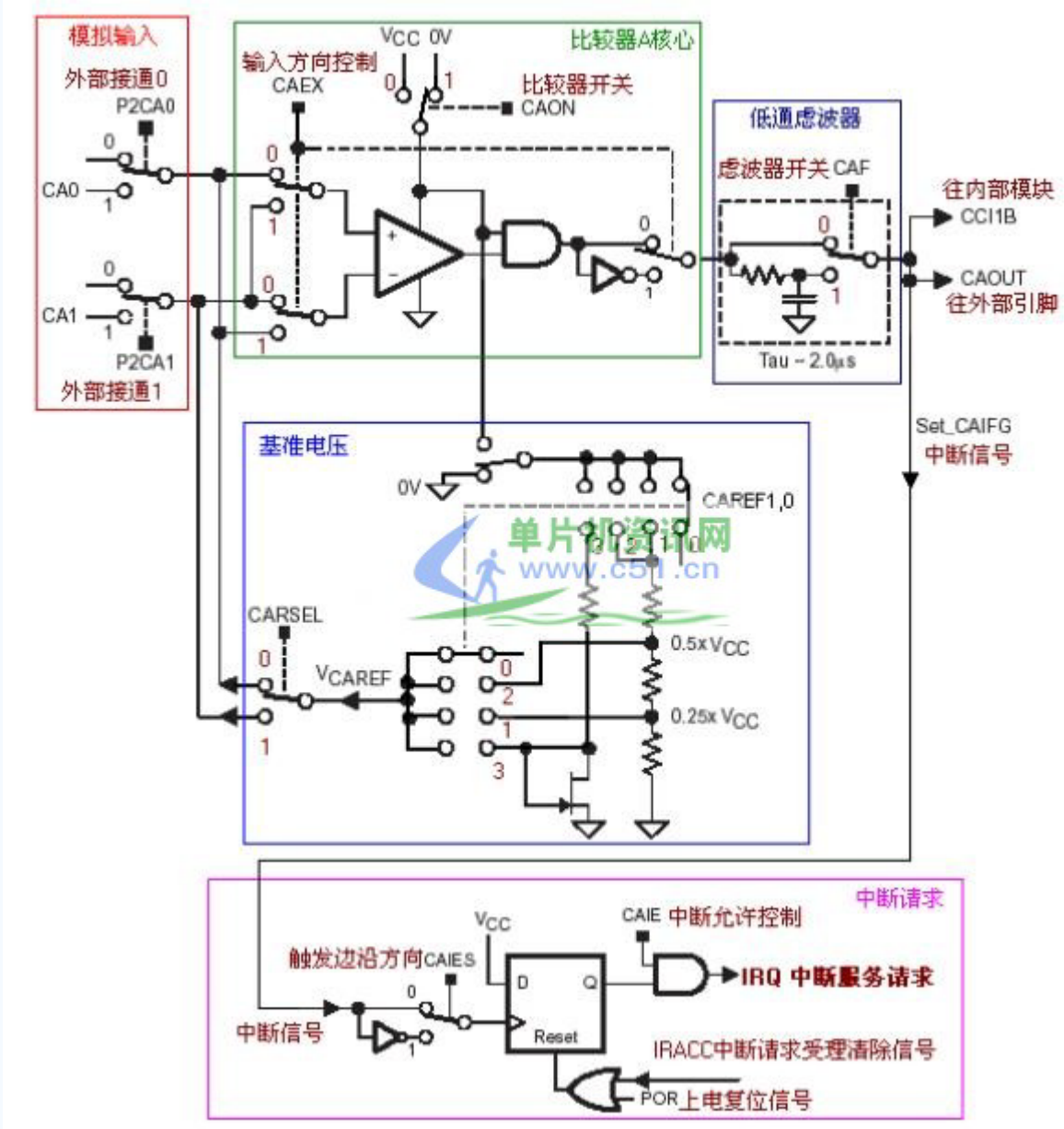




MSP430 教程 13：MSP430 单片机比较器 A 模块

比较器 A 模块

以下图可以看出比较器 A 的结构大概可以分 4 部分构成，分别为模拟输入，比较器 A 核心，低通虑波器，基准电压部分和中断部分组成。



首先，整个比较器 A 的工作必需由 CAON 位置为 1 时才能工作的，此位属 CACTL1 控制寄存器。单片机上电时此位是为 0 的，也就是说比较器是不工作的。

以下大概讲述几个部分电路的功能和一些相关信息。

模拟输入电路：

外部模拟引脚信号 CA0，CA1(正负端)可以分别由 P2CA0,P2CA1 位控制开或关。经过软件

的设置可以分别与内部的几个基准电压进行比较（0.5VCC,0.25VCC,三极管门值电压）或外部其中的电压进行比较。

应用的硬件比较可以分为以下三种组合：
两个外部引脚输入信号进行比较
其中一个外部引脚信号与内部的 0.5VCC 或 0.25VCC 比较
其中一个外部引脚信号与内部基准电压比较

参考电压发生器

参考电压电路是可以由 CARSEL，CARERF0，CARERF1 位来控制电压的产生。通过软件设置可以选择几种电压输出到比较器的输入中作为比较，当然此参考电压也可以通过单片机的引脚往外部提供参考电压之用。

比较器 A 核心

比较器 CAON 位控制开关，CAEX 位控制位控制方向。

低通虑波器

低通虑波器只需一个 CAF 位来控制此虑波器的功能开与关。此虑波器功能是用于消除比较器输出信号的毛刺，以保证信号的质量和中断请求的可靠性。

中断请求

比较器 A 模块是具有中断功能的，如比较器功能 CAIE 中断允许开了，在 CAIF 信号产生时将生产中断（当然 GIE 要为 1 时）。比较器 A 模块是具有中断独立向量的，是一个单独的中断，CUP 接受请求后会硬件自动清除中断标志位 CAIFG。

比较器模块相关寄存器说明

CACTL1 比较器控制寄存器 1

CACTL1 比较器 A 控制寄存器 1

7	6	5	4	3	2	1	0
CAEX	CARESL	CAREF1	CAREF0	CAON	CAIES	CAIE	CAIFG

CAEX: 控制内部比较器 A 的输入信号和输出信号的方向

CARSEL:控制内部参考电压加到比较器 A 的正输入端还是负输入端

由结构图可以看出,CAEX,CARSEL 在不同设置时,比较器 A 输入端的所加的参考电压是有不同的.

请参考以下列表:

CARSEL CAEX 参考电压接入端

0 0 内部参考源加到比较器的正端

0 1 内部参考源加到比较器的负端

1 0 内部参考源加到比较器的负端

1 1 内部参考源加到比较器的正端

CAREF1, CAREF0 选择参考源:

0 使用外部参考

1 选择 0.25VCC 为参考电压

2 选择 0.5VCC 为参考电压

3 选择二极管电压为参考电压,须参见具体 IC 的资料

CAON: 控制比较器 A 的打开与关闭

0 关闭比较器工作

1 打开比较器工作

CAIES: 中断边沿触发模式选择

0 上升沿使中断标志 CAIFG 置位

1 下降沿使中断标志 CAIFG 置位

CAIE:比较器中断允许

0 禁止中断

1 允许中断

CAIFG: 比较器中断标志

0 没有中断请求

1 有中断请求标志信号

CACTL2 比较器控制寄存器 2

CACTL2 比较器 A 控制寄存器 2

7	6	5	4	2	1	0
CACTL2.7	CACTL2.6	CACTL2.5	CACTL2.4 P2CA1	P2CA0	CAF	CAOUT

P2CA1： 控制输入端 CA1

0 外部引脚信号不与比较器 A 连接

1 外部引脚信号与比较器 A 连接

P2CA0:控制输入端 CA0

0 外部引脚信号不与比较器 A 连接

1 外部引脚信号与比较器 A 连接

CAF: 选择比较器输出端是否经过 RC 低通滤波器

0 开通 RC 低通滤波器

1 直通信号

CAOUT: 比较器 A 输出的信号

0 CA0 小于 CA1

1 CA0 大于 CA1

CAPD 端口禁止寄存器

比较器 A 模块的输入输出与 IO 口共用引脚,可以控制 IO 端口输入缓冲器的通断开关.CAPD 控制位初始化为 0,则端口输入缓冲器有效.当相应位为 1 时,端口输入缓冲器无效。

程序范例：

```
#include <msp430x11x1.h>
void main (void)
  {
    WDTCTL = WDTPW WDTHOLD;              // 停止 WDT
    CAPD |= 0x08;                        // 断开与 IO 端口输入
    CACTL2 = P2CA0;                      // 设置 P2.3 为 comp
    CCTL0 = CCIE;                        // CCR0 允许中断
    TACTL = TASSEL_2 ID_3 MC_2;     // SMCLK/8,计数模式
    _EINT();                             // 开总中断
```

```
    while (1) //  循环
  {
    CACTL1 = 0x00;                          //  没有参考电压
    _BIS_SR(LPM0_bits);                     //  进入 LPM0
    CACTL1 = CAREF0 CAON;                     // 0.25*Vcc=P2.3,  比较器开
    _BIS_SR(LPM0_bits);                     //  再次进入 LPM0
    CACTL1 = CAREF1 CAON;                     // 0.5*Vcc=P2.3,  比较器开
    _BIS_SR(LPM0_bits);                     //  再次进入  LPM0
    CACTL1 = CAREF1 CAREF0 CAON;     // 0.55V on P2.3,比较器开
    _BIS_SR(LPM0_bits);                     //  再次进入 LPM0
  }
}

// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  _BIC_SR_IRQ(LPM0_bits);                   //退出 LMP0 模式
}
```

MSP430 教程 14：MSP430 单片机 ADC12 模块

MSP430 模数转换模块--ADC12

MSP430 单片机的 ADC12 模块是一个 12 位精度的 A/D 转换模块,他具有高速度,通用性等特点。大部分都内置了 ADC 模块.而有些不带 ADC 模块的片子,也可通过利用内置的模拟比较器来实现 AD 的转换。在系列产品中，我们可以通过以下列表来简单地认识他们的 ADC 功能实现。

系列型号 ADC 功能实现 转换精度
MSP430X1XX2 比较器实现 10 位
MSP430F13X ADC 模块 12 位
MSP430F14X ADC 模块 12 位
MSP430F43X ADC 模块 12 位
MSP430F44X ADC 模块 12 位
MSP430X32X ADC 模块 14 位

从以下 ADC12 结构图中可以看出，ADC12 模块中是由以下部分组成：输入的 16 路模拟开关，ADC 内部电压参考源，ADC12 内核，ADC 时钟源部分，采集与保持/触发源部分，ADC 数据输出部分，ADC 控制寄存器等组成。

输入的 16 路模拟开关
16 路模拟开关分别是由 IC 外部的 8 路模拟信号输入和内部 4 路参考电源输入及 1 路内部温度传感器源及 AVCC-AVSS/2 电压源输入。外部 8 路从 A0-A7 输入，主要是外部测量时的模拟变量信号。内部 4 路分别是 Veref ADC 内部参考电源的输出正端，Vref-/Veref- ADC 内部参考电源负端(内部/外部)。1 路 AVCC-AVSS/2 电压源和 1 路内部温度传感器源。片内温度传感器可以用于测量芯片上的温度，可以在设计时做一些有用的控制；在实际应用时用得较多。而其他电源参考源输入可以用作 ADC12 的校验之用，在设计时可作自身校准。

ADC 内部电压参考源
ADC 电压参考源是用于给 ADC12 内核作为一个基准信号之用的，这是 ADC 必不可少的一部分。在 ADC12 模块中基准电压源可以通过软件来设置 6 种不同的组合。AVCC(Vr)，Vref ,Veref ,AVSS(Vr-),Vref-/Veref-。

ADC12 内核
ADC12 的模块内核是共用的，通过前端的模拟开关来分别来完成采集输入。ADC12 是一个精度为 12 位的 ADC 内核，1 位非线性微分误差，1 位非线性积分误差。内核在转换时会参用到两个参考基准电压，一个是参考相对的最大输入最大值，当模拟开关输出的模拟变量大

于或等于最大值时 ADC 内核的输出数字量为满量程，也就是 0xfff；另一个则是最小值，当模拟开关输出的模拟变量大小或等于最大值时 ADC 内核的输出数字量为最低量程，也就是 0x00。而这两个参考电压是可以通过软件来编程设置的。

ADC 时钟源部分

ADC12 的时钟源分有 ADC12OSC，ACLK，MCLK，SMCLK。通过编程可以选择其中之一时钟源，同时还可以适当的分频。

采集与保持,触发源部分

ADC12 模块中有着较好的采集与保持电路，采用不的设置有着灵活的应用。关于这方面的详情请参考手册上的寄存器说明，此部分我们日后再作补上。

ADC 数据输出部分

ADC 内核在每次完成转换时都会将相应通道上的输出结果存贮到相应用通道缓冲区单元中，共有 16 个通道缓冲单元。同时 16 个通道的缓冲单元有着相对应的控制寄存器，以实现更灵活的控制。

ADC 控制寄存器

ADC12CTL0 转换控制寄存器 0

ADC12CTL1 转换控制寄存器 1

ADC12IE 中断使能寄存器

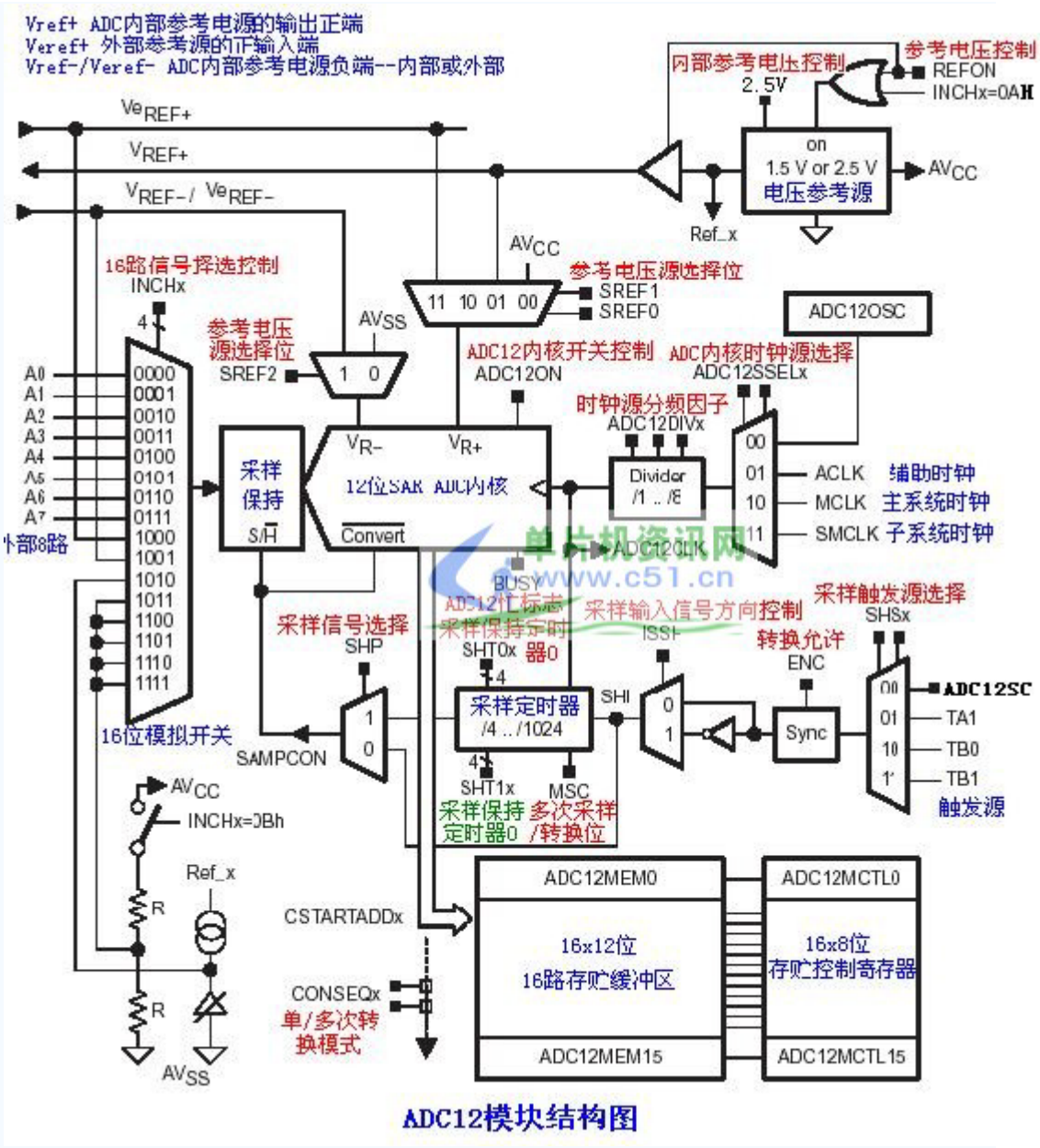
ADC12IFG 中断标志寄存器

ADC12IV 中断向量寄存器

ADC12MEM0-15 存储控制寄存器 0-15

ADC12MCTL0-15 存储控制寄存器 0-15

MSP430 ADC12 模块结构图



ADC12模块结构图

ADC12 应有例程

```
//*******************************************************************************
*


#include <msp430x14x.h>
//*****************************************
//表区
unsigned char number_table[]={'0','1','2','3','4','5','6','7','8','9'};
unsigned char display_buffer[]={0x00,0x00,0x00,0x00,0xff};

//******************波特率***********300 600 1200 2400 4800 9600 19200 38400 76800
115200const
```

```
//*************************************[0]**[1]**[2]*[3]**[4]**[5]***[6]***[7]****[8]****[9]*
unsigned char BaudrateUBR0[] ={0x6D,0x36,0x1B,0x0D,0x06,0x03, 0xA0, 0xD0, 0x68, 0x45};
unsigned const char BaudrateUBR1[] ={0x00,0x00,0x00,0x00,0x00,0x00, 0x01, 0x00, 0x00, 0x00};
unsigned const char BaudrateUMCTL[]={0x22,0xD5,0x03,0x6B,0x6F,0x4A, 0xC0, 0x40, 0x40, 0x4A};

unsigned char timp;
//变量区
unsigned int ADC0 ;
//子程序声明
void init (void); //初始化
void ADC12setup(void); //ADC12 初始化
void BaudrateSetup(unsigned char U0); //UART0 初始化
void data_converter(unsigned char *p,unsigned int vaule); //数据变换
void send_data(unsigned char *p); //串行口发送数组
//*****************************************
void main(void)
{
init();
//主循环
for (;;)
{
LPM0;
ADC12CTL0 |= ADC12SC; //sampling open,AD 转换完成后(ADC12BUSY=0),ADC12SC 自动复位;
while((ADC12IFG & BIT0) == 0); //等转换结束
ADC0 = ADC12MEM0; //读转换数据值,同时清 ADC12IFG0 标志
data_converter(display_buffer,ADC0); //数据变换
send_data(display_buffer); //发送数据
}
}

//*************************************************************************
***
void init(void)
```

```
{
WDTCTL = WDTPW WDTHOLD; //  停止 WDT
P1DIR=0x01;P1OUT=0x0f; //LED 设置
BaudrateSetup(6);
ADC12setup();
_EINT(); //  全局中断使能
}

//*************************************************************************
*****
//串口接收中断,退出 LPM0 模式.
#pragma vector=USART0RX_VECTOR
__interrupt void usart0_rx (void)
{
LPM0_EXIT;
}

//*************************************************************************
*****
//ADC12 初始化
void ADC12setup(void)
{
//ADC12 设置**************************
P6SEL |= 0x01; //使用 A/D 通道  A0
ADC12CTL0 = ADC12ON ; //开 ADC12 内核,设 SHT0=2 (N=4)
ADC12CTL1 = SHP ; //SAMPCON 信号选为采样定时器输出
//ADC12 内部参考电压设置
ADC12CTL0 |= REF2_5V; //选用内部参考电压为 2.5V
ADC12CTL0 |= REFON; //内部参考电压打开
ADC12MCTL0 |= SREF_1; //R =2.5V R-=VSS
//转换允许
ADC12CTL0 |= ENC ; //转换允许(上升沿)
ADC0=0x00;
}

//*************************************************************************
*****
```

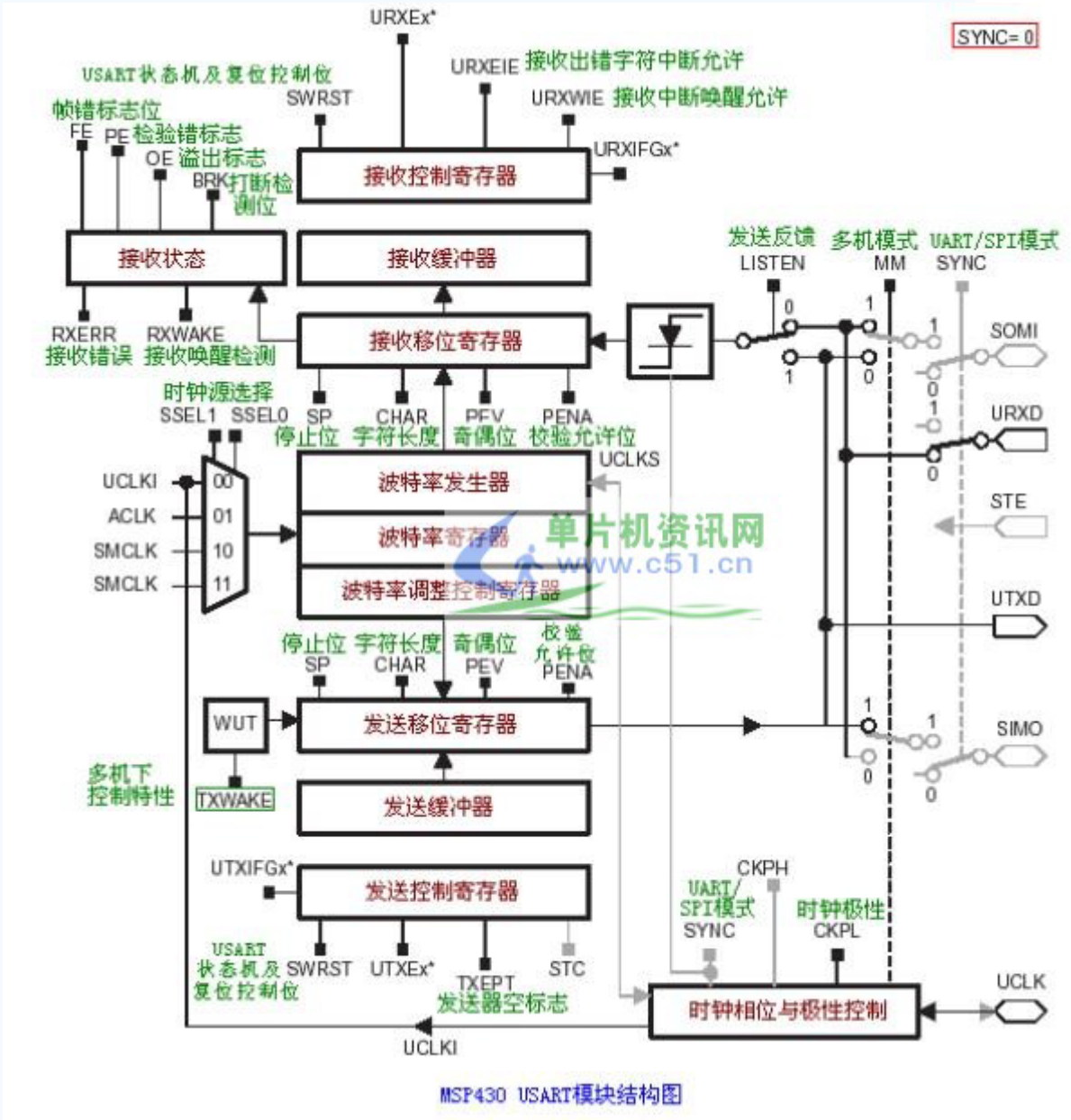
```
//UART0 初始化
void BaudrateSetup(unsigned char U0)
{
unsigned int i;
if(U0>5) //当 U0>5 时,启用 XT2
{
BCSCTL1 &= ~XT2OFF; //启动 XT2,
do
{ IFG1 &= ~OFIFG; //清 OSCFault 标志
for(i=0xFF;i>0;i--); //延时等待
}
while((IFG1 & OFIFG) != 0); //查 OSCFault,为 0 时转换完成
BCSCTL2 |= SELS; //SMCLK 为 XT2
}
//UART0
P1OUT=0x00;
if(U0>5){UTCTL0=SSEL1;} // 时钟源:SMCLK
else{UTCTL0=SSEL0;} // 时钟源:ACLK
UCTL0 &= ~SWRST; // SWRST 复位， USART 允许
UCTL0=CHAR; // 8bit
ME1|=UTXE0 URXE0; // Enable Tx0,Rx0
IE1|=URXIE0; // RX 使能
UBR00=BaudrateUBR0[U0]; // 低位分频器因子
UBR10=BaudrateUBR1[U0]; // 高位分频器因子
UMCTL0=BaudrateUMCTL[U0]; // 波特率调整因子
P3SEL |= 0x30; // 将 P3.4,5 使用外围模块 = USART0 TXD/RXD
P3DIR |= 0x10; // 将 P3.4 设为输出(发),P3.5 默认为输入(收)
}

//*************************************************************************
*****
//数据变换
void data_converter(unsigned char *p,unsigned int value)
{
unsigned int m,n,j=0;
p[0]=number_table[value/1000];
m=value┼ 00;
```

```
p[1]=number_table[m/100];
n=m┼ 0;
p[2]=number_table[n/10];
j=n┼ ;
p[3]=number_table[j/1];
}

//*****************************************************************************
*****
//串行口发送数组
void send_data(unsigned char *p)
{unsigned int n;
timp=RXBUF0;
for(n=0;p[n]!=0xff;n )
{
while ((IFG1 & UTXIFG0) == 0); // USART0 发送 UTXIFG0=1,表示 UTXBUF 准备好发送一
下字符
TXBUF0 = p[n];
}
}
//*****************************************************************************
*****
//ADC12 模块例程结束
```

MSP430 教程 15：MSP430 单片机串行通讯模块

串行异步通讯模块

在 MSP430 系列产品中,每一款型号都能实现串行通讯的功能,在 MSP430F1X11 系列中,可以用定时器 A 和软件编程结合实现串行通讯功能。而在其它系列产品中都带有硬件的串行通讯模块 USART；另外，MSP430F14X 系列产品中还带了两个串行通讯模块。而在 MSP430F15X，F16X 系列中 USART0 还可以实现 IIC 总线通讯。在 UART 模块中带有 UART 串行异步通讯和 SPI 同步通讯硬件资源。



MSP430 USART模块结构图

图是 USART 模块结构图，从下图可以看出 USART 模块分别由波特率部分：波特率发生器，串行通讯接收/发送控制寄存器。

接收部分，发送部分，端口 IO 部分。

串行异步通讯的特点：
1-异步通讯模式，包括线路空闲/地址位通信协议。
2-有两个单独的移位寄存器，输入/输出移位寄存器（如下图）。
3-传输 7 位或 8 位数据，可采用奇偶或无校验。
4-可编程实现波特率调整。
5-分别发，收单独中断。
6-有效地检测到起始位实现从低功耗唤醒。
7-状态标志检测错误或者地址位。

串行同步通讯（SPI）的特点：
1-动持 3 线/4 线的 SPI 通讯。
2-支持主机模式与从机模式。
3-收发有单独的缓冲器，移位寄存器。
4-收发有单独的中断。
5-时钟极性和相位可编程。
6-主机模式的时钟频率可编程。
7-7 位/8 位字符长度。

有关 USART 的详细应有原理，建议初学者参考清华大学出版社的<<MSP430 系列 16 位超低功耗单片机原理与应用>>。

串行异步通讯应用例程

```
//*****************************************
//MSP430F149 串口行实验程序
//P3.4 为发送,P3.5 为接收
//晶体使 32768HZ.
//程序描述:利用串口调试软件;向串口发送一个字符,MSP430 单片机接收到后从低功耗中唤醒.并将收接缓冲区的字符再发送到//电脑上的调试软件中.单片机发送完后又进入低功耗状态.
//*****************************************

#include <msp430x14x.h>

void main(void)
{
```

```
WDTCTL = WDTPW WDTHOLD; //  停止 WDT

UCTL0 = CHAR; //  设串口控制寄存器,设为 8 位字符格式

UTCTL0 = SSEL0; //  设串口控制寄存器所使用的时钟,选择 UCLK = ACLK

UBR00 = 0x0D; //  波特率设置 32k/2400 - 13.65
UBR10 = 0x00;

UMCTL0 = 0x6D; //  波特率调整器设置

ME1 |= UTXE0 URXE0; //  模块允许寄存器设置,使能 USART0 TXD/RXD

IE1 |= URXIE0; //  中断允许寄存器设置,接收中断允许

P3SEL |= 0x30; //  将 P3.4,5 使用外围模块  = USART0 TXD/RXD
P3DIR |= 0x10; //  将 P3.4 设为输出(发),P3.5 默认为输入(收)

_EINT(); //  全局中断使能

// Mainloop
for (;;)
{
LPM3; //  进入 LPM3 模式,等待字符接收.

while ((IFG1 & UTXIFG0) == 0); // USART0 发送 UTXIFG0=1,表示 UTXBUF 准备好发送一
下字符

TXBUF0 = RXBUF0; //  将收到缓冲区字符送发送区
}
}

//串口接收中断,退出 LPM3 模式.
#pragma vector=USART0RX_VECTOR
__interrupt void usart0_rx (void)
{
LPM3_EXIT;
```

```
}

//*************************************
//例程结束
```

MSP430 教程 16：MSP430 单片机的框架程序

MSP430 单片机的框架程序(转)下面给出 MSP430 的程序框架，我们可以在此基础上修改以及添加自己所需的程序。

```
/*****************************************************************************
*
文件名：main.c
描述：MSP430 框架程序。适用于 MSP430F149，其他型号需要适当改变。
不使用的中断函数保留或者删除都可以，但保留时应确保不要打开不需要的中断。
保留中断函数，编译器将会为 BSL 密码填充所有的字节。
版本：1.0 2005-1-13
*****************************************************************************/
//头文件
＃i nclude <MSP430x14x.h>
//函数声明
void InitSys();

int main( void )
{
WDTCTL = WDTPW WDTHOLD; //关闭看门狗
InitSys(); //初始化
start:
//以下填充用户代码

LPM3; //进入低功耗模式 n，n：0~4。若不希望进入低功耗模式，屏蔽本句
goto start;
}
/*****************************************************************************
*
系统初始化
*****************************************************************************/
void InitSys()
{
unsigned int iq0;
//使用 XT2 振荡器
BCSCTL1&=~XT2OFF; //打开 XT2 振荡器
do
```

```
{
IFG1 &= ~OFIFG; // 清除振荡器失效标志
for (iq0 = 0xFF; iq0 > 0; iq0--); // 延时，等待 XT2 起振
}
while ((IFG1 & OFIFG) != 0); // 判断 XT2 是否起振
    BCSCTL2 =SELM_2 SELS; //选择 MCLK、SMCLK 为 XT2
    //以下填充用户代码，对各种模块、中断、外围设备等进行初始化
    _EINT(); //打开全局中断控制，若不需要打开，可以屏蔽本句
}
    /*****************************************************************************
*
端口 2 中断函数
*****************************************************************************/
#pragma vector=PORT2_VECTOR
__interrupt void Port2()
{
//以下为参考处理程序，不使用的端口应当删除其对于中断源的判断。
if((P2IFG&BIT0) == BIT0)
{
//处理 P2IN.0 中断
P2IFG &= ~BIT0; //清除中断标志
//以下填充用户代码
    }
else if((P2IFG&BIT1) ==BIT1)
{
//处理 P2IN.1 中断
P2IFG &= ~BIT1; //清除中断标志
//以下填充用户代码
    }
else if((P2IFG&BIT2) ==BIT2)
{
//处理 P2IN.2 中断
P2IFG &= ~BIT2; //清除中断标志
//以下填充用户代码
    }
else if((P2IFG&BIT3) ==BIT3)
{
```

```
//处理 P2IN.3 中断
P2IFG &= ~BIT3; //清除中断标志
//以下填充用户代码
    }
else if((P2IFG&BIT4) ==BIT4)
{
//处理 P2IN.4 中断
P2IFG &= ~BIT4; //清除中断标志
//以下填充用户代码
    }
else if((P2IFG&BIT5) ==BIT5)
{
//处理 P2IN.5 中断
P2IFG &= ~BIT5; //清除中断标志
//以下填充用户代码
    }
else if((P2IFG&BIT6) ==BIT6)
{
//处理 P2IN.6 中断
P2IFG &= ~BIT6; //清除中断标志
//以下填充用户代码
    }
else
{
//处理 P2IN.7 中断
P2IFG &= ~BIT7; //清除中断标志
//以下填充用户代码
    }
    LPM3_EXIT; //退出中断后退出低功耗模式。若退出中断后要保留低功耗模式，将本句屏
蔽
}
    /*****************************************************************************
*
USART1 发送中断函数
*****************************************************************************/
#pragma vector=USART1TX_VECTOR
__interrupt void Usart1Tx()
```

```
{
//以下填充用户代码

LPM3_EXIT; //退出中断后退出低功耗模式。若退出中断后要保留低功耗模式，将本句屏蔽
}
   /*****************************************************************************
*
USART1 接收中断函数
*****************************************************************************/
#pragma vector=USART1RX_VECTOR
__interrupt void Ustra1Rx()
{
//以下填充用户代码

LPM3_EXIT; //退出中断后退出低功耗模式。若退出中断后要保留低功耗模式，将本句屏蔽
}
   /*****************************************************************************
*
端口 1 中断函数
多中断中断源：P1IFG.0~P1IFG7
进入中断后应首先判断中断源，退出中断前应清除中断标志，否则将再次引发中断
*****************************************************************************/
#pragma vector=PORT1_VECTOR
__interrupt void Port1()
{
//以下为参考处理程序，不使用的端口应当删除其对于中断源的判断。
if((P1IFG&BIT0) == BIT0)
{
//处理 P1IN.0 中断
P1IFG &= ~BIT0; //清除中断标志
//以下填充用户代码
   }
else if((P1IFG&BIT1) ==BIT1)
{
//处理 P1IN.1 中断
P1IFG &= ~BIT1; //清除中断标志
//以下填充用户代码
```

```
    }
else if((P1IFG&BIT2) ==BIT2)
{
//处理 P1IN.2 中断
P1IFG &= ~BIT2; //清除中断标志
//以下填充用户代码
    }
else if((P1IFG&BIT3) ==BIT3)
{
//处理 P1IN.3 中断
P1IFG &= ~BIT3; //清除中断标志
//以下填充用户代码
    }
else if((P1IFG&BIT4) ==BIT4)
{
//处理 P1IN.4 中断
P1IFG &= ~BIT4; //清除中断标志
//以下填充用户代码
    }
else if((P1IFG&BIT5) ==BIT5)
{
//处理 P1IN.5 中断
P1IFG &= ~BIT5; //清除中断标志
//以下填充用户代码
    }
else if((P1IFG&BIT6) ==BIT6)
{
//处理 P1IN.6 中断
P1IFG &= ~BIT6; //清除中断标志
//以下填充用户代码
    }
else
{
//处理 P1IN.7 中断
P1IFG &= ~BIT7; //清除中断标志
//以下填充用户代码
    }
```

```
    LPM3_EXIT; //退出中断后退出低功耗模式。若退出中断后要保留低功耗模式，将本句屏蔽
}
    /*****************************************************************************
*
定时器 A 中断函数
多中断中断源：CC1~2 TA
*****************************************************************************/
#pragma vector=TIMERA1_VECTOR
__interrupt void TimerA1()
{
//以下为参考处理程序，不使用的中断源应当删除
switch (__even_in_range(TAIV, 10))
{
case 2:
//捕获/比较 1 中断
//以下填充用户代码
    break;
case 4:
//捕获/比较 2 中断
//以下填充用户代码
    break;
case 10:
//TAIFG 定时器溢出中断
//以下填充用户代码
    break;
}

LPM3_EXIT; //退出中断后退出低功耗模式。若退出中断后要保留低功耗模式，将本句屏蔽
}
    /*****************************************************************************
*
定时器 A 中断函数
中断源：CC0
*****************************************************************************/
#pragma vector=TIMERA0_VECTOR
__interrupt void TimerA0()
```

```
{
//以下填充用户代码

LPM3_EXIT; //退出中断后退出低功耗模式。若退出中断后要保留低功耗模式，将本句屏蔽
}
  /*****************************************************************************
*
AD 转换器中断函数
多中断源：摸拟 0~7、VeREF 、VREF-/VeREF-、（AVcc-AVss）/2
没有处理 ADC12TOV 和 ADC12OV 中断标志
*****************************************************************************/
#pragma vector=ADC_VECTOR
__interrupt void Adc()
{
//以下为参考处理程序，不使用的中断源应当删除
if((ADC12IFG&BIT0)==BIT0)
{
//通道 0
//以下填充用户代码
  }
else if((ADC12IFG&BIT1)==BIT1)
{
//通道 1
//以下填充用户代码
  }
else if((ADC12IFG&BIT2)==BIT2)
{
//通道 2
//以下填充用户代码
  }
else if((ADC12IFG&BIT3)==BIT3)
{
//通道 3
//以下填充用户代码
  }
else if((ADC12IFG&BIT4)==BIT4)
{
```

```
//通道 4
//以下填充用户代码
    }
else if((ADC12IFG&BIT5)==BIT5)
{
//通道 5
//以下填充用户代码
    }
else if((ADC12IFG&BIT6)==BIT6)
{
//通道 6
//以下填充用户代码
    }
else if((ADC12IFG&BIT7)==BIT7)
{
//通道 7
//以下填充用户代码
    }
else if((ADC12IFG&BIT8)==BIT8)
{
//VeREF
//以下填充用户代码
    }
else if((ADC12IFG&BIT9)==BIT9)
{
//VREF-/VeREF-
//以下填充用户代码
    }
else if((ADC12IFG&BITA)==BITA)
{
//温度
//以下填充用户代码
    }
else if((ADC12IFG&BITB)==BITB)
{
//（AVcc-AVss）/2
//以下填充用户代码
```

```
    }
    LPM3_EXIT; //退出中断后退出低功耗模式。若退出中断后要保留低功耗模式，将本句屏蔽
}
    /*****************************************************************************
*
USART0 发送中断函数
*****************************************************************************/
#pragma vector=USART0TX_VECTOR
__interrupt void Usart0Tx()
{
//以下填充用户代码

LPM3_EXIT; //退出中断后退出低功耗模式。若退出中断后要保留低功耗模式，将本句屏蔽
}
    /*****************************************************************************
*
USART0 接收中断函数
*****************************************************************************/
#pragma vector=USART0RX_VECTOR
__interrupt void Usart0Rx()
{
//以下填充用户代码

LPM3_EXIT; //退出中断后退出低功耗模式。若退出中断后要保留低功耗模式，将本句屏蔽
}
    /*****************************************************************************
*
看门狗定时器中断函数
*****************************************************************************/
#pragma vector=WDT_VECTOR
__interrupt void WatchDog()
{
//以下填充用户代码

LPM3_EXIT; //退出中断后退出低功耗模式。若退出中断后要保留低功耗模式，将本句屏蔽
}
```

```
    /*****************************************************************************
*
比较器 A 中断函数
*****************************************************************************/
#pragma vector=COMPARATORA_VECTOR
__interrupt void ComparatorA()
{
//以下填充用户代码

LPM3_EXIT; //退出中断后退出低功耗模式。若退出中断后要保留低功耗模式，将本句屏蔽
}
    /*****************************************************************************
*
定时器 B 中断函数
多中断源：CC1~6 TB
*****************************************************************************/
#pragma vector=TIMERB1_VECTOR
__interrupt void TimerB1()
{
//以下为参考处理程序，不使用的中断源应当删除
switch (__even_in_range(TBIV, 14))
{
case 2:
//捕获/比较 1 中断
//以下填充用户代码
    break;
case 4:
//捕获/比较 2 中断
//以下填充用户代码
    break;
case 6:
//捕获/比较 3 中断
//以下填充用户代码
    break;
case 8:
//捕获/比较 4 中断
//以下填充用户代码
```

```
    break;
case 10:
//捕获/比较 5 中断
//以下填充用户代码
    break;
case 12:
//捕获/比较 6 中断
//以下填充用户代码
    break;
case 14:
//TBIFG 定时器溢出中断
//以下填充用户代码
    break;
}
    LPM3_EXIT; //退出中断后退出低功耗模式。若退出中断后要保留低功耗模式，将本句屏蔽
}
    /*****************************************************************************
*
定时器 B 中断函数
中断源：CC0
*****************************************************************************/
#pragma vector=TIMERB0_VECTOR
__interrupt void TimerB0()
{
//以下填充用户代码
    LPM3_EXIT; //退出中断后退出低功耗模式。若退出中断后要保留低功耗模式，将本句屏蔽
}
    /*****************************************************************************
*
不可屏蔽中断函数
*****************************************************************************/
#pragma vector=NMI_VECTOR
__interrupt void Nmi()
{
//以下为参考处理程序，不使用的中断源应当删除
```

```
if((IFG1&OFIFG)==OFIFG)
{
//振荡器失效
IFG1 &= ~OFIFG;
//以下填充用户代码
   }
else if((IFG1&NMIIFG)==NMIIFG)
{
//RST/NMI 不可屏蔽中断
IFG1 &= ~NMIIFG;
//以下填充用户代码
   }
else //if((FCTL3&ACCVIFG)==ACCVIFG)
{
//存储器非法访问
FCTL3 &= ~ACCVIFG;
//以下填充用户代码
   }
   LPM3_EXIT; //退出中断后退出低功耗模式。若退出中断后要保留低功耗模式，将本句屏蔽
}
   /*****************************************************************************
*
基本定时器中断函数
*****************************************************************************/
#pragma vector=BASICTIMER_VECTOR
__interrupt void BasTimer()
{
//以下填充用户代码
   LPM3_EXIT; //退出中断后退出低功耗模式。若退出中断后要保留低功耗模式，将本句屏蔽
}
```

MSP430 教程 17：MSP430 单片机开发总结

1.＃i nclude<>指要在编辑器设定目录下，＃i nclude""指的是在当前工程目录下。

2.要调用另一个文件中的函数，要把这个函数文件放到当前工程目录下，并且在工程中添加此文件。

3.命名中不能有-,比如:byq-ee 会认为是错误的，要用下划线。

4.用 IAR 软件仿真时，可以加入变量，如果是查看 I/O 信息只需加入 PXIN,PXOUT 即可。

5.IAR 在处理字符时，要注意，是字符处理结尾标志，他和其他编辑软件是不同的。比如我们长用字符处理回

自动在结尾处加，但 IAR 有些是不加的，这就要十分注意。

6.如果只用到 LFX1 的低速时钟，9600bit/s 传输的话，接收会出现问题，原因是误差太大，可以设置到 4800 以下。

7.在写 FLASH 时要注意其工作频率在 257K~476k 之间，如果不是，则会出现错误。而且 FLAGH 只能写入 0，这样就出现了

必须先擦除在写入的模式。

8.当 IO 口作为输入时，要根据平时的状态加电阻，平时为高时，加个上拉电阻，平时为低时加个下拉电阻以增加稳定性。

9.在 FLASH 写时一定要关外部中断。

10.MSP430 一般是不要 RC 复位的，一般只要接个 100K 左右电阻就可以了，如果要加电容，它的大小要根据以下两个标准

选择：

下载程序不会出现下载不了

程序上电会能稳定复位

11.用&表达式作为判断时，不要忘记加括号。

12.不要使用中断嵌套。

同时，为了使用 C 语言来编写 MSP430 的高质量代码需要注意。

微处理器一般用于特定环境和特定用途,出于成本、功耗和体积的考虑，一般都要求尽量节省使用资源，并且，由于微处理器硬件一般都不支持有符号数、浮点数的运算，且运算位有限，因此，分配变量时必须仔细。另外要说明的是，速度和存储器的消耗经常是 2 个不可兼顾的目标，在多数情况下，编程者必须根据实际情况作出权衡和取舍。

需要注意的事项如下：

1) 通常在满足运算需求的前提下，尽量选择为变量定义字节少的数据类型。

比如最常用的 int 和 char，int 是 16 位的，char 是 8 位的，如果没有必要，不要使用 int，而且使用 char 也最好使用 unsigned char。运行时，可以在变量窗口看到，使用类型为 unsigned char 的变量是 16 进制的格式，而使用 int 的是十进制格式，如果 char 没有定义为 unsigned，会出现负号，如果没有必要的话，在 430 中是不需要负数的。

2) 尽量不用过长的数据类型，如 long、long long 和 double

3) MSP430 的 C 编译器不支持位寻址，所以运算中尽量减少位操作，对于只有“是”和“否”的变量，如果 RAM 容量允许，则可分配为 unsigned char 类型，可提高运算速度。如果分配为某字节的某个位，可以减少存储器的消耗，但是会降低运算速度

4) 避免使用浮点数，尽量使用定点数进行小数运算。如果必须使用浮点数，则尽量用 32 位的 float，而不是 64 位的 double

5) 尽量将变量分配为无符号数据类型

6) 对于指针变量，如果声明后其值不再改变，则声明为 const 类型，这样编译器编译时能更好的优化生成的代码

7) 尽可能的使用局部变量而非全局变量或者静态变量（static）。这样有利于编译器编译时更好的优化生成的代码

8) 避免对局部变量使用 &取地址符。因为这样会使编译器无法把此变量放在 CPU 的寄存器中，而是放在 RAM 中，从而失去了优化的机会

9) 仅在模块内使用的变量声明为 static，有利于优化代码

10) 如果堆栈空间有限，尽量减少函数调用的层次和递归调用

11) 如果传送参数过多，可将参数组成一个数组或者结构体，然后用指针传递

12) 某些变量在中断程序和普通级别程序中都会被用到，所以必须加以保护。

将变量声明为 volatile 类型，编译器优化时就不会移动它，对它的访问不会被延迟。

为保证对 volatile 的变量不被打断，为此，可以在访问它的部分（即访问它的函数）前面加上__moniter 的声明