摘要

半导体技术的发展使得基于 IP 核的 SoC 系统在嵌入式领域得到了广泛应用。其中主从关系架构的异构多核系统，是拥有用于任务控制的通用处理器核和面向应用定制的专用处理器核。

异构多核嵌入式系统上的操作系统，一般运行在通用处理核上，而计算量大的部分运行在专用处理器核上。现有软件系统对异构多核的支持，主要是用驱动程序，或一组系统调用把辅助核抽象成可以被应用程序直接使用的接口，应用程序一般直接编程以使用辅助核。这种方法在系统中运行单个应用程序的时候能获得很好的效果，并不适用于多个应用程序同时运行的情况。

本文认为可以从软硬件协同的角度，探索在该体系下的操作系统任务调度方法，调度系统的部分功能由硬件协同完成。

本文将辅助核任务分成预处理，任务运行以及后续处理三个阶段。预处理阶段生成辅助核可以运行的执行环境，包括代码和数据；任务运行阶段则是任务在辅助核上的运行过程；后续处理则是任务运行完成的后续处理工作。第二阶段运行在辅助核上，而预处理和后续处理阶段运行在主核进程上，管理着辅助核任务的创建和销毁，由操作系统的调度器进行调度。

本文设计的硬件调度器完成第二阶段辅助核任务的调度管理。这是一个支持先来先服务和动态优先级两种调度策略的 IP 核，可以按不同的调度策略调度辅助核任务在辅助核上运行。

通过本文设计的软硬件协同任务调度方法，测试可知辅助核计算任务吞吐量提高了 8.1%，用于辅助核任务调度的时间减少了 7.8%。

总之，本文设计的软硬件协同任务调度方法，通过设计硬件调度器，配合软件层面的操作系统调度系统，可以灵活而高效地支持异构多核的体系中的任务调度管理。

关键词 多核，异构，操作系统，编程模型，软硬件协同，调度

Abstract

As the development of semiconductor technology, IP based SoC system is becoming popular in the embedded system field. One is the heterogeneous multi-core system in the master/slave model. The general processor core as master is responsible for the controlling of system while the slave processor cores which are designed for specific purpose.

The operating system for this architecture is running on the master core, and the part in the application with large computation is running in the slave cores. For the slave cores, device driver or system call can be used as a interface for application. But they are not applicable when there are more than one application in the system.

This paper designs a hardware scheduler to reinforce the general software scheduler in the operating system for heterogeneous multi-core architecture.

In this paper the task for slave core is design in three parts, the pre-processing, running, post-processing. Pre-processing part is responsible for the creating of the context of the slave task, including the code and data, and the post-processing part is responsible for processing the results of computation. The running part, the computing of data is the core of slave task, is scheduled by the hardware scheduler while the pre-processing and post-processing running in the master core is scheduled by the software scheduler.

The hardware scheduler this paper designed is support first come fist service (FCFS) and dynamic priority scheduling strategies.

With this hardware-software co-schedule method, after testing and analyzing, the slave tasks throughput in the system is increased 8.1%, and the scheduling time for slave tasks is reduced 7.8%, than the general scheduling method in software.

In short, the method that hardware-software co-schedule method, hardware scheduler reinforce the software scheduler, can be flexible and efficient support the task scheduling of heterogeneous multi-core system.

Keywords Multi-core, Heterogeneous, Operating System, Programming Model, cooperating with hardware and software, Scheduling

图目录

表目录

第1章 绪论

1.1 嵌入式异构多核体系概述

半导体以及相应的集成电路制造工艺的迅猛发展，集成在芯片上的晶体管的密度不断的提高，使得越来越多的元件都可以集成到一块单一的芯片上。但是芯片由于晶体管数量增加而发热量越来越大，晶体管的电流泄露问题也随之越来越严重。随着晶体管数量的增加，各种各样的CPU设计技术用来提升CPU的计算性能，但是，晶体管数目增加一倍，所得到提升的性能却只有原来的30%[1]。处理器的设计在工艺设计和性能提升上都面临着巨大的挑战，而多核成为了解决这些问题的一种途径[2]。

嵌入式系统往往是针对特定应用设计的专门的系统，相比通用计算机，在系统的设计和实现上都较为简单，在半导体技术成熟的今天，在单个芯片上把各种不同用途的功能模块整合在一起，不但可以提升模块间的通信速度，提升系统性能，而且还可以减少芯片的面积，从而也降低芯片的能耗[3]。很多的专用于图像计算，数字信号处理的应用中，为了突破单个处理核速度的限制，都有采用单芯片多核心的设计方法[4][5]。

有着越来越多的面向特定应用的系统采用硬件多核的方式来提升系统的计算能力。在[3]设计了一个拥有多个DSP处理核的多核系统来提升系统计算能力。还有在[6]中实现了一个基于多处理核的用于加速JPEG计算的系统。在[7]中实现了一个用于加速MPEG2视频编解码的多处理器系统，并为此硬件系统专门设计了软件编程模型以及MPEG2编解码任务的调度方式。同样在[8]中实现一个用于MPEG-4计算加速的嵌入式SoC体系结构，以及[9]中是一个为HDTV应用而专门设计的异构多核系统，多个处理核中既有为HDTV计算专门设计的处理核，也有运行系统管理和HDTV计算任务调度的传统的嵌入式通用处理核。

可以看到为了满足各种各样的应用需求，各种针对特定应用而设计的异构多核系统越来越流行。在这种体系中，除了针对特定应用优化设计的处理核外，还有一个负责操作系统运行的通用处理核。不单单是嵌入式领域，在高性能服务器系统领域，IBM的CELL体系架构也采用了这样用一个异构多核的设计方式。

在这种异构多核的体系架构中，通常把运行操作系统的通用的处理核称为主处理核或者主核。而把针对特定应用设计的处理核称为辅助处理核或者辅助核。如前面提到的DSP，还有针对JPEG，MPEG2，MPEG-4以及HDTV应用设计的

处理核都属于这一类。这些辅助处理核往往只能运行针对特定应用设计的专用指令集，所以运行在辅助核上的辅助核任务需要主核的管理和控制。

1.2 多核操作系统和编程模型

在支持异构多核体系的软件系统中，操作系统运行在主核上，而辅助核则通过各种各样的方式抽象成软件接口。例如以驱动程序，系统调用，或者虚拟文件系统的方式[10]抽象出软件接口供应用程序直接使用。在驱动程序支持辅助核的方式中，从操作系统的角度来看，辅助核是一个拥有强大计算能力的设备，应用程序发挥辅助核强大计算能力的方式就是使用辅助核驱动程序。采用系统调用方式支持辅助核的方式中，支持辅助核的软件接口被包装成操作系统内核中的一组系统调用。而虚拟文件系统则是把辅助核的使用包装成一个虚拟文件系统，通过对文件系统的操作来实现对辅助核的操作。

在[11]文中设计了一个辅助核和主核在编程模型上地位相等的模型。在这个模型中，运行在辅助核上的应用程序可以和运行在主核上的应用程序一样调用系统调用，同时也可以和主核上的应用程序一样平等的被操作系统调度。这个模型是通过一个支持 DSP 的程序程序库和专门的软件中断及相应的中断处理函数来实现的。

异构多核体系提供了强大的计算处理能力，软件系统对这种体系结构的支持方式以及相应编程模型直接关系着硬件计算能力的利用率、程序员使用硬件系统的方便性等一系列问题。

总的来说，[11]中所使用的方式比较灵活，从应用程序的角度来看，辅助核和主核地位一样，应用程序能够使用传统的编程方式来使用异构多核中的各种不同类型的处理核。不过编程的方便性需要提供一组复杂的底层应用程序库和软件中断及相应的中断处理函数的支持。嵌入式领域的应用日新月异，如果对每一种不同的异构多核体系都开发一套程序库以及中断及中断处理函数，将是一个非常大的工作量，这必然会延长产品的开发周期。

而对于使用虚拟文件系统或者驱动程序的方式来支持辅助核的方式，在实现上较为简单。尤其是采用驱动程序的方式在开发比较快，而虚拟文件系统的方式虽然比驱动程序的方式复杂一点，但其在编程模型上的先进性可以使得开发应用程序时较为简单直观。

但是，无论采用驱动程序还是虚拟文件系统的方式支持异构多核体系，都是把辅助核硬件抽象成相应的软件接口，在应用程序中通过使用这些软件接口对辅

助核硬件进行直接的操作。这种方式用于测试辅助核的功能完整性以及应用在单任务的系统中是很合适的，实现方式和使用方式都很简单，但是，当有多个应用程序使用辅助核硬件的时候，由于这种软件直接操作硬件的方式，使得应用程序很难发现某个辅助处理核是否已经被其他程序所使用，辅助核资源使用的冲突就发生了。

此外，定义一组系统调用来支持辅助核也是一种方法。通过对辅助核进行底层抽象，辅助核在各个应用程序上的使用分配由操作系统内核进行调度。这样就可以解决辅助核管理的问题。但是从底层看，这需要复制大量的内核设施，同时还要的大量的新系统调用的配合，才能提供必要的功能。这种方法在实现上也颇为复杂[10]。

1.3 研究目的

从上一节对异构多核体系上的软件支持模型的分析中可以看到，在支持主核和辅助核这种体系的操作系统[12]中，如何灵活而有效的调度管理应用程序任务在辅助核运行成了一个很大的问题。无论是以驱动程序还是虚拟文件系统的方式把辅助核抽象成软件接口，都很难满足多任务的系统需求。而定义一组系统调用来支持的辅助核的方法却在实现上比较复杂，面对各种各样的异构多核体系，都需要实现各自独立的一组系统调用，软件系统的可移植性上比较差。

针对这些问题，本文认为可以从软硬件协同工作的方式来解决异构多核体系中的任务调度问题。

1.4 本文工作

本文提出了一个面向嵌入式异构多核体系的软硬件协同任务调度的方法。在此方法中，把运行在异构多核体系结构上的应用程序分成运行在主核上的任务线程及主核任务和运行在辅助核上的任务线程即辅助核任务两种类型。对这两种类型的任务采用分层次的调度管理方式，两类任务各自由相应的调度器进行调度管理。主核任务由操作系统调度器进行调度管理，辅助核任务则由本文设计的一个基于 IP 核的连接到片上总线的硬件调度器来调度管理。在本文中，重点对辅助核的管理和调度辅助核任务的硬件调度器做了深入研究。

本文主要在以下几个方面开展了研究工作：

1) 分析嵌入式异构多核体系的特性，并研究了针对异构多核体系的操作系统支持以及应用程序编程模型。

2) 设计软硬件协同任务调度的模型，辅助核任务由硬件调度器进行调度管理，而主核任务则由操作系统的调度器来调度管理。
3) 设计并实现了基于IP核的硬件调度器，该调度器支持先来先服务和动态优先级两种调度策略。并定义硬件调度器和辅助核之间的控制通讯接口。
4) 设计辅助核任务的编程模型。并按照此编程模型编写测试应用程序对整个软硬件协同任务调度模型进行验证。

通过软件和硬件协同调度的方式，把多个任务有序、高效的运行在多核系统中，降低操作系统支持异构多核系统的复杂度。并提供一个和现有编程模型相似的，易于程序员使用的软件编程模型，降低编程复杂度。

1.5 论文的组织

本文第一章是绪论，简要介绍了本文所做研究的背景，本文的研究内容和目的，还有介绍一下本文的组织结构。在第二章中，介绍了与本文相关的背景内容和相关的技术简介。第三章介绍了软硬件协同的任务调度方法，并简要的介绍了嵌入式异构多核体系。第四章详细介绍了硬件调度器的设计，同时介绍了硬件调度器和辅助核之间控制通讯接口以及硬件调度器的软件接口抽象。第五章介绍了支持操作系统对软硬件协同任务调度方法的支持，并介绍了相应的编程模型。第六章则介绍了软硬件协同任务调度器系统在FPGA开发板上的实现和测试验证。第七章对本文的内容进行了总结，同时对未来的工作做了展望。

1.6 本章小结

本章介绍了本文研究的背景，并介绍了本文研究的主要内容以及研究目的。同时介绍了本论文的组织结构。

第2章 相关背景及技术简介

2.1 多核体系结构和嵌入式系统

2.1.1 多核处理器的发展状况

按照摩尔定律的表述，集成电路上可容纳的晶体管数目，约每隔 18 个月便会增加一倍，而价格下降一半。这个由英特尔创世人戈登•摩尔提出的定律引领半导体市场 40 年之久，时至今日却面临着技术局限的限制。

一直以来，半导体行业，尤其是处理器芯片的设计坚持不断小型化的路线来提升主频，在芯片技术上直达纳米级的水平。但随着芯片电路“线宽”的越来越小，一个芯片集成近 10 亿个的晶体管，芯片中的电流也很容易泄露，芯片的功耗和散热成了很大的问题，阻碍了晶体管密度的进一步提升。而且，仅仅通过提高晶体管数量，提升处理器频率的方法，已经很难实现性能的突破性提升。体系结构从单周期，多周期，流水线，到超标量，SIMD，超线程（SMT），晶体管数量越来越多，实现技术越来越复杂，但性能的提升却越来越困难。Sun 微系统公司的首席技术架构师 Marc Tremblay 说，“增加一倍的晶体管数量，本文所能得到的性能提升只有原基础上的百分之二十。” [1]

同时，不管是服务器高吞吐的计算数据还是桌面系统的应用多样化，计算机计算对并行化的需求也越来越高。如 SMT（Simultaneous multithreading）技术，又称为超线程技术，在一个处理器上实现了硬件并行执行的能力，但是不断增加的芯片面积提高了生产成本，而且由于技术上的越来越复杂，使得设计和验证所花费的时间也变得更长[13][14]。此外，通过在一个计算机上汇集多个处理器，各个处理器之间共享内存子系统以及 I/O 子系统的对称多处理器 SMP（Symmetrical Multi-Processing）的体系结构在一定程度上提升的计算机系统的并行性及计算能力[15]。但是通过此种方式，把通用处理器其构建成集群的技术难度却非常大。

而一种称为 CMP（Chip Multiprocessors，单芯片多处理器）[16]技术似乎为处理器性能的继续提升提供了一个方式。CMP 是指在一个芯片上集成多个处理器核心的一种实现方式，CMP 技术又成为多核技术（Multi-core），从实质上讲，芯片上的每一个微处理器都是一个相对简单但是功能却是独立的单线程的处理器，不过这些微处理器之间联系较为紧密，可能共享二级 cache 或者一级 cache，并且常常是共享 FSB 总线接口。从 CMP 的设计思路上可以看到，它拥有处理器芯片

上的硬件多线程处理能力（SMT），同时在设计上同 SMP 有很多的相似之处。可以通过对硬件多线程处理技术（SMT）以及对成多处理器系统的对比来进一步分析 CMP 的优势所在。

处理器的设计通常提高处理器性能的方法都是基于提高处理器指令集并行（ILP, Instruction Level Parallelism）和线程级并行（TLP, Thread Level Parallelism），一般是把指令的执行分成几步，通过使用流水线的方式让多条指令在单个指令周期下同时在处理器上运行。按此方法，把流水线细分为更多的部分，使得这种深度流水线或称超级流水线能在单位时间内执行的指令数得到显著的增加，从而提高处理器的整体性能。

多发射类型的处理器一般包括超标量处理器（Super Scalar Processor）和超长指令字（Very Long Instruction Word，VLIW）处理器，允许在一个单位时间周期内发射多个操作执行以减少处理器的 CPI（Cycle Per Instruction,平均指令周期），从而更好的利用处理器的功能部件资源。超标量处理器为了能在单个时钟周期内发射执行多条指令，一般拥有多个功能部件，比如多个定点运算部件、浮点运算部件和 Load/Store 部件等。VLIW 与超标量不同，他是通过编译器来寻找指令级并行，然后编码成一条可同时发射多条微指令组合而成的超长指令[18]。

当处理器速度的不断增加，使得处理器访问内存的时间延迟相比与处理器的速度变得越来越不可忍受，数据的存取显得跟不上处理器的处理速度。通过设计使用 cache 可以很有效的缓解处理器速度和访问内存时间延迟之间矛盾。但是一级 cache 失效（miss）的时候还是会需要处理器等待不少的时钟周期从低级存储系统获得数据。处理器设计中的乱序执行（out-of-order OoO）技术，或者称为动态发射超标量，可以解决这个问题，在 cache 失效的时候，此技术可以让 CPU 不按顺序执行指令，而是执行那些不需要访问 cache 的指令。此外，这种处理器有多组运算单元来支持几组独立指令序列的并行运行。乱序执行技术的效率在很大程度取决于运行的指令序列的指令级并行（ILP）程度，如果运行的指令序列指令级并行程度很低，那么这种技术不会发挥很好的效果，而必须像先前一样等待 cache 失效后的许多个时钟周期。

SMT（Simultaneous multithreading）技术是乱序执行处理技术的一种扩展技术。其本质目的是增加可供处理器执行的指令序列。SMT 处理器可以在同一时间从不同的线程选取指令序列来执行。忽略高层的数据共享，不同线程的指令序列在数据上没有依赖性，这就增加了可供处理器运行指令级并行的独立的指令序列。同时，一个额外的好处是，由于不同线程指令序列的差异性，使得不同指令

序列可以利用处理器的相应不同的资源器件。

传统的多线程操作系统都支持多个进程或者线程同时利用一个处理器，操作系统通过把处理器的运算时间分割成时间片分配给各个进程或者线程。SMT 技术处理器可以在同一时间运行多个线程，在其中一个线程在等待外部资源而阻塞指令运行的时候，另外的线程可以共享利用处理器的资源继续运行，从而充分利用处理器的计算时间。严格来说一个线程运行在 SMT 技术的处理器比运行在非 SMT 技术的 CPU 上相对来说要慢一些，因为他和其他线程共享了 CPU 的计算资源。不过从宏观的角度来看，SMT 技术的 CPU 由于其计算资源的高利用率，它的运算效率相比非 SMT 技术的 CPU 要快很多[16][19][20]。

而对成多处理器（SMP）是通过把多个通用的处理器通过一条高速的板级总线连接在一起紧密耦合的一种体系结构[22]。通用的处理器可以是支持 SMT 的也可以是不支持 SMT 的。每个处理器共享同样的全局内存、磁盘和 I/O 设备。在体系结构上比较直观，从操作系统的管理角度，系统将任务对了对称分配给多个处理器之上，从而极大的提高的整个系统的数据处理能力。

虽然 SMP 能在一定程度上提高硬件的计算并行性，但是由于系统总线带宽限制，多个处理器之间协调的复杂性等原因，处理器的数目很难增加。这在很大程度上限制了 SMP 性能的提升空间。

CMP 和 SMT 一样，致力于发觉计算的线程级并行性，在大规模集成电路技术的发展，在芯片容量足够大的情况下，本文可以把 CMP 看成是对称多处理机结构或者说分布式共享处理机设计集成在同一个芯片里面，芯片内的各个处理器并行执行不同的线程或者进程。CMP 又称为多核（Multi-core），和 SMT 不同，CMP 上每个处理核心都是独立的，他们各自执行自己的指令级序列[21][23]。

2.1.2 多核处理器分类

多核处理器按结构的不同可以分成同构多核和异构多核两种，其判断的标准是根据芯片上集成的多个处理器核心是否相同。同构多核大多数由通用的处理器组成，多个处理器核心执行相同或者类似的任务。异构多核除了通用处理器核心之外，多集成了某些针对特定的应用进行设计的处理核心，如 DSP、ASIC、多媒体处理器、VLIW 处理器、网络包处理器等等。

1、同构多核处理器

美国斯坦福大学在 1996 开发的 Hydra 处理器[24]，集成了四个相同的 MIPS R3000 处理器核心。

IBM 在 2001 年推出的 Power4 处理器[26]包含两个处理器核心，两个处理核

拥有各自的一级指令缓存以及数据缓存和共享的二级缓存，同时 Power4 是第一款集成了处理器模块间高速 Fabric 光纤接口控制器的处理器，处理器模块之间带宽高达 35GB/s。

接着 IBM 又推出了包含了 4 个处理器核心的服务器处理器，每个处理器核心又支持 SMT。Sun 公司的 Niagara 芯片技术下推出的 UltraSparc T1 处理器[28]集成了 8 个处理器内核，利用 CoolThreads 多线程技术，该多核处理器可以同时并行处理 32 个线程。Niagara II 将在 2007 年诞生，内核依然是 8 个，不过可以同时支持 64 个线程，性能将会使 T1 芯片的两到三倍。

在 x86 微处理器领域，AMD 在 2005 年 4 月推出双核处理器 Opteron[27]，专用于服务器和工作站。Operon 引入 HyperTransport 超传输技术来消除传统的前端总线瓶颈，降低了内存的访问延迟。紧接着 AMD 又推出了 Athlon 64 X2 双核系统产品用于 PC 领域。英特尔在 AMD 之后也推出了双核至强（Xeon）以及双核 Pentium D[1]。在 2006 年 5 月，英特尔也发布了其用于服务器的双核处理器 Dempsey，6 月份又推出升级版的双核芯片 Woodcrest。Intel Xeon Clovertown 四核芯片也已经提前上市，随后的 Intel Xeon Tigerton 、AMD Barcelona、Intel Dunnington 等众多四核、八核服务器 CPU 也都将在 2007 年登场亮相[25]。

可以看到，同构多核处理器在 PC 以及服务器上的得到了广泛的应用。此外，异构多核处理器也已经开始进入成熟的应用。

2、异构多核处理器

多核处理器设计的一个问题是处理器芯片中单个核的尺寸大小问题。比如，服务器系统在同等芯片面积的情况下追求的是最大化的吞吐量。很明显，在这种情况下，处理器设计将采用数量较多而单个处理核结构简单的方案。不过一些应用情况，如桌面用户可能更关系的是单个程序的运行速度，也就是说需要有一个运行能力很强大的处理器核，也就是需要把多核处理器设计成数量相对较少而每个处理核又有相对较强大的处理能力。

但是现实情况下，应用的需求情况并不可能像上面提到的情况这么简单，很多时候很多应用既希望拥有强大的计算吞吐能力，又同时拥有强大的单线程的计算处理能力，同时有更加复杂的情况可能需要根据系统实际运行的上下文来判断。所以，如何设计处理核的大小就显得非常的重要。一种很好的选择就是采用异构多核的处理器设计方案，把简单的，低功耗的处理核同复杂但高性能的处理核整合在一起。

相比同构多核的设计的简单，异构多核需要在处理器设计，操作系统，编译

技术等多个方面做特殊的设计，不过其巨大的优势可以为服务器提高强大的性能。异构多核的芯片以 IBM Cell 为典型代表，它由一个或数个通用处理器核和数量较多的协处理器核组成。通用处理器负责操作系统的运行一些任务的分配和调度，而协处理器核则是对高密集的计算做特殊优化的向量处理器核，提供强大的计算吞吐能力。如图 2-1 所示，在 CELL 处理器中，包含了一个 Power 结构的处理器核心（PPE，Power Processor Element）和 8 个辅助处理核心（SPE, Synergistic Processor Element）[29][30][31]。

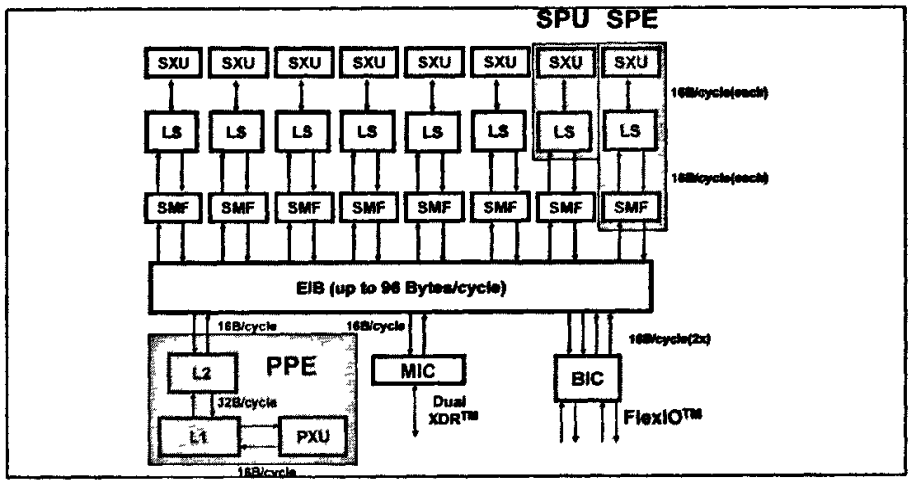

图 2-1 Cell 微架构的整体框架图，引用自[29]

而在 2005 年的 IDF 技术峰会上，Intel 对外公布了 Many Core 超多核发展蓝图。Many Core 采用的就是“主核心+协处理器”的异构架构，其中的某一个或几个内核可以被置换为若干数量的 DSP 逻辑，而保留下来的 X86 核心执行所有的通用任务以及对特殊任务的分派。

2.2 嵌入式多核 SoC

2.2.1 系统级芯片 SoC

集成电路设计从晶体管的集成发展到逻辑门的集成，现在又发展到 IP 的集成，即 SoC（System-on-Chip）设计技术。在电子信息产品的开发过程中，使用 SoC 技术可以有效的缩短产品的开发周期。系统级芯片 SoC 主要是在单个芯片上集成一个完整的系统，对所有或者部分的电子电路进行模块封装（IP）[37]。而一个完整的系统通常包含微处理器，存储器系统，以及其他一些外围的 I/O 设备控制器等[33]。

更进一步的说，SoC 就是在一个芯片上使用预定义模块 IP（Intellectual Property）来定制一个芯片系统的过程。从方法学的角度来看，SoC 是一套极大规模集成电路的设计方法学，包括 IP 核可复用设计/测试方法及接口规范、系统芯片总线式集成设计方法学、系统芯片验证和测试方法学。SOC 是一种设计理念，就是将各个可以集成在一起的模块集成到一个芯片上，他借鉴了软件的复用概念，也有了继承的概念。也可以说是包含了设计和测试等更多技术的一项新的设计技术[34][35][36]。

2.2.2 FPGA 和 SoC

ASIC（Application Specific Intergrated Circuits）即专用集成电路，是指应特定用户要求和特定电子系统的需要而设计、制造的集成电路。ASIC 的特点是面向特定用户的需求，品种多、批量少，要求设计和生产周期短，它作为集成电路技术与特定用户的整机或系统技术紧密结合的产物，与通用集成电路相比具有体积更小、重量更轻、功耗更低、可靠性提高、性能提高、保密性增强、成本降低等优点。

FPGA（Field Programmable Gate Array）即现场可编程门阵列。它是作为专用集成电路（ASIC）领域中的一种半定制电路而出现的，既解决了定制电路的不足，又克服了原有可编程器件门电路数有限的缺点。FPGA 的使用非常灵活，同一片 FPGA 通过不同的编程数据可以产生不同的电路功能。FPGA 在通信、数据处理、网络、仪器、工业控制、军事和航空航天等众多领域得到了广泛应用。随着功耗和成本的进一步降低， FPGA 还将进入更多的应用领域。

在过去的 20 年里，FPGA 在门数上已经提高了三个数量级，利用市场上大量存在的 IP（知识产权）形式，基于 FPGA 的 SoC 设计正在越来越受到设计工程师的欢迎。FPGA 技术的发展为硬件设计提供了更为方便的测试和验证方式。使用 FPGA 厂商提供的 EDA 工具能够很方便的进行体系结构描述，重复使用现有的硬件 IP 核缩短了系统的开发周期，最后通过 EDA 工具在 FPGA 上进行验证实现。

2.2.3 软硬件协同设计

嵌入式系统设计有两种方式：一是针对一个特定的硬件进行软件开发；二是根据一个已有的软件把它实现成硬件结构。前者是一个软件开发问题，后者是一个软件固化的问题。这两种设计方法都是相对独立的设计方式，硬件和软件之间的边界很容易产生一些兼容性的问题。使用软硬件协同设计的方式可以解决这个问题，从系统需求开始，对于硬件和软件的功能划分有一个明确的设计，在此设计的基础上对整个系统进行评估，从而找到一个比较理想的系统设计方案。在系

统的实现过程中，也始终保持硬件和软件的同步开发，这样，在系统实现过程能对系统进行实时的测试，最终实现满足设计的目标系统[38]。

2.2.4 芯片内高速总线

SoC 的设计过程中，最具特色的是 IP 复用技术。即选择所需功能的 IP 核，集成到一个芯片中用。由于 IP 核的设计千差万别，IP 核的连接就成为构造 SoC 的关键，对芯片内通讯的研究是一个很重要的方向[53][54]。片上总线（On-Chip Bus，OCB）是实现 SoC 中 IP 核连接最常见的技术手段，它以总线方式实现 IP 核之间数据通信，。与板上总线不同，片上总线不用驱动底板上的信号和连接器，使用更简单，速度更快。一个片上总线规范一般需要定义各个模块之间初始化、仲裁、请求传输、响应、发送接收等过程中驱动、时序、策略等关系。

目前 SoC 上使用较多的三种片上总线标准——ARM 的 AMBA、Silicore 的 Wishbone 和 IBM 的 CoreConnect[38]。

AMBA（Advanced Microcontroller Bus Architecture）总线规范是 ARM 公司设计的一种用于高性能嵌入式系统的总线标准。它独立于处理器和制造工艺技术，增强了各种应用中的外设和系统宏单元的可重用性。AMBA 总线规范是一个开放标准，可免费从 ARM 获得。

Wishbone 最先是由 Silicore 公司提出的，现在已被移交给 OpenCores 组织维护。由于其开放性，现在已有不少的用户群体，特别是一些免费的 IP 核，大多数都采用 Wishbone 标准。

CoreConnect 是由 IBM 开发的片上总线通信链，它使多个源的芯片核相互连接成为一个完整的新芯片成为可能。CoreConnect 技术使整合变得更为容易，而且在标准产品平台设计中处理器、系统以及外围的核可以重复使用，以达到更高的整体系统性能。

2.2.5 嵌入式多核 SoC

SoC 具有较高的集成度，很适合低功耗的应用，不过很多注重成本的设计都会在基本的系统上在集成包含更多的功能特性与可编程的元素。这是由于行业标准和协议的不断改变的事实密不可分的，可以知道，无论是 GSM 电话、MP3、DivX/MPEG4、网络通讯或者其他任何产品标准，其协议都以非常快的速度变化着，所以从生产的成本（工具、流水线的建立等）角度来考量，对某一特定的产品模块进行重复利用是一项很重要，这意味着系统对软件的要求更高了，因此也需要更多的软硬件协作。

软硬件协同设计会使得系统集成的时间延长，并产生大量的、只能针对特定

硬件使用的软件、因此大量的可编程微处理器被加入到系统中，虽然只要处理器的速度足够快，软硬件交互引起的性能下降问题都可以迎刃而解，但是无论从成本还是功能来看，通过提高处理器的主频来提升系统性能都是一种很不经济的解决方法。

比如，在利用 DSP 处理连续数据的应用中。开发者完全可以利用一个低端 DSP 芯片来处理标准的媒体流协议。相对于仅仅使用高端处理器组成的系统，这种把处理器和 DSP 多种计算单元混合的方法使得系统的功耗损失非常小，而且可以显著降低系统的开发成本。DSP 的特定指令集与专用存储器，总线结构使其能够完成较复杂的数字处理算法，但是这些特性往往不支持高级语言，因此基于 DSP 的很多应用必须采用汇编语言来编码。

由于不同 DSP 的汇编指令与编程模型不同，所以 DSP 之间的代码移植非常困难，这已成为应用开发的瓶颈，为解决这个瓶颈，一种方案是根据功能将应用代码分为两部分：必须由 DSP 执行的代码和可以被其他处理器执行的代码，DSP 只需要处理前者即可，这样就出现了多核（多处理器）设计。

总的来说，在嵌入式领域，由于有着各种各样的应用需求，从网络协议、MPEG 编解码、MP3 到 DSP 等等各种各样的需求在嵌入式 SoC 的实现中都以处理核的方式来实现。典型的嵌入式多核可以把系统按照通用嵌入式处理器核加专用于某项特定应用的处理器核的方式来实现。在这种异构多核的体系模式下，微处理器被确定为主处理核，而其他为专门应用设计的处理核可以称为辅助处理核。

其中主处理核运行嵌入式操作系统，充分利用微处理器的优势，包括合理的管理中断相应，大量的存储资源，虚拟的或者被保护的存储器以及简单的上下文切换等各种操作。而辅助处理核则运行某项特定应用的计算任务，充分利用辅助处理核的专长。

嵌入式系统异构多核体系架构，一般是一个或两个通用处理器核负责任务指派功能，而诸如协处理器、加速器和外设等功能都可以由专门的辅助核来完成。这种设计用以实现处理器执行效率以及整体性能的最大化。

嵌入式系统有着各种各样的应用， GSM 电话、MP3、Divx/MPEG4、网络通讯、加密解密等各种应用都是以模块的方式存在。尤其在 SoC 设计盛行的现在，通过把面向专门应用的 IP 核模块和通用微处理器以及各种外设的控制器组合，将形成一个控制和计算能力都很强大的 SoC 系统。

2.3 操作系统对多核体系结构的支持

2.3.1 支持多核体系的操作系统模型

多核处理器带来了更强大的并行处理能力、更高的计算密度以及更低的时钟频率，并且在很大程度上减缓了处理器的散热和功耗的问题。从上一节中可以看到，几大主要芯片厂商都推出了双核、四核甚至八核的多核处理器。为了有效的利用多核技术，需要计算机软件根据多核硬件平台进行针对性的设计。为了减少开发人员的工作，设计合适的操作系统是一项很重要的工作。例如，多核处理器管理共享的资源就是一个需要研究的问题，多个处理器核共享二级缓存、内存子系统、中断子系统以及外设。所以操作系统需要确保资源访问的安全性。此外，如何实现各个不同处理器核上的应用程序的互相协调，高效的IPC（进程间通信，interprocess communication），共享内存数据和同步原语（synchronization primitives）也是很重要的研究点。

为了解决这些众多的问题，首先要研究支持多核处理器的操作系统的模型。多处理核的操作系统模式主要有三种，非对称多处理（Asymmetric multiprocessing，AMP）模式，对称多处理（Symmetric multiprocessing，SMP）模式，混合多处理（Bound multiprocessing，BMP）模式[39]。

1、非对称多处理（AMP）模式

在非对称多处理（AMP）模式下，每个处理器核心都运行一个独立的操作系统或者一个操作系统的实例。在开发人员看来，AMP模式的操作系统运行环境和传统的单核处理器环境相类似，开发人员可以用传统的方式开发和运行程序。AMP操作系统的模式还可以有同构AMP和异构AMP之分。在同构的AMP操作系统模式中，每个核运行的操作系统都是一样的，开发人员只要选择一个分布式的操作系统即可。在异构的环境下情况就有点复杂，因为每个核上运行的操作系统都是可以不同的，这样的话就需要各个不同类型的操作系统之间有一个统一的通讯和协调方式。

在AMP系统中，通常情况下，系统的硬件资源需要在系统启动时就静态的分配。包括物理内存的分配、外设调用和中断处理器等。当然也可以采用动态分配硬件资源的方式，不过这样操作就会变得非常的复杂。在分配硬件的方法中，一种比较可行的方式是，对于所有的处理核，都使用一个同意的硬件抽象层，使得每个处理核在运行的时候都觉得自己是独自使用了全部的硬件资源。

2、对称多处理（SMP）模式

AMP模式虽然让开发人员可以延续以往的开发经验而不需要学习新的开发

技术，但是多个操作系统镜像运行在硬件上，使得硬件资源的分配以及多个任务的协调变得非常复杂。SMP 模式提供了一个解决方法，就是只运行一个操作系统的镜像运行在多核处理器上。这样，硬件资源都由唯一的一个操作系统进行管理和分配。而操作系统通过使用提供一些标准的原语提供给应用程序来共享使用系统的硬件资源。此外，由于各个程序之间使用一个操作系统的内的 IPC 机制，使得程序之间的协调变得容易而快速。

在高端服务器的 SMP 环境中，开发人员已经使用 POSIX 标准的 API（特别是 pthread API）开发出可以同时运行在单核处理器和多核处理器的应用程序。所以，开发人员可以借鉴以往运行在对成多处理器系统的上的操作系统。实际上，对称多处理器上运行的 SMP 操作系统，不需要做大的改动就可以运行在多核处理器上，并利用多核处理器的并行处理能力。一个设计良好的 SMP 操作系统允许多个应用线程协同地运行在任何一个内核上。这种协同性使得应用程序任何时候都可以利用芯片的整体计算能力。如果操作系统能提供适当的优先权和线程优先排序能力，就能帮助应用开发人员确保 CPU 为最需要的应用服务。

3、混合多处理（BMP）模式

混合多处理（BMP）模式是，一个操作系统的实例可以同时管理所有 CPU 内核，但每个应用被锁定于某个指定的核心。这是一种对 SMP 和 AMP 的折中。

2.3.2 支持对称处理机与多核的操作系统

传统的通用的操作系统都是支持多任务执行的，对唯一的一个计算核心通过分时，即把 CPU 的运算时间划分成长短基本相同的时间片，轮流分配给各个任务使用，从而实现单个 CPU 执行多个任务的能力。对于多核，或称为片上多处理器这种新的体系架构，操作系统并不需要多少修改就已经能够很好支持了，从软件角度来看，多核处理器和多路处理器（SMP）是一样的，所有针对单核多处理器的软件优化方式都可以用在多核处理器系统上。

例如 windows NT 之后的 Windows 系列操作系统，其中可以支持 SMP 的都可以支持多核。而对于 Linux 操作系统内核，其 SMP 版本的内核能够很好的支持多核 CPU， Linux 2.0 内核是第一个支持对称多处理器硬件的内核，在近 10 年的发展进程中，尤其从 1999 年到现在，Linux 对多处理器的支持越来越受到重视[40]。

在早些时候，Linux2.0 内核通过使用一种粗粒度的锁来保证系统的完整性，其原则就是：一个正在内核态运行的进程除非交出控制权或者要求进入睡眠，否则不能被另一个欲进入内核态的进程打断。也就是说在任意时刻只能有一个处理器是运行内核态的操作系统代码。这是一种安全而易于实现的方式，不过这种方

式对于充分利用多处理器的性能存在着很明显的不足。在 Linux 以后改进的版本中，操作系统内核采用了一种细粒度的锁来支持这种多处理器的体系机构。通过把操作系统的内核代码划分为临界区的方式，在保证系统完整性的前提下，提升了操作系统利用多处理器的性能优势。

不管是 WindowsNT 操作系统还是 Unix 的一些商业版本，如 Solaris，AIX 等以及开源的 Linux 操作系统，都采用一种核心级线程和用户级线程两种线程的模型，操作系统的调度在核心态完成，这样调度器可以专注于核心态线程在处理器上的调度。

在 2.6 版本的 Linux 中，提出了一种新的 O（1）的进程调度器，此调度器可以更好的支持 SMP 系统。它的优势就是在平衡了多个 CPUS 的负载均衡的同时又能有效的兼顾 cache 的有效性。在传统的调度过程中，当某进程从一个 CPU 迁移到另一个 CPU 上的时候，在原 CPU 上的 cache 内容即为失效，这将延迟任务在新的 CPU 上访问内存的时间延迟。

Linux 2.6 内核设计为给每个 CPU 建立一个就绪运行队列，并把就绪任务划分为 140 个优先级，高 100 个优先级用于实时任务，低 40 个优先级用户普通任务。并给每个就绪任务一段时间片，当任务把时间片用完后转移到另一个期满就绪队列并重新分配时间片，等到原活动就绪队列的所有任务的时间片均用完之后启用期满就绪队列来体会活动就绪队列。这样，调度器提供了一个各个任务公平使用 CPU 的机会，并实现了任务和 CPU 的绑定。通过给每一个 CPU 设立单独的任务队列，可以有效的实现系统中各个 CPUs 的负载均衡。

为了支持 SMP 系统，在操作系统启动阶段，把 CPUs 分为 BSP 和 AP 两种类型，由于 BIOS 代码并不是支持多线程的，所以在 SMP 中，系统必须让所有 AP 进入中断屏蔽状态，不与 BSP 一起执行 BIOS 代码。为了达到这一目的，可以利用两种手段：1、利用系统硬件本身进行处理；2、系统硬件与 BIOS 程序一起处理。在后一种方法中，BIOS 程序将其它 AP 置于中断屏蔽状态，使其休眠，只选择 BSP 执行 BIOS 代码中的后继部分。BIOS 要同时完成对 APIC 以及其他与 MP 相关的系统组件初始化过程，并建立相应的系统配置表格，以便操作系统使用。操作系统在 BSP 上完成内核加载，操作基本初始化工作后，通过使用处理器间中断系统（IPI）BSP 初始化各个 AP，最后让各个 AP 运行 idle 进程。最后 BSP 创建 1 号 init 进程，由 init 进程完成最后的系统启动工作。

在中断处理上，为了支持 SMP，在硬件上需要 APIC 中断控制系统的支持。Linux 操作系统的 SMP 版本和 UP 版本有所不同，对于 UP 版本，宏 Cli（关中断）

和Sti（开中断）就是简单的ch和sti指令；而对于SMP版本，内核不仅要禁止本地CPU，还要暂时避免其它CPU处理IRQ（中断请求），宏cli和sti实际上是对函数_global_cli()和_global_sti()的调用。global_sti()是开中断函数，它首先判断CPU是否正在处理IRQ，如果没有，则释放全局IRQ锁，然后允许在此CPU上继续进行中断。

2.3.3 支持主从式处理机的操作系统

在主从式的操作系统模型中，系统中所有的系统级服务以及内核态都运行在一个物理的处理器或处理器核上，这个处理器或处理器核就被称为主处理器，而系统中其他所有的处理器都称为从处理器，这些从处理器只能执行用户代码。

以用户态执行的进程可以在系统中任何一个处理器上执行。但是，当一个进程执行一次系统调用的时候，他就必须切换到主处理器上。一旦系统调用完成，进程就可以再次运行在任意处理器上。运行在从处理器上的用户态进程所产生的任何陷阱（比如缺页或者算术异常等）也会使得进程切换到主处理器上。此外，所有的设备驱动程序和设备中断处理程序也都只能运行在主处理器上。

从操作系统内核的角度来看，主从处理器这样的设计模式保留了单处理器的执行环境。这就能让单处理器的内核只需要很少就可以运行在这种模式的多处理器或者多核体系上，而且可以在任意数量的处理器或处理器核上执行[12]。

2.4 支持异构多核的编程模型

在支持异构多核体系的软件系统中，操作系统运行在主核，而辅助核则通过各种各样的方式抽象成软件接口。比如以驱动程序，系统调用，或者虚拟文件系统的方式[10]抽象出软件接口供应用程序使用。在驱动程序支持辅助核的方式中，从操作系统的角度来看，辅助核是一个拥有强大计算能力的设备，应用程序发挥辅助核强大计算能力的方式就是使用辅助核驱动程序。采用系统调用方式支持辅助核的方式中，支持辅助核的软件接口被包装成操作系统内核中的一组系统调用。而虚拟文件系统则是把辅助核的使用包装成一个虚拟文件系统，通过对文件系统的操作来实现对辅助核的操作

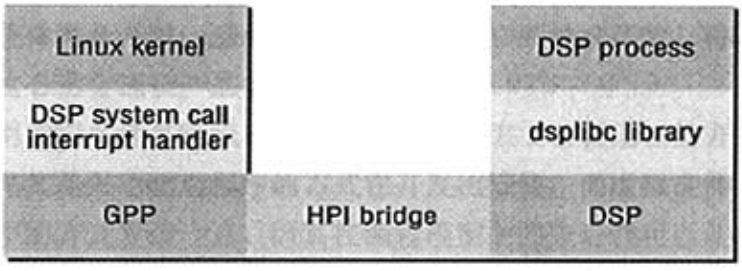

图 2-2 应用程序的执行模式[11]

在[11]文中设计了一个辅助核和主核在编程模型上地位相等的模型。在这个模型中，运行在辅助核上的应用程序可以和运行在主核上的应用程序一样调用系统调用，同时也可以和主核上的应用程序一样平等的被操作系统调度。这个模型是通过一个支持 DSP 的程序程序库和专门的软件中断及相应的中断处理函数来实现的。如图 2-2 所示，通过硬件 HPI 接口进行底层通讯，通过使用 dsplibc 库编写 DSP 程序，当调用系统调用时，触发一个软件中断给 GPP 上的为 DSP 编写的软件中断的处理函数，该函数调用相应的操作系统函数调用完成 DSP 程序的系统调用任务并返回相应的结果。

异构多核体系提供了强大的计算处理能力，软件系统对这种体系结构的支持方式以及相应编程模型直接关系着硬件计算能力的利用率、程序员使用硬件系统的方便性等一系列问题。

在 CELL 处理器中，通过使用虚拟文件系统[10]来使 CELL 的协处理器进行具体化，它和 Linux 很多类似的文件系统，例如 procfs、sysfs 或 mqueue 类似，这几种文件系统并不需要使用一个分区来存储数据，而是会将所有的资源都保存在内存中，同时可以使用一些常见的系统调用，例如 open、read 和 getdents，从而在用户空间和内核空间之间进行通信。

这种称为“spufs”的虚拟文件系统中，每一个文件目录都是一个协处理器可以执行的任务上下文，通过使用 open，read，write 等操作来操作 spufs 虚拟文件系统就可以实现对协处理器的控制和数据交互。

2.5 异构多核体系操作系统支持的不足

总的来说，[11]中所使用的方式比较灵活，从应用程序的角度来看，辅助核和主核地位一样，应用程序能够使用传统的编程方式来使用异构多核中的各种不同类型的处理核。不过编程的方便性需要提供一组复杂的底层应用程序库和软件

中断及相应的中断处理函数的支持。嵌入式领域的应用日新月异，如果对每一种不同的异构多核体系都开发一套程序库以及中断及中断处理函数，将是一个非常大的工作量，这必然会延长产品的开发周期。

而对于使用虚拟文件系统或者驱动程序的方式来支持辅助核的方式，在实现上较为简单。尤其是采用驱动程序的方式在开发比较快，而虚拟文件系统的方式虽然比驱动程序的方式复杂一点，但其在编程模型上的先进性可以使得开发应用程序时较为简单直观。

定义一组系统调用来支持辅助核也是一种方法。通过对辅助核进行底层抽象，辅助核在各个应用程序上的使用分配由操作系统内核进行调度。这样就可以解决辅助核管理的问题。但是从底层看，这需要复制大量的内核设施，同时还要的大量的新系统调用的配合，才能提供必要的功能。这种方法在实现上也颇为复杂[10]。

无论采用驱动程序还是虚拟文件系统的方式支持异构多核体系，都是把辅助核硬件抽象成相应的软件接口，在应用程序中通过使用这些软件接口对辅助核硬件进行直接的操作。这种方式用于测试辅助核的功能完整性以及应用在单任务的系统中是很合适的，实现方式和使用方式都很简单，但是，当有多个应用程序使用辅助核硬件的时候，由于这种软件直接操作硬件的方式，使得应用程序很难发现某个辅助处理核是否已经被其他程序所使用，辅助核资源使用的冲突就发生了。一种解决方法就是每个应用程序使用辅助核都需要采用互斥锁机制，但这样不利于计算任务在各个辅助核均衡调度，对硬件资源造成极大的浪费。

本文就以上一些问题提出了通过采用软硬件协同的方式对任务进行调度。辅助核任务通过本文设计的硬件调度器进行调度，作为传统操作系统调度系统的一个补充。

2.6 本章小结

本章介绍了与本文设计研究内容相关的一些技术背景内容。包括多核体系的发展和技术要点，以及嵌入式系统在运用多核技术上的一些相关技术点。尤其是关于嵌入式多核 SoC 的概念，做了比较详细的介绍。此外，本章也简要的介绍了操作系统对多核体系如何支持的一些相关技术内容，并分析了现有一些技术的不足，提出了本文设计的软硬件协同的任务调度模型。

第3章 软硬件协同任务的调度模型

3.1 多核操作系统调度的软硬件划分

在本文研究的异构多核体系中，有通用处理核和面向特定应用设计的辅助处理核（SPE，Synergistic Processor Element）两类处理器核。运行在通用处理核和运行在辅助核上的两类任务并不能相互迁移，所以需要调度相应的任务到相应的处理核上。由于辅助核的数量或者类型上可能很多，在这种情况下，需要考虑以下几个问题：

第一是辅助核 SPE 任务调度的灵活性的实现；

第二是软件编程模型的简单性；

第三就是要考虑系统的性能。

本文提出了设计一个控制 SPE 任务调度 IP 核，结合嵌入式操作系统的进程调度，来实现硬件调度和软件调度协同工作的任务调度模型。

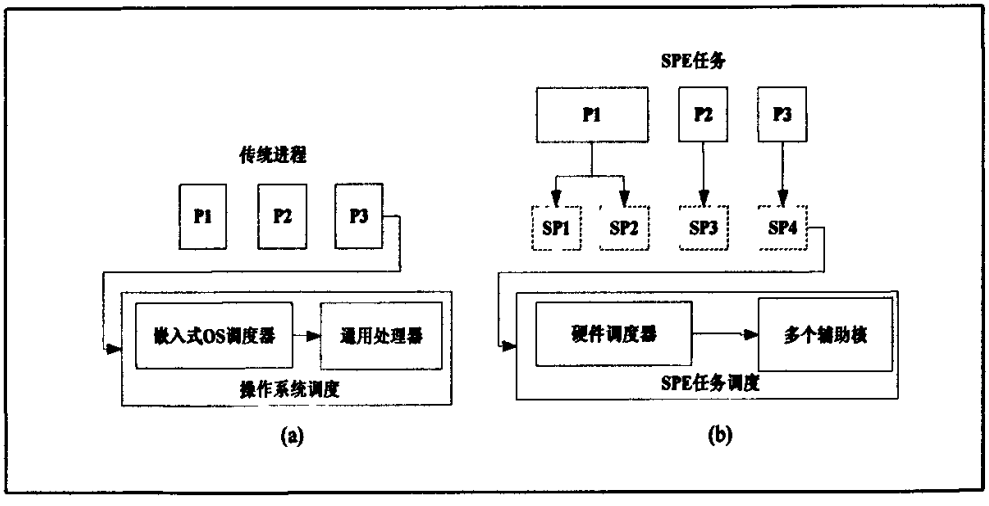

图 3-1 软硬件协同调度模型

在异构多核体系结构中，运行在通用处理器上的进程本文将称为传统进程，而运行在辅助核上的一个运行上下文本文将称为 SPE 任务。SPE 任务是由传统进程派生出来的，相对传统进程而已它并不是一个独立的进程。

如图 3-1 所示，（a）图是传统进程的调度过程。当传统进程启动的时候，嵌

入式 OS 把它添加到运行在主核上的嵌入式 OS 调度器的就绪队列中，并调度到通用处理器上运行。(b) 图是 SPE 任务的调度过程，如 (b) 图中传统进程 P1 派生出来的 SP1 和 SP2 这两个运行上下文，同时 P1 把 SP1 和 SP2 这两个 SPE 任务添加到硬件调度器中，由硬件调度器把 SPE 任务调度到 SPE 辅助核运行。

在本文设计的系统中，硬件调度器接受来自主核的 SPE 任务，并把这些 SPE 任务存放在一个硬件设计的就绪任务队列上。硬件调度器 IP 核负责管理所有的辅助核，管理各个辅助核的启动，重启并从任务队列上选取合适的 SPE 任务调度到各个辅助核上。另外，硬件调度器还负责把运行完毕的 SPE 任务发送回主存并通知运行在主核上的派生该 SPE 任务的进程。从操作系统以及用户应用程序的角度来看，只需管理和调用一个统一的逻辑接口，也就是硬件调度器的接口，就可以灵活方便的使用各个辅助处理核，至于 SPE 任务在辅助和上的运行，调度等工作则均由硬件调度器完成。在这个统一接口支持下，应用程序的编程模型也就变得简单，在操作系统上，也不需要特意编写各种系统调用接口就能够使得多个 SPE 完全透明。

3.2 软硬件协同的调度流程

传统进程的启动、调度均由嵌入式操作系统负责，兼容现有的操作系统调度器。而为了降低操作系统调度器的复杂度以及编程模型的复杂度，SPE 任务的调度管理由硬件调度器协助操作系统调度器进行调度管理。

由于辅助核是为专门的应用定制的，运行专门的 SPE 指令级。从操作系统的角度来看，辅助核不能进行资源管理，不能运行任何操作系统内核态的代码，不具备完全独立运行进程的能力。本文把运行在辅助核上的 SPE 任务分为三个阶段：

第一个阶段是 SPE 任务预处理部分，这一阶段负责创建一个能在辅助核上运行的 SPE 任务的上下文，包括符合 SPE 格式的 SPE 指令集以及所要处理的数据。这些代码和数据的集合将在下一阶段发送到硬件调度器，由硬件调度器调度到辅助核上执行。这一阶段由运行在主核上的进程完成，这个进程相应的也就是该 SPE 任务的控制进程。

第二个阶段就是辅助核 SPE 的执行部分， 辅助核从主存空间内载入在第一阶段初始化好的 SPE 任务上下文并启动 SPE 任务的运行。当任务执行完毕后，把执行结果数据同步回内存空间。

第三个阶段也是运行在主核上的，主要处理第二阶段在 SPE 上的处理结果。

以上三个阶段都是顺序执行，不能调换执行顺序，因此，SPE 任务的调度过程如下：

(1) 由运行在主核上的进程派生的 SPE 任务添加到 SPE 硬件调度器的任务队列里。运行在主核上的操作系统只要支持应用程序向 SPE 硬件调度器添加任务的功能即可。不具体负责 SPE 任务与具体辅助核的绑定；

(2) 接着 SPE 硬件调度器的调度单元从 SPE 就绪任务队列里面选取任务并分配到空闲状态的 SPE 上执行；

(3) 当 SPE 任务执行完毕，硬件调度单元通过中断通知主核相应的控制进程进行后期的数据处理。

硬件调度器的调度单元只负责从任务队列里选取最前端的一个 SPE 任务并把它分配给空闲的 SPE 进行执行。多个任务的执行顺序即任务的调度策略通过对就绪任务队列不同的排列方式来设定。

在软件上，操作系统通过提供一组系统调用来支持硬件调度器的使用。很好的屏蔽了多个 SPE 之间资源管理，底层执行平台的一些细节。

3.3 嵌入式异构多核 SoC 平台基础

在本文设计的异构多核 SoC 中采取主从式的模式，系统中包含了两种处理核，一种是通用处理器器核，本文称为主处理核，简称主核，另一种是针对密集型计算应用而定制的专用处理核，本文称为辅助处理核，简称辅助核。

整个 SoC 由一个通用处理器的主处理核 PPE（PowerPC Processor Element）、四个辅助处理核 SPE 以及内存控制器、LCD 控制器、以太网控制器、中断控制器、串口控制器等 IP 模块组成，如图 3-2 所示。

主处理核，辅助处理核以及其他各类 I/O 设备控制器均通过片上高速总线 CoreConnect 进行互联。CoreConnect 总线架构包括处理器本机总线（PLB）、片上外围总线（OPB）、一个总线桥、两个仲裁器，以及一个设备控制寄存器（DCR）总线[42][43][44][45]，在本文中并未使用到 DCR 总线。

OPB 总线连接低速和低性能的系统资源，而 PLB 总线连接高速设备。通过“PLB to OPB”桥 PLB 和 OPB 连接在一起。

其中五个处理器核以及内存控制和 LCD 控制器连接到 PLB 高速总线上。PLB 高速总线拥有高带宽的数据吞吐能力，可以满足处理器核之间，处理器核与内存之间的大数据量的传输和信号通讯的需求。而其他对速度要求不是很高的外围设备，如以太网控制器，串口控制器，中断控制器则连接到 OPB 低速总线上。

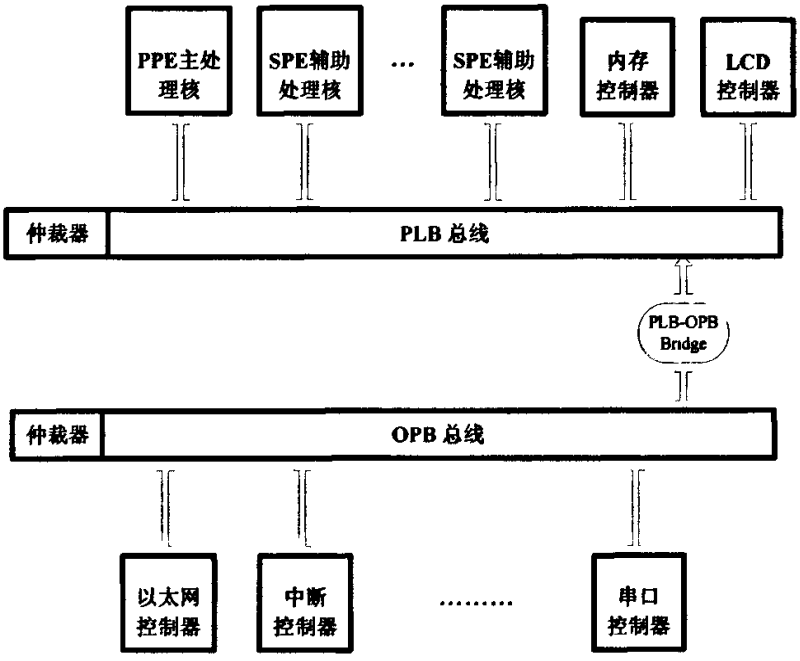

图 3-2 嵌入式异构多核体系设计图

主处理核 PPE 是嵌入式通用处理器 PowerPC405，该处理核兼容 PowerPC 的指令集。辅助处理核 SPE 则是使用了自主设计的 IP 处理核。SPE 是针对特定应用而定制的专用的处理核 IP，使用特定的 SIMD 的指令集。为了更好的理解和利用 SPE，在这里分别就 SPE 的各个模块进行介绍。

SPE 主要由以下几个模块构成：辅助处理器单元（Synergistic Processor Unit，即 SPU）、本地存储单元（Local Store，即 LS）、内存流控制器（Memory Flow Controller，即 MFC）及 DMA 模块、控制寄存器组和 MMIO（内存映射输入输出）模块用以主核 PPE 对 SPE 进行各种的操作和控制，最后是 SPU 连接到总线上的两个总线接口，如图 3-3 所示。

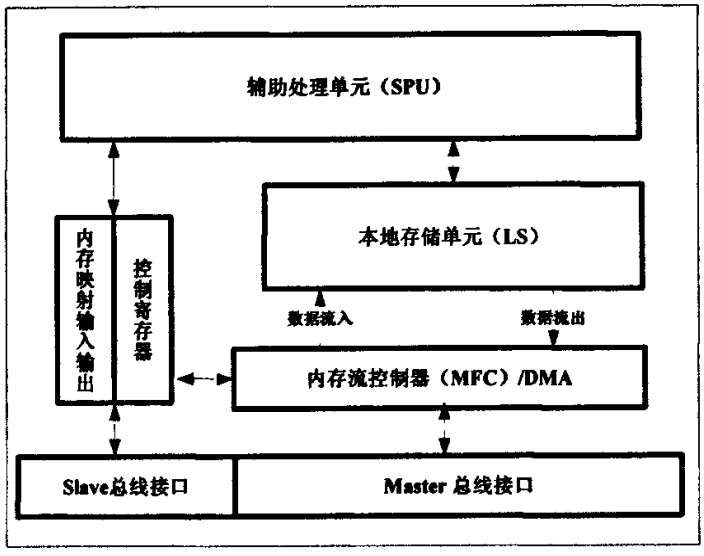

图 3-3 SPE 结构图

SPU 作为 SPE 的主体部分，是完成 SPE 的任务计算的功能部件，LS 供 SPU 进行数据的存取，物理上和 Cache 一样，但逻辑上 LS 是 SPU 能够直接访问的内存，是 SPE 的本地内存。MFC 及 DMA 模块负责系统内存与 LS 之间的数据传输、保护、同步等的操作，因此 MFC 和 DMA 模块联合使用连接总线上。

SPE 主要是用户大数据量的计算，实际使用中处理的数据可能是 32 位，64 位，128 位不等。为了提高计算单元的利用性，SPE 设计成支持 128 位 SIMD 指令的处理核。SPE 指令比 Intel 的 MMX 指令[46]及 SSE 指令[47]都要简单。当然，出基本的计算指令之外，SPE 也支持装载存储指令、跳转指令、逻辑运行指令等。SPE 指令集的所有指令是 32 位长度，其中前 6 位是统一编排的操作码。SPE 设计有 8 个 128 位的通用寄存器和 8 个 32 位的短通用寄存器。

SPU 作为 SPE 的核心计算部件，能够快速的进行数据的运行操作，是一个特别设计的处理器内核，包括有取指模块、指令译码模块、寄存器组、计算模块、内存读写模块等。所有指令序列按序递交、执行，每个周期执行一条指令。SPU 并没有实现流水线设计。

在逻辑上，LS 是 SPE 的内存存储单元，是 SPU 取指以及数据读写操作的唯一场所。LS 针对 SPU 的计算特性而专门设计。LS 的数据带宽设计成 128 位，同时为了满足 SPU 的数据读写以及在 DMA 模块控制下与主核内存空间进行数据传输的同时进行，而且保证两者速度的要求，LS 使用双口 SRAM 的设计方式。此

外，由于 SPE 不运行任何有关操作系统的内核态的核心代码，所以 LS 空间完全有用户程序自己维护和管理，也不需要提供 LS 空间的保护支持。LS 上的硬件异常，错误访问等问题都需要程序员在编写程序时通过遵循一定的编程模型来保证 LS 的安全性。

3.4 本章小结

本章介绍了软硬件协同的任务调度方法的模型，分别就多核操作系统调度的软硬件划分，以及调度流程做了阐述。并介绍了作为本文研究基础的嵌入式异构多核 SoC。

第4章 硬件调度器软硬件设计

4.1 硬件调度器IP核框架设计

硬件调度器以独立IP核的方式连接到PLB总线上，如图4-1所示。在系统实现上，硬件调度器和SPE辅助核以相同的方式连接到总线上，硬件调度器和四个辅助核SPE在总线上地位是一样的。不过

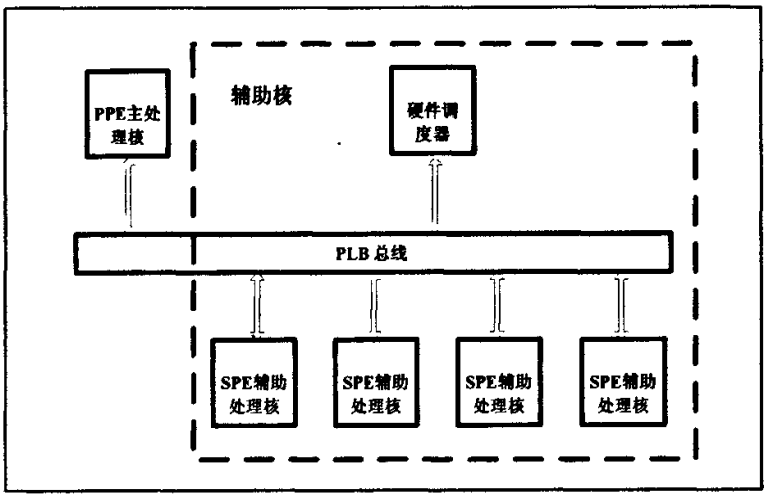

图 4-1 SoC中的硬件调度器

在逻辑上，硬件调度器是辅助核SPE的管理者，也正因为这种逻辑上的管理者和被管理者的关系，辅助核SPE由硬件调度器统一管理，硬件调度器支持中断，而四个辅助核SPE不支持中断。

从PPE主处理核的角度来看，硬件调度器和四个SPE是一个整体，主核只需控制管理硬件调度器，而四个SPE的管理工作则全部有硬件调度器负责。

在硬件调度器的具体设计上，如图4-2所示，有以下几个模块：总线接口模块、控制寄存器组和中断模块、任务队列模块、硬件调度器的控制模块。

在总线接口的设计上，硬件调度器采用了和SPE相同的Slave和Master两路总线接口的方式，Slave 总线接口是设计为硬件调度器被主核控制的接口，Slave接口不能主动往总线上发送请求，只能被动的等待和接收并处理从总线上过来的请求。而Master总线接口不仅可以接收并处理来自总线的服务请求，同时还可以主动发送请求到总线以控制其他连接到总线上的IP核模块。Master总线接口是硬

件调度器控制辅助核 SPE 的主要接口。

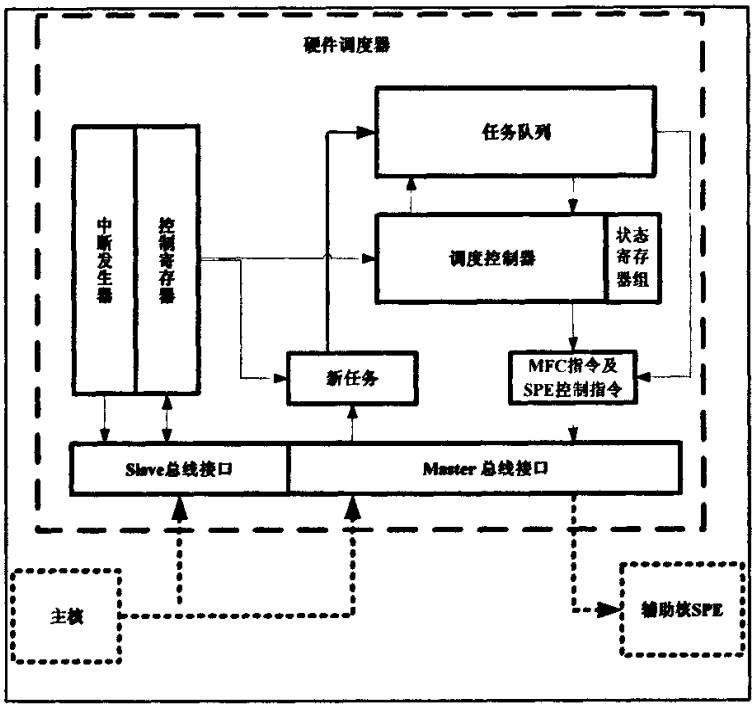

图 4-2 硬件调度器的模块设计

控制寄存器组和中断发生器控制着硬件调度器的运行，是主核控制管理硬件调度器的硬件接口。而硬件调度器里的调度控制器则是调度器执行任务调度控制的管理模块，包括任务队列的管理，任务到辅助核 SPE 上的分配，SPE 启动运行管理等工作都由这个模块进行控制。

任务队列是一系列 FIFO 硬件缓存部件的集合，通过不同规则来组织这个任务队列可以实现 SPE 任务按照不同的策略进行任务的调度。硬件调度器支持任务先到先服务和动态优先级两种调度策略。

4.2 SPE 任务队列的管理

4.2.1 任务控制模块的格式

就像运行在主核的传统的应用程序进程用进程控制块（PCB）来管理相应进程的数据状态信息一样，在辅助核上运行的 SPE 任务也需要有一组的数据来标识 SPE 任务的属性。在本文中，把这种表示管理 SPE 任务的进程控制信息称为 SCB。通过对 SCB 的管理来实现对 SPE 任务的管理。

表 4-1 SPE 任务控制模块格式

名称	长度	描述
Pid	Uint32	主核控制线程的 pid
Phy_add	Uint32	SPE 任务在主存空间的起始地址
Size	Uint32	SPE 任务的大小
Priority	Uint16	SPE 任务的优先级
Class	Uint16	SPE 任务的类型

在表 4-1 中定义了一个 SCB 的一些最基本的信息。32 位的 pid 是与该 SPE 任务关联的运行在主核上的线程的进程号。用来管理 SPE 任务与控制线程之间的映射关系。32 位的 Phy_add 定义了 SPE 任务在主存储空间，也就是系统内存上的地址空间，通过配合使用 32 位的 Size，也就是 SPE 任务的代码和数据段的大小来控制辅助核 SPE 使用 DMA 模块把 SPE 任务从主存空间搬运到 SPE 的本地地址空间 LS。Priority 定义了 SPE 的优先级。

在本文中设计的硬件调度器支持两种策略的任务调度管理方式，一种是先来先服务（FCFS）的的串行化的任务调度策略，另外一种是基于动态优先级的任务调度策略。在先来先服务的调度策略下，Priority 属性域的值不起任何作用，只有在任务调度策略采用基于优先级的策略时，Priority 才会被硬件调度器使用。在这种模式下，SPE 任务的优先级继承生成该 SPE 任务的管理线程的优先级。至于 Class 属性域是设计用来区别不同类型的 SPE 任务的。在本文的设计中，所有的 SPE 任务都是基于同一种辅助核 SPE 以及基于同一指令集的 SPE 任务下完成的。Class 属性并未使用。

4.2.2 任务队列的组织形式

本文所设计的硬件调度器支持先来先服务调度策略和基于动态优先级调度策略两种策略。任务调度策略均通过对硬件调度器中的任务队列的组织来实现。

采用何种调度的策略，有很多方面需要考虑，其中可能的有：

1. 资源利用率——使得 SPE 的使用力尽可能的高；
2. 响应时间——使得某个应用程序的响应时间尽可能的少；
3. 公平性——确保每个应用程序都能够获得合理的 SPE 的使用份额。

对辅助核来说，任务的运行是串行化的，它并不支持任务的抢占和被替换。所以如传统操作系统的时间片轮询调度，抢占调度等方式并不适合本文这种专为特定应用设计，追求数据的吞吐处理能力的系统。在本文中，硬件调度器着重设

计串行化的先来先服务的调度策略，同时也根据主核应用程序的优先级来实现硬件调度器基于动态优先级的调度策略。

任务队列是一系列 FIFO 硬件缓存部件的集合，通过一定规则组织这个任务队列可以实现 SPE 任务按照不同的策略进行任务的调度。任务队列的组织方式由硬件调度器的调度控制器进行控制。

在先来先服务（FCFS）的调度策略下，任务队列的实现方式较为简单，硬件 FIFO 缓存以一条线性顺序连接起来。在使用过程中，新的 SPE 任务的添加在线性链表的一端，而任务的调度执行则从另一端选取。

基于动态优先级调度的任务队列实现上稍微复杂一点。操作系统应用程序的 0 到 70 的优先级被硬件调度器划分成高优先级，普通优先级，低优先级三个优先级。三个优先级各自使用一个独立的任务队列，在添加 SPE 任务的时候，根据 SPE 任务的优先级添加到相对应的独立的任务队列。而硬件调度器的调度模块从队列中选取任务运行的时候，首先选取的是高优先级任务队列中的任务，当高优先级任务执行完毕后在选取低优先级的任务队列中的任务。

可以看到，当有高优先级的任务在队列中的时候，低优先级的任务永远不会有被调度的机会。为了防止低优先任务饿死的情况，硬件调度器每间隔一段时间把任务队列中的任务的优先级提升一次，也就是把普通优先级的任务变为高优先级的任务，把低优先级的任务变为普通优先级的任务。在本文设计的系统中，当 SPE 任务被调度的计数到一定值的后，更新任务队列中就绪任务的优先级。在本文中，初步把这个计数设顶为辅助核数目的两倍。

如果要支持多种类型的辅助核，任务队列就需要针对不同的辅助核类型建立不同的队列，也就是说辅助核任务按照他的 Class 属性分别添加到各个不同的任务队列里面。在本文中，只实现基于同一类型任务队列的调度管理方法。如果要进行扩展使得整个调度器支持多个类型的辅助核任务，则只要对现有的任务队列进行扩展，不同类型的辅助核任务使用各自不同的任务队列。

4.2.3 控制通讯原语

系统中存在两类地址空间，全局地址空间和 SPE 本地地址空间。全局地址空间即系统地址空间，也就是主核运行的地址空间。本地地址空间则是 SPE 辅助核中 LS 的地址空间。操作系统以及应用程序的主进程和若干子线程均运行在系统空间也就是全局地址空间中，而 SPE 任务则运行在 LS 本地地址空间中，包括取指，数据读写操作都在本地地址空间中进行。

主核和辅助核 SPE 有着完全独立的地址空间，SPE 操作的所有数据都来自

LS，为了从主存把SPE的操作数据，包括SPE执行代码和操作数据，搬运到SPE的LS上，需要借助SPE的DMA控制模块从主存获取数据。因此在，SPE的设计过程中以及硬件调度器对SPE的操作控制原语中需要有一个统一的通讯方式。表4-2中定义了SPE进行存储控制的原语定义。定义了Get和Put两天通讯原语指令。Get指令把内存里面地址以Source Addr（32 bit）开头，长度为Length（12 bit）的内容，放到LS中地址从Target Addr（16 bit）开头的地方。而Put指令将LS里面地址以Source Addr （16 bit）开头，长度为Length（12 bit）的内容，放到内存里面地址从Target Addr（32 bit）开头的地方。

表 4-2 存储控制原语的格式

指令名称	操作码	操作数		
Get	4`b0001	Length (12`b)	Target Addr (16`b)	Source Addr (32`b)
	把内存里面地址以Source Addr（32 bit）开头，长度为Length（12 bit）的内容，放到LS中地址从Target Addr（16 bit）开头的地方			
Put	4`b1101	Length(12`b)	Source Addr (16`b)	Target Addr (32`b)
	将LS里面地址以Source Addr (16 bit)开头，长度为Length（12 bit）的内容，放到内存里面地址从Target Addr（32 bit）开头的地方			

表 4-3 SPE控制原语的格式

指令名称	操作码	操作数
Run	4`b0101	60`b000000000000000000000000000000000000000000000000000000000000
	启动SPE开始运行	
Stop	4`b0110	60`b000000000000000000000000000000000000000000000000000000000000
	停止SPE的运行	

而表4-3则定义了硬件调度器控制SPE运行的控制原语。通过这两条控制原语控制SPE的启动和停止。

4.3 SPE任务的调度流程

4.3.1 新SPE任务的添加

SPE任务有运行在主核上的线程生产一个SPE任务上下文，通过使用硬件调度器抽象出来的软件控制接口，把SPE任务控制模块的内容发送给硬件调度器。而SPE任务在具体SPE上的执行，调度等所有控制过程均由硬件调度器来完成。

一个新的SPE任务从控制线程里生成，并把生成的SPE任务添加到硬件调度器的任务队列里，如图4-3所示。

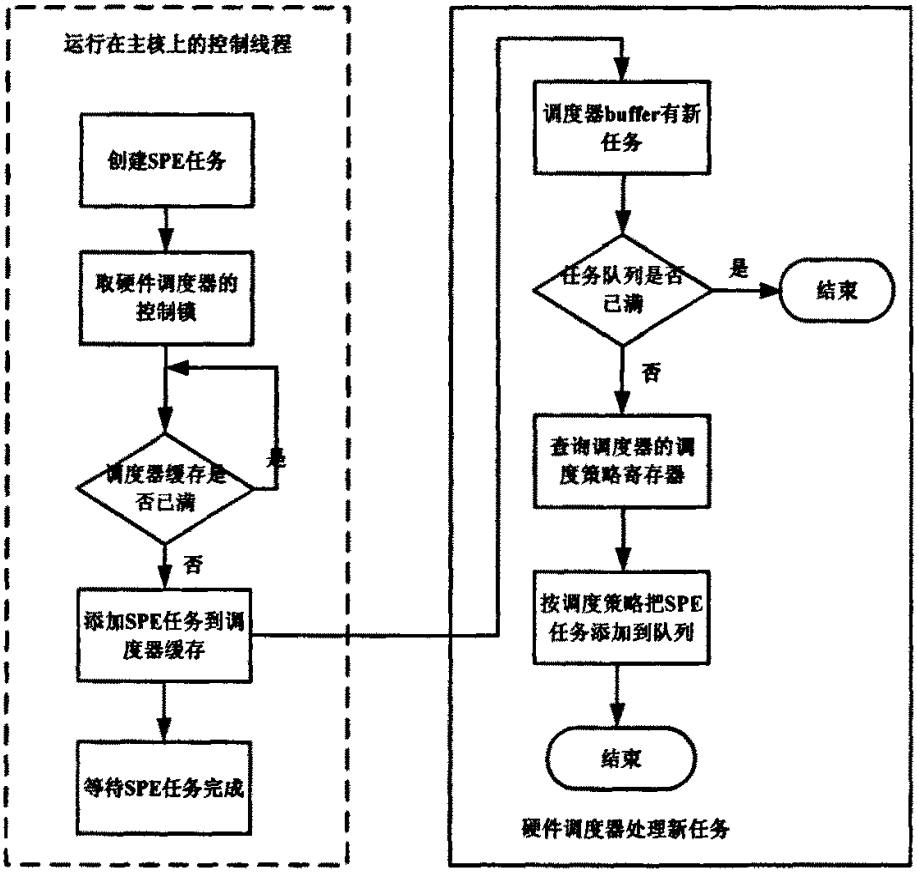

图 4-3 新 SPE 任务添加到硬件调度器任务队列的流程

4.3.2 任务到 SPU 处理核的分配

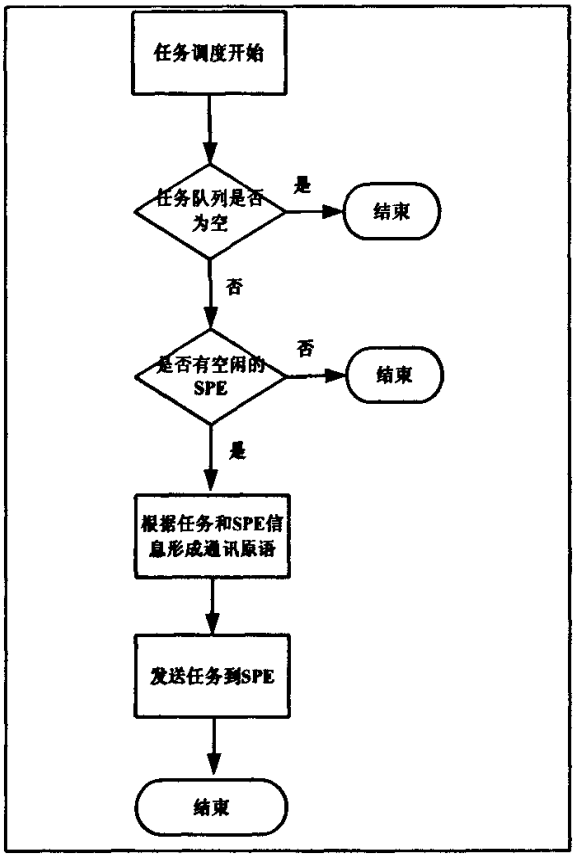

图 4-4 SPE 任务调度控制过程

如图 4-4 所示，由于任务调度策略的管理已经通过对任务队列排序的方式在添加任务的过程完成了，所以在硬件调度器选取 SPE 任务进行调度的时候只需要从任务队列的一端选取一个任务即可。选取了 SPE 任务之后，硬件调度器通过 SPE 任务的 SCB 模块的信息以及硬件调度去标识 SPE 状态的状态寄存器形成一组用于控制 SPE 启动以及从主存载入 SPE 任务的控制通讯原语指令。并通过硬件调度器的 Master 总线接口发送到相应的辅助核 SPE。

4.3.3 SPU 任务执行完毕后的处理

所有的辅助核 SPE 都由硬件调度器进行统一的管理，为了节省系统的中断资源以及降低 SPE 设计开发的复杂度，所有 SPE 都不支持中断。所以硬件调度器定期对每个 SPE 采用轮询的方式查询状态。

在硬件调度器中使用两个 32 位的寄存器来标识 SPE 的空闲与否以及任务完

成与否。32 位的寄存器每一位对应一个辅助核 SPE，最多可以标识 32 个辅助核 SPE 的状态。这两个状态寄存器分别称为 SPE_SR 和 TASK_SR，两个 32 位的状态寄存器在本系统均只是使用了前 4 位，对应于本系统中的 4 个辅助核 SPE。

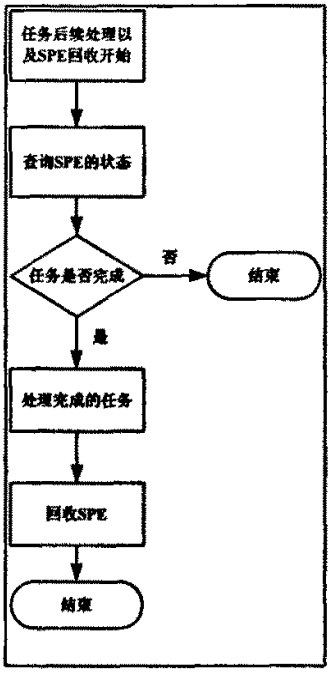

图 4-5 SPE 任务的后期处理过程

如图 4-5 所示，对各个辅助核进行轮流查询的目的是为了处理 SPE 任务完成后的一些后续工作，同时回收 SPE 处理核，供下一轮的任务调度使用。首先是查询各个 SPE 的状态，对于在 SPE_SR 标识为 0 的未被使用的 SPE 可以略过，查询 SPE_SR 相应位为 1 的 SPE，并查看 SPE 的运行状态，如果 SPE 上的任务已经完成，则把 TASK_SR 的相应位置为 1，并处理 SPE 任务的后期处理任务，把 SPE 上 LS 的内容通过 DMA 以及总线接口同步回主核上的应用程序控制线程。最后把 SPE_SR 相应位置为 0。通过硬件调度器的中断通知主核处理相应的 SPE 任务。

4.4 硬件调度器的软件接口

4.4.1 功能抽象

应用程序使用的硬件调度器方式是通过一个统一的控制接口进行控制。从上一节有关 SPE 任务的创建，调度等流程图可以看到，在应用程序软件编程的接口设计上，直接和硬件调度器的只有新任务添加到硬件调度去器这么一个过程。该

接口首先获取硬件调度器的控制操作权限，也就是获取硬件调度去器的硬件锁。此硬件锁的作用是确保任意时刻只有一个控制线程操作硬件调度器，用以保证硬件调度器的数据完整性。接着，通过对硬件调度器的控制接口进行操作，把如表4-1定义的SPE任务进程控制模块的信息发送给硬件调度器。操作成功后，控制线程进入休眠，等待硬件调度器再次把它唤醒。

4.4.2 软件接口

在软件的实现上，主要定义以下几个接口：

Getlock和Freelock两个接口，这两个皆空用于控制线程向硬件调度器获取操作控制权限和释放这个权限。在控制线程要进行SPE任务添加之前以及操作完成之后均需要配对使用这组操作。

Send_SPE(&SCB)，这个接口是负责控制线程向硬件调度器发送SPE任务的。SCB结构即是SPE任务的各项信息。

Run_SPE 接口，这个接口是紧接着上面 Send_SPE 接口之后运行的。在Send_SPE成功运行之后，控制线程运行Run_SPE使得自己进入阻塞状态等待SPE任务的执行结束。

SPE_end接口，这是一个以中断处理函数实现的函数接口。当SPE任务执行结束之后，硬件调度器产生一个中断，中断处理函数接口SPE_end同过SPE任务的SCB信息唤醒相应的控制线程，在控制线程同硬件调度器之间的协作控制下，完成SPE任务的后期处理工作。

4.5 本章小结

本章详细介绍了硬件调度器的设计，对硬件调度器的各个模块进行详细阐述。并介绍了硬件调度器调度SPE任务的流程以及需要的控制通讯原语。最后，介绍了支持硬件调度器以及应用程序使用硬件调度器的软件接口抽象。

第5章 嵌入式异构多核操作系统和编程模型

5.1 操作系统对软硬件协同任务调度的支持

选用的嵌入式 Linux 操作系统的所有系统服务以及应用程序内核态部分都在主核上运行。SPE 不承担任何与内核态相关的代码运行。SPE 由统一的硬件调度器进行管理，操作系统不直接操作管理 SPE 处理核，而是通过对硬件调度器增加一组系统调用支持来控制硬件调度器。如图 5-1 所示，操作系统以及应用程序通过一组系统调用接口对硬件调度器进行操作。

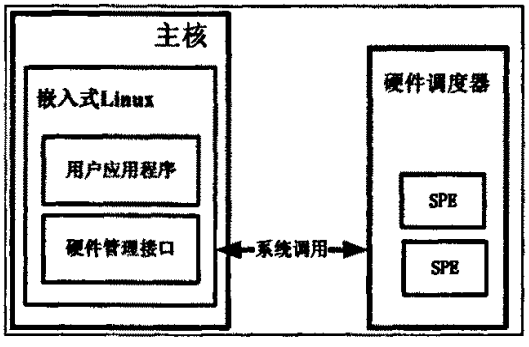

图 5-1 操作系统支持模型

5.2 运行在 SPE 上的程序上下文

SPE 是一个有着自己独立指令集的处理核，指令集简单，处理核的功能也较单一。在这里，把由 SPE 指令集及其处理数据组成的，在 PPE 的控制下，能独立运行在 SPE 上的代码和数据集合称为 SPE 程序上下文或 SPE 任务。SPE 不直接支持复杂的内存管理（包括分段，分页，堆栈等），所以 SPE 程序上下文是由连续的指令序列段和数据段运行在 SPE 的 LS，没有设计显式的分段。SPE 任务的存储空间就是直接寻址到本地 LS 的地址空间。

在 SPE 任务的编写划分过程中，为了是 SPE 实现的简单化，要求 SPE 任务的大小，包括指令序列以及所操作的数据不超过 LS 的容量。这样，当应用程序在主核上启动运行后，通过 SPE 提供的控制通讯接口操作 DMA 把 SPE 任务从应用程序在主核的全局地址空间上装载到 LS 的地址空间，接着控制 SPE 去执行 SPE 任务。此外，要尽量让 SPE 任务的大小划分成接近于 LS 的大小，这样可以最大限度的利用 LS 空间，发挥 SPE 的执行能力[51]。

5.3 支持软硬件协同任务调度的编程模型

如图 5-2 所示，使用不同的编译工具分别编译汇编运行在主核上的主程序以及可以运行在 SPE 上的 SPE 目标文件。在本系统的设计中，由于主核程序和 SPE 程序数据交互非常少，所以没有采用将主核目标文件和 SPE 目标文件链接成同意的可执行文件的方式[52]。把运行在主核上的程序使用 PowerPC 编译工具从源代码编译成可在 PowerPC 的主核上运行的可执行文件。而 SPE 汇编源代码则汇编成 SPE 目标文件单独存储。

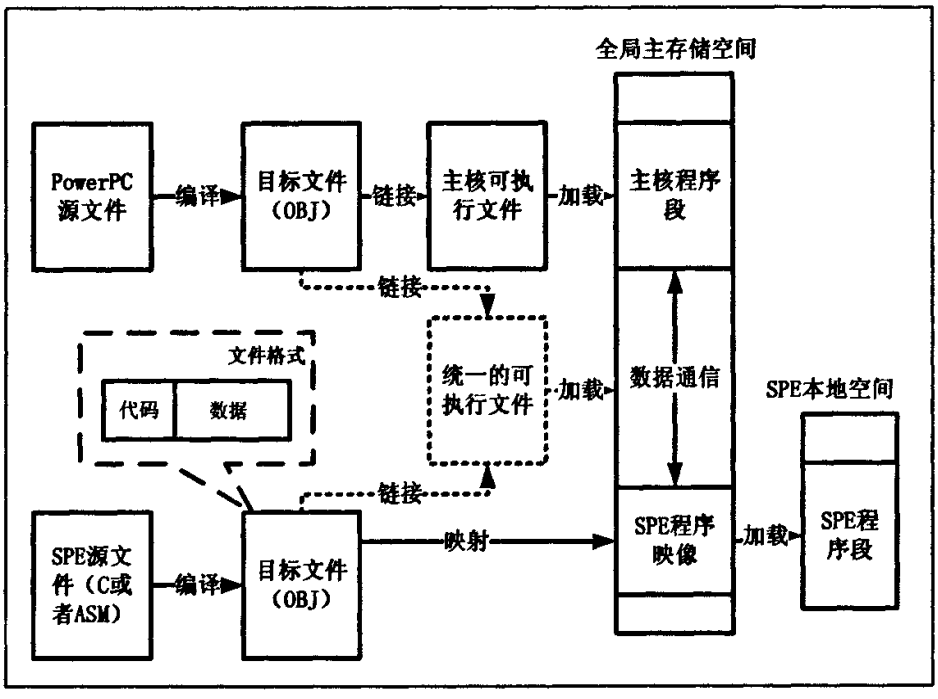

图 5-2 SPE 程序的编程模型[51]

运行在主核上的应用程序通过一段为SPE设计的加载程序把SPE目标文件加载到内存，并通过主核同 SPE 的通讯接口把 SPE 进程上下文发送到 SPE 上进行执行。详细的执行过程如图 5-3 所示，应用程序在主核上启动，接着加载 SPE 的目标文件到内存空间，组织成一个能在 SPE 上运行的 SPE 任务，然后通过操作系统内控制 SPE 的驱动程序模块启动并初始化 SPE，并把 SPE 任务发送到 SPE，等待 SPE 执行完毕后整理 SPE 的计算结果。在辅助核 SPE 上，其 SPE 任务的运行过程与主核对其的控制过程同步，分成四个步骤，分别是启动或者初始化 SPE，通过数据传输总线从主存读取 SPE 任务并执行，最后把执行的数据结果同步回主存。

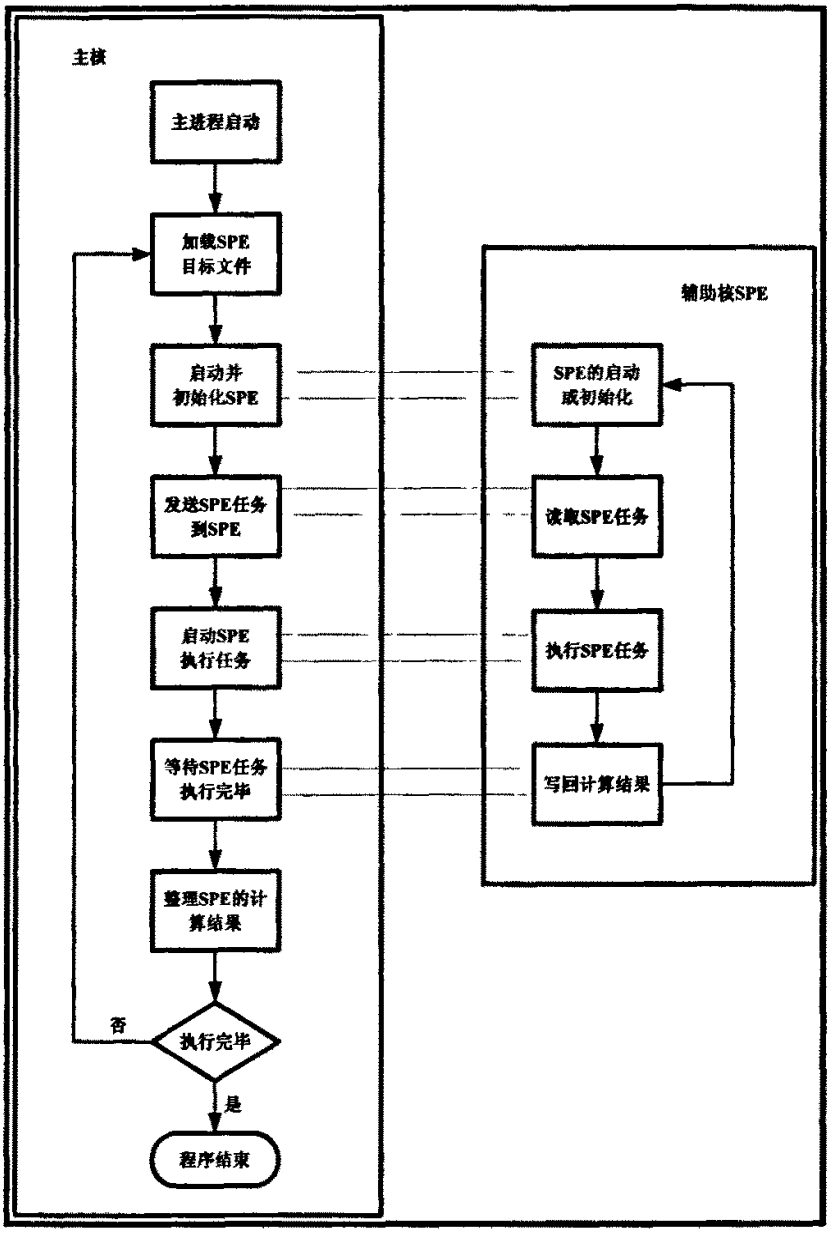

图 5-3 应用程序的执行流程

5.4 本章小结

本章介绍了操作系统对软硬件协同任务调度的支持，并介绍了运行在 SPE 上的 SPE 任务的上下文的组织形式，以及支持软硬件协同任务调度的编程模型。

第6章 SoC 的实现和验证

6.1 系统硬件平台

异构多核体系以及软硬件协同任务调度的 SoC 实现平台是基于 Virtex-4 技术的 Xilinx ML403 开发板。ML403 开发板是 Xilinx 公司专门为嵌入式 SoC 开发设计的集硬件，软件于一体的解决方案。图 6-1 就是 ML403 开发板，图中间的白色芯片是型号为 XC4VFX12-FF668-10C 的 FPGA 芯片，芯片内共计有 5472 个 Slice。同时在 FPGA 芯片上还内置了一个 PowerPC405 的硬核。另外，开发板上集成了非常多的用于构建嵌入式系统的常见的外围设备。包括存储器系统，64M 的 DDR SRAM，512M 的 CF 卡，以及 LCD 接口，RS-232 串口，PS/2 接口，AC97 接口，以太网接口，USB 接口，JTAG 接口，GPIO 接口等等[49]。

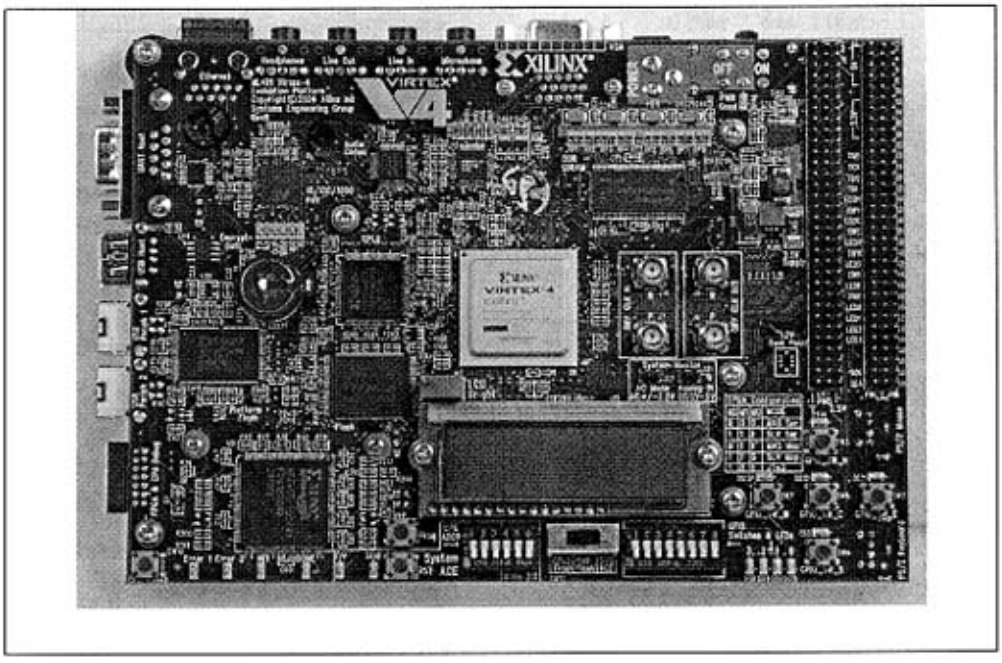

图 6-1 Xilinx Virtex-4 ML403 FPGA 开发板外观[50]

另外，在开发工具上，Xilinx 公司提供了一整套用于嵌入式系统的设计和开发的 EDA 工具以及各种 IP 核。EDK（Embedded system Development Kit）工具是嵌入式系统设计和整合的综合开发集成环境。

在 EDK 的帮助下，本文搭建一个以 PowerPC405 为核心的，整合 4 个自主开

发的辅助核 SPE 的多核心的体系机构。同时设计一个硬件调度器 IP 核来管理这 4 个辅助核 SPE。图 6-2 是 SoC 系统的设计图，PowerPC 硬核，4 个 SPE 以及硬件调度器和内存子系统，LCD 控制器都连接在 PLB 总线上，而其他对速度要求不是很高的外围设备控制器均连接在 OPB 总线上。

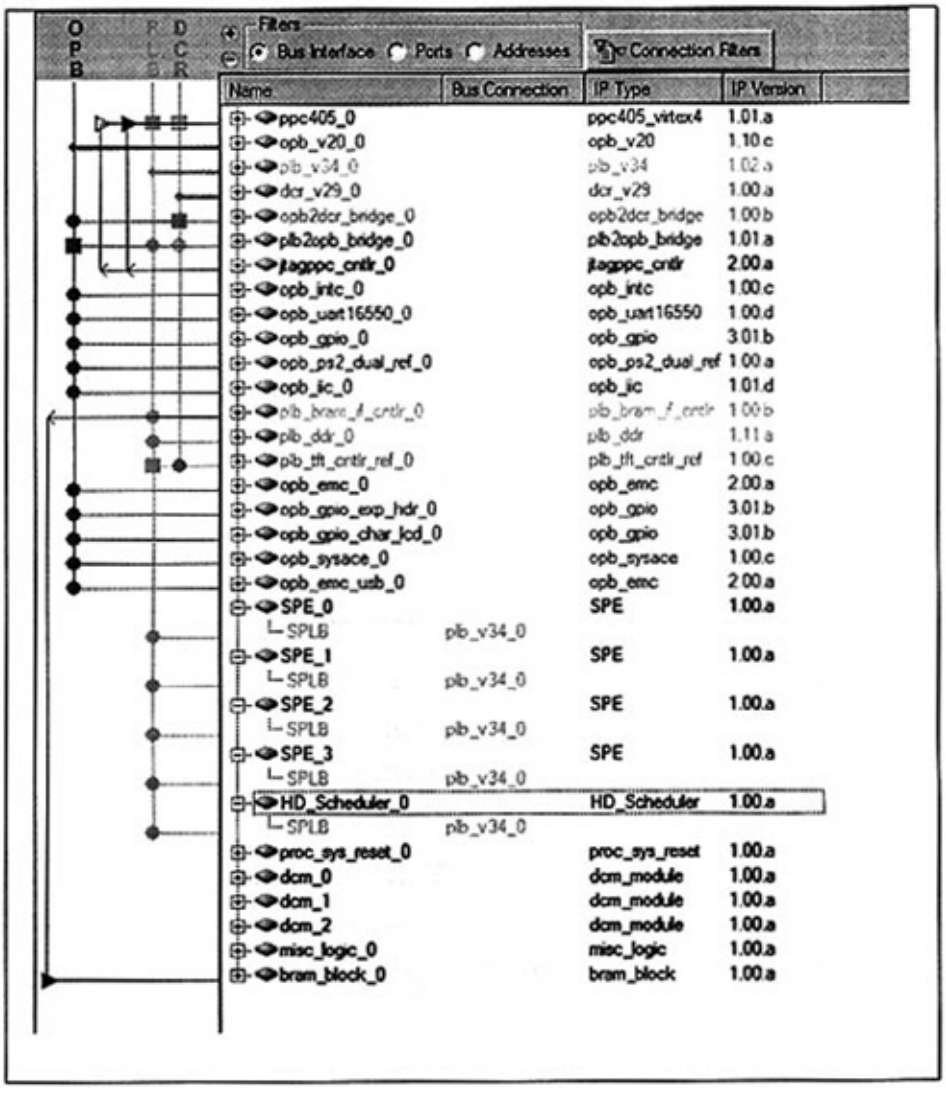

图 6-2 SoC 系统在 EDK 中的整体设计图

在图 6-2 所示构建了一个硬件体系结构之后，本文需要把内存子系统，硬件调度器，辅助核 IP 核以及全部的外设 I/O 控制器都总线的可视范围内建立一个统

一编址的地址空间。表格 6-1 就是本文 SoC 系统的内存映射分配表。可以看到硬件调度器分配了低地址为 0xCF600000，高地址为 0xCF60FFFF 的地址空间。4 个 SPE 也分别分配了 4 段不同的地址空间。

表 6-1 SoC 系统的内存映射分配表

模块名	低地址	高地址
plb_tft_cntlr_ref_0	0B1011000000	0B1011001111
plb_ddr_0	0B11111111111111111111111111111111	0B00000000000000000000000000000000
plb_ddr_0	0x00000000	0x0FFFFFFF
plb_ddr_0	0B11111111111111111111111111111111	0B00000000000000000000000000000000
plb_ddr_0	0B11111111111111111111111111111111	0B00000000000000000000000000000000
plb_ddr_0	0B11111111111111111111111111111111	0B00000000000000000000000000000000
opb_emc_usb_0	0x28FFFF00	0x28FFFFFF
opb_emc_0	0x29700000	0x297FFFFF
opb_emc_0	0x29800000	0x29FFFFFF
opb_gpio_exp_hdr_0	0x40000000	0x4000FFFF
opb_gpio_char_lcd_0	0x40020000	0x4002FFFF
opb_gpio_0	0x40040000	0x4004FFFF
opb_uart16550_0	0x40400000	0x4040FFFF
opb_iic_0	0x40800000	0x4080FFFF
opb_intc_0	0x41200000	0x4120FFFF
opb_sysace_0	0x41800000	0x4180FFFF
opb_ps2_dual_ref_0	0x7A400000	0x7A40FFFF
SPE_3	0xCDA00000	0xCDA0FFFF
SPE_2	0xCDA20000	0xCDA2FFFF
SPE_1	0xCDA40000	0xCDA4FFFF
SPE_0	0xCDA60000	0xCDA6FFFF
HD_Scheduler_0	0xCF600000	0xCF60FFFF

最后，通过 ISE8.1 和 EDK8.1 对整个 SoC 系统设计进行综合，生成能下载到 FPGA 开发板上的 bit 硬件配置文件。

6.2 系统软件环境

基于ML403开发板，Xilinx公司提供了Linux，QNX等操作系统软件支持的demo，当然Xilinux公司并没有提供这些软件系统的源代码。为了有一个灵活而有效的支持本文的SoC的软件系统，本文采用开源的Linux系统，针对SoC系统进行修改定制以支持本文的系统。

在源代码来源上，Linux内核采用由Paul Mackerras维护的Linux 2.4 for PowerPC，该代码可以从http://ppc.bkbits.net/ 获得。

在开发环境上，通过使用kegel提供的crosstool工具定制了一套面向PowerPC体系结构的交叉编译环境。

在内核的定制过程中，首先把由EDK根据SoC硬件设计的平台生成的BSP（Board Support Package）添加到内核中，同时在内核配置的选项中，选择上让内核支持ML403开发板上Xilinx on-chip System ACE接口的选项以及各类硬件平台需要支持的选项。

在对SPE以及硬件调度器的支持上，通过向内核添加控制硬件调度器的模块以及添加一个处理硬件调度器中断的中断处理函数来支持系统的SoC硬件系统。

一个操作系统内核总是要运行在一个根文件系统之上，在这个根文件系统之上，有一些系统启动登录所必须的配置文件以及一些系统常用的工具。创建根文件系统大致需要一下一些步骤，首先是建立一些必须的文件目录，比如/bin，/dev，/etc等文件目录，装载glibc，/etc文件下的sample配置文件，以及一些常见的工具程序（这些工具可以崇一个叫做Busybox的程序包中获得。在本文的项目中，本文使用了ML403开发板用于演示MontaVista demo所用的文件系统测试与验证

为了能够在硬件设计下载到FPGA后，软件系统能够使用FPGA硬件系统，Xilinx设计了一套整合FPGA硬件配置文件和软件ELF可执行文件的管理系统，即SystemACE系统，该系统把硬件bit文件和软件elf文件整合成一个ace文件，SystemACE系统从CF卡里读取ace文件，配置硬件FPGA芯片，同时启动整合在ace文件的的elf程序。在本文的应用中，把硬件体系和Linux内核ELF文件整合成ACE文件来使用，从而实现硬件系统的配置和Linux内核的启动。在本文的项目中，通过划分CF卡的一个小分区来存放最终的ace文件，而CF剩余的空间格式化成一个ext2的文件系统分区用来当作操作系统的运行的文件系统。

6.3 测试和验证

为了测试整个SoC系统，尤其是操作系统调度器和硬件调度器协同分层调度系统中的不同类型的任务的模型。本文分别采用了了几组不同的程序的组合情况来对系统进行验证和测试。

表 6-2 用于测试应用程序列表

应用程序	属性
P1	4个SPE任务，高优先级
P2	4个SPE任务，普通优先级
P3	6个SPE任务，高优先级
P4	6个SPE任务，普通优先级
P5	100个SPE任务，高优先级
P6	100个SPE任务，普通优先级

设计用于测试的应用程序过程中，本文把应用程序按两个属性分别进行分类。一种分类方法是把应用程序分成SPE任务多于物理辅助核数和不超过物理辅助核数。另一种分类方法是把应用程序分成普通优先级和高优先级的。详细的应用程序列表如表6-2所示。从表中可以看到，P1，P2是包含的SPE任务数不超过物理辅助核SPE个数的两个应用程序，但是两个应用程序的优先级不同。而所有SPE任务都是一段相同的用于计算字符串相似度的SPE程序目标代码。

在6-3中，本文用表6-2定义的6个测试程序共组合出九组测试程序组合。首先是有着不同SPE任务个数的应用程序分别进行单独测试。接着，第四，第五以及第六组则分别用两个相同的应用程序同时运行的方式，测试了整个SoC尤其软硬件协同任务调度系统工作状况。最后三组通过组合相同程序但不同优先级运行的方式两两运行测试。为了进行测试比较，本文通过传统的编程人员手工调度SPE任务的方式和本文设计的硬件调度器进行调度的方式进行比较。在传统的调度中，当有多个应用程序需要使用辅助核的时候，通过使用互斥锁进行管理。

表 6-3 测试分组和测试结果汇总

分组	应用程序	运行时间（秒）传统调度	总计	运行时间（秒）硬件调度	总计
第一组	P2	10.2	10.2	10.1	10.1
第二组	P4	13.5	13.5	13.4	13.4
第三组	P6	211.6	211.6	210.1	210.1
第四组	P2	22.3	22.3	20.8	20.4
	P2	21.8		19.8	
第五组	P4	28.3	29.8	25.3	26.1
	P4	29.8		26.1	
第六组	P6	219.7	219.7	208.2	208.2
	P6	217.5		201.7	
第七组	P1	17.6	25.1	12.1	27.2
	P2	25.1		27.2	
第八组	P3	21.3	31.9	18.6	32.6
	P4	31.9		32.6	
第九组	P5	193.9	289.6	165.2	264.5
	P6	289.6		264.5	

首先是三类应用程序分别在传统调度和硬件调度的模式下运行时间的比较，运行结果如表 6-4 所示，分析可知，在系统中只有单个应用程序的时候，通过传统的编程人员手工调度和使用硬件调度器进行调度在效果上不相上下，如图 6-3 所示。

表 6-4 单个应用程序基于传统调度和硬件调度运行效果比较

应用程序	运行时间（秒）传统调度	运行时间（秒）硬件调度器
P2	10.2	10.1
P4	13.5	13.4
P6	211.6	210.1

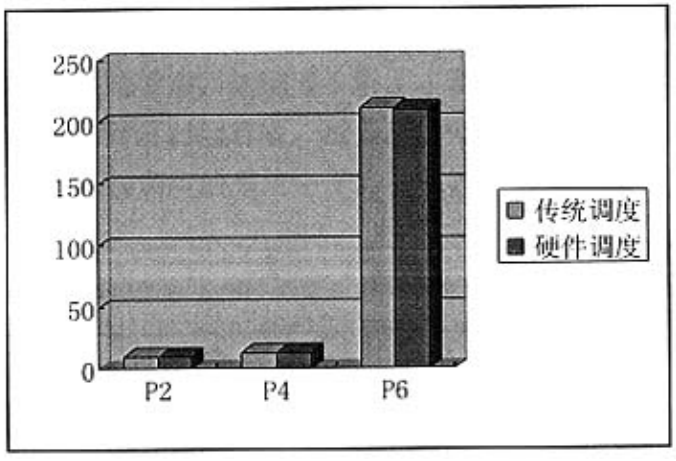

图 6-3 单个应用程序基于传统调度和硬件调度运行效果比较

应用程序P1，P3，P5分别是P2，P4，P6的高优先级程序。普通优先级和高优先级下的同一个程序同时在系统中运行，测试结果如表所示，运行结果为高优先级应用程序的所用时间。

表 6-5 高优先级应用程序基于传统调度和硬件调度的运行时间比较

应用程序	P1+P2	P3+P4	P5+P6
传统调度	17.6	21.3	193.9
硬件调度	12.1	18.6	165.2

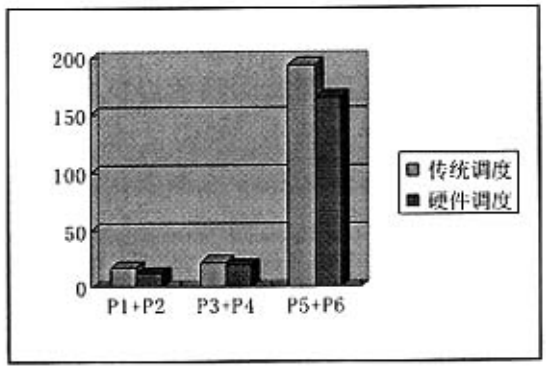

图 6-4 高优先级应用程序基于传统调度和硬件调度的运行时间比较

对表6-5的结果分析可知，采用硬件调度的系统中高优先级的应用程序以较短的时间运行完成，硬件调度系统对应用程序实时性运行上比传统系统有更好的优势。如图6-4所示。

在第四到六组则分别采用两个应用程序同时在系统中进行测试。表6-6是测试运行结果。分析可知，采用硬件调度器的系统应用程序整体的运行时间都比采用传统调度方式运行的系统要短，如图6-5所示。系统在硬件调度器的支持下，同样的计算任务，使用的计算时间较短，任务的吞吐量较大。应用程序运行时间在硬件调度的系统中比传统调度的系统中平均快8.1%。

表6-6 两个应用程序基于传统调度和硬件调度同时运行的时间比较

应用程序	2xP2	2xP4	2xP6
传统调度	22.3	29.8	219.7
硬件调度	20.8	26.1	208.2

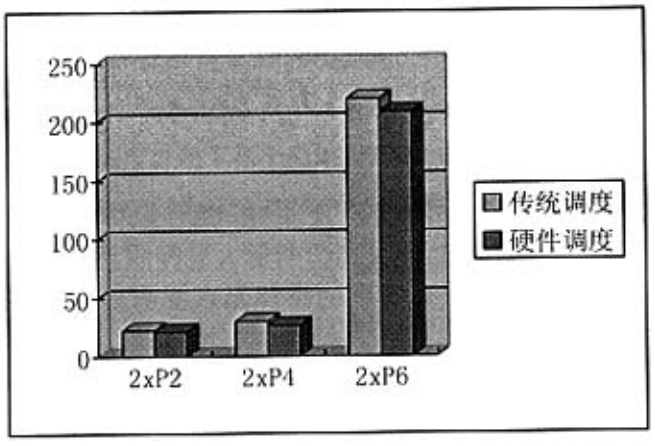

图6-5 两个应用程序基于传统调度和硬件调度同时运行的时间比较

在图6-5中，比较宏观的对比了两个应用程序同时运行的时候硬件调度和传统调度的调度性能。从图中，可以看到硬件调度器在调度性能上比较出色，为了更精确的度量系统的吞吐量，本文从另一个角度来对比不同调度方法下，系统对硬件资源的利用情况。所有应用程序的SPE任务都是相同的，所以通过对应用程序完成一个SPE任务的平均时间的长短可以对比同一程序在不同调度方法下单个SPE任务调度和运行的平均时间。图6-6就是对三个不同应用程序在传统调度和硬件调度两种方法下单个SPE任务的调度和运行时间，简称为SPE任务时间。由于各个SPE任务的运行时间都是相同的，所以硬件调度方法下运行时间的节省就任务调度过程中的性能提升。

表 6-7 不同调度方法下单个应用程序中 SPE 任务时间对比

应用程序	P2	P4	P6
传统调度下 SPE 任务时间（秒）	2.55	2.25	2.12
硬件调度下 SPE 任务时间（秒）	2.53	2.23	2.1

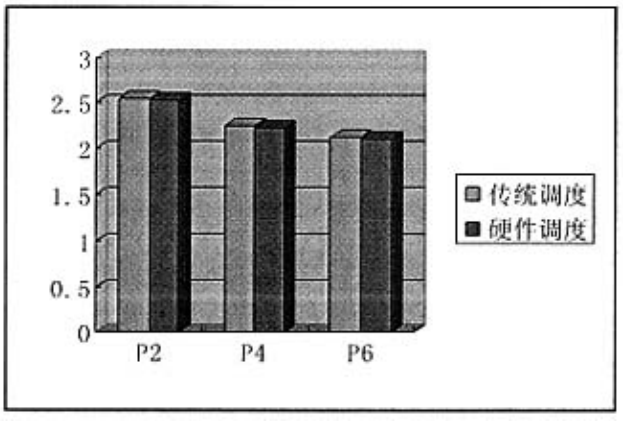

图 6-6 不同调度方法下单个应用程序中 SPE 任务时间对比

单个应用程序在系统中运行的 SPE 任务时间虽然在硬件调度方法比在传统方法中短一点，但并不是特别明显，硬件调度比传统调度大约可以节约 0.8%的时间。在图 6-7 里，分别就两种调度方法对不同个数应用程序同时运行情况下的 SPE 任务时间。

图 6-7（a）是对传统调度方法下的 SPE 任务时间对比，可以看到，由于应用程序数量的增加，程序间对辅助核的竞争导致 SPE 任务时间的显著增加。而图 6-7（b）可以看到基于硬件调度方法下的 SPE 任务时间虽然也随着应用程序增加而增加，但并不像传统调度方法下那么明显。平均在 SPE 任务时间上比传统调度方法节约 7.8%。

表 6-8 不同调度方法下不同数量应用程序 SPE 任务时间的对比

传统调度	P2	P4	P6		硬件调度	P2	P4	P6
单个应用程序	2.55	2.25	2.12		单个应用程序	2.53	2.23	2.1
两个应用程序	2.79	2.48	2.2		两个应用程序	2.55	2.32	2.18
三个应用程序	2.98	2.64	2.43		三个应用程序	2.65	2.37	2.32

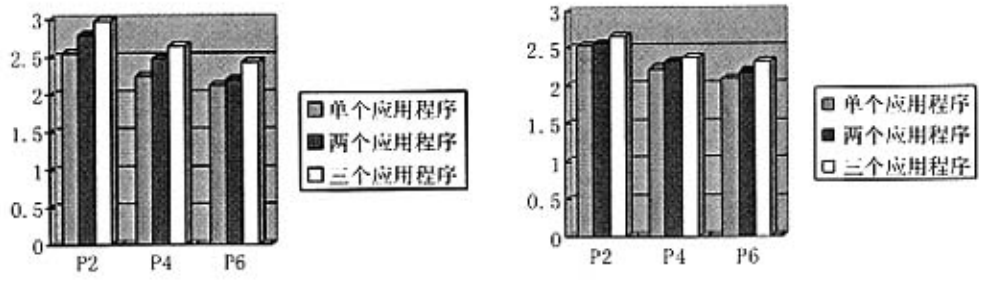

图 6-7 不同调度方法下不同数量应用程序 SPE 任务时间的对比

总的来说，传统操作系统调度系统和硬件调度系统结合的软硬件协同的调度模型在提高系统的计算吞吐量，以及简化应用程序开发难度等各个方面都有很好的效果。

6.4 本章小结

本章介绍了 SoC 系统实现的硬件平台以及软件平台。并介绍了系统的测试和验证。经测试分析表明，本文设计的软硬件协同任务调度的方法能有效的提高系统的计算吞吐量，简化基于异构多核应用程序的编程模型。

第7章 总结和展望

7.1 结论

本文论述了面向嵌入式异构多核体系的软硬件协同任务调度的方法。针对基于片上总线的异构多核嵌入式系统结构对操作系统支持以及软件编程模型的需求，本文从对异构多核体系结构的分析开始，设计了一种兼容现有操作系统和编程方式软硬件协同的任务调度方法。该方法能充分利用现有操作系统的任务管理功能以及多线程的编程方式，设计并实现了一个基于硬件 IP 核方式的硬件调度模块，负责辅助核任务在辅助核上调度管理工作。

通过在 SoC 平台上实现包含硬件调度器的 SoC 硬件系统，和支持此异构多核 SoC 系统的软件平台，本文实现一个辅助核任务由硬件调度器调度管理，主核任务由操作系统调度器进行管理的分层次的软硬件协同的任务调度体系。

具体的来说，在本文中主要在以下几个方面做了研究工作：

1) 分析了嵌入式异构多核体系的特性，并研究了针对异构多核体系的操作系统支持以及应用程序编程模型。
2) 设计了软硬件协同任务调度的模型，辅助核任务由硬件调度器进行调度管理，而主核上的应用程序则由操作系统的调度器来调度管理。
3) 设计并实现了基于 IP 核的硬件调度器，该调度器支持先来先服务和动态优先级两种调度策略。并定义了硬件调度器和辅助核之间的控制通讯接口。
4) 设计了辅助核任务的编程模型。并按照此编程模型编写测试应用程序对整个软硬件协同任务调度模型进行测试验证。

本文设计的软硬件协同的任务调度方法有效的降低了软件系统对嵌入式异构多核体系的支持难度，通过设计和使用硬件调度器，明显提高了辅助核任务的吞吐量。

7.2 展望

但是，由于时间，人力，设备等各个方面的因素限制，本文研究的软硬件协同的任务调度方法还有很多不完善的地方，还有很多可以进一步深入研究和探讨的地方。

1) 在本文中，设计的硬件调度器只支持一种类型的辅助核，按照集成电路和计算机体系结构的发展趋势，在嵌入式系统中，多核技术的应用将会

越来越普及，不仅仅在核的数量上会有显著的增加，而且在核的种类上也会变得很多很多。所以，在下一步的研究方向，就是要让硬件调度器支持多种类型的辅助核。

2) 本文的硬件调度器只支持先来先服务和动态优先级两种调度策略，随着核的数量以及核的种类的增加，必然要求硬件调度器在调度功能有更强大的功能。

3) 在本文中，主要是通过硬件调度去器的设计来协助传统操作系统的任务调度。不过由于硬件调度器的参与了任务调度的某些部分，在这种新的调度模型下，操作系统的调度策略如何优化等研究方向都需要进一步的探讨。

参考文献

[1] D. Geer, "Chip makers turn to multicore processors," IEEE Computer, vol. 38, no. 5, pp. 11-13, 2005

[2] Gepner, P., M.F. Kowalik. Multi-Core Processors: New Way to Achieve Hig h System Performance [C]. Proceedings of the International Symposium on Para llel Computing in Electrical Engineering (PARELEC'06). IEEE Computer Society. 2006: 9-13.

[3] Chiodo, M., et al. Hardware-software codesign of embedded systems [J]. IE EE Micro. 1994, 14(4): 26-36.

[4] Maruyama, M., et al. A 200 MIPS image signal multiprocessor on a single chip [C]. Proceedings of the Solid-State Circuits Conference (ISSCC 2005). Di gest of Technical Papers. IEEE, 1990: 122-123, 279.

[5] Maruyama, M., et al. An image signal multiprocessor on a single chip [J]. IEEE Journal of Solid-State Circuits. 1990, 25(6): 1476-1483.

[6] Seng Lin Shee,"Heterogeneous multiprocessor implementations for JPEG__a case study", International Conference on Hardware Software Codesign

[7] Ning Zhang, Chwan-Hwa Wu, Study on Adaptive Job Assignment for Multi processor Implementation of MPEG2 Video Encoding, IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, VOL. 44, NO. 5, OCTOBER 1997

[8] Mladen Berekovic′, Hans-Joachim Stolberg, and Peter Pirsch, A multicore sy stem-on-chip architecture for MPEG-4 streaming video, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 12, NO. 8, AUGUST 2002

[9] A. Beri′c, R. Sethuraman, C. A. Pinto, H. Peters, G. Veldman, P. van de H aar, and M. Duranton. Heterogeneous Multiprocessor for High Definition Video. In ICCE'06, pages 401 – 402, 2006.

[10]Bergmann, A., K. Hacker. Spufs: The Cell Synergistic Processing Unit as a virtual file system [EB/OL]. [http://www-128.ibm.com/developerworks /linux/library/pa-cell/?S_TACT=105AGX52&S_CMP=cn-a-l].

[11]Gordon McNutt and Todd Fischer. USING LINUX TO CONTROL DSP PR OCESSES IN MIXED-PROCESSOR SYSTEMS. Embedded Edge, 2001

[12]Curt Schimmel. UNIX Systems for Modern Architectures.Addison Wesley,20 03,4

[13]Tullsen D.M., Eggers S.J., Levy H.M. Simultaneous Multithreading: maximiz ing on-chip parallelism. In:Proc. of 22nd Annual International Symposium on C omputer Architecture, Santa Margherita Ligure, Italy,1995. 392-403

[14]Hirata, H., Kimura, K., Nagamine, S., Mochizuki, Y., Nishimura, A. et al. An Elementary processor architecture with simultaneous instruction issuing from multiple threads. In: Proc. of the 19th International Symposium on Computer Architecture, 1992. 136-145

[15]Microprocessor Report. October 24, 1994

[16]Guttag, K., R.J. Gove, J.R. Van Aken A single-chip multiprocessor for mult imedia: the MVP. Computer Graphics and Applications, IEEE. 1992, 12(6): 53-6 4.

[17]Eggers S.J., Emer J.S., Levy H.M., Lo J.L., Stamm R.L. and Tullsen D.M. Simultaneous multithreading: aplatform for next-generation processors. IEEE Mi cro, 1997, 17(5):12-19

[18]Agerwala, T., S. Chatterjee. Computer architecture: challenges and opportunit ies for the next decade [J]. IEEE Micro, 2005. 25(3): p. 58-69.

[19]Marr D.T., Binns F., Hill D.L., Hinton G., Koufaty D.A., Miller J.A., and Upton M. Hyperthreading technology architecture and microarchitecture: a hypert ext history. Intel Technology J. 2002, 6, 1

[20]S.Hily and A.Seznec. Branch prediction and simultaneous multithreading. In: Proc. of International Conference on Parallel Computer Architecture and Compi lation Technology, 1996

[21]J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Appro ach, 2nd ed., Morgan-Kaufman, San Mateo, Calif., 1996.

[22]B. Catanzaro, Multiprocessor System Architectures, Prentice Hall, Englewood Cliffs, N.J., 1994.

[23]An architecture for high-performance scalable shared-memory multiprocessors exploiting on-chip integration, Acacio, M.E.; Gonzalez, J.; Garcia, J.M.; Duato, J.Parallel and Distributed Systems, IEEE Transactions on On page(s): 755- 768, Volume: 15, Issue: 8, Aug. 2004

[24]Olukotun, K., et al. The case for a single-chip multiprocessor [J]. ACM SI GPLAN Notices. 1996, 31(9): 2-11.

[25] Intel. Intel® Core™2 Duo processor. http://www.intel.com/products/processor_number/eng/chart/core2duo.htm

[26]POWER4 system microarchitecture. http://www.research.ibm.com/journal/rd/461/tendler.html

[27]Keltcher, C.N. McGrath, K.J. Ahmed, A. Conway, P. The AMD Opteron p rocessor for multiprocessor servers IEEE Micro Publication Date: March-April 2003 Volume: 23, Issue: 2 On page(s): 66- 76

[28]K. Normoyle et al., "The UltraSPARC Port Architecture," Proc. Hot Interco nnects Symp. III, 1995, available from author at kevin.normoyle@eng.sun.com.

[29]Pham, D.C., et al. Overview of the architecture, circuit design, and physic al implementation of a first-generation cell processor [J]. IEEE Journal of Solid-State Circuits. 2006. 41(1): 179-196.

[30]IBM Corporation, Cell Broadband Engine Programming Handbook, <http://w ww-306.ibm.com/chips/techlib/techlib.nsf/products/Cell_Broadband_Engine>,2006.

[31]IBM Corporation, Cell Broadband Engine Registers, <http://www-306.ibm.co m/chips/techlib/techlib.nsf/products/Cell_Broadband_Engine>,2006.

[32]IBM Corporation, Cell Broadband Engine Hardware Initialization Guide, <ht tp://www-306.ibm.com/chips/techlib/techlib.nsf/products/Cell_Broadband_Engine>,20 06.

[33]Tummala, R.R. Madisetti, V.K. Georgia Inst. of Technol., Atlanta, GA; Sys tem on chip or system on package?IEEE Design & Test of Computers Publicati on Date: Apr-Jun 1999 Volume: 16, Issue: 2 On page(s): 48-56

[34]Sarkar, S., S. Chanclar G, S. Shinde. Effective IP reuse for high quality SO C design [C]. Proceedings of IEEE International SOC Conference, 2005. 2005: 217-224

[35] Bocchi, M., et al. A system level IP integration methodology for fast SOC design [C]. Proceedings of International Symposium on System-on-Chip, 2003. 2003: 127-130.

[36]Cesario, W.O., et al. Multiprocessor SoC platforms: a component-based desi gn approach [J]. IEEE Design and Test of Computers. 2002, 19(6): 52-63.

[37] Daewook, K., K. Manho, G.E. Sobelman. DCOS: cache embedded switch a rchitecture for distributed shared memory multiprocessor SoCs [C]. Proceedings of 2006 IEEE International Symposium on Circuits and Systems (ISCAS 2006). 2006: 979-982

[38] 杨刚，肖宇彪，陈江等.32 位嵌入式系统与 SoC 设计导论。电子工业出版社，2006 年 4 月，书号，7-121-01316-9。

[39]Qnx_multicore_white_paper http://www.qnx.com/download/feature.html?progra mid=12178

[40]Linux howto. http://tldp.org/HOWTO/SMP-HOWTO.html

[41]周卫华，吴永明，高珍。 Linux 操作系统内核对 SMP (对称多处理器) 的支

持。计算机应用研究， 2002。

[42]IBM Corporation, CoreConnect Bus Architecture, <http://www-306.ibm.com/chips/techlib/techlib.nsf/products/CoreConnect_Bus_Architecture>, 1999.

[43]IBM Corporation, On-Chip Peripheral Bus, http://www-306.ibm.com/chips/techlib/techlib.nsf/products/CoreConnect_Bus_Architecture, 2001.

[44] IBM Corporation, Processor Local Bus (128-bit), http://www-306.ibm.com/chips/techlib/techlib.nsf/products/CoreConnect_Bus_Architecture, 2004.

[45] IBM Corporation, Device Control Register Bus 3.5 Architecture Specifications, http://www-306.ibm.com/chips/techlib/techlib.nsf/products/CoreConnect_Bus_Architecture, 2006.

[46]Peleg, A., Weiser, U. MMX technology extension to the Intel architecture [J]. Micro, IEEE. 1996, 16(4): 42-50.

[47]Raman, S.K., Pentkovski, V., Keshava, J. Implementing streaming SIMD extensions on the Pentium III processor [J]. Micro, IEEE. 2000, 20(4): 47-57

[48]Andrew S.Tanenbaum;Albert S.Woodhull. Operating Systems: Design and Implementation （Second Edition） Prentice Hall

[49]http://china.xilinx.com/products/boards/ml403/reference_designs.htm

[50]http://www.xilinx.com/bvdocs/images/ipcenter/product_images/HW-V4-ML403_1.jpg

[51]戴鸿君。基于异构多核与组件化软件的嵌入式系统研究[D]。 杭州，浙江大学，2007。

[52]Chen Tianzhou, Dai Hongjun. A Software Platform for Component-based Communication Protocols in the Embedded System [J]. Journal of Ubiquitous Computing and Intelligence.

[53]Tianzhou Chen, Guobing Chen, Hongjun Dai, Qinsong Shi. A function-based on-chip communication design on the heterogeneous multi-core architecture [C]. Proceedings of 2007 International Conference on Multimedia and Ubiquitous Engineering (MUE 2007).

[54]Yu Yonggang, Chen Tianzhou, Dai Hongjun. Design and Analyze the Communication in the Multi-Core Soc Driven by Petri Net [C]. Proceedings of Third International Conference on Autonomic and Autonomous Systems (ICAS2007).

攻读硕士学位期间主要的研究成果

软件著作权

2006SR02119　面向 Xscale 微架构的应用程序安装向导软件 V1.0
2006SR02267　面向局域网通讯的 ICU 可视电话系统软件 V1.0
2006SR01854　基于 Xscale 微架构的音频采集测试程序软件 V1.0
2006SR02266　面向嵌入式设备的高性能 MPEG-4 编解码库软件 v1.0

专利申请

200710066924.1　面向异构多核体系的进程调度方法
200710066931.1　面向异构多核体系的段页式存储空间管理方法
200710066932.6　面向异构多核体系的分段式存储空间管理方法
200710066933.0　面向异构多核体系的分页式存储空间管理方法

获得奖项

首届“二六三”杯浙江大学软件设计大赛“二等奖”，2005.9
首届“二六三”杯浙江大学软件设计大赛“最佳用户体验”（第一名），2005.9
"挑战杯"飞利浦科技多米诺大赛“团体优秀奖”，2005.9

致谢

研究生生涯即将结束，在浙江大学六年的求学之路也将画上了句号。在浙江大学计算机学院，特别是在浙江大学-Intel 技术中心学习和研究过程中，得到了许多老师和同学的帮助，在此向他们表示以下感谢。

首先要感谢的是我的导师陈天洲教授。从 2004 年进入技术中心以来，陈老师对我的学习，科研以及生活上都给予了很大的指导和帮助。陈老师就像大海里的灯塔，黑夜里的明灯，照耀着我前进的方向。从陈老师那里我学到了如何脚踏实地的做学问，也学到了如何为人处世。我还要感谢浙江大学计算机学院的施青松老师、施敏华老师、李莹老师和张勤老师。多年来，他们在学习上、科研上、生活上也都给予了我很大的帮助，让我学习到了很多有用的知识。

我还要感谢实验室中我的组长戴鸿君博士，以及在实验室或者曾经在实验室工作学习过的陈国兵、严力科、田晓帆、陈学亮、叶敏娇、蒋宁、贺臻杰、黄彧、姚磊、胡威、黄江伟、谢斌、赵懿、吴心亮、梁晓、沙峰、钱杰、王祥生、马吉军等等，他们对我的研究课题提供了很多帮助和指导，帮我解决了很多困难和问题。

我还要特别感谢我的父母，是他们含辛茹苦的把我养大，对我的学习教育培训付出很多的心血，即使在大学求学过程中，他们依然在远方默默支持着我。我还要感谢我的妹妹，她也一直支持这我的求学，我在外求学不能陪伴父母的时候，都是她照顾陪伴着父母。

还有很多老师、同学和朋友都对我的学习和成长给予了很大的帮助，在此无法一一列举。谢谢他们的关心和帮助，我会以更加努力的工作、更加真诚的待人，来回报他们。谢谢。

黄振宝

2007 年 5 月 10 日