



# MSP430 系列

**超低功耗**

**16位单片机原理与应用**

**Microcontroller**  
MSP430

胡大可 主编

北京航空航天大学出版社

<http://www.buaapress.cn.net>

73-6782  
0952

阅 8 清

# MSP430 系列超低功耗 16 位单片机

## 原理与应用

胡大可 主编

北京航空航天大学出版社  
<http://www.buaapress.cn.net>

## 内 容 简 介

TI 公司的 MSP430 系列微控制器是一个近期推出的单片机品种,它在超低功耗和功能集成上都有一定的特色,尤其适合应用在自动信号采集系统、液晶显示智能化仪器、电池供电便携式装置、超长时间连续工作设备等领域。本书对这一系列产品的原理、结构及内部各功能模块作了详细的说明,并以方便工程师及程序员使用的方式提供软件和硬件资料。由于 MSP430 系列的各个不同型号基本上都是这些功能模块的不同组合,因此,掌握本书的内容对于 MSP430 系列的原理理解和应用开发都有较大的帮助。本书的内容主要根据 TI 公司的《MSP430 Family Architecture Guide and Module Library》一书及其他相关技术资料编写。

本书供高等院校自动化、计算机、电子等专业的教学参考及工程技术人员实用参考,亦可做为应用技术的培训教材。

### 图书在版编目(CIP)数据

MSP 430 系列超低功耗 16 位单片机原理与应用/胡大可主编. —北京:北京航空航天大学出版社,2000.6  
ISBN 7-81012-990-2

I. M... II. 胡... III. 单片微型计算机  
IV. TP368.1

中国版本图书馆 CIP 数据核字(2000)第 30865 号

### MSP430 系列超低功耗 16 位单片机原理与应用

胡大可 主编

责任编辑 马广云

责任校对 陈 坤

北京航空航天大学出版社出版发行

北京学院路 37 号(邮编 100083) 发行部电话:(010)82317024 发行部传真:(010)82328026

<http://www.buaapress.cn.net>

E-mail: [press01@publib.bj.cninfo.net](mailto:press01@publib.bj.cninfo.net)

北京宏文印刷厂 印装 各地书店经销

\*

开本:787×1092 1/16 印张:17 字数:432千字

2000年6月第1版 2000年6月第1次印刷 印数:0~5000册

ISBN 7-81012-990-2/TP·404 定价:29.50元

## 序 言

温州利尔达电子器材公司的发展,与电子元器件的应用和发展是息息相关的。从最初经营收音机等小家电使用的电容器、电阻、晶体管等器件开始,经历了74系列、C4000、C4500等逻辑芯片时代。现在,公司的经营品种过渡到了以各种单片机为主体的几十个系列、一万多种型号元器件的经营模式。公司的经营结构也发生了很大变化,由一家器件贸易商发展成为一个以技术为主导,具有新产品研究、生产、客户技术支持等服务能力,且科、工、贸为一体的综合型企业。

当今,为满足各种适时控制系统、智能化仪表等领域的应用需要,世界各大单片机生产厂商开发出了一些各具特色的单片机,其中美国TI(Texas Instruments,德州仪器)公司的MSP430系列单片机就是一种以超低功耗为主要特色的16位单片机。而温州利尔达电子器材公司,则是美国TI公司MSP430系列单片机的中国代理商。为使我国的工程设计人员能更快地利用MSP430系列单片机的特色设计出低功耗产品,温州利尔达电子器材公司组织编写了这本书。

在单片机技术应用开发的过程中,选择一个性能价格比合适的单片机型号是至关重要的,当应用的场合需要满足低功耗、高集成度、宽环境温度范围、高精度A/D转换等技术性能要求时,TI公司的MSP430系列则是一个较为理想的选择。

MSP430系列单片机具有超低功耗、16位指令字、A/D转换器、串行通信接口、硬件乘法器、LCD驱动电路及高抗干扰能力等技术特点,因此,特别适合应用在智能仪表、防盗系统、智能化家用电器、电池供电便携式设备等产品之中。

本书全面介绍了MSP430系列单片机的基本结构以及各功能模块的原理和应用,内容详尽,在开发过程中是不可缺少的。目前,MSP430系列单片机在国内的应用正在逐步推广,相信本书对选用MSP430系列单片机开发产品的技术人员将会有较大的帮助。

为方便客户,我公司总部(温州)以及杭州分公司(杭州康尔达电子有限公司)、北京办事处均可办理MSP430系列单片机业务,并提供技术支持和服务。随着市场的发展,我公司将在国内其他大、中城市设立办事机构,以求为客户提供更便捷的服务。

本书的出版发行得到了北京航空航天大学出版社及马广云博士的大力支持,在此表示感谢。

温州利尔达电子器材公司将在提供器件、开发工具、技术支持,以及介绍关于MSP430系列技术资料等方面继续作出努力。

温州利尔达电子器材公司董事长  
陈贤兴

2000年6月

## 前 言

单片机技术,或者称为微控制器技术,已经在人类生活的方方面面都得到应用。它特别适合用在各种专用、小型、省电、可移动的设备之中。它多年来的快速发展使得它的家族越来越庞大、系列品种越来越多,而且技术上往往也各有特色。

TI公司的MSP430系列是一个特别强调超低功耗的单片机品种,很适合应用在各种功率消耗要求特别低的场合。它具有一定的技术特点。在这个系列中有多个型号,它们是由一些基本功能模块按不同的应用目标组合而成的。本书集中介绍所有这些基本功能模块的原理、控制方法及应用。书中内容适合MSP430系列的各个型号。

TI公司的MSP430系列又可分为几个分支,例如:11x、31x、32x、33x等,而且还在不断发展。从应用角度,又可根据程序存储器的类型分为ROM(C型)、OTP(P型)、EPROM(E型)、Flash Memory(F型)等,以适应开发各个阶段的需要。

阅读本书的读者应具备数字电路知识,最好有其他单片机品种的开发经历。

本书内容参考了TI公司的《MSP430 Family Architecture Guide and Module Library》一书及其他一些有关技术资料。在此感谢TI公司对引述《MSP430 Family Architecture Guide and Module Library》一书的授权同意。读者对该书有关译文内容的引述如有疑问,请参考TI公司的原著。

在成书过程中,得到研究生季燕飞、潘卫江、吴一聪、叶隽等人的协助,同时也得到TI公司MSP430系列产品代理商——利尔达电子器材公司的大力支持。

由于作者的学识水平有限,书中难免有错误和不妥之处,恳请读者批评指正。

作 者

2000年4月

## 本书表述约定

表示信号和处理机状态的符号的简要说明如下：

- ADC A/D 转换器。
- CPUOff mode 保持 RAM 及 I/O 信号不变的低功耗模式。  
用辅助时钟(32 768 Hz 晶振)工作的模块处于活动状态。
- DCO 数字控制振荡器。
- LCD 液晶显示器。
- FF 触发器。
- MAB 存储器地址总线。位于各内部模块之间。可以在 4 ~ 16 位范围内的任意宽度。它与 MS 信号一起定义了物理地址。
- MDB 存储器数据总线。位于各内部模块之间。可以是 8 位或 16 位宽度。
- MS 模块选择。为预解码地址空间。它与 MAB 一起定义了物理地址。
- MSFR 模块特殊寄存器。是特殊寄存器的预解码地址空间(00h ~ 0Fh)。
- OscOff mode 最低功耗模式。保持 RAM 及 I/O 信号不变。晶振停止。
- OTP 单次可编程。
- POR 上电复位。
- PUC 上电清除,“1”设置处理机启动条件。
- SAR 逐位逼近寄存器。
- SCI 处理同步及异步协议的串行通信接口。
- SCG 系统时钟发生器。
- SFR 特殊功能寄存器。
- SPI 串行外围接口(广泛应用的同步串行通信协议)。
- TBD 待定义。
- TOS 堆栈顶。
- UART 通用异步收发(最广泛应用的串行通信协议)。
- USART 通用同步异步收发。
- WD、WDT 看门狗、看门狗定时器。

寄存器位类型约定：

- rw 读/写。
- r 只读。
- r0 读出为“0”。
- w 只写。
- (w) 无寄存功能,写“1”将产生一个脉冲。读出总是为“0”。
- -0、-1 发生 PUC 后的状态。
- -(0)、-(1) 发生 POR 后的状态。
- h0 由硬件复位

运算符：

- @ 寄存器间接寻址。

# 目 录

本书表述约定

## 第 1 章 MSP430 系列

- 1.1 特性与功能..... 1
- 1.2 系统关键特性..... 1
- 1.3 MSP430 系列的各种型号..... 2

## 第 2 章 结构概述

- 2.1 CPU..... 4
- 2.2 代码存储器..... 5
- 2.3 数据存储器..... 5
- 2.4 运行控制..... 5
- 2.5 外围模块..... 5
- 2.6 振荡器、倍频器和时钟发生器..... 6

## 第 3 章 系统复位、中断和工作模式

- 3.1 系统复位和初始化..... 7
- 3.2 中断系统结构..... 8
- 3.3 中断处理..... 10
  - 3.3.1 SFR 中的中断控制位..... 12
  - 3.3.2 外部中断..... 14
- 3.4 工作模式..... 16
- 3.5 低功耗模式..... 18
  - 3.5.1 低功耗模式 0 和模式 1..... 19
  - 3.5.2 低功耗模式 2 和模式 3..... 19
  - 3.5.3 低功耗模式 4..... 20
- 3.6 低功耗应用要点..... 20

## 第 4 章 存储器组织

- 4.1 存储器中的数据..... 22
- 4.2 片内 ROM 组织..... 22
  - 4.2.1 ROM 表的处理..... 23
  - 4.2.2 计算分支跳转和子程序调用..... 23
- 4.3 RAM 与外围模块组织..... 23
  - 4.3.1 RAM..... 24
  - 4.3.2 外围模块——地址定位..... 25
  - 4.3.3 外围模块——SFR..... 27

## 第 5 章 16 位 CPU

5.1 CPU 寄存器 .....	29
5.1.1 程序计数器 PC .....	29
5.1.2 系统堆栈指针 SP .....	29
5.1.3 状态寄存器 SR .....	31
5.1.4 常数发生寄存器 CG1 和 CG2 .....	32
5.2 寻址模式 .....	33
5.2.1 寄存器模式 .....	34
5.2.2 变址模式 .....	34
5.2.3 符号模式 .....	35
5.2.4 绝对模式 .....	36
5.2.5 间接模式 .....	37
5.2.6 间接增量模式 .....	38
5.2.7 立即模式 .....	39
5.2.8 指令的时钟周期与长度 .....	40
5.3 指令集概述 .....	42
5.3.1 双操作数指令 .....	42
5.3.2 单操作数指令 .....	43
5.3.3 条件跳转 .....	43
5.3.4 模拟指令的简短格式 .....	44
5.3.5 其他指令 .....	45
5.4 指令分布 .....	45
<b>第 6 章 硬件乘法器</b>	
6.1 硬件乘法器的操作 .....	48
6.2 硬件乘法器的寄存器 .....	51
6.3 硬件乘法器的 SFR 位 .....	52
6.4 硬件乘法器的软件限制 .....	52
6.4.1 硬件乘法器的软件限制——寻址模式 .....	52
6.4.2 硬件乘法器的软件限制——中断程序 .....	52
<b>第 7 章 振荡器与系统时钟发生器</b>	
7.1 晶体振荡器 .....	54
7.2 处理机时钟发生器 .....	55
7.3 系统时钟工作模式 .....	56
7.4 系统时钟控制寄存器 .....	58
7.4.1 模块寄存器 .....	58
7.4.2 与系统时钟发生器相关的 SFR 位 .....	59
7.5 DCO 典型特性 .....	60
<b>第 8 章 数字 I/O 配置</b>	
8.1 通用端口 P0 .....	61
8.1.1 P0 的控制寄存器 .....	61



8.1.2	P0 的原理图 .....	63
8.1.3	P0 的中断控制功能 .....	66
8.2	通用端口 P1、P2 .....	66
8.2.1	P1、P2 的控制寄存器 .....	67
8.2.2	P1、P2 的原理图 .....	69
8.2.3	P1、P2 的中断控制功能 .....	70
8.3	通用端口 P3、P4 .....	71
8.3.1	P3、P4 的控制寄存器 .....	71
8.3.2	P3、P4 的原理图 .....	72
8.4	LCD 端口 .....	73
8.5	LCD 端口——定时器/端口比较器 .....	74
<b>第 9 章</b>	<b>通用定时器/端口模块</b>	
9.1	定时器/端口模块操作 .....	76
9.1.1	定时器/端口计数器 TPCNT1——8 位操作 .....	76
9.1.2	定时器/端口计数器 TPCNT2——8 位操作 .....	76
9.1.3	定时器/端口计数器——16 位操作 .....	76
9.2	定时器/端口寄存器 .....	77
9.3	定时器/端口 SFR 位 .....	80
9.4	定时器/端口在 A/D 中的应用 .....	81
9.4.1	R/D 转换原理 .....	81
9.4.2	分辨率高于 8 位的转换 .....	83
<b>第 10 章</b>	<b>定时器</b>	
10.1	Basic Timer1 .....	84
10.1.1	Basic Timer1 寄存器 .....	84
10.1.2	SFR 位 .....	86
10.1.3	Basic Timer1 的操作 .....	86
10.1.4	Basic Timer1 的操作——LCD 时钟信号 $f_{LCD}$ .....	87
10.2	8 位间隔定时器/计数器 .....	88
10.2.1	8 位定时器/计数器的操作 .....	88
10.2.2	8 位定时器/计数器的寄存器 .....	89
10.2.3	与 8 位定时器/计数器有关的 SFR 位 .....	91
10.2.4	8 位定时器/计数器在 UART 中的应用 .....	91
10.3	看门狗定时器 .....	101
10.3.1	看门狗定时器寄存器 .....	102
10.3.2	看门狗定时器的中断控制功能 .....	103
10.3.3	看门狗定时器操作 .....	104
10.4	8 位脉宽调制定时器 PWM .....	106
10.4.1	操作 .....	106
10.4.2	PWM 寄存器 .....	107

<b>第 11 章 Timer_A</b>	
11.1	Timer_A 的操作 ..... 110
11.1.1	定时器操作 ..... 110
11.1.2	捕获模式 ..... 116
11.1.3	比较模式 ..... 118
11.1.4	输出单元 ..... 118
11.2	Timer_A 的寄存器 ..... 120
11.2.1	Timer_A 控制寄存器 TACTL ..... 120
11.2.2	捕获/比较控制寄存器 CCTL ..... 121
11.2.3	Timer_A 中断向量寄存器 ..... 123
11.3	Timer_A 的应用 ..... 127
11.3.1	Timer_A 增计数模式应用 ..... 127
11.3.2	Timer_A 连续模式应用 ..... 128
11.3.3	Timer_A 增/减计数模式应用 ..... 130
11.3.4	Timer_A 软件捕获应用 ..... 131
11.3.5	Timer_A 处理异步串行通信协议 ..... 132
11.4	Timer_A 的特殊情况 ..... 133
11.4.1	CCR0 用做周期寄存器 ..... 133
11.4.2	定时器寄存器的启/停 ..... 134
11.4.3	输出单元 Unit0 ..... 135
<b>第 12 章 USART 外围接口——UART 模式</b>	
12.1	异步操作 ..... 137
12.1.1	异步帧格式 ..... 137
12.1.2	异步通信的波特率发生器 ..... 138
12.1.3	异步通信格式 ..... 140
12.1.4	线路空闲多处理机模式 ..... 140
12.1.5	地址位格式 ..... 142
12.2	中断与控制功能 ..... 143
12.2.1	USART 接收允许 ..... 143
12.2.2	USART 发送允许 ..... 144
12.2.3	USART 接收中断操作 ..... 144
12.2.4	USART 发送中断操作 ..... 145
12.3	控制与状态寄存器 ..... 146
12.3.1	USART 控制寄存器 UCTL ..... 146
12.3.2	发送控制寄存器 UTCIL ..... 148
12.3.3	接收控制寄存器 URCTL ..... 148
12.3.4	波特率选择和调制控制寄存器 ..... 150
12.3.5	USART 接收数据缓存 URXBUF ..... 151
12.3.6	USART 发送数据缓存 UTXBUF ..... 151

12.4	UART 模式——低功耗模式应用特性	151
12.4.1	由 UART 帧启动接收操作	151
12.4.2	时钟频率的充分利用与 UART 模式的波特率	153
12.4.3	节约 MSP430 资源的多处理机模式	154
12.5	波特率的计算	154
<b>第 13 章 USART 外围接口——SPI 模式</b>		
13.1	USART 的同步操作	158
13.1.1	SPI 模式中的主模式——MM=1,SYNC=1	160
13.1.2	SPI 模式中的从模式——MM=0,SYNC=1	161
13.2	中断与控制功能	162
13.2.1	USART 接收允许	162
13.2.2	USART 发送允许	163
13.2.3	USART 接收中断操作	164
13.2.4	USART 发送中断操作	165
13.3	控制与状态寄存器	166
13.3.1	USART 控制寄存器	166
13.3.2	发送控制寄存器 UTCTL	167
13.3.3	接收控制寄存器 URCTL	168
13.3.4	波特率选择和调制控制寄存器	169
13.3.5	USART 接收数据缓存 URXBUF	169
13.3.6	USART 发送数据缓存 UTXBUF	169
<b>第 14 章 液晶显示驱动</b>		
14.1	LCD 驱动基本原理	171
14.2	LCD 控制器/驱动器	174
14.2.1	LCD 控制器/驱动器功能	175
14.2.2	LCD 控制与模式寄存器	177
14.2.3	LCD 显示内存	179
14.2.4	LCD 操作软件例程	182
14.3	LCD 端口功能	186
14.4	LCD 与端口模式混合应用实例	187
<b>第 15 章 A/D 转换器</b>		
15.1	概述	189
15.2	A/D 转换操作	190
15.2.1	A/D 转换	190
15.2.2	A/D 中断	193
15.2.3	A/D 量程	193
15.2.4	A/D 电流源	194
15.2.5	A/D 输入端与多路切换	194
15.2.6	A/D 接地与降噪	196

---

15.2.7 A/D 输入与输出引脚 .....	196
15.3 A/D 控制寄存器 .....	197
<b>第 16 章 其他模块</b>	
16.1 晶体振荡器 .....	201
16.2 上电电路 .....	201
16.3 晶振缓冲输出 .....	202
<b>附录 A 外围模块地址分配</b>	
<b>附录 B 指令集描述</b>	
B1 指令汇总 .....	211
B2 指令格式 .....	212
B3 不增加 ROM 开销的指令模拟 .....	214
B4 指令说明 .....	216
B5 用几条指令模拟的宏指令 .....	244
<b>附录 C EPROM 编程</b>	
C1 EPROM 操作 .....	246
C2 快速编程算法 .....	247
C3 通过串行数据链路应用“JTAG”特性的 EPROM 模块编程 .....	248
C4 通过微控制器软件实现对 EPROM 模块编程 .....	248
<b>附录 D MSP430 系列单片机参数表</b>	
<b>附录 E MSP430 系列单片机产品编码</b>	
<b>附录 F MSP430 系列单片机封装形式</b>	

# 第 1 章 MSP430 系列

本章讨论 MSP430 系列单片机(或称微控制器)的特性及其控制模拟信号处理的特殊能力。系列的全部成员均为软件兼容。软件通过公共软件库、设计技术及开发工具,可以方便地在系列中的各型号间移植。

MSP 430 系列单片机的 CPU 设计成适合各种应用的 16 位结构。它采用“冯 - 纽曼结构”,因此, RAM、ROM 和全部外围模块都位于同一个地址空间内。

## 1.1 特性与功能

- 多达 64 KB 寻址空间,包含 ROM、RAM、闪存 RAM 和外围模块。将来计划扩大至 1 MB。
- 通过堆栈处理,中断和子程序调用层次无限制。
- 仅 3 种指令格式,全部为正交结构。
- 尽可能做到 1 字/指令。
- 源操作数有 7 种寻址模式。
- 目的操作数有 4 种寻址模式。
- 外部中断引脚: I/O 引脚具有中断能力。
- 中断优先级: 对同时发生的中断按优先级别处理。
- 嵌套中断结构: 中断程序可以被更高优先级的中断请求打断。
- 外围模块地址为存储器分配: 全部寄存器不占用 RAM 空间,均在模块内。
- 片上 USART: 发送与接收有各自的中断。
- 定时器中断可用于事件计数、时序发生、PWM 等。
- 看门狗功能。
- A/D 转换器(10 位或更高精度)有 8 个输入端,可作为恒流源。
- 具有 EPROM 型及 OTP 型。
- 具有 LCD 驱动电路。
- 用 FL1.1 和 32 768 Hz 晶振获得稳定的处理机时钟频率。
- 正交指令组简化了程序的开发: 所有指令可以用所有寻址模式。
- 已开发了 C-编译器。
- 模块设计思想: 所有模块采用存储器分配。

## 1.2 系统关键特性

- 超低电流消耗: CPUOff 和 OscOff 模式。
- 可在电压降至 2.5 V 情况下工作。



- IBM 兼容;
- DOS 5.0 或更高版本;
- Windows 3.1、3.11 或 Windows 95;
- 486 或更高性能处理机;
- 8 MB 存储器;
- 3.5 in 软盘驱动器;
- 硬盘有 5 MB 可用空间。

MSP430 系列单片机的特性汇总如表 1.1 所列。

表 1.1 MSP430 系列特性汇总

	MSP430x310	MSP430x320	MSP430x330
最大内部时钟频率/MHz	1.1 @3 V 3.3 @5 V	1.1 @3 V 2.2 @5 V	1.1 @3 V 2.2 @5 V
晶振频率/kHz	32.768	32.768	32.768
工作温度/°C	40 ~ +85	-40 ~ +85	-40 ~ +85
程序存储器			
MSP430Cxxx	4 KB/8 KB/12 KB ROM	8 KB ROM	24 KB ROM
MSP430Pxxx	8 KB OTP	16 KB OTP	32 KB OTP
MSP430Exxx	8 KB EPROM	16 KB EPROM	32 KB EPROM
存储器扩展	无	无	无
内部 RAM/B	256/512	256/512	1 024
数据 EEPROM	无	无	无
模块			
硬件乘法器	无	无	有
P0, 8 位, 支持中断	有	有	有
P1, 8 位, 支持中断			有
P2, 8 位, 支持中断			有
P3			有
P4			有
看门狗定时器	有	有	有
Basic Timer1/实时钟	有	有	有
8 位定时器/计数器	有	有	有
定时器/端口, 1×8 位	有	有	有
Timer_A, 16 位	无	无	无
同步通信	无	无	SPI 模式
异步通信	定时器/端口 + 软件	定时器/端口 + 软件	UART 模式或 定时器/端口 + 软件
LCD 驱动	23×4 段	21×4 段	30×4 段
ADC/恒流源	见定时器/端口	有	见定时器/端口
DAC	无	无	无
I/O 引脚	9	9	40
输入引脚	1	7	1
输出引脚	27	25	34
中断/复位			
外部中断	11	11	1 + 24
向量数	16	16	16
封装类型	56 QFP	64 SSOP	100 QFP

## 第 2 章 结构概述

本章说明 MSP430 系统的基本功能。

MSP430 系列器件包含以下主要功能：

- CPU；
- 程序存储器(ROM 或 EPROM)；
- 数据存储器(RAM 或 EEPROM)；
- 运行控制；
- 外围模块；
- 振荡器和倍频器。

MSP430 系列采用存储器-存储器结构,即用一个公共的空间对全部功能模块进行寻址;同时,用精简指令组对所有功能模块进行操作。MSP430 的系统结构如图 2.1 所示。

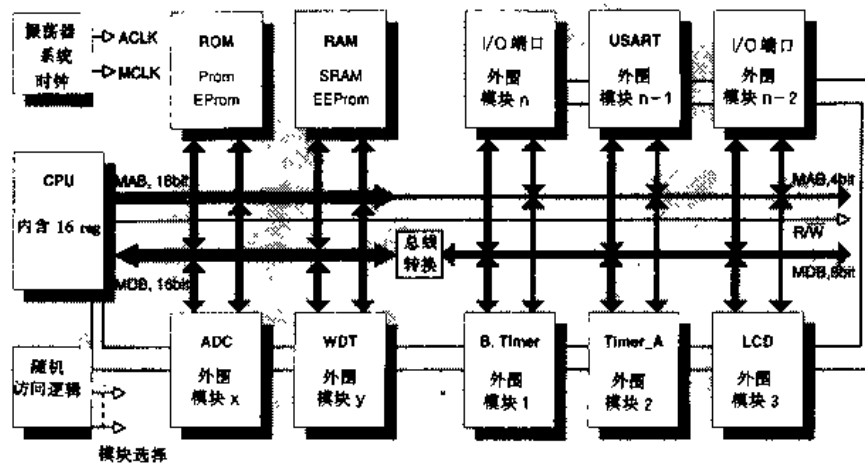


图 2.1 MSP430 系统结构

### 2.1 CPU

CPU 运行正交设计、对模块高度透明的精简指令集。它由一个 16 位 ALU、16 个寄存器和一套指令控制逻辑组成。其中 4 个寄存器有特殊用途,即程序计数器 PC、堆栈指针 SP、状态寄存器 SR 和,以及常数发生器 CG1、CG2。除了 R3/CG2 和 R2/CG1 外,所有寄存器都可作为通用寄存器来用所有指令操作。常数发生器用于指令执行时提供常数,而不是用于存储数据。对 CG1、CG2 访问的寻址模式可以区分常数的数据。PC、SR 和 SP 配合精简指令组所实现的控制,使应用开发可实现复杂的寻址模式和软件算法。





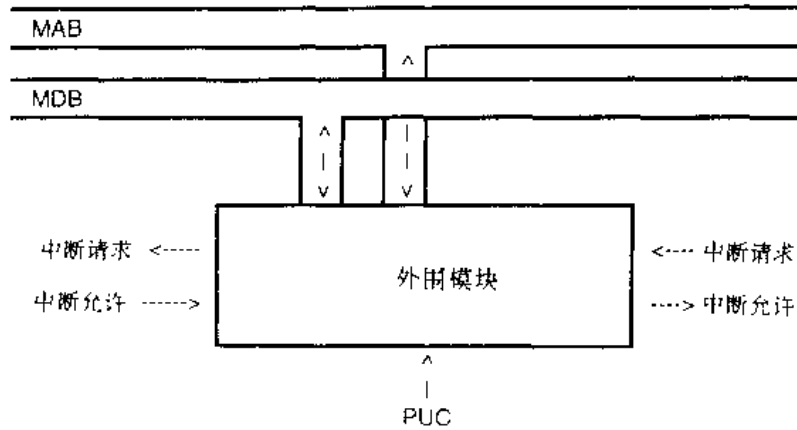


图 2.2 外围模块的连接总线

## 2.6 振荡器、倍频器和时钟发生器

振荡器是专门为通用的低功耗 32 768 Hz 时钟晶振设计的。除了晶体外接外,所有的模拟元件都集成在片内。

这一振荡器对于一些以低频工作的模块来说是直接信号源。对于 CPU 和其他模块,则将晶振频率用一个锁频环电路(FLL)进行倍频。FLL 在上电后以最低频率开始工作,并通过控制一个数控振荡器(DCO)来调整到适当的频率。

长时间工作的频率偏离受晶体和振荡器稳定性的限制。

供处理机工作的时钟发生器的频率固定在晶振的倍频上,并提供时钟信号 MCLK。

## 第3章 系统复位、中断和工作模式

### 3.1 系统复位和初始化

MSP430可以有4种复位来源：在VCC端加上供电电源、在 $\overline{\text{RST/NMI}}$ 端输入低电平信号、可编程看门狗定时器超时和在对WDTCTL寄存器写入时密钥不符。系统复位功能如图3.1所示。

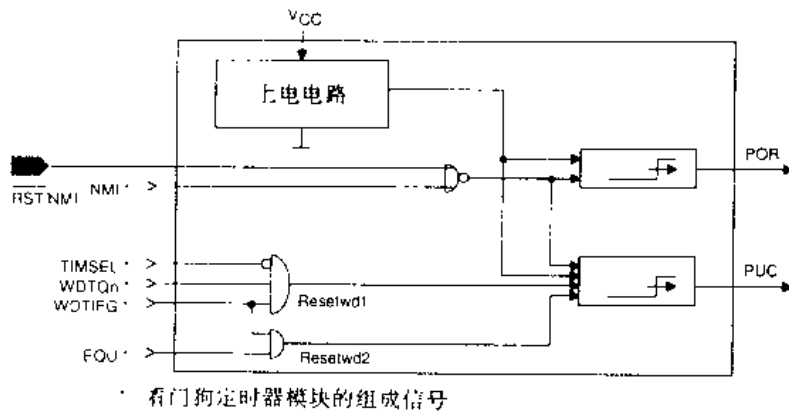


图 3.1 系统复位功能

发生复位后，程序查询各复位源的标志。程序能确定复位源，以执行适当的复位操作。MSP430 在发生 VCC 上电后开始硬件初始化，即

- 全部 I/O 引脚切换成输入状态。
- I/O 标志复位，细节见各外围模块的说明。
- 将复位向量 0FFFEh 中包含的地址加载入 PC 中。CPU 从上电清除(PUC)向量中包含的地址开始运行。
- 状态寄存器(SR)复位。
- 用户程序必须对除 PC 与 SR 外的全部寄存器进行初始化(如 SP、RAM 等)。
- 决定工作频率的系统时钟从 DCO 的最低频率开始工作。启动晶振时钟后，频率调整到目标值。

$\overline{\text{RST/NMI}}$  引脚在加载电压  $V_{CC}$  后设置成复位功能。引脚的复位功能一直保持到不选此功能为止。处于复位功能状态下，在 $\overline{\text{RST/NMI}}$ 引脚上拉低至 GND，然后释放，则 MSP430 按以下顺序开始工作。

- 将在复位向量地址 0FFFEh 中包含的地址加载入 PC。
- 在释放 $\overline{\text{RST/NMI}}$ 引脚后，CPU 从复位向量中所含的地址开始运行。
- 状态寄存器 SR 复位。

- 除 PC 与 SR 外,用户程序对全部寄存器进行初始化(如 SP、RAM 等)。
- 对外围模块中的寄存器进行处理。
- 决定工作频率的系统时钟从 DCO 的最低频率开始工作。启动晶振时钟后,频率调整到目标值。

### 3.2 中断系统结构

有如下 3 类中断:

- 系统复位;
- 非屏蔽中断;
- 可屏蔽中断。

引起系统复位的中断源如下:

- 加电源电压 @POR, PUC;
- $\overline{\text{RST}}/\text{NMI}$  引脚加低电平(选择复位模式) @POR, PUC;
- 看门狗定时器溢出(选择看门狗模式) @PUC;
- 看门狗定时器密钥不符(写 WDTCTL 时口令发生错误) @PUC。

非屏蔽中断由以下情况产生:

- $\overline{\text{RST}}/\text{NMI}$  引脚有上升沿信号(选择 NMI 模式);
- 振荡器故障。

#### 注意: 振荡器故障

振荡器故障只可由单独的允许位 OFIE 屏蔽。即使通用中断允许 GIE 复位也不能禁止。

可屏蔽中断源如下:

- 看门狗定时器溢出(选择定时器模式);
- 其他有中断能力的外围模块。

MPS430 中断优先级结构如图 3.2 所示。各模块的中断优先级由模块连接链决定,越接近 CPU/NMIRS 的模块,其中断优先级越高。

复位/非屏蔽中断模式选择如图 3.3 所示。因为作用在同一个引脚上,复位和非屏蔽中断只能交替工作。有关的控制位位于看门狗定时器控制寄存器(WDTCTL)内,也有口令保护。WDTCTL 的位定义如下:

WDTCTL (0120h)

7							0
HOLD	NMIES	NMI	TMSEL	CNTCL	SSEL	ISI	ISO
rw-0	rw-0	rw-0	rw-0	(w) 0	rw-0	rw-0	rw-0

对 WDTCTL 各位说明如下:

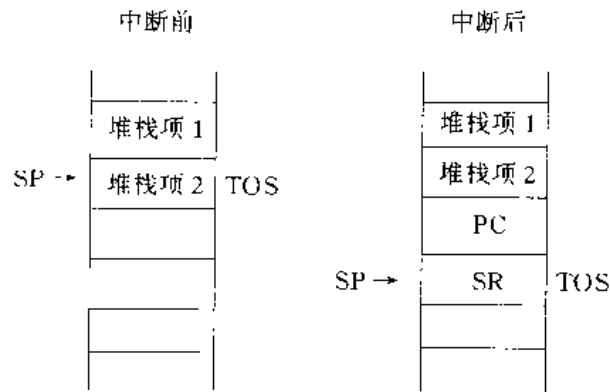
位 5: NMI 位用于选择  $\overline{\text{RST}}/\text{NMI}$  输入引脚的功能。PUC 后复位。

NMI = 0:  $\overline{\text{RST}}/\text{NMI}$  引脚按复位输入端工作。





- 在单一中断源标志中的中断请求标志位自动复位,多中断源标志仍保持置位以等待软件服务。
- 通用中断允许位 GIE 复位, CPUOff 位、OscOff 位和 SCG1 位\* 复位, 状态位 V、N、Z 和 C 复位。
- 将相应的中断向量值装入 PC, 程序从该地址继续执行中断处理。

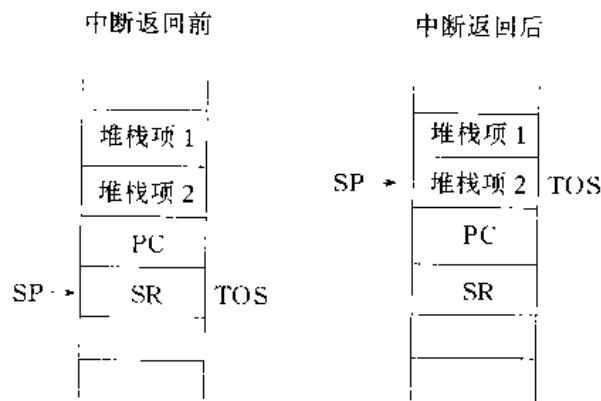


中断响应从接受中断请求开始到执行相应的中断服务程序的首条指令,持续 6 个周期。中断处理程序结束的指令是:

RETI

它执行以下内容:

- 将 SR 从堆栈中推出。被中断的程序回到与中断前完全相同的状态,包括 OscOff、CPUOff 和 GIE 位。GIE 位在中断期间的值被栈顶项取代。它总是“1”,因为在接受中断请求前它已预先置位。
- 将 PC 从堆栈中推出。



以 RETI 指令从中断服务程序返回,需 5 个周期。

如果 GIE 位在中断处理程序内置位,则允许中断请求嵌套。

含有 GIE 位的状态寄存器 SR/R2 位于 CPU 内,状态寄存器 SR 的位定义如下:

\* SCG0 不改变, FLL 环路控制保持原有工作状态。

15	9	8	7						0
留作将来增强用	V	SCG1	SCG0	OscOff	CPUOff	GIE	N	Z	C
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

除了 GIE 位,其他中断源可以被控制位单独或成组地允许/禁止。中断允许标志集中位于两个地址的 SFR 中。中断请求对程序流的控制可以方便地通过充分运用中断允许和屏蔽来调整。硬件只对被允许的中断源中的最高优先级者服务。

### 3.3.1 SFR 中的中断控制位

大多数中断控制位、中断标志和中断允许位集中在少数几个 SFR 中。这些 SFR 以字节形式位于低地址区。只能以字节指令访问 SFR。

地址	7	0
000Fh	未定义或未实现	
000Eh	未定义或未实现	
000Dh	未定义或未实现	
000Ch	未定义或未实现	
000Bh	未定义或未实现	
000Ah	未定义或未实现	
0009h	未定义或未实现	
0008h	未定义或未实现	
0007h	未定义或未实现	
0006h	未定义或未实现	
0005h	模块允许 2; ME2.x	
0004h	模块允许 1; ME1.x	
0003h	中断标志 2; IFG2.x	
0002h	中断标志 1; IFG1.x	
0001h	中断允许 2; IE2.x	
0000h	中断允许 1; IE1.x	

MSP430 各型号支持各个模块内的 SFR。除了 NMI 外,各模块中断源可以单独允许实现中断功能及操作。各配置位的完全软件控制使得应用软件可在中断允许屏蔽时激活系统。

#### 1. 中断允许 1、2

位	缩写	初始状态*	说明
IE1.0	WDTIE	复位	看门狗定时器允许,选中 Watchdog 模式时无效
IE1.1	OFIE	复位	振荡器故障中断允许
IE1.2	POIE.0	复位	针对 I/O P0.0
IE1.3	POIE.1	复位	针对 I/O P0.1 或 8 位定时器/计数器
IE1.4		复位	保留,未定义
IE1.5		复位	保留,未定义
IE1.6		复位	保留,未定义

\* 初始状态是指发生 PUC 后的状态。对于 WDTIFG 见有关说明。



IE1.7		复位	保留,未定义
IE2.0	URXIE	复位	USART 接收中断允许
IE2.1	UTXRIE	复位	USART 发送中断允许
IE2.2	ADIE/TPIE	复位	ADC 或定时器/端口中断允许(适合 320 型)
IE2.3	TPIE	复位	定时器/端口(适合 310、330 型)
IE2.4		复位	保留,未定义
IE2.5		复位	保留,未定义
IE2.6		复位	保留,未定义
IE2.7	BTIE	复位	Basic Timer 中断允许

## 2. 中断标志寄存器 1、2

位	缩写	初始状态	说明
IFG1.0	WDTIFG	不变	溢出或密钥不符时置位
		复位	VCC 上电或 $\overline{\text{RST}}$ /NMI 引脚有复位条件
IFG1.1	OFIFG	置位	振荡器发生故障时置位
IFG1.2	POIFG.0	复位	针对 I/O P0.0
IFG1.3	POIFG.1	复位	针对 I/O P0.1 或 8 位定时器/计数器
IFG1.4	NMIFG	复位	$\overline{\text{RST}}$ /NMI 引脚信号
IFG1.5			保留,未定义
IFG1.6			保留,未定义
IFG1.7			保留,未定义
IFG2.0	URXIFG		USART 接收标志
IFG2.1	UTXIFG		USART 发送就绪
IFG2.2	ADIFG	复位	ADC 转换结束时置位
IFG2.3			保留,未定义
IFG2.4			保留,未定义
IFG2.5			保留,未定义
IFG2.6			保留,未定义
IFG2.7	BTIFG	不变	Basic Timer 标志

## 3. 模块允许 1、2

位	缩写	初始状态	说明
ME1.0			保留,未定义
ME1.1			保留,未定义
ME1.2			保留,未定义
ME1.3			保留,未定义
ME1.4			保留,未定义
ME1.5			保留,未定义
ME1.6			保留,未定义
ME1.7			保留,未定义
ME2.0	URXE		USART 接收允许
ME2.1	UTXE		USART 发送允许
ME2.2			保留,未定义
ME2.3			保留,未定义



P0 有 6 个控制 I/O 引脚的寄存器:

- 输入寄存器。
- 输出寄存器。
- 方向寄存器。
- 中断标志。 有 6 个标志, 含有用做中断输入的 I/O 引脚的信息。  
0: 无等待响应的中断。  
1: 因引脚信号跳变产生等待响应的中断。  
写入“0”将中断标志复位。  
写入“1”将中断标志置位。器件以发生中断事件时的方式来处理。
- 中断沿选择。 寄存器对每一 I/O 引脚有一中断触发跳变选择位。  
0: 发生低/高跳变时中断标志置位。  
1: 发生高/低跳变时中断标志置位。
- 中断允许。 寄存器对引脚 P0.2 ~ P0.7 有 6 位分别允许中断事件触发中断请求。  
0: 禁止中断请求。  
1: 允许中断请求。

I/O 引脚 P0.2 ~ P0.7 中断处理编程举例:

```

; I/O 引脚 P0.2 ~ P0.7 处理中断
;
IOINTR    PUSH    R5                ; 保存 R5
          MOV.B   &P0IFG, R5       ; 读中断标志
          BIC.B   R5, &P0IFG       ; 用读入数据清除状态标志
          ; 其他各位不清除!
          EINT    ; 允许中断嵌套
;
; R5 包含引起中断的 I/O 引脚的信息, 处理由此开始
;
...
...
POP       R5                ; 处理完成, 恢复 R5
RETI     ; 从中断返回
...
...
; 中断向量表定义
.sect    "IO27_vec", 0FFE0h
.word   IOINTR                ; ROM 中 I/O 引脚(2 ~ 7)向量
;
.sect    "RST_vec", 0FFFEh    ; 中断向量
.word   RESET

```

## 2. 端口 P1 和 P2

P1 与 P2 完全相同。对 P1 和 P2 模块各分配一个单独的向量。引脚 P1.0 ~ P1.7 和 P2.0 ~ P2.7 可用做中断源。向量包含因中断事件引发而被装入 PC 的存储器地址。

P1 和 P2 分别有如下 7 个寄存器,用于控制 I/O 引脚。

- 输入寄存器。
- 输出寄存器。
- 方向寄存器。
- 中断标志。 有 8 个标志,含有用做中断输入的 I/O 引脚的信息。  
0: 无等待响应的中断。  
1: 因引脚信号跳变产生等待响应的中断。  
写入“0”将中断标志复位。  
写入“1”将中断标志置位。器件以发生中断事件时的方式来处理。
- 中断沿选择。 寄存器对每一 I/O 引脚有一中断触发跳变选择位。  
0: 发生低/高跳变时中断标志置位。  
1: 发生高/低跳变时中断标志置位。
- 中断允许。 对引脚 PX.0 ~ PX.7,寄存器有 8 位,分别允许中断事件触发中断请求。  
0: 禁止中断请求。  
1: 允许中断请求。
- 功能选择寄存器。

#### 注意: 数字端口 P0、P1 和 P2 的中断处理

只有跳变(不是静态电平)才引起中断。中断程序必须对多源中断标志复位。所谓多源中断标志是指 P0IFG.2 ~ P0IFG.7、P1IFG.0 ~ P1IFG.7 和 P2IFG.0 ~ P2IFG.7。而单源标志 P0IFG.0 和 P0IFG.1 在得到服务时自动复位。当 RETI 执行后仍有中断标志置位(因引脚信号跳变发生在中断服务期间)时,在 RETI 指令执行完成后会再次发生中断。这保证了软件能发现每一次跳变。

### 3.4 工作模式

MSP430 的工作模式以先进的方式支持超低功耗的各种要求。这是通过各模块的智能化运行管理和 CPU 的状态组合而得到的。一个中断事件可将系统从各种工作模式中唤醒,而 RETI 指令又使运行返回到中断事件发生前的工作模式,如图 3.4 所示。

MSP430 系列为超低功耗应用开发出了不同功耗的工作模式。

用 CMOS 技术设计的超低功耗系统有 3 个主要目的:

- 解决运行速度和数据流量与低功耗设计的冲突;
- 将各模块的电流消耗降至最低;
- 将活动状态限制至最低要求。

有 6 种工作模式,可由软件组合。

1) 活动模式——AM

各种活动的外围模块的组合。

2) 低功耗模式 0——LPM0

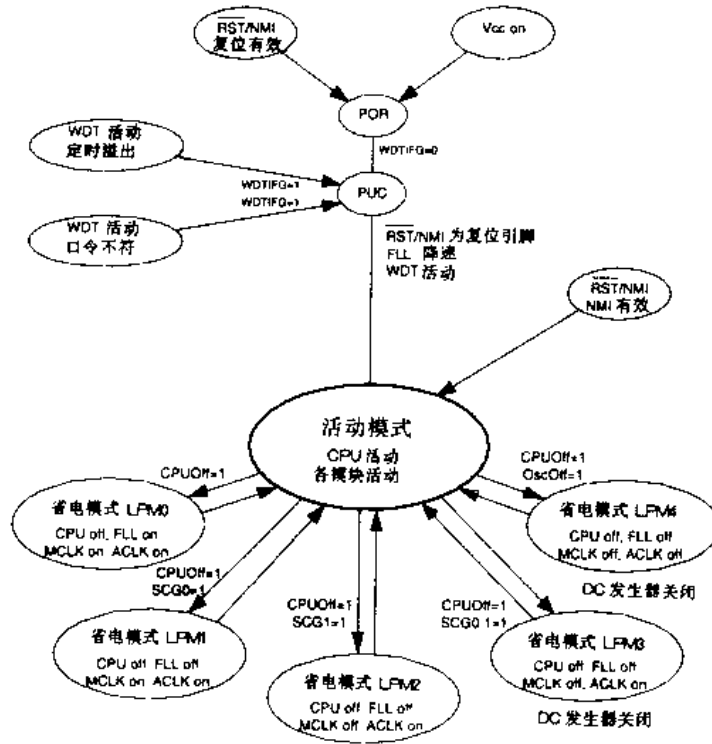
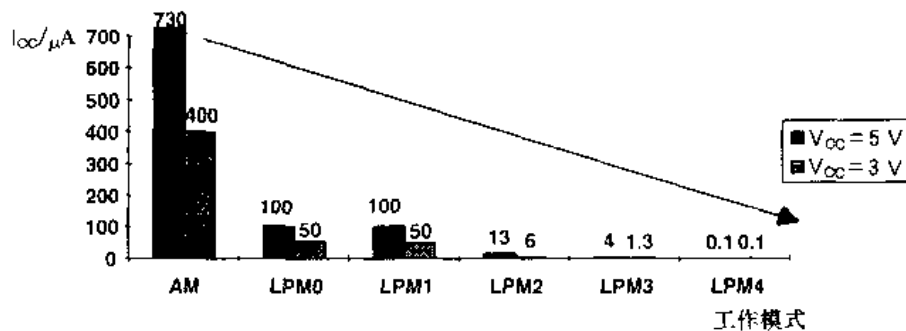


图 3.4 MSP430 工作模式状态图

- CPUOff 置位, CPU 停止活动。
  - CPUOff 不会使外围模块停止运行。
  - ACLK 和 MCLK 信号活动, MCLK 的锁频环控制活动。
  - @ SCG1 = 0, SCG0 = 0, OscOff = 0, CPUOff = 1。
- 3) 低功耗模式 1——LPM1
- CPUOff 置位, CPU 停止活动。
  - CPUOff 不会使外围模块停止运行。
  - MCLK 的锁频环控制停止。
  - ACLK 和 MCLK 信号活动。
  - @ SCG1 = 0, SCG0 = 1, OscOff = 0, CPUOff = 1。
- 4) 低功耗模式 2——LPM2
- CPUOff 置位, CPU 停止活动。
  - CPUOff 不会使外围模块停止运行。
  - MCLK 的锁频环控制停止。
  - ACLK 信号活动。
  - @ SCG1 = 1, SCG0 = 0, OscOff = 0, CPUOff = 1。
- 5) 低功耗模式 3——LPM3
- CPUOff 置位, CPU 停止活动。
  - CPUOff 不会使外围模块停止运行。
  - MCLK 的锁频环控制和 MCLK 信号停止。

- DCO 的 DC 发生器(到 MCLK 发生器)关闭。
  - ACLK 信号活动。
  - @ SCG1 = 1, SCG0 = 1, OscOff = 0, CPUOff = 1。
- 6) 低功耗模式 4——LPM4
- CPUOff 置位, CPU 停止活动。
  - CPUOff 不会使外围模块停止运行。
  - MCLK 的锁频环控制停止。
  - DCO 的 DC 发生器(到 MCLK 发生器)关闭。
  - ACLK 信号停止, 晶振停止。
  - @ SCG1 = X, SCG0 = X, OscOff = 1, CPUOff = 1。

不同工作模式对应的典型的电流消耗如图 3.5 所示。



资料来源: TI Data sheet SLASE07, January 1996(MSP430C312/314)

图 3.5 不同模式对应的典型电流消耗

对于各外围模块和 CPU 的活动状态,通过选择适当的低功耗模式、选择停止外围模块的部分运行,或者将它们整个停止等方法来控制。用根据应用需求设计的软件,有多种途径来组成最低电流消耗的系统。在 SFR 中含有能允许和禁止外围模块运行功能的控制位。一个例子是 LCD 外围模块中的模拟电压发生器的允许/禁止,它通过一个寄存器位来打开或关闭。影响电流消耗并支持从低功耗模式快速启动的最常用的控制位位于状态寄存器 SR 中。有 4 位用于控制 CPU 和系统时钟发生器。

对于支持活动模式 AM 的间歇运行和限制全速工作模式的时间周期,这 4 个控制位(即 CPUOff、OscOff、SCG0 和 SCG1)是非常有用的。工作模式控制位包含在 SR 中的最大优点是,在中断服务期间它作为运行状态保护在堆栈中。只要 SR 信息不改变,在 RETI 指令执行后,处理机能继续保持在中断事件发生前的工作模式。通过处理保存在堆栈中的数据或改变 SP,也可以选择另一个程序流向。对堆栈和 SP 的指令组访问的方便使得设计者可以分别优化程序结构。

### 3.5 低功耗模式

由 SFR 中的各模块允许位来确定各自功耗控制器工作状态的配置。由用户程序定义外围模块的活动或停止。通过可禁止的各部分的漏电流,被禁止模块的电流消耗被降低。模块

中惟一活动的是完成使受控模块进入允许状态或传递中断请求(例如发生外部硬件中断)给 CPU 的部分。

各模块允许的选择有多达 5 种可能的省电模式: CPU 关闭模式(LPM0)和 4 种系统时钟发生器的运行组合。当控制位 CPUOff、SCG1、SCG0、OscOff(位于 SR 中)置位时,就会进入上述状态。系统时钟发生器模块因 SCG1、SCG0 和 OscOff 位的状态而从 4 种省电模式中被重新激活。这将在系统时钟发生器部分详细说明。

#### 1. 进入中断程序

如果一个允许的中断唤醒 MSP430,那么就会进入中断程序并开始处理。

- SR 和 PC 保存入堆栈,保存了中断事件发生时的现场。
- 随后 SR 中的工作模式控制位 OscOff、SCG1 和 CPUOff 自动被复位。

#### 2. 从中断程序返回

有两种从中断服务程序返回的方法:

- 置位低功耗模式位后返回。从中断返回后,PC 指向下一条指令。由于被恢复后的低功耗模式禁止了 CPU 的活动,PC 所指的指令不执行。
- 复位低功耗模式位后返回。从中断返回后,程序从对 SR 中的 OscOff 或 CPUOff 置位的指令之后的地址继续执行。

### 3.5.1 低功耗模式 0 和模式 1

对 SR 中的 CPUOff 置位可选择进入低功耗模式 0 或 1。置位后 CPU 立即停止运行,系统内核的常规操作停止。CPU 的操作暂停,直至有任一中断请求或复位发生。所有内部总线停止活动。系统时钟发生器的继续工作和时钟信号 MCLK 及 ACLK 的活动取决于 SR 中的其他 3 位,即 SCG0、SCG1 和 OscOff。SCG1 定义 MCLK 运行于 ACLK 的倍数,或按 DCO 最近一次的控制信号运行。

被允许并得到 MCLK 或 ACLK 信号的外围模块处于活动状态。I/O 端口的全部引脚以及 RAM 和寄存器保持不变。所有被允许的中断事件可以从此状态唤醒程序。

```

; == = Main program flow with switch to CPUOff Mode == = == = == =
;
    BIS    #18h, SR           ; 进入 LPM0 并允许中断控制位 GIE
                                ; PC 在执行这一指令时增加,指向程序的后续步骤
    ...                       ; 如果中断服务期间 CPUOff 复位,则程序继续
; == = Interrupt service routine == = == = == =
...
...
    RETI                      ; RETI 恢复中断前的 CPU 状态
                                ; SR 中的 GIE、CPUOff、OscOff、SCG1 和 SCG0 位于状态寄
                                ; 存器中,该状态寄存器在执行从中断返回时恢复

```

### 3.5.2 低功耗模式 2 和模式 3

对 SR 中的 CPUOff 和 SCG1 置位可选择进入低功耗模式 2 或 3。置位后 CPU 和 MCLK 立即停止运行。它们暂停直至有任一中断请求或复位发生。所有内部总线停止活动。SCG1

定义 MCLK 在系统回到活动模式时运行于 ACLK 的倍数,或按 DCO 最近一次的控制信号运行。

被允许并得到 ACLK 信号的外围模块处于活动状态。工作时需要 MCLK 信号的外围模块会因为 MCLK 信号活动的停止而停止。I/O 端口的全部引脚以及 RAM 和寄存器保持不变。所有被允许的、不依赖于 MCLK 的中断事件可以从此状态唤醒。

### 3.5.3 低功耗模式 4

全部活动部件停止,只有 RAM、端口和寄存器的内容保持。只能由被允许的外部中断唤醒。

在启动 LPM4 前,软件要考虑在这一低功耗模式期间系统需要的条件。最重要的两点是针对运行环境的,即对 DCO 和周期性操作的影响。运行环境定义的频率合成应被保持或校正。校正在周围环境需要系统对频率作大的改变时可能发生。当存在周期性操作应用时,应该考虑锁频环可能失控,余留的时间片不足以将锁频环保持在校正操作范围之内。

以下例子说明低功耗模式 4 的进入(OscOff)。

```
BIS #B8h, SR      ; 进入 LPM4, 并允许 GIE
                  ; CPU 切换到 LPM 模式。DCO 操作被允许
                  ; 在中断程序期间 LPM4 将结束时, DCO 运行已准备好
...               ; 如果在中断程序中 OscOff 复位, 则程序继续
...               ; 否则, 仍保持在 OscOff 模式
```

## 3.6 低功耗应用要点

当电流消耗是系统应用的重要指标时,应该考虑一些常规原则:

- 将不用的 FETI 输入端连接到 VSS;
- 关闭 LCD 及模块,可能时包括外部的模拟电压发生器;
- JTAG 端口 TMS、TCK 和 TDI 不要连接到 VSS;
- CMOS 输入端不能有浮空的节点,将所有输入端接适当的电平;
- 选择尽可能低的运行频率,既针对内核,同样也针对各外围模块;
- 如果用了 LCD,则选择尽可能弱的驱动能力,或者将它关闭;
- 充分利用中断驱动软件的特性——程序能快速地启动运行。



## 第 4 章 存储器组织

MSP430 系列的存储器空间采用“冯-纽曼结构”，代码存储器(ROM、EPROM、RAM)和数据存储器(RAM、EEPROM、ROM)由同一组地址及数据总线放在一个地址空间中。图 4.1 所示为总的存储地址空间示意图。

物理上完全分离的存储区域，例如内部 ROM、RAM、SFR，外围模块，以及外接的存储器，都位于一个公共地址空间中。总的寻址空间在小存储模式时为 64 KB，在大存储模式时为 1 MB。小存储模式采用线性寻址空间；在大存储模式时，代码访问的地址空间为 16 个 64 KB 段，数据访问的地址空间为 16 个 64 KB 页。

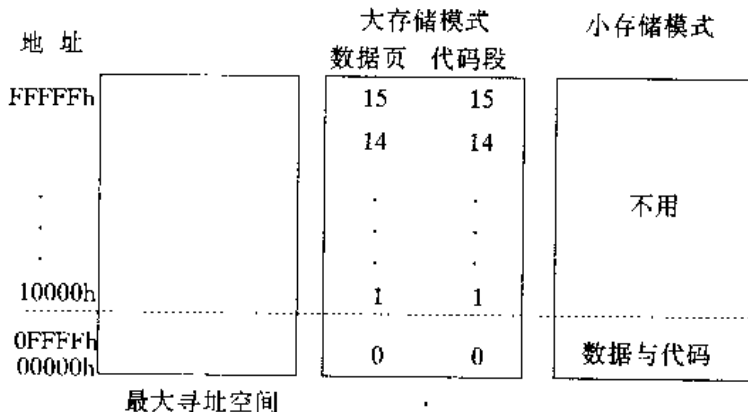


图 4.1 总的存储器地址空间

存储器构成为 64 KB 或更少时采用小存储模式，安排在最低的 64 KB。这时不必考虑寻址时的代码段和数据页。

小存储模式的空间结构和数据宽度如图 4.2 所示。

地址	7	0	功能	寻址
0FFFFh	中断向量表		ROM	字/字节
0FFE0h				
0FFDFh				
	程序存储器 跳转控制表 数据表等		ROM	字/字节
0200h	数据存储器		RAM	字/字节
01FFh	16 位外围模块		Timer、ADC 等	字
0100h				
0FFh	8 位外围模块		I/O、LCD、 定时器/端口等	字节
010h				
0Fh	特殊功能寄存器		SFR	字节
00h				

图 4.2 基本存储空间中的存储器分配

数据总线可以是 16 位或 8 位宽度。对于可以以字访问的模块,其数据宽度始终是 16 位。而其他的 8 位模块只能用字节指令访问。程序存储器(ROM)和数据存储器(RAM)既可用字节指令也可用字指令访问。部分外围模块以 16 位或 8 位实现。访问它们时必须用适当的字指令或字节指令。

许多外围模块与 CPU 的连接(见图 4.3)是通过 8 位存储器数据总线(MDB)、存储器地址总线(MAB)的低 5 位、2 个模块允许信号、2 条中断控制/请求线和上电信号实现的。访问这些模块总是用字节指令。其他 16 位外围模块的连接通过 16 位 MDB,完全支持字处理,访问必须用字指令。

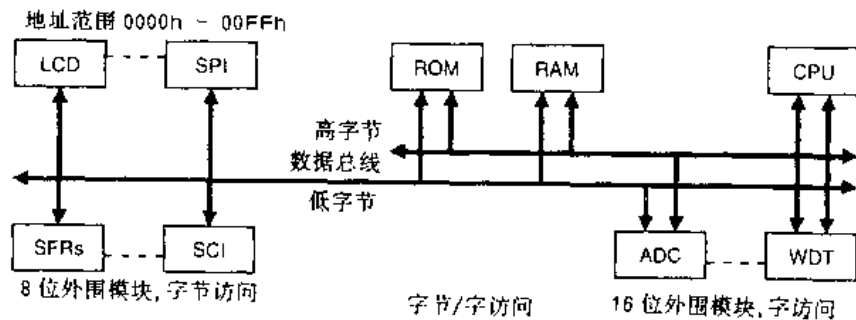


图 4.3 外围模块与 CPU 连接示意图

## 4.1 存储器中的数据

字节数据可以定位在偶地址或奇地址。字数据定位在偶地址,其低字节在偶地址,高字节在下一个奇地址。字节结构存储器中的位、字节和字如图 4.4 所示。

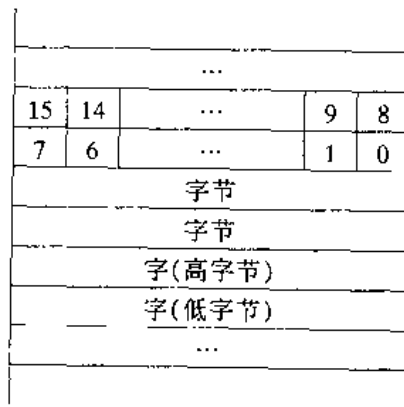


图 4.4 字节结构存储器中的位、字节和字

## 4.2 片内 ROM 组织

片内 ROM 可能有各种体积,直至 64 KB。公共地址空间由 SFR、外围模块寄存器、数据和代码存储器共享。SFR 和外围模块安排在 0000h ~ 01FFh 的空间中。余下的地址空间 0200h ~ FFFFh 由数据和代码存储器分享。

不同体积 ROM 的开始地址都在同一个地址 0FFFFh。中断向量中最高优先级位于最高的 ROM 字地址。PC 计数,也即指令执行顺序,则按另一方向,即由低地址到高地址。根据执行指令的寻址模式,PC 将在执行时增加 2、4 或 6;但这不包括程序流向控制指令,即跳转、条件跳转和子程序调用等。图 4.5 所示为 ROM 结构示意图。

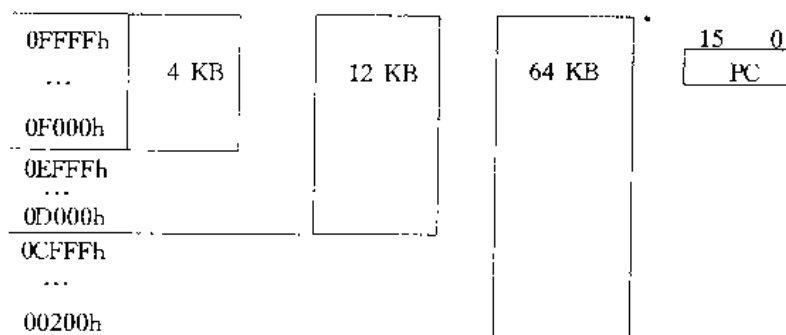


图 4.5 ROM 结构

中断向量和上电向量在从地址 0FFFFh 开始的 ROM 中。向量中含有相应中断处理程序的 16 位入口地址。

#### 4.2.1 ROM 表的处理

MSP430 的结构允许 ROM 存放大的数表。可以用所有的字和字节指令访问这些表。这一点为提高编程的灵活性和节省 ROM 空间带来各种好处。例如:

- 在 ROM 中存放用于显示字符转换的输出可编程逻辑阵列(Output - PLA);
- 可根据需要设计 OPLA 的项数(项数无限制);
- OTP 型自动包含了 OPLA 编程能力;
- 计算表访问(例如棒图显示);
- 表处理支持的程序流向控制。

表处理是十分重要的特性,它能导致非常快速清晰的编程风格。特别是对于传感器应用,为了数据线性化和补偿,将传感器数据存入表中作表处理是一个好方法。

#### 4.2.2 计算分支跳转和子程序调用

用标准指令实现计算分支跳转和子程序调用是可能的。CALL 和 BR 指令与其他指令采用同样的寻址模式(见编程举例)。

计算分支跳转和子程序调用最理想的寻址模式是间接-间接。充分利用这一编程特点可以得到与常规 8 位和 16 位微控制器不同的程序结构。利用软件的状态处理可以方便地处理大量的子程序调用,而不是利用“标志”来控制程序流向。

计算分支跳转和子程序调用在一个 64 KB 代码段内有效。

### 4.3 RAM 与外围模块组织

利用适当的指令后缀,对整个 RAM 可按字节或字访问。外围模块安排在两个不同的地

址空间。

- SFR 硬件是面向字节的, 安排在地址 00h ~ 0Fh 中。
- 硬件面向字节的外围模块安排在地址 010h ~ 0FFh 中。
- 硬件面向字的外围模块安排在地址 0100h ~ 01FFh 中。

### 4.3.1 RAM

RAM 可以用做代码存储器和数据存储器。而代码访问总是对偶地址的, 指令助记符的后缀定义了访问的数据是字还是字节。

举例:

```
ADD.B    &TCDATA, TCSUM_L      ;字节访问
ADDC.B   TCSUM_H                ;字节访问
ADD      R5, SUM_A  ≡ ADD.W   R5, SUM_A  ;字访问
ADDC     SUM_B      ≡ ADDC.W  SUM_A      ;字访问
```

一个字由高字节(位 15 ~ 8)和低字节(位 7 ~ 0)组成, 总是对准偶地址。字节和字操作如图 4.6 所示。

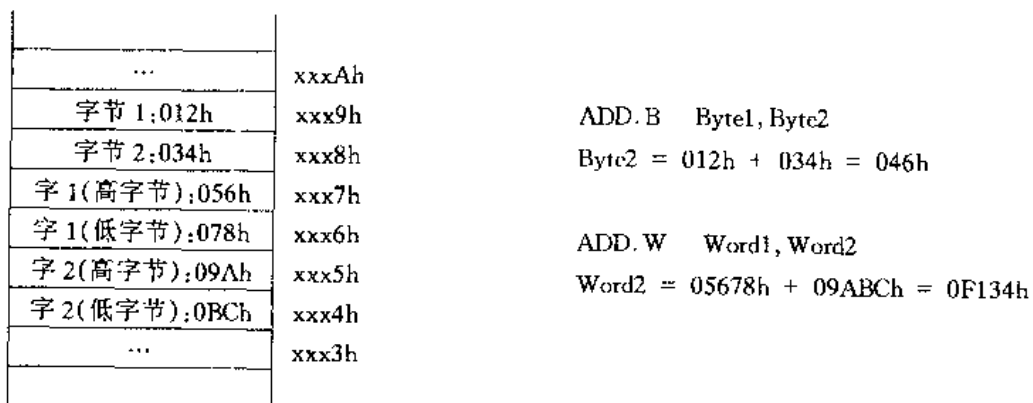


图 4.6 字节和字操作

对堆栈和 PC 全部为字操作, 并且对准偶地址。

字 - 字和字节 - 字节操作同样能得到准确的操作数结果和状态位设置。

字 - 字操作	字节 - 字节操作
R5 = 0F28Eh	R5 = 0223h
EDE .EQU 0212h	EDE .EQU 0202h
Mem(0F28Eh) = 0FFFEh	Mem(0223h) = 05Fh
Mem(0212h) = 00112h	Mem(0202h) = 043h
ADD @R5, &EDE	ADD.B @R5, &EDE
Mem(0212h) = 00110h	Mem(0202h) = 0A2h
C = 1, Z = 0, N = 0	C = 0, Z = 0, N = 1



地址	7	0	功能	寻址
01FFh	16 位外围模块		Timer、ADC 等	字
0100h				
0FFh	8 位外围模块		I/O、LCD、8 位定时器/端口等	字节
010h				
0Fh	特殊功能寄存器		SFR	字节
00h				

图 4.7 外围模块结构

地址	说明
1F0h~1FFh	保留
1E0h~1EFh	保留
1D0h~1DFh	保留
1C0h~1CFh	保留
1B0h~1BFh	保留
1A0h~1AFh	保留
190h~19Fh	保留
180h~18Fh	保留
170h~17Fh	Timer_A
160h~16Fh	Timer_A
150h~15Fh	保留
140h~14Fh	保留
130h~13Fh	乘法器
120h~12Fh	看门狗定时器
110h~11Fh	A/D
100h~10Fh	保留

图 4.8 外围模块地址分配——字模块

## 2. 字节模块

字节模块是经 MDB 的低 8 位连接的模块。总是以字节格式来访问字节模块。在写操作时,由于硬件的原因模块只取了低字节。

字节指令对字节模块的操作无任何限制。用字指令对字节外围模块作读访问会在高字节产生不可预知的结果。对于写入字节模块的字数据,其低字节写入相应的模块寄存器,而高字节可忽略不计。

外围模块的地址空间被组织成 16 个帧,如图 4.9 所示。



MSP430 系列各型号支持各自外围模块的 SFR 的功能。每个模块可以单独允许,以实现中断功能和各种操作。配置位的完全软件控制使得应用软件在中断允许屏蔽时可对系统需求作出响应。

系统的功耗受当前允许的模块数和模块功能的影响。禁止一个活动模块会减少功耗。有两个部分是不能禁止的,即 ROM 和 RAM。处理机内核可以切换到禁止模式,即 CPUOff 模式。这时所有的内部功能都被禁止了,因为这时 CPU 和总线停止活动。



## 第 5 章 16 位 CPU

PC 和工作寄存器的宽度相同带来了各种新的特性。例如,有 7 种寻址模式。

MSP430 采用“冯-纽曼结构”,RAM、ROM 在同一地址空间中,使用一组地址数据总线。

### 5.1 CPU 寄存器

14 个 16 位寄存器(R0、R1,以及 R4 ~ R15)用于存放数据和地址。这些寄存器位于 CPU 内。利用它们能寻址达 64 KB(ROM、RAM、EEPROM、外围模块,……),而不必分段。

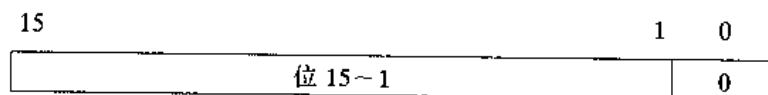
全部 CPU 寄存器如表 5.1 所列。其中对有特殊用途的寄存器作了标明。对于寄存器 R0、R1、R2、R3,由于它们的特殊功能在通用性上有限制,将在以后说明。

表 5.1 不同功能的寄存器

功能	简写
程序计数器 PC	R0
堆栈指针 SP	R1
状态寄存器 SR/常数发生器 CG1	R2
常数发生器 CG2	R3
工作寄存器 R4	R4
工作寄存器 R5	R5
⋮	⋮
工作寄存器 R13	R13
工作寄存器 R14	R14
工作寄存器 R15	R15

#### 5.1.1 程序计数器 PC

16 位程序计数器 PC 决定将要执行的下一条指令。每条指令占据偶数字节,即 2 B、4 B 或 6 B。指令的访问发生在字边界,因此,PC 是对准偶地址的。PC 在取指令周期中增加 2,以指向紧接当前执行指令后的字地址。利用紧跟当前指令的字信息可能有 4 种寻址模式(立即寻址、变址寻址、绝对寻址和符号寻址模式)。程序计数器 PC 的结构如下:



#### 5.1.2 系统堆栈指针 SP

系统堆栈指针 SP 总是对准偶地址,因此,在中断服务中是以字数据访问堆栈。系统堆栈指针 SP 用于保存子程序调用和中断服务时的返回地址。它采用先减后加的方案。这一方案的好处是栈顶(TOS)数据项可用。SP 可被用户软件所用(PUSH 和 POP 指令),但是必须记



PUSH R12 ; R12 存入 0xxxh-6, SP 指向同一地址  
 POP R12 ; 从 0xxxh-6 恢复数据入 R12, SP 指向 0xxxh-4  
 MOV @SP+, R5 ; 数据项 I3 送 R5 (从堆栈推出), 与 POP 指令相同  
 PUSH #1  
 POP R8

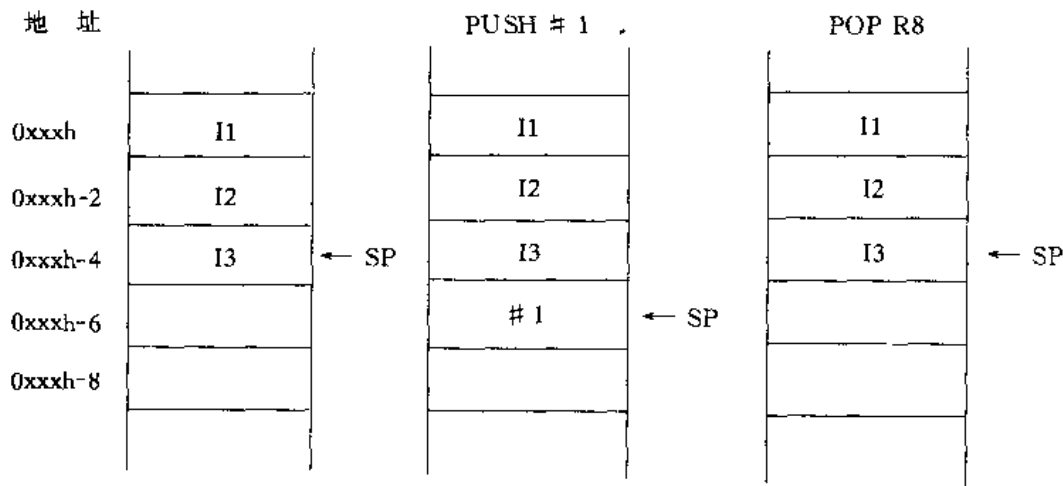


图 5.1 堆栈使用

### 5.1.3 状态寄存器 SR

状态寄存器 SR 含有 CPU 的各状态位:

- V: 溢出位。
- SCG1: 系统时钟发生器控制位 1。
- SCG0: 系统时钟发生器控制位 0。
- OscOff: 晶振关闭位。
- CPUOff: CPU 关闭位。
- GIE: 通用中断允许位。
- N: 负数位。
- Z: 零位。
- C: 进位位。

状态寄存器 SR 的位定义如下:

15	9	8	7						0
留作未来增强用	V	SCG1	SCG0	OscOff	CPUOff	GIE	N	Z	C
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

对 SR 的状态位说明如下:

- V: 当算术运算结果超出有符号数范围时置位, 对字节和字格式数均有效。  
 ADD(.B), ADDC(.B) 当出现以下情况时置位, 即  
 正数 + 正数 = 负数

- 负数 + 负数 = 正数  
其他情况下均复位。
- SUB(.B), SUBC(.B), CMP(.B) 当出现以下情况时置位,即  
正数 - 负数 = 负数  
负数 - 正数 = 正数  
其他情况下均复位。
- SCG1、SCG0: 控制系统时钟发生器的 4 种活动状态,因此它们将影响处理机运行。
- OscOff: 置位使晶振进入关闭模式,即除 RAM 内容、端口和寄存器保持活动外,全部活动部件停止。只可能在 GIE 置位条件下由外部中断或由 NMI 唤醒。如果不同时对 CPUOff 置位,则不能对它置位。
- CPUOff: 置位使 CPU 进入关闭模式,即除 RAM、端口、寄存器和特别允许的外围模块(如 Basic Timer、UART、…)保持活动外,全部活动停止。所有允许的中断可以唤醒这一状态。
- GIE: 置位使被允许的中断可以处理,复位禁止所有中断。GIE 位由中断复位,由 RETI 指令恢复。它也可用指令改变。
- N: 运算结果为负时置位。  
字运算: 将 N 位设置为运算结果的位 15。  
字节运算: 将 N 位设置为运算结果的位 7。
- Z: 当运算结果为 0 时置位,不为 0 时复位。
- C: 当运算结果产生进位时置位,无进位时复位。  
字运算: 进位是指字运算结果发生进位。  
字节运算: 进位是指字节运算结果发生进位。  
某些指令以 Z 位的非来修改 C 位。

**注意: 状态位 V、N、Z 和 C**

状态位 V、N、Z 和 C 只在某些指令执行时改变。请看指令组详细说明以及 MSP 软件用户指南。

### 5.1.4 常数发生寄存器 CG1 和 CG2

经常使用的常数可由常数发生器 R2 和 R3 产生,而不必占用一个 16 位字。所用常数的数值由寻址位 As 来定义,如表 5.2 所列。

表 5.2 常数发生器 CG1、CG2 的值

寄存器	As	常数	说明
R2	00	-----	寄存器模式
R2	01	(0)	绝对寻址模式
R2	10	00004h	+4, 位处理
R2	11	00008h	+8, 位处理
R3	00	00000h	0, 字处理
R3	01	00001h	+1
R3	10	00002h	+2, 位处理
R3	11	0FFFFh	-1, 字处理

使用这种方法产生常数的优点如下:

- 不需要特殊的指令;
- 对7种最常用的常数不需要额外的字操作数;
- 缩短指令周期,即不经过MDB就能直接访问。

当6种常数之一被用做立即寻址模式的源操作数时,汇编程序会自动转为利用R2或R3的方式。状态寄存器SR/R2(用做源或目的寄存器)只能用于寄存器模式。其他的寻址位As组合支持绝对寻址和位处理,而不必增加代码。R2与R3用于常数模式时不能进行显式寻址,它们只能作为一个寄存器数据源。

常数发生器使得某些指令可用别的指令来模拟。用这种方法使CPU变得异常简单。整个指令组只需要27条指令。例如单操作数指令:

```
CLR    dst
```

可以用双操作数指令以同样长度来模拟,即

```
MOV    R3, dst
```

或等价地有

```
MOV    #0, dst
```

其中“#0”由汇编程序以“As=00”的R3来取代。这样做的优点如下:

- 指令长度为单字;
- CPU内不需要额外的硬件和控制操作;
- 源操作数为寄存器寻址,即对常数(#0)的读取不需要额外的取周期。

## 5.2 寻址模式

用对源操作数的全部7种寻址模式和对目的操作数的全部4种寻址模式可以访问整个地址空间。位数表明了As和Ad模式位的内容。表5.3列出了As和Ad的寻址模式。

表 5.3 As/Ad 寻址模式

As/Ad	寻址模式	语法	说明
00/0	寄存器模式	Rn	寄存器内容为操作数
01/1	变址模式	X(Rn)	(Rn + X)指向操作数, X存于后续字中
01/1	符号模式	ADDR	(PC + X)指向操作数, X存于后续字中,使用了变址模式的X(PC)
01/1	绝对模式	&ADDR	指令后续字含绝对地址
10/-	间接寄存器模式	@Rn	Rn为指针指向操作数
11/-	间接增量模式	@Rn+	Rn为指针指向操作数,然后Rn增加
11/-	立即模式	#N	指令后续字含立即数N,使用了间接增量模式的@PC+

**注意：寻址模式**

将 PC 当做工作寄存器的寻址与寻址模式的正常用法相同。由 PC 指向当前执行指令的后续字来形成特定的寻址模式。

以下用例子详细解说 7 种寻址模式。大多数例子对源和目的操作数用了相同的寻址模式；但是对于一条指令，任何有效的源和目的操作数的寻址模式组合都是可能的。

**5.2.1 寄存器模式**

汇编源程序  
MOV R10, R11

ROM 中内容

MOV R10, R11
--------------

长度： 1 或 2 个字。

操作： 移动 R10 内容到 R11。R10 不受影响。

备注： 对源和目的操作数均有效。

举例： MOV R10, R11

执行前	执行后		
R10 <table border="1"><tr><td>0A023h</td></tr></table>	0A023h	R10 <table border="1"><tr><td>0A023h</td></tr></table>	0A023h
0A023h			
0A023h			
R11 <table border="1"><tr><td>0FA15h</td></tr></table>	0FA15h	R11 <table border="1"><tr><td>0A023h</td></tr></table>	0A023h
0FA15h			
0A023h			
PC <table border="1"><tr><td>PC<sub>old</sub></td></tr></table>	PC <sub>old</sub>	PC <table border="1"><tr><td>PC<sub>old</sub> + 2</td></tr></table>	PC <sub>old</sub> + 2
PC <sub>old</sub>			
PC <sub>old</sub> + 2			

**注意：寄存器中的数据**

因为寄存器中的数据是字数据，任何寄存器-寄存器操作总是字操作，必须用字指令。

**5.2.2 变址模式**

汇编源程序  
MOV 2(R5), 6(R6)

ROM 中内容

MOV X(R5), Y(R6)
------------------

X = 2
-------

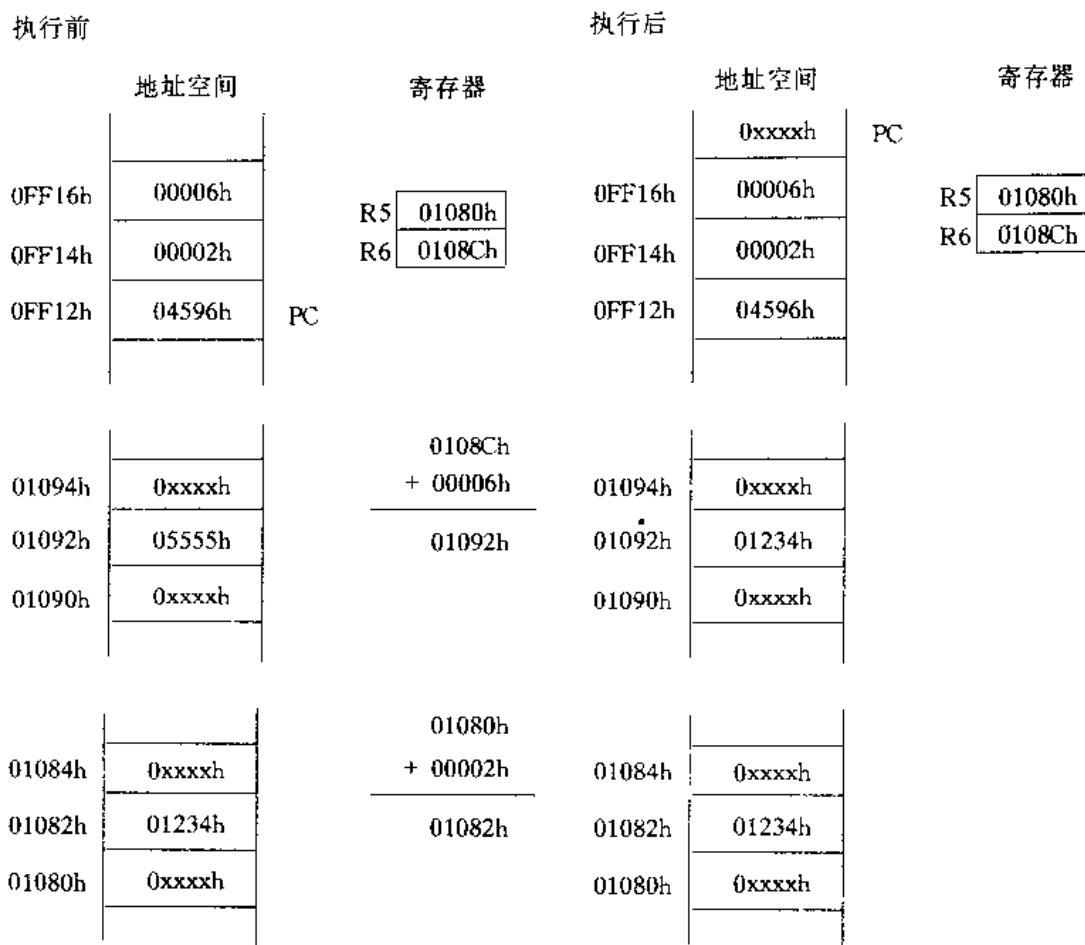
Y = 6
-------

长度： 2 或 3 个字。

操作： 移动源地址内容(R5 + 2 的内容)到目的地址(R6 + 6 的内容)。源和目的寄存器(R5 和 R6)不受影响。变址模式中 PC 自动增加，因此，程序继续执行下一条指令。

备注： 对源和目的操作数均有效。

举例： MOV 2(R5), 6(R6)



### 5.2.3 符号模式

汇编源程序  
MOV EDE, TONI

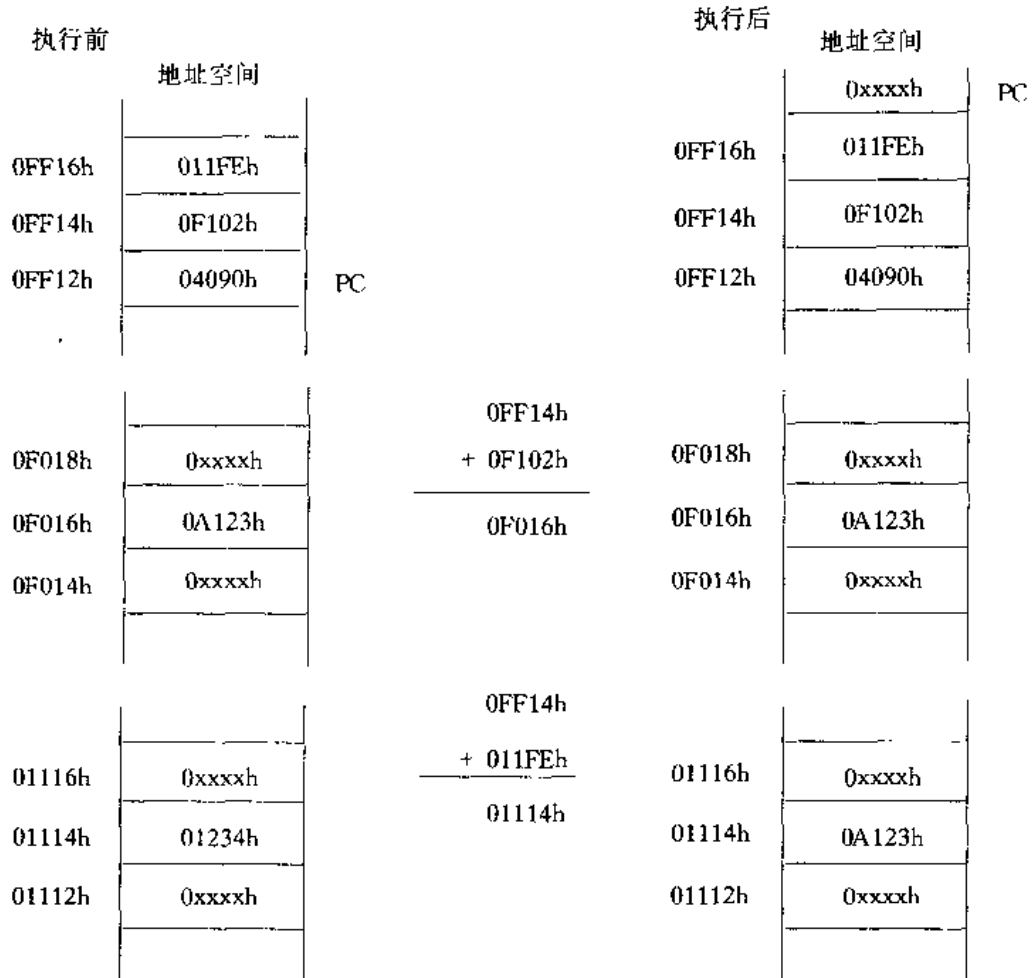
ROM 中内容
MOV X(PC), Y(PC)
X = EDE - PC
Y = TONI - PC

长度： 2 或 3 个字。

操作： 移动源地址 EDE 内容(PC + X 的内容)到目的地址 TONI(PC + Y 的内容)。指令的后续字包含 PC 与源和目的地址的差。汇编程序自动计算并插入偏移量 X 与 Y。符号模式中 PC 自动增加, 因此, 程序继续执行下一条指令。

备注： 对源和目的操作数均有效。

举例： MOV EDE, TONI ;源地址 EDE = 0F016h  
;目的地址 TONI = 01114h



### 5.2.4 绝对模式

汇编源程序  
MOV &EDE, &TONI

ROM 中内容
MOV X(0), Y(0)
X = EDE
Y = TONI

长度： 2 或 3 个字。

操作： 移动源地址 EDE 内容到目的地址 TONI。指令的后续字包含源和目的地址的绝对地址。汇编程序自动计算并插入偏移量 X 与 Y。绝对模式中 PC 自动增加，因此，程序继续执行下一条指令。

备注： ;对源和目的操作数均有效。

举例： MOV &EDE, &TONI ;源地址 EDE = 0F016h  
;目的地址 TONI = 01114h



执行前		PC	执行后	
地址空间	内容		地址空间	内容
0FF16h	01114h	PC	0FF16h	01114h
0FF14h	0F016h		0FF14h	0F016h
0FF12h	04292h		0FF12h	04292h
0F018h	0xxxxh		0F018h	0xxxxh
0F016h	0A123h		0F016h	0A123h
0F014h	0xxxxh		0F014h	0xxxxh
01116h	0xxxxh		01116h	0xxxxh
01114h	01234h		01114h	0A123h
01112h	0xxxxh		01112h	0xxxxh

此寻址模式主要用于定位绝对的、固定地址的硬件外围模块。对它们用绝对寻址可保证软件的透明度,例如位置独立代码(PIC)编程技术。绝对模式只用于代码段 0。

### 5.2.5 间接模式

汇编源程序  
MOV @R10,0(R11)

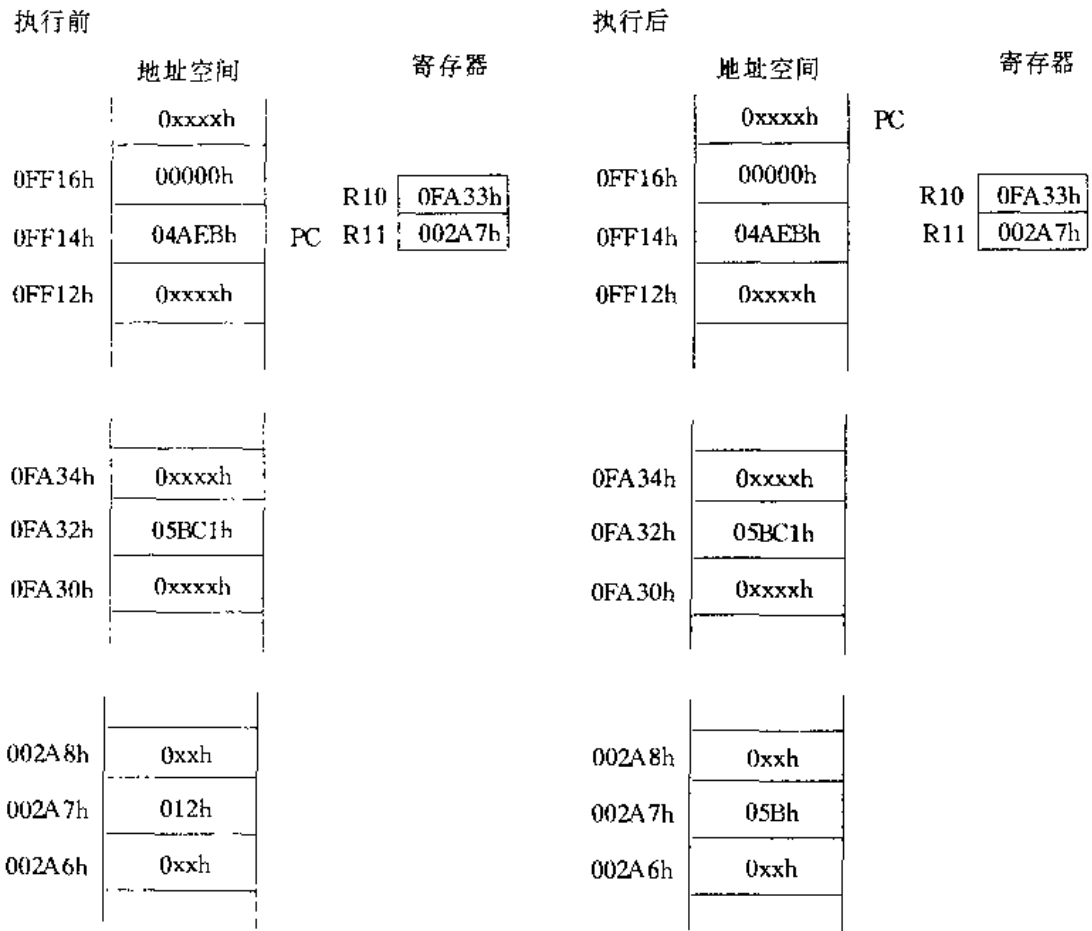
ROM 中内容  
MOV @R10,0(R11)

长度: 1 或 2 个字。

操作: 移动源地址内容(R10 的内容)到目的地址(R11 的内容)。寄存器不受影响。

备注: 仅对源操作数有效。对目的操作数的替代方法是 0(Rd)。

举例: MOV.B @R10,0(R11)



### 5.2.6 间接增量模式

汇编源程序  
MOV @R10+,0(R11)

ROM 中内容  
MOV @R10+,0(R11)

长度： 1 或 2 个字。

操作： 移动源地址内容(R10 的内容)到目的地址(R11 的内容)。R10 取数后增加 1(字节操作)或增加 2(字操作),它指向下一地址而不需要额外开销。这对于表处理非常有用。

备注： 仅对源操作数有效。对目的操作数的替代方法是 0(Rd)并加 1 条指令“INCD Rd”。

举例： MOV @R10+,0(R11)



执行前		执行后	
地址空间		地址空间	
0FF16h	01192h	0FF18h	0xxxxh
0FF14h	00045h	0FF16h	01192h
0FF12h	040B0h	0FF14h	00045h
		0FF12h	040B0h
010AAh	0xxxxh	0FF16h + 01192h	010AAh
010A8h	01234h	010A8h	00045h
010A6h	0xxxxh		0xxxxh

PC

### 5.2.8 指令的时钟周期与长度

CPU 的运行速度并不依赖于个别指令。它取决于指令的格式和寻址模式。表 5.4 和表 5.5 中的时钟周期数是针对片内振荡器频率的。

#### 1. I 类格式指令

表 5.4 I 类格式指令

寻址模式		周期数	指令长度/字	实例
As	Ad			
00, Rn	0, Rm	1	1	MOV R5, R8
	0, PC	2	1	BR R9
00, Rn	1, x(Rm)	4	2	ADD R5, 3(R6)
	1, EDE		2	XOR R8, EDE
	1, &EDE		2	MOV R5, &EDE
01, x(Rn) 01, EDE 01, &EDE	0, Rm	3	2	MOV 2(R5), R7
			2	AND EDE, R6
			2	MOV &EDE, R8
01, x(Rn) 01, EDE 01, &EDE	1, x(Rm)	6	3	ADD 3(R4), 6(R9)
	01, TONI		3	CMP EDE, TONI
	1, &TONI		3	MOV 2(R5), &TONI
			3	ADD EDE, &TONI
10, @Rn	0, Rm	2	1	AND @R4, R5
10, @Rn	1, x(Rm)	5	2	XOR @R5, 8(R6)
	1, EDE		2	MOV @R5, EDE
	1, &EDE		2	XOR @R5, &EDE

续表 5.4

寻址模式		周期数	指令长度/字	实例
As	Ad			
11, @Rn+	0, Rm	2	1	ADD @R5+, R6
	0, PC	3	1	BR @R9+
11, #N	0, Rm	2	2	MOV #20, R9
	0, PC	3	2	BR #2AEh
11, @Rn+	1, x(Rm)	5	2	MOV @R9+, 2(R4)
11, #N	1, EDE		3	ADD #33, EDE
11, @Rn+	1, &EDE		2	MOV @R9+, &EDE
11, #N			3	ADD #33, &EDE

## 2. II类格式指令

表 5.5 II类格式指令

寻址模式 A(s/d)	周期数		指令长度/字	实例
	RRA RRC SWPB SXT	PUSH/ CALL		
00, Rn	1	3/4	1	SWPB R5
01, x(Rn)	4	5	2	CALL 2(R7)
01, EDE	4	5	2	PUSH EDE
01, &EDE				SXT &EDE
10, @Rn	3	4	1	RRC @R9
11, @Rn+ (见“注意”)	3	4/5	1	SWPB @R10
11, #N			2	CALL #81h

### 注意：II类格式指令的立即模式

指令 RRA、RRC、SWPB 和 SXT 不能对目的操作数用立即寻址模式。这会产生不可预知的程序运行结果。

## 3. III类格式指令

Jxx 指令无论跳转与否都需要相同的周期数。

时钟周期： 2 个。

指令长度： 1 个字。

## 4. 其他指令或操作

RETI 时钟周期： 5 个。

指令长度： 1 个字。

中断 时钟周期： 6 个。

WDT 复位 时钟周期： 4 个。

复位(RST/NMI) 时钟周期： 4 个。

### 5.3 指令集概述

本节对指令集作简单介绍。

指令对 SR 位的影响表示如下：

- \* 影响状态位；
- - 不影响状态位；
- 0 状态位复位；
- 1 状态位置位。

指令的源操作数和目的操作数部分由两个字段定义(寻址模式见 5.2 节),即

- src 源操作数,由 As 和 S-reg 定义。
- dst 目的操作数,由 Ad 和 D-reg 定义。
- As 寻址位,表示源操作数的寻址模式。
- S-reg 源操作数使用的工作寄存器。
- Ad 寻址位,表示目的操作数的寻址模式。
- D-reg 源操作数使用的工作寄存器。
- B/W 字节或字操作。
  - 0: 字操作。
  - 1: 字节操作。

#### 注意：目的地址

目的地址可以是 64 KB 范围内任何一处。数据将写入的操作数目的地址必须是能够写入的地址,否则数据将丢失。

#### 5.3.1 双操作数指令

双操作指令格式如下：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
操作码				S-Reg				Ad	B/W	As	D-Reg				

			状态位			
			V	N	Z	C
MOV	src, dst	src → dst	-	-	-	-
ADD	src, dst	src + dst → dst	*	*	*	*
ADD C	src, dst	src + dst + C → dst	*	*	*	*
SUB	src, dst	dst - .not. src + 1 → dst	*	*	*	*
SUB C	src, dst	dst + .not. src + C → dst	*	*	*	*
CMP	src, dst	dst - src	*	*	*	*
DADD	src, dst	src + dst + C → dst (dec)	x	x	*	*
AND	src, dst	src .and. dst → dst	0	*	*	*
BIT	src, dst	src .and. dst	0	*	*	*

BIC	src, dst	.not. src .and. dst → dst	- - - -
BIS	src, dst	src .or. dst → dst	- - - -
XOR	src, dst	src .xor. dst → dst	* * * *

**注意：CMP 和 SUB 指令**

除了结果的保存外，指令 CMP 与 SUB 相同。这一点对于指令 BIT 与 AND 也是同样的。

**5.3.2 单操作数指令**

单操作指令格式如下：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
操作码										B/W	As	D/S - Reg			

			状态位			
			V	N	Z	C
RRC	dst	C → MSB → ... LSB → C	*	*	*	*
RRA	dst	MSB → MSB → ... LSB → C	0	*	*	*
PUSH	src	SP - 2 → SP, src → @SP	.	.	.	.
SWPB	dst	字节交换	.	.	.	.
CALL	dst	SP - 2 → SP	.	.	.	.
		PC + 2 → 堆栈, dst → PC				
RETI		TOS → SR, SP → SP + 2	*	*	*	*
		TOS → PC, SP → SP + 2				
SXT	dst	位 7 → 位 → ... → 位 15	0	*	*	*

CALL 指令可以使用所有寻址模式。例如使用符号模式(地址)、立即模式(#N)、绝对模式(&EDE)或变址模式(X(Rn)),指令的后续字包含地址信息。

**5.3.3 条件跳转**

条件跳转指令可使程序产生相对于 PC 的分枝跳转。可跳转地址是相对于跳转指令执行时 PC 的 -511 ~ +512 个字的范围内。10 位的 PC 偏移量是一个 10 位的有符号数,乘 2 后与 PC 相加。条件跳转不影响状态位。

取指令码和增加 PC 按以下公式：

$$PC_{\text{new}} = PC_{\text{old}} + 2 + PC_{\text{offset}} \times 2$$

条件跳转指令格式如下：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
操作码			条件			10 位 PC 偏移量									

JEQ/JZ	label	Z = 1 时, 跳转到 label
JNE/JNZ	label	Z = 0 时, 跳转到 label
JC	label	C = 1 时, 跳转到 label

JNC	label	C = 0 时, 跳转到 label
JN	label	N = 1 时, 跳转到 label
JGE	label	(N . XOR . V) = 0 时, 跳转到 label
JL	label	(N . XOR . V) = 1 时, 跳转到 label
JMP	label	无条件跳转到 label

### 5.3.4 模拟指令的简短格式

使用常数发生器的基本指令可以表示为更为流行的模拟指令(见表 5.6)。对状态位的影响根据基本指令执行的结果来确定。

表 5.6 模拟指令的简短格式

	指令助记符		说明	状态位				替代的模拟指令
				V	N	Z	C	
算 术 指 令	ADC[.W]	dst	C 位加到目的操作数	*	*	*	*	ADDC #0, dst
	ADC.B	dst	C 位加到目的操作数	*	*	*	*	ADDC.B #0, dst
	DADC[.W]	dst	C 位十进制加到目的操作数	*	*	*	*	DADD #0, dst
	DADC.B	dst	C 位十进制加到目的操作数	*	*	*	*	DADD.B #0, dst
	DEC[.W]	dst	目的操作数减 1	*	*	*	*	SUB #1, dst
	DEC.B	dst	目的操作数减 1	*	*	*	*	SUB.B #1, dst
	DECD[.W]	dst	目的操作数减 2	*	*	*	*	SUB #2, dst
	DECD.B	dst	目的操作数减 2	*	*	*	*	SUB.B #2, dst
	INC[.W]	dst	目的操作数加 1	*	*	*	*	ADD #1, dst
	INC.B	dst	目的操作数加 1	*	*	*	*	ADD.B #1, dst
	INCD[.W]	dst	目的操作数加 2	*	*	*	*	ADD #2, dst
	INCD.B	dst	目的操作数加 2	*	*	*	*	ADD.B #2, dst
	SBC[.W]	dst	从目的操作数减去 C 位	*	*	*	*	SUBC #0, dst
	SBC.B	dst	从目的操作数减去 C 位	*	*	*	*	SUBC.B #0, dst
逻 辑 指 令	INV[.W]	dst	目的操作数取反	*	*	*	*	XOR #0FFFFh, dst
	INV.B	dst	目的操作数取反	*	*	*	*	XOR.B #0FFFFh, dst
	RLA[.W]	dst	算术左循环	*	*	*	*	ADD dst, dst
	RLA.B	dst	算术左循环	*	*	*	*	ADD.B dst, dst
	RLC[.W]	dst	经 C 位算术左循环	*	*	*	*	ADDC dst, dst
	RLC.B	dst	经 C 位算术左循环	*	*	*	*	ADDC.B dst, dst
数 据 指 令 ( 常 用 )	CLR[.W]		清除目的操作数	-	-	-	-	MOV #0, dst
	CLR.B		清除目的操作数	-	-	-	-	MOV.B #0, dst
	CLRC		C 位复位	-	-	-	0	BIC #1, SR
	CLR N		N 位复位	-	0	-	-	BIC #4, SR
	CLR Z		Z 位复位	-	-	0	-	BIC #2, SR
	POP	dst	数据出栈	-	-	-	-	MOV @SP+, dst
	SETC		C 位置位	-	-	-	1	BIS #1, SR
	SET N		N 位置位	-	1	-	-	BIS #4, SR
	SET Z		Z 位置位	-	-	1	-	BIS #2, SR
	TST[.W]	dst	目的操作数测试	0	*	*	*	CMP #0, dst
	TST.B	dst	目的操作数测试	0	*	*	*	CMP.B #0, dst





	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
00x																
04x																
08x																
0Cx																
10x	RRC	RRC. B	SWPB		RRA	RRA. B	SXT		PUSH	PUSH. B	CALL		RETI			
14x																
18x																
1Cx																
20x	JNE/ JNZ															
24x	JEQ/ JZ															
28x	JNC															
2Cx	JC															
30x	JN															
34x	JGE															
38x	JL															
3Cx	JMP															
40x...4Cx	MOV, MOV. B															
50x...5Cx	ADD, ADD. B															
60x...6Cx	ADDC, ADDC. B															
70x...7Cx	SUBC, SUBC. B															
80x...8Cx	SUB, SUB. B															
90x...9Cx	CMP, CMP. B															
A0x...ACx	DADD, DADD. B															
B0x...BCx	BIT, BIT. B															
C0x...CCx	BIC, BIC. B															
D0x...DCx	BIS, BIS. B															
E0x...ECx	XOR, XOR. B															
F0x...FCx	AND, AND. B															

图 5.2 核心指令分布

## 第 6 章 硬件乘法器

由于硬件乘法器可视为一个未集成入 CPU 的 16 位外围模块,因此, CPU 的结构和指令都没有改变。乘法中两个操作数均被存入乘法器的寄存器中,用户在输入第二个操作数之后便可以读取结果。也就是说该乘法操作并不需要额外的时钟周期。

MSP430 系列采用的硬件乘法器模块能够在不改变基本结构的条件下增加其功能。可采用的乘法操作如下:

- 16 位  $\times$  16 位;
- 16 位  $\times$  8 位;
- 8 位  $\times$  16 位;
- 8 位  $\times$  8 位。

硬件乘法器模块支持 3 种乘法:无符号数相乘(MPY)、有符号数相乘(MPYS)、无符号相加(MAC)。硬件乘法器模块与总线系统的连接原理图如图 6.1 所示。

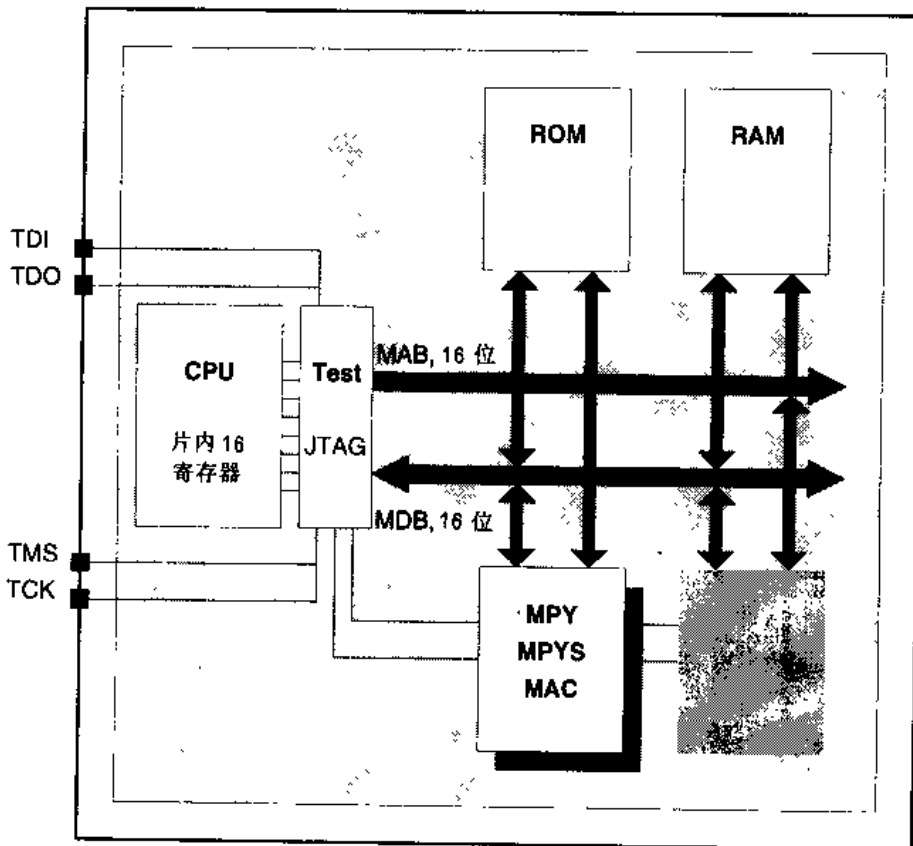


图 6.1 硬件乘法器模块与总线系统的连接



```
.ENDIF
```

```
*****
* 例:将硬件乘法结果加入 64 位 RAM 数据 *
*****
      ADD      &RESLO, &RAM      ; 结果低字加入 RAM
      ADDC     &RESHI, &RAM+2    ; 结果高字加入 RAM+2
      ADC      &RAM+4            ; 进位加入扩展字
      ADC      &RAM+6            ; 进位加入扩展字
```

上面程序代码为 32 B, 执行周期为 32 个(16 位×16 位)。

## 2. 有符号数相乘, 16 位×16 位、16 位×8 位、8 位×16 位、8 位×8 位

```
*****
* 将 2 个操作数送入硬件乘法器的寄存器 *
* 如果有 1 个操作数为 8 位, 则需要作符号扩展 *
* 用常数 OPERAND1 和 OPERAND2 来标明是否为字节数据 *
*****
OPERAND1 .EQU      0                ; 0: 操作数 1 为字(16 位)
                                           ; 8: 操作数 1 为字节(8 位)
OPERAND2 .EQU      0                ; 0: 操作数 2 为字(16 位)
                                           ; 8: 操作数 2 为字节(8 位)

MPY      .EQU      0130h
MPYS     .EQU      0132h
MAC      .EQU      0134h
OP2      .EQU      0138h
RESLO    .EQU      013Ah
RESHI    .EQU      013Ch
SUMEXT   .EQU      013Eh

      .BSS        OPER1, 2, 200h
      .BSS        OPER2, 2
      BSS         RAM, 8

      .IF OPERAND1 = 0
      MOV         &OPER1, &MPYS      ; 装入操作数 1
                                           ; 定义有符号乘法
      .ELSE
      MOV.B       &OPER1, &MPYS      ; 装入操作数 1
                                           ; 定义有符号乘法
      SXT        &MPYS                ; 字节数据经符号扩展成字数据
      .ENDIF
      .IF OPERAND2 = 0
      MOV         &OPER2, &OP2       ; 装入操作数 2 并开始有符号乘法运算
      .ELSE
      MOV.B       &OPER2, &OP2       ; 装入操作数 2
      SXT        &OP2                ; 符号扩展并开始有符号乘法运算
```





### 6.3 硬件乘法器的 SFR 位

由于硬件乘法器的乘法运算十分快速且不需要中断介入,因此未用 SFR 位。

### 6.4 硬件乘法器的软件限制

在使用硬件乘法器时,有两种情况必须引起注意,即

- 用间接或间接增量寻址模式处理结果。
- 在中断例程中使用硬件乘法器。

#### 6.4.1 硬件乘法器的软件限制——寻址模式

访问乘法结果的方式有 3 种:变址寻址、间接寻址或间接增量寻址。对于变址寻址模式,包括符号寻址和绝对寻址模式,访问结果寄存器时没有任何限制;但是,当用间接寻址和间接增量寻址模式访问结果寄存器时,在装入操作数 2 与读取任一结果寄存器的指令之间;至少需要插入一条指令。例子如下:

```

*****
* 例:操作数 1 与操作数 2 相乘
*****
RESLO    .SET      013Ah          ; RESLO; RESLO 的地址
          PUSH     R5              ; R5 用于存放 RESLO 地址
          MOV      #RESLO, R5
          MOV      &OPER1, &MPY   ; 装入操作数 1
                                     ; 定义无符号乘法
          MOV      &OPER2, &OP2   ; 装入操作数 2 并开始乘法运算
*****
* 例:硬件乘法结果加入 64 位 RAM 数据
*****
          NOP                      ; 在装入操作数 2 和用间接寻址模式处理结果
                                     ; 之间至少要相隔一个周期
          ADD      @R5+, &RAM      ; 结果低字加入 RAM
          ADDC    @R5, &RAM+2     ; 结果高字加入 RAM+2
          ADC     &RAM+4          ; 进位加入 RAM+4
          ADC     &RAM+6          ; 进位加入 RAM+6
          POP     R5

```

上面的例子表明,间接寻址或间接增量寻址模式与绝对寻址模式相比,需要更多的时钟周期和代码来将乘法结果传送到目的地址。显然,并不需要用间接或间接增量寻址模式来访问绝对地址的硬件乘法器。

#### 6.4.2 硬件乘法器的软件限制——中断程序

整个乘法程序有 3 步:



- 将第一操作数 OP1 写入硬件乘法器,同时确定了乘法的类型;
- 将第二操作数 OP2 写入硬件乘法器,乘法开始;
- 处理 RESLO、RESHI、SUMEXT 寄存器中的乘法结果。

如果在主程序中使用硬件乘法,那么考虑以下情况是必要的。当硬件乘法不在主程序而在中断程序中时,由于进入中断服务程序将把通用中断允许位复位,硬件乘法器不会受到后续中断的影响;但是在一般情况下,为了遵循中断程序尽量简短的原则,乘法及整个数据处理过程最好是在中断程序外完成。

在中断程序中的乘法将对主程序中的乘法程序产生影响。

#### 1. 中断发生在第一操作数 OP1 传入硬件乘法器后

第一操作数 OP1 地址的 2 位最低有效位决定了乘法的类型,而这个信息无法用任何后来的操作恢复;因此,在最初两步操作(即传送第一及第二操作数给乘法器)之间,不允许接受中断请求。

#### 2. 中断发生在第二操作数 OP2 传入硬件乘法器后

经过最初两步,乘法结果已经被存放在相应的寄存器 RESLO、RESHI 和 SUMEXT 中,可在中断服务程序中加以保存。例如,可以暂存在堆栈(PUSH)中,并在做完另一次乘法后将其恢复(POP);但是,这样将使中断程序为此增加代码和运行周期。如果在进行乘法操作前关闭中断(DINT),并在乘法结束后打开中断(EINT),就可以避免上述情况的发生;但是,这一方法有个缺点:如果中断事件恰好发生在这段时间内,则紧急中断被抑制的可能就大大增加了。

#### 3. 原则性建议

一般说来,当主程序中已经用了硬件乘法,就要避免在中断程序中使用硬件乘法。专门的应用软件、应用软件库或其他所有系统包含的软件都应考虑在内。各种不同方法的讨论都表明,那样应用的结果其负而影响多于正面的。遵循原则性建议,使中断程序简短些是最好的实现方法。

## 第 7 章 振荡器与系统时钟发生器

振荡器和系统时钟发生器的主要设计目标是廉价和低功耗。

为达到系统廉价,外接器件缩减到只有一个普通晶振。使用低频晶体和含有倍频器的振荡器可满足系统时钟速度与低功耗这两个要求。

通常,极低功耗设备会增加各种工作模式来强制实现一些功能,例如:启动时序,长时间相对于电压、温度、时间的频率稳定性,高度稳定的实时时钟时基等。

小电流的实时应用有两个互相矛盾的要求:满足节能要求的低频系统时钟和为了快速响应事件请求的高频系统时钟。尤其对于电池供电的应用系统,特别关注电流的消耗。而在实时应用中为了响应外部事件或定时请求,偶尔也必须要求快速。

理论上,一个能快速启动并允许用于各种不同功耗模式的处理机时钟发生器可以解决这一矛盾;但是,另一方面,快速启动通常伴随着不可接受的低的频率稳定性。设计多个时钟源或为时钟设计各种不同的工作模式,可以解决某些外围部件实时应用的时钟要求,如低频通信、LCD 显示、定时器、计数器等。图 7.1 所示为时钟发生器的原理图。

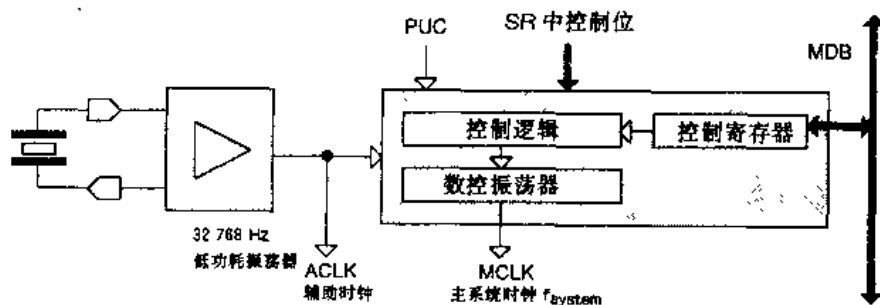


图 7.1 时钟发生器的原理

低频晶振的输出为 CPU 运行和外部模块提供了时钟信号。MSP430 的振荡器采用广泛使用的普通晶振,且不需要其他元件。

从降低电流消耗的观点看,CPU 和外部模块的不同要求需要两种时钟信号:

- 由晶振频率提供的辅助时钟信号 ACLK;
- 具有更高频率的系统时钟信号 MCLK,其频率为晶振频率的整倍数。

### 7.1 晶体振荡器

特殊设计的振荡器满足了低功耗及使用 32 768 Hz 晶振的要求。晶振只经过两个引脚的连接,不需要其他外部元件。所有保证工作稳定的元件或移相电容都集成在芯片中。

两个因素决定了选择熟悉和广泛使用的手表晶振,即低功耗的振荡器和时基、廉价。

由于 SR 中 OscOff 的复位,振荡器上电后即开始工作。对 OscOff 置位可使它停止。状态寄存器 SR 位定义如下:

15	9	8	7						0
留作未来增强用	V	SCG1	SCG0	OscOff	CPUOff	GIE	N	Z	C
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

## 7.2 处理机时钟发生器

为适应系统和具体应用的不同需要,微控制器的系统时钟必须满足如下不同的要求:

- 高频率,以便能对系统硬件请求和事件作出快速反应;
- 低频率,以便将电流消耗降至最少;
- 稳定的频率,以满足定时应用,如实时钟 RTC;
- 低 Q 值振荡器,以保证开始及停止操作没有时间延迟。

以上所提的互相抵触的基本要求是无法实现的,或者用高 Q 值高频晶体振荡器,或者用低 Q 值 RC 振荡器。上述合适的电流消耗和稳定的频率又要求用低频晶振。MSP430 的折衷办法是用一个低频晶振,并将其倍频至标称的工作频率范围,即

$$f_{\text{system}} = N \times f_{\text{crystal}}$$

有许多将晶振频率倍频至系统频率的方法,其中的一些是实用的。最熟悉是锁相环技术 PLL(Phase Locked Loop)和锁频环技术 FLL(Frequency Locked Loop)。

频繁而不定时的间断工作模式使 PLL 技术在系统中应用有两大缺点。PLL 是跟随二阶响应的系统。所有的开关操作模式都是由相位失锁引起的,因此,需要连续不断作“失步模式”的处理。宽范围的关闭时间状态与闭环电路中的模拟滤波积分器的应用是矛盾的。最终利用相位与/或频率的偏离和频率不合适引起电容电量的自动改变,直至系统重新锁相。

结合数字控制振荡器 DCO(Digital Controlled Oscillator)的 FLL 技术避免了这两个严重的问题。

DCO 主要有以下特点:

- 快速启动;
- 用数字信号(而非模拟信号)进行控制。

除了这些优点外,有一点需要仔细考虑,即频率随供电电压、环境温度的变化。

DCO 是完全单调的。

FLL 工作时可作为一个连续频率积分器。一个增/减计数器根据环路控制持续修正倍频因子 N。跟随或更新的频率与晶振频率相一致。当采用 32 768 Hz 晶振时,30.5 μs 修正一次。

频率的积累误差与晶振的相同。相邻机器周期间的的时间偏差通常不超过 10%。系统频率与时间的关系如图 7.2 所示。

系统时钟的启动操作取决于原有的机器状态。在一次 PUC 中,DCO 复位至它的最低频率。在 PUC 状态消除后控制逻辑开始工作,而控制逻辑的正常工作需要频率稳定的晶振。

10 位频率积分器控制 DCO 工作频率。在 PUC 后开始于 0 的频率积分器增计数至所选定的针对系统频率的数值 N。当频率积分器需要最大数,即需要 1 024 时,这一过程需要稍长于晶振频率的周期数,而不是所选择的 10 位数值。控制逻辑系统的运行是不定期的。

控制器运行在间断情况下的应用要注意系统频率控制的处理条件。频率积分器的修正可

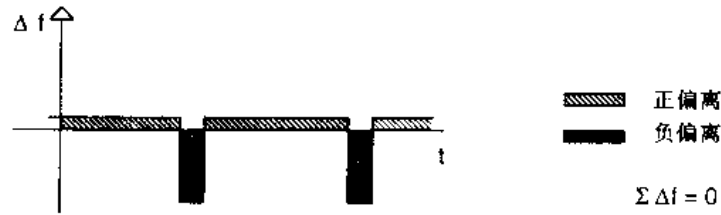


图 7.2 系统频率与时间的关系

以是在每一晶振周期( $30.5 \mu\text{s}$  @  $32\,768 \text{ Hz}$ )只增加  $f_{\text{system}}/N$  的周期。为了避免时间积累误差,长的积分周期是必须的。图 7.3 为系统频率发生器原理图。

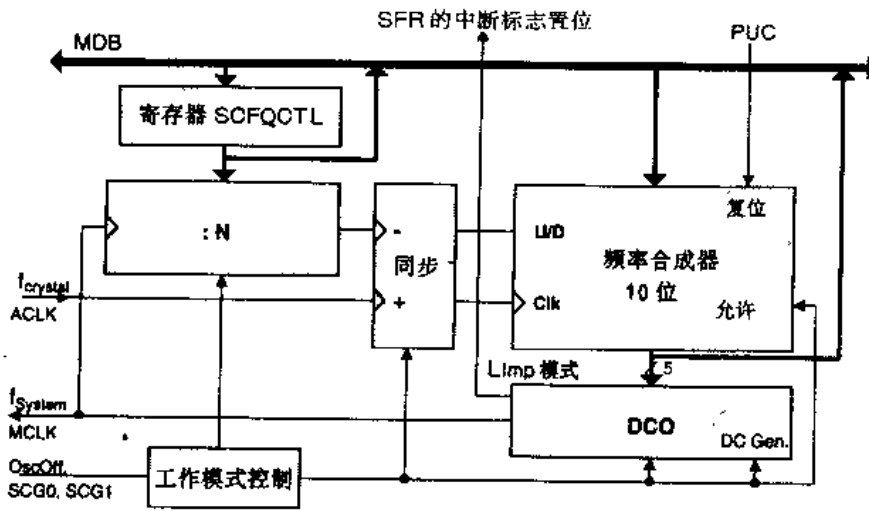


图 7.3 系统频率发生器原理图

如果 DCO 处于它的上限或下限频率,则 SFR 中的两个标志位允许应用程序恢复对系统的控制。

可以方便地通过访问标志 SCF10 与/或 SCF11 来检测运行是在频率的上限还是下限。

### 7.3 系统时钟工作模式

系统时钟发生器和晶振是由位于状态寄存器中的 3 个控制信号位来进行控制的。这 3 个信号在 4 种不同的上电条件下复位。

这 3 个控制信号使系统工作在不同的运行条件下(见表 7.1),对优化整个系统的功耗提供了最大的灵活性。在这 3 个信号的某些组合中,系统时钟 MCLK 停止工作,而频率积分器中已有的控制值被保留。

表 7.1 系统在 3 个控制信号控制下的各种运行条件

SCG1	SCG0	OscOff	晶振	DC 发生器	DCO	环路控制	说明
0	0	0	ON	ON	ON	ON	PUC 后状态 晶振、DCO 是活动的 环路控制正在工作
0	1	0	ON	ON	ON	OFF	省电模式 LPM1 晶振、DCO 是活动的 环路控制停止
1	0	0	ON	ON	OFF	OFF	省电模式 LPM2 晶振、DC 发生器是活动的 DCO、环路控制停止
1	1	0	ON	OFF	OFF	OFF	省电模式 LPM3 晶振是活动的 其余功能停止
x	x	1	OFF	OFF	OFF	OFF	省电模式 LPM4 全部功能停止 $f_{MCLK} = f_{ACLK} = 0 \text{ Hz}$

这 3 个控制信号提供了 5 种不同的功耗工作模式。充分利用它们能实现超低功耗的应用。所有这些不同的模式为系统应用提供了一种可能性,即系统可能工作在最短的时间片内,并在每一时间片内优化电流消耗。

SCG0 位控制 FLL 环。当其为“0”时,FLL 环工作;为“1”时,FLL 环关闭。

### 1. 从 PUC 启动

系统时钟控制寄存器 SCFQCTL 由 PUC 置为 01Fh,同时,频率积分器复位。这使得系统频率被置为最低值;然后,计数持续增加,直至锁定在系统频率上,这一频率等于晶振频率的 N 倍。

### 2. 低功耗模式 LPM4(晶振关闭)

在振荡器关闭模式期间,处理机的所有部件均停止工作,电流消耗达到最小限度。只有在系统上电电路检测到低电平或任一请求异步响应中断的外部中断事件时,才会开始工作。在此程序流程期间,应当适当允许可能用到的中断。

系统时钟发生器脱离振荡器关闭模式的启动顺序如下:

- 继续产生由频率积分器输出数值和 DCO 特征值定义的现有系统频率;
- 在晶振尚未开始工作时,频率积分器以  $f_{\text{system}}/N$  的频率连续减计数,直至 DCO 工作在它的最低频率;
- 晶振开始工作后,环路控制将频率积分器调整到跟随  $f_{\text{system}} = N \times f_{\text{crystal}}$  的数值。

### 3. 低功耗模式 LPM3(DC 发生器关闭)

在 DC 发生器关闭模式期间,只有晶振是活动的。设置基本时序条件的 DC 发生器的 DC

电流关闭。由于电路的高阻设计,使功耗被抑制。从 DC 发生器关闭的省电模式启动 DCO,要花一段时间( $t_{DC(On)}$ )才能工作在选择的频率下。时间范围在 ns~ $\mu$ s 之间。

#### 4. 低功耗模式 LPM2(DCO 关闭)

在 LPM2 模式期间,晶振和 DC 发生器仍然是活动的,因此可以快速启动。启动的延迟限于几个门延时。

#### 5. 低功耗模式 LPM1(FLL 关闭)

在 LPM1 模式期间,晶振、DC 发生器及 DCO 仍然是活动的。处理机及它的全部外围模块拥有全部功能,无任何限制。工作频率取决于频率积分器的输出数值。这一数值和 DCO 的特征值共同决定了 MCLK 信号的频率,它与系统频率  $f_{system}$  相同。

因为振荡器已经工作,启动没有时间延迟。环路控制异步激活,并伴有轻微的频率偏移;但是它的调节是快速和不定期的。

## 7.4 系统时钟控制寄存器

系统时钟发生器通过 3 个通用模块寄存器和 SFR 与处理机的其他部分相关联。通用模块寄存器位于较低安排字节模块的外围模块地址。3 个工作状态控制位 SCG0、SCG1 和 OscOff 由 CPU 的状态寄存器 SR 提供。

### 7.4.1 模块寄存器

两个 8 位寄存器控制系统时钟发生器。由用户软件将乘法系数 N 存入其中一个寄存器。另一个寄存器存放用于各种工作模式的控制位和信号。它只能用字节指令访问。

#### 1. 系统时钟频率控制

寄存器 SCFQCTL 的位定义如下:

SCFQCTL (052h)

7							0
M	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
rw-0	rw-0	rw-0	rw-1	rw-1	rw-1	rw-1	rw-1

寄存器 SCFQCTL 控制晶振频率的倍数。用 7 位指明范围  $(3+1) \sim (127+1)$ 。

$$f_{system} = (x \times 2^6 + x \times 2^5 + x \times 2^4 + x \times 2^3 + x \times 2^2 + x \times 2^1 + x \times 2^0 + 1) \times f_{crystal}$$

寄存器 SCFQCTL 在 PUC 后的缺省值是 31,它产生的因子为 32。

$f_{system}$  的范围是理论上的,它依赖于 DCO 的频率可调整范围(详细资料见电气特性)。

#### 注意: SCG 的倍频因子

寄存器 SCFQCTL( $2^6 \sim 2^0$ )控制晶振频率的倍数。7 位的值必须在 3~127 范围内。任何低于 3 的值会引起不可预知的结果,而起过 127 的值会强制 MCLK 的频率超过器件的性能指标。

#### 2. 系统时钟频率积分器

系统时钟频率积分器位定义如下:

SCF10 (050h)

7							0
0	0	0	FN_4	FN_3	FN_2	2 <sup>1</sup>	2 <sup>0</sup>
r	r	r	rw-0	rw-0	rw-0	rw-0	rw-0

SCF11 (051h)

7							0
2 <sup>9</sup>	2 <sup>8</sup>	2 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

由系统时钟频率积分器的输出来控制 DCO。该输出值可由 SCF11 和 SCF10 读出。表达式如下:

$$N_{DCO} = (x \times 2^9 + x \times 2^8 + x \times 2^7 + x \times 2^6 + x \times 2^5) + (1 - M) \times (x \times 2^4 + x \times 2^3 + x \times 2^2 + x \times 2^1 + x \times 2^0)$$

SCF10 中的位 2~4 定义 DCO 的标称频率( $f_{NOM}$ ),如下表所列。

FN_4	FN_3	FN_2	频率
0	0	0	$f_{NOM}$
0	0	1	$2 \times f_{NOM}$
0	1	x	$3 \times f_{NOM}$
1	x	x	$4 \times f_{NOM}$

#### 7.4.2 与系统时钟发生器相关的 SFR 位

SFR 中的 2 位根据系统时钟发生器执行的功能来控制系统的相互控制。这 2 位是:

- 振荡器故障中断标志 OFIFG(位于 IFG1.1, 初始状态不变);
- 振荡器故障中断允许 OFIFE(位于 IE1.1, 初始状态为复位)。

这一中断标志位是多源中断请求之一。当选择 NMI 功能时, 同一个中断向量也用于 RST/NMI 引脚的事件。这是不可屏蔽中断, 即通用中断允许位 GIE 不能禁止。由于共用同一个中断向量, 并且在 PUC 之后振荡器故障信号即为有效, 因此, 这一中断标志位不会自动复位。

有 3 种情况需要软件来处理, 即

- 在 PUC 之后, 必须编制一段程序来识别或设置振荡器状态, 以防止因振荡器故障信号的有效而将 OFIFG 永久置位。OFIFG 必须由软件复位。PUC 将 OFIFE 复位, 因此, 这时不会发生中断请求。
- 当从振荡器故障来的中断请求被接受时, 中断允许位 OFIFE 自动复位以阻止继续到来的中断请求, 直至软件作出适当的响应使振荡器故障信号失效。在到达这一状态后, OFIFE 可以按模块中断的一般规则重新置位。振荡器故障事件不受通用中断允许位(GIE)的影响。
- 中断标志 OFIFG 可用于在中断服务程序的开始识别中断源。OFIFG 的置位与 NMI 事件无关。它的作用是支配性的。

**注意：中断标志 OFIFG**

当中断请求被接受并得到服务时，中断标志 OFIFG 仍保持置位。这是强制性的，因为它是与 NMI 中断一起构成多源中断，并要指示软件处理振荡器故障事件。首先服务 OFIFG 的条件使得这一事件的优先级比 NMI 事件更高。

## 7.5 DCO 典型特性

DCO 的频率随供电电压和温度而变化，如图 7.4 所示。在锁频环工作情况下，这对于应用并不重要，因为控制周期与 ACLK 的周期相同。用 32 768 Hz 晶振时 DCO 的周期仅为 30.5  $\mu\text{s}$ 。DCO 的特征值如图 7.5 所示。

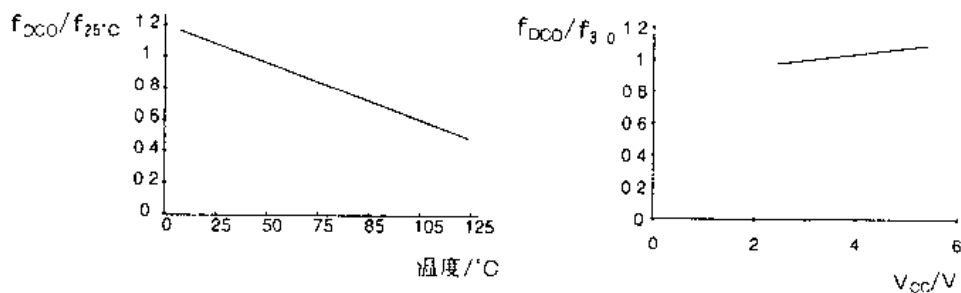


图 7.4 DCO 频率与供电电压和温度的关系

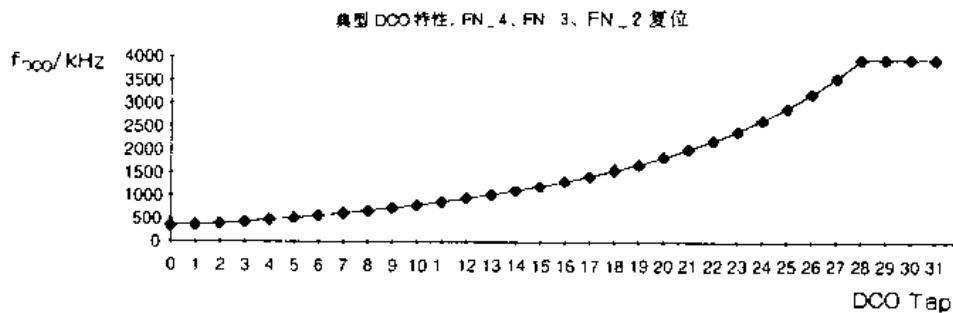


图 7.5 DCO 的特征值

**注意：DCO Taps**

系统时钟频率积分器寄存器 SCFH 的高 5 位被送入 DCO。如果寄存器 SCFQCTL 中的调制位 M 置位，系统频率仅由 DCO Taps 来确定。



## 第 8 章 数字 I/O 配置

### 8.1 通用端口 P0

通用端口 P0(见图 8.1)及其所有功能可以按引脚单独选择,并且每个信号都可作为一个中断源。

有 6 个寄存器用于端口 I/O 引脚的控制。

通用模块寄存器位于较低的安排字节模块的外围模块地址。这些寄存器必须用字节指令以绝对寻址模式访问。

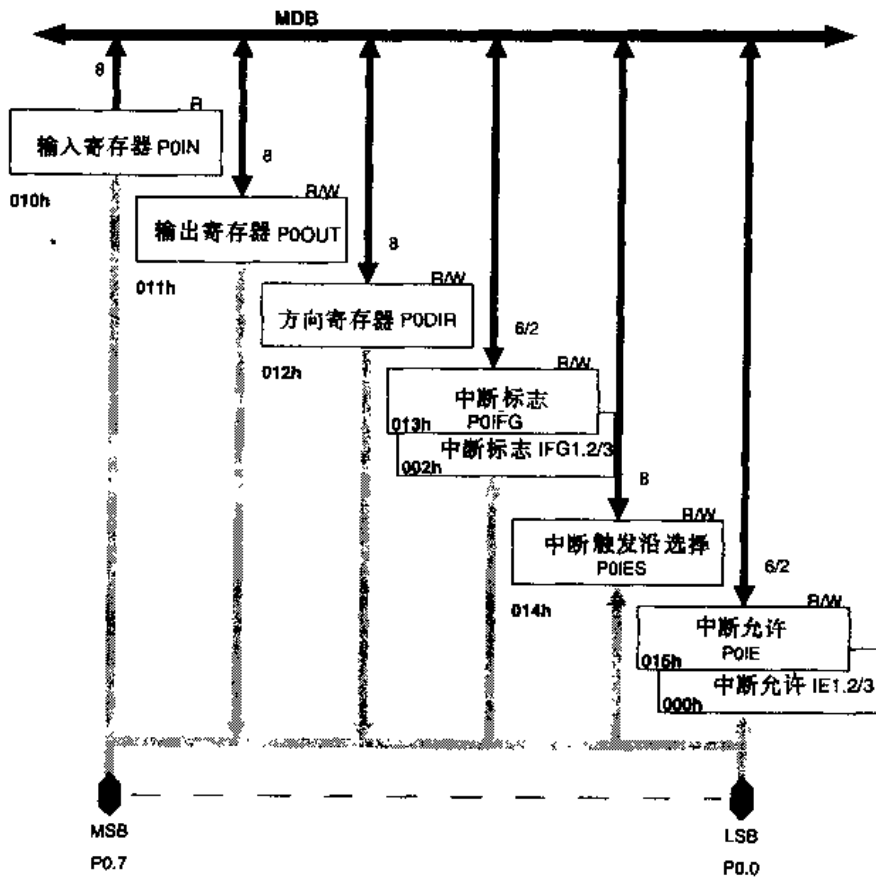


图 8.1 端口 P0 的结构

#### 8.1.1 P0 的控制寄存器

端口 P0 通过 8 位 MDB 和 MAB 与处理机内核相连。它必须通过字节指令访问。它的 6 个控制寄存器为数字 I/O 功能提供了最大的灵活性：

- 每个 I/O 位均可独立编程；



P0.0 中断标志位于 SFR 中。

**注意：中断标志 POIFLG.2~POIFG.7**

中断标志 POIFG.2~POIFG.7 只用了一个中断向量,是多源中断。当任一事件引起的中断进行服务时,POIFG.2~POIFG.7 不会自动复位。必须用软件来判定是对哪一个事件服务,并将相应的标志复位。

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

### 5. 中断触发沿选择寄存器 POIES

对应每个 I/O 引脚,中断触发沿选择寄存器都有一位用以选择哪一种电平跳变触发中断标志。每一位具有以下意义:

- 0: 电平由低到高跳变使标志位置位。
- 1: 电平由高到低跳变使标志位置位。

**注意：改变 POIES 位**

POIES 中各位的任何改变都可能使相应的中断标志位置位。

### 6. 中断允许寄存器 POIE

在寄存器 POIE 中,6 个 I/O 引脚 P0.2~P0.7 都有一位用以允许因中断事件产生中断请求。P0.0 和 P0.1 的中断允许位位于 SFR IE1.2 和 IE1.3 中。寄存器 POIE 的位定义如下:

POIE (015h)

7						0	
POIE.7	POIE.6	POIE.5	POIE.4	POIE.3	POIE.2		
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r0	r0

各位的意义如下:

- 0: 禁止中断。
- 1: 允许中断。

**注意：P0 的中断敏感性**

只有跳变才能引起中断请求,而静态电平则不能。

中断服务程序必须对多源中断标志 POIFG2~POIFG7 复位。单源中断标志 POIFG.0 和 POIFG.1 会在服务时自动复位。

如果在执行 RETI 指令后中断标志仍然置位(因为在中断服务期间引脚上又发生跳变),则在 RETI 指令完成后中断将再次发生。这一点保证了每次跳变都能被软件发现。

## 8.1.2 P0 的原理图

### 1. 端口 P0 的 P0.3~P0.7

端口 P0 的引脚逻辑是由 5 个寄存器(P0DIR、P0OUT、POIFG、POIE、POIES)和一个只读的寄存器(P0IN)构成。P0.3~P0.7 在设计上完全相同,如图 8.2 所示。

中断标志位可由适当的输入条件置位,也可以由软件置位。另外,修改方向控制位或中断

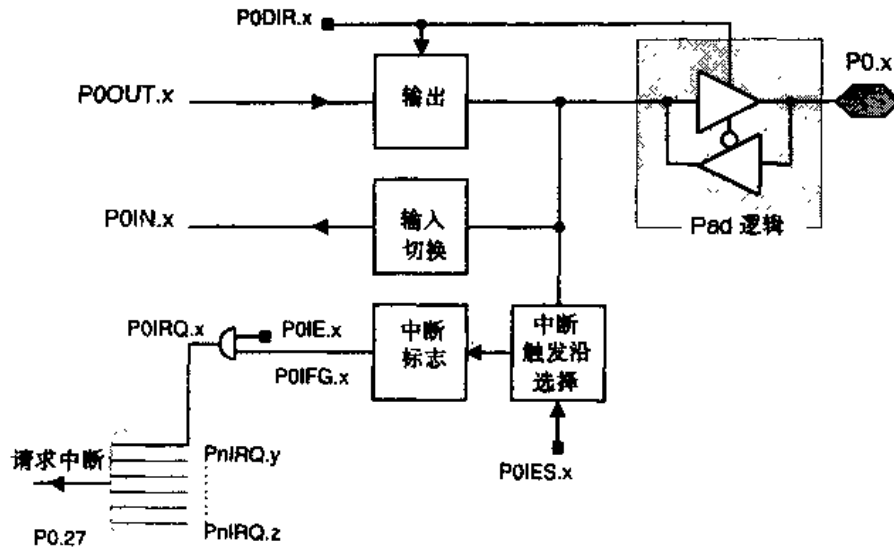


图 8.2 P0.3~P0.7 原理图

触发沿选择位也可能产生触发条件。

端口 P0 的 P0.2~P0.7 共用一个公共的中断向量。当 P0.2~P0.7 中断被接受时, 中断标志位不会自动复位。P0IFG.2~P0IFG.7 的各标志位必须用软件在相应的中断服务程序中复位。

## 2. 端口 P0 的 P0.2

P0.2(见图 8.3)与 P0.3~P0.7 稍有不同。它的输出信号既可由 P0OUT.2 决定, 也可由定时器/计数器的信号 TXD 决定。当输出控制寄存器的输出 TXE 置位时, TXD 信号被选择成为输出信号; 同时, Pad 逻辑切换成输出, 这时引脚与方向寄存器 P0DIR.2 无关。

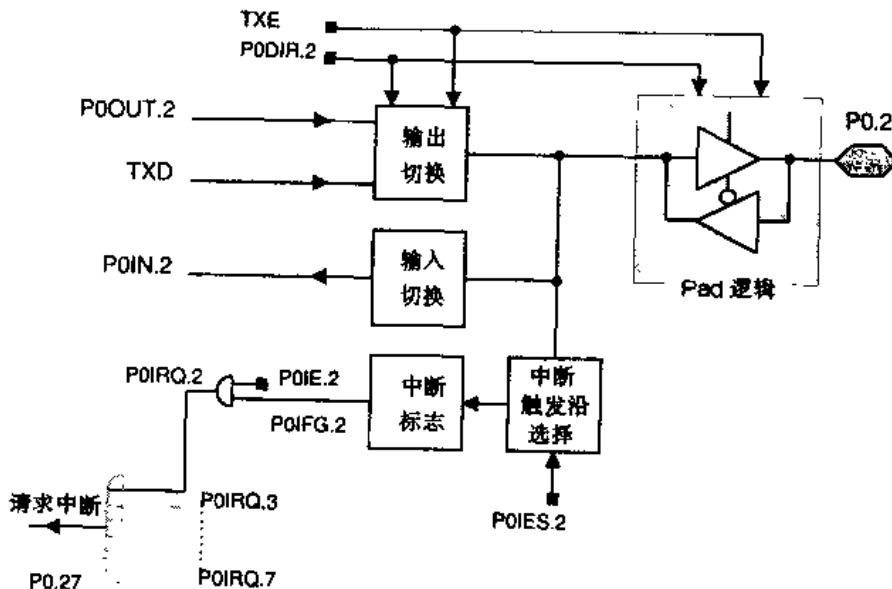


图 8.3 P0.2 的原理图

中断标志 P0IFG.2 与中断标志 P0IFG.3~P0IFG.7 共用一个中断向量。



### 8.1.3 P0 的中断控制功能

端口 P0 有 8 个中断标志位、8 个中断允许位、8 个中断事件触发沿选择位和 3 个中断向量。

3 个中断向量分别分配给

- P0.0;
- P0.1/RXD;
- P0.2~P0.7。

P0 中的两个信号 P0.0 和 P0.1/RXD 用于特定的信号处理。SFR 中的 4 位和 P0 地址帧 (address frame) 的 2 位用来处理中断事件:

- P0.0 的中断标志位 P0IFG.0(位于 IFG1.2 中,初始状态为复位);
- P0.1/RXD 中断标志位 P0IFG.1(位于 IFG1.3 中,初始状态为复位);
- P0.0 中断允许位 P0IE.0(位于 IE1.2 中,初始状态为复位);
- P0.1/RXD 中断允许位 P0IE.1(位于 IE1.3 中,初始状态为复位);
- P0.0 中断触发沿选择位(位于 P0IES.0,初始状态为复位);
- P0.1/RXD 中断触发沿选择位(位于 P0IES.1,初始状态为复位)。

两个中断标志都为单源中断,并且都会在中断服务后自动复位。中断允许位和中断触发沿选择位保持不变。

余下的 6 个 I/O 信号 P0.2~P0.7 的中断控制位位于 I/O 地址帧。每一个信号都有 3 位用以定义对中断事件的反应,即

- 中断标志位(P0IFG.2~P0IFG.7)。
- 中断允许位(P0IE.2~P0IE.7)。
- 中断触发沿选择位(P0IES.2~P0IES.7)。

中断标志位 P0IFG.2~P0IFG.7 是多源中断之一。当满足两个条件,即 P0IE.x( $2 \leq x \leq 7$ )置位且通用中断允许位 GIE 置位时,P0.2~P0.7 中的一个或多个上的任意中断事件将产生中断请求。它们共用一个中断向量,因此,中断标志位不会在中断服务后自动复位。

中断服务程序完成中断源的检测,并将相应的中断标志位复位。

**注意:** 多源中断标志 P0IFG.2~P0IFG.7

多源中断标志 P0IFG.2~P0IFG.7 在中断请求被接受和得到服务时保持置位。对于多源中断这是必须的。每个标志必须由它自己的中断服务程序复位。

## 8.2 通用端口 P1、P2

通用端口 P1 和 P2(见图 8.6)由各个引脚选择的功能来组成其全部功能,并且每一个信号都可用做中断源。

它们各有 7 个寄存器用来控制端口的引脚。

通用模块寄存器位于安排字节外围模块的低端地址空间。寄存器必须用字节指令以绝对寻址模式进行访问。

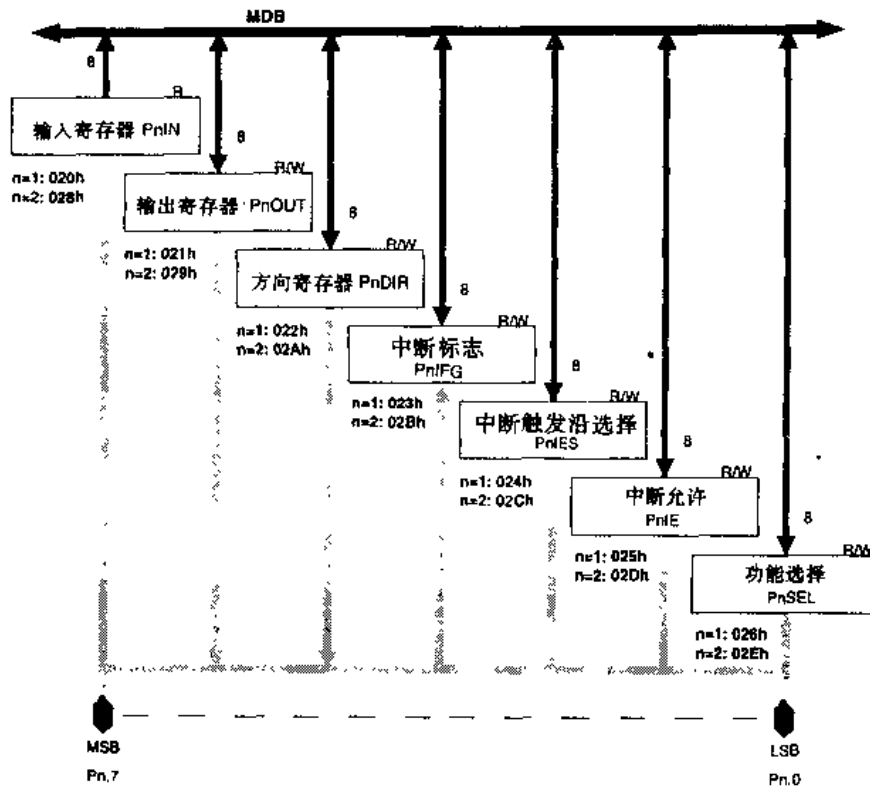


图 8.6 端口 P1、P2 的结构

### 8.2.1 P1、P2 的控制寄存器

端口 P1 和 P2 通过 8 位 MDB 和 MAB 与处理机内核相连。它们必须通过字节指令来访问。

7 个控制寄存器为数字 I/O 功能的应用提供了最大的灵活性：

- 所有 I/O 位均可独立编程；
- 可以有各种输入、输出和中断条件的组合；
- P1 和 P2 的 8 位全部可用于对外部事件的中断处理。

P1 和 P2 的 7 个寄存器分别如下：

寄存器	缩写	寄存器类型	地址	初始状态
输入寄存器	P1IN	只读	020h	-----
输出寄存器	P1OUT	读写	021h	不变
方向寄存器	P1DIR	读写	022h	复位
中断标志位	P1IFG	读写	023h	复位
中断触发沿选择	P1IES	读写	024h	不变
中断允许	P1IE	读写	025h	复位
功能选择寄存器	P1SEL	读写	026h	复位
输入寄存器	P2IN	只读	028h	-----
输出寄存器	P2OUT	读写	029h	不变
方向寄存器	P2DIR	读写	02Ah	复位

中断标志位	P2IFG	读写	02Bh	复位
中断触发沿选择	P2IES	读写	02Ch	不变
中断允许	P2IE	读写	02Dh	复位
功能选择寄存器	P2SEL	读写	02Eh	复位

以上所有寄存器为 8 位。必须用字节指令以绝对寻址模式进行访问。

### 1. 输入寄存器 P1IN 和 P2IN

这两个只读寄存器扫描 I/O 引脚上的信号。这些引脚应选择为输入方向。

**注意：对输入寄存器 P1IN、P2IN 写入**

对只读寄存器的写入会在写有效期间增加电流消耗。

### 2. 输出寄存器 P1OUT 和 P2OUT

如果 I/O 引脚用做输出,则 8 位的输出寄存器提供相应位的输出缓存信息。输出缓存可用所有包含目的操作数的指令进行修改。在读取时,输出缓存的内容与引脚定义方向无关。改变方向不会改变输出缓存的内容。

### 3. 方向寄存器 P1DIR 和 P2DIR

每个寄存器含有相互独立的 8 位,用于定义每个 I/O 引脚的方向。各位在 PUC 后复位。

- 0: I/O 引脚切换成输入模式。
- 1: I/O 引脚切换成输出模式。

### 4. 中断标志寄存器 P1IFG 和 P2IFG

每个寄存器含有对应于 I/O 引脚的 8 个标志位,表示是否有中断挂起。

- 0: 没有中断挂起。
- 1: 由于 I/O 引脚电平跳变引起中断挂起。

对寄存器 P1OUT、P1DIR 以及 P2OUT 和 P2DIR 的操作,也可能引起 P1IFG 或 P2IFG 中相应位的置位。

对某一个中断标志写“0”使它复位。

**注意：中断标志 P1IFG.0 ~ P1IFG.7、P2IFG.0 ~ P2IFG.7**

每一组中断标志 P1IFG.0 ~ P1IFG.7 和 P2IFG.0 ~ P2IFG.7 只用一个中断向量,它们都是多源中断向量。当中断被接受时,这些标志位不会自动复位。由中断服务程序确定服务的事件,并将相应的标志复位。

任何外部中断事件必须等于或长于 1.5 倍 MCLK 的时间,以保证该中断请求被接受并使相应的中断标志置位。

### 5. 中断触发沿选择寄存器 P1IES 和 P2IES

对应于每一个 I/O 引脚 P1.0 ~ P1.7 和 P2.0 ~ P2.7,寄存器中都有一位选择触发中断标志的电平跳变。该位的意义如下:

- 0: 电平由低到高使标志置位。
- 1: 电平由高到低使标志置位。



**注意：P1IES、P2IES 各位的改变**

改变 P1IES 和 P2IES 中的位会引起相应中断标志位置位。

PnIES. x	PnIN. x	PnIFG. x
0→1	0	不变
0→1	1	可以置位
1→0	0	可以置位
1→0	1	不变

**6. 中断允许寄存器 P1IE 和 P2IE**

每一个 I/O 引脚 P1.0 ~ P1.7 和 P2.0 ~ P2.7 都在寄存器 P1IE 和 P2IE 中有一相应的允许位,用以允许中断事件。

允许位的意义如下:

- 0: 禁止中断请求。
- 1: 允许中断请求。

**注意：P1、P2 的中断敏感性**

只有跳变能够引起中断,而静态电平则不能。

由于它们是多源中断,中断程序必须清除全部中断标志。

如果在执行 RETI 指令后中断标志仍然置位(因为在中断服务期间又发生跳变),则在 RETI 指令完成后中断将再次发生。这一点保证了每次跳变都能被软件发现。

**7. 功能选择寄存器 P1SEL 和 P2SEL**

每个寄存器都有相互独立的 8 位来定义访问 I/O 引脚的功能。端口功能或者一个定义的模块功能向引脚中输入数据,或从引脚中取数据。各位在 PUC 后复位,即

- 0: 端口功能,输出或输入数据由端口定义。
- 1: 模块功能,输出或输入数据由模块定义。

**注意：用 P1SEL、P2SEL 选择功能**

如果 PnSEL. x 置位,则中断触发沿选择电路被禁止。输入信号将不能改变中断标志。

当功能选择控制位 PnSEL 复位时,中断触发沿选择和中断标志位具有完全特性。

**8.2.2 P1、P2 的原理图**

端口 P1 和 P2(见图 8.7)各引脚的逻辑是相同的。每一位都可读写。

当 PnSEL. x 置位选择了模块功能时,输入信号传入引脚对应的外围模块。当 PnSEL. x 复位时,输入模块的信号将保持最后输入信号的电平。当保持的电平与引脚电平不同时,将复位的控制位置位会改变输入模块的信号。



### 8.3 通用端口 P3、P4

通用端口 P3 和 P4 相同,由各个引脚选择的功能来组成其全部功能。每个引脚可以选择按端口特性工作或在内部的外围模块控制下工作。图 8.8 为端口 P3、P4 结构示意图。各有 4 个寄存器控制 P3 和 P4。

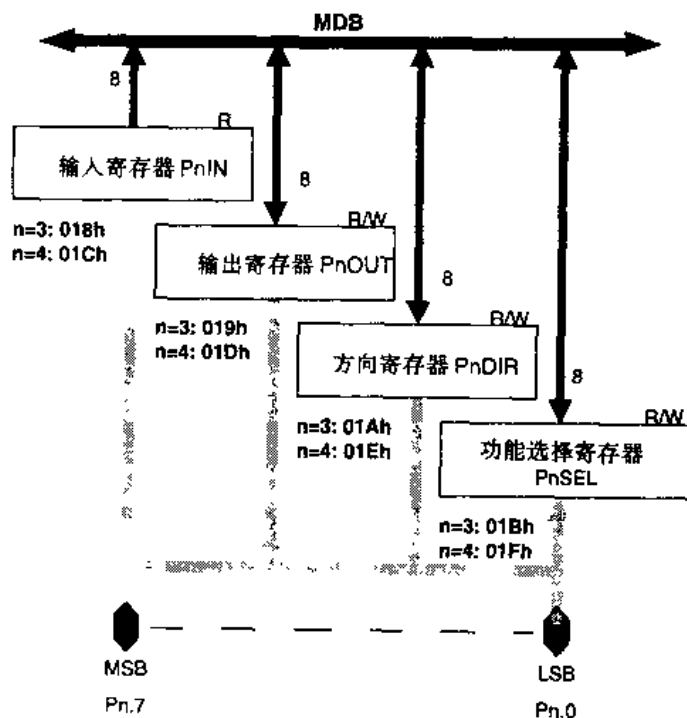


图 8.8 端口 P3、P4 的结构

通用模块寄存器位于安排字节外围模块的低端地址空间。寄存器必须用字节指令以绝对寻址模式进行访问。

#### 8.3.1 P3、P4 的控制寄存器

端口 P3 和 P4 通过 8 位 MDB 和 MAB 与处理机内核相连。它们必须通过字节指令以绝对寻址模式访问。

4 个控制寄存器为数字 I/O 功能的应用提供了最大的灵活性:

- 所有 I/O 位均可独立编程;
- 可以有各种输入组合;
- 可以有各种端口和模块功能的组合。

每个端口的 4 个寄存器是:

寄存器	缩写	寄存器类型	地址	初始状态
输入寄存器	P3IN	只读	018h	---
输出寄存器	P3OUT	读写	019h	不变
方向寄存器	P3DIR	读写	01Ah	复位

端口选择寄存器	P3SEL	读写	01Bh	复位
输入寄存器	P4IN	只读	01Ch	-----
输出寄存器	P4OUT	读写	01Db	不变
方向寄存器	P4DIR	读写	01Eh	复位
端口选择寄存器	P4SEL	读写	01Fh	复位

以上所有寄存器均为 8 位, 必须用字节指令访问。

### 1. 输入寄存器 P3IN、P4IN

这两个只读寄存器扫描 I/O 引脚上的信号。这些引脚应选择为输入方向。

**注意: 对输入寄存器 P3IN、P4IN 写入**

对只读寄存器的写入会在写有效期间增加电流消耗。

### 2. 输出寄存器 P3OUT、P4OUT

如果 I/O 引脚用做输出, 则 8 位的输出寄存器提供相应位的输出缓存信息。输出缓存可用所有包含目的操作数的指令进行修改。在读取时, 输出缓存的内容与引脚定义方向无关。改变方向不会改变输出缓存的内容。

### 3. 方向寄存器 P3DIR 和 P4DIR

每个寄存器含有相互独立的 8 位, 用于定义 I/O 引脚的方向。各位在 PUC 后复位, 即

- 0: I/O 引脚切换成输入模式。
- 1: I/O 引脚切换成输出模式。

### 4. 功能选择寄存器 P3SEL 和 P4SEL

每个寄存器都有相互独立的 8 位来定义 I/O 引脚的功能。端口功能或者一个已定义的模块功能用于在各引脚中输入或输出数据。各位在 PUC 后复位, 即

- 0: 端口功能, 输出或输入数据由端口定义。
- 1: 模块功能, 输出或输入数据由模块定义。

## 8.3.2 P3、P4 的原理图

端口 P3 和 P4 各引脚的逻辑在特殊设备配置中定义。将引脚定义为纯数字端口或兼有数字端口和模块功能。图 8.9 为 P3.x、P4.x 原理图。

只作为数字端口的引脚只受相应的端口寄存器位的控制。

兼有数字端口和模块功能的引脚由以下控制位控制, 即

- 端口控制位(当相应的选择位 PnSEL.x 复位);
- 模块控制位(当相应的选择位 PnSEL.x 置位)。

全部 8 个端口可由硬件分别配置成以下状态:

- 仅作为端口引脚;
- 仅作为模块功能引脚;
- 由软件配置用于端口或模块功能。

具体的实现在器件的数据手册中介绍。

当 PnSEL.x 置位选择了模块功能时, 输入信号经引脚传入对应的外围模块。当 PnSEL.x 复位时, 输入模块的信号将保持最后输入信号的电平。当保持的电平与引脚电平不同时, 将复

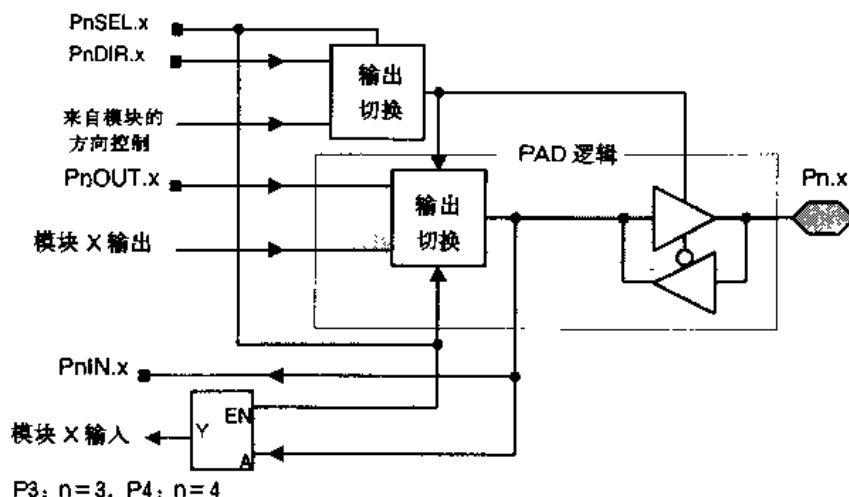
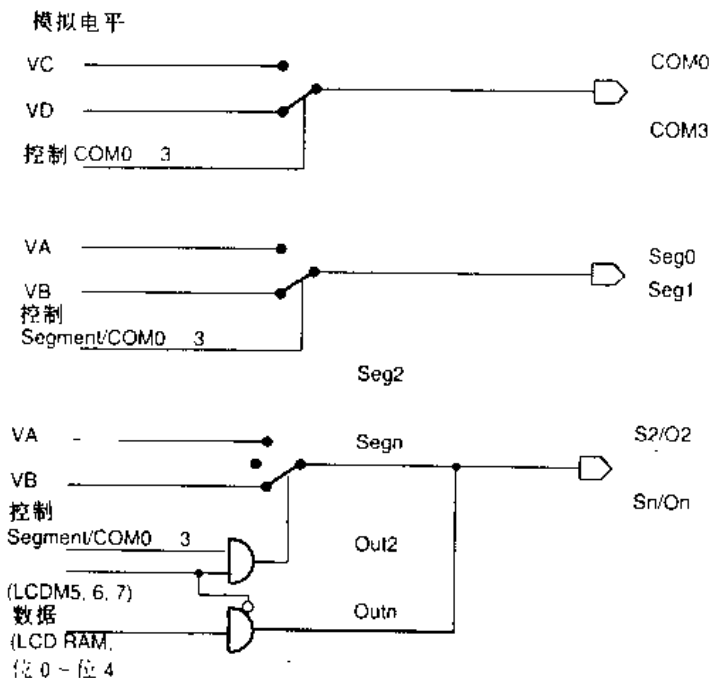


图 8.9 P3.x 和 P4.x 的原理图

位的控制位置位会改变输入模块的信号。

### 8.4 LCD 端口

LCD(Liquid Crystal Display)端口可选择作为 LCD 显示驱动或作为提供静态信号的数字输出。图 8.10 为 LCD 原理图。LCD 的控制用公共端(COM)和段码端(SEG)的输出台阶信号来驱动以 2MUX 或以更高速率复合扫描所需要的模拟信号。



注意：信号 VA、VB、VC 及 VD 来自 LCD 的模拟电压发生器

图 8.10 LCD 原理图



## 第 9 章 通用定时器/端口模块

通用定时器/端口模块(见图 9.1)支持多种系统功能:

- 多达 6 个独立输出;
- 2 个 8 位计数器,可级联成 16 位模式;
- 用于斜率转换原理的 A/D 转换的精密比较器。

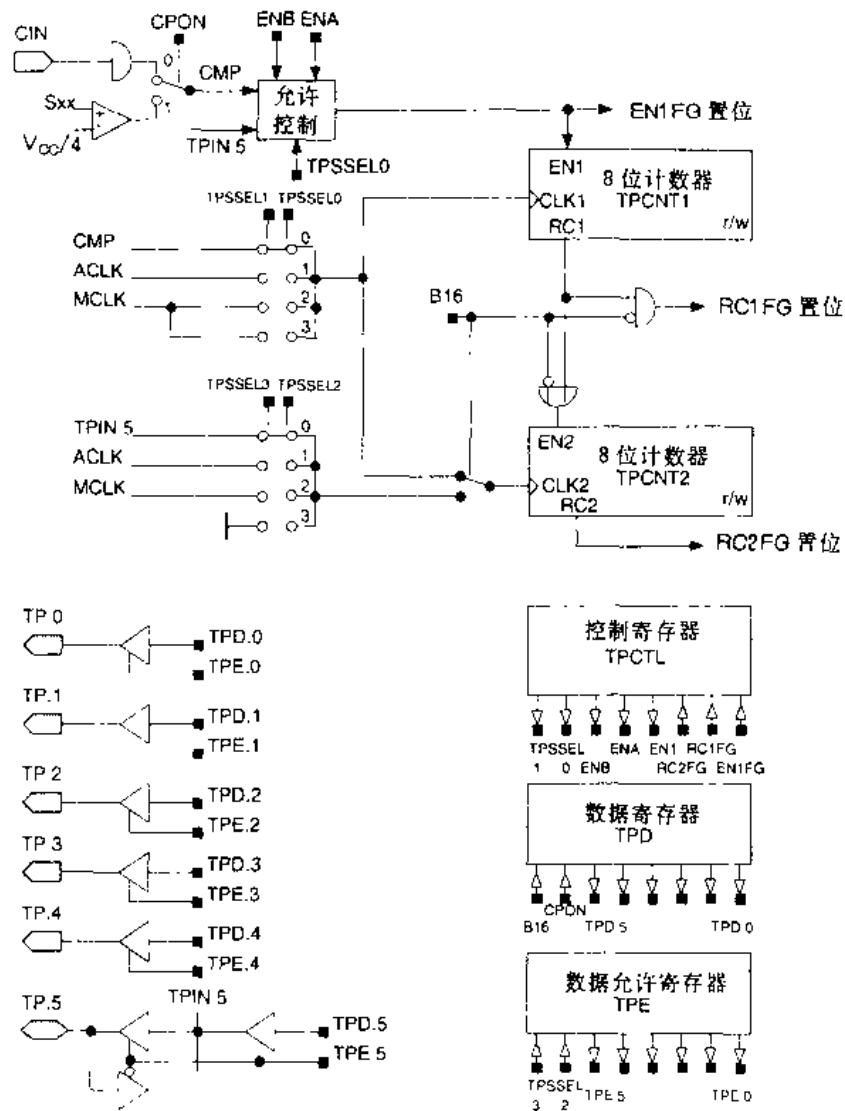


图 9.1 定时器/端口的结构

## 9.1 定时器/端口模块操作

定时器/端口模块可由控制寄存器 TPCNTL 来配置,以实现各种方式的操作。

### 9.1.1 定时器/端口计数器 TPCNT1——8 位操作

计数器 TPCNT1 可用适当的指令读写。当选中 CMP 或 ACLK 时,对定时器数据的读操作可与时钟源异步。用软件作 2 到 3 次采样并作多数表决可保证数据的准确性。

当时钟源是 MCLK 时,读出数据是正确的。因为 MCLK 同样也是指令的时钟频率,所以读出的数据仅是计数器状态的一个样本,并且如果 EN1 置位,则计数器将持续增加。

计数器可在任何时刻写入。如果时钟计数在写操作和读操作之间发生,那么在写入数据后再重新读出,数据可能不同。

计数器可以用 3 种时钟源:MCLK、ACLK 和 CMP。

当计数器的允许输入 EN1 置位时,计数器在输入时钟的上升沿增 1。当 ENA 和 ENB 信号之一或全部置位时,计数器被允许。随着系统复位,这两位也复位,且计数器即被禁止。将它们复位能冻结计数值。

当计数值为 0FFh 时,进位信号 RC1 变高。它的下降沿使寄存器 TPCTL 的 RC1FG 位置位。

RC1FG 在计数器由 0FFh 卷回到 000h 时置位。当 ENA 和 ENB 不作为禁止源而 EN1 切换至禁止时,标志位 EN1FG 置位。当允许位 TPIE 置位且 3 个标志 RC1FG、RC2FG 和 EN1FG 中任一置位时,发生中断请求。标志位 RC1FG、RC2FG 和 EN1FG 必须由软件复位。

### 9.1.2 定时器/端口计数器 TPCNT2——8 位操作

由于允许信号和时钟信号不同,计数器 TPCNT2 和计数器 TPCNT1 的操作也不同。计数器总是以 8 位工作。它有 3 种时钟信号源可供选择:MCLK、ACLK 和 TPIN.5。

当计数值为 0FFh 时,进位信号 RC2 变高。它的下降沿使寄存器 TPCTL 的 RC2FG 置位。

中断标志 RC2FG 在 TPCNT2 由 0FFh 卷回到 000h 时置位。

### 9.1.3 定时器/端口计数器——16 位操作

8 位计数器 TPCNT1 和 TPCNT2 可级联成一个 16 位计数器,如图 9.2 所示。将控制寄存器的 B16 置位实现这一操作。

以字节访问对计数器进行读写操作。对 TPCNT1 和 TPCNT2 数据的读写按顺序进行。当访问是在计数时进行时,这一点要特别考虑。

TPCNT1 的允许位也是 16 位计数器的允许位。TPCNT1 和 TPCNT2 的时钟信号是相同的。选择由 TPSSEL0 和 TPSSEL1 决定。这一模式中 TPSSEL2 和 TPSSEL3 不起作用。



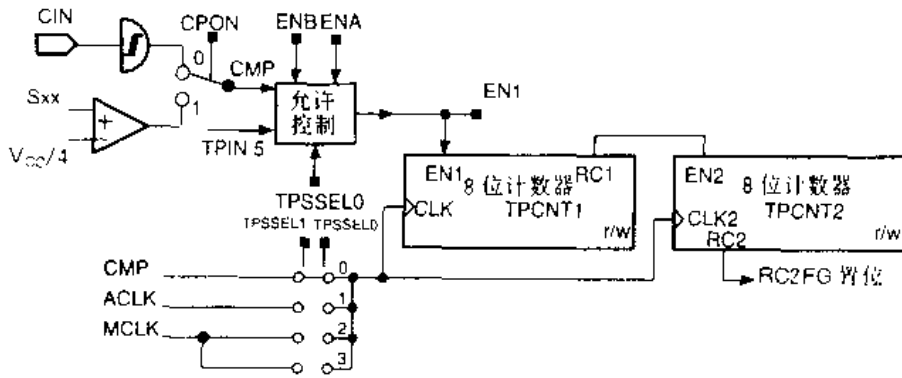


图 9.2 定时器/端口计数器

由 ENA、ENB、TPSSSEL0 和 TPSSSEL1 4 个信号控制级联计数、计数允许和时钟源, 如下所示:

ENB	ENA	TPSSSEL1	TPSSSEL0	EN1	CLK1
0	0	0	0	0	CMP
0	0	0	1	0	ACLK
0	0	1	x	0	MCLK
0	1	0	0	1	CMP
0	1	0	1	1	ACLK
0	1	1	x	1	MCLK
1	0	0	0	$\overline{\text{TPIN.5}}$	CMP
1	0	0	1	$\overline{\text{TPIN.5}}$	ACLK
1	0	1	0	$\overline{\text{TPIN.5}}$	MCLK
1	0	1	1	$\overline{\text{TPIN.5}}$	MCLK
1	1	0	0	$\overline{\text{CMP}}$	CMP
1	1	0	1	$\overline{\text{CMP}}$	ACLK
1	1	1	0	$\overline{\text{CMP}}$	MCLK
1	1	1	1	$\overline{\text{CMP}}$	MCLK

伴随引脚 CIN、Sxx 的信号或 3 个时钟 ACLK、MCLK、CMP 之一, 可使计数器停止或计数。将 TPIN.5、反向 TPIN.5、CMP 或反向 CMP 信号作用于计数器允许信号 EN1, 16 位计数器便会在每个 ACLK 或 MCLK 周期增 1。这一特性可用于测量 TPIN.5 或 CMP 引脚信号的宽度。进位信号 RC2 在计数值为 0FFFFh 时置位。

当计数器由 0FFFFh 卷回到 00000h 时, 标志位 RC2FG 置位。当允许信号 EN1 禁止时, 标志位 EN1FG 置位。允许信号 EN1 的源是 TPIN.5 或 CMP 信号。当经软件用 ENA 和 ENB 禁止 EN1 时, EN1FG 不会置位。

## 9.2 定时器/端口寄存器

定时器/端口模块的硬件是字节结构, 须由字节指令(后缀“B”)访问。

寄存器	缩写	寄存器类型	地址	初始状态
TP 控制寄存器	TPCTL	读写	04Bh	复位
TP 计数器 1	TPCNT1	读写	04Ch	不变
TP 计数器 2	TPCNT2	读写	04Dh	不变
TP O/P 数据寄存器	TPD	读写	04Eh	复位
TP 数据允许寄存器	TPE	读写	04Fh	复位

### 1. 定时器/端口控制寄存器

控制寄存器中的信息决定定时器/端口模块的操作。定时器/端口控制寄存器的位定义如下:

TPCTL (04Bh)

7				0			
TPSSEL1	TPSSEL0	ENB	ENA	EN1	RC2FG	RC1FG	EN1FG
rw-0	rw-0	rw-0	rw-0	r-0	rw-0	rw-0	rw-0

对 TPCTL 的各状态位说明如下:

位 0: 如果允许信号来自 CMP 或 TPIN.5, 则计数器 TPCNT1 的允许信号 EN1 的下降沿使允许标志 EN1FG 置位。EN1FG 由事件置位, 须由软件复位; 否则将保持置位。EN1FG 可用在定时器/端口中断服务程序中, 以确定是允许位 EN1 还是进位标志 RC 在计数器从 0FFh 卷回到 000h 时置位引起中断。

位 1: 计数器 TPCNT1 从 0FFh 卷回到 000h(溢出)由 RC1FG 指示。这一事件使 RC1FG 置位, 同时它必须由软件复位; 否则将保持置位。  
可用于中断服务程序中判断中断事件来源。

位 2: 计数器 TPCNT2 从 0FFh 卷回到 000h(溢出)由 RC2FG 指示。这一事件使 RC2FG 置位, 同时它必须由软件复位; 否则将保持置位。  
可用于中断服务程序中判断中断事件来源。

位 3、4、5: 计数器 TPCNT1 的允许信号 EN1 可读。它的值由 ENA、ENB、TPSSEL0 决定。

ENB	ENA	TPSSEL0	EN1
0	0	x	0
0	1	x	1
1	0	0	$\overline{\text{TPIN.5}}$
1	0	1	TPIN.5
1	1	0	$\overline{\text{CMP}}$
1	1	1	CMP

位 6、7: TPSSEL0 和 TPSSEL1 控制选择计数器 TPCNT1 的 3 个时钟源之一。

TPSSEL1	TPSSEL0	CLK1
0	0	CMP
0	1	ACLK
1	x	MCLK





当选择8位计数器模式时,有3个中断源可能请求中断服务,即EN1下降沿、TPCNT1溢出(RC1)、TPCNT2溢出(RC2)。在16位计数器模式(B16置位)中,有2个中断源可能请求中断服务,即EN1下降沿、TPCNT2溢出。

中断请求位必须在中断程序中复位;否则,在GIE置位或RETI指令执行后立即又会发生中断请求。在PUC后定时器/端口中断允许位TPIE复位,这时就不会发生中断请求。当TPIE和GIE置位而又不发生PUC和NMI时,系统/CPU会执行中断服务。

## 9.4 定时器/端口在A/D中的应用

定时器/端口模块可为电阻或电容式传感器提供A/D转换功能。

对于温度测量,最常见的传感器是具有正、负温度系数的热敏电阻。硅传感器、NTC电阻和铂电阻都是这类传感器。

### 9.4.1 R/D转换原理

测量一个电容的放电时间可以将电阻值转变为数字信号。该电容在放电前先行充电。图9.5所示为RC充放电时序图。

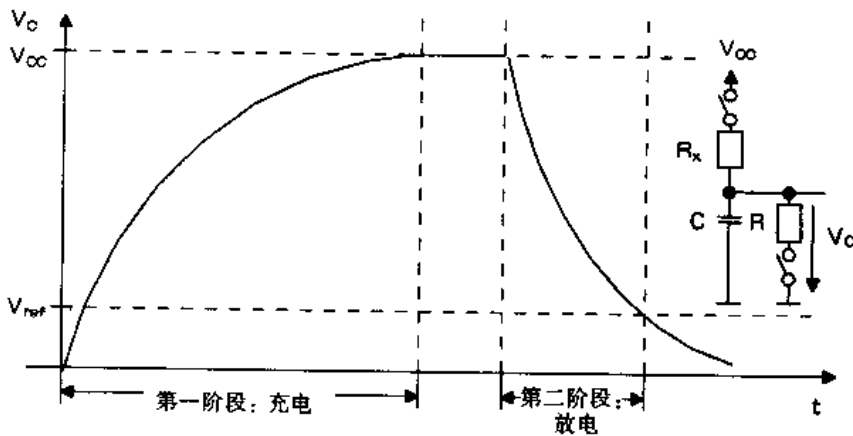


图 9.5 RC 充放电时序图

用电压比较器和计数器来测量电容放电至参考电压值  $V_{ref}$  的时间  $t$ 。由于比较器的输出作为计数器的允许信号,只要电容  $C$  的电压在  $V_{ref}$  之上,计数器就会一直在增加。

这种时间测量方法的原理公式如下:

$$t = -R \times C \times \ln(V_{ref}/V_{CC})$$

$$t = N \times t_{clock}$$

$$N \times t_{clock} = -R \times C \times \ln(V_{ref}/V_{CC})$$

$$N = -R \times C \times f_{clock} \times \ln(V_{ref}/V_{CC})$$

式中:  $R$ ——电阻值;

$C$ ——电容值;

$V_{CC}$ ——电源电压;

$T_{clock}$ 、 $f_{clock}$ ——分别为时钟的周期和频率;

$N$ ——电容放电时间的时钟周期数。

其中  $C$ 、 $f_{\text{clock}}$  和  $V_{\text{ref}}/V_{\text{CC}}$  已知即可确定阻值。利用一个稳定的参考电阻作变换(见图 9.6), 传感器测量可由下式得出。

$$N_{\text{meas}}/N_{\text{ref}} = (-R_{\text{meas}} \times C \times \ln(V_{\text{ref}}/V_{\text{CC}}))/(-R_{\text{ref}} \times C \times \ln(V_{\text{ref}}/V_{\text{CC}}))$$

$$N_{\text{meas}}/N_{\text{ref}} = R_{\text{meas}}/R_{\text{ref}}$$

$$R_{\text{meas}} = R_{\text{ref}} \times (N_{\text{meas}}/N_{\text{ref}})$$

式中:  $R_{\text{ref}}$ 、 $R_{\text{meas}}$ ——分别为参考电阻与待测电阻的电阻值;

$N_{\text{ref}}$ 、 $N_{\text{meas}}$ ——分别为参考电阻和待测电阻所对应的电容放电时间的时钟周期数。

其中, 假定电路用同一个电容, 在两个测量阶段的电压和时钟周期是相同的。

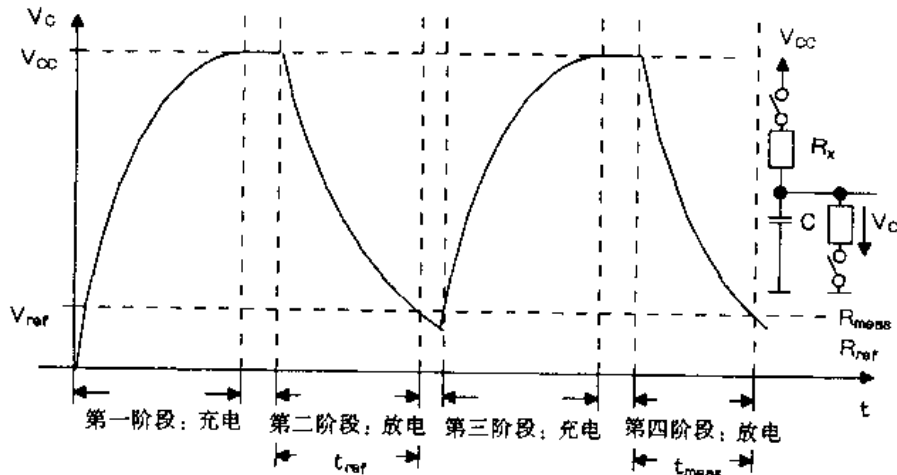


图 9.6 在 R/D 变换中利用  $R_{\text{ref}}$  和  $R_{\text{meas}}$  的充放电时序图

如图 9.6 所示, 在阶段 I 和 III, 电容  $C$  通过电阻  $R_x$  充电至  $V_{\text{CC}}$ 。然后通过  $R_{\text{ref}}$  或  $R_{\text{meas}}$  放电。电流受阻值限制。如果只用  $R_{\text{ref}}$ , 则电容的放电时间就可以精确测定。图 9.7 为转换原理电路图。

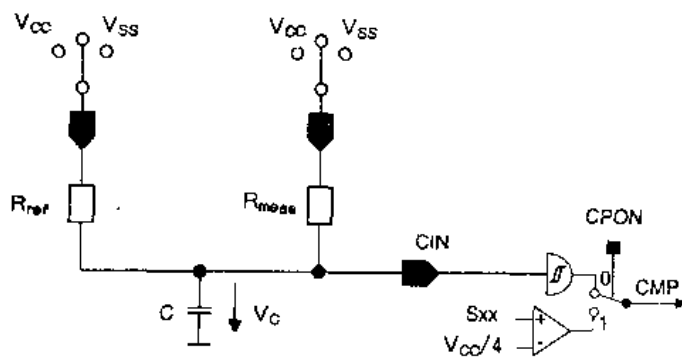


图 9.7 转换原理电路图

当其中一个电阻接至  $V_{\text{SS}}$  时电容  $C$  放电。所有的寄生放电通路都会影响电容放电至  $V_{\text{ref}}$  的时间。

### 9.4.2 分辨率大于8位的转换

转换由比较器、16位计数器和数字输出来实现。图9.8为一个ADC应用实例原理图。

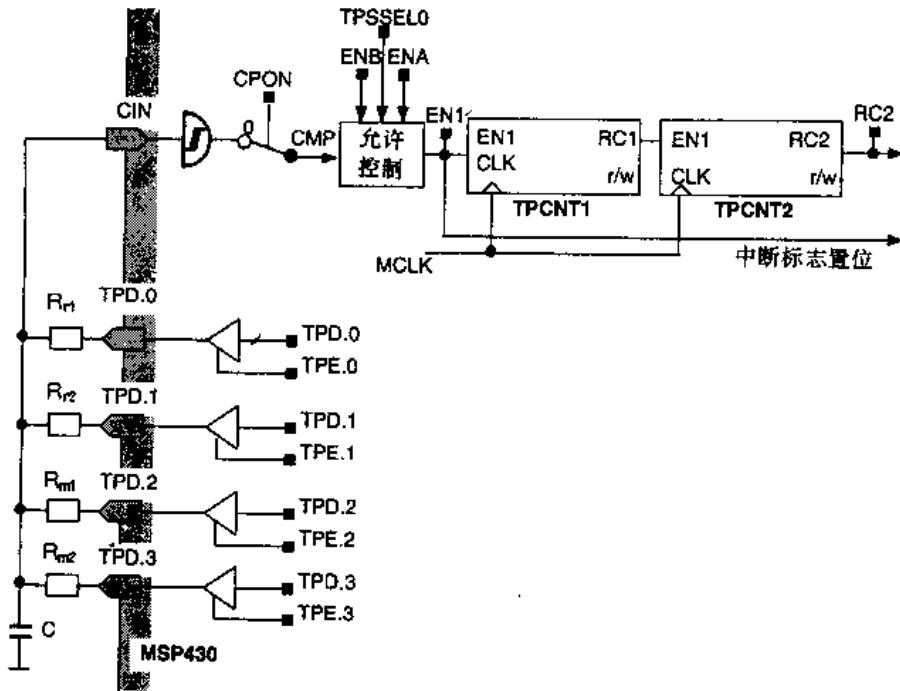


图 9.8 ADC 应用实例

在实例中,外部元件是2个参考电阻  $R_{r1}$  和  $R_{r2}$  和2个待测电阻  $R_{m1}$  和  $R_{m2}$ 。

用控制寄存器 TPCTL 来配置图 9.8 中的电路。图 9.8 中电路已简化为只有有效连接和功能模块。控制寄存器 TPCTL 写入 9Eh, 即 B16、TPSSEL1、TPSSEL0、ENB 和 ENA 置位。

电容 C 通过  $R_{r1}$  与/或  $R_{r2}$  充电至  $V_{CC}$ 。

4 个放电阶段的时间计算公式如下:

$$t_{r1} = N_{r1} \times T_{MCLK} = -R_{r1} \times C \times \ln(V_{ref}/V_{CC})$$

$$t_{r2} = N_{r2} \times T_{MCLK} = -R_{r2} \times C \times \ln(V_{ref}/V_{CC})$$

$$t_{m1} = N_{m1} \times T_{MCLK} = -R_{m1} \times C \times \ln(V_{ref}/V_{CC})$$

$$t_{m2} = N_{m2} \times T_{MCLK} = -R_{m2} \times C \times \ln(V_{ref}/V_{CC})$$

电阻  $R_{m1}$  和  $R_{m2}$  的计算公式如下:

$$(R_{r1} - R_{r2}) / (R_{mx} - R_{r2}) = (N_{r1} - N_{r2}) / (N_{mx} - N_{r2})$$

$$R_{mx} = R_{r2} + (R_{r1} - R_{r2}) \times (N_{mx} - N_{r2}) / (N_{r1} - N_{r2})$$

$$R_{m1} = R_{r2} + (R_{r1} - R_{r2}) \times (N_{m1} - N_{r2}) / (N_{r1} - N_{r2})$$

$$R_{m2} = R_{r2} + (R_{r1} - R_{r2}) \times (N_{m2} - N_{r2}) / (N_{r1} - N_{r2})$$

## 第 10 章 定时器

### 10.1 Basic Timer1

Basic Timer(见图 10.1)工作的目的是支持软件和各种外围模块工作在低频率、低功耗条件下。

下面是软件功能受到晶振稳定性控制的一些例子：

- 实时时钟 RTC；
- 键盘去抖动；
- 软件时间加一特性。

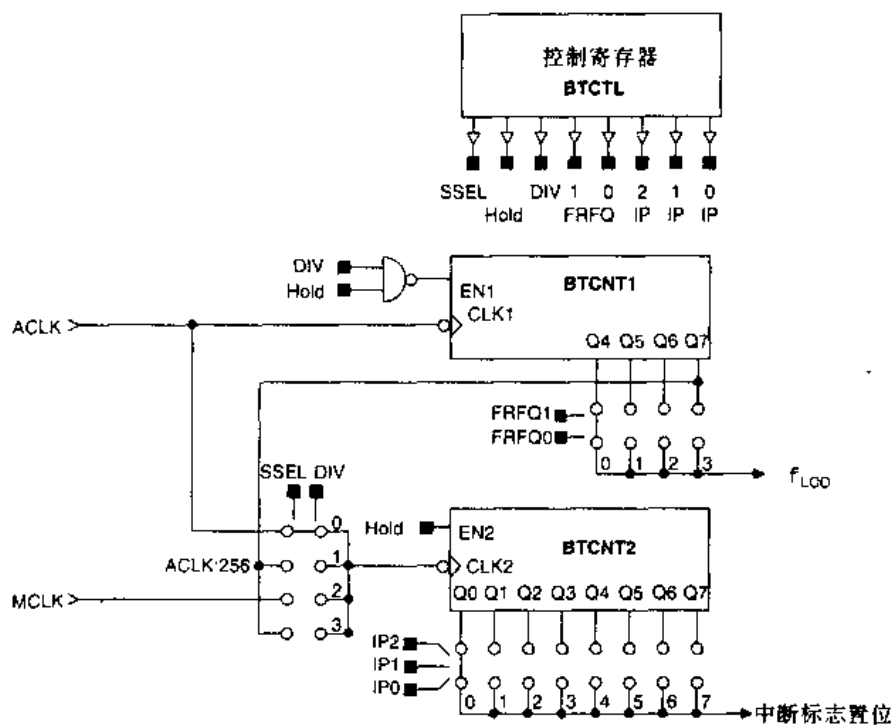


图 10.1 Basic Timer 结构

Basic Timer 向别的外围模块提供低频控制信号。软件可以访问 2 个 8 位计数器。

控制寄存器 BTCTL 的标志位控制或选择不同的工作模式。寄存器 BTCTL、8 位计数器 BTCNT1 和 BTCNT2 可用软件控制。当系统上电复位、看门狗溢出或其他情况发生时，寄存器各位保持无定义或不变状态。用户程序通常在初始化时定义 Basic Timer 的工作状态。

#### 10.1.1 Basic Timer1 寄存器

Basic Timer1 模块的硬件是字节结构的，必须用字节指令访问。





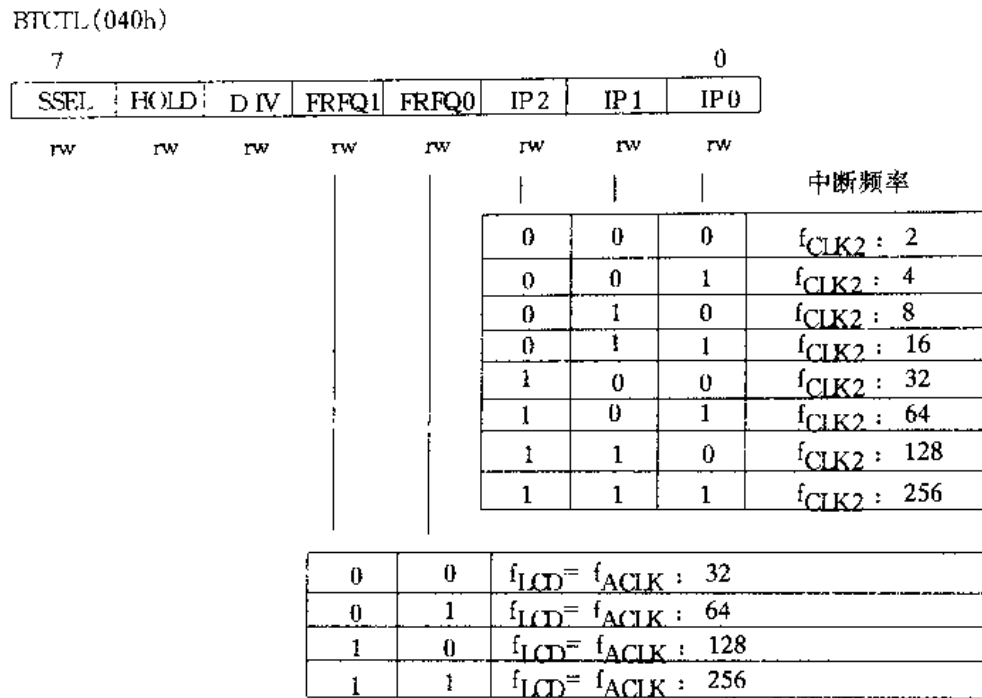
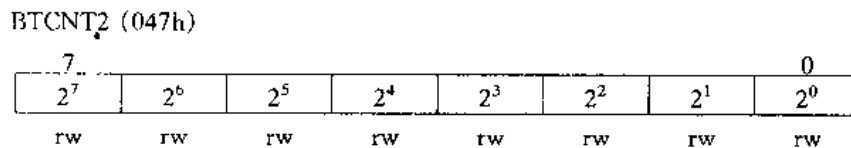


图 10.2 Basic Timer1 寄存器功能

周期可以由 BTCTL 中的 IP0 ~ IP2 位中选择 8 个触发器输出之一。BTCNT2 的结构如下:



计数器 Q0 ~ Q7 的输出可读出, 并且能用软件写入。

### 10.1.2 SFR 位

SFR 的各位根据 Basic Timer 的如下选定功能, 实现与系统的交互控制, 即

- Basic Timer 中断标志 BTIFG(位于 IFG2.7);
- Basic Timer 中断允许位 BTIE(位于 IE2.7)。

HOLD 位禁止模块的所有功能, 并把功耗降低到最低程度, 即只有漏电流。

当计数器被允许或禁止时不会发生额外计数。系统对通用模块寄存器 BTCTL 的访问不会影响计数。它可以用通常的方式进行读写操作。

中断标志和中断允许位遵循一般的模块中断规则。除了受各自的中断允许控制外, 中断请求还受控于通用中断允许位 GIE。PUC 使中断允许标志 BTIE 复位。当 Basic Timer 的中断请求被接受时, 中断标志 BTIFG 复位。

### 10.1.3 Basic Timer1 的操作

Basic Timer 总是按时钟 ACLK 或 MCLK 作增计数。SSEL 控制信号选择辅助时钟 ACLK 或主时钟 MCLK(系统时钟  $f_{system}$ ) 作为 BTCNT2 的时钟。

中断可用做系统的控制, 它是单源中断。

Basic Timer 可以工作在两种不同模式下:

- 2 个独立的 8 位定时器/计数器;
- 1 个 16 位定时器/计数器。

### 1. 8 位计数器模式

在 8 位计数器模式中, BTCNT1 总是按 ACLK 作增计数。计数器读出时必须考虑计数器时钟(ACLK)和系统时钟(MCLK)的异步特性。计数器可用软件进行与计数时钟异步的写操作。

BTCNT2 时钟信号可由 SSEL 和 DIV 从 MCLK、ACLK 和 ACLK/256 中选择之一。BTCNTL 按选定时钟作增计数。

8 个定时器输出之一可用于对 Basic Timer 中断标志置位。当选择 ACLK 或 ACLK/256 为时钟时, 读写访问可以是异步的。

### 2. 16 位计数器模式

DIV 置位选择 16 位定时器/计数器模式。这时, BTCNT1/BTCNT2 时钟源是 ACLK 信号。

HOLD 位停止这 2 个 8 位计数器工作。

## 10.1.4 Basic Timer1 的操作——LCD 时钟信号 $f_{LCD}$

LCD 外围模块用  $f_{LCD}$  产生 COM 和 SEG 行的时序信号,  $f_{LCD}$  由 ACLK 产生。当用 32 768 Hz 晶振时,  $f_{LCD}$  可以是 1 024 Hz、512 Hz、256 Hz 或 128 Hz。用 FRFQ1 和 FRFQ2 可正确选择帧频。合适的  $f_{LCD}$  频率取决于 LCD 每帧的字符数据量和 LCD 扫描重复速率。图 10.3 是 LCD 在 3MUX 扫描时的频率选择例子。

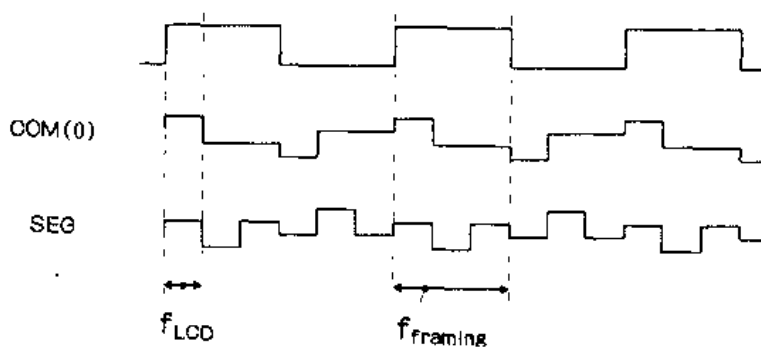


图 10.3 为 LCD 选择的频率(3MUX 的例子)

3MUX 举例如下:

LCD 数据:

$$f_{\text{framing}} = 100 \text{ Hz}, \dots, 30 \text{ Hz}$$

频率:

$$f_{\text{LCD}} = 6 \times f_{\text{framing}}$$

$$f_{\text{LCD}} = 6 \times 100 \text{ Hz}, \dots, 6 \times 30 \text{ Hz} = 600 \text{ Hz}, \dots, 180 \text{ Hz}$$

可选  $f_{\text{LCD}}$ :

$$1\,024 \text{ Hz}, 512 \text{ Hz}, 256 \text{ Hz} \text{ 或 } 128 \text{ Hz}$$

$$f_{\text{LCD}} = 256 \text{ Hz}, \text{FRFQ1} = 1, \text{FRFQ0} = 0$$

## 10.2 8 位间隔定时器/计数器

8 位间隔定时器支持 3 个主要功能：

- 串行通信或数据交换；
- 脉冲计数或脉冲累加；
- 定时器。

图 10.4 为 8 位定时器/计数器原理图。

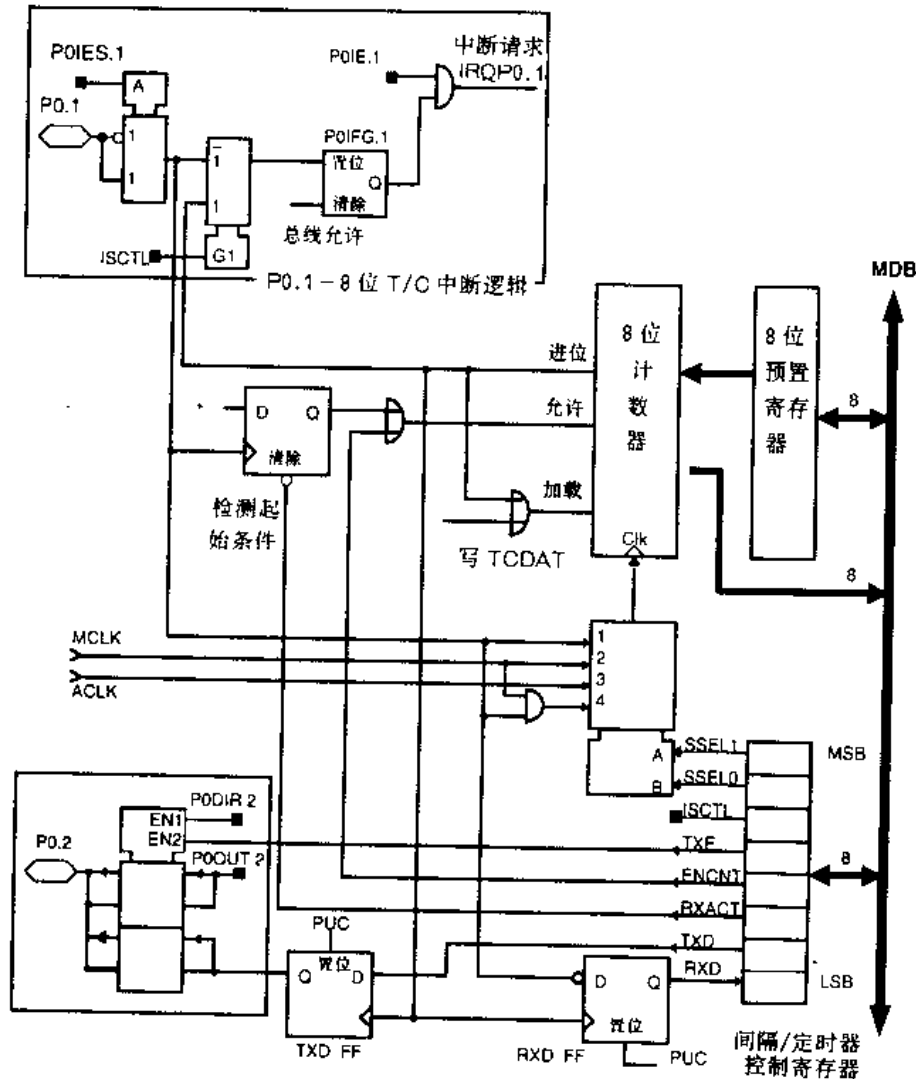


图 10.4 8 位定时器/计数器原理图

### 10.2.1 8 位定时器/计数器的操作

8 位定时器/计数器包括以下主要模块：

- 8 位带预置数寄存器的增计数器；
- 8 位控制寄存器；
- 输入时钟选择器；

- 触发沿检测, 例如异步通信协议中的起始位;
- 输入和输出数据锁存, 由 8 位计数器的进位信号触发。

### 1. 8 位带预置数寄存器的增计数器

8 位计数器由控制寄存器的 2 位 (SEL0、SEL1) 选定的输入时钟作增计数。计数器的 2 个输入 (Load、Enable) 控制它的操作。图 10.5 为 8 位计数器原理图。

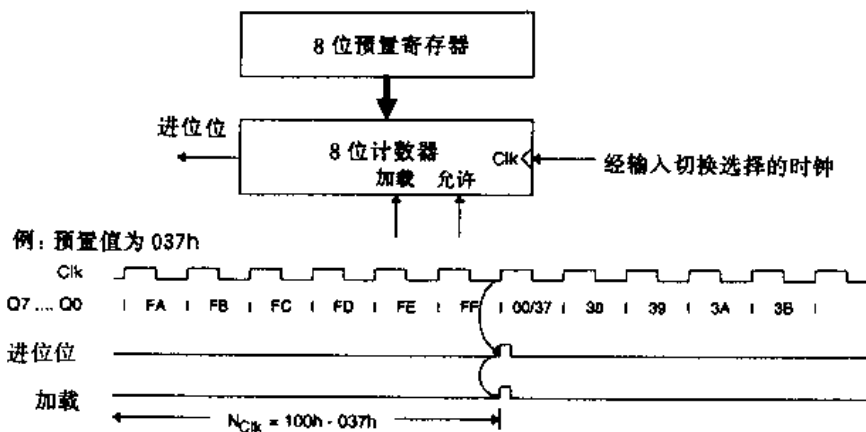


图 10.5 8 位计数器原理图

两输入之一控制置数功能。置数操作将预置数寄存器中的数据装入计数器。对计数器的写操作则把预置数寄存器的内容写入计数器。

所有指令均可对预置数寄存器进行读写操作。预置数寄存器相当于一个缓存器, 并且可以在计数器完成预置后立即进行写操作。

另一个输入则允许计数操作。当允许信号置位时, 计数器在每个输入时钟的上升沿加 1。

### 2. 8 位控制寄存器

8 位控制寄存器的内容用于选择定时器/计数器的工作方式和控制各种功能。

### 3. 输入时钟选择器

用 8 位控制寄存器提供的 2 个信号选择 8 位增计数器的时钟源。这 4 个时钟源是 MCLK、ACLK、由 P0.1 引脚进入的外部信号以及由 P0.1 引脚和 MCLK 进行逻辑与产生的信号。

### 4. 边沿检测

类似 UART 协议的串行协议, 需要检测起始位边沿, 以确定接收到的发送数据的起始位。

### 5. 输入和输出数据锁存、RXD 触发器和 TXD 触发器

将数据锁存入输入和输出数据锁存器的时钟信号是 8 位计数器的进位信号。两个锁存器在使用时像 1 位缓存器, 并且其输出按预先定义时序变化。

## 10.2.2 8 位定时器/计数器的寄存器

定时器/计数器模块硬件通过 8 位 MDB 和 MAB 受 CPU 控制。它必须用字节指令访问。

寄存器	缩写	类型	地址	初始状态
T/C 控制寄存器	TCCTL	读写	042h	复位
预置数寄存器	TCPLD	读写	043h	不变



;

预置数寄存器 TCPLD 可以用地址 043h 访问。

### 3. 8 位计数器数据

8 位计数器的数据可以用地址 044h 读出。对计数器进行写操作将把预置数寄存器中的数据而不是指令中的数据装入计数器。

### 10.2.3 与 8 位定时器/计数器有关的 SFR 位

8 位定时器/计数器没有独立的中断标志位,它和 P0 端口共享中断标志位。由 TCCTL 中的 ISCTL 位选择中断标志的中断源。

用 P0/RXD.1 信号或 8 位计数器的进位作为中断标志的中断源。用 SFR 中的 2 位和 P0 地址帧中的 1 位来处理 P0/RXD.1 的中断事件:

- P0/RXD.1 中断允许位: P0IE.1(位于 IE1.3, 初始状态为复位);
- P0/RXD.1 中断边沿选择位: P0IES.1(位于 P0IES, 初始状态为复位)。

这一中断标志是单源中断标志,它在中断服务时自动复位,而允许位和边沿选择位保持不变。

### 10.2.4 8 位定时器/计数器在 UART 中的应用

定时器/计数器外围模块具有支持用软件进行串行数据交换的特性。数据交换包括接收周期和发送周期。外围硬件支持半双工协议。

软件操作可以分为 3 个类型以支持各种应用的不同情况和不同需求:

- 在每个接收周期后立即控制位信息;
- 在每个接收周期后立即控制一帧的所有位信息;
- 在接收周期完成后将数据帧存入内存并校验。

#### 1. 异步通信 UART 协议

UART 协议是串行位流协议。如图 10.6 所示,它的帧包括起始位、1 ~ 8 个数据位、1 个可选择校验位、1 个可选择地址位或 1 ~ 2 个停止位。最低位在起始位后首先发出。

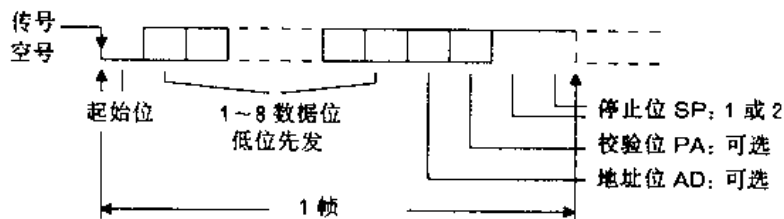


图 10.6 异步通信格式

#### 2. UART 协议接收模式

定时器/计数器作为定时器工作,它的进位锁存引脚 P0.1 上的位信息。下降沿,即起始位的传号至空号的跳变表明一帧的开始。对每 1 位在位的中间扫描读取。图 10.7 为异步通信帧中位的扫描示意图。

软件 UART 紧密结合了接收周期的开始和波特率。如果定时器的时钟频率不是波特率

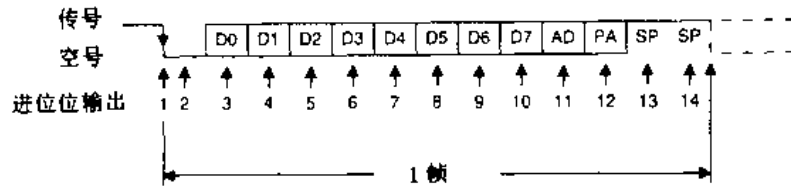


图 10.7 异步通信帧中位的扫描

的整数倍,则各位的读取时序会变化。本节的例子工作在波特率为 2 400 和晶振频率为 32 768 Hz 的条件下。结果是每一位都有自己的间隔时序,因此有各自的预置值。

### 3. UART 协议发送模式

定时器/计数器作为定时器工作,它的进位控制寄存器 TCCTL 中 TXD 位的锁存。软件 UART 将为 TXD 位装入要从 P0.2 引脚发送的值。定时器的第二个进位把 TXD 位送到 TXD 触发器。当控制寄存器中的 TXE 位置位时,数据允许从 P0.2 脚发送。TXE 的复位则禁止缓存器与 P0.2 连接,同时,使 TXD 触发器输出置位。它相当于 UART 格式中定义的传号状态。图 10.8 为异步通信帧中位的发送示意图。

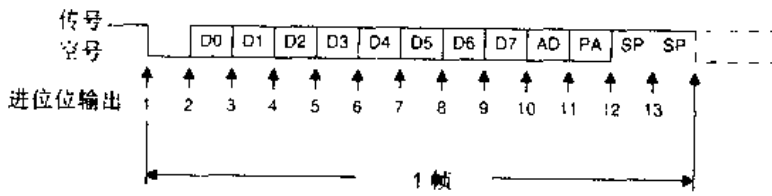


图 10.8 异步通信帧中位的发送

软件 UART 的发送时序取决于波特率。如果定时器的时钟频率不是波特率的整数倍,则各位的发送时序会变化。本节的例子工作在波特率为 2 400 和晶振频率为 32 768 Hz 的条件下。结果是每一位都有自己的间隔时序,因此有各自的预置值。

每种通信都要有识别数据传输中发生错误的能力。有 4 种出错状态,其定义如下:

- 校验错;
- 溢出错;
- 帧错;
- 打断检测(break detect)。

除这些传统的原理外,还可选择一个支持通信协议处理的功能,即识别多帧数据块的起始和报文的目的地。工业标准采用了两种不同的方式进行识别,即线路空闲多处理机协议和地址位多处理机协议。

### 4. 线路空闲多处理机模式的格式

数据块被空闲时间分隔。在字符第一个停止位后收到 10 个以上传号状态即检测到线路空闲。当采用两位停止位时,第二个停止位被看做线路空闲的第一个传号。线路空闲后收到的第一个字符是地址字符。图 10.9 是接收器检测有一个或两个停止位的线路空闲周期。

推荐发送 11 位而不是 10 位的空闲周期。

精确的空闲周期可以得到有效的地址字符识别。多帧数据块的第一个字符可以被识别为



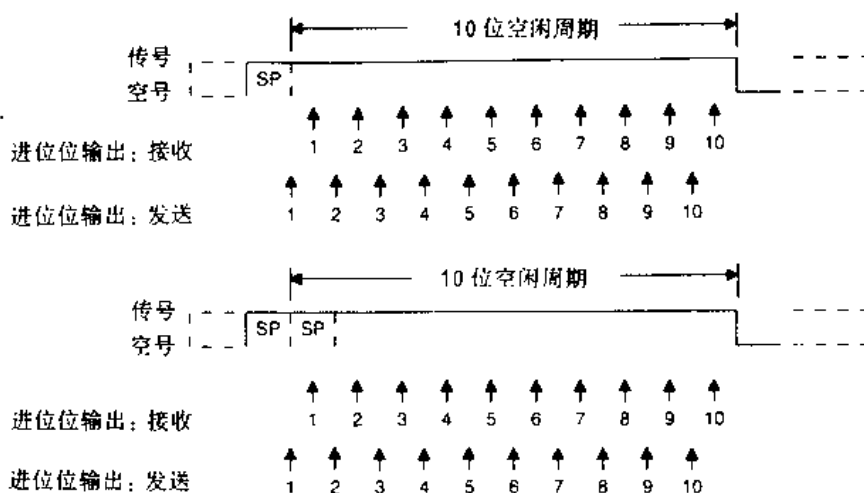


图 10.9 UART 空闲周期

地址。数据块中的帧间空闲周期不应超过检测 10 位空闲周期的时间。图 10.10 为线路空闲多处理协议示意图。

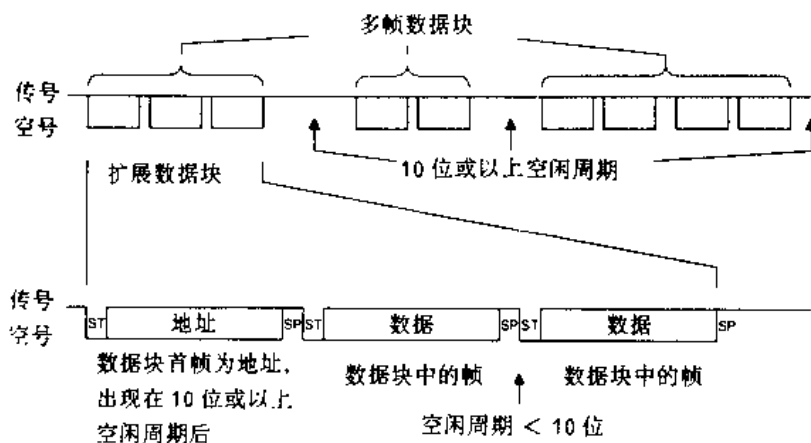


图 10.10 线路空闲多处理协议

### 5. 地址位多处理机模式格式

每个字符额外包含 1 位作地址识别。数据块(报文)的第一个字符地址位被置位。这表明这个字符是地址。图 10.11 为地址位多处理机模式的格式。

### 6. 发送/接收应用实例

这个程序实例采用的是 8 位定时器/计数器特性的串行异步通信协议。它有以下特点：

- 波特率为 2 400；
- $f_{ACLK} = 32\ 768\ \text{Hz}$ ；
- 偶校验；
- 2 个停止位；
- 半双工。

定时器/计数器的进位信号取代 P0.1 信号作为中断源。相应的中断向量中包含发送/接收中断处理程序。程序的第一条指令控制程序进入发送接收部分,并用 TXRX 作为工作模式

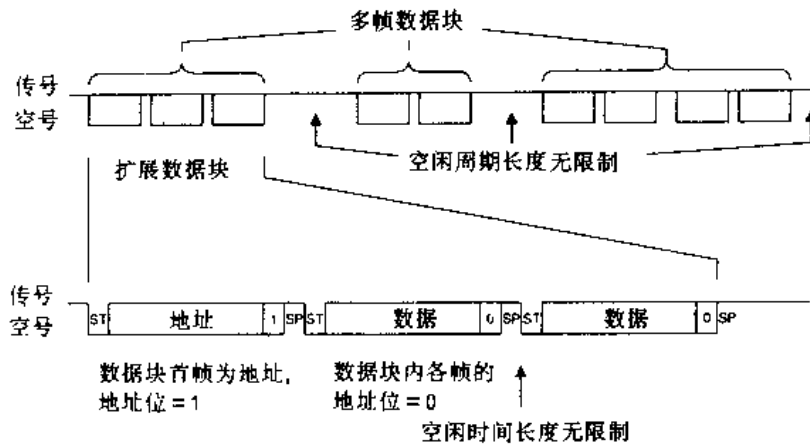


图 10.11 地址位多处理机模式的格式

指示。

```

; ----- 定义中断向量 -----
.SECT      "RXTX_vec", 0FFF8h      ; P0.1 或 8 位定时器/计数器的进位的向量
.Word     VECRXTX                  ; UART 处理程序的地址
...
    
```

两个进位信号间的时间间隔对每次接收或发送周期都是不相同的。选定波特率 2 400 和晶振频率 32 768 Hz 则要求除数为  $32\ 768 / 2\ 400 = 13.67$ 。用 14 - 13 - 14 序列作为除数来实现这一理想的除数。

```

; 符号定义
RXD      .EQU      1                ; TCCTL 的接收数据位
TXD      .EQU      2                ; TCCTL 的发送数据位
RXACT    .EQU      4
ENCNT    .EQU      8                ; TCCTL 的计数器允许位
TXE      .EQU      010h             ; 1: TX 缓存有效
                                              ; 0: TX 缓存输出为三态

ISCTL    .EQU      020h
TCCTL    .EQU      042h             ; 定时器/计数器控制寄存器地址
TCPLD    .EQU      043h             ; 定时器/计数器预置数寄存器地址
TCDAT    .EQU      044h             ; 定时器/计数器地址
BitTime1 .EQU      0100h - 0Eh      ; (14/32 768) s = 427.2 μs, 位长
BitTime1_2 .EQU    0100h - 07h      ; 半位长
BitTime2 .EQU      0100h - 0Dh      ; (13/32 768) s = 396.7 μs, 位长
AdP0_0   .EQU      0h               ; 中断允许 1 寄存器地址(SFR)
IEnP0_0  .EQU      08h             ; 中断允许 1 寄存器位(SFR)
ParVal   .EQU      0h               ; 校验位, 选择偶校验

;
(P0.1 及 TC 中断允许)
; 用于数据处理的寄存器或 RAM
Rcstatus .EQU      0200h            ; RAM (或寄存器), 保存接收序列状态
    
```

TXStatus	.EQU	0201h	; RAM (或寄存器), 保存发送序列状态
TXData	.EQU	R6	; 发送数据寄存器
RCDData	.EQU	R6	; 接收数据(RXD)保存到高字节
Parity	.EQU	0yyyh	; LSB 为校验位状态。用初值决定奇或偶校验
Bend	.EQU	2 x 12	

在程序的主循环中,用表的数据作为输出并把接收数据放入表中来演示发送和接收数据序列的功能。

```

; ----- 帧的发送: 输出表中数据 -----
; Ry 为表指针
;
      MOV      # Table2, Ry      ; 将发送数据表首地址复制到 Ry
L $ 5  CRL.B   & TXStatus
      CMP     # TabEnd + 1, Ry  ; 所有帧已发送了吗?
      JEQ    TabFin            ; 是, 发送停止, 程序继续执行
      MOV.B  @Ry +, TXData     ; 发送信息送入 TXData
      CALL   # TXInit          ; 否, 发送初始化
TXStat CMP     # Bend, TXStatus ; 一帧输出完成?
      JEQ    L $ 5             ; 是, 发送表中下一数据
      JMP    TXStat           ; 否, 等待发送完成
      ...
      ...

Table2 .      0xxh Byte      ; 发送数据表首地址
      ...
      ...

TabEnd .      0zzh Byte      ; 发送数据表末地址
      ...
      ...

TabFin ...                ; 表中数据发送完成, 程序由此继续执行
      ...

; ----- 准备接收一帧 -----
; Rx 为表指针
;
      MOV     # Table1, Rx     ; 将接收数据表首地址复制到 Rx
      CALL   # RCPrep         ; 接收下一帧
      ...

; ----- 接收帧的处理: 将帧存入表中 -----
RECCMPL RLA    RCDData        ; 高字节调整(取消校验位)
      SWPB   RCDData          ; 交换到低字节
      MOV.B  RCDData, 0(Rx)   ; 存入表中
      INC   Rx                ; 表指针增加
      CALL  # RCInit          ; 准备接收下一帧
      ...
      ... ; 继续执行背景程序

```

### 7. 发送模式应用实例

波特率为 2 400, ACLK, 8 位数据, 偶校验, 2 个停止位。

发送模式用了 8 位定时器/计数器、预置数寄存器、控制寄存器、时钟选择和 TXD 数据锁存等功能, 如图 10.12 所示。

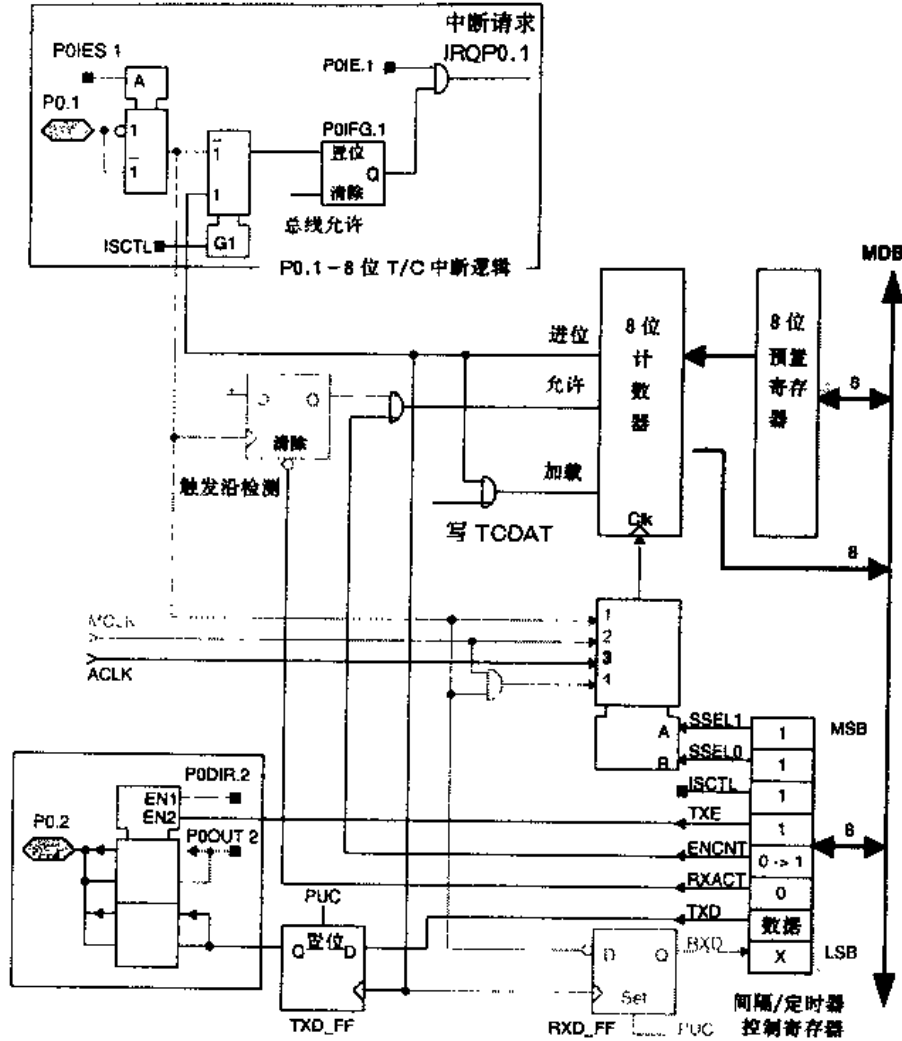


图 10.12 8 位定时器/计数器结构在波特率为 2 400、ACLK 时钟下的发送应用实例

在开始串行通信字符发送前, 有如下一些工作状态须定义, 即

- 允许输出缓存 → TXE 置位;
  - 选定输入 8 位定时器的时钟 → 用 SSEL0 复位和 SSEL1 置位来选择 ACLK;
  - 用中断源控制位 ISCTL 选定时器进位信号。
- P0.1 的方向和中断触发沿应适当选定。
- 将预置数寄存器装入数据。
  - 对计数器的写操作将把预置数寄存器中的数据装入定时器。
  - RXACT 位复位。

```

; ----- 准备发送周期 -----
TXINIT    MOV.B    # BitTime1_2, &TCPLD ; 装入半位长,
    
```

```

; 如果有必要,则在起始位发送前禁止 P0.1 输
; 出缓存
MOV.B    #072h, &TCCTL    ; TXD = 1(初值),选 ACLK=1, TXE = 1
MOV.B    #0, &TCDAT      ; 用空写操作将数据写入 8 位定时器/计数器
MOV.B    #BitTime1, &TCPLD ; 将发送首位的位长装入预置数寄存器
BIS.B    #ENCNT, &TCCTL   ; 设置发送开始条件
BIS.B    #IEaP0_0, &AdP0_0 ; 允许 P0.1 中断, SFR 地址为 0
CLR.B    &TXStatus       ; 准备暂存寄存器
MOV.B    #ParVal, Parity  ; ParVal = 0, 偶校验; ParVal = 1, 奇校验
RET

; ----- 发送/接收周响应, UART 处理程序 -----
; 以下两条指令决定发送或接收数据。因为共用一个中断向量,这是必须的
VECRCTX  BIT.B    #RXACT, &TCCTL    ; 检查有效中断处理模式
          JNZ      RCINTRPT         ; 接收模式有效,跳转
;
TXINTRPT  PUSH    R5                ; RXACT = 0, 发送
          MOV.B   &TXStatus, R5     ; 用 TXStatus 作分支跳转
          BR      TXTAB(R5)        ;
TXTAB     .Word   TXStat0           ; 起始位      ; Bitime2, 13 个 ACLK 周期
          .Word   TXStat1           ; 位 0, LSB   ; Bitime1, 14 个 ACLK 周期
          .Word   TXStat1           ; 位 1       ; Bitime1, 14 个 ACLK 周期
          .Word   TXStat2           ; 位 2       ; Bitime2, 13 个 ACLK 周期
          .Word   TXStat1           ; 位 3       ; Bitime1, 14 个 ACLK 周期
          .Word   TXStat1           ; 位 4       ; Bitime1, 14 个 ACLK 周期
          .Word   TXStat2           ; 位 5       ; Bitime2, 13 个 ACLK 周期
          .Word   TXStat1           ; 位 6       ; Bitime1, 14 个 ACLK 周期
          .Word   TXStat1           ; 位 7       ; Bitime1, 14 个 ACLK 周期
          .Word   TXPar             ; 校验位     ; Bitime2, 13 个 ACLK 周期
          .Word   TXStop           ; 停止位 1   ; Bitime1, 14 个 ACLK 周期
          .Word   TXStop           ; 停止位 2   ; Bitime1, 14 个 ACLK 周期
          .Word   TXCCmpl          ; 帧发送结束
TXStat0   BIC.B   #TXD, &TCCTL     ;
          MOV.B   #BitTime2, &TCPLD ; 将 13/32 768 s 装入预置数寄存器
          JMP     TXRET
TXStat2   MOV.B   #BitTime2, &TCPLD ; 将 13/32 768 s 装入预置数寄存器
          JMP     L$3
TXStat1   MOV.B   #BitTime1, &TCPLD ; 将 14/32 768 s 装入预置数寄存器
L$3       RRA     TXData           ; LSB 移入进位位
          JNC     L$1
L$2       BIS.B   #TXD, &TCCTL     ; TCCTL 中的 TXD 位置位
          XOR.B   Parity           ; 计算“1”的个数作校验
          JMP     IXRET           ; 位输出完成
L$1       BIC.B   #TXD, &TCCTL     ; TCCTL 中的 TXD 复位

```



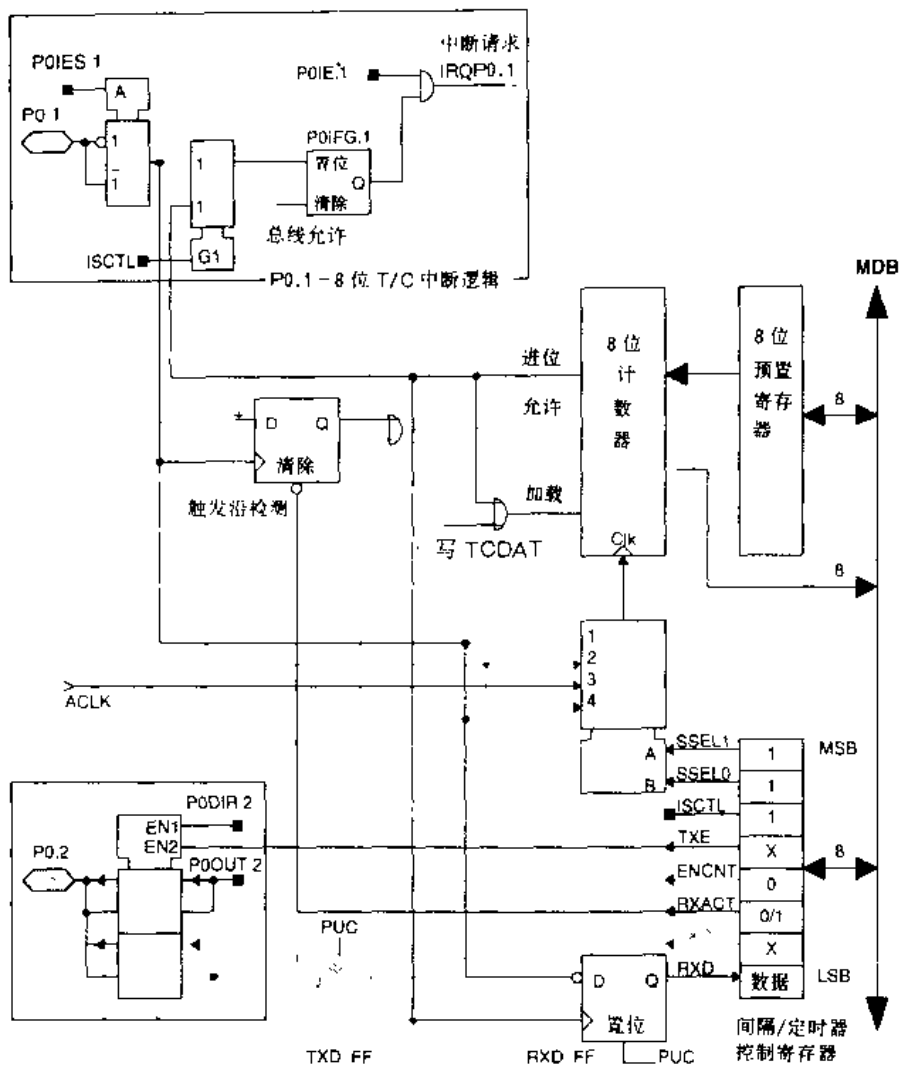


图 10.13 8 位定时器/计数器结构在波特率为 2 400、ACLK 时钟下接收应用实例

只要 RXACT 和 ENCNT 复位,定时器/计数器就停止。将 RXACT 置位以允许下降沿检测开始进入接收状态。P0.1 端口上的起始位的第一个边沿将边沿检测锁存器的输出置位。它将保持置位直到下一个 RXACT 复位。

一旦边沿检测锁存器置位,定时器就开始工作。第一个定时时间开始,并且经过程序设定的时间,P0.1 的逻辑电平锁存到 RXD 寄存器中。此后中断请求发生。

对第一位的中断处理是可选的,并可以测试起始位是否存在。当缺少起始位时接收被中止。当接收周期继续进行时,下一个定时需要在预置数寄存器中装入适当的值。

后续各位在中断程序中的处理都是相同的。

在中断服务程序中应处理:

- 存储 RXD 位信息;
- 准备下一位的时序;
- 可选的处理内容包括更新校验信息,当校验出错时决定程序流向,寻找地址位信息;
- 如果接收位应是停止位,则作停止位测试。

**注意: UART 协议, LSB/MSB 顺序**

异步通信中首先发送最低位。为了接收时顺序正确,可用 RRC 指令收集数据。

```

; ----- 接收中断处理程序 -----
RCINTRPTPUSH      R5                ; 接收中断程序
                                   ; R5 暂存接收位的指针

        MOV.B      &RCstatus, R5
        BR         RCTAB(R5)

;
RCTAB      .Word    RCstat0          ; 接收起始位, 置位 0 接收时间, 13ACLK
           .Word    RCstat1          ; 接收位 0, 置位 1 接收时间, 14ACLK
           .Word    RCstat1          ; 接收位 1, 置位 2 接收时间
           .Word    RCstat2          ; 接收位 2, 置位 3 接收时间
           .Word    RCstat1          ; 接收位 3, 置位 4 接收时间
           .Word    RCstat1          ; 接收位 4, 置位 5 接收时间
           .Word    RCstat2          ; 接收位 5, 置位 6 接收时间
           .Word    RCstat1          ; 接收位 6, 置位 7 接收时间/MSB
           .Word    RCstat1          ; 接收位 7, 置校验位接收时间
           .Word    RCstat2          ; 接收校验位, 置停止位 1 接收时间
           .Word    RCstop1         ; 接收停止位 1, 置停止位 2 接收时间
           .Word    RCstop2         ; 接收停止位 2, 置停止位 2 接收时间
           .Word    RCCmpl          ; 完成一帧接收

; ----- 接收起始位: 检测空号 -----
RCstat0  BIT.B      #RXD, &TCCTL    ; 检查起始位
        JC         RCErrror        ; 出错: 起始位是传号, 非空号
        MOV.B     #BitTime2, &TCPLD ; 起始位正确, 位长装入预置数寄存器
        JMP      RCRET

;
RCstat2  MOV.B     #BitTime2, &TCPLD ; 位定时 2 装入预置数寄存器
        JMP      RCBit

RCstat1  MOV.B     #BitTime1, &TCPLD ; 位定时 1 装入预置数寄存器
RCBit   BIT.B      #RXD, &TCCTL    ; RXD 位送进位位
        JNC      RCRET             ; RXD 位 = 进位位 = 0?, 是, 跳转
        RRC      RCDData           ; RXD 位送 MSB
        INC.B    &Parity           ; RXD 位 = 1, 累计“1”
        JMP      RCRET1

RCRET   RRC      RCDData           ; RXD 位送 MSB
RCRET1  INCD.B    &RCstatus
RCCmpl  POP      R5
        RETI

;
; 校验位与其他位的接收相同, 在接收停止位 1 时检查校验位

```



```

;
RCstop1  BIT.B      # 1, &Parity      ; 检查校验位, 应为 0
         JNZ        RCErrror         ; 校验错
;
RCstop2  MOV.B      # BitTime1, &TCPLD ; 位定时 1 装入预置数寄存器
         BIT.B      # RXD, &TCCTL     ; 检查停止位为传号
         JNZ        RCRET            ; 停止位为传号, 正确
;
; 出错处理: 尝试接收新起始位
RCErrror POP        R5
         CALL       RCINIT           ; 再次初始化接收程序
         RETI
    
```

### 10.3 看门狗定时器

看门狗定时器 WDT (Watchdog Timer) 的主要功能是当程序发生错误时执行一个受控的系统重新启动。图 10.14 为看门狗定时器原理图。如果超过了选定的定时时间, 则发生系统复位。如果应用中不需要此功能, 那么可以把它当做定时器。当选定定时时间到达后将产生中断。

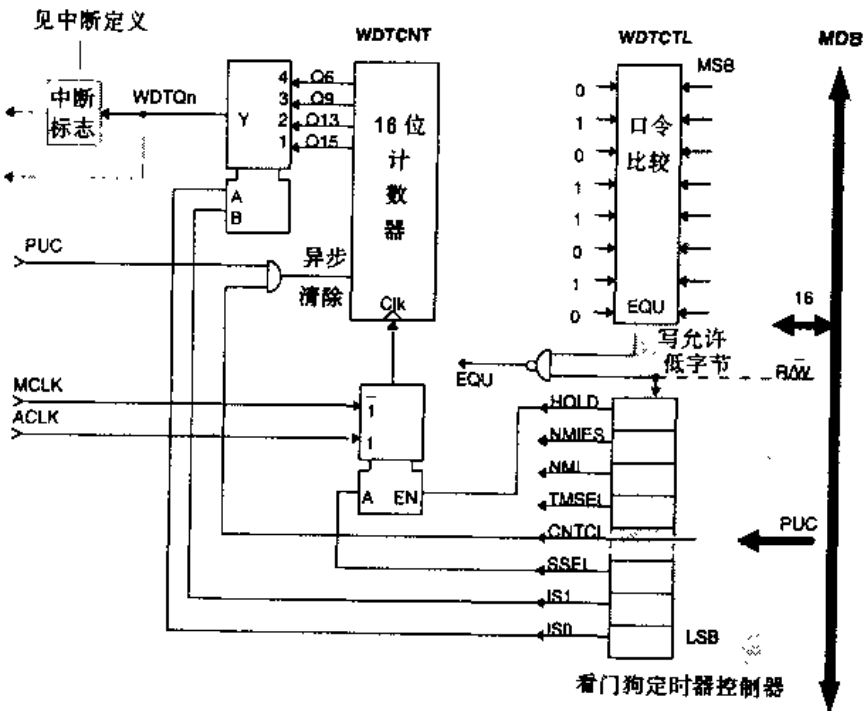


图 10.14 看门狗定时器原理图

看门狗定时器特性如下:

- 8 种软件可选的定时时间;
- 2 种工作模式, 即看门狗或定时器;

- 在看门狗模式下超过定时后,则产生系统复位,在定时模式下则产生中断请求;
- 出于安全原因,对 WDT 控制寄存器的写操作须用口令;
- 用停止模式支持超低功耗特性。

### 10.3.1 看门狗定时器寄存器

#### 1. 看门狗定时器的控制寄存器的结构

看门狗定时器的计数器 WDTCNT 是 16 位增计数器。对它不能直接用软件访问。通过 WDTCTL 对 WDTCNT 进行控制。它是位于字地址 0120h 的低字节的一个 16 位读/写寄存器。所有读写操作都要用字指令,即不带后缀或用后缀“.W”。在两种工作模式(看门狗或定时器)中,只有含有正确口令的指令才能写入 WDTCTL。WDTCTL 的位定义如下:

WDTCTL (0120h)

7							0
HOLD	NMIES	NMI	TMSEL	CNTCL	SSEL	ISI	ISO
rw 0	rw-0	rw-0	rw-0	r0(w)	rw-0	rw-0	rw-0

对看门狗定时器控制寄存器各位说明如下。

位 0、1: ISO、ISI 位用于选择 WDTCNT 中 4 个输出之一。

设晶振频率为 32 768 Hz,系统频率为 1 MHz,可以有以下定时时间:

SSEL	ISI	ISO	定时/ms	
0	1	1	0.064	$T_{MCLK} \times 2^6$
0	1	0	0.5	$T_{MCLK} \times 2^9$
1	1	1	1.9	$T_{ACLK} \times 2^6$
0	0	1	8	$T_{MCLK} \times 2^{13}$
1	1	0	16	$T_{ACLK} \times 2^9$
0	0	0	32	$T_{MCLK} \times 2^{15} \leftarrow \text{PUC 后的值}$
1	0	1	250	$T_{ACLK} \times 2^{13}$
1	0	0	1 000	$T_{ACLK} \times 2^{15}$

位 2: SSEL 位选择 WDTCNT 的时钟源。

SSEL = 0: WDTCNT 以系统频率为时钟。

SSEL = 1: WDTCNT 以 ACLK 为时钟,即晶振频率(32 768 Hz)。

位 3: CNTCL 位,在两种模式中,写入“1”使 WDTCNT 从 00000h 重启动,读出无定义。

位 4: TMSEL 位选择工作模式,即看门狗或定时器。

TMSEL = 0: 看门狗模式。

TMSEL = 1: 定时器模式。

位 5: NMI 位选择  $\overline{\text{RST}}$ /NMI 引脚功能。在 PUC 后它被复位。

NMI = 0:  $\overline{\text{RST}}$ /NMI 输入作为复位输入。

只要  $\overline{\text{RST}}$ /NMI 引脚保持“低”,内部 PUC 信号一直有效(电平触发)。



令错误将触发 PUC 信号。该信号自动将整个系统中相应的寄存器复位。而 WDT CNT 各位的复位将影响系统配置,使 WDT 置为看门狗模式,并使 RST/NMI 引脚被设为复位结构方式。

当 WDT 被设为定时器模式时,在设定的定时时间到达时 WDTIFG 标志置位,并且它请求常规的中断服务。WDT 是单源中断,当系统处理中断时中断标志自动复位。允许位保持不变。WDT 中断允许位和通用中断允许 GIE 位应置为允许中断状态。定时器模式的中断向量地址与看门狗模式的不同。

### 10.3.3 看门狗定时器操作

看门狗定时器模块可置成两种模式:看门狗模式和间隔定时器模式。

#### 1. 看门狗模式

上电或系统复位后,看门狗定时器自动进入看门狗模式。此时,在看门狗控制寄存器 WDTCTL 中各位和看门狗计数器 WDT CNT 中各位复位。WDTCTL 寄存器中的初始状态导致定时时间为 32 ms(在系统时钟频率为 1 MHz 时)。由于系统时钟发生器中的数字控制振荡器 DCO 置为以最低频率工作,软件有大约 32 600 个时钟周期来响应这一重要事件。初始状态选为 WDT CNT 在系统频率下工作,允许应用程序开始运行,并在看门狗时间段的中间使它复位。

用做看门狗模式时,软件必须周期性地在 WDTCTL 的 CNTCL 位中写“1”来使 WDT CNT 复位,以防止超过选定的定时时间。如果因程序发生问题,计数器不再被复位而超过了定时时间,那么就会产生复位和激活系统上电清除信号(PUC)。系统从上电复位的地址重新启动。可通过检测 SFR 中的 IFG1 的位 0 来判断复位原因。定时时间可以通过 SSEL、ISO 和 ISI 位来选定。

#### 2. 定时器模式

WDTCTL 的 TMSEL 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。定时时间可以通过软件对 WDTCTL 寄存器的 CNTCL 位置位来开始。

#### 注意:看门狗定时器,改变定时时间

改变定时时间而不同时清除 WDT CNT 将导致不可预料的系统立即复位或中断。定时时间改变应伴随计数器清除,并在一条指令中完成。例如“MOV #05A0Ah, &WDTCTL”。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。

在正常工作时,改变时钟源可能导致 WDT CNT 额外的计数时钟。

#### 3. 工作在低功耗模式

系统时钟发生器可运行在 5 种不同的模式下。其中有 3 种 MCLK 和 ACLK 是活动的,一种只有 ACLK 活动,还有一种 ACLK 和 MCLK 都不活动。

应用程序要求设置看门狗定时器来处理根据不同 ACLK 和 MCLK 条件下的硬件响应。

- CPUOFF 模式:程序运行停止。软件必须定义好在此模式期间的工作条件。
- MCLKOFF 模式/LMP2、3: ACLK 活动而 MCLK 停止。当选择 ACLK(32 768 Hz)时,看门狗定时器继续工作,并根据选定的工作模式通过系统复位或定时器中断(如果允许)来唤醒 CPU;当选择 MCLK 时,看门狗定时器将停止工作直到 MCLK 重新启动。





PWMCTL 寄存器中的控制标志 OS 定义 PWM 输出的极性。

当 OS=0 时, PWMDT 中的值表示 PWM 输出为“高”时的 PWM 计数器时钟脉冲数。

当 OS=1 时, 输出极性相反, PWMDT 中的值表示 PWM 输出为“低”时的 PWM 计数器时钟脉冲数。

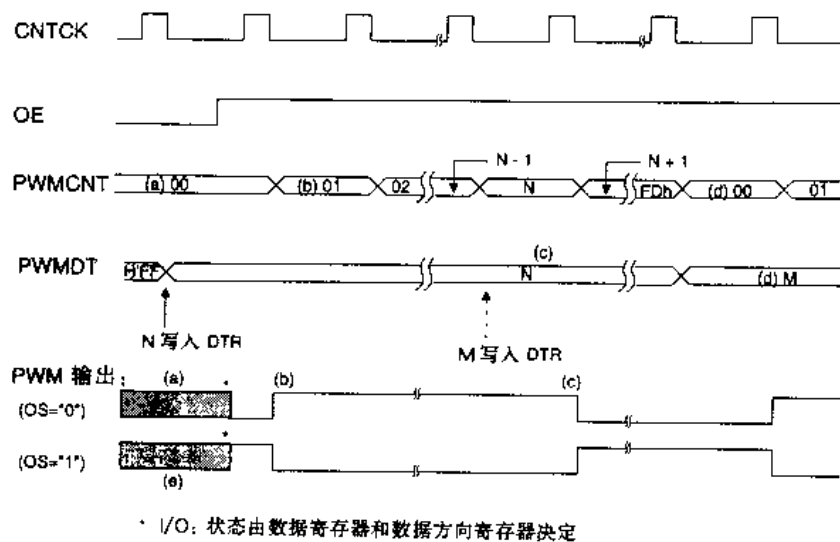


图 10.16 PWM 时序原理图

### 10.4.2 PWM 寄存器

通过 8 位 MDB 和 MAB 来控制 PWM 定时器模块。对它必须用字节指令访问。

寄存器	缩写	寄存器类型	地址	初始状态
PWM 定时器控制寄存器	PWMCTL.1	读写	58h	复位
PWM 占空比缓存	PWMDTB.1	读写	59h	复位
PWM 占空比寄存器	PWMDTR.1	读写	5Ah	复位
PWM 定时器计数器	PWMCNT.1	读(写)(见“注意”)	5Bh	复位
PWM 定时器控制寄存器	PWMCTL.2	读写	5Ch	复位
PWM 占空比缓存	PWMDTB.2	读写	5Dh	复位
PWM 占空比寄存器	PWMDTR.2	读写	5Eh	复位
PWM 定时器计数器	PWMCNT.2	读(写)(见“注意”)	5Fh	复位

#### 注意：改变定时器的计数

定时器的计数器是读写寄存器, 但是其写功能只用于测试目的。  
实际应用程序不应写这些寄存器

#### 1. 定时器的计数器 PWMCNT

计数器 PWMCNT 结构如下:

PWMCNT (05Bh 或 05Fh)

7							0
2 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

PWMCNT 是 8 位增计数器。当 PWMCTL 中的输出允许位 OE 置位时,计数器开始按时钟选择位 SSEL2 ~ SSEL0 选定的内部时钟源脉冲计数。从 000h 计数到 0FDh 后,计数又从 000h 开始。

PWM 定时器的计数器可读写;但是写功能只用于测试,应用程序不能对这些定时器计数器写入,因为这将产生不可预料的效果。

当发生 PUC 时,PWM 定时器计数器初始化为 000h,同时 OE 位复位。

## 2. 占空比缓存寄存器 PWMDTB

PWMDTB (059h 或 05Dh)

7							0
2 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

占空比缓存寄存器保存占空比因子。当计数器从 0FDh 计数到 000h 时,这一占空比因子写入占空比寄存器。

PWMDTB 在 OscOff 模式下和系统复位时置为 000h。

### 注意: 改变 PWM 占空比因子

只有在 PWMDTB 中写入新值才能改变占空比。任何直接对占空比寄存器的写操作将导致在运行周期中产生一个随机占空比。下一周期会用已进入占空比缓存的新占空比因子运行。

## 3. 占空比寄存器 PWMDT

PWMDT (05Ah 或 05Eh)

7							0
2 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>
rw-1	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1

占空比寄存器确定输出脉冲的占空比。可以用 1/254 的分辨率选择范围在 0% ~ 100% 的任何占空比。在 PWMDT 中写 000h 则给定 0% 的占空比,写 127(07Fh)则给定 50% 的占空比,写 254(0FEh)则给定 100% 的占空比。

定时器计数器连续地与占空比寄存器的内容作比较。如果 PWMDT 值不为 0,那么当计数器从 00h 增到 01h 时 PWM 输出置位。当计数器增到 PWMDT 的值时,PWM 输出又回到零。如果 PWMDT 值是 0(占空比 0%),则 PWM 输出恒定为 0。

PWMDT 是双缓存的。当定时器计数器在运行时对 PWMDTB 写入的新值,只有当计数器发生从 0FDh 到 000h 的变化后才变为有效。PUC 后,OE 位复位,定时器计数停止,这时新值写入后立即生效。当对 PWMDT 进行读操作时,读出的是当前的有效值。

占空比寄存器在发生系统复位和在 OscOff 模式下被初始化成 0FFh。

## 4. PWM 定时器控制寄存器 PWMCTL

PWMCTL 的位定义如下:



PWMCTL (058h 或 05Ch)

7							0	
-	-	SSEL2	SSEL1	SSEL0	CMPM	---	OS	OE
rw-0	rw-0	rw-0	rw-0	r	rw-0	rw-0	rw-0	rw-0

PWM 控制寄存器是 8 位可读写寄存器。它选择时钟源并控制 PWM 的输出。

对 PWM 控制寄存器各位说明如下：

位 0： 输出允许(OE)，允许 PWM 计数器和 PWM 输出。

OE = 0： PWM 输出禁止。PWMTIC 清除为 00h 并停止。

OE = 1： PWM 输出允许。PWMTIC 运行。

位 1： 输出选择(OS)，选择正或负逻辑信号作为 PWM 输出。

OS = 0： 正逻辑，正向脉冲，1 = 高(初始值)。

OS = 1： 负逻辑，负向脉冲，1 = 低。

位 2、7： 保留。不能改变并且读出为“0”。

位 3： CMPM，只读，可检测比较信号的输出。当定时器计数器 PWMCNT 和占空比寄存器 PWMDT 相同时，它读出为“1”。

位 4~6： 时钟选择。可从 8 个由 MCLK 和 ACLK 分频得来的时钟源选择之一。

SSEL2	SSEL1	SSEL0	时钟源
0	0	0	MCLK
0	0	1	MCLK/4
0	1	0	MCLK/16
0	1	1	ACLK
1	0	0	ACLK/4
1	0	1	ACLK/8
1	1	0	ACLK/16
1	1	1	ACLK/128

根据时钟源频率，可以计算出分辨率、周期和 PWM 的输出频率，即

$$\text{分辨率} = 1/\text{时钟源频率}$$

$$\text{PWM 周期} = \text{分辨率} \times 254 = 254/\text{时钟源频率}$$

$$\text{PWM 频率} = 1/\text{PWM 周期} = \text{时钟源频率}/254$$

$$\text{占空比} = \text{PWMDT}/254$$



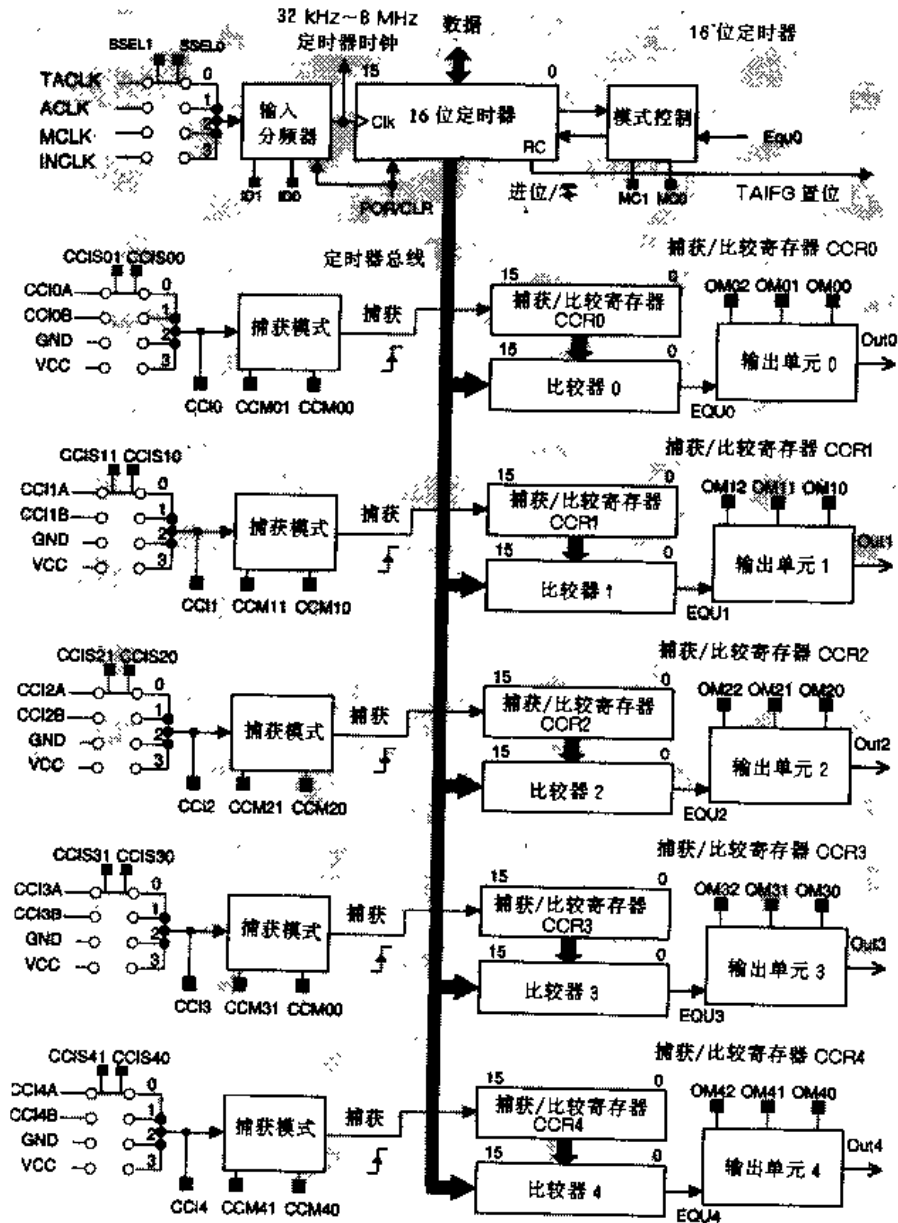


图 11.1 Timer\_A 原理图

数到 X, 此后捕获/比较寄存器也装入这一数据 X, 这时比较的结果是不匹配的。  
有如下 4 种定时器工作模式:

模式控制	模式	含义
MC1 MC0		
0 0	停止	定时器暂停
0 1	增计数	定时器增计数至等于比较寄存器 0 的值
1 0	连续	定时器连续增计数
1 1	增/减计数	定时器增计数到等于比较寄存器 0 的值, 然后又减计数到 0

1) 停止模式



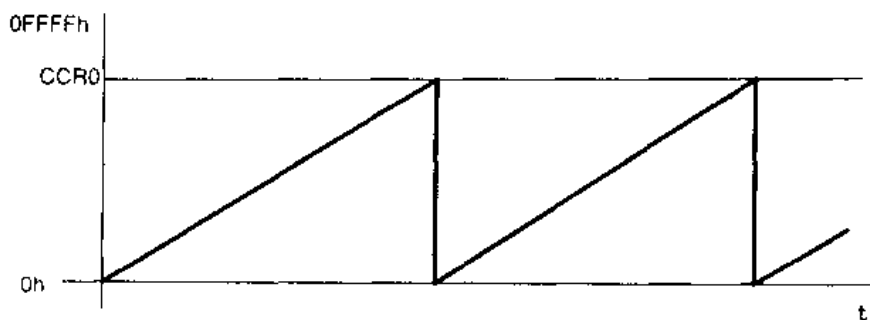


图 11.5 增计数模式下定时器的输出

TAIFG 标志置位。所有中断标志的置位与各自的中断允许位无关。

如果通用中断允许位和相应的中断允许位置位,则产生中断请求。

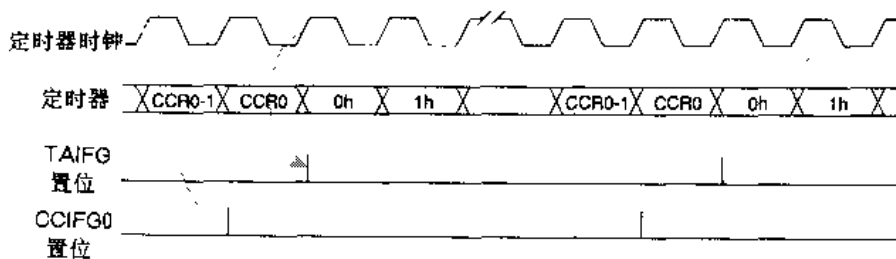


图 11.6 增计数模式下定时器及有关标志时序图

### 3) 连续计数模式

定时器从寄存器的当前值开始计数。如图 11.7 所示,当计到 0FFFFh 后,又从 00000h 开始重新计数。

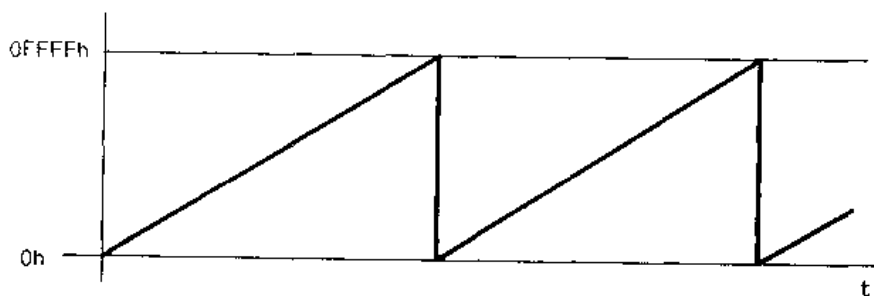


图 11.7 连续计数模式下定时器的输出

如果需要多个定时信号,则可用连续计数模式。中断处理程序在相应的比较寄存器 CCR<sub>x</sub> 上加上一个时间差。这个时间差是当前时刻(即相应的 CCR<sub>x</sub> 中的数值)到下一次中断发生时刻所经历时间。

图 11.8 为连续计数模式下定时器及有关标志的时序图。

当定时器从 0FFFFh 计数到 0 时,TAIFG 置位。中断标志的置位与相应的中断允许位无关。如果通用中断允许位(GIE)和各自相应的中断允许位置位,则产生中断请求。

捕获/比较寄存器 CCR0 工作方式与连续模式中的其他寄存器一样。

### 4) 增减计数模式

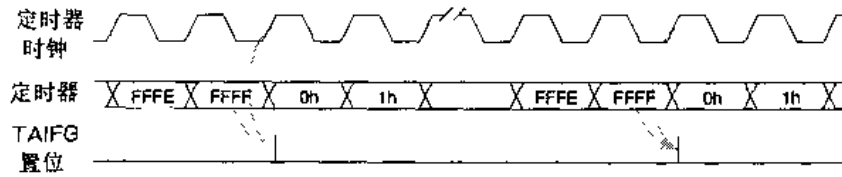


图 11.8 连续计数模式下定时器及有关标志的时序图

定时器增计数到与 CCR0 的值相等,如图 11.9 所示,然后反向作减计数到 0。

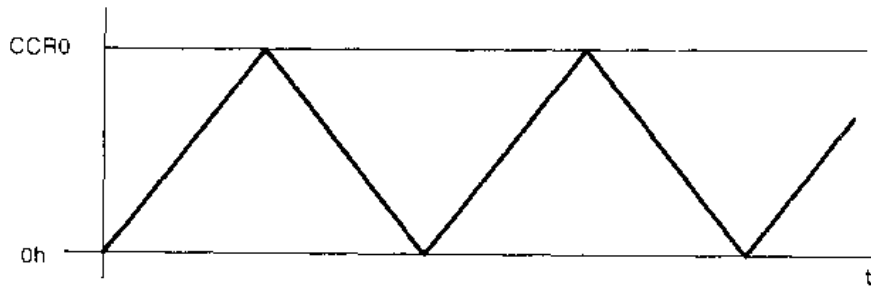


图 11.9 增减计数模式下定时器输出

计数方向由触发器锁存。触发器在 0000h 时置位以使定时器增计数,并且当计数到等于 CCR0 的值时复位,以使定时器锁存减计数模式。

计数周期由比较寄存器 CCR0 定义。它是 CCR0 寄存器数值的 2 倍。

图 11.10 为增减计数模式下定时器及有关标志的时序图。

当定时器从“CCR0 - 1”增计数到 CCR0 时,中断标志 CCIFG0 置位。

当定时器从 0001h 减计数到 0000h 时,中断标志 TAIFG 置位。

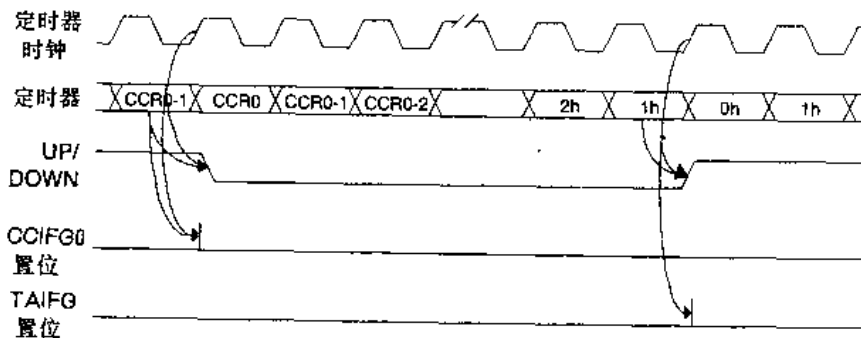


图 11.10 增减计数模式下定时器及有关标志的时序图

### 3. 捕获/比较模块

图 11.11 为捕获比较模块原理图。5 个相同的模块为实时处理提供了灵活的控制手段。任一模块寄存器都可用于捕获应用事件的发生时间,或产生定时间隔。如果相应的中断允许,那么每完成一个时间捕获或一次定时间隔,捕获/比较模块都将产生中断。由 CCTLx 中的模式位 CAPx 选择比较(CAPx 复位)或捕获(CAPx 置位)工作模式。在捕获模式下,控制字 CCTLx 中的 CCMx1 和 CCMx0 定义产生捕获功能的条件,即不捕获、上升沿捕获、下降沿捕获或两个沿都发生捕获。

中断允许位 CCIEx 和中断标志位 CCIFGx 用于捕获和比较模式中。当发生捕获或比较

事件时, CCIFG 置位。控制位 CAPx 定义选择采用捕获或比较模式。

捕获输入 CCIxA 和 CCIxB 连接外部引脚或内部信号。不同的 MSP430 型号连到 CCIxA 和 CCIxB 的信号也不相同。

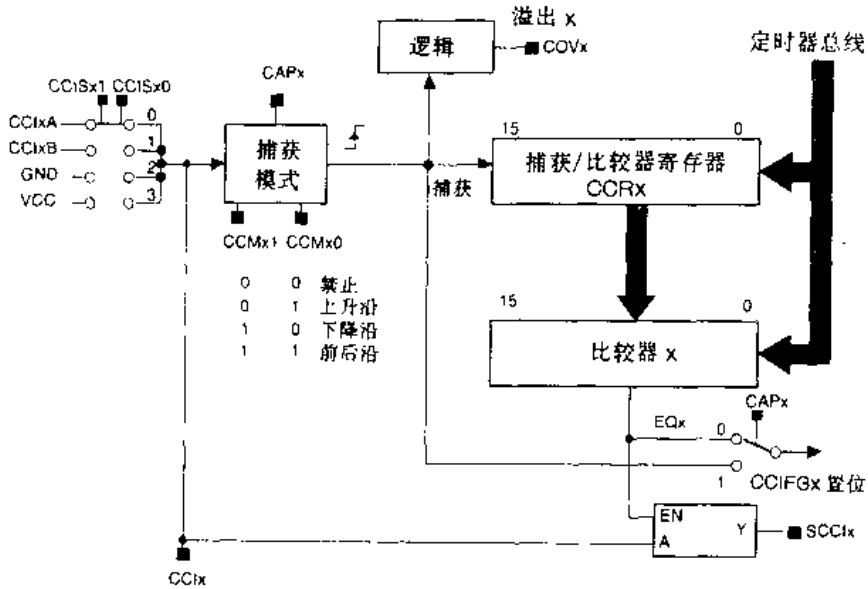


图 11.11 捕获/比较模块

由控制位 CCISx1 和 CCISx0 选择捕获逻辑的输入信号源。可以经过 CCIx 位或者通过与比较信号 EQUx 同步,用软件直接读取它。同步位 SCCIx 支持串行通信协议的软件处理。捕获信号与定时器时钟可以异步。采用同步或异步捕获信号可支持不同的应用状况。

图 11.12 为捕获/比较模块与定时器时钟同步原理图。图 11.13 为捕获信号与定时器时钟信号同步时序图。

捕获信号与定时器时钟同步,它将捕获/比较中断标志置位,并将定时器数值存入捕获寄存器。它们的同步特性可避免定时器数据和捕获信号的时间竞争。用捕获/比较控制寄存器 CCTLx 中的同步捕获信号位 SCSx 选择捕获信号模式。

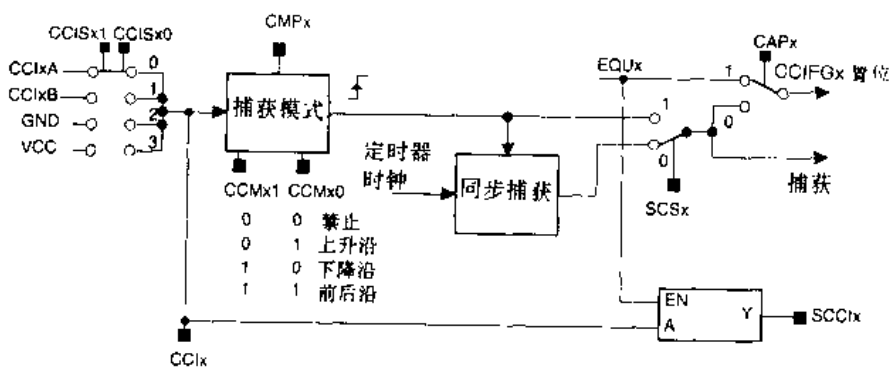


图 11.12 捕获/比较模块与定时器时钟同步原理图

非同步捕获信号支持低速定时器时钟的应用。捕获事件与定时器时钟可能产生时间竞争,因而导致捕获数据无效。软件应验证数据并加以校正。

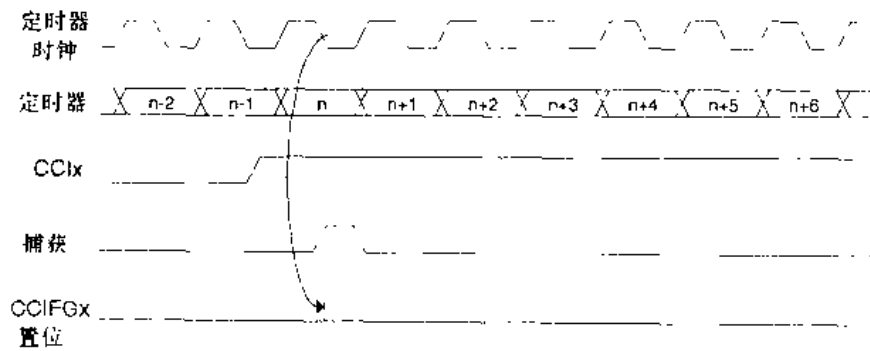


图 11.13 捕获信号与定时器时钟信号同步时序图

；软件实例：异步捕捉信号处理

；

；捕捉/比较寄存器 CCRx 的数据由相应的中断程序软件产生，因此，只会在 CCRIFG 置位后产

；生。定时器的时钟远慢于系统时钟 MCLK

；

```

CCRx_Int_hand ... ; 中断处理开始
...
...
CMP &CCRx, &TAR ; 检查 CCRX = TAR
JEQ Data_Valid
MOV &TAR, &CCRx ; CCRx 数据有错，改为定
; 时器数据
Data_Valid ... ; CCRx 数据正确
...
...
RETI

```

### 11.1.2 捕获模式

位于控制字 CCTLx 中的模式位 CAPx 的置位将选定捕获模式。捕获模式用于时间事件的精确定位。它可以用于速度计算或时间测量中。如果在选定的输入引脚上发生选定脉冲触发沿(上升沿、下降沿,或两者皆可),则定时器计数的值将被复制到捕获寄存器 CCRx 中。可选择 3 个不同的输入源: CCIxA、CCIxB, 或者通过捕获/比较控制寄存器 CCTLx 中的 CCISx1/CCISx0 位选定的 CPU/软件信号。

完成捕获后,则

- 控制字 CCTLx 中的中断标志 CCIFGx 置位;
- 如果通用中断允许位 GIE 和相应的中断允许位 CCIEEx 置位,则产生中断请求。

须用字指令来对捕获/比较寄存器 CCRx 进行操作。将计数器的最新值复制并保存在其中。它提供溢出逻辑。复位表示在下次捕获完成前捕获数据已被读取。如果捕获数据还未读取时第二次捕获数据已锁存,则寄存器 CCTLx 中的溢出位 COVx 置位。检查这一位可以使程序从失去同步状态中恢复。



只有在另一次捕获发生前捕获数据已完成读取,捕获才复位。如果没有完成读取操作,则溢出标志位将置位。图 11.14 所示为捕获模式状态图。

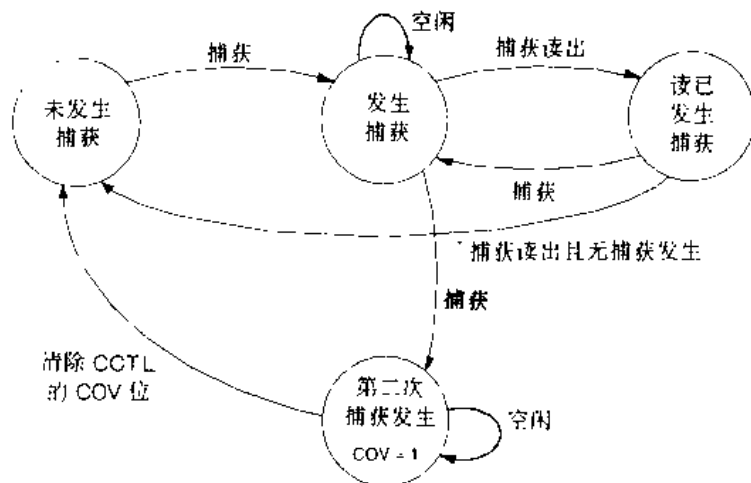


图 11.14 捕获模式状态图

需要用软件来复位溢出位 COVX。

；软件实例：溢出条件捕捉数据的处理

；

；捕捉/比较寄存器 CCRx 数据在检测到溢出后的下一条指令产生

；程序中判断并决定是正常处理还是出错处理

；

CGRx\_Int\_hand ... ; 中断处理开始

...

...

MOV &CCRx, RAM\_Buffer

BIT #COV, &CCTLx

JNZ Overflow\_Hand

...

...

...

RETI

Overflow\_Hand BIC #COV, &CCTLx ; 复位捕捉溢出标志, 恢复同步

...

...

RETI

**注意：捕获与定时器暂停**

当定时器暂停时捕获应该停止。顺序应先是先停止捕获功能,再停止定时器计数。捕获功能重新开始时,顺序是先开始捕获功能,再开始定时器计数。

### 11.1.3 比较模式

如果控制字 CCTL<sub>x</sub> 中的 CAP<sub>x</sub> 复位, 则选择比较模式。这时所有捕获硬件停止工作。如果定时器计数值等于比较寄存器 x 中的值, 那么

- 位于控制字 CTL<sub>x</sub> 中的中断标志 CCIFG<sub>x</sub> 置位。
- 如果 GIE 和 CCIE<sub>x</sub> 置位, 则产生中断请求。
- EQU<sub>x</sub> 信号输出到输出单元 OU<sub>x</sub> 中。根据选定的输出模式, 信号可以是置位、复位, 或将输出 OUT<sub>x</sub> 翻转(如果 OUTMOD<sub>x</sub>>0)。

必须用字指令对捕获/比较寄存器 CCR<sub>x</sub> 进行访问。它保持比较值。捕获模式的溢出逻辑在此模式下无效。

当定时器值大于或等于 CCR0 值时, EQU0 信号为真; 当定时器值等于相应的 CCR1 ~ CCR4 的值时, EQU1 ~ EQU4 信号为真。

### 11.1.4 输出单元

输出单元(见图 11.15)支持 PWM 或 DAC 的应用。按 3 个控制位选择的功能, 由捕获/比较寄存器的输出 EQU0 和 EQU<sub>x</sub> 控制输出逻辑。由对应于捕获/比较模块的 5 个输出单元 OU0 ~ OU4 执行。控制位 OM<sub>x</sub>0、OM<sub>x</sub>1 和 OM<sub>x</sub>2 位于控制寄存器 CCTL<sub>x</sub> 中。

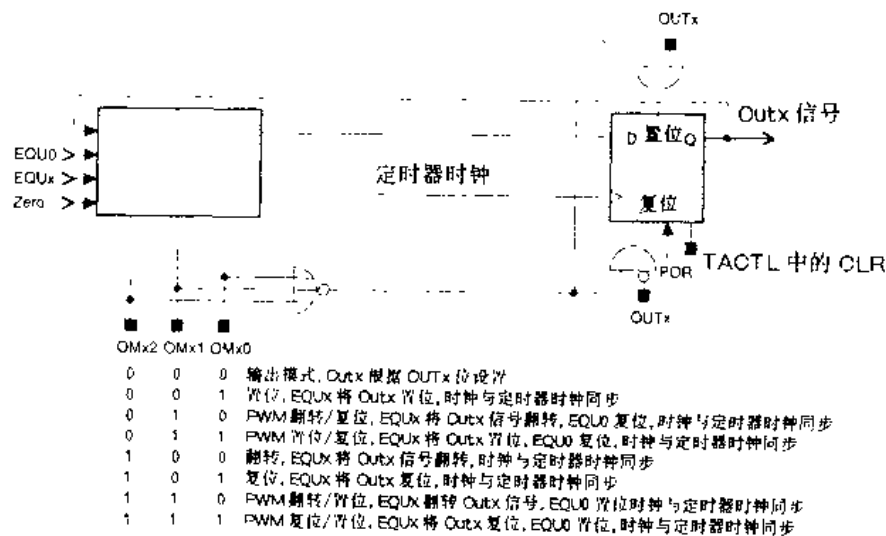


图 11.15 输出单元

如果 OM<sub>x</sub>0、OM<sub>x</sub>1 和 OM<sub>x</sub>2 选定了输出模式 0, 则由控制位 OUT<sub>x</sub> 决定 Outx 信号。输出从现有电平开始, 而与选定的模式无关。

#### 1. 增计数模式

当定时器增计数到 CCR<sub>x</sub> 以及由 CCR0 计数到 0 时, Outx 信号根据选定的输出模式发生变化。图 11.16 为在输出模式 3 下增计数模式实例。

#### 2. 连续计数模式

当定时器计数到 CCR<sub>x</sub> 和计数到 CCR0 时, Outx 信号根据选定的输出模式发生变化。图 11.17 为在输出模式 3 下连续模式实例。

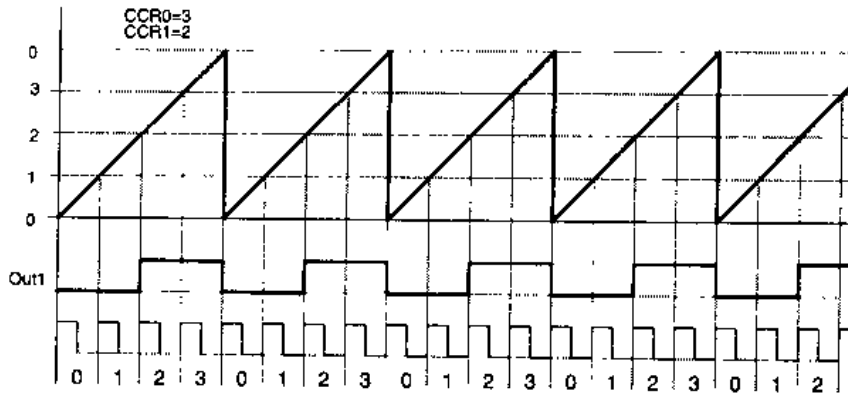


图 11.16 输出单元——在输出模式 3 下增计数模式实例

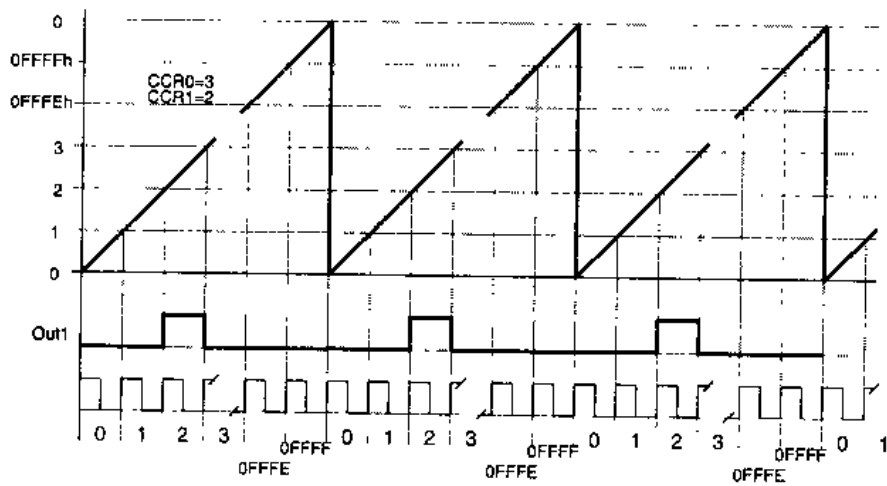


图 11.17 输出单元——在输出模式 3 下连续模式实例

### 3. 增/减计数模式

当定时器增计数到 CCR<sub>x</sub> 和减计数到 CCR<sub>x</sub> 时, Out<sub>x</sub> 信号根据选定的输出模式发生改变。图 11.18 为在输出模式 3 下增/减计数模式实例。

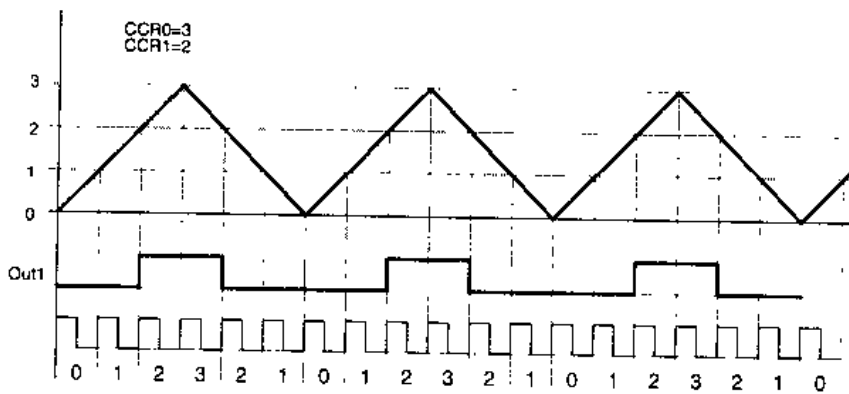


图 11.18 输出单元——在输出模式 3 下增/减计数模式实例

## 11.2 Timer \_ A 的寄存器

16 位定时器模块的硬件是字结构的, 必须用字指令来访问。

寄存器	缩写	寄存器类型	地址	初始状态
Timer _ A 控制寄存器	TACTL	读写	160h	POR 复位
Timer _ A 寄存器	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	CCR0	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	166h	POR 复位
捕获/比较寄存器 2	CCR2	读写	176h	POR 复位
捕获/比较控制寄存器 3	CCTL3	读写	168h	POR 复位
捕获/比较寄存器 3	CCR3	读写	178h	POR 复位
捕获/比较控制寄存器 4	CCTL4	读写	16Ah	POR 复位
捕获/比较寄存器 4	CCR4	读写	17Ah	POR 复位
中断向量寄存器	TAIV	只读	12Eh	(POR 复位)

地址 16Ch、16Eh、17Ch 和 17Eh 保留作将来扩展用。

### 11.2.1 Timer \_ A 控制寄存器 TACTL

全部关于定时器及其操作的控制位都位于定时器控制寄存器 TACTL 中。在 POR 信号后, 所有位都自动复位。但 PUC 不会影响它们。控制寄存器必须用字指令访问。TACTL 的位定义如下:

TACTL (160h)

15					0									
未用					SSEL2-0		ID1-0		MCI-0		未用	CLR	TAIE	TAIFG
rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	(w)-	rw-	rw-
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)		(0)

对 TACTL 各位说明如下:

位 0: TAIFG, 指示定时器溢出事件。

增计数模式: 如果定时器由 CCR0 计数到 0h, 则 TAIFG 置位。

连续模式: 如果定时器由 0FFFFh 计数到 0h, 则 TAIFG 置位。

增/减计数模式: 如果定时器减计数到 0h, 则 TAIFG 置位。

位 1: 定时器溢出中断允许位 TAIE。如果置位, 则溢出将产生中断请求, 不产生复位。

位 2: 定时器清除位 CLR。定时器和输入分频器在 POR 后或 CLR 置位时复位。CLR 由硬件自动复位, 其读出始终为“0”。定时器在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。

位 3: 未用。

位 4~5: 模式控制位。

MCI	MCO	控制模式	说明
0	0	停止	定时器暂定
0	1	增计数到 CCR0	计数至 CCR0 并从 0h 重新开始
1	0	连续, 增计数	增计数, 总共 65 536 步
1	1	增、减计数	增计数至 CCR0, 减计数至 0h, ……

位 6~7: 输入分频控制位。

ID1	ID0	分频	说明
0	0	PASS	输入信号直通定时器
0	1	/2	输入信号 2 分频
1	0	/4	输入信号 4 分频
1	1	/8	输入信号 8 分频

位 8~10: 定时器进入输入分频器的输入时钟源选择。

SSEL2	SSEL1	SSEL0	输入信号	说明
0	0	0	TACLK	用特定的外部引脚信号
0	0	1	ACLK	用辅助时钟 ACLK
0	1	0	MCLK	用系统时钟 MCLK
0	1	1	INCLK	见器件说明
1	x	x	---	保留

位 11~15: 未用。

#### 注意: 修改 Timer\_A

如果定时器工作在 ACLK 或外部时钟 TACLK, 则任何对定时器寄存器 TAR 的写入将导致不可预料的结果。这时 CPU 的 MCLK 和定时器时钟是异步的, 可能引起时间竞争。

#### 注意: Timer\_A 控制位的改变

如果用 TACTL 控制寄存器中的控制位来改变定时器工作, 则在修改时定时器应停止, 尤其是修改输入选择位、输入分频位和定时器清除位时。输入时钟和软件所用的系统时钟异步可能引起时间竞争, 使定时器响应出错。

推荐的指令顺序如下:

1. 修改控制寄存器和停止定时器。
2. 启动定时器工作。

例: MOV #0286h, &TACTL ; ACLK/8, 定时器停止, 定时器复位  
BIS #10h, &TACTL ; 开始连续增计数

## 11.2.2 捕获/比较控制寄存器 CCTL

每个捕获/比较模块都有它自己的控制字 CCTL<sub>x</sub>。

CCTLx (162h~16Eh)

15

0

捕获模式	输入选择	SCS	SCCI	未用	CAP	OUTMODx	CCIE	CCI	OUT	COV	CCIFG
rw-	rw-rw-rw-rw-	rw-	rw-	rw-	rw-	rw-rw-rw-rw-	r	rw-	rw-	rw-	
(0)	(0)(0)(0)	(0)	(0)	(0)	(0)	(0)(0)(0)	(0)	(0)	(0)	(0)	

POR 使 CCTLx 的所有位复位, 而 PUC 不影响这些位。

对 CCTLx 各位说明如下:

位 0: CCI<sub>FGx</sub>, 捕获/比较中断标志。

捕获模式: 置位表示在寄存器 CCR<sub>x</sub> 中捕获了定时数值。

比较模式: 置位表示定时器计数值等于比较寄存器 CCR<sub>x</sub> 之值。

CCIFG<sub>0</sub> 标志: 当中断请求被接受时, CCIFG<sub>0</sub> 自动复位。

CCIFG<sub>1~4</sub> 标志: 读中断向量字后, 确定中断向量的标志自动复位。如果中断允许位复位, 则不会生成中断向量字, 但标志仍能置位。标志 CCIFG<sub>1~4</sub> 需要由软件来复位。

位 1: 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。不会有使 COV 置位的捕获事件。

当 CAP=1 时, 选择捕获模式。如果在捕获寄存器的值被读出前再次发生捕获事件, 则溢出标志 COV 置位。程序可检测 COV 来判断原值读出前是否又发生了捕获事件。读捕获寄存器时不会使溢出标志复位。

位 2: OUT, 如果 OUTMOD<sub>x</sub> 为 0, 则 OUT<sub>x</sub> 位对应于输出状态。

位 3: CCI<sub>x</sub>, 捕获/比较输入信号。

捕获模式: 选定的输入信号(CCI<sub>xA</sub>、CCI<sub>xB</sub>、VCC 或 GND)可读出。

比较模式: CCI 复位。

位 4: CCIE<sub>x</sub>, 中断允许位, 允许或禁止捕获/比较模块 x 的中断请求信号。当中断允许位、CCIFG<sub>x</sub> 和 GIE 置位时, 产生中断请求。

0: 中断禁止。

1: 中断允许。

位 5~7:

选择输出模式	说明
0 仅输出	OUT <sub>x</sub> 位的数据即 Out <sub>x</sub> 信号
1 置位	比较信号 EQU <sub>x</sub> 使 Out <sub>x</sub> 信号置位
2 PWM 翻转/复位	比较信号 EQU <sub>x</sub> 使 Out <sub>x</sub> 信号翻转, EQU <sub>0</sub> 使 Out <sub>x</sub> 信号复位
3 PWM 置位/复位	比较信号 EQU <sub>x</sub> 使 Out <sub>x</sub> 信号置位, EQU <sub>0</sub> 使 Out <sub>x</sub> 信号复位
4 翻转	比较信号 EQU <sub>x</sub> 使 Out <sub>x</sub> 信号翻转
5 复位	比较信号 EQU <sub>x</sub> 使 Out <sub>x</sub> 信号复位
6 PWM 翻转/置位	比较信号 EQU <sub>x</sub> 使 Out <sub>x</sub> 信号翻转, EQU <sub>0</sub> 使 Out <sub>x</sub> 信号置位
7 PWM 复位/置位	比较信号 EQU <sub>x</sub> 使 Out <sub>x</sub> 信号复位, EQU <sub>0</sub> 使 Out <sub>x</sub> 信号置位

位 8: CAP, 定义捕获/比较模块和相应的中断模块工作在捕获模式或比较模式。

- 0: 比较模式。  
1: 捕获模式。
- 位 9: 只读, 始终为零。
- 位 10: SCCI<sub>x</sub>, 捕获/比较输入信号, 与比较输出 EQU<sub>x</sub> 同步。选定的输入信号 (CCI<sub>x</sub>A、CCI<sub>x</sub>B、VCC 或 GND) 用比较器相等信号 EQU<sub>x</sub> 锁存在透明锁存器中, 且可读。
- 位 11: SCS, 捕获/比较信号可以用于与定时时钟异步或同步模式。  
异步模式 (SCS 复位) 允许在请求时立即将 CCIFG 置位和捕获定时器值。如果捕获源的周期远大于定时器时钟周期, 则这一模式很有用。如果定时器时钟和捕获源发生时间竞争, 则捕获寄存器的值可能会出错。  
同步模式 (SCS 置位) 是常用的, 且捕获总是有效的。  
0: 异步捕获。  
1: 同步捕获。
- 位 12~13: CCIS0 和 CCIS1, 输入选择。  
在捕获模式下, 此两位定义提供捕获事件的输入源; 在比较模式下这两位无用。  
0: 选择 CCI<sub>x</sub>A。  
1: 选择 CCI<sub>x</sub>B。  
2: GND。  
3: VCC。

位 14 - 15:

选择捕获模式	说明
0 禁止	禁止捕获
1 上升沿	上升沿捕获
2 下降沿	下降沿捕获
3 上升和下降沿	在上升沿与下降沿都捕获

#### 注意: 同时捕获和捕获模式选择

如果通过捕获/比较寄存器 CCR<sub>x</sub> 中的 CAP 位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/比较寄存器中的值是不可预料的。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。

例: `BIS #CAP, &CCTL2` ; 用 CCR2 选择捕获模式

`XOR #CCIS1, &CCTL2` ; 软件捕获; CCIS0 = 0, 捕获模式 = 3

### 11.2.3 Timer\_A 中断向量寄存器

16 位 Timer\_A 有 2 个中断向量:

- 在 Timer\_A 中断中, 捕获/比较寄存器 CCR0 中断向量具有最高优先级。捕获/比较寄存器 CCR0 能用于定义在增计数和增/减计数模式中的周期; 因此, 它需要最快速的服务。
- 其他捕获/比较寄存器共用一个复合向量。16 位中断向量字 TAIV 指示当前最高级

中断。

### 1. CCR0 中断向量

如果定时器计数值等于比较寄存器的值,则与 CCR0 相关的中断标志置位。捕获/比较中断标志如图 11.19 所示。

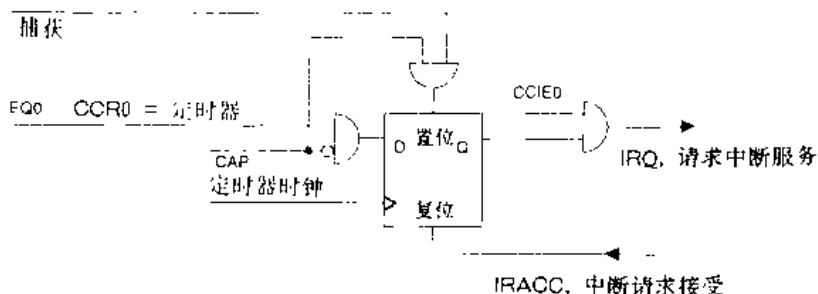


图 11.19 捕获/比较中断标志

捕获/比较寄存器 0 具有最高优先级,可用它的中断向量来加速实时处理。

### 2. 向量字 TAIFG 以及 CCIFG1 ~ CCIFG4 标志

图 11.20 为捕获/比较寄存器中断向量原理图。

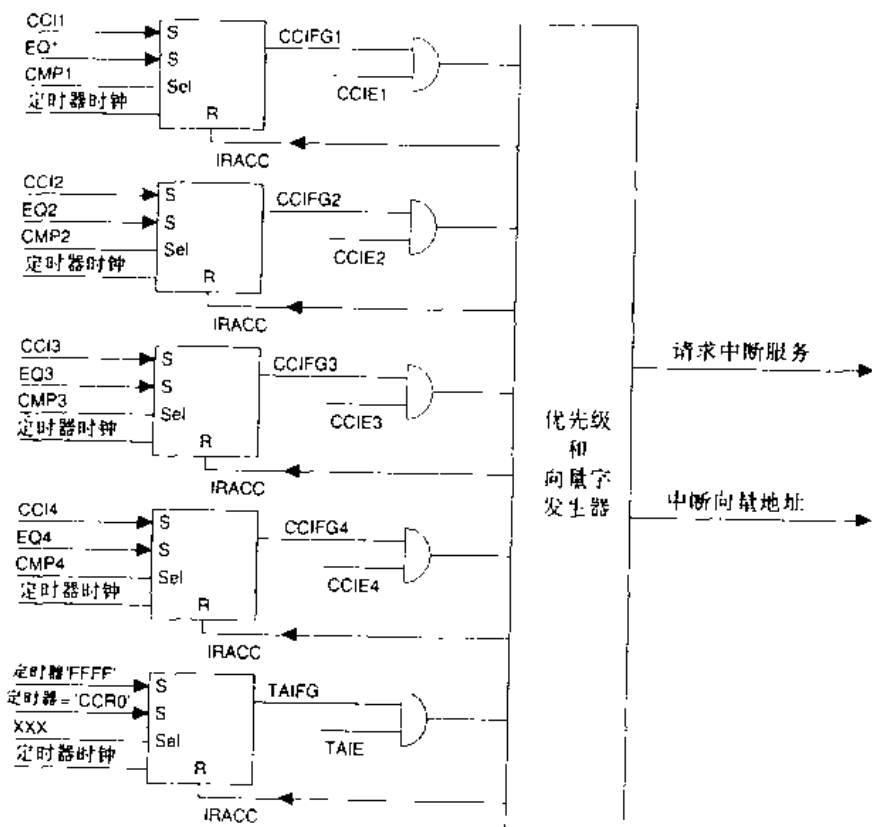


图 11.20 捕获/比较寄存器中断向量原理图

一个中断向量与 TAIFG 标志及其他 4 个捕获/比较寄存器 CCR1~CCR4 相关,结合成一个中断优先级结构,即具有最高优先级的 CCIFG<sub>x</sub> 标志产生从 0(无标志置位)到 12 的优先级码。这一编码可以加到 PC 上来得到相应的中断程序入口。向量字 TAIV 以 16 位字与 PC 相





```

; 中断标志 CCIFGx 和 TAIFG 由硬件复位。只对最高优先级响应中断标志复位
TIM_HND      $                               ; 中断响应                               6
              ADD      &TAIV, PC              ; 跳转表加偏移量                               3
              RETI                               ; 向量 0: 无中断                               5
              JMP      TIMMOD1                 ; 向量 2: 模块 1                               2
              JMP      TIMMOD2                 ; 向量 4: 模块 2                               2
              JMP      TIMMOD3                 ; 向量 6: 模块 3                               2
              JMP      TIMMOD4                 ; 向量 8: 模块 4                               2
;
; 模块 5 定时器溢出处理: 定时器寄存器扩展到 RAM 的 TIMEXT (MSBs) 中
;
TIMOVH                               ; 向量 12: TIMOV 标志
              INC      TIMEXT                   ; 定时器溢出处理                               4
              RETI                               5
;
TIMMOD2                               ; 向量 4: 模块 2
              ADD      #NN, &CCR2              ; 加时间差                               5
              ...                               ; 处理由此开始
              RETI                               ; 返回主程序                               5
;
;
TIMMOD1                               ; 向量 2: 模块 1
              ADD      #MM, &CCR1              ; 加时间差                               5
              ...                               ; 处理由此开始
              RETI                               ; 返回主程序                               5
; 模块 3 处理程序中提供检查有其他挂起中断的方法:
; 需要增加 5 个周期, 但是, 如有中断挂起, 可节省 9 个周期
;
TIMMOD3                               ; 向量 6: 模块 3
              ADD      #PP, &CCR3              ; 加时间差                               5
              ...                               ; 处理由此开始
              JMP      TIM_HND                 ; 检查挂起中断                               2

.SECT  "VECTORS", 0FFF0h              ; 中断向量
.WORD  TIM_HND                          ; 捕捉/比较模块 1~4 和定时器溢出
; TAIFG 的向量
.WORD  TIMMOD0                          ; 捕捉/比较模块 0

```

如果 FLL 关闭, 则需要增加 2 个额外时钟周期, 来使 CPU 系统的启动与系统的时钟 MCLK 同步。

对不同的中断源, 程序的开销包括中断等待和从中断返回时钟周期数(非处理任务本身)。

- 捕获/比较模块 CCR0            11 个时钟周期。
- 捕获/比较模块 CCR1 ~ CCR4    16 个时钟周期。

- 定时器溢出 TAIFG 14 个时钟周期。

#### 4. 定时极限

用 TAIV 寄存器和上述软件, 利用一个比较寄存器时两个事件之间的最短重复时间是:

$$t_{CRmin} = t_{taskmax} + 16 \times T_{cycle}$$

式中:  $t_{taskmax}$ ——在中断程序(例如计数器加 1)中完成任务的最长时间(最差情况);

$T_{cycle}$ ——MCLK 频率下时钟周期。

利用一个捕获寄存器时两个事件之间的最短重复时间是:

$$t_{CLmin} = t_{taskmax} + 16 \times T_{cycle}$$

### 11.3 Timer\_A 应用

#### 11.3.1 Timer\_A 增计数模式应用

如果定时周期不是 65 536 个时钟周期, 则可用增计数模式。连续模式的定时周期是 65 536 个时钟周期。CCR0 用于定义定时周期。

##### 1. 输出单元 OU0 的性能

输出单元 OU0 工作在 4 种模式下, 即输出模式 0、输出模式 1、输出模式 4 和输出模式 5。因为同时利用了 EQU0 信号, 所以其他 4 种模式不可用。

##### 2. 输出单元 OU1 到 OU4 的性能

输出单元 OU1~OU4 以及它们的驱动电路完全相同, 它们具有相同的特性。每个都可以工作在相同或不同的模式下。

应用软件可以选择和控制信号的产生或定时值的捕获。图 11.21 介绍了不同输出模式的基本功能。举例时用 OUT1 作为说明。

- |         |   |
|---------|---|
| 定时器:    | 定时器重复从 0 增计数到 CCR0。   |
| 输出模式 0: | 输出信号 Outx 由每个捕获/比较模块中的控制寄存器 CCTLx 中的 OUTx 位定义。它与任何定时功能无关, 并且是在软件的完全控制下。         |
| 输出模式 1: | 当定时器计数值等于 CCR1 时输出置位。由 EQU0 信号(CCIFG0)引起的中断可用于比较寄存器 x 的修改。                      |
| 输出模式 2: | 当定时器计数值等于 CCR1 时输出翻转。当定时器计数值等于 CCR0 时输出复位, 同时定时器也复位。基本应用是 PWM 功能或与其他输出一起产生相位关系。 |
| 输出模式 3: | 当定时器计数值等于 CCR1 时输出置位。当定时器计数值等于 CCR0 时输出复位, 同时定时器也复位。基本应用是 PWM 功能或与其他输出一起产生相位关系。 |
| 输出模式 4: | 当定时器计数值等于 CCR1 时输出翻转。输出周期是定时器周期的两倍。与任何其他输出的相位关系取决于 CCRx 数据的选择。                  |
| 输出模式 5: | 当定时器计数值等于 CCR1 时输出复位。由 EQU0 信号(CINT0)引起的中断可用于修改比较寄存器 x。                         |
| 输出模式 6: | 当定时器计数值等于 CCR1 时输出翻转。当定时器计数值等于 CCR0   |

输出模式 7: 当定时器计数值等于 CCR1 时输出复位。当定时器计数值等于 CCR0 时输出置位,同时定时器也复位。基本应用是 PWM 功能或与其他输出一起产生相位关系。

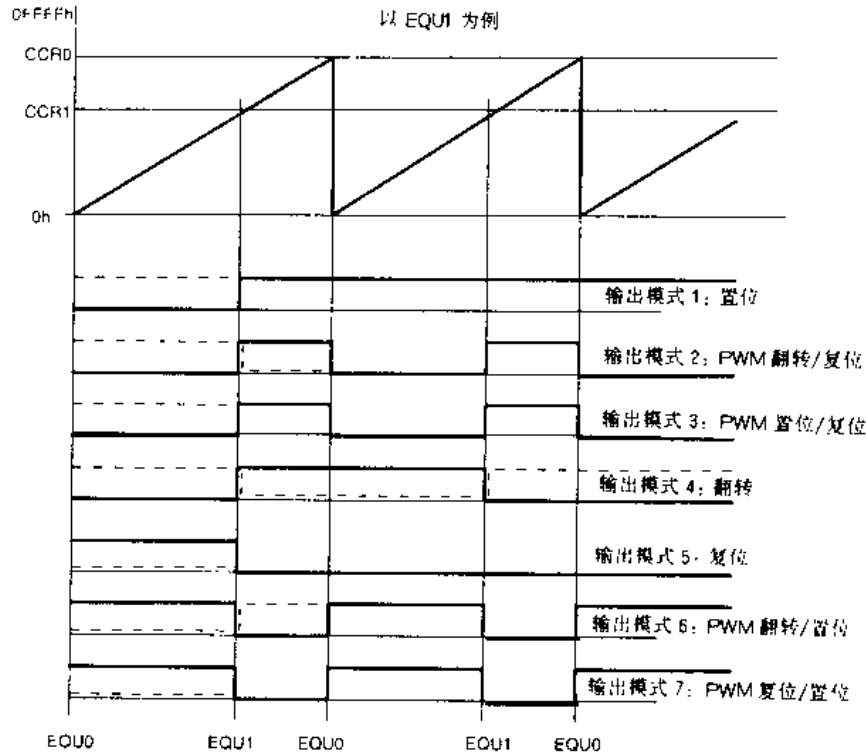


图 11.21 增计数模式的输出单元

### 11.3.2 Timer\_A 连续模式应用

如果 65 536 个时钟周期的定时周期对实际应用可用时,则可用这一模式。连续模式的主要应用是产生独立的软件时序。捕获/比较寄存器 CCR0 与其他 4 个捕获/比较寄存器 CCR<sub>x</sub> 的应用方法相同。

所有的输出模式对不同类型的應用都会有用。通过 CCTL<sub>x</sub> 寄存器的输出模式字 OMX2~OMX0 可选择各种输出模式下可能的输出信号。

应用软件可以选择和控制信号的产生或定时值的捕获。图 11.22 介绍了不同输出模式的基本功能。输出 OUT0 和 OUT1 仅作为说明之用。CCR0 中的值大于 CCR1 中的值。

定时器: 定时器重复从 00000h 增计数到 0FFFFh。

输出模式 0: 输出信号 Out<sub>x</sub> 由每个捕获/比较模块中的控制寄存器 CCTL<sub>x</sub> 中的 OUT<sub>x</sub> 位定义。它与任何定时功能无关,并且是在软件的完全控制下。

输出模式 1: 当定时器计数值等于 CCR1 时输出置位。由 EQU0 信号(CCIFG0)引起的中断可用于比较寄存器 x 的修改。

输出模式 2: 当定时器计数值等于 CCR1 时输出翻转。当定时器计数值等于 CCR0 时输出复位,同时定时器也复位。基本应用是产生脉冲。

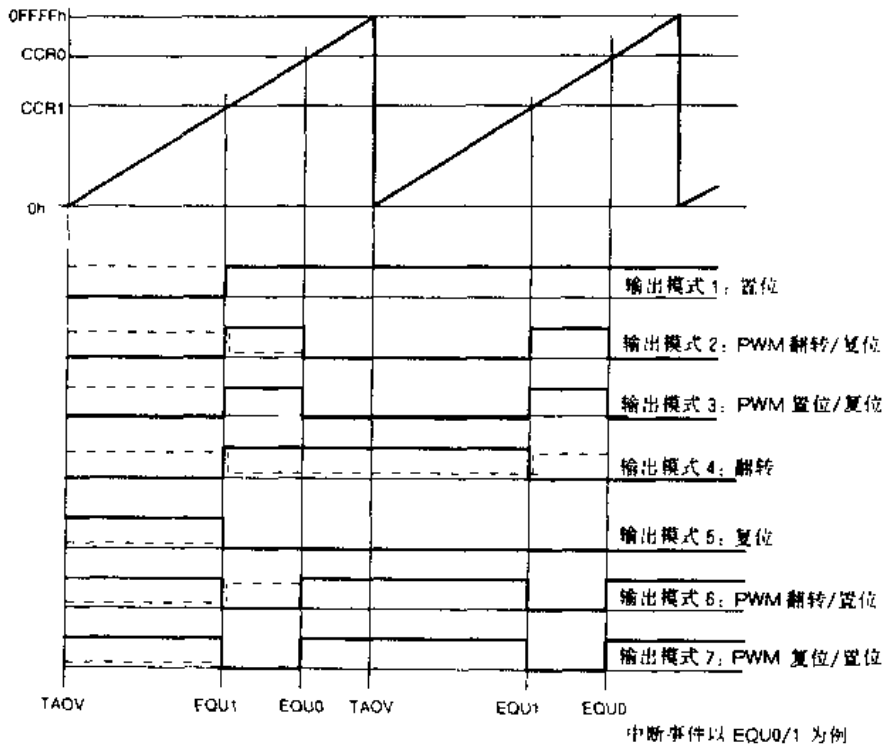


图 11.22 连续模式中的输出单元

- 输出模式 3: 当定时器计数值等于 CCR1 时输出置位。当定时器计数值等于 CCR0 时输出复位。基本应用是产生脉冲。
- 输出模式 4: 当定时器计数值等于 CCR1 时输出翻转, 输出周期是定时器周期的两倍。与任何其他输出的相位关系取决于 CCR<sub>x</sub> 数据的选择。
- 输出模式 5: 当定时器计数值等于 CCR1 时输出复位。由 EQU0 信号(CINT0)引起的中断可用于修改比较寄存器  $x_0$ 。
- 输出模式 6: 当定时器计数值等于 CCR1 时输出翻转。当定时器计数值等于 CCR0 时输出置位。基本应用是产生脉冲。
- 输出模式 7: 当定时器计数值等于 CCR1 时输出复位。当定时器计数值等于 CCR0 时输出置位。基本应用是产生脉冲。

连续模式可方便地被应用软件用于产生定时时间间隔, 如图 11.23 所示。如果中断允许,

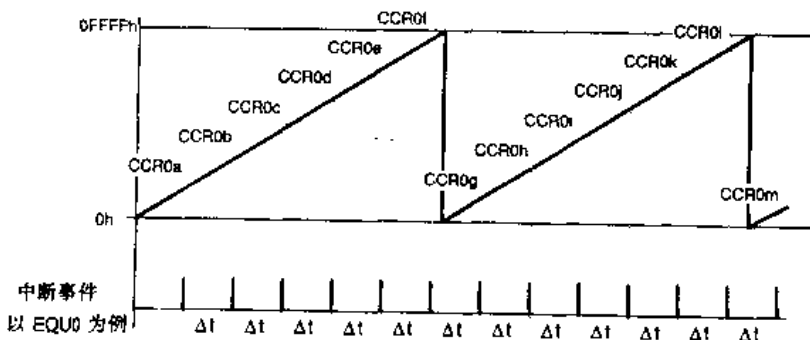


图 11.23 连续计数模式下产生定时间间隔时序图



在增减计数模式中,连续运行会产生 2 个中断:由捕获/比较模块 CCR0 产生的中断和当时器在减计数到达 0 时产生的中断。这 2 个中断可用于进行适当的输出脉冲修改。

增减计数模式使应用程序可能利用两个输出信号之间的“死区时间”。例如,两个输出驱动一个 H-电桥时,必须保证不同时输出为高以防止过载。可在短暂的可编程时间,即死区时间里,将两个输出都切换为“低”。相反的情况也可以,即如果需要两个输出也可编程为永不同时为“低”。在图 11.25 中, $t_{dead}$ 为

$$t_{dead} = T_{timer} \times (CCR1 - CCR3)$$

其中: $t_{dead}$ ——两个输出都必须为“低”的时间;

$T_{timer}$ ——定时器寄存器输入频率的周期;

$CCR_x$ ——比较寄存器 x 的值。

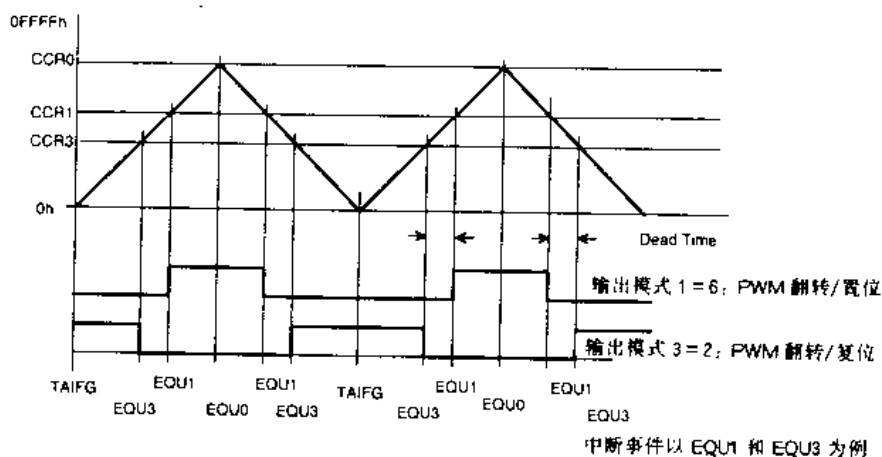


图 11.25 增减计数模式的输出单元(II)

### 11.3.4 Timer\_A 软件捕获应用

每个捕获/比较寄存器能被软件用于获得时间标记。它可用于各种目的,例如:

- 测量软件程序所用时间;
- 测量硬件事件之间的时间;
- 测量系统频率。

用 CCISx1、CCISx0 以及通过 CCMx1 和 CCMx0 选定的捕获模式,可执行软件捕获(见图 11.26)。捕获模式可选择用捕获信号 CCIx 的上升沿、下降沿或两个沿进行捕获。当捕获模式选择在两个沿进行捕获时,实现最简单。捕获输入信号选择为  $V_{CC}$ /高或 GND/低。CCISx1 置位,根据 CCISx0 选择是用  $V_{CC}$ /高还是 GND/低作为捕获信号。

用软件捕获的软件实例如下:

```

; 软件实例:软件捕获
;
; 捕捉/比较寄存器 CCRx 数据由软件产生
; 假定 CCMx1、CCMx0 和 CCISx1 置位
; 由 CCISx0 位选择 CCIx 信号为“高”或“低”
;

```





串行异步通信用给定的波特率发送和接收数据。工业标准规定了几种波特率。接收与发送可使用相同或不同的波特率。接收开始于信号的下降沿。接收器与此下降沿同步,且后续各位按选定的波特率接收。

用比较功能经输出单元从选定引脚上移出数据可以实现发送功能。每次在中断程序中设定数据可确保波特率。用模式 1 使输出引脚置位,用模式 5 使引脚复位。

用捕获/比较功能,通过控制寄存器的 SCCI<sub>x</sub> 位将引脚上的数据移入存储器可以实现接收功能。捕获定时器数据即接收信号的下降沿可识别接收开始时间。然后同一个捕获/比较模块改为比较功能。比较数据是捕获时间加上由波特率决定的半位时间。第一位用第一个比较事件 EQU<sub>x</sub> 锁存。后续位的扫描根据选定的波特率以同样方法完成。与位扫描相关的中断服务程序收集一个字的所有位,以供以后软件处理之用。

当选定为半双工通信时,需要用一个捕获/比较模块。用两个捕获/比较模块可实现全双工模式。在半双工模式中,接收和发送必须依次运行,且可以只用一条数据线。在全双工模式中,接收和发送可以并行执行。异步通信协议处理的时序如图 11.28 所示。

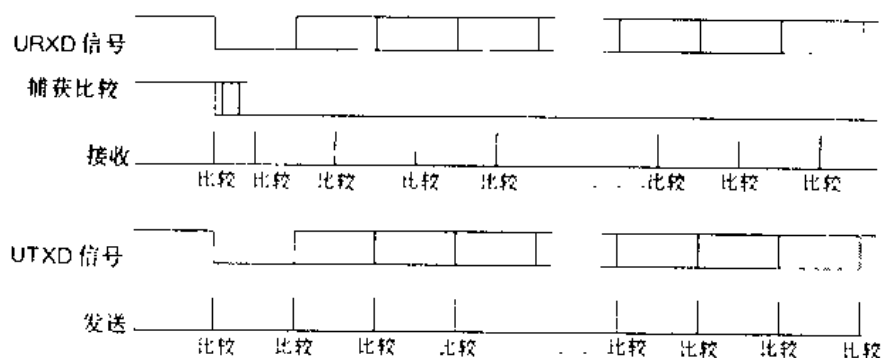


图 11.28 Timer\_A, 异步通信协议处理的时序图

## 11.4 Timer\_A 的特殊情况

本节讨论一些可能的特殊情况。所有定时器和比较功能的一条基本原理是,要执行选定的功能,定时器寄存器必须通过定时器时钟增 1 或减 1。

### 11.4.1 CCR0 用做周期寄存器

比较寄存器在定时器寄存器增计数的前半周期与定时器寄存器进行比较。如果 CCR0 用做周期寄存器,且新周期等于或大于旧周期,那么定时器将工作到新周期(见图 11.29),且无特殊情况需要注意。如果 CCR0 用做周期寄存器,且新周期小于旧周期(见图 11.30),那么当新值在定时器时钟的高电平时写入 CCR0 时,定时器在下一个时钟上升沿开始改变。定时器继续在下一个定时器时钟有效边沿加 1。如果新值在定时器时钟的低电平时写入 CCR0,那么定时器在定时器时钟的下一个上升沿加 1,并在第二个上升沿开始改变。

以上例子说明了增计数模式的不同情况。在增减计数模式下,当定时器工作在增计数,响应相同。如果在减计数时改变 CCR0,则定时器继续减计数到零。

定时器在增计数模式和增减计数模式中,以图 11.31 所示方式开始计数。

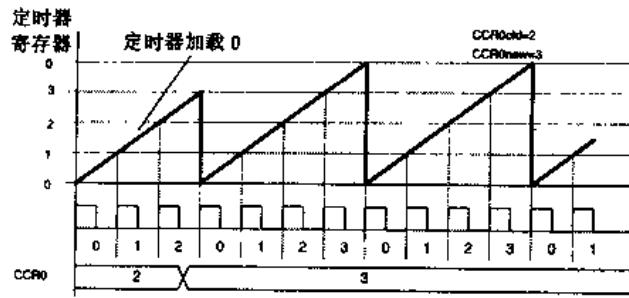


图 11.29 增计数模式下 CCR0 用做周期寄存器的时序图——新周期 $\geq$ 旧周期

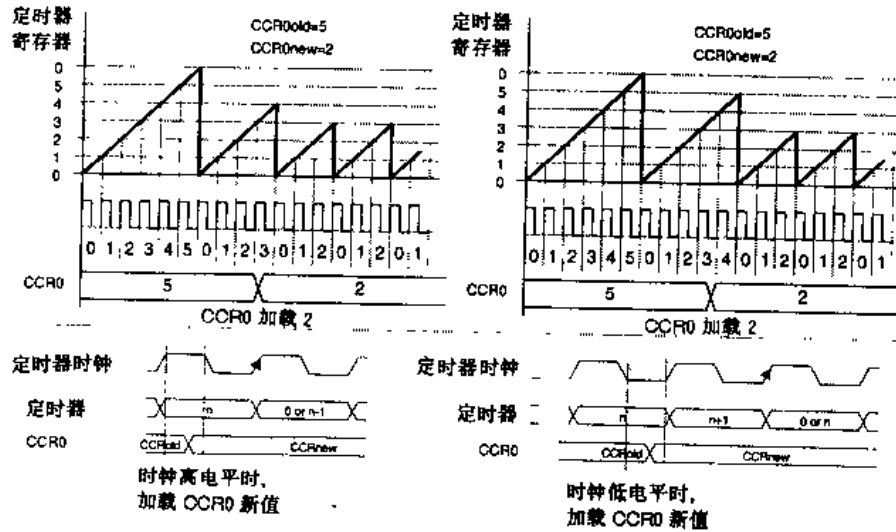


图 11.30 增计数模式下 CCR0 用做周期寄存器的时序图——新周期 $<$ 旧周期

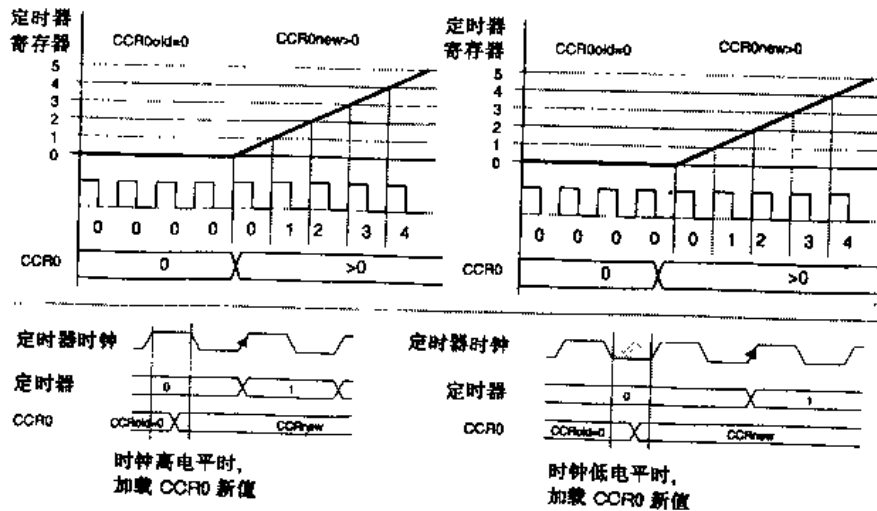


图 11.31 在增计数和增减计数模式下定时器开启时序图

### 11.4.2 定时器寄存器的启/停

定时器寄存器的开始和停止遵循与 CCR0 周期寄存器相同的基本规则,如图 11.32 所示。

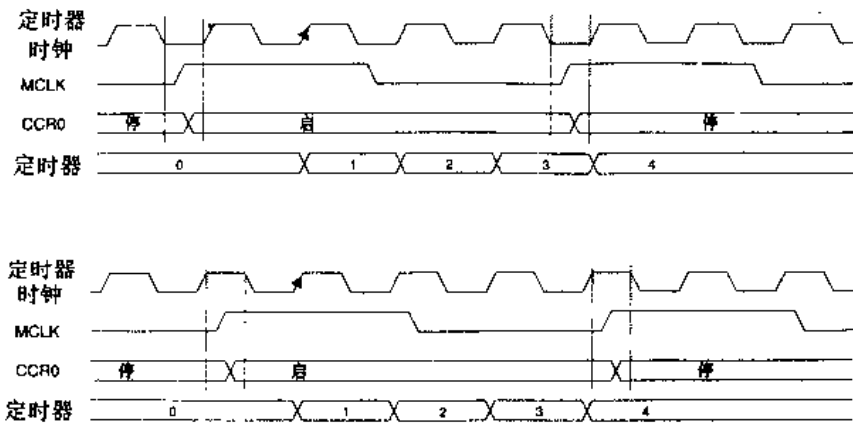


图 11.32 定时器寄存器的启/停时序图

选定的计数模式在定时器时钟的下降沿时装入。如果选中 3 个工作模式之一,则定时器寄存器在下一个上升沿加 1。如果定时器寄存器被停止,则下一个时钟上升沿不再使定时器寄存器加 1。

### 11.4.3 输出单元 0

所有输出单元具有相同的结构。输入用不同的控制信号定义特殊操作。控制信号中的其中两个是:相关模块  $x$  中的“定时器比较相等”寄存器(CCR $x$ )的比较器输出和模块 0 中的“定时器比较相等”寄存器(CCR0)的比较器输出。当模块  $x$  是输出单元 0 时,不是所有可能的工作模式都可以采用。图 11.33 为输出单元 0 的原理图。建议采用的模式为 0、1、4 和 5。

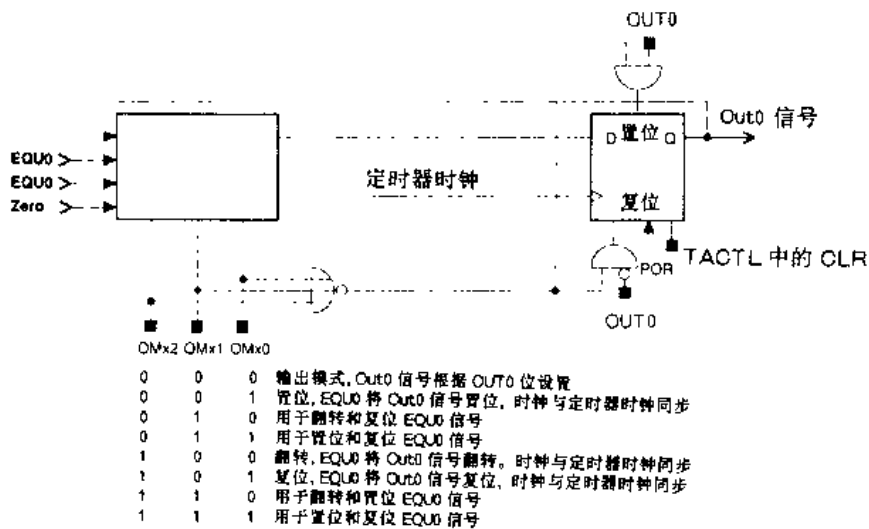


图 11.33 输出单元 0 原理图

## 第 12 章 USART 外围接口——UART 模式

通用串行同步异步通信接口 USART (Universal Synchronous Asynchronous Receive/Transmit) 是一个串行通道,它允许 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移入、移出 MSP430。USART 实现了两种功能,使得串行通信可以不同的模式进行。第一种功能是为熟悉的串行异步通信协议 UART;第二种功能是串行外围模块接口功能 SPI,它也得到了广泛应用。尽管这两种功能的硬件是公用的,但仍要针对它们的最终选择加以说明,在应用环境中通常就称为 UART 或 SPI。经过适当的软硬件设计,这两种功能可以交替使用。用控制寄存器 UCTL 中的控制位 SYNC 来选择所需的模式,即

- SYNC = 0: 选择异步通信模式 UART。
- SYNC = 1: 选择同步通信模式 SPI。

USART 以字节外围模块与 CPU 相连。微控制器通过 3 或 4 个引脚与外部系统相连。

图 12.1 是完整的 USART 功能框图。

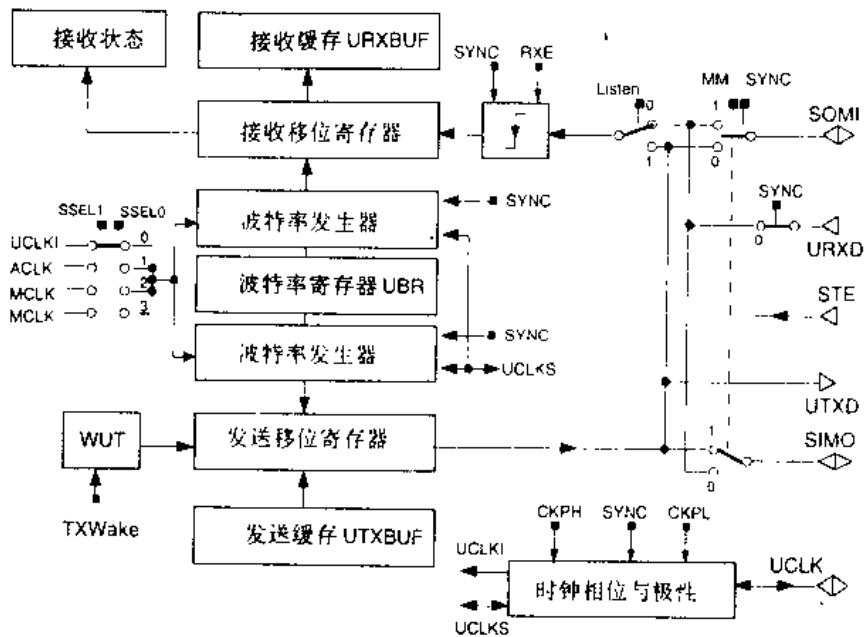


图 12.1 完整的 USART 功能框图

USART 控制寄存器 UCTL 的控制位 SYNC 复位时选择异步通信模式。

串行异步通信特性如下:

- 异步模式,包括线路空闲/地址位通信协议;
- 两个移位寄存器,串行数据移入 URXD,从 UTXD 移出;
- 从最低位开始的数据发送和接收;
- 可编程的发送/接收数据传输率;
- 状态标志。

图 12.2 为第一种模式,即 UART 模式下的 USART 功能框图。

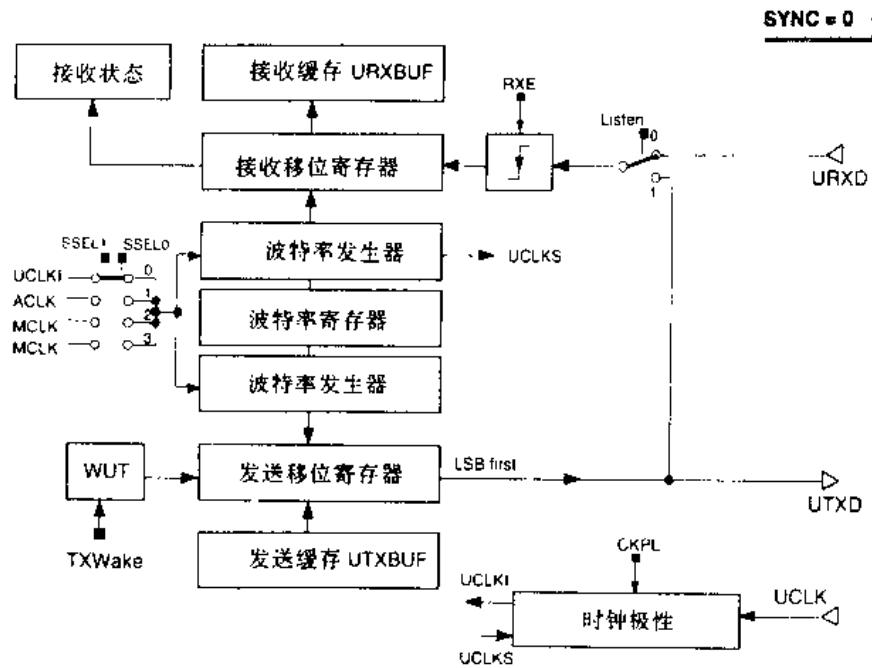


图 12.2 USART 功能框图——UART 模式

## 12.1 异步操作

在异步模式下,接收器实现自身与帧的同步;但外部发送接收设备并不使用这同一个时钟源。波特率是在本地产生的。

### 12.1.1 异步帧格式

异步帧格式(见图 12.3)由 1 个起始位、7 或 8 个数据位、奇/偶/无校验位、1 个地址位(地址位模式)、1 或 2 个停止位组成。通过选择时钟源和波特率寄存器的数据来确定位周期。

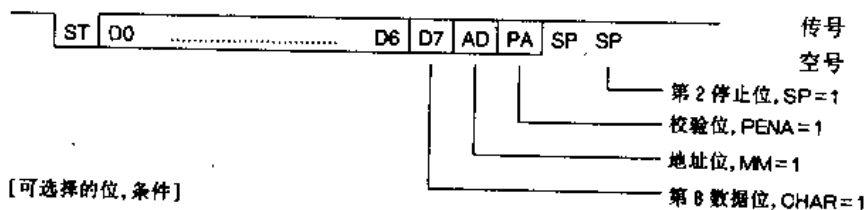


图 12.3 异步帧格式

以接收到有效的起始位来初始化接收操作。它包括检测 URXD 端口的下降沿,然后以 3 次采样多数表决方法取值,其中 2 次采样必须为 0。采样发生在下降沿后 BRCLK 周期的  $n/2 - x$ 、 $n/2$  和  $n/2 + x$  处。这一过程实现错误起始位的拒收及帧中各位的中心定位功能,在各位中心采用多数表决读数。其中  $x$  值的范围是 BRCLK 的  $1/32 \sim 1/63$ ;但是,在最低的 BRCLK,  $x$  的取值取决于波特率发生器的分频。图 12.4 为以  $n$  或  $n+1$  个 BRCLK 周期为例

的异步位格式。

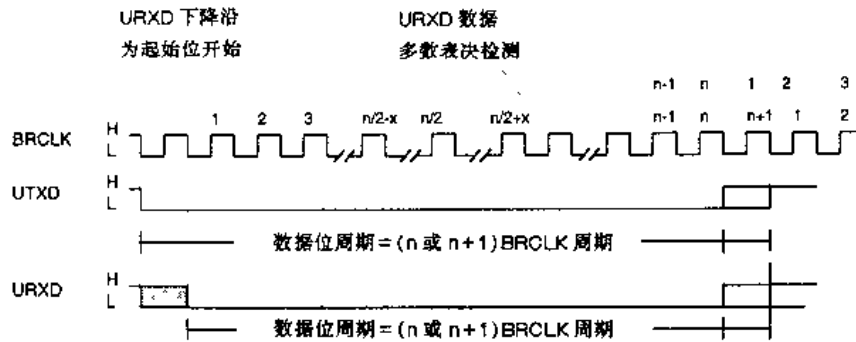


图 12.4 异步位格式——以 n 或 n+1 时钟周期为例

### 12.1.2 异步通信的波特率发生器

MSP430 的波特率发生器与其他标准的串行通信接口适配器实现的方法不同。

#### 1. 标准的波特率发生

标准的实现方法是——将一时钟源预分频,再加一固定分频器。该分频器通常是 16 分频。图 12.5 为标准的波特率发生原理图。波特率与时钟 BRCLK 的频率及分频因子 N 的关系为

$$\text{波特率} = f_{\text{BRCLK}} / (N \times 16)$$

用这一常用的方法发生波特率无法得到接近预分频输入频率  $f_{\text{BRCLK}}$  的波特率。例如,分频因子 N 不能为 18,同样非整数因子也无法使用,如 13.67。

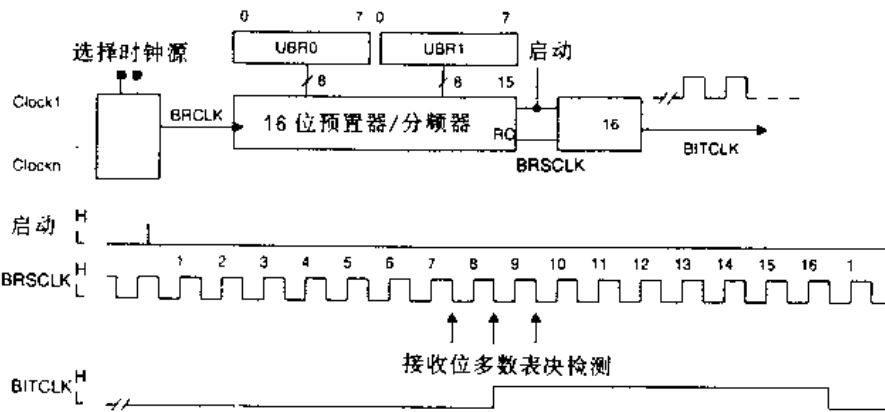


图 12.5 标准的波特率发生——非 MSP430 采用

例 1: 设 BRCLK 信号频率为 32 768 Hz, 波特率要求为 4 800, 则分频因子为 6.83; 但在标准波特率发生器里, 最小的因子为 16, 因此, 晶振频率和发生的波特率不能满足要求。

例 2: 设 BRCLK 信号的频率为 1.04 MHz (32 × 32 768 Hz), 波特率要求为 19 200, 则分频因子为 54.61; 但在标准波特率发生器里, 较接近的分频因子为 48 (3 × 16) 和 64 (4 × 16), 因此, 晶振频率和产生的波特率不能满足要求。晶振频率需要特别挑选以满足通信要求。而其他的原则, 诸如电流消耗、简单的实时钟功能或系统成本限制等就无法考虑了。

#### 2. MSP430 的波特率发生

MSP430 波特率发生器 (见图 12.6) 用了一个预分频/分频器和一个调整器。即使晶振频

率不是所需波特率的整数倍,这一组合也能正常工作,而且使通信协议可以工作在最大的波特率。采用这一技术,即使是手表晶体(32 768 Hz),波特率达到 4 800 和 9 600 也是可能的。它带来的优点很明显,使将复杂的 MSP430 工作模式选择在低功耗成为可能。

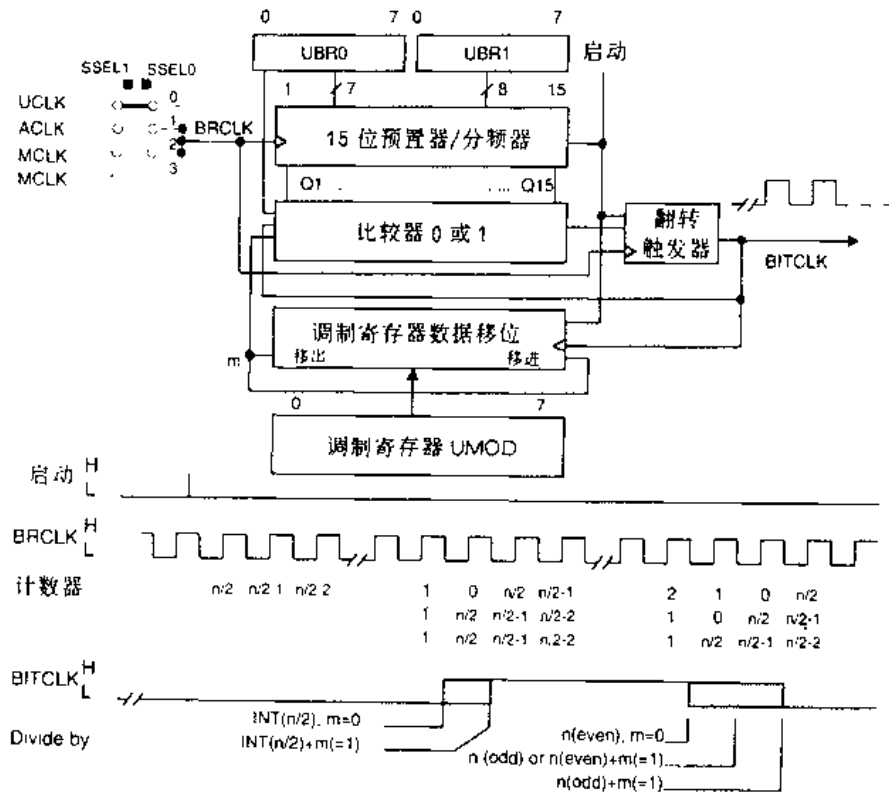


图 12.6 MSP430 波特率发生—— $n$  或  $n+1$  时钟周期举例

调整寄存器的 LSB 位(最低位)最先用于调整。它由起始位开始。置位的调整位使分频因子增加 1。

例 1: 假定 BRCLK 的频率为 32 768 Hz, 波特率要求为 4 800, 则分频因子为 6.83。MSP430 的 USART 采用分频因子 6 加上调整寄存器加载 6Fh(01101111)来产生波特率。即分频器按顺序 7、7、7、7、6、7、7、6、...来分频。在 8 位调整位都使用后,再重复这一顺序。

例 2: 假定 BRCLK 信号的频率为 1.04 MHz, 波特率要取为 19 200, 则分频因子为 54.61。MSP430 的 USART 采用分频因子 54(36h)加上调整寄存器加载 D5h 来产生波特率, 即分频器按顺序 55、54、55、54、55、54、55、55、...来分频。在 8 位调整位都使用后,再重复这一顺序。

表 12.1 列出了在手表晶体 32 768 Hz(ACLK)和 MCLK(设为 32 倍 ACLK)时,标准波特率所需的波特率寄存器和调整寄存器的数值。同时列出接收时的误差。此外,接收时的同步误差也应该加以考虑。

表 12.1 常用波特率、波特率数据及误差

波特率	分频		ACLK				MCLK (32×ACLK)			
	ACLK	MCLK	UBR1	UBR0	UMOD	最大误差/%	UBR1	UBR0	UMOD	最大误差/%
75	436.91	13 981.00	1	B4	FF	-0.1/0.3	36	9D	FF	0.0/0.1
110	297.89	9 532.51	1	29	FF	0.0/0.5	25	3C	FF	0.0/0.1
150	218.45	6 990.50	0	DA	55	0.0/0.4	1B	4E	FF	0.0/0.1
300	109.23	3 495.25	0	6D	22	-0.3/0.7	0D	A7	00	-0.1/0.0
600	54.61	1 747.63	0	36	D5	-1/1	06	D3	FF	0.0/0.3
1 200	27.31	873.81	0	1B	03	-4/3	03	69	FF	0.0/0.3
2 400	13.65	436.91	0	0D	6B	-6/3	01	B4	FF	0.0/0.3
4 800	6.83	218.45	0	06	6F	-9/11	0	DA	55	0.0/0.4
9 600	3.41	109.23	0	03	4A	-21/12	0	6D	03	-0.4/1.0
19 200		54.61					0	36	6B	-0.2/2.0
38 400		27.31					0	1B	03	-4/3
76 800		13.65					0	0D	6B	-6/3
115 200		9.10					0	09	08	5/7

对于接收模式和发送模式计算了最大误差。接收模式误差是针对每一位扫描在理想的中间位置的积累定时误差。发送模式误差是针对理想的位周期的积累定时误差。

MCLK 的最高频率见器件手册,它可以超过所举例子中的频率。

### 12.1.3 异步通信格式

当选择异步模式时,USART 模块支持 2 种多处理机通信模式。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。信息以一个多帧数据块,从一个指定的源送达一个或多个目的位置。USART 可识别数据块的起始,并能抑制接收端处理中断和状态信息,直至数据块的起始被识别。在这两种多处理机模式下,USART 数据交换过程可以用数据查询方式也可用接收中断方式来实现。

这两种异步多处理机协议,即线路空闲和地址位多处理机模式,实现了在多处理机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至最低,以节省电流消耗或处理所用资源。控制寄存器的 MM 位用来确定是地址位模式还是线路空闲多处理机模式。这两种格式采用唤醒发送、地址特性(TXWake 位)和激活(RXWake 位)等功能。URXWIE 和 URXIE 位控制这些模式的发送和接收。

### 12.1.4 线路空闲多处理机模式

在这种模式下,数据块被一段空闲时间分隔。在字符的第一个停止位之后接收到 10 个以上的“1”,则表示检测到接收线路空闲。线路空闲多处理机协议如图 12.7 所示。



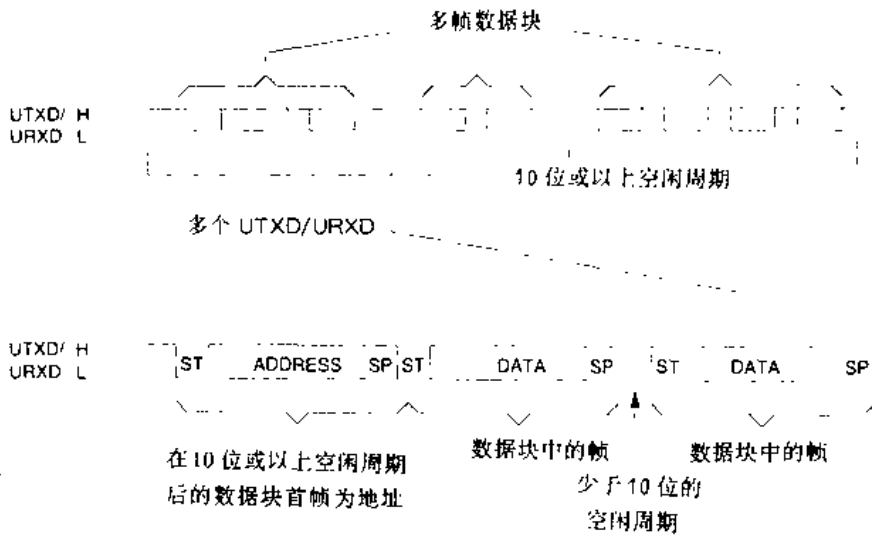


图 12.7 线路空闲多处理机协议

如果采用 2 位停止位,则将第 2 个停止位看做空闲周期的第 1 个传号。空闲周期后的第一个字符是地址字符。RXWake 位可作为地址字符的标记。在线路空闲多处理机格式中,当接收字符是地址字符时 RXWake 置位,并送入接收缓存中。USART 接收空闲检测如图 12.8 所示。

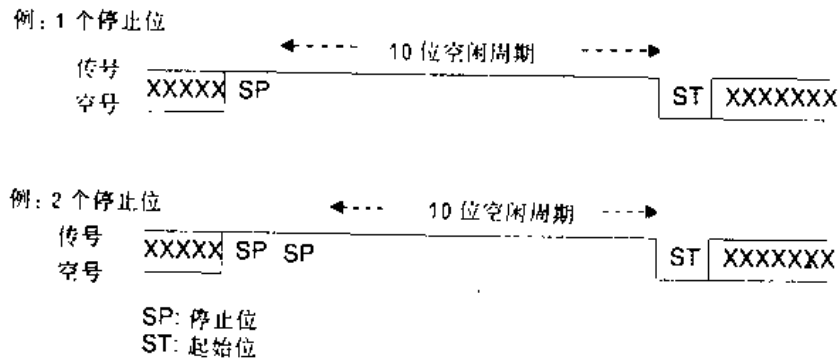


图 12.8 USART 接收空闲检测

正常情况下,如果 USART 接收控制寄存器的 URXWIE 置位,则字符在接收器以通常方法拼装;但是,不会将该字符送入接收缓存 URXBUF,也不会产生中断。只有当接收到地址字符时,接收器才暂时激活,以将字符送入 URXBUF,同时,将中断标志 URXIFG 置位。相应的错误状态标志位也会置位。应用软件可以验证接收到的地址。如果地址匹配,则应用软件将处理后续字符并执行适当的操作;如果地址不匹配,则处理机继续等待下一个地址字符的到来。URXWIE 位不会自动修改,必须由用户根据接收地址字符和非地址字符的需要去修改。

在线路空闲多处理机模式下,为了产生有效的地址字符识别,可以产生精确的空闲周期。与 TXWake 位相关的是临时唤醒标志 WUT(见图 12.9)。WUT 是一个内部标志,与 TXWake 一起构成双缓存。当发送器从 UTXBUF 装入数据时,WUT 也从 TXWake 装入,同时 TXWake 复位。



图 12.9 双缓存的 WUT 和 TX 移位寄存器

按下面步骤发送空闲帧以标识一个地址字符。首先 TXWake 位应置位,将任意数据(内容无关)写入 UTXBUF (UTXIFG 应置位)。当发送移位寄存器空 (TXEPT 置位)时,UTXBUF 的内容被送入发送移位寄存器,同时 TXWake 的值移入 WUT。如果此时 WUT 置位,则要发送的起始位、数据位和校验位被抑制,发送一个正好 11 位的空闲周期。在地址字符识别空闲周期之后移出串行端口的下一个数据是 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。将任意内容的字符写入 UTXBUF 是必要的。只有这样才能使 TXWake 的值移入 WUT 中。图 12.10 为 USART 发送空闲周期产生示意图。

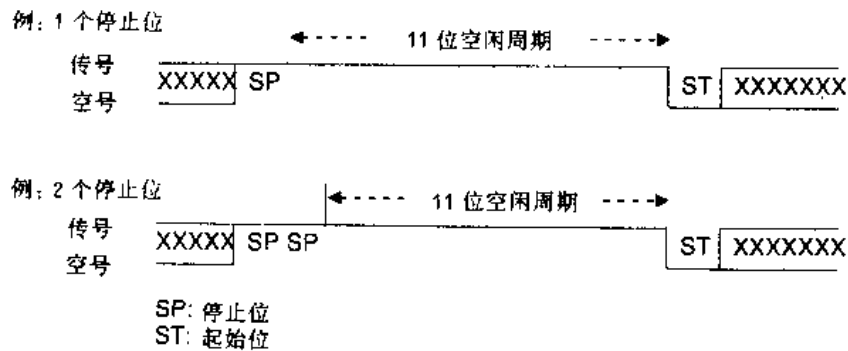


图 12.10 USART 发送空闲周期的产生示意图

### 12.1.5 地址位格式

在这种模式下,字符包含一附加的位作为地址标识。数据块的第一个字符携带一个置位的地址位,以表明该字符是一地址。当接收字符是地址时,RXWake 位置位,并且被送入接收缓存 URXBUF(允许接收时)。

正常情况下,如果 USART 的 URXWIE 位置位,则数据字符按通常方式在接收器拼装;但是,它们不会被送入接收缓存 URXBUF,也不会产生中断。只有当接收到一个地址位置位的字符时,接收器才暂时被激活,以将字符送入 URXBUF,同时 URXIFG 置位。相应的错误状态标志将置位。应用软件按有效利用资源、降低功耗的原则作后续操作。应用软件可验证接收到的地址。如果匹配,则处理机将读取数据块的后续数据;如果地址不匹配,则处理机将等待下一地址字符的到来。地址位多处理机协议如图 12.11 所示。

在地址位多处理机模式下,通过写 TXWake 位来控制字符的地址位。每当字符从 UTXBUF 传送至发送器时, TXWake 位装入字符的地址位。然后 TXWake 位将被 USART 清除。



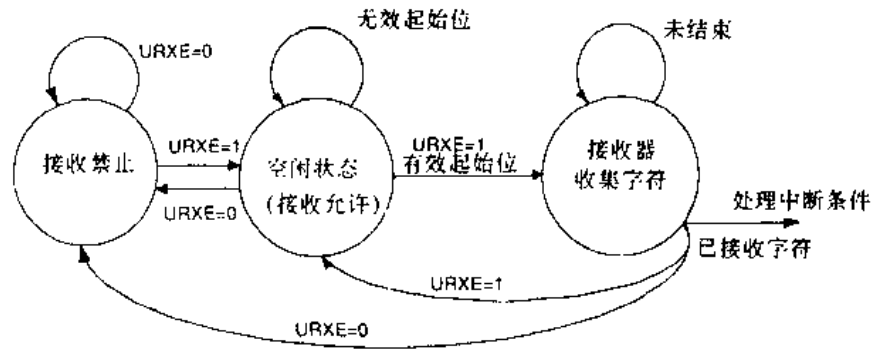


图 12.12 接收允许 URXE 状态图

**注意：URXE 重允许，UART 模式**

在接收器完全禁止时，接收器的重允许与通信线路的数据流是异步的。在接收器接收数据前查找线路空闲状态可以实现同步。

**12.2.2 USART 发送允许**

发送允许位 UTXE 的置位或复位，能允许或禁止串行数据线路上字符的发送。当 UTXE 复位时，已激活的发送不会停止，在完成已写入发送缓存内的全部数据的发送后发送才被禁止。如果发送已完成，则对发送缓存写入数据不会产生发送操作。发送允许状态图如图 12.13 所示。

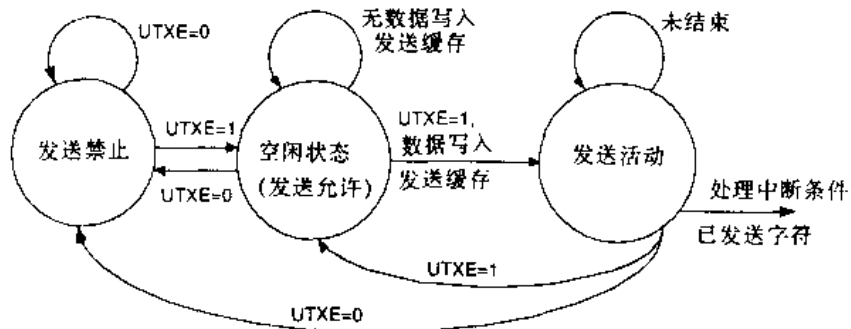


图 12.13 发送允许状态图

当 UTXE 复位时，发送缓存能照常写入，但不会启动发送。一旦 UTXE 置位，缓存内字符的发送立即开始。这一个字符能正确发送。

**注意：写 UTXBUF、UART 模式**

在 UTXBUF 未就绪且发送允许 (UTXE 置位) 时，不要对它写入数据；否则，移出的字符可能是随机的。

**12.2.3 USART 接收中断操作**

每次接收字符并将其装入接收缓存时，接收中断标志 URXIFG 置位或保持不变。

- 当 URXEIE 复位时，错误字符（校验、帧或打断错）不会使中断标志 URXIFG 置位，即



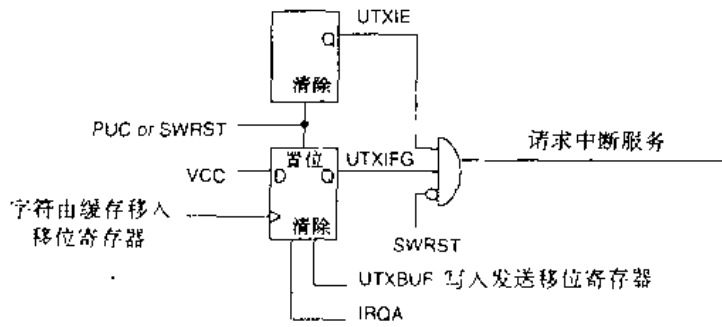


图 12.15 发送中断条件

### 12.3 控制与状态寄存器

USART 模块的硬件是字节结构的，必须用字节指令访问(后缀“B”)。

寄存器	缩写	类型	地址	初始状态
USART 控制寄存器	UCTL	读/写	070h	见位说明
发送控制寄存器	UTCTL	读/写	071h	见位说明
接收控制寄存器	URCTL	读/写	072h	见位说明
调整控制寄存器	UMCTL	读/写	073h	不变
波特率寄存器 0	UBR0	读/写	074h	不变
波特率寄存器 1	UBR1	读/写	075h	不变
接收缓存	URXBUF	读/写	076h	不变
发送缓存	UTXBUF	读	077h	不变

除非在功能描述中说明，发生 PUC 后各位的值都是随机的。

通过 PUC 或 SWRST 位使 USART 执行复位。PUC 后，SWRST 仍保持置位，且 USART 保持这一状态直到通过 SWRST 的复位来禁止 USART 复位。

SYNC 位决定 USART 模块处于异步还是同步模式。控制寄存器各位的功能在这两种模式下可以不同。本节的各位说明是指异步模式的，即 SYNC=0。它们在同步模式下的功能在 USART 的 SPI 部分中进行说明。

#### 12.3.1 USART 控制寄存器 UCTL

控制寄存器内的信息决定了 USART 的基本操作。例如：选择通信协议、通信模式和校验位。在通过 SWRST 复位来禁止 USART 复位之前，各位应根据选择的模式进行编程。USART 控制寄存器位定义如下：

UCTL (070h)

7				0			
PENA	PEV	SP	CHAR	Listen	SYNC	MM	SWRST
rw 0	rw 0	rw 0	rw 0	rw 0	rw 0	rw 0	rw 1

对 UCTL 中各位说明如下：

- 位 0: 如果 SWRST 置位, 则 USART 的状态机构和运行标志被初始化成复位状态。所有受影响的逻辑保持在复位状态, 直至 SWRST 复位。这意味着一次系统复位后, 只有对 SWRST 复位, USART 才能重新被允许。接收和发送允许标志 URXE 和 UTXE 不会因 SWRST 而更改。
- 位 1: 多处理机模式(地址位/线路空闲)。  
USART 模块支持两种多处理机协议: 线路空闲和地址位。选择多处理机模式会影响自动地址解码功能的操作。  
MM = 0: 线路空闲多处理机协议。  
MM = 1: 地址位多处理机协议。  
执行常规的异步协议时须将 MM 复位。
- 位 2: USART 模块的模式和功能选择。  
SYNC 位选择 USART 外围接口模块的功能。USART 的一些控制位在 UART 模式和 SPI 模式中有不同的功能。  
SYNC = 0: UART 模式。  
SYNC = 1: SPI 模式。
- 位 3: Listen 位选择是否将发送数据在内部反馈给接收器。  
Listen = 0: 无反馈。  
Listen = 1: 发送信号由内部反馈给接收器。每个 MSP430 的 USART 发送的数据同时被 USART 的接受器接收, 因此, 不再接收任何外部信号。
- 位 4: 字符长度。  
选择字符以 7 或 8 位发送。7 位字符不用 URXBUF 和 UTXBUF 的最高位, 这一位填“0”。  
CHAR = 0: 7 位。  
CHAR = 1: 8 位。
- 位 5: 停止位数。  
决定发送时停止位数。但是接收器只检测 1 位停止位。  
SP = 0: 1 位停止位。  
SP = 1: 2 位停止位。
- 位 6: 校验奇偶位。  
如果 PENA 置位(校验允许), 则 PEV 位按发送或接收字符、地址位(地址位多处理机模式)和校验位中的“1”的数量定义奇校验或偶校验。  
PEV = 0: 奇校验。  
PEV = 1: 偶校验。
- 位 7: 校验允许位。  
如果禁止校验, 则发送时不会产生校验位, 接收时也不期望收到这一位。因为校验位不是数据位之一, 所以接收到的校验位不传送入 URXBUF 中。在地址位多处理机模式中, 地址位包括在校验计算中。  
PEN = 0: 校验禁止。  
PEN = 1: 校验允许。

**注意：传号、空号定义**

传号电平与空闲状态信号电平相同。空号电平与传号电平相反。起始位总是空号。

**12.3.2 发送控制寄存器 UTCTL**

寄存器 UTCTL 控制与发送操作相关的 USART 硬件。UTCTL 的位定义如下：

UTCTL (071h)

7							0
未用	CKPL	SSEL1	SSEL0	URXSE	TXWake	未用	TXEPT
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

对 UTCTL 中各位说明如下：

- 位 0：发送器空标志 TXEPT 在发送移位寄存器和 UTXBUF 为空时置位，当数据写入 UTXBUF 时复位。它也在 SWRST 置位时置位。
- 位 1：未用。
- 位 2：TXWake 位用于控制多处理机通信模式的发送特性。通过装入 UTXBUF 启动一次发送，用 TXWake 位状态初始化地址识别特性。它不必清除，一旦被传送至 WUT(Wake Up Temporay, 暂时唤醒)，USART 硬件自动将它清除；SWRST 位也能将它清除。
- 位 3：接收触发沿控制位置位请求接收中断服务，这时相应的允许位和 GIE 应置位。用这一位的优点是能在中断服务程序中启动微控制器时钟系统(包括 MCLK)，还能通过调整模式控制位来保持运行。即使系统因关闭 MCLK 进入低功耗模式，USART 也能在选定的 MCLK 下工作。
- 位 4、5：时钟源选择 0 和 1。时钟源选择位确定用于波特率发生器的时钟源。
- |             |     |               |
|-------------|-----|---------------|
| SSEL1、SSEL0 | 0   | 选择外部时钟 UCLKI。 |
|             | 1   | 选择辅助时钟 ACLK。  |
|             | 2、3 | 选择系统主时钟 MCLK。 |
- 位 6：时钟极性 CKPL。CKPL 位控制 UCLKI 信号的极性。
- CKPL = 0：UCLKI 信号与 UCLK 极性相同。
- CKPL = 1：UCLKI 信号与 UCLK 极性相反。
- 位 7：未用。

**12.3.3 接收控制寄存器 URCTL**

URCTL 控制与接收操作相关的 USART 硬件，并保存由最新写入 URXBUF 的字符引起的出错状况和唤醒条件。一旦 FE、PE、OE、BRK、RXERR 或 RXWake 的任何一位置位，不能通过接收下一个字符来复位。它们的复位要通过访问接收缓存 URXBUF、USART 的软件复位 SWRST、系统复位 PUC 或直接用指令修改。

URCTL 的位定义如下：



URCTL (072h)

7							0
FE	PE	OE	BRK	URXEIE	URXWIE	RXWake	RXERR
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

对 URCTL 中各位说明如下:

位 0: 接收错误位 RXERR 置位则表明有一个或多个出错标志(FE、PE、OE 或 BRK) 置位。当用指令清除出错位时,它不会复位。

位 1: 接收唤醒检测。

当接收的字符是一地址字符且被送入接收缓存时, RXWake 位置位。

地址位多处理机模式: 接收字符地址位置位时, RXWake 位置位。

线路空闲多处理机模式: 接收字符前检测到 URXD 线路空闲(11 位传号)时, RXWake 置位。

访问接收缓存 URXBUF、USART 软件复位 SWRST 或系统复位 PUC 都将使 RXWake 复位。

位 2: 接收唤醒中断允许位 URXWIE 选择设置中断标志位的字符类型。

URXWIE = 0: 接收到的每一个字符都将使 URXIFG 置位。

URXWIE = 1: 只有地址字符才使 URXIFG 置位。在两种多处理机模式中工作相同。

唤醒中断允许的特征依赖于接收出错字符的特征。见 URXEIE 位说明。

位 3: 接收出错字符中断允许位 URXEIE 选择是否允许出错字符使 URXIFG 置位。

URXEIE = 0: 每一个接收的出错字符都不改变 URXIFG 位。

URXEIE = 1: 根据 URXWIE 位的设置,所有字符都可使 URXIFG 置位。

URXEIE	URXWIE	字符出错	地址字符	接收字符后的 URXIFG
0	x	1	x	不变
0	0	0	x	置位
0	1	0	0	不变
0	1	0	1	置位
1	0	x	x	置位(接收所有字符)
1	1	x	0	不变
1	1	x	1	置位

位 4: 当一次打断发生并且 URXEIE 置位时,打断检测位 BRK 置位。如果 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,则被识别为打断。在检测到打断后,不能通过接收字符清除标志;只有通过 SWRST 位、系统复位和读取 URXBUF 来复位。

位 5: 当一个字符写入 URXBUF 时前一字符还未被读出,则溢出标志 OE 置位。这时前一字符因被覆盖而丢失。OE 通过 SWRST 位、系统复位和读取 URXBUF 来复位。

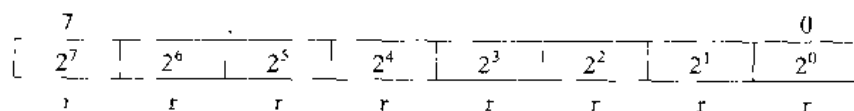
位 6: 当一个接收字符中“1”的个数和它的校验位不符并被装入接收缓存时,校验错标



### 12.3.5 USART 接收数据缓存 URXBUF

接收缓存 URXBUF 含有此前从接收移位寄存器传来的数据。读取 URXBUF 中数据将使复位接收出错位、RXWake 位和中断标志 URXIFG 位复位。URXBUF 的位定义如下：

URXBUF (076h)



在 7 位字长模式下, URXBUF 的最高位总是 0。

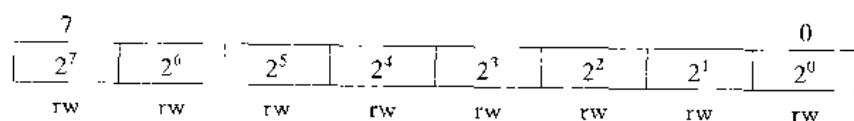
当接收和控制条件为真时, 接收缓存装入当前接收的字符。

URXEIE	URXWIE	装入 URXBUF	PE	FE	BRK
0	1	无差错地址字符	0	0	0
1	1	所有的地址字符	x	x	x
0	0	无差错字符	0	0	0
1	0	所有字符	x	x	x

### 12.3.6 USART 发送数据缓存 UTXBUF

发送缓存含有当前要由发送器发送的数据。发送数据缓存 UTXBUF 结构如下：

UTXBUF (077h)



UTXIFG 标志表示 UTXBUF 已准备好接收下一个要发送的字符。

将数据写入 UTXBUF 将初始化发送功能。如果发送移位寄存器为空或即将为空, 则数据的发送立即开始。

只有当 UTXBUF 为空时, 数据才能写入缓存; 否则, 可能发送不可预料的字符。

## 12.4 UART 模式——低功耗模式应用特性

MSP430 有许多功能和操作特性支持实现基于 MSP430 结构的超低功耗系统, 即

- 利用对 UART 帧的敏感作为启动条件, 系统可从任意的处理模式开始运行;
- 用最低输入时钟频率来实现波特率;
- 支持多处理机模式来减少 MSP430 资源的使用。

### 12.4.1 由 UART 帧启动接收操作

当波特率发生采用系统主时钟 MCLK 时, 能最有效地在接收通道作启动检测; 但是, 整个系统可以在没有 MCLK 的情况下工作。接收启动条件(见图 12.16)是 URXD 信号的下降沿。

在 URXIE 和 GIE 允许时,能触发中断标志 URXS 来请求一次中断服务。这样 MSP430 将返回到活动模式,并具备在 MCLK 和 ACLK 活动状态下的全部系统性能。

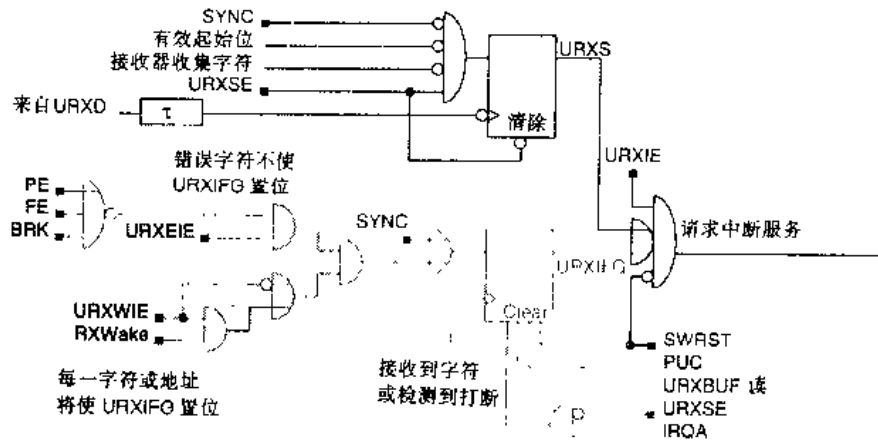


图 12.16 接收启动条件

有 3 种字符流不能使中断标志 URXIFG 置位,即

- 错误字符 (URXEIE = 0);
- 地址字符 (URXWIE = 1);
- 检测到无效起始位。

中断软件应处理这些情况。中断处理必须正确设置时钟系统环境。时钟系统将连续运行 (意味着电流消耗) 到通过软件来修改配置。如果 CPU 工作在活动模式,且时钟系统正常工作,就不必使用起始条件检测。

图 12.17、图 12.18 和图 12.19 为用 URXS 标志接收起始位的时序图。

进入 USART 模块的 URXD) 信号首先进入去毛刺电路,使毛刺不能触发接收起始位标志 URXS。这可防止 URXD 线路上的小毛刺启动通信模块。在噪声环境里,因为毛刺不会启动系统和 USART,所以电流消耗也会降低。

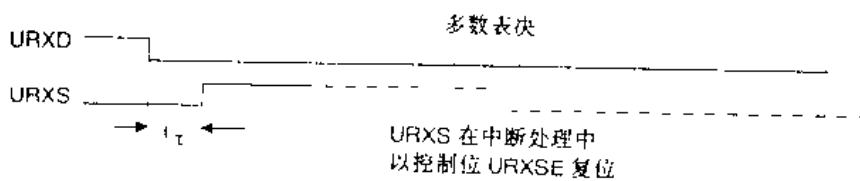


图 12.17 用 URXS 标志接收起始位时序——接受起始位

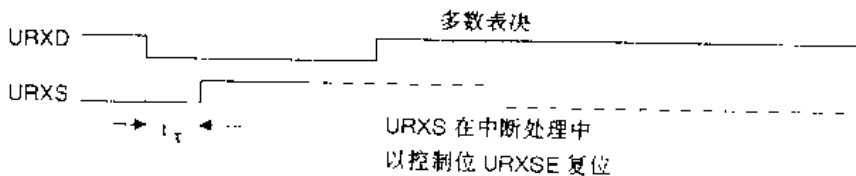


图 12.18 用 URXS 标志接收起始位时序——不接受起始位

当 URXD 信号超过去毛刺时间  $t_r$ , 但起始位的多数表决检测失败时, UART 停止字符的





相对误差。MSP430 发送位定时误差示意图如图 12.21 所示。

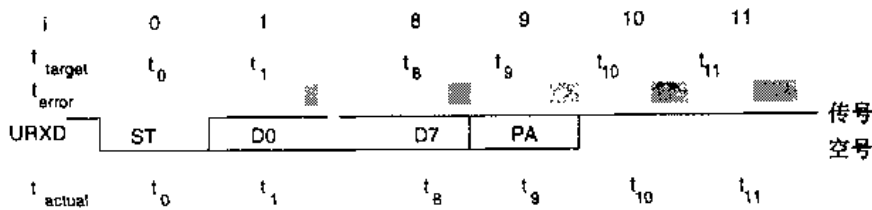


图 12.21 MSP430 发送位定时误差

即使每位误差很小,最终也会产生大的误差。必须考虑到它们是积累的而不是相对的。每一位的误差可通过下面公式来计算,即

$$\text{误差} = \frac{\sum_{i=0}^{n-1} t_{\text{actual}i} - \sum_{i=0}^{n-1} t_{\text{target}i}}{t_{\text{baudrate}}} \times 100\%$$

或

$$\text{误差} = \left( \frac{\text{波特率}}{f_{\text{BRCLK}}} \times \left( (i+1) \times \text{UBR} + \sum_{j=0}^{i-1} m_j \right) - (i+1) \right) \times 100\%$$

式中:  $t_{\text{target}i}$ ——目标波特率定时;

$t_{\text{actual}i}$ ——实际波特率定时;

$t_{\text{baudrate}}$ ——波特率定时;

$f_{\text{BRCLK}}$ ——输入频率,选自 UCLK、ACLK 或 MCLK;

$i$ —— $i=0$  针对起始位,  $i=1 \sim 7$  分别针对数据位 D0~D7。

## 2. 例 1

设: 波特率 = 2 400;

$f_{\text{BRCLK}} = 32\,768 \text{ Hz (ACLK)}$ ;

UBR = 13(因为理想的分频因子为 13.67);

$m = 6\text{h}$ ,  $m_7 = 0$ ,  $m_6 = 1$ ,  $m_5 = 1$ ,  $m_4 = 0$ ,  $m_3 = 1$ ,  $m_2 = 0$ ,  $m_1 = 1$ ,  $m_0 = 1$ , 最低位  $m_0$  最先使用。

每一位的误差计算如下:

起始	误差 = (波特率/ $f_{\text{BRCLK}}$ × ((0 + 1) × UBR + 1) - 1) × 100 % = 2.54 %
D0	误差 = (波特率/ $f_{\text{BRCLK}}$ × ((1 + 1) × UBR + 2) - 2) × 100 % = 5.08 %
D1	误差 = (波特率/ $f_{\text{BRCLK}}$ × ((2 + 1) × UBR + 2) - 3) × 100 % = 0.29 %
D2	误差 = (波特率/ $f_{\text{BRCLK}}$ × ((3 + 1) × UBR + 3) - 4) × 100 % = 2.83 %
D3	误差 = (波特率/ $f_{\text{BRCLK}}$ × ((4 + 1) × UBR + 3) - 5) × 100 % = -1.95 %
D4	误差 = (波特率/ $f_{\text{BRCLK}}$ × ((5 + 1) × UBR + 4) - 6) × 100 % = 0.59 %
D5	误差 = (波特率/ $f_{\text{BRCLK}}$ × ((6 + 1) × UBR + 5) - 7) × 100 % = 3.13 %
D6	误差 = (波特率/ $f_{\text{BRCLK}}$ × ((7 + 1) × UBR + 5) - 8) × 100 % = -1.66 %
D7	误差 = (波特率/ $f_{\text{BRCLK}}$ × ((8 + 1) × UBR + 6) - 9) × 100 % = 0.88 %
校验	误差 = (波特率/ $f_{\text{BRCLK}}$ × ((9 + 1) × UBR + 7) - 10) × 100 % = 3.42 %
停止 1	误差 = (波特率/ $f_{\text{BRCLK}}$ × ((10 + 1) × UBR + 7) - 11) × 100 % = -1.37 %

停止 2 误差 = (波特率 /  $f_{BRCLK} \times ((11 + 1) \times UBR + 8) - 12) \times 100\% = 1.17\%$

波特率寄存器和调整寄存器所需要的标准波特率数据如表 12.2 所列。这些数据是针对 32 768 Hz 手表晶振 (ACLK) 的, 并假定 MCLK 的频率是 ACLK 的 32 倍。表中列出了发送和接收的计算误差。在接收时还应考虑同步误差。

表 12.2 常用波特率、波特率数据和误差

波特率	分频信号		ACLK (32 768 Hz)			最大 TX 误差/%	最大 RX 误差/%	同步 RX 误差/%	MCLK (1 048 576 Hz)			最大 TX 误差/%	最大 RX 误差/%
	ACLK	MCLK	UBR1	UBR0	UMOD				UBR1	UBR0	UMOD		
75	436 91	13 981 00	1	1B	FF	0 1/0 3	-0 1/0 3	+2	36	9D	FF	0 0/0 1	+2
110	297 89	9 532 51	1	29	FF	0 0/0 5	0 0/0 5	+3	25	3C	FF	0 0/0 1	+3
150	218 45	6 990 50	0	DA	55	0 0/0 4	0 0/0 4	+2	1B	4E	FF	0 0/0 1	+2
300	109 23	3 495 25	0	6D	22	-0 3/0 7	0 3/0 7	+2	0D	A7	00	-0 1/0 0	+2
600	54 61	1 747 63	0	36	D5	1/1	1/1	+2	06	D3	FF	0 0/0 3	+2
1 200	27 31	873 81	0	1B	03	4/3	4/3	+2	03	69	FF	0 0/0 3	+2
2 400	13 65	436 91	0	0D	6B	6/3	6/3	+4	01	14	FF	0 0/0 3	+2
4 800	6 83	218 45	0	06	6F	-9/11	-9/11	+7	0	DA	55	0 0/0 4	+2
9 600	3 41	109 23	0	03	4A	-21/12	21/12	+15	0	6D	03	0 4/1 0	+2
19 200		54 61							0	36	6B	-0 2/2 0	+2
38 400		27 31							0	1B	03	4/3	+2
76 800		13 65							0	0D	6B	6/3	+4
115 200		9 10							0	09	08	-5/7	+7

同步误差来源于 URXD 端口数据信号与内部时钟系统之间的异步定时。接收信号与 BRCLK 时钟是同步的。BRCLK 时钟频率比位定时快 16~31 倍。

- $f_{BRCLK} = f_{BRCLK}, \quad N \leq 1Fh$
- $f_{BRCLK} = f_{BRCLK}/2, \quad 20h \leq N \leq 3Fh$
- $f_{BRCLK} = f_{BRCLK}/4, \quad 40h \leq N \leq 7Fh$
- $f_{BRCLK} = f_{BRCLK}/8, \quad 80h \leq N \leq FFh$
- $f_{BRCLK} = f_{BRCLK}/16, \quad 100h \leq N \leq 1FFh$
- $f_{BRCLK} = f_{BRCLK}/32, \quad 200h \leq N \leq 3FFh$
- $f_{BRCLK} = f_{BRCLK}/64, \quad 400h \leq N \leq 7FFh$
- $f_{BRCLK} = f_{BRCLK}/128, \quad 800h \leq N \leq FFFh$
- $f_{BRCLK} = f_{BRCLK}/256, \quad 1000h \leq N \leq 1FFFh$
- $f_{BRCLK} = f_{BRCLK}/512, \quad 2000h \leq N \leq 3FFFh$
- $f_{BRCLK} = f_{BRCLK}/1 024, \quad 4000h \leq N \leq 7FFFh$
- $f_{BRCLK} = f_{BRCLK}/2 048, \quad 8000h \leq N \leq FFFFh$

对于起始位检测的目标波特率定时  $t_{target0}$  是波特率定时  $t_{baudrate}$  的一半, 因为位测试是在位周期的中间进行的。其他后续位的目标波特率定时  $t_{targeti}$  就是波特率定时  $t_{baudrate}$ 。

$$\text{误差} = \frac{t_{actual0} + t_{target0}}{0.5 \times t_{target0}} + \frac{\sum_{i=1}^{n-1} t_{actuali} - \sum_{i=1}^{n-1} t_{targeti}}{t_{targeti}} \times 100\%$$

或

$$\text{误差} = \left( \frac{\text{波特率}}{f_{BRCLK}} \times ; 2 \times [m0 + INT(UBR/2)] + (i \times UBR + \sum_{i=1}^n mi) ; - 1 - i \right) \times 100\%$$



式中： $f_{BRCLK}$ ——输入频率，选自 UCLK、ACLK 或 MCLK；  
 $i$ —— $i=0$  针对起始位， $i=1\sim 7$  分别针对数据位 D0~D7。

MSP430 接收位定时误差如图 12.22 所示。

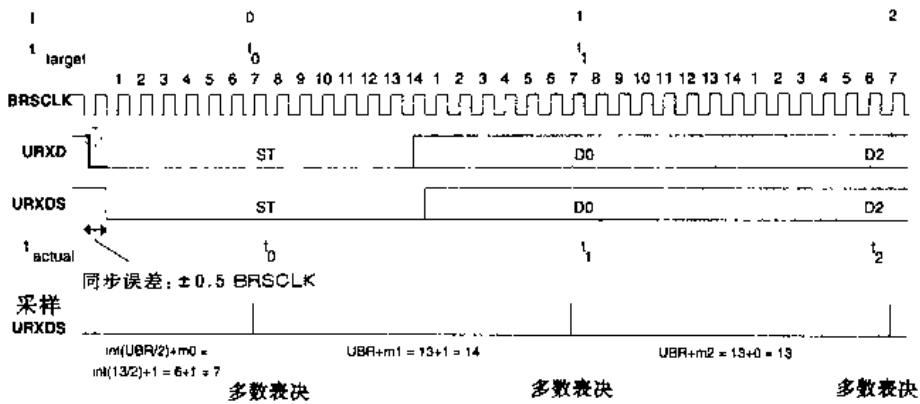


图 12.22 MSP430 接收位定时误差

### 3. 例 2

设：波特率 = 2 400；

$f_{BRCLK} = 32\ 768\ \text{Hz}$  (ACLK)；

UBR = 13, 因为理想的分频因子为 13.67；

$m = 6Bh$ ,  $m_7 = 0$ ,  $m_6 = 1$ ,  $m_5 = 1$ ,  $m_4 = 0$ ,  $m_3 = 1$ ,  $m_2 = 0$ ,  $m_1 = 1$ ,  $m_0 = 1$ , 最低位  $m_0$  最先使用。

误差计算如果如下：

起始	误差 = (波特率/ $f_{BRCLK} \times (2 \times (1 + 6) + (0 \times UBR + 0)) - 0 - 1) \times 100\% = 2.54\%$
D0	误差 = (波特率/ $f_{BRCLK} \times (2 \times (1 + 6) + (1 \times UBR + 1)) - 1 - 1) \times 100\% = 5.08\%$
D1	误差 = (波特率/ $f_{BRCLK} \times (2 \times (1 + 6) + (2 \times UBR + 1)) - 1 - 2) \times 100\% = 0.29\%$
D2	误差 = (波特率/ $f_{BRCLK} \times (2 \times (1 + 6) + (3 \times UBR + 2)) - 1 - 3) \times 100\% = 2.83\%$
D3	误差 = (波特率/ $f_{BRCLK} \times (2 \times (1 + 6) + (4 \times UBR + 2)) - 1 - 4) \times 100\% = -1.95\%$
D4	误差 = (波特率/ $f_{BRCLK} \times (2 \times (1 + 6) + (5 \times UBR + 3)) - 1 - 5) \times 100\% = 0.59\%$
D5	误差 = (波特率/ $f_{BRCLK} \times (2 \times (1 + 6) + (6 \times UBR + 4)) - 1 - 6) \times 100\% = 3.13\%$
D6	误差 = (波特率/ $f_{BRCLK} \times (2 \times (1 + 6) + (7 \times UBR + 4)) - 1 - 7) \times 100\% = -1.66\%$
D7	误差 = (波特率/ $f_{BRCLK} \times (2 \times (1 + 6) + (8 \times UBR + 5)) - 1 - 8) \times 100\% = 0.88\%$
校验	误差 = (波特率/ $f_{BRCLK} \times (2 \times (1 + 6) + (9 \times UBR + 6)) - 1 - 9) \times 100\% = 3.42\%$
停止 1	误差 = (波特率/ $f_{BRCLK} \times (2 \times (1 + 6) + (10 \times UBR + 6)) - 1 - 10) \times 100\% = -1.37\%$
停止 2	误差 = (波特率/ $f_{BRCLK} \times (2 \times (1 + 6) + (11 \times UBR + 7)) - 1 - 11) \times 100\% = 1.17\%$

### 4. 波特率总结

系统选择产生适合于串行通信位流的波特率，允许波特率接近 USART 的输入时钟。通过调整各位的定时能达到低的积累误差。实际工作中，如误差在 20%~30% 以内是可以进行正确的串行通信的。

## 第 13 章 USART 外围接口——SPI 模式

同步接口是一个串行通道,它允许 7 位或 8 位数据流以外部或内部编程的速率移入或移出 MSP430。USART 以字节外围模块与 CPU 连接。它通过 3 或 4 个外部引脚使微控制器与外部系统相连。

USART 的串行同步通信特性如下:

- 控制寄存器中的控制位 SYNC 置位,以选择同步通信模式;
- 经 SOMI、SIMO、UCLK 和 STE 引脚连接,支持 3 或 4 线 SPI 操作;
- 可选择主模式或从模式;
- 接收和发送有各自的移位寄存器(URXBUF 和 UTXBUF);
- 接收和发送双缓存;
- 时钟极性和时钟相位可控;
- 主模式的时钟频率可控;
- 字符长度为 7 或 8 位。

图 13.1 为 USART 在 SPI 模式下的原理框图。

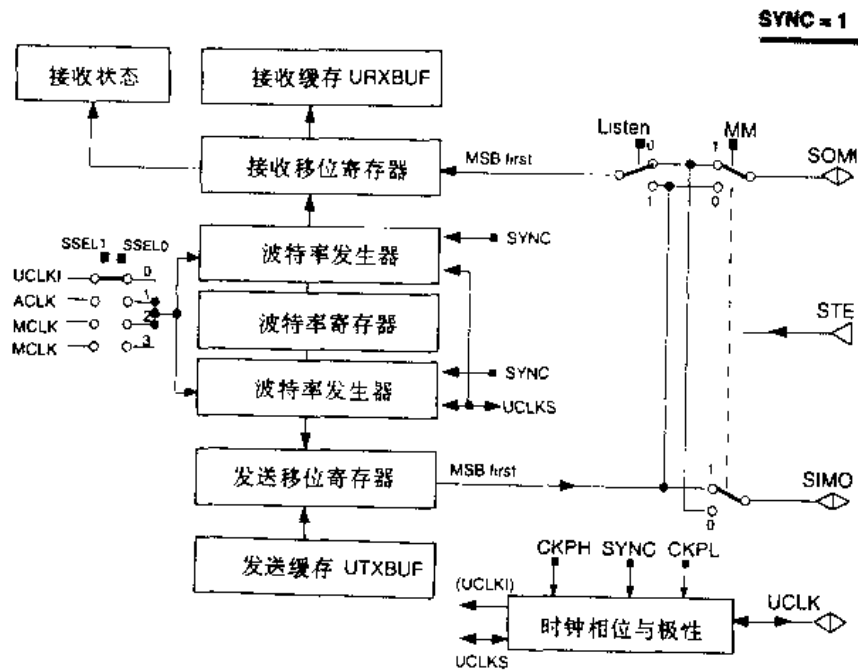


图 13.1 USART 原理框图——SPI 模式

### 13.1 USART 的同步操作

同步模式数据和时钟信号用于发送和接收串行数据。主机提供时钟和数据,从机利用这

一时钟将串行信息移进或移出。4 线 SPI 模式用附加控制线, 来允许从机数据的发送和接收。它由主机控制。

有 3 或 4 个信号用于数据交换, 即

- SIMO: 从进、主出。
- SOMI: 从出、主进。
- UCLK: USART 时钟, 由主机驱动, 从机用它发送和接收数据。
- STE: 从机发送允许, 用于 4 线模式中控制多主从系统中的多个从机。

图 13.2~图 13.4 说明, 当 USART 用同步模式与另一设备的串行通道互联时, 使用一个公共的发送接收移位寄存器。MSP430 可以是主机(见图 13.2)或从机(见图 13.4), 操作是相同的。主机通过发送 UCLK 信号来初始化发送。主机数据在一个时钟沿移出发送移位寄存器, 在另一个反向沿移入接收移位寄存器。从机数据的移位操作与主机的相同, 用公共的移位寄存器接收和发送数据。主机和从机的发送和接收是同时进行的。

由应用软件来决定数据是否有意义, 有以下情况:

- 主机发送数据, 从机发送伪数据;
- 主机发送数据, 从机发送数据;
- 主机发送伪数据, 从机发送数据。

主机可在任意时候初始化发送并控制 UCLK。由软件协议来确定主机知道何时从机广播数据。

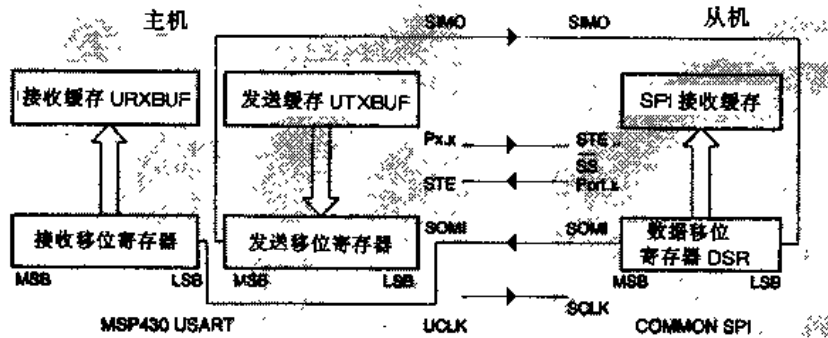


图 13.2 MSP430 USART 作为主机, 具有 SPI 的设备作为从机

图 13.3 是 7 位字长串行同步数据发送的例子。接收移位寄存器的初始内容是“00”。

- A: 从机写 98h 到 DSR, 并等待主机将数据移出;
- B: 主机写 B0h 到 UTXBUF, 它立即送到发送移位寄存器, 且开始发送;
- C: 完成第一个字符, 中断标志置位;
- D: 从机从它的接收缓存中读出 58h(右对齐);
- E: 从机写 54h 到 DSR, 并等待主机将数据移出;
- F: 主机从它的接收缓存 URXBUF 中读出 4Ch(右对齐);
- G: 主机写 E8h 到发送缓存 UTXBUF 中并开始发送;
- H: 完成第二个字符, 中断标志置位;
- I: 主机收到 2Ah, 从机收到 74h(右对齐)。



一旦移位寄存器为空,已写入发送缓存 UTXBUF 的数据移入移位寄存器,并启动在 SIMO 引脚的数据发送,且先发送最高有效位。同时,接收数据移入接收移位寄存器,当移完所选定的位数时,接收移位寄存器中的数据移入 URXBUF,并使中断标志 URXIFG 置位。最高有效位首先移入接收移位寄存器。数据以右对齐的方式存入接收缓存 URXBUF。如果这时前一数据未被读取,则溢出错位 OE 置位。

**注意: USART 同步主模式,接收初始化**

为了接收一个字符,主机必须向发送缓存 UTXBUF 写入数据。当发送移位寄存器为空时,接收开始,数据移入这一移位寄存器。接收和发送总是在时钟脉冲的两个反向沿处一起发生。

用发送中断标志 UTXIFG 或接收中断标志 URXIFG 可以完成协议的控制。数据从移位寄存器发送给从机后,可立即用 UTXIFG 标志将数据从缓存中移入移位寄存器,开始一次发送。从机接收定时应确保能及时获取数据。URXIFG 标志指示数据移出、移入完成的时间。主机可利用 URXIFG 确定从机已准备好接收新的数据。

任一标准的数字端口,包括 STE 在内,用标准数字端口功能可选择一个从机。为了能够访问 SOMI 数据线和接收 UCLK 的时钟信号,从机使用 STE 信号。

在 4 线 SPI 主机模式(SYNC=1、STC=0、MM=1)下,激活的主机用 STE 信号防止与别的主机发生总线冲突。如果相应的 PnSEL 位选择模块功能,则 STE 引脚成为输入。主机在 STE 信号为“高”时正常操作。当 STE 置为“低”时,例如另一设备申请成为主机,这时当前的主机作出如下反应:

- 将驱动 SPI 总线的 SIMO 和 UCLK 的引脚置为输入;
- 出错位 FE 和 URCTL 中的中断标志 URXIFG 置位。

总线冲突消除,即 SIMO 和 UCLK 不再驱动总线线路;同时,出错了标志通知软件,系统完整性被破坏。当 STE 为低电平时,SIMO、UCLK 引脚被强制变为输入;当 STE 返回高电平,系统将返回到由相应控制位定义的状态。

在 3 线模式中,STE 输入信号与主机无关。

### 13.1.2 SPI 模式中的从模式——MM=0、SYNC=1

当选择同步模式并且主机模式位 MM 复位时,微控制器进入从机模式。

为接收外部主机提供的串行移位时钟,UCLK 引脚工作在输入状态。数据传输速率由这个时钟信号而不是内部的比特率发生器决定。在开始 UCLK 之前,由 UTXBUF 装入到移位寄存器的数据,在主机提供的 UCLK 信号作用下,通过 SOMI 引脚发送。同时,将 UCLK 时钟的反向沿 SIMO 引脚上的串行数据移入接收移位寄存器。

接收中断标志 URXIFG 置位,表示数据已被接收并已传送给接收缓存。如果当新数据写入接收缓存时,前一个接收到的数据还未被读取,则溢出标志将置位。

在 4 线 SPI 从机模式(MM=0、SYNC=1、STC=0)下,STE 信号被从机用作发送、接收允许信号。它由主机提供。当 STE 信号为“高”时,禁止接收和发送;当 STE 信号为“低”时,允许接收和发送。无论何时 STE 信号变为“高”,任何已启动的接收操作都将暂停,直到 STE 信号再次变为“低”时再继续。STE 信号用于允许一个从机访问数据线路。当 STE 为“高”时,SOMI 成为输入状态。

## 13.2 中断与控制功能

USART 外围模块有发送和接收两个主中断源。有两个独立的中断矢量,一个对应于接收,一个对应于发送。USART 的控制位位于 SFR 地址范围内。

控制位	缩写	初始状态
接收中断标志	URXIFG	复位(用 PUC/SWRST)
接收中断允许	URXIE	复位(用 PUC/SWRST)
接收允许	URXE	复位(用 PUC)
发送中断标志	UTXIFG	置位(用 PUC/SWRST)
发送中断允许	UTXIE	复位(用 PUC/SWRST)
发送允许	UTXE	复位(用 PUC)

USART 的发送和接收是并行进行的,并且使用同步主机的同一个波特率发生器。在同步从机中,UCLK 引脚上的外部时钟用于接收和发送。

### 13.2.1 USART 接收允许

接收允许位 URXE 允许或禁止接收器从 URXD/SOMI 数据线上进行位流的收集。当禁止 USART 接收( $URXE=0$ )时,在完成已开始的接收操作后停止接收。如果无接收操作发生,则立即停止接收。在同步模式中,这时的 UCLK 不能移动数据到接收移位寄存器中。

#### 1. MSP430 作为主机时的接收

当 MSP430 的 USART 选择为 SPI 主机模式时,它的 3 线和 4 线模式的接收操作是相同的。图 13.5 为 MSP430 为主机时的接收允许 URXE 状态图。

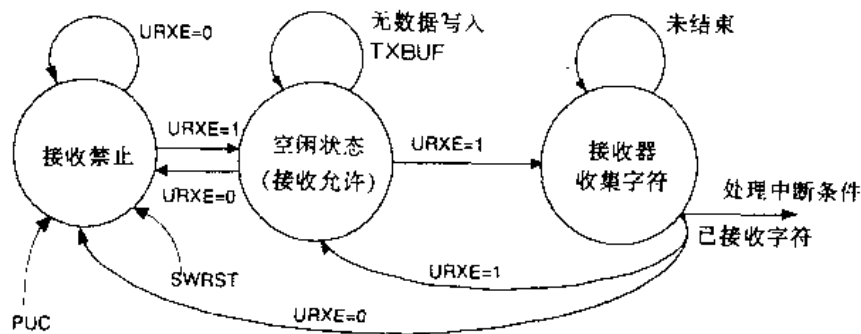


图 13.5 接收允许 URXE 的状态图——MSP430 为主机

#### 2. MSP430 作为从机时的接收——3 线模式。

当 MSP430 的 USART 选择 SPI 的从机模式时,3 线和 4 线模式的接收操作是不同的。图 13.6 为 MSP430 为从机/3 线模式时的接收允许 URXE 状态图。在 3 线模式中,接收操作一旦启动,没有外部 SPI 接收控制信号能使它停止。只有上电清除 PUC、软件复位 SWRST 或接收允许 URXE 能停止接收操作,并使 USART 复位。

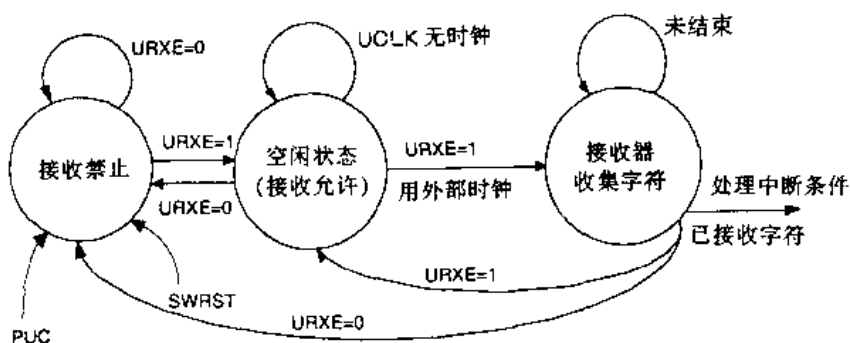


图 13.6 接收允许 URXE 的状态图——MSP430 为从机/3 线模式

**注意：URXE 再允许, SPI 模式**

由于接收器的完全禁止,接收器的再允许与通信线路的数据流是异步的。要实现与数据流的同步,必须像通常的 3 线 SPI 模式那样采用软件协议来实现。

**3. MSP430 作为从机时的接收——4 线模式**

图 13.7 为 MSP430 为从机/4 线模式的接收允许 URXE 状态图。在 4 线模式中,STE 引脚上的外部 SPI 接收控制信号可停止已经启动的接收操作。上电清除 PUC、软件复位 SWRST 或 URXE 也能停止接收操作,并使操作控制状态机复位。无论何时当 STE 为“高”,接收操作都暂停。

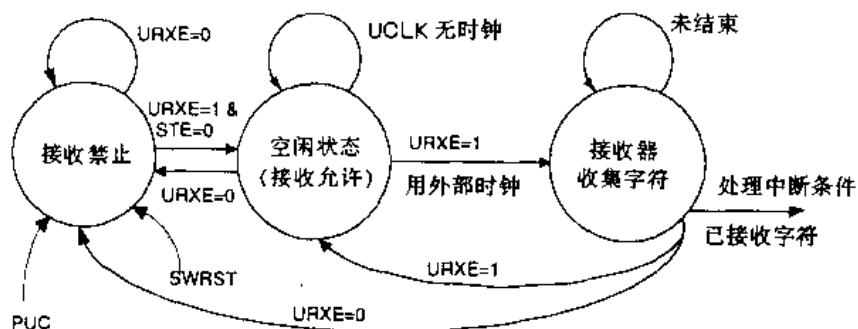


图 13.7 接收允许 URXE 的状态图——MSP430 为从机/4 线模式

**13.2.2 USART 发送允许**

发送允许位 UTXE 允许或禁止字符移位到串行数据线。它的复位将禁止发送器;但是,进行中的发送并不暂停,直到将已写入发送缓存的数据全部发送完毕。继续向发送缓存写入数据不会产生数据发送。如果 UTXBUF 就绪,则发送请求将一直保持。一旦 UTXE 置位并且发送器为空,则立即启动数据发送。STE 的低电平信号将从总线上取消活动的主机(4 线模式)。STE 为“低”表明有另一主机请求成为活动的主机。

**1. USART 发送允许——MSP430 是主机**

MSP430 为主机时的发送允许状态图如图 13.8 所示。







UTXIE 因 PUC 或 SWRST 位而复位。系统复位 PUC 或软件复位 SWRST 使 UTXIFG 置位, 但使 UTXIE 复位, 以保证完整的中断控制能力。

### 13.3 控制与状态寄存器

USART 硬件是字节结构, 因此必须以字节指令访问(后缀“B”)。

寄存器	缩写	寄存器类型	地址	初始状态
USART 控制寄存器	UCTL	读写	070h	见各
发送控制寄存器	UTCTL	读写	071h	位
接收控制寄存器	URCTL	读写	072h	说明
调整控制寄存器	UMCTL	读写	073h	不变
波特率寄存器 0	UBR0	读写	074h	不变
波特率寄存器 1	UBR1	读写	075h	不变
接收缓存	URXBUF	读写	076h	不变
发送缓存	UTXBUF	读	077h	不变

除非在功能描述中说明, 否则, PUC 后所有位是随机的。

由 PUC 或 SWRST 位执行 USART 复位。PUC 后 SWRST 保持置位, USART 维持在复位状态, 直到通过 SWRST 复位来结束 USART 复位。PUC 后禁止 SPI 模式。

USART 模块工作在由 SYNC 位定义的异步模式或同步模式下。控制寄存器的各位在两种模式中功能上有差异。下面各位及其功能的描述都是在同步模式下, 即 SYNC=1。异步模式下的功能在 USART 串行接口 UART 模式部分中描述。

#### 13.3.1 USART 控制寄存器

控制寄存器的信息决定了 USART 模块的基本操作。由寄存器位来选择通信模式和字符位数。在通过 SWRST 复位使 USART 被允许之前, 对各位应根据选择的模式进行编程。控制寄存器 UCTL 的位定义如下:

UCTL (070h)

7							0
未用	未用	未用	CHAR	Listen	SYNC	MM	SWRST
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

对 UCTL 中各位说明如下:

位 0: 如果 SWRST 置位, 则 USART 的状态和运行标志被初始化成复位状态。所有受影响的逻辑保持在复位状态, 直至 SWRST 复位。这意味着一次系统复位后, 只有将 SWRST 复位, USART 才能重新被允许。

位 1: 当 MM 位置位时, 选择主机模式; 当 MM 位复位时, 选择从机模式。

位 2: 选择 USART 外围模块的模式。

SYNC 位选择 USART 外围接口模块的功能。USART 的一些控制位在 UART 模式和 SPI 模式中有不同的功能。

SYNC=0: UART 模式。

SYNC=1: SPI 模式。

位 3: Listen 位选择是否将发送数据内部反馈给接收器。

位 4: 字符长度。

选择字符以 7 或 8 位发送。对于 7 位字符,不用 URXBUF 和 UTXBUF 的最高位,这一位填“0”。

CHAR=0: 7 位。

CHAR=1: 8 位。

位 5~7: 未用。

### 13.3.2 发送控制寄存器 UTCTL

控制寄存器 UTCTL 控制与发送有关的 USART 硬件。UTCTL 的位定义如下:

UTCTL (071h)

7							0
CKPH	CKPL	SSEL1	SSEL0	未用	未用	STC	TXEPT
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

对 UTCTL 中各位说明如下:

位 0: 发送移位寄存器和 UTXBUF 都为空时,发送器空标志 TXEPT 置位,并在数据写入 UTXBUF 时复位。它在发生 SWRST 时置位。

位 1: 从机发送控制位 STC 选择是否将 STE 引脚信号使用在主机和从机中。

STC=0: 选择 SPI 的 4 线模式。STE 信号用于使主机避免总线冲突,或用于在从机模式中控制发送和接收的允许。

STC=1: 选择 SPI 的 3 线模式。STE 在主机、从机模式中不起作用。

位 2,3: 未用。

位 4,5: 时钟源选择 0、1。

只有当选择主机模式时,由时钟源选择位来定义用于波特率发生器的时钟源。

SSEL1、SSEL0 0 选择外部时钟 UCLK。

1 选择辅助时钟 ACLK。

2、3 选择主系统时钟 MCLK。

在主机模式下,不能选 UCLK 上的外部时钟,因为主机经 UCLK 将时钟提供给所有从机。

在从机模式下,SSEL1、SSEL0 是无关紧要的。这时,总是用外部 UCLK 信号作为它的时钟。

位 6,7: 时钟极性 CKPL 和时钟相位 CKPH(见图 13.13)。

CKPL 位控制 SPICLK 信号的极性。

CKPL=0: 低电平为无效电平。数据在 UCLK 的上升沿输出。输入数据在下降沿锁存。

CKPL=1: 高电平为无效电平。数据在 UCLK 的下降沿输出。输入数据在上

升沿锁存。

CKPH 位控制 SPICLK 信号的相位。

CKPH=0: 正常的 UCLK 时钟。

CKPH=1: UCLK 被延迟半个周期。

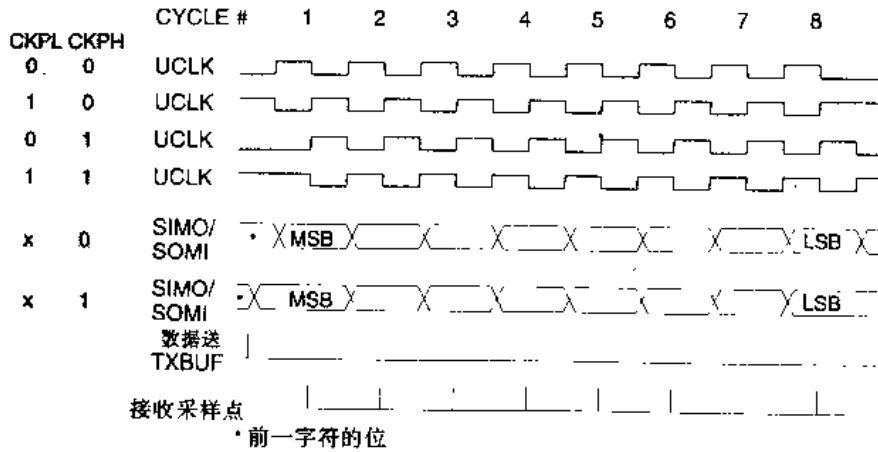


图 13.13 USART 的时钟相位和极性

当工作在 CKPH 置位时,同步模式 USART 在发送移位寄存器中装入数据,并在 UCLK 的第一个沿之前准备好数据的第一位。数据在 UCLK 的第一个沿锁存,在第二个沿发送。

### 13.3.3 接收控制寄存器 URCTL

URCTL 控制与接收相关的 USART 硬件,并保存出错信息。URCTL 的位定义如下:

URCTL (072h)

7							0
FE	未定义	OE	未定义	未用	未用	未定义	未定义
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

对 URCTL 中各位说明如下:

位 0: 未定义,由 USART 硬件驱动。

位 1: 未定义,由 USART 硬件驱动。

位 2: 未用。

位 3: 未用。

位 4: 未定义,由 USART 硬件驱动。

位 5: 当一个字符写入 URXBUF 时前一字符还未被读出,则溢出标志 OE 置位。这时,前一字符被覆盖而丢失。OE 通过用 SWRST 位、系统复位、读取 URXBUF 和用指令来复位。

位 6: 未定义,由 USART 硬件驱动。

位 7: 帧错误。仅仅当选择 4 线模式时,因总线冲突使有效主机停止并在 STE 引脚信号出现负跳变而导致 FE 位置位。FE 位通过用 SWRST 位、系统复位、读 URXBUF 和用指令来复位。

### 13.3.4 波特率选择和调制控制寄存器

波特率发生器用波特率选择寄存器的 UBR1 和 UBR0 来产生串行数据流的位定时。最小的分频因子为 2。UBR0 和 UBR1 的结构如下：

UBR0 (074h)

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

UBR1 (075h)

7							0
$2^{15}$	$2^{14}$	$2^{13}$	$2^{12}$	$2^{11}$	$2^{10}$	$2^9$	$2^8$
rw	rw	rw	rw	rw	rw	rw	rw

波特率的表达式如下：

$$\text{波特率} = f_{\text{BRCLK}} / (\text{UBR} + (m_7 + m_6 + \dots + m_0) / 8), \quad \text{UBR} = [\text{UBR1}, \text{UBR0}]$$

在上机模式下, 可选择的最大发送波特率为波特率发生器输入时钟频率的一半; 在从机模式下, 取决于在 UCLK 上提供的外部时钟。

在串性同步通信时不需要用调整控制寄存器。建议将它复位(各位置为“0”)。调整控制寄存器 UMCTL 的位定义如下：

UMCTL (073h)

7							0
m7	m6	m5	m4	m3	m2	m1	m0
rw	rw	rw	rw	rw	rw	rw	rw

### 13.3.5 USART 接收数据缓存 URXBUF

接收缓存 URXBUF 含有此前从接收移位寄存器传来的数据。由 SWRST 位或 PUC 清除。读取 URXBUF 中数据, 将使接收出错位和中断标志 URXIFG 复位。URXBUF 结构如下：

URXBUF (076h)

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

在 7 位字长模式下, URXBUF 的最高位总是为“0”。

### 13.3.6 USART 发送数据缓存 UTXBUF

发送缓存含有当前需要发送器发送的数据。USART 发送缓存 UTXBUF 结构如下：

UTXBUF (077h)

7							0
$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UTXIFG 标志表示 UTXBUF 已准备好接收下一个要发送的字符。

在主机模式下, 将数据写入 UTXBUF 则初始化发送功能。如果发送移位寄存器为空或即将为空, 则数据的发送立即开始。

当选择每个字符 7 位时, 移入发送缓存的数据必须左对齐, 因为最高有效位首先发送。

## 第 14 章 液晶显示驱动

### 14.1 LCD 驱动基本原理

液晶显示是通过环境光来显示信息的。其本身并不主动发光,因此功耗很低,只要求液晶周围有足够的光强。

液晶必须由交流电压驱动。直流驱动会损坏液晶。交流驱动是其功耗的主要因素。从驱动电路来看,液晶等效为一个电容。它的一个电极叫背极(back plane)或公共极(common plane),由 COM<sub>n</sub> 信号驱动;另一电极叫段极,由 SEG<sub>n</sub> 信号驱动。交流驱动的频率很低,在 30~1 000 Hz 范围内。LCD 生产厂家的数据手册会给出确切的 LCD 频率范围。

目前已经开发了多种驱动液晶显示方法。当要显示的段数、所需的引脚数、显示的对亮度、温度范围等要求不同时,采用的方法也不同。

多路寻址方法能有效地减少所需的引脚。

MSP430 系列的 LCD 模块支持的驱动方法有如下 4 种:

- 静态驱动;
- 2MUX 或 1/2 占空比,1/2 偏压;
- 3MUX 或 1/3 占空比,1/3 偏压;
- 4MUX 或 1/4 占空比,1/3 偏压。

静态方法只需一个引脚作为公共极(COM0),且每 1 段都需要一个引脚,即

$$\text{引脚数} = 1 + \text{段数}$$

2MUX 方法需要 2 个引脚作为公共极(COM0、COM1),每 2 段需要一个引脚,即

$$\text{引脚数} = 2 + \text{段数}/2$$

3MUX 方法需要 3 个引脚作为公共极(COM0、COM1、COM2),每 3 段需要一个引脚,即

$$\text{引脚数} = 3 + \text{段数}/3$$

4MUX 方法需要 4 个引脚作为公共极(COM0、COM1、COM2、COM3),每 4 段需要一个引脚,即

$$\text{引脚数} = 4 + \text{段数}/4$$

增加多路转接数能减少引脚数。

例如显示 80 段,各方法所需的引脚数如下:

- 静态驱动: 引脚数 = 1 + 80 = 81。
- 2MUX: 引脚数 = 2 + 80/2 = 42。
- 3MUX: 引脚数 = 3 + 80/3 = 30。
- 4MUX: 引脚数 = 4 + 80/4 = 24。

### 1. 静态驱动方法

对于静态驱动方法,每根 SEG 线(段线)驱动 1 段。

图 14.1 说明用该驱动方法,LCD 显示一位“5”时,各引脚的连接和驱动电压波形。

### 2. 2MUX——1/2 偏压

对于 2MUX 驱动方法,每根段线驱动 2 段。

图 14.2 说明用该驱动方法,LCD 显示一位“5”时,各引脚的连接和驱动电压波形。

### 3. 3MUX——1/3 偏压

对于 3MUX 驱动方法,每根段线驱动 3 段。

图 14.3 说明用该驱动方法,LCD 显示一位“5”时,各引脚的连接和驱动电压波形。

### 4. 4MUX——1/3 偏压

对于 4MUX 驱动方法,每根段线驱动 4 段。

图 14.4 说明用该驱动方法,LCD 显示一位“5”时,各引脚的连接和驱动电压波形。

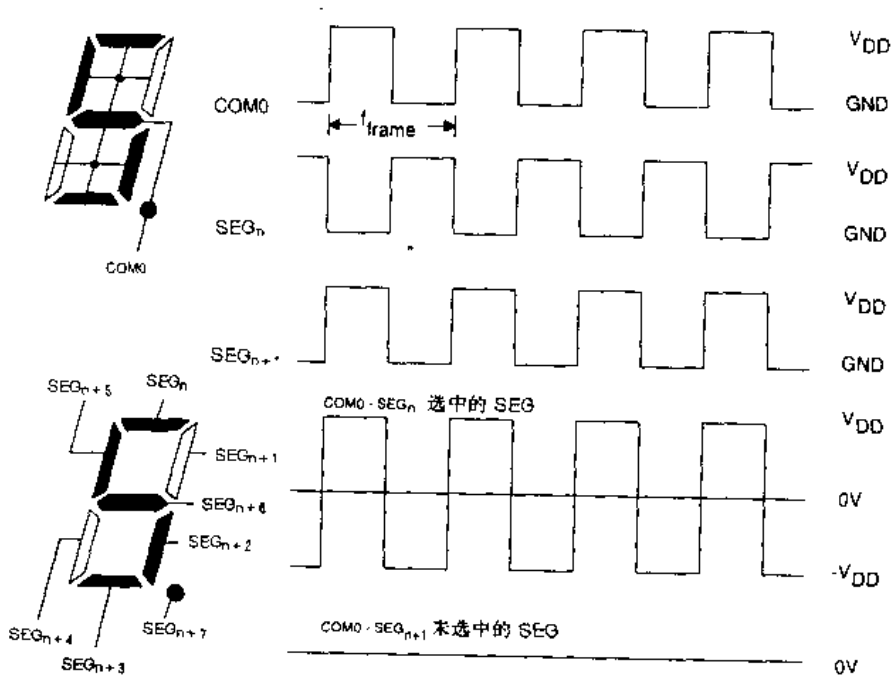


图 14.1 静态驱动波形实例



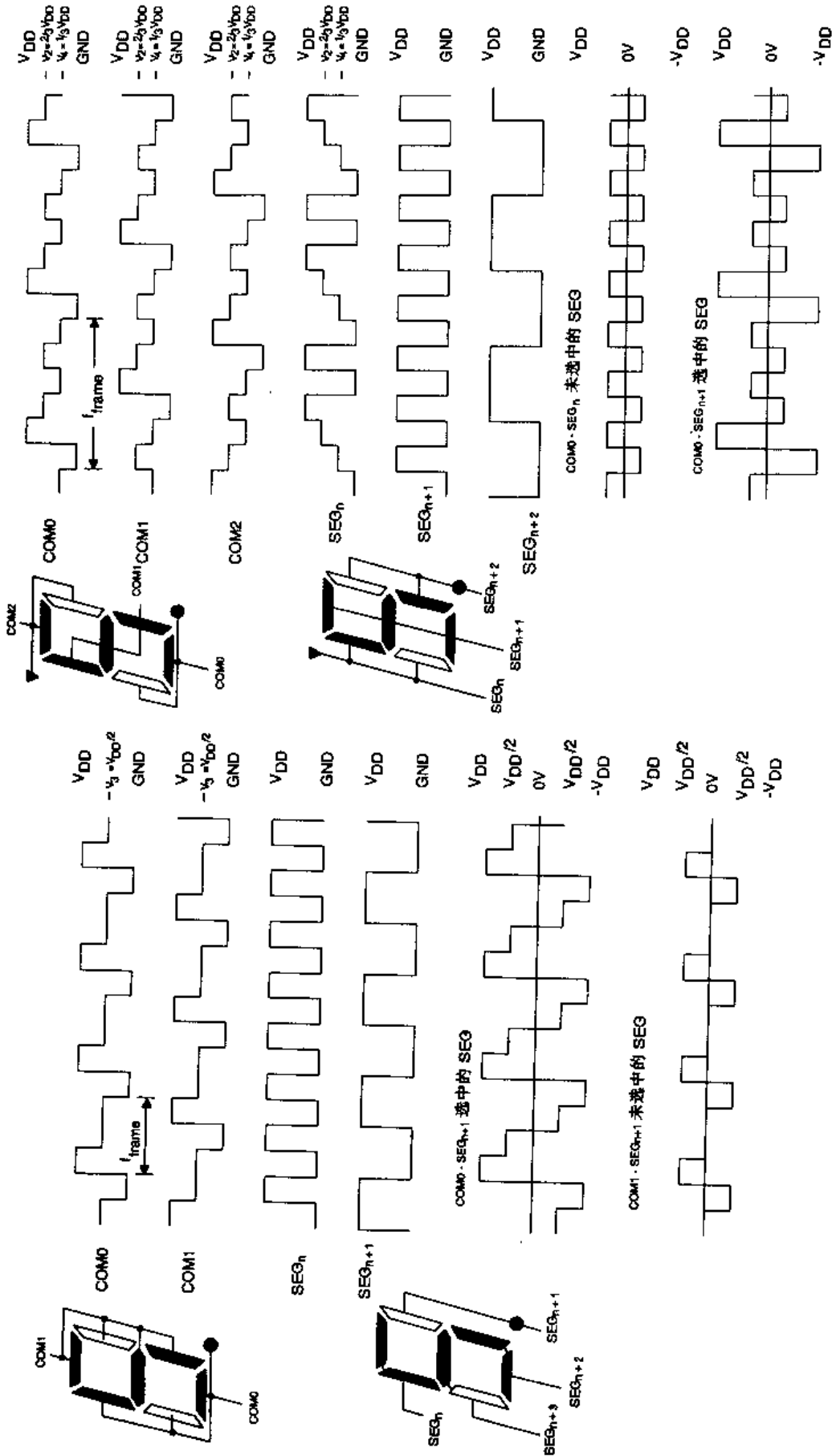


图 14.3 3MUX 驱动波形实例

图 14.2 2MUX 驱动波形实例



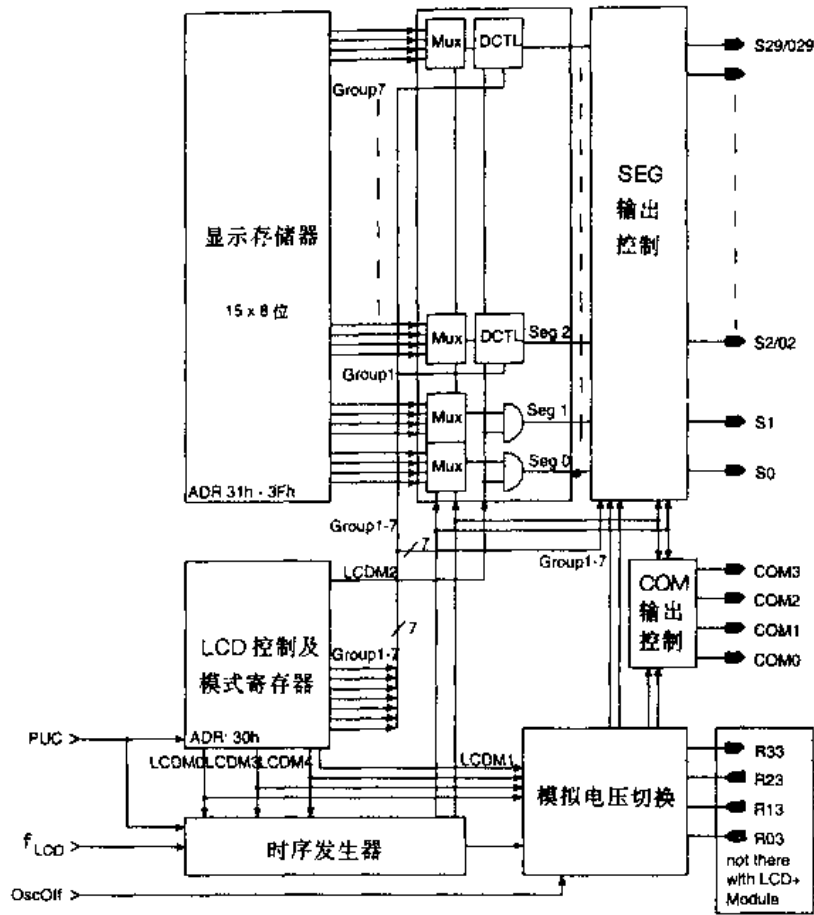


图 14.5 LCD 控制器/驱动器功能框图

表 14.1 LCD 模块和 LCD+ 模块的区别

	LCD 模块	LCD+ 模块
模拟电压	由外部产生 选项： · 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：R33、R23、R13、R03	由内部产生
控制位 LDCM1	未用	选择电阻网络阻抗
控制位 LDCM0	停止定时发生器	· 停止定时发生器 · 停止电阻网络电流

### 14.2.1 LCD 控制器/驱动器功能

LCD 驱动器的功能如下：

- 自动从显示内存读取数据，并产生 SEG、COM 信号。
- 可选择 4 种不同的驱动模式，即
  - 静态模式；
  - 2MUX, 1/2 偏压；

- 3MUX, 1/3 偏压;
- 4MUX, 1/3 偏压。

BT(Basic Timer)中,有两位用于选择 4 种不同的扫描频率。

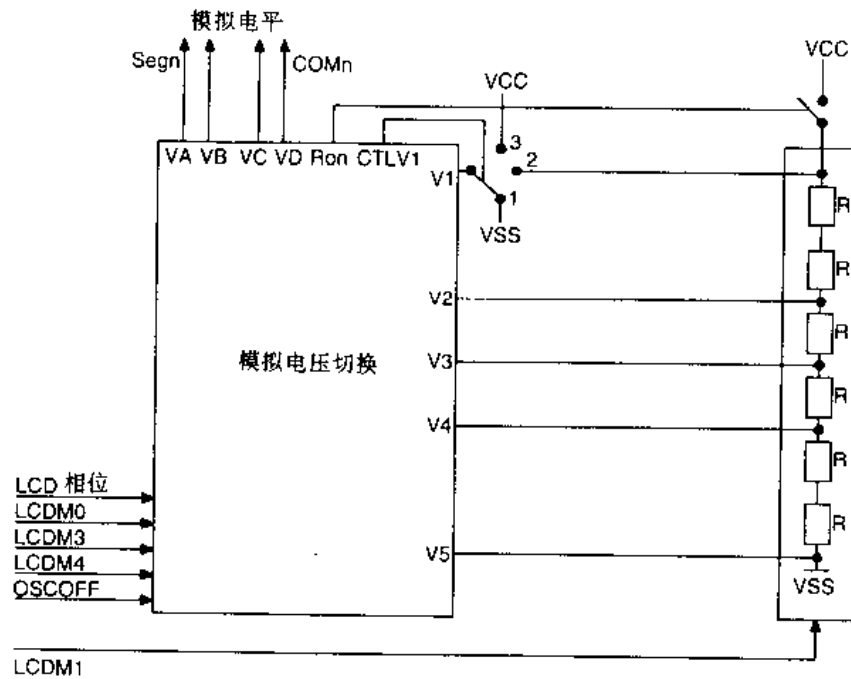
- 段线输出端口可切换成一般的输出端口。
- 没有存储段信息的显示内存可用做普通内存。
- 用辅助时钟(ACLK)经 BT 来操作。
- 仅针对 LCD+ 模块的功能:
  - 用电阻网络为 LCD 驱动提供各种模拟电压;
  - 由控制寄存器 LCDCTL 中的一个控制位来控制电阻网络和 V1 的通断。

LCD 驱动线的帧频  $f$  如下:

- 静态方法  $f = 1/2 \times f_{LCD}$ ;
- 2MUX  $f = 1/4 \times f_{LCD}$ ;
- 3MUX  $f = 1/6 \times f_{LCD}$ ;
- 4MUX  $f = 1/8 \times f_{LCD}$ 。

### 1. LCD+ 模块

LCD+ 模块由内部发生模拟电压,如图 14.6 所示。



OSCOFF	LCDM4	LCDM3	LCDM0	VA	VB	VC	VD	RON	CTLV1
X	X	X	0	0	0	0	0	OFF	1*
1	X	X	X	0	0	0	0	OFF	1*
0	0	0	1	V5/V1	V1/V5	V5/V1	V1/V5	OFF	3*
0	0	1	1	V5/V1	V1/V5	V3/V3	V1/V5	ON	2*
0	1	X	1	V5/V1	V2/V4	V4/V2	V1/V5	ON	2*

\* 到开关位置

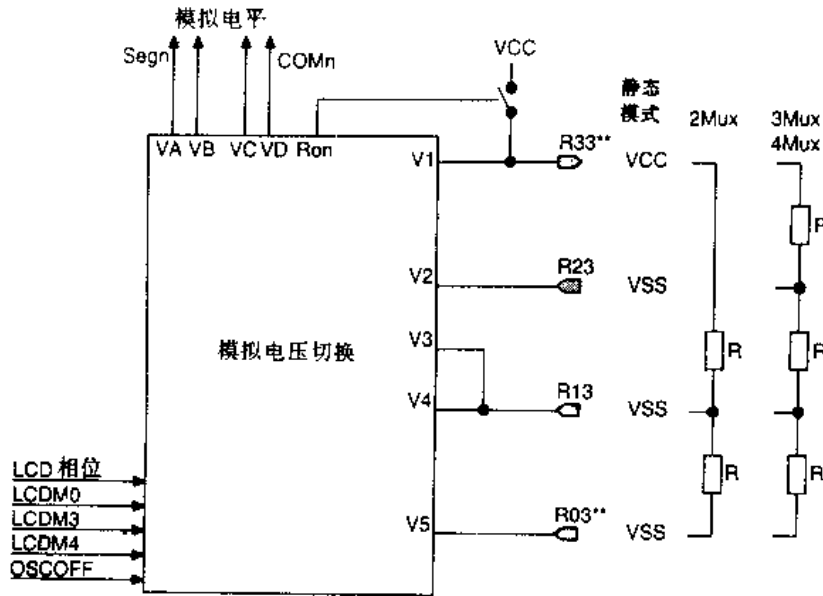
图 14.6 LCD+ 模块的内部模拟电压发生原理图

当状态寄存器的 OscOff 置位时,电阻网络电源切断,并与 LCDM0 无关。

在静态模式下,由于只用 V1 和 V5,模拟电压发生器关闭。这降低了电流消耗。

### 2. LCD 模块

LCD 模块由外部提供模拟电压,加在 R33、R23、R13、R03 引脚上,如图 14.7 所示。其中 R33 和 R03 引脚的电压可选。请阅器件数据手册。



OSCOFF	LCDM4	LCDM3	LCDM0	VA	VB	VC	VD	RON *
X	X	X	0	0	0	0	0	OFF
1	X	X	X	0	0	0	0	OFF
0	0	0	1	V5/V1	V1/V5	V5/V1	V1/V5	OFF
0	0	1	1	V5/V1	V1/V5	V3/V3	V1/V5	ON
0	1	X	1	V5/V1	V2/V4	V4/V2	V1/V5	ON

\* 到开关位置

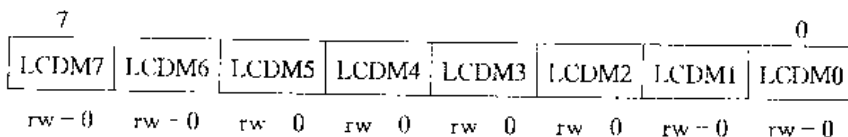
\*\* 电源引脚 V1、V5 可选。不用 R33、R03 的器件将 V1 接 VCC, V5 接 VSS。外部电阻网络应接 VCC 和 VSS。

图 14.7 LCD 模块的外部模拟电压

### 14.2.2 LCD 控制与模式寄存器

LCD 控制与模式寄存器定义各种操作条件。LCD 模块是字节结构,必须用字节指令访问。LCD 控制与模式寄存器的位定义如下:

LCDCTL (030h)



对 LCDCTL 中各位说明如下:

LCDM0: LCDM0=0: 定时发生器关闭。COM 线(公共线)和 SEG 线(段线)输出为“低”。

选做输出端口线的输出不受影响。

对于 LCD+ 模块, 切断电阻网络的电源。

LCDM0 = 1: COM 线、SEG 线按显示内存的数据输出信号。

选做输出端口线的输出不受影响。

对于 LCD+ 模块, 对电阻网络供电只适用于 2MUX、3MUX、4MUX。

LCDM1: 通过选择模拟电压发生器的内阻来选择 LCD 驱动电压幅度。这只对 LCD+ 模块有效。

LCDM1 = 0: 模拟发生器的内阻为高阻抗。

LCDM1 = 1: 模拟发生器的内阻为低阻抗。

LCDM2、3、4: 选择显示模式, 并可将来段输出切换到不选电平, 如表 14.2 所列。

LCDM2 禁止或允许段输出。它位于显存输出与段输出之间的并串转换电路中, 通过和段信息的“与”来实现。保存在显存内的段信息不变。这一功能的主要目的是支持动态显示。上面表中 R33 与 R03 是可选择引脚。

LCDM5、6、7: 选择输出段信息或端口信息的组合(见表 14.3)。选为端口功能的输出由显存各位驱动, 并不再作为 LCD 段线。

表 14.2 显示模式与偏压选择

LCDM4	LCDM3	LCDM2	显示模式	LCD+ 偏压	LCD 偏压
x	x	0	不影响, 显示关闭, 所有段信号是不选电平。端口输出保持稳定		
0	0	1	静态	1/1	R33*, R03*
0	1	1	2MUX	1/2	R33*, R13, R03*
1	0	1	3MUX	1/3	R33*, R23, R13, R03*
1	1	1	4MUX	1/3	R33*, R23, R13, R03*

\* 可选引脚。

表 14.3 段线与端口输出线的分组

LCDM7	LCDM6	LCDM5	0 组	1 组	2 组	3 组	4 组	5 组	6 组	7 组
0	0	0	S0 ~ S1	O2 ~ O5	O6 ~ O9	O10 ~ O13	O14 ~ O17	O18 ~ O21	O22 ~ O25	O26 ~ O29
0	0	1	S0 ~ S1	S2 ~ S5	O6 ~ O9	O10 ~ O13	O14 ~ O17	O18 ~ O21	O22 ~ O25	O26 ~ O29
0	1	0	S0 ~ S1	S2 ~ S5	S6 ~ S9	O10 ~ O13	O14 ~ O17	O18 ~ O21	O22 ~ O25	O26 ~ O29
0	1	1	S0 ~ S1	S2 ~ S5	S6 ~ S9	S10 ~ S13	O14 ~ O17	O18 ~ O21	O22 ~ O25	O26 ~ O29
1	0	0	S0 ~ S1	S2 ~ S5	S6 ~ S9	S10 ~ S13	S14 ~ S17	O18 ~ O21	O22 ~ O25	O26 ~ O29
1	0	1	S0 ~ S1	S2 ~ S5	S6 ~ S9	S10 ~ S13	S14 ~ S17	S18 ~ S21	O22 ~ O25	O22 ~ O25
1	1	0	S0 ~ S1	S2 ~ S5	S6 ~ S9	S10 ~ S13	S14 ~ S17	S18 ~ S21	S22 ~ S25	O26 ~ O29
1	1	1	S0 ~ S1	S2 ~ S5	S6 ~ S9	S10 ~ S13	S14 ~ S17	S18 ~ S21	S22 ~ S25	S26 ~ S29

注意: LCD 控制位

LCDM5、LCDM6、LCDM7 在发生 PUC 时复位。



COM2: 位 2 对应  $S_n$ , 位 6 对应  $S_{n+1}$ 。

• 4MUX 驱动: COM0: 位 0 对应  $S_n$ , 位 4 对应  $S_{n+1}$ 。

COM1: 位 1 对应  $S_n$ , 位 5 对应  $S_{n+1}$ 。

COM2: 位 2 对应  $S_n$ , 位 6 对应  $S_{n+1}$ 。

COM3: 位 3 对应  $S_n$ , 位 7 对应  $S_{n+1}$ 。

1. 静态驱动方法的显存

图 14.10 为用静态方法驱动时显存使用示意图。静态驱动方法只使用一条 COM 线, 即 COM0。用位 0 和位 4 来存储段信息, 其余各位可用做普通内存。可显示的段数最多为 30。

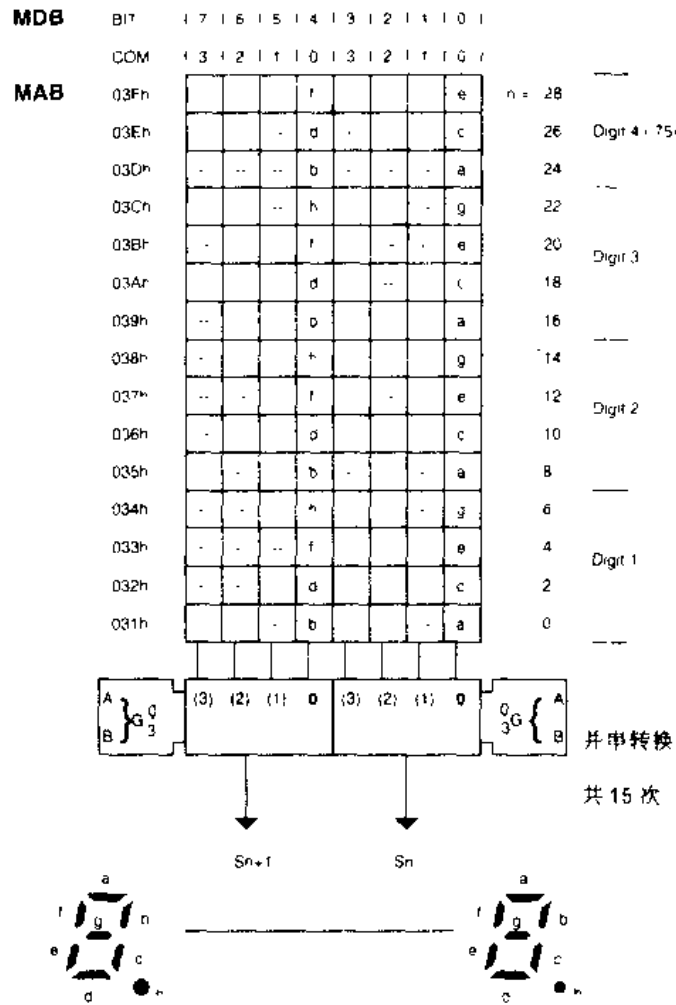


图 14.10 静态驱动时显存的使用

2. 2MUX——1/2 偏压驱动方法的显存

图 14.11 为用 2MUX 方法驱动时显存使用示意图。2MUX 方法使用 2 条 COM 线, 即 COM0、COM1。由位 0、1、4 和 5 存储段信息, 其余各位可用作普通内存。可显示的段数最多为 60。

3. 3MUX——1/3 偏压驱动方法的显存

图 14.12 为用 3MUX 方法驱动时显存使用示意图。3MUX 方法使用 3 条 COM 线, 即 COM0、COM1、COM2。由位 0、1、2、4、5 和 6 存储段信息, 其余各位可用做普通内存。可



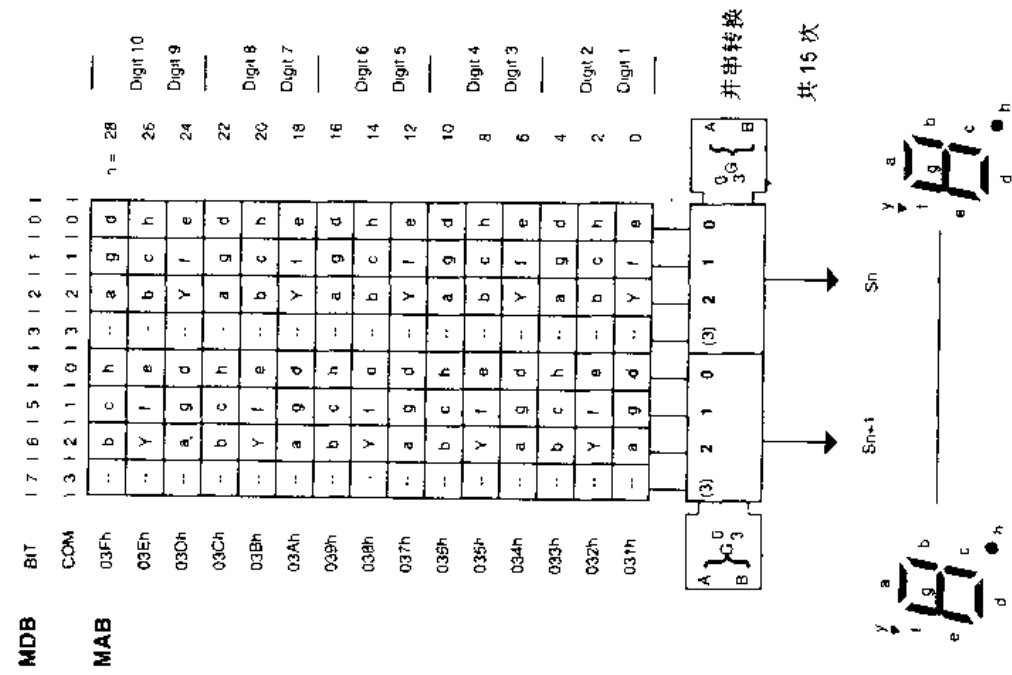


图 14.12 3MI.X 方法驱动时显存的使用

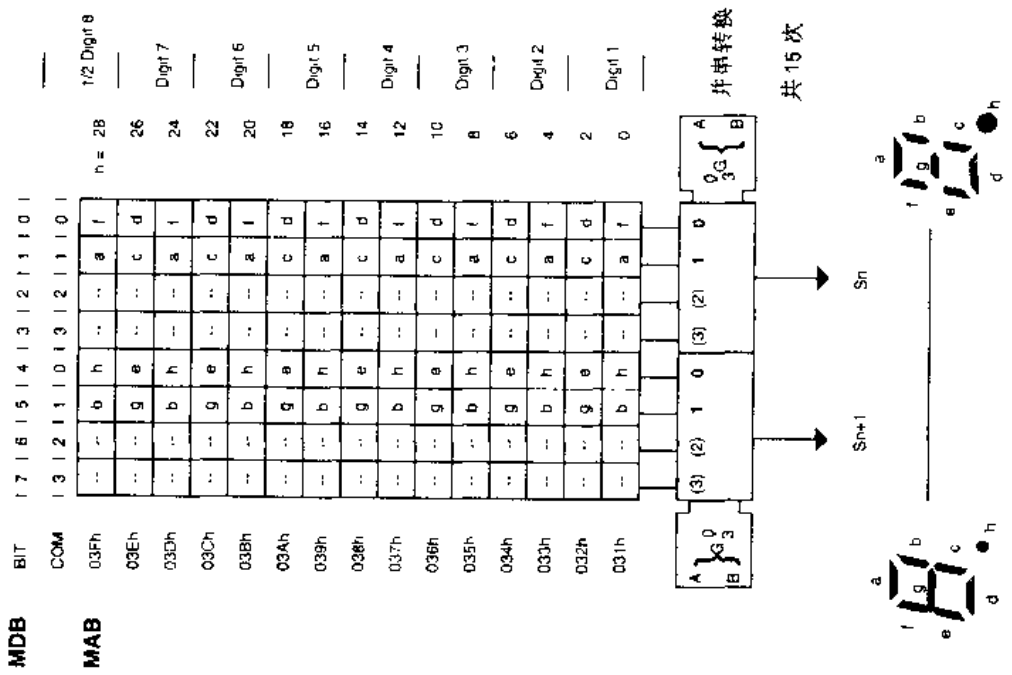


图 14.11 2MI.X 方法驱动时显存的使用

显示的段数最多为 90。

4. 4MUX——1/3 偏压驱动方法的显存

图 14.13 为用 4MUX 方法驱动时显存使用示意图。4MUX 方法使用 4 条 COM 线,即 COM0、COM1、COM2 和 COM3。由位 0、1、2、3、4、5、6 和 7 存储段信息。可显示的段数最多为 120。

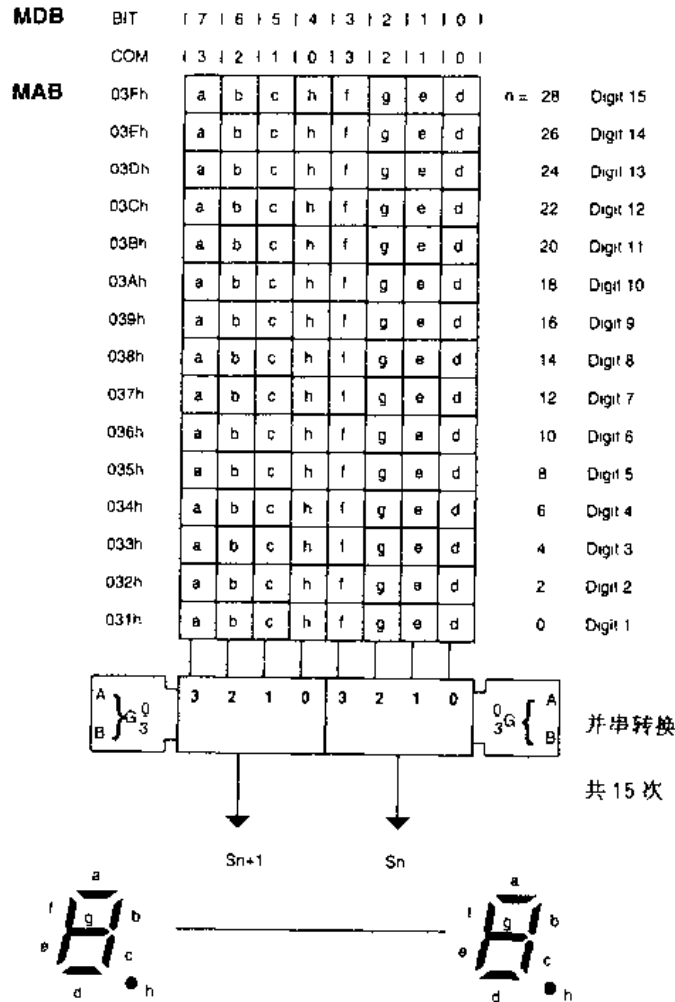


图 14.13 4MUX 方法驱动时显存的使用

14.2.4 LCD 操作软件例程

这一节中提供了 LCD 显示数字的例程。显示数字由标准用法的 7 段组成。

1. 应用 4MUX——1/3 偏压的 LCD 例程

```

sect "led4mux", 0F000h
; 4MUX 模式是最简单的显示方式。每字的全部 8 段在同一显存字节中
;
a EQU 080h
b EQU 040h
    
```

```

c      .EQU      020h
d      .EQU      001h
e      .EQU      002h
f      .EQU      008h
g      .EQU      004h
h      .EQU      010h
;
; 显示寄存器 Rx(000m)表示的字
; 表中为 Rx 表示的字的各显示段
;
...
LCD1   .EQU      00031h      ; LCD 存储器地址
...
...
LCD15  .EQU      0003Fh
;
...
...
MOV.B  Table(Rx), &LCDn    ; n = 1, ..., 15
...                               ; 全部 8 段写入显存
...
;
Table  .BYTE     a+b+c+d+e+f  ; 显示“0”
       .BYTE     b+c          ; 显示“1”
       ...
       ...
       .BYTE     b+c+d+e+g    ; 显示“d”
       .BYTE     a+d+e+f+g    ; 显示“E”
       .BYTE     a+c+f+g      ; 显示“F”

```

## 2. 应用 3MUX——1/3 偏压的 LCD 例程

```

.sect   "lcd3mux", 0F000h
; 3MUX 模式支持每字 9 段而非 8 段, 每字的 9 段安排在 1.5 个显存字节中
;
a      .EQU      0040h
b      .EQU      0400h
c      .EQU      0200h
d      .EQU      0010h
e      .EQU      0001h
f      .EQU      0002h
g      .EQU      0020h
h      .EQU      0100h

```



```

d          EQU    004h
e          .EQU   040h
f          .EQU   001h
g          .EQU   080h
h          .EQU   010h
; 显示寄存器 Rx(000m)表示的字
; 表中为 Rx 表示的字的各显示段
...
LCD1      .EQU   00031h
...
...
LCD15     .EQU   0003Fh
...
...
MOV.B     Table(Rx), Ry          ; 段信息装入暂存
MOV.B     Ry, &LCD n           ; (Ry) = 0000 0000 gebh cdaf
; 注意:
; LCD 显存字节各位全部写入
RRA       Ry                   ; (Ry) = 0000 0000 0geb hcda
RRA       Ry                   ; (Ry) = 0000 0000 00ge bhcd
MOV.B     Ry, &LCD n+1         ; 注意:
; LCD 显存字节各位全部写入
...
...
;
Table     .BYTE   a+b+c+d+e+f    ; 显示“0”
...
        .BYTE   a+b+c+d+e+f+g+h ; 显示“8”
...
...
        .BYTE
...
;

```

#### 4. 应用静态驱动的 LCD 例程

```

        .sect     "led1mux", 0F000h
; 静态模式中, 每个字的 8 段安排在 4 个显存字节中
;
a          .EQU   001h
b          .EQU   010h
c          .EQU   002h
d          .EQU   020h
e          .EQU   004h

```

```

f          .EQU      040h
g          .EQU      008h
h          .EQU      080h
; 显示寄存器 Rx(000m)表示的字
; 表中为 Rx 表示的字的各显示段
;
...
LCD1      EQU      00031h
...
LCD15     .EQU      0003Fh
...
MOV.B     Table(Rx), Ry          ; 段信息装入暂存
; (Ry) = 0000 0000 hfdb geca
MOV.B     Ry, &LCDn             ; 注意:
; LCD 显存字节各位全部写入
RRA      Ry                    ; (Ry) = 0000 0000 0bfd bgec
MOV.B     Ry, &LCDn+1           ; 注意:
; LCD 显存字节各位全部写入
RRA      Ry                    ; (Ry) = 0000 0000 00hf dbge
MOV.B     Ry, &LCDn+2           ; 注意:
; LCD 显存字节各位全部写入
RRA      Ry                    ; (Ry) = 0000 0000 000h fdhg
MOV.B     Ry, &LCDn+3           ; 注意:
; LCD 显存字节各位全部写入
...
Table     .BYTE      a + b + c + d + e + f      ; 显示“0”
          .BYTE      b + c                    ; 显示“1”
          ...
          .BYTE
          ...

```

### 14.3 LCD 端口功能

LCD 的 COM 线和 SEG 线数量极大,封装后占用的引脚也多,这会限制集成度。为了充分利用段线,LCDM5~LCDM7 可选择 4 根段线一组的功能是作为段线还是作为多用途的一般输出端口(见表 14.3)。显存的位信息定义了信号线的逻辑状态。输出为数字信号,接地或接 VCC。

图 14.14 为段线或端口输出线示意图。

段线信号 S<sub>xx</sub> 是 LCD 显示驱动信号的一部分,且按 COM<sub>n</sub> 线的帧时序输出调制电压。作为输出端口的输出信号 O<sub>xx</sub> 是静态的。按显存的位提供数字电平。位 0~3 对应奇数

段线( $n=3,5,\dots$ ), 位 4~7 对应偶数段线( $n=2,4,\dots$ )。实际使用的位取决于驱动方法。

输出信号  $O_{xx}$  的逻辑状态取决于显存各位的状态。根据奇数或偶数线的选择, 对应于位 0~3 或位 4~7。

- $xx=2,4,\dots,28$ :  $O_{xx}$  由位 0~3 定义。
- $xx=3,5,\dots,29$ :  $O_{xx}$  由位 4~7 定义。

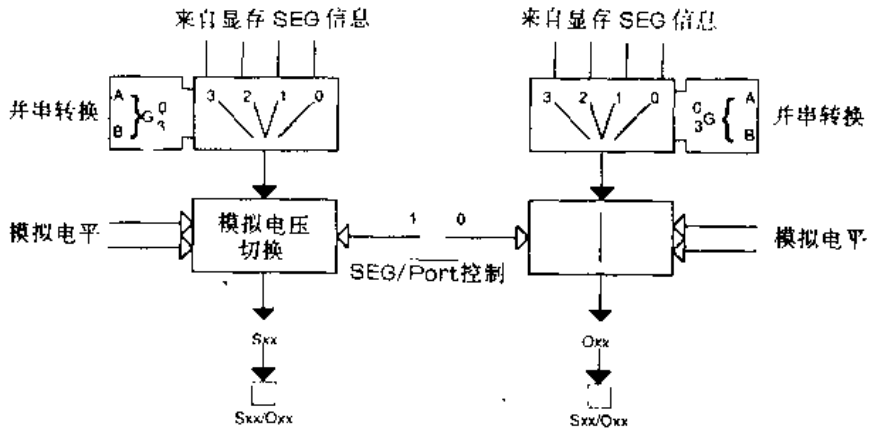


图 14.14 段线或端口输出线

### 14.4 LCD 与端口模式混合应用实例

图 14.15 的例子使用了混合模式: 用 4MUX 方法驱动 LCD 显示 13 位数字, 同时使用一个端口组, 输出 4 位数字信号。

**注意: LCD 端口输出**

任一 LCD 端口输出由 4 位定义。一组的 4 位应该是同一电平, 否则输出是不稳定的。假如 O28 要输出高电平, 位 0~3 都应为高。

下面为软件举例: 设置 O28、O29 不变。

```
LCD15    .EUQ    0003Fh
          BIS.B  00Fh, &LCD15
```

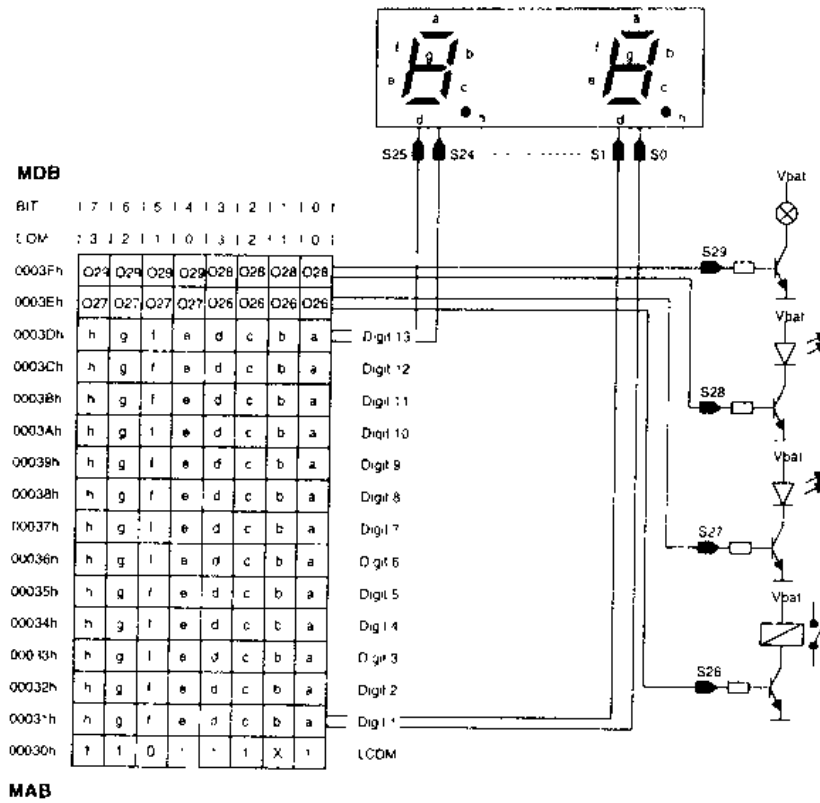


图 14.15 应用实例



## 第 15 章 A/D 转换器

A/D 模块特性如下:

- 8 个 A/D 输入通道;
- 4 个模拟输入端可作为可编程电流源(经外接电阻 REXT);
- 可作比例测量或绝对值测量;
- 内置的采样/保持电路;
- 有转换结束的中断标志(EOC);
- ADAT 寄存器可将转换结果保存到下一次转换的开始;
- 低功耗;
- 独立完成转换,不需要 CPU 额外的处理开销;
- 可编程为 12 或 14 位的分辨率;
- 4 个可编程量程可达到 14 位的动态范围;
- 快速的转换时间;
- 大的供电电压范围;
- 可覆盖整个 A/D 转换范围。

### 15.1 概 述

(12+2)位 A/D 转换器是字访问的外围模块。通过读取 ADAT 寄存器,转换结果出现在 16 位总线上。必须注意,当转换开始后,逐位逼近寄存器(SAR)上出现转换的各位。它们立即进入 ADAT 寄存器,并且直到通过对 ACTL 寄存器中的开始转换位(SOC)置位,初始化转换过程才被清除。由于 SAR 对 MDB 是透明的,转换进程可以通过对只读的 ADAT 寄存器读取来监视。SOC 置位为新数据清除 SAR 寄存器,同时为下一次转换启动 A/D 转换器时钟。

图 15.1 为 ADC(Analog-to-Digital Converter)模块结构示意图。

模块有 8 个可选模拟输入通道,经多路切换进入转换器的输入电路,这样可以在任意时刻对任一通道进行转换。其中的 A0、A1、A2、A3 4 个通道还可作为 4 个电流源输出,其值可以经外接电阻 REXT 设定。为了实现比例测量,这些电流源可以开通输出(一次一个)以驱动外接传感器。将外部稳压源接于 SVCC 引脚可进行精密的绝对测量。

8 个模拟通道也可作为数字输入。对 AEN 寄存器的相应位编程可以使数字信号在全部的 8 个通道或者所选择的通道上输入。通道上的数据可以从 AIN 寄存器上读出。当模拟信号正在转换时,邻近通道的数字信号变化会引起交互干扰,导致噪声和错码输出。

根据 ACTL 寄存器中的位 11 的状态,A/D 转换器有两种工作模式,即 12 位或者(12+2)位。当输入信号的电压范围已知时,将位 11 复位后,ACTL 寄存器中的两位可以定义所要求的转换电压范围。A/D 转换器会在 4 个电压范围之一对输入信号采样,然后转换成 12 位分辨率的数据。在这样的非自动模式下,实际上也使转换器的工作产生 14 位的动态范围。而在

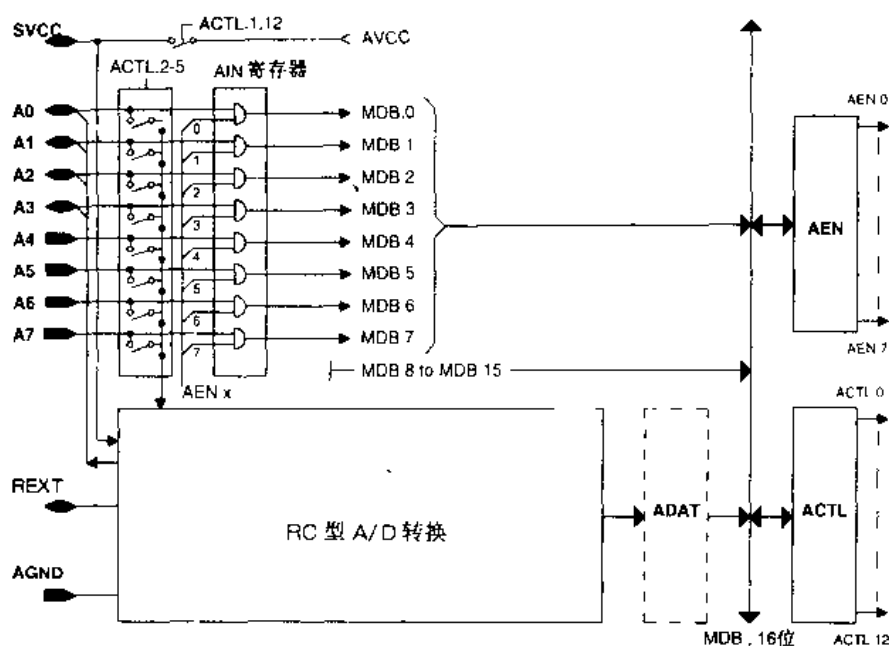


图 15.1 ADC 模块结构图

对位 11 置位所选择的自动模式下,电压范围是由转换器自动选择来达到 14 位效果的。输入信号采样两次,一次是确定选择电压范围的高 2 位,后一次是为作转换确定余下的 12 位,以获得(12+2)位的结果。在这两种模式中,当转换结束(EOC)时,中断标志会自动设置,以通知处理机一次转换已经完成。EOC 信号同时关闭 A/D 时钟直至下一次 SOC 位置位,以减少功耗。

**注意: ADC、SOC**

一次转换在启动后必须在下一次转换启动前完成,否则会产生不可预测的转换数据。

微处理机内核可经过内部系统总线与 A/D 通信。这需要提供适当的模块地址以及为 ACTL 和 AEN 寄存器提供所要求的条件。微处理机从 ADAT 寄存器中将转换数据读回。

在省电模式下,整个 A/D 转换器关闭以停止电流消耗;但是,这只有在 SVCC 不是外部驱动时才正确。当发生转换启动信号或上电信号时,转换器被唤醒;但是,为保证转换精度,转换器要经过  $6\ \mu\text{s}$  才能达到就绪状态。

## 15.2 A/D 转换操作

### 15.2.1 A/D 转换

#### 1. A/D 转换原理

图 15.2 为 ADC 原理图。上电后须对 ACTL 寄存器编程,以确定是作比例检测还是绝对检测,以及检测范围的选择是非自动的还是自动的。对于非自动模式,检测范围位一旦选定后,在转换期间即不可改变,以免使转换结果失效。

对 ACTL 寄存器中的 SOC 位置位,将激活 A/D 时钟并开始一次新的转换。转换器的工作

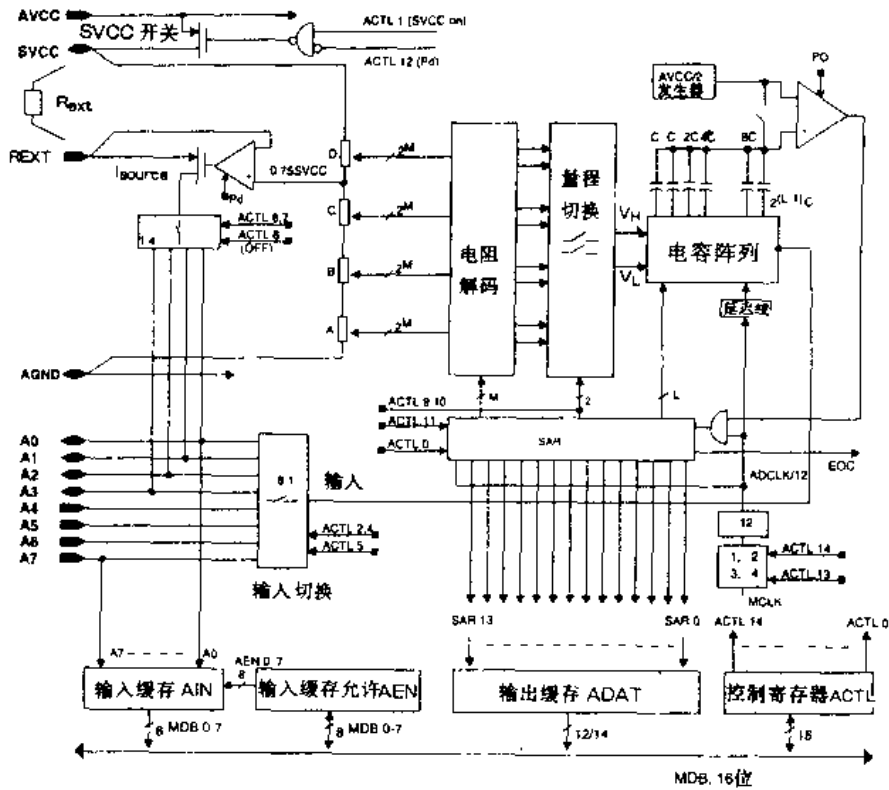


图 15.2 ADC 原理图

基于逐位逼近技术：先用一个电阻阵列分辨  $M$  位高位，然后用一个开关电容阵列分辨余下的  $L$  位低位。

由  $2^M$  个等权值的电阻组成的电阻阵列构成一个 D/A。由  $L$  个电容器组成的电容阵列构成一个电荷分配型 A/D。电容器是 2 进制权值，即它们的容量从最小值开始，按 2 的幂增加。电容的数量对应转换器的检测电压范围或输出码的低  $L$  位。

当选择了感兴趣的模拟通道后，采样过程就开始了。模拟输入电压采样到电容阵列的上电极。采样后模拟开关与 A/D 断开，因此，这时外部模拟信号可不必输入。

在电阻阵列上完成一次逐次逼近搜索以找到对应于输入电压的电压范围。从一个电阻上得到电压  $V_H$  和  $V_L$ ，同时得到高位。电容阵列根据电压差 ( $V_H - V_L$ ) 用一个类似的从最高位电容开始的逐位逼近搜索来得到低位。

电容阵列中从最大电容到最小电容连续作开关切换，因此，初始存储的电荷在电容上不断地再分配。电阻阵列的开关连接和电容阵列下电极的开关连接将改变上电极的电压，使它尽可能接近输入电压  $V_{in}$ 。开关的设置对应于 2 进制码 (12 位或 (12 + 2) 位)，它可以表示为一个分式  $V_{in}/V_{ref}$ 。

上电极电压由一个带偏移消除电路的比较器来监视，以确定输入电压是高于还是低于上电极电压，并产生决定逐位逼近搜索方向的数字控制信号。

当最小电容接入后，仍然会存在一个最小电压差异，这就是转换器的分辨率，或者可表示为  $V_{ref}/2^n$ ，其中的  $n$  是数据位数。

转换过程完成后，上电极的电压差根据转换器的最低位分辨率已尽可能地接近于 0。转换结束信号 (EOC) 指示在 ADAT 寄存器中可以读出一个 12 位或 (12 + 2) 位转换数据以作进

一步的处理。

## 2. A/D 转换时序

省电模式位的复位将激活 ADC 模块。为使内部偏压能准确地建立,激活过程至少需要  $6\ \mu\text{s}$ 。

A/D 转换工作在 ADCLK 频率的  $1/12$  的时钟频率下。ADCLK 频率的选择要适合电路特性。由于与模拟输入采样和网络转换有关的内部的时间常数,如果 ADCLK 频率太高,则无法保证精确的 12 位转换。由于电容阵列的放电,ADCLK 频率太低也无法保证精度,即使输入信号在采集期间是保持有效和稳定的。用 ACTL 寄存器中的 2 位(ADCLK)可以选择准确的 ADCLK 频率,即对 MCLK 时钟作  $1\sim 4$  分频。

模拟信号采样消耗 12 个 ADCLK 时钟脉冲,且 12 位的转换需要  $7\times 12$  个 ADCLK 时钟周期。当 ACTL.11 复位,且 A/D 转换工作在预选测量范围的 12 位转换模式时,时间消耗即如上述。总的 12 位转换消耗 96 个 ADCLK 时钟周期。图 15.3 为在 12 位转换模式下 A/D 转换时序图。

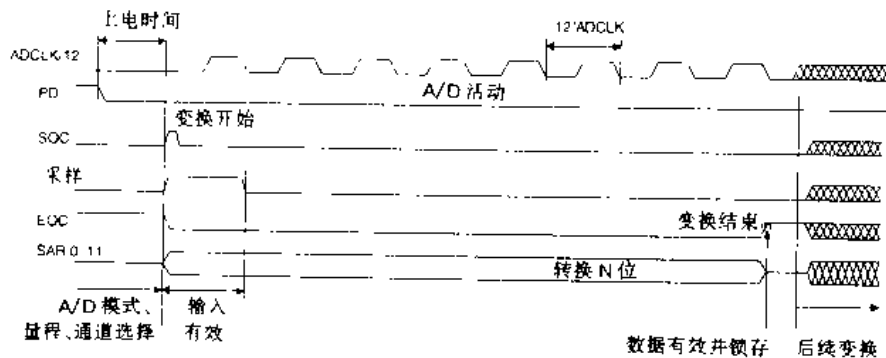


图 15.3 ADC 时序——12 位转换

当 ACTL.11 置位时, A/D 工作在自动选择测量范围的  $(12+2)$  位转换模式下。图 15.4 为在  $(12+2)$  位转换模式下 A/D 转换时序图。模拟输入信号要采样两次,每次消耗 12 个 ADCLK 时钟周期。经第一次对输入信号的采样,完成测量范围的转换需花费的时间为 24 个 ADCLK 时钟周期。在第二次对输入信号采样后,进行耗时 84 个 ADCLK 时钟周期的 12 位转换。因此,总的  $(12+2)$  位转换需要 132 个 ADCLK 时钟周期。

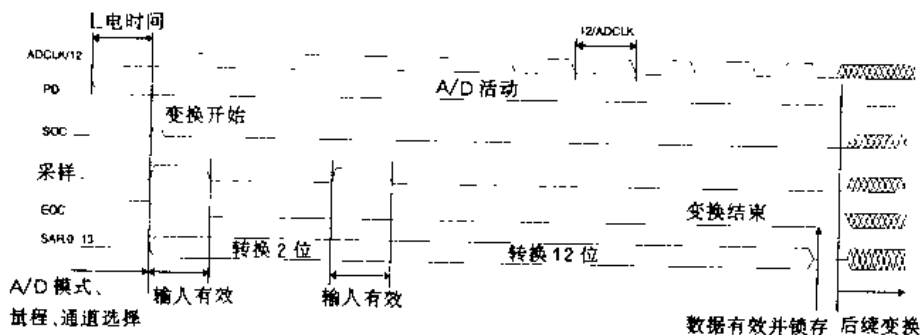


图 15.4 ADC 时序—— $(12+2)$  位转换

为了保证转换的精度,在采样期间输入信号必须保持有效和稳定。在整个转换期间相邻通道上也不应有活动的数字信号,以保证不会因供电电源毛刺、接地电压变化和交互干扰等造

成的误差而引起信号品质下降。

A/D 转换器采用了电荷再分配方法。当输入端经内部开关切换到采样端时,因为开关的作用会引起转移电流流入或流出模拟输入端。图 15.5 为 A/D 转换时输入采样时序。

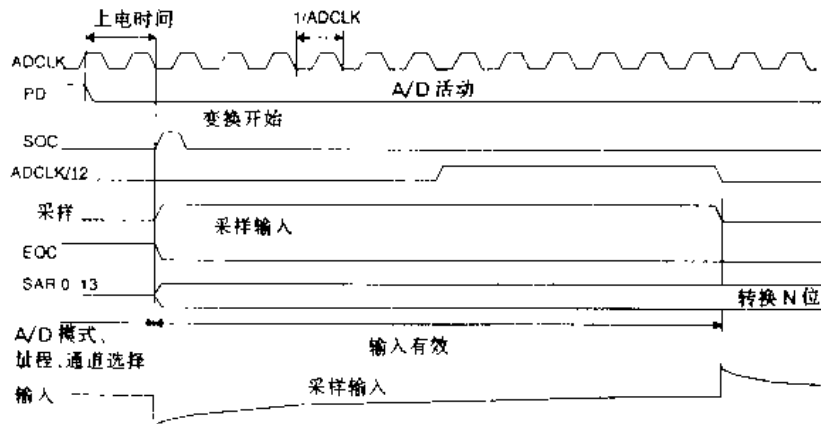


图 15.5 ADC——输入采样时序

这些电流跳变发生在内部采样脉冲的上升沿和下降沿。它们在引起问题之前会很快地衰减和消失,因为它们的时间常数比转换器内部的等效 RC 时间常数要小。模拟输入端可看做由 32 pF(电容阵列)和 2 kΩ 电阻(模拟开关接通)串接的等效 RC 电路。然而当外部信号源的动态内阻很大时,这些跳变在设定的采样时间内可能并未消失,从而难以保证 12 位或(12+2)位的精度。

### 15.2.2 A/D 中断

当 A/D 转换结束时,EOC 信号变高,并用对中断标志位 ADIFG 置位来激活 A/D 中断电路,通知系统的各个部分 A/D 转换已完成。当中断允许位 ADIE 置位时,即会发生中断请求。

### 15.2.3 A/D 量程

可以选择 4 个测量范围之一,以在一个指定的测量范围内产生 12 位精度的结果。如果 ACTL.11 复位,则可以得到等效的 14 位动态范围。测量范围的定义要在转换开始之前通过 ACTL.9 和 ACTL.10 来确定。如果 ACTL.11 置位,则转换器在采样期间通过对输入信号两次采样的方法找到适当的测量范围。一次采样用于选择测量范围,第二次采样实现 12 位转换,因此可得到全量程的具有(12+2)位精度的结果。

测量范围如下:

$0.00 \times V_{ref} \leq V_{in} < 0.25 \times V_{ref}$	范围 A
$0.25 \times V_{ref} \leq V_{in} < 0.50 \times V_{ref}$	范围 B
$0.50 \times V_{ref} \leq V_{in} < 0.75 \times V_{ref}$	范围 C
$0.75 \times V_{ref} \leq V_{in} < 1.00 \times V_{ref}$	范围 D

其中  $V_{ref}$  是 SVCC 端的电压。它或者由外部供给,或者连接到 AVCC 上。由 ACTL.12 控制 SVCC 开关的连接。

选定适当的测量范围后,通过相应控制位的选择,输入信号被连接到转换器的输入端。

A/D 转换器对选定通道的输入信号作处理,并且通过 ADAT 寄存器,软件可以对转换结果作访问。

在任一测量范围内的数码(10 进制)如下:

$$N_{10D} = \text{INT} \left[ V_{in} \times 2^{14} / V_{ref} - 2^{13} \times \text{ACTL}.10 - 2^{12} \times \text{ACTL}.9 \right]$$

式中: ACTL.10 和 ACTL.9 分别是 ACTL 寄存器的第 10 位和第 9 位。

对于 12 位转换模式, N 的范围如下:

$0000h \leq N \leq 0FFFh$	范围 A
$0000h \leq N \leq 0FFFh$	范围 B
$0000h \leq N \leq 0FFFh$	范围 C
$0000h \leq N \leq 0FFFh$	范围 D

对于(12+2)位转换模式,则 N 的范围如下:

$$0000h \leq N \leq 3FFFh$$

#### 注意: ADC 偏移电压

对 SVCC 或 AGND 端的寄生阻抗会引起在电阻阵列两端产生电压下降。这种电压下降引起的任何偏移电压( $V_{io}$ ),都会对输出数码和转换关系造成失真。

### 15.2.4 A/D 电流源

图 15.6 为 A/D 电流源原理图。有 4 个模拟 I/O 端可以作为电流源输出。电流源输出( $I_{source}$ )可以经外部电阻  $R_{ext}$  设定在引脚 A0、A1、A2、A3 上提供。其值为

$$I_{source} = (0.25 \times V_{ref}) / R_{ext}$$

式中:  $V_{ref}$ ——SVCC 端电压;

$R_{ext}$ ——SVCC 与 REXT 间的外接电阻值。

在作比例测量时,输入通道(仅对 A0、A1、A2、A3)上经阻抗元件产生的电压( $V_{in}$ )为

$$V_{in} = (0.25 \times SVCC) \times (R_{sens} / R_{ext})$$

式中:  $R_{sens}$ ——外接阻抗元件的电阻。

当 A/D 转换器在传感器应用中与外接电阻元件相联时,需要精密的恒流源。这样输入信号能以同样方式参照作为恒流源的供电电压或参考电压。从而在进行比例测量时,只要参考电压稳定,将不受它精度的影响。

### 15.2.5 A/D 输入端与多路切换

#### 1. 模拟输入

模拟输入信号采样后加在内部电容上,并且在转换期间保持。电容上的电荷由信号源提供。充电时间规定为 12 个 ADCLK 周期的采样时间。因此,外部信号源的电阻和动态阻抗须受限制,以保证 RC 时间常数要足够短,从而确保 12 位精度要在分配好的采样时间内完成输入信号的建立。时间常数的典型值为  $0.8/f_{ADCLK}$ 。高信号源阻抗对转换器精度有不利影响,这不仅因为 RC 建立稳定的时间特性,也由于漏电流或 DC 输入平均电流(输入开关电流)引起的输入端电压降。对于 12 位转换器,因漏电流引起的以 LSB 表示的误差为

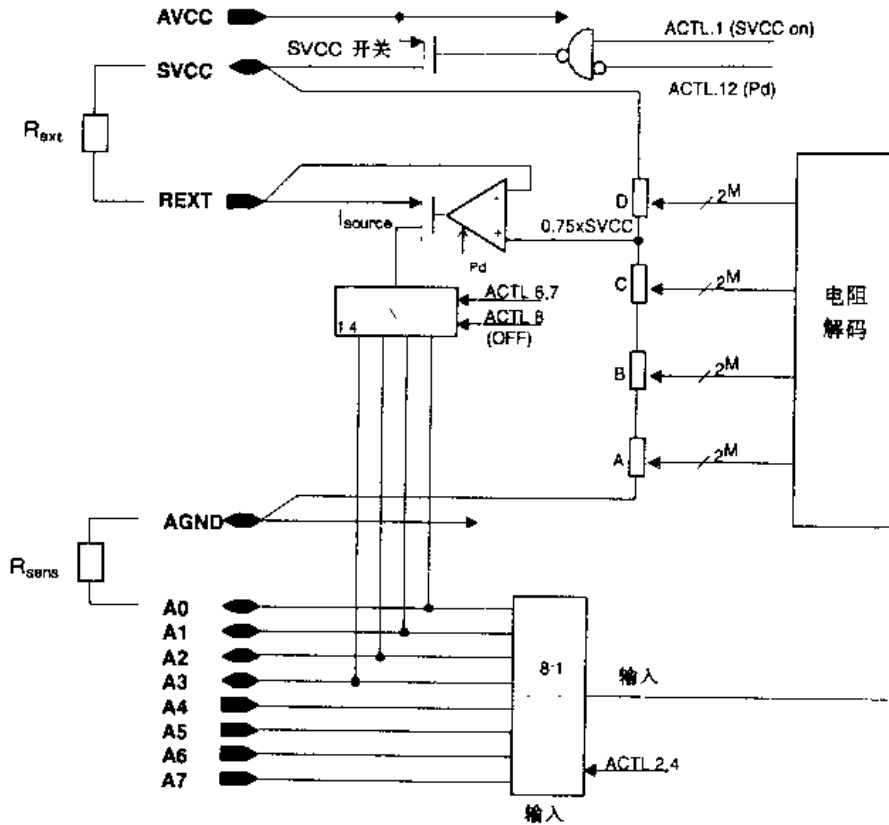


图 15.6 A/D 电流源

$$\text{Error}(\text{LSB}) = 4 \times \text{漏电流}(\mu\text{A}) \times \text{信号源阻抗}(\text{k}\Omega) / V_{\text{ref}}(\text{V})$$

例：漏电流 = 50 nA，信号源阻抗 = 10 kΩ， $V_{\text{ref}} = 3 \text{ V}$ ，则会产生 0.7 LSB 的误差。

这一点同样适用于参考电压源的输出阻抗。它应足够小以确保能在  $(0.2/f_{\text{ADCLK}})$  秒内建立稳定，因产生漏电流引入的误差应远小于 1 LSB。

## 2. 模拟多路开关

模拟多路开关(见 15.7)用 ACTL 中的控制位确定选择 8 个单端输入通道之一。它用 T 形开关使通道间的耦合减至最小,以避免对输入模拟信号的影响。工作时将未选中的通道与 A/D 绝缘,并将中间节点接地 AGND,而分布电容的接地消除了交互干扰。

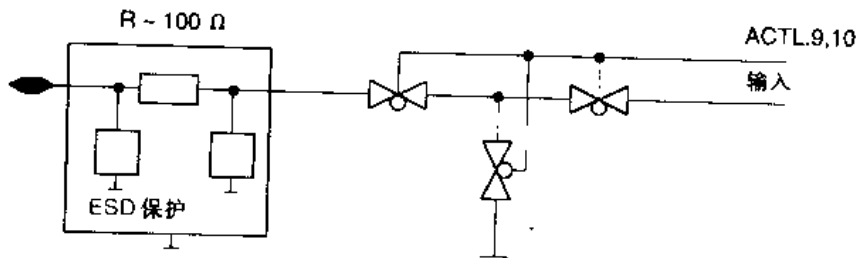


图 15.7 模拟多路开关

存在交互干扰的原因是总是存在跨越开关以及开关之间的寄生电容。它可能有多种形式。例如：一个关断开关的输入和输出间,一个关断的模拟输入通道与一个相邻的开通的输出通道之间,引起的误差会进入输出的数码中。为了实现高精度的转换,必须采用屏蔽和各种

常见的 PCB 布线技术将各种交互干扰减至最少。

### 15.2.6 A/D 接地与降噪

对任何一个高分辨率转换( $\geq 12$  位), 必须特别注意 PCB 布线和接地方案, 以消除地电流环路、寄生元件参数效应和噪声。有许多标准技术可以在各种应用注意事项中找到。

地电流环路是这样形成的, A/D 的电阻分压器的回传电流流过了与其他模拟或数字电路公共的一段线路。如果不注意, 这样的电流会产生一个小的偏移电压, 叠加在 A/D 转换器的参考电压和输入电压上。避免地电流环路的方法之一是对 AGND 采用星形连接方案。在这种方案中, 地电流或参考电流不会流过公共的输入回路, 从而消除了引起误差的电压。图 15.8 为 A/D 的接地和消除噪声原理图。

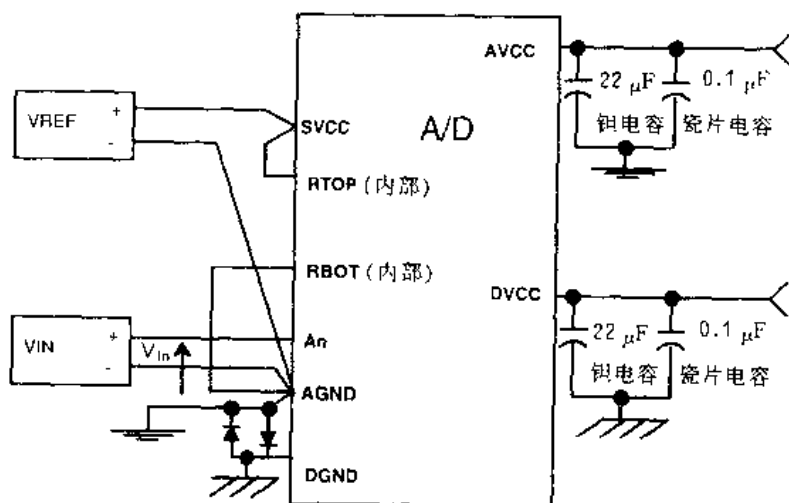


图 15.8 A/D 的接地和消除噪声原理图

数字地 DGND 与模拟地 AGND 可以采用星形方案连接在一起; 但如果采用分离的供电方案, 则用两个反向连接的二极管可以限制低于  $\pm 700$  mV 的电压差。

此外, 因数字开关电路和开关电源造成叠加在供电线路上的纹波和噪声很让人伤脑筋。

通常内部噪声很小, 折算成输入信号远小于一个 LSB, 因此输出代码非常稳定; 然而噪声经过供电电路及接地线耦合到设备上, 噪声的作用加强了, 信号会引起不稳定。这可以通过多次采样求平均的方法来改善。另一方面, 如果 SVCC 端的电压降低, 那么相当于 LSB 对应的绝对数值降低, 这样也造成噪声的作用加强。因此, 为达到所要求的精度, 干净而无噪声的电路是最重要的。

仔细地安排旁路电容, 建立回到各自的地回路, 对稳定供电电流和降低噪声是有帮助的。

### 15.2.7 A/D 输入与输出引脚

#### 1. 输入引脚

有两种不同类型的输入信号, 即由通道端 A0 ~ A7 输入的模拟信号, 以及由 REXT、SVCC 端输入的模拟信号。











## 第 16 章 其他模块

### 16.1 晶体振荡器

晶振工作所需的所有元件都集成在 MSP430 内,因此不需要别的外接器件。晶振电路是按超低功耗要求设计的,因此,印制电路设计应使晶体和 MSP430 器件间的连线要短。图 16.1 为晶振原理图。

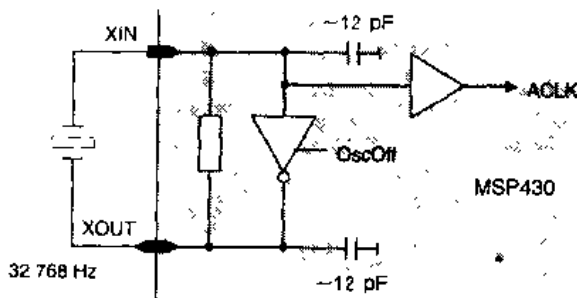
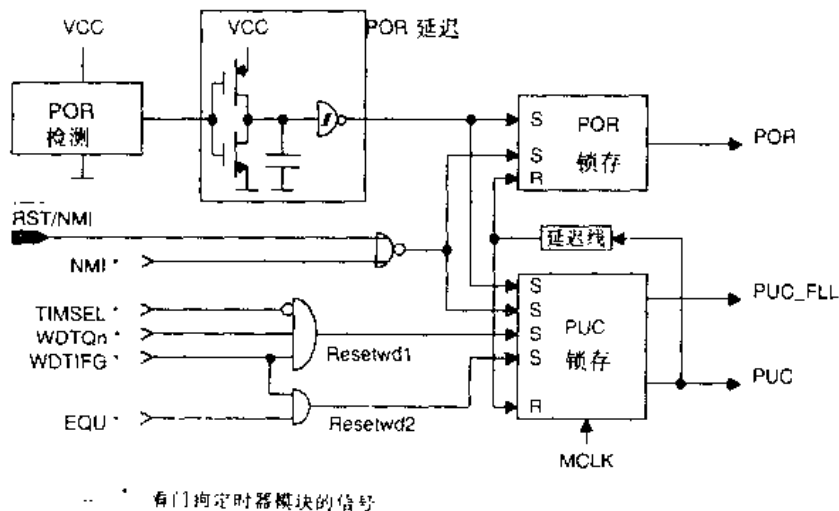


图 16.1 晶振原理图

在 OscOff 模式下, ACLK 信号保持为高电平。

### 16.2 上电电路

上电电路是系统复位电路的一部分,它由两部分组成,即上电复位检测和上电复位延迟,如图 16.2 所示。为了给系统提供复位条件, POR 的输出延迟送到 POR 锁存器和 PUC 锁存器,以保证对这两个锁存器复位。



看门狗定时器模块的信号

图 16.2 上电复位和上电清除电原理图

VCC 的上升时间较快时(见图 16.3), POR 延迟足够长的时间, 以保证 POR 信号在上电后能正确地对电路作初始化。

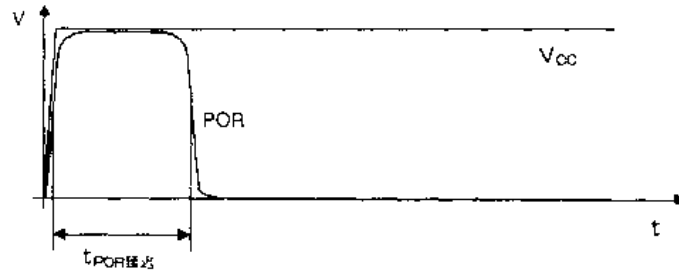


图 16.3 VCC 上电时间较快时的上电复位

VCC 的上升时间较慢时(见图 16.4), POR 通过检测 VCC 来确定 POR 信号, 以保证 POR 信号在上电后能正确地对电路初始化。

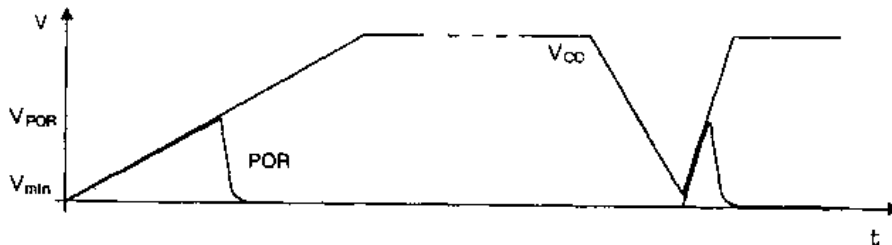


图 16.4 VCC 上电时间较慢时的上电复位

图 16.4 中, 供电电压 VCC 跌落到  $V_{min}$  以下, 以确保第二个 POR 信号在 VCC 电压又一次上升时发生。如果 VCC 不能跌落到小于  $V_{min}$ , 则 POR 信号不会产生, 上电条件不能建立。

### 16.3 晶振缓冲输出

由控制寄存器 CBCTL 选择晶体缓冲输出的频率。图 16.5 为晶振缓冲原理图。

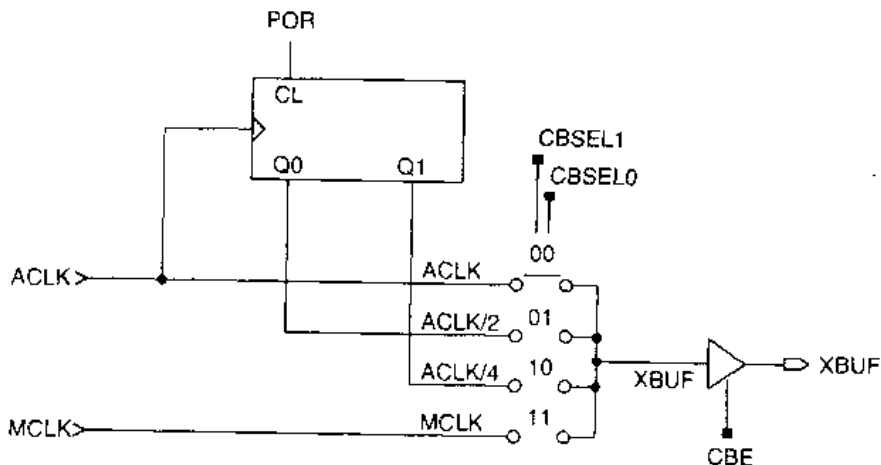


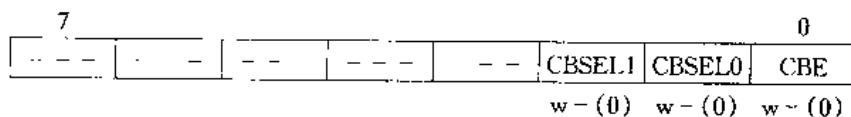
图 16.5 晶振缓冲原理图

由时钟缓冲输出外围模块的控制寄存器控制 XBUF 引脚输出的频率及输出缓冲的三态。

分频器运行在正确操作必需的最小值。例如,当选择 ACLK 或 MCLK,或者 CBE 置位时,它将暂停工作。

控制寄存器的 3 位 CBSEL1、CBSECO、CBE 用 POR 信号复位。在 VCC 上电过程中,或者在 RST/NMI 选择复位功能时接地,POR 信号有效。控制寄存器 CBCTL 的位定义如下:

CBCTL (053h)



对 CBCTL 各位说明如下:

位 0: CBE 位控制输出缓冲的三态。

CBE = 1: 输出缓冲允许。

CBE = 0: 输出缓冲禁止。

在上电复位(POR)时,输出缓冲总是禁止的。外部元件不会得到所选的频率。

位 1,2: CBSEL1 和 CBSEL0 位选择在 XBUF 引脚上提供的信号频率。

CBSEL1	CBSEL0	XBUF 引脚信号频率
0	0	$f_{\text{ACLK}}$ (POR 后状态)
0	1	$f_{\text{ACLK}}/2$
1	0	$f_{\text{ACLK}}/4$
1	1	$f_{\text{MCLK}}$





Bit #	7	6	5	4	3	2	1	0
002Fh	P2SEL.7 rw-0	P2SEL.6 rw-0	P2SEL.5 rw-0	P2SEL.4 rw-0	P2SEL.3 rw-0	P2SEL.2 rw-0	P2SEL.1 rw-0	P2SEL.0 rw-0
002Eh	P2IE.7 rw-0	P2IE.6 rw-0	P2IE.5 rw-0	P2IE.4 rw-0	P2IE.3 rw-0	P2IE.2 rw-0	P2IE.1 rw-0	P2IE.0 rw-0
002Dh	P2IES.7 rw-0	P2IES.6 rw-0	P2IES.5 rw-0	P2IES.4 rw-0	P2IES.3 rw-0	P2IES.2 rw-0	P2IES.1 rw-0	P2IES.0 rw-0
002Ch	P2IFG.7 rw-0	P2IFG.6 rw-0	P2IFG.5 rw-0	P2IFG.4 rw-0	P2IFG.3 rw-0	P2IFG.2 rw-0	P2IFG.1 rw-0	P2IFG.0 rw-0
002Bh	P2DIR.7 rw-0	P2DIR.6 rw-0	P2DIR.5 rw-0	P2DIR.4 rw-0	P2DIR.3 rw-0	P2DIR.2 rw-0	P2DIR.1 rw-0	P2DIR.0 rw-0
002Ah	P2OUT.7 rw-0	P2OUT.6 rw-0	P2OUT.5 rw-0	P2OUT.4 rw-0	P2OUT.3 rw-0	P2OUT.2 rw-0	P2OUT.1 rw-0	P2OUT.0 rw-0
0028h	P2IN.7 rw-0	P2IN.6 rw-0	P2IN.5 rw-0	P2IN.4 rw-0	P2IN.3 rw-0	P2IN.2 rw-0	P2IN.1 rw-0	P2IN.0 rw-0
0027h	P1SEL.7 rw-0	P1SEL.6 rw-0	P1SEL.5 rw-0	P1SEL.4 rw-0	P1SEL.3 rw-0	P1SEL.2 rw-0	P1SEL.1 rw-0	P1SEL.0 rw-0
0026h	P1IE.7 rw-0	P1IE.6 rw-0	P1IE.5 rw-0	P1IE.4 rw-0	P1IE.3 rw-0	P1IE.2 rw-0	P1IE.1 rw-0	P1IE.0 rw-0
0025h	P1IES.7 rw-0	P1IES.6 rw-0	P1IES.5 rw-0	P1IES.4 rw-0	P1IES.3 rw-0	P1IES.2 rw-0	P1IES.1 rw-0	P1IES.0 rw-0
0024h	P1IFG.7 rw-0	P1IFG.6 rw-0	P1IFG.5 rw-0	P1IFG.4 rw-0	P1IFG.3 rw-0	P1IFG.2 rw-0	P1IFG.1 rw-0	P1IFG.0 rw-0
0023h	P1DIR.7 rw-0	P1DIR.6 rw-0	P1DIR.5 rw-0	P1DIR.4 rw-0	P1DIR.3 rw-0	P1DIR.2 rw-0	P1DIR.1 rw-0	P1DIR.0 rw-0
0022h	P1OUT.7 rw-0	P1OUT.6 rw-0	P1OUT.5 rw-0	P1OUT.4 rw-0	P1OUT.3 rw-0	P1OUT.2 rw-0	P1OUT.1 rw-0	P1OUT.0 rw-0
0021h	P1IN.7 rw-0	P1IN.6 rw-0	P1IN.5 rw-0	P1IN.4 rw-0	P1IN.3 rw-0	P1IN.2 rw-0	P1IN.1 rw-0	P1IN.0 rw-0
0020h								
001Fh	P4SEL.7 rw-0	P4SEL.6 rw-0	P4SEL.5 rw-0	P4SEL.4 rw-0	P4SEL.3 rw-0	P4SEL.2 rw-0	P4SEL.1 rw-0	P4SEL.0 rw-0
001Eh	P4DIR.7 rw-0	P4DIR.6 rw-0	P4DIR.5 rw-0	P4DIR.4 rw-0	P4DIR.3 rw-0	P4DIR.2 rw-0	P4DIR.1 rw-0	P4DIR.0 rw-0
001Dh	P4OUT.7 rw-0	P4OUT.6 rw-0	P4OUT.5 rw-0	P4OUT.4 rw-0	P4OUT.3 rw-0	P4OUT.2 rw-0	P4OUT.1 rw-0	P4OUT.0 rw-0
001Ch	P4IN.7 rw-0	P4IN.6 rw-0	P4IN.5 rw-0	P4IN.4 rw-0	P4IN.3 rw-0	P4IN.2 rw-0	P4IN.1 rw-0	P4IN.0 rw-0
001Bh	P3SEL.7 rw-0	P3SEL.6 rw-0	P3SEL.5 rw-0	P3SEL.4 rw-0	P3SEL.3 rw-0	P3SEL.2 rw-0	P3SEL.1 rw-0	P3SEL.0 rw-0
001Ah	P3DIR.7 rw-0	P3DIR.6 rw-0	P3DIR.5 rw-0	P3DIR.4 rw-0	P3DIR.3 rw-0	P3DIR.2 rw-0	P3DIR.1 rw-0	P3DIR.0 rw-0
0019h	P3OUT.7 rw-0	P3OUT.6 rw-0	P3OUT.5 rw-0	P3OUT.4 rw-0	P3OUT.3 rw-0	P3OUT.2 rw-0	P3OUT.1 rw-0	P3OUT.0 rw-0
0018h	P3IN.7 rw-0	P3IN.6 rw-0	P3IN.5 rw-0	P3IN.4 rw-0	P3IN.3 rw-0	P3IN.2 rw-0	P3IN.1 rw-0	P3IN.0 rw-0
0017h								
0016h	P0IE.7 rw-0	P0IE.6 rw-0	P0IE.5 rw-0	P0IE.4 rw-0	P0IE.3 rw-0	P0IE.2 rw-0	P0IE.1 rw-0	P0IE.0 rw-0
0015h	P0IES.7 rw-0	P0IES.6 rw-0	P0IES.5 rw-0	P0IES.4 rw-0	P0IES.3 rw-0	P0IES.2 rw-0	P0IES.1 rw-0	P0IES.0 rw-0
0014h	P0IFG.7 rw-0	P0IFG.6 rw-0	P0IFG.5 rw-0	P0IFG.4 rw-0	P0IFG.3 rw-0	P0IFG.2 rw-0	P0IFG.1 rw-0	P0IFG.0 rw-0
0013h	P0DIR.7 rw-0	P0DIR.6 rw-0	P0DIR.5 rw-0	P0DIR.4 rw-0	P0DIR.3 rw-0	P0DIR.2 rw-0	P0DIR.1 rw-0	P0DIR.0 rw-0
0012h	P0OUT.7 rw-0	P0OUT.6 rw-0	P0OUT.5 rw-0	P0OUT.4 rw-0	P0OUT.3 rw-0	P0OUT.2 rw-0	P0OUT.1 rw-0	P0OUT.0 rw-0
0011h	P0IN.7 rw-0	P0IN.6 rw-0	P0IN.5 rw-0	P0IN.4 rw-0	P0IN.3 rw-0	P0IN.2 rw-0	P0IN.1 rw-0	P0IN.0 rw-0
0010h								

图 A.2 数字 I/O 组——字节访问

\*) 中断允许位和中断标志位包含在 SFR 组中。

Bit #	7	6	5	4	3	2	1	0
定时器/端口允许寄存器 TPE 04Fh	TPSSSEL3 rw-0	TPSSSEL2 rw-0	TPE5 rw-0	TPE4 rw-0	TPE3 rw-0	TPE2 rw-0	TPE1 rw-0	TPE0 rw-0
定时器/端口数据寄存器 TPD 04Eh	B15 rw-0	CPON rw-0	TPD.5 rw-0	TPD.4 rw-0	TPD.3 rw-0	TPD.2 rw-0	TPD.1 rw-0	TPD.0 rw-0
定时器/端口计数器 1 TPCNT1 04Ch	2 <sup>7</sup> rw	2 <sup>6</sup> rw	2 <sup>5</sup> rw	2 <sup>4</sup> rw	2 <sup>3</sup> rw	2 <sup>2</sup> rw	2 <sup>1</sup> rw	2 <sup>0</sup> rw
定时器/端口控制寄存器 TPCTL 04Bh	TPSSSEL1 rw-0	TPSSSEL0 rw-0	ENB rw-0	ENA rw-0	EN1 rw-0	RC2FG rw-0	RC1FG rw-0	EN1FG rw-0
计数器 8 位数据 Basic Timer, BTCNT2 0047h	2 <sup>7</sup> rw	2 <sup>6</sup> rw	2 <sup>5</sup> rw	2 <sup>4</sup> rw	2 <sup>3</sup> rw	2 <sup>2</sup> rw	2 <sup>1</sup> rw	2 <sup>0</sup> rw
计数器 8 位数据 Basic Timer, BTCNT1 0046h	2 <sup>7</sup> rw	2 <sup>6</sup> rw	2 <sup>5</sup> rw	2 <sup>4</sup> rw	2 <sup>3</sup> rw	2 <sup>2</sup> rw	2 <sup>1</sup> rw	2 <sup>0</sup> rw
计数器 8 位数据 定时器/计数器 TCDAT 0044h	TCDAT.7 rw	TCDAT.6 rw	TCDAT.5 rw	TCDAT.4 rw	TCDAT.3 rw	TCDAT.2 rw	TCDAT.1 rw	TCDAT.0 rw
8 位预置寄存器 定时器/计数器 TOPLD 0043h	TCPLD.7 rw	TCPLD.6 rw	TCPLD.5 rw	TCPLD.4 rw	TCPLD.3 rw	TCPLD.2 rw	TCPLD.1 rw	TCPLD.0 rw
8 位控制寄存器 定时器/计数器 TCC7L 0042h	SSEL1 rw-0	SSEL0 rw-0	ISCTL rw-0	TXEN rw-0	ENCNT rw-0	RXACT rw-0	TXD rw-0	RXD rw-0
Basic Timer, BTC7L 0040h	SSEL rw	Reset* Hold** rw	DIV rw	FRQ1 rw	FRQ0 rw	IP2 rw	IP1 rw	IP0 rw

图 A.4 8 位定时器/计数器组、Basic Timer 组、定时器/端口组——字节访问

Bit #	7	6	5	4	3	2	1	0
LCD 存储器 16 003Fh	S28C3 rw	S28C2 rw	S28C1 rw	S28C0 rw	S28C3 rw	S28C2 rw	S28C1 rw	S28C0 rw
LCD 存储器 14 003Eh	S27C3 rw	S27C2 rw	S27C1 rw	S27C0 rw	S26C3 rw	S26C2 rw	S26C1 rw	S26C0 rw
LCD 存储器 13 003Dh	S25C3 rw	S25C2 rw	S25C1 rw	S25C0 rw	S24C3 rw	S24C2 rw	S24C1 rw	S24C0 rw
LCD 存储器 12 003Ch	S23C3 rw	S23C2 rw	S23C1 rw	S23C0 rw	S22C3 rw	S22C2 rw	S22C1 rw	S22C0 rw
LCD 存储器 11 003Bh	S21C3 rw	S21C2 rw	S21C1 rw	S21C0 rw	S20C3 rw	S20C2 rw	S20C1 rw	S20C0 rw
LCD 存储器 10 003Ah	S19C3 rw	S19C2 rw	S19C1 rw	S19C0 rw	S18C3 rw	S18C2 rw	S18C1 rw	S18C0 rw
LCD 存储器 8 0039h	S17C3 rw	S17C2 rw	S17C1 rw	S17C0 rw	S16C3 rw	S16C2 rw	S16C1 rw	S16C0 rw
LCD 存储器 6 0037h	S15C3 rw	S15C2 rw	S15C1 rw	S15C0 rw	S14C3 rw	S14C2 rw	S14C1 rw	S14C0 rw
LCD 存储器 7 0036h	S13C3 rw	S13C2 rw	S13C1 rw	S13C0 rw	S12C3 rw	S12C2 rw	S12C1 rw	S12C0 rw
LCD 存储器 6 0035h	S11C3 rw	S11C2 rw	S11C1 rw	S11C0 rw	S10C3 rw	S10C2 rw	S10C1 rw	S10C0 rw
LCD 存储器 5 0034h	S9C3 rw	S9C2 rw	S9C1 rw	S9C0 rw	S8C3 rw	S8C2 rw	S8C1 rw	S8C0 rw
LCD 存储器 4 0033h	S7C3 rw	S7C2 rw	S7C1 rw	S7C0 rw	S6C3 rw	S6C2 rw	S6C1 rw	S6C0 rw
LCD 存储器 3 0032h	S5C3 rw	S5C2 rw	S5C1 rw	S5C0 rw	S4C3 rw	S4C2 rw	S4C1 rw	S4C0 rw
LCD 存储器 2 0031h	S3C3 rw	S3C2 rw	S3C1 rw	S3C0 rw	S2C3 rw	S2C2 rw	S2C1 rw	S2C0 rw
LCD 存储器 1 0030h	S1C3 rw	S1C2 rw	S1C1 rw	S1C0 rw	S0C3 rw	S0C2 rw	S0C1 rw	S0C0 rw
LCD 控制与模式 LCDC 0030h	LCDM7 rw-0	LCDM6 rw-0	LCDM5 rw-0	LCDM4 rw-0	LCDM3 rw-0	LCDM2 rw-0	LCDM1 rw-0	LCDM0 rw-0

图 A.3 LCD 寄存器组——字节访问

注意：LCD 存储器位按 MSP430 规则命名。各位名称的第一部分是相应的 SEG 线，第二部分是相应的 COM 线。例如，对于用了 SEG4 和 COM3 的段的命名为 S4C3。



Bit #	7	6	5	4	3	2	1	0
07Fh								
发送缓存 TXBUF 077h	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
接收缓存 RXBUF 076h	r	r	r	r	r	r	r	r
波特率寄存器 UBR1 075h	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
波特率寄存器 UBR0 074h	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
调整控制 UMCTL 073h	m7	m6	m5	m4	m3	m2	m1	m0
接收控制 URCTL 072h	r/w-0	undef. r/w-0	OE r/w-0	undef. r/w-0	unused r/w-0	unused r/w-0	undef. r/w-0	undef. r/w-0
发送控制 UTCTL 071h	CKPH r/w-0	CKPL r/w-0	SSEL1 r/w-0	SSEL0 r/w-0	unused r/w-0	unused r/w-0	STC r/w-0	TXEPT r/w-1
USART 控制 UCTL 070h	unused r/w-0	unused r/w-0	unused r/w-0	CHAR r/w-0	Listen r/w-0	SYNC r/w-0	MM r/w-0	SWRST r/w-1

图 A.7 USART 组, 选择 SPI 模式: SYNC 位 = 1——字节访问

Bit #	15	14	13	12	11	10	9	8
11Fh								
A/D 转换器 数据寄存器 ADAT 118h	r0	r0	R1 <sup>*)</sup> r0	R0 <sup>*)</sup> r0	2 <sup>11</sup> r0	2 <sup>10</sup> r0	2 <sup>9</sup> r0	2 <sup>8</sup> r0
保留未用 116h								
A/D 转换器 控制寄存器 ACTL 114h	r0	r0	r0	r/w-1	r/w-0	r/w-0	r/w-0	r/w-0
A/D 转换器 输入允许寄存器 AEN 112h	r0	r0	r0	r0	r0	r0	r0	r0
A/D 转换器 输入数据寄存器 AIN 110h	r0	r0	r0	r0	r0	r0	r0	r0

\*) 当 ACTL.11 = 0 时, ADAT.12 和 ADAT.13 读出为 0; 否则, 读出 R0 和 R1。

Bit #	7	6	5	4	3	2	1	0
11Eh								
AD 转换器 数据寄存器 ADAT 118h	2 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>
保留未用 116h								
AD 转换器 控制寄存器 ACTL 114h	r/w-0	r/w-0	r/w-0	r/w-0	r/w-0	r/w-0	r/w-0	r/w-0
AD 转换器 输入允许寄存器 AEN 112h	r/w-0	r/w-0	r/w-0	r/w-0	r/w-0	r/w-0	r/w-0	r/w-0
AD 转换器 输入数据寄存器 AIN 110h	AIN.7	AIN.6	AIN.5	AIN.4	AIN.3	AIN.2	AIN.1	AIN.0

\*) 当 ACTL.11 = 0 时, ADAT.12 和 ADAT.13 读出为 0;  
当 ACTL.11 = 1 时, R0 和 R1 可读。

图 A.8 A/D 转换器寄存器组——字访问

Bit #	15	14	13	12	11	10	9	8
Timer_A 中断向量 TAIV 12Eh	r0	r0	r0	r0	r0	r0	r0	r0
看门狗定时器 控制寄存器 WDCTL 120h	w0	w1	w0	w1	w1	w0	w1	w0
Bit # <th>7</th> <th>6</th> <th>5</th> <th>4</th> <th>3</th> <th>2</th> <th>1</th> <th>0</th>	7	6	5	4	3	2	1	0
Timer_A 中断向量 TAIV 12Eh	r0	r0	r0	r(0)	r(0)	r(0)	r(0)	r0
看门狗定时器 控制寄存器 WDCTL 120h	HOLD	NMIES	NMI	TMSEL	CNTCL	SSEL	IS1	IS0
	r/w-0	r/w-0	r/w-0	r/w-0	(w),r0	r/w-0	r/w-0	r/w-0

图 A.9 看门狗/定时器寄存器和 Timer\_A 中断向量寄存器组——字访问

Bit #	15	14	13	12	11	10	9	8
和扩展字 SumExt 013Eh	r	r	r	r	r	r	r	r
结果高字 ResHi 013Ch	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
结果低字 ResLo 013Ah	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
操作数 2 OP2 0138h	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
乘加操作数 1 MAC 0134h	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
有符号乘操作数 1 MPYS 0132h	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
无符号乘操作数 1 MPY 0130h	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
Bit #	7	6	5	4	3	2	1	0
和扩展字 SumExt 013Eh	r	r	r	r	r	r	r	r
结果高字 ResHi 013Ch	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
结果低字 ResLo 013Ah	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
操作数 2 OP2 0138h	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
乘加操作数 1 MAC 0134h	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
有符号乘操作数 1 MPYS 0132h	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
无符号乘操作数 1 MPY 0130h	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

扩展寄存器 SumExt 保存 16×16 位乘积的符号(MPYS), 或者乘加运算的溢出部分(MAC)。SumExt 寄存器为

- 0FFFh 当乘法运算结果为负。
- 0h 当乘法运算结果为正。
- 0h 当乘加运算无溢出。
- 1h 当乘加运算有溢出。

图 A.10 乘法器寄存器组——字访问



## 附录 B 指令集说明

MSP430 的 CPU 内核的结构来自于采用具有高度透明格式的 RISC 指令的设计思想。指令分为硬件实现的内核指令和利用这一硬件结构且有更高效率的模拟指令。模拟指令利用了内核指令和内装的常数发生器 CG1 和 CG2。本附录对内核指令和模拟指令作说明。在程序例子中采用模拟指令助记符。

根据指令的寻址方式,它在程序存储器中所占空间为 1~3 个字。

每条指令至少用 1 个字。对于变址、符号、绝对和立即寻址方式,需要外加占用 1 个字的程序存储器空间。这 4 种寻址方式适用于源操作数。变址、符号和绝对寻址适用于目的操作数。

这样,一条指令加上它的源操作数和目的操作数可能要用 1~3 个字的代码存储器。

### B.1 指令汇总

指令	参数	简要说明	状态位 (V N Z C)
* ADC[.W]; ADC.B	目的	目的 + C → 目的	* * * *
ADD[.W]; ADD.B	源,目的	源 + 目的 → 目的	* * * *
ADDC[.W]; ADDC.B	源,目的	源 + 目的 + C → 目的	* * * *
AND[.W]; AND.B	源,目的	源 .and. 目的 → 目的	0 * * *
BIC[.W]; BIC.B	源,目的	.not.源 .and. 目的 → 目的	- - -
BIS[.W]; BIS.B	源,目的	源 .or. 目的 → 目的	- - - -
BIT[.W]; BIT.B	源,目的	源 .and. 目的	0 * * *
* BR	目的	转移到...	- - - -
CALL	目的	PC + 2 → 堆栈,目的 → PC	- -
* CLR[.W]; CLR.B	目的	清除目的数	- - - -
* CLRC		进位位 C 复位	- - - 0
* CLRN		负数位 N 复位	- 0 - -
* CLRZ		零位 Z 复位	- - 0 -
CMP[.W]; CMP.B	源,目的	目的 - 源	* * * *
* DADC[.W]; DADC.B	目的	目的 + C → 目的(十进制)	* * * *
DADD[.W]; DADD.B	源,目的	源 + 目的 + C → 目的(十进制)	* * * *
* DEC[.W]; DEC.B	目的	目的 - 1 → 目的	* * * *
* DECD[.W]; DECD.B	目的	目的 - 2 → 目的	* * * *
* DINT		关中断	- - - -
* EINT		开中断	- - - -
* INC[.W]; INC.B	目的	目的 + 1 → 目的	* * * *
* INCD[.W]; INCD.B	目的	目的 + 2 → 目的	* * * *

* INV[.W]; INV.B	目的	目的取反	* * * *
JC/JHS	Label	C=1 时跳转至 Label	- - - -
JEQ/JZ	Label	Z=1 时跳转至 Label	- - - -
JGE	Label	(N.XOR.V)=0 时跳转至 Label	- - - -
JG	Label	(N.XOR.V)=1 时跳转至 Label	- - - -
JMP	Label	无条件跳转至 Label	- - - -
JN	Label	N=1 时跳转至 Label	- - - -
JNC/JLO	Label	C=0 时跳转至 Label	- - - -
JNE/JNZ	Label	Z=0 时跳转至 Label	- - - -
MOV[.W]; MOV.B	源,目的	源→目的	
* NOP		空操作	- - - -
* POP[.W]; POP.B	目的	栈顶→目的, SP+2→SP	- - - -
PUSH[.W]; PUSH.B	源	SP-2→SP, 源→@SP	- - - -
RETI		栈顶→SR, SP+2→SP	* * * *
		栈顶→PC, SP+2→SP	
* RET		栈顶→PC, SP+2→SP	- - - -
* RLA[.W]; RLA.B	目的	算术左移	* * * *
* RLC[.W]; RLC.B	目的	经进位位左移	* * * *
RRA[.W]; RRA.B	目的	MSB→MSB→...LSB→C	0 * * *
RRC[.W]; RRC.B	目的	C→MSB→...LSB→C	* * * *
* SBC[.W]; SBC.B	目的	从目的减去进位位	* * * *
* SETC		进位位 C 置位	- - - 1
* SETN		负数位 N 置位	- 1 - -
* SETZ		零位置位	- - 1 -
SUB[.W]; SUB.B	源,目的	目的 + .not.源 + 1→目的	* * * *
SUBC[.W]; SUBC.B	源,目的	目的 + .not.源 + C→目的	* * * *
SWPB	目的	字节交换	- - - -
SXT	目的	位 7→位 8→...→位 15	0 * * *
* TST[.W]; TST.B	目的	目的数测试	0 * * 1
XOR[.W]; XOR.B	源,目的	源.xor.目的→目的	* * * *

注意: 带“\*”号的指令是模拟指令

所有带“\*”号的指令是模拟指令。模拟指令利用 CPU 的结构和内核指令实现, 并且有更高的代码效率和执行速度。

## B.2 指令格式

### 1. 双操作数指令(内核指令)

双操作数指令的 16 位代码分为 4 个字段:

- 操作码, 4 位 [OP-Code];
- 源操作数, 6 位 [S-Reg + As];
- 字节操作标志, 1 位 [B/W];



- 目的操作数, 5 位 [D-Reg + Ad]。

源操作数字段由 2 个寻址位和 4 位寄存器号(0~15)组成。目的操作数字段由 1 个寻址位和 4 位寄存器号组成(0~15)。字节标志 B/W 指明指令是作为字节(B/W=1)还是字(B/W=0)来操作。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP code				S-Reg				Ad	B/W	As	D-Reg				

指令	参数	简要说明	状态位				
			V	N	Z	C	
ADD[.W]	ADD.B	源, 目的	源 + 目的 → 目的	*	*	*	*
ADDC[.W]	ADDC.B	源, 目的	源 + 目的 + C → 目的	*	*	*	*
AND[.W]	AND.B	源, 目的	源 .and. 目的 → 目的	0	*	*	*
BIC[.W]	BIC.B	源, 目的	.not. 源 .and. 目的 → 目的	-	-	-	-
BIS[.W]	BIS.B	源, 目的	源 or. 目的 → 目的	-	-	-	-
BIT[.W]	BIT.B	源, 目的	源 .and. 目的	0	*	*	*
CMP[.W]	CMP.B	源, 目的	目的 - 源	*	*	*	*
DADD[.W]	DADD.B	源, 目的	源 + 目的 + C → 目的 (十进制)	*	*	*	*
MOV[.W]	MOV.B	源, 目的	源 → 目的	-	-	-	-
SUB[.W]	SUB.B	源, 目的	目的 \ .not. 源 → 目的	*	*	*	*
SUBC[.W]	SUBC.B	源, 目的	目的 + .not. 源 + C → 目的	*	*	*	*
XOR[.W]	XOR.B	源, 目的	源 .xor. 目的 → 目的	*	*	*	*

#### 注意: 操作时状态寄存器作为目的操作数

所有将状态寄存器作为目的操作数的操作, 用运算结果来覆盖状态寄存器的内容, 而指令操作对状态位额定影响不会发生。例: ADD #3, SR; 操作: (SR) + 3 → SR

## 2. 单操作数指令(内核指令)

单操作数指令的 16 位代码分为 3 个字段:

- 操作码, 9 位, 高 4 位为“0001”;
- 字节操作标志, 1 位, [B/W];
- 目的操作数, 6 位, [D-Reg + Ad]。

目的操作数字段由 2 个寻址位和 4 位寄存器号(0~15)组成。目的操作数字段的位置与双操作数相同。字节标志 B/W 指明指令是作为字节(B/W=1)还是字(B/W=0)来操作。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	x	x	x	x	x	B/W	As	D/S Reg				

指令	参数	简要说明	状态位				
			V	N	Z	C	
RRA[.W]	RRA.B	目的	MSB → MSB → ... → LSB → C	0	*	*	*
RRC[.W]	RRC.B	目的	C → MSB → ... → LSB → C	*	*	*	*
PUSH[.W]	PUSH.B	源	SP - 2 → SP, 源 → @SP	-	-	-	-

SWPB	目的	字节交换	-	-	-	-
CALL	目的	SP-2→SP	.	-	-	-
RETI		PC+2→堆栈, 目的→PC				
		TOS→SR, SP+2→SP	*	*	*	*
		TOS→PC, SP+2→SP				
SXT	目的	位 7→位 8...→位 15	0	*	*	*

### 3. 条件跳转指令和无条件跳转指令(内核指令)

条件跳转指令和无条件跳转指令的 16 位代码分为 2 个字段:

- 操作码, 6 位;
- 跳转偏移量, 10 位。

操作码字段由 3 位操作码和 3 位跳转条件组成。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	条件			10 位 PC 偏移量									

条件跳转的跳转范围为距当前地址的 -512~+512 个字的范围内。由汇编器计算带符号的偏移量作为代码中的操作数。

指令	参数	简要说明
JC/JHS	Label	C = 1 时, 跳转到 Label
JEQ/JZ	Label	Z = 1 时, 跳转到 Label
JGE	Label	(N .XOR. V) = 0 时, 跳转到 Label
JL	Label	(N .XOR. V) = 1 时, 跳转到 Label
JMP	Label	无条件跳转到 Label
JN	Label	N = 1 时, 跳转到 Label
JNC/JLO	Label	C = 0 时, 跳转到 Label
JNE/JNZ	Label	Z = 0 时, 跳转到 Label

**注意: 条件跳转与无条件跳转**

条件跳转与无条件跳转指令不影响状态寄存器。

跳转指令发生时, 用偏移量(offset)修改 PC, 即

$$PC_{new} = PC_{old} + 2 + 2 \times offset$$

跳转指令不发生时, 程序继续执行后续指令。

## B.3 不增加 ROM 开销的指令模拟

下述指令可以用 RISC 指令模拟, 而且不增加 ROM 开销。汇编器接受这些助记符, 并插入合适额定内核指令。

**注意: 指令的模拟**

用 R2 和 R3 的内容可以模拟下述指令。R2(CG1)含有立即数 2 和 4; R3(CG2)含有 -1、0FFFFh、0、+1 和 +2。读出数取决于寻址方式, 汇编器根据所用的立即数设置适当的寻址位。

指令助记符	说明	状态位				替代的模拟指令
		V	N	Z	C	
算术指令						
ADC[.W]	目的 C 位加到目的操作数	*	*	*	*	ADDC #0,目的
ADC.B	目的 C 位加到目的操作数	*	*	*	*	ADDC.B #0,目的
DADC[.W]	目的 C 位十进制加到目的操作数	*	*	*	*	DADD #0,目的
DADC.B	目的 C 位十进制加到目的操作数	*	*	*	*	DADD.B #0,目的
DEC[.W]	目的 目的操作数减 1	*	*	*	*	SUB #1,目的
DEC.B	目的 目的操作数减 1	*	*	*	*	SUB.B #1,目的
DEC2[.W]	目的 目的操作数减 2	*	*	*	*	SUB #2,目的
DEC2.B	目的 目的操作数减 2	*	*	*	*	SUB.B #2,目的
INC[.W]	目的 目的操作数加 1	*	*	*	*	ADD #1,目的
INC.B	目的 目的操作数加 1	*	*	*	*	ADD.B #1,目的
INC2[.W]	目的 目的操作数加 2	*	*	*	*	ADD #2,目的
INC2.B	目的 目的操作数加 2	*	*	*	*	ADD.B #2,目的
SBC[.W]	目的 从目的操作数减去 C 位	*	*	*	*	SUBC #0,目的
SBC.B	目的 从目的操作数减去 C 位	*	*	*	*	SUBC.B #0,目的
逻辑指令						
INV[.W]	目的 目的操作数取反	*	*	*	*	XOR #0FFFFh,目的
INV.B	目的 目的操作数取反	*	*	*	*	XOR.B #0FFFFh,目的
RLA[.W]	目的 算术左循环	*	*	*	*	ADD 目的,目的
RLA.B	目的 算术左循环	*	*	*	*	ADD.B 目的,目的
RLC[.W]	目的 经 C 位算术左循环	*	*	*	*	ADDC 目的,目的
RLC.B	目的 经 C 位算术左循环	*	*	*	*	ADDC.B 目的,目的
数据指令(常用)						
CLR[.W]	清除目的操作数	-	-	-	-	MOV #0,目的
CLR.B	清除目的操作数	-	-	-	-	MOV.B #0,目的
CLRC	C 位复位	-	-	-	0	BIC #1,SR
CLRN	N 位复位	-	0	-	-	BIC #4,SR
CLRZ	Z 位复位	-	-	0	-	BIC #2,SR
POP	目的 数据出栈	-	-	-	-	MOV @SP+,目的
SETC	C 位置位	-	-	-	1	BIS #1,SR
SETN	N 位置位	-	1	-	-	BIS #4,SR
SETZ	Z 位置位	-	-	1	-	BIS #2,SR
TST[.W]	目的 目的操作数测试	0	*	*	*	CMP #0,目的
TST.B	目的 目的操作数测试	0	*	*	*	CMP.B #0,目的
程序控制指令						
BR	目的 跳转	-	-	-	-	MOV 目的,PC
DINT	关中断	-	-	-	-	BIC #8,SR
EINT	开中断	-	-	-	-	BIS #8,SR
NOP	空操作	-	-	-	-	MOV #0h, #0h
RET	从子程序返回	-	-	-	-	MOV @SP+,PC

## B.4 指令说明

本节分指令说明全部的内核指令和模拟指令(按字母顺序),并作为解释和应用提示给出一些例子。

带后缀“.W”或无后缀的指令执行字操作。

带后缀“.B”的指令执行字节操作。

* ADC[.W]	进位位加入目的操作数
* ADC.B	进位位加入目的操作数
语法	ADC        目的操作数                    或        ADC.W    目的操作数 ADC.B    目的操作数
操作	目的操作数 + C → 目的操作数
模拟	ADDC        ≠ 0, 目的操作数 ADDC.B    ≠ 0, 目的操作数
说明	将进位位 C 与目的操作数相加。目的操作数原值消失。
状态位	N: 结果为负时置位, 否则复位。 Z: 结果为零时置位, 否则复位。 C: 目的操作数从 0FFFFh 增加至 00000h 时置位, 否则复位; 目的操作数从 0FFh 增加至 000h 时置位, 否则复位。 V: 发生算术溢出时置位, 否则复位。
模式位	OscOff、CPUOff 和 GIE 不受影响。
例 1	R13 所指向的 16 位计数器加入 R12 所指向的 32 位计数器。 ADD        @R13, 0(R12)                    ; 加入低字 ADC        2(R12)                         ; 进位位加入高字
例 2	R13 所指向的 8 位计数器加入 R12 所指向的 16 位计数器。 ADD.B     @R13, 0(R12)                    ; 加入低字节 ADC.B     1(R12)                         ; 进位位加入高字节
ADD[.W]	源操作数加入目的操作数
ADD.B	源操作数加入目的操作数
语法	ADD        源操作数, 目的操作数    或    ADD.W    源操作数, 目的操作数 ADD.B     源操作数, 目的操作数
操作	源操作数 + 目的操作数 → 目的操作数
说明	源操作数加入目的操作数。源操作数不变。目的操作数原值消失。
状态位	N: 结果为负时置位, 否则复位。 Z: 结果为零时置位, 否则复位。 C: 结果产生进位位时置位, 否则复位。 V: 发生算术溢出时置位, 否则复位。
模式位	OscOff、CPUOff 和 GIE 不受影响。
例 1	R5 加 10, 发生进位位时跳转至 TONI。

	ADD	#10, R5	
	JC	TONI	; 有进位位
	...		; 无进位位
例 2	R5 加 10, 发生进位位时跳转至 TONI.		
	ADD.B	#10, R5	; 10 加入 R5 的低字节
	JC	TONI	; 如(R5)≥246, 产生进位位, [0Ah+0F6h]
	...		; 无进位位
<b>ADDC[.W]</b>	<b>源操作数及进位位加入目的操作数</b>		
<b>ADDC.B</b>	<b>源操作数及进位位加入目的操作数</b>		
语法	ADDC	源操作数, 目的操作数	或 ADDC.W 源操作数, 目的操作数
ADDC.B	源操作数, 目的操作数		
操作	源操作数 + 目的操作数 + C → 目的操作数		
说明	源操作数及进位位加入目的操作数。源操作数不变, 目的操作数原值消失。		
状态位	N: 结果为负时置位, 否则复位。		
	Z: 结果为零时置位, 否则复位。		
	C: 结果产生进位位时置位, 否则复位。		
	V: 发生算术溢出时置位, 否则复位。		
模式位	OscOff、CPUOff 和 GIE 不受影响。		
例 1	R13 所指的 32 位计数器加入 R13 加 11 个字所指的 32 位计数器。		
	ADD	@R13+, 20(R13)	; 计数器低字相加, 无进位位
	ADDC	@R13+, 20(R13)	; 计数器高字相加, 同时加入低字相加的进位位
	...		
例 2	R13 所指的 24 位计数器加入 R13 加 11 个字所指的 24 位计数器。		
	ADD.B	@R13+, 10(R13)	; 计数器低字节相加, 无进位位
	ADDC.B	@R13+, 10(R13)	; 计数器中字节相加, 同时加入低字节相加的进位位
	ADDC.B	@R13+, 10(R13)	; 计数器高字节相加, 同时加入中字节相加的进位位
	...		
<b>AND[.W]</b>	<b>源操作数逻辑与目的操作数</b>		
<b>AND.B</b>	<b>源操作数逻辑与目的操作数</b>		
语法	AND	源操作数, 目的操作数	或 AND.W 源操作数, 目的操作数
AND.B	源操作数, 目的操作数		
操作	源操作数 .AND. 目的操作数 → 目的操作数		
说明	源操作数与目的操作数逻辑与, 结果存入目的操作数。		
状态位	N: 最高位为 1 时置位, 否则复位。		
	Z: 结果为零时置位, 否则复位。		
	C: 结果非零时置位, 否则复位。		
	V: 复位。		

模式位	OscOff、CPUOff 和 GIE 不受影响。		
例 1	以 R5 各位为屏蔽码(#0AA55h)和 TOM 相与,结果为零跳转至 TONI。		
	MOV	#0AA55h, R5	; R5 装入屏蔽码
	AND	R5, TOM	; TOM 与屏蔽码相与
	JZ	TONI	;
	...		; 结果非零
			;
			;
			; 或
			;
			;
	AND	#0AA55h, TOM	
	JZ	TONI	
例 2	屏蔽码(#0A5h)和 TOM 的低字节相与,结果为零跳转至 TONI。		
	AND.B	#0A5h, TOM	; 用 0A5h 屏蔽 TOM 的低字节
	JZ	TONI	;
	...		; 结果非零
<b>BIC[.W]</b>	<b>目的操作数的位清除</b>		
<b>BIC.B</b>	<b>目的操作数的位清除</b>		
语法	BIC	源操作数,目的操作数	或 BIC.W 源操作数,目的操作数
	BIC.B	源操作数,目的操作数	
操作说明	.NOT. 源操作数 .AND. 目的操作数 → 目的操作数		
状态位	源操作数反向后和目的操作数逻辑与,结果存入目的操作数。源操作数不变。		
	N: 不变。		
	Z: 不变。		
	C: 不变。		
	V: 不变。		
模式位	OscOff、CPUOff 和 GIE 不受影响。		
例 1	RAM 中字 LEO 的最高 6 位清除。		
	BIC	#0FC00h, LEO	; 清除 LEO 的最高 6 位
例 2	RAM 中字节 LEO 的最高 5 位清除。		
	BIC.B	#0F8h, LEO	; 清除 LEO 的最高 5 位
例 3	端口引脚 P0.0 和 P0.1 的清除。		
	P0OUT	.equ 011h	; 端口 P0 地址定义
	P0-0	.equ 01h	
	P0-1	.equ 02h	
	BIC.B	#P0-0 + P0-1, &P0OUT	; P0.0 和 P0.1 清除为零
<b>BIS[.W]</b>	<b>目的操作数的位置位</b>		
<b>BIS.B</b>	<b>目的操作数的位置位</b>		
语法	BIS	源操作数,目的操作数	或 BIS.W 源操作数,目的操作数
	BIS.B	源操作数,目的操作数	







$SP - 2 \rightarrow SP$   
 $PC \rightarrow @SP$                       PC 保存到栈顶  
 暂存  $\rightarrow PC$                       目的操作数存入 PC  
 说明    在 64 KB 地址空间发生子程序调用。可用所有寻址方式。返回地址(后续指令的地址)保存在堆栈中。CALL 为字指令。  
 状态位    状态位不变。  
 例        例子中列举了所有寻址方式。

CALL    #EXEC                      ; 调用标号 EXEC 或立即数地址(如: #0A4b)  
    ;  $SP - 2 \rightarrow SP, PC + 2 \rightarrow @SP, @PC + \rightarrow PC$   
 CALL    EXEC                      ; 调用 EXEC 中包含的地址  
    ;  $SP - 2 \rightarrow SP, PC + 2 \rightarrow @SP, X(PC) \rightarrow PC$   
    ; 间接寻址  
 CALL    &EXEC                     ; 调用绝对地址 EXEC  
    ;  $SP - 2 \rightarrow SP, PC + 2 \rightarrow @SP, X(0) \rightarrow PC$   
    ; 间接寻址  
 CALL    R5                         ; 调用 R5 中包含的地址  
    ;  $SP - 2 \rightarrow SP, PC + 2 \rightarrow @SP, R5 \rightarrow PC$   
    ; R5 间接寻址  
 CALL    @R5                        ; 调用 R5 所指地址中包含的地址  
    ;  $SP - 2 \rightarrow SP, PC + 2 \rightarrow @SP, @R5 \rightarrow PC$   
    ; 间接的 R5 间接寻址  
 CALL    @R5 +                      ; 调用 R5 所指地址中包含的地址, 并且 R5 增加 2  
    ; 下一次执行这条指令将访问地址表中的下一个地址  
    ;  $SP - 2 \rightarrow SP, PC + 2 \rightarrow @SP, @R5 \rightarrow PC$   
    ; 间接的 R5 间接增量寻址  
 CALL    X(R5)                      ; 调用 R5 + X 所指地址中包含的地址  
    ; (如: 从 X 开始的地址表)  
    ; X 可以是地址或标号  
    ;  $SP - 2 \rightarrow SP, PC + 2 \rightarrow @SP, X(R5) \rightarrow PC$   
    ; 间接的 R5 + X 间接寻址

\* CLR[.W] 清除目的操作数

\* CLR.B 清除目的操作数

语法    CLR            目的操作数            或            CLR.W            目的操作数

         CLR.B        目的操作数

操作    0  $\rightarrow$  目的操作数

模拟    MOV            #0, 目的操作数

         MOV.B        #0, 目的操作数

说明    目的操作数被清除。

状态位    状态位不变。

例 1    将 RAM 字 TONI 清除。

CLR        TONI                      ; 0  $\rightarrow$  TONI

例 2 将寄存器 R5 清除。

```
CLR    R5
```

例 3 将 RAM 字节 TONI 清除。

```
CLR.B  TONI          ; 0 → TONI
```

#### \* CLRC 清除进位位

语法 CLRC

操作 0 → C

模拟 BIC #1, SR

说明 进位位 C 被清除。CLRC 指令是字指令。

状态位 N: 不变。

Z: 不变。

C: 0。

V: 不变。

模式位 OscOff、CPUOff 和 GIE 不受影响。

例 R13 所指 16 位 10 进计数器与 R12 所指 32 位计数器相加。

```
CLRC          ; C=0, 初始化
```

```
DADD @R13, 0(R12) ; 16 位计数器与 32 位计数器的低字相加
```

```
DADC 2(R12)      ; 进位位与 32 位计数器高字相加
```

#### \* CLRN 清除负数位

语法 CLRN

操作 0 → N

模拟 BIC #4, SR

说明 常数 0004h 取反(0FFFBh)且将 SR 逻辑与。结果存入 SR。CLRN 是字指令。

状态位 N: 0。

Z: 不变。

C: 不变。

V: 不变。

模式位 OscOff、CPUOff 和 GIE 不受影响。

例 将 SR 中的 N 位清除, 以回避被调用子程序对负数的特殊处理。

```
CLRN
```

```
CALL SUBR
```

```
...
```

```
...
```

```
SUBR JN SUBRET ; 如果输入为负, 则立即返回主程序
```

```
...
```

```
...
```

```
...
```

```
SUBRET RET
```

#### \* CLRZ 清除零位

语法 CLRZ

操作 0 → Z  
 模拟 BIC #2, SR  
 说明 常数 0002h 取反(0FFFDh)且将 SR 逻辑与。结果存入 SR。CLRZ 是字指令。  
 状态位 N: 不变。  
 Z: 0。  
 C: 不变。  
 V: 不变。

模式位 OscOff、CPUOff 和 GIE 不受影响。

例 清除 SR 中的零位。

CLRZ

**CMP[.W]** 比较源操作数和目的操作数

**CMP.B** 比较源操作数和目的操作数

语法 CMP 源操作数,目的操作数 或 CMP.W 源操作数,目的操作数  
 CMP.B 源操作数,目的操作数

操作 目的操作数 + .NOT.源操作数 + 1 或 目的操作数 - 源操作数

说明 从源操作数中减去目的操作数。这以源操作数的补码相加来实现。两个操作数均不变,结果不保存,只影响状态位。

状态位 N: 结果为负时置位,否则复位(即:源操作数  $\geq$  目的操作数)。

Z: 结果为零时置位,否则复位(即:源操作数 = 目的操作数)。

C: 发生进位时置位,否则复位。

V: 发生算术溢出时置位,否则复位。

模式位 OscOff、CPUOff 和 GIE 不受影响。

例 1 R5 和 R6 比较。如果相等,则程序跳转至 EQUAL。

CMP R5, R6 ; R5 = R6 ?

JEQ EQUAL ; 是,跳转

例 2 两个 RAM 区比较。如果不相等,则程序跳转至 ERROR。

MOV #NUM, R5 ; 比较的字数

L \$1 CMP &BLOCK1, &BLOCK2 ; 字相等 ?

JNZ ERROR ; 否,跳转至 ERROR

DEC R5 ; 是否全部比较 ?

JNZ L \$1 ; 否,进行下一个字的比较

例 3 将用 EDE 和 TONI 寻址的 RAM 字节地行比较。如要相等,则程序跳转至 EQUAL。

CMP.B EDE, TONI ; MEM(EDE) = MEM(TONI) ?

JEQ EQUAL ; 是,跳转

例 4 检查连接于 P0.0 和 P0.1 的两键。如果按键 1,则程序跳转至 MENU1;如果按键 2,则程序跳转至 MENU2。

POIN .EQU 010h

KEY1 .EQU 01h

KEY2 .EQU 02h



和 R4 中为高字)。

CLRC ; 清除进位位  
 DADD R5, R3 ; 低字相加  
 DADD R6, R4 ; 高字及进位位相加  
 JC OVERFLOW ; 发生进位位时跳转到差错处理程序

例 2 RAM 中字节 CNT(2 位 10 进计数器)加 1。

CLRC ; 清除进位位  
 DADD.B #1, CNT ; 10 进计数器加 1  
 或  
 SETC  
 DADD.B #0, CNT ; ≡DADC.B CNT

\* DEC[.W] 目的操作数减 1

\* DEC.B 目的操作数减 1

语法 DEC 目的操作数 或 DEC.W 目的操作数

DEC.B 目的操作数

操作 目的操作数 - 1 → 目的操作数

模拟 SUB #1, 目的操作数

模拟 SUB.B #1, 目的操作数

说明 目的操作数减 1。原值消失。

状态位 N: 结果为负时置位, 否则复位。

Z: 目的操作数初值为 1 时置位, 否则复位。

C: 目的操作数初值为 0 时复位, 否则置位。

V: 发生算术溢出时置位, 否则复位。

目的操作数初值为 08000h 时置位, 否则复位。

目的操作数初值为 080h 时置位, 否则复位。

模式位 OscOff、CPUOff 和 GIE 不受影响。

例 1 R10 减 1。

```

    DEC    R10    ; R10 减 1
; 移动 EDE 开始 255 字节数据块到 TONI
; TONI 不能在 EDE ~ EDE + 0FEh 范围内以防覆盖
    MOV    #EDE, R6
    MOV    #255, R10
L$1  MOV.B  @R6+, TONI - EDE - 1(R6)
    DEC    R10
    JNZ    L$1

```

例 2 LEO 地址的字节减 1。

```

    DEC.B  LEO    ; MEM(LEO)减 1
; 移动 EDE 开始 255 字节数据块到 TONI
; TONI 不能在 EDE ~ EDE + 0FEh 范围内以防覆盖
    MOV    #EDE, R6
    MOV.B  #255, LEO

```

```

L $ 1    MOV.B   (@R6+, TONI - EDE - 1(R6)
          DEC.B   LEO
          JNZ     L $ 1

```

**\* DECD[.W] 目的操作数减 2**

**\* DECD.B 目的操作数减 2**

语法    DECD        目的操作数        或        DECD.W    目的操作数  
          DECD.B    目的操作数

操作    目的操作数 - 2 → 目的操作数

模拟    SUB            #2, 目的操作数

模拟    SUB.B        #2, 目的操作数

说明    目的操作数减 2。原值消失。

状态位   N: 结果为负时置位, 否则复位。

         Z: 目的操作数初值为 2 时置位, 否则复位。

         C: 目的操作数初值为 0 或 1 时复位, 否则置位。

         V: 发生算术溢出时置位, 否则复位。

              目的操作数初值为 08001h 或 08000h 时置位, 否则复位。

              目的操作数初值为 081h 或 080h 时置位, 否则复位。

模式位   OscOff、CPUOff 和 GIE 不受影响。

例 1     R10 减 2。

```

          DECD     R10        ; R10 减去 2
; 移动 EDE 开始 255 字数据块到 TONI
; TONI 不能在 EDE ~ EDE + 1FCh 范围内以防覆盖
          MOV     #EDE, R6
          MOV     #510, R10
L $ 1     MOV     (@R6+, TONI - EDE - 2(R6)
          DEC     R10
          JNZ     L $ 1

```

例 2     LEO 地址的字节减 2。

```

          DECD.B  LEO        ; MEM(LEO)减去 2

```

状态字节 STATUS 减 2。

```

          DECD.B  STATUS

```

**\* DINT 关中断**

语法    DINT

操作    0 → GIE

模拟    BIC    #8, SR

说明    关闭全部中断。

         常数 08h 取反且将 SR 逻辑与, 结果存入 SR。

状态位   N: 不变。

         Z: 不变。

         C: 不变。

V: 不变。

模式位 OscOff、CPUOff 不受影响, GIE 复位。

例 清除通用中断允许位 GIE, 以确保 32 位计数器数据传输不中断。这可避免传输期间因中断使计数器发生改变。

```
DINT                                ; GIE 位控制的所有中断被禁止
MOV     COUNTHI, R5                 ; 复制计数器
MOV     COUNTLO, R6
EINT                                       ; GIE 位控制的所有中断被允许
```

**注意: 关中断**

当执行 DINT 指令时发生中断请求, 关中断指令后续的指令将执行。任何程序段要避免中断影响, DINT 指令的执行必须提前程序段至少一条指令。

**EINT 开中断**

语法 EINT

操作 1 → GIE

模拟 BIS #8, SR

说明 允许全部中断。

常数 08h 和 SR 进行逻辑或, 结果存入 SR。

状态位 N: 不变。

Z: 不变。

C: 不变。

V: 不变。

模式位 OscOff、CPUOff 不受影响。GIE 置位。

例 SR 中的 GIE 位置位。

; P0.2 ~ P0.7 的中断服务程序

; 中断处于系统的最低级

; P0IN 是端口读寄存器地址, P0IFG 是中断事件锁存寄存器地址

;

```
PUSH.B    &P0IN
```

```
BIC.B     @SP, &P0IFG    ; 复位已被接受的中断标志
```

; 原有 P0 中断标志保存在堆栈

```
EINT                                           ; 允许其他中断
```

```
BIT       #Mask, @SP
```

```
JEQ       MaskOK        ; 由 MASK 识别标志, 跳转
```

...

```
MaskOK    BIC           #Mask, @SP
```

...

```
INCD     SP                ; 将 SP 调整到中断程序开始位置
```

```
RETI
```

**注意：开中断**

EINT 指令的后续指令总是要执行的,甚至有中断服务请求正在挂起。

\* **INC[.W]** 目的操作数加 1\* **INC.B** 目的操作数加 1

语法 INC 目的操作数 或 INC.W 目的操作数  
INC.B 目的操作数

操作 目的操作数 + 1 → 目的操作数

模拟 ADD #1, 目的操作数

说明 目的操作数增加 1, 原值消失。

状态位 N: 结果为负时置位, 否则复位。

Z: 目的操作数初值为 0FFFFh 时置位, 否则复位。

目的操作数初值为 0FFh 时置位, 否则复位。

C: 目的操作数初值为 0FFFFh 时置位, 否则复位。

目的操作数初值为 0FFh 时置位, 否则复位。

V: 目的操作数初值为 07FFFh 时置位, 否则复位。

目的操作数初值为 07Fh 时置位, 否则复位。

模式位 OscOff、CPUOff 和 GIE 不受影响。

例 1 取消软件堆栈(非系统堆栈)顶的字节数据。

```
SSP      .EQU    R4
          INC     SSP      ; 用增加指令取消栈顶项 TOSS (Top of SW Stack)
          ; SSP 是字寄存器, 不能用 INC.B
```

例 2 处理过程的状态字节 STATUS 增加。增至 11 时跳转至 OVFL。

```
INC.B    STATUS
CMP.B    #11, STATUS
JEQ      OVFL
```

\* **INCD[.W]** 目的操作数加 2\* **INCD.B** 目的操作数加 2

语法 INCD 目的操作数 或 INCD.W 目的操作数  
INCD.B 目的操作数

操作 目的操作数 + 2 → 目的操作数

模拟 ADD #2, 目的操作数

模拟 ADD.B #2, 目的操作数

说明 目的操作数增加 2。原值消失。

状态位 N: 结果为负时置位, 否则复位。

Z: 目的操作数初值为 0FFFEh 时置位, 否则复位。

目的操作数初值为 0FEh 时置位, 否则复位。

C: 目的操作数初值为 0FFFEh 或 0FFFFh 时置位, 否则复位。

目的操作数初值为 0FEh 或 0FFh 时置位, 否则复位。



V: 目的操作数初值为 07FFEh 或 07FFFh 时置位, 否则复位。

目的操作数初值为 07Eh 或 07Fh 时置位, 否则复位。

模式位 OscOff、CPUOff 和 GIE 不受影响。

例 1 不占用寄存器而取消栈顶项。

PUSH R5 ; R5 是计算结果, 保存在系统堆栈中

...

INCD SP ; 用 SP 增加 2 来取消栈顶项 TOS

; SP 是字寄存器, 不能用 INCD.B

RET

例 2 栈顶字节增加 2。

INCD.B 0(SP) ; TOS 的字节增加 2

\* INV[.W] 目的操作数取反

\* INV.B 目的操作数取反

语法 INV 目的操作数

INV.B 目的操作数

操作 .NOT. 目的操作数 → 目的操作数

模拟 XOR #0FFFFh, 目的操作数

模拟 XOR.B #0FFh, 目的操作数

说明 目的操作数取反。原值消失。

状态位 N: 结果为负时置位, 否则复位。

Z: 目的操作数初值为 0FFFFh 时置位, 否则复位。

目的操作数初值为 0FFh 时置位, 否则复位。

C: 结果非零时置位, 否则复位。

V: 目的操作数初值为负时置位, 否则复位。

模式位 OscOff、CPUOff 和 GIE 不受影响。

例 1 R5 取补(2 的补码)。

MOV #00AEh, R5 ; R5 = 000AEh

INV R5 ; R5 = 0FF51h

INC R5 ; R5 = 0FF52h, 原值的补码

例 2 存储器字节 LEO 取补。

MOV.B #0AEh, LEO ; MEM(LEO) = 0AEh

INV.B LEO ; MEM(LEO) = 051h

INC.B LEO ; MEM(LEO) = 052h, 原值的补码

JC 有进位位时跳转

JHS 无符号数大于等于时跳转

语法 JC label

JHS label

操作 如果 C = 1: PC + 2 × 偏移量 → PC

如果 C = 0: 继续执行后续指令

说明 检查进位位。如果 C=1, 则指令低 10 位所含带符号偏移量乘 2 后加入 PC。如



**JL 有符号数小于时跳转**

语法 JL label

操作 如果  $(N \oplus V) = 1$ :  $PC + 2 \times \text{偏移量} \rightarrow PC$

如果  $(N \oplus V) = 0$ : 继续执行后续指令

说明 检查负数位 N 和溢出位 V。如果只有一个置位,则指令低 10 位所含带符号偏移量乘 2 后加入 PC。如果 N、V 同时置位或复位,则执行跳转指令的后续指令。JL 指令用于有符号整数的比较。

状态位 状态位不受影响。

例 当 R6 所含数据小于 R7 所指存储器地址的数据时,程序跳转至 EDF。

```
CMP    @R7, R6          ; R6 < (R7)?, 有符号数比较
```

```
JL     EDF              ; 是, R6 < (R7)
```

```
...                               ; 否, 继续执行
```

```
...
```

**JMP 无条件跳转**

语法 JMP label

操作  $PC + 2 \times \text{偏移量} \rightarrow PC$

说明 指令低 10 位所含带符号偏移量乘 2 后加入 PC。

状态位 状态位不受影响。

提示 可以用这一单字指令取代 BR 指令,用于实现相对于 PC 在  $-511 \sim +512$  个字的范围内的程序跳转。

**JN 负数时跳转**

语法 JN label

操作 如果  $N = 1$ :  $PC + 2 \times \text{偏移量} \rightarrow PC$

如果  $N = 0$ : 继续执行后续指令

说明 检查负数位。如果  $N = 1$ ,则指令低 10 位所含带符号偏移量乘 2 后加入 PC。如果  $N = 0$ ,则执行跳转指令的后续指令。

状态位 状态位不受影响。

例 COUNT 与 R5 相减。如果结果为负数,则程序跳转至 L\$1 并清除 COUNT。

```
SUB    R5, COUNT        ; COUNT - R5 → COUNT
```

```
JN     L$1              ; 如果 COUNT < 0, 则 PC=L$1, COUNT 清除
```

```
...                               ; COUNT ≥ 0
```

```
...
```

```
...
```

```
...
```

```
L$1    CLR    COUNT
```

```
...
```

```
...
```

```
...
```

**JNC 无进位位时跳转****JLO 无符号数小于时跳转**

语法	JNC     label JLO     label
操作	如果 C = 0: PC + 2 × 偏移量 → PC 如果 C = 1: 继续执行后续指令
说明	检查进位位。如果 C=0, 则指令低 10 位所含带符号偏移量乘 2 后加入 PC。如果 C = 1, 则执行跳转指令的后续指令。JNC/JLO 指令用于无符号数(0~65 535)的比较。
状态位	状态位不受影响。
例 1	R6 与 BUFFER 相加。如果发生溢出, 则跳转至 ERROR 执行差错处理程序。 <pre> ADD      R6, BUFFER      ; BUFFER + R6 → BUFFER JNC      CONT            ; C=0, 跳转至 CONT ERROR    ...              ; C=1, 差错处理程序 ... ... ... CONT     ...              ; 正常程序继续执行 ... ... </pre>
例 2	如果状态字节含有 1 或 0, 则跳转至 STL2。 <pre> CMP.B    #2, STATUS JLO      STL2            ; STATUS &lt; 2 ... ...                    ; STATUS ≥ 2, 继续执行 </pre>
<b>JNE</b>	<b>不等时跳转</b>
<b>JNZ</b>	<b>非零时跳转</b>
语法	JNE     label JNZ     label
操作	如果 Z = 0: PC + 2 × 偏移量 → PC 如果 Z = 1: 继续执行后续指令
说明	检查零位。如果 Z=0, 则指令低 10 位所含带符号偏移量乘 2 后加入 PC。如果 Z=1, 执行跳转指令的后续指令。
状态位	状态位不受影响。
例	R7 与 R8 不同时跳转至 TONI。 <pre> CMP      R7, R8          ; 比较 R7 与 R8 JNE      TONI            ; 若不同, 则跳转 ... ...                    ; 若相同, 则继续执行 </pre>
<b>MOV[.W]</b>	<b>移动源操作数到目的操作数</b>
<b>MOV.B</b>	<b>移动源操作数到目的操作数</b>
语法	MOV     源操作数, 目的操作数   或   MOV.W 源操作数, 目的操作数 MOV.B 源操作数, 目的操作数
操作	源操作数 → 目的操作数

说明 源操作数移动到目的操作数。源操作数不变,目的操作数的原值消失。

状态位 状态位不受影响。

模式位 OscOff、CPUOff 和 GIE 不受影响。

例 1 表 EDE 的字数据复制到表 TOM。表长度为 020h。

```

MOV      #EDE, R10          ; 准备指针
MOV      #020h, R9         ; 准备计数器
Loop     MOV      @R10+, TOM - EDE - 2(R10) ; R10 作为两表的指针
DEC      R9                ; 计数器减 1
JNZ     Loop               ; 计数器 ≠ 0, 继续复制
...
...
...

```

例 2 表 EDE 所含字节数据复制到表 TOM。表长度为 020h。

```

MOV      #EDE, R10          ; 准备指针
MOV      #020h, R9         ; 准备计数器
Loop     MOV.B    @R10+, TOM - EDE - 1(R10) ; R10 作为两表的指针
DEC      R9                ; 计数器减 1
JNZ     Loop               ; 计数器 ≠ 0, 继续复制
...
...

```

#### \* NOP 空操作

语法 NOP

操作 无操作

模拟 MOV #0, #0

说明 不执行操作。NOP 指令用于软件调试期间消除指令,或定义等待时间。

NOP 指令主要用于两个目的:

- 占据 1、2 或 3 个存储器字;
- 调整软件时序。

状态位 状态位不受影响。

#### 注意: 其他可以模拟空操作的指令

还有其他一些指令可以用不同的周期和代码字空间来模拟空操作。

```

例:  MOV    0(R4), 0(R4)    ; 6 个周期,   3 个字
      MOV    @R4, 0(R4)    ; 5 个周期,   2 个字
      BIC    #0, EDE(R4)   ; 4 个周期,   2 个字
      JMP    $ + 2         ; 2 个周期,   1 个字
      BIC    #0, R5        ; 1 个周期,   1 个字

```

\* POP[.W] 从堆栈推出字到目的操作数

\* POP.B 从堆栈推出字节到目的操作数

语法 POP 目的操作数

	POP.B	目的操作数
操作	@SP →	目的操作数
	SP + 2	→ SP
模拟	MOV	@SP + , 目的操作数 或 MOV.W @SP + , 目的操作数
模拟	MOV.B	@SP + , 目的操作数
说明	将栈顶项移动到目的操作数。SP 增加 2。	
状态位	状态位不受影响。	
例 1	R7 和 SR 从堆栈恢复。	
	POP	R7 ; 恢复 R7
	POP	SR ; 恢复 SR
例 2	从堆栈恢复 RAM 字节 LEO 的数据。	
	POP.B	LEO ; 将栈顶项的低字节移动到 LEO。
例 3	R7 从堆栈恢复。	
	POP.B	R7 ; 将栈顶项的低字节移动到 R7, R7 额定高字节为 00。
例 4	R7 所指存储器及 SR 从堆栈恢复。	
	POP.B	0(R7) ; 栈顶项的低字节移动到 R7 所指地址
		; 例: R7 = 203h, Mem(R7) = 栈顶项的低字节
		; 例: R7 = 20Ah, Mem(R7) = 栈顶项的低字节
	POP	SR

**注意: 系统堆栈指针**

系统堆栈指针增加时总是 2, 而与指令的字节后缀无关。这样做的原因是因为 SP 不仅用于 POP 指令, 也用于 RETI 指令。

**PUSH[.W] 字压入堆栈****PUSH.B 字节压入堆栈**

语法 PUSH 源操作数 或 PUSH.W 源操作数

PUSH.B 源操作数

操作 SP - 2 → SP

源操作数 → @SP

说明 SP 减少 2, 源操作数移动到 SP 所指 RAM 字地址(TOS)。

状态位 状态位不受影响。

模式位 OscOff、CPUOff 和 GIE 不受影响。

例 1 SR 及 R8 保存入堆栈。

PUSH SR ; 保存 SR

PUSH R8 ; 保存 R8

例 2 外围模块寄存器 TCDAT 的内容保存入堆栈。

PUSH.B &TCDAT ; 保存 8 位外围模块地址 TCDAT 到堆栈

**注意：系统堆栈指针**

系统堆栈指针减少时总是 2, 而与指令的字节后缀无关。这样做的原因是因为 SP 不仅用于 PUSH 指令, 也用于中断服务程序。

**\* RET 从子程序返回**

语法 RET

操作 @SP → PC

SP + 2 → SP

模拟 MOV @SP+, PC

说明 将由 CALL 指令压入堆栈的返回地址移动到 PC。程序从 CALL 指令的后续地址继续执行。

状态位 状态位不受影响。

**RETI 从中断服务程序返回**

语法 RETI

操作 TOS → SR

SP + 2 → SP

TOS → PC

SP + 2 → SP

说明 1. 用栈顶项取代 SR, 使它恢复为中断开始时的数值。SP 增加 2。  
2. 用栈顶项取代 PC, 使 PC 恢复成中断开始时的地址。SP 增加 2。

状态位 N: 从系统堆栈恢复。

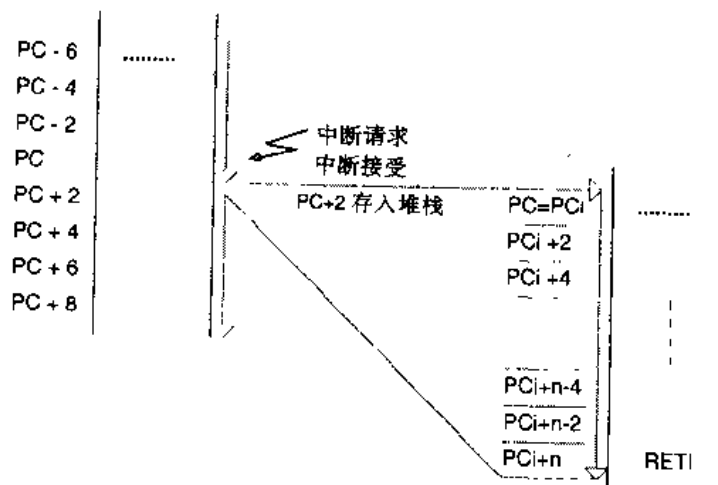
Z: 从系统堆栈恢复。

C: 从系统堆栈恢复。

V: 从系统堆栈恢复。

模式位 OscOff、CPUOff 和 GIE 从系统堆栈恢复。

例 主程序被中断。



\* **RLA[.W]** 算术左移\* **RLA.B** 算术左移

语法 RLA 目的操作数 或 RLA.W 目的操作数  
RLA.B 目的操作数

操作  $C \leftarrow MSB \leftarrow MSB - 1 \leftarrow \dots \leftarrow LSB + 1 \leftarrow LSB \leftarrow 0$

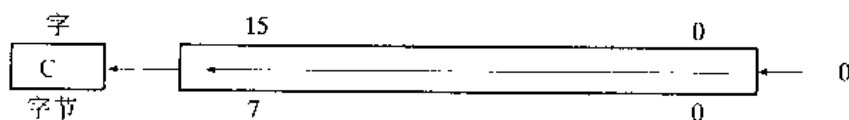
模拟 ADD 目的操作数,目的操作数

ADD.B 目的操作数,目的操作数

说明 目的操作数左移一位。MSB 移入进位位, LSB 填 0。RLA 指令相当于有符号数乘 2。

当  $04000h \leq$  目的操作数初值  $< 0C000h$  时, 发生溢出, 因为结果符号发生改变。

当  $040h \leq$  目的操作数初值  $< 0C0h$  时, 发生溢出, 因为结果符号发生改变。



状态位 N: 结果为负数时置位, 否则复位。

Z: 结果为零时置位, 否则复位。

C: 装入 MSB 内容。

V: 若发生算术溢出, 即

$04000h \leq$  目的操作数初值  $< 0C000h$

或  $040h \leq$  目的操作数初值  $< 0C0h$

则置位, 否则复位。

模式位 OscOff、CPUOff 和 GIE 不受影响。

例 1 R7 乘 4。

RLA R7 ; R7 左移 ( $\times 2$ ), 模拟指令 ADD R7, R7

RLA R7 ; R7 左移 ( $\times 4$ ), 模拟指令 ADD R7, R7

例 2 R7 低字节乘 4。

RLA.B R7 ; R7 低字节左移 ( $\times 2$ ), 模拟指令 ADD.B R7, R7

RLA.B R7 ; R7 低字节左移 ( $\times 4$ ), 模拟指令 ADD.B R7, R7

**注意: RLA 替代**

汇编器不能识别指令:

RLA @R5+ 或 RLA.B @R5+

必须用以下指令替代, 即

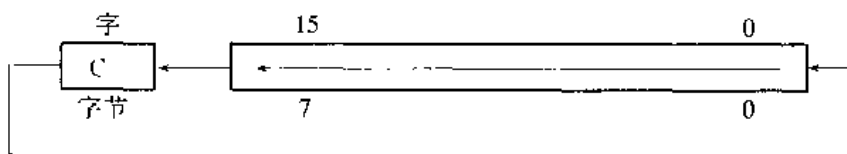
ADD @R5+, -2(R5) 或 ADD.B @R5+, -1(R5)

\* **RLC[.W]** 经进位位左移\* **RLC.B** 经进位位左移

语法 RLC 目的操作数 或 RLC.W 目的操作数  
RLC.B 目的操作数



操作  $C \leftarrow MSB \leftarrow MSB-1 \leftarrow \dots \leftarrow LSB+1 \leftarrow LSB \leftarrow C$   
 模拟 ADDC 目的操作数,目的操作数  
 说明 目的操作数左移一位。进位位移入 LSB, MSB 移入进位位。



状态位 N: 结果为负数时置位, 否则复位。  
 Z: 结果为零时置位, 否则复位。  
 C: 装入 MSB 内容。  
 V: 若发生算术溢出, 即  
 $04000h \leq \text{目的操作数初值} < 0C000h$   
 或  $040h \leq \text{目的操作数初值} < 0C0h$   
 则置位, 否则复位。

模式位 OscOff、CPUOff 和 GIE 不受影响。

例 1 R5 左移一位。

RLC R5 ;  $(R5 \times 2) + C \rightarrow R5$

例 2 POIN.1 信息移入 R5 的 LSB。

BIT.B #2, &POIN ; POIN.1 的信息  $\rightarrow$  进位位

RLC R5 ; 进位位 = POIN.1  $\rightarrow$  R5 的 LSB

例 3 MEM(LEO)的内容左移一位。

RLC.B LEO ;  $MEM(LEO) \times 2 + C \rightarrow MEM(LEO)$

例 4 POIN.1 的信息移入 R5 的 LSB。

BIT.B #2, &POIN ; POIN.1 的信息  $\rightarrow$  进位位

RLC.B R5 ; 进位位 = POIN.1  $\rightarrow$  R5 的 LSB, R5 的高字节复位

**注意: RLC 和 RLC.B 的模拟**

汇编器不能识别指令 RLC @R5+。必须用指令 ADDC @R5+, -2(R5) 替代。

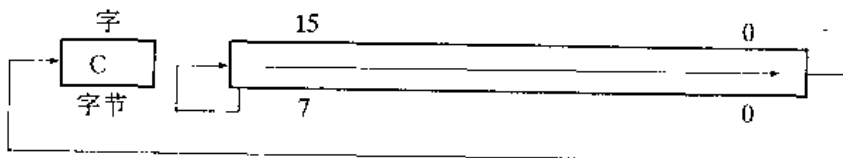
## RRA[.W] 算术右移

### RRA.B 算术右移

语法 RRA 目的操作数 或 RRA.W 目的操作数  
 RRA.B 目的操作数

操作  $MSB \rightarrow MSB, MSB \rightarrow MSB-1, \dots, LSB+1 \rightarrow LSB, LSB \rightarrow C$

说明 目的操作数右移一位。MSB 保持, 从 MSB 开始各位右移, LSB 移入进位位。



状态位 N: 结果为负数时置位, 否则复位。

Z: 结果为零时置位, 否则复位。

C: 装入 LSB 内容。

V: 复位

模式位 OscOff、CPUOff 和 GIE 不受影响。

例 1 R5 右移一位。MSB 保持原值。操作相当于算术除以 2。

```
RRA    R5                ; R5/2 → R5
; R5 的值 × 0.75 (0.5 + 0.25)
PUSH   R5                ; 暂存 R5 入堆栈
RRA    R5                ; R5 × 0.5 → R5
ADD    @SP+, R5         ; R5 × 0.5 + R5 = 1.5 × R5 → R5
RRA    R5                ; (1.5 × R5) × 0.5 = 0.75 × R5 → R5
; 或
RRA    R5                ; R5 × 0.5 → R5
PUSH   R5                ; R5 × 0.5 → TOS
RRA    @SP              ; TOS × 0.5 = 0.5 × R5 × 0.5 = 0.25 × R5 → TOS
ADD    @SP+, R5         ; R5 × 0.5 + R5 × 0.25 = 0.75 × R5 → R5
```

例 2 R5 低字节右移一位。MSB 保持原值。操作相当于算术除以 2。

```
RRA.B  R5                ; R5/2 → R5: 只对低字节操作, R5 高字节为 0
; R5 低字节的值 × 0.75 (0.5 + 0.25)
PUSH.B R5                ; 暂存 R5 低字节入堆栈
RRA.B  R5                ; R5 × 0.5 → R5
ADD.B  @SP+, R5         ; R5 × 0.5 + R5 = 1.5 × R5 → R5
RRA.B  R5                ; (1.5 × R5) × 0.5 = 0.75 × R5 → R5
; 或
RRA.B  R5                ; R5 × 0.5 → R5
PUSH.B R5                ; R5 × 0.5 → TOS
RRA.B  @SP              ; TOS × 0.5 = 0.5 × R5 × 0.5 = 0.25 × R5 → TOS
ADD.B  @SP+, R5         ; R5 × 0.5 + R5 × 0.25 = 0.75 × R5 → R5
```

**RRC[, W]** 经进位位右移

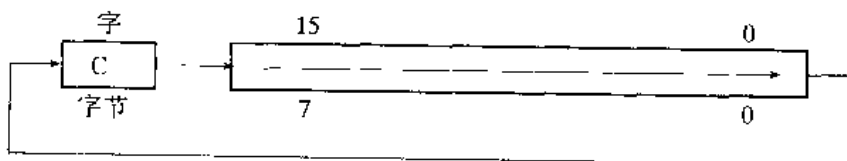
**RRC.B** 经进位位右移

语法 RRC 目的操作数 或 RRC.W 目的操作数

RRC 目的操作数

操作  $C \rightarrow \text{MSB} \rightarrow \text{MSB} - 1 \cdots \text{LSB} + 1 \rightarrow \text{LSB} \rightarrow C$

说明 目的操作数右移一位。进位位移入 MSB, LSB 移入进位位。



状态位 N: 结果为负数时置位, 否则复位。



操作  $1 \rightarrow C$   
 模拟 BIS #1, SR  
 说明 进位位置位。  
 状态位 N: 不变。  
       Z: 不变。  
       C: 置位。  
       V: 不变。  
 模式位 OscOff、CPUOff 和 GIE 不受影响。  
 例 模拟 10 进制减法。

R6 - R5 (10 进制)

设: R5 = 3 987, R6 = 4 137

```

DSUB  ADD      #6666h, R5 ; R5 的内容从 0~9 转换成 06h~0Fh 表示
                                ; R5 = 03987 + 6666 = 09FEDh
                                ; 取反, 结果为 0~9 表示
                                ; R5 = .NOT. R5 = 06012h
                                ; 准备: 进位位 = 1
                                ; 以加法模拟减法;
                                ; (10000 - R5 - 1)
                                ; R6 = R6 + R5 + 1
                                ; R6 = 4137 + 06012 + 1 = 1 0150 = 0150

```

\* SETN 负数位置位

语法 SETN  
 操作  $1 \rightarrow N$   
 模拟 BIS #4, SR  
 说明 负数位置位。  
 状态位 N: 置位。  
       Z: 不变。  
       C: 不变。  
       V: 不变。

模式位 OscOff、CPUOff 和 GIE 不受影响。

\* SETZ 零位置位

语法 SETZ  
 操作  $1 \rightarrow Z$   
 模拟 BIS #2, SR  
 说明 零位置位。  
 状态位 N: 不变。  
       Z: 置位。  
       C: 不变。  
       V: 不变。

模式位 OscOff、CPUOff 和 GIE 不受影响。

SUB[.W] 目的操作数减源操作数

SUB.B 目的操作数减源操作数

语法 SUB 源操作数,目的操作数 或 SUB.W 源操作数,目的操作数

SUB.B 源操作数,目的操作数

操作 目的操作数 + .NOT.源操作数 + 1 → 目的操作数

或

目的操作数 - 源操作数 → 目的操作数

说明 从目的操作数中减去源操作数。这可用加上源操作数取反加 1 来实现。源操作数不变。目的操作数的原值消失。

状态位 N: 结果为负数时置位, 否则复位。

Z: 结果为零时置位, 否则复位。

C: 当 C=1 时置位, 否则复位; 或者当不发生借位时置位, 否则复位。

V: 发生算术溢出时置位, 否则复位。

模式位 OscOff、CPUOff 和 GIE 不受影响。

例 1 见 SBC 指令的例子。

例 2 见 SBC.B 指令的例子。

注意: 借位相当于进位位的非

借位相当于进位位取反。

借位

进位位

Yes

0

No

1

SUBC[.W] SBB[.W] 目的操作数减源操作数及借位(进位位的非)

SUBC.B SBB.B 目的操作数减源操作数及借位(进位位的非)

语法 SUBC 源操作数,目的操作数 或

SUBC.W 源操作数,目的操作数 或

SBB 源操作数,目的操作数 或

SBB.W 源操作数,目的操作数 或

SUBC.B 源操作数,目的操作数 或

SBB.B 源操作数,目的操作数

操作 目的操作数 或 .NOT.源操作数 + C → 目的操作数

目的操作数 - 源操作数 - 1 + C → 目的操作数

说明 从目的操作数中减去源操作数。这可以加上源操作数取反及进位位来实现。源操作数不变。

目的操作数原值消失。

状态位 N: 结果为负数时置位, 否则复位。

Z: 结果为零时置位, 否则复位。

C: 当 C=1 时置位, 否则复位; 或者当不发生借位时置位, 否则复位。

V: 发生算术溢出时置位, 否则复位。

模式位 OscOff、CPUOff 和 GIE 不受影响。

例 1 两个浮点数的 24 位尾数相减。低字在 R13 和 R10 中,高字节在 R12 和 R9 中。

SUB.W R13, R10 ; 低 16 位减

SUBC.B R12, R9 ; 高 8 位减

例 2 R10 与 R11(高字节)的 16 位计数器减去 R13 所指 16 位计数器。

SUB.B @R13-, R10 ; 低字节减

SUBC.B @R13, R11 ; 高字节减,带进位位

注意:借位相当于进位位的非

借位相当于进位位取反。

借位	进位位
Yes	0
No	1

### SWPB 字节交换

语法 SWPB 目的操作数

操作 位 15 ~ 8 ↔ 位 7 ~ 0

说明 目的操作数的高低字节交换。

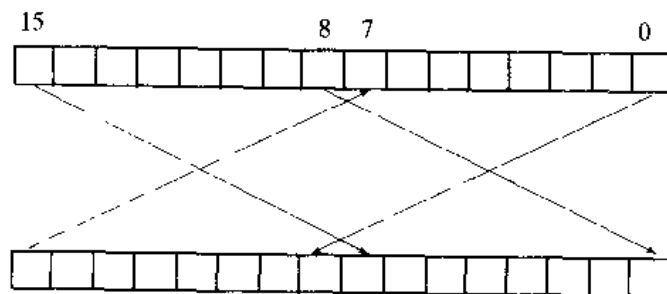
状态位 N: 不变。

Z: 不变。

C: 不变。

V: 不变。

模式位 OscOff、CPUOff 和 GIE 不受影响。



例 1 MOV #040BFh, R7 ; R7 = 0100000010111111

SWPB R7 ; R7 = 1011111101000000

例 2 R5 × 256, 结果存入 R5、R4。

SWPB R5 ;

MOV R5, R4 ; 字节交换后的值复制到 R4

BIC #0FF00h, R5 ; 修正结果

BIC #00FFh, R4 ; 修正结果

### SXT 符号扩展

语法 SXT 目的操作数

操作 位 7 → 位 8 → … → 位 15

说明 低字节的符号扩展到高字节。







	JMP L \$ 1	
DSUB 源操作数,目的操作数	ADD #6666h,源操作数	; 10 进制减法
	INV 源操作数	; 源操作数被改变!
	SETC	
	DADD 源操作数,目的操作数	; 源操作数 - 目的操作数
		; (10 进制)
NEG 目的操作数	INV 目的操作数	; 求目的操作数的相反数
	INC 目的操作数	
RL 目的操作数	ADD 目的操作数,目的操作数	; 循环左移
	ADDC #0,目的操作数	
RR 目的操作数	CLRC	; 循环右移
	RRC 目的操作数	
	JNC L \$ 1	
	BIS #8000h,目的操作数	
L \$ 1	...	

## 附录 C EPROM 编程

本附录描述 MSP430 的 EPROM 模块。EPROM 可由紫外光擦除和电编程。含有 EPROM 模块的 MSP430 器件提供多次编程的窗口封装或一次性编程的 OTP 封装。

### C.1 EPROM 操作

CPU 可从 EPROM 读取数据和指令。当 TDI/VPP 引脚加上编程电压后, CPU 也可向 EPROM 模块写入。读 EPROM 和读其他内部外围模块一样。编程和读取可以在字节或字边界进行。

#### 1. 擦除

编程前应先擦除整个 EPROM, 只要让透明窗受到紫外光的照射即可。

#### 注意: 环境光对 EPROM 的影响

普通环境光也含擦除 EPROM 所需波长。对有窗器件编程时, 窗口必须用不透光的封条封住。紫外光照射 EPROM 时, 如果 EEPROM 模块也在同一芯片上, 则它也会被擦除。EEPROM 中的有用数据应该重新编程。

数据可以通过内部集成的“JTAG”特性串行编程, 或通过应用软件的一部分编程。“JTAG”实现了内部的安全机制。一旦“安全熔丝”激活, 由于“JTAG”的作用便不能访问器件。被“JTAG”被永久性地切换到旁路模式。

#### 2. 编程

编程时必须在 TDI/VPP 引脚外加电压, 为编程提供所需的电压和电流。在器件手册的电特性中标明了编程所需的最短时间。

加了外部电压后, 由 EPROM 控制寄存器 EPCTL 控制 EPROM 的编程。EPROM 经擦除后各位为“1”, 编程后对应的位将变为“0”。

EPROM 模块的编程操作既可以按字节或字(见图

MSP 430 字格式编程

...
xxxAh DEF0
xxx8h 9ABC
xxx6h 5 6 7 8
xxx4h 1 2 3 4
...

MSP 430 字节格式编程

...
xxxB h DE
xxxA h F0
xxx9 h 9A
xxx8 h BC
xxx7 h 56
xxx6 h 78
xxx5 h 12
xxx4 h 34
...

图 C.1 MSP430 字和字节格式编程

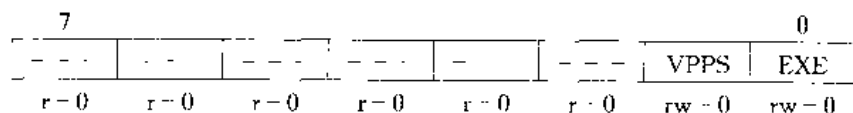
C.1), 也可按不同长度的数据块, 甚至是整个模块。所有的位在编程前必需先擦除, 使它们最

终都为“1”。编程操作可对单一器件进行,甚至可在器件安装在系统时进行。提供的外部编程电压应该在器件手册所要求的范围内。“JTAG”引脚的电平在器件手册中定义,通常是 CMOS 电平。

### 3. EPROM 控制寄存器 EPCTL

EPCTL 的位定义如下:

EPCTL (054h)



对 EPCTL 各位说明如下:

位 0: 执行位 EXE 控制初始化和终止对 EPROM 模块的编程。在 EXE 位置位前必须在 TDI/VPP 引脚加外部电压。定时条件在器件手册中说明。

位 1: VPPS 位置位,使外部编程电压与 EPROM 模块相连。VPPS 位应在 EXE 位置位前位置。它可与 EXE 位一起复位。在编程操作时 VPPS 位不能复位。

### 4. EPROM 保护

当“安全熔丝”激活时,通过串行测试和编程接口“JTAG”对 EPROM 的访问被禁止。激活“安全熔丝”是通过串行指令移入“JTAG”实现的。激活“安全熔丝”是不可逆的,任何对内部系统的访问都被中断。这时遵循标准 IEEE1149.1 的旁路功能有效。

## C.2 快速编程算法

快速编程模式(见图 C.2)通常用于向 EPROM 写入数据。已编程为逻辑“0”只能通过紫外光擦除。

快速编程使用两种类型脉冲:初期脉冲和最终脉冲。典型的初期脉冲宽度为  $100 \mu\text{s}$  (见最近的器件手册)。每次基本脉冲后对编程数据作校验。如果连续失败 25 次,则此次编程操作失效。如果读取了正确的数据,那么使用最终脉冲编程,它是已用初期脉冲数的 3 倍。

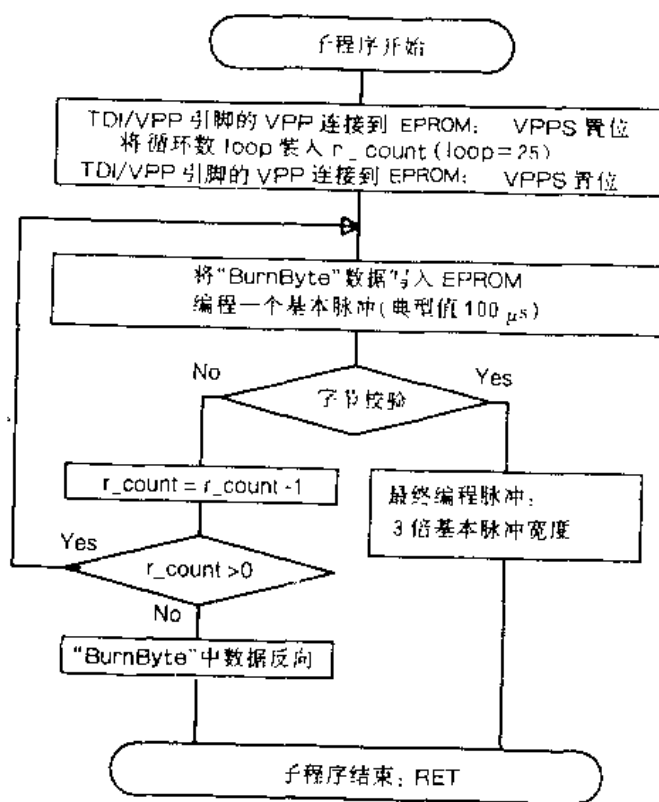


图 C.2 快速编程算法流程图

### C.3 通过串行数据链路应用“JTAG”特性的 EPROM 模块编程

“JTAG”的硬件连接(见图 C.3)是通过 4 个引脚外加 GND 或 VSS 实现的。4 个引脚即 TMS、TCK、TDI(VPP)、TDO/TDI。

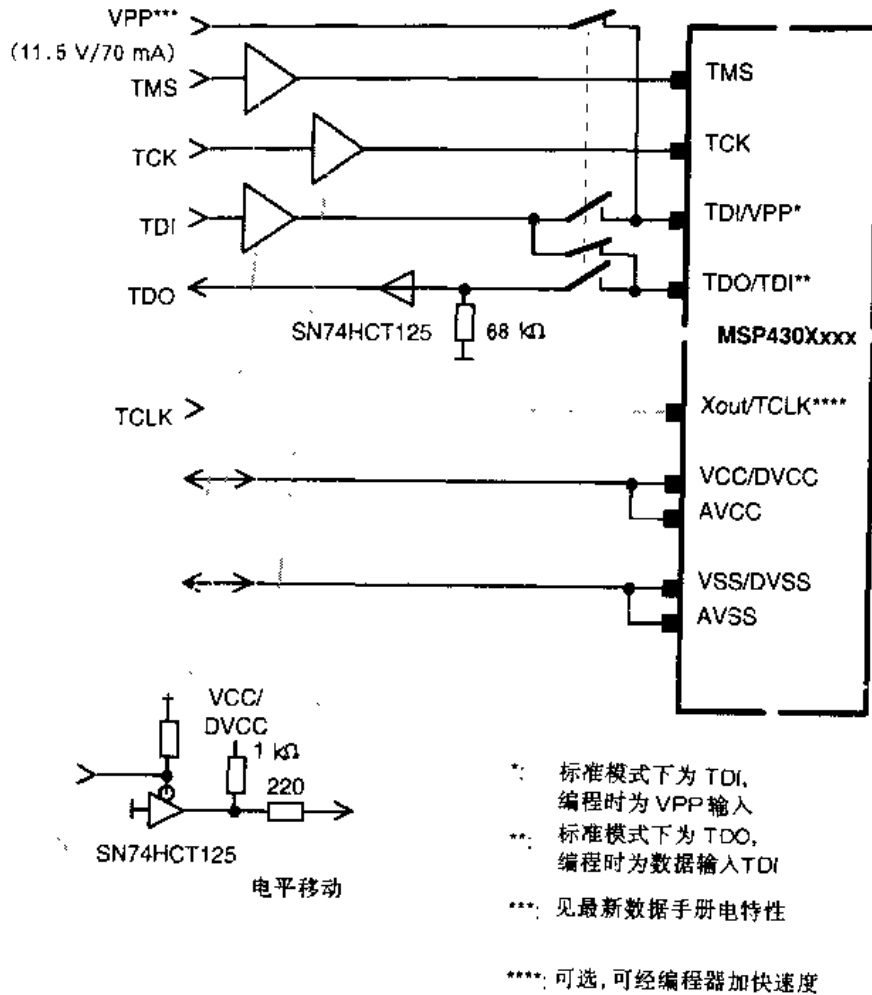
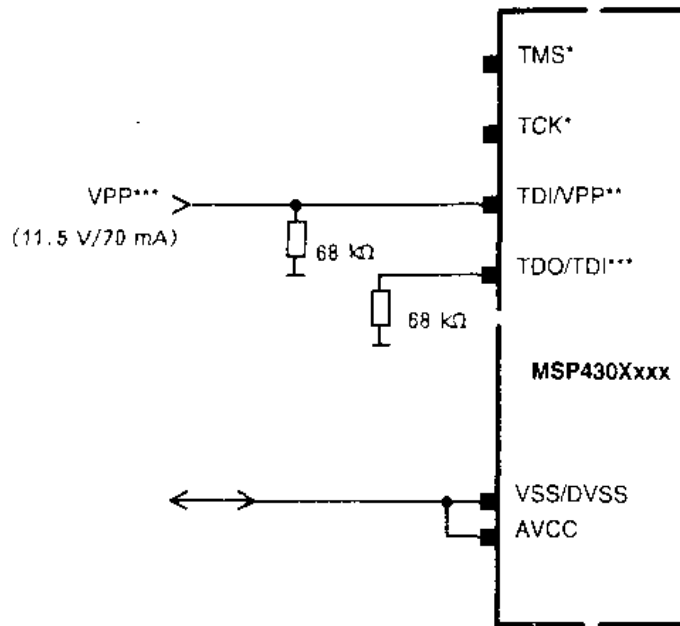


图 C.3 “JTAG”的硬件连接

### C.4 通过微控制器软件实现对 EPROM 模块编程

对 EPROM 模块编程所要求的硬件很简单,即 TDI/VPP 引脚加编程电压(见图 C.4),并运行适当的软件即可。控制 EPROM 编程的软件不能在要编程的 EPROM 模块上运行,因为不可能同时从 EPROM 读指令和往 EPROM 写数据。软件需要在另一个存储器上运行,例如 ROM、RAM 或其他的 EPROM 模块。



- TMS 和 TCK 引脚有内部上拉电阻
- MSP430 的 ROM 型在 TDI/VPP 引脚有上拉电阻, 而 MSP430Pxxx 和 MSP430Exxx 型没有。为防止输入引脚的浮空, 要外加一个上拉或下拉电阻。
- 为防止 TDO/TDI 在用 TDI 功能时输入引脚的浮空, 要外加一个上拉或下拉电阻。

图 C.4 对 EPROM 模块编程所要求的硬件结构原理图

以下软件是使用控制器的软件对 EPROM 编程的例子。该实例是用快速编程算法向 EPROM 写一个字节(见图 C.5)。代码写入与位置无关, 但是, 要在使用前先行加载(例如装入 RAM)。编程软件运行在 RAM 中, 避免了与写入 EPROM 的冲突。要写的数据(字节)放在 RAM 中的“BurnByte”地址。EPROM 的目标地址由 SET 指令定义的寄存器指针确定。定时调整为  $1 \mu\text{s}$  一个周期。如果选择其他的循环时间/处理机频率, 则应该按照不同的操作条件调整软件。

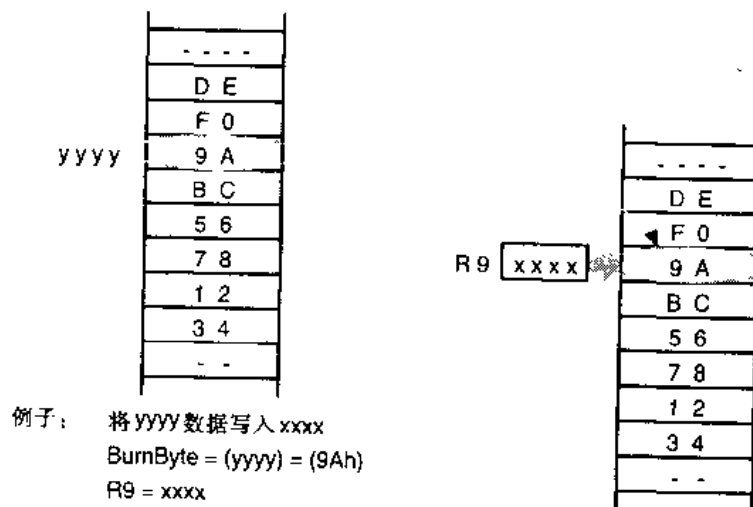


图 C.5 用快速编程算法向 EPROM 写入 1 个字节示例

当有数据往 EPROM 写入时,不能同时执行程序代码。在以下实例中,用一个子程序把程序代码移到另一个存储器中,如装入 RAM。程序流程图图 C.6 所示:

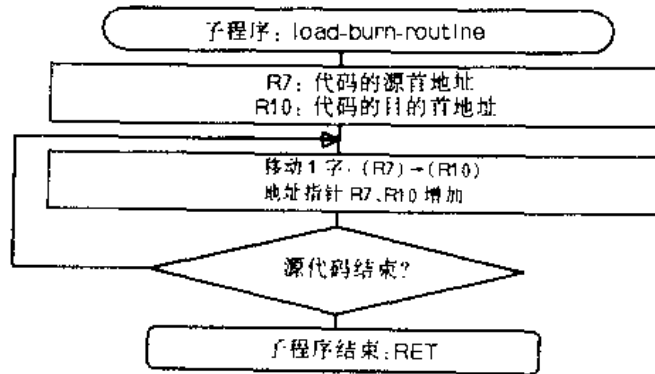


图 C.6 用快速编程算法向 EPROM 写 1 个字节流程图

```

;-----
; 子程序定义:
; 编程代码移入 RAM (load_burn_routine)
; EPROM 中编程 1 个字节 (Burn_EPROM, 可重定位)
;-----
-
EPCTL      .set      054h          ; EPROM 控制寄存器
VPPS       .set      2            ; 编程电压位
EXE        .set      1            ; 执行位
BurnByte   .set      0220h        ; 待写入数据的地址
Burn_orig  .set      0222h        ; 编程程序在 RAM 中的首地址
loops      .set      25
r_timer    .set      r8           ; 1 μs ≈ 1 个周期
pointer    .set      r9           ; EPROM 地址指针
; r9 在子程序调用前由主程序保护
r_count    .set      r10
lp         .set      3            ; dec r_timer          ; 1 个周期
; jnz          ; 2 个周期
ov         .set      2            ; mov # (100 - ov)/lp, r_timer ; 2 个周期
; 子程序: 将 EPROM 编程程序重定位到 RAM
; 编程子程序: 代码地址可浮动!
RAM_Burn_EPROM
        .set      Burn_orig
load_burn_routine
        push      r9
        push      r10
        mov       # Burn_EPROM, R9      ; 装入源指针
        mov       # RAM_Burn_EPROM, R10 ; 装入目的指针
load_burn1  mov       @R9, 0(R10)      ; 移动 1 个字

```

```

        incd    R10                ; 目的指针 + 2
        incd    R9                 ; 源指针 + 2
        cmp     #Burn_end, R9      ; 比较表的末地址
        jnc     load_burn1
        pop     r9
        pop     r10
        ret

; 子程序, 写 1 个字节到 EPROM 中
Burn_EPROM    dint                ; 确保编程正确
timing        mov.b    #VPPS, &EPCTL ; 加 VPPS
            push     r_timer        ; 保存寄存器
            push     r_count
            mov     #loops, r_count ; 2 个周期 = 2 μs
Repeat_Burn   mov.b    &BurnByte, 0(pointer) ; 数据写入 EPROM, 6 个周期 = 6 μs
            bis.b    #EXE, &EPCTL   ; 开 EXE, 4 个周期 = 4 μs
            ; 总周期(加 VPP 到开 EXE)
            ; 12 个周期 = 12 μs (min.)
            mov     #(100 - ov)/lp, r_timer ; 100 μs 编程脉冲
wait_100      ; 开始, 实际时间为 102 μs
            dec     r_timer
            jnz     wait_100
            bic.b   #EXE, &EPCTL   ; 关 EXE 及编程脉冲
            mov     #4, r_timer     ; 校验已编程 EPROM 前至少等待 10 μs
wait_10       ;
            dec     r_timer
            jnz     wait_10         ; 实际时间超过 13 μs
            cmp.b   &BurnByte, 0(pointer) ; 校验: 数据 = 编程数据
            jnc     Burn_EPROM_bad  ; 数据 ≠ 编程数据, 跳转
; 如果 EPROM 中编程数据正确, 则继续
            mov.b   &BurnByte, 0(pointer) ; 再次写入 EPROM
            bis.b   #EXE, &EPCTL   ; 开 EXE
            add     #(0ffffh - loop), r_count ; 成功编程的循环数
final_puls    mov     #(300 - ov)/lp, r_timer ; 开始 3 × 100 μs × N 个编程脉冲
wait_300      ;
            dec     r_timer
            jnz     wait_300
            inc     r_count
            jn     final_puls
            clr.b   &EPCTL        ; 关 EXE 及 VPPS
            jmp     Burn_EPROM_end
Burn_EPROM_bad
            dec     r_count        ; 失败: 减循环计数
            jnz     Repeat_Burn    ; 循环未完: 再次尝试

```











```
        inv. b    &BurnByte           ; 恢复写入标志
                                           ; EPROM 编程地址未变
Burn EPROM_end
        pop      r_timer
        pop      r_count
        eint
        ret
Burn_end
```



### 附录 D MSP430 系列单片机参数表

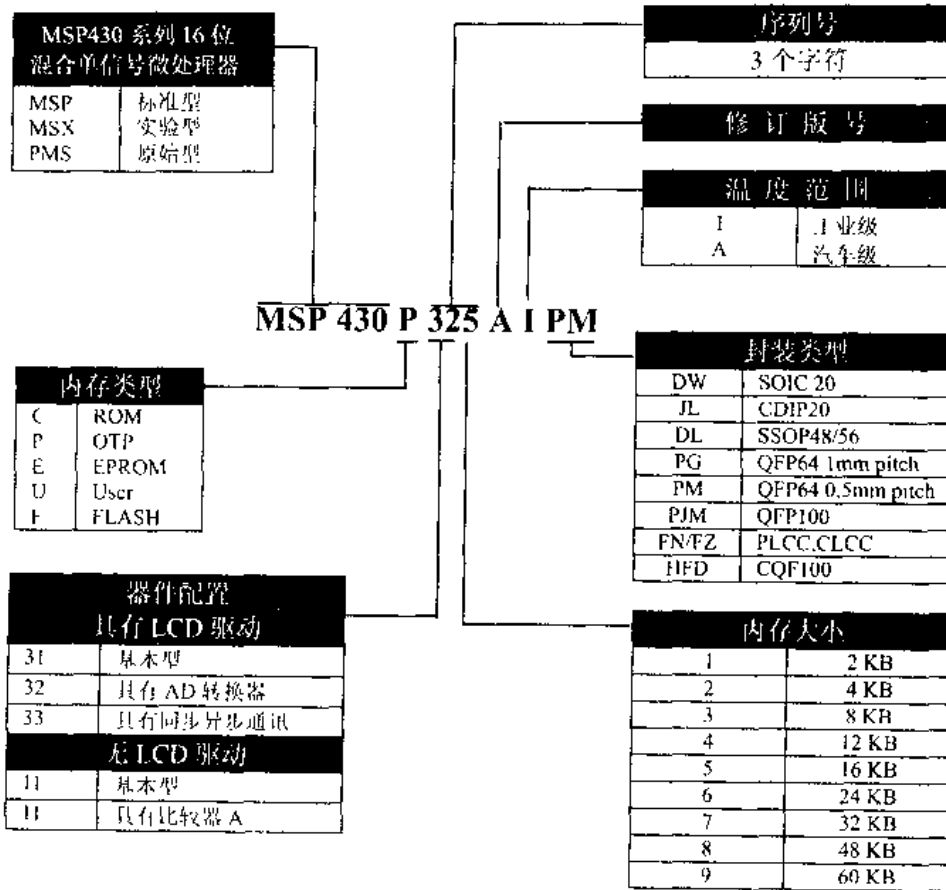
型号	EPROM	OTP	ROM	FLASH	Memory /KB	RAM /B	ADC	LCD 驱动	捕获/比较 正定时器	通用串行 发送接收	硬件 乘法器	定时器
MSP430F1101DW				F110	1	128	slope	none	yes	S / W	no	2
MSP430C1111DW			C111		2	128	slope	none	yes	S / W	no	2
MSP430C1121DW			C112		4	256	slope	none	yes	S / W	no	2
MSP430P1121DW		P112			4	256	slope	none	yes	S / W	no	2
PMS430E1121JL	E112				4	256	slope	none	yes	S / W	no	2
MSP430F1121DW				F112	4	256	slope	none	yes	S / W	no	2
MSP430F11011DW				F1101	1	128	slope	none	yes	S / W	no	2
MSP430C11111DW			C1111		2	128	slope	none	yes	S / W	no	2
MSP430C11211DW			C1121		4	256	slope	none	yes	S / W	no	2
MSP430F11211DW				F1121	4	256	slope	none	yes	S / W	no	2
MSP430C311SIDL			C311		2	128	slope	64	no	S / W	no	6
MSP430C312IDL			C312		4	256	slope	92	no	S / W	no	6
MSP430C313IDL			C313		8	256	slope	92	no	S / W	no	6
PMS430E313FZ	E313				8	256	slope	92	no	S / W	no	6
MSP430C314IDL			C314		12	512	slope	92	no	S / W	no	6
MSP430C315IDL			C315		16	512	slope	92	no	S / W	no	6
MSP430P315SIDL		P315			16	512	slope	92	no	S / W	no	6
PMS430E315FZ	E315				16	512	slope	92	no	S / W	no	6
MSP430C323IPG			C323		8	256	14 位	84	no	S / W	no	6
MSP430C325IPG			C325		16	512	14 位	84	no	S / W	no	6
MSP430P325IPM		P325			16	512	14 位	84	no	S / W	no	6
PMS430E325FZ	E325				16	512	14 位	84	no	S / W	no	6
MSP430C325AIPG			C325A		16	512	14 位	84	no	S / W	no	6
MSP430P325AIPM		P325A			16	512	14 位	84	no	S / W	no	6
MSP430C336IPJM			C336		24	1 024	slope	120	yes	H / W	yes	7
MSP430C337IDL			C337		32	1 024	slope	120	yes	H / W	yes	7
MSP430P337IPJM		P337			32	1 024	slope	120	yes	H / W	yes	7
PMS430E337HFD	E337				32	1 024	slope	120	yes	H / W	yes	7
MSP430C337AIPJM			C337A		32	1 024	slope	120	yes	H / W	yes	7
PMS430E337AHFD	E337A				32	1 024	slope	120	yes	H / W	yes	7

## 附录 E MSP430 系列单片机产品编码

型号	封装	脚位	型号	封装	脚位
 DW	SOP	20	 PG	QFP	64
 JL	CDIP	20	 PJM	QFP	100
 SxDL	SSOP	48	 HFD	CFP	100
 DL	SSOP	56	 FZ	JLCC	68
 PM	QFP	64			
 FN	PLCC	68			

1cm

## 附录 F MSP430 系列单片机封装形式





Powered by xiaoguo's publishing studio  
QQ:8204136