北京机械工业学院

硕士学位论文

# 基于CygnalF040单片机的RTOS51的研究与开发

姓名：辛亮

申请学位级别：硕士

专业：控制理论与控制工程

指导教师：王东兴

20070301

# 摘要

随着实时操作系统应用的日益广泛，其自身结构的设计也变得更易于移植，而用户的应用程序均需运行于RTOS之上，RTOS根据各个任务的要求,进行分配CPU时间和系统资源的管理，同时还要进行调度的优化，以保证实时性和可靠性。靠用户自己编写的实现上述功能的内核一般是不现实的，而这种需求又是相当普遍的。在这种形势之下，高性能实时操作系统（RTOS）内核的应用就成为一种必然的发展趋势。

实时操作系统是任何必须在指定的有限时间内给出响应的系统。本文以电力系统故障诊断终端的基本构件为应用背景，论述了以C8051F040单片机为核心的硬件平台的设计和在此平台下的基于实时操作系统Small RTOS51的软件设计方法。首先介绍了关于实时操作系统的几个基本概念，即：嵌入式系统、嵌入式操作系统、实时操作系统和嵌入式实时操作系统，并且结合嵌入式系统的发展，说明嵌入式系统的应用价值。

通过对μC/OS-II及Small RTOS51这2种实时内核在内核RAM需求、任务调度算法和异常处理等工作的对比，选择了Small RTOS51作为软件开发平台，然后通过前/后台系统和实时内核的对比，清晰地说明了实时内核平台下关键任务的响应时间的可预测性。接下来介绍了Small RTOS51提供的典型的系统服务，重点介绍任务调度、信号量、中断、消息队列的概念及典型应用。

在硬件设计部分，针对具体的应用背景，选择Silicon Lab公司的C8051F040单片机作为控制器。软件设计部分，本文以电力系统故障诊断终端的基本构件为应用背景，论述了在以C8051F040单片机为核心的硬件平台基础上移植实时操作系统Small RTOS51的软件方法，首先简要介绍了实时系统软件开发的基本概念、目标和方法，在此基础上进行任务接口定义的思想设计了下列驱动函数：键盘、LCD、实时时钟、模拟量输入/输出、开关量的输入输出和CAN总线通信协议。

最后，对全文进行概要的总结。

关键词：实时操作系统；Small RTOS51；C8051F040；CAN 总线

# ABSTRACT

with applying real time operation system reality gradually broad, whose oneself architectural design also become be easier transplanting, consumer's application needs without exception working in of RTOS, the RTOS request according to each missions, carries out the administration assigning CPU time and system resource , want to carry out the optimization managing too at the same time, with guarantee real time nature and reliability. Self write realization is unrealistic like as said or narrated above functional core, but this need is pretty common by the consumer. Under this circumstances, high-performance real time operation system (RTOS) core application becomes one kind of inevitable developing trend right away.

System real time operation system must be given to out within the limited time being appointed as respond to as being any. Hardware platform centering on C8051F040 monolithic machine design and lower real time operation system Small RTOS51-based here platform software design procedures the main body of a book has been discussed with electric system malfunction diagnose terminal basic component for the background applying. First, have introduced several basic concepts about real time operation system , have been: And implant dyadic system , implant style OS , real time OS and implant dyadic real time operation system, be tied in wedlock implanting development of dyadic system, show implanting dyadic systematic application value.

Pass to the micro μC/OS-II and Small RTOS51 examine the contrasts working such as RAM need , task scheduling algorithm and anomaly treatment including this 2 kinds real time cores , RTOS51 having chosen Small develops platform as the software , the contrast by former backstage supporter/ system and the real time core , the platform having explained the real time core limpidly issue and then key mission REST predictability. Have introduced the representative system service that Small RTOS51 provides then, the concept and model concentrating on introducing the task scheduling , signal amount , interruption , information queue apply.

C8051F040 monolithic machine designing a part , being aimed at the company who applies background , chooses Silicon Lab concretely in hardware is a controller. The software has designed that the part , the main body of a book take electric system malfunction diagnose terminal basic component as the background applying , has

discussed software method transplanting real time operation system Small RTOS51 on the basis in hardware platform centering on C8051F040 monolithic machine, the thought that first brief the development basic concept having introduced real time system software , the interface carrying out a mission on the target and method , here basis define has designed the following drive function: Keyboard , LCD , reality from time to time

In the end, summary of the essentials have been carried on to the full paper.

**Key Words:** real time operating systems; Small RTOS51; C8051F040; CAN bus

# 学位论文版权使用授权书

本人完全了解北京机械工业学院关于收集、保存、使用学位论文的规定，同意如下各项内容：按照学校要求提交学位论文的印刷本和电子版本；学校有权保存学位论文的印刷本和电子版，并采用影印、缩印、扫描、数字化或其它手段保存论文；学校有权提供目录检索以及提供本学位论文全文或者部分的阅览服务；学校有权按有关规定向国家有关部门或者机构送交论文的复印件和电子版；在不以赢利为目的的前提下，学校可以适当复制论文的部分或全部内容用于学术活动。

学位论文作者签名：郭亮

2007 年 3月23日

---

经指导教师同意，本学位论文属于保密，在　　年解密后适用本授权书。

指导教师签名：　　学位论文作者签名：

年　月　日　　年　月　日

# 硕士学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，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，本学位论文的研究成果不包含任何他人创作的、已公开发表或者没有公开发表的作品的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

签名：

2007 年 3 月

# 第1章 引言

## 1.1 概述

### 1.1.1 嵌入式系统及实时系统[1][2]

嵌入式系统被定义为：以应用为中心,以计算机技术为基础,软件、硬件可裁减，适应应用系统对功能、可靠性、成本、体积和功耗严格要求的专用计算机系统。嵌入式系统是将应用程序和操作系统与处理器硬件集成在一起的系统。它以应用为中心，软硬件可裁减，能够适应应用系统对功能、可靠性、成本、体积、功耗等综合性严格要求。

实时系统是任何必须在指定的有限时间内给出相应的系统。在这种系统中，时间起到重要的作用，系统成功与否不仅要看是否输出了逻辑上正确的结果，而且要看他是否在指定时间内给出了这个结果。

按照对时间要求的严格程度，实时系统被划分为硬实时（hard real time）、固实时（firm real time）和软实时（soft real time）。硬实时系统是指系统响应绝对要求在指定时间范围内；软实时系统中即使相应也重要，但是偶尔响应慢了亦可接受；而在固实时系统中，不能及时相应会造成服务质量的下降。

### 1.1.2 前、后台系统[3]

对于前、后台系统来说，应用程序一般是一个无限的循环，可称为超循环系统。循环中调用相应的函数完成相应的操作，这部分可以看成后台行为。中断服务程序处理异步时间，这部分可以看成前台行为。后台也可以叫做任务级，前台也叫中断级，时间相关性很强的关键操作一定是靠中断服务程序来保证的。因为中断服务提供的信息一直要等到后台程序走到该处理这个信息这一步时才能得到进一步处理，所以这种系统在处理的及时性上比实际可以做到的要差。这个指标称作任务级响应时间。最坏情况下的任务级响应时间取决于整个循环的执行时间。因为循环的执行时间不是常数，程序经过某一特定部分的准确时

间也不能确定。进而，如果程序修改了，循环的时序也会受到影响。

很多基于微处理器的产品采用前、后台系统设计，例如微波炉、电话机和玩具等。在另外一些基于微处理器的应用中，从省电的角度出发，平时微处理器处在停机状态，所有工作都靠中断服务来完成。

### 1.1.3 实时操作系统 [3][4]

实时操作系统（Real Time Operation System ，简称 RTOS）是一段在嵌入实系统启动后首先执行的背景程序。用户的应用程序是运行于 RTOS 之上的各个任务，RTOS 根据各个任务的要求，进行资源（包括存储器、外设等）管理、消息管理、任务调度和异常处理等工作。在 RTOS 支持的系统中，每个任务都有一个优先级，RTOS 根据各个任务的优先级，动态地切换各个任务，保证对实时性的要求。而任务之间的切换应当以优先级为根据，只有具有优先服务方式的 RTOS 才是真正的实时操作系统，而时间分片方式和协作方式的 RTOS 并不是真正的“实时”。

通常设计接触的通用操作系统（如 Windows、Linux 等）都是分时操作系统。实时操作系统能及时响应外部事件的请求，在规定的时间内完成对该事件的处理，并控制所有实时任务协调一致地运行。与分时操作系统相比，具有多路性、独立性、及时性、交互性、可靠性的特点。

实时操作系统所遵循的最重要的设计原则是：采用各种算法和策略，始终保证系统行为的可预测性（predictability）。可预测性是指在系统运行的任何时刻，实时操作系统的资源调配策略都能为争夺资源（CPU、内存等）的多个实时任务合理地分配资源，使每个实时任务的实时性要求都能得到满足。

## 1.2 实时操作系统的发展 [1][2]

随着互联网应用的迅速发展，嵌入式微处理器的应用日益广泛，无处不在，从波音飞机到移动电话，都有嵌入式微处理器的存在。在嵌入式微处理器的应用开发中，嵌入式实时操作系统（简称 RTOS）是核心软件，就像我们日常所用计算机的桌面系统中，微软公司的 Windows 98 一样重要。RTOS 已经在全球形成了 1 个产业，据美国 EMF(电子市场分析)报告，1999 年全球 RTOS 市场产值

达 3.6 亿美圆，而相关的整个嵌入式开发工具（包括仿真器、逻辑分析仪、软件编译器和调试器）则高达 9 亿美圆。

一、 RTOS 的发展历史

从 1981 年 Ready System 发展了世界上第 1 个商业嵌入式实时内核（VRTX32），到今天已经有近 20 年的历史。20 世纪 80 年代的产品还只支持一些 16 位的微处理器，如 68k,8086 等。这时候的 RTOS 还只有内核，以销售二进制代码为主。当时的产品除 VRTX 外，还有 IPI 公司的 MTOS 和 80 年代末 ISI 公司的 PSOS。产品主要用于军事和电信设备。进入 20 世纪 90 年代，现代操作系统的设计思想，如微内核设计技术和模块化设计思想，开始渗入 RTOS 领域。老牌的 RTOS 厂家如 Ready System（在 1995 年与 Microtec Research 合并），也推出新一代的 VRTXsa 实时内核，新一代的 RTOS 厂家 Windriver 推出了 Vxwork。另外在这个时期，各家公司都有力求摆脱完全依赖第三方工具的制约，而通过自己收购、授权或使用免费工具链的方式，组成 1 套完整的开发环境。例如，ISI 公司的 Prismt、著名的 Tornado(Windriver)和老牌的 Spectra(VRTX 开发系统)等。

进入 20 世纪 90 年代中期，互联网之风在北美日渐风行。网络设备制造商、终端产品制造商都要求 RTOS 有网络和图形界面的功能。为了方便使用大量现存的软件代码，他们希望 RTOS 厂家都支持标准的 API，如 POSIX， Win32 等，并希望 RTOS 的开发环境与他们已经熟悉的 UNIX，Windows 一致。这个时期代表性的产品有 Vxwork，QNX, Lynx 和 WinCE 等。

二、 RTOS 市场和技术的发展变化

可以看出，进入 20 世纪 90 年代后，RTOS 在嵌入式系统设计中的主导地位已经确定，越来越多的工程师使用 RTOS，更多的新用户愿意选择购买而不是自己开发。我们注意到，RTOS 的技术发展有以下一些变化：

1. 因为新的处理器越来越多，RTOS 自身结构的设计更易于移植，以便在短时间内支持更多种微处理器。

2. 开放源码之风已波及 RTOS 厂家。数量相当多的 RTOS 厂家出售 RTOS 时，就附加了源程序代码并含生产版税。

3. 后 PC 时代更多的产品使用 RTOS，它们对实时性要求并不高，如手持设备等。微软公司的 WinCE,Plam OS, Java OS 等 RTOS 产品就是顺应这些应用而开发出来的。

4. 电信设备、控制系统要求的高可靠性，对ＲＴＯＳ提出了新的要求。瑞典 Enea 公司的ＯＳＥ和 WindRiver 新推出的 Vxwork ＡＥ对支持ＨＡ（高可用性）和热切换等特点都下了一番功夫。

5. Windriver 收购了ＩＳＩ，在 RTOS 市场形成了相当程度的垄断，但是由于 Windriver 决定放弃 PSOS，转为开发 Vxwork 与 PSOS 合二为一版本，这便使得 PSOS 用户再一次走到重新选择 RTOS 的路口，给了其他 RTOS 厂家 1 次机会。

6. 嵌入式 Linux 已经在消费电子设备中得到应用。韩国和日本的一些企业都推出了基于嵌入式 Linux 的手持设备。嵌入式 Linux 得到了相当广泛的半导体厂商的支持和投资，如 Intel 和 Motorola。

总体来说，第一阶段是嵌入式技术的早期阶段，以功能简单的专用计算机或单片机为核心的可编程控制器形式存在，具有监测、伺服、设备指示等功能。这一阶段系统的主要特点是：系统结构和功能都相对单一，处理效率较低，存储容量较小，几乎没有用户接口，比较适合于各类专用领域中。第二阶段是以嵌入式 CPU 和嵌入式操作系统为标志的嵌入式系统。这一阶段系统的主要特点是：计算机硬件出现了高可靠、低功耗的嵌入式 CPU，如 Power PC 等，各类商业嵌入式操作系统开始出现并得到迅速发展，嵌入式操作系统能运行于各种不同类型的微处理器上，兼容性好；操作系统内核精小、效率高，并且具有高度的模块化和扩展性；具备文件和目录管理、设备支持、多任务、网络支持、图形窗口以及用户界面等功能；具有大量的应用程序接口(API)，开发应用程序简单；嵌入式应用软件丰富。第三阶段是以芯片技术和 Internet 技术为标志的嵌入式系统。随着微电子技术的迅速发展，SOC(片上系统 )使嵌入式系统越来越小，功能却越来越强大。

## 1.3 研究实时操作系统 Small RTOS 的目的和意义 [5]

提倡在嵌入式应用中使用 RTOS 的最主要原因是提高系统的可靠性，使系统的实时性可以达到理论上可以达到的最优值。在嵌入式应用中，尤其是控制系统，不“死机”是最起码的要求。一方面，在系统的硬件设计上尽量提高抗干扰能力，满足电磁兼容性的要求；另一方面就是在软件上采取措施。传统的嵌入式软件都是前后台系统，整个结构是个大循环，在遇到强干扰时，程序在

任何一处产生死循环或破坏都会引起死机，只能靠看门狗复位。而对于 RTOS 管理的系统，不仅可以将应用程序分解成若干独立的任务，而且可以另外启动一个监控任务，监视所有任务的运行状况。

提倡在嵌入式应用中使用 RTOS 的另一个原因是提高开发效率。应用程序被分解成多个任务，应用程序开发变得更加容易，便于维护，易读易懂，提高了开发效率，缩短了开发周期。当然，使用实时内核需要增加额外 ROM/RAM 的开销以及 2%～4%的 CPU 负荷，而且实时内核本身会增加产品的成本。

## 1.4 选题背景

目前，我国对于大部分供电电力设备的监控和管理上基本还停留在人工巡查这种比较原始的管理方式下（例如铁路道口的信号电源），不但管理效率低下，而且供电质量和水平都得不到有效的保证。本文以实时操作系统 small RTOS51 为基础，以电力系统故障诊断终端的基本构件为应用背景，论述了在以 C8051F040 单片机为核心的硬件平台基础上移植实时操作系统 small RTOS51 的软件方法，主要探讨了一个用于电力系统现场监控终端单元的设计过程，对提高配电自动化水平有着重要的现实意义和经济价值。希望能够为提高配电自动化水平提供一些思路，尽一点微薄之力。

SOC 技术是当今超大规模集成电路（VLSI）的发展趋势，由于 SOC 具有片上资源丰富、高性能、低功耗、低成本的特点，已经成为当今集成电路技术应用的主流。鉴于上述嵌入式 RTOS 的优点，通过嵌入式 RTOS 来管理 SOC 内部的硬件资源，研究嵌入式 RTOS 平台下软件开发方法具有实际应用价值。

本文以 Silicon Lab 公司的 C8051F040 单片机为对象，重点研究在 Small RTOS51 平台下开发实时操作系统软件的方法。全文共分 7 章。

第 1 章为引言。

第 2 章为总体方案设计及可行性分析，重点分析和评价了 RTOS 的选型。

第 3 章介绍实时内核平台下应用软件的设计方法，通过前/后台系统和实时内核的对比，说明实时内核管理下可以使任务响应的时间达到最优化，介绍了任务调度、信号量、消息队列的概念及典型应用。

第 4 章为系统的硬件设计部分，简要介绍了以铁路终端测控系统单元为参照对象，设计了以 C8051F040 单片机为核心的硬件平台。

第 5 章以任务调度为核心，介绍了 Small RTOS51 在 C8051F040 单片机上的移植过程，编写 OS_CPU_C.C、OS_CPU.H、OS_CPU_A.ASM 这几个文件，同时给出测试代码。

第 6 章在需求分析的基础上，以 DARTS 设计方法为模型，设计系统的应用软件及相应模块的驱动程序设计过程，重点考虑数据流程和接口的设计。

第 7 章为全文结论。

## 1.5 本课题研究的内容

在本系统设计中，有如下要求：

1. 内核移植，任务调谐。
2. 结合主板资源开发应用接口函数。
3. 编写 Small RTOS51 程序并保证能够正常运行。

系统实现的功能为：

1. 实现单相电压、电流的测量及断电纪录。
2. 设备开关状态的测量，变位信号等。
3. 对断路器的分合控制。
4. CAN 总线通信，能够上传数据。

# 第 2 章　总体方案设计及可行性分析

系统设计的目标是实现具有 CAN 总线通信功能的实时操作系统设计,并编写相应的驱动函数。

## 2.1　系统主要功能要求及技术指标

系统的功能要求及技术指标如下：

（1）系统软件要求建立在 RTOS 的平台下；

（2）模拟量输入、输出的精度要优于 0.5%；

（3）实现 CAN 总线通信的功能；

（4）工作温度满足 0～70℃；

（5）具有越限报警功能；

## 2.2　系统开发的主要技术难点

系统开发的主要技术难点有 3 个。第一，RTOS 的选型及移植；第二，实时 CAN 总线通信功能的实现；第三，各功能模块驱动程序的编写。

## 2.3　系统总体方案设计的可行性分析

系统的硬件设计不是太复杂，按照典型的铁路智能终端单元的设计方案，只要保证模拟量的精度、预留足够的 I/O 点即可。系统设计主要解决上述 3 个大问题。

### 2.3.1　RTOS 的选型 [1][6]

近 10 年来，嵌入式实时操作系统得到飞速的发展，从支持 8 位微控制器到 16 位、32 位、甚至 64 位，从支持单--品种的微处理器芯片到支持多品种微处理器芯片，从只有实时内核到除了内核外还提供其他功能模块，如：高速

文件系统、TCP/ IP 网络系统、GUI 系统等。据嵌入式系统杂志( Embedded Systems Programming) 的最新报道。世界各国已有 40 多家公司成功地推出了 200 余种可供嵌入式应用的实时操作系统。其中几个著名的操作系统是：Microtec Research 公司(MRI) 的 VRTX，Integrated System 公司( ISI) 的 pSOS，Wind River 公司的 VxWorks，QNX 系统软件公司的 QNX，ATI 的 Nucleus，以及 RTLinux 和 μC/OS-II 等。

下面就 RTLinux、VxWorks、μC/OS-II 和 Small RTOS51 这 3 种典型的操作系统为例，着重从内核结构、调度算法和开发环境几方面做简单的介绍。

#### 2.3.1.1 RTLinux[1][6]

RTLinux 是在 Linux 操作系统的低层中加载一个非常精简的 RT-Kernel，处理实时任务，而原有的内核(Linux kernel)在运行时可以看作 RT-Kernel 的任务，而且相当于专用 RTOS 中优先级最低的任务。美国新墨西哥理工学院的开发小组致力于 RTLinux 的开发工作。

（1）内核结构。RTLinux 的系统结构中，RT-kernel 对硬件中断进行处理，并用软件模拟中断控制器，对于编程语言，Linux 内核对软件中断控制器的操作就如同原来对硬件中断一样。实时任务不同于 Linux 的进程，缩短了上下文切换的时间；不使用虚拟内存；RT-kernel 本身是不可抢占的，但是由于非常短小精悍，造成的延时比较短。

（2）调度算法。实时任务的调度方式有 2 种：周期性调度与中断唤醒。RTLinux 带一个纯优先级的调度器，用户可以根据需要编写自己的调度器。

（3）开发环境。现阶段一般沿用 GCC、GDB 等工具，缺乏实时专用的编译与调试工具。

RTLinux 兼容 POSIX1003.1b 规范。

#### 2.3.1.2 VxWorks[1][6]

（1）内核结构。VxWorks 是专门为嵌入式微处理器设计的高模块化、高性能的实时操作系统，将一些基本功能封装在内核中，其它功能由用户级服务进程实现。

（2）调度算法。VxWorks 的任务具有高度可抢占性（preemptive），设有 256 级任务优先级，任务切换延迟不随任务数目的增加而变化，为了防止抢占带

来的优先级反转，采用了优先级继承机制。

（3）开发环境。开发调试采用宿主机/目标机的方式，有完善的主机端编程、调试、代码管理工具（Tornado）。其中包括驻留目标机的调试通信代理，主机目标机仿真，可调试系统级程序的远程调试工具，图形化的源代码编辑器和项目管理工具等。

VxWorks 广泛应用于高科技产品中，包括消费电子设备、工业自动化、无线通信产品、医疗仪器、数字电视与多媒体设备，具有很好的安全性、容错性以及系统灵活性。虽然它们都提供专有的接口，但是缺乏应用的高效性，网络连接功能较差，系统 API 对应用程序开发支持相对较弱。

#### 2.3.1.3 μC/OS-II[5]

μC/OS-II 是著名的源代码公开的实时内核，是专为嵌入式应用设计的，可用于绝大多数 8 位、16 位和 32 位单片机或 DSP。已经有了近十年的使用实践，有许多成功应用该实时内核的实例，但是在小 RAM 系统下（如 51 系统）缺乏应用的灵活性，占用空间较多，不能很好的满足实际需要。

它的主要特点如下：（1）有源代码。（2）可移植性（portable）。μC/OS-II 源码绝大部分是用移植性很强的 ANSI C 写的，与微处理器硬件相关的部分是用汇编语言写的，便于移植到其它的微处理器上。（3）可固化（ROMable）。（4）可裁减（scalable）。可以只使用应用程序需要的系统服务。（5）占先式性（preemptive）。总是运行就绪条件下优先级最高的任务。（6）可确定性。系统任务的执行时间不依赖于用户应用程序任务数目的多少。（7）任务堆栈。每个任务都有自己独立的任务堆栈，以便降低应用程序对 RAM 的需求。（8）系统服务。μC/OS-II 提供丰富的系统服务，例如信号量、互斥型信号量、事件标志、消息邮箱、消息队列、内存管理等功能。（9）稳定性与可靠性。

#### 2.3.1.4 Small RTOS51

Small RTOS51 具有以下特点：

（1） 公开源代码。只要遵循许可协议，任何人可以免费获得源代码。

（2） 可移植。Small RTOS51 把与 CPU 相关部分压缩到最小，与 CPU 无关部分用 ANSI C 编写。

（3） 可固化。Small RTOS51 为嵌入式系统设计，如果有固化手段，可以

嵌入到产品中成为产品的一部分。

（4） 占先式。Small RTOS51 可以管理 16 个用户任务，每个任务优先级不同。Small RTOS51 总是运行就绪条件下优先级最高的任务。

（5） 中断管理。中断可以使正在执行的任务挂起。如果优先级更高的任务被中断唤醒，则高优先级的任务在中断嵌套全部退出后立即执行。中断嵌套层数可达 255 层。如果需要，可以禁止中断嵌套管理。

（6） RAM 需求小。Small RTOS51 为小 RAM 系统设计，因而 RAM 需求小，相应的系统服务也少。Small RTOS51 可以在没有任何外部数据存储器的单片 80C51 系统上运行，但应用程序仍然可以访问外部存储器。

在实际开发过程中，Small RTOS51 源代码以应用程序头文件的方式被引用，和应用程序在 IDE 环境中同时编译。

#### 2.3.1.5 总结

由于随着嵌入式系统的发展，使得嵌入式软件设计向软件平台靠近，单片机软件设计不再是单一线程结构方式，而是逐步采用多任务的设计思想。实时内核也称为实时操作系统或 RTOS。使用它使得实时应用程序的设计、扩展和维护变得更容易。不需要大的改动就可以增加新的功能。通过应用程序分割为若干独立的任务，RTOS 使得应用程序的设计过程大为简化。使用可剥夺性的内核时，所有时间要求苛刻的事件都得到了尽可能快捷、有效的处理。通过有效的服务：如信号量、邮箱、队列、延时、超时等；RTOS 使得资源得到更好的利用。

目前 8 位单片机在嵌入式系统中，应用还是最广泛的，51 多任务系统中，keil c51 所带的 RTX Full 太大（6k 多），且需要外部 ram，又无源代码，很多时候不实用。RTX Tiny 虽然小（900 多字节），但是任务没有优先级和中断管理，也无源代码，也不太实用；对于 RTLinux，VxWorks 也不适用；而 μC/OS-II 虽有源代码，但是它太大，又需要外部 ram，所有函数又必须是重入函数，用在 51 这类小片内 RAM 的单片机上有点勉强。

而对于 Small RTOS51 来说，Small RTOS51 正是为小 RAM 系统设计，RAM 需求比上述几种实时内核要小得多，非常适用于 51 这种资源比较少的系统上。故在本设计中选用了 Small RTOS51 实时内核。

### 2.3.2 RTOS 的移植 [5]

所谓移植，就是使一个实时内核能在其它的微处理器或微控制器上运行。要保证移植成功，处理的选型非常关键，同时也需要熟练编译器、理解操作系统基本概念。

#### 2.3.2.1 RTOS 移植对处理器的基本要求 [5]

Small RTOS51 为小 RAM 系统设计，对 RAM 需求小，相应的系统服务也少。运行 Small RTOS51 必须有一个基于 51 系列单片机的 C 语言编译器。如果使用这些代码，则需要 Keil C51 编译器。当不使用消息队列时，需要 Keil C51 V6.14 以上版本；当使用消息队列是，需要 Keil C51 V6.23 以上版本；其次，必须有一个完全兼容 51 的单片机。

Small RTOS51 可以在没有任何外部数据存储器的单片机上运行，但应用程序仍然可以访问外部存储器。Small RTOS51 可以使用 C51 支持的全部存储器模块，选择记忆模型仅影响应用目标的位置。一般来说，Small RTOS51 应用程序工作在校模式下。Small RTOS51 没有按照 bank switching 程序设计，不能使用 code banking 程序。

为了节省 RAM，Small RTOS51 把所有自由内部 RAM 分配给当前任务。

当任务通过调用系统函数进行任务切换时，保存任务环境需要的 RAM 空间字节数为：

3 + 到 OSSCHED 时的调用层数×2

当任务通过中断进行任务切换时，保存任务环境需要的 RAM 空间字节数为：

中断发生时堆栈使用量 + 15

这 15 个字节包括 R0~R7、ACC、B、PSW、DPH、DPL 和变量 Os_Enter_Sum。

#### 2.3.2.2 RTOS 移植需要的知识准备和技术支持

要保证 Small RTOS51 的移植顺利进行，还需要一定的知识准备和技术支持。

（1）对操作系统基本概念的掌握。从前没有操作系统应用的经验，需要对操作系统的基本概念和基本原理有概念上的了解，可以通过参考相关的教材来弥补这方面的知识。

（2）Small RTOS51 源代码的熟练程度。虽然从前没有 Small RTOS51 的应

用经验，参考文献 1 对 Small RTOS51 的源码进行详细的分析和描述，用 2 个月左右的时间应该能对 Small RTOS51 达到一个初步应用的水平。

（3）对编译器的熟练程度。从前有过 Keil、CodeWarrior、ICC 编译器的应用经验，可以作为借鉴，即使选择新的编译器，应该很快熟练。

（4）C 语言和汇编语言混合编程的掌握。这一点要求主要取决于编译器的选择，如果编译器不支持插入行汇编代码，就需要 C 语言和汇编语言混合编程。所以在选择的编译器最好支持插入行汇编代码，否则就需要对选定控制器汇编语言非常的熟练。

（5）技术支持。如果选择没有使用过的控制器和编译器，在最初开发过程中就会有一些技术问题，所以一定要选择有成功的开发范例可借鉴的或者是有能力提供技术支持的控制器和编译器。

### 2.3.3 CAN 总线实时通信功能的实现

总线通信方式采用 CAN 总线通信方式。之所以采用 CAN 通信网，这是因为传统的电力系统监控系统中所使用的控制网，多为 RS-485 网或 BITBUS 网，但它们在应用中存在以下缺陷[22]：

- BITBUS 和 RS-485 网络上只能有一个主节点，无法构成多机冗余系统，导致系统可靠性变差；
- 数据通信方式是命令响应式，从节点只有在收到主节点的命令后才能响应，致使一些重要的变位信息得不到及时上送，系统灵活性差，实时性差；
- BITBUS 和 RS-485 网络纠错能力差。

CAN 总线是一个造价低廉而又适用于电力系统监控的控制网。将 CAN 总线用于电力系统，有着 BITBUS 和 RS-485 网无法比拟的优势，并能从根本上改变电力监控系统的性能，CAN 总线主要具有以下优点[1]：

- CAN 是到目前为止惟一有国际标准的现场总线。
- CAN 为多主方式工作，网络上任一节点均可在任意时刻主动地向网络上其它节点发送信息，而不分主从。
- 在报文标识符上，CAN 上的节点分成不同的优先级，可满足不同的实时要求，优先级高的数据最多可在 134μs 得到传输。
- CAN 采用非破坏性仲裁技术。当多个节点同时向总线发送信息出现冲突

时，优先级较低的节点主动地退出发送，而最高优先级的节点可不受影响地继续传输数据，从而大大节省了总线冲突仲裁时间。尤其是在网络负担很重的情况下，也不会出现网络瘫痪情况。

- CAN 节点只需通过对报文的标识符滤波即可实现点对点、一点对多点及全局广播等几种方式传送接收数据。
- CAN 的直接通信距离最远可达 10km（速率 5kbps 以下）；通信速率最高可达 1Mbps（此时的通信距离最长为 40m）。
- 报文采用短帧结构，传输时间短，受干扰概率低，保证了数据出错率极低。
- CAN 的每帧信息都有 CRC 校验及其他检错措施，具有极好的检错效果。
- CAN 的通信介质可为双绞线，同轴电缆或光纤，选择灵活。
- CAN 节点在错误严重的情况下具有自动关闭输出功能，以使总线上其他节点的操作不受影响。

## 2.4 本章小结

综上所述，硬件平台的设计属于比较典型的应用，完全能够实现，主要是微控制器的选型要慎重，否则会影响进度。软件方面主要包括对 Small RTOS51 的认识过程、Small RTOS51 的移植和 RTOS 平台下的软件实现，只要前期知识准备充分，也应该能够实现。对于 CAN 总线控制器，最好选择能提供详细使用手册、对应的驱动程序及应用范例的型号。

# 第 3 章　实时内核平台下软件的设计方法

## 3.1　前、后台系统软件的缺陷 [5]

传统的监控系统软件设计多采用如图 3-1 的结构，这种结构可称为前/后台系统或者超循环系统。

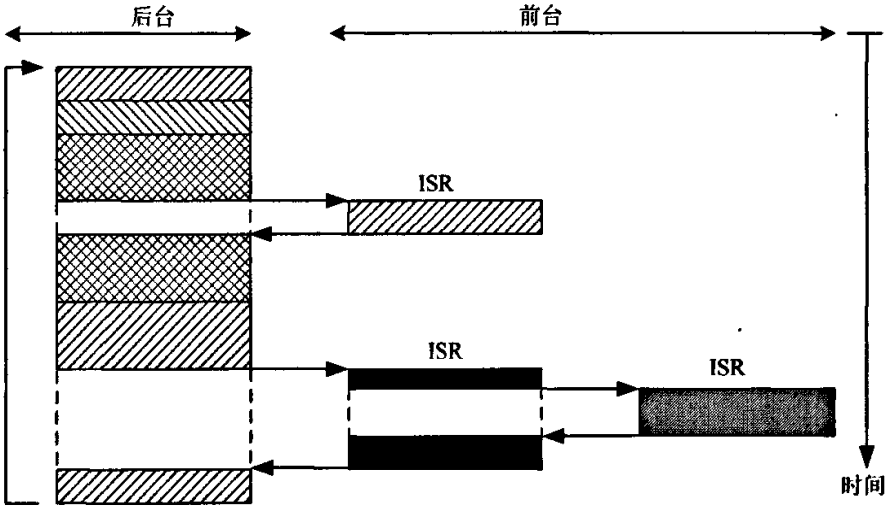


图 3-1　前/后台系统

应用程序是一个无限循环，循环中调用相应的函数完成相应的操作，这部分可以看成后台行为；而前台的中断服务程序（ISR）负责处理异步事件，即中断。为了保证关键操作的实时性，就必须在 ISR 中处理，但这样做又会增加 ISR 的执行时间。而一个 ISR 的时间应该非常短，只是简单地从中断设备读取参数和状态信息，而将参数和状态信息的处理交给任务去完成。这些参数和状态信息只能等到后台循环体执行到负责处理它们的模块时才能被处理，这段等待时间称为任务响应时间。最大任务响应时间由整个循环体的执行时间决定，但循环体的执行时间并不固定，因此任务响应时间也不固定。并且，当程序代码发生变化时，循环体的执行时间也会相应地改变。

## 3.2 实时内核 [5]

当关键操作所限定的时间大于循环体最大执行时间时，前/ 后台系统才能够正常工作，但通常循环体最大执行时间很难确定。而且采用前/ 后台软件系统，要在系统中增加新功能非常困难。一种更好的软件机制是采用RTOS内核。RTOS内核是实时操作系统的核心部分，完成操作系统的基本功能，也可以说是实时操作系统的最小子集。它负责CPU 的时间分配，以确保所有的实时任务都能被及时处理。采用实时操作系统可以把一项工作划分为多个任务，按任务的重要性为其分配优先级，每个任务完成简单的功能，并认为自己独占CPU。此时CPU总的工作量并无改变，但其各项工作具有了优先级，而且其执行顺序可以进行调配。RTOS还给应用程序提供了信号量管理、邮箱、消息队列以及时间延时等系统服务，提高了CPU的工作效率。实时内核分为不占先式内核和占先式内核。

### 3.2.1 非占先式内核 [5]

非占先式内核要求每个任务自我放弃CPU的所有权。非占先式调度法也称作合作性多任务，各个任务彼此合作共享一个 CPU。异步事件还是由中断服务来处理。中断服务可以使一个高优先级的任务由挂起状态变为就绪状态。但中断服务以后控制权还是回到原来被中断了的那个任务，直到该任务主动放弃CPU的使用权时，那个高优先级的任务才能获得CPU的使用权。

在任务及，非占先式内核允许使用不可重入函数。每个任务都可以调用非可重入性函数，而不必担心其他任务可能正在使用该函数，从而造成数据的破坏，因为每个任务要运行到完成时才释放CPU的控制权。当然该部可充入性函数本身不得有放弃CPU控制权的企图。

使用非占先式内核时，任务级响应时间比前、后台系统快得多。此时的任务级响应时间取决于最长的任务执行时间。

非占先式内核的另一个优点是，几乎不需要使用信号量保护共享数据。运行着的任务占有 CPU，而不担心被别的任务抢占。但这也不是绝对的，在某种情况下，信号量还是用得着的。处理共享I/O设备时仍需要使用互斥型信号量。

非占先式内核最大的缺陷在于其相应时间。高优先级任务已进入就绪态，但还不能运行，也许要等很长时间，直到当前运行着的任务释放 CPU。与前、后台系统一样，非占先式内核的任务级响应时间是不确定的，不知道什么时候

最高优先级的任务才能拿到 CPU 的控制权。这完全取决于应用程序什么时候释放 CPU。

总之，非占先式内核允许每个任务运行，直到该任务自愿放弃 CPU 的使用权。中断可以打入运行着的任务，中断服务完成以后将 CPU 控制权还给被中断了的任务。任务级响应时间要大大好于前、后台系统。

### 3.2.2 占先式内核 [5]

当系统响应时间很重要时，要使用占先式内核。Small RTOS 属于占先式内核。占先式内核总能使就绪态的最高优先级任务得到 CPU 的使用权。当一个正在运行着的任务使一个比它优先级高的任务进入就绪态时，当前任务的 CPU 的使用权就被剥夺了，或者说被挂起了，更高优先级的任务立刻得到了 CPU 的使用权。如果是中断服务子程序使一个高优先级的任务进入就绪态，中断完成时，被中断的任务被挂起，优先级高的任务开始运行。占先式内核的执行过程如图 3-2 所示。

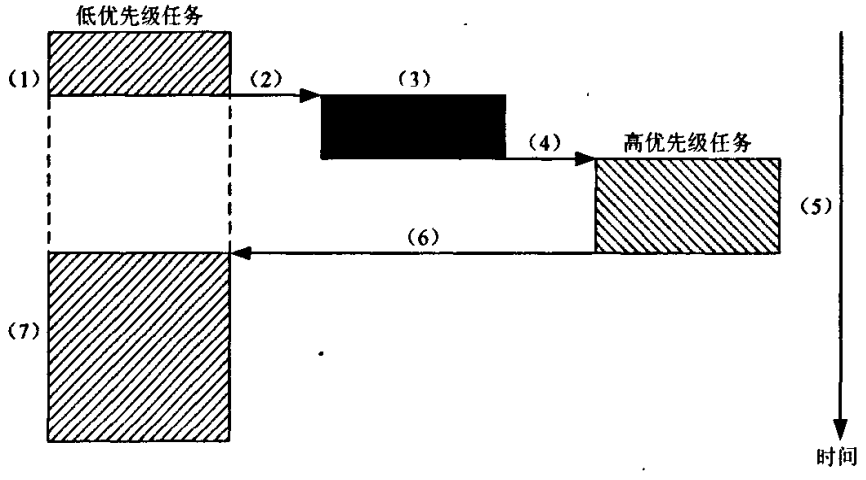


图 3-2 占先式内核

（1）一个低优先级的任务正在执行；（2）一个异步事件向 CPU 发中断；（3）CPU 执行 ISR，响应中断；（4）在 ISR 结束时，激活一个更高优先级的任务；（5）CPU 执行该任务；（6）任务结束或者被另一个时间中断；（7）内核调度原来低优先级的任务执行。

使用占先式内核，最高优先级的任务何时可以执行、何时可以得到 CPU 的使用权都是可知的，占先式内核使任务响应的时间达到最优化。在使用占先式内核时，应用程序不应直接使用不可重入函数，调用不可重入函数时，应满足互斥条件，这一点需要使用互斥信号量来实现，因为低优先级的任务和高优先级的任务可能会同时调用同一个函数，

## 3.3 Small RTOS51 内核的系统功能

### 3.3.1 任务管理 [5][17]

一个任务，也称为一个线程，是一段简单的程序，可以认为自己独享 CPU 的服务。实时内核允许将一项工作分解成多个相互独立的任务。诸如键盘扫描、人机界面、显示、时钟/ 日历、串行通信、模拟输入监视、数据记录、控制环路等功能都可以划分为一个或多个任务。每个任务都是整个应用程序的一部份，都被赋予一定的优先级，有自己一套 CPU 寄存器和堆栈空间。在大多数情况下，并非所有的任务都是实时性任务，对非实时任务应分配较低的优先级。μC/OS-II 可以根据任务的需要，分配不同的堆栈空间，避免不必要的内存空间的浪费。每个任务都是一个无限循环，都可能处在以下 5 种状态：休眠态（DORMANT）、就绪态（READY）、运行态（RUNNING）、挂起态（WAITING）及被中断态（ISR）。一个任务的休眠态，是指该任务驻留在内存中但不被多任务调度；就绪态是指任务已经准备好，可以运行，但是由于该任务的优先级比正在运行任务的优先级低，暂时不能运行；运行态是指任务掌握了 CPU 的使用权，正在执行；挂起态指任务等待某一事件的发生（例如等待某外设的 I/O 操作，等待某共享资源由暂不能使用到可以使用的状态，等待定时脉冲的到来、或者等待超时信号的到来等）；被中断态是指中断发生时，CPU 提供相应的中断服务，原来正在运行的任务暂不能运行。

图 3-3 表示 Small ROTS51 中一些函数提供的服务，这些函数使任务从一种状态变为另一种状态。

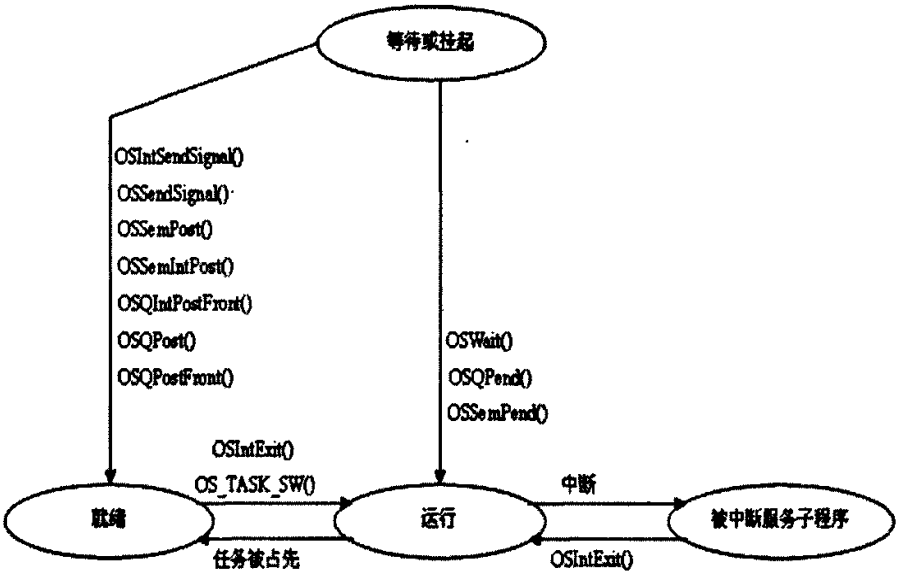


图3.3 Small RTOS51 任务状态转换图

说明如下：

（1） 运行 **Small RTOS**： 在 main 函数中调用 OSStart()后系统开始运行，OSStart()结束后运行优先级为 0 的任务。注意：在调用 OSStart()前不要使能总中断标志，系统会自动执行开中断宏 OS_EXIT_CRITICAL()。

（2） 建立任务：Small RTOS 使用静态方法建立任务，在 config.h 中有这样一段代码：

```
#ifdef IN_OS_CPU_C
extern void TaskA(void);
extern void TaskB(void);
extern void TaskC(void);
void (* const TaskFuction[OS_MAX_TASKS])(void)={TaskA,TaskB,TaskC};
//函数数组 TaskFuction[]保存了各个任务初始 PC 指针,其按任务 ID(既优先级次序)顺序保存
#endif
```

用户只要定义好 OS_MAX_TASKS,TaskFuction[],任务将自动运行。其中 TaskA 等为任务对应函数。注意：1、ID 为 0 的任务为最高优先级任务,系统

运行后首先执行它。2、ID 为 OS_MAX_TASKS 的任务由系统定义。

（3） 删除任务：Small RTOS 1.12 版不允许删除任务，因此，每个任务必须为无限循环结构。

```
while(1)
{
    OSWait(K_SIG, 0);          // 任务睡眠，等待任务唤醒
    处理过程
}
```

任务的唤醒用 OSTaskResume(TASK_ID);完成。

（4） 任务睡眠： 任务可以调用 OSWait()使自己睡眠，其原型如下：

unsigned char OSWait(uint8 typ, uint8 ticks);

功能描述: 系统等待函数,任务调用此函数可以等待一定时间或信号

输入: typ: 等待事件类型,目前可以取以下值,或是其中任意个值的按位或

K_SIG: 等待信号

K_TMO: 等待超时

ticks : 等待超时时的系统节拍数

输出 : NOT_OK : 参数错误

TMO_EVENT : 超时到

SIG_EVENT : 有信号

（5） 如果任务等待信号,另一个任务可以调用 OSSendSignal()使其进入就绪状态，其原型如下：

void OSSendSignal(uint8 TaskId);

功能描述: 中断中给指定任务发送信号,既使指定任务就绪

输 入: TaskId : 任务 ID

输 出: 无

（6） 中断调用 OSIntSendSignal()也可以唤醒睡眠任务，其原型如下：

void OSIntSendSignal(uint8 TaskId);

功能描述: 任务中给指定任务发送信号,既使指定任务就绪

输 入: TaskId : 任务 ID

输 出: 无

### 3.3.2 时间管理 [5][7]

时钟节拍是特定的周期性中断。这个中断可以看作是系统心脏的脉动。中断之间的时间间隔取决于不同应用，一般 10~200ms。时钟的节拍时中断时的内核可以将任务延时若干个整数时钟节拍，并且当任务等待事件发生时，提供等待超时的依据。时钟节拍率越快，则系统的额外开销越大。

各种实时内核都有将任务延时若干个时钟节拍的功能。然而这并不意味着掩饰的精度是一个时钟节拍，只是在每个时钟节拍中断到来时对任务延时做一次裁决而已。

因为只是在每个时钟节拍中断到来时对任务延时做一次裁决，所以如果任务将自身延迟一个时钟节拍的时间，则实际的延时并不是精确的 1 个时钟节拍的时间。实际上可能有三种情况：

第一种情况：系统负载较轻，延迟小于一个节拍。但实际上每次延时多少是变化的，这就引起了任务执行时间的抖动。这是因为延时请求可以发生在两次时钟节拍中断之间的任一时刻，从请求到下一次时钟节拍中断的时间多少有一些变化。

第二种情况：系统负载较重，延时可能几乎没有发生。如果任务将自身延时一个时钟节拍的请求刚好发生在下一个时钟节拍之前，则每个任务的每次执行几乎是立即开始的。因此，如果要求任务的延迟至少为一个时钟节拍，则要多定义一个延时时钟节拍，则要多定义一个延时时钟节拍。换句话说，如果想要哦将一个任务至少延迟 5 个时钟节拍，得在程序中延时 6 个时钟节拍。

第三种情况：系统负载很重，延时可能超过 1 个时钟节拍。这种情况应当避免。当低优先级任务申请延时一个时钟节拍后，高优先级任务有了繁重的任务，所有有关优先级的任务（加上中断服务程序）的执行时间长于一个时钟节拍。此时，尽管这个申请延时的任务已经就绪，也必须等待高优先级任务退出才能执行，延时的时间也就取决于高优先级任务何时退出了。这在某些应用中或许是可以的，而在多数情况下是不可接受的。

上述情况在所有的实时内核中都会出现。这与 CPU 负荷有关，也可能与系统设计不正确有关。以下是这类问题可能的解决方案：

① 增加微处理器的时钟频率；

② 增加时钟节拍的频率；

③ 重新安排任务的优先级；

④ 避免使用浮点运算（如果非使用不可，则尽量用单精度数）；

⑤ 使用能较好的优化程序代码的编译器；

⑥ 时间要求苛刻的代码用汇编语言编写；

Small RTOS51 需要用户提供周期性的信号源，用于实现任务的延时和超时的时间基准，这个周期性的信号源就是时钟节拍. 节拍的频率一般为 10～100 次/s，即 10～100Hz. 时钟节拍频率越高，时间精度越高，系统地额外负荷也越重. 时钟节拍的实际频率取决于用户程序的精度. 时钟节拍源可以是专门的硬件定时器，也可以来自 50Hz/60Hz 工频信号.

用户必须周期性调用 Small RTOS51 的时钟节拍服务函数 OSTimeTick()。方法有两种：一种是时钟节拍源中断服务程序中调用；另一种是时钟节拍源中断服务程序是一个优先级比较高的任务进入执行状态，这个任务再调用 OSTimeTick()。时钟节拍源中断服务程序是与系统相关的。因为在标准 C 中不能直接处理 CPU 的寄存器，所以时钟节拍源中断服务程序一般要用汇编语言编写。然而，如果用户所用的编译器支持相应的语言扩展，那么也可以用 C 语言来编写。

**时钟节拍中断服务子程序示意代码如下**

```
void OSTickISR(void) interrupt OS_TIME_ISR
 {
   关中断;
   保存 CPU 寄存器的值;
   #if TICK_TIMER_SHARING>1
    static unit8 TickSum=0;
   #endif
   #if EN_USE_TICK_TIMER>0
    UserTICK
    .........
    #endif
}
```

### 3.3.3 信号量[5][7]

信号量是一种协议机制，可以用来：控制对共享资源的访问（互斥）、标记某事件的发生、任务间同步。信号量相当于任务执行的钥匙，其功能与令牌环网中的令牌相似。如果某任务采用了信号量机制，只有拥有该钥匙时，任务才能继续执行。如果信号量被另一个任务占用，则该任务挂起，直到信号量当前的拥有者释放信号量。当某任务需要独占某资源时，可用二进制信号量实现互斥。例如，某任务在向一台行式打印机送数据前发申请独占该打印机，即令信号量为0（打印机不可用）。如果此时有另一个任务也要向该打印机送数据，也发独占打印机的申请，就会出现申请失败，表明该打印机已为别的任务独占，因此不能再向其发送数据，一直等待该信号量，直到前一个任务使用完打印机，将信号量置为1（打印机可用）时，才重新唤醒该等待任务。信号量还可以标记事件的发生，此时使用计数信号量。例如，可以利用信号量在任务中处理从串口接收的数据，而不用在 ISR 中处理。处理数据的应用程序挂起在信号量上。当接收到一个字符时，由 ISR 从串口读回数据，放入环型缓冲区中，然后 ISR 标记信号量（比如使信号量加1），通知等待任务已收到一个字符。标记信号量激活等待任务到就绪态。ISR 结束后，内核判断该等待任务是不是当前优先级最高的任务，如果是，则恢复该任务的执行，应用程序从环型缓冲区取出字符进行处理。μC/OS-II 内核允许用户指定最大等待时间，即超时设置，这样如果通信线上无数据传输时，等待任务可进行相应处理。当然，对于本例来说，频繁地使用信号量会给 CPU 造成大量的额外开销，因为在低优先级任务和等待字符的任务之间要进行上下文切换。因此更好的解决方法是只在收到某特定字符，譬如回车符时才标记信号量。这样，只有当从串口接收了一条完整的命令后才会唤醒高优先级任务。

在 Small RTOS51 中，用一个0到（OS_MAX_SEMS-1）的值做索引标识一个信号量，所有对信号量的访问都通过它访问。Small RTOS 在使用一个信号量之前，首先要初始化该信号量,也即调用 OSSemCreate()函数，对信号量的初始计数值赋值，该初始值为 0 到 255 之间的一个数。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，那么该信号的初始值应设为1（例如，把它当作二值信号量使用）。最后，如果该信号量是用来表示允许任务访问 n 个相同的资源，那么该初始值

显示应该是 n，并把该信号量作为一个可计数的信号量使用。信号量使用的内存空间由用系统分配。

### 3.3.4 任务间通信—消息队列 [5][7]

Small RTOS51 内核提供了消息队列服务，完成两个任务之间、任务和 ISR 之间进行数据交换。消息队列用于给任务发消息。通过内核提供的服务，任务或中断服务子程序可以将一条消息放入消息队列。同样，一个或多个任务可以通过内核服务从消息队列中得到消息。通常，消息队列传递的是一个指针，以便任务可以通过它发送和接收任意类型数据（即消息，也就是指针指向的内容）。然而，为了节省 RAM 的占用（这是 Small RTOS51 的设计目标），Small RTOS51 的消息队列是以字节变量（unit8 型变量，范围是 0~255）作为消息，而不是以指针指向的内容作为消息。也就是说，消息队列发送一个消息实质是将一个范围为 0~255 的值村到消息队列中。而不是将一个指针村到消息队列中。类似的，从消息队列中获得一个消息就是得到一个范围为 0~255 的值。这个 0~255 的值用户可以任意解释。通常先进入消息队列的消息先传给任务，也就是说，任务先得到的是先进入消息队列的消息，即先进先出的原则（FIFO）。然而 Small RTOS51 也允许使用后进先出的方式（LIFO）。

当一个以上的任务要从消息队列接收消息时，每个消息队列有一张等待消息任务的等待列表。如果消息队列中没有消息，即消息队列是空，则等待消息的任务就被放入等待消息任务列表中，直到有消息到来。通常，内核允许等待消息的任务定义等待超时的时间。如果在限定时间内任务没有收到消息，则该任务就进入就绪台并开始运行，同时返回出错代码，指出出现等待超时错误。一旦一则消息放入消息队列，则该消息将传给等待消息的任务中优先级最高的那个任务，或是最先进入等待消息任务列表的任务。

内核提供的典型消息队列服务如下：

① 消息队列初始化，队列初始化时总是清为空。

② 方一则消息到队列中去（POST）。

③ 等待一则消息的到来（PEND）。

④ 无等待取得消息。如果队列中有消息则任务可以得到消息，消息从队列中取走；但如果此时队列为空，则内核部将该任务挂起，只是用特别的返回代

码通知调用者，队列中没有消息。

与一般的 RTOS 不同，Small RTOS51 的消息队列是以字节变量（uint8 型变量，范围为 0 到 255）作为消息，而不是以指针指向的内容作为消息。也就是说，消息队列发送一个消息实质是将一个 0 到 255 的数值存到消息队列中，而不是将一个指针存到消息队列中。类似的，从消息队列中获得一个消息就是得到一个范围为 0 到 255 的值。这个 0 到 255 的值用户可以任意解释。如果用户确实需要多个字节作为一个整体传递，可以有两个方法：一是消息队列仅仅传递数据的索引，真实数据放在另外的地方；一是一次向消息队列中发送多个消息。消息队列使用的内存空间是由用户分配。

### 3.3.5 本章小结

本章首先通过前/后台系统和实时内核管理的系统之间的对比，说明在 RTOS 平台上开发软件的优点，接下来介绍了 Small RTOS51 提供的典型的系统服务。重点介绍任务调度、信号量、消息队列的概念，并结合曾遇到的问题说明其典型应用。对实时内核平台下软件设计有一个基本的说明。

# 第 4 章　硬件设计

## 4.1　硬件需求

由于系统采用的总线通讯方式选用 CAN 总线通讯方式，而在此款单片机中集成了 CAN 总线协议控制器，我们只需要外加总线驱动器，即可以很容易构架一个基于 CAN 总线的智能测控节点。

由于我们需要对电流及电压量等交流电参数进行交流采样，需要选用一块模数转换器芯片。模数转换器的技术指标包括转换时间，分辨率，输入模拟电压范围，相对精度，温度系数，电源功率等。但对于诊断终端来说，需要考虑的两个最重要的性能指标一个是分辨率，还有一个就是转换速率，而其它指标由于误差较小，对于终端单元来说可以忽略。在诊断终端中，由于各通道共用一个 A/D 转换器，因此至少要求所有通道轮流转换所需时间的总和小于采样间隔，此外，除考虑全部通道的采样保持器切换时间和微机读写时间外，还要考虑所有通道的模拟量经 A/D 转换成数字量的时间的总和，一般情况下，转换速率在几个 μs 到几十个 μs 的芯片均可选用，对于分辨率来说过高的分辨率意味着高昂的价格，依照本终端的性能要求，12 位到 16 位精度的 A/D 均可选用。在 040 单片机内集成了一个 12 位的转换速率为 100ksps 的 SAR 型模数转换器，同时 040 内部还集成了电压基准。这一点完全符合我们的要求。

CygnalF040 单片机内部还集成了 3 个回差电压可调的模拟比较器，以及 6 个 PCA(可编程计数阵列)，我们可以很方便地利用这些片上单元构架频率测量功能模块。

## 4.2　微控制器的选型及介绍 [8]

当前主流的微控制器从 8 位、16 位到 32 位共有几十种，根据 2.3.2.1 中提出的要求，选择了 Silicon Laboratories 公司的 C8051F040。

C8051F040 是一款性能价格比非常高的 SOC，具有高速、流水线结构的 CIP-51 内核（可达 25MIPS），指令集与传统的 MCS-51 完全兼容，在系统构架、

外围设备等方面有了很大的改进，使得集成度更高，运行速度更快。

### 4.2.1 C8051F040 模拟外设

C8051F040是完全集成的混合信号片上系统型MCU，具有64 个数字I/O 引脚，片内集成了一个CAN2.0B 控制器。下面列出了一些主要特性：

- ★ 高速、流水线结构的8051 兼容的CIP-51 内核（可达25MIPS）。
- ★ 控制器局域网（CAN2.0B）控制器，具有32 个消息对象，每个消息对象有其自己的标识。
- ★ 全速、非侵入式的在系统调试接口（片内）。
- ★ 真正12 位、100 ksps 的ADC，带PGA 和8 通道模拟多路开关。
- ★ 允许高电压差分放大器输入到12/10 位ADC（60V 峰-峰值），增益可编程。
- ★ 真正8 位500 ksps 的ADC，带PGA 和8 通道模拟多路开关。
- ★ 两个12 位DAC，具有可编程数据更新方式。
- ★ 64KB可在系统编程的FLASH 存储器。
- ★ 4352（4K+256）字节的片内RAM。
- ★ 可寻址64KB 地址空间的外部数据存储器接口。
- ★ 硬件实现的SPI、SMBus/ I2C 和两个UART 串行接口。
- ★ 5 个通用的16 位定时器。
- ★ 具有6 个捕捉/比较模块的可编程计数器/定时器阵列。
- ★ 片内看门狗定时器、VDD 监视器和温度传感器。

具有片内VDD 监视器、看门狗定时器和时钟振荡器的C8051F040是真正能独立工作的片上系统。所有模拟和数字外设均可由用户固件使能/禁止和配置。FLASH 存储器还具有在系统重新编程能力，可用于非易失性数据存储，并允许现场更新8051 固件。片内JTAG 调试电路允许使用安装在最终应用系统上的产品MCU 进行非侵入式（不占用片内资源）、全速、在系统调试。该调试系统支持观察和修改存储器和寄存器，支持断点、观察点、单步及运行和停机命令。在使用JTAG 调试时，所有的模拟和数字外设都可全功能运行。

### 4.2.2 C8051F040 高速的 8051 微内核

采用流水线结构，与标准的8051 结构相比指令执行速度有很大的提高。在标准8051 中，除MUL 和DIV 以外所有指令都需要12 或24 个系统时钟周期，最大系统时钟频率为12-24MHz。在C8051F040中，对于70%的指令的执行时间为1 或2 个系统时钟周期，只有4 条指令的执行时间大于4 个系统时钟周期。

### 4.2.3 C8051F040 存储器

C8051F040有标准的8051 程序和数据地址配置。它包括256字节的数据RAM，其中高128字节为双映射。用间接寻址访问通用RAM的高128字节，用直接寻址访问128字节的SFR 地址空间。SFR地址空间可包含多达256个SFR页。通过SFR分页，可以控制大量用于控制和配置片内外设所需要的SFR。数据RAM 的低128字节可用直接或间接寻址方式访问。前32个字节为4个通用寄存器区，接下来的16字节既可以按字节寻址也可以按位寻址。C8051F040还另有位于外部数据存储器地址空间的4KB的RAM块和一个可用于访问外部数据存储器的外部存储器接口（EMIF）。这个片内的4KB RAM块可以在整个64K外部数据存储器地址空间中被寻址（以4K为边界重叠）。外部数据存储器地址空间可以只映射到片内存储器、只映射到片外存储器、或两者的组合（4KB以下的地址指向片内，4KB以上的地址指向EMIF）。图4.1 给出了MCU系统的存储器结构。

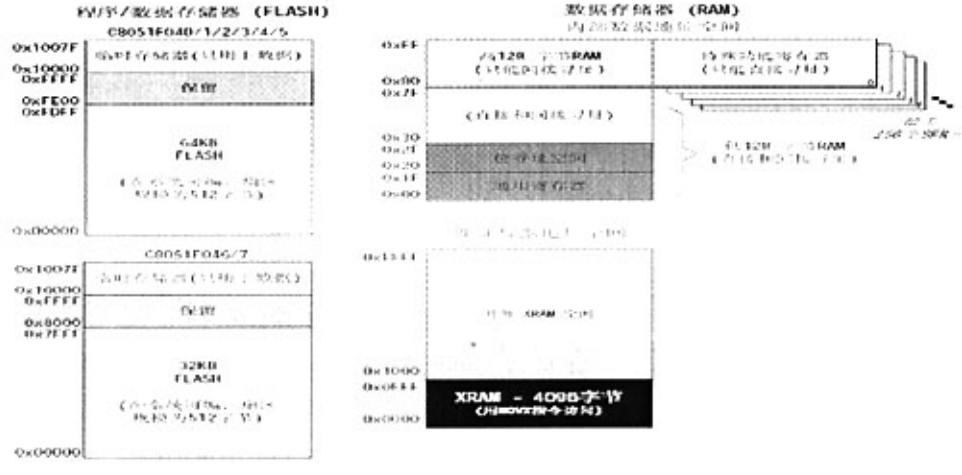


图4.1 片内存储器组织

### 4.2.5 C8051F040 的时钟源

MCU 内部有一个独立运行的时钟发生器，在复位后被默认为系统时钟。如果需要，时钟源可以在运行时切换到外部振荡器，外部振荡器可以使用晶体、陶瓷谐振器、电容、RC 或外部时钟源产生系统时钟。时钟切换功能在低功耗系统中是非常有用的，它允许MCU 从一个低频率（节电）外部晶体源运行，当需要时再周期性地切换到高速内部振荡器（可达25MHz）。

综上所述，C801F040 单片机是一款性能价格比非常高的 SOC，且满足本设计应用。最主要的一点是片上丰富的资源覆盖了 3.1 中要求的大部分模拟、数字设备，体现了 SOC 的优点。

## 4.3 系统硬件框图

根据4.2中的硬件需求，以C8051F040单片机为核心设计了系统的硬件电路，框图如图 4-2 所示（硬件原理图见附录）。具体的模块在下面几节详细介绍。

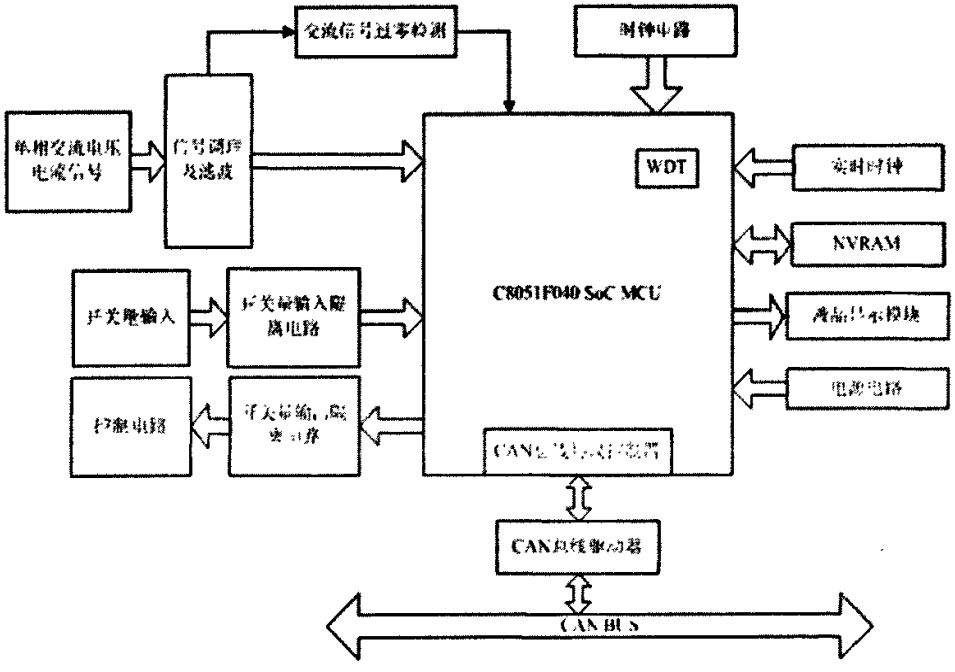


图 4-2 系统硬件框图

## 4.4 系统硬件框架

系统从总体上分成主控模块，交流电信号输入及调理模块，人机接口模块，CAN 总线通信模块以及开关量输入输出通道 5 大模块，参见图 3.1。各模块的功能如下：

**1. 主控模块**

主控模块所完成的主要功能为执行相应算法，对采样进来的电力参数进行处理，计算出交流电压电流信号的有效值，以及有功功率和功率因素，同时完成故障录波及越限报警功能。

**2. 交流电信号输入及调理模块**

该模块的主要功能为将输入交流电压电流信号调理成满足 040 片内模数转换器量程要求的电压信号，并为电网频率测量提供测量电信号。

**3. 人机接口模块**

该模块主要功能是完成人机的对话与交流。物理上表现为液晶显示模块与键盘。核心功能为显示当前所测电网的工作状态，设置开出开关量的分合闭锁以及调整故障越限的各类参数。由于采用了全程菜单驱动方式，使得该模块具有较好的友好性。

**4. CAN 总线通信模块**

该模块的主要功能为完成与上位机的数据通信功能。

**5. 开关量输入输出通道**

开关量输入输出通道的主要功能为接收并执行遥控指令，控制开出刀闸的分合以及闭锁；接收并执行遥信指令，监测开入刀闸的开合状态。

### 4.7.1 CAN 总线通信模块关键器件及其选型[9]

**1. CAN 总线协议控制器**

CAN 总线是一种适合工业现场自动控制的计算机局域网络，在网络的层次结构中，数据链路层和物理层是保证通信质量至关重要，不可或缺的部分，也是网络协议最复杂的部分。CAN 控制器就是扮演这个角色，它是以一块可编程芯片上的逻辑电路组合来实现这些功能的。通过对它编程，CPU 可以设置它的工作方式，控制它的工作状态，进行数据的发送与接收，把应用层建立在它的

基础上。

C801F040 单片机内置有 CAN 总线控制器，不需要另外选取 CAN 总线控制器。

**2. CAN 总线驱动器**

CAN 总线驱动器提供了 CAN 控制器与物理总线的接口，是影响系统网络性能的关键因素之一。本设计中选用 PHILIP 公司生产的 TJA1050T 总线驱动器，该器件是 82c250 的替代品。TJA1050 相对于 82C250 来说除了增强了 EMC 特性外，最大的改进在于增加了一个超时定时器，用于对 TXD 端的低电平进行监视，这样就可以避免由于节点故障而造成 TXD 端长时间低电平时，总线上其它的节点也无法通信的情况的出现。该器件提供了 CAN 控制器与物理总线之间的接口以及 CAN 总线的差动和接收功能。TJA1050 的主要特性如下[27]-[28]：

- 完全符合ISO 11898 标准
- 高速率最高达1Mbit/s
- 电磁辐射(EME)极低
- 电磁抗干扰(EMI)性极高
- 不上电的节点不会对总线造成扰动
- TxD 引脚有防止箝位在显性总线电平的超时功能
- 静音模式中提供了只听模式和Babbling Idiot 保护
- 保护总线引脚防止汽车环境中的瞬态干扰
- 输入级和3.3V 以及5V 的器件兼容
- 输出驱动器受到温度保护
- 防止电池对地的短路
- 至少可以连接110 个节点

## 4.8 本章小结

以测控系统的终端单元的基本构件为参照对象，设计了以 C8051F040 单片机为核心的硬件平台，由于 C8051F040 单片机具有丰富的片上资源，只需进行少数扩展就能满足系统的需求。

# 第 5 章 Small RTOS 在 C8051F040 单片机上的移植

## 5.1 Small RTOS51 的版本 [5]

Small RTOS51 版本号定义方式为：a.bc.d。a:主版本号，一般重大改变时改变它。bc:次便本号，一般功能增加时改变它。d:同一版本的修订序号。

Small RTOS51 典型的版本包括 Small RTOS51 1.12.1 版修正 OS_MAX_TASKS 为 8、16 时的 bug；1.12.0 版，OS_MAX_TASKS 就是用户任务数量；1.00 版，使用户可以进行更多的配置，可以禁止中断管理，51 系列支持软的非屏蔽中断,并调整目录结构等。

本文采用的是 Small RTOS51 最新版本 1.12.1 版本。

## 5.2 IDE 环境和适配器的选择 [11]

到目前为止，支持 C8051F 系列单片机的 IDE 有 2 种，一种是 Silicon Lab 公司免费提供 C8051F 系列单片机的集成开发环境 Cygnal IDE，用户界面友好，操作方便；另一种是 Keil uVision2 IDE，在使用时需要安装针对 C8051F 系列单片机的动态链接库。在对 Cygnal IDE 测试过程中发现编译器的效率要低于 Keil uVision2 IDE。因此本项目选择了 μVision2 V2.38A 版本，C 编译器版本是 C51.exe V7.06，汇编器的版本是 A51.exe V7.07。

支持 C8051F 系列单片机的适配器只有一种，即 Silicon Lab 公司的 EC2。

## 5.3 Small RTOS51 在 C8051F040 单片机上的移植 [5]

（1）在 **os_cpu.h** 中定义几个宏：

```
#define    OS_INT_ENTER() OSIntNesting++
  /* 中断嵌套管理*/
#define OS_ENTER_CRITICAL()
 EA = 0,Os_Enter_Sum++
```

```
/* 关中断*/
#define  OS_EXIT_CRITICAL()    if (--Os_Enter_Sum==0) EA = 1
/* 开中断 */
#define  HIGH_BYTE   0
/* uint16 的高位字节*/
#define  LOW_BYTE    1
/* uint16 的低位字节*/
#define  OS_TASK_SW()  OSCtxSw()
/* 任务切换函数*/
```

OS_ENTER_CRITICAL()、OS_EXIT_CRITICAL()分别定义为关中断和开中断在特定的 c 编译器的表示方法。HIGH_BYTE、LOW_BYTE 定义 uint16 型变量在特定的 c 编译器的存储方法，如果高位字节的地址小于低位字节的地址（如 keil c51），则 HIGH_BYTE 为 0，LOW_BYTE 为 1。否则（如 8086 系列），HIGH_BYTE 为 1，LOW_BYTE 为 0。OS_INT_ENTER 则将变量 OSIntNesting 加 1。它仅在中断服务程序中使用。如果中断嵌套层数可能超过 255，OS_INT_ENTER 要防止 OSIntNesting 溢出。OS_TASK_SW()定义非中断中任务切换时执行的指令，可以是一条软中断指令（例如在 8086 系列 CPU 上），或仅仅是函数调用（如 keil c51）。

（2）定义与编译器无关的变量类型：

```
typedef unsigned char   uint8;
/* 定义可移植的无符号 8 位整数关键字*/
typedef signed    char  int8;
/* 定义可移植的有符号 8 位整数关键字*/
typedef unsigned int    uint16;
/* 定义可移植的无符号 16 位整数关键字*/
typedef signed    int   int16;
/* 定义可移植的有符号 16 位整数关键字*/
typedef unsigned long   uint32;
/* 定义可移植的无符号 32 位整数关键字*/
typedef signed    long  int32;
/* 定义可移植的有符号 32 位整数关键字*/
```

**（3）在 os_cpu_c.c 和 os_cpu_a.asm 中定义几个函数：**

OSStart、OSIntCtxSw、OSTickISR、OSIdle 和 OS_TASK_SW()最终调用的函数或中断。

OSStart：初始化任务并让 ID 为 0 的任务执行。同时允许中断。

定义如下：

```
void OSStart(void)
 {
        初始化除 ID 为 0 以外所有任务堆栈;
        OSTaskID = 0;
        使堆栈指针指向 ID 为 0 的任务堆栈空间;
        OS_EXIT_CRITICAL();
        使程序指针指向 ID 为 0 的任务的程序首地址;
  }
```

OSIntCtxSw:中断中任务切换函数，定义如下：

```
/**中断使任务放弃 CPU 环境保存函数**/
RSEG   ?PR?OSIntCtxSw?OS_CPU_A
OSIntCtxSw:
    USING    0
                                    ;是否是优先级最低任务
    MOV      A,#OS_MAX_TASKS
    XRL      A,OSTaskID
    JNZ      OSIntCtxSw_0
                                    ;是则不需要保存所有寄存器
;SP=SP-13-4                         ;4:两层函数调用堆栈，13：寄存器数目
    MOV      A,#(-17)
    ADD      A,SP
    MOV      SP,A        ;跳转到 OSCtxSw，同时通知 CPU 中断处理完成
    MOV      A, #LOW   OSCtxSw
    PUSH     ACC
    MOV      A, #HIGH OSCtxSw
    PUSH     ACC
```

```
    RETI                        ;需要保存所有寄存器
OSIntCtxSw_0:
;SP=SP-4                        ;4:两层函数调用堆栈
    MOV     A,#0FCH
    ADD     A,SP
    MOV     SP,A ;设置标志：任务再次恢复运行时需要恢复所有寄存器
    MOV     DPTR,#OSMapTbl
    MOV     A,OSTaskID
#if OS_MAX_TASKS < 9
    MOVC    A,@A+DPTR
    CPL     A
    ANL     OSFastSwap,A
#else
    CLR     C
    SUBB    A,#8
    JC      OSIntCtxSw_1
    MOVC    A,@A+DPTR
    CPL     A
    ANL     OSFastSwap,A
    SJMP    OSIntCtxSw_2
OSIntCtxSw_1:
    MOV     A,OSTaskID
    MOVC    A,@A+DPTR
    CPL     A
    ANL     OSFastSwap+1,A
OSIntCtxSw_2:
#endif
#if EN_REENTRANT > 0
    PUSH    ?C_XBP
    PUSH    ?C_XBP+1
#endif
```

```
    PUSH    Os_Enter_Sum                ;保存关中断计数器
    ;跳转到堆栈处理，同时通知 CPU 中断处理完成
    MOV     A, #LOW  C_OSCtxSw
    PUSH    ACC
    MOV     A, #HIGH C_OSCtxSw
    PUSH    ACC
    RETI
```

OSIntCtxSw 由 OSIntExit 直接调用，堆栈指针调整为中断程序调用 OSIntExit 前的状态即为执行若干出栈指令。堆栈空间变换可以参照 keilc51 目录下 Os_cpu_c.c 文件中被注释的 C_OSCtxSw 函数，C_OSCtxSw 还包括 堆栈指针指向新的堆栈。OSTickISR 为系统节拍中断服务程序

定义如下：

```
#pragma disable          /* 除非最高优先级中断，否则，必须加上这一句*/
void OSTickISR(void) interrupt OS_TIME_ISR
{   #if TICK_TIMER_SHARING >1
    static uint8 TickSum = 0;
    #endif
    #if EN_USER_TICK_TIMER > 0
    UserTickTimer();                          /* 用户函数*/
    #endif
    #if TICK_TIMER_SHARING >1
    TickSum = (TickSum + 1) % TICK_TIMER_SHARING;
    if (TickSum != 0)
    { return; }
    #endif
    OS_INT_ENTER();                          /* 中断开始处理  */
    #if EN_TIMER_SHARING > 0
    OSTimeTick();                       /* 调用系统时钟处理函数*/
    #else
    OSIntSendSignal(TIME_ISR_TASK_ID);
    /* 唤醒 ID 为 TIME_ISR_TASK_ID 的任务*/
```

```
#endif
    OSIntExit();            /* 中断结束处理*/
}
```

其中有一些 CPU 的禁止中断和保存任务环境由 CPU 自动处理或是自动处理一部分。OSIdle()优先级最低的任务，定义如下：

```
void OSIdle(void)
{
    while(1)
    {
      PCON = PCON | 0x01;       /* CPU 进入休眠状态 */
    }
}
```

OSCtxSw ()最终调用的函数或中断：非中断中任务切换函数，定义如下

```
RSEG  ?PR?OSCtxSw?OS_CPU_A
OSCtxSw:
   USING   0
#if EN_REENTRANT > 0
    PUSH     ?C_XBP
    PUSH     ?C_XBP+1
#endif
    PUSH     Os_Enter_Sum              ;保存关中断计数器
    ;设置标志：任务再次恢复运行时不必恢复所有寄存器
    MOV      DPTR,#OSMapTbl
    MOV      A,OSTaskID
#if OS_MAX_TASKS < 9
    MOVC     A,@A+DPTR
    ORL      OSFastSwap,A
#else
    CLR      C
    SUBB     A,#8
    JC       OSCtxSw_1
```

```
    MOVC    A,@A+DPTR
    ORL     OSFastSwap,A
    LJMP    C_OSCtxSw
OSCtxSw_1:
    MOV     A,OSTaskID
    MOVC    A,@A+DPTR
    ORL     OSFastSwap+1,A
#endif
    LJMP    C_OSCtxSw
```

## 5.4 测试移植代码

创建的 3 个测试任务及源码如下：

```
OSTaskCreate(TASKA, NULL, 0);
OSTaskCreate(TASKB, NULL, 1);
OSTaskCreate(TASKC, NULL, 2);
void TASKA(void)
{
    DisAStr(2,2,"HELLO")                    // 显示“HELLO”
    OSTimeDly(1);
}
void TASKB(void)
{
    DisAStr(3,1,"WORLD");                   // 显示“WORLD”
    OSTimeDly(5);
}
void TASKC(void)
{
    DisAStr(4,3,"！！！");                  // 显示 “！！！”
    OSTimeDly(10);
}
```

多任务调度开始后，通过设置断点可以清晰地看到运行情况。

高优先级的任务 TASKA()能首先被调度运行，运行后延时函数 OSTimeDly（）被调用，显示 “HELLO”；延时一个时钟节拍，没有更高优先级任务，则继续执行 TASKA（），直到延时 5 个时钟节拍到，TASKB（）就绪并执行，显示“WORLD”；延时 15 个时钟节拍到，TASKC（）开始执行，显示“！！！”；

最终显示结果为：

HELLO
WORLD
！！！

通过以上分析可以认为移植结果是正确的。

## 5.5 本章小结

在操作系统平台下开发程序，首先要掌握内核。上述过程对 Small RTOS51 在 C8051F040 单片机移植过程中的关键点进行详细的分析和说明，不但为后续的软件开发搭建一个可靠的平台，而且上述移植过程有助于对任务堆栈、任务调度的理解。同时也积累一些嵌入式软件调试的经验。

# 第 6 章　系统软件设计

## 6.1　实时系统软件开发基本概念 [13][14]

开发实时软件系统最重要的三个基本概念是：并发、模块化和有限状态机。并发是通过并发任务(也叫并发进程) 的划分，使得每一任务代表一个顺序任务(Task) 或者顺序组件(Component) 来执行，从而保证在整个系统中有多个任务并行执行。模块化将系统分解为结构定义良好的、小的、可管理的单元，是信息隐藏（Information Hide）方法之一，其目的是提高软件模块的重用性。有限状态机用来描述实时系统行为，通过系统的状态转换分析，可以对系统进行动态验证和确认。

围绕三个基本概念，实时软件系统软件开发技术主要目标是：并行任务分解、软件重用性和实时系统行为(尤其是动态行为) 定义。建立在这三个目标基础之上的软件开发技术有：实时系统结构化分析（RTSA）、Jackson系统开发（JSD）、NRL方法、实时软件系统设计（DARTS）和面向对象分析与设计（OOA & OOD）。

DARTS方法给出了划分任务的准则和定义任务间接口的机制。在需求分析的基础上，DARTS设计方法主要步骤有：数据流分析（Data Flow Analysis）、任务划分（Decomposition into Tasks）、任务接口定义（Definition of Task Interface）、任务设计（Task Design）。在数据流分析中,DARTS 使用结构化分析的方法建立系统的模型，在任务划分、接口定义阶段则大量采用信息隐藏思想，而在任务设计中，却采用JSD 技术来组织任务，并设计模块。DARTS 建立了一套完整的任务划分准则,在任务接口设计中使用消息队列进行任务之间信息交换和任务同步，从而非常适合用来开发特定应用的实时系统。

本系统软件设计主要参照DARTS的设计方法。

## 6.2　软件需求

在 Small RTOS51 平台下，实现下列 API 函数，在功能实现的前提下，要求

使用 Small RTOS51 提供的系统服务，例如时间延时功能、信号量及消息队列等。

（1）键盘部分。独立式键盘，键盘 API 函数实现下述功能：可配置的缓冲区；支持自动重复；跟踪一个键被按下的时间。

（2）LCD 部分。LCD 的 API 函数实现如下功能：LCD 的初始化配置；显示 ASCII 字符串。

（3）ADC 部分。API 函数实现如下功能：数据读取；ADC 故障实时检测。

（4）DAC 部分。API 函数实现简单的输入功能。

（5）CAN 总线通信部分。实现 CAN 总线通信协议。

## 6.3 键盘驱动函数的实现 [15]

### 6.3.1 键盘驱动程序流程

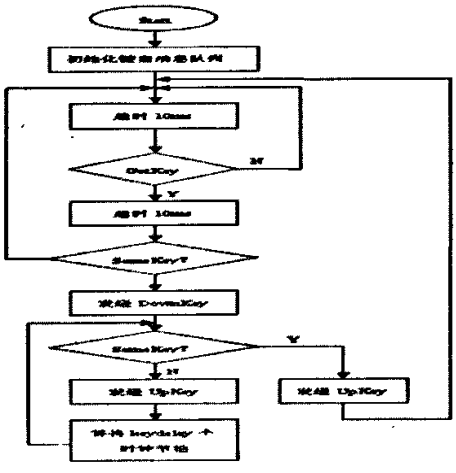

图 6.1 键盘驱动程序流程图

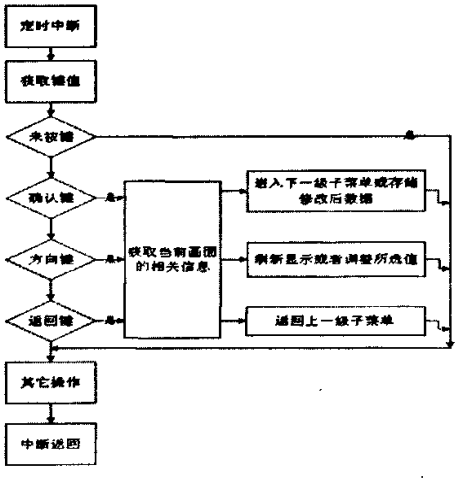


图 6.2 人机界面软件框图

本系统采用独立式键盘设计，菜单驱动程序设计的实质就是对键盘进行周期性处理，使用消息队列传递键盘消息来处理按键问题；解析键值，执行功能模块同时刷新画面。在中断发生后，首先获取键值 void getkey (void)；应用系统函数完成去抖延时；当确认有键按下时，发送 DOWNKEY 消息；松开时发送 UPKEY 消息，并依据键值的不同来进行解析：未按键：在没有按键的情况下，

直接跳过键盘解析模块，其键值为 0x79；方向键：依照当前系统所处的工作状态分成两种情况，如果当前显示画面为菜单选择状态，则上下方向键用来移动游标，达到改变选项的目的。如果是设定画面，则左右方向键用来移动游标，改变设定选项，而上下键用来改变对应选项设定值，其键值为 0xe9、0xd9，0xb9、0xf1；确认键：这里也分成两种情况，如果当前显示画面为菜单选择状态，则确认键用来进入下一级子菜单或者功能模块。而当前画面为设定画面时，确认键用来确认存储所更改的设定值，其键值为 0xf1；返回键：返回键用来返回上一级菜单，返回键值为 0xf8。键盘流程图如图 6.1 所示，人机界面软件框图如图 6.2 所示。

键盘扫描模块使用 2 个内核提供的服务：时间延时和信号量。通过一个独立的任务 KeyTask()来扫描键盘，当应用程序调用 KeyInit()时，创建 KeyTask()，一旦任务被创建，将每 1 个时钟节拍执行一次。

### 6.3.2 键盘接口驱动程序中内核服务的说明

OSTimeDly()功能。为了直接以毫秒表示键盘扫描周期，使用 OSTimeDly()设置键盘扫描延时。此函数首先把设置的时间值转化成系统的时钟节拍，再通过调用 OSTimeDly()功能实现任务调度。既实现了延时功能又保证更高优先级任务的执行。

信号量的使用。此处使用计数式信号量，标志键盘有效码值这一事件的发生。在键盘初始化函数 KeyInit()中，由 KeySemPtr = OSSemCreat(0)创建信号量，并且计数值设置为 0；当系统检测到键盘码值并且成功保存到接收缓冲区后，通过 OSSemPost(KeySem)发送信号量；在读取键盘码值时，首先调用 OSSemPend(KeySem)功能来等待信号量，如果接收缓冲区有键盘码值，读取此值，否则继续进行多任务调度。

### 6.3.3 键盘接口函数

图 6-3 显示了键盘模块的接口框图。

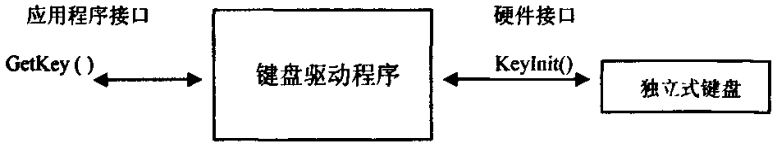


图 6.3 键盘接口函数

应用程序通过下面 2 个函数完成对键盘接口的操作：GetKey()和 KeyInit()。键盘模块可以通过调用 GetKey（）函数来实现。应用程序通过 GetKey()函数获得键盘扫描码，如果没有键盘被按下，调用任务将被挂起或者等待用户设定的超时时间。KeyInit()初始化模块代码，在应用键盘接口模块时，必须通过调用此函数完成硬件端口初始化和键盘扫描任务的创建。

实例如下：

```
void XLTASK1(void)      /***按键使得 LCD 显示时钟***/
{
   uint8 i,KEYvalue;
    while(1)
     {
       KEYvalue=KEY&0xf9;      /***健值解析，第一个按键***/
       if(KEYvalue == 0xF1)
        {
          for(i=0;i<3;i++)       *(time+i)=*(disTime+i);
          SetTime(disTime);   if((*disTime)<23)
           *disTime=(*disTime)+1;
           else    *disTime=0;
           OSTimeDly(1);
          }
       else if(KEYvalue == 0xb9)     /***第二个按键***/
         {
           for(i=0;i<3;i++)
          *(time+i)=*(disTime+i);
```

```
        SetTime(disTime);
      if (*(disTime+1)<59)
          *(disTime+1)=(*(disTime+1))+1;
      else
          *(disTime+1)=0;
          OSTimeDly(1);
      }
    else if(KEYvalue == 0xd9)   /***第三个按键***/
     {
       for(i=0;i<3;i++)
      *(time+i)=*(disTime+i);
       SetTime(disTime);
      if((*(disTime+2))>0)
          *(disTime+2)=(*(disTime+2))-1;
       else
           *(disTime+2)=59;
          OSTimeDly(1);
      }
  else
     {
      LcdPos(4,9);
       DisAStr("out Put");
     OSTimeDly(1);
        }}}
```

## 6.4 LCD 驱动函数的实现 [15]

### 6.4.1 LCD 驱动程序流程

和键盘驱动程序流程相比，LCD 的驱动流程要复杂的多，所有的操作都是顺序结构，所以不用单独建立任务来管理 LCD 接口。

### 6.4.2 LCD 驱动程序中内核服务的说明

OSTimeDly()功能。在 LCD 初始化过程中，对其内部寄存器的配置过程需要反复的延时，为了直接以毫秒表示 LCD 的延时时间，使用 OSTimeDly ()实现延时功能。

信号量的使用。为了防止在多任务调度的过程中不同的任务同时访问 LCD，使用系统提供的二元信号量功能。

### 6.4.3 LCD 接口函数

LCD 的 API 函数实现如下功能：LCD 的初始化配置；显示 ASCII 字符串。图 6 为 LCD 模块的接口框图。应用程序通过下面 8 个函数完成对 LCD 接口的操作 LcdPos（）、LcdRead（）、LcdWDat（）、LcdInit（）、DisANum（）、LcdWrite（）、DisAStr（）、LCDWFont（）、BcdToDec（）、DecToBcd（）。LcdInit（）是模块的初始化代码。由于使用 SmallRTOS51 提供的服务，LcdInit（）必须在使用其它函数之前激活，并且在多任务启动之后调用，LcdInit（）初始化硬件端口配置、创建信号量、设置 LCD 的工作模式。另外，通过 LCDWFont 函数可以自定义 8 个 5×8 像素的字符或符号。例如定义一个下向箭头“↓”，程序清单为：

```
    void XLCDWFont(void)
{
   uint8 i;
    uint8 font[ ]= {0x04,0x04,0x04,0x04,0x15,0x0e,0x04,0x00};
    for ( i=0;i<8;i++ )
        {
           LcdWCmd ( 0x40+I ) ;
           LcdWDat ( font[i] )  ;
        }
}
```

演示结果为：“↓”

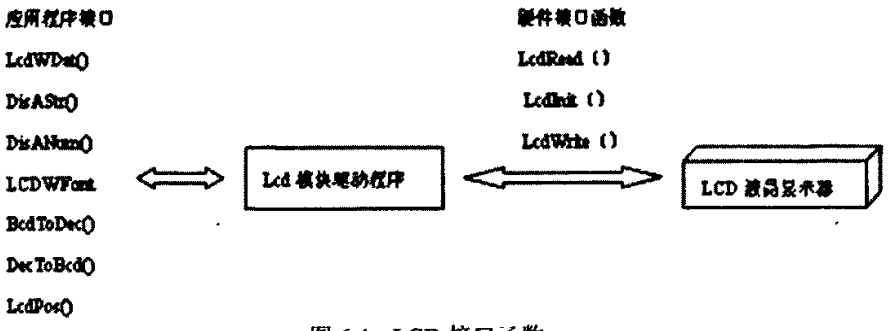


图 6.4 LCD 接口函数

实例如下：

```
/******* 功能描述: LCD 显示主函数，用户程序从这里执行****\
void main(void)
{
    WDTCN = 0xde;    /***看门狗操作***/
    WDTCN = 0xad;
    ExtOsc();
    SysInit();          /***系统初始化***/
    Timer0Init();       /***定时器 0***/
    LcdInit();          /***LCD 初始化***/
    OSStart();          /***系统函数调用，执行任务***/
}

void XLTASK2(void)    /***在 LCD 中显示实时时钟***/
{
     while(1)
        {
            GetTime(time);   LcdPos(1,2);
             DisAStr("<DS1743 DISPLAY>");
             LcdPos(2,5);   DisAStr("[");
             DisANum(*time,2); LcdWDat(':');
```

```
            DisANum(*(time+1),2); LcdWDat(':');
            DisANum(*(time+2),2); LcdWDat(']');
          OSTimeDly(1);
      }
}
```

演示结果为：10（小时）：20（分钟）：20（秒）

## 6.5 实时时钟驱动函数的实现 [15]

DS1743 内部集成的实时时钟相关寄存器位于 RAM 中的最高八个地址，即控制寄存器位于地址 1FF8，年，月，日，星期，时，分，秒寄存器位于地址 1FF9-1FFF。控制寄存器有两个位，读取位和写入位，通过对这两个位写入 1 中止外部时钟寄存器的更新动作，以预防在更新过渡期间读到错误的时钟数据。主函数体共有两个：Ds1743GetTime()、Ds1743SetTime()，其中 uint8 DecToBcd (uint8 number) 与 uint8 BcdTDeco (uint8 number)为十进制与 BCD 码相互转换函数。

1) 实时时钟的读取

首先将‘1’写入控制寄存器的读取位，中止时钟的更新动作，然后再依次从相关时间寄存器获取时间信息。

2) 实时时钟的设定

首先将‘1’写入控制寄存器的写入位，中止时钟的更新动作，然后再依次设定相关时间寄存器的时间信息区取出。

3) 实时时钟由任务 DS1743TASK（）完成对时间的读取。如图所示：

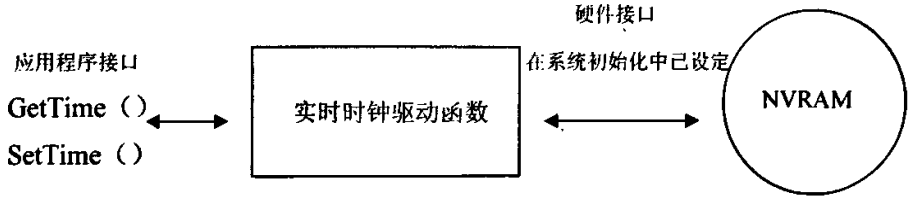


图 6.5 实时时钟驱动函数

### 6.5.3 实时时钟驱动程序内核服务的说明

本系统中在实时时钟驱动程序中未使用信号量

## 6.6 开关量驱动函数的实现

如图 6.6，开关量输入部分模块的作用是为故障诊断终端正确监测开关量的状态提供输入通道。读取开关量的输入比较简单，需要读取多少离散输入，只需提供微处理器多少并行输入线即可，微处理器只需读取输入口，而屏避不需要的输入，并根据输入状态作出判断。

更新离散输出是一种直接的操作，但比更新离散输入根据有技巧性。本系统中离散输入输出由一个人物组成（DIOTASK（）），该任务按照一定的时间间隔来执行。

开关量输入输出部分模块的底层驱动函数主要包括两个函数，如图所示，分别是：

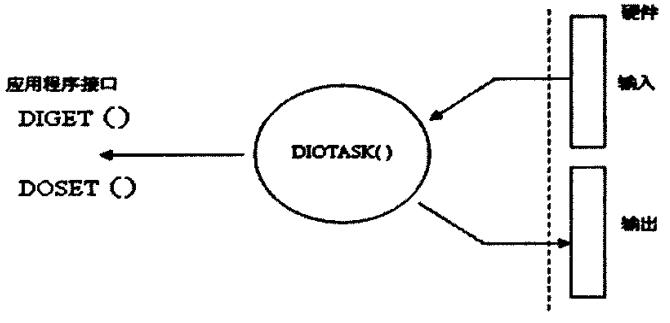


图 6.6 离散输入输出的实现

uchar DIGet(void) //获取开关量的状态

void DOSet(uchar dat) //设定开关量的状态

## 6.7 CAN 总线驱动函数的实现

### 6.7.1 CAN 控制器协议寄存器

该协议寄存器是用来配置CAN 控制器，处理各种中断，监控总线状态以及

置控制器为测试模式。CAN 控制器协议寄存器可使用C8051 MCU 特殊功能寄存器通过索引方式间接访问，其中有些还可以很方便的通过C8051 内部特殊功能寄存器直接寻址来访问。这部分的寄存器有：CAN 控制寄存器（CAN0CN）、CAN 状态寄存器（CAN0STA）、CAN 测试寄存器（CANTST）、错误计数寄存器、位定时寄存器和波特率预比列因子扩展寄存器。其中，CAN0CN、CAN0STA 和CANTST 可通过C8051 MCU 特殊功能寄存器直接访问，其它的只能通过间接访问。

### 6..7.2 CAN 报文对象借口寄存器

CAN 控制器中有两组报文对象接口寄存器，它们用来配置报文RAM中32个报文对象是用来向CAN 总线发送数据，还是从CAN 总线接收数据。当其中的一组被设置为向报文RAM中写数据，另一组则从报文RAM 中读取数据。利用此接口寄存器可以避免CPU 访问报文RAM 与CAN 报文接收和发送缓冲转移之间的冲突。所有的报文对象都存储在报文RAM 里面，通过报文对象寄存器对其进行访问和配置，这些寄存器要通过C8051 的CAN0ADR 和CAN0DAT 寄存器，使用间接索引地址方式来访问。这部分寄存器有：IFX命令请求寄存器、IFX 命令屏蔽寄存器、IFX 屏蔽寄存器1、IFX 屏蔽寄存器2、IFX 仲裁寄存器1、IFX 仲裁寄存器2、IFX 报文控制寄存器、IFX 数据寄存器A1、IFX 数据寄存器A2、IFX 数据寄存器B1 和IFX 数据寄存器B2。

### 6.7.3 CAN 报文处理寄存器

所有的报文处理寄存器都是只读寄存器。通过读取它们的值可以实时地判断相应报文对象的状态，从而使CAN 控制器能正确运行。它们的标识位由CAN0ADR、CAN0DATH 和CAN0DATL 通过索引方式间接来访问。报文处理寄存器提供中断、错误、发送/接收请求和新数据信息。这部分的寄存器包括：中断寄存器、发送请求寄存器、新数据寄存器、中断队列寄存器和报文有效寄存器。那么，通过CAN0ADR、CAN0DATH 和CAN0DATL 寄存器以索引方式间接访问CAN控制器中寄存器的过程会怎样呢？因为每个CAN 控制器寄存器都有一个索引号，如果要访问某一CAN 控制寄存器，只需将此寄存器的索引号写入CAN0ADR 寄存器，而数据读/写操作通过CAN0DATH 和CAN0DATL 来

完成。例如：如果需要对位定时寄存器重新配置时，只需向CAN0ADR 寄存器中写入0X03，将新配置的数据的低字节写入CAN0DATL中，高字节写入CAN0DATH 中。

### 6.7.4 CAN 控制器应用时寄存器配置

下面就CAN 控制器在应用时，根据所要完成功能的不同而需要做的不同配置做具体描述。这包括报文对象初始化处理、发送对象配置、接收对象配置、中断处理配置；另外，还有发送对象的更新、位定时寄存器等配置。2.1 报文对象初始化处理报文RAM 中的报文对象（除MsgVal、NewDat、IntPnd 和TxRqst）配置不受芯片复位的影响。所有的报文对象在使用前必须由CPU 来初始化为零或者被设置为无效。报文对象的配置是通过相应的接口寄存器来设置其屏蔽码、仲裁场、控制场和数据场值，而这一设置过程由相应的IFX 命令请求寄存器来完成。

当CAN 控制寄存器中的Init 位置零，CAN 内核中的CAN 协议控制器状态机制和报文处理状态机制将控制C_CAN 的内部数据流。接收到的报文通过接收滤波后都存放在报文RAM中，而得到传输请求的报文都要移入CAN 内核的移位寄存器中并通过CAN 总线传出。

### 6.7.5 CAN 发送对象的配置

当报文对象作为发送对象时，仲裁寄存器（ID28-0 和Xtd 位）将被应用，它们定义了即将发送的报文识别符和类型，如果使用11 位识别符（标准帧），那么使用的是ID28～ID18，而ID17～ID0 将被忽视。如果TxIE 位被置位，则IntPnd 位在此报文对象被成功发送后被置位；如果RmtEn 位被置位，在接收到匹配的远程帧将引起TxRqst 位被置位。若数据寄存器（DLC3-0，Data0-7）将被使用，TxRqst 和RmtEn 在数据有效前不会被置位。屏蔽寄存器（Msk28-0、Umask、Mxtd 和MDir 位）可以用来（UMask='1'）允许相同识别符的数据帧组被接收。

### 6.7.6 CAN 中断处理

在所有中断中，状态中断具有最高优先级，报文对象的中断优先级随着报文编号的增大而减小。如果有几个中断产生，那么CAN 中断寄存器将指向优先级最高的中断，而不是按中断先后顺序排列。状态中断通过读取状态寄存器来清除，报文中断通过清除报文对象的IntPnd 位来清除。处于中断寄存器中的中断识别符Intld 能表明中断的原因，如果这个寄存器的值为0，没有中断产生；否则，有中断发生。CPU 控制着状态寄存器的改变是否可以引起中断（CAN 控制寄存器中的EIE 和SIE位）；当中断寄存器的值不为0（CAN 控制寄存器中的IE 位）时中断队列是否有效。CPU 有两种方式判断报文中断源，每一种是判断中断寄存器中的Intld 位；另一种是顺序扫描中断发生寄存器。

### 6.7.7 CAN 控制器的应用

CAN 总线一般用在工业检测和控制现场，它将各功能模块连接在一起组成一个现场级通信网络。在本应用中，CAN 总线完成下位机各部分之间的通信以及各下位机与上位机之间的通信。下位机以单片机C8051F040 为核心，上位机由PC 机构成。下位机的CAN通信物理层的电路在下面将做详细说明，为了完成上位机与下位机的通信，需要外加一块PC-CAN 通信卡。

### 6.7.8 CAN 通信软件实现

此下位机CAN 通信部分主要完成的任务是：将现场检测到的数据传送给上位机或其它的下位机节点；同时，上位机可以对下位机的相关参考进行设置，即下位机还须接收一定量的数据。

由上可知，此节点的CAN 通信主要包括系统初始化程序、发送程序、接收程序等。软件部分设计的好坏将直接决定系统能否正常工作，对于初次接触CAN 总线系统的设计人员来说是一个难点，也是一个重点。在本例中，系统软件采用结构化程序设计方案，使其具有较好的模块性和可移植性，对于不同的系统功能或不同的应用环境，可以方便地进行编程重组。

### 6.7.9 CAN 系统初始化初始化

初始化程序主要完成对所有的报文对象进行初始化（一般将所有值置零），对CAN 控制寄存器（CAN0CN）、位定时寄存器（BITREG）进行设置，还要对发送报文对象和接收报文对象分别进行初始化。其中，位定时寄存器的设置较为复杂，这里我们使用外部晶振为8MHz，CAN 通信速率为500k/s，得到BITREG 的初始值为0x2301。主程序中规定对象初始化、发送和接收初始化，最后才启动CAN 处理机制（对BITREG 和CAN0CN 初始化）。

### 6.7.10 CAN 接口函数流程

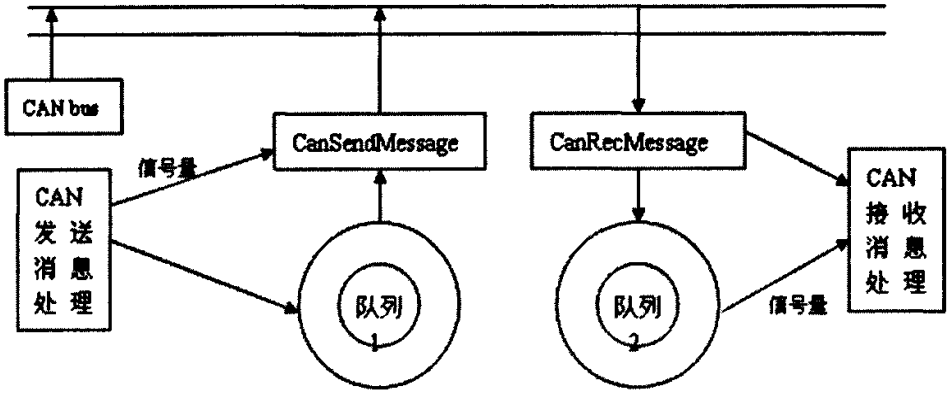


图6.7 Can接口函数

如图6.7所示：

（1） 发送程序

CAN 报文发送是由CAN 控制器自动完成的，用户只需根据接收到的远程帧的识别符，将对应的数据转移到发送缓冲寄存器，然后将此报文对象的编码写入命令请求寄存器启动发送即可，而发送由硬件来完成。这里，我们使用定时更新发送报文对象中的数据，数据的发送有控制器自动完成，当其收到一个远程帧时，就将具有相同识别符的数据帧发送出去。其发送程序结构如下：

```
Void transmit_message(char MsgNum)
```

```
{
SFRPAGE=CAN0_PAGE;/*指向CAN0 页面*/
CAN0ADR=IF1CMDMSK; /*向IF1 命令屏蔽寄存器写入命令*/
CAN0DAT=0X0083;
CAN0ADR=IF1ARB2; /*指向IF1 仲裁寄存器2*/
CAN0DATH|=0x80;
CAN0ADR=IF1DATA1;/*指向数据场的第一个字节*/
for(i=0;i<4;i++)
{
CAN0DAT=can_temp[i];/*将4 字节数据写入发送缓冲器*/
}
CAN0ADR=IF1CMDRQST;
CAN0DATL=MsgNum;
/*将报文对象编号写入，则数据发送到对应的报文对象中*/
}
```

**（2） 接收程序**

CAN 报文的接收与发送一样，是由CAN 控制器自动完成的，接收程序只需从接收缓存器中读取接收的数据，再进行相应的处理即可。其基本方法与发送程序一致，只是接收程序采用中断方式。在此应用中，接收程序主要接收上位机对下位机的参数设置数据，只有当修改时才需要接收数据，所以采用中断方式处理比较合适。接收程序结构如下：

```
void receive_data(void)
{
SFRPAGE=CAN0_PAGE;/*指向CAN0 页面*/
CAN0ADR=IF1CMDMSK; /*向IF1 命令屏蔽寄存器写入命令*/
CAN0DAT=0X0083;
CAN0ADR=IF1ARB2; /*指向IF1 仲裁寄存器2*/
CAN0DATH|=0x80;
CAN0ADR=IF1DATA1;/*指向数据场的第一个字节*/
for(i=0;i<4;i++)
{
```

```
CAN0DAT=can_temp[i];/*将4 字节数据写入发送缓冲器*/
}
CAN0ADR=IF1CMDRQST;
CAN0DATL=MsgNum;
/*将报文对象编号写入，则数据发送到对应的报文对象中*/
}
```

（3） 接收程序

CAN 报文的接收与发送一样，是由CAN 控制器自动完成的，接收程序只需从接收缓存器中读取接收的数据，再进行相应的处理即可。其基本方法与发送程序一致，只是接收程序采用中断方式。在此应用中，接收程序主要接收上位机对下位机的参数设置数据，只有当修改时才需要接收数据，所以采用中断方式处理比较合适。接收程序结构如下：

```
Void receive_data(void)
{
SFPRAGE=CAN0_PAGE;/*指向CAN0 页面*/
CAN0ADR=IF2CMDMSK; /*向IF2 命令屏蔽寄存器写命令*/
CAN0DAT=0x003F;
CAN0ADR=IF2CMDRQST;
/*将报文对象编号写入命令请求寄存器，对应地接收*/
CAN0DATL=MsgNum;/*得到数据就从报文RAM 中移到数据缓冲器中*/
CAN0ADR=IF2DATA1; /*指向数据场的第一个字节*/
for(i=0;i<4;i++){ /*读取4 个字节数据*/
CAN_RX[i]=CAN0DAT;
}
```

### 6.7.11 CAN 驱动函数流程

如图所示，一般情况下最基本的 CAN 总线节点的通信软件包括三个部分:CAN 节点的初始化，消息的传送，消息的接收。首先建立缓冲区，对 CAN 节点进行初始化，主要包括以下几个部分：CAN 引脚输出方式的设置，波特率参数的设置，消息对象的初始化，以及当允许接收或发送中断时对相应中断允

许位的设置。要进入初始化状态，必须设定 CAN 控制寄存器中的 Init 位为 1。设定完成后再复位 Init 位退出初始化状态。初始化程序主要完成对所有的报文对象进行初始化（一般将所有值置零），对 CAN 控制寄存器（CAN0CN）、位定时寄存器（BITREG）进行设置，还要对发送报文对象和接收报文对象分别进行初始化。主程序中规定对象初始化、发送和接收初始化，启动 CAN 处理机制（对 BITREG 和 CAN0CN 初始化），下面为 CAN 启动程序：

```
void StartCAN (uint8 count)
{
    WORD bps;
    Switch (count)                                    // 波特率参数设置
    {.....}
    EIE2 |= 0x20;                                     // 开中断
}
```

CAN 报文发送是由CAN 控制器自动完成的，用户只需根据接收到的远程帧的识别符，将对应的数据转移到发送缓冲寄存器，然后将此报文对象的编码写入命令请求寄存器启动发送即可，而发送由硬件来完成。这里，我们使用定时更新发送报文对象中的数据，数据的发送有控制器自动完成，当其收到一个远程帧时，就将具有相同识别符的数据帧发送出去。

CAN 报文的接收与发送一样，是由CAN 控制器自动完成的，接收程序只需从接收缓存器中读取接收的数据，再进行相应的处理即可。其基本方法与发送程序一致，只是接收程序采用中断方式。在此应用中，接收程序主要接收上位机对下位机的参数设置数据，只有当修改时才需要接收数据，所以采用中断方式处理比较合适。在对缓冲区进行读写过程中，引入读、写两个信号量分别对缓冲区两端的操作进行同步。这样设计的优点在于：当用户任务想写但缓冲区满时，在信号量上睡眠，让CPU运行别的任务，待ISR从缓冲区读走数据后唤醒此睡眠的任务；类似地，当用户任务想读但缓冲区空时，也可以在信号量上睡眠，待外部设备有数据来了再唤醒。由于SmallRTOS51的信号量提供了超时等待机制，CAN口当然也具有超时读写能力。

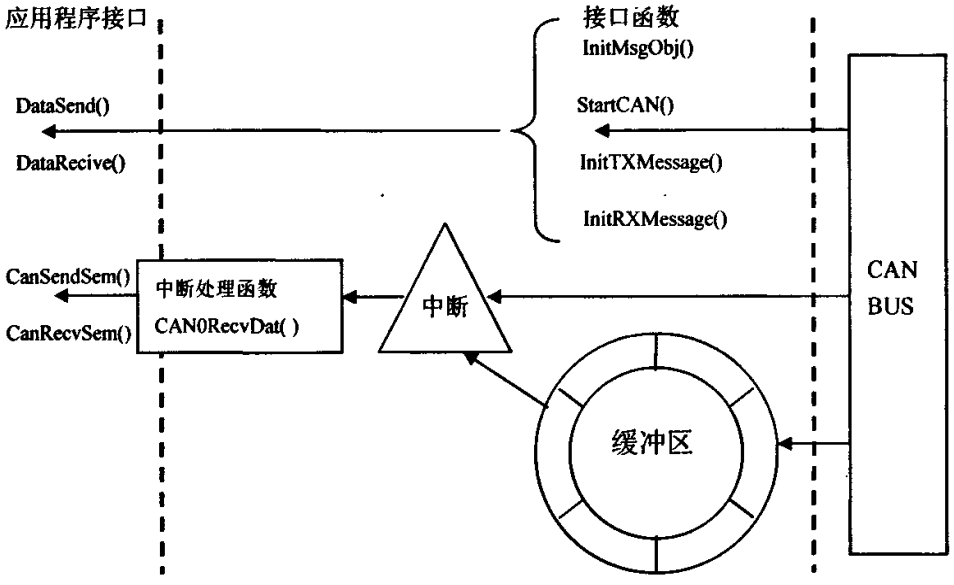


图6.8 CAN总线驱动函数流程

## 6.8 模拟量输入/输出驱动函数的实现

### 6.8.1 模拟量输入/输出的算法及驱动程序流程

模拟电信号输入通道包括两部分，交流信号调理电路和频率测量电路。交流信号调理电路的主要作用是将铁路信号电源的交流电压电流大信号转换成模数转换器可测量的信号范围；频率测量电路的功能是为频率测量提供一路可供测量的电信号。

目前，交流电参数的测量方法有两种，一种是直流采样法，传统的变送器多采用这种方法，直流采样方法是将对经过整流的直流量进行采样，然后对采样值做一次比例变换即得到所测量的值。直流采样存在诸多问题，例如测量精度直接受整流电路的精度和稳定性影响，而整流电路参数调整困难且受波形影响较大。因此，为获得高精度，高稳定性的测量结果，必然采用交流采样法。

交流采样法是按一定规律对被测交流信号的瞬时值进行采样，再用一定的数学算法求得被测量值，用软件功能代替硬件功能的计算方法。交流采样法主

要包括包括同步采样法、准同步采样法、非同步采样法等几种。，同步采样法就是指对交流电信号在一个周期内等间隔均匀采样，即满足关系式$T = N*T_S$，式中 $T$ 为交流电信号的周期，$N$ 为采样点数，$T_S$为采样间隔，但在实现过程中因为种种原因例如定时器的分辨率等因素很难做到严格意义上的同步采样，这时，可通过适当增加采样数据量和增加迭代次数来提高测量准确度，这种方式就是准同步采样法。也可以使用固定的采样间隔，通过软件判断处理调整采样值，使采样周期与信号周期（或信号周期的整数倍）的差值小于一个采样间隔的测量方法，这就是非同步采样法。交流同步采样的实现方法有 2 种：一是硬件同步采样法；二是软件同步采样法。硬件同步采样法采用锁相环跟踪基波频率达到同步采样的目的，而软件法则是通过控制定时器的溢出达到上述目的。硬件法结构复杂，成本高，软件法结构简单，但误差较大，两种方法各有优缺点。

对于交流交流采样法，我采用均方根算法。其基本思路是：

对于连续周期交流信号的电压有效值，电流有效值及有功功率有下式成立

$$U = \sqrt{\frac{1}{T}\int_0^T u^2(t)dt} \tag{5.1}$$

$$I = \sqrt{\frac{1}{T}\int_0^T i^2(t)dt} \tag{5.2}$$

$$P = \frac{1}{T}\int_0^T u(t)i(t)dt \tag{5.3}$$

式中 $u(t),i(t)$分别对应于交流信号的瞬时值，$T$对应于交流信号的周期。对以上积分公式进行离散化处理，推导可得出相对应的计算电压、电流的有效值和平均功率的离散化公式，离散积分公式（均方根算法）

$$U = \sqrt{\frac{1}{N}\sum_{n=1}^{N} u^2(n)} \tag{5.4}$$

$$I = \sqrt{\frac{1}{N}\sum_{n=1}^{N} i^2(n)} \tag{5.5}$$

$$P = \frac{1}{N}\sum_{n=1}^{N} u(n)i(n) \tag{5.6}$$

$$COS\Phi = \frac{P}{UI} \qquad (5.7)$$

通过均方根算法要计算出交流电信号的电流，电压有效值，首先必须确定每个周期的采样点数 $N$，然后在一个整周期内均匀采样 $N$ 点，获得 $N$ 个采样值。为此，我们必须首先对交流电信号的周期进行测量。

周期测量的基本原理是，交流电信号首先经过测频通道转换成矩形波，然后送入片内可编程计数阵列(PCA,Programmable Counter Array)进行测量，通过软件设置片内 PCA 工作在上跳沿触发状态，每一次波形的上跳沿都会触发一次捕获，将当前 PCA 的计数值送到捕获寄存器，同时引发一次中断，在中断服务子程序中将本次捕获的计数值与前次捕获的计数值相减即得到对应的交流电信号的周期。

测量得到的周期经过 N 等分后送入定时器作为初始值，定时器工作在自动重装状态，每次溢出都会触发一次中断，在中断服务子程序中依次对电压电流进行模数转换，得到对应的测量值，将测量值存入对应的环形采样缓冲中，每当完成 32 个采样点的采样存储后，即置位处理标志位；主程序循环查询处理标志位，当标志位置起时，即使用均方根算法计算得出电流，电压有效值和有功功率等电力参数。

模拟量驱动程序的流程如图 6-11 所示。

实例如下：

```
void TASKA(void)                /*数据处理计算任务*/
{   uint8 i;
    OSSemCreate(DISPLAY_SEM,1);         /*创建用于显示的信号量*/
    while(1)
    {   OSSemPend(DISPLAY_SEM,0);       /*等待信号量*/
        for(i=0;i<4;i++)
          {   DataPrePro();             /*数据处理*/   }
        OSTimeDly(5);
     }
}
```

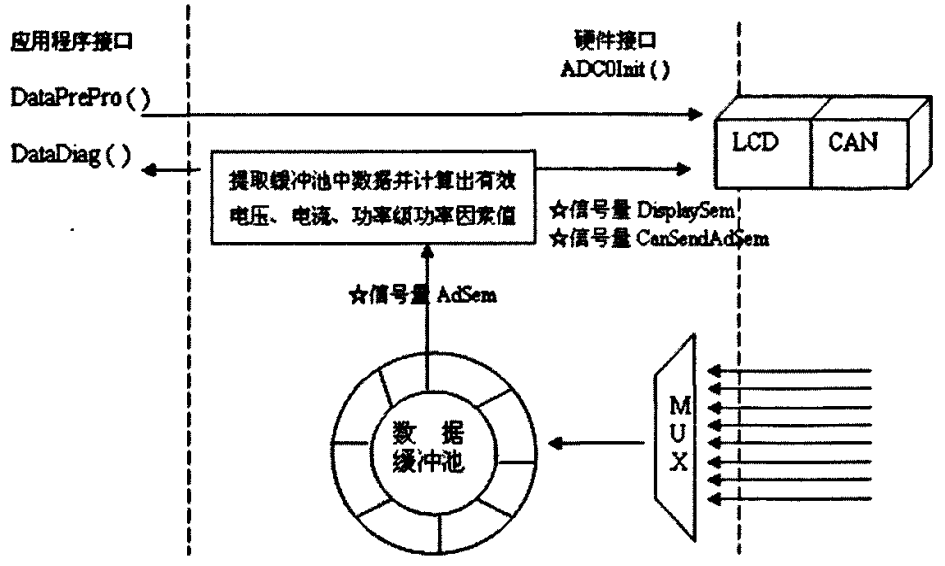


图 6-11 模拟量驱动程序的流程

```
void TASKB(void)                          /*显示计算后的电流电压值*/
{   uint8 temp; VAL * pVal;
    while (1)
    { GetTime(time);    if(valPtr==0)                 /*校正显示值的指针*/
      temp=39;
       else     temp=valPtr-1;
       pVal=&val[temp];
      DisAStr(1,2,"<System OverView>");DisAStr(2,1,"V[");
      DisANum(pVal->volInt,3);          LcdWDat('.');
      DisANum(pVal->volFrac,1);         DisAStr(2,8,"]V I[");
      DisANum(pVal->curInt,2);          LcdWDat('.');
      DisANum(pVal->curFrac,2);         DisAStr(2,17,"]A");
      DisAStr(3,1,"                     ");       DisAStr(3,1,"P[");
      DisANum(pVal->powInt,1);LcdWDat('.');DisANum(pVal->powFrac,2);
      DisAStr(3,7,"]KW");     DisAStr(4,1,"                          ");
```

```
        DisAStr(4,1,"PF[");        DisANum(pVal->powFac/100,1);
       LcdWDat('.');    DisANum(pVal->powFac-(uint8)(pVal->powFac/100)*100,2);

        DisAStr(4,8,"] T[");         DisANum(*time,2);
        LcdWDat(':');          DisANum(*(time+1),2);
         LcdWDat(':');          DisANum(*(time+2),2);
         LcdWDat(']');    OSTimeDly(10);
      }
}
/**************TIMER3ISR 中断服务子程序*****************************/
#pragma disable
/* 除非最高优先级中断或是不需要系统管理的中断，否则，必须加上这一句   */
void Timer3ISR (void) interrupt 14
{   static uint16 sampIn=1;              OS_INT_ENTER();
     /* 中断开始处理*/
    TF3=0;
    SFRPAGE = ADC0_PAGE;
    AMX0SL = 0x00;      // Select adc0.0 sens as ADC mux output
    AD0BUSY=1;
    while (~AD0INT);                        // wait for conversion complete
      AD0INT=0;
      *(volSampAry+sampIn)= ADC0;               // read voltage ADC value
      AMX0SL= 0X01;            // Select adc0.1 sens as ADC mux output
      AD0BUSY=1;
    while(~AD0INT);                           // wait for conversion complete
      AD0INT=0;
      *(curSampAry+sampIn)= ADC0;               // read current ADC value
      sampIn++;
      if(sampIn==320)
      {sampIn=0,CanNcFlag=0;}       if(sampIn%64==0)
      {    OSSemIntPost(DISPLAY_SEM);     }
```

```
    OSIntExit();
}
/********采样数据计算*****************************************/
void DataPrePro(void)
{
    uint8 i;
    uint16 volSampDat,curSampDat;
    uint32 temp0,temp1,sum=0,sum0=0,sum1=0,sum2=0,sum3=0;
    for(i=0;i<16;i++)
      {
        volSampDat=*(volSampAry+sampOut); //取出电压采样值
        curSampDat=*(curSampAry+sampOut); //取出电流采样值
if(volSampDat>=0x0800&&curSampDat>=0x0800||volSampDat<0x0800&&curSam
pDat<0x0800)
    { //电流电压均大于 2048
      if(volSampDat>=0x0800)          //电压采样值超过 2048
        {     temp0 = volSampDat-0x0800;//电压间隔为-0x800
              temp1 = curSampDat-0x0800;
        }
       else{    temp0 = 0x0800-volSampDat;
              temp1 = 0x0800-curSampDat;}
         sum0+=temp0*temp0;//电压与电压相乘
         sum1+=temp1*temp1;//电流与电流相乘
         sum2+=temp0*temp1;//电压與电流相乘}
        else //否则.....(与上同)
         { if(volSampDat>=0x800)
            { temp0 = volSampDat-0x0800;
              temp1 = 0x0800-curSampDat;}
          else {temp0 = 0x0800-volSampDat;
                temp1 = curSampDat-0x0800; }
          sum0+=temp0*temp0;
```

```
            sum1+=temp1*temp1;
            sum3+=temp0*temp1; }
        sampOut++;   if(sampOut==320) sampOut=0;}
    sum0=sqrt(sum0>>4);
//左移 1 乘 2,右移 1 除 2;calculate the valid value of voltage 计算电压有效值
    sum1=sqrt(sum1>>4);
    if(sum2>=sum3)
       sum=(sum2-sum3)>>4;     //calculate the valid power 计算功率有效值
    else
       sum=0;
    val[valPtr].powFac=(uint8)(sum*100/(sum0*sum1));   //第一个数组里面的功率因素;计算功率因素
    sum0=(sum0*100*24)>>11; //求真实电压基准
    val[valPtr].volInt=sum0/10;   /*电压有效值*/
    valPtr++;
    if(valPtr==40)   {valPtr =0; }}
```

## 6.8 文件结构

为了使 Small RTOS51 内核、API 和应用程序之间有一个清晰的结构，把所有源文件统一在 1 个文件夹 Small RTOS51 中。在 Small RTOS51 文件夹中再定义 5 个文件夹和 1 个 readme 文本文件，即：H、KEILC51、OS、API、MAIN 和 readme。H 文件夹保存 Small RTOS51 的配置头文件、相关的 C 语言头文件及与 CPU 相关的宏的源码；KEILC51 文件夹保存和处理器有关的 2 个函数（指定 MCU 的移植）；OS 文件夹保存 Small RTOS51 与 CPU 无关的核心代码；API 文件夹保存应用 api 函数；MAIN 文件夹保存具体项目的应用文件；readme 对整个文件结构做详细的说明。

这样清晰的结构，便于把本设计移植到不同的开发项目中，而且有利于代码的维护。

## 6.9 本章小结

主要是借助具体的应用实例，重点研究在 RTOS 平台下应用软件的设计的思路和解决问题的方法。

# 第7章 结论

本文以测控系统的终端检测单元为应用背景，在Small RTOS51实时软件平台下开发了系统外围器件的驱动程序。主要研究了Small RTOS51软件平台的搭建和驱动程序的设计方法。通过RTOS平台下和传统前/后台系统软件设计思想的对比可以看出，RTOS内核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，将一个复杂的问题划分成许多子问题，用任务来实现，通过给任务分配不同的优先级来保证系统首先处理实时性要求最高的任务。另外，由于加入低优先级的任务不会影响高优先级任务的执行，只需简单地加入任务，就可以进行系统扩展。

本系统所有接口函数的设计都遵守软件复用的原则，只要稍做改动即可移植到其它的处理器或控制器中。

实时内核的应用是一个逐渐探索和经验积累的过程，本文只是在Small RTOS51软件平台完成了应用软件的开发，对于系统性能的总体性把握（例如优先级设置、任务分解的方法）并没有做深入的研究。

# 致谢

本文是在王东兴教授的悉心指导下完成的。王教授渊博的专业知识、严谨的治学作风、孜孜不倦的创新精神、敏锐的科学洞察力和深入透彻的分析能力都给我留下了深刻的印象，使我受益匪浅，必将在我今后的学习、工作和生活中产生很大的影响，让我终身受益；王教授忘我的工作热忱、真诚的待人态度、博大无私的胸怀更值得我永远学习。在此论文完成之际，谨向恩师致以诚挚的谢意。

感谢在我求学生涯中给予我关怀和教诲的双亲，他们纯朴的生活作风和坚强的拼搏精神一直是我前进的动力，终将伴我远行。

最后感谢所有曾给予我帮助、支持、关心和鼓励的老师、同学和朋友们。

# 参考文献

[1] 林建民 嵌入式操作系统技术发展趋势 计算机工程 2001 27（10）：1-4
[2] 宋延昭 嵌入式操作系统介绍及选型原则 工业控制计算机 2005（7）：41
[3] 朱巍 嵌入式实时操作系统及其应用 单片机与嵌入式系统应用 2002（8）：5-7
[4] 刘波，马连川等 嵌入式实时操作系统选用的初步分析 北方交通大学学报 2000 24（5）：105-108
[5] （美）拉伯罗斯著，邵贝贝等译 嵌入式实时操作系统 北京航空航天大学初版社 2003.5
[6] 涂刚，阳富民等 嵌入式操作系统综述 计算机应用研究 2000（11）：4-5
[7] 洪英，陈曦 嵌入式实时多任务程序设计 计算机应用 2000 20（7）：10-12
[8] C8051F040/1/2/3 混合信号 ISP FLASH 微控制器数据手册 潘琢金译 2002.10
[9] RTL8019AS 用户手册 台湾 Realtek 公司
[10] http：//web51.hw-server.com/Web51-Hardware-DEDIGN KIT
[11] 徐爱钧，彭秀华 Keil Cx51 V7.0 单片机高级语言编程与 μVision2 应用实践 电子工业出版社 2004.6
[12] 周世杰，秦志光等 实时软件系统开发技术 小型微型计算机系统 2004.1
[13] Hassan Gomaa. Software design methods for the design of large scale real-time systems[J] Systems Software, 1994,25(2): 127-146
[14] Gomma H. A software design method for real-time systems [J] Computer Practices,1984
[15] （美）拉伯罗斯著，袁勤勇等译 嵌入式系统构件 机械工业出版社 2002.2。