

华东师范大学计算机科学技术系

DEP.OF COMPUTER SCIENCE

EAST CHINA NORMAL UNIVERSITY

MSP430系列超低功耗16位单片机原理与应用

<http://www.Microcontrol.cn> 微控设计网

中国MSP430单片机专业网站



嵌入式系统概述

I 始于微型机时代的嵌入式应用

计算机—1946年

20世纪70年代，出现微处理器

将微型机迁入到对象体系中

计算机失去了原有的形态和通用的计算机功能

教材结构

- | 概述
- | MSP430单片机结构
- | MSP430指令系统与程序设计
- | MSP430单片机片内外围模块
- | MSP430单片机应用

概述

- | 单片微型计算机
 - √ 单片机的概念
 - √ 单片机的特点
 - √ 单片机的应用

 - | MSP430系列单片机
 - √ MSP430系列单片机的特点
 - √ MSP430系列单片机的发展与应用

 - | MSP430应用选型
 - √ MSP430系列单片机命名规则
 - √ MSP430系列单片机选项
 - | 思考题与习题
-

微处理器的发展

- 一是朝着面向数据运算、信息处理等功能的系统机方向发展。系统机以速度快、功能强、存储量大、软件丰富、输入/输出设备齐全为主要特点，采用高级语言编程，适用于数据运算、文字信息处理、人工智能、网络通信等场合。
- 另一方面，在有些应用领域中，如智能化仪器仪表、电讯设备、自动控制设备、汽车乃至家用电器等，要求的运算、控制功能相对并不很复杂，但对体积、成本、功耗等的要求却比较苛刻。为适应这方面的需求，产生了一种将中央处理器、存储器、I/O接口电路以及连接它们的总线都集成在一块芯片上的计算机，即所谓的单片微型计算机，简称单片机（Single Chip Microcomputer）。单片机在设计上主要突出了控制功能，调整了接口配置，在单一芯片上制成了结构完整的计算机，因此，单片机也称为微控制器（MCU）

单片机的特点

- | 小巧灵活、成本低、易于产品化，它能方便的组装成各种智能式控制设备以及各种智能仪器仪表。
- | 面向控制，能针对性的解决从简单到复杂的各类控制任务，因而能获得最佳性能价格比。
- | 抗干扰能力强，适应温度范围宽，在各种恶劣环境下都能可靠的工作，这是其他机型无法比拟的。
- | 可以很方便的实现多机和分布式控制。使整个系统的效率和可靠性大为提高。

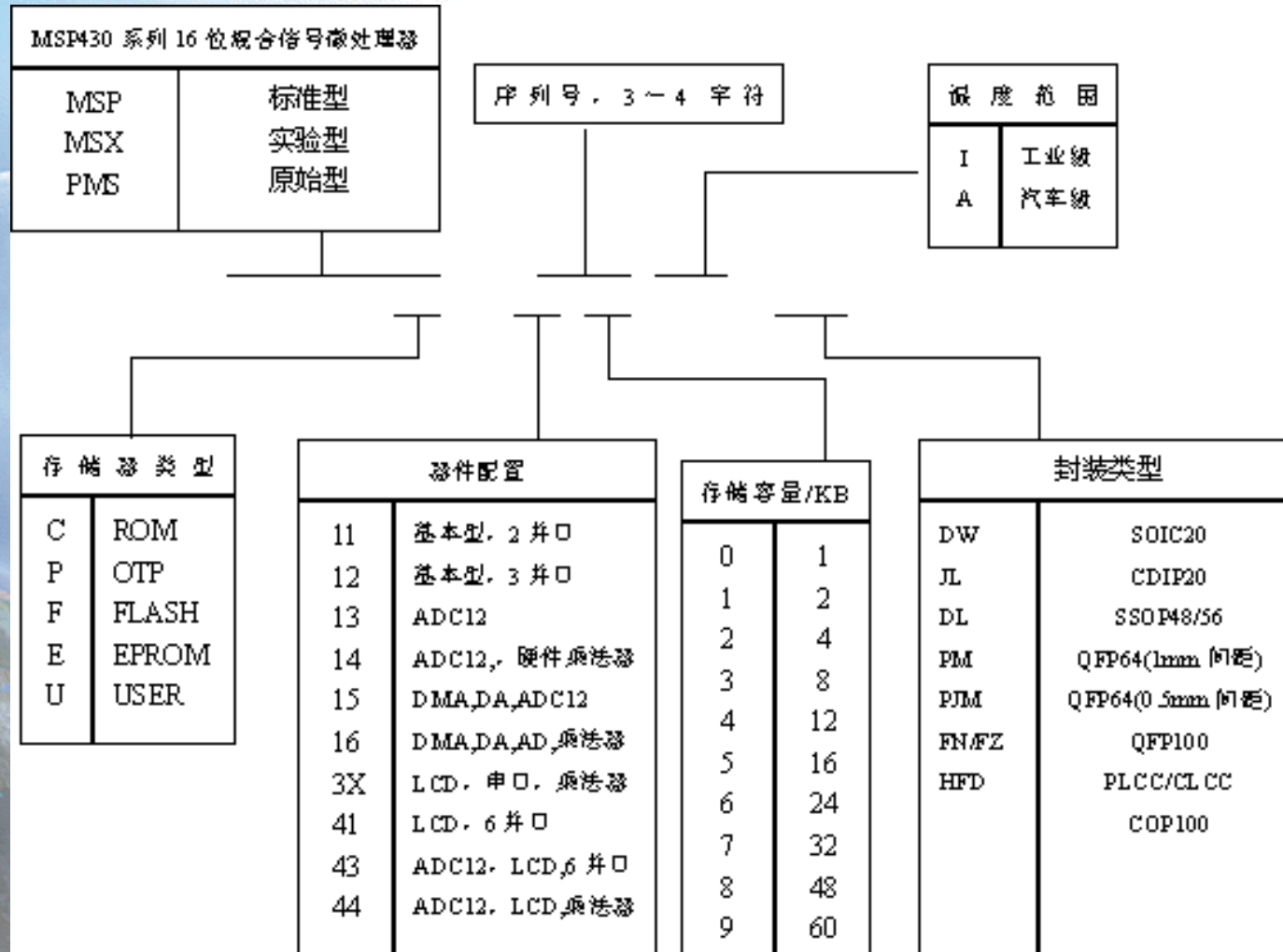
单片机的应用

- | 工业控制：单片机的结构特点决定了它特别适用于各种控制系统。它既可以作单机控制器，有可作为多级控制的前沿处理机用于控制系统，应用领域相当广泛。例如：用于各种机床控制、电机控制、工业机器人、各种生产线、各种过程控制、各种检测系统等。在军事工业中：导弹控制、鱼类制导控制、智能武器装置、航天导航系统等。在汽车工业中：点火控制、变速器控制、防滑刹车、排气控制等。
- | 智能化的仪器仪表：单片机用于包括温度、湿度、流量、流速、电压、频率、功率、厚度、角度、长度、硬度、元素测定等和各类仪器仪表中，使仪器仪表数字化、智能化、微型化，功能大大提高。
- | 日常生活中的电器产品：单片机可用于电子秤、录像机、录音机、彩电、洗衣机、高级电子玩具、冰箱、照相机、家用多功能报警器等。
- | 计算机网络与通信方面：单片机可用BIT BUS、CAN、以太网等构成分布式网络系统，还可以用于调制解调器、各种智能通信设备（例如小型背负式通信机、列车无线通信等）、无线遥控系统等。
- | 计算机外部设备：单片机可以用于温氏硬盘驱动器、微型打印机、图形终端、CRT显示器等。

MSP430系列单片机特点

- | 超低功耗
- | 强大的处理能力
- | 高性能模拟技术及丰富的片上外围模块
- | 系统工作稳定
- | 方便高效的开发环境

MSP430系列单片机命名规则



MSP430系列产品

型号	F L A S H	R A M	A D C	D A C	D M A	L C D 段 数	U S A R T	比较 器	硬 件 乘 法 器	定 时 器	封 装 类 型	I/ O
MSP430F1101A	1k	128B	Slope				软件	√		2	20SOIC, TSSOP	14
MSP430F1111A	2k	128B	slope				软件	√		2	20SOIC, TSSOP	14
MSP430F1121A	4k	256B	slope				软件	√		2	20SOIC, TSSOP	14
MSP430F1122	4k	256B	10bit				软件			2	20SOIC, TSSOP	14
MSP430F1132	8k	256B	10bit				软件			2	20TSSOP	14
MSP430F1222	4k	256B	10bit				硬件			2	28SOIC, TSSOP	22
MSP430F123	8k	256B	slope				硬件	√		2	28SOIC, TSSOP	22
MSP430F1232	8k	256B	10bit				硬件1			2	28SOIC, TSSOP	22
MSP430F133	8k	256B	12bit				硬件1	√		3	64LQFP	48
MSP430F135	16k	256B	12bit				硬件1	√		3	64LQFP	48

MSP430F147	32k	1KB	12bit				硬件2	√	√	3	64LQFP	48
MSP430F1471	32k	1KB	slope				硬件2	√	√	3	64LQFP	48
MSP430F148	48k	1KB	12bit				硬件2	√	√	3	64LQFP	48
MSP430F1481	48k	2KB	slope				硬件2	√	√	3	64LQFP	48
MSP430F149	60k	2KB	12bit				硬件2	√	√	3	64LQFP	48
MSP430F1491	60k	2KB	slope				硬件2	√	√	3	64LQFP	48
MSP430F155	16k	512B	12bit	12bit	√		硬件1	√		3	64LQFP	48
MSP430F156	24k	512B	12bit	12bit	√		硬件1	√		3	64LQFP	48
MSP430F157	32k	1KB	12bit	12bit	√		硬件1	√		3	64LQFP	48
MSP430F167	32k	1KB	12bit	12bit	√		硬件2	√	√	3	64LQFP	48
MSP430F168	48k	2KB	12bit	12bit	√		硬件2	√	√	3	64LQFP	48
MSP430F169	60k	2KB	12bit	12bit	√		硬件2	√	√	3	64LQFP	48

MSP430F412	4K	256B	slope			96	软件	✓		3	64LQFP	48
MSP430F413	8K	256B	slope			96	软件	✓		3	64LQFP	48
MSP430FE423	8K	256B	16bit			128	硬件1	✓	✓	3	64QFP	14
MSP430FE425	16K	512B	16bit			128	硬件1	✓	✓	3	64QFP	14
MSP430FE427	32K	1KB	16bit			128	硬件1	✓	✓	3	64QFP	14
MSP430FW423	8K	512B	16bit			96	软件	✓		3	64QFP	48
MSP430FW425	16K	512B	16bit			96	软件	✓		3	64QFP	48
MSP430FW427	32K	1KB	16bit			96	软件	✓		8	64LQFP	48
MSP430F435	16K	512B	12bit			160	硬件1	✓		5	80,100 LQFP	48
MSP430F436	24K	1KB	12bit			160	硬件1	✓		5	80,100 LQFP	48
MSP430F437	32K	1KB	12bit			160	硬件1	✓		5	80,100 LQFP	48
MSP430F447	32K	1KB	12bit			160	硬件2	✓	✓	5	100LQFP	48
MSP430F448	48K	2KB	12bit			160	硬件2	✓	✓	5	100LQFP	48
MSP430F449	60K	2KB	12bit			160	硬件2	✓	✓	5	100LQFP	48

关于2系列

MSP430F1xx Versus MSP430F2xx

	1xx	2xx
CPU Clock	8MHz	16MHz
Wakeup	6us	1us
Stand-by	<2uA	<1uA
BOR	Some	ALL
Flash ISP	2.7V	2.2V
P1/2	-	Pull-up / Down
Oscillator	+20%	+2.5%
OscFault	HF	HF/LF
Watchdog	SW	SW Invalid Address Clock Fault
BSL	2^256	Hackproof

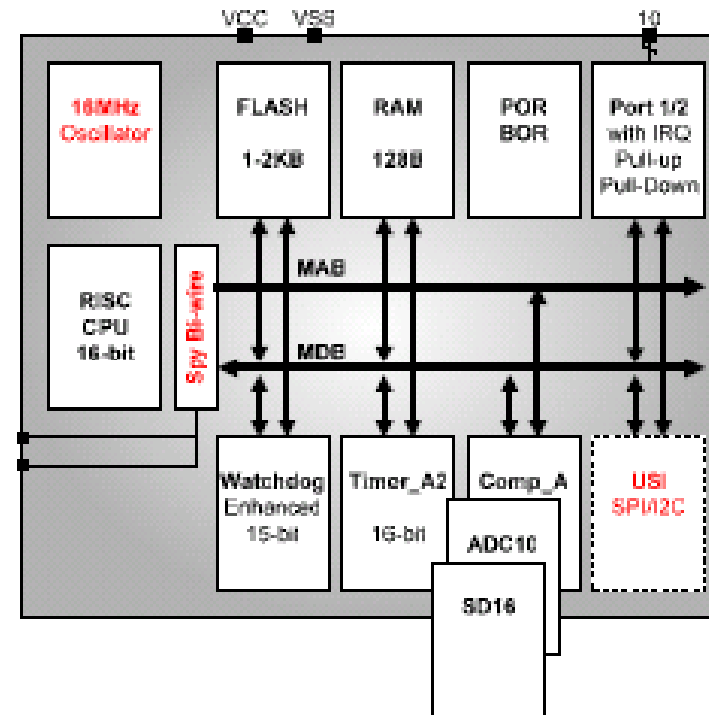
2X faster

1/2 power

Better

New MSP430F20xx

- **Tiny 14-pin packaging**
QFN 4x4mm, TSSOP, PDIP
- **Fast 0-to-16 MIPS <1us**
- **Flexible**
 - <<1uA RTC and 200uA/MIPS
 - 1.8 to 3.6V operation
 - Zero-power BOR
 - Failsafe OSC/WDT+
 - ± 2.5% programmable DCO
 - 2.2V Flash ISP
 - Pull-up/down port resistors
 - 128B RAM and 1/2kB ISP Flash
 - Spy Bi-Wire emulation
 - Universal Serial Interface (USI)
 - Comparator/10-bit/16-bit ADC

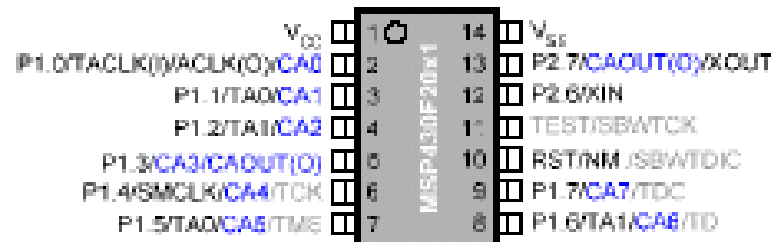
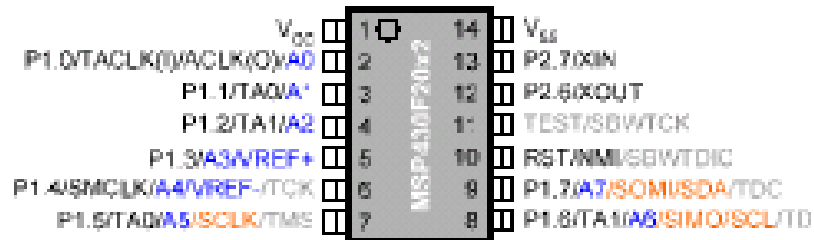
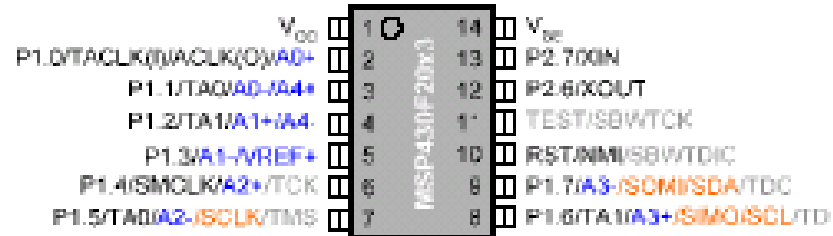


F20xx Family

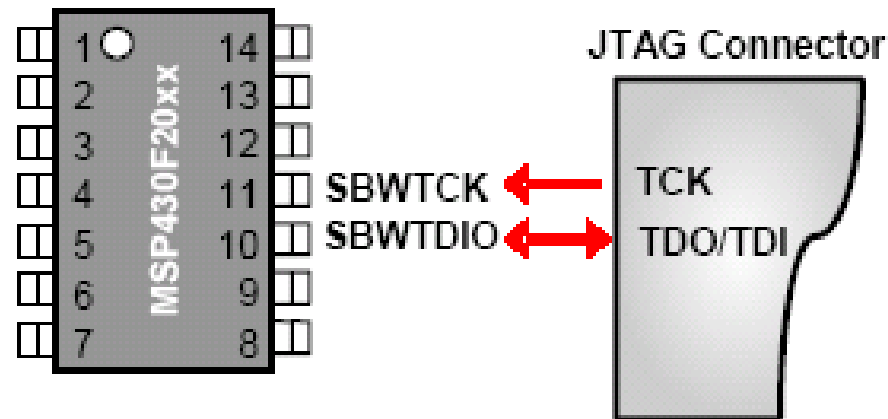
Device	Flash	RAM	Timer	Analog	USI	1ku Price
F2001	1KB	128B	WDT+/A2	Comp_A		\$0.55
F2011	2KB	128B	WDT+/A2	Comp_A		0.70
F2002	1KB	128B	WDT+/A2	ADC10	✓	0.99
F2012	2KB	128B	WDT+/A2	ADC10	✓	1.15
F2003	1KB	128B	WDT+/A2	SD16_A	✓	1.49
F2013	2KB	128B	WDT+/A2	SD16_A	✓	1.65

- **Worlds lowest power MCU**
- **Tiny, fast and flexible**
- **Non-compromised architecture**

F20xx Family Compatibility



F20xx Easy In-System Emulation

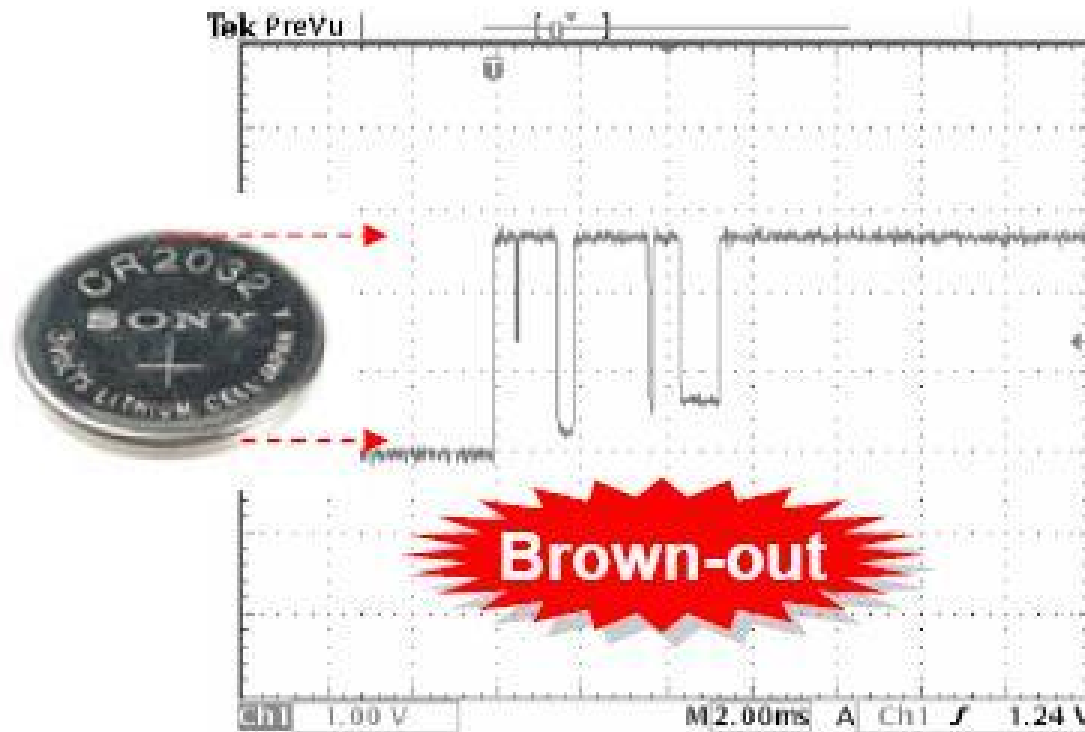


- **Unobtrusive 2-wire Spy Bi-Wire Emulation**

- Full-speed, single step
- Hardware breakpoints
- Clock control

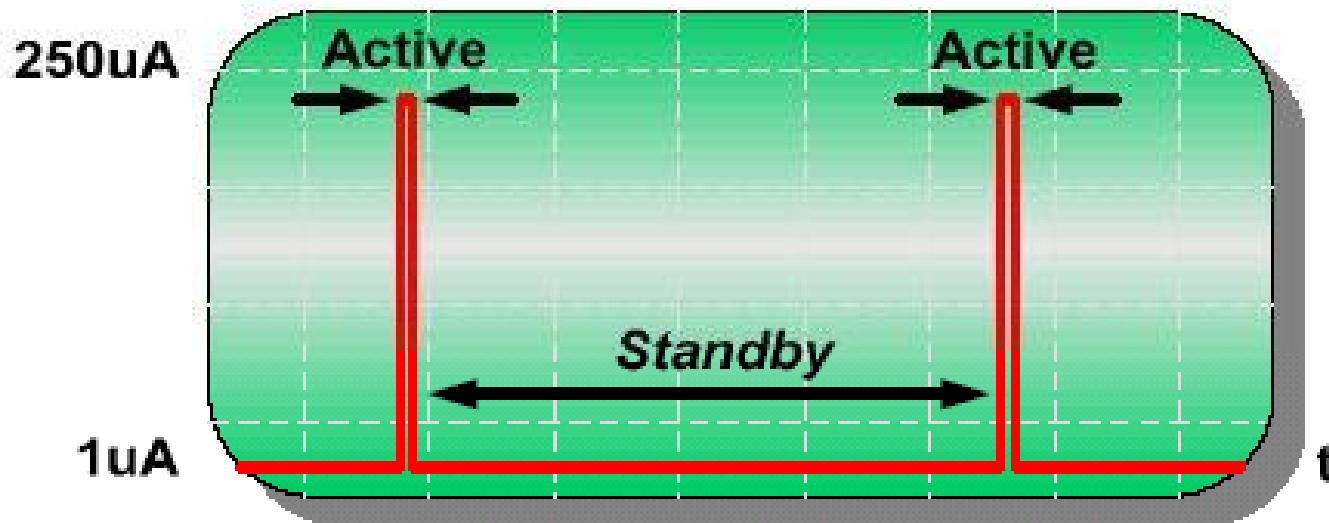
- **Compatible with installed MSP430 IDEs**

All **F2xx** Have Zero Power BOR



- MSP430 BOR is always-on and zero-power

Achieving ***Ultra-low Power***



- Max time in Ultra-low Power ***LPM3*** standby mode
- ***Active*** Performance on-demand
- Minimum active duty cycle

第一章习题

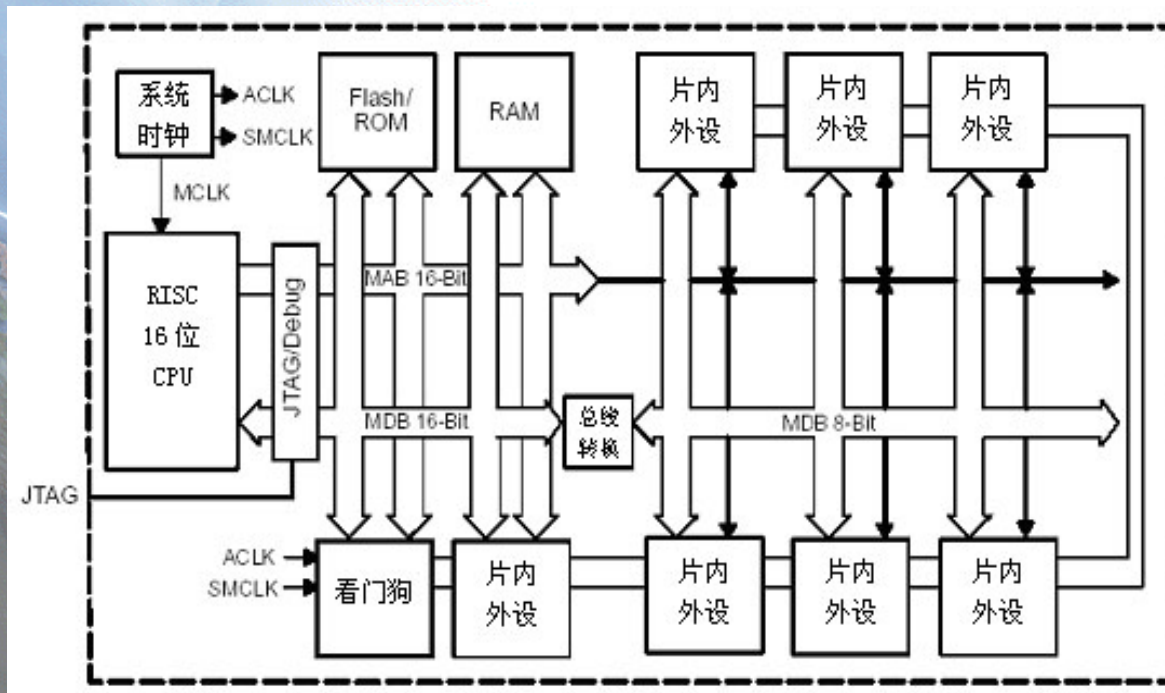
- | 微处理器的发展方向是什么？
- | 单片机的概念是什么？
- | 单片机和我们通常所用的微型计算机有什么区别和联系？
- | 单片机常见的应用领域有哪些？
- | 如何理解MSP430系列单片机的“单片”解决能力？
- | MSP430系列单片机最显著特性是什么？
- | 如何理解MSP430系列单片机的低功耗特性？
- | 为什么MSP430系列单片机特别适用于电池供电和手持设备？
- | 如何理解MSP430系列单片机的强大处理能力？在开发环境方面，MSP430系列单片机和传统单片机相比，有哪些显著优势？
- | 构成MSP430系列单片机的各类存储器有什么特点？各自适用于哪些场合？
- | MSP430系列单片机应用选型的依据是什么？

MSP430单片机结构

- | MSP430单片机结构概述
- | MSP430系列产品
 - √ 无LCD驱动系列产品
 - √ 有LCD驱动系列产品
- | MSP430 CPU结构和特点
- | MSP430存储器和地址空间
 - √ 程序存储器
 - √ 数据存储器
 - √ 外围模块寄存器
- | 思考题与习题

MSP430结构

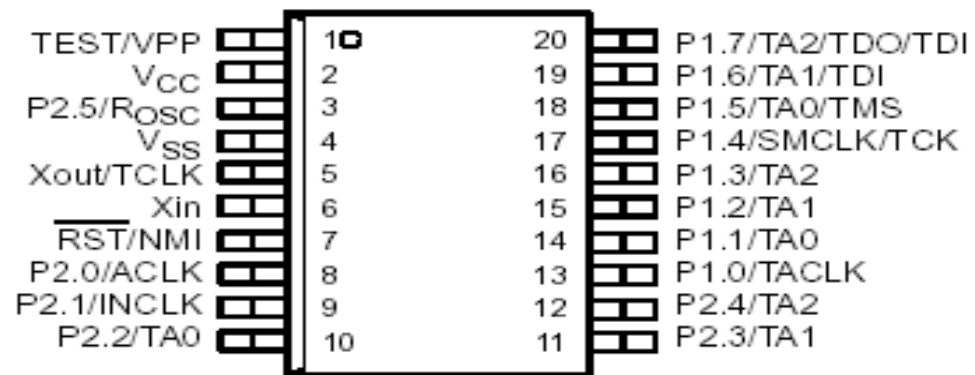
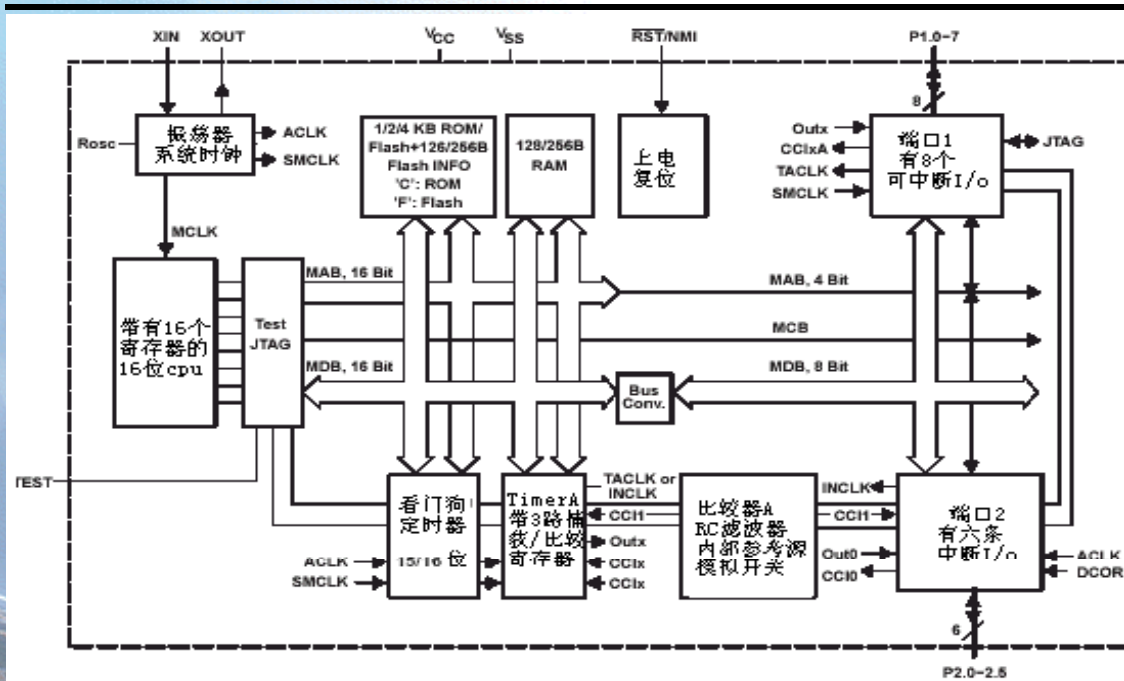
- 16位CPU通过总线连接到存储器和外围模块。
- 直接嵌入仿真处理，具有JTAG接口。
- 能够降低功耗，降低噪声对存储器存取的影响。
- 16位数据宽度，数据处理更为有效。



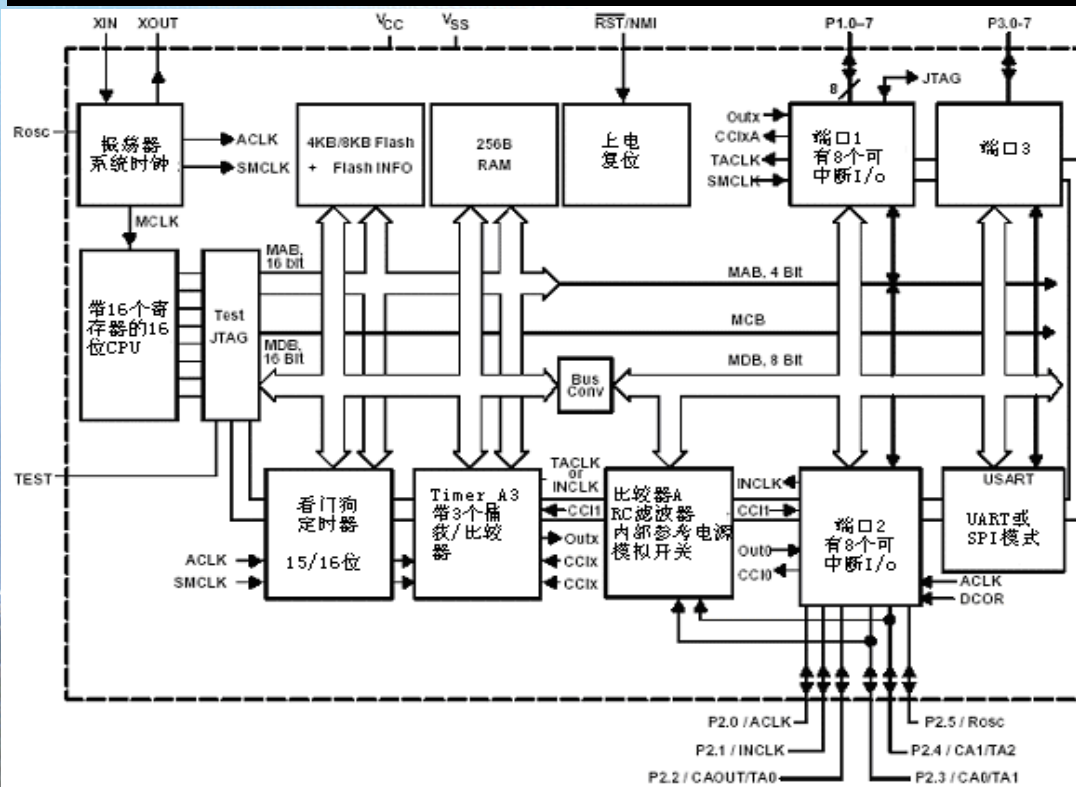
MSP430系列单片机包含以下主要功能部件：

- 丨 **CPU：** MSP430系列单片机的CPU和通用微处理器基本相同，只是在设计上采用了面向控制的结构和指令系统。MSP430的内核CPU结构是按照精简指令集和高透明的宗旨而设计的，使用的指令有硬件执行的内核指令和基于现有硬件结构的仿真指令。这样可以提高指令执行速度和效率，增强了MSP430的实时处理能力。
 - 丨 **存储器：** 存储程序、数据以及外围模块的运行控制信息。有程序存储器和数据存储器。对程序存储器访问总是以字形式取得代码，而对数据可以用字或字节方式访问。其中MSP430各系列单片机的程序存储器有ROM、OTP、EPROM和FLASH型。
 - 丨 **外围模块：** 经过MAB、MDB、中断服务及请求线与CPU相连。MSP430不同系列产品所包含外围模块的种类及数目可能不同。它们分别是以下一些外围模块的组合：时钟模块、看门狗、定时器A、定时器B、比较器A、串口0、1、硬件乘法器、液晶驱动器、模数转换、数模转换、端口、基本定时器、DMA控制器等。
-

MSP430X11X系列

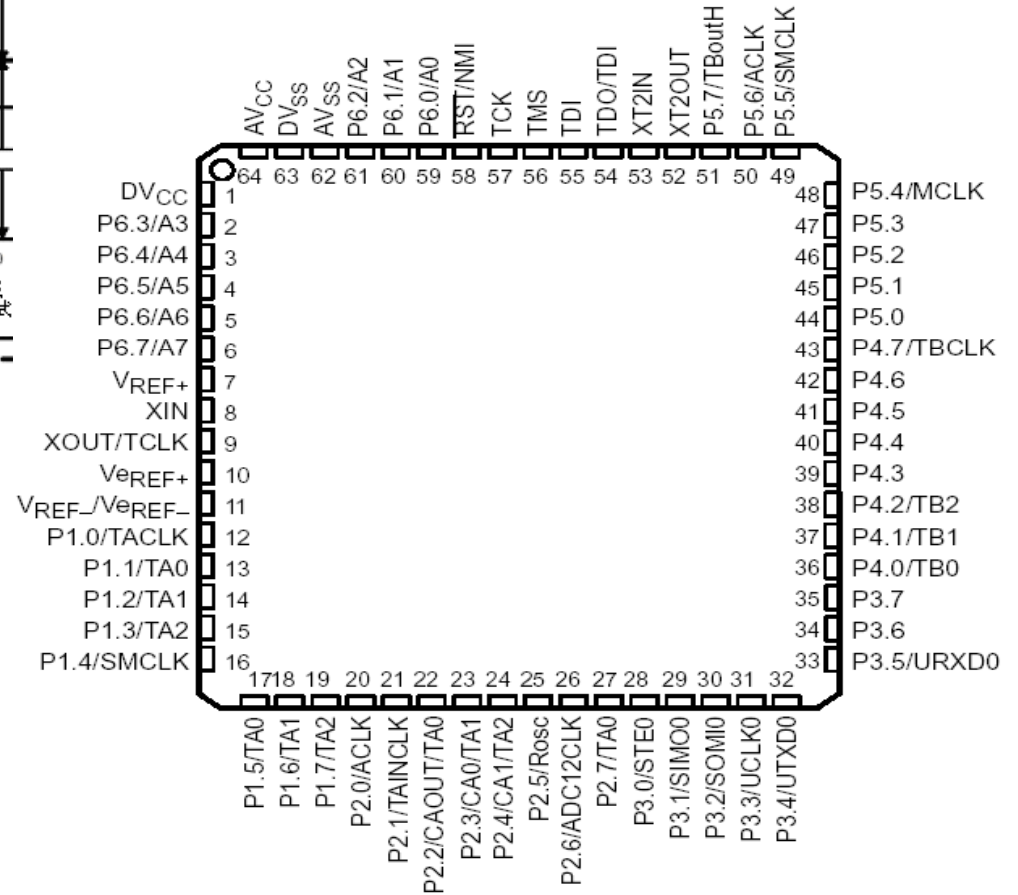
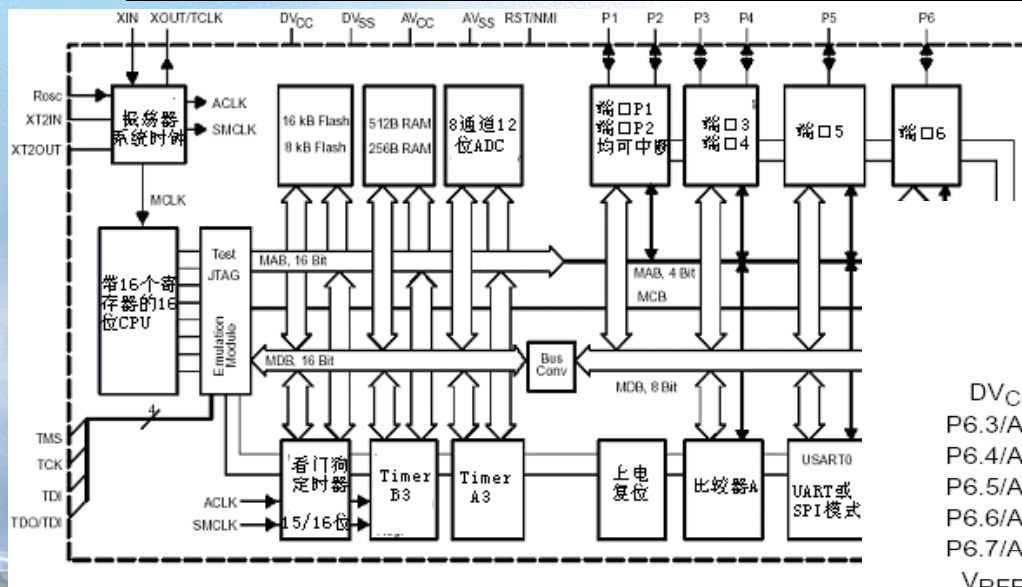


MSP430X12X系列

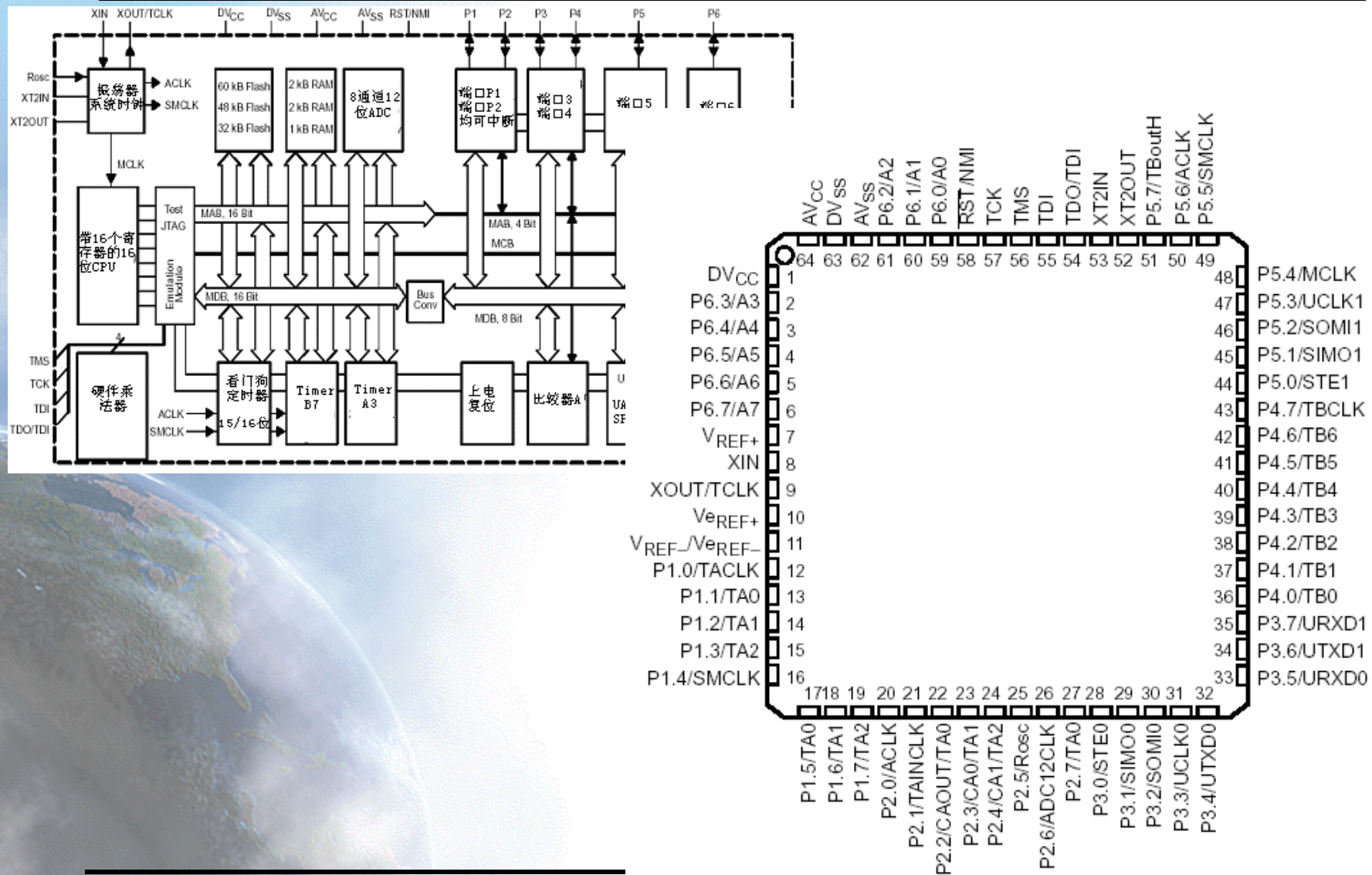


TEST	1	28	P1.7/TA2/TDO/TDI
V _{CC}	2	27	P1.6/TA1/TDI
P2.5/R _{osc}	3	26	P1.5/TA0/TMS
V _{SS}	4	25	P1.4/SMCLK/TCK
XOUT	5	24	P1.3/TA2
XIN	6	23	P1.2/TA1
RST/NMI	7	22	P1.1/TA0
P2.0/ACLK	8	21	P1.0/TACLK
P2.1/INCLK	9	20	P2.4/CA1/TA2
P2.2/CAOUT/TA0	10	19	P2.3/CA0/TA1
P3.0/STE0	11	18	P3.7
P3.1/SIM00	12	17	P3.6
P3.2/SOMIO	13	16	P3.5/URXD0
P3.3/UCLK0	14	15	P3.4/UTXD0

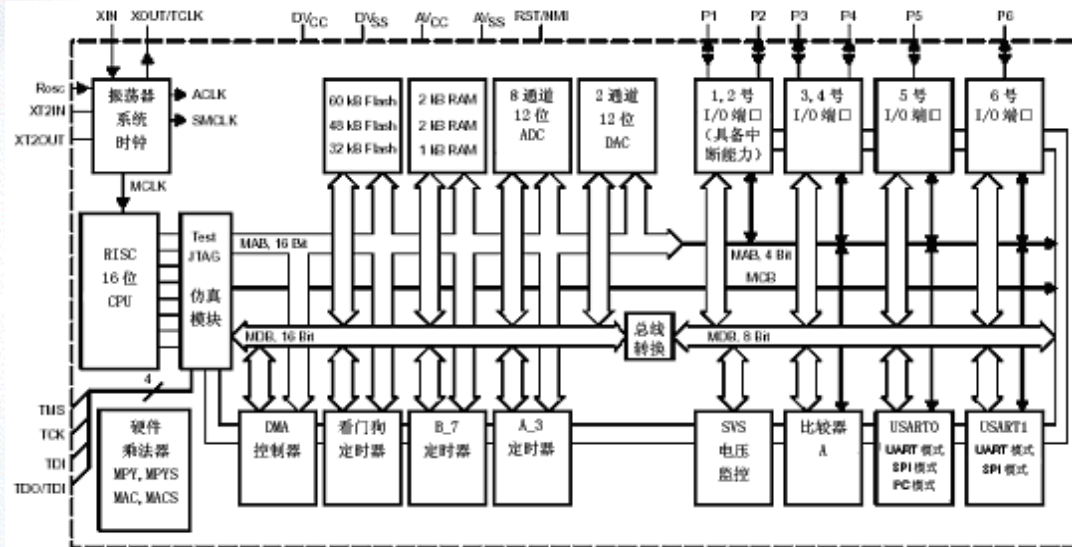
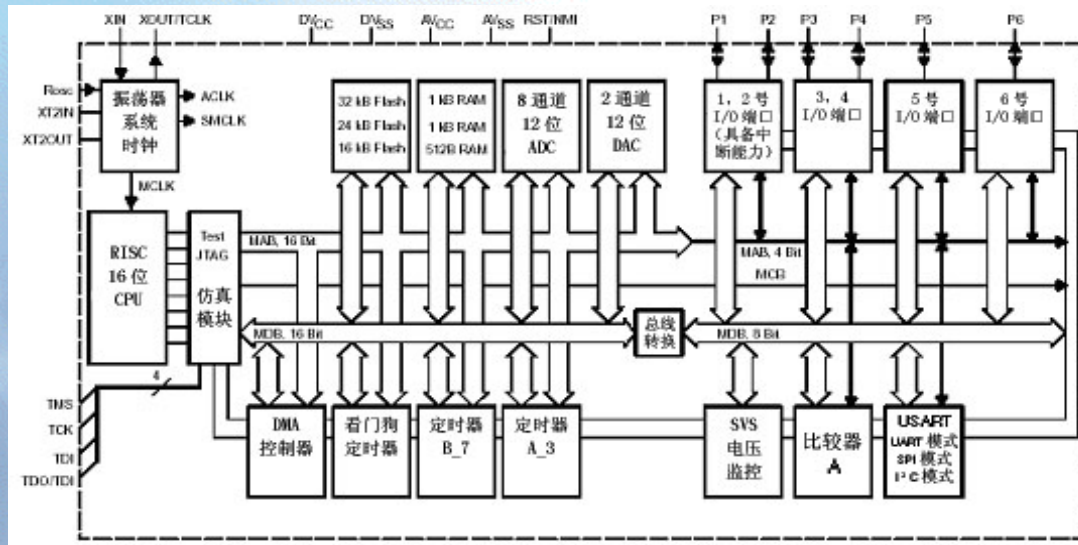
MSP430X13X系列



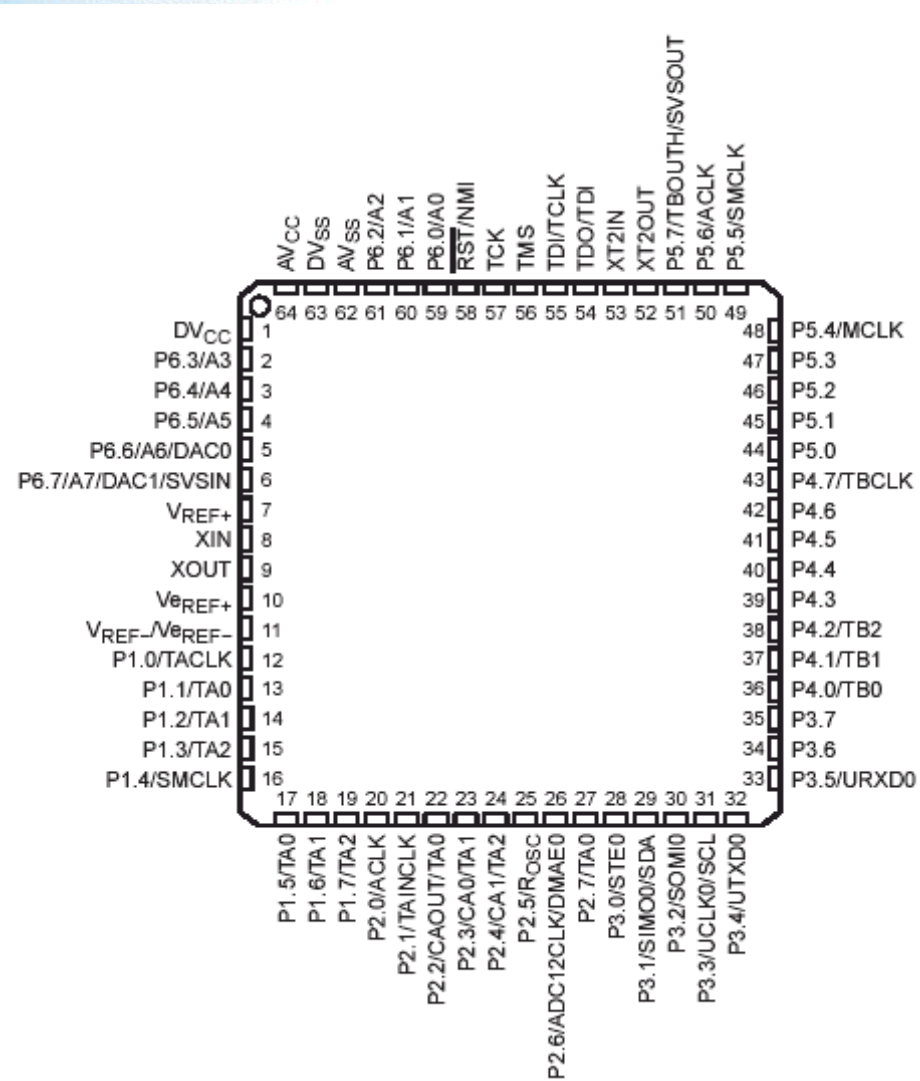
MSP430X14X系列



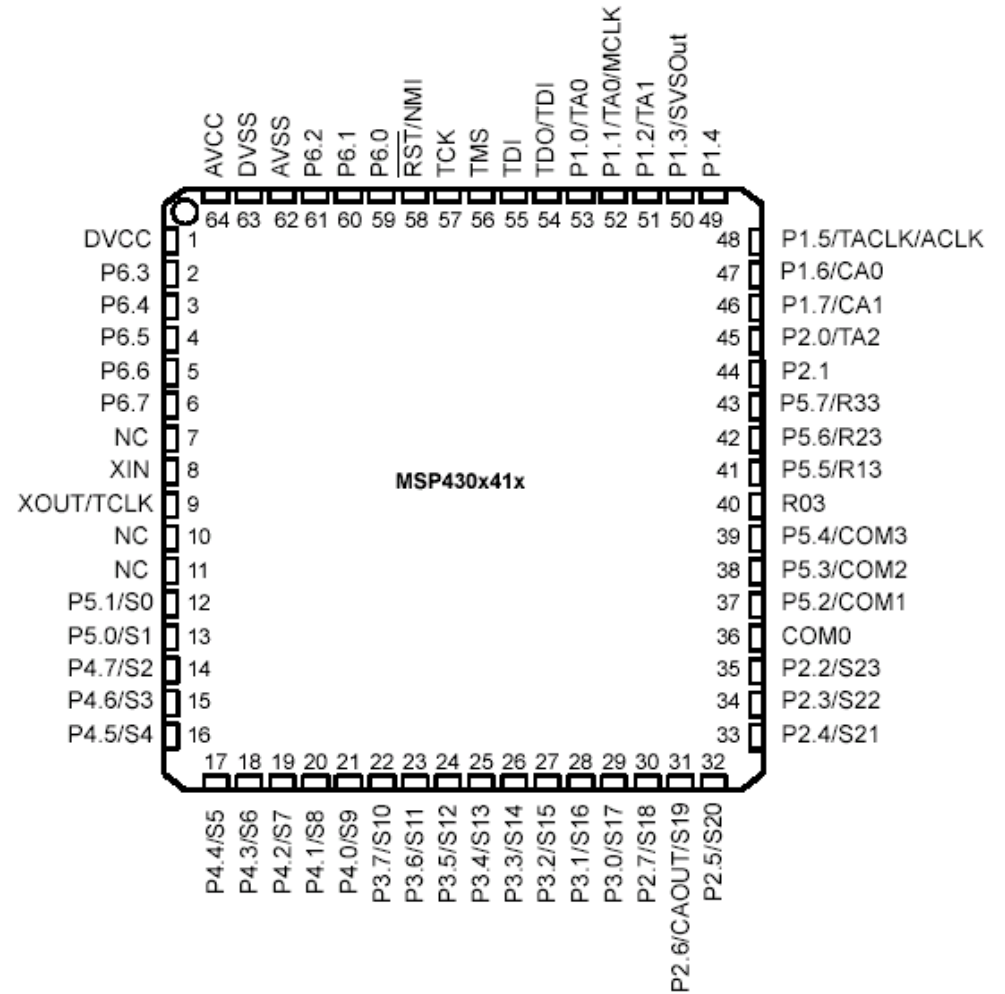
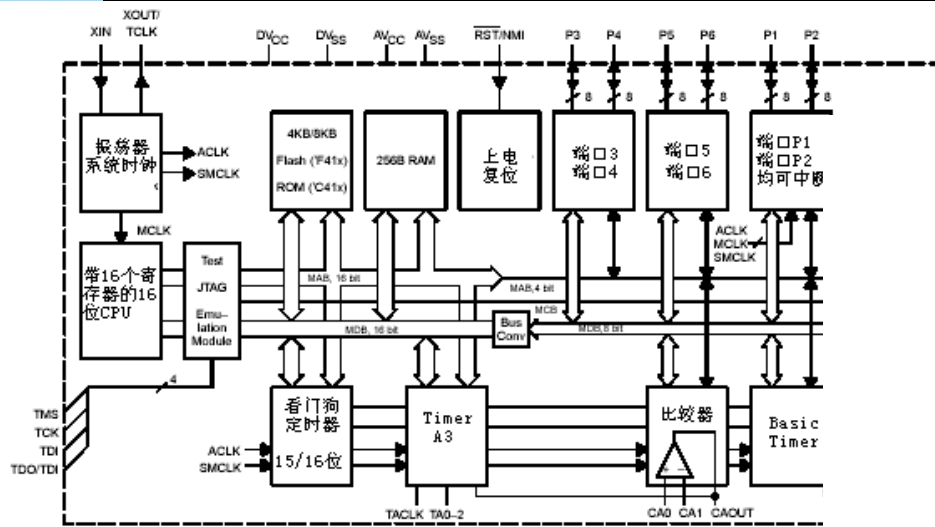
MSP430F15X/F16(1)X 系列



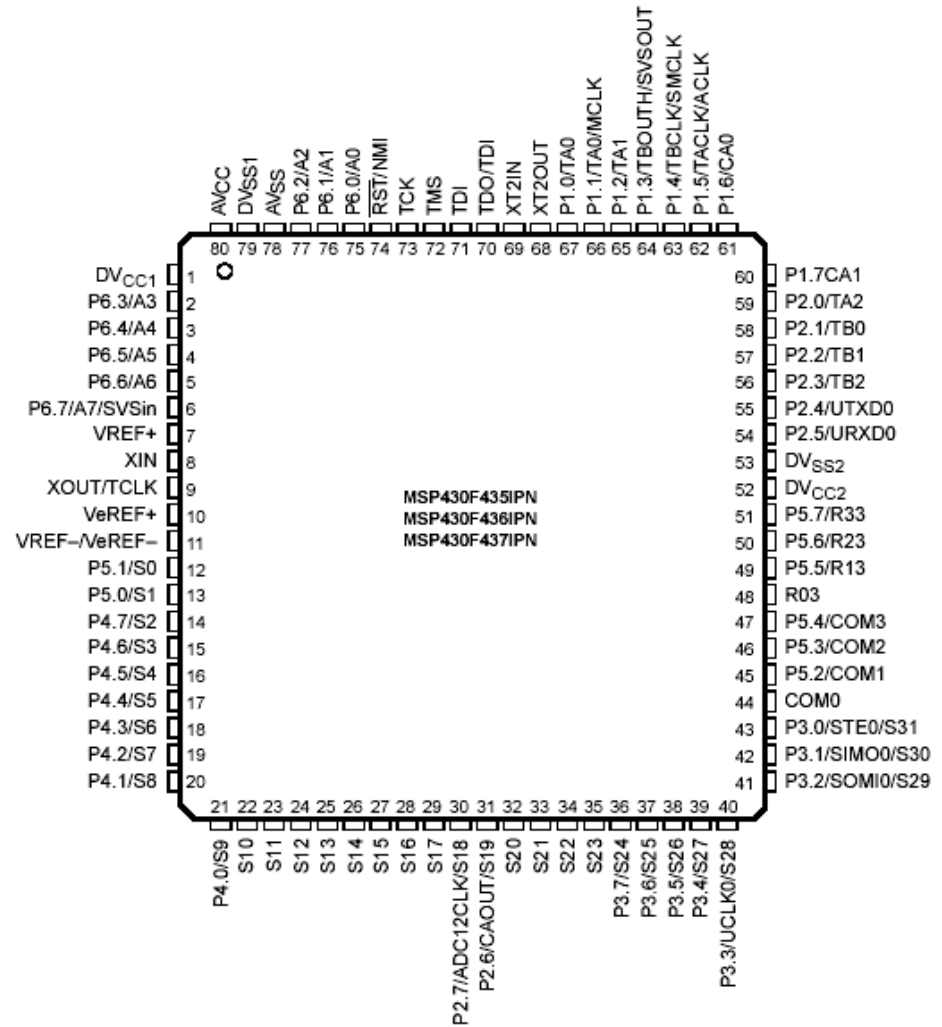
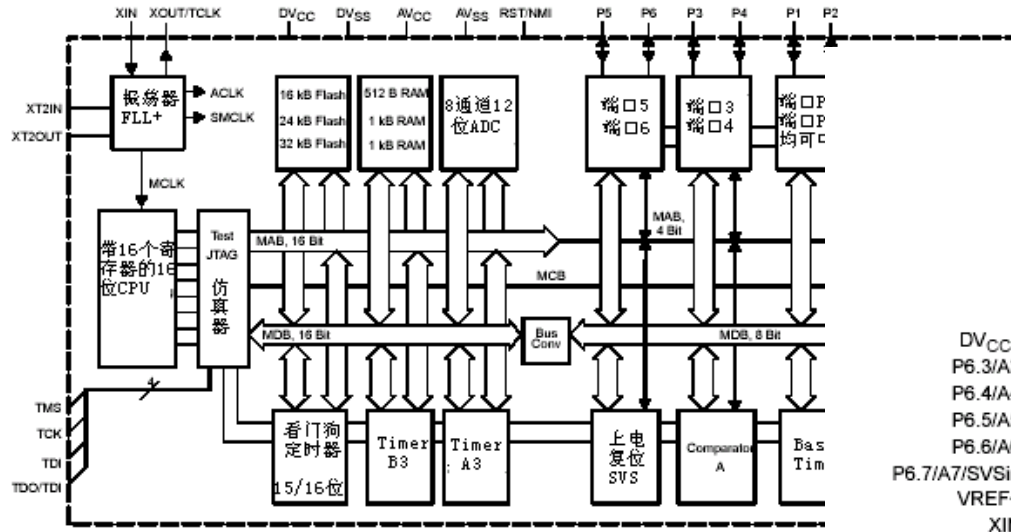
MSP430F15X/F16(1)X 系列



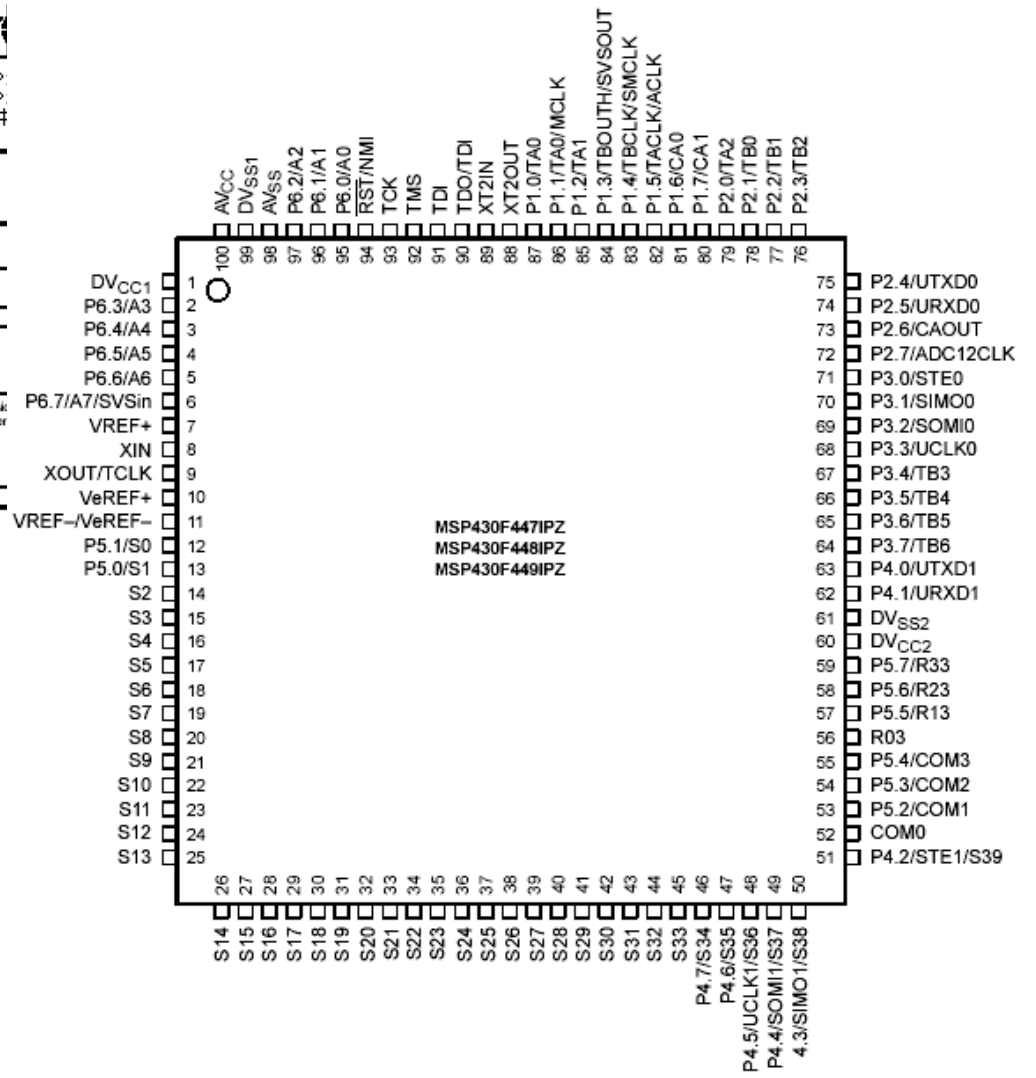
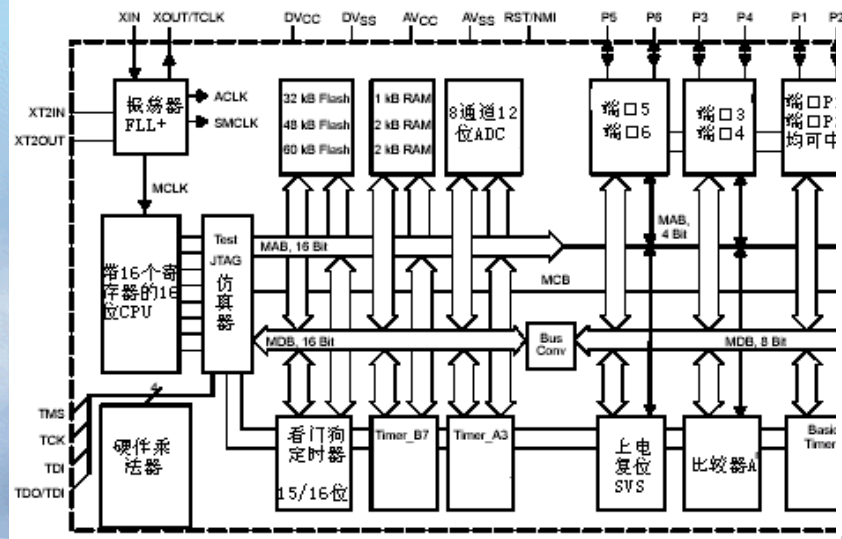
MSP430X41X系列



MSP430F43X系列



MSP430F44X系列



MSP430 CPU的主要特征

- | 精简指令集高度正交化
- | 寄存器资源丰富
- | 寄存器操作为单周期
- | 16位地址总线
- | 常数发生器
- | 直接的存储器到存储器访问

MSP430 CPU的16个寄存器

简 写	功 能
R0	程序计数器PC
R1	堆栈指针SP，指向堆栈的栈顶
R2	状态寄存器SR/常数发生器CG1
R3	常数发生器CG2
R4	通用寄存器R4
...
R15	通用寄存器R15

状态寄存器SR /R2/常数发生器1

状态标志：

- Ø C :进位标志
- Ø Z :零标志
- Ø N :负标志
- Ø V :溢出标志

控制标志

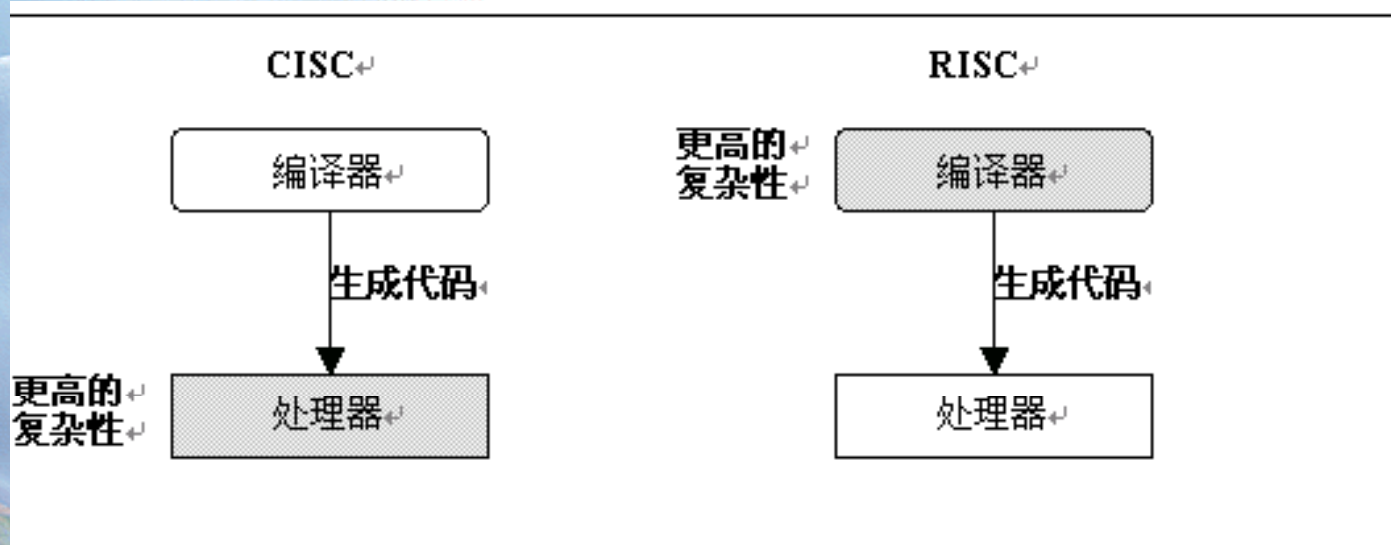
- GIE :中断标志位
- CPUOFF :CPU控制位
- OscOFF:晶振控制位
- SCG0,SCG1:时钟控制位

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

常数发生器CG1和CG0

- | CLR dst
- | MOV #0, dst
- | DEC dst
- | SUB #1, dst

MSP430 CPU----RISC



CISC 对 RISC，CISC 强调硬件的复杂性；RISC 注重编译器的复杂性

MSP430存储器结构和地址空间



0FFFFH	中断向量表
0FFED0H 0FFDFH	程序存储器 跳转控制表 数据表等
	引导存储器 (FLASH)
	数据存储器
0200H 01FFH	16 位外围模块
0100H 0FFH	8 位外围模块
010H 0FH	特殊功能寄存器
00H	

本章小结

- 在结构上MSP430系列单片机集成了一部计算机的各个基本组成部分。虽然其工作原理与普通微机并无差异，但MSP430系列单片机在结构上更加突出了体积小、功能强、面向控制的特点，具有很高的性能价格比。
 - MSP430系列单片机由CPU、存储器和外围模块组成，这些部件通过内部地址总线、数据总线和控制总线相连构成单片微机系统。
 - MSP430的内核CPU结构是按照精简指令集的宗旨来设计的。具有丰富的寄存器资源、强大的处理控制能力和灵活的操作方式。
 - MSP430的存储器结构采用了统一编址方式，可以使得对外围模块寄存器的操作象普通的RAM单元一样方便、灵活。MSP430存储器的信息类型丰富，并具有很强的系统外围模块扩展能力。
-

思考题与习题

1. MSP430系列FLASH型单片机有什么优势？
2. MSP430X1XX系列单片机的主要特征是什么？
3. MSP430X4XX系列单片机的主要特征是什么？
4. MSP430X1XX和MSP430X4XX系列单片机有什么区别和联系？
5. MSP430F15/16X和其他型号系列单片机相比有哪些特点？
6. MSP430F15X和MSP430F16X有什么区别和联系？
7. 单片机和典型微型计算机在结构上有什么区别？
8. MSP430系列单片机内部包含哪些主要功能部件？
9. MSP430系列单片机的CPU有哪些“面向控制”的特性？
10. MSP430系列单片机的CPU寄存器有什么特点？应该如何正确应用？
11. MSP430系列单片机的直接寻址能力为多少字节？
12. MSP430系列单片机CPU状态寄存器的作用是什么？各位的含义是什么？
13. MSP430系列单片机CPU常数发生器的作用是什么？
14. MSP430系列单片机存储器的组织方式是什么？
15. MSP430系列单片机存储器的组织方式与CPU的RISC结构有什么关系？
16. 为什么说MSP430系列单片机还有很大的系统外围模块扩展能力？

MSP430指令系统与程序设计

- | 指令系统概述
 - | 寻址方式
 - | 指令系统介绍
 - √ 数据传送类指令
 - √ 数据运算类指令
 - √ 逻辑操作指令
 - √ 位操作指令
 - √ 跳转与程序流程的控制类指令
 - | 程序设计
 - √ 程序设计基础
 - √ 汇编语言程序设计
 - √ C语言程序设计
 - | 思考题与习题
-

MSP430单片机片内外围模块

- | 时钟模块
 - | 低功耗结构
 - | MSP430各种端口
 - | 定时器
 - | MSP430液晶驱动模块
 - | 硬件乘法器
 - | FLASH存储器模块
 - | 比较器A
 - | DMA控制器
 - | MSP430系列通用串行通信模块的异步模式
 - | MSP430系列通用串行通信模块的同步模式
 - | MSP430系列通用串行通信模块的I2C模式
 - | MSP430模数转换模块
 - | MSP430数模转换模块
 - | 思考题与习题
-

MSP430单片机应用

- | MSP430常用接口设计
 - √ 键盘接口
 - √ LED显示接口
 - √ 液晶显示接口
 - √ 常用LED驱动功率接口
 - √ 继电器型驱动接口
 - | MSP430片内外围模块应用
 - √ 定时器
 - √ 比较器
 - √ SPI同步操作
 - √ A/D D/A 和DMA
 - | MSP430单片机应用设计举例
 - √ 自校准变频电源
 - √ 超低功耗手持式电子斜度计/加速度计
 - | 思考题与习题
-

Msp430时钟模块

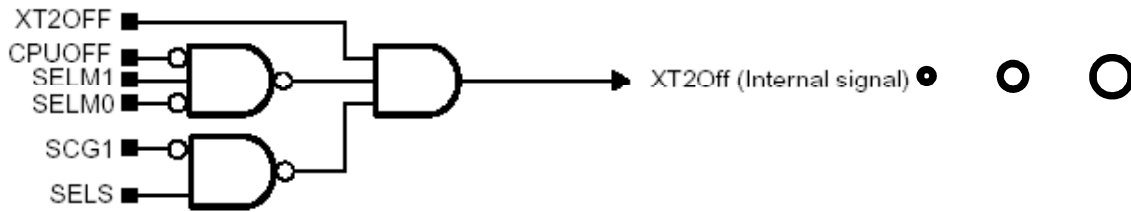
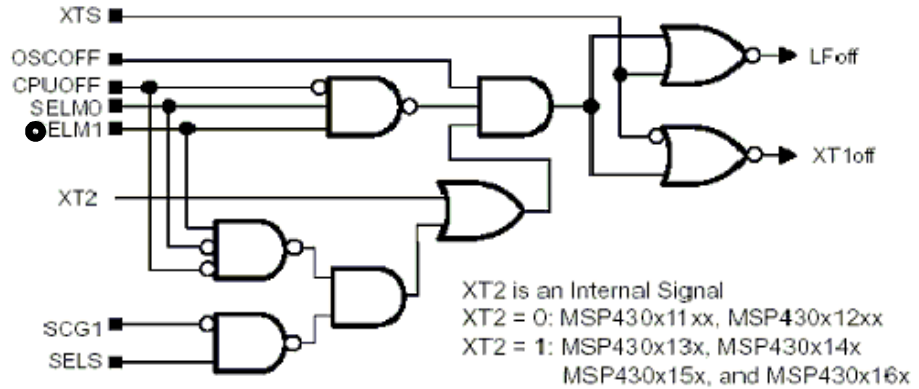
- | 高速晶体振荡器
- | 低速晶体振荡器
- | 数字控制振荡器DCO
- | 锁频环FLL以及锁频环增强版本FLL+

系统时钟必须满足以下不同要求:

- | 高频率，用于对系统硬件需求和外部事件快速反应。
- | 低频率，用于降低电流消耗。
- | 稳定的频率，以满足定时应用，如实时时钟RTC。
- | 低Q值振荡器，用于保证开始及停止操作最小时间延迟。

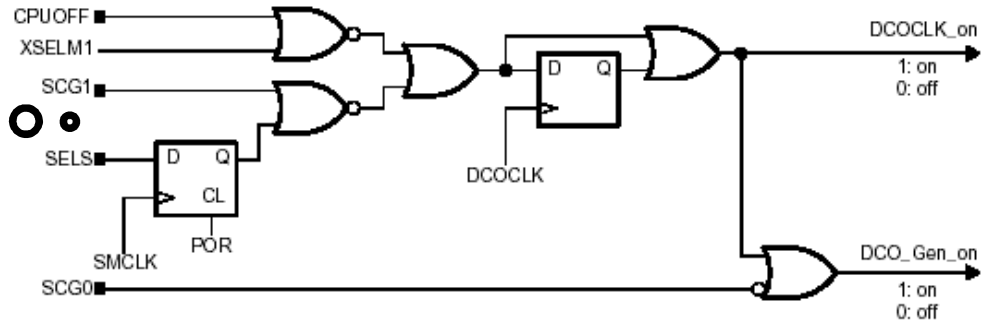
振荡器控制逻辑

**LFXT1
振荡器控制
逻辑**

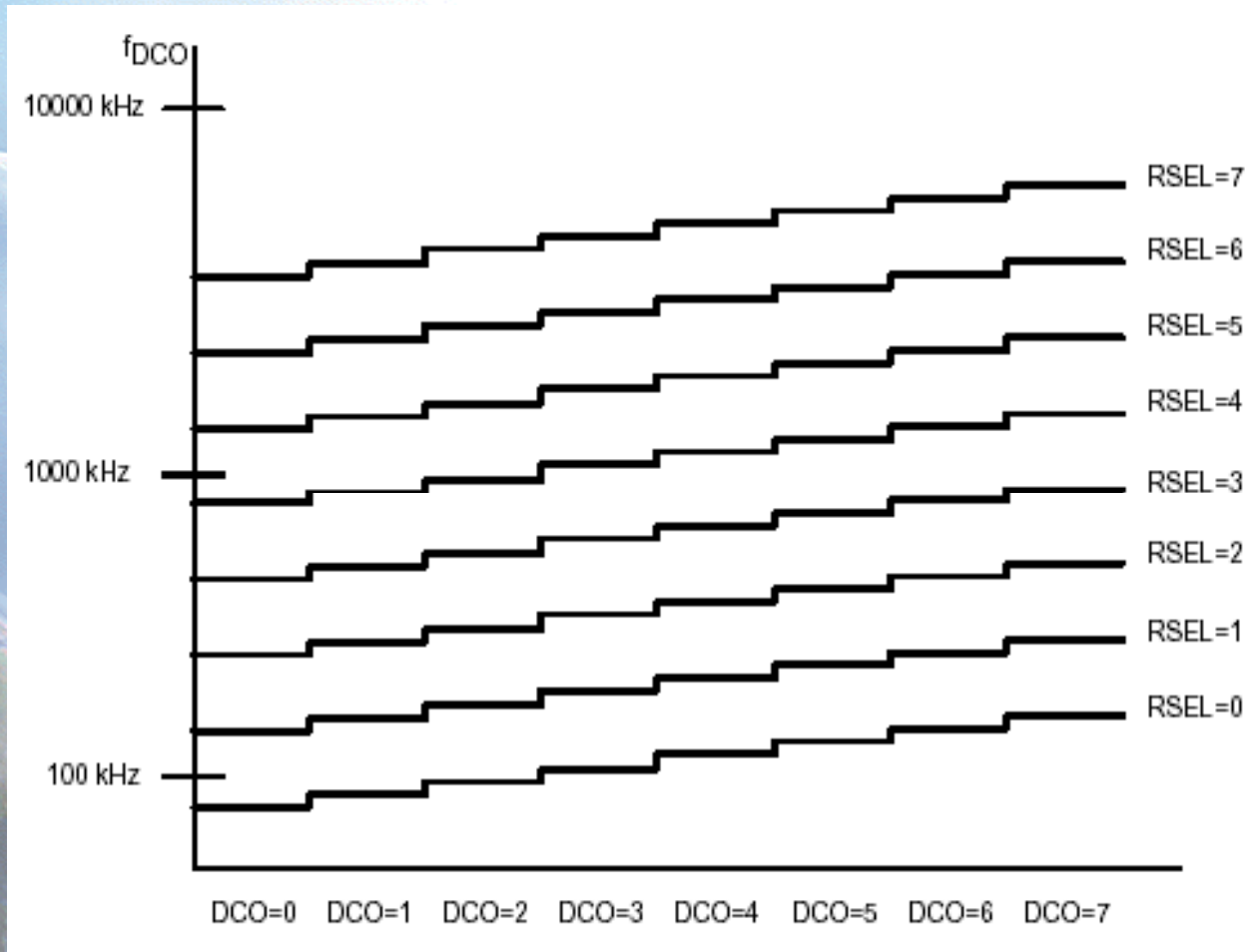


**XT2振荡器控
制逻辑**

**DCO振荡器
控制逻辑**



DCO频率的调节



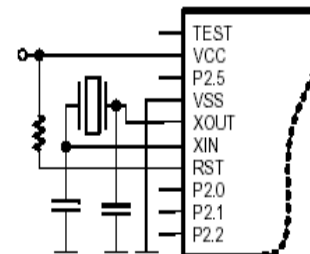
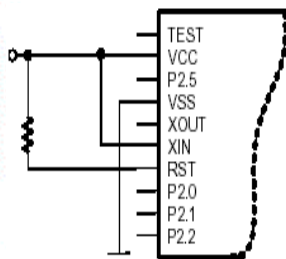
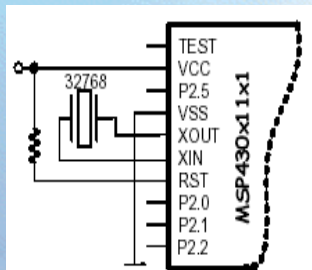
基础时钟模块工作方式和相关寄存器设置

例1设MCLK = XT2, SMCLK = DCOCLK, 将MCLK由P5.4输出。(MSP430X14X中引脚P5.4和MCLK复用)。

实现上述功能的程序如下:

```
#include <msp430x14x.h>
void main(void)
{
    unsigned int i;
    WDTCTL = WDTPW + WDTHOLD;    // 停止看门狗
    P5DIR |= 0x10;                // P5.4 输出
    P5SEL |= 0x10;                // P5.4 用作MCLK输出
    BCSCCTL1 &= ~XT2OFF;         // XT2有效
    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
    BCSCCTL2 |= SELM1;           // MCLK = XT2
    for (;;)
}
```

根据实际连接情况，确定ACLK、SMCLK和MCLK时钟源。



ACLK:
LFTX1 (32768)

MCLK:
DCOCLK或者LFTX1

SMCLK:
DCOCLK或者LFTX1

ACLK:
0

MCLK:
DCOCLK

SMCLK:
DCOCLK

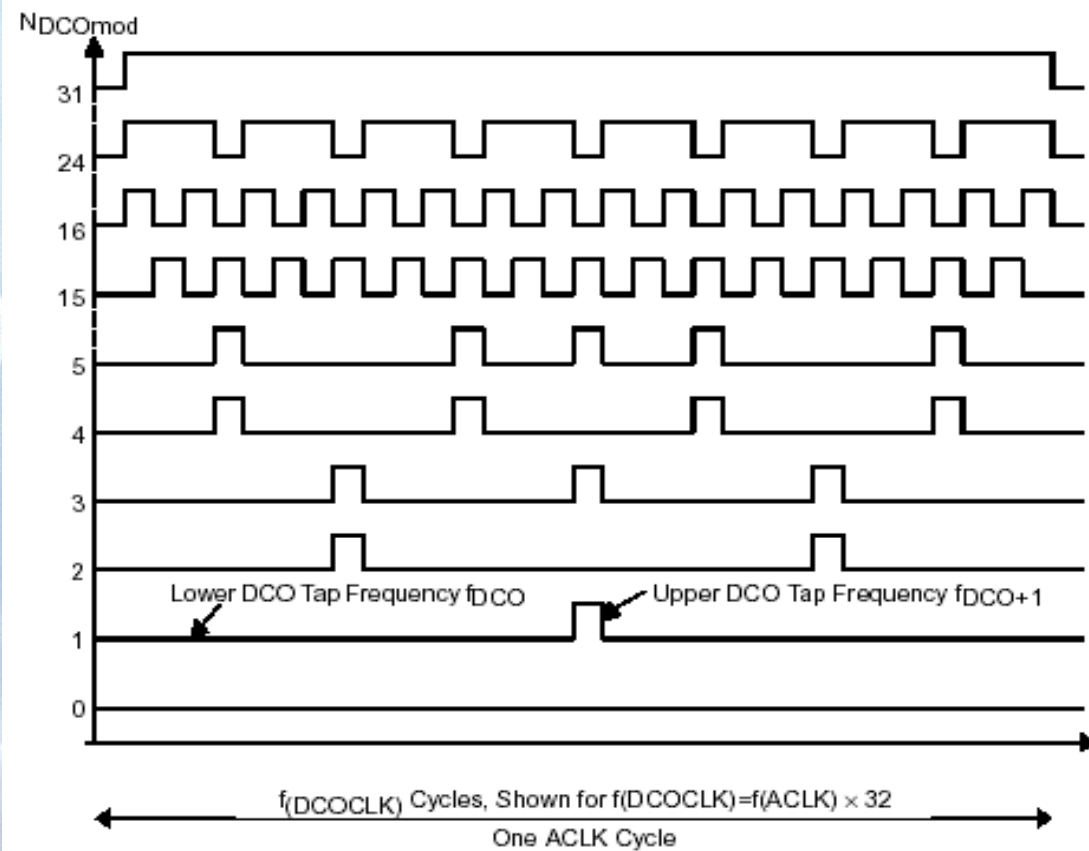
ACLK:
LFTX1 (高频模式)

MCLK:
DCOCLK或者LFTX1(高频模式)

SMCLK:
DCOCLK或者LFTX1(高频模式)

*ACLK只能来源于LFTX1。
MSP430X11X1内部只有LFTX1和DCO，没有XT2。
LFTX1只有工作于高频模式才需要外接电容。*

FLL+工作模式

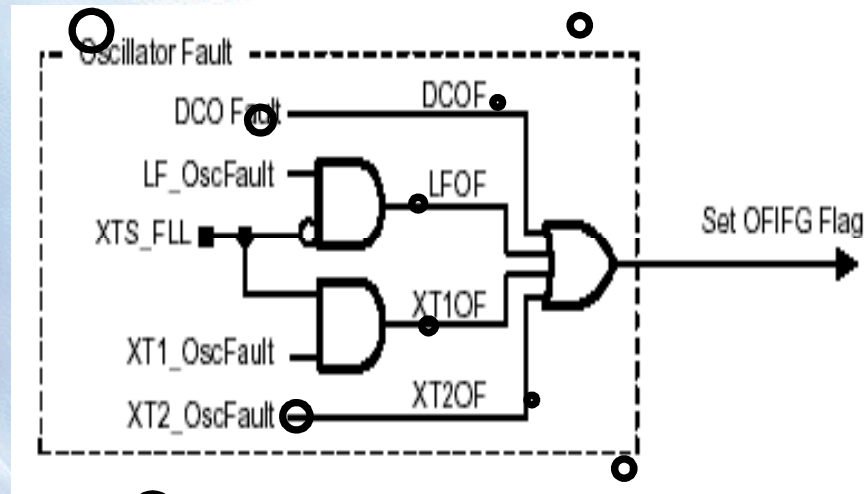


在每32个DCOCLK时钟周期中，调制器通过混合相邻两个DCOCLK周期可以克服周期累加的变化

FLL+失效控制

LFXT1 振荡器
在低频模式
(LF) 下
失效

DCO 振荡器
失效



LFXT1 振荡器在高
频模式(HF) 下失
效

XT2 振荡
器失效

调整FLL+

- | 保证FLL+锁定位（SCG0在状态寄存器中）并把它置位；关闭反馈环控制
- | 把新数值装入调整寄存器SCFQCTL（调整位M，乘数N）
- | 将DCO控制位置位，调整器高位置位：SCFI1=OFH，使得芯片以尽可能低的频率工作
- | 选择DCO+控制位为1或者0
- | 将控制寄存器SCFI0装入新的数值
- | 还原或设置FLL+控制位

FLL+模块应用举例

例1 设: $ACLK = LFXT1 = 32768\text{Hz}$, 令 $MCLK = SMCLK = DCOCLK = (n+1) \times ACLK$, 并将MCLK和ACLK分别通过P1.1和P1.5输出。

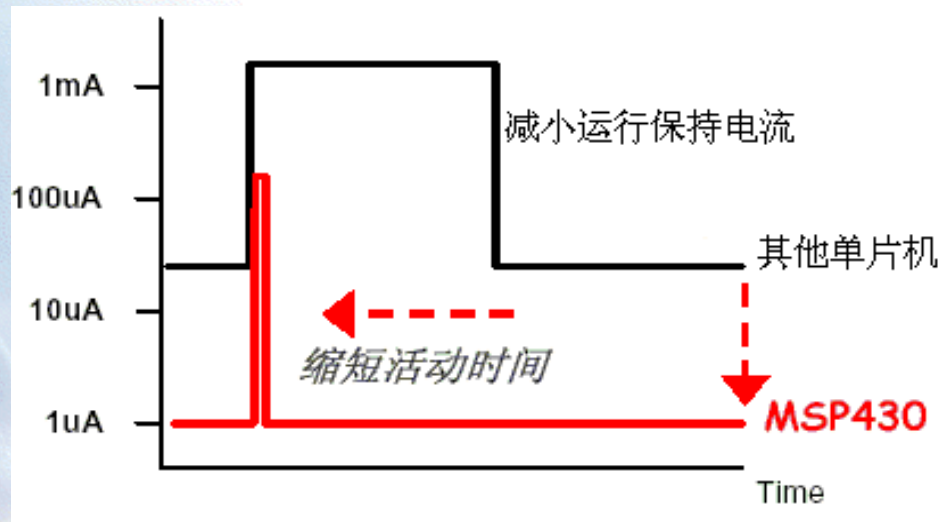
程序代码如下

```
#include "msp430x44x.h"
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // 停止看门狗
    SCFI0 |= FN_2;
    FLL_CTL0 = XCAP18PF;
    SCFQCTL = 74; // (74+1) × 32768 = 2.45Mhz
    P1DIR = 0x22; // P1.1 & P1.5 输出
    P1SEL = 0x22; // P1.1 & P1.5输出 MCLK & ACLK
    while(1);
}
```

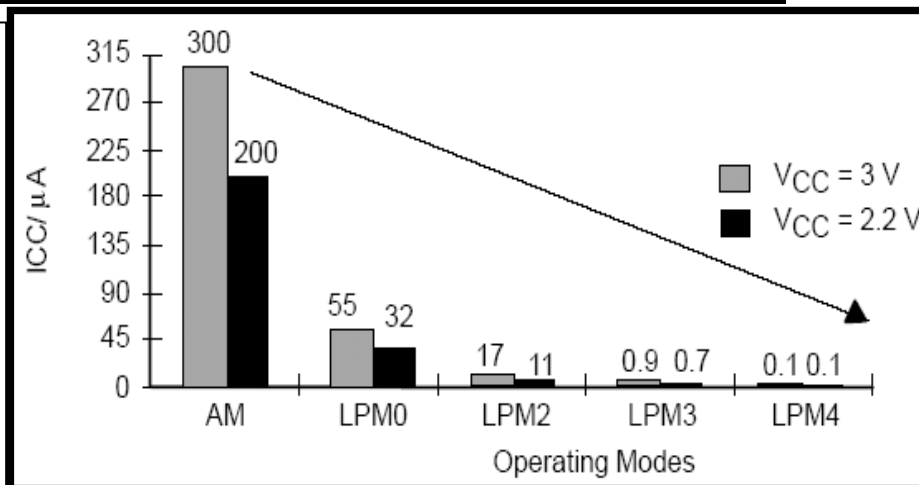
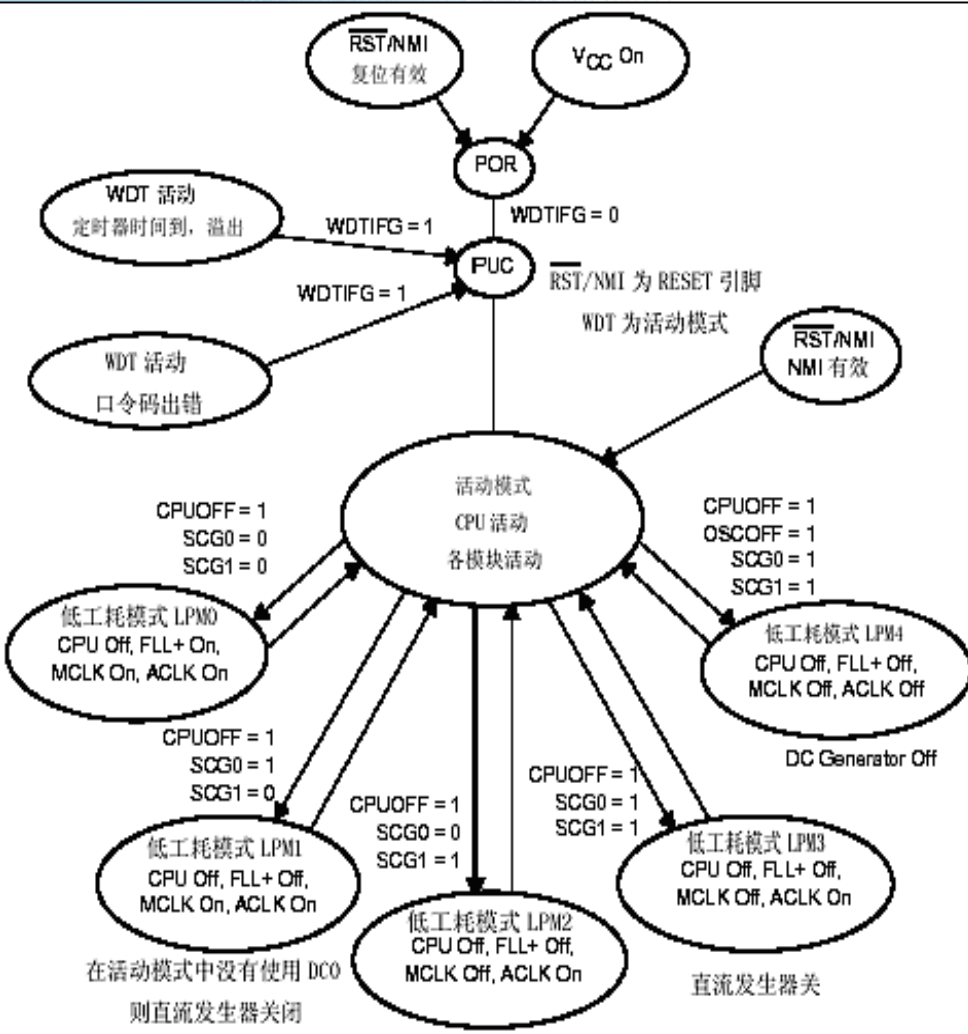
内部DCO = 2.45Mhz, P1.1--> MCLK = 2.45Mhz, P1.5--> ACLK = 32khz

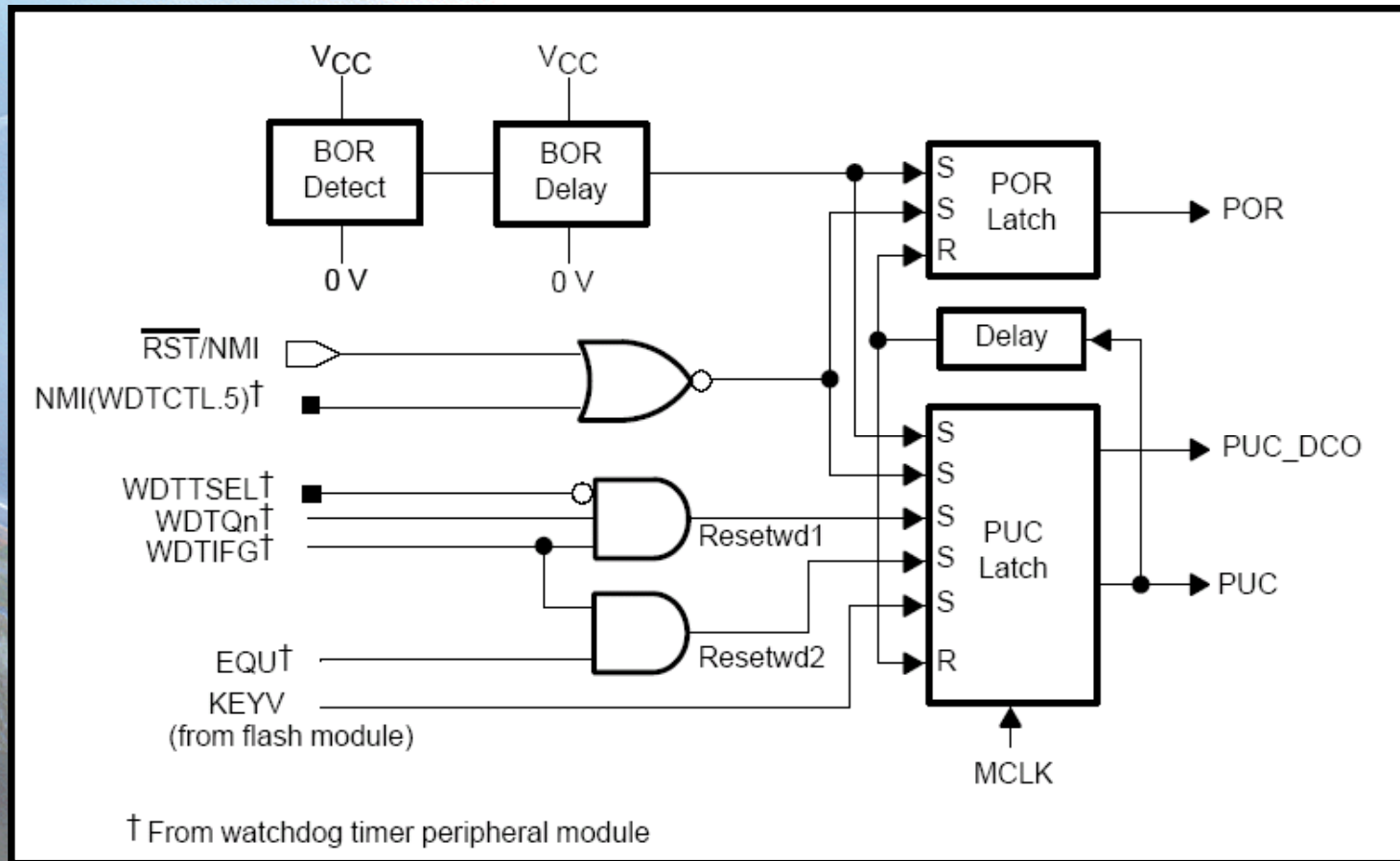
低功耗

- 使用内部时钟发生器（DCO）无需外接任何元件
- 选择外接晶体或陶瓷谐振器，可以获得最低频率和功耗
- 采用外部时钟信号源
- 瞬间响应特性



MSP430 工作模式状态





A POR is a device reset. A POR is only generated by the following two events:

- Powering up the device
- A low signal on the $\overline{\text{RST/NMI}}$ pin when configured in the reset mode

A PUC is always generated when a POR is generated, but a POR is not generated by a PUC. The following events trigger a PUC:

- A POR signal
- Watchdog timer expiration when in watchdog mode only
- Watchdog timer security key violation
- A Flash memory security key violation

Device Initial Conditions After System Reset

After a POR, the initial MSP430 conditions are:

- The $\overline{\text{RST}}/\text{NMI}$ pin is configured in the reset mode.
- I/O pins are switched to input mode as described in the *Digital I/O* chapter.
- Other peripheral modules and registers are initialized as described in their respective chapters in this manual.
- Status register (SR) is reset.
- The watchdog timer powers up active in watchdog mode.
- Program counter (PC) is loaded with address contained at reset vector location (0FFFEh). CPU execution begins at that address.

Port	Register	Short Form	Address	Register Type	Initial State
P1	Input	P1IN	020h	Read only	–
	Output	P1OUT	021h	Read/write	Unchanged
	Direction	P1DIR	022h	Read/write	Reset with PUC
	Interrupt Flag	P1IFG	023h	Read/write	Reset with PUC
	Interrupt Edge Select	P1IES	024h	Read/write	Unchanged
	Interrupt Enable	P1IE	025h	Read/write	Reset with PUC
	Port Select	P1SEL	026h	Read/write	Reset with PUC

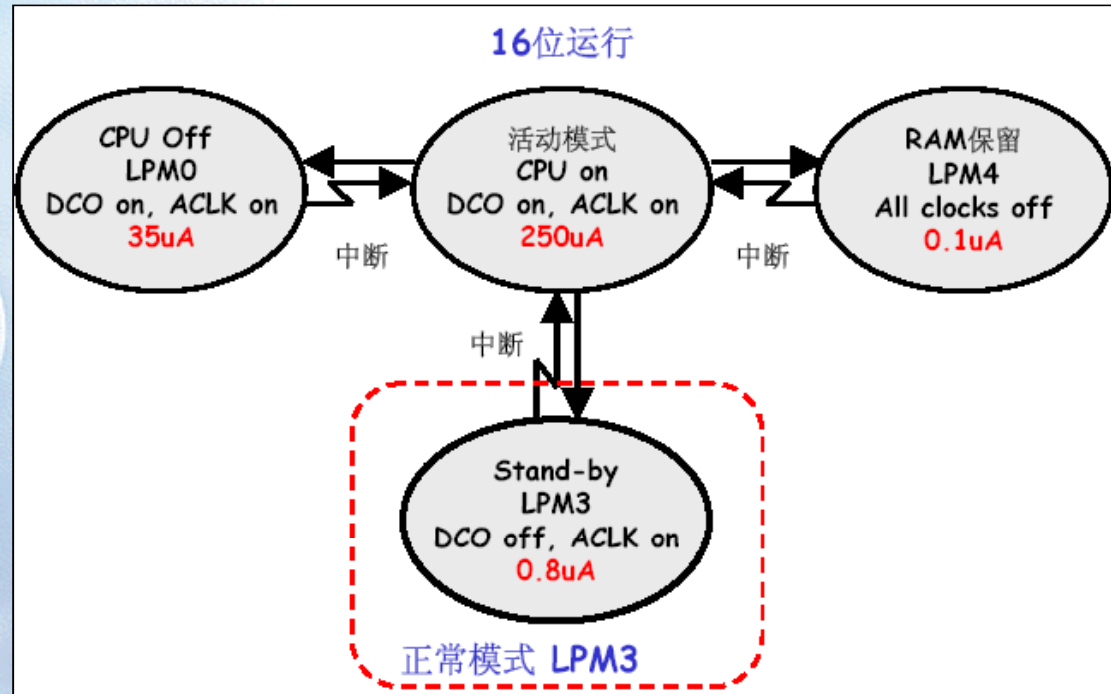
P3	Input	P3IN	018h	Read only	–
	Output	P3OUT	019h	Read/write	Unchanged
	Direction	P3DIR	01Ah	Read/write	Reset with PUC
	Port Select	P3SEL	01Bh	Read/write	Reset with PUC

Software Initialization

After a system reset, user software must initialize the MSP430 for the application requirements. The following must occur:

- Initialize the SP, typically to the top of RAM.
- Initialize the watchdog to the requirements of the application.
- Configure peripheral modules to the requirements of the application.

Additionally, the watchdog timer, oscillator fault, and flash memory flags can be evaluated to determine the source of the reset.



为了充分利用CPU低功耗性能，可以让CPU工作于**突发**状态。
在通常情况下，根据需要使用软件将CPU设定到某一种低功耗工作模式下，
在需要时使用**中断**将CPU从休眠状态中唤醒，完成工作之后又可以进入相应的休眠状态。

系统响应中断的过程

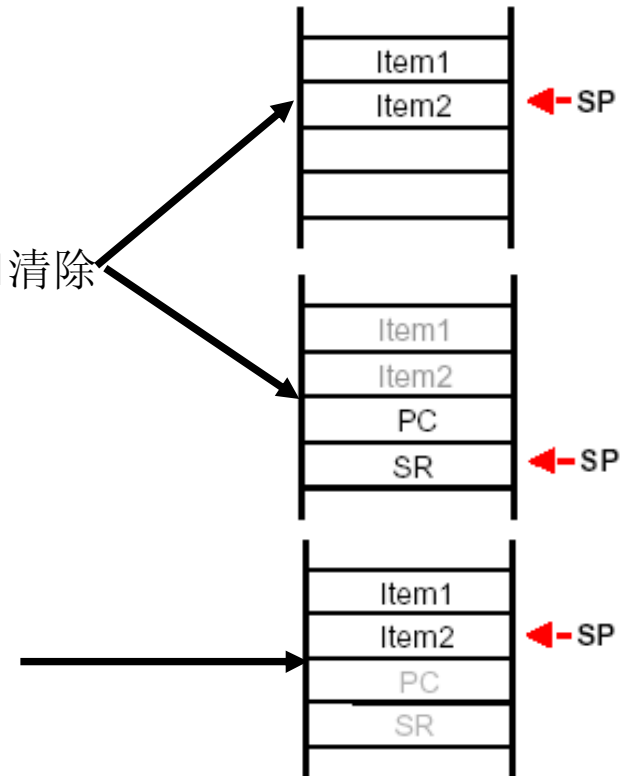
I 硬件自动中断服务

- PC入栈
- SR入栈
- 中断向量赋给PC
- GIT、CPUOFF、OSCOFF和SCG1清除
- IFG标志位清除（单源中断标志）

I 执行中断处理子程序

I 执行RETI指令（中断返回）

- SR出栈（恢复原来的标志）
- PC出栈



低功耗0转变为低功耗3

例：系统初始化完毕之后工作于低功耗模式0，中断事件触发到活动模式，中断处理结束后进入到低功耗模式3。

```

; 主程序
.....; 初始化操作开始

```

```

.....;
.....; 初始化完毕

```

```

BIS #GIE+CPUOFF, SR; 主程序中设置低功耗模式0

```

```

;.....; 程序在这里停止
;

```

```

; 中断子程序
.....; 中断处理开始

```

```

.....; 中断处理结束

```

```

BIS #GIE+CPUOFF+SCG1+SCG0, 0 (SP); 设置SR为低功耗模式3

```

```

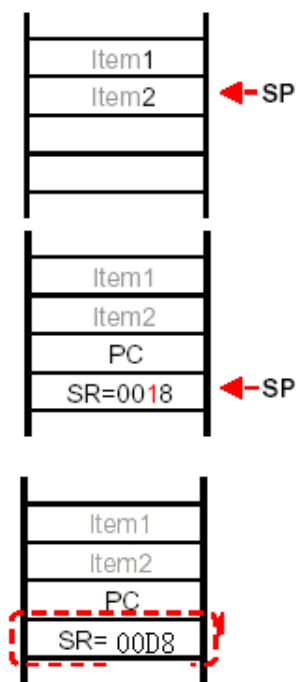
RETI; 中断返回

```

```

; 系统进入低功耗模式3。
.....

```



低功耗到活动状态转换

```
; 主程序
.....; 初始化操作开始
.....;
.....; 初始化完毕
BIS #GIE+CPUOFF, SR ; 主程序中设置低功耗模式0
L1 操作1
L2 操作2
.....
;
; 中断子程序
.....; 中断处理开始
.....
.....;中断处理结束
BIC #CPUOFF, 0 (SP) ; 设置SR为活动模式
RETI; 中断返回, 系统进入活动模式。
```

低功耗原则

| 一般的低功耗原则:

最大化LPM3的时间, 用32KHz晶振作为ACLK时钟, DCO用于CPU激活后的突发短暂运行
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

在冗长的软件计算中使用单周期的CPU寄存器

避免频繁的子程序和函数调用

尽可能直接用电池供电

| 设计外设时的常规原则:

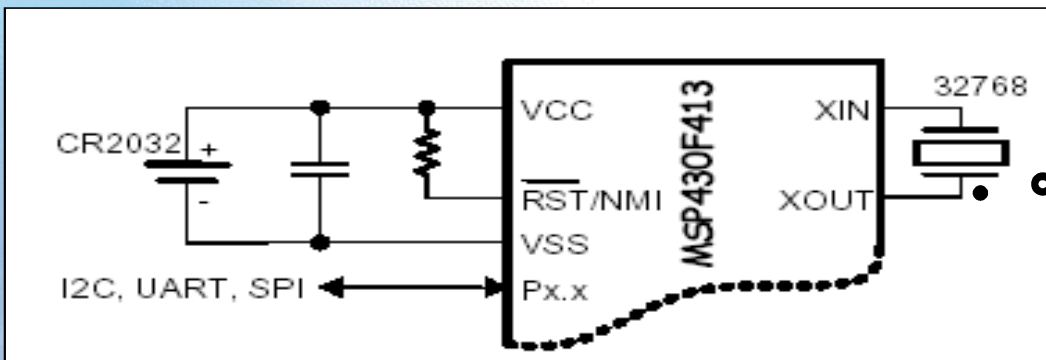
将不用的FETI输入端连接到VSS

JTAG端口TMS、TCK和TDI不要连接到VSS

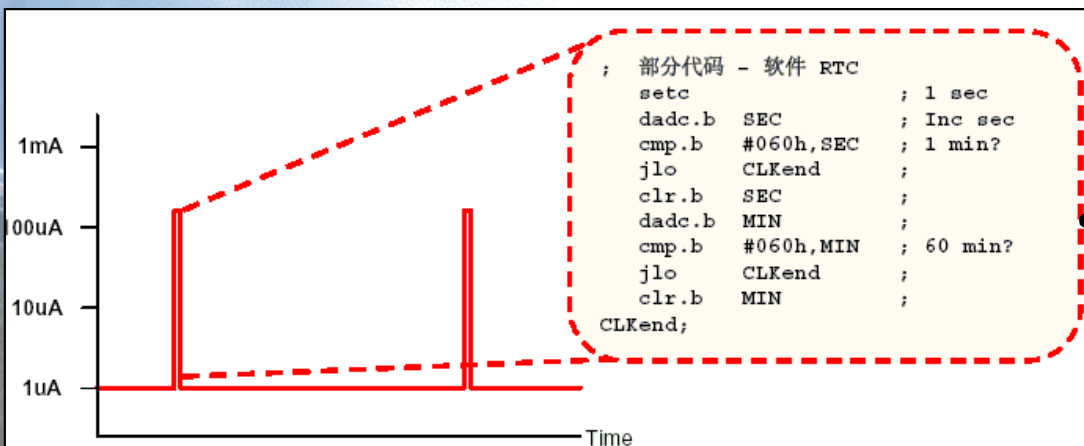
CMOS输入端不能有浮空节点, 将所有输入端接适当的电平

不论对于内核还是对于各外围模块, 选择尽可能低的运行频率, 如果不影响功能应设计自动关机

超低功耗嵌入式实时时钟



MSP430F413通过外接
32768Hz晶体



MSP430 CPU工作于
突发状态

$$\begin{aligned}
 \text{平均电流} &= \text{LPM3 电流} + \text{工作期间电流} \\
 &= 0.80\mu\text{A} + 250\mu\text{A} \times 100\mu\text{s} / 1000000\mu\text{s} \\
 &= 0.80\mu\text{A} + 0.030\mu\text{A} \\
 &= 0.83\mu\text{A}
 \end{aligned}$$

运行期间电流消耗
250uA

在1S时间段内程序运行
时间仅为100us

LPM3 电流为0.80uA

MSP430的端口

I 类型丰富

P1, P2,
P3, P4, P5, P6,
S和COM

I 功能丰富

I/O
中断能力
其他片内外设功能
驱动液晶

I 寄存器丰富

P1与P2各有7个寄存器
P3、P4、P5、P6有四个寄存器

表4-6 MSP430 的端口

器件	P1	P2	P3	P4	P5	P6	S	COM
MSP430X11X	√	√						
MSP430X12X	√	√	√					
MSP430X13/14/15/16X	√	√	√	√	√	√		
MSP430X4XX	√	√	√	√	√	√	√	√

不能使用不存在的资源，使用端口的时候要和具体的器件相关

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

```
#include <msp430x44x.h>
```

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

```
void main(void)
```

```
{
```

```
    unsigned long tmp;
```

```
    WDTCTL = WDTHOLD + WDTPW;
```

```
    P5OUT = 0x02;
```

```
    while(1)
```

```
    {
```

```
        P5DIR = 0x02;
```

```
        P5OUT ^= 0x02;
```

```
        for(tmp=0; tmp<67500; tmp++);
```

```
    }
```

```
}
```

```
//关闭看门狗
```

```
//设置P5.1输出为1
```

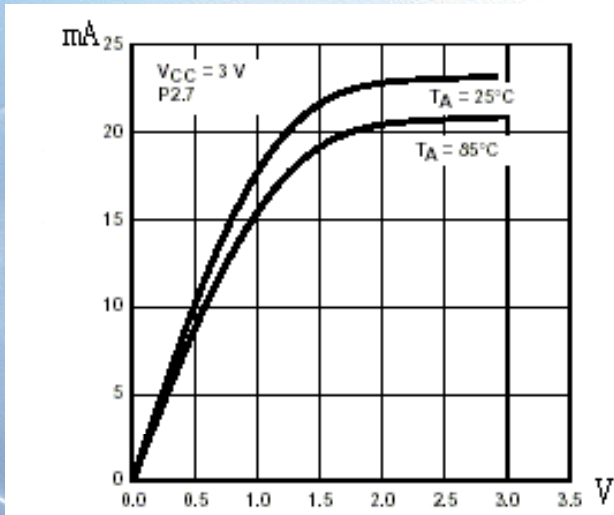
```
//循环
```

```
//使能P5.1为输出
```

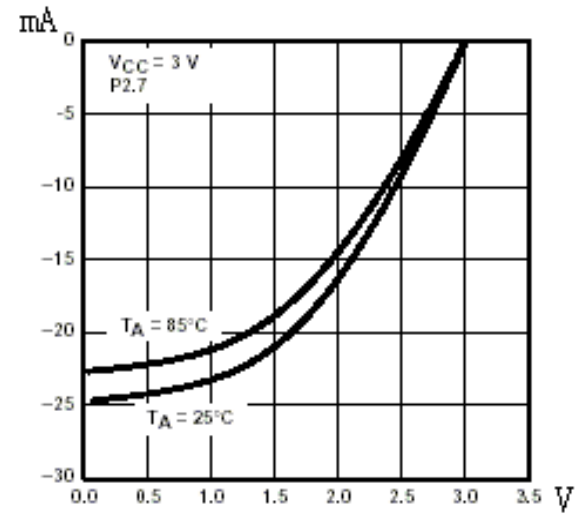
```
//对输出置反
```

```
//延时
```

端口数据输出特性



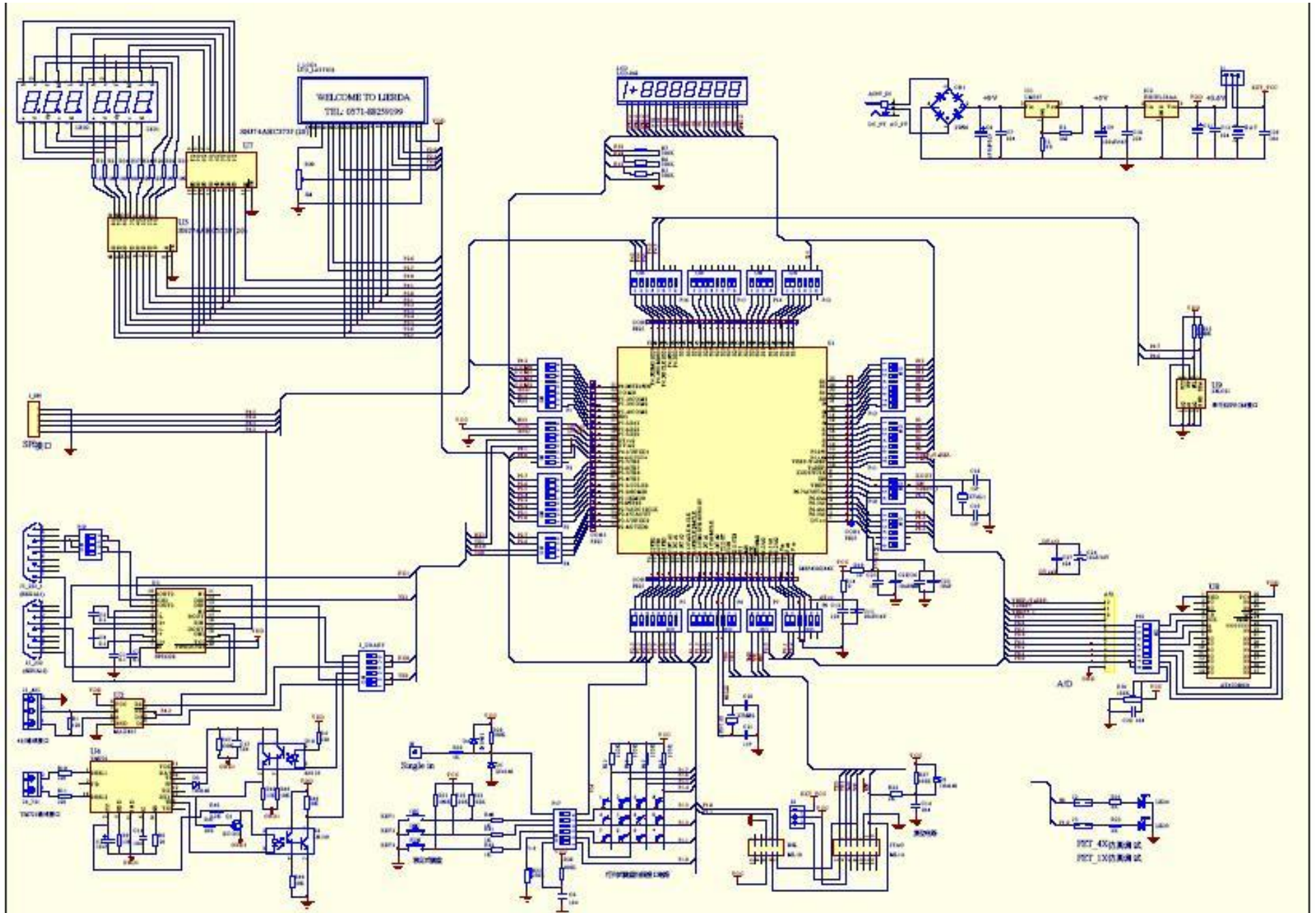
低电平输出特性

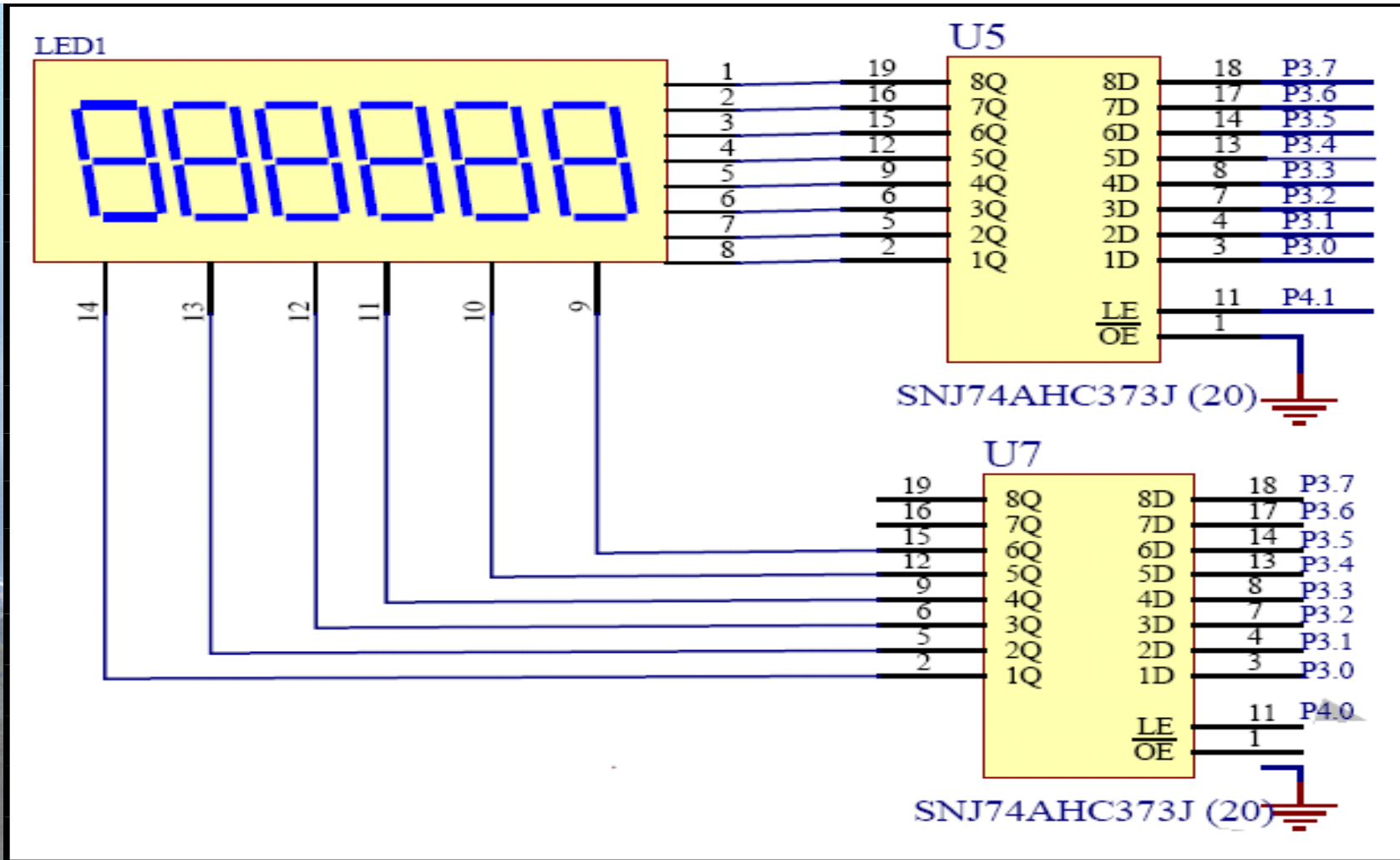


高电平输出特性









0—d7H, 1—14H, 2—CDH, 3—5DH, 4—1EH,
~~5—5BH, 6—dbH, 7—15H, 8—DFH, 9—5FH~~

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

```
MSP430-TEST44X Demo Using BasicTimer interruption to  
output 0~9 on each LED recuersively
```

```
// 说明：使用Basic Timer中断使各个LED逐个循环显示0~9
```

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

```
#include<msp430x44x.h>
```

```
int value=0,pos=0 ;
```

```
// 两变量分别表示LED上输出的值与位置
```

```
int nNumber[]={  
0xd7,0x14,0xcd,0x5d,0x1e,0x5b,0xdb,0x15,0xdf,0x5f};
```

```
// 定义字型码，与数组下标0到9一一映射
```

```
int nPosition[]={0x01,0x01<<1,0x01<<2,0x01<<3,0x01<<4,0x01<<5};
```

```
// 数组内容为地址码的反码
```

```
// 在pos个LED上输出num的宏
// P4OUT=0x01;           // P4.0输出
// P3OUT=nPosition[pos]; // P3OUT存储所需地址码的反码, pos从0到5
// P3OUT^=0xff;         // P3OUT取反, 输出地址码
// P4OUT=0x02;           // P4.1输出
// P3OUT=nNumber[num];  // 输出字型码

#define SHOW(pos,num)
{P4OUT=0x01;P3OUT=nPosition[pos];P3OUT^=0xff;
P4OUT=0x02;P3OUT=nNumber[num];}
```

```
void main(){  
    WDTCTL=WDTPW+WDTHOLD;           // 停止看门狗  
    IE2|=BTIE;                       // 使能BT中断  
    FLL_CTL0|=XCAP14PF;              //  
    BTCTL=BTDIV+BTIP1+BTIP0;        // 125ms中断间隔  
    P3DIR=0x7f;                      // P3.0到P3.7输出  
    P4DIR=0x03;                      // P4.0与P4.1输出  
    P3OUT=0 ;P4OUT=0;                // 清空P3OUT,P4OUT  
    _EINT();                          // 使能系统总中断  
    for(;;){  
        _BIS_SR(LPM3_bits);          // 进入LPM3  
        _NOP();                      //  
    }  
}
```

| //Basic Timer中断服务程序

| #pragma vector=BASICTIMER_VECTOR

| __interrupt void basic_timer(void)

| {

| SHOW(pos,value); // 在pos个LED上输出value

| value=(value+1)%10; // value从0到9循环

| if(value==0)

| pos=(pos+1)%6; // 如果已输出0到9, 在下一个LED上输出

| }

```
#include <msp430x44x.h>
```

```
void main (void ){
```

```
    unsigned i;
```

```
    WDTCTL=WDTPW+WDTHOLD;           //停止看门狗
```

```
    P5DIR |= BIT1;                   //P5.1为输出
```

```
    P5OUT &= ~ BIT1;                 //P5.1为低
```

```
    for (i=0;i<60000;i++)            //延时
```

```
    P5OUT |= BIT1;                   //P5.1为高
```

```
    for (i=0;i<60000;i++)            //延时
```

```
    WDTCTL=WDTPW;                     //启动看门狗
```

```
    while (1);
```

```
}
```

```
#include <msp430x44x.h>

void main (void ){

    unsigned i;
    //0<=i<=65535

    WDTCTL=WDTPW+WDTHOLD;           //停止看门狗
    P5DIR |= BIT1;                  //P5.1为输出
    P5OUT &= ~ BIT1;                //P5.1为低
    for (i=0;i<60000;i++)           //延时
    P5OUT |= BIT1;                  //P5.1为高
    for (i=0;i<60000;i++)           //延时
    WDTCTL=WDTPW;                   //启动看门狗
    while (1);{

    WDTCTL=WDTPW+WDTCNTCL;          //计数器清零}}

MSP430-82
```

```
#include <msp430x11x1.h>


---


void main(void){
WDTCTL = WDT_ADLY_250;           // 设置看门狗定时时间为0.25ms
IE1 |= WDTIE;                   // WDT使能
P1DIR |= 0x01;                  //设置 P1.0 为输出
_EINT();                         //中断允许
for (;;) {
    _BIS_SR(LPM3_bits);          // 进入LPM3
    _NOP();                      // 验证，可用C-SPY观察 }}
// Watchdog Timer interrupt service routine
interrupt[WDT_VECTOR] void watchdog_timer(void)
{
    P5OUT ^= 0x01;              // P5.1输出取反
}


---


```

定时器

- | 看门狗定时器
- | 基本定时器
- | 定时器A
- | 定时器B

表4-9 WDT 的定时时间

SSEL	IS1	IS0	定时时间/ms	
0	1	1	0.04 <u>60.064</u>	$t_{smclk} \times 2^6$
0	1	0	0.50 <u>51</u>	$t_{smclk} \times 2^9$
1	1	1	1.91 <u>95</u>	$t_{aclk} \times 2^6$
0	0	1	<u>88.19</u>	$t_{smclk} \times 2^{13}$
1	1	0	16.01 <u>5.63</u>	$t_{aclk} \times 2^9$
0	0	0	3232.77	$t_{smclk} \times 2^{15}$ (PUC 复位后的值)
1	0	1	250	$t_{aclk} \times 2^{13}$
1	0	0	1000	$t_{aclk} \times 2^{15}$

看门狗定时器应用举例

中断允许
WDTIE 位于
IE1.0, 初始
状态为复位

```
#include <msp430x14x.h>
void main(void)
{
    WDTCTL = WDT_MDLY_32; // 定时周期为32ms
    IE1 |= WDTIE; // 使能WDT中断
    P1DIR |= 0x01; // P1.0输出
    _EINT(); // 系统中断允许
    for (;;)
    {
        _BIS_SR(CPUOFF); // 进入 LPM0
        _NOP();
    }
    // 看门狗中断服务子程序
    #pragma vector= WDT_VECTOR
    __interrupt void watchdog_timer (void)
    {
        P1OUT ^= 0x01; // P1.0取反
    }
}
```

两种中断,
两个级别,
两种处理

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

```
*
```

```
* 文件名称:
```

```
* main_c.c
```

```
* 文件说明:
```

```
* 程序用于对看门狗的两种工作模式进行测试
```

```
* 当setWDT (0) 时, 程序测试看门狗的看门狗模式
```

```
* 当setWDT (1) 时, 程序测试看门狗的定时器模式
```

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

```
#include <msp430x44x.h>
```

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

```
* 设置看门狗的两种模式
```

```
* 在系统加电后, 默认的子系统频率是1M, 设置WDT延迟为250ms
```

```
* 那么系统执行0x1ffff 条指令后系统如果没有重新设置WDT,
```

```
* WDT将导致系统复位
```

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



```

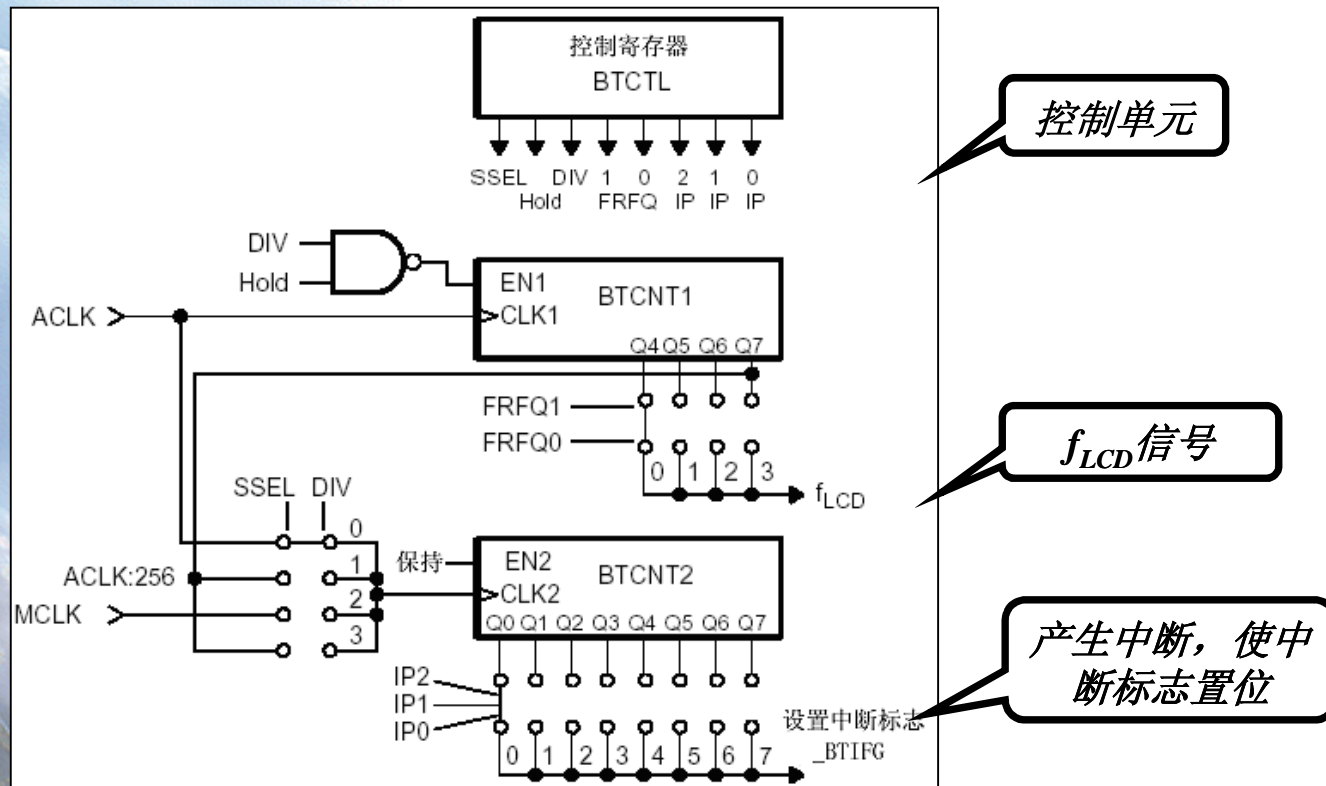
void setWDT(int mode)
{
    unsigned int tmp;
    if(mode==1)
    {
        // 定时器模式
        WDTCTL = WDT_ADLY_250; // 250ms 中断间隔
        IE1 |= WDTIE; // 使能 WDT 中断
        P5DIR |= 0x02; // P5.1 output mode
        _EINT(); // 开中断
    } else if(mode==0)
    {
        //看门狗模式
        WDTCTL = WDT_ARST_250; //设置看门狗时间间隔为250ms
        //在系统加电后，默认子系统时钟频率是1M
        for(tmp=0x7fff;tmp>0;tmp--); //延迟
        P5DIR |= 0x02; //P5.1 output mode
        P5OUT |= 0x02; //light LED4
        for(tmp=0x7fff;tmp>0;tmp--); //延迟， 执行一次tmp--至少要用2个指令
    }
}

```

```
/*  
* 函数说明.  
*  
*   WDT中断函数  
*  
*****/  
  
#pragma vector = WDT_VECTOR  
__interrupt void WDT_Interrupt(void)  
{  
    P5OUT ^= 0x02;  
}  
/*  
* main()函数  
*****/  
  
void main(void)  
{  
  
    setWDT(0);           //设置程序运行在看门狗模式  
}
```

基本定时器

- 支持软件和各种外围模块工作在低频率、低功耗条件下



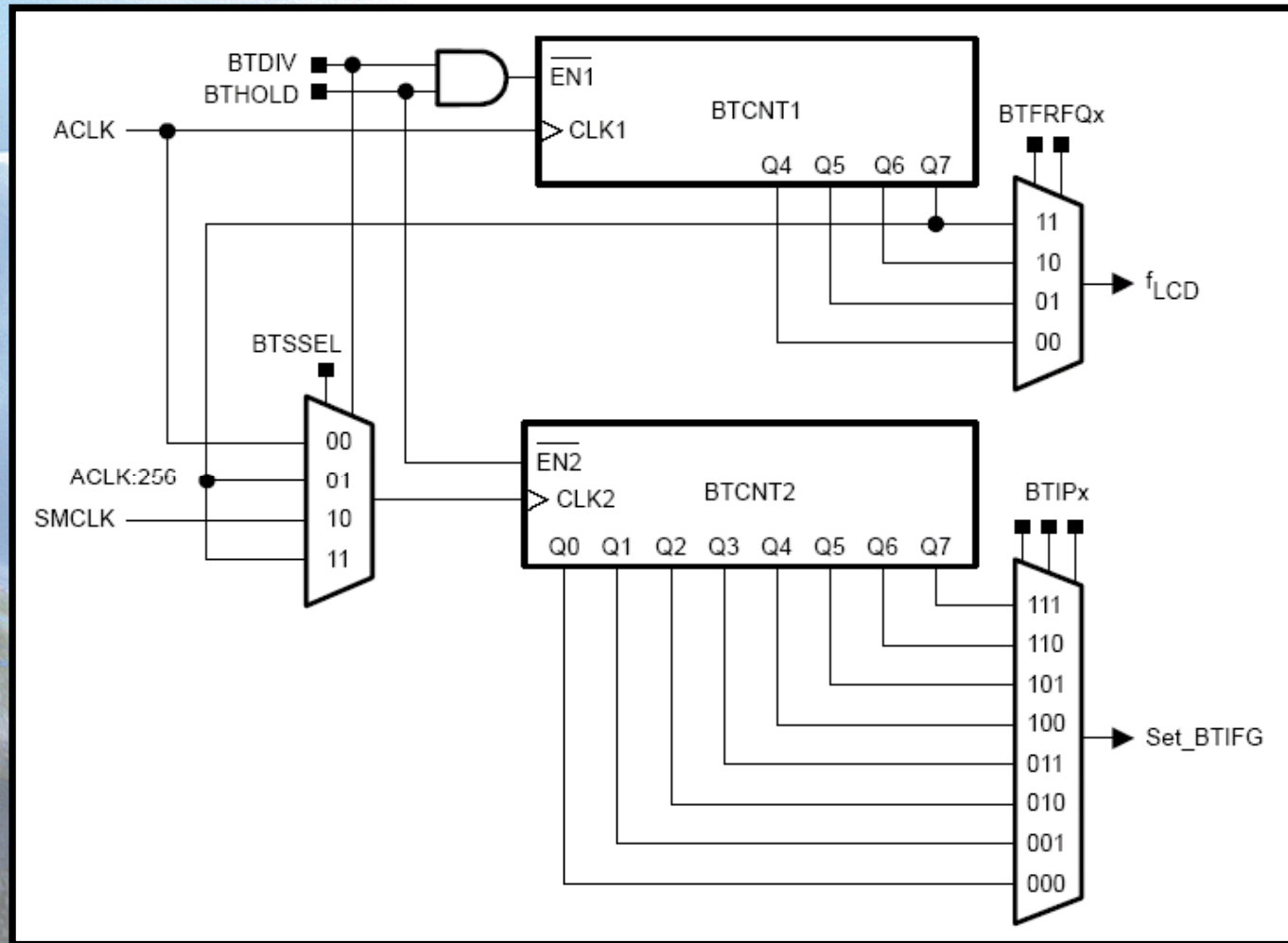


表4-11 f_{LCD} 信号频率

FRFQ1	FRFQ0	f_{LCD}
0	0	$f_{ACLK}/CLK2 \rightarrow 32$
0	1	$f_{ACLK}/CLK2 \rightarrow 64$
1	0	$f_{ACLK}/CLK2 \rightarrow 128$
1	1	$f_{ACLK}/CLK2 \rightarrow 256$

表4-12 BTCNT2 的定时中断频率

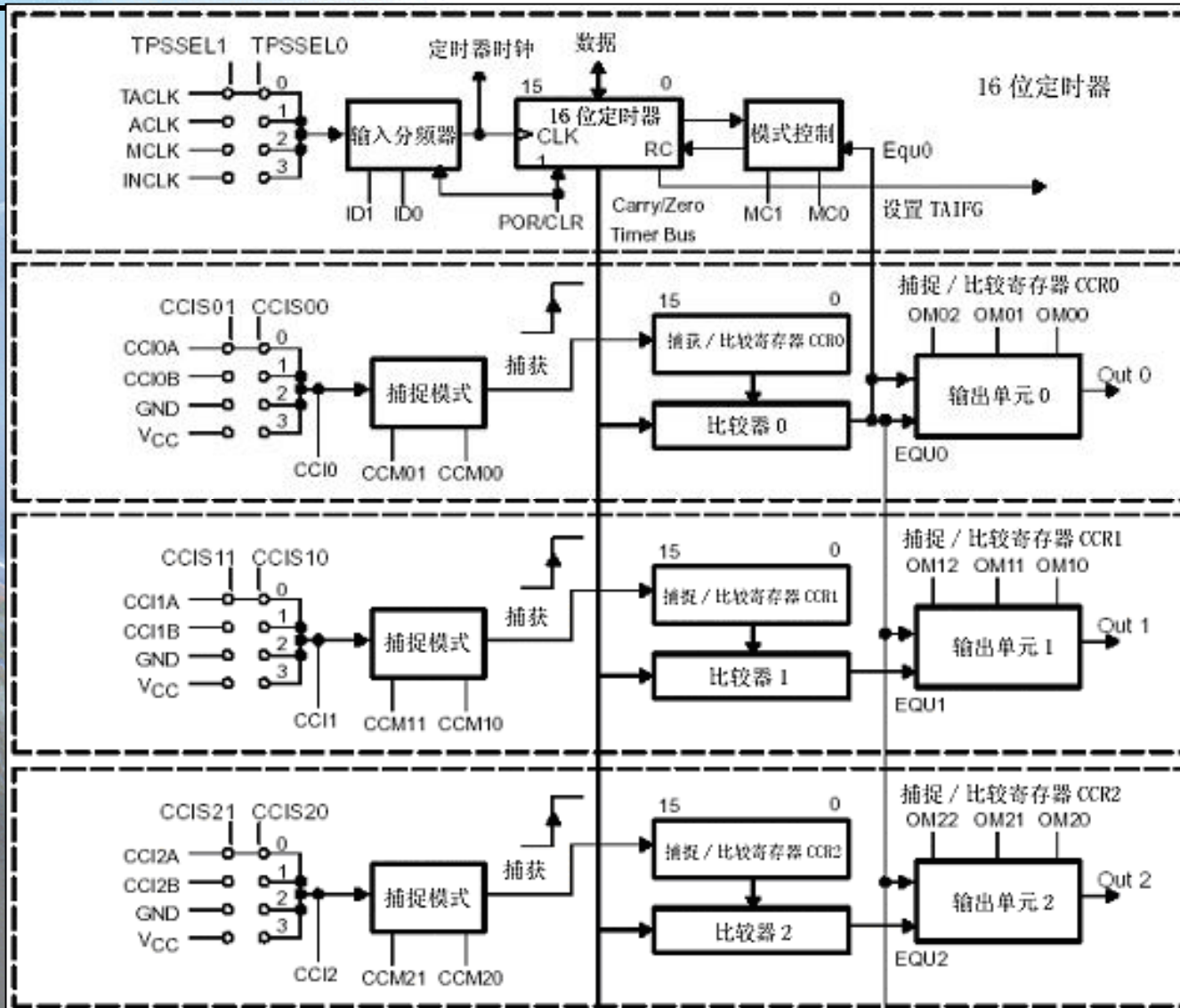
IP2	IP1	IP0	中断频率
0	0	0	$f_{CLK2}/2$
0	0	1	$f_{CLK2}/4$
0	1	0	$f_{CLK2}/8$
0	1	1	$f_{CLK2}/16$
1	0	0	$f_{CLK2}/32$
1	0	1	$f_{CLK2}/64$
1	1	0	$f_{CLK2}/128$
1	1	1	$f_{CLK2}/256$

```
#include <msp430x44x.h>
void main(void)
{
WDTCTL = WDTPW + WDTCTL;
IE2 |= BTIE;           //使能BT 中断
FLL_CTL0 |= XCAP14PF;
BTCTL = BTDIV + BTIP1 + BTIP0;    // 125ms中断间隔
P5DIR |= 0x02;        // P5.1 输出
_EINT();              // 使能系统总中断
for (;;)
{
    _BIS_SR(LPM3_bits);    // 进入 LPM3
    _NOP();
}
}
// Basic Timer 中断服务子程序
#pragma vector= BASICTIMER_VECTOR
__interrupt void basic_timer (void)
{
    P5OUT ^= 0x02;        //取反P5.1
}
```

定时器A特性

- | 输入时钟可以有多种选择，可是慢时钟，快时钟以及外部时钟
- | 虽然没有自动重载时间常数功能，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。
- | 可实现串行通讯
- | 完善的中断服务功能
- | 4种计数功能选择
- | 8种输出方式选择
- | 支持多时序控制
- | DMA使能

定时器A结构



定时器A功能模块

- 丨 **计数器部分**：输入的时钟源具有4种选择，所选定的时钟源又可以1、2、4或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
- 丨 **捕获/比较器**：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是为了提高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，输入和输出都决定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**。
- 丨 **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**

表4-14 Timer_A 的寄存器

寄存器	缩写	读写类型	地址	初始状态
Timer_A 控制寄存器	TACTL	读写	160H	POR 复位
Timer_A 计数器	TAR	读写	170H	POR 复位
捕获/比较控制寄存器 0	CCTL0	读写	162H	POR 复位
捕获/比较寄存器 0	CCRO	读写	172H	POR 复位
捕获/比较控制寄存器 1	CCTL1	读写	164H	POR 复位
捕获/比较寄存器 1	CCR1	读写	174H	POR 复位
捕获/比较控制寄存器 2	CCTL2	读写	168H	POR 复位
捕获/比较寄存器 2	CCR2	读写	178H	POR 复位
中断向量寄存器	TAIV	读写	12EH	POR 复位

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

TAR 16位计数器 该单元就是执行计数的单元，是计数器的主体

Timer_A有多个捕获/比较模块，每个模块都有自己的控制字CCTLx

CCR_x 捕获/比较寄存器 在捕获/比较模块中，可读可写。

TAIV保存中断向量

15-10	9	8	7	6	5	4	3	2	1	0
未用	SSEL1	SSEL0	ID1	ID0	MC1	MC0	未用	CLR	TAIE	TAIFG

表4-15 Timer_A 时钟源

SSEL1	SSEL0	输入时钟源	说明
0	0	TACLK	用特定的外部引脚信号
0	1	ACLK	辅助时钟
1	0	MCLK	系统时钟
1	1	INCLK	见器件说明

ID1, ID0 输入分频选择

00 不分频;
01 2分频;
10 4分频;
11 8分频。

MC1, MC0 计数模式控制位。
00 停止模式;
01 增计数模式;
10 连续计数模式;
11 增/减计数模式。

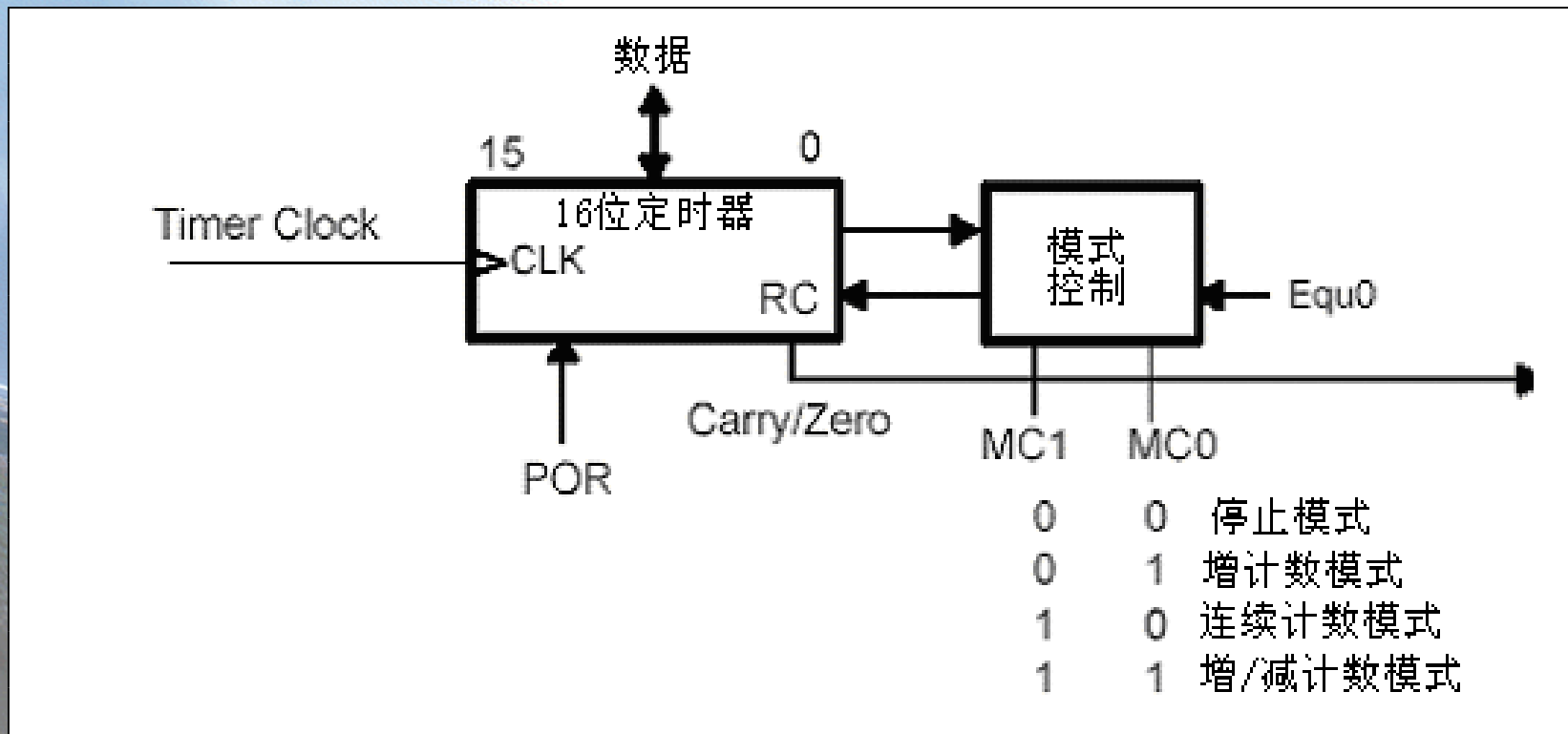
CLR 定时器清除位

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：当定时器由 CCR0 计数到 0 时，TAIFG 置位。
连续计数模式：当定时器由 0FFFFH 计数到 0 时，TAIFG 置位。
增/减计数模式：当定时器由 1 减计数到 0 时，TAIFG 置位。

定时器计数模式

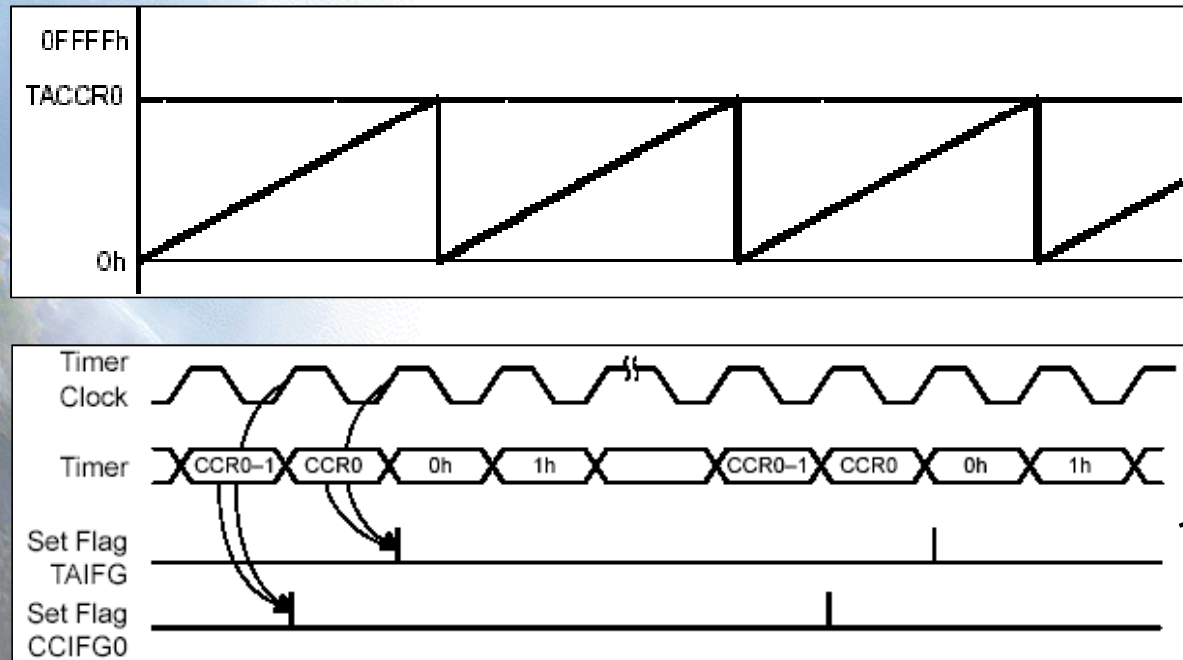


停止模式

- 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，计数器将从暂停时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_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控制位来清除定时器的方向记忆特性。

增计数模式

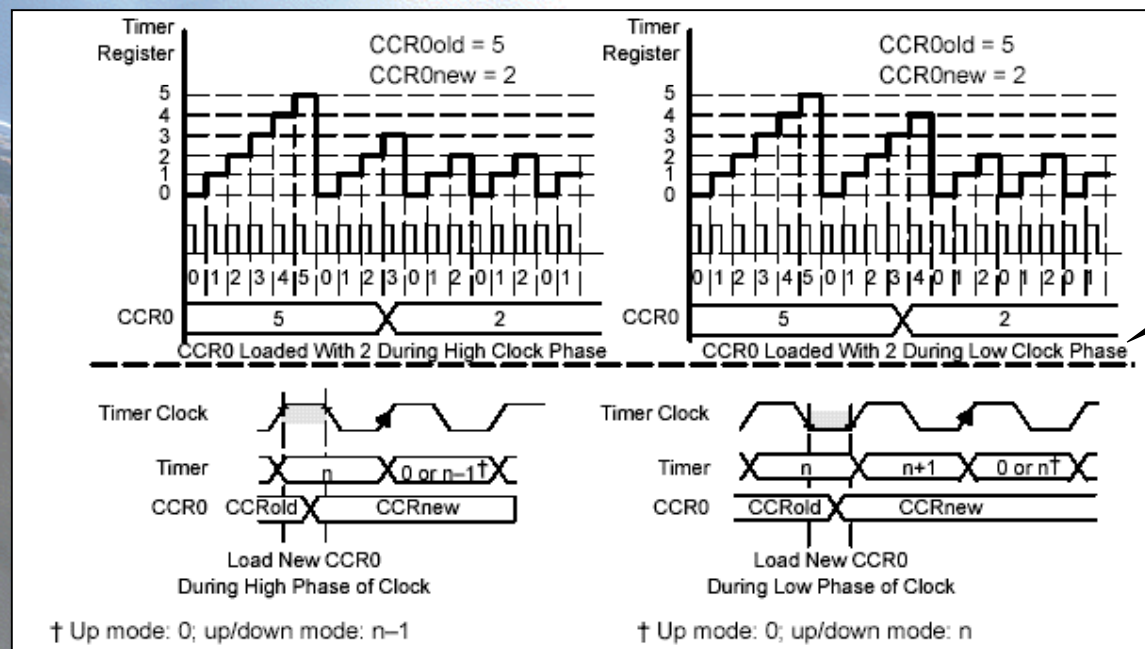
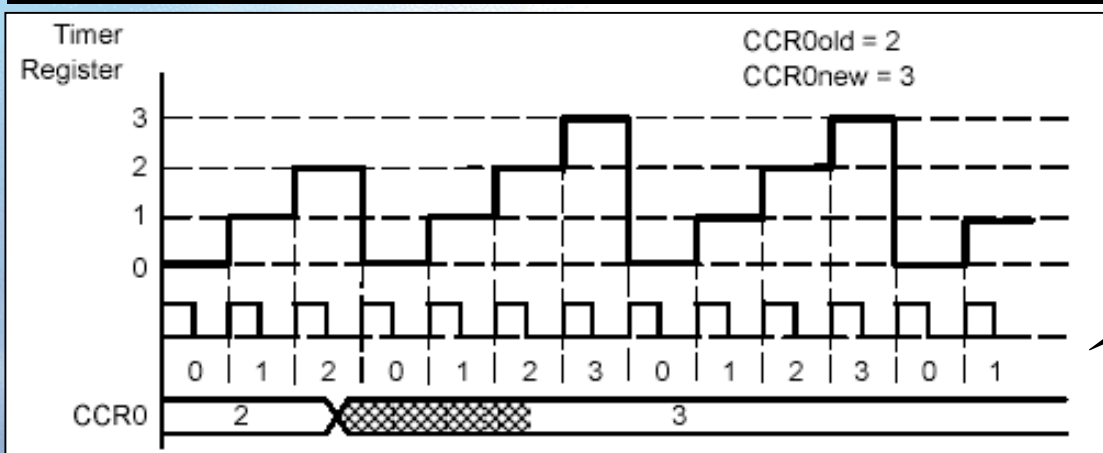
- 捕获/比较寄存器**CCR0**用作Timer_A增计数模式的周期寄存器，因为CCR0为16位寄存器，所以该模式适用于定时周期小于65 536的连续计数情况。计数器**TAR**可以增计数到CCR0的值，当计数值与CCR0的值相等(或定时器值大于CCR0的值)时，定时器复位并从0开始重新计数。



增计数模式的
计数过程

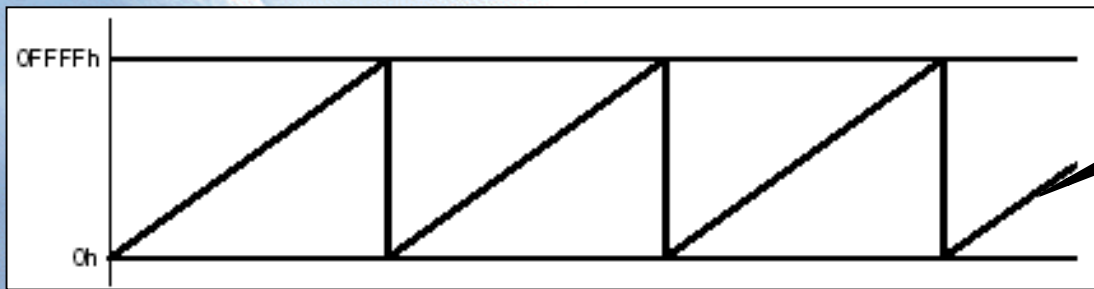
增计数模式的
中断标志位设置

改变CCR0值重置计数周期一增计数方式

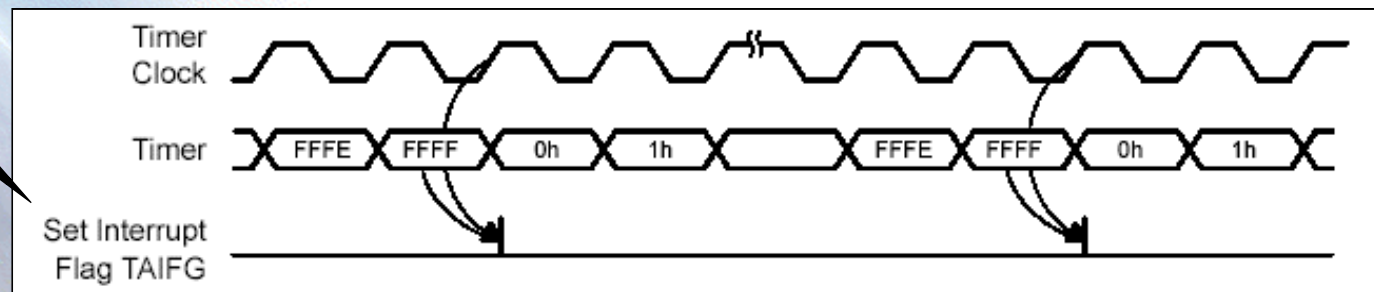


连续计数模式

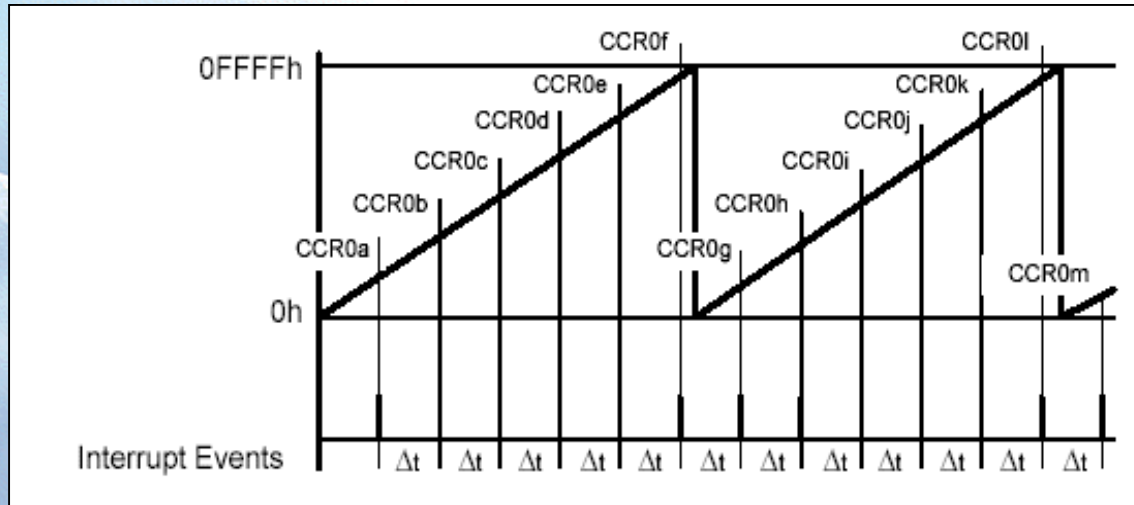
- 在需要65 536个时钟周期的定时应用场合常用连续计数模式。定时器从当前值计数到0FFFFH后，又从0开始重新计数



标志位
设置



连续计数模式的典型应用

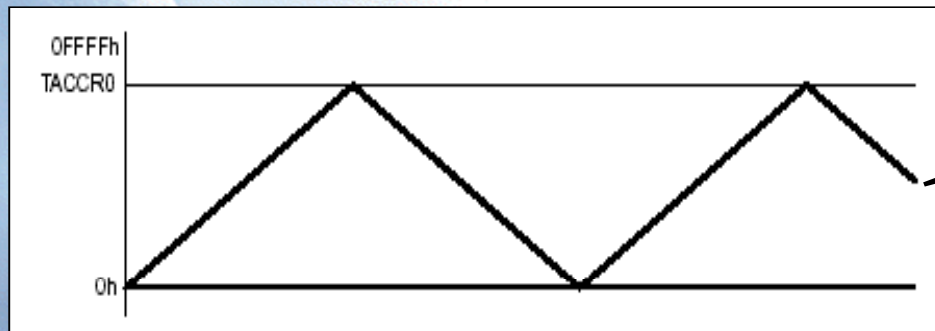


产生多个独立的时序信号：利用捕获比较寄存器捕获各种其他外部事件发生的定时器数据

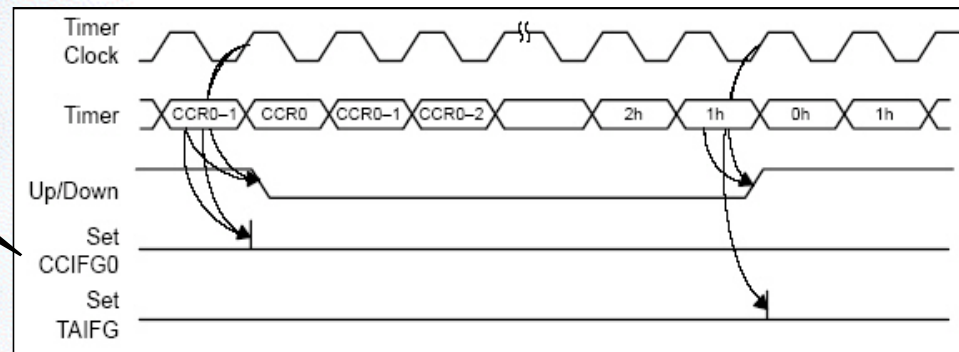
产生多个定时信号：通过中断处理程序在相应的比较寄存器 CCR_x 上加上一个时间差来实现。这个时间差是当前时刻（既相应的 CCR_x 中的值）到下一次中断发生时刻所经历的时间

增/减计数模式

- 需要对称波形的情况经常可以使用增/减计数模式，该模式下，定时器先增计数到CCR0的值，然后反向减计数到0。计数周期仍由CCR0定义，它是CCR0计数器数值的2倍。



标志位
设置



设：ACLK = TACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO = 32×ACLK = 1.048576MHz, 要求从P5.1输出一个方波。

示例程序代码如下：

```
#include <msp430x44x.h>
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;
    TACTL = TASSEL0 + TACLK; // ACLK, 清除 TAR
    CCTLO = CCIE;
    CCR0 = 1000; //方波频率：32768/1000/2=16.384
    P5DIR |= 0x02;
    TACTL |= MC0; // Timer_a增计数模式
    _EINT();
    for (;;)
    {
        _BIS_SR(LPM3_bits); // 进入 LPM3
        _NOP();
    }
}
// Timer A0中断服务程序
#pragma vector= TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
    P5OUT ^= 0x02; // 取反P5.1
}
```

CCTLx 捕获/比较控制寄存器

15 14 _↕	13 12 _↕	11 _↕	10 _↕	9 _↕	8 _↕	7 6 5 _↕	4 _↕	3 _↕	2 _↕	1 _↕	0 _↕
CAPTMOD 1~0 _↕	CCIS 1~0 _↕	SCS _↕	SCCI _↕	_↕	CAP _↕	OUTMODx _↕	CCIEx _↕	CCIx _↕	OUT _↕	COV _↕	CCIFGx _↕

该寄存器在**POR**信号后全部复位，但在**PUC**信号后不受影响。

CAPTMOD1~0 选择捕获模式

- 00 禁止捕获模式；_↕
- 01 上升沿捕获；_↕
- 10 下降沿捕获；_↕
- 11 上升沿与下降沿都捕获。

CCIS1~0捕获模式下选择捕获事件输入源

- 00 选择 CCIxA；
- 01 选择 CCIxB；
- 10 选择 GND；₊
- 11 选择 Vcc。_↕

SCS 选择捕获信号与定时时钟同步、异步关系

- 0 异步捕获；
- 1 同步捕获。

CAP 选择捕获模式还是比较模式

- 0 比较模式；
- 1 捕获模式。

OUTMODx 选择输出模式

000 输出; ↵
001 置位; ↵
010 PWM 翻转/复位;
011 PWM 置位/复位;
100 翻转; ↵
101 复位; ↵
110 PWM 翻转/置位;
111 PWM 复位/置位;

CCIEx 捕获/比较模块中断允许位

0 禁止中断
1 允许中断.

COV 捕获溢出标志

0 没有捕获溢出。
1 发生捕获溢出。

OUT 输出信号

0 输出低电平。
1 输出高电平。

CCIx 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: CCIx 复位。↵

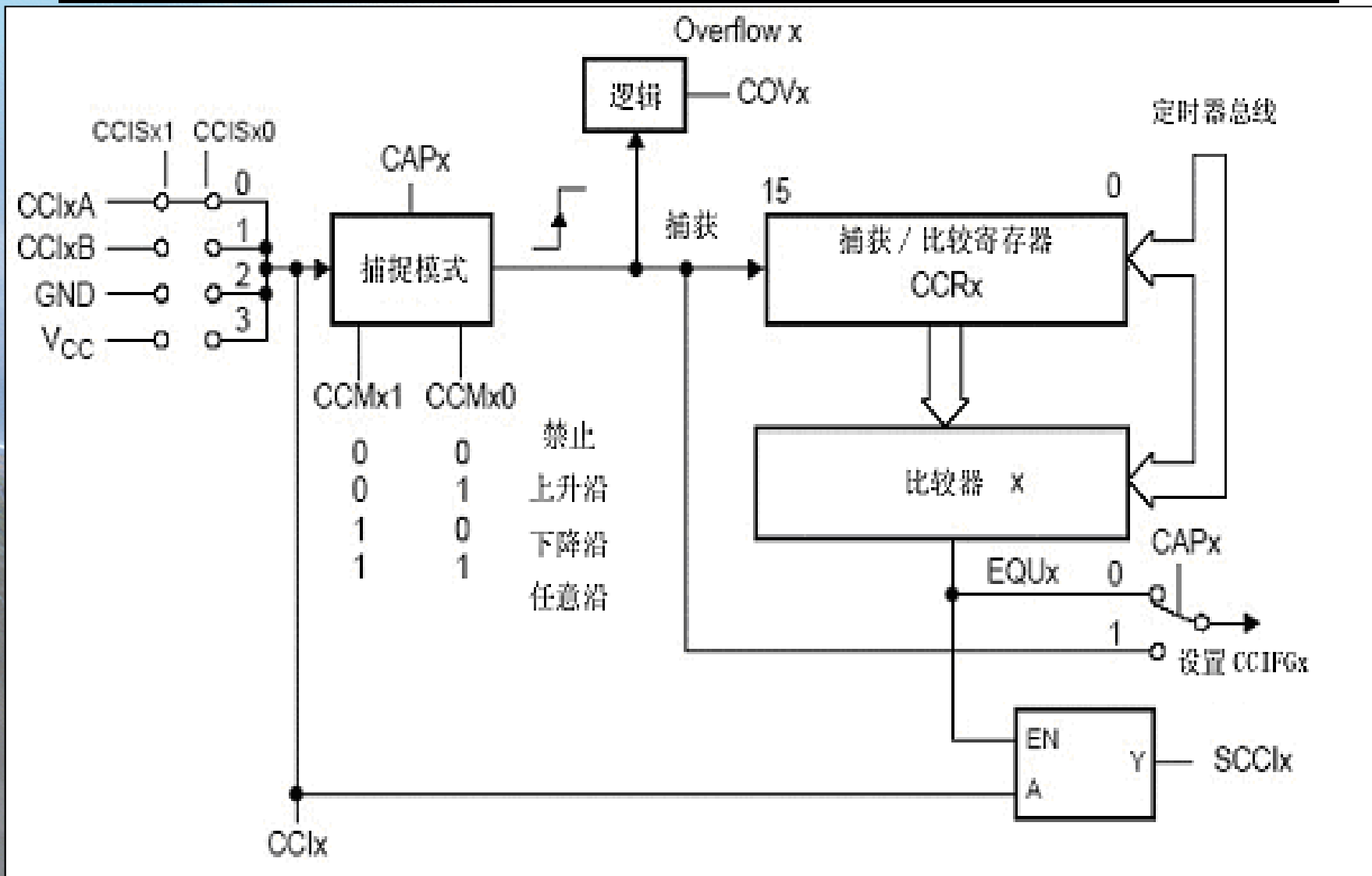
CCIFGx 捕获比较中断标志

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

SCCIx 比较相等信号 EQUx 将选中的捕获\比较输入信号 CCIx (CCIxA, CCIxB, Vcc 和 GND) 进行锁存, 然后可由 SCCIx 读出。

捕获/比较模块

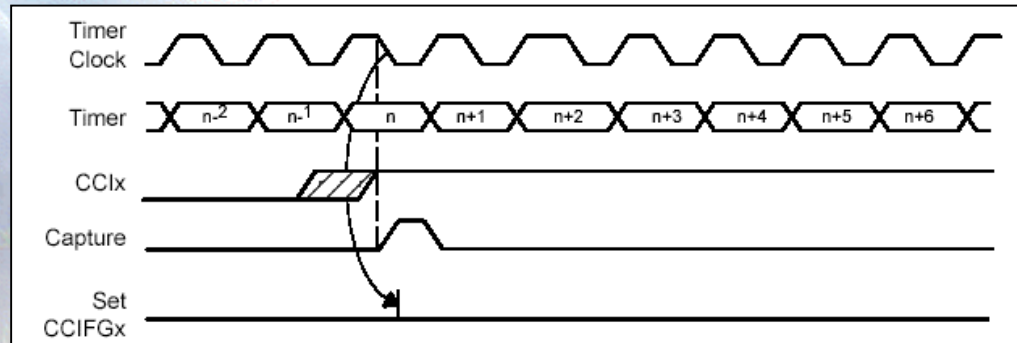


捕获模式

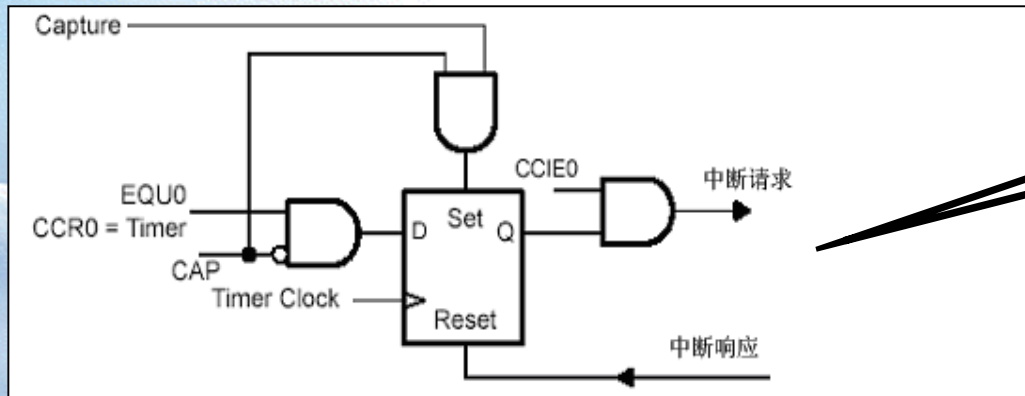
- 当CCTL_x中的CAP_x=1，该模块工作在捕获模式。这时如果在选定的引脚上发生设定的脉冲触发沿（上升沿、下降沿或任意跳变），则TAR中的值将写入到CCR_x中。
- 每个捕获\比较寄存器能被软件用于时间标记。可用于各种目的

测量软件程序所用时间
测量硬件事件之间的时间
测量系统频率

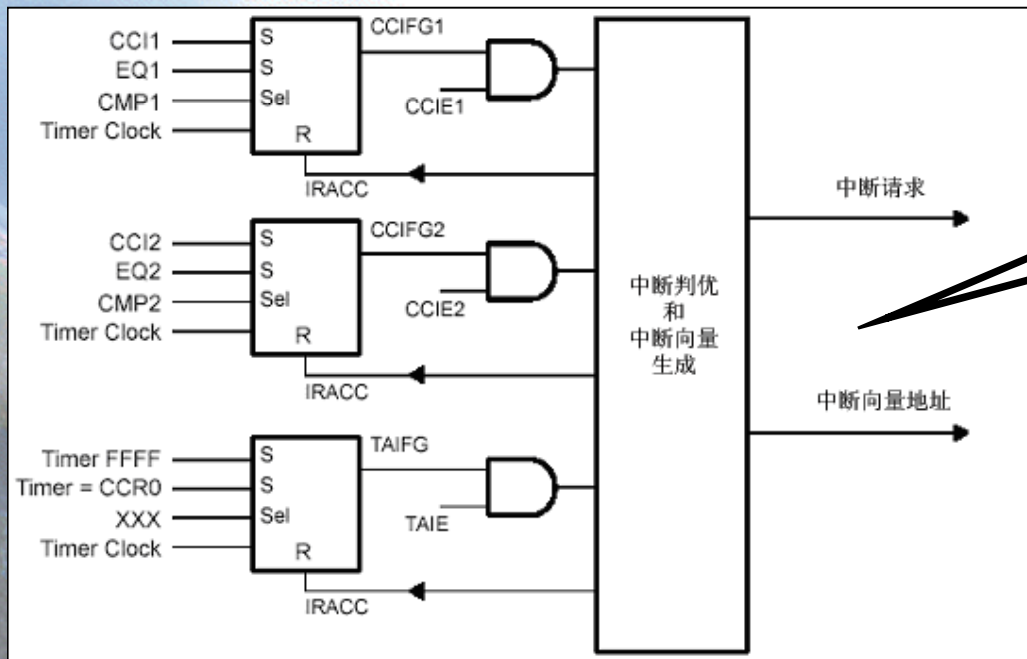
- 当捕获完成后，中断标志位CCIFG_x 被置位。



定时器A中断



CCR0中断

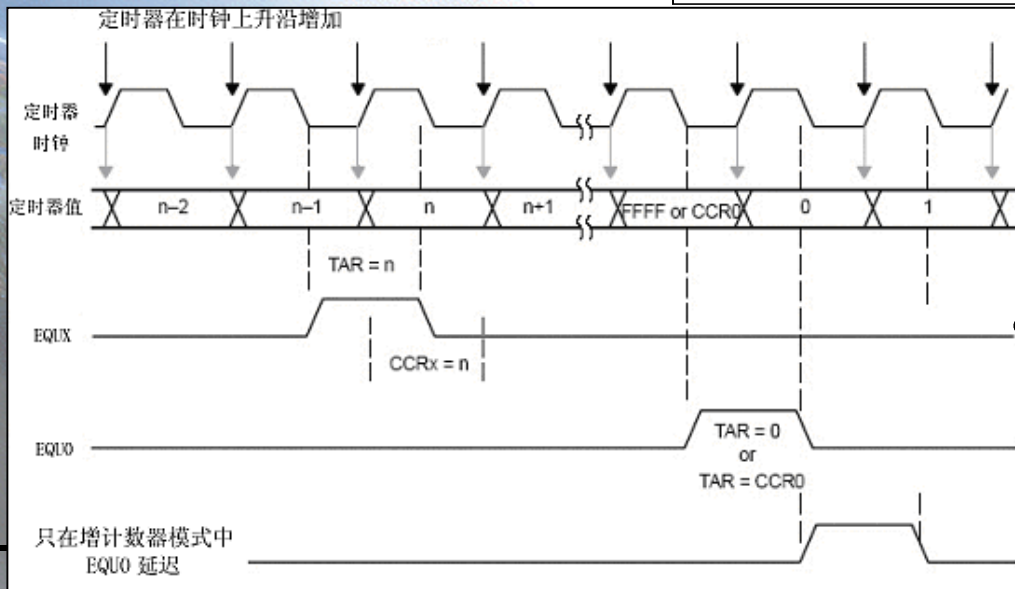
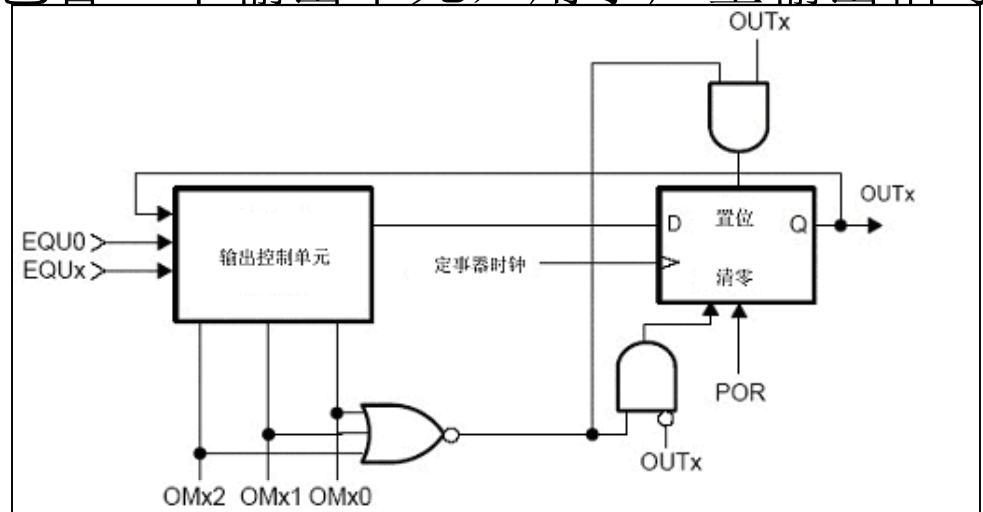


CCR1~CCRx
和定时器中断

输出单元

每个捕获/比较模块包含一个输出单元，用于产生输出信号

输出单元的结构



输出单元时序

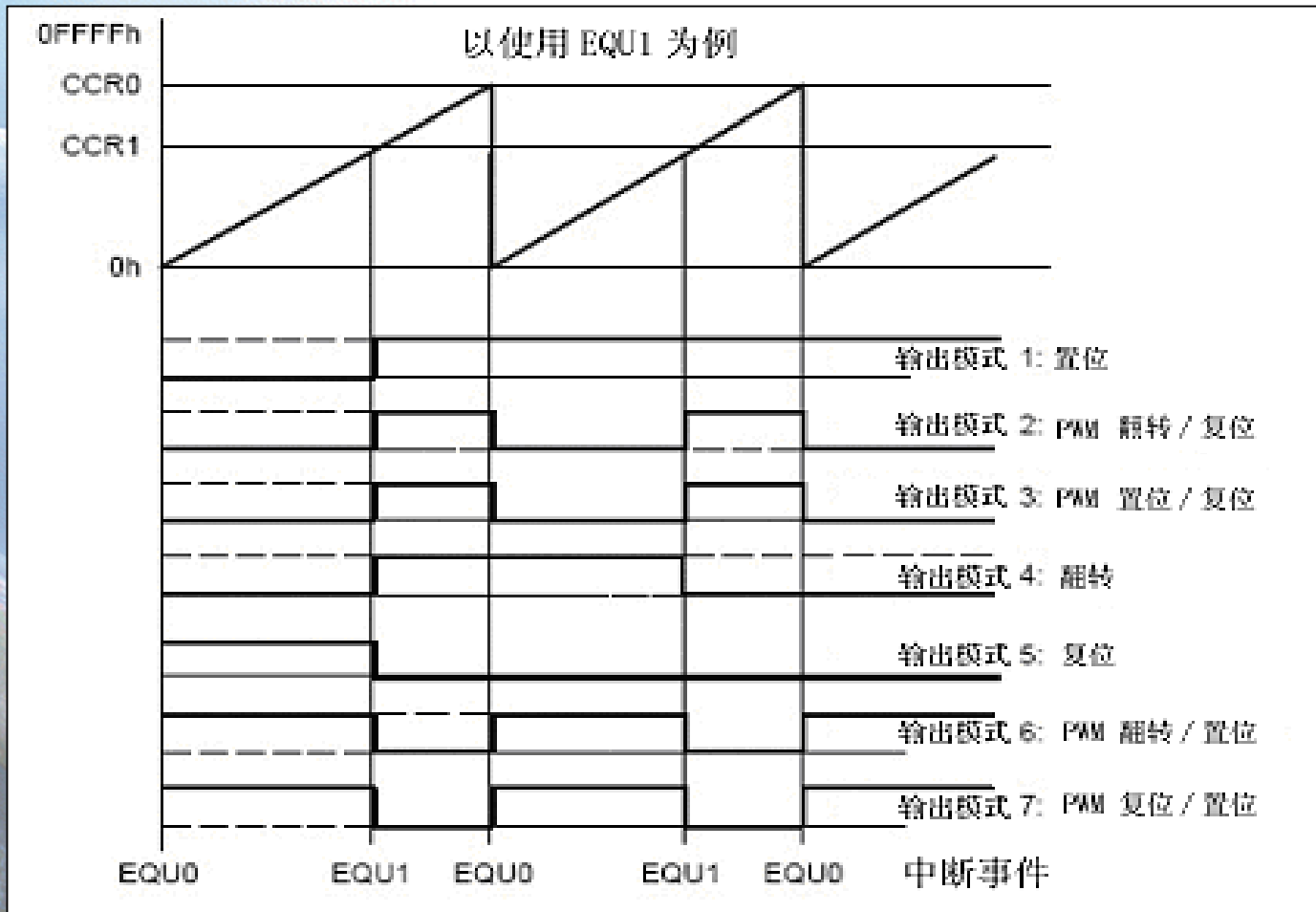
输出模式

- 输出模式0 输出模式：输出信号 OUT_x 由每个捕获/比较模块的控制寄存器 $CCTL_x$ 中的 OUT_x 位定义，并在写入该寄存器后立即更新。最终位 OUT_x 直通。
- 输出模式1 置位模式：输出信号在TAR等于 CCR_x 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 CCR_x 时翻转，当TAR的值等于CCR0时复位。
- 输出模式3 PWM置位/复位模式：输出在TAR的值等于 CCR_x 时置位，当TAR的值等于CCR0时复位。

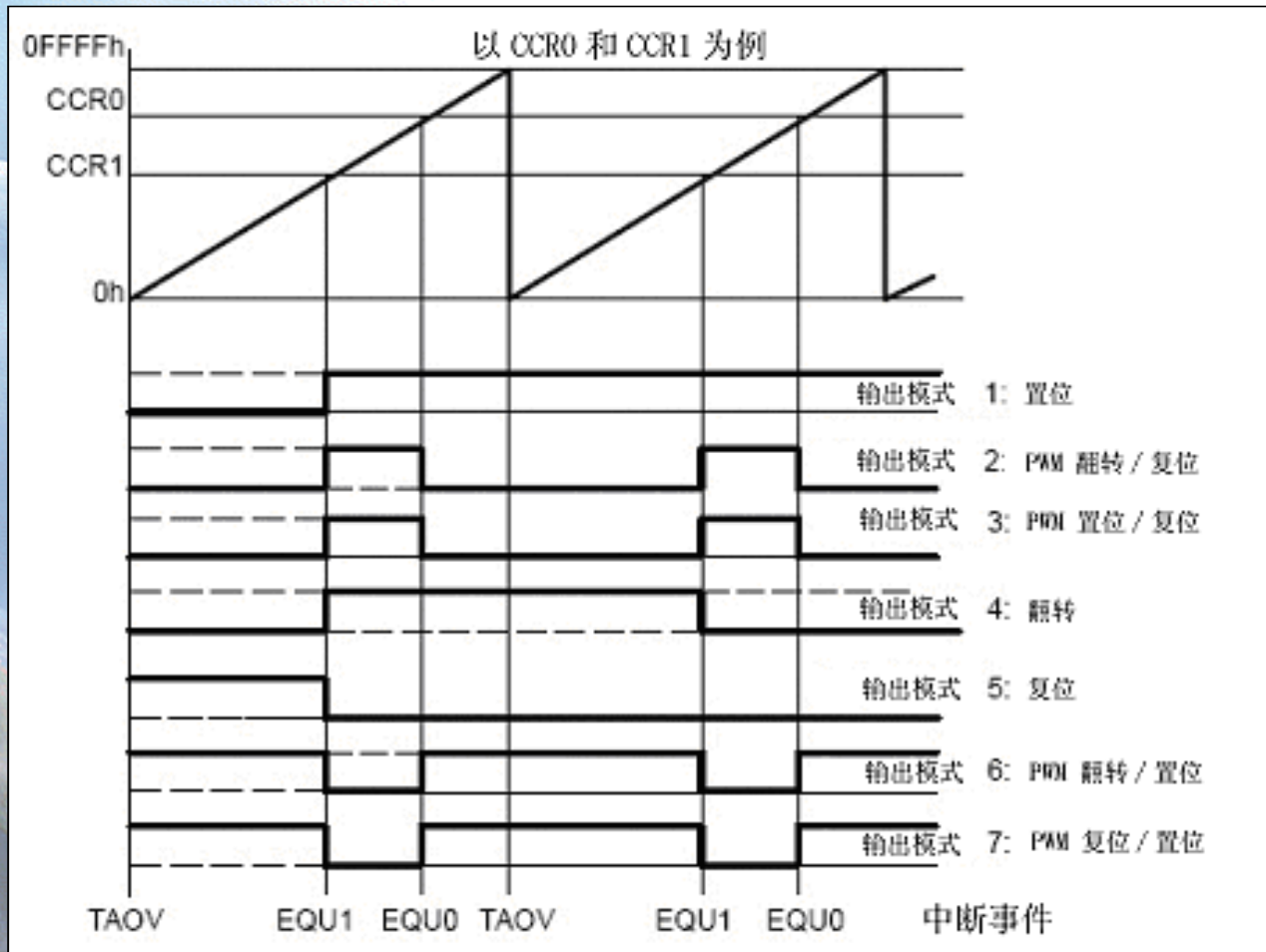
输出模式

- 输出模式4 翻转模式：输出电平在TAR的值等于CCR_x时翻转，输出周期是定时器周期的2倍。
- 输出模式5 复位模式：输出在TAR的值等于CCR_x时复位，并保持低电平直到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CCR_x时翻转，当TAR值等于CCR0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CCR_x时复位，当TAR的值等于CCR0时置位。

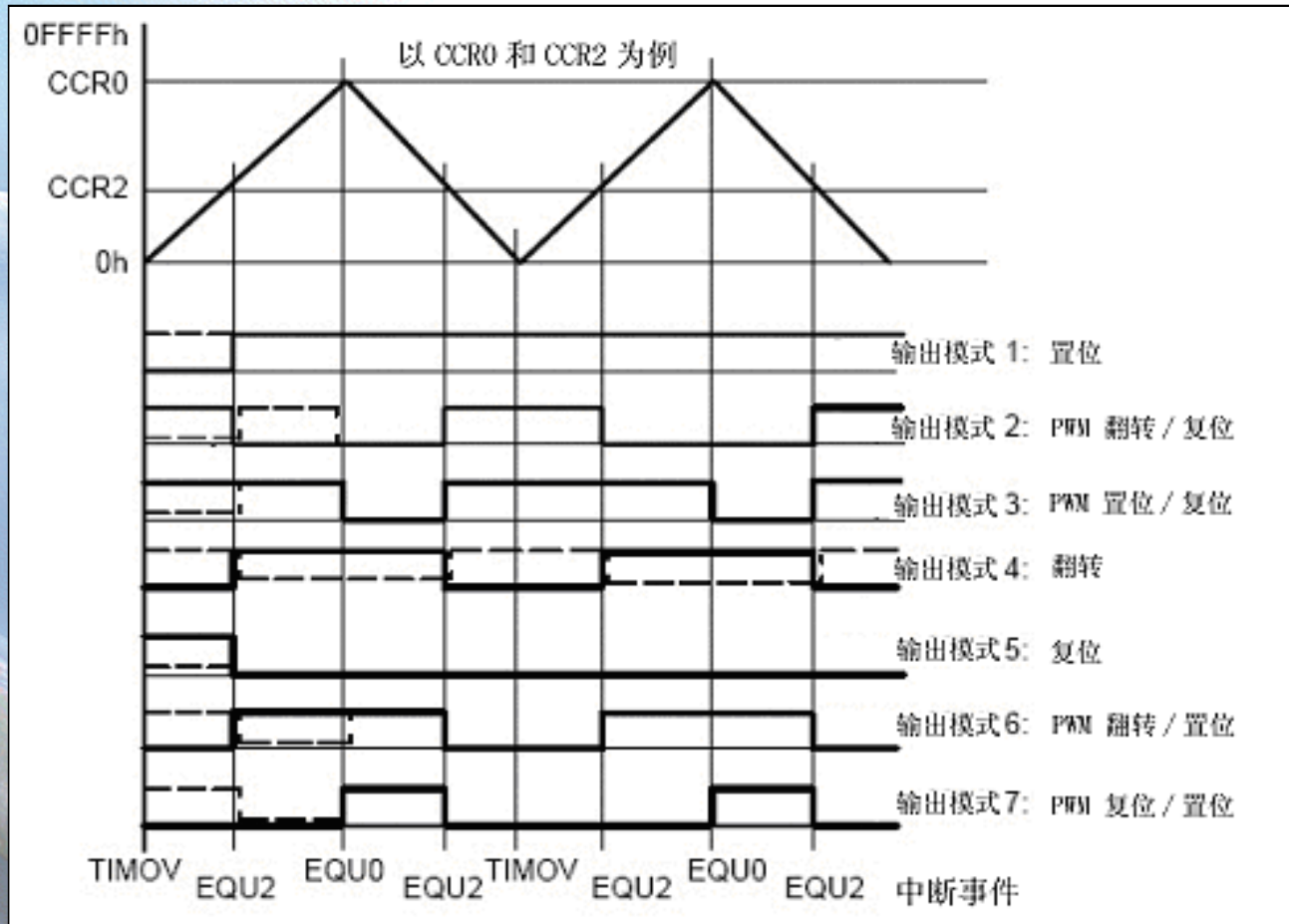
增计数模式输出实例



连续计数模式下的输出波形



增/减计数模式下的输出实例



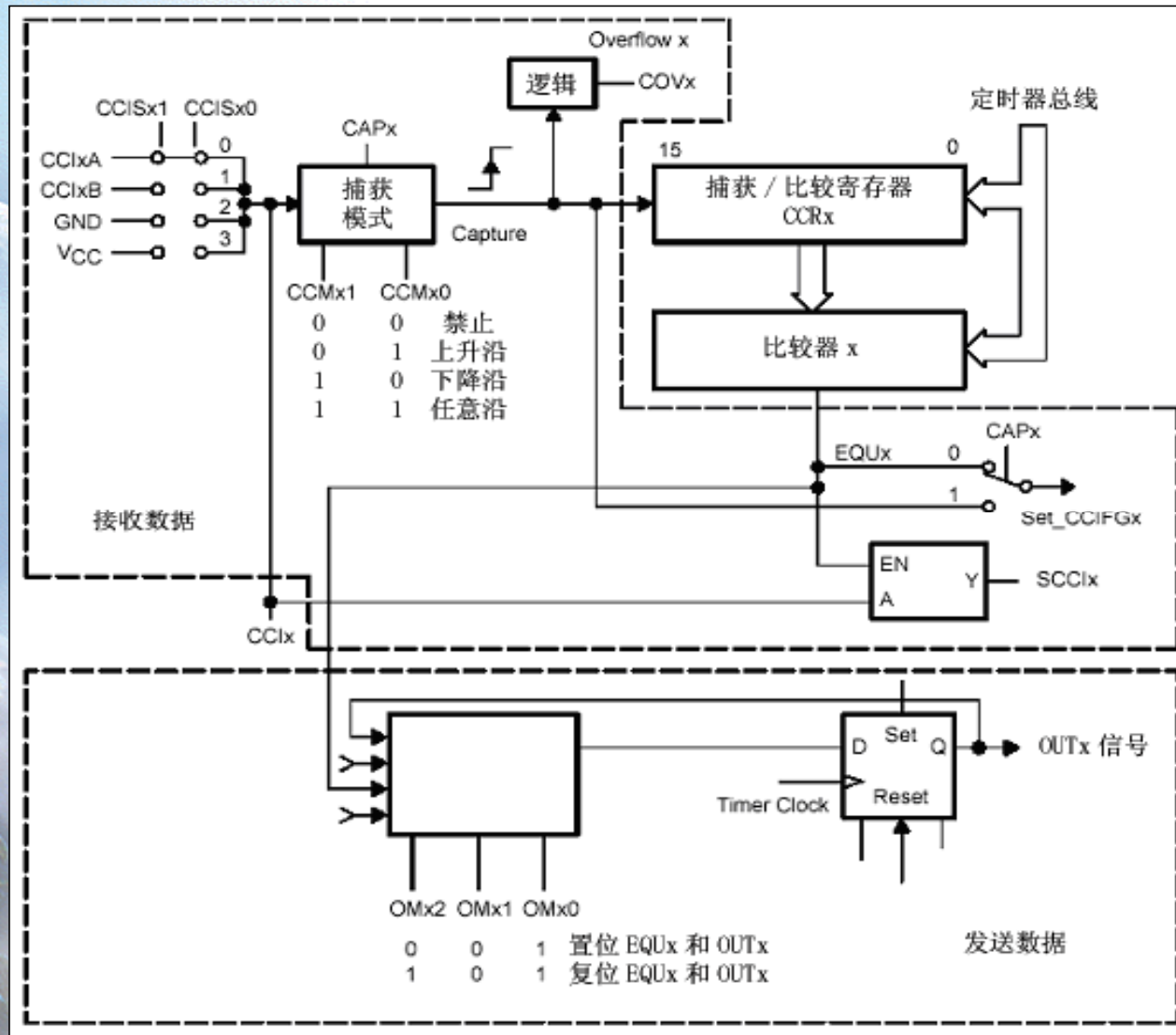
TIMER_A应用——实现异步串行通信

Timer_A具有实现异步串行通信的一些特征

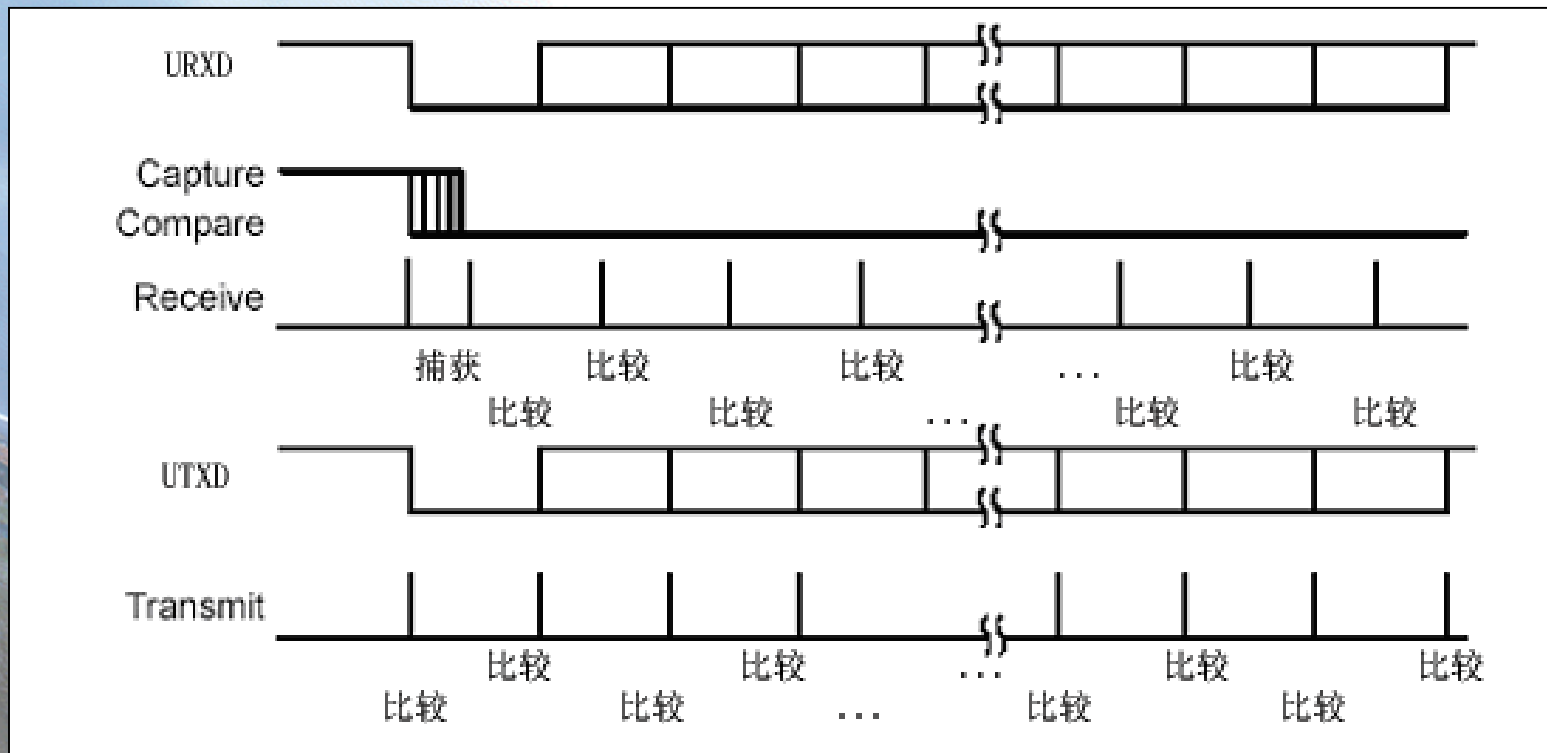
能够自动检测起始位
可以硬件方式产生波特率，范围从75~115200波特
硬件锁存接收和发送的数据
全双工方式

- 捕获功能可以捕捉选定输入引脚的状态的变化，它可以选择捕捉上升沿、下降沿、前后沿。如果捕捉到了相应的变化，则定时器计数值将被复制到捕获比较寄存器 **CCR_x** 中，并会产生相应的中断。在串行通信中正是利用捕获功能的特点来捕捉起始位的信息。
- 比较功能是借助比较器不断地将 **CCR_x** 中的设定值与定时器中的计数值相比较，当二者相等时就产生中断，并产生设定的输出，利用比较功能可以获得精确的时间间隔，利用该特性可以构造一个精确的波特率发生器，为串行通信提供时间基准。

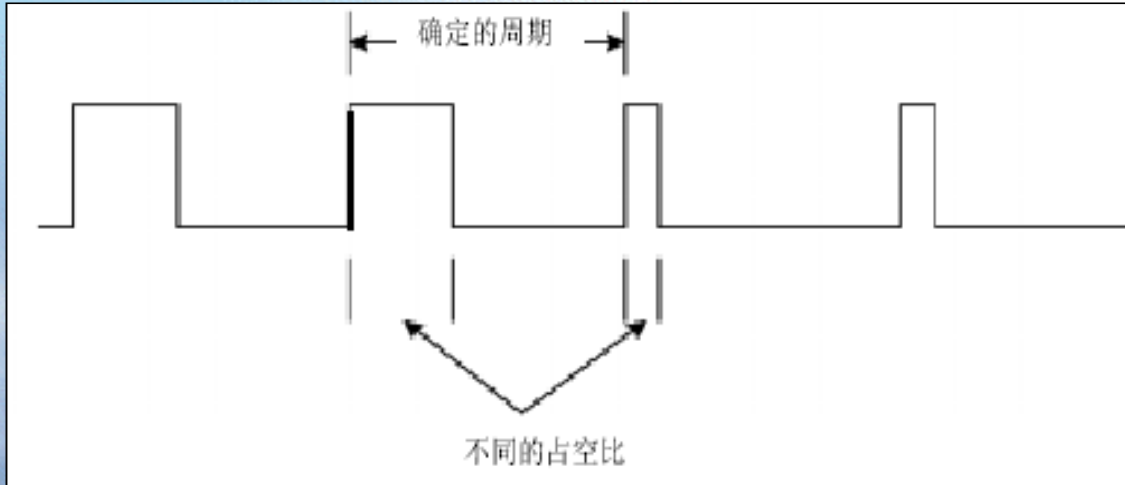
TIMER_A应用——实现异步串行通信



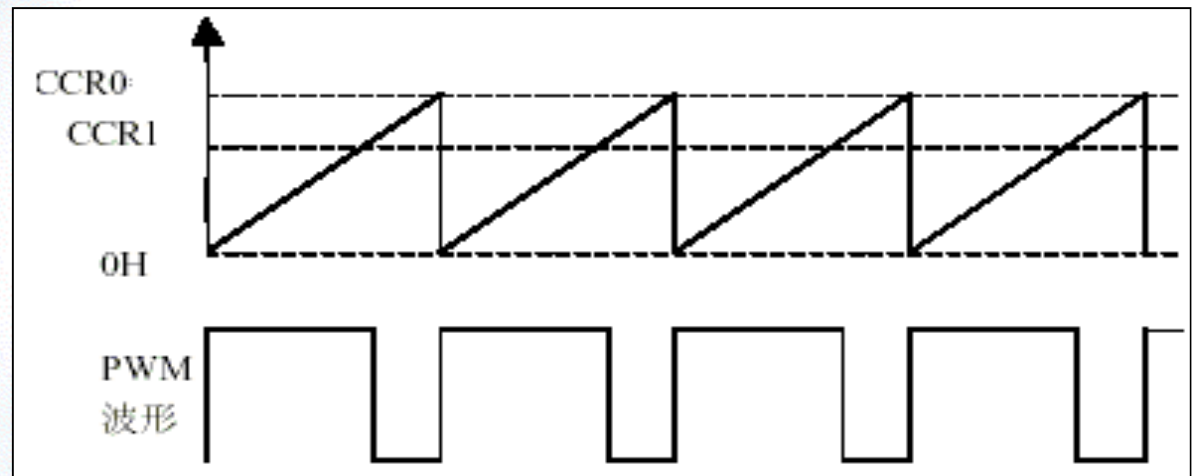
TIMER_A应用——实现异步串行通信



Timer_A实现PWM



PWM信号示意



Timer_A产生 PWM

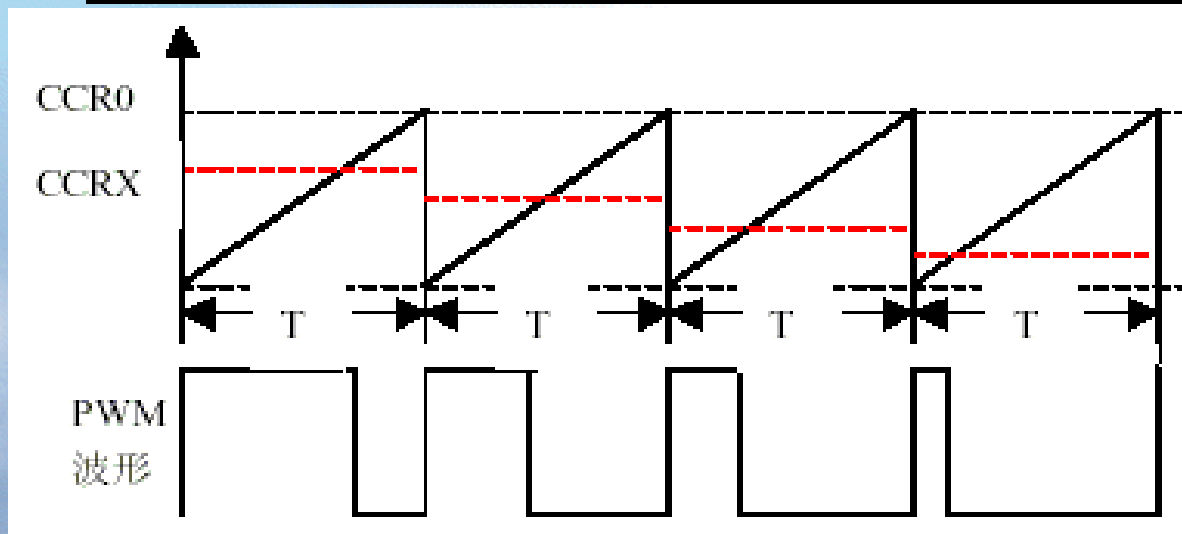
Timer_A实现PWM举例

例：设 $ACLK = TACLK = LFXT1 = 32768$, $MCLK = SMCLK = DCOCLK = 32 \times ACLK = 1.048576\text{Mhz}$ ，利用Timer_A输出周期为 $512 / 32768 = 15.625\text{ms}$ 、占空比分别为75%和25%的PWM矩形波：

```
#include <msp430x44x.h>
void main(void) {
WDTCTL = WDTPW +WDTHOLD;
FLL_CTL0 |= XCAP14PF;
TACTL = TASSEL0 + TACLR;           // ACLK, 清除 TAR
CCR0 = 512-1;                      // PWM周期
CCTL1 = OUTMOD_7;
CCR1 = 384;                          //占空比 384/512=0.75
CCTL2 = OUTMOD_7;
CCR2 = 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;                       // Timer_A 增计数模式
    for (;;) {
        _BIS_SR(LPM3_bits);         // 进入 LPM3
        _NOP();                      }}

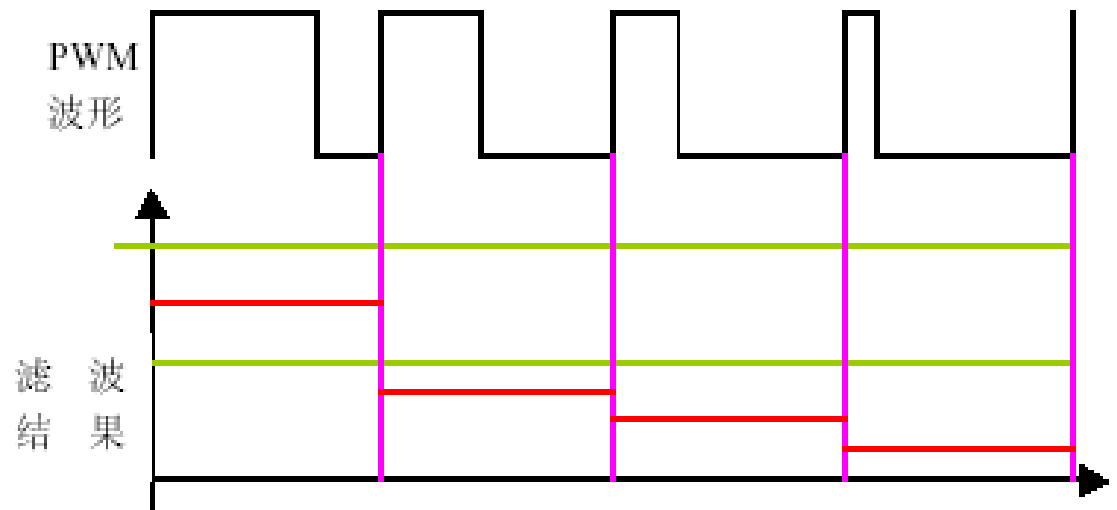
```

PWM信号

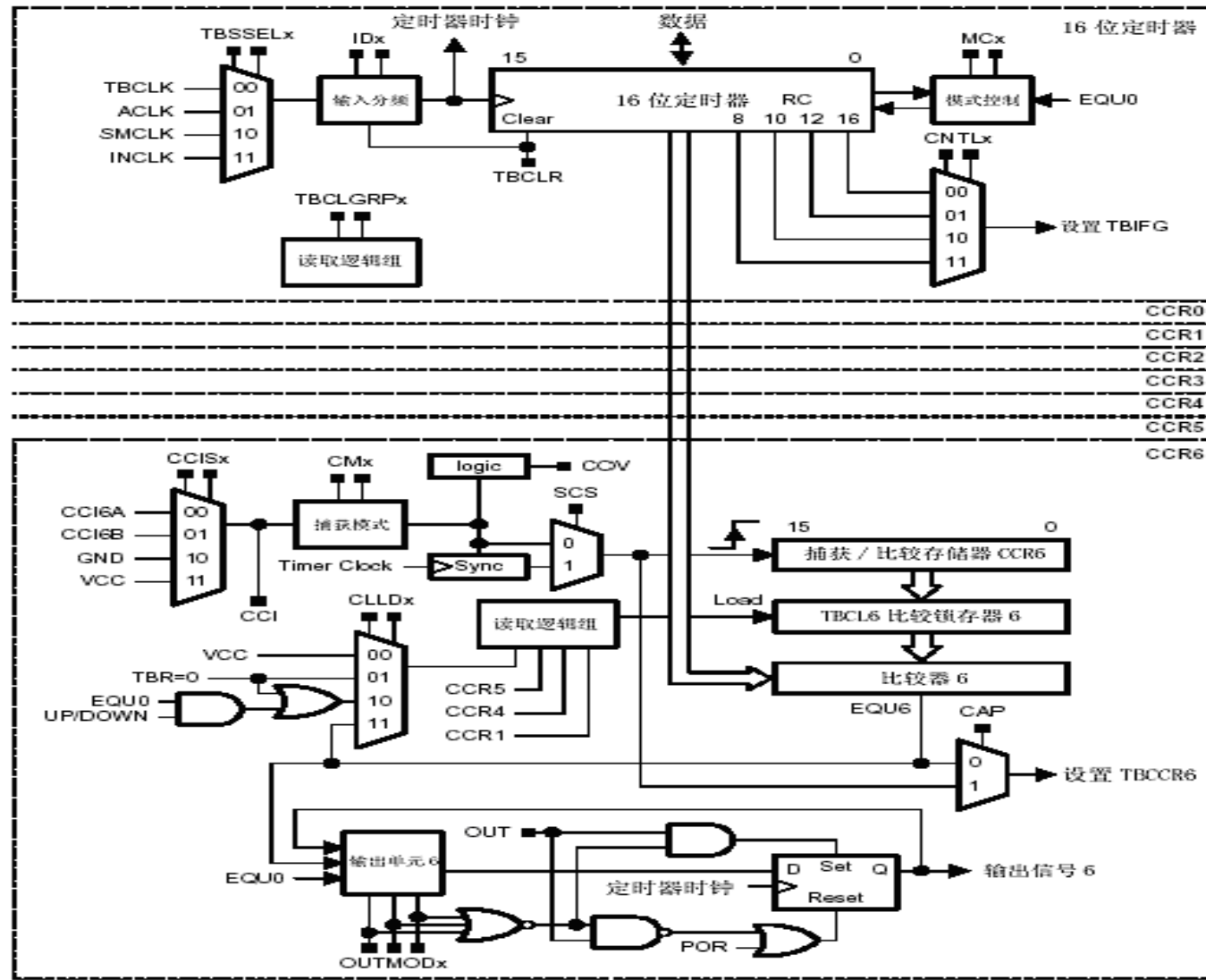


调整PWM信号
占空比

PWM信号经滤波
输出



TIMER_B



Timer_B和Timer_A共同的特征

- | 4种工作模式
- | 具有可选，可配置的计数器输入时钟源
- | 有多个独立可配置捕获/比较模块
- | 有多个具有8种输出模式的配置输出单元
- | DMA使能
- | 中断功能强大，中断可能源自于计数器的溢出，也可能源自于各捕获/比较模块上发生的捕获事件或比较事件。

Timer_B与Timer_A不同之处

- | Timer_B计数长度为8位，10位，12位和16位可编程，而Timer_A的计数长度固定为16位。
 - | Timer_B中没有实现Timer_A中的SCCI寄存器位的功能。
 - | Timer_B在比较模式下的捕获/比较寄存器功能与Timer_A不同，增加了比较锁存器
 - | 有些型号芯片中的Timer_B输出实现了高阻输出
 - | 比较模式的原理稍有不同：在Timer_A中，CCR_x寄存器中保存与TAR相比较的数据；而在Timer_B中，CCR_x寄存器中保存的是要比较的数据，但并不直接与定时器TBR相比较，而是将CCR_x送到与之相对应的锁存器之后，由锁存器与定时器TBR相比较。从捕获/比较寄存器向比较锁存器传输数据的时机也是可以编程的，可以是在写入捕获/比较寄存器后立即传输，也可以是由一个定时事件来触发。
 - | Timer_B支持多重的、同步的定时功能；多重的捕获/比较功能；多重的波形输出功能（比如PWM信号）。而且，通过对比较数据的两级缓冲，可以实现多个PWM信号周期的同步更新
-

| 液晶的发现是由奥地利植物学家**F· Reinetzer**在一百年前完成的

| **1961**年，美国**RCA**公司普林斯顿试验室的一个年轻电子学者**F· Heimeier** 领导开始研究

| **70**年代中期，液晶显示已经形成一个产业

| 液晶——液体晶状物

| 液晶显示器——两块玻璃中间夹了一层（或多层）液晶材料

| 液晶的物理特性，通电时导通，排列变得有秩序，使光线容易通过；不通电时排列混乱，阻止光线通过。通过和不通过的组合就可以在屏幕上显示出图像来。

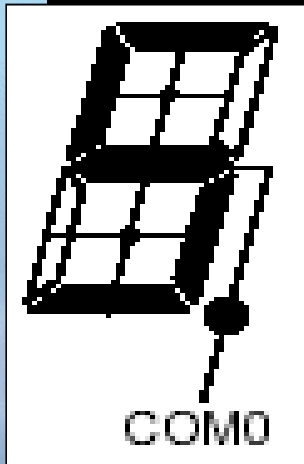
液晶驱动模块

- | MSP430的一个片内外围模块
- | 按照显示缓存的内容来产生SEG和COM信号。
- | 含有驱动外部直接相连的LCD的全部功能模块。

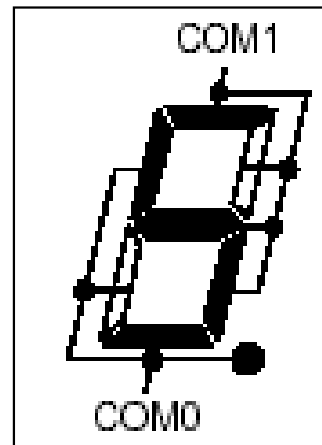
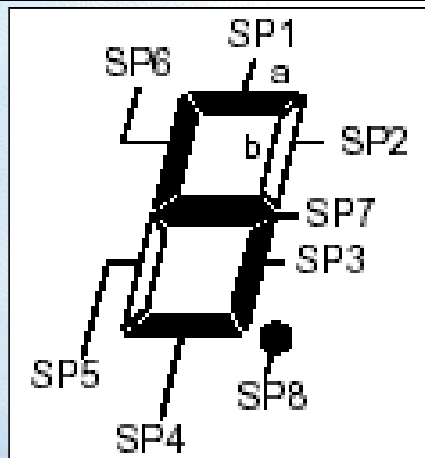
MSP430液晶驱动模块主要特点

- | 具有显示缓存器
 - | 所需的SEG、COM信号自动产生
 - | 4种驱动方法
 - | 多种扫描频率
 - | 段输出端口可以切换为通常输出端口
 - | 显示缓存器可作为一般存储器
 - | 用ACLK经Basic Timer产生频率
-

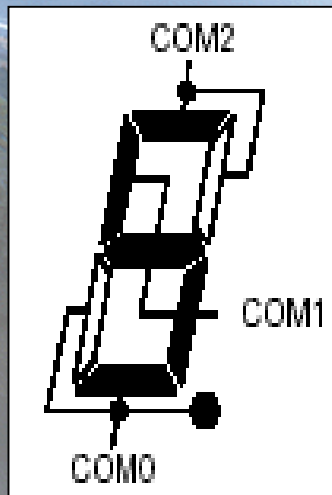
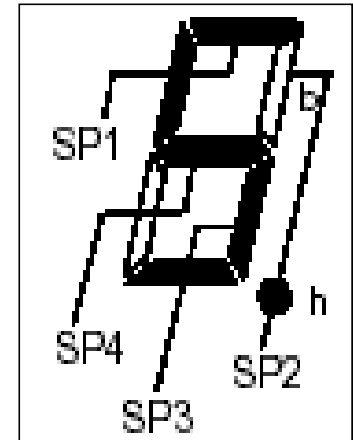
液晶驱动方法



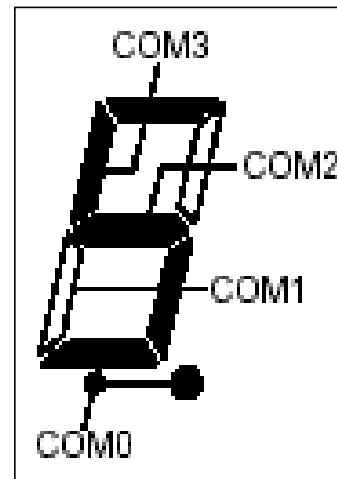
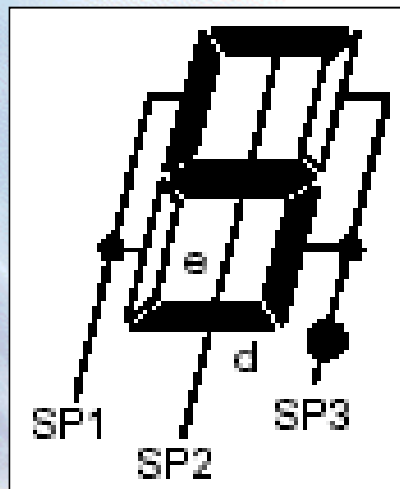
静态驱动



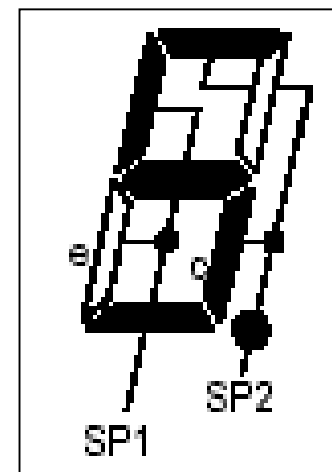
2MUX 驱动



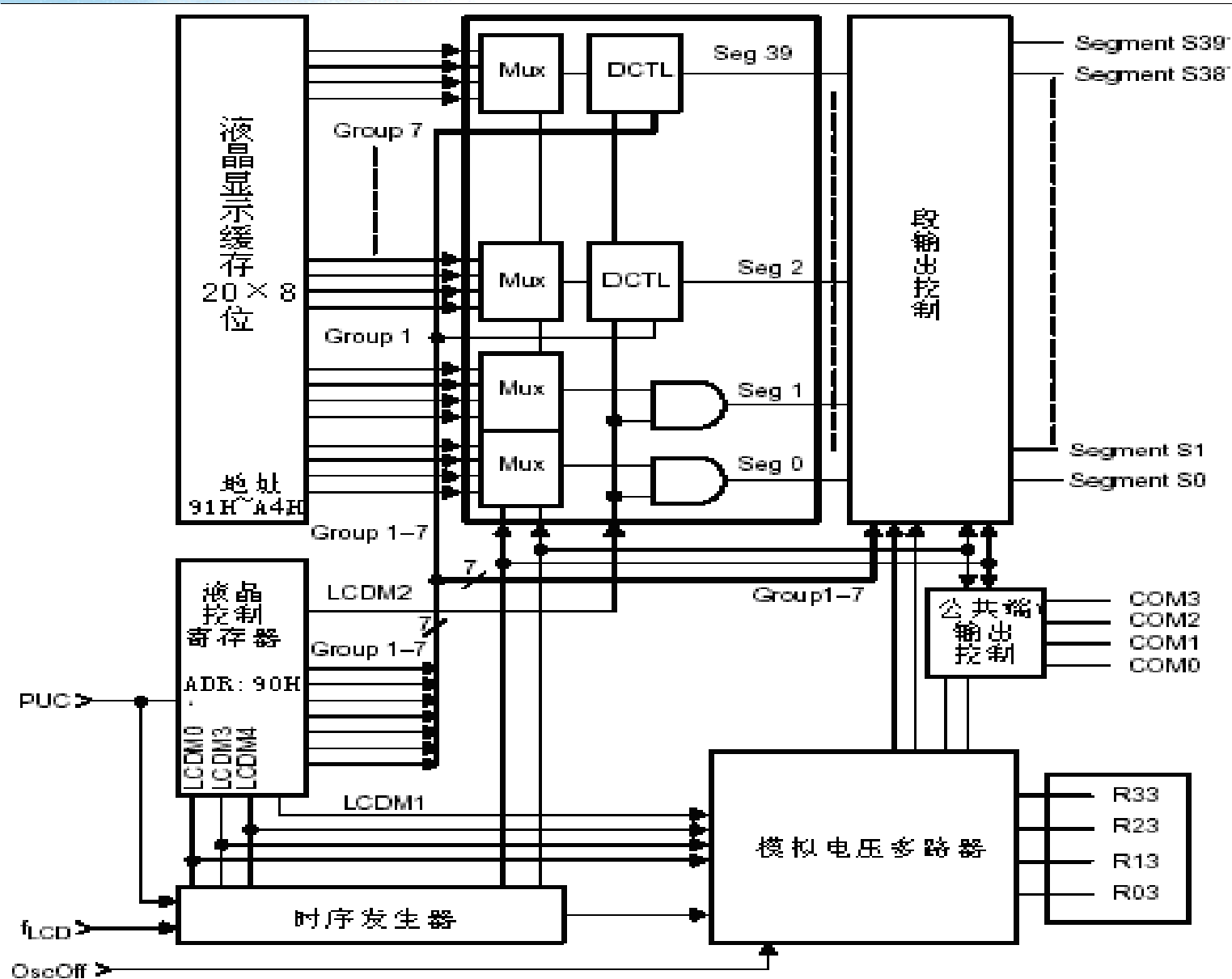
3MUX 驱动

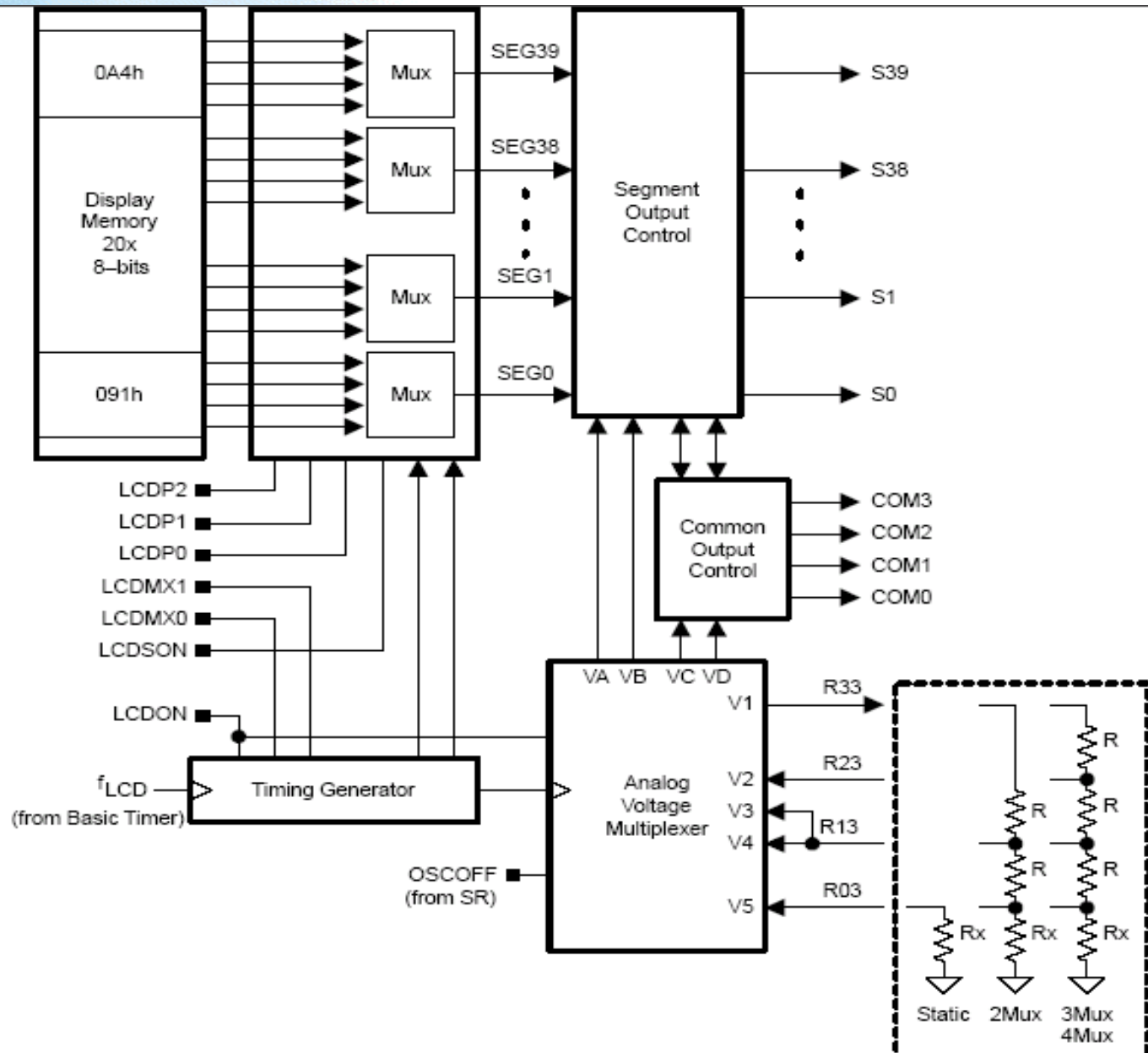


4MUX 驱动



液晶驱动模块功能结构





液晶显示缓存器和段、公共极输出控制

- 液晶显示缓存器各个位与液晶的段一一对应。存储位置位则可以点亮对应的液晶段，存储位复位液晶段变暗。段、公共极输出控制能够自动从显示缓存器读取数据，送出相应信号到液晶玻璃片上。因为不同器件驱动液晶的段数不同，所以液晶显示缓存器的数量也不一样。数量越大，驱动能力越强，显示的内容就越多。

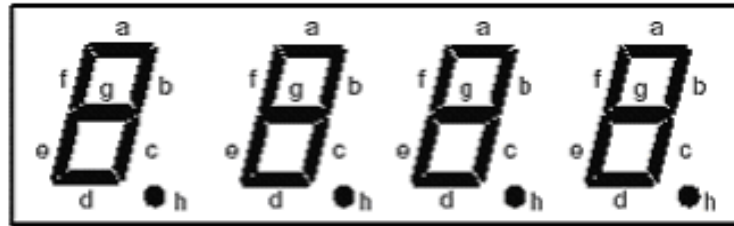
Associated Common Pin	3	2	1	0	3	2	1	0	Associated *4xx Segment Line	
Address	7							0	n	
0A4h	--	--	--	--	--	--	--	--	38	39, 38
0A3h	--	--	--	--	--	--	--	--	36	37, 36
0A2h	--	--	--	--	--	--	--	--	34	35, 34
0A1h	--	--	--	--	--	--	--	--	32	33, 32
0A0h	--	--	--	--	--	--	--	--	30	31, 30
09Fh	--	--	--	--	--	--	--	--	28	29, 28
09Eh	--	--	--	--	--	--	--	--	26	27, 26
09Dh	--	--	--	--	--	--	--	--	24	25, 24
09Ch	--	--	--	--	--	--	--	--	22	23, 22
09Bh	--	--	--	--	--	--	--	--	20	21, 20
09Ah	--	--	--	--	--	--	--	--	18	19, 18
099h	--	--	--	--	--	--	--	--	16	17, 16
098h	--	--	--	--	--	--	--	--	14	15, 14
097h	--	--	--	--	--	--	--	--	12	13, 12
096h	--	--	--	--	--	--	--	--	10	11, 10
095h	--	--	--	--	--	--	--	--	8	9, 8
094h	--	--	--	--	--	--	--	--	6	7, 6
093h	--	--	--	--	--	--	--	--	4	5, 4
092h	--	--	--	--	--	--	--	--	2	3, 2
091h	--	--	--	--	--	--	--	--	0	1, 0

Sn+1
Sn

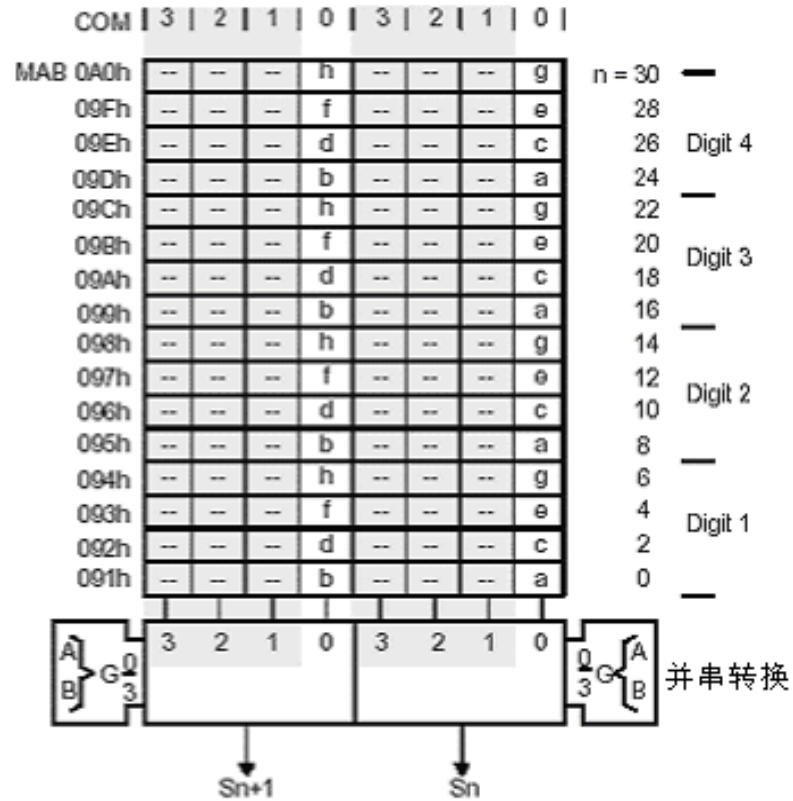
静态方式显示缓存器中位与液晶段的对应关系

输出引脚与显示元件的连接

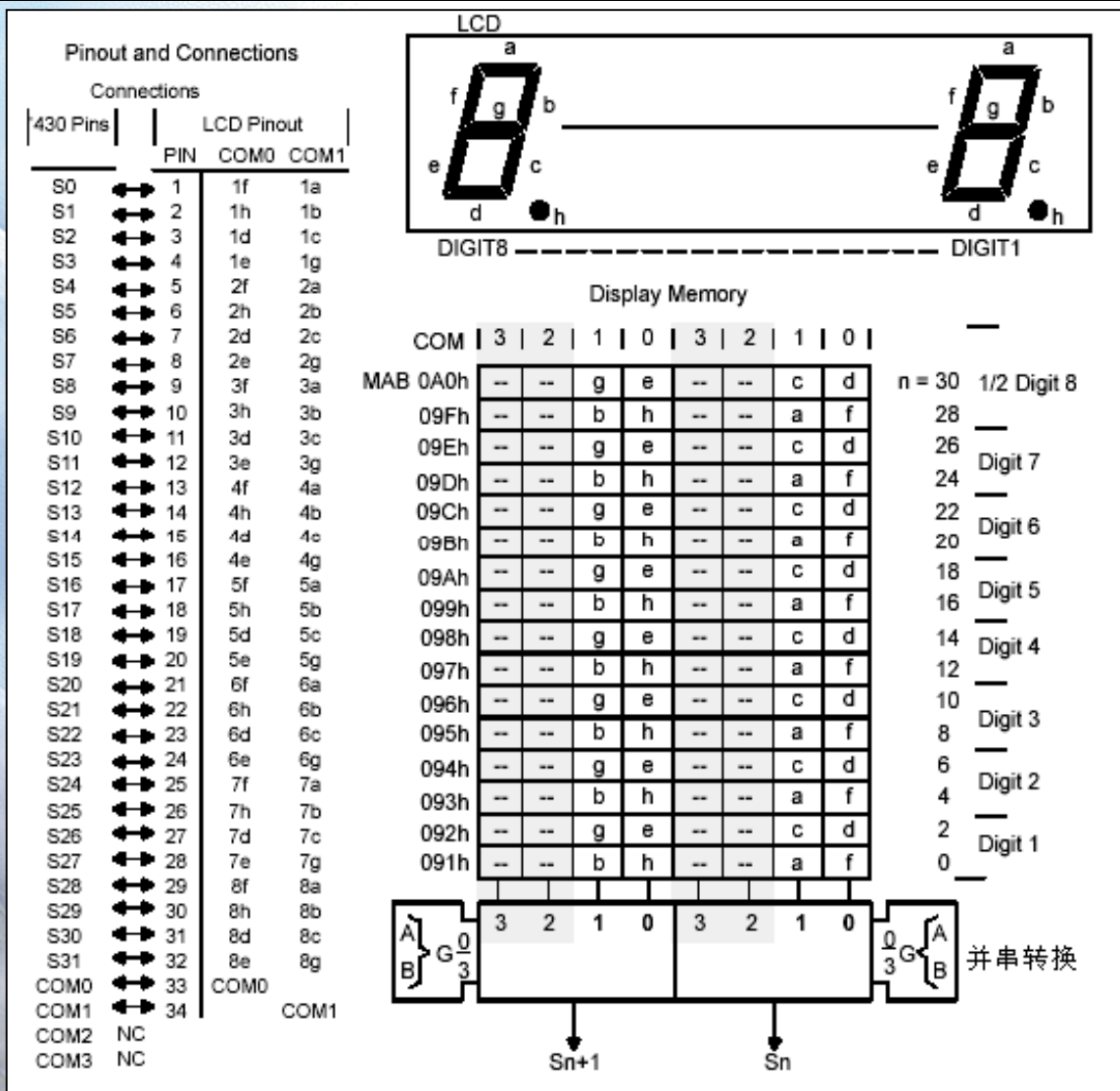
430 Pins		LCD Pinout	
PIN	COM0	PIN	COM0
S0	↔	1	1a
S1	↔	2	1b
S2	↔	3	1c
S3	↔	4	1d
S4	↔	5	1e
S5	↔	6	1f
S6	↔	7	1g
S7	↔	8	1h
S8	↔	9	2a
S9	↔	10	2b
S10	↔	11	2c
S11	↔	12	2d
S12	↔	13	2e
S13	↔	14	2f
S14	↔	15	2g
S15	↔	16	2h
S16	↔	17	3a
S17	↔	18	3b
S18	↔	19	3c
S19	↔	20	3d
S20	↔	21	3e
S21	↔	22	3f
S22	↔	23	3g
S23	↔	24	3h
S24	↔	25	4a
S25	↔	26	4b
S26	↔	27	4c
S27	↔	28	4d
S28	↔	29	4e
S29	↔	30	4f
S30	↔	31	4g
S31	↔	32	4h
S31	↔	32	4h
COM0	↔	33	COM0
COM1	NC		
COM2	NC		
COM3	NC		



显示缓存器



2MUX方式显示缓存器中位与液晶段的对应关系



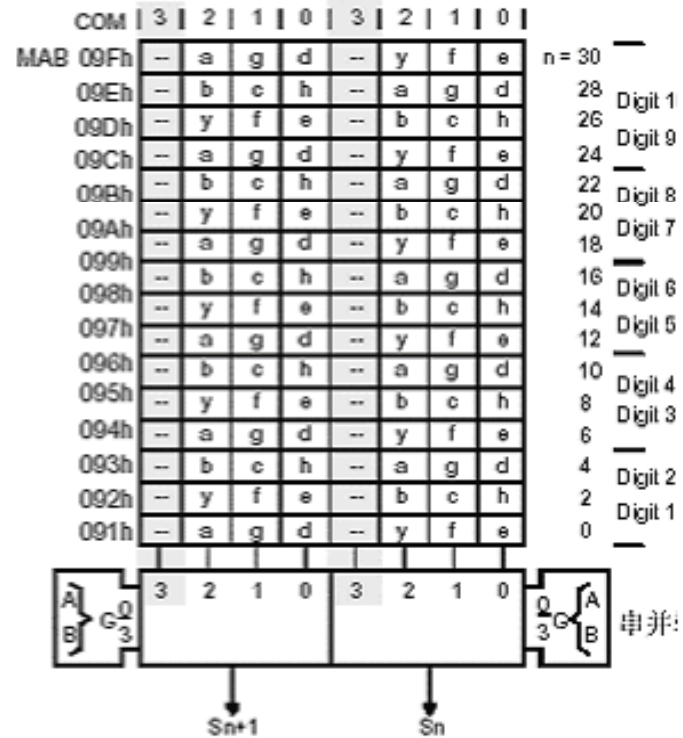
3MUX方式显示缓存器中位与液晶段的对应关系

输出引脚与显示元件的连接

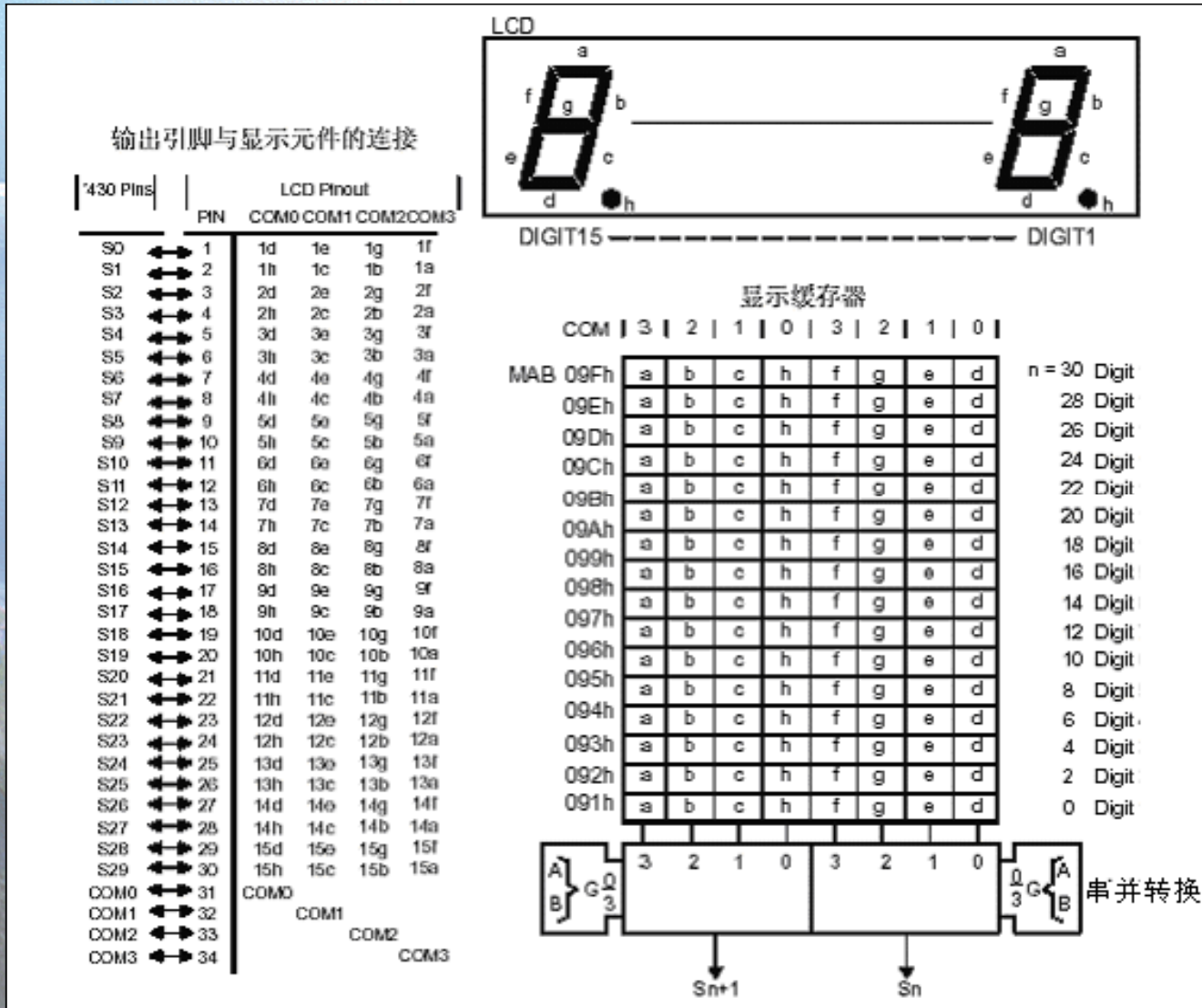
430 Pins	LCD Pinout		
	PIN	COM0	COM1 COM2
S0	1	1e	1f 1y
S1	2	1d	1g 1a
S2	3	1h	1c 1b
S3	4	2e	2f 2y
S4	5	2d	2g 2a
S5	6	2h	2c 2b
S6	7	3e	3f 3y
S7	8	3d	3g 3a
S8	9	3h	3c 3b
S9	10	4e	4f 4y
S10	11	4d	4g 4a
S11	12	4h	4c 4b
S12	13	5e	5f 5y
S13	14	5d	5g 5a
S14	15	5h	5c 5b
S15	16	6e	6f 6y
S16	17	6d	6g 6a
S17	18	6h	6c 6b
S18	19	7e	7f 7y
S19	20	7d	7g 7a
S20	21	7h	7c 7b
S21	22	8e	8f 8y
S22	23	8d	8g 8a
S23	24	8h	8c 8b
S24	25	9e	9f 9y
S25	26	9d	9g 9a
S26	27	9h	9c 9b
S27	28	10e	10f 10y
S28	29	10d	10g 10a
S29	30	10h	10c 10b
COM0	31	COM0	
COM1	32	COM1	
COM2	33	COM2	
COM3	NC		



显示缓存器



4MUX方式显示缓存器中位与液晶段的对应关系



实验板显存结构

h, e, f, c, d, g, b, a

| 0—7BH , 1—12H, 2—4FH, 3—1FH, 4—36

| 5—3DH, 6—7DH, 7—13H, 8—7FH, 9—3FH

在各种驱动方式下，液晶显示所需要的帧频率和 f_{LCD} 的关系为：

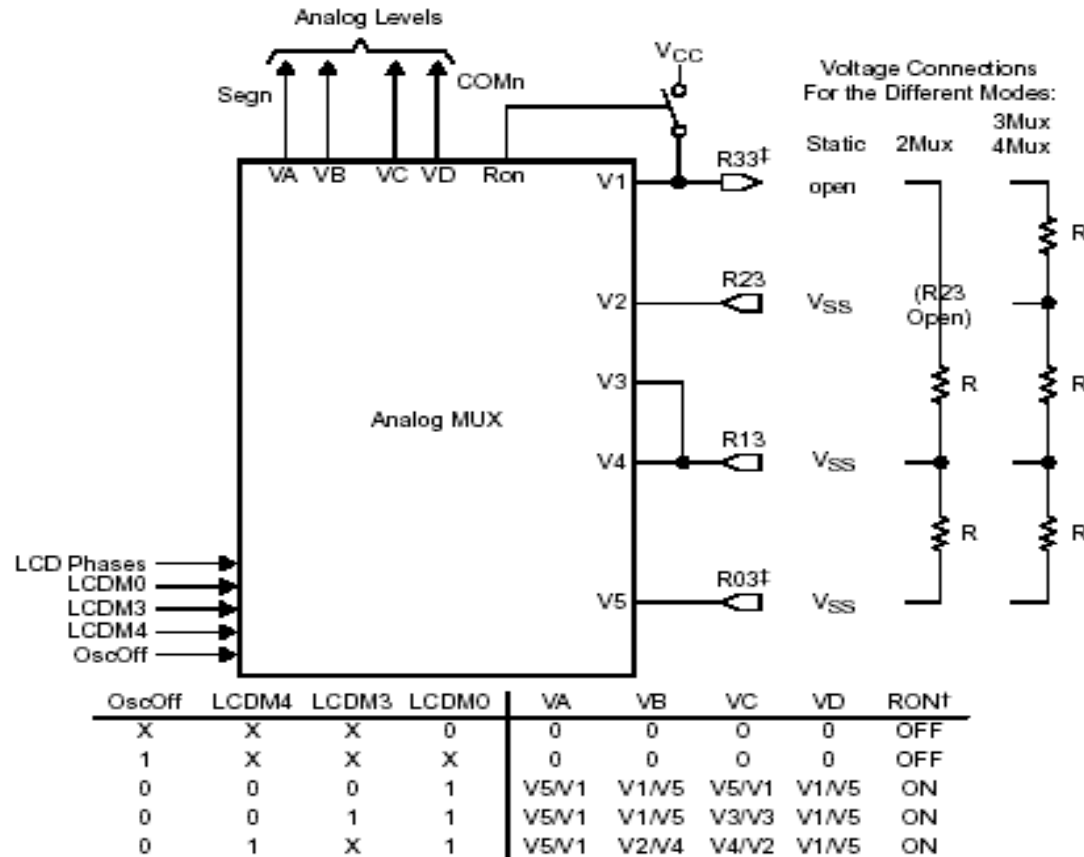
n 静态驱动方式 $f_{frame} = f_{LCD} \div 2$

n 2 MUX驱动方式 $f_{frame} = f_{LCD} \div 4$

n 3 MUX驱动方式 $f_{frame} = f_{LCD} \div 6$

n 4 MUX驱动方式 $f_{frame} = f_{LCD} \div 8$

液晶模拟电压多路器



2个输入端: $R23$ 、 $R13$, $V1 = V_{cc}$, $V5 = V_{ss}$

3个输入端: $R23$ 、 $R13$ 、 $R03$, $V1 = V_{cc}$

4个输入端: $R23$ 、 $R13$ 、 $R03$ 、 $R33$

7↕	6↕	5↕	4↕	3↕	2↕	1↕	0↕
LCDM7↕	LCDM6↕	LCDM5↕	LCDM4↕	LCDM3↕	LCDM2↕	LCDM1↕	LCDM0↕

LCDM7, 6, 5 选择输出段或端口信息的组合, 如表 4-19 所列。选作端口功能的输出由显示各位驱动, 并不再作为 LCD 段线。↕

表4-19 输出段或端口信息组合↕

LCDM7↕	LCDM6↕	LCDM5↕	段功能↕	Group 0↕	Group 1↕	Group 2↕	Group 3↕	Group 4↕	Group 5↕	Group 6↕	Group 7↕
0↕	0↕	0↕	仅用于 端口↕	0↕	0↕	0↕	0↕	0↕	0↕	0↕	0↕
0↕	0↕	1↕	S0~S15↕	1↕	0↕	0↕	0↕	0↕	0↕	0↕	0↕
0↕	1↕	0↕	S0~S19↕	1↕	1↕	0↕	0↕	0↕	0↕	0↕	0↕
0↕	1↕	1↕	S0~S23↕	1↕	1↕	1↕	0↕	0↕	0↕	0↕	0↕
1↕	0↕	0↕	S0~S27↕	1↕	1↕	1↕	1↕	0↕	0↕	0↕	0↕
1↕	0↕	1↕	S0~S31↕	1↕	1↕	1↕	1↕	1↕	0↕	0↕	0↕
1↕	1↕	0↕	S0~S35↕	1↕	1↕	1↕	1↕	1↕	1↕	0↕	0↕
1↕	1↕	1↕	S0~S39↕	1↕	1↕	1↕	1↕	1↕	1↕	1↕	0↕

LCDM4↵	LCDM3↵	LCDM2↵	显示模式↵	LCD 偏压↵
X↵	X↵	0↵	显示关闭, 端口输出保持稳定↵	
0↵	0↵	1↵	静态↵	R33*, R03*↵
0↵	1↵	1↵	2MUX↵	R33*, R13, R03*↵
1↵	0↵	1↵	3MUX↵	R33*, R23, R13, R03*↵
1↵	1↵	1↵	4 MUX↵	R33*, R23, R13, R03*↵

LCDM0 定时发生器开关, 在使用时只有需要才打开 LCD 模块, 休眠状态下通过 LCDM0 = 0 可以关闭 LCD 模块, 有利于系统低功耗。↵

0 定时发生器关闭。COM 线与段驱动端低电平。↵

1 定时发生器打开。COM 线与段驱动端将按照液晶显存的数据输出对应的信号。↵

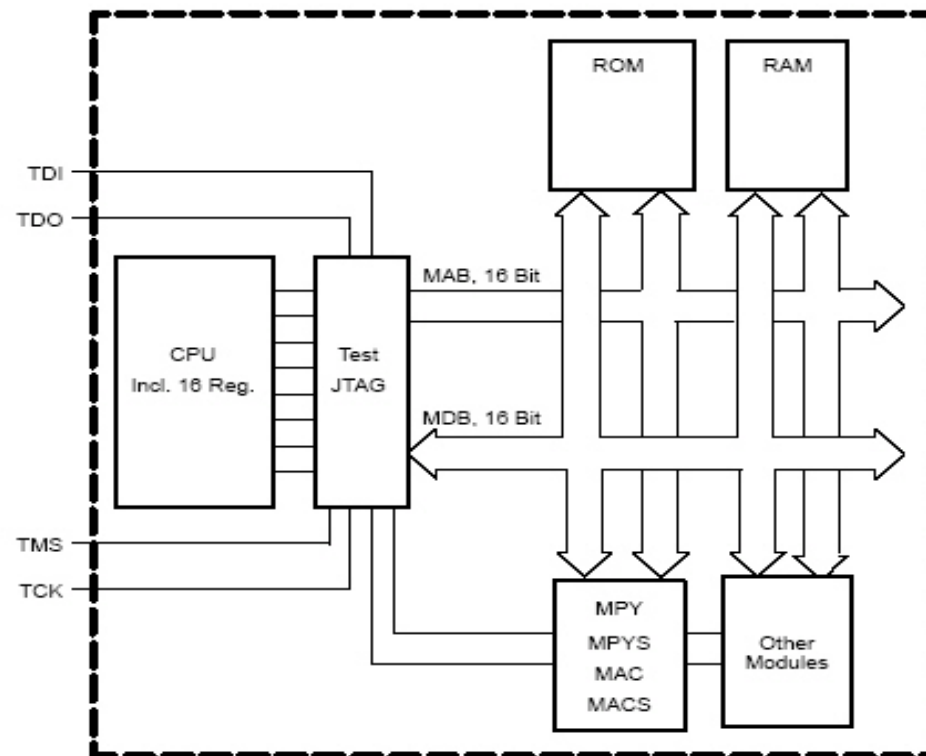
4MUX显示举例

```
#include <msp430x44x.h>
char digit[10] = {
0xEB, /* "0" LCD segments a+b+c+d+e+f , 每个字的8段安排在一个字节中。*/
0X60, /* "1" */
0XC7, /* "2" */
0XE5, /* "3" */
0X6C, /* "4" */
0XAD, /* "5" */
0XAF, /* "6" */
0XE0, /* "7" */
0XEF, /* "8" */
0XED /* "9" */ };
void main(void)
{
    int i;
    WDTCTL = WDTPW + WDTHOLD; // 停看门狗
    FLL_CTL0 |= XCAP14PF; // 配置FLL+
    LCDCTL = LCDON + LCD4MUX + LCDP2; // 4Mux, S0-S17
    BTCTL = BTFRFQ1; // 基本定时器输出fLCD
    P5SEL = 0xFC; // 公共极和 Rxx 选择

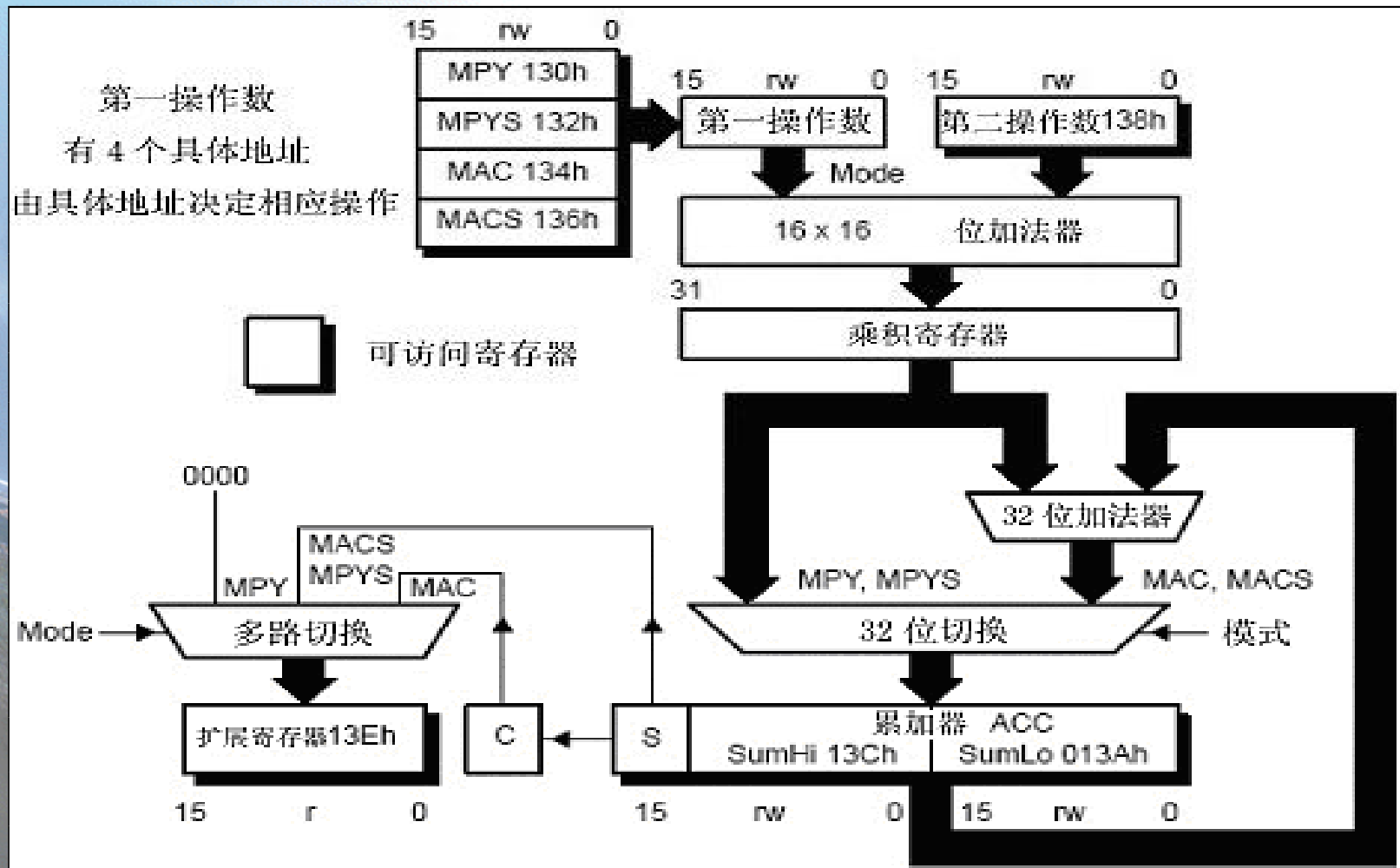
    for (;;)
    {
        for (i=0; i<7; ++i) // 显示"6543210"
            LCDMEM[i] = digit[i];
    }
}
```

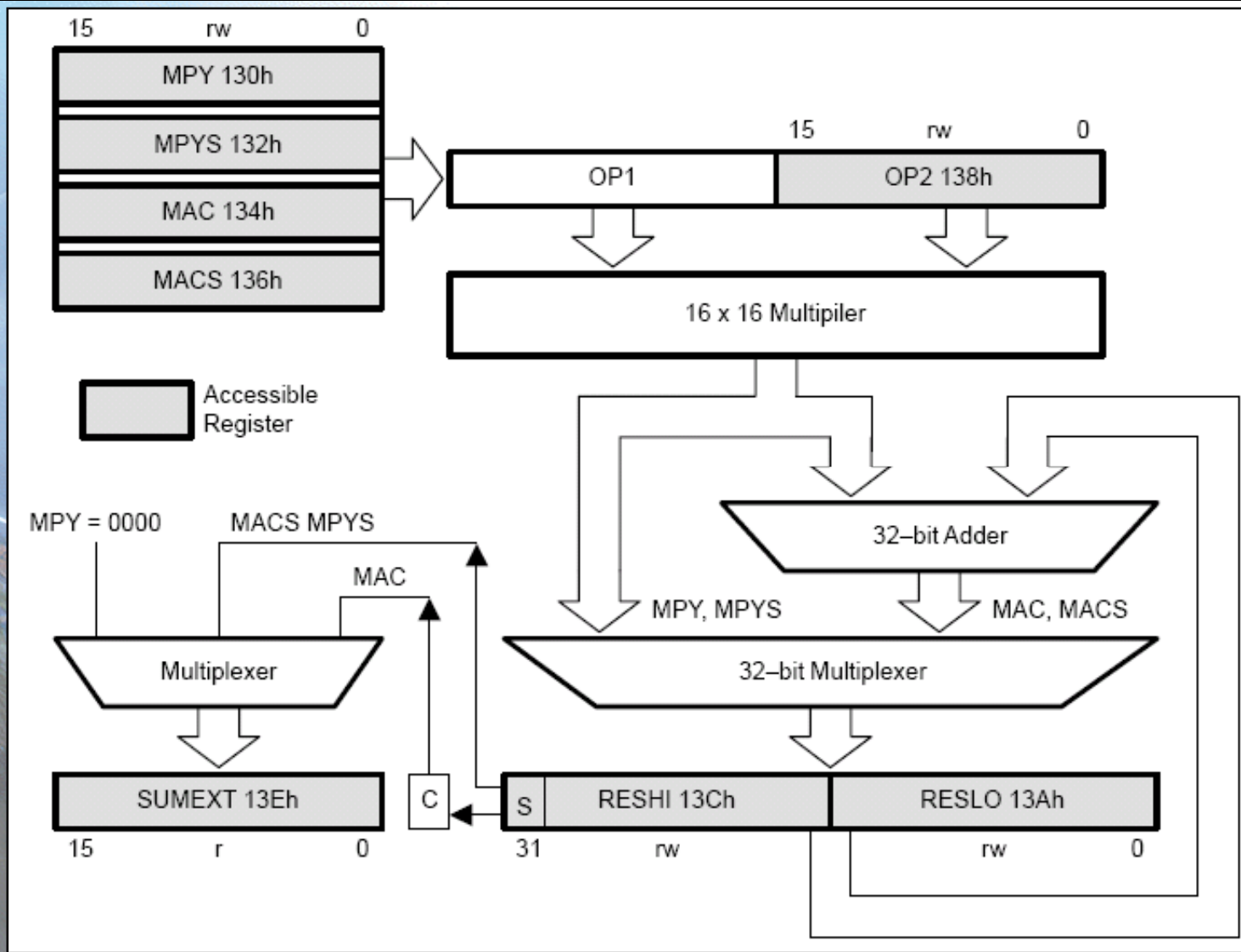
硬件乘法器通过内部总线与CPU相连

- MSP430可以在不改变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。硬件乘法器大大加强了MSP430的功能并提供了软硬件相兼容的范围，提高了数据处理能力。



16×16位硬件乘法器的结构





硬件乘法器的结构

- | 操作数寄存器：OP1和OP2，第一个操作数可来源于4个寄存器：MPY，MPYS，MAC及MACS，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，一般需要4个周期数。
- | 结果寄存器：结果高字寄存器（RESHI）、结果低字寄存器（RESLO）及结果扩展寄存器（SUMEXT）。寄存器RESHI和RESLO的内容为两个16位数相乘的32位乘积结论。而寄存器SUMEXT的内容由执行的乘法模式及乘积的结果决定。

硬件乘法器寄存器

- | MPY 操作数1，指示操作数为无符号数相乘
- | MPYS 操作数1，指示操作数为有符号数相乘。
- | MAC 操作数1，指示操作数为无符号数累加。
- | MACS 操作数1，指示操作数为有符号数累加。
- | OP_2 操作数2。
- | RESLO 结果低字寄存器。
- | RESHI 结果高字寄存器。
- | SUMEXT 结果扩展寄存器。

硬件乘法器使用注意事项

- | 第二个操作数写入完毕，乘法运算就开始。一般在取出结果之前插入1~2条指令，以保证运算时间的需要。
- | 其次，在一个器件中只有一个硬件乘法器，如果遇到多处使用的情况，必须在每一次使用完成后再进行下一次使用
- | 结果扩展寄存器（SUMEXT）的内容，与运算类型及运算结果都有关系。
- | 不论进行何种运算，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的地址依次为：
0130h, 0132h, 0134h, 0136h, 0138h。

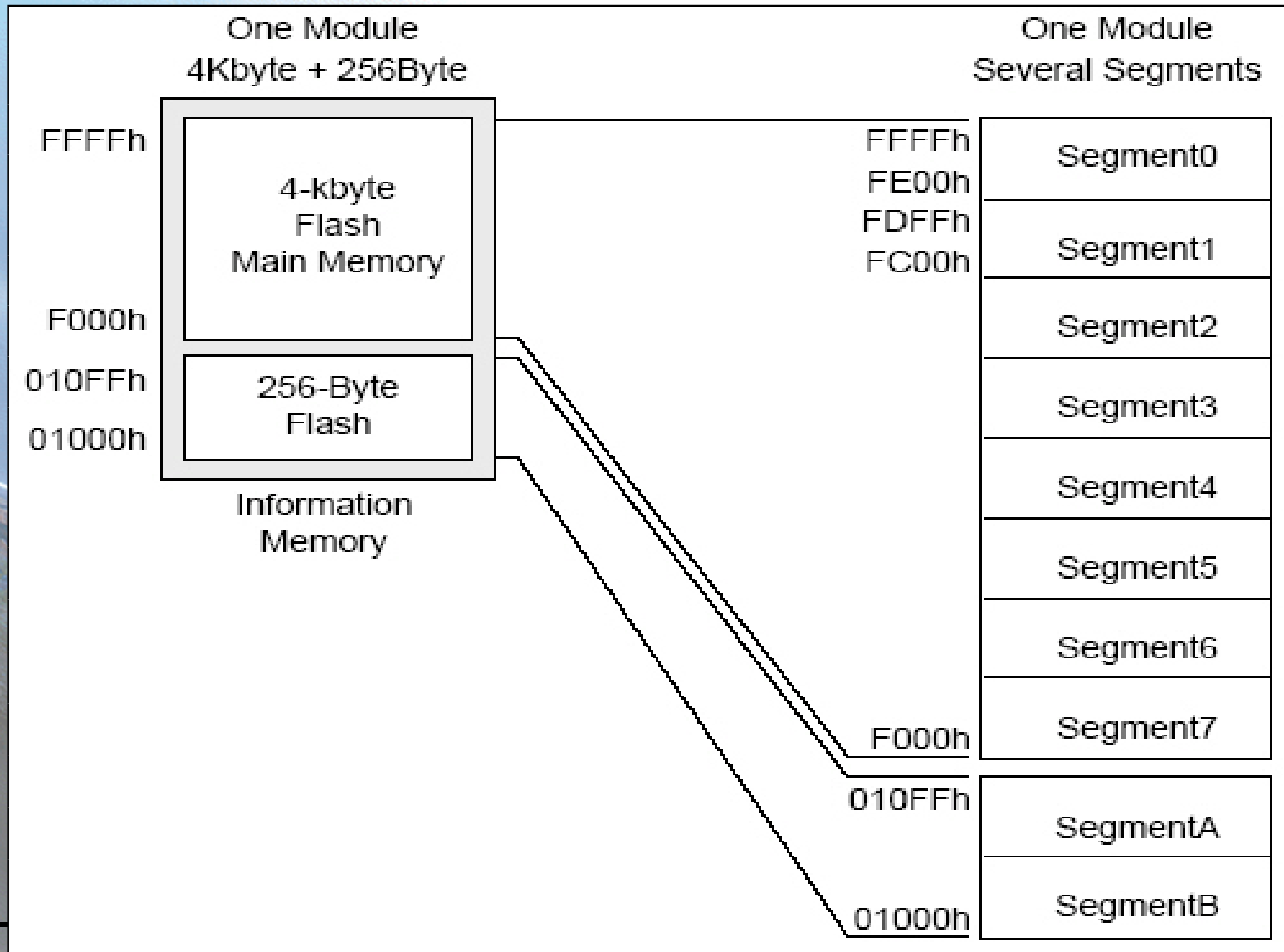
FLASH存储器模块 主要特点

- | 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
- | 数据保持时间从10年到100年不等
- | 可编程次数从100到100,000次
- | 60K空间编程时间<5秒
- | 保密熔丝烧断后不可恢复，不能再对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

FLASH存储器模块 主要特点

- | 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
 - | 数据保持时间从10年到100年不等
 - | 可编程次数从100到100,000次
 - | 60K空间编程时间<5秒
 - | 保密熔丝烧断后不可恢复，不能再对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
-

部分容量的FLASH段与地址的对应关系



Programming Flash Memory Devices

There are three options for programming an MSP430 flash device. All options support in-system programming:

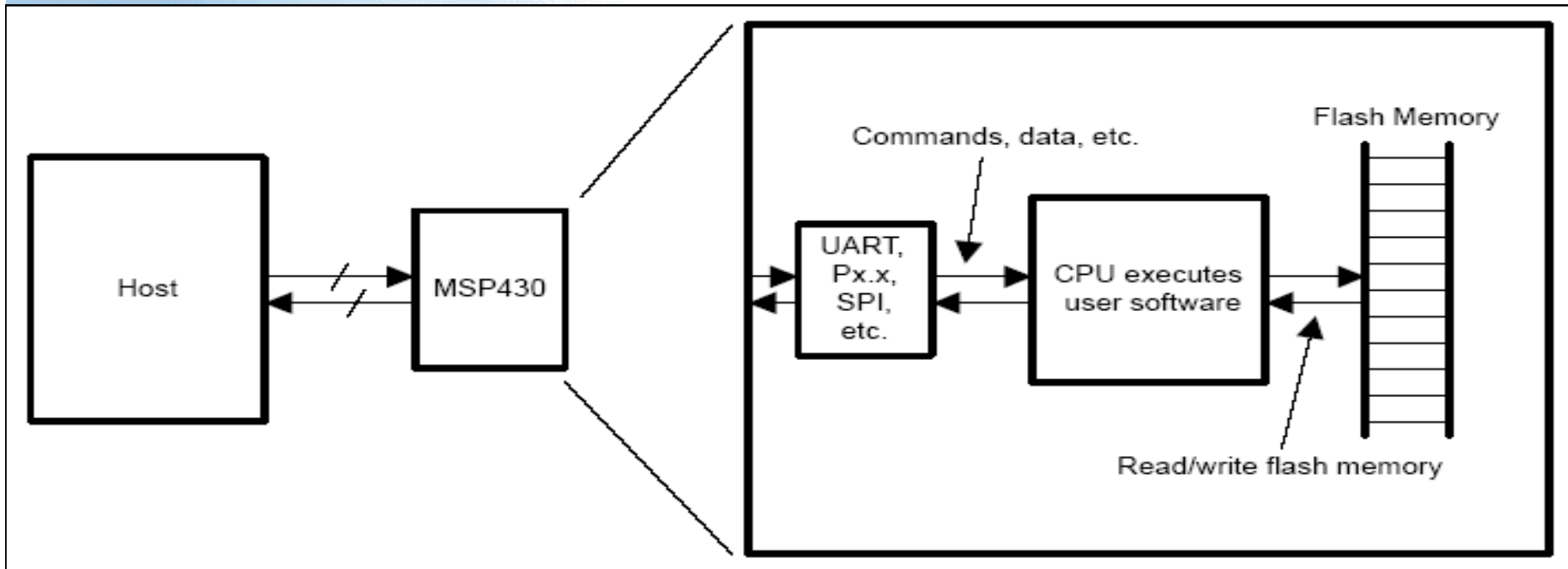
- Program via JTAG
- Program via the Bootstrap Loader
- Program via a custom solution

Programming Flash Memory via the Bootstrap loader (BSL)

Every MSP430 **flash** device contains a **bootstrap loader**.

The **BSL** enables users to read or program the flash memory or RAM using a **UART** serial interface. Access to the MSP430 flash memory via the BSL is protected by a 256-bit, user-defined password.

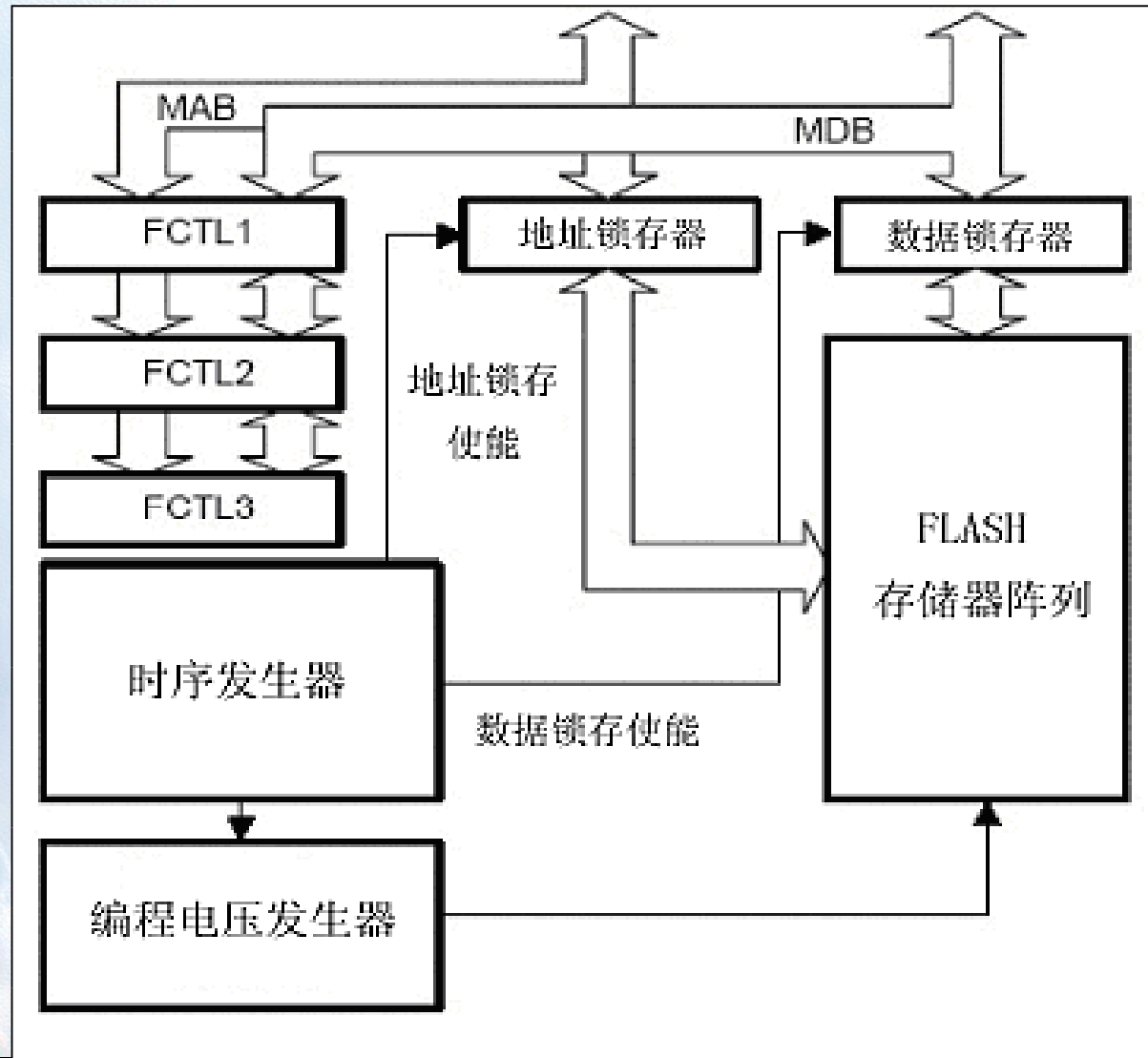
Programming Flash Memory via a Custom Solution



Programming Flash Memory via a Custom Solution

The ability of the MSP430 CPU to write to its own flash memory allows for in-system and external custom programming solutions as shown in Figure. The user can choose to provide data to the MSP430 through any means available (UART, SPI, etc.). User-developed software can receive the data and program the flash memory. Since this type of solution is developed by the user, it can be completely customized to fit the application needs for programming , erasing, or updating the flash memory.

FLASH存储器的结构框图



FLASH存储器的主要功能部件

- | 控制寄存器：控制FLASH存储器的擦除与写入
- | FLASH存储器阵列：存储体
- | 地址数据锁存器：擦除与编程时执行锁存操作
- | 编程电压发生器：产生编程电压
- | 时序发生器：产生擦除与编程所需所有时序控制信号

Erase Modes

MERAS	ERASE	Erase Mode
0	1	Segment erase
1	0	Mass erase (all main memory segments)
1	1	Erase all flash memory (main and information segments)

Write Modes

BLKWRT	WRT	Write Mode
0	1	Byte/word write
1	1	Block write

Erase Cycle from Within Flash Memory

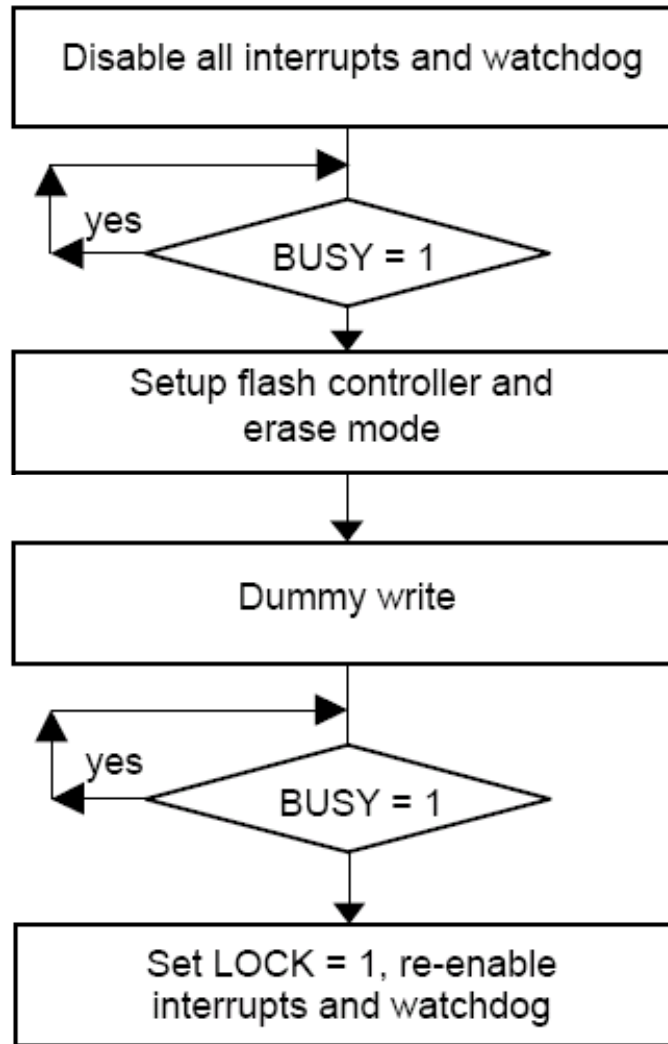
Disable all interrupts and watchdog

Setup flash controller and erase mode

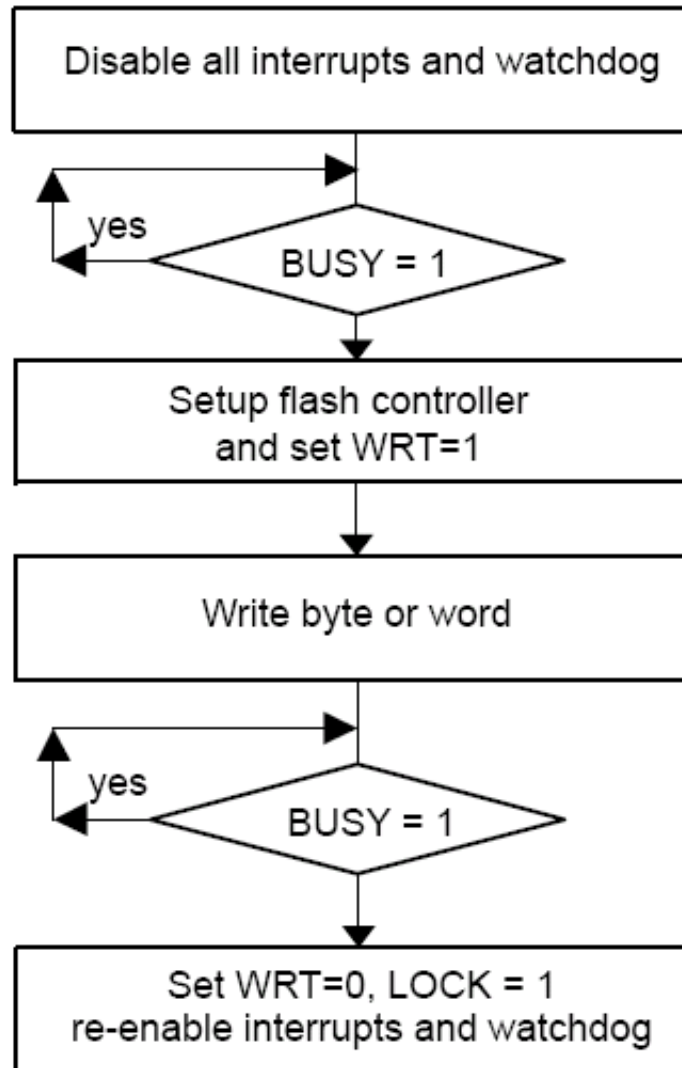
Dummy write

Set LOCK=1, re-enable Interrupts and watchdog

Erase Cycle from Within RAM



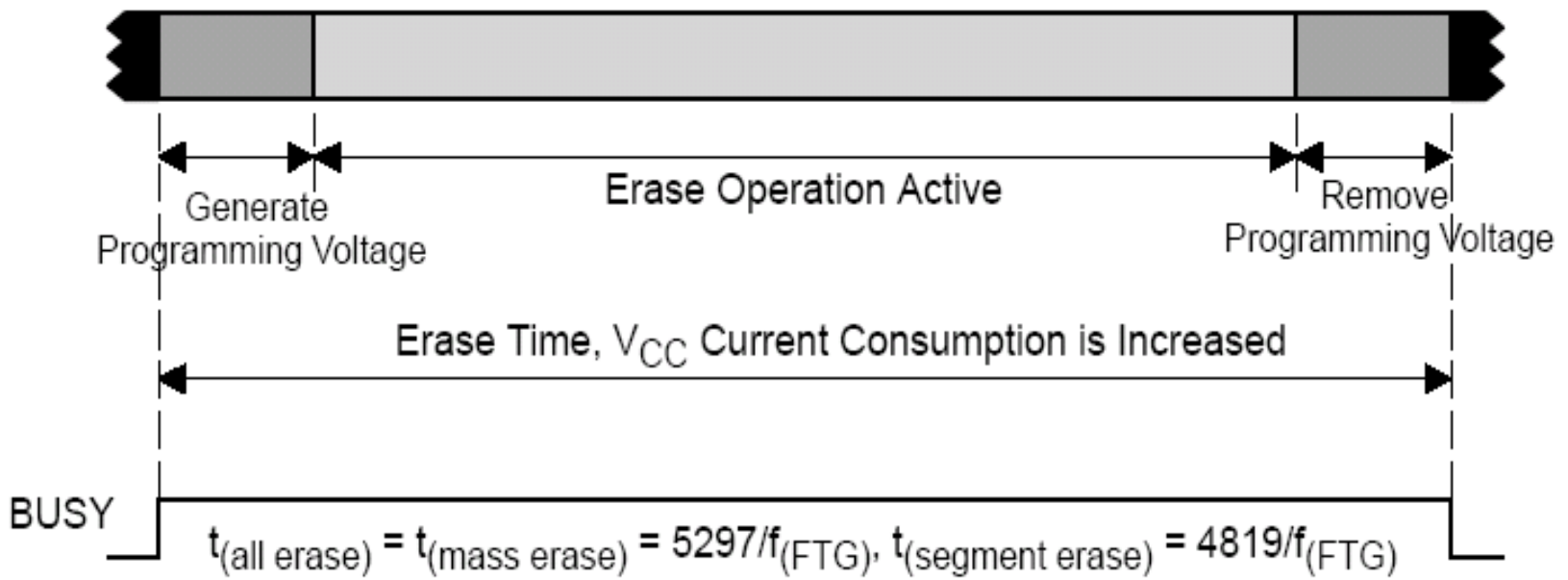
Initiating a Byte/Word Write from RAM



擦除操作

- | 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，则将它复位
- | 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。
- | 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全擦除，则设置RASE=1同时MERAS=1。
- | 对擦除的地址范围内任意位置作一次空写入，用以启动擦除操作。如果空写的地址在不能执行擦除操作的段地址范围内，则写入操作不起作用

Erase Cycle Timing

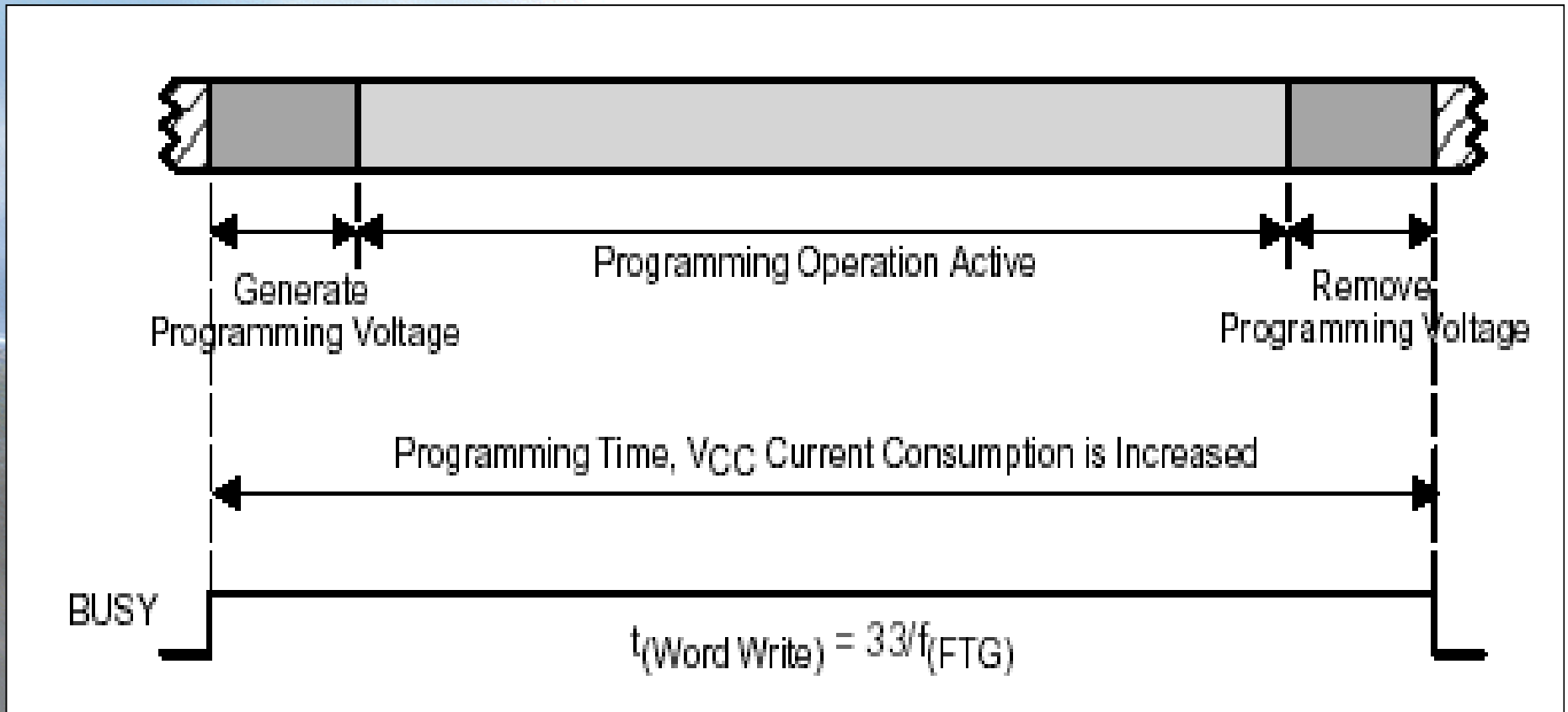


Any erase is initiated by a dummy write into the address range to be erased. The dummy write starts the flash timing generator and the erase operation. Figure shows the erase cycle timing. The BUSY bit is set immediately after the dummy write and remains set throughout the erase cycle. BUSY, MERAS, and ERASE are automatically cleared when the cycle completes. **The erase cycle timing is not dependent on the amount of flash memory present on a device. Erase cycle times are equivalent for all MSP430 devices.**

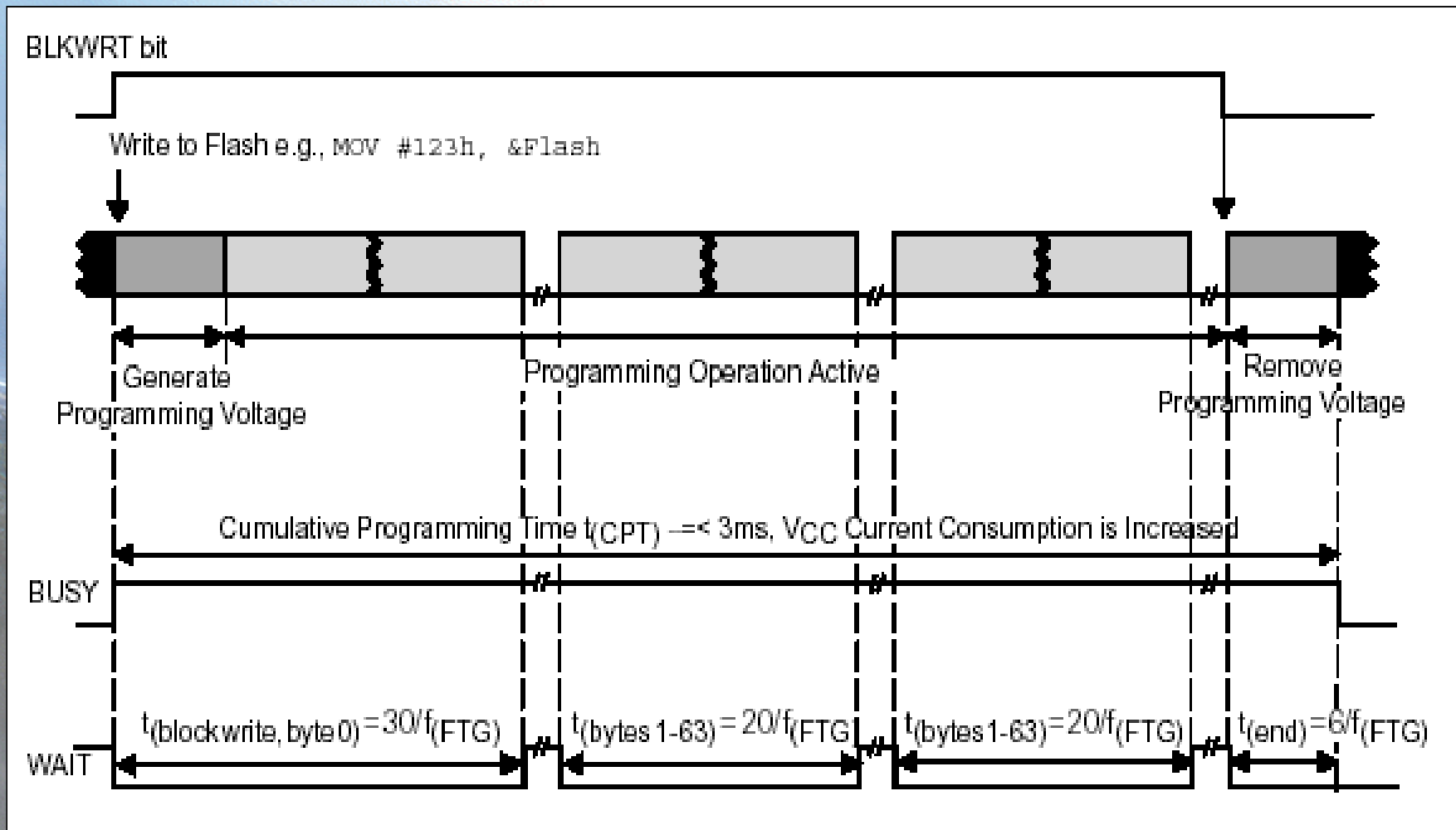
FLASH编程操作

- | 选择适当的时钟源以及合适的分频因子
 - | 如果Lock=1，将它复位
 - | 监视BUSY位，直到BUSY=0是才可进入下一步
 - | 如果写入单字或单字节，则将设置WRT=1
 - | 如果是块写或多字、多字节顺序写入，则将设置WRT=1，
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
-

单字或单字节写入周期



块写入周期



FLASH错误操作的处理

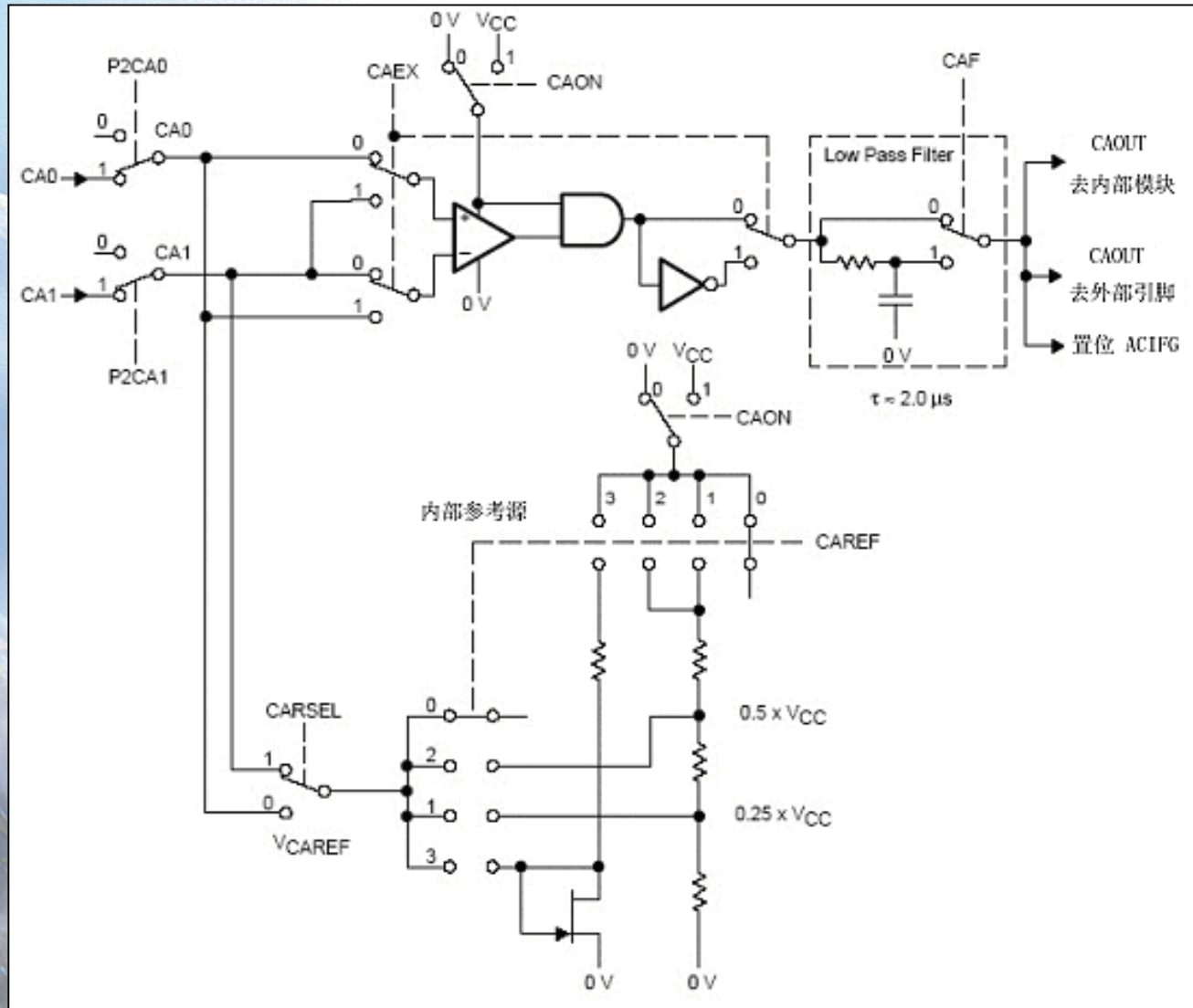
- 如果写入高字节口令码错误，则引发**PUC**信号：小心操作可以避免；
- 在对**FLASH**操作期间读**FLASH**内容，会引发**ACCVFIG**状态位的设置：小心操作可以避免
- 因为在对**FLASH**操作期间，需要较长的时间，如果这时看门狗定时器的数据接近尾声，则看门狗定时器溢出：建议用户程序在进行**FLASH**操作之前先停掉看门狗定时器，等操作结束之后再打开看门狗

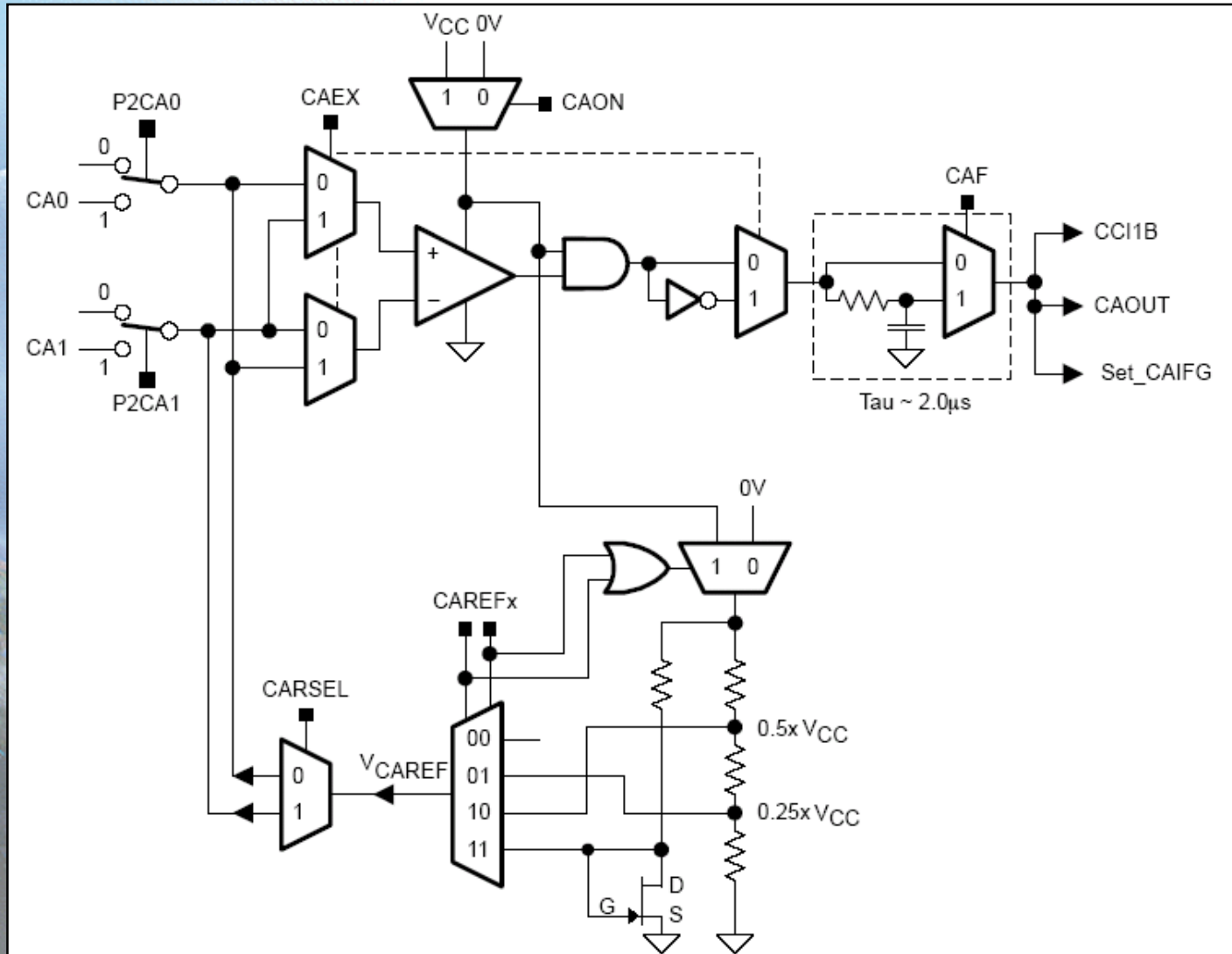
-
- 所有的FLASH类型的MSP430器件的0段都包含有中断向量等重要的程序代码，如果对其进行擦除操作，将会引起严重的后果：建议用户程序在进行FLASH操作之前，先将该段的重要数据（或程序代码）保存到RAM中或写入到其他暂时未用的段中，等待该段操作完毕再还原那些数据（或程序代码）；同时一定不要使正在执行的程序处在正要被擦除的段中；也不要FLASH操作期间允许中断的发生。
-

| Flash Memory Controller Interrupts

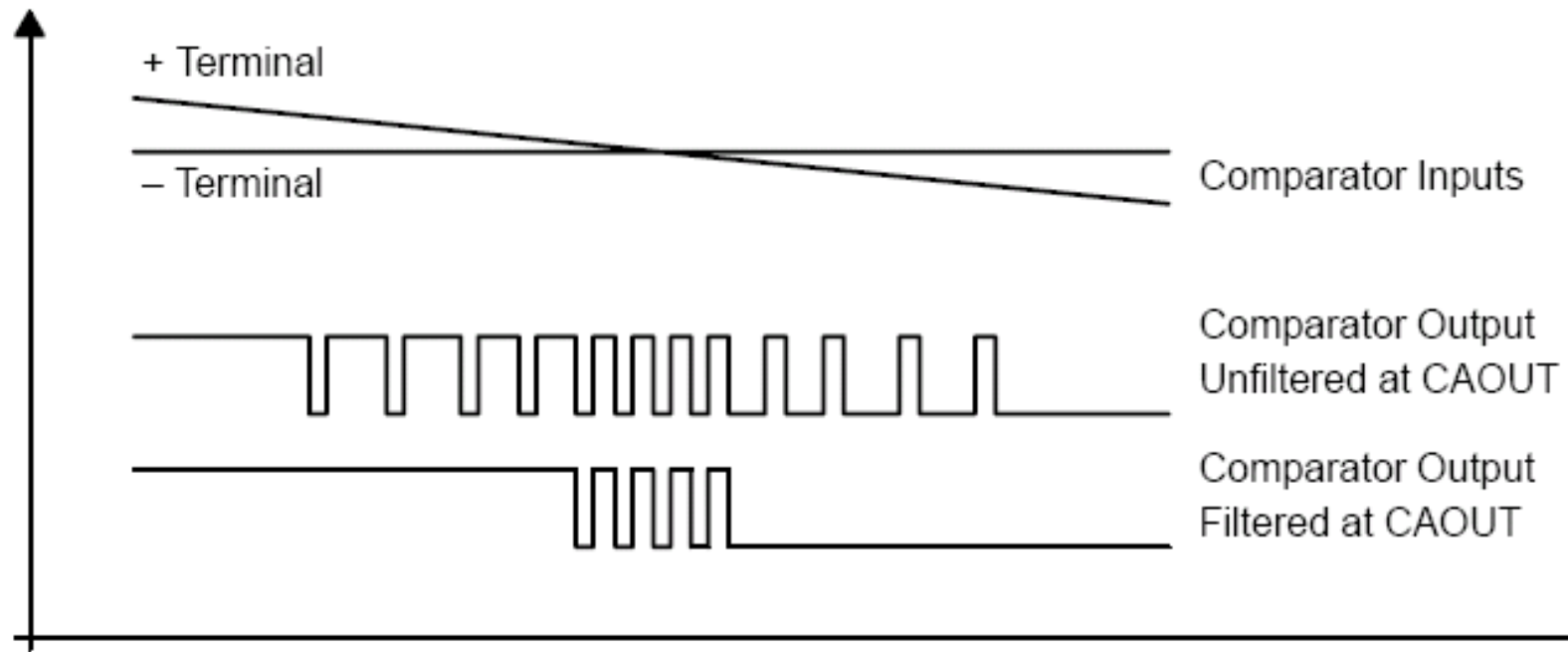
- | The flash controller has **two interrupt sources**, **KEYV**, and **ACCVIFG**. **ACCVIFG** is set when an access violation occurs. When the **ACCVIE** bit is re-enabled after a flash write or erase, a set **ACCVIFG** flag will generate an interrupt request. **ACCVIFG** sources the **NMI** interrupt vector, so it is **not** necessary for **GIE** to be set for **ACCVIFG** to request an interrupt. **ACCVIFG** may also be checked by software to determine if an access violation occurred. **ACCVIFG** must be reset by software.
- | The key violation flag **KEYV** is set when any of the flash control registers are written with an incorrect password. When this occurs, a **PUC** is generated immediately resetting the device.

比较器A的结构

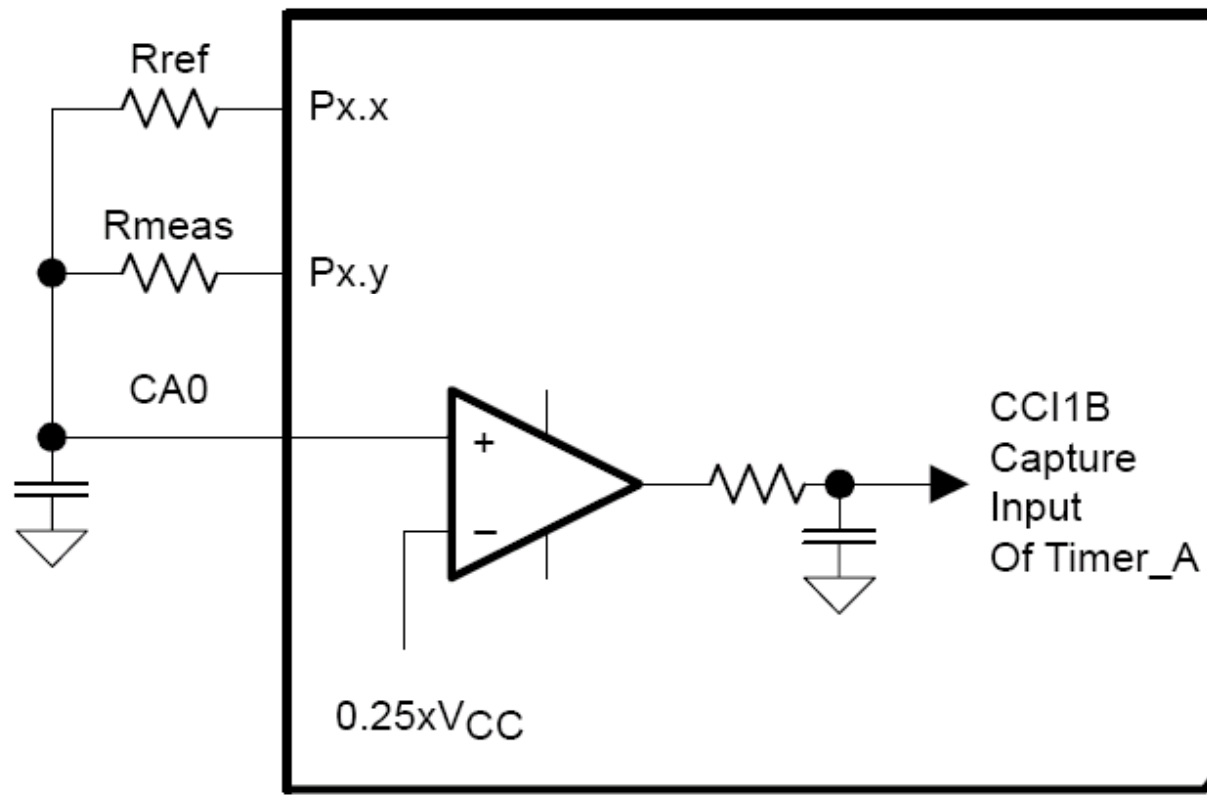




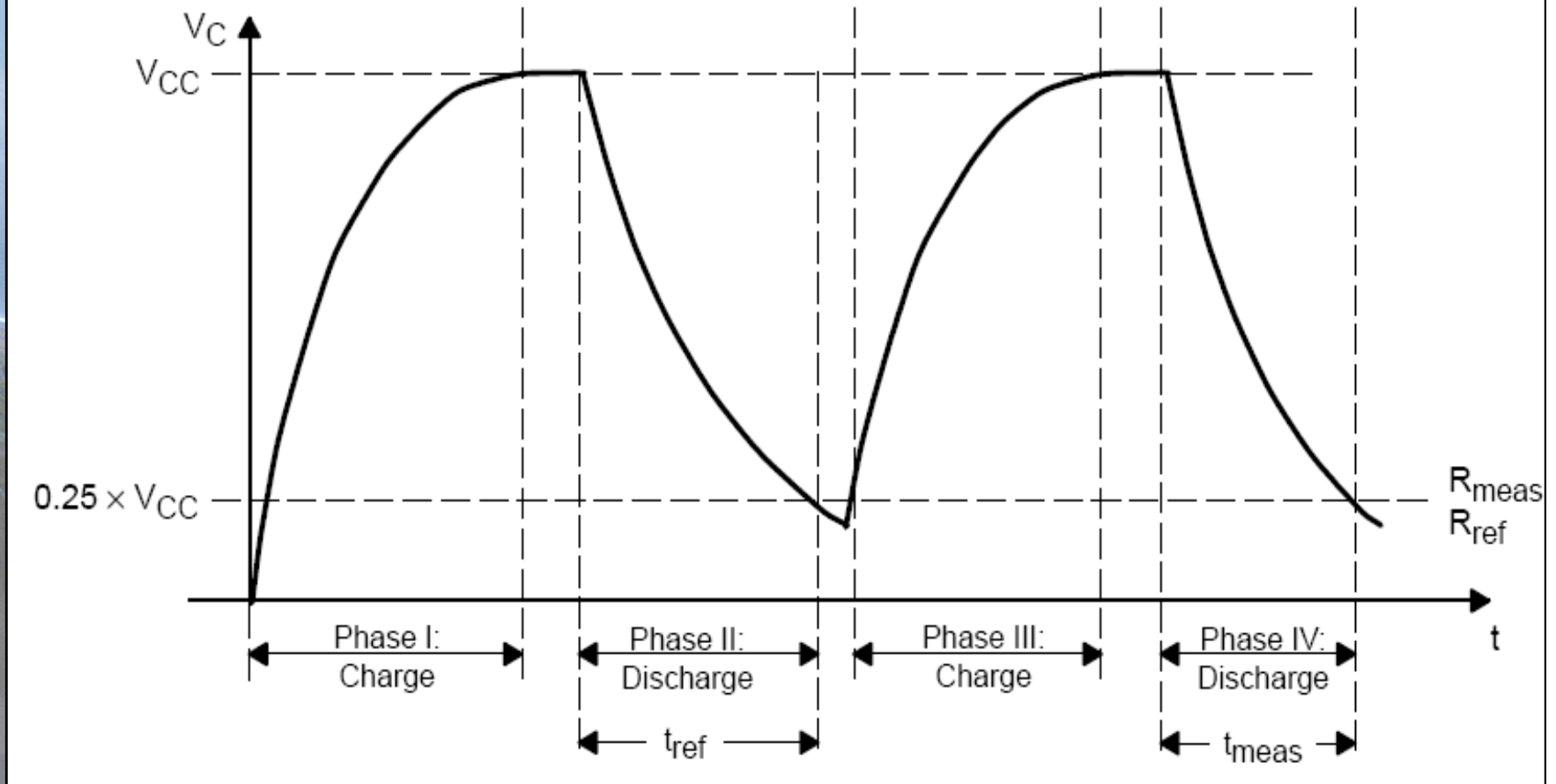
RC-Filter Response at the Output of the Comparator



Temperature Measurement System



Timing for Temperature Measurement Systems



$$\frac{N_{meas}}{N_{ref}} = \frac{-R_{meas} \times C \times \ln \frac{V_{ref}}{V_{CC}}}{-R_{ref} \times C \times \ln \frac{V_{ref}}{V_{CC}}}$$

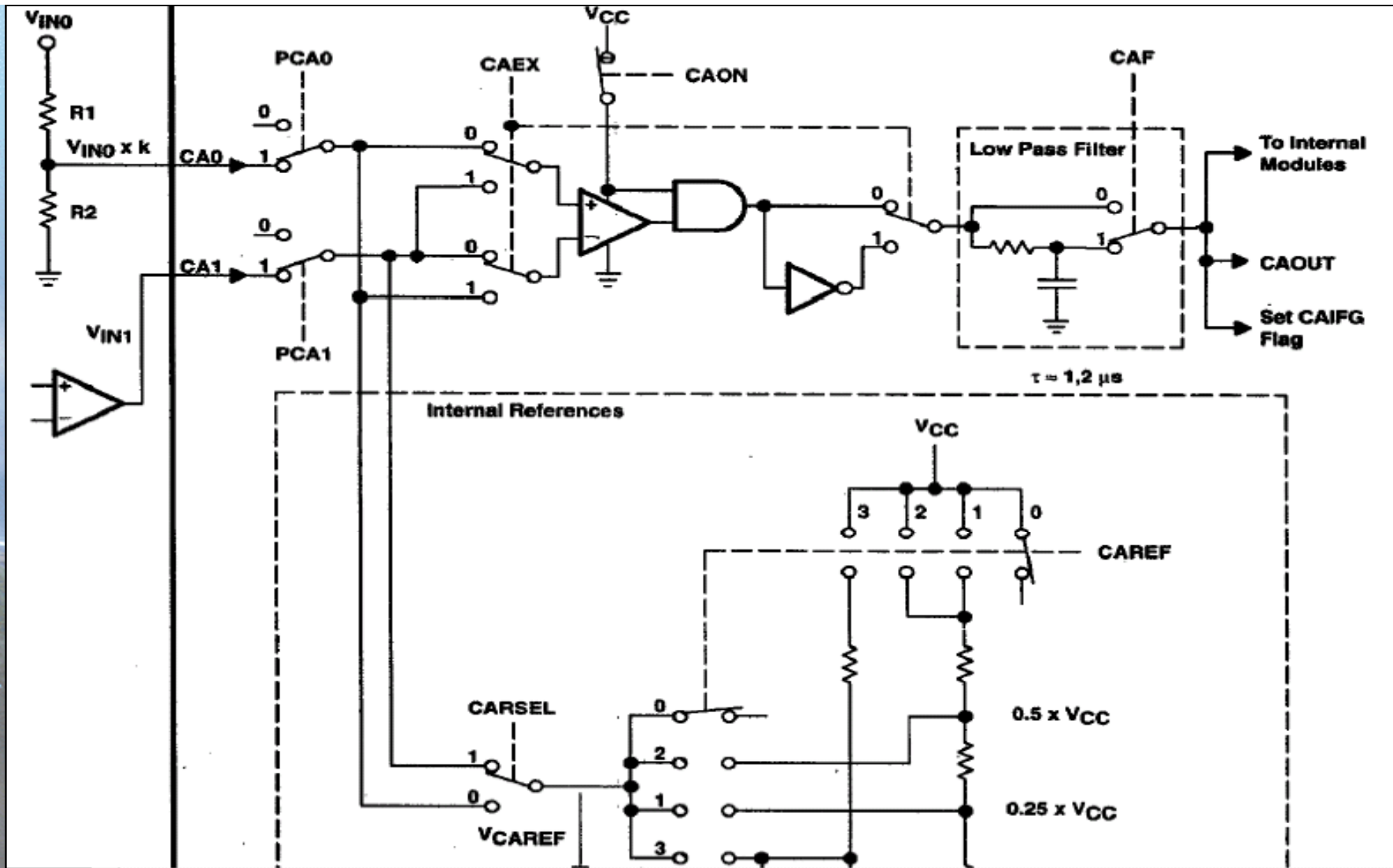
$$\frac{N_{meas}}{N_{ref}} = \frac{R_{meas}}{R_{ref}}$$

$$R_{meas} = R_{ref} \times \frac{N_{meas}}{N_{ref}}$$

比较器A的功能

- 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。
- 参与比较的两个电压CA0和CA1可以是外部或者内部基准电压。任何组合都是可能的。

两个外部输入比较
每个外部输入与 $0.5V_{CC}$ 或 $0.25V_{CC}$ 比较
每个外部输入与内部基准电压比较

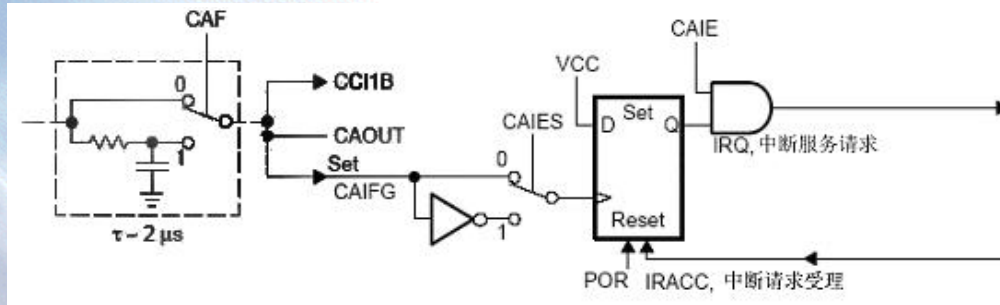


如果 $V_{IN0} \times (R2 / (R1 + R2)) > V_{IN1}$ CAOUT=1 否则 CAOUT=0

比较器A的中断

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中断允许（CAIE置位）、系统总中断允许（GIE置位）



中断响应后，因为比较器A具有独立中断向量，是单源中断，硬件会自动清除中断标志位CAIFG

比较器A应用举例

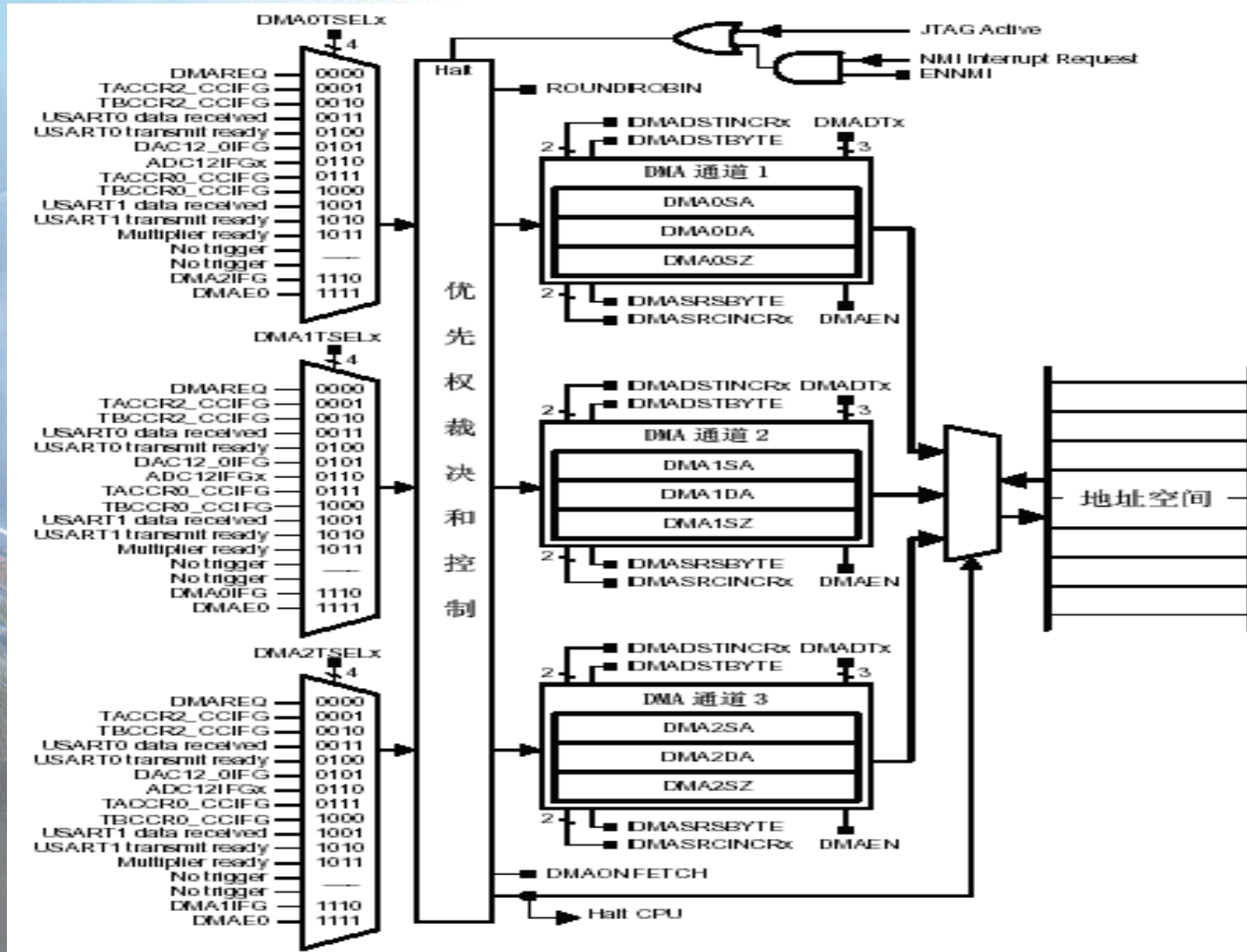
- 电压检测：P2.3输入的未知电压接到比较器A正端，片内参考电压 $0.25V_{cc}$ 接到比较器A负端，如果未知电压大于 $0.25V_{cc}$ ，P1.0置位，否则P1.0复位。

```
#include "msp430x11x1.h"
void main (void)
{
    WDTCTL = WDTPW + WDTHOLD; // 停止看门狗
    P1DIR |= 0x01;           // P1.0 输出
    CACTL1 = CARSEL + CAREF0 + CAON; // 0.25 Vcc = -comp
    CACTL2 = P2CA0;         // P2.3 = +comp
    while (1)
    {
        if ((CAOUT & CACTL2))
            P1OUT |= 0x01; // CAOUT = 1, 置位 P1.0
        else P1OUT &= ~0x01; // 否则复位
    }
}
```


MSP430 DMA控制器的特性

- | 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65536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或输出时，CPU也可以处于超低功耗模式而不需唤醒
- | 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节。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
- | 四种传输寻址模式：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

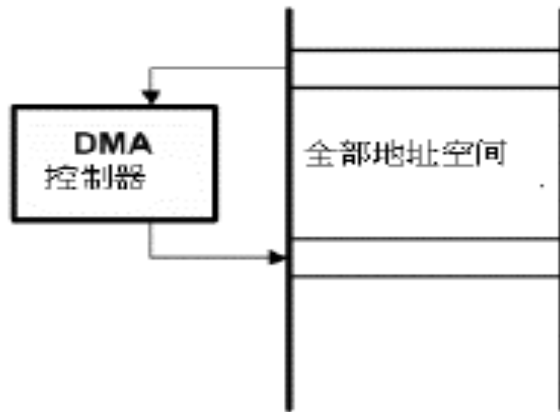
MSP430 DMA控制器的结构



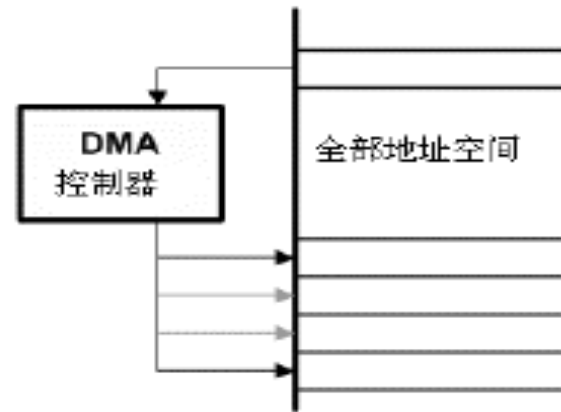
DMA控制器的功能模块：

- 3个独立的传输通道：通道0、通道1和通道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，确定哪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）、Timer_A CCR2输出、Timer_B CCR2输出、I²C 数据接收准备好、I²C 数据发送准备好、USART接收发送数据、DAC12模块DAC12IFG、ADC12模块的ADC12IFGx、DMAxIFG、DMAE0 外部触发源。并且还具有触发源扩充能力
-

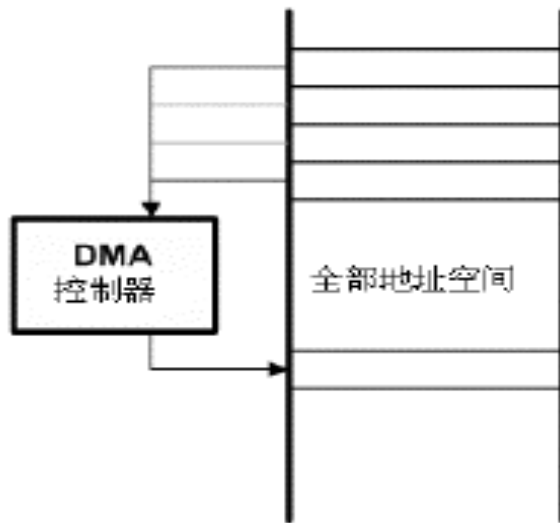
DMA控制器的寻址



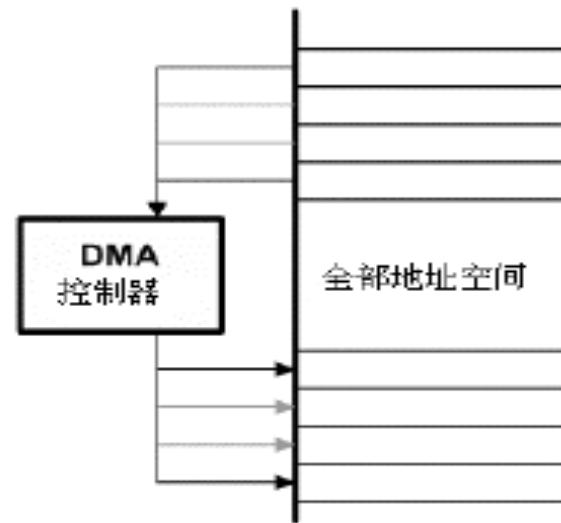
固定的地址到固定的地址



固定的地址到块地址



块地址到固定地址



块地址到块地址

DMA传输模式

- | 单字或者单字节传输
- | 块传输
- | 突发块传输
- | 重复单字或者单字节传输
- | 重复块传输
- | 重复突发块传输

DMA应用举例

- 利用DMA控制器将数据块由RAM的220h-240h单元传输到240h-260h单元

```
#include <msp430x16x.h>
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // 停看门狗
    P1DIR |= 0x01;           // P1.0输出
    DMA0SA = 0x0220;         // 起始地址
    DMA0DA = 0x0240;         // 目的地址
    DMA0SZ = 0x010;         // 传输规模
    DMA0CTL = DMADT_5 + DMASRCINCR_3 + DMADSTINCR_3 + DMAEN; // 重
    复块传输, 起始地址、目的地址增量
    for (;;)                 // 重复块传输
    {
        P1OUT |= 0x01;       // 置位P1.0
        DMA0CTL |= DMAREQ;   // 触发块传输
        P1OUT &= ~0x01;     // 清除P1.0
    }
}
```

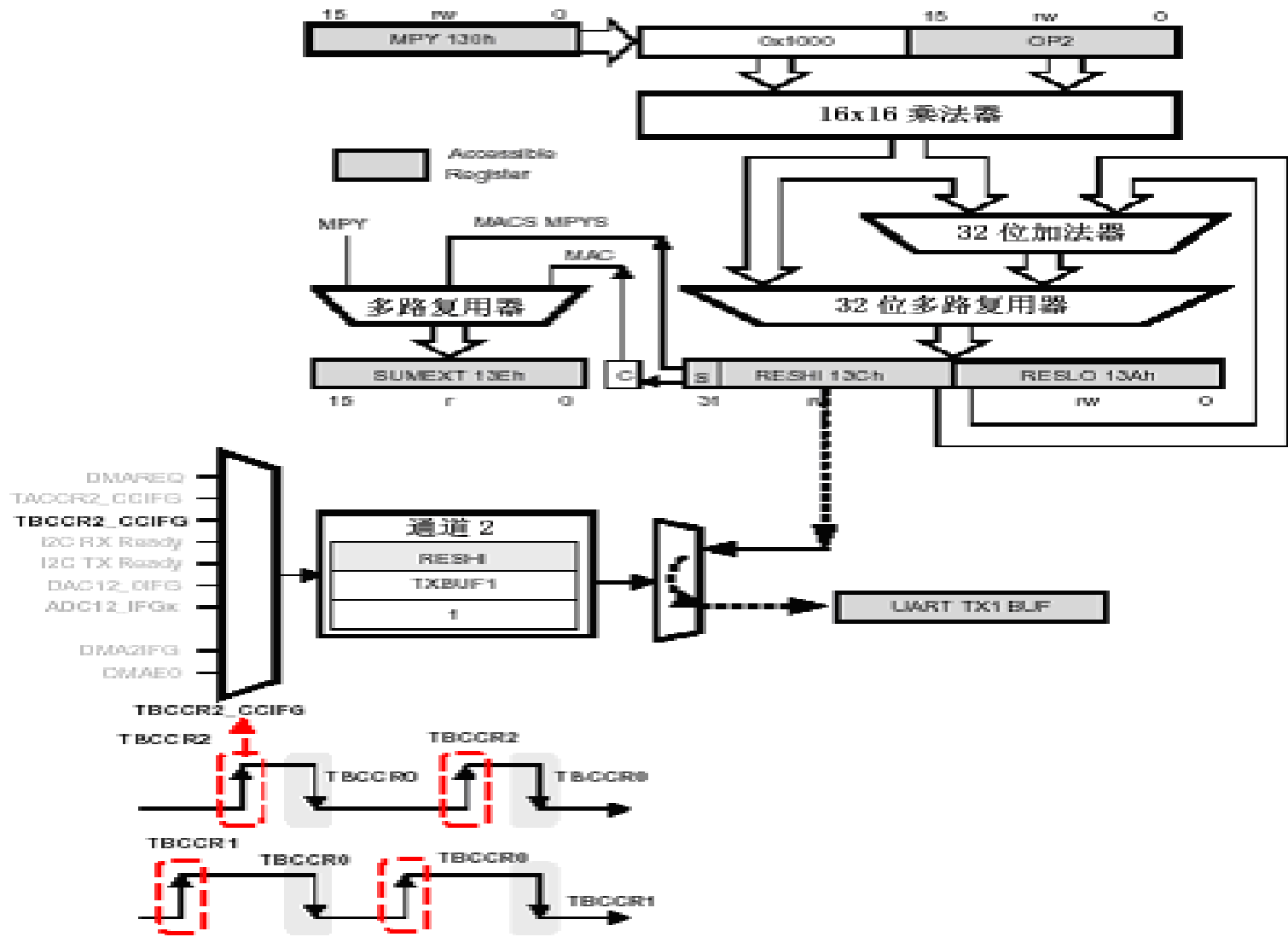
DMA应用举例

I 通过TACCR2触发DMA控制器给端口P1输出一个字节串

```
#include <msp430x16x.h>
const unsigned char testconst[] = { 0x00, 0x03, 0x02, 0x03, 0x00, 0x01 };
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;
    P1DIR |= 0x03;      // P1.0/1.1 输出
    DMACTL0 = DMA0TSEL_1; // CCR2IFG 触发
    DMA0SA = (unsigned int)testconst; // 起始地址
    DMA0DA = P1OUT_;    // 目的地址
    DMA0SZ = sizeof testconst; // 传输规模
    DMA0CTL = DMADT_4 + DMASRCINCR_3 + DMASBDB + DMAEN; // 重复单字节传输
    , 起始地址增量, DMA使能
    TACTL = TASSEL_2 + MC_2; // SMCLK, 连续计数模式
    _BIS_SR(LPM0_bits);    //进入 LPM0
}
```

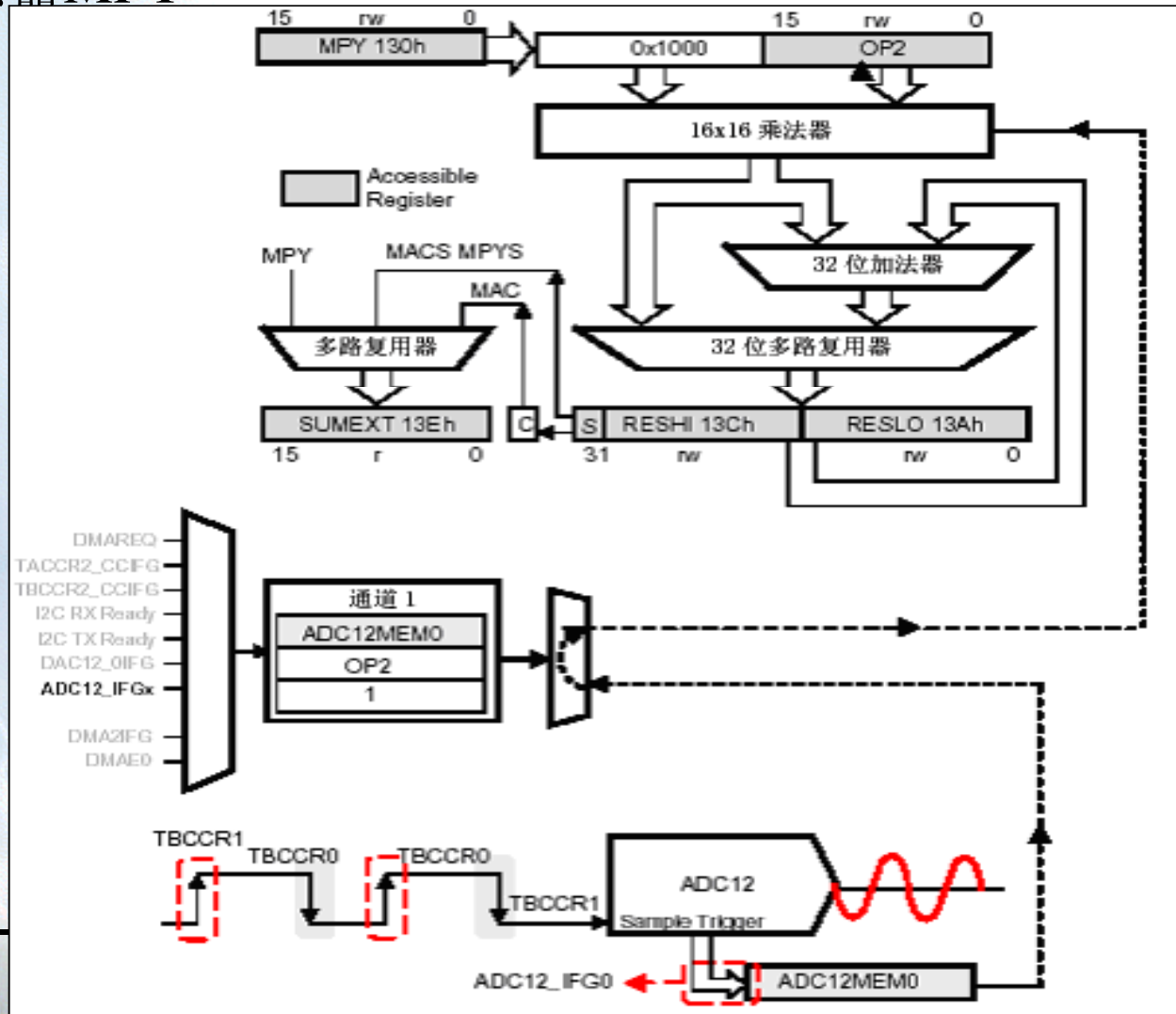
DMA应用举例

DMA传输使硬件乘法器的运算结果通过串口输出。

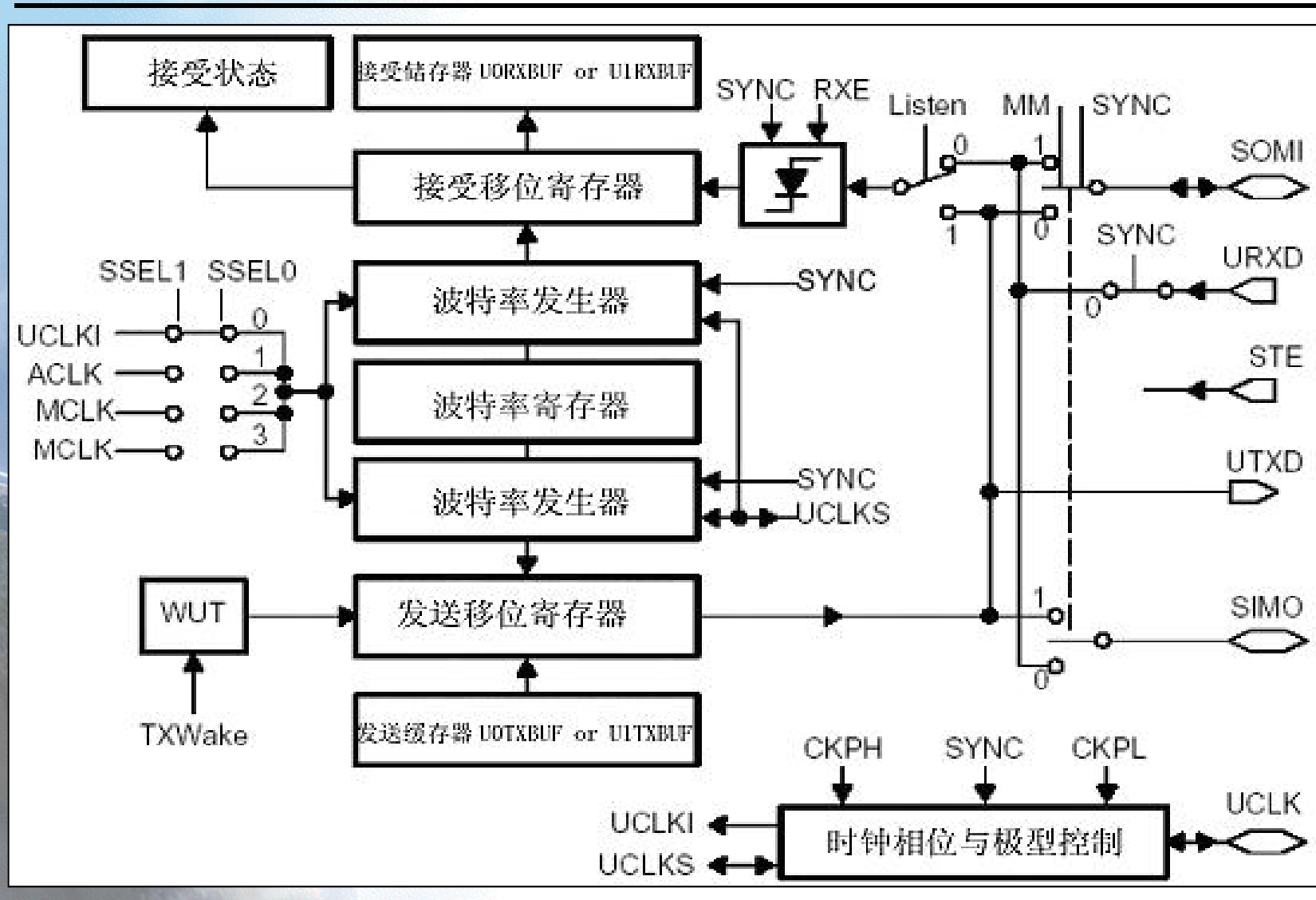


DMA应用举例

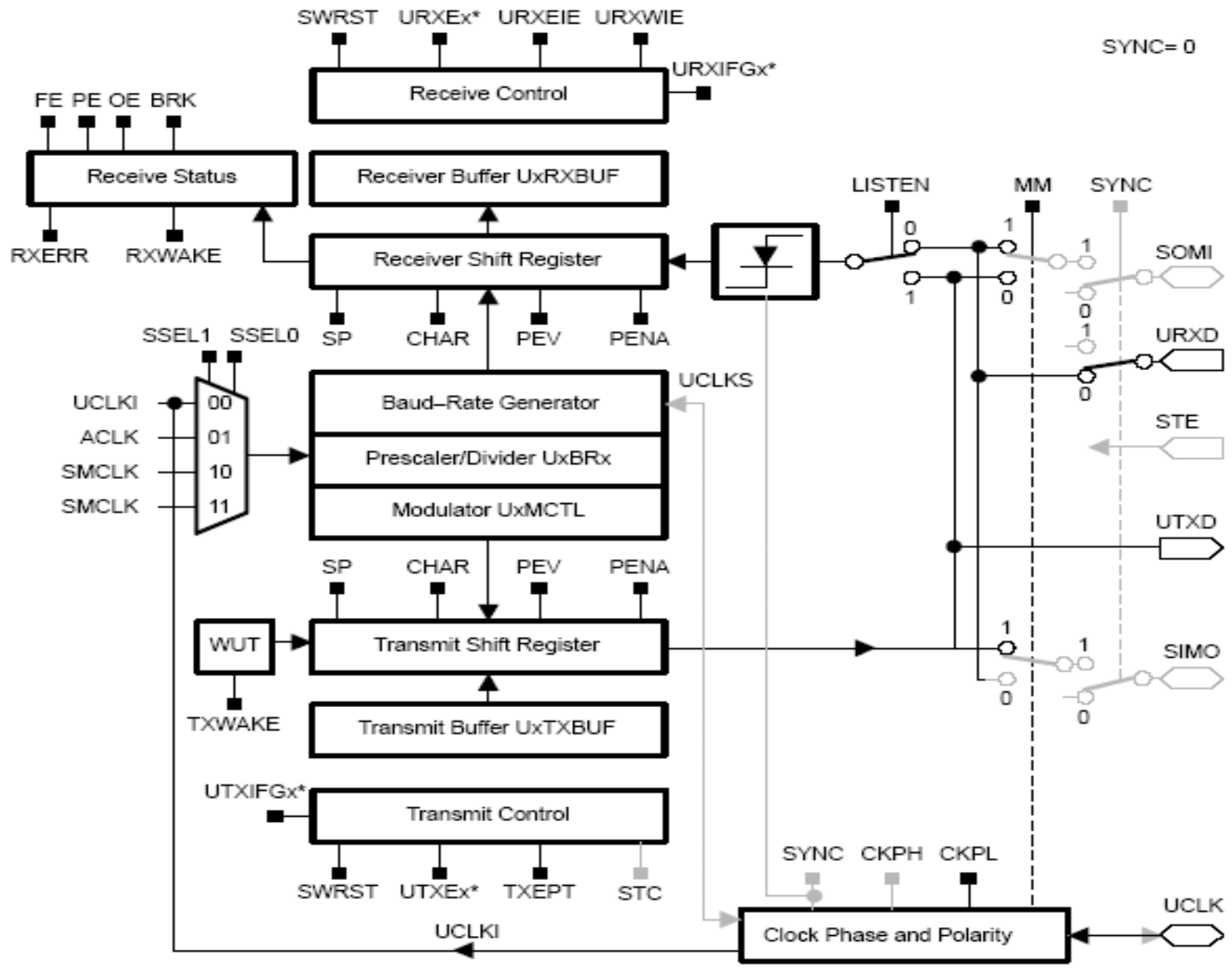
- ADC12转换的结果通过 DMA 控制器传送至高速的运算部件硬件乘法器MPY



USART模块结构



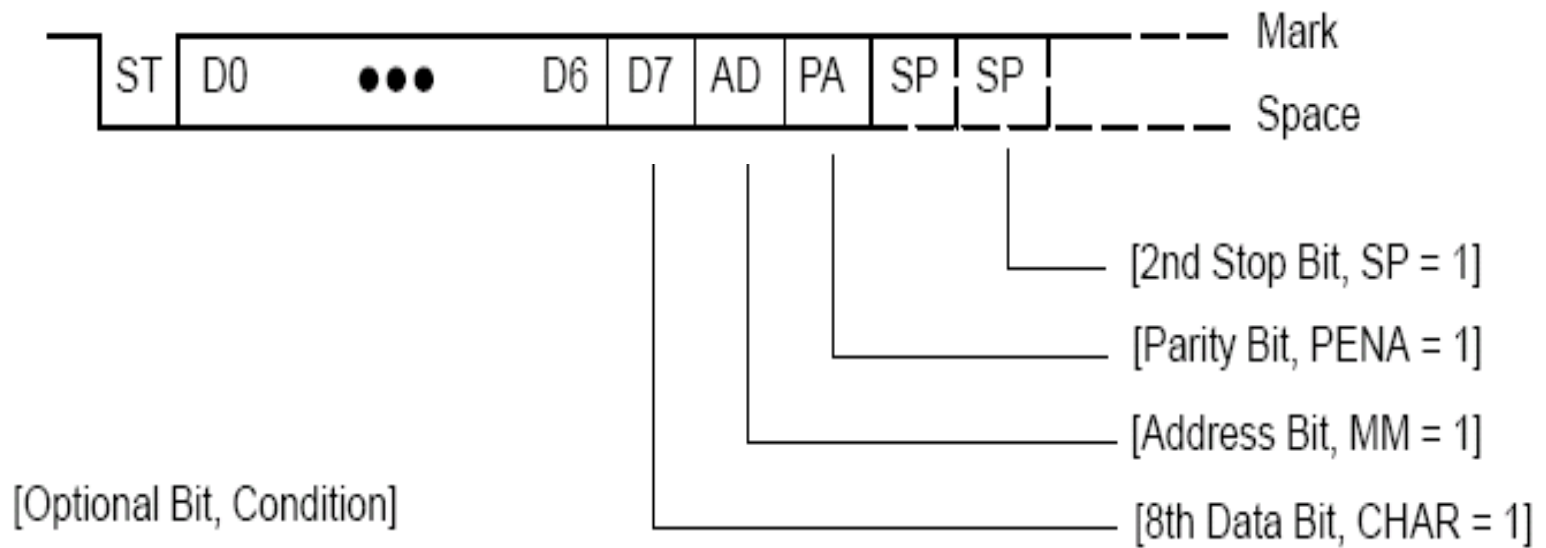
UART 模块结构 异步



串行异步通信特点

- | 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

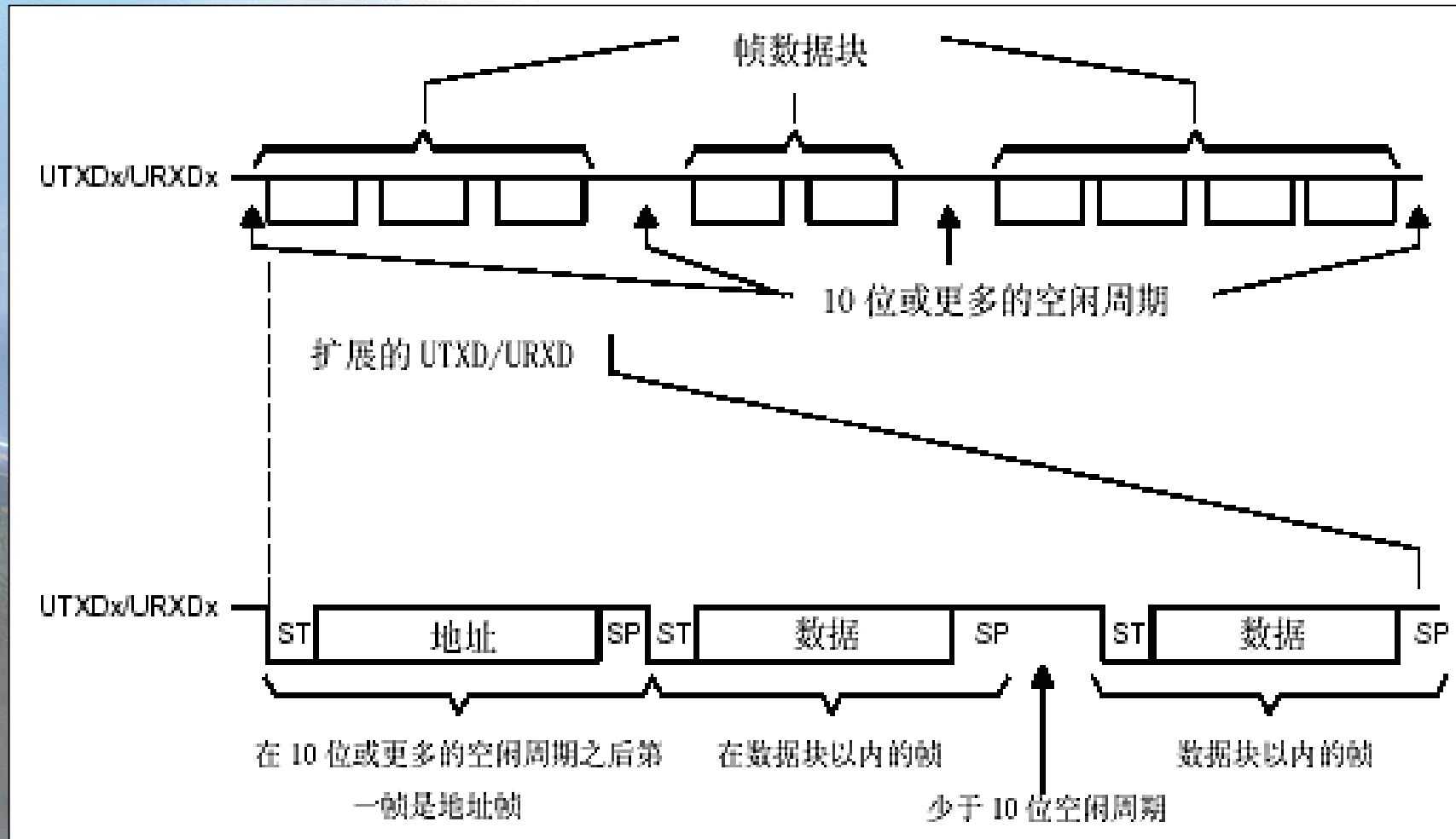
Character Format



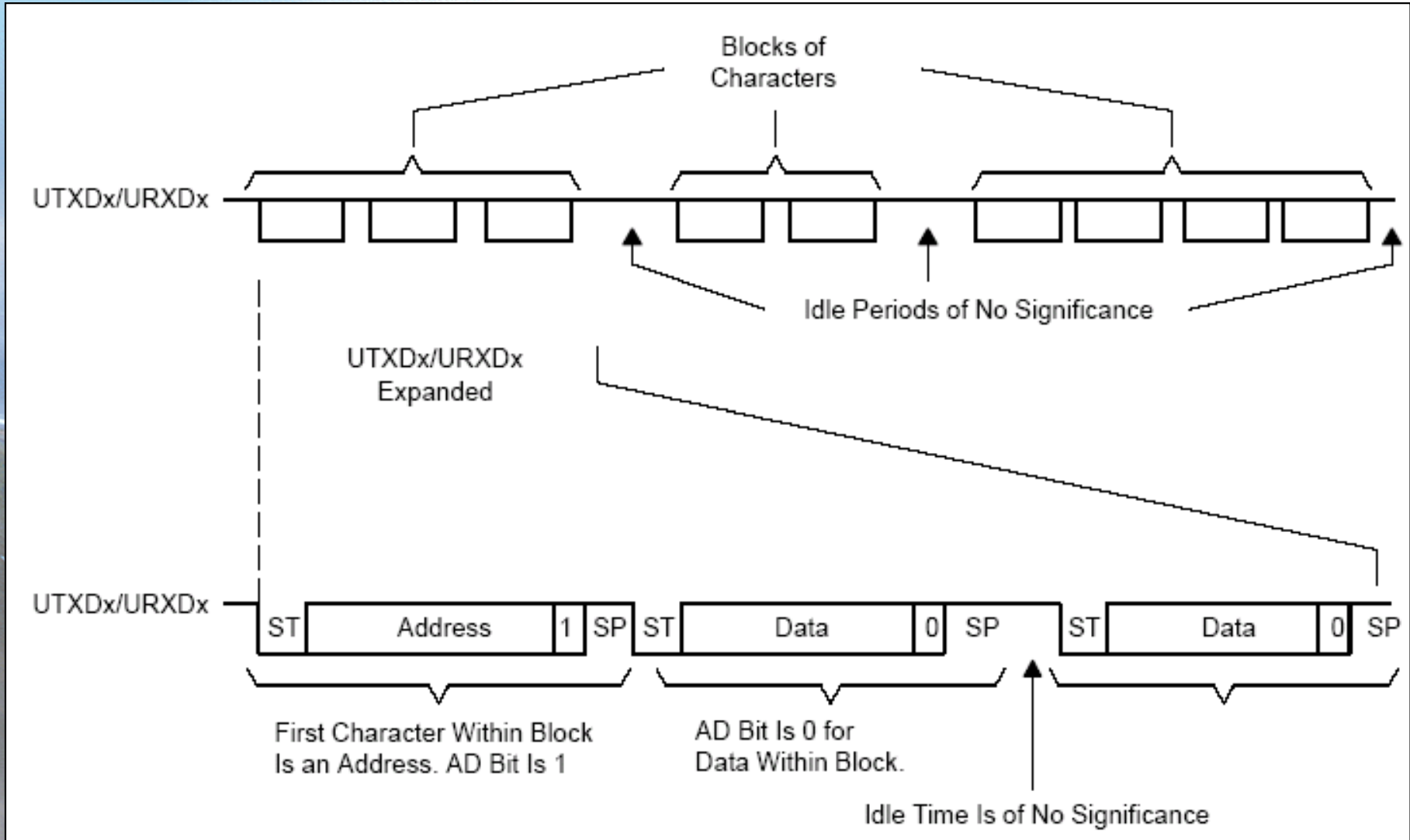
异步多机通信模式

- | When two devices communicate asynchronously, **the idle-line format** is used for the protocol.
- | When three or more devices communicate, the USART supports **the idle-line and address-bit multiprocessor** communication formats.

线路空闲多机模式



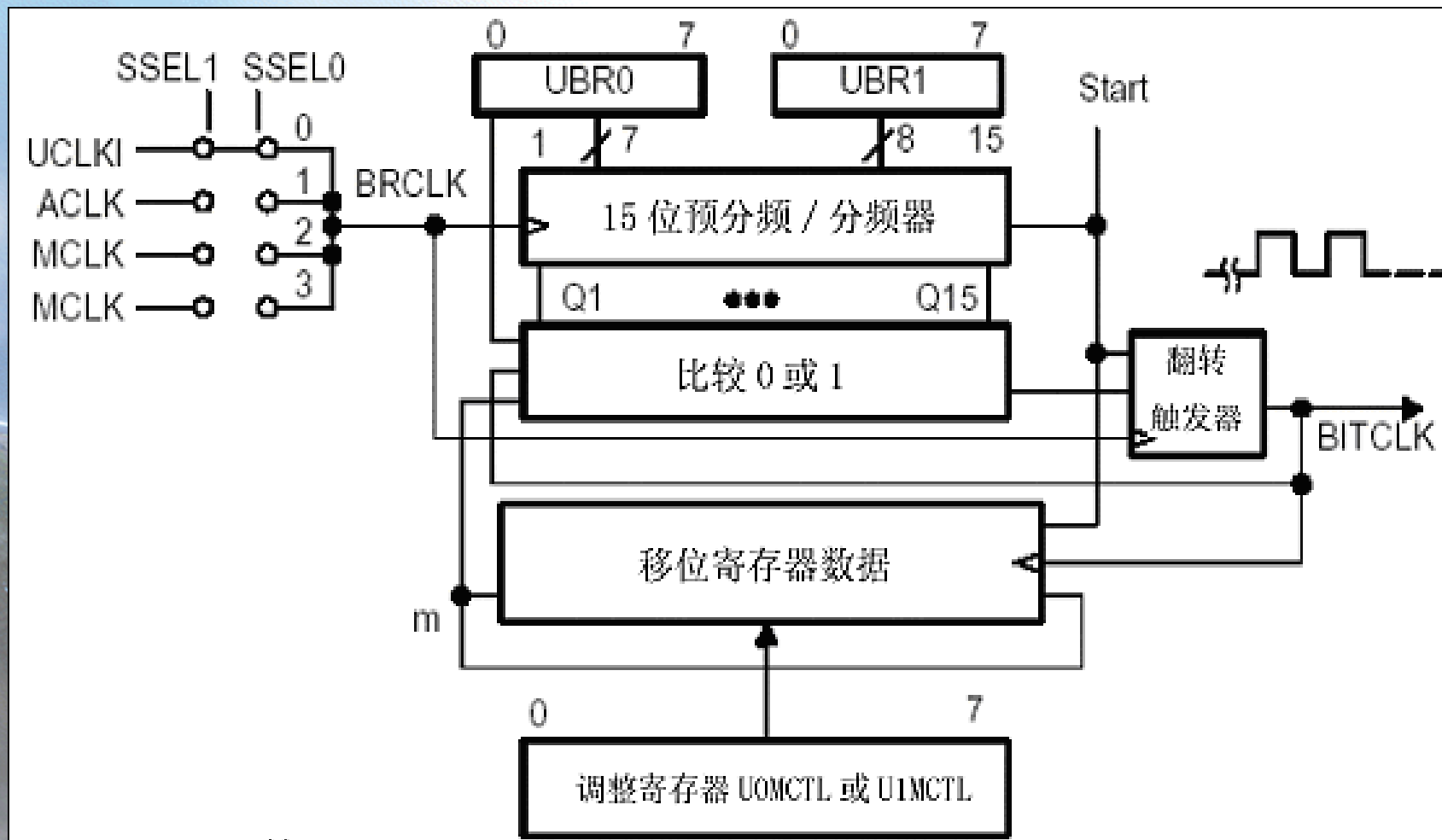
地址位多机模式



串行操作自动错误检测

- **FE 标志帧错误：** 当一个接收字符的停止位为0并被装入接收缓存，接收的为一个错误的帧，那么帧错标志被设置成1，即使在多停止位模式时也只检测第一个停止位。同样，丢失停止位意味着从起始位开始的同步特性被丧失，也是一个错误帧。在同步的4线模式时，因总线冲突使有效主机停止，并在STE引脚信号出现下降沿时使FE位设置为1
- **PE 奇偶校验错误：** 当接收字符中1的个数与它的校验位不相符，并被装入接收缓存时，发生校验错，设置PE为1
- **OE 溢出错误标志：** 当一个字符写入接收缓存URXBUF时，前一个字符还没有被读出，这时前一个字符因被覆盖而丢失，发生溢出（同步与异步情况相同）
- **BRK 打断检测标志：** 当发生一次打断同时URXEIE置位时，该位被设置为1，表示接收过程被打断过。RXD线路从丢失的第一个停止位开始连续出现至少10位低电平被识别为打断

波特率的产生



| Note: Initializing or Re-Configuring the USART Module

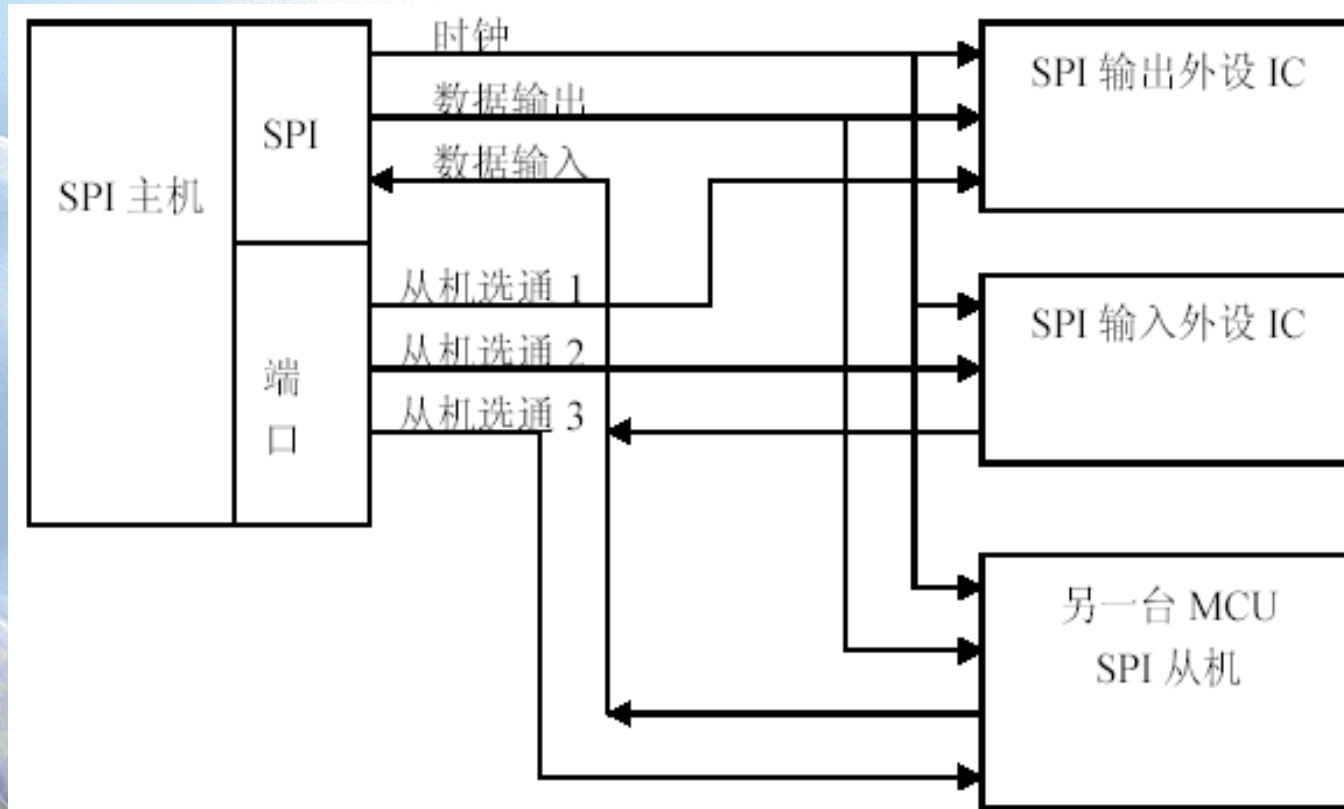
The required USART initialization/re-configuration process is:


- 1) Set SWRST (BIS.B #SWRST, &UxCTL)
- 2) Initialize all USART registers with SWRST = 1 (including UxCTL)
- 3) Enable USART module via the MEx SFRs (URXEx and/or UTXEx)
- 4) Clear SWRST via software (BIC.B #SWRST, & UxCTL)
- 5) Enable interrupts (optional) via the IEx SFRs (URXIEx and/or UTXIEx)

Failure to follow this process may result in unpredictable USART behavior.

```
UCTL0|=SWRST;
UCTL0|=CHAR;           // 8-bit 字符
UTCTL0=SSEL0+SSEL1;   // UCLK=SMCLK
UBR00=0xa0;           // 在4MHz下进行 9600波特率通信
UBR10=0x01;           // 4000000/9600=416.67
UMCTL0=0x5e;           // 调整寄存器(0.67*8=5.4)
UCTL0&=~SWRST;
ME1|= (UTXE0 + URXE0); // 使能 USART0 TXD/RXD
IE1|= URXIE0 ;
IFG1 = 0x00;
P2SEL |= 0x30;         // P2.4,P2.5 = USART0 TXD/RXD
P2DIR |= 0x10;
```

SPI典型结构





当MSP430 USART模块控制寄存器UCTL的SYNC位置位且I2C位复位时，串行模块工作在SPI模式，通过4线（SOMI，SIMO，UCLK及STE）或者3线（SOMI，SIMO，UCLK）同外界通信

-
- SIMO** Slave in, master out
Master mode: SIMO is the data output line.
Slave mode: SIMO is the data input line.

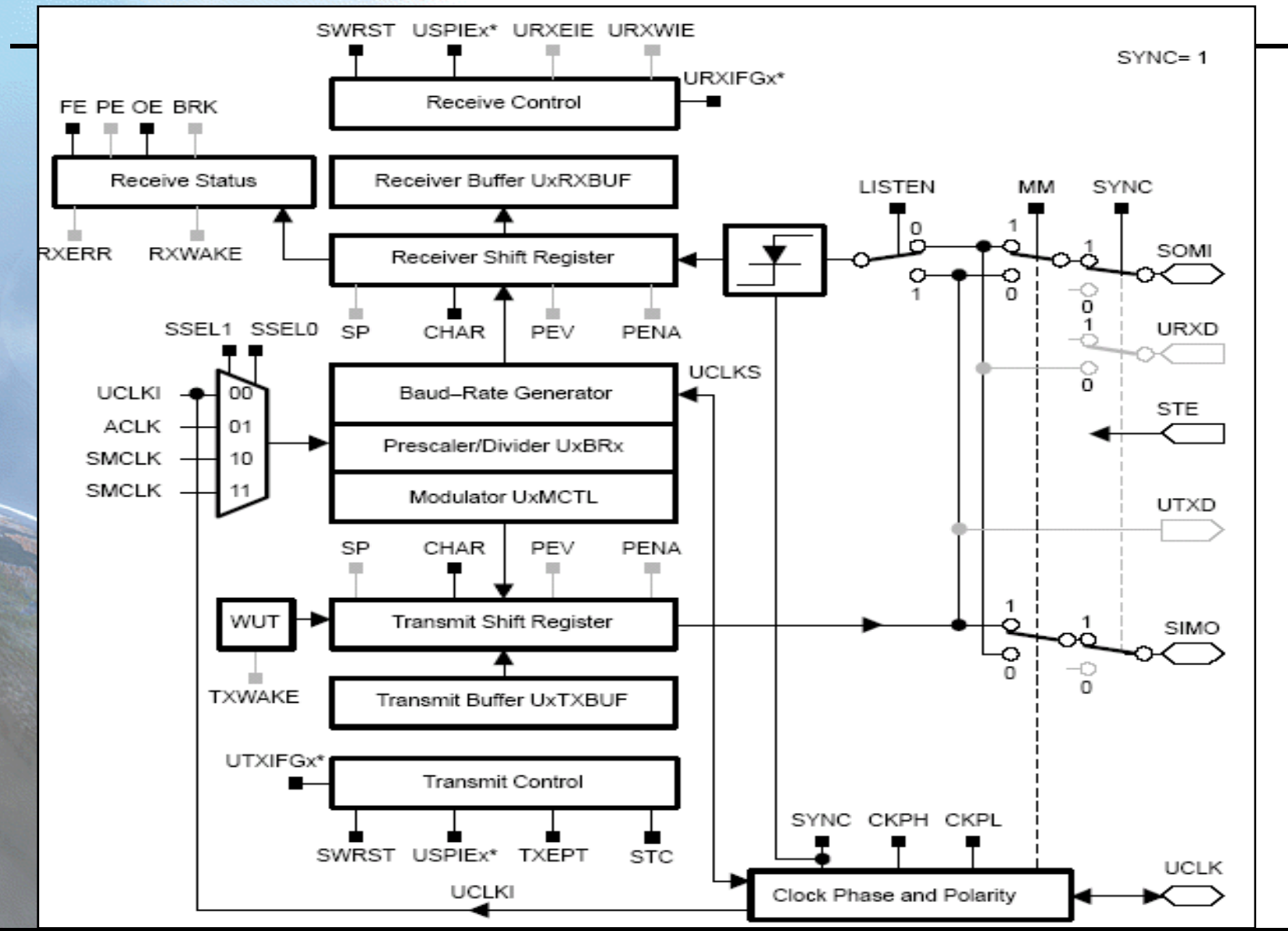
 - SOMI** Slave out, master in
Master mode: SOMI is the data input line.
Slave mode: SOMI is the data output line.

 - UCLK** USART SPI clock
Master mode: UCLK is an output.
Slave mode: UCLK is an input.
-

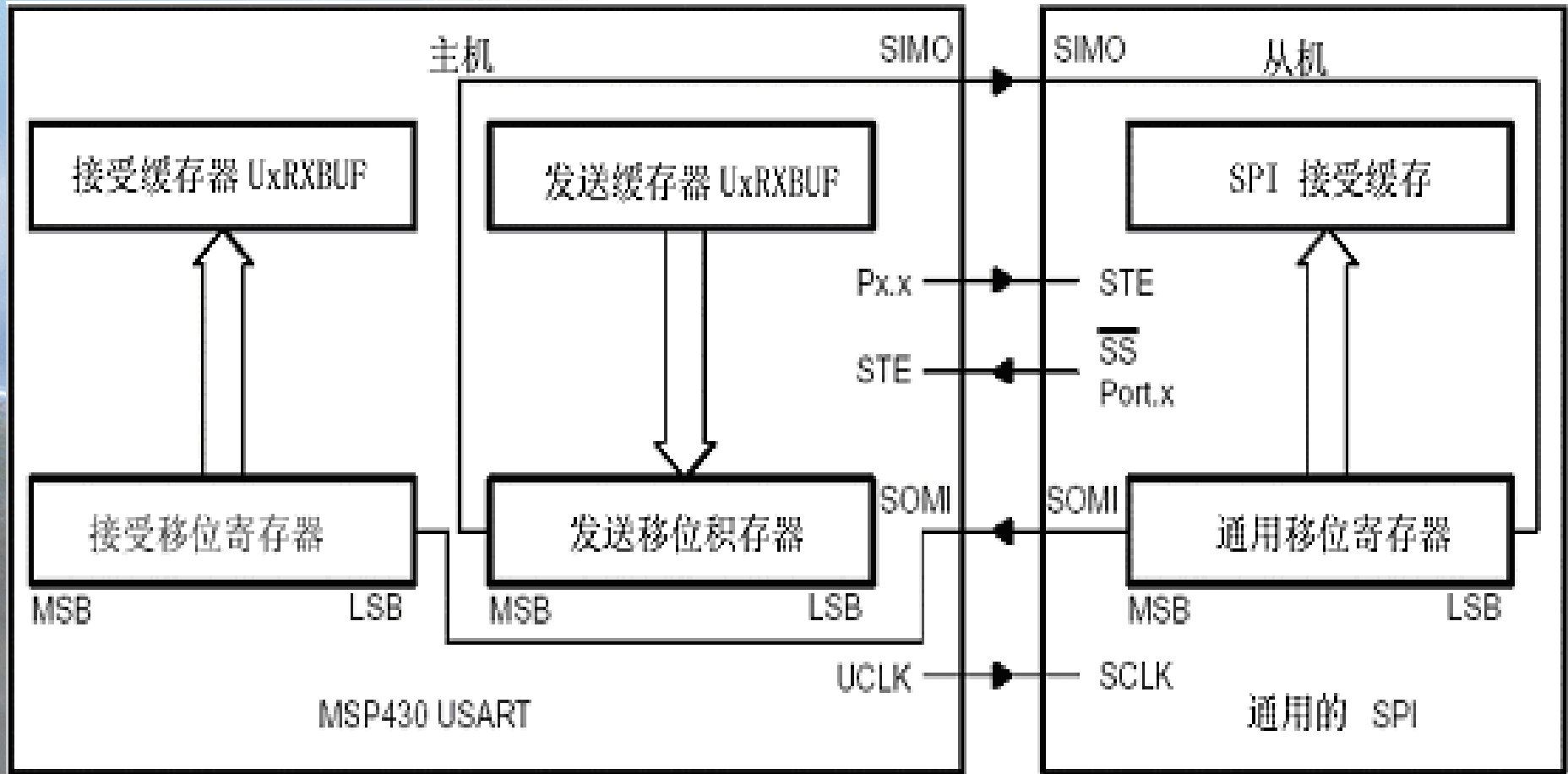
-
- **STE** Slave transmit enable. Used in 4-pin mode to allow multiple masters on a single bus. Not used in 3-pin mode.
4-Pin master mode:
When STE is high, SIMO and UCLK operate normally.
When STE is low, SIMO and UCLK are set to the input direction.
4-pin slave mode:
When STE is high, RX/TX operation of the slave is disabled and SOMI is forced to the input direction.
When STE is low, RX/TX operation of the slave is enabled and SOMI operates normally.

MSP430的同步通信模块特点

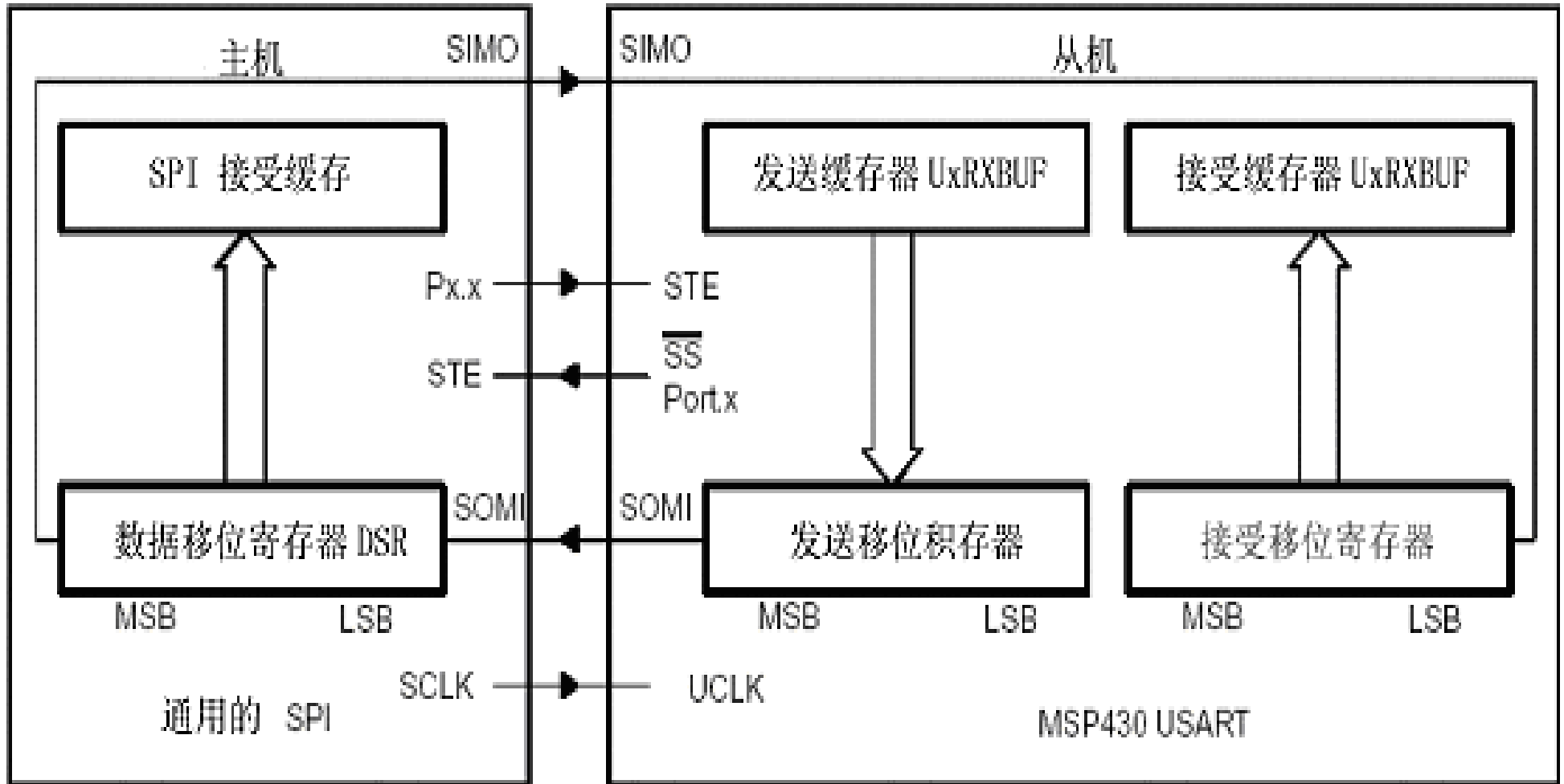
- | 支持3线或4线SPI操作
- | 支持主机模式与从机模式
- | 接收和发送有单独的移位寄存器
- | 接收和发送有独立的缓冲器
- | 接收和发送有独立的中断能力
- | 时钟的极性和相位可编程
- | 主模式的时钟频率可编程
- | 7位或8位字符长度



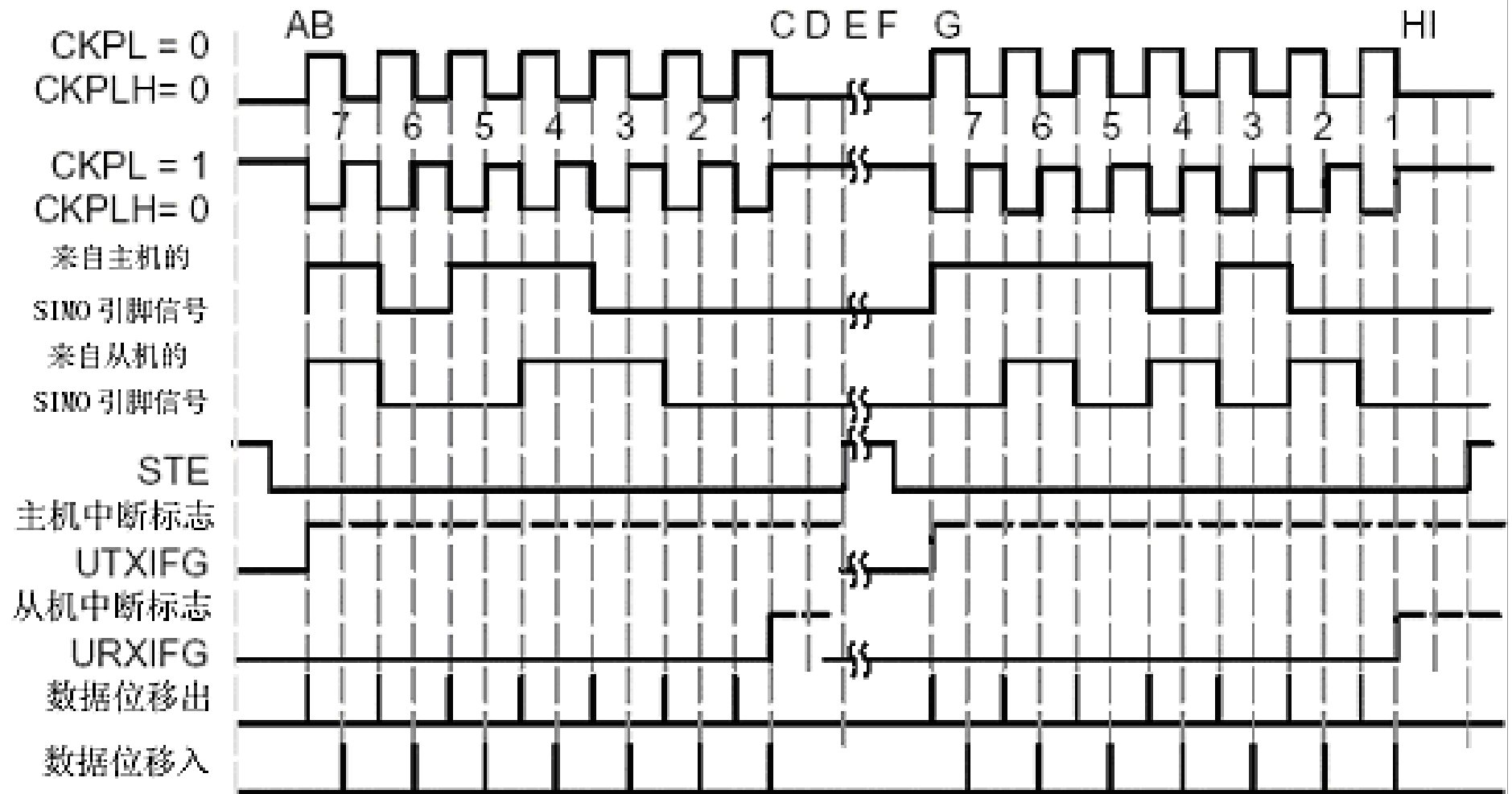
SPI的主机模式

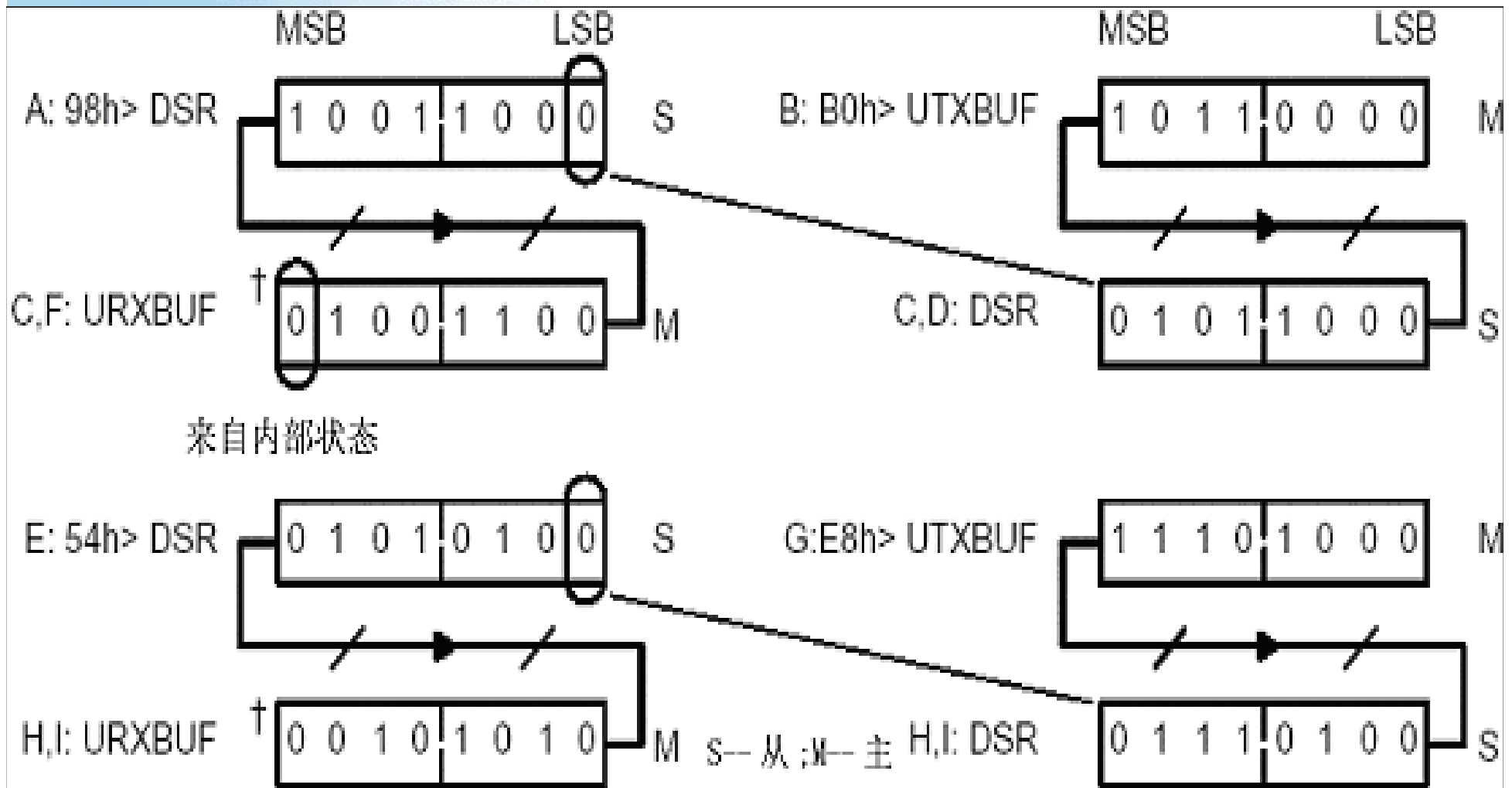


SPI的从机模式



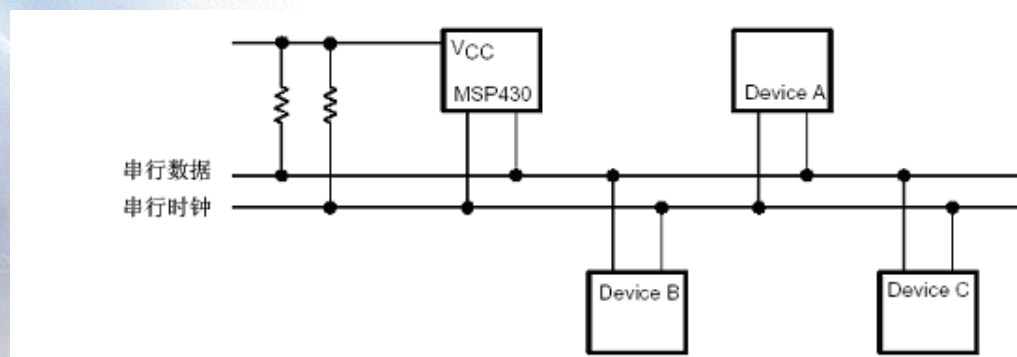
同步通信举例



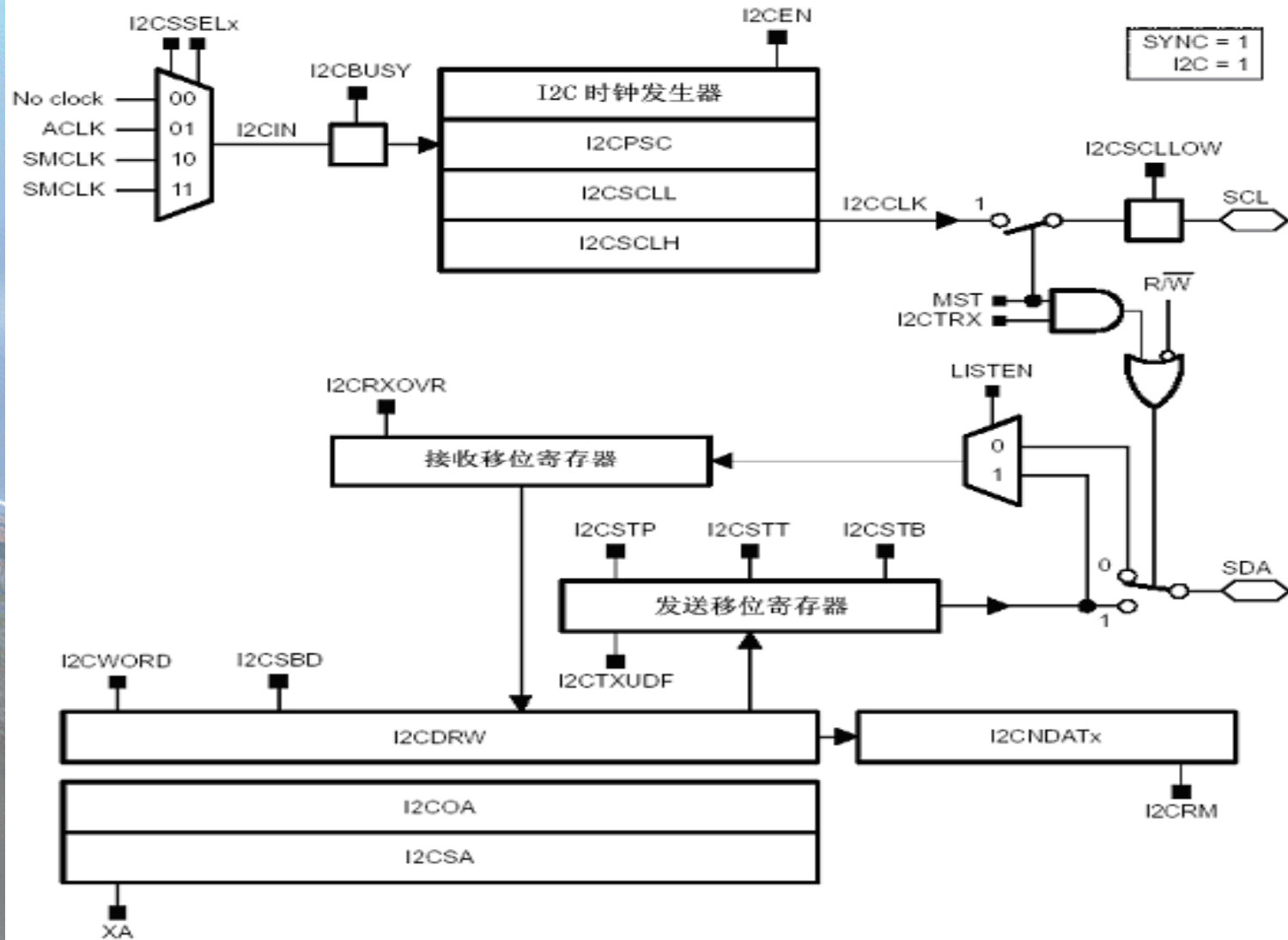


I2C概述

- 在现代电子系统中，有为数众多的**IC**需要进行相互之间以及与外界的通信。为了提高硬件效率和简化电路设计而广泛使用**Inter-IC**。
- Inter-IC (I2C)** 总线是一种用于内部**IC**控制的具有多端控制能力的双线双向串行数据总线系统。能够用于替代标准的并行总线，连接各种集成电路和功能模块。**I2C**器件的应用能够减少电路间连线，减小电路板尺寸，降低硬件成本，并提高了系统可靠性
- MSP430**和有关设备互连



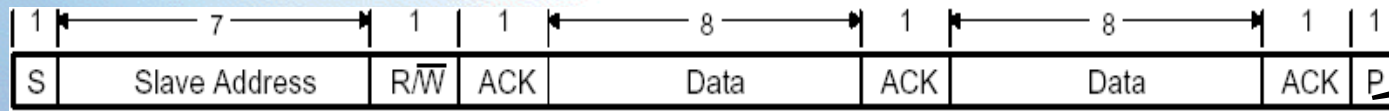
MSP430 I2C模块结构



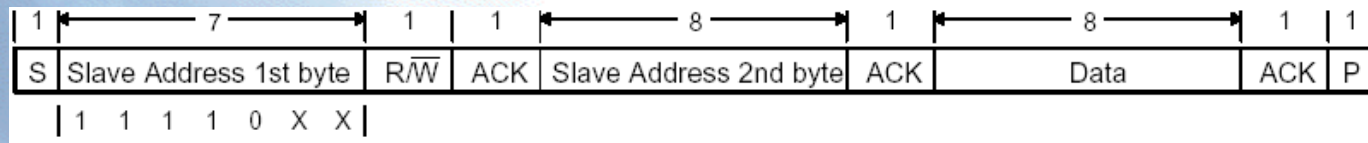
MSP430系列I2C模块的主要特征

- | 符合I2C规范V2.1
- | 读写采取先进先出缓冲结构
- | 可编程时钟发生器
- | 16位数据访问可达到总线的最大吞吐率
- | 自动数据字节计算
- | 支持低功耗模式
- | 从接收根据检测到开始信号自动将MSP430从LPM_x模式唤醒
- | 两个DMA触发源
- | 中断功能丰富
- | 只能用USART0实现I2C操作

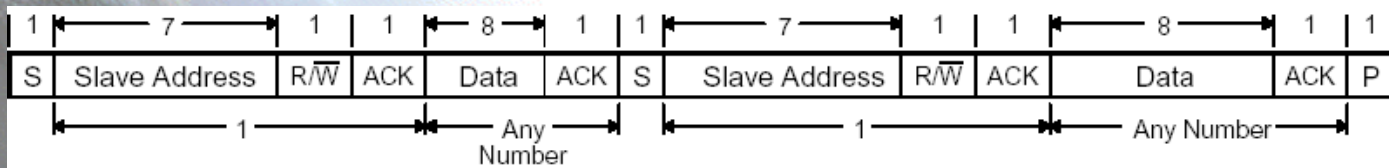
I2C的寻址模式



7位寻址模式



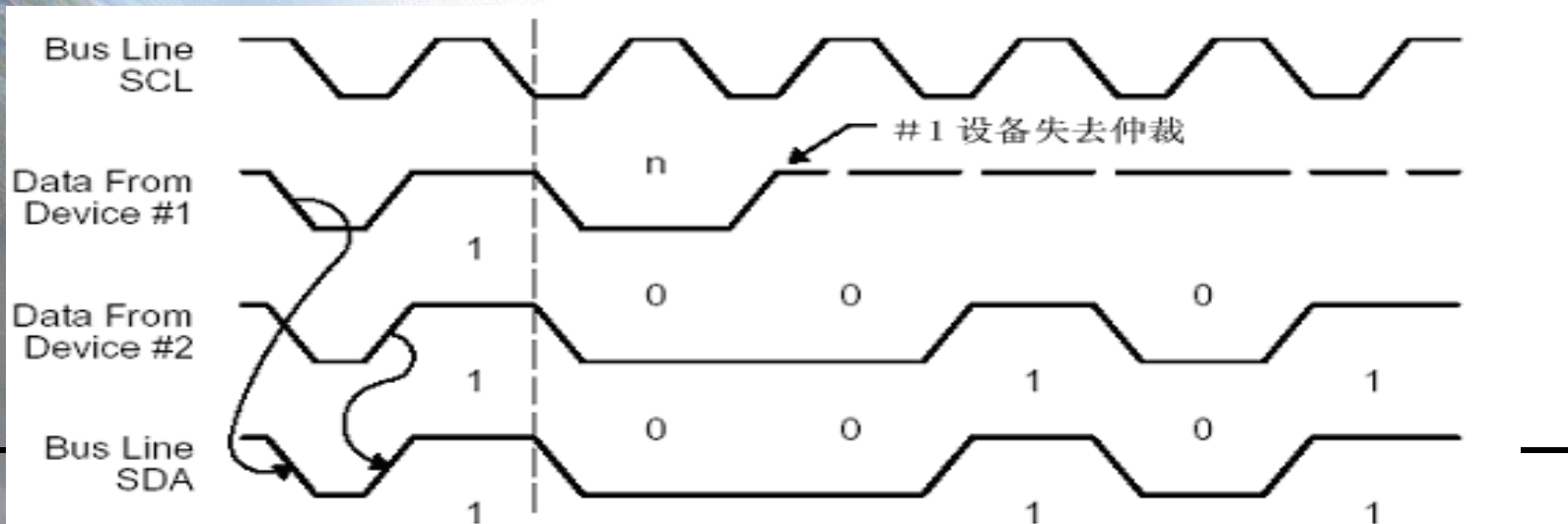
10位寻址模式



重复产生起始

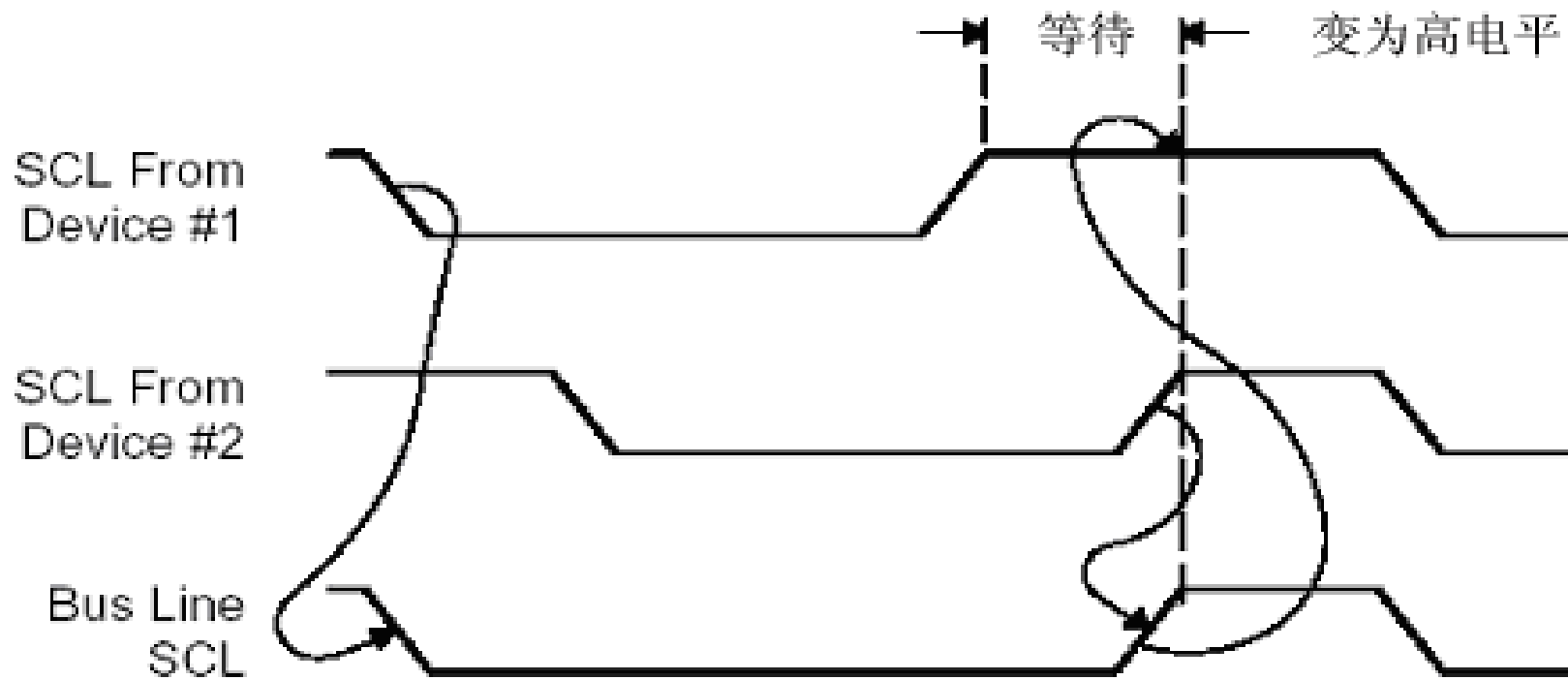
总线仲裁

- 当两个设备同时发出起始位进行数据传输时，相互竞争的设备使它们的时钟保持同步，正常发送数据。没有检测到冲突之前，每个设备都认为只有自己在使用总线
- 仲裁过程中使用的数据就是相互竞争的设备发送到SDA线上的数据。第一个检测到自己发送的数据和总线上数据不匹配的设备就失去仲裁能力。如果两个或更多的设备发送的第一个字节的内容相同，那么仲裁就发生在随后传输中。也许直到相互竞争的设备已经传输了许多字节后，仲裁才会完成。



时钟同步

- 仲裁过程中，要对来自不同主设备的时钟进行同步处理。在SCL上第一个产生低电平的主设备强制其他主设备也发送低电平，SCL保持为低，如果某些主设备已经结束低电平状态，就开始等待，直到所有的主设备都结束低电平时钟。



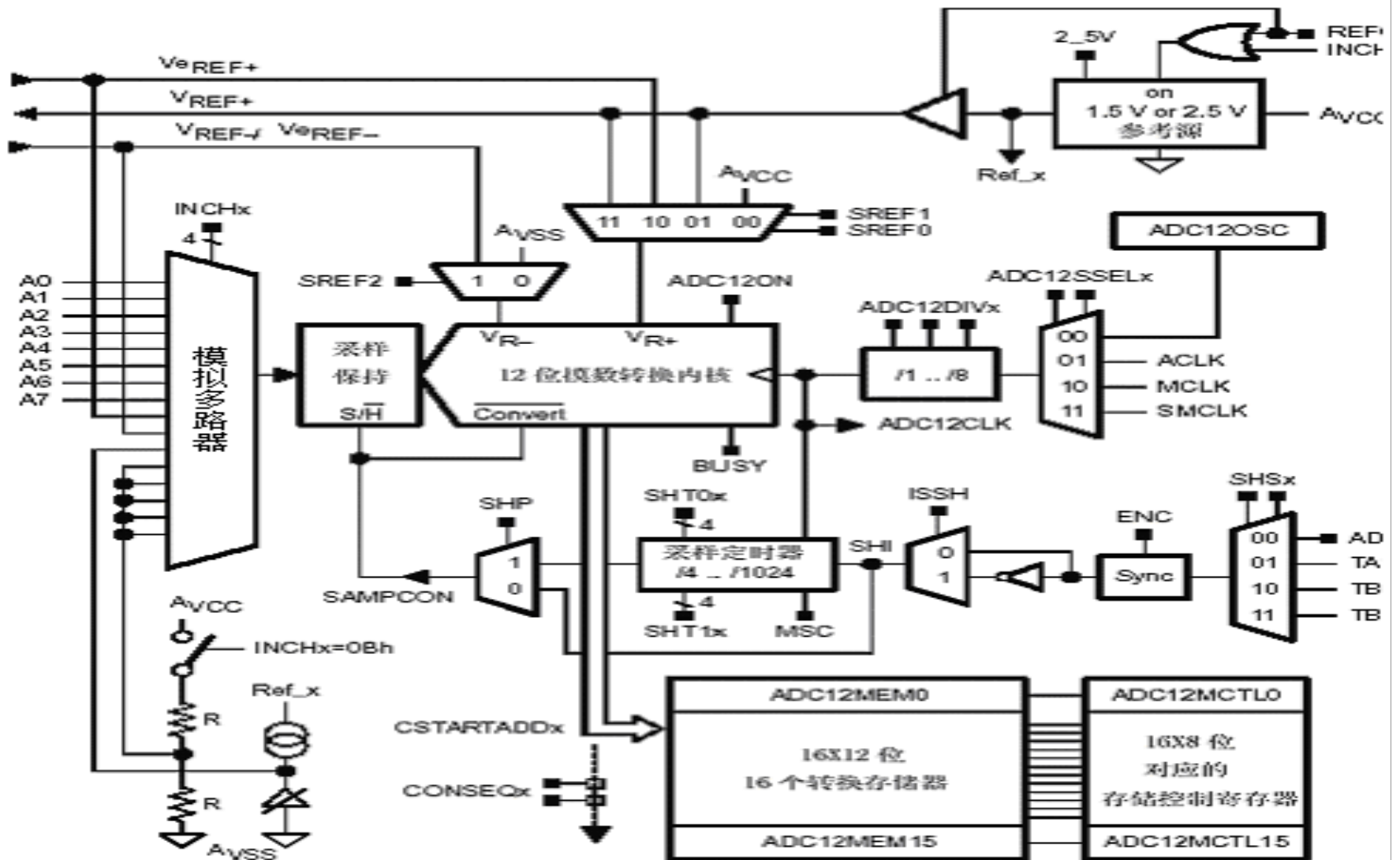
重新配置I2C模块进行UART或者SPI操作

- | 清除I2C、I2CEN和SYNC位（CLR.B &U0CTL）
- | 设置SWRST位（MOV.B #SWRST, &U0CTL）
- | 进行UART或者SPI模式的初始化
- ✓ 在SWART=1情况下初始化所有USART寄存器（包括UxCTL）
- ✓ 通过特殊功能寄存器MEx使能USART模块（URXEx, UTXEx之一或全部）
- ✓ 软件清除SWRST位（BIC.B #SWRST, &UxCTL）
- ✓ 通过特殊功能寄存器IEx中断使能（URXIEx, UTXIEx之一或全部）（可选）

复位条件下配置I2C模式

- | 在SWRST=1情况下选择I2C模式（BIS.B #SYNC+I2C, &U0CTL）
- | 清除I2CEN位（BIC.B #I2CEN, &U0CTL）
- | 在I2C=0情况下重新配置I2C模块
- | 软件设置I2CEN（BIS.B #I2CEN, &U0CTL）

$$N_{ADC} = 4095 \times \frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}}$$



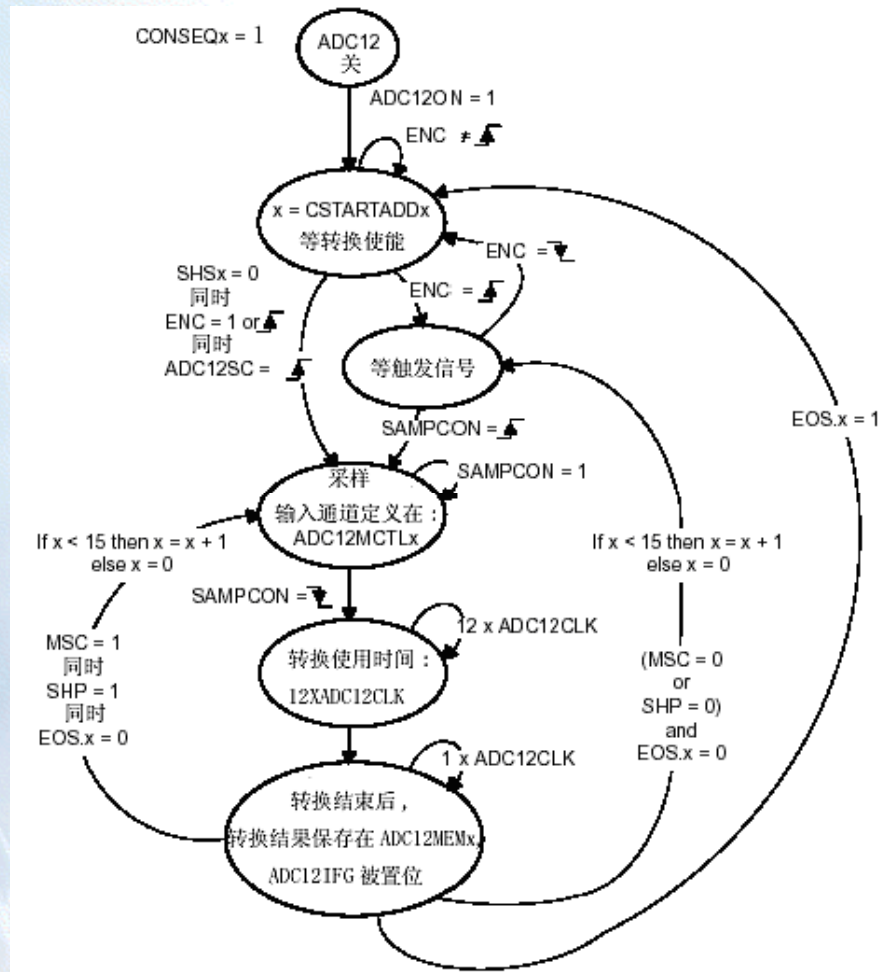
ADC12主要特点

- | 12位转换精度，1位非线性微分误差，1位非线性积分误差
 - | 有多种时钟源提供给ADC12模块，而且模块本身内置时钟发生器
 - | 内置温度传感器
 - | Timer_A/Timer_B硬件触发器
 - | 配置有8路外部通道与4路内部通道
 - | 内置参考电源，并且参考电压有6种组合
 - | 模数转换有4种模式
 - | 16字转换缓存
 - | ADC12可关断内核支持超低功耗应用
 - | 采样速度快，最高可达200ksps
 - | 自动扫描
 - | DMA使能
-

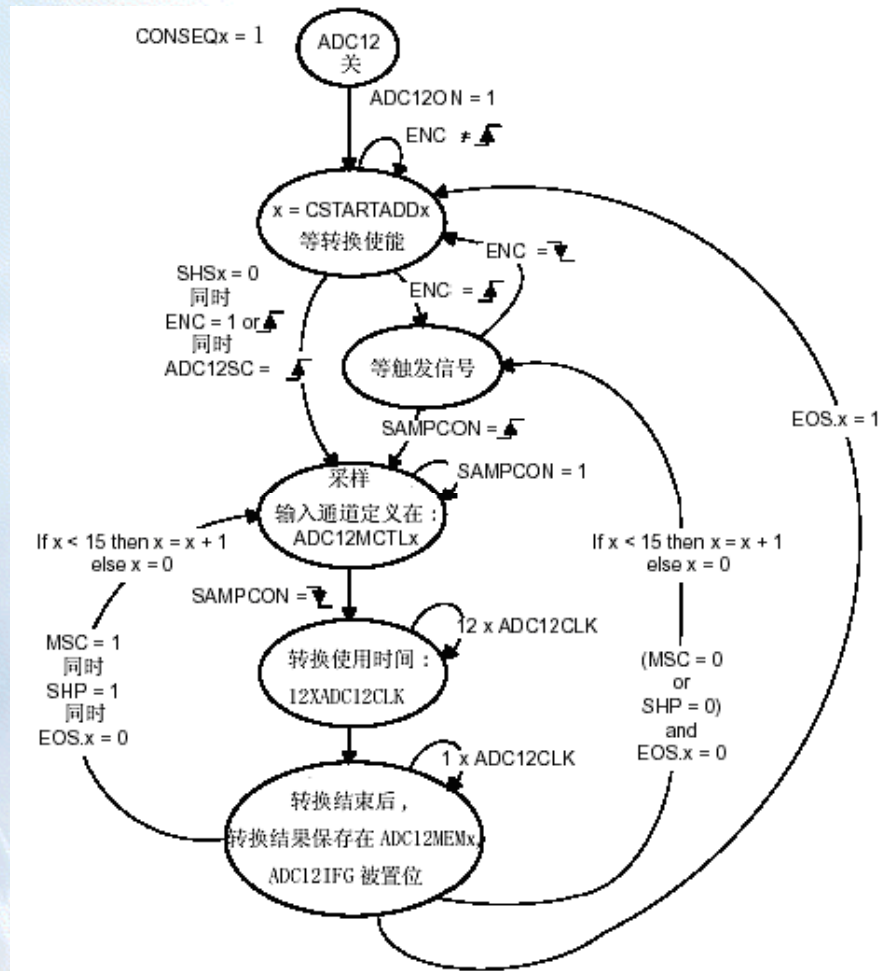
ADC12转换模式

- | 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

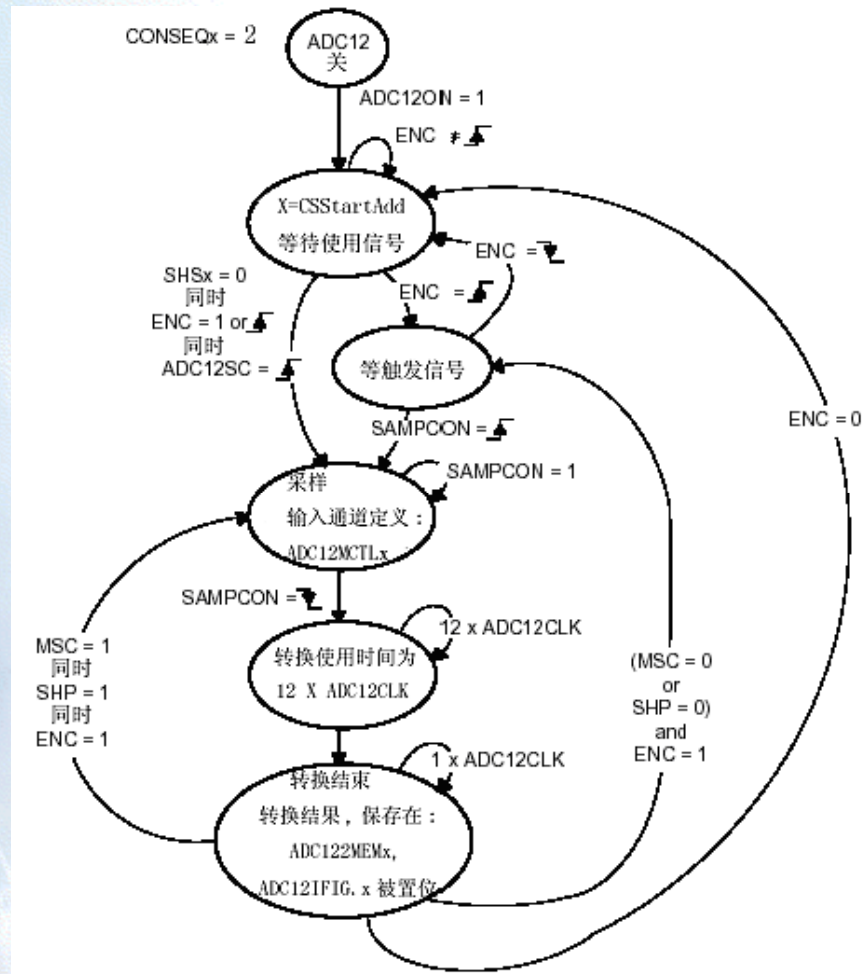
单通道单次转换模式状态



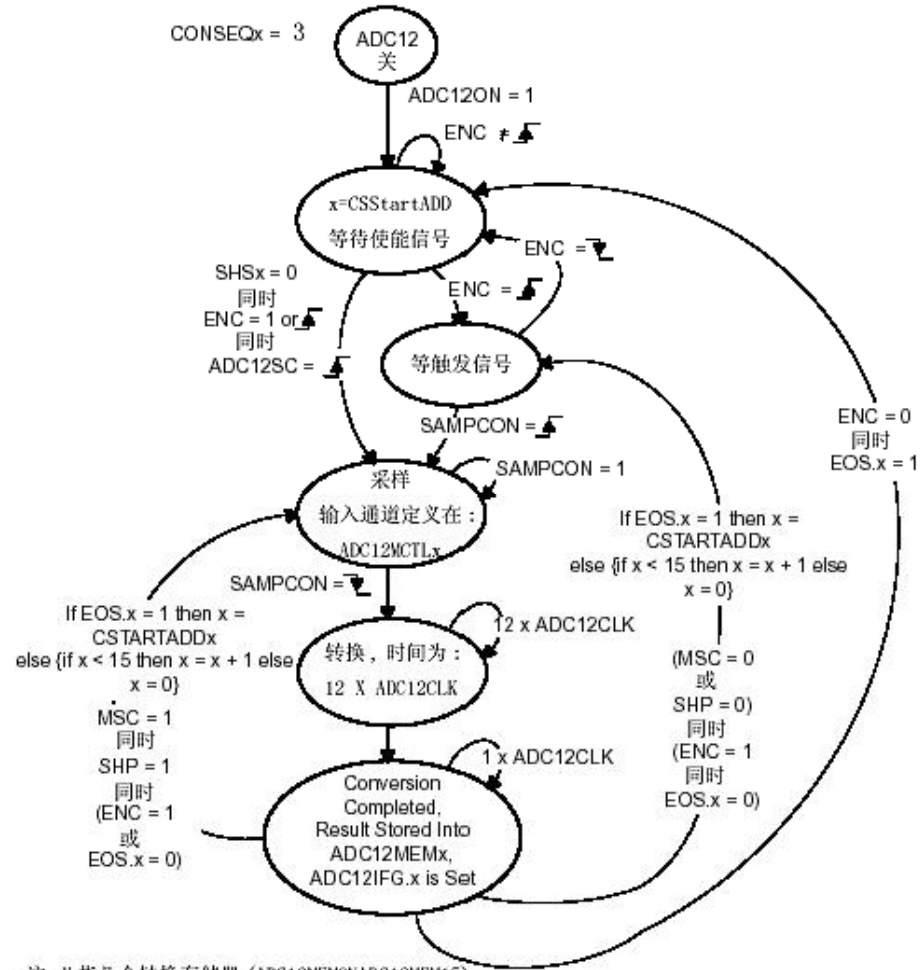
序列通道单次转换状态



单通道多次模式的状态



序列通道多次转换状态



注: X 指几个转换存储器 (ADC12MEM0—ADC12MEM15) 和转换存储控制寄存器 (ADC12MCTL0—ADC12MCTL15)

ADC12应用举例

I 使用外部参考源

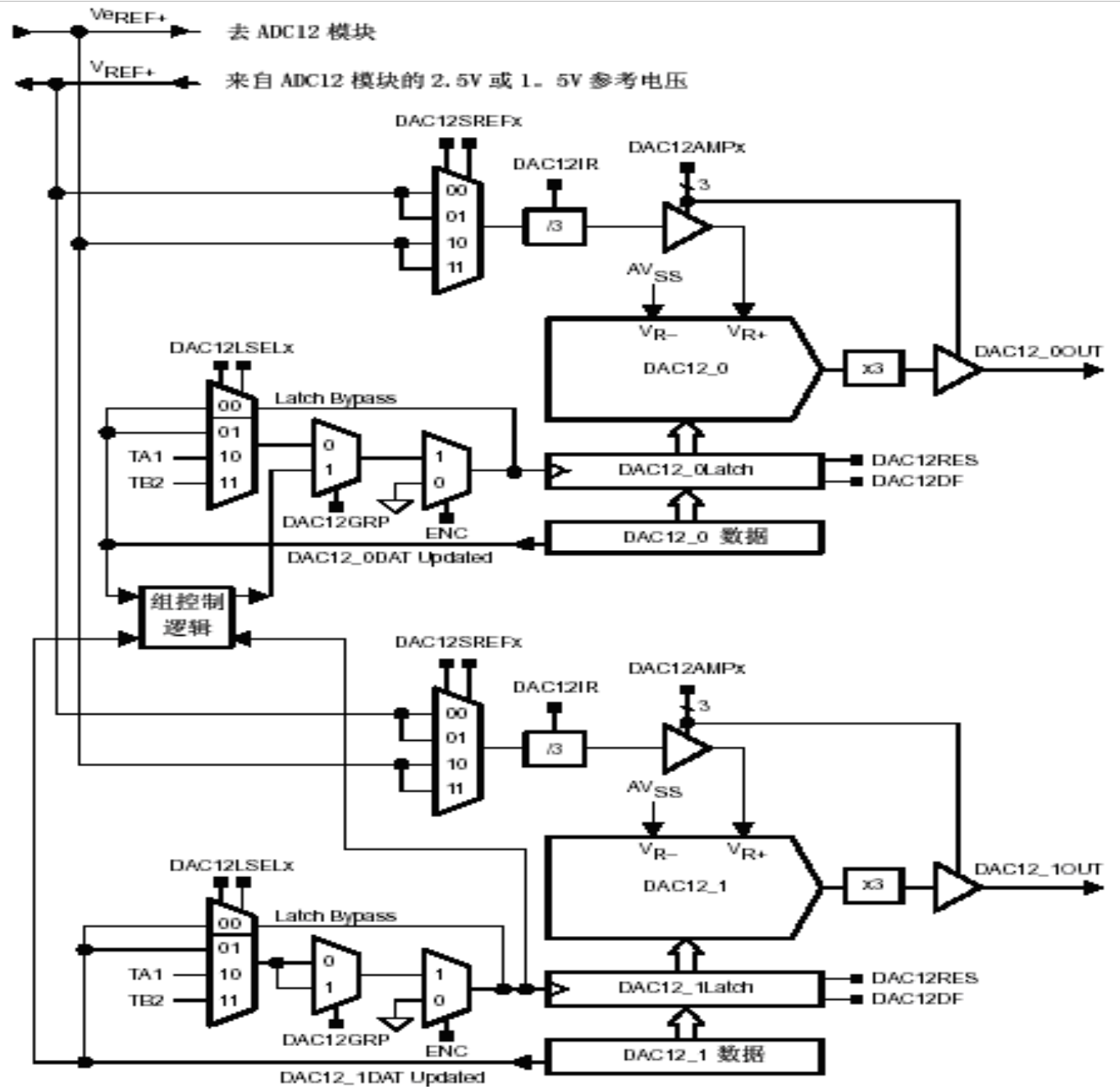
```
#include "msp430x44x.h"
void main(void)
{
    WDTCTL = WDTPW+WDTHOLD;
    P6SEL |= 0x01;           // 使能A/D 通道A0
    ADC12CTL0 = ADC12ON+SHT0_2; // 打开 ADC12, 设置
    采样时钟
    ADC12CTL1 = SHP;         // 使用采样时钟
    ADC12MCTL0 = SREF_2;    // Vr+ = VeREF+ (外部
)
    ADC12CTL0 |= ENC;       // 使能转换
    while (1)
    {
        ADC12CTL0 |= ADC12SC; // 开始转换
        while ((ADC12IFG & ADC12BUSY)==0);
        _NOP();
    }
}
```

ADC12应用举例

I 使用内部参考源

```
#include "msp430x44x.h"
void main(void)
{
    unsigned int i;
    WDTCTL = WDTPW+WDTHOLD;
    P6SEL |= 0x01;           // 使能 A/D 通道A0
    ADC12CTL0 = ADC12ON+SHT0_2+REFON+REF2_5V;
    ADC12CTL1 = SHP;
    ADC12MCTL0 = SREF_1;    // Vr+=Vref+
    for ( i=0; i<0x3600; i++) // 为参考源启动提供延迟
    {
    }
    ADC12CTL0 |= ENC;      // 使能转换
    while (1)
    {
        ADC12CTL0 |= ADC12SC; // 开始转换
        while ((ADC12IFG & BIT0)==0);
        _NOP();
    }
}
```

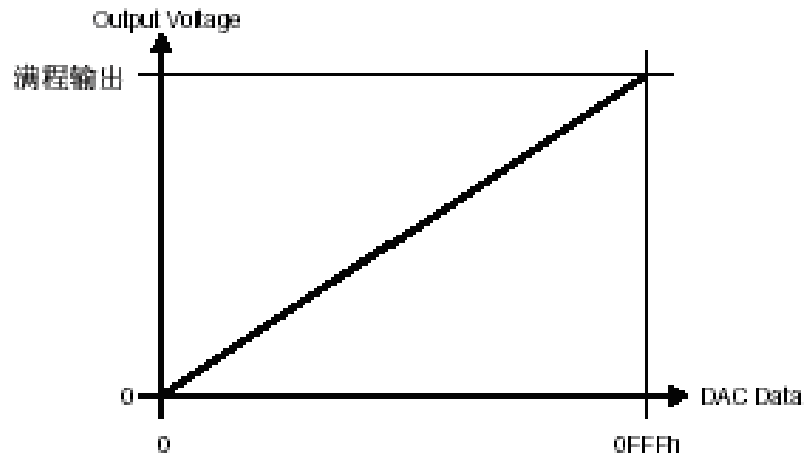

DAC12的结构



DAC12的主要特征

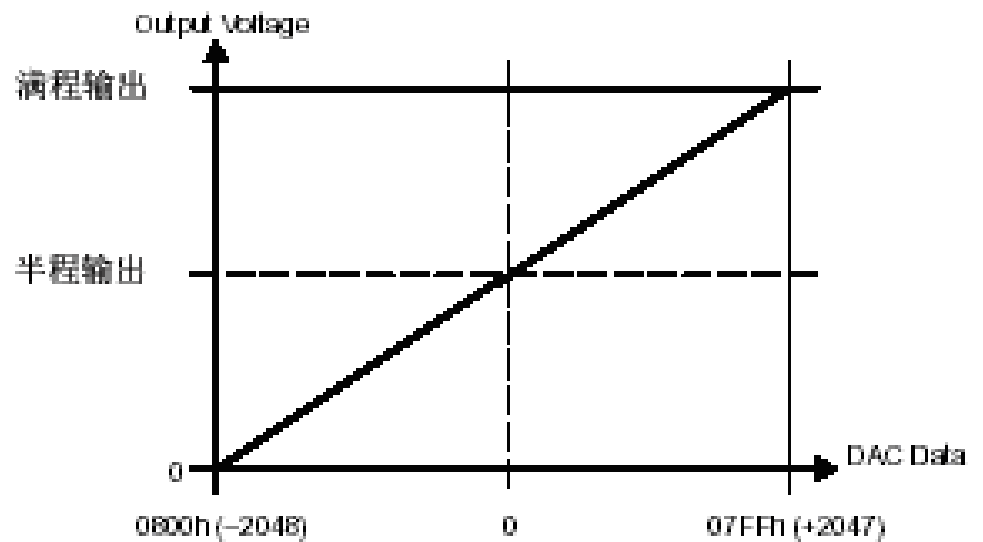
- | 8位、12位分辨率
- | 可编程的时间对能量消耗
- | 内部或外部参考电压
- | 支持无符号和有符号数据输入
- | 具有自校验功能
- | 二进制或者二的补码形式
- | 多路DAC同步更新
- | 可直接存储器存取

DAC12_xDAT的数据格式

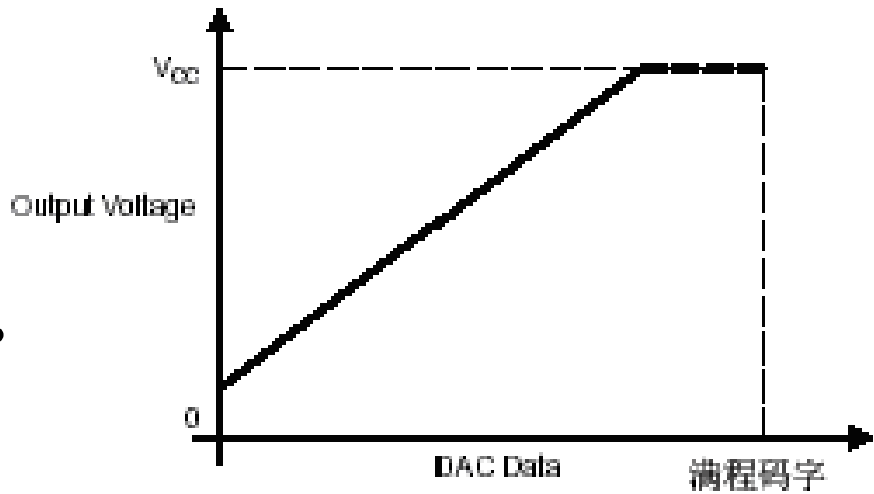
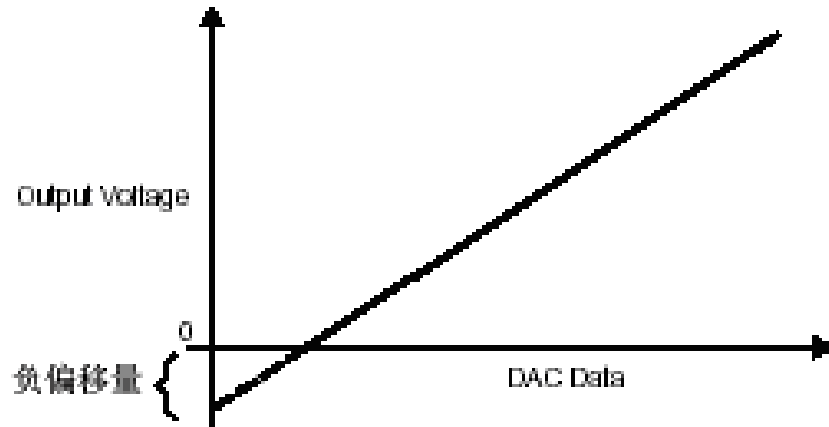


DAC12采用12位
二进制数格式

DAC12使用2
的补码形式

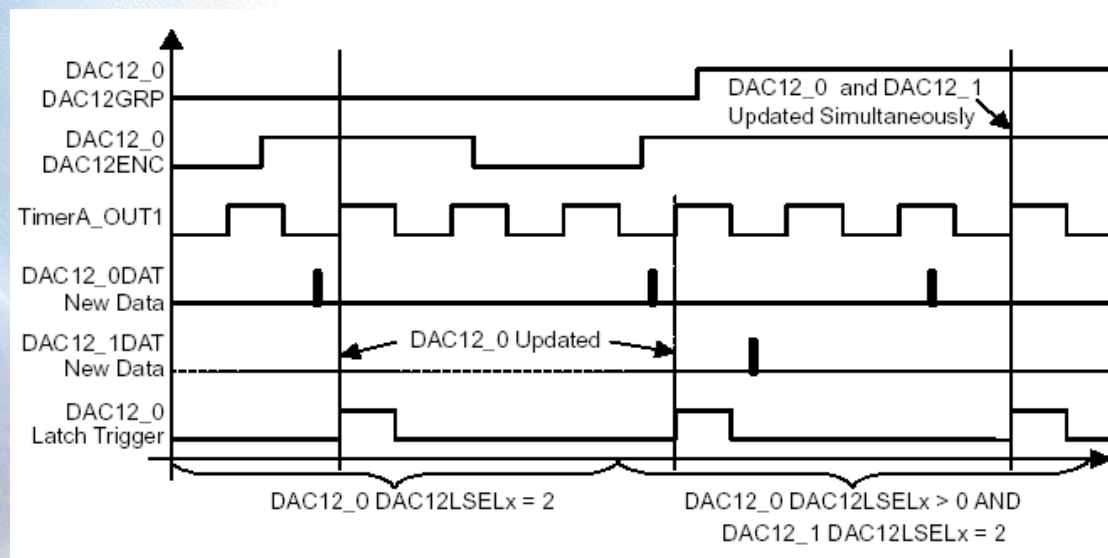


校正DAC12输出



组合多个DAC12模块

- MSP430x15x以及MSP430x16x中，DAC12_0和DAC12_1通过设置DAC12_0的DAC12GRP位实现组合。当DAC12_0和DAC12_1处于组合状态，只有两个转换通道的DAC12LSELx大于零并且DAC12ENC置位这两个条件同时满足情况下，才可以由DAC12_1DAC12LSELx位选择两个DAC的更新触发源。



DAC12应用举例

- **阶梯波的产生：**在一定时间范围内，每隔一段时间，输出幅度递增一个恒定值。阶梯波可以通过延迟程序或定时器来配合DAC12产生。
- **三角波的产生：**三角波是由两段直线组成，先输出一个线性增长的波形，达到最大值时，再送出一个线性减少的波形，这两个波形合到一起就成为三角波。可通过控制DAC12的输入值递增、递减来实现。
- **不规则信号的产生：**可以把不规则信号的采样值，存储在程序存储器中，然后用查表的方法读出这些值，送到DAC12一个通道后输出到Y轴上，同时利用另一个DAC12通道在X轴送出锯齿波，以产生水平扫描线。两个DAC12通道信号的频率应保持一定的比例关系，从而能够使显示波形保持同步。当然也可用这种方法产生规则的波形，如正弦波等。

在嵌入式操作系统领域，由Jean J.Labrosse开发的 μ C/OS，由于开放源代码和强大而稳定的功能，曾经一度在嵌入式系统领域引起强烈反响。而其本人也早已成为了嵌入式系统会议（美国）的顾问委员会的成员。

μ C/OS-II可以大致分成核心、任务处理、时间处理、任务同步与通信，CPU的移植等5个部分。

■ 嵌入式系统是以嵌入式计算机为核心，面向用户、面向产品、面向应用，软硬件可裁减的，适用于对功能、可靠性、体积、成本、功耗等综合性能有严格要求的计算机系统。随着嵌入式系统的广泛应用，传统的前/后台程序开发机制已经不能满足日益复杂和实际的实现要求，因而现场常常采用嵌入式实时操作系统RTOS（Real Time Operation System）开发实时多任务系统。嵌入式实时操作系统一般可以提供多任务的~~任务调度、时间管理、任务间通信和同步以及内存管理MMU（Memory Manager Unit）等重要服务，使得嵌入式应用程序易于设计和扩展。采用RTOS可以使嵌入式产品更可靠、开发周期更短。在嵌入式应用中~~使用RTOS已经成为当前嵌入式应用的一个热点。

完成简单功能的嵌入式系统一般不需要操作系统。如，以前许多MCS51系列单片机组成的小系统就只是利用软件实现简单的控制环路；但是随着所谓后PC时代的来临，嵌入式系统设计日趋复杂，嵌入式操作系统就必不可少了。

嵌入式RTOS在系统实时高效性、硬件的相关依赖性、软件固化以及应用的专业性等方面具有较为突出的优势。一般而言，嵌入式操作系统不同于一般意义的计算机操作系统，它有占用空间小、执行效率高、方便进行个性化定制和软件要求固化存储等特点。

从20世纪80年代起，国际上就有一些IT组织、公司，开始进行商用嵌入式操作系统和专用操作系统的研发。这其中涌现了一些著名的嵌入式操作系统，如Microsoft公司的 WinCE 和 WindRiver System公司的 VxWorks就分别是非实时和实时嵌入式操作系统的代表。但是商用产品的造价都十分昂贵，用于一般用途会提高产品成本从而失去竞争力。

UC/OS和 uClinux操作系统是用两种性能优良、源码公开且被广泛应用的免费嵌入式操作系统，可以作为研究实时操作系统和非实时操作系统的典范。

■ uC/OS和 uClinux操作系统，是当前得到广泛应用的两种免费且开源的嵌入式操作系统。UC/OS适合小型控制系统，具有执行效率高、占用空间小、实时性能优良和可扩展性强等特点，最小内核可编译至2KB。

UClinux则是继承标准Linux的优良特性，针对嵌入式处理器的特点设计的一种操作系统，具有内嵌网络协议、支持多种文件系统，开发者可利用标准Linux先验知识等优势。其编译后目标文件可控制在几百KB量级。

■ 嵌入式操作系统是嵌入式系统软硬件资源的控制中心，它以尽量合理的有效方法组织多个用户共享嵌入式系统的各种资源。其中用户指的是系统程序之上的所有软件。所谓合理有效的方法，指的就是操作系统如何协调并充分利用硬件资源来实现多任务。复杂的操作系统都支持文件系统，方便组织文件并易于对其规范化操作。

■ 嵌入式操作系统还有一个特点是，针对不同的平台，系统不是直接可用的，一般需要经过针对专门平台的移植操作系统才能正常工作。

■ 进程调度、文件系统支持和系统移植是在嵌入式操作系统实际应用中最常见的问题。

■ 不管是对于初学者，还是有经验的工程师， μ C/OS开放源代码的方式使其不但知其然，还知其所以然。通过对于系统内部结构的深入了解，能更加方便地进行开发和调试；并且在这种条件下，完全可以按照设计要求进行合理的裁减、扩充、配置和移植。通常，购买RTOS往往需要一大笔资金，使得一般的学习者望而却步；而 μ C/OS对于学校研究完全免费，只有在应用于盈利项目时才需要支付少量的版权费，特别适合一般使用者的学习、研究和开发。自1992第1版问世以来，已有成千上万的开发者把它成功地应用于各种系统，安全性和稳定性已经得到认证，现已经通过美国FAA认证。

■ 核心部分(OSCore.c) 是操作系统的处理核心，包括操作系统初始化、操作系统运行、中断进出的前导、时钟节拍、任务调度、事件处理等多部分。能够维持系统基本工作的部分都在这里。

■ 任务处理部分(OSTask.c) 任务处理部分中的内容都是与任务的操作密切相关的。包括任务的建立、删除、挂起、恢复等等。因为 μ C/OS-II是以任务为基本单位调度的，所以这部分内容也相当重要。

■ 时钟部分(OSTime.c) μ C/OS-II中的最小时钟单位是`timetick`（时钟节拍）。任务延时等操作是在这里完成的。

■ 任务同步和通信部分 为事件处理部分，包括信号量、邮箱、邮箱队列、事件标志等部分；主要用于任务间的互相联系和对临界资源的访问

■ 与CPU的接口部分 是指 μ C/OS-II针对所使用的CPU的移植部分。由于 μ C/OS-II是一个通用性的操作系统，所以对于关键问题上的实现，还是需要根椐具体CPU的具体内容和要求作相应的移植。这部分内容由于牵涉到SP等系统指针，所以通常用汇编语言编写。主要包括中断级任务切换的底层实现、任务级任务切换的底层实现、时钟节拍的产生和处理、中断的相关处理部分等内容。

■ MSP430最常用的C编译器应该就是IAR Embedded WorkBench。对于这一编译器来说，通过分析和研究，发现它有以下规律。

■ (1) 函数调用

如果是函数级调用，编译器会在函数调用时先把当前函数PC压栈，然后调用函数，PC值改变。

如果被调用的函数带有参数，那么，编译器按照以下的规则进行。

最左边的两个参数如果不是**struct**（结构体）或者**union**（联合体），将被赋值到寄存器，否则将被压栈。函数剩下的参数都将被压栈。根据最左边的那两个参数的类型，分别赋值给R12（对于32位类型赋值给R12:R13）和R14（对于32位类型赋值给R14:R15）。

(2) 中断调用

如果是在中断中调用中断服务子程序的话，编译器将把当前执行语句的PC压栈，同时再把SR压栈。接着，根据中断服务子程序的复杂程度，选择把R12~R15中的寄存器压栈。然后，执行中断服务子程序。中断处理结束后再把Rx寄存器出栈，SR出栈，PC出栈。把系统恢复到中断前的状态，使程序接着被中断的部分继续运行。

任务调度主要是协调任务对计算机系统资源（如内存、I/O设备、CPU）的争夺使用。进程调度又称为CPU调度，其根本任务是按照某种原理为处于就绪状态的进程分析CPU。由于嵌入式系统中内存和I/O设备一般都和CPU同时归属于某进程，所以任务调度和进程调度概念相近，很多场合不加区分。

■ 进程调度可分为“剥夺型调度”和“非剥夺型调度”两种基本方式。所谓“非剥夺型调度”是指：一旦某个进程被调度执行，则该进程一直执行下去直至该进程结束，或由于某种原理自行放弃CPU进入等待状态，才将CPU重新分配给其它进程。所谓“剥夺型调度”是指：一旦就绪状态中出现优先权更高的进程，或者运行的进程已用满了规定的时间片时，便立即剥夺当前进程的运行（将其放回就绪状态），把CPU分配给其它进程。

■ 作为实时操作系统，uC/OS采用的是可剥夺型实时多任务内核。可剥夺型的实时内核在任何时候都运行就绪了的最高优先级的任务。uC/OS中最多可以支持64个任务，分别对应优先级0~63，其中0为最高优先级。调度工作的内容可以分为两部分：最高优先级任务的寻找和任务切换。

■ 其最高优先级任务表来实现的。UC/OS中的每一个任务都有独立的堆栈空间，并有一个称为任务控制块TCB（Task Control Block）的数据结构，其中第一个成员变量就是保存的任务堆栈指针。任务调度模块首先用变量 OSTCBHighRdy记录当前最高级就绪任务的TCB地址，然后调用 OS_TASK_SW（）函数来进行任务切换。