MSP430 系列十六位超低功耗单片机
教学实验系统实验教程

赵建 谢楷 沈雪亮 张宝 梁海军 杨乐林 度明光 徐常志 编写

西安电子科技大学测控工程与仪器系

2006 年 5 月

第一部分

MSP430 系列单片机系统原理

第一章 MSP430 单片机概述

MSP430 系列单片机是美国德州仪器(TI)1996 年开始推向市场的一种 16 位超低功耗的混合信号处理器(Mixed Signal Pocessor)。称之为混合信号处理器，主要是由于其针对实际应用需求，把许多模拟电路、数字电路和微处理器集成在一个芯片上，以提供“单片”解决方案。

1.1 MSP430 系列单片机的特点

虽然 MSP430 系列单片机推出时间不是很长，但由于其卓越的性能，在短短几年时间里发展极为迅速，应用也日趋广泛。MSP430 系列单片机针对各种不同应用，包括一系列不同型号的器件。主要特点有：

1．超低功耗

MSP430 系列单片机的电源电压采用 1.8~3.6V 低电压，RAM 数据保持方式下耗电仅 0.1uA，活动模式耗电 250pA／MIPS(MIPS：每秒百万条指令数)，IO 输入端口的漏电流最大仅 50nA。

MSP430 系列单片机有独特的时钟系统设计，包括两个不同的时钟系统：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。由时钟系统产生 CPU 和各功能模块所需的时钟，并且这些时钟可以在指令的控制下打开或关闭，从而实现对总体功耗的控制。由于系统运行时使用的功能模块不同，即采用不同的工作模式，芯片的功耗有明显的差异。在系统中共有种活动模式(AM)和 5 种低功耗模式(LPM0~LPM4）。

另外，MSP430 系列单片机采用矢量中断，支持十多个中断源，并可以任意嵌套。用中断请求将 CPU 唤醒只要 6us，通过合理编程，既以降低系统功耗，又可以对外部事件请求作出快速响应。

在这里．需要对低功耗问题作一些说明。

首先，对一个处理器而言，活动模式时的功耗必须与其性能一起来考察、衡量，忽略性能来看功耗是片面的。在计算机体系结构中，是用 W／MIPS(瓦特／百万指令每秒)来衡量处理器的功耗与性能关系的，这种标称方法是合理的。MSP430 系列单片机在活动模式时耗电 250uA／MIPS，这个指标是很高的(传统的 Mcs51 单片机约为 10～20mA／MIPS)。　　其次，作为一个应用系统，功耗是整个系统的功耗，而不仅仅是处理器的功耗。比如，在一个有多个输入信号的应用系统中，处理器输入端口的漏电流对系统的耗电影响就较大了。MSP430 单片机输入端口的漏电流最大为 50nA，远低于其他系列单片机(一般为 l~10uA)。

另外，处理器的功耗还要看它内部功能模块是否可以关闭．以及模块活动情况下的耗电．比如低电压监测电路的耗电等。还要注意，有些单片机的某些参数指标中．虽然典型值可能很小，但最大值和典型值相差数十倍，而设计时要考虑到最坏情况，就应该关心参数标称的最大值，而不是典型值。总体而言，MSP430 系列单片机堪称目前世界上功耗最低的单片机，其应用系统可以做到用一枚电池使用 10 年。

2．强大的处理能力

MSP430 系列单片机是 16 位单片机，采用了目前流行的、颇受学术界好评的精简指令集(RISC)结构，一个时钟周期可以执行一条指令(传统的 MCS51 单片机要 12 个时钟周期才可以执行一条指令)，使 MSP430 在 8MHz 晶振工作时，指令速度可达 8MIPS(注意：同样 8MIPS 的指令速度，在运算性能上 16 位处理器比 8 位处理器高远不止两倍)。不久还将推出 25～30MIPS 的产品。

同时，MSP430 系列单片机中的某些型号，采用了一股只有 DSP 中才有的 16 位多功能硬件乘法器、硬件乘．加(积之和)功能、DMA 等一系列先进的体系结构，大大增强了它的数据处理和运算能力，可以有效地实现一些数字信号处理的算法(如 FFT、DTMF 等)。这种结构在其他系列单片机中尚未使用。

3．高性能模拟技术及丰富的片上外围模块

MSP430 系列单片机结合 TI 的高性能模拟技术，各成员都集成了较丰富的片内外设。视型号不同可能组合有以下功能模块：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、I)，硬件乘法器，液晶驱动器，10 位，12，14 位 ADC，12 位 DAC，1^2C 总线，直接数据存取(DMA)，端口 1-6(P1-P6)，基本定时器(Basic Timer)等。

其中，看门狗可以在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，可设计出高精度(10~11 位)的 A/D 转换器：16 位定时器(Timer A 和 TimerB)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；多功能串口(USART)可实现异步、同步和 1^2C 串行通信，可方便地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，IO 输出时，不管是灌电流还是拉电流，每个端口的输出晶体管都能够限制输出电流(最大约 25mA)，保证系统安全：PI、P2 端口能够接收外部上升沿或下降沿的中断输入；12 位 A/D 转换器有较高的转换速率，最高可达 200Kb/s，能够满足大多数数据采集应用：LCD 驱动模块能直接驱动液晶多达 160 段；F15x 和 F16x 系列有两路 12 位高速 DAC，可以实现直接数字波形合成等功能：硬件 1^2C 串行总线接口可以扩展 1^2C 接口器件：DMA 功能可以提高数据传输速度，减轻 CPU 的负荷。

MSP430 系列单片机的丰富片内外设，在目前所有单片机系列产品中是非常突出的，为系统的单片解决方案提供了极大的方便。

4．系统工作稳定

上电复位后，首先由 DCO_CLK 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。这种结构和运行机制，在目前各系列单片机中是绝无仅有的。另外，MSP430 系列单片机均为工业级器件，运行环境温度为 -40~+85"C，运行稳定、可靠性高，所设计的产品适用于各种民用和工业环境。

5．方便高效的开发环境

目前 MSF430 系列有 OTF 型、FLASH 型和 ROM 型 3 种类型的器件，国内大量使用的是 FLASH 型。这些器件的开发手段不同，对于 OTF，型和 ROM 型的器件是使用专用仿真器开发成功之后再烧写或掩膜芯片。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，因此采用先通过 JTAG 接口下载程序到 FLASH 内，再由 JTAG 接口控制程序运行、读取片内 CPU 状态，以及存储器内容等信息供设计者调试，整个开发(编译、调试)都可以在同一个软件集成环境中进行。这种方式只需要一台 PC 机和一个 JTAG 调试器，而不需要专用仿真器和编程器。开发语言有汇编语言和 C 语言。目前较好的软件开发工具是 IAR WORKBENCH V3.10。这种以 FLASH 技术、JTAG 调试、集成开发环境结合的开发方式，具有方便、廉价、实用等优点，在单片机开发中还较为少见。其他系列单片机的开发一般均需要专用的仿真器或编程器。另外，2001 年 TI 公司又公布了 BOOTSTRAP 技术，利用它可在保密熔丝烧断以后，只要几根硬件连线，通过软件口令字(密码)，就可更改并运行内部的程序，这为系统固件的升级提供了又一方便的手段。BOOTSTRAP 具有很高的保密性，口令字可达 32 个字节长度。

1.2 MSP430 系列单片机的发展和应用

TI 公司从 1996 年推出 MSP430 系列开始到 2000 年初，推出了 33x、32X、3lx 等几个系列。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。每个系列有 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。MSP430 的 3XX 系列，在国内几乎没有使用。随着 FLASH 技术的迅速发展，TI 公司也将这一技术引入 MSP430 系列单片机中。2000 年推出了 F11X / 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。在 2000 年 7 月推出了带 ADC 或硬件乘法器的 F13x、F14x 系列。在 2001 年 7 月到 2002 年又相继推出了带 LCD 控制器的 F4lx、F43x、F44x。TI 在 2003 到 2004 年期间推出了 F15x 和 F16x 系列产品。在这一新的系列中，有了两个方面的发展。一是增加了 RAM 的容量，如 F1611 的 RAM 容量增加到了 10KB，这样就可以引入实时操作系统(RTOS)或简单文件系统等。二是从外围模块来说，增加了 I^2C、DMA、DAC12 和 SVS 等模块。近两年，TI 公司针对某些特殊应用领域，利用 MSP430 的超低功耗特性，还推出了些专用单片机，如专门用于电量计量的 MSP430FE42x，用于水表、气表、热表等具有无磁传感模块的 MsP430FW42x，以及用于人体医学监护(血糖、血压、脉搏等)的 MSP430FG42X 单片机。用这些单片机来设计相应的专用产品，不仅具有 MSP430 的超低功耗特性，还能大大简化系统设计。根据 TI 在 MsP430 系列单片机上的发展计划．在今后将陆续推出性能更高、功能更强的 F5XX 系列，这一系列单片机运行速度可达 25~30MIPS，并具有更大的 FLASH(128KB)及更丰富的外设接 ISP(CAN、USB 等)。

MSP430 系列单片机不仅可以应用于许多传统的单片机应用领域，如仪器仪表、自动控制以及消费品领域，更适合用于一些电池供电的低功耗产品，如能量表(水表、电表、气表等)、手持式设备、智能传感器等，以及需要较高运算性能的智能仪器设备。

第二章 MSP430F169 单片机简介

2.1 特点

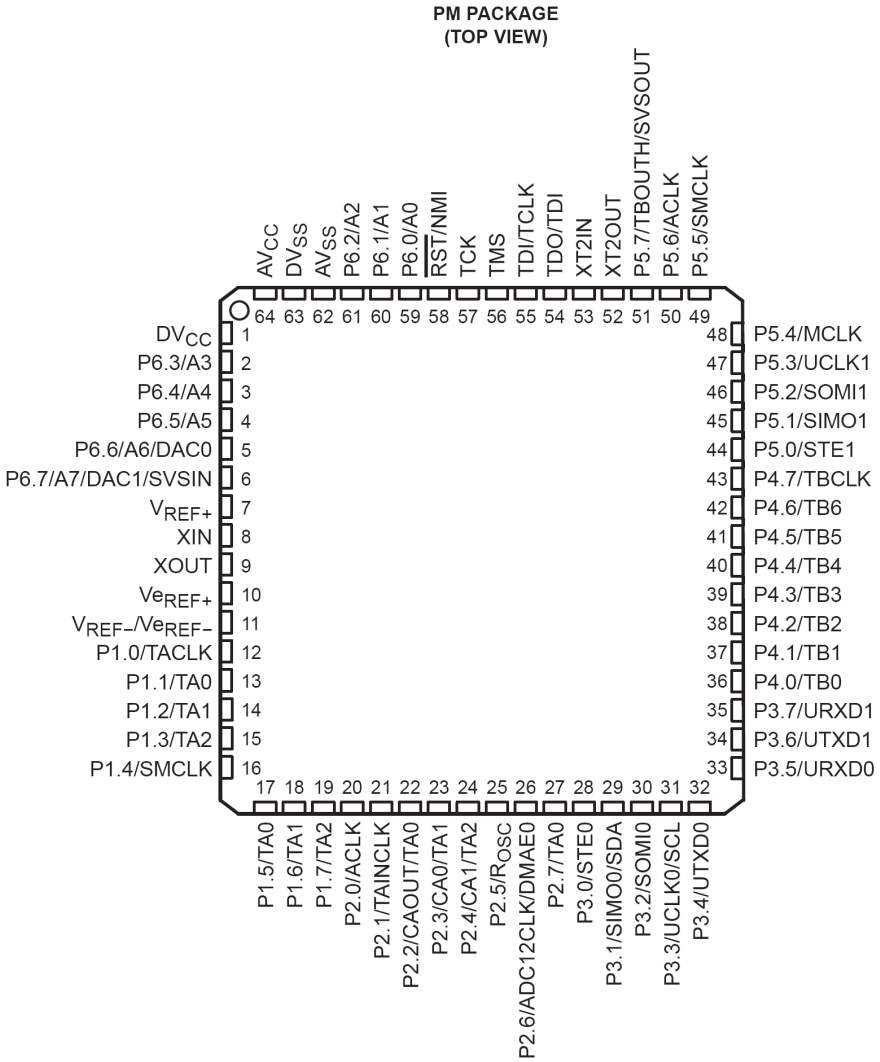
- 工作电压范围：1.8～3.6V
- 超低功耗：
  活动模式：330uA，@1MHz，2.2V
  待机模式：1.1uA
  关闭模式(RAM 保持)：0.2uA
  5 种省电模式
- 从等待方式唤醒时间：6us
- 16 位 RISC 结构，125ns 指令周期
- 内置三通道 DMA。
- 12 位 A / D 带采样保持内部参考源。
- 双 12 位 D / A 同步转换。
- 16 位定时器 Timer_A。
- 16 位定时器 Timer B。
- 片内比较器 A
- 串行通信 USART0(UART、SPI、I2C)接口
- 串行通信 USARTI(UART、SPI)接口
- 具有可编程电平检测的供电电压管理器，监视器。
- 欠电压检测器
- Bootstrap Loader
- 串行在线编程，无需外部编程电压，可编程的保密熔丝代码保护

2.2 器件系列

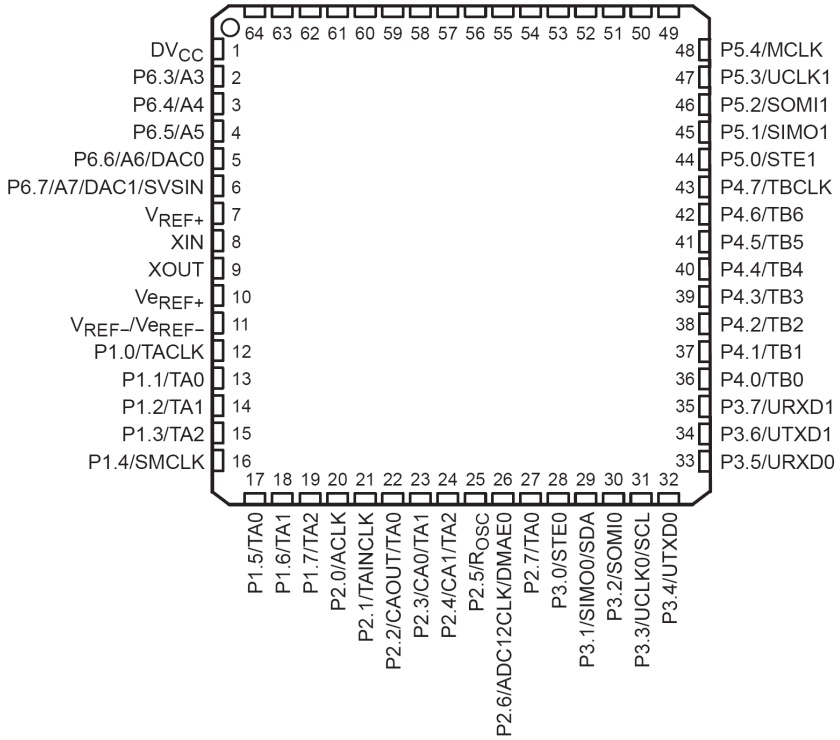
- MSP430F167：32KB+256B flash 存储器 1KB RAM
- MSP430F168：48KB+256B flash 存储器 2KB RAM
- MSP430F169：60KB+256B flash 存储器 2KB RAM
- MSP430F1610：32KB+256B flash 存储器 5KB RAM
- MSP430F1611：48KB+256Bflash 存储器 10KBRAM
- MSP430F1612：55KB+256B flash 存储器 5KB RAM

2.3 封装和引脚图

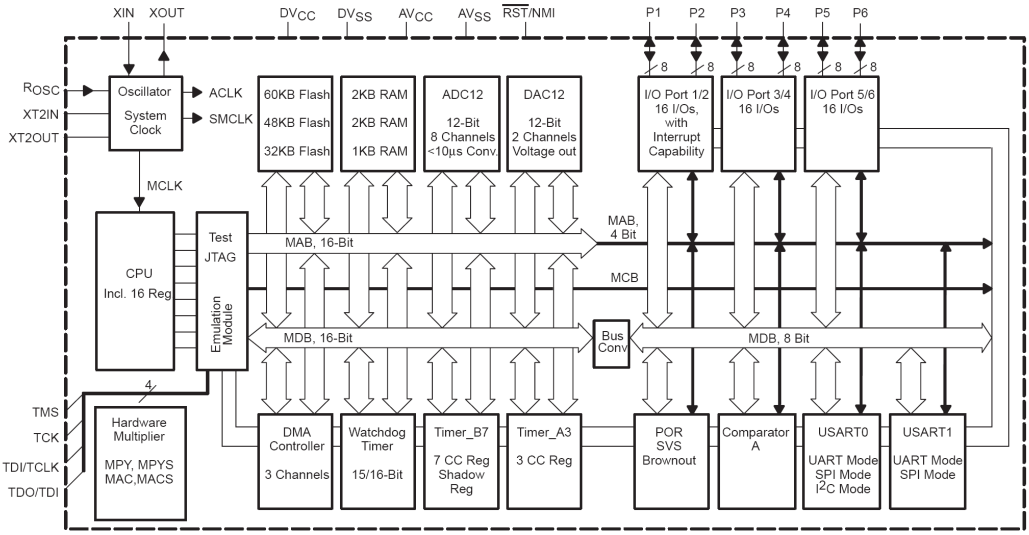
MSP430F167，MSP430F168，MSP430F169



PM PACKAGE
(TOP VIEW)



2.4 结构原理框图



MSP430F15x 和 MSP430F16x 系列单片机特性和结构极为相似，都是在 MSP430F14x 基础上增加了 DMA 控制器、1^2C 模块、DAC 转换模块。MSP430F15x 和 MSP430F16x 的结构差别在于：MSP430F15x 片内只有一个 USART 模块，没有硬件乘法器；而 MSP430F16x 片内具有硬件乘法器和两个 USART 模块。

2.5 管脚定义

引脚 名称	序号	I/O	说明
AVcc	64		模拟电源正端，只为 ADC 和 DAC 的模拟部分供电
AVss	62		模拟电源负端，只为 ADC 和 DAC 的模拟部分供电
DVcc	1		数字电源正端，为所有数字部分供电
DVss	63		数字电源负端，为所有数字部分供电
P1.0/TACLK	12	I/O	通用数字 I/O 引脚；定时器 A 时钟信号 TACLK 输入
P1.1/TA0	13	I/O	通用数字 I/O 引脚；定时器 A 捕获 CCIOA 输入；比较 OUT 输出；BSL 发送
P1.2/TAl	14	I/O	通用数字 I/O 引脚/定时器 A 捕获 CCI1A 输入；比较 OUTl 输出
P1.3/TA2	15	I/O	通用数字 I/O 引脚；定时器 A 捕获 CCl2A 输入；比较 0UT2 输出
P1.4/SMCLK	16	I/O	通用数字 I/O 引脚；SMCLK 信号输出
P1.5/TA0	17	I/O	通用数字 I/O 引脚；定时器 A 比较 OUT0 输出
P1.6/TAl	18	I/O	通用数字 I/O 引脚；定时器 A 比较 OUT1 输出
P1.7/TA2	19	I/O	通用数字 I/O 引脚；比较器 A 比较 OUT2 输出
P2.0/ACLK	20	I/O	通用数字 I/O 引脚；ACLK 输出
P2.1/TAINCLK	2l	I/O	通用数字 I/O 引脚；定时器 A 的 INCLK 的时钟信号

P2.2/CAOUT/TA0	22	I/O	通用数字 I/O 引脚；定时器 A 捕获 CCIOB 输入； 比较器输出
P2.3/CA0/TAl	23	I/O	通用数字 I/O 引脚；定时器 A 比较 OUTl 输出； 比较器 A 输入
P2.4/CA1/TA2	24	I/O	通用数字 I/O 引脚；定时器 A 比较 OUT2 输出； 比较器 A 输入
P2.5/Rosc	25	I/O	通用数字 I/0 引脚；定义 DCO 标称频率的外部电阻输入
P2.6/ADCl2CLK/DMAE0	26	I/O	通用数字 I/O 引脚；转换时钟 ADC12； DMA 通道 0 外部触发器
P2.7/TA0	27	I/O	通用数字 I/O 引脚；定时器 A 比较 OUT0 输出
P3.0/STE0	28	I/O	通用数字 I/O 引脚； USART0/SPI 模式从设备传输使能端
P3.l/SIMO0/DSDA	29	I/O	通用数字 I/O 引脚； USART0/SPI 模式的从入/主出；I^2C 数据
P3.2/SOMIO	30	I/O	通用数字 I/O 引脚；USART0/SPI 模式的从出；主入
P3.3/UCLK0/SCL	31	I/O	通用数字 I/O 引脚； USART0/SPI 模式的外部时钟输入；I^2C 时钟输出
P3.4/UTXD0	32	I/O	通用数字 I/O 引脚；USART0/SPI 模式的传输数据输出
P3.5/URXD0	33	I/O	通用数字 I/O 引脚；USART0/SPI 模式的接收数据输入
P3.6	34	I/O	通用数字 I/O 引脚；USART1/UART 模式接收数据输入
P3.7	35	I/O	通用数字 I/O 引脚；USART1/UART 模式发送数据输出
P4.0/TB0	36	I/O	通用数字 I/O 引脚； 捕获 I/P 或者 PWM 输出端口；定时器 B7CCR0
P4.1/TB1	37	I/O	通用数字 I/O 引脚； 捕获 I/P 或者 PWM 输出端口；定时器 B7CCRl
P4.2/TB2	38	I/O	通用数字 I/O 引脚； 捕获 I/P 或者 PWM 输出端口；定时器 B7CCR2
P4.3/TB3	39	I/O	通用数字 I/O 引脚； 捕获 I/P 或者 PWM 输出端口；定时器 B7CCR3
P4.4/TB4	40	I/O	通用数字 I/O 引脚； 捕获 I/P 或者 PWM 输出端口；定时器 B7CCR4
P4.5/TB5	41	I/O	通用数字 I/O 引脚； 捕获 I/P 或者 PWM 输出端口；定时器 B7CCR5
P4.6/TB6	42	I/O	通用数字 I/O 引脚； 捕获 I/P 或者 PWM 输出端口；定时器 B7CCR6
P4.7/TBCLK	43	I/O	通用数字 I/O 引脚；输入时钟 TBCLK--定时器 B7
P5.0	44	I/O	通用数字 I/O 引脚；USARTl/SPI 模式从设备传输使能端
P5.1	45	I/O	通用数字 I/O 引脚； USARTI/SPI 模式的从输入；主输出
P5.2	46	I/O	通用数字 I/O 引脚；USARTI/SPI 模式的从输出；主输入
P5.3	47	I/O	通用数字 I/O 引脚；USARTl/SPI 模式的外部时钟输入； USART0/SPI 模式的时钟输入
P5.4/MCLK	48	I/O	通用数字 I/O 引脚；主系统时钟输入
P5.5/SMCLK	49	I/O	通用数字 I/O 引脚；子系统时钟输出

P5.6/ACLK	50	I/O	通用数字 I/O 引脚；辅助时钟输出
P5.7/1TBOUTH/SVSOUT	5l	I/O	通用数字 I/O 引脚；定时器 B7；SVS 比较输出 将所有 PWM 数字输出端口为高阻态；
P6.0/A0	59	I/O	通用数字 I/O 引脚；12 位 ADC 模拟输入 A0
P6.1/Al	60	I/O	通用数字 I/O 引脚；12 位 ADC 模拟输入 A1
P6.2/A2	61	I/O	通用数字 I/O 引脚；12 位 ADC 模拟输入 A2
P6.3/A3	2	I/O	通用数字 I/O 引脚；12 位 ADC 模拟输入 A3
P6.4/A4	3	I/O	通用数字 I/O 引脚；12 位 ADC 模拟输入 A4
P6.5/A5	4	I/O	通用数字 I/O 引脚；12 位 ADC 模拟输入 A5
P6.6/A6/DAC0	5	I/O	通用数字 I/O 引脚；12 位 ADC 模拟输入 A5； DAC0 输出
P6.7/A7/DACl	6	I/O	通用数字 I/O 引脚，12 位 ADC 模拟输入 A5， DAC1 输入；SVS 输入
nRST/NMI	58	I	复位输入；非屏蔽中断输入或者 Bootstrap Load 启动 (BSL 方式)
TCK	57	I	测试时钟；TCK 使芯片编程测试和 Bootstrap Loader 启动的时钟输入端口
TDI/TCLK	55	I	测试数据输入； TDI 用作数据输入端口或者测试时钟的输入端口
TDO/TDI	54	I/O	测试数据输出； TDO/TDI 数据输入或者编程数据输出引脚
TMS	56	I	测试模式选择；TMS 用作芯片编程和测试的输入端口
VeREF+	10	I	外部参考电压输入
VREF+	7	O	内部参考电压的正输出引脚
VREF-/VREF+	1l	O	内部参考电压或者外加参考电压的引脚
XIN	8	I	晶振 XT1 的输入端口
XOUT	9	I/O	晶振 XT1 的输出端口
XT2IN	53	I	晶振 XT2 的输入
XT20UT	52	O	晶振 XT2 输出

第三章 BCS 基本时钟系统

Basic Clock System

MSP430F169 单片机的基本时钟系统由高速晶体震荡器，低速晶体震荡器，数字控制震荡器等部件构成。各个震荡器产生的时钟信号可以通过软件的设置分配到 ACLK，MCLK，SMCLK 三路重要的时钟信号通道上。

一般来说，单片机的时钟系统必须满足下列要求

- 高频率，用来系统硬件需求，运算和外部事件的快速响应。
- 低频率，用于降低系统的电流消耗。
- 稳定的频率，以满足定时的需要，例如 RTC 实时时钟。

注意：下面基本时钟系统所提及的寄存器，控制方式等只适用于 MSP430F169 单片机。

3.1 基本时钟系统概述

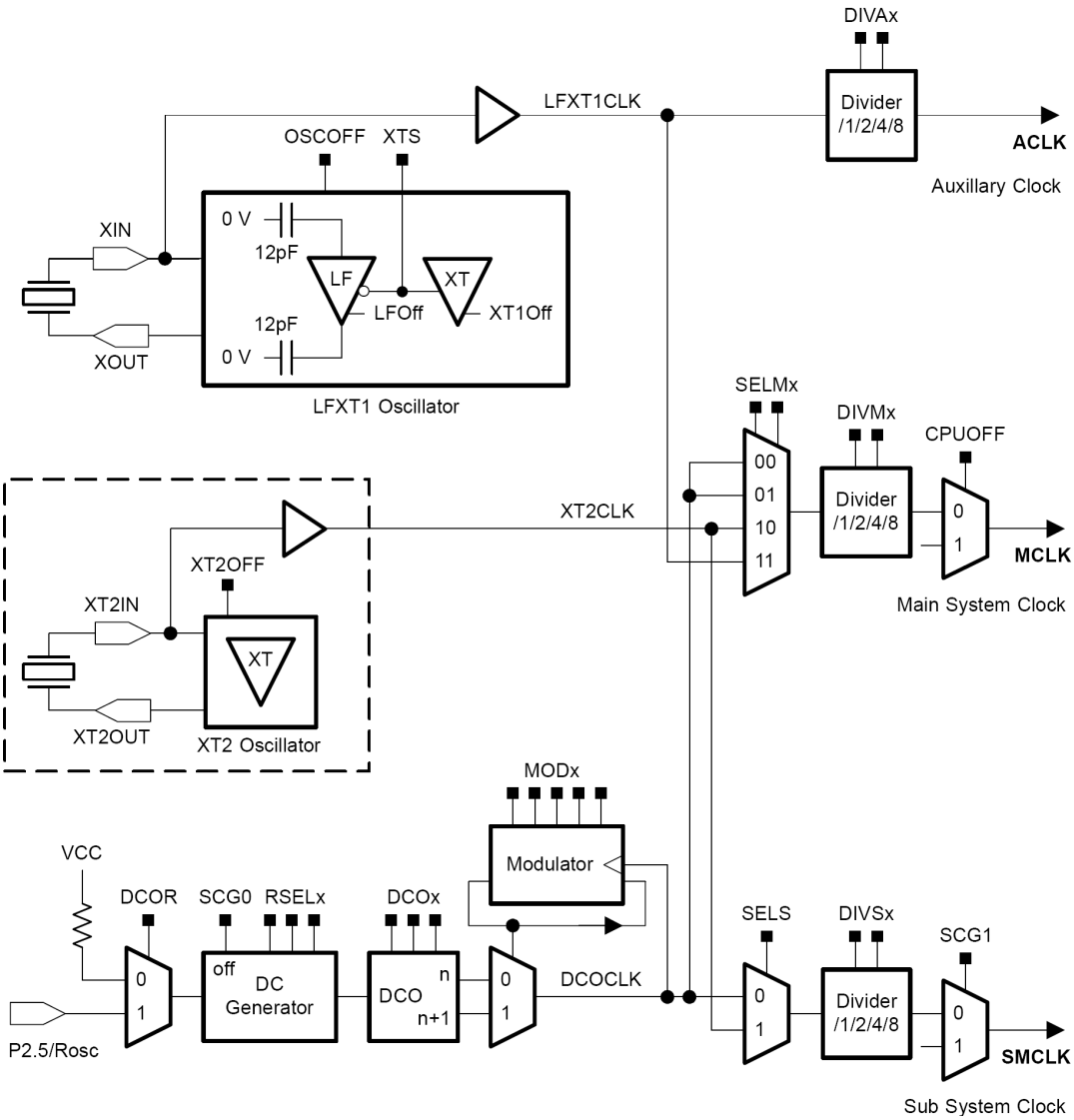
时钟源概述

基本时钟模块包括 3 个时钟输入源：
[1] LFXT1CLK
默认工作在低频模式(32.768kHz)手表晶振
也可以通过外接 450kHz～8MHz 的高速晶体振荡器或谐振器工作在高频模式。
[2] XT2CLK
可选择的高频振荡器，可以通过标准的晶体振荡器、谐振器或外接 450kHz～8MHz 的时钟源工作。
[3] DCOCLK
内部数控 RC 振荡器。

时钟信号概述

通过这些基本的时钟模块，我们可以得到 3 个有用的时钟信号：
[1] ACLK 辅助时钟（Auxillary Clock）
ACLK 是 LFXT1CLK 时钟源经 1、2、4、8 分频后得到的。
ACLK 可由软件选择作为各个外围模块的时钟信号，一般用于低速外设。
[2] MCLK 主系统时钟（Main System Clock）
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。
MCLK 通常用于 CPU 运行，程序的执行和其他使用到高速时钟的模块。
[3] SMCLK 子系统时钟（Sub System Clock）
SMCLK 可由软件选择来自 XT2CLK 或 DCOCLK，然后经 1、2 、4、8 分频。
SMCLK 通常用于高速外围模块。

基本时钟系统结构原理图



3.2 时钟源

[1] 低速晶体震荡器（LFXT1）

手表晶振(32.768kHz)经过 XIN 和 XOUT 引脚直接连接到单片机，不需要其他外部器件（内部有 12pF 的负载电容）。此时 LFXT1 振荡器工作于低频模式（XTS=0）。

如果单片机外接高速晶体振荡器或谐振器时，OSCOFF=0 可使 LFXT1 振荡器工作于高频模式（XTS=1）。此时高速晶体振荡器或谐振器经过 XIN 和 XOUT 引脚连接，并且需要外接电容，电容的大小根据晶体振荡器或谐振器的特性来选择。

如果 LFXT1CLK 信号没有用作 SMCLK 或 MCLK 信号，可用软件将 OSCOFF=1 以禁止 LFXT1 工作以减少单片机耗电。

[2] 高速晶体震荡器

XT2 振荡器产生 XT2CLK 时钟信号，它的工作特性与 LFXT1 振荡器工作在高频模式时类似。如果 XT2CLK 没有用作 MCLK 和 SMCLK 时钟信号，可用控制位 XT2OFF 禁止 XT2 振荡器。

[3] 数控震荡器

单片机的 XT2 振荡器产生的时钟信号可以经过 1、2、4、8 分频后当作系统主时钟 MCLK。当振荡器失效时，DCO 振荡器会被自动选为 MCLK 的时钟源。

DCO 振荡器的频率可由软件对 DCOx、MODx 和 RSELx 位的设置来调整。当 DCOCLK 信号没有用作 SMCLK 和 MCLK 时钟信号时，可以用控制位 SCG0 禁止直流发生器。

在 PUC 信号之后，DCOCLK 被自动选作 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)让 OSCOFF=1

(2)让 OFIFG=0

(3)延时等待至少 50us

(4)再次检查 OFIFG，如果 OFIFG=1，重复(3)、(4)步骤，直到 OFIFG＝0 为止。

3.3 基本时钟系统寄存器

MSP4630F169 单片机的基本时钟系统寄存器

寄存器名称	寄存器缩写
DCO 控制寄存器	DCOCTL
基本时钟系统控制寄存器 1	BCSCTL1
基本时钟系统控制寄存器 2	BCSCTL2

[1] DCOCTL　　DCO 控制寄存器

DCOx：　DCO 频率选择

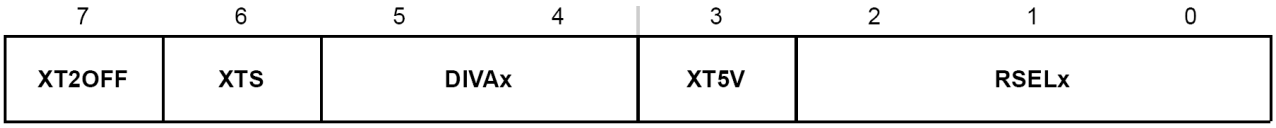
用来选择 8 种频率，可分段进行调节 DCOCLK 频率。该频率是建立在 RSELx 选定的频段上。

MODx：DAC 调制器设定

控制切换 DCOx 和 DCOx+1 选择的两种频率，来微调 DCO 的输出频率。

如果 DCOx 常数是 7，表示已经选择最高频率，此时 MODx 失效，不能用来进行频率调整。

[2] BCSCTL1　　基本时钟系统控制寄存器 1



XT2OFF：XT2 高速晶振控制

此位用于控制 XT2 振荡器的开启与关闭。

0：　XT2 高速晶振开

1：　XT2 高速晶振关

XTS：LFXT1 高速/低速模式选择

0：　LFXT1 工作在低速晶振模式（默认）

1：　LFXT1 工作在高速晶振模式

DIVAx：ACLK 分频选择

0：　不分频

1：　2 分频

2：　4 分频

3：　8 分频

XT5V：不使用

通常此位复位 XT5V=0

RSELx：DCO 震荡器的频段选择

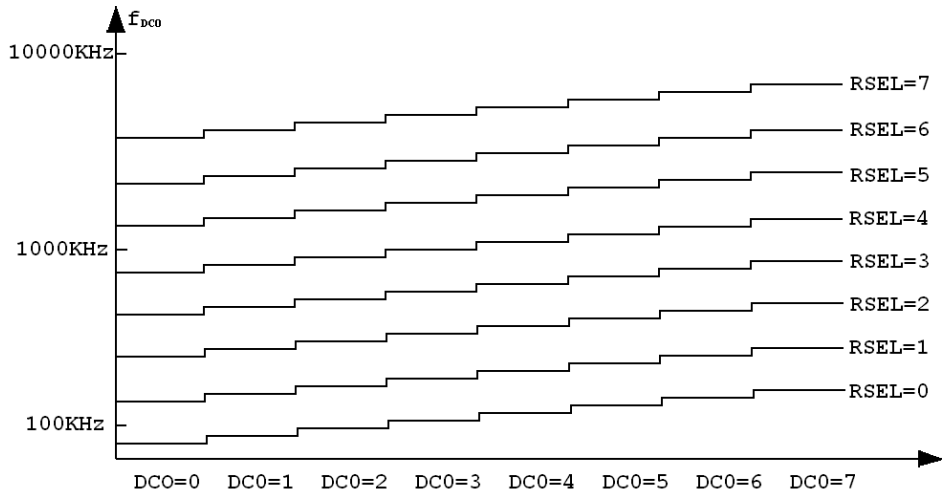
该 3 位控制某个内部电阻以决定标称频率。

0：　选择最低的标称频率

.....

7：　选择最高的标称频率

DCO 的频率的调节图（DCOx 和 RSELx 决定 DCO 频率）



[3] BCSCTL2　　基本时钟系统控制寄存器 2

7	6	5	4	3	2	1	0
SELMx		DIVMx		SELS	DIVSx		DCOR

SELMx：选择 MCLK 时钟源

0：MCLK 时钟源为 DCOCLK（默认）

1：MCLK 时钟源为 DCOCLK

2：MCLK 时钟源为 XT2CLK

3：MCLK 时钟源为 LFXT1CLK

DIVMx：选择 MCLK 分频

0：不分频（默认）

1：2 分频

2：4 分频

3：8 分频

SELS：选择 SMCLK 时钟源

0：SMCLK 时钟源为 DCOCLK（默认）

1：SMCLK 时钟源为 XT2CLK

DIVSx：选择 SMCLK 分频

0：不分频（默认）

1：2 分频

2：4 分频

3：8 分频

DCOR：选择 DCO 震荡电阻

0：内部电阻

1：外部电阻

3.4 基本时钟系统头文件定义

```
/************************************************************
* Basic Clock Module
************************************************************/
#define DCOCTL_           (0x0056)  /* DCOCTL 的定义 */
DEFC(   DCOCTL          , DCOCTL_)
#define BCSCTL1_          (0x0057)  /* BCSCTL1 的定义 */
DEFC(   BCSCTL1         , BCSCTL1_)
#define BCSCTL2_          (0x0058)  /* BCSCTL2 的定义 */
DEFC(   BCSCTL2         , BCSCTL2_)
/* DCOCTL 的位定义 */
#define MOD0              (0x01)   /* Modulation Bit 0 */
#define MOD1              (0x02)   /* Modulation Bit 1 */
#define MOD2              (0x04)   /* Modulation Bit 2 */
#define MOD3              (0x08)   /* Modulation Bit 3 */
#define MOD4              (0x10)   /* Modulation Bit 4 */
#define DCO0              (0x20)   /* DCO Select Bit 0 */
#define DCO1              (0x40)   /* DCO Select Bit 1 */
#define DCO2              (0x80)   /* DCO Select Bit 2 */
/* BCSCTL1 的位定义 */
#define RSEL0             (0x01)   /* Resistor Select Bit 0 */
#define RSEL1             (0x02)   /* Resistor Select Bit 1 */
#define RSEL2             (0x04)   /* Resistor Select Bit 2 */
#define XT5V              (0x08)   /* XT5V should always be reset */
#define DIVA0             (0x10)   /* ACLK Divider 0 */
#define DIVA1             (0x20)   /* ACLK Divider 1 */
#define XTS               (0x40)   /* LFXTCLK 0:Low Freq. / 1: High Freq. */
#define XT2OFF            (0x80)   /* Enable XT2CLK */
/* BCSCTL1 的 DIVA 的功能定义 */
#define DIVA_0            (0x00)   /* ACLK Divider 0: /1 */
#define DIVA_1            (0x10)   /* ACLK Divider 1: /2 */
#define DIVA_2            (0x20)   /* ACLK Divider 2: /4 */
#define DIVA_3            (0x30)   /* ACLK Divider 3: /8 */
/* BCSCTL2 的位定义 */
#define DCOR              (0x01)   /* Enable External Resistor : 1 */
#define DIVS0             (0x02)   /* SMCLK Divider 0 */
#define DIVS1             (0x04)   /* SMCLK Divider 1 */
#define SELS              (0x08)   /* SMCLK Source Select 0:DCOCLK / 1:XT2CLK/LFXTCLK */
#define DIVM0             (0x10)   /* MCLK Divider 0 */
#define DIVM1             (0x20)   /* MCLK Divider 1 */
#define SELM0             (0x40)   /* MCLK Source Select 0 */
#define SELM1             (0x80)   /* MCLK Source Select 1 */
/* BCSCTL1 的 DIVS 的功能定义 */
#define DIVS_0            (0x00)   /* SMCLK Divider 0: /1 */
#define DIVS_1            (0x02)   /* SMCLK Divider 1: /2 */
#define DIVS_2            (0x04)   /* SMCLK Divider 2: /4 */
#define DIVS_3            (0x06)   /* SMCLK Divider 3: /8 */
/* BCSCTL1 的 DIVM 的功能定义 */
#define DIVM_0            (0x00)   /* MCLK Divider 0: /1 */
#define DIVM_1            (0x10)   /* MCLK Divider 1: /2 */
#define DIVM_2            (0x20)   /* MCLK Divider 2: /4 */
#define DIVM_3            (0x30)   /* MCLK Divider 3: /8 */
/* BCSCTL1 的 SELM 的功能定义 */
#define SELM_0            (0x00)   /* MCLK Source Select 0: DCOCLK */
#define SELM_1            (0x40)   /* MCLK Source Select 1: DCOCLK */
#define SELM_2            (0x80)   /* MCLK Source Select 2: XT2CLK/LFXTCLK */
#define SELM_3            (0xC0)   /* MCLK Source Select 3: LFXTCLK */
```

3.5 基本时钟系统例程

```
// 设 ACLK=MCLK=LFXT1=HF，将 MCLK 通过 P5.4 输出
#include<msp430x16x.h>
void main(void)
{
  unsigned int i;
  WDCTL = WDTPW + WDTHOLD;          // 停看门狗
  P5DIR |= 0x10;                    // P5.4 输出
  P5SEL |= 0x10;                    // P5.4 = MCLK
  BCSCTL1 |= XTS;                   // ACLK = LFXT1 = HF 模式
  do
  {
    IFG1 &= ~OFIFG;                 // 清除振荡器失效标志
    for(i = 0Xff;i > 0;i--);        // 稳定时间
  }
  while((IFG1 & OFIFG) != 0);       // 如果振荡器失效标志存在
  BCSCTL2 |= SELM1 + SELM0;         // MCLK = LFXT1
  While(1){_NOP();}
}


// 时钟设置函数
// 系统时钟设定
// DCO 设置为 3030KHz
// ACLK 为 LFXT1(低频模式)
// MCLK 为 XT2CLK
// SMLCK 为 XT2CLK
void BCSInit (void)
{
  DCOCTL  = 0x60 + 0x00;
  BCSCTL1 = DIVA_0 + 0x07;
  BCSCTL2 = SELM_2 + DIVM_0 + SELS + DIVS_0;
}
```

第四章 LPM 超低功耗模块

Low Power Module

MSP430 系列单片机是一个特别强调低功耗的单片机系列，尤其适用于采用电池长时间供电的工作场合。

MSP430 应用系统价格和电流消耗等因素会影响 CPU 与外围模块对时钟的需求，所以系统所使用不同的时钟信号：ACLK，MCLK，SMCLK。用户可以通过程序可以选择高频和低频，这样可以根据实际需要来选择适合的系统时钟频率，从而合理的利用系统的电源，实现整个系统的超低功耗。

MSP430 系统提供丰富的软硬件组合形式，能够达到最低功耗并发挥最优的系统性能。

4.1 低功耗模块简介

低功耗是 MSP430 单片机一个最显著的特点。MSP430 的丰富的时钟源使其能达到最低功耗并发挥最优系统性能。用户可通过实际需要选择不同的时钟源（ACLK、MCLK 和 SMCLK），从而实现整个系统的超低功耗。

4.2 低功耗控制

当系统时钟发生器基本功能建立之后，状态寄存器 SR 中的 SCG1、SCG0、OscOff 和 CPUOff 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，中断处理后，又可恢复先前工作方式。

在中断处理子程序中可以间接访问堆栈数据从而修改这些控制位；在中断返回后单片机会以另一种功耗方式继续运行。各控制位的功能如下：

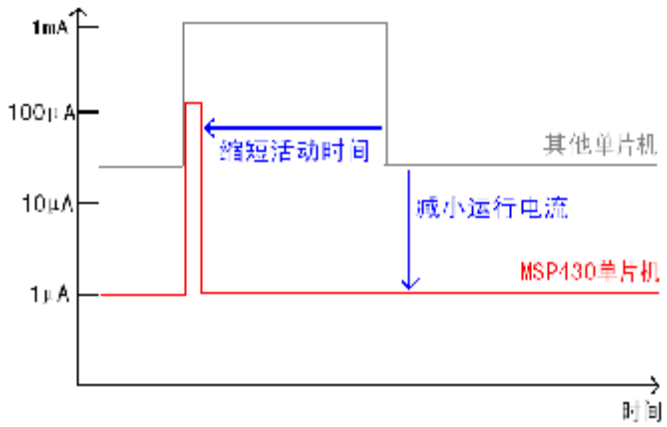
SCG1： 当 SCG1 复位时，使能 SMCLK；
当 SCG1 置位时则禁止 SMCLK。

SCG0： 当 SCG0 复位时，直流发生器被激活；
只有当 SCG0 置位且 DCOCLK 信号未用于 MCLK 或 SMCLK，直流发生器才被禁止。
（编者按：直流发生器为 BCS 中的 DC Generator，也有翻译成数控发生器）

注意：当电流关闭时（SCG=0），DCO 的下次启动会有一些微秒级的延迟。

OscOff：当 OscOff 复位时，LFXT 晶体振荡器被激活；
当 OscOff 被置位且不用于 MCLK 或 SMCLK，LFXT 晶体振荡器才被禁止。

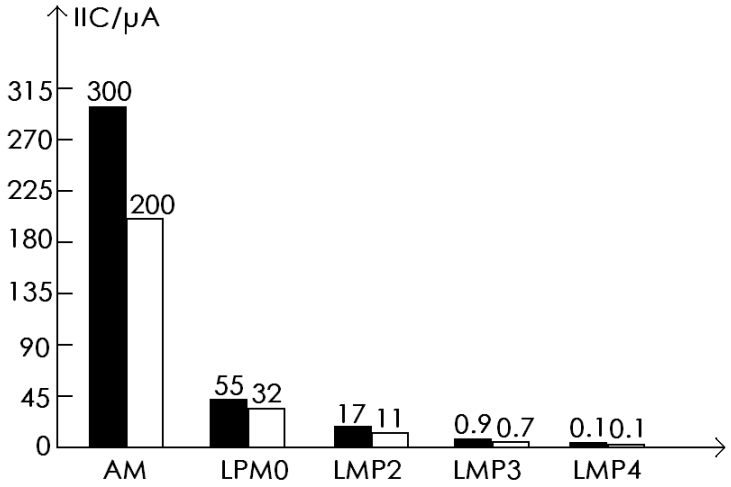
CPUOff：当 CPUOff 复位时，用于 CPU 的时钟信号 MCLK 被激活；
当 CPUOff 置为，MCLK 停止。



MSP430 的 LPM 瞬时响应

4.3 低功耗工作模式

MSP430 可工作在一种活动模式（AM）和五种低功耗模式(LPM0~LPM4)下。通过软件设置控制位 SCG1、SCG0、OscOff 和 CPUOff，MSP430 可进入相应的低功耗模式。各种低功耗模式又可通过中断方式返回活动模式。不同的工作模式耗电情况不同，具体如下表所示：



由上图可以看出，由 AM LPM4 模式单片机工作电流成倍下降。担不同工作模式对 CPU 状态、振荡器及时钟的活动状态不同。具体如下表所示。

工作模式	控制位	CPU 状态、振荡器及时钟
活动模式 （AM）	SCG1=0 SCG0=0 OscOff=0 CPUOff=0	CPU 处于活动状态 MCLK 活动 SMCLK 活动 ACLK 活动
低功耗模式 0 （LPM0）	SCG1=0 SCG0=0 OscOff=0 CPUOff=1	CPU 处于禁止状态 MCLK 禁止 SMCLK 活动 ACLK 活动
低功耗模式 1 （LPM1）	SCG1=0 SCG0=1 OscOff=0 CPUOff=1	CPU 处于禁止状态 若 DCO 未用作 SMCLK 或 MCLK，则自流发生器禁止，否则任保持活动。 MCLK 禁止；SMCLK 活动；ACLK 活动
低功耗模式 2 （LPM2）	SCG1=1 SCG0=0 OscOff=0 CPUOff=1	CPU 处于禁止状态 若 DCO 未用作 SMCLK 或 MCLK，则 DCO 自动被禁止。 MCLK 禁止；SMCLK 禁止；ACLK 活动
低功耗模式 3 （LPM3）	SCG1=1 SCG0=1 OscOff=0 CPUOff=1	CPU 处于禁止状态 DCO 被禁止；自流发生器被禁止。 MCLK 禁止；SMCLK 禁止 ACLK 活动

低功耗模式 4 （LPM4）	SCG1= x SCG0= x OscOff=1 CPUOff=1	CPU 处于禁止状态 DCO 被禁止；自流发生器被禁止。 所有振荡器停止工作。 MCLK 禁止；SMCLK 禁止；ACLK 活动

4.4 低功耗应用原则

- 最大化 LMP3 的时间，用 32.768KHZ 晶振作为 ACLK 时钟，DCO 用于 CPU 激活后突发短暂运行。
- 用接口模块代替软件驱动功能。
- 用中断控制程序运行。
- 用可计算的分支代替标志位测试产生的分支。
- 用快速查表代替冗长的软件计算。
- 避免频繁的子程序和函数调用。
- 在必要时才开启外围模块。

4.5 低功耗模块头文件定义

在 msp430x16x. h 对低功耗宏定义如下所示：

```
#ifndef __IAR_SYSTEMS_ICC /* Begin #defines for assembler */
#define LPM0             (CPUOFF)
#define LPM1             (SCG0+CPUOFF)
#define LPM2             (SCG1+CPUOFF)
#define LPM3             (SCG1+SCG0+CPUOFF)
#define LPM4             (SCG1+SCG0+OSCOFF+CPUOFF)
/* End #defines for assembler */
```

注意：以上宏定义是对汇编的定义

在 intrinsics.h 对低功耗宏定义如下所示：

```
/*Functions for controlling the processor operation modes.*/

#define __low_power_mode_0() (__bis_SR_register(  __SR_GIE     \
                                        | __SR_CPU_OFF))
#define __low_power_mode_1() (__bis_SR_register(  __SR_GIE     \
                                        | __SR_CPU_OFF  \
                                        | __SR_SCG0))
#define __low_power_mode_2() (__bis_SR_register(  __SR_GIE     \
                                        | __SR_CPU_OFF  \
                                        | __SR_SCG1))
#define __low_power_mode_3()                 \
  (__bis_SR_register(  __SR_GIE              \
                     | __SR_CPU_OFF          \
                     | __SR_SCG0             \
                     | __SR_SCG1))
#define __low_power_mode_4()                 \
  (__bis_SR_register(  __SR_GIE              \
                     | __SR_CPU_OFF          \
                     | __SR_SCG0             \
                     | __SR_SCG1             \
                     | __SR_OSC_OFF))
#define __low_power_mode_off_on_exit()         \
  (__bic_SR_register_on_exit(  __SR_CPU_OFF    \
                         | __SR_SCG0       \
                         | __SR_SCG1       \
                         | __SR_OSC_OFF))
```

注意：上述是对 C 语言的定义

进入 LPM0 模式，可以通过调用__low_power_mode_0();函数来完成。
进入 LPM1 模式，可以通过调用__low_power_mode_1();函数来完成。
进入 LPM2 模式，可以通过调用__low_power_mode_2();函数来完成。
进入 LPM3 模式，可以通过调用__low_power_mode_3();函数来完成。
进入 LPM4 模式，可以通过调用__low_power_mode_4();函数来完成。
退出 LPM 模式，则调用__low_power_mode_off_on_exit();来实现。

4.6 低功耗模块例程

```
#include <msp430x16x.h>

// P1 中断服务子程序
#pragma vector = PORT1_VECTOR
__interrupt void P1_IRQ(void)
{
  P1IFG = 0;                          // 清除中断标志
  __low_power_mode_off_on_exit();     // 退出低功耗模式
}

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;
  P1DIR &= BIT0;                      // P1.0 输入
  P1IE  |= BIT0;                      // P1.0 开中断
  P1IES |= BIT0;                      // P1.0 下降沿中断
  _EINT();

  while(1)
  {
    _NOP();                           // 空语句
    __low_power_mode_0();             // 进入 LPM0 模式
    // 当程序执行到上述语句后, 程序会停止到该语句, 并进入低功耗状态
    // 当 P1.0 出现下降沿的时候, 在 P1 的中断服务子程序退出低功耗模式
    // 则会再次运行_NOP();语句, 然后在进入底功耗
  }
}
```

第五章 I/O 端口

Input / Output Port

端口是 MSP430 及其重要的资源。MSP430F169 单片机共有六组端口，每组端口均为 8 位并且均可独立编程，但每组端口均不提供位操作指令。目前 MSP430 所有系列总线均不对外开放，端口不但用于输入/输出，还可为 MSP430 系统扩展等应用提供必要的逻辑控制信号

5.1 I/O 端口概述

MSP430F169 端口的主要特点为：

n 类型丰富：MSP430F169 有 P1、P2、P3、P4、P5、P6 共六组独立的端口。

n 功能丰富：MSP430F169 各组端口功能如下表所示。

端口	功能
P1、P2	I/O、中断能力、其它片内外设功能
P2、P3、P4、P6	I/O、其它片内外设功能

（编者按：其他类型的单片机还有 S，COM 端口，是用做段码液晶驱动的，F196 没有）

n 丰富的寄存器

MSP430 各种端口有大量的控制寄存器供用户使用，从而提高了端口的输入/输出灵活性。其中 P1 和 P2 有 7 个寄存器，P3～P6 各有 4 个寄存器。通过设置寄存器，可实现以下功能：

① 每个 I/O 都可独立编程

② 允许任意组合输入、输出和中断

③ P1 和 P2 所有 8 位全都可做外部中断处理

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

5.2 端口控制寄存器

端口 P1 具有输入/输出、中断和外部模块功能，这些功能可通过 7 个控制寄存器的设置来实现。下面介绍各控制寄存器特点及其使用：

[1] PxDIR　输入/输出方向寄存器

7	6	5	4	3	2	1	0
PxDIR.7	PxDIR.6	PxDIR.5	PxDIR.4	PxDIR.3	PxDIR.2	PxDIR.1	PxDIR.0

相互独立的 8 位分别定义了 Px 口的 8 位的输入输出方向。在 PUC 复位后 PIDIR 各位均复位。

使用输入/输出功能时，应先定义端口方向。作为输入时，只能读；作为输出时，可读可写。

PxDIR.x：端口输入输出方向控制

0： 输入模式

1： 输出模式

操作：

```
P1DIR  |=0x10;       // P1.4 作输出，其余各位端口方向不变。
P1DIR  &=0x7f;       // P1.7 作输入，其余各位端口方向不变。
```

[2] PxIN　　输入寄存器

7	6	5	4	3	2	1	0
PxIN.7	PxIN.6	PxIN.5	PxIN.4	PxIN.3	PxIN.2	PxIN.1	PxIN.0

该寄存器是只读寄存器。只能通过读取该寄存器内容才能知道 Px 口的输入信号的状态。
读出此寄存器的内容中，只有 Px 口设为输入的数据位有效。
对于 Px 口设为输出的那些位，一般来说，PxIN.x = PxOUT.x
PxIN.x：端口输入的电平
　　0：　端口输入低电平
　　1：　端口输入高电平
操作：

```
    unsigned char Temp;
    P1DIR  &=0x77 ;      // P1.3 和 P1.7 输入
     Temp = P1IN;        // Temp 为在已定义的一变量，Temp 中只要第 7 位和第四位有效。
```

[3] PxOUT　　输出寄存器

7	6	5	4	3	2	1	0
PxOUT.7	PxOUT.6	PxOUT.5	PxOUT.4	PxOUT.3	PxOUT.2	PxOUT.1	PxOUT.0

该寄存器可读可写，读取时，其内容与 Px 口引脚定义无关。改变方向寄存器的内容，此寄存器内容不受影响。
PxOUT.x：端口输出的电平
　　0：　端口输出低电平
　　1：　端口输出高电平
注意：
P1OUT.0 = 1（P1.0 输出高），但是 P1DIR.0 = 0（该引脚为输入模式），则此时 P1.0 为输入；
如果将 P1DIR.0 = 1（该引脚为输出模式），则此时 P1.0 为输出，并且输出为高电平。

操作：

```
    P1DIR  |=0x88;  // P1.3 和 P1.7 输出
    P1OUT  |=0x88;  // P1.3 和 P1.7 输出高电平
```

[4] PxSEL　引脚功能选择寄存器

7	6	5	4	3	2	1	0
PxSEL.7	PxSEL.6	PxSEL.5	PxSEL.4	PxSEL.3	PxSEL.2	PxSEL.1	PxSEL.0

该寄存器可读可写，如果有该引脚具有特殊功能的话，则可以通过该寄存器使用特殊功能。

PxSEL.x：引脚功能选择

0：　该引脚的普通 O/I 端口

1：　该引脚的功能端口

[5] PxIFG　中断标志寄存器

7	6	5	4	3	2	1	0
PxIFG.7	PxIFG.6	PxIFG.5	PxIFG.4	PxIFG.3	PxIFG.2	PxIFG.1	PxIFG.0

该寄存器只有 P1 和 P2 口才有，该寄存器有 8 个标志位，标志相应引脚是否有中断请求。

PxIFG.x：中断标志

0：　该引脚无中断请求

1：　该引脚有中断请求

[6] PxIE　中断允许寄存器

7	6	5	4	3	2	1	0
PxIE.7	PxIE.6	PxIE.5	PxIE.4	PxIE.3	PxIE.2	PxIE.1	PxIE.0

该寄存器只有 P1 和 P2 口才有，该寄存器有 8 个标志位，标志相应引脚是否能响应中断请求。

PxIFG.x：中断允许标志

0：　该引脚中断禁止

1：　该引脚中断允许

[7]　PxIES　中断触发沿控制寄存器

7	6	5	4	3	2	1	0
PxIES.7	PxIES.6	PxIES.5	PxIES.4	PxIES.3	PxIES.2	PxIES.1	PxIES.0

该寄存器只有 P1 和 P2 口才有，该寄存器有 8 个标志位，标志相应引脚的中断触发沿。

PxIFG.x：中断触发沿选择

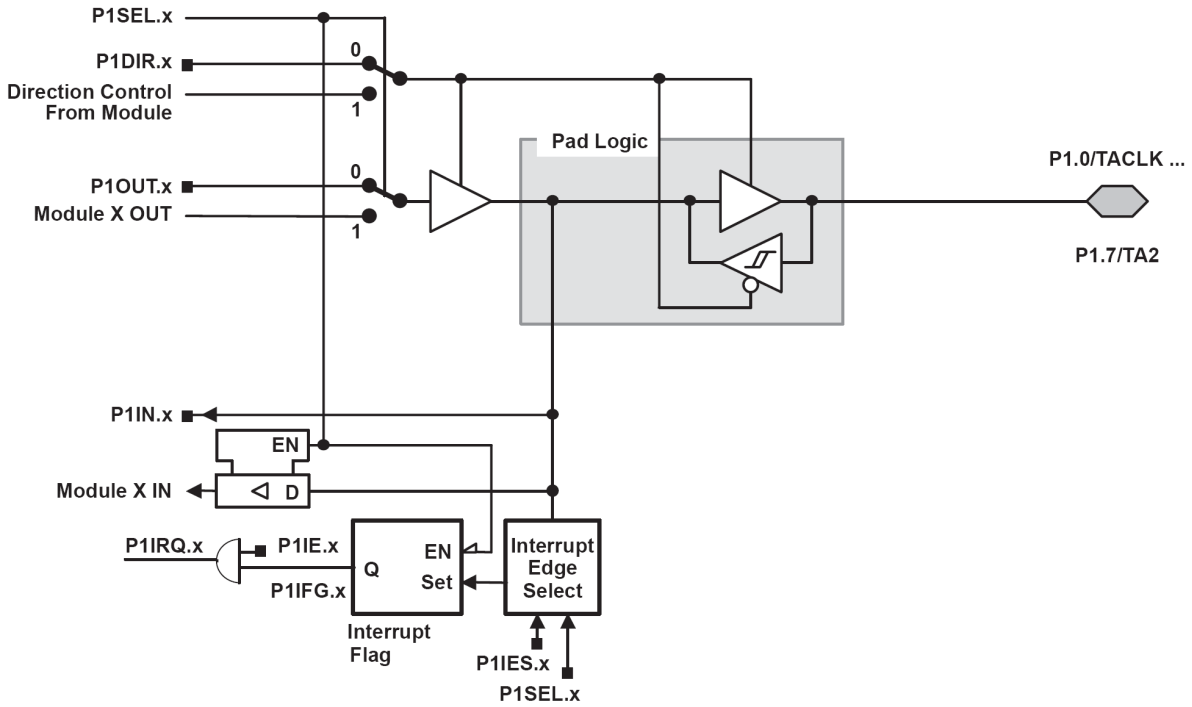
0：　上升沿产生中断

1：　下降沿产生中断

5.3 端口原理图

P1 端口的引脚原理图（P1.0 ~ P1.7）

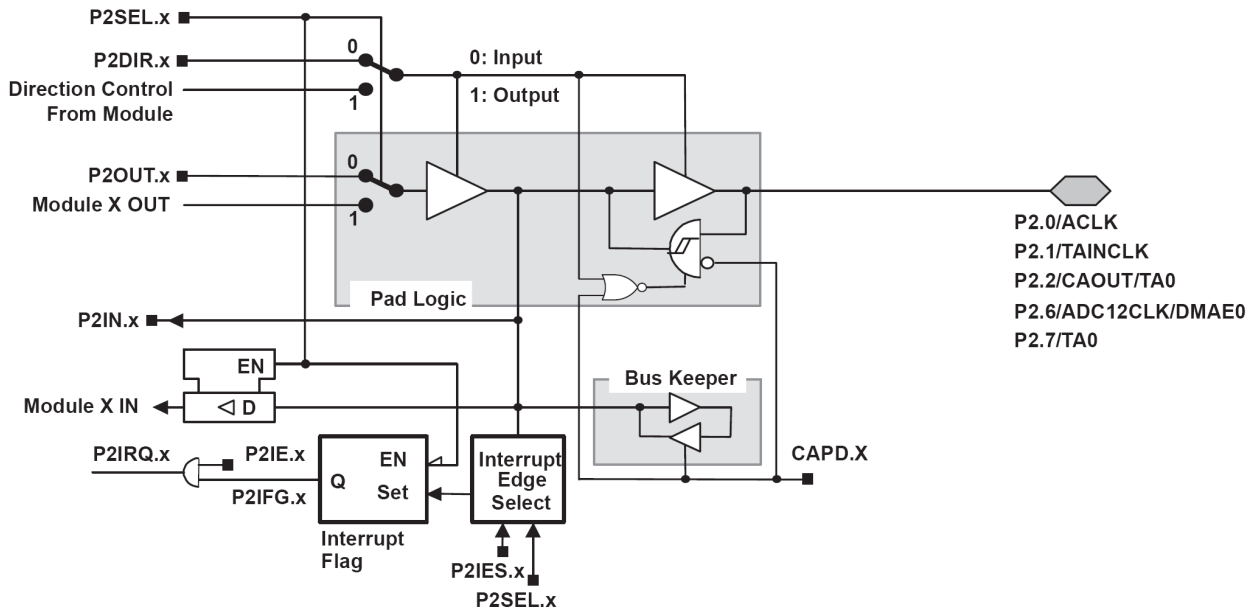
port P1, P1.0 to P1.7, input/output with Schmitt-trigger



PnSel.x	PnDIR.x	Dir. CONTROL FROM MODULE	PnOUT.x	MODULE X OUT	PnIN.x	MODULE X IN	PnIE.x	PnIFG.x	PnIES.x
P1Sel.0	P1DIR.0	P1DIR.0	P1OUT.0	DV_{SS}	P1IN.0	TACLK†	P1IE.0	P1IFG.0	P1IES.0
P1Sel.1	P1DIR.1	P1DIR.1	P1OUT.1	Out0 signal†	P1IN.1	CCI0A†	P1IE.1	P1IFG.1	P1IES.1
P1Sel.2	P1DIR.2	P1DIR.2	P1OUT.2	Out1 signal†	P1IN.2	CCI1A†	P1IE.2	P1IFG.2	P1IES.2
P1Sel.3	P1DIR.3	P1DIR.3	P1OUT.3	Out2 signal†	P1IN.3	CCI2A†	P1IE.3	P1IFG.3	P1IES.3
P1Sel.4	P1DIR.4	P1DIR.4	P1OUT.4	SMCLK	P1IN.4	unused	P1IE.4	P1IFG.4	P1IES.4
P1Sel.5	P1DIR.5	P1DIR.5	P1OUT.5	Out0 signal†	P1IN.5	unused	P1IE.5	P1IFG.5	P1IES.5
P1Sel.6	P1DIR.6	P1DIR.6	P1OUT.6	Out1 signal†	P1IN.6	unused	P1IE.6	P1IFG.6	P1IES.6
P1Sel.7	P1DIR.7	P1DIR.7	P1OUT.7	Out2 signal†	P1IN.7	unused	P1IE.7	P1IFG.7	P1IES.7

P2 端口的引脚原理图（P2.0~2.2，P2.6~P2.7）

port P2, P2.0 to P2.2, P2.6, and P2.7 input/output with Schmitt-trigger



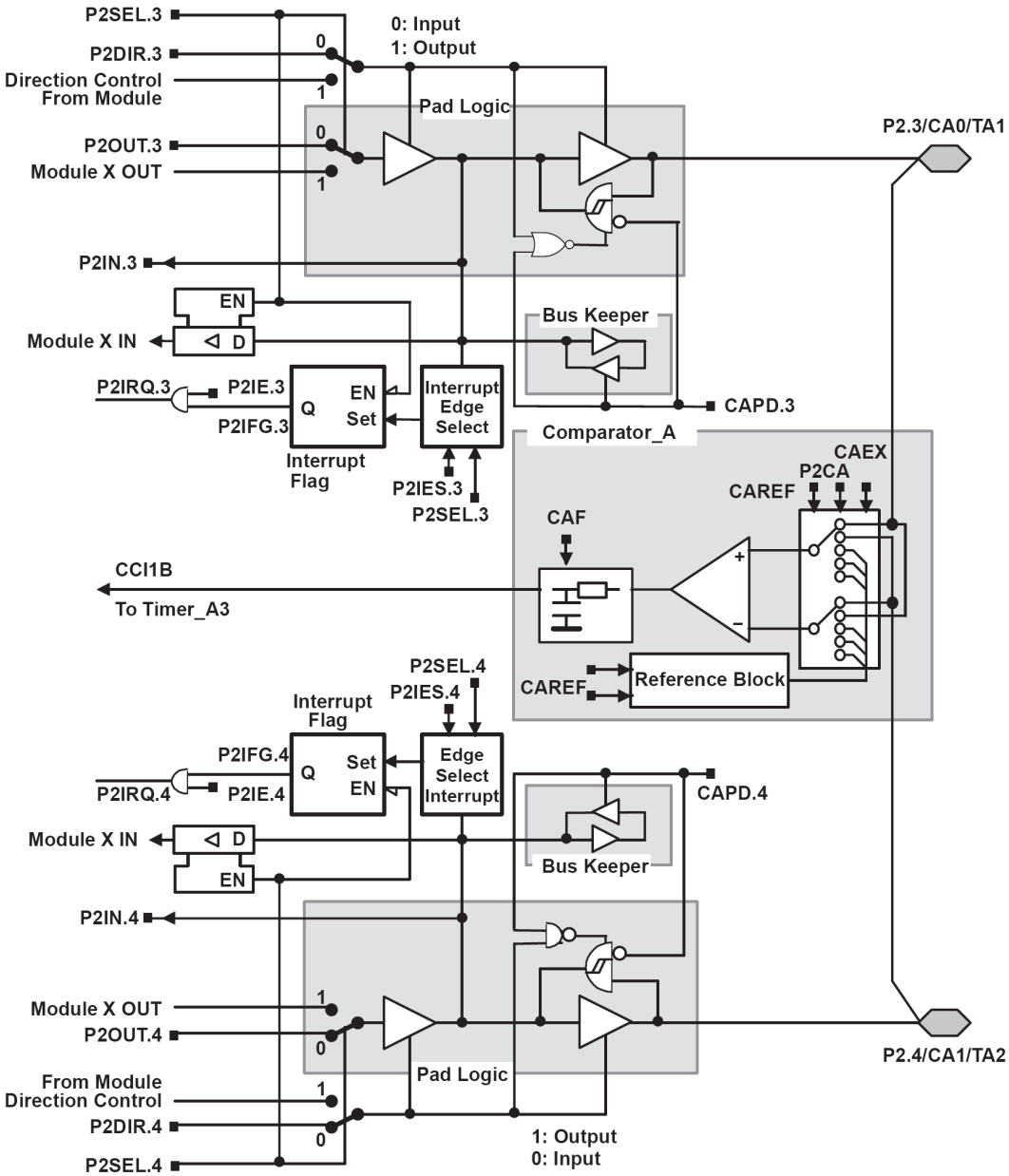
x: Bit Identifier 0 to 2, 6, and 7 for Port P2

PnSel.x	PnDIR.x	Dir. CONTROL FROM MODULE	PnOUT.x	MODULE X OUT	PnIN.x	MODULE X IN	PnIE.x	PnIFG.x	PnIES.x
P2Sel.0	P2DIR.0	P2DIR.0	P2OUT.0	ACLK	P2IN.0	unused	P2IE.0	P2IFG.0	P2IES.0
P2Sel.1	P2DIR.1	P2DIR.1	P2OUT.1	DV_{SS}	P2IN.1	INCLK‡	P2IE.1	P2IFG.1	P2IES.1
P2Sel.2	P2DIR.2	P2DIR.2	P2OUT.2	CAOUT†	P2IN.2	CCI0B‡	P2IE.2	P2IFG.2	P2IES.2
P2Sel.6	P2DIR.6	P2DIR.6	P2OUT.6	ADC12CLK¶	P2IN.6	DMAE0#	P2IE.6	P2IFG.6	P2IES.6
P2Sel.7	P2DIR.7	P2DIR.7	P2OUT.7	Out0 signal§	P2IN.7	unused	P2IE.7	P2IFG.7	P2IES.7

† Signal from Comparator_A
‡ Signal to Timer_A
§ Signal from Timer_A
¶ ADC12CLK signal is output of the 12-bit ADC module
Signal to DMA, channel 0, 1 and 2

P2 端口的引脚原理图（P2.3，P2.4）

port P2, P2.3 to P2.4, input/output with Schmitt-trigger

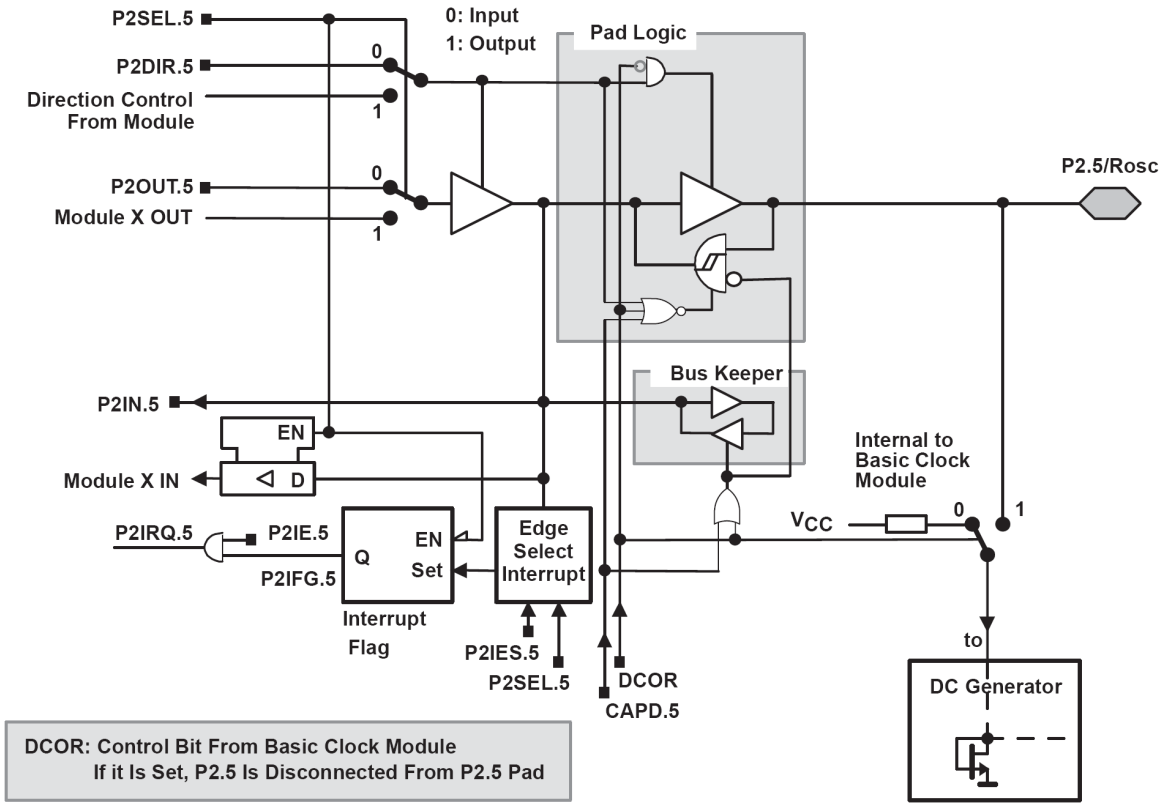


PnSel.x	PnDIR.x	DIRECTION CONTROL FROM MODULE	PnOUT.x	MODULE X OUT	PnIN.x	MODULE X IN	PnIE.x	PnIFG.x	PnIES.x
P2Sel.3	P2DIR.3	P2DIR.3	P2OUT.3	Out1 signal†	P2IN.3	unused	P2IE.3	P2IFG.3	P2IES.3
P2Sel.4	P2DIR.4	P2DIR.4	P2OUT.4	Out2 signal†	P2IN.4	unused	P2IE.4	P2IFG.4	P2IES.4

† Signal from Timer_A

P2 端口的引脚原理图（P2.5）

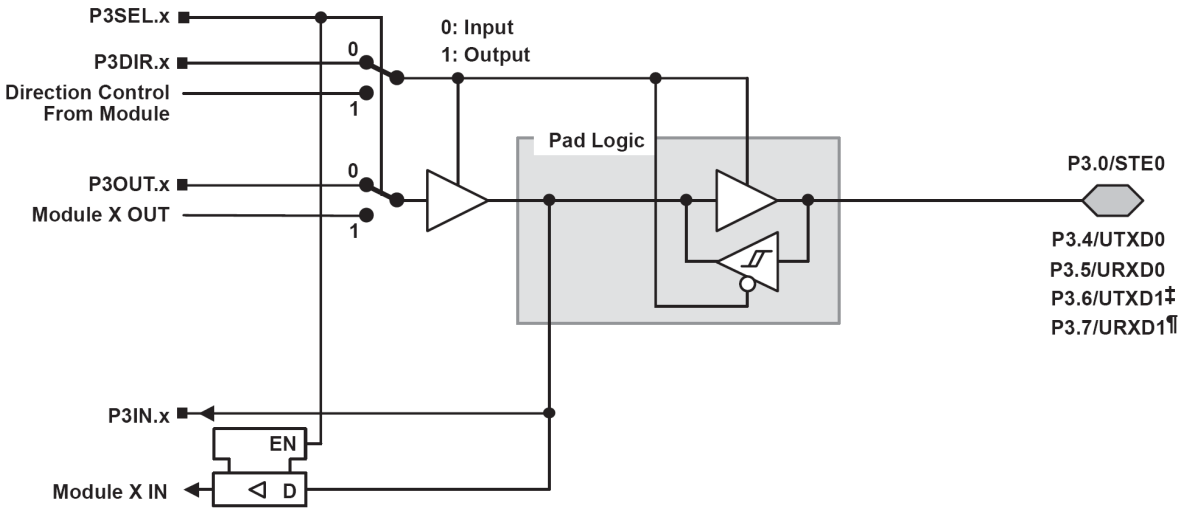
port P2, P2.5, input/output with Schmitt-trigger and R_{osc} function for the basic clock module



PnSel.x	PnDIR.x	DIRECTION CONTROL FROM MODULE	PnOUT.x	MODULE X OUT	PnIN.x	MODULE X IN	PnIE.x	PnIFG.x	PnIES.x
P2Sel.5	P2DIR.5	P2DIR.5	P2OUT.5	DV_{SS}	P2IN.5	unused	P2IE.5	P2IFG.5	P2IES.5

P3 端口的引脚原理图（P3.0，P3.4~P3.7）

port P3, P3.0 and P3.4 to P3.7, input/output with Schmitt-trigger



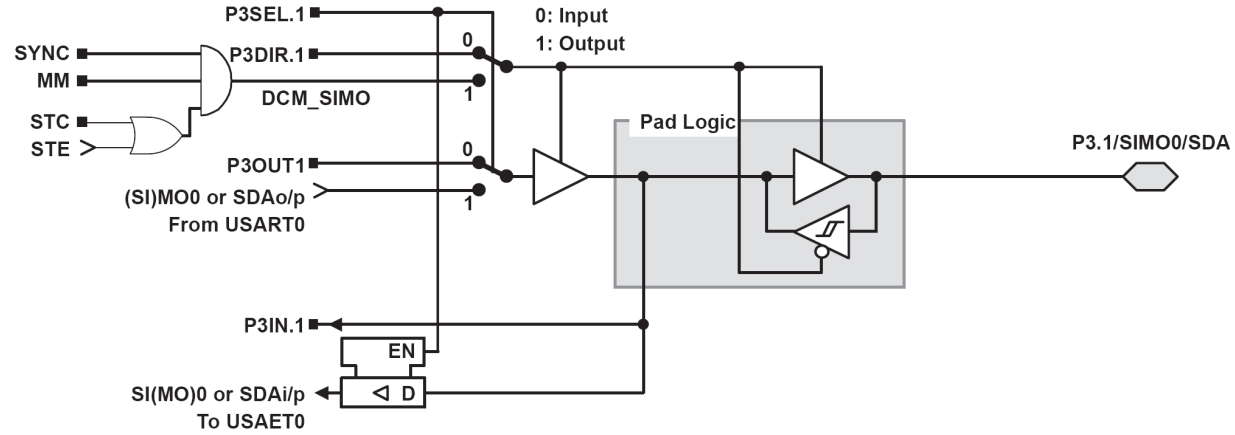
x: Bit Identifier, 0 and 4 to 7 for Port P3

PnSel.x	PnDIR.x	DIRECTION CONTROL FROM MODULE	PnOUT.x	MODULE X OUT	PnIN.x	MODULE X IN
P3Sel.0	P3DIR.0	DV_{SS}	P3OUT.0	DV_{SS}	P3IN.0	STE0
P3Sel.4	P3DIR.4	DV_{CC}	P3OUT.4	UTXD0†	P3IN.4	Unused
P3Sel.5	P3DIR.5	DV_{SS}	P3OUT.5	DV_{SS}	P3IN.5	URXD0§
P3Sel.6	P3DIR.6	DV_{CC}	P3OUT.6	UTXD1‡	P3IN.6	Unused
P3Sel.7	P3DIR.7	DV_{SS}	P3OUT.7	DV_{SS}	P3IN.7	URXD1¶

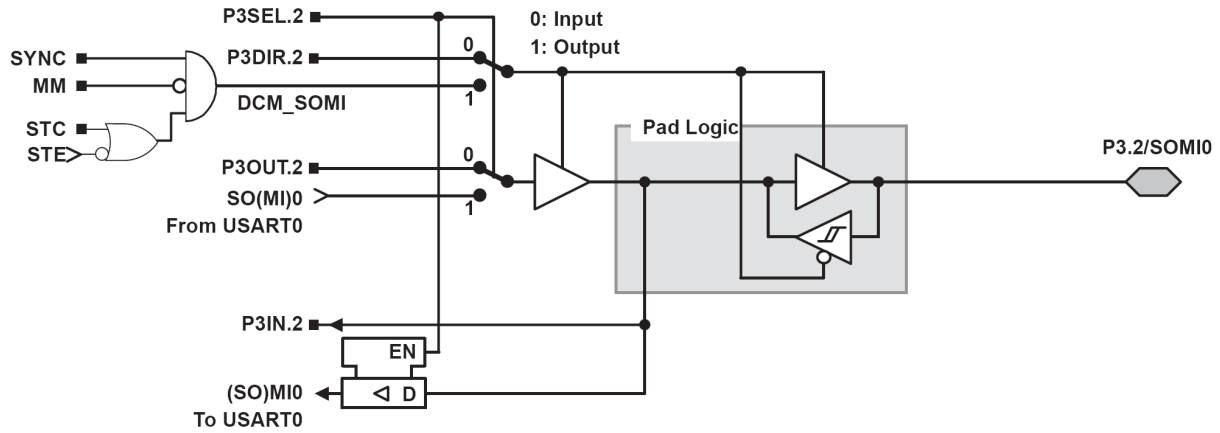
† Output from USART0 module
‡ Output from USART1 module
‡ Input to USART0 module
¶ Input to USART1 module

P3 端口的引脚原理图（P3.1，P3.2，P3.3）

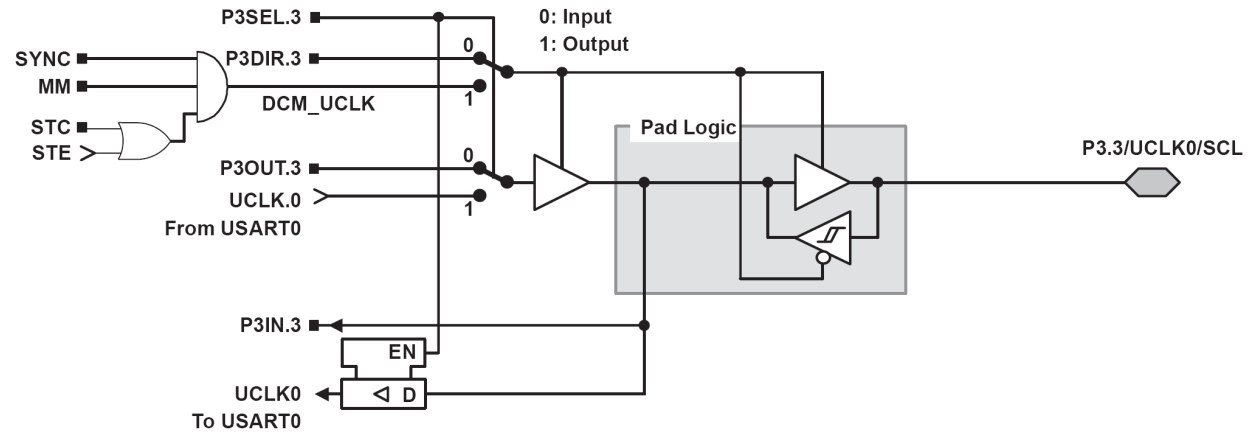
port P3, P3.1, input/output with Schmitt-trigger



port P3, P3.2, input/output with Schmitt-trigger



port P3, P3.3, input/output with Schmitt-trigger



NOTE: UART mode: The UART clock can only be an input. If UART mode and UART function are selected, the P3.3/UCLK0 is always an input.

SPI, slave mode: The clock applied to UCLK0 is used to shift data in and out.

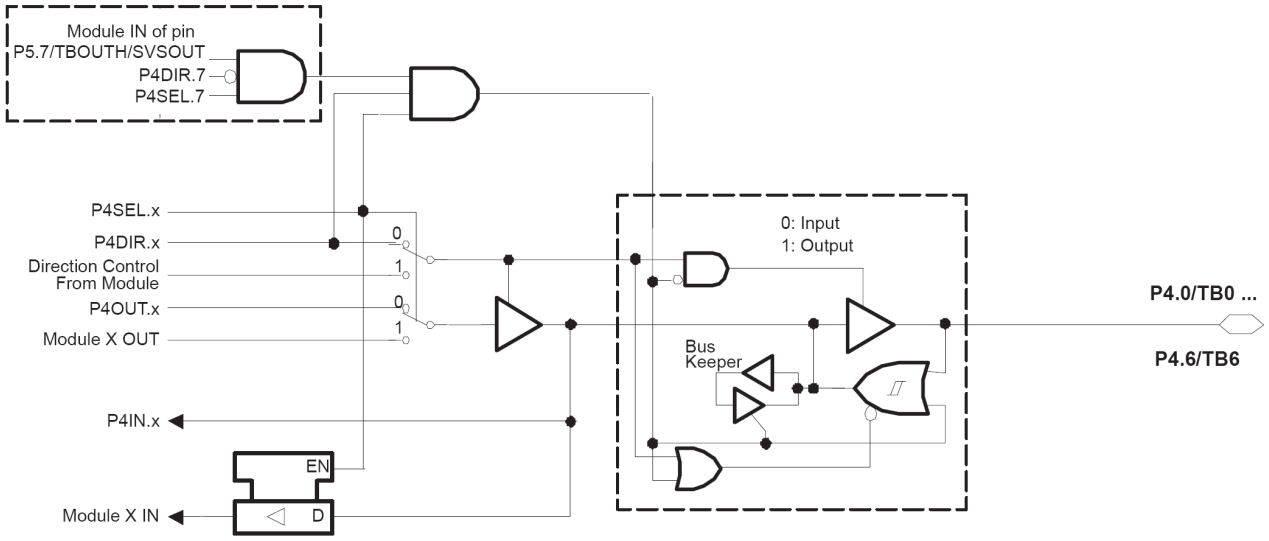
SPI, master mode: The clock to shift data in and out is supplied to connected devices on pin P3.3/UCLK0 (in slave mode).

I^2C, slave mode: The clock applied to SCL is used to shift data in and out. The frequency of the clock source of the module must be ≥ 10 times the frequency of the SCL clock.

I^2C, master mode: To shift data in and out, the clock is supplied via the SCL terminal to all I^2C slaves. The frequency of the clock source of the module must be ≥ 10 times the frequency of the SCL clock.

P4 端口的引脚原理图（P4.0~P4.6）

port P4, P4.0 to P4.6, input/output with Schmitt-trigger



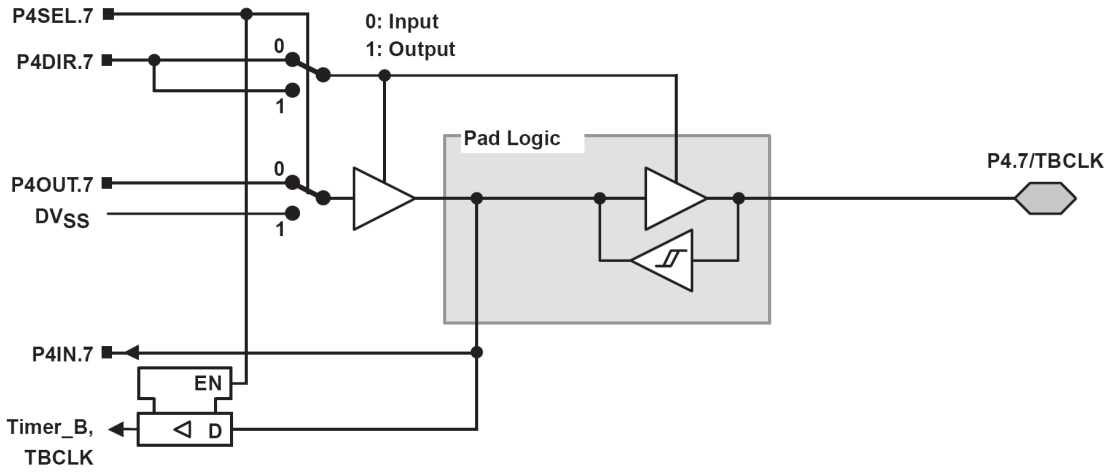
x: Bit Identifier, 0 to 6 for Port P4

PnSel.x	PnDIR.x	DIRECTION CONTROL FROM MODULE	PnOUT.x	MODULE X OUT	PnIN.x	MODULE X IN
P4Sel.0	P4DIR.0	P4DIR.0	P4OUT.0	Out0 signal†	P4IN.0	CCI0A / CCI0B‡
P4Sel.1	P4DIR.1	P4DIR.1	P4OUT.1	Out1 signal†	P4IN.1	CCI1A / CCI1B‡
P4Sel.2	P4DIR.2	P4DIR.2	P4OUT.2	Out2 signal†	P4IN.2	CCI2A / CCI2B‡
P4Sel.3	P4DIR.3	P4DIR.3	P4OUT.3	Out3 signal†	P4IN.3	CCI3A / CCI3B‡
P4Sel.4	P4DIR.4	P4DIR.4	P4OUT.4	Out4 signal†	P4IN.4	CCI4A / CCI4B‡
P4Sel.5	P4DIR.5	P4DIR.5	P4OUT.5	Out5 signal†	P4IN.5	CCI5A / CCI5B‡
P4Sel.6	P4DIR.6	P4DIR.6	P4OUT.6	Out6 signal†	P4IN.6	CCI6A

† Signal from Timer_B
‡ Signal to Timer_B

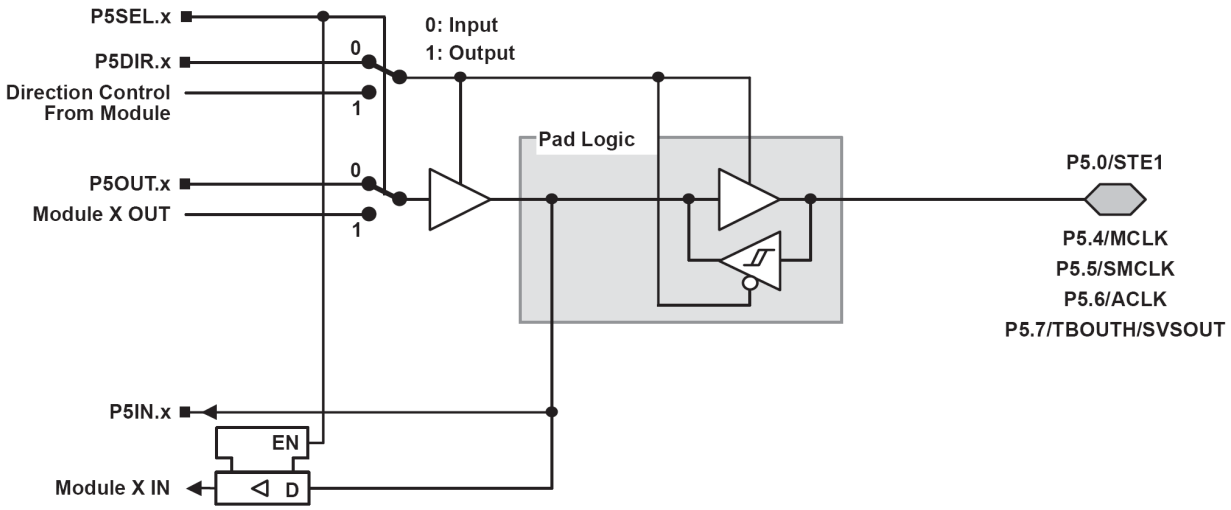
P4 端口的引脚原理图（P4.7）

port P4, P4.7, input/output with Schmitt-trigger



P5 端口的引脚原理图（P5.0，P5.4~P5.7）

port P5, P5.0 and P5.4 to P5.7, input/output with Schmitt-trigger



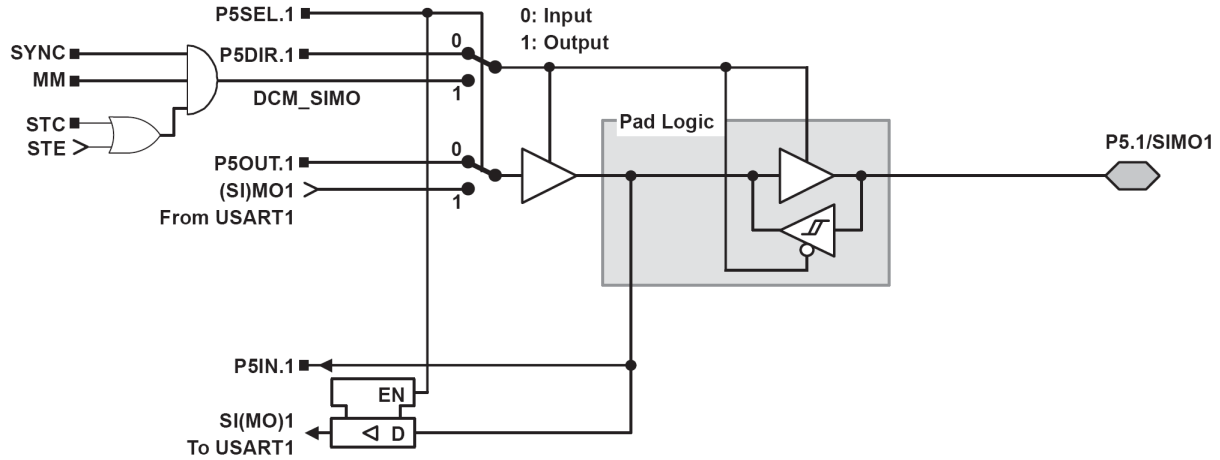
x: Bit Identifier, 0 and 4 to 7 for Port P5

PnSel.x	PnDIR.x	Dir. CONTROL FROM MODULE	PnOUT.x	MODULE X OUT	PnIN.x	MODULE X IN
P5Sel.0	P5DIR.0	DV_{SS}	P5OUT.0	DV_{SS}	P5IN.0	STE.1
P5Sel.4	P5DIR.4	DV_{CC}	P5OUT.4	MCLK	P5IN.4	unused
P5Sel.5	P5DIR.5	DV_{CC}	P5OUT.5	SMCLK	P5IN.5	unused
P5Sel.6	P5DIR.6	DV_{CC}	P5OUT.6	ACLK	P5IN.6	unused
P5Sel.7	P5DIR.7	DV_{SS}	P5OUT.7	SVSOUT	P5IN.7	TBOUTHiZ

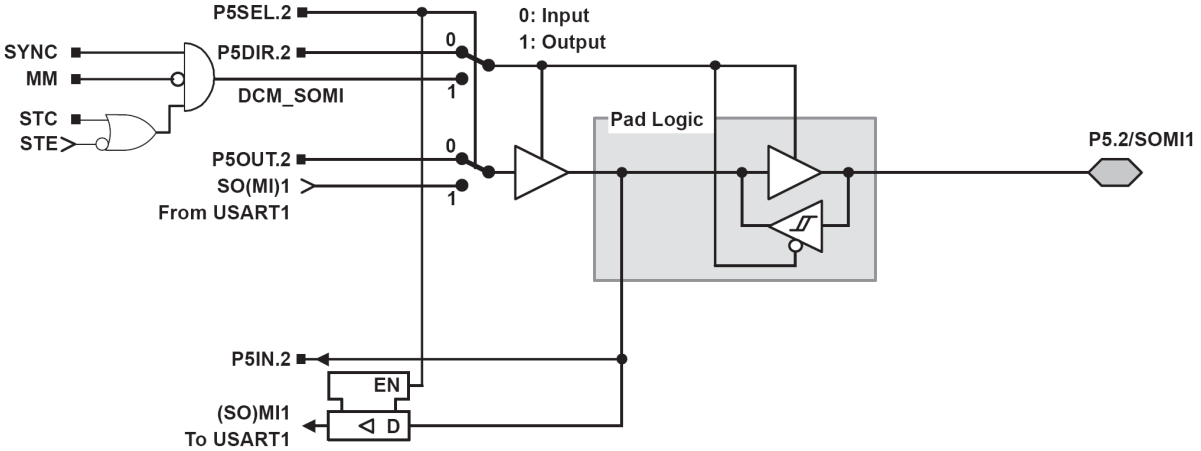
NOTE: TBOUTHiZ signal is used by port module P4, pins P4.0 to P4.6. The function of TBOUTHiZ is mainly useful when used with Timer_B7.

P5 端口的引脚原理图（P5.1，P5.2，P5.3）

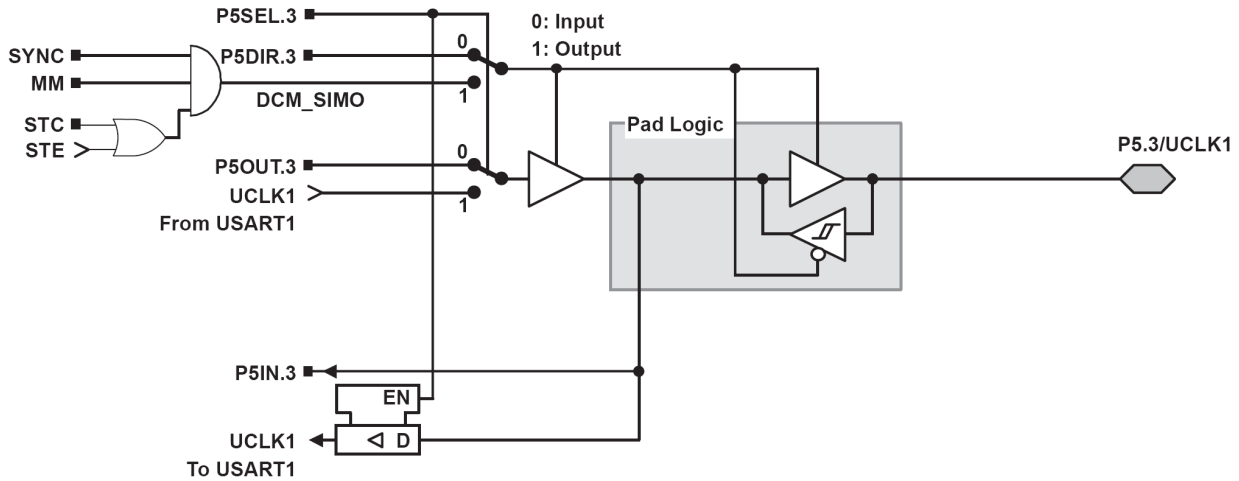
port P5, P5.1, input/output with Schmitt-trigger



port P5, P5.2, input/output with Schmitt-trigger



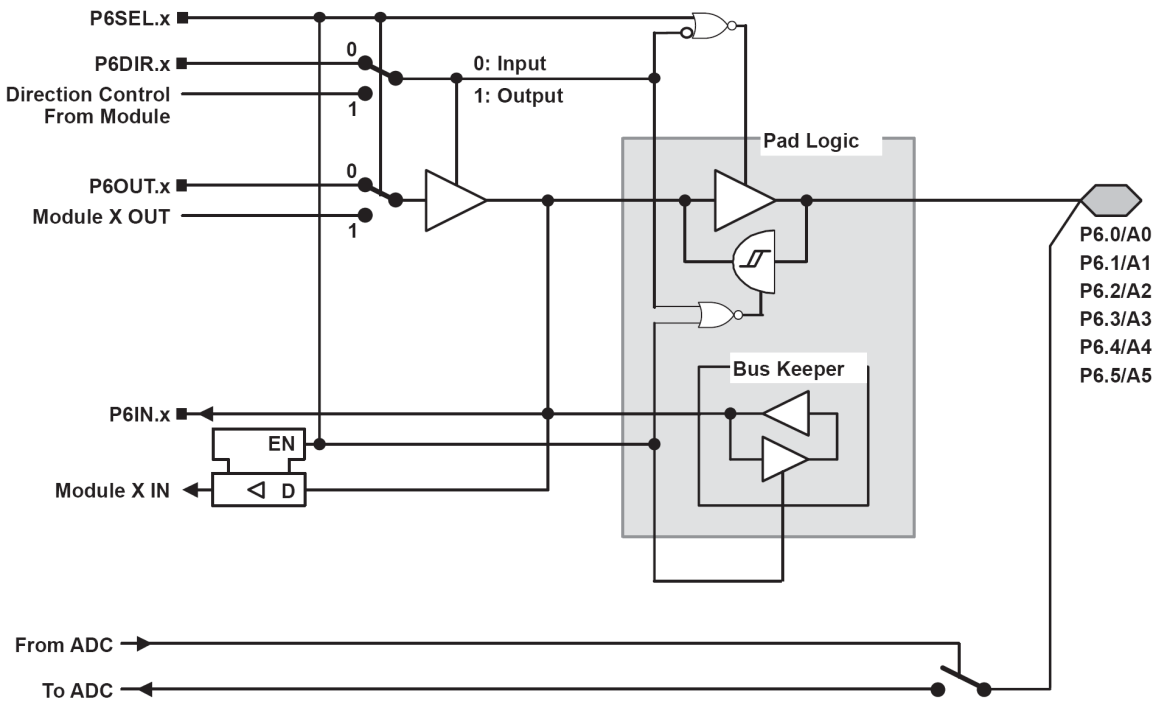
port P5, P5.3, input/output with Schmitt-trigger



NOTE: UART mode: The UART clock can only be an input. If UART mode and UART function are selected, the P5.3/UCLK1 direction is always input.
SPI, slave mode: The clock applied to UCLK1 is used to shift data in and out.
SPI, master mode: The clock to shift data in and out is supplied to connected devices on pin P5.3/UCLK1 (in slave mode).

P6 端口的引脚原理图（P6.0~P6.5）

port P6, P6.0 to P6.5, input/output with Schmitt-trigger



x: Bit Identifier, 0 to 5 for Port P6

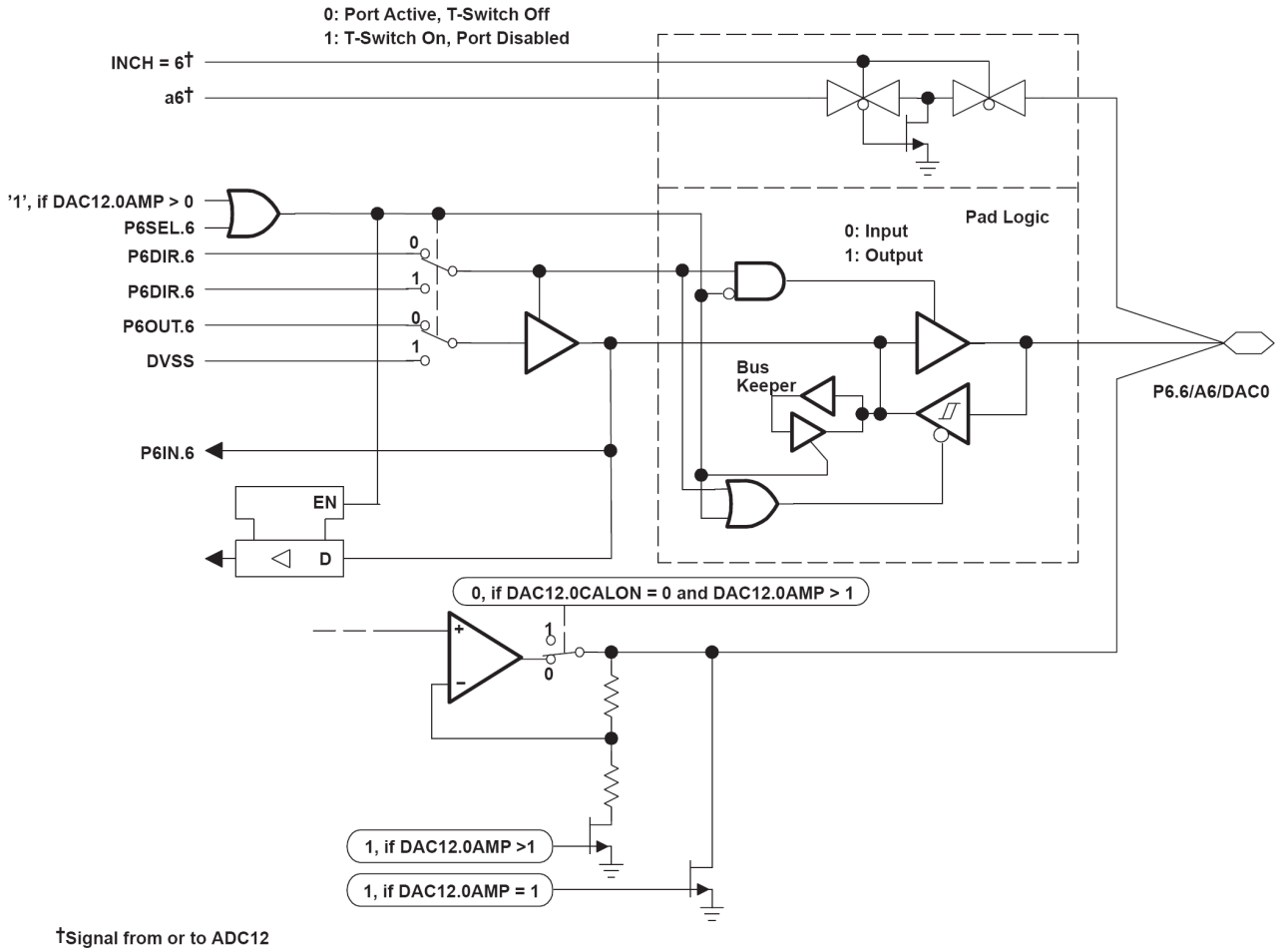
NOTE: Analog signals applied to digital gates can cause current flow from the positive to the negative terminal. The throughput current flows if the analog signal is in the range of transitions 0→1 or 1←0. The value of the throughput current depends on the driving capability of the gate. For MSP430, it is approximately 100 μA.
Use P6SEL.x=1 to prevent throughput current. P6SEL.x should be set, even if the signal at the pin is not being used by the ADC12.

PnSel.x	PnDIR.x	DIR. CONTROL FROM MODULE	PnOUT.x	MODULE X OUT	PnIN.x	MODULE X IN
P6Sel.0	P6DIR.0	P6DIR.0	P6OUT.0	DV_{SS}	P6IN.0	unused
P6Sel.1	P6DIR.1	P6DIR.1	P6OUT.1	DV_{SS}	P6IN.1	unused
P6Sel.2	P6DIR.2	P6DIR.2	P6OUT.2	DV_{SS}	P6IN.2	unused
P6Sel.3	P6DIR.3	P6DIR.3	P6OUT.3	DV_{SS}	P6IN.3	unused
P6Sel.4	P6DIR.4	P6DIR.4	P6OUT.4	DV_{SS}	P6IN.4	unused
P6Sel.5	P6DIR.5	P6DIR.5	P6OUT.5	DV_{SS}	P6IN.5	unused

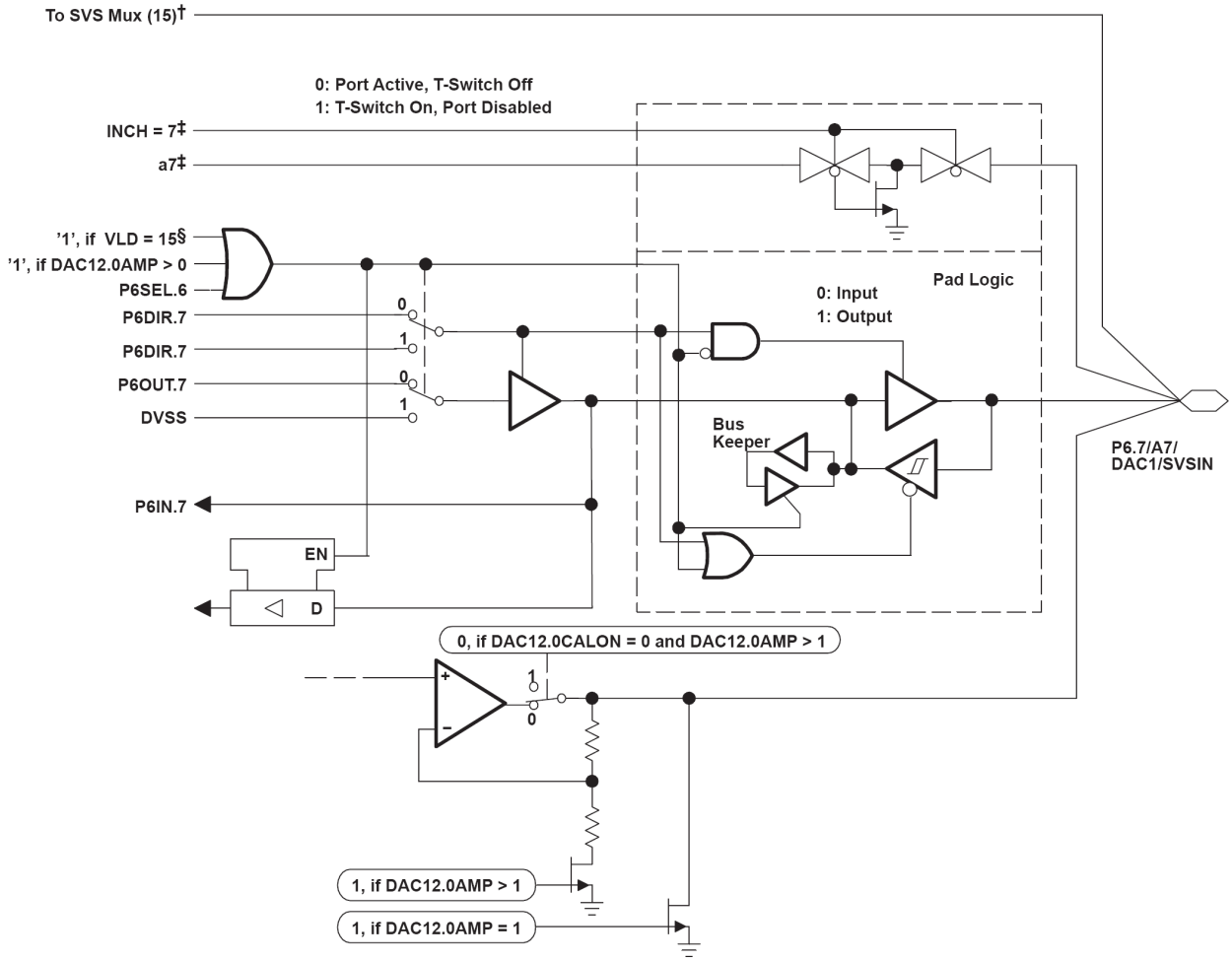
NOTE: The signal at pins P6.x/Ax is used by the 12-bit ADC module.

P6 端口的引脚原理图（P6.6，P6.7）

port P6, P6.6, input/output with Schmitt-trigger

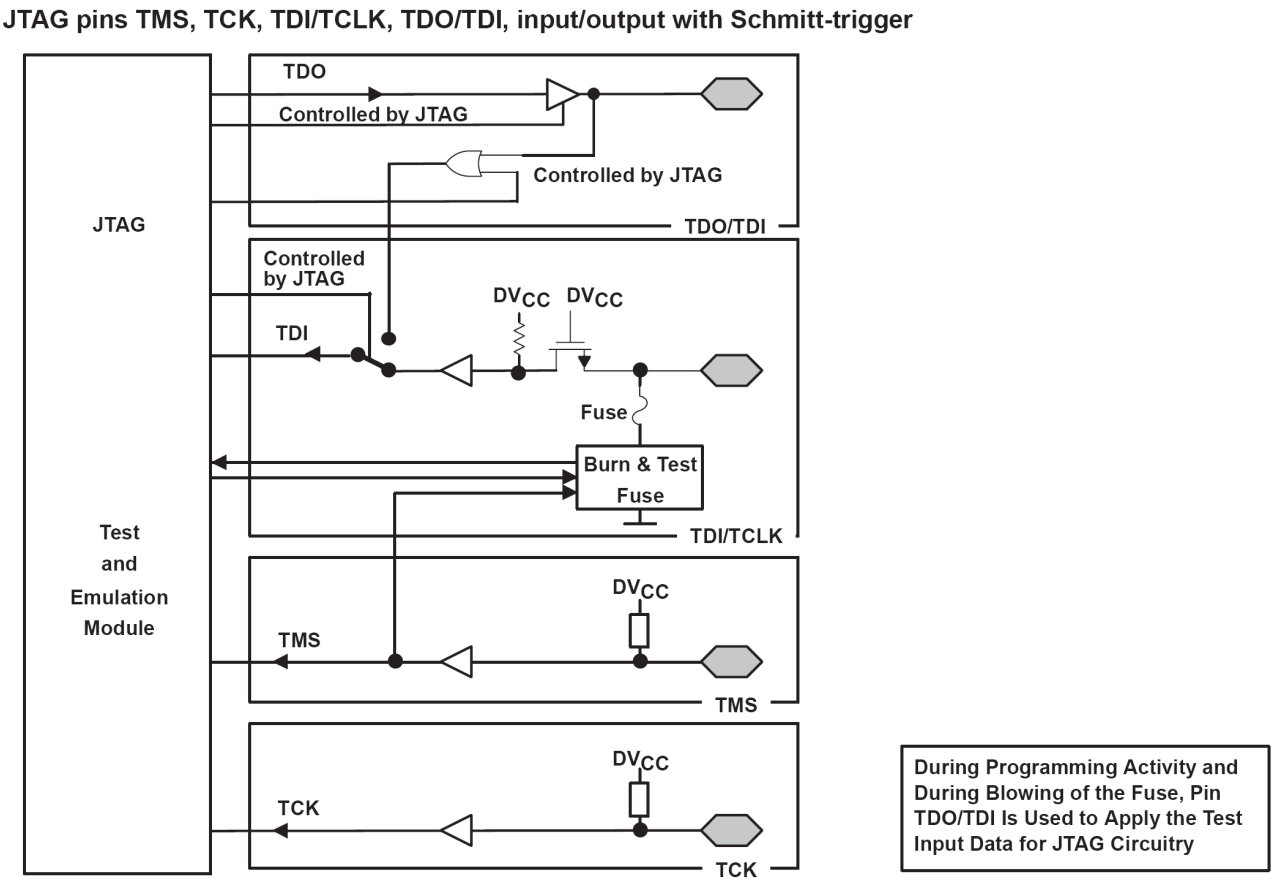


port P6, P6.7, input/output with Schmitt-trigger



†Signal to SVS Block, Selected if VLD = 15
‡Signal From or To ADC12
§VLD Control Bits are Located in SVS

JTAG 端口的引脚原理图（TMS，TCK，TDO，TDI）



5.4 使用说明

使用端口应注意以下事项：

- 在使用端口前要定义好端口方向。
- 若使用 P1 口的部分引脚作中断方式，在开总中断之前务必要设置好 P1IFG、P1IES、P1IE 寄存器的相应位，并确保相应引脚为输入方向。
- 为了尽量降低功耗，对连接引脚应设定为 I/O 功能并设为输出。由于其管脚未连接，所以其输出状态不重要。
- 中断标志需要软件清除

5.5 输入/输出端口头文件介绍

在 MSP430X16X..h 对端口定义如下所示：

```
#define  P1IN_             (0x0020)  /* Port 1 Input */
READ_ONLY DEFC( P1IN          , P1IN_)
#define  P1OUT_            (0x0021)  /* Port 1 Output */
DEFC(   P1OUT           , P1OUT_)
#define  P1DIR_            (0x0022)  /* Port 1 Direction */
DEFC(   P1DIR           , P1DIR_)
#define  P1IFG_            (0x0023)  /* Port 1 Interrupt Flag */
```

```
DEFC(   P1IFG           , P1IFG_)
#define  P1IES_           (0x0024)  /* Port 1 Interrupt Edge Select */
DEFC(   P1IES           , P1IES_)
#define  P1IE_            (0x0025)  /* Port 1 Interrupt Enable */
DEFC(   P1IE            , P1IE_)
#define  P1SEL_           (0x0026)  /* Port 1 Selection */
DEFC(   P1SEL           , P1SEL_)
#define  P2IN_            (0x0028)  /* Port 2 Input */
READ_ONLY DEFC( P2IN          , P2IN_)
#define  P2OUT_           (0x0029)  /* Port 2 Output */
DEFC(   P2OUT           , P2OUT_)
#define  P2DIR_           (0x002A)  /* Port 2 Direction */
DEFC(   P2DIR           , P2DIR_)
#define P2IFG_           (0x002B)  /* Port 2 Interrupt Flag */
DEFC(   P2IFG           , P2IFG_)
#define  P2IES_           (0x002C)  /* Port 2 Interrupt Edge Select */
DEFC(   P2IES           , P2IES_)
#define  P2IE_            (0x002D)  /* Port 2 Interrupt Enable */
DEFC(   P2IE            , P2IE_)
#define  P2SEL_           (0x002E)  /* Port 2 Selection */
DEFC(   P2SEL           , P2SEL_)
/************************************************************
* DIGITAL I/O Port3/4
************************************************************/
#define  P3IN_            (0x0018)  /* Port 3 Input */
READ_ONLY DEFC( P3IN          , P3IN_)
#define  P3OUT_           (0x0019)  /* Port 3 Output */
DEFC(   P3OUT           , P3OUT_)
#define  P3DIR_           (0x001A)  /* Port 3 Direction */
DEFC(   P3DIR           , P3DIR_)
#define P3SEL_           (0x001B)  /* Port 3 Selection */
DEFC(   P3SEL           , P3SEL_)
#define  P4IN_            (0x001C)  /* Port 4 Input */
READ_ONLY DEFC( P4IN          , P4IN_)
#define  P4OUT_           (0x001D)  /* Port 4 Output */
DEFC(   P4OUT           , P4OUT_)
#define  P4DIR_           (0x001E)  /* Port 4 Direction */
DEFC(   P4DIR           , P4DIR_)
#define  P4SEL_           (0x001F)  /* Port 4 Selection */
DEFC(   P4SEL           , P4SEL_)
/************************************************************
* DIGITAL I/O Port5/6
************************************************************/
#define  P5IN_            (0x0030)  /* Port 5 Input */
READ_ONLY DEFC( P5IN          , P5IN_)
#define  P5OUT_           (0x0031)  /* Port 5 Output */
DEFC(   P5OUT           , P5OUT_)
#define  P5DIR_           (0x0032)  /* Port 5 Direction */
DEFC(   P5DIR           , P5DIR_)
#define  P5SEL_           (0x0033)  /* Port 5 Selection */
DEFC(   P5SEL           , P5SEL_)
#define  P6IN_            (0x0034)  /* Port 6 Input */
READ_ONLY DEFC( P6IN          , P6IN_)
#define  P6OUT_           (0x0035)  /* Port 6 Output */
DEFC(   P6OUT           , P6OUT_)
#define  P6DIR_           (0x0036)  /* Port 6 Direction */
DEFC(   P6DIR           , P6DIR_)
#define  P6SEL_           (0x0037)  /* Port 6 Selection */
DEFC(   P6SEL           , P6SEL_)
```

注：

用户可直接使用 P1IN、P1OUT、P1DIR、P1IFG、P1IES、P1IE、P1SEL 对 P1 口操作；

用 P2IN、P2OUT、P2DIR、P2IFG、P2IES、P2IE、P2SEL 对 P2 口操作；

用 P3IN、P3OUT、P3DIR、P3SEL 对 P3 口操作；

用 P4IN、P4OUT、P4DIR、P4SEL 对 P4 口操作；

用 P5IN、P5OUT、P5DIR、P5SEL 对 P5 口操作；
用 P6IN、P6OUT、P6DIR、P6SEL 对 P6 口操作。

5.6 输入/输出端口使用例程

例一：用软件定时使发光二级管闪烁。

程序代码如下：

```
#include <msp430x16x.h>
void main (void)
{
  unsigned char i;
  WDTCTL=WDTPW+WDTHOLD;     // 关看门狗
  P4DIR  |= 0X10;           // P4.0 作输出（在实验系统板中 P4.0 接有一个发光二极管）
  While(1)
  {
    i=50000;
    P4OUT ^=0X01;
    While(i--);             // 软件定时
  }
}
```

例二：在系统板上实现按一次按键，对应的发光二极管状态改变一次。

```
#include <msp430x16x.h>
void main(void)
{
  WDTCTL=WDTPW+WDTHOLD;
  P4DIR = 0XfF;
  P4OUT = 0Xff;
  P1OUT = 0xff;
  P1DIR = 0xf0 ;                // All pins but button to output
  P1IES = 0x0f;                 // Button int on falling edge
  P1IFG = 0;
  P1IE = 0x0f;                  // enable P1.0--P1_3 interrupt
  _EINT();                      // 开总中断
  while(1)
  _BIS_SR(LPM4_bits);           // 进入低功耗模式 4
}

#pragma vector = PORT1_VECTOR
__interrupt void PORT1_ISR(void)   // 键盘中断
{
  unsigned int i;
  unsigned char temp;
  temp = P1IFG;
  for(i=0;i<0x1fff;i++);            // 消抖
  if((P1IN&temp)==(!temp))
   {
     switch(temp)
       {
         case 1: P4OUT ^=0X01;break; //P4.4 状态改变，即 LED1 状态改变
         case 2: P4OUT ^=0X02;break; //P4.4 状态改变，即 LED2 状态改变
         case 4: P4OUT ^=0X04;break; //P4.4 状态改变，即 LED2 状态改变
         case 8: P4OUT ^=0X08;break; //P4.4 状态改变，即 LED2 状态改变
       }
   }
  P1IFG = 0;                        // 清除中断请求位
}
```

第六章 WDT 看门狗定时器

Watchdog Timer Module

6.1 看门狗定时器概述

看门狗定时器（WDT）是 MSP430 系列单片机中常用的一种部件。

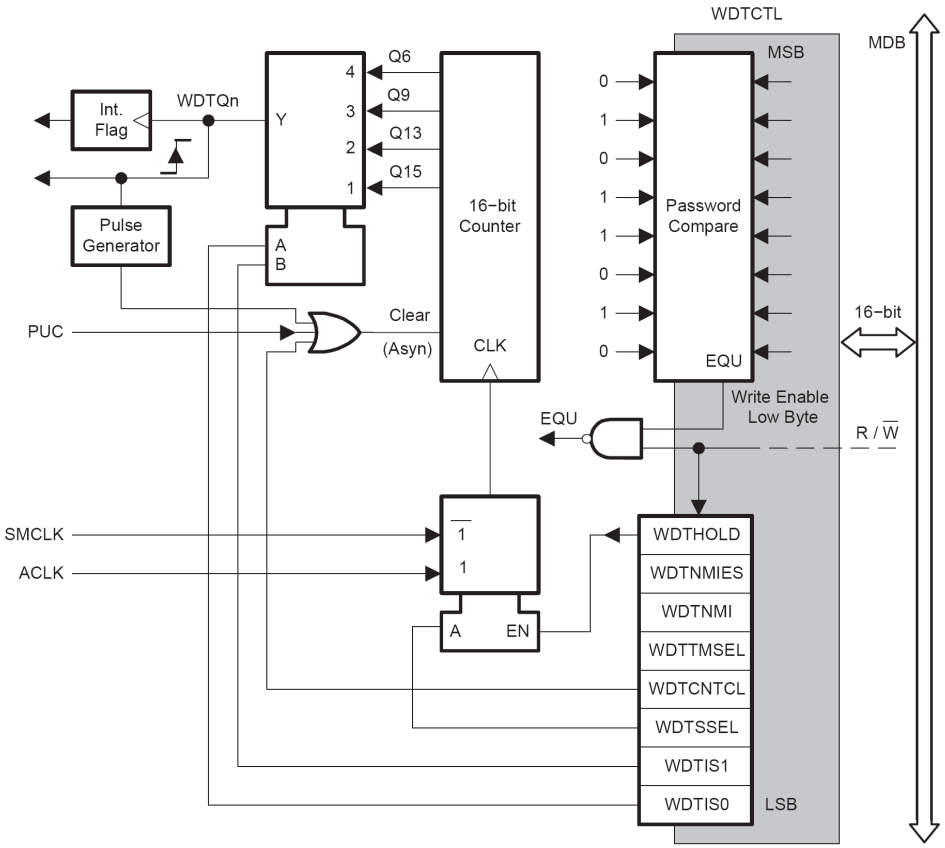
在工业现场，往往会由于供电电源、空间电磁干扰或其他原因引起强烈的干扰噪声。这些干扰作用于数字器件，极易使其产生误动作，从而失去应有的控制功能，引起 MSP430 发生“程序跑飞”事故。若不进行有效的处理，程序就不能回到正常的状态，从而失去应有的控制功能。

看门狗定时器正是为了解放这类问题而产生的，尤其是在具有循环结构的程序任务中更为有效。

在正常操作器件，一次 WDT 定时时间到，将产生一次器件复位。如果通过编制程序使 WDT 定时时间稍大于程序执行一遍所用的时间时，并且程序执行过程中都有对看门狗定时器清零的指令，使计数器重新计数，则程序正常执行时，就会在 WDT 定时时间到达之前对 WDT 清零，不会产生 WDT 溢出，如果由于干扰使程序跑飞，则不会在 WDT 定时时间到达之前执行 WDT 清零指令，WDT 就会产生溢出，从而产生系统复位 CPU 需用重新运行用户程序，这样程序就可以又恢复正常运行状态。

MSP430 看门狗除了具有上述系统监测的特定用途之外，还可以作为内部定时器来使用，当选择的时间到达之后，和其他定时器一样产生一个定时中断。此外 WDT 还可以被完全停止活动以支持超低功耗应用。

6.2 看门狗定时器结构

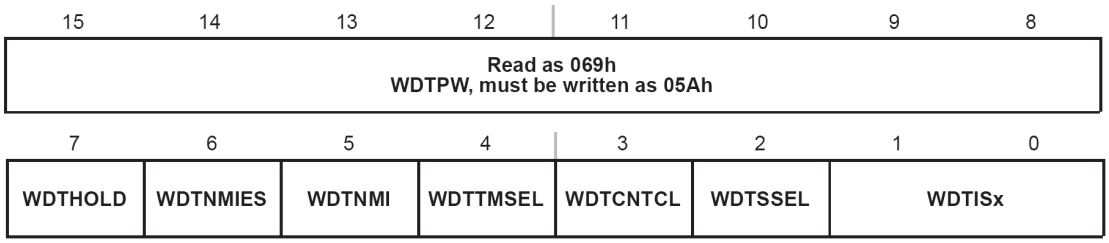

WDTCTL
MDB
MSB
LSB
Int. Flag
WDTQn
Pulse Generator
Y
4
3
2
1
Q6
Q9
Q13
Q15
A
B
16-bit Counter
CLK
Clear
(Asyn)
PUC
SMCLK
ACLK
0
1
0
1
1
0
1
0
Password Compare
EQU
16-bit
Write Enable Low Byte
R / W
EQU
1
1
A
EN
WDTHOLD
WDTNMIES
WDTNMI
WDTTMSEL
WDTCNTCL
WDTSSEL
WDTIS1
WDTIS0

6.3 看门狗定时器寄存器

Register	Short Form	Register Type	Address	Initial State
Watchdog timer control register	WDTCTL	Read/write	0120h	06900h with PUC
SFR interrupt enable register 1	IE1	Read/write	0000h	Reset with PUC
SFR interrupt flag register 1	IFG1	Read/write	0002h	Reset with PUC†

† WDTIFG is reset with POR

[1] WDTCTL　看门狗控制寄存器



WDTCTL 由两部分组成：

高 8 位是对 WDT 操作的控制命令。要写入操作 WDT 的控制命令，出于安全原因必须先正确写入高字节看门狗口令。口令位 5AH，如果口令写错将导致系统复位。

读 WDTCTL 时，不需要口令，可直接读取地址 120H 中的内容，读出数据低字节位 WDTCTL 的值，高字节始终位 69H。WDTCTL 除了看门狗定时器的控制位之外，还有两个用于设置 NMI 引脚功能。

WDTISx：选择看门狗定时器的计时输出

其中 T 是 WDTCNT 的输入时钟源周期。

0　$T\times2^{15}$

1　$T\times2^{13}$

2　$T\times2^{9}$

3　$T\times2^{6}$

WDTSSEL：选择 WDTCNT 的时钟源

0　SMCLK

1　ACLK

由 WDTISx 及 WDTSSEL 位便可确定 WDT 定时时间，因此通过软件对计数器设置不同的初始值就可实现不同时间的定时。与其他定时器不同之处在于，WDT 最多只能定时 8 种和时钟源相关的时间。

下表列出了 WDT 可选的定时时间（晶振为 32.768KHz，SMCLK=1MHz）。

WDTSSEL	WDTIS1	WDTIS0	定时时间/ms	
0	1	1	0.064	$T_{smclk}\times2^{6}$
0	1	0	0.51	$T_{smclk}\times2^{9}$
1	1	1	1.95	$T_{aclk}\times2^{6}$
0	0	1	8.19	$T_{smclk}\times2^{13}$
1	1	0	15.63	$T_{aclk}\times2^{9}$
0	0	0	32.77	$T_{aclk}\times2^{15}$
1	0	1	250	$T_{aclk}\times2^{13}$
1	0	0	1000	$T_{aclk}\times2^{15}$

WDTCNTCL：看门狗计数器清零位

0： 无操作

1： 将 WDTCTL 的值清零

当该位为 1 时，清除 WDTCNT。

WDTTMSEL：工作模式选择

0 看门狗模式；

1 定时器模式。

WDTNMI：选择 RST/NMI 引脚功能（在 PUC 后被复位）

0 RST/NMI 引脚为复位端；

1 RST/NMI 引脚为边沿触发的非屏蔽中断输入。

WDTNMIES：选择中断的边沿触发方式

0 WDT 功能激活；

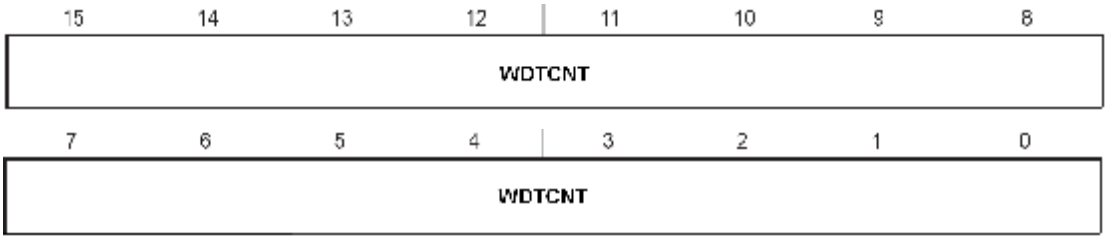
1 时钟禁止输入，计数停止。

WDTHOLD：停止看门狗计数器

0 WDT 功能激活

1 时钟禁止输入，计数停止

[2] WDTCNT 计数单元



WDTCNT 时 16 位增计数器，由于 MSP430 所选定的时钟电路产生的固定周期脉冲信号对计数器进行加法计数。如果计数器事先被预置的初始状态不同，那么从开始计数到计数溢出为止的时间就不同。WDTCNT 不能直接通过软件直接通过软件存取，必须通过看门狗定时器的控制寄存器 WDTCTL 来控制。

[3] IE1 中断允许寄存器

WDTIE：看门狗定时器模式中断允许控制位

0 看门狗定时器中断禁止

1 看门狗定时器中断允许

NMIIE：非屏蔽中断允许控制位

0 禁止非屏蔽中断

1 允许非屏蔽中断

[4] IFG1　　中断标志寄存器

7	6	5	4	3	2	1	0
			NMIIFG				WDTIFG

WDTIFG：看门狗定时器模式中断标志

0　无看门狗定时器中断标志

1　有看门狗定时器中断标志

NMIIFG：非屏蔽中断标志

0　无非屏蔽中断标志

1　有非屏蔽中断标志

6.4 看门狗定时器操作

用户可以通过 WDTCTL 寄存器中的 WDTTMSEL 和 WDTHOLD 控制位设置 WDT 工作在看门狗模式、定时器模式和低功耗模式。

[1] 看门狗模式

由于在上电复位或系统复位时，WDTCNT 和 WDTCL 两寄存器内容被全部清除（晶振为 32768Hz，SMCLK=1MHz）：

复位后，工作在看门狗模式，以 ACLK 为时钟，WDT 定时时间为 32ms。

上述情况将导致 WDT 的运行自动进入看门狗模式。

所以，用户软件一般都需要进行如下操作：

- 进行 WDT 的初始化：设置合适的时间（通过 SSEL、IS0、IS1 位来选定）。
- 周期性的对 WDTCNT 清零：防止 WDT 溢出，保证 WDT 的正确使用。
- 在看门狗模式下，如果计数器超过了定时时间，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。
- 如果系统不用看门狗功能，应该在程序开始处禁止看门狗功能。

[2] 定时器模式

WDTCTL 的 WDTTMSEL 位置位选择定时器模式。这一模式产生选定时间的周期性中断。定时时间可以通过 WDTCTL 的 WDTCNCTL 位置位来开始。

- 改变定时时间而不同时清楚 WDTCNT 将导致不可预料的系统立即复位或中断。定时时间可以通过 WDTCTL 的 CNCTL 位置位来开始。
- 如果先后分别进行清除和定时时间选择，则可能立即引起不可预料的系统复位或中断。
- 另外，在终场工作时。改变时钟源可能导致 WDTCNT 额外的计数时钟。

[3] 低功耗模式

当系统不需要 WDT 做看门狗和定时器时，可关闭 WDT 以减小功耗。

控制位 WDTHOLD=1 时关闭 WDT，这时看门狗停止工作。

6.5 看门狗定时器头文件定义

在 msp430x14x.h 头文件中定义如下：

```
/************************************************************
* WATCHDOG TIMER
************************************************************/

#define WDTCTL_           (0x0120)  /* Watchdog Timer Control */
DEFW(   WDTCTL          , WDTCTL_)
/* The bit names have been prefixed with "WDT" */
#define WDTIS0            (0x0001)
#define WDTIS1            (0x0002)
#define WDTSSEL           (0x0004)
#define WDTCNTCL          (0x0008)
#define WDTTMSEL          (0x0010)
#define WDTNMI            (0x0020)
#define WDTNMIES          (0x0040)
#define WDTHOLD           (0x0080)

#define WDTPW             (0x5A00)

/* WDT-interval times [1ms] coded with Bits 0-2 */
/* WDT is clocked by fMCLK (assumed 1MHz) */
#define WDT_MDLY_32        (WDTPW+WDTTMSEL+WDTCNTCL)       /* 32ms interval (default) */
#define WDT_MDLY_8         (WDTPW+WDTTMSEL+WDTCNTCL+WDTIS0)                /* 8ms     " */
#define WDT_MDLY_0_5       (WDTPW+WDTTMSEL+WDTCNTCL+WDTIS1)                /* 0.5ms   " */
#define WDT_MDLY_0_064     (WDTPW+WDTTMSEL+WDTCNTCL+WDTIS1+WDTIS0)         /* 0.064ms " */

/* WDT is clocked by fACLK (assumed 32KHz) */
#define WDT_ADLY_1000      (WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL)               /* 1000ms  " */
#define WDT_ADLY_250       (WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL+WDTIS0)        /* 250ms   " */
#define WDT_ADLY_16        (WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL+WDTIS1)        /* 16ms    " */
#define WDT_ADLY_1_9       (WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL+WDTIS1+WDTIS0)/* 1.9ms   " */

/* Watchdog mode -> reset after expired time */
/* WDT is clocked by fMCLK (assumed 1MHz) */
#define WDT_MRST_32        (WDTPW+WDTCNTCL)                /* 32ms interval (default) */
#define WDT_MRST_8         (WDTPW+WDTCNTCL+WDTIS0)                         /* 8ms     " */
#define WDT_MRST_0_5       (WDTPW+WDTCNTCL+WDTIS1)                         /* 0.5ms   " */
#define WDT_MRST_0_064     (WDTPW+WDTCNTCL+WDTIS1+WDTIS0)                  /* 0.064ms " */

/* WDT is clocked by fACLK (assumed 32KHz) */
#define WDT_ARST_1000      (WDTPW+WDTCNTCL+WDTSSEL)                        /* 1000ms  " */
#define WDT_ARST_250       (WDTPW+WDTCNTCL+WDTSSEL+WDTIS0)                 /* 250ms   " */
#define WDT_ARST_16        (WDTPW+WDTCNTCL+WDTSSEL+WDTIS1)                 /* 16ms    " */
#define WDT_ARST_1_9       (WDTPW+WDTCNTCL+WDTSSEL+WDTIS1+WDTIS0)          /* 1.9ms   " */
```

上述定义中 WDT_MRST_32 的定义里面包括了(WDTPW+WDTCNTCL)，
所以我们使用 WDT 的时候可以直接写成下面的形式：

做定时器用：

```
WDTCTL = WDT_ADLY_1000;     // 以 ACLK 延迟 1000ms
WDTCTL = WDT_MDLY_32;       // 以 SMCLK 延迟 32ms
// 记得要开 IE1 中的 WDTIE 和_EINT();
```

做看门狗用：

```
WDTCTL = WDT_ARST_1000;     // 以 ACLK 设定 1000ms 的看门狗
WDTCTL = WDT_MRST_32;       // 以 SMCLK 设定 32ms 的看门狗
// 由于在看门狗的定义里面包含 WDTCNCTL
// 所以需要喂狗的时候只需要再写一边 WDTCTL = WDT_MRST_32; 就行了
```

6.6 看门狗定时器示例程序

例：使用看门狗定时器功能产生一个方波（周期的取反 P1.0）。

```
#include <msp430x16x.h>

void main(void)
{
  WDTCTL = WDT_MDLY_32;     // 以 SMCLK 进行 32ms 的延迟
  IE1 |= WDTIE;             // 开 WDT 中断
  P1DIR |= BIT0;            // 信号输出
  _EINT();                  // 总中断开

  while(1)
  {
    _NOP();
  }
}

// 看门狗 中断服务子程序
#pragma vector = WDT_VECTOR
__interrupt void WDT_IRQ(void)
{
  P1OUT ^= BIT0;        // 对引脚取反
}
```

第七章 TimerA 16 位定时器 A 模块

16-bit Timer/Counter with three Capture/Compare Timer Module

7.1 16 位定时器 A 模块概述

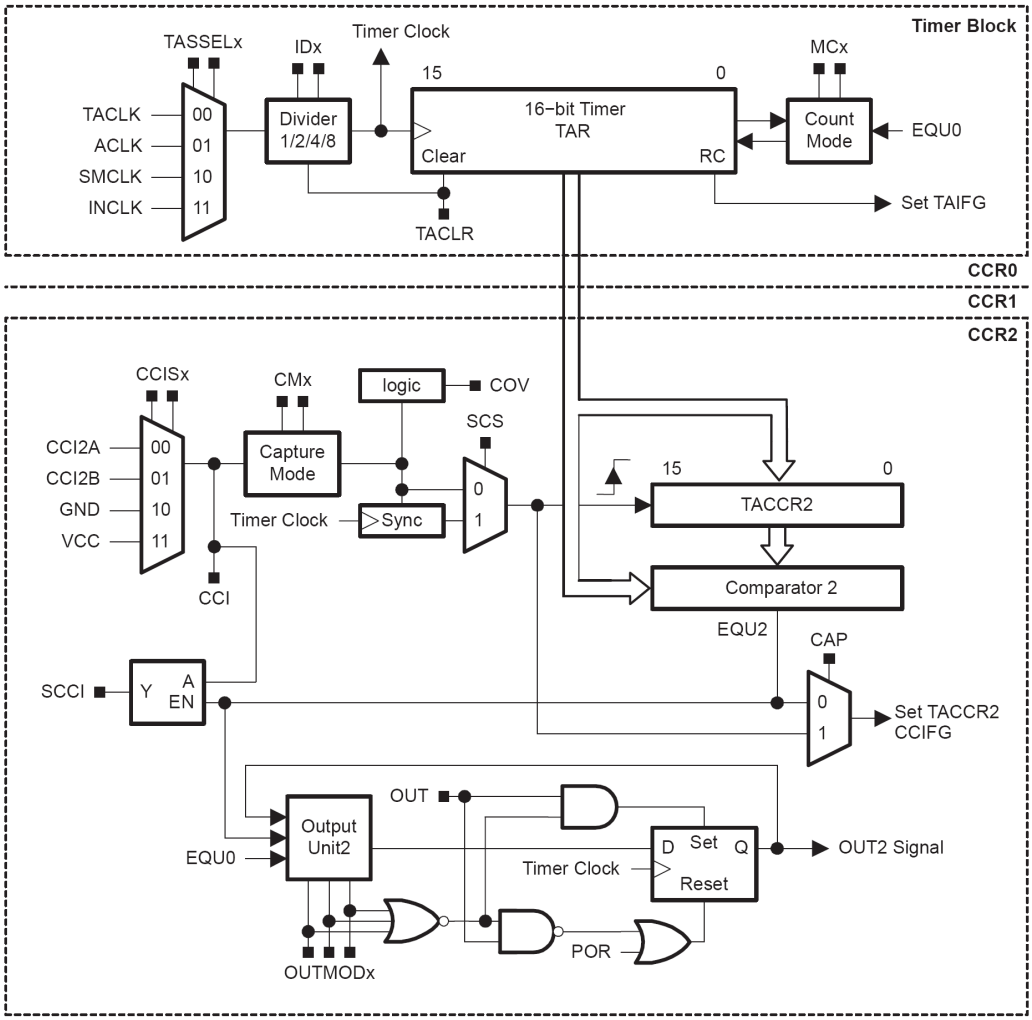
TI 推出的所有 MSP430 系列单片机都有定时器 A(TIMER_A)，它是程序设计的核心，它有一个十六位定时器和多路比较/捕获通道组成。每一个比较/捕获通道都以十六位定时器的定时功能为核心进行单独的控制。MSP430 系列单片机的 TIMER_A 有以下特性：

- 输入时钟可以有多种选择，可以是慢时钟，快时钟以及外部时钟
- 虽然没有自动重装功能，但产生的定时脉冲或 PWM（脉宽调制）信号没有软件带来的误差
- 不仅能捕获外部事件发生的时间还可锁定其发生时的高低电平
- 可实现串口通信
- 完善的中断服务功能
- 4 种计数功能选择
- 8 路输出方式选择
- 支持多时序控制
- DMA 使能

MSP430 系列单片机的 TIMER_A 结构复杂，功能强大，适合应用于工业控制，如数字化电机控制，电表和手持式仪表的理想配置。它给开发人员提供了较多灵活的选择余地。例如，虽然在采用廉价的单片机进行产品设计时，用 RC 充放电原理测量已经是很平常的事。但是，由于单片机比较廉价，往往分辨率很低。MSP430 系列单片机采用 16 位的 TIMER_A 定时器，再加上内部的比较器，至少能达到 10 位的 A/D 测量精度；利用 TIMER_A 生成的 PWM 能用软件任意改变占空比和周期，配合滤波器件可方便地实现 D/A 转换；当 PWM 不需要修改占空比和时间时，TIMER_A 能自动输出 PWM，而不需利用中断维持 PWM 输出。

7.2 16 位定时器 A 模块结构

TIMER_A 的结构原理图。



TIMER_A 由以下部分组成：

[1] 计数器部分

输入的时钟源具有 4 种选择，所选择的时钟源又可以 1、2、4、8 分频作为计数频率，TIMER_A 可以通过选择 4 种工作模式灵活的完成定时/计数功能。

[2] 捕获/比较

用于捕获事件发生的时间或产生时间间隔，捕获比较功能的引入主要时为了提高 I/O 端口处理事务的能力和速度。不同的 MSP430 单片机 TIMER_A 模块中所含有的捕获/比较器的数量不一样，每个捕获/比较器的结构完全相同，输入和输出都取决于各自所带的控制寄存器的控制字，捕获/比较器相互之间工作完全独立。

[3] 输出单元

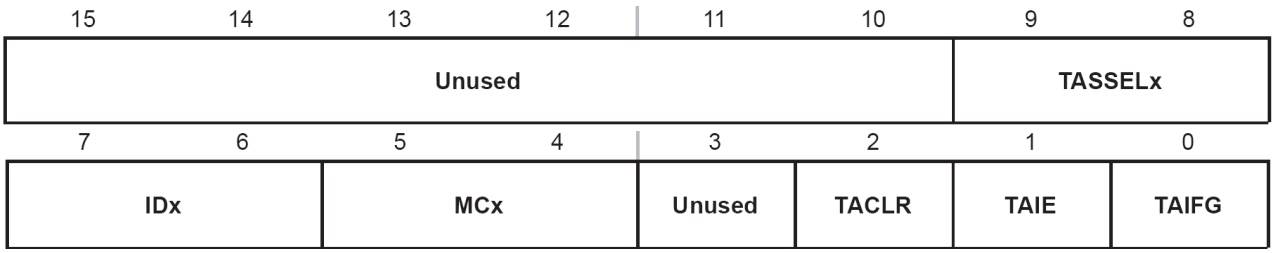
具有可选的 8 种输出模式，用于产生用户需要的输出信号，支持 PWM。

7.3 16 位定时器 A 模块寄存器

16 位定时器 A 模块寄存器有下列：

寄存器名称	寄存器缩写
TimerA 控制寄存器	TACTL
TimerA 计数器	TAR
TimerA 中断向量寄存器	TAIV
捕获/比较寄存器 0	TACCR0
捕获/比较寄存器 1	TACCR1
捕获/比较寄存器 2	TACCR2
捕获/比较控制寄存器 0	TACCTL0
捕获/比较控制寄存器 1	TACCTL1
捕获/比较控制寄存器 2	TACCTL2

[1] TACTL　　TIMER_A 控制寄存器



全部关于定时器及其操作的控制位都包含在定时器控制寄存器 TACTL 中。POR 信号后 TACTL 的所有位都自动复位，但在 PUC 信号后不受影响。TACTL 各位的定义如下：

TASSELx：选择定时器进入输入分频器的时钟源

0： TACLK 特定的外部引脚时钟
1： ACLK 辅助时钟
2： MCLK 系统时钟
3： INCLK 器件特有时钟

IDx：输入分频选择

0： 不分频
1： 2 分频
2： 4 分频
3： 8 分频

MCx：计数模式控制位

0： 停止模式
1： 增计数模式
2： 连续计数模式
3： 增/减计数模式

TACLR：定时器清除位

POR 或 CLR 置位时定时器和输入分频器复位。CLR 由硬件自动复位，其读出始终为 0。定时器再下一个有效输入沿开始工作。如果不是被清除模式控制位暂停，则定时器以增计数模式开始工作。

0： 无操作
1： 清除 TAR，时钟分频，计数模式的设置。清除设置后自动清零

TAIE：定时器中断允许位

0： 禁止定时器溢出中断

1： 允许定时器溢出中断

TAIFG：定时器溢出标志位

增计数模式时：当定时器由 CCR0 计数到 0，TAIFG 置位；

连续计数模式时：当定时器由 0FFFFH 计数到 0 时，TAIFG 置位

增/减计数模式时：当定时器由 CCR0 减计数到 0 时，TAIFG 置位。

0： 没有 TA 中断请求

1： 有 TA 中断请求

[2] TAR TIMER_A 计数器

15	14	13	12	11	10	9	8
TARx							
7	6	5	4	3	2	1	0
TARx							

该单元就是执行计数的单元，时计数器的主体，其内容可读可写。

[3] TACCTLx TIMER_A 捕获/比较控制寄存器 x

15	14	13	12	11	10	9	8
CMx		CCISx		SCS	SCCI	Unused	CAP
7	6	5	4	3	2	1	0
OUTMODx			CCIE	CCI	OUT	COV	CCIFG

TIMER_A 有多个捕获/比较模块，每个模块都有自己的控制字 TACCTLx，这里 x 为捕获/比较模块序号。该寄存器再 POR 信号后全部复位，但在 PUC 信号后不受影响。该寄存器中各位的定义如下：

CMx：选择捕获模式

0： 禁止捕获模式

1： 上升沿捕获

2： 下降沿捕获

3： 上升沿和下降沿都捕获

CCISx：在捕获模式中用来定义提供捕获事件的输入源

0： 选择 CCIxA

1： 选择 CCIxB

2： 选择 GND

3： 选择 Vcc

SCS：选择捕获信号与定时时钟同步/异步关系

异步捕获模式允许在请求时立即将 CCIFG 置位和捕获定时器值，适用于捕获信号的周期远大于定时器周期的情况。但是，如果定时器时钟和捕获信号发生时间竞争，则捕获寄存器的值可能出错。

0： 异步捕获

1： 同步捕获

SCCI：同步比较/捕获输入

比较相等信号 EQU 信号将选中的捕获/比较输入信号 CCI 进行锁存，然后可由 SCCI 读出。

CAP：选择捕获模式/比较模式

如果通过捕获/比较寄存器 TACCTLx 中的 CAP 使工作模式从比较模式变为捕获模式，那么不应同时进行捕获，否则，在捕获/比较寄存器中的值使不可预料的。

推荐的指令顺序如下：（1）修改控制寄存器，由比较模式切换到捕获模式。（2）捕获

0： 比较模式

1： 捕获模式

OUTMODx：选择输出模式

0： 输出

1： 置位

2： PWM 翻转/复位

3： 置位/复位

4： 翻转

5： 复位

6： PWM 翻转/置位

7： PWM 复位/置位

CCIE：捕获/比较模块中断允许位

0： 禁止中断（TACCRx）

1： 允许中断（TACCRx）

CCI：捕获/比较模块的输入信号

捕获模式：由 CCIS0 和 CCIS1 选择的输入信号可通过该位读出

比较模式：CCI 复位

OUT：输出信号

如果 OUTMODx 选择输出模式 0(输出)，则该位对应于输入状态。

0： 输出低电平

1： 输出高电平

COV：捕获溢出标志

当 CAP=0 时，选择比较模式.捕获信号发生复位。没有使 COV 置位的捕获事件。

当 CAP＝1 时，选择捕获模式。如果捕获寄存器的值被读出前在此发生捕获事件，则 COV 置位。程序可检测 COV 来判断原值读出前是否又发生捕获事件。读捕获寄存器时不会使溢出标志复位，须用软件复位。

0： 没有捕获溢出

1： 发生捕获溢出

CCIFG：捕获比较中断标志

捕获模式：寄存器 CCRx 捕获了定时器 TAR 值时置位。

比较模式：定时器 TAR 值等于寄存器 CCRx 值时置位。

0： 没有中断请求（TACCRx）

1： 有中断请求（TACCRx）

[4] TACCRx　　TIMER_A 捕获/比较寄存器 0

15	14	13	12	11	10	9	8
TACCRx							
7	6	5	4	3	2	1	0
TACCRx							

在捕获/比较模块中，可读可写。

在捕获方式，当满足捕获条件，硬件自动将计数器 TAR 数据写入该寄存器。如果测量某窄脉冲(高电平)脉冲长度，可定义上升沿和下降沿都捕获。在上升沿时，捕获一个定时器数据，这个数据在捕获寄存器中读出；再等待下降沿到了，在下降沿时又捕获一个定时器数据；那么两次捕获的定时器数据就时窄脉冲的高电平宽度。

其中 CCR0 经常用作周期寄存器，其他 CCRx 相同。

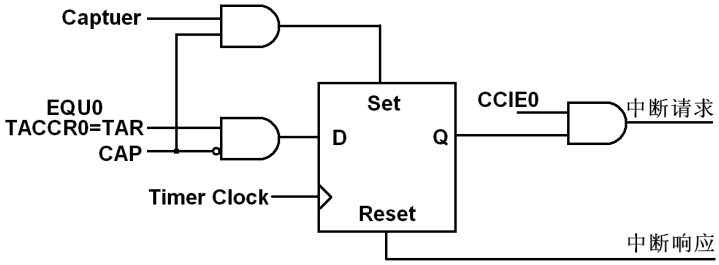
[5] TAIV　TIMER_A 中断向量寄存器

15	14	13	12	11	10	9	8
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
0	0	0	0	TAIVx			0

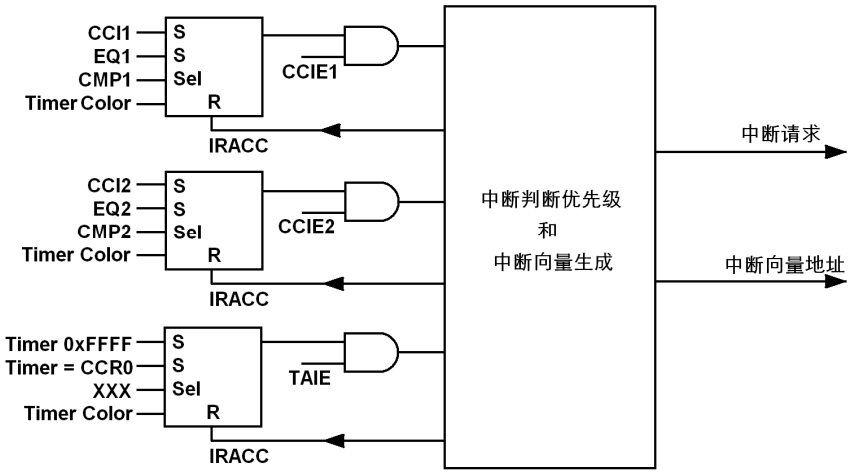
TIMER_A 中断可由计数器溢出引起，也可以来自捕获/比较寄存器。每个捕获/比较模块可独立编程，由捕获/比较外部信号以产生中断。外部信号可以是上升沿，也可以是下降沿，也可以两者都有。

Timer_A 模块使用两个中断向量，一个单独分配给捕获/比较寄存器 CCR0，另一个作为共用中断向量用于定时器和其他的捕获/比较寄存器。

TAIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending	–	
02h	Capture/compare 1	TACCR1 CCIFG	Highest
04h	Capture/compare 2	TACCR2 CCIFG	
06h	Reserved	–	
08h	Reserved	–	
0Ah	Timer overflow	TAIFG	
0Ch	Reserved	–	
0Eh	Reserved	–	Lowest



捕获/比较寄存器 CCR0 中断向量具有最高的优先级，因为 CCR0 能用于定义增计数和增/减计数模式的周期，因此，它需要最高的服务。CCIFG0 在被中断服务时能自动复位。



CCR1~CCRx 和定时器共用另一个中断向量，属于多源中断，对应的中断标志 CCIFG1~CCIFGx 和 TAIFG1 在读中断向量字 TAIV 后，自动复位。如果不访问 TAIV 寄存器，则不能自动复位，须用软件清除；如果对应的中断允许位复位（不允许中断），则将不会产生中断请求，但中断标志仍然存在，这时须用软件清除。

7.4 16 位定时器 A 模块定时工作原理

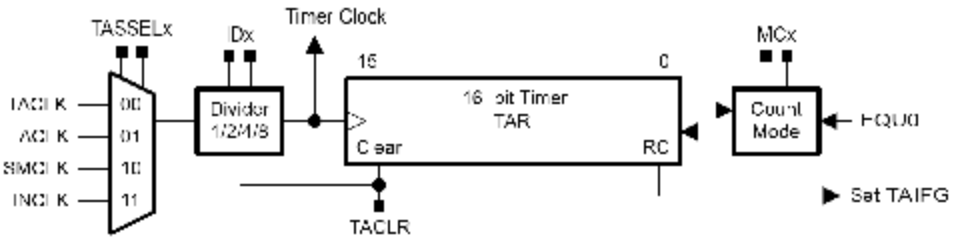
TIMER_A 共有 4 种技术模式，可以根据需要，灵活选用：

- 停止模式
- 增计数模式
- 连续技术模式
- 增/减计数模式

[1] 停止模式

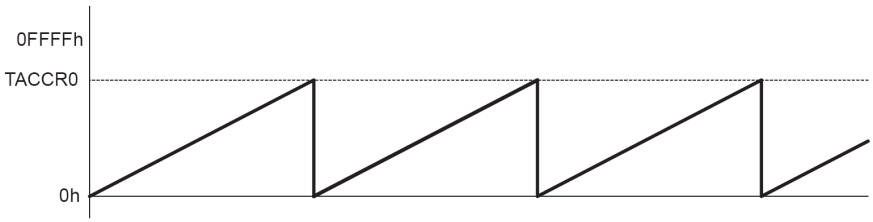
停止模式用于定时器暂停，并不发生复位，所有寄存器现行的内容在停止模式结束后都可用。当定时器暂停后重新计数时，计数器将从暂停的值开始以暂停前的计数方向计数。

例如，停止模式前，TIMER_A 工作于增/减计数模式并且处于下降计数方向，停止模式后，TIMER_A 仍然工作于增/减计数模式，从暂停前的状态开始继续沿着下降方向开始计数。如果不能这样，则可通过 TACTL 中的 CLR 位控制位来清除定时器的方向记忆特性。

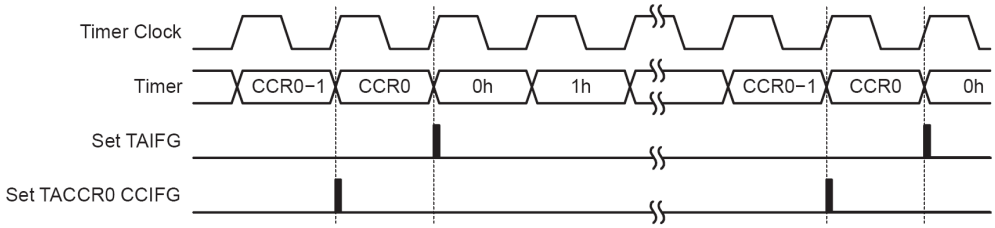


[2] 增计数模式

捕获/比较寄存器 TACCR0 用作 TIMER_A 增计数模式的周期寄存器，因为 TACCR0 为 16 为寄存器，所以该模式适用于定时器周期小于 65536 的连续计数情况。计数器 TAR 可以增计数到 TACCR0 的值，当计数值于 TACCR0 的值相等(或定时器值大于 TACCR0 的值)时，定时器复位并从 0 开始重新计数。



中断标志位的设置过程。当定时器的值等于 TACCR0 的值时，设置标志位 CCIFG0(捕获比较中断标志)为 1，而当定时器从 TACCR0 计数到 0 时，设置标志位 TAIFG（定时器溢出标志）为 1。



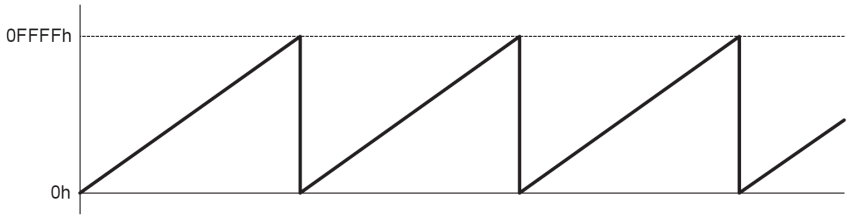
计数过程中还可以通过改变 TACCR0 的值来重置计数周期。

当新周期大于旧周期时，定时器会直接增计数到新周期。

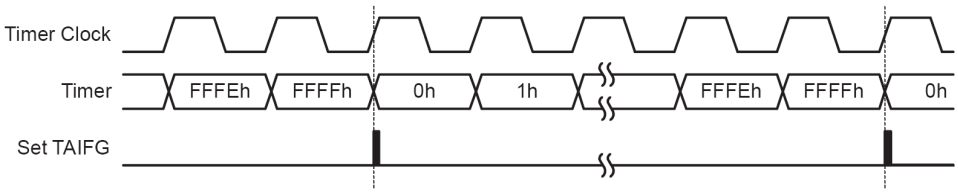
当新周期小于旧周期时，改变 TACCR0 时的定时器时钟相位会影响定时器响应新周期的情况。时钟为高时改变 TACCR0 的值，则定时器会在下一个时钟周期上升沿返回到 0，如果时钟周期为低时改变 TACCR0 的值，则定时器接受新周期并在返回到 0 之前，继续增加一个时钟周期。

[3] 连续计数模式

在需要 65536 个时钟周期的定时应用场合常用连续计数模式。定时器从当前值计数到 0xFFFF 后，又从 0 开始重新计数。



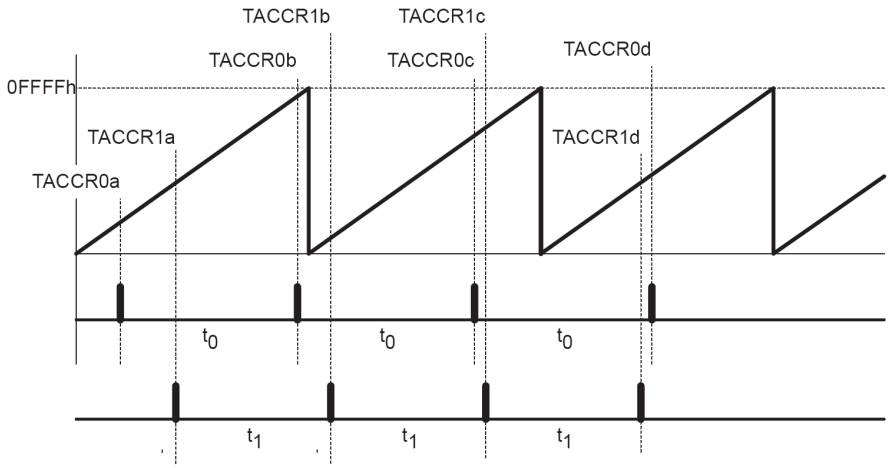
当定时器从 0xFFFF 计数到 0 时，设置标志位 TAIFG。



连续计数模式的典型应用：

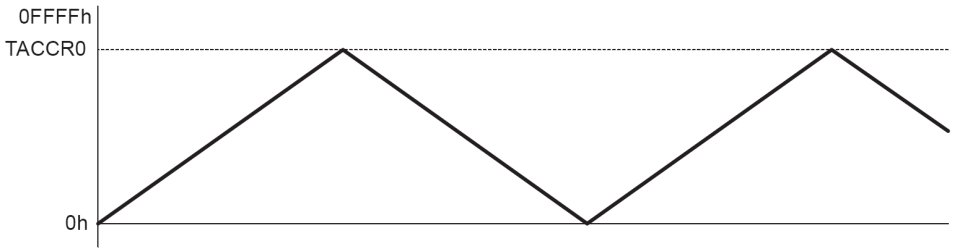
- **n** 产生多个独立的时序信号。利用捕获比较寄存器捕获各种其他外部事件发生的定时器数据。
- **n** 产生多个定时信号。通过中断处理程序在相应的比较寄存器 TACCRx 上加上一个时间差来实现。这个时间差是当前时刻（即相应的 TACCRx 中的值）到下一次中断发生时刻所经历的时间。

连续计数模式中产生多个定时信号：

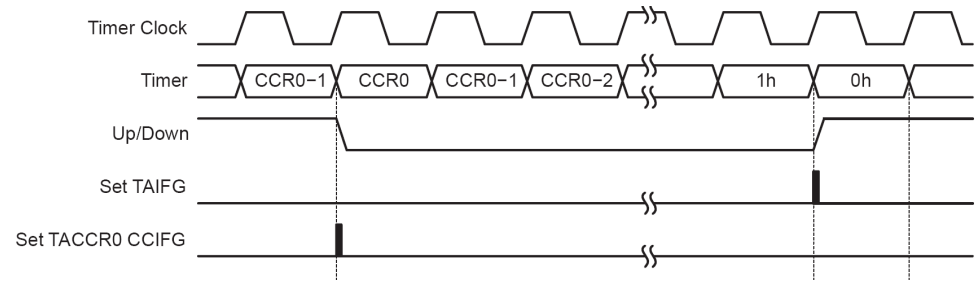


[4] 增/减计数模式

需要生成对称波形的情况经常可以使用增/减计数模式，该模式下定时器先增计数模式到 TACCR0 的值，然后反向减计数到 0。计数周期仍由 TACCR0 定义，它是 TACCR0 计数器数值的 2 倍。增/减计数模式时计数器中数值的变化情况如图。



标志为的设置情况如图，定时器 TAR 的值从 TACCR0-1 计数到 TACCR0 时，中断标志 CCIFG0 置位；当定时器从 1 减计数到 0 时，中断标志 TAIFG 置位。



在增/减计数模式过程中，也可以通过改变 TACCR0 的值来重置计数周期。在增计数阶段，新周期要在减计数完成（计数到 0）后才开始由效。

7.5 16 位定时器 A 模块捕获/比较工作原理

TIMER_A 有多个相同的捕获/比较模块，为实时处理提供灵活的手段，每个模块都有可用于捕获事件发生的时间或产生定时间隔。当发生捕获事件发生或定时时间到达时将引起中断。

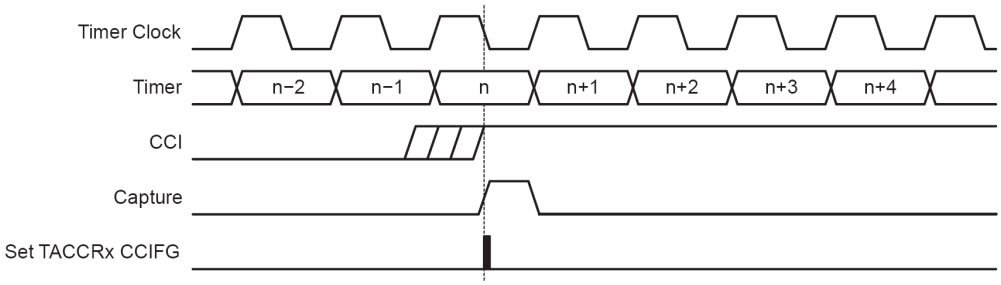
捕获/比较寄存器与定时器总线相连：可在满足捕获条件时，将 TAR 的值写入捕获寄存器；可在 TAR 的值与比较器值相等时，设置标志位。通过 TACCTLx 中的 CAP 选择模式，该模块既可用于捕获模式，也可用于比较模式。用 CMx 选择捕获条件，可以禁止捕获，上升沿捕获，下降沿捕获或者上下沿都捕获。可用 CCISx 选择捕获的输入信号源，输入信号可以来自外部引脚，也可来自内部信号，还可暂存在一个触发器中由 SCCI 信号输出。

[1] 捕获模式

当 TACCTLx 中的 CAP=1，该模块工作在捕获模式，这时如果在选定的引脚上发生设定的脉冲触发沿（上升沿，下降沿或任意跳变），则 TAR 中的值将写到 TACCRx 中。

每个捕获/比较寄存器都能被软件用于时间标记，可用于各种目的。例如，测量软件程序所用时间，测量硬件事件间的时间，测量系统频率。

当捕获完成后，中断标志位 CCIFG 被置位。如果总的中断允许位 GIE 允许，相应的中断允许位 CCIE 也允许，则将产生中断请求。



[2] 比较模式

比较方式主要用于为软件或应用硬件产生定时，还可为 D/A 转换功能或者马达控制等各种用途产生脉宽调制(PWM)输出信号。独立的输出模块被分配给各个捕获/比较寄存器的每一个，输出模块可以独立运行于比较功能，或以各种方式触发。

当 TACCTLx 中的 CAP＝0，该模块工作在比较模式。这时与捕获有关的硬件停止工作，在计数器 TAR 中计数值等于比较器中的值时设置标志位，产生中断请求；也可结合输出单元产生所需要的信号。

3 个捕获/比较器在比较模式时设置 EQUx 信号有差别：

当 TAR 的值大于等于 TACCR0 中的数值时，EQU0=1；

当 TAR 的值等于相应的 TACCR1 或 TACCR2 的值时，EQU1=1 或 EQU2＝1。

7.6 16 位定时器 A 模块输出工作原理

每个捕获/比较模块都包含一个输出单元，用于产生输出信号。每个输出单元有 8 种工作模式，可产生基于 EQUx 的多种信号。输出单元的结构及时序如图。

最终的输出信号源于一个 D 触发器。该触发器的数据输入源于输出控制模块，输出控制模块又分 3 个输入信号（EQU0，EQU1，EQU2 和 OUTx）经模式控制位 OMx0,OMx1 和 OMx2 运算后输出到 D 触发器。D 触发器的置位端和复位端也都将影响到最终的输出。D 触发器的时钟信号为定时器的时钟，在时钟为低电平时采样 EQU0 和 EQU1，EQU2，在时钟的下一个上升沿锁存入 D 触发器中。

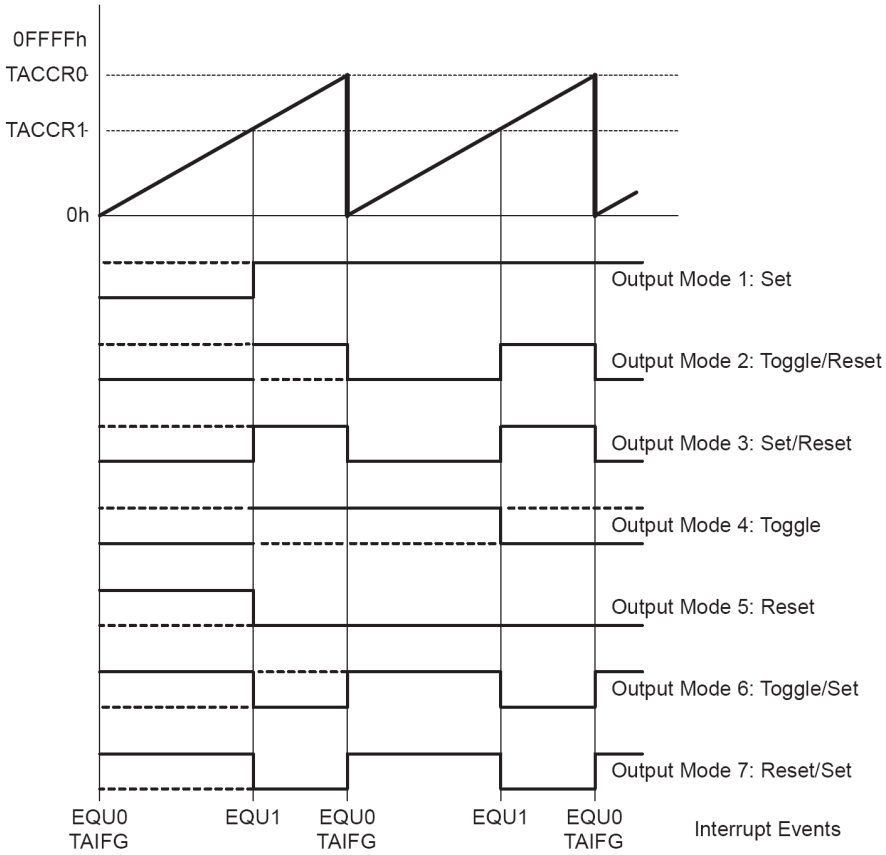
输出模式由模式控制位 OMx0，OMxc1 及 OMx2 决定，共有 8 种输出模式，分别对应 OMx2，OMx1 及 OMx0 的值。除模式 0 外，其他的输出都在定时时钟上升沿发生变化。输出模式 2、3、6、7 不适合输出单元 0，因为 EQUx=EQU0。

- 输出模式 0　输出模式：
  输出信号 OUTx 由每个捕获/比较模块的控制寄存器 TACCTLx 中的 OUTx 位定义，
  并在写入该寄存器后立即更新，最终位 OUTx 直通。
- 输出模式 1　置位模式：
  输出信号在 TAR 等于 TACCRx 时置位，
  并保持置位到定时器复位或选择另一种输出模式为止。
- 输出模式 2　PWM 翻转/复位模式：
  输出在 TAR 的值等于 TACCRx 时翻转，
  当 TAR 的值等于 TACCR0 时复位。
- 输出模式 3　PWM 置位/复位模式：
  输出在 TAR 的值的等于 TACCRx 时置位，
  当 TAR 的值等于 TACCR0 时复位。
- 输出模式 4　翻转模式：
  输出电平在 TAR 的值等于 TACCRx 时翻转，
  输出周期时定时器周期的 2 倍。
- 输出模式 5　复位模式：
  输出在 TAR 的值等于 TACCRx 时复位，
  并保持低电平知道选择另一种输出模式。
- 输出模式 6　PWM 翻转/置位模式：
  输出电平在 TAR 的值等于 TACCRx 时翻转，
  当 TAR 值等于 TACCR0 时置位。
- 输出模式 7　PWM 复位/置位模式：
  输出电平在 TAR 的值等于 TACCRx 时复位，
  当 TAR 的值等于 TACCR0 时置位。

输出单元在控制位的控制下，有 8 种输出模式输出信号。这些模式与 TAR，TACCRx，TACCR0 的值有关。在增计数模式下，当 TAR 增加到 TACCRx 或从 TACCR0 计数到 0 时，OUTx 信号按选择的输出模式发生变化。

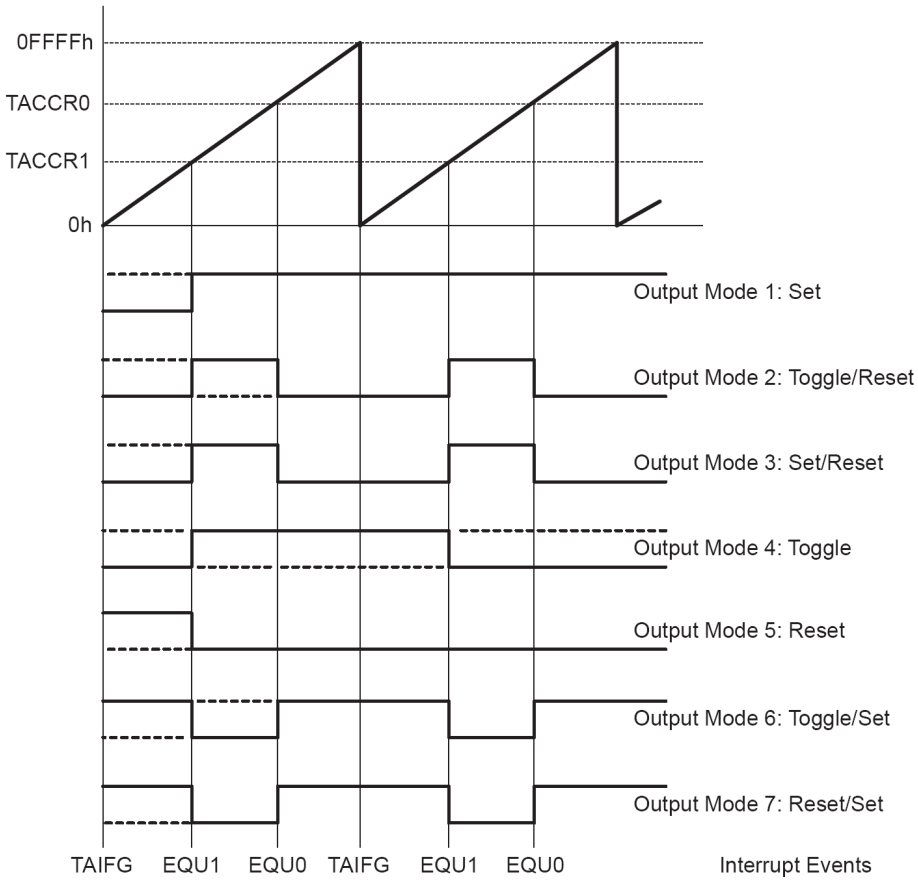
连续计数模式下的输出波形与增计数模式一样，只是计数器在增计数到 TACCR0 后还要继续计数到 0xFFFF，这样就延长了计数器计数到 TACCR1 的数值后的时间。这是与增计数模式不同的地方。

连续计数模式下的输出波形：



在增/减计数模式下的输出实例如图，这时的各种输出波形与定时器增计数模式或连续计数模式不同。当定时器在任意计数方向等于 TACCRx 时，OUTx 信号都按选择的输出模式发生改变。

在增/减计数模式下的输出实例：



7.7 16 位定时器 A 模块头文件定义

```
/************************************************************
* Timer A3
************************************************************/

#define TAIV_            (0x012E)  /* Timer A Interrupt Vector Word */
READ_ONLY DEFW( TAIV         , TAIV_)
#define TACTL_           (0x0160)  /* Timer A Control */
DEFW(   TACTL          , TACTL_)
#define TACCTL0_         (0x0162)  /* Timer A Capture/Compare Control 0 */
DEFW(   TACCTL0        , TACCTL0_)
#define TACCTL1_         (0x0164)  /* Timer A Capture/Compare Control 1 */
DEFW(   TACCTL1        , TACCTL1_)
#define TACCTL2_         (0x0166)  /* Timer A Capture/Compare Control 2 */
DEFW(   TACCTL2        , TACCTL2_)
#define TAR_             (0x0170)  /* Timer A */
DEFW(   TAR            , TAR_)
#define TACCR0_          (0x0172)  /* Timer A Capture/Compare 0 */
DEFW(   TACCR0         , TACCR0_)
#define TACCR1_          (0x0174)  /* Timer A Capture/Compare 1 */
DEFW(   TACCR1         , TACCR1_)
#define TACCR2_          (0x0176)  /* Timer A Capture/Compare 2 */
DEFW(   TACCR2         , TACCR2_)

/* Alternate register names */
#define CCTL0            TACCTL0   /* Timer A Capture/Compare Control 0 */
#define CCTL1            TACCTL1   /* Timer A Capture/Compare Control 1 */
#define CCTL2            TACCTL2   /* Timer A Capture/Compare Control 2 */
#define CCR0             TACCR0    /* Timer A Capture/Compare 0 */
#define CCR1             TACCR1    /* Timer A Capture/Compare 1 */
#define CCR2             TACCR2    /* Timer A Capture/Compare 2 */
#define CCTL0_           TACCTL0_  /* Timer A Capture/Compare Control 0 */
#define CCTL1_           TACCTL1_  /* Timer A Capture/Compare Control 1 */
#define CCTL2_           TACCTL2_  /* Timer A Capture/Compare Control 2 */
#define CCR0_            TACCR0_   /* Timer A Capture/Compare 0 */
#define CCR1_            TACCR1_   /* Timer A Capture/Compare 1 */
#define CCR2_            TACCR2_   /* Timer A Capture/Compare 2 */

#define TASSEL2          (0x0400)  /* unused */       /* to distinguish from USART SSELx */
#define TASSEL1          (0x0200)  /* Timer A clock source select 0 */
#define TASSEL0          (0x0100)  /* Timer A clock source select 1 */
#define ID1              (0x0080)  /* Timer A clock input devider 1 */
#define ID0              (0x0040)  /* Timer A clock input devider 0 */
#define MC1              (0x0020)  /* Timer A mode control 1 */
#define MC0              (0x0010)  /* Timer A mode control 0 */
#define TACLR            (0x0004)  /* Timer A counter clear */
#define TAIE             (0x0002)  /* Timer A counter interrupt enable */
#define TAIFG            (0x0001)  /* Timer A counter interrupt flag */

#define MC_0             (0*0x10u)  /* Timer A mode control: 0 - Stop */
#define MC_1             (1*0x10u)  /* Timer A mode control: 1 - Up to CCR0 */
#define MC_2             (2*0x10u)  /* Timer A mode control: 2 - Continous up */
#define MC_3             (3*0x10u)  /* Timer A mode control: 3 - Up/Down */
#define ID_0             (0*0x40u)  /* Timer A input divider: 0 - /1 */
#define ID_1             (1*0x40u)  /* Timer A input divider: 1 - /2 */
#define ID_2             (2*0x40u)  /* Timer A input divider: 2 - /4 */
#define ID_3             (3*0x40u)  /* Timer A input divider: 3 - /8 */
#define TASSEL_0         (0*0x100u) /* Timer A clock source select: 0 - TACLK */
#define TASSEL_1         (1*0x100u) /* Timer A clock source select: 1 - ACLK  */
#define TASSEL_2         (2*0x100u) /* Timer A clock source select: 2 - SMCLK */
#define TASSEL_3         (3*0x100u) /* Timer A clock source select: 3 - INCLK */

#define CM1              (0x8000)  /* Capture mode 1 */
#define CM0              (0x4000)  /* Capture mode 0 */
#define CCIS1            (0x2000)  /* Capture input select 1 */
```

```
#define CCIS0            (0x1000)  /* Capture input select 0 */
#define SCS              (0x0800)  /* Capture sychronize */
#define SCCI             (0x0400)  /* Latched capture signal (read) */
#define CAP              (0x0100)  /* Capture mode: 1 /Compare mode : 0 */
#define OUTMOD2           (0x0080)  /* Output mode 2 */
#define OUTMOD1           (0x0040)  /* Output mode 1 */
#define OUTMOD0           (0x0020)  /* Output mode 0 */
#define CCIE             (0x0010)  /* Capture/compare interrupt enable */
#define CCI              (0x0008)  /* Capture input signal (read) */
#define OUT              (0x0004)  /* PWM Output signal if output mode 0 */
#define COV              (0x0002)  /* Capture/compare overflow flag */
#define CCIFG            (0x0001)  /* Capture/compare interrupt flag */

#define OUTMOD_0          (0*0x20u)  /* PWM output mode: 0 - output only */
#define OUTMOD_1          (1*0x20u)  /* PWM output mode: 1 - set */
#define OUTMOD_2          (2*0x20u)  /* PWM output mode: 2 - PWM toggle/reset */
#define OUTMOD_3          (3*0x20u)  /* PWM output mode: 3 - PWM set/reset */
#define OUTMOD_4          (4*0x20u)  /* PWM output mode: 4 - toggle */
#define OUTMOD_5          (5*0x20u)  /* PWM output mode: 5 - Reset */
#define OUTMOD_6          (6*0x20u)  /* PWM output mode: 6 - PWM toggle/set */
#define OUTMOD_7          (7*0x20u)  /* PWM output mode: 7 - PWM reset/set */
#define CCIS_0           (0*0x1000u) /* Capture input select: 0 - CCIxA */
#define CCIS_1           (1*0x1000u) /* Capture input select: 1 - CCIxB */
#define CCIS_2           (2*0x1000u) /* Capture input select: 2 - GND */
#define CCIS_3           (3*0x1000u) /* Capture input select: 3 - Vcc */
#define CM_0             (0*0x4000u) /* Capture mode: 0 - disabled */
#define CM_1             (1*0x4000u) /* Capture mode: 1 - pos. edge */
#define CM_2             (2*0x4000u) /* Capture mode: 1 - neg. edge */
#define CM_3             (3*0x4000u) /* Capture mode: 1 - both edges */
```

7.8 16 位定时器 A 模块示例程序

```
// TIMER_A 实现 PWM
// PWM 信号是一种具有固定周期和不同占空比的数字信号，如图 4-47 所示。
// 如果 TIMER_A 定时器的计数器工作在增计数方式，输出采用模式 7(复位，置位模式)，
// 则可利用寄存器 CCR0 控制 PWM 波形的周期，用某个寄存器 CCRx 控制占空比。
// 这样 TIMER_A 就可以产生出任意占空比的 PWM 波形，如图 4-48 所示。
// TIMER_A 实现 PWM 举例：设 ACLK=TACLK=LFXT1=32768Hz,MCLK=SMCLK=DCOCLK=32×ACLK=1.048576MHz，
// 利用 TIMER_A 输出周期为 512/32768=15.625ms,占空比分别为 75%和 25%的 PWM 矩形波。
// 实例代码如下：

#include <msp430x16x.h>

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;
  TACTL = TASSEL0 + TACLR;      // ACLK, 清除 TAR
  TACCR0 = 512 - 1;             // PWM 周期
  TACCTL1 = OUTMOD_7;
  TACCR1 = 384;                 // 占空比 384/512=0.75
  TACCTL2 = OUTMOD_7;
  TACCR2 = 128;                 // 占空比 128/512=0.25
  P1DIR |= 0x04;                // P1.2 输出
  P1SEL |= 0x04;                // P1.2 TA1
  P2DIR |= 0x01;                // P2.0 输出
  P2SEL |= 0x01;                // P2.0 TA2
  TACTL |= MC0;                 // TA 增计数模式

  While(1){_NOP();}
}

// 由程序知：P1.2>>CCR1>>75%PWM；P2.0>>CCR2>>25%PWM.周期为 15。625ms。
// 可以随时间任意变化的 PWM 信号的占空比，如图 4-49 所示，具体做法如下：
// a.保持 CCR0 值(周期不变)，b，改变 CCRx 值（改变占空比）。
```

第八章 HM 硬件乘法器

Hardware Multiplier

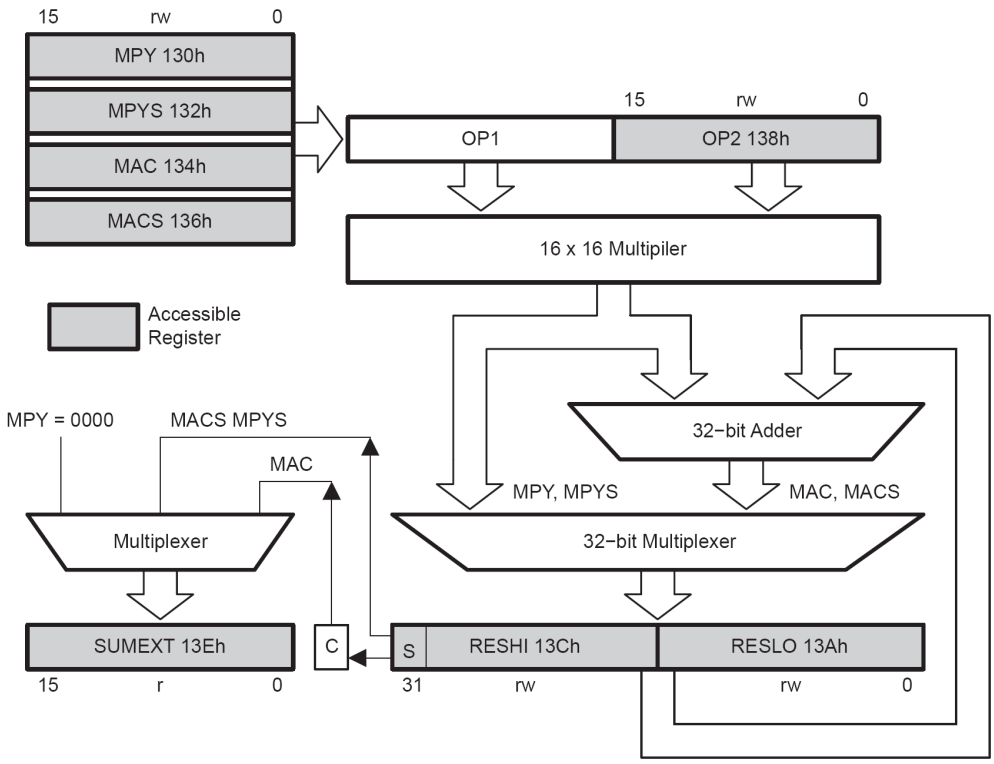
8.1 硬件乘法器概述

硬件乘法器是一个通过内部总线与 CPU 相连的 16 位外围模块。MSP430 单片机可以在部改变 CPU 结构和指令的情况下增加功能，这种结构特别适用于对运算速度要求很严格的情况。硬件乘法器大大提高了 MSP430 单片机的数据处理能力，其支持的运算如下：

- 无符号乘法（MPY）
- 有符号乘法（MPYS）
- 无符号乘加（MAC）
- 有符号乘加（MACS）
- 16×16 位、8×16 位、16×8 位、8×8 位。

8.2 硬件乘法器的结构

硬件乘法器的结构框图如下图所示。



n 操作数寄存器：

OP1（第 1 操作数）

OP2（第 2 操作数）

第一个操作数可源于 MPY、MPYS、MAC 和 MACS 四个寄存器，它们能确定乘法的类型。当第二个操作数写入后，相应的乘法操作立即执行，一般需 4 个周期。

n 结果寄存器：

RESHI（结果高字节寄存器）

RESLO（结果低字节寄存器）

SUMEXT（结果扩展寄存器）

寄存器 RESHI 和 RESLO 的内容为两个 16 位相乘的 32 位乘积结果。而寄存器 SUMEXT 的内容由执行的乘法模式及乘积结果决定。

下表说明了它们的关系：

寄存器	MPY	MPYS		MAC		MACS	
OP1	×	＋－	＋－	两数积+ACC ≤ 0xFFFFFFFF	两数积+ACC > 0xFFFFFFFF	两数积+ACC > 0x7FFFFFFF	两数积+ACC ≤ 0x7FFFFFFF
OP2	×	＋－	＋－				
SUMEXT	0	0x0000	0xFFFF	0000H	0001H	0FFFFH	0000H

8.3 硬件乘法器寄存器

[1] MPY 操作数 1
指示操作数为无符号数相乘
[2] MPYS 操作数 1
指示操作数为有符号数相乘
[3] MAC 操作数 1
指示操作数为无符号数累加
[4] MACS 操作数 1
指示操作数为有符号数累加
[5] OP_2 操作数 2
[6] RESLO 结果低字节寄存器
[7] RESHI 结果高字节寄存器
[8] SUMEXT 结果扩展寄存器
以上寄存器均是 16 位的字寄存器，在使用时，用户可以以字节操作或字操作，这样就形成了不同位数乘法的 4 种运算。

8.4 硬件乘法器运算的步骤及使用注意事项

使用硬件乘法器运算的步骤

- 写第一操作数 OP1，来源于 MPY、MPYS、MAC 和 MACS 四个寄存器。
- 写第二操作数 OP2，写入完毕，乘法运算开始进行。
- 保存结果，存放在 RESHI、RESLO 及 SUMEXT 中。

使用注意事项

- 第二个操作数写入完毕乘法就开始。一般在取出结果前插入 1~2 条指令，以保证足够的运算时间。
- 在一个单片机中只有一个硬件乘法器，若遇到多处使用，须在每次用完之后再进行下一次运算。
- 结果扩展寄存器 SUMEXT 的内容与运算类型及运算结果有关。
- 无论进行何种运算，只要操作数类型为 8×8 型，操作过程就要使用寄存器的绝对地址，而不能使用符号形式。寄存器 MPY、MPYS、MAC、MACS 和 OP2 的地址分别为：0130h , 0132h , 0134h , 0136h , 0138h。

8.5 硬件乘法器头文件定义

在 msp430x16x. h 中硬件乘法器定义如下所示：

```
/************************************************************
* HARDWARE MULTIPLIER
************************************************************/
#define MPY_              (0x0130)  /* Multiply Unsigned/Operand 1 */
DEFW(   MPY             , MPY_)
#define MPYS_             (0x0132)  /* Multiply Signed/Operand 1 */
DEFW(   MPYS            , MPYS_)
#define MAC_              (0x0134)  /* Multiply Unsigned and Accumulate/Operand 1 */
DEFW(   MAC             , MAC_)
#define MACS_            (0x0136)    /* Multiply Signed and Accumulate/Operand1 */
DEFW(   MACS            , MACS_)
#define OP2_              (0x0138)  /* Operand 2 */
DEFW(   OP2             , OP2_)
#define RESLO_            (0x013A)  /* Result Low Word */
DEFW(   RESLO           , RESLO_)
#define RESHI_            (0x013C)  /* Result High Word */
DEFW(   RESHI           , RESHI_)
#define SUMEXT_           (0x013E)  /* Sum Extend */
READ_ONLY DEFW( SUMEXT        , SUMEXT_)
```

用户可使用 MPY、MPYS、MAC、MACS 和 OP2,进行运算。

8.6 硬件乘法器示例程序

例一：　用硬件乘法器实现两无符号字符型数组相乘，结果放在另一无符号整型数组中。

说明：　Data1[7]和 Data2[7]　待乘数组。

Result[7]保存结果数组。

```
#include  "msp430x16x.h"

unsigned int  Result[7];
unsigned char Data1[7];
unsigned char Data2[7];

void main(void)
{
  unsigned char i;
  WDTCTL = WDTPW + WDTHOLD;     // 关看门狗
  for(i=0; i<7; i++)
  {
    Data1[i] = 10 * i;          // 对两数组赋值
    Data2[i] = 25 * i;
  }
  for(i=0; i<7; i++)
  {
    MPY = Data1[i];
    OP2 = Data2[i];
    _NOP();                  // 延迟
    _NOP();
    _NOP();
    Result[i] = RESLO;       // 保存结果，由于是 8×8 型，所以未用到 RESHI；
  }
}
```

例二： 用硬件乘法器实现两无符号整型数组相乘，结果放在另一无符号长整型数组中。

```
#include "msp430x16x.h"

unsigned long int Result[7];
unsigned int  Data1[7];
unsigned int  Data2[7];
unsigned char i;

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;
  for(i=0; i<7; i++)
  {
    Data1[i] = 250 * i;
    Data2[i]= 130 * i;
  }
  for(i=0; i<7; i++)
  {
    MPY = Data1[i];
    OP2 = Data2[i];
    _NOP();
    _NOP();
    _NOP();
    Result[i] = RESHI;
    Result[i] <<=16;
    Result[i] += RESLO;
  }
}
```

第九章 FLASH 存储模块

Flash Memory Module

9.1 FLASH 存储器模块概述

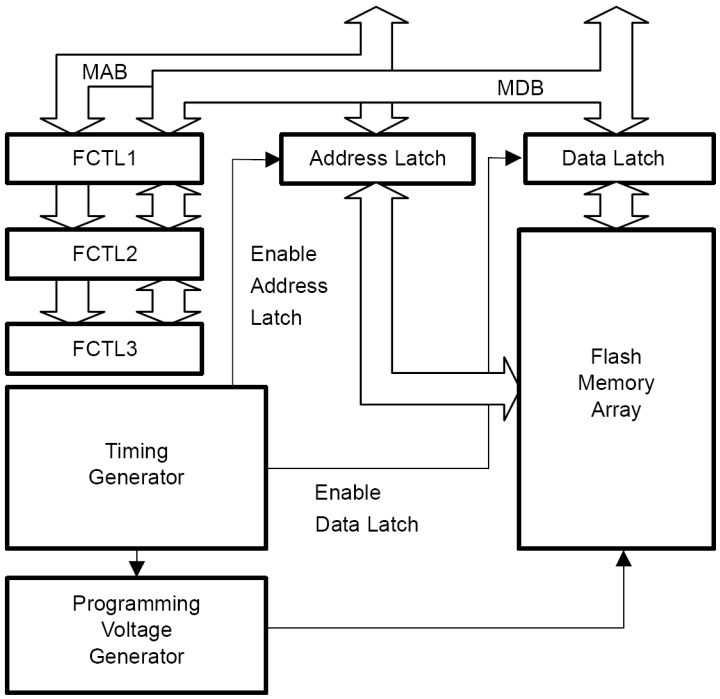
MSP430F169 所具有的嵌入式 FLASH 存储器同 EEPROM 一样是电可擦除的可编程存储器，它的主要特点是可以位、字节和字编程，也可以通过 JTAG，BSL，ISP 接口编程，2.7V~3.6V 编程电压，具有保密熔丝，和 60KB 空间 5 秒内编程的速度，擦除次数可达 10 万次。

MSP430 单片机的 FLASH 存储器的主要特点有：

- 编程可以使用位、字节和字操作；
- 可以通过 JTAG、BSL 和 ISP 进行编程；
- 1.8～3.6V 工作电压，2.7～3.6V 编程电压；
- 擦除/编程次数可达 100000 次；
- 数据保持时间从 10 年到 100 年不等；
- 60KB 空间编程时间<5s；
- 保密熔丝烧断后不可恢复，不能再用 JTAG 进行任何访问；
- FLASH 编程/擦除时间由内部硬件控制，无需任何软件干预。

9.2 FLASH 存储器结构

[1] FLASH 存储器原理框图



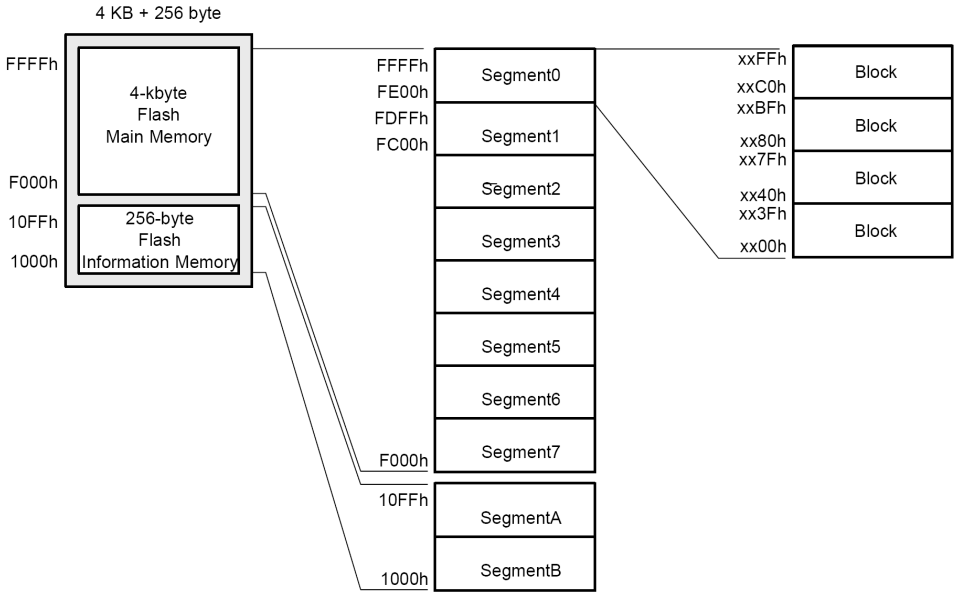
[2] FLASH 存储器结构

FLASH 存储器主要用作程序代码、数据表格以及用户信息的存储，可以多次擦除和写入数据。写入和擦除方式可通过 JTAG 接口，也可以由用户调用芯片内的驻留软件(bootstrap 装载器)来实现。

FLASH 存储器被分割为很多段。写进 FLASH 的单元可以是单独的位、字节或者字，但是擦除的最小单元是段（Segment）。

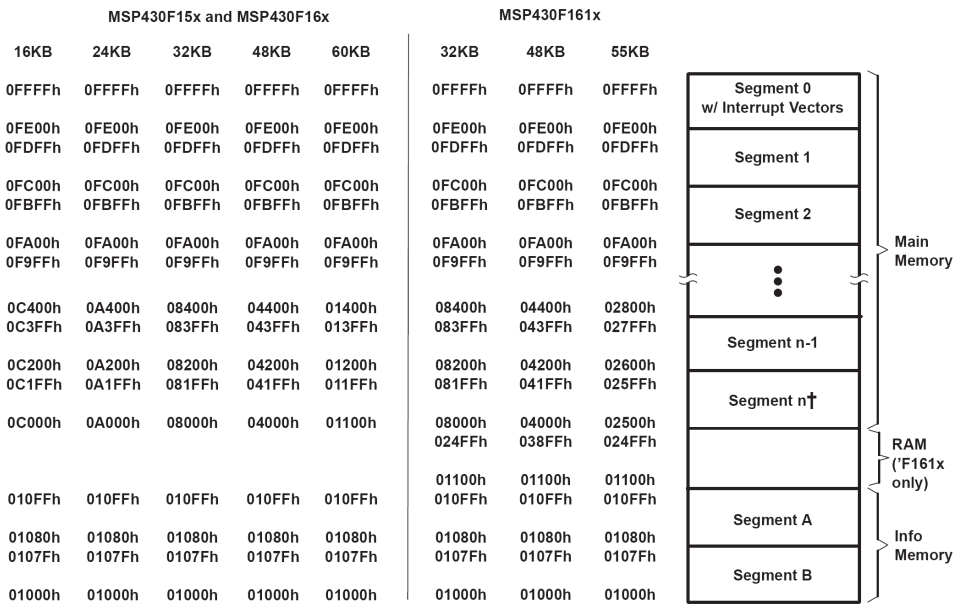
FLASH 存储器分为主存储器和信息存储器部分，在操作上二者没有区别，代码和数据可以放在任意一种存储器区域。二者的区别仅在于段的大小和他们的物理地址。信息存储器区域有 2 个 128 字节的段，主存储器区域有 2 个或多个 512 字节的段。

段又可以细分为很多块（Block），一块的大小为 64 字节，起始地址分别为 0xx00h、0xx40h、0xx80h、0xxC0h，结束地址分别为 0xx3Fh、0xx7Fh、0xxBFh、0xxFFh。



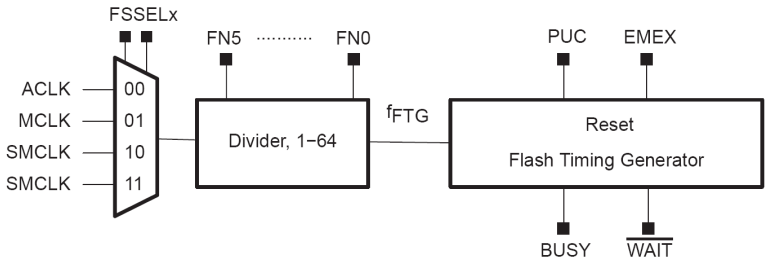
MSP430F169 单片机的 FLASH 地址分配图：

MSP430F169 的 FLASH 大小为 60K，FLASH 开始于 0x1100 结束于 0xFFFF，InfoA 为 0x1080~0x10FF，InfoB 为 0x1000~0x107F



[3] FLASH 存储器的时序发生器

FLASH 时序发生器的操作频率应该在 257KHz～476KHz 的范围内变化。FLASH 的读写操作由时序发生器来控制。



时序发生器通过 FSSELx 可取自 ACLK、SMCLK、MCLK 时钟源的信号。通过 FNx 位可将选中的时钟源进行分频以满足时序发生器的操作频率的要求。

9.3 FLASH 存储器的控制寄存器

FLASH 存储器的控制寄存器包括如下三组寄存器：

寄存器名称	寄存器缩写
FLASH 控制寄存器 1	FCTL1
FLASH 控制寄存器 2	FCTL2
FLASH 控制寄存器 3	FCTL3

[1] FCTL1　　FLASH 控制寄存器 1

15	14	13	12	11	10	9	8
FRKEY, Read as 096h FWKEY, Must be written as 0A5h							
7	6	5	4	3	2	1	0
BLKWRT	WRT	Reserved	Reserved	Reserved	MERAS	ERASE	Reserved

FRKEY：读操作安全键值

当对 FLASH 控制寄存器执行读操作的时候，需要对其高 8 位写入 0x96，

否则操作无效，发生非法访问，使 ACCVIFG 置位，进入出错中断。

FWKEY：写操作安全键值

当对 FLASH 控制寄存器执行读操作的时候，需要对其高 8 位写入 0xA5，

否则操作无效，发生非法访问，使 ACCVIFG 置位，进入出错中断。

BLKWRT：段编程控制位

如果有较多的连续数据要编程到某一段或某几段，则可选择这种方式，这样可缩短编程时间。

在一段程序完毕，再编程其他段时，需对该位先复位再置位，在下一条指令执行前 WAIT 位应为 1。

0：　未选用段编程方式

1：　选用段编程方式

WRT：编程控制位

0：　不编程，如对 FLASH 写操作，发生非法访问，使 ACCVIFG 置位。

1：　编程。

MERAS：主存储器控制擦除位

对 MSP430F169 来说主存储器控为由 0x1100~0Xffff 的 Main Memory。

0： 不擦除

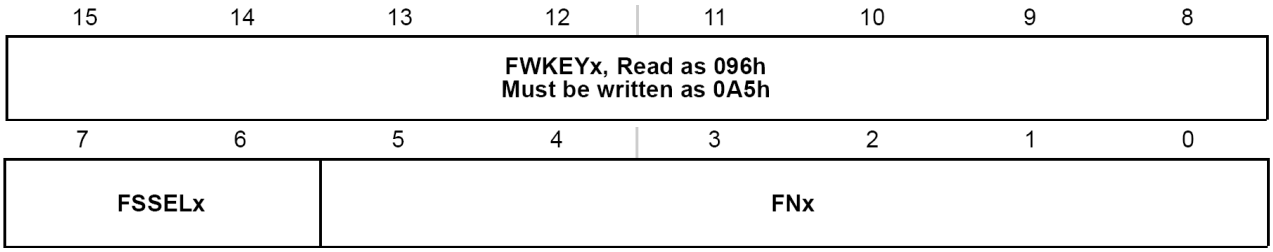
1： 擦除，设置后，对主存储器写的时候启动擦除操作，完成后该位自动复位。

ERASE：段擦除控制位

0： 不擦除

1： 擦除，设置后，对某段执行写操作时擦除该段，完成后该位自动复位。

[2] FCTL2 FLASH 控制寄存器 2



FSSELx：选择时钟源

0： ACLK

1： MCLK

2： SMCLK

3： SMCLK

FNx：分频系数选择位

分频系数为 FNx＋1。

0： 不分频

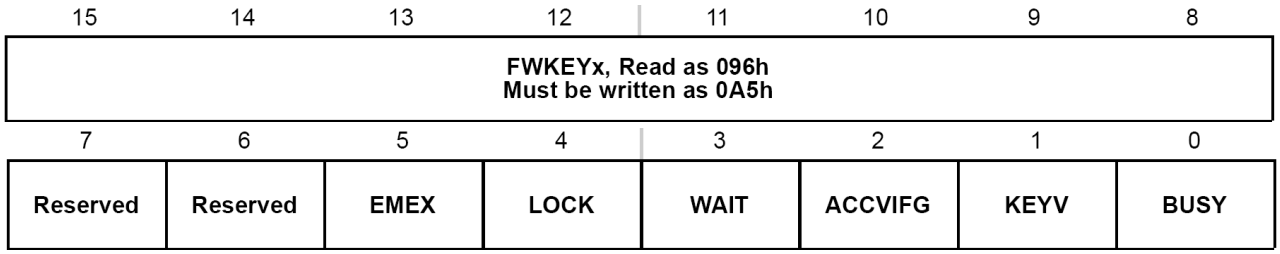
1： 2 分频

2： 3 分频

...

63：64 分频

[3] FCTL3 FLASH 控制寄存器 3



EMEX：紧急退出位

对 FLASH 的操作失控时使用该位作紧急处理。

0： 无作用

1： 立即停止对 FLASH 的操作

LOCK：锁定位

给已经编程好的 FLASH 存储器加锁。该位可由用户程序写入，也可由硬件自动设置。

可在编程、段擦除、主存擦除期间置位，置位后当前操作能正常结束。

在段擦除模式中，如果 BLKWRT＝1 且 WAIT＝1 时将 LOCK 位置位，则 BLKWRT 和 WAIT 都将复位，段编程模式结束。

如果在段编程模式中发生非法访问，则 ACCVIFG 和 LOCK 位将置位。

0： 不加锁，可以对 FLASH 进行读，写，擦除操作

1： 加锁，不能对 FLASH 进行读，写，擦除操作

WAIT：等待指示信号位（只读）

在段编程模式中指示 FLASH 存储器可以接受编程数据。如果 BLKWRT 位复位或 LOCK 位置位，则 WAIT 自动复位，段编程结束使得 WAIT 置位。在 BLKWRT＝1 时，每一次成功写入之后，BUSY 位被置位，指示其他的字或字节单元可以被编程操作。

0： 段编程操作已经开始，操作进行中

1： 段编程操作有效，当前数据已经正确的写入 FLASH，后续编程数据被列入计划

ACCVIFG：非法访问中断标志

当对 FLASH 阵列进行编程或擦除操作时不能访问 FLASH，否则将使得该位复位。如果非法访问中断允许同时总的中断也允许，则将执行 NMI 中断程序。

0： 没有对 FLASH 存储器的非法操作

1： 有对 FLASH 存储器的非法操作

KEYV：安全键值出错标志位

0： 对三个控制寄存器操作时 FWKEY 和 FRKEY 正确

1： 对三个控制寄存器操作时 FWKEY 和 FRKEY 错误

BUSY：忙标志位（只读）

该位表示 FLASH 模式现在的状态，是否正在对其操作，现在忙与否。

每次编程或擦除之前都应该检查 BUSY 位。

当编程或擦除启动以后，时序发生器将自动设置该位为 1，操作完成后，BUSY 位自动复位。

0： FLASH 存储器不忙

1： FLASH 存储器忙

9.4 FLASH 存储器的操作

FLASH 存储器的默认模式为读模式。在读模式，FLASH 不被擦除且不被写入，FLASH 的时序发生器和电压发生器被禁止，对存储器的操作等同于 ROM。

FLASH 存储器无需额外的外加电压而进行在系统编程。FLASH 的写入/擦除模式可通过 BLKWRT、WRT、MERS、ERASE 位来进行选择。在编程或者擦除期间不允许对 FLASH 进行读写操作。

[1] FLASH 存储器的擦除操作

对 FLASH 要写入数据，必须先擦除相应的段，对 FLASH 存储器的擦除必须是整段地进行，可以一段一段地擦除，也可以多段一起擦除，但不能一个字节一个字节或一个字一个字地擦除；擦除之后各位为 1。

擦除操作即是对相应地址范围内任意位置进行一次空写入，用以启动擦除操作。在空写入之后 BUSY 位被置位并且持续整个擦除周期。当擦除完成之后，BUSY、MERAS、ERASE 位将被自动清零。擦除周期地长短并不是由 FLASH 存储器的数量来决定，对整个 MSP430F169 来说，擦除周期是等同的。

在一个擦除周期之前必须关掉中断；在擦除操作完成之后，中断可以再次打开。任何发生在擦除周期内的中断将使相关位置位，在使能之后将产生一个中断请求。

对 FLASH 进行擦除操作的顺序如下：

- 选择适当的时钟源和分频因子，为时序发生器提供正确时钟输入；
- 如果 LOCK＝1，将其复位；
- 监视 BUSY 标志位，只有当 BUSY＝0 时才可以执行下一步，否则一直监视 BUSY 位；
- 如果擦除一段，则设置 ERASE＝1；
- 如果擦除多段，则设置 MERAS＝1；
- 如果整个 FLASH 全部擦除，则设置 ERASE＝1，MERAS＝1；
- 对擦除的地址范围内任意位置进行一次空写入，用以启动擦除操作；
  如果空写入的地址在不能执行擦除操作的段地址范围内，写入操作将不会起作用。

对 FLASH 的擦除操作要做如下 4 件事：

- 对 FLASH 控制寄存器写入适当的控制位；
- 监视 BUSY 位；
- 空写一次；
- 等待。

[2] FLASH 存储器的编程操作

对 FLASH 进行编程操作的顺序如下：

- 选择适当的时钟源和分频因子；
- 如果 LOCK＝1，将其复位；
- 监视 BUSY 标志位，只有当 BUSY＝0 时才可以执行下一步；
- 如果写入单字节或单字，则设置 WRT＝1；
- 如果是块写或多字、多字节顺序写入，则设置 WRT＝BLKWRT＝1；
- 将数据写入选定地址时启动时序发生器，在时序发生器的控制下完成整个过程。

块写操作在 64 字节的分界处需要特殊的软件支持，它们是如下操作：

- 等待 WAIT 位直到 WAIT＝1，表明最后一个字或字节写操作结束；
- 将控制位 BLKWRT 复位；
- 保持 BUSY 位为 1，直到编程电压撤离 FLASH 模块；
- 在新块被编程之前，等待 trcv (编程电压恢复时间)时间。

对 FLASH 的写入操作要做如下 4 件事：

- 对 FLASH 控制寄存器写入适当的控制位；
- 监视 BUSY 位；
- 写一个数据；
- 继续写直到结束。

[3] FLASH 存储器错误操作的处理

- 如果写入高字节口令码错误，将引发 PUC 信号；
- 在对 FLASH 操作期间读 FLASH 内容，将会引发 ACCFIG 状态位的设置；
- FLASH 操作需要较长的时间，可能引起看门狗定时器溢出(建议在进行 FLASH 操作之前先停掉看门狗，等操作完成再打开看门狗)；

9.5 FLASH 存储器模块头文件定义

```
/************************************************************
* Flash Memory
************************************************************/
/* Flash Memory 控制寄存器定义*/
#define FCTL1_            (0x0128)  /* FLASH Control 1 即 FCTL1 */
DEFW(   FCTL1           , FCTL1_)
#define FCTL2_            (0x012A)  /* FLASH Control 2 即 FCTL2 */
DEFW(   FCTL2           , FCTL2_)
#define FCTL3_            (0x012C)  /* FLASH Control 3 即 FCTL3 */
DEFW(   FCTL3           , FCTL3_)
/* 安全键值定义 */
#define FRKEY             (0x9600)  /* Flash key returned by read */
#define FWKEY             (0xA500)  /* Flash key for write */
#define FXKEY             (0x3300)  /* for use with XOR instruction */
/* 位定义 */
#define ERASE             (0x0002)  /* Enable bit for Flash segment erase */
#define MERAS             (0x0004)  /* Enable bit for Flash mass erase */
#define WRT               (0x0040)  /* Enable bit for Flash write */
#define BLKWRT            (0x0080)  /* Enable bit for Flash segment write */
#define SEGWRT            (0x0080)  /* 旧的定义 功能同 BLKWRT */
/* 分频定义 */
#define FN0               (0x0001)
/* Devide Flash clock by 1 to 64 using FN0 to FN5 according to: */
#define FN1               (0x0002)
/*  32*FN5 + 16*FN4 + 8*FN3 + 4*FN2 + 2*FN1 + FN0 + 1 */
#ifndef FN2
#define FN2               (0x0004)
#endif
#ifndef FN3
#define FN3               (0x0008)
#endif
#ifndef FN4
#define FN4               (0x0010)
#endif
#define FN5               (0x0020)
/* 时钟选择位定义 */
#define FSSEL0            (0x0040)  /* Flash clock select 0 */
#define FSSEL1            (0x0080)  /* Flash clock select 1 */
/*时钟选择功能定义*/
#define FSSEL_0           (0x0000)  /* Flash clock select: 0 - ACLK */
#define FSSEL_1           (0x0040)  /* Flash clock select: 1 - MCLK */
#define FSSEL_2           (0x0080)  /* Flash clock select: 2 - SMCLK */
#define FSSEL_3           (0x00C0)  /* Flash clock select: 3 - SMCLK */
/* 位定义 */
#define BUSY              (0x0001)  /* Flash busy: 1 */
#define KEYV              (0x0002)  /* Flash Key violation flag */
#define ACCVIFG           (0x0004)  /* Flash Access violation flag */
#define WAIT              (0x0008)  /* Wait flag for segment write */
#define LOCK              (0x0010)  /* Lock bit: 1 - Flash is locked (read only) */
#define EMEX              (0x0020)  /* Flash Emergency Exit */
```

9.6 FLASH 存储器模块示例程序

```
//信息段 A、B 的擦除和写入操作
#include <msp430x16x.h>
unsigned char value;            // 写入到信息段 A 的 8 位数据
// 函数声明
void write_SegA(unsigned char value);
void copy_A2B(void);
// 主函数
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;
  FCTL2 = FWKEY + FSSEL0 + FN0;         // 定义时钟
  value = 0;                            // 初始化 value=0
  while(1)
  {
    Write_SegA(value++);                // 写信息段 A
    copy_A2B();                         // 信息段 A 内容复制到信息段 B
  }
}

// 写信息段 A 函数
void write_SegA(unsigned char value)
{
  unsigned char *Flash_ptr;             // Flash 指针
  unsigned int i;
  Flash_ptr = (unsigned char *)0x1080;  // 初始化 Flash 指针
  FCTL1 = FWKEY + ERASE;                // 允许擦除
  FCTL3 = FWKEY;                        // 解锁
  *Flash_ptr = 0;                       // 空写，启动擦除
  FCTL1 = FWKEY + WRT;                  // 允许写
  for(i =0;i < 128;i ++)                // 循环写信息段 A 的 128 字节
  {
    *Flash_ptr++ = value;
  }
  FCTL1 = FWKEY;
  FCTL3 = FWKEY + LOCK;                 // 锁定
}

// 将信息段 A 内容拷备到信息段 B 函数
void copy_A2B(void)
{
  unsigned char *Flash_ptrA;
  unsigned char *Flash_ptrB;
  unsigned int i;
  Flash_ptrA = (unsigned char *)0x1080;     // 初始化信息段 A 指针
  Flash_ptrB = (unsigned char *)0x1000;     // 初始化信息段 B 指针
  FCTL1 = FWKEY + ERASE;
  FCTL3 = FWKEY;
  *Flash_ptrB = 0;
  FCTL1 = FWKEY + WRT;
  for(i = 0;i < 128;i ++)
  {
    *Flash_ptrB++ = *Flash_ptrA++;
  }
  FCTL1 = FWKEY;
  FCTL3 = FWKEY +LOCK;
}
```

第十章 CA 比较器 A 模块

Comparator A Module

10.1 比较器 A 模块概述

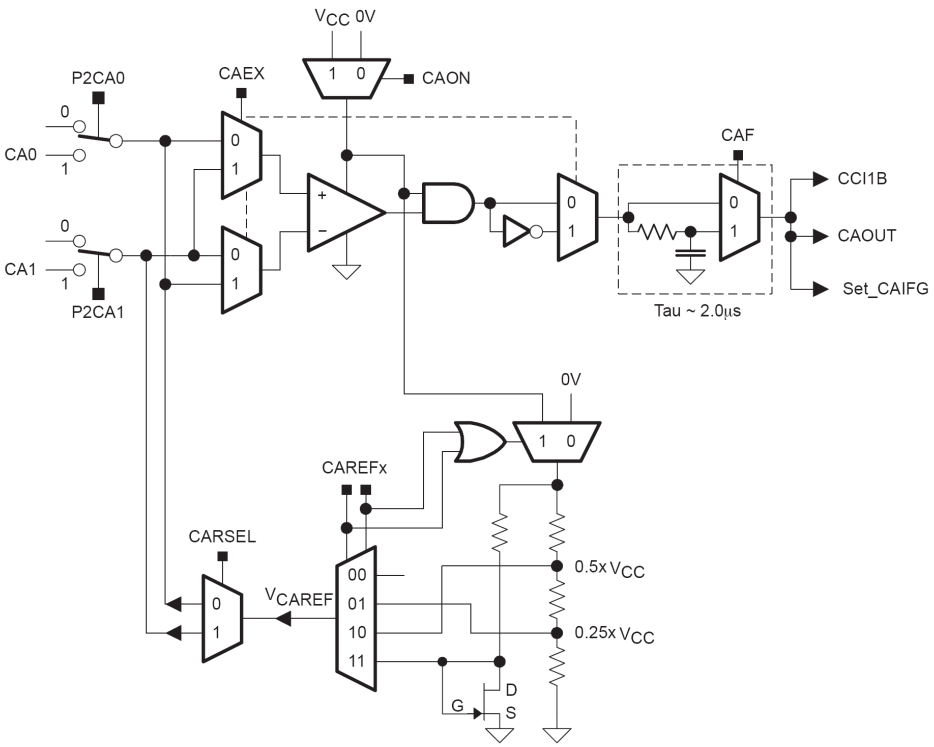
MSP430F169 含有比较器 A 模块，它是为精确比较测量而设计的，如电池电压监测，产生外部模拟信号，测量电流，电容和电阻，结合其他模块还可以实现精确的 A/D 模数转换功能。

比较器 A 模块支持精确的 A/D 转换、供电电压管理以及外部模拟信号的监管。
比较器 A 的特点包括：

- 同相端和反相端输入复用；
- 软件选择 RC 滤波器作为比较器输出；
- 输出作为 Timer_A 的捕获输入；
- 软件控制端口输入缓冲
- 具有中断能力
- 可选择参考电压发生器
- 比较器和参考电压发生器可以低功耗。

10.2 比较器 A 模块的结构

比较器 A 的原理框图



比较器 A 有两个模拟量输入端 CA0 和 CA1、一个模拟比较器、参考电压发生器和输出滤波器，还有一些控制单元。

比较器 A 的主要功能是指出两个输入电压 CA0 和 CA1 的大小关系，然后设置输出信号 CAOUT 的值。如果 CA0>CA1,则 CAOUT＝1；否则 CAOUT＝0。

[1] 模拟输入端

参与比较的两个模拟信号通过正、负两个输入电压端 CA0 和 CA1 进入比较器 A，输入电流极小。这两个输入端可由用户软件设置，最终能够选择 6 种信号(CA0、CA1、0.5Vcc、0.25Vcc、三极管阈值电压和外部参考源)，而且能够进行多种组合比较。P2CA0 和 P2CA1 将控制外部引脚连接至比较器 A 的情况。

[2] 参考电压发生器

参考电压发生器通过调节介入电阻来达到产生不同电压的目的，可以产生 4 种参考电压：0.5Vcc、0.25Vcc、三极管阈值电压和外部参考源。另外，比较器 A 可以适合于低功耗应用，该模块可以通过控制位 CAON 软件打开/关闭，不用时关闭以使电流消耗最小。

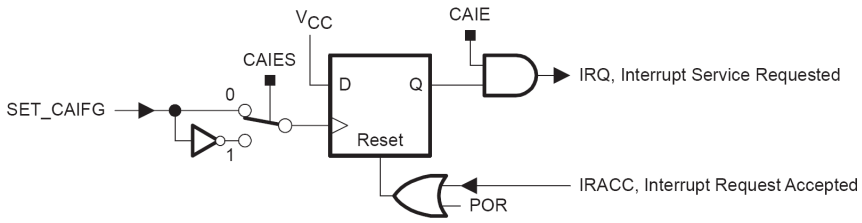
[3] 比较器

用于比较。其中与门负责将比较输出信号整形，控制位 CAEX 选择正向还是反相输出。

[4] 输出电路

CAF 是缓慢变化的输入电压稳定性控制位，可以将 RC 低通滤波器切换到比较器的输出端，可以消除比较输出信号的“毛刺”。最终输出信号的上升沿或下降沿可以设置为具有中断能力。如果不使用中断，可将输出信号送给内部其他模块，作为其他模块的一个输入信号；还可以由外部引脚引出。

[5] 比较器 A 的中断



中断电路如图所示。比较器 A 响应中断的条件为：

- 有中断源：比较器模块有比较结果输出；
- 设置中断标志：CAIES 选择比较器输出的上升沿或下降沿使中断标志 CAIFG 置位；
- 中断允许：比较器 A 的中断允许(CAIE 置位)、系统总中断允许(GIE 置位)。

中断响应后，硬件会自动清除中断标志位 CAIFG。

10.3 比较器 A 模块的寄存器

比较器 A 模块的寄存器有下列：

寄存器名称	寄存器缩写
比较器 A 控制寄存器 1	CACTL1
比较器 A 控制寄存器 2	CACTL2
比较器 A 端口禁止寄存器	CAPD

[1] CACTL1 比较器 A 控制寄存器 1

7	6	5	4	3	2	1	0
CAEX	CARSEL	CAREFx		CAON	CAIES	CAIE	CAIFG

CAEX：比较器的输入端，控制比较器 A 的输入信号和输出方向

0： 比较器的正输入端接 P2CA0，正输入端接 P2CA1，输出同相

1： 比较器的正输入端接 P2CA1，正输入端接 P2CA0，输出反相

CARESL：选择内部参考源加到比较器 A 的正端或负端

当 CAEX=0 时：

0： 选择内部参考源加到比较器 A 的正端

1： 选择内部参考源加到比较器 A 的负端

当 CAEX=1 时：

0： 选择内部参考源加到比较器 A 的负端

1： 选择内部参考源加到比较器 A 的正端

CAREFx：选择参考源

0： 内部参考电源关闭，使用外部参考源

1： 0.25 * Vcc

2： 0.50 * Vcc

3： 三极管阈值电压

CAON：比较器 A 开关控制

0： 关闭

1： 开启

CAIES：中断触发沿选择

0： 上升沿触发

1： 下降沿触发

CAIE：中断允许

0： 中断禁止

1： 中断允许

CAIFG：比较器中断标志

0： 没有中断请求

1： 有中断请求

[2] CACTL2 比较器 A 控制寄存器 2

7	6	5	4	3	2	1	0
Unused				P2CA1	P2CA0	CAF	CAOUT

P2CA1：控制输入端 CA1

0： P2CA1 不接 CA1

1： P2CA1 接 CA1

P2CA0：控制输入端 CA0

0： P2CA0 不接 CA0

1： P2CA0 接 CA0

CAF：选择比较器输出端是否经过 RC 低通滤波器

0： CAOUT 不经输出滤波

1： CAOUT 使用输出滤波

CAOUT：比较器 A 的输出

写操作无效

、

[3] CAPD 比较器 A 端口禁止寄存器

7	6	5	4	3	2	1	0
CAPD7	CAPD6	CAPD5	CAPD4	CAPD3	CAPD2	CAPD1	CAPD0

比较器 A 模块的输入输出与 I/O 共用引脚，CAPD 可以控制 I/O 端口输入缓冲器的通断开关。当输入电压不接近 Vss 或 Vcc 时，CMOS 型的输入缓冲器可以起到分流作用。

CAPDx：端口输入缓冲器控制

0： 输入缓冲使用

1： 输入缓冲禁止

10.4 比较器 A 模块头文件定义

```
/************************************************************
* Comparator A
************************************************************/
/* 比较器 A 控制寄存器定义 */
#define CACTL1_           (0x0059)  /* Comparator A Control 1 */
DEFC(   CACTL1          , CACTL1_)
#define CACTL2_           (0x005A)  /* Comparator A Control 2 */
DEFC(   CACTL2          , CACTL2_)
#define CAPD_             (0x005B)  /* Comparator A Port Disable */
DEFC(   CAPD            , CAPD_)
/* CACTL1 位定义 */
#define CAIFG             (0x01)    /* Comp. A Interrupt Flag */
#define CAIE              (0x02)    /* Comp. A Interrupt Enable */
#define CAIES             (0x04)    /* Comp. A Int. Edge Select: 0:rising / 1:falling */
#define CAON              (0x08)    /* Comp. A enable */
#define CAREF0            (0x10)    /* Comp. A Internal Reference Select 0 */
#define CAREF1            (0x20)    /* Comp. A Internal Reference Select 1 */
#define CARSEL            (0x40)    /* Comp. A Internal Reference Enable */
#define CAEX              (0x80)    /* Comp. A Exchange Inputs */
/* 参考电源选择 功能定义 */
#define CAREF_0           (0x00)    /* Comp. A Int. Ref. Select 0 : Off */
#define CAREF_1           (0x10)    /* Comp. A Int. Ref. Select 1 : 0.25*Vcc */
#define CAREF_2           (0x20)    /* Comp. A Int. Ref. Select 2 : 0.5*Vcc */
#define CAREF_3           (0x30)    /* Comp. A Int. Ref. Select 3 : Vt*/
/* CACTL2 位定义*/
#define CAOUT             (0x01)    /* Comp. A Output */
#define CAF               (0x02)    /* Comp. A Enable Output Filter */
#define P2CA0             (0x04)    /* Comp. A Connect External Signal to CA0 : 1 */
#define P2CA1             (0x08)    /* Comp. A Connect External Signal to CA1 : 1 */
#define CACTL24           (0x10)
#define CACTL25           (0x20)
#define CACTL26           (0x40)
#define CACTL27           (0x80)
/* CAPD 位定义*/
#define CAPD0             (0x01)    /* Comp. A Disable Input Buffer of Port Register .0 */
#define CAPD1             (0x02)    /* Comp. A Disable Input Buffer of Port Register .1 */
#define CAPD2             (0x04)    /* Comp. A Disable Input Buffer of Port Register .2 */
#define CAPD3             (0x08)    /* Comp. A Disable Input Buffer of Port Register .3 */
#define CAPD4             (0x10)    /* Comp. A Disable Input Buffer of Port Register .4 */
#define CAPD5             (0x20)    /* Comp. A Disable Input Buffer of Port Register .5 */
#define CAPD6             (0x40)    /* Comp. A Disable Input Buffer of Port Register .6 */
#define CAPD7             (0x80)    /* Comp. A Disable Input Buffer of Port Register .7 */
```

10.5 比较器 A 模块头示例程序

```
// 电压检测：P2.3 输入的未知电压接到比较器 A 的正端，片内参考电压 0.25Vcc 接到比较器 A 的负端，
// 如果未知电压大于 0.25Vcc，P1.0 置位，否则 P1.0 复位。

include <msp430x16x.h>

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;
  P1DIR |= 0x01;
  CACTL1 = CARSEL + CAREF0 +CAON;       // 0.25Vcc 连接到比较器的负端
  CACTL2 = P2CA0;                       // 外部引脚信号连接到比较器的正端
  while(1)
  {
    if((CAOUT & CACTL2))
    P1OUT |= 0x01;                      // CAOUT = 1，置位 P1.0
    else
    P1OUT &= ~0x01;                     // 否则复位
  }
}
```

第十一章 DMA 模块

Direct Memory Access Module

11.1 DMA 模块概述

目前，只有 MSP430F15X/16X 系列单片机具有 DMA 控制器，从而能够为数据高速传输提供保证，例如，通过 DMA 控制器可以直接将 DAC 转换存贮器的内容传到 RAM 单元。

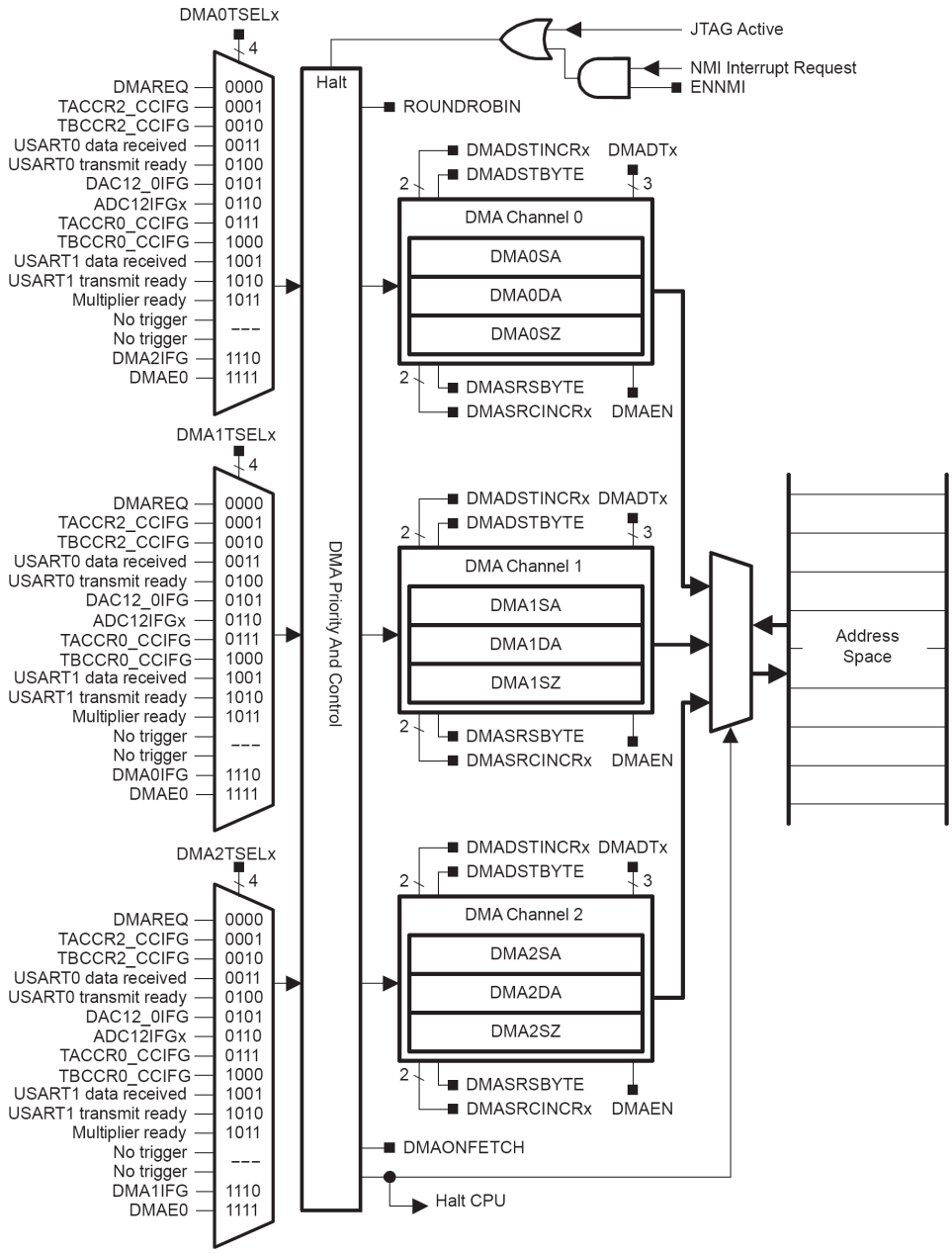
扩展的DMA具有来之所有外设的触发器，不需要CPU的干预即可提供先进的可配置的数据传输能力，从而加速了基于 MCU 的信号处理进程，DMA 传输的触发来源对 CPU 来说是完全透明的，DMA 控制器可在内存与外部及外部硬件之间进行精确的传输控制。DMA 消除了数据传输延迟时间以及各种开销，从而可以解放 16 为 RISC CPU，以便其将更多的时间用于处理数据，而非执行正在处理的任务。

11.2 DMA 控制器结构与功能

DMA 控制器具有如下特性：

- 数据传送不需要 CPU 介入，完全由 DMA 控制器自行管理。
- 在整个地址空间范围内传输数据，块方式传输可达 65556 字节。
- 能够提高片内外设数据吞吐能力，实现高速传输，每个字或者字节的传输仅需要 2 个 MCLK。
- 减少系统功耗，即使在片内外设进行数据输入获输出时，CPU 也可以处于超低功耗模式而不需要唤醒。
- 字节和字数据可以混合传送：DMA 传输可以是字节到字节，字到字，字节到字或者字到字。当字到字节传输时，只有字中较低字节能够传输，当从字节到字传输时，传输到字的低字节，高字节被自动清零。
- 4 种传输寻址模式：固定地址到固定地址、固定地址到块地址、块地址到固定地址以及块地址到块地址。
- 触发方式灵活：边沿或者电平触发。
- 单个、块或突发块传输模式：每次触发 DMA 操作，可以根据需要传输不同规模的数据。

DMA 结构原理图



DMA 控制器包含以下几个功能模块：

- 3 个独立的传输通道：通道 0，通道 1，通道 2。每个通道都有源地址寄存器，目的地址寄存器，传送数据长度寄存器和控制寄存器，每个通道的触发请求可以分别允许和禁止。
- 可配置的通道优先权：优先权裁决模块，传输通道的优先级可以调整，对同时有触发请求的通道进行优先级裁决，确定那个通道的优先级最高。MSP430 的 DMA 控制器可以采用固定优先级，还可以采用循环优先级。
- 程序命令控制模块：每个 DMA 通道开始传输之前，CPU 要编程给定相关的命令和模式控制，以决定 DMA 通道传输的类型。
- 可配置的传送触发器：触发源选择模块，DMAREQ（软件触发），TACCR2 输出，TBCCR2 输出，IIC 数据接收准备好，IIC 数据发送准备好，USART 接收发送数据，DAC12 模块 DAC12IFG，ADC12 模块 ADC12IFGx，DMAxIFG，DMAE0 外部触发源。并且还具有触发源扩充能力。

11.3 DMA 控制器相关操作

[1] 选择 DMA 触发源

每个 DMA 通道的初始化都是完全独立的，可以通过各自的控制位 DMAxTSELx 来选择自己的触发事件：当相应的触发源准备就绪，希望进行 DMA 请求。DMAxTSELx 只有在寄存器 DMACTLx 的控制位 DMAEN＝0 的时候才能被修改。

[2] 确定 DMA 触发信号方式

在单字或者单字节传输模式，每次 DMA 传输都需要单独的有效触发信号，在块传输或者突发块传输模式下，只需要一次触发信号就可以进行块或者突发块传输。

MSP430 的 DMA 控制器支持两种信号触发方式：

- 边沿触发方式：当控制位 DMALEVEL 复位时，触发信号的上升沿可以触发 DMA 操作。
- 电平触发方式：当控制位 DMALEVEL 置位时，为高电平触发分方式。

当控制位 DMALEVEL 和 DMAEN 被置位并且触发源信号也为高电平时，才可以触发 DMA 操作。

只有应用外部触发源 DMAE0 时，才需要采用电平触发方式。当使用电平触发方式，推荐操作模式选择控制位 DMADTx={0,1,2,3},因为在触发 DMA 操作后，DMAEN 位能够自动复位。当 DMALEVEL=1 时，如果当前进行的是块或者突发块传输，则 DMA 触发信号要在整个传输过程中保持高电平，如果在传输过程中，触发信号变为低电平，则 DMA 控制器停止传输，当触发信号恢复高电平或者 DMA 的寄存器由软件修改后，DMA 控制器才可以继续刚才的状态执行。

[3] DMA 控制器的寻址

DMA 控制器有 4 种寻址模式：

- 固定地址到固定地址
- 固定地址到块地址
- 块地址到固定地址
- 块地址到块地址

每个 DMA 控制器通道传输寻址模式的设置完全独立。

DMA 控制器的寻址模式可以通过寄存器 DMASRCINCR 和 DMADSTINCR 的控制位进行设置。在 DMA 传输时 DMASRCINCR 和 DMADSTINCR 分别为源地址和目的地址服务。

[4] 选择 DMA 传输模式

- DMA 有 6 种传输模式：
- 单字或者单字节传输。
- 块传输
- 突发块传输。
- 单字或者单字节传输。
- 重复块传输。
- 重复突发块传输。

每个 DMA 控制器的通道可以独立设置各自的传输模式，另外 DMA 传输模式和寻址方式是分别定义的，两者之间没有必然的关系。

11.4 DMA 控制器传输模式

[1] 单字或者单字节传输

当 DMA 通道被定义为单字或者单字节传输模式，每个字或者字节的传输都要党组触发。设置 DMADTx=0 就定义了单字或者单字节传输模式，规定的传输完毕后 DMAEN 位自动清除，如果需要再次传输，必须重新置位 DMAEN。如果设置 DMADTx＝4 为重复单字或者单字节传输模式，DMAEN 位一直保持置位，每次触发伴随一次传输。

DMAxSZ 寄存器保存传输的单元个数，如果该寄存器为 0，则没有传输。传输之前 DMAxSZ 寄存器的值写入到一个临时的寄存器中，每次操作之后 DMAxSZ 做减操作。当 DMAxSZ 减为零的时候，它所对应的临时寄存器将原来的值重新置入 DMAxSZ，同时相应的 DMAIFG 标志置位。

[2] 块传输模式

在块传输模式，每次触发可以传输一个数据块。设置 DMADTx=1 为块传输模式，每个数据块传输完毕，DMAEN 位自动清除，在触发传输下一个数据块之前，该位要被重新置位。在传输某个数据块期间，其他的传输请求将被忽略。设置 DMADTx=5 为重复块传输模式，某个数据块传输完毕，DMAEN 位仍然保持置位，之后，新的触发可以引起又一次数据块传送。

DMAxSZ 寄存器保存数据块所包含的单元个数。DMASRCINCR 和 DMADSTINCR 反映在数据块传输过程中的目的地址和源地址的变化情况。

在块传输或者重复块传输过程中，DMAxSA，DMAxDA，DMAxSZ 寄存器的值写入到对应的临时寄存器中，DMAxSA，DMAxDA 寄存器所对应的临时值在块传输过程中增加或者减少，而 DMAxSZ 在块传输过程中减计数，始终反映当前数据块还有多少单元没有传输完毕，当 DMAxSZ 减为 0，它所对应的临时寄存器将原来的值重新置入 DMAxSZ，同时相应的 DMAIFG 被置位。

在块传输过程中，CPU 暂停工作，不参与数据的传输。数据块需要 2×MCLK×DMAxSZ 个时钟周期。当每个数据块传输完毕，CPU 按照暂停前的状态重新开始执行。

第十二章 USART 串行通信模块（UART）

The Universal Synchronous / Asynchronous Receive / Transmit Module

12.1 异步串行通信模块概述

串口是系统与外界联系的重要手段，在嵌入式系统开发和应用中，经常需要上位机实现系统调试及现场数据的采集和控制。一般是通过上位机本身配置的串行口，通过串行通信技术，和嵌入式系统进行连接通信。

MSP430 系列的单片机可实现的串行通信功能：

- USART 硬件直接实现。
- 通过定时器软件实现。

片内具有 USART 模块的 MSP430 系列单片机，由于系列不同，片内可包含一个或多个 USART 块。USART 模块可以自动从任何一种低功耗模式 LPMx 开始自动工作。所有 USART0 和 USART1 都可以实现两种通信方式：USRT 异步通信和 SPI 同步通信。另外，MSP430F16X 系列单片机的 USART0 还可以实现 I^2C 通信。其中，UART 异步通信和 SPI 同步通信的硬件是通用的，经过适当的软件设计，这两种通信方式可以交替使用。

编者按：

MSP430F169 单片机的串行通信模块 USART 包括 3 个部分：

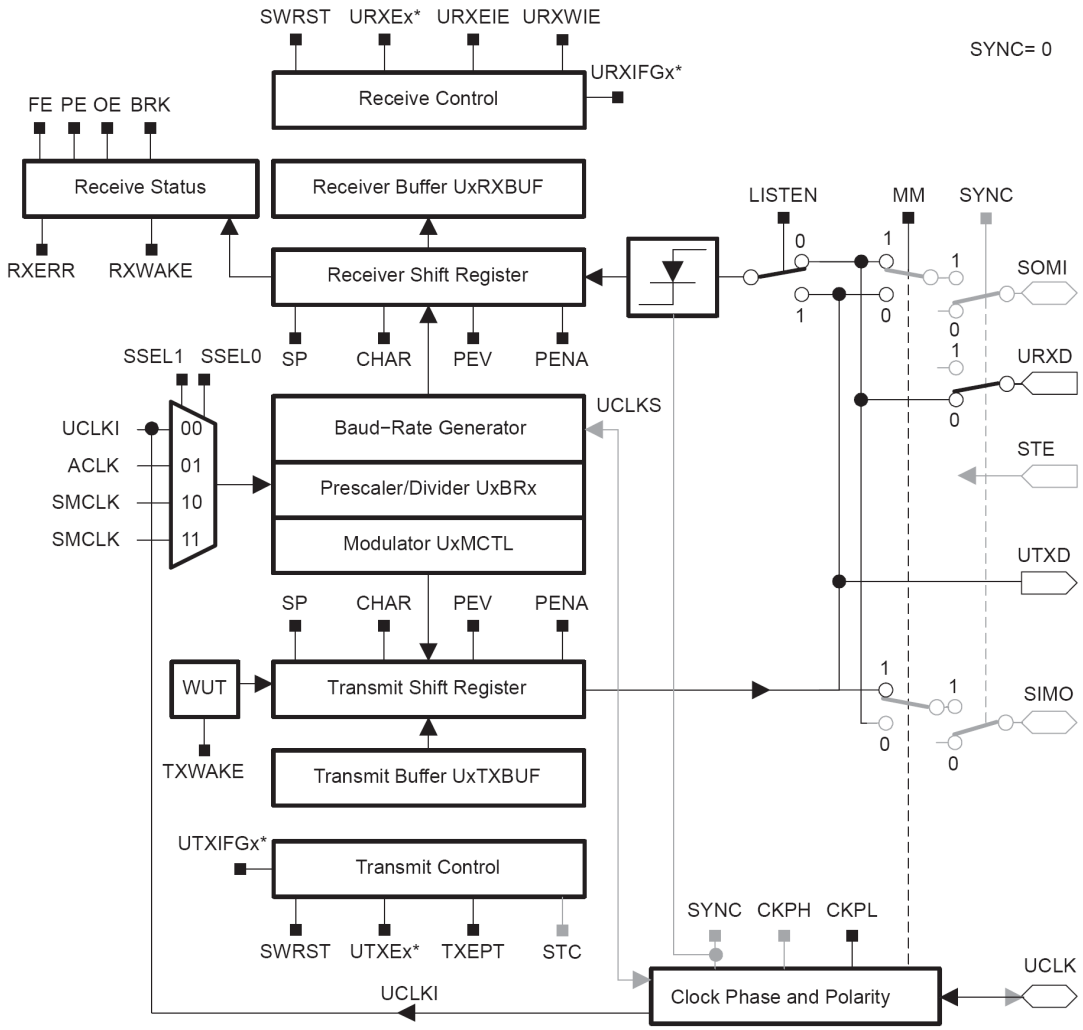
[1] UART 异步串行通信

[2] SPI 同步串行通信

[3] I^2C 同步串行通信

本章仅针对 UART 异步串行通信

12.2 异步串行通信模块原理图



** Refer to the device-specific datasheet for SFR locations*

USART 模块结构上图所示。该模块包含 4 个部分：

- 波特率部分：控制串行通信数据接收和发送的速度。
- 接收部分：接收串行输入的数据。
- 发送部分：发送串行输出的数据。
- 接口部分：完成并/串、串/并转换。

12.3 异步串行通信模块寄存器

硬件 USART 方式可实现串行通信时，允许 7 或 8 位串行位流以预先编程的速率或外部时钟确定的速率输入、输出 MSP430。用户对 USART 的使用是通过对硬件原理和通信协议理解下，在一系列寄存器设置之后，由硬件自动实现数据的输入输出。

MSP430 器件中有的型号有两个通信模块 USART0 和 USART1，因此它们有两套寄存器。

串口 0（USART0）的控制寄存器

Register	Short Form	Register Type	Address	Initial State
USART control register	U0CTL	Read/write	070h	001h with PUC
Transmit control register	U0TCTL	Read/write	071h	001h with PUC
Receive control register	U0RCTL	Read/write	072h	000h with PUC
Modulation control register	U0MCTL	Read/write	073h	Unchanged
Baud rate control register 0	U0BR0	Read/write	074h	Unchanged
Baud rate control register 1	U0BR1	Read/write	075h	Unchanged
Receive buffer register	U0RXBUF	Read	076h	Unchanged
Transmit buffer register	U0TXBUF	Read/write	077h	Unchanged
SFR module enable register 1†	ME1	Read/write	004h	000h with PUC
SFR interrupt enable register 1†	IE1	Read/write	000h	000h with PUC
SFR interrupt flag register 1†	IFG1	Read/write	002h	082h with PUC

† Does not apply to ’12xx devices. Refer to the register definitions for registers and bit positions for these devices.

串口 1（USART1）的控制寄存器

Register	Short Form	Register Type	Address	Initial State
USART control register	U1CTL	Read/write	078h	001h with PUC
Transmit control register	U1TCTL	Read/write	079h	001h with PUC
Receive control register	U1RCTL	Read/write	07Ah	000h with PUC
Modulation control register	U1MCTL	Read/write	07Bh	Unchanged
Baud rate control register 0	U1BR0	Read/write	07Ch	Unchanged
Baud rate control register 1	U1BR1	Read/write	07Dh	Unchanged
Receive buffer register	U1RXBUF	Read	07Eh	Unchanged
Transmit buffer register	U1TXBUF	Read/write	07Fh	Unchanged
SFR module enable register 2	ME2	Read/write	005h	000h with PUC
SFR interrupt enable register 2	IE2	Read/write	001h	000h with PUC
SFR interrupt flag register 2	IFG2	Read/write	003h	020h with PUC

编者按：两套串口相互独立，且寄存器结构相同，这里我们仅介绍一套串口，用 Ux 表示

[1] UxCTL 串口控制寄存器

7	6	5	4	3	2	1	0
PENA	PEV	SPB	CHAR	LISTEN	SYNC	MM	SWRST

PENA：校验允许位

0 校验禁止；

1 校验允许。

校验允许时，发送端发送校验，接收端接收该校验。地址位多机模式中，地址位包含校验操作。

PEV：奇偶位校验位，该位在校验允许时有效

0 奇校验；

1 偶校验。

SPB：停止位选择。决定发送的停止位数，但接收时接收器只检测 1 位停止位。

0 1 位停止位

1 2 位停止位。

CHAR：字符长度

0 7 位

1 8 位。

LISTEN：反馈选择。选择是否讲发送数据由内部反馈给接收器

0 无反馈

1 有反馈，发送信号由内部反馈给接收器。

SYNC：USART 模块的模式选择

0 UART 模式（异步）

1 SPI 模式（同步）。

MM：多机模式选择位

0 线路空闲多机协议；

1 地址多机协议。

SWRST：控制位

该位的状态影响着其他一些控制位和状态位的状态。在串行口的使用过程中，这一位是比较重要的控制位。一次正确的 USART 模块初始化应该是这样的顺序：先在 SWRST=1 情况下设置串行口；然后设置 SWRST=0；最后如果需要中断，则设置相应的中断使能。

[2] UxTCTL　　串口发送寄存器

7	6	5 4	3	2	1	0
Unused	CKPL	SSELx	URXSE	TXWAKE	Unused	TXEPT

CKPL：时钟极性控制位

0　UCLKI 信号与 UCLK 信号极性相同；

1　UCLKI 信号与 UCLK 信号极性相反；

SSELx：时钟源选择位

这两位确定波特率发生器的时钟源

0　外部时钟 UCLKI。

1　辅助时钟 ACLK。

2　子系统时钟 SMCLK。

3　子系统时钟 SMCLK。

URXSE：接收触发沿控制位

0　没有接收触发沿检测；

1　有接收触发沿检测。

TXWAKE：传输唤醒控制

0　下一个要传输的字符为数据

1　下一个要传输的字符为地址。

TXEPT：发送器空标志

在异步模式与同步模式时不一样

0　正在传输数据或者发送缓冲器（UTXBUF）有数据；

1　表示发送移位寄存器和 UTXBUF 空或者 SWRST=1。

[3] UxRCTL　串口接受寄存器

7	6	5	4	3	2	1	0
FE	PE	OE	BRK	URXEIE	URXWIE	RXWAKE	RXERR

FE：帧错误标志

0　没有帧错误；

1　帧错误

PE：校验错误标志位

0　校验正确；

1　校验错误。

OE：溢出标志位

0　无溢出

1　有溢出。

BRK：打断检测位

0　没有被打断；

1　被打断。

URXEIE：接收出错中断允许位

0　不允许中断，不接收出错字符并且不改变 URXIFG 标志位；

1　允许中断，出错字符接收并且能够置位 URXIFG。

URXWIE：接收唤醒中断允许位

当接收到地址字符时，该位能够置位 URXIFG，当 URXEIE=0,如果接收内容有错误，该位不能置位 URXIFG。

0　所有接收的字符能够置位 URXIFG;

1　只有接收到地址字符才能置位 URXIFG。

RXWAKE：接收唤醒检测位

在地址位多机模式，接收字符地址位置位时，该机被唤醒，在线路空闲多机模式，在接收到字符前检测到 URXD 线路空闲时，该为被唤醒，RXWAKE 置位。

0　没有被唤醒，接收到的字符是数据；

1　唤醒，接收的字符是地址。

RXERR：接收错误标志位

0　没有接收错误；

1　有接收错误。

[4] UxBR0 波特率控制寄存器 0

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

[5] UxBR1 波特率控制寄存器 1

7	6	5	4	3	2	1	0
2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8

UxBR0 和 UxBR1 两个寄存器用于存放波特率分频因子的整数部分。

其中 UXBR0 位低字节，UXBR1 为高字节。两字节和起来为一个 16 位字，成为 UBR。在异步通信时，UBR 的允许值不小于 3。如果 UBR<3，则接收和发送会发生不可预测的错误。

[6] UxMCTL 波特率调整寄存器

7	6	5	4	3	2	1	0
m7	m6	m5	m4	m3	m2	m1	m0

如果波特率发生器的输入频率 BRCLK 不是所需的波特率的整数倍，带有一小数，则整数部分写入 UBR 寄存器，小数部分由调整控制寄存器 UxCTL 的内容反映。波特率由以下公式计算：

波特率 = BRCLK / (UBR+ (M7+M6+..+M0) / 8)

其中 M0,M1,…M6 及 M7 为控制器 UxMCTL 中的各位。调整寄存器的 8 为分别对应 8 次分频，如果 M＝1，则相应次的分频增加一个时钟周期；如果 Mi＝0，则分频计数器不变。

[7] UxRXBUF 串口接受缓冲寄存器

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

接收缓存从接收移位寄存器最后接收的字符，可由用户访问。

当接收和控制条件为真时，接收缓存装入当前接收到的字符，如下表所列：

条件		结果			
URXEIE	URXWIE	装入 URXBUF	PE	PE	BRK
0	1	无差错地址字符	0	0	0
1	1	所有地址字符	X	X	X
0	0	无差错字符	0	0	0
1	0	所有字符	X	X	X

[8] UxTXBUF　串口发送缓冲寄存器

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

发送缓存内容可以传送至发送移位寄存器，然后有 UxTXDx 传输。对发送缓存进行写操作可以复位 UxTXIFGx。

[9] ME1　ME2　串口模块控制寄存器

ME1　　模块允许寄存器 1

7	6	5	4	3	2	1	0
UTXE0	URXE0						

ME2　　模块允许寄存器 2

7	6	5	4	3	2	1	0
		UTXE1	URXE1				

UTXE0：串口 0 的发送允许
URXE0：串口 0 的接收允许
UTXE1：串口 1 的发送允许
URXE1：串口 1 的接收允许

0　禁止
1　允许

[10]　IFG1　IFG2　串口中断标志控制寄存器

IFG1　中断标志寄存器 1

7	6	5	4	3	2	1	0
UTXIFG0	URXIFG0						

IFG2　中断标志寄存器 2

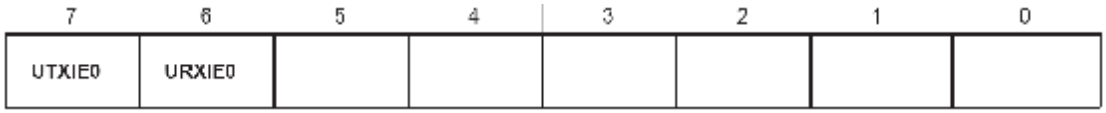
7	6	5	4	3	2	1	0
		UTXIFG1	URXIFG1				

UTXIFG0：串口 0 的发送中断标志
URXIFG0：串口 0 的接收中断标志
UTXIFG1：串口 1 的发送中断标志
URXIFG1：串口 1 的接收中断标志

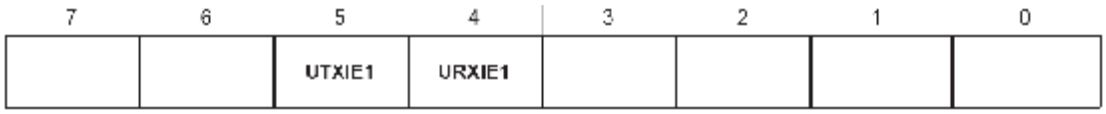
0　无中断请求标志
1　有中断请求标志

[11]　IE1　IE2　串口中断控制寄存器

IE1　中断控制寄存器 1



IE2　中断控制寄存器 2



UTXIE0：串口 0 的发送中断允许

URXIE0：串口 0 的接收中断允许

UTXIE1：串口 1 的发送中断允许

URXIE1：串口 1 的接收中断允许

0　中断请求禁止

1　中断请求允许

12.4 异步串行通信模块操作

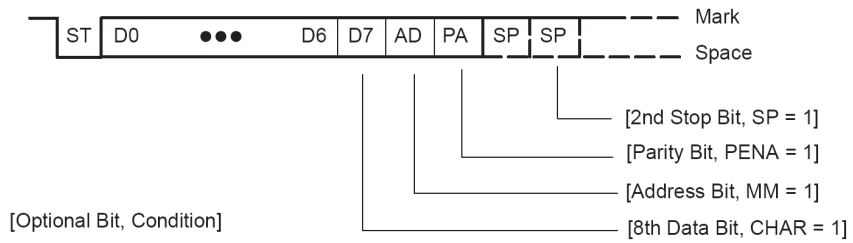
MSP430 串行异步通信模式通过两个引脚，即接收引脚 URXD 和发送引脚 UTXD 与外界相连。

异步通信的特点如下：

- 异步模式，包括线路空闲/地址位通信协议。
- 两个独立移位寄存器：输入移位寄存器和输出移位寄存器。
- 传输 7 位或 8 位数据，可采用奇校验或偶校验或者无校验。
- 从最低位开始的数据发送和接收。
- 可编程实现分频因子为整数或小数的波特率。
- 独立的发送和接收中断。
- 通过有效的起始位检测将 MSP430 从低功耗唤醒。
- 状态标志检测错误或者地址位。

[1] 异步通信字符格式

异步通信字符格式由 4 部分组成：起始位、数据位、奇偶校验位和停止位，如下图所示。其中用户可以通过软件设置数据位、停止位的位数，还可以设置奇偶位的有无。通过选择时钟源的波特率寄存器的数据来确定位周期。



接收操作以收到有效起始位开始。起始位由检测 URXD 端口的下降沿开始，然后以 3 此采样多数表决的方法取值。如果 3 次采样至少两次是 0 才表明是下降沿，然后开始接收初始化操作，这一过程实现错误起始位的拒收和帧中各数据的中心定位功能。

MSP430 可以处于低功耗模式，通过上述过程识别正确起始位后，MSP430 可以被唤醒，然后按照通用串口接口控可以制寄存器中设定的数据格式，开始接收数据，直到本帧采集完毕。

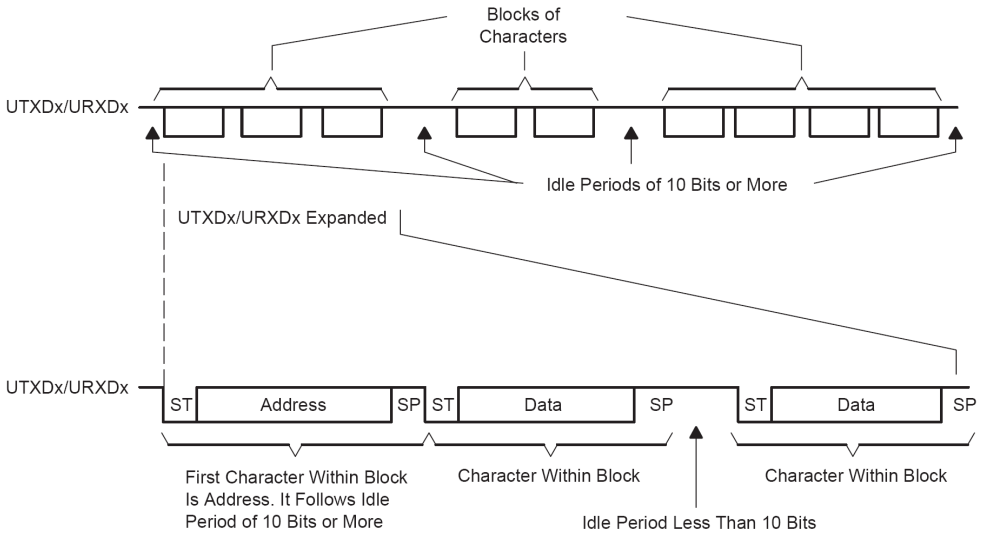
异步模式下，传送数据是以字符为单位传送得。因为每个字符在起始位处被唤醒，所以传输时多个字符可以一个接一个地连续传送，也可以断续发送和接收。

[2] 异步多机通信模式

在异步模式下，USART 支持两种多机通信模式，即线路空闲和地址位多机模式。在信息以一个多帧数据快，从一个指定地源传送到一个或者多个目的位置。在同一个串行链路上，多个处理机之间可以用这些格式来交换信息，实现了在多处理器通信系统间的有效数据传输。它们也用于使系统的激活状态压缩最低，以节省电流消耗或处理器所用资源。控制寄存器的 MM 位用来确定这两种模式。这两种模式采用唤醒发送、地址特性和激活等功能。

[2-1] 线路空闲多机模式

在这种模式下，数据块被空闲时间分割。在字符的第一个停止位之后，收到 10 个以上的 1，则表示检测到接收线路空闲，如下图所示。



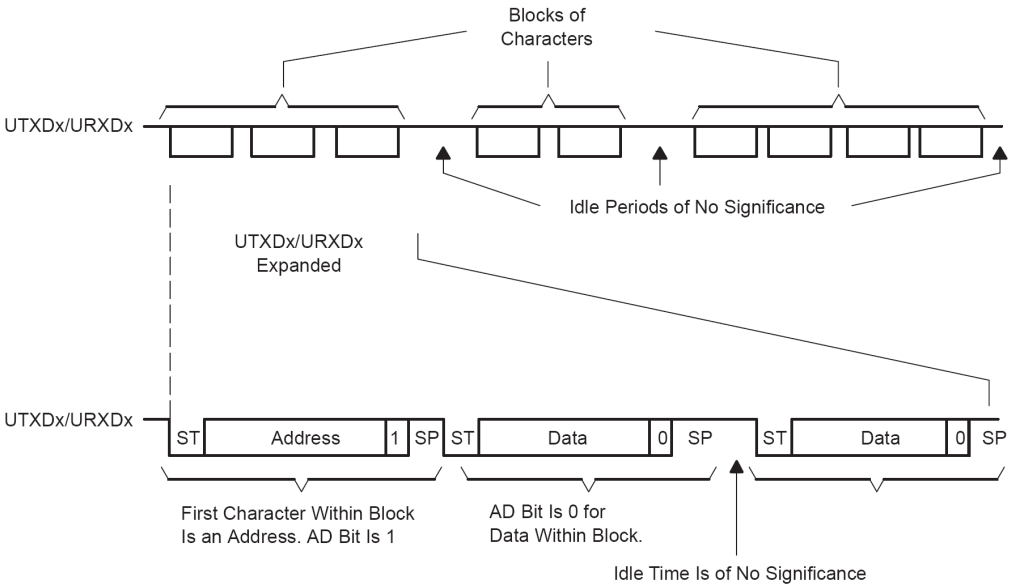
如果采用两位停止位，怎第二个停止位被认为空闲周期的第一个标志。空闲周期的第一个字符是地址字符。RXWAKE 位于可用于地址字符的标志。当接收到的字符是地址字符时，RXWAKE 被置为，并送入接收缓存。

用发送空闲帧来识别地址字符的步骤如下：

- TXWAKE=1，将任意数据写入 UTXBUF(UTXIFG=1)。当发送移位寄存器为空时，UTXBUF 的内容将被送入发送移位寄存器，同时 TXWAKE 的值移入 WUF。
- 如果此时 WUT=1，则将发送的起始位、数据位及校检位等被抑止，发送一个正好 11 位的空闲周期。
- 在地址字符识别空闲周期后移出串口的下一个数据是 TXWAKE 置位后写入 UTXBUF 中的第二个字符。当地址识别被发送后，写入 UTXBUF 中的第一个字符被抑制，并在以后被忽略。这时需随便往 UTXBUF 中写入一个字符，以便能将 TXWAKE 的值移入 WUF 中。

[2-2] 地址位多机模式

地址位多机模式的格式如下图所示。在这种模式下，字符包含一个附加的位作为地址标志。数据块的第一个字符带有一个置位的地址位，用以表明该字符是一个地址。当接收字符是地址时，RXWAKE 置位，并且将接收的字符送入接收缓存 URXBUF。



在 USART 的 URXWIE=1 时，数据字符在通常方式的接收器内拼装成字节，但他们不会被送入接收缓存，也不产生中断。只有当接收到一个地址位为 1 的字符时，接收器才被激活，接收到的字符被送往 URXBUF，同时 URXIFG 被置位。如果有错误，则相应的错误标志被设置。应用软件在判断后作出相应的处理。在地址位多机模式下，通过写 TXWAKE 位控制字符的地址位。每当字符由 UTXBUF 传送到发送器时，TXWAKE 位装入字符的地址位，再由 USART 将 TXWAKE 位清除。

[3] 串行操作自动错误检测

USART 模块接收字符时，能够自动进行校验错误、帧错误、溢出错误和打断状态检测。各种检测错误的含义标志如下：

- **n** FE：标志帧错误

  当一个接受字符的停止位为 0 并被装入接收缓存，则认为接收到一个错误桢。

- **n** PE：奇偶校验错误

  当然接收的 1 的个数与它的效验位不相符。

- **n** OE：溢出错误标志

  当然一个字节写入 UxRXBUF 时，前一个字符还没有被读出，则溢出。

- **n** BRK：打断检测标志

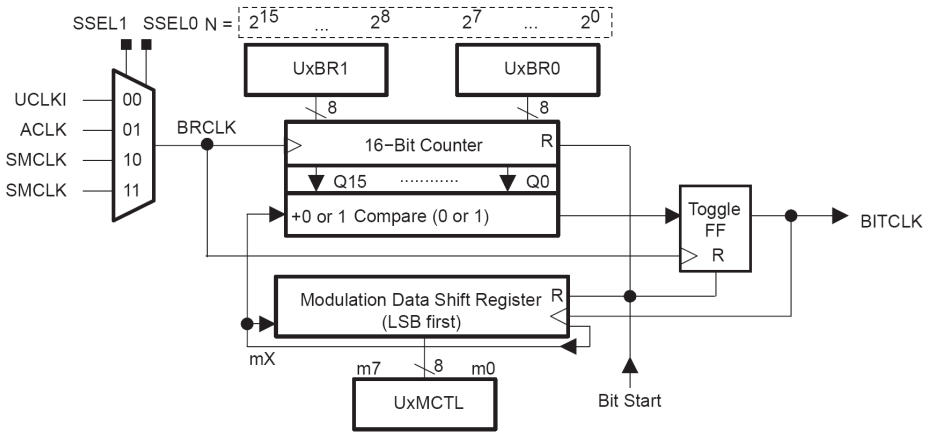
  当发生一次打断同时 UxRXEIE 置位，该位为 1。

当上述任何一种错误出现，位 RXERR 置位，表明有一个或者多个出错标志（FE、PE、OE 和 BRK 等）被置位。

如果 URXEIE=0 并且有错误被检测到，那么接收缓存不接收任何数据。如果 URXEIE=1，接收缓存接收字符，相应的错误检测标志置位，直到用户软件复位或者接收缓存内容被读出，才能复位。

[4] 波特率的产生

在异步通信中，波特率是很重要的指标，表示为每秒钟传送二进制数码的位数。波特率反映了异步串行通信的速度。波特率发生器产生同步信号表明各位的位置。波特率部分由时钟输入选择和分频、波特率发生器、调整器和波特率寄存器组成。串行通信时，数据接收和发送的速率就由这些构件控制。下图为其较为详细的结构。



整个模块的时钟源来自内部 3 个时钟或外部输入时钟，由 SSEL1 和 SSEL0 选择，以决定最终进入模块的时钟信号 BRCLK 的牝鹿。时钟信号 BRCLK 送入一个 15 位的分频器，通过一系列的硬件控制，最终输出两移位寄存器使用的移位时钟 BITCLK 信号。这个信号（BITCLK）的产生，由图的下半部分可以看出，是分频器在起作用。当计数器减技术到 0 时，输出触发器翻转，送给 BITCLK 信号周期的一半就是定时器（分频计数器）的定时时间。

下面介绍波特率的设置与计算。采集每位数据的时候，在每位数据的中间都要进行 3 次采样，以多数表决的原则进行数据标定与移位接收操纵，如此依次采集。由此看出，分频因子要么很大，要么是整数，否则，由于采集点的积累偏移，会导致每帧后面的几位数据采样点不在其数据的有效范围内。

MSP430 的波特率发生器使用一个分频计数器和一个调整器，能够用低时钟频率实现高速通信，从而在系统低功耗的情况下实现搞性能的串行通信。使用分频因子加调整的方法可以实现每一帧额各位有不同的分频因子，从而保证每个数据钟的 3 个采样状态都处于有效的范围内。

分频因子 N 由分频计数器的时钟（BRCLK）的频率和所需的波特率来决定：

N= BRCLK /波特率

如果使用常用的波特率与常用晶体产生的 BRCLK ，则一般得不到整数的 N,还有小数部分。分频计数器实现分频因子的整数部分，调整器使得小数部分尽可能准确。分频因子定义如下：

N 为目标分频因子：UBR 为 UXBR0 中的 16 位数据值：MX 为调整寄存器（UXMTCL）中的各数据位。

波特率可由下式计算：

波特率 = BRCLK / N = BRCLK / (UBR + (M7 + M6 + … +M0) / 8)

例如：

BRCLK=32.768KHz，要产生 BITCLK=2400Hz，分频器的分频系数为 32768 / 2400 = 13.65。所以设置分频器的计数值为 13。接下来用调整寄存器的值来设置小数部分的 0.65。调整器是一个 8 位寄存器，其中每一位分别对应 8 次分批情况，如果对应位 0，则分频器按照设定的分频系数分频计数；如果对应位为 1，则分频器按照设定的分频系数加 1 分频计数。按照这个原则 0.65*8=5,也就是说，8 次分频计数过程中应有 5 次加 1 计数，3 次不加 1 计数。调整寄存器的数据由 5 个 1 和 3 个 0 组成。调整器的数据每 8 次周而复始循环使用，最低位最先调整，比如设置调整寄存器的值为 6BH（01101011），当然也可以设置其他值，必须有 5 个 1，而且 5 个 1 要相对分散。即分频器按顺序 13、14、14、13、14、13、14、14 来分频。在 8 位调整器调整位都使用后，再重复这一顺序。

[5] 常用波特率设置表

	Divide by		A: BRCLK = 32,768 Hz						B: BRCLK = 1,048,576 Hz				
Baud Rate	A:	B:	UxBR1	UxBR0	UxMCTL	Max. TX Error %	Max. RX Error %	Synchr. RX Error %	UxBR1	UxBR0	UxMCTL	Max. TX Error %	Max. RX Error %
1200	27.31	873.81	0	1B	03	−4/3	−4/3	±2	03	69	FF	0/0.3	±2
2400	13.65	436.91	0	0D	6B	−6/3	−6/3	±4	01	B4	FF	0/0.3	±2
4800	6.83	218.45	0	06	6F	−9/11	−9/11	±7	0	DA	55	0/0.4	±2
9600	3.41	109.23	0	03	4A	−21/12	−21/12	±15	0	6D	03	−0.4/1	±2
19,200		54.61							0	36	6B	−0.2/2	±2
38,400		27.31							0	1B	03	−4/3	±2
76,800		13.65							0	0D	6B	−6/3	±4
115,200		9.1							0	09	08	−5/7	±7

12.5 异步串行通信模块操作例程

[1] 初始化串口

初始化串口 0，允许接收和发送，允许接收中断，禁止发送中断，8bit 字符发送
时钟 ACLK = 32.768KHz，波特率 4800
注意：对串口寄存器操作的时候务必保证 SWRST=1，设置完成后 SWRST=0。

```
// 串口初始化函数
void InitUSART(void)
{
  U0CTL   |= CHAR;              // 8bit 字符
  U0TCTL  |= SSEL0;             // ACLK
  U0BR1    = 0x00               // 4800 波特率
  U0BR0    = 0x06
  U0MCTL   = 0x6f
  U0CTL   &= ~SWRST;            // 设置完成
  ME1     |= UTXE0 + URXE0;     // 接收发送允许
  IE1     |= URXIE0;            // 接受发送中断
  P3SEL   |= (0x80 + 0x40);     // 引脚切换到特殊功能上
  _EINT();   // 不要忘了开中断
}
```

[2] 串口发送函数

采用软件查询式发送，将 1 个字节写入发送寄存器，然后等待发送完成的标志。
本方法适合波特率较高的场合（大于 4800）

```
// 所涉及的全局变量
unsigned char TBuff[8];             // 发送缓冲区
unsigned char RBuff[8];             // 接受缓冲区
unsigned char Flag_Receive = 0;     // 标志: 接受到有效数据

// 串口发送函数(不需要开发送中断)发送一个数组(共 8 个字节)
void USART_Send(unsigned char *pData)
{
  unsigned char i;
  for(i=0; i<8; i++)
  {
    TXBUF0 = pData[i];              // 装入发送寄存器
    while ((IFG1 & UTXIFG0) == 0);  // 判断:发送是否完成
    IFG1 &= ~(UTXIFG0);
  }
}
```

[3] 串口接受函数

在 RAM 开辟接收缓冲区，等到接收的数据组满足要求或者为一桢数据时，才处理。
中断中接收。

```
// 所涉及的全局变量
unsigned char TBuff[8];             // 发送缓冲区
unsigned char RBuff[8];             // 接受缓冲区
unsigned char Flag_Receive = 0;     // 标志: 接受到有效数据

// 串口接受函数(需要开接受中断)
#pragma vector=UART0RX_VECTOR
__interrupt void USART0_RXIRQ (void)
{
  unsigned char Temp;
  // 暂存接受数据
  Temp = RXBUF0;
  // 8 字节接受缓冲队列
  RBuff[0] = RBuff[1];
  RBuff[1] = RBuff[2];
  RBuff[2] = RBuff[3];
  RBuff[3] = RBuff[4];
  RBuff[4] = RBuff[5];
  RBuff[5] = RBuff[6];
  RBuff[6] = RBuff[7];
  RBuff[7] = Temp;
  // 判断数据有效(有效则设置标志位，等待处理)
  if (RBuff[0] == 0xd0){Flag_Receive = 1; return; }else{Flag_Receive = 0;}
}
```

[4] 串口校验 CRC16 函数

常用函数，使用的 CRCKey = 0xA001
输入一个数组和长度，则计算出该部分的 CRC 值。

```
// CRC-16 循环冗余效验函数
unsigned int Caculate_CRC16(unsigned char *DAT, unsigned char Lenth)
{
  // *DAT 指向要计算 CRC 的数组, Lenth 为数据的有效长度
  unsigned int CRC = 0xffff;    // CRC 的初始值为 FFFF
  unsigned char i;
  unsigned char j;
  for(i=0; i<Lenth; i++)
  {
    CRC = CRC ^ DAT[i];         // 和当前字节异或一次
    for(j=0; j<8; j++)          // 循环 8 次
    {
      if(CRC & 0x01)            // 判断最低位，如果为 1
      {
        CRC = CRC >> 1;         // 右移一位
        CRC = CRC ^ 0xA001;     // 和多相式异或
      }
      else                      // 判断最低位，如果为 0
      {
        CRC = CRC >> 1;         // 右移一位
      }
    }
  }
  return(CRC);
}
```

第十三章 ADC12 模数转换模块

12bit A / D Conversion Module

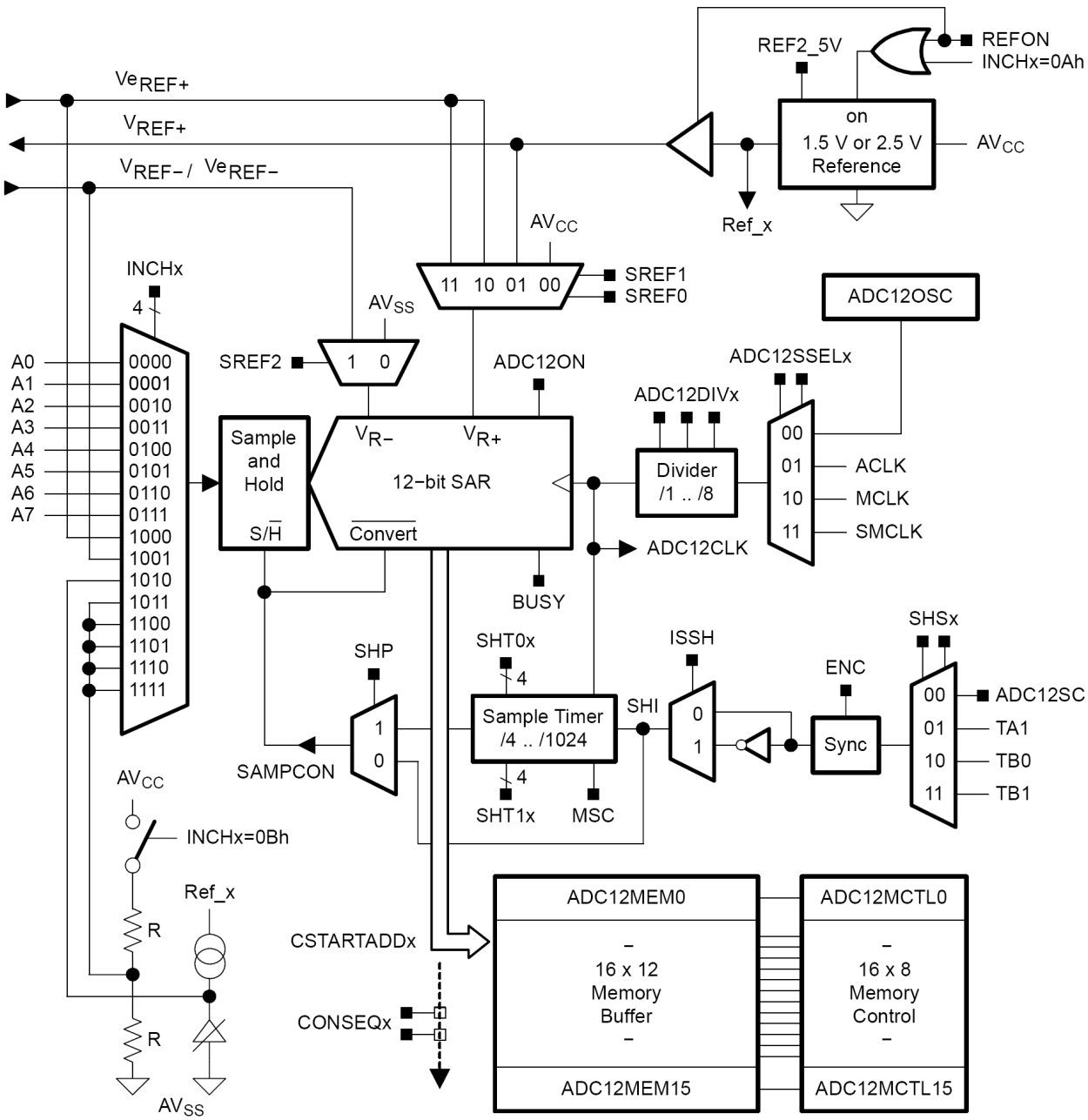
13.1 模数转换概述

在 MSP430 的实时控制和智能仪表等应用系统中，控制或测量对象的有关变化量，往往是一些连续变化的量，如压力，温度，流量，速度等。利用传感器把各种物理量测量出来，转换为电信号，经过模数转换（Analog to Digital Conversion）转变成数字量，这样模拟量才能被 MSP430 处理和控制。
分析和设计 MSP430 的 ADC 相关应用时涉及到一些相关术语，ADC 模块常用的指标有如下几个：
[1]分辨率
分辨率表示输出数字量变化一个相邻的数码所需要出入的模拟电压的变化量。
他定义为转换器的满刻度电压与 2^n 的比值，其中 n 为 ADC 的位数。对于 MSP430 的 12 位 ADC 转换器，如果 Vref = 2.5V，则分辨率为 2.5V/4096=0.0006，即分辨率为 0.6mV。
[2]量化误差
量化误差和分辨率是统一的，量化误差是由于有限数字对模拟数值进行离散取值量化而引起的误差，因此量化误差理论上为一个单位分辨率，即±1/2LSB.
[3]转换精度
ＡＤＣ模块的转换精度反映了一个实际 ADC 模块在量化上与一个理想的 ADC 模块进行模数转换的差值，可表示为绝对误差或者相对误差，与一般的仪表的定义相似。
[4]转换时间
指 ADC 模块完成依次模数转换所需要的时间，转换时间越短越能适应高速度变化的信号，其和 ADC 结构位数有关。
[5]其他
此外还应该考虑电压范围，工作温度，接口特性，输出形式的性能。

本实验系统所使用的 MSP430F169 所带的 ADC 模块为 12 位模数转换模块 ADC12。

13.2 ADC12 结构

ADC12 模块能够实现 12 位精度的模数转换，ADC12 结构如图。



ADC12 主要包括以下几个功能模块：

[1]参考电压发生器

所有的 ADC 和 DAC 模块都需要一个基准信号，这个信号就是我们常说的 Vref+，Vref-。

MSP430 的 ADC12 模块内部带有参考电源，通过控制 REFON 信号来启动内部参考电源，并且通过 REF2_5V 控制内部参考电源产生 1.5V 或者 2.5V 的 Vref+。

最后给 ADC 模块转换器的参考电压 Vr+和 Vr-通过 SREF_x 设置 6 种组合方式：

Vr+可以在 AVcc（系统模拟电源），Vref+（内部参考电源），Veref+（外部输入的参考电源）之间选择，

Vr-可以在 AVss（系统模拟地），Vref-/Veref-（内部或外部参考电源）。

[2]模拟多路通道

ADC12 可以选择多个通道的模拟输入，但是 ADC12 只有一个转换内核，所以这里有用到了模拟多路通道，每次接通一个信号到 ADC 转换内核上。模拟多路通道包括了 8 路外部信号通道（A0~A7）和 4 路内部信号通道（Vref+，Vref-/Veref-，（AVcc-AVss）/2，片内温度传感器）

[3]具有采样保持功能的 12 位模数转换内核

转换内核是有一个采样保持器和一个转换器组成。由于 ADC 转换需要一定的时间，对高速变化的信号进行瞬时采样时，不等 ADC 转换完成，外部输入的信号就已经改变。所以在 ADC 转换器前加入了采样保持器，一旦 ADC 开始转换，采样保持器则进行保持，即使现场输入的信号的变化比较快，也不会影响到 ADC 的转换工作。

12 位的 ADC 转换器将 Vr+和 Vr-之间分割为 2^{12}（4096）等份，然后将输入的模拟信号进行转换，输出 0~4095 的数字。如果输入电压 Vin≤ Vr-则结果为 0，Vin≥Vr+结果为 4095。

[4]采样转换时续控制电路

这部分浩阔各种时钟信号，ADC12CLK 转换时钟，SAMPCON 采样转换信号，SHT 控制采样周期，SHS 控制采样触发来源，ADC12SSEL 选择内核时钟，ADC12DIV 时钟分频。

由图可以看出来 SAMPCON 信号高的时候采样，低的时候转换。而 SAMPCON 有 2 个来源，一来自采样定时器，另一路由用户自己控制，通过 SHP 选择。

[5]转换结果存储

ADC12 一共有 12 个转换通道，设置了 16 个转换存储器用于暂时存储转换结果，合理设置后，ADC12 硬件会自动将转换的结果保存到相应的存储器里。

ADC12 主要特点：

- 12 位转换精度，1 位非线形误差，1 位非线形积分误差
- 多种时钟源给 ADC12 模块，切本身自带时钟发生器
- 内置温度传感器
- TimerA/TimerB 硬件触发器
- 8 路外部通道和 4 路内部通道
- 内置参考电压源和 6 种参考电压组合
- 4 种模式的模数转换
- 16bit 的转换缓存
- ADC12 关闭支持超低功耗
- 采用速度快，最高 200Kbps
- 自动扫描
- DMA 使能

13.3 ADC12 寄存器

控制寄存器：

寄存器类型	寄存器缩写	寄存器含义
转换控制寄存器	ADC12CTL0	转换控制寄存器 0
	ADC12CTL1	转换控制寄存器 1
中断控制寄存器	ADC12IFG	中断标志寄存器
	ADC12IE	中断控制寄存器
	ADC12IV	中断向量寄存器
储存控制寄存器	ADC12MCTL0~ADC12MCTL15	通道控制寄存器
	ADC12MEM0~ADC12MEM15	通道储存寄存器

[1] ADC12CTL0　　转换控制寄存器 0

15	14	13	12	11	10	9	8
SHT1x				SHT0x			
7	6	5	4	3	2	1	0
MSC	REF2_5V	REFON	ADC12ON	ADC12OVIE	ADC12 TOVIE	ENC	ADC12SC

ADC12CTL0 是 ADC12 最重要的控制寄存器之一。其中灰色的寄存器只有当 ENC=0 时才能更改。

ADC12SC：采样转换控制位（和 SHP,ISSH,ENC 有关）

在 ENC=1，ISSH=0 的情况下：

SHP=1 时：ADC12SC 由 0 变 1 时，启动 A/D 转换，转换完成后 ADC12SC 自动复位

SHP=0 时：ADC12SC 高电平时采样，ADC12SC 复位围启动一次转换

其中 ENC=1 表示转换允许，ISSH 表示输入信号为同相输入信号，

SHP=1 表示采　样信号 SAMPCON 来自于采样定时器，

SHP=0 表示 SAMPCON 采样有 ADC12SC 直接控制。

注意：当软件启动一次 A/D 转换时，ADC12SC 和 ENC 要在一条语句内完成设置。

ENC：转换允许位

0　ADC12 为初始状态，不能启动 A/D 转换

1　首次转换由 SAMPCON 的上升沿启动

注意：

[1]在 CONSEQ=0（单通道单次转换）的情况下，当 ADC12BUSY=1 时，

ENC=0 则会结束转换进程，并且得到错误结果。

[2]在 CONSEQ≠0（非单通道单次转换）的情况下，当 ADC12BUSY=1 时，

ENC=0 则转换正常结束，得到正确结果

ADC12TVIE：转换时间溢出中断允许（多次采样请求）

当前转换还没有完成时，又得到一次采样请求，如果 ADC12TVIE 允许的话，会产生中断。

0　允许发生转换时间溢出产生中断

1　禁止发生转换时间溢出产生中断

ADC12OVIE：溢出中断允许（ADC12MEMx 多次写入）

当 ADC12MEMx 还没有被读出的时候，而又有新的数据要求写入 ADC12MEMx 时，如果允许则会产生中断

0　允许溢出中断

1　禁止溢出中断

ADC12ON：ADC12 内核控制

0　关闭 ADC12 内核实现低功耗

1　开启 ADC12 内核

REFON：内部基准电压发生器控制

0　关闭内部基准电压发生器

1　开启内部基准电压发生器

REF2_5V：内部基准电压选择 1.5V/2.5V

0　选择 1.5V 内部参考电压

1　选择 2.5V 内部参考电压

MSC：多次采样/转换控制位

当 SHP=1，CONSEQ≠0 时，MSC 位才能生效

0　每次转换需要 SHI 信号的上升沿出发采样定时器

1　首次转换需要 SHI 信号的上升沿出发采样定时器，以后每次转换在前一次转换结束后立即进行

SHT0x：0~7 通道的采样保持器时间控制

定义了 ADC12MEM0~7 中转换采样时序与采样时钟的关系

保持时间越短，采样速度越快，反映电压波动明显

$T_{sample}= 4\times T_{ADC12CLK}\times N$（N<13 时 $N = 2^n$，n>13 时，N=256）

SHT1x：8~15 通道的采样保持器时间控制

定义了 ADC12MEM8~15 中转换采样时序与采样时钟的关系

保持时间越短，采样速度越快，反映电压波动明显

$T_{sample}= 4\times T_{ADC12CLK}\times N$（N<13 时 $N = 2^n$，n>13 时，N=256）

[2] ADC12CTL1　　转换控制寄存器 1

15	14	13	12	11	10	9	8
CSTARTADDx				SHSx		SHP	ISSH
7	6	5	4	3	2	1	0
ADC12DIVx			ADC12SSELx		CONSEQx		ADC12 BUSY

其中灰色的寄存器只有当 ENC=0 时才能更改。

CSTARTADD：单通道模式转换通道/多通道模式守通道

定义单次转换的启始地址或者序列通道转换的首地址。

SHSx：采样触发源选择

0　ADC12SC

1　TimerA.OUT1

2　TimerB.OUT1

3　TimerB.OUT2

SHP：采样信号 SAMPCON 选择

0 SAMPCON 信号来自采样触发输入信号

1 SAMPCON 信号来自采样定时器，由采样输入信号的上升沿触发

ISSH：采样输入信号同向/反向

0 采样信号为同相输入

1 采样信号为反相输入

ADC12DIVx：ADC12 时钟分频控制

ADC12 时钟源的分频因子选择位，分频因子为（x+1）

ADC12SSELx：ADC12 时钟选择

0 ADC12OSC（ADC12 内部时钟源）

1 ACLK

2 MCLK

3 SMCLK

COMSEQx：转换模式

0 单通道单次转换

1 序列通道单次转换

2 单通道多次转换

3 序列通道多次转换

ADC12BUSY：忙标志（转换中...）

0 表示 ADC12 没有活动的操作

1 ADC12 正在采样/转换期间，忙~~

[3] ADC12MCTLx 通道储存控制寄存器

7	6	5	4	3	2	1	0
EOS	SREFx			INCHx			

通道储存控制寄存器控制各转换寄存器必须选择的基本条件。

EOS：多通道转换末通道标志

0 序列没有结束

1 该序列中最后一次转换

SREFx：基准源选择

0 Vr+=AVcc, Vr-=AVss

1 Vr+=Vref+, Vr-=AVss

2,3 Vr+=Veref+, Vr-=AVss

4 Vr+=AVcc, Vr-=Vref-/Veref-

5 Vr+=AVcc, Vr-=Vref-/Veref-

6,7 Vr+=AVcc, Vr-=Vref-/Veref-

INCHx：所对应的模拟电压输入通道

0~7 A0~A7

8 Veref+

9 Veref-/Vref-

10 片内温度传感器

11~15 （AVcc-AVss）/2

[4] ADC12MEMx　　通道储存寄存器

15	14	13	12	11	10	9	8
0	0	0	0	Conversion Results			
7	6	5	4	3	2	1	0
Conversion Results							

该组寄存器为 12 位寄存器，用来存放 A/D 转换结果，其中只用到了低 12 位，高 4 位为 0。

[5] ADC12IFG　　中断标志寄存器

15	14	13	12	11	10	9	8
ADC12 IFG15	ADC12 IFG14	ADC12 IFG13	ADC12 IFG12	ADC12 IFG11	ADC12 IFG10	ADC12 IFG9	ADC12 IFG8
7	6	5	4	3	2	1	0
ADC12 IFG7	ADC12 IFG6	ADC12 IFG5	ADC12 IFG4	ADC12 IFG3	ADC12 IFG2	ADC12 IFG1	ADC12 IFG0

ADC12IFGx：中断标志位

对应于 ADC12MEMx，当 A/D 转换完成后，数据被存入 ADC12MEMx，此时 ADC12IFGx 标志置位。

[6] ADC12IE　　中断控制寄存器

15	14	13	12	11	10	9	8
ADC12IE15	ADC12IE14	ADC12IE13	ADC12IE12	ADC12IE11	ADC12IE10	ADC12IE9	ADC12IE8
7	6	5	4	3	2	1	0
ADC12IE7	ADC12IE6	ADC12IE5	ADC12IE4	ADC12IE3	ADC12IE2	ADC12IE1	ADC12IE0

ADC12IEx：中断允许位

对应于 ADC12IFGx，如果 ADC12IEx 允许，则当 ADC12IFGx 置位时会进入 ADC12 的中断服务程序。

[7] ADC12IV　　中断向量寄存器

由于 ADC12 是一个多源中断，有 18 个中断标志，但是只有一个中断向量。则 18 个中断标志按照优先级安排对中断标志的响应。

ADC12IV 内容	中断源	中断标志	优先级
0x0000	无中断	无	无
0x0002	ADC12MEMx 溢出	ADC12OV	最高
0x0004	转换时间溢出	ADC12TOV	
0x0006	ADC12MEM0	ADC12IFG0	
0x0008	ADC12MEM1	ADC12 IFG1	
0x000a	ADC12MEM2	ADC12 IFG2	
0x000c	ADC12MEM3	ADC12 IFG3	
0x000e	ADC12MEM4	ADC12 IFG4	
0x0010	ADC12MEM5	ADC12 IFG5	
0x0012	ADC12MEM6	ADC12 IFG6	
0x0014	ADC12MEM7	ADC12 IFG7	
0x0016	ADC12MEM8	ADC12 IFG8	
0x0018	ADC12MEM9	ADC12 IFG9	
0x001a	ADC12MEM10	ADC12 IFG10	
0x001c	ADC12MEM11	ADC12 IFG11	
0x001e	ADC12MEM12	ADC12 IFG12	
0x0020	ADC12MEM13	ADC12 IFG13	
0x0022	ADC12MEM14	ADC12 IFG14	
0x0024	ADC12MEM15	ADC12 IFG15	最低

13.4 ADC12 转换模式

ADC12 模块一共提供了 4 钟转换模式

- 单通道单次转换
- 序列通道单次转换
- 单通道多次转换
- 序列通道多次转换

不论用户使用何种模式，都需要注意以下问题

- 设置具体的转换模式
- 输入模拟信号
- 选择启动信号
- 关注结束信号
- 存放转换数据
- 采用查询或者中断方式来读取数据

[1] 单通道单次转换

进行单通道单次转换需要注意以下的设置

- 设置通道控制寄存器，设置采样通道和参考电压，ADC12MCTLx
- 单通道转换的地址 CSTARTADD，对应于上面的 ADC12MCTLx
- 相对应的中断标志 ADC12IFGx

下面是软件查询式的单通道单次转换所需要对 ADC12 进行的设置

```
//-------------------------------------------------------------------------
// 单通道单次转换的通道不一定要是 ADC12MCTL0, 程序中使用了 ADC12MCTL4, 但是采样的通道
// 是 A0, MSP430 中的 ADC12 通道设置是非常灵活的.
// ADC12 单通道单次转换(软件查询式)
void ADC12_Sampling_SingleChannelSingleConvert(void)
{
  // ADC12 控制寄存器设置
  // 内核开启, 启动内部基准, 选择 2.5V 基准, 设置采样保持时间
  ADC12CTL0 = ADC12ON + REFON + REF2_5V + SHT0_2;
  // 时钟源为内部震荡器, 出发信号来自采样定时器, 转换地址为 ADC12MCTL4
  ADC12CTL1 = ADC12SSEL_0 + SHP + CSTARTADD_4;
  // 转换通道设置
  ADC12MCTL4 = SREF_1 + INCH_0;       // 参考电压:V+=Vref+,V-=AVss  ADC 通道:A0
  // 启动转换
  ADC12CTL0 |= ENC + ADC12SC;         // 转换使能开始转换
  while((ADC12IFG & 0x0010) == 0);    // 软件查询中断标志, 等待转换结束
  _NOP();                             // 处理
}
//-------------------------------------------------------------------------
```

下面是中断查询式的单通道单次转换所需要对 ADC12 进行的设置

```
//-------------------------------------------------------------------------
// ADC12 单通道单次转换(中断查询式)
void ADC12_Sampling_SingleChannelSingleConvert(void)
{
  // ADC12 控制寄存器设置
  ADC12CTL0 = ADC12ON + REFON + REF2_5V + SHT0_2;
  ADC12CTL1 = ADC12SSEL_0 + SHP + CSTARTADD_4;
  // 转换通道设置
  ADC12MCTL4 = SREF_1 + INCH_0;       // 参考电压:V+=Vref+,V-=AVss  ADC 通道:A0
  // 中断允许
  ADC12IE = 0x0010;
  _EINT();
  // 启动转换
  ADC12CTL0 |= ENC + ADC12SC;         // 转换使能开始转换
  __low_power_mode_0();               // 进入低功耗模式, 等待转换结束
}

// ADC12 中断向量
#pragma vector = ADC_VECTOR
__interrupt void ADC12_IRQ(void)
{
  _NOP();    // 处理
  __low_power_mode_off_on_exit();  // 中断结束时, 退出低功耗模式
}
//-------------------------------------------------------------------------
```

[2] 序列通道单次转换

进行序列通道单次转换需要注意以下的设置

- 设置若干个通道控制寄存器，设置采样通道和参考电压，ADC12MCTLx，最后一个通道需要加 EOS
- 序列通道转换的首地址 CSTARTADD，对应于上面的第一个 ADC12MCTLx 相对应的最后一个通道的中断标志 ADC12IFGx 下面是软件查询式的序列通道单次转换所需要对 ADC12 进行的设置

```
//-------------------------------------------------------------------------
// ADC12 序列通道单次转换(软件查询式)
void ADC12_Sampling_SequenceChannelsSingleConvert(void)
{
  // ADC12 控制寄存器设置
  ADC12CTL0 = ADC12ON + REFON + REF2_5V + SHT0_2;
  // CONSEQ_1 表示当前模式为序列通道单次转换, 起始地址为 ADC12MCTL4, 结束地址 ADC12MCTL6
  ADC12CTL1 = ADC12SSEL_0 + SHP + CONSEQ_1 + CSTARTADD_4;
  // 转换通道设置
  ADC12MCTL4 = SREF_1 + INCH_0;          // 参考电压:V+=Vref+,V-=AVss  ADC 通道:A0
  ADC12MCTL5 = SREF_1 + INCH_1;          // 参考电压:V+=Vref+,V-=AVss  ADC 通道:A1
  ADC12MCTL6 = SREF_1 + INCH_10 + EOS;  // 参考电压:V+=Vref+,V-=AVss  ADC 通道:片内温度传感器
  // 启动转换
  ADC12CTL0 |= ENC + ADC12SC;          // 转换使能开始转换
  while((ADC12IFG & 0x0040) == 0);    // 等待转换结束
  _NOP();                              // 处理
}
//-------------------------------------------------------------------------
```

[3] 单通道多次转换

进行单通道多次转换需要注意以下的设置

- 设置通道控制寄存器，设置采样通道和参考电压，ADC12MCTLx
- 单通道转换地址 CSTARTADD，对应于上面的 ADC12MCTLx
- 单通道的中断标志 ADC12IFGx
- 多次转换只能使用中断查询式读数

下面是软件查询式的单通道多次转换所需要对 ADC12 进行的设置

```
//-------------------------------------------------------------------------
// ADC12 单通道多次转换
void ADC12_Sampling_SingleChannelSequenceConvert(void)
{
  // ADC12 控制寄存器设置
  // 对于多次转换需要设置 MSC
  ADC12CTL0 = ADC12ON + REFON + REF2_5V + SHT0_2 + MSC;
  // CONSEQ_2 表示当前模式为单通道多次转换, 转换地址为 ADC12MCTL4
  ADC12CTL1 =   ADC12SSEL_0 + SHP + CONSEQ_2 + CSTARTADD_4;
  // 转换通道设置
  ADC12MCTL4 = SREF_1 + INCH_4 + EOS;    // 参考电压:V+=Vref+,V-=AVss  ADC 通道:A4
  // 中断允许
  ADC12IE = 0x0010;
  _EINT();
  // 启动转换
  ADC12CTL0 |= ENC + ADC12SC;     // 转换使能开始转换
}

// ADC12 中断向量
#pragma vector = ADC_VECTOR
__interrupt void ADC12_IRQ(void)
{
  _NOP();    // 处理
}
//-------------------------------------------------------------------------
```

[4] 序列通道多次转换

进行序列通道多次转换需要注意以下的设置

- 设置若干个通道控制寄存器，设置采样通道和参考电压，ADC12MCTLx，最后一个通道需要加 EOS
- 序列通道转换的首地址 CSTARTADD，对应于上面的第一个 ADC12MCTLx
- 相对应的最后一个通道的中断标志 ADC12IFGx
- 多次转换只能使用中断查询式读数 下面是软件查询式的序列通道多次转换所需要对 ADC12 进行的设置

```
//-------------------------------------------------------------------------
// ADC12 序列通道多次转换
void ADC12_Sampling_SequenceChannelSequenceConvert(void)
{
  // ADC12 控制寄存器设置
  ADC12CTL0 = ADC12ON + REFON + REF2_5V + SHT0_2 + MSC;
  ADC12CTL1 = ADC12SSEL_0 + SHP + CONSEQ_3 + CSTARTADD_4;
  // 转换通道设置
  ADC12MCTL4 = SREF_1 + INCH_4 + EOS;    // 参考电压:V+=Vref+,V-=AVss  ADC 通道:A4
  ADC12MCTL5 = SREF_1 + INCH_5 + EOS;    // 参考电压:V+=Vref+,V-=AVss  ADC 通道:A5
  ADC12MCTL6 = SREF_1 + INCH_6 + EOS;    // 参考电压:V+=Vref+,V-=AVss  ADC 通道:A6
  // 中断允许
  ADC12IE = 0x0040;
  _EINT();
  // 启动转换
  ADC12CTL0 |= ENC + ADC12SC;     // 转换使能开始转换
}

// ADC12 中断向量
#pragma vector = ADC_VECTOR
__interrupt void ADC12_IRQ(void)
{
  _NOP();    // 处理
}
//-------------------------------------------------------------------------
```

第十四章 DAC12 数模转换模块

12bit D / A Conversion Module

14.1 数模转换概述

MSP430 带有的 DAC12 模块，可以将运算处理的结果转换为模拟量，以便操作被控制对象的工作过程。MSP430 的 DAC12 模块是 12 位 R 阶，电压输出的数模转换模块，在使用的过程中可以被设置为 8 位或者 12 位转换模式，并能够和 DMA 控制器结合使用。当 MSP430 包含有多个 DAC12 模块的时候，MSP430 可以对他们进行统一管理。

MSP430 的 DAC12 模块只要性能指标有以下几点：

[1]分辨率：这相指标反映了数字在最低位上的数字变化 1 时的模拟输出变化。对 12 位 DAC 来说，其分辨率达到了 1/4096，即 0.025%。

[2]偏移误差，它指的是输入数字为 0 的时候，输出模拟量对 0 的误差

[3]线性度：指的是 DAC 模块的实际转移特性与理想直线之间的最大偏差。

[4]转换速度：即每秒种转换的次数

[5]其他因素：例如数字输入特性，输入的码制，数据格式；输出特性，输出的是电流还是电压；等...

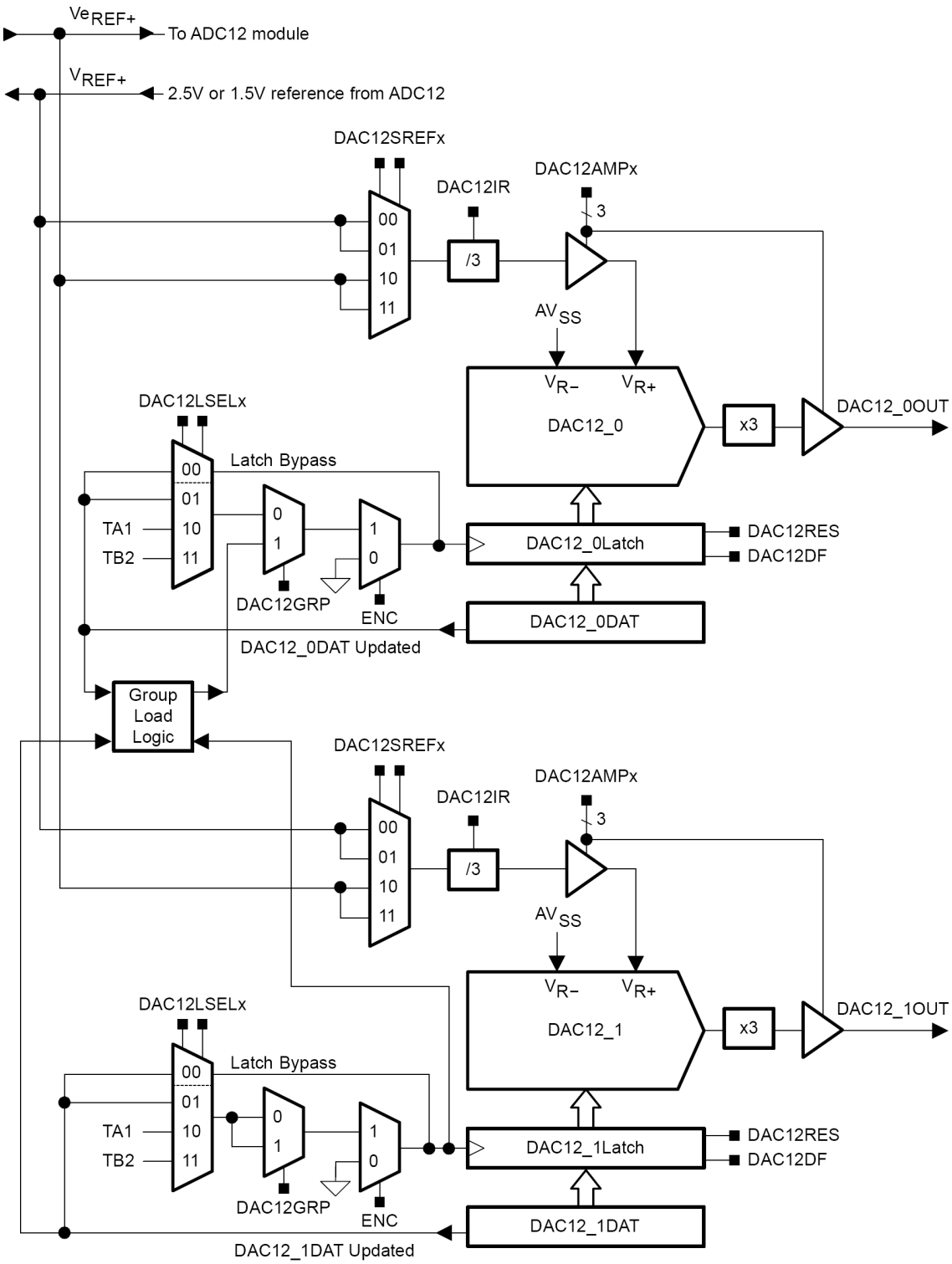
14.2 DAC12 结构与性能

MSP430F169 单片机的 DAC12 模块有 2 个 DAC 通道，这两个通道在操作上是完全平等的。并且可以用 DAC12GRP 控制位将多个 DAC12 通道组合起来，实现同步更新，硬件还能确保同步更新独立于任何中断或者 NMI 事件。

DAC12 的主要特征

- 8 位，12 位分辨率
- 可编程的时间对能量的消耗
- 内部和外部的参考电压选择
- 支持无符号和有符号的数据输入
- 具有自效验功能
- 二进制或者二进制的补码形式
- 多路 DAC 同步更新
- 可以直接用存储器存取

DAC12 结构原理图

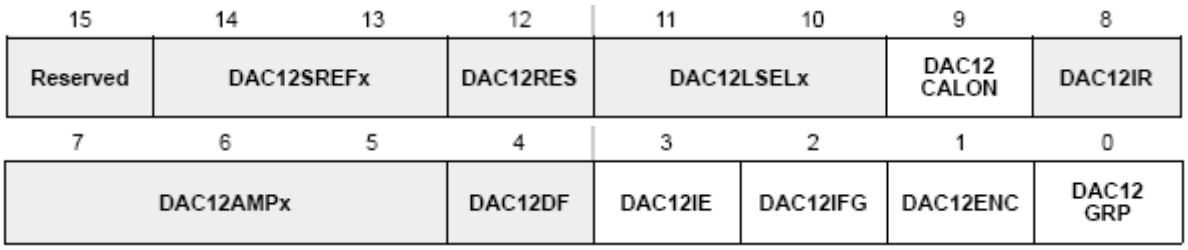


14.3 DAC12 寄存器

MSP430F169 的 DAC12 模块寄存器有以下几组：

寄存器	缩写
DAC12_0 控制寄存器	DAC12_0CTL
DAC12_0 数据寄存器	DAC12_0DAT
DAC12_1 控制寄存器	DAC12_1CTL
DAC12_1 数据寄存器	DAC12_1DAT

[1] DAC12_xCTL DAC12 控制寄存器



其中灰色的部分只有在 DAC12ENC=0 的时候才能修改，位 DAC12GRP 只有在 DAC12_1 才使用。

DAC12REFx：选择 DAC12 的参考源

0，1 Vref+

2，3 Veref+

DAC12RES：选择 DAC12 分辨率

0 12 位分辨率

1 8 分辨率

DAC12LSELx：锁存器触发源选择

当 DAC12 锁存器得到触发之后，能够将锁存器中的数据传送到 DAC12 的内核。

当 DAC12LSELx=0 的时候，DAC 数据更新不受 DAC12ENC 的影响。

0 DAC12_XDAT 执行写操作将触发（不考虑 DAC12ENC 的状态）

1 DAC12_XDAT 执行写操作将触发（考虑 DAC12ENC 的状态）

2 Timer_A3.OUT1 的上升沿

3 Timer_B7.OUT2 的上升沿

DAC12CALON：DAC12 校验操作控制

置位后启动校验操作，校验完成后自动被复位。校验操作可以校正偏移误差。

0 没有启动校验操作

1 启动校验操作

DAC12IR：DAC12 输入范围

设定输入参考电压和输出的关系

0 DAC12 的满量程为参考电压的 3 倍（不操作 AVcc）

1 DAC12 的满量程为参考电压

DAC12AMPx：DAC12 运算放大器设置

0 输入缓冲器关闭，输出缓冲器关闭，高阻

1 输入缓冲器关闭，输出缓冲器关闭，0V

2　输入缓冲器低速低电流，输出缓冲器低速低电流
3　输入缓冲器低速低电流，输出缓冲器中速中电流
4　输入缓冲器低速低电流，输出缓冲器高速高电流
5　输入缓冲器中速中电流，输出缓冲器中速中电流
6　输入缓冲器中速中电流，输出缓冲器高速高电流
7　输入缓冲器高速高电流，输出缓冲器高速高电流

DAC12DF：DAC12 的数据格式

0　二进制
1　二进制补码

DAC12IE：DAC12 的中断允许

0　禁止中断
1　允许中断

DAC12IFG：DAC12 的中断标志位

0　没有中断请求
1　有中断请求

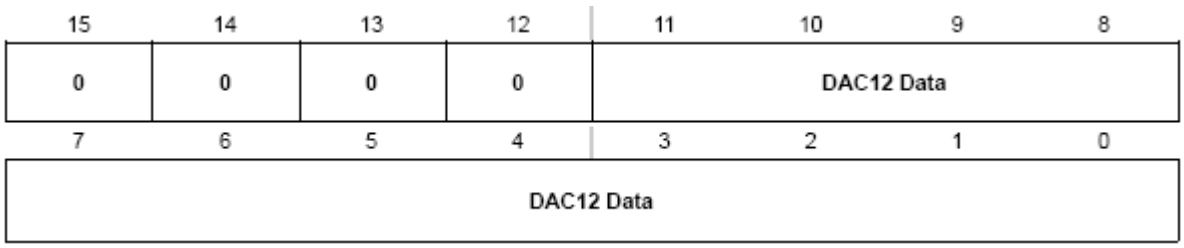
DAC12ENC：DAC12 转换控制位

DAC12LSEL>0 的时候，DAC12ENC 才有效。
0　DAC12 停止
1　DAC12 转换

DAC12GRP：DAC12 组合控制位

0　没有组合
1　组合

[2] DAC12_xDAT　　数据寄存器



DAC12_xDAT 高 4 位为 0，不影响 DAC12 的工作。如果 DAC12 工作在 8 位模式的话，DAC12_xDAT 的值最大为 0x00FF；ADC12 工作在 12 位模式时，DAC12_xDAT 的最大值为 0x0FFF。

14.4 DAC12 的操作

通过了解 DAC12 的操作可以深入的理解 DAC12 的家够和原理，DAC12 的操作都是有软件进行设置的。

[1]选择参考电压

参考电压是唯一影响 DAC12 输出量的模拟参数，是 DAC12 转换模块的重要部分。DAC12 可以选择内部或者外部的参考电压,其中内部的参考电压来自与 ADC12 中的内部参考电压发生器生成的 1.5V 或者 2.5V 参考电压。

DAC12 参考电压的输入和电压输出缓冲器的时间和功耗可以通过编程来控制使其工作的最佳状态（DAC12AMPx 控制）。

[2]DAC12 内核

用 DAC12RES 位选择 DAC12 的 8 位或者 12 位精度。DAC12IR 位控制 DAC12 的最大输出电压为参考电压的 1 或 3 倍（在不超过电源电压的情况下）。DAC12DF 设置写到 DAC12 的数据的格式。

位数	DAC12RES	DAC12IR	输出电压格式
12 位	0	0	$V_{OUT}=V_{REF}\times3\times(DAC12\_xDAT)/4096$
12 位	0	1	$V_{OUT}=V_{REF}\times1\times(DAC12\_xDAT)/4096$
8 位	1	0	$V_{OUT}=V_{REF}\times3\times(DAC12\_xDAT)/256$
8 位	1	1	$V_{OUT}=V_{REF}\times1\times(DAC12\_xDAT)/256$

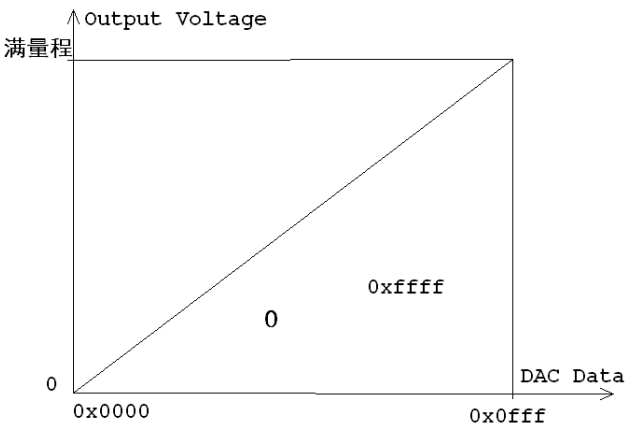
[3]更新 DAC12 的电压输出

DAC12 输出引脚和断口 P6 以及 ADC12 模拟输入复用，当 DAC12AMPx>0 时，不管当前端口 P6 的 P6SEL 和 P6DIR 对应的位的状态是什么。该引脚自动被选择到 DAC12 的功能上。

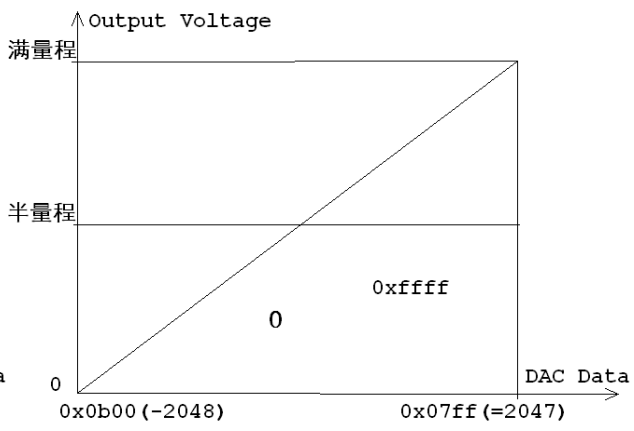
DAC12_xDAT 可以直接将数据传送到 DAC12 的内核以及 DAC12 的两个缓冲器。DAC12LSELx 位可以触发对 DAC12 的电压输出更新。当 DAC12LSELx=0 的时候，数据锁存变的透明，不管当前的 DAC12ENC 处于什么状态，只要 DAC12_xDAT 被更新，则 DAC12 的内核立刻被更新。当 DAC12LSELx=1 的时候，除非有新的数据写入 DAC12_xDAT，否则 DAC12 的数据一直被缩存。当 DAC12LSELx=2、3 的时候，数据在 TA 的 TACCR1 或者 TB 的 TBCCR2 输出信号上升沿时刻被锁存。当 DAC12LSELx>0 的时候，DAC12ENC 用来使能 DAC12 的锁存。

[4]DAC12_xDAT 的数据格式

DAC12 支持二进制数或者 2 的补码输入。



当采用二进制数格式时



当采用 2 的补码时

[5]校正 DAC12 的输出

DAC12 存在偏移误差，可以通过 DAC12 自动校正偏移量，

在使用 DAC12 之前。通过设置控制位 DAC12CALON 能够初始化偏移校正操作，操作完成后 DAC12CALON 会自动复位。

14.5 DAC12 的设置和应用

DAC12 使用的时候需要注意的问题：

- 参考电压的选择，如果使用内部参考电压则需要在 ADC12 模块里面打开内部参考电压发生器，ADC12 内核不用开。
- 在 MSP430F169 单片机上，DAC12 的 0 通道使用的是 A6，1 通道使用的是 A7 的引脚。注意如果使用了 DAC12 的 2 个通道，A6 和 A7 就不能使用。
- 校正 DAC12 的偏移误差
- 设置 DAC12 的位数和满量程电压（满量程电压最高为 AVcc）
- 设置 DAC12 的触发模式

下面是 DAC12 的 0 通道设置初始化代码：

```
// -------------------------------------------------------
// 初始化 DAC12 通道 0
void InitDAC12_0(void)
{
  // 效验 DAC
  DAC12_0CTL |= DAC12CALON;                          // 启动效验 DAC
  while((DAC12_0CTL & DAC12CALON) != 0){_NOP();}  // 等待效验完成

  // 控制寄存器设置
  // 选择输入缓冲器中速中电流，输出缓冲器中速中电流，12 位 DAC，满电压输出为内基准，自动更新数据
  DAC12_0CTL  = DAC12AMP_5 + DAC12IR + DAC12LSEL_0;
  DAC12_0CTL |= DAC12SREF_0;
  DAC12_0CTL |= DAC12ENC;                             // 启动 DAC 模块(DAC12LSEL_0 时此句可以省)

  DAC12_0DAT  = 0x0000; // 初始化电压
}
// 以后更改输出电压，只需要更改 DAC12_0DAT 值即可。
// -------------------------------------------------------
```

第二部分

MSP430 系列单片机教学实验系统

第一章 系统板总体介绍

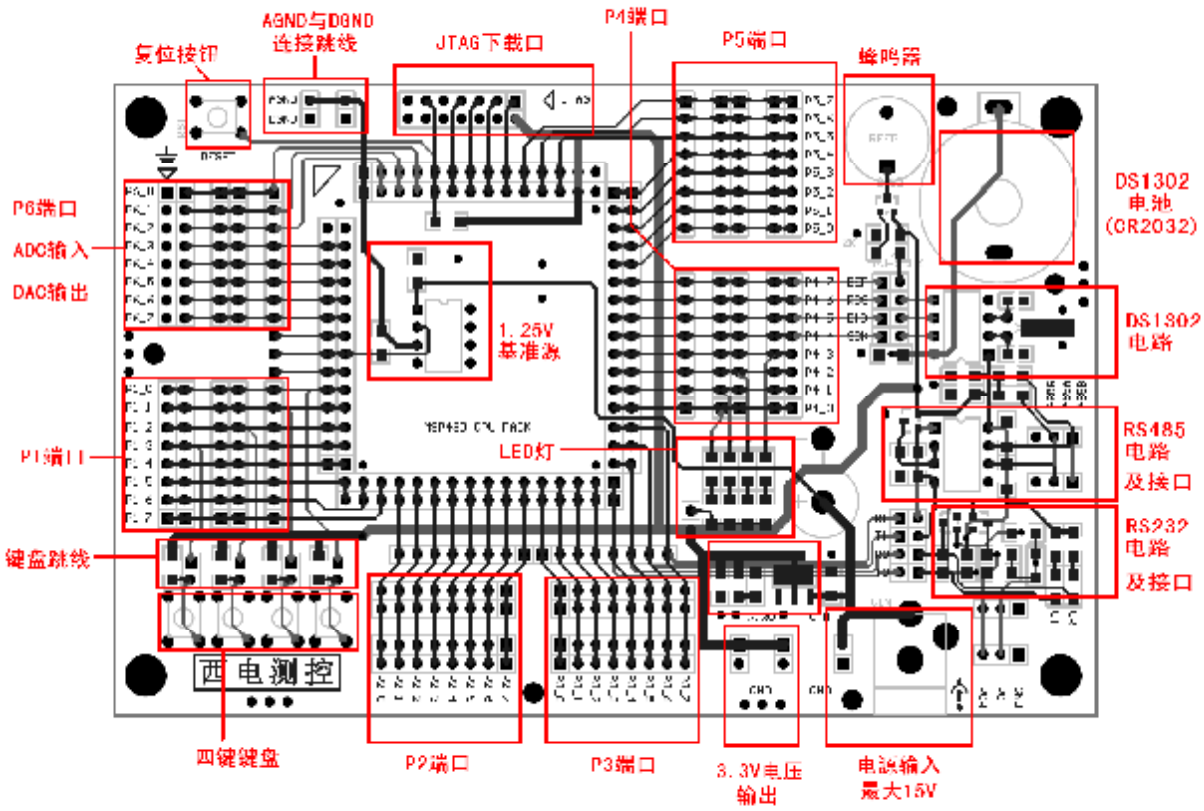
该实验板由 CPUPACK，系统主板，键盘板三部分构成。

CPUPACK 部分将 MSP430F169 单片机的 I/O 口引出使之便于更换维护

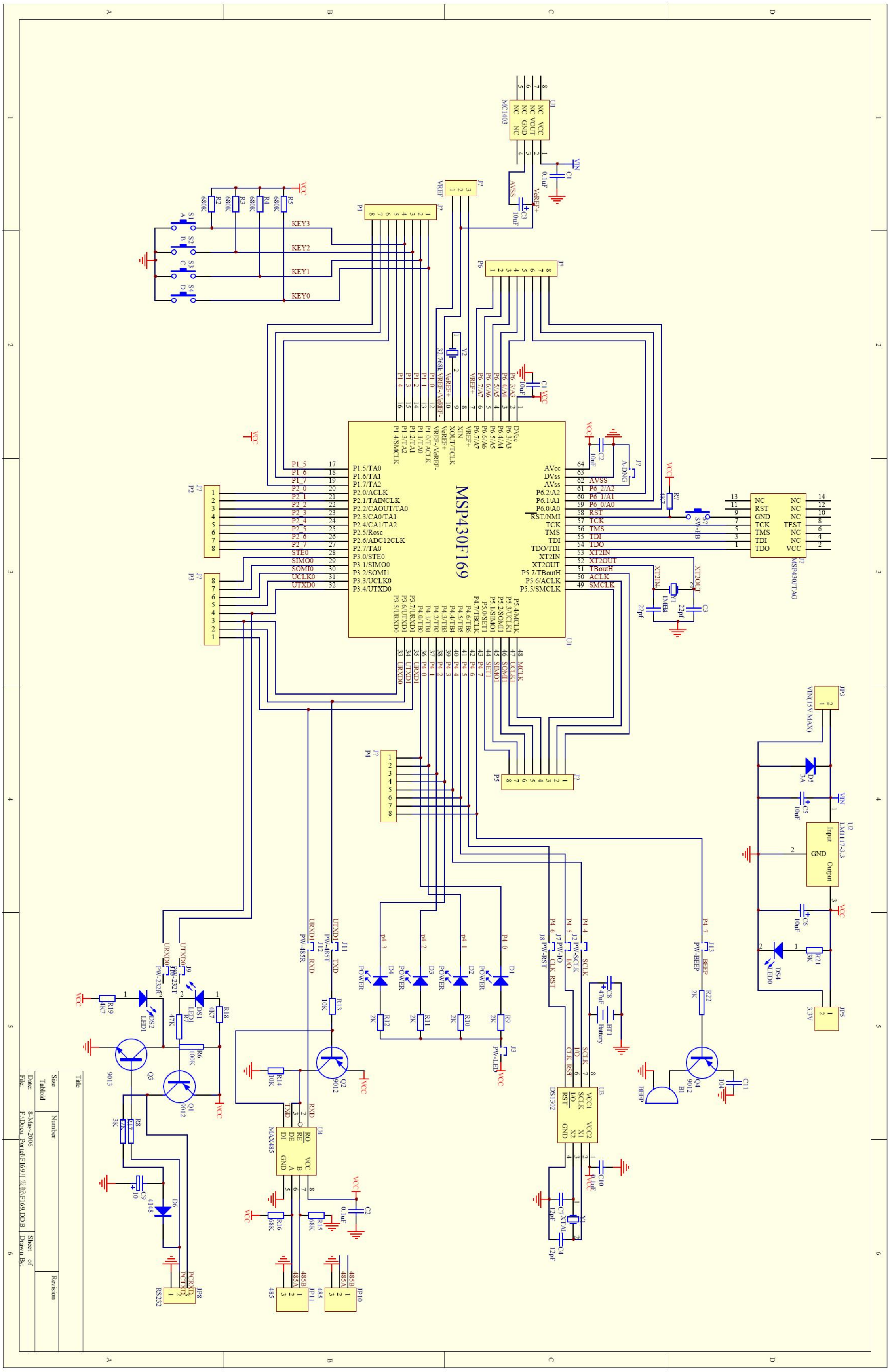
键盘为 4X4 行列键盘，引出 8 个输出口

系统主板为该系统的主体，其将 MSP430F169 的端口逐一引出并未限制其用途，这样增加了系统的灵活性。

在不降低系统灵活特色的前提下系统板扩充了 2.5V 基准（MC1403），实时钟（DS1302），RS485 通信模块（MAX485），RS232 通信模块，四键键盘，蜂鸣器等模块，这些器件与 MSP430 自身集成的 USART，ADC，DAC，I/O 组合，方便了系统使用。此外这些模块均用跳线与 CPU 相连，不用时可以将跳线断开，不影响 I/O 口的正常操作。



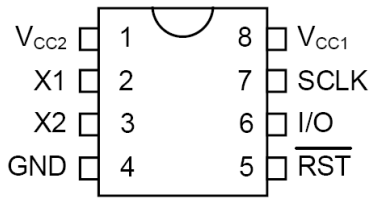
1.1 实验板电路原理图



第二章 外围电路介绍

2.1 DSl302 实时时钟芯片

DSl302 芯片是美国 Dallas 公司推出的串行接口实时时钟芯片，可对时钟芯片备份电池进行涓流充电。由于该芯片具有体积小、功耗低、接口容易、占用 CPU 的 I / O 口线少等主要特点，故该芯片可作为实时时钟广泛应用于智能化仪器仪表中。DS1302 带备份电源的、8 脚、具有 I^2C 串行通信功能的高性能、低功耗，提供秒、分、时、日、周、月、年日历功能。
DS1302 的芯片引脚如图

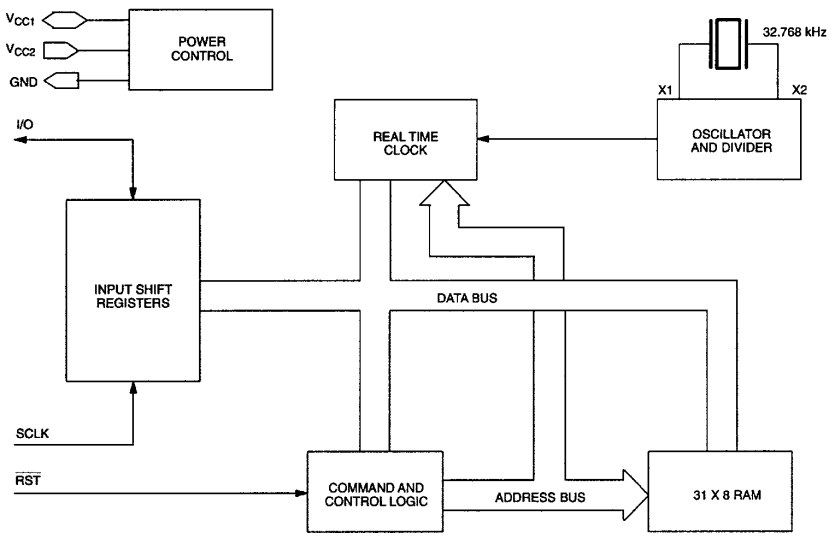


DS1302 8-Pin DIP (300-mil)

引脚

Vcc1： 备用电池电源脚，可以接 3.6V Ni-MH 充电电池
Vcc2： 常用电源脚，接 5V 直流电
GND： 接地
X1： 接 32.768KHz 的晶振。并通过 12pF 电容接地
X2： 接 32.768KHz 的晶振。并通过 12pF 电容接地
RST： 与 CPU 通信的 SPI 总线的片选线
SCLK： 与 CPU 通信的 SPI 总线同步时钟线
I/O： 与 CPU 通信的 SPI 总线输入输出数据线

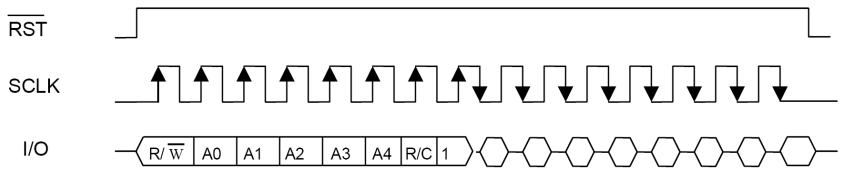
DS1302 的内部原理



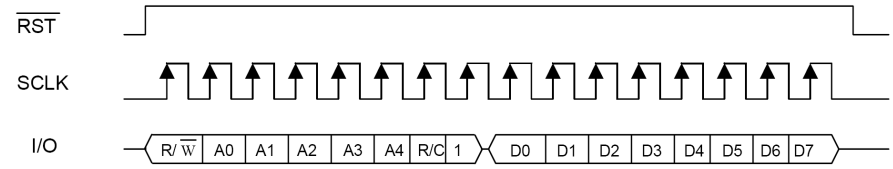
数据读写时续

RST 脚，SCLK 脚，I/O 脚构成的总线传输数据有两种模式：单位模式和触发模式。

SINGLE BYTE READ

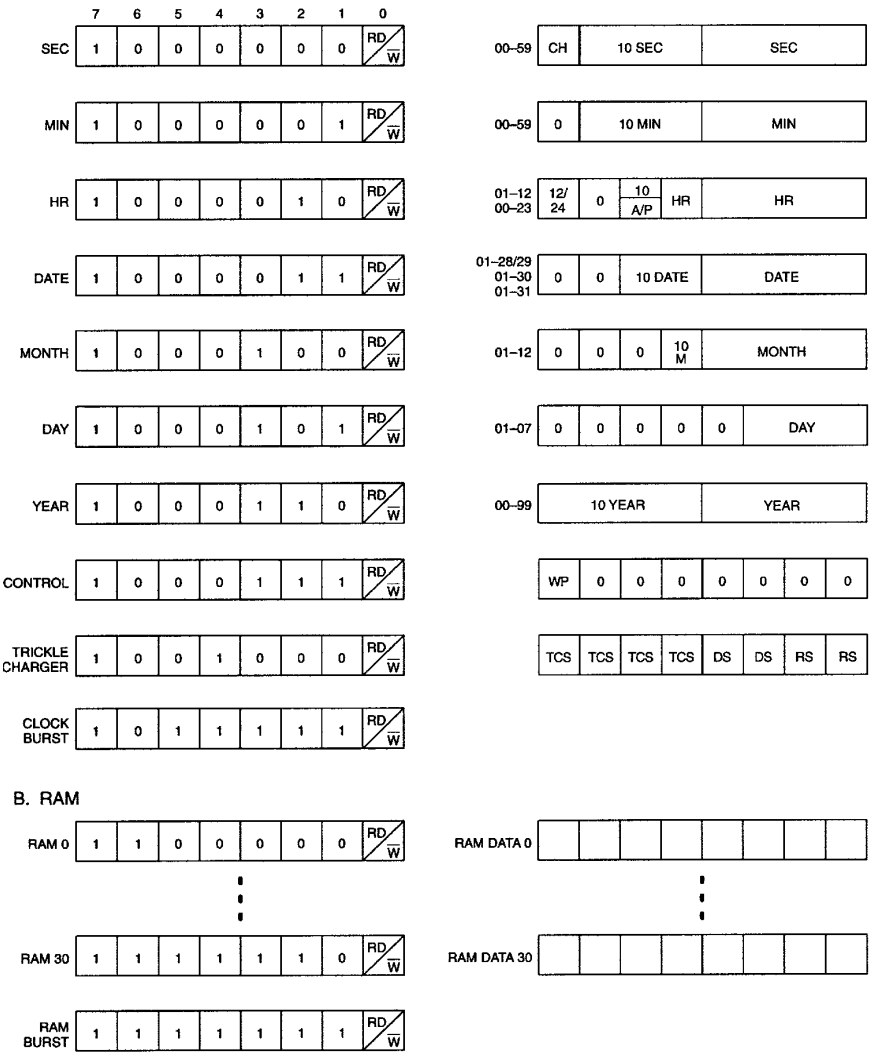


SINGLE BYTE WRITE



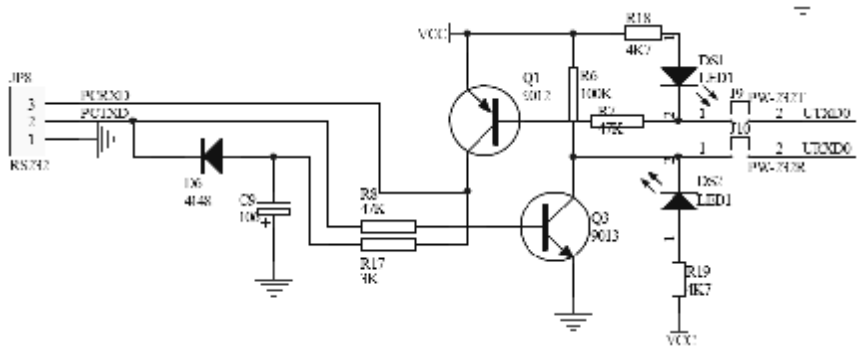
In burst mode, $\overline{RST}$ is kept high and additional SCLK cycles are sent until the end of the burst.

时钟寄存器和 RAM 地址



2.2 RS232 电路

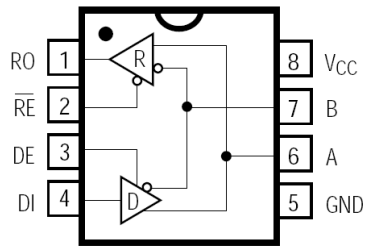
RS232 部分是由两个三极管借用计算机 232 总线上 TXD 空闲时的输出电压巧妙的将单片机的 3.3V 逻辑电平转化为计算机能够接受的正负 11V 电平。该部分原理图如下



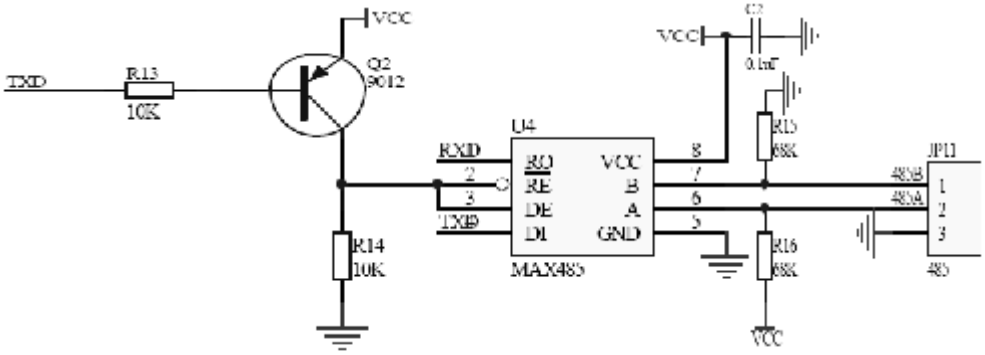
单片机从 UTXD0 发送 3.3V 电平当 UTXD0 为高时 PCRXD 输出 VCC，当 UTXD0 为低时 PCRXD 输出低电平。当 PCTXD 为低电平时 URXD0 为高电平当 PCTXD 为高时 URXD0 为低。这样就完成了 TTL 电平到 RS232 电平的转换。

2.3 RS485 电路

RS485 转换电路主要采用了 MAXIM 公司的 MAX485 接口芯片该芯片接口图如下



系统板上该部分电路原理图如下：



该电路能够自动切换输入输出方向平时该电路能正常接收上位网络的数据当单片机发送数据时当发送低电平时 MAX485 的 DE 脚为高选择到输出状态输出数据 0，由于平时空闲总线上数据默认为 1，所以发送 1 时虽然 MAX485 并没有切换到输出状态但其数据早已出现在了总线上。这样就完成了 485 输入输出的自动切换。

第三部分

MSP430 系列单片机教学实验教程

实验一　设置基本时钟系统

实验目的

设置基本时钟系统

实验要求

熟练掌握对 MSP430 基本时钟系统的操作和时钟资源的分配。

实验内容

[1] 了解 MSP430F169 的时钟资源。
了解 3 个时钟源，低频震荡器 XT1，高频震荡器 XT2，数控震荡器 DCO。
了解 3 个系统时钟通道 ACLK，MCLK，SMCLK。
ACLK 的时钟源只能来自 XT1。
MCLK 的时钟源能来自 XT1，XT2，DCO。
SMCLK 的时钟源能来自 XT2，DCO。

[2] 启动震荡器
了解震荡器失效标志的含义。
XT1 和 DCO 震荡器上电即启动。
XT2 震荡器需要操作 XT2OFF 方能启动
掌握如何查询失效标志和处理震荡器失效问题

[3] 分配时钟资源
上电复位后默认 XT2 关，ACLK 来自 XT1，MCLK 和 SMCLK 都来自 DCO。
掌握通过对寄存器的操作分配时钟信号：
设置 ACLK 来自 XT1，MCLK 来自 XT2，SMCLK 来自 XT2。
各个时钟通道的分频自定。

[4] 时钟信号的输出
时钟信号可以由 P5.6、P5.5、P5.4 输出。
需要修改 I/O 寄存器将管脚设置为输出和功能脚模式。

实验注意事项

[1] 检测震荡器是否工作，如果震荡器失效，则会有失效标志产生。在清除了失效标志以后不要立刻再次去检查失效标志，最好延迟一段时间再去检查，因为震荡器起震需要一段毫秒级的时间。

[2] 注意上电后默认的时钟分配状态

[3] 程序开始要关闭看门狗（默认是开的），使用 WDTCTL = WDTPW + WDTHOLD;语句。

实验二　调整 DCO 发生器

实验目的

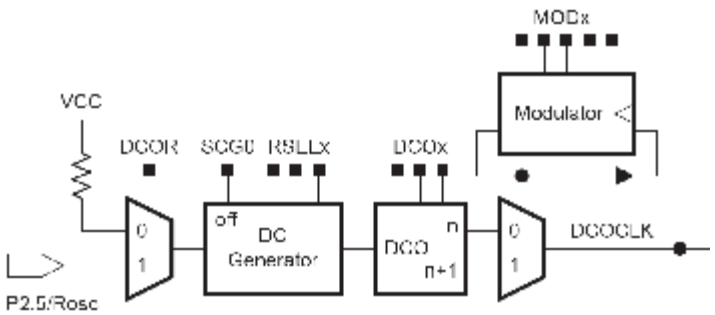
调整 DCO 发生器

实验要求

熟练掌握调整 DCO 发生器的信号频率。

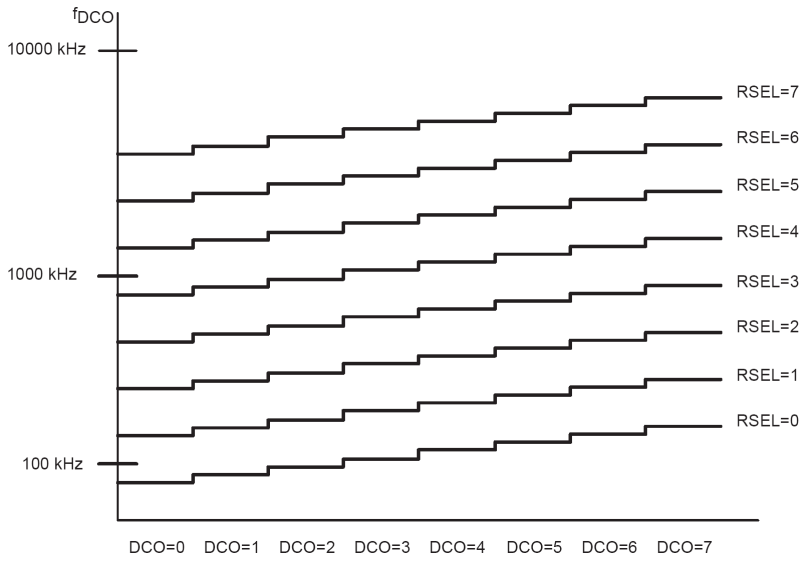
实验内容

[1]　了解 DCO 发生器的原理



VCC 经过电阻产生直流电流（DCOR 选择使用内部还是外部电阻）
直流电流进入 DC Generator 产生调整过的直流电流（通过 RSEL 的选择来调整电流大小）
整过的直流电流进入 DCO 震荡器，则 DCO 产生震荡信号（信号频率由 DCOx 决定，但是频段收直流电流影响，所以 DCOx 和 RSELx 共同决定 DCO 频率）
产生的 DCOCLK 信号再经过 Moduletor 进行微调（微调由 MODx 控制）

[2]　频率的选择
频率由 DCOx 和 RSELx 共同决定。
如果使用了外部电阻的话，频率还和电阻有关。
下图为 DCOx 和 RSELx 与频率的关系图。



[3] 设置好 DCO 频率，并将 MCLK 时钟源选择为来自 DCO，然后通过引脚输出，用示波器观察或用频率计测量频率。

[4] 在线修改 DCOx 和 RSELx 的相关寄存器观察频率的改变（不需要重新下载程序）。

[5] 修改 DCOx 相关寄存器，观察每次频率的增加/减少幅度，应该在 10%左右。

[5] 修改 MODx 相关寄存器观察信号的微调。

实验注意事项

[1] DCO 产生的频率不是稳定的频率，由于信号是有单片机内部的电路产生，所以受外界的影响比较大，其中尤其是温度的影响。可以尝试用手指按住单片机，然后观察频率的缓慢变化。

[2] 由于 DCO 频率的不稳定，所以需要注意 DCO 所使用的场合，一般情况下使用在对频率不需要很精确的场合。

如果时钟要供其他模块使用，例如定时器模块，串口通讯模块等时，需要在单片机外部设计晶振并连接至合适的引脚（低频 32.768KHz 的连接到 XT1，8MHz 以上的连接到 XT2）

[3] MSP430 系列单片机可以不接任何晶振工作，但是这样没有 ACLK，只有 MCLK 和 SMCLK，而且 MCLK 和 SMCLK 全部来自于 DCO。

[4] 设计时钟部分硬件电路时需要注意器件手册上面的数据指标

附表

DCO 选择外部电阻的数据

DCO when using R_{OSC} (see Note 1)

PARAMETER	TEST CONDITIONS	V_{CC}	MIN	NOM	MAX	UNIT
f_{DCO}, DCO output frequency	R_{sel} = 4, DCO = 3, MOD = 0, DCOR = 1, T_A = 25°C	2.2 V		1.8±15%		MHz
		3 V		1.95±15%		MHz
D_t, Temperature drift	R_{sel} = 4, DCO = 3, MOD = 0, DCOR = 1	2.2 V/3 V		±0.1		%/°C
D_V, Drift with V_{CC} variation	R_{sel} = 4, DCO = 3, MOD = 0, DCOR = 1	2.2 V/3 V		10		%/V

NOTES: 1. R_{OSC} = 100kΩ. Metal film resistor, type 0257. 0.6 watt with 1% tolerance and T_K = ±50ppm/°C.

基本操作环境中对 XT1 和 XT2 的要求

			MIN	NOM	MAX	UNITS
Supply voltage during program execution, V_{CC} (AV_{CC} = DV_{CC} = V_{CC})		MSP430F15x/16x/161x	1.8		3.6	V
Supply voltage during flash memory programming, V_{CC} (AV_{CC} = DV_{CC} = V_{CC})		MSP430F15x/16x/161x	2.7		3.6	V
Supply voltage during program execution, SVS enabled (see Note 1), V_{CC} (AV_{CC} = DV_{CC} = V_{CC})		MSP430F15x/16x/161x	2		3.6	V
Supply voltage, V_{SS} (AV_{SS} = DV_{SS} = V_{SS})			0		0	V
Operating free-air temperature range, T_A		MSP430F15x/16x/161x	−40		85	°C
LFXT1 crystal frequency, $f_{(LFXT1)}$ (see Notes 2 and 3)	LF selected, XTS=0	Watch crystal		32.768		kHz
	XT1 selected, XTS=1	Ceramic resonator	450		8000	kHz
	XT1 selected, XTS=1	Crystal	1000		8000	kHz
XT2 crystal frequency, $f_{(XT2)}$		Ceramic resonator	450		8000	kHz
		Crystal	1000		8000	
Processor frequency (signal MCLK), $f_{(System)}$		V_{CC} = 1.8 V	DC		4.15	MHz
		V_{CC} = 3.6 V	DC		8	

DCO 频率典型值

DCO (see Note 1)

PARAMETER	TEST CONDITIONS		MIN	NOM	MAX	UNIT
$f_{(DCO03)}$	R_{sel} = 0, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.08	0.12	0.15	MHz
		V_{CC} = 3 V	0.08	0.13	0.16	
$f_{(DCO13)}$	R_{sel} = 1, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.14	0.19	0.23	MHz
		V_{CC} = 3 V	0.14	0.18	0.22	
$f_{(DCO23)}$	R_{sel} = 2, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.22	0.30	0.36	MHz
		V_{CC} = 3 V	0.22	0.28	0.34	
$f_{(DCO33)}$	R_{sel} = 3, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.37	0.49	0.59	MHz
		V_{CC} = 3 V	0.37	0.47	0.56	
$f_{(DCO43)}$	R_{sel} = 4, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	0.61	0.77	0.93	MHz
		V_{CC} = 3 V	0.61	0.75	0.90	
$f_{(DCO53)}$	R_{sel} = 5, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	1	1.2	1.5	MHz
		V_{CC} = 3 V	1	1.3	1.5	
$f_{(DCO63)}$	R_{sel} = 6, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	1.6	1.9	2.2	MHz
		V_{CC} = 3 V	1.69	2.0	2.29	
$f_{(DCO73)}$	R_{sel} = 7, DCO = 3, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	2.4	2.9	3.4	MHz
		V_{CC} = 3 V	2.7	3.2	3.65	
$f_{(DCO47)}$	R_{sel} = 4, DCO = 7, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V/3 V	f_{DCO40} × 1.7	f_{DCO40} × 2.1	f_{DCO40} × 2.5	MHz
$f_{(DCO77)}$	R_{sel} = 7, DCO = 7, MOD = 0, DCOR = 0, T_A = 25°C	V_{CC} = 2.2 V	4	4.5	4.9	MHz
		V_{CC} = 3 V	4.4	4.9	5.4	
S_{Rsel}	$S_R = f_{Rsel+1} / f_{Rsel}$	V_{CC} = 2.2 V/3 V	1.35	1.65	2	
S_{DCO}	$S_{DCO} = f_{(DCO+1)} / f_{(DCO)}$	V_{CC} = 2.2 V/3 V	1.07	1.12	1.16	
D_t	Temperature drift, R_{sel} = 4, DCO = 3, MOD = 0 (see Note 2)	V_{CC} = 2.2 V	−0.31	−0.36	−0.40	%/°C
		V_{CC} = 3 V	−0.33	−0.38	−0.43	
D_V	Drift with V_{CC} variation, R_{sel} = 4, DCO = 3, MOD = 0 (see Note 2)	V_{CC} = 2.2 V/3 V	0	5	10	%/V

NOTES: 1. The DCO frequency may not exceed the maximum system frequency defined by parameter processor frequency, $f_{(System)}$.
2. This parameter is not production tested.

XT1 输入引脚自身的电容

crystal oscillator, LFXT1 oscillator (see Note 1)

PARAMETER		TEST CONDITIONS		MIN	NOM	MAX	UNIT
C_{XIN}	Integrated input capacitance	XTS=0; LF oscillator selected, V_{CC} = 2.2 V/3 V			12		pF
		XTS=1; XT1 oscillator selected, V_{CC} = 2.2 V/3 V			2		
C_{XOUT}	Integrated output capacitance	XTS=0; LF oscillator selected, V_{CC} = 2.2 V/3 V			12		pF
		XTS=1; XT1 oscillator selected, V_{CC} = 2.2 V/3 V			2		
V_{IL}	Input levels at XIN	V_{CC} = 2.2 V/3 V (see Note 2)	XTS = 0 or 1 XT1 or LF modes	V_{SS}		0.2 × V_{CC}	V
V_{IH}			XTS = 0, LF mode	0.9 × V_{CC}		V_{CC}	
			XTS = 1, XT1 mode	0.8 × V_{CC}		V_{CC}	

NOTES: 1. The oscillator needs capacitors at both terminals, with values specified by the crystal manufacturer.
2. Applies only when using an external logic-level clock source. Not applicable when using a crystal or resonator.

XT2 输入引脚自身的电容

crystal oscillator, XT2 oscillator (see Note 1)

PARAMETER		TEST CONDITIONS	MIN	NOM	MAX	UNIT
C_{XIN}	Integrated input capacitance	V_{CC} = 2.2 V/3 V		2		pF
C_{XOUT}	Integrated output capacitance	V_{CC} = 2.2 V/3 V		2		pF
V_{IL}	Input levels at XIN	V_{CC} = 2.2 V/3 V (see Note 2)	V_{SS}		0.2 × V_{CC}	V
V_{IH}			0.8 × V_{CC}		V_{CC}	V

NOTES: 1. The oscillator needs capacitors at both terminals, with values specified by the crystal manufacturer.
2. Applies only when using an external logic-level clock source. Not applicable when using a crystal or resonator.

实验三　I/O 操作

实验目的

控制单片机 I/0

实验要求

熟练掌握对 MSP430 的 I/O 的方向，高低，中断，中断触发的控制，以及 I/O 与功能管脚的选择。

实验内容

[1] 设置 I/0 的控制寄存器，点亮 P4.0~P4.3 的 LED 灯。
首先需要将 P4.0~P4.3 的方向寄存器 P4DIR 设置为输出，P4DIR |= 0x0f;
然后设置 P4.0~P4.3 的输出寄存器 P4OUT 设置为低，P4OUT &= ~(0x0f);

[2] 尝试从 I/O 的读入状态
需要把 I/O 口设置为输入模式
然后读取 PxIN 寄存器

[3] 通过其他 I/O 的输入来控制 LED 灯的亮和灭

实验注意事项

[1] 单片机的输出模式，使用的推挽级输出，具有很强的驱动能力。和 MSC51 单片机的上拉输出不一样。

[2] 把引脚设置为输出模式，则高电平基本接近电源电压，低电平就相当于接地。

[3] 当把引脚和其他设备和单片机相接的时候需要注意电平匹配和方式引脚冲突

[4] 电平匹配
虽然 MSP430 单片机也能接收 5V 的电平，但是强烈建议连接到单片机的高电平电压不要超过 3.3V
如果实在需要 MSP430 单片机接收 5V 的电平，而不想增加额外的器件时，请在引脚输入线上串联 1K~10K 的电阻，方式引脚烧毁。

[5] 引脚冲突
如果和 MSP430 单片机的相连的器件也具有很强的输入和输出能力的话。
需要注意，如果 MSP430 单片机引脚输出为高电平，而对方引脚输出为低电平，则 MSP430 单片机引脚会输出大电流，则可能回烧毁。

实验四　中断服务程序

实验目的

响应中断

实验要求

熟练掌握编写中断服务程序，响应和处理中断请求。

实验内容

[1] 中断服务程序的格式

```
// P2 中断服务子程序
#pragma vector = PORT2_VECTOR
__interrupt void P2_IRQ(void)
{
  switch(P1IFG)
  {
    case 0x01: // 引脚0
    case 0x02: // 引脚1
    case 0x04: // 引脚2
    case 0x08: // 引脚3
    case 0x10: // 引脚4
    case 0x20: // 引脚5
    case 0x40: // 引脚6
    case 0x80: // 引脚7
  }
}
```

中断服务程序最主要的是中断向量的设置

[2] 用 P2.0~P2.3 的引脚中断，控制 P4.0~P4.3 的 LED 闪烁

设置 P2.0~P2.3 为输入状态

设置 P2.0~P2.3 开中断

设置 P2.0~P2.3 中断为上升沿中断

开全局中断_EINT();

设置 P4.0~P4.3 为输出模式

写 P2 中断服务子程序，如果某引脚有中断，则改变相对应的 LED 状态。

实验注意事项

[1] 在中断服务程序里面不要占用大量的时间，如果有大量的数据需要处理，则可以使用在中断服务程序中设置标志变量，然后在主程序里面判断标志变量值，然后处理的方法。

[2] 中断服务程序只有在模块中断允许开启和全局中断开启时才有效

[3] 有的多个中断源使用一个中断向量，则需要在进中断的时候用 switch~case 语句判断相关寄存器到底是哪个中断源发出的中断请求。

[4] I/O 口里面只有 P1 和 P2 口能响应中断，设计硬件电路的时候需要注意。

实验五　超低功耗模式

实验目的

实现单片机工作在超低功耗模式

实验要求

熟练掌握 MSP430 超低功耗模式和中断一起使用。

实验内容

[1] 了解超低功耗模式
了解不同的超低功耗模式，所关闭的模块情况。

[2] 了解利用中断由超低功耗模式下唤醒，处理中断请求，然后回到超低功耗模式
使用__low_power_mode_0();进入 LMP0 模式
使用__low_power_mode_1();进入 LMP1 模式
使用__low_power_mode_2();进入 LMP2 模式
使用__low_power_mode_3();进入 LMP3 模式
使用__low_power_mode_4();进入 LMP4 模式
在中断服务程序的结尾使用__low_power_mode_off_on_exit();在中断结束时回到正常模式下。

[3] 在上个实验的基础上，使 P2.0~P2.3 的引脚在没有中断产生的时候，处于某种超低功耗模式下。然后由 P2.0~P2.3 的中断唤醒，改变 LED 状态，再回到原有的超低功耗模式下。

实验注意事项

[1] 当用__low_power_mode_0();进入超低功耗模式下时，程序指针始终指向__low_power_mode_0();语句的下一条语句。

[2] 理论上退出超低功耗模式语句，需要写在中断程序的末尾结束处。

[3] 在 LPM3 和 LPM4 中响应中断时，会出现时钟频率变慢的情况，这是由于 LPM3 和 LPM4 下 DCO 的直流发生器被关闭，唤醒后 DCO 需要一段时间恢复。

附表

由 LPM3 唤醒的时间延迟

wake-up LPM3

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
$t_{(LPM3)}$	Delay time	V_{CC} = 2.2 V/3 V, $f_{DCO} \geq f_{DCO43}$			6	µs

各种超低功耗模式的耗电情况

MSP430F15x/16x supply current into AV_{CC} + DV_{CC} excluding external current (AV_{CC} = DV_{CC} = V_{CC})

PARAMETER		TEST CONDITIONS		MIN	NOM	MAX	UNIT
$I_{(AM)}$	Active mode, (see Note 1) $f_{(MCLK)}$ = $f_{(SMCLK)}$ = 1 MHz, $f_{(ACLK)}$ = 32,768 Hz XTS=0, SELM=(0,1)	T_A = −40°C to 85°C	V_{CC} = 2.2 V		330	400	µA
			V_{CC} = 3 V		500	600	
	Active mode, (see Note 1) $f_{(MCLK)}$ = $f_{(SMCLK)}$ = 4,096 Hz, $f_{(ACLK)}$ = 4,096 Hz XTS=0, SELM=3	T_A = −40°C to 85°C	V_{CC} = 2.2 V		2.5	7	µA
			V_{CC} = 3 V		9	20	
$I_{(LPM0)}$	Low-power mode, (LPM0) $f_{(MCLK)}$ = 0 MHz, $f_{(SMCLK)}$ = 1 MHz, $f_{(ACLK)}$ = 32,768 Hz XTS=0, SELM=(0,1) (see Note 1)	T_A = −40°C to 85°C	V_{CC} = 2.2 V		50	60	µA
			V_{CC} = 3 V		75	90	
$I_{(LPM2)}$	Low-power mode, (LPM2), $f_{(MCLK)}$ = $f_{(SMCLK)}$ = 0 MHz, $f_{(ACLK)}$ = 32.768 Hz, SCG0 = 0	T_A = −40°C to 85°C	V_{CC} = 2.2 V		11	14	µA
			V_{CC} = 3 V		17	22	
$I_{(LPM3)}$	Low-power mode, (LPM3) $f_{(MCLK)}$ = $f_{(SMCLK)}$ = 0 MHz, $f_{(ACLK)}$ = 32,768 Hz, SCG0 = 1 (see Note 2)	T_A = −40°C	V_{CC} = 2.2 V		1.1	1.6	µA
		T_A = 25°C			1.1	1.6	
		T_A = 85°C			2.2	3.0	
		T_A = −40°C	V_{CC} = 3 V		2.2	2.8	
		T_A = 25°C			2.0	2.6	
		T_A = 85°C			3.0	4.3	
$I_{(LPM4)}$	Low-power mode, (LPM4) $f_{(MCLK)}$ = 0 MHz, $f_{(SMCLK)}$ = 0 MHz, $f_{(ACLK)}$ = 0 Hz, SCG0 = 1	T_A = −40°C	V_{CC} = 2.2V / 3 V		0.1	0.5	µA
		T_A = 25°C			0.2	0.5	
		T_A = 85°C			1.3	2.5	

NOTES: 1. Timer_B is clocked by $f_{(DCOCLK)}$ = 1 MHz. All inputs are tied to 0 V or to V_{CC}. Outputs do not source or sink any current.
2. WDT is clocked by $f_{(ACLK)}$ = 32,768 Hz. All inputs are tied to 0 V or to V_{CC}. Outputs do not source or sink any current. The current consumption in LPM2 and LPM3 are measured with ACLK selected.

实验六　定时器 A 的基本操作（一）

实验目的

实现单片机使用定时器 A 进行定时操作。

实验要求

熟练掌握对 MSP430 定时器 A 模块增计数模式的操作。

实验内容

[1] 了解定时器 A 的模块所需要的设置
时钟源设置，运行模式设置，中断设置，周期设置等

[2] 设置 TACCR0 的值
设置 TACCTL0 的中断 CCIE 位。
设置 TACTL 里面的时钟源设置
把定时器 A 的运行模式由停止模式改为增计数模式，则 TA 开始运行
TAR 在 TA 运行器件进行增计数操作，当 TAR 的值等于 TACCR0 的值时会触发 TIMERA0_VECTOR 中断向量

[3] 在 TIMERA0_VECTOR 中断向量的中断服务程序里面轮流改变 LED 的状态
使 LED 显示为跑马灯的效果。

[4] 使用 TACCR1 来实现上述效果。（使用 TIMERA1_VECTOR 中断向量）

实验注意事项

[1] TA 里面所涉及的中断有好多个，但是使用了 2 个中断向量
TACCR0 里的 CCIE 位，能触发 TIMERA0_VECTOR 中断向量
TACCR1 里的 CCIE 位，能触发 TIMERA1_VECTOR 中断向量
TACCR2 里的 CCIE 位，能触发 TIMERA1_VECTOR 中断向量
TACTL 里的 TAIE 位，能触发 TIMERA1_VECTOR 中断向量
通过 TAIV 来判断中断源来自那里

[2] 不要在 TA 运行中修改 TACCR0 的值，有可能引发不可预计的周期错误。
可以在 TACCR0 的中断里面修改。

[3] 由中断产生到进入中断需要一点时间，一般在 10 个 MCLK 左右

实验七　定时器 A 的基本操作（二）

实验目的

实现单片机使用定时器 A 进行定时操作。

实验要求

熟练掌握对 MSP430 定时器 A 模块连续计数模式的操作。

实验内容

[1] 了解连续计数模式和增计数模式在使用上和设置上的不同。

[2] 多中断触发练习
使用连续计数模式连续的触发 TACCR0，TACCR1，TACCR2 和 TA 的中断
设置 TACCR0 的值，开启 TACCR0 的中断
设置 TACCR1 的值，开启 TACCR1 的中断
设置 TACCR2 的值，开启 TACCR2 的中断
设置 TACTL 里面的 TAIE 中断开启
使 0 < TACCR0 < TACCR1 < TACCR2 < 65535
如果出现 TACCR0 中断则改变 P4.0 的 LED 状态
如果出现 TACCR1 中断则改变 P4.1 的 LED 状态
如果出现 TACCR2 中断则改变 P4.2 的 LED 状态
如果出现 TAIE 中断则改变 P4.3 的 LED 状态

[3] 观察 LED 闪烁顺序，总结中断触发时机的不同和规律。
如果 LED 闪烁过快，可以将 TACTL 的时钟源设置为 ACLK。

[4] 将工作模式设置为增减计数模式，然后观察中断产生时机。

实验注意事项

[1] TACTL 里的 TAIE 位的中断触发时机：TAIE 的中断在 TAR 回到 0 的时候产生

实验八　PWM 波形产生

实验目的

利用 TA 实现单片机的 PWM 波形输出

实验要求

掌握 MSP430 定时器 A 的高级操作。

实验内容

[1] 了解 PWM 的产生原理
如果把 TACCR0 设置在比较模式下，当 TAR=TACCR0 的时候，EQU0=1
如果 TACCTL0 里面设置了输出模式，则会在 TA0 的输出引脚上输出一个信号。
信号的样子由 TACCTL0 里面的输出模式来定义。

[2] 写一段程序验证 TA0 的输出。
TA0 引脚在 P1.1，需要将 P1.1 切换到功能脚模式，并且选择输出
当 TACCR0 中断的时候，在 TA0 引脚会有相应的电平输出

[3] 产生 PWM 波形
产生 PWM 波形需要 TACCR0 和 TACCR1 同时工作
TACTL 需要设置在增计数模式下
TACCR0 的值控制 PWM 的周期，TACCR1 的值控制占空比
所以一般说来 TACCR0 的值要大于 TACCR1
然后在 TACCTL1 里面选择你想要的输出模式，打开 TA1 引脚（P1.1）
运行后，观察波形的输出
在产生 PWM 的过程中不需要开中断

[3] 一般来说一个带有 3 路比较输出的 TA（TIMERA3 模块）只能输出 2 路 PWM 波形
一路是 TACCR0 和 TACCR1 工作，在 TA1 上输出波形
另一路是 TACCR0 和 TACCR2 工作，在 TA2 上输出波形
在 TA0 上输出的只是 TACCR0 的周期波形

实验注意事项

[1] TACCTLx 默认工作在比较模式下，只要设置了输出模式，并打开了引脚，就有输出。

[2] 比较输出模式下的 PWM 输出不是依靠中断来产生的，所以开中断不是必须的。

[3] TB 的工作和使用方法和 TA 一样，寄存器名字就是将 TA 改成 TB，例如 TACCR0，在 TB 里面就是 TBCCR0

[4] F169 的 TB 是（TIMERB7 模块）具有 7 个比较功能，可以产生 6 路 PWM

实验九　看门狗定时器

实验目的

设置基本时钟系统

实验要求

熟练掌握对 MSP430 基本时钟系统的操作和示众资源的分配。

实验内容

[1] 了解看门狗定时器的结构和原理。
看门狗定时器不能象 TA 一样任意的设置时间，而是选择几个固定的时间分频。
看门狗定时器可以工作在定时器模式和看门狗模式。
对看门狗定时器的寄存器操作需要密码，写密码为 0x5A，可以使用宏定义 WDTPW。

[2] 用做定时器。
设置 WDTCTL 选择时钟和分频，设置工作模式。
定时器的中断需要在 IE1 里面开启，由 IE1 里面的 WDTIE 位控制。

[3] 用做看门狗，则不用开中断。
如果在定时器计数完成前用 WDTCLR 位清除 WDTCNT 的计数值，则不会复位

[4] 用做定时器，实现 P4.0~P4.3 的 LED 跑马灯效果。
用做看门狗，按时喂狗（清除 WDTCNT 的计数值），防止复位。

实验注意事项

[1] 看门狗定时器的中断在 IE1 寄存器里面

[2] 可以使用头文件定义的看门狗设置
例如：
用 WDTCTL = WDT_ADLY_1000;　设置看门狗用做来自 ACLK 的 1 秒定时
用 WDTCTL = WDT_MRST_8;　　设置看门狗用做来自 SMCLK 的 8 豪秒复位
用 WDTCTL = WDT_ARST_1000;　设置看门狗用做来自 ACLK 的 1 秒复位

实验十　FLASH 的读写

实验目的

对 FLASH 的读写操作

实验要求

熟练掌握对 MSP430 自身的 FLASH 的读写访问。

实验内容

[1] 了解 FLASH 的结构和原理。
FLASH 被分做一个个 512 字节的“段”，里面有细分为 64 字节 1 个的“块”
FLASH 操作的时钟发生器
FLASH 只能对字节进行读和写操作，但是一旦写入就不能改写。
对 FLASH 的内容改写只有在擦除一“段”FLASH 后才能将改写后的内容写入。

[2] 对 FLASH 进行编程操作
选择适当的时钟源和分频因子；
如果 LOCK＝1，将其复位；
监视 BUSY 标志位，只有当 BUSY＝0 时才可以执行下一步；
如果写入单字节或单字，则设置 WRT＝1；
如果是块写或多字、多字节顺序写入，则设置 WRT＝BLKWRT＝1；
将数据写入选定地址时启动时序发生器，在时序发生器的控制下完成整个过程。

[3] 对 FLASH 的写入操作要做如下 4 件事：
对 FLASH 控制寄存器写入适当的控制位；
监视 BUSY 位；
写一个数据；
继续写直到结束。

[4] 可以通过 IAR 里自带的 MEMORY 监视 FLASH 的内容。
以检查写入的数据是否正确。

实验注意事项

[1] 时钟频率应该设置在 300K 左右，过高会导致写入数据出错，甚至会复位。

[2] 如果写入高字节口令码错误，将引发 PUC 信号；

[3] 在对 FLASH 操作期间读 FLASH 内容，将会引发 ACCFIG 状态位的设置；

[4] FLASH 操作需要较长的时间，可能引起看门狗定时器溢出
(建议在进行 FLASH 操作之前先停掉看门狗，等操作完成再打开看门狗)；

实验十一　串口操作

实验目的

用串口与计算机通信

实验要求

熟练掌握对 MSP430 的串口模块的基本操作。

实验内容

[1] 了解串口模块
了解串口控制寄存器，串口发送控制寄存器，串口接收控制寄存器，波特率寄存器，发送缓冲寄存器，接收缓冲寄存器。

[2] 最常用的串口初始化
在 U0CTL 里设置 CHAR 位，选择发送的 BIT 为 8BIT 模式
在 U0TCTL 设置 SSEL0 和 SSEL0 来选择时钟 ACLK 或者 SMCLK
在 U0BR1，U0BR0，U0MCTL 设置波特率参数
更改 U0CTL 中 SWRST 使其为 0，完成串口寄存器的设置
打开模块允许，在 ME1 寄存器中设置 UTXE0 和 URXE0 位来允许接收和发送
打开中断允许，在 IE1 寄存器里设置 URXIE0 和 URXIE0 位来允许接收和发送的中断
设置 P3SEL 将 P3.4 和 P3.5 设置为功能模式 引脚
最后_EINT();开全局中断

[3] 串口的发送
当串口初始化以后，向 TXBUF0 写入一个字节，则这个字节将被发送出去
通过检查 IFG1 寄存器中 UTXIFG0 标志，如果位为 1，则发送完成
IFG1 寄存器中 UTXIFG0 标志需要手动清除
所以串口的发送中断不是一定要开启的

[4] 串口的接收
串口的接收需要在中断中进行。进入中断后从 RXBUF0 寄存器中读取接受到的数据
在串口的接收中断里做的事情要尽可能的少，以便有时间接收下一个字节。

实验注意事项

[1] 串口的模块允许设置是必须的，在 ME1 寄存器中设置 UTXE0 和 URXE0 位来允许接收和发送
[2] 发送的中断标志需要手动清除（由于没有可以响应的中断，进中断则自动清除）
[3] 波特率的设置需要通过计算得到，常用的参数设置可以查表

实验十二　数模转换

实验目的

用 A/D 采样模拟信号

实验要求

掌握对 MSP430 的 ADC12 模块的基本操作。

实验内容

[1] 了解 ADC12 模块的结构和原理
了解内部电压源，通道选择器，采样保持器，采样保持定时器，AD 转换内核，通道寄存器等之间的关系。
对于使用 A/D 模块需要注意下面的问题
A/D 转换时钟的选择，参考电压的选择，采集通道和顺序的控制，以及采样的速率。
对于寄存器需要了解每个寄存器所设置的内容，以及设置内容的作用。

[2] 初始化 ADC12 模块实现单通道单次采集
设置 ADC12CTL0，内核开启，启动内部基准，选择 2.5V 基准，设置采样保持时间
设置 ADC12CTL1，时钟源为内部震荡器，触发信号来自采样定时器，转换地址为 ADC12MCTL0
然后设置 ADC12MCTL0，需要设置的内容是该通道所使用的参考电压和对应的模拟输入通道
上述设置完成后就可以准备启动转换了

[3] 但通道单次数模转换
将 ADC12CTL0 寄存器里的 ENC 和 ADC12SC 同时置位，则开始转换。
此时查看 ADC12IFG 寄存器中代表 ADC12CTL0 的 BIT0 位的值，让值为 1 则说明转换结束
转换结束后，可以由 ADC12MEM0 中读取转换的结果
这样的做法是软件查询转换结果。

[4] 在第 2 项中，增加在 ADC12IE 中 BIT0 的中断允许打开，然后开启全局中断
将 ADC12CTL0 寄存器里的 ENC 和 ADC12SC 同时置位，则开始转换。
转换开始后，可以进入低功耗模式，则程序停止。
等到 ADC12 的中断被响应，在中断服务程序中读取 ADC12MEM0 的转换结果，然后在中断结束后退出低功耗模式。继续执行后面的处理程序等。
这样的做法是中断查询转换结果。

实验注意事项

[1] 设置具体的转换模式以及启始通道，默认是 ADC12MEM0 所对应的通道。
[2] 输入模拟信号范围和参考电压应该匹配。内部可以产生 1.5V 或者 2.5V 的参考电压。
[3] 选择启动信号，常见的方法是 ENC 和 ADC12SC 同时置位，高级的有利用 TA 等的输出触发的
[4] 关注结束信号，不管是用软件查询转换结果，还是中断查询转换结果。
[5] 存放转换数据，转换完成后就存放在 ADC12MEMx 中，及时读出来存到其他变量里面。

实验十三　模数转换

实验目的

用 D/A 输出模拟信号

实验要求

掌握对 MSP430 的 DAC12 模块的基本操作。

实验内容

[1] 了解 DAC12 模块的结构和原理。
DAC12 可以选择内部或者外部的参考电压，其中内部的参考电压来自与 ADC12 中的内部参考电压发生器生成的 1.5V 或者 2.5V 参考电压。
用 DAC12RES 位选择 DAC12 的 8 位或者 12 位精度。
DAC12 输出引脚和断口 P6 以及 ADC12 模拟输入复用。
DAC12 支持二进制数或者 2 的补码输入。
DAC12 可以自动校正偏移量。
对于寄存器需要了解每个寄存器所设置的内容，以及设置内容的作用。

[2] 效验 DAC。
由于 DAC12 的偏移误差，所以初始化 DAC12 模块首先需要效验 DAC。
将 DAC12_0CTL 寄存器中的 DAC12CALON 位置 1 来启动 DAC 效验，并查询该位，如果发现该位变为 0，则说明效验完成。

[3] 初始化寄存器
在 DAC12_0CTL 寄存器中，选择输入缓冲器中速中电流，输出缓冲器中速中电流，
选择 DAC12 内核为 12 位 DAC，满电压输出为内基准，自动更新数据。
然后将 DAC12_0CTL 寄存器中的 DAC12ENC 置位启动 DAC 模块

[4] 更改输出电压
随时更改 DAC12_0DAT 的值，则会在几个 us 内输出心的电压。

实验注意事项

[1] 如果 DAC12LSEL_0 时此句可以省的时候，启动 DAC12 只需要往 DAC12_0DAT 写入数据即可。

[2] 设置 DAC12 的位数和满量程电压，如果参考电压为 2.5V，量程设置为 3 倍参考电压的话，则理论最大输出电压为 7.5V，但是实际的满量程电压最高为 AVcc

[3] 在 MSP430F169 单片机上，DAC12 的 0 通道使用的是 A6，1 通道使用的是 A7 的引脚。注意如果使用了 DAC12 的 2 个通道，A6 和 A7 就不能使用。

实验十四　暖气计费表

实验目的

基于 MSP430 系列单片机的暖气计费表

实验要求

[1] 通过测上/下水温度和流量，计算出供热量并显示；
[2] 可人为控制或设定供暖时间；
[3] 具备现场分段标定功能；
[4] 可通过 IC 卡交费后使用；
[5] 可通过仪表接口构成局域网管理网络。

实验内容

[1] 根据实验的要求要设计出合理的电路原理图以及电路印刷板图。
[2] 暖气计费需要知道以下几种参数：上水温度，下水温度和流量，所以需要选择适合的温度传感器和流量计。
[3] 为了可人为控制或设定供暖时间，则需要一个实时时钟，并且为了控制阀门，需要一个标准的控制信号输出。
[4] 对于现场分段标定功能，则需要充分利用 FLASH 的特点，在程序编写上实现该功能。
[5] 为了能使用 IC 卡交费，则需要设计与 IC 卡的接口电路
[6] 通过仪表接口构成局域网管理网络，可以使用最新的 M-BUS 仪表网络组网。

实验注意事项

[1] 由于涉及程序量比较的大，所以程序的编写需要简明扼要，清晰第一，效率第二。
[2] 合适传感器的选择，温度传感器的选择比较多，流量传感器需要根据管道特点来选择。
[3] 为了控制阀门，需要一个标准的控制信号输出。这个需要去做调查。
[4] IC 卡有很多种型号，需要选择一个性价比较好的。
[5] 组建网络，需要考虑程序上的负担，以及网络的性能，通讯距离，可靠性。

实验十五　智能化气表

实验目的

基于 MSP430 系列单片机的智能化气表

实验要求

[1] 通过测量管道气体流量，计算出用气量并显示；
[2] 可检测天然气泄漏并声光报警；
[3] 具备现场分段标定功能；
[4] 可通过 IC 卡交费后使用；
[5] 可通过仪表接口构成局域网管理网络。

实验内容

[1] 根据实验的要求要设计出合理的电路原理图以及电路印刷板图。
[2] 暖气计费需要知道以下几种参数：气体流量，由于所测量的气体有可能是可燃性气体，所以需要选择适合的气体流量计。
[3] 检测天然气泄漏并声光报警，需要有检测可燃性气体浓度的电路，选择合适的气体传感器，利用单片机自带的模数转换模块实现起来并不难。
[4] 对于现场分段标定功能，则需要充分利用 FLASH 的特点，在程序编写上实现该功能。
[5] 为了能使用 IC 卡交费，则需要设计与 IC 卡的接口电路
[6] 通过仪表接口构成局域网管理网络，可以使用最新的 M-BUS 仪表网络组网。

实验注意事项

[1] 由于涉及程序量比较的大，所以程序的编写需要简明扼要，清晰第一，效率第二。
[2] 合适的传感器的选择，需要根据测量气体特性和输出信号特性来选择。
[3] IC 卡有很多种型号，需要选择一个性价比较好的。
[4] 组建网络，需要考虑程序上的负担，以及网络的性能，通讯距离，可靠性。