哈尔滨理工大学

硕士学位论文

实时操作系统任务调度算法的硬件实现

姓名：王显山

申请学位级别：硕士

专业：@

指导教师：李岩

201103

实时操作系统任务调度算法的硬件实现

摘 要

实时操作系统在整个嵌入式系统中扮演着重要的角色，控制着整个系统的工作与运转，实时操作系统一个性能的优劣将对整个系统的所有性能产生直接的影响。已有的实时操作系统内核是加在应用程序中的软件，它不仅增加了存储空间的用量，而且增加了应用程序的额外负荷。尤其在实时性较强的场合，在限定时间内响应处理任务已经成为了对实时操作系统的一个基本要求。

针对实时操作系统的开销导致应用程序可执行性降低的问题，单纯依靠改进调度算法已不能使其实时性有显著的提高，所以提出将实时操作系统内核硬化到FPGA平台上的设计方案，作为独立的硬件模块与处理器并行执行。建立由中断控制器、输入/输出寄存器和实时任务管理模块组成的硬件实时操作系统总体结构。其主要工作过程：通过数据总线把相应的命令和参数发送到指定硬件逻辑单元的输入寄存器中，硬件逻辑单元作出相应的处理，并将处理结果送到相应的输出寄存器中，以供CPU进行读取。

本文以嵌入式实时操作系统 μC/OS-II 为研究对象，修改 μC/OS-II 中由软件实现的数据结构，根据硬件逻辑电路的并行性特点，搭建各个功能模块的硬件逻辑结构，整个设计采用VHDL硬件描述语言描述各个功能模块，利用Xilinx公司的ISE 8.2软件环境进行系统调试分析，完成功能仿真验证。

本文主要设计并实现了任务管理模块和信号量管理模块的硬件逻辑电路。任务管理模块中对μC/OS-II的任务调度算法进行改进和硬化，在uC/OS-II内核原有的基于优先级抢占式调度算法的基础上，扩展相同优先级任务的调度算法，去除了原系统对每个任务必须有不同优先级的要求，采用硬件逻辑实现实时操作系统中的任务管理模块，使其实时性和确定性显著提高，充分发挥了多任务潜在的并行性；分析并改进 μC/OS-II 中对信号量的管理和应用，设计并实现信号量管理模块的硬件逻辑电路，降低了频繁查表和访问内存带来的系统开销。将实时操作系统的调度功能由原来的纯软件实现转变为硬件实现，将极大的提高实时操作系统的实时性以及处理能力。

关键词 实时操作系统；任务管理；硬件任务调度器；现场可编程门阵列

Hardware Implementation of Real Time Operating System Task Scheduling Algorithm

Abstract

Real-time operating system plays an important role in the embedded systems, controls the work and operation of the whole system. The merits of a performance of real-time operating system have a direct impact on all the performance of the system.Existing real-time operating system kernel is the software that is added to applications. It not only increases the amount of storage space, but also increases applications of additional load, especially in a strong real-time occasion. Response to processing tasks within the time limit has become a basic requirement for real-time operating system.

Overhead for real-time operating system cause the application to reduce the enforceability.Only improving scheduling algorithms can not make real-time increased significantly. So this paper proposed hardware real-time operating system FPGA-based design. Real-time operating system kernel is hardened to the FPGA platform. As a separate hardware module execute in parallel with the processor.Established by the interrupt controller, input/output registers and real-time task management module of the overall structure of the hardware real-time operating system. The main working process: Through the data bus to send the appropriate commands and parameters to a specific hardware logic unit input registers, hardware logic units take appropriate actions, and the results are sent to the corresponding output registers for the CPU to read.

In this paper, embedded real time operating system μC/OS-II uses as the research object. Modify the μC/OS-II in the data structure by the software.According to the parallelism features of the hardware logic, build each module of the hardware logical structure. The whole design utilizes VHDL hardware description language to describe each function module.Using Xilinx's ISE 8.2 software to analysis system debugging, to complete function simulation.

In this paper, design and implement the hardware logic for the task

management module and the semaphore management module. Task management module improves and hardens the μC/OS-II task scheduling algorithm.Based on the uC/OS-II kernel original priority-based preemptive scheduling algorithm. Extend the same priority task scheduling algorithm.Remove the requirement for each task of the original system having a different priority. Use hardware logic to implement the real-time operating system task management module, deterministic and real-time are improved significantly. Give full play to the potential multi-task parallelism. Analysis and improve the semaphore management and application of μC/OS-II. Design and implement the hardware logic for semaphore management module.Reduce the overhead for frequently look-up table and access memory. Make scheduling function of real-time operating system from pure software into hardware to implement, it greatly improves real-time and processing of the real-time operating system.

Keywords real-time operating system, task management, hardware task scheduler, field programmable gate array

第1章 绪论

1.1 课题来源

随着嵌入式技术的发展，实时操作系统RTOS(Real Time Operating System)越来越多地应用在嵌入式系统中，如：航空航天、工业控制、汽车电子和核电站建设等众多领域。仅仅依靠任务调度算法改进已经不能使现有的纯软件实现的实时操作系统的实时性有显著的提高。如果采用硬件逻辑实现 RTOS 中的任务调度、中断处理和定时器管理等功能，则可使其实时性和确定性都有显著提高[1]。因为硬件逻辑电路独立于处理器运行，不占用处理器的处理时间，所节省的时间用于执行任务程序，从而提高了任务集合的可靠性、可调度性和实时性，具有一定的研究和使用价值。

1.2 课题研究的目的和意义

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，如果能够采用硬件逻辑实现 RTOS 任务调度器，将会带来很多优点。

首先，增强了系统运行的可靠性。任务调度器得以硬化后，其核心代码通常为只读的，因此系统在运行中的可靠性非常高，保证了用户对计算机的最基本需求，减少了因为异常而导致操作系统工作不正常的情况。

其次，操作系统的运行速度更快。其原理类似于BIOS的工作模式，硬件逻辑是物理并行执行的，所以与纯软件实现的实时操作系统调度内核相比，硬件调度内核的执行速度更快，可以最大限度提高系统的并行性和资源的利用率，而且这种优势在多任务并行和高时钟节拍的情况下将更加明显，更加提高了操作系统的实时反应速度和数据处理能力。

最后，方便用户进行管理。用户同操作系统进行交互是通过计算机管理文件进行的，对用户而言，当操作系统任务管理模块得以硬化后，仅仅需要保证计算机管理文件的正确性和可靠性即可。用户可以实时修改或者保存管理文件，提出一些的软硬件更新信息，从而使系统运行的更加流畅。

未来操作系统的发展方向是多核技术、家庭娱乐技术以及互联网技术等。安全性、易用性和稳定性将成为操作系统更为重要的特性[2]。将操作系统硬化，

有效地减少了管理计算机的成本，避免了复杂化，同时还能满足用户的各种需要。所以在计算机和互联网改变人们生活方式，使生产率得到提高的时候，操作系统任务调度器的硬化，将会使得这一进程得到加强和加速[3]。

实时操作系统任务调度器的硬化随着EDA(Electronic Design Automation)设计技术的发展、可编程逻辑器件及相关技术的提高，它在很多处理系统尤其是嵌入式系统与单片机系统中，将会有广泛的应用前景[4,5]。

1.3 实时操作系统任务调度研究现状及分析

1.3.1 国外研究现状

实时操作系统并不是一个新生的事物，从20世纪80年代起，国际上就有一些IT组织、公司开始进行商用嵌入式系统和专用操作系统的研发，任务调度算法一直是其研究的核心内容。

软件实现的实时操作系统国外有μC/OS-II、LynxOS和VxWorks等[6,7]。

μC/OS-II是美国嵌入式系统专家Jean J.Labrosse用C语言编写的源码公开的、抢先式多任务的强实时操作系统[8]。在μC/OS-II管理的64个任务中，任务的优先级的数值越高，表示该任务的优先级越低，在目前的版本中，任务的标识符相当于任务的优先级。μC/OS-II的任务调度算法为多任务按优先级抢占式调度，即系统一直在执行最高优先级的就绪态任务。在μC/OS-II系统中，所有任务的优先级必须不相同，优先级是任务的唯一标识。即使存在两个任务具有相同的重要性，它们的优先级也必须不同，这也就意味着低优先级的任务要想进入运行状态，必须等高优先级的任务进入等待或挂起状态，否则永远得不到执行。

LynxOS是一个分布式、嵌入式、可规模扩展的实时操作系统，LynxOS支持线程概念[9]。LynxOS内核是基于优先级抢占式的内核，LynxOS是以线程为运行实体的，线程也是LynxOS内核中调度的实体。线程列表中的线程在调度器中只有两种状态：运行状态和就绪状态。LynxOS内核共支持三种调度算法：FIFO(First In First Out)，RR(Round Robin)，DEFAULT。FIFO（先进先出）调度算法是指具有相同优先级的线程就绪时，首先运行就绪队列中队首的线程，然后依次运行。只有更高优先级的线程或当前线程主动放弃运行时，才会中断当前的线程。RR（时间片轮转）调度算法是指具有相同优先级的线程运行时，每个线程在CPU上运行一个固定的时间片，当线程用完分给它的时间片后，让

位于同优先级的其他线程。DEFAULT和RR类似，只不过它的时间片大小可以动态编程，每个线程可以不一样。线程的优先级和调度策略都可以通过程序设置并改变。

VxWorks系统调度策略是基于优先级抢占式的调度算法，调度内核总是将CPU分配给处于就绪状态的优先级最高的任务。当系统内核发现转变到就绪态任务的优先级比当前正在运行的任务的优先级高，内核立即重新调度，保存当前运行任务的上下文到其堆栈中，把当前正在运行的任务转变为就绪状态，插入到就绪队列中，然后运行就绪队列中的高优先级任务。VxWorks允许任务调用TaskPriority()函数动态改变自己的优先级。VxWorks系统可以将优先级抢占式的调度算法与轮转调度算法相结合，轮转调度算法试图让优先级相同的、处于就绪状态的任务公平地分享使用CPU资源[8]。

从20世纪80年代开始国外有人提出了硬件实时操作系统的概念，美国的Jaehwan Lee and Vincent John Mooney在分析比较了用软硬件分别实现RTOS的调度器之后，提出用专用的硬件IP核实现RTOS调度器，将会很大程度上提高RTOS的工作效率。任务调度是实时操作系统的核心，任务间的通信、外部事件的处理以及中断处理等都离不开任务调度的参与。而且随着实时操作系统功能的不断完善与增强，任务间的相互关系变得越来越复杂，需要与更多的外围设备进行交互，任务调度器需要不断的参与其中进行任务调度，从而导致系统性能的急剧下降，响应处理事件的实时性降低。任务调度则成为了提升RTOS性能的瓶颈，所以提高RTOS的整体性能则首先应当从提高任务调度的性能着手。将任务调度算法硬件化，无疑可以提升任务调度的性能，从而提高整个实时操作系统的性能[10-12]。

在20世纪90年代中期，日本丰田工业大学的Takumi Nakano教授曾开发出一款名为Silicon OS矽晶片，为了实现实时操作系统与处理器并行工作，利用VLSI技术将操作系统硬化到一块芯片上，进一步提高了实时操作系统的实时性和高可靠性。所谓Silicon OS是指，将传统的实时操作系统内核uITRON分为Micro Kernel、Interface和Silicon TRON三部分。采用HDL语言实现硬核中的任务管理、中断处理和任务间的相互通信等纯软件实现的实时操作系统的主要任务。Silicon TRON在硬核中同时处理来自外部中断的异步请求和来自系统调用的同步请求。根据以上条件，任务调度器根据任务的就绪状态通过硬核上的调度算法以最快的方式选择要执行的任务。当选择出的新任务不是当前正在执行的任务时，Silicon TRON发出中断信号请求执行任务切换。使用Silicon TRON操作系统调度程序只需根据执行的优先级别进行调度，但在通常的系统有中断处理程

序和任务两种类型优先事项[13,14]。Silicon OS硬核采用硬件语言SFL设计，逻辑综合采用0.8um CMOS工艺，当资源数和任务数是16时，STRON-I用到的逻辑门数是40K，每个对象诸如：任务，事件控制块，信号量，时钟，调度和控制用到的逻辑门数本别为：21933，390，2010，3913，8791和1439，总的HDL代码是8059行[15-17]。

1.3.2 国内研究现状

从计算机工业的发展趋势看，集成电路的规模越来越大，并且造价也在不断降低，以硬件代替软件已经成为发展趋势，所以发展硬件是计算机发展的一个主要方向。

目前，国内研究硬件实时操作系统方面论文并不多，其中，东北大学的尹震宇博士、赵海教授、徐久强教授等提出了一种基于硬件操作系统(HOS)的设计结构，通过硬件实现了以往由操作系统复杂软件实现的系统调度、控制等处理过程[18]。通过在51处理器内核的基础上添加任务调度等HOS设计，在Xilinx公司XC2S300E FPGA中进行测试[19,20]，实现了一款针对低端家电嵌入式系统带有HOS支持的处理器[21]。尹震宇博士、赵海教授提出一种硬操作系统(HardwareOS，HOS)的概念，通过添加微码处理逻辑及硬件处理模块与处理器并行执行，在机器指令级上实现处理器线程级别的并发控制、内存管理机制以及为某些特定应用环境而实现的处理功能等。在处理器硬件上实现以往通过操作系统软件实现的部分机制，简化用户开发嵌入式软件的复杂度[22]。通过基于硬件的调度器来实现多线程的调度管理，实现基于硬件的时间片轮询调度算法。简化了在多线程条件下用户编程的复杂度，提高了处理器执行多线程时的整体效率，减少了线程调度在操作系统中的花销，增强了处理器在多线程运行环境下对线程的保护。

辽东学院计算中心张大伟采用了Xilinx公司的Virtex-IIPro系列的XC2VP30芯片，实现硬件调度器调度方法[23]。硬件调度器主要由三部分组成：调度核、任务管理器、通信接口。通信接口主要完成CPU和调度核之间指令的翻译转换，实现CPU与调度核之间的通信。调度核控制任务管理器，将处于就绪态优先级最高的任务和等待挂起的任务发送给任务管理器，同时与CPU进行信息交互，从CPU获取系统时间等各种有用信息。任务管理器则负责任务状态的相互转换，将调度核发送过来的处于就绪态优先级最高的任务标志为运行态，交由调度核并发送给CPU，将等待挂起的任务置入等待队列。完全用专用的硬

件 IP 核实现。

中国科技大学的周学海教授提出了一种硬件多线程处理器的方法[24]。研究目前已有的硬件多线程处理器之后，根据流水线的执行过程，准确地分析了流水线深度、线程切换频度等对硬件多线程处理器性能的影响，提出了一种提高硬件多线程微处理器性能的方法，可以彻底消除线程切换过程中的开销，从而提高硬件多线程处理器的性能。

浙江大学的陈天洲教授提出了一种基于 CPU/FPGA 混合架构上的硬件线程执行机制的方法[25]，利用 Scratch-Pad 存储器[26,27]，在 CPU/FPGA 混合架构上做软硬件协同设计，硬件线程以 Scratch-Pad 作为共享数据存储区，减少软硬件之间的数据通信量，并且提升了软件硬化的可操作性。利用 VHDL 来描述硬件线程执行逻辑，最终在 FPGA 上实现该硬件线程执行过程。

解放军信息工程学院的崔建华，孙红胜，王保进提出了基于外部处理器十 FPGA 的硬件平台结构，在 FPGA 上设计一种简单硬件实时操作系统[28]，在 Actel 公司的 APA075 实验开发板上实现了任务调度、中断管理、定时器管理等实时操作系统基本功能。任务调度内核是实时操作系统的核心，负责任务的管理和调度；中断管理模块负责响应管理外部的中断；定时器模块负责设定任务周期和任务延时的管理。任务控制块队列通过使用 FPGA 的片内寄存器来实现，为硬件任务调度内核所应用，调度内核通过多级比较器，能确定当前时刻处于最高优先级的就绪态任务，并将该任务的信息通过接口总线发送到处理器。因为硬件逻辑是物理上并行执行的，所以与纯软件实现的实时操作系统相比，硬件调度内核的执行速度更快[29]。

国内外的研究现状总结：主要从硬件和软件两个方面来提高实时操作系统的高可靠性和实时性等性能。提高实时性的方法很多都是依靠软件来实现，但也可以依靠硬件来提高操作系统的实时性。为了提高软件的可重用性，更为了减少上市时间的压力，人们越来越多的通过软件方面来实现实时操作系统。但是，实时任务管理中的事件管理、时间管理、任务调度和任务切换等，给实时操作系统带来了很大的开销，这些开销通过软件技术已经很难减少，这形成了提高 RTOS 性能的瓶颈。人们为解决这个瓶颈，研究把实时操作系统的一部分功能通过硬件来实现，从而达到提高可靠性和实时性的目的，进而解决这个瓶颈。将操作系统任务管理器硬化到 FPGA 上，使它成为一个硬核芯片与处理器并行执行，硬件实现任务管理器无疑将降低处理器系统开销，从而提高实时系统任务集合的可调度性，具有一定的研究价值，但目前的研究主要集中在基于简单的静态优先级任务管理调度的硬化，更加合理的任务调度算法、任务间的

通信和用硬件实现上下文交换还有待研究。

1.4 课题主要研究内容

本论文以轻型实时操作系统 μC/OS-II 为研究对象，学习实时操作系统的内核结构，认真分析其内核运行机制，掌握实时操作系统的工作原理，主要利用 FPGA 技术实现操作系统的硬件任务调度器。文中修改了 μC/OS-II 中的一些数据结构，使之更有利于发挥硬件的并行性，提出了由输入寄存器、输出寄存器和控制器等功能模块组成的硬件任务调度器。本文主要研究提高操作系统实时性的任务调度算法，并利用 FPGA 技术完成任务管理器的硬件实现，其主要内容如下：

1. 分析提高嵌入式系统实时性的途径，基于软件的实时任务管理器的构架，总结局限性及其改进思路。

2. 任务管理模块的硬件设计。任务调度机制是嵌入式实时操作系统的核心技术。以 μC/OS-II 内核为基础，划分任务管理模块的硬件逻辑，改进其任务调度算法，根据任务管理的系统调用设计相应的硬件逻辑电路。

3. 信号量管理模块的硬件设计。分析 μC/OS-II 中信号量管理的原理，修改其数据结构，设计硬件信号量管理模块，实现信号量管理的申请信号量和释放信号量的 P/V 操作。

4. 深入了解 FPGA 的设计流程，设计各个功能模块，首先用 VHDL 语言进行功能描述，然后下载到在 Xilinx 公司的 FPGA 平台上进行语法检查，功能仿真，在线调试，验证设计的可行性。

1.5 课题主要工作

全文共分五章，具体结构安排如下：第 1 章是绪论，主要介绍了课题的来源，研究背景及意义，国内外的研究现状等。第 2 章分析了基于软件的实时任务管理器的构架，针对如提高嵌入式系统处理器的实时性和性价比,剖析了基于实时操作系统的实时任务管理器的机理，简要分析了这些技术的局限性，提出提高嵌入式系统的实时性改进思路。第 3 章分析了 μC/OS-II 的任务管理和调度，提出硬件实时操作系统的结构和运行原理。第 4 章对实时操作系统中的任务管理和信号量管理进行了硬件设计。第 5 章是功能时序仿真，并对仿真做了详细的分析，验证所提出的方案的可行性。最后，总结本文内容，展望下一步工作。

第2章 操作系统的实时性

2.1 引言

实时性是实时操作系统中非常重要的一个性能。随着嵌入式产品上市时间的缩短和嵌入式系统的应用愈来愈广泛，嵌入式操作系统得到越来越多的使用，但是，基于软件的实时操作系统本身要额外占用处理器资源，又制约了操作系统实时性的进一步提高。为了提高嵌入式系统的实时性，人们采用了很多方法。

可以通过软件级并行技术来提高实时操作系统的实时性，但采用硬件级的并行技术方法更有效，特别是在CPU中的并行技术，对于提高一般体系结构的处理器性能的方法主要有三种：提高主频率；优化处理器的内部结构，从而降低每条指令的平均时钟周期数；编译器对程序进行优化，进而减少生成指令的数量，或者是降低所生成的指令的平均时钟周期数。

在很长一段时间内，提升处理器的主频遵循莫尔定律，处理器的主频很短的时间内就能翻一翻。然而，最新研究资料表明：由于门级和线路的延迟已经逼进极限，处理器的主频已经很难得意提高。所以，单纯的依靠提高处理器的主频和提高硬件的速度来提高计算机性能的方法已经成为过去，这些方法已经渐渐没有潜力可挖。并且提高处理器主频大大增加了功耗，这对对有低功耗要求的嵌入式系统不太适合。

随着微电子技术的发展集成电路的集成度越来越高，这样，硬件并行性在单个芯片内将得到巨大的发展，操作系统的并行计算技术可以按并行粒度的粗细进行划分，并行粒度可以从粗到细划分为如下等级：软件级并行、多处理器并行、多线程并行、指令级并行等[30]。

2.2 提高嵌入式系统实时性的途径

2.2.1 软件级并行

最粗粒度的并行计算是软件级并行，它是采用操作系统来运行多任务的伪并行，CPU被多个任务分时地占用，通常的调度方法有：时间片轮询调度和优先级抢占式调度。在实时嵌入式操作系统中，一般采用优先级抢占式调度的实

时操作系统，采用中断的方式来解决一些对实时性要求较高的任务。操作系统的实时性受任务调度算法的时间、任务上下文的切换时间和实时操作系统的时钟节拍限制。

任务切换一般发生在如下几种情景：高优先级任务抢占低优先级任务运行；中断服务程序时，响应中断任务；任务时间片完成后，执行新的时间片任务。任务切换过程增加了应用程序的额外负荷，CPU的内部寄存器越多，额外负荷就越重。软件级并行的实时操作系统本身要占用处理器的执行时间。

2.2.2 处理器级并行

处理器级并行是粗粒度的并行计算，多处理器操作系统在许多方面沿用单处理器多道程序分时操作系统所具有的功能。如资源分配、资源管理、存储器管理和保护、死锁防止和处理、异常终止或例外处理，等等。

单处理器多道程序系统主要是为用户建立多个虚拟处理器来模拟多处理器的环境，让程序能并发执行，以提高资源的利用率和系统的处理能力。在多处理中，实际有多台处理器，就可以真正实现多个进程的并行执行。所以，多处理器操作系统希望能增强程序执行的并行性，以获得更高的系统处理能力和处理速度。

在嵌入式软件层级的任务处理上，不同任务的运行占用不同的指令流，每个处理器核处理一条指令流，利用不同指令流中的空间并行性来实现多处理器的并行技术。采用多处理器实现并行技术，虽然整体的运算速度得到了提高，但是处于优先级较高的单个任务，由于其只能运行在单个的处理器上，任务上下文切换所需要的开销并没有得到减少，任务的执行速度也没有提高，所以处于优先级较高的任务的实时性没有获得提高。

多处理器并行技术也存在不少缺点，多处理器并行的资源利用率相对不高，如取指单元和逻辑处理等很多资源，在多处理器并行中并不能得到共享。还有，处理器需要等待缓存重填其缺失，才能得以运行。多处理器并行技术的性价比并没有得到提高，其成本会随着处理器核的增多而呈线性增长。单位硅面积的利用率降低了，但是其功耗却因为硅面积的利用率的降低而增大了。多处理器级并行技术仅仅有限地提高了系统的实时性，对于那些要求低功耗、高性价比、高实时性的实时嵌入式操作系统来说，多处理器并行技术并不十分适合。

处理器级并行还有另外一种方式：主处理器+协处理器模式，协处理器用于减轻主处理器的处理任务，主处理器处理指令，协处理器可以通过提供配置寄

存器来扩展内核的处理能力，协处理器可以有多个。典型的协处理器有：数学协处理器和图形协处理器等。

在实时嵌入式操作系统中，采用多处理器并行技术并没有使任务上下文切换的开销得以改善，要想提高系统的实时性，有效地减小任务上下文切换的开销，人们必须研发新的计算技术。线程级并行技术便是在这种条件下应运而生。目前线程级并行技术已经成为计算机体系结构领域的一个重要的分支。

2.2.3 线程级并行

线程级并行技术是中等粒度的并行计算，在处理器中实现多线程的并行计算。开发线程级并行技术的代价并不是很高，主要是因为程序中的线程存在固有的并行性。软件级并行技术中任务上下文切换的开销通过线程级并行技术可以有效地得以缓解，线程级并行有效地提高了系统的实时性和处理器的执行效率。

多线程(Multithreading)并行是指允许多个线程以重叠的方式共享一个单独处理器的功能单元。处理器复制了每个线程独立的状态来有效地支持这种共享。例如每个线程需要系统提供单独的程序计数器，单独的寄存器副本，以及单独的页表。存储器共享是通过虚拟内存机制获得的，该机制同时支持多道程序设计。但是，相应的硬件逻辑必须满足相对较快的线程切换的需求，线程切换远比进程切换更有效，进程切换往往需要上万个处理器时钟周期。

在实时嵌入式操作系统中，多线程并行技术通常采用实时操作系统来实现，每个线程分时占用处理器。为了保护每个线程独有的数据信息，在线程切换时，都要进行不同线程的上下文进出堆栈的切换，上下文的切换占用处理器的时间，访问存储器的速度比处理器的运行速度慢的多，所以当访问存储器时，处理器就会出现等待的情况。线程级并行技术通过实时操作系统来实现几乎不用增加多少硬件成本，但线程切换的开销大，细粒度的等待无法消除，如由缓存缺失引起的处理器等待。

多线程并行技术是通过处理器设置多个寄存器组来实现，寄存器组保存每个线程的上下文，包括程序指针、执行状态和临时数据，这样就可以快速地进行多线程之间的上下文切换。多线程并行技术提高了处理器的利用率，使处理器的处理单元处于满负荷运行状态，而代价仅仅是增加有限的硅面积，总体来说还是大大提高了处理器的性价比，这对实时嵌入式操作系统来说极为重要。

2.2.4 指令级并行

指令级并行是细粒度的并行计算技术，程序员不需要知道指令级并行是怎么实现的，因为指令级并行技术对程序员来说是透明的。处理器的各种性能通过指令级并行技术能得到有效地提高。

通过提高处理器指令级并行性的能力可以提高指令级并行性。处理器在每个时钟周期执行多条指令的能力被称为指令级并行性。处理器希望在每个时钟周期都能处理尽可能多的指令条数，因此要求处理器能够标识出程序中可以并行执行的指令，并且具有足够的资源来执行可并行执行的多条指令。

根据不同的指令调度技术已经产生了多种新型的指令级并行结构，有超标量(Superscalar)、超长指令字(VLIW, Very Long Instruction Word)、超流水线(Superpipelining)等。

超标量(Superscalar)：超标量处理器是采用多指令流水线，每个周期可发射多条指令，且每个周期可由多条流水线生成多个结果。只有相互独立的指令可并行执行而不会导致等待状态。在超标量处理器中，配置多套功能部件、指令译码电路和多组总线，并且寄存器也备有多个端口和多组总线。超标量处理器主要靠编译程序来优化编排指令的执行顺序，将能并行的指令搭配成组，硬件不对指令顺序进行调整。在超标量处理器中，主要采用动态指令调度技术。

超长指令字(VLIW)：在VLIW的处理器中，在编译时，编译程序找出指令间潜在的并行性，将多个能并行执行的不相关或无关的操作先行压缩组合在一起，形成一条有多个操作段的超长指令。运行时，不再用软件或硬件来检测其并行性，直接由这条超长字指令控制机器中多个相互独立的功能部件并行操作。每个操作段控制其中的一个功能部件，相当于同时执行多条指令。因此，硬件结构和指令系统简单，是一种单指令多操作码多数据的系统结构。VLIW处理机能否成功很大程度上取决于代码压缩的效率，其结构的目标码与一般的计算机不兼容。

超流水线(Superpipelining):超流水线处理机着重开发时间并行性，在公共的硬部件上采用较短的时钟周期，深度流水来提高速度，需使用多相高频时钟，时钟频率高达100MHz到500MHz。没有高速时钟机制，超流水线处理机是无法实现的。

指令级并行技术对程序员来说是透明的，程序员不需要关心指令级并行技术是如何实现的，指令级并行有效地提高了处理器的性能。

2.3 基于 RTOS 的实时任务管理

2.3.1 任务管理

事件的类型有三种：周期性事件、非周期性事件和非周期性发生的周期性事件。我们把一个事件看作一个任务，则有两种执行方式：顺序执行和并发执行。

顺序执行是任务独占系统资源，按照顺序一个、一个地执行，具有封闭性，不手外界干扰。并发执行是几个任务竞争CPU并发执行，具有动态的特征。

一个任务主要由以下三部分组成：任务控制块TCB(Task Control Block)、任务执行的程序代码和任务加工处理的数据集。任务是一个独立的实体，断断续续地执行，可能处于以下四种基本状态之一，睡眠态、就绪态、运行态和挂起态。

任务从建立到消亡，任务的状态会在以上四种状态之间发生转变，任务状态的转变会引起任务的调度。任务状态改变的原因大部分是由于外部事件引起的，还可能受调度内核提供的一些系统函数的影响。任务控制块是系统对任务进行管理和控制的依据，它记录了任务的各种活动状态。当建立任务的时候，操作系统会给每个任务分配一个任务控制块；当任务消亡的时候，相应的任务控制块所占用的资源会被释放还给系统。

2.3.2 任务调度

操作系统进行任务调度的目的是让相应事件得到及时响应并处理，让处理器处理优先级高的任务。任务调度算法基于优先级可以大略分为以下三种：速率单调RM(Rate Monotonic)算法；最早截止时间优先EDF(Earliest Deadline First)算法；可达截止期最早优先EFDF(Earliest Feasible Deadline First)算法。

1．**单调速率算法** 速率单调算法是一种适用于可抢占的硬实时周期性任务调度的静态优先级调度算法。单调速率算法事先为每个任务分配一个与事件频率成正比的优先级，依据任务周期的长短对相应优先级进行顺序排列，任务的周期越长，其相应的优先级越低，任务调度时总是运行周期最短的任务[31]。

需要任务调度有两种情形：当处于运行态的任务，因为某种原因而挂起后，需要从就绪队列中取出首元素，把运行任务的指针指向该元素。假设任务由其它状态转化为就绪态，任务调度器比较运行任务和就绪队列首元素，优先级高

的任务占有处理器运行[32]。

速率单调算法相对简单，但是效率较高，任务的优先级是根据相应的周期来指定的，可以依据任务的紧急程度来决策随机事件的任务的优先级。

2. 截止期最早优先算法 截止期最早优先算法是根据任务的开始截止时间来确定任务的优先级。截止时间越早，其优先级越高。该算法要求在系统中保持一个实时任务队列，该队列按各任务截止时间的早晚排序，具有最早截止时间的任务排在队列的最前面。调度程序在选择任务时，总是选择就绪队列中的第一个任务，为之分配处理器，是指投入运行。截止期最早优先算法存在一定的不去，已经过了截止期或者马上要过截止期的任务，由于其处于最高优先级，仍然得到了运行[33]。

3. 可达截止期最早优先算法 可达截止期最早最优先算法是对截止期最早优先算法的改进，依据任务截止期的顺序对处于就绪的任务进行排列，具有最早的可达截止期任务的优先级最高，在任务调度的时候，已经超过截止期的任务不被调度。如果一个事务t的截止期是当前时间可达到的，乃指$t+(E-P)\leq d$，这里t为当前时间，E、P分别为事务T的执行时间估算和已执行时间，d为其截止期，也就是说，任务预计能在截止时刻前完成。

在可达截止期最早优先算法中，运行任务的时间由系统时钟来累计，把空闲任务的截至时刻设置为无限大，把其估算时间设为0，从而在进行任务调度时，至少有一个任务在就绪队列中满足任务调度的需求。截至时刻决定了任务的优先级，所以一些不可达的任务可能没有得到处理而消亡了。

2.3.3 任务切换

在实时操作系统中，存在着很多的任务，一些任务并不与外部时间相关，硬件方面的相关中断也是有限的，所以需要一种软件的方式来模拟中断，任务就是中断的一种补充，相反，我们也可以把中断服务程序看成是具有较高优先级的任务。

调度器确定更重要的任务该运行了，于是调用任务切换函数，任务的上下文其实就是CPU中的全部寄存器内容，任务切换代码需恢复该任务在CPU使用权被剥夺时保存下来的全部寄存器的值，以便让这个任务继续运行。

任务切换主要完成以下功能，当任务调度器决定停止任务1的运行，并开始运行任务2时，其切换过程如下：将处理器寄存器内容推入到任务1的堆栈中；将任务2寄存器值从栈中恢复到处理器的寄存器中。

任务切换时间是指从一个任务切换到另一个任务所需要的时间。任务切换时间相对于任务执行时间是很短的，但是如果切换过于频繁，其开销也不可小看，所以设计应用程序时，要避免过度的任务切换。

当执行应用程序进行任务处理时，任务调度器决定是否进行任务切换，需要切换时，则调用任务分配器进行任务切换。任务的上下文切换和改变执行指令流等均由任务分配器完成。执行指令流主要由三部分构成：应用程序、中断服务程序和内核程序。当系统函数被一个任务或中断服务程序调用时，则执行内核的系统函数，当执行指令流退出内核时，任务分配器把控制权交给应用程序中的任务。任务调度器中的调度算法决定任务的执行顺序。任务调度器在任务进行系统调用时被作为一个函数，在系统调用要结束时被系统函数调用。当中断服务程序进行系统调用时，任务调度器要等到中断服务程序完全执行完才被调用。

2.4 本章小结

本章以上提出的技术都有其局限性。基于软件实现的实时任务管理，其任务调度和任务切换的开销已经成为改善系统实时性的瓶颈。处理器与实时操作系统这两种技术的跨度比较大，能够同时很好的掌握这两种技术的研究人员不多。多线程处理器已经对任务切换的开销降低到了很小，但任务调度和事件管理还需要软件实现。如果能用硬件的方法来实现实时操作系统中的大多数功能，特别是任务调度和任务切换方面，将能很大程度地提高操作系统的实时性。

第3章 任务管理的硬件设计

3.1 引言

实时性是实时嵌入式操作系统很重要的特性，为了提高这一特性，可以从硬件和软件两个方面解决。提高实时性的很多方法可以用软件，也可以用硬件来实现，其性价比决定了最终使用哪种方法。随着半导体技术的不断发展，硬件的成本在越来越下降，硬件的方法被更多的采用来实现原理用软件解决的功能。

实时操作系统被主要应用在软件方面，因此可以提高软件的可重用性，并且减少了上市时间的压力。但是，基于软件实现的实时操作系统的并行计算技术已经很难减少实时任务管理中的任务调度、时间管理、事件管理、任务切换等带来了一定的开销，这形成了提高实时操作系统性能的瓶颈。研究人员开始用硬件的方法来解决上述的瓶颈问题，把实时操作系统的部分功能用硬件来实现，来达到提高实时性的目的。把用硬件实现的部分称为实时任务管理器(RTM，Real-Time Task Manager)，通常包括任务调度、时间管理、事件管理、任务切换等部分功能。

3.2 μC/OS-II 的任务管理和调度

3.2.1 μC/OS-II 的任务及其控制结构

1. μC/OS-II 操作系统的控制结构 操作系统主要负责资源管理和分配程序，其主要功能是保持程序的正确执行，并且合理分配进程在执行过程中所需要的各种资源。所以，进程是操作系统的控制结构核心，对各种资源的数据结构的管理都应该围绕进程展开。操作系统应该了解每个进程和资源的目前状态，这样才能够合理地管理进程和资源。操作系统通过构造一组表来有效地管理和维护进程和资源的相关信息。

2. μC/OS-II 中的任务 想要完成一个复杂的应用程序时，往往是通过把这个大的任务分割成众多的小任务，然后完成这些众多的小任务，进而最终完成了这个大任务。通过这种做法应用程序变得更加地容易维护。μC/OS-II 中所指

的小任务就是进程，μC/OS-II 是一个能对这些小任务合理地进行管理和调度的多任务操作系统[34]。

从设计应用程序的角度看，可以把 μC/OS-II 中的任务看作为线程，是能够解决问题的 C 语言函数和与其相关的数据结构构成的实体。从任务被存储的结构来看，主要把 μC/OS-II 中的任务分为三个部分，任务控制块、任务的堆栈和任务的程序代码。任务的各种属性由任务控制块来保存，任务的上下文执行环境放在任务的堆栈中保存，任务的程序代码是任务所要执行的部分[35]。为了方便管理，μC/OS-II 把任务链接成一个任务链表，如图 3-1 所示。

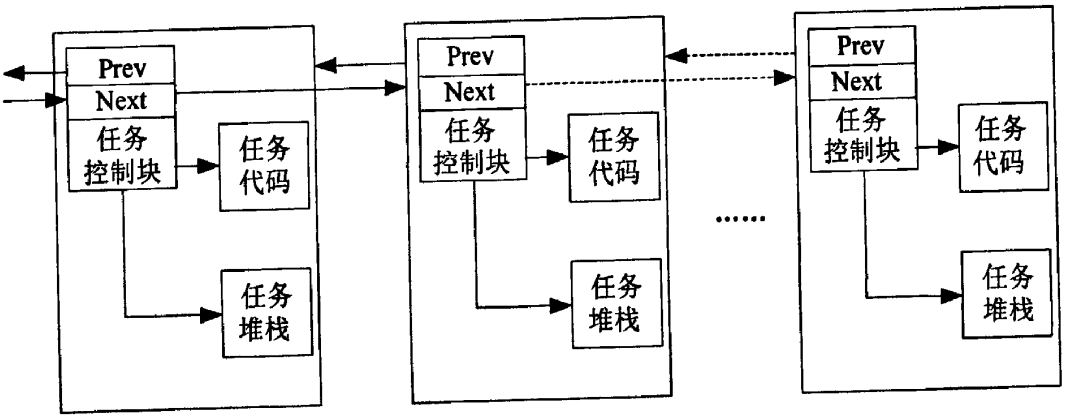

图3-1 任务在内存中的结构

Fig. 3-1 The task structure in memory

3.2.2 μC/OS-II 的任务管理

μC/OS-II中最多可以管理64个任务，每个任务的优先级必须不同，所以共有64个优先级，并且每个任务都有独立的堆栈空间。一个任务的优先级的值越高，反映出这个任务的优先级越低。创建每个任务的时候，都有一个相应的任务控制块TCB被系统初始化，TCB在操作系统中相当于任务的身份证。μC/OS-II的TCB中包含了任务的ID号、任务的堆栈指针、任务的当前状态和任务的优先级等信息。μC/OS-II的TCB数据结构定义如下：

```
typedef struct os_tcb{
 OS_STK   *OSTCBStkPtr;          //任务的堆栈指针
 #if OS TASK CREATE_EXT_EN       //控制在外部对任务控制块进行扩充
 OS_STK   *OSTCBStkBottom;       //指向任务堆栈的底部
 INT32U   OSTCBStkSize;          //任务堆栈的大小
 INT16U   OSTCBId;               //任务的 ID 号
```

```
#endif
INT8U    OSTCBStat;              //任务的状态
INT8U    OSTCBPrio;              //任务的优先级
struct os_tcb   *OSTCBNext;      //实现任务控制块双链表
struct os_tcb   *OSTCBPrev;
.....
INT16U OSTCBDly;                 //任务的延时时钟滴答数
}OS_TCB;
```

μC/OS-II 中的任务控制块有一些自己独特的特点，可以通过上述 TCB 的定义得知。其中包含了宏定义控制的条件编译，用户可以根据具体的时间情况对 TCB 进行裁剪，以达到节省存储空间的目的。每个任务都有自己单独的堆栈空间，并能够对其进行合理的管理。通过 TCB 可以随时查看任务的当前状态，提高了任务管理器的任务调度效率。

在 μC/OS-II 中，每个任务可以有如下五种状态：休眠态(dormant)、就绪态(Ready)、运行态(Running)、等待或挂起态(Pending)和中断态(Interrupt)。任务状态转换如图 3-2 所示。

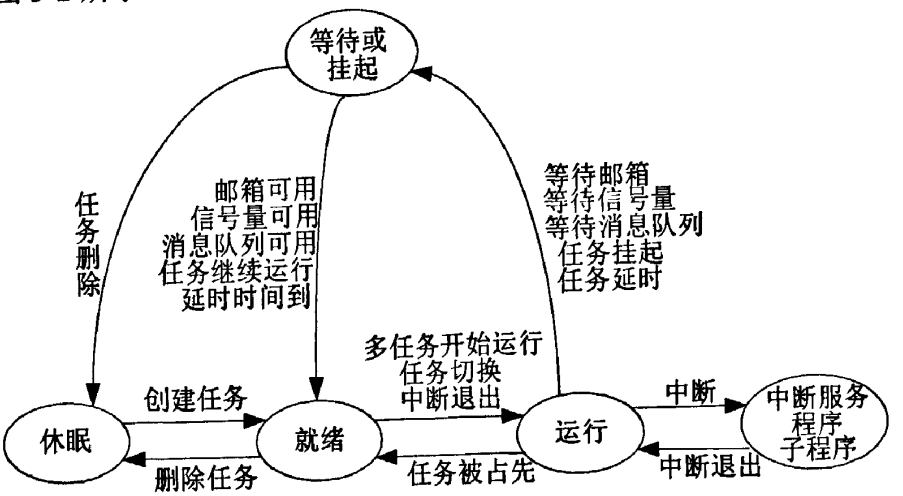

图3-2 任务状态转换图

Fig. 3-2 The task state transition diagram

3.2.3 μC/OS-II 的任务调度

1．μC/OS-II 的调度策略 嵌入式操作系统 μC/OS-II 的任务调度思想是：总是让处于就绪态的最高优先级任务运行。为了时刻保证这一调度思想，总是在

任务调用系统函数后或者执行完中断服务程序后，调度器决策出优先级最高的就绪任务并运行该任务。μC/OS-II 采用的是可抢占式最高优先级的调度算法，该算法的关键在于如何用最短的时间决策出处于就绪态的优先级最高的任务，以便进行任务的上下文切换。对于查找优先级最高的任务，最简单且易于实现的方法是顺序检索，即把系统中的所有任务根据优先级从高到低进行排序，查找优先级最高的就绪任务时，可以从队列的头节点开始向下查找，遇到的第一个处于就绪态的任务就是要查找的任务。但是，这个顺序检索的方法具有明显的缺点，查找不同优先级任务所花费的系统时间是不同的，当系统中存在较多任务时，顺序检索会影响到任务调度的效率，很大程度上影响了这个系统的实时性。所以，μC/OS-II 通过使用哈希表来确定处于就绪态的优先级最高的任务，这也是 μC/OS-II 任务调度算法的最大特点之一。μC/OS-II 通过哈希表法，实现了快速查找处于就绪态的优先级最高任务。

2. μC/OS-II 的任务就绪表 μC/OS-II 通过在 RAM 上建立了一个记录表，可以使系统能够明确地知道哪些任务已经处于就绪状态，哪些任务没有处于就绪状态，每个在系统中的任务都会在这个记录表中占据一位，并且用 1 和 0 分别表示任务处于就绪态或是非就绪状态，这个记录表被称为任务就绪表。μC/OS-II 的调度器进行任务调度的依据就是任务就绪表，任务就绪表的结构如图 3-3 所示。

在 μC/OS-II 中用一个数据类型为 INT8U 的数组 OSRdyTbl[]来存储任务就绪表，如图 3-3 右侧的 8*8 阵列共有 64 个任务。这个数组存了 8 个元素，每个数组元素表示了 8 个任务是否就绪的状态，所以可以把每 8 个任务看作是一个任务组。为了更加方便对就绪任务的查找，μC/OS-II 中又定义了一个 8 位的变量 OSRdyGrp，这个变量的每一位分别对应 OSRdyTbl[]的一个任务组。如果 OSRdyTbl[]的某个任务组中有任务为就绪状态，则 OSRdyGrp 的相应位为 1。例如 OSRdyGrp=01000010，则表示 OSRdyTbl[1]和 OSRdyTbl[6]任务组中有任务处于就绪状态。

3. μC/OS-II 的任务调度 在 μC/OS-II 中，任务调度主要是通过任务调度器来完成。任务调度器主要负责两项工作：一是通过任务就绪表找到处于就绪态的优先级最高的任务；而是实现任务的上下文切换。

因为操作系统主要是通过任务的 TCB 来管理任务，所以任务调度器在做任务上下文切换的工作前要首先获得待运行任务的 TCB 指针和当前任务的 TCB 指针。

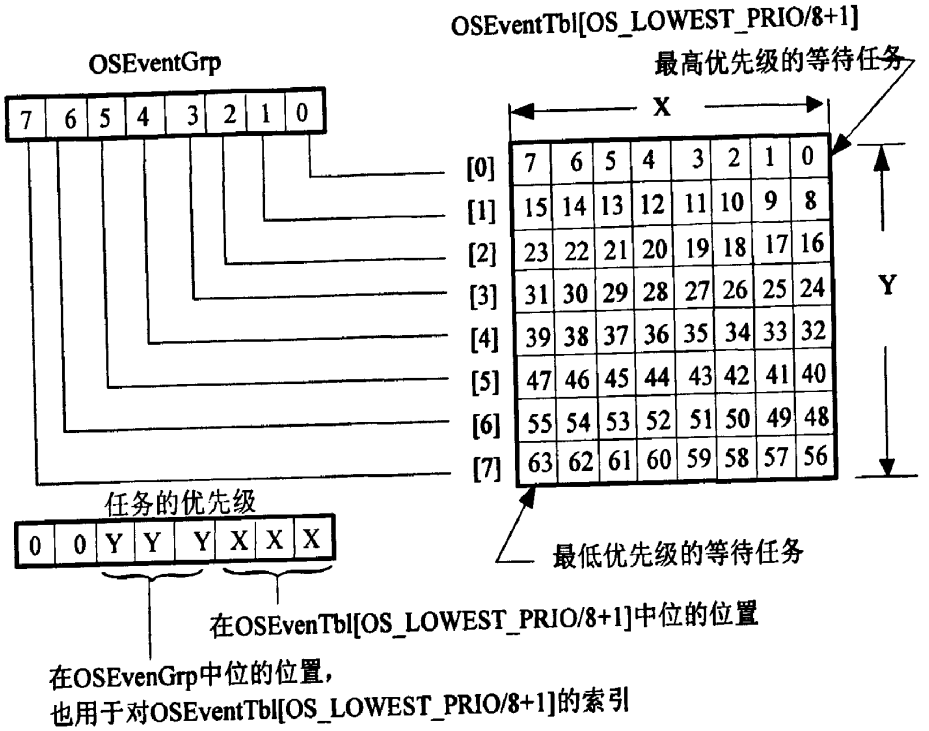

图3-3 任务就绪表

Fig. 3-3 The task ready list

任务调度函数结构如下[36]：

```
void OS_Sched(void)
{ 关中断;
 if(不是中断服务子程序而且并未调用任务调度上锁函数)
   { 查找处于就绪态的最高优先级任务;
     if(优先级最高就绪任务不是当前执行的任务)
       {调用 OS_TASK_SW();}
   }
   开中断;
}
```

OS_TASK_SW()函数主要完成任务的上下文切换。任务的上下文切换实际就是对任务当前运行数据的切换，也就是对当前处理器中堆栈数据的切换。被中止的运行任务的堆栈指针应该保存到该任务的 TCB 中，待运行任务的堆栈指针应由该任务的 TCB 中转存到处理器的 SP 中。简单地表示，就是 μC/OS-II 切换任务时，恢复被切换任务的运行环境，恢复处理器中的堆栈数据。任务调度

器的任务上下文切换的工作流程如图 3-4 所示。

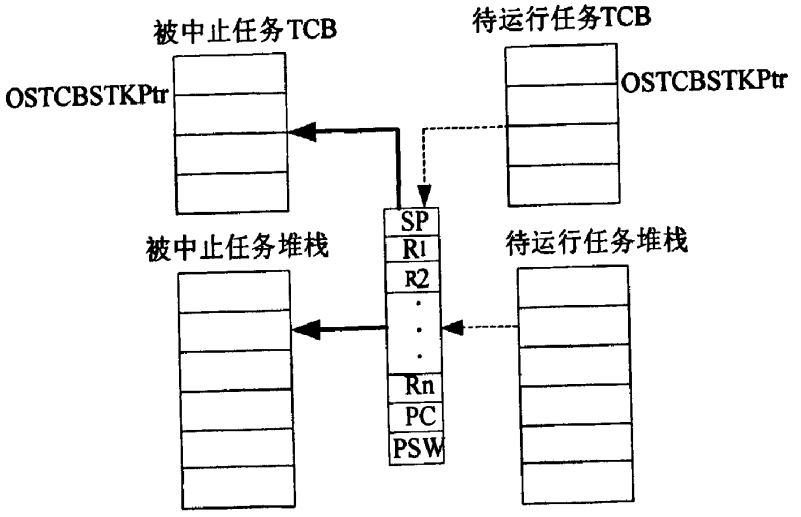

图3-4 调度器的任务切换

Fig. 3-4 Task switching for scheduler

3.3 硬件实时操作系统的结构和运行原理

3.3.1 硬件实时操作系统的提出

实时性和可靠性是实时操作系统所强调的两大特性，这两个特性除了与 CPU 的速度和访问存储器的速度这些计算机硬件有关，还与实时操作系统的软件相关。通常硬件是实时的，而软件却未必。实时系统中的软件是由实时系统应用软件和实时操作系统两部分有机结合而形成的，其中实时操作系统管理并协调各项工作，起着主导作用，主要为应用软件提供良好的运行环境和开发环境[37]。实时操作系统是各种系统调用的集合，是底层的程序模块，应用程序所要求的任务都由这些系统调用程序去一一完成，软、硬件实时操作系统系统调用的执行流程如图 3-5 所示[38]。

软件实时操作系统的系统调用的执行流程如下：

1. 应用程序分析任务目的，进行参数设置，调用实时操作系统中的系统函数来完成任务。

2. 处理器通过执行实时操作系统的系统调用来完成任务调度程序，进而完成应用程序的功能。

3. 当处理器需要进行任务切换时，则执行实时操作系统的上下文切换函数。

4．最后，将实时操作系统的执行结果返回给应用程序。

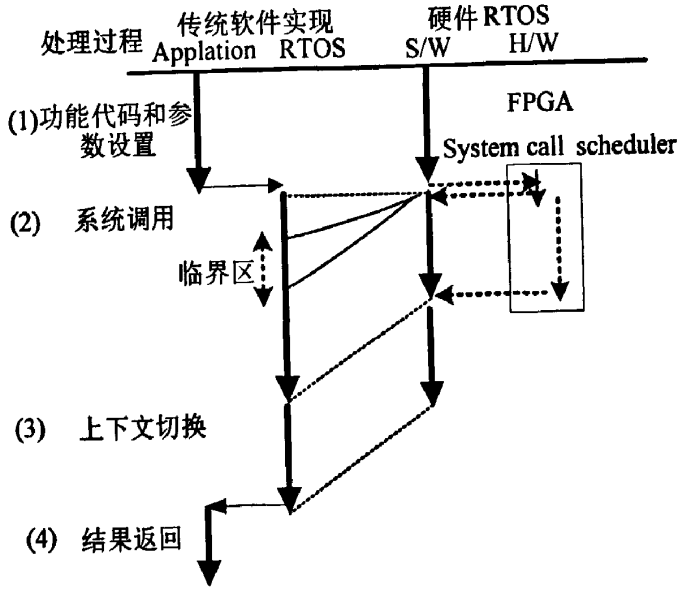

图3-5 软、硬件RTOS系统调用处理流程

Fig. 3-5 Software and hardware RTOS processing of system calls

对软件实时操作系统的执行流程进行分析，发现影响应用程序执行速度的主要因素是软件实时操作系统系统调用的执行时间。通过软件实现的实时操作系统中，操作系统的内核的系统函数和应用程序统一编译运行，它们在处理器中是串行执行的，当内核的系统调用函数被处理器执行时，应用程序必须被迫停止下来，将处理器让给实时操作系统内核的系统调用函数，直到内核的系统调用函数返回结果，应用程序才得以继续运行。因此在软件实现的实时操作系统中，应用程序的执行速度受操作系统内核的系统调用函数的影响很大。

当使用硬件来实现实时操作系统的复杂多线程程序时，一定会提高实时操作系统的实时性，因为硬件逻辑是与处理器并行执行，独立于处理器运行操作系统内核的系统调用函数，硬件实时操作系统的工作过程如图 3-6 所示。

硬件实时操作系统的执行流程如下：

1．应用程序调用系统服务。系统服务对用户是透明的，由操作系统内核的系统函数实现，调用相应的系统函数来完成响应的服务，并不需要了解函数实现的细节。

2．系统接口函数被系统调用，通过数据总线将功能代码和参数设置传给相应电路的输入寄存器。

3．系统调用由硬件逻辑单元根据输入寄存器中的内容来执行。

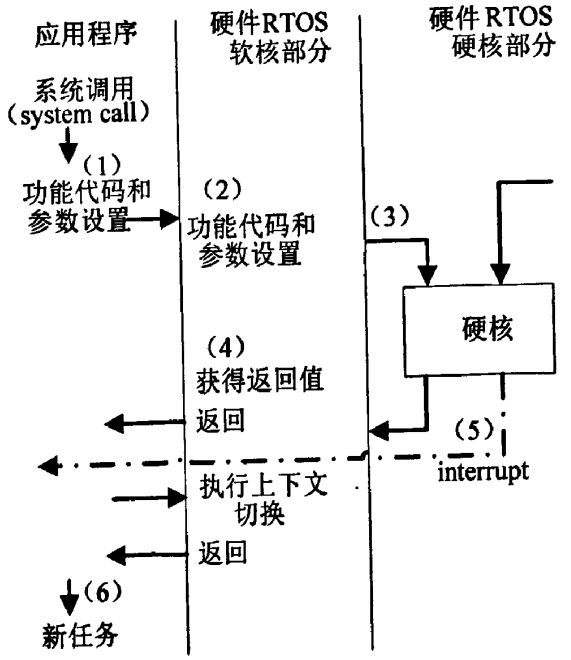

图3-6 硬件RTOS系统调用处理流程

Fig. 3-6 Hardware RTOS processing of system calls

4. 外部事件的异步请求可以在硬件逻辑执行系统调用的过程中被接收，所以，系统调用被硬件逻辑执行完后可以转向以下的处理过程之一：

(1) 若高优先级的任务经过系统调用后变为就绪状态，则硬件逻辑部分向处理器发出任务上下文切换的硬件中断请求。

(2) 若中断任务经过外部事件请求后进入就绪状态，则向处理器发出要求任务切换的硬件中断请求。

(3) 执行完系统调用后，处理结果将采用中断的形式通知并返回给处理器。

5. 实时操作系统的任务上下文切换由中断处理函数来完成。

6. 处理器执行新任务。

目前，由于硬件芯片的造价在不断地降低和 EDA 技术的不断发展，“软件硬件化”已经成为一个新的热点研究趋势。将实时操作系统的部分功能模块采用硬件实现之后，充分发挥了同处理器并行执行的特性，使实时操作系统的响应时间相比于传统软件实现的实时操作系统提高了很多。

3.3.2 硬实时任务管理器的架构

我们可以用硬件逻辑来实现实时操作系统的部分功能，当用硬件的方式实

现实时操作系统中有关任务管理的一些功能后，实时操作系统中任务管理和切换的开销将得到大幅度的降低。

用硬件逻辑实现实时任务管理器(RTM，Real-Time Task Manager)，主要用硬件逻辑实现了任务的数据结构，支持实时操作系统的主要功能任务管理和任务调度。利用接口控制器实时任务管理器与处理器进行通讯，可以采用总线、双口寄存器或内存映射来实现接口控制器与处理器进行通讯。

基于FPGA的实时任务管理器的架构如图3-7，实时任务管理器核心分为两部分：任务控制寄存器和任务控制逻辑。任务的各种状态保存在任务控制寄存器中，并作为与处理器的控制接口，时间管理、事件管理和任务调度等模块的逻辑控制由任务控制逻辑来完成。实时任务管理器通过总线与接口控制器通信，接口控制器充当了实时任务管理器与处理器的桥接作用，使实时任务管理器成为了与处理器相对无关的独立模块，可以与处理器集成在一个芯片上，并且作为IP核通过接口控制器与处理器进行灵活的通信，当实时任务管理器作为FPGA上的一个组件时，更易于商业化。

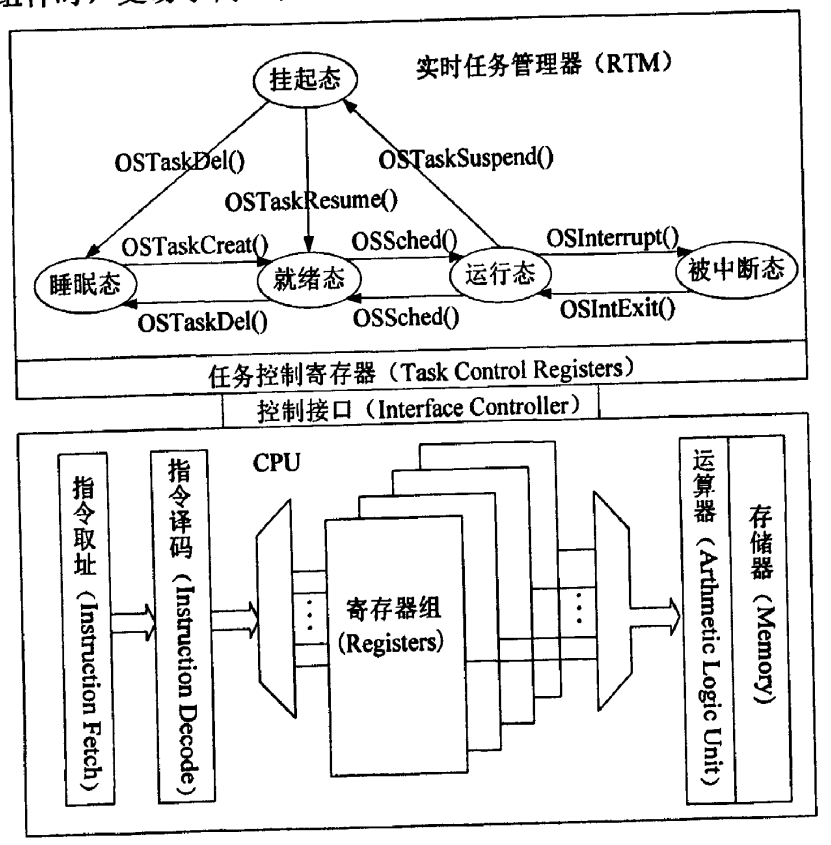

图3-7 基于RTM的FPGA架构

Fig. 3-7 RTM architecture of FPGA

实时任务管理器的组织结构如图3-8，任务的各种状态都保存在任务控制寄存器中，使任务控制器具有与其他模块进行通信接口的作用。通过对时钟节拍的类型和时钟节拍长度的定义的掌握由时间管理器最终选择时钟节拍的长度，并且对所有任务的延时域进行递减。当某个任务的延时域被减为0时，则把该任务的状态域的Delay位清零。System Clock是系统时钟。根据发生事件的标识由事件管理器找出在等待该事件的任务中的最高优先级任务，并且对该任务的Event位清零。通过优先级域和状态域的信息，任务调度器根据基于优先级的调度算法找出在最高优先级的就绪任务。

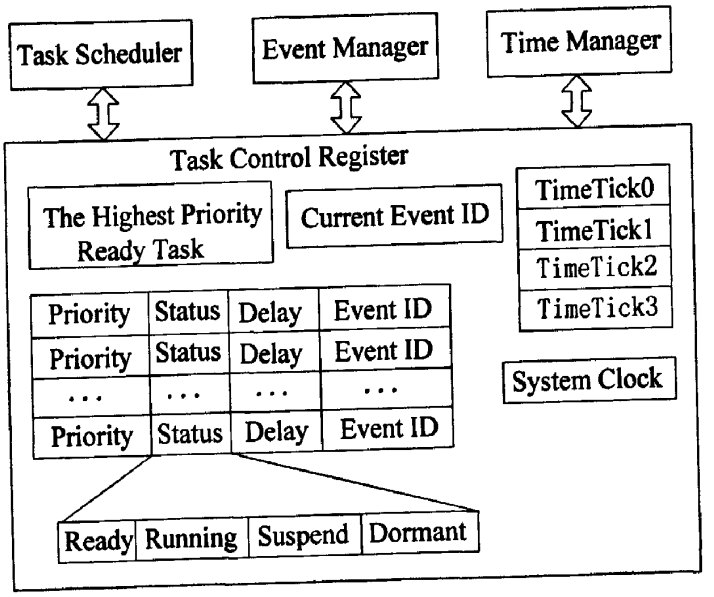

图3-8 RTM组织结构

Fig. 3-8 RTM structure

3.4 本章小结

本章首先分析了μC/OS-II的特点，并给出μC/OS-II内核中任务管理和调度的原理，任务管理是实时操作系统的核心，分析并找出了影响实时操作系统实时性的决定因素，针对传统实时操作系统内核占用系统资源、影响系统实时性的问题，提出硬件实时操作系统的结构和运行原理，并提出用硬件电路实现实时操作系统中的系统调用和任务调度器的方案。重点给出了采用FPGA实现实时任务管理器的架构。

第4章 基于硬件实时任务管理器的实现

4.1 引言

实时操作系统的核心是任务调度，任务调度负责外部事件的处理、中断处理以及各个任务间的通信。任务之间的关系随之系统功能的增强与完善变得越来越复杂，各任务与外围设备的通讯更加频繁，任务调度需要不断地进行调度，进而使事件的实时响应能力和系统的性能急剧地降低。所以任务调度变成了提高实时操作系统性能的瓶颈，要想提高实时操作系统的整体性能必须从提高任务调度的性能开始。如果将任务调度用硬件逻辑来实现，一定可以提升任务调度的性能，进而提高了整个实时操作系统的性能，好的调度算法更是任务管理器的核心。

4.2 任务管理的硬件设计

4.2.1 任务管理模块的硬件框架设计

在μC/OS-II中的任务管理主要由以下几部分构成：建立任务、删除任务、挂起任务、恢复任务、查询任务和任务调度等。其中通过系统调用由用户调用完成的是建立任务、删除任务、挂起任务、恢复任务和查询任务；由系统函数调用来完成任务调度，不能由用户来直接调用。所以，本文将任务管理模块分成了任务调度器的硬件实现和系统函数的硬件实现两个部分。

如图4-1所示。以下几个部分构成了任务管理器的硬件结构：控制器，输入寄存器，输出寄存器，任务调度器的硬件电路，任务系统调用的硬件逻辑电路和TCB寄存器组。

输入寄存器将得到的参数和功能代码传递给控制器，控制器将生成相应的控制信号，并将控制信号发送给相应的硬件模块。

控制器将建立任务、删除任务、改变任务的优先级、挂起任务或恢复任务等控制信号发送给任务系统调用的硬件逻辑电路，硬件逻辑电路根据接受信号的不同，译码处理输入寄存器中的数据，然后写入相应任务的TCB寄存器或者修改相应任务的TCB对应值。对相应任务的TCB的各个寄存器的操作其实就是

对相应任务的处理，虽然实现原理相对简单，但是逻辑电路的实现很复杂。

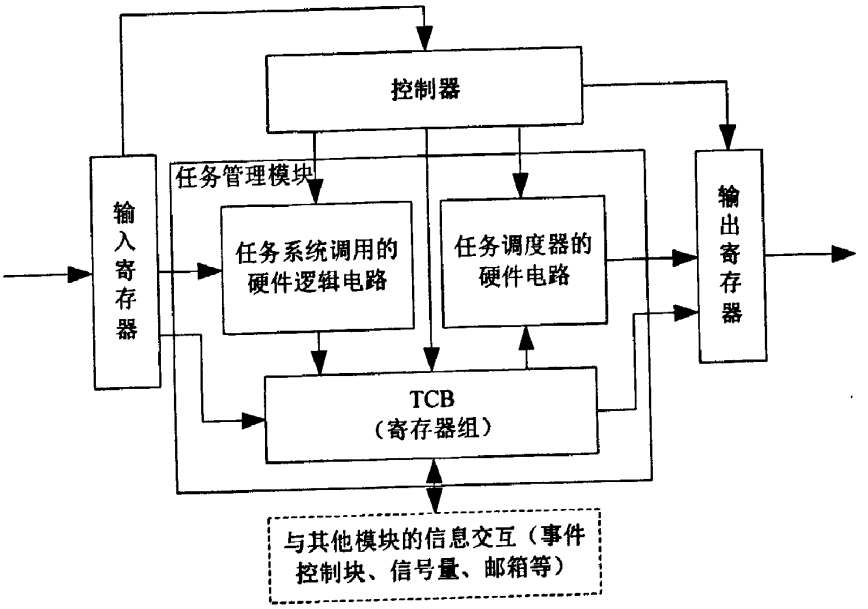

图 4-1 任务管理的硬件结构框图

Fig. 4-1 Hardware strcture of task management

根据TCB中的任务状态位由任务调度器硬件电路的硬件任务调度算法决策出处于最高优先级的就绪态任务。当决策出的最高优先级就绪任务与处理器正在执行的任务不相同时，中断请求信号由调度器的硬件逻辑电路发出，请求执行任务的上下文切换。最后将要执行的下一个任务的ID号传递给输出寄存器。

任务控制块TCB是任务存在的唯一标识，也是任务管理模块中的一个重要的数据结构，任务管理模块的主要任务就是维护TCB表，在嵌入式操作系统μC/OS-II中，通过一个链表实现了TCB数据结构，并且把它存放在系统的RAM中，但是采用硬件逻辑电路并不能高效地实现基于链表的TCB数据结构，所以，本文采用FPGA的片内寄存器来实现TCB的数据结构。

4.2.2 任务管理系统调用函数的硬件设计

用硬件逻辑来实现任务管理的结构如图4-2所示。任务管理系统主要由建立/删除任务、挂起/恢复任务、查询任务和任务调度等组成。将参数1和参数2传递过来的参数经过输入端口进行接收，参数1是系统分配给任务的ID号，参数2是包含对任务管理的一些信息，当任务建立时，参数中包含任务优先级、任务代码段地址、任务的参数指针和分配给任务的堆栈栈底指针等，将这些信息写入

任务的TCB中。根据任务的ID号每个任务都有一个相应的TCB，输出端口将会根据任务的ID号选择并且输出相应任务TCB中的信息。

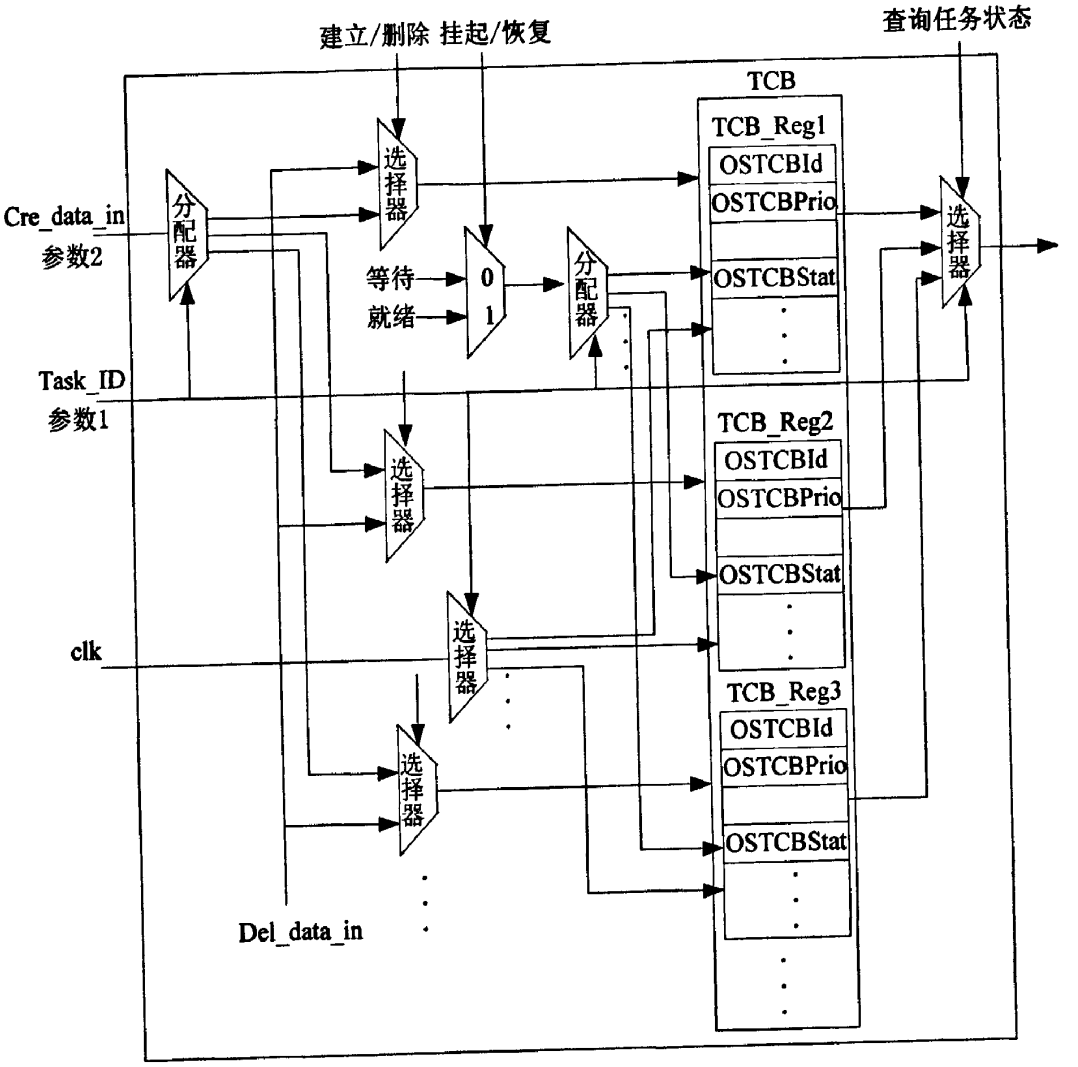

图 4-2 任务系统调用的硬件实现电路图

Fig. 4-2 Hardware implement of task system calls

如图4-2所示，通过硬件逻辑实现的任务管理系统主要由分配器、选择器和TCB寄存器组成。

1. 分配器 分配器的主要作用是根据参数1传递的任务ID号信息，将在建立任务时参数2传递过来的信息传递到相应的逻辑电路上，并且最终会传递到任务相应的TCB中。将多路选择器的输入端与分配器的输出端相连接，则将参数2所传递的信息传递到选择器的输入端，作为多路选择器的输入。在对任务进行挂

起和恢复时，分配器会依据相应任务的ID号，将任务状态位的信息传递到相应TCB寄存器的状态位。当建立或者删除任务时对相应任务TCB寄存器读写的VHDL代码：

```
PROCESS(CLK, OSTCBStkBottom, OSTCBStackSize, OSTCBId,
        OSTCBEvtPtr, OSTCBDelay, OSTCBStat, OSTCBPrio)
BEGIN
IF(clk'event and clk='1') THEN
CASE Task_ID IS
WHEN "000"=>
    temp0_ TCBStackBtm <=OSTCBStkBottom;
    temp0_ TCBStackSize <=OSTCBStackSize;
    temp0_ TCBId        <=OSTCBId;
    temp0_ TCBEvtPtr    <=OSTCBEvtPtr;
    temp0_ TCBDelay     <=OSTCBDelay ;
    temp0_ TCBStat      <=OSTCBStat;
    temp0_ TCBPrio      <=OSTCBPrio;
    ……
WHEN "111"=>
    temp7_ TCBStackBtm <=OSTCBStkBottom;
    temp7_ TCBStackSize <=OSTCBStackSize;
    temp7_ TCBId        <=OSTCBId;
    temp7_ TCBEvtPtr    <=OSTCBEvtPtr;
    temp7_ TCBDelay     <=OSTCBDelay ;
    temp7_ TCBStat      <=OSTCBStat;
    temp7_ TCBPrio      <=OSTCBPrio;
WHEN OTHERS=>NULL;
END CASE;
END IF;
END PROCESS;
```

2．选择器 在建立和删除任务时将相关信息传递给选择器的输入端，在Create 或 Delete 控制信号的控制下，选择多路选择器的一路进行输出，多路选择器的输出端与相应任务的 TCB 寄存器直接相连，将相关信息直接写入 TCB 寄存器中。当控制信号 Create 有效时，参数 2 的信息写入相应 TCB 中，并且把

建立任务的进程激活，完成创建任务的相关工作。当控制信号 Delete 有效时，把相应任务的 TCB 中的信息清空，写入系统初始化值。当执行挂起任务或者恢复任务时，仅仅需要依据任务的 ID 号，修改其相应的 TCB 寄存器中的状态位的值即可。实现选择器的 VHDL 代码如下：

```
PROCESS(Create, Delete, clk )
BEGIN
temp<=Create & Delete;
IF(clk'event and clk='1')THEN
st_out<=Create or Delete;
CASE temp IS
WHEN "01" =>
    temp_data_ TCBStackBtm <="00000000";
    temp_data _ TCBStackSize <="00000000";
    temp_data _ TCBId          <=OSTCBId;
    temp_data _ TCBEvtPtr      <="00000000";
    temp_data _ TCBDelay       <="0000";
    temp_data _ TCBStat        <="00";
    temp_data _ TCBPrio        <="000";
WHEN "10" =>
    temp_data_ TCBStackBtm <=OSTCBStkBottom;
    temp_data_ TCBStackSize <=OSTCBStackSize;
    temp_data_ TCBId           <=OSTCBId;
    temp_data_ TCBEvtPtr     <=OSTCBEvtPtr;
    emp_data_ TCBDelay       <=OSTCBDelay ;
    temp_data_ TCBStat       <=OSTCBStat;
    temp_data_ TCBPrio       <=OSTCBPrio;
    ……
WHEN OTHERS => NULL;
END CASE;
END IF;
END PROCESS;
```

3．TCB寄存器 通过软件实现的实时操作系统，被任务信息占用的TCB和空闲的TCB分别以任务链表和空闲任务链表的形式在内存中存在。但是采用硬

件逻辑电路并不能高效地实现基于软件算法的链表或者队列，因为通过链表实现的数据结构，只有在读取了前一节点的内容后，才能通过指针读取后续节点中的信息，完全抑制了硬件逻辑电路的并行特性。因此，在本文中采用FPGA的片内寄存器来实现TCB的数据结构，进而减少了链表的查找时间，提高了系统的执行效率。

4.2.3 任务调度器的硬件设计

任务调度是实时操作系统的核心工作，总是让处于就绪状态优先级最高的任务运行是嵌入式操作系统μC/OS-II的任务调度思想。本文采用硬件逻辑电路实现硬件任务调度器，进而实现了高效的任务调度算法，如图4-3所示。把任务的优先级作为硬件任务调度器的选择条件，把TCB寄存器的状态位同硬件任务调度器直接相连，只要任务的状态位或者其优先级发生变化，任务调度器就会重新进行任务调度。

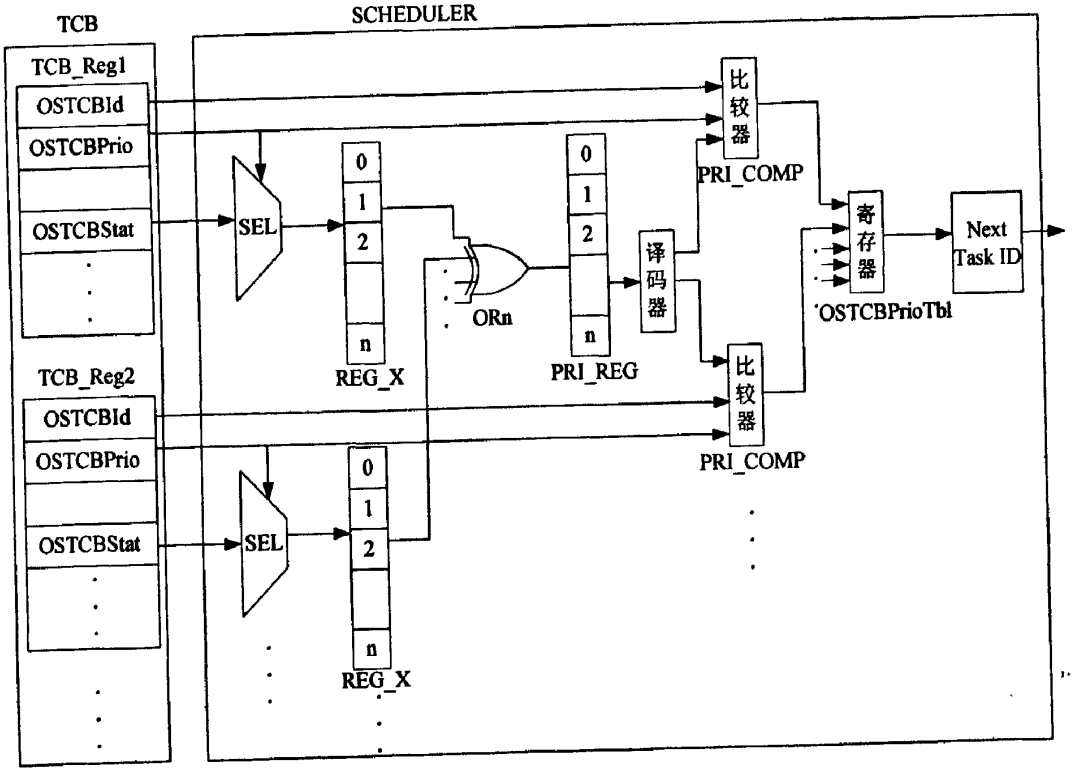

图 4-3 任务调度器的硬件实现电路图

Fig. 4-3 Hardware implement of scheduler

在图4-3中，TCB寄存器中的状态位与数据分配器直接相连，并且以TCB寄存器中的优先级OSTCBPrio作为选择条件，把状态位OSTCBState中的值传递到相应的输出通道，分配器输出通道的值传递给REG_X寄存器，并且按位对应存储，然后把所有的REG_X寄存器中的值进行按位或运算，最后把运算结果传递给PRI_REG寄存器。如图4-4所示，实时操作系统中优先级的个数对应优先级寄存器PRI_REG的位数，寄存器中高位的优先级低于低位的优先级，如果优先级寄存器PRI_REG的某位为1，就表示有相应位优先级的任务进入就绪状态。将寄存器PRI_REG中的值传递给译码器，经译码器译码后选出处于就绪态的最高优先级。PRI_COMP比较器的作用是把任务的优先级OSTCBPrio作为比较内容，把任务的ID号作为索引，将译码器选出的处于就绪态的最高优先级与每个任务TCB中的优先级相比较，如果值相同，则输出相应任务的ID号，否则输出值为0，最后将算得的处于就绪态优先级最高的任务的ID号送给寄存器OSTCBPrioTbl，如果处于就绪态的最高优先级任务不只一个，则输出优先级最高的排队时间最长的任务的ID号。

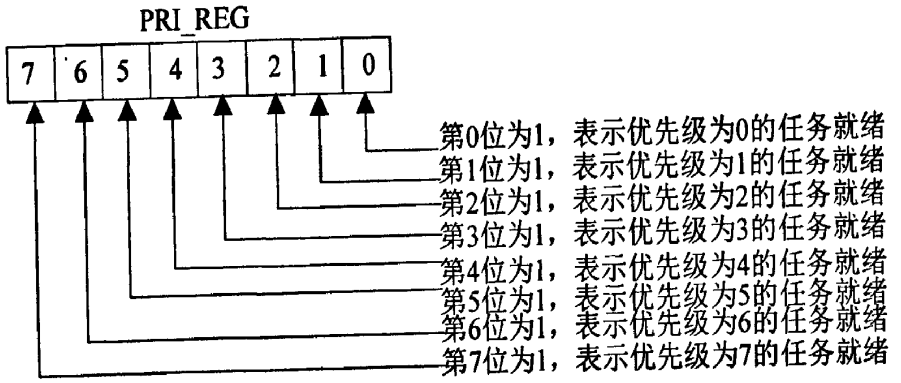

图 4-4 优先级寄存器 PRI_REG

Fig. 4-4 Register of PRI_REG

4.3 调度器核心数据结构

4.3.1 任务控制块和任务控制块列表

嵌入式实时操作系统的核心技术是任务调度机制。基于嵌入式操作系统μC/OS-II 的思想，任务在被建立后都有一个相对应的 TCB 来管理相应任务的信息。本文通过使用 FPGA 的片内寄存器来实现任务调度内核的任务控制块 TCB 队列。当任务使用处理器的权利被剥夺时，可以用 TCB 来保存任务被剥夺处理

器使用权后的信息状态。

实现硬件任务调度内核的关键在于设计一个合理的 TCB，该任务控制块应该能够依据任务调度算法合理的分配优先级，并且方便优先级的比较，能够决策出最终的结果。依据以上要求，设计 TCB 的数据结构。任务的识别符 ID 号为 OSTCBId，任务的优先级为 OSTCBPrio，用以表示任务的各种状态的为 OSTCBStat，任务延时时钟节拍为 OSTCBDelay，任务的堆栈大小为 OSTCBStackSize，任务的栈底指针为 OSTCBStkBottom，指向事件控制块的指针为 OSTCBEvtPtr。用 VHDL 表示 TCB 的参数列表：

```
TYPE tcb IS RECORD
        OSTCBId             : std_logic_vector(0 to n);
        OSTCBPrio           : std_logic_vector(0 to n);
        OSTCBStat           : std_logic_vector(0 to n);
        OSTCBDelay          : std_logic_vector(0 to n);
        OSTCBStackSize      : std_logic_vector(0 to n);
        OSTCBStkBottom      : std_logic_vector(0 to n);
        OSTCBEvtPtr         : std_logic_vector(0 to n);
END RECORD;
```

以上设计的 TCB 数据结构保留了 μC/OS-II 中的原有部分参数意义不做改变，并且删减了部分参数，具体差别详见参考文献[34]。

任务的状态控制寄存器为 OSTCBStat，对该寄存器进行读操作时，可以返回任务的当前状态，如就绪态或者挂起状态；对该寄存器进行写操作时，则使状态寄存器进入写入信息的状态。对于这些参数的改动最为重要的是 OSTCBPrio 和 OSTCBDelay 这两个参数，系统被改进成了支持不同任务具有相同优先级的可剥夺型调度内核。当任务被创建时，赋予唯一的初值给任务的识别码 OSTCBId，赋予任务优先级 OSTCBPrio 合适的值，允许不同的任务具有相同的优先级，赋予 OSTCBDelay 初值 0。任务在运行的过程时，不允许改变任务的 OSTCBId 和 OSTCBPrio，任务在进入就绪状态后，每过一个系统节拍，处于就绪状态任务的 OSTCBDelay 寄存器加 1，随着任务的运行，就绪状态任务的 OSTCBDelay 随着系统节拍不断地增加，直到该任务被运行，则将 OSTCBDelay 寄存器清零。系统任务调度算法的规则是：任务的 OSTCBPrio 的值越小，优先级越高，越优先执行；当任务的 OSTCBPrio 的值相同时，则 OSTCBDelay 的值越大越优先执行。

4.3.2 任务调度算法及实现

任务调度内核是实时操作系统的核心，其主要的工作内容是依据调度算法，在每一个需要进行任务调度的时刻决策出将要执行的下一个任务，并且适时地进行任务上下文的现场切换。所以，系统的任务调度内核应该维护一系列的就绪任务队列和等待任务队列等，并且在每个需要调度的时刻，依据任务的优先级对就绪队列进行重新排列，以决策出处于就绪态的最高优先级任务。在嵌入式操作系统uC/OS-II基于优先级抢占式调度算法内核的基础上，扩展了不同任务具有相同优先级的调度算法，任务的优先级用以决策任务被执行的顺序，处于最高优先级的任务被最先执行，当不同的任务具有相同优先级时，由任务的等待时间决定任务执行的顺序。任务调度器总是会运行处于就绪态优先级最高并且等待时间最长的任务。对于实时操作系统的内核是可抢占式的内核，系统在任何时刻总是运行最高优先级的就绪态任务，即高优先级的任务一旦处于就绪状态，就抢占处理器开始运行。所以系统调度算法的优劣一定影响实时操作系统的性能。

在研究uC/OS-II的任务调度算法后可以分析出，每个任务和相应的优先级在uC/OS-II中是一一对应的，因此通过任务调度算法决策出最高优先级就绪任务后，同时也确定了任务相应的TCB，具体通过任务块优先级列表OSTCBPrioTbl[]来表示这个一一对应关系，当一个任务的优先级OSTCBPrio确定后，就能通过通过优先级列表OSTCBPrioTbl[]用类似索引功能找到相应任务的TCB[39]。当不同的任务可以拥有相同的优先级的时候，则原来指向单一任务TCB的优先级列表OSTCBPrioTbl[]被改进为指向多个任务的TCB，具有相同优先级的不同任务根据等待时间的长短排成一个TCB队列。优先级列表OSTCBPrioTbl[]和任务TCB的关系如图4-5所示，相同队列的TCB具有相同的优先级，不在同一队列的TCB的优先级不同。

任务调度器与TCB的数据通信用VHDL代码表示如下：

```
CASE TCBPrio IS
WHEN"000"=>IF RDY='1'THEN x<="00000001";
            ELSE x<="00000000";
            END IF;
WHEN"001"=>IF RDY='1' THEN x<="00000010";
            ELSE x<="00000000";
            END IF;
```

```
WHEN"010"=>IF RDY='1' THEN x<="00000100";
            ELSE x<="00000000";
            END IF;
WHEN"011"=>IF RDY='1' THEN x<="00001000";
            ELSE x<="00000000";
            END IF;
.....
WHEN OTHERS=>IF RDY='1' THEN x<="10000000";
            ELSE x<="00000000";
            END IF;
END CASE;
```

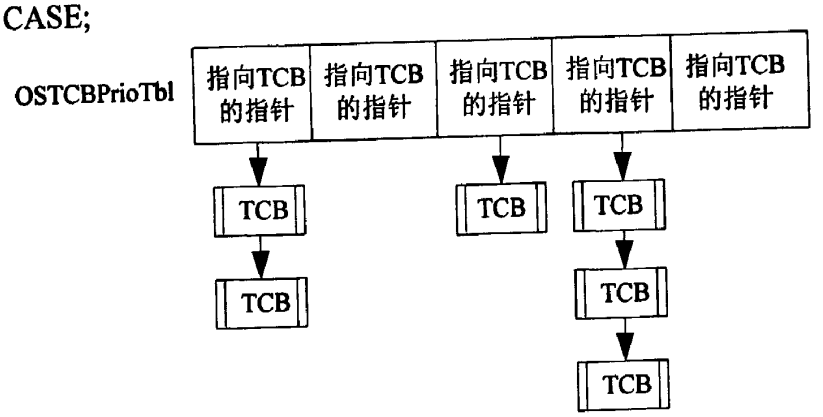

图 4-5 OSTCBPrioTbl 的结构示例

Fig. 4-5 Structure example of OSTCBPrioTbl

4.4 信号量管理的硬件设计

4.4.1 信号量管理的硬件总体设计

用硬件逻辑实现的信号量管理模块主要由双向计数器、比较判断器、译码器和事件控制块 ECB 组成，如图 4-6 所示。

1. 译码器 处理器传递数据给译码器，译码器把传送过来的数据进行译码输出产生的控制信号：信号量建立信号 cre_sem，信号量删除信号 del_sem，信号量申请信号 pend_sem，信号量释放信号 post_sem。当信号量建立信号 cre_sem 有效时，系统会统计各种可用资源的数量，并把数值写入信号量寄存器组 Sem_cnt_regs 中，可以通过信号量寄存器组 Sem_cnt_regs 对各种资源的使用情

况进行查询。当信号量申请信号 pend_sem 和信号量释放信号 post_sem 有效时，控制相应的信号量 Cnt 值作加 1 或减 1 操作。

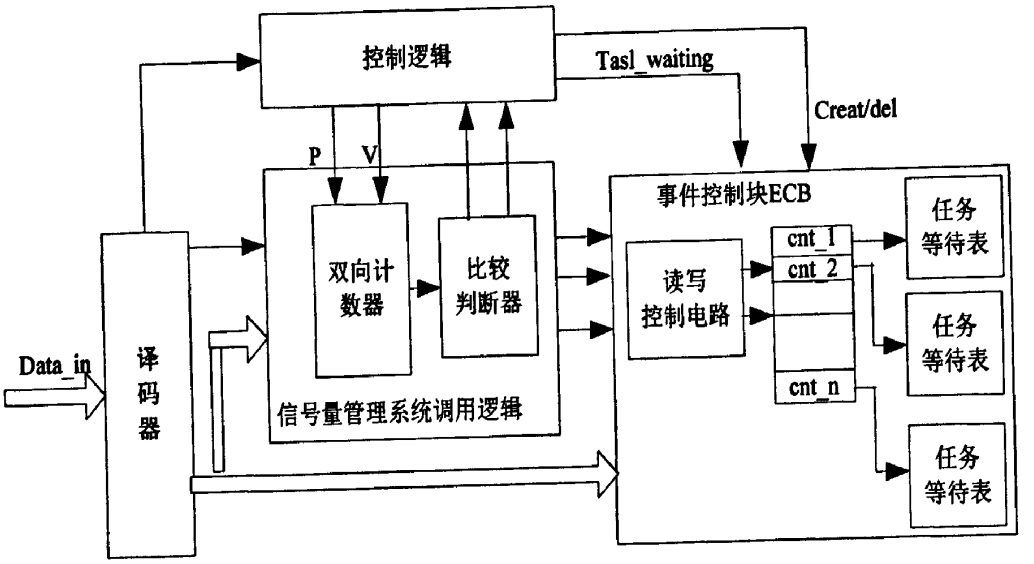

图 4-6 硬件信号量管理的工作原理图

Fig. 4-6 Hardware strcture of semaphore management modules

2．双向计数器和比较判断器 由比较判断器和双向计数器构成了信号量管理模块的系统调用逻辑。双向计数器主要是实现了对信号量的加 1 和减 1 的操作；比较判断器的主要作用是将信号量 Cnt 的值与零做比较，并且依据比较判断器给出的结果转入相应的入口。

3．信号量的事件控制块 由任务等待表和信号量寄存器组构成了信号量事件控制块 ECB。不同种类的信号量存放在不同的寄存器中，在建立系统的时候就给出了相互对应的关系，每个信号量具有唯一的标识符 ID 号。采用 FPGA 的片内寄存器来实现信号量的任务等待列表。系统中任务的个数对应信号量寄存器的位数，每个任务对应信号量寄存器的一位，信号量寄存器的初始值均为 0，表示系统中没有任务等待该信号量。当有任务等待信号量时，将相应信号量寄存器的相应位置 1；当有任务释放信号量时，将会唤醒等待该信号量的优先级最高的任务，并将信号量寄存器的相应位清零。信号量寄存器每实施一次置位或复位操作，必须将相应任务改变状态的申请提交给控制器，同时把任务的 ID 号提交给任务管理模块。

4.4.2 信号量管理系统调用的硬件设计

信号量管理的系统调用函数的硬件逻辑实现电路如图 4-7 所示，主要由具有预置功能的双向计数器和比较器组成。

具有预置功能的双向计数器的主要功能是做加 1 和减 1 运算。当信号量被创建时，将双向计数器的预置引脚 SET 置 1，把信号量的初始值写入信号量寄存器中；当对信号量申请操作时，将计数器的逆向引脚 DOWN 置 1，计数器作减 1 操作；当对信号量释放操作时，将计算器的正向引脚 UP 置 1，计数器加 1 操作。

双向计数器把信号量的 Cnt 值传递给比较器，比较器则把 Cnt 值与 0 作比较。当信号量申请信号 pend_sem 有效时，如果 Cnt≤0，则传出 Task_Wait 信号，通知事件控制块 ECB 的读写控制电路，把申请此信号量的任务的 ID 号根据其优先级写入相应信号量的任务等待列表中。当信号量释放信号 post_sem 有效时，如果 Cnt<0，则表示有任务正在等待该信号量的释放，从该信号量的任务等待列表中查找出优先级最高的任务，并且同时向任务管理器发出 Task_Ready 信号，通知任务管理器将该任务转变为就绪状态，触发调度器进行调度。

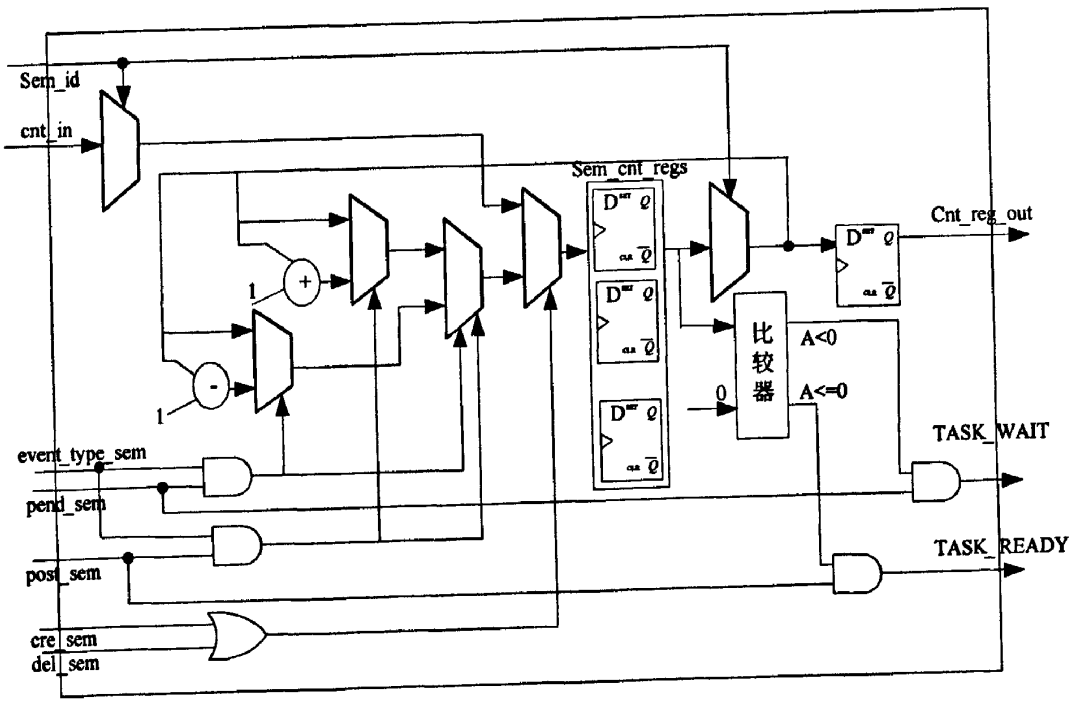

图 4-7 信号量系统调用的硬件实现电路图

Fig. 4- Hardware implement of system calls of semaphore management

以下给出信号量的建立/删除、信号量申请和释放的实现过程。

1. **信号量建立/删除** 当信号量建立信号 cre_sem 和信号量删除信号 del_sem 有效时，应用程序把数据 cnt_in 传递给信号量硬件逻辑电路。当信号量被建立时，传递的 cnt_in 值表示系统中该信号量可用的总数量；当信号量被删除时，传递的 cnt_in 值为 0，即系统中无可用的此信号量。把信号量建立时的值存入信号量寄存器组 Sem_cnt_regs 中。当信号量被建立的时候，该信号量的任务等待列表中应该没有等待的任务，所以将对应该信号量的任务等待列表置空。当信号量被删除时，同时也应该删除该信号量的相应任务等待列表中的所有任务，如果任务等待列表中的任务并不需要其它的资源，则将该任务转变为就绪状态。

2. **信号量申请（P 操作）** 当信号量申请信号 pend_sem 有效时，首先，依据申请信号量的 Sem_id 号从信号量寄存器组 Sem_cnt_regs 的相应寄存器中取得欲申请信号量的 Cnt 值。把当前信号量的 Cnt 值做减 1 操作，再写回相应的信号量寄存器中。然后把做减 1 操作后信号量 Cnt 的值传递给比较器与 0 作比较，如果 Cnt≥0，则申请该信号量的任务获得信号量资源继续执行；如果 Cnt<0，则给出 Task_Wait 信号，通知事件控制块 ECB 的读写控制电路，将申请此信号量的任务的 ID 号写入相应信号量的任务等待列表中。

3. **信号量释放（V 操作）** 当信号量释放信号 post_sem 有效时，首先，依据释放信号量的 Sem_id 号从信号量寄存器组 Sem_cnt_regs 的相应寄存器中取得欲释放信号量的 Cnt 值。把当前信号量的 Cnt 值做加 1 操作，再写回相应的信号量寄存器中。然后把做加 1 操作后信号量 Cnt 的值传递给比较器与 0 作比较，如果 Cnt>0，则没有任务等待当前释放的信号量，执行完信号量释放后任务继续执行；如果 Cnt≤0，则有任务等待当前释放的信号量，从该信号量的相应任务等待列表中查找出优先级最高的任务，同时给出 Task_Ready 信号，通知任务管理器将任务转变为就绪状态，触发调度器进行任务调度。

4.5 本章小结

本章针对解决提高实时操作系统的实时性任务调度这一瓶颈问题，提出将任务调度硬件化，从而提高实时操作系统的性能。重点给出了任务管理的硬件设计和任务调度算法的硬件实现，并完成了任务间通信所需信号量的硬化。用硬件实现，只要接收从总线传过来的数据，一系列的操作由硬核实现。因为硬核逻辑是物理并行执行的，不占用处理器的时间，所以提高了任务的可调度性和实时性。

第5章 仿真及实验结果分析

了解开发流程和开发工具是进行 SOPC 开发的基础。本章首先介绍了本研究项目所采用的 Xilinx 公司提供的软件开发环境 ISE(Integrated Software Environment，集成软件环境)与 EDK(Embedded Developments，嵌入式开发工具集)和硬件平台 XUP Viterx PRO-II。然后对用 VHDL 编写的代码进行调试、综合和功能仿真，并且对功能仿真进行重点分析，以求验证设计的可行性。

5.1 XUP Viterx-II PRO 硬件开发平台

Xilinx 公司是 FPGA 器件的发明者，也是全球最大的可编程逻辑器件制造商，尤其在通信领域，Xilinx 参与一些通信标准的制定，并且提供各种系统集成和系统解决方案，而且还是通信器件的供应商。本研究项目所用的实验平台是 Xilinx 公司提供的 XUP Virtex-II Pro 硬件开发平台，其平台中包含了 PowerPC 405 硬核处理器。Virtex-II Pro 系列的 XC2VP30 主要技术特征如表 5-1 所示[40]。

表 5-1 Virtex-II Pro XC2VP30 主要技术特征

Table. 5-1 The main technical characteristics of Virtex-II Pro XC2VP30

型号	Slice 数目	分布式 RAM 容量	块 RAM 容量	Power PC	专用乘法器数	DCM 数目	Rocket I/O	最大可用 I/O
XC2VP30	13696	428Kb	2448Kb	2	136	8	8	644

本研究课题所使用的开发板如图 5-1 所示，该开发板是 Xilinx 公司设计的 XUP Virtex-II Pro 开发系统。实验开发板上提供了两个 32 位的 PowerPC 405 硬核处理器，提供了 100MHz 系统时钟和 75MHz 的 SATA 时钟，I/O bank 提供了 8 个，DCM 提供了 8 个，LogicCells 提供了 30816 个，18kb 的 Block RAM 提供了 136 个，其中 Block RAM 的最大容量为 2448KB，可配置的最大 I/O 管脚为 644 个，并且该实验开发板还提供了一路 USB 配置端口和一路 JTAG 配置端口。

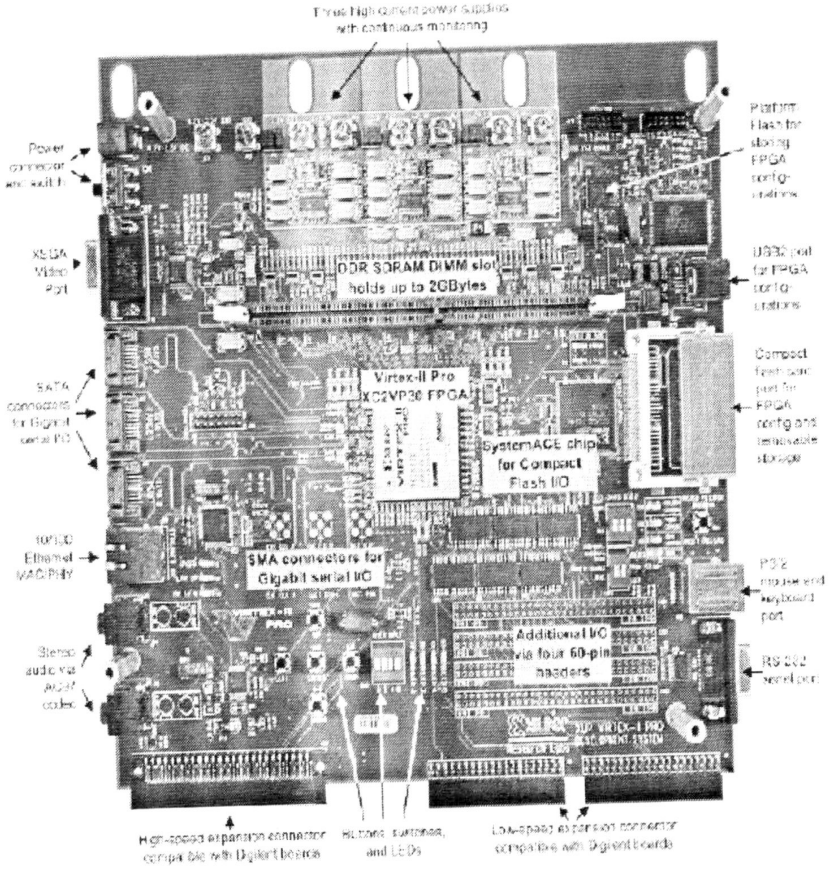

图 5-1 Virtex-II Pro 开发板

Fig. 5-1 Virtex-II Pro development board

5.2 Xilinx 嵌入式开发工具集

5.2.1 集成软件环境 ISE

Xilinx 公司所提供的 ISE 开发软件操作简便，并且 Xilinx 公司 FPGA 芯片市场占有率很大，所以致使其 ISE 开发软件成为通用的 FPGA 工具软件。ISE 可以与第三方软件互相补充，使 EDA 设计更加完善，给予用户更加丰富的 Xilinx 开发平台[41]。

ISE 的主要功能包括设计输入、综合、仿真、实现和下载，如图 5-2 所示，包含了 FPGA 研发的整个过程，从 ISE 的功能上看，并不需要借助第三方 EDA 软件。

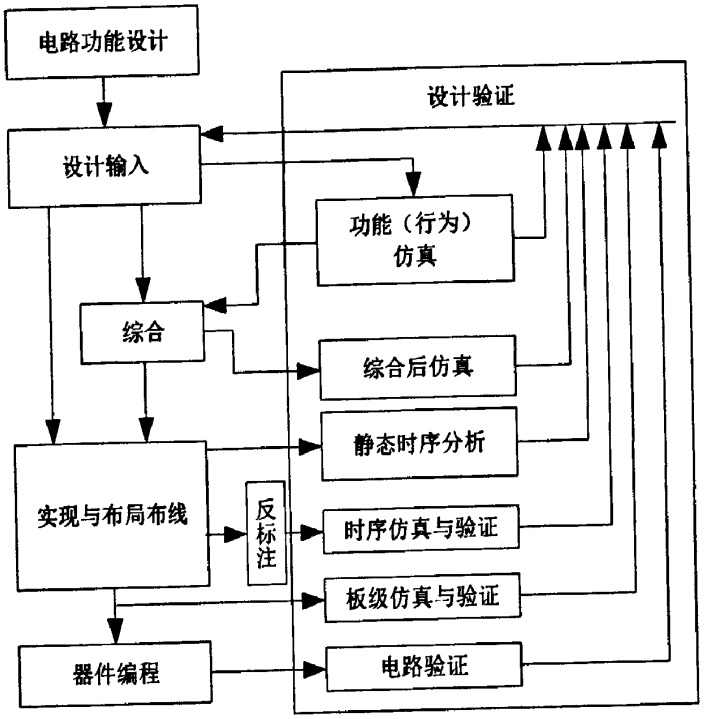

图 5-2 FPGA 设计流程图

Fig.5-2 Flow chart of FPGA design

如表 5-2 所示为使用 ISE 软件开发环境进行 FPGA 设计的过程中可能用到的各种设计工具[42]。

表 5-2 ISE 设计工具表

Table.5-2 ISE design tools table

设计输入	综合	仿真	实现	下载
HDL 文本编辑器 ECS 原理图编辑器 StateCAD 状态机编辑器 Core Generator Constraint Editor	XST FPGA Express (Synplify 和 LeonardoSpectrum)	HDL Bencher (ModelSim)	Translate MAP Place and Route Xpower	BitGen IMPACT

5.2.2 嵌入式开发套件 EDK

嵌入式开发套件(EDK)是设计嵌入式处理系统的集成软件解决方案。EDK自带了许多工具和IP，可以用来设计完整的嵌入式处理器系统，主要包括Xilinx平台工作室(Xilinx Platform Studio，XPS)和软件开发套件(Software Development Kit，SDK)[43]。

因为嵌入式系统设计既包括硬件设计又包括软件设计，所以设计一个完整的嵌入式系统包含众多步骤，每个步骤相互独立，但又相辅相成。因为嵌入式系统的软件和硬件都可以进行裁剪，所以设计一个嵌入式系统并不需要每个步骤都完成。EDK 的完整开发流程如图 5-3 所示。

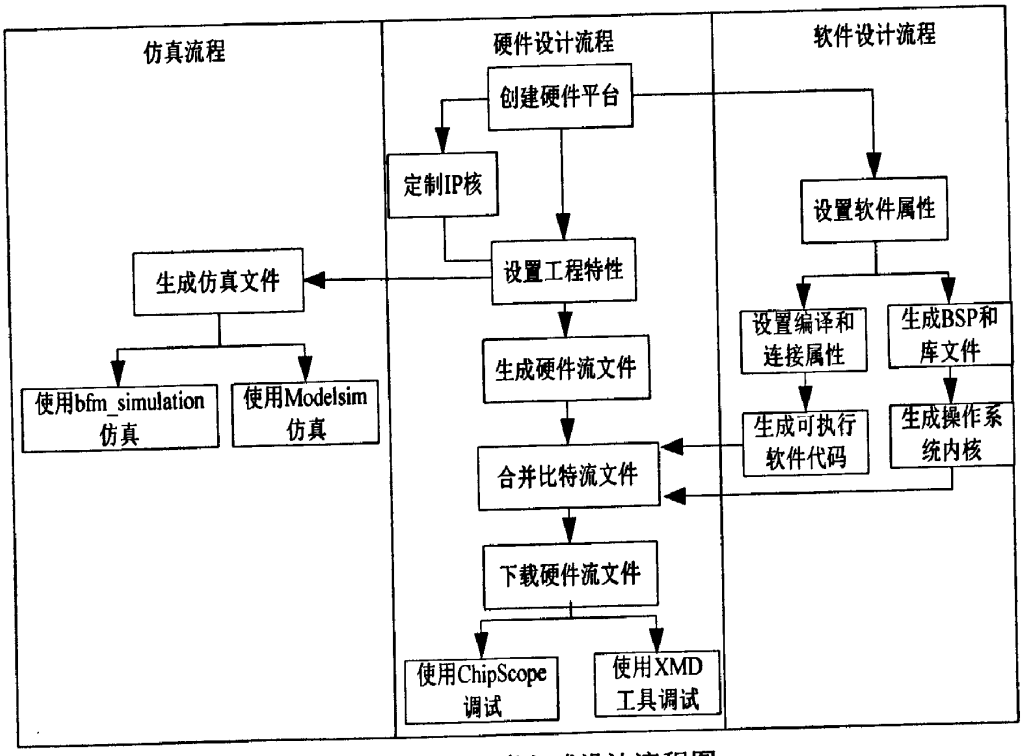

图 5-3 完整的嵌入式设计流程图

Fig. 5-3 Complete embedded design flow chart

5.3 时序仿真及实验结果分析

5.3.1 任务管理模块仿真结果及分析

通过ISE 8.2软件对用硬件逻辑实现的任务管理模块进行功能仿真，如图5-4

所示。

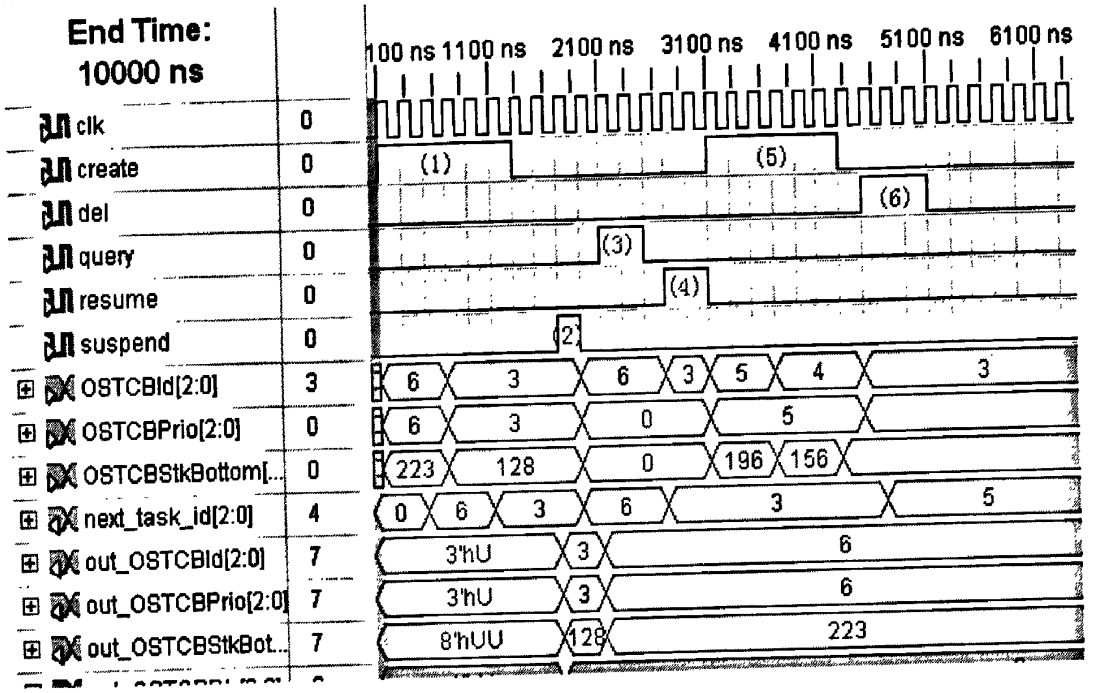

图5-4 硬件任务管理的功能仿真图

Fig. 5-4 Function simulation diagram of hardware task management

任务管理模块的仿真结果分析如下：

1．首先建立任务的OSTCBId号为6，该任务的优先级也为6，由于此时系统中并没有别的任务，所以当系统的时钟上升沿到来时，next_task_id的值为6。然后建立任务的OSTCBId号为3，该任务的优先级也为3，优先级高于OSTCBId号为6的任务，所以抢夺处理器的使用权，则next_task_id的值为3。

2．将任务的OSTCBId号为3的任务进行挂起，则任务调度器进行重新调度，next_task_id的值应为6，处理器执行OSTCBId号为6的任务。

3．依据任务的OSTCBId号为索引，查询OSTCBId号为3和OSTCBId号为6的任务的状态，把从任务TCB中查询得到的相应任务状态传送给处理器。

4．恢复被挂起的OSTCBId号为3的任务，则此时处于就绪态的最高优先级任务的OSTCBId号为3，所以next_task_id为3，OSTCBId号为3的任务抢占处理器继续执行。

5．依次建立两个任务，OSTCBId号分别为5和4，其优先级均为5。

6．将OSTCBId号为3的任务删除，则此时处于就绪态的最高优先级并且等待时间最长的任务的是OSTCBId号为5的任务，所以next_task_id为5。

假设系统中存在的任务数为8，则设计整个系统所需要的资源情况如表5-3所示。由表5-3易知，FPGA所提供的资源完全可以满足设计系统的需求。

表 5-3 系统在 XC2VP30FF896C 上实现所需硬件资源

Table.5-3 Resources required of hardware realization in XC2VP30FF896C

Logic Utilization	Used	Available	Utilization
Number of Slices	506	13698	3%
Number of Block	85	558	15%
Number of Slices Flip Flops	746	27394	3%
Number of 4 input LUTs	423	27394	2%

5.3.2 信号量管理模块仿真结果及分析

通过 ISE 8.2 软件对信号量管理模块进行仿真，建立/删除信号量的功能仿真如图 5-5 所示，申请/释放信号量的功能仿真如图 5-6 所示。

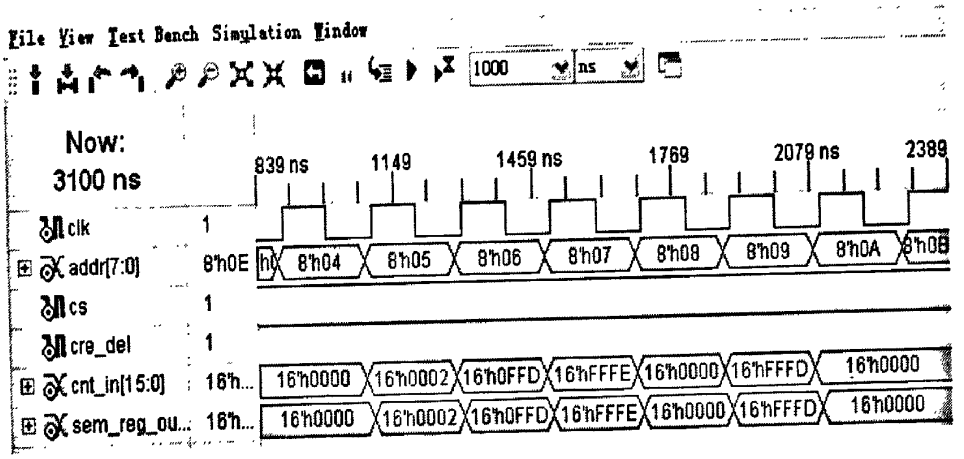

图 5-5 创建/删除一个信号量

Fig. 5-5 Creat or delete a semaphore

信号量的建立或删除的功能仿真如图 5-5 所示，其仿真步骤如下：

1. 当信号量被建立时，依据信号量在寄存器组 Sem_cnt_regs 中的偏移地址，把信号量的初始值写入相应的寄存器中。例如分别将值0x0002写入05H单元中，将值 0x0FFD 写入 06H 单元中等等。

2. 当信号量被删除时，依据信号量被建立时在寄存器组 Sem_cnt_regs 中的偏移地址，把存放该信号量的相应寄存器的值清零。例如删除存放在 06H 单元

中的信号量，即把 cnt_in 的值设置为 0，重新写入 06H 单元中。

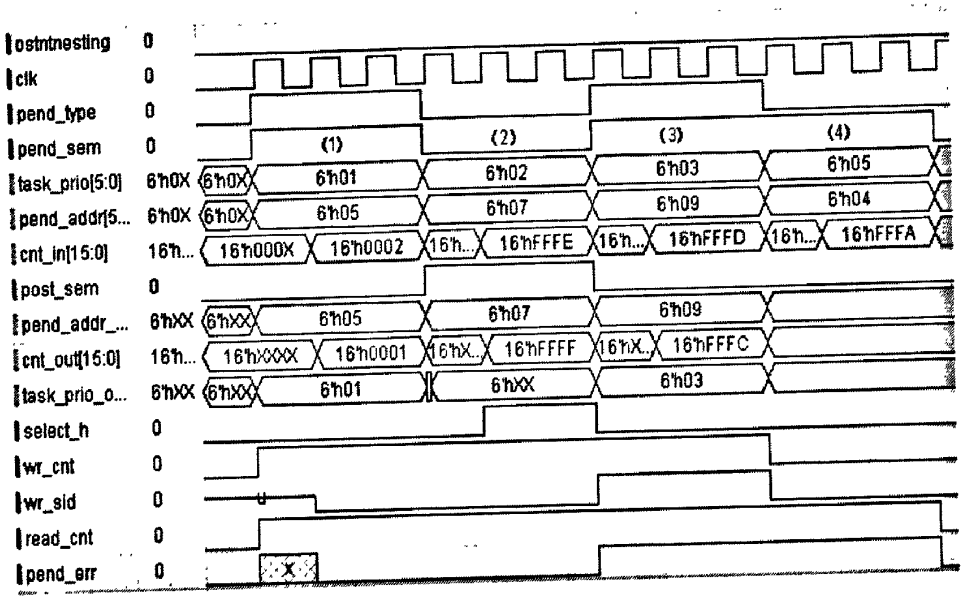

图 5-6 申请及释放一个信号量

Fig. 5-6 Pend or post a semaphore

信号量的申请或释放的功能仿真如图 5-6 所示，其仿真步骤如下：

1. 寄存器 sem_cnt_reg5 中的信号量被优先级为 1 的任务所申请。申请信号量的 pend_sem 信号置为高电平，系统会给出该信号量在寄存器组 Sem_cnt_regs 中的偏移地址 05H，依据偏移地址取得该信号量的 Cnt 值为 0x0002，把该信号量的 Cnt 值做减 1 操作后写回原地址单元，并同时将该信号量的 Cnt 值传递给比较器与 0 做比较，因为 Cnt>0，所以申请该信号量的任务可以获得信号量继续执行。

2. 寄存器 sem_cnt_reg7 中信号量被优先级为 2 的任务释放。释放信号量的 post_sem 信号被置为高电平，系统同时会给出该信号量在寄存器组 Sem_cnt_regs 中的偏移地址 07H，从中取得该信号量的 Cnt 值为 0xFFFE<0，Cnt 值以补码的形式存放在寄存器中。把该信号量的 Cnt 值做加 1 操作后写回原地址单元，并同时将该信号量的 Cnt 值传递给比较器与 0 做比较，因为 Cnt≤0，则表示有任务等待该信号量的释放， Select_h 信号有效，从该信号量的相应任务等待列表中查找出最高优先级的任务转变为就绪状态，触发任务调度器进行任务调度。

3. 优寄存器 sem_cnt_reg9 中的信号量被优先级为 3 的任务所申请。所申请的信号量在寄存器组 Sem_cnt_regs 中的偏移地址为 09H。依据偏移地址取得该信号量的 Cnt 值，并对该值做减 1 操作，Cnt 的值变为 0xFFFC<0，则说明系统

中没有可用的该信号量资源，应该将申请该信号量的任务写入相应的任务等待列表中，并且告知任务管理器将该任务阻塞。

设计信号量管理模块所需要的资源如表 5-4 所示。

表 5-4 硬件信号量管理的资源使用情况

Table.5-4 Resources required of hardware semaphore realization in XC2VP30FF896C

Logic Utilization	Used	Available	Utilization
Number of Slices	1445	13698	10%
Number of BRAMs	2	137	1%
Number of bonded IOBs	131	558	23%
Number of Slice Flip Flops	1703	27394	6%
Number of 4 input LUTs	2204	27394	8%

5.4 本章小结

本章首先介绍了课题研究过程所应用的软件开发环境 ISE 与嵌入式开发套件 EDK，还介绍了硬件开发平台 XUP Viterx PRO-II。然后重点阐述了本文设计的任务管理模块和信号量管理模块使用 ISE 8.2 进行功能仿真结果的分析，通过所得出的功能仿真结果充分显示了所设计的硬件逻辑电路的正确性，并且证明了此设计的可行性。

结论

本文首先分析了国内外实时操作系统的发展现状以及未来的发展趋势，然后具体分析了嵌入式实时操作系统 μC/OS-II 的内核原理以及 FPGA 的技术特点，提出了以 FPGA 为控制核心，利用硬件代替软件实现硬件任务调度器，确定了系统的总体设计方案，并实现了任务管理和信号量管理模块，利用 Xilinx 公司的 ISE 8.2 软件进行系统调试分析完成功能验证。

本文具体完成了以下的工作：

1．深入分析嵌入式实时操作系统 μC/OS-II 内核的工作原理，指出通过软件实现的传统实时操作系统存在的不足，以及未来实时操作系统的发展趋势，借助实时操作系统局部硬化的优势，提出了实时任务管理器硬件化的设计方案。

2．修改 μC/OS-II 中的一些数据结构，以便用硬件逻辑来实现，例如链表和队列等，由于链表只有通过访问前驱节点后才能访问后续节点的信息，浪费了访存的时间，不适合硬件逻辑的并行性。

3．任务管理模块重点设计了硬件任务调度器。在 μC/OS-II 内核基于优先级的抢占式调度算法的基础上扩展同优先级任务的调度算法，突破了原系统对任务数量的限制，去除了原系统对每个任务必须有不同优先级的要求。本文采用 FPGA 的片内寄存器来实现任务控制块 TCB，多任务潜在的并行性得到了充分地发挥。查找就绪任务不再通过就绪表，而是将调度器与 TCB 的状态位寄存器直接相连，节省了访存的时间。任务管理的硬件实现保持了系统调用的正确性，同时降低了处理器开销，减少了系统调用的执行时间。

4．信号量管理模块是是实时操作系统频繁运行的程序段，把信号量管理模块通过硬件逻辑来实现，可以充分提高操作系统的实时性。系统中信号量资源都有相应的信号量寄存器来存储，每种信号量资源依据 ID 号对应相应的任务等待列表，通过 P、V 操作方便地申请和释放信号量资源，进而改变任务等待列表中任务的状态。

本文已经用硬件逻辑设计了任务管理模块和信号量管理模块，但是还需要在以下几个方面做出改进和完善：

1．支持更多的调度算法。目前基于优先级的调度算法算法单一，可以提出更多适合实时任务需要的调度算法，如时间片轮转，增加调度算法的灵活性。

2．增强任务间的同步和通信。目前只设计了信号量管理模块，如果增加设计消息队列和消息邮箱，将会使各任务之间互相传递信息更加灵活。

参考文献

[1] 宋延昭. 嵌入式操作系统介绍及选型原则[J]. 工业控制计算机，2005，18(7)：41-42.

[2] 张力，王铁. 操作系统的固化[J]. 电脑知识与技术，2006，(8)：170-171.

[3] 刘秋平. 嵌入式操作系统. 科技创新导报[J]. 2007，33：12-13.

[4] 屈玉贵，赵保华，森下严. 计算机操作系统中进程管理的固化[J]. 计算机研究与发展，1987，24(11)：62-65.

[5] 崔建华，孙红胜，王保进. 硬件实时操作系统的设计与实现[J]. 电子技术应用，2008，(5)：34-37.

[6] 胡曙辉，陈健. 几种嵌入式实时操作系统的分析与比较[J]. 单片机与嵌入式系统应用，2007，(5)：5-8.

[7] 菜长安，万小霞. 4 种嵌入式实时操作系统的两种主要技术分析和选择[J]. 重庆工商大学学报：自然科学版，2007，24(2)：161-166.

[8] 季志均，马文丽，陈虎. 4 种嵌人式实时操作系统关键技术分析[J]. 计算机应用研究，2005，(9)：4-8.

[9] 高峰，王自强. 硬实时操作系统—LynxOS[J]. 计算机应用与软件，2005，22(3)：63-64.

[10] V.MOONEY III，J.LEE，A.DALEBY，et al. A comparison of the RTU hardware RTOS with a hardware/software RTOS[C]. In：Design Automation Conference (ASP-DAC'03)，2003，pp. 683-688.

[11] MOONEY，V.J，III，BLOUGH，D.M. A Hardware-Software real-time operating system framework for SOCs[J]. IEEE Design and Test of Computers Magazine，2002，19(6)：44-52.

[12] T. SAMUELSSON，M. AKERHOLM，P. NYGREN，et al. A Comparison of Multiprocessor Real-Time Operating Systems Implemented in Hardware and Software[C]. In：International Workshop on Advanced Real-Time Operating System Services (ARTOSS'03)，2003，pp. 44-52.

[13] T.NAKANO，U.ANDY. ITABASHI，et al. Hardware Implementation of a Real-time Operating System[J]. Proceedings of the Twelfth TRON Project International Symposium，IEEE Computer Society Press，Nov，1995，pp. 34-42.

[14] PAUL KOHOUT，BRINDA GANESH，BRUCE JACOB. Hardware Support

for Real-time Operating Systems[C]. CODES-ISSS'03, October 1-3, 2003, Newp -ort California, USA. Automation and Test in Europe Conference (DATE'03), 2003, pp. 45-51.

[15] T. NAKANO, U. ANDY, M. ITABASHI, et al. VLSI Implementation and Evaluation of a Real-Time Operating System[C]. Transactions of IEICE, Vol. J78-D-I, No.8 (Aug. 1995), in Japanese, pp. 679-685.

[16] P.KOHOUT, B.GANESH and B.JACOB. Hardware support for real-time operating systems[C] . Design , Automation and Test in Europe Conference(DATE'03), 2003, pp. 45-51.

[17] VINCENT J, MOONEY III. Hardware/software partitioning of operating systems[C]. In: Design, Automation and Test in Europe Conference (DATE'03), 2003, vol.2, pp. 338-339.

[18] 黄智伟, 王彦. FPGA 系统设计与实现[M]. 北京: 电子工业出版社. 2005: 13-154.

[19] 陈大林, 任祖平. 基于单片机的步进电机运行控制系统设计[J]. 伺服控制, 2005, 6(2): 54-56.

[20] 李志军, 李欣然, 石吉银等. 用 CPLD 实现多通道数据采集系统的 A/D 转换器控制电路设计[J]. 继电器, 2006, 34(12): 53-57.

[21] 尹震宇, 赵海, 张文波等. 一种嵌入式硬件多线程处理器的研究[J]. 东北大学学报: 自然科学版, 2006, (09): 239-244.

[22] 尹震宇, 赵海, 王金英等. 一种嵌入式处理器上的 HOS 设计[J]. 计算机工程, 2008, 34(05): 268-270.

[23] 张大伟. RTOS 调度器的软硬件实现[J]. 世界电子元器件, 2007, (1), 48-50.

[24] 王传福, 周学海. 提高硬件多线程处理器性能的方法[J]. 计算机工程, 2007, 33 (04): 239-241.

[25] 马吉军. CPU/FPGA 混合架构上硬件线程执行机制的研究[D]. 浙江大学, 2008.

[26] 高丰, 刘鹏, 姚庆栋. 基于系统集成芯片的 RT 的软硬件划分算法的研究[J]. 信号处理, 2001, 17 (增刊) : 582-585.

[27] 刘鹏, 李东晓, 姚庆栋等. 面向 HDTV 解码应用的 RISC 核的软硬件协同设计[C]. 中国电子学会电路与系统学会第 16 届年会(ICCAS2001). 宁波 2001-5.

[28] CHEN TIANZHOU, WU XINGLIANG, HU WEI. Research on OS-Aware

Embedded Power-Saving Architectre[C]. The 2rd Joint Conference on Harmonious Human Machine Environment, HHME2006, PCC06, pp. 52-59.

[29] ADOMAT J, FURUNAS J, INDH L, et al. Real-Time Kernel in Hardware RTU : A step towards deterministic and high performance real-time systems[C]. In Proceedings of eighth Euromicro Workshop on Real-Time Systems, 1996, pp. 683-688.

[30] 侯觅. 基于硬件的实时任务管理器的研究[D]. 上海交通大学, 2007.

[31] 张波. 基于 μC/OS-II 实时多任务调度算法的研究[D]. 广东工业大学, 2009.

[32] 田杰. 嵌入式 Linux 操作系统实时机制的研究[D]. 中南大学, 2006.

[33] 张旭, 牛连强, 付红旭. μC/OS-II 核任务调度模块的分析与改进[J]. 单片机与嵌入式系统应用. 2005, (4): 71-76.

[34] JEAN J. LABROSSE. 嵌入式实时操作系统 μC/OS-II[M]. 邵贝贝等译. 北京: 北京航空航天大学出版社, 2001: 178-185.

[35] 魏建业. 嵌入式操作系统 μC/OS-II 分析、移植与应用研究[D]. 山东师范大学, 2007.

[36] 吴欢. 基于 ARM 处理器的 μC/OS-II 现与应用研究[D]. 武汉邮电科学研究院, 2007.

[37] 曹聪, 范廉明. 操作系统原理与分析[M]. 北京: 科学出版社[M]. 2003: 49-55.

[38] T.NAKANO, Y.KOMATSUDAIRA, A.SHIOMI, et al. VLSI Implementation of a Real-time Operating System[C]. Proceedings of the ASP-DAC 97 Asia and South Pacific, January 1997, pp. 679-680.

[39] 高富强, 秦昌硕, 游纪原等. UC/OS-II 内核扩充时间片轮转调度算法的设计[J]. 计算机应用。2009(4): 1128-1142.

[40] 薛小刚, 葛毅敏. Xilinx ISE 9.X FPGA/CPLD 设计指南[M]. 北京: 人民邮电出版社, 2007, 8: 89-145.

[41] 谈世哲, 李健, 管殿柱. 基于 Xilinx ISE 的 FPGA/CPLD 设计与应用[M]. 北京: 电子工业出版社, 2009, 8: 16-25.

[42] 刘敏. 基于 FPGA 的多功能信号源系统总线设计与实现[D]. 国防科学技术大学, 2009.

[43] 田耘, 徐文波. Xilinx FPGA 开发使用教程[M]. 北京: 清华大学出版社. 2009: 144-145.

攻读硕士学位期间所发表的学术论文

[1] 李岩，王显山．实时操作系统任务调度算法的硬件实现[J]．计算机工程与应用，2010，46(35)．

致谢

本论文从选题到完成，每一步都是导师在李岩教授细心的指导和帮助下完成的，倾注了李岩老师大量的心血。导师拥有渊博的知识在学术领域，工作十分认真负责，导师的严谨治学风范和深厚的理论水平都给我留下了深刻的印象，将会让我受益终身。导师李岩教授无论在课题研究的理论上还是在实践中，都给了我很多的帮助。在此，谨向培养和帮助我的导师李岩教授表示崇高的敬意的衷心的感谢。

李老师不仅教授我们知识，为我们解决学习中所遇到的困难，而且还关心我们的日常生活，在社会压力日趋增大的今天，开导我们要有乐观的心态。能在李岩老师的指导下度过我的研究生生活，我感到十分的幸运，老师的教诲，将使我终生铭记。再次感谢李老师在这两年多研究生生活对我的悉心指导和帮助。

在课题的研究和论文的完成过程中，得到了实验室同学贾小梨、谷萍萍、李萍和曹帅等的建议和帮助，在此表示我的感谢。感谢我的朋友们对我的关心和支持。

最后，要感谢我的父母一直给予我默默的爱和支持，有了你们的爱和支持我才能顺利完成我的求学路。