中南大学

硕士学位论文

嵌入式实时操作系统μC/OS在C51平台上的移植及应用

姓名：娄小平

申请学位级别：硕士

专业：计算机技术

指导教师：陈志刚

20090509

摘 要

随着现代计算机技术的迅速发展，嵌入式操作系统以其简洁、高效等优点扮演了越来越重要的角色。嵌入式产品已经成为信息产业的主流，被广泛应用于移动计算设备、网络设备、工控设备、信息家电、汽车电子、娱乐设备、仪器仪表等领域。

依据经济成本考虑，本项目没有去购买昂贵的商业RTOS（实时操作系统），而是选择了源代码公开、体积小且可裁剪移植性好的 μC/OS-II，选择C51微处理器为应用平台，根据实际项目要求把μC/OS-II移植到C51单片机上。移植的过程主要集中在三个文件的重新编写上：一个头文件OS_CPU.H、一个C代码文件OS_CPU_C.C和一个汇编文件OS_CPU_A.ASM文件。还针对C51的小内存所做的优化，对可重入问题的进行了分析与解决，基本达到应用的需要。最后以 μC/OS-II 实时内核为基础，把它应用于智能验钞机系统，运用了三大检伪手段（磁性检测、荧光检测和宽度检测），实现了智能验钞机启停监测、按键监控、信息处理、数据输出和参数设置等多任务的设计目的。基于 μC/OS-II 内核的程序，进行多任务的分配和调用，构建智能验钞机的软件构架，结合外部中断服务程序，具体的完成项目中软件应用的部分，形成了一个具有实际意义的嵌入式系统应用软件。

嵌入式实时操作系统 μC/OS-II 具有多任务处理等特点，将其应用到实际项目智能验钞机上，使用可剥夺性的实时内核，对验钞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。且移植过程使用 C 语言开发，简化了程序的设计过程，降低了开发难度，提高了开发效率、可维护性和可移植性。但还存在一些不足和需要改进和地方。文章最后对这些方面做了阐述，提出了改进思路和方法，为下一步的工作提供了有意义的参考。

关键词 嵌入式系统，μC/OS-II，移植

ABSTRACT

With modern computer technology to the rapid development of embedded operating system for its simplicity, efficiency, etc. played an increasingly important role. Embedded products have become the mainstream of the information industry, are widely used in mobile computing devices, network equipment, industrial equipment, information appliances, automotive electronics, entertainment equipment, instrumentation and other fields.

Basis for considering the economic costs, the project did not go to buy expensive commercial RTOS (real-time operating system), but opted for the open source code, small size and portability can be a good cut μC / OS-II, to choose for the application of microprocessor C51 platform, based on actual project requirements of the μC / OS-II ported to the C51 single-chip microcomputer. Transplantation mainly concentrated in the process of re-preparation of three documents: one header file OS_CPU.H, a C code file OS_CPU_C.C and a compilation of documents OS_CPU_A.ASM documents. C51 also made a small memory optimization, re-entrant to the problem are analyzed and resolved to meet the basic needs of the application. Finally, μC / OS-II real-time kernel-based, it applies to smart Detector systems, the use of three pseudo-seizure means (magnetic detection, fluorescence detection and the width of detection), the realization of a smart start and stop monitoring Detector , keystroke monitoring, information processing, data output and parameter settings are designed to multi-task. Based on μC / OS-II kernel procedures, conduct multi-task allocation and call to build the framework of Intelligent Detector software, combined with the external interrupt service routine, the specific software applications to complete the project in part, to form a meaningful embedded system applications.

Embedded real-time operating system μ C / OS-II has features such as multi-tasking to deal with its application to the actual project on Intelligent Detector, can be deprived of the use of real-time kernel, the task demanding all the time when the events have been as far as possible, fast, effective treatment. Transplantation and the use of C language development process and simplifies the process of the design process and reduce the development of the difficulty of the development of improved efficiency, maintainability and portability. However, there are still some shortcomings and need to be improved and local. Finally, the article done on these areas and put forward ideas and methods to improve for the next step to provide a meaningful work of reference.

KEY WORDS embedded system, μC / OS-II, transplantation

原创性声明

本人声明，所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了论文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得中南大学或其他单位的学位或证书而使用过的材料。与我共同工作的同志对本研究所作的贡献均已在论文中作了明确的说明。

作者签名：姜小平　　日期：2009年5月9日

学位论文版权使用授权书

本人了解中南大学有关保留、使用学位论文的规定，即：学校有权保留学位论文并根据国家或湖南省有关部门规定送交学位论文，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以采用复印、缩印或其它手段保存学位论文。同时授权中国科学技术信息研究所将本学位论文收录到《中国学位论文全文数据库》，并通过网络向社会公众提供信息服务。

作者签名：姜小平　导师签名　日期：2009年5月9日

第一章 绪论

1.1 嵌入式操作系统简介

一、嵌入式系统

嵌入式系统是以应用为中心，软硬件可裁减的，适用于对功能、可靠性、成本、体积、功耗等综合性严格要求的专用计算机系统[1]。具有软件代码小、高度自动化、响应速度快等特点，特别适合于要求实时和多任务的体系。嵌入式系统主要由嵌入式处理器、相关支撑硬件、嵌入式操作系统及应用软件系统等组成，它是可独立工作的“器件”。嵌入式操作系统与通用平台操作系统不同，它往往嵌入到机器或设备内部运行，一般用户无法看到它的运行界面[2]。

二、嵌入式操作系统

作为嵌入式系统（包括硬、软件系统）极为重要的组成部分的嵌入式操作系统，通常包括与硬件相关的底层驱动软件、系统内核、设备驱动接口、通信协议、图形界面、标准化浏览器等。嵌入式操作系统具有通用操作系统的基本特点，如能够有效管理越来越复杂的系统资源；能够把硬件虚拟化，使得开发人员从繁忙的驱动程序移植和维护中解脱出来；能够提供库函数、驱动程序、工具集以及应用程序。与通用操作系统相比较，嵌入式操作系统在系统实时高效性、硬件的相关依赖性、软件固态化以及应用的专用性等方面具有较为突出的特点[3]。

1、与其他类型的操作系统相比，嵌入式操作系统具有以下一些特点[4][5]。

(1)体积小。嵌入式系统有别于一般的计算机处理系统，它不具备像硬盘那样大容量的存储介质，而大多使用闪存（Flash Memory）作为存储介质。这就要求嵌入式操作系统只能运行在有限的内存中，不能使用虚拟内存，中断的使用也受到限制。因此，嵌入式操作系统必须结构紧凑，体积微小。

(2)实时性。大多数嵌入式系统都是实时系统，而且多是强实时多任务系统，要求相应的嵌入式操作系统也必须是实时操作系统(RTOS)。实时操作系统作为操作系统的一个重要分支已成为研究的一个热点，主要探讨实时多任务调度算法和可调度性、死锁解除等问题。

(3)特殊的开发调试环境。提供完整的集成开发环境是每一个嵌入式系统开发人员所期待的。一个完整的嵌入式系统的集成开发环境一般需要提供的工具是编译/连接器、内核调试/跟踪器和集成图形界面开发平台。其中的集成图形界面开发平台包括编辑器、调试器、软件仿真器和监视器等。

2、嵌入式操作系统伴随着嵌入式系统的发展经历了四个比较明显的阶段：

第一阶段：无操作系统的嵌入算法阶段，以单芯片为核心的可编程控制器形

式的系统，具有与监测、伺服、指示设备相配合的功能。应用于一些专业性极强的工业控制系统中，通过汇编语言编程对系统进行直接控制，运行结束后清除内存。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。

第二阶段：以嵌入式 CPU 为基础、简单操作系统为核心的嵌入式系统。CPU 种类繁多，通用性比较差；系统开销小，效率高；一般配备系统仿真器，操作系统具有一定的兼容性和扩展性；应用软件较专业，用户界面不够友好；系统主要用来控制系统负载以及监控应用程序运行。

第三阶段：通用的嵌入式实时操作系统阶段，以嵌入式操作系统为核心的嵌入式系统。能运行于各种类型的微处理器上，兼容性好；内核精小、效率高，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）；嵌入式应用软件丰富。

第四阶段：以基于 Internet 为标志的嵌入式系统。这是一个正在迅速发展的阶段。目前大多数嵌入式系统还孤立于 Internet 之外，但随着 Internet 的发展以及 Internet 技术与信息家电、工业控制技术等结合日益密切，嵌入式设备与 Internet 的结合将代表着嵌入式技术的真正未来。

三、嵌入式实时操作系统

C.M.Krishna 和 Kang G.Shin 在实时系统设计中对实时系统所作的一个定义[6]：“任何一个对外部激励信号需要有一个确定的时间响应的计算机系统就是个实时系统。”在这里，一个确定的时间响应的意思就是，在一个实时系统中运行的任务必须有一个死限(Deadline)。实时操作系统可进一步分为两种，一种称为“硬实时”，另一种则称为“软实时”[7]。在硬实时操作系统中，各任务不仅要执行无误而且要做到准时。而软实时操作系统中系统的宗旨是使各个任务运行得越快越好，并不要求限定某一任务必须在多长时间内完成。大多数实时操作系统使两者得结合[2]。

嵌入式实时操作系统在目前的嵌入式应用中用得越来越广泛，尤其在功能复杂、系统庞大的应用中显得愈来愈重要。

1. 嵌入式实时操作系统提高了系统的可靠性。在控制系统中，出于安全方面的考虑，要求系统起码不能崩溃，而且还要有自愈能力。不仅要求在硬件设计方面提高系统的可靠性和抗干扰性，而且也应在软件设计方面提高系统的抗干扰性，尽可能地减少安全漏洞和不可靠的隐患。长期以来的前后台系统软件设计在遇到强干扰时，使得运行的程序产生异常、出错、跑飞，甚至死循环，造成了系统的崩溃。而实时操作系统管理的系统，这种干扰可能只是引起若干进程中的一

个被破坏，可以通过系统运行的系统监控进程对其进行修复。通常情况下，这个系统监视进程用来监视各进程运行状况，遇到异常情况时采取一些利于系统稳定可靠的措施，如把有问题的任务清除掉。

2．提高了开发效率，缩短了开发周期。在嵌入式实时操作系统环境下，开发一个复杂的应用程序，通常可以按照软件工程中的解耦原则将整个程序分解为多个任务模块。每个任务模块的调试、修改几乎不影响其他模块。商业软件一般都提供了良好的多任务调试环境。

3．提高了执行效率。虽然引入操作系统，会带来额外的 ROM/RAM 的开销以及 2-4%的 CPU 资源，但并不意味着执行效率的降低。例如，如果程序需要等待一段时间，一般用程序延时或定时器延时。无论何种方法，CPU 不再工作效率很低，而用操作系统，等待的时候 CPU 可以处理其他工作，从而提高的执行效率。

1.2 国内外嵌入式操作系统的发展与现状

国外嵌入式操作系统已经从简单走向成熟，主要有 Windows CE、PalmOS、QNX、 Vxwork、嵌入式 Linux、μC/OS-II 等。国内的嵌入式操作系统研究开发有 2 种类型，一类是基于国外操作系统二次开发完成的，如海信的基于 Windows CE 的机顶盒系统；另一类是中国自主开发的嵌入式操作系统，如凯思集团公司自主研制开发的嵌入式操作系统 Hopen OS（“女娲计划”）等[8]。

Windows CE 内核较小，能作为一种嵌入式操作系统应用到工业控制等领域。其优点在于便携性、提供对微处理器的选择以及非强行的电源管理功能。内置的标准通信能力使 Windows CE 能够访问 Internet 并收发 email 或浏览 Web。除此之外，Windows CE 特有的与 Windows 类似的用户界面使最终用户易于使用。Windows CE 的缺点是速度慢、效率低、价格偏高、开发应用程序相对较难。

3Com 公司的 Palm OS 在掌上电脑和 PDA 市场上独占其霸主地位，它有开放的操作系统应用程序接口(API)，开发商可根据需要自行开发所需的应用程序。

QNX 是由加拿大 QSSL 公司开发的分布式实时操作系统,它由微内核和一组共操作的进程组成,具有高度的伸缩性,可灵活地剪裁，最小配置只占用几十 KB 内存。因此，可以广泛地嵌入到智能机器、智能仪器仪表、机顶盒、通讯设备、PDA 等应用中去。

Hopen OS 是凯思集团自主研制开发的嵌入式操作系统，由一个体积很小的内核及一些可以根据需要进行定制的系统模块组成。其核心 Hopen Kernel 一般为 10KB 左右大小，占用空间小，并具有实时、多任务、多线程的系统特征。

WindRiver 公司的 VxWorks 支持各种工业标准,包括 POSIX、ANSI C 和 TCP/IP 网络协议。VxWorks 运行系统的核心是一个高效率的微内核，该微内核支持各种

实时功能，包括快速多任务处理、中断支持、抢占式和轮转式调度。微内核设计减轻了系统负载并可快速响应外部事件[9]。

常见的嵌入式 linux 系统，包括 RT-Linux、uClinux、Embedix、XLinux、PoketLinux、MidoriLinux,以及我们国家的红旗嵌入式 linux。除了智能数字终端领域以外，Linux 在移动计算平台、智能工业控制、金融业终端系统，甚至军事领域都有着广泛的应用前景。这的源码公开，可以根据需要修改。但由于它是从 Linux 演化而来。并非一开始就是做为嵌入式操作系统的，因而大多是非占先式调度策略，实时性不好。但随着研究的深入已出现实时内核的 Linux 操作系统。内核比较复杂，一般用于 32 位芯片的场合。

μC/OS-II 是由 Jean. labrasse 于 1992 年编写的一个嵌入式实时多任务操作系统。最早这个系统叫做μC/OS，后来经过多年的应用和修改，在 1999 年 Jean. labrasse 推出了μC/OS-II，并在 2000 年得到了美国联邦航空管理局的标准认证，说明了μC/OS-II 具有足够的稳定性和安全性。μC/OS-II 是用 ANSI 的 C 语言编写的，包含一小部分汇编代码，使之可以供不同架构的微处理器使用。至今，从 8 位到 64 位，μC/OSII 已在超过 40 种不同架构的微处理器上运行[10]。

1.3 课题的背景与意义

虽然现在嵌入式操作系统的发展到了基于 Internet 为标志的嵌入式系统，但并不意味着，处于第二、三阶段的嵌入式系统已经十分完善了。正如尽管目前 32 位的嵌入式芯片已成为发展趋势，然而 8 位芯片因其低价、高效、稳定、能耗低等不可替代的价值，一直占相当的市场份额。8 位芯片中 8051 系列单片机是目前在嵌入式系统领域中，应用最为广泛的微处理机之一。

目前，基于 51 单片机的 RTOS 中比较有名气的有 Keil C51 所带的 RTX Full 和 RTX Tiny。以下对这此进行简单的介绍。

RTX51 是一个用于 8051 系列单片机的多任务实时操作系统。有两个不同的 RTX51 版本可以利用。其中 RTX51 Full 使用四个任务优先权完成同时存在时间片轮转调度和抢先的任务切换。RTX51 工作在与中断功能相似的状态下，信号和信息可以通过邮箱系统在任务之间互相传递。您可以从一存储池中分配和释放内存；可以强迫一个任务等待中断、超时，或者是从另一个任务或中断发出信号、信息。而 RTX51 Tiny 是一个 RTX51 的子集，可以很容易地在没有任何外部存储器的单片 8051 系统上运转；但它仅支持时间片轮转任务切换和使用信号进行任务切换（即非抢占式的），不支持抢占式的任务切换，不包括消息队列，没有存储器池分配程序。

8051 系列单片机一般只有很少的 ROM 和 RAM 资源，如 P89C51 只有 4 KB Flash

和 128 字节 RAM。但 RTX51 Full 自身代码有 6 K 多字节，且需要大量外部 RAM，又无源代码，很多时候不实用，不利于学习。RTX Tiny 虽然小（自身占用 900 多字节 ROM），但是任务没有优先级和中断管理，也无源代码，也不太实用（目前 Keil 已经把 RTX Tiny 的源码提供给其正版用户，全部是汇编代码）。

因此，构建一种基于 8051 的嵌入式实时操作系统有着十分重要的应用价值。

本文的工作内容就是在对 μC/OS-II 研究和分析的基础上，以 μC/OS-II 实时内核为基础并对其进行改进，把它应用于智能验钞机系统，同时在智能验钞机硬件平台上移植 μC/OS-II 嵌入式操作系统，同时设计基于 μC/OS-II 内核的程序，进行多任务的分配和调用，构建智能验钞机的软件构架，并在多任务分配的基础上，在各个任务中调用底层的驱动模块的 API 函数，结合外部中断服务程序，具体的完成项目中软件应用的部分，按照项目要求形成一个具有实际意义的嵌入式系统应用软件。

1.4 本文的主要工作及内容安排

本文围绕 μC/OS-II 嵌入式操作系统进行分析研究，在深入理解 μC/OS-II 嵌入式实时操作系统的原理和结构的基础上，在验钞机系统中实际应用，其中包括将 μC/OS-II 嵌入式操作系统进行移植，并在移植的基础上，根际应用需要，设计和划分多任务，形成系统软件的构架，同时调用底层驱数，在每个任务中完成具体应用部分。研究过程中，采取了理论分析和明相结合的方法，验证了研究和工作的正确性和可行性。论文总共分为五章，结构安排如下：

第 1 章绪论概述嵌入式系统的发展过程和分类，及其在各领域内的应用，以及嵌入式系统的发展现状。

第 2 章分析和研究 μC/OS-II 嵌入式操作系统的原理和结构，以此作为项目的理论基础，并重点分析了 μC/OS-II 操作系统的任务调度、时钟管理、内存管理、任务通信、可重入问题的分析与解决及中断系统设计。

第 3 章主要论述操作系统在 C51 系统上的移植，使用开发平台 Keil C 、将 μC/OS-II 与移植相关的文件进行了修改，详细论述了移植过程中具体实现和优化措施。

第 4 章主要论述了智能验钞机多任务实时系统应用软件的开发，μC/OS-II 在该应用项目上的配置，驱动程序的设计。

第 5 章总结全文内容，提出了本课题有待于进一步深入研究的问题，并展望嵌入式系统的研究发展趋势。

第二章　嵌入实时操作系统μC/OS-II内核解析

内核是提供系统运行的基本功能和基本操作的一组程序模块，是对硬件处理器及有关资源进行的首次改造，以便给进程的执行提供良好的运行环境。内核是提供系统运行的基本功能和基本操作的一组程序模块，是对硬件处理器及有关资源进行的首次改造，以便给进程的执行提供良好的运行环境。

μC/OS-II包括任务调度、时钟和中断管理、内存管理、资源管理（信号量、邮箱、消息队列）四大部分，没有文件系统、网络接口、输入输出界面。有64个优先级，系统占8个，用户可以创建56个任务，不支持时间片轮转。它的工作核心原理就是“近似的每时每刻总是让优先级最高的就绪任务处于运行状态”。

2.1　μC/OS-Ⅱ软件体系结构

嵌入式实时操作系统μC/OS-II内核采用模块化结构，使得μC/OS-II结构清晰，源代码可读性好，给操作系统的移植也带来方便。

μC/OS-Ⅱ的结构以及它与硬件的关系如图2-1[10]所示，其文件结构可以分为三个部分，即移植μC/OS-Ⅱ（与处理器类型有关的代码）、μC/OS-Ⅱ（与处理器类型无关的代码）和μC/OS-Ⅱ配置文件（与应用程序有关）。

从图中可以看出，在应用程序中使用μC/OS-Ⅱ时需要用户提供的是应用软件和μC/OS-Ⅱ的配置部分。本章对μC/OS-Ⅱ系统的内核解析只涉及μC/OS-Ⅱ（与处理器类型无关的代码）和μC/OS-Ⅱ配置文件（与应用程序有关）两大模块。与处理器相关的代码在OS_CPU.H（包括用#define设置一些常量的值，声明的数据类型和用#define声明的宏），OS_CPU_C.C（用C语言编写的简单函数）和OS_CPU_A.ASM（编写的汇编语言函数）三个文件中。这些将在下一章移植的实现中具体阐述。

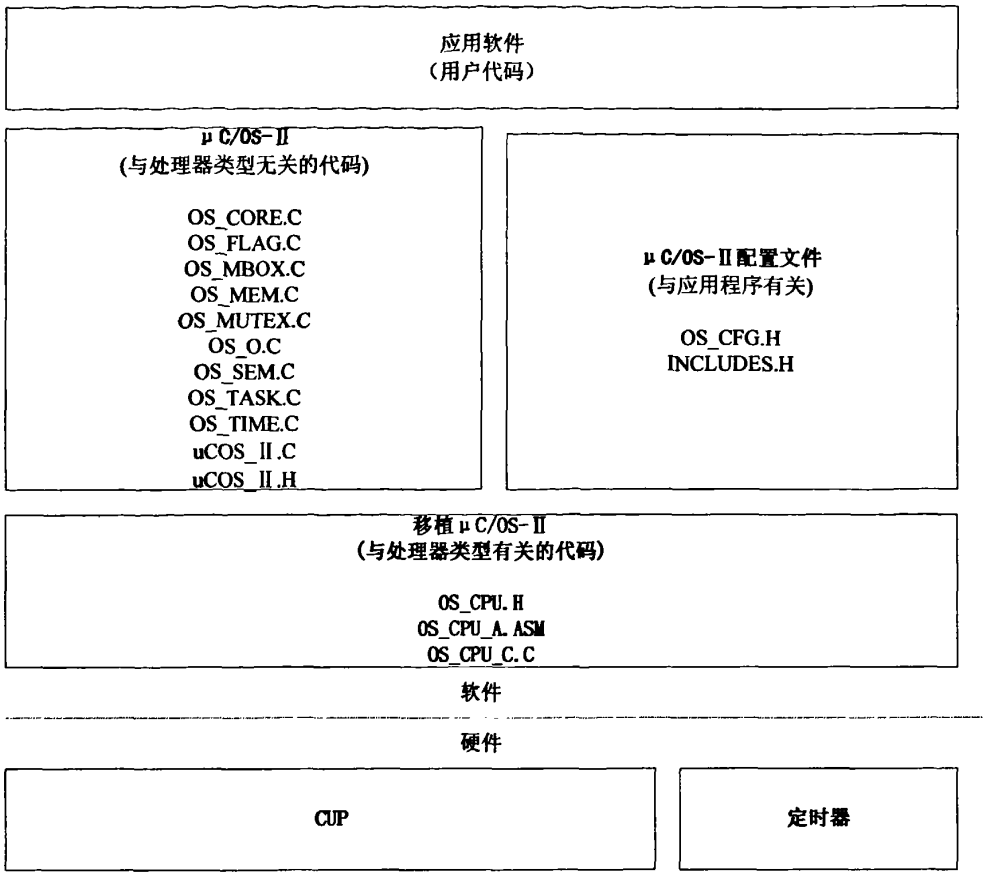


图 2-1　μC/OS-Ⅱ硬件/软件结构图

2.2　μC/OS-Ⅱ实时内核

2.2.1 任务调度

μC/OS-Ⅱ提供多任务的能力。多任务处理是调度的过程和几个任务之间切换 CPU；单 CPU 在几个有序的任务之间进行切换。多任务处理提供构造应用成为一组小的，专注的共享处理器的任务的能力。多任务处理的最重要的方面之一是允许应用程序开发者管理实时应用固有的复杂性。μC/OS-Ⅱ可以使应用程序更容易设计和维护。任务是一个简单的程序，可以认为它完全占有 CPU。实时应用程序的设计过程包括把问题分割成为多个任务，每个任务负责完成问题的一部分。μC/OS-Ⅱ允许创建多达 254 个应用任务。对于许多嵌入式系统来说，254 个任务可以用于复杂的产品设计。

一、任务

任务，也称作线程，是一个简单的程序，该程序可以认为 CPU 完全只属于该程序自己。实时应用程序的设计过程，包括如何把问题分割成多个任务，每个任

务都是整个应用的某一部分，每个任务都被赋予一定的优先级，有它自己的一套 CPU 寄存器和自己的栈空间。典型地、每个任务都是一个无限的循环。在 μC/OS-II 中，总是运行进入就绪态任务中优先级最高的那一个。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（Scheduler）完成的。

二、任务状态

图 2-2[10]是 μC/OS-II 控制下的任务状态转换图。在任一给定的时刻，任务的状态一定是在这五种状态之一。

1、睡眠态（DORMANT）

指任务驻留在程序空间之中，还没有交给 μC/OS-II 管理。把任务交给 μC/OS-II 是通过调用下述两个函数之一：OSTaskCreate()或 OSTaskCreateExt()。任务的建立可以是在多任务运行开始之前，也可以是动态地被一个运行着的任务建立。

2、就绪态（READY）

当任务一旦建立，这个任务就进入就绪态准备运行。如果一个任务是被另一个任务建立的，而这个任务的优先级高于建立它的那个任务，则这个刚刚建立的任务将立即得到 CPU 的控制权。一个任务可以通过调用 OSTaskDel()返回到睡眠态，或通过调用该函数让另一个任务进入睡眠态。

3、运行态（RUNNING）

调用 OSStart()可以启动多任务。OSStart()函数运行进入就绪态的优先级最高的任务。就绪的任务只有当所有优先级高于这个任务的任务转为等待状态，或者是被删除了，才能进入运行态。

4、等待状态（WAITING）

正在运行的任务可以通过调用两个函数之一将自身延迟一段时间，这两个函数是 OSTimeDly()或 OSTimeDlyHMSM()。这个任务于是进入等待状态，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()使延迟了的任务进入就绪态。

正在运行的任务期待某一事件的发生时也要等待，手段是调用以下 3 个函数之一：OSSemPend()，OSMboxPend()，或 OSQPend()。调用后任务进入了等待状态（WAITING）。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。当事件发生了，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。

5、中断服务态（ISR）

正在运行的任务是可以被中断的，除非该任务将中断关了，或者 μC/OS-II

将中断关了。被中断了的任务就进入了中断服务态（ISR）。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，从中断服务子程序返回之前，μC/OS-Ⅱ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。如果中断服务子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才能继续运行。

当所有的任务都在等待事件发生或等待延迟时间结束，μC/OS-Ⅱ执行空闲任务（idle task），执行 OSTaskIdle()函数。

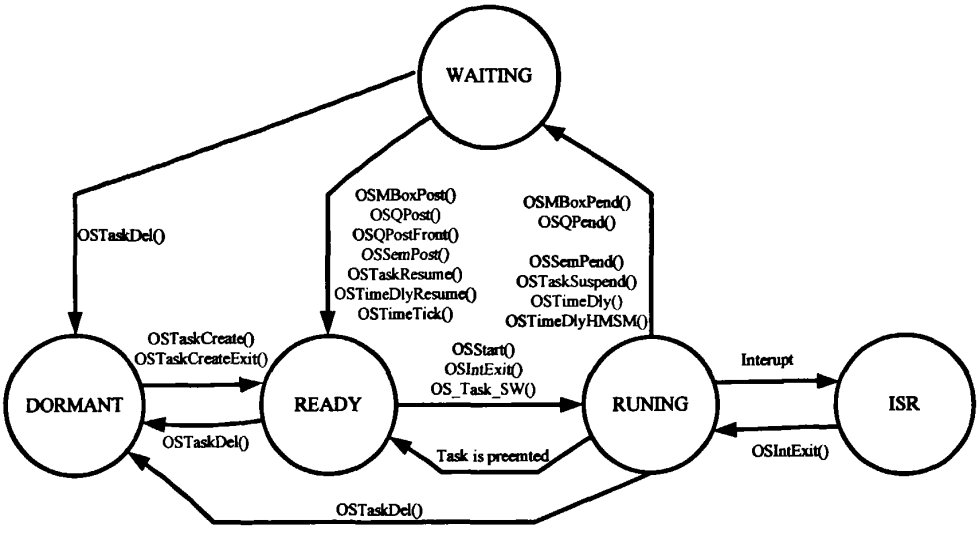


图 2-2 任务的状态

三、μC/OS-II 中的任务的结构

操作系统作为资源管理和分配程序，其本质任务是自动控制程序执行，并满足进程执行过程中提出的资源使用请求。因此，操作系统的核心控制结构是进程结构，资源管理的数据结构将围绕进程结构展开。为了有效地管理进程和资源，操作系统必须掌握每一个进程和资源的当前状态。从效率出发，操作系统的控制结构及其管理方式必须是简明有效的，通常是通过构造一组表来管理和维护进程和每一类资源的信息。

在设计一个较为复杂的应用程序时，通常把一个大任务分解成多个小任务，然后通过运行这些小任务最终达到完成大任务的目的。这种做法也使应用程序容易维护。因此，现代操作系统几乎都是多任务系统。在μC/OS-II 中，与上述小任务对应的程序实体叫做“任务”，实质上是一个线程（也有的文献中称之为进程）。μC/OS-II 就是一个能对这些小任务的运行进行管理和调度的多任务操作

系统。

从应用程序设计的角度看，μC/OS-II 的任务就是一个线程，就是一个用来解决用户问题的 C 语言函数和与之相关联的一些数据结构而构成的一个实体。

从任务的存储结构看，μC/OS-II 的任务由三部分组成：即任务程序代码、任务堆栈和任务控制块。其中，任务控制块用来保存任务属性；任务堆栈用来保存任务的工作环境；任务代码是任务的执行部分。为了便于管理，μC/OS-II 把每一个任务都作为一个节点，然后将它们链接成一个任务链表，如图 2-3[11]所示。

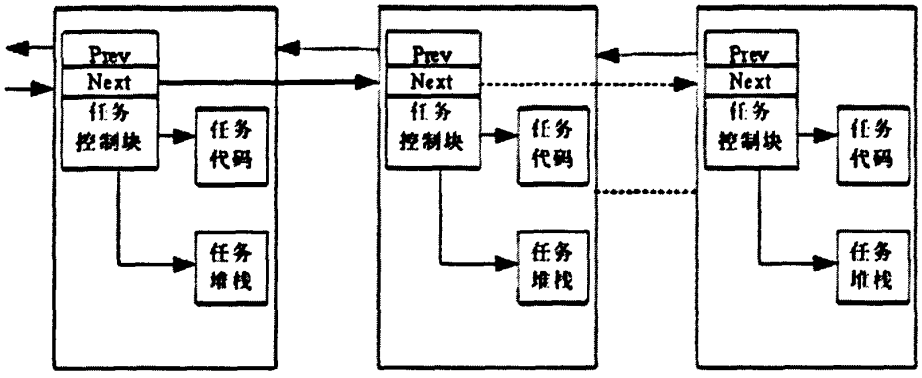


图 2-3 任务在内存表中的结构——任务链表

由于嵌入式系统完成的是对一个装置或设备的控制任务，任务的功能相对固定，因此一般情况下，嵌入式实时操作系统支持的典型任务应该是一个无限循环结构，即函数从来不返回。μC/OS-II 是多任务系统，而任务函数又从来不返回无限循环或者执行一次就删除，因此函数只能是下面的两种形式（程序清单 2.1 和程序清单 2.2）：

程序清单 2.1

```
void YourTask(void*pdata)
{
    任务初始化代码;
    for(;;)
    {
        用户代码;
        /*调用μC/OS-II 的服务函数之一*/
        OSMboxPend();
        OSQPend();
        OSSendPend();
        OSTaskDel(OS_PRIO_SELF);
        OSTaskSuspend(OS_PRIO_SELF);
```

```
            OS TimeDly();
            OS TimeDlyHMSM();
            用户代码;
        }
    }
```

或者

程序清单 2.2

```
void YourTask(void*pdata)
{
    用户代码;
    OSTaskDel(OS_PRIO_SELF);
}
```

从任务的代码看，任务实质上就是一个返回类型为 void 的函数，并在函数的无限循环中完成用户的工作。对于执行多次的任务用第一种形式，对于只执行一次的任务用第二种形式。两者都有一个 μC/OS-II 的系统调用以保证函数不返回和让出 CPU 资源。这里应注意，返回类型必须为 void，以便允许用户应用程序向该任务传递任何类型的参数。

四、就绪表

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，包括 0 和 OS_LOWEST_PRIO 在内。当 μC/OS-II 初始化的时候，最低优先级 OS_LOWEST_PRIO 总是被赋给空闲任务 idle task。注意，最多任务数目 OS_MAX_TASKS 和最低优先级数是没有关系的。用户应用程序可以只有 10 个任务，而仍然可以有 32 个优先级的级别（如果用户将最低优先级数设为 31 的话）。

每个任务的就绪态标志都放入就绪表中的，就绪表中有两个变量 OSRedyGrp 和 OSRdyTbl[]。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 8 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 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 μC/OS-II 对 RAM（数据空间）的需求量。

为确定下次该哪个优先级的任务运行了，内核调度器总是将 OS_LOWEST_PRIO 在就绪表中相应字节的相应位置 1。OSRdyGrp 和 OSRdyTbl[]之间的关系见图 2-4 [10]，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，

当 OSRdyTbl[2]中的任何一位是 1 时，OSRdyGrp 的第 2 位置 1，

当 OSRdyTbl[3]中的任何一位是 1 时，OSRdyGrp 的第 3 位置 1，

当 OSRdyTbl[4]中的任何一位是 1 时，OSRdyGrp 的第 4 位置 1，

当 OSRdyTbl[5]中的任何一位是 1 时，OSRdyGrp 的第 5 位置 1，

当 OSRdyTbl[6]中的任何一位是 1 时，OSRdyGrp 的第 6 位置 1，

当 OSRdyTbl[7]中的任何一位是 1 时，OSRdyGrp 的第 7 位置 1，

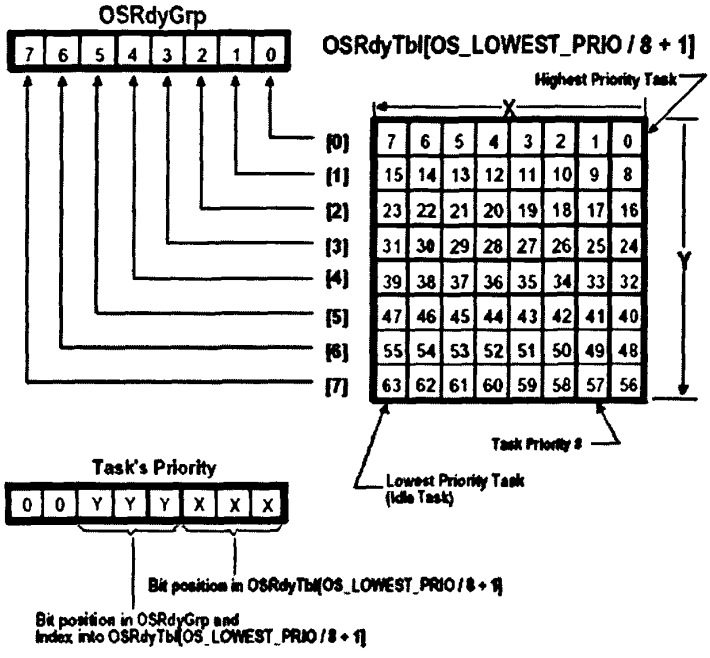


图 2-4 任务就绪表

使一个任务进入就绪状态的操作是：

程序清单 2.3

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

从就绪表中删除一个任务的操作是：

程序清单 2.4

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

其中，Prio 是任务的优先级，OSMapTbl[]的值为

Index	Bit Mask (Binary)
0	00000001
1	00000010

2	00000100
3	00001000
4	00010000
5	00100000
6	01000000

找出进入就绪态的优先级最高的任务：

程序清单 2.5

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

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（见文件 OS_CORE.C）屏蔽字，用于限制 OSRdyTbl[]数组的元素下标在 0 到 7 之间。

五、任务切换的实现

任务级的切换首先从就绪表中找出当前优先级最高的任务的优先级，然后就是 OS_TASK_Sw()要做的事情。首先将 CPU 寄存器中需要保护的内容推入当前任务的任务栈中，并将栈顶指针存入其任务控制块 (TCB)中的 OSTAtkPtr。根据找到的处于就绪态且拥有最高优先级的任务的优先级得到其 TCB，将其 OSTCBStkPtr 赋给处理器的 SP，再将任务栈中保护的内容出栈到 CPU 的寄存器，新的任务开始投入运行[12]。示意代码如下所示。

程序清单 2.6

```
//OSCtxSw()的示意代码
OSCtxSw:
将当前任务所有处理器寄存器的值推入当前任务栈;
Save the stack pointer at OSTCBCur->OSTCBStkPtr;
Call OSTaskSwHook();
OSTCBCur=OSTCBHighRdy;
OSPrioCur=OSPrioHighRdy;
将 OSTCBHighRdy->OSTCBStkPtr 装入处理器的栈指针;
从栈中弹出所有寄存器的值;
```

2.2.2 时钟和中断

一、时钟节拍

μC/OS-Ⅱ(其它内核也一样)要求用户提供定时中断来实现延时与超时控制等功能。这个定时中断叫做时钟节拍，它一般每秒发生 10 至 100 次。但是，随着对实时性要求的提高，频率有上升的趋势：以 Linux 为例，到内核 2.6.14 为止，在大多数体系结构上，时钟滴答频率已变成可配置。基于 386 的 32 位系统现在的默认值是 250Hz——即每秒 250 次滴答——但在建立内核时，给出 100、250 和 1,000Hz 作为选项。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，系统的负荷就越高。

μC/OS-Ⅱ提供了这样一个系统服务：申请该服务的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。实现这个系统服务的函数叫做 OSTimeDly()。调用该函数会使μC/OS-Ⅱ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它的任务通过调用OSTimeDlyResume()取消了延时，它就会马上进入就绪状态。注意，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。

各种实时内核都有将任务延时若干个时钟节拍的功能。然而这并不意味着延时的精度是 1 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作一次裁决，所以如果任务将自身延迟 1 个时钟节拍的时间，则延时并不是精确的 1 个时钟节拍的时间。实际上可能有 3 种情况

图 2-5 到 图 2-7[6]示意任务将自身延迟一个时钟节拍的时序。阴影部分是各部分程序的执行时间。请注意，相应的程序运行时间是长短不一的，这反映了程序中含有循环和条件转移语句(即 if/else, switch, ? : 等语句)的典型情况。时间节拍中断服务子程序的运行时间也是不一样的。

第一种情况如图 2-4[10]所示，优先级高的任务和中断服务超前于要求延时一个时钟节拍的任务运行。可以看出，虽然该任务想要延时 20mS，但由于其优先级的缘故，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。

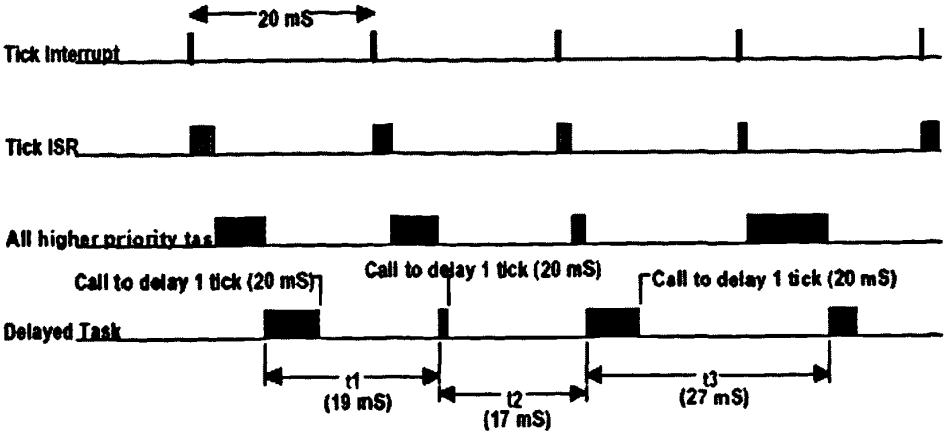


图 2-5 将任务延迟一个时钟节拍(第一种情况)

第二种情况，如图 2-5 所示，所有高优先级的任务和中断服务的执行时间略微小于一个时钟节拍。如果任务将自己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，这个任务的再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的话，则要多定义一个延时时钟节拍。换句话说，如果想要将一个任务至少延迟 5 个时钟节拍的话，得在程序中延时 6 个时钟节拍。

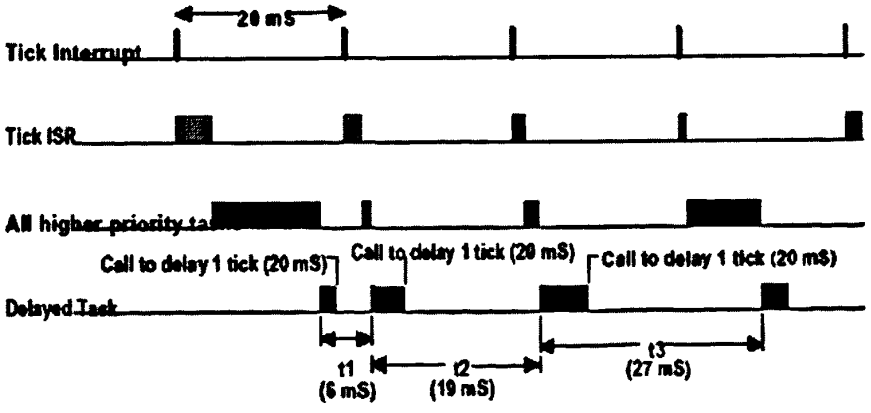


图 2-6 将任务延迟一个时钟节拍(第二种情况)

第三种情况，如图 2.27 所示，所有高优先级的任务加上中断服务的执行时间长于一个时钟节拍。在这种情况下，拟延迟一个时钟节拍的任务实际上在两个时钟节拍后开始运行，引起了延迟时间超差。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。

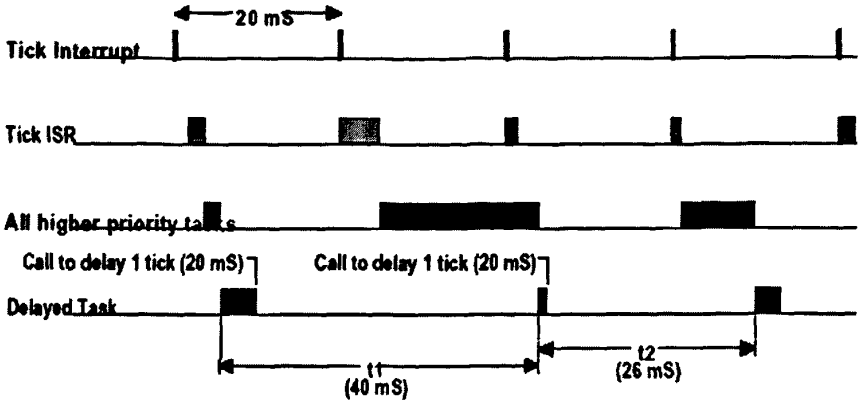


图 2-7 将任务延迟一个时钟节拍(第三种情况)

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：

- 增加微处理器的时钟频率
- 增加时钟节拍的频率
- 重新安排任务的优先级
- 避免使用浮点运算(如果非使用不可，尽量用单精度数)
- 使用能较好地优化程序代码的编译器
- 时间要求苛刻的代码用汇编语言写
- 如果可能，用同一家族的更快的微处理器做系统升级。如从 8086 向 80186 升级，从 68000 向 68020 升级等。

但不管怎么样，抖动总是存在的。

二、中断系统设计

中断是一种硬件机制，用于通知 CPU 有个异步事件发生了。中断一旦被识别，CPU 保存部分(或全部)现场(Context)即部分或全部寄存器的值，跳转到专门的子程序。该了程序称为中断服务子程序(ISR)。中断服务子程序做事件处理，处理完成后，程序回到：

- 在前后台系统中，程序回到后台程序；
- 对不可剥夺型内核而言，程序回到被中断了的任务；
- 对可剥夺型内核而言，让进入就绪态的优先级最高的任务开始运行。

在嵌入式操作系统中，对中断处理十分重视，可以说多数嵌入式操作系统都是事件驱动的[15]。中断使得 CPU 可以在事件发生时才予以处理，而不必让微处理器连续不断地查询(Polling)是否有事件发生。通过两条特殊指令：关中断(Disable interrupt)和开中断(Enable interrupt)可以让微处理器不响应或响应中断。在实时环境中，关中断的时间应尽量的短。关中断影响中断延迟时间，

关中断时间太长可能会引起中断丢失[16]。微处理器一般允许中断嵌套，也就是说在中断服务期间，微处理器可以识别另一个更重要的中断，并服务于那个更重要的中断。如图 2-10 所示[10]。

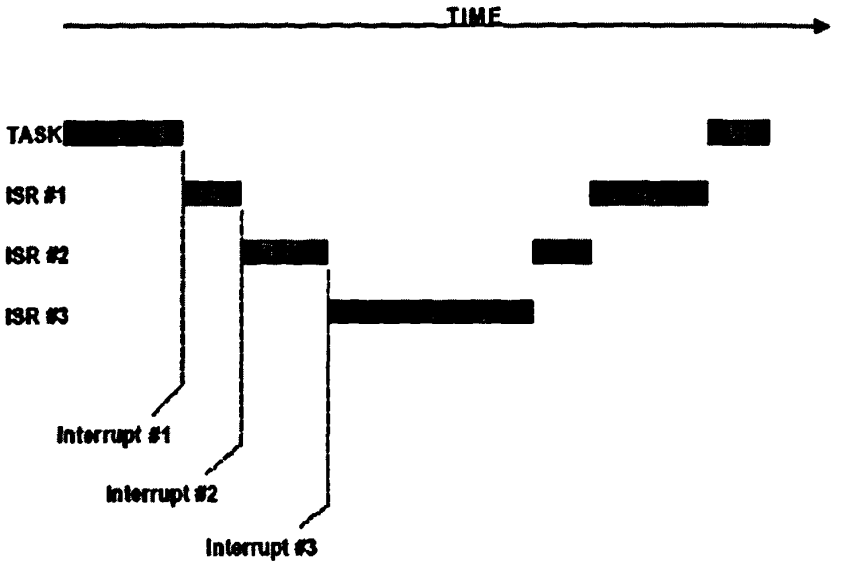


图 2-10 中断嵌套

1、中断延迟

中断延迟(interrupt delay)可能实时内核最重要的指标就是中断关了多长时间。所有实时系统在进入临界区代码段之前都要关中断，执行完临界代码之后再开中断。关中断的时间越长，中断延迟就越长。中断延迟由式[2.1]给出。

中断延迟 = 关中断的最长时间 + 开始执行中断服务子程序的第一条指令的时间。　　[2.1]

2、中断响应

中断响应(interrupt response)定义为从中断发生到开始执行用户的中断服务子程序代码来处理这个中断的时间。中断响应时间包括开始处理这个中断前的全部开销。典型地，执行用户代码之前要保护现场，将 CPU 的各寄存器推入堆栈。这段时间将被记作中断响应时间。

对前后台系统，保存寄存器以后立即执行用户代码，中断响应时间由[2.2]给出。

中断响应时间 = 中断延迟 + 保存 CPU 内部寄存器的时间　　[2.2]

对于不可剥夺型内核，微处理器保存内部寄存器以后，用户的中断服务子程序代码全立即得到执行。不可剥夺型内核的中断响应时间由表达式[2.3]给出。

中断响应时间 = 中断延迟 + 保存 CPU 内部寄存器的时间　　[2.3]

对于可剥夺型内核，则要先调用一个特定的函数，该函数通知内核即将进行

中断服务，使得内核可以跟踪中断的嵌套。对于 μC/OS-Ⅱ说来，这个函数是 OSIntEnter()，可剥夺型内核的中断响应时间由表达式[2.4]给出：

中断响应 = 中断延迟 + 保存 CPU 内部寄存器的时间 + 内核的进入中断服务函数的执行时间　　[2.4]

中断响应是系统在最坏情况下的响应中断的时间，某系统 100 次中有 99 次在 50μs 之内响应中断，只有一次响应中断的时间是 250μs，只能认为中断响应时间是 250μs。

3、中断恢复时间

中断恢复时间(Interrupt Recovery)定义为微处理器返回到被中断了的程序代码所需要的时间。在前后台系统中，中断恢复时间很简单，只包括恢复 CPU 内部寄存器值的时间和执行中断返回指令的时间。中断恢复时间由[2.5]式给出。

中断恢复时间 = 恢复 CPU 内部寄存器值的时间 + 执行中断返回指令的时间　　[2.5]

和前后台系统一样，不可剥夺型内核的中断恢复时间也很简单，只包括恢复 CPU 内部寄存器值的时间和执行中断返回指令的时间，如表达式[2.6]所示。

中断恢复时间 = 恢复 CPU 内部寄存器值的时间 + 执行中断返回指令的时间　　[2.6]

对于可剥夺型内核，中断的恢复要复杂一些。典型地，在中断服务子程序的末尾，要调用一个由实时内核提供的函数。在 μC/OS-Ⅱ中，这个函数叫做 OSIntExit()，这个函数用于辨定中断是否脱离了所有的中断嵌套。如果脱离了嵌套(即已经可以返回到被中断了的任务级时)，内核要辨定，由于中断服务子程序 ISR 的执行，是否使得一个优先级更高的任务进入了就绪态。如果是，则要让这个优先级更高的任务开始运行。在这种情况下，被中断了的任务只有重新成为优先级最高的任务而进入就绪态时才能继续运行。对于可剥夺型内核，中断恢复时间由表达式[2.7]给出。

中断恢复时间 = 判定是否有优先级更高的任务进入了就绪态的时间 + 恢复那个优先级更高任务的 CPU 内部寄存器的时间 + 执行中断返回指令的时间　　[2.7]

4、中断延迟、响应和恢复

注意，对于可剥夺型实时内核，中断返回函数将决定是返回到被中断的任务，还是让那个优先级最高任务运行。是中断服务子程序使那个优先级更高的任务进入了就绪态。在后一种情况下，恢复中断的时间要稍长一些，因为内核要做任务切换。

5、中断处理时间

虽然中断服务的处理时间应该尽可能的短，但是对处理时间并没有绝对的限制。不能说中断服务必须全部小于 100μS，500μS 或 1mS。如果中断服务是在任何给定的时间开始，且中断服务程序代码是应用程序中最重要的代码，则中断服务需要多长时间就应该给它多长时间。然而在大多数情况下，中断服务子程序应识别中断来源，从叫中断的设备取得数据或状态，并通知真正做该事件处理的那个任务。当然应该考虑到是否通知一个任务去做事件处理所花的时间比处理这个事件所花的时间还多。在中断服务中通知一个任务做时间处理(通过信号量、邮箱或消息队列)是需要一定时间的，如果事件处理需花的时间短于给一个任务发通知的时间，就应该考虑在中断服务子程序中做事件处理并在中断服务子程序中开中断，以允许优先级更高的中断打入并优先得到服务。

2.2.3 内存管理

为了便于内存的管理，在μC/OS-II 中使用内存控制块（memory control blocks）的数据结构来跟踪每一个内存分区，系统中的每个内存分区都有它自己的内存控制块。下面是是内存控制块的定义。

程序清单 2.7

```
typedef struct {
    void    *OSMemAddr;
    void    *OSMemFreeList;
    INT32U   OSMemBlkSize;
    INT32U   OSMemNBlks;
    INT32U   OSMemNFree;
} OS_MEM;
```

- OSMemAddr 是指向内存分区起始地址的指针。它在建立内存分区时被初始化，在此之后就不能更改了。
- OSMemFreeList 是指向下一个空闲内存控制块或者下一个空闲的内存块的指针，具体含义要根据该内存分区是否已经建立来决定。
- OSMemBlkSize 是内存分区中内存块的大小，是用户建立该内存分区时指定的。
- OSMemNBlks 是内存分区中总的内存块数量，也是用户建立该内存分区时指定的。
- OSMemNFree 是内存分区中当前可以得空闲内存块数量。

如果要在μC/OS-II 中使用内存管理，需要在 OS_CFG.H 文件中将开关量

OS_MEM_EN 设置为 1。这样 μC/OS-II 在启动时就会对内存管理器进行初始化。

2.2.4 资源管理

在单任务系统中，任务是线性执行，任务不可能被抢占，所以不需要同步来保护共享资源与临界资源，同时单任务也不存在数据交换的问题，但对于多任务操作系统，会出现与单任务系统不同的问题，进程间通信与同步就是为了解决这些问题而提出的特有机制，它们为多任务系统提供了不同进程的通信机制，同时也提供了对于临界资源和共享资源的保护。

进程间通信与同步是多任务系统中的不同表现形式，，对于嵌入式操作系统，进程间通信与同步处于同一地址空间，进程间通信机制同时可以用作同步机制。任务间信息的传递有两个途径：通过全程变量或发消息给另一个任务。用全程变量时，必须保证每个任务或中断服务程序独享该变量。中断服务中保证独享的唯一办法是关中断。如果两个任务共享某变量，各任务实现独享该变量的办法可以是关中断再开中断，或使用信号量。任务只能通过全程变量与中断服务程序通讯，而任务并不知道什么时候全程变量被中断服务程序修改了，除非中断程序以信号量方式向任务发信号或者是该任务以查询方式不断周期性地查询变量的值。要避免这种情况，可以考虑使用邮箱或消息队列。

同步可以利用信号量实现，如图 2-7 所示，用来实现同步机制的信号量初始化为 0，信号量用于这种类型的同步称作单向同步(unilateral rendezvous)。一个任务作 I/O 操作，然后等信号回应。当 I/O 操作完成，中断服务程序（或任务）发出信号，该任务得到信号后继续往下执行。

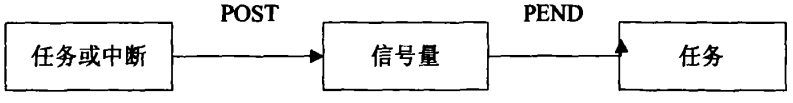


图 2-8 用信号量使任务与中断同步

如果内核支持计数式信号量，信号量的值表示尚未得到处理的事件数，可能会有一个以上的任务在等待同一事件的发生，则这种情况下，内核会根据以下原则之一发信号给相应的任务：

- 发信号给等等待事件发生的任务中优先级最高的任务
- 发信号给最先开始等待事件发生的那个任务

根据不同的应用，发信号以标识事件发生的中断服务或任务也可以是多个，两个任务可以用两个信号量同步它们的行为，如图 2-8 所示，这叫做双向同步(bilateral rendezvous)。

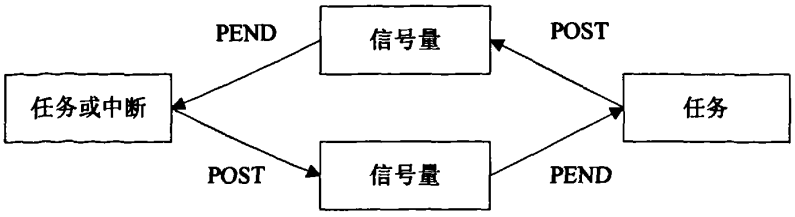


图 2-9 两个任务用两个信号量同步

在μC/OS-II 中，有多种方法可以保护任务之间的共享数据和提供任务之间的通讯。主要有关闭中断和打开中断、上锁和开锁、信号量等。

一、关闭和打开中断

当两个任务或者一个任务和一个中断服务子程序共享数据时，可以利用宏 OS_ENTER_CRITICAL()和 OS_EXIT_CRITICAL()来关闭和打开中断，以实现任务之间的通讯和同步。在进入临界段时关中断，处理完毕后再开中断。这使得μC/OS-II 能够避免同时有其它任务或中断服务进入临界段。关中断的时间是评价系统实时性的一个重要指标，关中断的时间越短，系统实时性越好。

二、上锁和开锁

利用函数 OSSchedLock()和 OSSchedUnLock()任务调度函数上锁和开锁，也可以实现数据的共享。给调度器上锁函数 OSSchedLock()用于禁止任务调度，直到任务完成后调用开锁函数 OSSchedUnLock()。调用 OSSchedLock()后，用户的应用程序不得使用任何能将现行任务挂起的系统调用。

第三章 μC/OS-II 在 C51 系统上的移植

为了实现智能验钞机系统应用程序，首先要在系统上移植一个操作系统。所谓移植，就是使一个实时内核能在某个微处理器或微控制器上运行[17]。μC/OS-II 支持几乎所有的 8、16 位、32 位微处理器，但并非所有机构的微处理器都能进行μC/OS-II 的移植，要使μC/OS-II 正常运行，处理器必须满足以下要求[10]：

1. 处理器的 C 编译器能产生可重入代码。
2. 用 C 语言就可以打开和关闭中断。
3. 处理器支持中断，并且能产生定时中断。
4. 处理器支持能够容纳一定量数据(可能是几千字节)的硬件堆栈。
5. 处理器有将堆栈指针和其它 CPU 寄存器读出和存储到堆栈或内存中的指令。

为了方便移植，大部分的μC/OS-II 代码是用 C 语言写的；但依赖于具体硬件结构的操作必须要用汇编语言来完成，如寄存器存放，硬件堆栈操作等。操作系统的移植非常依赖于微处理器具体的机构和所用的编译器，本文以 C51 结构微处理器为例，编译器使用 Keil μVersion2,对μC/OS-II 进行移植。

3.1 Keil C 简介

如前所述，大部分的μC/OS-II 代码是用 C 语言写的。因此，移植 μC/OS-II 需要一个 C 编译器，本项目选择 Keil C51 作为开发平台。

Keil C51 是美国 Keil Software 公司出品的基于 80C51 内核的微处理器软件开发平台，内嵌多种符合当前工业标准的开发工具，可以完成从工程的建立、管理、软件编译、链接，到目标代码的生成、软件仿真和硬件仿真等完整的开发流程。尤其 C 编译工具在产生代码的准确性和效率方面达到了较高的水平，而且可以附加灵活的控制选项，在开发大型项目时非常理想。Keil C51 集成开发环境的主要功能有以下几点：

- μVersion2 for Windows™ ：是一个集成开发环境，将项目管理、源代码管理和程序高度等组合在一个功能强大的环境中；
- C51 国标标准优化 C 交叉编译器：从 C 源代码产生可重定位的目标模块
- A51 宏汇编器：从 80C51 汇编源代码产生可重定位的目标模块；
- BL51 链接器/定位器：组合由 A51 和 C51 产生的可重定位的目标模块，生成绝对目标模块；
- LIB51 库管理器：从目标模块生成链接器可以使用的库文件；
- OH51 目标文件到 HEX 格式的转换器：从绝对目标模块生成 Intel HEX 文

件；

● RTX-51 实时操作系统：简化了复杂的衬里应用软件项目的设计。

C51 V7 版本是目前最高效、灵活的 8051 开发平台。它可以支持所有 8051 的衍生产品，也可以支持所有兼容的仿真器，同时支持其它第三方开发工具。因此，本次使用 V7 作为开发平台。

原则上我们不用修改与处理器无关的代码，但是由于 Keil C 编译器的特殊性，这些代码仍要多处改动。因为 Keil C 在缺省情况下，编译的代码不可重入，而多任务系统要求并发操作导致重入，所以要在每个 C 函数及其声明后标注 reentrant 关键字。

3.2 MCU 运行环境

C51 单片机采用的是冯.诺伊曼提出的经典计算机体系结构框架，即一台算机是由运算器、控制器、存储器、输入设备和输出设备共五个基本部分组成[18]。C51 单片机在一块芯片上集成了 CPU、RAM、ROM、定时器/计数器和多功能 I/O 口等[19] 。本项目所使用验钞机的的 MCU 采用的是 Atmel 的 T89C51AC2，外部晶振频率为 20MHz，共 32KB 的代码空间。下面描述本系统所采用单片机 T89C51AC2 的运行环境。

T89C51AC2 单片机有 4 个存储器空间，分别用来安排 4 种不同功用的存储器：

1. 内部数据存储器；
2. 特殊功能寄存器；
3. 程序存储器；
4. 外部数据存储器。

内部数据存储器和特殊功能寄存器集成于片内，程序存储器和外部数据存储器则安排在片外。内部 RAM 共 128 字节，地址范围为 00H-7FH[14]。单片机的前 32 个单元（地址 00H-FFH）称为寄存器区。其中，每 8 个寄存器形成一个寄存器组[20]。通过对特殊功能寄存器 PSW 中 RS1、RS0 两位的编程设置，可选择任一寄存器组为工作寄存器组。从原理上讲，任务的切换应该包含所有的寄存器，总共 32 个寄存器，这样对于指令周期只有 1M 的单片机是个不小的开销，而且每个任务栈必须包含这 32 个字节的开销，因此资源开销太大。所以，现在目前的多任务系统仅对寄存器 0 组进行保护，当然要这有个前提条件，就是系统自始至终都必须使用第 0 组寄存器，别的寄存器只能用作一般内部 RAM。

单片机的特殊功能寄存器位于单片机内部 RAM 的 0x80-0xFF 区间[21]，其中和程序运行环境相关的寄存器有 PCL，PCH，PSW，ACC，B，DPL，DPH。这些寄存器在任务切换时需要全部保存在任务堆栈，在任务重新调度时按照原来次序依次从

堆栈恢复到对应寄存器。寄存器的任务分配如表3.1所示。

表3-1 寄存器功能分配表

寄存器	说明
PCH, PCL	程序计数器
PSW	程序状态字，它表示程序的相关状态
ACC, B	累加器，是ALU单元能直接取用的两个寄存器
DPL, DPH	数据指针计数器，表示当前数据指针的地址

3.3 移植的实现

μC/OS-II代码中90%以上与硬件无关的代码用C语言写成，但涉及到数据类型的重定义、堆栈结构的设计、任务切换时状态的保存和恢复等问题的大部分代码由于与处理器有关，是用汇编语言实现的，代码不足200行。移植所要做的工作，就是在不同的处理器上用汇编语言来改写与处理器有关的代码，以及其他与处理器特性相关的部分。

对 μC/OS-II进行移植到不同处理器平台时，需要解决的主要问题有：

(1)数据类型的重定义。对于一个操作系统来说，基于其上进行开发的应用系统一般都使用高级语言，高级语言都有自己的数据类型。但对于不同的处理器由于字长不相同，造成同一数据类型在不同处理器中会有不同的解释，所以对不同的处理器应该重新进行数据类型的定义。

(2)堆栈结构的设计。当同一个操作系统应用于不同处理器或同一处理器的不同应用系统时，由于各应用系统所追求的性能各有特点，就会要求与性能有很大关系的堆栈结构尽可能与本系统所追求的性能一致。

(3)任务切换时的状态保存与恢复。这是多任务操作系统最主要的工作，也是最频繁的工作。所以任务切换在实现时的正确与否是操作系统运行时的基本保证，同时它的简洁与否决定操作系统的效率。

根据上一章的阐述，在μC/OS-II移植过程中涉及以上问题的代码都包含在文件OS_CPU.H、OS_CPU_C.C、OS_CPU_A.ASM中。因此移植的主要工作也在源代码的基础上围绕着这三个文件的改写展开。下面就详细阐述这三个文件的修改。

3.3.1 OS_CPU.H文件的修改

OS_CPU.H文件定义了与CUP相关的基本信息，包括了用#define语句定义的、与处理器相关的常数、宏以及类型。

1. 定义编译器相关的数据类型

因为不同的微处理器有不同的字长，所以μC/OS-II 的移植包括了一系列的数据类型定义，以确保其可移植性。尤其是μC/OS-II 代码从不使用 C 语言中的 short,int 及 long 等数据类型，因为它们是与编译器相关的，是不可移植的。相反，定义的整型数据结构等既是可移植的，又很直观。这里数据类型的 BOOLEAN 要定义为 unsigned char 类，而不能是 bit 类型，因为 bit 类型不能用在数组和结构体中[11]。

2. 定义进入与退出临界区宏

与所有的实时内核一样，μC/OS-II 需要先禁止中断再访问代码的临界段，并且在访问完毕后重新允许中断。这就使得μC/OS-II 能够保护临界段代码免受多任务或中断服务例程(ISR)的破坏。中断禁止时间是商业实时内核公司提供的重要指标之一，因为它将影响到用户的系统对实时事件的响应能力。关中断函数 OS_ENTER_CRITICAL()和开中断函数 OS_EXIT_CRITICAL()总是成对出现的，如程序清单 3.1 所示。最简单的方法是分别设置为 EA=0 和 EA=1，即在 OS_ENTER_CRITICAL()中调用处理器指令来禁止中断，以及在 OS_EXIT_CRITICAL()中调用允许中断指令，不少移植实例都是这么处理的。这样做存在着小小的问题，如果用户在禁止中断的情况下调用了μC/OS-II 函数，在从μC/OS-II 返回的时候，中断可能会变成是允许的了。为避免这种现象的发生，我们引入关中断计数器 Os_Enter_Sum，保证了调用这类μC/OS-II 函数前后，中断的开关状态不会改变[24]。

程序清单 3.1

```
//μC/OS-II 函数访问临界资源的一般形式
{
    OS_ENTER_CRITICAL();
    /* μC/OS-II 临界代码段 */
    OS_EXIT_CRITICAL();
}
```

3. 设置堆栈增长方向 OS_STK_GROWTH

绝大多数微处理器和微控制器的堆栈是从上往下递减的，但是也有某些处理器使用的是相反的方式，μC/OS-II 被设计成对 2 种情况都可以处理，只要再用配置常数 OS_STK_GROWTH 指定堆栈的方向就可以了：

置 OS_STK_GROWTH 为 0 表示堆栈从下（低地址）往上（高地址）递增；

置 OS_STK_GROWTH 为 1 表示堆栈从上（高地址）往下（低地址）递减。

OS_STK_GROWTH 设置为 0，因为 C51 堆栈是从下往上增长（1 是向下增长，0 是向上增长）。

4. 定义任务切换函数 OS_TASK_SW()为 OSCtxSw()

OS_TASK_SW()是一个宏，它是在μC/OS-II从低优先级任务切换到最高优先级任务时被调用的，OS_TASK_SW()总是在任务级代码中被调用的。因为 C51 没有软中断指令，所以用程序调用代替。子程序和中断两者的堆栈格式相同，RETI 指令复位中断系统，RET 则没有。实践表明对于 C51 用子程序调用入栈用中断返回指令 RETI 出栈是没有问题的，反之中断入栈 RET 出栈则不行。总之对于入栈，程序调用与中断调用效果是一样的，可以混用。在没有中断发生的情况下复位中断系统也不会影响系统正常运行。

程序清单 3.2 比较了一下μC/OS-II移植到了 C51 上的代码跟针对 Intel 80 x86 处理器的相应代码。

程序清单 3.2

```
#define  OS_ENTER_CRITICAL()  EA = 0,Os_Enter_Sum++            /* 禁止中断*/
#define  OS_EXIT_CRITICAL()   if (--Os_Enter_Sum==0) EA = 1    /* 允许中断*/
#define  OS_STK_GROWTH        0
#define  OS_TASK_SW()         OSCtxSw()    /* 任务切换函数*/

//比较一下，μC/OS-II在 Intel 80x86 上的相应代码为：

#define  OS_ENTER_CRITICAL()  asm {PUSHF; CLI}
#define  OS_EXIT_CRITICAL()   asm  POPF
#define  OS_STK_GROWTH        1
#define  uCOS                 0x80       // 中断向量号
#define  OS_TASK_SW()         asm  INT   uCOS
```

3.3.2 OS_CPU_C.C 文件修改

此文件完成任务的初始化，是用 C 语言编写的。μC/OS-II是采用软件中断来进行任务切换的，所以要初始化任务的栈结构，使任务的堆栈结构和中断的堆栈结构一样，所有的寄存器都保留在堆栈中。

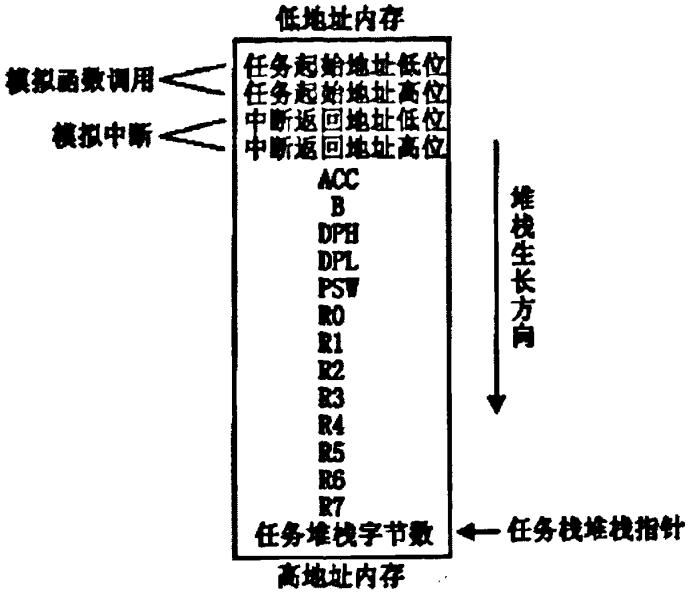
μC/OS-II的移植实例要求用户编写六个简单的 C 函数：

- 初始化任务堆栈函数 OSTaskStkInit()
- 任务创建钩挂函数 OSTaskCreateHook()
- 任务删除钩挂函数 OSTaskDelHook()
- 任务切换钩挂函数 OSTaskSwHook()
- 统计任务钩挂函数 OSTaskStatHook()

● 定时钩挂函数 OSTimeTickHook()

唯一必要的函数是 OSTaskStkInit()，其它五个函数必须得声明但没必要包含代码，它们是由系统函数调用的钩子函数。OSTaskStkInit()函数由任务创建函数 OSTaskCreate()调用，功能是初始化任务堆栈。由于各种处理器的寄存器对堆栈的操作方式不尽相同，因此该函数需要用户在进行μC/OS-II 移植时，按所使用的处理器由用户来编写。为了编写该函数，必须理解 T89C51AC2 的堆栈，及怎样设置软件堆栈。堆的大小可根据任务的实际情况自行确定。5 个系统钩子函数可直接定义它们为空函数。

所设计的任务堆栈的初始结构为：



堆栈结构说明：

(1) Keil C51 编译器支持在 C 语言中写中断处理程序，当进入由 C 语言写的中断处理程序时，在 CPU 将中断返回地址压入堆栈后，紧接着，Keil C51 编译器自动将 ACC、B、DPH、DPL、PSW、R0、R1、R2 、R3 、R4 、R5 、R6、R7 按照先后顺序压入堆栈。在退出中断处理程序之前，应调用 OSIntExit()以完成可能会发生的任务切换；而此时若发生任务切换，则在 OSIntExit()中将调用汇编函数 OSIntCtxSw()，在这里，将直接返回到程序断点处，从而屏蔽掉由 Keil C51 编译器生成的退出中断前的一系列出栈指令。由于所设计的堆栈结构与 Keil C51 编译器压栈的顺序是一致的，因此，仍可以在 C 语言中写中断处理程序。

(2)“任务堆栈字节数”保存了从“任务起始地址低位”一直到“R7”的任务堆栈中的字节数目。由于 51 系列单片机内部 RAM 小（仅 128-8 个字节），通过中断切换任务要保存的内容多：寄存器（R0～R7，A，B，PSW，DPTR 和 PC），全

局变量和局部变量。如果不把所有空闲 RAM 分配给当前任务做堆栈，则堆栈肯定会溢出，故只好以空间换时间。定义了一个数组保存所有任务的堆栈的项端和底端，在任务切换时，把不是当前任务的堆栈空间上移或下移，直至所有空间都提供给当前任务使用。

3.3.3 OS_CPU_A.ASM 文件修改

移植工作的难点是在 OS_CPU_A.ASM 文件上，这里用户需要编写 4 个汇编语言函数：OSStartHighRdy()、OSCtxSw()、OSIntCtxSw()和 OSTickISR()。其中，OSTickISR()用 C 语言写，即用定时器中断处理了程序来代替它。因此，只需要写 3 个汇编语言函数。

对这几个函数的关键代码段作详尽的分析。

```
NAME OS_CPU_A. ASM
?STACK SEGMENT IDATA
        RSEG ?STACK
OSStack:
        DS 60H
OSStkStart  IDATA  OSStack
```

在程序的首部，定义了所用到的一些外部变量以及系统堆栈区。当程序进入 main()函数时，堆栈指针 SP 指向(#OSStkStart- 1)。

另外，由于硬件堆栈区被定义为 96byte(60H)，因此，每个任务的任务栈也不能超过此限制。

- 调用的运行优先级最高的就绪任务的 OSStartHighRdy()函数

该函数是在操作系统初始化并建立了至少一个任务之后被调用的，它首先找到当前就绪的最高优先级任务，并从该任务控制块 OS_TCB 中取出堆栈指针，然后从堆栈中弹出全部寄存器（包括 SP 指令指针寄存器），并 RET 返回。

首先 OSStartHighRdy()必须调用 OSTaskSwHook()函数，将 OSRunning 标志设为“真”。接下来，OSStartHighRdy()需要得到指向最高优先级任务的任务堆栈的指针，在程序中，该堆栈指针是被存放到 DPTR 中的。当 DPTR 指向 OSTCBHighRd->OSTCBStkPt 以后，就可以恢复具有最高优先级的任务运行环境了。由于 51 系列单片机的堆栈区最大仅为 128 byte，而所定义的可用的硬件堆栈区仅 96 byte，空间十分有限。为了在这样一个硬件环境中进行任务切换，将外部 RAM 保存的最高优先级任务的任务堆栈中的运行环境先转移到硬件堆栈区。由于在每个任务栈的最顶端存放着该任务栈中的有效字节数，为此，首先应得到这个需要从任务堆栈转移到系统硬件堆栈中去的字节数，然后调整 DPTR 值，使其指向实际需要转移的堆栈字节的最高地址处，之后就可以进行实际的堆栈内容转移。当任务栈中的内容全都转移到硬件堆栈中后，调整当前任务栈的指针，使当前任务栈看起来就像被清空了一样。之后，由于在 μC/OS-II 中，处于就绪状态的任务的堆栈结构看起来就像刚发生过中断一样，所有的寄存器都被保存在堆

栈中。要想运行最高优先级任务，需要将所有处理器寄存器按顺序从任务堆栈中恢复出来，并且执行中断返回指令。这样，CPU 就将开始恢复执行最高优先级任务的第 1 条指令。

程序清单 3.4

```
?PR?OSStartHighRdy SEGMENT CODE
RSEG PR? OSStartHighRdy
CLR EA
LCALL_?OSTaskSwHook
MOV DPTR,#OSRunning
MOV A,#O1H
MOVX@DPTR, A
；接下来，需要得到指向 OSTCBHighRdy->OSTCBStkPtr 的指针。
MOV DPTR,#OST CBHighRdy
INC DPTR
MOVX A，@DPTR
MOV R0,A
INC DPTR
MOVX A，@DPTR
MOV R1，A
MOV DPH，R0
MOV DPL，R1
INC DPTR
MOVX A，@DPTR
MOV R0，A
INC DPTR
MOVX A，@DPTR
MOV R1，A
MOV DPH，R0
MOV DPL，R1
MOVX A，@DPTR
MOV R4，A
MOV R2，DPH
MOV R3，DPL
DEC R2
```

```
    CJNE R3，#OFFH，next1
    DEC R2
    Nextl：
    MOV DPH，R2
    MOV DPL，R3；初始化时，SP 指向硬件堆栈区中的固定地址（#OSStkStart
－1）＋#2 处
    MOV A，SP
    ADD A，R4
    MOV SP，A；现在 SP 指向硬件堆栈区的栈顶
    MOV R0，A；接下来就可以进行实际的堆栈内容的转移了
    Transfer1：
    MOVX A，@DPTR
    MOV @R0，A
    DCE R0
    MOV R2，DPH
    MOV R3，DPL
    DEC R3
    CJNE R3，#OFFH，next2
    DEC R2
    Next2：
    MOV DPH，R2
    MOV DPL，R3
    DJNZ R4，transfer1；当任务栈中的内容全部转移到硬件堆栈中后，调整当
前任务栈的指针，使当前任务栈看起来就像被清空了一样
    MOV DPTR，#OST CBHighRdy
    INC DPTR
    MOVX A，@DPT R
    MOV R0，A
    INC DPTR
    MOVX A，@DPT R
    MOV R1，  A
    MOV DPH，R0
    MOV DPL，R1
    INC DPTR
```

```
MOV A，R2
MOVX @DPTR， A
INC DPTR
MOV A，R3
MOVX@DPTR，A；将处理器寄存器从硬件堆栈中恢复过来
POP 7
POP 6
POP 5
POP 4
POP 3
POP 2
POP 1
POP 0
POP PSW
POP DPL
POP DPH
POP B
POP ACC
SETB EA
RFTI
```

- 任务切换函数 OSCtxSw()

该函数是由于执行进入任务切换宏 OS_TASK_SW 而进入的，它是一个任务级的切换函数，它的主要任务是保存当前任务的 CPU 现场并恢复最高优先级任务的 CPU 现场。

首先，OSCtxSw()按照同 OSTaskStkInit()函数一样的顺序将处理器的寄存器保存到堆栈中，接着，OSCtxSw()需要得到指向当前任务的任务堆栈的指针，在程序中，该堆栈指针是被存放到 DPTR 中的。当 DPTR 指向 OST CBCur->OST CBStkPtr 以后，把此时硬件堆栈中保存的当前任务的运行环境先转移到当前任务的位于外部 RAM 中的任务堆栈中，以便以后若当前正被切换出去的任务重新又被切换进来时正确地恢复运行环境。

接下来，根据当前堆栈指针 SP 的值首先计算出在当前硬件堆栈中有多少字节需要转移到相应的任务堆栈中去，之后进行实际的堆栈内容转移。当任务栈中的内容全都转移到硬件堆栈中后，将转移字节数也存到任务栈中，以便以后使用。

之后，调整 DPTR 的值，使其始终指向任务栈的最高地址处。同时调整堆栈指针 SP 的值，使其看起来就象是硬件堆栈中的所有内容都被弹出了一样。

然后，根据当前 DPTR 的值，来调整当前任务栈的指针，使当前任务栈看起来就像被压满了一样。之后，OSCtxSw()应调用 OSTaskSwHook()函数，并目需要将指向当前任务的指针指向要恢复运行的任务，将新任务的优先级复制给当前任务的优先级，也就是说，新的任务取代了当前任务。

完成了以上这些工作后，将需要切换进来的任务的任务栈的内容转移到硬件堆栈中来。当任务栈中的内容全都转移到硬件堆栈中后，调整当前任务栈的指针，使当前任务栈看起来就像被清空了一样。

之后，由于 μC/OS-II 中，处于就绪状态的任务的堆栈结构看起来就像刚发生过中断一样，所有的寄存器都被保存在堆栈中，要想运行被切换进来的任务，需要将所有处理器寄存器按顺序从任务堆栈中恢复出来，并且执行中断返回指令。这样，CPU 就将开始恢复执行最高优先级任务的第一条指令。

程序清单 3.5

```
?PR?OSCtxSw SEGMENT CODE
RSEG PR?OSCtxSw
_?OSCtxSw:
PUSH ACC
PUSH B
PUSH DPH
PUSH DPL
PUSH PSW
PUSH 0
PUSH 1
PUSH 2
PUSH 3
PUSH 4
PUSH 5
PUSH 6
PUSH 7; 接下来，得到指向 OSTCBCur- > OSTCBStkPtr 的指针
IntCtxSw I:
    MOV DPTR，#OSTCBCur
    INC DPTR
    MOVX A，@DPT R
```

```
        MOV R0, A
        INC DPTR
    MOVX A，@DPT R
    MOV R1，A
    MOV DPH，R0
    MOV DPL，R1
    INC DPTR
    MOVX A，@DPT R
    MOV R0，A
    INC DPTR
    MOVX A，@DPT R
    MOV R1，A
    MOV DPH，R0
    MOV DPL，R1；接着，根据当前堆栈指针 SP 的值首先计算出在当前硬件
堆栈中有多少字节需要转移到相应的任务堆栈中去
    MOV A，#OSStkStart－1
    ADD A，#2
    MOV R0，A
    MOV A,，SP
    CLR C
    SUBB A，R0
    MOV R4，A；R4 中保存着这个需要转移的字节数
    MOV R5，A
    MOV SP，R0；调整 SP
    ；接下来，就可进行实际的堆栈内容转移了
    transfer2:
    INC R0
    INC DPTR
    MOV A，@R0
    MOVX @ DPTR，A
    DJNZ R4，transfer2
    INC DPTR
    MOV A, R5；R5 中保存着这个需要转移的字节数
    MOVX @DPTR，A；将此字节数存入任务栈
```

```
；接下来，调整当前任务栈的指针，使其看起来就像被压满了一样。
MOV R2，DPH
MOV R3，DPL
MOV DPTR，#OSTCBCur
INC DPTR
MOVX A，@DPTR
MOV R0，A
INC DPTR
MOVX A，@DPT R
MOV R1，A
MOV DPH，R0
MOV DPL，R1
INC DPTR
MOV A，R2
MOVX@DPT R， A
INC DPTR
MOV A，R3
MOVX@ DPTR，A
LCALL_? OSTaskSwHook；接着，使 OSTCBCur 等于 OSTCBHighRdy
MOV DPTR，#OSPrioHighRdy
MOVX A,@DPTR
MOV DPTR，#OSPrioCur
MOVX @ DPTR，A；得到指向此任务栈的指针
MOV DPTR，#OSTCBHighRdy
INC DPTR
MOVX A,@DPTR
MOV R0, A
INC DPTR
MOVX A，@DPTR
MOV R1, A
MOV DPH，R0
MOV DPL，R1
INC DPTR
MOVX A，@DPTR
```

```
    MOV R0，A
    INC DPTR
    MOVX A，@DPTR
    MOV R1，A
    MOV DPH，R0
    MOV DPL，R1；得到需要从任务堆栈转移到硬件堆栈中去的字节数
    MOVX A，@DPTR
    MOV R4，A；调整 DPTR，使其指向实际需要转移的堆栈字节的最高地址
数
    MOV R2，DPH
    MOV R3，DPL
    DEC R3；这时 SP 指向硬件堆栈区中的固定地址“（#OSStkStart－1）＋#2”
处
    MOV A，SP
    ADD A，R4
    MOV SP，A；现在 SP 指向硬件堆栈区的栈顶，就好象任务栈中的内容已经
全部转移到了硬件堆栈中了一样
    MOV R0，SP；
    Transfer3：
    MOVX A，@DPTR
    MOV @R0,A
    DEC R0
    MOV R2，DPH
    MOV R3，DPL
    DEC R3
    CJNE R3，#OFFH，next4
    DEC R2
    Next4：
    MOV DPH，R2
    MOV DPL，R3
    DJNZ R4，transfer3；当任务栈中的内容全都转移到堆栈中后，调整当前任
务栈的指针，使当前任务栈看起来就像被清空了一样
    MOV DPTR，#OSTCBHighRdy
    INC DPTR
```

```
MOVX A，@DPTR
MOV R0，A
INC DPTR
MOVX A，@DPTR
MOV R1，A；R0:R1=OSTCBHighRdy
MOV DPH，R0；DPTR 指向 OSTCBHighRdy
MOV DPL，R1
INC DPTR
MOV A，R2
MOVX @DPTR，A
INC DPTR
MOV A，R3
MOVX @DPTR，A；将处理器寄存器从硬件堆栈中恢复出来
POP 7
POP 6
POP 5
POP 4
POP 3
POP 2
POP 1
POP 0
POP PSW
POP DPL
POP DPH
POP B
POP ACC
SETB EA
RETI
```

- 中断任务切换函数 OSIntCtxSw()

  该函数的工作是在中断处理程序退出时进行任务切换。

  OSIntCtxSw()的代码大部分都与 OSCtxSw()相同，仅仅在以下两点有所区别：

  由于中断已经发生，此处不需要再保存寄存器；

  OSIntCtxSw()需要调整堆栈指针，去掉堆栈中一些不需要的内容，以使堆栈

中止包含任务的运行环境。

程序清单 3.6

```
PR?_OSIntCtxSw SEGMENT CODE
RSEG PR?_OSIntCtxSw
_?_OSIntCtxSw():
DEC SP；调整堆栈指针
DEC SP
DEC SP
DEC SP
LJMP IntCtxSwI
```

- 时钟节拍服务函数 OSTickISR()

该函数是时钟中断处理程序，它的任务是在定时发生的时钟中断到来时调用OSTimeTick函数，处理延时结束的任务，然后调用OSIntExit()函数根据优先级进行任务切换，这样就保证了多任务环境的实时性。如果需要进行任务切换，则OSIntExit恢复新任务CPU现场并用IRET返回新任务，否则OSIntExit()将返回调用者OSTickISR()，最后OSTickISR()返回被中断的任务。

μC/OS-II要求用户提供一个周期性的时钟源，来实现时间的延迟或超时功能，时钟节拍应该每秒发生10~100次。为了完成任务，可以使用硬件定时期，这里，用定时期T0来实现。

程序清单 3.7

```
Void T01_vilsrTmr0(void) interrupt T0INT
 {
OSIntEnter();
OSTimeTick();
OSIntExit();
}
```

至此，所要做的移植工作便告一段落，要让系统能够正常工作，还需要进行优化、编写驱动程序、进行系统配置、裁减、创建任务。

3.4 可重入问题的分析与解决

C51用户手册有以下说明[13]：许多标准库函数具有可重入性，但用户手册中特别说明的程序外，标准库函数都没有重入性。其中包括部分数学运算函数、大部分的字符串处理函数、流处理函数以及全部的内存分配函数。这要求使用者在多任务开发中必须明确其使用的函数是否为可重入的，增加了使用者的工作量。

若多任务中使用了非重入函数，则必须将该函数改写为重入函数[14]

可重入函数可以被一个以上的任务调用，而不必担心数据被破坏。可重入函数任何时候都可以被中断，一段时间后又可以继续运行，而相应的数据不会丢失。由于μC/OS-II 是抢占式的实时多任务内核，同一个函数可能会被不同的任务调用，也可能会被中断，因此，移植μC/OS-II 要求 C 语言编译器可以产生可重入函数。但是正常情况下 Keil C51 编译器中的函数不能重入。原因是由于 8051 系列微控制器的硬件堆栈很小，硬件堆栈指针 SP 最多只能在内部 256 字节的 RAM 内移动，不能够指向 64K 的外部 RAM 空间。Keil C51 编译器不是把局部变量分配到堆栈中，而是把局部分量分配到内存固定地址（当然首先尽可能使用寄存器）。这样，局部变量实际上就是全局变量了。即使在函数一开始就开始关闭中断，也不能保证其重入性，因为在关中断函数中已经执行一些指令，把参数保存到所分配的内存固定地址了，这些指令使得函数不可重入。当编译器把所有局部变量分配到寄存器时，函数是可重入的。但当一个函数调用一个外部函数时，编译器认为所有寄存器变量已经发生变化；当一个变量要存储信息，直到调用这个函数后再使用时，这个变量就不可能分配到寄存器中，这个函数也就不可重入了。为了在 Keil C51 中实现可重入函数，可以使用“reentrant”关键字声明该函数是可重入的。编译器可根据编译模式为可重入函数在内部 RAM 或外部 RAM 空间开辟一个模拟堆栈来存储可重入函数的参数和局部变量。可重入函数的返回地址仍然保存在硬件堆栈中。Cx51 编译手册不推荐使用模拟堆栈，原因是受 8051 寻址方式的限制，模拟堆栈访问的效率很低。

为提高效率，我们引入了一种新的方法：如果我们在调用外部函数前，把还要使用的变量保存到自己的堆栈中，如程序清单 3.8 所示，调用外部函数后再恢复，如程序清单 3.9，这样，在 Keil C51 V6.23 或更高版本中编译，相关函数就可重入了。

程序清单 3.8

```
#ifdef __C51__
    SP++;
    *((uint8 data *)SP) = Data;
#endif
```

程序清单 3.9

```
#ifdef __C51__
    Data = *((uint8 data *)SP);
    SP--;
#endif
```

3.5 系统优化

由于μC/OS-II并不是专为8位机的小内存系统而设计的，很多方面并没有考虑到节省内存。但是对于仅有128个RAM单元的80C51单片机而言，设计嵌入式操作系统一定要尽最大努力节约每一个RAM的使用，否则，就可能由于操作系统本身占用的内存过多，而完全没有实用价值了。

3.5.1 任务

鉴于C51单片机提供的存储空间相对较小，μC/OS-II的内核优先级调度函数用256字节的查找表,该表只能放在外部RAM，不仅读写周期长，也浪费程序空间。如果把最多任务个数降低到16个，能大大减少内存占用，并降低系统的复杂性。

针对本项目，使用的任务个数不会超过16个，可以对内核的数据结构和关键调度算法进行优化，而原有优先就绪表的结构如图2-3所示。

一、与任务相关的数据结构

为了充分利用 RAM，我们没有定义一个结构体（struct），把与任务相关的数据集中起来，而是根据数据的特点分别定义，并用尽量少的数据表示尽可能多的内容，也就是通过PU执行时间的增加换取内存占用的减少。

1. 常数表OSMapTbl[]

常数表 OSMapTbl[]是为了简化代码而定义的一个数据表格。它在很多地方使用到，主要作用是对标志的置位和复位，其定义如下：

程序清单 3.10

```
INT8U const OSMapTbl[] = {ox01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x00};
```

2. 就绪表

每个任务被赋予不同的优先级，从0到OX_MAX_TASKS（OX_MAX_TASKS表示最大任务数，由用户根据实际需要配置）。就绪表定义如下：

程序清单 3.11

```
#if OS_MAX_TASKS < 9
uint8 data OSFastSwap[1];                    /* 任务是否可以快速切换 */
#else
uint8 data OSFastSwap[2];
#endif
```

当用户的任务数小于9（OS_MAX_TASKS < 9）时，就绪表占用1个字节，当

用户任务数在 9～16 时，就绪表占用 2 个字节（对比 2.1.3 节，可见节省的内存很是可观）。就绪表每一位代表一个任务，当这一位为 0 时表示任务处于等待或挂起状态；当这一位为 1 时，表示任务处于就绪、运行或被中断状态。

二、与任务相关的算法

1. 使任务进入就绪态

函数首先判断参数（将要就绪的任务 ID）是否有效，如果参数太大（大于等于 OS_MAX_TASKS），则直接返回。由于设计时，把 ID 为 OS_MAX_TASKS 的任务设为最大任务，不允许用户直接操作，所以判断边界为 OS_MAX_TASKS。如果参数有效，则程序先关中断，因为已经进入临界区了。当用户任务小于 9 时，那么它直接将 OSTaskRuning 的相应位置 1。当用户任务数在 9～16 时，就要分两种情况了处理。当任务 ID 小于 8 时，其表示位在 OSTaskRuning 的低字节中，当任务 ID 在 9～15 时，其表示位在 OSTaskRuning 的高字节中，将相应的位置 1。最后开中断。

HIGH_BYTE，LOW_BYTE 定义 INT16U 型变量在特定 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 C)，则 HIGH_BYTE 为 0，LOW_BYTE 为 1；否则（如 8086 系列），HIGH_BYTE 为 1，LOW_BYTE 为 0。程序清单如下：

程序清单 3.12

```
void OSIntSendSignal(INT8U TaskId)
{
    if(TaskId < OS_MAX_TASKS)
    {
        OS_ENTER_CRITICAL();
#if OX_MAX_TASKS < 9
        OSTaskRuning |= OSMapTbl[TaskId];
#else
        if (TaskId < 8)
        {
            ((INT8U *)(&OSTaskRuning))[LOW_BYTE] |= OSMapTbl[TaskId];
        }
        else
        {
            ((INT8U *)(&OSTaskRuning))[HIGH_BYTE] |= OSMapTbl[TaskId];
        }
#endif
```

```
        OS_EXIT_CRITICAL();
    }
}
```

2. 使任务进入等待或挂起状态

此函数与使任务进入就绪态的函数十分相似。函数也是首先判断参数是否有效，如果参数太大，则直接返回。如果参数有效，则程序先关中断。当用户任务小于 9 时，那么它直接将 OSTaskRuning 的相应位清 0。当用户任务数在 9～16 时，就要分两种情况了处理。当任务 ID 小于 8 时，其表示位在 OSTaskRuning 的低字节中，当任务 ID 在 9～15 时，其表示位在 OSTaskRuning 的高字节中，将相应的位清 0 即可，然后开是中断。程序清单如下：

程序清单 3.13

```
void OSClearSignal(INT8U TaskId)
{
    if(TaskId < OS_MAX_TASKS)
    {
        OS_ENTER_CRITICAL();
#if OX_MAX_TASKS < 9
        OSTaskRuning &= ~OSMapTbl[TaskId];
#else
        if (TaskId < 8)
        {
            ((INT8U *)(&OSTaskRuning))[LOW_BYTE] &= ~OSMapTbl[TaskId];
        }
        else
        {
            ((INT8U *)(&OSTaskRuning))[HIGH_BYTE] &= ~OSMapTbl[TaskId];
        }
#endif
        OS_EXIT_CRITICAL();
    }
}
```

3. 任务调度

μC/OS-II 总是运行进入就绪状态的任务中优先级最高的一个。调度器就是用于确定哪个任务优先级最高和下面该运行哪个任务了。任务级的调度由函数

OSSched()完成。OSSched()的所有代码都属于临界区代码。函数一开始就关中断。查找进入就绪态并且优先级最高的任务实质是从最高优先级的任务（ID 为 0）起依次检测其是否为 1。为了最大限度地节约内存，当用户任务数小于 9 时，就绪表为 1 个字节，只需要一次循环。当用户任务数大于 8 时,，就绪表为 2 个字节，理论上也可只要一次循环，但是那样在 8 位机上运行效率很低，所以拆成 2 次循环。当所有用户任务均处于等待或挂起状态时，优先级最低的系统预定义任务将得以运行。循环将查找到的优先级最高的任务 ID 存于全局变量 OSNextTaskID 中，供函数 OS_TASK_SW()使用。然后调用 OS_TASK_SW()进行实际的任务切换，最后开中断。程序清单如下：

程序清单 3.14

```
void   OSSched(void) small
{
    INT8U temp;
    OS_ENTER_CRITICAL();
    if (OSIntNesting == 0)
    {
#if OX_MAX_TASKS < 9
        temp = OSTaskRuning;
        for (OSNextTaskID = 0; OSNextTaskID < OS_MAX_TASKS; OSNextTaskID++)
        {
            if ((temp & 0x01) != 0)
            {
                break;
            }
            temp = temp >> 1;
        }
        OS_TASK_SW();
#else
            temp = OSTaskRuning % 256;
            for (OSNextTaskID = 0; OSNextTaskID < 8; OSNextTaskID++)
            {
                if ((temp & 0x01) != 0)
                {
                    goto TaskSw;
```

```
                }
                temp = temp >> 1;
            }

            temp = OSTaskRuning / 256;
            for (OSNextTaskID = 0; OSNextTaskID < 8; OSNextTaskID++)
            {
                if ((temp & 0x01) != 0)
                {
                    break;
                }
                temp = temp >> 1;
            }
TaskSw:
    OS_TASK_SW();
#endif
    }
    OS_EXIT_CRITICAL();
}
```

程序切换分两步完成：将被挂起的任务的寄存器压入堆栈，然后将较高优先级任务的寄存器值从堆栈中恢复到寄存器中。

3.5.2 消息队列

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 void 型指针，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）。μC/OS-II 也是这样处理的。但是如果使用这种方式，传递一个消息至少要 3 个字节：2 个字节的指针，至少一个字节存放消息的内容。为了节省 RAM 的占用（这是 8 位操作系统的一个重要设计目标），我们把消息队列改成以 char 型变量（范围为 0～255）作为消息，而不是以指针指向的内容作为消息，这个 0～255 的值用户可以任意解释。

第四章　智能验钞机多任务实时系统应用软件的开发

在充分了解了μC/OS-Ⅱ内核之后，就要将其应用于实际的项目——智能验钞机系统的设计。这是一个实际的开发项目，需要将μC/OS-Ⅱ移植到该系统主板上的T89C51AC2芯片当中，采用μC/OS-Ⅱ嵌入式操作系统，根据实际应用和要求来划分在系统中任务，从而形成系统软件的构架，在构架搭建好之后，在每个任务中完成所需要的应用部分，从而使整个系统的软、硬件成为一个整体，完成实际检测的需要。

嵌入式系统在开发过程一般都采用“宿主机/目标板”开发模式，即利用宿主机(PC机)上丰富的软硬件资源及良好的开发环境和调试工具来开发目标板上的软件，然后通过交叉编译环境生成目标代码和可执行文件，通过直接烧录或串口/USB/以太网等方式下载到目标板上，利用交叉调试器在监控程序运行，实时分析，最后将程序下载固化到目标机上，完成整个开发过程。

4.1 项目背景

我国现金流通规模庞大，银行出纳柜台现金处理工作繁重，验钞机已成为不可缺少的设备。验钞机集计数和辨伪于一身，随着印刷技术、复印技术和电子扫描技术的发展，伪钞制造水平越来越高，加上人民币因使用率高而污损严重等不足，使得我们必须不断提高点钞机的检伪性能。

目前，市面流通的纸币主要是我国发行的第四套(1987年4月27日开始发行)和第五套(1999年10月1日开始发行)人民币，其中大面额纸币的机器检伪主要是利用磁性检测——纸币中的磁性安全线，磁性油墨，在磁头上运动时产生的磁性波，荧光检测——纸币经红外线、紫外线的穿透或反射后的物理特征，及宽度检测来判断。

现有的点钞机控制软件一般是用汇编语言编写，无操作系统，代码非常难懂，而且不好维护和修改，移植性差。我们移植一个嵌入式实时操作系统到单片机上，并使用易学好用，便于移植、复用的C51语言，简化了程序的设计过程，使得资源得到更好的利用，提高了开发效率和可移植性；与此同时我们还设计实现了新的验钞机的算法。我们在普遍采用的51系列芯片上，通过大量的实验，设计并实现了基于嵌入式实时操作系统的智能钞票鉴伪仪软件系统，在不改变验钞机微控制器及外围电路的基础上，大幅度提高鉴伪的精度和准确度。

4.2 项目分析

智能验钞机一般兼有验钞和点钞的功能，点钞功能实现比较简单，只需要验钞流程完毕之后计数管计 1，循环验钞功能，计数管循环计数就可以了，而验钞功能是项目实现的重点和难点，本文将重点阐述验钞的流程。国家规定伪钞鉴别仪必须具备两种以上的鉴伪手段[38]，本项目是通过宽度检测、磁性检测和荧光检测来实现的，图 4-1 为单张钞票检伪的实际工作流程，下面将结合硬件从这三方面来说明检伪的原理系统的设计与实现。

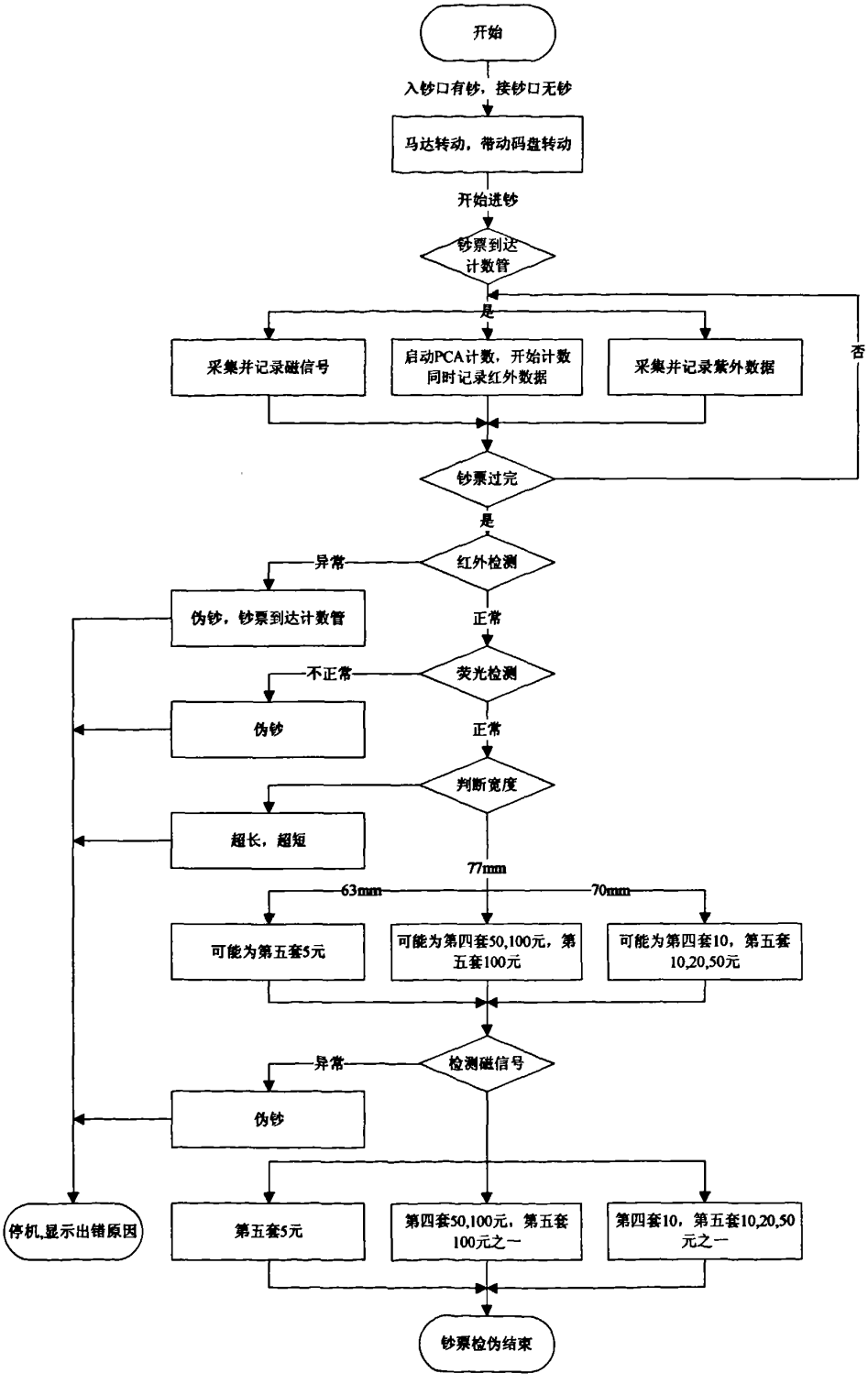


图 4-1 单张钞票的检伪流程

4.2.1 宽度检测

验钞机的检伪对象是第四套和第五套人民币中面额较大纸币，标准宽度有三种，分别为 63mm、70mm 和 77mm。其中 63mm 宽度的有第五套 5 元纸币；70mm 宽度的有第四套 5/10 元(检伪时不做区分)，第五套的 10、20、50 元面额纸币；77mm 宽度的有第四套 50，100，第五套的 100 元面额纸币。宽度判断是检伪中很重要的一个环节，测量和计算准确了，可以为进一步检测提供良好的基础——只要在对应的几种版本和面额的纸币中做判断了，从而降低了后续检测的工作量和难度。

一、算法描述

宽度检测部分由码盘、发光二极管与接收管组成。主要用来检测人民币的宽度，具有在左右计数管的配合下对过点钞机的人民币倾斜度进行检测，并根据该倾斜度用算法对人民币宽度进行修正。人民币在平行过与倾斜着过时码盘所测量出的宽度是不一样的。因此人民币清分机在人民币在倾斜通过时必须要能做出检测并予以修正，从而求出人民币的实际宽度。设计的修正方法是，T89C51AC2 的 PCA 中断服务程序中计数由码盘引起的 PCA 中断的次数，人民币进左右计数管时记下已经数到的次数，出左右计数管后记下已经数到的次数。分别记下人民币进出左右计数管时已经数到的由码盘引起的 PCA 中断的次数，根据人民币进出左右计数管时刻由码盘引起的 PCA 中断的次数的差值大小对测量的人民币宽度值进行修正，求出人民币准确宽度。

这个过程的平面示意图是：

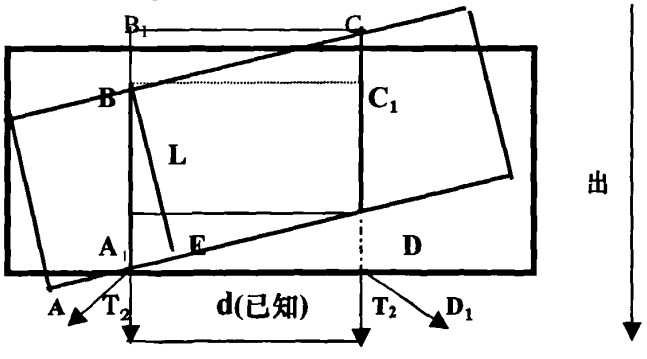


图 4-2 宽度计算

在这个图 4-2 中，有两个矩形(相当于一张平行放置，一张倾斜放置)，其中 d= T_1T_2为已知长度，点 A 为左计时器开始计时点，也是第一个入点，点 D 为右计时器的入点，B 为左计时器的出点，C 为右计时器的出点，AD_1，A_1D，BC_1，B_1C，都平行于 T_1T_2 。DD_1 ，BB_1可以测量出，AB，DC 也可以测量出来(即都为已知)，L=BE 垂直于线段 AD,则 BE 为所要求的倾斜矩形的宽度。

要根据单片机的计算原则设计，并且在设计得出的最后算法时为了快速计算尽量的不要应用开方、次数比较高的函数和正余弦、正余切函数。

显然，当是无倾斜的人民币通过左右计时，其宽度可用 $\frac{AB+CD}{2}$ 得出，当人民币的倾斜度很小(一般认为小于 5 度)时，人民币的宽度也可以用 $\frac{AB+CD}{2}$ 近似的求出。这个可以通过验算很容易地证明。

当人民币的倾斜度大于 5 度的一个不是太斜的范围内时(≤42.5。若倾斜度太大，磁性检测、荧光检测、紫外检测和红外检测等各项测试指标都会变的很不标准，且经过计算在倾斜度大于 43 度时采样值就被自动舍弃。因此我们只记 42.5 度以下的值，这个思想在宽度处理过程中体现出来。)钞票过左右计数管有两种情况：非倾斜过钞(倾斜度小于 5 度)和倾斜过钞(倾斜度在 5 度到 43 度之间)。对于非倾斜过钞，我们直接确定其宽度。对于倾斜过钞，我们设计取逼近多项式 BE = c(39382-a^2)/39382 来计算人民币的宽度。它们具体确定过程如下：

其中 a 和 c 的确定如下。

Lsp 表示钞票左起点的码盘数；

Lep 表示钞票左出点的码盘数；

Rsp 表示钞票右起点的码盘数；

Rep 表示钞票右出点的码盘数；

Left 表示钞票左边宽度的码盘数；

Right 表示钞票右边宽度的码盘数；

Dif 表示钞票两边的宽度的码盘数之差；

End_f 表示钞票两边的出点的码盘数之差。

在实际的程序实现时，根据经验，通过对所获得的数据的分析和统计以及反复验证，为了提高计算的精度，将 lsp,lep,rsp,rep 的计算值比实际测到的值放大一倍，并给出 a 和 c 的取值依据，如下：

a 取值为(rsp+rep-lsp-lep)/2(a=$\frac{BB_1+DD_1}{2}$)

(1)当钞票两边的宽度的码盘数之差小于 6(dif<=6)时（非倾斜过钞）

① 如果钞票右起点的码盘数为 0(rsp=0)， c 取值为左边宽度的码盘数(c=left)或右边宽度的码盘数 (c=right)；

② 如果钞票右起点的码盘数不为 0 。

a. 如果钞票两边的出点的码盘数之差小于 4(end_f<4)并且钞票右起点的码盘数小于等于 2。若钞票右边宽度的码盘数大于等于钞票左边宽度的码盘数 c 取值为右边宽度的码盘数(c=right)，否则 c 取值为左边宽度的码盘数(c=left)。

b.如果钞票右边宽度的码盘数比钞票左边宽度的码盘数大 2 以上并且钞票右起点的码盘数小于等于 6 (right-left>=2&&rsp<=6)， c 取值为左边宽度的码盘数(c=left)。

c. 如果不符合上面两种情况那么 c 取值为左右两边宽度的码盘数的平均值(c=(right+left)/2)。

计算宽度：宽度＝c。

(2)当钞票两边的宽度的码盘数之差大于 6(dif>6)时（倾斜过钞），依次检查符合下列哪种情况

①如果钞票右出点的码盘数比钞票左出点的码盘数大 4 以上并且钞票右起点的码盘数小于等于 6 (rep-lep>4&&rsp<=6) 时 c 取值为左右两边宽度的码盘数的平均值(c=(right+left)/2)。

②如果左出点的码盘数大于右出点的码盘数并且钞票右起点的码盘数小于等于 2(lep>rep&&rsp>=2)或者钞票两边的出点的码盘数之差小于 4 并且钞票右起点的码盘数大于等于 8(end_f<4&&rsp>=08))时 c 取值为钞票左右出点的码盘数的平均值(c=(rep+lep)/2)。

③如果右出点的码盘数小于 6(rsp<6) ：

若钞票右边宽度的码盘数大于等于钞票左边宽度的码盘数 c 取值为右边宽度的码盘数(c= right)，否则 c 取值为左边宽度的码盘数(c=left)。

④如果不符合上面的情况那么 c 取值为左右两边宽度的码盘数的平均值(c=(right+left)/2)。

计算宽度：宽度＝c(39382-a^2)/39382。

若为了在硬件基础上提高计算的精度而采用格数为 100 孔的码盘来计数。此时有 100 个孔，那么码盘转一圈可计数 100 次(每个孔和其间隔产生一个跳变)，一次代表约 1.21mm。由于将码盘数放大了一倍，可以认为码盘转一圈计数 200 次，一次代表 0.605mm。不过由上逼近多项式的推导可以看出求的系数 0.32 与实际宽度无关，那么可以通过计算可以得到新的标准，如下：

63*200/121=105；70*200/121=116；77*200/121=128；用取特殊点的方法求出要找的范围。

宽度为 63mm 时，根据多项式算得 BE_{max}=109 ， BE_{min}=105。则其偏差为 4*0.605=2.42mm。因纸币在有时会发生一定的变形，所以误差范围定为[104，109]。

宽度为 70mm 时，根据多项式算得 BE_{max}=120， BE_{min}=116。同上误差范围定为[115,120]。

宽度为 77mm 时，根据多项式算得 BE_{max}=133， BE_{min} =127。同上误差范围

定为[126,133]。

所以，在本验钞机斜纸币的宽度可由逼近多项式 BE = c(39382-a^2)/39382 计算而得到。在各种人民币点钞机中都可以采用此逼近多项式计算人民币的宽度。可应用于银行系统中，大型企业的清分机和点票机中。另外本多项式是采用逆向推导的计算方法得出来的，因此它的稳定性很可靠，在人民币没有发生很大收缩和变形时，且倾斜角度在有效范围内，一般采样设计时度数小于 43 度时它测的数据不会产生很大的误差。此逼近多项式的产生，在智能点钞机中就给出一个新的计算方法。在程序中每计算一张用时 700 多微秒，是一个高效算法。

二、计算宽度的程序实现

宽度程序（kuandu.c）中根据采集到的钞票的左右出入点码盘数计算出钞票的宽度值，或者报与宽度有关的钞票异常错误，包括半张、纸条、中间有缝、残张、连张等异常。

程序开始，由于在宽度程序中要用到相关的红外信息，所以开始就将存储在钞票缓冲区红外信息读出到全局变量中。这些红外信息包括左右红外信息数组以及它们的个数，左右红外值中的最大值最小值，半张标志。

先根据在采集红外信息时是否置半张标志判是否半张票据。然后再根据红外信息的后面几个值判是否纸条（因为如果是纸条，它的后面几个红外值将比较的大）。

接着判是否钞票中间有缝。（通常是看红外的最大值以及超过某一标准值（0xe0）的红外值的个数。）通常三种情况将会判为中间有缝。(1)右红外值中超过 0xe0 的个数超过三个；(2)右红外值中超过 0xe0 的个数超过三个并且左红外最大值大于 0xe0；(3)左右红外值中超过 0xe0 的个数和超过 3 个。三种情况依次判断。

再计算宽度，计算宽度有两种情况：一种是非斜钞，另一种是斜钞。非斜钞宽度计算相对简单，其宽度可以直接确定。斜钞的宽度先确定好参数，再根据式子 c(39382-a^2)/39382 计算出宽度。

算好宽度后，根据算好的宽度值可判为以下几种情况：

1. 宽度小于 0x61（度量单位：码盘个数×2）：这种情况判为残张；

2. 宽度大于等于 0x61 且小于 0x6E：这种情况修正为 0x3f，对应钞票的宽度为 63mm；

3. 宽度大于等于 0x6e 且小于等于 0x71：这种情况修正为 0x11，对应 63mm 和 70mm 的临界区；

4. 宽度大于等于 0x72 且小于 0x7b：这种情况修正为 0x46，对应钞票的宽度为 70mm；

5. 宽度大于等于 0x7b 且小于等于 0x7f：这种情况修正为 0x12，对应 70mm 和 77mm 的临界区；

6. 宽度大于等于 0x80 且小于等于 0x98：这种情况修正为 0x4d，对应钞票的宽度为 77mm。

宽度程序返回 5 类宽度值或者有关宽度的异常钞票错误。这 5 类宽度将再 cijian.c 程序中被引用，和磁检结合在一起用于识别钞票的版别和面额。并且标志临界宽度的两个宽度（0x11 和 0x12）将在 cijian.c 作出修正，最终钞票的宽度将只有 0x4d, 0x46, 0x3f 三个值，也即只有 77mm, 70mm, 63mm 三个值。

4.2.2 磁性检测

钞票上有两种磁性部位，分别是磁性油墨和磁性安全线。钞票的磁性部位在与磁头磨擦时，会产生特定的磁信号。第四套的 50 和 100 元面额及第五套 5 元以上的钞票都有上面两种磁性。我们通过三组磁头分路检测磁性油墨和金属条的磁分布，能够准确识别第四套 10 元以上和第五套 5 元以上的钞票版别(第四套和第五套)和面额。

一、磁性油墨

磁性防伪油墨是采用具有磁性的粉末材料（ 氧化铁和氧化铁中掺钴等）作为一种功能成分所制作的防伪印刷油墨。它是最常应用的防伪油墨，其突出的特点是外观色深，检测仪器简单，主要用于印刷银行票证的磁性编码文字和符号，具有记录和存储信息的功效，将印有磁性编码的票证投入磁码识读器中可辨识真伪[29]。在本验钞机上设计了两组小磁头，用以检测出微弱磁性油墨，通过有无磁性，和磁信号的分布来判断其版别和面额。

二、磁性安全线

磁性安全线在目前的钞票防伪中占据了很重要的地位，在第四套人民币中的 50、100 元面额钞票中开始使用，并在第五套人民币中全面使用。对第四套人民币的磁性辨伪最初是检测钞票安全线有无磁性，后来提高到对钞票的磁性分布和强弱进行分析，但由于客观原因要进一步提高辨伪水平比较困难。第五套人民币由于提高了机读性能，给鉴别仪准确辨别人民币真伪创造了条件。

安全线为比较薄、比较窄的金属或塑料线，宽度在 1 mm 左右。厚度在 0. 03 mm 上下，造纸的过程中把他夹在纸张上固定的位置。灌入了这种编码信号后，就如同人有了“指纹”一样，不但可以利用这些“钞票指纹”来分辨真伪，而且可以准确无误地分辨“面额”与“版本”。每种券别（5、10、20、50、100 元）所灌充的信号都不尽相同，如图所示：人民币安全线中的磁信号很有规律，由若

干个“单信号”构成一“组”，再由若干“组”结成“链”，这种信号必须由特殊的专用设备灌充，灌充之后难以抹掉，伪钞很难做到这一点。

我们通过一组磁头来对些信号进行解码，当票面经过由磁阻型传感器磁头时，产生相应的磁通量的变化，致使磁敏电阻值随之变化。该变化的电阻值转变成电压信号后经后级电路进行信号的处理，再送给单片机进行真伪的判别。单片机通过分析磁信号的间隔和占空比，将信号分为若干组，用该信号和预先存储的人民币固有的信号进行比较，即可识别。

三、程序设计实现

磁检程序(Cijian)根据采集到的磁信息结合宽度，检红外信息，参照相关检荧光信息，判别钞票的面额和版本，或识别为伪钞异常时给出原因。

使用到的信息来自钞票信息缓冲区的磁信息，包括中磁脉冲宽度信息，左磁个数，右磁个数，中磁个数，左磁脉冲宽度最大值，右磁脉冲宽度最大值。先将这些磁信息读入到全局变量中，再对这些全局变量中的信息进行处理。

Cijian 程序在任务 4 中调用。带返回值，返回值要么为真钞，如下：

返 回 值：0x01　new100
　　0x02　new50
　　0x03　new20
　　0x04　new10
　　0x05　new5
　　0x11　old100
　　0x12　old50
　　0x13　old10,old5

如果判为不是真钞，即不能识别为以上版别、面额，则返回通过磁检标记（0x00），并置相应错误原因代码于错误代码中（errcode）。在本程序中，只有 JC 错和 JD 错，JB 在任务 4 中 judge_cixinhao()中与判夹版一起处理，即在检查不通过安全线检查时，返回 0x00 后在 judge_cixinhao()中的 DEFAULT 中一次处理。特别注意:在判断老版币（widthold77 和 widtholdcritical77）时,当 JD 开关关闭（不检磁分布），如果检查发现 JD 错将不报 JD 错但是会报 JB 错（通过返回 0xfe 在 judge_cixinhao()中的 DEFAULT 中处理）。同样的对于判老版币的 JC 时也是如此处理。而在判断新版 100 元时是先判 JC，如果判 JC 开关关闭再判 JB 安全线，在这里如果检 JC 关闭则不检 JC 只检 JB。

cijian 根据磁信号结合宽度和红外检测，判别钞票的面额和版别，如果不能判别(假钞或异常)则给出错误的原因。在磁检中可能会检测出三类错误:JB(安全线个数和特征)，JC（左右磁个数和特征），JD（磁的分布特征）。其中新版币

判 JB，新版 100 还需判左右磁。旧版币判 JC 和 JD。判断过程如下：首先根据中磁个数和中磁宽度数组中前 12 个宽度之和((mmcount>=10&&count12>=39&&kuandu==0X4d)||((kuandu!=0X4d)&&((mmcount>=5&&count12>=20)||(count12>=30))))区分新版币和旧版币，然后分别判断新版币和老版币的面额。

1.判断新版币的面额 judge_new()：因为新版 20 元具有两个非常明显的特征，最容易识别，所以先根据它的特征来识别是否新 20 元，不是再判是否其它。如果不是新版 20 元，再结合宽度来判其它面额的新版币。新版币有 5 个宽度：

0x4d(77mm)：只有 100 的宽度才为 77mm，所以当宽度为 0x4d 时判是否满足新 100 安全线特征和左右磁特征，若满足则返回代表新 100 的代号，若不满足则返回错误的原因：JC 或 JB，JB 在 main 中的 judge_cixinhao 中判断。

0x46(70mm)：当宽度落在这个范围之内的时候，如果先前所判票据面额（钞票标准）是新 100 或旧 100，则检查票据是否通过临界宽度内新 100 元票据检测，如能通过则计为新 100 并修正宽度，否则检查是否通过新 50 的磁检，如能通过计为新 50，否则再检查：当先前所判票据面额（钞票标准）是新 5 元则检测是否通过新 5 元磁检，若能通过计为新 5 元并修正宽度，否则在进行新 10 元的磁检。

0x3f(63mm)：当宽度落在这个范围之内的时候，如果先前所判票据面额（钞票标准）是新 50 或旧 50，则检查票据是否通过新 50 元票据磁检，如能通过则计为新 50 并修正宽度，否则检查：如果前面所判票据面额（钞票标准）是新 10 元，则检查是否通过新 10 元磁检，若能通过则计为新 10 元，否则判为新 5 元。

0x11(介于 63mm 和 70mm)：落在这个宽度内的钞票，可能是新 50 元，新 10 元，新 5 元。先根据前 12 中磁宽度是否可能是新 50，如果可能则判是否通过新 50 的磁检，若能通过则计为新 50 并修正宽度，否则判是否新 5 元，若是计新 5 元，否则再判是否新 10 元，若是计新 10 元，否则再根据先前所判票据面额（钞票标准）放松检查新 10 元条件，符合条件则判为新 10 元，否则再根据先前所判票据面额（钞票标准）放松检查新 5 元条件，符合条件则判为新 5 元，如果条件还不能满足，要么判为 new10,要么判为 new5,如果前面所判钞票为 new10,则先判为 new10，否则判为 new5。

0x12(介于 70mm 和 77mm)：落在这个宽度内的钞票，可能是新 100 元，新 50 元，新 10 元。先根据前 12 中磁宽度是否可能是新 100 或新 50。根据统计资料，落在此区域中新 100 比新 50 的要多，因此判定时偏向新 100。先判是否通过新 100 在临界宽度内的磁检，若通过则计新 100 元并修正宽度，否则再判是否通过新 50 元磁检，若能通过则计新 50 元并修正宽度，否则（不能判为新 100 元和新

50 元）则判是否通过新 10 元磁检。

在 judge_new()中，调用了以下四个子程序：

(1)widthnew77()：判新版 100。

(2)widthnewcritical77()：判处在临界宽度 70－77mm 的新版 100。

(3) widthnew70()：判新版 50。

(4)widthnewcritical77()：判处在临界宽度 70－77mm 的新版 100。

(5) widthnew70()：判新版 50。

(6) jian10()：判新版 10。

2. 判断老版币的面额 judge_old()：判断老版币先检查红外，若不能通过红外检测则返回。通过红外检测后方可检测其真伪和面额。老版币也有 5 个宽度：

0x4d（77mm）：可能是旧 100 或旧 50，类同，当钞票标准是该种币时可适当放松判别的条件。

0x46（70mm）：如果先前的钞票判为新 100、50 或旧 100、50,则判临界宽度内的旧版钞票，并修正宽度，如果不是旧版的 100 元 50 元，通过红外光鉴后判为旧 10 元或旧 5 元。

0x3f（63mm）：若之前判为旧旧 10 或旧 5 或新 10 并且通过红外监测则判为旧 10 元或旧 5 元并修正宽度。

0x11（介于 63mm 和 70mm）：通过红外光鉴后判为旧 10 元或旧 5 元

0x12（介于 70mm 和 77mm）：通过红外光鉴后，判临界宽度内的旧版钞票，并修正宽度。

在 judge_old()中调用了两个子程序：

(1)widthold77:判旧版 100 或旧版 50。

(2)widtholdcritical77()：判处在临界宽度 70－77mm 的旧版 100 和旧版 50，旧版 10 或旧版 5。

4.2.3 荧光检测

光检是根据真钞纸质特殊，在经光线的穿透或反射后，光信号的强度有其独有特性，从而达到检伪的目的。通过又可分为红外检伪和紫外(荧光)检伪。

一、红外穿透检伪

红外穿透的工作原理是利用人民币的纸张比较坚固、密度较高以及用凹印技术印刷的油墨厚度较高，因而对红外信号的吸收能力较强来辨别钞票的真假。人民币的纸质特征与假钞的纸质特征有一定的差异,用红外信号对钞票进行穿透检测时，它们对红外信号的吸收能力将会不同，利用这一原理，可以实现辨伪。需要注意的是，油墨的颜色与厚度同样会造成红外穿透能力的差异。因此，必须对

红外穿透检测的信号进行数学运算和比较分析[30]。

我们用检测宽度的两组红外传感器来实现红外穿透检伪，在判断进钞的同时，将红外值保存到数据缓冲区中。本项目使用的是红外发射和接收配对的发光二级管，当发射红外线去控制相应的受控装置(钞票)时，其控制的距离与发射功率成正比。为了增加红外线的控制距离，红外发光二极管工作于脉冲状态，因为脉动光(调制光)的有效传送距离与脉冲的峰值电流成正比，只需尽量提高峰值I_P，就能增加红外光的发射距离。提高I_P的方法是减小脉冲占空比，即压缩脉冲的宽度T，一些彩电红外遥控器，其红外发光管的工作脉冲占空比约为1/4-1/3；一些电气产品红外遥控器，其占空比是 1/10。减小占空比还可使小功率红外发光二极管的发射距离大大增加。

要使红外发光二极管产生调制光，只需在驱动管上加一定频率的脉冲电压。用红外发光二极管发射红外线去控制受控装置(钞票)时，有相应的红外光电转换元件红外接收二极管。红外线发射与接收的方式有2种：直射式和反射式。直射式指发光管和接收管相对安放在发射与受控物的两端，中间相距一定距离；反射式指发光管和接收管并列在一起，平时接收管始终无光照，只在发光管发出的红外光遇到反射物时，接收管收到反射回来的红外线才工作。对于纸张的质量和油墨的厚度都使用红外发射和接收配对的二极管，发光管和接收管相对安放在发射与受控物(钞票)的两端，中间相距一定距离。由于纸张的好坏和油墨的厚度大小使红外接收管电压不同，进而使A/D转换值不同，因而可以区分真假钞。

二、紫外穿透检伪

我们是通过一对紫外传感器来实现紫外检伪的，当钞票到达计数管时，就可以进行紫外信号采集了，同时计数管计1。与红外穿透的工作原理类似，紫外检伪的工作原理是针对人民币的纸质进行检测，而纸质的识别是通过对印制纸币的纸质性能的检测来完成。人民币采用专用纸张制造（含85%以上的优质棉花），由特种纤维组成，具有完全吸收(无反射)紫外线的特点。假钞通常采用经漂白处理后的普通纸进行制造，不吸收紫外线，而反射紫外线。经漂白处理后的纸张在紫外线（波长为365nm的蓝光）的照射下会出现荧光反应（在紫外线的激发下衍射出波长为420－460nm的蓝光），人民币则没有荧光反应。本验钞机正是利用这一原理来设计。

本验钞机运用紫外光源对运动钞票进行照射并同时用硅光电池检测钞票的荧光反映，来判别钞票真假。在点钞机工作时，欲判断纸币的真伪，要求紫外线探测器快速将光信号转换为电信号。在各种转换品中，硅光电池对近紫外线的相对灵敏度要高一些，响应速度比硒光电池快 5—6 个数量级，工作稳定性优于1-2%/年，所以选用硅光电池。硅光电池是利用光生伏特效应来工作，其光谱范

围为 200—1100nm，其光谱灵敏度的峰值波长位于近红外区(800nm)，在紫外，其灵敏度较低，比兰、紫光还低。

紫外线则由紫光管产生。紫光管产生的峰值波长为 351nm(365nm)，此外，还有大量的紫光及兰光。所以假币在紫光管的照射下，其反射光中除紫外线外，还有紫色及兰包光，故在探测器中，硅光电池输出的电信号中今有大量的非紫外部分(兰色及紫色所产生的电信号)。因此，在判断时易产生误判。所以，关键是对探测器的处理及对紫光管输出谱线的纯化。

1、对探测器的处理

假币在紫光管的照射下，反射光中除紫外线外还有紫光及兰光为了谱线纯正应对探测器处理。我们让反射光通过两个滤色片，一块深兰色(QB24)，一块紫色(ZB3)，则到达探测器的只有紫外线(300—400hm)。这样到达带有滤色片的硅光电池的光谱中只有紫外线，并且对 351nm 最为敏感，从而避免了其它杂散光的影响，保证了检测的准确性。

2、紫光管输出谱线的纯化

紫光管应用在验钞机中，将灯管做成直管形，4W 、6W 、8W。其工作原理与普通荧光灯相同，属于热阴极低气压放电灯。其电参数和尺寸与普通荧光灯一样。由前可知，紫光管谱线纯正与否是点钞机准确判断的关键，而其谱线纯正的关键取决其所用的原材料黑光粉及玻璃管。采用不同的黑光粉，其发射主峰不同，即 351nm 或 365nm，前为低效粉，后为高效粉；光衰与效率也不一样。玻管也很重要，它应全部吸收可见光，保证紫光管输出光谱的纯正。

4.3 系统实现

验钞机是机电一体化产品，涉及机械、电、光、磁、软件等多个领域的知识，需要各方面互相配合，本文仅从软件设计方面加以阐述。

系统主要由一个主程序，5 个用户任务（分别是启停监测、按键监控、信息处理、数据输出和参数设置）3 个中断（分别是定时器 0、定时器 2、PCA 中断）及调试模块，操作系统，底层硬件驱动模块组成。

开机后，主程序初始化各系统变量、自检后创建 5 个任务后，一直处于就绪状态，启停监测和按键监控任务定时由定时器 0 激活。在启停监测到有钞时，启动马达，激活 PCA 码盘中断，一方面采集磁信号，另一方面再激活定时器 2，采集左右计数管数据，当钞票两边都已经到达计数管后，发送信号激活荧光采集任务；当钞票两边都已经离开计数管后激活信息处理任务，进行检伪、计数、显示、出错等处理。操作系统的功能是任务调度和内存管理。底层硬件驱动有 AD 转换、

马达驱动，eeprom 读写。下面就这些模块详细描述。系统结构如图，最底层是系统硬件，其设计由硬件工程师完成，其上是接口/驱动层，再上是操作系统层，最上方是应用层。系统硬件如图 4-3 所示包括 CPU/定时器、LED、键盘、串口蜂鸣器、码盘、磁头、马达以及红外/紫外传感器。本文将从接口/驱动层开始对上面三层的设计与实现一一进行阐述。

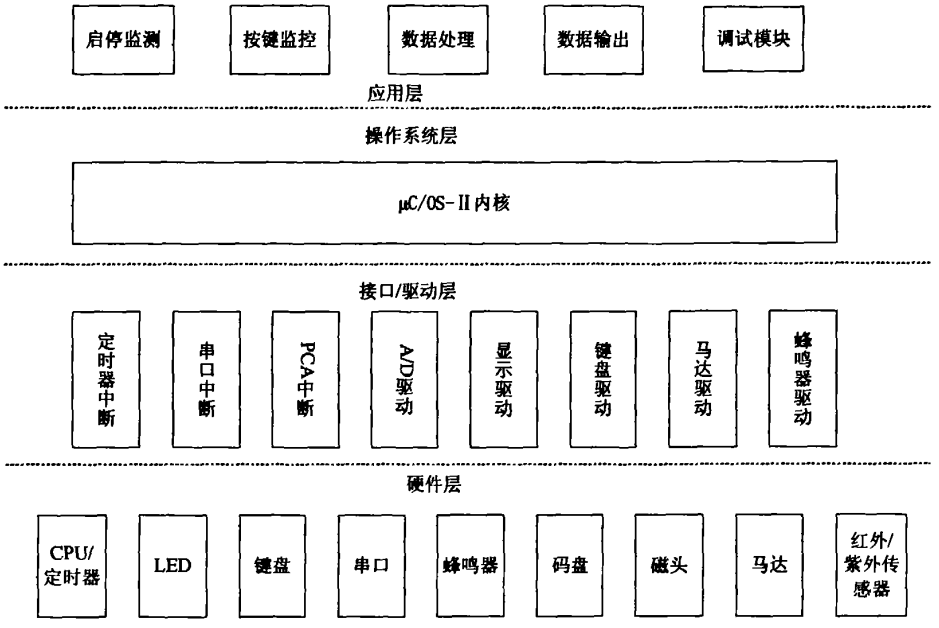


图 4-3 系统结构图

4.3.1 接口/驱动层

嵌入式系统的开发需要与硬件紧密结合，协同工作。驱动程序是用来向操作系统提供一个访问、使用硬件设备的接口。中断是一种硬件机制，指 CPU 对系统中或系统外发生的异步事件的响应。异步事件是指无一定时序关系的随机发生的事件，如外部设备完成了数据传输任务，某一实时控制设备出现异常情况等。在本系统中，主要用于采集处理钞票信息。

一、A/D 转换驱动

我们的计算机与单片机只能处理数字信号，凡是有模拟信号的地方，都要转化成数字信号,这就是用 A/D 转换。T89c51AC2 内部有 ADC (Analog-to-Digital Converter) 模数转换器。ADCON 的低 3 位表示 AD 通道，能支持 8 个采样通道，通过一个模拟多路复用器，ADC 能从 8 个通道中选择一个，作为输入电压。它进行一次 A/D 转换的时间是 20ms.

通道0和通道1分别对应左右计数管，通道2对应启停管，通道3是扩充通道位，其对应通道的具体如下：

右计数管	TRR	0X00	AN0				
左计数管	TLR	0x01	AN1				
启停管	STAR	0X02	AN2				
防漏管	END	0X03	AN3		TO	0x13	AN3
左水印	SYL	0X23	AN3	右水印	SYR	0X33	AN3
透射	JE	0X43	AN3	反射	JY	0X53	AN3
美元	JU	0x63	AN3				

在开始采样前先确定其他的AD采样已经完毕。

ADCON

2-0 SCH2:0 Selection of Channel of Convert

3 ADSSET Start and Status

Set to start an A/D conversion

Cleared by hardware after completion of the conversion

4 ADEOC End Of Conversion

Set by hardware when ADC result is ready to be read. This flag can gengerate an interrupt.

Must be cleared by software.

5 ADEN Enable/Standby Mode

Set to enable ADC.

Clear for Standby mode(power dissipation 1 uW).

0X03 - 0X63 共享AN3输入，具体为那个信号输入取决于P2.0-P2.2,

当P2.0-P2.2为000时，选择0x03,

当P2.0-P2.2为001时，选择0x13,

当P2.0-P2.2为010时，选择0x23,

当P2.0-P2.2为011时，选择0x33,

当P2.0-P2.2为100时，选择0x43,

当P2.0-P2.2为101时，选择0x53,

当P2.0-P2.2为110时，选择0x63.

在赋通道值时为避免相互干扰采用按位赋值的方式，因此可以将启停防漏，荧光反射透射分开写。在程序中借助寄存器ACC和B，AD采样值在ADDH中，先赋给B再返回传出。

程序清单 4.1

```
INT8U AD(channel)
{
    for(; !(ADCON & 0X08) ;); //将 ADCON 的第 3 位是忙标志，当一个 AD 转换进行时被置位，转换完成后被硬件复位
    ADCON |= 0x07;
    ADCON &= channel;          //设置 AD 通道
    ADCON |= 0X08;             //开始转换

    for(; !(ADCON & 0X10)!=0X10 ;); //ADCON 的第 4 位是转换完成标志
    return (ADCON & 0X10);
}
```

二、EEPROM 读写驱动

EEPROM，或称 E^2PROM，全称为电可擦除可编程只读内存(Electrically-Erasable Programmable Read-Only Memory)。相比 EPROM，EEPROM 不需要用紫外线照射，也不需取下，就可以用特定的电压，来擦除芯片上的信息，以便写入新的数据。数据在断电后仍能长期保存，所以可用来存放硬件设置数据。

T89c51AC2 带有 2K 字节的片上 EEPROM 内存，在 XRAM 内存空间的地址为 0000h - 07FFh. EECON 是 EEPROM 的控制寄存器，第 0 位是忙标志位，置位时表示正在进行写入；第 1 位是使能空间位，置位时能进行寻址；第 4-7 位是写入命令位，先置为 5Xh，再置为 Axh 就会开始写入。

程序清单 4.2

```
//将数据 cx 写入 EEPROM 中的 p 地址
void wr_eeprom(char xdata *p,char xdata cx)
{
    While(!(EECON&0x01)) ;  // wait while eeprom is working
    EECON |=0x02;
    *p= cx;
    EECON |=0x50;
    EECON |=0xA0;
}
//从 EEPROM 中的 p 地址读取数据
char rd_eeprom(char xdata *p)
{
```

```
    While(!(EECON&0x01)) ;   // wait while eeprom is working
    EECON |= 0x02;
    return(*p);
}
```

三、 LED 显示驱动

用单片机驱动 LED 数码管有很多方法，按显示方式分，有静态显示和动态（扫描）显示，按译码方式可分硬件译码和软件译码之分。静态显示就是显示驱动电路具有输出锁存功能，单片机将所要显示的数据送出后就不再管，直到下一次显示数据需要更新时再传送一次新数据，显示数据稳定，占用很少的 CPU 时间。

本系统使用的是动态显示、软件译码，如图 4 所示。

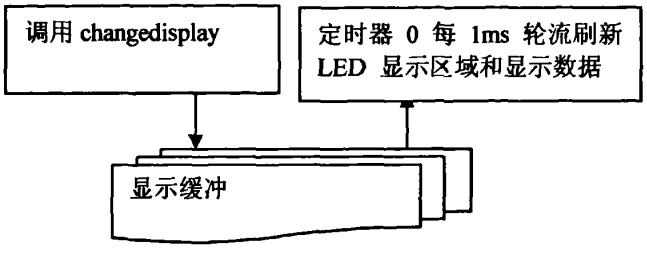


图 4: LED 显示驱动

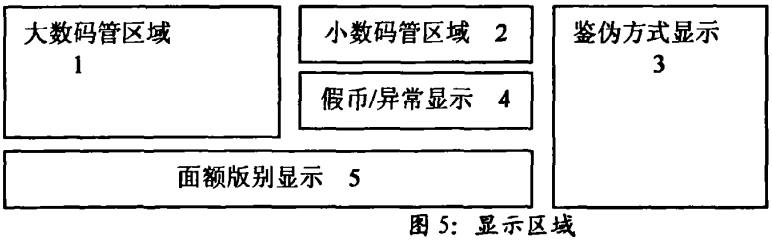


图 5: 显示区域

动态驱动是将所有数码管的 8 个显示笔划的同名端连在一起，另外为每个数码管的公共极 COM 增加位选通控制电路，位选通由各自独立的 I/O 线控制，当单片机输出字形码时，所有数码管都接收到相同的字形码，但究竟是那个数码管会显示出字形，取决于单片机对位选通 COM 端电路的控制，所以我们只要将需要显示的数码管的选通控制打开，该位就显示出字形，没有选通的数码管就不会亮。通过分时轮流控制各个数码管的的 COM 端，就使各个数码管轮流受控显示，这就是动态驱动。在轮流显示过程中，每位数码管的点亮时间为 1～2ms，由于人的视觉暂留现象及发光二极管的余辉效应，尽管实际上各位数码管并非同时点亮，但只要扫描的速度足够快，给人的印象就是一组稳定的显示数据，不会有闪烁感，动态显示的效果和静态显示是一样的，能够节省大量的 I/O 端口。本系统在定时器 0 中断中刷新显示，频率达 1000HZ，显示十分稳定。程序如下所示：

程序清单 4.3

```
//显示函数
void display()
{
    P4_1=0;
    P4_0=1;

    P0=dispdata;
    P4_0=0;
    P0=disparea;
    P4_1=1;
    P4_1=0;
}
```

disparea 表示显示哪一个区域，即点亮哪一个数码管，显示字符 dispdata 一般为如下编码字符（共阴）。

程序清单 4.4

```
char code seg[]={
    0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f   // 0,1,2,3,4,5,6,7,8,9
    ,0x77,0x7c,0x39,0x5e,0x79,0x71};                    // a, b, c, d, e, f
```

四、键盘驱动

键盘是人机交互的主要设备，按键在按下和松开时均会产生抖动，抖动时间一般为 5~10ms。按键抖动会引起一次按键被误读多次。此键盘共有 3 个键，对应串口 p2.5/p2.6/p2.7，低电平有效。在单片机获得 P2.x 口为低的信息后，不是立即认定 S1 已被按下，而是延时 20 毫秒或更长一些时间后再次检测 P2.x 口，如果仍为低，说明 S1 的确按下了，这实际上是避开了按键按下时的抖动时间。而在检测到按键释放后再对键值处理。本文采用了软件去抖动算法，即在检测出按键闭合后延时一段时间再进行判断，如果仍与开始结果一致，则认为按键有效。

程序清单 4.5

```
INT8U keyIn(void)
{
    INT8U   key;
    key=(P2&0xe0);                          //P2 高 3 位对应按键，低电平有效
    while(1)
    {
```

```
            OSTimeDly(15);
            if (key==(P2&0xe0))        //两次结果一样, 按键有效
                break;
            else
                key=(P2&0xe0);
        }
        return(~key&0xe0);
    }
```

五、串口通信中断

由于系统是在 windows 环境下开发，运行在验钞机 MCU 上，这就给调试带来极大的不便，为能观察到系统运行时各变量的值的变化，我们把需要测试的数据通过串口传输到 PC 机上，以便调试。串口通信是使用中断方式进行的。

程序清单 4.6

```
//串口中断函数
#pragma disable
void serial ( void) interrupt 4              //中断号 4
{
    bit    start_trans = 0;
    OS_INT_ENTER();
    if (TI || start_trans)
    {
        TI = 0;
        if (ostart != oend)
        {
            if (!sendstop)
            {
                SBUF = outbuf[ostart++ & (OLEN-1)];
                sendfull = 0;
            }
        }
        else
        {
            sendactive = 0;
        }
    }
```

```
    OSIntExit();
}
```

六、定时器中断

1. 定时器 0 中断（系统时钟节拍）

定时器 0 是系统的时钟节拍，其中断可以看作是系统心脏地跳动，中断之间的时钟间隔取决于不同的应用。时钟的节拍式中断使得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。

程序清单 4.7

```
void OSTick0ISR()  interrupt 1
{
    TL0 = TIMER0_L;
    TH0 = TIMER0_H;
    OSIntEnter();
    OSTimeTick();
    OSIntExit();
}
```

2. 定时器 2 中断（左右计数管（红外）数据采集）

定时器 2 中断服务程序监视过钞情况，采集过钞原始数据：红外信号。根据采集到的红外信息记录钞票的宽度信息。对采集到的红外信息找出其中的最大值最小值以及判断是否半张票据。将宽度信息，左右红外的最大值最小值，半张标志存入钞票特征信息中。

定时器 2 的配置在 PCA 中断，定时 140 微秒，当 CCF1=1 即处理码盘中断时启动。

monitor 为定时器 2 中断服务程序，监视过钞情况，采集过钞原始数据，采集红外信号。定时器 2 在 PCA 中断服务程序种启动。定时器 2 在 PCA 的控制下监视过钞情况并采集红外信息。

中断服务程序中由左右计数管通过采集红外信息监视过钞。左右计数管的红外采集各有 8 种状态。第 0、1、2 三种状态监视钞票是否确实进入了计数管下，第 4、5、6 三种状态监视钞票是否确实出了计数管。第 3 种状态为采集红外信息状态。左右计数管同步于第 7 种状态，决定推出中断服务程序。

0 状态时左右计数管都判是否有钞票进入了左右计数管，有钞票进入了计数管下：记下左右入点的码盘计数值，先进入（通过状态的比较）的那一侧的起点为 0，进入 1 状态进行证实，若确实有钞进入则进入 2 状态再确认一次，否则回到状态 0 等钞进入计数管下。

在状态 2 若再磁确认计数管下有钞，则认为钞票已经进入了计数管下，可以

开始采集红外信息和计算所采集到红外值的最大值和最小值了，并且发信号给任务 3，表示任务 3 可以开始采集荧光信息了。若在状态 2 中不能判断计数管下有钞则回到状态 0 等钞进入计数管下。

在状态 3，采集红外信息并计算最大值和最小值，当用于采集红外信息和荧光信息的 48 个码盘时间用完了或由于紧急停机时有钞在左右计数管下时进入后续状态 4、5、6 检查钞票是否出了左右计数管。

在状态 4，如果判断钞票离开了计数管，则计下当前码盘值作为离开点，同样钞票离开也必须经过连续三次证实才认为钞票确实离开了计数管下。在 5、6 状态下每次不能证实钞票离开时回到状态 4 进行监测。在 6 状态时，如果另一侧的状态还没有到达状态 3 认为是半张，置标志位注明此张票据为半张，同时强置另一侧状态转入 7 状态。如果连续三次证实钞票已经离开则认为钞票确实离开，状态转入 7 状态。左右计数管同步于状态 7 时才认为整张钞票离开，此时为下一张钞票信息的存放做好初始化，并通知任务 0 表示钞票信息采集完成，可以开始对钞票的信息进行处理了。

程序清单 4.8

```
void OSTick0ISR()   interrupt 3
{
    TL2 = TIMER2_L;
    TH2 = TIMER2_H;

    OSIntEnter();
    Time2Tick ();      //因为这是与操作系统无关的函数，所以不以 OS 开头
    OSIntExit();
}
```

七、PCA 中断（码盘、左、中、右磁中断）

可编程计数阵列(Programmable Counter Array)PCA 是一个 16 位定时器，它有 5 个定时标志寄存器（CCAPnH+CCAPnL，n=0，1，2，3，4），把 PCA 划分成 5 个定时模块，这 5 个模块分别有一个单片机引脚与之对应。与标准的定时器/计数器相比，PCA 对 CPU 的干扰小，定时更精确，减少了软件设计。

1. 简单的工作原理

PCA 中断与定时器 2 中断紧密结合在一起，在 PCA 中断服务程序中重置定时器 2 并启动它。PCA 本身只设置一次，即在开机机器初始化时。PCA 中断由外部信号引发。当捕获到码盘信号跳变，捕获到左磁信号，捕获到右磁信号，捕获到中磁信号时会产生 PCA 中断，进入 PCA 的中断服务处理程序。在系统的三个中断中，PCA 的中断优先级次于定时器 0 中断。

PCA 中使用 1，2，3，4 路通道捕获跳变信号。均设置为捕获到正跳变信号时触发中断。在 PCA 中断服务程序中，计数发生的由码盘引起的 PCA 中断的次数，作为 monitor 中监测钞票进出计数管的入点与出点参数，也作为磁脉冲宽度的量度标准。

PCA 中，计数码盘引起的 PCA 中断次数作为钞票入点与出点参数，为钞票宽度和磁脉冲宽度的量度标准；计数左磁个数，右磁个数，中磁个数，并计算左磁脉冲最大宽度和右磁脉冲最大宽度，记录中磁脉冲宽度信息，将这些信息记入钞票的特征信息缓冲区内。

2、程序设计实现：

PCA 中断服务程序，控制过钞监视状态，控制定时器 2 中断产生从而监视过钞情况。采集过钞原始数据，主要是有关磁的信息，包括左磁个数，右磁个数，中磁个数，左磁脉冲最大宽度，右磁最大宽度，中磁脉冲宽度信息。

PCA 中使用了 PCA 的四个通道分别捕获码盘信号正跳变（CCF1=1），左磁信号正跳变（CCF2=1），右磁信号正跳变（CCF3=1），中磁信号正跳变（CCF4=1），这四个跳变都可触发 PCA 中断服务程序。

PCA 中断服务程序控制过钞时红外信息的采集，并记录过钞时的磁信息。码盘信号，左磁信号，右磁信号，安全线中磁信号正跳变触发该服务中断服务程序。当中断服务程序是右码盘信号触发时，启动定时器 2，监视过钞情况，控制在定时器 2 中断服务程序中采集红外信息，控制任务 3 采集荧光信息，并为采集钞票入计数管和出计数管提供计数依据。当中断服务程序是由左磁信号，右磁信号，安全线中磁信号触发时则记录钞票的磁信息：中磁个数，中磁宽度，左右磁个数，左右磁最大宽度。不管是由哪种跳变触发该服务程序，由于有 CCF1,CCF2,CCF3,CCF4 来记录是否跳变发生，该程序都会检查一遍是否有码盘信号正跳变，左磁信号正跳变，右磁信号正跳变，安全线中磁信号正跳变已经发生（通过依次检查 CCF1,CCF2,CCF3,CCF4 是否置 1），检查到有跳变产生就进行相应的处理，或计数，或计算宽度，或启动定时器 2 等，

3、PCA 中断配置与应用

PCA 中断配置如程序清单 4.8 所示，T89C51SC2 芯片支持 5 路信号捕捉，本项目中用到了其中的 4 路，都是信号下降沿触发的。PCA 中断处理程序如程序清单 4.9 所示。

程序清单 4.8

```
CMOD=0X00;              //单片机空闲模式 PCA 工作
CCAPM0=0X11;            //码盘信号，下降沿触发
CCAPM1=0X11;            //安全线中磁信号，下降沿触发
```

```
CCAPM2=0X11;                    //左磁信号，下降沿触发
CCAPM3=0X11;                    //右磁信号，下降沿触发
CCAPM4=0;                       //不使用
```

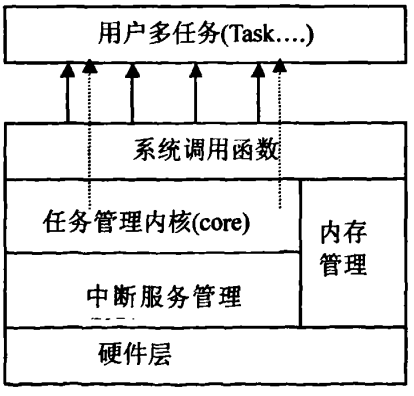
程序清单 4.9

```
void    pca(void) interrupt 6
{
    OS_INT_ENTER();
    EC=0;                               //PCA 中断关闭

    PcaInt();

    EC=1;                               //PCA 中断使能
    OSIntExit();
}
```

4.3.2 操作系统层

为了让μC/OS-Ⅱ能更好的应用，我们必需根据实际需求进行配置、裁减。在修改优先就绪表和去掉消息队列模块之后，系统占用的 ROM 空间从 9K 下降到了 7K。

结构模型如下所示：



一、时钟设置

一般来说，慢速处理器或者低电源系统可以选择较小的时钟频率，反之，强大的机器或者有多媒体或紧急实时应用需求的机器将受益于较高的钟频率。时钟节拍的频率越高，系统对时间的延时的控制也越精确，实时性也就越好。本系统中，由于不少地方需要精确控制时间，如马达控制、按键防抖动等，特别是因为

数码管 LCD 的数量比较多，要保证不闪烁，需比较高的时钟频率，因此把时钟频率设为 1000Hz，即每 ms 为一个时钟节拍。较高时钟频率一般也会导致较高利用 CPU，也即比较耗电。但本系统使用交流供电，不依靠电池，所以不造成影响。本系统使用定时器 0 作为系统时钟，外接晶振频率为 20MHz，在定时器模式下，计数器由单片机脉冲经 12 分频后计数。因此，定时器定时时间 T 的计算公式为：

$$T=\frac{(TM-TC)\times 12}{F_{osc}} \qquad [4.3]$$

$$TC=TM-\frac{F_{osc}\times T}{12} \qquad [4.4]$$

式中 TM 为计数器从初值开始作加 1 计数到计满为全 1 所需要的时间，TM 为模值，和定时器的工作方式有关；f_{osc}是单片机晶体振荡器的频率，TC 为定时器的定时初值。当定时器 0 设置为工作方式 1（16 位方式）时，$TM=2^{16}=65536$，$f_{osc}=20\times10^{6}$，$T=1\times10^{-3}$，因此，定时器的初始值 TC 为：

$$TC=65536-\frac{20\times10^{6}\times1\times10^{-3}}{12}=63869=0XF97D \qquad [4.5]$$

二、最大任务数

我们在μC/OS-II 定义了一个常量 OS_MAX_TASKS，取值范围为 1～16，最大任务数就是实际用户的任务数，此值按实际任务数量设置，在本项目中设为 5。

三、INT16U 型变量的存储方法

LOW_BYTE 和 HIGH_BYTE(这两个宏是原μC/OS-II 中没有定义的)表示 INT16U 型变量在特定 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 C），则 HIGH_BYTE 为 0，LOW_BYTE 为 1。

四、内存管理、消息队列

在小型系统中，为避免过多的开销，内存管理可以省略，将 OS_CFG.H 文件中将开关量 OS_MEM_EN 以及相关宏设置为 0。这样μC/OS-II 在启动时就不会对内存管理器进行初始化。

本项目中只用到消息邮箱，不需要消息队列，同样设置为 0，这样可以节省系统占用的代码空间

程序清单 4.10

```
#define TIMER0_L    0x7D
#define TIMER0_H    0xF9


#define OS_MAX_TASKS          5


#define LOW_BYTE              1
#define HIGH_BYTE             0
```

```
#define OS_SEM_EN                0
#define OS_SEM_ACCEPT_EN         0
#define OS_SEM_DEL_EN            0
#define OS_SEM_QUERY_EN          0

#define OS_Q_EN                  0
#define OS_Q_ACCEPT_EN           0
#define OS_Q_DEL_EN              0
#define OS_Q_FLUSH_EN            0
#define OS_Q_POST_EN             0
#define OS_Q_POST_FRONT_EN       0
#define OS_Q_POST_OPT_EN         0
#define OS_Q_QUERY_EN            0
```

4.3.3 应用层

应用层的开发主要包括完善各中断程序中应用处理函数和各应用模块的设计。

一、中断处理

本系统有 4 个中断，除串口中断程序相对独立外，其它 3 个驱动程序与验钞机的应用密切相关。

1、OSTimeTick()

要保证验钞机的显示稳定，不闪烁，就必须定时刷新，所以在这个函数中加入显示处理操作。

2、PcaInt()

在此函数中进行码盘、安全线信号、左磁信号、右磁信号处理。

程序清单 4.11

```
void PcaInt(void)
{
    if(CCF0==1)                    //码盘
    {
        CCF0=0;
        // 码盘计数及处理
    }
```

```
    if(CCF1==1)                    //安全线信号
    {
        CCF1=1;
        //安全线信号计数及处理
    }

    if(CCF2==1)                    //左磁信号
    {
        CCF2=0;
        //左磁信号计数及处理
    }

    if(CCF3==1)                    //右磁信号
    {
        CCF3=0;
        //右磁信号计数及处理
    }
}
```

3、TimeTick2()

马达启动时激活定时器 2，用来判断钞票是否已到达计数管，并采集红外、紫外信息(为简单起见，未在程序中加入宽度处理和纠错代码)。

程序清单 4.12

```
void TimeTick2(void)
{
    lstate 为左计数管过钞状态，初始为 0
    rstate 为右计数管过钞状态，初始为 0

    if(如果左计数管下有钞)
    {
        if(lstate==0)
        {
            lstate++;
        }
```

```
        if(lstate==1)        //左边钞票进入
        {
            if(rstate==1)  //右边钞票进入
            {
                lstate++;
            }
        }
        if(lstate==2)        //两边有钞
        {
            采集紫外信号;
            保持红外信号;
        }
    }
    else
    {
        if(lstate==2)        //左边钞票离开
        {
            lstate==3;
        }
        if(rstate==3)              //右边钞票离开
        {
            lstate=rstate=0;    //钞票过完，回复初始状态
            激活钞票处理任务;
        }
    }

    if(如果右计数管下有钞)
    {
        if(rstate==0)
        {
            rstate++;
        }
        if(rstate==1)        //右边钞票进入
        {
```

```
            if(lstate==1)  //左边钞票进入
            {
                rstate++;
            }
        }
        if(rstate==2)       //两边有钞
        {
            采集紫外信号;
            保持红外信号;
        }
    }
    else
    {
        if(rstate==2)       //右边钞票离开
        {
            rstate==3;
        }
        if(lstate==3)           //左边钞票离开
        {
            lstate=rstate=0;    //钞票过完，回复初始状态
            激活钞票处理任务;
        }
    }
}
```

二、多任务设计

本项目所移植的μC/OS-Ⅱ，如第二章所述，是一个多任务实时内核，允许建立多达63个用户任务，根据程序建立和运行的情况决定什么时候从一个任务切换到另一个任务[10]。一个应用任务看起来和其他C的函数一样，有函数返回类型，有形式参数变量，但是任务是不会返回的。故返回参数必须定义为void。

根据智能验钞系统的功能特点，本系统设置了5个用户任务，分别是启停监测、按键监控、信息处理、数据输出和参数设置。这5个任务分别是：

1、启停监测

启停监测通过入钞口，接钞口是否有钞，来决定是否启动马达。

启停检测设有5个状态：

状态0为常规监测（静止）状态，即还未过钞，或者过钞结束并已取走钞

票。当检测到启停有钞，且防漏无钞时开始启动马达，并转至状态 2

状态 1 为出错停机后的监控状态（静止），当防漏无钞时，或按任意键启动马达，并转至状态 2

状态 2 为马达运行时候的监控状态当钞票过完后停机，并根据是否出错、防漏是否有钞转至相应状态

状态 3 为防漏有钞票时候的监控状态（静止），当防漏无钞时，或启停有钞时启动马达，并转至状态 2

状态 4 为预置张数达到而停机时的监控状态（静止），当防漏无钞时启动马达，并转至状态 2

2、按键监控

主要用于设置系统进入不同的功能模式，典型的有正常模式、预置模式、参数设置模式、计数模式。还可以显示当前钞票宽度、红外、紫外和磁信号等数据，以便调试。

组合键：F+键：　设置功能选项开关；F-键：设置反转时间；F+-键：进入调试状态；←键：显示左右磁单键；F 键：设置工作状态；+、-键：在预置状态加减预置数。

3、信息处理

在每一张钞票全部通过计数管后，系统激活此任务，用于判断这张钞票的宽度是否超长、超短，是否残缺、有破洞等，正常的话再判断真伪、面额和版别。

4、数据输出

当钞票数据处理完成后，将钞票的面额、版别及检验张数输出到显示屏上，如果出错则停机，鸣蜂鸣器报警，并显示出错原因。

5、参数设置。

由于验钞的机械、电器性能、使用环境的差异，使其与标准机在标准状态有所不同，为提高系统的灵活性和准确性，我们可以把一些相关参数写入 eeprom 中，用户可以根据用户手册自行修改。

4.3.4 嵌入式系统的调试

调试是开发过程中必不可少的环节，通用的桌面操作系统与嵌入式操作系统在调试环境上存在明显的差别。前者，调试器与被调试的程序往往是运行在同一台机器、相同的操作系统上的两个进程，调试器进程通过操作系统专门提供的调用接口控制、访问被调试进程。后者（又称为远程调试），被调试的程序则运行于基于特定硬件平台的嵌入式操作系统（目标操作系统）。这就带来以下问题：

调试器与被调试程序如何通信，被调试程序产生异常如何及时通知调试器，调试器如何控制、访问被调试程序，调试器如何识别有关被调试程序的多任务信息并控制某一特定任务，调试器如何处理某些与目标硬件平台相关的信息（如目标平台的寄存器信息、机器代码的反汇编等）[25]。调试方案有：

1.模拟调试：直接在主机上进行调试，使用软件模拟目标运行环境，主要进行语法和逻辑上的调试。

2.软件调试：主机和目标板通过某种接口（通常是串口）连接，主机上提供调试界面，待调试软件下载到目标板上运行。这种方式的先决条件是要在主机和目标板之间建立通信联系。

3.BDM/JTAG 调试：这种方式调试除了主机和目标板之外，还需要一个额外的调试装置，该装置与目标板通过 BDM/JTAG 等调试接口相连，与主机通过串口、并口、网口或 USB 相连。待调试的软件通过该调试装置下载到目标板上运行。

4.全仿真调试：仿真器完全或部分取代目标板上的部件（例如机械部分或 MCU），因而目标系统对开发者来说完全是透明的、可控的。由于仿真器自成体系，调试时既可以连接也可以不连接目标板。

此外，还可利用示波器调试，通过示波器，我们能直观的看到传感器、芯片引脚信号的变化，并保存和输出。可以高效查看、浏览和分析波形数据。

4.4 本章小结

市面上验钞机生产厂家普遍使用的前后台系统，汇编语言编写程序，实时性较差。我们通过移植μC/OS-Ⅱ操作系统，使用 C 语言开发，简化了程序的设计过程，降低了开发难度，提高了开发效率、可维护性和可移植性。

本章完成了对智能验钞机这个商业化产品的设计和研发。商业化产品出于节约成本的角度采用了 8 位芯片的 T89C51AC2 单片机。首先对验钞机实际的应用进行了需求分析，设计了钞票的检伪流程。运用了宽度检测、磁性检测和荧光检测三种检伪手段，并对这三种检伪手段的实现设计了具体的方法。根据实际应用和要求划分在系统中的任务，接下来进行系统的实现，编写接口/驱动层程序，完善优化操作系统层配置，系统构架建立好以后，在每个任务中完成需要的应用部分，使整个系统的软、硬件成为一个整体，达到商业化产品的要求。

第五章 总结与展望

本论文系统地概述了嵌入式系统的发展过程、分类、应用及其发展方向。在深入研究了μC/OS-II 内核工作原理的基础上，详细论述了μC/OS-II 在 C51 平台上的移植，移植过程中各种关键问题的分析与解决，特别是针对 C51 内存小的特点，对该操作系统的内核，进行了一系列地修改和优化，以满足实际应用的需要，从而得到一个灵活、高效、可配置的面向 C51 的多任务实时操作系统，并将其应用到实际项目上。

5.1 项目完成结果

本文以 T89C51AC2 为嵌入式实时系统微处理器，以μC/OS-II 为嵌入式实时操作系统内核，以嵌入式系统设计原理和工业控制实际应用为核心，从理论上和技术方法上开展了一系列研究实现了一个完整的嵌入式应用系统。

主要工作有:

1.全面系统地概述了嵌入式系统的发展过程和分类，及其在各个领域内的应用，以及嵌入式系统的发展方向。着重剖析了嵌入式实时操作系统μC/OS-II 实时内核的工作原理。在这些知识的基础上，依据本项目经济成本考虑，没有去购买昂贵的商业 RTOS（实时操作系统），而是选择了源代码公开、体积小且可裁剪移植性好的μC/OS-II，选择 C51 微处理器为应用平台，提出了在 T89C51AC2 硬件平台上移植嵌入式实时操作系统μC/OS-Ⅱ。

2. 详细的论述了μC/OS-II 内核移植的具体过程，及其在具体应用中配置与优化。根据对μC/OS-II 内核模块的分析，移植的过程主要是修改与处理器相关的文件的代码，集中体现在三个文件的重新编写上：一个头文件 OS_CPU.H、一个 C 代码文件 OS_CPU_C.C 和一个汇编文件 OS_CPU_A.ASM 文件。另外还针对 C51 的小内存所做的优化，对可重入问题的进行了分析与解决，基本达到应用的需要。

3. 以μC/OS-II 实时内核为基础，把它应用于智能验钞机系统。设计运用了宽度检测、荧光检测和磁性检测三大检伪手段。根据实际需要，设计了五个任务(启停监测、按键监控、信息处理、数据输出和参数设置)和三个中断(定时器 0 中断、定时器 2 中断和 PCA 中断)。基于μC/OS-II 内核的程序，进行多任务的分配和调用，构建智能验钞机的软件构架，并在多任务分配的基础上，在各个任务中调用底层的驱动模块的 API 函数，结合外部中断服务程序，具体的完成项目

中软件应用的部分，形成了一个具有实际意义的嵌入式系统应用软件。可剥夺性的实时内核，对验钞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。且移植过程使用C语言开发，简化了程序的设计过程，降低了开发难度，提高了开发效率、可维护性和可移植性。

二、本项目特色：

（一）使用了嵌入式实时操作系统μC/OS-II

μC/OS-II 可以管理 16 个任务，每个任务优先级不同；支持中断嵌套；是为小 RAM 系统设计，所需 RAM 极小。

实时操作系统使得实时应用程序的设计和扩展变得容易。通过应用程序把设计分割为若干独立的任务，μC/OS-II 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、处理延时和超时等，使得资源得到更好的利用。

（二）8位微处理器芯片的运用

虽然目前嵌入式系统的研究和应用如火如荼，但是重心大都放在基于16位、32位甚至64处理器的产品上，如手机、PDA、MP3、MP4等。由于资源有限，更重要的是为观念所限，在嵌入式应用中占有相当比重的基于8位处理器的产品开发中，鲜有引入嵌入式操作系统，而且大多使用汇编语言编写。另一方面，不少从事单片机应用的人，在谈到“嵌入式系统”领域时，往往理解成基于32位嵌入式处理器，从事网络、通信、多媒体等的应用。即使接触过8位处理器的嵌入式操作系统，一般也只是当作学习操作系统原理的途径，很少有应用到实际产品当中。

总之，本文不但移植、优化了μC/OS-II操作系统，更重要的是将其应用到实际项目当中。应用程序的开发过程中，全部使用C语言，大幅提高了开发的效率，降低了开发的难度，更兼有系统结构清晰，可读性好、可移植性高等优点，在总体功能上，程序的执行效率没有明显的降低，只需多占用一些系统资源。因为单个模块执行的效率高，不等于整个程序的执行效率高。一个很简单的例子，在不引入操作系统的情况下，如果程序需要等待一段时间，一般用程序延时或定时器延时，无论何种情况，CPU都不再处理其他工作，效率很低。而使用操作系统，等待的时候CPU可以处理其他工作，效率得到提高。

5.2 开发的难点

1. 在C/OS-II的移植过程中，为了节省系统资源，本文对原系统的任务结构和调度算法进行了修改。任务调度是操作系统的最重要的核心之一(另一个是

内存管理)，这需要对操作系统的内核有深入的理解。

2. 在内核的 OS_CPU_A.ASM 的移植中，要写 3 个汇编语言函数。这需要深入地学习 C51 汇编语言。

3. 在项目过程中，常常在时间和空间、效率和可读性之间权衡。使用函数能减少使用空间，使用宏能加快执行速度；使用汇编能提高执行效率，用 C 语言可读性强（最终还是全部用 C 语言，改善整体结构——如系统时序的设计，效果会更好)。

4. 应用程序中时序设计。本验钞机的最高点钞速度达 1000 张/分钟，正常情况下，平均每张钞票的数据采集时间加上处理时间只有 60ms(在极端情况下，时间会更少——如有时钞票未能完全捻开，使得相邻两张钞票的过钞间隔大为减小)，而要采集、处理的数据多：红外、紫外、安全线信号、磁性油墨信号，对于一张 100 元钞票的数据采集次数超过 100，加之定时刷新 LED 需要花费不少时间，这对于 8 位处理器是一个极大的挑战。

5.2 存在问题

虽然本项目已经过严格的测试，进入到实际应用阶段，但是由于时间有限，还没有对系统的实时性从理论上进行分析。影响嵌入式操作系统实时性的因素操作系统有很多，主要有：

1. 常用系统调用平均运行时间。即系统调用效率，是指内核执行常用的系统调用所需的平均时间。

2. 任务切换时间。是指事件引发切换后，从当前任务停止运行、保存运行状态（CPU 寄存器内容），到装入下一个将要运行的任务状态、开始运行的时间间隔。

3. 任务抢占时间，是高优先级的任务从正在运行的低优先级任务中获得系统控制权所消耗的时间。

4. 中断响应时间。是指从中断发生到开始执行用户的中断服务程序代码来处理该中断的时间。

系统调用效率可以通过编写统计任务进行分析，在《μC/OS-II—源码公开的实时嵌入式操作系统》[10] 中已有类似例子可以借用。

我们可以借助 Keil C 强大的调试功能，来分析任务切换时间、抢占时间和中断响应时间。

指令周期是执行一条指令所需要的时间，一般由若干个机器周期组成。MCS-51 有固定的机器周期，一个机器周期共包含 12 个振荡脉冲，即机器周期就

是振荡脉冲的 12 分频[26]。显然，如果使用 12MHz 的时钟频率，一个机器周期就是 1us，而如本项目中使用 20MHz 的时钟频率，一个机器周期就是 0.6us。

在调试状态下，切换到“Disassembly windows”汇编窗口，观察任务切换、抢占和中断响应时所执行的指令条数和每条指令的指令周期，得到总指令周期数，再乘以每个周期 0.6us 即可得到相应的总时间。

但是，上面只是静态情况下的理论数值。实际情况下，由于中断和中断嵌套的存在，以及在此情况下会出现优先级反转[27]的现象，使得切换时间、抢占时间和中断响应时间要大于理论值，只能通过实验来解决了。

此外，单片机的存储空间限制了程序的 C 语言化。操作系统本身占用了 9K 的程序空间和 2K 的数据空间，这对于仅有 64K 空间的单片机来说会限制后续程序功能的扩展。解决办法是继续优化 μC/OS-II 系统，将无用的功能模块去掉。

5.3 嵌入式系统展望

一个好的嵌入式系统是以应用为中心，以计算机技术为基础，软件和硬件可剪裁，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[28]。小型嵌入式系统通常分为实时和非实时两类，实时系统是对某些特殊任务的相应处理时间有很高的要求，不但要求任务能完成，并且要求在时限到来之前提前完成，而非实时在这方面就相对低一些，一般只要求任务能够完成即可。现代工业的发展，对于时间要求越来越严格，因此实时嵌入式系统成为业界的热点。

“基于 C51 的 μC/OS-II 移植和应用”是对嵌入式实时操作系统应用的一个拓展，在一定的硬件平台条件下，采用一些源码公开的嵌入式操作系统，根据芯片的特点，对其内核作一定的修改，定制出一套特定用处的操作系统。这样的系统通常比较小型，灵活易修改，对应用的针对性强，可以增加嵌入式系统的适用性。本课题通过移植 μC/OS-II 操作系统，使得整个程序结构化，易读易懂，而且大大缩短了应用程序的开发周期，为系统的升级和扩展打下了良好的基础。

参 考 文 献

[1] RAJ KAMAL 著，陈曙晖译.嵌入式系统—体系结构、编程和设计.北京：清华大学出版社，2005.

[2] 唐寅.实时操作系统应用开发指南.北京：中国电力出版社，2002.7.

[3] 桑楠.嵌入式系统原理及应用开发技术.北京:北京航空航天大学出版社，2002年.

[4] Steven F.Barrett.Embedded Systems[M]，北京：电子工业出版社，2006.

[5] 李伯成.单片机及嵌入式系统[M]，北京：清华大学出版社，2005[4]

[6] 陈晗斐.实时操作系统的若干关键问题研究，浙江大学博士论文 2004.7

[7] 朱瑾.基于 Linux 嵌入式操作系统实时性的研究.硕士学位论文沈阳工业大学.2005.3

[8] 刘丙成. μC/OS-II 内核分析及其平台的构建：硕士学位论文.呼和浩特:内蒙古工业大学，2005.

[9] VxWorks programmers guide, WindRiver System Inc.

[10] JEAN J.LABROSSE 著，邵贝贝译. μC/OS-II—源码公开的实时嵌入式操作系统.北京:中国电力出版社，2001.

[11] 杨屹.在 51 单片机上固化 uCOS51 的说明. http://www.hjhj.com.

[12] 田泽.嵌入式系统开发与应用.北京:北京航空航天大学出版社，2005.1.5.

[13] Keil Elektronik GmbH.and Keil Software,Inc. Cx51 Compiler Optimizing C Compiler and Library Reference for Classic and Extended 8051 Microcontrollers User's Guide[M].

[14] 黄亮亮，朱欣华.基于实时多任务操作系统 μC/OS-II 的 C8051F 系列单片机应用系统开发测控技术 2005 年第 24 卷第 9 期

[15] 李江常，葆林. 嵌入式操作系统设计中的若干问题计算机工程 2000.6 第 20 卷第 6 期:89

[16] 刘从新，曾维鲁. 51 单片机实时操作系统的构建三峡大学学报 2003.10 第 25 卷第 5 期:446

[17] 江平新，容太平. μC/OS-II 在 cso51F 上的移植.单片机与嵌入式系统应用，2003 年 9 期.

[18] 李玉峰，倪虹霞编著.MCS-51 系列单片机原理与接口技术.北京：人民邮电出版社，2004.5.

[19] 胡大可，李培弘，方路平编著.基于单片机 8051 的嵌入式开发指南.北京：电子工业出版社，2003.1.

[20] 何立民著.单片机应用系统设计.北京：北京航空航天大学出版社，1990.2

[21] 谢维成，杨加国等著.单片机原理与应用及 C51 程序设计.北京：清华大学出版社，2006.8.

[22] 成毅，庹先国，高嵩. PS/2 键盘在 51 单片机系统中的应用.中国测试技术.2004 年，05 期.

[23] RAJ KAMAL 著，陈曙晖译.嵌入式系统—体系结构、编程和设计.北京：清华大学出版社，2005.

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

[25] 熊竞．如何对嵌入式系统进行调试，http://www.spasvo.com/html/ceshi/20080624-176.html

[26] 杨全胜. 现代微机原理与接口技术. [M]，北京：电子工业出版社，2002

[27] 赵奇，索晓冉. 实时系统优先级反转研究，计算机应用研究，2008 年第 25 卷第 6 期.

[28] 罗蕾.嵌入式实时操作系统及应用[M]，北京：北京航空航天大学出版社，2005.

[29] 廖志文.防伪印刷最前沿——浅谈人民币防伪印刷技术，印刷世界，2007 年第 01 期.

[30] 验钞机的辨伪原理 http://www.cnnsr.com.cn/jtym/wzxx/cpzx/20051020/20051020162430143 85.shtml

[31] 吴莲贵，李维. μC/OS-II 在 C51 系列单片机上的移植，现代计算机 2008 年第三期（总第二七九期）

[32] 任哲.嵌入式实时操作系统 μC/OS-II 原理及应用.北京：北京航空航天大学出版社，2006

[33] 董余红，阎俊武.μC/OS-II 在 C51 系列单片机上的移植，导弹与航天运载技术 2004 年第 6 期（总第 273 期）

[34] 周文.μC/OS-II 在 AT89S51 单片机上的移植 计算机与现代化 2008 年第 4 期（总第 152 期）

[35] 魏洪兴，等.嵌入式系统设计与实例开发试验教程[M].北京:清华大学出版社，2005. 190- 199.

[36] [美]鲍尔.嵌入式微处理器系统设计实例[M]苏建平，等译北京:电子工业出版社，2004

[37] 田志鑫，张雷，赵明扬. 在 51 单片机上移植 μC/OS-II 关键问题的解决.《微计算机信息》(嵌入式与 SOC)2007 年第 23 卷第 12-2 期

[38] 索双富，孙晋厚，肖丽英. 点钞机鉴伪技术发展趋势 机械设计与制造 2007年12月

[39] 刘连浩，罗安，王加阳，等智能型点钞机的研制[J] 机电一体化，2003, 9(1): 11-13

[40] KEIL C51 使用详解 PDF. http://www.mcul6.com/microprocessor/Keil- C51 一 PDF.htm

[41] 谭浩强.C 程序设计[M].北京:清华大学出版社,2002

[42] 嵌入式开发论坛 http://www.cevx.com/bbs/index.php

[43] Qian Ji, Qian Dongping, Zhang Mengjie. A digit recognition system for paper currency identification based on virtual instruments [J]. Information and Automation,2006:46 一 49.

[44] Lu Z,Hein J.Control-theoretic dynamic frequency and voltage scaling for multimedia workloads[C].Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems.New York,NY,USA:ACM Press,2002:156-163

[45] Xia Feng,Dai Xiaohua.Feedback scheduling of real-time control tasks in power-aware embedded systems[C].Proceeding of the Second International Conference on Embedded Software and Systems (ICESS'OS). Washington, DC, USA: IEEE, 2005:513-519

[46] eclipsefocus: Motorola joins eclisp, propses Tml project, embedded computing design, 2006.10

[47] Texas Instruments Incorporated.TMS320VC5402 Peripherals Reference Guide [Z],Texeas:Texeas Instruments Incorporated,2001

[48] Fumiaki Takeda, Sigeru Omatu,High speed paper currency recognition by neural networks,IEEE Trans on Neural Network,1995,6(1):73-77

[49] A Frosini,M Gori,Pprimi.A neural network based model for paper currency recognition and verification.IEEE Trans on Neural Networks,1996,7(6):1482-1490

[50] Specification/data Information,May 1996

[51] T89C51AC2.Atmel.http://www.atmel.com

[52] T89C51AC2 Errata.Atmel.http://www.atmel.com

[53] T89C51AC2 UART Bootloader.Atmel.http://www.atmel.com

[54] C Flash Drivers for the T89C51AC2 for Keil Compilers rev 1.2.0.Atmel. http://www.atmel.com

[55] 陈涛，周金运，彭孝东. 红外光电管及其在验钞机中的应用. 现代电子技术，2005，18：6~7

[56] 索双富，孙晋厚，肖丽英. 点钞机鉴伪技术发展趋势. 机械设计与制造，2007，12：199~201

致 谢

完成论文的时光忙碌艰辛而又充实，尤其对我这样已走入婚姻家庭，要做好工作还要照顾家庭、孩子、老人，并且要兼顾学习的女人，更是不易。终于完成了！ 三年前我被录取为中南大学信息与科学工程学院的工程硕士时，刚刚本科毕业工作一年。三年里我收获了爱情、家庭，现在已为人母，有了两个可爱的双胞胎女儿。

衷心的感谢我的学位论文导师陈志刚教授！一直就觉得您接纳我作为您的学生我已足够幸运，您在繁重劳累的工作之余，耐心地阅读我艰涩、冗长的论文，提出了宝贵的修改意见，还勉励我多积累和思考。可惜我愚钝不够勤勉，离老师的期待太远！我还要特别感谢曾志文老师热情关怀和悉心指导。在我撰写论文的过程中，曾老师倾注了大量的心血和汗水，仔仔细细的阅读我的论文，在论文的研究方法以及成文定稿方面，我都得到了曾老师悉心细致的教诲和无私的帮助，在此表示真诚地感谢。

感谢我的父母，你们满怀期待辛苦养育了我，当年辛苦拉扯了我，今天又在帮我拉扯我的孩子，女儿觉得很对不起你，也很懂得感恩。感谢我的爱人戴军，请记得我们相亲相爱一家人就是世间最美好的生活！感谢我的两个女儿，你们是我最大的牵挂！

攻读学位期间主要的研究成果

1．作者攻读硕士学位期间已经发表的论文(1 篇)

[1] 娄小平，陈志刚. 基于 VC 的字模图文系统的设计与实现. 电脑与信息技术，2009，1

[2]娄小平，戴军. 集群通信中一种语音加密方法的研究与实现. 湖南文理学院学报自然科学版 2008.4