同济大学电子与信息工程学院

硕士学位论文

基于嵌入式实时操作系统的任务调度研究与应用

姓名：毛磊

申请学位级别：硕士

专业：控制理论与控制工程

指导教师：萧蕴诗

20070301

摘要

嵌入式系统有一个共同的特性，即对系统的响应时间有严格要求。随着实时嵌入式系统结构的日益复杂，运行环境的不确定因素增多，操作系统中任务调度受到广泛关注。

本文首先介绍 RTOS 中经典的调度策略，分析其各种任务调度算法的局限性。然后从实时系统的不确定性、不可预测性角度入手，针对实时系统中硬实时任务与软实时任务并存情况，分析混合任务并介绍混合任务调度算法，提出了混合任务调度模型框架。在对任务反馈调度研究中，根据 Chenyang Lu 和 John A.Stankovic 提出的反馈控制实时调度算法，对 PI 控制器详细分析并介绍利用率、丢失率、利用率丢失率反馈调度算法，总结了各调度算法优劣。对于控制与调度协同设计问题，提出以 Qoc 作为评价基础，给出基于 Qoc 的调度策略，并构建基于实时控制与调度协同设计模型，使用弹簧算法灵活处理扰动到来对被控系统的影响。最后在反馈混合任务调度研究中，以反馈控制混合任务调度框架为基础，可调软硬实时任务负载带宽为出发点，准入与 Qos 调节器为软实时任务负载调节手段，提出了反馈调度的水箱模型以及控制器的控制算法，仿真结果验证了该算法的可行性。

通过研究 μC/OS-II 的任务管理、内核调度方面的内容，移植 μC/OS-II 于 TMS320LF2407A 硬件平台上。在 μC/OS-II 实时内核调度的可扩展性研究基础上，修改内核，设计实时任务和软件结构，并引入基于反馈控制混合任务调度算法。实践验证该调度算法有效提高系统的鲁棒性。

关键词：任务调度，混合实时任务，反馈调度，μC/OS-II，Qoc

ABSTRACT

The embedded system has such a common characteristic that they have strictly time constraint. As the embedded system is getting more complicated and usually in uncertain circumstances , task schedule is attracting wide spread attention.

Firstly, the thesis introduces the classical scheduling policy in RTOS and analyzes the limitation of all kinds task scheduling algorithms. Secondly , in the viewpoint of real-time system's uncertainty and unpredictability ,mixed real-time task is analyzed and its scheduling algorithm and frame and model is proposed .In the study of task feedback schedule, detail analysis in PI controller architecture and FC-U,FC-M and FC-UM is given on the basis of Chenyang Lu and John A.Stankovic's feedback control real-time scheduling theory. And also the algorithms' metrics and disadvantages are concluded. When it comes to the control and schedule co-design problem , model of real-time control and scheduling co-design and Qoc scheduling policy are proposed by using flexible spring algorithm dealing with disturbance arrival with the evaluation foundation of Qoc.Finally, in the field of feedback control and mixed real-time scheduling ,the tank model based on feedback schedule and its controller algorithm are presented by tuning soft real-time tasks load with access and Qos regulator when guaranteeing hard real-time tasks' load bandwidth in the frame of feedback control mixed real-time task scheduling .The simulation results proves its feasibility.

The μC/OS-II system is ported on TMS320LF2407A hardware platform after studying the task manage and kernel portion of μC/OS-II. On the basis of its scheduling polity extensional study, its kernel is revised and tasks and software architecture is designed by porting feedback control and mixed task scheduling algorithm. The result validates its effectiveness in improving robustness of real-time system.

Key Words: task Scheduling, mixed real-time task , feedback scheduling ,μC/OS-II,

Qoc

学位论文版权使用授权书

本人完全了解同济大学关于收集、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：毛磊

2007年3月12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

同济大学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：王磊

2007 年 3 月 12 日

第 1 章 绪论

1.1 概述

1.1.1 嵌入式实时系统

在计算机技术和信息技术高速发展的今天，计算机和计算机技术大量地应用在我们的日常生活中，广泛应用的嵌入式计算机便是其中的一种。目前，从航天飞机到家用微波炉，嵌入式系统广泛应用于工业、交通、能源、通信、科研、医疗卫生、国防以及日常生活等领域，并发挥着极其重要的作用。

工业控制、舰船武器系统控制、航空航天等领域的多数嵌入式系统有一个共同的特性：对系统的响应时间有严格要求，这些系统也被称为嵌入式实时系统。

嵌入式实时系统一般包括硬件和软件两部分。硬件包括处理器 / 微处理器、存储器及外设器件和 I / O 端口、图形控制器等。软件部分包括操作系统软件（OS）（要求实时和多任务操作）和应用程序编程。应用程序控制着系统的运作和行为；而操作系统控制着应用程序编程与硬件的交互作用。

嵌入式系统的硬件部分核心是嵌入式微处理器。嵌入式微处理器一般就具备以下 4 个特点：

1）对实时多任务有很强的支持能力，能完成多任务并且有较短的中断响应时间，从而使内部的代码和实时内核的执行时间减少到最低限度。

2）具有功能很强的存储区保护功能。这是由于嵌入式系统的软件结构已模块化，而为了避免在软件模块之间出现错误的交叉作用，需要设计强大的存储区保护功能，同时也有利于软件诊断。

3）可扩展的处理器结构，以能最迅速地开展出满足应用的最高性能的嵌入式微处理器。

4）嵌入式微处理器必须功耗很低，尤其是用于便携式的无线及移动的计算和通信设备中靠电池供电的嵌入式系统更是如此，如需要功耗只有 mW 甚至 μW 级。

而嵌入式系统的软件核心是实时操作系统(RTOS)构成的。通常 RTOS 都采用基于优先级的调度方式，内核严格根据任务的优先级，将 CPU 资源分配给它们使用。目前已经有很多成熟的 RTOS(Real Time Operation System)，诸如 VxWorks, pSOS, PalmOS, OS-9, LynxOS, QNX, Nucleus, RT-Linux, μC/OS-II, eCOS, WINCE 等等。它们大体上可分为两种——商用型和免费型。商用型的实时操作系统功能稳定、可靠，有完善的技术支持和售后服务，但往往价格昂贵。免费型的实时操作系统在价格方面具有优势，目前主要有 Linux 和 μC/OS。如表 1.1 所示。

表 1.1 实时操作系统概况

操作系统	简介
VxWorks	是美国 WindRiver 公司于 1983 年设计开发的一种嵌入式实时操作系统，由于具有高性能的系统内核和友好的用户开发环境，在嵌入式实时操作系统领域牢牢占据着一席之地，值得一提的是，美国 JPL 实验室研制的著名“索杰纳”火星车采用的就是 VxWorks 操作系统[2][3]。它的突出特点是：高可靠性、高实时性和易裁剪性。它是目前嵌入式系统领域中使用最广泛、市场占有率最高的操作系统。
QNX	加拿大 QNX 公司的产品。QNX 是在 x86 体系上开发出来的，它是一个实时的、可扩充的、建立在微内核和完全地址空间保护基础之上的 QNX 实时操作系统，实时、稳定、可靠、强壮，具有模块化程度高、剪裁自如、易于扩展的特点。作为多任务的实时操作系统，QNX 的内核只提供操作系统最基本服务，如任务间通信、同步、时钟等，再通过任务间通信将任务组织起来构成完整的系统，是名副其实的微内核的操作系统。
RT-Linux	硬实时操作系统，由 Fsmlabs 公司开发的基于标准 Linux 的嵌入式操作系统，对 Linux 改动量小[4]，可充分利用 Linux 平台现有的丰富软件资源，此外 RT-Linux 的实时特性与硬件密切相关。到目前为止 RT-Linux 已经成功应用于从航天飞机的空间数据采集、科学仪器测控到电影特技图像处理等领域。
μC/OS-II	它之所以能够引起巨大影响的一个非常重要的原因在于它是第一个公开实现机制的实时操作系统内核。它的绝大部分源码是用 ANSI C 实现的，可移植性较强；与微处理器硬件相关的那部分是用汇编语言实现的，已经压到最低限度，便于移植到其他微处理器上[5][6]。

续表 1.1 实时操作系统概况

操作系统	简介
WinCE	Windows CE 是微软开发的面向小内存 32 位移动智能连接设备而开发的模块化嵌入式实时操作系统[8]。它是有优先级的多任务操作系统，支持 Win32 API 子集，支持多种的用户界面硬件与通信方式的非硬实时操作系统 。它提供与 PC 机类似的图形界面和主要的应用程序。

1.1.2 课题背景

本课题是“微小力测试平台”研究基础上，在“振动系统监控校验保护仪”的研制项目背景下提出的。该系统是由数个子系统组成的，如图 1.1 所示。

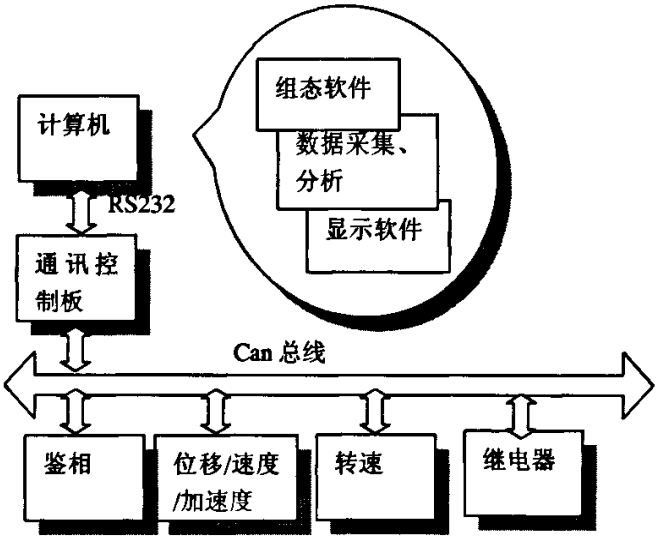

图 1.1 振动系统监控校验保护仪系统构成图

其中每个子系统负责完成各自已既定功能的实现。各子系统之间采用 CAN BUS 或者串口通讯，通讯子系统负责与外界通讯。为了满足系统的性能指标要求，每个系统使用 TMS320F2407A DSP，并在其上移植了 μC/OS-II 实时内核，创建了多个任务有这些：CAN 总线通讯、状态检测、故障监控、报警保护、组态分析、人机界面、串口通讯。由于串口通讯是先通过操作上位机的软件作为输入，设定的信息再经过串口传送到 DSP 来实现的，设定的动作是随机发生的；

同时 CAN 总线通讯自身的网络复杂性，网络数据传输的随机性；外接设备发生的不确定状况，造成系统运行时不确定性，以上种种这一切都是对系统的扰动，这种随机性，不确定性，突发性，导致 CPU 利用率过载，任务丢失率提高。研究中发现，在一定负载情况下， 如果串口上出现干扰信号，系统有时就会陷入“过载”状态，这是因为串口通讯任务的优先级较低，系统“忙于”处理“其他任务”，则会“忽略”了较低优先级的任务，同时优先级较高任务也会由于时限没达到而导致系统性能下降。由此可知，单纯的经典调度策略面对不确定负载时是存在问题的。

μC/OS-Ⅱ是基于优先级的占先式实时内核，并未集成反馈任务调度策略，通过修改实时内核，添加反馈任务调度算法，并基于反馈控制混合任务调度设计应用程序。在论文的第四章详细说明了其应用过程。并对改进后的系统进行验证。通过对比改进前后的 CPU 利用率曲线可知，证明引入基于反馈控制调度方法后的系统能够有效抑制“过载”现象，提高整体性能。

1.2 国内外研究现状

实时调度有多种分类方式，根据建立调度表和可调度性分析是否脱机还是是联机实现，可分为静态调度和动态调度；按系统分类可分为单处理器调度、集中式多处理器调度和分布式调度；按任务是否可抢占，可分为抢占调度和不可抢占调度；按实时性要求，又可以分为硬实时调度和软实时调度。

这里从静态与动态调度角度分类主要包括固定优先级调度和动态优先级调度。

其中固定优先级调度比较成熟的有 RM(Rate Monotonic)调度，早在 1973 年 Liu, Leyland 就提出了适用于可抢占的硬实时周期性任务的静态调度算法，并对其可调度性判定问题进行了研究；随着理论研究方面的不断进展，目前的问题已经集中在如何找到更好更快的可调度性判定方法以及如何扩展 RM 算法；DM 调度算法是在其基础上发展起来的，解决了任务周期小于等于截止期限时 RM 调度的局限性。

动态优先级调度又包括几种：以 EDF 为代表且较为成熟；后来发展起来的 PD(Predictive Deadline)、TBS(Total Bandwidth Sever)调度在一定程度上解决了过载的问题；较新的方法有 DBP(Distance Based Priority)、自适应调度等，但都有

待进一步完善[11]。

实时调度近年来逐渐成为实时系统研究中的热点问题，国外很多学者、研究机构把控制理论中的一些思想和方法应用到实时计算系统中。

美国Virginia大学的Stankovic教授领导的下一代实时计算实验室提出了一个基于反馈控制理论的自适应实时系统整体框架，称之为反馈控制实时调度[53]。反馈控制实时调度用控制理论分析和建立调度系统的模型，把实际的反馈控制技术和调度算法结合在一起，连续地调整调度器来保持最优的性能。反馈控制实时调度能够处理资源不充分和负载不可预测的动态系统，使系统能满足更多任务的时限，提高了系统吞吐率和性能，但这些研究采用的理论基础是经典控制理论，对于实时系统非线性、时变特性没有很好解决。

英国约克(York)大学在支持实时系统设计、实现和校验的方法、体系及工具上进行了大量的研究工作，并提出了基于价值的实时调度方法，以及价值的分配策略。

瑞典的 ARTES 研究中心在实时系统上的研究也十分具有影响力。他们提出了实时系统的硬、软件协同设计思想，并开发了支持实时应用的异构多处理机系统的定义、校验、分析和合成的设计环境。此外，该研究中心还进行了集成控制与调度的研究，将控制理论和调度理论有机地结合在一起，并使用属性语法和递增语义分析在线地实施对任务最坏运行时间的分析及在线地产生处理异常延迟的代码。

此外，在嵌入式实时系统仿真方面，国外瑞典Lund工学院的Dan Henriksson和Anton Cervin 等学者开发的一种基于Matlab 的实时控制与网络控制仿真工具箱Truetime。利用该工具箱，研究人员可构建分布式实时控制系统的动态过程、控制任务执行以及网络交互的联合仿真环境。在该仿真环境中，可以研究各种调度策略和网络协议对控制系统性能的影响。

国内一些开源项目比如 SkyEye 硬件模拟平台，该平台实际是一个开源软件，旨在通用的 Linux 和 Windows 平台上实现一个纯软件集成开发环境，模拟常见的嵌入式计算机系统。可以在 SkyEye 上运行多种嵌入式操作系统和各种系统软件，并可对他们进行源码级的分析和测试。

1.3 论文组织结构

全文分五个章节，第一章是绪论部分，阐述嵌入式实时系统的构成、特点，课题背景，国内外实时调度理论的研究现状。第二章从国内外现有的调度理论出发，分析几个经典的实时调度理论。第三章详细研究了实时控制与任务调度整合设计方法，首先是引入混合任务概念，研究混合任务调度算法，然后讨论了反馈任务调度算法，在反馈调度基础上，讨论并研究了控制与任务调度协同设计方法，最后综合地提出了反馈控制混合任务调度算法，并给出仿真。第四章介绍 “振动系统监控校验保护仪” 项目，阐述系统的功能需求，介绍系统软、硬件实现方法——将 μC/OS-II 在 TMS320LF2407ADSP 上移植 μC/OS-II，详细介绍了 μC/OS-II 移植到 DSP 上的过程，通过描述系统在运行过程中出现的问题，提出了解决方法分析并实现了反馈控制混合任务调度方法。通过与未引入反馈调度系统的 CPU 利用率曲线比较，证明引入反馈控制混合任务调度方法后的系统具有更加好的鲁棒性，更好的性能表现。第五章对全文做以总结，并提出下一步可以进行探讨的几个方向。

第 2 章 实时操作系统中经典任务调度策略研究

2.1 嵌入式实时操作系统基本概念

2.1.1 嵌入式实时操作系统特点

实时内核也称为实时操作系统或 RTOS (Real Time Operation System)。它的使用使得实时应用程序的设计和扩展变得很容易，不需要大的改动就可以增加新的功能。RTOS 的引入解决了嵌入式软件开发标准化的难题。随着嵌入式系统中软件比重不断上升，应用程序越来越大，对开发人员，应用程序接口，程序档案的组织管理成为一个大的课题。通过将应用程序分割成若干独立的任务，RTOS 使得应用程序的设计过程大为简化。RTOS 中最关键的部分是实时多任务内核，它的基本功能包括任务管理，定时器管理，存储器管理，资源管理，事件管理，系统管理，消息管理，队列管理，旗语管理等。这些管理功能是通过内核服务函数形式交给用户调用的，也就是 RTOS 的 API 函数 。使用可占先性内核时，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，消息队列，延时，超时等，RTOS 使得资源得到更好的利用。

目前实时系统界存在的各种各样的 RTOS，而评价 RTOS 性能指标的有这么几方面：

(1) 系统响应时间：系统在发出处理要求到系统给出应答信号的时间。

(2) 任务切换时间：多任务之间进行切换而花费的时间。

(3) 中断延迟时间：从接收到中断信号到操作系统做出响应并完成进入中断服务程序的时间。

2.1.2 实时任务及其特性

任务是指完成某一特定功能的软件实体，它是实时调度中的基本单位。一个实时任务具有的如下基本特性:任务到达时间、任务就绪时间、任务截止期、任务到达频率。根据任务到达频率的不同，任务可以被分为周期性任务和非周

期性任务。对于周期性任务，其相邻两次到达时间之间的间隔是一个固定的常数周期。非周期性任务的相邻两次到达时间的间隔是任意的，没有任何限制。同时，这类任务的到达时间事先是不可预测的，且其特性在任务到达前是不知道的。

此外，任务还具有其它一些与调度相关的特性:开始时间、等待时间、转向时间。等待时间是任务为了取得运行权而需要等待的时间。它是由于其它更高级别任务的抢占或由于该任务要访问一些共享资源而被阻塞所造成的，其大小通常是由调度器的实现方法决定的。转向时间是指任务从就绪到完成的时间。如果忽略开销，则这一时间为任务运行时间与等待时间之和。同时，根据任务按照对截至期限满足情况的不同，可以把实时任务分为以下四种：

1) 硬实时任务： 又称硬截止期任务。是指允许在规定的时间内完成的任务，不允许它的任何任务超时。若有任务未在截至期限内完成，则会对系统造成不可估量的损失。

2) 软实时任务： 又称软截止期任务。允许任务超时，但超时后的计算结果仍然有一定的意义，并且其意义随着超时时间的增加而下降。

3) 固实时任务： 又称固定截止期任务，是软实时任务的一个特例。指允许任务超时，但若任务超时，则该任务的计算结果没有任何意义。

4) 弱强实时任务：通常是周期任务，并且具有这样的特性：允许周期任务的一些任务实例超时，但这些超时的任务实例的分布应满足一定的规律[16]。将这种要求称为超时分布约束。若不满足超时分布约束，则会造成系统动态失效。

2.1.3 实时任务间的相关性

任务间的相关性有以下几种顺序相关性。这是指某些任务之间必须按照一定的先后顺序开始运行。它是由与应用相关的处理顺序、任务间交换数据的需要或任务间通讯而导致的。如果任务间不存在顺序相关性，则称这些任务是独立的。资源相关性包括任务互斥地使用主动资源及任务共享或互斥地使用被动资源。高优先级的任务可以抢占低优先级任务的运行。此外，当若干个任务同时发出运行请求时，具有最高优先级的任务将首先占有处理器。这与策略相关的约束:这主要是指任务的可抢占性，而且是由应用的性质所决定的。对于可抢占任务，如果必要，它在任何时刻都可被其它任务中断;对于非抢占任务，一旦

它开始运行，则在它完成之前都不能被中断。策略导致的约束是由应用的额外要求所引起的。例如，某些任务必须要在某个处理器上运行或者某些任务必须要在某个特定时间运行等。

2.2 多任务调度算法策略的研究

在单处理器中，如何较好地进行任务调度，是这个系统性能好坏的重要指标之一。而在多处理器系统中，如何简单且高效地把任务分配给各个处理器，从而发挥多处理器速度快的特点。调度策略定义了如何选择从就绪状态提升为执行状态的进程。每个多任务操作系统都使用了一定的调度策略。调度策略的好坏直接影响系统的实时性，CPU 的功耗。调度策略的主要指标有 CPU 利用率以及截至期限丢过率等。一个有效的调度满足如下几个条件：

(1)每个处理器在任意时刻只分配最多一个任务。

(2)每个任务在任意时刻最多被分配给一个处理器。

(3)任务只有被释放后才能被调度。

(4)基于所用的调度算法，分配给每个任务的处理器时间总和等于其最大或者实际处理器执行时间。

(5) 所有的优先权和资源使用限制得到满足。

2.2.1 时间片轮转法

当两个或两个以上任务有同样优先级，内核允许事先确定的一个任务运行一段时间，然后切换给另一个任务，也叫时间片调度(time slicing)。内核在满足以下条件时，把 CPU 控制权交给下一个就绪态的任务：

(1) 当前任务已无事可做；

(2) 当前任务在时间片还没结束时已经完成了。

时间片轮转调度算法主要是为分时系统设计的。其中时间片是一个重要的参数，不能取的过大或过小，通常为10至100ms数量级。就绪队列可以看成是一个环形队列，CPU调度程序轮流地把CPU分给就绪队列中的每个进程，时间长度为一个时间片。Linux操作系统就是采用时间片轮转的调度算法。

2.2.2 先来先服务调度算法

先来先服务(First-Come-First-Served, FCFS)调度策略根据任务到达就绪队列的时间来分配处理机，一旦一个任务获得了处理机，就一直运行到结束，先来先服务是非剥夺调度。这种调度从形式上讲是公平的，但它使短任务要等待长任务的完成，重要的任务要等待不重要任务的完成。从这个意义上讲又是不公平的。先来先服务调度使响应时间的变化较小，因此它比其它大多数调度都可预测，由于这种调度方法不能保证良好的响应时间，在处理交互式用户时很少用这种方法。FCFS 算法容易实现，它忽略了服务时间以及其它的一些标准在周转或等待时间方面对性能的影响。在当今系统中，先来先服务算法很少作为调度模式，而是常常嵌套在其它的调度模式中。例如，许多调度模式根据优先级将处理机分配给任务。但具有相同优先级的任务却按先来先服务进行分配。

2.2.3 单调速率算法

单调速率调度算法(Rate-Monotonic Scheduling Algorithm)是由 Liu 和 Layland 在"Scheduling Algorithm for Multiprogramming in a Hard Real Time Environment"提出的， 它开创了实时调度研究领域的新时代，该文提出的 RM 调度算法被证明是最优静态实时调度算法，其后静态实时调度算法研究领域的大量工作均以改进 RM 调度为基础。

该算法是根据任务的周期分配优先权:任务的周期越短，其优先级就越高。RM 算法是一种静态的调度算法，任务的优先权在任务执行之前已经确定且不会随时间变化。此外，RM 算法是抢优的，即当前执行的任务会被新到达的优先级更高的任务打断。做如下假设：

(1) 在单个 CPU 中，所有的任务都是周期性运行的。

(2) 忽略上下文切换时间。

(3) 任务间没有数据依赖。

(4) 任务的执行时间是恒定的。

(5) 所有任务的截止期限都在它们周期的结束点上。

(6) 优先级最高的就绪任务一定会被选择执行。

RM 算法归纳为：优先级按照周期来指定，周期最短的进程优先级最高。

Liu 和 Layland 利用临界时刻分析证明了 RM 优先级指定方法是最优的。

在 RM 调度中，被调度任务集满足下面的条件，则可调度[13]：

$$U_n = \sum_{i=1}^{n} \frac{E_i}{T_i} \le n(2^{\frac{1}{n}} - 1) \tag{2.1}$$

这个条件是充分而非必要的，利用 RM 算法调度的必要条件是：

$$U_n = \sum_{i=1}^{n} (\frac{E_i}{T_i}) \le 1, \forall i, 1 \le i \le n \tag{2.2}$$

表 2.1 RM 调度算法对应的任务数与 CPU 最高利用率

任务数	$n(2^{1/n}-1)$
1	1.000
2	0.828
3	0.779
4	0.756
5	0.743
⋮	⋮
8	0.693

这里 E_i 是任务 i 的最长执行时间， T_i 是任务 i 的执行周期，表 2.1 给出了 $n(2^{1/n}-1)$ 的值。n 是系统中的任务数，对于无穷多个任务，极限值是 $\ln 2$ 或 0.693，这就意味着使用 RM 要使任务都满足截止时限要求，所有有时间条件要求的任务 i 的总 CPU 利用率应小于 70%。值得一提的是，这里说的是有时间条件要求的任务，系统中还可以有对时间没有要求的任务，使得 CPU 的利用率可以达到 100%。但达到 100%并不好，程序没有修改的余地，也没有办法增加新功能。作为系统设计的一条原则，CPU 利用率应小于 60%到 70%。

举例来说明：设一个进程的周期用 T 表示，执行时间用 E 表示，用 P(T,E) 来表示这个进程。假设现在有两个周期进程，分别为 P1(4,2)、P2(6,3)根据 RM 的原则，P1 的优先级高于 P2。下面分析任务的执行情况，选取所有任务周期的最小公倍数作为一帧的长度，调度情况如下：

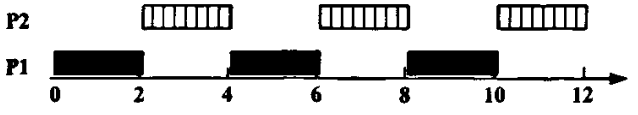

图 2.1 RM 调度算法举例

二个任务同时在 0 时刻到达，由于 P1 的优先级最高，可以立刻执行。二个时间单元之后，P1 结束从就绪态退出直到它的下一个周期到来。在时刻 2，P2 作为就绪状态中优先级最高的进程开始执行。在时刻 4，P1 的下一个周期在 t=4 开始，它中断了 P2 的执行。P2 从来就没有在自己的截至期限中完成任务，在接下来的 12 个时间单元内，调度情况与上述相同。

根据 CPU 的利用率

$$U = \sum_{i=1}^{n} \frac{E_i}{T_i} \tag{2.3}$$

上面的例子中，U=100%, 显然超出了 CPU 的负荷，根据 RM 调度算法计算知道该任务集是不可调度的。

虽然 RM 算法有一定局限性，但它有很多优势，可以在很多实际的系统中使用：

- 调度开销小；
- 平均处理机利用率较高；
- 可保持瞬间过载情况下的稳定性；
- 易于扩展用于解决非周期进程的调度问题。

2.2.4 期限最近者优先调度算法

期限最近者优先(Earliest Deadline First，简称 EDF)是另一个著名的调度策略。它是一个动态优先级方案——在任务的执行期间根据它的启动时间改变优先级。它可以达到比 RM 更高的 CPU 利用率。

EDF 算法中，任务优先级在其初始时并不固定而根据它们的绝对最后期限改变，在 EDF 算法调度的处理器中总是执行所有任务中最早达到绝对最后期限的任务。

EDF 算法是一种最优的单处理器动态调度算法，也就是说，假如 EDF 算法在单处理器上对一个任务集不能合理调度，那么其他算法也不能够对该任务集合理调度。

EDF 调度算法的实现步骤如下：

(1) 检查任务队列中所有就绪任务。

(2) 比较所有任务最后时间期限。

(3) 将具有最先时间期限的任务给予最高优先级即最先执行该任务。

EDF 算法的吸引力在于效率高、容易计算和推断，这是动态优先级调度研究的一个主要组成部分。EDF 的缺点在于，理论表明这种算法能对可调度负载进行优化，但是它不能解决过载问题。发生过载时，EDF 性能退化很快。这种情况不会对严格的硬实时系统造成问题。EDF 调度的实现相对容易。执行队列总是由下一个时限来分类。当一个任务处于激活状态时必须将该任务加入到执行队列中，或者如果任务的时限在当前正在执行的任务的时限之前，那么该任务就会抢先占用当前的任务。如果所有的任务都是周期性的(这些任务的出现仅仅由于时间的流逝)，那么抢先占用就没有必要。这样就简化了调度程序并且降低了任务切换的开销。如果任务可以在周期中的任何时刻执行，那么可实现性分析就简单可行。也就是说，早期的结果是可以接受的，并且任务准备好随时执行。从另一个角度来看，这样就允许任务的执行可以根据周期小幅变化。微小的时限控制要在伪多项式时间类型中进行分析。EDF 算法在实际应用中也存在一些问题，例如动态调度系统开销太大、在过载情况下会出现不稳定现象、优先级反转等。

2.2.5 经典调度策略的局限性

先来先服务和时间片轮转算法原理过于简单，只适合于任务模型比较简单的情况，不能解决随机发生的非周期任务调度问题。而单调速率算法是针对周期任务的，周期越短优先级越高，并且仅在满足式 2.1 的条件下才能对各周期任务进行合理调度，如果系统中还包含非周期任务，则需要将其转化为周期任务再用 RM 理论加以分析。EDF 调度虽然理论上可以达到 100%的 CPU 利用率，但是稳定性极差，同时 EDF 是在线的动态调度方法，运行时所需内存开销较大，统计各个任务的运行时间也很耗时，其应用有很大的局限性。以上的经典调度策略在简单的系统中可以“胜任”，但在大型的实时系统、复杂的实时系统中或者不确定因素较多的系统中，则需要加以修改完善以及灵活运用，才能满足系统整体性能要求。

第 3 章 实时控制与任务调度整合研究

3.1 引言

嵌入式系统不是单纯的做一件事情，而是有很多个任务。这就会涉及多个任务在一个处理器或多个处理器上的调度。本章以嵌入式操作系统为平台，结合系统内核的调度算法，根据任务按照周期和非周期之分，首先对混合任务进行介绍，给出混合任务调度的模型和算法。然后介绍了反馈任务调度算法，基于 PI 控制器对 CPU 利用率、丢失率等控制调度算法进行研究，并给出了利用率、丢失率、利用率丢失率反馈调度算法的对比与总结。通过引入反馈调度概念，对控制与调度进行整合研究，提出 Qoc 调度策略，给出了基于 Qoc 的实时控制与任务调度协同调度模型算法，并通过仿真验证该算法能够有效地减少 CPU 资源的消耗和提高控制的性能。最后在协同调度模型的基础上，提出反馈混合任务调度模型框架，并给出控制器算法与仿真。

3.2 混合任务调度研究

许多实时应用系统涉及混合任务集的调度处理，其中不仅有周期任务与非周期任务，而且任务可能具有硬截止期、固定截止期或者软截止期。具有混合任务集的实时调度算法的目标是：

（1）保证硬截止期任务满足截止期

（2）最大化固定截止期任务的完成数量

（3）最小化软截止期任务的平均响应时间

在实时混合任务调度中，为了提高非周期任务调度的性能，人们提出了许多非周期任务调度算法，主要分成如下两类：基于服务器的算法和基于空闲时间的算法。 基于服务器算法主要指在保证满足周期任务截止期的前提下，引入一个或者几个额外的周期任务，使用指定的 CPU 带宽作为服务器来处理非周期任务。基于空闲时间算法主要通过离线或者在线分析从周期任务调度的空隙获得尽可能多的时间来处理非周期任务。

3.2.1 混合任务调度算法

这里首先介绍空闲时间算法大致概况，然后详细讨论一种基于空闲时间的混合任务调度算法。

空闲时间偷取算法是以处理软截止期的非周期任务的，它是基于离线地计算可用的空闲时间数量。系统运行时启动一个具有最高优先级空闲时间偷取任务，当有非周期任务到达时它利用所有可用的空闲时间调度这些任务，当所有空闲时间已经耗尽而任务没有完成，这个任务将被挂起直到系统有多个可用的空闲时间。

另一方面，时间片移位(Slot Shifting)算法，主要针对静态调度系统中的混合任务集的联合调度问题。该算法首先为周期任务构造一个静态调度表，满足任务的时间、同步与通信需求；接着离线计算空闲处理时间（Spare Capacities）的大小与分布，记录每个空闲处理时间的位置以及随后的周期任务执行容许的偏移时间。

由于时间片移位算法只能应用于静态调度系统，而双重优先级方法是一种相对优雅且空闲时间调度处理的软实时任务的算法。此算法是 Davis 等人提出的，算法中定义三个优先级层次：高、中与低。每个硬实时任务被分配两个优先级，一个在高优先级层次，另一个在低优先级层次。通常任务之间的优先级顺序在两个优先级层次上保持一致，而非周期任务运行在中优先级层次上。每个硬实时任务的开始时刻运行在其较低的优先级上。当一个硬实时任务继续在低优先级层次上可能导致截止期丢失时，其优先级将被提升为其在高优先级层次上的优先级。这样，软实时任务优先于未经历优先级提升的硬实时任务，相应地推迟了硬实时任务的执行，因此相当于从硬实时任务偷取了空闲时间，从而提高了软实时任务的响应时间。

这里提出的一种基于空闲时间的混合任务调度算法，给出如下假设：

假设实时系统中包含两类任务：硬实时任务与软实时任务。系统具有固定数量的硬实时任务 T_i，i=n。这些任务是周期，并且其截止期等于周期或者最小间隔到达时间；而软实时任务是具有固定截止期的非周期任务。每个非周期软实时任务的最大执行时间是已知的，以便让调度器决定接受哪些非周期软任务，拒绝哪些非周期软实时任务。同时假设所有的非周期软实时任务可被抢占，保证每个非周期软实时任务如果不能在一帧内完成时，可以在以后的帧执行。系

统调度的目的是在于保证所有硬实时任务满足截止期，并且尽量降低软实时任务的截止期丢失率。

首先介绍可调度测试：在每个非周期软实时任务到达时，调度器进行可接受测试，以考查此时在不影响系统现有任务及时执行的情况下，是否还有能力处理这个新到达的任务。如果根据现有调度，系统有足够的剩余时间在其截止期限前处理完这个新到达的非周期软实时任务，并且不会导致系统中其他任务截止期丢失，调度器就接受这个任务，反之调度器就拒绝。

在系统运行时，可能同时有多于一个的非周期软实时任务等待被测试，一种较好的排序方法是基于最早截止期限优先(EDF)方法，即新到达的非周期软实时任务以它们的截止期限非递减顺序在等待队列中排序——截止期限越早，在等待队列中的位置越靠前。调度器总是测试队列头的任务，在调度完成后移除该任务或拒绝之。

假设一个非周期软实时任务 $S(d, e)$具有截止期限 d 和最大执行时间 e，在第 t 帧做可接受测试。对于新到达的非周期软实时任务，其可接受测试都是在每一帧的开始进行的，假设在第 t 帧测试是否接受或拒绝非周期软实时任务 $S(d, e)$，测试包括下面两个步骤：

(1) 考查从当前到任务 S 的截止期限前，总的空闲时间是否大于等于其执行时间，如果否，则拒绝之，如果是，还要进行下一步。

(2) 考虑调度器如果接受 S，系统中其他的非周期软实时任务是否不能及时完成。如果不受影响则接受之，否则拒绝之。

当 S 的截止期限 d 在第 $l+1$ 帧，$l=t$，显然这个任务必须在第 l 帧或更早调度完成，这时当且仅当帧 $t, t+1, \ldots l$ 总的空闲时间 $\sigma_c(t,l)$ 大于等于执行时间 e，这个任务才能够及时完成[24]。因此，如果 $\sigma_c(t,l) < e$，调度器就应该拒绝接受该任务。

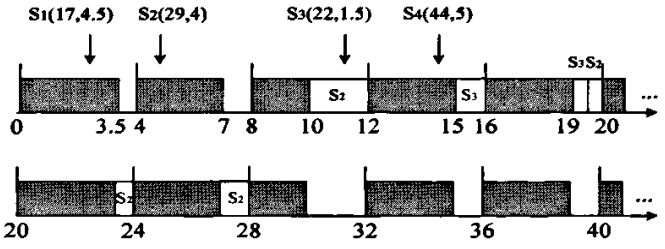

图 3.1 基于空闲时间的非周期软实时事件调度

另外，考虑到调度器有可能让这个新的非周期软实时任务先于以前接受的非周期软实时任务执行，还应该检查接受该任务是否引起系统中其他非周期软实时任务不能按时完成，则可接受条件除了 $\sigma_c(t,l)=e$ 外，还有要保证系统中其他任务不受影响。

在每帧中，当周期任务调度完后，调度器马上令等待着的非周期软实时任务利用这些空闲时间依次执行。如图 3.1 所示。

注意图中四个非周期软实时任务 S_1、S_2、S_3、S_4，除了 S_1 外，其余都是在周期任务的空闲时隙中执行，因此非周期软实时任务永远不会影响到周期任务。为了进行可接受测试，调度器需要知道从帧 i 到帧 k 的总共空闲时间 $\sigma_c(i,k)$。可以在运行前就计算好初始的 $\sigma(i,k)$，$i,k=1,2,\ldots,N$，把它们存储在 N^2 个单元内[26~27]。由此，以后任意两帧间的空闲时间都可以由以下的方法计算得出。设 $0<j<j'$，且 $i,k=1,2,\ldots,N$，第 j 个周期中的第 i 帧到第 j' 个周期中的第 k 帧的空闲时间计算公式为：

$$\sigma(i+(j-1)N,k+(j'-1)N)=\sigma(i,N)+\sigma(1,k)+(j-j'-1)\sigma(1,N) \qquad (3.1)$$

一般地，假设在当前帧 t 开始时，系统中有 n_s 个非周期软实时任务，称为 $S_1, S_2, \ldots, Sn_s$。令 d_k 和 e_k 分别代表 S_k 的截止期限和执行时间，ξ_k 代表在此帧之前任务 S_k 已经执行了的时间，设被测试的任务 $S(d,e)$的截止期限处在第 l+1 帧，从帧 t 到帧 l 的总空闲时间 $\sigma_c(t,l)$ 可以根据初始存储的 $\sigma(t,l)$ 由下式求出：

$$\sigma_c(t,l)=\sigma(t,l)-\sum_{d_k\le d}(e_k-\xi_k) \qquad (3.2)$$

上式中求和表达式涉及所有截止期限小于等于 d 的非周期软实时任务，由于这些帧内的空闲时间必须先用来执行这些非周期软实时任务尚未执行完的部分，仅仅它们余下的时间才可以被任务 $S(d, e)$利用。如果$\sigma_c(t,l)=0$，非周期软实时任务 S 可能被系统接受，这时在 S 截止期限前的空闲时间总和为：

$$\sigma = \sigma_c(t,l) - e \tag{3.3}$$

可以说任务 S 有 s 个单位的空闲时间，仅当 s=0 时，任务才可能被系统接受。一旦被接受，这个空闲时间 s 就会被存储起来以备后用。

基于 EDF 的调度算法，接受任务 S 可能会引起现有的截止期限大于 d 的非周期软实时任务 S_k 不能及时完成。特殊地，如果接受了 S，S_k 的空闲时间 s_k 减去 S 的执行时间 e，如果结果不小于 0，那么 S_k 将不会错过截止期限。调度器必须考虑系统现存的所有截止期限小于 d 的非周期软实时任务，只有当它们中的任何一个减得的空闲时间都不小于 0，任务 S 才通过可接受测试而被调度器所接受[28]。

由此可知，调度器用以进行可接受测试的数据包括：

(1) 初始计算好的从帧 i 到帧 k 的空闲时间$\sigma\ (i,k)$，$i, k = 1,2,\ldots,N$;

(2) 每个非周期软实时任务 S_k 在当前帧 t 之前已经执行了的时间 ξ_k ；

(3) 当前系统中现存的每个非周期软实时任务的空闲时间 s_k。

利用空闲时间的非周期软实时事件调度算法，是对系统中随机出现的非周期软实时任务进行调度的一种有效方法，简单易实现，但占用内存空间较多。

3.2.2 混合任务调度模型框架研究

如同上节假设，存在两类任务,一类是硬实时周期任务，另一类是周期和非周期的软实时任务。这里给出一个混合实时任务调度框架，其中主要包括实时任务解析器，非周期软实时任务队列，周期软实时任务对列，非周期软实时任务调度器，硬实时任务队列，调度任务对列，调度器。由非周期软实时任务调度器和调度器共同组成联合调度器如图 3.2 所示。

软实时任务到达后，由实时任务解析器负责对任务分析，根据任务的周期性和非周期性分别把任务放入队列中。周期软实时任务队列和硬实时任务队列直接进入调度任务队列，然后给调度器进行调度，而非周期实时任务队列则直接进入到非周期软实时任务调度器进行调度；非周期软实时任务调度器是对调

度器的一种补充调度，根据混合任务调度算法，改善常规调度器的调度算法。满足系统性能要求，保证硬实时任务截止期按时完成，软实时任务达到低的任务丢失率。

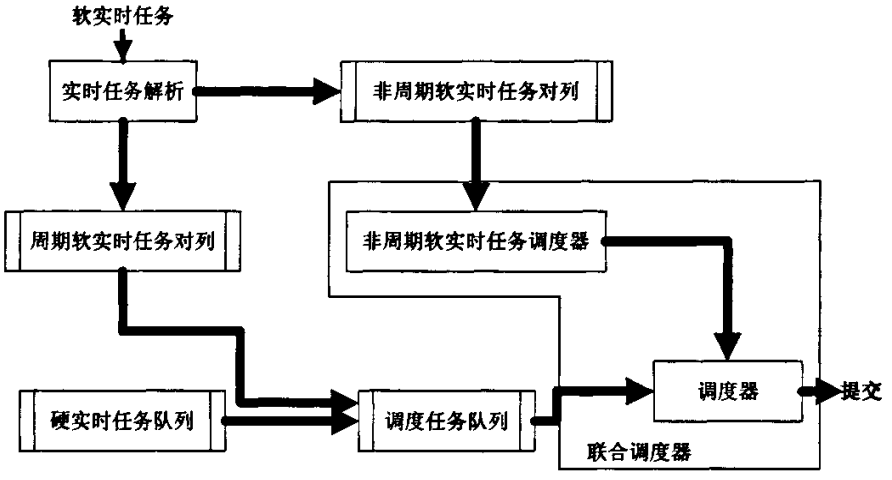

图 3.2 混合实时任务调度框架

3.3 反馈任务调度研究

3.3.1 反馈控制实时任务调度概况

目前，实时任务调度主要有三个范例：RM 算法、EDF 算法和 Spring 算法。虽然它们构成了实时调度的整体框架，但是现实中的很多问题并不能仅仅通过使用这些算法来解决，因为它们都是开环调度算法。开环调度算法在可预见环境(系统的工作流能够被准确的建模)下能够很好的工作，但是在不可预见环境下性能就很差。而当前的工程技术的应用中，在开放的不可预测的环境中运行的实时应用正在快速的发展。在这些应用中，使用开环调度算法只能导致高代价、低利用率。为了在不可预测环境中保证调度的性能，出现了反馈控制调度算法，这是因为控制理论中的反馈能够使系统趋于稳定，控制理论中的反馈与决策理论中的调度相结合，就构成了反馈调度的思想。当前反馈控制理论已经成功的应用到自适应系统的设计中，而主要还是应用在机械和电气系统中，本节所述的是由 Chenyang Lu 和 John A.Stankovic 等人所提出的反馈控制实时调度

算法(FC-RTS)，并将反馈控制理论应用到嵌入式系统任务调度的领域中。

3.3.2 反馈控制实时任务调度的框架结构

反馈控制实时调度的框架设计主要包括下列元素：

(1) 把反馈控制结构映射到实时系统适应资源调度的一个调度结构。

(2) 一个性能规范和度量来表征适应实时系统的瞬态和稳态性能。

(3) 基于控制理论的，满足系统性能要求的资源调度算法设计的方法。

3.3.2.1 控制理论与实时调度

典型的反馈控制系统由控制器、被控对象、执行器和传感器组成。如图 3.3 所示。系统必须根据实际的应用确定控制变量、参考值和调节变量，控制变量指的是被测或者被测输出的大小，参考值表示控制变量的理想值，而系统误差指的是当前控制变量的值与参考值的差。控制器通过改变调节变量的值来影响控制变量。基于反馈控制的系统机理为：

1. 系统通过周期性地监测和比较控制变量和参考值来确定系统误差；
2. 控制器以系统误差作为输入，通过一定的控制函数计算调节变量；
3. 执行器通过改变调节变量来对系统进行控制。

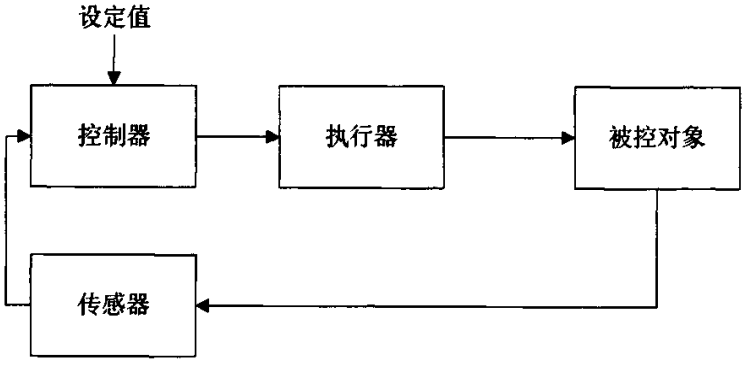

图 3.3 反馈控制系统结构图

在实时调度系统中，把调度系统当成反馈控制系统，调度器作为控制器，调度器利用反馈控制机制在不可预测的情况下可以取得满意的系统性能。控制系统经常使用的 PID 算法在反馈调度系统也可使用，这主要原因是 PID 在控制界比较熟悉，算法比较简单；不需要对被控对象精确建模，只要基于系统的大

致模型就能获得满意的系统性能;目前基于 PID 的改进先进算法很多也很成熟，对于复杂系统，可以使用改进算法便于控制系统。

3.3.2.2 反馈控制实时调度结构

如图3.4所示，反馈控制实时调度结构是由一个反馈控制环和一个基础调度器组成的，这个反馈控制环包括一个监视器、一个控制器、一个负载调节器。其中监视器作为检测机构，负载调节器作为执行机构，主要用于调整CPU的分配与使用，调节手段包括准入控制、Qos调节或者综合使用准入与Qos调节，根据不同的系统需求与特点考虑使用不同的方法。反馈控制环在每个采样周期都被激活。

在每个采样时刻k，Qos调节器根据控制输入D(k+1)调整任务的Qos等级，这样就能动态调整总的预计利用需求。Qos调节器就是强制使总的利用需求达到控制器要求的预计利用需求。Qos调节器使用一种Qos最优算法来使系统值达到最大。在最简单的情况下，每个任务有两个Qos等级，Qos调节器实际上成为一个准入控制器。每个任务的到来都会激活Qos调节器，这个准入检测隔离了由任务到达率的变化带来的干扰。

基本调度器用某种调度算法(如EDF或者RM)来调度进入系统的任务。

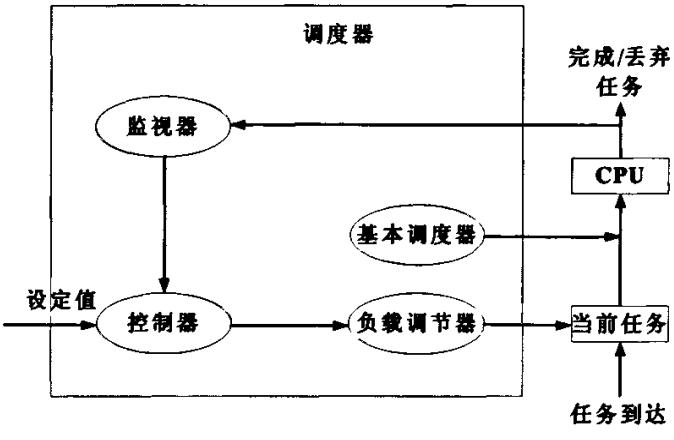

图 3.4 反馈调度实时控制结构

在这个闭环系统中，相应的控制相关变量及性质如下：

(1) 被控变量:

被控变量是由调度器控制的性能度量变量。实时系统的被控变量包括截止时间丢失率 M(k)，CPU 利用率 U(k)，它们都是定义在一个时间窗((k-1)W，kW)里的，W 是采样周期，k 是采样瞬间。在第 k 个采样瞬间的丢失率 M(k)被定义为在采样窗((k-1)W,kW)里的错过截止时间的任务数目除以完成的或者中止的任务的总数。丢失率通常是实时系统性能度量的最重要的变量。在第 k 个采样瞬间的利用率 U(k)是在采样窗((k-1)W, kW)里 CPU 的被占用时间的百分比,它是考虑实时系统的开销和吞吐量时的被控变量。

(2) 设定值:

表示受控变量取性能参考值的时候，系统能够得到期望的性能。调度系统中期望丢失率 Ms 和期望 CPU 利用率 U_S。

(3)操作变量:

在反馈调度结构里，操作变量是系统里所有任务的总的预计利用率 B(k)= $\sum_i U_i[l_i(k)]$，其中 l_i 是任务 T_i 在第 k 个采样窗里的 QoS 级别。

3.3.2.3 性能指标与度量

对于一个反馈控制实时任务调度系统的基本要求包括：(1)系统必须是稳定的；(2)系统的动态性能满足所需要的指标；(3)静态误差满足要求。其中实时系统的主要性能指标以及典型的两种负载形式有：

- 稳定性：如果实时系统的丢失率和利用率是在有界输入条件下输出在限定的范围内，那么该系统就是稳定的。
- 瞬态响应：描述 CPU 资源调度系统对负载变化的动态响应。
- 超调量σ%：丢失率或利用率的最大偏离量除以丢失率或利用率的设定值，

  $Mo=(M_{max}-Ms)/Ms$，　$Uo=(U_{max}-Us)/Us$。
- 过渡过程时间 Ts：描述了系统达到期望的利用率和丢失率的稳态的快慢。
- 到达负载：对于这种类型的过载，负载的变化△L=(Ln-Lm)是由新任务的到达产生的。
- 内部负载：对于这种类型的过载，负载的变化△L = (Ln - Lm)是由已经在系统内部任务的利用率增加产生的。内部负载可以看作干扰输入，也可以看作系统模型参数的变化。

3.3.3 反馈控制实时调度算法模型

在上述的 FC-RTS 结构和性能指标的基础上，运用反馈控制理论方法设计 FC-RTS 算法。首先是建立 FC-RTS 结构中被控系统的模型。被控系统的输入是估计利用率，输出是丢失率 M (k)和利用率 U(k)。尽管建立非线性、时变实时

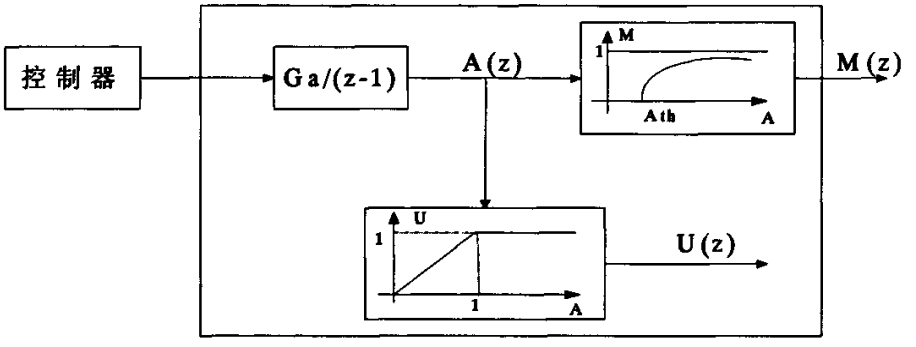

图 3.4 实时 CPU 反馈调度系统模型

系统的精确模型很困难，但可以用一个线性模型来近似替代。实时 CPU 反馈调度系统模型方框图如图 3.2 所示。

该系统由两部分组成，一部分是控制环节。另一部分是被控对象。被控对象是具有非线性的时变环节。由于每个任务准确的执行时间是不可知的，也是时变的，所以总的实际执行需求利用率 A(k)可能与总的预计需求利用率 B(k)不同:

$$A(k)=Ga(k)B(k) \tag{3.4}$$

式中，Ga(k)—利用率比率，它是一个时变的比率，表示实际的利用率 A(k)相对于估计的被请求的利用率 B(k)的变化程度。例如，当 Ga(k)=1.5 表示实际的被请求利用率 A(k)是估计的被请求利用率的 1.5 倍。由于 Ga(k)是时变的，在设计时为了保证稳定性，文献[53][56]采用了保守的设计方法，用 Ga(k)的最大值 $G_A=\max\{Ga(k)\}$来替代系统模型 Ga(k)，这就必然影响系统的动态品质。

在利用率这个回路中，总的被请求的利用率 A(k)与被控变量之间的关系是非线性的。当 CPU 未充分利用时，利用率 U(k)等于总的被请求的利用率 A(k):

$$U(k)=A(k),\quad A(k)=1 \tag{3.5}$$

当 CPU 过载时，也就是总的被请求的 CPU 利用率 100%时候

$$U(k)=1,\ A(k)>1 \tag{3.6}$$

在丢失率这个回路中，丢失率 M(k)与利用率 U(k)互斥，当总的被请求的利用率 A(k)低于可调度性阈值 $A_{th}(k)$时：

$$M(k)=0,A(k)=A_{th}(k) \tag{3.7}$$

实时调度理论中，不同的实时调度理论在不同负载情况下的可调度阈值 $A_{th}(k)$都已经被推导出来[55][57]。由于 $A_{th}(k)<1$，所以利用率的饱和区(A(k)> 1)与丢失率的饱和区(A(k)< $A_{th}(k)$)是互斥的，即在任何时刻，至少有一个被控变量不会饱和。

当 $A(k)>A_{th}(k)$的时候，M(k)通常是随着总的需求利用率 A(k)的增加而非线性增加的。M(k)和 A(k)之间的关系可以通过在性能参考值 M_s 附近微分来进行线性化。可以得到错过率因子 G_m：

$$G_m=\frac{dM(k)}{dA(k)} \tag{3.8}$$

$$M(k)=M(k-1)+G_m(A(k)-A(k-1)),\ A(k)>A_{th}(k) \tag{3.9}$$

通过控制理论方法对模型进行 z 变换，对于建立的 FC-RTS，根据式 3.4-3.9，可以得到它们的饱和区以外经过变换分别得到的传递函数：

- 利用率：　$P_U(z)=Ga(z)/(z-1),A(k)<1$　(3.10)
- 丢失率：　$P_M(z)=G_mGa(z)/(z-1),A(k)>A_{th}(k)$　(3.11)

3.3.4 反馈控制实时调度控制器设计

基于前述的 FC-RTS 结构和性能指标的基础上，设计一个简单的控制器，具有如下特点：

(1) 保证系统稳定。

(2) 稳态误差为零。

(3) 动态特性较好。

选用比较经典的 PID 控制器，通过调节控制器的控制参数，仿真选择合适的控制参数，考虑到微分作用可能使得负载变化引起的噪声信号放大，产生干扰，并且微分作用往往使得系统稳定性能变坏。因此排除微分作用直接使用 PI 控制。

对于 PI 控制器，离散化后的控制器的传递函数

$$D(z)=Kp(1+\frac{K_I}{1-z^{-1}}) \tag{3.12}$$

采用如图 3.5 所示的 PI 控制方框图，通过对利用率反馈控制回路在线性范围条件下的阶跃响应进行仿真试验，系统的传递函数如式 3.13 所示：

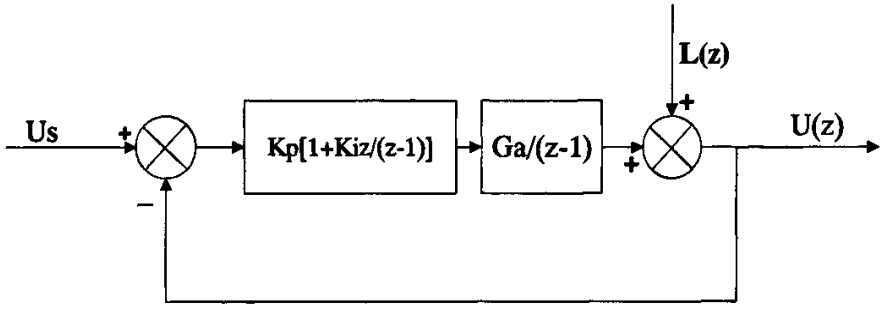

图 3.5 PI 控制方框图

$$H(z)=\frac{K_pG_a[(K_I+1)z-1]}{z^2+[K_pG_a(1+K_I)-2]z+1-K_pG_a} \tag{3.13}$$

可以看到闭环系统的特征方程为二阶方程，它存在两个根。根据控制理论的知识和有关文献可知要使离散系统稳定必须使它的特征根落在单位圆内并且极点越靠近原点它的过渡过程时间越短，考虑系统最恶劣的情况(Ga=2)系统仿真结果如图3.6所示。系统的系统的PI参数，以及对应过度过程时间和误差绝对值的积分(IAE)如表3.1所示，其中时间单位为0.1毫秒，采样周期为0.05秒。

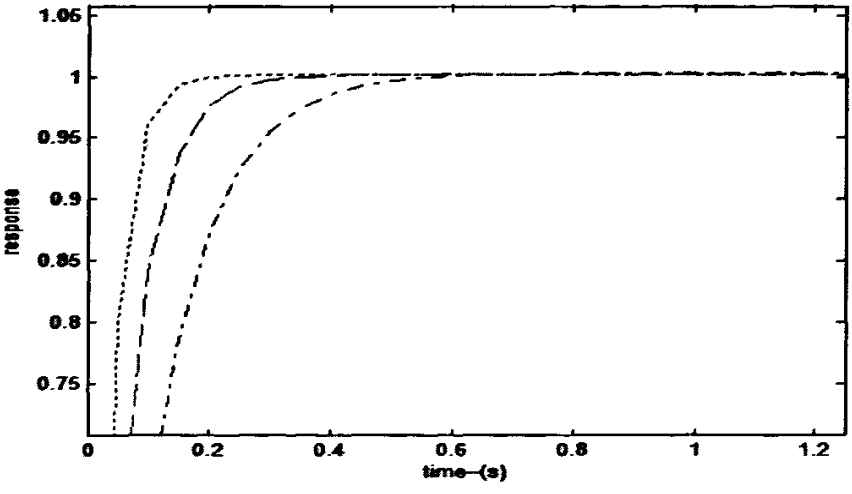

图 3.6　PI 控制仿真结果

表 3.1　PI 参数、过渡时间和 IAE

PI 参数	图线类型	过渡时间(s)	误差绝对值的积分(IAE)
K_p=0.2,K_I=0.001	虚线	0.35	0.134
K_p=0.3,K_I=0.001	长划线	0.2	0.090
K_p=0.4,K_I=0.001	点划线	0.12	0.068

在实际仿真过程中，比例系数对系统的影响较大，而相对而言，积分系数对系统影响较小。

3.3.5 反馈控制调度算法分析

在文献[59][56]中，Chengyang Lu 等人提出的反馈控制调度算法，包括利用率反馈调度算法(FC-U)，丢失率反馈调度算法(FC-M)以及利用率丢失率反馈调度算法(FC-UM),其中控制器采用 PI 控制。FC-U、FC-M 的控制回路方框图如 3.7、3.8 所示，FC-UM 的算法模型图如 3.9 所示。

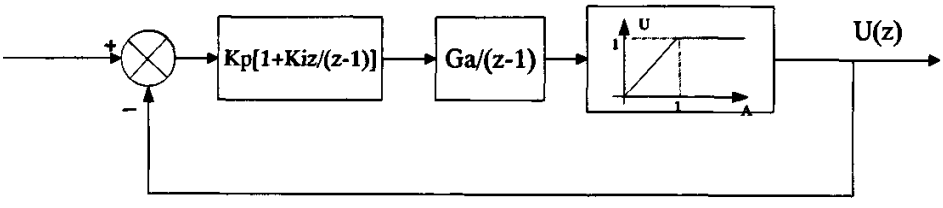

图 3.7 利用率反馈控制回路方框图

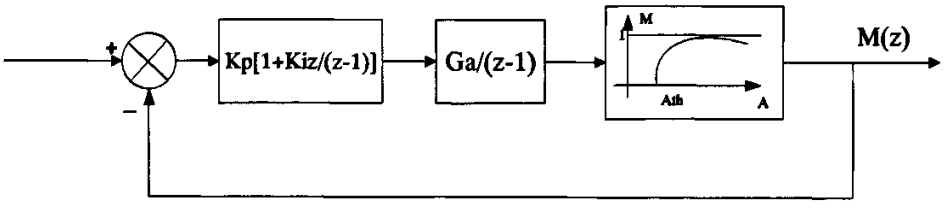

图 3.8 丢失率反馈控制回路方框图

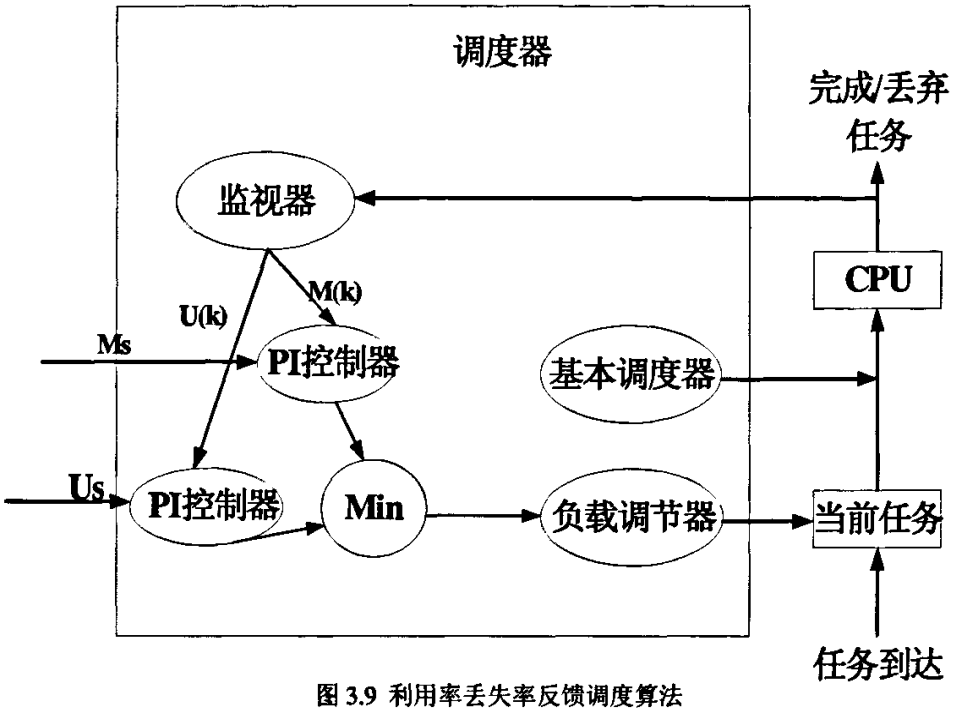

图 3.9 利用率丢失率反馈调度算法

利用率反馈控制调度算法FC-U，采用一个利用率控制回路控制利用率U(k)；丢失率反馈控制调度算法FC-M，使用一个丢失率控制回路来直接控制系统的丢失率M(k)。

在利用率丢失率反馈调度算法 FC-UM 中，监视器对 CPU 的利用率和丢失率进行监测，将测量到的利用率和丢失率分别与其设定值相减，得到利用率和丢失率误差，作为控制器的输入，然后把两个控制器的输出的最小值作为负载调节器的输入值，对提交到系统的任务进行准入控制，基本调度器按照一定的调度算法调度准入的任务。由于利用率丢失率反馈控制算法对利用率反馈控制算法和丢失率反馈控制算法两个控制信号进行取最小运算，这意味着该调度算法在稳态时，取得相对保守的性能。当 Us=Ath(k)时，利用率控制起主导作用。当 Us>Ath(k)时，丢失率控制起主导作用。 以上这些算法的优缺点如表 3.2 所示。

魏立峰等人在文献[60]中，提出利用率丢失率混合线性反馈控制调度算法(FC-LUM),这里对该调度算法的结构框图进行修改，使用 PI 控制器，如图 3.10 所示。

表 3.2 FC-U,FC-M,FC-UM 算法优缺点比较

	缺点	优点
FC-U	在 CPU 利用率处于饱和时，不能检测出系统过载的程度，即过渡时间比线形模型时的长；不合适利用率阈值未知而且悲观的系统。	适合利用率阈值已知并且不悲观的系统，在牺牲利用率的情况下，系统能取得满意的控制效果；在远离饱和区时，瞬间响应比较快，过渡时间较短。
FC-M	设定值不可能太大，由于饱和特性，系统偏差小，瞬间响应差，过渡时间随着丢失率设定值减少而增加；适用于可以容忍时限丢失的软实时任务。	不需要已知利用率阈值，丢失率回路总是改变总的任务所需利用率 A(k),使其稳定在利用率的可调度阈值附近，可以取得较高的 CPU 利用率。
FC-UM	很难设定一个合理的利用率设定值 Us，改善系统品质不大。	同时兼顾利用率与丢失率控制的优点。

FC-LUM 调度算法的控制决策如下：

- 如果系统丢失率 M(k)=0,此时丢失率处于非线性饱和状态，利用率 U(k)=A(k)=Ath(k)=Us=100%,利用率 U(k)处于线形工作状态，选择 FC-U 控制。
- 如果系统丢失率 M(k)>0,此时利用率 U(k)=Ath(k),即利用率处于或接近非线性饱和状态，而丢失率处于线形工作状态，选择 FC-M 控制，使利用率稳定在利用率可调度阈值 Ath(k)附近。

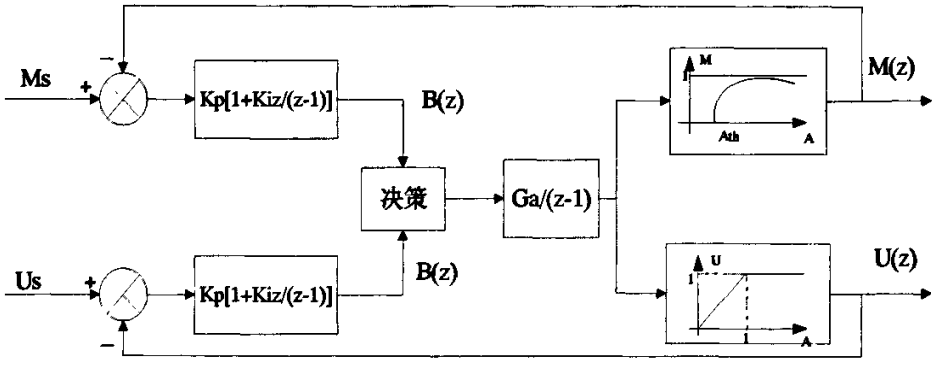

图 3.10　利用率丢失率混合线形反馈控制调度框架

FC-LUM 调度算法通过 FC-U 调度算法和 FC-M 调度算法动态实时切换，使实时系统处于全局线形工作状态，可以获得较好的动态响应。

3.4 控制与任务调度协同设计研究

3.4.1 控制与任务调度的系统设计概括

在实时系统中，尤其在基于控制的实时系统中，系统经常受到外界扰动，使用离散控制理论设计闭环系统，设计值如采样周期，采样延迟等在实时系统中往往成为固定时限。传统实时系统中，固定时限法忽略了控制任务执行时被控对象的动态特性，即它消弱了控制任务的自适应性，阻碍了系统对扰动的快速反应能力，以及在稳态和扰动状态下，对系统资源的合理调度，当系统中存在多控制任务时候，动态性能变得很差。采用这种设计方法，往往使控制器执行动作不能动态地随着外界变化而改变。例如当系统受到外界扰动时候，需要提高系统采样频率，使得系统尽快恢复到稳定状态；当系统恢复到稳定状态后，需要降低系统采样频率以节省系统使用资源。

本节从提高闭环系统性能出发，通过引入 IAE 性能指标，作为 Qoc(Quality of Control)的评价基础，对控制任务时限进行灵活分配调整，改善闭环动态性能，最后给出基于 Qoc 的实时控制与调度协同设计模型与算法。

3.4.2 控制与任务调度的设计性能指标和度量

3.4.2.1 控制任务的时间属性与约束

在闭环控制系统中，回路控制为周期性任务，对于周期性任务，时间属性由四元组<a_i, s_i, f_i, d_i>表示，分别代表任务的到达时间，执行时间，完成时间和截至时间。

例如在一个单回路控制系统中，系统由采样（A/D 转换），控制输出（D/A 转换），控制计算，数据更新这几部分组成。通常控制器执行如下算法：

```
LOOP
    Wait(ClockInterrupt);                    '等待计时器中断
    A_D_Conversion;                          'A/D转换
```

```
CalculateOutput;                    'control quantity calculation
D_A_Conversion;                     'D/A conversion
UpdateState;                        'update data
END
```

通常，数据更新这个过程占用时间忽略，而 A/D 转换需要耗费时间 T_s，D/A 转换消耗时间 T_a，控制计算使用了 T_c，那么整个反馈延迟如下定义：L 为整个闭环从采样，控制计算到控制输出所需要的时间，即：$L=T_s+T_a+T_c$。设 T 代表控制周期。（T>>L）如图所示：

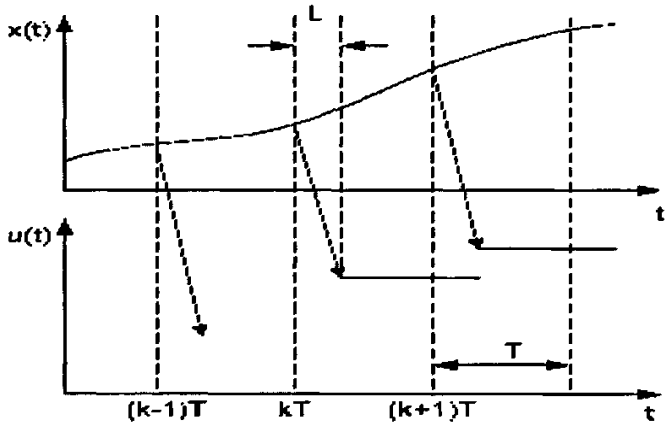

图 3.11 反馈延迟对控制输出的影响

在可抢占性RTOS中，低优先级任务往往被高优先级抢占对CPU访问权，任务得不到及时响应，由此出现了抖动现象，如图所示。

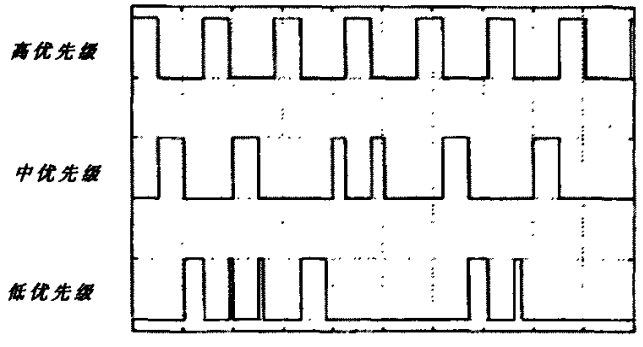

图 3.12 低优先级任务出现抖动

3.4.2.2 控制任务周期序列集合概念

设存在一个这样的集合 CH，使得所有满足使对象稳定，任务可调度的控制任务周期 hi∈CH,i∈N。如果存在一组 $h_k,h_{k+1},.....h_{k+n},\forall h_{k+i}\in CH,i=n$,那么这一组序列可以表示成 seq<$h_k$>，($h_{min}=h_k=h_{max}$)。因此 seq<$h_{max}$>指序列 $h_{max},h_{max},...h_{max}$；同理 seq<$h_{min}$>指序列 $h_{min},h_{min},...h_{min},(h_{min},h_{max}\in CH)$。

3.4.2.3 基于 Qoc 的性能评价基础

在经典控制系统中，系统性能指标主要是：快速性，准确性和稳定性。除了这些指标以外，系统误差一直是控制器设计时候所关注的。这里使用另外一种常用的性能指标：误差绝对值的积分(IAE)，它定义为：

$$IAE=\int_{t_0}^{t_f}|y_{des}(t)-y_{act}(t)|dt \tag{3.14}$$

这里 y_{des} 是设定值，y_{act} 是实际响应值，t_0 和 t_f 分别代表初始时间和结束时间。基于 IAE，以如下方式定义 Qoc:

$$Qoc(y_{act}:seq<h_k>,h_k\in CH)=\frac{IAE(y_{act}:seq<h_{max}>)-IAE(y_{act}:seq<h_k>)}{IAE(y_{act}:seq<h_{max}>)-IAE(y_{act}:seq<h_{min}>)} \tag{3.15}$$

y_{act}:seq<h_k>表示控制任务以这个序列seq<h_k>执行控制时候，采样得到的 y_{act}。由此可以看出Qoc被映射在[0,1]范围内。Qoc和IAE成反比，也就是IAE越大，即系统误差越大，Qoc越小，控制品质越差；当IAE越小，即系统误差越小，Qoc越大，控制品质越好。

seq<h_k>中采样周期选择直接影响控制性能，这里假设 h_k 都一样，即 seq<h_k>序列是一恒值序列。以直流伺服电机为例，在单控制任务环境下，选择一组恒值序列（从 3ms 到 18ms），得出基于 Qoc 控制性能参数，如表 3.3 所示。

从表 3.3 中可以看出，控制任务运行在采样周期为 3ms 时系统获得最好的 Qoc；而运行在 18ms 时候，系统获得最差的 Qoc。也就是在 3ms 时扰动产生的误差比起 18ms 时候要小得多。因此，可以得出这样一个结论：一个控制任务采样周期所选择得恒值序列越小，那么系统的误差也越小（获得更好得 Qoc），系统的动态性能也越好。

表 3.3 恒值序列对应的 IAE 和 Qoc

h_k(ms)	IAE	Qos
18	0. 928	0. 000
16	0. 591	0. 707
12	0. 509	0. 878
8	0. 482	0. 935
4	0. 459	0. 983
3	0. 451	1. 000

3.4.3 基于 Qoc 的实时控制与调度协同设计模型与算法研究

在实时调度问题中，面对多控制任务，当被控对象受到外界扰动时，如何改善控制系统的动态响应并且保证所有任务可调度；如何优化多控制任务的控制性能。以下，首先讨论一下基于 Qoc 的调度策略，使用一个例子来仿真说明；然后针对控制延迟和抖动给出一个补偿算法，最后引入实时控制与调度协同设计模型，并予以仿真。

3.4.3.1 Qoc 调度策略

实时控制系统中，控制系统的控制性能与CPU资源的合理高效应用一直是一对矛盾。为了在保证好的控制性能下尽可能的充分利用CPU资源，并且在控制性能与CPU资源上做一个较好的折中。于是把控制任务分为稳态过程和暂态过程分成两个方面。在自动控制原理中，区分稳态过程和暂态过程的区别是进入5%或2%内的超调就认为进入了稳态阶段，而在这之前属于暂态阶段。如果能够让被控对象迅速地进入稳定状态，在过渡时期时给控制任务分配更多CPU资源，而当被控对象进入稳定过程时候，减少使用的CPU资源，并把带宽留给其他任务使用，这个就是Qoc调度策略思想。具体如下：

这里假设控制任务按照采样周期序列执行如下动作：

(1) 当控制系统在稳态时，控制任务以原有的周期执行。

(2) 当发现扰动时，通过改变 seq<h_k>，提高 Qoc 尽快消除扰动。

扰动时刻到来时，在保证所有任务可调度的前提下，选择较小 seq<h_k>提高控制任务的 Qoc。在最差情况下，序列 seq<h_{max}>虽然不能提高 Qoc，但确保证

系统稳定性，满足控制性能要求。

Qoc 调度另外需要注意的两方面，扰动到来的确立以及所有任务可调度的判断。Qoc 调度方法需要对扰动进行辨识，并保证在选择较小 seq<hk>时所带来 CPU 利用率提高这个约束条件下，整个系统任务可调度，也就是满足 CPU 高利用率，同时任务低丢失率。这两方面将在以下几节中给出。

3.4.3.2 Qoc 调度策略仿真

仿真环境选择 Matab 工具箱 TrueTime，系统共有三个控制任务，分别控制一个直流伺服电机，采用 PID 控制，比例，微分，积分系数分别为 K,T_d,T_i，微分增益为 N，对象和控制器如下：

被控对象：
$$G(s)=\frac{1000}{s(s+1)} \tag{3.16}$$

PID 控制器：
$$\begin{aligned} P(t) &= K(r(t)-y(t)); \\ D(t) &= a_d D(t-h)+b_d(y(t-h)-y(t)); \\ I(t) &= I(t-h)+a_i(r-y(t)) \end{aligned} \tag{3.17}$$

其中
$$a_d=\frac{T_d}{Nh+T_d}, b_d=\frac{NKT_d}{Nh+T_d}, a_i=\frac{Kh}{T_i} \tag{3.18}$$

三个控制任务周期分别为 10ms，6ms，18ms，任务执行时间都为 2ms。K=0.96，T_i=0.12，T_d=0.049，N=10，h_i 分别为任务周期。考虑最坏情况，三个控制对象都是受到外加扰动。采用 RM 调度算法，同时对三个控制对象加入阶跃为 5 的扰动，得到结果如图所示：

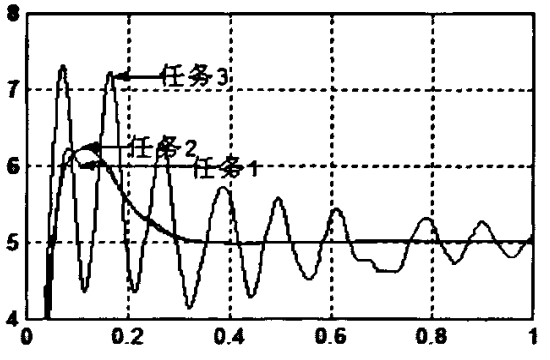

图 3.13 固定时限下被控对象在扰动下响应

表 3.4 恒值序列对应的 IAE 和 Qoc

H_k(ms)	IAE	Qoc
6	0. 401	1. 000
10	0. 434	0. 906
18	0. 750	0. 000

可以看出，任务 2 的动态性能最好，任务 3 动态性能最差，这和表 3.4 一致。根据 RM 调度规则：当前 CPU 利用率为：U=2/6+2/10+2/18=0.63，由于系统还存在非控制任务，考虑到系统裕量，控制任务 CPU 利用率应当小于等于 70。这里设 h_{min}=6, h_{max}=18 。同时，因为任务 2 的 Qoc 指数接近 1，这里简化设计，选择 seq<h_k>,h_k=12, 200ms 后恢复至原有周期。在扰动时 CPU 利用率 U=2/6+2/10+2/12=0.7。系统响应如图 3.14 所示，在改变周期时控制任务采样如图 3.15 所示：

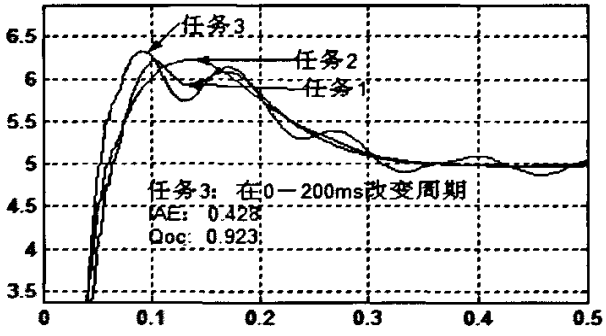

图 3.14 改变任务 3 采样周期后系统响应图

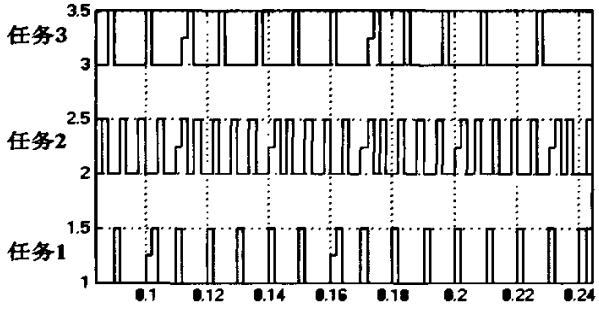

图 3.15 周期改变时控制任务采样

显然在扰动到来时改变任务 2 周期后，动态性能获得显著提高。

3.4.3.3　延迟和抖动的补偿设计

在实际系统应用中，不可避免碰到计算延迟和周期抖动的问题，为了方便计算延迟和抖动时间，这里使用了任务调度宏周期 P（所有任务周期的最小公倍数），即$P=LCM(T_1, T_2, \ldots T_n)$,这里$T_i$代表第i个任务的周期，$i<=N$。当回路任务数和调度算法固定时，所有任务在一个宏周期内的调度队列是固定的，因此每一个任务的延迟和抖动也是固定值。通过离线分析，建立每个任务在一个周期内的抖动和延迟列表$(h_1,h_2,\ldots h_n,t_1,t_2,\ldots t_n)$，每次对系统状态值进行估计，在控制算法上做出补偿。在控制计算时刻，控制量U应当以系统在输出时刻的采样值来计算，输出时间A_i和采样时间S_i由于存在一个L_i反馈延迟$(L_i=S_i-A_i)$, 如图3.16所示。所以需要估计从当前采样到输出执行时间段L_i上系统输出的变化量，以确定适当的控制输出。这里采用了一种改进的线形补偿算法如下：

(1) 采样 y_i,S_i,查列表 $h_{i-1},t_{i-1},t_i,y_{i-1},Y_{i-1}$

(2) $K_{i-1}=(y_i-Y_{i-1})/(h_i-t_{i-1})$ (3.19)

(3) $K_{i-2}=(Y_{i-1}-y_{i-1})/t_{i-1}$ (3.20)

(4) $K_i=K_{i-1}+(h_{i-1}-t_{i-1})/t_{i-1}*(K_{i-1}-K_{i-2})$ (3.21)

(5) $Y_i=y_i+t_i*K_i$ (3.22)

(6) $e_i= r-Y_i$ (3.23)

(7) 控制算法计算

(8) 数据更新

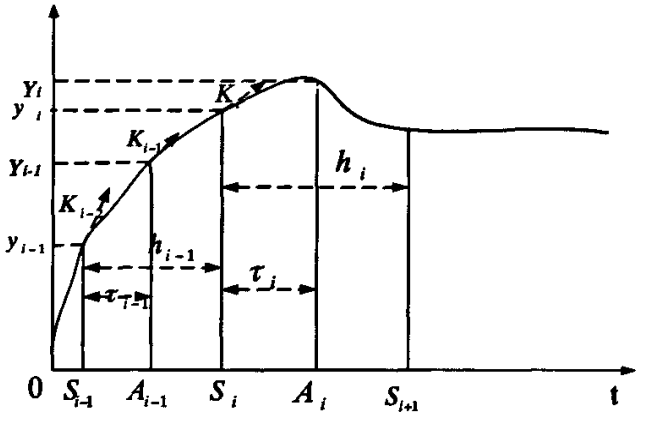

图 3.16 抖动和延迟计算示意图

3.4.3.4 实时控制与调度协同设计模型

在传统的控制系统结构上，加入了一个调度器，它用来调节采样周期，通过双反馈结构来达到控制与调度的同步。经过对误差的判断来分析是否扰动到来，根据控制对象所特有的采样周期选择，调整控制器中任务的采样周期，然后控制器依据新的采样周期重新计算控制器参数来弥补采样周期的变化。由于在实际应用中，对Qoc性能参数的直接计算比较困难，但作为一种评价手段，其调度策略是值得借鉴的。本节首先讨论基于控制的Qoc调度策略可调参数模型，然后基于此模型提出基于实时控制与调度协同设计模型。

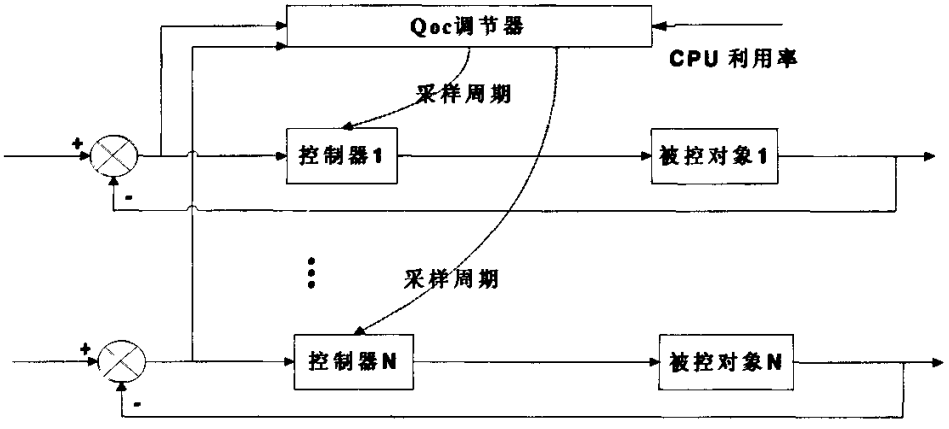

图 3.17 基于控制的 Qoc 调度可调参数模型

如图所示，在该系统中同时存在 N 组控制任务，每个闭环控制负责控制本身的系统稳定，而 Qoc 调节器则对闭环中控制器的控制参数调整，CPU 利用率 U 作为 Qoc 控制器的参考输入。在系统运行前，首先根据任务特性设计好可选择的采样周期 seq<h_k>序列，假设每个控制任务所需要的 CPU 负载是

$$U_i=C_i/T_i \tag{3.24}$$

其中 C_i 为任务执行时间，当运行时，由于采样周期改变 CPU 负载改变为$\triangle U_i$，如果满足

$$U+\triangle U_i<U_{th} \tag{3.25}$$

则称控制任务可调度，其中 U_{th} 即为系统可调度阈值。在每个闭环控制系统中，当扰动到来时，通过 Qoc 调节器改变其采样周期，相应地控制器参数发生变化以此来补偿采样周期变化。

Qoc 调节器算法如下：

(1) 当误差$|e|>e_{max}$时，被控系统处于过渡时期，请求 Qoc 调节器进行采样周期调整。当误差$|e|<e_{max}$时，系统处于稳定状态，转到(4)。

(2) Qoc 调节器对当前任务可调度性进行判断，如果不可调度，则跳到(5)；否则转到(3)。

(3) 选择合适的采样周期 seq<h_k>序列，请求控制器重新计算参数 。

(4) 如果采样周期序列 seq<h_k>改变过，则请求 Qoc 调节器重新调整采样周期调整，并通知控制器恢复原参数；否则直接退出。

(5) 退出。

然而，由于以上算法并没有充分考虑到系统整体 CPU 利用率，而且采用离线选择的 seq<h_k>在应对可调度判断时处理不够灵活；CPU 利用率受到很大的限制，使得并非每个控制任务在面临扰动到来时都能采用 Qoc 调度策略来改进其控制性能；CPU 利用率的不确定性以及利用率变化的不稳定性等等都是上述模型存在的弊端。

对基于控制的 Qoc 调度可调参数模型进行修改，采用反馈控制任务调度框

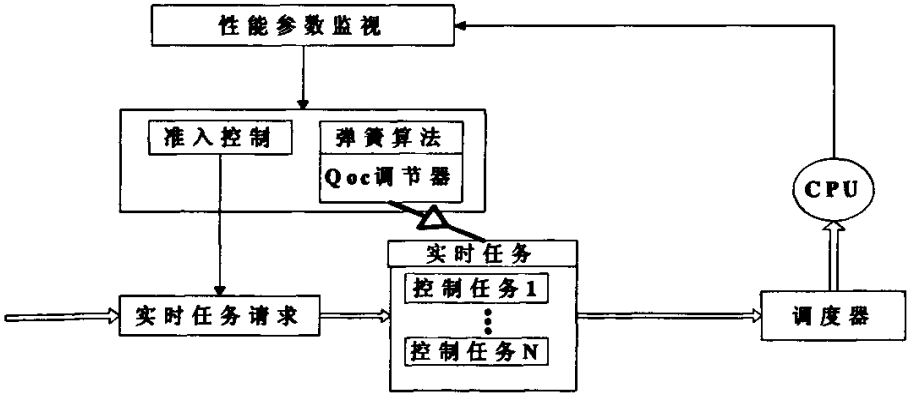

图 3.18 实时控制与调度协同设计模型

架在其模型上做出改进，如图 3.18 所示。该模型由性能参数监视模块，准入模块、调度模块、Qoc 调节器、弹簧算法模块等组成。准入模块负责控制进入系统的实时任务数，通过调整实时任务的流量对 CPU 的负载进行控制，可以防止瞬间超载。准入模块根据监视模块提供的系统负载值对实时任务进行准入控制。弹簧算法给每个控制任务分配利用率变化带宽，并计算出采样周期序列 seq<h_k>，通过 Qoc 调节器调节控制任务的参数，修改各个控制任务中的控制器

参数，准确反映系统中实时任务的负载，以此达到系统处理尽可能多实时任务的目的。调度模块负责确定任务执行的先后顺序，其中主要使用具有较高效率的 EDF 算法进行调度。

控制任务记为 $J_i=(S_i, C_i, D_i, T_{imin}, T_{imax})$，其中 S_i 为任务 i 的到来时间，C_i 为任务的执行时间，D_i 为截止期限，T_i 为周期。这里提供的任务执行时间和周期的最大值 T_{imax}、最小值 T_{imax}，可用于弹簧算法在计算实时系统参数时的参考值、初始值以及实际参数值的参考范围。在系统运行时，性能监视模块将对实时任务的执行时间和周期进行估计和计算。当扰动来临时，根据 Qoc 指标，弹簧算法重新计算控制任务采样周期，Qoc 调节器在获得较准确值的基础上对实时任务的参数进行调整和控制。这样做的最终目的是保证实时系统满足可调度性条件，以防止任务错过截止期限，同时在允许的范围内调度尽可能多的实时任务投入运行。

准入模块主要起到了一个阀门的作用，当系统负载无法满足当前实时任务需求时候，准入模块就截止新到来的任务。准入模块主要根据当前 CPU 利用率与设定值的差值进行准入工作。性能监视模块主要获取当前系统负载 $U_{act}(t)$，实时任务任务周期 $T_i(t)$。

设对每个控制任务 j 分配 N 级 Qoc(N=2),表示为 Qocij(i=0,1)，根据 Qoc 调度策略，Qoc 值越大，相对于控制性能愈佳。当被控系统扰动来到时，向 Qoc 调节器申请提升 Qoc 等级；在满足可调度约束下，弹簧算法依据提升的 Qoc 等级重新分配任务的利用率 U_i，并相应计算出采样周期序列 $seq<h_k>$。当系统进入稳态时候，任务周期恢复到扰动前的值，同时 Qoc 降级。这样做的目的是当多于一个被控系统受到外界扰动时候，系统能够合理分配 CPU 剩余利用率，同时也允许其他任务能进入系统中。

弹簧算法[58]的思想是调整实时系统中每个任务的负载（U=C/P）来改变系统的负载，将系统负载类比成多个串联在一起的弹簧系统，而任务的负载就是单个的弹簧。对系统的负载作调整相当于对弹簧系统进行拉伸和压缩，具体到单个任务的负载变化要参考其弹性系数。由于任务的负载由任务的执行时间和周期决定，所以任务的弹性系数主要包含执行时间和周期这两项。前面已经定义了实时任务的描述法：$J_i=(S_i, C_i, D_i, T_{imin}, T_{imax})$，其中 T_{imin}、T_{imax} 就决定了任务的弹性范围。可以看到，任务负载 U 的取值范围是 $[U_{imin}=C_i/T_{imax}, U_{imax}=C_i/T_{imin}]$，所以任务的负载可以在这个范围内进行调整。

通过对弹簧算法的扩展，首先计算 CPU 富余的利用率△U，如式 3.26 所示。其中 Us 表示为 CPU 利用率设定值，U_{iold} 是实时任务 i 调节前的负载值，U_{inew} 是调节后的负载值。E_i 是任务的弹性系数，由任务的实时特点确定，E_v 是所有可调节任务的弹性系数之和，计算公式如 3.29。E_i 值根据任务的 Qoc 值确定，和任务的控制性能有关。这里利用 Qoc 的加权值设置任务的弹性系数,如式 3.28。

$$\triangle U=U_s-U_{act} \tag{3.26}$$

$$U_{inew}=U_{iold}+\triangle U*(E_i/E_v) \tag{3.27}$$

$$E_i=w_i*Qoc_{ij} \tag{3.28}$$

$$E_v=\sum E_i \tag{3.29}$$

式 3.28 中 w_i 为权值，根据控制任务重要性影响。由于一般在控制任务中，运行时间是固定不变的，因此可以根据获取得到的利用率求得任务的采样周期如式 3.30 所示。

$$T_i=C_i/U_{inew} \tag{3.30}$$

当 T_i 超出[T_{imin}，T_{imax}]范围内时，固定任务采样周期为任务周期的上下限，同时重新计算 U_{inew}，按式(3.31)求得。

$$U_{inew}=C_i/T_i \tag{3.31}$$

在上述模型中，为了避免造成计算误差和破坏任务集的可调度性，提出下面几项改进：

1) 根据任务重要性区别，选择一定比例的任务，将其设为不可调节任务，从而保证这些任务的正确执行，方法就是调整这些任务的弹性范围，也就是 Qoc 的权值，使得较为重要的任务获得更多 CPU 带宽。
2) 由于可能出现部分未知扰动，引起系统负载超越设定值，一般在设置 Us 时留有一定余量，这样能保证系统不会出现较大的丢失率。
3) 在调节的时机的选择上，新到来的任务受到准入模块控制，当在可调度条件满足下，进入就绪队列，否则阻塞。

3.4.4 仿真

使用 matlab 中 TrueTime 工具箱仿真。假设系统同时具有三个控制任务，分别控制一个直流伺服电机，采用 PID 控制，比例，微分，积分系数分别为 K，T_d，T_i，微分增益为 N；对象和控制器如式 3.16、3.17、3.18 所示。其中 K=0.96，

T_i=0.12，T_d=0.049，N=10，h_i 分别为任务周期。三个控制任务周期范围分别为[8ms~ 10ms]、[4ms~ 6ms]、[10ms ~16ms]，任务执行时间都为 2ms。采用 EDF 调度算法，则每个控制任务 CPU 负载范围为[20% ~ 25%]、[33.3% ~ 50%]、[12.5%~ 20%]。整个控制系统的 CPU 负载范围为[65.8% ~95%],满足 EDF 调度算法，并设置 CPU 负载利用率设定值 U_s=0.90。

在系统中为每个任务的设置 2 级 Qoc。由于控制对象都一致，这里简化设置，其中控制任务 1 的 Qoc 值为[0.85， 0.9]，同理任务 2 和任务 3 分别为[0.9，0.95]，[0.9，0.95]；同时权值设为 w_1=w_2=w_3=1。系统方框图如图 3.19 所示：

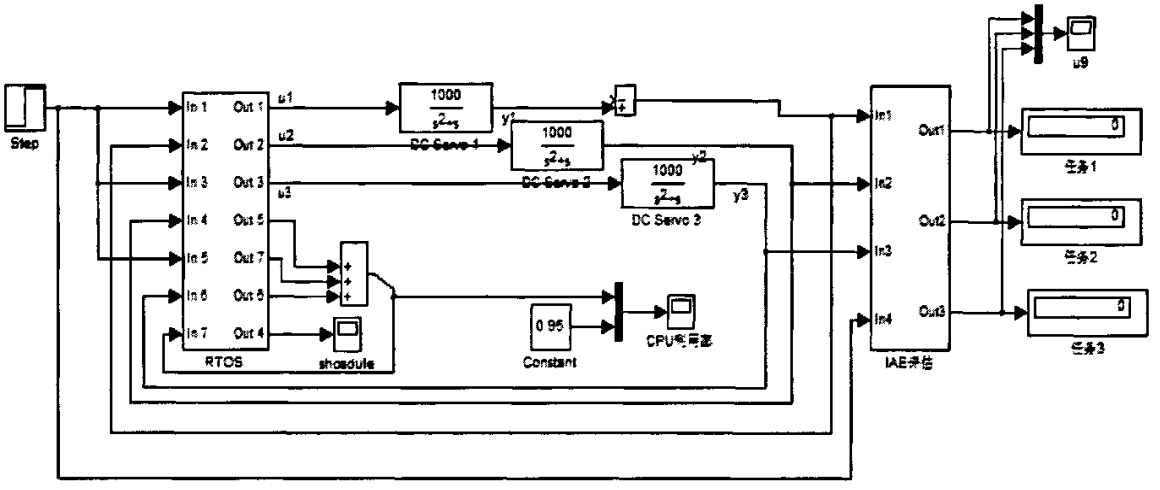

图 3.19 控制与任务调度协同设计 matlab 仿真方框图

考虑最坏情况，同时对三个任务加入幅值为 5 的阶跃扰动，仿真时间为 1s，获得 CPU 利用率仿真图如图 3.20 所示。

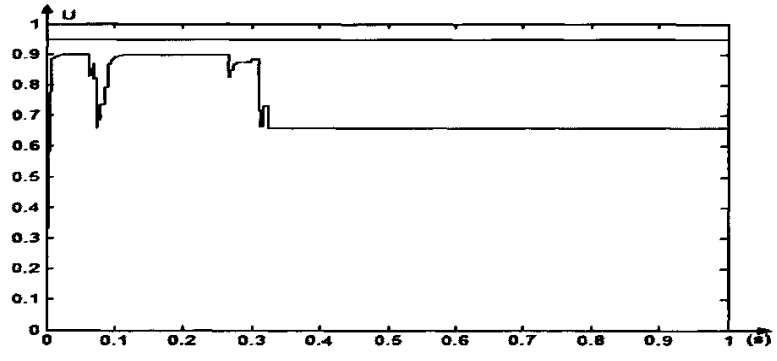

图 3.20 U_s=0.9 时 CPU 利用率仿真图

当 3 个控制任务在扰动到来时候，迅速提高 Qoc 等级，CPU 负载迅速整大，达到 90%左右，这时主要受到来自外在的扰动；在所有控制任务趋于稳定时候，CPU 利用率降低，最后稳定在 65.8%左右。

注意到 U_s=0.9，而控制任务所有利用率峰值为 95%，U_s 的设定限制了部分 CPU 利用率带宽，当 Us=0.95 时，CPU 利用率如图 3.21 所示。

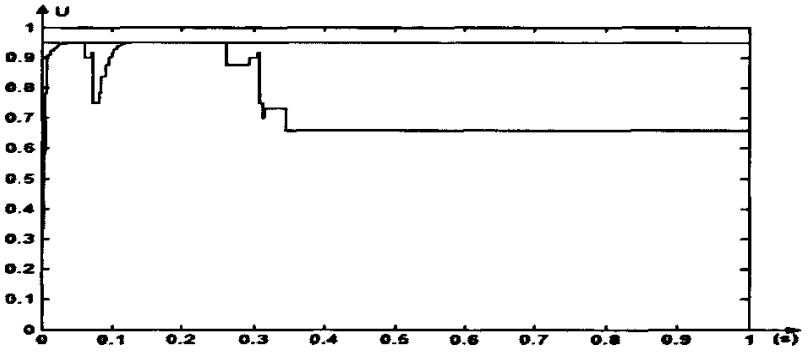

图 3.21 U_s=0.95 时 CPU 利用率仿真图

仿真结果显示，虽然控制任务在面临外在扰动时候能通过灵活的 Qoc 调度策略，并根据系统带宽使用弹簧算法进行动态调整采样周期，但在仿真中可以看到，CPU 负载在初期增加后，有一段时间处于利用率下降阶段，这主要是判断是否进入扰动只是采用阈值逻辑门限方法而造成的，可以通过仿人智能等先进智能算法进行改进。

3.5 反馈控制混合任务调度研究

本节主要从反馈控制这个角度对混合任务调度进行研究，基于反馈控制框架重新构建混合任务调度器，把反馈控制应用到混合实时任务调度中去。使得非周期软实时任务的截止期丢失率能够控制在期望的范围之内，同时 CPU 利用率也稳定在期望值附近。

3.5.1 反馈控制混合任务调度概括

在混合任务调度中，系统包含硬实时与软实时任务，其中软实时任务具有

固定的截止期，硬实时任务是提前确定的，而软实时任务是未知的。硬实时任务必须保证截至期，以此满足系统的硬性能要求；对软实时任务而言，给出如下假设：

▲每个软实时任务 T_i 具有 N 个 Qos 等级(N=1),表示为 $T_{ij}(0=j=N-1)$;

▲任务 T_i 的不同服务等级的实例 T_{ij} 要求不同的执行时间 $C_{i,j}$，有 $C_{i,j}>C_{i,j+1}$;

▲任务 T_i 的不同服务等级的实例 T_{ij} 具有相同的相对截止期 D_i。

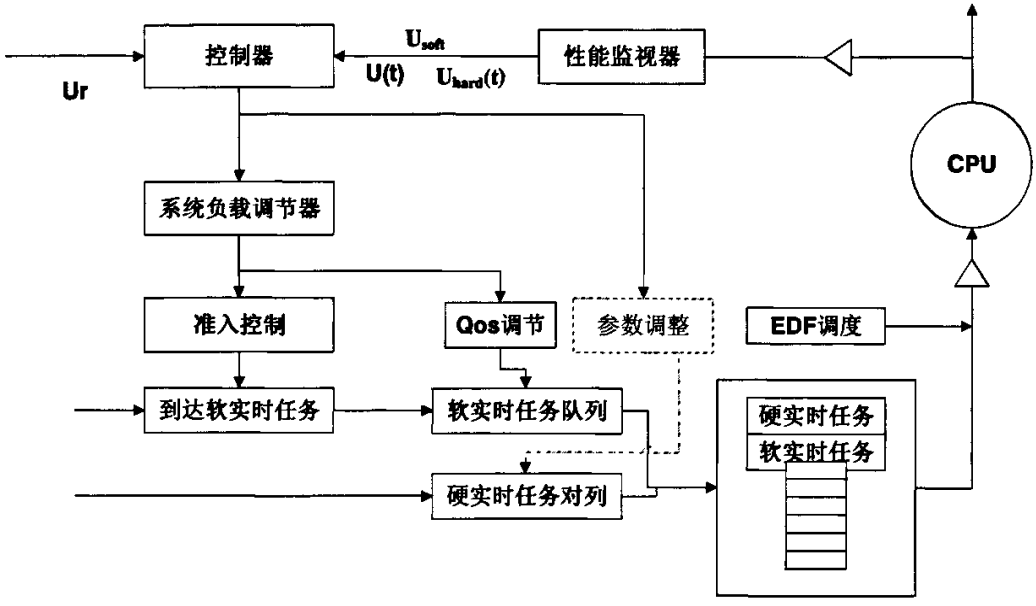

图 3.22 反馈控制混合任务调度模型框架

3.5.2 反馈控制混合任务调度模型算法与框架

在反馈控制混合任务调度系统中，由于同时具有硬实时任务与非周期的软实时任务，截止期丢失率是实时任务调度系统中的重要指标，而丢失率很大程度上依赖于系统负载，即 CPU 利用率，因此选择 CPU 利用率 U(t)做为被控变量。混合任务调度的控制目标是在满足硬实时任务同时并尽量满足软实时任务。选取任务估计的软实时任务 CPU 利用率余量$\triangle U_{soft}$作为控制变量，对准入控制与 Qos 调节模块进行控制。而控制器将采用水箱模型算法对利用率以及丢失率进行控制如图 3.22 所示，该图给出了该模型的一个体系结构图。

(1) 任务处理器

主要由软实时任务队列、硬实时任务队列、EDF 调度器以及 EDF 任务队列组成。EDF 任务队列是根据系统当前存在的任务，包括硬实时与软实时任务重新进行按照截止期升序排列，以便于 EDF 调度器进行调度。框架中的反馈控制主要控制与调节软实时任务的负载，以及硬实时任务中可调任务的部分参数。

(2) 准入控制与 Qos 调节

准入控制机制主要用于控制进入系统的软实时任务的负载，当一个新软实时任务 T_i 到达，准入控制器决定这个任务能否被接受。假如当前系统内的软实时任务估计 CPU 利用率 $U_{soft}(t)$，如果任务 T_i 的服务等级 j 满足式(3.32)

$$U_{soft}(t)+u_{ij}=U_{ss}, \tag{3.32}$$

在本模型中，采用式 3.32 的改变形式如式 3.33 所示：

$$u_{ij}=U_{ss}-U_{soft}=\triangle U_{soft} \tag{3.33}$$

其中 U_{ss} 表示分配给软实时任务的 CPU 带宽阈值，u_{ij} 表示 T_{ij} 所要求的 CPU 利用率。当满足上式时，则 T_i 能够以服务等级 j 被接受。如果任务 T_i 以最高服务等级不能够被接受，Qos 调节机制能够调整接受任务的服务等级来改变任务所要求的 CPU 利用率。如果 T_i 以最低的服务等级仍不能满足准入控制需要，则被拒绝，进入等待任务队列。

而对于在系统中已经运行的任务中，如果出现过载时，Qos 调节机制调整任务的服务等级来改变任务所要求的 CPU 利用率。当如果任务以最低的服务等级仍然不能满足时，则利用准入控制使该任务进入等待任务队列。当系统相对空闲时，任务从等待任务出列，以新到达任务身份出现。

计算系统当前软实时的估计 CPU 利用率,采用式 3.34 的计算方法，硬实时任务的计算方法也如此。其中任务队列为 Q，任务执行时间为 C_i。

$$U_{soft}(t)=\sum_{T_i\in Q}\frac{C_i}{D_i} \tag{3.34}$$

(3)性能监视器

性能监视主要负责获取当前系统任务的 CPU 利用率、截止期丢失率、硬实时任务的 CPU 负载以及丢失率。

(4) 系统负载调节器

负载调节器主要对控制器输出进行一定比例映射。这里直接把控制器输出传递给准入控制与 Qos 调节模块。

(5)控制器

采用预留 CPU 带宽以及可调硬实时任务 CPU 带宽阈值，以此保证硬实时

任务截止期零丢失率，同时又兼顾到软实时任务的到来与运行。控制器的设定值 U_r 是整个系统的任务负载设定值。控制器同时获取估计的硬实时任务负载 $U_{hard}(t)$。设硬实时任务 CPU 利用率阈值为 U_s，则相应的软实时任务 CPU 利用率阈值为 U_{ss},如式 3.35 所示。

$$U_{ss}=U_r-U_s \tag{3.35}$$

可以用这样一个水箱模型近似描述一个混合任务调度系统。接受的任务被看作流入 CPU 的液体，而完成的任务被看作流出 CPU 的液体，CPU 利用率被建模为一个高度为 1 的水箱，液体的高度 $U_{total}(t)$对应当前所有接受的任务的估计 CPU 利用率。如果水箱中液体高度为 1，表示估计的 CPU 利用率为 100%，而每个被接受的任务为水箱增加相应于其处理器要求数量的液体。水箱模型如图 3.23 所示。

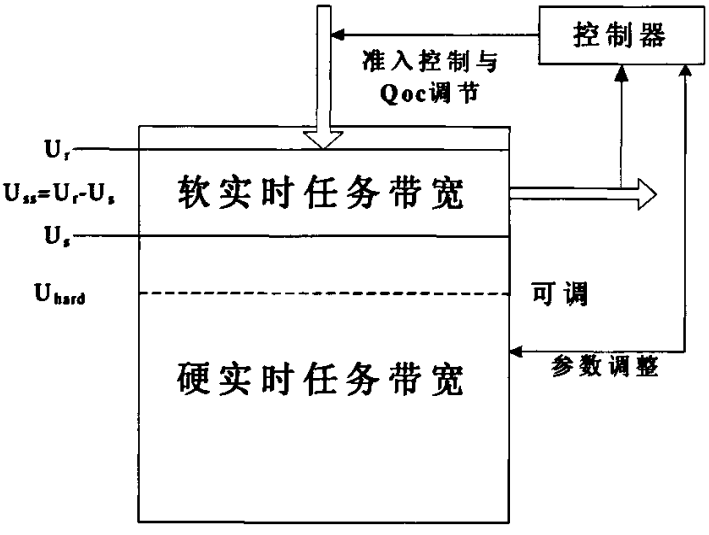

图 3.23 反馈控制混合任务调度水箱模型

该模型从本质上来说是通过调节 U_r 与 U_s 来达到混合任务调度的控制目标。如图，控制器有两路输出，一路是准入控制与 Qos 调节，另一路有两方面作用。一是对硬实时任务其中可调节任务参数进行调整，例如上节中所描述的控制任务在外在扰动来到时候改变采样周期进行快速恢复；二是通过改变 U_s 阈值，使得硬实时任务能够保证零丢失率。控制器的输入有 U_{hard}，U_{soft} 以及 U_{total}. U_{total} 与 U_{hard}、U_{soft} 有如下关系。

$$U_{total}=U_{hard}+U_{soft} \tag{3.36}$$

在理想情况下，只要水箱的液位低于 1，则调度器就能保证所有任务满足截止期。但由于事物的动态特性以及估计的误差，仍然会存在一定的截止期丢失率，因此设置 U_r=1。尽管这个模型只是现实任务调度的近似，而且 $U_{total}(t)$ 也只是实际需求的 CPU 利用率的估计，但是它非常适合调度系统中 CPU 利用率的概念。这里水箱模型被分割为两部分，分别对应软实时任务带宽与硬实时任务的带宽，并且这两部分相对固定，但是硬实时任务通常能够利用软实时任务的部分带宽。控制器的控制算法如下，从这几种情况考虑：

A. 如果系统处于正常运行状态下，即估计系统 CPU 利用率没有发生过载现象时:$U_{hard}<U_s$,$U_{soft}<U_{ss}$ 时，控制器计算式 3.37 得

$$\triangle U_{soft}= U_{ss}-U_{soft} \tag{3.37}$$

$\triangle U_{soft}$ 就是控制器输出给准入控制与 Qos 调节器的，对软实时任务进行准入与 Qos 性能调节。

B. 如果估计的硬实时任务发生过载时，即 $U_r>U_{hard}>U_s$，提升 U_s 至 U_{hard} 以上，保证硬实时任务的带宽，而 U_r 保持不变；如果估计的软实时任务负载未过载，则此时软实时任务负载带宽减小，直到 $U_s=U_r$ 时，软实时任务负载带宽为 0。如果估计的软实时任务负载过载，则$\triangle U_{soft}=U_{ss}-U_{soft}$ 并输出至准入控制与 Qos 调节器。如果硬实时任务利用率下降到原 U_s 值后，则硬负载阈值恢复到 U_s 上。

C. 如果估计的硬实时任务发生超载时，即 $U_{hard}>U_r$ 时，提升 U_r 到 CPU 利用率最大值，对于 EDF 调度为 1。此时，软实时任务带宽为 0，所有软实时任务都截止。直到估计的硬实时任务利用率下降到原 U_r 以下，这时软实时任务负载带宽才存在。当硬实时任务利用率下降到原 U_s 值后，硬负载阈值恢复到 U_s 上,U_r 也恢复到原值上。

D. 当估计的软实时任务负载过载时，而估计的硬实时任务未过载，并且总的利用率 U_{total} 未过载，即这时 $U_{soft}>U_{ss}$, $U_{total}<U_r$。那么$\triangle U_{soft}$=0,输出至准入控制与 Qos 调节器。

E. 在估计的硬实时任务负载未过载时，通过采用 3.4.3.4 一节中的 Qoc 调度策略与弹簧算法对可调硬实时任务的参数进行调整，提高系统整体性能。

F. 如果同时面对多个软实时任务到来时，采用从最低服务等级准入方式，让更多的任务能进入系统中。

3.5.3 仿真

在 3.4.3.5 这节的仿真模型基础上，通过加入反馈控制混合任务控制器算法以及 Qos 调节器与准入控制控制块，对系统进行仿真。模拟嵌入式系统，其中三个硬实时任务是使用 PID 控制三个直流伺服电机的控制任务；另一个硬实时任务是负责混合任务控制器以及 Qos 调节器计算。而软实时任务则通过离散事件模拟，根据 Qos 等级这里划分三级，每个软实时任务的截止期限固定为其周期。三个软实时任务的周期分别为[20ms,18ms,25ms],软实时任务的执行时间分别为[1.2ms 1ms 0.8ms],[1ms 0.8ms 0.6ms],[1.4ms 1.2ms 1ms]。调度算法采用 EDF 调度，仿真系统方框图如图 3.24 所示：

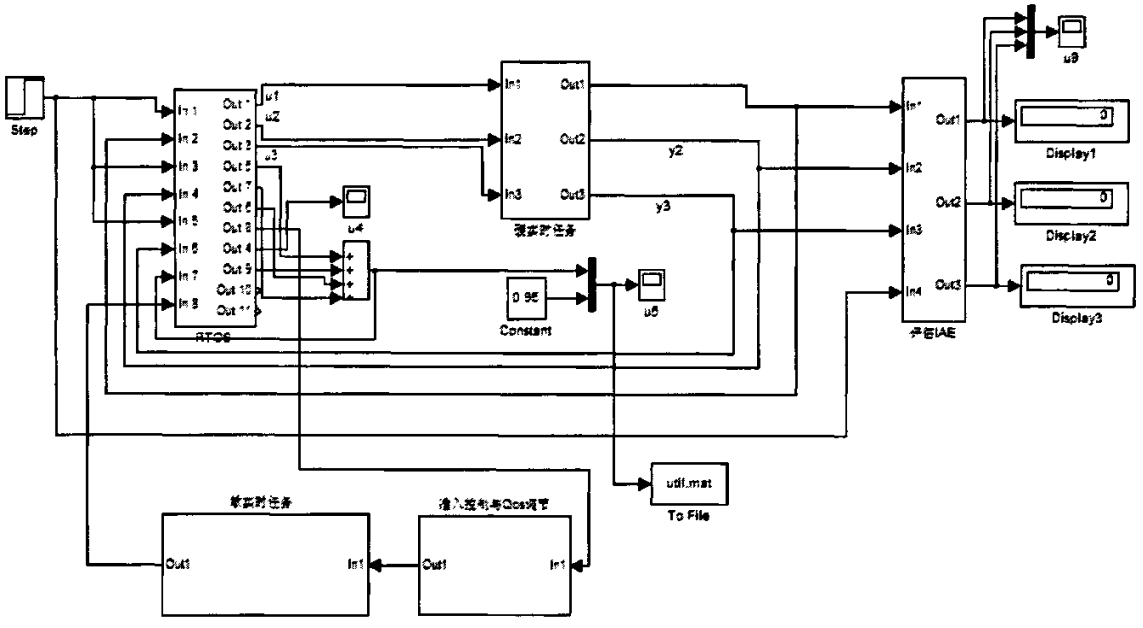

图 3.24 反馈控制混合任务调度仿真方框图

设定 U_r=0.95，U_s=0.8，对系统加入阶跃扰动，三个软实时任务同时到达系统。仿真如图 3.25 所示。图中 U_{total} 代表 CPU 利用率，而 U_{soft} 代表软实时任务负载。图 3.26 中可以明显看出硬实时任务在系统初期，突破 U_s 设定值，此时 U_s 设定值上升，当硬实时任务负载下降到原定阈值以下后，U_s 恢复到设定值

上。系统根据硬实时任务要求突破设定值，来满足硬截止期需要 。

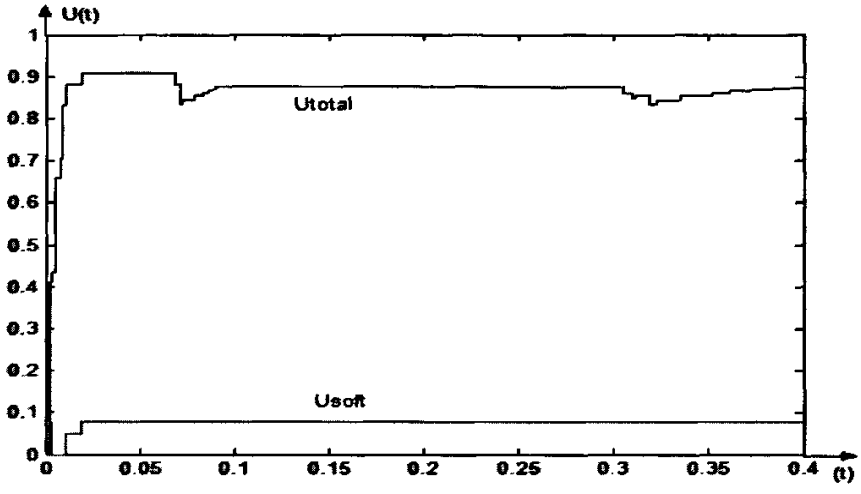

图 3.25 反馈控制混合调度 CPU 利用率

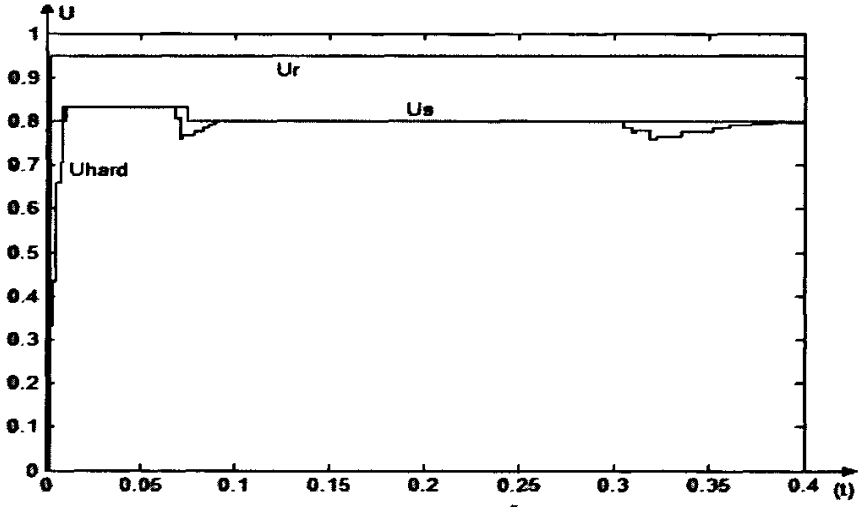

图 3.26　U_s、U_r、Uhard 变化图

在仿真中发现，U_r 设定值如果设得较高，而 U_s 设定比较低时，当硬实时任务负载要求过大时，系统会提升 U_s 值到达一个比较合理值。当 U_r 设得比较低时，系统会调整硬；实时任务阈值 U_s 超越 U_r 达到一个新的高峰，这时软实时任务负载此时的带宽不存在。当经过一段时间系统 U_s 才恢复到原值，这是该算

法不足之处，因此一般阈值选择 U_r 要偏高。为了保证一定的软实时任务负载的裕量，U_s 则相应地取得相对低一点。

3.6 本章小节

本章主要从实时系统的不确定性、不可预测性这个角度入手，针对实时系统中硬实时任务与软实时任务同时存在的情况，从几个方面研究并探讨了任务调度策略。

在第二章的经典调度基础上，通过引入反馈调度概念，并作为主线贯穿于本章。首先根据任务按照周期与非周期之分，对混合任务进行分析，介绍了混合任务调度算法，其中详细讨论一种基于空闲时间的混合任务调度算法，并提出混合任务调度的模型框架。在对反馈任务的调度研究中，根据 Chenyang Lu 和 John A.Stankovic 提出的反馈控制实时调度算法，对 PI 控制器详细分析并介绍了利用率、丢失率、利用率丢失率反馈调度算法，综合地分析了各调度算法优劣。通过对反馈调度概念的引入，针对控制与调度协同问题，以 Qoc 作为评价基础，给出了基于 Qoc 的调度策略，并提出基于实时控制与调度协同设计模型，使用弹簧算法根据控制任务的优先等级，对有限 CPU 利用率再分配，灵活地应对扰动到来对被控系统的影响。最后在混合任务调度方面，使用反馈控制，以反馈控制混合任务调度框架为基础，可调软硬实时任务负载带宽为出发点，准入与 Qos 为软实时任务负载调节杠杆，提出了反馈调度的水箱模型以及控制器的控制算法，仿真结果显示在合理分配系统 CPU 利用率阈值以及硬实时任务负载带宽情况下，该算法的应用在混合任务调度中在保证硬实时任务的零截止期丢失率的灵活特点。

第 4 章 基于嵌入式实时操作系统任务调度应用

4.1 振动系统监控校验保护仪研制项目概述

在“微小力测试平台”研究基础上，结合“振动系统监控校验保护仪”的项目，以本特利 3500 振动监控保护系统系统为蓝本，设计一套功能相同的框架式监测系统 9700。该系统主要用来对大型旋转机构进行监控与保护的，因为其在电厂的大量使用，又名汽轮机监测系统简称 TSI。它是一种可靠的能连续不断地测量汽轮机发电机转子和汽缸的机械工作参数的监控系统，可用于显示机组的运行状况，提供输出信号给记录仪；并在超过设定的运行极限时发出报警。另外，还能提供使汽机自动停机、传感器校验以及用于故障诊断的测量。

4.1.1 系统构成与分析

振动监控保护系统主要由软硬件系统两部分组成：框架式检测系统和上位计算机软件。由上位机对该系统进行组态设置，动态数据获取分析，状态监视，而框架式检测系统完成对电涡流位移、振动速度、加速度、转速等信号的放大，检波，DSP 处理，分析并且在系统发生故障时能够组态设置报警和输出控制等功能。

系统结构如图 4.1、4.2 所示，硬件系统主要分功能、电源、母板共以下 8 类卡件组成：

功能卡件：

(1) 位移/速度/加速度监测模块

(2) 转速(包括零转速)监测模块

(3) 键相器模块

(4) 框架接口模块

(5) 报警继电器输出模块

(6) 监测模块后板(信号调理电路)

电源卡件：　　两组冗余智能电源供应

母板卡件：　　电源、信号连通底板

其中位移/速度/加速度监测模块简称 42 振动模块，可以接受来自位移、速度、加速度传感器信号，通过对这些信号的处理完成各种不同的振动和位置测量，并将处理的信号与用户组态的报警值进行比较。转速监测模块和 42 振动模块类似，只是对转速信号进行监测。键相器模块用来为其他监视模块提供相位信号，该模块接受来自电涡流传感器或电磁式传感器信号，并转换成数字相位信号。框架接口模块(RIM)是系统与组态、显示和状态监测软件连接的主要接口，并能对系统提供自检功能。报警继电器模块提供四通道继电器输出，每个通道可单独编程以执行所需要的表决逻辑。

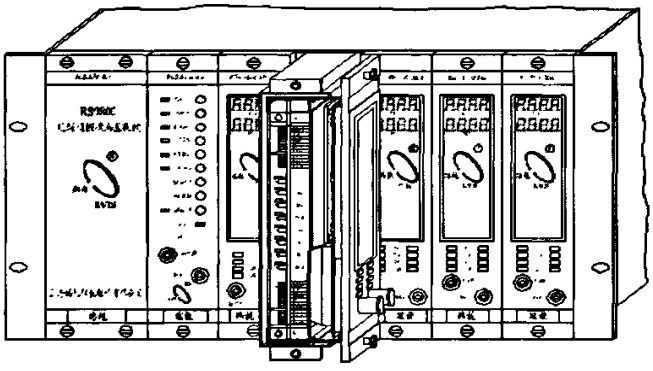

图 4.1 振动监控校验保护系统

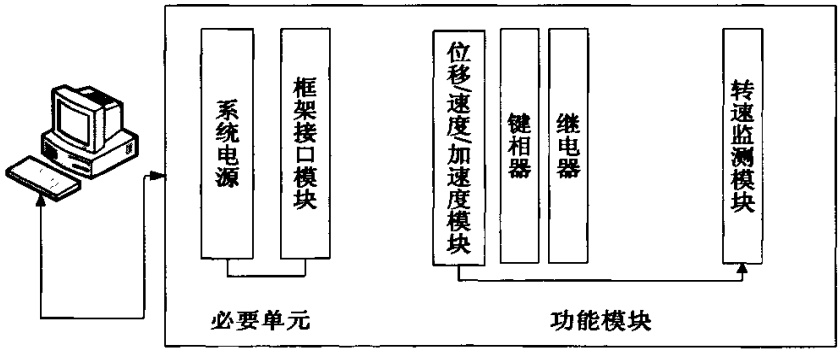

图 4.2 振动监控校验保护系统结构图

由于在该系统中最具代表性的是 42 振动模块以及框架接口模块，其他模块只是在其基础上删减功能，这里以 42 振动模块为例。

4.1.2 42 振动模块功能描述及结构

42 振动模块由数据采样、数据处理、数据输出、辅助调试等模块组成。数据采集 A/D 采用 14 位高速芯片，FPGA 负责采样数据。数据处理芯片采用 DSP+FPGA 模式， DSP 和 FPGA 间以 CAN 总线和双口 RAM 两种方式相连，其中的 FPGA 用于数据的采集、及其外围的管理和数据的处理。数据处理是此部分的核心，FPGA 负责对采样信号进行 FFT 变换，得到更加准确基频和倍频的测量数据。与框架接口模块的接口采用 CAN 通讯方式，同时系统保留 SCI 接口，可使用其他设备读取模块数据模块信息。数据输出中，测量数据采用 12 位 D/A、U/I 转换输出得到，控制数据以 CAN 方式送至通讯管理模块。辅助调试接口包括 LED、LCD，供调试时使用。DSP 代码保存在 FLASH 中，在系统上电时从 FLASH 载入程序存储空间，EEPROM 保存系统自检、状态、组态数据等信息。该硬件总体图如 4.3 所示，其功能列表如表 4.1。

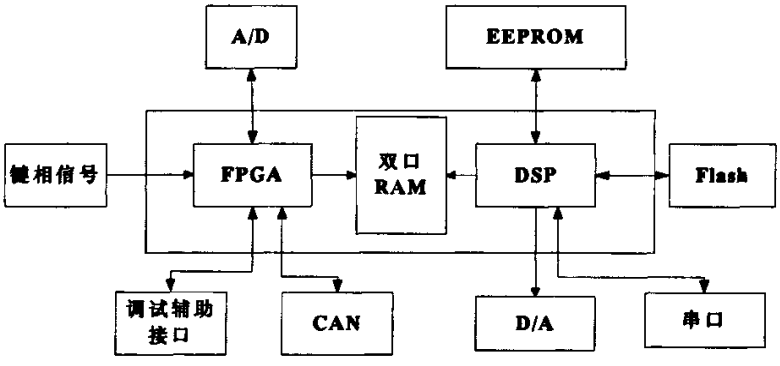

图 4.3 42 振动模块硬件总图

表 4.1 42 振动模块功能列表

功能名称	功能描述
CAN 通讯、SCI 通讯	处理解包打包，接受发送数据
A/D、D/A	A/D 采样，D/A 输出
状态监测控制	组态、状态分析、报警
存储控制	EEPROM 数据存储
系统初始化、重启动	FLASH 数据载入
外围接口操作	LCD、LED 等

4.1.3 系统软件设计

系统中需要完成的工作有：CAN 通讯，SCI 通讯，A/D 与 D/A，状态监测控制，存储控制，系统初始化、重启动，外围接口操作等。这些任务对时间的要求各不相同，例如，状态监测控制、A/D 与 D/A 是系统主要完成的周期任务，而且任务到达频率较高；CAN 通讯与 SCI 通讯属于实时的任务，且它们的到达时间随机，具有截止期限要求，可以视为软实时任务；存储控制由于其到达时间的随机性，属于非周期任务；外围接口操作(液晶显示系统运行状态、响应操作)可以视为非实时任务，在系统的空闲时间执行。采用通常的前后台系统无法保证实时性，因此使用了占先式实时内核 μC/OS-II，它可以按照优先级的高低来调度多个任务，通过对内核改造，使得任务级响应时间得以最优化。选用 μC/OS-II 的原因是：它是源码公开的实时内核，采用优先级占先式调度，使用简单，内核精简可配置，易于更改，容易移植到各种微处理器上。

状态监测控制模块又可以分成几个相应独立的小模块：组态、组态分析、状态监测、报警模块。组态模块主要对组态软件下载的用户组态数据进行组合配置；组态分析模块用来分析组态逻辑意义，并记录于实时逻辑组态数据中；状态监测模块主要用于对系统自身硬件系统自检以及采样得到的各参量状态数据。对 A/D 与 D/A 操作采用定时器中断，获取来自双口 RAM 中存放的 FPGA 的 A/D 采样值，同时 D/A 根据组态要求输出数值。CAN 和 SCI 则由外部中断触发，并负责系统对外的通讯。

开始为了使设计简化，系统的 6 个任务，按照紧急程度分配优先级，各个任务由 μC/OS-II 根据优先级来进行调度。软件用 C 语言编写，主函数中完成系统硬件、内核、任务初始化后，启动实时内核，再动态地创建 6 个任务，分配不同的优先级，由系统自动调度。任务间的通讯通过邮箱和消息来实现，软件状态转换图见图 4.4，主代码清单如下：

```
 void main(void)
{
    HardInit();                                                    //硬件初始化
    OSInit();                                                      //内核初始化
    OSTaskCreate(TaskInit, (void *)0, (void *)TaskInitStk, 0);     //任务初始化
    OSStart();                                                     //启动实时内核
}
void TaskInit(void *data)
```

```
{
  RTI 启动代码;
  OSStatInit();                                          //系统统计任务初始化
  OSTaskCreate(SysChk, (void *)0, (void *)SysChkStk, 1);      //状态监测控制，优先级 1
  OSTaskCreate(SCanCom, (void *)0, (void *) SCanComStk, 6); //CAN、串行通讯，优先级 6
  OSTaskCreate(Storage, (void *)0, (void *) StrorageStk, 8);          //存储控制，优先级 8
  OSTaskCreate(AffInterface, (void *)0, (void *) AffInterfaceStk, 15);  //外围接口，优先级 15
  系统监控代码;
}
```

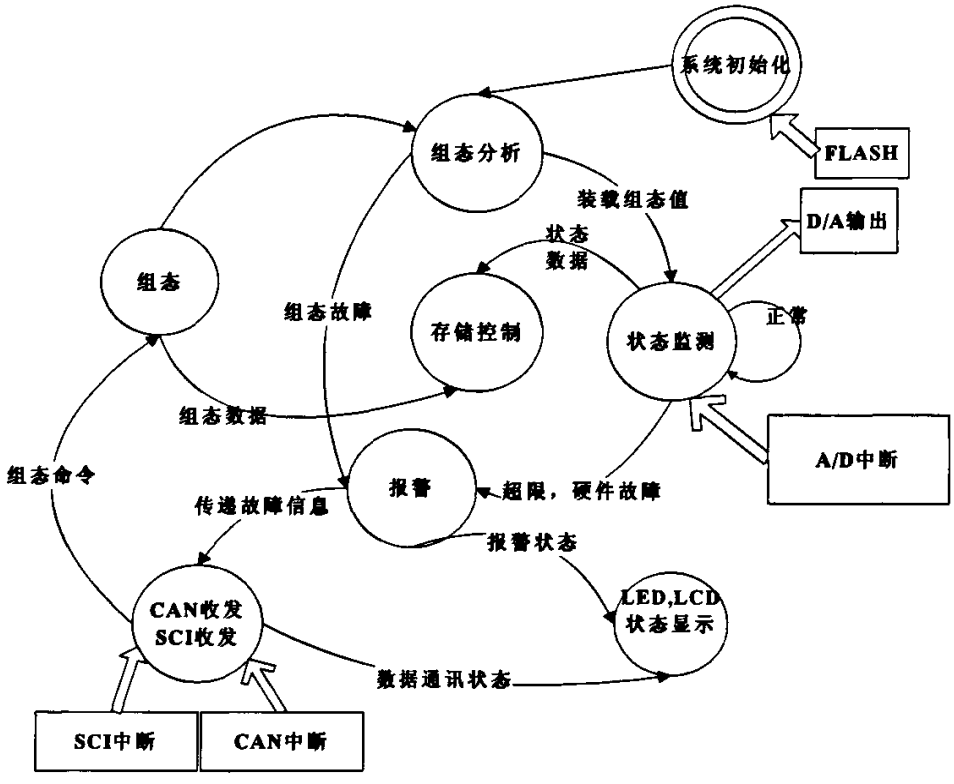

图 4.4 42 振动模块系统软件转换图

```
void SysCheckProcess(void * argv)
  {
  OSTaskCreate(AlertProcess,(void*)0 (void *)&AlertStk, 2);          //报警，优先级 2
  OSTaskCreate(ConfigProcess,(void*)0 (void *)&ConfigStk, 3);     //组态、分析，优先级 3
          ...
```

}

上述方法是根据任务的紧急程度定义优先级顺序，完全依靠 μC/OS-II 的调度函数，根据优先级进行占先式调度。然而在实际应用中，当串口上请求数据时，并且系统正在以比较高频率采样时，容易造成偶然间断性采样数据不准，以及没有额外的资源来执行优先级较低的其他任务，比如 LCD 显示不实时等等，也就是说系统陷入了“过载”。

4.2 μC/OS-II RTOS 在系统中的移植

4.2.1 μC/OS-II 系统研究

μC/OS-II 是一个基于优先级的多任务可占先式的实时内核，它提供了最基本的任务管理、时间管理、任务之间的通信与同步、内存管理功能。最多支持 64 个任务。多任务运行的实现是靠 CPU 在许多任务之间切换、调度，多任务运行使 CPU 的利用率得到最大的发挥，并使应用程序模块化，寄存器与栈空间划分见图 4.5。

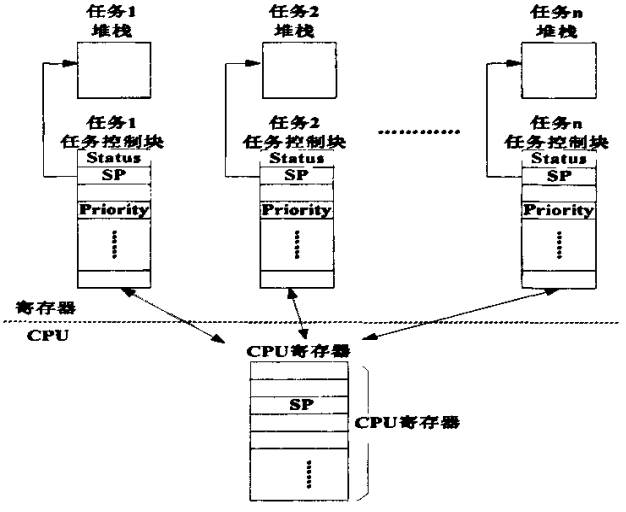

图 4.5 μC/OS-II 中多任务寄存器与栈空间

每个任务都有一个任务控制块，它是一个数据结构，在任务建立的时候 OS_TCB 将被赋值，当任务的 CPU 控制权被剥夺时，μC/OS-II 用它来保存该任务的状态，当任务重新得到 CPU 控制权时，任务控制块能确保该任务从被中断的那一点开始继续执行。每个任务控制块包含操作系统所需的关于任务的所有信息。凡涉及任务调度、中断处理、任务通信和同步等操作，都需要读取和修改系统中的任务控制块。所有的任务控制块都是放在任务控制块列表数组 OSTCBTbl[ ]中的，在 μC/OS-II 初始化的时候，所有任务控制块被链接成空任务控制块的单向链表。当任务一旦建立，空控制块指针 OSTCBFreeList 指向的任务控制块便赋给了该任务，然后 OSTCBFreeList 的值调整为指向下一个空的任务控制块，一旦任务被删除，它的任务控制块就会回到空任务控制块链表中。

每个任务一般都是一个无限的循环，每个任务都处于以下 5 种状态之一：休眠态、就绪态、运行态、挂起态(等待某一事件发生)和被中断态。休眠态相当于该任务驻留在内存中，但并不被多任务内核所调度；就绪态意味着该任务已经准备好，可以运行了，但由于该任务的优先级比正在运行的任务的优先级低，还暂时不能运行；运行态的任务是指该任务掌握了 CPU 的控制权，正在运行中；挂起态也可以叫做等待事件状态，指该任务在等待某一事件的发生(例如等待某个外设的 I/O 操作，等待定时脉冲的到来或等待超时信号的到来以结束目前的等待)；被中断指发生中断时，CPU 提供相应的中断服务，原来正在运行的任务暂时不能运行，就进入了被中断状态。图 4.6 表示 μC/OS-II 中一些函数提供的服务，这些函数使任务从一种状态变到另一种状态。

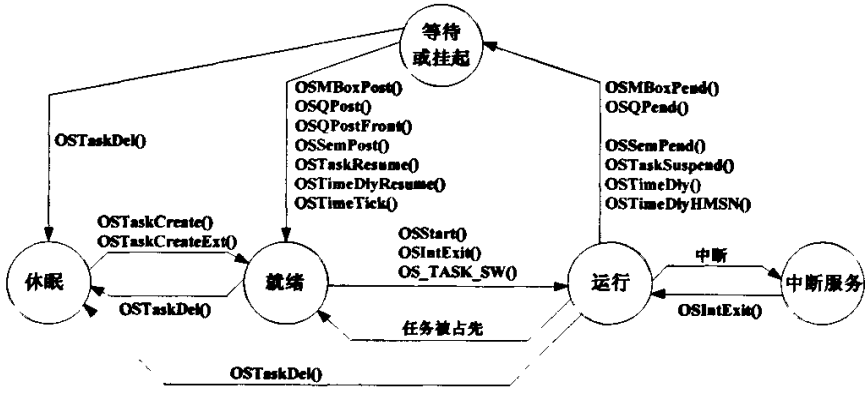

图 4.6 μC/OS-II 中任务状态及其切换

任何时刻只能有一个任务处于运行状态，当多任务内核决定运行另外的任

务时，它保存正在运行任务的当前状态，即 CPU 寄存器中的全部内容。入栈工作完成以后，将下一个要运行任务的当前状态从该任务的栈中重新装入 CPU 寄存器，并开始下一个任务的运行，这个过程称为任务切换，任务切换过程增加了应用程序的额外负荷。

每个任务的就绪态标志都放入就绪表中，就绪表中有两个变量 OSRdyGrp 和 OSRdyTbl[ ]。任务进入就绪态时，就绪表 OSRdyTbl[ ]中相应元素的相应位也置位。为了找到那个进入就绪态的优先级最高的任务，只需要查另外一张表(优先级判定表 OSUnMapTbl)，返回的字节就是该组任务中就绪态任务优先级最高的那个任务所在的位置。

μC/OS-II 总是运行进入就绪态任务中优先级最高的那一个，确定哪个优先级最高、下面轮到哪个任务运行的工作是由调度器(scheduler)完成的。任务的调度是由 OSSched( )完成的，中断级的调度是由另一个函数 OSIntExit( )完成的。

```
void OSSched (void)
{
    INT8U   y;
    OS_ENTER_CRITICAL( );
    If ( ( OSLockNesting | OSIntNesting ) == 0) {                          (1)
        y  =  OSUnMapTbl  [OSRdyGrp] ;                                     (2)
        OSPrioHighRdy = ( INT8U) ( (y << 3) + OSUnMapTbl [OSRdyTbl [y] ] ) ;   (2)
        If (OSPrioHigh != OSPrioCur) {                                     (3)
            OSTCBHighRdy = OSTCBPrioTbl [ OSPrioHighRdy] ;                 (4)
            OSCtxSwCtr++;                                                  (5)
            OS_TASK_SW( ) ;                                                (6)
        }
    }
}
```

(1) 如果在中断服务子程序中调用 OSSched( )，此时中断嵌套层数 OSIntNesting > 0，或者由于用户至少调用了一次给任务调度上锁函数 OSSchedLock( )，使 OSLockNesting > 0，则任务调度函数将退出，不做任何任务调度。

(2) 如果不是在中断服务子程序调用 OSSched( )，并且任务调度是允许的，

即没有上锁，则任务调度函数将找出那个进入就绪态且优先级最高的任务。

(3) OSSched( )检验这个优先级最高的任务是不是当前正在运行的任务，以此来避免不必要的任务调度。

(4) 为实现任务切换，OSTCBHighRdy 必须指向优先级最高的那个任务控制块 OS_TCB，这是通过以 OSPrioHighRdy 为下标的 OSTCBPrioTbl[ ] 数组中的那个元素赋给 OSTCBHighRdy 来实现的。

(5) 统计计数器 OSCtxSwCtr 加 1，以跟踪任务切换次数。

(6) 调用 OS_TASK_SW( )来完成实际上的任务切换。

任务切换很简单，由以下两步完成，将被挂起任务的微处理器寄存器内容推入堆栈，然后将较高优先级任务的寄存器从栈中恢复到 CPU 寄存器中。换言之，μC/OS-II 运行就绪态的任务所要做的一切，只是恢复所有的 CPU 寄存器内容并运行中断返回指令，为了做任务切换，运行 OS_TASK_SW( )，人为模仿了一次中断。多数微处理器提供软中断指令或者陷阱指令 TRAP 来实现上述操作。OSSched( )的所有代码都属于临界区代码，在寻找进入就绪态的优先级最高的任务的过程中，为防止中断服务子程序把一个或几个任务的就绪位置位，中断总是被关掉的。为缩短切换时间，OSSched( )全部代码可以用汇编语言写，但为了增加可读性、可移植性和将汇编语言代码最少化，OSSched( )是用 C 写的。

从 OSSched( )的代码可以看出，μC/OS-II 的调度相当简洁，效率也很高，其中的原因有以下两点：

- 如果调度的原则并非严格地按优先级，而是还要考虑其他的因素，那就不止是简单的找到优先级最高的就绪进程就能解决问题了。嵌入式实时操作系统通常都是严格地按优先级调度，而通用操作系统则一般还要考虑时间片以及累计运行时间的问题，所以通用操作系统就比较复杂。
- 如果优先级的使用并非唯一的，从而多个进程可以使用相同的优先级，那就还有个在具有相同优先级的就绪进程之间怎样调度的问题，这就使得调度的过程复杂化了，μC/OS-II 不允许不同的任务有相同的优先级。

4.2.2 μC/OS-II 在 TMS320LF2407A 上的移植

下图说明了 μC/OS-II 的结构以及与硬件平台的关系。

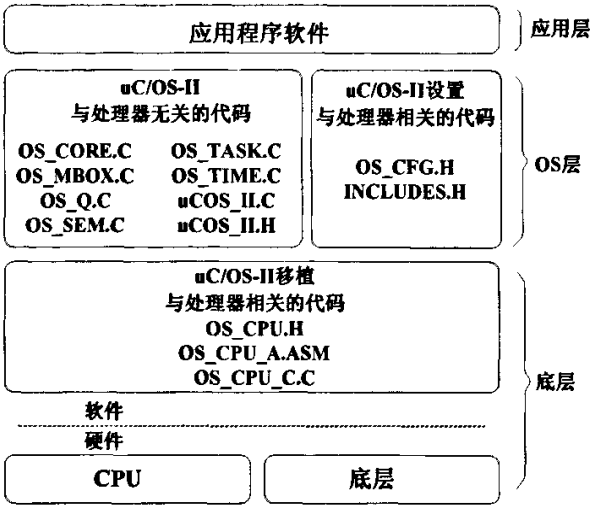

图 4.7 μC/OS-II 硬件和软件体系结构

(1) 从图 4.8 中可以看出，要将 μC/OS-II 移植到硬件平台上，只需设置和改写 4 个函数：配置 OS_CFG.H，根据硬件环境改写 OS_CPU.H、OS_CPU_A.ASM、OS_CPU_C.C。其中 OS_CFG.H 设置几个常数值；OS_CPU.H 声明 10 个数据类型并定义 3 个与中断有关的宏；OS_CPU_A.ASM 编写四个汇编语言函数(3 个有关任务切换时的堆栈调整，1 个有关定时)；OS_CPU_C.C 用 C 语言编写 6 个简单的函数(1 个堆栈初始化，5 个接口函数可编写也可为空)。

移植 μC/OS-II 的主要工作是声明与硬件相关的数据类型，定义与中断有关的宏定义，定义堆栈增长方向宏定义，编写堆栈初始化函数，Hook 接口函数，任务级上下文切换函数，中断级上下文切换函数以及系统时钟定时服务函数等，具体说明如下：

- 由于 μC/OS-II 考虑到通用性，因此在内核中使用了自定义数据类型，与编译器无关。在移植中应首先将其声明为编译器可识别的类型，例如：

```
typedef unsigned int     INT16U      /* 无符号 16 位整数 */
```

- μC/OS-II 在内核中通过禁止中断来保护临界区，因此需要在 C 语言中插入汇编代码，为了隐藏编译器提供的具体实现方法，μC/OS-II 采用宏定义的方法来访问中断操作，例如：

```
#define OS_ENTER_CRITICAL()  asm{"DINT"}
```

- μC/OS-II 中需要一个时钟资源来实现延时和期满功能。因此在 F2407 中，可以使用 RTI 中断来实现，时钟节拍 ISR 的原型如下：

```
void OSTickISR(void)
{
  保存上下文;
  OSIntEnter();
  OSTimeTick();
  OSIntExit();
  恢复上下文;
  中断返回;
}
```

- 在内核中，通过设置独立的任务堆栈保护上下文，来实现多任务的运行。因此了解 F2407 的堆栈结构很重要。F2407 本身具有 8 级深度硬件堆栈，在用 C 语言编程时，编译器自动维护一个软件堆栈，经过查阅 TI 的技术文档可知，软件堆栈的结构如图 4.8 所示，按照这样的结构，通过调整 AR1(SP)的值为每个任务配置独立的堆栈，就可以使得各个任务互不干扰，完全独享 DSP 的资源，实现多任务并发。TI 的 C 编译器在中断中按照图 4.9 所示的顺序将 DSP 的寄存器和硬件堆栈保存到软件堆栈中，因此在中断初始化函数 OSTaskStkInit()中按照这样的顺序编写，使得初始堆栈就像刚发生过中断并将所有的寄存器保存到堆栈中的情形一样。在任务级切换函数 OSCtxSw()中，首先应保存当前任务的上下文，然后调整 AR1(SP)，恢复新任务的上下文并执行中断返回指令。在中断级切换函数 OSIntCtxSw()中，首先要调整堆栈指针以去掉在调用 OSIntExit()和 OSIntCtxSw()过程中压入堆栈的多余内容。由于该函数是在中断服务程序中调用的，在进入中断后已经保存了被中断任务的上下文，所以只要再调整 AR1(SP)并恢复新任务即可。OSIntCtxSw()是 μC/OS-II 中唯一与编译器有关的函数，因为堆栈指针的调整必须是明确的，而这主要依赖编译器的编译方式，编译选项等。

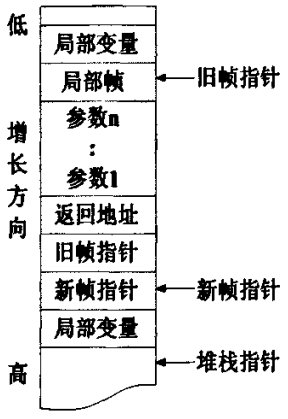

图 4.8 软件堆栈结构图

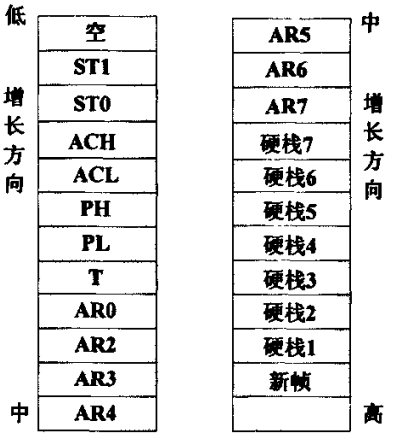

图 4.9 中断上下文入栈顺序

- F2407 支持多种中断方式，包括可屏蔽硬中断 INT1~INT6，不可屏蔽硬中断 RS、NMI，不可屏蔽软中断 INTR k 以及软件陷阱 TRAP。因此使用 INTR 31 软中断来调用 OSCtxSw(),使用 INT1 硬中断来调用 OSTickISR()；相应的中断向量表为：

```
        .global _c_int0
        .global _OSTickISR
        .global _OSCtxSw

        .sect  ".vectors"
RESET   B     _c_int0        ; 上电复位向量
  ; 硬件可屏蔽中断向量
INT1    B     _OSTickISR  ; 实时中断向量
...
  ;软件中断向量
INT31 B   _OSCtxSw    ; 任务级中断切换服务向量
        .end
```

- 内核移植完成之后，在用户程序中只需加入#include "uCOS_II.C",然后作为一个文件进行编译链接重定位即可。在用户程序中，应该按照如下的顺序启动实时内核：

```
void main (void)                    /* 主函数 */
{
  硬件初始化;
  OSInit();              /* 内核初始化 */
  调用 OSTaskCreate()创建初始化任务 TaskInit();
```

```
    OSStart();          /* 开始多任务调度 */
}
void TaskInit (void *data)
{
    硬件时钟初始化;
    创建用户任务 Task1~n;
    定时检查系统状态;
    定时复位看门狗;
}
```

4.2.3 μC/OS-II 任务调度算法改造

μC/OS-II 采用的是固定优先级的分配策略，其实现模块 OSTaskChangePrio()仅以新优先级替换掉旧优先级，功能简单。事实上，在实时系统中常常需要采取其他优先级分配方案，如截止期最早优先算法 EDF、速率单调算法 RM 等。截止期最早优先算法的实质是使截止期最早的任务优先级最高。这就需要将就绪队列中的任务按截止期排序并转换为与系统当前时间一致的形式。为此，在 OS_TCB 任务控制块与全局变量中增加表 4.2 所列的参数。

表 4.2 OS_TCB 任务控制块与全局变量中增加的参数

参数	功能描述
INT16U Deadline	加入 OS_TCB，记录任务的截止期，在任务调度时用当前系统时间 OSTime 和 Cstime 的差更新 Deadline
INT16U Cstime	加入 OS_TCB，记录任务调用时的系统时间
INT16U CsRec[64][3]	全局变量，记录每个优先级任务的调用时间、截止时间、优先级附加值（记录任务优先级的变化），任务创建或销毁时要根据其截止期分配合理的优先级并插入此数组，在任务发生调用时用数组来更新其相应项的值

在 OSInit()中将 Deadline 和 Cstime 初始化为 0。上电后系统先调用 OSInit()进行初始化，再调用 OSTaskCreate() 来创建任务。在 μC/OSII 内核中，任务的优先级由用户指定。为了实现截止期最早优先算法，需要由系统指定任务优先级，但需要用户指定其截止期。因此，将 OSTaskCreate()中的参数 INT8U Prio 改为 INT8U Deadline，并定义局部变量 INT8U Prio 以记录分配给任务的优先级。μC/OS-II 内核中优先级范围为 0～64，OSTaskCreate()模块利用 PrioCreate()模块获得任务优先级,在 OSTimeTick 中该模块先用系统时间来更新每个任务的截

止期，如果截止期为零，则删除这个任务，再按从低到高的顺序逐个与每个任务比较截止期。找到一个截止期低于当前任务截止期的任务时，记录下此任务的优先级并对数组内容进行如下调整。

```
if(Deadline>CsRec[i][1])
{  int j=i;
   while(CsRec[j][1]&&(j>0))j--;
  if(j){
  for(j;j<=i;j++){
     CsRec[j][0]=CsRec[j-1][0];
      CsRec[j][1]=CsRec[j-1][1];
     CsRec[j-1][2]++;}
     }
  CsRec[i][1]=Deadline;
 }
```

实现任务调度 OSSched()模块，通过查表确定当前就绪任务中最高的优先级。为了不破坏系统的调用机制，在查表之前要更新 OSTCBPrioTbl[OSTCBHighRdy]的值。

OSTCBPrioTbl[OSTCBHighRdy] ＝ OSTCBPrioTbl[OSTCBHighRdy]->prio ＋CsRec[OSTCBHighRdy][2];

再查表取出优先级最高的任务进行比较和任务切换。系统通过 OSTaskDel()模块删除一个任务，在删除前，需要先更新 prio 的值。prio＝prio+CsRec[prio][2]; 在删除任务后还要将 CsRec 与 prio 对应位的三列清零。

4.3 μC/OS-Ⅱ中反馈调度算法扩展研究

由于 μC/OS-II 有统计任务 OSTaskStat( )功能，同时给用户留有接口 OSTaskStatHook( )可以进行统计任务的扩展，此函数每秒被 OSTaskStat( )调用一次，可以把统计信息显示出来。还有 OSTimeTickHook( )能够提供软定时中断。

另外一个接口函数 OSTaskSwHook( )能够协助完成反馈任务调度的功能。此函数可以测量每个任务的执行时间；统计每个任务的调度频率；统计每个任务运行时间的总和。这些统计结果都存储在每个任务的 TCB 扩展数据结构中。

每次任务切换时 OSTaskSwHook( )都被调用，但调用期间中断一直被禁止，因此应尽量减少其内部代码。

每次任务切换发生的时候，OSTaskSwHook( )先调用 PC_ElapsedStop( )函数来获取任务的运行时间，PC_ElapsedStop( )要和 PC_ElapsedStart( )一起使用，上述两个函数用到了处理器的定时器 2。其中 PC_ElapsedStart()功能为启动定时器开始计数；而 PC_ElapsedStop( )功能为获取定时器的值，然后清零，为下一次计数做准备。从定时器取得的计数将被拷贝到 time 变量中。然后 OSTaskSwHook( )调用 PCElapsedStart( )重新启动定时器做下一次计数。如果任务分配了 TCB 扩展数据结构，其中的计数器 TaskCtr 进行累加。TaskCtr 可以统计任务被切换的频繁程度，也可以检查某个任务是否在运行。TaskExecTime 用来记录函数从切入到切出的运行时间，TaskTotExecTime 记录任务总的运行时间。统计每个任务的上述两个变量，可以计算出一段时间内各个任务占用 CPU 的百分比，TaskType 记录任务是属于软实时任务，还是硬实时任务。OSTaskStatHook( )函数会显示这些统计信息。

利用 OSTimeTickHook( )、OSTaskStatHook( )和 OSTaskSwHook( )几个接口函数，可以将各个任务的运行时间信息测量出来，根据任务周期与截止期，OS 可以分析计算出任务利用率，以此为反馈调度算法提供有效数据。

4.4 反馈控制混合任务调度方法实现

仍以 4.1.2 小节的系统为例，为了完成功能需求，系统创建了多个任务——CAN 通讯，SCI 通讯，A/D 与 D/A，状态监测控制，存储控制，系统初始化启动，外围接口操作等。SCI 通讯是外部人为通过上位机触发，外围接口操作主要是 LCD 动态显示采样数据波形。由于采用了 EDF 调度系统，任务需要重新创建。六个任务的时间特性参数估计如下表所示：

表 4.3 系统中六个任务的时间特性

任务名称	估计周期	每个周期内的最大执行时间
CAN 通讯、SCI 通讯	80 ms	20 ms
A/D、D/A	10 ms	100 μs
状态监测控制	150 ms	8 ms
存储控制	100 ms	10ms

续表 4.3 系统中六个任务的时间特性

任务名称	估计周期	每个周期内的最大执行时间
系统初始化、重启动	10 ms	200 μs
外围接口操作	200 ms	45 ms

由于状态监测控制任务还动态生成两个任务，它们的时间特性如下：

表 4.4 动态生成两个任务时间特性

任务名称	估计周期	每个周期内的最大执行时间
报警	20ms	2ms
组态、分析	100ms	10ms

根据式 2.3，进行可调度性判断：

$$U=\sum_{i=1}^{8}\frac{E_i}{T_i}=\frac{20}{80}+\frac{100}{10000}+\frac{8}{150}+\frac{10}{100}+\frac{200}{10000}+\frac{45}{200}+\frac{2}{20}+\frac{10}{100}=84.9\%$$

由上可知，这样的系统是可调度的，而且还有裕量。截止期等同于周期。同样建立如下任务：

```
OSTaskCreate(SysChk, (void *)0, (void *)SysChkStk, 150);                    //状态监测控制
OSTaskCreate(SCanCom, (void *)0, (void *) SCanComStk,80);                   //CAN、串行通讯
OSTaskCreate(Storage, (void *)0, (void *) StrorageStk, 100);                //存储控制
OSTaskCreate(AffInterface, (void *)0, (void *) AffInterfaceStk, 200);       //外围接口
OSTaskCreate(AlertProcess,(void*)0 (void *)&AlertStk, 20);                  //报警
OSTaskCreate(ConfigProcess,(void*)0 (void *)&ConfigStk, 100);               //组态、分析
OSTaskCreate(FeedBackProcess,(void*)0 (void *)&FeedBackStk, 100);           //反馈控制
```

FeedBack 任务是用来反馈控制混合调度算法。用 OSTaskSwHook( )统计各个任务的执行时间，任务的性质等信息，保存在 OS_TCB 的扩展数据结构中，并记录当前任务利用率。另外用一个软定时器产生 100ms 定时，软定时器可以利用系统定时器产生：通过在 OSTimeTickHook( )里对系统 tick 计数，当达到(INT8U)((OS_TICKS_PER_SEC)/10)次后，计数器复位，同时调用 MboxPost( )发消息通知 FeedBack( )任务。

OSTaskSwHook( )的具体实现代码如下：

```
void   OSTaskSwHook()
{
    INT16U       time;
    Task_USER_DATA   *puser;
```

```
    time = PC_ElapsedStop();
    PC_ElapsedStart();
    puser = OSTCBCur->OSTCBExtPtr;
    if ( puser != (void *)0 ) {
        puser ->TaskCtr++;
        puser ->TaskExecTime = time;
        puser ->TaskTotExecTime += time;
        puser->Utilization=TaskExecTime*100/puser->deadline;
}

}
```

FeedBack ( )的实现代码(部分)如下：

```
void FeedBack(void *data)
{
INT8U  err,i;
void    *msg;
   FeedBackMbox = OSMboxCreate((void *)0);
   while(1) {
       msg = OSMboxPend(FeedBackMbox, 0, &err);
       if(err == OS_NO_ERR) {
                  统计硬实时任务利用率；
                        统计软实时任务利用率；
                        MixScheduling（）；//混合任务调度
                        QosRegulate();       //Qos 调节
                    }
   }
}
```

MixScheduling()的实现代码(部分)如下：

```
 Void   MixScheduling（）
{   INT8U deltaPlus=0.00;
    INT16U Urmax=0.98;
     Uss= Ur- Us;
   if   (Uhard<= Usold)
      { Us= Usold;
```

```
            Ur= Urold
        }
if  (Uhard< =Us && Usoft<=Uss )
        deltaUsoft= Uss- Usoft;
elseif ( Uhard< =Ur && Uhard>= Us )
        {Us= Uhard+ deltaPlus;
        Uss= Ur- Us;
        deltaUsoft= Uss- Usoft;
      }
Elseif ( Uhard>= Ur )
        { Ur= Urmax;
        Us= Ur;
        Uss=0;
        deltaUsoft= Uss- Usoft;
      }
elseif (Usoft> =Uss& Utotal<= Ur)
        deltaUsoft=0;
   }
```

利用 OSTaskStatHook( )还可以将当前 CPU 的利用率显示到屏幕上面，同时可以把数据保存起来,在 OSTaskStatHook( )中还可以建立错误日志。

4.5 系统性能验证

42 振动模块通过组态设置，与配套振动源组合，可以对传感器进行校验。振动源采用 5Kg 的激振器做为振动台，最大推力可达到 50 牛顿，最大振动位移可达±3mm，空载最大加速度可达 42g（410m/s^2）。通过 42 振动模块 D/A 输出正弦波去控制激振器，在不同位移下，改变不同频率点，测量速度传感器与激振器标准回馈信号，LCD 动态显示采样波形，并保存采样值，上位机通过串口组态并获取采样值，分析传感器性能。

如图 4.10 所示。对激振器控制采用闭环控制：激振器输出标准传感器的位移、速度、加速度信号，同时被校传感器也输入给 42 振动模块这些参量，根据

组态对 42 振动模块设定参数值，采用分区 PID 控制算法，D/A 输出正弦控制信号。

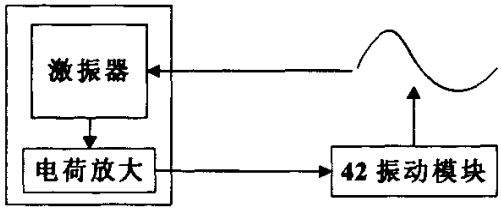

图 4.10 振动校验示意图

4.5.1 外界负载扰动分析

由于 D/A 采用定时器中断输出，而定时器中断任务比其他所有任务的优先级都高，每次中断服务程序到来时，都需要对当前运行任务进行任务切换。当控制信号是相对较低的频率的正弦信号时，定时器中断周期值比较大，此时任务切换并不是很频繁，这对 CPU 资源的消耗并不很大；当信号频率明显上升时，由于定时器周期减小，任务切换频率明显加剧，CPU 疲于对传感器进行采样、平均化计算以及切换任务中。如果这时到来外部中断 SCI 任务，那么对正在工作中的 A/D 采样、存储数据、D/A 输出是一种负载到来扰动，由于串口的人为操作的不确定性，以及扰动到来的非周期性，同时一般串口通讯具有重试机制，可以把该扰动看成具有固定截止期的非周期软实时任务。实验表明，此时容易出现以下几种现象：

(1) 获得的通过串口上传上来的数据帧很容易发生错误
(2) 42 振动模块不能对 SCI 中断不能及时响应
(3) 采样保存数据明显有部分值发生偏差
(4) LCD 液晶动态显示波形发生延迟

4.5.2 系统改进及性能分析

按照前面的设计方法(软件结构和反馈混合任务调度算法)，在实际系统上运行时发现上述的现象明显减少，只是偶然 SCI 中断不能及时响应。根据反馈控制混合任务调度模型及框架，系统在收到 SCI 中断后，由于 CPU 利用率带宽

在满足硬实时任务时，未能提供给新进入软实时任务以足够的 CPU 负载带宽，只能阻塞该软实时任务。通过修改处理程序以及上下位机通讯握手协议，当任务被阻塞后，自动进入队列；当系统相对空闲时，任务出列并发送给上位机数据上传要求，让上位机重发一次。

在系统运行时，将 CPU 利用率显示在屏幕上，把数据导出，用图形表示出来，可以看到即使在有该外来负载扰动到来的情况下，系统仍然可以较好地运行，CPU 利用率的变化也比较平缓。

观察到 CPU 利用率见图 4.11，从加入串口负载扰动后开始观察，纵坐标为 CPU 利用率。

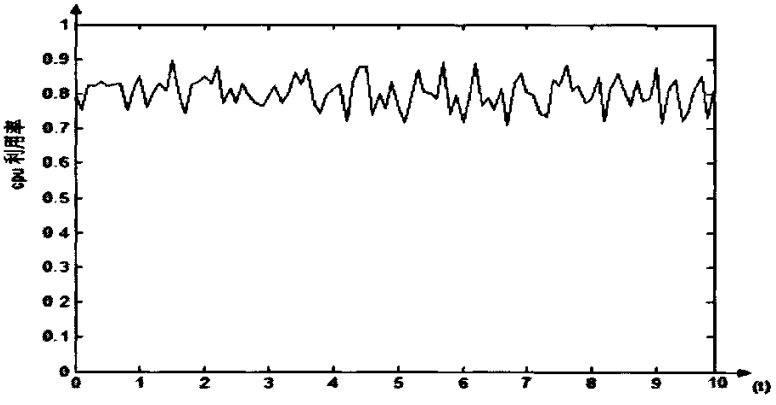

图 4.11 改进后 CPU 利用率

如果在没有引入反馈控制混合任务调度算法的系统中，当串口负载扰动到来时，系统的 CPU 利用率见图 4.12。

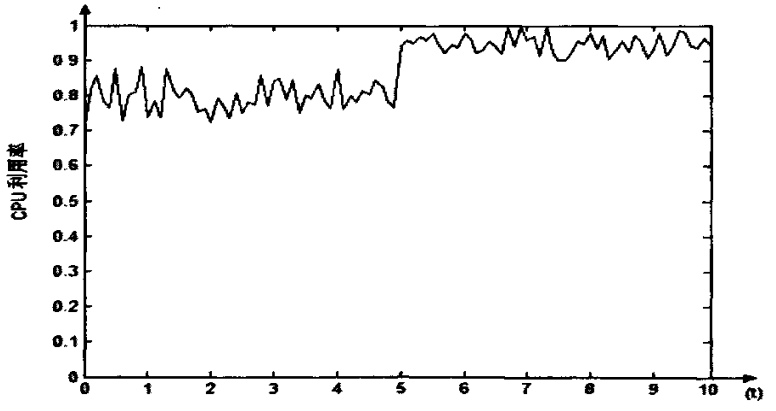

图 4.12 改进前 CPU 利用率

从 4.11 图可以看出：当系统引入反馈控制混合任务调度算法后，即使有串口负载扰动，CPU 仍然可以良好地运行，利用率稳定在 80%左右；而 4.12 图中可知，加入扰动后系统几乎利用率接近为 0.95。由此可知，在 μC/OS-II 中引入了反馈控制混合任务调度策略的实时内核抗干扰性能更加优良，系统更加可靠。

然而，由于系统在统计 CPU 利用时是软定时器触发进行计算，在定时器启停过程中，以及在任务切换、反馈任务计算延迟等存在一定误差，这是单处理器利用以及 μC/OS-II 的实时性粒度所决定的。虽然提高了系统对外界的鲁棒性，但这是以系统内核消耗更大的存储空间和采用复杂的调度算法来换取的。

4.6 本章小结

本章首先介绍了课题背景："振动系统监控校验保护仪"的研制项目，对系统的框架结构以及软件架构做了详细说明，并从其中 42 振动模块软件设计出发，通过对 μC/OS-II 实时内核的研究，对内核移植、改造与软件开发，根据反馈控制混合任务调度思想，实现了反馈混合任务调度方法。对传感器校验实验中发现串口负载到来对实时系统的影响，通过实践验证了改进算法后系统的鲁棒性。

第 5 章 结论与展望

5.1 结论

随着计算机技术的飞速发展与普及，实时系统已经成为人们生产和生活中不可或缺的组成部分,实时系统结构的日益复杂，运行环境的不确定性因素增多，任务调度越来越受到研究人员的关注。本文对原有的实时理论进行扩充，在满足新需求的同时，进一步提高其应用水平。本文的主要工作如下：

1) 研究了 RTOS 中的经典调度方法，分析了这些方法的局限性。
2) 分析混合任务，介绍了混合任务调度算法，其中详细讨论一种基于空闲时间的混合任务调度算法，并提出混合任务调度的模型框架。
3) 分析了反馈控制实时调度算法，对 PI 控制器详细分析并介绍了利用率、丢失率、利用率丢失率反馈调度算法，总结各调度算法优劣。
4) 研究了控制与调度协同设计，以 Qoc 作为评价基础，给出了基于 Qoc 的调度策略，并提出基于实时控制与调度协同设计模型，使用弹簧算法根据控制任务的优先等级，对有限 CPU 利用率再分配，灵活地应对扰动到来对被控系统的影响。
5) 在反馈控制混合任务调度方面，以反馈控制混合任务调度框架为基础，可调软硬实时任务负载带宽为出发点，准入与 Qos 为软实时任务负载调节手段，提出了反馈调度的水箱模型以及控制器的控制算法。
6) 设计反馈控制调度仿真环境，对控制与调度协同设计模型、反馈控制混合任务调度算法仿真，并获得预期效果。
7) 研究了 μC/OS-II 的任务管理、内核调度方面的内容，进而把 μC/OS-II 移植到 TMS320F2407A 硬件平台上，修改内核，设计实时任务和软件结构，完成“振动系统监控校验保护仪”项目中 42 振动模块设计。
8) 研究 μC/OS-II 实时内核调度的可扩展性，在现有系统的平台上，引入了基于反馈控制任务调度算法，并比较在负载扰动到来时，系统 CPU 利用率变化，实践验证该调度算法具有提高系统的鲁棒性。

5.2 进一步的工作方向

今后的研究可以在以下几个方面进一步展开：

1) 混合任务分析中对软实时任务多种到达模式的研究，同时基于混合任务调度的模型框架给出多种混合任务具体算法，仿真并实现。
2) 研究基于二级层次调度模型算法，把反馈控制调度算法应用在二级层次调度模型下，同时并存多种调度算法，对软硬实时任务分类调度。
3) 对于控制与任务调度协同设计，研究控制利用率权值 w_i 的变化对系统整体 Qoc 影响，改进弹簧算法，使得在灵活分配 CPU 利用率同时，整体系统的控制性能提高。
4) 深入研究反馈控制混合任务调度算法中提出的水箱模型，建立水箱的动态模型，研究软硬实时任务负载特性，改进控制器算法。
5) 基于先进控制理论与智能理论，研究反馈控制先进调度算法。仿真并对部分先进智能算法在实时调度理论中应用的局限性以及先进性做出概括。
6) 研究 TrueTime 工具箱内核，编写反馈调度算法 API 函数，建立实时系统性能评价机制，通过使用功能强大的 ARM 平台，移植 RT-Linux，细化系统响应粒度，提高整体实时系统实时性能。

致谢

转眼间两年半的硕士研究生生活即将结束，也许是这段时光太美好，所以让人感觉如此短暂。我非常庆幸能够在同济大学过程控制研究室渡过我这段难忘的人生旅程，因为这里有最认真负责的老师和最踏踏实实的同学，正是这样一种环境给了我学术研究的巨大动力，正是这样一个团结的集体让我在遇到困难的时候能够平心静气，刻苦攻关，可以说是因为有了我的老师、同学，才有了今天这篇论文，才有了充实丰富的研究生学习生活。

在临将毕业之际，我要对所有支持帮助过我的老师和同学表达我最深切的感谢。

首先感谢我的导师萧蕴诗教授，是他的高瞻远瞩为我确定好研究的大方向，是他的敏捷与睿智在我出现问题的时候能及时指出，是他的严谨负责熏陶了我，使我对自己的工作也精益求精。萧老师不仅在学术上是我们的榜样，他做人的塌实和严谨也将对我的未来发展产生深远的影响。

感谢岳继光教授，何斌、苏永清、吴继伟副教授，王娥娅老师，他们对全体同学的负责和悉心关照给我留下了深刻的印象。在朝夕相处的这段日子里，他们在学术上给了我莫大的鼓励，在生活上给了我无微不至的关心，他们的鼓励是我自信的源泉。

还要感谢在校期间教育和关心过我的所有老师，无论何时何地，您们的教育之恩，我将永远不会忘记。

感谢与我同在一个实验室的同学，周洋、王超、邹素瑞、王一丁、王剑、马旻文、杨柳、杨歆怡等，同时也要感谢我的师姐陆萍以及我的师妹师弟们。和这些聪明勤奋的同学在学术讨论中，我得到了许多启发，同时也要感谢他们在平时生活中对我的帮助。

感谢西门子威迪欧汽车电子中国有限公司的同事们，他们无论在生活上还是在技术上都给与了我很多帮助，也给我的学生生涯增添了很过丰富色彩。

感谢父母和家人在生活和学习上给予的关怀、支持和鼓励，感谢他们多年来的养育之恩，他们总是默默地支持我做出选择，不断鼓励我向更高的方向前进。感谢我的朋友王强，肖波，邵健青，朱伟，金坤，肖涛等，感谢他们一直给与我新的认识与机会，以及给与我的无私的支持。

感谢所有给予我支持和帮助的老师和同学，感谢同济大学对我的培养。

谨以此文，表达我深深的感激！

毛磊

2007 年 3 月

参考文献

[1] Jane W.S.Liu, Real-Time Systems(实时系统影印版). 北京：高等教育出版社，2002

[2] 孔祥营，柏桂枝. 嵌入式实时操作系统 Vxworks 及其开发环境 Tornado. 北京：中国电力出版社，2003

[3] Mario Aldea Rivas, Michael GonzAlez Harbour. Ada RTS POSIX--Extending Ada's Real-Time System Annex with the POSIX Scheduling Services. Ada Letters, 2001, Vol XXI(I).

[4] 胡志刚等. 嵌入式 Linux 实时性方法. 中南大学学报，2004，Vol.35(4): 638~642

[5] 王苗田. 嵌入式系统设计与实例开发——基于 ARM 微处理器与 μC/OS-II 实时操作系统. 北京：清华大学出版社，2003

[6] Jean J.Labrosse 著，邵贝贝译. μC/OS-II 源码公开的实时操作系统. 北京：中国电力出版社，2001

[7] 王京起，黄健，沈中杰. 嵌入式可配置实时操作系统 eCOS 技术及实现机制. 北京：电子工业出版社，2005

[8] 叶宏材. Windows CE.NET 嵌入式工业用控制器及其自动控制系统设计. 北京：清华大学出版社，2005

[9] 陆丽娜，伍卫国等. 分布式操作系统. 北京：电子工业出版社，2002

[10] Karsten Albers, Frank Slomka. An Event Stream Driven Approximation for the Analysis of Real-Time Systems. Proceedings of the 12th 16th Euromicro Conference on Real-Time Systems, 2004, IEEE.

[11] 王知学. 嵌入式操作系统调度算法研究：[博士学位论文]. 上海：中国科学院研究生院，2003

[12] 徐涛，赵川. SystemC 中抢占式进程调度的建模. 计算机应用与软件，Vol.21,No.7,pp41～44

[13] 王永吉，陈秋萍. 单调速率及其扩展算法的可调度性判定. 软件学报，2004，Vol.15(6):799~814

[14] 罗蕾. 嵌入式实时操作系统及其应用开发. 北京：北京航空航天大学出版社，2005

[15] Wayne Wolf 著. 孙玉芳等译. 嵌入式计算系统设计原理. 北京：机械工业出版设，2004

[16] Maryline SILLY, Houssine CHETTO. An Optimal Algorithm for Guaranteeing Sporadic Tasks in Hard Real-Time Systems. Proceedings of the Second IEEE Symposium, 1990. 578~585

[17] Kevin Jeffay. Scheduling Sporadic Tasks with Shared Resources in Hard-Real-Time Systems. IEEE Real-Time Systems Symposium, 1992. 89~99

[18] 邹勇等. 开放式实时系统中的自适应调度方法. 计算机学报，2004，Vol.27(1):58~65

[19] Kevin Jeffay, Donald F.Stanat, et al. On Non-Preemptive Scheduling of Periodic and Sporadic Tasks. IEEE, 1991. 60~67

[20] Thaker, G.H., Lardieri, P.J., Krecker, D.K. et al. Empirical quantification of pessimism in

state-of-the-art scheduling theory techniques for periodic and sporadic DRE tasks. Real-Time and Embedded Technology and Applications Symposium, 2004. 490~499

[21] Cervin, A., Eker, J. Feedback scheduling of control tasks. Decision and Control, 2000, Vol. 5: 4871~4876

[22] A.K.Mok. Fundamental Design Problems of Distributed System for the Hard Real-Time Environment: [PhD thesis].

[23] 郑大钟，赵千川. 离散事件动态系统. 北京：清华大学出版社，2001

[24] K. Jeffay, F. D. Smith, A. Moorthy, et al. Proportional Share Scheduling of Operating System Service for Real-Time Applications. IEEE 19th Real-Time Systems Symposium, 1998.

[25] Jefferies, N. Sporadic partitions of binomial coefficients. Electronics Letters, 1991, Vol. 27(15): 1334~1336

[26] Anderson, J.H., Srinivasan, A. Early-release fair scheduling. Real-Time Systems, 2000. 35~43

[27] Bernat G., Burns A. New Results on Fixed Priority Aperiodic Servers. Proceeding of 20th IEEE Real-Time Systems Symposium, Phoenix, AR, USA: 68~79

[28] Vieira, Sibelius Lellis. Sporadic task scheduling with overload conditions. Advances in Engineering Software, 1999, Vol.30(1):1~11

[29] 王强. 混合实时事务调度与并发控制研究：[博士学位论文]. 北京：中国科学院研究生院，2003

[30] Geniet, Dominique. Scheduling hard sporadic tasks with regular languages and generating functions. Theoretical Computer Science, 2004, Vol.313(1):119~132

[31] Robert Davis, Alan Burns. Optimal Priority Assignment for Aperiodic Tasks with Firm Deadlines in Fixed Priority Pre-Emptive Systems. Information Processing Letters, 1995, Vol.53:249~254

[32] Baruah, S.K., Mok, A.K, Rosier, L.E. Preemptively scheduling hard-real-time sporadic tasks on one processor. Real-Time Systems Symposium, 1990. 182~190

[33] Vazhkudai, S., Schopf, J.M. Predicting sporadic grid data transfers. High Performance Distributed Computing, 2002. 188~196

[34] Jongwon Lee, Sungyoung Lee et al. Schedul in Hybrid System--Scheduling of Hard Aperiodic Tasks in Hybrid Static/Dynamic Priority Systems. ACM SIGPLAN Notices, 1995, Vol 30(i).

[35] Rivas, M.A., Harbour, M.G.. Evaluation of new POSIX real-time operating systems services for small embedded platforms. Real-Time Systems, 2003. 161~168

[36] Execution Scheduling - [Information technology-- Portable Operating System Interface (POSIX)-Part 1: System Application Program Interface (API) [C Language]]. ISO/IEC Std 9945-1, ANSI/IEEE Std 1003.1 , July 12 1996. 287

[37] Rivas, M.A., Gonzalez Harbour, M. POSIX-compatible application-defined scheduling in MaRTE OS. Real-Time Systems, 2002. 67~75

[38] 万柳. 嵌入式实时操作系统 VxWorks 内核调度机制分析. 计算机应用与软件，2004，Vol.21(6):51~52

[39] 何希才等. 通用电子电路应用 400 例. 北京：电子工业出版社，2005. 234

[40] 全新实用电路集粹丛书编辑委员会. 科教、娱乐应用电路集粹. 北京:机械工业出版社，2005. 172

[41] Raj Kamal 著. 嵌入式系统——体系结构、编程与设计（, 陈曙晔等译）. 北京：清华大学出版社，2005

[42] S.L.Vieira、M.F,Magalhaes. On-line Sporadic Task Scheduling in Hard Real-Time Systems. IEEE Real-Time Systems, 1994. 186~191

[43] Urs Loher. Sporadic Information Sources. IEEE International Symposium, 1995. 44

[44] Tei-Wei Kuo, Wang-Ru Yang et al. A Class of Rate-Based Real-Time Scheduling Algorithms. IEEE TRANSACTIONS ON COMPUTERS, 2002, Vol.51(6).

[45] 谢长生，马进德，黄浩. 基于 μC/OS-II 的任务调度策略研究. 计算机工程与科学，2004，Vol.26(8):70~73

[46] 涂刚.软实时系统任务调度算法研究：[博士学位论文]. 武汉：华中科技大学，2003

[47] Damir Isovid, Gerhard Fohler. Efficient Scheduling of Sporadic, Aperiodic, and Periodic Tasks with Complex Constraints. IEEE, 2000.

[48] 金宏，王宏安等. 模糊反馈控制实时调度算法. 软件学报，2004，Vol.15(6):791~798

[49] Alan Burns 著, 王振宇译. 实时系统与编程语言. 北京：机械工业出版社，2004

[50] 张大波. 嵌入式系统原理、设计与应用. 北京：机械工业出版社，2005

[51] C.Lu,J.A.Stankovic,G.Tao.and S.H.Son. Feedback Control Real-Time Scheduling: Framework,Modeling, and Algorithms[J].Real-Time Systems Journal, July 2002,23(1/2): 85-126

[52] G.Bernat, A.Burns. Specification and Analysis of Weakly Hard Real-Time Systems. PhD thesis. Department de Ciencies Matematiques Informatica,Universitat de les Illes Balears, Spain, 1998.

[53] G.Bernat, A.Burns, A.Llamosi. Weakly Hard Real-Time Systems. IEEETransactions on Computers, 2001, 50(4): 308～330

[54] G.Bernat, A.Burns. Combining (n m)-hard deadlines with dual priority scheduling. In Proc. 18th IEEE Real-Time Systems Symposium. San Francisco USA. 1997. 46～57

[55] M.Klein, T.Ralya, B.Pollak,R.Obenza,M.G.Harbour. A Practitioner's Handbook for Real-Time Analysis fro Guide to Rate Monotonic Analysis for Real-Time Systems[M].Kluwer Academic Publishers, 1993.

[56] Chenyang Lu. Feedback Control Real-Time Scheduling[D] .Ph.D. Thesis University of Virginia,May,2001

[57] J.A.Stankovic,M.Spuri, K.Ramamritham, and G.C.Buttazzo.Deadline Scheduling for Real-Time Systems EDF and Related Algorithms[M]. Kluwer Academic Publishers,1998.

[58] G. Bernat, R. Cayssials. Guaranteed On-Line Weakly Hard Real-Time Systems.In Proc. 22th IEEE Real-Time Systems Symposium. London, England. 2001. 25～26

[59] Chenyang Lu, John A. Stankovic, Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms. Journal of Real-Time Systems,Special Issue On Control-Theoretical Approaches to Real-Time Computing, 2001

[60] 魏立峰,于海斌-基于混合线性反馈控制结构的软实时调度算法研究[J]，信息与控制，2003 ,32(6):490-494

[61] C.Lu,J.A.Stankovic,G.Tao, and S.H.Son. Design and Evaluation of a Feedback Control EDF Scheduling Algorithm. Proceedings of the IEEE Real-Time System Symposium, Phoenix, AZ, December 1999

附录 A　系统硬件原理图

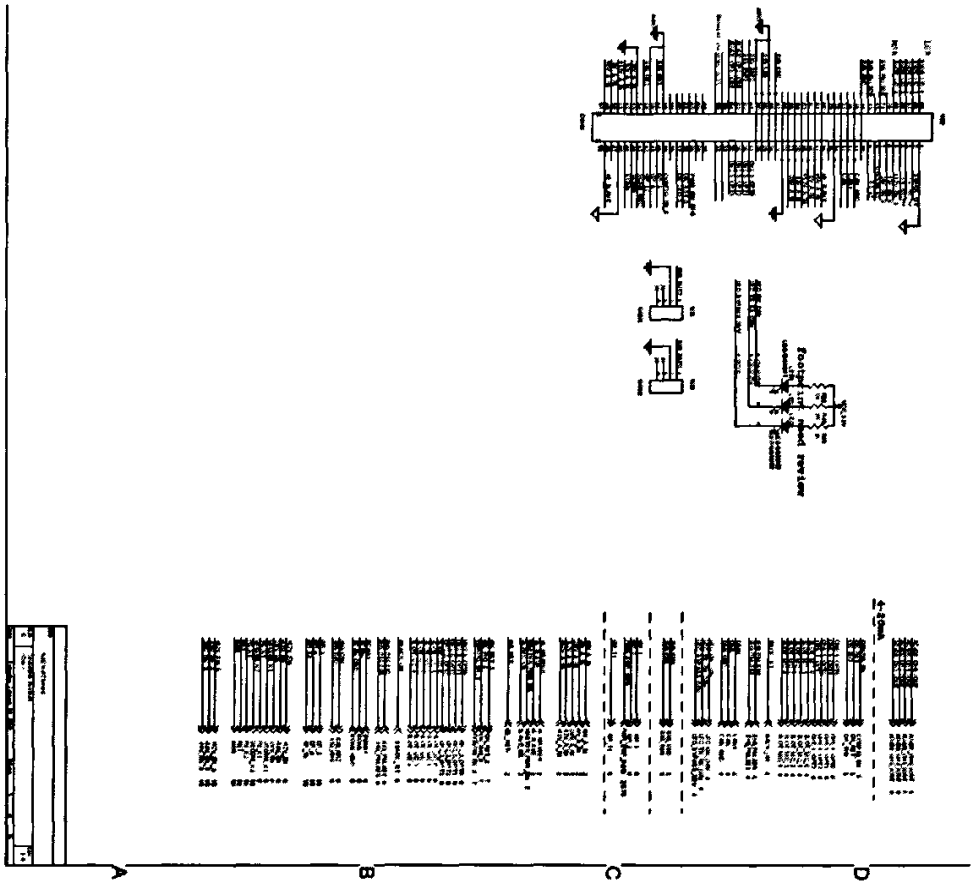

图 A1 42 接口电路原理图

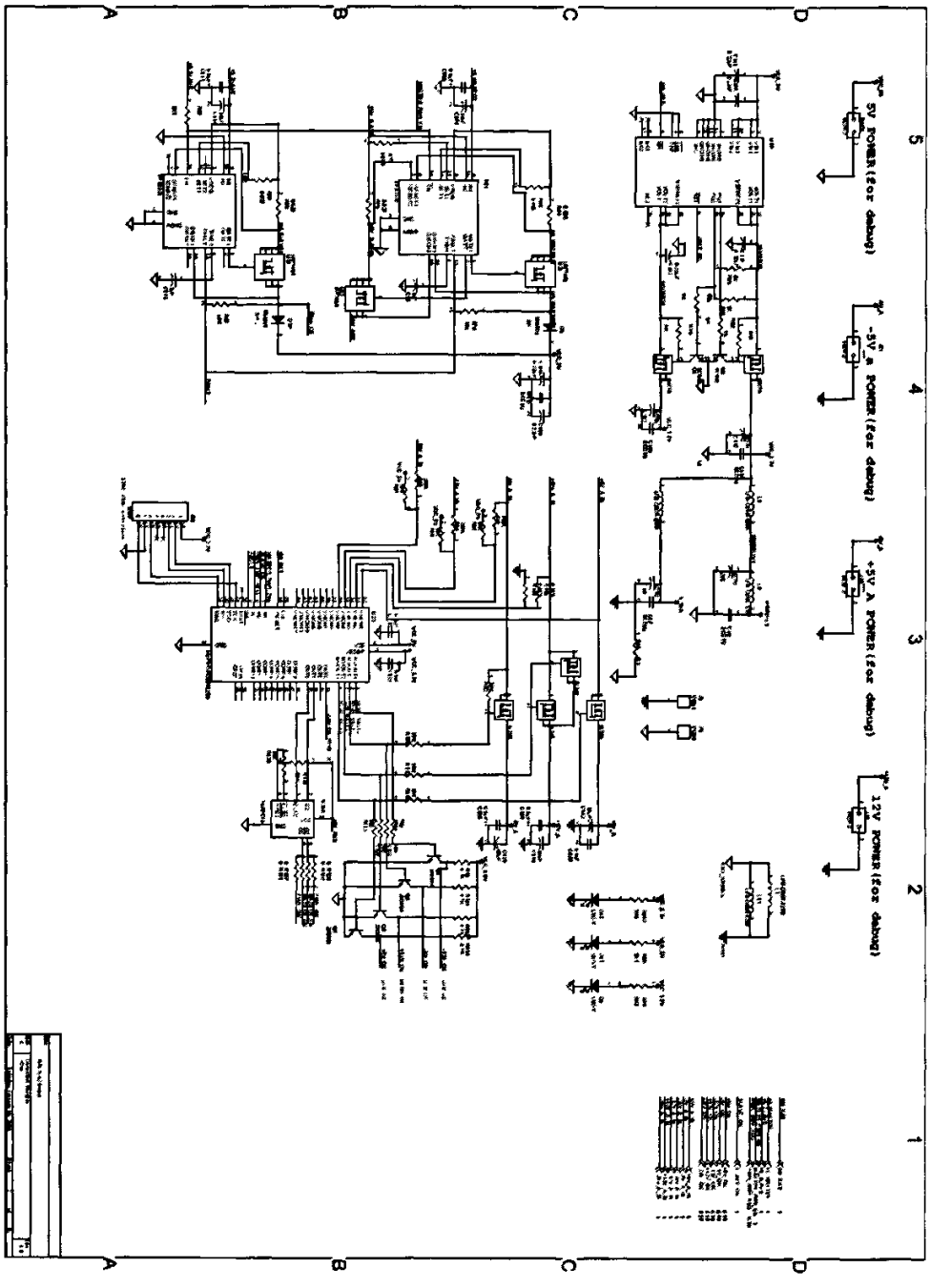

图 A2 系统供电电路原理图

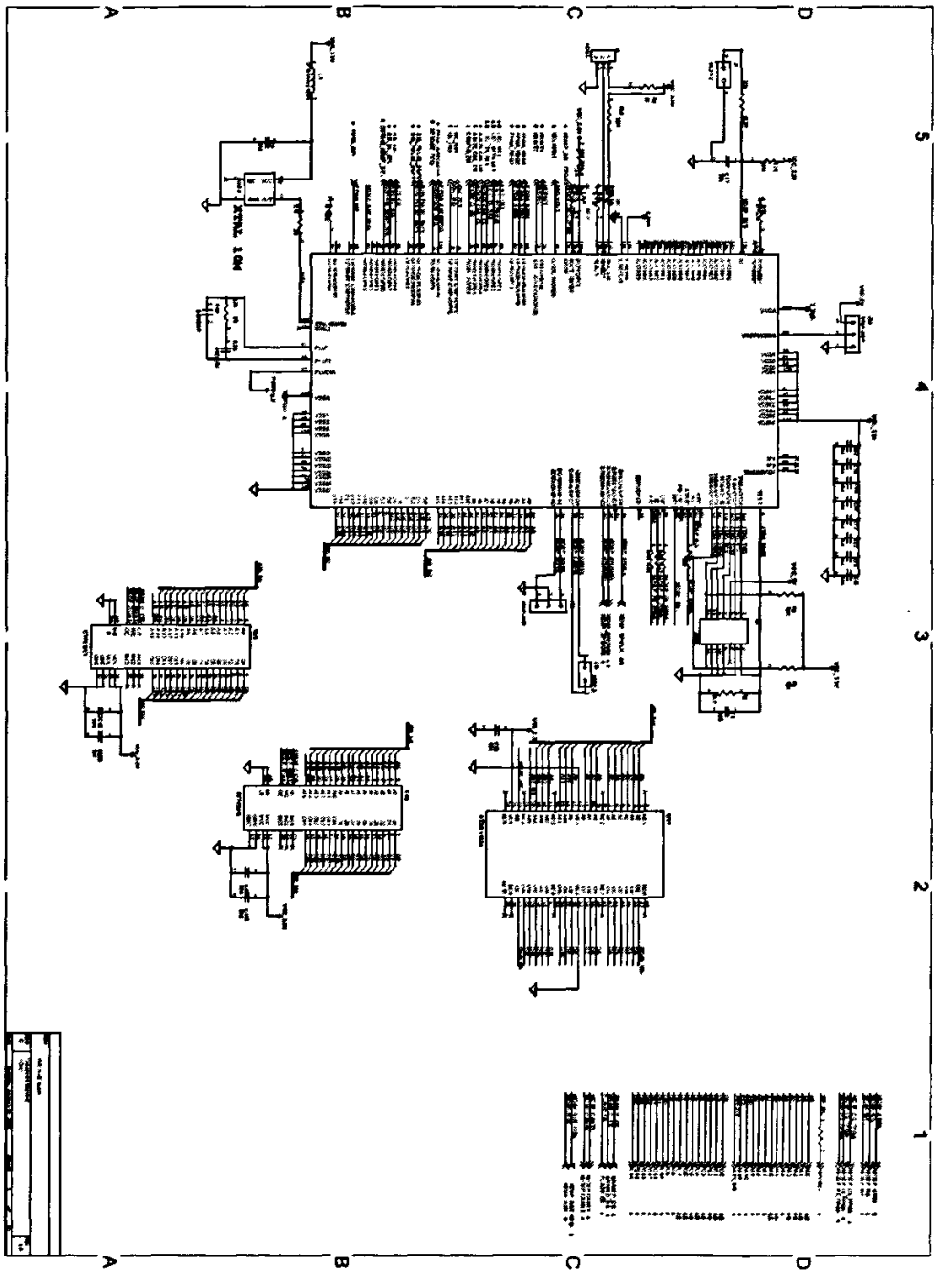

图 A3 DSP 电路原理图

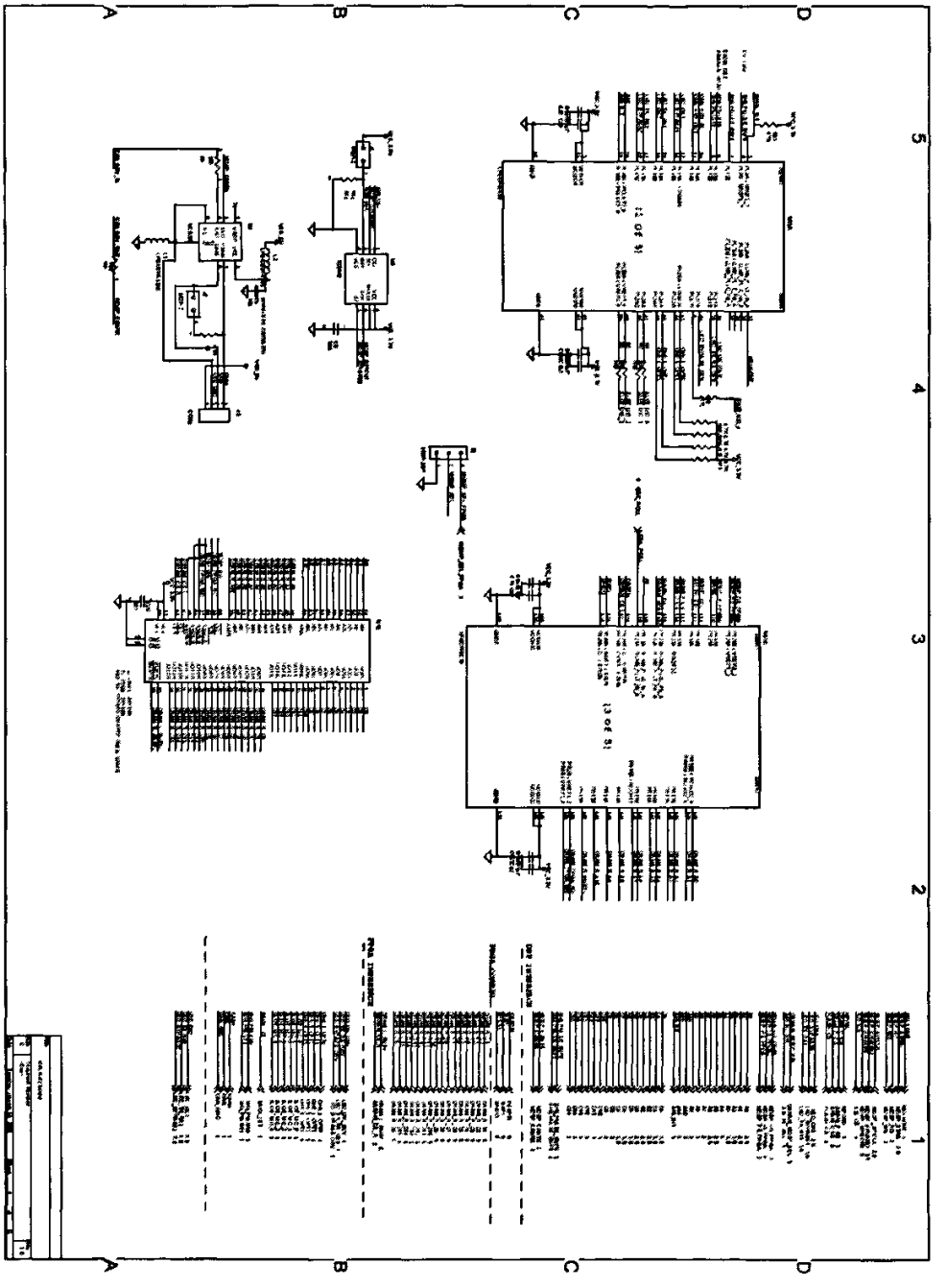

图 A4 FPGA(a)原理图

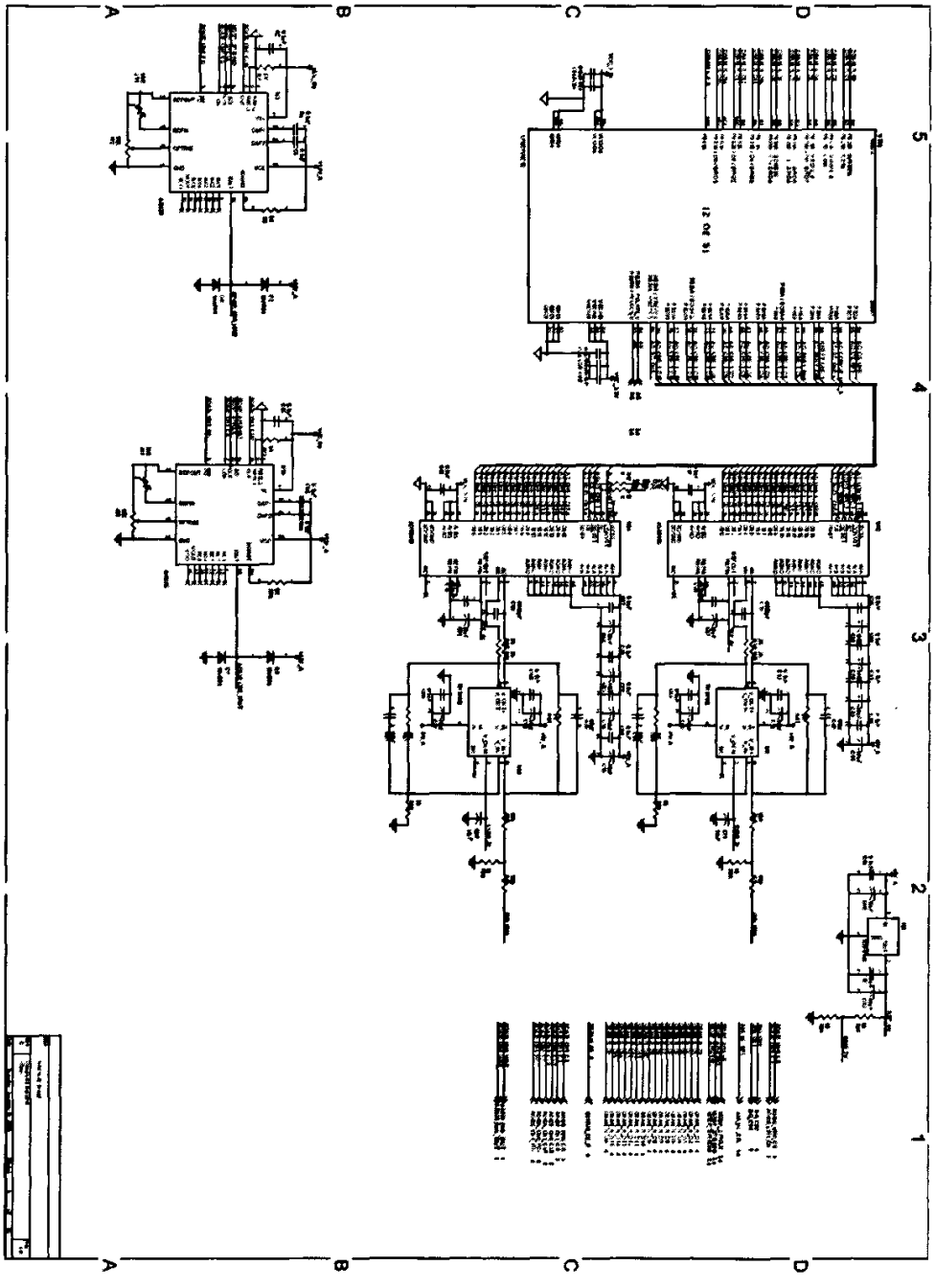

图 A5 FPGA(b)原理图

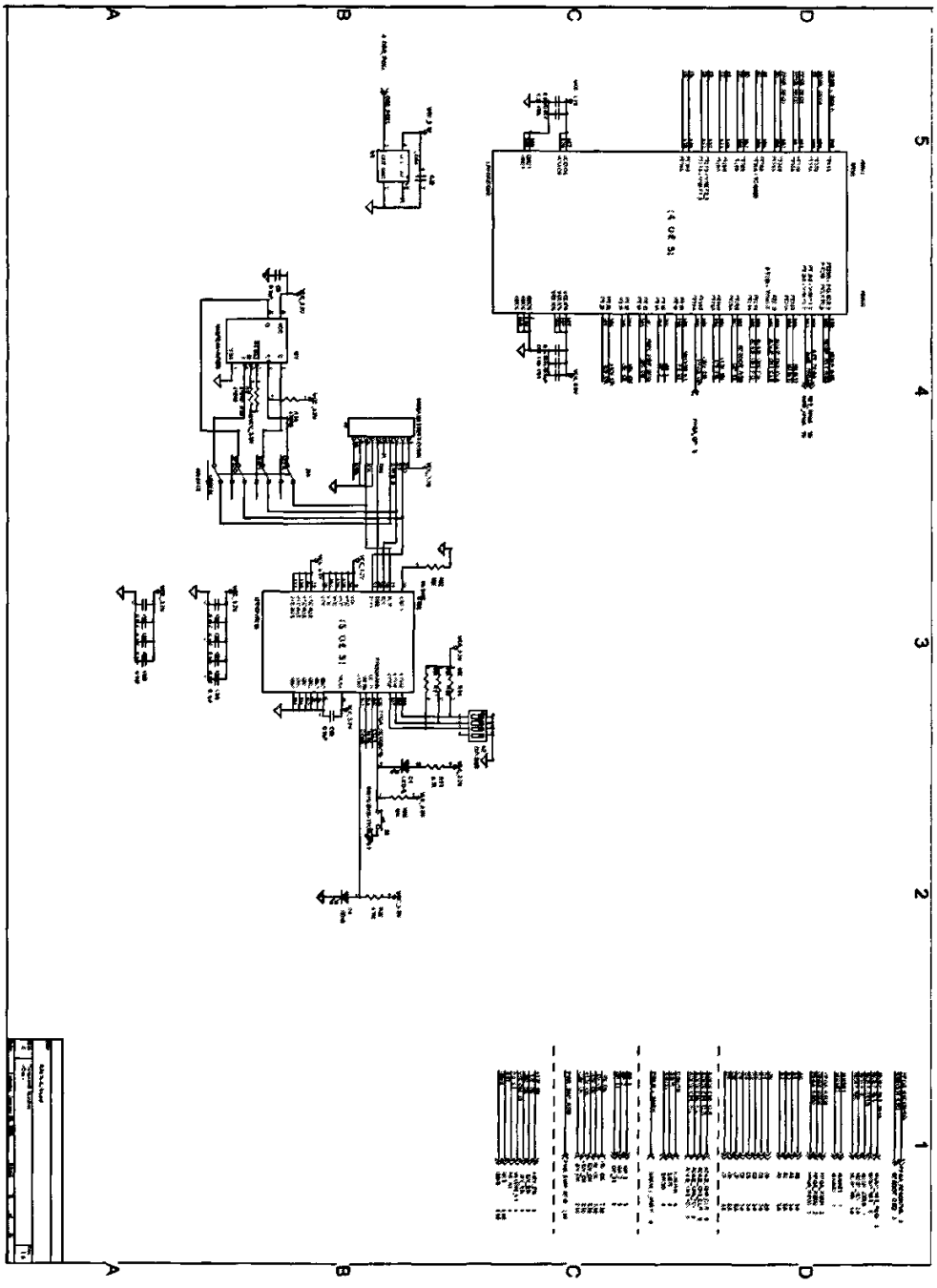

图 A6 FPGA(c)原理图

个人简历　在读期间发表的学术论文与研究成果

个人简历：

毛磊，男，1981 年 12 月生。

2004 年 6 月毕业于上海电力学院 自动化专业 获学士学位。

2004 年 9 月入同济大学读硕士研究生。

已发表论文：

[1] 毛磊，萧蕴诗，何斌，岳继光. 实时系统的控制与任务调度协同设计. 计算机测量与控制，2007, Vol.15(3):352-353

科研项目：

[1] 2005-04～2005-06：211 实验室建设，杭州高校自动化研究所 “过程控制教学实验台”开发与调试

[2] 2005-07～2005-10：国家自然基金项目，上海市局管基金项目配套“微小力测试平台”开发

[3] 2005-11～2006-10：上海瑞视电子仪表有限公司 同济大学过程控制实验室 "振动系统监控校验保护仪"开发