《嵌入式编程》实验指导书

—— 基于 S12 MCU 应用开发平台、
CodeWarrior4.6 编译器及实时操作系统 UC/OS-II

安阳工学院飞思卡尔实验室

2009 年 4 月

引　言

在当前数字信息技术和网络技术高速发展的后PC（Post-PC）时代，随着国内外各种嵌入式产品的进一步开发和推广，形式多样的数字化智能产品应运而生，嵌入式技术越来越和人们的生活紧密结合，远远超过了通用PC机的应用领域[1]。嵌入式系统带来的工业年产值超过一万亿美元，在IT产业中占有很大的比重[2]。

嵌入式系统（Embedded System）通常被定义为：以应用为中心，以计算机技术为基础，软件硬件可裁剪，适用对功能、可靠性、成本、体积、功耗严格要求的专用计算机系统。嵌入式系统的关键技术包括以应用为中心的硬件平台设计和面向应用的操作系统及软件产品的开发[3]。

广义上讲，凡是带有微处理器的专用软硬件系统都可称为嵌入式系统，它的核心部件是嵌入式微处理器，该领域知名的厂商包括瑞萨、Freescale、ST、TI、Infineon、Atmel、NEC[2]。其中Freescale 公司的MCU虽然进入国内较晚，但其已享誉国际，其MC9S08 系列MCU低价位、高性能 8Bit MCU是Freescale公司未来主推的产品，其主要面向中低端嵌入式应用[4]。但由于它们没有操作系统，管理系统硬件和软件的能力有限，在实现复杂多任务功能时，往往困难重重，甚至无法实现。尤其对嵌入式工程师的开发设计带来很大的难度。

面对如上问题，我们应采取一种开发模式来解决这些问题[5]。嵌入式应用开发需要功能强大的开发工具，包括编译器、调试器、写入器等等。主流的开发工具往往将这几项功能集成在一起，并与一个简单的开发硬件相配合，例如CodeWarrior™ for S08 和S08BDM调试器，其功能十分强大，但是，开发人员在进行应用开发时仍然要经历：关键硬件电路测试、硬件原型设计、软件设计、软硬件调试等过程，该过程会耗费工程师的大量时间。如果能开发一套软硬件系统，既能够支持硬件电路测试和硬件原型设计，又能提供可重用软件资源，同时也支持软硬件调试，将极大的提高工程师的开发效率和开发质量。

目 录

第1章　概述

1.1　嵌入式发展史即现状

嵌入式系统的核心是微控制器（MCU）, MCU 是嵌入式计算机的主要形态，应用最广。1971 年 11 月 Intel 公司首次宣布 4004 的 4 位微处理器，1974 年 12 月 Fairchild(仙童)公司推出了 8 位 MCU F8，开创了 MCU 的初级阶段。1976 年 Intel 公司推出了 MCS-48 MCU，它极大地促进了 MCU 的变革。1980 年，Intel 公司在 MCS-48 的基础上，推出了 MCS-51，它使 MCU 的应用跃上了一个新台阶。2002 年摩托罗拉公司（2004 年 6 月更名为 Freescale 半导体公司）的 8 位 MCU、16 位 MCU、32 位 MCU 并行发展，增加了市场份额，方便用户的选型。根据 Dataquest 的资料显示，Motorola 公司的 MCU 在国际市场上占有最大份额。

据调查，目前国际上已有两百多种嵌入式操作系统，而各种各样的开发工具、应用于嵌入式开发的仪器设备更是不可胜数。在国内，虽然嵌入式应用、开发很广，但该领域却几乎还是空白，只有三两家公司和极少数人员在从事这方面工作。随着计算机技术及集成电路技术的发展，嵌入式技术日渐普及，在通讯、网络、工控、医疗、电子等领域发挥着越来越重要的作用。我国嵌入式系统产业的人才需求量也一路高涨，嵌入式开发将成为未来几年最热门最受欢迎的职业之一。另据权威部门统计，我国目前嵌入式软件人才缺口每年为 20 万人左右。

1.2　嵌入式产品的开发流程

嵌入式产品的开发应遵循软件工程的原则，要清晰、合理地放置您的工作资料。对于一些中小型嵌入式产品或应用系统的开发过程，在进行基本需求与功能分析之后，可以进入设计阶段。这里对设计阶段开始后的工作过程，提出一些建议和步骤，供参考：

第一步，根据需求分析所需的离散量 IO 和开关量 IO，以功能模块内聚化为主导，选择合适的 MCU，以 MCU 系统为核心，分析 I/O 该 MCU 的 IO 是否匹配；其中需注意以下几点：

（1)选择功能模块内聚化的 MCU 不仅减少了外部芯片数量,降低开发成本和系统功耗，还大大提高了系统的稳定性。

（2）选择 MCU 时要考虑的因素有：处理性能、功耗、价格、封装形式、软硬件开发工具、设计者的熟悉程度等。一个 MCU 的性能取决于多个因素，如时钟频率、内部寄存器大小、I/O 口个数、集成何种外设模块等。

（3）对于许多嵌入式系统设计来说，目标不是在于挑选速度最快并且功能最强的 MCU(这样的 MCU 往往价格较高)，而是选择对于完成功能最合适的 MCU。

（4)MCU 的总 I/O 口个数应略多于系统功能所需的个数，以备功能扩展和调试时使用。

（5）对于使用到的外设功能模块应尽可能集成在 MCU 的内部，以简化硬件系统，减少系统工作功耗，提高系统的可靠性。

（6）尽量选择设计者较为熟悉和开发工具完备的芯片型号，可以减少开发周期，提高开发效率。

第二步，选择一个合适的评估工具，利用该工具可以评估系统的大部分输入/输出与通信模块。也可以自行搭建硬件评估系统。有一些开发系统带有较大的 IC 扩展区域，利用现有的 MCU 可以进行新系统的一些初步评估。在此过程中，子程序尽可能规范设计，把与 MCU 引脚相关的程序分离出来，做成独立的子程序（MCU 的引脚分配应独立列表）。

第三步，有了以上的工作，就可以对所选的 MCU 进行硬件系统的设计工作了。设计时，应以 MCU 为核心，逐个硬件模块进行。进行 PCB 布局时，滤波电容应靠近芯片引脚并尽

量加粗电源线。PCB 板上要充分考虑预留测试点和 MCU 引脚接口，以便后续测试和功能扩展时使用。最好对于每一硬件模块都用丝印线框在一起并用文字注明。

第四步，对硬件系统及模块进行测试，这一步非常重要。硬件板制作出来之后，应首先焊接 MCU（或其插座）及其支撑电路，利用软件测试其是否正常工作。若正常工作，再从由简单的硬件模块到复杂的硬件模块逐步焊接上去，每个模块焊接之后，都必须编写简单的测试程序进行测试，测试正确方可进行下一步，并把注意点记入备忘录中，以便进行硬件修改。这样，可以完成硬件的第一版工作，若不正常，不可以进行下一步工作。

第五步，若正常，则进行软件系统设计。有了硬件系统和基本软件模块，在此基础上进行软件设计。可以参照软件工程的一些工作规范进行。对于不使用嵌入式实时操作系统的应用系统，可以参考面向硬件对象编程和程序模块化封装的思想，将与硬件相关的驱动程序与功能性程序分割开来，以提高程序运行效率和可读性。对于基于嵌入式实时操作系统的应用系统，则必须合理的安排和划分任务，充分发挥嵌入式实时操作系统的性能和优势。

第六步，软件系统测试。测试是一个过程，它的根本任务是发现系统中的缺陷，在系统开发过程中，测试是一个基本要素，它有助于提高系统的品质。测试的目标是使得系统更加完善，对环境的适应能力更强。为了达到测试目标，每一个测试过程都包含这些：制定计划、列出测试清单和执行测试用例。接下来接收用户信息反馈、完善、文档分类整理等工作，使一个项目有头有尾，同时也积累开发经验与素材。

注意：每个过程的工作必须写出设计文挡。

1.3 嵌入式业界对人才素质需求分析

嵌入式系统领域门槛较高，嵌入式开发人员不仅要懂较底层软件，对软件专业水平要求较高，市场上需要的嵌入式人才如必须具备 C 语言编程和汇基本编语言编程经验、protel 99SE、嵌入式操作系统（uc/os-ii 或嵌入式 Linux）经验、内核裁剪经验、操作系统移值经验、驱动程序开发经验等。而企业真正需求的是具有丰厚的理论知识，熟练的专业技能，崇高的职业素养，动手能力、应用能力、团队合作精神等集一体的综合素质嵌入式人才。

1.4 嵌入式技术学习方法

有关嵌入式技术的学习方法，因人而异，但是它还是有普遍的规律可循的，这里根据一般的学习习惯总结以下两点建议。

1.4.1 循序渐进 逐步深入

对于刚接触嵌入式的初学者，首先，要清楚自己需要从中获取什么信息，了解自己要学什么，且明确为什么要学习嵌入式应用技术。想学好嵌入式应用技术，书本知识是相当重要的，理解嵌入式系统的原理和基本编程方式是以后的学习和自主创新设计的基础。

其次，学习嵌入式应用技术，要选择入门的芯片。MCU 种类繁多，在应用中需要对各种 MCU 都有所了解，以确定最佳的性价比。通常的方法是学习一种典型的 MCU 系列达到实用程度，在应用中首先选择使用该系列。如果确实要用其它系列 MCU 时，只需寻找两种系列 MCU 的不同点对比学习，这样可在较短时间内掌握并很快的用起来。选择入门的 MCU 系列要选择在国际上比较流行，系列比较齐全，在若干年内不会被淘汰，且抗干扰性强、性能价格比较高的产品。

再次，针对硬件的基本驱动程序的编写也很重要，它是低层硬件和上层应用软件沟通的桥梁。此时，需要对 MCU 的内存地址分配等芯片内部的资料非常的清楚，而且也要具备熟练实用 C 编程的能力。

最后，在硬件和低层软件都具备的条件下，设计上层软件，将这个嵌入式产品用起来，实现其价值，给人们带来便利。此时，需要清楚低层驱动接口才可以与上层应用软件充分的

衔接。

1.4.2 软硬件协同学习

软硬件协同学习是需要一定硬件基础的。在具备一定硬件基础的前提下，再去学习其它的 MCU 也比较容易上手，这时如果勤于实践，积极的编写代码不断的调试，通过实践理解 MCU 原理，掌握基本的编程规范与方法，最终达到设计应用系统。

学习嵌入式应用技术是在学习硬件的同时调试软件，然后加深对 MCU 的理解，通过软件“打通”硬件，观察硬件响应现象，体会软件功能。且必须与输入输出接口的学习连在一起。MCU 系统离不开各种输入输出接口，学习 MCU，就必须学习各种接口的原理，通过接口的输入输出体现 MCU 程序的功能。

1.5 实验指导书总体结构

1.5.1 编书的目的

编写本书的目的是想让大家知道嵌入式到底是什么、要学什么、学了有什么用；通过做实验能入门、接着熟悉、最后达到自行设计开发。首先，将接触到本实验指导书提供的硬件平台，对其硬件模块化设计思想在此就不多说，在第二章将有详细阐述，希望大家可以掌握这种硬件设计方法，并可以灵活的运用的到其它的设计中；其次，学会基于软件重用的思想进行编程。软件重用的功能是强大的，有时我们编写功能类似的软件模块，就没有必要再重新去写程序，只需将以前的程序拿过来稍加修改便可以达到目的，提高了开发效率，而且以前的程序稳定性也比较又保障，这样稍加修改的程序也可以保证开发质量；再次，学会用立体的眼光看待每一个程序。对于程序初学者看程序，喜欢用顺序的方式，遇到一两句不懂的语句就卡在那走不动了，这样的读程序习惯实际上是不对的。其实每个程序都是不同功能模块的一个调用关系，应有一种先从整体把握程序，再细思每句的含义的思维方式，这样可以很快的抓住程序的功能，对程序的整体把握正确，程序中每一句都是为这个功能服务的，自然也就很容易弄懂。在整个嵌入式系统的开发过程中，软件设计是最重要的，必须有良好的学习方法，才能应对不用的应用。最后，学会平台的设计使用方法，可以在本平台的基础上设计一些具有应用性的功能扩展，切实的解决一些实际问题，这才是本平台设计的真正目的。最后可以根据这种设计思想开发自己的开发平台。

本实验指导书中提供的实验分为三种类型：

（1）验证性实验

该类型实验主要体现在第四章和第五章的前两个实验，设置其目的是：1.熟悉实验环境；2.熟悉平台基本用法；3.掌握最基本的编程技巧。

（2）设计性实验

该类型实验主要体现在第四章和第五章的除验证性实验以外的实验，设置其目的是：1.掌握功能模块的硬件设计；2.学会设计基本硬件驱动；3.学会在实时操作系统上设计多任务的驱动程序。

（3）综合性实验

该类型实验主要体现在第六章，设置其目的是：1.全面掌握软硬件之间的协同设计；2.统筹考虑要设计的内容及设计实现需要用到的硬件功能模块，并采用软件重用的思想将不用功能模块的基本驱动程序或多任务驱动程序加以修改最快速度的实现设计要求；3.增加实验的趣味性，激发学生的创新能力，可以在此平台上设计更多的趣味性强的实验。

1.5.2 编排结构

本实验指导书共分六章内容

（1）第一章 概述 该部分主要讲了嵌入式系统发展史及现状、嵌入式产品的开发流程、嵌入式业界对人才素质要求、嵌入式技术学习方法及实验建议。

（2）第二章 S12 应用开发平台及 MCU 介绍　该部分主要讲了平台的设计思路、硬件结构、提供的资源、工作原理、使用说明及 MC9S12DG128 概述。

（3）第三章 工具使用　该部分主要讲了在实验中要用到的 BDM 调试器使用方法、CodeWarrior4.6 使用方法及超级终端使用方法。

（4）第四章 基本硬件编程实验 该部分通过十个针对硬件编程的实验讲述基于硬件的基本编程方法，该部分实验分为验证性和设计性两种类型。

（5）第五章 多任务程序实验　该部分通过九个多任务实验讲述如何移植实时操作系统，如何从多任务的角度分析基于实时操作系统的不同模块的功能驱动实现，该部分实验分为验证性和设计性两种类型。

（6）第六章 综合实例　该部分把第五章的多任务实验综合了起来，实现多任务的简单应用，同学们可以根据该实例设计自己的应用实例。

第2章　S12MCU 应用开发平台介绍

2.1 设计思路

2.1.1 设计思路概述

平台设计的目的为两方面：一方面，为嵌入式工程师提供一整套的开发支持，通过提供一系列的可重用硬件设计方案、可重用的 RTOS 软件平台、可重用的结构化的软件模块达到嵌入式系统设计重用的目的，达到有效缩短应用开发周期，提高开发质量；另外一方面，帮助学习嵌入式的学生快速入门。

为了满足 S12 系列 MCU 的开发需求，硬件系统采用核心区+扩展区的结构方式。核心区设计成一个最小系统，采用标准总线的方式扩展到扩展区，本平台经对全系列 S12 MCU 的考察，基于总线基本兼容，采用引脚最完整的 MC9S12DG128，选用其外部 IO 定义成标准互连总线。互连总线一端连接需要的核心区，另一端连接扩展区。此外，核心区也可单独设计成一块核心板，然后用欧式插槽和应用开发板相连。

应用开发板的设计为保持通用性，提供嵌入式系统常用的功能模块，将这些模块挂接到互连总线；为了满足不同应用，这些模块还支持重组，因此，这些模块不应该挂接到总线的固定部位，应该在这些模块与互连总线之间插入可通断的总线开关。母板提供的外设不可能满足所有的应用需求，因此，应该支持扩展，最典型的情况是根据应用开发的需求扩展原型设计电路。此外，由于 S12 MCU 支持 BDM 后台调试，开发平台提供 BDM 调试器电路，最灵活方式是单独设计一个 DBM 调试器。综上所述，最终形成开发平台的三件套硬件电路板的方案。即如下图 **2-1**。

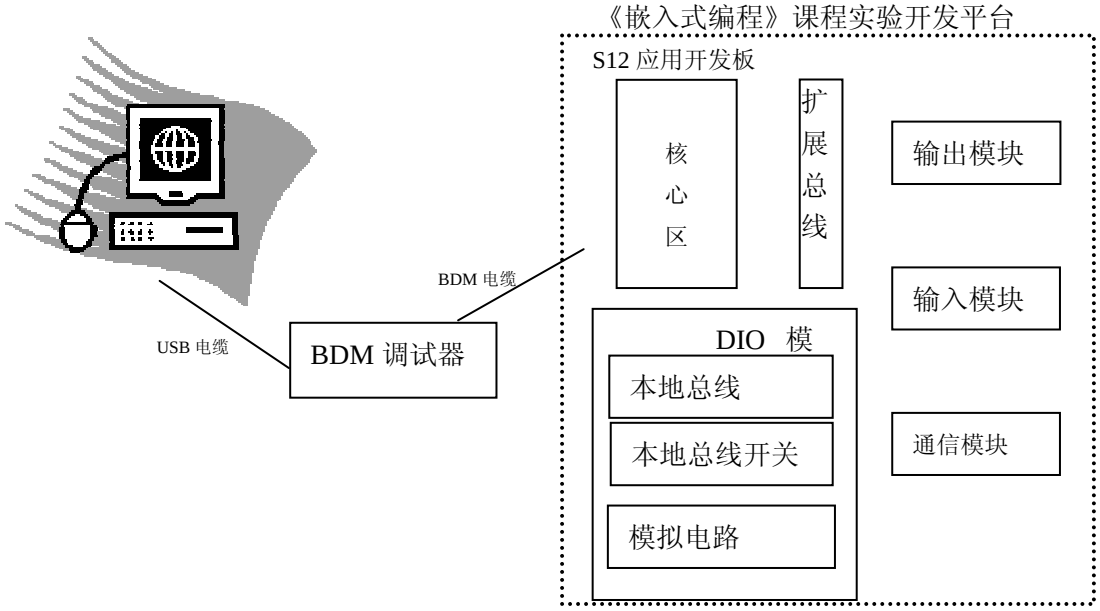

图 2-1　　S12 应用开发板及 BDM 调试器与 PC 机互连

2.1.2 采用关键技术

（1）基于互联总线的硬件系统可重用结构设计技术

S12MCU 嵌入式应用开发平台采用结构化组件的方法来设计了核心子板、应用开发板、扩展开发模块，基于开放式的 S12 互通总线组实现互联。

（2）软硬件协同设计技术

平台设计时，充分考虑了硬件和软件的设计要求和优化设计方面，在整个设计的声明周期，软硬件的设计移植是保持并行的，在设计过程中两者交织在一起，相互支持，相互提供开发的平台。

（3）软件重用技术

平台的软件按照可重用方法设计，采用标准 C 编程、采用层次化和结构化的编程方式，设计了嵌入式实时操作系统 μC/OS-II 下的多任务设备驱动程序，可以很容易移植到其它的系统上。

（4）RTOS 下多任务程序设计技术

平台的软件采用了基于前后台系统和实时操作系统两种开发方式。

前后台系统是实现实时系统的一种方式。其工作模式是：后台程序是一个无限循环，在循环中查询每个事件是否发生，每个任务是否具备运行条件，如果是，则处理这个事件或执行这个任务。所以前后台模式中系统在响应时间上比实际上可以做到的要差；而在移植实时操作系统 μC/OS-II 内核的情况下，设置基于实时多任务的驱动程序便可弥补这点不足，且除此之外，实时操作系统 μC/OS-II 还具有可单独编程、提高系统的稳定性和可靠性、系统的软件维护与功能扩展。

（5）BDM 调试技术

2.2 平台硬件结构

2.2.1 从层次化的结构角度分析

S12 MCU 应用开发平台主要有核心层、应用层和扩展层构成。核心层主要有核心区或单独的核心板构成，是该平台的大脑部分；其次是应用层，有扩展区构成，上面集成多个功能模块，是该平台的躯干部分；扩展层有扩展总线构成，可以扩展用户自己设计的电路模块。采用核心板或核心区、应用开发板既独立又相互联系的设计方式，独立体现在：核心板或核心区只设计其最小系统，所有的功能引脚都通过欧式插座引出，其芯片内部的功能模块可在扩展区上体现；相互联系体现在：核心板通过应用开发板上的欧式插槽相互连接，其功能引脚对应的接入母板，扩展区上将芯片内部的功能模块具体体现。

2.2.2 从结构化组件角度分析

结构化组件主要体现在扩展区上上，本地总线开关灵活的决定本地总线是否挂接到核心板上，其实现的具体方式就是采用总线跳选的方法，本平台具体有 16 个功能模块，如串口通信、液晶显示器 LCD 模块、定时器管理器、数码管显示模块等等，每个模块都设置有本地总线开关，若本地总线开关被跳选，则该模块才真正的被挂接到 S12MCU 上，才能对其编程，验证效果。

2.2.3 从可扩展的开发模块角度分析

本平台并不是单纯的提供了 S12MCU 核心芯片的外部扩展的 16 个功能模块，因为其核心区是灵活的，可以设 计 S12 系列其他的 MCU 作为核心板，由于芯片的不同，其功能模块肯定也不大相同，这就需要我们外部扩展与 S12MCU 芯片中不同的功能模块，为了它们都可以自由的挂接到核心芯片上，我们在母板上设计了 S12 系列 MCU 最全的 112 引脚的扩展总线。

2.3 平台资源

2.3.1 平台硬件资源

S12MCU 应用开发平台集成了强大的硬件资源，并为用户提供了多种选择，使用户可以进行各种相关的实验。

（1） 提供+12V、+5V 电源，含瞬时短路保护和过流保护；
（2） 基于 BDM 的后台调试功能；
（3） 4*4 的矩阵式按键；
（4） SPI 接口的串转并、并转串模块，配置八位拨动开关和八位 LED；
（5） IIC 接口的时钟日历模块；
（6） 两路硬件去抖模块；
（7） RS-232 和 RS-485 串行通信模块；
（8） 一个模拟量输入电位器、两路靠 PWM 方式实现的 D/A 电路；
（9） 采用 SPI 总线扩展 MC33880 芯片，提供智能功率开关；
（10） 采用 PWM 控制的 MC33886 芯片，提供桥式驱动电流；
（11） 1 个 16 列*2 行点阵字符 LCD 模块及接口，一个 128*64 的点阵液晶模块及接口，一个 6 位 8 段 LED 模块及接口（用户可任选）；；
（12） 两路 LIN 总线模块；
（13） 两路 MSCAN 总线模块；
（14） Zigbee 无线通信模块及接口（用户可任选）；
（15） 核心区（提供 S12MCU 工作的最小电路）详见 2.6.2；
（16） BDM 调试器，提供程序下载、擦除、调试；

2.3.2 平台软件资源

配置了标准子模块的基本驱动程序。如：键盘、LED、LCD 驱动程序为用户提供了最基本的人机交互功能；SPI、IIC 驱动程序为用户扩展接口提供了方便；SCI、CAN 等通信模块的驱动程序为用户实现通信功能的应用程序提供了方便；ATD 模块的驱动程序为用户采集信息

和控制电路提供了方便。

配置了基于实时多任务的驱动程序。为用户提供了一套基于 UC/OS-II 的实时多任务驱动程序。如键盘、LED、LCD、钟点、时间管理、离散输入输出、模拟输入输出、SCI 通信等。并且这些多任务程序可以方便的移植到其他的实时操作系统，或单独不基于实时操作系统运行。

S12MCU 应用开发板可以进行各种单片机实验，具体包括：

（1） 单片机 I/O 口控制实验，如拨动开关信号输入，LED 发光二极管控制，按键输入等实验；
（2） 定时器输出 PWM 实验；
（3） AD 采集输出实验；
（4） 蜂鸣器驱动实验；
（5） 结合单片机 I/O 口控制实验和蜂鸣器驱动的电子琴实验；
（6） 串转并、并转串的 SPI 接口实验；
（7） 7 段式动态数码管输出实验；
（8） 16*2 字符 LCD 输出实验、128*64 点阵液晶输出实验；
（9） RS232 串口通信实验；
（10） RS485 差分串行通信实验；
（11） IIC 总线实验，如：日历时钟，Zigbee 无线通信（用户选择）；
（12） LIN 总线通信实验；
（13） MSCAN 总线通信实验；
（14） MC33880 智能开关功率实验；
（15） MC33886 由 PWM 控制的桥式驱动电流实验；

以上各种实验，不仅可以做基本的驱动程序实验，也可以做基于 UC/OS-II 的多任务实验。本指导书提供了基本驱动程序的实验和基于 UC/OS-II 的实验，最后给出了一个综合的多任务案例。

2.4 平台工作原理

S12MCU 应用开发平台，采用 CoderWarrior 集成开发环境和 BDM 调试器进行调试程序和在线编程。另外通过操作系统自带的超级终端通过串口利用监控程序也可以调试开发板，不过前提是必须通过 BDM 调试器把监控程序下载到 MCU 中。

CoderWarrior 4.6 是面向 H12 或 S12 为 CPU 的单片机嵌入式应用开发的软件包，。包括集成开发环境 IDE、处理器专家库、全芯片仿真，可视化参数显示工具、项目工程管理器、C 交叉编译器、汇编器、链结器以及调试器。（具体的使用方法可参见第三章第二节）

利用 BDM 模块可以对片内 FLASH 进行擦除和写入操作。实现应用程序的动态调试。通过 MCU 内嵌的 BDM 模块，在不影响单片机正常运行的情况下，BDM 调试器可以读/写单片机的存储器，实现应用程序的动态调试。配置与修复 MCU 内部资源，对应用程序进行加密处理等。BDM 调试电路中的 BDM 连接头采用 IDC6 连接子，线序由飞思卡尔公司定义，该电路十分简

单，实质就是将 S12MCU 的 BKGD 引脚和电源、地线、复位控制线引出即可。

BDM 调试器是用来初次向目标板下载程序用的，也可以将目标板上的单片机 FALSH 中的旧程序擦除。这就是 BDM 调试器的 FALSH 写入、擦除功能，也称为编程功能。只具备这种编程功能的简单开发工具也称作编程头，内有 FLASH 的单片机都能在线编程，开发内有 FALSH 的单片机至少需要一个有编程头功能的下载工具。这里称之为 BDM 调试器而不称为编程头，是因为 BDM 调试器还实现了编程器之外的许多调试功能——它还支持大量的调试命令。能做出有多少功能的编程器其实主要看单片机本身的 BDM 接口是否接受调试命令，以及能不能提供更多的调试信息。S12 单片机的 BDM 接口可以提供很多调试信息，包括 CPU 运行时时的动态信息，实际上，与之通信的 BDM 调试器内部有一个单片机。（具体的使用方法参见第三章第一节）

超级终端是一个通用的串行交互软件，很多嵌入式应用的系统内含串行通信程序，如果这部分程序采用标准 ASIIC 码通信协议，就可以通过超级终端与嵌入式系统交互，使超级终端成为嵌入式系统的“显示器”或人机交互界面。超级终端的工作原理并不复杂，它是将用户的键盘输入的 ASIIC 码即时发向串口（采用 TCP 协议时是发往网口，这里只说串口的情况），但并不显示输入。它在 CRT 上显示的信息是从串口接收到 ASIIC 的字符。一般情况下，嵌入式系统中的串行通信程序应该完成的任务是：

（1）将自己的启动信息、过程信息主动发到运行有超级终端的主机；

（2）将接收到的字符返回到主机，同时发送需要显示的字符（如命令的响应等）到主机

在此基础上，超级终端才可以真正成为嵌入式系统的人机交互界面。（具体使用方法参见第三章第三节）

2.5 平台使用方法

本平台硬件组成包括：基于 MC9S12DP512 MCU 的应用开发板，BDM 调试器及一根 BDM 连接线，标准 12V 电源，RS-232 标准通信串口线和一根 USB 线。

具体的实验步骤如下：

（1）在实验之前，需要注意硬件连接。用 BDM 连接线将开发板板与 BDM 调试器连接，此时须注意接线时应注意板上各自表示的一脚应对应连接，否则将引起程序下载错误；接着用串口线将 PC 机与开发板连接，注意：此时一定要是掉电状态，否则将可能烧坏串口芯片；用 USB 线将 PC 与 BDM 调试器连接，整个硬件连接完毕；最后接通电源，注意：要最后接通电源为防止带电拔插，而且电源也应根据不同模块接 5V 或 12V 电源，大部分采用的是 5V 电源供电。

（2）用 CodeWarrior 4.6 软件进行建造工程、编写源码、编译、连接、调试、下载。可以利用 BDM 调试器调试程序，如有错误，返回程序源码修改，直至程序正确下载。BDM 调试器的使用调试界面介绍可参考第三章 3.1 小节；另一种则是通过 S12 串行监控，实现对程

序.S19 文件的正确下载，该方法不常用。

（3）实验做完以后先切断电源关闭实验系统，再拔掉连接在 PC 机上的 USB 线，然后拆除系统各组件之间的连接和及其他线路。

2.6 MC9S12DG128 概述

2.6.1 MC9S12DG128 的特征

MC9S12DG128 是 Freescale 公司推出的 S12 系列微控制器中的一款增强型 16 位微控制器。其集成度高，片内资源丰富，接口模块包括 SPI、SCI、I2C、A／D、PWM 等。它不仅在汽车电子、工业控制、中高档机电产品等应用领域具有广泛的用途，而且在 FLASH 存储控制及加密方面也有很强的功能。

MC9S12DG128 微控制器采用增强型 16 位 S12CPU，片内总线时钟频率最高可达 25 MHz；片内资源包括 8KBRAM、128KB FLASH、2KBEEPROM；SCl、SPI、PWM 串行接口模块；PWM 模块可设置成 4 路 8 位或 2 路 16 位，可宽范围选择逻辑时钟频率；它还提供 2 个 8 路 10 位精度 A／D 转换器、控制器局域网模块 CAN 和增强型捕捉定时器，并支持背景调试模式(BDM)。

S12 的核心：

16 位 S12CPU：20 位 ALU，指令队列，增强型索引寻址；
多种外部总线接口(MEBl)；
模块映射控制机制(MMC)；
中断控制(1NT)；
断点(BKP)；
背景调试模块(BDM)。

CRG 时钟和复位发生器：

锁相环(PLL)；
看门狗(COPwatchdog)；
实时中断(RTl)；
时钟监视器(CM)。

带中断功能的 8 位和 4 位端口：

可编程的上升沿或下降沿触发。

存储器：

128 KB FLASH；
2 KB EEPROM；
8 KB RAM。

2 个 8 通道模／数转换器：

10 位精度；
外部触发转变功能。

3 个 1 Mbps 的 CAN 总线模块，兼容 CAN2.0A／B：

5 个接收缓冲器，3 个发送缓冲器；
4 个独立的中断通道，分别是发送中断、接收中断、错误中断和唤醒中断；
低通滤波器唤醒功能。

增强型捕捉定时器：

16 位计数器，7 位预分频功能；
8 个可编程输入捕捉或输出比较通道；
4 个 8 位或 2 个 16 位脉冲累加器。

8 个 PWM 通道：

每个通道的周期和占空比由程序决定；
8 位 8 通道或 16 位 4 通道；
各通道独立控制；
脉冲在周期内中心对称或左对齐输出；
可编程时钟选择逻辑；
紧急事件关断输入；
可作为中断输入。

串行口：

2 个异步串行通信接口(SCl)；
2 个同步串行设备接口(SPl)；
Byteflight 模块。

I2C 总线：

兼容 I2C 总线标准；
多主 I2C 总线模块。

LQFP—112 和 QFP—80 封装选择：

5 V 输入和带驱动能力 I／O；
5 V A／D 转换器输入；
50 MHz 系统频率(相当于 25 MHz 总线频率)；
单线背景调试模块；
片上硬件断点。

下图 **2-2** 是 MC9S12DG128MCU 的整体结构框图，显示了所有主要外设系统和所有外

设引脚。

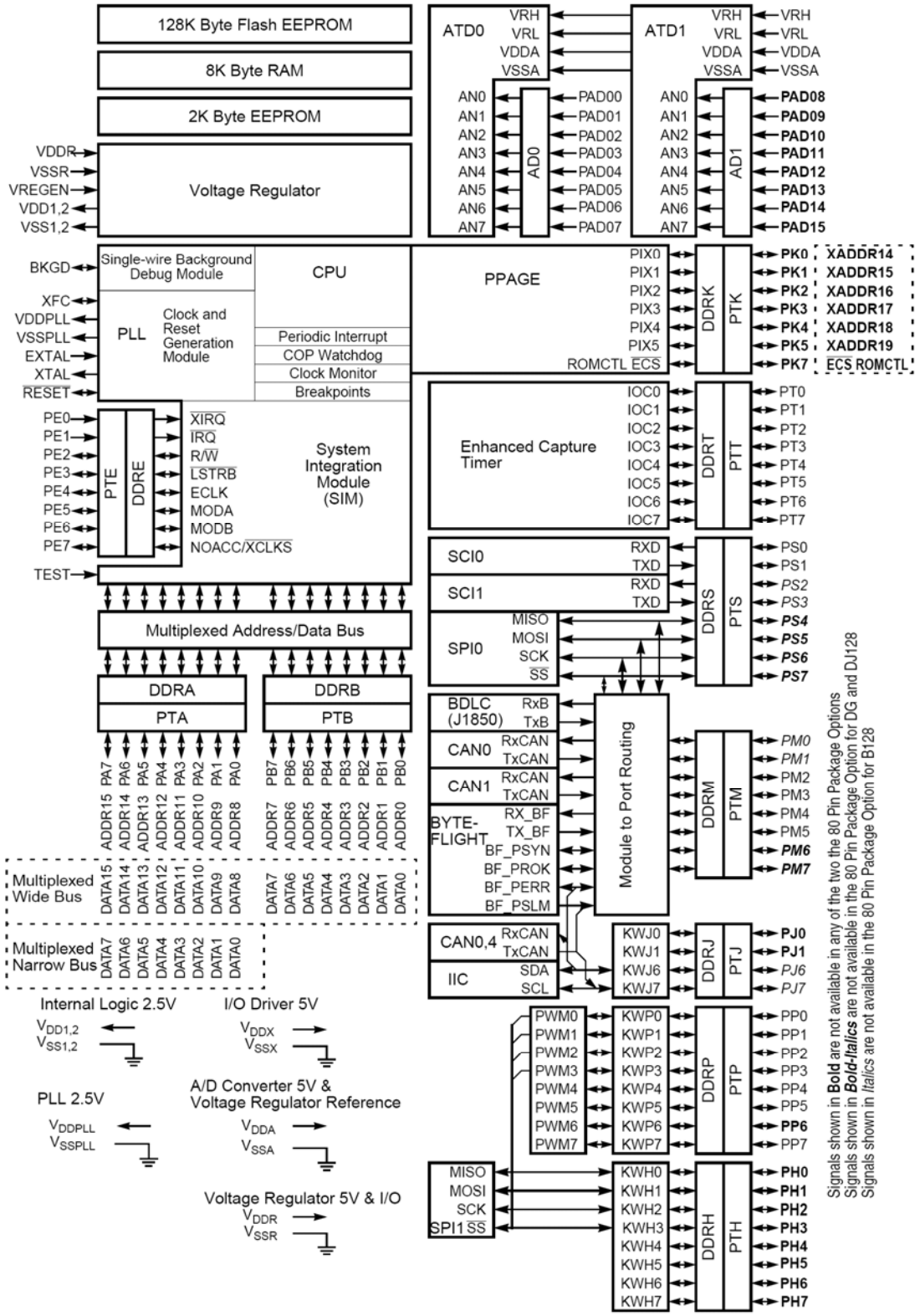

图 **2-2**　MC9S12DG128 的结构框图

2.6.2　MC9S12DG128 核心模块电路介绍

第一，电源电路

MC9S12DG128 芯片手册推荐使用 5V 供电，5V 直流电源经常用，我们采用 5V 的稳压电源加电容滤波的方案，用于解决 MCU 供电的问题。

第二，晶振时钟电路

为了给 MCU 提供时钟信号，必须紧密结合芯片手册配置时钟电路和复位电路。

第三，BDM 调试接口

如何解决对 MCU Flash ROM 写入、擦除。飞思卡尔公司的 S08 系列 MCU 片内都有支持 BDM 调试的模块。

第四，通信电路

MCU 一般通过 SCI 口与 PC 进行通信，一片 MAX3232 就能实现 MCU 与 PC 串口之间的串行通信。

第五，接口

根据不同的需要设计方案会有所不同。这里考虑到基本系统要尽量把所有可能用到的功能引脚都引出来，这里采用两个 64 针的欧式插座作为信号引出接口。

第六，综合设计

把各部分放到一个系统里使其成为一个整体完成基本系统设计。

2.6.3 MC9S12DG128 复位、中断

（1）特征简介

复位和中断特征包括：

★ 多种复位源可灵活配置系统且操作可靠
— 发现上电复位(POR)
— 低电压检测(LVD)使能
— 使能外部 RESET 引脚
— 计算机正常操作监控模块(COP)看门狗使能和两个溢出时间选择
— 非法操作码检测
— 背景调试主机发的连续命令
★ 复位状态寄存器(SRS)指出最近一次发生复位的原因
★ 为每个模式分开设置中断矢量（见表 **2-1**）

（2）MCU 复位

复位提供一种从已知的初始条件设置来启动处理的方法。在复位期间，大部分控制和状态寄存器都设为复位默认值，从地址$FFFE～$FFFF 取出两字节的复位向量送到程序计数器 PC。片上外设模块不可用，I/O 引脚初始化设为通用的高阻抗输入引脚无上拉。条件码寄存器(CCR)的 I 位被置位来屏蔽中断，直到用户程序初始化堆栈指针(SP)和系统控制设置。

MC9S12DG128 提供 5 个复位源：

★ 上电复位(POR)
★ 低电压检测(LVD)
★ 外部复位
★ 时钟监视复位
★ 看门狗复位

（3）中断

中断是提高微控制器工作效率的一种重要手段，中断方式与查询方式相比，可使 MCU 的程序设计更加高效与灵活，可以提高嵌入式系统的实时处理能力，扩大其应用范围，尤其是在低功耗应用系统中，中断更是一个必要的技术手段。可以说 MCU 的中断系统的功能如何在某种程度上决定了 MCU 的用途，中断功能强大与否也是评判 MCU 性能的一个重要指

标。

S12MCU 以向量中断方式提供了一个功能强大的异常处理系统，并且有灵活的配置方式供使用者选择。

中断系统一般具有以下特点：

★能实现中断响应、中断服务和中断返回。当一中断源发出中断请求时，MCU 决定是否响应这一请求。如果允许响应这个中断请求，MCU 能够由硬件自动保护断点，转而执行相应的中断服务程序。中断处理完后能通过指令自动恢复断点，返回原中断处继续执行被中止的程序。

★能实现中断优先级配置。当两个或更多个中断源同时发出中断申请时，优先级较高的中断申请首先得到处理。

★能实现中断嵌套。中断处理过程中，有优先级较高的中断请求时，MCU 能暂停正在执行的中断处理程序，转去响应与处理优先级较高的中断申请，结束后再返回原先优先级较低的中断处理过程。

★能通过软件实现模拟中断的功能，便于中断的调试。

S12MCU 提供了丰富的中断源：2～122 个(I 位)可屏蔽中断向量、1 个(X 位)可屏蔽中断、1 个不可屏蔽软件中断或者背景调试模式申请向量、1 个指令陷阱中断向量、3 个系统复位向量。

正常状况下，S12MCU 按照存储器中程序指定的顺序有条不紊地执行代码。这时即使因为出现跳转使 PC 指针发生突变，程序的执行仍然是可预测和可控制的。

S12 的中断可分为：可屏蔽中断与不可屏蔽中断，两种中断都可以通过对 CCR(X、I 位)的设置对其进行屏蔽。

X 位控制不可屏蔽中断，I 位控制可屏蔽中断。X、I 位为。时分别允许相应的中断。不同的是 X 位复位后一旦被清零就不能再次对其进行设置，无论是通过直接设置、堆栈弹出、位清除等都不能再次对该位置 1；而 I 位是任意时刻可设置的，随时可以打开和关闭可屏蔽中断。也就是说，不可屏蔽中断只能被设置一次，而可屏蔽中断则可以多次打开与关闭。不可屏蔽终端一旦打开，无法关闭，仅用于十分关键的环节。

S12 采用一套独特的机制实现对异常的处理，在 0xFF80～0xFFFF 地址空间中设置了一个向量映射表，每一个向量对应一种异常的处理程序的地址。向量表中的每个向量占用 2 字节空间，所以 S12 最大可有 64 个中断向量。

S12MCU 的每种复位操作均有相应的中断矢量，存放在 64 KB 内存区的最后 128 字节，每个中断向量占用 2 字节，指向中断服务程序入口。

MCU 在中断响应时，依据中断信号的来源在中断向量表中对应的位置取得中断向量的 2 个字节地址即读取中断处理程序的人口地址，进而依此地址转到相应的中断服务程序。

向量表的首地址固定在$FF80 不变，其内容由用户在程序设计阶段给定，一般不能在运行阶段时更改。中断向量表如下表所列。

★第 1 列是中断向量地址，每个中断向量占用 2 字节；

★第 2 列是中断源；

★第 3 列是 CCR 寄存器中对该中断的屏蔽设置位，即控制位；

★第 4 列是对该中断使能进行的设置位；

★第 5 列是用户在程序中设定该中断优先级时需要写 HPRIO 寄存器的代码。

序　号	矢量地址	中断源	CCR 中屏蔽位	使能位	最高优先级设定
1	$FFFE、$FFFF	复位	无	无	-
2	$FFFC、$FFFD	时钟监视器	无	COPCTL(CME、FCME)	-

		动作			
3	$FFFA、$FFFB	看门狗溢出动作	无	COP 溢出速率已选定	-
4	$FFF8、$FFF9	非法指令陷阱	无	无	-
5	$FFF6、$FFF7	软件中断 SWI	无	无	-
6	$FFF4、$FFF5	外部中断 XIRQ	控制位 X	无	-
7	$FFF2、$FFF3	外部中断 IRQ	控制位 I	INTCR(IRQEN)	$F2
8	$FFF0、$FFF1	实时中断	控制位 I	CRGINT(RTIE)	$F0
9	$FFEE、$FFEF	定时器通道 0	控制位 I	TIE(C0I)	$EE
10	$FFEC、$FFED	定时器通道 1	控制位 I	TIE(C1I)	$EC
11	$FFEA、$FFEB	定时器通道 2	控制位 1	TIE(C21)	$EA
12	$FFE8、$FFE9	定时器通道 3	控制位 I	TIE(C3I)	$E8
13	$FFE6、$FFE7	定时器通道 4	控制位 I	TIE(C4I)	$E6
14	$FFE4、$FFE5	定时器通道 5	控制位 I	TIE(C5I)	$E4
15	$FFE2、$FFE3	定时器通道 6	控制位 I	TIE(C6I)	$E2
16	$FFE0、$FFE1	定时器通道 7	控制位 I	TIE(C7I)	$E0
17	$FFDE、$FFDF	定时器溢出	控制位 I	TSCR2(TOI)	$DE
18	$FFDC、$FFDD	脉冲累加器 A 溢出	控制位 I	PACTL(PAOV1)	$DC
19	$FFDA、$FFDB	脉冲累加器有效沿	控制位 I	PACTL(PA1)	$DA
20	$FFD8、$FFD9	SPI0 串行口	控制位 I	SPICR1(SPIE、SPTIE)	$D8
21	$FFD6、$FFD7	SC10 串行口	控制位 I	SCICR2(TIE、TCIE、RIE、ILIE)	$D6
22	$FFD4、$FFD5	SCI1 串行口	控制位 I	SCICR2(TIE、TCIE、RIE、ILIE)	$D4
23	$FFD2、$FFD3	ATD0	控制位 I	ATDCTL2(ASCIE)	$D2
24	$FFD0、$FFD1	ATD1	控制位 I	ATDCTL2(ASCIE)	$D0
25	$FFCE、$FFCF	端口 J	控制位 I	PIEJ(PIEJ7、PIEJ6、PIEJ1、PIEJ0)	$CE
26	$FFCC、$FFCD	端口 H	控制位 I	PIEH(PIEH7～PIEH0)	$CC

27	$FFCA、$FFCB	模数递数器下溢	控制位 I	MCCTL(MCZI)	$CA
28	$FFC8、$FFC9	脉冲累器 B 溢出	控制位 I	PBCTL(PBOVI)	$C8
29	$FFC6、$FFC7	锁相环锁定	控制位 I	PLLCR(LOCKIE)	$C6
30	$FFC4、$FFC5	CRG 自时钟方式	控制位 I	PLLCR(SCMIE)	$C4
31	$FFC2、$FFC3	BDLC 接口	控制位 I	DLCBCR1(IE)	$C2
32	$FFC0、$FFC1	IIC 总线	控制位 I	IBCR(IBIE)	$C0
33	$FFBE、$FFBF	SP11 串行口	控制位 I	SPICR1(SPIE、SPTIE)	$BE
34	$FFBC、$FFBD	保留			
35	$FFBA、$FFBB	EEPROM 存储器	控制位 I	ECNFG(CCIE、CBEIE)	$BA
36	$FFB8、$FFB9	FLASH 存储器	控制位 I	FCNFG(CCIE、CBEIE)	$B8
37	$FFB6、$FFB7	CAN0 唤醒	控制位 I	CANRIER(WUPIE)	$B6
38	$FFB4、$FFB5	CAN0 错误	控制位 I	CANRIER(CSCIE、OVRIE)	$B4
39	$FFB2、$FFB3	CAN0 接收	控制位 I	CANRIER(RXFIE)	$B2
40	$FFB0、$FFB1	CAN0 传送	控制位 I	CANTIER(TXEIE[2：0])	$B0
41	$FFAE、$FFAF	CAN1 唤醒	控制位 I	CANRIER(WUPIE)	$AE
42	$FFAC$FFAD	CAN1 错误	控制位 I	CANRIER(CSCIE、OVRIE)	$AC
43	$FFAA、$FFAB	CAN1 接收	控制位 I	CANRIER(RXFIE)	$AA
44	$FFA8、$FFA9	CAN1 传送	控制位 I	CANTIER(TXEIE[2：0])	$A8
45	$FFA6、$FFA7	BFRX 结果寄存器非空	控制位 I	BFRIER(RCVFIE)	$A6
46	$FFA4、$FFA5	BF 接收	控制位 I	BFBUFCTL[15：0](IENA)	$A4
47	$FFA2、$FFA3	BF 同步	控制位 I	BFRIER(SYNAIE、SYNNIE)	$A2
48	$FFA0、$FFA1	BF	控制位 I	BFBUFCTL[15:0](IENA)，BFGIER(OVRNIE、ERRIE、SYNEIE、SYNLIE、ILLPIE、\LOCKIE、WAKEIE)，BFRIER(SLMMIE)	$A0
49	$FF98、$FF9F	保留			
50	$FF96、$FF97	CAN4 唤醒	控制位 I	CANRIER(WUPIE)	$96
51	$FF94、$FF95	CAN4 错误	控制位 I	CANRIER(CSCIE、OVRIE)	$94
52	$FF92、$FF93	CAN4 接收	控制位 I	CANRIER(RXFIE)	$92
53	$FF90、$FF91	CAN4 传送	控制位 I	CANTIER(TXEIE[2：0])	$90
54	$FF8E、$FF8F	端口 P 中断	控制位 I	PIEP(PIEP 7～PIEP0)	$8E
55	$FF8C、$FF8D	PWM 紧急关断	控制位 I	PWMSDN(PWMIE)	$8C
56	$FF80、$FF88	保留			

表 2-1 中断向量表

第3章　开发工具使用

3.1 BDM 使用方法

由于 S12 系列 MCU 中具有一个 BDM 调试硬件模块，平台配备开源 BDM 调试器。其主要特点包括后台调试和单线调试通信，前者属于非侵入调试，对实时系统的调试非常适用；后者意味着调试仅占用一根通信线。BDM 调试具有应用程序的擦除与下载、实现应用程序的动态调试和配置与修复 MCU 内部资源，对应用程序进行加密处理等功能。

S12MCU 的功能除了运行频率的提高和功能模块的增加、增强外，更重要的是增加了 RTOS 和后台调试 BDM 调试技术。对于片内 FLASH 空白的 MCU，采用 BDM 方式将监控程序下载下去，并在 MCU 运行时可对其进行动态调试。

下面着重讲一下 BDM 调试技术：

（1） 硬件接线

如图 2-3 所示：

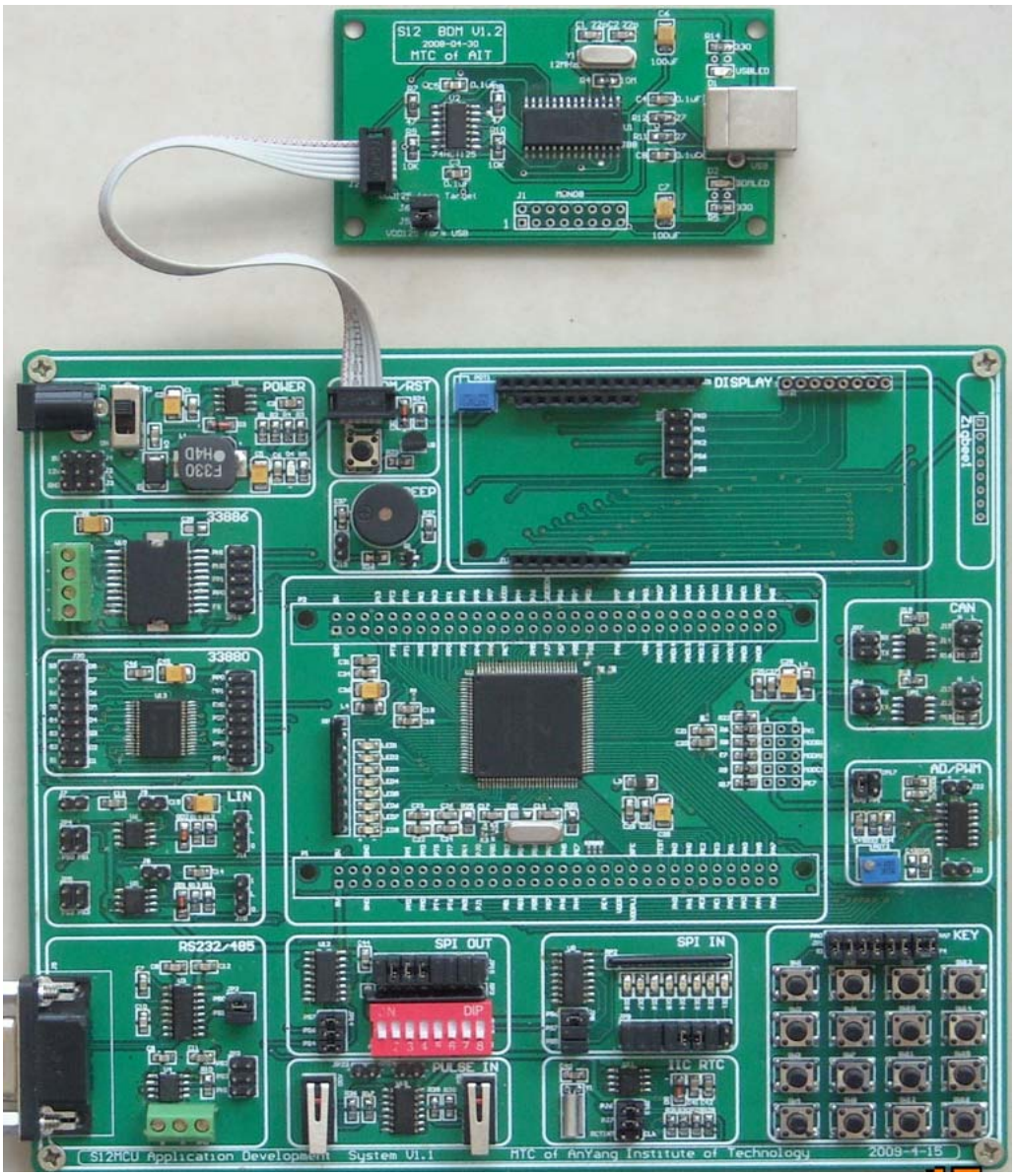

图 2-3

注意事项：

① 在连接 BDM 线的时候，连接线中的红线必须连接在 BDM 板上的双排插座（J2）中标号为“1”的引脚，另一端必须和核心板上的引脚“BDM_IN”的“1”脚对应连接，不能插反，否则将不能正常下载。

② 连接好各个硬件之后，在连接 BDM 与核心板的时候，要注意 BDM 上接插点旁边的 JP4 跳线，如果该跳线能够提供给核心板所需要的+5v 电源，那么就不需要连接电源。

③ 如果不能够提供则须连接电源，注意最后插电源。实验完毕之后须先拔电源在拆线，防止烧毁芯片。

④ 将 USB 接口连接到电脑 USB 口上。

（2） 安装 BDM 的驱动

注意：Codewarrior 4.6 及其以上版本不需要安装 BDM 驱动。此步骤可跳过。

第一次使用 BDM 进行调试时，会弹出如图 2-4 提示：

图 2-4

选择“从列表或指定位置安装（高级）（S）”，如图 2-5：

图 2-5

在弹出的对话框中选择要安装的 Open Source BDM 驱动程序的路径，如图 2-6：

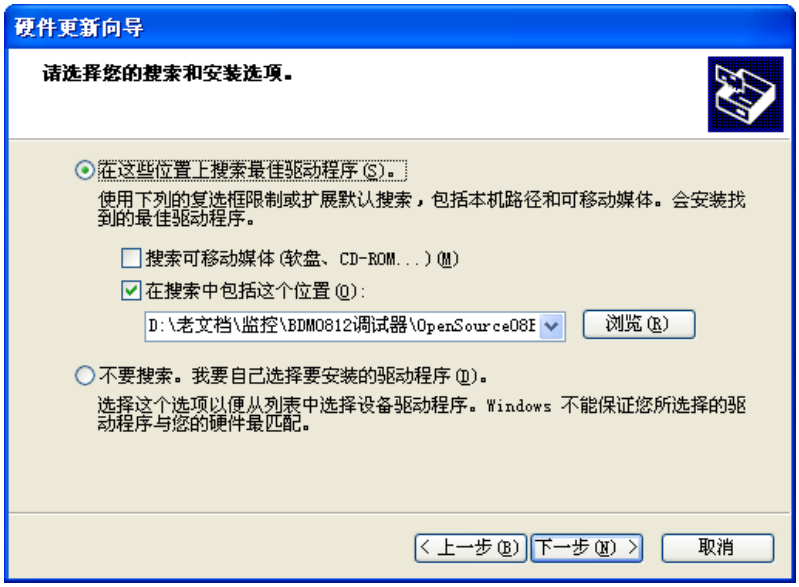

图 2-6

选择完路径，然后点击“下一步”，出现如图 2-7 的对话框，表明驱动程序正在安装。

图 2-7

单击“完成”就完成了驱动程序的安装，如图 2-8。

图 2-8

（3） 硬件电路板路线说明

跳线	跳线说明	设置
JP1	FLASH 察除	跳线冒打开
JP2	FLASH 编程	2、3 短接
JP3	FLASH 编程	2、3 短接
JP4	目标板供电选择	短接目标板（5V）供电

表 2-2

（4） 在编译器里进行连接

如图 2-9 所示：

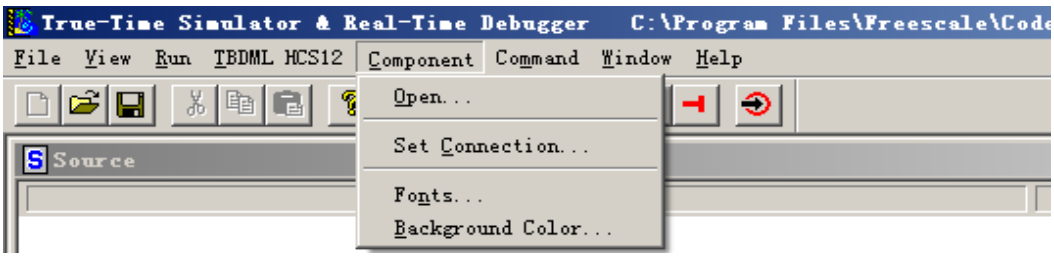

图 2-9

选择 Compent->Set Connection，弹出如图 2-10 对话框，

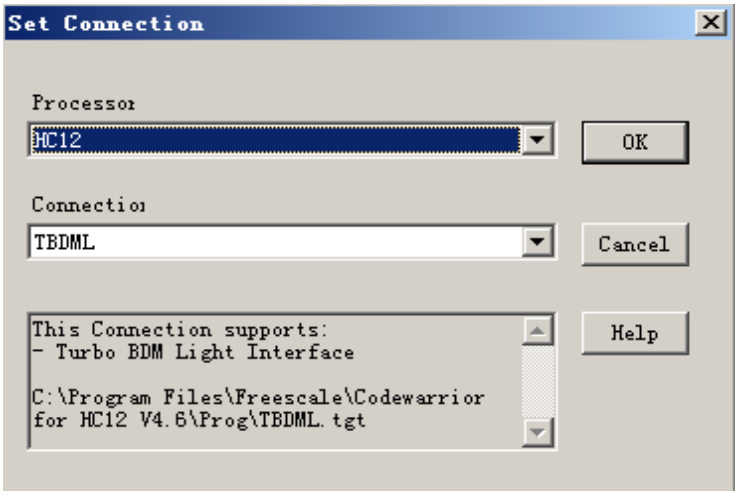

图 2-10

按照图 2-10 进行选择，然后点击“OK”，就可以连接上了，

（4）可能出现的问题及其解决办法

通常，在连接过程中会出现一些问题，下面讲述如何解决。

错误 1：

图 2-11

第一种解决办法：也是最简单的办法，将和 BDM 连接的插头拔掉，断开和 BDM 的连接，再重新插上，然后按照（3）重新进行连接。

第二种解决办法：将 BDM 驱动卸载，然后重新安装。具体操作如下：

右键单击“我的电脑”->属性->硬件->设备管理器，按下图所示选中 Open Source BDM, 如图 2-12：

图 2-12

然后点右键将会弹出如图 2-13 对话框，

图 2-13

选择“卸载”，在弹出的如图 2-14 对话框中点击确定。

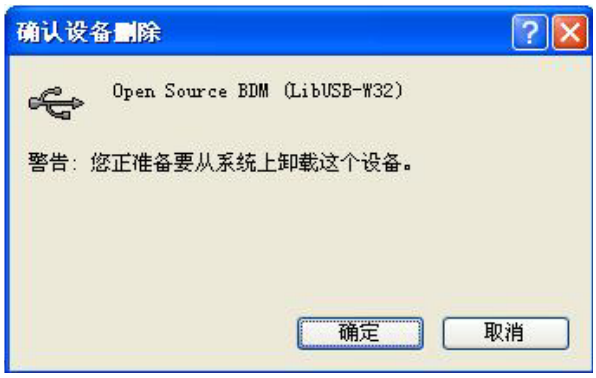

图 2-14

等卸载完成，在按照步骤（2），重新安装 BDM 驱动即可。

（5）下载程序

进行上述步骤，可从接下来的 3.2 节中**（2）项目的编译连接**的第三步开始进行程序擦除和下载。

（6）程序调试

当程序下载完毕后，单击工具栏中绿色箭头图标，即可观察实验现象，如图 2-15，

图 2-15

并非每个程序都可以正确显示实验现象，这时我们就需要进行程序调试了。

在 True-Time Simulatot & Real-Time Debugger 环境中单击右键，选择 Open Source File

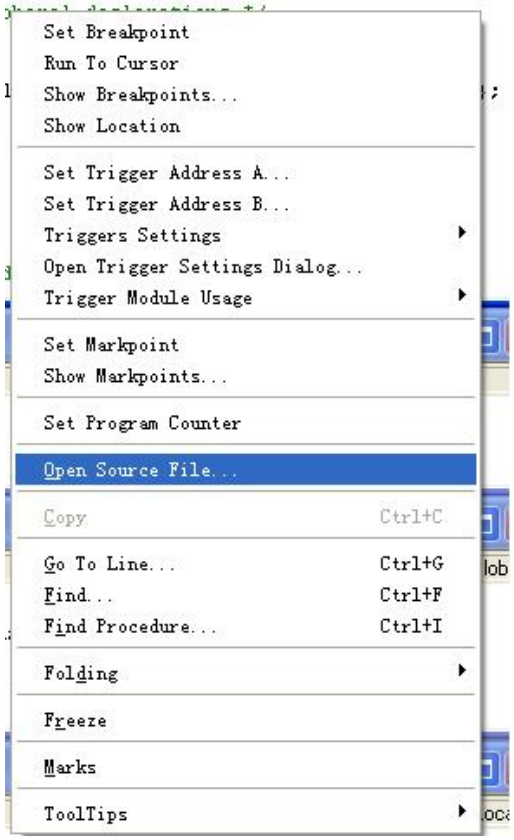

图 2-16

在弹出的对话框中选择自己要调试的.C 文件，按上一节所述，执行单步调试。

调试环境中各个窗口的说明在下一节将作详细说明。

3.2 CodeWarrior 4.6 使用方法

本实验系统使用的是 Freescale 公司的专门面向其所有 MCU 与 DSP 嵌入式应用开发的工具“Freescale CodeWarrior”。它是面向 S12 或 S12 为 CPU 的单片机嵌入式应用开发的软件包，包括集成开发环境 IDE、处理器专家库、全芯片仿真，可视化参数显示工具、项目工程管理器、C 交叉编译器、汇编器、链结器以及调试器。

下面将以一个简单的例子说明 CodeWarroir 如何使用。

（1）工程创建

第一步：安装好 CodeWarrior 后，打开 CodeWarrior 窗口，然后按照如图 2-17 所示进行操作：

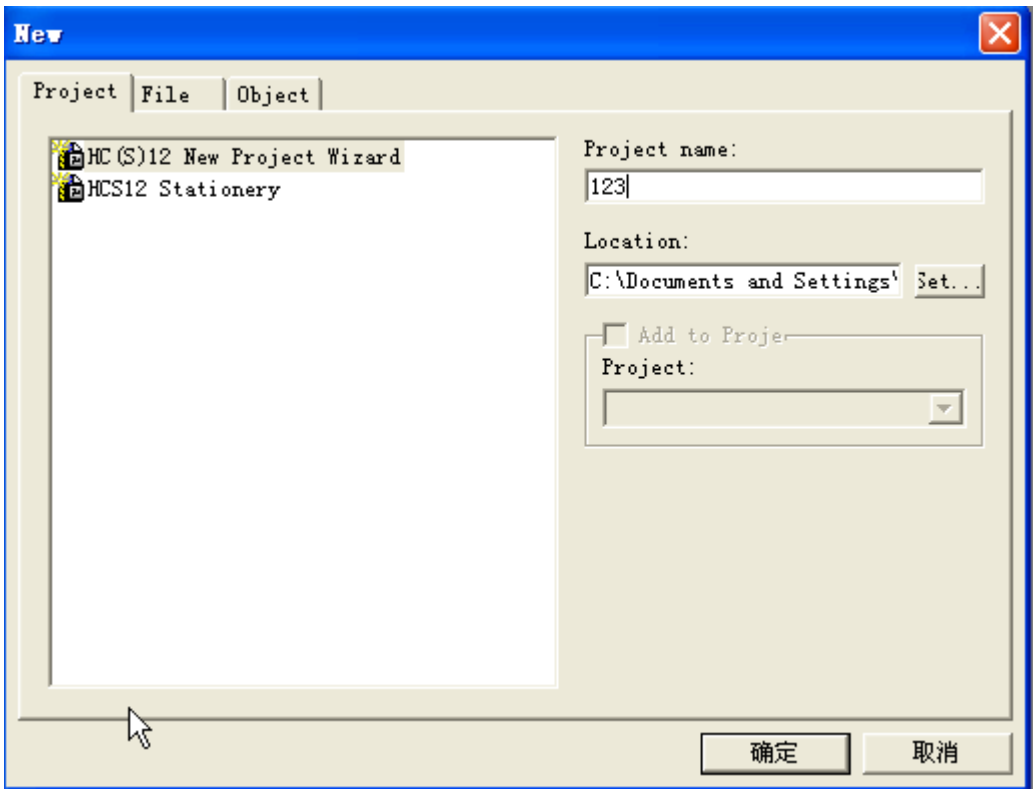

图 2-17

第二步：在上图中，点击“下一步”。出现图 2-18 所示画面。

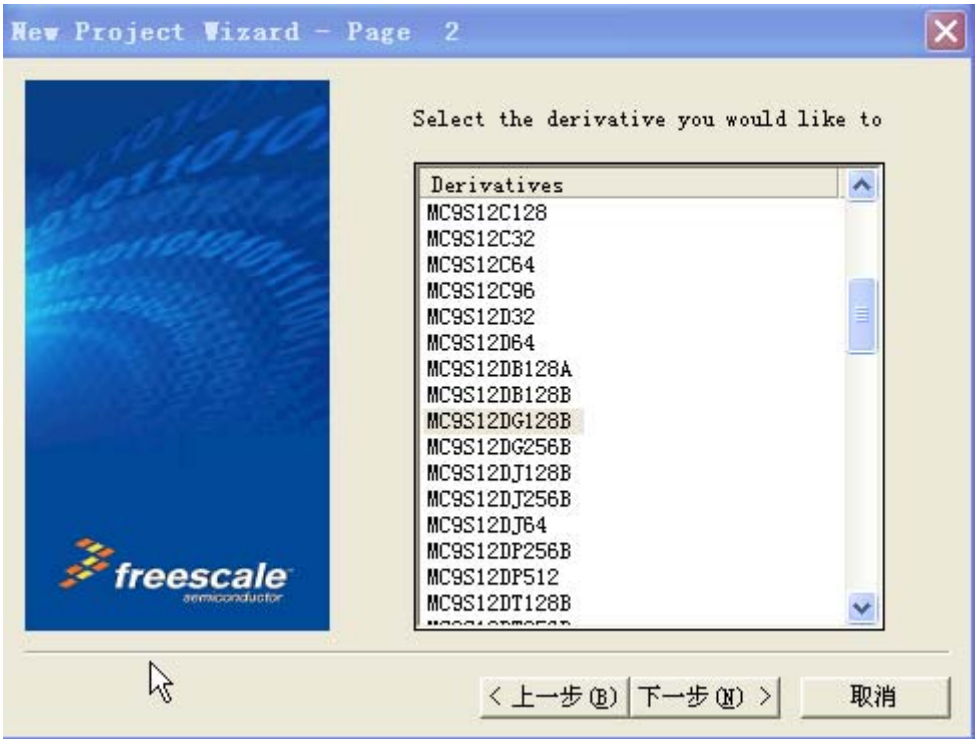

图 2-18

第二步：在上图中，点击“下一步”。出现图 2-19 所示画面。

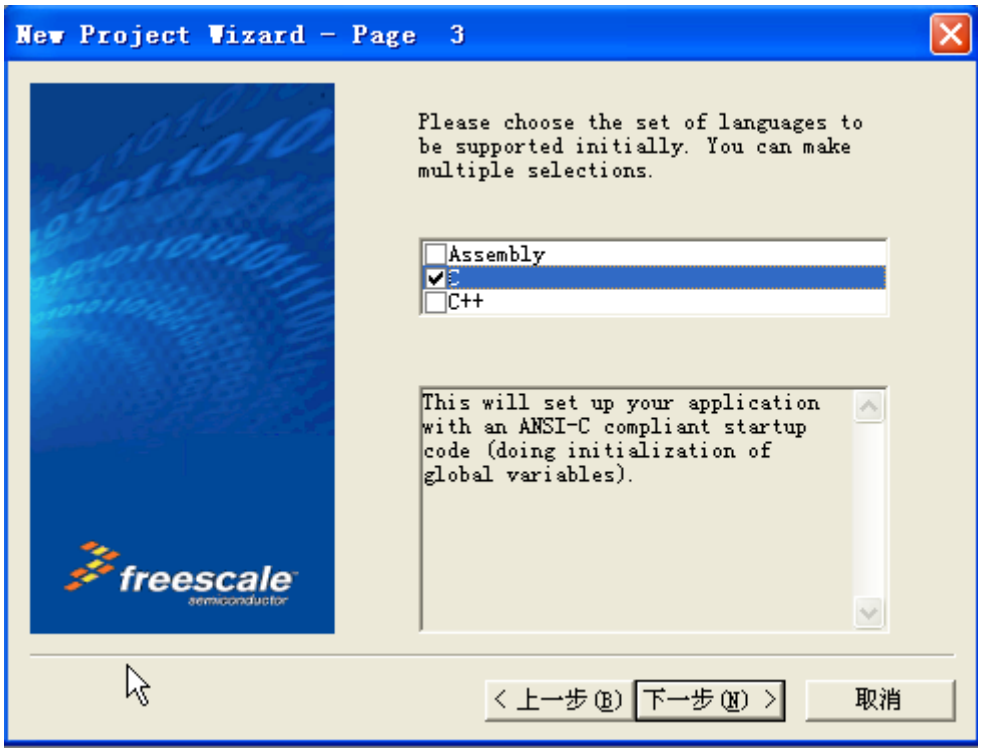

图 2-19

在图 2-19 中选择要所用的编写语言，可以有多种选择。本实验指导书以 C 语言为主，所以我们这里选择使用：“C”，如果程序中嵌入汇编，则 Relocatable assembly 项也要选中。输入要新建的工程名，本例中为“123”。选择存放的路径。单击下一步，将弹出如图 2-20 所示的窗口：

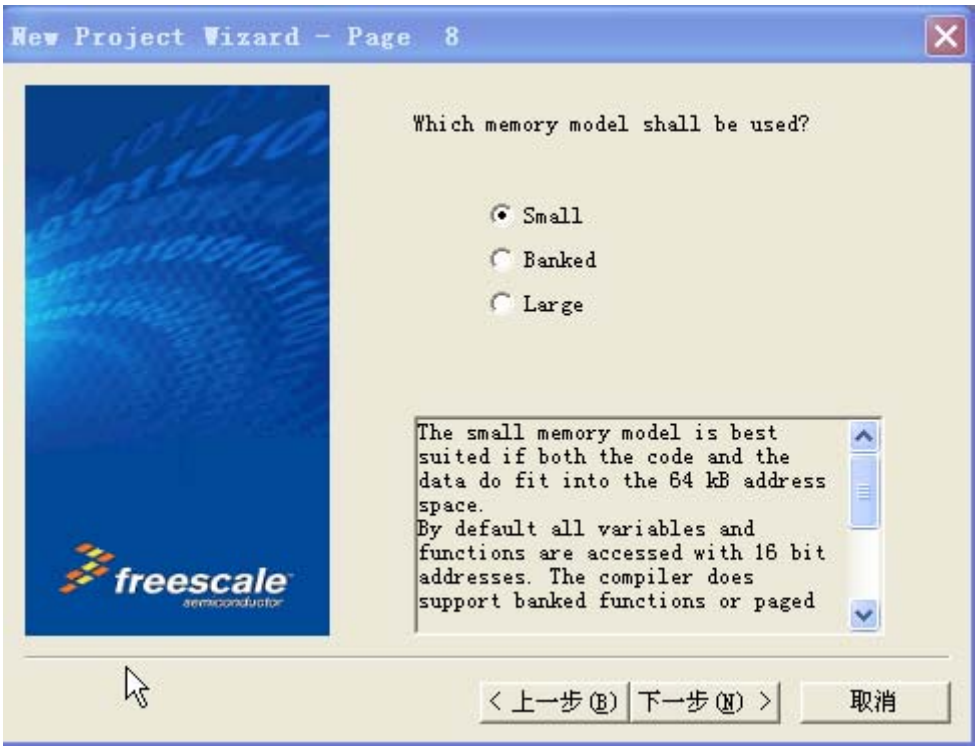

图 2-20

第三步：在上图中单击“下一步”后将弹出如图 2-21 所示的窗口，选中 TBDML.

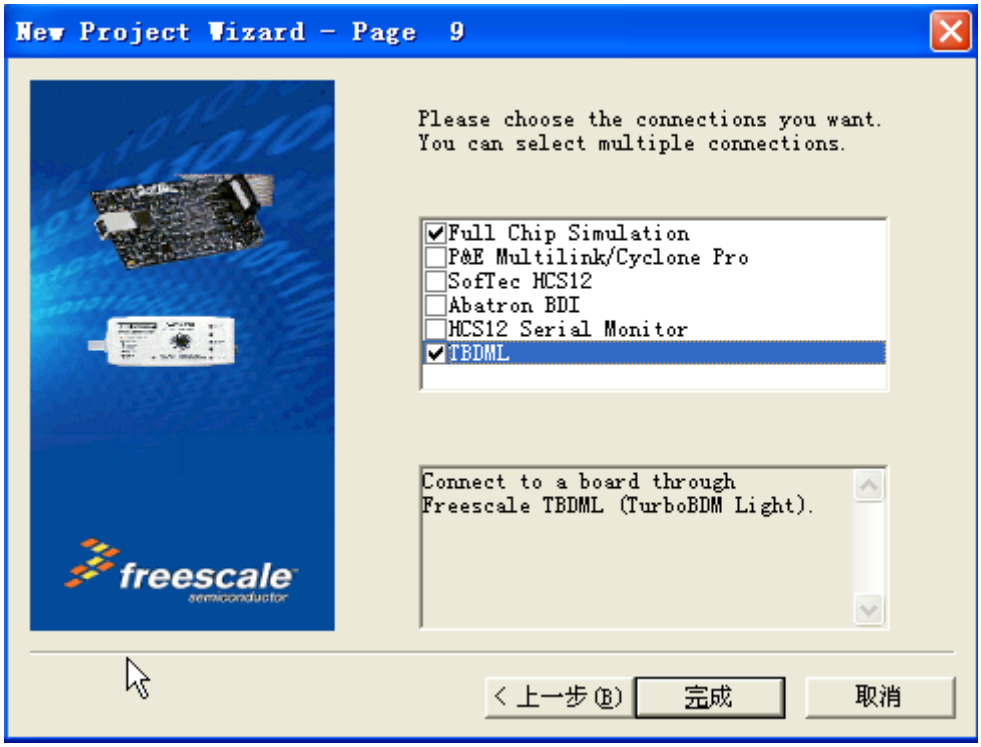

图 2-21

第四步：在上图中，保持默认选项，单击“完成”，将弹出图 2-22 所示的窗口：

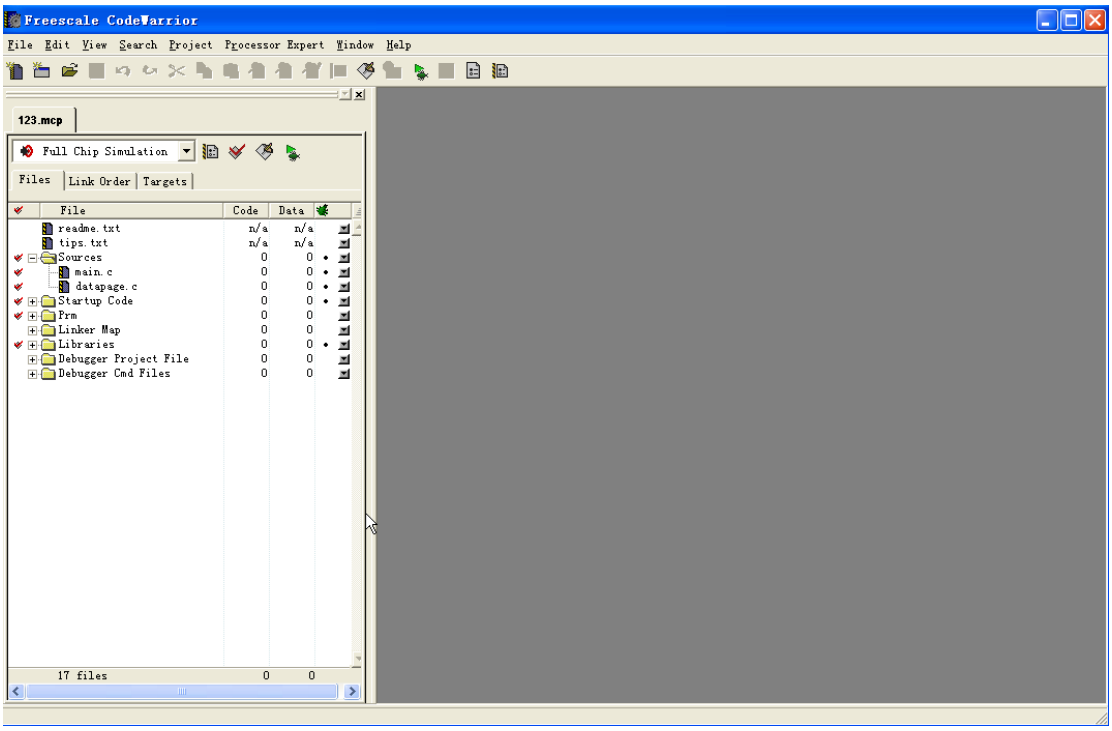

图 2-22

图 2-22 为构建成功后的窗口，打开列表框中的“SOURCE”文件夹，双击 main.c 文件，

将看到 codewarrior 自动生成的 main.c 文件，可在录入自己的程序代码。

（2）项目的编译连接

第一步：输入自己的代码完毕后，确定“编译”“Make”无误后，单击“Debug”，弹出如图 2-23：

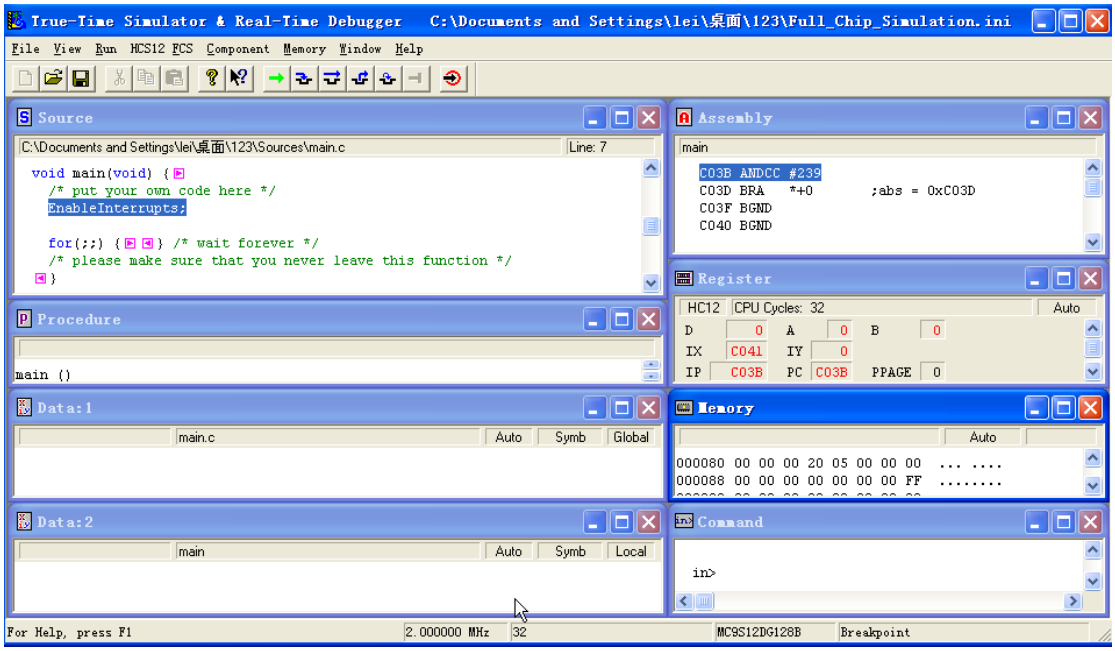

图 2-23

第二步：在图 2-23 中点击“Component”下的“ Set connection”，选择如图 2-24 所示，单击 OK 按钮。

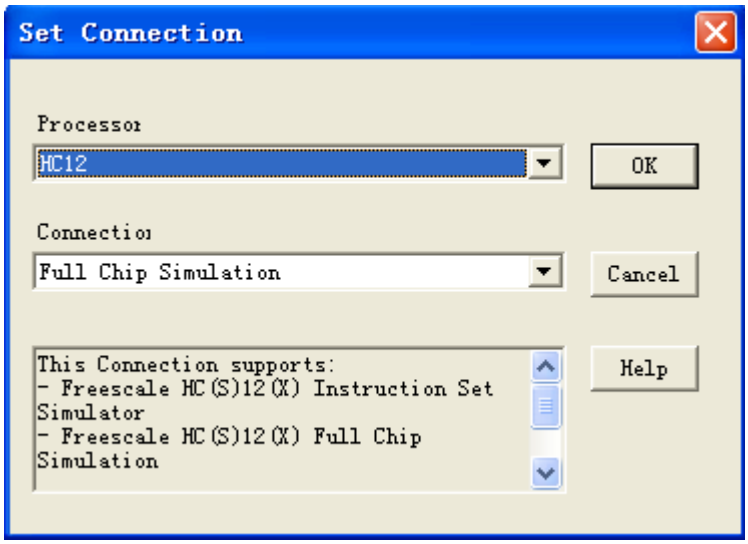

图 2-24

第三步：在菜单栏中选中“HCS12 open source BDM”命令的下拉菜单，单击“Flash”。弹出图 2-25 所示界面：

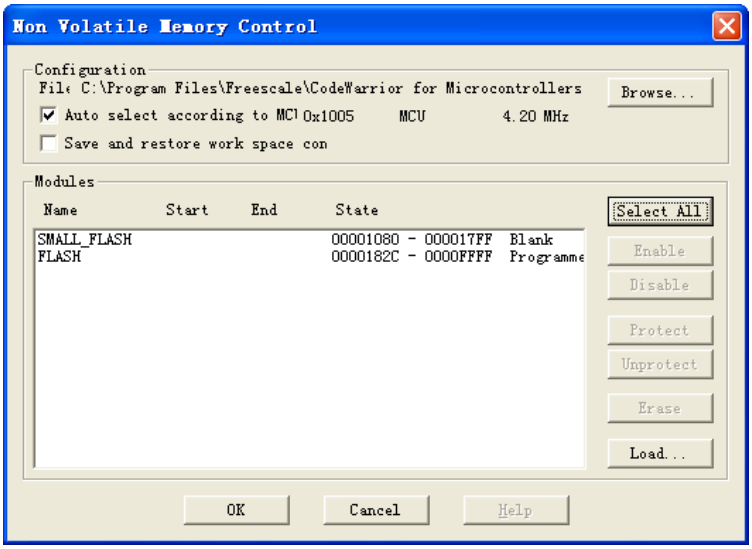

图 2-25

第四步：在图 2-25 中选中 Select All，然后单击“Erase”，OK。

第五步：在“HCS12 open source BDM”下拉菜单中选中“Load”。弹出下图 2-26：

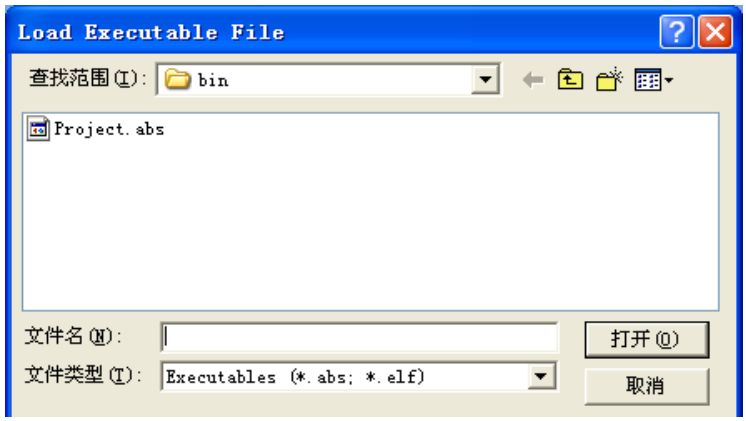

图 2-26

在图 2-26 中直接双击.abs 文件将程序下载至核心板中。

（3）软件调试

工具栏：  按钮名称从左至右依次为：“START\CONTINUE”，“SINGE STEP”，“STEP OVER”，“STEP OUT”，“ASSEMBLY STEP”，“HALT”，“RESET TARGET”。

（4）该调试界面中还有七个命令窗口，从左到右依次为 ：

1）“SOURCE”：显示程序源代码。

2）“DATE”：显示程序所用到的数据（全局变量和局部变量）。

3）“COMMAND”：显示当前的所使用的命令，例如“STEPPED”，它就是“SINGE STEP”。

4）“ASSEMBLY”：显示当前的程序的汇编语言。

5）“REGISTER”：显示当前 CPU 寄存器的各值。

6）“PROCEDURE”：显示当前所运行的程序所在程序入口，因为是程序入口为 ENTRY（），所以为“ENTRY（）”。

7）“MEMORY”：显示内存中的值。

关于 CodeWarrior 4.6 使用方法可以参考它的相关帮助文档，本处不再赘述。也可以单击实时仿真器菜单栏中的“HELP”中的“HELP TOPICS”。通过在以上各个窗口观察就可知道程序那里出了问题，将其进行改正，就可看到正确的实验现象。

3.3 超级终端的使用方法

（1）打开“开始”->“所有程序”->“附件”->“通讯”->“超级终端”，弹出如下图2-27，设置本次连接的“名称”，这里命名可以任意，此处设置为“sy1”，点击“确定”。

图 2-27

（2）接着弹出如图 2-28，选择“连接时使用：”的适当端口，一般这里选择“COM1”。不过也应该考虑串口线连接 PC 机的实际情况，然后，单击“确定”。

图 2-28

（3）弹出如下图 2-29 后设置如下，点击“确定”。

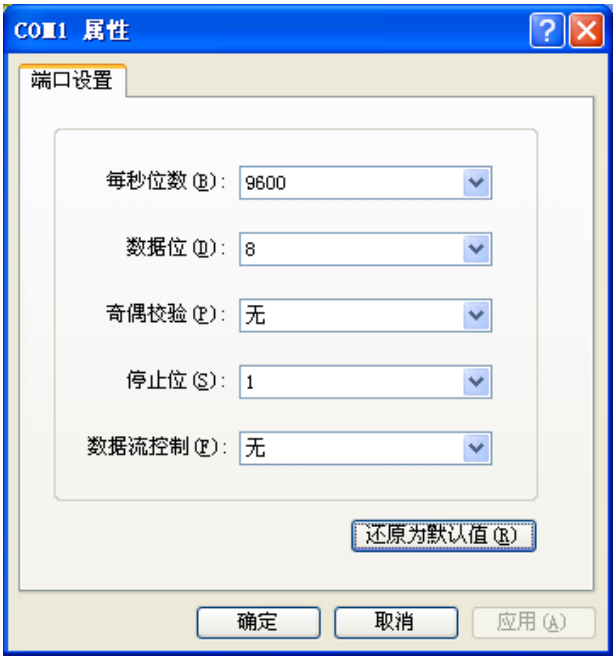

图 2-29

（4）进入如下图 2-30，超级终端设置完毕。

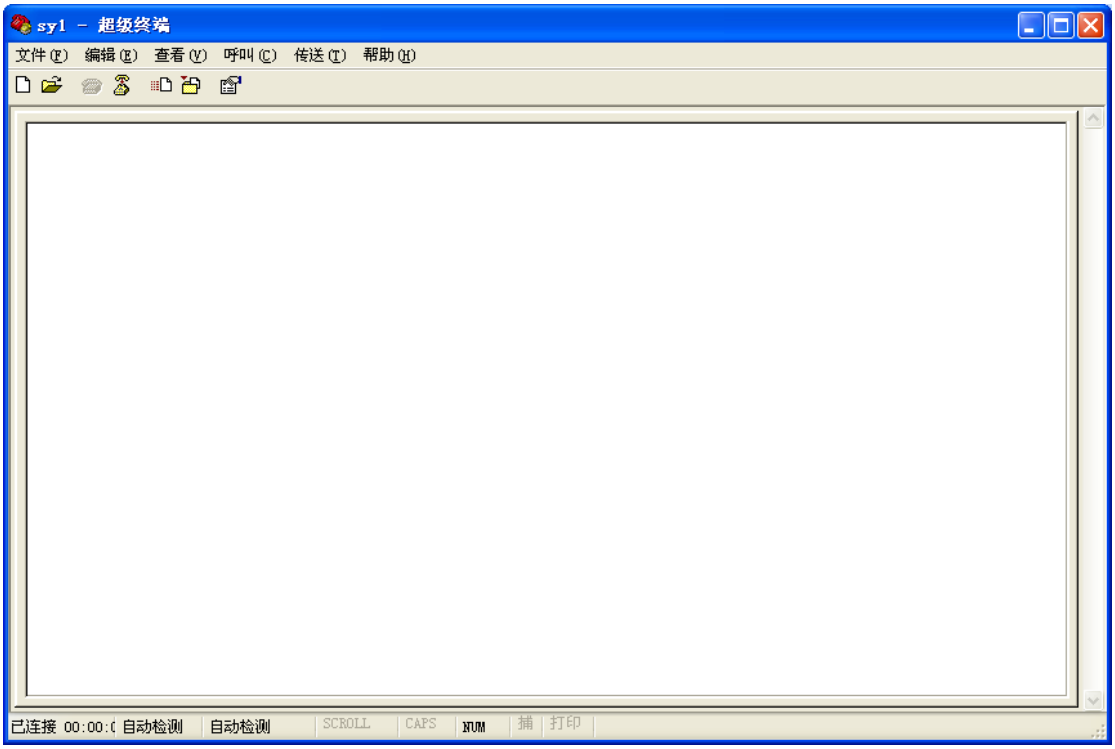

图 2-30

第4章　基本驱动程序实验

4.1　基于 S12 开发板跑马灯实验

4.1.1　实验目的

1．掌握本实验平台软硬件的使用方法。

2．掌握 MC9S12 基本工作原理、基本 I/O 端口设置及其工作特性。

3．理解和掌握 MCU 编程的基本原理及寄存器的使用方法。

4．初步理解利用汇编和 C 语言如何访问硬件。

5．掌握汇编伪指令的用法、S12CPU 指令。

6. 学会使用程序延时，并会大概估算延迟时间。

7．学会 CodeWarrior 4.6 使用方法。

4.1.2　实验要求

1. 编写一个程序，在 S12 板上 C 口高位的四个小灯实现间隔 5ms 的跑马灯。用汇编和 C 语言同时实现代码（本实验中附有参考代码）。

2. 通过本实验掌握汇编语言中如下伪指令用法：起点伪指令 ORG、赋值伪指令 EQU、字节常量伪指令 FCB、双字节常量伪指令 FDB、字符常量伪指令 FCC 等；

3.通过本实验理解汇编语言的特点，如 IO 控制到位、对延时的精准控制、延时周期的设置等；

4.学会用汇编语言如何进行系统初始化，即 PLL、堆栈、IO 的初始化，在堆栈初始化时若考虑程序是否自立，在.prm 文件中如何修改；如何用 C 语言实现本实验的功能，体会两者的不同。

4.1.3　实验准备

1．参考附件 A 中的 S12 板上板的原理图了解本实验中需要点亮的 LED 灯的端口。

2．参考第三章中 CodeWarrior 4.6 的使用方法进一步加深对该编译器的功能及生成文件的作用。

3．参考《单片机认知与实践》一书中第三章 3.2.3 汇编管理指令或其它书籍深入理解汇编语言的伪指令用法、汇编语言特点、其与 C 的区别、汇编语言如何系统初始化设置。

4．参考《单片机认知与实践》一书中第二章数字电路基础有关内容，掌握 LED 灯的工作特性。

5. 要知道延时的时间，首先要知道每一条指令的执行时间，这个时间和 CPU 的工作频率还有指令的执行周期有直接关系。同学们可参考 S12CPU 指令集。

4.1.4　实验指导

S12 应用开发板 LED 原理图：

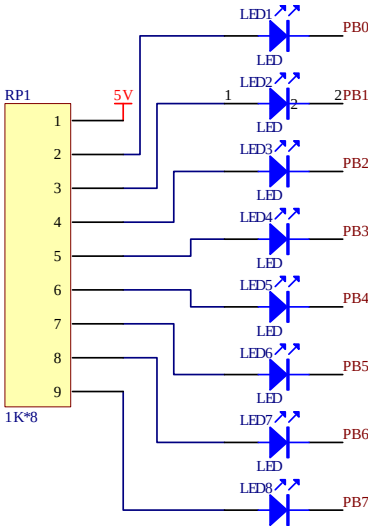

图 4-1 PORTB 端口 LED 原理图

本实验从汇编角度和 C 语言的角度来设计实现 S12 板上 C 口高四位灯点亮实现跑马灯。不过具体从哪个角度来设计本实验程序，其设计思路均如下：

（1）变量,常量和寄存器地址的定义

首先要考虑如何使用一些变量和 MCU 的映像寄存器的地址，端口地址，宏等。对于小程序我们直接在源代码中定义即可；然而对于大型程序来说，可能要用到很多模块寄存器，这时候若在源代码中定义，则会使源文件看上去很复杂，不易阅读。那么我们的解决方法就是，把这些变量和寄存器定义集中起来放到一个定义文件中，即我们所说的“头文件”。这样我们可在源文件的头部直接用”include”指令把头文件包含到源文件里，然后在程序中直接使用这些便于理解的标识。

对于汇编程序，可使用伪指令“EQU”来对要使用的寄存器和 IO 接口定义；对于 C 程序则采用预处理命令宏替换“#define”定义。汇编程序的头文件就是后缀为.inc 的配置文件或.def 定义文件；而在 C 程序中的头文件则是后缀名为.h 的文件。

为了使程序便于理解阅读，以及方便以后代码的重用。我们也可以把一些同类型子程序从主程序文件中独立出来，放到另一个.C 文件中。

（1）硬件初始化

HCS12 单片机中CPU的频率是总线频率的 2 倍，由于S12 内部时钟为 24MHz，外部总线采用 16MHz高频晶振，所以通过$f_{ICGOUT} = f_{ext} * P * N/R$公式可得N/R为 2（N=8，R=4），即内部时钟发生器控制器 1、2 设置。

SCI1BDH=0x00; SCI1BDL=0x1a;正好为 9600、串口。

（2）端口初始化

PORTB 初始化为‘0’为‘亮’、‘1’为‘暗’；DDRB 初始化为‘0’为‘输入’、‘1’为‘输出’。

（3）延时点亮

在设计延时子程序时，本实验提供两个方案即汇编延时和 C 延时，两者最大的区别是：汇编延时准确，而 C 延时的误差相对较大。

①汇编延时

采用软件延迟时，延时时间与两个因素有关，一个是微控制器总线时钟频率，一个是循环次数。由于微控制器总线时钟频率为 24MHz。首先设计一个具有较小总线周期时延的子程序 DELAY，然后再通过多次循环调用此子程序来实现较长延时。

```
DELAY:
```

```
            LDX #$0200    ;外循环次数
DELAY1:     LDY #$0500    ;内循环次数
DELAY2:     DEY
            BNE DELAY2
            DEX
            BNE DELAY1
```

②C 延时

本实验提供的 C 程序中延时子程序约为 1000 个总线周期，在这里的可将 C 程序反汇编后采用与汇编程序同样的方法计算其总延时。

（4）自立程序

自立程序就是指芯片中只有自己所编写的程序时，上电后也可直接运行无需其它工具或程序辅助。编程时把程序开始地址存放到复位地址里即可。实际上就是复位中断矢量，其实集成开发环境在连接文件.prm 中已设置了复位向量地址定义实现，也可在源程序文件的最后设置中断矢量。注意两者不要重复定义。如汇编中如下：

```
Org $fffe
Fdb main
```

在 C 语言中如下：

```
typedef void (*near tLsrFunc)(void);//声明一个无符类型的指针函数
const    tLsrFunc _vect[] @0xfffe={ //向量 FFFE 的向量地址为函数 main 的地址
     main
}
```

4.1.5 实验内容

从汇编的角度分析：

```
            XDEF Entry            ; export 'Entry' symbol
            ABSENTRY Entry        ; for absolute assembly: mark this as application
entry point


; include derivative specific macros
            INCLUDE 'mc9s12dg128.inc'
          ORG $2000
 main:
Entry:      LDAA #$ff   ;设置端口B为输出
            STAA DDRB
            SEC         ;将C=1主要是循环左移做准备
            LDAA #$FE
SHIFT:
            STAA PORTB  ;跳转到延时
            BRA DELAY
LOOP:       ROLA
            BRA SHIFT
DELAY:
            LDX #$0200    ;外循环次数
DELAY1:     LDY #$0500    ;内循环次数
DELAY2:     DEY
            BNE DELAY2
```

```
        DEX
        BNE DELAY1
        BRA LOOP
Org $fffe
Fdb main
```

4.1.6 实验步骤

1. 硬件连接且在 PC 机上安装 BDM 驱动，需参照第三章 3.1 节安装设置 BDM 驱动；并连接 BDM 线到核心板，注意接线完成后方可上电，掉电后方可进行接线的拔插。

2. 参照第三章 3.2 节 CodeWarrior 4.6 使用方法，打开本实验程序文件，编译运行代码。

3. 将.abs 文件下载到目标板上。

4. 对汇编程序调试。

4.1.7 实验思考

由于本实验用 C 两种语言来实现，体会两种语言的不同。

如何改写代码使 LED 灯可以按照自己的要求闪烁，比如实现奇偶灯等。

4.2 SCI 串行口实验

4.2.1 实验目的

1.掌握 MCU 与外界进行通信的重要方式串行通信接口。

2.掌握 SCI 的通信格式和工作原理。

3.掌握 SCI 的编程及原理，同时学习编程的规范。

4.掌握 RS-232C 的工作方式以及串行通信功能的 MCU 最小系统的硬件电路原理。

5.理解 S12 的中断机制原理。

4.2.2 实验要求

编写一个以中断方式接收字符，查询方式发送字符或字符串的程序，实现 PC 与 MCU 的通信。

在编程和测试的过程中要深入理解中断向量表的作用，以及如何添加中断到中断向量表中。

4.2.3 实验准备

在做本实验之前应该预习与串行通信接口 SCI 有关的知识。参考邵贝贝教授编著的《单片机认识与实践》第五章异步串行通信，这需要花费大概 20—30 分钟。了解串行通信的基本概念是必要的，如通信格式、比特率、奇偶校验、传输方式、RS-232C 总线标准及电平转换。了解这些概念时请考虑以下几个问题：① 传输数据时，每个字节之间是如何区分的？② 发送一位数据的持续时间数多少？③ 用什么方法知道传输是正确的？④ 可以传输多远？

SCI 发送器和接收器的工作是独立的，但使用同样的数据格式和波特率，请仔细了解一下发送器和接收器的功能，可参考《单片机认知与实践》。

本实验涉及 SCI 的有 8 个寄存器，其中 2 个比特率寄存器，3 个控制寄存器，2 个状态寄存器，1 个数据寄存器，只要理解和掌握这 8 个寄存器的用法，就可以进行 SCI 模块编程。同时可参考《单片机认知与实践》。

4.2.4 实验指导

SCI 模块硬件原理图如图 4-2：

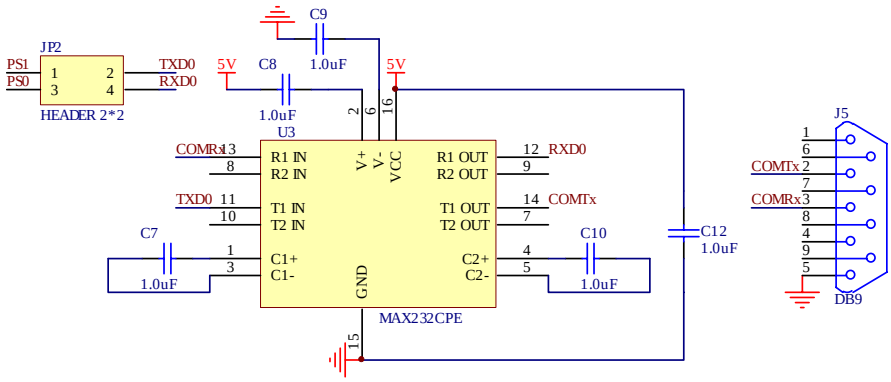

图 4-2　　SCI 硬件原理图

（1）硬件初始化是针对 MCU 编程最最基础的设置，在以后的实验中也都是必不可少的，且在实验一已经详细介绍清楚，在此及以后的实验中不再累赘。

（2）所有型号 MCU 的串行通信接口 SCI 都具有发送引脚 TxD，接收引脚 RxD，这些都是 TTL 电平引脚。要利用这两个引脚与外界实现异步串行通信，必须利用 MAX232C 芯片将 TTL 电平和 RS-232 电平互为转换来完成。

①其发送与接收过程实现如下：

发送过程是 MCU 的 14（TxD）（TTL 电平）经过 MAX232C 的 9(TxD)送到 MAX232C 的内部，在内部 TTL 电平被“提升”为 RS-232 电平，通过 7{TxDout} 发送出去。

接收过程是外部 RS-232 电平经过 MAX232C 的 8（RxDin）进入到 MAX232 的内部，在内部 RS-232 电平被“降低”为 TTL 电平，经过 10（TxD）送到 MCU 的 15（RxD），进入 MCU 内部。

该部分 SCI 通信电路可参考附录 A 中的核心板原理图。

②从 SCI 基本原理角度来看，接收时，把外部单线输入的数据变成一个字节的并行数据送入 MCU 内部；发送时，把需要发送到的一个字节的并行数据转换为单线输出。

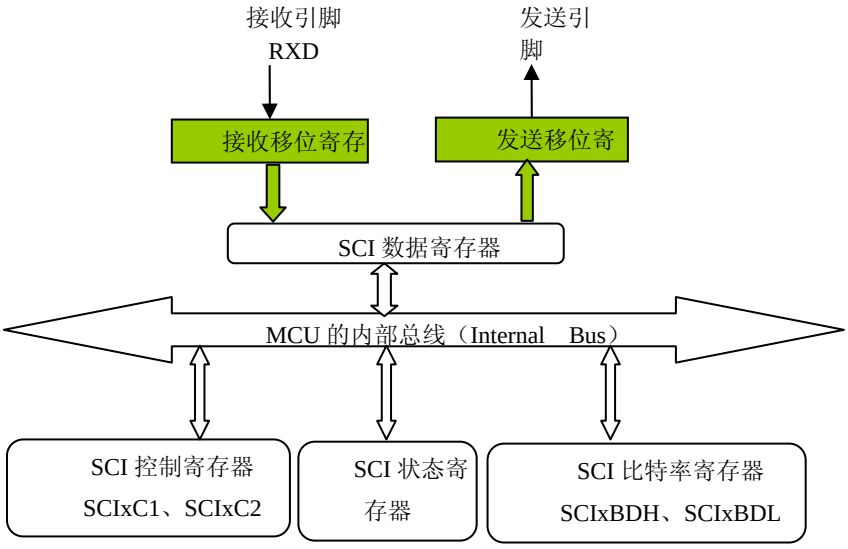

图 4-3

如图 4-3 所示，发送与接收的实际工作是通过“发送移位寄存器”和“接受移位寄存器”完成的。实际编程时，程序员并不直接与“发送移位寄存器”和“接收移位寄存器”打交道，只与数据寄存器打交道，所以 MCU 中并没有设置“发送移位寄存器”和“接收移位

寄存器”的映像地址。发送时，通过判定状态寄存器的相应位，来了解是否可以发送一个新的数据。若可以发送，则将待发送的数据放入“SCI 数据寄存器”中就可以了，剩下的工作由 MCU 自己自动完成：将数据从“SCI 数据寄存器”送到“发送移位寄存器”，硬件驱动将“发送移位寄存器”的数据按位以规定的比特率移到发送引脚 TXD，供对方接收。接收时，数据按位从接收引脚 RxD 进入“接收移位寄存器”，当收到一个完整字节时，MCU 会自动将数据送入“SCI 数据寄存器”，并将状态寄存器的相应位改变，供程序员判定并取出数据。

（3）对于 SCI 的软件开发，通常采用 8 位数据、无校验、1 个停止位、9600 波特率，用查询方式发送、接收数据。不论用查询方式还是中断方式进行串行通信编程，在程序初始化时均须对 SCI 进行初始化，主要进行：①定义比特率；②写控制字到 SCI 控制寄存器 1（SCI1C1）设置是否允许 SCI、数据长度、输出格式、选择唤醒方法、是否校验等（LOOPS、M、WAKE 和 PEN）。③写控制字到 SCI 控制寄存器 2（SCI1C2）。设置是否允许发送与接收、是中断接收还是查询接收等（TIE、TCIE、RIE、ILIE、TE、RE、RWU 和 SBK）。

不过在用查询方式从 SCI 口接收字符时，如果没有字符输入，则 CPU 将无限地查询下去。此时 CPU 完全被查询程序占用，不能再做别的事情，这是对 CPU 资源的浪费。而采用中断方式接收字符时，CPU 可以正常处理其他任务。当 SCI 模块接收到字符时，通过发出一个中断信号申请一个中断服务，接着在中断服务子程序中读取接收字符并作相应处理。因此在实际系统中采用查询方式接收字符是不太合适的。

然而对于发送字符，情况会更复杂一点。

基于上述原因，在我们在参考程序中设计了一个中断方式接收字符，查询方式发送字符或字符串的函数。

（4）如何在程序中定义中断以及如何使中断矢量就位，下面分别以 C 和汇编两种语言说明。（由于采用监控调试程序时，监控程序本身和中断矢量是受保护的，则用户的中断位置就要重定位。即此时要特别注意中断矢量的定位。而在使用 BDM 调试时，用户中断可以直接使用内存空间中的中断矢量区）

例如：在汇编中：

在代码最后加入：

```
ORG $FFD6
FDB   SCIIRQ
```

在 C 中使用关键字 interrupt 来定义中断函数，

然后把中断添加到中断向量表（在 prm 文件）中：

```
VECTOR ADDRESS 0Xffd6 get_char;
```

4.2.5 实验步骤

1..在 CodeWarrior 4.6 里对提供的程序编译，并正确下载到核心板上。

2.在编译前可以修改程序实现从串口发送 1 字符或发送 N 字符、接收 1 字符或接收 N 字符。

4.2.6 实验思考

如何用查询方式实现上述功能？如何用中断方式发送字符？串口还可以应用哪些方面？以及其重要性？

4.3 SPI 串行口实验

4.3.1 实验目的

1．掌握 S12 单片机 SPI 主从接口的工作原理。

2．掌握 SPI 串行口相关寄存器的设定方法。

3．学会运用 C 语言编写 SPI 串行口模块的方法。

4．了解母板上芯片 74HC165 和 HCT595 的功能。

5．实现通过开关来控制调试灯的亮灭。

4.3.2 实验要求

实现一个程序，通过利用 SPI 串行口通信的主从接口的工作原理，对 SPI 进行读写操作，体现出 74HC165 和 74HCT595 的功能，同时在母板上用开关进行控制调试灯的亮灭。

4.3.3 实验准备

在做本实验之前应该预习与串行通信接口 SPI 有关的知识。利用 SPI 进行 MCU 之间的数据传输时，应明确主机和从机的概念，主机的程序控制着数据传输，从机的程序必须配合主机的工作，完成传输任务。在 MCU 扩展外设结构中，仍然使用主机-从机概念，那么应该知道 MCU 必须工作在主机方式，外设处于从机方式。

SPI 模块有 5 个 8 位寄存器:数据寄存器 SPIDR,控制寄存器 SPICR1 和 SPICR2,波特率寄存器 SPIBR 和状态寄存器 SPISR。

下面以 SPI0 为例进行简单介绍 SPI 模块的寄存器。

（1）数据寄存器 SPI0DR（SPI0 Data Register）

SPI0DR 由两个独立的数据寄存器组成:只能写入的发送数据寄存器和只能读出的接收数据寄存器，它们共用一个地址$00DD。

写入时，为要发送的 8 位数据，记为：T7～T0；

读出时，为接收的 8 位数据，记为：R7～R0。

（2）控制寄存器 SPI0CR1（SPI0 Control Register 1）

SPI0 控制寄存器 1 一般情况下只能复位时写一次，以后无须再对其写入，不能更改对 SPI 的设置，SPI0CR1 的地址为$0X00D8

（3）控制寄存器 SPI0CR2（SPI0 Control Register 2)

MODFEN 位，模式错误标志位。MODFEN=1，允许 MODF 标志的设置；MODFEN=0，禁止 MODF 标志的设置

BIDIROE 位，双向工作模式输出允许位。为 1 则输出缓冲区允许；为 0 则禁止。

SPISWAI 位，等待模式 SPI 停止位。为 1 则在等待模式下 SPI 停止产生时钟信号；为 0 则在等待模式下 SPI 时钟信号正常

SPC0 位为串行控制位。

（4）波特率寄存器 SPI0BR（SPI0 Baud Rate Register），其地址为：$00DA

SPPR2~SPPR0 位，为波特率预分频选择位。

SPR2~SPR0 位，为波特率选择位。

（5）状态寄存器 SPI0SR（Status Register）

SPIF 位，SPIF 中断标志位。SPIF=1，说明新的数据到达 SPI 数据寄存器；SPIF=0，说明数据没有传输结束。

SPTEF 位，SPI 发送空的中断标志位。

MODF 位，模式错误标志位。

更多的 SPI 串行口信息在教材中有讲解。

理解 74HC165 和 74HC595 芯片原理，请参考相应的芯片手册。

4.3.4 实验指导

S12 微控制器一共有两个SPI 接口，实验板上为大家引出了一个，其引脚与PORTP 复用。PORTP0－3 是SPI1。SPI 的原理在实验教材中已经有详细的说明，在此不再赘述。SPI 是一种串行接口，但我们实际生活中所用的许多的数据量都是并行的，因此需要将其转

换为并行数据。在本实验中我们使用了74HC165 芯片来完成串行数据转换为并行数据的功能，用74HC595 完成并行数据转换为串行数据的功能。

实验板上还提供了八位DIP 开关和八位小灯，可以作为二进制数的输入输出部分。

SPI模块硬件原理图如下图4-4。

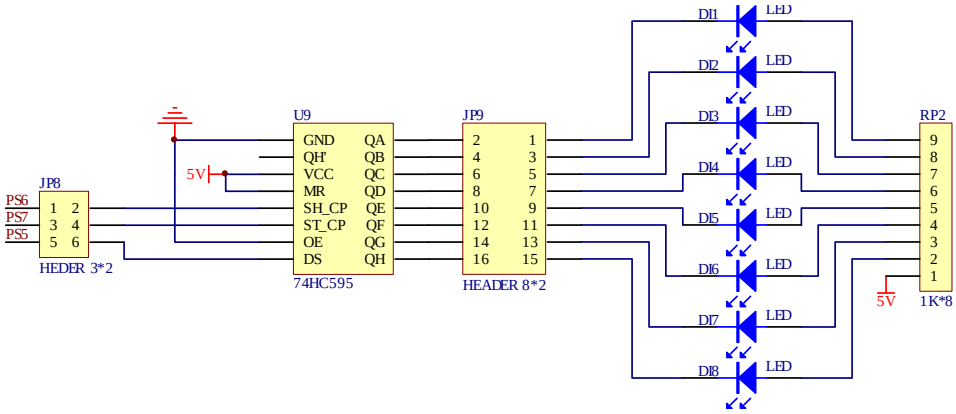

图 4-4　74HC165 原理图

八位数字量显示模块。排阻起到限流保护的功能。接入电平为高时，相应的 LED 不发光；接入电平为低时，相应的 LED 发光。

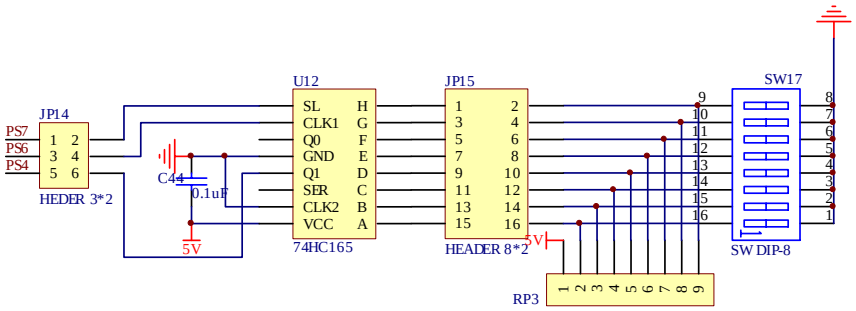

图 4-5　74HC595 原理图

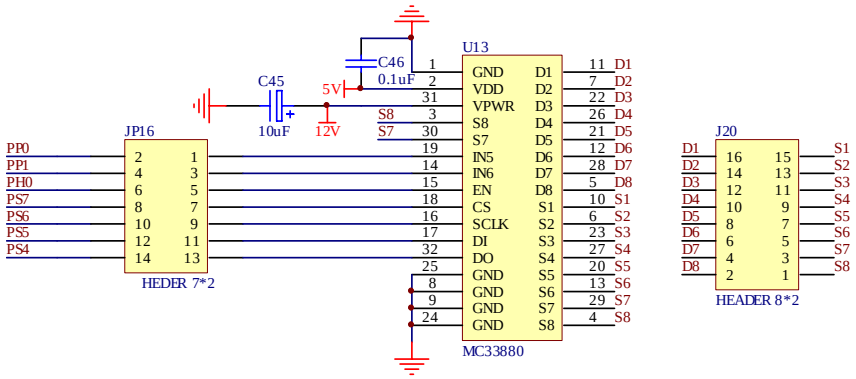

图 4-6　　MC33880 原理图

4.3.5　实验步骤

1. 通过跳线连接上图芯片。

2. 在 CodeWarrior 4.6 里对提供的程序编译，并正确下载到核心板上。

3. 将编译生成的 19 文件下载在核心板上运行后，在子母板上的开关进行拔动，对应的调试灯将显示。

4.3.6　实验思考

1. 参考 MC9S12DP256 或DG128 的DATASHEET，掌握SPI 各寄存器的使用方法。
2. 实验中将 SPI 作为主机使用，怎样正确地设置SPI 的工作模式？
3. SPI 通讯的时钟波特率应设为多少？怎样计算实际得到的时钟波特率？
4. 芯片 74HC165 接收串行数据时，需要在哪些端口的输出信号时序上做准备工作？
5. 芯片 74HC595 发送串行数据时，需要做哪些准备工作？
6. 发送数据时，如何启动数据发送
7. 接收数据时，如何判断数据是否已接收完成？

4.4　KeyBoard 输入实验

4.4.1　实验目的

1.通过本次实验掌握键盘的结构。

2 熟练运用 C 语言编写键盘中断模块程序。

3 掌握如何识别按键以及解决去抖问题。

4.4.2　实验要求

1.通过本次实验要明确键盘识别原理，以及如何去除键盘抖动。

2.编写一个独立式按键程序，可识别独立式按键中按下的键，并将其对应的按键状态用到 8 个发光二极管来显示。

3.编写一个矩阵式键盘程序，识别小键盘上按下的各键，并将其对应的键值通过超级终端和 LCD 显示出来。

4.4.3　实验准备

在做本实验之前应该预习与键盘模块有关的知识。请参考《嵌入式应用技术基础教程》的第 9 章的前 3 节内容，这大概需要花费 10—15 分钟。首先要知道什么是键盘？然后要了解键盘识别的一些基本问题，例如：如何识别键盘上的按键？如何区分按键是被真正地按下，还是抖动？第三，如何处理重建问题？

对于本实验中用到串口模块和 LCD 模块也应回顾一下其工作原理。

4.4.4　实验指导

S12 应用开发板键盘模块原理图 4-7：

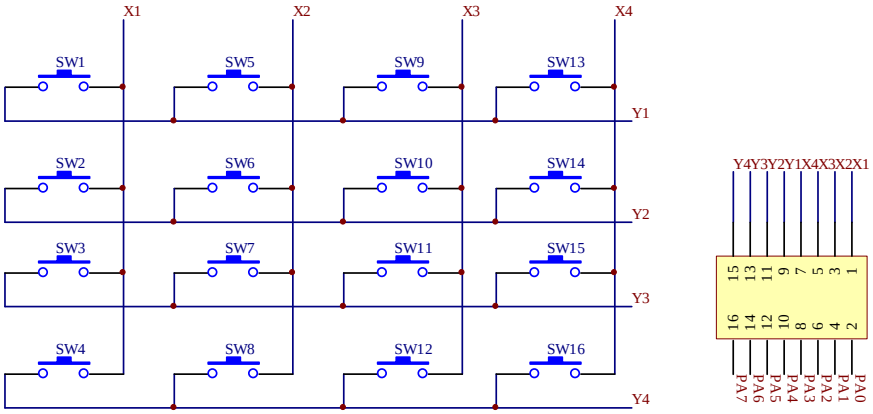

图 4-7　键盘模块原理图

S12MCU 的 PTA 口 8 根引脚可以配置成键盘接口，称为 KBI 模块，可以非常方便的实现矩阵键盘，其程序十分简单。

键盘按结构不同可分为独立式按键键盘和行列式键盘两类，每类按译码方法又可分为编码式及非编码式两种。若键盘上闭合键的识别由专用硬件实现，称为编码键盘；而靠软件实现键盘上闭合键的识别的称为未编码键盘。

独立式非编码键盘占用较多 I/O 位口（每键占一个），行列式非编码键盘用较少的 I/O 线实现较多按键，这两种键盘的检测、去抖、识别键值等工作都有软件完成。另外，由于这两种键盘的各按键间相互无制约关系，可实现多键同时按下的处理。

独立式编码键盘接口的主要部件是优先编码器，这种键盘不能实现多键同时按下的处理。行列式编码键盘的编码方法有静态和动态两种，静态法接口主要由一个行编码器和一个列编码器构成，动态法接口可采用计数器、译码器和数据选择器来构成。这两种键盘由硬件完成键的编码任务；如果接口中加入去抖电路，则软件也不用考虑按键去抖；如果接口中加入中断请求器件，则键盘的检测可以在中端服务程序中进行。

本平台提供的独立式和行列式两种非编码键盘，即需要我们用软件来识别按键。

实际上，键盘需要解决的核心问题包括扫描和去抖。

扫描就是确认我们按下的键到底是哪一个，而独立式键盘实际是 8 个按键对应 8 位 A 口中的每一位，那么我们在编程的时候直接用 8 位的二进制就可实现 8 个按键的编码值。由于是 8 个键值，那么直接用端口的值（端口引脚的状态）来表示就可以，即每个键对应一个 lED 灯。然而对于矩阵式键盘可先将编码值存放在一个合适的数组结构中以便于编程。

对于矩阵式键盘识别键盘上的闭合键，通常采用行扫描法或行反转法。

①采用行扫描法识别闭合按键时，使行线端口工作在输出方式，列线端口工作在输入方式。

先将键盘的第 0 行输出低电平，其余各行输出高电平，然后读取列值，如果列值中某位为低电平，则表明行列交叉处的键被按下，若列值全为高电平则继续扫描下一行，知道扫描到某列值有低电平或扫描完所有的行为止。可据扫描的结果判断是否有键按下和按下时哪一个行那一列的交点处的按键。

②采用行反转法识别闭合按键时，需在识别过程中改变行线端口和列线端口的工作状态。

首先使行线端口工作在输出方式，列线端口工作在输入方式，通过输出口向行线全部输出低电平，然后通过输入口读取列线值，如果此时有某一键按下，则必定会有某一列线值为低电平。然后改变两端口的工作状态，使行线端口工作在输入方式，列线工作在输出方式，其中闭合键所在行线上的值必定为低电平。当一个键被按下时，必定可读取到一对唯一的列值和行值。据这一对列值和行值就可以判断按下的是哪一行那一列交点处的键。

对于去抖的问题，即可采用硬件机制，也可采用软件机制。硬件就是加上一个电容，可使两端电压不能突变。软件机制中比较理想的是采用状态机机制（第五章的键盘任务实验中将详细介绍），也可在程序中加入一定延时，延时到键状态稳定后再读取键的状态。

根据上述分析，我们在编程时，首先是键盘系统初始化；然后是调用键盘中断子程序，键盘识别子程序和键盘编码子程序;最后还要编写延时子程序。独立式中断按键的驱动程序，其主要包含两部分。一部分是初始化，初始化中要安排好键盘的工作方式，并使之支持中断；另一部分是键盘中断服务程序，其中去抖需要适当处理。下面是键盘识别的大致流程：

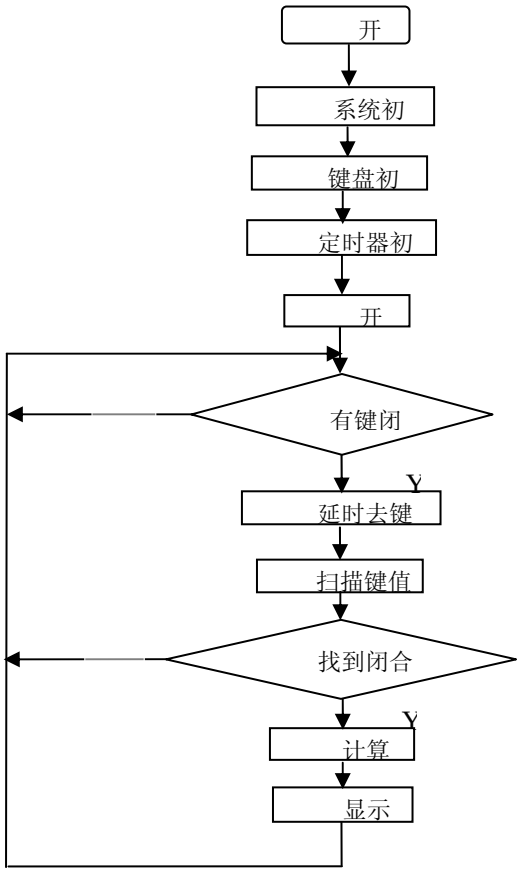

图 4-8

4.4.5 实验思考

如何 在 4*4 键盘输入状态下，结合 lCD 模块和 SCI 模块编程原理，使输入值同时显示到超级终端和液晶板上？如何利用 8 个 IO 口设计多于 16 个键的方法。

4.5 动态数码管显示实验

4.5.1 实验目的

1．了解 LED 模块相关信息及其与各端口的相连。

2．掌握 LED 模块的使用方法。

3．学会使用 C 语言来编写显示 LED 程序。

4.5.2 实验要求

用 C 语言引编写一段程序，其功能使 8 位动态数码管上分别从左向右开始显示 0~9 数据，呈现跑马灯效果。

4.5.3 实验准备

（1）8 段数码管一般由 8 个发光二极管（Llight-emitting diode，LED）组成，每一个位段就是一个发光二极管。一个 8 段数码管分别由 a、b、c、d、e、f、g 位段，外加上一个小数点的位段 h（或记为 dp）组成。实物外型见图 4-9。

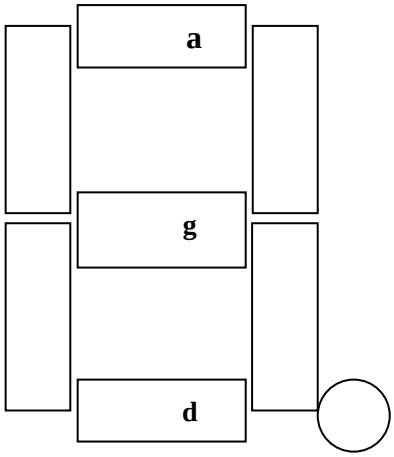

图 4-9　数码管外形

（2）8 段数码管可分为共阳极和共阴极两种，如图 4-10 所示。共阴极 8 段数码管的信号端高电平有效，只要在各个位段上加上相应的信号即可使相应的位段发光，共阳极的 8 段数码管则相反，在相应的位段加上低电平即可使该位段发光。

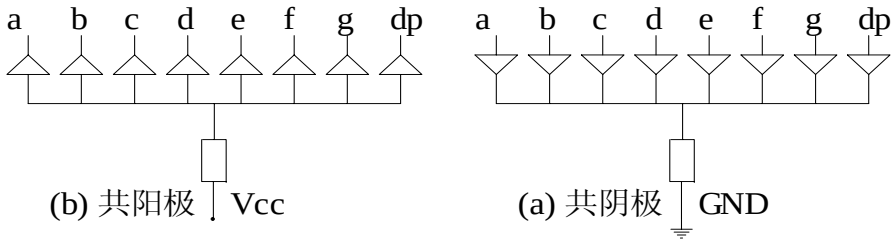

图 4-10

(3)连排数码管，利用 CS5 、CS4、CS3、CS2、CS1、CS0 控制各个数码管的位选信号，每个时刻只能让一个数码管有效，即 CS5 、CS4、CS3、CS2、CS1、CS0 只能有一个为 0，例如令 CS3=0，其它的全为 1，则数据线上的数据体现在第一个数码管上，其他则不受影响。要让各个数据管均显示需要的数字，则必须逐个使相应位选信号为 0，其他位选信号为 1，并将要显示的一位数字送到数据线上。这种方法叫“位选线扫描法”。虽然每个时刻只有一个数码管有效，但只要延时适当，由于人眼的“视觉暂留效应”(约 100ms 左右)，看起来则是同时显示的。

了解更多想阅读相关的教材。

4.5.4　实验指导

S12 应用开发板动态数码管原理图见附录 A，下面是动态数码管接口原理图：

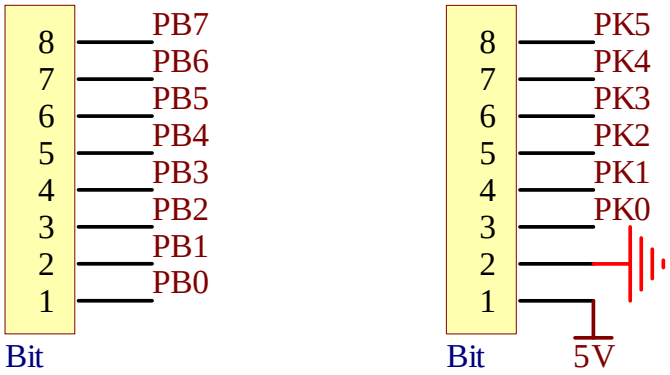

图 4-11

4.5.5 实验步骤

1．在 CodeWarrior 4.6 里对提供的程序编译，并正确下载到核心板上。

2．将编译生成的 S19 文件下载到核心板上运行后，母板上的 LED 模将会被驱动，从左边开始显示 0~9，并且向右边移动，形成跑马灯效果。

4.5.6 实验思考

如何理解“动态数码管”中“动态”的概念？

4.6 液晶显示实验

4.6.1 实验目的

1 掌握 LCD 模块的 C 语言编程方法。

2.理解点阵字符型液晶显示器的工作原理。

3.掌握启动代码的工作过程以及连接文件的设置。

4.6.2 实验要求

1. 定义一段想在 LCD 上显示的字符，通过 LCD 模块的功能，从而在 LCD 显示板上显示定义的字符。

2.在第一个要求的基础上，编写一个采用中断方式向 LCD 模块和串口同时发送字符的程序，即可在 LCD 显示板上和超级终端同时显示从 PC 机的键盘上发送的字符。

3.通过本实验，掌握针对 LCD 模块编程时，具体怎么进行功能设置，要求提供一些什么样的接口函数。

4.6.3 实验准备

在预习实验的时候，首先需要掌握本次实验采用的 LCD 模块芯片 HD44780 的结构、功能和特点，以及 HD44780 引脚的功能与时序信息。具体知识可以上网查询 HD44780 的芯片手册。可参考《嵌入式应用技术基础教程》的第 13 章或者李晶皎《液晶显示器的 C 语言程序设计》的第 3 章。这需要花费大概 20—30 分钟。了解一下 LCD 的制作工艺也是必要的，如 LCD 的基本结构是什么？如液晶为什么会透光？为什么要在面板上加储存电容？LCD 的分类方法有那几种？LCD 内部结构是什么？HD44780 指令集中各条指令的功能？

本实验主要涉及 HD44780 指令集中的清屏命令，这一命令也是最常用的命令。对于其他命令大家也应该掌握，以便实现想要的更多功能。

本实验还用到了上一个实验的串口模块功能，大家可再回顾一下 SCI 通信原理。

4.6.4 实验指导

LCD 模块硬件原理图：

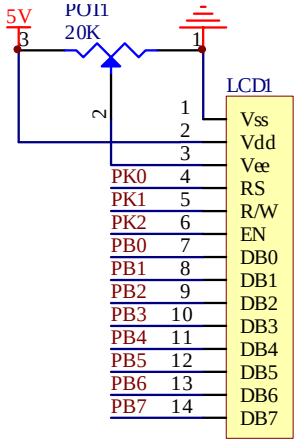

图 4-12　　LCD 硬件原理图接口

本次实验要用到的液晶模块是专门用于显示字符、数据、符号等的点阵型液晶显示模块。HD44780 的外部接口信号一般有 14 条，其中与 MCU 的接口有 8 条数据线、3 条控制线。本实验采用的 HD44780 型 LCD 模块的主要引脚功能说明如下：

1) RS 输入寄存器选择：1 数据寄存器，0 指令寄存器；
2) R/W 输入读写操作选择：1 读，0 写；
3) E 输入使能信号，R/W= 0，E 下降沿有效，R/W= 1，E 高电平有效；
4) DB0~DB7 八条数据总线。

首先我们要明确 LCD 模块中各个寄存器的功能实现原理。

从编程角度看，HD44780 内部主要由指令寄存器（IR）、数据寄存器（DR）、忙标志（BF）、地址计数器（AC）、显示数据寄存器（DD RAM）、字符发生器 ROM（CG、ROM）、字符发生器 RAM（CG RAM）及时序发生电路构成。

① 44780 有 IR 和 DR 两个寄存器，MCU 是通过这两个寄存器来对显示模块进行控制的，启动 HD44780 的内部操作前，控制信息都存储在 IR 和 DR 中，进而与 MCU 进行通信。IR 存储指令代码，DR 存储 MCU 要写到 DDRAM 和 CGRAM 的数据，这些数据的写入由内部操作自动完成。

② 要显示的字符或数字等信息在芯片内部都是有专门的字符代码，即在每个显示芯片内部都有一个字符库。了解字符库之后再来了解一下字符发生器 ROM（CGROM），它是已经固化在 HD44780 内部的字模库，MCU 只需写入某个字符的字符代码，HD44780 将以其作为字模库的地址将该字符输出给驱动器显示。编写程序时需要用到字符库。字符发生器 RAM（CGRAM）是提供给用户自造特殊字符使用的。

③ DDRAM 是显示数据寄存器，存储 8 位字符代码表示的显示数据。

④ HD44780 读写时序是很重要的如果片选信号以确定，时序不到相应的读写也是不能完成的。HD44780 读写时序是由使能信号 E 完成的，E 信号是正脉冲信号，不操作时为低电平状态，操作时产生一个正脉冲。

HD44780 将依据 R/W 信号端的电平状态确定读操作还是写操作：R/W=0 时为写操作；R/W=1 时为读操作。R/W 信号的宽度必须要大于 E 信号的宽度才能保证 MCU 正确操作。同时 HD44780 依据 RS 端的电平确定总线上的数据是指令代码还是显示数据 RS=0 选通指令寄存器通道，数据总线传输的是指令代码或标志位；RS=1 选通数据寄存器通道，数据总线传输的是显示数据或自定义字符的字模数据。

通过上述分析我们可知 LCD 的工作原理如下：

首先设置信号，应该先设置 RS、R/W 的状态，接着设置数据，最后复位 RS、R/W 的

状态；同时 HD44780 芯片将以该字符的字符代码作为字模库的地址将该字符输出给驱动器显示。

在进行读操作，且 E 信号处于高电平时，控制器将所需 DR 中的数据送到数据总线上，供 MCU 读取；在进行写操作，且 E 信号的下降沿处时根据片选 RS 的状态将数据总线上的数据写入相应的寄存器中，即所显示的字符写到数据寄存器 DR 中，控制命令写到控制寄存器 IR 中。同时 DR 中的数据由内部操作自动写入 DDRAM 或 CGRAM 以供驱动器显示。在整个编程过程中我们要注意跟时序的配合。在编程时可通过修改相关指令码来实现很多功能的任务，如清屏、显示单行或双行、以及画面显示开或关状态、光标或画面滚动等。

由于 HD44780 芯片需要花费一定的时间处理发送给它的命令或数据。由于这个原因，在发送一个命令或数据后，在发送下一个命令或数据前，需要等待指定的时间。注意，HD44780 本身允许 MCU 读“忙(BUSY)”的状态。MCU 读出该状态以便确定 HD44780 是否准备接收另一个命令或更多的数据。BF=1 时，HD44780 正在进行内部操作，不能接受外部指令或数据。写入下一条指令之前必须确保 BF=0。忙标志 BF 提供了 HD44780 是否正准备接收另一个命令或更多数据的真正的信息。

根据上述分析，针对 HD44780LCD 模块我们在参考程序中设计了 7 个显示的与其相关的关键子程序。

首先是液晶显示初始化和液晶显示使能的子程序，然后是写命令，写数据，写地址，写字符，写字符串等子程序。下图是参考框图 1。

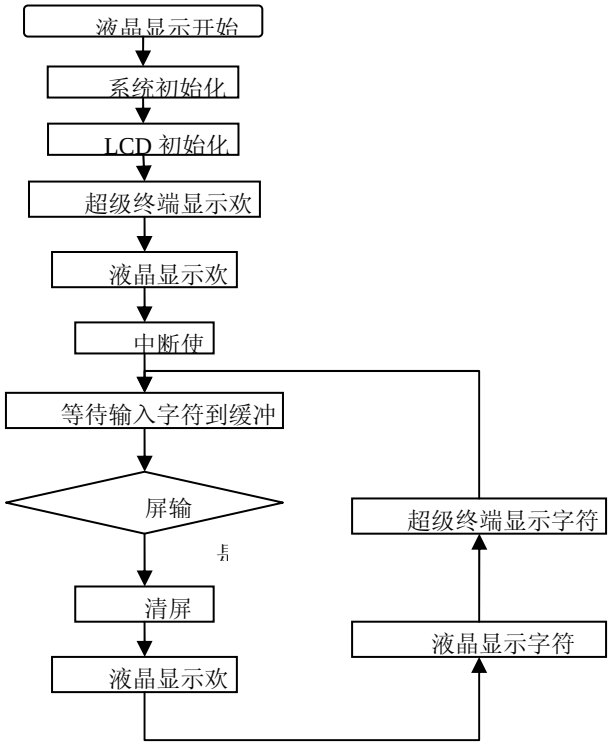

图 4-13

4.6.5 实验思考

本实验是通过 PC 键盘输入字符，那么如何实现利用本平台提供的键盘模块输入字符，然后在液晶显示板上显示？如何在 LCD 上实现字符的动态显示？

4.7 定时器模块实验

4.7.1 实验目的

1. 学习 S12 系列单片机内部定时器的使用方法。
2. 理解定时器管理器的工作原理。
3. 掌握使用定时器模块的方法。
4. 学会运用 C 语言编写简单的定时程序。

4.7.2 实验要求

本实验要求学生在理解定时器管理器工作原理和内部定时器使用方法的基础上，能够独立运用 C 语言编写简单的定时程序，设置控制和状态寄存器各位的值使其达到预期的目的。进而深刻理解定时器在该实验中是如何实现定时功能的。

任务一：本实验主要体现输入捕捉功能用来监视外部的输入信号，用导线将 PT0 与 JP5 对应的开关相连，拔动开关产生电平变化时，PT0 将捕捉发生的跳变（上升沿，下降沿都捕捉），利用中断处理将得到电平发生变化的时刻。实验设置定时器时钟=总线时钟/128，电平发生变化的时刻为：溢出次数*（$FFFF*定时器时钟周期）+此时 TC0 的值*定时器时钟周期

任务二：本实验主要任务是用程序的方法在规定时刻输出需要的电平，实现对调试灯的闪熄控制。实验设置为：定时器时钟=总线时钟/128，当 TC 设置为 FFFF 时（与自由计数器比较），电平输出时刻为$FFFF*定时器时钟周期，相当 0.35S,故调试灯将隔 0.35S 发生变化。

任务三：本实验主要的任务是实现脉冲累加器记录输入引脚上发生的有效边沿事件的次数，用导线将 PT7 与 JP5 对应的开关相连，当拔动开关产生的低电平到高电平时（上升沿有效）将触发脉冲累加器计数加 1，其触发的次数将在超级终端上显示出来。

任务四：本实验主要模数递减计数器作为时基定时产生中断，从而实现对调试灯定时的控制。实验设置模数递减计数器输出的时钟分频系数为 16，这样定时器时钟=总线时钟/16；当 MCCNT=0XFFFF 时，时基定时时间为 0XFFFF*定时器时钟周期*中断次数，相当于 0.87S，故调试灯将隔 0.87S 发生变化。

4.7.3 实验准备

在做本实验之前应严格要求自己把 S12 数据手册有关定时器部分的相关内容预习一遍。参考邵贝贝教授编著的《单片机认识与实践》第六章单片机片内 I/O 模块中的定时器一节了解定时器管理器的基本功能和内部结构。熟悉定时器管理器有关的概念。如：输入捕捉、定时输出比较、脉冲累加器、模数递减计数器等。在预习过程中应思考中断频率和总线频率、分频因子、计数器预置值的关系？思考定时器的作用在该模块中具体实现的方法和定时器在该试验中的工作原理？

熟悉一下系统实时中断状态和控制寄存器。

4.7.4 实验指导

16 位的自由运行计数器在单片机复位结束后从$0000 开始连续计数。当达到最大值$FFFF 时，翻转到$0000，同时将状态寄存器的溢出位 TOF 置 1，然后从新开始计数。如果允许定时器溢出中断，将会得到一次定时器的溢出中断服务；如果使用预置计数，则可以得到更精确和便于计算的溢出中断。

预置计数寄存器是一个 16 位寄存器，程序向这个寄存器写入一个确定的数值，则计数器每进行一次计数都会将计数与这个寄存器的值进行比较。如果相同就产生溢出，计数器翻转到 0 并重新开始计数。也就是说此时自由运行计数器没有计数到$FFFF 就产生溢出并重新计数。从单片机的角度看，一个单位时间就是定时器的自由运行计数器计数的时间间隔，所以在任何时候都可以通过读取计数器的值而判断经过了多长时间。该实验就是根据这一特点

通过使用预置计数实现定时器在某一单位时间内实现定时中断。

定时器的状态和控制寄存器的各位都有不同的作用，在设定时可以只对需要更改的某一位进行读/写，也可以对整个寄存器进行读/写。在系统初始化时，一般按照下属步骤进行状态和控制寄存器的设置：

（1）根据需要确定各位的值；

（2）由各位的值得到整个寄存器的值；

（3）将这个值写入控制和状态寄存器就可以完成各位的设定。

S12 系列 MCU 的定时器模块在 HCl2 的标准定时器模块(Standard Timer Module，TIM)基础上增加了一些新的功能，加强了原来的输入捕捉及脉冲累加器的一些功能，扩大了应用范围，因此 S12 的定时器模块称为增强型定时器模块(Enhanced Capture Timer Module，ECT)，其结构框图如下图所示。ECT 功能相当于高速的 I / O 口，由一个 16 位自由运行计数器、8 个 16 位的输入捕捉 / 输出比较通道、2 个 16 位脉冲累加器以及一个 16 位模数递减计数器(MDC)组成。

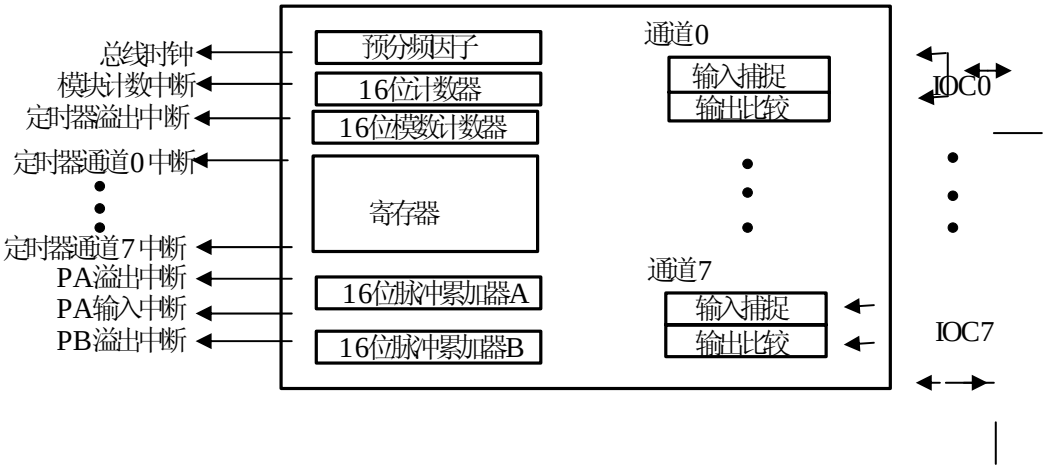

图 4-14　定时器接口模块功能框图

S12 增强型定时器模块 ECT 具有如下特点：

8 个具有 16 位缓冲寄存器的输入捕捉通道；

4 个 8 位脉冲累加器可以通过级联形成 2 个 16 位的脉冲累加器；

1 个具有 4 位预分频器的 16 位递减模数计数器；

4 个可选的延迟计数器用来增强输入抗干扰能力。

ECT 实际上是一个 16 位的可编程计数器，它的基本时钟频率可以通过预分频器设置，用于产生波形输出、测量输入波形、统计脉冲或边沿个数以及作为时间基准。它也可以在不需 S12MCU 干预的情况下产生脉宽调制输出。它不仅具有一个 16 位递减模数计数器，而且是一个功能完善的定时器，具有自动重装载和中断能力，又可为 IC、PAL 寄存器向保持寄存器的传送提供定时控制信号。MDC 还可作为具有定时中断功能的独立时钟基准。

了解更多关于定时器模块，请阅读相关资料介绍。

对于实时中断机制，编程时主要是设置一下系统实时中断状态和控制寄存器。

4.7.5　实验思考

三种实现计数和定时方式：（1）完全硬件方式（2）完全软件方式（3）可编程计数器/定时器的优缺点。

4.8　A/D 转换实验

4.8.1 实验目的

1．掌握 A/D 模块的功能。

2．了解 S12 单片机 A/D 模块的使用方法。

3．学会使用 C 语言对 A/D 模块进行编程。

4．掌握嵌入式 C 语言对 ATD 模块的编程技巧。

4.8.2 实验要求

1．利用 S12 的 ADC 模块将一路模拟电压转换成数字，通过小灯表现出来；或者在 LCD 上显示出来，建议使用两种显示方式；

2．实现连续转换和结果显示；

3．实现多路转换和结果显示；

4.8.3 实验准备

1．理解和掌握 MC9S12 模数转换的基本概念和特点。

掌握模拟数字转换模块的特征，例如：在 10 位转换精度及 2MHz 的转换频率条件下，单次 A/D 转换只需 14 微秒；数据结果左/右对其；转换完成标志或转换完成产生中断；8/10 位可选转换器；左对齐有符号数据模式；模拟输入多路器多达 8 个模拟输入通道；单一或连续转换模式。

掌握 ATD(模拟数字转换)的两种操作模式、ATD 的中断、复位、采样以及功能。详细内容请参考 S1260 芯片手册第十四章（2—5）小节。

2．理解和掌握解与 S08 模拟数字转换（ATD）模块相关的寄存器和控制位。

在做这部分实验时，要理解以下几个最基本的寄存器：

ATDCTL2：控制寄存器。主要设置A/D 标志位清除方式、A/D 采样触发方式、是否允许A/D 采样完成中断等。

ATDCTL3：控制寄存器。主要设置每次A/D 转换采样几路电平、采样结果的存储方式等。

ATDCTL4：控制寄存器。主要设置A/D 转换精度、A/D 转换时钟频率等。

ATDCTL5：控制寄存器。主要设置A/D 转换结果的对齐方式和数据类型，以及A/D 的采样模式（连续采样/单词采样，顺序转换/单通道转换等）

ATDSTAT0：状态标志寄存器。包括A/D 转换完成标志，出错标志、转换结果存储索引等标志位

ATDTEST1：测试寄存器。

ATDSTAT1：标志寄存器。标识一次A/D 转换中各通道的完成情况。

以上寄存器的具体内容和其他与ADC 模块相关的寄存器请参看datasheet 相应章节。

在本实验中，用到了这些基本寄存器的每一位的功能，需要掌握寄存器每一位的功能是什么？详细内容请参考 S12 芯片手册第十四章第六小节“模数转换寄存器和控制位”。

3．试验中用到的硬件电路主要是电位器的应用，如图 4-15 所示：

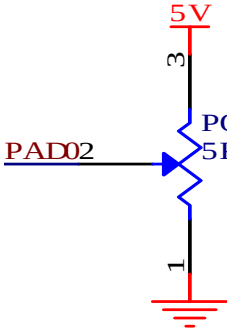

图 4-15　电位器模块

在图 4-15 中可以知道，必须给电位器提供一个 3.3V 的基准高电压，另外要提供一个基准低电压接地，通过调节电位器实现不同电位的输入。调节过的电压通过 ADC0 和 ADC1 两个通道输送到 S12 的 ATD 模块中去，模拟电压经过转换变成数字量，通过 B 口的小灯显示出来。

嵌入式 C 语言和 C 语言还有一定的区别，需要学习《嵌入式微控制器》这一本书中的第四部分“微控制器 C 编程”。

4.8.4 实验指导

在该实验中，模拟输入信号的参考基准电压是 3.3V，通过原理图我们看以看出，3.3V 这一参考基准电压应连接在参考基准高电压（VREFH）引脚上，参考基准低电压（VREFL）引脚应接地。这样就能给电位器提供一个可应用的基准电压。

经过调节电位器，可以决定要输入 ATD 模块的电压是多大，ATD 可以把输入的模拟电压转换成二进制数字，通过小灯表示出来，小灯亮的为 0，不亮的为 1。当我们手动调节电位器时，小灯的亮灭会发生相应的改变。

在该实验中，首先要定义时钟发生器，以确定用什么样的时钟模式；然后要保持这种时钟模式以及初始化 ATD 模块；最后打开中断、执行循环；在循环中的第一条语句就是选择一个通道，并且执行该通道上的转换，然后程序要执行一定的延时，因为转换需要一定的时间。在初始化 ATD 模块时，要给模拟数字转换引脚时能寄存器赋值，赋值的目的是选择要使用的模数转换通道；然后给模数转换控制寄存器赋值，赋值的目的是设置转换的状态；最后给模数转换状态和控制寄存器赋值，在此赋值的目的是选择应用通道的同时打开该通道的转换中断功能。

一般情况下，模拟信号在输入 ATD 模块的时候，首先要经过多路输入选择器，此时，通过三位译码器来选择要转换的通道。在 ATD 中使用逐次逼近寄存器（特区）架构。ATD 包含所有必要的执行一个单一信号模数转换器的基本原理。写 ATD1SC 寄存器开始一个新的转换。写 ATD1C 寄存器会打断目前正在执行的转换，但是不能开始一个新的转换。写 ATD1PE 寄存器也会终止目前正在执行的转换，但是也不能开始一个新的转换。当写 ATD1SC 寄存器完成时，现在正在运行的转换将被终止，同时将会开始一个新的转换。

ATD 有一个模式控制单元，这个控制单元与样本持有机制和逐次逼近寄存器机制联系，必要时收集样本和执行转换。模式控制单元发出信号，样本持有机制开始收集采样，逐次逼近寄存器机制开始接受样本。在接受样品期间结尾时， 样本持有机制发信号逐次逼近寄存器机制开始从模拟式到数字式的转换进程。当逐次逼近机制寄存器向模式控制单元发送结束信号时，转换过程被结束。对于 V_{REFL} 和 V_{REFH} ，逐次逼近机制寄存器使用本身内设置的信号电平作为参考，不必依赖样本持有机制的传送。

模式控制单元组织模数转换，指定样本输入放入的通道和把转换后输出的数据从逐次逼近机制寄存器移动到结果寄存器。结果寄存器由一个双端口寄存器组成。逐次逼近机制寄存器通过一个端口向结果寄存器写数据，同时单位总想通过另一个端口把数据读出。

样本持有机制接收模拟信号，当逐次逼近机制寄存器机制内的存储节点电容充电时保存它们。一次只能保存一个采样，所以样本持有机制和逐次逼近机制不能同时运行，即使它们被独立的配置。当样本持有机制不采样时，它会禁止自己的内部时钟。模拟输入信号是单极的。信号电压必须在 V_{SSAD} 和 V_{DDAD} 潜在的范围内。样本持有机制不要求执行特殊的转换。具体的内容请参考 S12 芯片手册。

4.8.5 实验结果

1．如何配置相关的寄存器？

2．怎样选择通道、怎样打开通道？

3．在实时调试中如何设置断点？在程序那个地址设置断点能够检验转换是否正确？

4.8.6 实验思考

1．如何配置相关的寄存器？

2．怎样选择通道、怎样打开通道？

3．在实时调试中如何设置断点？在程序那个地址设置断点能够检验转换是否正确？

4.9 PWM 模块实验

4.9.1 实验目的

1．掌握 PWM 模块的基本功能。

2．掌握 S12 单片机 PWM 模块所使用的硬件电路及使用方法。

3．掌握嵌入式 C 用语言的编程方法。

4.9.2 实验要求

1..使用单片机内部的 PWM 模块使蜂鸣器按一定的占空比工作。

2.通过对 PWM 占空比的调节实现蜂鸣器发不同频率的声音。

4.9.3 实验准备

首先要理解和掌握 PWM 的基本概念和特征。PWM 可以根据不同的需要配置成不同的操作模式，要掌握不同操作模式的应用。一般情况下应用的四种模式是：输入捕捉、输出比较、边沿排列 PWM 模式和中心沿排列 PWM 模式。关于这四种模式的概念和解释，请参考 S12 芯片手册 “功能描述”和《单片机认识与实践》 “定时器”。

其次需要掌握与 PWM 有关的寄存器。在 S12 芯片中，与 PWM 相关的寄存器和定时器也是相关的，因为 PWM 是单片机定时器的组成部分，所以在此所要理解的寄存器都是以 PWM 开始书写的。下面简单介绍几个需要掌握的寄存器：（1）PWM 启动寄存器（PWME）本寄存器的 8 个 bits 分别用来开关 8 路 PWM 的通道；（2）PWM 极性寄存器（PWMPOL），本寄存器的 8 个 bits 分别用来设定 8 路 PWM 的通道输出波形的起点电平。（3）PWM 时钟选择寄存器（PWMCLK）本寄存器的 8 个 bits 分别用来设定 8 路 PWM 通路的时钟来源；（4）PWM 预分寄存器（PWMPRCLK）本寄存器用来设定 Clock A 和 ClockB 的预分频因子；（5）PWM 波形对齐寄存器（PWMCAE），本寄存器的 8 个 bits 用来分别选择 8 路 PWM 通路输出是中心对称还是左对称。（6）PWM 控制寄存器（PWMCTL）本寄存器提供 PWM 模块操作时的几个控制位。；（7）Clock A 分频寄存器（PWMSCLA），本寄存器用来设定 Clock SA 的频率：Clock SA= Clock A /（2*PWMSCLA）；（8）PWM 通道计数寄存器（PWMCNTx），此 8 个寄存器分别为 8 个通道的波形输出计数器；（9）PWM 通道周期寄存器（PWMPERx），此 8 个寄存器分别为 8 个通道设定方波的周期；（10）PWM 通道脉宽寄存器（PWMDTY x），此 8 个寄存器分别为 8 个通道设定脉宽；（10）PWM 关闭寄存器（PWMSDN）,本寄存器用来设定紧急情况下 PWM 的自关闭功能。要理解和掌握这些寄存器详细的功能和使用方法。

看懂并理解 S12 的原理图，能清楚板上各个模块怎样连接到子板的扩展引脚上去的；重点能清除，蜂鸣器电路是怎样供电的？通过那个口与 S12 芯片相连接的？

最后，要理解和掌握本实验用到的基本硬件电路。

4.9.4 实验指导

S12 应用开发板 PWM 模块硬件原理图：

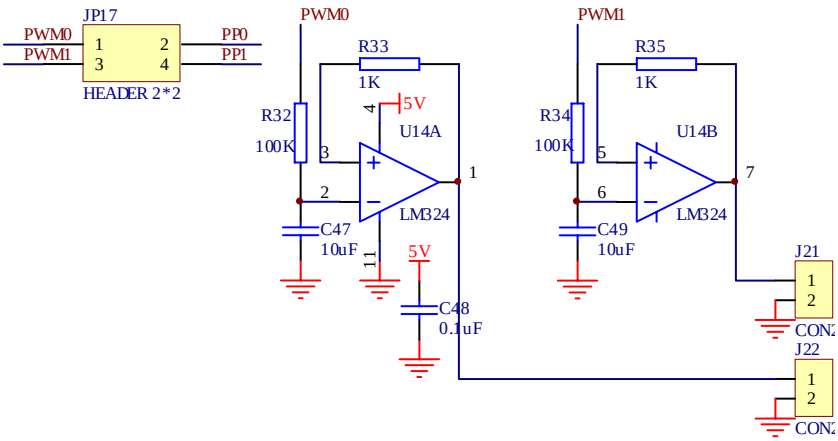

图 4-16　　PWM 硬件原理图

在该实验中，硬件连接非常的简单，将 PP0 与一个 LED 的负极相连即可。我们只需要连接好跳线即可。

使用单片机内部 PWM 模块调制产生不同的脉宽的方波，观察方波经过低通滤波器的 D/A 转换效果。当然，此时要根据公式：占空比=[(PWMDTYx+1) / (PWMPERx+1)] ×100 %，在程序上进行修改 PWMDTYx 与 PWMPERx 的值才能改变灯的亮度)。

PWM 的基本操作模式有四种：输入捕捉、输出比较、边沿排列 PWM 模式和中心沿排列 PWM 模式。在本实验中应用了 PWM 的输出比较功能。在输出比较发生时，程序就会进入中断服务子程序。在程序中可以改变的值，所以可以通过程序来设置占空比的大小，改变占空比也就改变了蜂鸣器高电压的时间，从而改变了蜂鸣器响亮的时间长短。

改变占空比的方式很多，结合该实验给出两个思路。1：改变模数寄存器的值；可以通过修改程序中的值，从而改变占空比的大小，这样也就改变了周期；2：在不改变周期的情况下改变通道数值寄存器的值来改变占空比。改变寄存器的值方式很多，可以在程序中直接修改数值，也可在程序中设置循环等方式逐渐增加或减小寄存器的值，达到调节占空比的目的。

PWM 可产生计时脉冲，这些计时脉冲是位置、极性、持续时间和频率可编程的。当计数器达到一个输出比较通道的通道数值寄存器内的值时，PWM 能设置、清除或连接通道管脚。在输出比较模式下，只有在一个 16 位寄存器的两个 8 位字节都被写入后，数值才会被转移到相应的定时器通道数值寄存器。这一关联顺序操作可通过写通道状态/控制寄存器进行手动复位。 一个输出比较事件可设置一个标志位，该标志位可随意产生一个 CPU 中断请求。具体内容请参考 S12 芯片手册第十章第五小节。

4.9.5　实验思考

1．如何正确配置 TPM 寄存器？

2．怎样通过调整占空比的大小改变蜂鸣器响的时间？

3．如果改变周期，不变占空比，实验现象有什么改变？

4.10　IIC 总线实验

4.10.1 实验目的

1.了解 IIC 总线中的基本概念。

2.掌握 IIC 总线的通信协议。

3.了解 PCF8563 芯片的各个寄存器设置。

4.10.2 实验要求

1.在了解 IIC 各寄存器配置的基础上，了解 PCF8563 秒，分，时，天，月，年，控制/状态寄存器等各寄存器如何配置，通过 IIC 总线通信协议，实现读取日历钟芯片时间日期并通过串口将其显示在超级终端上。例如以“#“开头，每两位为一组，分别表示秒，分，小时，天，星期，月，年，的方式显示。

2.比较 SPI 接口和 IIC 总线的功能特点。

3.采用中断方式从串口接受字符串的过程中，在一个完整的帧（字符串）接受完之前，要注意把总中断关掉。

4.10.3 实验准备

在进行本实验之前，需要知道 IIC 总线的信号线的组成及其典型接法，一般具有 IIC 总线的器件其 SDA 和 SCL 管脚都是漏极开路（或集电极开路）输出结构，因此实际使用时，SDA 和 SCL 信号线都必须要加上拉电阻，且上拉电阻一般取值 3~10k 欧，并了解开漏结构的好处。

进行实验之前，还需要知道 IIC 总线的几个基本概念： ①发送器②接收器③主机④从机，此外，还需要了解 IIC 总线数据传送速率以及 IIC 总线上数据的有效性。

通常，一个标准的 IIC 通信由四部分组成：START 信号；从地址传输；数据传输；STOP 信号。在实验之前，需要知道什么时候产生 START 信号，什么时候开始从地址传输，数据传输又是怎么开始的，怎样进行传输的，STOP 信号什么时候产生。其中，还需要知道一个重要的知识：应答位。在不同的传输模式下，应答位的产生是不同的。

最后需要了解的就是 IIC 寄存器配置，还需要会利用公式计算 IIC 总线频率。

4.10.4 实验指导

IIC 总线硬件原理图如下图 4-17：

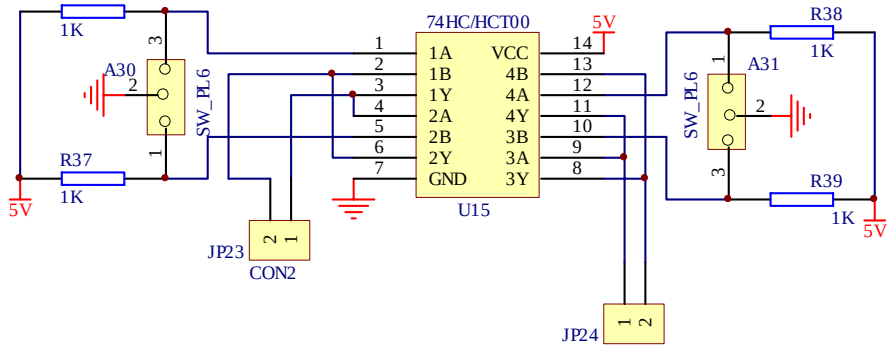

图 4-17　　PCF8563 原理图

IIC 总线通信协议描述：

起始条件：当 SCL 处于高电平期间时，SDA 从高电平向低电平跳变时产生起始条件。总线在起始条件产生后便处于忙的状态。起始条件常常简记为 S。

停止条件：当 SCL 处于高电平期间时，SDA 从低电平向高电平跳变时产生停止条件。总线在停止条件后处于空闲状态。停止条件简记为 P。

从地址传输：在起始条件之后马上开始的第一个传输字节使由主设备传输的从机地址。该地址最后一位告诉从设备数据传输的期望方向。

1=读传输，从发送数据给主

0=写传输，主发送数据给从。

仅仅匹配主设备传输的调用地址的从设备通过送回一个应答位相应。这通过在第九个时钟拉低 SDA 完成。

数据传输：数据只有在 SCL 为低时可以改变，在 SCL 为高时保持不变。SCL 对于每一个数据位有一个时钟脉冲，MSB 首先传输，每一个数据字节紧接在在一个第九应答位之后，应答位由接收器件告知。即一个完整的传输字节需要 9 个时钟周期。

应答位的产生：在 IIC 总线传输数据过程中，每传输一个字节，都要跟一个应答状态位。接收器接收数据的情况可以通过应答位来告知发送器。应答位的时钟脉冲仍由主机产生，而应答位的数据状态则遵循“谁接收谁产生”的原则，即总是由接收器产生应答位。主机向从机发送数据时，应答位由从机产生；主机从从机接收数据时，应答位由主机产生。IIC 总线标准规定：应答位为 0 表示接收器应答(ACK)，常常简记为 A;为 1 则表示非应答（NACK），常常简记为~A。发送器发送完 LSB 之后，应当释放 SDA 线（拉高 SDA，输出晶体管截止），以等待接收器产生应答位。如果接收器在接收完最后一个字节的数据，或者不能再接收更多的数据时，应当产生非应答来通知发送器。发送器如果发现接收器产生了非应答状态，则应当终止发送。

由以上分析：在我们的参考程序中对 IIC 模块设计了 5 个基本功能子程序分别为：IIC 模块初始化；MCU 从从机读 1 个字节；MCU 向从机写 1 个字节；MCU 从从机读 N 个字节；MCU 向从机写 N 个字节。

4.10.5 实验思考

思考如何使程序实现报警功能以及闹钟功能。

比较 SPI 接口和 IIC 总线的功能特点。

4.11 LIN 总线通信实验

4.11.1 实验目的

4.11.2 实验要求

4.11.3 实验准备

4.11.4 实验指导

LIN 总线硬件原理图：

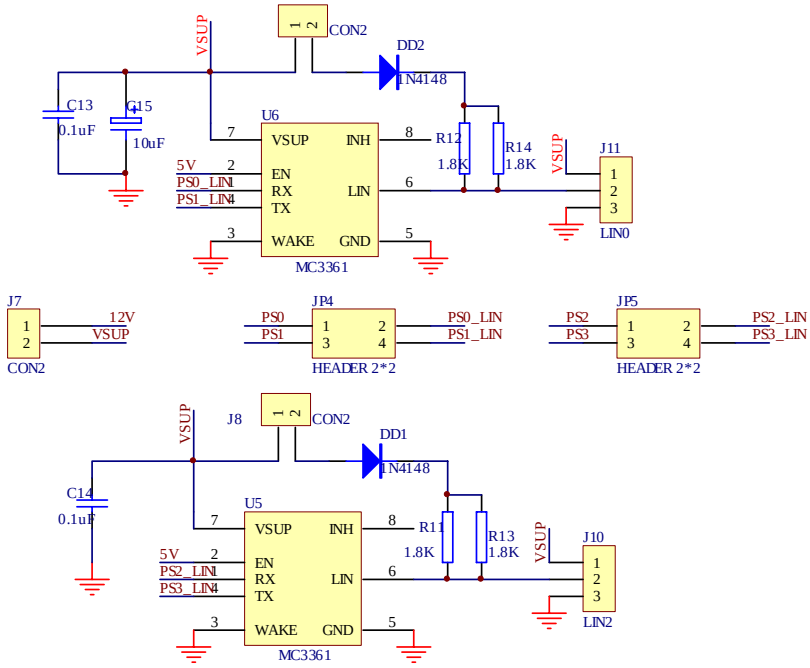

图 4-18　　LIN 总线硬件原理图

4.11.5 实验思考

4.12 MSCAN 模块通信实验

4.12.1 实验目的

初步了解 CAN 总线，熟悉其通信协议，并掌握 S12 单片机 MSCAN 模块的使用方法

4.12.2 实验要求

利用 MSCAN 模块的 Loop Back 模式，自发自收数据。

4.12.3 实验准备

1．报文滤波

CAN 总线允许用户设定一组标识符放在报文中，接收节点可以根据报文的标识符来决定是否接收该报文，我们称之为“报文滤波”。

CAN 总线中传送信息是以打包方式传送的，单元叫做报文或帧，一共有4 中不同的帧，格式：数据帧、远程帧、出错帧、超载帧。

本实验中用户需要了解数据帧的帧结构：帧起始、仲裁场、控制场、数据场、CRC场、ACK场、帧结束。其中仲裁场包含一段标识符，在最新的CAN2.0B 的协议中根据标识符的长度将数据帧分成标准帧和扩展帧，标准帧中的标识符长度为11 位，而扩展帧的标识符长度达到29 位。编程时，用户可以设定选用标准帧或是扩展帧，设定的寄存器在发送缓冲区中（IDR1）。接收节点需要预先设定可以接收的标识符（CANIDAR0~7），根据标识符长度不同可以设定为2 个32 位的标识符、4 个16 位的标识符、8 个8 位的标识符或是直接将该滤波器关闭。关闭后，节点就不再接收任何帧了。用户也可以选择忽略标识符滤波，通过设定标识符屏蔽寄存器(CANIDMR0~7)，可以按位选择哪一位可以忽略。

2．发送缓冲区和接收缓冲区

在发送缓冲区的设计上，MSCAN 内置了三个具有局部优先级的发送器，某一时刻只有一个缓冲区处于前台状态，当一次发送结束之后，模块会通过一个特定的算法（见教材285页）来选定下面将哪一个缓冲区推入前台。这样可以允许MSCAN 总是将优先级最高的数据帧先发送出去。

3． 自发自收模式

为了验证发送器和接收器功能正常，MSCAN 设置了一个自发自收模式（LOOP BACK），在这种模式下，发送管脚和接收管脚自动在芯片内部短接，发送的内容直接送到接收器，实验中正是使用了这种模式来验证发送和接收程序的正确。需要说明，这种模式除了没有物理层驱动电路外，与实际应用情况完全一样。

4.12.4 实验指导

在开始接收和发送之前，首先要对MSCAN 模块初始化，MSCAN 模式中提供了一个初始化模式，有些寄存器只能在初始化模式下才能写。

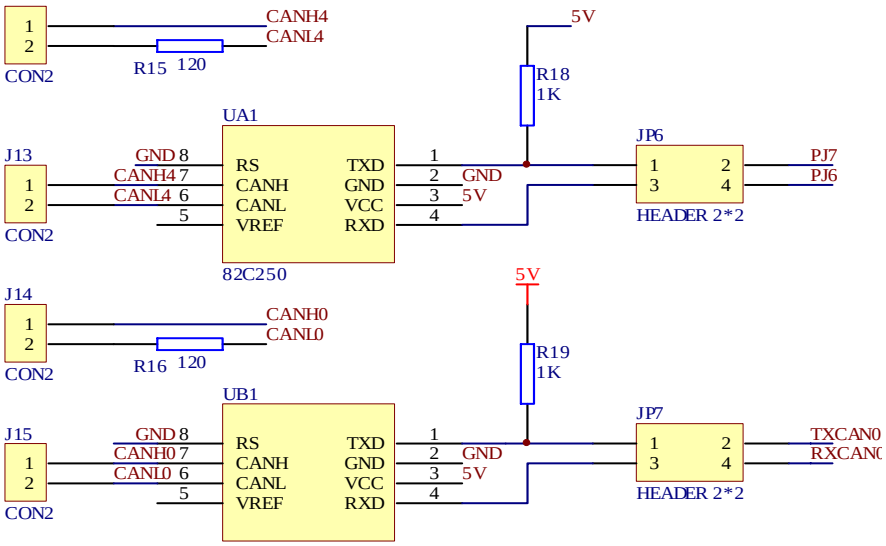

图 4-19　MSCAN 总线模块

4.12.5 实验思考

1. 同为总线型接口，CAN、SPI 和IIC 都是如何辨别属于自己节点的数据的？
2. CAN 总线共有哪几种帧格式，它们各自的作用是什么？
3. 为什么 MSCAN 的发送器采用三个缓冲区的设计？

第5章　多任务程序实验

5.1　UC/OS—II 的移植实验

5.1.1　实验目的

1.通过本次实验能够加深理解 UC/OS-II 的移植过程，和 UC/OS-II 的实时内核。

2.为以后的掌握深层次模块的移植奠定基础。

3.进一步掌握 UC/OS 移植的细节。

5.1.2　实验要求

将 UC/OS-II 移植到 S12 单片机中，实现简单的任务调度（通过串口在超级终端上显示不同的字符）。

5.1.3　实验准备

1. 硬件：S12 子母板，串口线，一套 BDM 调试工具，USB 线，5V 电源

2. 所谓移植，就是使一个实时内核能在其他的微处理器或微控制器上运行。要实现 UC/OS-II 的移植，需要做两方面的工作：一是重新定义内核的大小和功能；二是为内核编写与硬件相关的代码。具体参考《嵌入式实时操作系统 UC/OS-II》第十三章移植 UC/OS-II.

明确 UC/OS-II 的代码结构和 UC/OS-II 移植的一般方法。

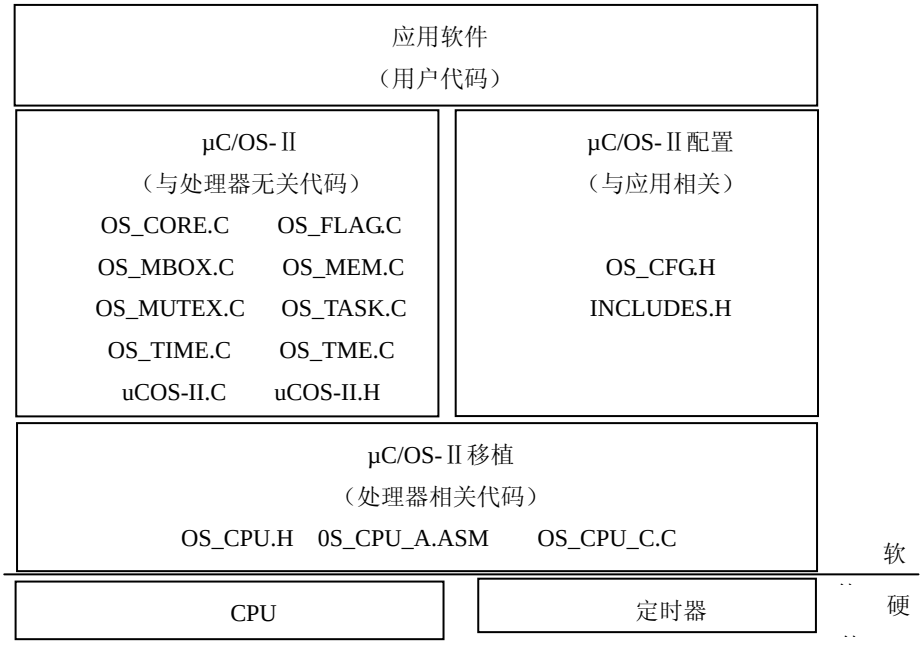

图 5-1

如图 5-1 所示，如果要实现 μC/OS-II 的移植，必须为其编写 OS_CPU_A．ASM ，OS_CPU_C．C，OS_CPU．H 3 个文件。其中前两个文件有 OSStartHighRdy(), OSCtxSw(),OSIntCtxSw(), OSTickISR()，OSTaskStkInit（）五个函数；OS_CPU.H 文件用于定义特定 CPU 的数据类型，定义相关的宏。这 3 个文件是与 CPU 特性有关的 。

而与 CPU 类型无关的 3 个文件 OS_CORE.C、OS_TASK.C、OS_TIME.C 是一定要有的。其它几个文件可以不包含进去。

用户根据自己的应用系统来定制合适的内核服务功能.包括两个文件:OS_CFG.H, INCLUDES.H. OS_CFG.H 是来配置内核, 用户根据需要对内核进行定制, 留下需要的部分, 去掉不需要的部分, 设置系统的基本情况. 比如系统可提供的最大任务数量, 是否定制邮箱

服务，是否需要系统提供任务挂起功能，是否提供任务优先级动态改变功能等等；头文件 INCLUDES.H 为整个实时系统程序所需要的文件,包括了内核和用户的头文件.

然后编写自己的任务代码，并启动 UC/OS-II。

5.1.4 实验步骤

1. 编写 INCLUDDES.H 头文件，可根据实际应用修改该文件。
2. 用 C 语言或汇编语言为内核编写与硬件有关的函数。
3. 定制合适的内核服务功能，编写测试程序。
4. 验证移植的 UC/OS-II 是否正常工作。

5.1.5 实验思考

S12 处理器与其它处理器相关技术细节的区别，如何在其它处理器上实现 UC/OS-II 的移植。

5.2 键盘模块实验

5.2.1 实验目的

1.掌握在有操作系统的 MCU 中矩阵键盘模块的 C 语言编程方法。

2.学习键盘在多任务下的工作原理。

3.掌握键盘是如何通过中断来实现扫描以及键盘如何去抖问题。

4.掌握任务间通信机制——信号量。

5.2.2 实验要求

1.对比独立式键盘的基本程序，学习键盘在多任务下是如何工作以及独立式与矩阵键盘的区别，比较它们的工作特点和是否工作在多任务下的区别。

2.充分理解采用信号量机制的任务间通信，分析此种机制的优点与缺点。

在本实验的参考代码主函数中，我们设计了 4 个任务按优先级由高到低分别是启动任务，任务 1 和任务 2，以及按键任务。启动任务，任务 1 和任务 2 主要是通过串口在超级终端上显示信息，而键盘则是通过 lED 灯把按键值显示出来。即在矩阵键盘上按下相应的键值对应 LED 上相应灯亮，如按下 B，会将对应的二进制编码的反码显示在指示灯 D40~D43 上,即 0100 通过这 4 个任务来完成对键盘的测试。

对于串口资源的使用，任务之间采用信号量机制来实现资源共享。而任务调度我们可采用系统里提供的 OSTimeDly()实现，任务调用 OSTimeDly()后，此时就绪态中的高优先级任务执行，一旦规定的时间期满，原来的就会立即进入就绪状态。当该任务在所有就绪任务中具有最高的优先级时，就会立即运行。

5.2.3 实验准备

在基本实验中我们已经知道了键盘的原理以及识别键盘时存在的问题。在本次实验中大家应首先回顾一下矩阵式键盘结构，同时要熟悉 OS 下键盘模块的相关代码。可参考《嵌入式系统构件》第 3 章的键盘章节。这个大概需要 45-60 分钟。

在 OS 下编写程序，对任务的管理主要是通过 OSStartHighRdy(), OSCtxSw(),OSIntCtxSw(), OSTickISR()，OSTaskStkInit（）这五个函数实现的。大家首先要理解这些函数的工作原理。以及任务间的通信机制如信号量、邮箱、队列等概念和原理。

本实验在任务间通信时主要采用的是信号量，此内容可参考《嵌入式实时操作系统 UC/OS-II》的第七章信号量管理，所以大家要着重理解一下本内容。

5.2.4 实验指导

本次实验使用的矩阵式键盘的电路图如下图 5-2 所示：

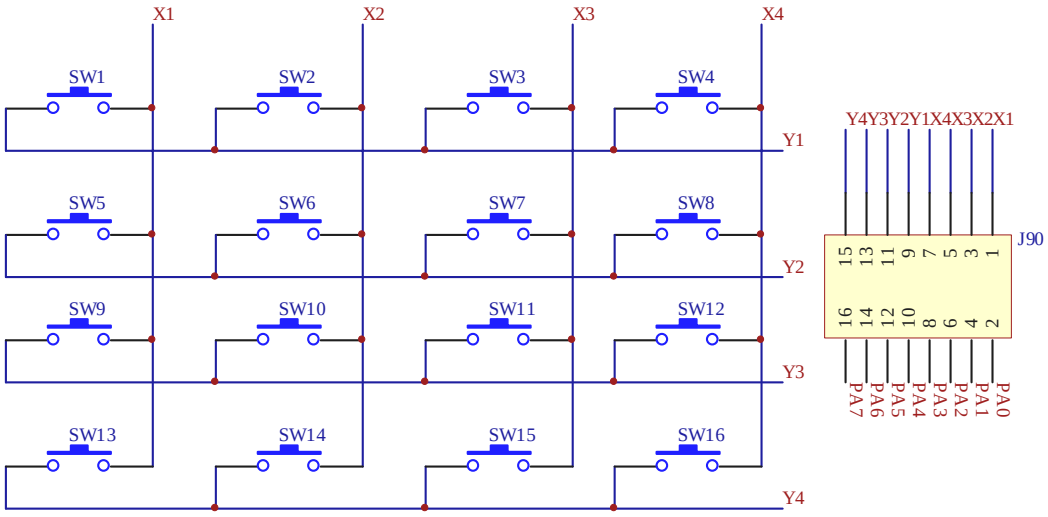

图 5-2

采用 4*4 键盘，因此将键盘的行、列共 8 条线连接到 PTA 口上，其中行 1－4 连接到 PTA0－3 上，列 1－4 连接到 PTA4－7 上。即“行”对应 S12 端口 A 的高四位。“列”对应端口 A 的低四位。按下相应的键会将按键的相应值输出到 LED 指示灯显示。如按下键 A，将会把十六进制 0xA 的二进制编码的反码显示在指示灯上,即 0101，因为 LED 指示灯为低电平时显示亮的状态，故将 0xA 的值取反会更直观。

键盘需要解决的核心问题包括扫描和去抖。键盘扫描有行扫描法和反转法两种。这两种方法实现过程，已在第四章的键盘实验中详细介绍，在此不累述。下面主要分析一下去抖机制（状态机原理）。

键盘扫描任务还要解决按键抖动的问题。为了方便理解，先在一个完整的按键过程中定义几个状态。第 N 次采样时无键按下，此时键处于初态；N+1 次采样时第一次采到有键按下，键进入次态，此时按键尚不稳定；N+2 采样时为连续第二次采到有键按下，键进入初稳态，次态之后初稳态之前的这段 20ms 的时间称为去抖周期； N+3 次采样时为连续第三次采到，键进入持续稳定态，第 N+I+1 次采样时为抬键后第一次被采到，此时键又回到了初态。因此把键态分为四种情况：初态，次态，初稳，稳定态；

为了使程序根据键态进行相应的处理，键盘扫描程序应为每一个按键设置状态单元，根据每个扫描周期采样到的键的即时状态是开或是关，再结合原来的键态得到现在的键态。可以建立一个状态机来扫描键盘。如图 5-3 所示状态机在每个扫描周期执行一次。

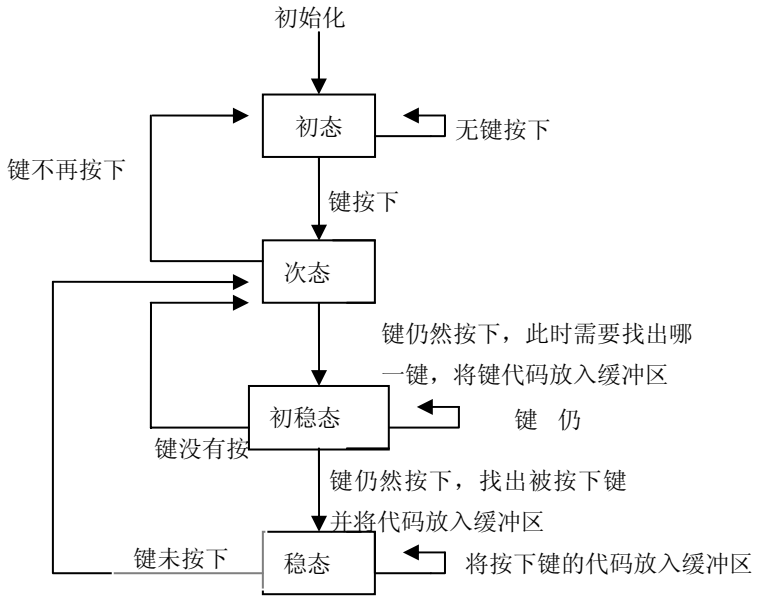

图 5-3

有上述分析得：我们可用一个单独的任务 KeyScanTask()来扫描键盘。当应用程序调用了 KeyInit()时，KeyScanTask()将被创建。一旦被创建后，KeyScanTask()将每 KEY_SCAN_TASK_DLY 毫秒执行一次。KEY_SCAN_TASK_DLY 应该被设置为产一个范围在的 10~30Hz 的扫描频率之间(即 33~100ms 之间）。在键盘扫描模块中，KeyScanTask ()函数发挥着相当重要的作用，它负责扫描任务，通过 keyInitport()函数、读取列状（KeyGetCol）、写行状态（KeySelRow）函数与硬件打交道来扫描相应的行和列。

键盘扫描任务并不解决所有的问题，它所做的工作包括扫描键盘、去除抖动、按盘编码等，当然最终要把按键编码送入缓冲区 keyBuf[]，应用程序可以透过接口函数从该缓冲区得到键编码值（可采用信号量机制来标志是否有按键发生），维护这个缓冲区还需要两个指针：读指针和写指针，以及一个键计数器。

通过上图和下图我们可以知道：

刚开始的时候，该状态机处于 KEY_STATE_UP()状态。当有键被按下时，则该状态机的状态改变为 KEY_STATE_DEBOUNCE()次态，随后它将执行 KEY_SCN_TASK 毫秒。请注意 μC/OS-II 中的函数 OSTimeDlyHSM()提供一个便利的方法来除去回弹且在规定的时间间隔内扫描键盘。

在该延迟后，KeyScanTask()在 KEY_STATE_DEBOUNCE 状态下执行代码，再次检查看是否有键被按下。如果键被释放，状态机将返回到 KEY_STATE_UP 状态。如果该键仍然被按下，则通过调用 KeyDecode()可以找到该扫描代码，并且通过 KeyBufIn()把它插入到这个循环的缓冲区中。如果缓冲区已经满了，KeyBufIn()将丢弃扫描码。KeyBufIn()也给键盘发信号量，允许应用程序通过 KeyGetKey()来获得该键的扫描码。然后这个状态机改变为 KEY_STATE_RPT_DLY 状态。

如果该键按下的时间多于 KEY_RPT_START_DLY 扫描时间，则自动重复功能将会启动。在这种情况下，该扫描码将插入被到缓冲区中，且状态改变为 KEY_STATE_RPT_DLY 状态。如果改键不再被按下，则该状态机的状态改为 KEY_STATE_DEBOUNCE 状态，用来去除被释放键的回弹。

在一个扫描周期后，KeyScanTask()在 Key_State_Rpt_Dly 状态下执行代码，其中被按下键的扫描码将每隔 KEY_RPT_DLY 扫描时间被插入到缓冲区中。像其他状态一样，如果键被释放，就需要去除回弹。

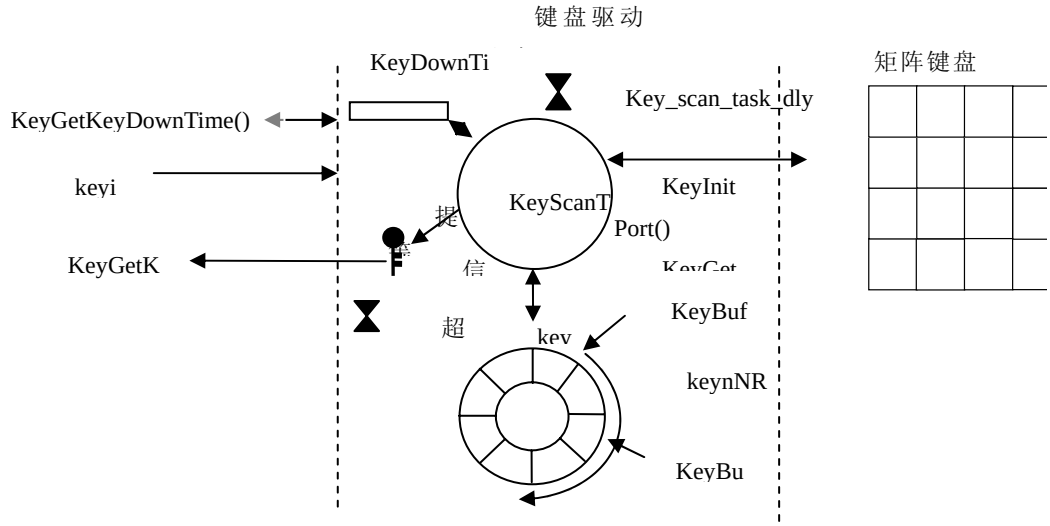

图 5-4

图 5-4 展示了 μC/OS-II 平台上工作的矩阵键盘模块的流程图。

5.2.5 实验思考

想一下，是否可采用别的通信机制，如何设计别的任务来测试？键盘 RTOS 下键盘模块与其它模块结合如何使用？

5.3 多路复用 LED 数码管显示实验

5.3.1 实验目的

1. 通过实验理解 LED 模块在 μC/OS-II 下的工作原理。
2. 掌握 LED 模块的多路复用技术的段码和位码是如何工作的。
3. 理解任务间通信时如何采用邮箱机制。
4. 掌握在有操作系统的 MCU 中 LCD 模块的 C 编程方法。

5.3.2 实验要求

1. 段码和位码都是由输出端口控制的，掌握它们是如何通过 MCU 来控制开关的情况。
2. 本次实验采用的 LED 模块采用多路复用技术，要知道什么是多路复用技术？
3. 编写一个程序在 LED 上显示想要的时钟日历。
4. 研究参考程序中主函数，看有几个任务，理解几个任务之间是如何调度的。

5.3.3 实验准备

实验之前需要查阅相关资料掌握一些有关 LED 数码管的基本知识，①首先需要熟悉一下 LED 模块的硬件结构，LED 简称发光二极管，是可发光的半导体设备，一般门槛电压在 1.6V~2.0V 之间， LED 数码管分共阳极与共阴极两种，其工作特点是，共阳极接高电平，使用低电平点亮数码管；共阴极接低电平，使用高电平点亮数码管。针对具体硬件看看共阴极与共阳极有什么区别。②熟悉七段数码显示器多路复用技术的特点，看看七段与通常所说的八段有什么不一样？③数码管主要是由段码与位码控制的，在电路图中什么是段码，什么是位码？同时请思考段码与位码是如何工作的？是如何控制 LED 的？

5.3.4 实验指导

（1）七段 LED 显示器由 7 条发光二极管组成，并按“日”字型排列。显示段可分别记为 a、b、c、d、e、f、g，有的还带有一个小数点 dp，可以用来显示数字和部分字符。一个 LED 硬件的具体电路图如下所示，

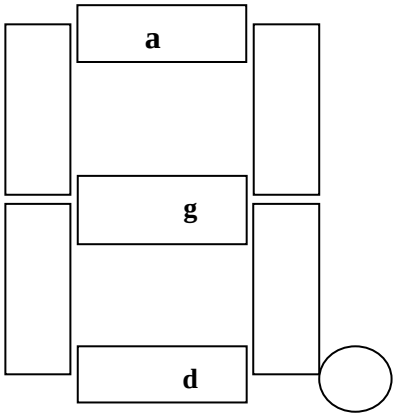

图 5-5

（2）7 段数码管可分为共阳极和共阴极两种，如下图所示。共阴极 7 段数码管的信号端高电平有效，只要在各个位段上加上相应的信号即可使相应的位段发光，共阳极的 7 段数码管则相反，在相应的位段加上低电平即可使该位段发光。

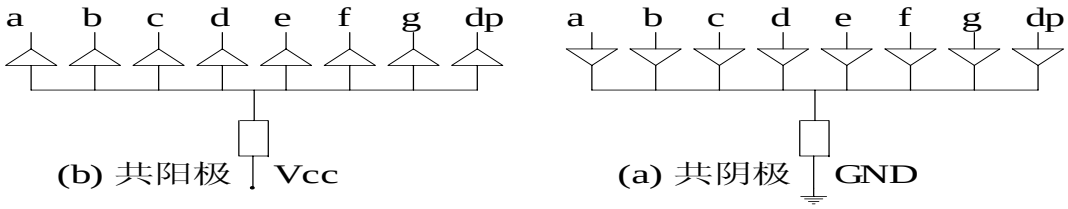

图 5-6

（3）7 段数码显示器多路复用技术的工作方式就是一个一个地轮流点亮各个七段数码显示器，通过恰当地选择点亮 7 段 LED 的时间间隔(1~5ms)，给人一种视觉暂留的效应，感觉每个 LED 都在同时显示。除了要给七段 LED 显示器提供段选码之外，还要提供一个位选码，位选码就是控制每个 7 段 LED 显示位轮流接地点亮的代码。更多相关具体信息请参照其它资料，下图是本次实验用到的数码管模块的电路图：

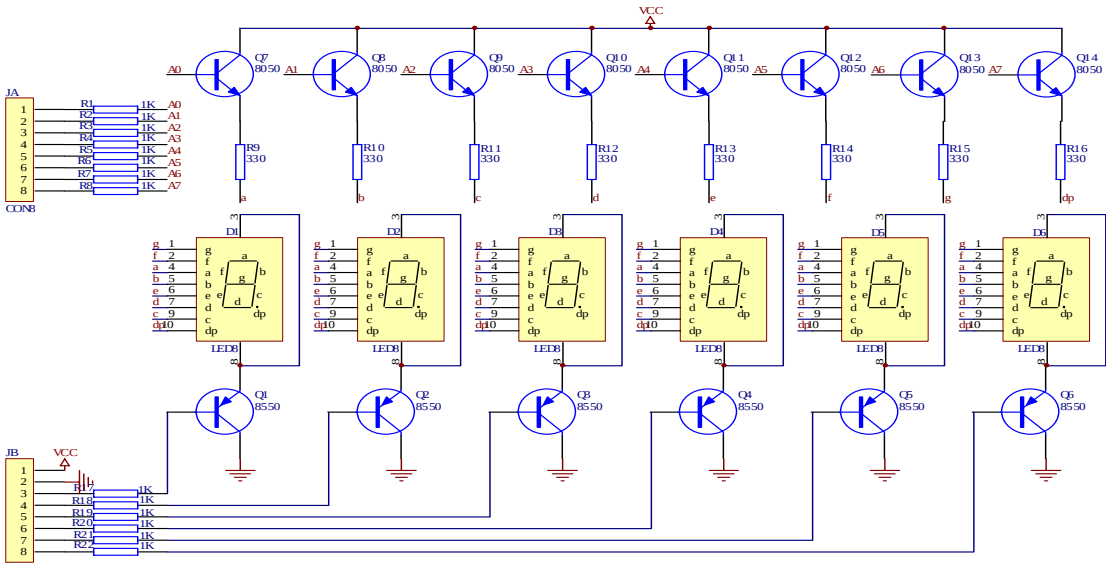

图 5-7

如图 5-7 所示，上面的那个输出端口是用来控制 LED 的“段”，即每一段是否点亮的情况。下面的输出端口是控制“位”的，即每一个 LED 的电源接通情况。MCU 通过使用一个输出端口，可以控制一个或多个 LED，通过在该端口的合适比特位上写一个 0 或 1 电位用

来开启 LED（本实验使用的是共阳极数码管，所以输送的是 1 电位）。如果需要额外的 LED，可以增加一个或更多的 8 位端口。这个额外的端口可以用来控制更多的数字或段。增加一个数字端口将增加 MPU 的系统开销，但是不会增加系统的电流消耗。如果愿意减少系统开销，可以增加段的端口。但要增加电流开销。

本实验提供的代码允许多路复用的 LED 多达 64 个（需要使用两个 8 位的输出端口），如果需要增加更多的七段数字或者离散的 LED，可以很容易的对该模块进行修改。

在软件中设计实现多路技术如图 5-8 所示。假设数字少于 8 个，同时需要硬件计时器，它能够以一定的速率产生中断：

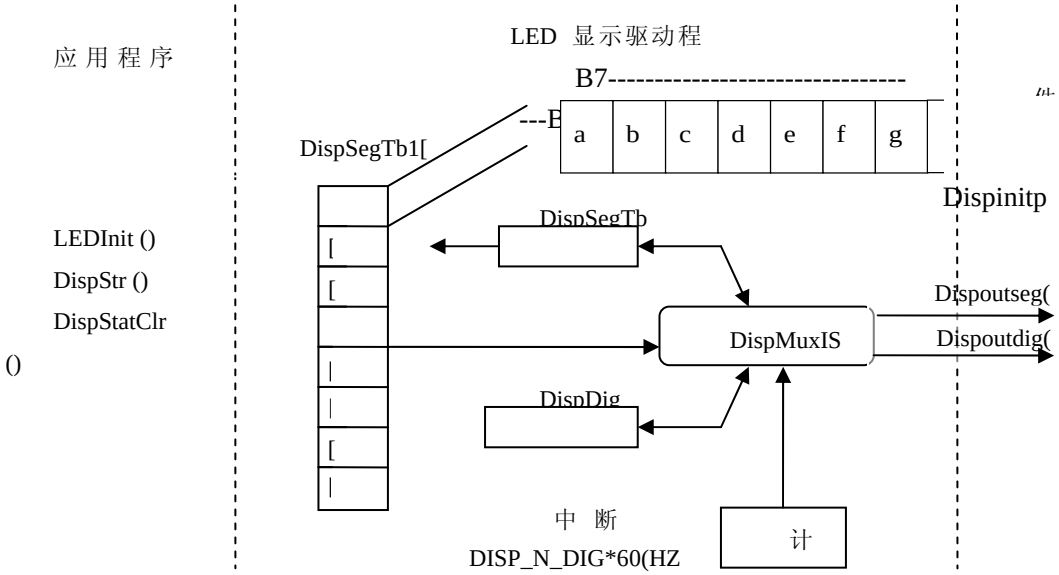

图 5-8

表 DispSegTbl[ ]包含了相对应数字的段字型数据，DispSegTbl[ ]中的第一个元素包括最左边数字的段字型。DispSegTblIx 是对应于段表的索引，该表将指向下一个要显示的数字，DispDigMsk 是一个掩码，用来选择下一个要显示的数字，相当于位选码，在任意时刻，仅有一个数字可以被显示。在选择下一个数字之前，DispMuxISR()将关闭当前数字的段，这个步骤用来防止显示闪烁，如果显示段在下一个数字被选择之前没有被关闭，以前的段将出现在新的被选的位上。DispMuxISR()只关心以一定的刷新速率更新显示。段显示数据如何写入 DispSegTbl[ ]由是任务级代码负责。

LEDInit ()初始化模块的内部变量，且初始化硬件端口。在这个函数中，建立一个消息邮箱,用于接收更新 LED 显示的数据。

DispClrScr()用来清除显示器。

DispStr ()用来显示一个 ASCII 字符串。

DispStatClr ()用来开启一个单独的 LED。

DispStatSet ()用来关闭一个单独的 LED。

最后我们还要配置一下系统文件：

首先在 OS_CPU_C.C 文件中添加需要调用 CLK 和 LED 中函数的声明。

```
extern void      ClkSignalClk(void);
```

此函数用于通知 ClkTask 任务更新秒计数器,此处为配合 LED 使用,提高 LED 刷新频率,时钟节拍需要增加为原始周期的十倍，此时系统时钟的周期为 2ms(1S=1000ms,1ms=1000us)。

```
extern void    DispMuxHandler (void);
```

此函数用于按将高的频率刷新 LED 显示数据,实际上同一时刻只有一个 LED 显示数据。

5.3.5 实验思考

可以增加更多的 LED，修改其中的部分代码以及相关参数能实现，试着修改。

5.4 字符 LCD 显示实验

5.4.1 实验目的

1.理解在 μC/OS-II 下 LCD 模块的工作特点。

2.掌握在有操作系统的 MCU 中 LCD 模块的 C 编程方法。

3. 理解任务间通信时如何采用事件标志组机制。

5.4.2 实验要求

1.在 μC/OS-II 下编写一个编写 3 个任务分别是启动任务；用于显示测试进度条的任务，进度条二比进度条一快；键盘任务。使按下不同的键相应的键值反应到 LED 上，或蜂鸣器会发出不同频率的声音，或者使 LCD 显示板实现清屏，光标移动，从制定位置开始显示字符串信息。

2.深入理解在本实验中 LCD 模块是如何把上层软件和底层连接起来的。

5.4.3 实验准备

该实验使用日立公司的 HD44780 芯片作为 LCD 模块的控制器，HD44780 的相关知识在前面的基本实验中有所提到，这里不再叙述。在实验之前重复复习 μC/OS-II 是最基本的要求，同时请认真的阅读 HD44780 的芯片手册，掌握该芯片有什么基本功能，参照第四章掌握它的工作原理，并把相关有用的资料记录下来，在下面的实验中把该芯片所有的功能都通过程序体现出来。

5.4.4 实验指导

在第四章中我们已经详细接受了 LCD 模块的工作原理，那么在这个实验中主要说明一下在操作系统下应该如何编程。

为了提供程序的可移植性，可重复性，一般在编写程序时采用分层的思想，即与纯硬件相关的封闭到底层函数接口中，则上层编程时就不需要考虑硬件接口问题，直接利用底层接口函数编写一下要实现的功能函数。

基于上述思想，LCD 的模块图如下图 5-9：

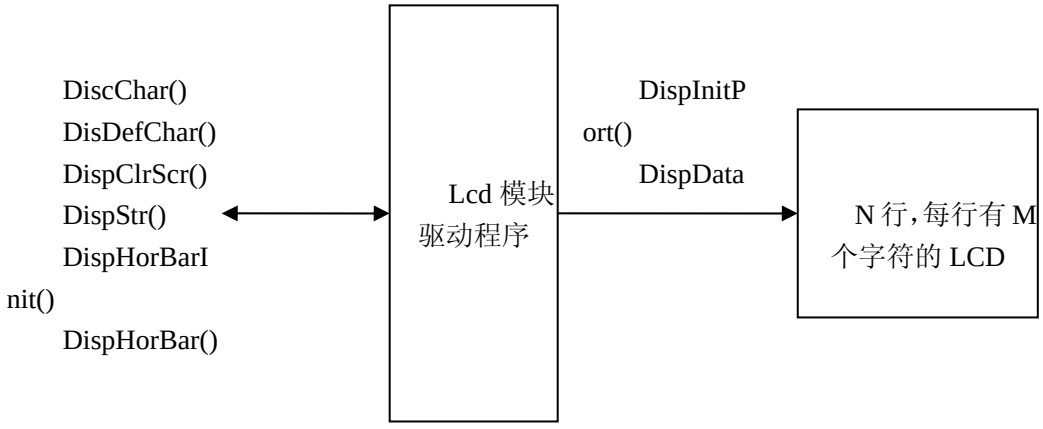

图 5-9

DiscChar 允许在显示器的任意位置显示单个字符。

DisDefChar()允许定义 8 个字符或符号，共 5*8 个像素。

DispHorBar()用于在屏幕的任意地方显示垂直的条形图。

DispHorBarInit()定义了 5 个带有标识符的特殊符号。在使用之前，仅需调用，除非是由于其他目的而要重新动态地定义符号标识符。

DispStr()允许在显示器任意位置显示字符串。

5.4.5 实验思考

该实验的显示模块利用信号量防止多个任务同时访问显示器。把该 LCD 模块的实验程

序与多任务下程序进行比较学习。以及多思考多如何让 LCD 模块在多任务中工作。

5.5 钟点日历时钟实验

5.5.1 实验目的

1.了解嵌入式系统中，时钟/日历采用的两种实现方法各自的优缺点。

2.熟练掌握如何用软件来实现时钟/日历功能。

3.进一步熟悉 UCOS/II 下多任务的创建以及多任务之间的通信机制

5.5.2 实验要求

1.利用 CLK 模块实现日历和时钟功能。

可以通过不同的接口函数获得不同形式的日期和时间格式,并可根据用户设置的参数设置日期和时间。

2.在与键盘模块结合时,通过按键 0~A,可以获得不同形式的日期和时间,具体格式参见代码 main.c 中的按键任务。其中按键 8 的相应功能被屏蔽,因为在从按键任务切换到串口发送任务时不能正常切换,需要进一步处理。

5.5.3 实验准备

在很多基于 MCU 的嵌入式系统中，时间的管理是很重要的，时钟/日历是一个很有用的模块。在嵌入式系统中，时钟/日历可以采用两种方法实现。

（1）硬件实现。就是使用专门的时钟/日历芯片来实现，时钟/日历芯片一般能直接与 MCU 相接。有些芯片提供一个内置的日历，有些芯片内嵌电池，在断电后也能走时。

（2）软件实现。利用软件实现的时钟/日历模块能实现硬件方法的大部分功能，而且软件方法需要的 ROM、RAM、CPU 时间很少，不增加系统成本，也可以很容易的扩展功能，例如闹钟等，但维护时钟/日历运行时比较繁琐。

当应用负担不了与时钟/日历芯片、电池和一个晶体振荡器有关的额外的花费时，利用软件对时钟/日历进行维护则是最好的解决方案。

本实验的准备工作具体可参看《嵌入式构件》。

5.5.4 实验指导

对微处理器来说，维护时钟/日历是一件繁琐的事情。首先需要的是一个周期时间源，他将按规律性间隔中断微处理器，这样的时间源是很容易找到的。AC 电源线频率（50 或 60hz）通常来说在很长的时间内都是精确的。对于短期内的精确性，用于微处理器的晶体振荡器也是一个好的选择。然而对一个应用程序来说，晶体频率必须分为几个小的。若应用软件运行在实时多任务的操作系统下，操作系统的时钟脉冲是一个放百年的周期时间源。

如果假定微处理器每 0.1 秒被中断，则软件仅需要维护整数计数器，用于记录十分之一秒。每一次中断就增加十分之一秒。若秒计数器从 59 溢出成为 0，则第二个计数器增加 1；若秒计数器从 59 溢出成为 0，则分钟计数器增加 1；等等。每 24 小时日计数器增加 1，月

计数器的溢出取决于当前月，并且如果某月为 2 月，则取决于该年是否为闰年。

5.5.5 实验内容

时钟/日历模块包含一个每秒执行一次的任务，该任务负责修改 8 个变量，应用程序可以透过接口函数访问这些变量。这些变量包括：CLKSec、CLKMin、CLKHr、CLKDay、CLKDow、CLKMonth、CLKYear。

时钟/日历中的日期和时间计数器由任务 CLKTask()进行修改，每秒钟执行一次，日期和时间计数器可看做共享的资源，从而必须获得一个互斥的信号量以访问这些计数器。

CLKTask()调用 ClKUpdateTime()更新时（CLKHr），分（CLKMin），秒（CLKSec）计数器。当时钟从 23：59：59 滚动 00：00：00 显示新的一天时，CLKUpdateTime（）返回 true。

CLKUpdateDate()更新年（CLKYear），月（CLKMonth），日（CLKDay）和星期（CLKDow）计数器。修改日期稍微有点复杂，因为需要记录当前月的天数。当前星期几可通过调用 CLKUpdateDOW（）得到，星期几是 0~6 之间的一个数，0 代表星期日。表 CLKMonthTbl[] 的使用大大简化了日和星期计数器的修改工作。

时钟/日历模块接口函数描述：

◆ ClkFormatDate (INT8U n, char *s)

将当前日期转换为 ASCII 字符串的格式以输出。

◆ ClkInit (void)

用于时钟/日历模块的初始化。

◆ ClkSetDate (INT8U month, INT8U day, INT16U year)

用于设置时钟/日历中的日期。

◆ ClkSetDateTime (INT8U month, INT8U day, INT16U year, INT8U hr, INT8U min, INT8U sec)

用于将时钟/日历设置为需要的日期和时间。

◆ ClkSetTime (INT8U hr, INT8U min, INT8U sec)用于设置时钟/日历的时间部分。

5.5.6 实验结果

（1）结果

1 键被按下：压缩格式的日期 MM-DD-YY，并显示在超级终端上。

2 键被按下：完整的日期包括:星期,月份,日,年，并显示在超级终端上。

3 键被按下：压缩格式的日期 YYYY-MM-DD，并显示在超级终端上。

4 键被按下：压缩格式的日期 HH:MM:SS，并显示在超级终端上。

5 键被按下：压缩格式的日期 HH:MM:SS AM/PM，并显示在超级终端上。

6 键被按下：压缩格式的日期 MM-DD-YY HH:MM:SS，并显示在超级终端上。

7 键被按下：压缩格式的日期 YYYY-MM-DD HH:MM:SS，并显示在超级终端上。

8 键被按下：此处程序有问题，不能正常切换到串口发送任务，故该判断语句被注释掉。

9 键被按下：设置日期为 2008-4-13，获得时间戳，并显示在超级终端上。

0 键被按下：设置时间日期为 2008-1-12, 4：13：3，获得时间戳，并显示在超级终端上。

A 键被按下：设置时间为 4：13：3，获得时间戳，并显示在超级终端上。

B，C，D 键被按下，无程序语句执行。

5.5.7 实验思考

如何改写程序使其实现闹钟功能。

5.6 计时器管理实验

5.6.1 实验目的

1、理解计时器的工作原理。

2、掌握定时器管理器模块的内部结构。

3、进一步掌握任务之间的信号量通信机制。

5.6.2 实验要求

深刻理解计时器内部模块、工作原理和各个函数的用法，掌握多任务下计时器的定时方法。该实验通过计时器检测超时情况，在启动一个操作的同时，启动一个计时器，监视条件的变化情况。若条件满足停止计时器，若计时器超时，则停止操作，并发出一个相应的信号。

编写一段程序使其实现两个倒计数计时器，每个计时器的精度为 0.1 秒。第一个计时器设置为 5 秒，第二个计时器设置为 10 秒。在指定的时间到达后 LCD 显示屏上输出“0 Out !”或“1 Out!”以标记相应的计时器达到指定的计时时间；在超级终端上也能观察到类似的结果。

5.6.3 实验准备

在做本实验之前应严格要求自己把 GB60 数据手册有关定时器部分的相关内容预习一遍。参考邵贝贝教授编著的《单片机认识与实践》第六章单片机片内 I/O 模块中的定时器一节，掌握定时器管理器的基本功能和内部结构；参照《嵌入式构件》的第七章计时器管理器理解计时器的工作原理和编程方法，计时器的数据结构，各个函数的具体作用以及各函数的参数设置。在预习的过程中应思考在多任务下如何利用定时器溢出中断的功能进行系统时钟编程？

5.6.4 实验指导

计时器管理器的工作原理是计时器内部含有计数寄存器，首先要设定初值，然后启动计时器。这样计时器就从初值开始累计减少（增加），这样减少（增加）到一定的数值后，计时器模块就发生一个中断，然后根据计数模式跳到合适的值重新计数，循环往复。当启动一个操作，等待一定的时间，然后停止操作这样的场合，计时器有非常重要的作用。通常该过程如下：

（1）启动一个操作（打开或关闭输出设备）。

（2）启动计时器。

（3）当计时器超过了所设定的时间后，停止操作（关闭或打开输出设备）。

这时就能从超级终端和 LCD 液晶显示器上观察输出效果。

分析定时器模块内部结构如下图所示：

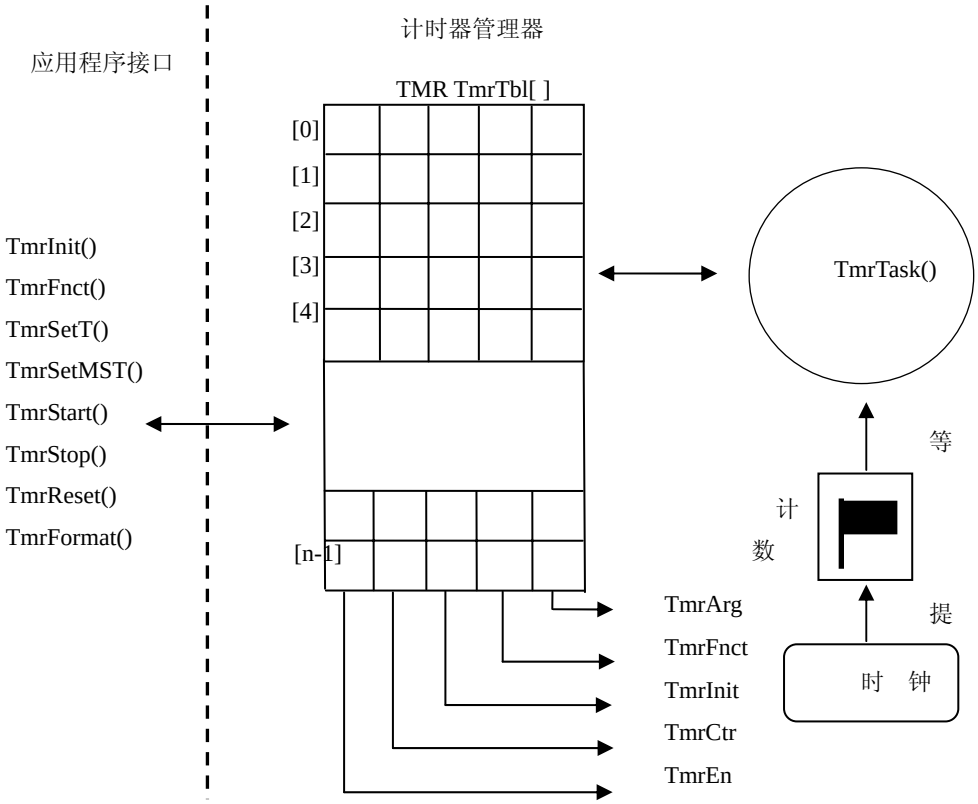

图 5-10

图 5-10 描述了计时器管理器模块的流程图，在此假设存在一个实时内核，该模块包括一个每 0.1 秒执行一次的简单任务，时钟脉冲 ISR 每经过 0.1 秒后就对计数信号发送一个信号。时钟脉冲就记录在信号中。

在应用程序中，计时器管理器任务负责更新尽可能多的倒计数计时器（在 TPM_MAX_TMR 中定义），最多可有 250 个计时器。当执行计时器管理器任务时，对每个有效的计时器，它扫描 TmrTbl[]中的所有元素。TmrTask()递减 TmrTbl[I].TmrCtr。若计时器为 0，则执行用户定义的函数。

5.6.5 实验结果

在指定的时间到达后 LCD 显示屏上输出“0 Out !”或“1 Out!”以标记相应的计时器达到指定的计时时间；在超级终端上也能观察到类似的结果。

5.6.6 实验思考

计时器管理器模块除能够维护精确的时间外在该模块中通过设置寄存器还能实现什么样的功能。

5.7 离散输入输出实验

5.7.1 实验目的

1．熟悉通过读取离散输入的值来控制离散输出。

2．掌握离散输入／输出模块利用信道管理应用程序以及各个接口函数的调用机制。

5.7.2 实验要求

1．通过拨位开关提供数字量的输入模块。当拨位开关状态位 OFF 时，相应的导线接点上的电压为(高“1”)；当拨位开关状态位 ON 时，相应的导线接插点上的电压为低（“0”)。

2．LED 模拟离散量的输出模块。当导线接插点的接入电平为高时，LED 处于暗状态；当导线接插点的接入电平为低时，LED 处于亮状态。

5.7.3 实验准备

参考《嵌入式系统构件》第八章，熟悉离散 IO 模块任务的管理：离散输入／输出模块由一个任务组成(DIOTask())，该任务按一定的时间间隔(DIO_TASK_DLY_TICKS)执行。DIOTask()可以管理应用程序所需要的多个离散输入和输出(每个最多 250)。离散输入／输出管理器调用 DIOInit()进行初始化。每个 DIO_TASK_DLY_TICKS，DIOTask()调用 DIRd()，DIUpdate()，DOUpadate()和 DOWr()。

5.7.4 实验指导

离散输入：

微处理器只需读取输入口，屏蔽不需要的输入，并根据输入状态做出判断。每个离散输入可以认为是一个逻辑信道。离散输入输出模块允许按需拥有逻辑信道数数量（最多 16 个）。

DITbl[]是一个表，包含每个离散输入信道的配置和运行信息。当离散输入信道 DICfgEdgeDetectFnct()配置了边沿检测并检测到转换时，执行用户定义的函数。DICfgMode()设置离散输入信道的操作模式，通过 DIGet()获取离散输入信道的当前值，所有信道值的改变和设置都要写入到 DITbl[]表中，离散输入任务

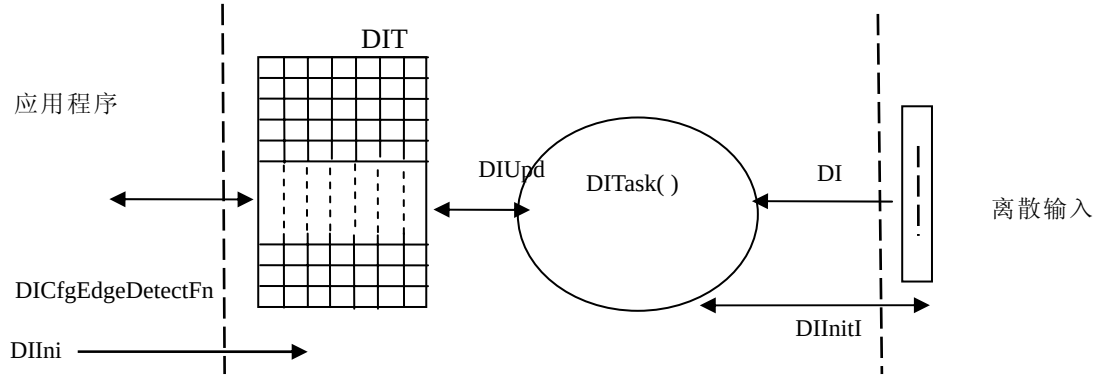

图 5-11

DITASK()通过 DIUpdate()函数更新和获得每个输入信道的信息。离散输入再通过硬件接口函数 DIRd()来读取并映射到 DITbl[i].DIIn 中。DIInitIO 是由 DIInit()调用的初始化代码，用来初始化物理硬件端口。

离散输出：

离散输出需要提供的微处理器具有足够多的并行锁存输出线。给每个离散输出一个逻辑地址通常从 0 到 15。每个离散输出可以认为是一样逻辑信道。该离散输入输出模块允许拥有最多 16 个逻辑信道。

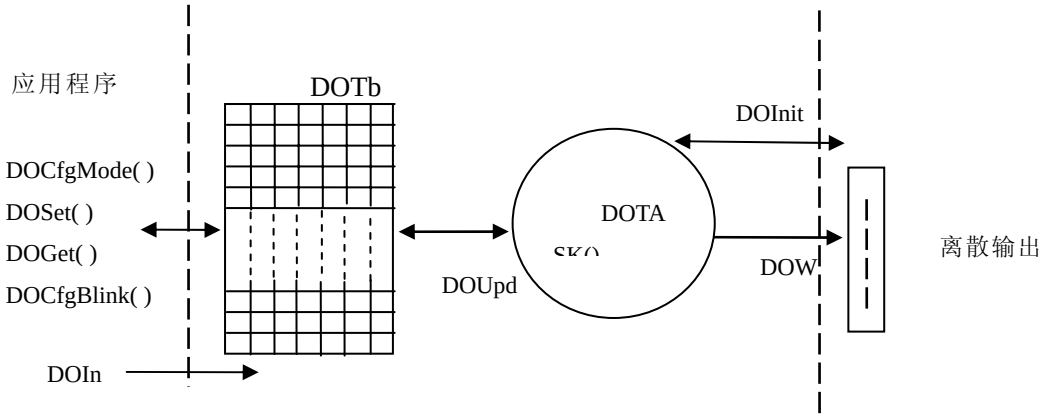

图 5-12

通过 DOInit()函数调用离散输出模块，DOInitIO 是由 DOInit()调用的初始化代码，用来初始化物理硬件端口。DOTbl[]是一个表，包含每个离散输出信道的配置和运行信息。该表中的值通过应用接口函数发生改变，并且离散输出任务 DOTASK()通过 DOUpdate()函数更新每个输出信道，再将更新后的值写入到该表中，DOTbl[]表在接口函数和离散输出任务之间起到了一个枢纽的作用，离散输出再通过接口函数 DOWr()将 DOTbl[i].DOOut 映射到硬件中。

5.7.5 实验结果

S08MCU 母板上的第一个和第二个 LED 指示灯以 50%的频率闪烁,第三个指示灯以 25%的频率闪烁。

5.7.6 实验思考

怎样通过离散输入控制灯的亮、灭，以及闪烁的频率？

5.8 模拟输入输出实验

5.8.1 实验目的

1.进一步理解 S12 单片机 ATD 模块的基本功能。

2.掌握模拟输入的过程原理。

3.掌握在 uCOS-II 下模拟输入模块的编程方法。

4. 理解多任务下的 ATD 实验的程序设计并与驱动程序比较有何不同。

5.8.2 实验要求

1.理解和掌握模拟输入模块的工作原理和各个函数的用法，掌握多任务下模拟输入的过程。结合 LCD 模块，在操作系统下，利用 ATD 模块的功能，把输入的模拟量转化为数字量的结果显示到 LCD 显示板上。

2.明确模拟输入的整个过程，即采样，滤波，量化以及相应的要用到的多路复用器，放大器，滤波器等各个器件的功能以及掌握模拟数字转换器的读取方法。

5.8.3 实验准备

首先回顾有关 ATD 模块的原理，然后熟悉模拟输入的整个过程以及 ADC 的读取方法，根据实验要求的不同，可采取不同的 ADC 读取方式，详细内容请参考《嵌入式系统构件》第 10 章“模拟输入/输出”。

在该实验中，用 LCD 显示出模拟数字转化的结果，所以要重新温习实验三“字符 LCD 显示实验”。虽然用到了 LCD 的显示，但是内部数据的传递是通过指针和消息邮箱来完成的，所以在做该实验之前要理解指针和消息邮箱的用法，请参考《嵌入式微控制器》第十四章的

第四小节“指针”和《嵌入式实时操作系统》第二版的第十章“消息邮箱管

5.8.4 实验指导

有关模拟数字的转换原理在此不累述，相关信息前参考该实验手册第四章实验八“模拟数字转换实验”。由于模拟输入要经过传感器，放大器，滤波器，多路复用，模拟转换器，最后到 MCU，要理解和掌握这一过程。根据不同的实验可能用到的传感器不同，要根据每个传感器的芯片手册来了解该传感器的特性以及用法。多路复用器的工作原理也需要掌握，接下来介绍一下如何读取模拟数字转换器（ADC）。

我们可设计驱动程序（函数）读取一个模拟输入信道，并将转换返回到应用程序。应用程序调用驱动程序，并传递给所希望的信道来读。驱动程序选择（通过多路复用器）所所想要的模拟信道开始读。开始转换前，或许需要等待几毫秒以便使信号通过多路复用器传播并使之稳定下来。如果不等多路复用器的输出稳定下来，则读取的数也可能不稳定。下一步，A/D 转换被触发以开始转换。然后驱动程序延迟一段时间以完成转换。此时要注意延迟时间必须比转换时间更长。延迟之后，驱动程序假设已完成并读取转换器。然后将二进制结果返回到应用程序。实际上读取模拟数字转换器的方法是取决于 ADC 把模拟电压转换成二进制代码的快慢。详细内容请参考《嵌入式系统构件》第 10 章“模拟输入/输出”。

因为该实验是在多任务下进行的，有必要理解在 μC/OS-II 平台上工作的多任务模拟输入输出模块的工作流程。

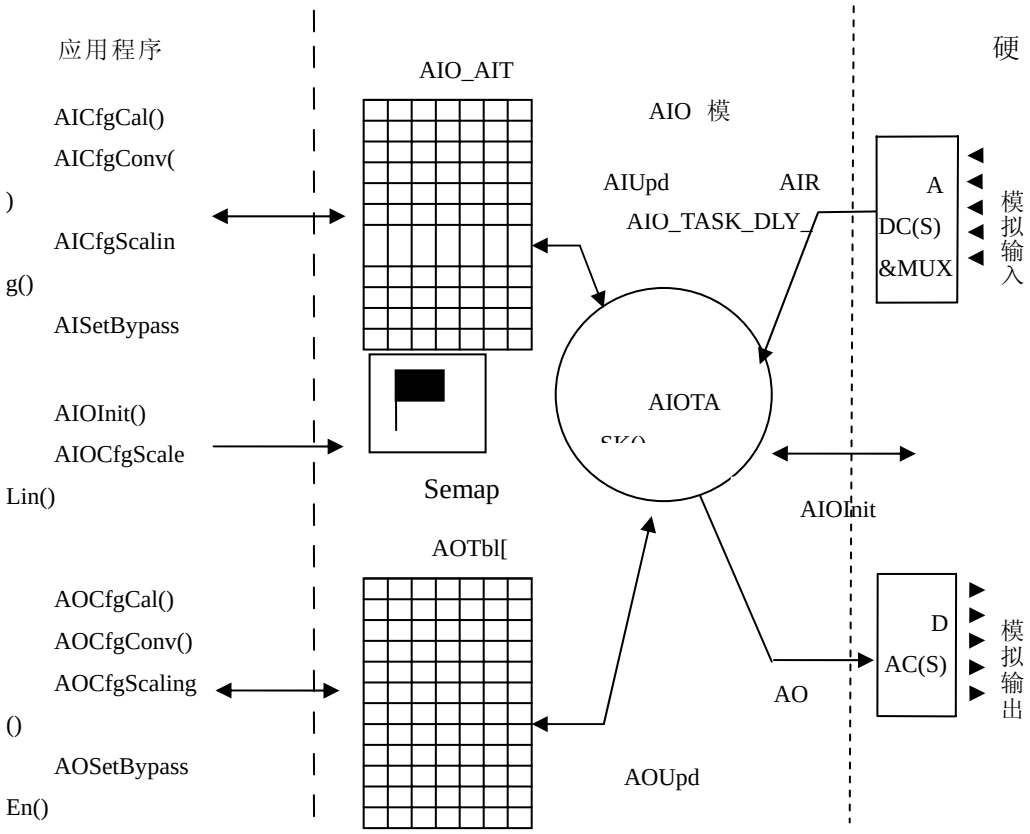

图 5-13

在多任务模式下，模拟输入输出模块，可允许读取并换算多大 250 个模拟输入信道，并可以更新所有 250 个模拟输出信道 。每个模拟输入信道在一个固定的时间间隔内扫描，并且可以对每个信道进行编程；同样，每个模拟输出信道在一个固定的时间间隔内更新，并且可以对每个信道的更新速率进行编程。模拟输入输出模块必须通过访问 AIOInit()进行初始化；AIOInit()进行初始化所有模拟输入信道、模拟输出信道、硬件、用于保证仅访问内部数

据结构（由模拟输入/输出模块所使用）的信号量，最终 AIOInit()创建 AIOTask()。

在此流程图中列出了许多应用程序接口函数，要理解和掌握这些函数的功能和应用。这些函数的功能和应用在《嵌入式系统构件》第 10 章“模拟输入/输出”中详细介绍，请参考。

在参考程序中 AIGet()是来获得模拟输入信道的当前值。主函数中我们设计了 3 个任务分别实现启动任务,LCD 显示任务，测试模拟输入任务。同时用了两个邮箱来传递数据，LCD 显示任务就是显示由两个消息邮箱发送来的数据。

5.8.5 实验思考

1．怎样调整在同一时间内模拟输入通道的扫描次数（不同的通道扫描次数不同）？

2．怎样调整在同一时间内模拟输出信道的更新次数？

3．怎样通过编程修改显示结果？

4．如何实现不同方式的模拟输出？

5.9 异步串行通信实验

5.9.1 实验目的

1. 熟悉 uCOS-II 下串口驱动程序的工作原理，数据的发送和接收机制。

2. 掌握信号量在通信中是如何实现在 uCOS-II 中的同步机制以及超时等待机制。

5.9.2 实验要求

在多任务模式下，通过信号量机制和缓冲区的作用实现异步串行通信中数据的接收和发送。

5.9.3 实验准备

1．熟悉异步串口通信的三种基本工作方式：

◆ 单工（Simplex）模式：数据传送是单向的，一端为发送端，另一端为接收端。这种传输方式中，除了地线之外，只要一根数据线就可以了。有线广播就是单工的。

◆ 全双工（Full-duplex）模式：数据传送是双向的，且可以同时接收与发送数据。这种传输方式中，除了地线之外，需要两根数据线，站在任何一端的角度看，一根为发送线，另一根为接收线。一般情况下，MCU 的异步串行通信接口均是全双工的。

◆ 半双工（Half-duplex）模式：数据传送也是双向的，但是在这种传输方式中，除了地线之外，一般只有一根数据线。任何一个时刻，只能由一方发送数据，另一方接收数据，不能同时收发。在 freescale 的 S12 系列 MCU 中，监控模式的通信就采用这种方式。其三种基本的串行通信模式如图 5-14：

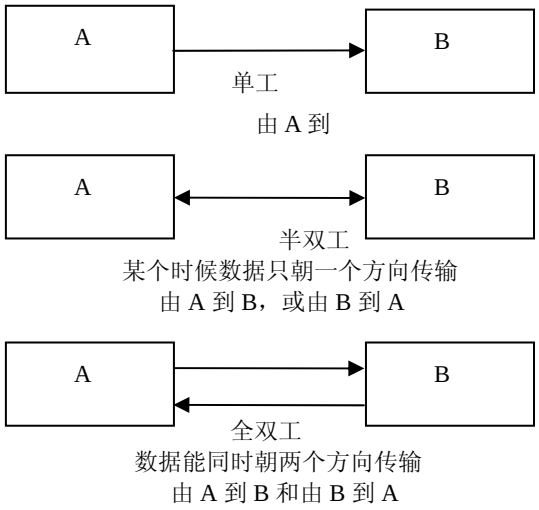

图 5-14　三种基本的串行通信模式

2．学习平台设计异步串行通信模块的三层模式，上层是应用程序接口，中间层负责数据处理和缓冲，下层负责直接驱动硬件，当硬件不同时，需要修改下层。

3．了解缓冲区的作用：由于串行设备存在外设处理速度和 CPU 速度不匹配的问题，所以需要一个缓冲区．实际上，单片机的异步串口中只有两个相互独立、地址相同的接收、发送缓冲寄存器 SBUF。在实际应用中，需要从内存中开辟两个缓冲区，分别为接收缓冲区和发送缓冲区。异步串行通信在多任务程序中使用了一个环状缓冲区将接收到的字符保存在缓冲区中，应用程序获得字符实际上是通过环状缓冲区获得的，这样做的目的是使系统可以在数据量较大时不会因为数据过多过快而遗失数据。

5.9.4　实验指导

每个串行口安排有两个环状缓冲区,同时有两个信号量：一个指示接收字节，另一个指示发送字节。两个缓冲区都存储在称做 COMM_RING_BUF 的结构中。每个环状缓冲区有以下四个要素：

1）储存数据（INT8U 数组）；

2）包含环状缓冲区中字节数的计数器；

3）环状缓冲区中指向被放置的下一字节的指针；

4）环状缓冲区中指向将被取出的下一个字节的指针。

接收数据：

应用程序需要调用 CommGetChar()获得接受的数据，在缓冲区空时会将任务挂起，为 CommGetChar()指定挂起超时可防止挂死应用程序，当收到字节时，任务将被“唤醒”，同时从串行端口接收字节。如下图所示：

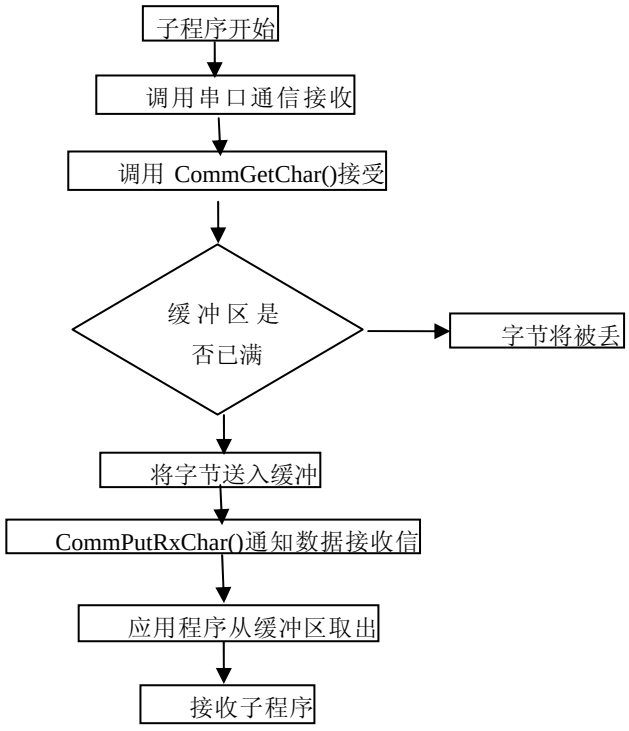

图 5-15

CommPutRxChar()是介于中间层和下层之间的接口函数。在接收字节的中断处理函数中，需要调用本函数。如果接收环状缓冲区还未满，则 CommtPutRxChar()将字节送入缓冲区。如果接收环状缓冲区已满，则该将被丢弃。当字节插入到缓冲区时，CommPutRxChar()

通知数据接收信号量，使之将数据已到的消息传达给所有的等待任务。

驱动的下层包括两个函数，CommISR()是接收中断服务函数，CommISRHandler()接收中断服务处理函数。

发送数据：

当需要发送数据给串行口时，CommPutChar()等待信号量。在初始化时，发送信号量初始化为缓冲区的大小，因此，当缓冲区没有空间时，CommPutChar()就挂起应用程序。只要串口捕获到发送字节，挂起的任务就将恢复。

CommGetTxChar()是介于中间层与下层之间的接口函数。在由串口发送字节时，在发送字节的中断处理函数中调用该函数。该函数表示“请传递给我下一个将要发送的字节”。如果发送缓冲区中至少还有一个字节，CommGetTxChar()就返回下一个从缓冲区发送的字节，如果缓冲区已空，则 CommGetTxChar()返回 NUL 字符，并告诉调用者缓冲区中已无数据可发。这将使调用者停止发送中断，直到有数据发送为止。从缓冲区中取出字节时，数据发送信号量接到信号，并传递给等待发送的任务“在传输缓冲区中已空余空间”。程序运行如下：

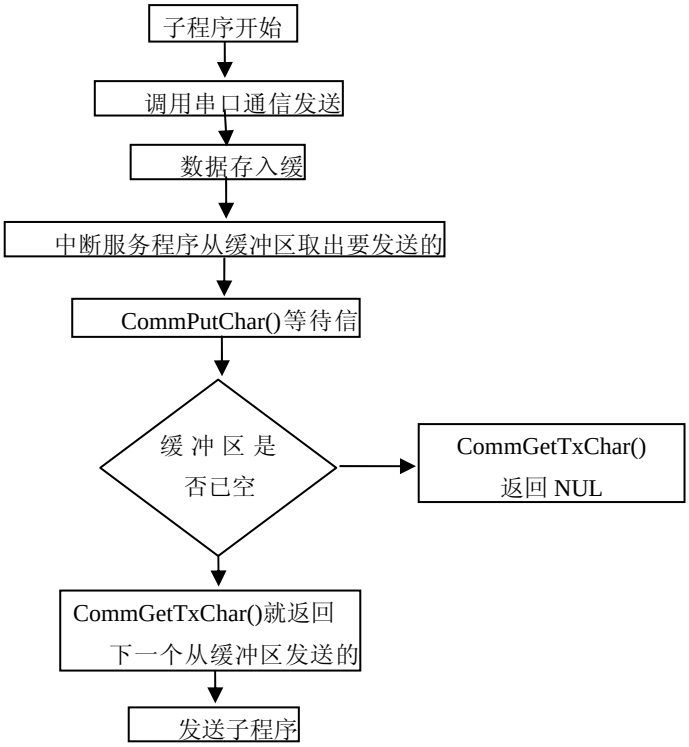

图 5-16

5.9.5 实验结果

超级终端上显示输入的字符，随着输入字符的增加 S12MCU 母板上的 LED 灯将累加点亮。

5.9.6 实验思考

接受发送字符和字符串是怎样实现的，它们之间有何区别;在多任务下串口通信模块在数据量较大时是怎样防止数据过多过快而遗失数据。

第6章　综合实例 UC/OS-II 下多 I/O 任务的实现

一、 实例简介

本实例集成了第五章的时钟日历模块、串口模块、键盘模块、LED 显示模块、模拟输入输出模块、离散输入输出模块共六个模块。

该实验通过串口显示 AD 采集的比例和电压值、日期和时间、最小电压值和最低电压值、八位离散量，以上显示的信息是一秒钟显示一次。

通过按键盘上的 0，1，2，3 键在动态数码管上分别显示，最低电压和最高电压、当前日期、当前时间、当前采集到的电压。

通过旋转电位器，当电压值不在最低电压和最高电压之间时，蜂鸣器会发出响声。

通过串口按照屏幕上的提示可以设置当前的日期时间和最低电压最高电压。

二、硬件系统

本实例用到的模块有 4*4 键盘模块、LED 模块、蜂鸣器模块、SPI 模块、ad 和 pwm 模块、串口模块。在以上实验中都已描述，此处不作赘述。

三、任务分析设计

1. 实例任务划分

本实例根据系统设计的要求，以及任务划分原理，该系统需要时钟日历模块、串口模块、键盘模块、LED 显示模块、模拟输入输出模块、离散输入输出模块。

通过键盘任务来识别当前按下的键，通过这个键码来对应 led 上显示的内容。

通过 LED 任务来显示要求显示的信息，如日期和时间、电压值等。

通过串口任务来获取串口发送过来的数据，通过这些数据来处理十分暂停显示信息还是修改数据。

通过 AD 任务采集数据并判断是否超出最低电压和最高电压的范围。

通过离散任务读取 SPI 口的离散开关量。

通过时钟日历模块在一秒钟发送一次采集到的数据，并把当前的日期和时间也发送到串口上，在超级终端上显示出来。

2. 实例任务设计

本实例采用多任务设计模式，故设计过程主要是分析需要的功能、划分任务、定义各任务的优先级、以及实现各个任务，当然还要考虑各个任务之间如何通信和同步。

实现这些任务的关键是多任务之间的信息交互和同步运行。如 AD 采集数据的中断处理；串口发送数据和接收数据任务的处理；通过 SPI 口采集离散数据等。

四、 程序详解

1.任务分析流程图

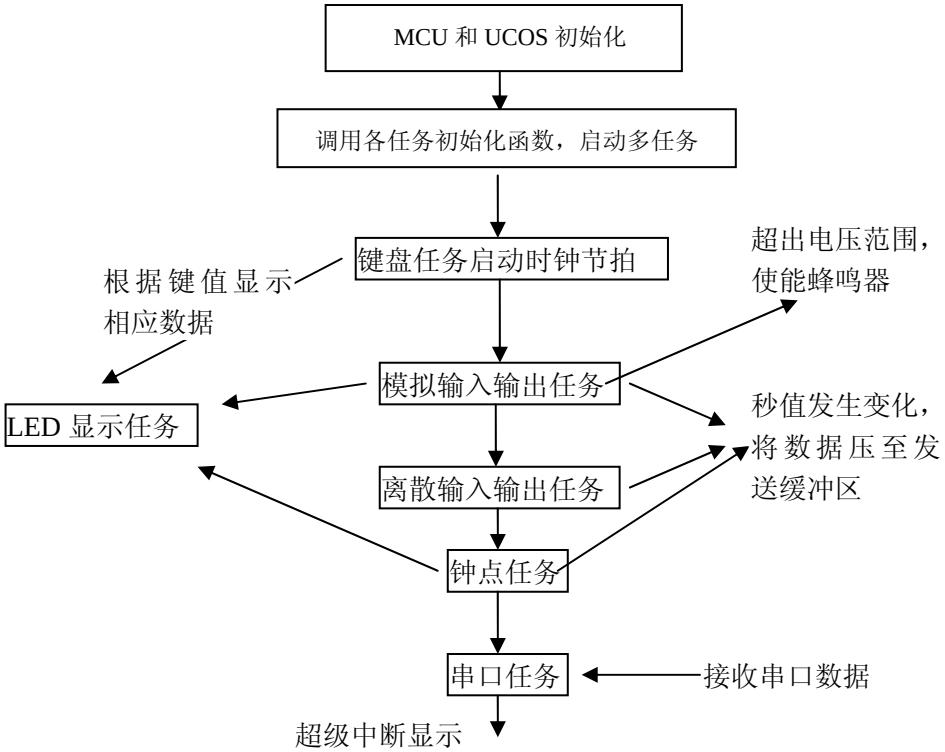

图 6-1

2.任务详细描述

（1）实时系统初始化

在调用μC/OS-Ⅱ的任何其它服务之前，μC/OS-Ⅱ要求用户首先调用系统初始化函数OSIint()。OSIint()初始化μC/OS-Ⅱ所有的变量和数据结构（见 OS_CORE.C）。

⑵硬件初始化

① 键盘初始化 KeyInitPort（）；

② 模拟输入输出初始化 AIOInit( )，该函数对 AD 模块和 PWM 模块进行初始化；

③ 离散输入输出初始化 DIOInit( )，该函数对 SPI 模块进行了初始化；

④ 串口初始化 sciint( )；

⑤ LED 液晶显示初始化 DispInit( )；

⑥ 定时器初始化 ECT_setup()，该函数是为钟点模块服务;

注：以上函数具体代码可参看程序包。

（2）主程序分析

①主程序介绍

```
void main(void)
{
   MCUInit();                    //芯片初始化
   OSInit();                     //ucos 初始化
   KeyInit();                    //创建键盘任务并初始化
    sciint();                    //创建串口任务并初始化
   ClkInit();                    //钟点任务初始化
   AIOInit();                    //模拟输入输出初始化
   DIOInit();                    //离散输入输出初始化
   DispInit();                    //LED 初始化
```

```
  ECT_setup();                  //定时器初始化
  disp_able=0;                  //串口显示使能，当按下 h 键时，串口不在发送采集到的数
                                  据，按下 g 时，重新发送数据
  disp_type=1;                  //LED 显示的数据类型
  setdata=0;                    //变量置 1 说明开始接收日期数据
  settime=0;                    //变量置 1 说明开始接收时间数据
  scicount=0;                   //累计接收数据的个数
  setad=0;                      //变量置 1 说明开始接收电压的最高值和最低值
  AD0_min=2.0;                  //预置 AD 的最大值和最小值
  AD0_max=3.5;
  AD0_alert=0;                  //电压蜂鸣器使能
  OSStart();                    //多任务启动
}
```

3. 键盘模块任务分析

（1）按键功能介绍

键盘只用到了 0-3 四个键，按下 4 个键分别对应 LED 上显示的内容。按下 0 键显示最高电压值和最低电压值；按下 1 键显示日期；按下 2 键显示时间；按下 3 键显示采集到的电压值和比例值。

（2）键盘任务

键盘任务首先需要根据键盘识别码确定哪一个键被按下，需要执行相应的操作。系统定义了一个全局变量 disp_type;

当按下 0 键时 disp_type=0，对应使能最低电压和最高电压的显示

当按下 1 键时 disp_type=1，对应使能日期的显示

当按下 2 键时 disp_type=2，对应使能时间的显示

当按下 3 键时 disp_type=3，对应使能当前电压的显示

4. LED 显示任务分析

LED 显示任务，因为 LED 时多路复用需要不断的刷新，所以作为任务的话，会浪费很多系统资源，所有作为定时器中断的方式执行，当需要显示数据的时候，其他任务只需调用显示接口，把相应的数据放入显示缓冲区即可，然后定时器中断到达时，直接刷新显示数据。如下所示：

```
void   tc_isr(void) {
 asm{
          ldaa $30        //save ppage to stack
          psha
          }
    OSIntEnter();
    OS_SAVE_SP();
  TFLG1_C0F=1;   //清中断标志
  DispMuxHandler();
  OSIntExit();//exit interrupt and task switch
          asm{
          pula
          staa  $30       //restore ppage from stack
```

```
        nop
        rti
        }
}
```

因为是在多任务程序下，并且程序放在不同的页面之间，所有在中断到来的时候，要保存当前的页面及 SP 地址。如同 UCOS 实时中断节拍的中断处理程序。

5.钟点任务

该任务的作用是在串口上显示相应的日期和时间，并每秒钟向串口打一次数据。该任务的具体程序在第五章已经讲到。

在秒值发生变化时，需要执行以下程序，判断当前 LED 显示什么数据，判断是否向串口发送数据。具体的程序细节参照 ClkUpdateTime（）中新加的程序代码。

6.模拟输入输出任务

该任务初始化了 MCU 的模拟模块和 PWM 模块，其中蜂鸣器用的 PWM1 模块，当超出电压范围时使能 PWM1 模块。该任务没两秒中采集一次数据，可以根据自己的情况更改任务执行间隔时间。具体的函数接口请参见第五章或者参照源码。该任务把采集到的数据都放到一个结构体中，当秒值发生变化时，判断十分显示当前的数据，如果显示就获取当前的电压值和比例值并通过调用串口接口函数 CommPutChar（）将数据压入发送缓冲区。

另外模拟输入采集到数据后会把电压值再转成比例值通过 PWM 输出，旋转电位器可看到 LED 的亮度会发生变化。

7.离散输入输出任务

该任务初始化了 SPI 接口，通过 SPI 总线采集八位拨动开关的离散值，然后把这些离散值在串口上显示，并在对应的八位 LED 上显示。

8.串口任务

串口任务通过中断接收数据，其中断服务处理程序只是把数据压入串口接收缓冲区。然后待串口任务执行时，读取串口的数据，并判断十分暂停串口输出采集到的数据或者是否开始设置日期和时间并开始介绍新数据。具体的任务流程请参考 SCIsendtask（）；

9. 程序的装配

程序的装配如下面的代码所示：

```
NAMES END
SEGMENTS
        RAM             = READ_WRITE    0x0800 TO     0x3FFF;
        ROM_4000        = READ_ONLY     0x4000 TO     0x7FFF;
        ROM_C000        = READ_ONLY     0xC000 TO     0xFEFF;
        PAGE_38         = READ_ONLY     0x388000 TO 0x38BFFF;
        PAGE_39         = READ_ONLY     0x398000 TO 0x39BFFF;
        PAGE_3A         = READ_ONLY     0x3A8000 TO 0x3ABFFF;
        PAGE_3B         = READ_ONLY     0x3B8000 TO 0x3BBFFF;
        PAGE_3C         = READ_ONLY     0x3C8000 TO 0x3CBFFF;
        PAGE_3D         = READ_ONLY     0x3D8000 TO 0x3DBFFF;
END
PLACEMENT                                       //各任务存放的地址
    TASK1CODESEG,                               //key
    TASK1STRINGSEG      INTO    PAGE_38;
    TASK2CODESEG,                               //sci
```

```
        TASK2STRINGSEG      INTO    PAGE_39;
        TASK3CODESEG,                       //led
        TASK3STRINGSEG      INTO    PAGE_3A;
        TASK4CODESEG,                       //clk
        TASK4STRINGSEG      INTO    PAGE_3B;
        TASK5CODESEG,                       //AIO
        TASK5STRINGSEG      INTO    PAGE_3C;
        TASK6CODESEG,                       //DIO
        TASK6STRINGSEG      INTO    PAGE_3D;
        DEFAULT_ROM         INTO    ROM_C000;
        DEFAULT_RAM         INTO    RAM;
    END
    ENTRIES
    KeyScanTask                         //各任务函数
    ClkTask
    AIOTask
    DIOTask
    SCIsendtask
    END
    STACKSIZE 0x1024                    //堆栈大小
    VECTOR 0 _Startup                   //程序入口
    VECTOR ADDRESS 0xFFF0 OSTickISR     //时钟节拍处理程序
    VECTOR ADDRESS 0xFFF6 OSCtxSw       //任务级切换处理程序
    VECTOR ADDRESS 0xFFD6 get_char      //串口获取数据
    VECTOR ADDRESS 0xFFD8 SPI_get       //获取离散数据
```

10. 程序运行的结果

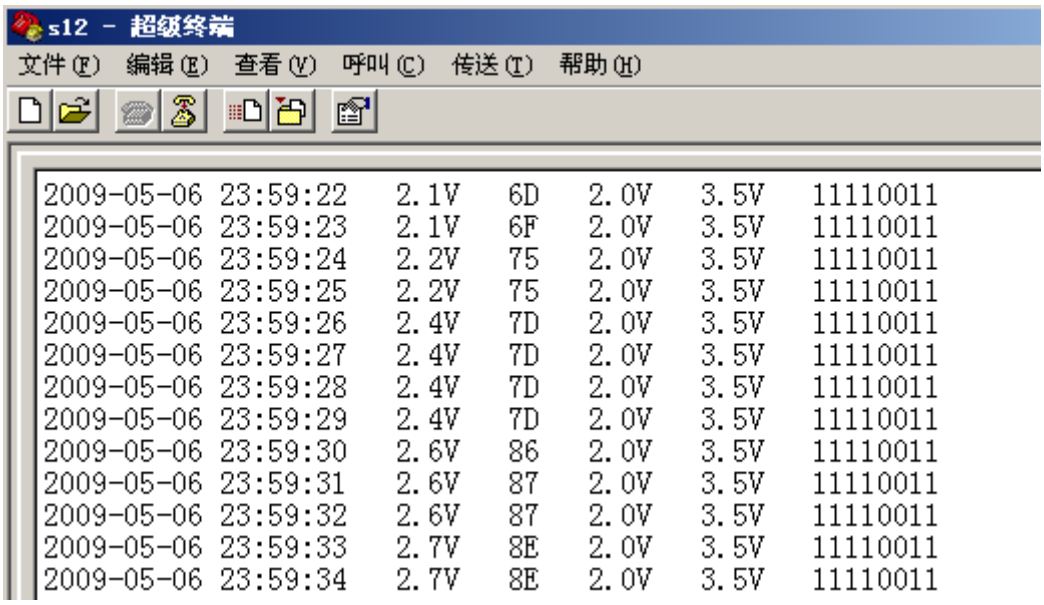

图 6-2

左侧两列显示的是日期和时间，每秒钟上面的数据更新一行。第 3 列显示的是电压。第 4 列显示的是比例值，通过旋转电位器，这两列值可以改变。第 5，6 列显示的是当前最高

的电压和最低的电压，可以通过串口设定这两列的值。最后一列显示的是当前采集到的离散值。

附录 A　S12ADB 原理图

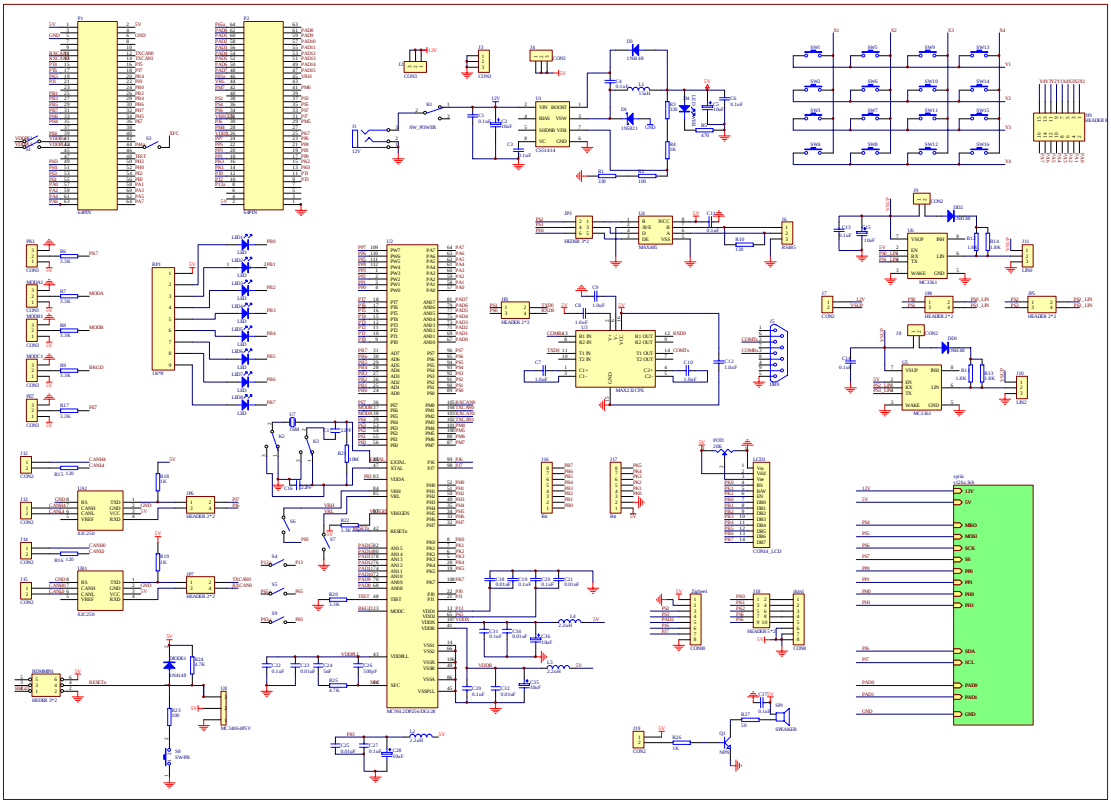

图 A-1 S12MCU 基本模块原理图

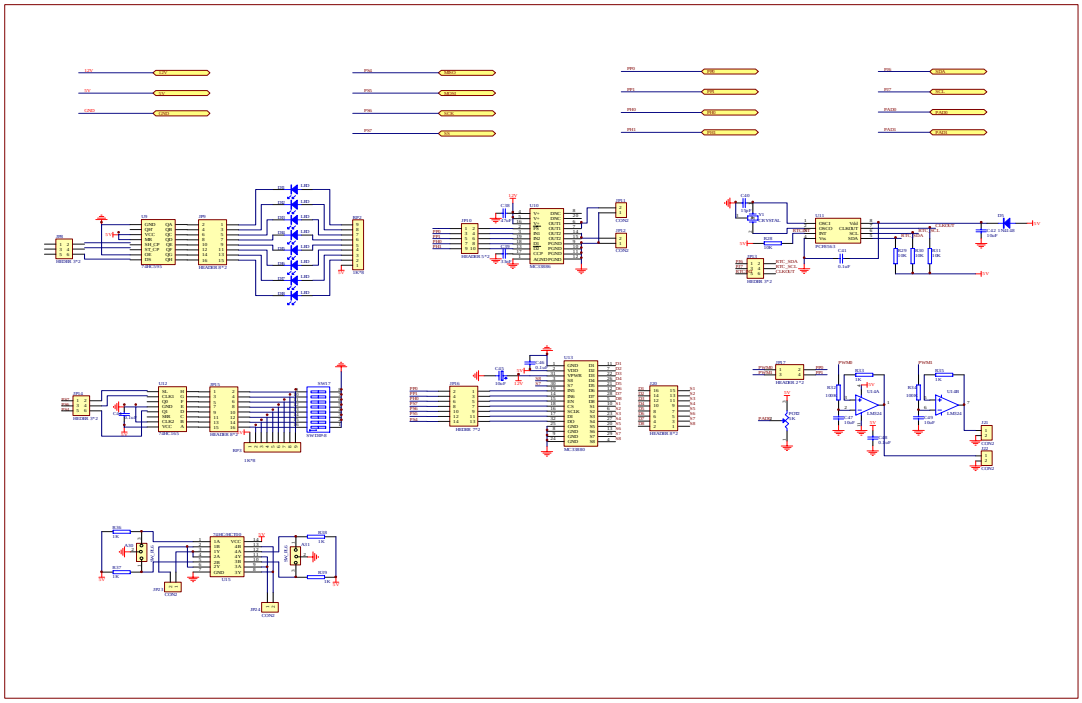

图 A-2 S12MCU 扩展模块原理图

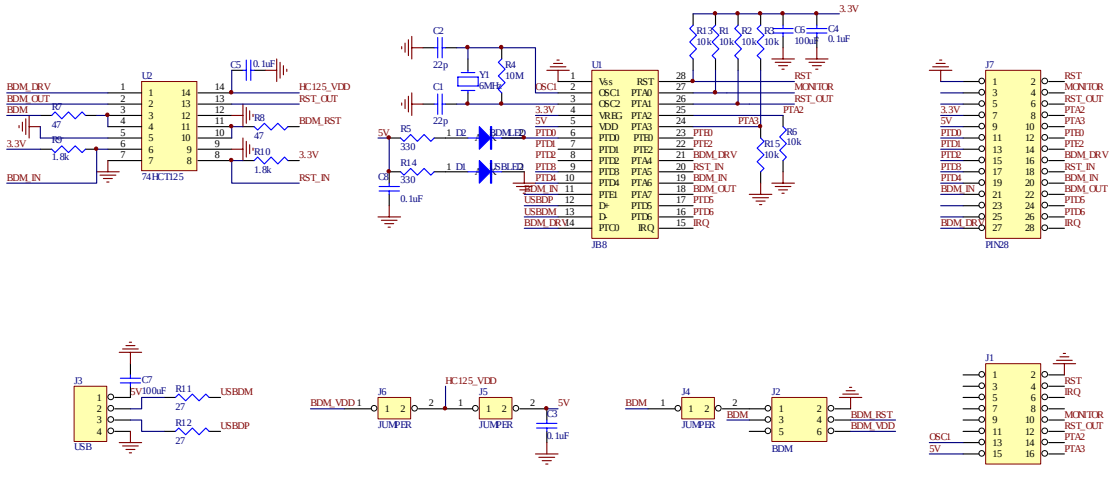

图 A-3 S12MCU BDM 调试器原理图

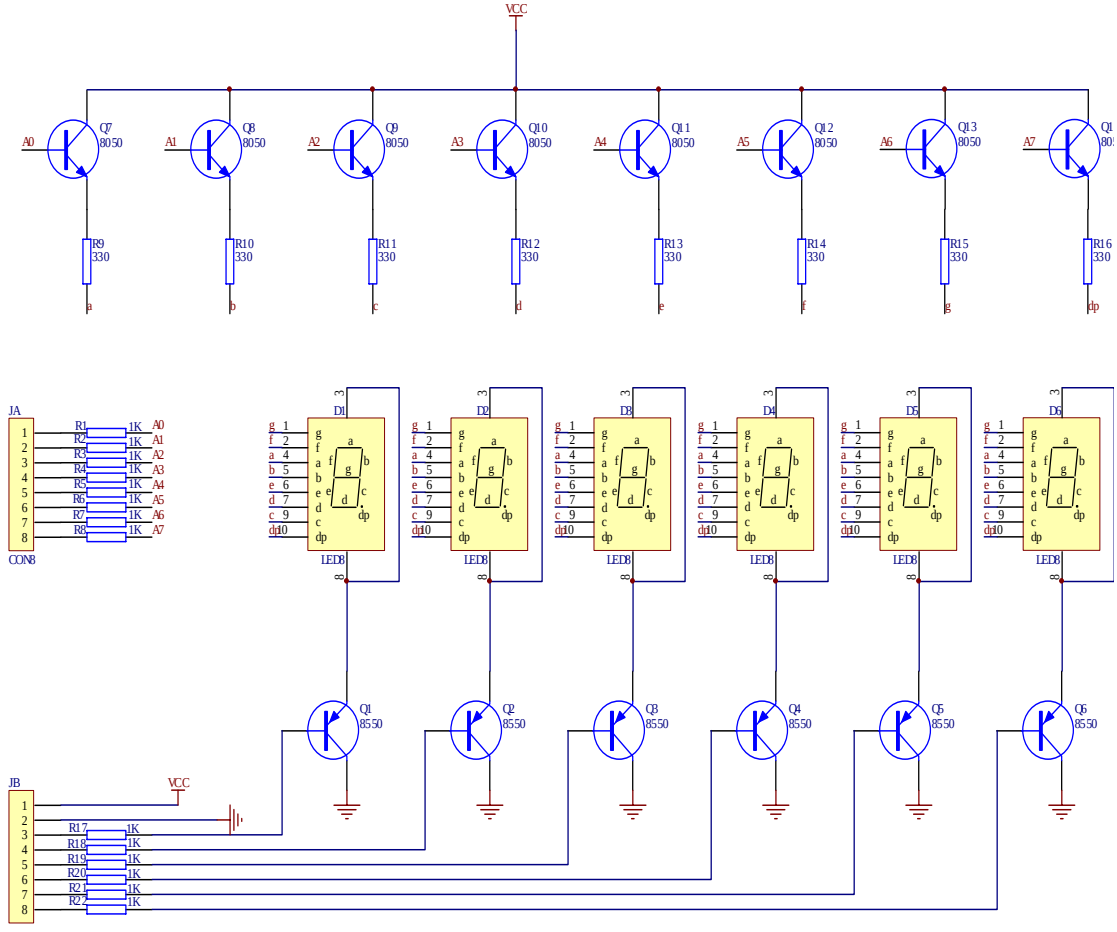

图 A-4 多路复用 LED 原理图