蘇州大學

硕士专业学位论文

（2005届）

一个小型实时操作系统的设计与应用

The Design and Application of One Small Real Time Operating System

研究生姓名 杨增汪

指导教师姓名 王宜怀（教授）

专业名称 软件工程

研究方向 嵌入式系统

论文提交日期 2005年4月

摘 要

本文着重研究了一种基于 MC9S08GB60 单片机的嵌入式实时操作系统(Real-Time OS)的设计方法。该系统以飞思卡尔（原摩托罗拉）公司生产的 HCS08 系列单片机中的 MC9S08GB60 CPU 为硬件实现平台。由于半导体工艺的发展和芯片设计水平的进步，MCU 的性能大幅度地提高，应用范围更加广泛。原先的前后台软件设计方法很难适应这种变化。为了将它的强大功能与人们对信息产品的更高要求结合起来，我们采用嵌入式实时操作系统的软件设计方法取代传统的前后台(超循环)软件设计方法。

RTOS 是一种新的系统设计思想和一个开放的软件框架。它具有操作系统的基本功能，可以对整个实时系统的运行进行控制，能根据系统中各个任务的轻重缓急，合理的在它们之间分配 CPU 和各种资源的占用时间。利用信号量、邮箱等功能提高 CPU 的使用效率。实时操作系统的实时多任务机制不仅能使应用系统满足实时性要求，而且简化了系统的开发设计过程，方便了系统扩展。

本论文共由 5 个部分构成。首先，我们简要介绍了嵌入式 RTOS 的基本概念；其次，描述了 MC9S08GB60 的最小系统，通常包括 PCB 板、监控程序和在线编程系统；接着，详细阐述了 SDF-RTOS 的内核结构、任务管理、时间管理及任务通讯和同步等模块，并给出测试程序及对内核占用资源进行分析；硬件开发平台的设计和暖气表的工作原理；再者，以暖气表为例，介绍应用系统的硬件和软件设计方法；最后，对本论文课题作了简要总结，叙述了尚待进一步解决的问题。

关键词：嵌入式 RTOS 最小系统 MC9S08GB60 RTOS 软件设计

作 者：杨增汪

指导老师：王宜怀

Abstract

This dissertation mainly introduces a method that design a Real-Time operation system based on MC9S08GB60. The smallest system of MC9S08GB60 is used as hardware platform, MC9S08GB60 is one of HCS08 series microprocessors produced by Freescale(a subsidiary of Motorola) Company. With the rapid progress of semiconductors producing technology and the higher level of chip designing, the performance of the MCU (Microcontroller Unit) has been improved greatly, and the application of it has been scaled greatly. It is difficult for the previous former-background software design method to adapt these changes. So we adopt Embedded Real-Time Operation System software design method that replaces traditional former-background (super loop) method, in order to combine its powerful function with the more demands that people bring forward for IT products.

RTOS is a new system design conception and an open software frame. It has operation system function that can control the whole Real-Time system and in order of importance and urgency of every task, it rationally assigns CPU and different resources holding time between them. It improves the service efficiency of CPU using semaphore, mailbox and so on Functions. The multitasking of RTOS not only meets the real time requirement of the embedded system but also simplifies the design process and makes it easier to expand the system.

This thesis is composed of five parts. In the first place, we briefly introduce the basic concepts of the Embedded RTOS; then, describe the smallest system of MC9S08GB60 MCU, which includes PCB board、monitor program and in-circuit programming system; thirdly, explain the design of core structure、task management、time management、task communication and synchronization and so on in detail, supply the test program and analyse the source which engrossed by the kernel; fourly, discuss the design method of the hardware and software on application system, it is a right example for the calorimeter of central heating; at last, handle this subject concisely and discusse some unsolved aspects of our studies and probe into the prospect of SDF-RTOS.

Key words: Embedded RTOS the Smallest System MC9S08GB60 RTOS Software Design

Written by Yang zengwang

Supervised by Wang Yihuai

苏州大学学位论文独创性声明及使用授权声明

学位论文独创性声明

本人郑重声明：所提交的学位论文是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含其他个人或集体已经发表或撰写过的研究成果，也不含为获得苏州大学或其它教育机构的学位证书而使用过的材料。对本文的研究作出重要贡献的个人和集体，均已在文中以明确方式标明。本人承担本声明的法律责任。

研究生签名：杨婷婷　日　期：2005.4.23

学位论文使用授权声明

苏州大学、中国科学技术信息研究所、国家图书馆、清华大学论文合作部、中国社科院文献信息情报中心有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括刊登）论文的全部或部分内容。论文的公布（包括刊登）授权苏州大学学位办办理。

研究生签名：杨婷婷　日　期：2005.4.23

导师签名：王宜怀　日　期：2005.4.23

第一章 绪 论

当今世界，以计算机技术、芯片技术和软件技术为核心的数字化技术取得了迅猛发展，不仅广泛渗透到社会、经济、军事、交通、通信等相关行业，而且深入到家电、娱乐、艺术、社会文化等各个领域，掀起了一场数字化技术革命。现代控制技术、多媒体技术与 Internet 的应用与普及，促使消费电子、计算机、通信一体化（3C）趋势步伐加快，嵌入式技术再度成为一个研究热点。

根据 IEEE（国际电气和电子工程师协会）的定义，嵌入式系统是“控制、监视或辅助设备、机器或甚至工厂操作的装置”[1]。由此可见嵌入式系统是硬件和软件的综合体。通常硬件部分包括处理器/微处理器、存储设备、外设器件、I/O 端口及图形控制器等；软件部分包括操作系统软件(OS)和应用程序，应用程序控制着系统的运作和行为，操作系统控制着应用程序编程与硬件的交互作用，有时设计人员把这两种软件组合在一起。

1.1 实时操作系统简介

1.1.1 实时操作系统概述

实时操作系统（Real Time Operating System, RTOS）是嵌入式应用软件的基础和开发平台。目前大多数嵌入式开发还是在 MCU/DSP 上直接编程，由一个主程序负责调度各个任务，没有 RTOS 的支撑。RTOS 是一段嵌入在目标代码中的程序，系统复位后首先执行，相当于用户的主程序，用户的其他应用程序都建立在 RTOS 之上。不仅如此，RTOS 还是一个标准的内核，将 CPU 时间、中断、I/O、定时器等资源都包装起来，留给用户一个标准的 API（系统调用），并根据各个任务的优先级，合理地在不同任务之间分配 CPU 时间。

1.1.2 RTOS 的一般体系结构

RTOS 的体系结构如图 1－1 所示。

一般来说，RTOS 内核表现为微内核(Micro Kernel)结构。所谓“微内核”是指将核心功能（如进程管理、任务通信、中断处理、进程调度）放在内核中，而将那些不是必需的功能和服务（如文件系统、存储管理、网络通信、设备管理等）作为内核之上可配置的部分。这样，整个操作系统就是由提供一些基本服务机制的微内核加上一些服务进程构成，系统的各个系统调用和服务都是由内核发消息到不同的服务进程，服务进程执行相应的操作，然后以消息的方式返回到内核。

1.1.3 实时操作系统与通用操作系统的区别

设计目标:作为由分时系统发展起来的通用操作系统，其设计的目标是追求最大的吞吐率、使整体性能最佳。而实时操作系统设计的目标是采用各种算法和策略，始终保证系统行为的可预测性(predictability)。由于目标的不同，因此在很多资源调度策略的选择上以及操作系统实现的方法上两者都具有较大的差异。

调度原则:通用操作系统为了达到最佳整体性能，其调度原则强调的是公平。而实时系统多采用基于优先级的可剥夺的调度策略，实时内核 uC/OS 正是这样的。

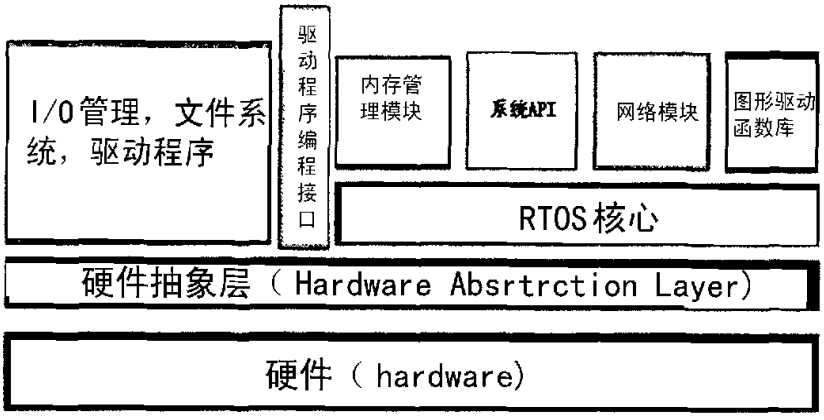


图 1－1 RTOS 体系结构图

内存管理:通用操作系统广泛使用了虚拟内存的技术，为用户提供一个功能强大的虚拟机。但因虚存机制引起的缺页调页现象会给系统带来不确定性，因此实时系统很少或有限的使用虚存技术。

系统的可重入性:在通用操作系统中，核心态系统调用往往是不可重入的，当一低优先级任务调用核心态系统调用时，在该时间段内到达的高优先级任务必须等到低优先级的系统调用完成才能获得 CPU，这就降低了系统的可预测性，因此，实时操作系统中的核心态系统调用往往设计为可重入的。

中断处理:通用操作系统中，大部分外部中断都是开启的，中断处理一般由设备驱动程序来完成，中断处理程序的优先级被设定为高于任何用户进程。但在实时系统中，中断的处理必须小心谨慎：首先不能打开所有的外部中断，因为过多的外部中断会带来不确定因素，无法保证任务的实时完成；对于重要的中断必须及时响应，所以中断屏蔽的时间越少越好。

1.2 RTOS 的发展阶段与趋势

RTOS 的研究是从二十世纪六十年代开始的，至今已有 40 多个年头了，RTOS 得到蓬勃的发展，在嵌入式领域内得到了广泛的应用。本节内容对 RTOS 的发展和现状作了简单回顾。

1.2.1 RTOS 的发展阶段

从系统结构上看，RTOS 到现在已经历了如下三个阶段：

(1) 早期的 RTOS

早期的实时操作系统还不能称为真正的 RTOS，它只是一个小而简单的、带有一定专用性的软件，功能较弱，可以认为是一种实时监控程序。它一般为用户提供对系统的初始化管理以及简单的实时时钟管理，有的实时监控程序也引入了任务调度及简单的任务间协调等功能，属于这类实时监控程序的有 RTMX 等。

(2) 专用实时操作系统

随着应用的发展，早期的 RTOS 已越来越显示出明显的不足了。有些实时系统的开发者为了满足实时应用的需要，自己研制与特定硬件相配的实时操作系统。这类专用实时操作系统在国外称为 Real-Time Operating System Developed in House，一般只能适用于特定的硬件环境，且缺乏严格的评测，移植性也不太好。属于这类实时操作系统的有 Intel 公司的 iMAX86 等。

(3) 通用实时操作系统

在各种专用 RTOS 中，一些多任务的机制如基于优先级的调度、实时时钟管理、任务间的通信、同步互斥机构等基本上是相同的，不同的只是面向各自的硬件环境与应用目标。在各种专用 RTOS 的基础上，将最低层不同的硬件特性屏蔽掉，为不同的应用环境提供标准的、可裁剪的系统服务组件，形成了通用实时操作系统。属于这类 RTOS 的有 Integrated System 公司的 pSOSystem、Intel 公司的 iRMX386、Wind River Systems 公司的 VxWorks、Microware 的 uC/OS 等。它们一般都提供了实时性较好的内核、多种任务通信机制、基于 TCP/IP 的网络组件、文件管理及 I/O 服务，提供了集编辑、编译、调试、仿真为一体的集成开发环境，支持用户使用 C、C++进行应用程序的开发。

1.2.2 RTOS 今后的发展方向

今后，RTOS 研究方向主要集中在如下几个方面：

(1) OS 的标准化研究

如今国外的 RTOS 开发商有数十家，提供了上百个 RTOS，它们各具特色。但这也给应用开发者带来难题，首先是应用代码的重用难，当选则不同的 RTOS 开发时，不能保护用户已有的软件投资，RTOS 的标准化研究越来越被重视。美国 IEEE 协会在 UNIX 的基础上，制定了实时 UNIX 系统的标准 POSIX1001.4 系列协议，但仍有许多工作还待完成。

(2) 多处理器结构 RTOS、分布式实时操作系统和实时网络的研究

实时应用的飞速发展，对 RTOS 的性能提出了更高的要求。单处理器的计算机系统已不能很好地满足某些复杂实时应用系统的需要，开发支持多处理器结构的 RTOS 厂家虽然已推出部分产品，如 QNX、Chorus、Plan9 等，但分布式实时操作系统的研究还未完全成熟，特别是在网络实时性和多处理器间任务调度算法上还需进一步研究。

(3) 集成的开放式实时系统开发环境的研究

RTOS 研究的另一个重要方向是集成开发环境的研究。开发实时应用系统，只有 RTOS 是不够的，需要集编辑、编译、调试、模拟仿真等功能为一体的集成开发环境的支持。开发环境的研究还包括网络上多主机间协作开发与调试应用技术的研究、RTOS 与环境的无缝连接技术等等。

1.3 常用 RTOS 简介

1.3.1 常用商品嵌入式 RTOS

(1) VxWorks

VxWorks 是美国 WindRiverSystem 公司 (WRS 公司或风河公司) 推出的一个实时操作系统。它以良好的可靠性和卓越的实时性被广泛地应用在通信、军事、航空、航天等高精尖技术及实时性要求极高的领域中，如卫星通讯、军事演习、弹道制导、飞机导航等。在美国的 F-16, FA-18 战斗机、B-2 隐形轰炸机和爱国者导弹上，火星探测器上都使用了 VxWorks。 VxWorks 是目前市场占有率最高的实时操作系统，被认为是最优秀的实时操作系统，但价格也是最昂贵的。

(2) pSOS

该操作系统属于 ISI 公司，不过 ISI 公司现已被 WinRiverSystem 公司兼并。这个系统是一个模块化、高性能的实时操作系统，专为嵌入式微处理器设计，提供一个完全多任务环境，开发者可以利用它来实现从简单的单个独立设备到复杂的、网络化的多处理器系统。

(3) Windows CE

Microsoft Windows CE 是一个简洁的，高效率的多平台操作系统，但不够实时，属于软实时操作系统。它的模块化设计允许它对从掌上电脑到专用的工业控制器等电子设备进行定制。无奈价格太高，使得整个产品的成本急剧上升。

(4) QNX

QNX 是一个实时的，可扩充的操作系统，它遵循 POSIX.1 (程序接口)和 POSIX.2 (Shell 和工具)、部分遵循 POSIX.1b(实时扩展)。它提供了一个很小的微内核以及一些可选的配合进程，因此 QNX 内核非常小巧(QNX4.x 大约为 12Kb)而且运行速度极快。这个灵活的结构可以使用户根据实际的需求将系统配置成微小的嵌入式操作系统或是包括几百个处理器的超级虚拟机操作系统。

(5) Palm OS

3Com 公司的 Palm OS 在 PDA 市场上占有很大的市场份额，它有开放的操作系统应用程序接口（API），开发商可以根据需要自行开发所需要的应用程序。目前已经有总共 3500 多个应用程序可以运行在 Palm Pilot 上，这些软件包括计算器、各种游戏、电子宠物、文本编辑等等。

1.3.2 公开源码的嵌入式 RTOS

(1) 嵌入式 Linux

嵌入式 Linux 是在 Linux 基础上做了简化，去掉了多用户及虚拟内存管理等，适用于嵌入式应用类 CPU。它由一个 Kernel（内核）及一些根据需要进行定制的系统模块组成。Kernel 一般有几百 KB 左右，它具有多任务、多进程的系统特征，有些还具有实时性。

源代码开放的 Linux 所取得的极大成功，使得嵌入式 Linux 作为一种可裁剪的软件平台系统，很可能发展成为未来嵌入式设备产品的绝佳资源。嵌入式 Linux 可应用于多种硬件平台，高度模块化的软件设计，使得添加部件非常容易，同时他本身内置网络支持，强大的语言编译器使得开发更加方便。[2]

(2) uC/OS-II

uC/OS-II 是一个高度可移植的、可裁剪、占先式、多任务的实时内核，它的绝大部分代码是 ANSIC 写成的。只有与 CPU 硬件相关的部分是用汇编语言写的。汇编语言部分被压缩到最低限度，为的是便于移植到任何一种其它的 CPU 上。现在，在航空电子设备的制造方面它已经被广泛的应用。自 1992 年 uC/OS 面世以来，该系统已经

被用于上百种产品。[3]

在应用于嵌入式系统方面，uC/OS-II 小得可以放在一张软盘上运行，程序代码可以小到 3KB，适当的裁剪也可以使数据空间变得更小。因此 uC/OS-II 的这些特性给嵌入式开发带来了很大的发展空间。

(3) eCos

eCos 全称是 Embedded Configurable Operating System。它诞生于 1997 年，可以说是嵌入式领域的一颗新星。其绝大多数代码使用 C++写作完成。eCos 最大的特点是模块化，内核可配置。它是一个针对 16 位、32 位和 64 位处理器的可移植开放源代码的嵌入式 RTOS。

eCos 具有相当丰富的特性和一个配置工具，后者能够让你选取你所需要的特性。eCos 最小版本只有几百个字节，一般，一个完整的网路应用，其二进制的代码也就 100K 字节左右。eCos 提供的 Linux 兼容的 API 能让开发人员轻松的将 linux 应用移植。[4]

1.4 本文工作与结构

本论文工作对象是一种基于 MC9S08GB60 单片机的嵌入式实时操作系统(RTOS)。MC9S08GB60 是摩托罗拉公司新近推出的一款性能优良强大、资源丰富、运行稳定并且价格低廉的 8 位 MCU。传统的前后台(超循环)设计方法大大限制了 MC9S08GB60 的性能的发挥的，尤其多任务运行时考虑的因素增多，同时任务之间的通讯困难，系统容易死锁，可靠性不高。RTOS 是一种新的系统设计思想和一个开放的软件框架。它具有操作系统的基本功能，可以对整个实时系统的运行进行控制，能根据系统中各个任务的轻重缓急，合理的在它们之间分配 CPU 和各种资源的占用时间。利用信号量、邮箱、消息队列等功能提高 CPU 的使用效率。由于 RTOS 可以将应用系统划分为多个任务，从而大大简化应用系统的设计，软件设计人员可以在不需大量改变系统其他任务的情况下增加或删除一个任务，大大缩短了开发时间和减少了软件的编写任务，适应单片机应用复杂化的要求。

全文内容结构安排如下：

第一章简要介绍了 RTOS 的定义、结构、发展和展望等相关知识。

第二章比较详细地介绍了最小系统和 MC9S08GB60,给出了 GB60 最小系统的构成：PCB 板、监控程序、在线编程系统。

第三章对 SDF-RTOS 软件系统设计中各个模块进行了详尽的分析描述,并给出了

测试程序，同时对多任务内核运行所占用的资源进行分析。

第四章通过对暖气表的软硬件设计，进一步测试 SDF-RTOS 的性能及可靠性。

第五章对所做的工作进行总结，指出课题进一步工作的展望。

第二章 最小系统

2.1 最小系统概述

最小系统是指能够使单片机正常运作的最小软件系统和最小硬件系统的总称。其中包括单片机及其复位电路、时钟电路、RS232 串口电路等几个部分。最小系统上的单片机内一般都要驻留用户监控程序，配合 PC 机高端相应的开发工具，通过串口实现单片机与 PC 机的通讯，可直接完成程序的下载和调试。最小系统是学习单片机原理、测试程序运行的最小载体，是单片机应用系统的设计基础。[5]

2.2 芯片选型

2.2.1 芯片选型的原则

在嵌入式产品设计中，硬件选型的好坏将直接影响着产品设计进度，同时也决定了产品的性能，还可能会影响到产品成形后的生产。因而硬件选型是嵌入式产品设计的一个重要环节。在硬件选型时应该综合考虑成本问题，开发的难易程序问题，元件购买途径问题，用户需求问题等等。以下将以主控芯片为例从技术角度来阐述一下对硬件的选型原则。由于在嵌入式的开发中，所有的设计都是围绕着主控芯片展开的，所以对它的选型问题就显得尤为重要，它应该遵循以下要原则：[6]

(1) 合理的 RAM 和 Flash 大小；

(2) 通用的 I/O 引脚数目；

(3) 内部包含的功能模块；

(4) 芯片的封装形式；

(5) 写入器，编译器和集成开发环境。

由于最终的用户只要求实现其功能而对使用哪一款芯片一般无要求，因此在设计过程中尽量选用自己比较熟悉的芯片开发，这样会节省开发的周期。RAM 和 Flash 的大小与 I/O 引脚的数目以及芯片的价钱基本上是成正比关系的，价钱越贵的芯片前两者也就越大。所以应该选择一个性价比相对比较合适的芯片才不至于浪费芯片资源，增加产品的成本。芯片的内部功能模块应结合用户的开发需求来定，尽量使所选择的芯片能满足大部分的用户要求，这样才能物尽所用。芯片的封装形式是出于对实验和后期生产的考虑。通常的封装形式有双列直插型和贴片形式，双列直插封装实验起来比较方便，而贴片形式相对比较麻烦，但是其体积小，节约空间，对于空间比较紧张的产品则必须选用这种形式。牵涉到芯片就必须要考虑到写入器，编译器以及集成开

发环境的问题。通常一个通用的写入器要几千到上万元，一个带编译器的集成开发环境也要几万元，这对于一般的实验性小项目是不适合的，只有进行自主开发，一次性开发好了，以后做类似的课题就可以进行移植，不必从头再来了。

根据上述的原则，结合苏州大学摩托罗拉单片机实验室所拥有的单片机性能，再综合考虑现有的技术支撑，我们选用 MC9S08GB60（以下简称 GB60）作为今后开发的主控芯片。

2.2.2 MC9S08GB60 功能特征

MC9S08GB60 是摩托罗拉（现为飞思卡尔）公司于 2003 年推出的 HCS08 系列 8 位微控制器四个产品型号之一，它是 HC08 系列的扩展。MC9S08GB60 芯片主要有以下特征：[7]

(1) 60Kb 的 Flash 存储器，具有密码保护和在线编成能力；

(2) 4KB 片内 RAM；

(3) 2 个串行通讯 SCI 口，1 个串行异步通信 SPI 口；

(4) 56 个通用 I/O；

(5) 8 通道的 TIMER/PWM；

(6) 8 个键盘中断口；

(7) 8 个模数转换口；

(8) 背景调试模式；

(9) 100Kbps 的 I^2C 总线；

(10) COP,IPQ,TRI 功能模块。

GB60 的功能结构框图参见附录 A.1,其主要的功能模块介绍如下：[8]

(1) CPU HCS08 核：GB60 的处理器使用了增强型的 HCS08 核，它虽然还是 8 位的处理器，但是比原来的 HC08 核已有了大大的增强，最高总线频率可达 40M，即最小的指令执行时间可达 25ns；最长的操作指令 RTI 需要 11 个指令周期，也只要 275ns。同时它增加了更多的 16 位操作指令，使得对于 16 位寄存器 HX 操作更加灵活方便。

(2) 存储器：GB60 寻址空间为 64K，存储器的映像图见附录 A.2，其中包括：61268 字节的 Flash、4096 字节的 RAM、127 字节的直接页寄存器和 44 字节的高页寄存器。

其中 Flash 区分为两块：一块大小为 1920 字节，通常我们在其中存放驻留在系统中的一些程序，比如监控等；另一块大小为 59348 字节，这一块用来存放用户程序。

(3) 串行通信模块：GB60 芯片中有两个串行通信模块，它们二者所用的寄存器各不相同，相互独立工作，互不影响。

(4) 定时器和脉宽调制模块：GB60 中有两个定时器和脉宽调制模块，它们和 PTD 口复用，PTD0～PTD2 对应模块 1 中的三个通道，PTD3～PT7 对应模块 2 中的 5 个通道，两个通道独立工作互不影响。

(5) 模数转换模块：GB60 中提供了 8 路的模数转换通道，它与 PTB 口复用。8 路通道互相独立工作。

(6) 背景调试模块：这一模块是新推出的增强型芯片中新增加的功能，它提供的背景调试方式数据通信速度快，调试方便，使得系统中实现写入和调试功能更加简单。

2.3 MC9S08GB60 最小系统硬件设计

以 MC9S08GB60 芯片（引脚图见附录 A.3）为核心的最小系统主要包括以下几个部分：时钟电路、串口电路、供电电路、 MC9S08GB60 引脚图、复位电路、调试小灯。其原理图如图 2－1 所示。各部分的功能如下：

(1) 时钟电路给单片机提供一个外接的 4MHz 的石英晶振；

(2) 串口的 RS-232 驱动电路可实现 TTL 电平到 RS-232 电平的电平转换；

(3) 供电电路主要是给单片机提供＋3.3V 电源；

(4) 复位电路通过按钮开关给单片机提供一个复位信号；

(5) 调试用小灯和单片机的 PTA 口相连，供调试程序用。

2.3.1 时钟电路

GB60 是一款增强型 8 位 MCU，系统最高总线频率可达 20M，性能上与 DSP 相当。单片机提供了自身的内部晶振，但由于该晶振源不太稳定，因此系统内部提供了一字节的晶振调整因子，可以根据需要将内部总线频率调整到一个更加精确的数值。

GB60 的最小系统提供了外部晶振及其辅助电路，再由内部锁相环倍频，将系统总线频率提高，并且获得的是一个精确的数值。该方法得到的总线频率更加可靠、稳定。因此在单片机正常工作时，选择了外部晶振源。

从原理图上看，最小系统电路非常简单。但由于时钟电路中的振荡器一方面为单片机系统提供时钟源，同时它又是一个很强的高频干扰源，故在 PCB 布局时，必须考

虑周全，否则引入很强的干扰，将严重影响系统的稳定可靠性。单片机系统时钟电路和电源电路 PCB 板布线应如下的规则进行，系统的电磁兼容性才能得到保证。这些规则如下：

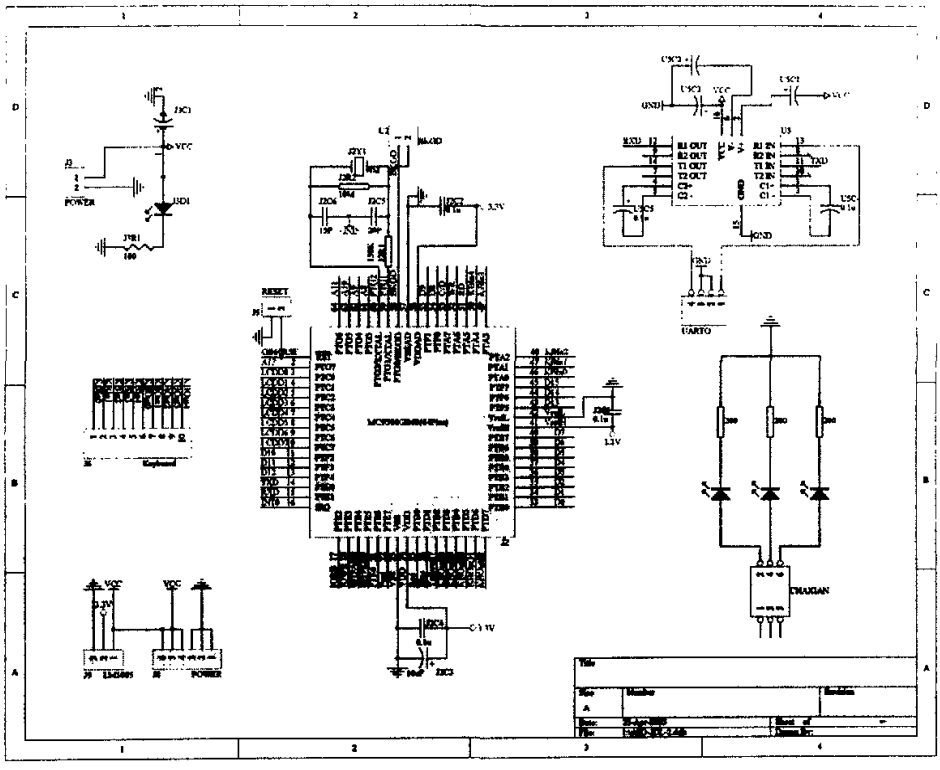


图 2—1 MC9S08GB60 最小系统原理图

(1) 晶振引脚必须紧靠单片机的 EXTAL 和 XTAL 脚，尽量缩短它们直接的距离，以减少线路上产生的耦合电容；

(2) 尽量让时钟信号回路周围电场趋近于零，用地线将时钟区圈起来；

(3) 石英晶体振荡器外壳要接地；

(4) 锁相环的滤波电路要尽量靠近 MCU；

(5) 石英晶振下面和对噪声特别敏感的器件下面不要走线；

(6) 单片机的每个电源端和接地端都要接一个去耦电容，去耦电容要尽量靠近 MCU。石英晶体振荡器外壳要接地。[9]

2.3.2 串行口驱动电路

通过串口驱动电路中的 RS-232 电平转换芯片，单片机可以利用异步串行通信协议同 PC 通信。我们选用 MAX 公司的 5V 逻辑电平转换芯片 MAX232 来实现 TTL 电平与

RS-232 电平之间的转换。MC9S08GB60 单片机有两个串口通信模块，我们选定 SCI1 做为串口通信，使用 PTE0 作为 TXD（数据发送），PTE1 作为 RXD（数据接收）。

2.3.3 电源电路

MC9S08GB60 的供电电压仅需要 3.3V，我们选用了 1085 芯片将 5V 电压转换为 3.3V，并保持稳定。

2.4 写入器及在线编程系统

2.4.1 MC9S08GB60 写入器

MC9S08GB60 作为摩托罗拉（现为 Freescale）公司新推出的 HCS08 系列 8 位 MCU 重要一员，具有背景调试控制器(BDC)和在线调试模块(DBG)。使用者可以通过背景调试控制器提供的单线接口来对 MCU 的 FLASH 和 RAM 执行读写操作，进而可以完成程序化作业，并利用芯片内部仿真器对用户程序进行测试或调试。同时，背景调试控制器也是单片机主要的开发接口，它提供了内存数据的访问方法和传统调试，比如 CPU 寄存器数值的修改、断点、单步指令执行。

HCS08 系列单片机中，地址和数据总线信息无法通过外部引脚获得，即使在测试模式下，也不可完成。调试工作，是通过单片机提供的单线接口发送指令到 CUP 来实现的。因此调试模块可以通过该接口，有选择地触发和跟踪总线信息，随时了解单片机内部的工作情况。背景调试控制器(BDC)的功能如下：

(1) 单独的模式选择和背景通信的测试引脚；

(2) BDC 寄存器不存在于内存区；

(3) SYNC 指令决定目标板的通信频率；

(4) 非干扰式指令访问内存；

(5) 积极背景模式指令访问 CPU 寄存器；

(6) “GO”和“TRACE1”指令；

(7) “BACKGROUND”指令将 CPU 从停止和等待模式下唤醒；

(8) BDC 中有一个硬件断点地址；

(9) BDC 允许的情况下，振荡器可以在停止模式下运行；

(10) 积极背景模式下不允许看门狗 COP。

调试模块(DBG)的功能如下：

(1) 两个触发比较器；

(2) 灵活的 8-word 16 位先进先出缓冲；

(3) 两种类型断点；

(4) 9 种触发方式。

背景调试控制器对外提供了一个 BKGD 引脚，该引脚主要完成目标版上单片机的写入，见图 2－2。最小系统中，由 M68HC908GP32 与引脚连接，通过 BKGD 对 GB60 写入独立编写的监控程序，并通过串行口和高端的在线编程系统验证了监控程序的工作情况。

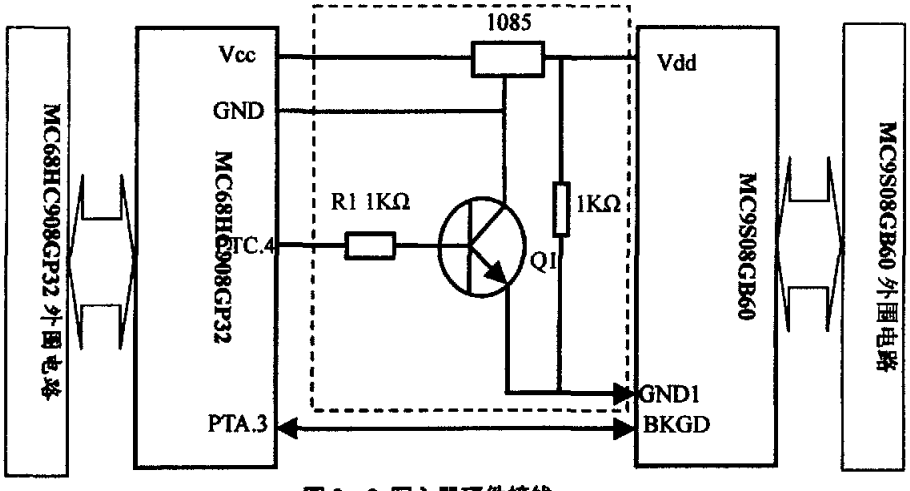


图 2－2 写入器硬件接线

向刚出厂的 GB60 写入监控程序需要解决如下若干问题：

(1) 进入 GB60 的背景调试工作方式；

(2) 通过 GB60 的时序，完成 GP32 与其的通信，包括读写 RAM 中单个字节和寄存器；

(3) 写 RAM 中多个字节（对应于一个独立的简单程序）；

(4) 通过指令“GO”，执行上述程序；

(5) 编写 FLASH 操作子程序，并给出完整的监控程序；

(6) 将监控程序写入 FLASH，并设定复位向量地址。

芯片内部有了用户独立定义的监控程序，便可以结合独立定义的在线编程系统，完成用户程序的下载，实现在线编程。该过程不仅省去了高价的开发工具，更提供了一个学习与锻炼的机会。

2.4.2 MC9S08GB60 在线编程

SD-HC08 在线编程开发系统下的 C 语言(简称 08C 语言)编程提供了 C 编译器、宏汇编、连接器、库管理和工程项目管理以及程序下载等功能，同时 08C 语言提供了在 C 语言中嵌入汇编指令和调用汇编指令构成的子程序，以及在汇编中调用 C 语言中的变量和子程序的混合编程的方法，这就给我们基于 MC9S08GB60 的 SDF-RTOS 的开发搭建了很好的平台。一个完整的 08C 语言程序通常至少包含如下 5 个部分：

- 汇编语言的启动文件：建立 C 语言程序的运行环境、初始化堆栈指针，对变量进行初始化；
- 存储器映像及 I/O 端口的头文件：每一个具体的 MCU,需要写一个有关 I/O 端口及寄存器定义的头问及文件；
- 主程序模块：主程序中，首先初始化数据端口及数据传送方向，初始化相关的模块，然后进入主循环；
- 中断服务子程序；
- 中断向量表：中断服务子程序的地址。

(1)新建文件

在 PC 机的 SD-HC08 开发环境中可以很方便地建立一个 08C 语言工程。一个 08C 语言工程一般由.prj、.src、.h、.c 等类型文件构成。其中.prj 和.src 文件是 08C 的工程文件和资源文件，用来记录工程中的所有源文件信息、目标代码的起始地址、目标系统的硬件信息等，而.c 和.h 文件分别是 08C 的源程序文件和头文件，源程序文件由若干个函数组成，一个 08C 语言工程必须包括一个 main 函数(主函数)。

(2)源文件编辑

在开发环境中可以很方便地对.c 和.h 文件进行编写、修改及保存等编辑操作。可以进行撤销、重复前面的编辑，查找、替换操作。

(3)源文件编译

打开或编写好了源文件后，在开发环境中进行编译工作，把一个工程下的所有文件编译成目标文件。如果源文件中有错，编译出错后，在源文件中会自动选中出错行，提示进行纠错。如果源文件编译成功后，系统会产生.o、.s、.lst、.s19、.mp 等文件。

(4)目标代码写入芯片

要将目标代码写入芯片，先要对目标代码 S19 文件进行分解，分离出程序数据区和复位矢量数据区，同时将程序数据区分割为 512 字节的页，并记录该页的首址。然

后建立 PC 机和 MCU 的通信线路。在建立通信线路前将 MCU 复位，MCU 会发送握手信号，同时等待 PC 机的返回信号。所以当 PC 机发送指定的返回信号后，PC 机和实验板的通信线路就建立了，此时 MCU 就处于等待接收 PC 机的下一个命令字状态。PC 机再发送写入芯片的命令，MCU 接收命令后，等待接收写入的总页数，然后接收每一页的首址及要写入的数据，MCU 先擦除再将数据写入，写完一页后将该页的数据读出送 PC 机，PC 机接收该页数据后校验是否有错，若无错，接着发送下一页，MCU 接收下一页的首址及要写入的数据，直到写完所有的页以及最后的复位矢量数据区的数据页。

2.5 最小系统集成与测试

PCB 板、监控程序、在线编程系统，三者的结合便是 GB60 的最小系统。PCB 板提供了最小系统的硬件平台；监控程序与在线编程系统之间给出双方预先定义的协议，实现串行口通信，其中包括数据通信和握手通信。因此，通过最小系统写入用户程序的流程为：(1)在线编程系统中编写用户程序；(2)对用户程序进行编辑、编译，生成 S19 文件；(3)复位单片机，建立单片机与在线编程系统之间的连接；(4)单片机在建立连接之后，首先将写 FLASH 和串行口通信程序放入 RAM，然后通过串行口接收 PC 机方的数据（包括中断向量），将其写入 FLASH；(5)跳转到用户程序首地址，执行用户程序。该过程之后，用户程序便驻留在 FLASH 区，单片机复位后，都将自动转入用户程序执行。

最小系统在 PC 端提供了一个用户程序写入接口，简化用户对 FLASH 的操作，完成 GB60 的在线编程。

现以一个实例来测试用户程序的执行情况。

```
//=========================================================
//函数名：InitTimer1
//功能：定时器 1 初始化成 1s 一次中断
//入口：无
//出口：无
//=========================================================
void InitTimer1(void)    /*定时器 1 初始化程序  每 1s 产生一次中断*/
{
  TPM1SC = 0b01001111;  /*允许定时器中断，暂不计数，分频系数为 128*/
```

```
    TPM1MOD=0x7A12;                    /*设 1 秒产生 1 次溢出中断*/
    TPM1CNT=0;                         /*计数寄存器初值=$0000*/
    asm("cli");                        /*放开中断*/
}
//=====================================================
//说明：主程序 main，包括 GB60 相关模块的初始化和主循环。
//入口：无
//出口：无
//=====================================================
void main(void)
{
    asm("sta $1800");                  /*看门狗喂食*/
                                       /*相应寄存器或变量的初始化*/
    PTADD|=0b00000001;                 /*定义 A 口第 0 位为输出 */
    PTAD|=0b00000001;                  /*初始定义 PTA.0 输出高电平*/
   InitTimer1();                       /*定时器初始化*/
    while(1)                           /*主循环*/
    asm("sta $1800");                  /*看门狗喂食*/
}
//=====================================================
//函数名：isrTimer1OV
//功能：定时器 1 中断程序，1 秒钟一次中断后的处理
//=====================================================
#pragma interrupt_handler isrTimer1OV
void isrTimer1OV(void)
{
    asm("sei");                        /*禁止中断*/
    asm("sta $1800");                  /*看门狗喂食*/
    TPM1SC&=127;                       /*清定时器溢出标志*/
    PTAD=0b00000001&(~PTAD);           /*PTA.0 位取反*/
```

```
    Send1ByteToPC(PTAD);        /*通过串行口发送小灯状态至 PC 机*/
    asm("sta $1800");                          /*看门狗喂食*/
    asm("cli");                                /*放开中断*/
}
```

以上三部分程序段组合在一起，构成一个完整的 GB60 最小系统板上的测试程序，程序的功能是：PTA.0 脚连接的小灯每隔一秒就会改变一次亮/暗状态；同时通过串行口将小灯状态发送至 PC 机。该程序编译后生成的 S19 文件，可以通过在线编程系统写入 GB60 指定的 FLASH 区，并直接运行。程序运行的结果与程序设定的功能完全吻合，说明程序运行正常。此外，在该程序的基础上，加入了若干无效语句，旨在增加 S19 的量；通过在线编程系统将 S19 文件写入 FLASH，运行的结果显示程序工作正常。再结合平时的用户程序，特别是移植的操作系统的正常工作，能够说明，最小系统及其在线编程系统的工作是完全正确的，也是完全可靠的。

第三章 SDF-RTOS软件系统设计

通常对于单片机的开发，一般都采用前后台(超循环)软件设计的方法。后台程序是一个无限循环，循环依次调用相应的函数来完成相应的操作。前台是中断服务程序，负责处理异步事件。前后台设计方法一般适合于一个单片机所控制的外设和执行的任务不多，并且实时性要求不高时的情况。如果所要求的应用比较复杂，控制的任务超过一定数量，或者要求对中断实时处理，它就无能为力了。这种限制使得HCS08系列单片机的强大功能无法完全发挥。在这种情况下，我们考虑使用嵌入式RTOS(实时操作系统)的软件设计方法。

SDF-RTOS是苏州大学摩托罗拉实验室基于HCS08系列单片机开发的一种小型的实时操作系统。它以MC9S08GB60最小系统为硬件开发平台，紧密结合HCS08系列单片机硬件特征，在满足系统功能的前提下尽可能简单，避免较大的系统开销。SDF-RTOS的总体功能如下：

(1)能同时调度最多10个实时任务；

(2)采用优先级抢占式调度方法，保证系统的实时响应时间；

(3)系统占用资源ROM控制10KB在内，RAM控制在512-1024B内；

(4)使中断延迟和调度延迟达到最小。

根据SDF-RTOS的总体功能，结合MCU运行RTOS的实际情况，主要设计了多任务内核、任务管理、时间管理、任务间通讯、中断管理等几个功能模块。在随后的功能模块设计描述中，我们对一些复杂的函数，通过示意性代码作详细介绍；而对一些简单的函数，只是说明一下功能和执行过程。由于RTOS的设计理论很成熟，我们在设计系统时借鉴了大量开放源代码的RTOS的设计思想和代码，如Small RTOS, uC/OS-II等，其中以u C/OS-II为主。[3][10]

3.1 内核结构

3.1.1 临界区的处理

代码的临界区是指处理时不可分割的代码。一旦这些代码开始执行，则不允许任何中断打断。例如在中断开始时，需要保存所有寄存器，此时就不允许中断程序被打断。

处理临界区(critical section)代码最常用的方法就是关中断，处理完毕后再开中断。这是为了能够避免同时有其他任务或中断服务进入临界区代码。关中断的时间

影响用户系统对实时事件的响应速度，所以必须减少到最小。

MC9S08GB60 的指令系统中有关中断和开中断指令，同时我们使用的由苏州大学 Motorola 单片机实验室开发的 SDIDE 在线编成系统支持混合编程，在 C 语言下能够很方便地嵌入汇编语言。这使得控制临界区的开关很容易实现。关中断/开中断的命令为：

关中断 SEI；

开中断 CLI；

3.1.2 任务定义

一个任务(task)，也称作一个线程，是一个简单的程序，是系统的基本组成元素，系统运行时的独立单元。每个任务都是整个应用的某一部分，每个任务被赋予一定的优先级，有它自己的一套 CPU 寄存器和自己的堆栈空间，在运行时认为 CPU 完全只属于该任务自己所用。

在具体设计时，任务是一个无限的循环，运行时根据具体情况在不同任务状态之间来回切换。任务在形式上看起来像其他 C 的函数一样，有函数返回类型，有形式参数变量，但是任务是绝不会返回的。如 LCD 显示任务的任务定义为：

void Task_ LCDDisplay (void *data);

根据应用系统的具体情况，系统中的任务数也不相同，但不能大于最多任务数。应用程序中的最多任务数(OS-MAX-TASKS)是在文件 OS_ CFG.H 中定义的。我们所设计的 RTOS 最多可以管理 12 个任务(主要根据 RAM 的使用情况)。

为了使内核能管理用户任务，用户必须在建立一个任务的时候，将任务的起始地址与其他参数一起传给函数 OSTaskCreate()。由该函数将任务提交给内核进行管理。

3.1.3 任务状态

图 3-1 是任务状态转换图。在任一时刻，任务的状态事实上是以下五种状态之一：休眠态、就绪态、运行态、挂起态、中断态。

• 休眠态：指任务驻留在程序空间之中，还没有交给 RTOS 内核调度管理。

• 就绪态：表明任务已经准备好，可以运行。但由于该任务的优先级低于目前运行任务的优先级，还暂时不能运行。

• 运行态：指该任务已经获得了 CPU 的控制权，正在运行中。

• 挂起态：也称为等待状态。指该任务正在等待某一事件的发生。如等待某外设

的I/O操作，等待某共享资源由暂时不可用变为可以使用等等。

• 中断态：正在运行的任务被中断，进入中断服务子程序。

当多任务内核决定运行另外的任务时，它保存正在运行任务的当前状态，即CPU所有寄存器的内容。这些内容保存在任务的当前状态保存区，也就是任务自己的任务堆栈或系统中断堆栈(在中断服务子程序中发生任务切换)之中。入栈工作完成以后，把下一个将要运行的任务的当前状态从该任务的状态保存区中重新装入CPU的寄存器，并开始下一个任务的运行。

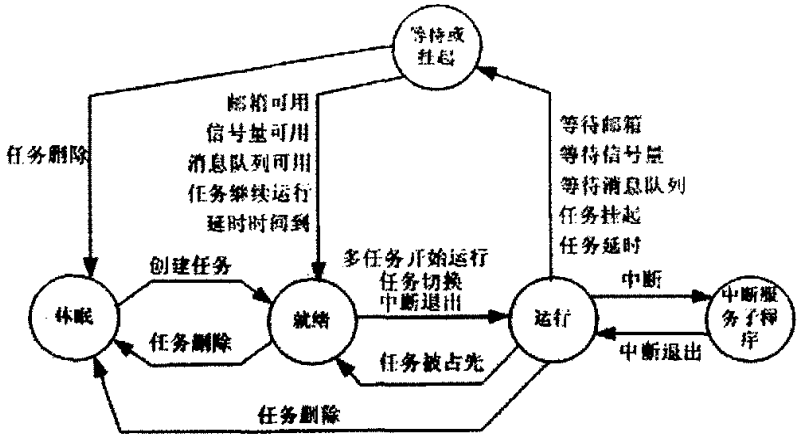


图 3-1 系统中任务的状态

当任务一旦建立，这个任务就进入就绪态准备运行。任务的建立可以是在多任务运行开始之前，也可以动态地被一个运行着的任务建立。一个任务可以返回到休眠态，或让另一个任务进入休眠态。只有当所有优先级更高的任务转为等待状态，或者是被删除了，就绪的任务才能进入运行态。

正在运行的任务可以通过函数OSTimeDly ()(见2.3.1节)将自身延迟一段时间。于是该任务进入挂起状态，等待这段时间过去，下一个优先级最高的、并进入了就绪态的任务立刻被赋予了CPU的控制权。等待的时间过去以后，系统服务函数OSTimeTick()(见2.1.8节)使被延迟的任务进入就绪态。

正在运行的任务期待某一事件的发生时也要挂起，手段是调用以下两个函数之一OSSemPend () , OSMboxPend ()(见2.4节)，调用后任务进入了挂起状态。当任务因等待事件被挂起(Pend)，下一个优先级最高的任务立即得到了CPU的控制权。当事件发生后，被挂起的任务进入就绪态。事件发生的报告可能来自另一个任务，也可能来自中断服务子程序。

在挂起的任务中，系统会根据任务的挂起原因(延时还是等待事件发生)，对被挂起的任务进行处理。一旦被挂起任务的条件满足，系统会将其放入就绪队列中，由调度程序进行调度运行。

如果中断没有被关闭，正在运行的任务时可以被中断的。被中断了的任务进人中断服务子程序。响应中断时，正在执行的任务被挂起，中断服务子程序控制了 CPU 的使用权。中断服务子程序可能会报告一个或多个事件的发生，而使一个或多个任务进入就绪态。在这种情况下，从中断服务予程序返回之前， 系统要判定，被中断的任务是否还是就绪态任务中优先级最高的任务。如果中断服务于程序使一个优先级更高的任务进入了就绪态，则新进入就绪态的这个优先级更高的任务将得以运行，否则原来被中断了的任务还会继续运行。

当所有的任务都处于等待事件发生或等待延迟时间结束的状态时，系统执行空闲任务。

3.1.4 任务控制块(OS_ TCB)

任务控制块是一个数据结构，用来保存任务的所有状态。当任务的 CPU 使用权被剥夺时，系统使用它来保存该任务的所有状态。当任务重新获得 CPU 使用权时，将任务控制块中保存的任务状态恢复，能确保任务从被切换的那一点丝毫不差的执行下去。任务建立时，对该任务的 OS_ TCB 进行初始化。OS_ TCB 全部驻留在 RAM 中。下面对 OS_ TCB 中比较重要的变量进行介绍：

.OSTCBStkPtr 是指向当前任务栈顶的指针。系统允许每个任务有自己的栈，OSTCBStkPtr 是 OS_TCB 数据结构中唯一的一个能用汇编语言来处置的变量，为了使从汇编语言中处理这个变量时较为容易，把 OSTCBStkPtr 放在数据结构的最前面（在偏移量为 0 的位置）。

.OSTCBStkBottom 是指向任务栈底的指针。MC9S08GB60 单片机的栈指针是递减的，即在存储器中堆栈从高地址向低地址方向分配，所以 OSTCBStkBottom 指向任务使用的堆栈空间的最低地址。

.OSTCBStkSize 任务堆栈的大小。其单位是指针元数目而不是字节。也就是说，如果堆栈中可以保存 10 个入口地址，每个地址宽度是 16 位的，则实际栈容量是 20 个字节。

.OSTCBNext 和.OSTCBPrev 用于任务控制块 OS_TCBs 的双重链接，任务控制块在运行时被组织成双向链表，该链表在时钟节拍函数 OSTimeTick()中使用，用于刷新各

个任务的任务延迟变量 OSTCBDly，每个任务的任务控制块 OS_TCB 在任务建立的时候被链接到链表中，在任务删除的时候从链表中被删除。

.OSTCBEventPtr 是指向事件控制块的指针，在任务间通讯一节中讨论。

.OSTCBMsg 是指向传给任务的消息的指针，在任务间通讯一节中讨论。

.OSTCBDly 当需要把任务延时若干时钟节拍时要用到这个变量，或者需要把任务挂起一段时间以等待某事件的发生，这种等待是有超时限制的。在这种情况下，这个变量保存的是任务允许等待事件发生的最多时钟节拍数。如果这个变量为 0，表示任务不延时，或者表示等待事件发生的时间没有限制。

.OSTCBStat 是任务的状态字。当.OSTCBStat 为 0，任务进入就绪态。为 1 时，表明任务在等待信号量事件发生。为 2 时，表明任务在等待邮箱事件发生。为 8 时，表明任务处于挂起状态。

.OSTCBPrio 是任务优先级。这个值越小，任务的优先级越高。

.OSTCBDelReq 是一个布尔量，用于表示该任务是否需要删除。

系统分配给应用程序的任务控制块 OS_TCBs 的最大数目与应用程序中的最任务数(在文件 OS_ CFG. H 中定义)相同。将 OS-MAX-TASKS 的数目设置为用户应用程序实际需要的任务数可以减小 RAM 的需求量。所有的任务控制块 OS_TCBs 都是放在任务控制块列表数组 OSTCBTbl[]中的。在系统初始化的时候，所有任务控制块 OS_TCBs 被链接成空任务控制块的单向链表。当任务一旦建立，

空任务控制块指针 OSTCBFreeList 指向的任务控制块便赋给了该任务，然后 OSTCBFreeList 的值调整为指向链表中下一个空的任务控制块。一旦任务被删除，它的任务控制块就会回到空任务控制块链表中。

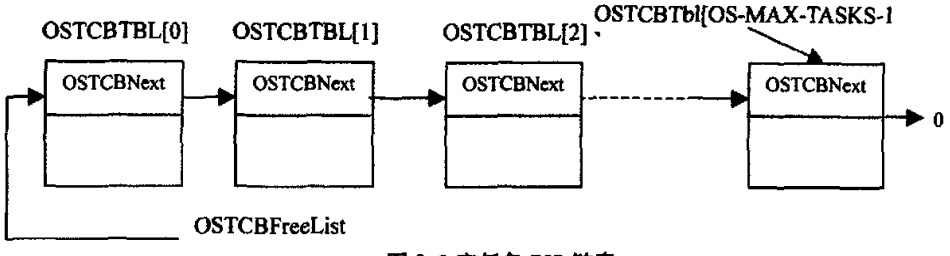


图 3-2 空任务 TCB 链表

3.1.5 任务优先级

任务优先级是应用程序按照该任务的重要性，给任务分配的一个系统中唯一的标志。在任务调度时将按照该任务的优先级进行调度。即按照该任务的重要程度调度任

务运行。

每个任务被赋予不同的优先级等级，从 0 级到最低优先级 OS_ LOWEST_ PRIO。当系统初始化的时候，最低优先级 OS_ LOWEST_ PRIO 总是被赋给空闲任务。最大任务数可以与最低优先级数不同。

3.1.6 就绪表

系统总是运行就绪状态中优先级最高的任务。每个任务的就绪态标志都放入就绪表(ready list)中，就绪表中有两个变量 OSRdyGrp(一维数组)和 OSRdyTbl[] (二维数组)。在 OSRdyGrp 中，任务按优先级分组，8 个任务为一组。OSRdyGrp 中的每一位表示 2 组任务中每一组中是否有进入就绪态的任务。任务进入就绪态时，就绪表 OSRdyTbl[]中的相应元素的相应位也置位。就绪表 OSRdyTbl[]数组的大小取决于常量 OS_ LOWEST_ PRIO。当用户的应用程序中任务数目比较少时，减少 OS_ LOWEST_ PRIO 的值可以降低系统对 RAM 空间的需求量。

OSRdyGrp 和 OSRdyTbl[]之间的关系见图 3－3，是按以下规则给出的：

当 OSRdyTbl [0]中的任何一位是 1 时，OSRdyGrp 的第 0 位置 1。

当 OSRdyTbl [1]中的任何一位是 1 时，OSRdyGrp 的第 1 位置 1。

通过以下代码将任务放入就绪表中，prio 是任务的优先级。

```
OSRdyGrp              |= OSMapTbl[prio >> 3];
OSRdyTbl[prio >> 3]   |= OSMapTbl[prio & 0x07];
```

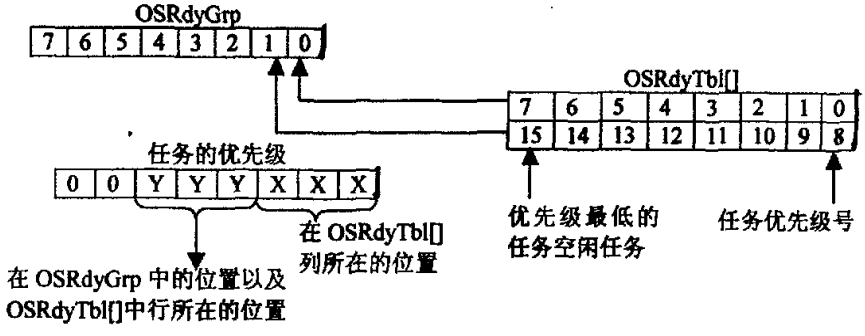


图 3-3　就绪表示意

可以看出，任务优先级的低三位用于确定任务在总就绪表 OSRdyTbl[]中的所在位。接下去的三位用于确定是在 OSRdyTbl[]数组的第几个元素。OSMapTbl[]是在 ROM 中的位掩码，用于限制 OSRdyTbl[]数组的元素下标在 0 到 1 之间。如果一个任务被删除了，则对以上算法中的代码做取反处理。

```
if ((OSRdyTbl[prio >> 3] &= ~OSMapTbl[prio & 0x07]) == 0)
                    OSRdyGrp &= ~OSMapTbl[prio >> 3];
```

以上代码将就绪任务表数组 OSRdyTbl[]中相应元素的相应位清零，而对于OSRdyGrp，只有当被删除任务所在任务组中全组任务一个都没有进入就绪态时，才将相应位清零。为了找到那个进入就绪态的优先级最高的任务，并不需要从OSRdyTbl[0]开始扫描整个就绪任务表，只需要查另外一张表，即优先级判定表OSUnMapTbl([256])。OSRdyTbl[]中每个字节的8位代表这一组的8个任务哪些进入就绪态了，低位的优先级高于高位。利用这个字节为下标来查OSUnMapTbl这张表，返回的字节就是该组任务中就绪态任务优先级最高的那个任务所在的位置。这个返回值在0到7之间。确定进入就绪态的优先级最高的任务是用以下代码完成的：

```
Y=OSUnMapTbl[OSRdyGrp];
X=OSUnMapTbl [OSRdyTbl [y];
Prio=(Y<<3)+X;
```

3.1.7 任务调度与切换

由于CPU资源是唯一的，所以同时只能有一个任务为运行态。任务调度机制将确定哪一个任务获得CPU资源。RTOS的实时性和多任务能力在很大程度上取决于它的任务调度机制。为了实现良好的实时性，我们选择可抢占式优先级调度策略。系统总是运行进入就绪态任务中优先级最高的那一个任务。如果该任务在运行中，由于某些原因激发了另一个优先级比它更高的任务，那么该任务将退出运行，保存它的全部运行状态，将CPU的控制权交给优先级更高的任务。即一个高优先级的就绪态任务可随时抢占当前正在运行的较低优先级的任务的CPU控制权。

(1)任务调度

确定哪个任务优先级最高，下面该哪个任务运行了的工作是由调度器完成的。任务级的调度是由函数OSSched()完成的。中断级的调度是由另一个函数OSIntExt()(见3.1.8节)完成的。系统任务调度所花的时间是常数，与应用程序中建立的任务数无关。OSSched()的所有代码都属临界区代码。在寻找进入就绪态的优先级最高的任务过程中，为防止中断服务子程序把一个或几个任务的就绪位置位，中断是被关掉的。

OSSched()示意性代码如下：

```
{ 关中断，进入临界区;
```

```
if ((OSLockNesting==0)&&(OSIntNesting==0))    /*中断服务子程序
                                              可调用且调度器没有上锁*/
找出就绪态优先级最高的任务;
if (OSPrioHighRdy != OSPrioCur )              /*优先级最高的任务应不是当
                                              前正在运行的任务*/
统计计数器 OSCtxSwCtr 加 1;                   /*跟踪任务切换次数*/
调用 OS_TASK_SW() ;                           /*实现任务切换*/
开中断，退出临界区;
}
```

如果不是在中断服务子程序中调用 OSSched()，并且任务调度是允许的，即没有上锁，则任务调度函数将找出进入就绪态且优先级最高的任务，进入就绪态的任务在就绪任务表中有相应的位置位。一旦找到优先级最高的任务， OSSched()就检验这个任务是不是当前正在运行的任务，以此来避免不必要的仔务调度。为实现任务切换，OSyCBHighRdy 必须指向优先级最高的那个任务控制块。

任务切换宏 OS_TASK_SW()定义为 MC9S08GB60 中的软中断指令 SWI。

(2)任务切换

任务切换分为两步：先将被挂起的任务的 CPU 寄存器推入堆栈；然后将较高优先级的任务的寄存器值从栈中恢复到寄存器中。系统中就绪任务的栈结构与中断发生后的栈结构很相似。为了做任务切换，运行 OS_TASK_SW(),人为模仿了一次中断，调用任务级切换函数 OSCtxSw()。实现任务切换。因此必须提供中断向量给任务级切换函数 OSCtxSw()。在 MC9S08GB60 中执行软中断指令 SWI 完成如下操作：PCL、PCH、X、A、CCR 依次入栈，中断向量表$FFFC～$FFFD 处地址装入 PC。

为缩短切换时间，OSCtxSw() 代码用汇编语言写。OSCtxSw()示意性代码如下：

```
{
 保存全部 CPU 寄存器;                    /*变址寄存器 X 的高位入当前任务堆栈*/
 将当前任务的堆栈指针保存到当前任务的 OS_TCB 中:
 OSTCBCur->OSTCBStkPtr = Stack pointer;
 调用用户定义的 OSTaskSwHook();
 OSTCBCur  = OSTCBHighRdy;     /*当前任务的指针指向要恢复运行的任务*/
 OSPrioCur = OSPrioHighRdy;    /*要恢复运行任务的当前任务的复制给当前任务
```

```
                                                    的当前任务的*/
  得到需要恢复的任务的堆栈指针:
  堆栈指针 = OSTCBHighRdy->OSTCBStkPtr;
  将所有处理器寄存器从新任务的堆栈中恢复出来;
  执行中断返回指令;
}
```

3.1.8 中断处理

中断服务子程序要用汇编语言来写，我们使用的SDIDE在线编成系统支持线汇编语言，可以将中断服务子程序代码放在C语言的程序文件中。

(1)设计符合RTOS设计要求的中断堆栈结构

在对内核结构的设计中，我们把任务切换分为任务级切换和中断级切换。其中任务级切换在3. 1.7节中已经介绍了。

MC9S08GB60单片机在发生中断时硬件已经默认执行了如下操作：

1、将中断发生时要执行的下一条指令的PCL、PCH值压入堆栈保存。将SR(状态寄存器)压入堆栈中保存。

2、变址寄存器X（低位）、累加器A、条件码寄存器CCR依次压入堆栈保存。

3、设置条件码寄存器CCR中的中断屏蔽位（I=1），禁止其他中断请求生效。

4、将用户自己定义的中断向量地址装入程序计数器PC中。

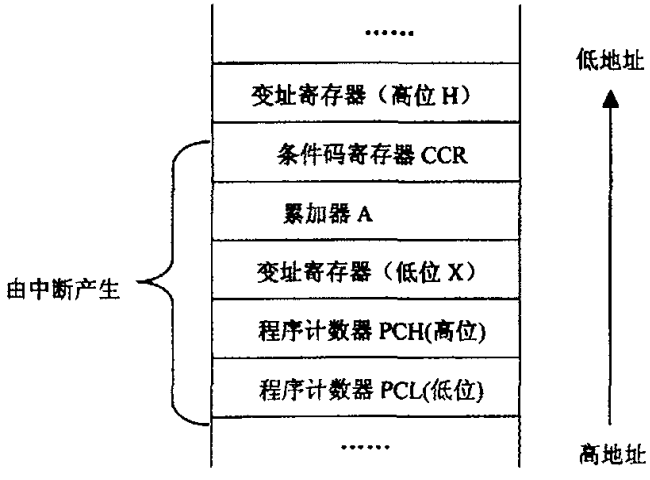


图3-4 MC9S08GB60运行RTOS时的中断堆栈结构

从以上MC9S08GB60单片机在中断时所执行的操作中，我们可以发现唯有变址寄

存器X的高位没有压入堆栈保存（主要是考虑与68HC05兼容）。但在RTOS中必须保存所有的寄存器，这样才能正确保存该任务的状态。根据以上分析，我们定义了如图3-4所示的MC9S08GB60运行RTOS时的中断堆栈结构，该中断结构不仅在所有的中断发生时使用(编写中断服务子程序要按此结构来设计)，而且在任务级切换和任务堆栈初始化时必须使用，可以说是该系统设计中的核心结构。[11]

(2) 中断服务子程序

中断服务子程序示意性代码如下：

```
{
  保存全部CPU寄存器;              /*变址寄存器X的高位入当前任务堆栈*/
  OSIntNesting直接加1;                              /*中断嵌套数加1*/
  if（OSIntNesting==1） {OSTCBCur->OSTCBStkPtr=SP};
                          /*中断嵌套数为1，在当前任务的任务控制
                                                    块中保存堆栈指针*/
  清中断源;
  重新开中断;
  执行用户代码做中断服务;
  调用OSIntExit()；              /*判断返回到被中断的任务，还是切换到
                                                    更高优先级的任务*/
  恢复所有CPU寄存器;
  执行中断返回指令;
}
```

(3) 脱离中断函数OSIntExit（）

调用脱离中断函数 OSIntExt()标志着将要退出中断服务子程序。此时系统要判定有没有优先级较高的任务被中断服务子程序(或任一嵌套的中断)唤醒了。如果有优先级高的任务进入了就绪态，系统就返回到那个高优先级的任务。如果需要进行任务切换，将调用 OSIntCtxSw()进行切换，保存的寄存器的值是在此时恢复的。如果处于中断嵌套中，OSIntExit()将直接返回上一层中断。

(4) 中断级任务切换函数OSIntCtxSw（）

OSIntCtxSw()的示意性代码如下：

```
{
```

调用用户定义的 OSTaskSwHook();

OSTCBCur = OSTCBHighRdy; /*当前任务的指针指向要恢复运行的任务*/

OSPrioCur = OSPrioHighRdy; /*要恢复运行任务的当前任务的复制给当前任务的当前任务的*/

得到需要恢复的任务的堆栈指针:

堆栈指针 = OSTCBHighRdy->OSTCBStkPtr;

将所有处理器寄存器从新任务的堆栈中恢复出来;

执行中断返回指令;

}

(5) 中断服务程序体设计

如上所述，中断服务子程序应主要包括进入中断，执行中断服务程序体，退出中断三个大的部分。其中中断服务程序体是中断所要进行的真正工作。我们上面讲的主要是 RTOS 如何在保持系统稳定的前提下进入中断和退出中断，这是固定和必须遵守的。但具体的中断服务程序体则要根据具体情况编写。用户中断服务中做的事要尽可能地少，把大部分工作留给中断处理任务去做。中断服务子程序通知某任务去做事的手段是调用以下函数之一：OSMboxPost (), OSSemPost ()。中断服务程序体完成以后，要调用 OSIntExt()退出中断。

3.1.9 时钟节拍

系统需要用户提供周期性信号源，用于实现时间延时和确认超时。节拍率应在每秒 5 次到 100 次之间。时钟节拍率越高，系统的额外负荷就越重。时钟节拍的实际频率取决于用户应用程序的精度。时钟节拍源使用专门的硬件定时器。在具体设计时，我们设计使用 MC9S08GB60 的定时器 TPM1 来作为时钟节拍源，时钟节拍率为每秒 10 次，时间间隔为 100ms。

用户程序必须在多任务系统启动以后，再启动时钟节拍源计时，也就是在调用 OSStart()之后做的第一件事是初始化定时器中断。否则，时钟节拍中断有可能在系统启动第一个任务之前发生，此时系统是处在一种不确定的状态之中，用户应用程序有可能会崩溃。

系统中的时钟节拍服务是通过在时钟中断服务子程序中调用 OSTimeTick()实现的。因为在 C 语言里不能直接处理 CPU 的寄存器，所以这段代码必须用汇编语言编写。在编写时钟节拍中断服务子程序时，在中断服务程序体中调用 OSTimeTick()，由它

执行具体操作。其他的要严格按照中断服务子程序的编写方法进行。

OSTimeTick()的工作是给每个用户任务控制块 OS_ TCB 中的时间延时项 OSTCBDly 减1(如果该项不为零的话)。OSTimeTick()从 OSTCBList 开始，沿着 OSTCB 双向链表做，一直作到空闲任务。当某任务的任务控制块中的时间延时项 OSTCBDly 减到了零，则将该任务置为就绪态。而被函数 OSTaskSuspend()挂起的任务则不会进入就绪态。OSTimeTick()的执行时间直接与应用程序中建立了多少个任务成正比。

OSTimeTick()还通过给一个 32 位的变量 OSTime 加 1 来累积从开机以来的时间，为以后的时间管理打下基础。

3.1.10 系统初始化

调用系统其他服务之前，系统要求用户首先调用系统初始化函数 OSInit ()，它初始化系统所有的变量和数据结构。主要完成以下工作：

- 初始化 32 位时钟节拍数。
- 初始化中断嵌套层数。
- 初始化任务调度锁定层数。
- 设定多任务调度尚未开始。
- 初始化任务切换数。以上均为 0。
- 初始化就绪表。表明没有任务就绪。
- 初始化优先级表。全部为 0。
- 定义 MC9S08GB60 单片机的堆栈增长方向为从高向低增长。
- 设定最大任务数 OS-MAX-TASKS 为 10。定义为 10 是考虑应用系统设计时的需求。
- 设定最小的任务优先级为 15，它被赋予给空闲任务。
- 建立空闲任务，该任务始终处于就绪态。具体分析见后面。
- 建立空闲任务的任务控制块，将它置于任务控制块链表的起始处。
- 建立任务控制块(OS_ TCB)的空数据结构缓冲区，该缓冲区允许系统从缓冲区中迅速得到或释放其中一个任务控制块。任务控制块的数目取决于最大任务数。
- 建立空事件表(OS_ EVENT)缓冲区。该缓冲区也是单向链表，允许系统从缓冲区中迅速得到或释放其中一个元素。

3.1.11 系统启动

多任务的启动是用户通过调用OSStart()实现的。但在系统启动之前，用户至少要建立一个应用任务。当调用OSStart()时，OSStart()从任务就绪表中找出用户建立的优先级最高任务的任务控制块，并将指针 OSTCBHighRdy 指向该任务控制块(OS_TCB)。然后，OSStart()调用高优先级就绪任务启动函数

OSStartHighRdy()。实质上，OSStartHighRdy()是将优先级最高的就绪任务的任务堆栈中保存的值弹回到 CPU 寄存器中，然后执行一条中断返回指令，中断返回指令强制执行该任务代码。OSStartHighRdy()将永远不返回OSStar()。

OSStartHighRdy()的示意性代码如下：

```
{
 调用用户定义的OSTaskSwHook();
 OSRunning  = TRUE;                          /*表明多任务调度开始*/
 得到将要恢复运行任务的堆栈指针：
     堆栈指针 = OSTCBHighRdy->OSTCBStkPtr;
 从新任务堆栈中恢复处理器的所有寄存器;
 执行中断返回指令;
 }
```

该函数功能是运行优先级最高的就绪任务，在调用OSStart()之前，用户必须先调用 OSInit()，并且已经至少创建了一个任务(系统初始化时已建立了空闲任务)。OSStartHighRdy()的默认指针 OSTCBHighRdy 指向优先级最高就绪任务的任务控制块(OS_TCB)，这是由OSStart()设置好的。显然， OSTCBHighRdy->OSTCBStkPtr 指向的是任务堆栈的顶端。任务堆栈的结构必须符合图3-4所设计的堆栈结构。

3.2 任务管理

3.2.1 建立任务

要使用RTOS管理任务，用户必须先使用OSTaskCreate()函数来建立任务。

函数原形：int OSTaskCreate(void (*task)(void *pd), void *pdata, OS_STK *ptos, int prio);

参数说明：task 是任务代码的指针；pdata 是指向任务开始执行时传递给任务的参数的指针；ptos 是指向任务堆栈的栈顶指针；prio 是该任务的优先级。

(1)函数 OSTaskCreate（）的示意性代码如下：

```
{
  prio 应在 0～OS_ LOWEST_ PRIO 之间;        /*否则，返回优先级无效错误*/
  关中断，进入临界区;
  规定优先级应还没有建立任务;                /*否则，返回任务已经存在错误*/
  标明该优先级上建立任务;
  开中断，退出临界区;
  调用 OSTaskStkInit();                      /*建立任务堆栈，返回堆栈栈顶*/
  调用 OSTCBInit();                  /*初始化 OS_ TCB，若初始化失败，则设置
                    OSTCBPrioTbl[prio]的入口为 0 ，放弃该任务优先级*/
   OSTaskCtr++;                              /*保存产生的任务数目*/
  IF (0SRunning==TRUE)  OSSched();           /*新建立的任务若为某个任务
                                  执行中所创建，则需任务调度，以判断新建
                                  立的任务是否有更高的优先级*/
  }
```

(2) 函数 OSTaskStkInit ()

负责建立任务的堆栈，并初始化。初始状态的堆栈模拟发生一次中断后的堆栈结构，该结构必须遵守图 3-4 所设计的中断堆栈结构。函数返回初始化后的堆栈栈顶指针。

函数原形：void *OSTaskStkInit (void (*task) (void *pd)， void *pdata，void *ptos， INT16U opt);

参数说明：当调用 *OSTaskCreate()*函数建立任务时，需要 4 个参数：*task* 是任务起始代码的指针；pdata 是当任务开始执行时传递给任务的参数的指针；ptos 是分配给任务堆栈的栈顶指针；prio 是分配给任务的优先级。OSTaskStkInit()在初始化堆栈时需要 task，pdata 和 ptos 三个参数，另一个参数 OPT 在此处没有使用，为扩展使用保留。

我们已知 MC9S08GB60 单片机堆栈是从上往下长的(从高地址往低地址方向增长)。函数 OSTaskStkInit()执行过程如下：

• 先定义一个指针，将它指向堆栈开始的位置，即堆栈栈顶，设为 stk，其指向的向量的值为*stk。

• 将*stk 的值赋为 ptos，保存堆栈指针，stk 减 1 移向下一个位置。

• 将*stk 的值赋为 task(任务启始代码的指针，即程序计数器的内容)，stk 减 1 移向下一个位。

• 将*stk 的值赋为 0x00， 初始化变址寄存器 X 和累加器 A，stk 减 1 移向下一个位置。

• 将*stk 的值赋为 0x00， 初始化条件码寄存器 CCR 和变址寄存器 H，stk 减 1 移向下一个位置。

• 将返回此时 stk 的值，即堆栈初始化完后的堆栈栈顶指针。

新任务创建后初始化的堆栈结构和内容见图 3－5。

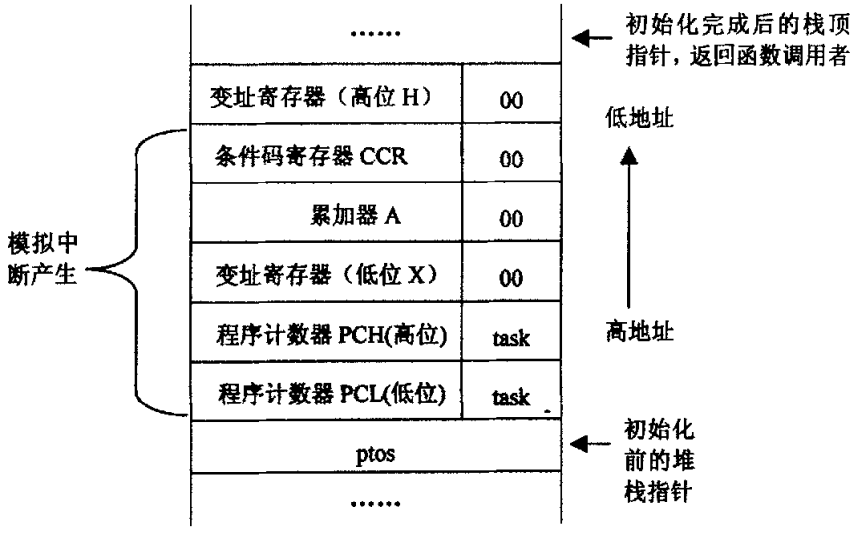


图 3-5 新任务创建后初始化的堆栈结构和内容

(3) 函数 OSTCBInit（）

初始化堆栈后，调用 OSTCBInit()，从空闲的 OS_ TCB 池中获得一个任务控制块，并根据该任务的具体情况初始化 OS_ TCB。OSTCBInit()将 OS_ TCB 插入到已建立任务的 OS_ TCB 的双向链表中。该双向链表开始于 OSTCBList，而一个新任务的 OS_ TCB 常常被插入到链表的表头。最后，将该任务处于就绪状态，并且 OSTCBInit()向它的调用者 OSTaskCreate()返回一个代码表明 OS_ TCB 已经被分配和初始化了。

3.2.2 删除任务

删除任务是将任务返回并处于休眠状态，并不是说任务的代码被删除了，只是任务的代码不再被系统调用。通过调用 OSTaskDel()就可以完成删除任务的功能。可以

通过指定 OS_ PRIO_ SELF 参数来删除自己。删除任务的时候要确保以下几点：

- 所要删除的任务并非是空闲任务，因为空闲任务不允许删除；
- 删除的操作不是在中断服务子程序中进行的；
- 要保证被删除的任务是确实存在的。

一旦所有的条件都满足了，该任务的 OS_ TCB 就会从所有的数据结构中被移除。分两步移去任务，以减少中断响应时间。首先，如果任务处于就绪表中，它会直接被移去。其次，如果任务处于邮箱或信号量的等待表中，它就从自己所处的表中被移去。

3.2.3 挂起任务

任务挂起是将任务的状态转为挂起状态。挂起任务可通过调用 OSTaskSuspend() 函数来完成。被挂起的任务只能通过调用 OSTaskResume( )函数来恢复。如果任务在被挂起的同时也在等待延时期满，那么，挂起操作需要被取消，而任务继续等待延时期满，并转入就绪状态。任务可以挂起自己或者其他任务。挂起任务的时候要满足以下几点：

- 空闲任务是不允许挂起的；
- 任务的优先级必须在 0 到 OS_ LOWEST_ PRIO 之间；
- 要保证被挂起任务是确实存在的。

一旦所有的条件都满足了，就可以将它从就绪表中被移除，同时将任务的 OS_ TCB 的状态设为 OS_ STAT_ SUSPENDB 标志，以表明任务正在被挂起。

3.2.4 恢复任务

被 OSTaskSuspend( )挂起的任务只有通过调用 OSTaskResume( )才能恢复。

只要不是空闲任务，同时具有合理有效的优先级的任务，通过清除任务的 OS_ TCB 中的 OS_ STAT_ SUSPEND 标志，同时将 OSTCBDly 域置为 0，只有以上两个条件都满足时，任务才脱离挂起态得以进入就绪状态。

3.3 时间管理

系统要求用户提供定时中断来实现延时与超时控制等功能。这个定时中断称为时钟节拍，它应该每秒发生 5 至 100 次。时钟节拍的实际频率是由用户的应用程序决定的。时钟节拍的频率越高，系统的负荷就越重。我们设计的时钟节拍为每秒 10 次，时间间隔为 100ms。

3.3.1 任务延时

在系统设计时，任务可以延时一段时间，这段时间的长短是用时钟节拍的数目来确定的。通过 OSTimeDly()函数调用来实现这种功能。调用该函数会使任务从就绪表中移除，系统进行一次任务调度，并且执行下一个优先级最高的就绪态任务。任务调用 OSTimeDly()后，一旦规定的时间期满或者有其他的任务通过调用 OSTimeDlyResume()取消了延时，它就会马上进入就绪状态。但是只有当该任务在所有就绪任务中具有最高的优先级时，它才会立即运行。

用户的应用程序在调用该函数时提供延时的时钟节拍数((1 到 65535 之间的数)。如果用户指定 0 值，则表明用户不想任务延时，函数会立即返回到调用者。非 0 值会使得任务延时函数 OSTimeDly()将当前任务从就绪表中移出。接着，这个延时节拍数会被保存在当前任务的 OS_ TCB 中，并且通过 OSTimeTick()，每隔一个时钟节拍就由时钟节拍中断服务子程序减少一个延时节拍数。

由于系统在时钟节拍前，就对提供延时的时钟节拍数减 1，所以如果一个任务只希望延迟一个时钟节拍，那么必须指定两个时钟节拍。

3.3.2 恢复延时的任务

延时的任务可以不等待延时期满，而是通过其他任务取消延时来使自己处于就绪态。这可以通过调用 OSTimeDlyResume()指定要恢复的任务的优先级来完成。

函数 OSTimeDlyResume()

函数原形：int OSTimeDlyResume(int prio);

参数说明：prio 要处理任务的优先级

函数 OSTimeDlyResume()示意性代码如下:

```
{ if (prio < OS_LOWEST_PRIO)                    /*对空闲任务无效  */
  关中断，进入临界区;
  ptcb = (OS_TCB *)OSTCBPrioTbl[prio];          /*取任务 TCB 的地址  */
  if (ptcb != 0)                  /*否则，开中断,返回“任务不存在”代码 */
  if (ptcb->OSTCBDly != 0)      /*否则,开中断,返回“无延时”代码  */
  ptcb->OSTCBDly = 0;                           /*强置 0,取消延时 */
  if (任务没有被挂起)               /*否则,开中断,返回“无错”代码 */
  将任务 置为就绪态;
  开中断，退出临界区;
  OS_Sched;                      /*检查被恢复的任务是否拥有最高优先级*/
```

返回“无错”代码;

}

3.3.3 获取时间

无论时钟节拍何时发生，系统都会将一个 32 位的计数器加 1。这个计数器在用户调用 OSStart()初始化多任务时从 0 开始计数。用户可以通过调用 OSTimeGet()来获得该计数器的当前值。也可以通过调用 OSTimeSet()来改变该计数器的值。

由于 MC9S08GB60 上增加和读取一个 32 位的数都需要数条指令，而这些指令都需要一次执行完毕，不能被中断打断，所以在访问 OSTime 的时候中断是关掉的。

3.4 任务之间的通信和同步

在多个任务并行运行过程中，任务之间由于共享系统的某些资源而会受到相互制约。另外，有些操作需要多个任务相互协作共同完成。因此，任务间必须进行信息交换。本节介绍两种用于数据共享和任务通信的方法：信号量和邮箱。所有的信号都被看成是事件(event)。所以把用于通信的数据结构叫做事件控制块 (ECB,Event Control Block)。

一个任务或者中断服务子程序可以通过事件控制块来向另外的任务发信号。一个任务可以等待另一个任务或中断服务子程序给它发送信号。但必须注意，只有在任务中可以等待事件发生，中断服务子程度是不能等待事件发生的，否则中断将一直处于等待状态而无法正常返回。对于处于等待状态的任务还可以给它指定一个最长等待时间，以此来防止因为等待的事件没有发生而无限期地等待下去。多个任务可以同时等待同一个事件的发生。在这种情况下，当该事件发生后，所有等待该事件的任务中，优先级最高的任务得到了该事件，并进入就绪状态，准备执行。

3.4.1 事件控制块 ECB

系统通过定义的 OS_EVENT 数据结构来维护一个事件控制块的所有信息。该结构中除了包含事件本身的定义，如用于信号量的计数器、用于指向邮箱的指针等，还定义了等待该事件的所有任务的列表。该结构包含以下内容：

.OSEventPtr 指针,只有在所定义的事件是邮箱或者消息队列时才使用。此时它指向一个消息。

.OSEventTbl[] 和 .OSEventGrp 类似 OSRdyTbl[]和 OSRdyGrp，包含的是

.OSEventCnt 当事件是一个信号量时，.OSEventCnt 是用于信号量的计数器。

.OSEventType 定义了事件的具体类型。它可以是信号量(OS_EVENT_TYPE _SEM)或邮箱(OS_EVENT_TYPE_ MBOX)。用户要根据该域的具体值来调用相应的系统函数，以保证对其进行的操作的正确性。

每个等待事件发生的任务都被加入到该事件事件控制块中的等待任务列表中，该列表包括.OSEventGrp 和.OSEventTbl[]两个域。在这里，所有的任务的优先级被分成2 组(每组 8 个优先级)，分别对应.OSEventGrp 中的第 0 位和第 1 位。当某组中有任务处于等待该事件的状态时，.OSEventGrp 中对应的位就被置位。相应地，该任务在.OSEventTbl[]中的对应位也被置位。.OSEventTbl[]数组的大小由系统中任务的最低优先级决定。这样，可以在任务优先级比较少的情况下，减少系统对系统 RAM 的占用量。

当一个事件发生后，该事件的等待事件列表中优先级最高的任务得到该事件。其算法与计算任务在任务就绪表中位置的算法相同。其原理为:任务优先级的最低 3 位决定了该任务在相应的.OSEventTbl[]中的位置，紧接着的 3 位则决定了该任务优先级在.OSEventTbl[]中的列。

对一个事件控制块一般有以下几种操作。

(1) 初始化一个事件控制块

当建立一个信号量或邮箱时，相应的建立函数 OSSemInit() , OSMboxInit()将通过调用 OSEventWaitListInit()对事件控制块中的等待任务列表进行初始化。该函数初始化一个空的等待任务列表，其中没有任何任务。该函数的调用参数只有一个，就是指向需要初始化的事件控制块。

(2) 使一个任务进入就绪态

当发生了某个事件，该事件等待任务列表中的最高优先级任务(Highest Priority Task HPT)置于就绪态时，该事件对应的 OSSemPost ()，OSMboxPost ()函数调用 OSEventTaskRdy()实现该操作。换句话说，该函数从等待任务队列中删除 HPT 任务，并把该任务置于就绪态。OSEventTaskRdy()要在中断禁止的状态下调用。

函数原形:void OSEventTaskRdy(OS_ EVENT *pevent,void *msg,int msk);

参数说明:pevent 指向事件控制块的指针；msg 指向消息的指针；msk 屏蔽码。

函数 OSEventTaskRdy()示意性代码如下:

```
{
  计算 HPT 在.OSEventTbl[]中的字节索引;
```

利用该索引得到HPT在.OSEventGrp中的位屏蔽码;

计算HPT在.OSEventTbl[]中相应位的位置;

计算出HPT的优先级;

从等待任务列表中删除该任务;

修改任务的TCB中需要改变的信息; /*.OSTCBDly清0、清指向该事件控制块的指针、若函数由OSMboxPost()调用，相应的消息传递给HPT 、根据传递参数msk，清位屏蔽码*/

```
if (ptcb->OSTCBStat == OS_STAT_RDY)
    {HPT插入到就绪任列表};
}
```

(3)使一个任务进入等待某事件发生状态

当某个任务要等待一个事件的发生时，相应事件的OSSemPend()或OSMboxPendO函数会调用OSEventTaskWait()函数将当前任务从就绪任务表中删除，并放到相应事件事件控制块的等待任务表中。

在该函数中，首先将指向事件控制块的指针放到该任务的任务控制块中，接着将任务从就绪任务表中删除，并把该任务放到事件控制块的等待任务表中。

(4)由于等待超时而将任务置为就绪态

当在预先指定的时间内任务等待的事件没有发生时，OSTimeTick()函数会因为等待超时而将任务的状态置为就绪。在这种情况下，事件的OSSemPend()或OSMboxPend()函数会调用OSEventTO()来完成这项工作。该函数负责从事件控制块中的等待任务列表里将任务删除，并把它置成就绪状态。最后，从该任务的任务控制块中将指向事件控制块的指针删除。调用OSEventTO()应当先关中断。

3.4.2 信号量

信号量实际上是一种约定机制，在多任务内核中普遍使用信号量用于:

- 控制共享资源的使用权;
- 标志某一个事件的发生;
- 使两个任务的行为同步;

系统中的信号量由两部分组成:一个是信号量的计数值，它是一个16位的无符号整数(0到65535之间)；另一个是由等待该信号量的任务组成的等待任务列表。

系统提供了3个对信号量进行操作的函数。它们是:OSSemCreate(), OSSemPend()

和 OSSemPost()。其中 OSSemPost()函数可以由任务或者中断服务子程序调用，而OSSemPend()函数只能由任务程序调用。

(1) 建立一个信号量 OSSemCreate ()

使用一个信号量之前，首先要建立该信号量，即调用 OSSemCreate()函数，对信号量的初始计数值赋值。如果信号量是用来表示一个或者多个事件的发生，那么该信号量的初始值应设为 0。如果信号量是用于对共享资源的访问，那么该信号量的初始值应设为 1。

函数原形:void OSSemCreate(int cnt);

参数说明：cnt 信号量的初始值

函数 OSSemCreate()示意性代码如下：

```
{ if (OSIntNesting==0)                    /*中断服务子程序不能创建信号量*/
  关中断，进入临界区;
  pevent=OSEventFreeList;          /*从空闲事件控制块链表中，获得一 ECB */
  if (OSEventFreeList != (OS_EVENT *)0)
    {调整 OSEventFreeList };
  开中断，退出临界区;
 if (pevent != (OS_EVENT *)0)
    { ECB 初始化 };                    /* .OSEventType =SEM 、.OSEventCnt=
                                              cnt、.OSEventPtr=0 */
调用 OSEventWaitListInit(pevent);          /*初始化 ECB 的等待任务列表*/
return(pevent);
}
```

(2) 等待一个信号量 OSSemPend ()

函数原形:void OSSemPend(OS_EVENT *pevent,int timeout,int *err);

参数说明:pevent 指向信号量事件控制块的指针；timeout 最大等待时间；err 指向返回错误的指针

函数 OSSemPend()示意性代码如下：

```
{ if (OSIntNesting==0)                          /*中断服务子程序不能等待*/
  if (pevent != (OS_EVENT *)0)                  /*指向 ECB 的指针应非空*/
  if (.OSEventType ==SEM)
```

```
关中断，进入临界区;
if (.OSEventCnt ==0)          /*否则>0，.OSEventCnt 减 1，开中断，退出*/
当前任务的 TCB 的.OSTCBStat 和.OSTCBDly 赋值;
                                        /*置睡眠态标志和最长等待时限*/
调用 OSEventTaskWait (pevent);                  /*使任务进入睡眠状态*/
开中断，退出临界区;
OS_Sched;                              /*让下一个优先级最高的任务运行*/
关中断，进入临界区;
if (.OSTCBStat ==SEM)                          /*否则，任务已得到信号量，
                                        .OSTCBEventPtr 清 0，开中断*/
调用 OSEventTo(pevent);                /*从等待任务列表中删除超时任务*/
开中断，退出临界区;
返回超时出错;
}
```

(3) 发送一个信号量 OSSemPost()

函数原形:void OSSemPost(OS_EVENT *pevent);

参数说明:*pevent* 指向信号量事件控制块的指针

函数 OSSemPost()示意性代码如下:

```
{ if (pevent != (OS_EVENT *)0)                  /*指向 ECB 的指针应非空*/
 if (.OSEventType ==SEM)
 关中断，进入临界区;
 if (pevent->OSEventGrp!= 0x00)                  /*有任务等待该信号量*/
  then {调用 OSEventTaskRdy();            /*把最高优先级任务从等待任务
                                          列表中删除并使其进入就绪态*/
        开中断，退出临界区;
        OS_Sched ;     /*检查该任务是否是系统中最高优先级的就绪任务*/
       返回正确代码;
       }
  else{ if (pevent->OSEventCnt<65535)   /*否则,开中断,返回信号量溢出*/
           pevent->OSEventCnt++;
```

开中断，退出临界区;

返回正确代码;

}

3.4.3 邮箱

邮箱是系统中的另一种通信机制，它可以使一个任务或者中断服务子程序向另一个任务发送一个指针型的变量。该指针指向一个包含了特定“消息”的数据结构。使用邮箱之前，必须先建立该邮箱。该操作可以通过调用 OSMboxCreate()函数来完成，并且要指定指针的初始值。

系统提供了 3 种对邮箱的操作:OSMboxCreate(), OSMboxPend() , OSMboxPost ()。

(1) 建立邮箱

OSMboxCreate ()函数基本上和函数 OSSemCreate()相似。不同之处在于事件控制块的类型被设置成 OS_ EVENT_ TYPE_ MBOX，以及使用.OSEventPtr 域来容纳消息指针，而不是使用.OSEventCnt 域。OSMboxCreate)函数的返回值是一个指向事件控制块的指针。这个指针在调用函数 OSMboxPend(),OSMboxPost()时使用。因此，该指针可以看作是对应邮箱的句柄。如果系统中已经没有事件控制块可用，函数将返回一个 NULL 指针。邮箱一旦建立，是不能被删除的。

(2) 等待一个邮箱中的消息

OSMboxPend()用来等待从一个邮箱中接收消息。同样该函数的功能和函数 OSSemPend()相似，下面只把二者执行过程中的不同之处说明介绍一下：

1、如果信号量本来就有效（信号量的计数值非 0)，则信号量的计数值递减，OSSemPend()返回“无错”代码给调用者；而邮箱中若有消息，则取出该消息，返回给调用函数，将 NULL 指针放入邮箱中，同时 OSMboxPend()返回“无错”代码给调用者。

2、若因等待而进入睡眠状态的任务，由 OSSemPost()发出的信号量唤醒，则只需把该任务的 TCB 中指向信号量 ECB 的指针删除即可；有其他任务的向该任务的邮箱发送消息后，将其唤醒，OSMboxPend()函数要检查是否有消息被放到该任务的任务控制块中，若有，对应的消息被返回到调用函数，同时将该任务的 TCB 中指向邮箱 ECB 的指针删除。

(3) 发送一个消息到邮箱中 OSMboxPost ()

函数 OSMboxPost()的功能与发送一个消息给信号量函数 OSSemPost()十分相似，

只是在没有任务等待信号量或邮箱中的消息时，有点异同：此时，信号量计数值只是简单的加 1，除非计数值大于 65534，方返回溢出代码；而 OSMboxPost()随之检查邮箱中是否有消息，因邮箱中只能保存一则消息，若已有消息，则返回“邮箱已满”代码，否则，指向消息的指针保存到邮箱中。[12]

3.5 SDF-RTOS 的测试

由于 SDF-RTOS 内核的源代码是同用户的应用程序一同编译，并融入用户应用代码之中，因此对 SDF-RTOS 进行测试时应有用户程序。我们设计了一个共有 2 个任务的用户程序，任务 1 在初始化时钟中断以后，就进入了一个死循环。在这个循环里，任务 1 一方面以 1s(秒)为周期改变并行 I/O 口 PORTA 第 0 个引脚的输出电压，另一方面每隔 4s 便向任务 2 发送 1 个信号。而任务 2 则始终等待任务 1 发来的信号，一旦收到信号，便改变并行 I/O 口 PORTA 第 1 个引脚的输出电压。具体的代码如下：

```
#include "includes.h"
#include "9S08GB60.h"
#define TASK_STK_SIZE  55                          /*定义任务栈的大小*/
INT8U TaskAStk[TASK_STK_SIZE];
INT8U TaskBStk[TASK_STK_SIZE];
void TaskA(void *pdata);
void TaskB(void *pdata);
void HardwareInit(void);
OS_EVENT  *sem;
void TaskA(void* pdata)
 {
  int count=0;
  asm("sta $1800");
  pdata = pdata;
  TimerInit();                          /*初始化时钟中断，设置时钟节拍为 10*/
       for(;;)
   {
     asm("sta $1800");
     PTAD &= 0xFE;
```

```
        OSTimeDly(5);                                    /*延时 0.5 秒*/
        PTAD |= 0x01;
        OSTimeDly(5);
      count++;
      if(count==4){
      OSSemPost(sem);                                    /*发送一个信号量*/
              count=0;
              }
      }
        asm("sta $1800");
}
void TaskB(void* pdata)
{
  unsign char err;
    asm("sta $1800");                                    /*看门狗喂食*/
      for(;;)
    {
      asm("sta $1800");
      OSSemPend(sem,0,&err);                             /*等待一个信号量*/
      PTAD &= 0xFD;
      OSSemPend(sem,0,&err);
      PTAD |= 0x02;
    }
}
void HardwareInit(void)
{
  asm("sta $1800");
   PTAD  = 0x00;
   PTADD = 0x0F;
   }
```

```
void main(void) {
    HardwareInit();                          /*硬件初始化，即初始化A口*/
    OSInit();                                /*初始化多任务环境*/
    sem=OSSemCreate(0);                      /*创建一个信号量*/
    OSTaskCreate(TaskA, (void*)0, (void*)&TaskAStk[TASK_STK_SIZE], 10);
    OSTaskCreate(TaskB, (void*)0, (void*)&TaskBStk[TASK_STK_SIZE], 9);
    OSStart();                               /*开始多任务调度*/
}
```

在主程序 main()中，用户必须先调用 OSInit()，然后创建各个任务和信号量等，最后调用 OSStart()，以启动内核运行，开始正常的任务调度。[13]

3.6 SDF-RTOS 中 RAM 的使用

通常MCU的RAM资源都比较有限，大一点的也只有几到几十千字节(9S08GB60片内 RAM 为 4KB)，因此合理使用有限的系统资源对单片机的开发就尤为重要。单片机初始化时,堆栈指针总是被初始化到最高地址（对堆栈战指针是递减的而言），全局变量从 RAM 的最低地址开始分配，全局变量定义后，其所占用的 RAM 空间就不变了。而栈空间是程序运行时动态变化的，若栈空间过度膨胀直至冲入全局变量区，系统就会崩溃。在多任务环境下，每个任务都有独立的栈空间，某一任务栈空间的破坏，也会导致整个系统崩溃。SDF-RTOS 的使用，使应用程序的设计得到简化，系统实时性得到保证，任务间的竞争由 RTOS 解决，但 SDF-RTOS 也将占用一定的系统资源。

3.6.1 SDF-RTOS 中 RAM 使用分析

SDF-RTOS 定义了如 OSPrioCur、OSRunning、OSTime 等共 30 个全局变量，总共占用了 45B 的空间。一个任务控制块指针表占用 2B，一个事件控制块指针表。

一个最基本的任务控制块占用 14B；使用信号量和消息邮箱，要分别增加一个指针变量，即增加 4B；删除任务 TCB 中增加一个布尔量，占用 1B，一个任务控制块占用 19B。需要多少个任务控制块，与最多任务数 OS_MAX_TASKS 有关。

事件控制块用于管理信号量、消息邮箱等事件，每个事件只提供建立、等待、发送 3 个基本功能，同时 OS_LOWEST_PRIO 设为 15，一个事件控制块共占用 8B。

在多任务环境下，每个任务都是单独的，每个任务都要有自己的栈空间。任务的

栈空间主要有以下几个用途：任务调度与中断、中断嵌套、子程序调用嵌套，函数的局部变量及函数调用中的参数传递。

由于 SDF-RTOS 下的任务调度切换或中断，要保存当前任务 CPU 的全部内容，对于 MC9S08GB60 而言，就是 6B；通常单片机应用中，中断嵌套很少超过 3 层，SDF-RTOS 中允许中断嵌套数为 3 层，SDF-RTOS 内核中不支持在嵌套的中断服务子程序运行时做任务调度，每增加一层中断嵌套给每个任务增加 6B 的栈空间；同样子程序的调用也占用堆栈空间，每一层子程序调用要保存调用的返回地址，对于 9S08GB60 就是 2B,设定最多调用层数为 6，故子程序调用需预留 12B 空间；子程序调用即函数调用中的参数传递是靠堆栈进行的，每个任务的局部变量也是保存在堆栈中的，任务建立函数 OSTaskCreate()为 SDF-RTOS 中参数传递最多的函数，共要传递 4 个参数，同时函数本身还有 2 个局部变量，总共占用了 10B。我们设计每个任务栈空间为 55B, 尚留有 9B 的冗余。

3.6.2 SDF-RTOS 中 RAM 占用估算

在实际的 8 位 MCU 开发中，很少有超过 10 个任务的，同时用于完成任务间的同步与通讯的信号量、消息邮箱等事件也很少超过 10 个。下面，我们估算 SDF-RTOS 在 MC9S08GB60 上运行最多 10 个任务、10 个事件、最低优先级数值为 15，并使用信号量和消息邮箱时的 RAM 占用量：

任务栈	55×10=550
任务控制块	19×10=190
任务控制块指针表	2×16=32
事件控制块	8×10=80
事件控制块指针表	2×10=32
其他内核全局变量	45
总计	929

由上可见，SDF-RTOS 运行 10 个任务，同时使用 10 个信号量、消息邮箱来完成任务间的同步与通讯，占用 MC9S08GB60 RAM 资源的四分之一，尚有 3KB 多可用于应用程序的变量。[9]

3.7 FLASH 空间的使用

由于 RTOS 是作为应用程序的背景程序和应用程序编译在一起的，所以无法从最

终编译的程序中判断 RTOS 系统本身占用了多少 FALSH 空间。一种近似的统计方法是逐个测试单独编译后的应用任务大小，然后由最终的程序大小减去各个应用任务的大小就是 RTOS 系统占用的 FLASH 空间。经测算，RTOS 系统占用的 FLASH 空间为 4K-5K 字节。

第四章 SDF-RTOS 在暖气表中的应用

嵌入式系统开发中的嵌入式处理器、实时操作系统、仿真器、调试器等要素的选择是至关重要的，对系统的应用会产生重大的影响。本系统设计完成后，在其上嵌入一具体应用，通过分析其运行效果，检测系统移植后的工作情况。

实现时，应用所设计的 RTOS 中的理论和函数设计一个实际的应用系统并且在所设计的硬件平台上运行。根据运行过程中的各种实际情况，对 GB60 施加不同的触发方式，如中断引脚加低电压或下降沿，外部 I/O 口写入持续的高电平、低电平、或跳变的电平，结合相应的液晶显示，判断系统内程序运行情况，测试系统的稳定性和健壮性。

4.1 暖气表硬件电路设计

4.1.1 暖气表工作原理

暖气表的主要任务是对用户消耗的热量进行计量。每个用户采用热水串联供暖，即一根热水进管和一根热水出管。进户的高温热水经暖气片散热后从回水管返回。在热交换系统中安装暖气表，当水流经系统时，我们在单位时间内对热水进入室内的温度（入口温度）和流出室内的温度（出口温度）以及热水流量进行采集。根据热量公式计算的两者之差就是用户在单位时间内使用的热量。[14]

4.1.2 方案设计

我们以 MC9S08GB60 为主控芯片，在其最小系统基础上实现电路的设计,具体电路原理图见附录 B。根据暖气表的工作原理，分析所要完成的功能，可将系统分为如下几个模块：

- 键盘模块：用于各种控制信号的输入。
- LCD 显示模块：用于显示数据和各种信息。
- 温度采集模块：实时采集供回水温度，并将信号放大后送到 MCU 的 A/D 转换端口。
- 流量采集模块：产生计量脉冲信号，来一个计量脉冲，表示有 10 升热水流过。
- 电源模块：为 MCU 及各功能电路提供稳定的合适的工作电压。

4.1.3 硬件设计

(1) 键盘模块

16 位(4X4)键盘主要用来向单片机传送各种控制信号，我们可以根据具体需要来定义 16 个键所对应的操作，单片机根据所传送键值的不同，执行不同的操作，达到与外部交互的目的。单片机在处理外部传送的键值时，采用中断而不是轮循的方式，这样可以尽量减轻 CPU 的负荷以及减少系统对键盘处理的时间，提高系统对键盘的实时响应性。

为了减少系统主板的尺寸和增加系统配置的灵活性，将 16 位键盘单独设计为一个小的电路板，通过一个 IX10 的接口与系统主板相连。如果不需要键盘时就可以将它从系统主板上拆下。在具体设计时，选定 MC9S08GB60 的 A 口作为处理器与键盘的接口。由 PTA0 一 PTA7 共 8 个管脚与键盘连接。

(2) LCD 模块

LCD 即 Liquid Crystal Display，叫做液晶显示。在实际应用中将液晶显示器件、连接件、集成电路、PCB 线路板、背光源等结构件装配在一起构成液晶显示模块，简称“LCM（LCD Module)”。本设计 LCD 显示采用南京国显电子公司的 PDC-605 型 LCM。该 LCD 模块内部已经集成了 HT1621 多功能 LCD 控制器，可以驱动多段 LCD 字符是其主要特征，可以构成 LCD 显示模块和显示系统，与控制器的通信连线仅需要 3—4 根，同时它还包括了省电命令，有效地减少本身的功耗。可以显示 6 位数字和 2 组冒号，每个数字和小数点占用一个字节，2 个冒号占用一个字节，对应的每个笔段位一个二进制位，当数据位为“1”是显示开，数据位为“0”时显示关。

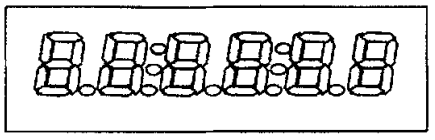
1）引脚定义

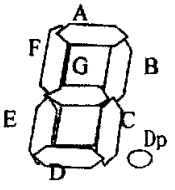
1. Vcc 电源+5V　2. Vss 地　3. DATA 数据　4. WR　5. RD 通常接 Vcc　6. CS

2）编码表

Bit / DATA	D7	D6	D5	D4	D3	D2	D1	D0
DATA1	F1	G1	E1	D1	A1	B1	C1	Dp1
DATA2	F2	G2	E2	D2	A2	B2	C2	Dp2
DATA3	F3	G3	E3	D3	A3	B3	C3	Dp3
DATA4	F4	G4	E4	D4	A4	B4	C4	Dp4
DATA5	F5	G5	E5	D5	A5	B5	C5	Dp5
DATA6	F6	G6	E6	D6	A6	B6	C6	Dp6
DATA7			后冒号				前冒号	

3） PDC-605 模块结构图





(3) 流量采集模块

流量传感器安装在管路系统上，用于计量流过供热回路的水的流量信号。具体设计中流量计采用湿式热水表，由于在水表的"0.01m^3"位安装的干簧管和 10 升齿轮上的磁铁作用，使得干簧管吸合则产生一个计量脉冲，送至 MC9S08GB60 的 16 脚 IRQ 端，表示"10 升齿轮"转过 1 周，流量累计 0 .01 m^3（10 升）。

(4) 温度采集模块

温度测量分别用两只特性相同的薄膜铂电阻 Pt100 完成，两只热电阻分别位于供热管路的进水口和出水口。在我们设计的测温电桥桥臂电阻值下（R24= R25=8KΩ，R26=100Ω），温度从 0℃变化到 100℃时，测温铂电阻 RT 上的输出电压变化范围是 0~24mV。而 MC9S08GB60 的 10 位的 A/D 转换精度的分度值为 3.22mV(3.3V/1024)。故采用精密运算放大器 OP07 对 RT 输出电压放大 137.5 倍后作为输入比较信号，送到 MCU 的 A/D 转换引脚。

经过电路分析可知，A/D 转换码 n（0~1023）与输入比较电压的对应关系式为：

$$V_{IN} = \frac{V_{REF}}{1024} \times n \qquad \text{（公式 4-1）}$$

当取 R24=R25=R 时，可以计算出输入 A/D 转换端口的输入比较电压为：

$$V_{IN} = \frac{R(R_T - R_{26})}{R^2 + RR_T + RR_{26} + R_{26}R_T} \times V_C \times K \qquad \text{（公式 4-2）}$$

式中 K 为运放的放大倍数，V_C 为 3.3V，是桥路电源电压，它与供给 MCU 的 A/D 转换的基准电压 VREF 共用，所以 V_C 的波动不会影响测量的稳定性和测量的准确度。[15] [16]

4.2 暖气表的软件设计

4.2.1 应用系统需求分析

根据暖气表的工作原理，暖气表的基本功能如下：

(1) 数据采集:流量采用脉冲计量中断的方式采集。两个温度采用对温度传

感器的阻值变化的方式采集；

(2) 热量计算:将采集的数据根据热量公式计算热量消耗；

(3) 数据显示:通过 LCD 显示目前的数据，包括热量、流量、温度等；

(4) 运行控制:通过键盘和系统传递控制命令。

以上是暖气表的基本功能需求，但为了更好的测试系统的性能，我们增加以下的功能:

(1) 时钟显示:使用 LCD 显示系统时间。这个模块可以在没有功能数据输出的情况下，显示系统内部的运行状态；

(2) 空闲任务:在没有其他任务运行的情况下，系统运行此空闲任务。

4.2.2 普通任务分析和划分

作为一个非常重要的软件层次，应用程序内部任务的合理划分对系统的运行效率、实时性和吞吐量影响极大。任务分解过细会引起任务频繁切换，增加系统开销。而任务分解不够彻底会造成原本可以并行的操作只能按顺序串行完成，从而减少了系统的吞吐量。为了达到系统效率和吞吐量之间的平衡与折衷，在应用设计中应遵循如下的任务分解规则：[17]

- 时间：两个任务所依赖的周期条件具有不同的频率和时间段；
- 异步性：两个任务所依赖的条件没有相互的时间关系；
- 清晰性/可维护性：两个任务可以在功能上或逻辑上互相分开；
- 具有慢速 I/O 操作的功能应当划分为单独任务；
- 不同优先级的操作划分为不同的任务；
- 各个任务间数据的通信量应该尽量小；
- 拥有大量运算的操作应当划分为一个单独的任务。

任务堆栈的分配：目前所有任务的任务堆栈都分配同样的大小，为 50 个字节，这只是一个基准数值。如果在任务数增多，RAM 不够使用的情况下，可以为每个任务单独定义任务堆栈大小。

根据上一节的需求分析，我们将整个系统划分为以下几个任务，并在划分的同时，定义各个任务任务优先级的大小。

(1) 空闲任务

这个任务是系统在初始化时建立的，永远处于就绪态，在没有其他任务进入就绪态时投入运行。它的优先级设为最低优先级，即 OS_ LOWEST_PRIO=15。

(2) LCD 时钟显示任务

该任务通过获取系统时钟节拍数，将其转换为时间显示（24 小时制），每隔一秒(即该任务每执行一次需要延迟 10 个时钟节拍)，在 LCD 上以 HH-MM-SS(小时一分钟一秒)的形式显示出来。如果系统运行正常，该显示会不断的变化。如果系统运行异常，该显示会停止。通过它的显示，我们也可以观察系统时钟数的变化，从而判断其他各个任务的运行是否正常。

该任务由于对实时性要求不高，所以其优先级可以定为 10。

(3) A/D 转换任务

A/D 转换主要用于检测两个温度的变化，采用铂热电阻将电阻阻值的变化转换为电压的变化，然后通过 A/D 转换的二个接口依次采集进水温度和出水温度。将采集后转换的信号存放在全局变量 Templ，Temp0。为了保证所采集数据的精确性，提高抗干扰能力，我们对施以算术平均值滤波措施，即连续分别对二个模拟信号采集 10 次，然后求平均值。由于对温度的采集是在一定时间内定时进行的，所以不必使用中断进行控制，而是使用系统延时，定时采集。

对温度变化信号的采集对时间要求相对较高，将其优先级定为 5。

(4) 键盘响应处理任务

键盘用于对正在运行的系统发送控制信号，为了处理键盘传送的键值，我们建立了一个邮箱 KeyBoardMbox，专门用于接收由键盘中断传送的键值。在没有按键被按下时，这个任务处于等待来自邮箱 KeyBoardMbox 消息的状态，即处于挂起状态。当有消息(键值)送到邮箱后，该任务根据所送键值的不同，作出相应的处理。可以通过 LCD 显示累计用热、进水温度、回水温度等数据，启动一个任务等等。

该任务由于是响应中断，所以其优先级定的较高，为 7。

(5) 计算任务

计算任务以用于暖气分户计量的暖气热量表为原形进行设计。

设在一定时间间隔内，热水在进入暖气片前，平均温度为 θ_f；在流出暖气片时，平均温度为 θ_r，q_m 为该时间间隔内流过的质量流量，则热水经暖气片所放射的热量 Q 为：

$$Q = c_p q_m(\theta_f - \theta_r) = c_p \rho q_v(\theta_f - \theta_r) \qquad \text{（公式 4-3）}$$

式中 c_p 为定压比热容，$c_p = k_0$ 视为常数。该方法计算简便，c_p 为常数，使得程序的计算量减少，计算速度大大加快。

温度通过检测铂电阻传感器 R_T 给出。A/D 转换电路对传感器转换后的结果是一

个 16 进制的数 n，将公式 4-1 和 4－2 联合将得到每一个 16 进制数 n 所对应的电阻值 R_n。根据 PT100 铂电阻在 0～600℃范围内，电阻随温度变化的非线性关系式（见公式 4－4），就可以计算出每一个 16 进制数 n 所对应的温度值 θ。

$$R_n = R_0(1 + \alpha T + \beta T^2) \qquad \text{（公式 4-4）}$$

其中 $\alpha = 3.90802\times10^{-3}$，$\beta = -5.80195\times10^{-7}$，$R_0$ 表示在 0℃时 PT100 的基础阻值，R_0=100Ω。

计算任务在单位时间内对采集的数据根据以上公式进行计算，所需要的温度由 A/D 转换电路测量，流量由流量中断服务子程序测量。这是暖气表系统的主要任务，并且由于是在单位时间内进行计算，对时间的要求较高，所以将起其优先级设为该应用系统中的最高，为 4。

4.2.3 中断服务子程序分析

在该系统中，除了正常运行的普通任务外，还有异步处理中断请求的中断处理子程序。中断处理子程序响应外部中断要求，仅存在一个中断源时，系统首先处理该中断处理子程序；当存在多个中断同时触发时，便有一个中断优先级选择，其响应顺序由 MC9S08GB60 中断优先级规定。MC9S08GB60 中断向量表见附录 A.4。

GB60 系统中，提供的中断源共有 26 个，其中引脚触发的中断有单片机复位中断、IRQ 中断、输入捕捉中断和键盘中断；软件中断有：SWI 软中断、低电压中断、ICG 中断、SPI 中断、SCI 中断、A/D 中断、IIC 和 RTI 中断。从附录 A.4 中可以看出，中断的优先级与硬件中断或软件中断无关，完全由单片机内部结构决定。其中，优先级最高的中断为单片机复位中断，任何时候复位，都将导致单片机重新进入监控程序的执行操作，而无需等待其他中断处理子程序的结束标志。相反，其他中断处理子程序之间发生嵌套时，必须等待当前正在执行的中断处理子程序执行完成后，才可以从后备队列中选择最高优先级的中断源，并依次顺序执行，直至最后所有中断源都得到响应。

(1) 复位中断处理子程序

单片机复位中断在任何时候都可以响应，该中断是系统唯一默认的中断，无需任何配置。该中断源也是系统内优先级最高的中断源，一旦进入，系统便重新初始化，一切均重新开始执行。因此，可以利用该功能，在特殊情况下，对单片机进行复位处理，跳过那些无法预测的外界干扰，保重系统稳定工作。

(2) 流量中断处理子程序

暖气表中使用 IRQ 引脚作为计量信号输入脚，根据外部提供的流量信号，触发 IRQ 中断处理子程序。该处理子程序中，实时记录流量信息至内存，运算后保留在内存中，并定期更新 FLASH。

计量信号的接收与处理，直接关系到水量计算及显示结构的正确性。因此，接收该信号的中断源的优先级比较高，仅次于复位中断。

(3) 键盘中断处理子程序

键盘中断是用来接收外部输入的命令，转化为系统识别的标志后，调用相应键盘响应处理任务。键盘是由若干个按键组成的开关矩阵，它是最简单的单片机的数字量输入设备，操作员通过键盘输入数据或命令，实现简单的人一机通信。键盘处理时，涉及的问题有如下三个：

•键的识别：系统中用了 4*4 的键盘，因为中断源一样，所以任一按键的中断处理子程序都一样，而从中识别出哪个键被按下的问题，便是键盘的识别。因为按键闭合后没有硬件识别，所以这里靠软件来识别按键，称为未编码按键。程序具体操作时，使用中断法。

•抖动问题：手按键时，一般会出现所按键在闭合位置与断开位置跳几下才稳定到闭合状态的情况，释放时也是如此，这便是抖动问题。抖动问题的解决办法是延时，在接收到键盘中断后，延迟扫描的时间。

•重建问题：所谓重建，就是两个或两个以上按键同时处于闭合情况的处理问题。该问题的处理方法有连锁法与巡回法。

对上述三个问题，键盘中断处理子程序中，按键识别时都有相应的处理。试验表明，该处理是必须的。

第五章 结束语

摩托罗拉公司于2003年新推出的HCS系列8位MCU,尤其是MC9S08GB60所提供的超低工作电流，具有4KB的RAM、60KB闪存，可高达20MHz的内部总线频率，性能甚至超过一些16位MCU。传统的前后台软件设计方法，大大限制了MCU性能的发挥。近年来，开放源代码的RTOS越来越多，同时实时操作系统的理论已很成熟。我们在苏州大学摩托罗拉实验室的支持下，开发基于MC9S08GB60最小系统的实时操作系统：SDF-RTOS。为使所设计的系统具有稳定性和高可靠性，在系统设计时参照了一些开放原代码的RTOS内核代码，如Small RTOS51, ECOS, uC/OS-II等，其中以美国Jean J.Labrosse先生设计和编写的“uC/OS-II (Micro C OS 2)为主。

在最小系统运行测试成功后，我们使用所设计的嵌入式实时操作系统的结构和函数，以暖气热量表为原形，设计了一个应用系统，并在硬件平台上成功运行。通过应用系统的运行已经证明了所设计的SDF-RTOS稳定和可靠的。

在使用RTOS设计应用系统的过程中，我们深深体会到采用RTOS软件设计方法的优点。不仅系统对中断的实时响应大大提高，而且由于RTOS将整个系统划分为多个任务，在编写程序和扩展功能时都很方便，只需增加相应的任务就可以。同时系统提供了结构清晰的内核和功能函数，应用系统只需直接调用就可以，缩短了程序开发时间。通过实践，我们相信RTOS一定会成为嵌入式软件设计的主要方法。

当然，在我们所设计的系统中也存在一些缺点，还有值得进一步探讨和学习的地方。如在RTOS下，如何使用降低MCU的功耗；多任务的调度策略能提供优先级占先式和时间片轮流以便在不同使用目的下选用不同的调度策略。

在软件系统设计方面，虽然所设计的RTOS已经能满足一般应用系统的需求，但由于MCU的RAM空间有限，系统资源消耗较大的功能，如文件系统管理、消息队列管理、优先级反转等功能都无法实现。这也是比较遗憾的地方。

致 谢

值此论文完成之际，首先感谢我的导师王宜怀教授。导师渊博的知识、严谨的治学作风、精益求精的工作态度、乐观的生活态度、热情待人的宝贵品质给我留下了深刻的印象，并时时影响着我、感染着我、激励着我。

苏大摩托罗拉实验室浓郁的学习研究氛围，团结务实的协作精神使我受益匪浅，特别感谢周海发、陈帅等师弟、师妹对我论文的完成所提供的帮助和支持。

我的同事和同学段陪同老师的刻苦努力、踏实严谨的学习研究精神，对我的学习、工作起到很大的促进。同时，他为我的论文提供了大力的支持，在这里谨向他们表示由衷的感谢。

感谢徐州师范大学物理系的领导和同事的关心及大力支持。

最后，衷心感谢我的爱人申文洁，尤其是我岳父、岳母，是他们克服了各种困难哺育我的儿子、操劳各种事务，同时给了我勇往直前、战胜一切困难的决心和力量。

参 考 文 献

[1]王田苗主编.嵌入式系统设计与实例开发.清华大学出版社.2003.10

[2]毛德操，胡希明著.嵌入式系统--采用公开源代码和 StrongARM/Xscale 处理器.浙江大学出版社.2003.10

[3][美]JeanJ.Labross 著，邵贝贝等译.嵌入式实时操作系统μC/OS-II（2 版）.北京航空航天大学出版社.2003.5

[4]蒋句平著.嵌入式可配置实时操作系统 eCos 开发与应用.机械工业出版社.2004.1

[5]刘光斌，刘东，姚志成编著.单片机系统实用抗干扰技术.人民邮电出版社.2003

[6]王宜怀著.单片机原理及其嵌入式应用教程.北京希望电子出版社，2002

[7]摩托罗拉 HCS08 系列 8 位微控制器. http://www.chinaecnet.com

[8]Motorola 推出 1.8V 超省电 MCU 扩大便携装置市场. http://www. eedesign-cn.com

[9]邵贝贝著.单片机嵌入式应用的在线开发方法.清华大学出版社.2004.10

[10]陈明计，周立功等编著.嵌入式实时操作系统 Small RTOS51 原理及应用.北京航空航天大学出版社.2004

[11]雷必成，吴高标.嵌入式实时操作系统μC/OS-II 的移植探讨.计算机应用.2003 年 22 卷第 5 期

[12]李强.基于 MSP430F148 单片机的嵌入式实时操作系统的研究与设计.北京工业大学硕士学位论文.2002.5

[13]邵贝贝等.嵌入式 RTOS 讲座.单片机与嵌入式系统应用. 2001 年 07 期至 12 期

[14]刘国汉，韩根亮.户用热表的原理分析与设计. 甘肃科学学报.第 14 卷第 3 期.2002 年 12 月

[15]汪 珺. 一种铂电阻的高精度温度测量系统.测试技术.2004 年 04 期

[16]李纲民 杨文波.铂电阻的A/D非线性校正.烟台大学学报(自然科学与工程版).第 16 卷第 1 期.2003 年 1 月

[17][美]Jane W.S.Liu 著.姬孟洛，李军等译.实时系统.高等教育出版社.2003.12

[18]胡伟，季晓衡编著.单片机C程序设计及应用实例.人民邮电出版社.2003.7

[19]谭浩强编著. C 程序设计.清华大学出版社.2002.4

[20]陈兴文，刘燕.单片机应用系统硬件调试技巧.现代电子技术，2000 年第 7 期

[21][美]Qing Li 著，王安生译.嵌入式系统的实时概念 .北京航空航天大学出版

社. 2004. 6

[22]赵学军. 单片机实时嵌入式操作系统微内核的设计. 桂林电子工业学院学报. 2002 年 6 月

[23]刘慧银，程建平，龚光华，王云飞，许庆丰编著. Motorola 微控制器 MC68HC08 原理及其嵌入式应用. 清华大学出版社，2001

[24]王铁勇，侯明善等. 嵌入式操作系统 μC/OS-II 的特点与应用. 控制工程 2003 年 01 期

[25]张立，桑伟. μC/OS-II 在凌阳单片机 SPCE061A 上的移植. 单片机与嵌入式系统应用. 2003 年 8 期

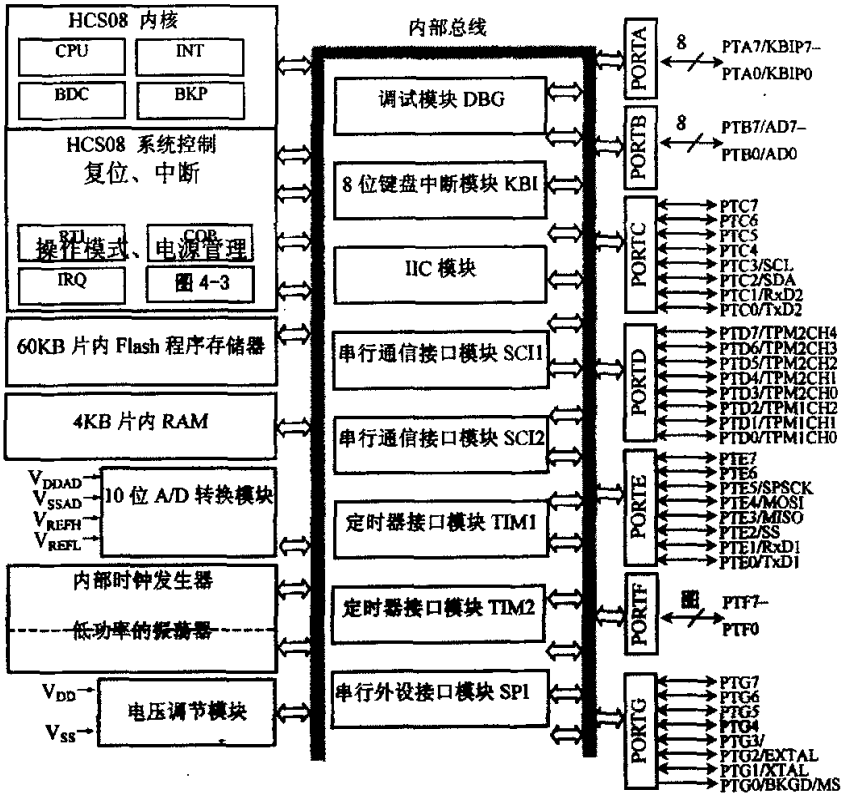
[26]袁新燕. 浅谈单片机应用系统的调试. 自动化与仪器仪表. 2000 年第 3 期

[27]Date Sheet—MC9S08GB06/D V1. Motorola Inc, 2003. 6.

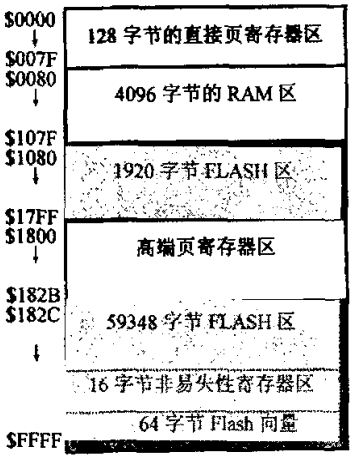
[28]HCS08 Family Reference Manaul. Volume I. Motorola Inc, 2003. 6.

附录 A MC9S08GB60 相关资料

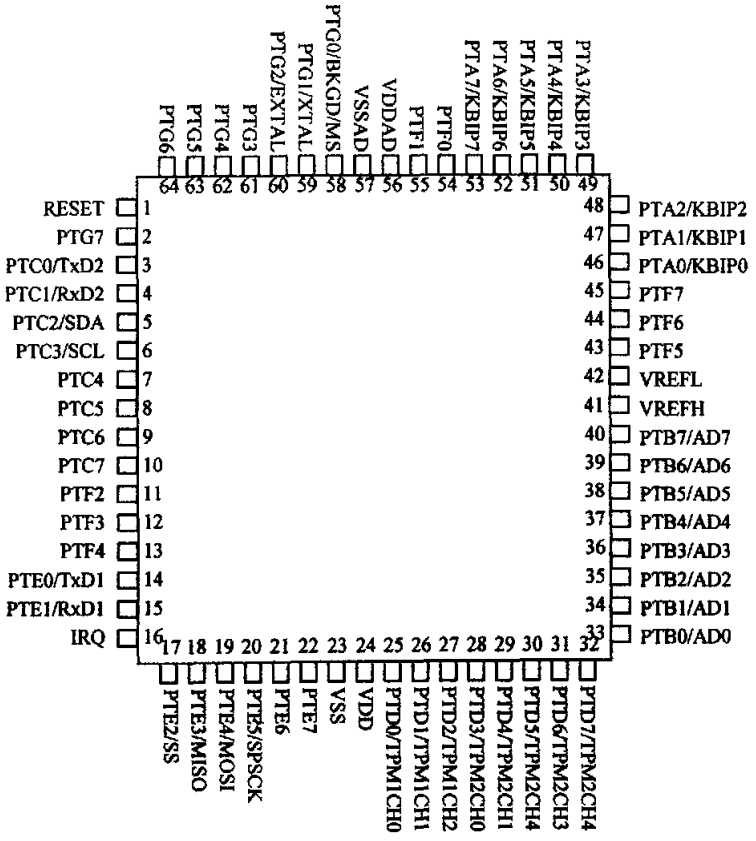
A.1 MC9S08GB60 结构框图



A.2 MC9S08GB60 存储器映像图



A.3 MC9S08GB60 引脚图



A.4 MC9S08GB60 中断向量表

中断编号	向量起始地址	中断源
0	0xFFFE	RESET
1	0xFFFC	Software Interrupt
2	0xFFFA	IRQ pin
3	0xFFF8	Low-voltage detect
4	0xFFF6	ICG
5	0xFFF4	TPM1 channel 0
6	0xFFF2	TPM1 channel 1
7	0xFFF0	TPM1 channel 2
8	0xFFEE	TPM1 overflow
9	0xFFEC	TPM1 channel 0
10	0xFFEA	TPM1 channel 1
11	0xFFE8	TPM1 channel 2
12	0xFFE6	TPM1 channel 3
13	0xFFE4	TPM1 channel 4
14	0xFFE2	TPM1 overflow
15	0xFFE0	SPI
16	0xFFDE	SCI1 error
17	0xFFDC	SCI1 receive
18	0xFFDA	SCI1 transmit
19	0xFFD8	SCI2 error
20	0xFFD6	SCI2 receive
21	0xFFD4	SCI2 transmit
22	0xFFD2	Keyboard pins
23	0xFFD0	AD conversion complete
24	0xFFCE	IIC control
25	0xFFCC	Real-time interrupt
26~31	0xFFCA	Unused Vector Space

附录 B 暖气表电路原理图