序

MSP430是美国TI(德州仪器)公司生产的超低功耗、高性能的16位微处理器。杭州利尔达科技有限公司在1999年同TI公司签署协议推广MSP430，成为TI公司的增值经销商。在利尔达公司不遗余力的推广下，MSP430从刚开始的只有寥寥无几的工程师和爱好者使用，发展到现在的成千上万的工程师在使用。MSP430单片机的产品已经遍布电子产品的每一个领域，如水气热电表、家用电器、智能家居、安防、工业控制、汽车、制造业等。MSP430所具有的很多显著的特点，使很多工程师一经使用即爱不释手。

到目前为止，TI公司已经推出了多款MSP430系列单片机，分别是MSP430x1xx、MSP430x2xx和MSP430x4xx系列，未来还会推出MSP430F5xx系列。

- MSP430x1xx系列单片机的型号从MSP430x11x到MSP430x16x，工作频率最高为8 MHz，Flash编程电压为2.7 V。
- MSP430x2xx系列单片机是增强型的单片机，性能在MSP430x1xx的基础上提高了很多，如功耗更低；Flash编程电压下降到2.2 V；若采用电池供电，还可以进一步延长电池的使用寿命；工作频率提高到16 MHz。
- MSP430x4xx具有LCD显示。在做具有LCD的产品设计时，可以优先考虑选用此款单片机；又由于LCD的引脚与I/O口是共用的，所以有些没有LCD显示的产品也可以选用。

MSP430是一款超低功耗的单片机，有很多低功耗的参数使用户可以轻松设计一款低功耗的产品(参数对于不同的型号会有所不同)：

- 0.1 μA掉电模式(仍然可以有相应的外部中断)；
- 0.8 μA待机模式(32768 Hz时钟保持运行)；
- 250 μA / 1 MIPS；
- <6 μs时钟唤醒时间；
- 零功耗BOR；
- <50 nA端口漏电流。

MSP430同时也是一款高性能的单片机，内部有很多的数字模块和模拟模块，使得用户在设计产品时可使用较少的外部器件，既降低了成本，又提高了产品性能。

- ADC12模块的A/D转换速度超过200 ksps，并具有连续转换功能，CPU可以在多次

A/D 转换完成后再去处理结果，提高了 CPU 的效率。

- 比较器有灵活的配置开关，用户可根据需要自行设置。
- USART 模块在设置波特率时，不再依赖晶振和波特率的整数倍关系，而是借助于模块中的波特率调整寄存器，可以对波特率进行微调，使整个通信的波特率误差控制在允许的范围内。工程师设计产品时，可以自由选择单片机的晶振，这大大地方便了设计。
- MSP430 的指令系统采用正交结构，共有 7 种寻址方式，源操作数支持 7 种寻址方式，目标操作数支持 4 种寻址方式。没有特殊结构的指令，容易记忆，编写程序不易出现差错。
- MSP430 的 Flash 支持多种编程方式，主要有 JTAG 编程、BSL 编程和自编程三种，可以轻松实现用户程序的升级功能，并可在 Flash 存储不需要频繁变化但需要掉电保护的数据。

MSP430 很便于开发。可采用 JTAG 调试接口，且每个芯片都有 JTAG 调试接口，具有调试功能，实际调试特别方便。IAR 仿真调试软件也特别易于使用。仿真工具非常便宜，而且到目前为止，MSP430 的仿真器可以兼容整个 MSP430 系列。开发工程师只要购买一个仿真器，即可使用不同款的 MSP430 设计产品。为降低读者学习和开发 MSP430 的成本，作者结合网上资料设计了简易的 JTAG 仿真器，读者可以按照图焊接进行学习，非常方便。

本书以 MSP430F149 为主线，介绍 MSP430 的基本结构、片上模块的基本原理以及许多常用的外设，并配备了详细的实验环节，将理论和实践很好地结合。MSP430 内部资源丰富，无法在一本书中面面俱到，但可喜的是 MSP430 的各种同名模块功能是完全一样的，读者很容易举一反三。同时，我们感谢作者在 MSP430 超低功耗 16 位单片机领域内所做的工作。

利尔达公司作为 TI 公司在中国的 MSP430 增值经销商，一直致力于 MSP430 的推广工作，拥有一支由很多资深 MSP430 技术工程师组成的团队，可以随时随地为广大 MSP430 爱好者和开发工程师提供技术帮助和支持。读者如有需要，可以登录 www.lierda.com 网站下载所需资料，或登录技术论坛 http://bbs.lierda.com 同广大的 MSP430 爱好者和工程师进行交流，也可以直接与我们联系。

利尔达科技有限公司总工程师

梁　源

2007 年 6 月

前 言

MSP430单片机是美国TI(德州仪器)公司近年推出的16位高性能混合信号处理器。由于它具有处理能力强、运算速度快、集成度高、外部设备丰富、超低功耗等优点，而且有很高的性价比，因此在许多领域内都得到了广泛的应用。

MSP430单片机可以采用汇编语言或C语言进行程序设计。它支持JTAG调试，其硬件仿真器是一个简单的并口转接器，并且适用于所有的MSP430单片机，大大降低了开发成本，也相对缩短了开发周期。其软件是由IAR公司提供的Embedded Workbench集成开发环境。此软件人机界面友好，易学易懂，并能很好地支持C语言开发。

本书分为两部分：基础篇和实践篇。基础篇为第1~7章，实践篇为第8~13章。

第1章从整体上介绍了MSP430单片机的特点和发展。读者通过学习本章，可以对MSP430单片机有一个初步的了解，并能够选择适合自己学习或进行产品开发所需要的机型。

第2章以MSP430F149单片机为例来讲解MSP430单片机的基础知识。本章内容主要包括MSP430单片机的内部结构、特点和应用，通过对CPU结构、内部Flash操作、系统复位、时钟模块、中断功能、看门狗以及最小应用系统介绍，达到对MSP430F149单片机的初步了解和应用，是本书最基础的部分。

第3~7章以MSP430F149为例，详细介绍了MSP430系列单片机所具有的基本I/O口、定时/计数器、通用串口、模拟比较器及ADC的基本结构，并通过举例讲解各单元的操作和使用方法。读者通过本章的学习，能够掌握MSP430F149单片机以及不同型号单片机的片内外设的应用。

第8~13章是本书的实践部分，重点介绍了一些应用实例。这一部分的学习有助于读者进一步应用MSP430单片机进行电子系统设计，主要包括RS485多机通信、大容量存储器扩展、基于LCD显示器数字电压表、I²C时钟芯片的应用、信号发生器以及USB接口开发等。各章节最后给出详细的设计说明、硬件电路和完整的程序设计。

本书具有以下主要特点：

- 主要以MSP430F149为例。在MSP430单片机中，MSP430F149具有一定的代表性，应用较广。
- 例子全部采用C语言编写，都是相对独立和完整的程序，是作者通过配套实验板调试的成果。程序都添加了详细的注释，便于阅读，完全可以直接应用。

- 配套光盘里包含了实践篇中所有程序代码和相关芯片的数据手册，方便读者查询和使用。
- 结构清晰，着重介绍 MSP430F149 单片机寄存器的基本功能和设置，以具体实践将读者带入 MSP430 单片机的设计中来。
- 作者专为本书设计了配套的 MSP430F149 实验板和简易的仿真器。通过在这个实验板进行具体的实验调试，有助于读者进一步掌握 MSP430 单片机的开发和应用。

本书由谢兴红、林凡强、吴雄英编写。孙旭副教授、闫萍老师参加了本书部分章节的编写和调试工作。

感谢郭勇教授、奚大顺教授一直以来对我工作上的关怀和指导，并对本书提出了改进意见。

感谢厦门的李冬发一直以来对我的指导和帮助。

感谢祝忠明副教授、邹彬老师对本书作出的贡献。

感谢孙旭副教授给我提供了一个良好的工作学习环境。

感谢学生蒋开强、黄伦忠、李夕红、符祥喜和刘良祥为本书编写作出的贡献。

感谢利尔达科技有限公司给作者提供 USB 仿真器和相关技术资料，感谢梁源总工程师在百忙之中为本书作序。

感谢谭敏，谢谢她一直以来对我工作的支持和对我生活的照顾，使我有更多时间投入到工作中。

最后，感谢我的父母及家人，谢谢他们多年来对我的培养和对我工作的支持。

由于作者水平有限，书中难免存在一些不足之处，欢迎广大读者批评指正，我的邮箱：xiexh@cdut.edu.cn。也可以到电子信息网 www.edires.net 论坛与我交流。

谢兴红

2007 年 10 月

于成都理工大学

目 录

基 础 篇

实 践 篇

基　础　篇

- 第 1 章　MSP430 单片机概述
- 第 2 章　MSP430F149 单片机基础知识
- 第 3 章　I/O 口基本操作
- 第 4 章　Timer_A 和 Timer_B
- 第 5 章　通用同步/异步收发器 USART
- 第 6 章　比较器 Comparator_A
- 第 7 章　ADC12

第1章

MSP430单片机概述

MSP430单片机是美国TI(德州仪器)公司于1996年推出的一种16位超低功耗的混合信号处理器。它将许多模拟电路外设(如ADC、DAC、模拟比较器等)和常用数字模块(如SCI、SPI、I²C、看门狗、PWM、CAP、定时/计时器)集成在芯片内部。通常对于一般实际应用单芯片便可以完全满足要求,这样可以降低外围控制电路的复杂性,节约PCB空间,同时也降低了设计成本,提高了系统的可靠性。

1.1 MSP430单片机的特点

MSP430系列单片机是一种超低功耗的微控制器系列,可使用电池长时间工作。MSP430系列是由各种不同的应用模块组合而成的多种型号。由于其具有16位的体系结构以及16位的CPU集成寄存器和常数发生器,可使MSP430实现代码效率最大化。数字控制振荡器使所有低功耗模式从唤醒到运行模式的唤醒时间小于6 μs。

MSP430单片机的主要特点如下:

- 超低功耗;
- 强大的处理能力,单周期指令周期,并且自带硬件乘法器;
- 高性能模拟技术及丰富的片上外围模块;
- 高稳定性,工业级产品;
- 方便、高效的集成开发环境;
- 温度适应范围宽,抗干扰能力强;
- 小巧灵活,性价比高。

MSP430单片机之所以具有超低的功耗,是因为其在降低芯片的电源电压及灵活可控的

运行时钟方面都有独到之处。

首先，MSP430 单片机的电源电压采用的是 1.8～3.3 V。当在 8 MHz 的时钟条件下运行时，芯片的电流为 200～400 μA，时钟关断模式的最低电流只有 0.1 μA。

其次，独特的时钟系统设计。MSP430 中有多个时钟源：内部 DCO 振荡器和外部高速、低速晶体振荡器。详细介绍可以参考后面章节。通过系统时钟模块来产生 CPU 和各功能模块所需的时钟源，并且这些时钟源可以在指令的控制下打开和关闭，从而实现对总体功耗的控制。

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

MSP430 单片机都集成了较丰富的片内外设。它们分别是看门狗定时器(WDT)、模拟比较器 A 、定时器 A(Timer_A)、定时器 B(Timer_B)、串口 0/1(USART0/1)、硬件乘法器、液晶驱动器、10 位/12 位 ADC、I^2C 总线直接数据存取(DMA)以及端口 1～6(P1～P6)等外围模块的不同组合。其中，看门狗可以使程序失控时迅速复位；模拟比较器进行模拟电压的比较，配合定时器可设计出 A/D 转换器；16 位定时器(Timer_A 和 Timer_B)具有捕获/比较功能，大量的捕获/比较寄存器可用于事件计数、时序发生、PWM 等；有的器件更具有异步、同步及多址访问串行通信接口，可方便地实现多机通信等应用；具有较多的 I/O 端口，最多达 6×8 条 I/O 口线；P1、P2 端口能够接收外部上升沿或下降沿的中断输入；10 位/12 位硬件 A/D 转换器具有较高的转换速率，最高可达 200 Kbps，能够满足大多数数据采集应用的需要；能直接驱动液晶多达 160 段；实现两路 12 位 D/A 转换；硬件 I^2C 串行总线接口实现存储器串行扩展；为了增加数据传输速度，而采用直接数据传输(DMA)模块。MSP430 系列单片机的这些片内外设为系统的单片解决方案提供了极大的方便。

上电复位后，系统时钟是由内部数字振荡器提供的，用户可以根据自己的需要改变系统时钟源。读者可以参考 2.4 节的系统时钟模块介绍。

目前，MSP430 系列有 OPT 型、Flash 型和 ROM 型三种器件。这些器件的开发手段不同，MSP430 系列单片机使用以 Flash 型为主。OPT 型和 ROM 型的器件是大量生产产品时由用户提供符合烧写格式的代码芯片给生产厂家，然后再烧写到内部存储器里，这样可大大降低系统成本；Flash 型则有十分方便的开发调试环境，器件片内有 JTAG 调试接口，还有可电擦写的 Flash 存储器，因此采用先下载程序到 Flash 内，再在器件内通过软件控制程序的运行，由 JTAG 接口读取片内信息供设计者调试使用的方法进行开发。这种方式只需要一台 PC 机和一个 JTAG 调试器。该 JTAG 调试器集仿真和编程功能于一体。

为了更好地适应工业级运行环境，MSP430 单片机的工作温度均为 -40～+ 85 ℃，所设

计的产品可以在工业环境下稳定运行。

1.2　MSP430 单片机的主要应用

(1) 工业控制

单片机的结构特点决定了它特别适用于各种控制系统。它既可作为单级控制器，又可作为多级控制的前沿处理机用于控制系统，应用领域相当广泛。例如：可应用于各种机床控制、电机控制、工业机器人、各种生产线、各种过程控制及各种检测系统等；在军事工业中，可应用于如导弹控制、鱼雷制导控制、智能武器装置、航天导航系统等；在汽车工业中，可应用于如点火控制、变速器控制、防滑刹车、排气控制等。

(2) 智能化仪器仪表

单片机用于包括温度、湿度、流量、流速、电压、频率、功率、厚度、角度、长度、硬度、元素等测定的各类仪器仪表中，使它们实现数字化、智能化、微型化、功能大大提高。

(3) 日常生活中的电器产品

单片机可用于电子秤、录像机、录音机、彩电、洗衣机、高级电子玩具、冰箱、照相机、家用多功能报警器等。

(4) 计算机网络与通信

单片机可用 BIT BUS、CAN、以太网等构成分布式网络系统，还可用于调制解调器、各种智能通信设备(如小型背负式通信机、列车无线通信等)、无线遥控系统等。

(5) 计算机外部设备

单片机可用于温氏硬盘驱动器、微型打印机、图形终端、CRT 显示器等。

1.3　MSP430 单片机的发展

MSP430 系列是具有精简指令集的、超低功耗的 16 位混合型单片机，于 1996 年问世。由于它具有极低的功耗、丰富的片内外设和方便灵活的开发手段，已成为众多单片机系列中一颗耀眼的新星。回顾 MSP430 系列单片机的发展过程，可以看出有以下三个阶段。

(1) 开始阶段

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

MSP430 的 33X、32X、31X 等系列具有 LCD 驱动模块，对提高系统的集成度较有利。每一系列有 ROM 型(C)、OTP 型(P)和 EPROM 型(E)等芯片。EPROM 型的价格昂贵，

运行环境温度范围窄，主要用于样机开发。这也表明了这几个系列的开发模式，即用户可以用EPROM 型开发样机，用 OTP 型进行小批量生产，而用 ROM 型进行大批量生产。

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

这个时期的 MSP430 已经显露出了超低功耗等一系列技术特点，但也有不尽如人意之处。它的许多重要特性（如片内串行通信接口、硬件乘法器、足够的 I/O 引脚等）只有 33X 系列才具备。33X 系列价格较高，比较适合于较为复杂的应用系统。当用户设计需要更多考虑成本时，33X 并不一定是最合适的。而片内高精度 A/D 转换器又只有 32X 系列才具备。

(2) 寻找突破，引入 Flash 技术

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

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

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

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

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

(3) 蓬勃发展阶段

在前一阶段引进新技术和进行内部调整，为 MSP430 的功能扩展打下了良好的基础。于是，TI 公司在 2002 年底和 2003 年期间又陆续推出了 F15X 和 F16X 系列产品。

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

2003 年，TI 公司还推出了专门用于电量计量的 MSP430FE42X 和用于水表、气表、热表上的具有无磁传感模块的 MSP430FW42X 单片机。

近年，TI 公司又推出了 MSP430F2XX 系列单片机，其主要特色在于具有更加超低的功耗

(2.2 V的编程电压)和更高速度的处理能力(16 MHz),引脚数和体积大大减小,又新增了 DIP 封装,这使它成为小型和手持设备等的系统设计的最佳选择。

1.4　MSP430 单片机介绍

1.4.1　主要功能部件

MSP430 系列单片机包含以下主要功能部件。

(1) CPU

MSP430 系列单片机的 CPU 和通用微处理器基本相同,只是在设计上采用了面向控制的结构和指令系统。MSP430 的内核 CPU 结构是按照精简指令集和高透明的宗旨而设计的,使用的指令有硬件执行的内核指令和基于现有硬件结构的仿真指令。这样可以提高指令执行速度和效率,增强了 MSP430 的实时处理能力。

(2) 存储器

存储程序、数据以及外围模块的运行控制信息,有程序存储器和数据存储器。对程序存储器访问总是以字节形式取得代码,而对数据可以用字(16 位)或字节方式访问。其中,MSP430 各系列单片机的程序存储器有 ROM、OTP、EPROM 和 Flash 型。

(3) 外围模块

经过 MAB(存储器地址总线)、MDB(存储器数据总线)、中断服务及请求线与 CPU 相连。MSP430 不同系列产品所包含外围模块的种类及数目可能不同。它们分别是以下一些外围模块的组合:时钟模块、看门狗、定时器 A、定时器 B、比较器 A、通用同步/异步串口 0/1、硬件乘法器、液晶驱动器、模/数转换、数/模转换、端口、基本定时器及 DMA 控制器等。

1.4.2　MSP430 各系列单片机简介

(1) MSP430F1XX

MSP430F1XX 系列单片机具有丰富的功能,既能作为带有比较器的简便低功耗控制器,又能作为完整的片上系统使用,其中包括多个高性能数据转换器、接口和乘法器。

下面就是 MSP430F1XX 系列芯片所有的硬件资源:

- 基本时钟系统(片内 DCO、8 MHz 或 32 kHz 可选);
- Timer_A3(带 3 个比较/捕获寄存器和 PWM 输出的 16 位定时器);
- Timer_B7(带 7 个比较/捕获寄存器和 PWM 输出的 16 位定时器);
- 比较器_A(精确的模拟比较器,常用于斜边(Slope)A/D 转换);

- 看门狗定时器/通用定时器；
- 6 个 I/O 端口，其中 1、2 端口有中断功能，每个端口 8 个引脚；
- USART0、USART1 和 SPI；
- I^2C、DMA；
- 12 位 A/D 转换器；
- DAC12 转换；
- 1～60 KB 的 Flash 存储容量。

(2) MSP430F2XX

新型的超低功耗 MSP430F2XX 系列将性能提高至 16 MHz。

MSP430F2XX 还在其他方面有了显著的增强，其中包括集成了误差为±1%的内置振荡器，软件可设定内部上拉/下拉电阻，以及增加了模拟输入数量等，从而进一步降低了外部元件的需要。

可在线编程的闪存也有所改进。其采用更小的 64 字节区段，并且有更低的 2.2 V 编程电压，从而省去了大部分系统的外部 EEPROM。

下面就是 MSP430F2XX 系列芯片所有的硬件资源：

- 基本时钟系统(片内 DCO、16 MHz 或 32 kHz 可选)；
- Timer_A3(带 3 个比较/捕获寄存器和 PWM 输出的 16 位定时器)；
- 在线比较器/斜边 A/D 转换；
- 看门狗定时器/通用定时器；
- 4 个 I/O 端口，其中 1、2 端口有中断功能；
- USI、USCI；
- 16 位 A/D 转换器；
- 1～32 KB 的 Flash 存储容量；
- 程序代码保护；
- 2 个配置放大器。

(3) MSP430F4XX

超低功耗 MSP430F4XX 系列带有集成 LCD 控制器，非常适用于低功耗测量与医疗应用。具有满足特定功能的外设，比如能够针对流量和电量测量提供单芯片解决方案。这些集成外设有助于减少芯片个数，并且降低系统成本与功耗。

下面就是 MSP430F1XX 系列芯片所有的硬件资源：

- 基本时钟系统(片内 DCO、8 MHz 或 32 kHz 可选)；
- Timer_A3(带 3 个比较/捕获寄存器和 PWM 输出的 16 位定时器)；
- Timer_B7(带 7 个比较/捕获寄存器和 PWM 输出的 16 位定时器)；
- 比较器_A(精确的模拟比较器，常用于斜边(Slope)A/D 转换)；

- 看门狗定时器/通用定时器；
- 最多10个I/O端口，其中1,2端口具有中断功能；
- USART0、USART1和SPI；
- 12位A/D转换器；
- 4～120 KB的Flash存储容量；
- I^2C、DMA；
- DAC12转换；
- 最多集成160段段码LCD；
- 最多3个运算放大器。

1.5 MSP430单片机的选型

应用MSP430系列单片机构建应用系统时，进行系统设计必须考虑选型的问题。

为了得到最容易实现设计目标且性价比高的机型，MSP430单片机选型基本遵循以下规则。

由于MSP430系列单片机在不断地开发，产品也在不断更新，这里不可能包括所有产品信息。

读者可以在TI公司网站(www.ti.com)或者MSP430中国代理商利尔达科技有限公司网站(www.lierda.com)上查询更详细的产品资料。

在芯片的选型上，主要需要考虑以下几个方面：

① 系统功耗资源要求；

② 芯片功耗要求(考虑引脚、体积)；

③ 系统存储器容量要求。

(1) MSP430F1XX系列单片机选型指南

MSP430F1XX系列单片机选型指南如表1.1所列。

(2) MSP430F2XX系列单片机选型指南

MSP430F2XX系列单片机选型指南如表1.2所列。

(3) MSP430F4XX系列单片机选型指南

MSP430F4XX系列单片机选型指南如表1.3所列。

本章主要讲述了MSP430系列单片机的特点、发展前景、应用领域及选型指南。通过对本章的学习，希望读者能够从整体上对MSP430单片机有一定的认识，并根据具体的需求选择合适的单片机型号。

表 1.1　MSP430F1XX 系列 MCU 选型指南

型　号	Flash KB	SRAM B	I/O	16 位定时/计数器 A	16 位定时/计数器 B	USART	I²C	DMA	掉电检测复位	Comp_A	ADC（通道/位数）	封　装
MSP430F1101A	1	128	14	3						Y	Slope	20 DGV,DW,PW,24 RGE
MSP430F1111A	2	128	14	3						Y	Slope	20 DGV,DW,PW,24 RGE
MSP430F1121A	4	256	14	3						Y	Slope	20 DGV,DW,PW,24 RGE
MSP430F1122	4	256	14	3					Y		8/10	20 DW,PW,32 RHB
MSP430F122	4	256	22	3						Y	Slope	28 DW,PW,32 RHB
MSP430F123	8	256	22	3						Y	Slope	28 DW,PW,32 RHB
MSP430F1222	4	256	22	3		1			Y		8/10	28 DW,PW,32 RHB
MSP430F1232	8	256	22	3		1			Y		8/10	28 DW,PW,32 RHB
MSP430F133	8	256	48	3	3	1				Y	8/12	64 PM,PAG,RTD
MSP430F135	16	512	48	3	3	1				Y	8/12	64 PM,PAG,RTD
MSP430F147	32	1024	48	3	7	2				Y	8/12	64 PM,PAG,RTD
MSP430F1471	32	1024	48	3	7	2				Y	Slope	64 PM,RTD
MSP430F148	48	2048	48	3	7	2				Y	8/12	64 PM,PAG,RTD
MSP430F1481	48	2048	48	3	7	2				Y	Slope	64 PM,RTD
MSP430F149	60	2048	48	3	7	2				Y	8/12	64 PM,PAG,RTD
MSP430F1491	60	2048	48	3	7	2				Y	Slope	64 PM,RTD
MSP430F155	16	512	48	3	3	1	Y	Y	Y	Y	8/12	64 PM,RTD[2]
MSP430F156	24	1024	48	3	3	1	Y	Y	Y	Y	8/12	64 PM,RTD[3]
MSP430F157	32	1024	48	3	3	1	Y	Y	Y	Y	8/12	64 PM,RTD[3]
MSP430F167	32	1024	48	3	7	2	Y	Y	Y	Y	8/12	64 PM,RTD[3]
MSP430F168	48	2048	48	3	7	2	Y	Y	Y	Y	8/12	64 PM,RTD[3]
MSP430F169	60	2048	48	3	7	2	Y	Y	Y	Y	8/12	64 PM,RTD[3]
MSP430F1610	32	5120	48	3	7	2	Y	Y	Y	Y	8/12	64 PM,RTD
MSP430F1611	48	10240	48	3	7	2	Y	Y	Y	Y	8/12	64 PM,RTD
MSP430F1612	55	5120	48	3	7	2	Y	Y	Y	Y	8/12	64 PM,RTD

表 1.2　MSP430F2XX 系列 MCU 选型指南

型　号	Flash/KB	SRAM/B	I/O	16 位定时/计数器 A	16 位定时/计数器 B	USCI	USI	DMA	掉电检测复位	Comp_A	ADC（通道/位数）	封　装
MSP430F2001	1	128	10	2					Y	Y	Slope	14 PW,N,16 RSA
MSP430F2011	2	128	10	2					Y	Y	Slope	14 PW,N,16 RSA
MSP430F2002	1	128	10	2			Y		Y		1/10	14 PW,N,16 RSA
MSP430F2012	2	128	10	2			Y		Y		1/10	14 PW,N,16 RSA
MSP430F2003	1	128	10	2			Y		Y		1/16	14 PW,N,16 RSA
MSP430F2013	2	128	10	2			Y		Y		1/16	14 PW,N,16 RSA
MSP430F2101	1	128	16	3					Y	Y	Slope	20 DGV,DW,PW,24 RGE
MSP430F2111	2	128	16	3					Y	Y	Slope	20 DGV,DW,PW,24 RGE
MSP430F2121	4	256	16	3					Y	Y	Slope	20 DGV,DW,PW,24 RGE
MSP430F2131	8	256	16	3					Y	Y	Slope	20 DGV,DW,PW,24 RGE
MSP430F2232	8	512	32	3	3	Y			Y		1/12	38 DA,40 RHA
MSP430F2252	16	512	32	3	3	Y			Y		1/10	38 DA,40 RHA
MSP430F2272	32	1024	32	3	3		Y		Y		1/10	38 DA,40 RHA
MSP430F2234	8	512	32	3	3	Y			Y		1/10	38 DA,40 RHA
MSP430F2254	16	512	32	3	3	Y			Y		1/10	38 DA,40 RHA
MSP430F2274	32	1024	32	3	3	Y			Y		1/10	38 DA,40 RHA
MSP430F2330	8	1024	32	3	3	Y			Y	Y	Slope	40 RHA
MSP430F2350	16	2048	32	3	3	Y			Y	Y	Slope	40 RHA
MSP430F2370	32	2048	32	3	3	Y			Y	Y	Slope	40 RHA

表 1.3　MSP430F4XX 系列 MCU 选型指南

型　号	Flash KB	SRAM.B	I/O	16 位定时 计数器 A	16 位定时 计数器 B	USART	LCD	DMA	掉电检测复位	Comp_A	ADC（通道/位数）	封　装
MSP430F412	4	256	48	3			96		Y	Y	slope	64 PM,RTD
MSP430F413	8	256	48	3			96		Y	Y	slope	64 PM,RTD
MSP430415	16	512	48	3.5			96		Y	Y	slope	64 PM
MSP430F417	32	1024	48	3.5			96		Y	Y	slope	64 PM
MSP430FW423	8	256	48	3.5			96		Y	Y	slope	64 PM
MSP430FW425	16	512	48	3.5			96		Y	Y	slope	64 PM
MSP430FW427	32	1024	48	3.5			96		Y	Y	slope	64 PM
MSP430F4250	16	256	32	3			56		Y		5/16	48 DL,RGZ
MSP430F4260	24	256	32	3			56		Y		5/16	48 DL,RGZ
MSP430F4270	32	256	32	3			56		Y		5/16	48 DL,RGZ
MSP430F423	8	256	14	3		1	128		Y		3/16	64 PM
MSP430F425	16	512	14	3		1	128		Y		3/16	64 PM
MSP430F427	32	1024	14	3		1	128		Y		3/16	64 PM
MSP430FE423	8	256	14	3		1	128		Y		3/16	64 PM
MSP430FE425	16	512	14	3		1	128		Y		3/16	64 PM
MSP430FE427	32	1024	14	3		1	128		Y		3/16	64 PM
MSP430F435	16	512	48	3	3	1	128/160		Y	Y	8/12	80 PN,100 PZ
MSP430F436	24	1024	48	3	3	1	128/160		Y	Y	8/12	80 PN,100 PZ
MSP430F437	32	1024	48	3	3	1	128/160		Y	Y	8/12	80 PN,100 PZ
MSP430FG437	32	1024	48	3	3	1	128	Y	Y	Y	8/12	80 PN
MSP430FG438	48	2048	48	3	3	1	128	Y	Y	Y	8/12	80 PN
MSP430FG439	60	2048	48	3	3	1	128	Y	Y	Y	8/12	80 PN
MSP430F447	32	1024	48	3	7	2	160		Y	Y	8/12	100 PZ
MSP430F448	48	2048	48	3	7	2	160		Y	Y	8/12	100 PZ
MSP430F449	60	2048	48	3	7	2	160		Y	Y	8/12	100 PZ
MSP430FG4616	92	4096	80	3	7	1	160	Y	Y	Y	12/12	100 PZ
MSP430FG4617	92	8192	80	3	7	1	160	Y	Y	Y	12/12	100 PZ
MSP430FG4618	116	8192	80	3	7	1	160	Y	Y	Y	12/12	100 PZ
MSP430FG4619	120	4096	80	3	7	1	160	Y	Y	Y	12/12	100 PZ

第2章

MSP430F149单片机基础知识

TI公司的MSP430系列是一种超低功耗微控制器系列，由多种满足不同需要的型号组成。该单片机功耗低，可使用户的应用系统长时间工作在电池供电系统中。由于其具有16位的体系结构及16位的CPU数据处理能力和常数发生器，可使MSP430实现代码效率最大化。内部DCO数字控制振荡器使所有低功耗模式唤醒到运行模式的唤醒时间小于6 μs。本书将以MSP430F149单片机为例对MSP430系列单片机进行系统讲解，本书所有程序包括配套的实验板均以MSP430F149单片机为例。通过很小的改变就可以方便地将程序用于其他型号的MSP430单片机。

2.1 CPU概述

2.1.1 CPU的特点

MSP430F149是TI公司最近推出的MSP430系列超低功耗微控制器中的一种，它是由2个16位定时器、8路快速12位A/D转换器、2个通用串行同步/异步通信信号接口(USART)和48个I/O引脚等构成的微控制器。其特点如下：

- 电源电压范围：1.8～3.6 V；
- 超低功耗：
- 待机模式：1.6 μA；
- 关闭模式(RAM保持)：0.1 μA；
- 活动模式：280 μA@1 MHz，2.2 V；
- 5种省电模式：

- 快速唤醒模式(6 μs 内从待机模式唤醒)；
- 16 位 RISC 结构；
- 内部集成看门狗定时器防止程序跑飞；
- 快速的数据处理能力内部自带硬件乘法器；
- 高处理速度，指令周期为 125 ns(8 MHz 工作方式)；
- 多个 I/O 支持中断模式满足系统对外部中断的需求；
- 通用串口支持 SPI、SCI 模式；
- 带内部参考、采样保持和自动扫描特性的 12 位 A/D 转换器；
- 有 3 个捕获/比较寄存器的 16 位定时器 Timer_A 支持 PWM 和 CAP 功能；
- 有 7 个捕获/比较寄存器的 16 位定时器 Timer_B 支持 PWM 和 CAP 功能；
- 片内集成模拟比较器；
- 片上集成 60 KB 的 Flash 和 2 KB 的 RAM 同时提供 256 字节的信息 Flash；
- 串行在线编程，无需外部编程电压，安全熔丝可编程代码保护 MSP430F149：60 KB+256 字节 Flash，2 KB 的 RAM；
- 可用封装：64 脚方形扁平封装(QFP)。

其典型应用包括数据采集和处理、工业现场控制和常用仪器仪表等。更详细的应用介绍读者可以参考 TI 公司提供的技术文档。

2.1.2 CPU 结构和引脚说明

MSP430F149 的 CPU 结构如图 2.1 所示。主要包含下列模块：基础时钟、看门狗定时器、

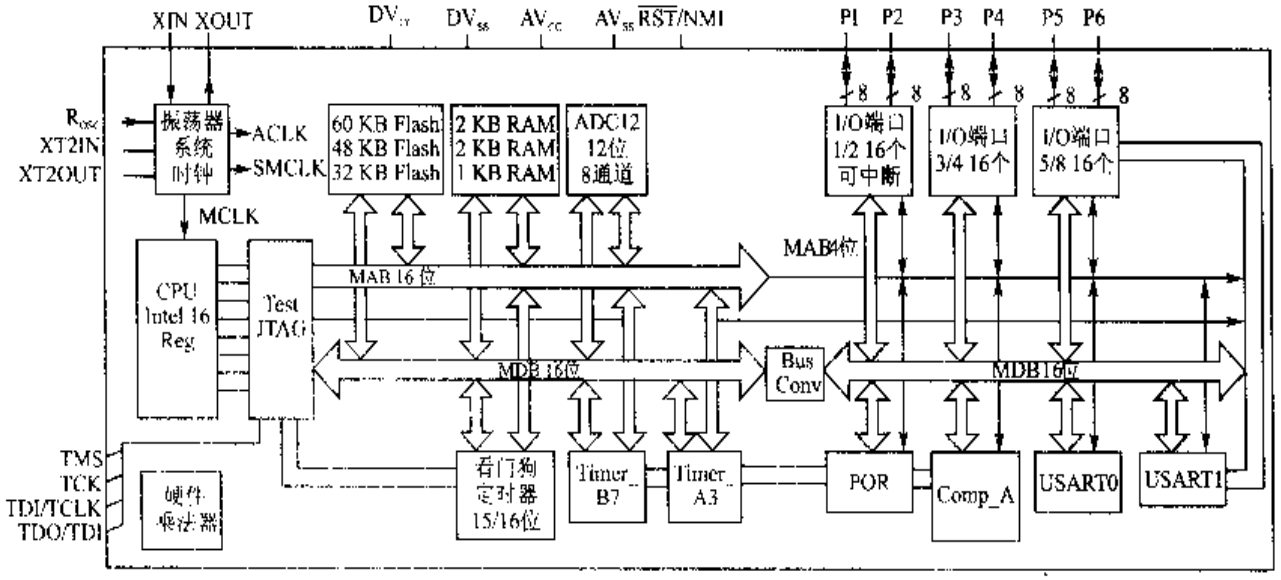


图 2.1 MSP430F149 的 CPU 结构

Timer_A、Timer_B、6 个 8 位并行端口(其中 P1、P2 具有中断功能)、模拟比较器 COMPARATOR_A、12 位 A/D 转换器、2 通道串行通信接口(通过软件选择 UART/SPI 模式)、1 个硬件乘法器、1 个 Flash 以及 2 KB 的 RAM。

MSP430F149 的 CPU 引脚排列见图 2.2,引脚的详细说明见表 2.1。

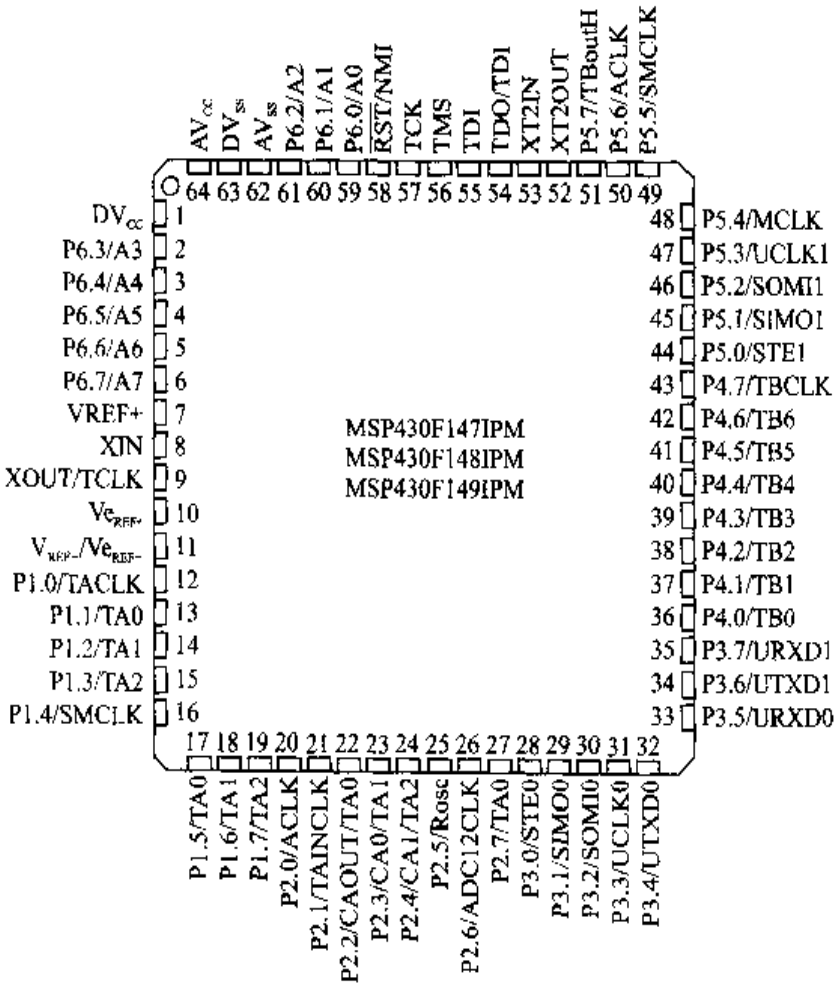


图 2.2　MSP430F149 的 CPU 引脚排列

表 2.1　MSP430F149 的 CPU 引脚的说明

引脚名称	引脚编号	I/O	功能说明
AV_{CC}	64		模拟电源正端,注意在使用内部 ADC 模块时,模拟电源和数字电源隔离以提高 ADC 采用精度
AV_{SS}	62		模拟电源负端,注意在使用内部 ADC 模块时,模拟地和数字地隔离以提高 ADC 采用精度
DV_{CC}	1		数字电源正端
DV_{SS}	63		数字电源负端

续表 2.1

引脚名称	引脚编号	I/O	功能说明
P1.0/TACLK	12	I/O	普通数字 I/O 引脚/Timer_A,时钟信号 TACLK 输入
P1.1/TA0	13	I/O	普通数字 I/O 引脚/Timer_A,捕获:CCI0A 输入,比较:OUT0 输出
P1.2/TA1	14	I/O	普通数字 I/O 引脚/Timer_A,捕获:CCI1A 输入,比较:OUT1 输出
P1.3/TA2	15	I/O	普通数字 I/O 引脚/Timer_A,捕获:CCI2A 输入,比较:OUT1 输出
P1.4/SMCLK	16	I/O	普通数字 I/O 引脚/SMCLK 信号输入
P1.5/TA0	17	I/O	普通数字 I/O 引脚/Timer_A,比较:OUT0 输出
P1.6/TA1	18	I/O	普通数字 I/O 引脚/ Timer_A,比较:OUT1 输出
P1.7/TA2	19	I/O	普通数字 I/O 引脚/Timer_A,比较:OUT2 输出
P2.0/ACLK	20	I/O	普通数字 I/O 引脚/ACLK 输出
P2.1/TAINCLK	21	I/O	普通数字 I/O 引脚/Timer_A:时钟信号 INCLK
P2.2/CAOUT/TA0	22	I/O	普通数字 I/O 引脚/Timer_A:捕获:CCI0B 输入/比较器_A 输出
P2.3//CA0/TA1	23	I/O	普通数字 I/O 引脚/Timer_A:比较:Out1 输出/比较器_A 输入
P2.4/CA1/TA2	24	I/O	普通数字 I/O 引脚/Timer_A:比较:Out2 输出/比较器_A 输入
P2.5/Rosc	25	I/O	普通数字 I/O 引脚/定义 DCO 标称频率的外部电阻输入
P2.6/ADC12CLK	26	I/O	普通数字 I/O 引脚/转换时钟－12 位 ADC
P2.7/TA0	27	I/O	普通数字 I/O 引脚/Timer_A,比较:Out0 输出
P3.0/STE0	28	I/O	普通数字 I/O 引脚/从发送使能－USART0/SPI 方式
P3.1/SIMO0	29	I/O	普通数字 I/O 引脚/从输入/主输出－USART0/SPI 方式
P3.2/SOMI0	30	I/O	普通数字 I/O 引脚/从输出/主输入－USART0/SPI 方式
P3.3/UCLK0	31	I/O	普通数字 I/O 引脚/外部时钟输入 USART0/UART/SPI 方式
P3.4/UTXD0	32	I/O	普通数字 I/O 引脚/发送数据输出 USART0/UART 方式
P3.5/URXD0	33	I/O	普通数字 I/O 引脚/接收数据输入－USART0/UART 方式
P3.6/UTXD1	34	I/O	普通数字 I/O 引脚/发送数据输出 USART1/UART 方式
P3.7/URXD1	35	I/O	普通数字 I/O 引脚/接收数据输入－USART1/UART 方式
P4.0/TB0	36	I/O	普通数字 I/O 引脚/Timer_B,捕获:CCI0A 或 CCI0B 输入,比较:OUT0 输出
P4.1/TB1	37	I/O	普通数字 I/O 引脚/Timer_B,捕获:CCI1A 或 CCI1B 输入,比较:OUT1 输出
P4.2/TB2	38	I/O	普通数字 I/O 引脚/Timer_B,捕获:CCI2A 或 CCI2B 输入,比较:OUT2 输出
P4.3/TB3	39	I/O	普通数字 I/O 引脚/Timer_B,捕获:CCI3A 或 CCI3B 输入,比较:OUT3 输出
P4.4/TB4	40	I/O	普通数字 I/O 引脚/Timer_B,捕获:CCI4A 或 CCI4B 输入,比较:OUT4 输出
P4.5/TB5	41	I/O	普通数字 I/O 引脚/Timer_B,捕获:CCI5A 或 CCI5B 输入,比较:OUT5 输出
P4.6/TB6	42	I/O	普通数字 I/O 引脚/Timer_B,捕获:CCI6A 或 CCI6B 输入,比较:OUT6 输出

续表 2.1

引脚名称	引脚编号	I/O	功能说明
P4.7/TBCLK	43	I/O	普通数字 I/O 引脚/输入时钟 TBCLK－Timer_B7
P5.0/STE1	44	I/O	普通数字 I/O 引脚/从发送使能　USART1/SPI 方式
P5.1/SIMO1	45	I/O	普通数字 I/O 引脚 从输入/主输出 USART1/SPI 方式
P5.2/SOMI1	46	I/O	普通数字 I/O 引脚/从输出/主输入－USART1/SPI 方式
P5.3/UCLK1	47	I/O	普通数字 I/O 引脚/外部时钟输入 －USART1/UART/SPI 方式，时钟输出－USART1/SPI 方式
P5.4/MCLK	48	I/O	普通数字 I/O 引脚/主系统时钟 MCLK 输出
P5.5/SMCLK	49	I/O	普通数字 I/O 引脚/次系统时钟 SMCLK 输出
P5.6/ACLK	50	I/O	普通数字 I/O 引脚/辅助时钟 ACLK 输出
P5.7/TBoutH	51	I/O	普通数字 I/O 引脚/切换所有 PWM 输出端口到高阻－Timer_B7 TB0 到 TB6
P6.0/A0	59	I/O	普通数字 I/O 引脚/12 位 ADC 的模拟输入 a0 端
P6.1/A1	60	I/O	普通数字 I/O 引脚/12 位 ADC 的模拟输入 a1 端
P6.2/A2	61	I/O	普通数字 I/O 引脚 12 位 ADC 的模拟输入 a2 端
P6.3/A3	2	I/O	普通数字 I/O 引脚 12 位 ADC 的模拟输入 a3 端
P6.4/A4	3	I/O	普通数字 I/O 引脚 12 位 ADC 的模拟输入 a4 端
P6.5/A5	4	I/O	普通数字 I/O 引脚/12 位 ADC 的模拟输入 a5 端
P6.6/A6	5	I/O	普通数字 I/O 引脚 12 位 ADC 的模拟输入 a6 端
P6.7/A7	6	I/O	普通数字 I/O 引脚/12 位 ADC 的模拟输入 a7 端
$\overline{\text{RST}}$/NMI	58	I	复位输入，非屏蔽中断输入端口，或引导装载程序启动(Flash 器件)
TCK	57	I	测试时钟 TCK 是用于器件编程测试或引导装载程序启动(Flash 器件)时钟输入端口
TDI/CLK	55	I	测试数据输入或测试时钟输入，器件保护熔丝连接到该引脚
TDO/TDI	54	I/O	测试数据输出端口，TDO/TDI 数据输出或编程数据输入端口
TMS	56	I	选择测试模式，TMS 用作一个器件编程和测试的输入端口
V_{REF+}	10	I	ADC 外部参考电压输入
V_{REF+}	7	O	ADC 内部参考电压正端输出
V_{REF-}/V_{eREF-}	11	I	内部 ADC 参考电压和外部施加的 ADC 参考电压的负端
XIN	8	I	晶体振荡器 XT1 的输入端口，可以连接标准晶体或手表晶体
XOUT	9	O	晶体振荡器 XT1 的输出端口
XT2IN	53	I	晶体振荡器 XT2 的输入端口，只能连接标准晶体
XT2OUT	52	O	晶体振荡器 XT2 的输出端

2.2 Flash 操作

2.2.1 存储空间组织

MSP430 系列单片机的存储空间采用的是冯·诺依曼结构，它的 Flash 和 RAM 位于同一地址空间，位于 0000H～FFFFH 范围内。在此范围内分别有 SFR、外围模块寄存器、数据存储器、程序存储器和中断向量表。图 2.3 所示为 MSP430F149 的存储器结构图。

注意：通过 JTAG 下载程序到芯片里面，一般是从 0FFDFH 开始向下存储的。对于寄存器的地址，如果是用 C 语言进行编程，则可以不用关心地址，这些都可以在 C 语言的头文件中查到。

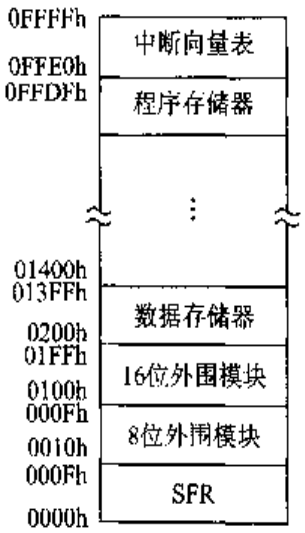


图 2.3 MSP430F149 的存储器结构图

2.2.2 Flash 操作

MSP430F149 单片机内部 Flash 的大小为 60 KB 的主 Flash 和两个信息段存储器(共 256 字节)。主 Flash 和信息段除了所在存储器的位置和大小不一样，基本操作完全相同。MSP430F149 单片机的信息段在实际应用中是常用的，它可以用来保存用户程序设置的工作参数，同时也很容易按照需求修改。在其他类型的单片机系统中，通常采用扩展外部 EEPROM 的方法来保存这些工作参数。如果读者已经是一个单片机产品设计工程师，则有很深的体会；如果是一个初学者，也没有关系，通过本书的学习将体会到使用内部信息段的优点。MSP430F149 单片机内部 Flash 和控制电路的基本结构如图 2.4 所示。

MSP430F149 单片机的 Flash 模块可以按位、字节和字访问，并且可以进行编程和擦除。Flash 模块采用一个控制器来控制编程和擦除操作。该控制器有 3 个寄存器，用来产生编程和擦除的时序，也用来控制提供编程和擦除的电压。Flash 模块的初始模式是读操作。在读操作下不能进行编程操作和擦除操作，此时时序发生器和电压发生器关闭，存储器等同于一个 ROM。MSP430 的内部 Flash 进行在系统编程时，不需要额外的外部电压，编程电压由内部编程模块产生。CPU 对内部的 Flash 进行操作时，在 BLKWRT、WRT、MERAS 和 ERASE 位置 1 后，即可执行字节/字的写操作、块写操作、段擦除操作、所有主存储器段擦除操作以及全部擦除操作。

MSP430F149 的 Flash 分成许多段。单个的位、字节和字可以写入 Flash 存储；但是擦除操作最小只能以段为单位，也可根据用户需要以块进行操作。

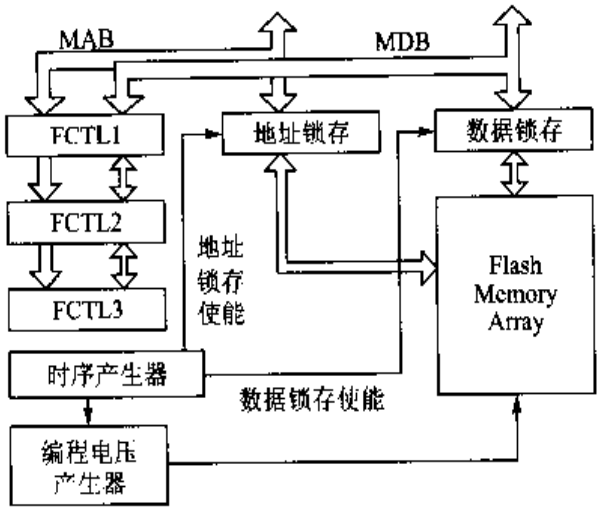


图 2.4　内部 Flash 和控制电路基本结构图

下面就是 MSP430F149 中 Flash 的一些特点：

- Flash 有 N 段主存储器和每段 128 字节的两段信息存储器(A 和 B)，主存储器每段长 512 字节。
- 段 0～N 可以一步擦除，也可以每段分别擦除。
- 段 A 和段 B 可以分别擦除，或与段 0～N 作为一组擦除。
- 安全熔丝烧断是不可逆的，然后对 JTAG 进一步访问无效。
- 编程/擦除电压内部产生，无需任何外部器件，但 VCC 电源电流增大。
- 编程擦除时间由闪速存储器中的硬件控制，无须软件干涉。
- 控制硬件作为闪速定时发生器，其输入频率应处于合适的频率范围，并应一直保持到编程和擦除操作完成。
- 在编程和擦除操作期间，没有代码能从程序存储区执行，此时必须通过设置 GIE、NMIE、ACCVIE 和 OFIE 位为 0 来禁止所有中断。如果不禁止中断，有可能使程序以为跑飞。
- 未编程的新器件在信息存储器中可能有一些已经编程的字节(用于生产测试)，在首次使用时应该执行一次擦除操作。

Flash MEMORY 分为主程序存储区和信息存储区。但是对它们读/写操作都一样，数据和代码都可以存储在任何一段中，唯一的区别是段的长度和地址不一样。图 2.5 中列出了 4 KB的主存储区和 256 字节信息区的分段情况。

MSP430F149 有 60 KB Flash，其信息区为 01000H～010FFH，其主程序区为 01100H～0FFDFH，中断向量为 0FFE0H～0FFFFH，如图 2.5 所示。其信息区分为 2 段，每段长度为 128 字节；其主程序区每段长度为 512 字节。

图 2.6 所示为读/写操作的时序产生器，读/写频率为 257～476 kHz。

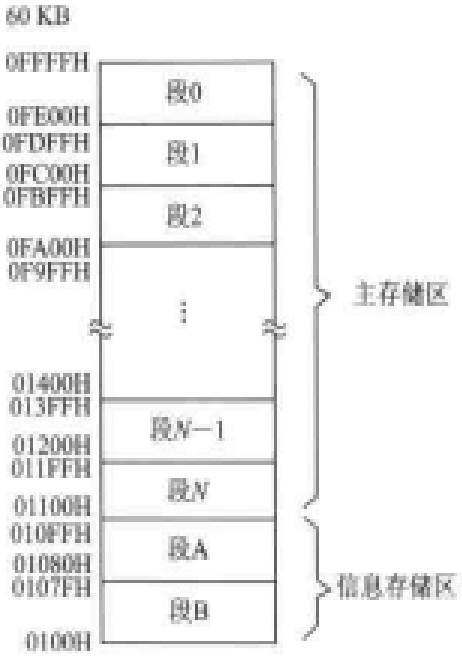


图 2.5 MSP430F149 单片机的存储器结构

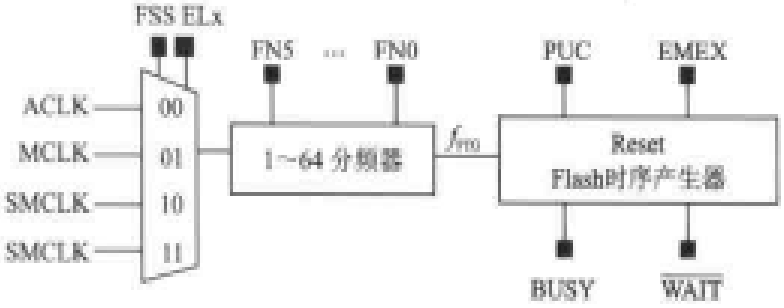


图 2.6 读/写操作的时序产生器

图 2.7 和图 2.8 所示分别为 Flash 擦除和编程操作所应遵循的时序。

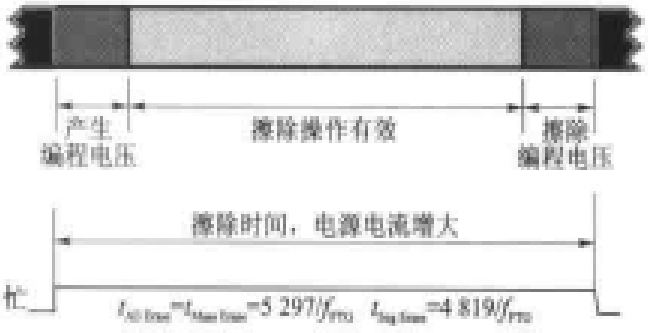


图 2.7 Flash 擦除时序

用户对 Flash 模块的操作是通过它的寄存器来完成的。Flash 操作所涉及到的寄存器有 FCTL1、FCTL2、FCTL3 和 IE1。下面通过分别介绍每个寄存器，来说明对 Flash 的操作。

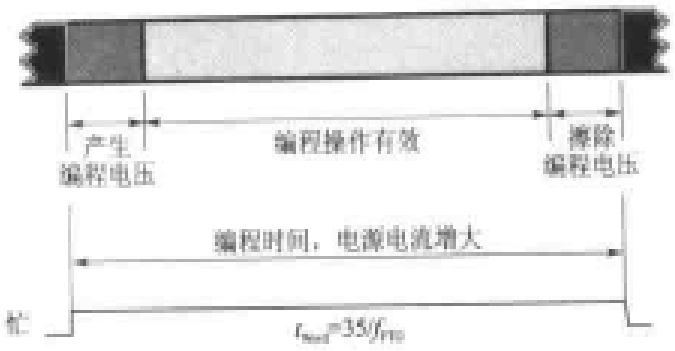


图 2.8　Flash 编程时序

1. FCTL1 寄存器

如图 2.9 所示，FCTL1 是一个 16 位寄存器，其在存储器的地址为 0128H。

15 … 8	7	6	5	4	3	2	1	0
安全键值	BLKWRT	WRT	Reserved	Reserved	Reserved	MERAS	ERASE	Reserved
	rw-0	rw-0	r0	r0	r0	rw-0	rw-0	r0

图 2.9　FCTL1 寄存器

该寄存器定义了 Flash 模块的擦除和编程操作的控制位。其中，高 8 位为安全键值，该字段读出的内容总是 96H，写入时必须为 A5H，否则不能进行操作。

下面分别对低 8 位进行介绍。

BLKWRT　Bit7　块块写模式位。复位值为 0。在进行块写操作时，WRT 位必须置 1；当设置 EMEX 位时，BLKWRT 自动复位。

0　块写模式关闭；

1　块写模式开启。

WRT　Bit6　写模式使能位。复位值为 0。当设置 EMEX 位时，WRT 自动复位。

0　不能进行写操作；

1　写操作允许。

MERAS、ERASE　Bits2～1　该两位用来控制擦除方式选择位。复位值为 0。当设置 EMEX 位时，这两位自动复位。

00　不擦除；

01　只擦除单个段；

10　擦除所有的主程序区；

11　擦除所有的主程序区和信息区。

2. FCTL2 寄存器

如图 2.10 所示，FCTL2 是一个 16 位寄存器，其在存储器的地址为 012AH。

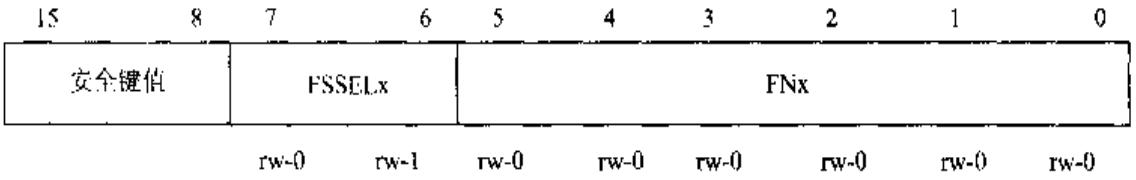


图 2.10 FCTL2 寄存器

该寄存器定义了 Flash 模块的擦除和编程操作所需要的时序时钟。其中，高 8 位为安全键值，该字段读出的内容总是 96H，写入时必须为 A5H，否则不能进行操作。

下面对低 8 位分别进行介绍。

FSSELx　Bits7～6　该两位用来定义 Flash 模块控制器时钟源的选择。复位值为 01。

00　ACLK；　10　SMCLK；

01　MCLK；　11　SMCLK。

FNx　Bits5～0　这 6 位定义了分频系数。分频系数为 FN5～FN0 的值。如当 FN5～FN0 的值为 2 时，其分频系数为 3。复位值为 1，分频系数为 2。

3. FCTL3 寄存器

如图 2.11 所示，FCTL3 是一个 16 位寄存器，其在存储器的地址为 012CH。

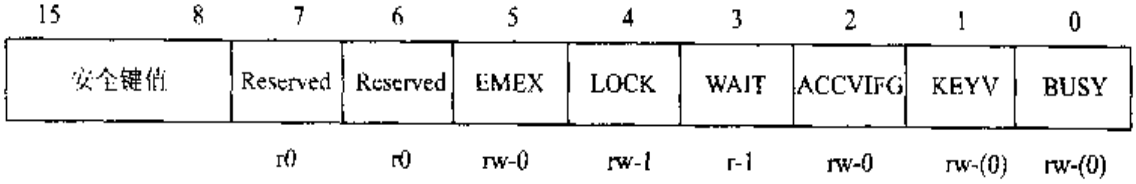


图 2.11 FCTL3 寄存器

该寄存器定义了 Flash 模块的擦除和编程操作的一些标志位。其中，高 8 位为安全键值，该字段读出的内容总是 96H，写入时必须为 A5H，否则不能进行操作。

下面分别对低 8 位进行介绍。

EMEX　Bit5　紧急退出位。当该位为 1 时，则立即退出对 Flash 的操作。

LOCK　Bit4　保护位。

0　不加锁，可以对 Flash 操作；

1　加锁，这时不能对 Flash 进行写和擦除操作。

WAIT　Bit3　等待指示位。该位显示 Flash 正在进行写操作。

		0　　还没准备好，此时不能进行写操作；
		1　　Flash 准备好下一次写操作。
ACCVIFG	Bit2	违反访问中断标志。该位只能软件清零。
		0　　没有中断挂起；
		1　　有中断挂起。
KEYV	Bit1	安全值错误标志。
		0　　安全值正确；
		1　　安全值错误。
BUSY	Bit0	忙标志。
		0　　不忙；
		1　　忙。

4. IE1 寄存器

如图 2.12 所示，IE1 是一个 8 位寄存器。与 Flash 操作有关的位只有第 5 位 ACCIE。该位是用来允许违反中断访问的控制位。当该位为 1 时，中断允许；当该位为 0 时，中断禁止。

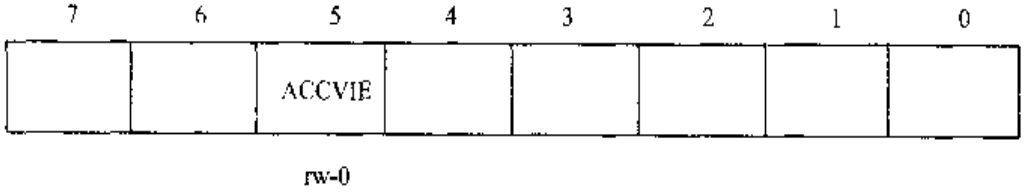


图 2.12　IE1 寄存器

通过上面对 Flash 操作所涉及到的寄存器的介绍，对 Flash 的操作过程一定有了一些了解。下面用一个例子来说明 Flash 操作的基本流程。其中，Flash 操作的流程图如图 2.13 所示。

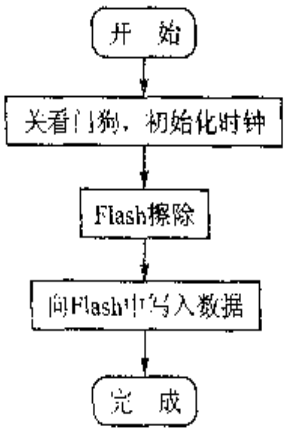


图 2.13　Flash 操作的流程图

下面的程序为具体的写内部 Flash 的实例，往内部信息段的三个地址写入数据 1、2、3，三个地址分别为 0x1080、0x1083、0x1085。在芯片写入数据前首先要把 Flash 擦除，擦除后的 Flash 数据全部为 0xFF，之后才能进行写操作。程序运行后，读者可以通过 IAR IDE 查看内部 Flash 的数据。

```
#include "msp430x14x.h"
#define  uchar  unsigned char
#define  uint  unsigned int
void  write_Seg(unsigned char value,unsigned int add);
unsigned char temp[128];
void int_clk()
```

```
{
    uchar i;
    BCSCTL1&=~XT2OFF;                          //打开 XT 振荡器
    BCSCTL2 =SELM1+SELS;                       //MCLK 为 8 MHz,SMCLK 为 1 MHz
    {
        IFG1 &=~OFIFG;                         //清除振荡错误标志
        for(i=0;i<100;i++)
    _NOP();                                    //延时等待
        ;
    while((IFG1 & OFIFG)!=0);                  //如果标志为 1,则继续循环等待
    IFG1&=~OFIFG;
}
delay()
{
    unsigned int i=0;
    for(i=0;i<=1000;i++)
        ;
}
void EraseFlashSegment( unsigned int add )
{
    unsigned char *ptrFlash;
    ptrFlash=(unsigned char *)add;
    FCTL1 = FWKEY + ERASE;                     // 使能擦除位
    FCTL3 = FWKEY;                             // 清除锁定位
    *ptrFlash=0x00;
    delay();
    delay();
    FCTL3 = FWKEY + LOCK;                      // 置锁定位为 1
}
void write_Seg (unsigned char value,unsigned int add)
{
    unsigned char * Flash_ptr;                 // Flash 访问指针
    Flash_ptr = (unsigned char *) add;         // 初始化指针
    FCTL3 = FWKEY;                             // 清除锁定位
    FCTL1 = FWKEY + WRT;                       // 使能写使能位
    while(BUSY & FCTL3);
    *Flash_ptr=value;
    FCTL1 = FWKEY;                             // 清除写使能
    FCTL3 = FWKEY + LOCK;                      // 置锁定位为 1
```

```
}

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;                    // 禁止看门狗
    int_clk() ;
    FCTL2 = FWKEY + FSSEL0 + FN0;                // MCLK/2 作为 Flash 操作时序的时钟
Generator
    EraseFlashSegment( 0x1080 );                 //擦除 SegmentA Flash
    write_Seg(1,0x1080);                         //写地址 0x1080 为 1
    write_Seg(2,0x1083);                         //写地址 0x1083 为 2
    write_Seg(3,0x1087);                         //写地址 0x1087 为 3
    while(1);
    }
```

通过上面的程序可以看出，只要设置好 FCTL1、FCTL2、FCTL3 和 IE1 就可以对 Flash 模块进行适当的读/写操作。

2.3　系统复位和工作模式

2.3.1　系统复位和初始化

1. 系统复位

图 2.14 所示为 MSP430 系统复位电路功能模块图。从图中可以看到，有两个复位信号：上电复位信号 POR(Power On Reset)和上电清除信号 PUC(Power Up Clear)。

其中：† 表示来自看门狗定时器外围模块；

‡ 表示带有断电检测(BOR)的器件；

= 表示不带有断电检测(BOR)的器件；

§ 表示带有电压监管(SYS)的器件。

POR 信号是器件的复位信号，此信号只有在以下事件发生时才会产生：

- 器件上电时；
- $\overline{\text{RST}}$/NMI 引脚配置为复位模式，当$\overline{\text{RST}}$/NMI 引脚产生低电平时。

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

- POR 信号发生时；

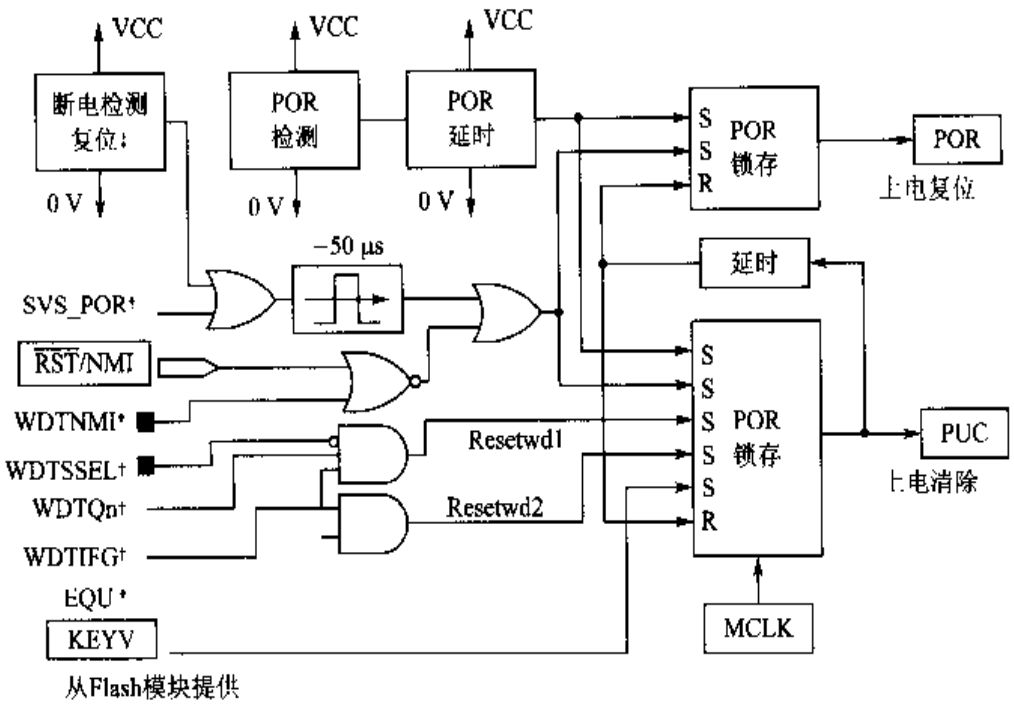


图 2.14　MSP430 系统复位电路功能模块

- 启动看门狗，看门狗定时器计满时；
- 向看门狗写入错误的安全参数值时；
- 向片内 Flash 写入错误的安全参数值时。

2. 系统复位后器件的初始化

当 POR 信号或 PUC 信号发生时引起器件复位后，器件的初始化状态为：

- $\overline{\text{RST}}$/NMI 引脚配置为复位模式；
- I/O 引脚为输入模式；
- 状态寄存器复位；
- 程序计数器(PC)装入复位向量地址 0FFFEH，CPU 从此地址开始执行。

其他模块的寄存器初始化，详见后面几章对各个模块的介绍。

下面介绍 POR 和 PUC 两者的关系。

POR 信号的产生会导致系统复位并产生 PUC 信号，而 PUC 信号不会引起 POR 信号的产生。无论是 PUC 或是 POR 信号产生复位后，都会使 MSP430 从地址 0FFFEH 开始读取复位中断向量，程序从中断向量所指向的地址开始执行。触发 PUC 信号复位的条件中，除了 POR 信号产生外，其他都可以通过读取相应的中断向量来判断是什么原因引发了复位。

系统在 POR 复位后的状态：

- $\overline{\text{RST}}$/NMI 引脚被设置为复位模式；
- 所有 I/O 引脚被设置为输入；

- 外围模块被初始化，其寄存器为器件手册上的默认值；
- 状态寄存器 SR 复位；
- 看门狗激活，进入工作模式；
- 程序计数器 PC 装入 0FFFEH 处的地址，微处理器从此地址开始执行程序。

在 POR 复位后，用户必须通过软件对一些寄存器进行如下设置：

① 初始化 SP 指针，一般指向 RAM 的顶部；

② 按系统要求设置看门狗；

③ 按系统和实际应用配置外围模块寄存器。

2.3.2　工作模式

MSP430 系列单片机主要用于低功耗系统，它可以设置成不同的操作模式。操作模式的设置需要考虑以下 3 个方面的要求：低功耗应用、速度和数据处理要求以及单个外围设备的最小电流消耗。根据系统的要求设置系统进入不同的模式。当进入低功耗模式后，系统时钟会停止，甚至主系统时钟都可以停止，这样系统的功耗只有微安(μA)级；外部的中断可以在6 μs 内将系统从低功耗唤醒来执行相应的操作。进入低功耗模式后，所有的 I/O 中断、RAM 和寄存器的内容都不会发生改变，并且所有的中断都可以将系统从低功耗模式唤醒，从而进入中断服务程序进行相应的处理。

MSP430 有 1 种活动模式和 5 种低功耗模式。而不同的模式是通过设置状态寄存器的 CPUOFF、OSCFF、SCG0、SCG1 等位来设置的。

各模式下的功耗比较和结构示意图分别如图 2.15 和图 2.16 所示。

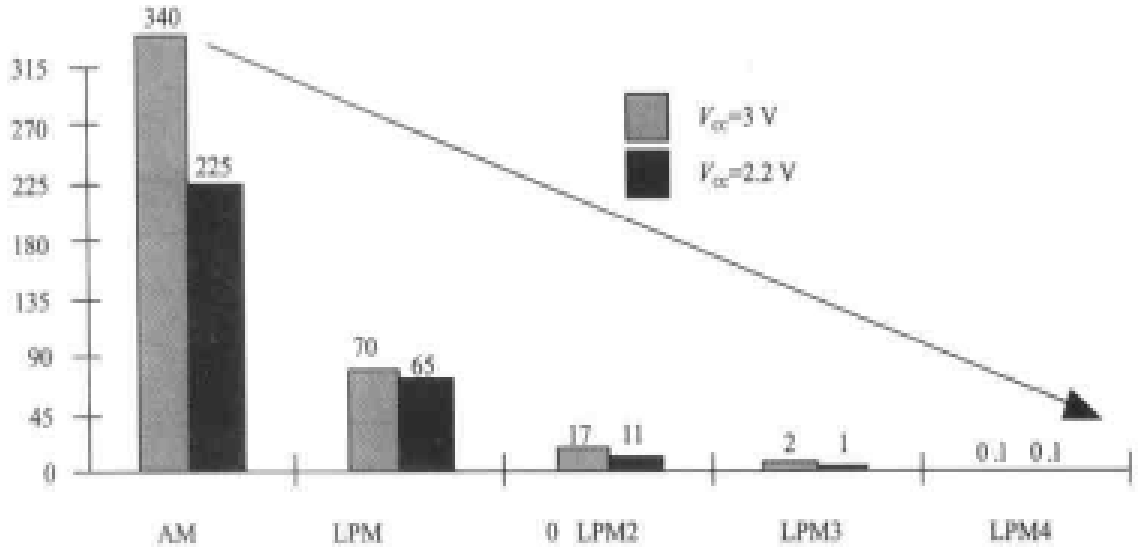


图 2.15　各模式下的功耗比较

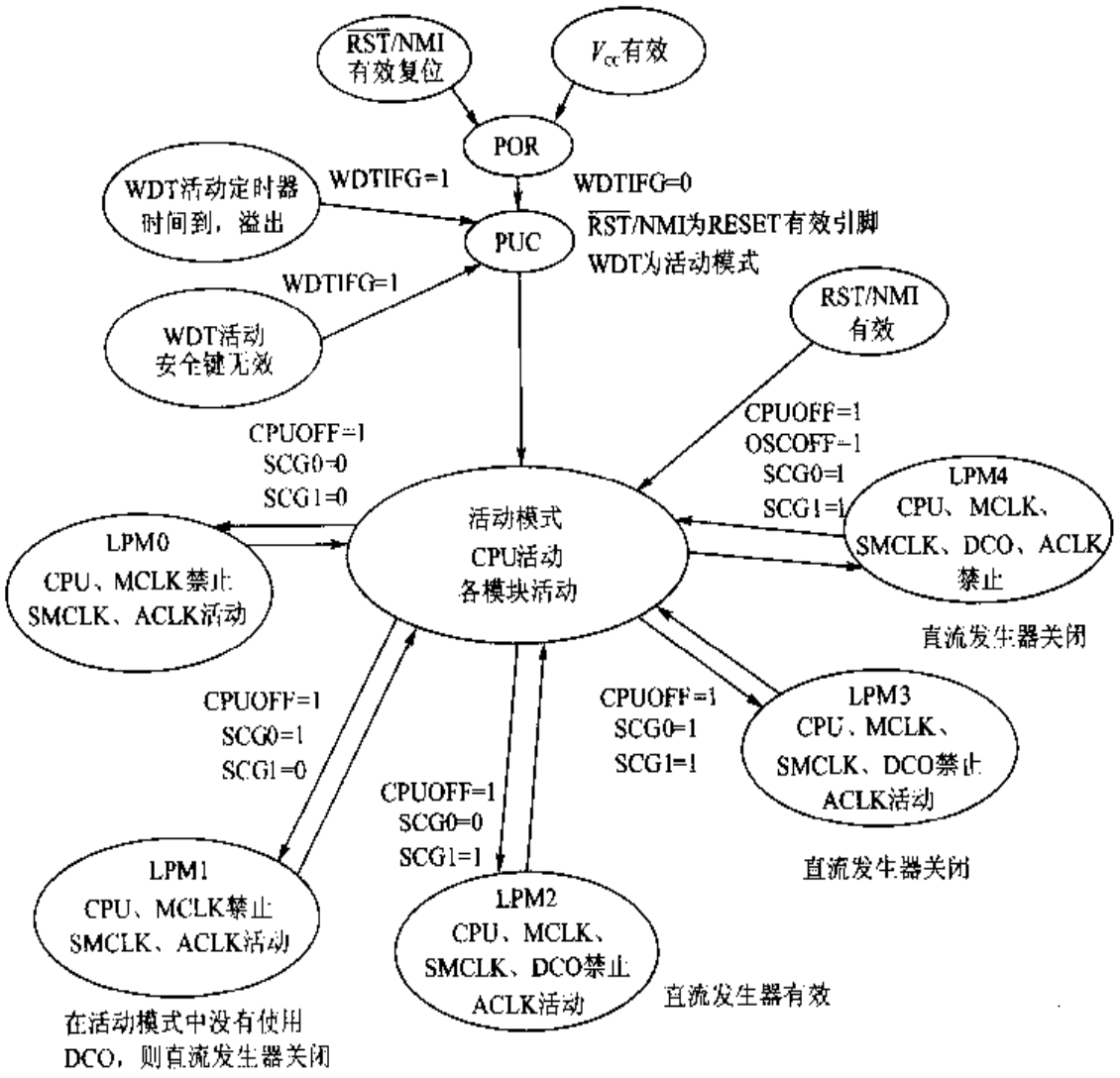


图 2.16　各个模式的结构示意图

各工作模式、各控制位及时钟的状态如表 2.2 所列。

表 2.2　各工作模式、各控制位及时钟的状态

工作模式	控制位	CPU 状态、振荡器及时钟状态
AM	SCG1=0， SCG0=0 Oscoff=0， CPUoff=0	CPU、MCLK、SMCLK、ACLK 均处于活动状态
LPM0	SCG1=0， SCG0=0 Oscoff=0， CPUoff=1	CPU、MCLK 禁止；SMCLK、ACLK 活动

续表 2.2

工作模式	控制位	CPU 状态，振荡器及时钟状态
LPM1	SCG1=0， SCG0=1 Oscoff=0， CPUoff=1	CPU 禁止；如果 DCO 未用作 MCLK 或 SMCLK，则直流发生器被禁止，否则仍然保持活动； SMCLK，ACLK 活动
LPM2	SCG1=0， SCG0=1 Oscoff=0， CPUoff=1	CPU、MCLK、SMCLK 禁止； 如果 DCO 未用作 MCLK 或 SMCLK，则自动被禁止； ACLK 活动
LPM3	SCG1=1， SCG0=1 Oscoff=0， CPUoff=1	CPU、MCLK、SMCLK 禁止； DCO、直流发生器被禁止； ACLK 活动
LPM4	SCG1=X， SCG0=X Oscoff=0， CPUoff=1	CPU、MCLK、SMCLK 禁止； DCO、直流发生器被禁止； ACLK 活动； 所有振荡器停止工作

接下来介绍跟工作模式相关的状态寄存器 SR。状态寄存器的位分配如图 2.17 所示。

15	9	8	7	6	5	4	3	2	1	0
Reserved		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	rw-0

图 2.17　状态寄存器 SR

对其中各个位的介绍如下：

V　　Bit8　　溢出标志位。当运算结果为负时，该位被置 1，否则为 0。

SCG1，SCG0　　Bits7～6　　用来选择系统 4 种不同的时钟。

00　　SMCLK、ACLK；

01　　SMCLK、ACLK；

10　　ACLK；

11　　ACLK。

OSCOFF　　Bit5　　晶体控制位。

0　　晶体处于工作状态；

1　　所有晶体振荡器停止。

CPUOFF　　Bit4　　CPU 控制位。

		1　CPU 进入关闭模式； 0　CPU 处于工作状态。
GIE	Bit3	中断控制位。该位为系统总的中断允许位。 1　允许中断； 0　禁止所有中断。
N	Bit2	负标志位。当运算结果为负时，该位被置 1，否则为 0。
Z	Bit1	零标志位。当运算结果为 0 时，该位被置 1，否则为 0。
C	Bit0	进位标志位。当运算结果产生进位时，CPU 将该位置 1，否则为 0。

2.4　MSP430 的基础时钟模块

MSP430F149 单片机的时钟模块结构如图 2.18 所示。MSP430F149 单片机的基础时钟主要是由低频晶体振荡器 LFXT1、高频晶体振荡器 LFXT2 及数字控制振荡器 DCO 组成。

MSP430 单片机的型号不同，对应的时钟模块也将有所不同。虽然不同型号单片机的时钟模块有所不同，但这些模块产生出来的结果是相同的。在 MSP430F13/14 中有 XT2 振荡器，而 MSP430F11X/F11X1 中是用 LFXT1CLK 来代替 XT2CLK 时钟信号的。在时钟模块中有 3 个(对于 F13 和 F14)时钟信号源(或 2 个时钟信号源，对于 F11X 和 F11X1)。各个时钟信号源介绍如下。

(1) LFXT1CLK：低频/高频时钟源。可以外接 32768 Hz 的时钟芯片或频率为 450 kHz～8 MHz的标准晶体或共振器。

(2) XT2CLK：高频时钟源。需要外接两个振荡电容器。可以外接频率为 450 kHz～8 MHz的标准晶体、共振器和外部时钟输入。较常用的晶体振荡器是 8 MHz 的。

(3) DCOCLK：内部数字可控制的 RC 振荡器。MSP430 单片机时钟模块提供 3 个时钟信号以供给片内各部电路使用，这 3 个时钟信号分别是：

① ACLK：辅助时钟信号。从图 2.18 中可以看出，ACLK 是从 LFXT1CLK 信号由 1/2/4/8 分频器分频后所得到的。由 BCSCTL1 寄存器设置 DIVA 相应位来决定分频因子。ACLK 可提供给 CPU 外围功能模块作时钟信号使用。

② MCLK：主时钟信号。从图 2.18 中可以看出，MCLK 是由 3 个时钟源所提供的。它们分别是：LFXT1CLK、XT2CLK 和 DCO 时钟源信号。MCLK 主要用于 MCU 和相关系统模块作时钟。同样可设置相关寄存器来决定分频因子及相关的设置。

③ SMCLK：子系统时钟。从图 2.18 中可以看出，SMCLK 由 2 个时钟源信号提供，它们分别是 XT2CLK 和 DCO。如果是 F11 或 F11X1 系列单片机，则由 LFXT1CLK 代替

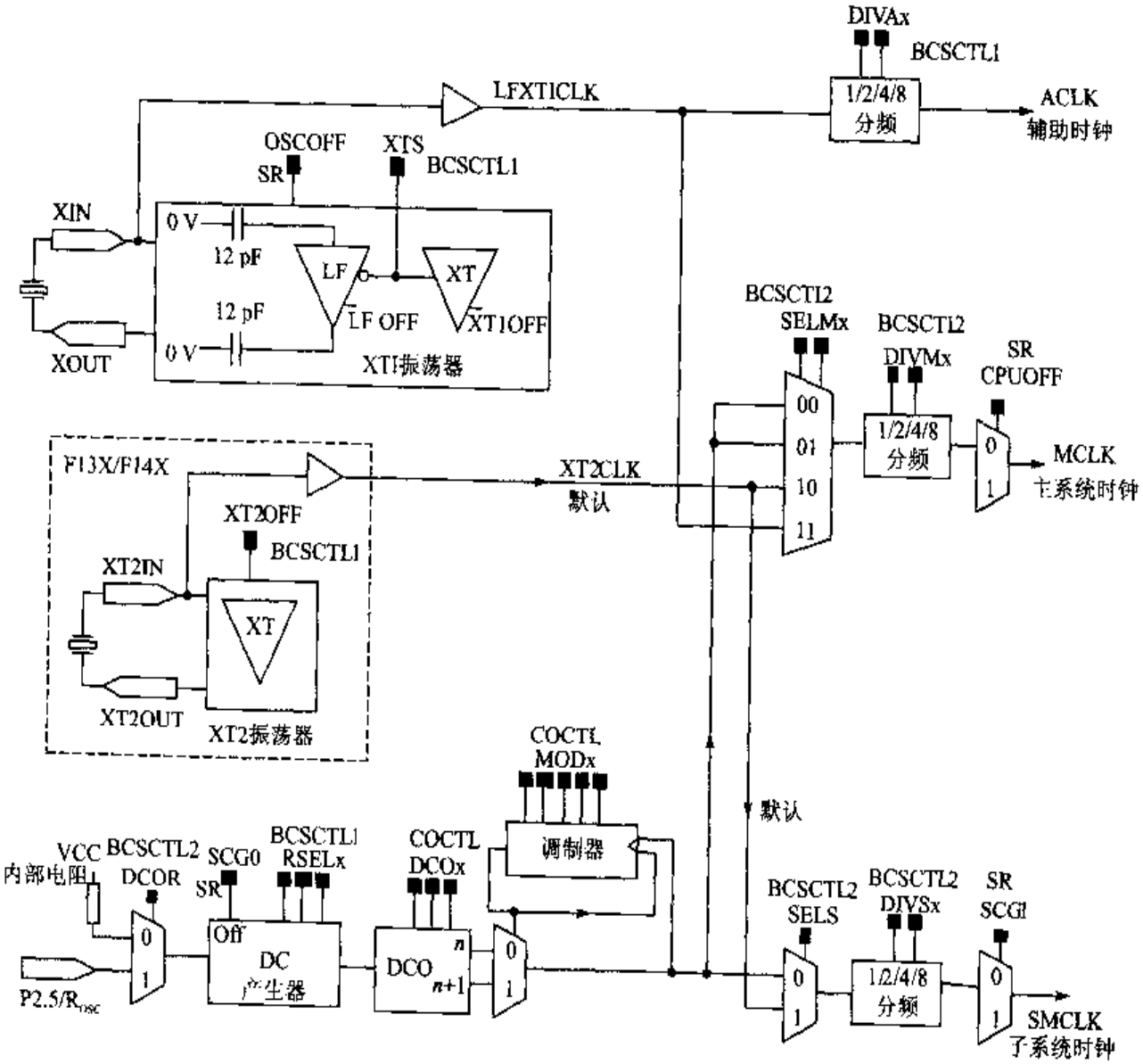


图 2.18　MSP430 单片机的时钟模块结构图

XT2CLK。同样可设置相关寄存器来决定分频因子及相关的设置。

2.4.1　基础时钟模块

当微处理器发生 PUC 复位后，MCLK 和 SMCLK 配置为 DCO 时钟模块(默认值为 800 kHz)，ACLK 配置为 LFXT1 时钟模块，并且为低频模式。修改 DCOCTL、BCSCTL1 和 BCSCTL2 寄存器的值，可以对系统和各个模块的时钟进行设置。设置状态寄存器的 SCG0、SCG1、OCSOFF 和 CPUOFF 位，可以设置微处理器的工作模式。下面将分别介绍低频振荡器 LFXT1、高频振荡器 LFXT2 和数字控制振荡器 DCO 的细节和操作。

1. 低频振荡器 LFXT1

低频振荡器 LFXT1 的结构如图 2.19 所示。LFXT1 支持超低功耗，它在低频模式下使用一个 32768 Hz 的晶体。32768 Hz 的晶体连接在 XIN 和 XOUT 引脚上，不需要任何电容，在低频模式下内部集成了电容。低频振荡器也支持高频模式和高速晶体，但连接时每端必须加电容。电容大小根据所接晶体频率的高低来选择。低频振荡器在低频和高频模式时都可以选择从 XIN 引脚接入一个外部输入时钟信号，但所接频率必须根据所设定的工作模式来选择，并且 OSCOFF 位必须复位。

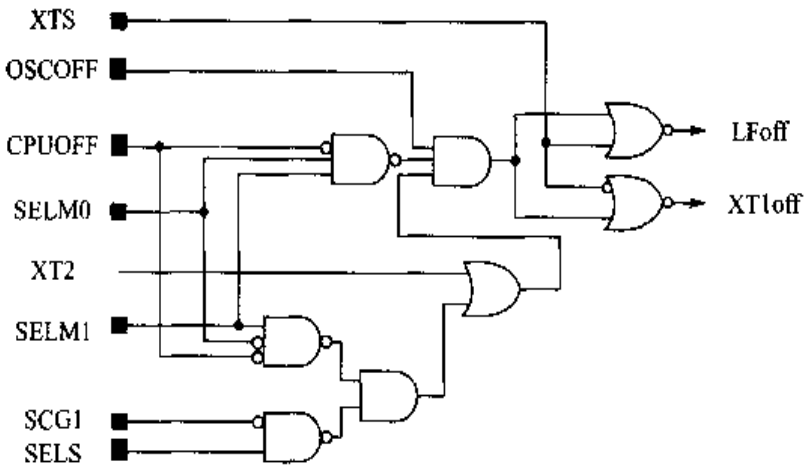


图 2.19 LFXT1 的结构图

其中，XT2 是内部信号：

XT2=0：MSP430X11XX、MSP430X12XX 序列；

XT2=1：MSP430X15XX、MSP430X16XX、MSP430X15XX、MSP430X16XX 序列。

2. 高频振荡器 LFXT2

高频振荡器 LFXT2 的结构如图 2.20 所示。LFXT2 作为 MSP430 的第二晶体振荡器。与低频相比，其功耗更大。

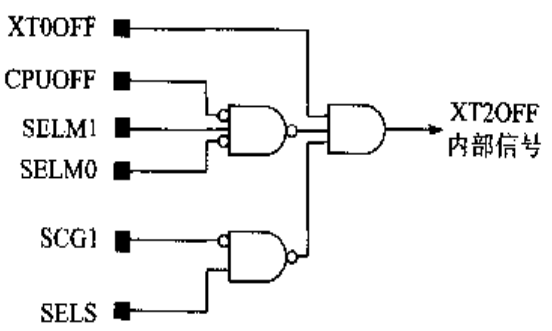


图 2.20 LFXT2 的结构图

高频晶体振荡器需要外接在 XIN2 和 XOUT2 两个引脚，并且必须外接电容。高频振荡器可以作为 SMCLK 和 MCLK 的时钟源。

高频振荡器可以选择从 XIN2 引脚接入一个外部输入时钟信号，但是所接的频率必须根据所设定的工作模式来选择，并且 XT2OFF 位必须复位。

3. 数字控制振荡器 DCO

DCO 振荡器的结构如图 2.21 所示。

DCO 是内部集成的 RC 类型振荡器。DCO 的频率会随温度和电压的变化而变化。

DCO 的频率精度比较差，但可以通过软件设置 DCOx、MODx 和 RSLEx 等位来调整其频率，从而增加它的稳定性。

当 DCO 不是作为 SMCLK 或 MCLK 的时钟源时，可以通过设置 SCG0 位来使 DCOCLK 失效。

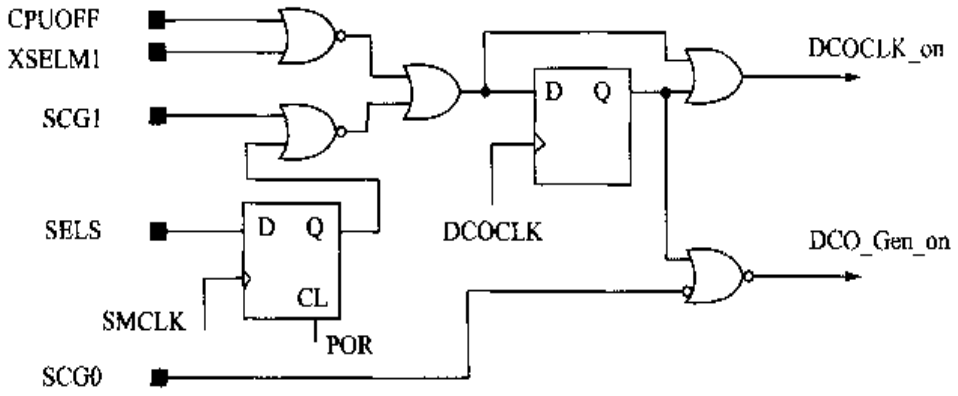


图 2.21　DCO 的结构图

当系统经复位(PUC)后，DCO 被选择为系统的 MCLK 和 SMCLKD 时钟源，RC 选择为内部电阻产生振荡。RSELx=3，DCOx=4，DCO 频率处于中间频率。

DCO 频率设置是经过以下几步来实现的：

① 通过设置 DCOR 位来选择是外部或内部电阻；

② 通过外部电阻或内部电阻来确定一个基准频率；

③ 通过设置 3 个 RSELx 位来分频；

④ 通过设置 3 个 DCOx 位来选择频率；

⑤ 通过 5 个 MODx 位来选择 DCOx 和 DCOx+1 之间的频率。

当 DCOx=07H 时，MODx 位对选择没有效果，因为已经达到最高频率了。

图 2.22 所示为 DCOx 和 RSELx 位对 DCO 频率调整的示意图。

DCO 可以工作在很高的频率上，内部电阻大约 300 kΩ，此时 DCO 的工作频率大约为 5 MHz。当使用一个 100 kΩ 的外部电阻时，此时 DCO 的工作频率可以达到 10 MHz。

需要注意的是，MCLK 不能高于该器件所规定的最高频率。DCO 的调制器混合 DCO 到 DCO+1 的频率产生一个稳定有效的中频频率来减小电磁干扰。这个功能是通过设置 MODx 位来实现的。当 MODx 位为 0 时，该功能失效。

而实际 DCO 周期 t 的计算公式为：

$$t = (32 - \mathrm{MODx}) \times t_{\mathrm{DCO}} + \mathrm{MODx} \times t_{\mathrm{DCO+1}}$$

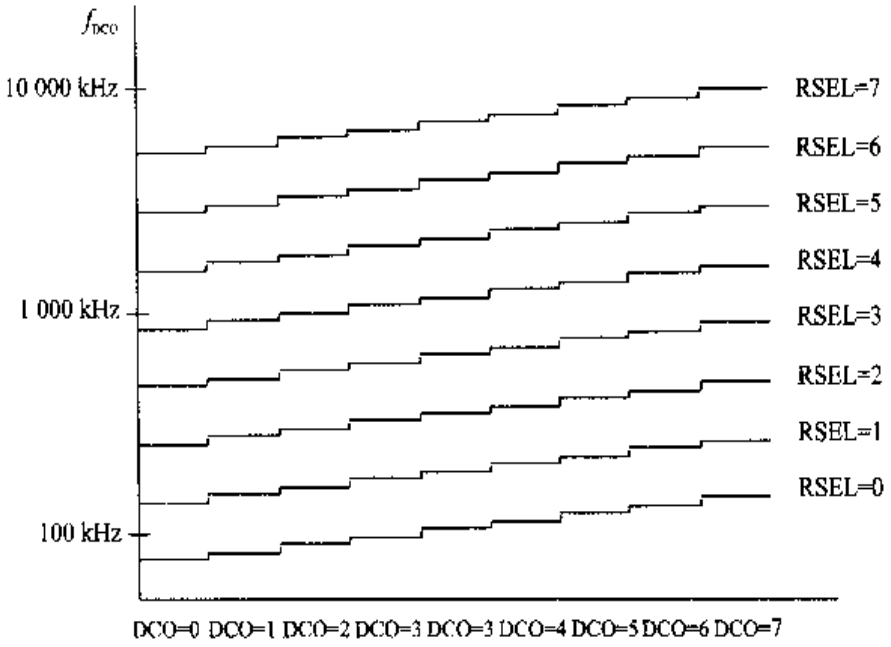


图 2.22　DCOx 和 RSELx 位对 DCO 频率的调整

2.4.2　时钟模块的寄存器

下面介绍时钟设置的相关寄存器。

1. DCOCTL 寄存器

DCOCTL 是一个 8 位 DCO 控制寄存器。该寄存器的位分配如图 2.23 所示。其中，DCO.0～DCO.2 选择内部 DCO 产生的 8 种频率之一，可以分段调节 DCOCLK 频率，相邻两种频率相差 10%。而频率由注入直流发生器的电流定义。

MOD.0～MOD.4 定义在 32 个 DCO 周期中插入的频率等于 F_{DCO+1} 周期个数。如果 DCO 常数为 7，表示已经选择最高频率，此时不能利用 MOD.0～MOD.4 进行频率调整。

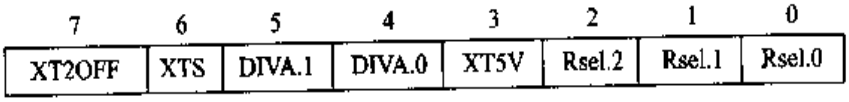
7	6	5	4	3	2	1	0
DCO.2	DCO.1	DCO.0	MOD.4	MOD.3	MOD.2	MOD.1	MOD.0

图 2.23　DCOCTL 寄存器

2. BCSCTL1 寄存器

BCSCTL1 是一个 8 位的基本时钟系统控制寄存器 1。该寄存器的位分配如图 2.24 所示。

下面对各个位分别进行介绍。



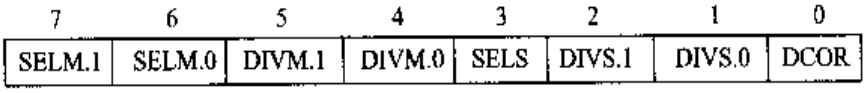
7	6	5	4	3	2	1	0
XT2OFF	XTS	DIVA.1	DIVA.0	XT5V	Rsel.2	Rsel.1	Rsel.0

图 2.24　BCSCTL1 寄存器

XT2OFF	Bit7	控制 XT2 振荡器的开启与关闭。 0　XT2 振荡器开启； 1　XT2 振荡器关闭。
XTS	Bit6	控制 LFXT1 工作模式，选择需结合实际晶体振荡器连接情况。 0　LFXT1 工作在低频模式(默认)； 1　LFXT1 工作在高频模式(必须连接与高频相应的高频时钟源)。
DIVA.1,DIVA.0	Bits5～4	控制 ACLK 分频。 00　不分频(默认)； 01　2 分频； 10　4 分频； 11　8 分频。
XT5V	Bit3	此位设置为 0。
Resl1.0,Resl1.1,Resl1.2	Bits2～0	这 3 位控制某个内部电阻以决定标称频率。 000　选择最低的标称频率； … 111　选择最高的标称频率。

3. BCSCTL2 寄存器

BCSCTL2 是一个 8 位的基本时钟系统控制寄存器 2。该寄存器的位分配如图 2.25 所示。



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

图 2.25　BCSCTL2 寄存器

下面对各个位分别进行介绍。

SELM.1,SELM.0	Bits7～6	选择 MCLK 时钟源。 00　时钟源为 DCOCLK(默认)； 01　时钟源为 DCOCLK； 10　时钟源为 LFXT1CLK(对于 MSP430F11/

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

11　时钟源为 LFXTICLK。

DIVM.1,DIVM.0	Bits5～4	选择 MCLK 分频。 00　1 分频(默认)； 01　2 分频； 10　4 分频； 11　8 分频。
SELS	Bit3	选择 SMCLK 时钟源。 0　时钟源为 DCOCLK(默认)； 1　时钟源为 LFXT1CLK(对于 MSP430F11/12X)，时钟源为 XT2CLK（对于 MSP430F13/14/15/16X)。
DIVS.1,DIVS.0	Bits2～1	选择 SMCLK 分频。 00　1 分频(默认)； 01　2 分频； 10　4 分频； 11　8 分频。
DCOR	Bit0	选择 DCO 内阻。 0　内部电阻； 1　外部电阻。

4. IE1 寄存器

IE1 是一个 8 位的中断使能寄存器 1。

该寄存器的位分配如图 2.26 所示。

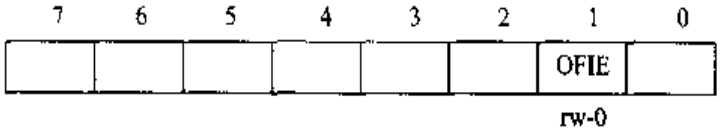


图 2.26　IE1 寄存器

下面对各个位分别进行介绍。

OFIE	Bit1	晶体出错中断使能控制位。 0　中断不允许； 1　中断使能。

5. IFG1 寄存器

IFG1 是一个 8 位的中断标志寄存器 1。该寄存器的位分配如图 2.27 所示。

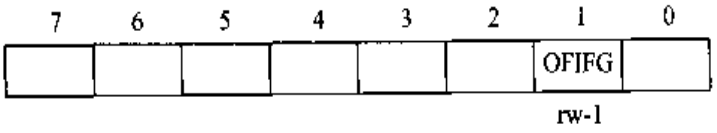


图 2.27　IFG1 寄存器

下面对各个位分别进行介绍。

OFIFG　Bit1　　晶体出错中断标志。复位值为 1。

0　　没有中断产生；

1　　有中断产生。

2.4.3　MSP430F149 时钟应用举例

MSP430F149 时钟应用的具体程序如下(本书中的所有程序都使用该时钟初始化程序，运行稳定，读者可按照需要自行修改)：

```
void int_clk()
{
    uchar i;
    BCSCTL1& = ~XT2OFF;                    //打开 XT 振荡器
    BCSCTL2| = SELM1 + SELS;               //MCLK 为 8 MHz,SMCLK 为 1 MHz
    {
        IFG1 &= ~OFIFG;                    //清除振荡错误标志
        for(i = 0; i < 100; i++)
    _NOP();                                //延时等待
        }
    while ((IFG1 & OFIFG) != 0);           //如果标志为 1,则继续循环等待
    IFG1& = ~OFIFG;
}
```

2.5　MSP430 的中断和特殊功能寄存器

2.5.1　中断结构和类型特点

中断是 MSP430 微处理器的一大特色。有效地利用中断可以简化程序，并且提高执行效

率和系统的稳定性。几乎 MSP430 系列单片机的每个外围模块都能产生中断，为 MSP430 针对事件(即外围模块产生的中断)进行的编程打下基础。MSP430 在没有事件发生时进入低功耗模式，事件发生时通过中断唤醒 CPU，事件处理完毕后 CPU 再次进入低功耗状态。由于 CPU 的运算速度和退出低功耗的速度很快，所以在应用中，CPU 大部分时间都处于低功耗状态，使得系统的整体功耗极大地降低。

MSP430 的中断源有很多，每个中断源对应有不同的优先级。

图 2.28 所示为 MSP430 的中断优先级结构。

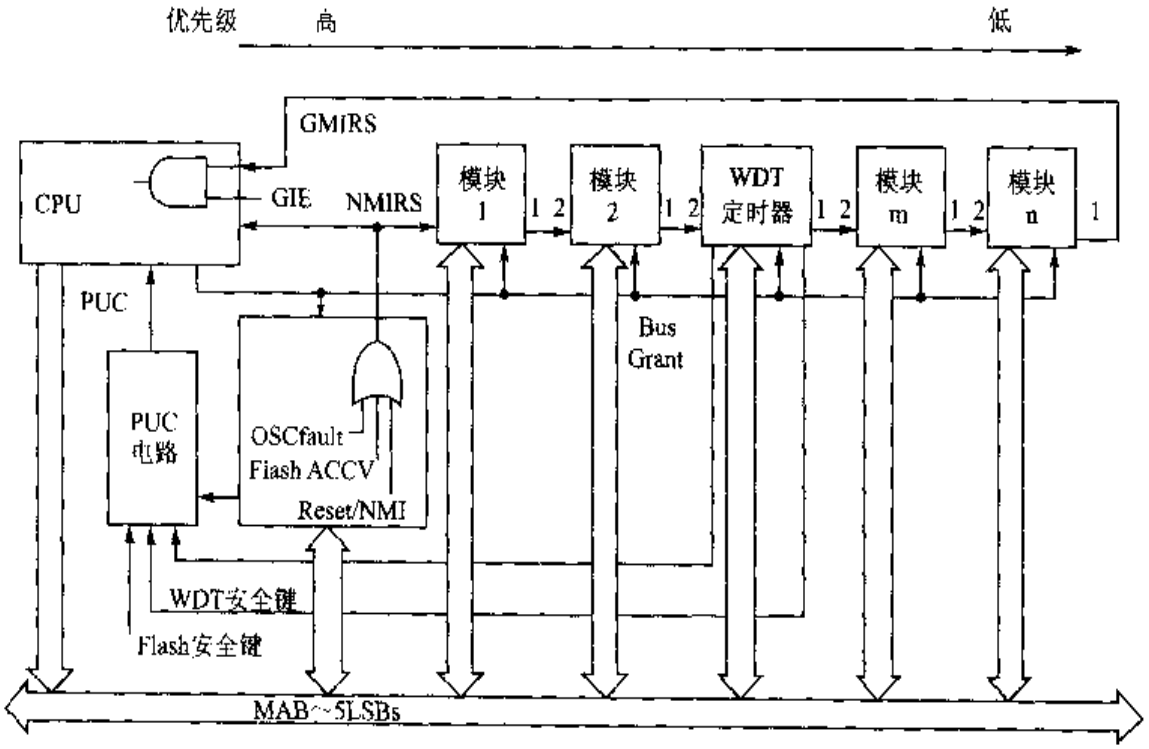


图 2.28 MSP430 的中断优先级结构图

MSP430 的中断分为 3 种：系统复位中断、不可屏蔽中断和可屏蔽中断。

1. 系统复位中断

在 2.3.1 小节中已经介绍过复位，这里需要注意的是系统复位后的地址在 0FFFEH。

2. 不可屏蔽中断

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

产生不可屏蔽中断有 3 种情况：上升沿电平出现在 NMI 引脚，当 NMI 引脚配置为 NMI

模式时；振荡器发生错误；Flash 的误操作。

当系统上电复位时，$\overline{RST}$/NMI 引脚被设置为复位模式。$\overline{RST}$/NMI 引脚选择$\overline{RST}$或者 NMI 功能是通过设置看门狗的寄存器 WDTCTL 来实现的。

当此引脚设置为复位状态时，CPU 会保持复位状态。当该引脚保持低电平且输入变成高电平时，CPU 开始从 0FFFEH 所指定的起始地址处执行程序。

当此引脚配置成 NMI 功能时，如果有一个边沿信号（上升沿或下降沿）被 WDTNMIES 位选择并且 NMIIE 中断允许，将会产生 NMI 中断，并且会产生 NMI 中断标志 NMIIFG。

当此引脚配置成 NMI 模式时，一个 NMI 中断产生，不会锁定 NMI 引脚为低电平。如果此时有不同的原因产生 PUC 复位并且 NMI 引脚为低电平时，系统将保持复位状态，这是因为 PUC 复位后将该引脚配置为复位模式。

当此引脚配置成 NMI 模式时，修改 WDTNMIES 位是否产生中断取决于该引脚实际的电平。当 NMI 中断边沿选择位在配置为 NMI 模式前修改时，则不会产生中断。

Flash 误操作产生中断在 2.2 节 Flash 操作中有介绍。

3. 可屏蔽中断

可屏蔽中断的中断来源于具有中断能力的外围模块，包括看门狗定时器工作在定时器模式时溢出产生的中断。每一个中断都可以由各自的中断控制寄存器中的对应控制位进行屏蔽，也可以由全局中断控制位屏蔽。

2.5.2　中断的响应过程

多个中断请求发生时，响应最高优先级中断。响应中断时，MSP430 单片机会将不可屏蔽中断控制位 SR. GIE 复位。因此，一旦响应了中断，即使有优先级更高的可屏蔽中断出现，也不会中断当前正在响应的中断而去响应较高优先级的中断。但 SR. GIE 复位不影响不可屏蔽中断，所以仍可接收不可屏蔽中断的中断请求。

中断响应的过程如下：

① 如果 CPU 处于活动状态，则完成当前指令；

② 若 CPU 处于低功耗状态，则退出低功耗状态；

③ 将下一条指令的 PC 值压入堆栈；

④ 将状态寄存器 SR 压入堆栈；

⑤ 若有多个中断请求，则响应最高优先级中断；

⑥ 单中断源的中断请求标志位自动复位，多中断源的标志位不变，等待软件复位；

⑦ 总中断允许位 SR. GIE 复位；SR 状态寄存器中的 CPUOFF、OSCOFF、SCG1、V、N、Z、C 位复位；

⑧ 相应的中断向量值装入 PC 寄存器，程序从此地址开始执行。

中断返回的过程如下：

① 从堆栈中恢复 PC 值，若响应中断前 CPU 处于低功耗模式，则可屏蔽中断仍然恢复低功耗模式；

② 从堆栈中恢复 PC 值，若响应中断前 CPU 不处于低功耗模式，则从此地址继续执行程序。

2.5.3 中断向量地址和特殊功能寄存器

1. 中断向量地址

MSP430F149 的中断向量地址是从 0FFE0H 到 0FFFFH，每个向量包含了中断服务程序指令序列的 16 位地址。

表 2.3 所列为 MSP430F149 的中断向量表，不同型号的单片机可以参考其数据手册。

表 2.3 MSP430F149 的中断向量表

中断源	中断标志	系统中断	字地址	优先级
上电、外部复位、看门狗、Flash	WDTIFG，KEYV[1]	复位	0FFFEH	15(最高)
NMI、晶体出错、Flash 非法访问	NMIIFG，OFIFG，ACCVIFG[2][3]	(非)屏蔽	0FFFCH	14
Timer_B7	BCCIFG0[5]	可屏蔽	0FFFAH	13
Timer_B7	BCCIFG1 到 BCCIFG6[4][5]	可屏蔽	0FFF8H	12
比较器 A	CAIFG	可屏蔽	0FFF6H	11
看门狗定时器	WDTIFG	可屏蔽	0FFF4H	10
USART0 接收	URXIFG0	可屏蔽	0FFF2H	9
USART0 发送	UTXIFG0	可屏蔽	0FFF0H	8
ADC_12	ADCIFG[3][4]	可屏蔽	0FFEEH	7
Timer_A3	CCIFG0	可屏蔽	0FFECH	6
Timer_A3	CCIFG1，CCIFG2，TAIFG[3][4]	可屏蔽	0FFEAH	5
P1.1(8 个标志)	P1IFG0 到 P1IFG7[3][4]	可屏蔽	0FFE8H	4
USART1 接收	URXIFG1	可屏蔽	0FFE6H	3
USART1 发送	UTXIFG1	可屏蔽	0FFE4H	2
P2.1(8 个标志)	P2IFG0 到 P2IFG7[3][4]	可屏蔽	0FFE2H	1
			0FFE0H	0(最低)

注：① 多中断源标志；② 中断标志位于模块中；③ 不可屏蔽：即独立的或通用的中断允许位都不能够禁止中断事件；④ (不可)屏蔽：独立的中断允许位能够禁止中断事件，但通用中断允许位不能禁止；⑤ MSP430F14X 系列中 Timer_B7 有 7 个捕获/比较寄存器，Timer_A3 有 3 个捕获/比较寄存器。

由表 2.3 可以看出，中断具有不同的中断向量地址，也有不同的中断优先级，优先级按数字大小从高到低排列，优先级高的先进入中断；同一个中断也有不同的中断源，在写程序时一定要加以区分，否则在进行中断处理时会出现错误。

2. 特殊功能寄存器

(1) 中断使能寄存器 1

中断使能寄存器 1 是一个 8 位的寄存器，该寄存器主要是使能某些模块的中断功能，其存储地址为 00H。中断使能寄存器 1 中的位分配如图 2.29 所示。

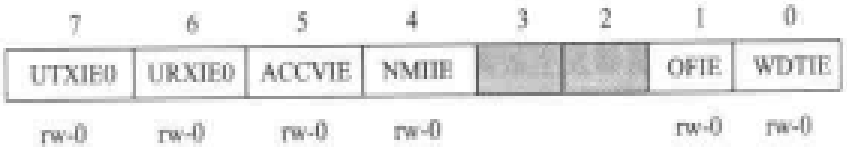


图 2.29　中断使能寄存器 1

下面对该寄存器中的各位进行介绍：

UTXIE0　Bit7　USART0，UART 和 SPI 发送中断使能信号。

URXIE0　Bit6　USART0，UART 和 SPI 接收中断使能信号。

ACCBIE　Bit5　Flash 访问错误中断使能信号。

NMIIE　Bit4　非屏蔽中断使能信号。

OFIE　Bit1　振荡器故障中断使能信号。

WDTIE　Bit0　看门狗定时器中断使能信号。

(2) 中断使能寄存器 2

中断使能寄存器 2 是一个 8 位的寄存器，该寄存器主要是使能某些模块的中断功能，其存储地址为 01H。中断使能寄存器 2 中的位分配如图 2.30 所示。

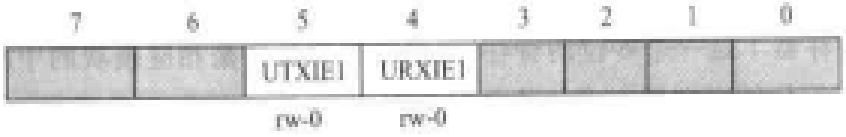


图 2.30　中断使能寄存器 2

下面对该寄存器中的各位进行介绍：

UTXIE1　Bit5　USART1，UART 和 SPI 发送中断使能信号；

URXIE1　Bit4　USART1，UART 和 SPI 接收中断使能信号。

(3) 中断标志寄存器 1

中断标志寄存器 1 是一个 8 位的寄存器，该寄存器主要是相应模块的中断标志位，其存储地址为 02H。中断标志寄存器 1 中的位分配如图 2.31 所示。

图 2.31 中断标志寄存器 1

下面对该寄存器中的各位进行介绍：

UTXIFG0 Bit7 USART0、UART 和 SPI 发送标志；

URXIFG0 Bit6 USART0、UART 和 SPI 接收标志；

NMIIFG Bit4 通过$\overline{\text{RST}}$/NMI 引脚置位；

OFIFG Bit1 振荡器发生故障时标志置位；

WDTIFG Bit0 溢出，或安全值错误，或 VCC 上电复位，或$\overline{\text{RST}}$/NMI 有复位条件时置位。

(4) 中断标志寄存器 2

中断标志寄存器 2 是一个 8 位的寄存器，该寄存器主要是相应模块的中断标志位，其存储地址为 03H。中断标志寄存器 2 中的位分配如图 2.32 所示。

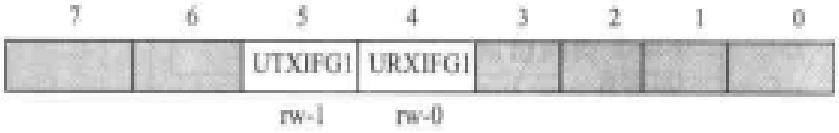


图 2.32 中断标志寄存器 2

下面对该寄存器中的各位进行介绍：

UTXIFG1 Bit5 USART1，UART 和 SPI 发送标志；

URXIFG1 Bit4 USART1，UART 和 SPI 接收标志。

(5) 模块使能寄存器 1

模块使能寄存器 1 是一个 8 位的寄存器，该寄存器主要是相应模块的中断标志位，其存储地址为 04H。模块使能寄存器 1 中的位分配如图 2.33 所示。

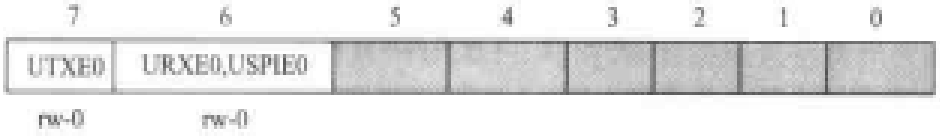


图 2.33 模块使能寄存器 1

下面对该寄存器中的各位进行介绍：

UTXE0 Bit7 USART0，UART 发送使能；

URXE0 Bit6 USART0，UART 接收使能；

USPIE0 Bit6 USART0，SPI(同步外设接口)发送和接收使能。

(6) 模块使能寄存器 2

模块使能寄存器 2 是一个 8 位的寄存器，该寄存器主要是相应模块的中断标志位，其存储地址为 05H。模块使能寄存器 1 中的位分配如图 2.34 所示。

图 2.34　模块使能寄存器 2

下面对该寄存器中的各位进行介绍：

UTXE1　Bit5　USART1,UART 接收使能；

URXE1　Bit4　USART1,UART 发送使能；

USPIE1　Bit4　USART1,SPI(同步外设接口)发送和接收使能。

2.6　看门狗定时/计数器

看门狗定时/计数器是一个 16 位多功能的定时器，可作为看门狗定时器使用，也可作为内部通用定时/计数器用。

2.6.1　基本介绍

看门狗定时/计数器的主要功能是当系统软件发生故障时不能正常复位看门狗，看门狗将产生一个复位信号使系统重新正常工作。如果系统不需要看门狗定时/计数器功能，这个模块可以用做内部定时并且可以产生可选的内部时钟中断。主要特点为：

- 4 个软件可选时间间隔；
- 看门狗模式；
- 内部时钟模式；
- 访问看门狗定时/计数器寄存器受密码保护，防止错误操作；
- 控制 RST/NMI 引脚功能；
- 时钟源可选；
- 在低功耗模式下可以禁止模块。

看门狗定时/计数器结构图如图 2.35 所示。

注意：上电后看门狗自动被使能。上电后看门狗被自动配置成使用 DCOCLK 的 32 ms 间隔复位，用户使用时必须在复位中断到时重新设置或者停止看门狗，否则系统会一直处于复位状态。

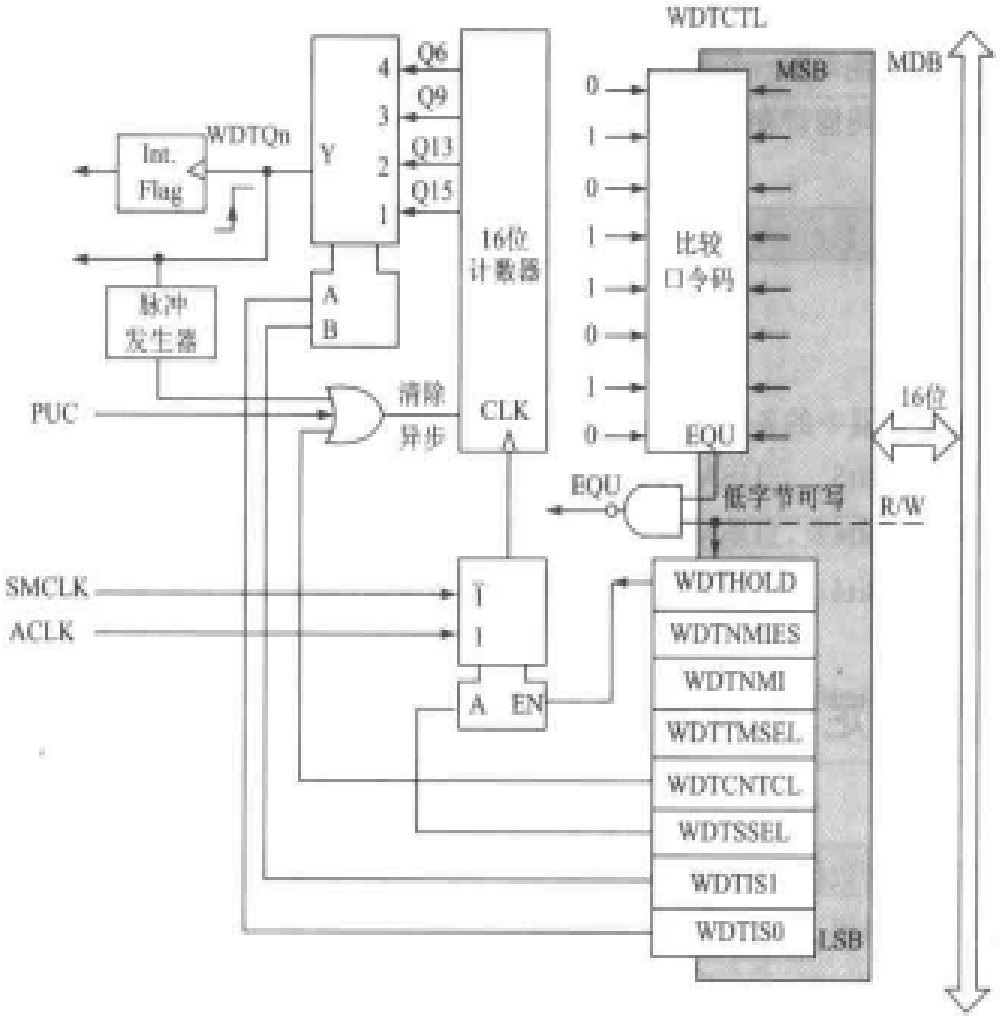


图 2.35　看门狗定时/计数器结构框图

2.6.2　看门狗时钟操作

通过 WDTCLT 寄存器，WDT 模块既可以配置成看门狗模式，也可以配置成内部时钟模式。WDTCTL 寄存器同时包含配置$\overline{\text{RST}}$/NMI 引脚的控制位，它是一个 16 位可读/写并且受密码保护的寄存器，不可以位寻址。当对 WDTCTL 写时，高位必须包括操作密码 05AH，否则就会产生一个系统复位信号，而不论工作在看门狗模式还是内部时钟模式；当对 WDTCTL 读时，高位总是 0x69H。

MSP430 序列有 5 种低功耗模式。在不同的低功耗模式下，使用不同的时钟信号。用户需求以及用户所选择的时钟类型决定了 WDT 模块如何被配置。例如，如果用户想使用 LPM3，则不能把 SMCLK 作为看门狗模式下的信号源，因为此时 SMCLK 不起作用，而且 WDT 不工作。如果用户不需要看门狗定时器，则 WDTHOLD 位可用来控制 WDTCNT，以降低功耗。

2.6.3　看门狗寄存器

1. WDTCTL 寄存器

WDTCTL 寄存器是一个 16 位的寄存器，看门狗的功能控制主要是通过设置该寄存器来实现的。WDTCTL 寄存器的各个位如图 2.36 所示。

15 … 8	7	6	5	4	3	2	1	0
WDTPW	WDTHOLD	WDTNMIES	WDTNMI	WDTTMSEL	WDTCNTCL	WDTSSEL	WDTISx	
rw−0 … rw−0	rw−0	rw−0	rw−0	rw−0	r0(w)	rw−0	rw−0	rw−0

图 2.36　WDTCTL 寄存器

下面将对 WDTCTL 寄存器的各个位进行详细介绍：

WDTPW　Bits15~8　看门狗密码位，通常读得到 0x69H，写时必须为 0x5AH，否则将系统复位。

WDTHOLD　Bit7　看门狗定时/计数器使能位。当不使用看门狗定时/计数器时，置该位为 1，可以节约系统功耗。

0　看门狗定时/计数器使能；

1　看门狗定时/计数器停止。

WDTNMIES　Bit6　看门狗定时器 NMI 边沿选择。当 WDTNMI=1 时，该位选择 NMI 中断的边沿方式，此时改变该位将触发一个 NMI。为了避免产生意外 NMI 中断，可以在 WDTNMI=0 时改变位。

0　NMI 上升沿有效；

1　NMI 下降沿有效。

WDTNMI　Bit5　看门狗定时 NMI 选择位，$\overline{\text{RST}}$/NMI 引脚功能选择位。

0　复位功能；

1　NMI 功能。

WDTTMSEL　Bit4　看门狗定时/计数器工作方式选择位。

0　看门狗模式；

1　内部定时模式。

WDTCNTCL　Bit3　看门狗定时/计数器清零，置 WDTCNTCL 为 1 将清零计数器。

0　不操作；

1　WDTCNT 清零。

WDTSSEL	Bit2	时钟选择。 0　SMCLK； 1　ACLK。
WDTISx	Bits1~0	看门狗或内部中断时间间隔选择。 00　看门狗时钟/32768； 01　看门狗时钟/8192； 10　看门狗时钟/512； 11　看门狗时钟/64。

2. IE1 寄存器

IE1 寄存器是一个 8 位的寄存器，主要用来进行中断使能设置。IE1 寄存器的各个位如图 2.37所示。

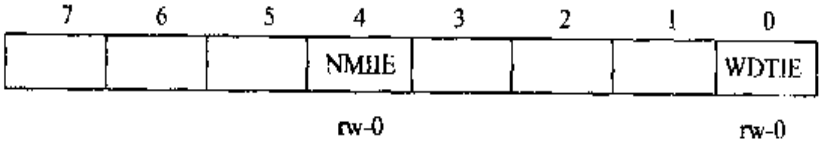


图 2.37　IE1 寄存器

下面将对 IE1 寄存器的各个位进行详细介绍：

NMIIE	Bit4	NMI 中断使能，由于 IE1 的其他位被其他模块使用，因此推荐使用 BIS.B 或者 BIC.B 指令来置位或清零该位。 0　中断不允许； 1　中断允许。
WDTIE	Bit0	看门狗定时器中断使能，在看门狗模式下不必置位该位。同样，由于 IE1 的其他位被其他模块使用，因此推荐使用 BIS.B 或者 BIC.B 指令来置位或清零该位。 0　中断不允许； 1　中断允许。

3. IFG1 寄存器

IFG1 寄存器是一个 8 位的寄存器，该寄存器主要用来进行中断标志位的设置。IFG1 寄存器的各个位如图 2.38 所示。

下面将对 IFG1 寄存器的各个位进行详细介绍：

NMIFG	Bit4	NMI 中断标志位，该位只能被软件复位。由于 IFG1 的其他位被其他模块使用，因此推荐使用 BIS.B 或者 BIC.B 指令来清零该位。

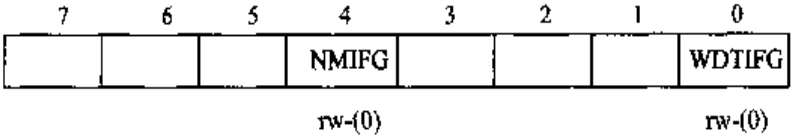


图 2.38　IFG1 寄存器

0　没有产生中断；

1　中断挂起。

WDTIFG　Bit0　看门狗定时器中断标志位在看门狗模式下，该位保持“1”直到被软件复位。在内部定时模式下，WDTIFG 被自动复位。同样，由于 IFG1 的其他位被其他模块使用，因此推荐使用 BIS.B 或者 BIC.B 指令来清零该位。

0　没有产生中断；

1　中断挂起。

2.6.4　基本应用举例

通过对看门狗模块的相关寄存器的介绍，读者对它有了一个大致了解。下面举例说明看门狗的具体操作。主要完成 3 个功能：关闭看门狗、使用内部定时模式及使用看门狗模式。

```
WDTCTL = WDTPW + WDTHOLD;                          //关闭看门狗
WDTCTL = WDTPW + WDTCNTCL;                         //看门狗复位
WDTCTL = WDTPW +  WDTCNTCL + WDTTMSEL + WDTIS0;    //内部定时模式，时间间隔为看门狗时钟/8192
```

2.7　MSP430F149 的最小系统

前面对 MSP430F149 作了一个基本的概述。通过以上章节的学习，使我们对 MSP430F149 的引脚、结构、特点、时钟、复位、存储器组织、Flash 模块操作、中断、看门狗等已经基本了解。本节将介绍 MSP430F149 的最小系统。与其他单片机相比（如 51 单片机），MSP430F149 有很大的 Flash 的空间，所以在最小系统中我们不用外扩存储器。MSP430F149 的最小系统只有四部分：电源、复位电路、晶体及简易仿真器 JTAG 下载线。

2.7.1　电　源

由于整个系统采用 5 V 和 3.3 V 供电，又考虑到硬件系统要求电源具有稳压功能和纹波

小的特点，另外也考虑到硬件系统的低功耗等特点，因此该硬件系统的电源先用 LM7805 稳压为 5 V 给外围模块电路供电，再用 SPX1117 芯片稳压得到 3.3 V 电压，给 CPU 和 3.3 V 外设供电。

电源电路如图 2.39 所示。

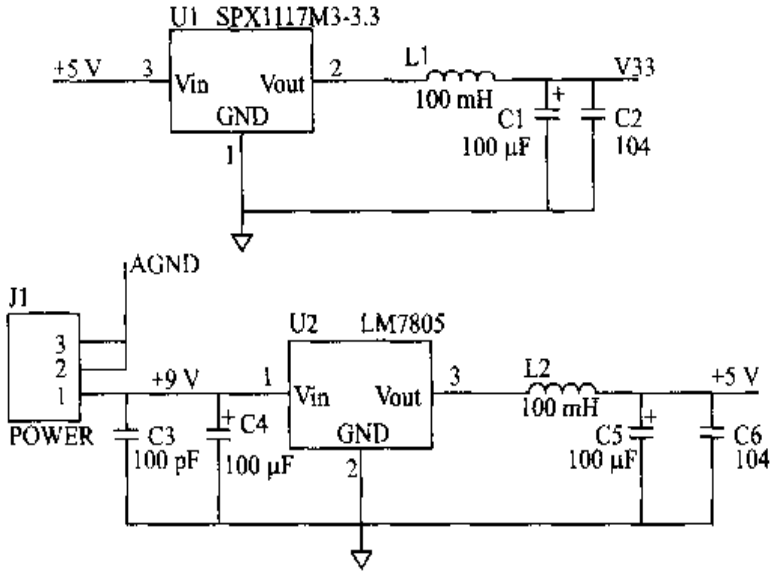


图 2.39 电源电路

为了使电源输出的纹波较小和降低其对整个系统的干扰，在输入端和输出端连接两个电容。(100 μF 的电解或钽电容和 0.1 μF 瓷片电容)滤除高频和低频干扰，在输出端得到稳定的直流工作电压。

2.7.2 复位电路

在包含了控制器的电路系统中，都有对应的复位电路，这样能使系统上电后很好地复位，使其处于稳定的运行状态。

一般简单的复位电路可以采用 RC 复位，但是很不稳定，可靠性不高，尤其在高速和庞大的系统中是根本不能采用的。

在 MSP430 系统中，我们都采用专门的复位芯片来提高系统的稳定性和可靠性。在这里采用 SP708S，该芯片提供可靠的复位。

图 2.40 所示为复位电路图。

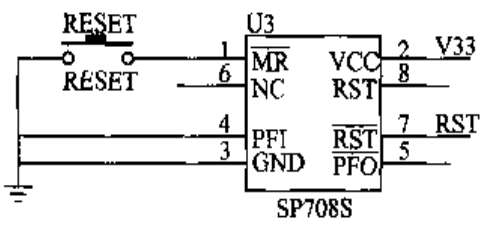


图 2.40 复位电路图

2.7.3　晶　体

2.4 节的系统时钟中已经介绍过了 MSP430 单片机的晶体连接方法，在这里我们只介绍一种很常用的连接方式：XIN1 和 XOUT1 连接时钟晶体低速晶体 32 768 Hz，XIN2 和 XOUT2 连接8 MHz的高速晶体。如图 2.41 所示。

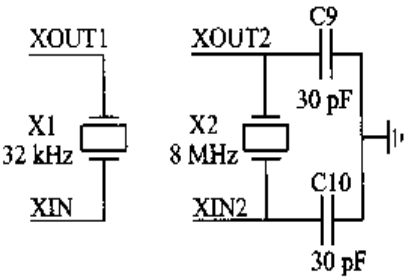


图 2.41　晶体连接图

2.7.4　简易仿真器 JTAG 下载线介绍

MSP430 的程序下载方式支持串行在线编程、系统可编程 ISP、JTAG 下载等。硬件仿真可以采用 JTAG 下载线，又因为 IAR 软件支持 JTAG 下载线在线调试，且可设置断点。现在市面上也有很多仿真器，大部分都是通过 JTAG 进行仿真调试的。下面介绍一种简易的 JTAG 下载线制作方法和与 MSP430 单片机的连接图。读者可以通过下面的电路图自己制作 MSP430 的简易仿真器。图 2.42 所示为 JTAG 的原理图，该简易仿真器是通过 PC 机的并口仿真(电路图见本书配套光盘)。

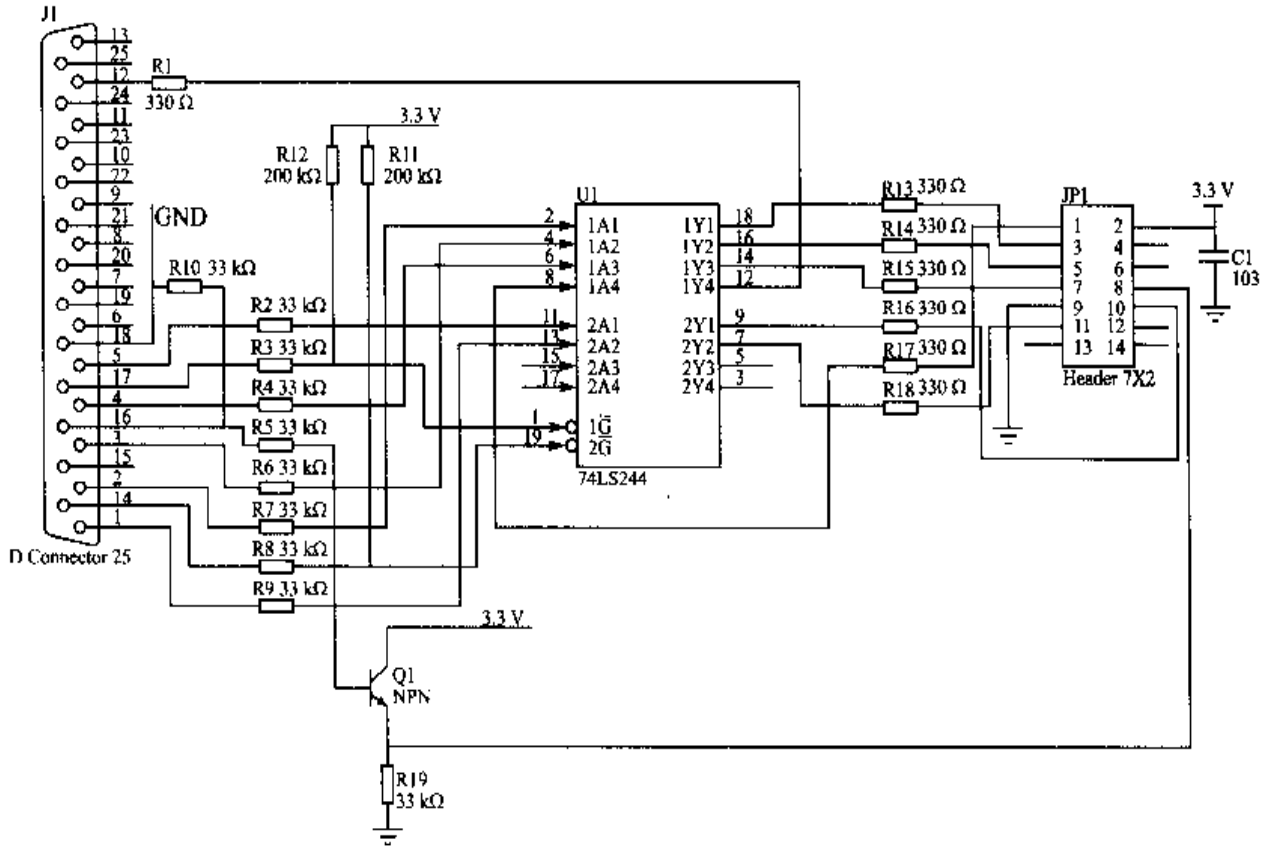


图 2.42　JTAG 与并口相连的原理图

图 2.43 所示为 MSP430 单片机的 JTAG 连接图，可见 JTAG 接口是一个 14 引脚的双排插座，JTAG 边界扫描的主要信号为 TDO、TDI、TMS、TCK 和 RST 信号。

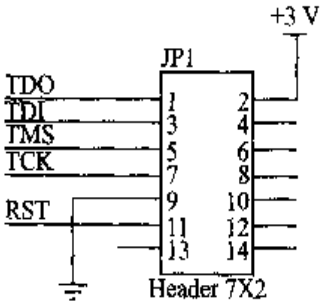


图 2.43 MSP430 的 JTAG 连接图

第3章

I/O口基本操作

MSP430F149单片机系列有6组I/O口：P1～P6。每个I/O口都有8个可以独立编程的引脚。MSP430F149单片机每个引脚都有第二功能，并且每个引脚可以单独设置成输入、输出或者第二功能。同时，P1和P2口具有中断功能，P1和P2口的每个引脚都可以单独设置成中断，并且都可以单独设置成上升沿或者下降沿触发中断。P1口的所有引脚共用一个中断向量，P2口的所有引脚也共用一个中断向量，但它们的优先级和向量地址不一样，见2.5节系统中断。

MSP430F149系列单片机的I/O口主要有以下特征：

- 每个I/O口可以位独立编程设置；
- 输入、输出可以任意结合使用；
- P1和P2口的中断功能位可以单独设置；
- 有独立的输入/输出寄存器。

3.1 I/O口的基本操作流程

MSP430F149的I/O口操作都是通过读/写寄存器来完成的。与MSP430F149单片机的I/O口中相关的寄存器有PxDIR、PxIN、PxOUT、PxSEL、PxIFG、PxIE和PxIES，其中后面三个是中断寄存器。P3、P4、P5、P6不具备中断功能，只与前面四个寄存器有关；P1、P2具有中断功能，与上面所有的寄存器都有关系，详细请参考3.2节寄存器说明部分。

当I/O口作为一般的输入/输出口使用时，其基本操作流程如下：

① 选择I/O口功能，基本I/O模式或其他模式(即设置PxSEL寄存器)；

② 设置方向寄存器(PxDIR)；

③ 读出外部输入值(PxIN)或写入相应值(PxOUT)；

对 I/O 中断操作的基本流程如下：

① 设置 I/O 模式；

② 设置中断触发方式(PxIES 寄存器)；

③ 允许中断(PxIE 寄存器)；

④ 开总中断(调用_EINT()或_BIS_SR(LPM4_bits + GIE)函数)；

⑤ 等待中断，有中断时执行中断服务程序。

3.2 寄存器

1. 功能选择寄存器 PxSEL

PxSEL(x=1,2,3,4,5,6)，设置相应的位为 1，则该位对应的引脚为外围模块的功能，即第二功能；设置相应的位为 0，则该位对应的引脚为普通 I/O 口。PxSEL 寄存器的各位如图 3.1 所示。其复位值全为 0，默认为 I/O 口功能。

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

图 3.1 PxSEL 寄存器的各位

注意：为了避免对某些位操作的时候影响其他的位，推荐使用位操作指令进行读/写，请参考 3.3 节具体的程序设计实例。

2. 方向控制寄存器 PxDIR

PxDIR(x=1,2,3,4,5,6)：该寄存器控制 Px 口的各个引脚的方向。设置相应的位为 1，则该位对应的引脚为输出；设置相应的位为 0，则该位对应的引脚为输入。PxDIR 寄存器的各个位如图 3.2 所示。其复位值全为 0。

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

图 3.2 PxDIR 寄存器的各位

3. 输出控制寄存器 PxOUT

PxOUT(x=1,2,3,4,5,6)：该寄存器控制 I/O 口的输出值。在输出模式下，设置相应的位为 1，则该位对应的引脚输出为高电平 1；设置相应的位为 0，则该位对应的输出为低电平 0。PxOUT 寄存器的各个位如图 3.3 所示。

注意：PxOUT 复位时其值不确定，在使用过程中应该先使 PxOUT 的值确定以后才设置方向寄存器。

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

图 3.3 PxOUT 寄存器的各位

4. 输入状态寄存器 PxIN

PxIN(x=1,2,3,4,5,6):该寄存器反映 I/O 口的输入值。在输入模式下,当 I/O 口相应输入为高电平时,该寄存器相应的位为 1;当 I/O 口相应输入为低电平时,该寄存器相应的位为 0。

PxIN 寄存器的各个位如图 3.4 所示。其复位值为随机值,该寄存器为只读寄存器,写无效。

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

图 3.4 PxIN 寄存器的各位

5. 中断允许寄存器 PxIE

PxIE(x=1,2):该控制器控制 I/O 口的中断允许。设置相应的位为 1,则该位对应的引脚允许中断;设置相应的位为 0,则该位对应的引脚不允许中断。PxIE 寄存器的各个位如图 3.5 所示。其复位值全为 0,默认为不允许中断。

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

图 3.5 PxIE 寄存器的各位

6. 中断触发选择寄存器 PxIES

PxIES(x=1,2):该寄存器控制 I/O 口的中断边沿选择。设置相应的位为 1,则该位对应的引脚选择下降沿触发中断;设置相应的位为 0,则该位对应的引脚选择上升沿触发中断。PxIES 寄存器的各个位如图 3.6 所示。其复位值全为 0,默认为上升沿触发中断。

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

图 3.6 PxIES 寄存器的各位

7. 中断标志寄存器 PxIFG

PxIFG(x=1,2):该寄存器为 I/O 口的中断标志寄存器。若相应的位为 1,则该位对应的引脚有外部中断产生;若相应的位为 0,则该位对应的引脚没有外部中断产生。PxIFG 寄存器的各个位如图 3.7 所示。其复位值全为 0,该寄存器必须通过软件复位,同时也可以通过软件写 1 产生相应中断。

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

图 3.7　PxIFG 寄存器的各位

注意：

- 当设置为第二功能的时候，MSP430 不会主动设置引脚的方向，必须通过 PxIN 寄存器去设置。
- 当设置为输出时，输入到外设的信号是锁存后的信号。当 PxSELx=1 时，内部输入信号跟随引脚端的信号；当 PxSELx　0 时，在 PxSELx 被复位以前，输入到外设的信号保持外设输入信号的值。
- 当 P1SEL 和 P2SEL 寄存器的位设置为 1，则 P1、P2 口的 I/O 中断功能失效。
- 在 P1 或 P2 口进行中断功能允许时，在进入中断程序时必须用软件对中断标志清零，中断标志不可以自动清零。

3.3　基本应用设计举例

在本书的配套实验板上，使用 MSP430F149 单片机的 P5 口扩展了一个 4×4 的键盘，同时采用 SPI 接口扩展了 4 位 LED 显示器。本例主要实现键值的循环显示。数码管显示的电路和程序在第 5 章的 SPI 模式中有详细介绍。这里主要介绍键盘键值的读取，目的是希望读者通过本实例来掌握 MSP430F149 单片机的 I/O 口的基本使用方法和键盘扫描的基本方法。

键盘接口电路如图 3.8 所示。

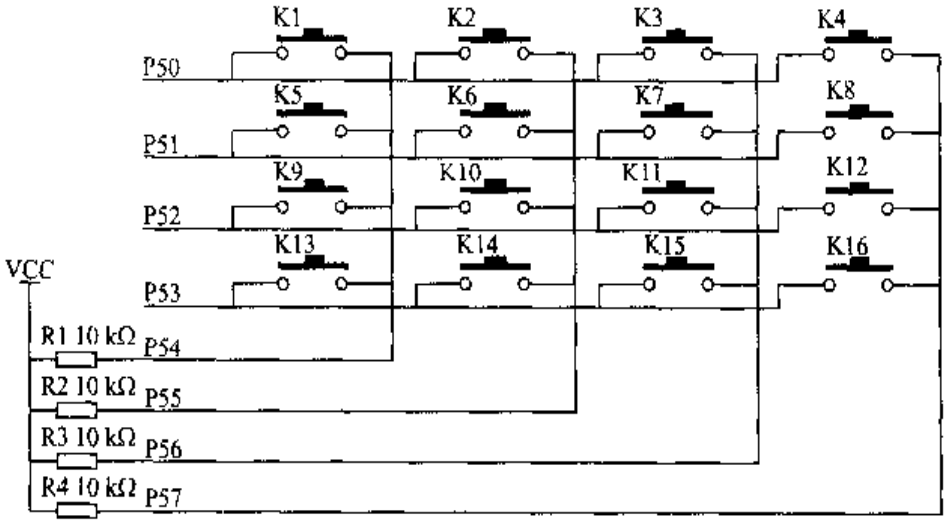


图 3.8　键盘接口电路(P5)

软件流程如图 3.9 所示。

具体程序如下：

```
#include <MSP430x14x.h>
```

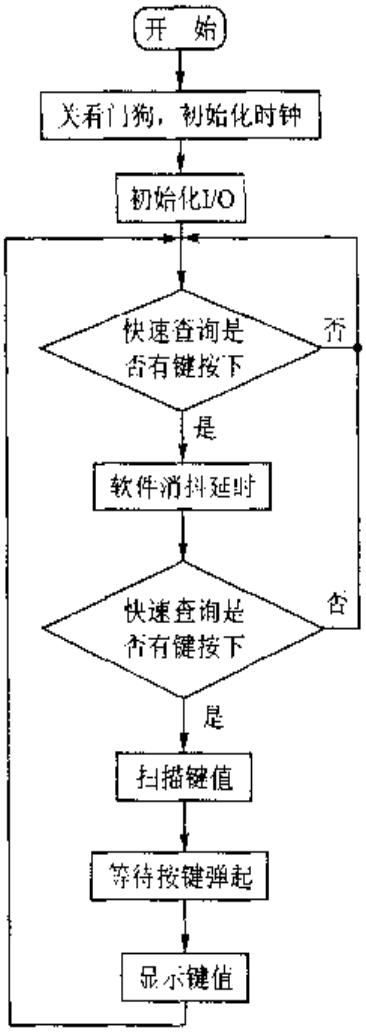


图 3.9 行列式键盘扫描程序流程图

```
#define uchar unsigned char
#define uint  unsigned int
#define led   BIT7
const uchar  segment[10] = {0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
uchar static key_flag = 0;
uchar static  count = 0;
uchar key1;
uchar kk;
void int_clk()
{
    uchar i;
    BCSCTL1& = ~XT2OFF;                    //打开 XT 振荡器
    BCSCTL2| = SELM1 + SELS;               //MCLK 为 8 MHz,SMCLK 为 1 MHz
    do
    {
```

```
        IFG1 &= ~OFIFG;                    //清除振荡错误标志
        for(i = 0; i < 100; i++)
            _NOP();                        //延时等待
        }
    while ((IFG1 & OFIFG) != 0);           //如果标志为1,则继续循环等待
    IFG1&= ~OFIFG;
}
delay()
{
  uint i;
  for(i=0;i <= 1000;i++)
       ;
}
void int_spi()
{
  U0CTL| = SWRST;
  U0CTL = CHAR + SYNC + MM;
  U0TCTL| = SSEL1 + SSEL0 + STC;           //三线模式 SMCLK 作为 SPI 时钟
  U0TCTL| = CKPH;
  U0TCTL& = ~CKPL;                         //CKPL CKPH:01 上升沿写数据
  U0BR0 = 0X02;
  U0BR1 = 0X00;
  U0MCTL = 0X00;
  ME1| = USPIE0;                           //使能 SPI
  U0CTL& = ~SWRST;
  IE1& = ~UTXIE0;
  IE1& = ~URXIE0;                          //禁止中断
  P3SEL| = 0x0E;
  P3DIR  = BIT2 + BIT4;                    //选择第二功能和 I/O 方向
}
write_byte(uchar byte)
{
  U0TXBUF = byte;
  while((IFG1&UTXIFG0) == 0);              //UTXIFG0 = 0,表示数据从发送
  IFG1& = ~UTXIFG0;
}
display(uchar data)
{
  P2DIR' = led;                            //对 I/O 口寄存器操作一般使用逻辑"与"运算,防止改变其他
                                           //位的值(本书程序均采用此方式,读者可以参考)
  P2OUT| = led;                            //使能 164CLK
  write_byte(segment[data]);               //写数据
```

```
  P2OUT& = ~led;                    //禁止 164CLK
}
//扫描一行根据参数 line 决定扫描具体哪一行
uchar scan_key(uchar line)
{
  uchar key;
  P5OUT = line;
  switch (P5IN&0Xf0)
  {
    case 0xe0:key = 1 + line;break;
    case 0xd0:key = 2 + line;break;
    case 0xb0:key = 3 + line;break;
    case 0x70:key = 4 + line;break;
    default:key = 0xff;break;
  }
  return(key);
}
void fast_scan()
{
  uchar temp;
  P5OUT& = 0XF0;                    //低位全部为 0,判断有无键按下
  temp = P5IN&0xf0;
  if(temp == 0xf0)key_flag = 0;
  else
  {
    key_flag = 1;                   //如果有键按下则标志为 1,否则为 0
  }
}
void over()
{
  uchar temp;
  P5OUT& = 0XF0;                    //低位全部为 0,判断有无键按下
  do
  {
    temp = P5IN&0xf0;
    if(temp != 0xf0)break;          //等待按键弹起
  }
  while(1);
}
//读取键值
uchar get_key()
```

```
{
  uchar key;
  while(1)
  {
    key = scan_key(0xe);
    if(key! = 0xff){key| = 0x80;P5OUT = 0X0f; break;}
    key = scan_key(0xd);
    if(key! = 0xff){key| = 0x40;P5OUT = 0X0f;break;}
    key = scan_key(0xb);
    if(key! = 0xff){key| = 0x20;P5OUT = 0X0f;break;}
    key = scan_key(0x7);
    if(key! = 0xff){key  = 0x20;P5OUT = 0X0f;break;}
  }
  switch(key)
  {
    case 0x8f:key = 0;break;
    case 0x90:key = 1;break;
    case 0x91:key = 2;break;
    case 0x92:key = 3;break;
    case 0x4E:key = 4;break;
    case 0x4F:key = 5;break;
    case 0x50:key = 6;break;
    case 0x51:key = 7;break;
    case 0x2C:key = 8;break;
    case 0x2D:key = 9;break;
    case 0x2E:key = 10;break;
    case 0x2F:key = 11;break;
    case 0x28:key = 12;break;
    case 0x29:key = 13;break;
    case 0x2A:key = 14;break;
    case 0x2B:key = 15;break;
    default:key = 0xff;break;
  }
  _NOP();
  return(key);
}
delay2()
{
```

```
  uint i = 0,j = 0;
  for(i = 0;i <= 200;i++)
  {
    for(j = 0;j <= 300;j++)
          ;
          }
}
int main( void )
{
  WDTCTL = WDTPW + WDTHOLD;          //关闭看门狗
  int_clk();                         //初始化时钟
  P5DIR = BIT0 + BIT1 + BIT2 + BIT3; //键盘初始化
  int_spi();                         //初始化 SPI
  display(0);
  display(0);
  display(0);
  display(0);
  while(1)
  {
    fast_scan();
    if(key_flag == 0) continue;      //判断是否有键按下,是则消抖,否则退出本次扫描
    key_flag = 0;
    delay2();
    fast_scan();
    if(key_flag == 1)
    {
      key_flag = 0;
      key1 = get_key();
      over();
      delay2();
      display(key1);
    }
  }
}
```

通过以上介绍可以看出,使用 MSP430 单片机进行行列式按键的扫描程序设计和其他单片机相似,完成按键扫描的程序编写,读者能够充分掌握 MSP430 单片机的基本 I/O 口的操作和使用。

第 4 章

Timer_A 和 Timer_B

4.1 定时器基本介绍

Timer_A 的结构框图如图 4.1 所示。它是一个 16 位的定时/计数器，拥有 3 个捕捉/比较

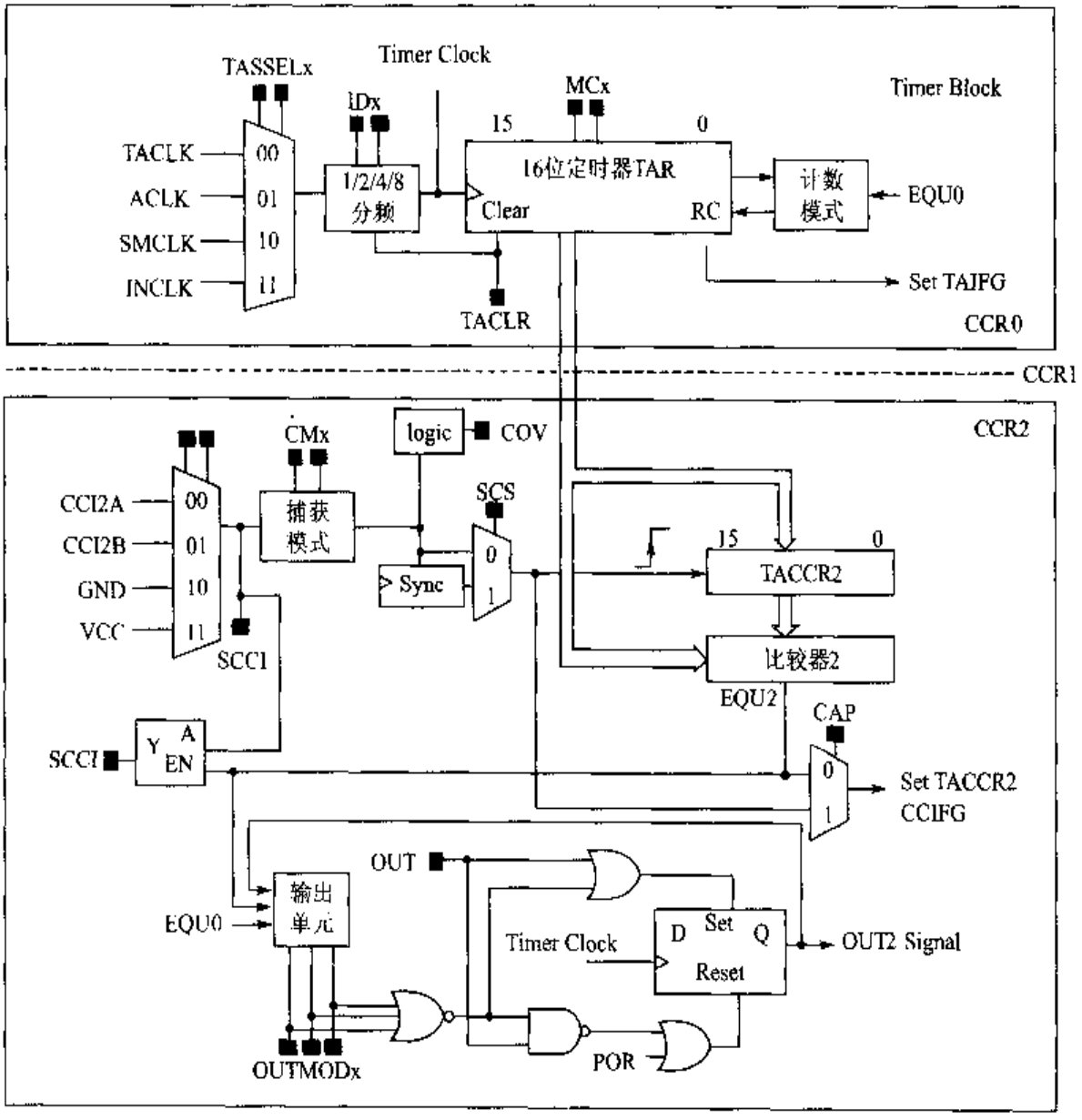


图 4.1 Timer A 结构框图

寄存器。Timer_A 支持多捕捉/比较、PWM 输出、内部定时。同时 Timer_A 拥有强大的中断功能。当计数器的溢出条件满足或者捕捉/比较寄存器条件满足时，都可产生中断。Timer_A 的主要特点如下：

- 16 位异步定时 计数器，有 4 种操作模式；
- 时钟源可选；
- 3 个捕捉/比较寄存器；
- 可以配置成 PWM 输出；
- 异步输入/输出锁存；
- 中断向量寄存器提供快速 Timer_A 中断译码功能。

4.2 Timer_A 定时/计数模式和操作说明

4.2.1 定时/计数模式

TAR 寄存器是一个 16 位的定时/计数寄存器。在时钟的上升沿，TAR 的内容会根据不同的模式而增加或者减小，TAR 可以通过软件进行读/写；另外，当定时/计数器发生溢出时将产生中断。

TAR 寄存器可以通过置位 TACLR 位来清零，同时置位 TACLR 位可以使时钟分频清零。

1. 停止模式

定时器暂停并不发生复位，所有寄存器现行的内容在停止模式后都可用。当定时器暂停后又重新计数时，计数器将从暂停时的值开始，并且以暂停前的计数方向计数。例如，停止模式前，定时器工作于增/减计数模式，并且处于下降计数方向；停止模式后，定时器仍然工作于增/减计数模式，并从暂停前的状态开始，继续沿着下降沿开始计数。如果不需要这样，则可通过 TACTL 中的 CLR 控制位来清除定时器的方向记忆特性。

2. 增加模式

计数器的周期长度等于 TACCR0 的值加 1，当时间周期不同于 0FFFFH 时，使用增加模式。选择该模式时，计数器重复加计数，当计数器的值大于比较寄存器 TACCR0 里面的内容时，计数器就马上变成 0 重新开始计数，其过程如图 4.2 所示。

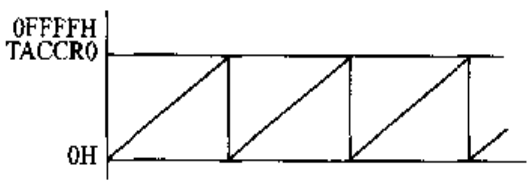


图 4.2 增加模式

如果在计数器的工作过程中改变 TACCR0 的内容即改变了计数周期，则当新的周期大于或等于以前的周期，或者大于当前计数器的值时，计数器将以新的周期作为计数周期。如果新的周期小于当前计数器的值，则计数器将变成 0，然后重新计数。但是当计数器变成 0 时会增加一个计数脉冲。

当计数器等于 TACCR0 时，TACCR0 的 CCIFG 中断标志置位。当计数器从 TACCR0 转到 0 时，会产生 TAIFG 中断标志。其工作时序如图 4.3 所示。

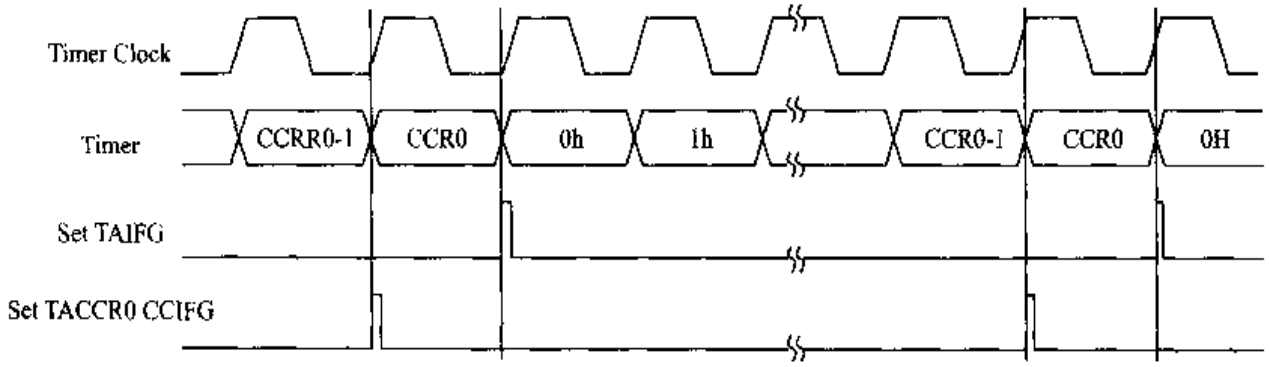


图 4.3　增加模式的工作时序图

定时器的定时周期 $T=$(1/定时器的频率)×(TACCR0 的值+1)。

3. 连续模式

连续模式下，计数器周期从 0H 到 0FFFFH。工作时序如图 4.4 所示。连续模式主要用于产生固定周期中断，同时配合比较寄存器可以用来产生 PWM。

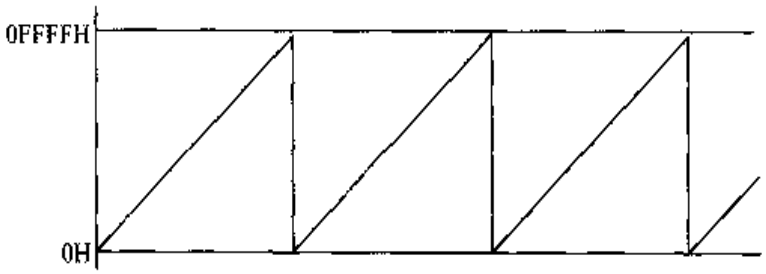


图 4.4　连续模式

当计数器的值从 0FFFFH 转到 0H 时，TAIFG 中断标志置位，时序如图 4.5 所示。

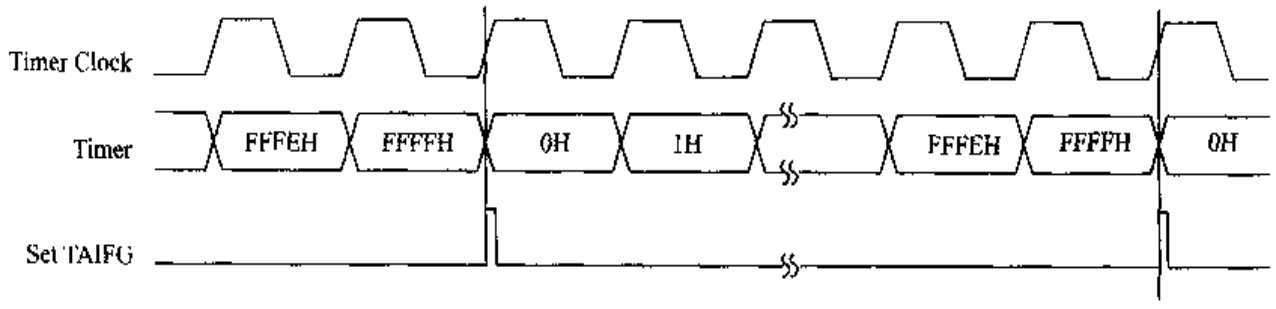


图 4.5　连续模式的工作时序图

下面介绍连续模式的使用和中断的产生。

连续模式主要用于产生独立的时间间隙和输出频率。图 4.6 表示了捕获/比较寄存器里有两个独立的时间间隔 t_0 和 t_1，这种情况下时间间隔将不受软件控制，而只受硬件控制。用所有的捕获/比较寄存器最多可以产生 3 个独立的时间间隔或输出频率。

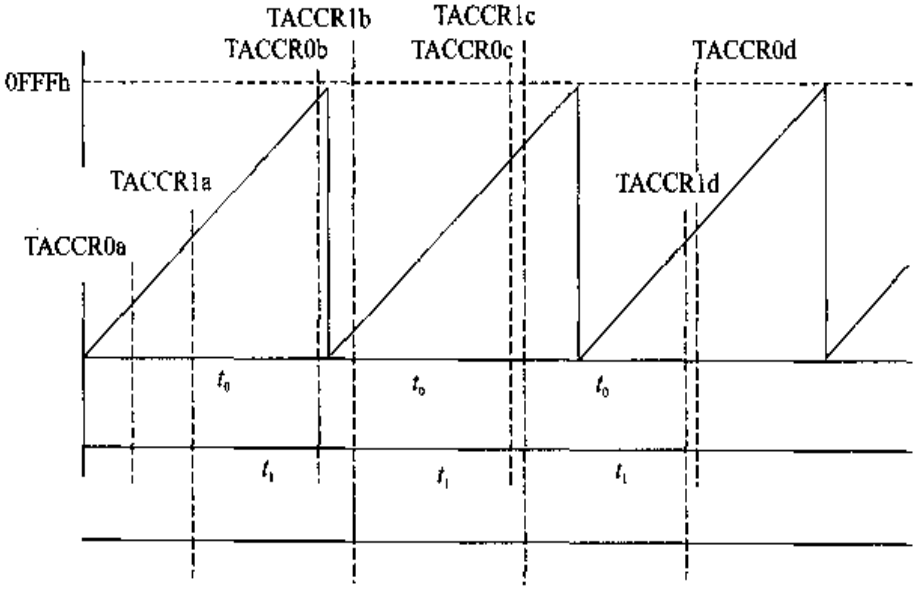


图 4.6 连续模式的中断

4. 加/减模式

加/减模式主要用于产生对称波形。计数器重复加计数到 TACCR0 的值，然后减计数到 0，所以周期等于 TACCR0 的 2 倍。

在计数过程中改变 TACCR0 的值有以下两种情况：

① 如果此时为减计数，则计数器计数到 0 后，新的周期才起作用；

② 如果此时为加计数，则又分为两种情况。如果新的周期值大于或等于当前计数器的值，那么计数器计数到新的周期值；如果新的周期值小于当前计数器的值，那么计数器将马上执行减计数。

注意：这个操作会增加一个额外的计数器时钟周期。

其工作过程如图 4.7 所示。

计数器的计数方向带锁存功能，这样就允许当用户停止计数以后重新开始计数时，计数方向不变。如果用户不需要此项功能，则可以通过置位 TACLR 来清零方向信号；同时 TAR 寄存器以及时钟分频也被清零。

中断标志 CCIFG、TAIFG 会在不同时间独立产生。当计数器的值加计数到 TACCR0 时产生 TACCR0 的 CCIFG 中断标志；当计数器的值从 0001H 减计数到 0000H 时产生 TAIFG 中断标志，如图 4.8 所示。

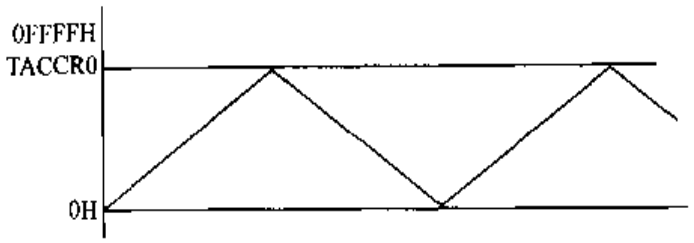


图 4.7　加/减模式

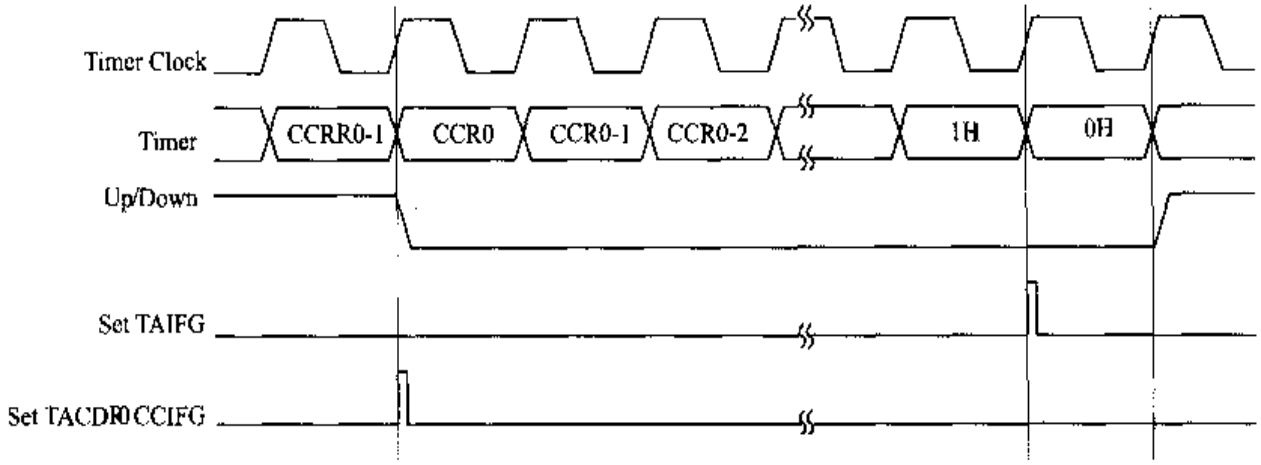


图 4.8　加/减模式的工作时序图

加/减模式同时支持产生带死区的 PWM，这一点将在 4.2.3 小节的比较输出模式中详细介绍。

4.2.2　捕捉和比较单元

1. 捕捉模式

当捕捉/比较控制寄存器的 CAP=1 时，选择捕捉(CAP)模式。

CAP 模式主要用于测量脉冲宽度(比如测量信号频率，通过传感器测量转速等)，记录事件发生时间等。

MSP430F149 的 CAP 输入引脚可以通过 CCISx 位选择，同时可以通过 CMx 位选择捕捉方式。

MSP430 单片机支持同步模式和异步模式。异步模式容易产生竞争冒险，推荐读者使用同步模式。

MSP430 单片机同时支持溢出判断逻辑。当前一次的捕捉值还没有被读取而第二次捕捉又发生时，COV 将被置 1 并且只可以软件清零。用户可以通过查询 COV 的状态判断是否有溢出发生，捕捉过程如图 4.9 所示。下面详细介绍这个过程。

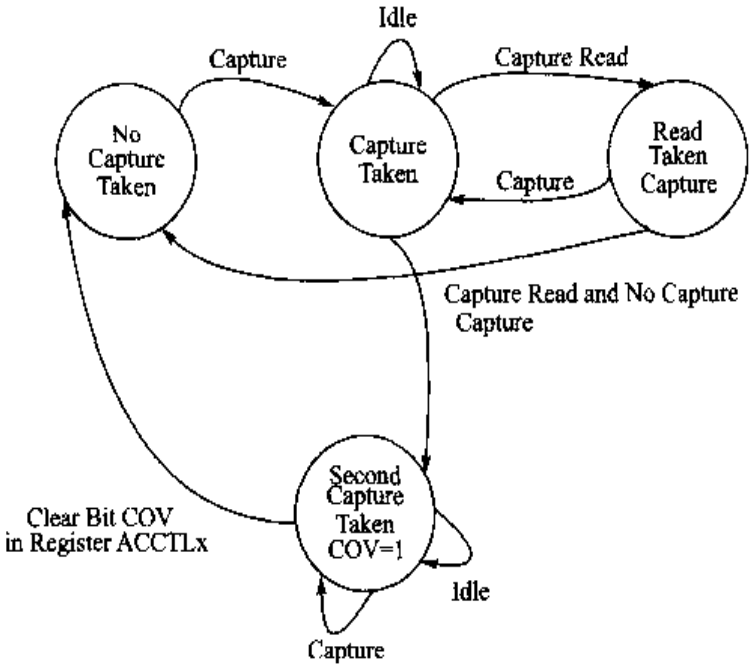


图 4.9 捕捉过程

等待捕捉发生时分三种情况：

① 如果此时没有捕捉发生，将等待；

② 如果此时读取到捕捉值，将到下一个读取值去继续等待；

③ 如果此时正在发生捕捉，将发生溢出。

读者可以清楚地看到，一个正确的捕捉过程包括：等待捕捉事件发生——捕捉发生——判断是否发生溢出并且读取捕捉值(如果发生溢出，则清零 COV 位并且报错，然后重新复位捕捉单元) ——等待下次捕捉。

程序设计举例：测量 CCI0A 引脚脉冲频率并在 LED 上显示。这里只提供捕捉单元的程序，当然也可以独立运行。具体程序读者可以参考 4.5 节中的“单片机的 P12 引脚信号频率测量”。

2. 比较单元

当 CAP=0 时选择比较模式。比较模式主要用来产生 PWM 信号或者用于产生特定时间间隔的中断。

当计数器 TAR 的值等于 TACCRx 的值时：

① 中断标志 CCIFG 置位；

② 内部信号 EQUx=1；

③ EQUx 根据不同输出模式影响输出；

④ 输入信号 CCI 被锁存到 SCCI。

4.2.3 输出模式

每个比较单元中包括一个输出单元，输出单元用来产生输出信号，比如 PWM 信号。每一个输出单元在 EQU0 和 EQUx 的基础上各有 8 种输出模式，输出模式主要由 OUTMODx 位控制，如表 4.1 所列。

除模式 0 外，所有的输出信号在时钟的上升沿改变。输出模式 2、3、6 和 7 不可以在输出单元 1 使用，因为此时 EQUx=EQU0。

表 4.1 输出模式

OUTMODx	MODE	功能描述
000	输出模式	输出信号由 OUT 位控制，可以参考 4.4 节寄存器说明部分
001	置位	当计数器的值到达 TACCRx 时，输出信号置位，并且一直保持到下一次计数器复位或者是其他方式被选择
010	PWM 翻转/复位	当计数器的值等于 TACCRx 的值时，输出信号翻转；当计数器等于 TACCR0 时，输出被复位。由于同时使用 TACCRx 和 TACCR0，所以输出单元 0 不可以使用该模式。3、6、7 方式同样如此
011	置位/复位	当计数器的值等于 TACCRx 的值时，输出信号置位，当计数器等于 TACCR0 时，输出被复位
100	翻转	当计数器的值等于 TACCRx 时输出被翻转，所以输出信号的周期是计数器周期的 2 倍。这一点可以参考不同模式下的时序图
101	复位	当计数器的值到达 TACCRx 时，输出信号置位，并且一直保持到其他方式被选择，影响输出状态
110	PWM 翻转/置位	当计数器的值等于 TACCRx 的值时，输出信号翻转；当计数器的值等于 TACCR0 时，输出被置位
111	PWM 复位/置位	当计数器的值等于 TACCRx 的值时，输出信号复位；当计数器的值等于 TACCR0 时，输出被置位

现在举一个简单例子说明当 Timer_A 工作在增加模式时不同模式下输出信号的改变，如图 4.10 所示。

本章将给出一个具体程序产生一个 PWM 信号供读者参考。

至于其他模式，读者可以查看参考文献[1]。

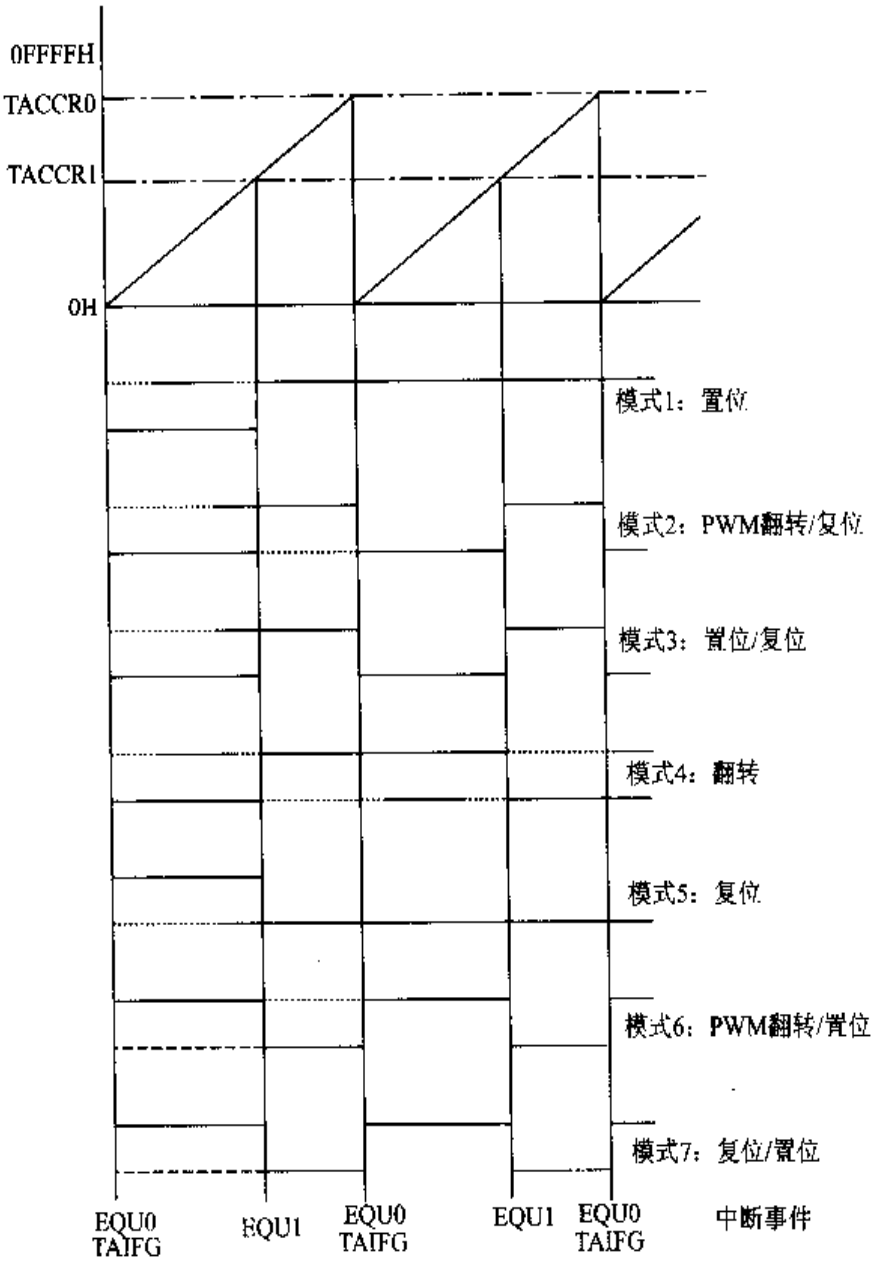


图 4.10 工作在增加模式时不同模式下输出信号的变化

4.2.4 Timer_A 的中断

与 Timer_A 模块相关的中断向量有以下两个：

① TACCR0 中断向量，处理 TACCR0 的 CCIFG 中断标志；

② TAIV 中断向量，处理所有其他的 CCIFG 和 TAIFG 中断标志。

注意：在捕捉模式下，当计数器的值被捕捉到时，相关 TACCRx 寄存器中 CCIFG 将被置位；在比较模式下，当计数器 TAR 等于相关的 TACCRx 的值时，CCIFG 被置位。软件可以对 CCIFG 标志清零或置位。如果

相应的中断被使能，CCIFG 将产生一个相关中断。其他 TAIFG 将共用一个中断向量，我们可以通过 TAIV 寄存器的值来判断具体是哪个中断。

(1) TTACCR0 中断

TACCR0 的中断标志 CCIFG 在 Timer_A 中具有最高优先级，并且享有特定的中断向量，当 CCIFG 中断被响应以后该位自动清零。详细说明请查看参考文献[1]。

(2) TAIV 和中断向量发生器

TACCR1 的 CCIFG、TACCR2 的 CCIFG 和 TAIFG 三个中断的优先级依次降低，且它们共用一个中断源。中断向量寄存器 TAIV 用来确定哪一个产生了中断请求，读取该寄存器的值就可以确定发生了什么中断。

4.3 Timer_B

图 4.11 所示为 Timer_B 的结构框图。从结构框图中可以看出，Timer_B 和 Timer_A 的结构几乎相同，只是 Timer_B 比 Timer_A 增加了比较锁存器，这使得用户可以更加灵活地控制比较数据更新的时机。多个比较锁存器成组工作，以达到同步更新比较数据的目的。这一功能在实际中用途很广，比如可以同步更新 PWM 信号的周期和占空比。

下面简单介绍一下 Timer_B 与 Timer_A 的共同点以及不同之处。

1. 两者的共同点

- 都有 4 种操作模式；
- 都具有可选、可配置的计数器输入时钟源；
- 都有多个独立可配置的捕获/比较模块；
- 都有多个具有 8 种输出模式的可配置输出单元；
- 中断功能都很强大。

2. 两者的不同之处

- Timer_B 的计数长度为 8 位/10 位/12 位/16 位可编程，而 Timer_A 的计数长度固定为 16 位；
- Timer_B 没有 Timer_A 中的 SCCI 寄存器位功能；
- Timer_B 的 TBCCRx 缓存寄存器是双向的；
- Timer_B 的输出可以被设置成高阻状态。

Timer_B 和 Timer_A 操作几乎相同，所以在这里就不详细介绍了。具体说明读者请查看参考文献[1]。

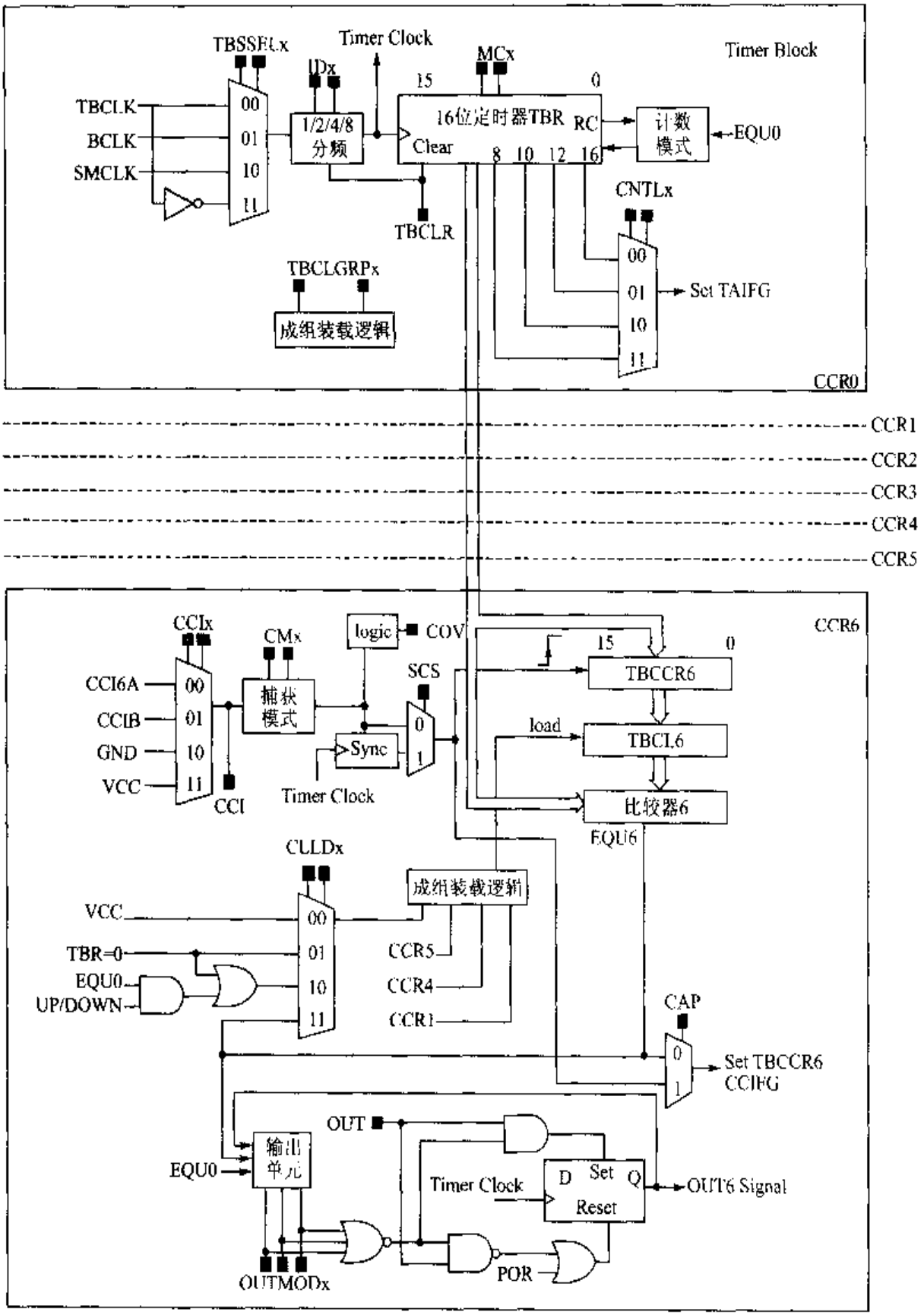


图 4.11　Timer_B 的结构框图

4.4 Timer_A 的寄存器

1. TACTL 寄存器

TACTL 是一个 16 位的寄存器，该寄存器包含了 Timer_A 作为定时器使用的所有控制位。TACTL 寄存器的各个位如图 4.12 所示。

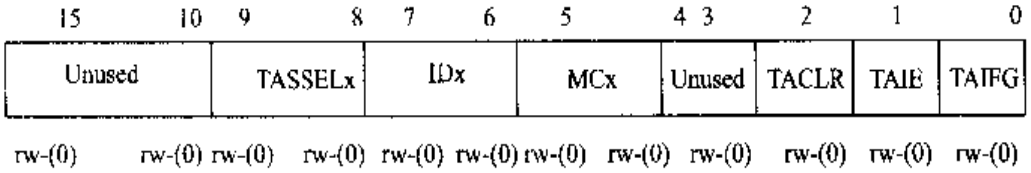


图 4.12 TACTL 寄存器

下面将对 TACTL 寄存器的各个位进行详细介绍。

Unused　Bits15~10

TASSELx　Bits9~8　Timer_A 时钟源选择。

00　TACLK；

01　ACLK；

10　SMCLK；

11　INCLK。

IDx　Bits7~6　选择时钟源的输入分频数。

00　/1；

01　/2；

10　/4；

11　/8。

MCx　Bits5~4　模式选择。

00　停止模式；

01　增加模式，计数器计到 TACCR0，计数周期为 TACCR0+1；

10　连续模式；

11　加减模式，计数器计数到 TACCR0，然后再到 0x0000。

Unused　Bit3　Unused。

TACLR　Bit2　Timer_A 清零。置该位为 1，将清零计数器 TAR、输入时钟分频位、计数方向信号。TACLR 自动清

零并且一直读为 0。

TAIE	Bit1	Timer_A 中断使能。 0　中断屏蔽； 1　中断使能。
TAIFG	Bit0	Timer_A 中断标志。 0　没有中断挂起； 1　有中断挂起。

2. TAR 寄存器

TAR 是一个 16 位的寄存器，该寄存器的内容表示了当前的计数值，内部可读/写。TAR 寄存器的各个位如图 4.3 所示。

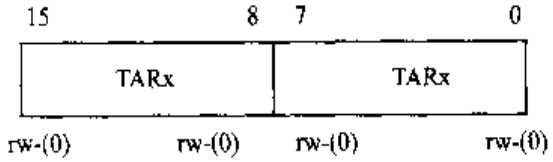


图 4.13　TAR 寄存器

3. TACCTLx 寄存器

TACCTLx 是一个 16 位的寄存器。Timer_A 有多个捕捉/比较模块，每个模块都有自己的控制寄存器 CCTLx。TACCTLx 寄存器的各个位如图 4.14 所示。

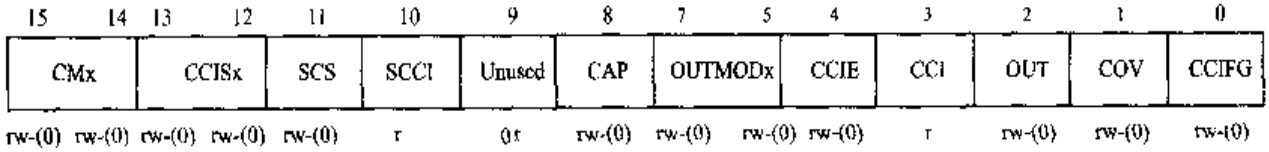


图 4.14　TACCTLx 寄存器

下面将对 TACCTLx 寄存器的各个位进行详细介绍。

CMx	Bits15~14	捕捉触发信号选择。 00　不触发捕捉； 01　上升沿触发捕捉； 10　下降沿触发捕捉； 11　上升沿和下降沿都触发捕捉。
CCISx	Bits13~12	捕捉/比较输入选择。通过该位选择输入信号，同时需要使 I/O 工作在外部模块方式，即相应的引脚 PxSEL 的位应该等于 1。 00　CCIxA；

		01　CCIxB； 10　GND； 11　VCC。
SCS	Bit11	捕捉同步或异步选择。 0　异步模式； 1　同步模式。
SCCI	Bit10	捕捉/比较同步输入。
Unused	Bit9	未使用，只读，通常读为 1。
CAP	Bit8	捕捉模式选择。 0　比较模式； 1　捕捉模式。
OUTMODx	Bits7~5	输出模式选择。 000　等于 OUT 位的值； 001　置位； 010　触发/复位； 011　置位/复位； 100　触发； 101　复位； 110　触发/置位； 111　复位/置位。
CCIE	Bit4	捕捉/比较中断使能，使能 CCIFG 标志产生的中断。 0　中断禁止； 1　中断使能。
CCI	Bit3	捕捉比较输入，被选择的输入可以通过该位读取。
OUT	Bit2	当选择模式 0 时，该位直接控制输出状态。 0　输出为低； 1　输出为高。
COV	Bit1	捕捉溢出标志位。当第一次的捕捉数还没有被读取的时候，又发生一个捕捉事件，该位将置 1。该位只可以软件清零。建议当发生溢出时，软件清零以后再使用捕捉单元。 0　没有发生溢出； 1　发生溢出。
CCIFG	Bit0	捕捉/比较中断标志。

0　　没有中断挂起；

1　　有中断被挂起。

4. TAIV 寄存器

TAIV 是一个 16 位的寄存器，该寄存器是 Timer_A 的中断向量寄存器。TAIV 寄存器的各个位如图 4.15 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3 ~ 1	0
0	0	0	0	0	0	0	0	0	0	0	0	TAIVx	0
r0	r0	r0	r0	r0	r0	r0	r0	r0	r0	r0	r0	r-(0)	r-(0) r0

图 4.15　TAIV 寄存器

Timer_A 的中断优先级如表 4.2 所列。

可以通过读取该寄存器的值判断是哪一个中断。

表 4.2　Timer_A 的中断优先级

TAIV 内容	中断源	中断标志	中断优先级
00H	没有中断标志	-	
02H	捕获/比较 1	CCR1 的 CCIFG	最高
04H	捕获/比较 2	CCR2 的 CCIFG	
06H	保留将来使用	-	
08H	保留将来使用	-	
0AH	定时器溢出	TAIFG	
0CH	保留将来使用	-	
0EH	保留将来使用		最低

4.5　程序设计举例

1. 通过 Timer_A 来产生脉冲信号

该实例主要通过 Timer_A 来产生一个脉冲信号，信号频率为 1 Hz，占空比为 1∶1，采用中断方式工作，定时器每 10 ms 中断一次，主函数用变量 time 来作为软件计数，满 50 次后也就是 500 ms 后，使 P1.2 取反，得到系统设计的方波信号。该实例的软件流程如图 4.16 所示。通过该例的学习，可以掌握定时器 A 的基本使用和能够学会正确地使用定时器。

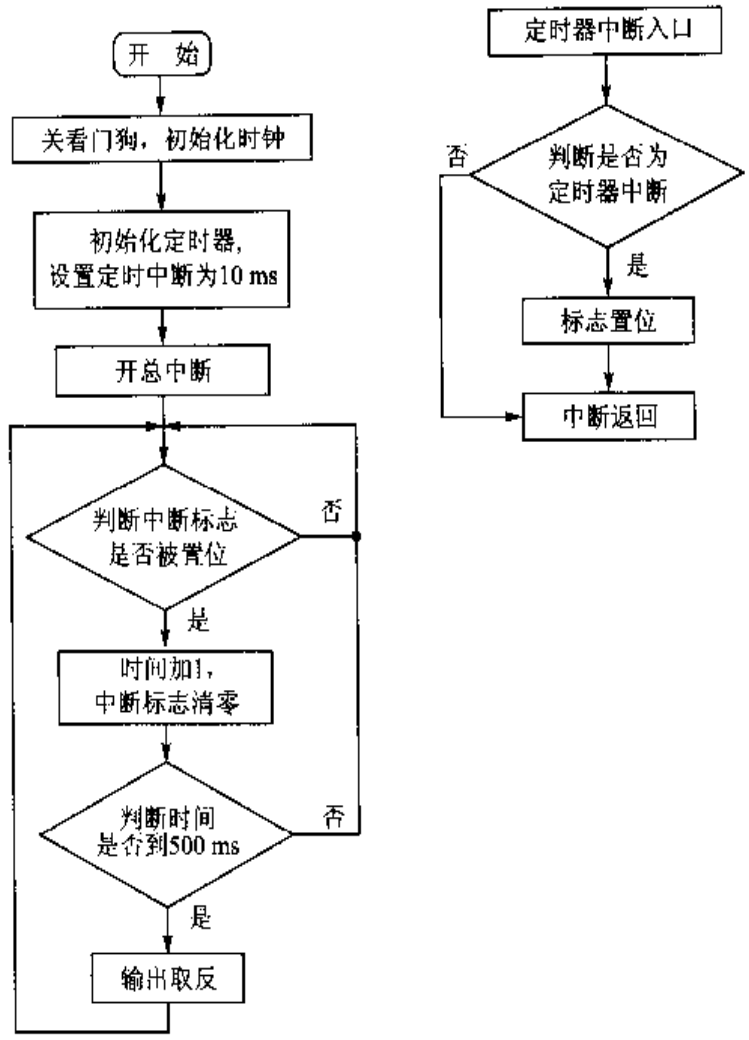


图 4.16　Timer_A 产生脉冲信号的软件流程图

具体程序如下：

```
#include <msp430x14x.h>
#define   uint  unsigned int
#define   uchar unsigned char
#define   OUT   BIT2
static uchar flag = 0;

void int_clk()
{
    uchar i;
    BCSCTL1&= ~XT2OFF;                    //打开 XT 振荡器
    BCSCTL2| = SELM1 + SELS;              //MCLK 为 8 MHz,SMCLK 为 1 MHz
    do
    {
        IFG1 &= ~OFIFG;                   //清除振荡错误标志
```

```
        for(i = 0; i < 100; i++)
        _NOP();                                    //延时等待
        ;
    while ((IFG1 & OFIFG) != 0);                   //如果标志为1,则继续循环等待
    IFG1& = ~OFIFG;
}
void int_timer()
{
    TACTL| = TASSEL1 + TACLR + ID0 + ID1;          //选择 SMCLK 作为定时器的时钟,8 分频
    TACTL| = MC0 + TAIE;                           //增加模式,并且使能中断
TACCR0 = 9999;                                     //中断周期为 10 ms
}
#pragma vector = TIMERA1_VECTOR
__interrupt void Timer_A(void)
{
    switch(TAIV)
{
    case 2:break;
    case 4:break;
    case 10:flag = 1;break;                        //10 ms 中断标志加 1
}
}
int main( void )
{
    uchar time = 0;
    WDTCTL = WDTPW + WDTHOLD;                      //关看门狗
    int_clk();                                     //初始化时钟
    int_timer();                                   //初始化定时器
    P1DIR| = OUT;                                  //设置 P12 为输出脚
    _EINT();                                       //开中断
    while(1)
    {
        while(flag)                                //查询中断标志
        {
            flag = 0;                              //清中断标志
            time++;                                //时间变量加 1
            if(time == 50)                         //等于 500 ms 取反
            {
                P1OUT = (~P1OUT&OUT);              //改变输出状态
```

```
                time = 0
            }
        }
    }
}
```

2. 单片机的 P12 引脚输入信号频率测量

该例使用 MSP430F149 的 P12 引脚的捕捉功能来测量信号的频率，利用两次捕捉的时间差，得到外部信号的周期即可知信号的频率，并且通过 LED 显示出来，软件流程如图 4.17 所示。由于频率低时捕捉会发生溢出，希望读者自己处理，实际应用中可以采取分段测试。作者在实际测试时，到超过 140 Hz 时就会发生溢出。

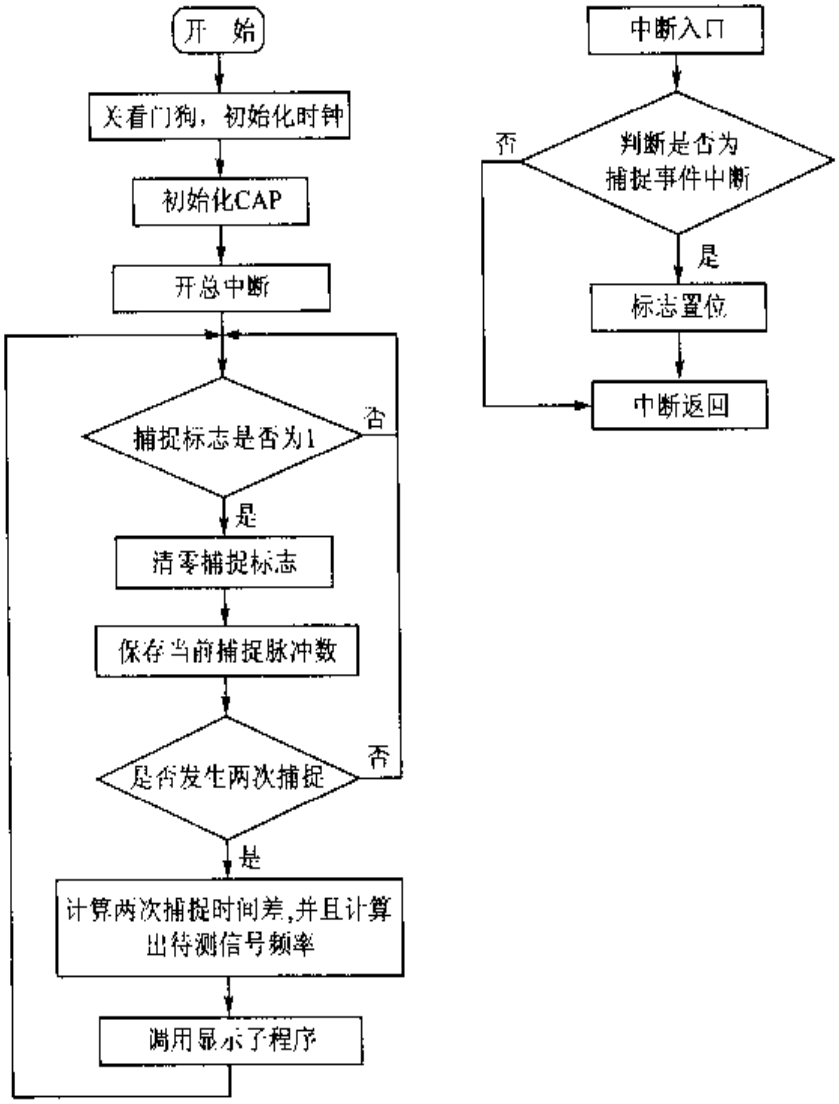


图 4.17 测量频率的软件流程图

具体程序如下：

```
#include <msp430x14x.h>
#define   uint  unsigned int
#define   uchar unsigned char
uint str[100] = 0 ;
uchar flag = 0;
#define led   BIT7
const uchar  segment[10] = {0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
float count = 0;
uint temp2 = 0,temp1 = 0;
uchar a1 = 0,a2 = 0,a3 = 0,a4 = 0;
void int_clk()
{
    uchar i;
    BCSCTL1&= ~XT2OFF;                    //打开 XT 振荡器
    BCSCTL2 = SELM1 + SELS;               //MCLK 为 8 MHz,SMCLK 为 1 MHz
    do
    {
        IFG1 &= ~OFIFG;                   //清除振荡错误标志
        for(i = 0; i < 100; i++)
        _NOP();                           //延时等待
        }
    while ((IFG1 & OFIFG) != 0);          //如果标志为 1,则继续循环等待
    IFG1&= ~OFIFG;
}
void int_spi()
{
    U0CTL| = SWRST;
    U0CTL| = CHAR + SYNC + MM;
    U0TCTL| = SSEL1 + SSEL0 + STC;        //三线模式 SMCLK 作为 SPI 时钟
    U0TCTL| = CKPH;
    U0TCTL&= ~CKPL;                       //CKPL CKPH:01 上升沿写数据
    U0BR0 = 0X02;
    U0BR1 = 0X00;
    U0MCTL = 0X00;
    ME1| = USPIE0;                        //使能 SPI
    U0CTL&= ~SWRST;
    IE1&= ~UTXIE0;
    IE1&= ~URXIE0;                        //禁止中断
    P3SEL = 0x0E;
```

```
    P3DIR| = BIT2 + BIT4;                    //选择第二功能和 I/O 方向
}
write_byte(uchar byte)
{
    U0TXBUF = byte;
    while((IFG1&UTXIFG0) = = 0);             //UTXIFG0 = 0,表示数据从发送
    IFG1& = ~UTXIFG0;
}
display(uchar data)
{
    P2DIR| = led;
    P2OUT| = led;                            //使能 164CLK
    write_byte(segment[data]);               //写数据
    P2OUT& = ~led;                           //禁止 164CLK
}
delay()
{
    uint i,j;
    for(i = 0;i <= 1000;i + + )
    {
    for(j = 0;j <= 1000;j + + )
            ;
    }
}
void int_cap()
{
    P1SEL = 0X04;                            //选择 P12 作为捕捉的输入端子
    TACCTL1 = CM0 + SCS + CAP + CCIE;        //上升沿触发捕捉,同步模式,使能中断
    TACTL| = TASSEL1 + MC1;                  //选择 SMCLK 时钟作为计数脉冲,不分频
}
#pragma vector = TIMERA1_VECTOR
__interrupt void Timer_A(void)               //中断服务程序
{
    switch(TAIV)
    {
        case 2:{flag = 1;}                   //置捕捉标志为 1
        case 4:break;
        case 10:break;
    }
```

```
}
void main()
{
    uchar temp = 0;
    WDTCTL = WDTPW + WDTHOLD;                    //关看门狗
    int_clk();                                   //初始化时钟
    int_cap();                                   //初始化 CAP
    int_spi();
    _EINT();                                     //开总中断
    while(1)
    {
        while(flag)                              //查询捕捉标志
        {
            _DINT();
            flag = 0;                            //清零捕捉标志
            str[temp] = TACCR1;                  //读时间 str
            temp++;
            if(temp == 2)
            {
                count = str[1] - str[0]; //保存 2 次捕捉信号的时间差,计算出信号频率
                count = 8000/count;
                count = count * 1000;
                temp2 = count;
                temp1 = temp2/1000;
                a1 = temp1;
                temp2 = temp2 - temp1 * 1000;
                temp1 = temp2/100;
                temp2 = temp2 - temp1 * 100;
                a2 = temp1;
                temp1 = temp2/10;
                temp2 = temp2 - temp1 * 10;
                a3 = temp1;
                a4 = temp2;
                display(a4);
                display(a3);
                display(a2);
                display(a1);
                str[0] = 0;
                str[1] = 0;
```

```
                temp = 0;
                delay();
                delay();
                }
                _EINT();
            }
        }
    }
```

读者可以自行修改该程序，提高频率测量的精度和测量范围。

3. P12 口输出一 PWM 信号

该例子主要通过 P12 口输出一 PWM 信号，频率为 10 kHz，占空比为 1∶1，通过修改 TACCR0 和 TACCR1 可以改变 PWM 波形的频率和占空比。读者通过该例能正确使用 MSP430F149 的 PWM 功能，掌握利用定时器产生 PWM 方法。软件流程图如图 4.18 所示。

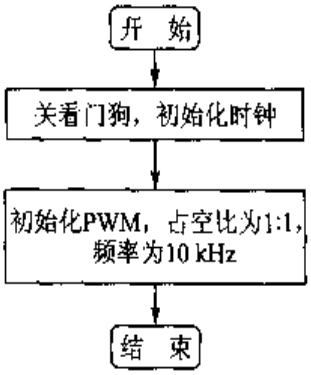


图 4.18 产生 PWM 信号的软件流程图

具体程序如下：

```
#include <msp430x14x.h>
#define   uint  unsigned int
#define   uchar unsigned char
#define   PWM   BIT2
void int_clk()
{
    uchar i;
    BCSCTL1& = ~XT2OFF;                    //打开 XT 振荡器
    BCSCTL2| = SELM1 + SELS;               //MCLK 为 8 MHz,SMCLK 为 1 MHz
    do
    {
        IFG1 &= ~OFIFG;                    //清除振荡错误标志
        for(i = 0; i < 100; i++)
        _NOP();                            //延时等待
        }
    while ((IFG1 & OFIFG) != 0);           //如果标志为 1,则继续循环等待
    IFG1& = ~OFIFG;
}
void int_pwm()
{
    P1SEL = PWM;                           //选择 P12 作为 PWM 输出
```

```
    P1DIR  = PWM;
    TACCR0 = 800;                          //PWM 信号频率 10 KHz
    TACCR1 = 400;                          //占空比 1∶1
    TACCTL1 = OUTMOD0 + OUTMOD1 + OUTMOD2;  //输出模式选择
TACTL  = TASSEL1 + MC0;
}
void main()
{
    WDTCTL = WDTPW + WDTHOLD;              //关看门狗
    int_clk();                             //初始化时钟
int_pwm();                                 //初始化 PWM
while(1);                                  //结束
}
```

通过以上3个程序的学习，读者应对定时器的操作有了大致的了解。MSP430单片机的定时器功能比我们熟悉的其他单片机功能要强得多，当然设置起来也会比较复杂，但只要设置好相应的寄存器就可以很好地使用。读者能够在这些程序的基础上加以修改，对MSP430单片机的定时/计数器有一个初步的认识，并且通过设计实例的分析，将它应用到工程实践中去。

第5章

通用同步/异步收发器 USART

通用同步/异步收发器采用一个硬件，支持两种通用串行总线模式：UART 接口（异步模式）和 SPI 接口（同步模式）。我们可以根据寄存器 UxCTL 的 SYNC 位来选择 USART 的工作模式。在 MSP430F149 中有两个相同的串行总线接口，可以分别或同时配置成两种模式，下面分别介绍其基本操作和应用。

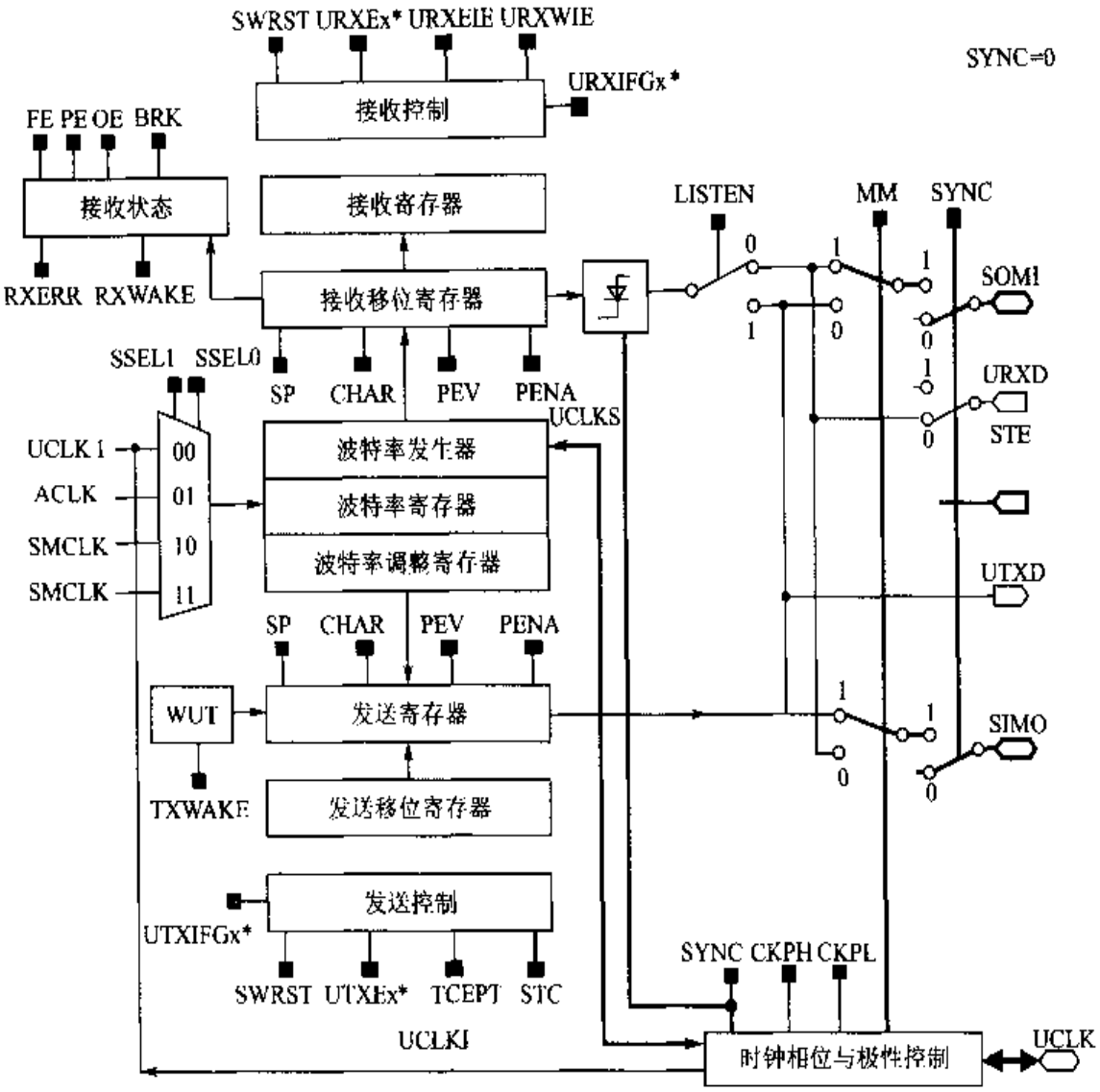
5.1 通用异步串行接口 UART

USART 作为通用异步收发器时的接口框图如图 5.1 所示。

SYNC 位为 0 时，USART 工作在异步模式下。在异步模式下，USART 通过 URXD、UTXD 这两个引脚与外部系统连接。

异步模式的主要特点是：

- 7 位或 8 位数据并且支持奇、偶、无校验位；
- 独立的接收和发送移位寄存器；
- 独立的接收和发送寄存器；
- 接收和发送时，低位在前；
- 空闲模式和地址模式支持多机通信；
- 接收开始边沿检测使 CPU 在低功耗模式下进入工作状态（LPMx）；
- 错误检测标志和挂起、地址检测功能；
- 独立的接收和发送中断功能。



注：x 表示 x 为 1 或 0。

图 5.1　USART 在异步模式下的硬件结构图

5.1.1　串口操作的基本步骤

串口操作的基本步骤如下：

① 置位 SWRST 来复位串口(串口复位主要通过两个操作实现：上电复位或置 SWRST 位为 1)；

② 初始化所有的 USART 寄存器；

③ 使能 USART 模块；

④ 清零 SWRST 位；

⑤ 如果需要，则使能中断。

注意：如果不按照这样的过程进行操作，可能会得到不确定的结果。

5.1.2 通用异步串口的数据格式

通用异步串口数据格式包括起始位、7 位或 8 位数据(软件可选)、奇偶校验位(软件可选)、地址位(MM=1 时有效)、1 位或 2 位停止位(软件可编程)。每位数据的周期通过所选择的时钟和波特率发生器来确定,一帧数据格式如图 5.2 所示。

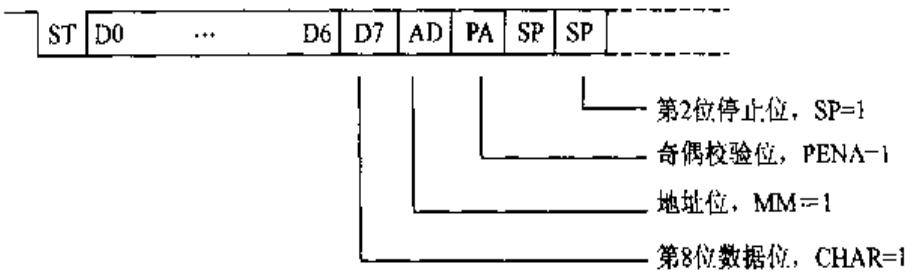


图 5.2 通用异步收发器的数据格式

5.1.3 异步通信模式

当两个器件异步通信时,我们使用空闲模式。当多个器件异步通信时,通用异步串口支持空闲模式或地址模式。下面详细介绍用于多机通信的两种模式。

1. 空闲模式

当 MM=0 时,空闲多机模式被选择。在空闲多机模式中,数据块被空闲时间段分成不同数据块。一个标准的空闲时间段由 10 位或更多位连续的"1"组成。如果停止位采用 2 位,那么最后一个停止位将被看成是空闲位的开始位。空闲位后面紧跟的就是地址位信息。用户在空闲位以后可以判断地址信号是不是自己的地址,如果是则继续接收数据,如果不是则等待接收下一个地址。

这里要重点说明一下地址信息和数据信息的区分以及相关的控制位。通过寄存器的 URXWIE 位可以控制数据接收是否允许。当 URXWIE=1 时,只接收地址信息,数据将被忽略。接收到地址信息后,判断是否和自己的地址相符合。如果符合,则清零 URXWIE 位准备接收数据;如果不符合,则保持 URXWIE=1 等待接收下一个地址。这样就很容易实现多机通信任务。

注意:硬件不会自动清零 URXWIE 位,当用户准备接收数据的时候需要用户软件清零该位。

在发送状态下,MSP430F149 产生空闲模式的步骤主要分为两个步骤,详细介绍请查看参考文献[1]。

(1) 置 TXWAKE 位为 1,当 UxTXBUF 准备好发送数据(UTXIFGx=1)时,写任意数据

到接收缓冲寄存器 UxTXBUF 里。

(2) 当 UxTXBUF 准备好发送数据(UTXIFGx=1)时，写需要的地址到 UxTXBUF，然后发送数据到相应的从机。

2. 地址模式

当 MM=1 时，地址模式被选择。在传送的数据中包含一个额外的位，用来表明此时发送的是地址还是数据。

URXWIE 位用来控制数据接收是否允许。当 URXWIE=1 时，接收到的数据信息将被忽略，并且不会产生相应的中断。当接收到一个地址信息时，我们可以判断是否和自己的地址符合，如果符合，则清零 URXWIE 等待接收数据信息；如果不符合，则保持 URXWIE=1 等待接收下一个地址。

注意：和空闲模式一样，硬件不会自动清零 URXWIE 位，当用户准备接收数据的时候需要用户软件清零。

在发送状态下，地址位受 TXWAKE 位控制。当 TXWAKE=1 时，发送地址信息；当 TXWAKE=0 时，发送数据信息。当数据写到 WUT 中或者 SWRST 位被置位时，TXWAKE 将自动清零。

5.1.4　UART 的波特率

USART 作异步使用时的时钟发生框图如图 5.3 所示。

由 UxBR1 和 UxBR0 组成 16 位计数器。在主机模式下，最大波特率为 BRCLK/2。在从机模式下，最大波特率为 BRCLK。因为分频因子一般不是整数，所以用 16 位计数器来调整整数部分，而小数部分则通过 UxMTCL 寄存器来微调。

分频因子 $N=UBR+(M7+M6+M5+M4+M3+M2+M1+M0)/8$，其中 UBR 为 16 位数据值，而 Mx 为调整寄存器 UxMTCL 中的各数据位。

波特率 $=BRCLK/N=BRCLK/[UBR+(M7+M6+M5+M4+M3+M2+M1+M0)/8]$。

5.1.5　异步模式下的寄存器

1. UxCTL 寄存器

UxCTL 寄存器是一个 8 位的寄存器，用来控制 USART 的基本操作。UxCTL 寄存器的各个位如图 5.4 所示。

下面将对 UxCTL 寄存器的各个位进行详细介绍。

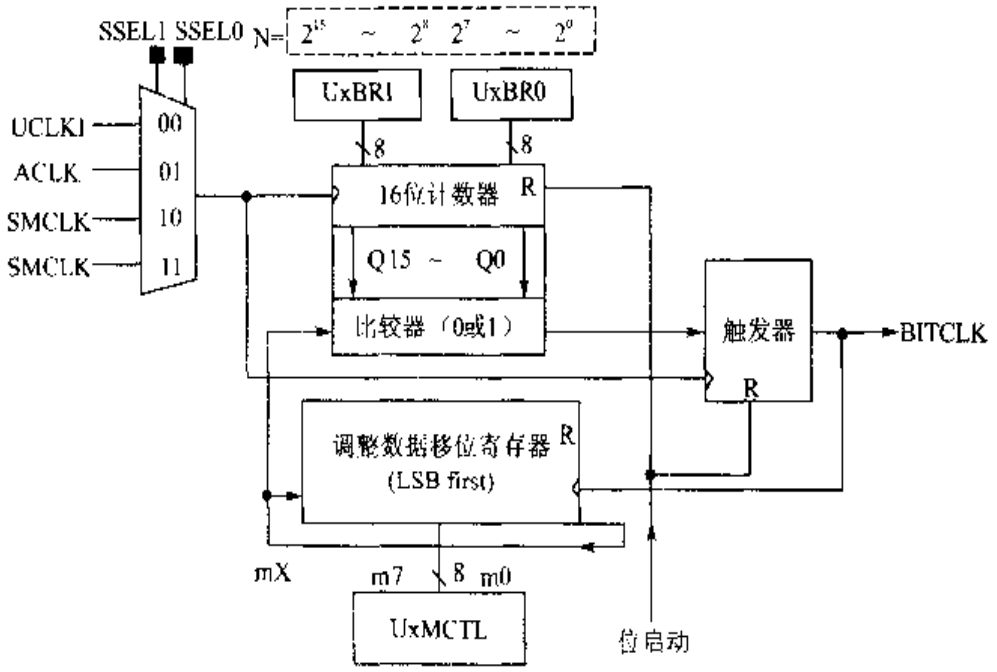


图 5.3 时钟产生框图

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

图 5.4 UxTCL 寄存器

PENA Bit7 奇偶使能位。

0 不使能奇偶位；

1 奇偶使能位，奇偶产生于(UTXDx)并且在(URXDx)接收。在地址－多机模式下，地址位包括在奇偶校验计算里。

PEV Bit6 奇偶校验选择位。该位在 PENA 使能时有效。

0 偶校验；

1 奇校验。

SPB Bit5 停止选择位，发送停止位位数。

0 1 位停止位；

1 2 位停止位。

CHAR Bit4 数据长度选择。

0 7 位数据；

1 8 位数据。

LISTEN Bit3 循环模式选择位。

0　禁止循环模式；

1　使能循环模式。

SYNC　Bit2　模式选择。

0　UART 模式；

1　SPI 模式。

MM　Bit1　多 CPU 模式选择。

0　空闲多机模式；

1　地址选择多机模式。

SWRST　Bit0　软件复位。

0　正常工作方式；

1　复位状态。

2. UxTCTL 寄存器

UxTCTL 寄存器是一个 8 位的寄存器，它用来控制 USART 的接收操作。UxTCTL 寄存器的各个位如图 5.5 所示。

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

图 5.5　UxTCTL 寄存器

下面将对 UxTCTL 寄存器的各个位进行详细介绍。

Unused　Bit7　未用。

CKPL　Bit6　时钟极性选择。

0　UCLK；

1　和 UCLK 相反。

SSELx　Bits5～4　串口工作时钟选择位。

00　UCLKI；

01　ACLK；

10　SMCLK；

11　SMCLK。

URXSE　Bit3　异步串口接收开始边沿选择。

0　禁止；

1　使能。

TXWAKE　Bit2　发送地址数据选择。

0　下一个要发送的是数据；

1　下一个要发送的是地址。

Unused　Bit1　未用

TXEPT　Bit0　发送空标志寄存器。

0　正在发送数据；

1　空闲。

3. UxRCTL 寄存器

UxRCTL 寄存器是一个 8 位的寄存器，它用来控制 USART 的接收操作。UxRCTL 寄存器的各个位如图 5.6 所示。

7	6	5	4	3	2	1	0
FE	PE	OE	BRK	URXEIE	UTXWIE	RXWAKE	RXERR
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

图 5.6　UxRCTL 寄存器

下面将对 UxRCTL 寄存器的各个位进行详细介绍。

FE　Bit7　帧出错标志位。

0　没有帧错误；

1　帧错误。

PE　Bit6　校验出错标志位。

0　校验正确；

1　校验错误。

OE　Bit5　溢出标志位。

0　无溢出；

1　有溢出。

BRK　Bit4　打断检测位。

0　没有被打断；

1　被打断。

URXEIE　Bit3　接收出错中断允许位。

0　不允许中断，不接收出错字符并且不改变 URXIFG 标志；

1　允许中断，出错字符接收并且能够置位 URXIFG。

URXWIE　Bit2　接收唤醒中断允许位。

0　所有接收到的字符都能够置位 URXIFG；

1　只能接收到地址字符才能够置位 URXIFG。

RXWAKE　Bit1　　接收唤醒检测位。

0　没有被唤醒，接收的字符是数据；

1　唤醒，接收的字符是地址。

RXERR　Bit0　　接收出错标志位

0　没有接收错误；

1　有接收到错误。

4. UxBR0、UxBR1 寄存器

UxBR0 和 UxBR1 寄存器都是 8 位的寄存器，用来确定波特率的整数部分。其中，UxBR0为低字节，UxBR1 为高字节。UxBR0 和 UxBR1 两个字节合起来为一个 16 位的字，成为 UxBR。有效波特率的范围是：3≤UxBR≤0FFFH。UxBR0 和 UxBR1 寄存器的各个位分别如图 5.7 和图 5.8 所示。

7	6	5	4	3	2	1	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

图 5.7　UxBR0 寄存器

7	6	5	4	3	2	1	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

图 5.8　UxBR1 寄存器

5. UxMCTL 寄存器

UxMCTL 寄存器是一个 8 位的寄存器，用来确定波特率的小数部分。UxMCTL 寄存器的各个位如图 5.9 所示。

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

图 5.9　UxMCTL 寄存器

6. UxRXBUF 寄存器

UxRXBUF 寄存器是一个 8 位的寄存器，用来接收数据。当有数据时，从该寄存器里读出数据。UxRXBUF 寄存器的各个位如图 5.10 所示。

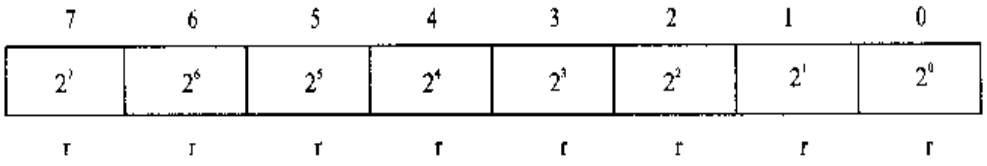


图 5.10 UxRXBUF 寄存器

7. UxTXBUF 寄存器

UxTXBUF 寄存器是一个 8 位的寄存器，用来发送数据。当有数据需要发送时，将数据写入到该寄存器中。UxTXBUF 寄存器的各个位如图 5.11 所示。

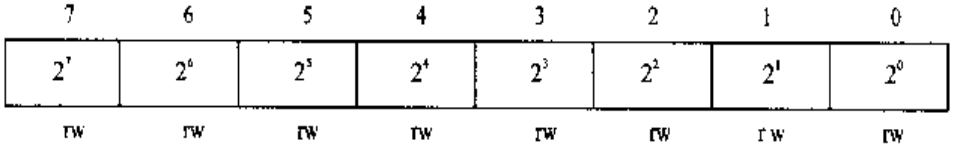


图 5.11 UxTXBUF 寄存器

8. ME1 寄存器

ME1 寄存器是一个 8 位的寄存器，用来使能 MSP430F1XX 序列中 USART0 的 TXD 和 RXD 位；而对于 MSP430X12XX 序列，则是通过 ME2 寄存器来设置的。ME1 寄存器的各个位如图 5.12 所示。

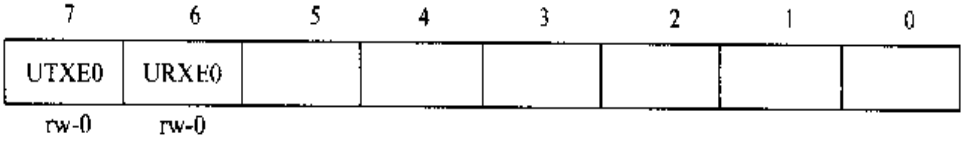


图 5.12 ME1 寄存器

下面将对 ME1 寄存器的各个位进行详细介绍。

UTXE0　Bit7　USART0 发送允许位。

0　不允许；

1　允许。

URXE0　Bit6　USART0 接收允许位。

0　不允许；

1　允许。

9. ME2 寄存器

ME2 寄存器是一个 8 位的寄存器，它用来使能 USART1 的 TXD 和 RXD 位，以及 MSP430X12XX 序列中 USART0 的 TXD 和 RXD 位。ME2 寄存器的各个位如图 5.13 所示。

下面对 ME2 寄存器的各个位进行详细介绍。

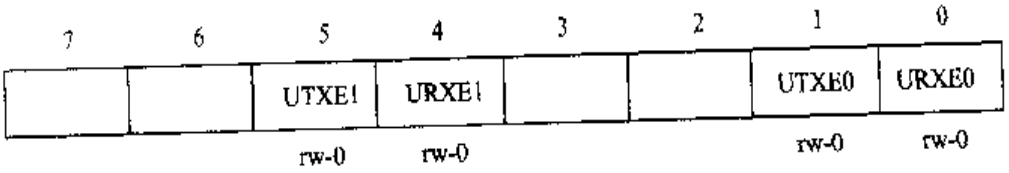


图 5.13　ME2 寄存器

UTXE1　Bit5　USART1 发送允许位。
0　不允许；
1　允许。

URXE1　Bit4　USART1 接收允许位。
0　不允许；
1　允许。

UTXE0　Bit1　USART0 发送允许位。
0　不允许；
1　允许。

URXE0　Bit0　USART0 接收允许位。
0　不允许；
1　允许。

10. IE1 寄存器

IE1 寄存器是一个 8 位的寄存器，用来对 MSP430F1XX 序列中 USART0 的中断使能进行设置；而对于 MSP430X12XX 序列，则是通过 IE2 寄存器来设置。IE1 寄存器的各个位如图 5.14所示。

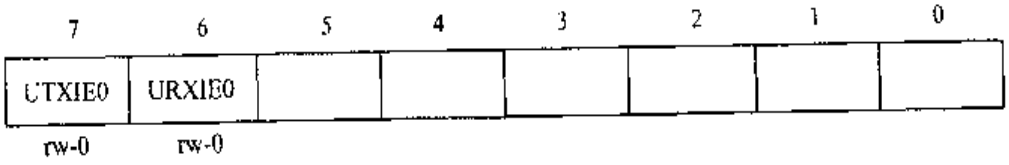


图 5.14　IE1 寄存器

下面对 IE1 寄存器的各个位进行详细介绍。

UTXIE0　Bit 7　USART0 发送中断允许位。
0　不允许；
1　允许。

URXIE0　Bit 6　USART0 接收中断允许位。
0　不允许；
1　允许。

11. IE2 寄存器

IE2 寄存器是一个 8 位的寄存器，用来设置 USART1 的中断使能，以及 MSP430X12XX 序列中 USART0 的中断使能。IE2 寄存器的各个位如图 5.15 所示。

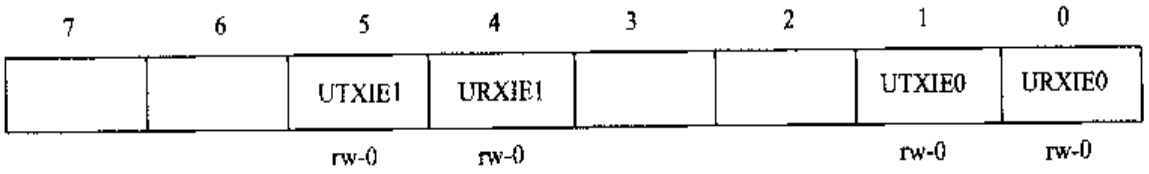


图 5.15 IE2 寄存器

下面将对 IE2 寄存器的各个位进行详细介绍。

UTXIE1 Bit5 USART1 发送中断允许位。

0 不允许；

1 允许。

URXIE1 Bit4 USART1 接收中断允许位。

0 不允许；

1 允许。

UTXIE0 Bit1 USART0 发送中断允许位。

0 不允许；

1 允许。

URXIE0 Bit0 USART0 接收中断允许位。

0 不允许；

1 允许。

12. IFG1 寄存器

IFG1 寄存器是一个 8 位的寄存器，用来设置 MSP430F1XX 序列中 USART0 的中断标志位；而对于 MSP430X12XX 序列，则是通过 IFG2 寄存器来设置的。IFG1 寄存器的各个位如图 5.16 所示。

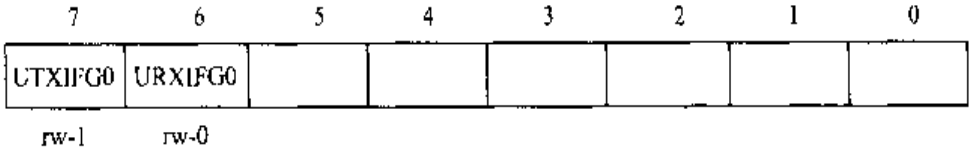


图 5.16 IFG1 寄存器

下面将对 IFG1 寄存器的各个位进行详细介绍。

UTXIFG0 Bit7 USART0 发送中断标志位。

0 没有中断；

1　中断挂起。

URXIFG0　Bit6　USART0 接收中断标志位。

0　没有中断；

1　中断挂起。

13. IFG2 寄存器

IFG2 寄存器是一个 8 位的寄存器，用来设置 USART1 的中断标志位，以及 MSP430X12XX 序列中 USART0 的中断标志位。IFG2 寄存器的各个位如图 5.17 所示。

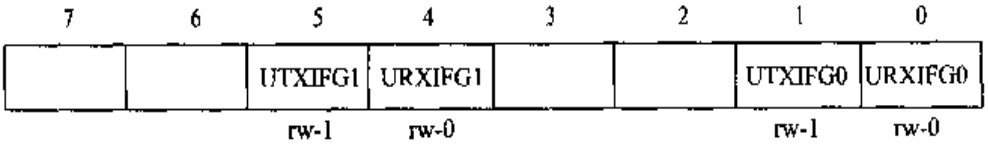


图 5.17　IFG2 寄存器

下面对 IFG2 寄存器的各个位进行详细介绍。

UTXIFG1　Bit5　USART1 发送中断标志位。

0　没有中断；

1　中断挂起。

URXFG1　Bit4　USART1 发送中断标志位。

0　没有中断；

1　中断挂起。

UTXIFG0　Bit1　USART0 发送中断标志位。

0　没有中断；

1　中断挂起。

URXIFG0　Bit0　USART0 接收中断标志位。

0　没有中断；

1　中断挂起。

5.1.6　UART 的设计举例

该实例是 MSP30F149 与计算机实现 RS232 通信，其中数据接收采用中断工作方式，数据发送采用查询工作方式。当单片机接收到 PC 机发来的数据时，单片机把数据返回给 PC 机。通过该实例让读者更进一步了解 RS232 通信的相关知识，以及 MSP430F149 单片机串行通信的基本设置过程。

1. 系统电路

由于 MSP430 的 I/O 电平是 3.3 V，因此电路中采用的电平转换芯片是 MAX3232。系统电路如图 5.18 所示。

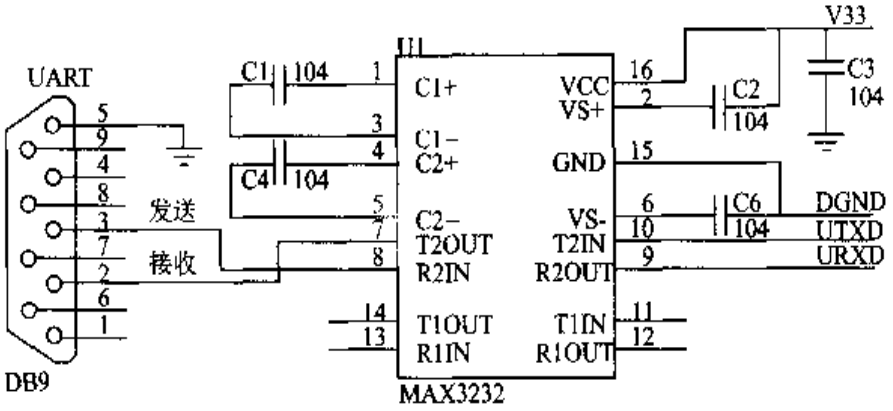


图 5.18 系统电路

2. 软件设计流程图

MSP30F149 与计算机实现 RS232 通信的主程序流程和中断子程序软件流程分别如图 5.19和图 5.20 所示。

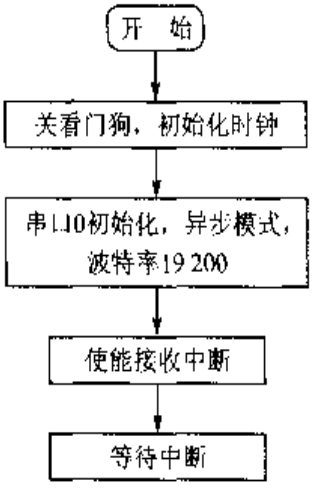


图 5.19 主程序流程图

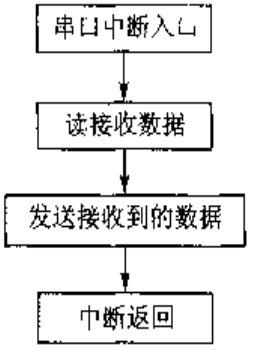


图 5.20 中断子程序流程图

3. 具体程序

```
#include <msp430x14x.h>
#define   uint  unsigned int
#define   uchar unsigned char
void int_clk()
```

```
{
    uchar i;
    BCSCTL1& = ~XT2OFF;                    //打开 XT 振荡器
BCSCTL2| = SELM1 + SELS;                   //MCLK 为 8 MHz,SMCLK 为 1 MHz
do
{
        IFG1 &= ~OFIFG;                    //清除振荡错误标志
    for(i = 0; i < 100; i++)
        _NOP();                            //延时等待
        }
    while ((IFG1 & OFIFG) != 0);           //如果标志为 1,则继续循环等待
    IFG1& = ~OFIFG;
}
int_usart()
{
    U0CTL  = SWRST;                        //复位串口
    U0CTL  = CHAR;                         //8 位数据
    U0TCTL| = SSEL1;                       //select SMCLK AS CLK
    U0BR0 = 0Xa0;
    U0BR1 = 0X01;
    UMCTL0 = 0Xc0;                         //波特率 19 200
    ME1 | = UTXE0 + URXE0;                 //使能接收和发送
    U0CTL& = ~SWRST;                       //清除串口复位信号
    IE1 = URXIE0;                          //使能接收中断
    P3SEL  = BIT4;
    P3SEL  = BIT5;                         //选择 I/O 口使用扩展功能和方向
    P3DIR  = BIT4;
}
sent_byte(uchar data)
{
    while((IFG1&UTX1FG0) == 0);            //判断发送缓冲区是否结束
    U0TXBUF = data;
}
#pragma vector = UART0RX_VECTOR
__interrupt  void UART0_RX_ISR(void)
{
    uchar data = 0;
    data = U0RXBUF;                        //读取接收到的数据并且发送到 PC 机
    sent_byte( data);
```

```
}
int main( void )
{
    WDTCTL = WDTPW + WDTHOLD;          //关闭看门狗
    int_clk();                         //系统时钟初始化
    int_usart();                       //初始化串口
    _EINT();                           //使能中断
    while(1);
}
```

该例子通过实际测试，满足设计要求。如果需要更加详细地了解，可以参考 5.1.5 小节寄存器说明部分或者相关数据手册。

5.2 SPI 接口

串行外围设备接口 SPI(Serial Peripheral Interface)总线是一种同步串行接口。因为 MSP430F149 本身具有硬件支持的 SPI 接口，所以软件的设置就非常简单，并且 CPU 有更多的时间去处理其他的任务。

SPI 总线可以连接多个主机和设备，因此 MSP430F149 单片机在连接 SPI 接口的 ADC/DAC、存储器、液晶等时，接口电路变得非常简单。

5.2.1 MSP430F149 单片机的同步操作

当 USART 模块的控制寄存器 UCTL 的 SYNC 位置位，并且 I^2C 位复位时将工作在同步模式下。通过 4 线(SOMI、SIMO、UCLK 和 STE)或 3 线(SOMI、SIMO 和 UCLK)与外界通信，具有如下特点：

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

5.2.2　SPI 初始化或重新配置流程

SPI 初始化或重新配置流程如下：

① 置位 SWRST；

② 在 SWRST=1 的情况下，初始化 USART 寄存器；

③ 通过 MEx SFRs 使能 USART 模块；

④ 通过软件置零 SWRST；

⑤ 如果需要，通过 IEx，SFRs 使能中断。

若不按此流程，将会得到不可预料的结果！

5.2.3　SPI 模式的引脚

引脚 SOMI、SIMO、UCLK 和 STE 用于 SPI 模式，其中 SOMI、SIMO 和 UCLK 在主机和从机模式下有差别，如表 5.1 所列。

表 5.1　主、从模式下的区别

引　脚	含　义	主机模式	从机模式
SIMO	从进主出	数据输出引脚	数据输入引脚
SOMI	从出主入	数据输入引脚	数据输出引脚
UCLK	USART 时钟	输出时钟	输入时钟

其中，STE 是从机模式发送/接收允许控制引脚，控制多主从系统中的多个从机。该引脚不用于 3 线 SPI 而用于 4 线 SPI 操作中，以使多主机共享总线，避免发生冲突。STE 具体含义如下。

① 工作在从机模式下：

STE　0，允许从机发送接收数据，SOMI 正常工作；

STE=1，禁止从机发送接收数据，SOMI 被强制进入输入状态。

② 工作在主机模式下：

STE=0，SIMO 和 UCLK 被强制进入输入状态；

STE=1，SIMO 和 UCLK 正常操作。

5.2.4　SPI 的操作方式

SPI 是全双工的，即主机在发送的同时也在接收数据，并且时钟的极性和相位也是可编程

的。只是要求相关设置与各设备匹配，该接口支持主机和从机两种模式。

1. SPI 主机模式

图 5.21 所示为 USART 模块为同步模式时，与另一个 SPI 从机设备的连接。

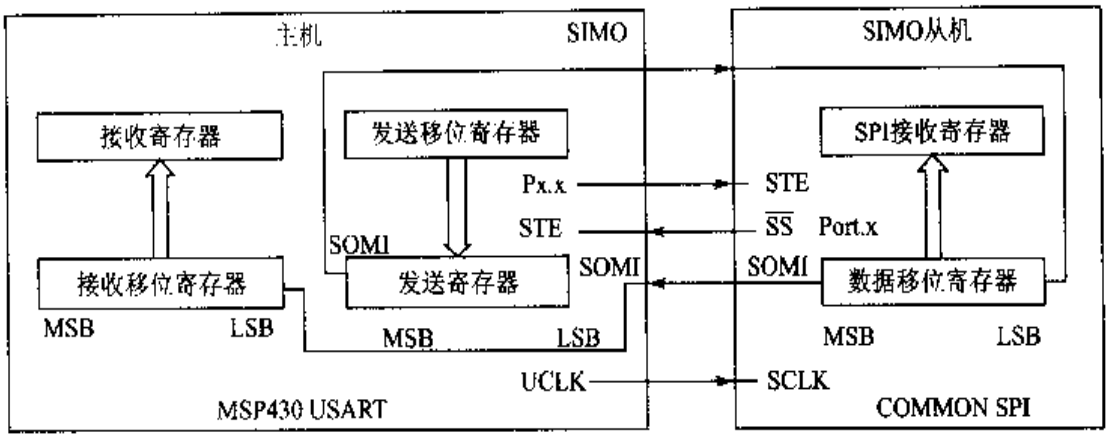


图 5.21　USART 模块与另一个 SPI 从机设备的连接

当控制寄存器中 MM=1 时，工作在主机模式。USART 通过 UCLK 信号控制通信，在第一个 UCLK 周期，数据由 SIMO 移出，并在相应的 UCLK 周期的中间从 SOMI 引脚锁存数据。每当移位寄存器的内容为空时，就把发送缓冲区的内容移入移位寄存器，并启动发送数据，高位在前；同时，接收到的数据移入移位寄存器中。当达到所设定的位数后，将之移入缓存 URXBUF 中，并设置中断标志 URIFG，表明接收到一个数据。在接收的过程中，最先收到的为最高有效位。当前一个数据未被读取时，将置位溢出位 OE。当数据从移位寄存器发给主机后，可立即用 UTXIFG 标志将数据从缓冲移入移位寄存器，开始一次发送数据操作。

注意：UTXIFG 标志不能代表数据移入完成。

URXIFG 表示数据移出完成，主机可利用 URXIFG 确定从机已经准备好接收数据。

在使用 4 线通信时，通过激活主机的 STE 信号来防止与别的主机发生总线冲突。主机在 STE 信号为高电平时正常工作。当 STE 信号为低电平时，表明另一个设备申请了主机，当前主机作出以下反应：

① SIMO 和 UCLK 引脚被强制输入，不再驱动 SPI 总线；

② 出错标志 FE 和 URCTL 中的中断标志 URXIFG 将被置位。

在 3 线模式当中，STE 输入信号与控制无关。

2. SPI 从机模式

当选择同步模式并且 MM=0 时，工作在从机模式。通信时钟来自外部主机，从机的 UCLK 引脚为输入状态。

在开始 UCLK 之前，由 UTXBUF 装入移位寄存器中的数据在主机提供的时钟下，通过从机的 SOMI 对外发送给主机。同时，在 UCLK 的反向沿，SIMO 引脚上的数据移入接收寄

存器中。如果接收中断标志 URXIFG=1，则表示数据已接收到缓存器中；如果前一个数据还没有被取出，则溢出位将被置位。

在使用4线同步通信时，STE 信号被从机用作发送和接收允许信号，它由主机提供：

① STE=0，该从机被允许接收和发送；

② STE=1，该从机被禁止接收和发送。

STE=1 时，已经启动的接收操作将被暂停，直到 STE=0。

5.2.5 SPI 的使能

SPI 模式下，接收/发送位 USPIEx 决定 SPI 是否使能。当 USPIEx=0 时，USART 将当前操作完成后停止；如果当前没有操作，则立刻停止。上电清除或置位 SWRST 时，将会立刻禁止 USART，并且使传输数据停止。

主、从机模式下发送使能示意图分别如图 5.22 和图 5.23 所示。当 USPIEx=0 时，任何数据写入 UxTXBUF 均无效，不会发送；只有在 USPIEx=1 和 BRCLK 有效时，写入 UxTXBUF 的数据才会发送。

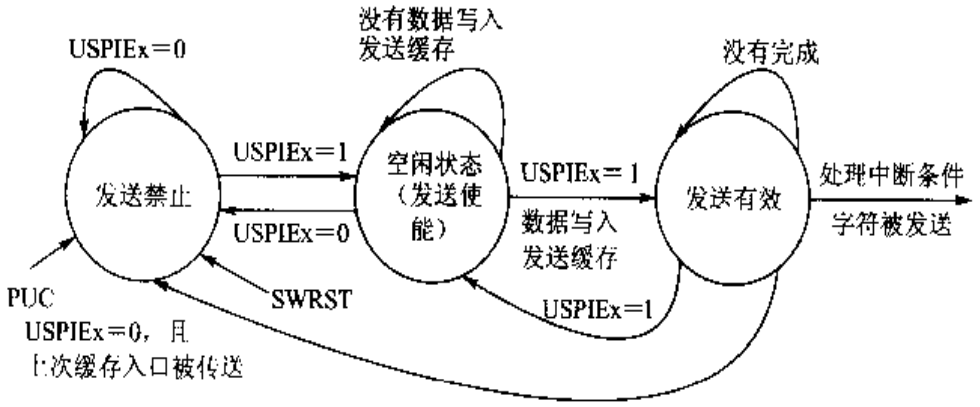


图 5.22 主机模式下发送使能示意图

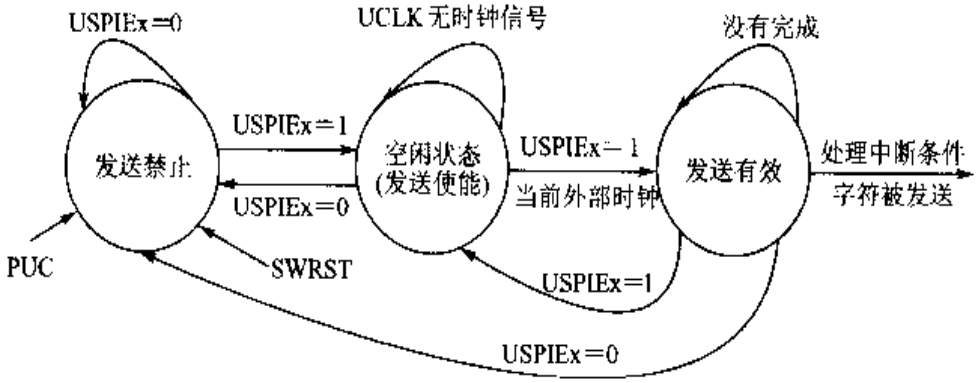


图 5.23 从机模式下发送使能示意图

主、从机模式下接收使能示意图分别如图 5.24 和图 5.25 所示。只有当 USPIEx=0 时，接收才使能。

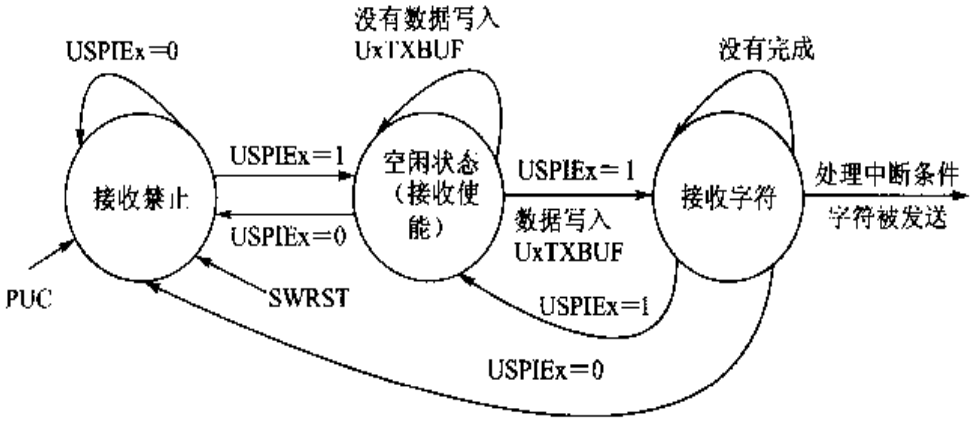


图 5.24 主机模式下接收使能示意图

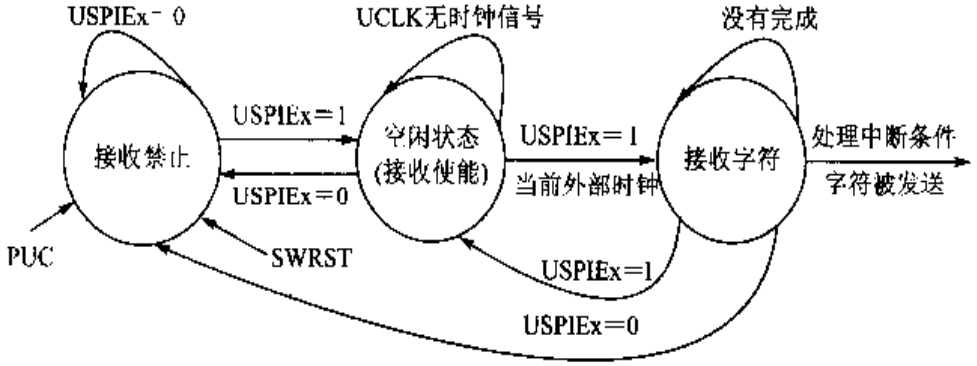


图 5.25 从机模式下接收使能示意图

5.2.6 SPI 的中断

1. 发送中断

USART 有一个发送中断和一个接收中断，图 5.26 所示为发送中断操作。

发送中断标志位 UTXIFGx 由发送器置位，同时表明 UxTXBUF 中可以装入下一个数据。此时，如果 UTXIEx 和 GIE 位为 1，就会产生一个中断请求。如果中断被执行或新的数据写入 UxTEBUF 中，UTXIFGx 标志将被自动复位。

注意：当 UTXIFGx=0 和 USPIEx=1 时，写入数据到 UxTXBUF，将会发生一个错误的结果。

2. 接收中断

接收中断操作如图 5.27 所示。

当数据被接收到 UxRXBUF 中时，接收中断标志 URXIFGx 将被置位。此时，如果 URX-

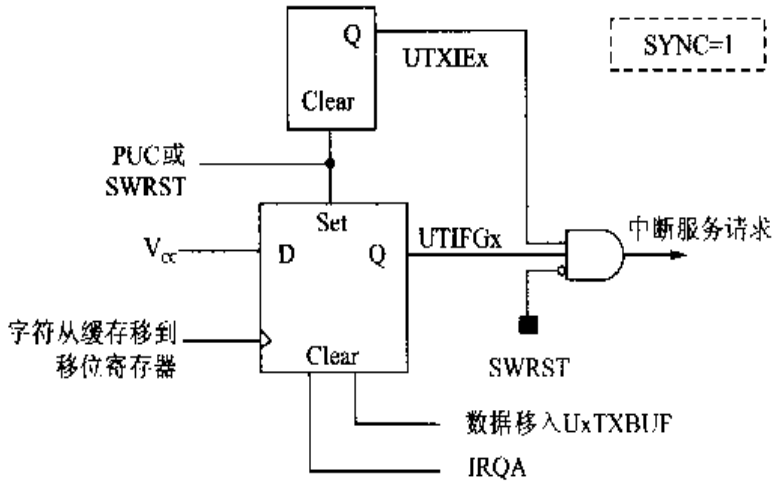


图 5.26　发送中断

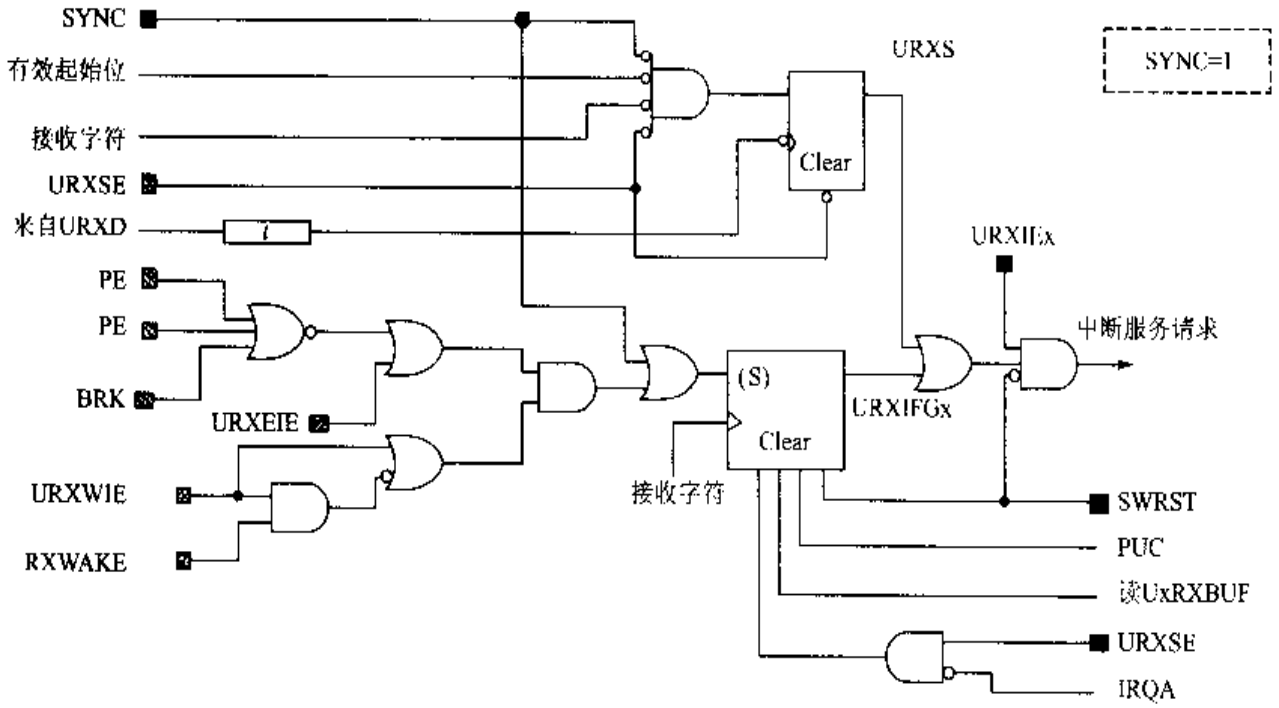


图 5.27　接收中断

IEx 和 GIE 位都为 1，就会产生一个中断请求；如果中断被执行或 UxRXBUF 中的数据被读取，则 URXIFGx 标志将被自动复位。

5.2.7　同步模式下的寄存器

1. UxCTL 寄存器

控制寄存器 UxCTL 的位分配见图 5.28。

7	6	5	4	3	2	1	0
Unused	Unused	I²C	CHAR	LISTEN	SYNC	MM	SWRST
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

图 5.28 UxCTL 寄存器

该寄存器的各位定义如下：

Unused Bits7~6 未使用。

I²C Bit5 模式选择位。当 SYNC=1 时，选择 SPI 或 I²C 模式。

0 SPI 模式；

1 I²C 模式。

CHAR Bit4 字符长度，选择 7 位或 8 位发送。当选择 7 位时，不发送或接收缓存的最高位，补 0。

0 7 位；

1 8 位。

LISTEN Bit3 选择是否将发送数据由内部反馈给接收器。

0 无反馈；

1 有反馈，通常又称为自闭环模式。

SYNC Bit2 USART 模块的模式选择。

0 UART 模式；

1 SPI 模式。

MM Bit1 主机模式或从机模式选择位。

0 从机模式；

1 主机模式。

SWRST Bit0 软件复位使能。

0 USART 复位释放，允许工作；

1 USART 处于复位状态。

2. UxTCTL 寄存器

发送控制寄存器 UxTCTL 的位分配见图 5.29。

7	6	5	4	3	2	1	0
CKPH	CKPL	SSELx		Unused	Unused	STC	TXEPT
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

图 5.29 UxTCTL 控制寄存器

该寄存器的各位定义如下：

CKPH	Bit7	时钟相位控制位。 0　SPICLK 使用正常的 UCLK 时钟； 1　UCLK 延时半个周期后作 SPICLK 信号。
CKPL	Bit6	SPICLK 信号时钟极性控制位。 0　时钟信号的低电平为无效电平，数据在 UCLK 上升沿输出，输入数据在 UCLK 的上升沿被锁存； 1　时钟信号的高电平为无效电平，数据在 UCLK 下降沿输出，输入数据在 UCLK 的上升沿被锁存。 数据的输入/输出与时钟信号相位和极性之间的关系如图 5.30 所示。
SSEL1,SSEL0	Bits5～4	时钟源选择位。这两位确定波特率的时钟源。 00　外部时钟 UCLK(仅对从机模式有效)； 01　辅助时钟 ACLK(仅对主机模式有效)； 10　子系统主时钟(仅对主机模式有效)； 11　子系统主时钟(仅对主机模式有效)。
Unused	Bits3～2	未使用。
STC	Bit1	从机发送控制位。 0　4 线模式； 1　3 线模式，此时 STE 不起作用。
TXEPT	Bit0	发送器空标志。 0　有数据正在发送或在发送缓冲器中； 1　发送移位寄存器和 UTXBUF 为空。

3. UxRCTL 寄存器

接收控制寄存器 UxRCTL 的位分配见图 5.31。

该寄存器的各位定义如下：

FE	Bit7	帧错误标志。 0　没有帧错误； 1　帧错误。
OE	Bit6	溢出标志位。 0　无溢出； 1　有溢出。
Unused	Bits5～0	未使用。

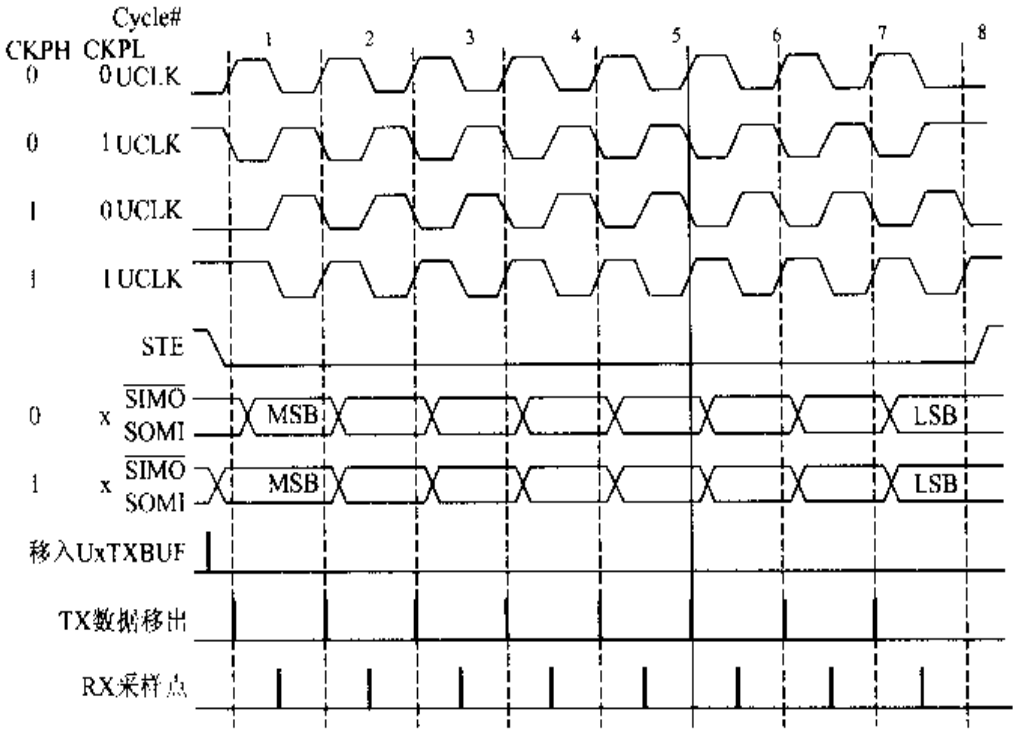


图 5.30 数据的输入/输出与时钟信号相位和极性之间的关系图

7	6	5	4	3	2	1	0
FE	Unused	OE	Unused	Unused	Unused	Unused	Unused
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

图 5.31 UxRCTL 控制寄存器

4. UxBR0、UxBR1 寄存器

波特率选择寄存器 UxBR0、UxBR1 的位分配分别如图 5.32 和图 5.33 所示。它们用来确定波特率的整数部分。其中 UxBR0 为低字节，UxBR1 为高字节。

7	6	5	4	3	2	1	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

图 5.32 UxBR0 寄存器

7	6	5	4	3	2	1	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

图 5.33 UxBR1 寄存器

5. UxMCTL 寄存器

波特率调整控制寄存器 UxMCTL 如图 5.34 所示。该寄存器不能用在 SPI 模式，并且要设为 00H。

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

图 5.34　UxMCTL 寄存器

6. UxRXBUF 寄存器

接收数据缓存 UxRXBUF 的位分配如图 5.35 所示。在 7 位数据时，接收缓存右对齐，最高位为 0。

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

图 5.35　UxRXBUF 寄存器

7. UxTXBUF 寄存器

发送数据缓存 UxTXBUF 的位分配如图 5.36 所示。在 7 位数据时，接收缓存右对齐，最高位为 0。

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

图 5.36　UxTXBUF 寄存器

5.2.8　应用设计举例

在本书的配套实验板上，使用 MSP430F149 单片机的 SPI 接口扩展了 74HC164 来驱动 LED 数码管。由于实验板的 SPI 接口同时需要读/写 DAC TLC5615 和 Flash M25P80，并且 74HC164 本身没有片选信号，在电路设计中我们采用一个“与”门和 MSP430F149 单片机的一个 I/O 口来控制 74HC164 的时钟信号，相当于片选信号，电路如图 5.37 所示。在配套实验板上扩展了 4 位 LED，可以实现的功能是循环显示数字 0～9。

实现该功能的程序流程图如图 5.38 所示。

具体程序如下：

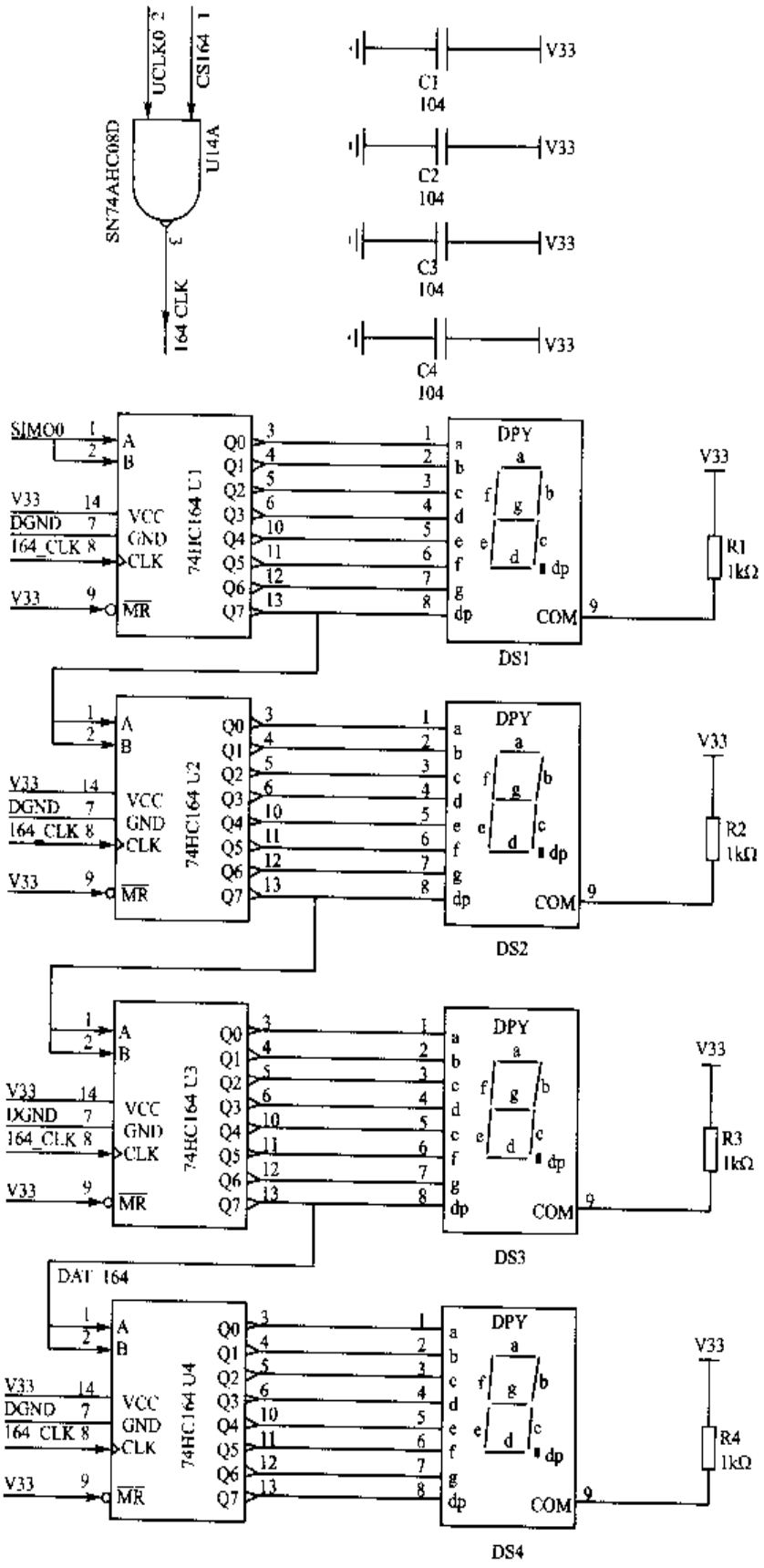


图 5.37　LED 显示原理图

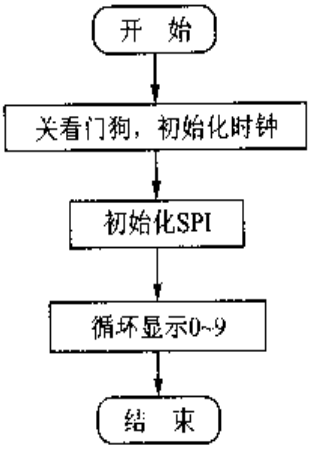


图 5.38　软件流程图

```
#include <msp430x14x.h>
#define uchar unsigned char
#define uint  unsigned int
#define led   BIT7
const uchar  segment[10] = {0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
void int_clk()
{
    uchar i;
    BCSCTL1&= ~XT2OFF;                    //打开 XT 振荡器
    BCSCTL2|= SELM1 + SELS;               //MCLK 为 8 MHz,SMCLK 为 1 MHz
    do
    {
        IFG1 &= ~OFIFG;                   //清除振荡错误标志
        for(i = 0; i < 100; i++)
        _NOP();                           //延时等待
        }
    while ((IFG1 & OFIFG) != 0);          //如果标志为 1,则继续循环等待
    IFG1&= ~OFIFG;
}
void int_spi()
{
    U0CTL|= SWRST;
    U0CTL  = CHAR + SYNC + MM;
    U0TCTL  = SSEL1 + SSEL0 + STC;        //三线模式 SMCLK 作为 SPI 时钟
    U0TCTL|= CKPH;
    U0TCTL&= ~CKPL;                       //CKPL CKPH:01 上升沿写数据
    U0BR0 = 0X02;
```

```
    U0BR1 = 0X00;
    U0MCTL = 0X00;
    ME1 |= USPIE0;                          //使能 SPI
    U0CTL&= ~SWRST;
    IE1&= ~UTXIE0;
    IE1&= ~URXIE0;                          //禁止中断
    P3SEL  = 0x0E;
    P3DIR |= BIT2 + BIT4;                   //选择第二功能和 I/O 方向
}
write_byte(uchar byte)
{
    U0TXBUF = byte;
    while((IFG1&UTXIFG0) == 0);             //UTXIFG0 = 0,表示数据从发送
    IFG1&= ~UTXIFG0;
}
display(uchar data)
{
    P2DIR|= led;
    P2OUT|= led;                            //使能 164CLK
    write_byte(segment_data);               //写数据
    P2OUT&= ~led;                           //禁止 164CLK
}
delay()
{
    uint i,j;
for(i = 0;i <= 1000;i++)
    {
        for(j = 0;j <= 1000;j++)
            ;
    }
}
int main( void )
{
    uchar i;
    WDTCTL = WDTPW + WDTHOLD;               //关闭看门狗
    int_clk();                              //初始化时钟
    int_spi();                              //初始化 SPI
    while(1)
    {
```

```
        for(i=0;i<=9;i++)
        {
        delay();
        display(i);                    //循环显示 0~9
        }
    }
}
```

从以上的例子可知，SPI 的数据输出和时钟连接到 74HC164，就可以实现串/并转换，74HC164 在单片机系统中常用作 LED 数码管的显示驱动，可以节省 I/O 口的资源。但由于 74HC164 没有片选信号，所以作了控制。读者在自己的系统中可以参考该电路，关于 SPI 更进一步的应用可参照实践篇第 9 章的 M25P80 部分。

第6章

比较器 Comparator_A

6.1 Comparator_A 概述

Comparator_A 是一个模拟比较器，它支持高精度的斜坡模拟/数字转换和电源电压管理，以及外部模拟信号监测功能。它的主要特点如下：

- 反相和同相输入信号可以通过多路选择器进行选择；
- 软件可编程比较输出 RC 滤波器；
- 输出可以作为 Timer_A 的捕捉输入；
- 输入缓冲器控制软件可编程；
- 中断功能；
- 可以选择参考电压发生器；
- 可关断比较器和比较器参考电压发生器。

比较器的基本结构如图 6.1 所示。

从图 6.1 中我们可以看到，MSP430 单片机的比较器单元主要由以下几个部分组成：

① 输入选择器，主要用于选择比较器输入信号。

② 参考电压发生器，主要实现内部模拟参考信号的产生。

③ 比较单元，实现输入信号的比较。当正向信号大于负向信号时，输出高电平；当负向信号大于正向信号时，输出低电平。

④ 输出滤波单元，主要实现对比较器的输出进行 RC 滤波。

下面将对各个单元的操作步骤和注意事项进行详细说明。

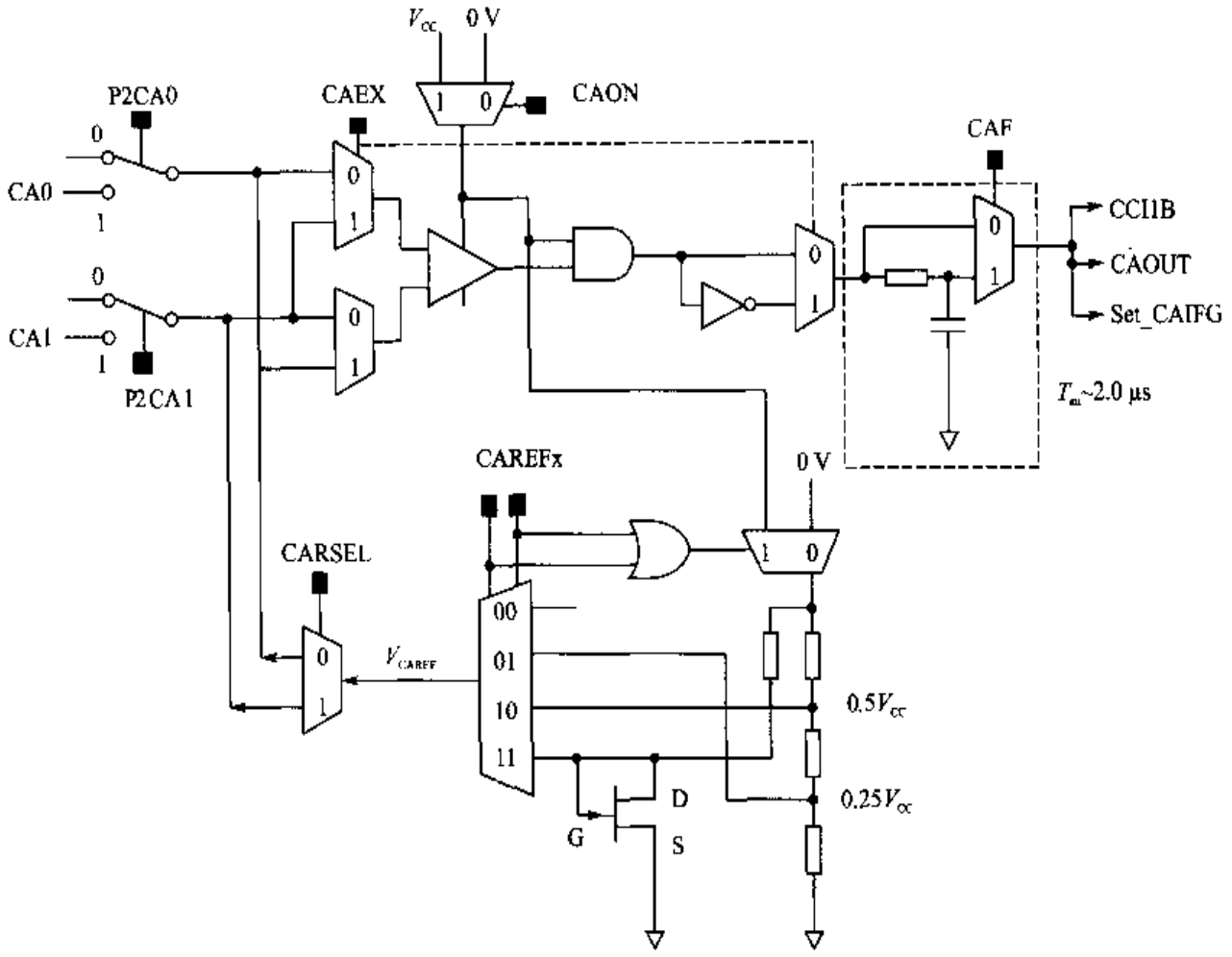


图 6.1　Comparator_A 基本结构图

6.2　Comparator_A 的工作原理和操作流程

6.2.1　基本工作原理

比较器 A 主要完成对比较器正向输入端和反向输入端的模拟信号的电压的比较。当正向输入信号大于反向输入信号时，比较输出信号为高电平，反之为低电平。注意：比较器实际输出可以通过寄存器的 CAEX 位进行取反，当 CAEX 等于 1 时输出取反。

同时，比较器单元可以通过寄存器操作来关断和打开。在不使用比较器的情况下，通过关断比较器可以降低系统功耗。当比较器被关闭后，CACTL2 寄存器的比较输出位 CAOUT 通常为低电平。

6.2.2 基本操作流程

Comparator_A 的基本操作流程如下：

① 打开比较器单元；

② 打开参考电压发生器单元(可选，如果比较器的输入信号全部是外部输入，则可关闭该单元)；

③ 选择相应输入信号(CA0、CA1 和内部参考)连接到比较器的输入端口；

④ 单片机的比较器 A 的外部输入端口 CA0 和 CA1 分别接至 P2.3 和 P2.4 口；

⑤ 选择配置相关寄存器(见 6.3 节寄存器说明部分)；

⑥ 使能中断信号(如果需要的话)；

⑦ 读取比较输出信号。

6.2.3 模拟输入信号选择

通过模拟选择单元电路选择输入信号 CAx、内部参考电压发生器和比较器的输入端口连接。具体操作请参考 6.3 节寄存器说明部分。

注意：当比较器 A 打开时，它的输入信号就应该连接相关外部输入信号、参考电源或地信号，否则会引起不可预料的中断并增加系统功耗。

寄存器的 CAEX 位控制输入多路开关，通过控制位 CARSEL 决定输入信号与比较器的正端或负端连接。

另外，CAEX 位还可以控制输出的反转。利用这个特性，用户可以测量或者补偿偏置电压。

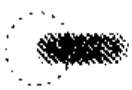
CA0、CA1 和内部参考电压发生器通过模拟选择单元电路选择连接到比较器的正、负输入端。

CA0、CA1 分别与外部引脚 P2.3 和 P2.4 连接。

6.2.4 输出滤波器选择

比较器的输出信号可以通过 CAF 位来选择是否采用内部输出滤波器。当两个输入端信号相差很小、内部电路和外部电路寄生效应、再加上外部信号耦合到两个输入端口、电源以及系统其他部分的干扰时，将会引起如图 6.2 所示的振荡。

通过图 6.2 可以清晰地看到使用输出滤波器和不使用输出滤波器的效果差别，因此建议使用比较单元的时候选择内部输出滤波器。



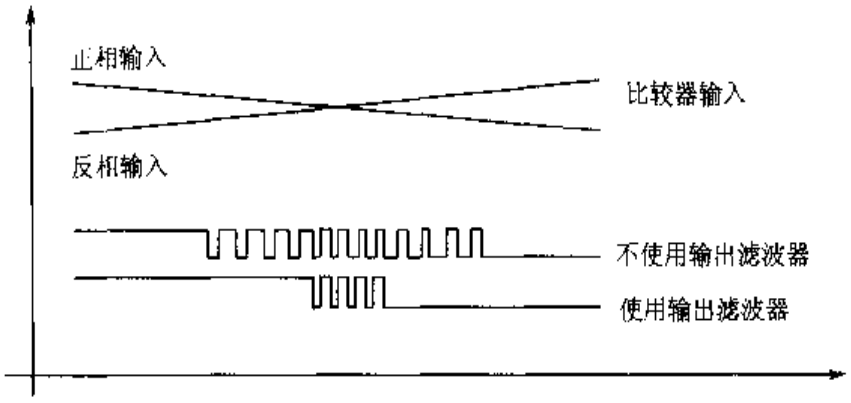


图 6.2　比较器输出端的振荡

6.2.5　参考电压的产生和选择

参考电压发生器主要用来产生 V_{CAREF} 信号，并且可以连接到比较器的任意一个输入端。寄存器的 CAREFx 位控制参考电压的产生，CARSEL 位控制参考电压和比较器输入端的连接（内部参考电压发生器可以产生多个参考电压，具体可以参考 6.3 节寄存器说明部分）。如果比较器的 2 个输入端口都和外部信号连接，则必须关闭内部参考电压发生器。

6.2.6　比较器中断

比较器 A 的中断相关包括 1 个中断标志和 1 个中断向量。

Comparator_A 的中断系统如图 6.3 所示。

中断标志 CAIFG 在比较器的输出上升沿或者下降沿被置位（通过 CAIES 位选择）。如果比较器中断使能位 CAIE 和总中断使能位 GIE 被置位，则 CAIFG 的置位将产生一个中断。中断被响应后，CAIFG 标志自动清零。

CAIFG 位也可以软件清零。

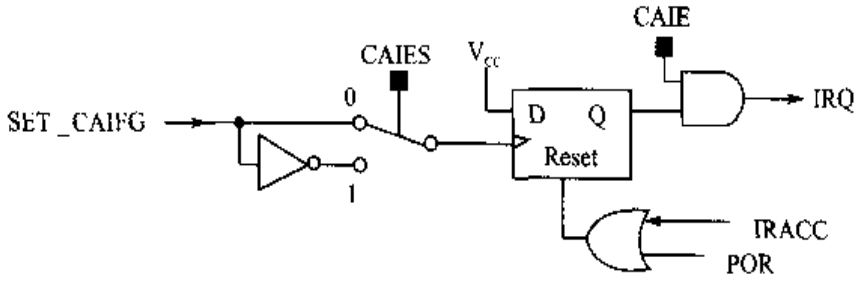


图 6.3　Comparator_A 的中断系统

6.3 寄存器说明

1. CACTL1 寄存器

CACTL1 是一个 8 位的寄存器，该寄存器包含比较器的大部分控制位。CACTL1 寄存器的各个位如图 6.4 所示。

7	6	5	4	3	2	1	0
CAEX	CARSEL	CAREFx		CAON	CAIES	CAIE	CAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

图 6.4 CACTL1 寄存器

下面对 CACTL1 寄存器的各个位进行详细介绍。

CAEX Bit7 比较器 A 正端和反向端输入交换，是比较器 A 的输出取反控制位。

0 输入见 CARSEL 位、输出不取反；

1 输入见 CARSEL 位、输出取反。

CARSEL Bit6 比较器 A 参考选择位，该位选择比较器的输入端和 V_{CAREF} 连接情况。

注意：如果参考电压选择为正端输入，那么外部引脚必须选择 CA1 连接到负端；如果参考电压选择为负端输入，那么外部引脚必须选择 CA0 连接到正端。当希望比较器的输入信号都和外部引脚连接的时候，必须关闭内部参考发生器。

当 CAEX=0 时：

0 V_{CAREF} 连接到输入的正端，CA1 或 CA0 可以作为负端输入，输出不反向。当内部参考电压发生器关闭时，CA1 输入到负端，CA0 输入到正端。

1 V_{CAREF} 连接到输入的负端，CA1 或 CA0 可以作为正端输入，输出不反向；当内部参考电压发生器关闭时，CA1 输入到负端，CA0 输入到正端。

当 CAEX=1 时：

0 V_{CAREF} 连接到输入的负端，CA1 或 CA0 可以作正负端输入，输出反向；当内部参考电压发生器关闭时，CA1 输入到正端，CA0 输入到负端。

1 V_{CAREF} 连接到输入的正端，CA1 或 CA0 可以作负端输入，输出反向；当内部参考电压发生器关闭时，CA1 输入到正端，CA0 输入到负端。

CAREF　Bits5～4　比较器 A 参考电压选择位，这两位选择参考电压 V_{CAREF}。

00　内部参考电压关闭；

01　参考电压等于 $0.25V_{CC}$；

10　参考电压等于 $0.50V_{CC}$；

11　选择二极管参考。

CAON　Bit3　比较器 A 使能，可以通过该位关闭比较器 A 以降低系统电源消耗。注意比较器和参考电压发生器是独立的，可以独立地对其进行操作。

0　关闭；

1　打开。

CAIES　Bit2　比较器中断触发边沿选择位。

0　上升沿；

1　下降沿。

CAIE　Bit1　比较器 A 中断使能信号。

0　禁止中断；

1　允许中断。

CAIFG　Bit0　比较器 A 中断标志位。

0　没有中断被挂起；

1　有中断被挂起。

2. CACTL2 寄存器

CACTL2 是一个 8 位的寄存器，包含比较器的控制位，主要是一些与输入/输出有关的信息。CACTL2 寄存器的各个位如图 6.5 所示。

7	6	5	4	3	2	1	0
Unused				P2CA1	P2CA0	CAF	CAOUT
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)

图 6.5　CACTL2 寄存器

下面将对 CACTL2 寄存器的各个位进行详细介绍。

Unused　Bits7～4　未使用位。

P2CA1　Bit3　CA1 引脚选择位。

0　CA1 不作为比较器的输入；

1　CA1 作为比较器的输入。

P2CA0　Bit2　CA0 引脚选择位。

0　CA0 不作为比较器的输入；

1　CA0 作为比较器的输入。

CAF　Bit1　比较器 A 输出滤波器选择。

0　比较器 A 输出滤波器不使能；

1　比较器 A 输出滤波器使能。

CAOUT　Bit0　比较器 A 输出信号，这一位反映比较器的的输出值，该位不可写（写该位无效）。

3. CAPD 寄存器

CAPD 是一个 8 位的寄存器，该寄存器用来控制比较器输入的缓冲功能。CAPD 寄存器的各个位如图 6.6 所示。

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

图 6.6　CAPD 寄存器

下面将对 CAPD 寄存器的各个位进行详细介绍。

CAPDx　Bits7～0　比较器 A 端口禁止位。这些位独立地禁止比较器 A 的输入缓冲，以防止寄生电流的产生。

0　输入缓冲使能；

1　输入缓冲禁止。

6.4　程序设计举例

本例主要实现 CA1 引脚的电压检测。当电压小于 0.25V_{CC} 时发出一声报警，提示用户电压过低。该例可以实现系统电源监测，读者可以通过调节 CA1 的电压来观察。实现该功能的程序流程如图 6.7 所示。

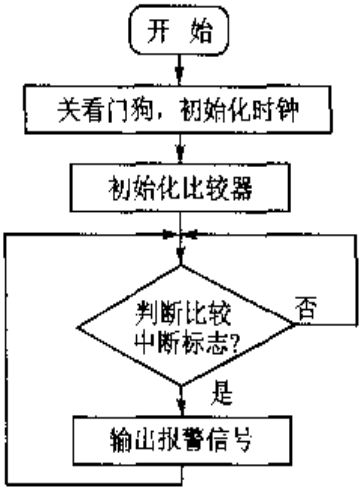


图 6.7　程序流程图

具体程序如下：

```
#include <msp430x14x.h>
#define  uint  unsigned int
#define  uchar  unsigned char
#define  beef  BIT6
uchar flag = 0;
```

```
void int_clk()
{
    uchar i;
    BCSCTL1& = ~XT2OFF;                    //打开 XT 振荡器
    BCSCTL2  = SELM1 + SELS;               //MCLK 为 8 MHz,SMCLK 为 1 MHz
    do
    {
        IFG1 &= ~OFIFG;                    //清除振荡错误标志
        for(i = 0; i < 100; i++)
        _NOP();                            //延时等待
        ;
    while ((IFG1 & OFIFG) != 0);           //如果标志为 1,则继续循环等待
    IFG1& = ~OFIFG;
}
void int_COMPARATORA()
{
    P2SEL  = 0X14;                         //P22、P24 分别作为比较器的输出和输入
    P2DIR  = 0X04;                         //设置 P22 作为输出
    CACTL1& = ~CAEX;                       //输出取反
    CACTL1 | = CARSEL + CAREF0 + CAON + CAIE + CAIES;     //选择参考电压接负端、CA1 接正端,允许
                                                          //中断下降沿触发中断,参考电压为 0.25Vcc
    CACTL2  = CAF + P2CA1;                 //使能输出滤波器,CA1 作为比较输入功能
}
#pragma vector = COMPARATORA_VECTOR
__interrupt void COMPARATORA(void)         //中断服务程序
{
    flag = 1;                              //置中断标志为 1
}
void delay()                               //软件延时
{
     uint i,j;
    for(i = 0;i <= 100;i++)
        ;
        for(j = 0;j<100;j++)
          ;
    ;
}
void main()
{
```

```
    WDTCTL = WDTPW + WDTHOLD;              //关看门狗
    int_clk();                             //初始化时钟
    delay();                               //调用软件延时子程序
    int_COMPARATORA();                     //初始化比较器
    _EINT();
    while(1)
      {
        while(flag)
            {
            P4DIR |= beef;
            P4OUT &= ~beef;                //P4.6 输出低电平
            delay();                       //调用软件延时子程序
            P4OUT |= beef;                 //P4.6 输出高电平
            flag = 0;                      //标志清零
            }
      }
}
```

由主函数可以看出，由于比较器本身的原因，即使输入发生微小的变化时，都会引起比较器的输出变化。虽然本程序采取了输出滤波功能，还是可以观察到如图 6.2 所示的波形，读者在使用时一定要注意这个特点。

通过对程序的学习，读者对 MSP430F149 单片机的比较单元有了一个深入的理解，同时可以在实际工程中加以应用。

第7章

ADC12

7.1 ADC模块基本介绍

ADC12支持12位高速模/数转换，可以在没有CPU干预的情况下进行16次独立采样并保存结果。ADC12模块主要由以下几个部分组成：SAR核、采样时钟电路、参考电压发生器、采样保持电路以及采样时间定时电路、多路模拟信号选择器、转换结果存储控制器及转换结果缓冲器。ADC12模块的结构如图7.1所示。

ADC12的主要特点如下：

- 大于200 ksps转换速度；
- 12位无失码转换；
- 软件或定时器可编程采样保持电路时间；
- 转换初始化可以通过用户软件、Timer_A或者Timer_B来实现；
- 片上参考电压软件可编程，1.5 V或2.5 V可选；
- 软件可编程选择内部或者外部参考电压；
- 8个独立的外部输入信号；
- 可采集内部温度传感器、AVcc和外部参考电压；
- 每个通道参考电压可以独立选择；
- A/D转换时钟软件可编程；
- 支持单通道单次、单通道重复、单次顺序及顺序重复采样模式；
- ADC核和参考电压模块可以独立关闭；
- ADC中断向量寄存器支持快速译码；
- 16个转换结果寄存器；
- DMA使能。

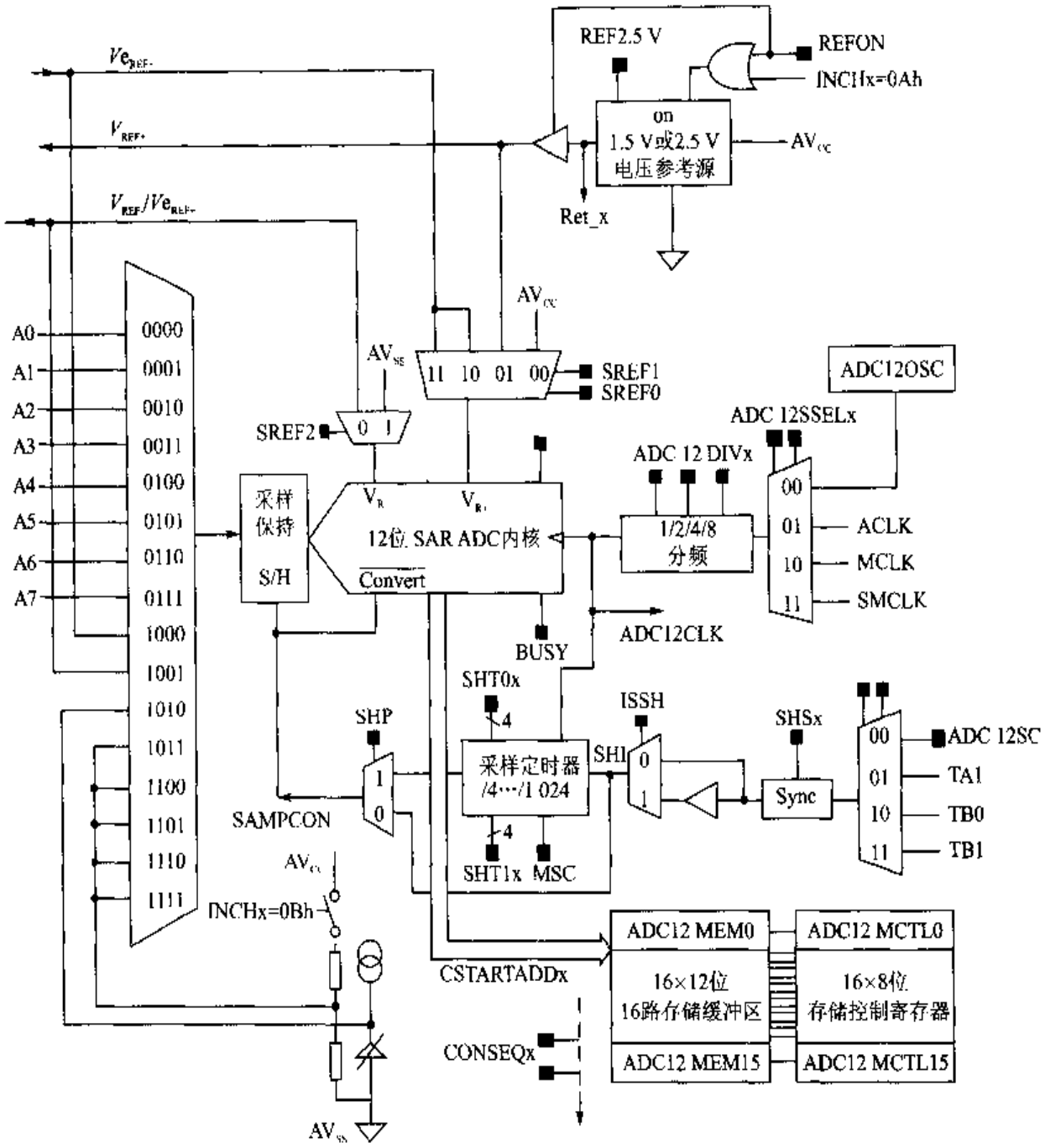


图 7.1 ADC 模块结构

7.2 ADC12 的操作流程

ADC12 操作的基本流程如下：

① 选择 ADC 转换时钟、参考电压、转换模式和存储器管理；

② 打开相关中断(可选)；

③ 启动 A/D 转换；

④ 进入中断或查询转换完成标志；

⑤ 采样转换时序；

⑥ 转换结果缓存及读取 A/D 转换值。

下面分别介绍每个步骤的操作方法。

1. 时钟选择

ADC12CLK 是 A/D 转换时钟，同其他的 ADC 器件一样，都需要在时钟的驱动下工作。同时，在脉冲采样工作方式下也用它来产生采样周期。我们可以通过 ADC12CTL1 寄存器的 ADC12SSELx 位选择时钟源。可以选择的时钟有 SMCLK、MCLK、ACLK 和内部振荡器 ADC12OSC。

同时，可以通过 ADC12DIVx 对时钟进行 1～8 倍分频，具体操作请参考 6.3 节寄存器说明部分。

注意：内部振荡器频率为 5 MHz，但是它的频率会由于器件的不同、电源电压和温度的改变而改变。在 A/D 转换过程中关闭时钟，转换将不会完成，并且结果是不正确的。

2. 参考电压

所有 A/D 转换器都需要一个参考电压源作为采样的基准，称为基准电压。MSP430F149 单片机的 ADC12 模块中自带参考电压发生器，可以得到 1.5 V 或 2.5 V 参考电压，并且参考电压还可以选择其他参考信号，具体操作请参考 6.3 节寄存器说明部分。

注意：为了正确的操作内部参考电压，我们需要在 V_{REF+}、V_{REF-} 和 AVss 之间加上电容。电容通常情况下取 10 μF 和 100 nF 并联，在系统中尽可能把 100 nF 的电容靠近单片机的参考引脚，而且 10 μF 的电容也要尽可能地靠近，这样电容才能起到稳定电压的作用。参考电源的最小建立时间为 17 ms，所以推荐打开参考电源的时候，过 17 ms 以后再使用。

当使用外部参考时，V_{R+} 和 V_{R-} 将分别通过 V_{eREF-} 和 V_{REF+} /V_{eREF-} 连接到 ADC12 模块内部。

3. ADC12 转换结果计算

ADC12 模块转换结果的计算公式如下：

$$N_{ADC} = 4095 \times \frac{V_{IN} - V_{R-}}{V_{R+} - V_{R-}} \tag{7-1}$$

其中，V_{IN} 等于输入模拟电压，V_{R+} 为参考电压的正电压，V_{R-} 为参考电压的负电压(一般取 0 V)。

4. 存储器管理

ADC12 模块包括 16 个 ADC12MEMx 转换结果存储器，每一个 ADC12MEMx 通过相关

的存储器控制寄存器 ADC12MCTLx 进行控制。详细介绍请参考 6.3 节寄存器说明部分。

ADC12CTL1 寄存器中 CSTARTADDx 位指明 ADC12MCTLx 控制寄存器的开始通道。如果是单次或者单次重复模式，则该位直接选择需要转换的信号；如果是多通道连续或者重复多通道连续模式，则 CSTARTADDx 指向 ADC12MCTLx 控制寄存器指向的通道。第一次转换结束后将自动转换下一通道，直到有通道的控制寄存器的 EOS 位等于 1；当转换结果被写入选择的 ADC12MEMx 寄存器时，相关的中断标志将置位。

5. 采样和转换时序

这部分控制采样及转换所需要的各种时序电路：ADC12CLK 转换时钟、SAMPCON 采样及转换信号、SHT 控制采样周期、SHS 控制的采样触发来源选择、ADC12SSEL 选择的内核时钟源及 ADC12DIV 选择的分频系数等。在时钟控制电路的指挥下，ADC12 的各部件能够协调工作。模拟到数字由采样输入信号 SHI 的上升沿初始化，SHI 信号通过 SHSx 位选择以下信号的一种：

① ADC12SC 位；

② 定时器 A 输出单元 1；

③ 定时器 B 输出单元 0；

④ 定时器 B 输出单元 1。

SHI 信号的极性可以通过 ISSH 位改变，SAMPCON 信号控制采样周期和启动转换开始。当 SAMPCON 信号等于高电平时，采样开始；SAMPCON 信号的下降沿开始转换。转换时间需要 13 个 ADC12CLK 时钟周期。通过设置 SHP 位可以选择两种不同的采样时序方法：外部采样模式和脉冲采样模式。

(1) 外部采样模式

当 SHI=0 时，采用直接采样方式。SHI 信号直接控制 SAMPCON 和采样周期的时间长度。当 SAMPCON 等于高电平时，采样有效。SAMPCON 信号的下降沿开始转换，如图 7.2 所示。

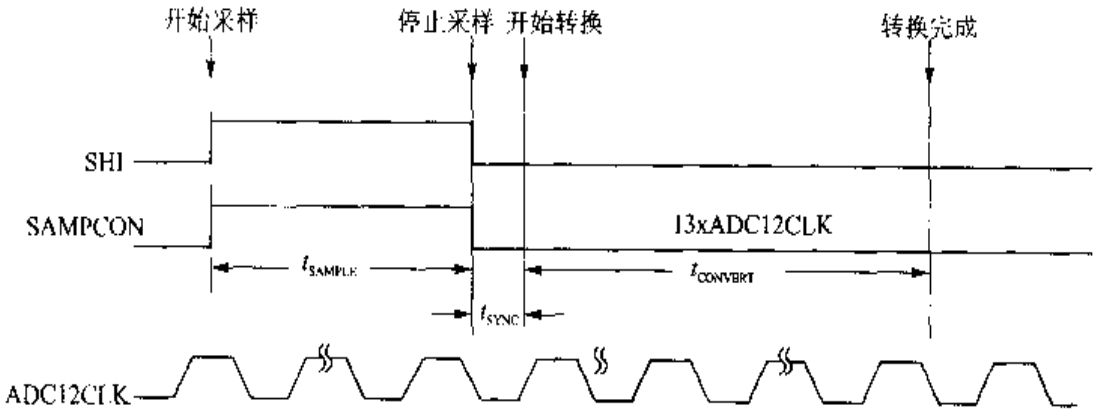


图 7.2 外部采样模式

(2) 脉冲采样

当 SHI=1 时，采用脉冲采样方式。

SHI 信号用于触发采样时钟，ADC12CTL0 寄存器的 SHT1x 和 SHT0x 位控制 ADC 内部采样时钟，控制 SAMPCON 采样周期。采样时钟保持 SAMPCON 信号为高电平，直到内部采样保持时钟规定时间，并且和 ADC12CLK 同步。

脉冲采样过程如图 7.3 所示。

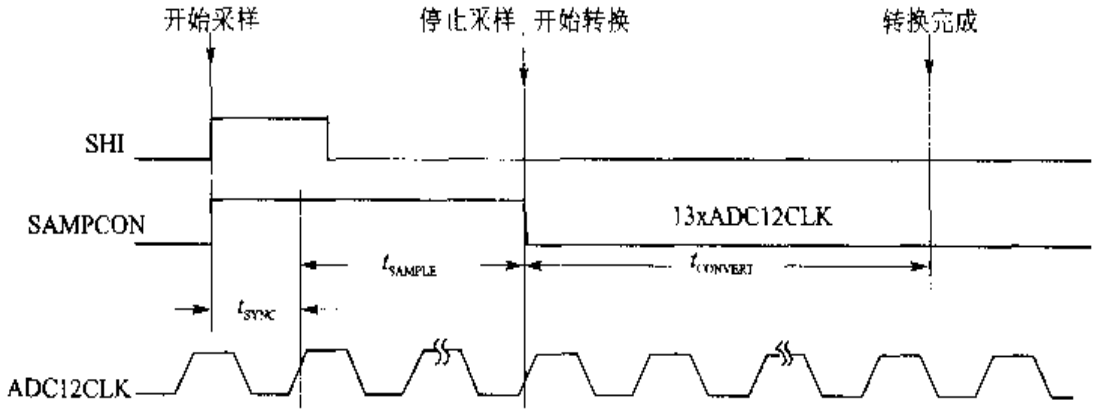


图 7.3 脉冲采样

6. 停止转换

停止转换根据工作模式的不同而不同。以下是推荐在不同模式下停止 ADC12 的方法：

- 当在单通道单次转换时，复位 ENC 位可以立即停止转换，并且此次转换结果不可预料。为了得到正确的结果，可以判断 ADC 转换忙信号，为 0 时再复位 ENC 位。
- 当在单通道连续转换时，复位 ENC 位会在此次转换完成后结束 ADC12。
- 当在多通道单次、连续转换时，复位 ENC 会在此次转换完成后结束 ADC12。

所有工作模式都可以通过设置 CONSEQx=0 和复位 ENC 位来立即结束 ADC12，此时数据不可预料。

注意：在连续模式下，如果 EOS=0，则复位 ENC 位不可以停止连续转换模式。为了停止 ADC12，应该首先选择单通道模式，然后再复位 ENC。

7. 转换结果缓存及读取 ADC 转换值

ADC12 有 12 个转换通道，设置了 16 个转换存储器用于暂存转换结果。合理设置之后，ADC12 硬件会自动将转换结果存放到相应的 ADC12MEMx 寄存器中，然后从这个寄存器中读出相应的值。

每个转换存储器 ADC12MEMX 都有自己对应的控制寄存器 ADC12MCTLX，控制寄存器控制各个转换存储器基本的工作方式。

7.3 转换模式

ADC12 有 4 种操作模式，分别为单通道单次转换、序列通道单次转换、单通道多次转换和序列通道多次转换，如表 7.1 所列。用户可以通过 ADC12CTL1 寄存器的 CONSEQx 位来进行选择，具体参考表 7.1 或者 6.3 节寄存器说明部分。

表 7.1 ADC12 的操作模式

CONSEQx		操作模式
0	0	单通道单次转换
0	1	序列通道单次转换
1	0	单通道多次转换
1	1	序列通道多次转换

1. 单通道单次转换

单次单通道转换通过 ADC12CTL1 寄存器的 CSTARTADDx 位控制转换结果被写入相应的 ADC12MEMx 寄存器。同时，通过寄存器 ADC12MCTLx 选择需要转换的输入信号通道以及参考电压。

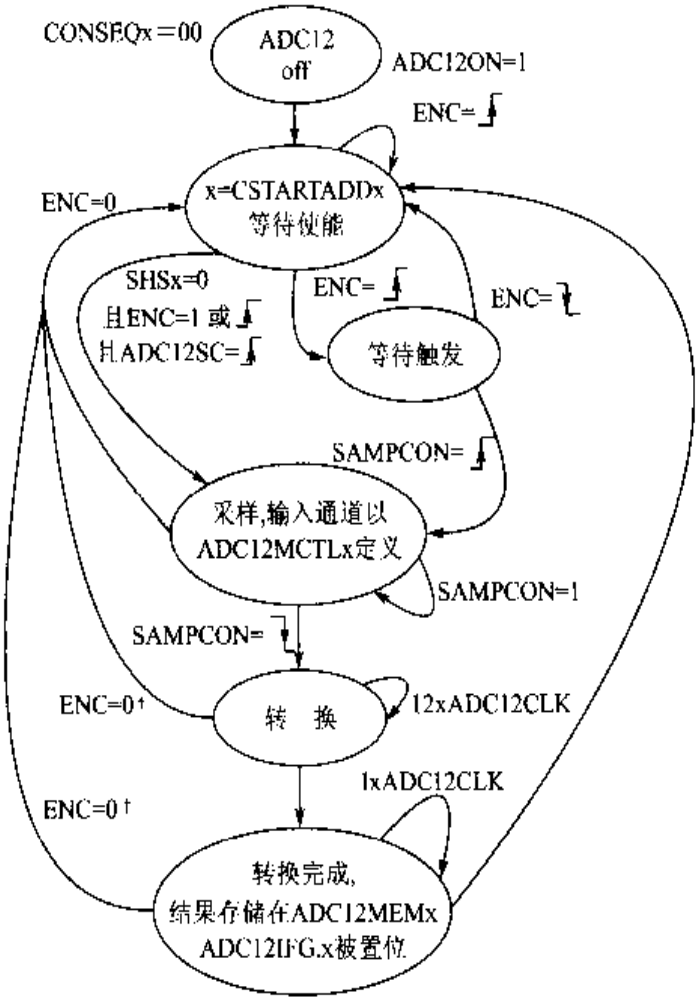


图 7.4 单通道单次模式的状态转换图

对单通道单次模式有以下设置：

- x=CSSTARTADD，指向选择的 ADC12MEMx 和 ADC12MCTLx 寄存器；
- ADC12MEMx 存放转换结果；
- ADC12MCTLx 寄存器中定义通道和参考电压。

当 ADC12SC 触发一次转换时，下一次转换就可以通过简单地设置 ADC12SC 位来启动。然而，当有其他任何触发源用于开始转换时，ENC 必须在每次转换之间固定。其他的采样输入信号将在 ENC 复位并置位前被忽略。

图 7.4 所示为单通道单次模式的状态转换图。该图描述了通过两种方式启动 ADC12 转换，详细请参考 6.3 节寄存器说明。其中，x 为指向 ADC12MCTLx 的指针；↓指转换结果不可预料。读者可以参考 7.5 节中的

相关例子。

2. 序列通道单次转换模式

对序列通道单次转换有如下设置：

- ADC12MEMx 存放结果；
- EOS=1 标志最后通道，非最后通道的 ESO 位都是 0，表示转换没有结束；
- ADC12MEMX 内存放转换结果；
- ADC12MCTLX 寄存器中定义通道和参考电压。

当有其他任何触发源用于开始转换时，ENC 必须在每次转换之间固定。其他的采样输入信号将在 ENC 复位并置位前被忽略。

图 7.5 所示为序列通道单次转换状态图。

其中，x 为指向 ADC12MCTL.x 的指针。

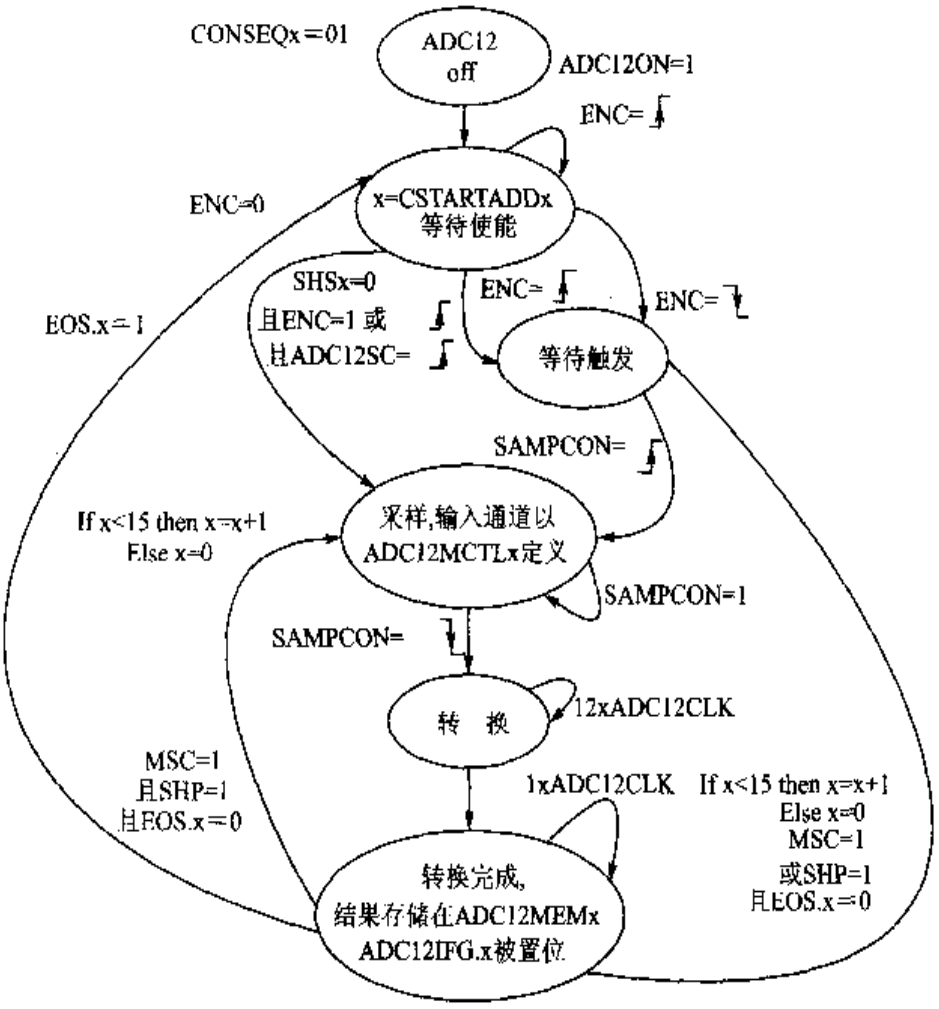


图 7.5 序列通道单次转换状态图

3. 单通道多次转换

对选定的通道作多次转换，直到关闭该功能或 ENC=0，有如下设置：

- x=CSSTARTADD，指示转换开始通道；
- ADC12MENX 存放转换结果；
- ADC12MCTLX 寄存器中定义通道和参考电压。

在这种模式下改变转换模式，不必先停止转换；在当前正进行的转换结束后，可以改变转换模式。

该模式的停止可以采用以下方法：

- 使 CONSEQ=0，改变为单通道单次转换；
- 使 ENC=0 直接使当前转换完成后停止；
- 使用单通道单模式替换当前模式，同时使 ENC=0。

图 7.6 所示为单通道多次转换的状态图。

其中，x 为指向 ADC12MCTLx 的指针。

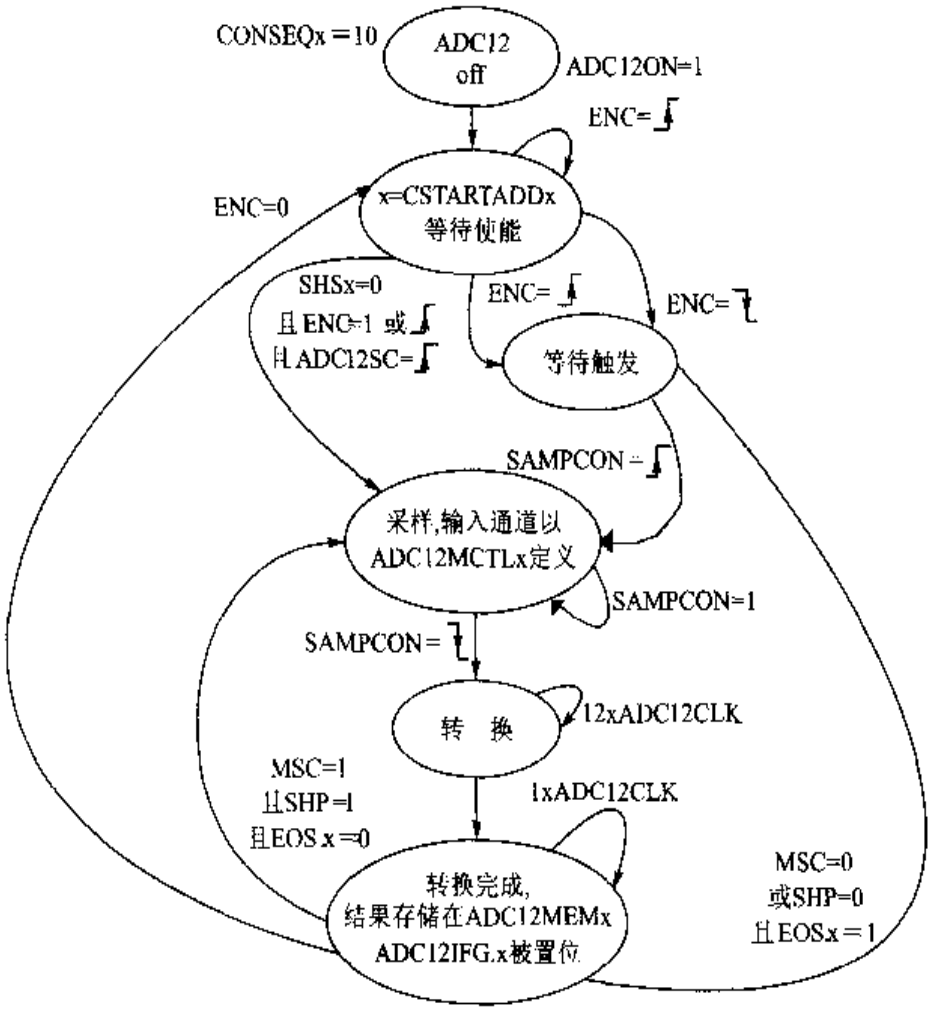


图 7.6 单通道多次转换的状态图

4. 序列通道多次转换

对序列通道作多次转换，直到关闭该功能或 ENC=0，同时有如下设置：

- X=CSSTARTADD，指示转换开始通道；
- EOC 标志系列通道中最后的通道；
- ADC12MCTLX 寄存器中定义通道和参考电压。

改变转换模式不必先停止当前转换。一旦改变模式（单通道单次模式除外），将在当前序列完成后立即生效。

图 7.7 所示为序列通道多次转换的状态图。其中，x 为指向 ADC12MCTLx 的指针。

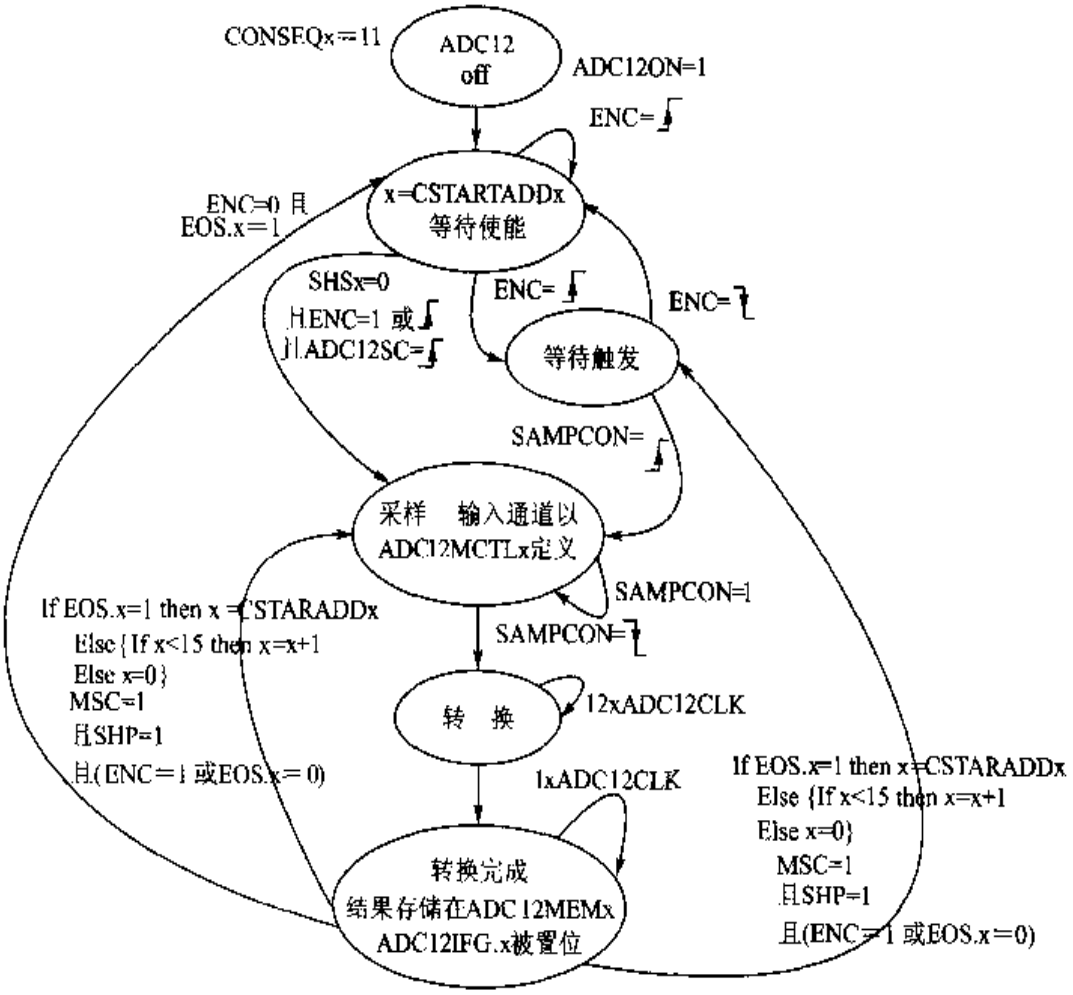


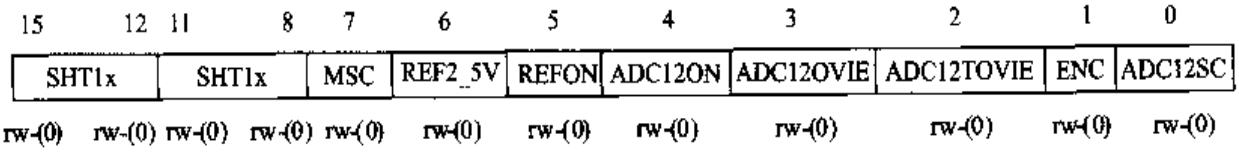
图 7.7 序列通道多次转换的状态图

7.4 ADC12 寄存器详细介绍

ADC12 有大量的控制寄存器提供给用户。下面便介绍 ADC12 的寄存器。用户可以根据实际需求通过软件来设置 ADC12 寄存器，从而灵活运用 ADC12 的各个功能模块。

1. ADC12CTL0 寄存器

ADC12CTL0 寄存器是一个 16 位的寄存器，该寄存器主要控制 ADC12 模块的工作。ADC12CTL0 寄存器的位分配如图 7.8 所示。



注：阴影部分只有当 ENC=0 时才能修改。

图 7.8 ADC12CTL0 寄存器

下面介绍 ADC12CTL0 寄存器的各个位。

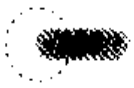
SHT1x　Bits15～12　采样保持时间选择位。当采用脉冲采样模式时，这些位定义寄存器 ADC12MEM8 到 ADC12MEM15 的采样保持时间，保持时间为 ADC12CLK 的个数。

SHT0x　Bits11～8　采样保持时间选择位。当采用脉冲采样模式时，这些位定义寄存器 ADC12MEM0 到 ADC12MEM7 的采样保持时间，保持时间为 ADC12CLK 的个数。

SHTx　Bits15～8　ADC12CLK cycles。

0000　4；
0001　8；
0010　16；
0011　32；
0100　64；
0101　96；
0110　128；
0111　192；
1000　256；
1001　384；
1010　512；
1011　768；
1100　1024；
1101　1024；
1110　1024；
1111　1024。

MSC　Bit7　多次采样转换控制位，只用于连续或重复模式。



0 采样时序，要求SHI上升沿来触发每次采样转换；

1 第一次SHI的上升沿触发采样时序，当上一次采样转换结束以后，以后的采样自动完成。

REF2_5V Bit6 内部参考电压选择位。

0 1.5 V；

1 2.5 V。

REFON Bit5 参考电压模块控制位。

0 参考电压模块关闭；

1 参考电压模块打开。

ADC12ON Bit4 ADC模块电源控制位。

0 ADC12关；

1 ADC12开。

ADC12OVIE Bit3 ADC12MEMx溢出中断使能。当ADC转换结果没有被读取，而且又完成了一次转换，结果写入ADC12MEMx时，将产生该中断。

0 溢出中断不使能；

1 溢出中断使能。

ENC Bit1 ADC转换使能。

0 ADC12转换器不使能；

1 ADC12转换器使能。

ADC12SC Bit0 ADC12转换开始，软件控制采样转换开始位。ADC12SC和ENC可以用一条指令同时置位。ADC12SC可以自动清零。

0 没有开始采样转换；

1 开始采样转换。

2. ADC12CTL1寄存器

ADC12CTL1寄存器是一个16位的寄存器，该寄存器主要控制ADC12模块的工作。ADC12CTL1寄存器的位分配如图7.9所示。

15 12	11 10	9	8	7 5	4 3	2 1	0
CSTARTADDx	SHSx	SHP	ISSH	ADC12DIVx	ADCSSELx	CONSEQx	ADC12BUSY
rw-(0) rw-(0)	rw-(0) rw-(0)	rw-(0)	rw-(0)	rw-(0) rw-(0)	rw-(0) rw-(0)	rw-(0) rw-(0)	rw-(0)

注：阴影部分只有当ENC=0时才能修改。

图7.9 ADC12CTL1寄存器

下面介绍 ADC12CTL0 寄存器的各个位。

CSTARTADDx	Bits15～12	转换开始地址。这些位用来选择单次转换或者连续多路转换的开始寄存器 ADC12MEMx，它的取值为 0～0x0F。
SHSx	Bits11～10	采样保持触发源选择位。 00　ADC12SC/位； 01　Timer_A 的 OUT1； 10　Timer_B 的 OUT0； 11　Timer_B 的 OUT1。
SHP	Bit9	采样保持脉冲模式选择。该位用来选择采样信号(SAMPCON)是采样时序的输出还是直接输入信号。如果不采用 TA1、TB0、TB1 和 ADC12CLK 来产生采样时钟信号时，SHP 位必须置 1，否则不能进行 A/D 转换。 0　SAMPCON 信号采用采样输入信号； 1　SAMPCON 信号采用采样时序电路产生的信号。
ISSH	Bit8	采样保持信号反转控制位。 0　不反转； 1　反转。
ADC12DIVx	Bits7～5	ADC12 时钟分频控制位。 000　不分频； 001　2 分频； 010　3 分频； 011　4 分频； 100　5 分频； 101　6 分频； 110　7 分频； 111　8 分频。
ADC12SSELx	Bits4～3	ADC12 时钟源选择控制位。 00　ADC12OSC； 01　ACLK； 10　MCLK； 11　SMCLK。

CONSEQx Bits2～1 ADC12 转换模式选择位。

00 单路、单次转换；

01 多路单次转换；

10 单路重复转换；

11 多路重复转换。

ADC12BUSY Bit0 ADC12 忙信号。

0 没有 ADC 正在转换；

1 有 ADC 正在转换。

3. ADC12MEMx 寄存器

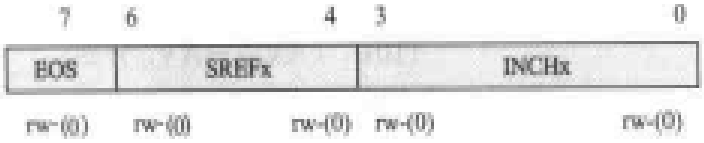
ADC12MEMx 寄存器是一个 16 位的寄存器，该寄存器主要存储转换得到的数据。ADC12MEMx 寄存器的位分配如图 7.10 所示。

图 7.10 ADC12MEMx 寄存器

Conversion Results Bits15～0 ADC12 的转换结果。12 位转换结果是向右对齐，15～12 位通常为 0，写该寄存器将破坏原有的转换结果。

4. ADC12MCTLx 寄存器

ADC12MCTLx 寄存器是一个 8 位的寄存器，该寄存器主要控制各个转换存储器的转换条件。ADC12MCTLx 寄存器的位分配如图 7.11 所示。



注：阴影部分只有当 ENC=0 时才能修改。

图 7.11 ADC12CTL1 寄存器

下面介绍 ADC12MCTLx 寄存器的各个位。

EOS Bit7 连续转换结束标志。注意当 ADC12MCTL15 的 EOS 位没有被置位时，ADC12MEM15 转换结束以后将跳到 ADC12MEM0。

0 不是连续转换结束通道；

1 连续转换结束通道。

SREFx　　Bits6～4　　参考选择控制位。

000　$V_{R-}=AV_{CC}$，且 $V_R=AV_{SS}$；

001　$V_{R+}=V_{REF+}$，且 $V_R=AV_{SS}$；

010　$V_{R+}=V_{eREF+}$，且 $V_R=AV_{SS}$；

011　$V_{R-}=V_{eREF-}$，且 $V_R=AV_{SS}$；

100　$V_{R+}=AV_{CC}$，且 $V_{R-}=V_{REF}/V_{eREF}$；

101　$V_{R+}=V_{REF+}$，且 $V_R=V_{REF-}/V_{eREF-}$；

110　$V_{R+}=V_{eREF-}$，且 $V_R=V_{REF-}/V_{eREF-}$；

111　$V_{R-}=V_{eREF+}$，且 $V_{R-}=V_{REF-}/V_{eREF-}$。

INCHx　　Bits3～0　　输入通道选择。

0000　A0；

0001　A1；

0010　A2；

0011　A3；

0100　A4；

0101　A5；

0110　A5；

0111　A7；

1000　V_{eREF+}；

1001　V_{REF}/V_{eREF-}；

0111　温度传感器；

1011　$(AV_{CC}-AV_{SS})/2$；

1100　$(AV_{CC}-AV_{SS})/2$；

1101　$(AV_{CC}-AV_{SS})/2$；

1110　$(AV_{CC}-AV_{SS})/2$；

1111　$(AV_{CC}-AV_{SS})/2$。

5. ADC12IE 寄存器

ADC12IE 寄存器是一个 16 位的寄存器，该寄存器主要控制各个转换存储器的转换条件。ADC12IE 寄存器的位分配如图 7.12 所示。

ADC12IEx　　Bits15～0　　中断使能位。这些位控制中断标志 ADC12IFGx 的使能或者不使能。

0　　中断不允许；

1　　中断允许。

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

图 7.12　ADC12IE 寄存器

6. ADC12IFG 寄存器

ADC12IFG 寄存器是一个 16 位的寄存器，该寄存器为 ADC12 模块的中断标志寄存器。ADC12IFG 寄存器的位分配如图 7.13 所示。

ADC12IFGx　Bits15~0　ADC12MEMx 中断标志位。当 ADC 转换结果被写入 ADC12MEMx 寄存器时，中断标志被置位；当相应的 ADC12MEMx 被读取或者软件复位时，将清零该位。

15	14	13	12	11	10	9	8
ADC12IFG15	ADC12IFG14	ADC12IFG13	ADC12IFG12	ADC12IFG11	ADC12IFG10	ADC12IFG9	ADC12IFG8
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC12IFG7	ADC12IFG6	ADC12IFG5	ADC12IFG4	ADC12IFG3	ADC12IFG2	ADC12IFG1	ADC12IFG0
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

图 7.13　ADC12IFG 寄存器

7. ADC12IV 寄存器

ADC12IV 寄存器是一个 16 位的寄存器，该寄存器为 ADC12 模块的中断向量寄存器。ADC12IV 寄存器的位分配如图 7.14 所示。

15	6	5	0
0		ADC12IVx	
r0	r0	r-(0)	r-(0)

图 7.14　ADC12IV 寄存器

下面介绍 ADC12IV 寄存器的各个位。

ADC12IVx　Bits15~0　ADC 中断向量表值，该寄存器指示当前中断是哪个中断。

000H　没有中断挂起；

002H　ADC12MEMx 溢出中断；

004H　转换时间溢出中断；
006H　ADC12MEM0 中断；
008H　ADC12MEM1 中断；
00AH　ADC12MEM2 中断；
00CH　ADC12MEM3 中断；
00EH　ADC12MEM4 中断；
010H　ADC12MEM5 中断；
012H　ADC12MEM6 中断；
014H　ADC12MEM7 中断；
016H　ADC12MEM8 中断；
018H　ADC12MEM9 中断；
01AH　ADC12MEM10 中断；
01CH　ADC12MEM11 中断；
01EH　ADC12MEM12 中断；
020H　ADC12MEM13 中断；
022H　ADC12MEM14 中断；
024H　ADC12MEM15 中断。

7.5　ADC12 应用举例

1. 单通道单次转换

本例用 ADC 中的 A0 通道单次采样来详细说明单通道单次操作流程，即只使用一个通道，且只采样一次。实现该功能的程序流程如图 7.15 所示。

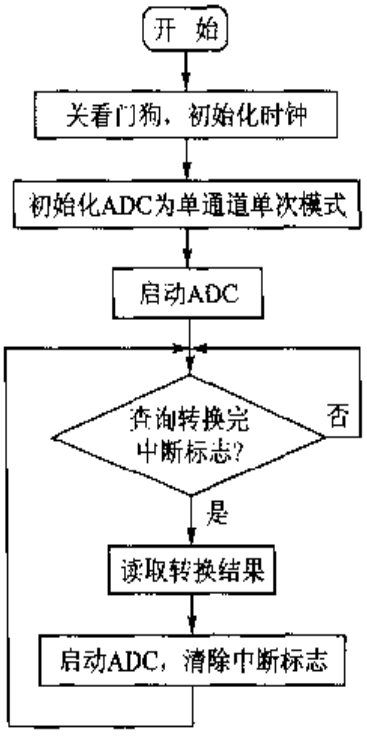
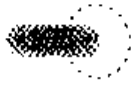


图 7.15　软件流程图

具体程序如下：

```
#include <msp430x14x.h>
#define  uint  unsigned int
#define  uchar unsigned char
static uchar adc_flag = 0;
uint AD_TEMP = 0;
void int_clk()
{
    uchar i;
```

```
    BCSCTL1&=~XT2OFF;                          //打开 XT 振荡器
    BCSCTL2 |= SELM1+SELS;                     //MCLK 8M and SMCLK 1M
    do
    {
        IFG1 &= ~OFIFG;                        //清除振荡错误标志
        for(i = 0; i<100; i++)
        _NOP();                                //延时等待
    }
    while ((IFG1 & OFIFG) != 0);               //如果标志为 1,则继续循环等待
    IFG1&=~OFIFG;
}

int_adc()
{
    P6SEL |= 0x01;                             //选择 AD 通道
    ADC12CTL0 |= ADC12ON + SHT0_2 + REF2_5V + REFON;  //采样保持时间为 16 个 ADC12CLK
    ADC12CTL1 |= ADC12SSEL1 + ADC12SSEL1;      //参考电压开启,选择 2.5 V
    ADC12MCTL0 = 0x10;                         // ref+ =REF2_5V, channel = A0
    ADC12IE = 0x01;                            // 使能转换中断
    ADC12CTL0 |= ENC;                          // 使能 A/D 转换器
}
#pragma vector = ADC_VECTOR
__interrupt void ADC12ISR (void)
{
    while((ADC12CTL1&0X01)==1);                //等待转换完
    adc_flag=1;
    AD_TEMP = ADC12MEM0;                       //设置 A/D 转换完成标志,并读取 ADC 值
}

int main( void )
{
    WDTCTL = WDTPW + WDTHOLD;                  //关闭看门狗
    int_clk();                                 //系统时钟初始化
    int_adc();                                 //初始化串口
    _EINT();                                   //使能中断
    adc_flag=1;
    while(1)
    {
        while(adc_flag==1)
        {
            ADC12CTL0 |= ADC12SC;              // 开启转换
```



```
        ADC12CTL0 &= ~ADC12SC;
        adc_flag = 0;                                    //清中断标志
    }
}
```

2. 单通道多次转换

本例用 ADC 中的 A0 通道多次采样来详细说明单通道多次操作流程，即一个通道采样多次，这样做可以实现在较短时间内进行多次采样取平均值。实现该功能的程序流程如图 7.16 所示。

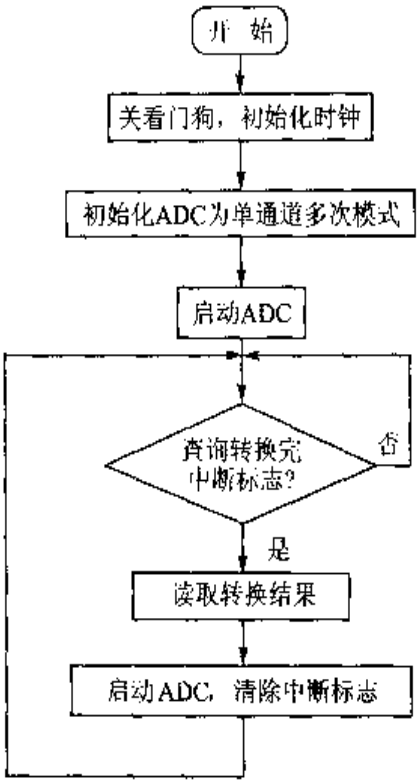


图 7.16 软件流程图

具体程序如下：

```
#include <msp430x14x.h>
#define  uint  unsigned int
#define  uchar unsigned char
uchar static adc_flag = 0;
uchar static count = 0;
void int_clk()
{
    uchar i;
    BCSCTL1&= ~XT2OFF;                                   //打开 XT 振荡器
    BCSCTL2| = SELM1 + SELS;                             //MCLK 为 8MHz,SMCLK 为 1MHz
    do
    {
        IFG1 &= ~OFIFG;                                  //清除振荡错误标志
        for(i = 0; i < 100; i++)
        _NOP();                                          //延时等待
        }
    while ((IFG1 & OFIFG) != 0);                         //如果标志为 1,则继续循环等待
    IFG1&= ~OFIFG;
}
int_adc()
{
    P6SEL  = 0X01;                                       //选择 A/D 通道
    ADC12CTL0 = ADC12ON + MSC + SHT0_2 + REF2_5V + REFON;  //采样保持时间为 16 个 ADC12CLK
                                                         //参考电压开启,选择 2.5 V
    ADC12CTL1 = SHP + CONSEQ_2;                          //多通道单次转换模式
    ADC12MCTL0 = 0X10;                                   // ref + = REF2_5V, channel = A0
```

```
    ADC12IE = 0x01;                                    // 使能转换中断
    ADC12CTL0 |= ENC;                                  // 使能 A/D 转换器
}
#pragma vector = ADC_VECTOR
__interrupt void ADC12ISR (void)
{
    uint results[10];
    adc_flag = 1;
    results[count] = ADC12MEM0;                        // 读取 A0 转换结果
    count ++ ;
}

int main( void )
{
    WDTCTL = WDTPW + WDTHOLD;                          //关闭看门狗
    int_clk();                                         //系统时钟初始化
    int_adc();                                         //初始化串口
    _EINT();                                           //使能中断
    ADC12CTL0 |= ADC12SC;                              // 开启转换
    while(1);
}
```

3. 多通道单次转换

本例用 ADC 中的 A0～A7 通道单次采样来详细说明多通道单次操作流程，即使用 P8 口 8 条口线实现对 8 个通道的模拟信号进行一次转换。实现该功能的程序流程如图 7.17 所示。

具体程序如下：

```
#include <msp430x14x.h>
#define  uint  unsigned int
#define  uchar unsigned char
uchar static adc_flag = 0;
void int_clk()
{
    uchar i;
    BCSCTL1&= ~XT2OFF;              //打开 XT 振荡器
    BCSCTL2|= SELM1 + SELS;         //MCLK 为 8 MHz,SMCLK 为 1 MHz
    do
    {
```

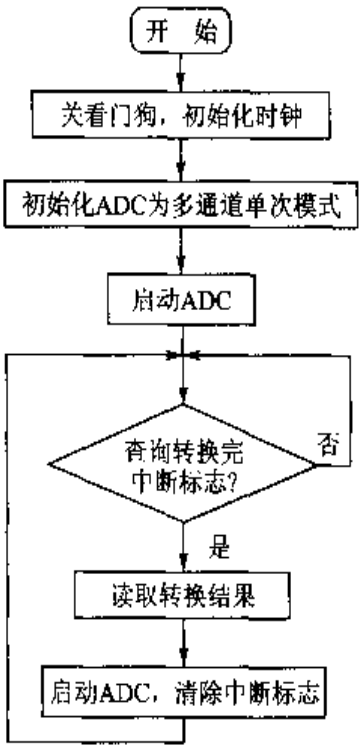
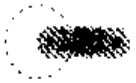


图 7.17　软件流程图

```
        IFG1 &= ~OFIFG;                         // 清除振荡错误标志
        for(i = 0; i < 100; i++)
        _NOP();                                 // 延时等待
        }
    while ((IFG1 & OFIFG) != 0);                // 如果标志为 1,则继续循环等待
    IFG1&= ~OFIFG;
}
int_adc()
{
    P6SEL |= 0x0F;                              // 选择 A/D 通道
    ADC12CTL0 = ADC12ON + MSC + SHT0_2 + REF2_5V + REFON;     // 采样保持时间为 16 个 ADC12CLK
                                                // 参考电压开启,选择 2.5 V
    ADC12CTL1 = SHP + CONSEQ_1;                 // SAMPCON 信号采用时序电路产生多通道单次转换模式
    ADC12MCTL0 = 0x10;                          // ref + = REF2_5V, channel = A0
    ADC12MCTL1 = 0x11;                          // ref + = REF2_5V, channel = A1
    ADC12MCTL2 = 0x12;                          // ref + = REF2_5V, channel = A2
    ADC12MCTL3 = 0x93;                          // ref + = REF2_5V, channel = A3, 结束转换通道选择
    ADC12IE = 0x08;                             // 使能转换中断
    ADC12CTL0 |= ENC;                           // 使能 A/D 转换器
}
#pragma vector = ADC_VECTOR
__interrupt void ADC12ISR (void)
{
    uint results[4];
    results[0] = ADC12MEM0;                     // 读取 A0 转换结果
    results[1] = ADC12MEM1;                     // 读取 A1 转换结果
    results[2] = ADC12MEM2;                     // 读取 A2 转换结果
    results[3] = ADC12MEM3;                     // 读取 A3 转换结果
    adc_flag = 1;                               // 下次转换标志写 1
}
int main( void )
{
    WDTCTL = WDTPW + WDTHOLD;                   // 关闭看门狗
    int_clk();                                  // 系统时钟初始化
    int_adc();                                  // 初始化串口
    _EINT();                                    // 使能中断
    adc_flag = 1;
    while(1)
    {
```



```
        while(adc_flag = = 1)
        {
            ADC12CTL0 | = ADC12SC;          // 开启转换
            adc_flag = 0;                   // 清中断标志
        }
    }
}
```

4. 多通道多次转换

本例用 ADC 中的 A0～A3 通道多次采样来详细说明多通道多次的操作流程，即对多个通道进行多次转换，可以实现对多个通道取平均值，且速度比较快。实现该功能的程序流程如图 7.18 所示。

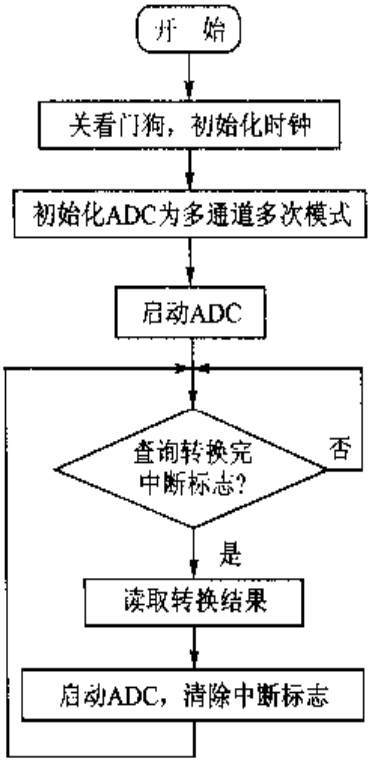


图 7.18 软件流程图

具体程序如下：

```
#include <msp430x14x.h>
#define uint unsigned int
#define uchar unsigned char

uint results0;          //定义 A0 转换结果数组
uint results1;          //定义 A1 转换结果数组
uint results2;          //定义 A2 转换结果数组
uint results3;          //定义 A3 转换结果数组
void int_clk()
{
    uchar i;
    BCSCTL1& = ～XT2OFF;          //打开 XT 振荡器
    BCSCTL2| = SELM1 + SELS;      //MCLK 为 8 MHz,SMCLK 为 1 MHz
    do
    {
        IFG1 & = ～OFIFG;                  //清除振荡错误标志
        for(i = 0; i < 100; i++)
        _NOP();                            //延时等待
        }
    while ((IFG1 & OFIFG) != 0);           //如果标志为 1,则继续循环等待
    IFG1& = ～OFIFG;
}
void int_adc()
{
    P6SEL | = 0x0F;                        //选择 A/D 通道
```

```
    ADC12CTL0 = ADC12ON + MSC + SHT0_2 + REF2_5V;      //采样保持时间为 16 个 ADC12CLK
                                                       //参考电压选择 2.5 V
    ADC12CTL1 = SHP + CONSEQ_3;                  //SAMPCON 信号采用时序电路产生多通道多次转换模式
    ADC12MCTL0 = INCH_0;                         // ref + = AVcc, channel = A0
    ADC12MCTL1 = INCH_1;                         // ref + = AVcc, channel = A1
    ADC12MCTL2 = INCH_2;                         // ref + = AVcc, channel = A2
    ADC12MCTL3 = INCH_3 + EOS;                   // ref + = AVcc, channel = A3, 结束转换通道选择
    ADC12IE = 0x08;                              // 使能转换中断
    ADC12CTL0 = ENC;                             // 使能 A/D 转换器
}

#pragma vector = ADC_VECTOR
__interrupt void ADC12ISR (void)
{

    results0 = ADC12MEM0;                        // 读 A0 转换值
    results1 = ADC12MEM1;                        // 读 A0 转换值
    results2 = ADC12MEM2;                        // 读 A0 转换值
    results3 = ADC12MEM3;                        // 读 A0 转换值
}
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;                    // 停止看门狗
    int_clk();
    int_adc();
    _EINT();
    ADC12CTL0 = ADC12SC;                         // 开启转换
    while(1);
}
```

通过本章对 MSP430F149 的 ADC12 的结构、寄存器、操作流程和注意事项的详细介绍，还有一些具体例子的学习之后，相信大家已经能够按照系统需要对 MSP430F149 的 A/D 模块进行设计以及应用了。

注意：由于配套实验板只留出 2 路 ADC 给用户，故以上 ADC 举例中除通道 0 和通道 1 可以调节输入电压观察采样结果外，其他通道得到的值均不确定。

实　践　篇

- 第 8 章　基于 PC 的 RS485 多机通信
- 第 9 章　Flash 存储器 M25P80 的应用
- 第 10 章　基于 FYD12864LCD 的数字电压表的设计
- 第 11 章　基于 PCF8563 的时钟设计
- 第 12 章　简易 USB 接口设计
- 第 13 章　任意信号发生器的设计

第8章

基于 PC 的 RS485 多机通信

8.1 RS485 简介

智能仪表是随着 20 世纪 80 年代初单片机技术的成熟而发展起来的。目前的世界仪表市场基本被智能仪表所垄断。究其原因，主要是企业信息化的需要。企业在仪表选型时，其中的一个必要条件就是要具有联网通信接口。采用 RS232 接口的仪表可以实现点对点的通信方式，但这种方式不能实现联网功能。随后出现的 RS485 解决了这个问题，下面我们简单介绍一下。

1. RS485 通信的基本特点

RS485 采用差分信号负逻辑，+2～+6 V 表示“0”，−6～−2 V 表示“1”。RS485 有 2 线制和 4 线制两种接线，4 线制只能实现点对点的通信方式，现很少采用。现在多采用的是 2 线制接线方式，这种接线方式为总线式拓朴结构，在同一总线上最多可以挂接 32 个结点。

在 RS485 通信网络中一般采用的是主从通信方式，即一个主机带多个从机。很多情况下，连接 RS485 通信链路时只是简单地用一对双绞线将各个接口的“A”、“B”端连接起来，而忽略了信号地的连接。这种连接方法在许多场合是能正常工作的，但却埋下了很大的隐患。这有以下两个原因。

① 共模干扰问题：RS485 接口采用差分方式传输信号，并不需要相对于某个参照点来检测信号，系统只须检测两线之间的电位差就可以了。但人们往往忽视了收发器有一定的共模电压范围，RS485 收发器的共模电压范围为−7～+12 V。只有满足上述条件，整个网络才能正常工作。当网络线路中共模电压超出此范围时，就会影响通信的稳定可靠，甚至损坏接口。

② EMI 问题：发送驱动器输出信号中的共模部分需要一个返回通路。如果没有一个低阻的返回通道(信号地)，就会以辐射的形式返回源端，整个总线就会像一个巨大的天线向外辐射电磁波。

由于 PC 机默认情况下只带有 RS232 接口，有两种方法可以得到 PC 上位机的 RS485 电路：① 通过 RS232/RS485 转换电路将 PC 机串口 RS232 信号转换成 RS485 信号，对于情况比较复杂的工业环境最好是选用防浪涌带隔离的产品。② 通过 PCI 多串口卡，可以直接选用输出信号为 RS485 类型的扩展卡。

2. RS485 电缆

一般场合下，采用普通的双绞线就可以了。在要求比较高的环境下，可以采用带屏蔽层的同轴电缆。在使用 RS485 接口时，对于特定的传输线路，从 RS485 接口到负载，其数据信号传输所允许的最大电缆长度与信号传输的波特率成反比，这个长度主要受信号失真及噪声等影响。理论上，RS485 的最长传输距离能达到1 200 m，但在实际应用中传输的距离要比1 200 m 短，具体能传输多远视周围环境而定。在传输过程中，可以采用增加中继的方法对信号进行放大，最多可以加 8 个中继。也就是说，理论上 RS485 的最大传输距离可以达到9.6 km。如果需要进行长距离传输，可以采用光纤为传播介质，在收、发端各加一个光电转换器。多模光纤的传输距离是 5～10 km，而采用单模光纤可达 50 km。

3. RS485 布网

网络拓扑一般采用终端匹配的总线型结构，不支持环形或星形网络。在构建网络时，应注意如下几点：

- 采用一条双绞线电缆作总线，将各个节点串接起来。从总线到每个节点的引出线长度应尽量短，以便使引出线中的反射信号对总线信号的影响最低。有些网络连接尽管不正确，但在短距离、低速率情况下仍可能正常工作，随着通信距离的延长或通信速率的提高，其不良影响会越来越严重。主要原因是信号在各支路末端反射后与原信号叠加，会造成信号质量的下降。
- 应注意总线特性阻抗的连续性，在阻抗不连续点就会发生信号的反射。下列几种情况容易产生这种不连续性：总线的不同区段采用了不同的电缆，或某一段总线上有过多收发器紧靠在一起安装，或由过长的分支线引出到总线。
- 在 RS485 组网过程中，另一个需要注意的问题是终端负载电阻问题。在设备少、距离短的情况下，不加终端负载电阻，整个网络能很好地工作；但是随着距离的增加，性能将降低。理论上，在每个接收数据信号的终点进行采样时，只要反射信号在开始采样时衰减到足够低，就可以不考虑匹配。但这一点在实际应用中难以掌握。美国 MAXIM 公司有篇文章在谈到如何判断在什么样的数据速率和电缆长度时需要进行匹配时，曾经提到一条经验性的原则：当信号的转换时间(上升或下降时间)，超过电信号沿总线单向

传输所需时间的 3 倍时就可以不加匹配。

一般终端匹配采用终端电阻方法，RS485 应在总线电缆的开始和末端都并接终端电阻。终端电阻在 RS485 网络中取 120 Ω，相当于电缆特性阻抗的电阻。因为大多数双绞线电缆的特性阻抗为 100～120 Ω。这种匹配方法简单有效，但有一个缺点，匹配电阻要消耗较大功率，对于功耗限制比较严格的系统不太适合。另外一种比较省电的匹配方式是 RC 匹配。利用一只电容 C 隔断直流成分可以节省大部分功率。但电容 C 的取值是个难点，需要在功耗和匹配质量间进行折衷。还有一种采用二极管的匹配方法，这种方案虽未实现真正的“匹配”，但它利用二极管的钳位作用能迅速削弱反射信号，达到改善信号质量的目的，节能效果显著。

4. RS485 组网的优点

RS485/MODBUS 是现在流行的一种布网方式，其特点是实施简单方便。而且现在支持 RS485 的仪表又非常多。特别是在油品行业，RS485/MODBUS 可以说是一统天下。现在的仪表商也纷纷转而支持 RS485/MODBUS，原因很简单。比如说，原来的 HART 仪表想买一个转换口非常困难，且价格高昂，而 RS485 的转换接口就便宜得多而且种类繁多。至少在低端市场，RS485/MODBUS 还将是最主要的组网方式，近几年内不会改变。

8.2　多机通信的应用举例

多机通信的基本原理在第 5 章中已经详细介绍了，下面主要介绍多机通信的应用。

多机通信主要应用在工业控制和智能检测中。在这些实际应用中，很多情况需要通过一个主机控制多个从机，通过从机对被控设备进行控制和状态检测，同时返回数据给主机。多机通信的基本结构如图 8.1 所示。

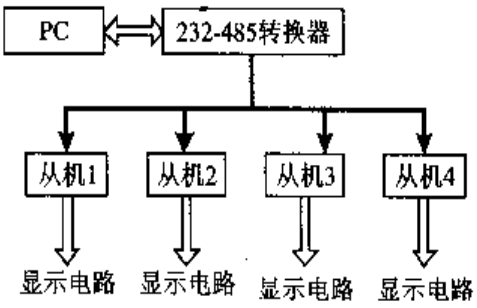


图 8.1　多机通信的基本结构图

下面通过一个具体例子说明 PC 机的程序设计。该实例是采用 RS485 通信，PC 机通过 RS232 口经过 RS485 转换器和从机连接，PC 端的程序采用 VC++6.0 来设计，其基本流程如下：

① 初始化串口，选择数据格式为 9 位方式；

② 读取选择状态（是发送地址还是发送数据）；

③ 接收到数据后，在对话框中显示出来。

下面给出了几个主要的程序。

1. 初始化程序

```
if(m_ctrlComm.GetPortOpen())
m_ctrlComm.SetPortOpen(FALSE);
m_ctrlComm.SetCommPort(1);                    //选择 com1
m_ctrlComm.SetInBufferSize(1024);
m_ctrlComm.SetOutBufferSize(512);
if( ! m_ctrlComm.GetPortOpen())
    m_ctrlComm.SetPortOpen(TRUE);             //打开串口
else
    AfxMessageBox("cannot open serial port");
m_ctrlComm.SetSettings("19200,S,8,1"); //波特率 9600,无校验,8 个数据位,1 个停止位
m_ctrlComm.SetParityReplace("");
m_ctrlComm.SetInputMode(1);                   //1:表示以二进制方式检取数据
m_ctrlComm.SetRThreshold(1);
//参数 1 表示每当串口接收缓冲区中有多于或等于 1 个字符时将引发一个接收数据的 OnComm
//事件
m_ctrlComm.SetInputLen(0);                    //设置当前接收区数据长度为 0
m_ctrlComm.GetInput();                        //先预读缓冲区以清除残留数据
```

发送程序

```
void CRs232Dlg::Onsent()
{
    UpdateData(TRUE);
    char flag = 0;
//int temp1 = 0;
    char temp = 0;
//temp1 = m_bb;
    if(m_select == 1){
        if(flag == 0)
        {
            m_ctrlComm.SetSettings("19200,M,8,1");
            temp = m_add;
            CByteArray array;
            array.RemoveAll();
            array.SetSize(1);
            array.SetAt(0, temp);
            m_ctrlComm.SetOutput(COleVariant(array));          // 发送地址
            flag = 1;
            m_select = 0;
```

```
            UpdateData(FALSE);
            return ;
        }
    }                                          //波特率9600,无校验,8个数据位,1个停止位
    UpdateData(FALSE);
    UpdateData(TRUE);
    char TxData[100];
    int Count = m_dl.GetLength();
    for(int i = 0; i < Count; i++)
        TxData[i] = m_dl.GetAt(i);
    CByteArray array;
    array.RemoveAll();
    array.SetSize(Count);
    for(i=0;i<Count;i++)
        array.SetAt(i, TxData[i]);
    m_ctrlComm.SetSettings("19200,S,8,1");
    m_ctrlComm.SetOutput(COleVariant(array));        // 发送数据
}
```

2. 运行效果图

运行效果见图8.2。

关于VC++的程序设计，读者可以参考相关书籍，这里不再详细说明。

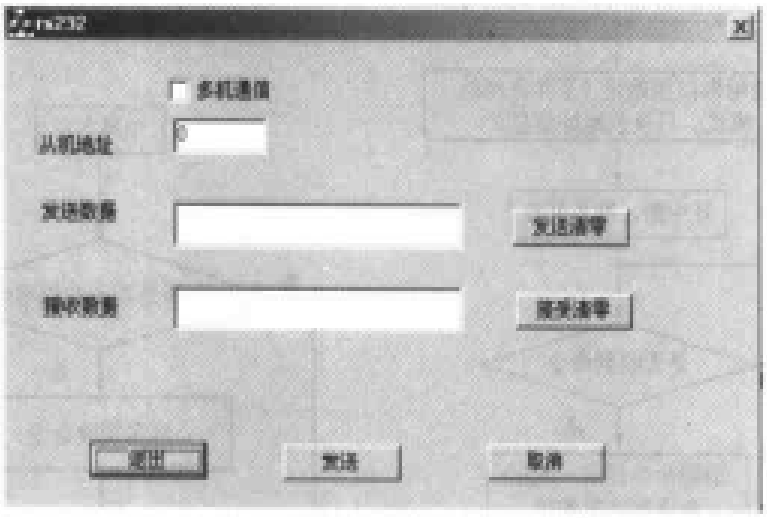


图8.2　运行效果

3. 系统主要完成的功能

本实例主要实现通过PC机控制多台下位机I/O输出的不同状态来解释不同的命令。本

例中控制 4 个下位机，当下位机接收到主机的地址信息时，下位机返回本机地址，表示接收命令成功，同时根据不同命令在 LED 上显示自己本身的地址和命令编码。读者可以通过本书配套实验板完成该设计，也可以根据实际要求自行设计。

本例的从机地址的编码是通过软件来设置的，分别为 0x01、0x02、0x03、0x04。在实际应用中可以采用拨码开关来选择地址，便于操作和统一程序。

另外，在实际应用中，从机通过解释主机命令，执行相应的操作，并且返回数据和状态信息。这里主要说明多机通信的基本原理和简单应用，故命令编码分别是 1、2、3、4。从机接收到命令后就显示出该命令编码。

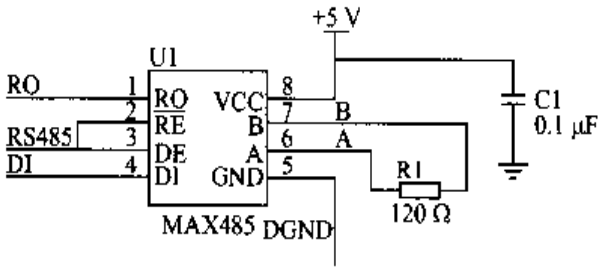
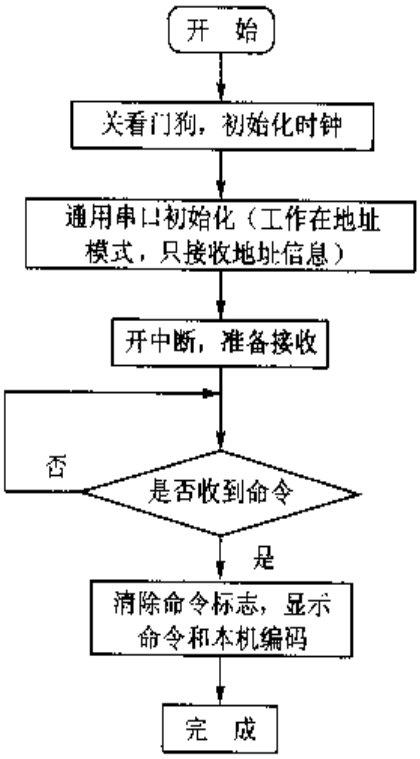


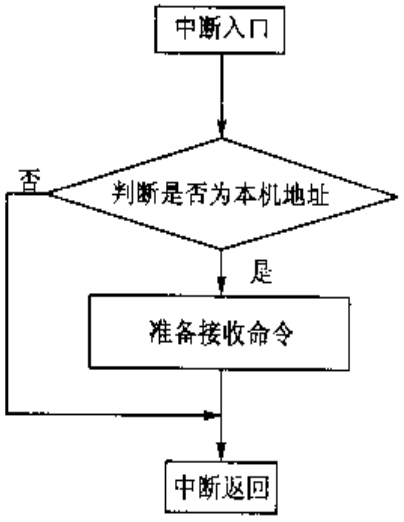
图 8.3　硬件电路设计

多机通信方式下，每个计算机有一个从机地址，PC 机通过地址信息对下位机进行寻址。该方式的优点是软件设计简单，扩展容易。

通信过程全部通过软件来判断，这样在通信数据量要求不太高并且 CPU 任务不太重的情况下可以采用本方法。由于本实例满足这样的要求，所以我们在设计中采用这种方式。相应的硬件电路和软件流程分别如图 8.3 和图 8.4 所示。



(a) 系统主程序流程图



(b) 接收中断流程图

图 8.4　流程图

具体的程序设计如下：

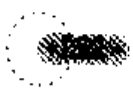
```
#include "msp430x14x.h"
#define  uint  unsigned int
#define  uchar  unsigned char
#define  RS485  BIT0
#define  LED1_ON  P6DIR|= BIT7,P6OUT&= ~BIT7
#define  LED1_OFF  P6DIR|= BIT7,P6OUT|= BIT7
#define led   BIT7                    //定义 LED 控制位
const uchar segment[11]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0xff};
uchar static add=0x5;
uchar static add_flag=0;
uchar static com_flag=0;
uchar com=0;
void int_clk()
{
    uchar i;
    BCSCTL1&= ~XT2OFF;              //打开 XT 振荡器
    BCSCTL2 |= SELM1+SELS;          //MCLK 为 8 MHz,SMCLK 为 1 MHz
    do
    {
        IFG1 &= ~OFIFG;             //清除振荡错误标志
        for(i = 0; i < 100; i++)
        _NOP();                     //延时等待
        }
    while ((IFG1 & OFIFG) != 0);    //如果标志为 1,则继续循环等待
    IFG1&= ~OFIFG;
}

/*************************************************************
程序名称:SPI 初始化子程序
参数:无
功能描述:
*************************************************************/
void int_spi()
{
    P3SEL|= 0x0E;
    U0CTL|= SWRST;
    U0CTL |= CHAR+SYNC+MM;
    U0TCTL|= SSEL1+SSEL0+STC;       //3 线工作模式,SMCLK 作为 SPI 时钟
    U0TCTL|= CKPH;
```

```
    U0TCTL&= ~CKPL;                    //CKPL CKPH:01 上升沿输出数据
    U0BR0 = 0X02;
    U0BR1 = 0X00;
    U0MCTL = 0X00;
    ME1 = USPIE0;                      //SPI 模块使能
    U0CTL&= ~SWRST;
    IE1&= ~UTXIE0;
    IE1&= ~URXIE0;
    P3DIR|= BIT2 + BIT4;               //设置输入/输出方向
}
write_byte(uchar byte)
{
    U0TXBUF = byte;
    while((IFG1&UTXIFG0) == 0);        //UTXIFG0 = 0,表示数据从发送
    IFG1&= ~UTXIFG0;
}
display(uchar data)
{
P2DIR|= led;
P2OUT|= led;                           //使能 164CLK
write_byte(segment[data]);             //写数据
P2OUT&= ~led;                          //禁止 164CLK
}
int_usart()
{
    U1CTL|= SWRST;
    U1CTL|= CHAR + MM;
    U1TCTL|= SSEL1;                    //串口工作时钟为 SMCLK
    U1RCTL|= URXWIE;
    U1BR0 = 0Xa0;
    U1BR1 = 0X01;
    UMCTL1 = 0Xc0;                     //波特率 19200
    ME2 = UTXE1 + URXE1;               //使能接收和发送
    U1CTL&= ~SWRST;
    IE2 = URXIE1;                      //使能接收中断
    P3SEL = BIT6 + BIT7;
    P3DIR|= BIT6;                      //设置 I/O 功能和方向
}
sent_byte(uchar data)                  //发送一个字节数据
```

```
{
    while((IFG2&UTXIFG1) == 0);
    U1TXBUF = data;
}
delay()
{
    uchar i;
    for(i = 0;i <= 200;i ++ )
        ;
}
# pragma vector = UART1RX_VECTOR
__interrupt void UART0_RX_ISR(void)
{
    uchar data;
    data = U1RXBUF;
if(data == add)                    //判断是否本机地址,若是则清除地址位准备接收数据,
                                   //并且地址标志写1;若不是则不执行
    {
        U1RCTL& = ~URXWIE;
        add_flag = 1;
        return;
    }
if(add_flag == 1)                  //判断是否已经接收到一个有效地址信息,是则返回本机地址
                                   //和接收到的命令数据
    {
        P2OUT = RS485;
        sent_byte(add + 0x30);
        sent_byte(data);
        com = data - 0x30;
        delay();
        P2OUT& = ~RS485;
        U1RCTL| = URXWIE;
        com_flag = 1;              //命令标志写1
        add_flag = 0;
    }
}
int main( void )
{
    WDTCTL = WDTPW + WDTHOLD;
```

```
    int_clk();
    int_usart();
    int_spi();                          //系统初始化
    P2DIR = RS485;
    P2OUT& = ~RS485;
    _EINT();
    while(1)
    {
    if(com_flag == 1)                   //判断命令标志是否为1,为1按照相关命令处理
    {
        if(com == 5) {LED1_ON;display(10);display(10);display(add);display(com);}
        if(com == 6) {LED1_OFF;display(10);display(10);display(add);display(com);}
        com_flag = 0;
    }
    }
    while(1);
}
```

通过上面的举例，希望读者能够掌握 RS485 通信的基本特点和主要使用方法，同时对多机通信有比较深刻的理解。



第9章

Flash 存储器 M25P80 的应用

9.1 概 述

很多系统中都需要在本地保存大量的数据和一些常数，然后进行本地处理。这就需要一个存储介质。在实际的应用中，许多数据采集系统采集的数据量都比较大，因此对系统的存储容量有很高的要求，一般普通的 EPROM 或者 Flash 的容量都难以满足要求。而大容量 Flash M25PXX 系列就能够很好地满足用户的需求，用户可以根据实际需要进行选择。

本书所采用的 M25P80 主要有如下特点：

- 是意法半导体公司推出的 8M 大容量串行接口 Nand Flash 器件；
- 采用 2.7～3.6 V 单电源供电，深度休眠仅需 1 μA 的电流；
- 标准的 SPI 接口，器件在上升沿接收数据，在下降沿发送数据，接口时钟最高为 25 MHz；
- M25P80 共有 16 个扇区，每个扇区有 256 个页，每个页有 256 字节；
- 支持最大 256 字节的快速页面编程操作(典型时间 1.5 ms)、快速的块(512 Kb)擦除操作(典型时间 2 s)和快速的整体(8 Mb)擦除操作(典型时间 8 s)；
- 每扇区擦除/编程周期超过 100000 次，超过 20 年的存储时间，并且具有操作暂停和硬件写保护功能。

因此在各种对成本有较高要求的计算机及消费电子产品，如打印机、光驱、无线局域网(WLAN)模块以及机顶盒(STB)中具有广泛的应用。

9.2 M25P80 与 MSP430F149 的硬件连接

M25P80 是 8 引脚的大容量 Flash，它的引脚分布如图 9.1 所示。相应引脚的功能介绍如

表 9.1 所列。

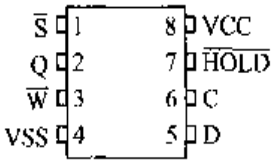


图 9.1 M25P80 的引脚图

表 9.1 M25P80 的引脚功能介绍

引脚号	引脚名称	引脚功能
1	S	片选信号,低电平有效
2	Q	串行数据输出
3	W	写保护信号,低电平有效
4	VSS	电源地
5	D	串行数据输入
6	C	串行时钟
7	HOLD	暂停信号,低电平有效
8	VCC	正电源

M25P80 与 MSP430F149 单片机的硬件连接非常简单,主要是片选信号。串行数据输入、输出以及时钟信号与 F149 的连接,如图 9.2 所示。

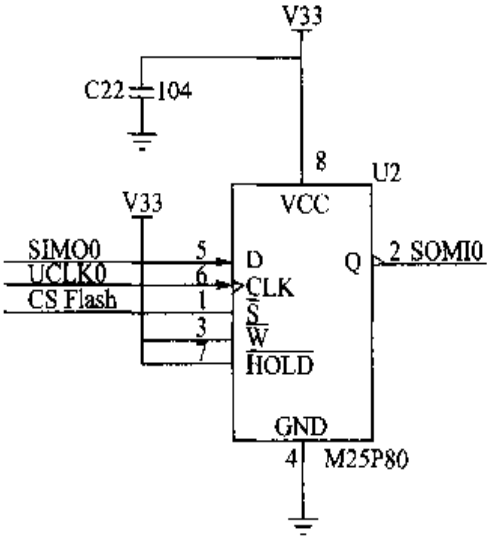


图 9.2 M25P80 与 MSP430F149 的硬件连接图

9.3 M25P80 的指令操作

M25P80 支持的操作指令共有 12 条,所有指令都是 8 位。操作步骤如下:先选中器件(片选信号拉低),然后输入 8 位操作指令字节,部分指令需要输入地址字节(0~3 字节),再把片选信号拉高(有些指令不要求),M25P80 即可以启动内部控制逻辑,自行完成相应的操作。下次操作时需要判断是否处于忙状态:如果忙则不能操作需要等内部操作完成后才可以进行下

一次操作；如果不忙就可以进行下一步操作。具体读者可以参考本书提供的程序。

M25P80 的状态寄存器是一个 8 位的寄存器，其格式如图 9.3 所示。

7	6	5	4	3	2	1	0
SWRD	0	0	BP2	BP1	BP0	WEL	WIP

图 9.3　状态寄存器的格式

下面将对状态寄存器的各个位进行详细介绍。

SWRD　Bit7　写保护状态位。该位为 1 和外部引脚 $\overline{W}$ 为低电平时，具有写保护功能。

BP2，BP1，BP0　Bits4～2　选择块写保护位。不同的位决定了写保护的区域，具体如表 9.2 所列。

WEL　Bit1　写使能锁存位。当等于 0 时，内部写使能锁存复位，这将使所有的写状态寄存器、编程、擦除将会无效。当等于 1 时可以对状态寄存器进行编程、擦除等操作。

WIP　Bit0　忙状态标志位。为 1 时表示处于忙状态。

指令详细说明如表 9.3 所列。

表 9.2　保护区域

状态寄存器内容			存储器内容	
BP2	BP1	BP0	保护区域	未保护区域
0	0	0	无	所有扇区
0	0	1	第 16 扇区	第 1～15 扇区
0	1	0	第 15、16 扇区	第 1～14 扇区
0	1	1	第 13～16 扇区	第 1～12 扇区
1	0	0	第 9～16 扇区	第 1～8 扇区
1	0	1	所有扇区	无
1	1	0	所有扇区	无
1	1	1	所有扇区	无

表 9.3　指令详细说明

指　令	功　能	指令代码	地址字节	读字节	数据字节
WREN	写使能	0000 0110	0	0	0
WRDI	写无效	0000 0100	0	0	0
RDSR	读状态寄存器	0000 0101	0	0	1～∞

续表 9.3

指 令	功 能	指令代码	地址字节	读字节	数据字节
WRSR	写状态寄存器	0000 0001	0	0	1
READ	读数据字节	0000 0011	3	0	1～∞
FAST_READ	高速读数据字节	0000 1011	3	1	1～∞
PP	页面编程	0000 0010	3	0	1～256
SE	扇区擦除	1101 1000	3	0	0
BE	块擦除	1100 0111	0	0	0
DP	深度休眠	1011 1001	0	0	0
RES	从深度休眠唤醒，并读电子签名	1010 1011	0	3	1～∞
	从深度休眠唤醒		0	0	0

下面将以写使能、唤醒、读状态寄存器、读数据字节、页写入、页读出和扇区擦除为例来分别说明各指令的具体用法。

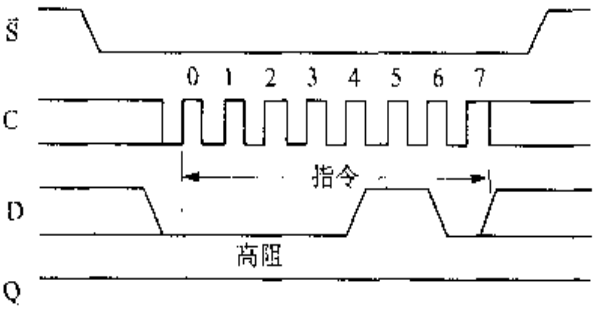


图 9.4 写使能时序图

1. 写使能(WREN)

写使能时序如图 9.4 所示。

写使能是用在对 WEL 位的置位，因为只有当 WEL = 1(外加引脚 $\overline{W}$ 为高电平)时，才能写状态寄存器、编程和擦除。操作步骤如下：先使片选信号有效，即拉低片选信号 $\overline{S}$；然后发命令 06H；再读状态寄存器，判忙，如果不忙则可以执行下面的指令。

2. 唤醒(RES)

唤醒时序如图 9.5 所示。

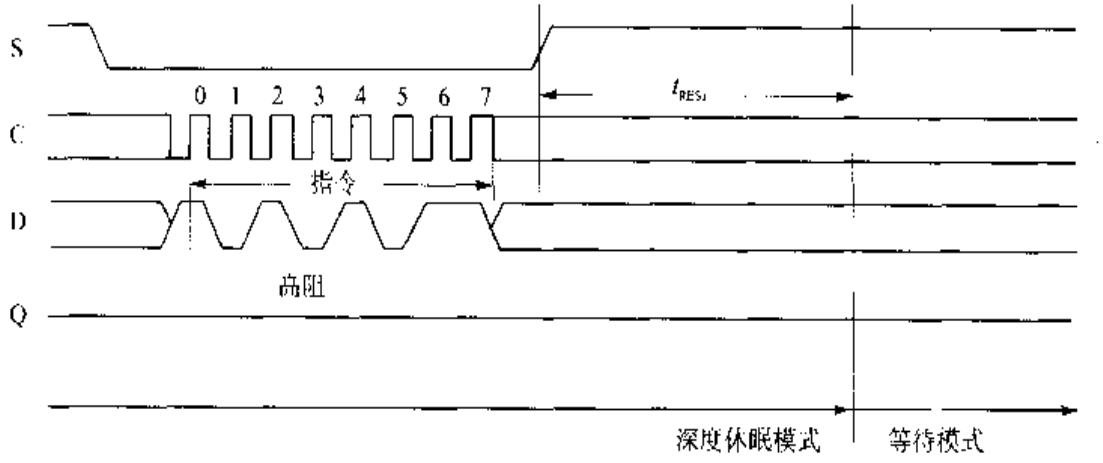


图 9.5 唤醒时序图

一旦该器件进入了休眠，就必须要用该指令对之进行唤醒；否则，以后的指令将无效。具体的操作步骤如下：先拉低片选信号 $\overline{S}$；再发指令 0ABH，它将在上升沿接收到数据；然后读状态寄存器，判忙，要等到不忙后才能执行以后的指令。

3. 读状态寄存器(RDSR)

读状态寄存器的时序如图 9.6 所示。

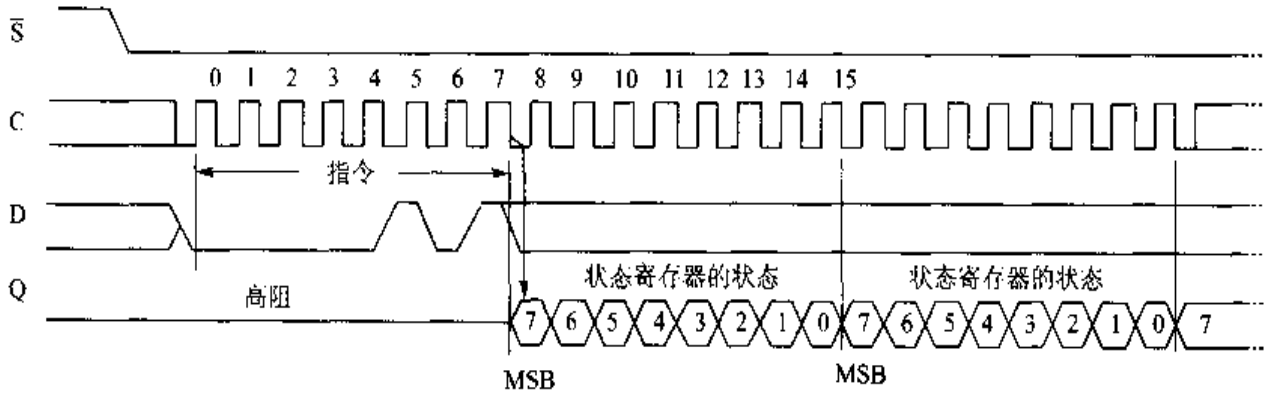


图 9.6　读状态寄存器的时序图

读状态寄存器这一操作可能在任何时候发生，包括写状态寄存器、编程和擦除，因为在实际的应用中都要求先判忙，即判断状态寄存器的最低位 WIP 位(WIP=1 表示处于忙状态，需要等待直到该位变为 0)，再进行相应的操作。具体的步骤如下：先拉低片选信号 $\overline{S}$；再由 SPI 发出命令 05H，在下 8 个时钟周期将会收到状态寄存器的状态，读回寄存器状态即可。

4. 扇区擦除(SE)

扇区擦除的时序如图 9.7 所示。

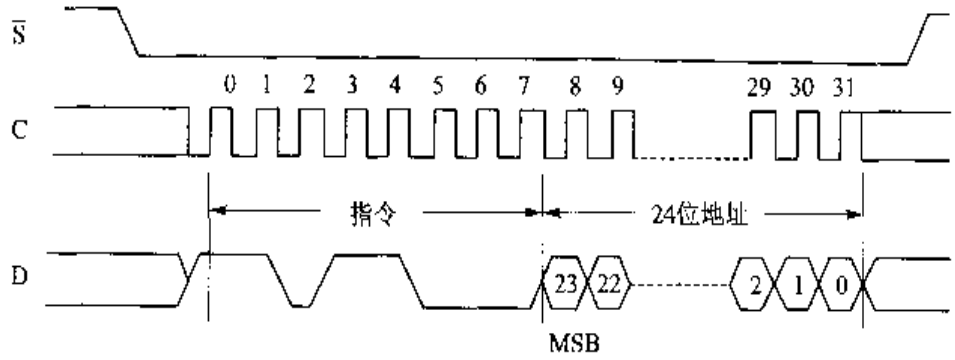


图 9.7　扇区擦除的时序图

本条指令可以对 M25P80 的任意扇区进行擦除，具体的操作步骤如下：先拉低片选信号 $\overline{S}$；再发送命令 0D8H；然后在下面的 24 个时钟周期内发送地址；最后使 $\overline{S}=1$，读状态寄存器的状态，不忙以后就可以执行下面的指令。

5. 多字节读出(READ)

多字节读出的时序如图 9.8 所示。

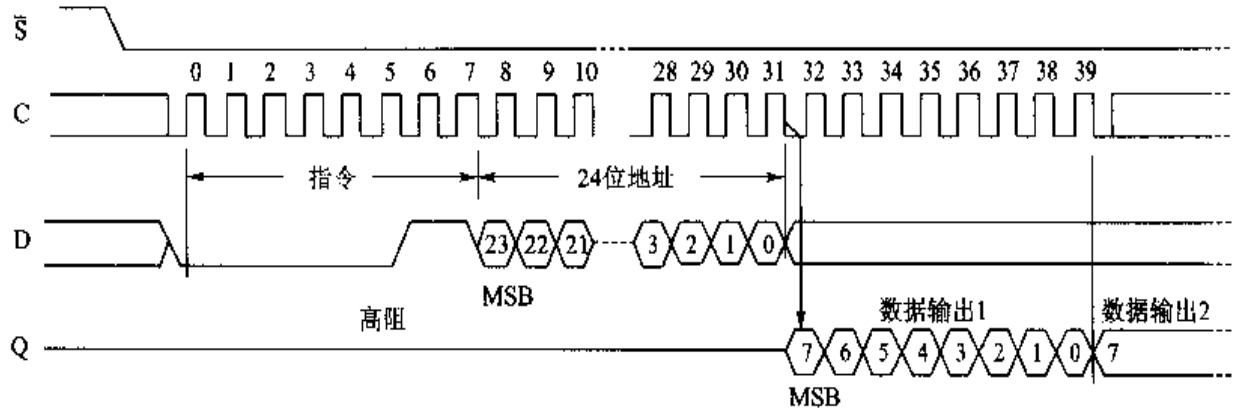


图 9.8 多字节读出的时序图

该条指令可以用于读出多字节的数据，具体的操作步骤如下：

先拉低片选信号 $\overline{S}$；再写入命令 03H；接着的 24 个时钟周期写入 3 字节的地址(A23～A20 是不关心的)，在以后的时钟周期将会输出读回的数据，地址是会自动增加的；最后使 $\overline{S}$=1，并且判忙，不忙则完成字节的读出，并执行下面的指令。

6. 页写入(PP)

页写入的时序如图 9.9 所示。

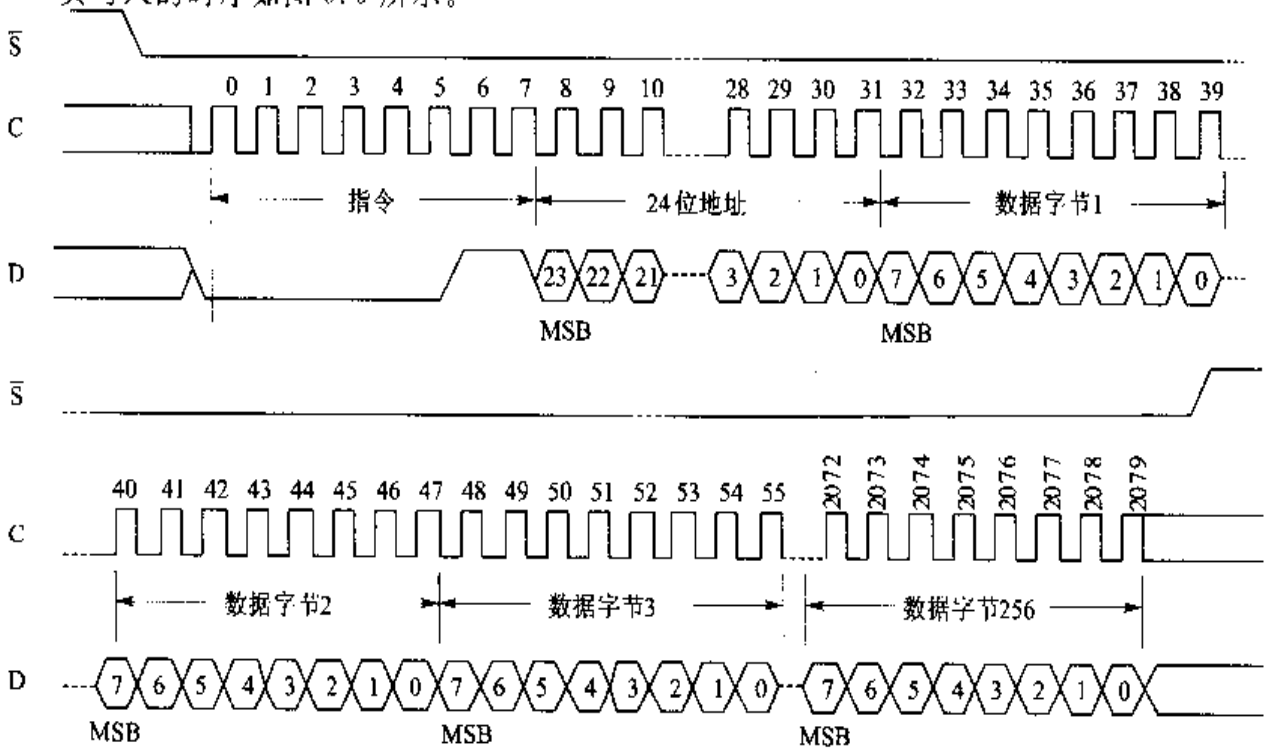


图 9.9 页写入的时序图

页写入指令一次可以写入 1～256 字节的数据，具体操作步骤如下：先拉低片选信号 $\overline{S}$；再送入命令 02H；然后在 24 个时钟周期送入 24 位地址（A23～A20 是不关心的）；再送入要写的数据，最多为 256 字节，最后拉高 $\overline{S}$，并且判忙，不忙就可以执行下面的指令。

9.4　设计举例

该实例的程序流程图如图 9.10 所示，主要实现向 Flash 写一组数据然后读出来，设计中包括了 M25P80 操作的基本子程序，读者可以根据实际要求组合成自己需要的程序。

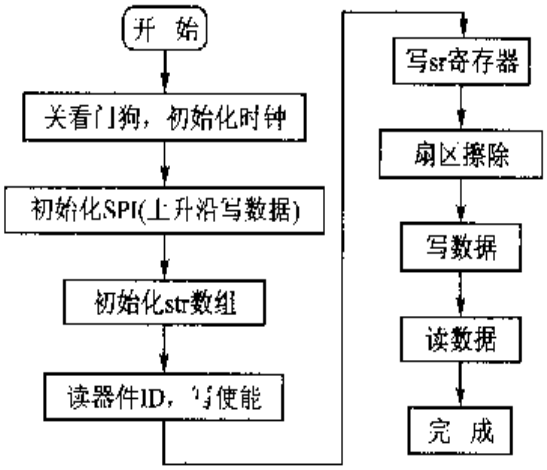


图 9.10　流程图

具体程序如下：

```
#include <msp430x14x.h>
#define uchar unsigned char
#define uint unsigned int
#define spi_cs BIT1
#define da_cs BIT6
#define Enable P2OUT& = ~spi_cs;
#define Disable P2OUT  = spi_cs;
uchar str[256];
uchar ID;
void int_clk()
{
    uchar i;
    BCSCTL1& = ~XT2OFF;                    //打开 XT 振荡器
    BCSCTL2| = SELM1 + SELS;               //MCLK 为 8 MHz,SMCLK 为 1 MHz
    do
    {
        IFG1 & = ~OFIFG;                   //清除振荡错误标志
        for(i = 0; i < 100; i++)
        _NOP();                            //延时等待
    }
    while ((IFG1 & OFIFG) != 0);           //如果标志为 1,则继续循环等待
    IFG1& = ~OFIFG;
}
void int_spi()
{
```

```
    P2DIR |= spi_cs;
    Disable;                          // 芯片使能端禁止
    U0CTL |= SWRST;
    U0CTL  = CHAR + SYNC + MM;
    U0TCTL |= SSEL1 + SSEL0 + STC;    //3 线模式,SMCLK 作为 SPI 的时钟源
    U0TCTL |= CKPH;
    U0TCTL &= ~CKPL;                  //CKPL CKPH:01 上升沿写数据,下降沿读数据
    U0BR0 = 0X02;
    U0BR1 = 0X00;
    U0MCTL = 0X00;
    ME1 |= USPIE0;
    U0CTL &= ~SWRST;
    IE1 &= ~UTXIE0;
    IE1 &= ~URXIE0;
    P3SEL |= 0x0E;
    P3DIR |= BIT2 + BIT4;
}
write_byte(uchar byte)
{
    U0TXBUF = byte;
    while((IFG1&UTXIFG0) == 0);
    IFG1 &= ~UTXIFG0;
}
uchar read_byte()
{
    uchar byte;
    U0TXBUF = 0X00;
    while((IFG1&UTXIFG0) == 0);
    IFG1 &= ~UTXIFG0;
    while((IFG1&URXIFG0) == 1);
    byte = U0RXBUF;
    return(byte);
}
void busy()
{
    uchar i;
    Enable;
    write_byte(0x05);                 //读状态寄存器
    while(1)
```

```
    {
        i = read_byte();
        i&=0x01;
        if(i==0) {Disable;break;}
    }
}
write_enable()
{
    Enable;
    write_byte(0x06);                    //写使能命令
    Disable;
    busy();
}
write_sr()
{
    write_enable();
    Enable;
    write_byte(0x01);                    //状态寄存器写值为 1
    write_byte(0x02);
    Disable;
    busy();
}
section_erase(uchar add1,uchar add2,uchar add3)
{
    write_enable();
    Enable;
    write_byte(0xD8);                    //块查处指令
    write_byte(add1);
    write_byte(add2);
    write_byte(add3);
    Disable;
    busy();
}
write_data(uchar add1,uchar add2,uchar add3,uchar *p)
{
    uchar i;
    Enable;
    write_byte(0x06);                    //写使能指令
    Disable;
```

```
    busy();
    Enable;
    write_byte(0x02);
    write_byte(add1);
    write_byte(add2);
    write_byte(add3);
    for(i=0;i<255;i++)
      {
        write_byte(i);
        p++;
      }
    Disable;
    busy();
}
read_data(uchar add1,uchar add2,uchar add3,uchar *p)
{
    uchar i;
    Enable;
    write_byte(0x03);
    write_byte(add1);
    write_byte(add2);
    write_byte(add3);
    for(i=0;i<255;i++)
    {
        *p=read_byte();
        p++;
    }
    Disable;
}
fast_read(uchar add1,uchar add2,uchar add3,uchar *p)
{
    uchar i;
    Enable;
    write_byte(0x0b);
    write_byte(add1);
    write_byte(add2);
    write_byte(add3);
    write_byte(0);
    for(i=0;i<255;i++)
```

```
        {
          *p = read_byte();
          p++;
        }
    Disable;
}
bulk_erase()
{
    write_enable();
    write_byte(0xc7);
    Disable;
    busy();
}
powerdown()
{
    Enable;
    write_byte(0xb9);
    Disable;
}
powerup()
{
    Enable;
    write_byte(0xab);
    Disable;
}
read_ID()
{
    Enable;
    write_byte(0xab);
    write_byte(0x00);
    write_byte(0x00);
    write_byte(0x00);
    ID = read_byte();
    Disable;
}
int main( void )
{
    uchar *p;
    uint i;
```

```
WDTCTL = WDTPW + WDTHOLD;
int_clk();
int_spi();
p = str;
for(i = 0;i <= 255;i++)
{
    str_i| = i;
}
ID = read_ID();
write_enable();
write_sr();
section_erase(0x00,0x00,0x00);
write_data(0x00,0x00,0x00,p);
read_data(0x00,0x00,0x00,p);
while(1);
}
```

在编程时需要注意下面两点：

- 器件在上升沿接收数据，在下降沿发送数据。不同的存储器可能有所不同，读者须看明手册。因此，在对 SPI 进行设置的时候注意匹配时序，详细请参考 5.2 节。
- 在写状态寄存器、编程和擦除操作过后都需要判忙，只有等到状态寄存器的 WIP 位为 0(表示不忙)之后，才能执行后面的命令。

第 10 章

基于 FYD12864LCD 的数字电压表的设计

10.1 系统概述

该系统主要实现对电压的测量。其中，ADC 采用 MSP430F149 单片机内部自带的 12 位 A/D 转换器，关于它的基本特点读者可以参考第 7 章 ADC12 单通道单次采样部分，这里不再详细讲述。由于 ADC 本身输入范围的限制，我们这里输入信号的最大值为 +2.5 V(读者可以选择其他的参考源，使输入信号最大可达 +3.3 V)。在实际应用中，可以增加前级信号调理电路使输入信号的范围扩大。本例的基本特点和主要指标如下：

- 输入范围：0～+2.5 V；
- 精度：高于 0.2%；
- 通道数：1 路(注：MSP430F149 单片机最多支持 8 通道，在配套实验板上提供给读者的有 2 路通道)；
- 采样频率：100 Hz；
- 显示器：FYD12864LCD 显示器。

10.2 FYD12864 LCD 介绍

液晶显示器主要由液晶屏和驱动部分构成，称为液晶模块。LCD 是 LCM(液晶模块)的一种习惯性叫法。

本章以 FYD12864－0402B 为例，说明 MSP430 单片机与 LCD 的接口与程序设计的具体实现方法和实现过程。

FYD12864 是使用 ST7920 控制器的，带有 4 位/8 位并行、3 线串行多种接口方式，内部含有国标一级、二级简体中文字库的点阵图形式液晶显示模块，其显示分辨率为 128×64，可以显示 8×4 行 16×16 点阵的汉字。该液晶与同类型的液晶相比，具有电路简单和价格低等优点。

其主要特点如下：

- 内置汉字库，有 8192 个 16×16 点汉字(简体、繁体可选)，16×8 点 ASCII 字符库；
- 通信方式支持串口和并口；
- 低电压低功耗(VDD：+3.3～+5 V)；
- 支持文本和图形输出方式；
- 内置 DC-DC 转换电路，无需外加电压；
- 无片选信号，简化软件设计；
- 背光方式：侧部高亮白色 LED，功耗仅为普通 LED 的 1/5、1/10；
- 工作温度：0～+55 ℃，存储温度－20～+60 ℃；
- 8192 个 16×16 点汉字，和 128 个 16×8 点 ASCII 字符集。

FYD12864 内部结构框图如图 10.1 所示。

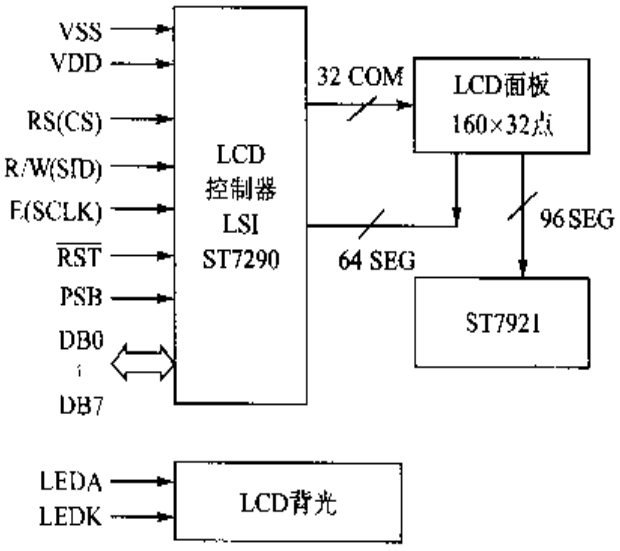


图 10.1　LCD 模块内部框图

10.2.1　FYD12864 的引脚说明

1. 串行接口引脚信号

FYD12864 共有 8 个与串行接口有关的引脚。

这 8 个引脚的相关介绍如表 10.1 所列。

表 10.1　串口接口引脚信号

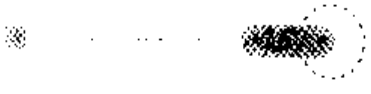
引脚号	引脚名称	电　压	功　能
1	VSS	0 V	电源地
2	VDD	5 V	电源正(3.0~5.5 V,当电压为 3 V 时需更改跳线或另购)
3	V_0	—	对比度调整(使用模块上的 RV 时,该脚悬空)
4	CS	H/L	模组片选端,高电平有效
5	SID	H/L	串行数据输入端
6	CLK	H/L	串行同步时钟:上升沿时读取 SID 数据
15	A	VDD	背光源电压+5 V
16	K	VSS	背光源负端 0 V

2. 并行接口引脚信号

FYD12864 共有 16 个与并行接口有关的引脚,这 16 个引脚的相关介绍如表 10.2 所列。

表 10.2　并行接口引脚信号

引脚号	引脚名称	电　压	引脚功能描述
1	VSS	0 V	电源地
2	VDD	3.0~+5 V	电源电压为正(3.0~5.5 V,当电压为 3 V 时需更改跳线或另购)
3	V_0	—	对比度调整(使用模块上的 RV 时,该脚悬空)
4	RS(CS)	H/L	RS="H",表示 DB7~DB0 为显示数据; RS="L",表示 DB7~DB0 为显示指令数据
5	R/W(SID)	H/L	R/W="H",E="H",数据被读到 DB7~DB0; R/W="L",E="H→L", DB7~DB0 的数据被写到 IR 或 DR
6	E(SCLK)	H/L	使能信号
7	DB0	H/L	三态数据线
8	DB1	H/L	三态数据线
9	DB2	H/L	三态数据线
10	DB3	H/L	三态数据线
11	DB4	H/L	三态数据线
12	DB5	H/L	三态数据线
13	DB6	H/L	三态数据线
14	DB7	H/L	三态数据线
15	A	VDD	背光源正端(+5 V)
16	K	VSS	背光源负端(见厂家数据手册)



3. 控制器接口信号

FYD12864 共有 2 个与控制器接口相关的引脚。这 2 个信号的不同组合对应了不同的功能，具体如表 10.3 所列。

表 10.3　控制器接口信号

RS	R/W	功能说明
L	L	MPU 写指令到指令暂存器(IR)
L	H	读出忙标志(BF)及地址记数器(AC)的状态
H	L	MPU 写入数据到数据暂存器(DR)
H	H	MPU 从数据暂存器(DR)中读出数据

4. E 信号

E 信号(在并行模式下使用)的不同状态对应了不同的操作，具体如表 10.4 所列。

表 10.4　E 信号

E 状态	执行动作	结果
高→低	I/O 缓冲→DR	配合/W 进行写数据或者指令
高	DR→I/O 缓冲	配合 R 进行读数据或者指令
低/低→高	不操作	不操作

5. 忙标志

忙标志(BF)提供内部工作情况：

BF＝1，表示模块在进行内部操作，此时模块不接收外部指令和数据；

BF＝0，模块为准备状态，随时可接收外部指令和数据。

10.2.2　内部模块介绍

1. 字型产生 ROM(CGROM)

字型产生 ROM(CGROM)提供 8192 个字型。此触发器是用于模块屏幕显示开和关的控制。DFF＝1 为开显示(DISPLAY ON)，DDRAM 的内容就显示在屏幕上；DFF＝0 为关显示(DISPLAY OFF)。

DFF 的状态是通过指令 DISPLAY ON/OFF 和 RST 信号控制的。

2. 显示数据 RAM(DDRAM)

模块内部显示数据 RAM 提供 64×2 个位元组的空间，最多可控制 4 行 16 字(64 个字)的中文字型显示。当写入显示数据 RAM 时，可分别显示 CGROM 与 CGRAM 中的字型。此模块可显示 3 种字型，分别是半角英数字型(16×8)、CGRAM 字型及 CGROM 的中文字型。

3. 字型产生 RAM(CGRAM)

字型产生 RAM 提供图象定义(造字)功能，可以提供 4 组 16×16 点的自定义图像空间。用户可以将那些内部字型没有提供的图像字型自行定义到 CGRAM 中，之后，这些字型便可和 CGROM 中的字型一样，来通过 DDRAM 显示在屏幕上。

4. 地址计数器 AC

地址计数器是用来储存 DDRAM/CGRAM 之一的地址，它可通过设定指令暂存器来改变。之后，只要读取或是写入 DDRAM/CGRAM 的值，地址计数器的值就会自动加 1。当 RS 为 0 且 R/W 为 1 时，地址计数器的值会被读取到 DB6～DB0 中。

5. 光标/闪烁控制电路

此模块提供硬体光标及闪烁控制电路，由地址计数器的值来指定 DDRAM 中的光标或闪烁位置。

10.2.3　指令说明

模块控制芯片提供两套控制命令：基本指令和扩充指令。

1. 基本指令

RE=0 时，模块控制芯片提供基本指令，见表 10.5。

表 10.5　指令表 1

指　令	指令码										功　能
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	
清除显示		0								1	将 DDRAM 填满"20H"，并且设定 DDRAM 的地址计数器(AC)到"00H"
地址归位		0								x	设定 DDRAM 的地址计数器(AC)到"00H"，并且将游标移到开头原点位置，这个指令不改变 DDRAM 的内容

续表 10.5

指　令	指令码										功　能
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	
显示状态开/关		0								B	D=1：整体显示 ON；C=1：游标 ON； B=1：游标位置反白允许
进入点设置	0	0		0	0	0	0		I/D	S	指定在数据的读取与写入时，设定游标的移动方向及指定显示的移位
功能设定	0	0		0	1	DL	X	RE	X	X	DL=0/1：4/8 位数据； RE=1：扩展指令操作； RE=0：基本指令操作
设定 CGRAM 地址	0	0			AC5	AC4	AC3	AC2	AC1	AC0	设定 CGRAM 地址
设定 DDRAM 的值	0	0		0	C5	AC4	AC3	AC2	AC1	AC0	设定 DDRAM 地址（显示地址） 第一行：80H～87H； 第二行：90H～97H
读取忙标志和地址	0	1	F	C6	AC5	AC4	AC3	AC2	AC1	AC0	读取忙标志（BF），可以确认内部动作是否完成，同时可以读取地址计数器（AC）的值
写入数据 RAM		0	数据								将数据 D7～D0 写入到内部的 RAM（DDRAM/CGRAM/IRAM/GRAM）
读出 RAM 的值	0	1	数据								从内部 RAM 读取数据 D7～D0（DDRAM/CGRAM/IRAM/GRAM）

2. 扩展指令

RE=1 时，模块控制芯片提供扩展指令，见表 10.6。

表 10-6　指令表 2

指　令	指令码										功　能
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	
待命模式		0								1	进入待命模式，执行其他指令都可以终止待命模式
卷动地址开关开启		0								SR	SR=1：允许输入垂直地址 SR=0：允许输入 IRAM 和 CGRAM 地址

续表 10.6

指　令	指令码										功　能
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	
反白选择		0							R1	R0	选择 2 行中的任一动作反白显示，并可决定反白与否。初始值 R1R0＝00，第一次设定为反白显示，再次设定变回正常
睡眠模式	0	0		0	0	0		SL	X	X	SL＝0：进入睡眠模式； SL＝1：脱离睡眠模式
扩展功能设定	0	0		0		CL	X	RE	G	0	CL＝0/1：4/8 位数据； RE＝1：扩展指令操作； RE＝0：基本指令操作； G＝1/0：绘图开关
设定绘图 RAM 地址	0	0		0 C6	0 AC5	0 AC4	AC3 AC3	AC2 AC2	AC1 AC1	AC0 AC0	设定绘图 RAM 先设定垂直(列)地址：AC6～AC0 再设定水平(行)：C3AC2AC1AC0 将以上 16 位地址连续写入

备注：LCD 在接收指令前，微处理器必须先确认其内部处于非忙碌状态，即读取 BF 标志时，BF 须为 0，方可接收新的指令；如果在送出一个指令前并不检查 BF 标志，那么在前一个指令和这个指令中间必须延长一段较长的时间，即等待前一个指令确实执行完成。图 10.2 所示为 LCD 工作于串行方式下的数据传输时序图，所有对 LCD 的数据/命令都必须严格遵循该时序。

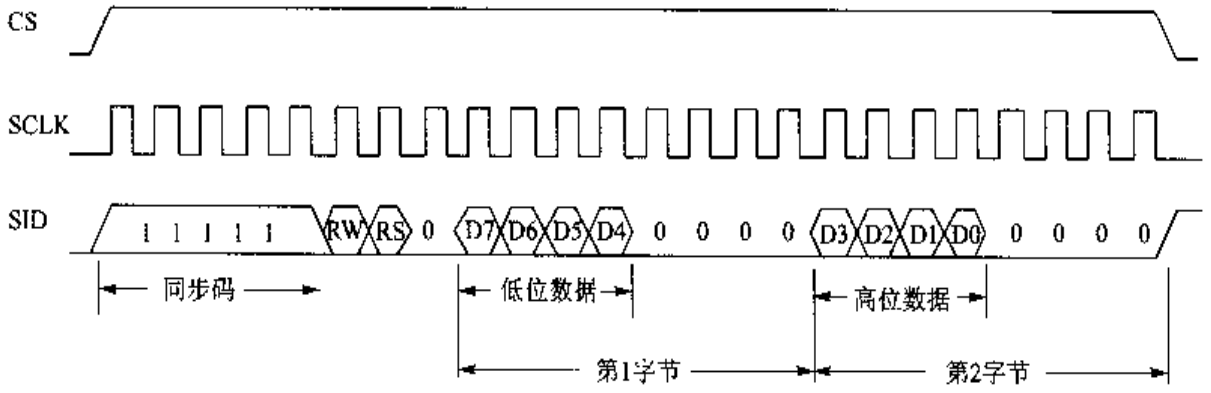


图 10.2　串行方式下数据传输时序图

由图可见，每写 1 个字节的数据共需要 3 个字节的数据输出。

第 1 字节为同步码：1 1 1 1 1 RW RS 0 共 8 位；

第 2 字节为：D7 D6 D5 D4 0 0 0 0 共 8 位，该字节由待输出字节的高 4 位与 4 个 0 组成；

第 3 字节为：D3 D2 D1 D0 0 0 0 0 共 8 位，该字节由待输出字节的低 4 位与 4 个 0 组成。

其中需要注意的是，数据传输时都是高位在前，低位在后，且 CS 一直为高有效。

10.3 系统电路设计

1. LCD 接口电路

LCD 显示器接口电路如图 10.3 所示。由于 MSP430 单片机没有外部总线接口，在本设计中采用串行工作方式，这样将使电路变得简单，更重要的是占用单片机的 I/O 资源少。下面以串行工作方式来举例说明。

2. ADC 采样电路

在配套实验板中，我们提供 2 路 ADC，并且可以通过板上自带的电位器来调节输入电压，这样能更好地方便读者使用。这里我们对通道 ADC0 进行采样，并且在 LCD 上显示出来。为提高系统的精度，ADC 的前级我们增加了一个 RC 滤波电路。另外，PCB 走线对 ADC 本身的采样精度也有很大的影响，读者可以参考一些 PCB 设计的相关书籍。

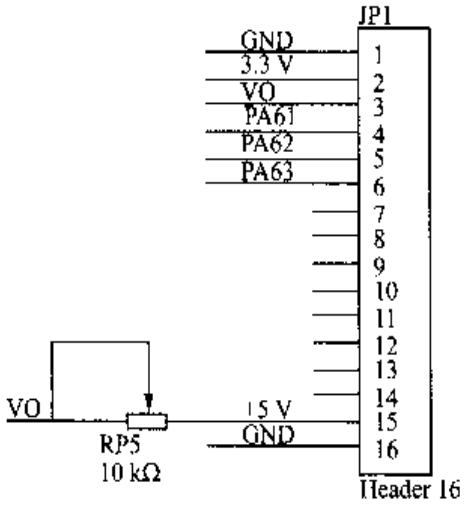


图 10.3 LCD 在串行方式接口电路

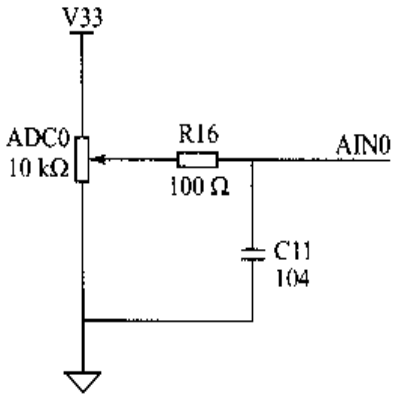


图 10.4 ADC 采样电路

10.4 程序设计

下面介绍数字电压表的程序设计，由 MSP430F149 片上的 ADC 进行 A/D 采样，然后通过 LCD 显示。其中，主程序流程、定时器子程序流程以及 ADC 子程序流程分别如图 10.5、

图 10.6和图 10.7 所示。通过程序的设计，使读者对片上 ADC 进一步熟悉；另外，在程序中给出了具体的 LCD 操作过程，可以很方便地应用到其他需要 LCD 显示的场合中去。

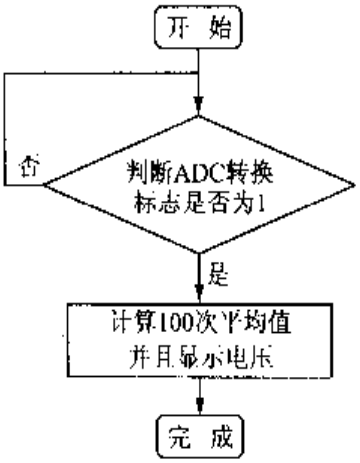


图 10.5　主程序流程图

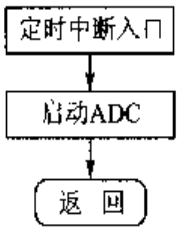


图 10.6　定时器子程序流程图

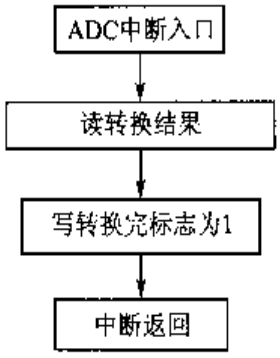


图 10.7　ADC 子程序流程图

具体程序如下：

```
#include <msp430x14x.h>
#define  uint  unsigned int
#define  uchar unsigned char
/*********************define  lcd*****/
#define LCD_CS  BIT2
#define LCD_DAT  BIT4
#define LCD_SCK  BIT3
#define WR BIT2
#define RS BIT1
/*************************************/

uchar charge = 0;
uchar SYNCH = 0xfe;                          //LCD 显示同步字
static uchar adc_flag = 0;
static uchar time_flag = 0;
static uchar count = 0;
uint AD_TEMP = 0;
float AD = 0;
void int_clk()
{
    uchar i;
    BCSCTL1& = ~XT2OFF;                      //打开 XT2 振荡器
```

```
    BCSCTL2 |= SELM1 + SELS;                    //MCLK 为 8 MHz,SMCLK 为 1 MHz
    do
    {
        IFG1 &= ~OFIFG;                         //清除振荡错误标志
        for(i = 0; i < 100; i++)
        _NOP();                                 //延时等待
    }
    while ((IFG1 & OFIFG) != 0);                //如果标志为 1,则继续循环等待
    IFG1&= ~OFIFG;
}
void get_clk()
{
    P6DIR |= LCD_SCK;
    _NOP();
    P6OUT&= ~LCD_SCK;
    _NOP();
    _NOP();
    _NOP();
    _NOP();
    _NOP();
    P6OUT |= LCD_SCK;
    _NOP();
    _NOP();
    _NOP();
    _NOP();
    _NOP();
    _NOP();
    _NOP();
}
void inport(uchar com,uchar dat)//com = 1,rs = 1,com = 0,rs = 0;
{
    uchar i,temp;
    P6DIR |= LCD_DAT;
    P6DIR |= LCD_CS;
    P6OUT |= LCD_CS;
    temp = SYNCH;
    temp&= ~WR;
    if(com)     temp |= RS;
    else        temp&= ~RS;
```

```
    for(i = 0;i <= 7;i ++ )
    {
        if(temp << i&0x80)        P6OUT| = LCD_DAT;
        else                      P6OUT& = ~LCD_DAT;
        get_clk();
    }
    temp = dat&0xf0;
    for(i = 0;i <= 7;i ++ )
    {
        if(temp<< i&0x80)         P6OUT| = LCD_DAT;
        else                      P6OUT& = ~LCD_DAT;
        get_clk();
    }
    temp = dat&0x0f;
    temp = temp<<4;
    for(i = 0;i <= 7;i ++ )
    {
        if(temp<< i&0x80) P6OUT  = LCD_DAT;
        else                      P6OUT& = ~LCD_DAT;
        get_clk();
    }
    P6OUT| = LCD_DAT;
    P6OUT& = ~LCD_CS;
}
print_string(uchar add,uchar * pt)
{
    uchar i;
    inport(0x00,add);
    for(i = 0;;i ++ )
    {
        if( * pt == \n') break;
        inport(0x01, * pt);
        pt ++ ;
    }
}
print_han(uchar h_neima,uchar l_neima)\
{
    inport(0x01,h_neima);
    inport(0x01,l_neima);
```

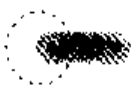
```
}
print_float(float data)
{
    uint temp;
    uchar a1,a2,a3,a4;
    temp = data * 1000;
    a1 = temp/1000;
    a2 = (temp - a1 * 1000)/100;
    temp = (temp - a1 * 1000) - a2 * 100;
    a3 = temp/10;
    a4 = temp % 10;
    inport(0x01,a1 + 0x30);
    inport(0x01,.);
    inport(0x01,a2 + 0x30);
    inport(0x01,a3 + 0x30);
    inport(0x01,a4 + 0x30);
}
void int_lcd()
{
    inport(0x00,0x30);
    inport(0x00,0x01);
    inport(0x00,0x02);
    inport(0x00,0x0c);
    inport(0x00,0x06);
}
int_adc()
{
    P6SEL| = 0x01;                                  //选择 A/D 通道
    ADC12CTL0 = ADC12ON + SHT0_2 + REF2_5V + REFON; //采样保持时间为 16 个 ADC12CLK
    ADC12CTL1  = ADC12SSEL1 + ADC12SSEL1;           //参考电压开启,选择 2.5 V
    ADC12MCTL0 = 0x10;                              // ref + = REF2_5V, channel = A0
    ADC12IE| = 0x01;                                // 使能转换中断
    ADC12CTL0| = ENC;                               // 使能 A/D 转换器
}
#pragma vector = ADC_VECTOR
__interrupt void ADC12ISR (void)
{
    while((ADC12CTL1&0X01) == 1);                   //等待转换完成
    adc_flag = 1;
```

```
    AD_TEMP = ADC12MEM0;                    //设置 A/D 转换完成标志,并读取 ADC 值
}
void int_timer()
{
    TACTL| = TASSEL1 + TACLR + ID0 + ID1;   //选择 SMCLK 作为定时器的时钟,8 分频
    TACTL| = MC0 + TAIE;                    //增加模式,并且使能中断
    TACCR0 = 9999;                          //中断周期为 10 ms
}
#pragma vector = TIMERA1_VECTOR
__interrupt void Timer_A(void)
{
    switch(TAIV)
    {
        case 2:break;
        case 4:break;
        case 10:if(adc_flag = = 0){ADC12CTL0 | = ADC12SC;ADC12CTL0 &= ~ADC12SC;}break;
                                            //10 ms 中断标志加 1
        default:break;
    }
}
int main( void )
{
    WDTCTL = WDTPW + WDTHOLD;               //关闭看门狗
    int_clk();                              //系统时钟初始化
    int_adc();                              //初始化串口
    int_lcd();                              //初始化 LCD
    int_timer();                            //初始化 TIMER
    _EINT();                                //使能中断
    while(1)
    {
        while(adc_flag = = 1)
        {
            AD + = AD_TEMP;
            count + + ;
            if(count = = 100)
            {
                AD = AD/100;
                AD = (AD * 2.5/4096);
                inport(0x00,0x80);
```

```
                print_han(0xb5,0xe7);
                print_han(0xd1,0xb9);        //电压
                inport(0x01,':');
                print_float(AD);
                inport(0x01,'V');            //显示电压值
                AD = 0;
                count = 0;
                AD_TEMP = 0;                 //变量清零
            }
            adc_flag = 0;                    //清中断标志
        }
    }
}
```

通过上面的学习，希望读者可以更进一步了解 MSP430F149 的 A/D 转换器，同时熟悉 LCD 的程序设计。

注意：工作在串口方式和工作电压在 3.3 V 时，需要对该液晶进行设置，具体可以参考厂家的数据手册。



第 11 章

基于 PCF8563 的时钟设计

在很多情况下，电子系统需要提供一个精确的时间，作为显示和记录事件发生的具体时间，并且为待处理数据提供时间参考。本实例采用 PCF8563 来实现数字时钟系统的设计。同时，该例中提供的 I^2C 程序包可以很方便地应用到其他器件。

11.1 PCF8563 芯片概述

实时时钟芯片 PCF8563 已经被广泛地应用到了现代电子系统中。PCF8563 具有 I^2C 总线接口，与 MCU 之间的连接只需要两个通用 I/O 端口。PCF8563 是一款低功耗的 CMOS 实时时钟/日历芯片，它提供一个可编程时钟输出、一个中断输出和掉电检测器。所有的地址和数据通过 I^2C 总线接口串行传递。最大总线速度为 400 Kb/s。每次读/写数据后，内嵌的字地址发生器会自动产生增量，适合连续字节的读/写操作。

基本特点如下：

- 低工作电流：典型值为 0.25 μA（V_{DD}=3.0 V，T_{amb}=25 ℃时）；
- 世纪标志；
- 大工作电压范围：1.0～5.5 V；
- 低休眠电流：典型值为 0.25 μA（V_{DD}=3.0 V，T_{amb}=25 ℃）；
- 400 kHz 的 I^2C 总线接口（V_{DD}=1.8～5.5 V 时）；
- 可编程时钟输出频率为：32.768 kHz，1024 Hz，32 Hz，1 Hz；
- 报警和定时器；
- 掉电检测器；
- 内部集成的振荡器电容；

- 片内电源复位功能；
- I²C 总线从地址：读为 0A3H，写为 0A2H；
- 开漏中断引脚。

11.2 PCF8563 引脚描述

PCF8563 是一个 8 脚的实时时钟/日历芯片，其引脚分布如图 11.1 所示。相应的引脚描述见表 11.1。

表 11.1 引脚描述

符 号	引脚号	描 述
OSCI	1	振荡器输入
OSCO	2	振荡器输出
$\overline{INT}$	3	中断输出(开漏;低电平有效)
VSS	4	电源地
SDA	5	串行数据 I/O
SCL	6	串行时钟输入
CLKOUT	7	时钟输出(漏极开路)
VDD	8	正电源

图 11.1 PCF8563 引脚图

11.3 PCF8563 的功能描述

PCF8563 有 16 个 8 位寄存器：1 个可自动增量的地址寄存器，1 个内置 32.768 kHz 的振荡器，1 个分频器，1 个可编程时钟输出，1 个定时器，1 个报警器，1 个掉电检测器和 1 个 400 kHz 的 I²C 总线接口。

内部 16 个寄存器分别为：

(1)内存地址为 00H、01H 的寄存器，用于作为控制寄存器和状态寄存器；

(2)内存地址为 02H~08H 的寄存器，用于时钟计数器(秒、分、小时、日、月、年计数器)；

(3)内存地址为 09H~0CH 的寄存器，用于报警寄存器(定义报警条件)；

(4)内存地址为 0DH 的寄存器，用于控制 CLKOUT 引脚的输出信号频率；

(5)内存地址为 0EH、0FH，分别用于定时器控制寄存器和定时器寄存器。

当 V_{DD} 低于 V_{low} 时，PCF8563 内嵌的掉电检测器，位 VL(Voltage Low，秒寄存器的位 7)

被置 1，用于指明可能产生不准确的时间信息。VL 标志位只能用软件清除。当 V_{DD} 慢速降低（例如以电池供电）达到 V_{low} 时，标志位 VL 被置位，这时可能会产生中断。所以在实时系统设计中都要给时钟芯片配备一个备用电池，防止在系统掉电时产生不准确的时间信息。掉电检测过程如图 11.2 所示。

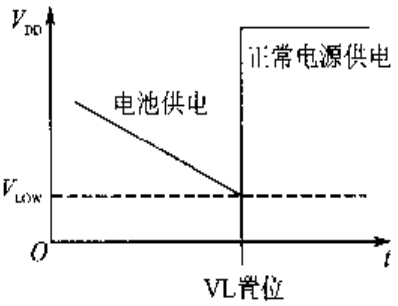


图 11.2　掉电检测过程

11.4　电路设计

图 11.3 所示为 PCF8563 的接口电路图，其中 C2 是为了提高时钟系统的稳定性而设置的。BAT1 是系统中的备用电池，在系统中的主电源掉电时为系统提供电能，保持时间的正确性，此时 PCF8563 会进入低功耗模式。一个普通的电池就可以保证实时时钟芯片数月甚至更长的准确时间。

图中的 D1 和 D2 使用了二极管的单向导电性。上系统电源供电时，备用电池不工作；系统掉电后，用备用电池充当电源。根据 IIC 规范，SDA、SCL 必须接上拉电阻，如图 11.3 所示。

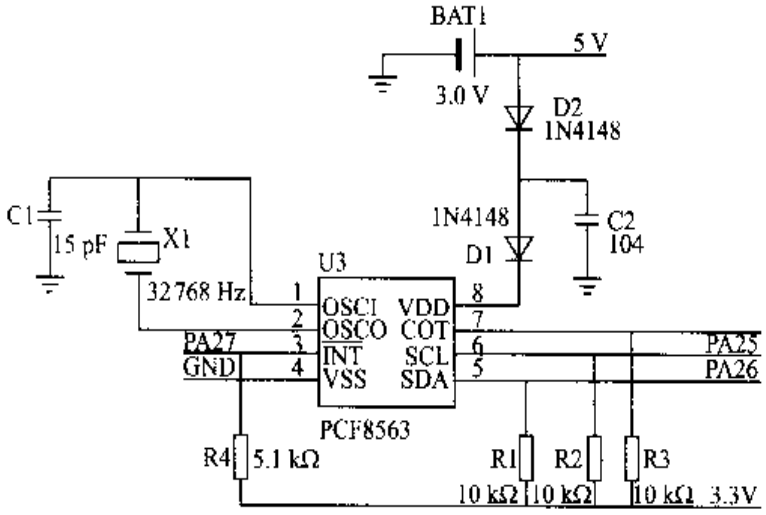


图 11.3　PCF8563 的接口电路

11.5　系统软件设计

系统软件设计的流程图如图 11.4 所示。该系统主要实现数字时钟功能，读者可以根据实际需要读取和设置时间信息，配套实验板已经写入现在时间，并且通过备用电池使时间走动。

读者可以直接读取得到现在的时间信息，也可以根据自己的需要修改程序。由于实验板提供的 LED 显示器只有 4 位，要想完整地显示时间信息，则需要采用 LCD 显示器。关于 LCD 显示器的基本原理读者可以参考第 10 章的内容，这里只是直接应用，不再进行详细讲解。同时，读者如果需要修改时间的话，可以通过实验板提供的键盘和相关的程序来进行，这里就不再详细说明了。

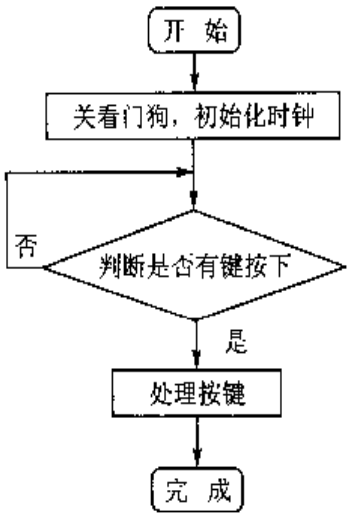


图 11.4 软件设计流程图

具体程序设计如下：

```
#include <msp430x14x.h>
//#include "lcd.c"
//#include "key.c"
#define  uint  unsigned int
#define  uchar unsigned char
#define  SDA  BIT0
#define  SCL  BIT1
#define  DIN  P1IN
#define  DOUT  P1OUT
#define  DDIR P1DIR
#define  S_SDA DDIR  = SDA,DOUT| = SDA
#define  C_SDA DDIR| = SDA,DOUT&= ~SDA
#define  S_SCL DDIR  = SCL,DOUT| = SCL
#define  C_SCL DDIR  = SCL,DOUT&= ~SCL
#define  SDA_IN DDIR&= ~SDA
#define  READ_SDA (DIN&SDA)
#define  delay() _NOP(),_NOP(),_NOP(),_NOP(),_NOP(),_NOP(),_NOP(),_NOP(),_NOP(),_NOP(),_
NOP(),_NOP(),_NOP(),_NOP(),_NOP(),_NOP(),\

_NOP(),_NOP(),_NOP(),_NOP(),_NOP(),_NOP(),_NOP(),_NOP(),_NOP(),_NOP(),_NOP(),_NOP(),_NOP
(),_NOP(),_NOP(),_NOP()
/*********************************************************/

uchar static key_flag=0;
/*********************************************************/
uchar temp=0;
uchar ack;
uchar rom_sed[9];
uchar rom_rec[7];

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

```
/*************************define lcd*****/
#define LCD_CS BIT2
#define LCD_DAT BIT4
#define LCD_SCK BIT3
#define WR BIT2
#define RS BIT1
/*************************************************/
uchar charge = 0;
uchar SYNCH = 0xfe;                    //LCD 显示同步字
uchar time = 0;
uchar hour = 0,min = 0,sec = 0;
uchar static adc_flag = 0;
void int_clk()
{
    uint i;
    do
    {
        BCSCTL1&= ~XT2OFF;             //SET XT2 IS ON
        IFG1&= ~OFIFG;                 //CLEAR Oscillator fault flag
        for(i = 0;i <= 1000;i++)
          _NOP();
    }
    while(IFG1&OFIFG!= 0);
    BCSCTL2|= SELM1 + SELS;            //SELECT XT2 FOR MCLK,FOR SMCLK AND DIVIDE FOR 0
                                       //ACLK 32 768 Hz
}
void get_clk()
{
    P6DIR|= LCD_SCK;
    _NOP();
    P6OUT&= ~LCD_SCK;
    _NOP();
    _NOP();
    _NOP();
    _NOP();
    _NOP();
    P6OUT|= LCD_SCK;
    _NOP();
    _NOP();
```

```
    _NOP();
    _NOP();
    _NOP();
    _NOP();
    _NOP();
}
void inport(uchar com,uchar dat)//com = 1,rs = 1,com = 0,rs = 0;
{
    uchar i,temp;
    P6DIR| = LCD_DAT;
    P6DIR| = LCD_CS;
    P6OUT| = LCD_CS;
    temp = SYNCH;
    temp& = ~WR;
    if(com)                         temp| = RS;
    else                            temp& = ~RS;
    for(i = 0;i <= 7;i++)
    {
        if(temp << i&0x80)          P6OUT| = LCD_DAT;
        else                        P6OUT& = ~LCD_DAT;
        get_clk();
    }
    temp = dat&0xf0;
    for(i = 0;i <= 7;i++)
    {
        if(temp << i&0x80)          P6OUT| = LCD_DAT;
        else                        P6OUT& = ~LCD_DAT;
        get_clk();
    }
    temp = dat&0x0f;
    temp = temp << 4;
    for(i = 0;i <= 7;i++)
    {
        if(temp << i&0x80)          P6OUT| = LCD_DAT;
        else                        P6OUT& = ~LCD_DAT;
        get_clk();
    }
    P6OUT = LCD_DAT;
    P6OUT& = ~LCD_CS;
```

```
}
print_char(uchar add,uchar data)
{
    inport(0x00,add);
    inport(0x01,data);
}
print_string(uchar add,uchar * pt)
{
    uchar i;
    inport(0x00,add);
    for(i = 0;;i++)
    {
        if( * pt == '\n') break;
        inport(0x01, * pt);
        pt++;
    }
}
print_han(uchar h_neima,uchar l_neima)\

    inport(0x01,h_neima);
    inport(0x01,l_neima);
}
print_float(float data)
{
    uint temp;
    uchar a1,a2,a3,a4;
    temp = data * 1000;
    a1 = temp/1000;
    a2 = (temp - a1 * 1000)/100;
    temp = (temp - a1 * 1000) - a2 * 100;
    a3 = temp/10;
    a4 = temp % 10;
    inport(0x01,a1 + 0x30);
    inport(0x01,'.');
    inport(0x01,a2 + 0x30);
    inport(0x01,a3 + 0x30);
    inport(0x01,a4 + 0x30);
}
void int_lcd()
```

```
{
    inport(0x00,0x30);
    inport(0x00,0x01);
    inport(0x00,0x02);
    inport(0x00,0x0c);
    inport(0x00,0x06);
}
/*************************************************************/
void delay2()
{
    uint i;
    for(i=0;i<=1000;i++)
        ;
}
uchar scan_key(uchar line)
{
    uchar key;
    P5OUT=line;
    switch(P5IN&0Xf0)
    {
        case 0xe0:key=1+line;break;
        case 0xd0:key=2+line;break;
        case 0xb0:key=3+line;break;
        case 0x70:key=4+line;break;
        default:key=0xff;break;
    }
    return(key);
}
void fast_scan()
{
    uchar temp;
    P5OUT&=0XF0;                    //低位全部为0,判断有无键按下
    temp=P5IN&0xf0;
    if(temp==0xf0)key_flag=0;
    else
    {
        key_flag=1;
    }
```

```
}
void over()
{
    uchar temp;
    P5OUT& = 0XF0;                    //低位全部为 0,判断有无键按下
    do
    {
        temp = P5IN&0xf0;
        if(temp = = 0xf0)break;
    }
    while(1);
}
uchar get_key()
{
    uchar key;
    while(1)
    {
        key = scan_key(0xe);
        if(key! = 0xff){key| = 0x80;P5OUT = 0X0f; break;}
        key = scan_key(0xd);
        if(key! = 0xff){key| = 0x40;P5OUT = 0X0f;break;}
        key = scan_key(0xb);
        if(key! = 0xff){key| = 0x20;P5OUT = 0X0f;break;}
        key = scan_key(0x7);
        if(key! = 0xff){key  - 0x20;P5OUT = 0X0f;break;}
    }
    key = key;
    switch(key)
    {
        case 0x8f:key = 0;break;
        case 0x90:key = 1;break;
        case 0x91:key = 2;break;
        case 0x92:key = 3;break;
        case 0x4E:key = 4;break;
        case 0x4F:key = 5;break;
        case 0x50:key = 6;break;
        case 0x51:key = 7;break;
        case 0x2C:key = 8;break;
```

```
            case 0x2D:key = 9;break;
            case 0x2E:key = 10;break;
            case 0x2F:key = 11;break;
            case 0x28:key = 12;break;
            case 0x29:key = 13;break;
            case 0x2A:key = 14;break;
            case 0x2B:key = 15;break;
            default:key = 0xff;break;
        }
        _NOP();
        return(key);
}
/*************************************************/
/**********
启动总线函数
**********/
void start()
{
        S_SDA;
        _NOP();
        S_SCL;
        delay();
        C_SDA;
        delay();
        C_SCL;
        _NOP();
        _NOP();
}
/**********
停止总线函数
**********/
void stop()
{
        C_SCL;
        _NOP();
        C_SDA;
        delay();
        S_SCL;
        delay();
```

```
    S_SDA;
    _NOP();
    _NOP();
    _NOP();
    _NOP();
}
/*********************************
发送一个字节函数
*********************************/
void SendByte(uchar c)
{
    uchar i;
    for(i=0;i<8;i++)
    {
        if((c<<i)&0x80)
            S_SDA;
        else
            C_SDA;
        delay();
        S_SCL;
        delay();
        C_SCL;
        delay();
    }
    S_SCL;
    delay();
    SDA_IN;
    _NOP();
    _NOP();
    _NOP();
    if(DIN&SDA==SDA)
        ack=0;
    else
        ack=1;
    delay();
    C_SCL;
    _NOP();
    _NOP();
}
```

```
/*********************************
停止总线函数
*********************************/
void send_ack(uchar a)
{
    if(a==0)
    {
        C_SDA;
        delay();
        S_SCL;
        delay();
        C_SCL;
        delay();
        S_SDA;
    }
    else
    {
        S_SDA;
        delay();
        S_SCL;
        delay();
        C_SCL;
        delay();
    }
}
/*********************************
接收一个字节函数
*********************************/
uchar RcvByte()
{
    uchar retc;
    uchar i;
    retc=0;
    SDA_IN;
    for(i=0;i<8;i++)
    {
        delay();
        S_SCL;
        delay();
```

```
        retc = retc << 1;
        SDA_IN;
        if(READ_SDA == SDA)//read p4in
        retc += 1;
        delay();
        C_SCL;
        delay();
    }
    return(retc);
}
/****************************************
接收一个字节函数
****************************************/
uchar ISendByte(uchar sla,uchar c)
{
    start();
    SendByte(sla);
    if(ack == 0)
        return(0);
    SendByte(c);
    if(ack == 0)
        return(0);
    stop();
    return(1);
}
/****************************************
发送一个数据串函数
****************************************/
uchar ISendstr(uchar sla,uchar suba,uchar *s,uchar no)
{
    uchar i;
    start();
    SendByte(sla);
    if(ack == 0)
        return(0);
    SendByte(suba);
    if(ack == 0)
        return(0);
    for(i = 0;i<no;i++)
```

```
    {
        SendByte( * s);
        if(ack == 0)
            return(0);
        s++;
        suba++;
    }
    stop();
    return(1);
}
/*******************************************
接收多个数据串函数
*******************************************/
uchar IRcvbyte(uchar sla,uchar *c)
{
    start();
    SendByte(sla+1);
    if(ack == 0)
        return(0);
    *c = RcvByte();
    send_ack(1);
    stop();
    return(1);
}
//check
uchar IRcvStr(uchar sla,uchar suba,uchar *s,uchar no)
{
    uchar i;
    start();
    SendByte(sla);
    if(ack == 0)
        return(0);
    SendByte(suba);
    if(ack == 0)
        return(0);
    start();
    SendByte(sla+1);
    if(ack == 0)
        return(0);
```

```
    for(i = 0;i<no  1;i++)
    {
        * s = RcvByte();
        send_ack(0);
        s++;
    }
    * s = RcvByte();
    temp = * s;
    send_ack(1);
    stop();
    return(1);
}
uchar static time_flag = 0;
void int_timer()
{
    TACTL' = TASSEL1 + TACLR + ID0 + ID1;          //选择 SMCLK 作为定时器的时钟,8 分频
    TACTL| = MC0 + TAIE;                           //增加模式,并且使能中断
    TACCR0 = 9999;                                 //中断周期为 10 ms
}
#pragma vector = TIMERA1_VECTOR
__interrupt void Timer_A(void)
{
    switch(TAIV)
    {
        case 2:break;
        case 4:break;
        case 10:time_flag = 1;break;               //10 ms 中断标志加 1
    }
}
uchar key = 0xff;
uchar static set_flag = 0;
uchar static ok_flag = 0;
uchar static cansel_flag = 0;
uchar static year_flag = 0;
uchar static month_flag = 0;
uchar static day_flag = 0;
uchar static hour_flag = 0;
uchar static minute_flag = 0;
uchar static second_flag = 0;
```

```
uchar static count = 0;
uchar static lcd_flag = 1;
uchar static iic_flag = 0;
struct RTC
{
    uchar year;
    uchar month;
    uchar day;
    uchar hour;
    uchar minute;
    uchar seconed;
};
struct RTC DATA;
/*
DATA.year = 0x07;
DATA.month = 0x01;
DATA.day = 0x01;
DATA.hour = 0x01;a
DATA.minute = 0x00;
DATA.second = 0x00; */
    uchar tt = 0;
    uchar static aa = 0;
int main( void )
{
    uchar time = 0.xx;
    uchar flag = 0;
    uchar count = 0;
    uchar temp[16] = {0};
    uchar i = 0;
    uchar *p;
    uchar bb = 0;
    WDTCTL = WDTPW + WDTHOLD;
    int_clk();
    int_lcd();
    P5DIR = BIT0 + BIT1 + BIT2 + BIT3;
    _EINT();
    while(1)
    {
        fast_scan();
```

```
if(key_flag == 1)
{
    delay2();
    key = get_key();
    over();
    delay2();
}
if(key! = 0xff)
{
    switch(key)
    {
        case 0: {lcd_flag = 1;temp[count] = key;count ++ ; if(count == 16) count = 0;
        key = 0xff;break;}
        case 1: {lcd_flag = 1;temp[count] = key;count ++ ; if(count == 16) count = 0;
        key = 0xff;break;}
        case 2: {lcd_flag = 1;temp[count] = key;count ++ ; if(count == 16) count = 0;
        key = 0xff;break;}
        case 3: {lcd_flag = 1;temp[count] = key;count ++ ; if(count == 16) count = 0;
        key = 0xff;break;}
        case 4: {lcd_flag = 1;temp[count] = key;count ++ ; if(count == 16) count = 0;
        key = 0xff;break;}
        case 5: {lcd_flag = 1;temp[count] = key;count ++ ; if(count == 16) count = 0;
        key = 0xff;break;}
        case 6: {lcd_flag = 1;temp[count] = key;count ++ ; if(count == 16) count = 0;
        key = 0xff;break;}
        case 7: {lcd_flag = 1;temp[count] = key;count ++ ; if(count == 16) count = 0;
        key = 0xff;break;}
        case 8: {lcd_flag = 1;temp[count] = key;count ++ ; if(count == 16) count = 0;
        key = 0xff;break;}
        case 9: {lcd_flag = 1;temp[count] = key;count ++ ; if(count == 16) count = 0;
        key = 0xff;break;}
        case 10: {set_flag = 1;count = 0;lcd_flag == 1;int_timer();
        key = 0xff;iic_flag = 0;break;}                    //初始化定时器
        case 11: {ok_flag = 1;count = 0;aa = 0; print_char(0x90,aa + 0x30);TACTL& = ~
        TAIE;key = 0xff;break;}
        case 12: {key = 0xff;lcd_flag = 1;if(count<15)count ++ ;break;}
        case 13: {key = 0xff;lcd_flag = 1;if(count! = 0)count -- ;break;}
        case 14: {iic_flag = 1;key = 0xff;break;}
        case 15: key = 0xff;break;
```

```
            default: break;
        }
    }
    if(lcd_flag == 1)
    {
        lcd_flag = 0;
        inport(0x00,0x80);
        for(i = 0;i<16;i++)
        {
            inport(0x01,temp[i] + 0x30);
        }
    }
    if(time_flag == 1)
    {
        time++ ;time_flag = 0;
    }
    if(time == 8)
    {
        //if(count!= 0)      bb = count - 1;
        // else
        bb = count;
        // TACTL& = ~TAIE;
        time = 0;
        // lcd_flag = 1;
        if((bb % 2)!= 0)
        {
            inport(0x00,bb/2 + 0x80);
            inport(0x01,temp[bb - 1] + 0x30);
        }
        else
        {
            inport(0x00,bb/2 + 0x80);
        }
        if(flag == 0)
        {
            flag = 1;inport(0x01,' ');
            lcd_flag = 0;
        }
        else
            ※
```

```
    {
        flag = 0;inport(0x01,temp[bb] + 0x30);
        lcd_flag = 1;
    }
    // TACTL  = TAIE;
}
if(ok_flag == 1)
{
    ok_flag = 0;
    inport(0x00,0x01);
    iic_flag = 1;
    rom_sed[0] = 0x00;
    rom_sed[1] = 0x00;
    rom_sed[2] = temp[15]|(temp[14] << 4);
    rom_sed[3] = temp[13]|(temp[12] << 4);
    rom_sed[4] = temp[11]|(temp[10] << 4);
    rom_sed[5] = temp[9]|(temp[8] << 4);
    rom_sed[6] = temp[7]|(temp[6] << 4);
    rom_sed[7] = temp[5]|(temp[4] << 4);
    rom_sed[8] = temp[3]|(temp[2] << 4);
    p = rom_sed;
    ISendstr(0xA2,0x00,p,9);                    //设置开始时间
}
if(iic_flag == 1)
{
    inport(0x00,0x89);
    delay();
    delay();
    delay();
    delay();
    delay();
    _NOP();
    _NOP();
    p = rom_rec;
    IRcvStr(0xA2,0x02,p,0x07);                  //读当前时间
    rom_rec[0]& = 0x7f;//s
    inport(0x01,(rom_re[0]&0x0f) + 0x30);
    rom_rec[0] = rom_rec[0] >> 4;
    inport(0x01,(rom_rec[0]&0x0f) + 0x30);
```

```
            rom_rec[1]&=0x7f;//m
            inport(0x01,(rom_rec[1]&0x0f)+0x30);
            rom_rec[1]=rom_rec[1]>>4;
            inport(0x01,(rom_rec[1]&0x0f)+0x30);
            rom_rec[2]&=0x3f;//x
            inport(0x01,(rom_rec[2]&0x0f)+0x30);
            rom_rec[2]=rom_rec[2]>>4;
            inport(0x01,(rom_rec[2]&0x0f)+0x30);
            rom_rec[3]&=0x3f;
            inport(0x01,(rom_rec[3]&0x0f)+0x30);
            rom_rec[3]=rom_rec[3]>>4;
            inport(0x01,(rom_rec[3]&0x0f)+0x30);
            rom_rec[4]&=0x07;
            inport(0x01,(rom_rec[4]&0x0f)+0x30);
            rom_rec[5]&=0x1f;
            inport(0x01,(rom_rec[5]&0x1f)+0x30);
            rom_rec[6]&=0xff;
            inport(0x01,rom_rec[6]+0x30);
        }
    }
}
```

由于 PCF8563 是具有 I²C 接口的器件，如果系统中还有其他的 I²C 接口器件，则它们的操作函数是公用的，它们之间的 I/O 也是公用的，通过片选来选择不同的芯片；但不能在总线上挂接两个或两个以上的同样器件，因为每一种 I²C 器件的从机地址都是相同的，否则将会造成误操作。由于该接口简单实用，这样使得单片机系统的 I/O 口线得到节约。

第 12 章

简易 USB 接口设计

当前，USB 外设的开发是一个热点。由于 USB 总线具有传输速度快、占用资源少及真正的即插即用等优点而越来越得到业界的青睐，但 USB 的开发要求设计人员对 USB 的标准、Firmware(固件)编程及驱动程序的编写等有较深入的理解，因此限制了 USB 的开发人员。而基于 FT245BM 芯片的 USB 产品开发，能够使研发人员在最短的周期内开发出相应的 USB 产品。该芯片由 FTDI (Future Technology Devices Intl. Ltd.)公司推出，使用简单，性能卓越。只要熟悉单片机编程及简单的 VB、VC 应用程序编程，就可很容易地进行产品开发。

12.1 FT245BM 主要特点

FT245BM 主要特点如下：

- 单片 USB 与并行转换芯片；
- 在 D2XX 驱动下，最大数据传输速度为 1 Mb/s；
- 在 VCP 驱动下，最大数据传输速度为 300 Kb/s；
- 有一个 128 字节的接收缓冲区和一个 384 字节的发送缓冲区；
- 支持 USB1.1 和 USB2.0 全速；
- 可选择 5 V 和 3.3 V 两种电压，以满足不同的处理器要求。

12.2 FT245BM 芯片功能

FT245BM 的主要功能是进行 USB 和并行 I/O 口之间的协议转换。一方面它可从主机

接收 USB 数据，并将其转换为并行 I/O 口的数据流格式发送给外设；另一方面外设可通过并行 I/O 口将数据转换为 USB 数据格式传回主机。中间的转换工作全部由芯片自动完成，开发者无须考虑硬件的设计。

FT245BM 内部主要由 USB 收发器、串行接口引擎(SIE)、USB 协议引擎和先进先出(FIFO)控制器等构成，如图 12.1 所示。USB 收发器提供 USB1.1/ 2.0 的全速物理接口到 USB 总线，支持 UHCI/OHCI 主控制器；串行接口引擎主要用于完成 USB 数据的串/并双向转换，并按照 USB1.1 规范来完成 USB 数据流的位填充/位反填充以及循环冗余校验码(CRC5/CRC16)的产生和检错；USB 协议引擎主要用来管理来自 USB 设备控制端口的数据流；FIFO 控制器处理外部接口和收发缓冲区间的数据转换。

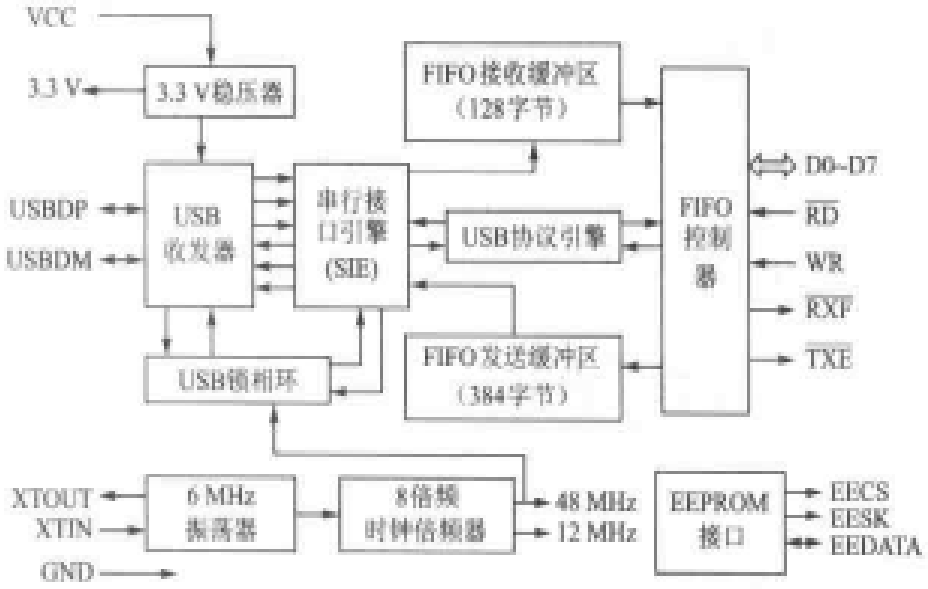


图 12.1 FT245BM 芯片功能框图

图 12.2 FT245BM 的外形

FIFO 控制器实现与单片机的接口，主要通过 8 根数据线(D0～D7)及读/写控制线(RD、WR)来完成与单片机的数据交互。FT245BM 内含两个 FIFO 数据缓冲区，一个是 128 字节的接收缓冲区，另一个是 384 字节的发送缓冲区。它们均用于 USB 数据与并行 I/O 口数据的交换缓冲区。

另外，FT245BM 还包括 1 个内置的 3.3 V 稳压器、1 个 6 MHz 的振荡器、8 倍频的时钟倍频器、USB 锁相环和 EEPROM 接口。FT245BM 采用 32 脚 PQFP 封装，体积小，易于和外设做到一块板上。其外形如图 12.2 所示。各引脚的具体功能如表 12.1 所列。

表 12.1　FT245BM 各引脚的具体功能

引脚号	引脚名称	类　型	引脚功能
25	D0	I/O	双向数据总线
24	D1	I/O	双向数据总线
23	D2	I/O	双向数据总线
22	D3	I/O	双向数据总线
21	D4	I/O	双向数据总线
20	D5	I/O	双向数据总线
19	D6	I/O	双向数据总线
18	D7	I/O	双向数据总线
16	RD	IN	由低变高时，允许从接收数据缓冲区读取数据
15	WR	IN	由高变低时，允许将数据发送到数据缓冲区
14	TXE	OUT	为低时，允许数据写入发送数据缓冲区
12	RXF	OUT	为低时，允许数据缓冲区的数据被读取
7	USBDP	I/O	USB 数据信号线 D+，接 1.5 kΩ 的上拉电阻到 3.3 V
8	USBDM	I/O	USB 数据信号线 D-
32	EECS	I/O	EEPROM 片选线
1	EESK	OUT	EEPROM 时钟线
2	EEDATA	I/O	EEPROM 数据线
10	PWREN	OUT	电源使能线
11	SI/WU	IN	发送立刻消息或唤醒信号线
1	RESET	IN	复位脚
5	RSTOUT	OUT	内部复位生成器的输出脚
27	XTIN	IN	时钟输入脚
28	XTOUT	OUT	时钟输出脚
31	TEST	IN	测试脚
6	3V3OUT	OUT	3.3 V 输出脚
3,26	VCC	PWR	+4.4～+5.25 V
13	VCCIO	PWR	控制引脚 10～12、14～16、18～25 的信号电平，+3.0～+5.25 V
9,17	GND	PWR	电源地
30	AVCC	PWR	内部模拟电源
29	AGND	PWR	内部模拟电源地

12.3 硬件设计

FT245BM 的简化典型硬件电路如图 12.3 所示。其中，时钟电路可由 6 MHz 晶振组成。引脚 PWREN 用于判断 USB 总线是处于挂起状态还是正常状态(此处悬空，因为不用考虑功耗与节能问题)。LED 主要用于指示 USB 电源。此外，在 PCB 板的设计中，数据线的走线应尽可能短且长度相等。如有可能还可以进行阻抗匹配测试，这样才能使 USB 接口的总线速度达到最大。

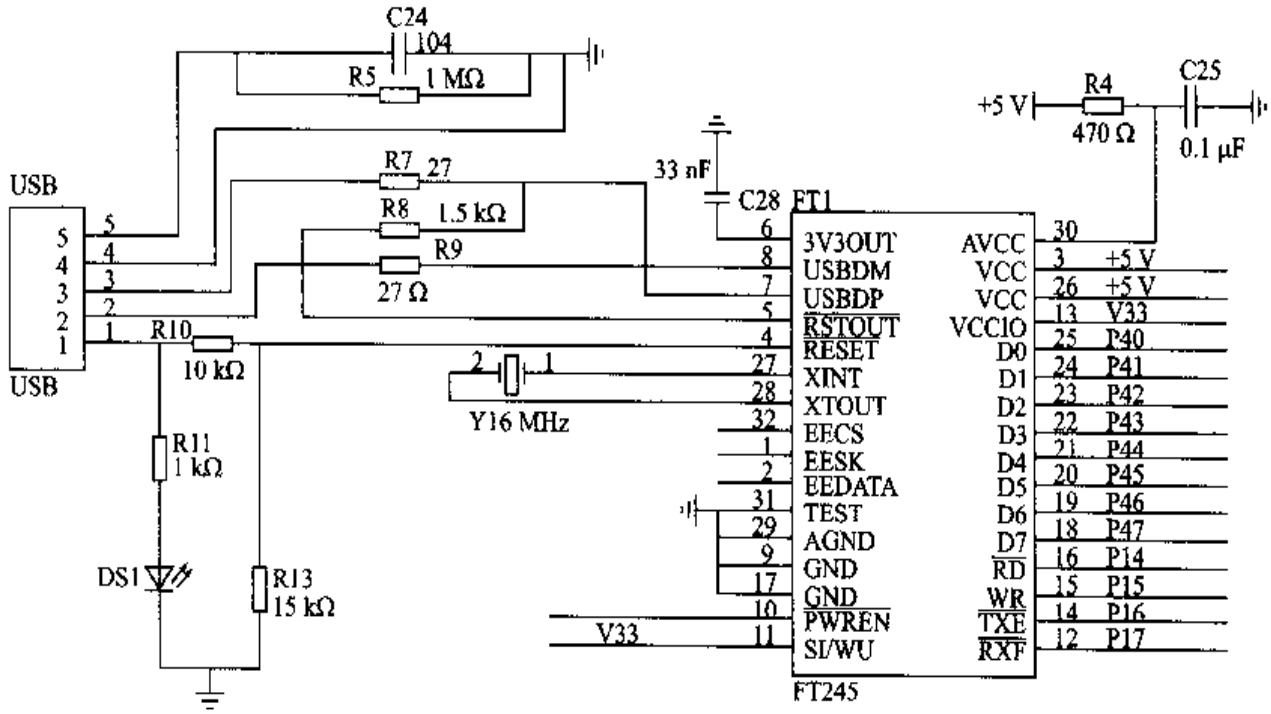


图 12.3 FT245BM 的典型硬件电路

12.4 系统软件设计

1. MSP430F149 编程思路

单片机和 USB 芯片之间的接口是并行的，它们之间的通信要满足如图 12.4 和图 12.5 所示的接收和发送数据的时序。

接收和发送数据时序图中的相关参数说明分别如表 12.2 和表 12.3 所列。

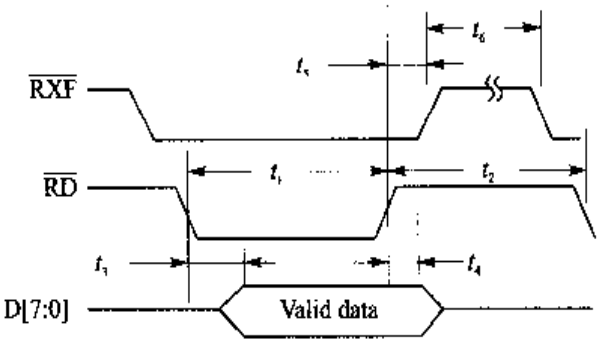


图 12.4　接收数据时序图

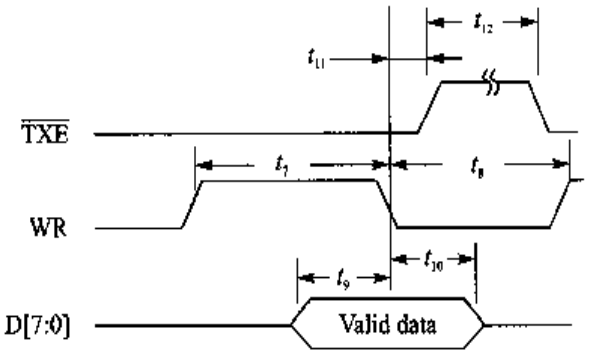


图 12.5　发送数据时序图

表 12.2　接收数据时序图中的相关参数说明

时　间	描　述	最小值/ns	最大值/ns
t_1	RD 有效脉冲宽度	50	
t_2	RD 到 RD 预转换时间	$50-t_6$	
t_3	RD 有效到有效数据	20	50
t_4	从 RD 无效有效数据保持时间	0	
t_5	RD 无效到 RXF#	0	25
t_6	RD 1 周期之后 RXF# 无效	80	

表 12.3　发送数据时序图中的相关参数说明

时　间	描　述	最小值/ns	最大值/ns
t_7	WR 有效脉冲宽度	50	
t_8	WR 到 WR 预转换时间	$50+t_{12}$	
t_9	WR 无效之前数据调整时间	20	
t_{10}	从 WR 无效数据保持时间	0	
t_{11}	WR 无效到 TXE#	5	25
t_{12}	WR 1 周期之后 TXE# 无效	80	

根据接收和发送数据的时序图，可以分析得到 MSP430F149 的编程思路。

其中，接收数据的编程思路如下：

① 采样到信号 RXF# 为低，则表明有接收来自 PC 机的数据，允许单片机通过 8 位数据总线 D0～D7 读取数据；

② 通过信号 RD# 由低到高的变化锁存数据（读入数据）；

③ 延迟一段时间后，重新开始下一字节的读取。

软件流程如图 12.6 所示。

发送数据的编程思路如下：

① 采样到信号 TXE＃为低，则表明可以向 PC 机发送数据，单片机通过 8 位数据总线 D0～D7 发送数据；

② 通过信号 WR 由高到低的变化锁存数据(读入数据)；

③ 再等到 TXE＃为低时表明一次数据的发送完成，可以进行下一次的数据发送。

软件流程图如图 12.7 所示。

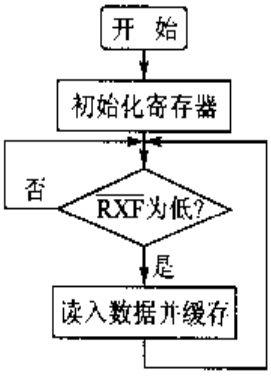


图 12.6 接收数据流程图

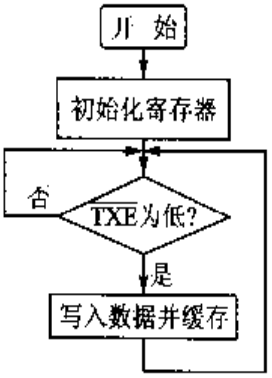


图 12.7 发送数据流程图

读者在把 FT245BM 用于其他场合时，只需参照时序图在适当的位置加入空指令即可。在接收时最好采用 RXF＝引发中断，这样保证数据不丢失。上位机的通信软件采用 VC 编写，具体内容参见配套光盘。

2. MSP430F149 具体程序代码

具体程序代码如下：

```
#include<msp430x14x.h>
#define uint unsigned int
#define uchar unsigned char
#define RXF BIT7//p1.7
#define TXF BIT6//p1.6
#define Read BIT4//p1.4
#define Write BIT5//p1.5
uchar static usb_flag = 0;
void int_clk()
{
    uchar i;
    BCSCTL1& = ~XT2OFF;                    //打开 XT2 振荡器
```

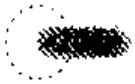
```
    BCSCTL2  = SELM1 + SELS;                    //MCLK 为 8 MHz,SMCLK 为 1 MHz
    do
    {
        IFG1 &= ~OFIFG;                         //清除振荡错误标志
        for(i = 0; i < 100; i++)
        _NOP();                                 //延时等待
    }
    while ((IFG1 & OFIFG) != 0);                //如果标志为 1,则继续循环等待
    IFG1&= ~OFIFG;
}
//写数据
void send_data(uchar data)
{
    while(P2IN&TXF == 1);                       //判断状态
    P1OUT  = Write;
    _NOP();
    _NOP();
    P4DIR = 0XFF;
    P4OUT = data;
    _NOP();
    _NOP();
    P1OUT&= ~Write;
    _NOP();
    _NOP();
}
//读数据
uchar recive_data()
{
    uchar data;
    P4DIR = 0X00;
    while(P2IN&RXF == 1);                       //判断状态
    P1OUT&= ~Read;
    _NOP();
    _NOP();
    data = P4IN;
    P1OUT|= Read;
    _NOP();
    _NOP();
    return (data);
```

```
}
void main( void )
{
    uchar temp;
    WDTCTL = WDTPW + WDTHOLD;                    //关闭看门狗
    int_clk();                                   //初始化时钟
    P1DIR  = Read + Write;
    P5DIR = 0xff;
    P1IE| = RXF;
    P1IES| = RXF;                                //I/O口功能和方向初始化
    _EINT();
    while(1)
    {
        while(usb_flag == 1)                     //判断接收数据标志,如果接收到数据则返回收数据
        {
            temp = recive_data();
            send_data( temp);
            usb_flag = 0;
        }
    }
}
//
#pragma vector = PORT1_VECTOR
__interrupt void rec_usb()
{
    P1IFG& = ~RXF;                               //清除中断标志
    usb_flag = 1;                                //接收数据标志写1
}
```

3. 上位机编程思路

FT245BM 的 USB 接口连接到主机后，必须在 PC 机上安装一个由 FTDI 公司免费提供的 D30104D2XX(Direct Drivers for 2kxp)驱动程序。该驱动程序兼容 Windows 98/ 98 SE、Windows 2000/ME/XP 等不同版本。当与主机连接成功后，在用户系统中就可以看见该设备并且能够进行应用程序的开发了。对于上位机应用程序的开发，本书配套光盘中 D2XXPG31.PDF 文档中有详细的介绍，这里只介绍编程思路。

首先，需要在 VC++工程文件中添加由 FTDI 公司免费提供的 FTD2XX.. H 和 FTD2XX.lib，然后通过调用其中的 API 函数，就可以对 FT245BM 进行状态查询、读和写等操作了。



接收 FT245BM 数据操作事件处理函数如下：

```
void CFT245Test1Dlg::ReadFT245BM()
{
    FT_STATUS Res;
    DWORD Rx;
    DWORD Tx;
    DWORD Ev;
    DWORD Br;
    DWORD i;
    char RxBuffer[256];
    CString cs;
    Res = FT_GetStatus(Usb_hWnd,&Rx,&Tx,&Ev);
    if (Res != FT_OK)
    {
        MessageBox("查询状态出错!");
        return;
    }
    if (Rx > 0)
    {
        Res = FT_Read(Usb_hWnd,RxBuffer,Rx,&Br);
        if (Res != FT_OK)
        {
            MessageBox("接收出错!");
            return;
        }
        for (i = 0;i < Rx;i++)
        {
            cs.Format("%d",RxBuffer[i]);
            m_Rdata = m_Rdata + cs + "\r\n";
        }
        UpdateData(FALSE);
    }
}
```

发送数据到 FT245BM 操作事件处理函数如下：

```
void CFT245Test1Dlg::SendFT245BM()
{
    DWORD C;
    BOOL b;
```

```
    FT_STATUS Res;
    b = UpdateData();
    if (b != 0)
    {
        Res = FT_Write(Usb_hWnd,&m_Value,1,&C);
        if (Res != FT_OK)
        {
            MessageBox("发送数据错误!");
        }
    }
}
```

利用 FT245BM 进行 USB 接口开发，简单且周期短，易于为工程实践所采用；同时，也可以直接用于新产品的开发，如数码相机、MP3 播放器及无线 Modem 等的 USB 接口。

第 13 章

任意信号发生器的设计

在现代电子系统设计过程中，信号发生器具有不可替代的作用，它可以提供调试、检测、调制等生产、科研、教学之用。由于现在市场上信号发生器价格较高，初学者难以承受。对于一般的应用，作者认为自己设计一个信号发生器是相当有价值的。传统的模拟信号发生器信号单一，而且调试不方便，因此这里我们不采用。本例中主要采用 TLC5615 结合 MSP430F149 单片机设计一个任意信号发生器，我们只需要改变信号波形数据就可以产生任意信号，同时 TLC5615 已经在工程应用中应用得比较广泛了，读者也可以把它应用在自己的系统中。

TLC5615 的基本特点如下：

- DAC 分辨率 10 位(配套实验板可以兼容 12 位 DAC TLV5618)；
- 输出信号幅度：0~5 V 可调；
- 输出信号类型：正弦波、三角波、方波以及其他自定义波形。

13.1 芯片概述

TLC5615 是 TI 公司设计的串行 10 位 D/A 转换芯片，性能比早期电流型输出的 DAC 要好。它只需要通过 3 根总线就可以完成 10 位数据的串行输入，易于与工业标准的微处理器或微控制器(单片机)接口，适用于电池供电的测试仪表、移动电话，也适用于数字失调与增益调整以及工业控制场合。其主要特点如下：

- 单 5 V 电源工作；
- 3 线串行接口；
- 高阻抗输入端；
- DAC 输出的最大电压为基准输入电压的 2 倍；

- 上电时内部自动复位；
- 微功耗：最大为 1.75 mW；
- 转换速率快：更新率为 1.21 MHz；
- 建立时间典型值为 12.5 μs。

13.2 内部功能框图

TLC5615 内部框图如图 13.1 所示。它主要由以下几部分组成：

- 10 位数/模转换电路；
- 一个 16 位移位寄存器用于接收串行移位进入的二进制数，一个级联的数据输出端 DOUT 用于扩展；
- 并行输入/输出的 10 位 DAC 寄存器，为 10 位 DAC 电路提供待转换的二进制数据；
- 电压跟随器为参考电压端 REFIN 提供很高的输入阻抗，大约 10 MΩ；
- ×2 电路提供最大值为 2 倍于 REFIN 的输出；
- 上电复位电路和控制电路。

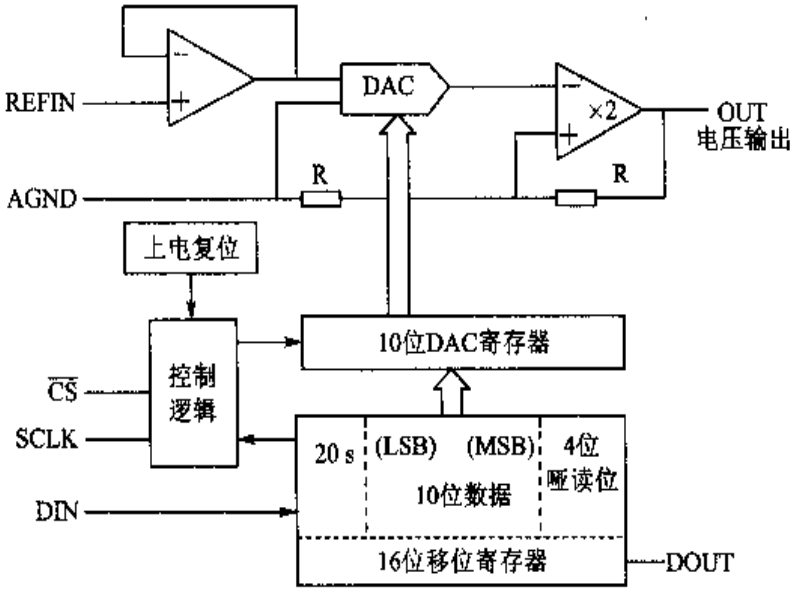


图 13.1 TL C5615 的内部功能框图

13.3 引脚描述

TLC5615 是一个 8 引脚的 DAC，其引脚分布如图 13.2 所示。表 13.1 为 TLC5615 的引脚功能描述，使用前须仔细了解。

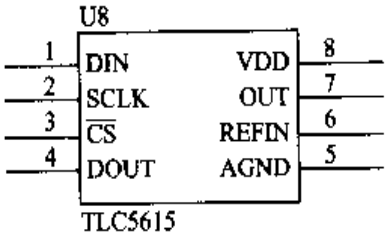


图 13.2　TLC5615 引脚封装图

表 13.1　TLC5615 引脚功能

引脚号	引脚名称	功　能
1	DIN	串行二进制数输入端
2	SCLK	串行时钟输入端
3	$\overline{CS}$	芯片选择，低有效
4	DOUT	用于级联的串行数据输出
5	GND	模拟地
6	REFIN	基准电压输入端
7	OUT	模拟电压输出端
8	VDD	正电源电压端

13.4　TLC5615 的工作时序

TLC5615 的工作时序如图 13.3 所示。可以看出，只有当片选信号$\overline{CS}$为低电平时，串行输入数据才能被移入 16 位移位寄存器。当$\overline{CS}$为低电平时，在每一个 SCLK 时钟的上升沿将 DIN 的一位数据移入 16 位移位寄存器。二进制数的最高有效位先传输，接着$\overline{CS}$的上升沿将 16 位移位寄存器的 10 位有效数据锁存于 10 位 DAC 寄存器中，供 DAC 电路进行转换。当片选信号$\overline{CS}$为高电平时，串行输入数据不能被移入 16 位移位寄存器。

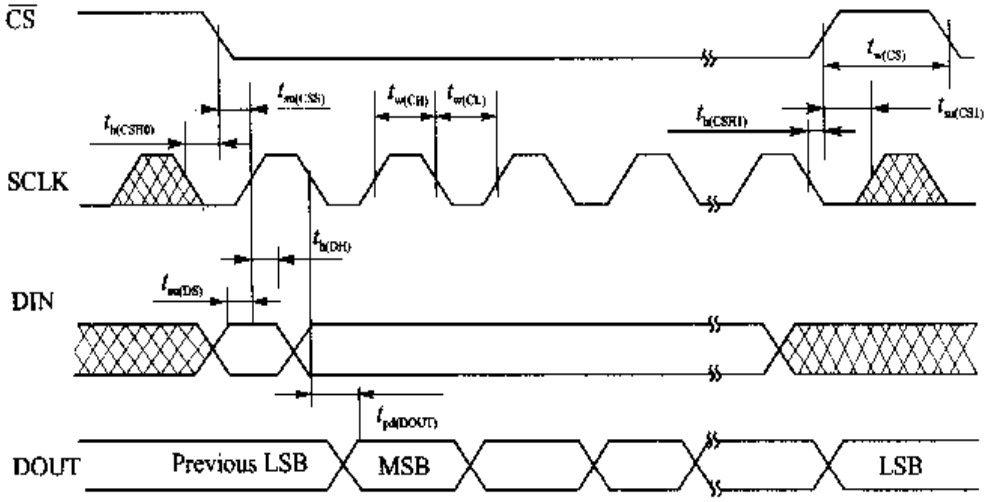


图 13.3　TLC5615 的时序图

从图中可以看出，最大串行时钟速率为：

$$f_{(sclk)max} = 1/\ t_{w(CH)} + t_{w(CL)} \approx 14\ \text{MHz}$$

串行接口说明：

当片选信号($\overline{CS}$)为低电平时，串行数据读入 16 位的移位寄存器中，高位在前，低位在后，并在 SCLK 的上升沿把数据移入输入寄存器。在$\overline{CS}$的上升沿把串行移入的数据传送到 DAC 寄存器。当$\overline{CS}$为高电平时，输入数据不能移入到输入寄存器，并且$\overline{CS}$电平的变化必须在 SCLK 为低时才能进行。16 位移位寄存器分为高 4 位虚拟位、低 2 位填充位以及 10 位有效位。

由于 TLC5615 内部自带 2 倍放大电路，输出电压为：

$$V_{OUT} = 2V_{REFIN} \times N/1\ 024$$

其中，V_{REFIN}是参考电压，N 为输入的二进制数。

13.5 系统电路设计

图 13.4 所示为 TLC5615 的硬件连接图。外接参考电压采用 TL431。输出电路采用 RC 低通滤波，滤除 DAC 本身的高频干扰。TLC5615 与 CPU 的接口为 3 线模式的 SPI 接口。

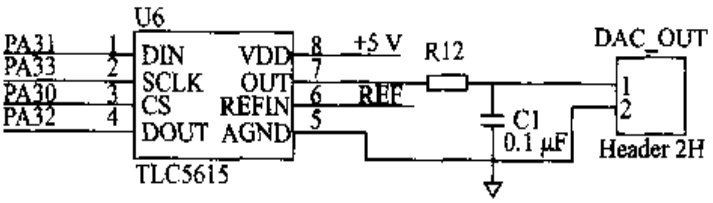


图 13.4 硬件连接图

13.6 系统程序设计思想

信号发生器的程序设计主程序流程和定时器中断子程序流程分别如图 13.5 和图 13.6 所示。它主要是通过单片机把预先准备好的信号数据表通过 DAC 还原输出，同时可以根据给定时间间隔来控制信号频率。本例中我们将产生一个正弦信号。读者可以更换数据表中的数据来产生其他类型的信号。在实验板上，读者可以根据键盘修改信号类型和频率。这里只给出产生正弦波的程序。

具体程序设计如下：

```
#include <msp430x14x.h>
#define uchar unsigned char
#define uint unsigned int
```

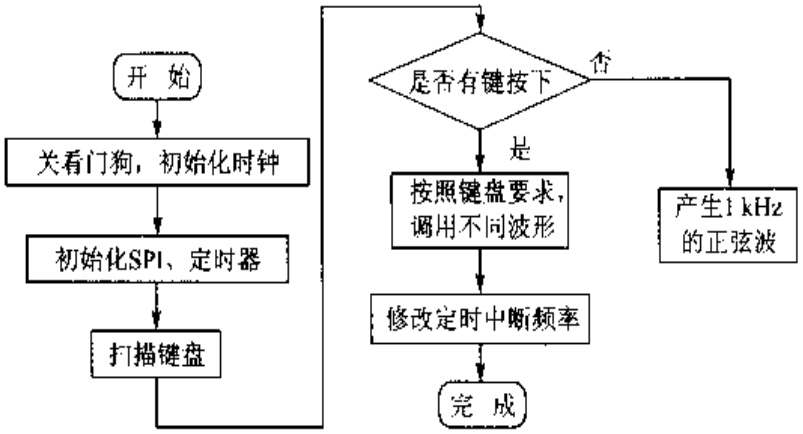


图 13.5　主程序流程图

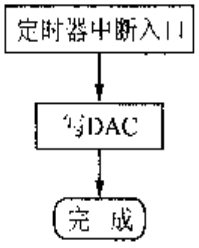


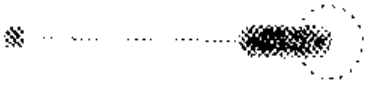
图 13.6　中断子程序流程图

```
#define spi_cs BIT5
#define da_cs BIT6
#define Enable P2OUT&=~spi_cs;
#define Disable P2OUT|=spi_cs;
uint temp=0,temp1=0;
uint aa[38];
float
str[37]={1.00,1.173,1.342,1.500,1.643,1.766,1.87,1.94,1.984,2.00,1.984,1.94,1.866,1.
766,1.643,1.50,1.342,1.173,1.00,0.826,0.658,0.5,0.357,0.234,0.134,0.060,0.015,0.00,0.
015,0.060,0.134,0.234,0.357,0.500,0.658,0.826,1.00};
float
sanjiao[37]={0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,0.9,0.8,0.7,0.6,0.5,0.4,0.3,0.
2,0.1};
float fangbo[21]={0.0,1.0};
void int_clk()
{
    uchar i;
    BCSCTL1&=~XT2OFF;                    //打开 XT2 振荡器
    BCSCTL2|=SELM1+SELS;                 //MCLK 为 8 MHz,SMCLK 为 1 MHz
    do
    {
        IFG1&=~OFIFG;                    //清除振荡错误标志
        for(i=0;i<100;i++)
        _NOP();                          //延时等待
    }
    while((IFG1&OFIFG)!=0);              //如果标志为 1,则继续循环等待
    IFG1&=~OFIFG;
}
```

```
void int_spi()
{
    P3SEL  = 0x0E;
    P2DIR  = spi_cs;
    Disable;                                    // 不使能器件选择
    U0CTL  = SWRST;
    U0CTL  = CHAR + SYNC + MM;
    U0TCTL  = SSEL1 + SSEL0 + STC;              // 3 线工作模式,SMCLK 作为 SPI 的时钟
    U0TCTL  = CKPH;
    U0TCTL& = ~CKPL;
    U0BR0 = 0X02;
    U0BR1 = 0X00;
    U0MCTL = 0X00;
    ME1  = USPIE0;
    U0CTL& = ~SWRST;
    IE1& = ~UTXIE0;
    IE1& = ~URXIE0;
    P3DIR  = BIT2 + BIT4;
}
write_byte(uchar byte)                          //通过 SPI 接口送出一个字节数据
{
    U0TXBUF = byte;
    while((IFG1&UTXIFG0) = = 0);
    IFG1& = ~UTXIFG0;
}
delay()
{
    uint i;
    for(i = 0;i <= 10000;i ++ )
        ;
}
void DAC_OUT(uint temp)
{
    temp = temp << 2;
    Enable;
    write_byte((temp >> 8)&0x0f);
    write_byte(temp);
    Disable;
}
```

```
void main( )
{
    uchar i = 0;
    WDTCTL = WDTPW + WDTHOLD;
    int_clk();                              //初始化时钟
    int_spi();
    for(i = 0;i<37;i++)
    {
        aa[i] = (str[i] + 1) * 204.8;
    }                                       //计算波形数据
    while(1)
    {
        for(i = 0;i<37;i++)
        {
            temp = aa[i] << 2;
            Enable;
            write_byte((temp >> 8)&0x0f);
            write_byte(temp);
            Disable;                        //连续写数据
        }
    }
}
```

本实例主要是通过 TLC5615 设计一个任意信号发生器，系统只是简单给出一些常见波形的信号数据。如果读者需要，可以任意修改波形数据达到设计要求。通过上面例子的介绍，读者可以将 TLC5615 应用到一个实际的系统当中去。如果要产生多种波形的发生器，简单的做法是把待输出波形码的周期值提取出来存放在一个数组中，再通过 DAC 输出即可。读者可以在本实验的基础上加以修改，以得到不同频率的波形。



附录 A

MSP430F149 单片机实验板

MSP430F149 单片机实验板是为广大单片机爱好者设计的，资源丰富，包括 Flash、DAC、ADC、RS485、RS232、USB、KEY、LED、LCD 及 MSP430F149 单片机自带的一些外围模块，不仅可以进行 MSP430F149 单片机整体资源特点验证性实验，而且可以进行多个设计性实验（比如数字电压表、频率计、信号发生器、RS485 多机通信等），非常适合与本书配套使用。同时也可以满足工程技术人员的学习要求，通过实验板的学习，可以完全掌握 MSP430 单片机的基本应用，能极大地缩短工程开发周期，是理论和实际完全结合的、集入门和提高于一体的功能性学习板。下面针对本书的配套实验板的特色加以说明。

基本功能特点

- 实验板配套简易并口 JTAG 调试器，用户可以直接使用，而无需重新购置仿真器；
- 2 路 12 位 ADC 输入和 1 路 10 位 DAC 输出（可以选择 2 路 12 位 DAC），满足一般的应用要求；
- 1 路 PWM 输出和 1 路 CAP 输入，方便读者理解 MSP430F149 单片机的内部比较和捕捉的使用方法；
- 标准 RS485 接口，可以进行 RS485 通信实验，多个实验板可组成 RS485 网络；
- 标准 RS232 接口，可以进行 RS232 通信实验；
- 液晶显示器采用 FYD128 * 64；
- 外接 4×4 键盘，满足大部分实际应用需求；
- USB 接口，方便与计算机进行数据交换；
- 8M 位 SPI 接口 Flash，可以满足大部分应用要求（最大可扩展到 128M 位）；
- RTC 时钟带备用电池，保证掉电后时钟继续有效；

- 4 个 LED 显示器，1 个蜂鸣器；
- 2 路模拟比较器信号输入；
- 4 位 LED 数码显示器；
- 高稳定性，模拟信号采集精度高于 0.2%；
- 所有程序全部测试通过；
- 可以采用 USB 或者外部 9～12 V 电源供电。

硬件原理

MSP430F149 单片机实验板电路原理图请参考附录 B。实验板结构如图 A.1 所示。

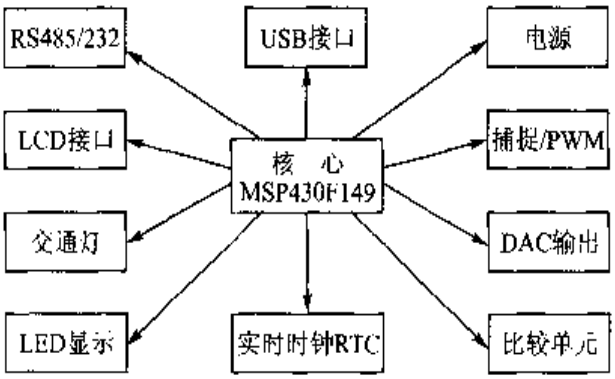


图 A.1　实验板结构

硬件资源

- CPU 采用 TI 公司的低功耗单片机 MSP430F149；
- LCD 采用 FYD12864(带字库)，板上留接口(控制器为 ST7920)；
- 4×4 键盘，板上留有接口；
- 8 Mb 串行 Flash M25P80，可以扩展到 128 Mb；
- 1 路(2 路)DAC TLV5615，或 TLV5618(可选)；
- 2 路 12 位 ADC，可以通过板上自带电位器调节，也可以接外部输入；
- RS485 通信接口；
- RS232 接口；
- USB2.0 接口；
- 4 位 LED 显示器；
- RTC PCF8563 带备用电池；
- 4 个发光二极管；
- 1 个蜂鸣器；

- 2 路模拟比较器信号输入；
- CAP 脉冲测量接口；
- PWM 信号输出接口。

配套提供的 MSP430F149 实验板可以满足大多数读者的学习需要。下面列举一些具有针对性的实验内容供读者参考。实验内容分成基础实验和设计性实验两部分。

基础实验

实验一　内部 Flash 操作

实验目的：掌握 MSP430F149 单片机内部信息 Flash 的基本操作流程，能够正确使用 MSP430F149 单片机内部的 Flash。

实验设备：实验板采用本书配套实验板及 PC 机一台。

实验内容：对内部信息 Flash 的读/写，以及掉电保存系统设置参数。读者可以根据要求编写适合自己系统的程序。

参考资料和参考程序见本书各模块举例，同时配套光盘中也有相应的程序。

MSP430F149 单片机内部用于 258 字节的信息 Flash，该 Flash 可以在使用户系统中保存。

实验二　时钟模块实验

实验目的：掌握 MSP430F149 单片机内部时钟模块的基本操作，根据具体需要选择不同的时钟，以达到低功耗的目的。

实验设备：实验板采用本书配套实验板及 PC 机一台。

实验内容：能够选择和调试 ACLK、MCLK、SMCLK 时钟源和频率。

参考资料和参考程序见本书各模块举例，同时配套光盘中也有相应的程序。

实验三　基本输入/输出实验

实验目的：掌握 MSP430F149 单片机输入/输出口的基本使用方法，在配套实验板上有很多资源可以做输入/输出口实验，比如 LED 灯、蜂鸣器等可供选择。

实验设备：实验板采用本书配套实验板及 PC 机一台。

实验内容：能够控制输出电平高低，读取输入电平。

参考资料和参考程序见本书各模块举例，同时配套光盘中也有相应的程序。

实验四　中断实验

实验目的：掌握 MSP430F149 单片机中断响应的基本流程，能够编写各外设模块的中断服务程序。

实验设备：实验板采用本书配套实验板及 PC 机一台。

实验内容：实验板提供的很多外围模块都具有中断功能，读者可以根据时间情况选择。

参考资料和参考程序见本书各模块举例，同时配套光盘中也有相应的程序。

实验五　RS232 通信实验

实验目的：掌握 RS232 通信基本原理以及 RS232 接口的电平转换的基本方法，能够熟练应用 MSP430F149 单片机内部通用串行接口与 PC 机进行通信，能够使用 VC++控件编写简单的程序。

实验设备：实验板采用本书配套实验板，PC 机一台，串口线一根。

实验内容：接收计算机发送的数据给单片机，同时不数据返回给计算机，读者可以根据需要编写其他程序。

参考资料和参考程序见本书各模块举例，同时配套光盘中也有相应的程序。

实验六　RS485 通信实验

实验目的：掌握 RS485 通信的基本方法和原理，熟练掌握 RS485 接口电路设计，MSP430F149 单片机内部通用串口，通过 RS485 接口和上机通信。

实验设备：实验板采用本书配套实验板及 PC 机一台。

实验内容：能够选择和调试 ACLK MCLK SMCLK 时钟源和频率。

参考资料和参考程序见本书各模块举例，同时配套光盘中也有相应的程序。

实验七　LCD 显示实验

实验目的：了解 LCD 显示器基本原理，熟练掌握 FYD12864 与 MSP430F149 单片机接口电路设计，能够进行 LCD 驱动程序设计。

实验设备：实验板采用本书配套实验板及 PC 机一台。

实验内容：在 LCD 的不同位置显示字符和汉字等。

参考资料和参考程序见本书各模块举例，同时配套光盘中也有相应的程序。

实验八　PWM 波产生实验

实验目的：了解 PWM 产生基本原理，熟练掌握 MSP430F149 单片机 PWM 信号产生的基本原理，能够熟练掌握 MSP430F149 单片机产生 PWM 的程序设计。

实验设备：实验板采用本书配套实验板及 PC 机一台，示波器一台。

实验内容：修改 PWM 信号的周期和占空比，并通过示波器观察 PWM 信号。

参考资料和参考程序见本书各模块举例，同时配套光盘中也有相应的程序。

实验九　LED 数码管显示实验

实验目的：了解 LED 显示基本原理，熟练掌握 MSP430F149 单片机的 SPI 基本程序设计，熟练掌握显示电路设计。

实验设备：实验板采用本书配套实验板及 PC 机一台。

实验内容：通过 SPI 向 74HC164 写不同数据，并且观察显示内容。

参考资料和参考程序见本书各模块举例，同时配套光盘中也有相应的程序。

实验十　SPI 扩展串行 Flash 扩展实验

实验目的：了解串行 Flash 基本原理，掌握 SPI 总线基本原理，熟练掌握 MSP430F149 单片机的 SPI 接口控制 M25P80 基本电路设计和程序设计。

实验设备：实验板采用本书配套实验板及 PC 机一台。

实验内容：写数据到 M25P80 中，并且读出来验证。

参考资料和参考程序见本书各模块举例，同时配套光盘中也有相应的程序。

实验十一　USB 与 PC 机通信实验

实验目的：了解 USB 通信基本原理，掌握 FT245 的基本工作原理，能够熟练掌握 MSP430F149 单片机与 FT245 电路设计和程序设计，能够进行简单的 PC 机程序设计。

实验设备：实验板采用本书配套实验板及 PC 机一台。

实验内容：接收 PC 机发送的数据，同时发送数据给 PC 机。

参考资料和参考程序见本书各模块举例，同时配套光盘中也有相应的程序。

实验十二　KEY 扫描实验

实验目的：掌握扫描式键盘的基本原理和电路设计，熟练掌握 MSP430F149 单片机 I/O 口的基本使用方法。

实验设备：实验板采用本书配套实验板及 PC 机一台。

实验内容：扫描键盘，并且在 LED 上显示。

参考资料和参考程序见本书各模块举例，同时配套光盘中也有相应的程序。

实验十三　定时器实验

实验目的：掌握定时器的基本原理，熟练掌握 MSP430F149 单片机 Timer_A 和 Timer_B 的基本工作原理和程序设计。

实验设备：实验板采用本书配套实验板及 PC 机一台。

实验内容：通过定时器产生一个秒脉冲，控制发光二极管的闪烁。

参考资料和参考程序见本书各模块举例，同时配套光盘中也有相应的程序。

设计性实验

设计性实验主要是在读者掌握 MSP430 单片机的基础知识后，结合其他学科知识设计出一些满足一般工程需要。具有一定使用价值的电子产品。它不仅是对基本知识的很好的复习，同时也是一个提高。本实验板还可以结合板上资源设计出其他功能的电子产品。这里举出其中比较有代表性的例子以供读者参考。

实验一　基于 PCF856 的数字时钟设计

实验目的：了解数字钟基本工作原理，熟练掌握 I^2C 程序设计和 MSP430F149 单片机与 PCF8563 的程序设计。

实验设备：实验板采用本书配套实验板及 PC 机一台。

实验内容：修改、读取和显示时间信息并且可以设置报警功能。

参考资料和参考程序见本书各模块举例，同时配套光盘中也有相应的程序。

实验二　数字频率计

实验目的：了解频率计的主要用途，掌握频率测量的基本方法和各种方法的优缺点，熟练

掌握使用 MSP430F149 的 CAP 功能进行频率测量的基本方法，理解频率计电路设计注意事项和误差分析。

实验设备：实验板采用本书配套实验板及 PC 机一台。

实验内容：对整形后的脉冲信号频率进行测量，同时测量占空比，并且通过板上 LED 或 LCD 显示出来。

参考资料和参考程序见本书各模块举例，同时配套光盘中也有相应的程序。

实验三　任意信号发生器

实验目的：了解信号发生器的主要用途，掌握采用 DAC 产生任意信号的基本方法，比较不同方法的优点和缺点，熟练掌握使用 MSP430F149 的 SPI 功能进行 DAC 扩展。

实验设备：实验板采用本书配套实验板 PC 机一台。

实验内容：根据需要更新信号表格产生不同信号，同时控制信号频率。

参考资料和参考程序见本书各模块举例，同时配套光盘中也有相应的程序。

实验四　数字电压表

实验目的：了解电压表的主要用途，掌握电压测量的基本方法，比较不同方法的优缺点，熟练掌握使用 MSP430F149 的 ADC 进行电压信号采集基本方法，了解 ADC 采集抗干扰的主要措施和原理，熟悉 A/D 转换器基本指标和衡量标准。

实验设备：实验板采用本书配套实验板及 PC 机一台。

实验内容：通过电位器调节 ADC0 输入电压，采集该电压并且通过显示 LED 或 LCD 显示出来。

参考资料和参考程序见本书各模块举例，同时配套光盘中也有相应的程序。

实验五　基于 MSP430 单片机的多机通信

实验目的：了解 RS485 通信的主要用途和优点，掌握 RS485 通信的基本原理，掌握 MSP430F149 单片机多机通信的基本原理和方法。能够熟练进行相关电路和程序设计。

实验设备：实验板采用本书配套实验板及 PC 机一台。

实验内容：详细参考第 8 章关于 485 多机通信介绍，也可以按照自己要求编写程序。

参考资料和参考程序见本书各模块举例，同时配套光盘中也有相应的程序。

实验板硬件资源分配如下：

(1) 4×4 键盘：占用资源为 P5 口，高 4 位接上拉电阻。

(2) LCD 显示器:LCD_CS 接 P62;

LCD_SID 接 P64;

LCD_SCLK 接 P63。

(3) RS485:占用 MSP430F149 单片机的串口 1,P2.0 控制方向。

(4) RS232:占用 MSP430F149 单片机的串口 0。

(5) USBFT245:数据线采用 P4,控制和状态分别接至 P14、P15、P16、P17。

(6) ADC 输入:占用 P60、P61。

(7) PWM 输出:占用 P12。

(8) CAP 输入:占用 P12。

(9) DACTLC5615:占用 SPI 接口 0,片选信号接至 P25。

(10) LED 数码管:占用 SPI 接口 0,片选信号接至 P27。

(11) LED 灯:占用 P67、P66、P65、P30。

(12) 蜂鸣器:占用 P22。

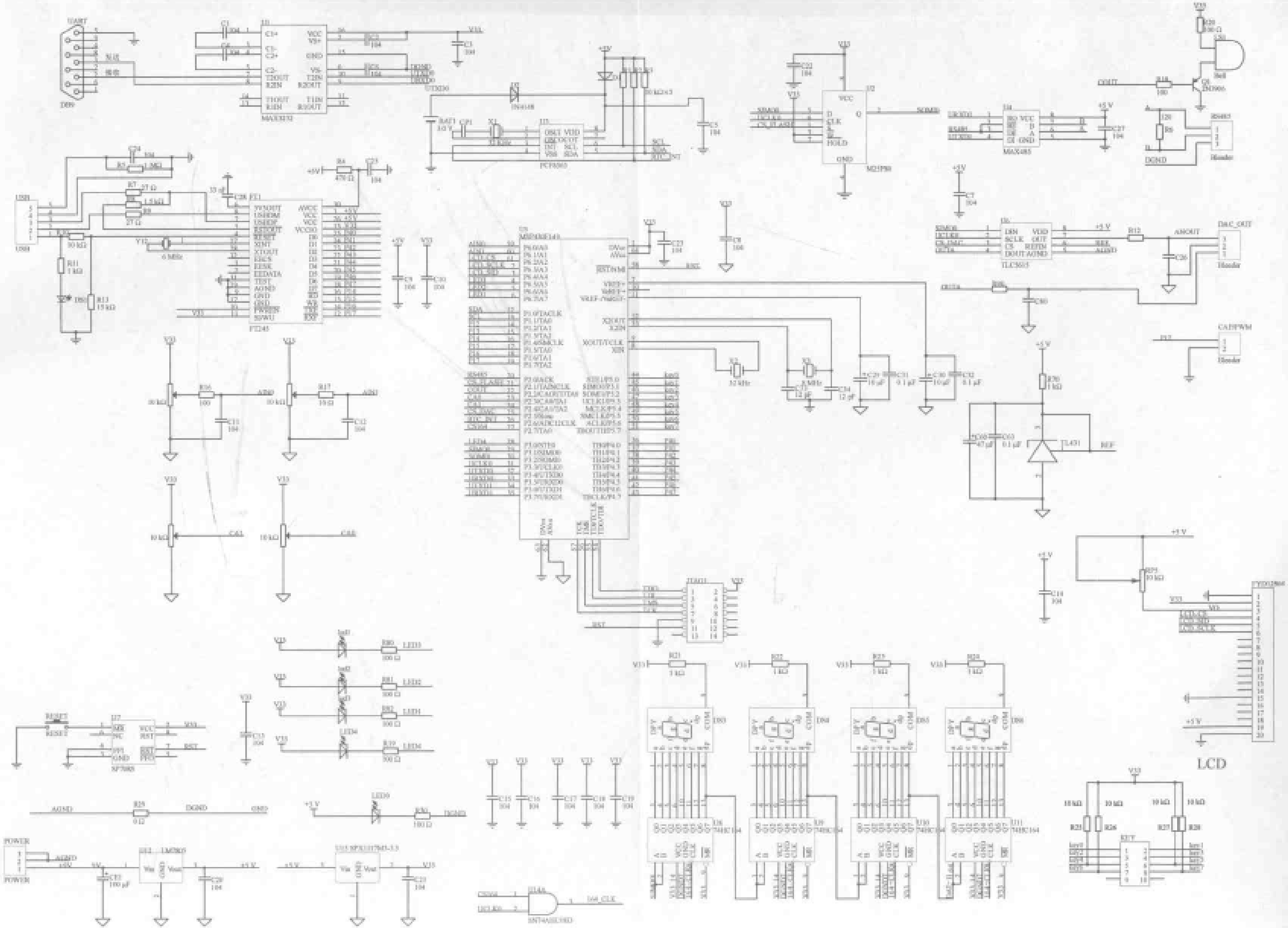
(13) PCF8563 时钟芯片:SDA 接至 P10、SCL 接至 P11,RTC_INT 接至 P26。

(14) M25P80 FLASH 芯片:SPI 接口 0,片选信号接至 P21。

附录 B

实验板电路图

实验板电路图如图 B.1 所示。


MSP430F149
MAX3232
FT245
PCF8563
M25P80
MAX485
TLC5615
TL431
74HC164
LCD

图B.1 实验板电路图

参考文献

[1] Slau 049. pdf Texas Instruments incorporated,2006.

[2] MSP430x13x, MSP430x14x, MSP430x14x1 MIXED SIGNAL MICROCONTROLLER.

[3] 周立功,夏宇闻. 单片机与 CPLD 综合应用技术[M]. 北京:北京航空航天大学出版社,2003.

[4] 胡大可. MSP430 系列超低功耗 16 位单片机原理与应用[M]. 北京:北京航空航天大学出版社,2000.

[5] 秦龙. MSP430 单片机应用系统开发典型实例[M]. 北京:电子工业出版社,2005.

[6] 秦龙. MSP430 单片机 C 语言应用程序设计实例精讲[M]. 北京:电子工业出版社,2006.

[7] 何为民. 低功耗单片微机系统设计[M]. 北京:北京航空航天大学出版社,1994.

[8] 魏小龙. MSP430 系列单片机接口技术及系统设计实例[M]. 北京:北京航空航天大学出版社,2002.

[9] 胡大可. MSP430 系列单片机 C 语言程序设计与开发[M]. 北京:北京航空航天大学出版社,2002.

[10] IAR Company, MSP430 Windows WorkBench, TEXAS INSTRUMENTS. 1999 72~83.

[11] 胡大可. MSP430 系列 FLASH 型超低功耗 16 位单片机[M]. 北京:北京航空航天大学出版社,2001.

[12] 杨小川. Protel DXP 设计指导教程[M]. 北京:清华大学出版社,2003.

[13] 薛钧义,张彦斌. 凌阳 16 位单片机原理及应用[M]. 北京:北京航空航天大学出版社,2003.

[14] 周启海. C 语言程序设计教程[M]. 北京:机械工业出版社,2004.

[15] 张红兵等. 利用 FT8U232AM 实现的 USB/RS232 转换器[J]. 国外电子元器件,2002(5).

[16] 范逸之. Visual Basic 与 RS232 串行通信控制[M]. 北京:中国青年出版社,2002.

[17] Future Technology Devices Intl Ltd. FT245BM Data Sheet. www. ftdichip. com. 2002.

[18] 李信. 16 位微型计算机原理与接口[M]. 天津:南开大学出版社,1995.

[19] 李宏,张家田 . 液晶显示器件应用技术[M]. 北京:机械工业出版社,2004.

[20] 李维提，郭强. 液晶显示应用技术[M]. 北京：电子工业出版社，2000.
[21] 张宏. USB 接口设计[M]. 西安：西安电子科技大学出版社，2002.
[22] 周坚. 单片机 C 语言轻松入门[M]. 北京：北京航空航天大学出版社，2006.
[23] 李广军，王厚军. 实用接口技术[M]. 成都：电子科技大学出版社，2001.
[24] 曹琳琳，曹巧媛. 单片机原理及接口技术[M]. 长沙：国防科技大学出版社，2000.
[25] 张晞，王德银，张晨. MSP430 系列单片机实用 C 语言程序设计[M]. 北京：人民邮电出版社，2005.
[26] 沈建华. MSP430 系列 16 位超低功耗单片机实践与系统设计[M]. 北京：清华大学出版社，2005.
[27] 沈建华，扬艳琴，翟骁曙. MSP430 系列 16 位超低功耗单片机实践与系统[M]. 北京：清华大学出版社，2004.
[28] http://www.altera.com
[29] http://www.21ic.com
[30] http://www.datasheet.com.cn
[31] http://www.icbase.com
[32] http://baike.baidu.com/view/215429.htm
[33] http://www.dldz.com/info/mcu/2006/1020/article_1131_1.html