ABSTRACT

Interrupt system is an important component of modern computer system, for failure handling, real-time control, software debug, communication between processors and I/O devices all depend on it. With the development of computer software and hardware, the concept of interrupt is expending unceasingly, and the functional requirement of interrupt system in modern processor is also changing constantly. Interrupt system must go along with the development of processor architecture in order to promote the performance, especially when processors has entered the era of multi-core, multi-thread and virtualization.

Base on virtual interrupt system in multi-core and multi-thread architecture, the aim of this study is to design and implement an interrupt system with virtualization features which supports multi-core architecture and multi-thread shared in X processor. As a part of the project on "High-Performance X Processor", the research is applied to practices directly.

The paper researches the basic theory of virtualized interrupt system in multi-core and multi-thread architecture. It discusses the requirement of such an interrupt system, and sums up the possible solutions. Based on above research, it proposes the software interface and interrupt model of virtualized interrupt system, which is suitable for X processor with multi-core and multi-thread architecture. It also illustrates how to design the system with the use of a special floating point exception prediction mechanism to resolve the precise interrupt problem, and a synchronous instruction mechanism to reduce hardware costs and to cut down complexity of components interconnection. A verification scheme is raised for the interrupt component of X processor, following implementation and synthesizes. The results of the experiment show that the function of design is right and the performance meets the requirements of X processor.

Key Words: Multi-core and multi-thread, Virtualized, Interruption System, Precise Interruption, External Interruption

表 目 录

图 目 录

独 创 性 声 明

本人声明所呈交的学位论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表和撰写过的研究成果，也不包含为获得国防科学技术大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文题目：多核多线程虚拟化中断系统的研究与实现

学位论文作者签名：祝帅君　　日期：2008 年 12 月 30日

学位论文版权使用授权书

本人完全了解国防科学技术大学有关保留、使用学位论文的规定。本人授权国防科学技术大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档，允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密学位论文在解密后适用本授权书。）

学位论文题目：多核多线程虚拟化中断系统的研究与实现

学位论文作者签名：祝帅君　　日期：2008 年 12 月 30日

作者指导教师签名：张民选　　日期：2008 年 12 月 30 日

第一章 绪论

随着处理器设计制造技术的不断发展，处理器的体系结构发生了深刻变化，具有高并行性的多核多线程技术成为当前提升处理器性能的主流；处理器性能的提高，又使得应用领域的虚拟化技术有了更宽广的发展空间，近年来不断改进，并逐渐被引入到处理器的设计当中。在处理器设计领域，多核多线程技术和硬件辅助的虚拟化技术，已经成为当前人们研究的热点。

1.1 研究背景

1.1.1 微处理器体系结构的发展

当前，依靠指令级并行技术（ILP：Instruction Level Parallelism）来提升处理器性能的方法早已遇到了瓶颈。另外，在应用方面随着在线数据库事务处理、Web服务处理等应用的发展，许多领域特别是商业数据处理领域也需要计算机系统支持更大的吞吐量计算，具备任务级并行处理能力。

为了提高处理器的并行性和性能，适应新的应用需求，人们提出了线程级并行（TLP：Thread Level Parallelism）的概念，并在此基础上发展出了两种线程级并行技术，分别是单片多处理器（CMP：Chip Multiprocessor）[1,2]技术和多线程（MT：Multi-threading）[3] 技术。单片多处理器技术的基本思想是在同一芯片内集成多个简单的处理器内核，多个任务在不同的内核上并行执行，从而提高系统性能；而多线程技术则是在一个处理器中提供对多个线程额外的硬件支持，以便处理器能够在遇到长延迟事件时快速的切换线程上下文，从而有效避免等待时间的浪费[3]。

近年来，微处理器体系结构又有了新的发展。最新的多核多线程结构[4, 5] 处理器建立在上述两种线程级并行技术的基础上，它们具备吞吐量计算的能力，并具有高度的并行处理能力。

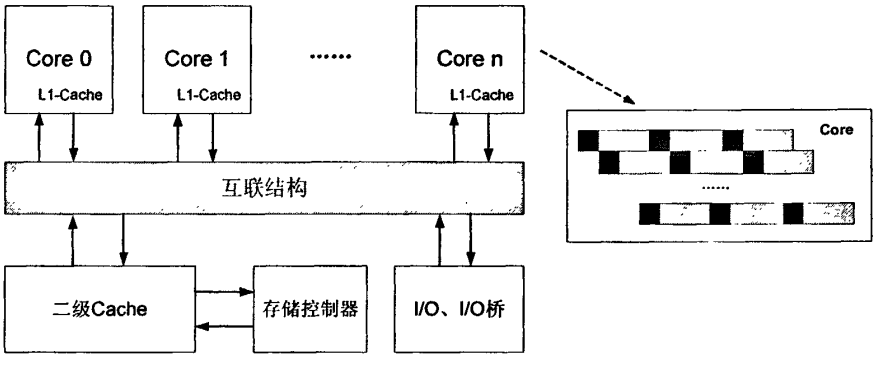


图 1.1 多核多线程处理器的典型结构

典型的多核多线程处理器结构如图 1.1 所示。它在一个片内集成多个多线程内核，并且还引入了 SOC 技术来构成一个以 CMP 为主体的片上系统。处理器的每个内核都采用了同时多线程或者硬件多线程等线程级并行技术。处理器核为每个线程维护了独立的运行状态，包括独立的寄存器文件、控制/状态寄存器和程序计数器等，线程可以作为一个完整的单元通过调度占用流水线，并独立地运行任务。此时，多核多线程处理器中的每个线程在逻辑上都可以看成是一个独立的 CPU，一般我们称为虚拟 CPU（VCPU，Virtual CPU）。

多核多线程结构可以提供一个密集的、高吞吐量的处理平台[6]，与传统处理器结构相比，它具备很大的优势：首先，它实现了真正的并发处理。通过在物理上包含多个处理器内核，每个内核都至少包含一套执行部件，它可以在不借助复杂硬件逻辑或者编译器软件的情况下真正并行执行多条指令。其次，多核多线程的流水线利用率非常高。多线程的硬件支持和快速切换线程上下文的能力使得处理器在运行的过程中，一旦某一线程被阻塞，处理器核中的执行流水线就会立即切换到另一个已经就绪的线程上执行，如图 1.2 所示[7]。

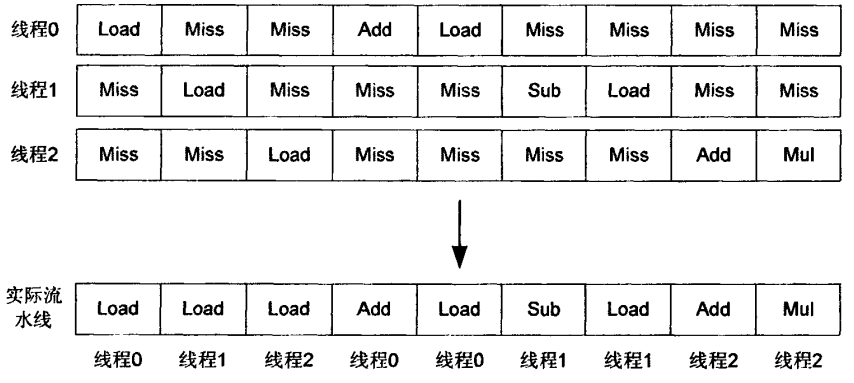


图 1.2 多个线程快速切换上下文以提高流水线利用率[7]

多核多线程处理器支持的是吞吐量计算[8]。它的设计目标并不是提高单个线程的执行速度，而是通过多核技术和多线程技术来提高系统整体处理能力。实际上，对于一个单任务程序来说，它在多核多线程处理器上运行时的效率可能不如在传统单处理器上运行时那么好：一方面，多核多线程处理器内核中的 Cache、执行流水线等资源是被多个线程共享的，单任务程序在多线程内核上运行时往往不能独占资源。另一方面，为了降低设计复杂度，多核多线程处理器内核的设计往往都比较简单，其发射部件的功能和执行流水线的深度都比不上传统单线程处理器。因此单线程在多核多线程处理器运行时其执行速度反而会变慢。从本质上来看，多核多线程处理器是通过牺牲部分单个线程的执行性能，利用在多个核内并发执行多个线程和快速切换线程现场以获得较高整体吞吐量的。因此，多核多线程处理器正好适合运行网络服务、数据处理这些领域中那些具备多任务特点的应用。

1.1.2 虚拟化技术的发展与处理器的虚拟化设计

随着计算机系统的广泛应用和处理器性能的不断提高，人们对计算资源的需求不断增加和总体计算资源过剩的矛盾日益尖锐。这一矛盾还带来许多新的问题：人们大量配置计算机设备，导致购买、维护等各项成本不断升高；设备的种类和数量繁多，管理和维护十分困难；设备升级导致原有应用需要修改和移植和等等。除此之外，人们对软件功能的需求也越来越高。在软件技术和网络技术日益发达的今天，我们需要运行应用的基础软件平台具有更高的安全性和便利性：一方面，可以使运行在其上的其它应用不会因为某个应用软件的崩溃而无法运行；另一面，人们可以通过网络技术很方便的传输和处理自己的事务。这些问题和需求，使得在近年来虚拟化技术再度发展，并成为人们研究的热点。

在计算机领域，虚拟化的含义十分广泛。一般来说，虚拟化是指在对计算机软硬件抽象化的基础上，通过创建某种计算资源的虚拟版本，以实现对计算机用户（终端用户、应用程序，或者操作系统）隐藏计算资源实际的物理特性[9,10]（如图 1.3）。通过虚拟化，可以使得多个操作系统及其应用程序同时运行在同一个硬件平台的多个虚拟机（VM）上，在提高计算资源利用率的同时，降低了设备管理和维护的难度；由虚拟化管理软件（VMM）负责管理所有的硬件资源，并提供虚拟机之间的安全隔离机制，使得各 VM 中所运行的操作系统和应用程序的安全性都大大提高，虚拟化平台中的某个 VM 的应用发生崩溃时，不会影响到其它的 VM 的运行。虚拟化的上述优势使得各大软件供应商和处理器厂商分别提出各种虚拟化技术，并将虚拟化特性引入到各自的软件或硬件产品中。

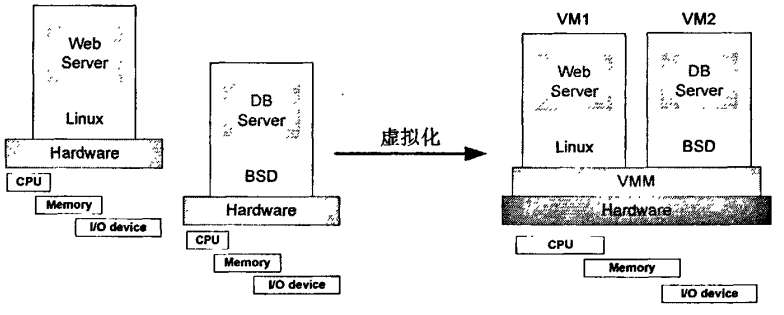


图 1.3 虚拟化应用示意图

VMM 需要向各 VM 中的客户操作系统（Guest OS）提供与真实硬件平台相同的硬件接口，这是实现系统虚拟化的基本要求。目前，根据 Guest OS 和 VMM 之间的接口关系，软件实现的虚拟化技术可以分为全虚拟化、半虚拟化以两类：

（1）全虚拟化技术[11,12,13]。全虚拟化技术是第一代系统虚拟化技术。在全虚拟化技术中，对于用户级别的指令，它采用在真实硬件平台上直接执行的方式以提高性能；对于 Guest OS 中的特权指令，则一般使用二进制翻译的方法，以一段

可以直接运行在真实硬件上的代码代替这些指令，实现虚拟化。

（2）半虚拟化技术[14,15,16]。半虚拟化和硬件辅助的虚拟化技术都属于第二代虚拟化技术，半虚拟化技术中，应用程序的代码仍然采用直接执行的方式运行，但是Guest OS中的特权指令不是采用翻译技术执行，而是通过在VMM和Guest OS之间定义特殊的虚拟化接口来实现（Hyper 调用接口）。半虚拟化技术需要修改Guest OS的内核代码，将其中非虚拟化的特权指令替换为Hyper调用。因此，半虚拟化在性能上要优于全虚拟化技术。

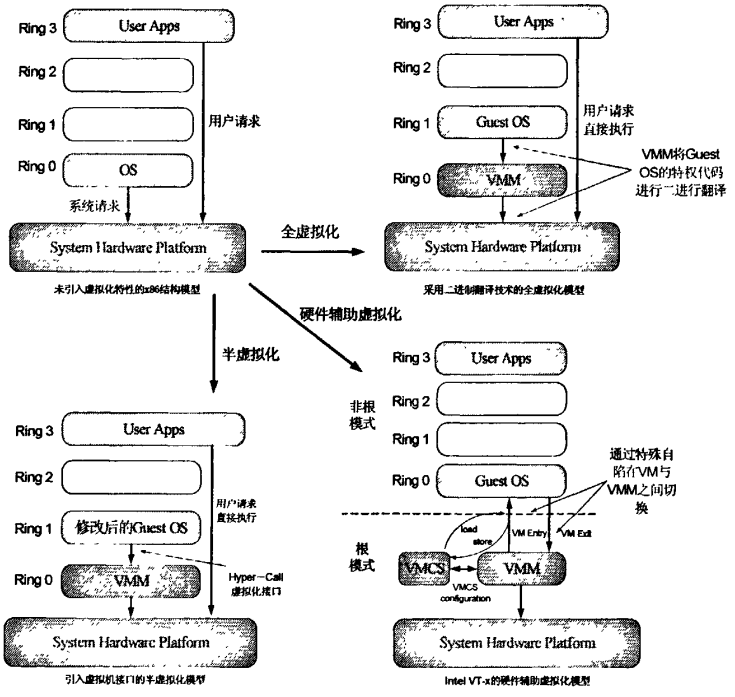


图 1.4 当前主要的虚拟化技术

表 1.1 三类主要的虚拟技术对比

虚拟化技术	全虚拟化	半虚拟化	硬件辅助虚拟化
实现方式	软件	软件	硬件辅助
主要技术	二进制翻译	加入Hyper调用	增加硬件接口
特点	隔离性好，实现简单	需要修改操作系统内核	需要修改硬件结构
性能	最差	优于全虚拟化技术	最好

由于软件技术的发展和硬件平台性能的提高，软件的虚拟化技术仍然具有很大的应用前景。尽管如此，完全依靠软件技术实现的虚拟化在性能上存在很大的缺陷，并且与应用的需求之间仍然存在很大的距离。对于Guest OS的异常、系统调用等事件的处理，软件方法都需要在VM与VMM之间的不断切换现场，从而产生了很大的开销，并严重影响了性能[17]。

随着虚拟化技术的不断发展，人们把目光投向了更底层的处理器硬件设计。依靠硬件上额外支持的虚拟化方案，不仅使得虚拟化的实现变得更加简单，而且可以容易的解决虚拟化过程中出现的上述性能问题。我们一般将这种硬件支持的虚拟化技术称为硬件辅助虚拟化技术。三类主要的虚拟化技术的对比如图 1.4 和表 1.1 所示。

伴随着虚拟化应用的推广和市场的扩大，应用在处理器上的硬件辅助的虚拟化技术被不断提出，并得到越来越得多的处理器产商的关注。x86 结构处理器上的硬件辅助虚拟化技术主要包括 Intel 的 VT [17-20]、AMD 的 V 技术[21] 等，RISC 结构处理器上的硬件虚拟化辅助技术则包括 Sun 的逻辑域技术[22]以及 IBM 的 APV 技术 [23]等。硬件辅助虚拟化技术的出现，使得虚拟化应用的性能得到了很大的提升。

1.2 课题研究的关键问题

随着微体系结构的不断发展，处理器进入了多核多线程时代。同时，由于当前计算机应用领域的需求，使得虚拟化技术不断发展并被引入到处理器设计当中。多核多线程结构的处理器具备任务级并行处理、吞吐量计算等特性，可以较好的适应虚拟化应用需求。因此，研究多核多线程结构下处理器的虚拟化设计，成为当今人们研究的热点。

本课题主要针对多核多线程架构中如何设计实现一个具备虚拟化特性的中断系统进行相关研究。中断一直以来都是计算机系统中的一个十分重要的概念[24]，现代计算机系统毫无例外地都采用了中断技术。在当代处理器当中，中断不但被用于处理外部设备和处理器内部紧急事件以及处理系统错误和故障，它还被处理器用于实时控制、被操作系统用于任务调度。当前，我们要在多核多线程结构的处理器中实现虚拟化的中断系统，既要考虑新结构处理器设计需求的变化，又要解决中断系统如何在硬件上支持虚拟化等问题。

1.2.1 多核多线程处理器中断系统的设计

同传统的单处理器相比，多核多线程处理器对中断系统的功能和性能的需求发生了很大变化。多核多线程结构下中断系统的设计必须解决以下问题：

- 中断系统对多线程的支持以及对处理器吞吐量的保证

与单线程处理器不同，多核多线程处理器的处理器核一般不使用深度流水、动态调度等复杂的指令级并行技术，而是通过多线程技术以实现核内的任务级并行。另外，多核多线程处理器追求的不是单个线程的性能，而是系统的整体吞吐量[25]。所以中断系统在加入对多线程的支持的同时，必须考虑到多核多线程结构

处理器吞吐量计算的特性，保证中断系统在处理某个线程的中断时不会影响到其它线程的中断处理。中断系统如何被多个线程共享，是多核多线程处理器中断系统设计时首先要解决的问题。

- 中断系统的功能和性能必须满足核间、核与 I/O 设备间的通信需求

在单核处理器中，设计者的主要精力放在了努力提高频率、提高指令级并行性（ILP：Instruction Level Parallelism）、提高 CPI 等方面。而在多核结构中，设计人员更关注的是核之间的任务、共享资源的分配以及提高线程级并行性等事务。核之间的通信机制、I/O 的分配以及核与 I/O 设备的通信机制都与处理器的中断系统密不可分。因此，多核多线程处理器中断系统的设计还必须充分考虑核间、核与 I/O 间通信的功能和性能需求。

1.2.2 中断系统的硬件辅助虚拟化设计

从功能和机制的角度来看，从硬件层次支持虚拟化应用的中断系统需要重点解决以下问题：

- 扩展的处理器特权等级

在引入虚拟化之后，虚拟化管理软件 VMM 一般运行在虚拟机和底层的物理硬件系统之间，为上层的客户操作系统提供一套虚拟的硬件平台，因此 VMM 的代码应该具有比 Guest OS 更高的控制权限。传统的处理器在设计时并没有考虑到虚拟化应用，在软件运行的特权级别上一般只区分了运行应用软件的用户级别和运行操作系统代码的特权级别。要从硬件层次解决，必须在处理器特权结构中加入新的特权等级，用于执行 VMM 的代码。当前各种硬件辅助虚拟化技术都对处理器的运行级别进行了扩展和设计[17]，只是实现的方式互有不同。一般来说，处理器特权模式的控制和切换，都离不开中断系统的支持。

- 设计中断系统的虚拟化接口

处理器中断系统的虚拟化接口包括两个方面：其一，是用户代码、操作系统级代码以及 VMM 代码之间的切换接口，体现在处理器运行时特权等级的切换以及各执行模式之间的自陷和调用；第二，要想从硬件上直接支持虚拟化应用，处理器的中断系统必须为各层次的软件分别提供必要的访问控制接口。传统非虚拟化处理器的中断系统没有严格的区分各层次软件在访问特权信息时处理器的运行机制（例如 X86 结构的处理器允许非特权指令访问 CPU 的特权状态），给 VMM 保护自身状态、VMM 和 Guest OS 之间传递控制权限等机制的实现造成了很大困难。因此，在为 VMM 的运行设置更高的特权级别的同时，处理器的中断系统必须为这一级别提供必要的权限保护，以及为 VMM 和 Guest OS 提供灵活的自陷接口和中断嵌套机制。

- 实现的外部中断机制的虚拟化

多核多线程处理器的每个核将多个线程集成在一起，核与核之间又通过片内总线或者交叉开关的方式与外部 I/O 接口互连。在多核多线程处理器当中，每个线程可以作为一个基本单位（虚拟 CPU）分配给虚拟机使用，VMM 软件会根据 VM 的负载情况，对所有的虚拟 CPU 进行调度和分配。外部 I/O 到虚拟 CPU，以及虚拟 CPU 之间都存在通信问题，这些通信事务都需要依靠中断来完成。因此，需要为多核多线程处理器设置虚拟化的外部中断接口，并解决外部 I/O 中断和机间中断到目标虚拟 CPU 的传递问题。

1.3 课题背景与主要工作

本课题是国家重大科研项目“高性能 X 处理器”的一部分。X 处理器是一款自主研发的多核多线程结构处理器，它采用了基于 SPARC V9 的扩展 RISC 指令集。处理器集成了 4 个内含 8 个硬件线程的处理器核，核之间通过交叉开关互连并共享 L2-Cache。为了满足当前的应用需求，X 处理器还采用了硬件辅助虚拟化技术，在指令集上进行了虚拟化扩展，并且在硬件上支持虚拟化应用。

本课题在针对多核多线程处理器中断系统设计以及中断系统虚拟化技术进行充分研究的基础上，根据多核多线程 X 处理器的需求与特点，提出了适用于 X 处理器、支持虚拟化应用的中断系统设计方案。根据该方案，我们定义了 X 处理器中断系统的软硬件接口，并对核内中断处理部件的进行了设计与实现。

课题的主要工作包括：

1、研究了多核多线程的处理器结构和虚拟化技术的相关理论。

2、分析了多核多线程处理器中断系统设计及其虚拟化的关键技术。

3、提出了一种适合于 X 处理器的中断系统设计方案。

4、根据上述设计方案，对 X 处理器的中断处理部件进行设计与实现。

5、完成了 X 处理器中断处理部件的模拟和验证。

1.4 本文的结构

本文共分为六章，各章组织如下：

第一章 绪论。介绍了课题研究的背景和研究的关键问题，阐述了课题研究的主要内容和工作。

第二章 多核多线程中断系统及其虚拟化。阐述了中断与中断系统的基本概念和中断处理的一般过程，讨论了多核多线程中断系统设计的关键问题，以及中断系统虚拟化设计的基本方法。

第三章 多核多线程 X 处理器虚拟化中断系统模型。本章主要提出了 X 处理

器中断系统的基本模型，定义了 X 处理器中断系统的软硬件接口，并阐述了 X 处理器的基本中断处理机制。

第四章 X 处理器中断处理部件的设计与实现。提出了 X 处理器中断处理部件的设计目标、总体结构，详细讨论了 X 处理器中断处理部件的模块设计与实现。

第五章 X 处理器中断处理部件的测试与验证。提出了 X 处理器中断处理部件的验证方案，阐述了中断处理部件的验证过程和验证内容，并给出了 X 处理器中断处理部件的验证结果和逻辑综合结果。

第六章 结束语。对全文工作做出了总结，并提出了课题需要进一步研究的相关问题。

第二章 多核多线程中断系统及其虚拟化

中断是计算机系统中不可或缺的重要组成。随着处理器结构的发展，处理器对中断系统的功能需求也在不断的发生变化。当前，处理器进入多核多线程和虚拟化时代，中断系统必须根据体系结构的发展进行不断的改进，才能提升处理器性能。本章主要阐述了中断与中断系统的基本概念和中断处理过程，讨论了多核多线程中断系统设计的关键问题，以及中断系统虚拟化设计的基本方法。

2.1 中断和中断系统

2.1.1 中断的基本功能

本文所述的中断是指在处理器运行期间，由于某种随机发生的事件（如指令异常、软硬件错误、设备中断请求、中断指令执行等）而使处理器暂停执行当前程序，转而执行另外一段程序，来做些必要的处理，以便满足突如其来的状况，完成后再返回暂停处继续执行原来的程序的过程[24]。本质上来说，中断是处理器为了处理当前异常、I/O 设备请求、软硬件错误，以及执行复位或者中断指令而改变指令流的一种行为。处理器设置中断的目的包括转移处理器的控制权限、处理程序中出现的异常、处理处理器运行过程中出现的软硬件错误、解决处理器与 I/O 设备运行速度不匹配问题、提高处理器效率等。

为了实现中断功能，除了进行必要的硬件支持之外，还需要一定的处理机制以及相应的软件代码。中断系统指的是用于中断的软、硬件及相应机制的总合。对于不同的处理器结构，中断系统的功能会有所区别，但是一般来说，一个处理器中断系统必须具备以下一些功能：

(1) 中断处理。中断系统能够响应中断请求、处理中断、中断处理完成之后返回正常指令流；

(2) 多中断源和中断优先级。中断系统可以处理来自多个中断源的中断请求，不同类型的中断应有不同的处理优先级；

(3) 中断屏蔽。中断系统可以通过必要的机制屏蔽部分类型的中断，当某种类型的中断被屏蔽时，处理器可以不去响应该类型的中断请求。

(4) 中断嵌套。中断系统的软硬件支持多重中断嵌套，允许更高优先级的中断打断当前正在处理的低优先级的中断。

2.1.2 中断的分类

根据引起中断的原因，现代处理器的中断一般分为由软件引起的中断和硬件

引起的中断。软件引起的中断[26]是指由与软件相关事件引起的中断的总和，一般来说包括：（1）中断指令产生的软中断；（2）指令异常；（3）软件执行错误和复位操作等。硬件引起的中断包括 I/O 设备中断、机间中断等。前者是指由 I/O 设备产生的中断，用于通知处理器处理某些 I/O 设备请求，完成 I/O 操作[27]；机间中断（IPI：Inter-Processor Interrupt）则是指在多机系统或者在多核处理器中，处理器间或者处理器核之间的中断请求。除此之外，硬件原因引起的中断还包括硬件定时器产生的定时器中断、硬件故障和硬件复位引起的中断等。

按照产生的时机及中断发生时处理器行为的区别，中断又可以分为精确中断、延迟中断和异步中断三类：（1）精确中断是当代处理器中断系统必须实现的功能，精确中断必须在发生任何软件可见的处理器状态改变之前进行处理，它必须确保被中断指令之前发射的所有指令均已执行，而其后的所有指令均未执行。这样中断服务程序在处理中断事件时，系统才能处于正确的状态。精确中断处理完成时处理器会恢复到中断之前状态，或者返回到某个特定状态。（2）延迟中断是指允许推迟到处理器状态改变之后处理的中断。延迟中断也是由某条特定指令触发的，中断发生时处理器状态有可能是被引起中断的指令本身改变的，也有可能被其后的指令所改变。（3）异步中断是与指令流异步的一种中断类型，它一般不是由某条指令直接引发，而是由异步事件产生的中断，例如 I/O 设备中断、机间中断、定时器中断等。一般来说，当一个异步中断返回时，处理器会从被中断的指令流处继续执行。

上述中断分类可以用表 2.1 总结如下。不同处理器的中断系统可以根据具体的需求实现上述的部分或者全部的中断类型。对于每一类中断的具体处理方法，也取决于处理器的定义和具体实现。

表 2.1 中断的基本类型

	软件引起的中断	硬件引起的中断
按中断源来分	自陷指令	I/O 设备中断请求
	指令异常	机间中断（IPI）
	定时器中断	硬件复位（POR、XIR）
	软件错误和复位	硬件故障
按中断行为分	精确中断（运算指令执行时发生的异常等）	
	延迟中断（Store 指令执行错误等）	
	异步中断（I/O 设备中断、定时器中断、软中断等）	

2.2.3 中断处理的一般过程

中断的处理过程一般可以分为中断请求、 中断判优、中断响应、中断服务和中断返回五个阶段[28]，如图 2.1 所示：

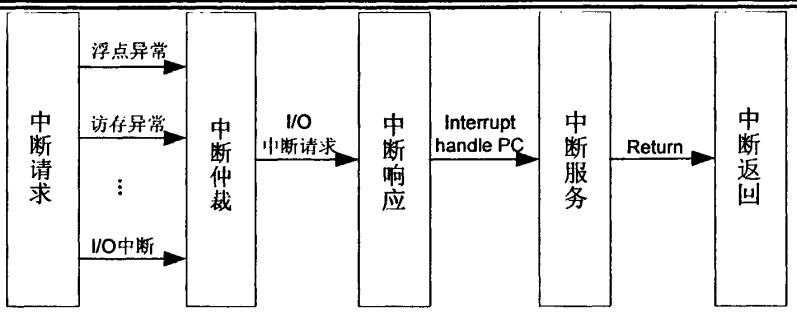


图 2.1 中断的一般工作过程

- 中断请求

中断请求可能来自于流水线的各站及外部的 I/O 设备。处理器对流水线出现中断有两种处理方法：集中式处理和分布式处理。集中式处理在流水线各站只检测中断请求而并不立即处理，它在流水线上设置专门的中断处理站，当指令进入该站时将按照指令顺序执行的次序对中断请求做出统一的处理；而在分布式处理方式中，只要在检测到中断请求就立即进行处理。

- 中断仲裁

在同一时间可能有来自不同部件或设备的中断请求到来，此时处理器必须根据一定的规则，从中选取出一个中断请求优先进行处理。现代处理器的中断系统一般都会约定不同中断类型的处理优先级，中断系统会根据优先级来选取当前所有请求中具有最高优先级的中断进行处理，这一过程一般被称为中断仲裁。需要指出的是，中断仲裁还受到中断屏蔽以及中断嵌套设置的控制。

- 中断响应

中断响应是指从中断发生到开始执行中断服务代码的过程。中断响应一般包括三个阶段：保存处理器状态、更新处理器状态、进行指令流的转换。为了在中断处理完成之后处理器可以返回中断指令处继续执行，中断系统必须保存中断发生时处理器的基本状态，包括部分或全部处理器状态/控制寄存器。为了执行中断服务程序，中断系统还必须在保存原有处理器状态之后，重新设置相应的状态/控制寄存器。上述过程之后，中断系统会根据中断类型和处理器原有状态等信息，产生中断服务程序的入口地址，并跳转到该地址处开始执行中断服务程序。

- 中断服务

中断服务是指中断服务程序处理指令异常、程序或硬件错误、I/O 设备请求或其它中断事件的过程。中断服务是一种软件行为，中断系统的硬件部分会根据中断系统约定软硬件接口将指令流转移到中断服务程序的入口处开始执行。

- 中断返回

中断服务程序完成相关处理任务之后，处理器会根据一定的规则返回到正常的指令流继续执行程序。中断返回的地址取决于具体的中断类型：对于精确中断

和异步中断，一般情况下都会返回到发生中断的指令处继续执行；对于复位中断或者延迟中断，处理器会恢复到一个特定的状态下，从特定的程序处执行（对于复位中断一般是初始化程序）。中断返回阶段一般会进行处理器状态的恢复（或设置）、指令流转移等操作。

2.2 多核多线程处理器中断系统的设计

在分析一个处理器的中断系统时，我们需要关注这个处理器中断系统基本组成和设计需求。对于多核多线程处理器中断系统的设计，我们关注的设计原则包括精确中断的实现、吞吐量计算的支持，以及外部中断机制的实现等方面。

2.2.1 精确中断的实现

实现精确中断必须满足以下三个条件[29]：（1）中断发生时正在执行的指令之前发射的所有指令必须已经执行完成，并且正确修改了处理器的状态。（2）在该指令之后的所有指令均为执行完成，并且还没有改变处理器状态。（3）如果中断是由指令执行引起的，那么需要保存的是该指令的 PC 值。这条指令可能已经执行完成，也可能还未开始执行，具体取决于不同的异常类型。

精确中断机制并不是多核多线程处理器特有的需求。不论是传统的单处理器还是多核多线程处理器，精确中断都是中断系统必须实现的功能。精确中断机制是处理器在处理完中断事件后能够返回到原有程序中断处继续执行的保证。

虽然单处理器和多核多线程处理器都存在精确中断冲突，但是两者实现精确中断机制所需要解决的问题又是不同的。传统单线程处理器精确中断冲突，主要由处理器引入的深度流水、乱序执行和推测执行等复杂技术引起，这些技术导致了指令的乱序执行和原有的程序顺序模型产生冲突[30]，从而产生了精确中断冲突。在单处理器中，精确中断冲突这一问题已经得到了很好的解决[29-31]。

多核多线程处理器核的设计一般不会使用多发射、乱序执行等复杂技术，它是通过多核技术和多线程技术来提升处理器性能的。另外，对于多核多线程结构的处理器来说，要将多个处理器核集成在同一个芯片上，核的设计也必须尽可能的简单。因此，多核多线程处理器中存在的精确中断冲突一般不是由指令的乱序执行引起。

同传统单处理器一样，多线程处理器一般包含多个功能部件，具备多条执行流水线，并且这些流水线的长度往往是不相等的。多核多线程处理器精确中断冲突主要由不等长的流水线设计引起。图 2.2 中所示即是一种典型的不等长流水线设计。其中，浮点流水线要长于整数流水线（这也是普遍存在的情况）。这种不等长的设计，是基于硬件开销和频率的考虑的结果：短的整水流水线可以减少流水

线旁路，减小面积；长的浮点流水线可以减少关键路径长度，提高频率。

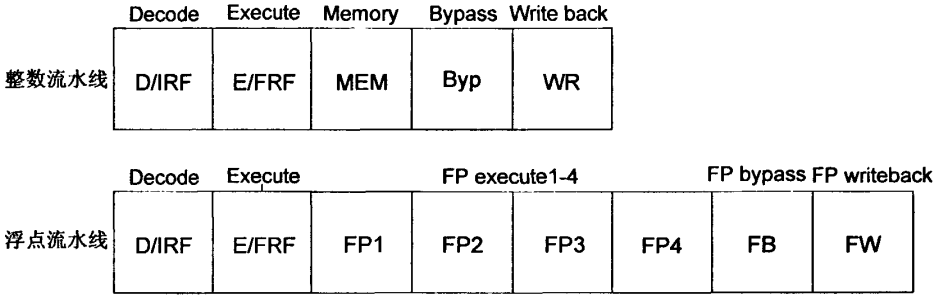


图 2.2 流水线不等长的设计

执行流水线的不等长设计会给多核多线程处理器实现精确中断带来潜在的异常冲突。紧跟浮点指令的之后的整数指令可能会在浮点指令检测到异常发生之前就更改处理器状态，从而导致浮点流水线无法实现精确中断。一个解决不等长流水线引起的精确中断问题的方法是在较长的浮点流水线中引入中断预测机制[32]，即通过必要的异常预测算法，在异常真正发生之前对正在执行的浮点指令进行预测，判断其是否会引起异常，从而可以尽早的对长流水线进行相应控制。利用浮点异常预测机制，我们就可以在后续的整数指令提交之前对浮点指令进行异常预测。一旦预测到浮点流水线可能发生异常，就可以停顿流水线或者排空流水线的方式保证指令的顺序模型。

2.2.2 吞吐量计算的支持

在多线程结构的处理器核中，中断处理部件往往被多个线程共享。因此，多核多线程处理器的中断系统必须加入对多线程的支持。由于核内可能有多条执行流水线，每条流水线可能被不同的线程所占用，所以某个核的中断处理部件在同一时间内所接收到中断请求可能来自于不同的线程。与单处理器的中断系统不同，多核多线程处理器的中断系统必须按照一定的策略选择处理某一个线程的中断请求，然后再对该线程的所有中断请求进行中断优先级的裁决。

在进行多核多线程处理器中断系统设计时，我们还必须考虑到如何尽可能的减少中断给流水线带来的性能损失以及消除中断处理可能导致的流水线阻塞。中断的产生和处理，必然会给流水线的性能带来一定的损失。除了发生中断时排空流水线造成性能损失[31]之外，中断转移、保存中断现场、中断返回恢复现场等操作的延迟如果不能很好的隐藏，都会使得流水线停顿等待。多核多线程处理器追求的是一种吞吐量计算，因此它比传统单处理器更加关注流水线的性能损失。多线程结构中，中断可能产生的流水线停顿和排空不仅会影响到单线程执行的性能，而且可能会使得其它线程的执行被某一线程的中断处理过程阻塞，从而影响到处理器的整体吞吐量。

由于在多核多线程处理器中，线程往往以细粒度调度的方式占用流水线，因此可以充分依靠线程切换来隐藏中断响应和中断返回的过程中可能存在的长延迟操作。例如，我们可以设计流水线的各功能部件向中断系统报告异常的同时，就立即排空本部件流水线中同一线程的指令，使得其它已准备好的线程可以快速的占用发射指令并占用流水线。同时，处理器的中断系统对异常请求进行保存，并按中断处理步骤对请求进行仲裁和响应。我们在某一线程恢复现场时也可以采用类似的做法，通过线程切换，来隐藏取值部件重新取指、以及线程现场恢复的时间。

要消除中断处理可能导致的流水线阻塞，需要减少中断响应时间和减少中断现场的保存恢复时间。中断响应时间是指从中断发生到开始执行中断服务代码的时间[33]。中断系统在产生中断服务程序入口地址以后，需要从该地址取出指令执行。由于中断服务程序一般存在于内存的特定区域当中，因此在发生中断时往往要进行访存操作，延长了中断响应的时间。减少中断响应时间的方法有很多，中断指令缓冲区[34] 和分级中断服务[35]是两种比较有效的方法。中断现场保存和恢复时间是指从保存中断现场和恢复中断现场所需要的时间。提高中断现场恢复速度的方法有很多[31,34]，在多核多线成处理器中，为了提高中断现场保存和恢复速度，消除可能引起的流水线阻塞，可以在为每个线程维护独立的处理器状态的同时，为每个线程设置单独的存储区域来保存该线程的处理器现。

2.2.3 外部中断机制的实现

高效的处理器核之间及处理器核与 I/O 设备的通信是多核多线程结构处理器一个很重要的设计目标。多核多线程处理器内部包含多个线程，所有的处理器核及核内的线程又共享着外部 I/O 设备。处理器核之间的各线程的通信、以及线程与外部 I/O 设备的通信都离不开中断系统的支持，前者需要中断系统提供机间中断机制，后者需要中断系统提供 I/O 中断的处理功能。

我们将机间中断和 I/O 中断统一称为外部中断。为了实现快速的外部中断，多核多线程处理器可以通过为每一个线程设置特殊的中断配置寄存器和外部中断向量来发送和接收外部中断。为了统一接口，减少设计复杂度，外部中断向量被用来同时接收机间中断和 I/O 中断。机间中断的发送根据中断配置寄存器的内容形成，并以报文的形式通过处理器内部的互连结构传输。这样，就不需要为机间中断设计单独的中断总线。由 I/O 设备产生 I/O 中断消息，也经由 I/O 桥通过片内的互连结构直接传输给处理器核的目标线程。这样，可以提高机间中断和 I/O 中断传递的速度。

2.3 中断系统的虚拟化设计

采用硬件辅助虚拟化设计的方法可以较容易的软件虚拟化技术中存在的性能问题。对于处理器中断系统的虚拟化设计，着重从硬件机制上解决以下问题：（1）设置更高权限的执行模式用于执行 VMM 的特权指令，在中断系统中为各特权模式的控制和转换提供必要的接口；(2)对中断系统的软硬件接口进行虚拟化设计，，满足各层次软件的中断控制和处理需求；（3）提供硬件辅助的虚拟化 I/O 中断、机间中断机制，满足外部中断虚拟化的需求。

2.3.1 执行模式的虚拟化

从处理器执行的特权模式来看，非虚拟化的处理器一般只具有用户（User Mode）和特权（Privileged Mode）两种执行模式。其中，用户模式一般用于执行应用程序的代码，而特权模式用于执行操作系统的特权代码。处理器在用户模式和特权模式之间通过中断请求、异常陷入和系统调用、完成/返回等方式进行模式转换，如图 2.3（A）所示：

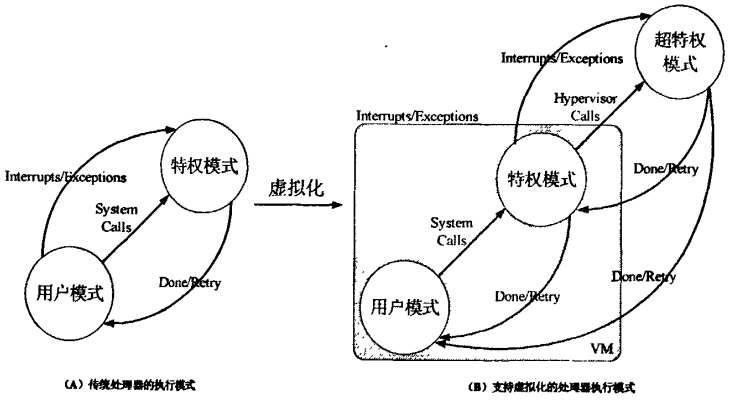


图 2.3 处理器执行模式的虚拟化设计举例

如果要在非虚拟化的处理器上运行虚拟化应用，为了保证 VMM 能够完全管理和控制实际的硬件资源，监控和维护各 VM 的状态，就必须使 VMM 的代码运行在特权模式上、使 VM 中操作系统的特权代码和用户程序的非特权代码运行同时运行在用户模式下，然后通过二进制翻译或者 VMM 的虚拟化调用，来解决 VM 中客户操作系统特权代码“直接执行”的问题[17]。这样，虚拟机中的操作系统每执行一次特权操作，都必须通过软件接口陷入到处理器的特权模式完成。从而导致整个虚拟化系统在 VM 和 VMM 之间进行频繁的切换现场。由于非虚拟化处理器的中断系统没有为这种陷入提供硬件接口，因此许多工作都必须由软件来完成，所产生的开销显然是很大的。

从中断系统的硬件层次进行虚拟化设计，通过设置新的处理器执行模式并提供硬件支持的自陷接口可以较好的解决虚拟化的性能问题。图 2.3（B）所示是处理器执行模式虚拟化的一种方式。新的处理器执行模式由原有的用户－特权两级模式变为用户－特权－超特权三级执行模式[25]。用户模式用于执行 VM 中应用程序的代码，特权模式用于执行 VM 中客户操作系统的特权代码，而超特权模式用于执行 VMM 中涉及到底层实际硬件管理和分配、以及 VM 管理和维护的更高权限的代码。硬件上为三种执行模式的切换提供必要的陷入/返回接口，使得用户程序、Guest OS、VMM 之间的状态切换变得更加高效和简单；从硬件上提供不同执行模式切换时现场保存和恢复，也使得虚拟化应用的性能得到了提升。

2.3.2 中断接口的虚拟化

在虚拟化结构中，VMM 为每一个虚拟机都提供一个完整的 CPU 抽象接口，包括控制一个 CPU 所必须的所有状态/控制向量。VMM 调度 VM 的启动、运行、挂起、关闭等操作，以及 VM 存储空间和 I/O 资源的分配的管理工作，都通过这个虚拟的 CPU 接口来完成。

虚拟的 CPU 接口可以由软件方式实现。软件为每个 VM 在内存中创建并维护一个独立的数据结构，用于保存 VM 以及其虚拟的 CPU 的状态信息。当 VMM 调度 VM 在处理器上运行时，VMM 根据将内存中 VM 的数据结构中的信息恢复实际处理器的状态。当 VM 被调度出流水线时，由 VMM 将 VM 运行时的处理器状态保存到管理结构中。显然这种方式的性能比较低效。

虚拟化的 CPU 接口也可以由硬件来实现。硬件提供给软件直接的接口就是处理器体系结构寄存器。为了实现中断系统的虚拟化，我们可以为各层次软件分别提供不同的接口寄存器来直接操作处理器，并分别为各虚拟机提供独立的硬件接口，对处理器进行直接的控制（图 2.4）。硬件实现的虚拟化寄存器接口可以带来以下两个方面的好处：首先，VMM 只需要在创建 VM 时给 VM 分配和初始化 VM 的状态寄存器，而在 VM 调度运行的过程中，只要 VM 个数没有超过硬件支持的最大负载，VM 的运行状态就不需要在处理器和内存之间恢复和保存，极大的减少了运行开销。其次，当处理器直接运行在特权级别时，这些硬件提供给 VM 的寄存器接口，可以直接被 VM 的客户操作系统访问，操作系统可以直接执行部分的特权操作，从而提高了虚拟化性能。

我们将与中断处理相关的体系结构寄存器进行了虚拟化设计，把软件可见的寄存器设置不同的访问权限：（1）特权状态寄存器，操作系统特权指令可以访问的寄存器。特权状态寄存器是为 VM 的客户操作提供的处理器接口，用于操作系统直接执行部分特权操作，包括浮点使能、寄存器窗口管理、中断屏蔽、特权模

式定时器、软件中断寄存器等等。（2）超特权状态寄存器，只能通过 VMM 的超特权指令进行访问的寄存器。超特权状态寄存器为 VMM 提供的虚拟化管理接口，包括用于控制硬件线程在超特权模式下执行状态的寄存器、超特权模式下定时器中断及其使能的寄存器、超特权中断向量表的基地址寄存器、超特权中断现场状态寄存器，以及用于定义特权模式和超特权模式下最大中断等级、全局寄存器个数的寄存器等。

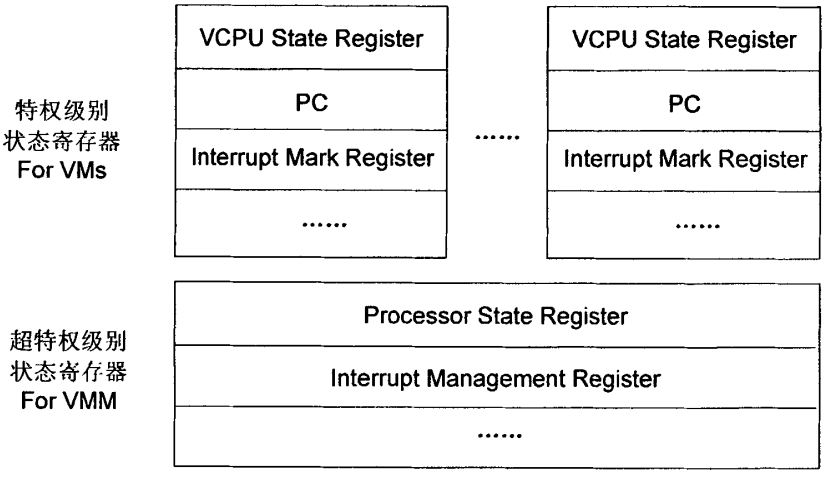


图 2.4 处理器虚拟化的寄存器接口示意图

为了提高安全性，一般还会约定上述寄存器的访问权限，在应用程序代码访问特权状态寄存器、应用程序或操作系统代码访问超特权寄存器，都会引起异常。

2.3.3 外部中断的虚拟化

为了实现外部中断的虚拟化，中断系统可以从硬件上为虚拟 CPU 维护两个虚拟化的中断队列——机间中断队列和 I/O 中断队列，分别用于保存和处理机间中断和 I/O 中断请求。队列的存储空间可以由 VMM 在内存中进行分配，也可以用特定的硬件存储结构实现，但是队列的头尾指针一般都由硬件维护，以提高处理效率。

处理器可以为每个虚拟 CPU 设置一个特殊的外部中断向量和向量中断，以检测是否有机间中断和 I/O 中断请求的到来。当外部中断报告到来时，硬件会根据外部中断的类型设置某一向量位。任何对外部中断向量的修改都会引起一次向量中断，使得虚拟 CPU 陷入到超特权模式中。于是外部中断消息的传输可以分成以下两步来完成：

（1）来自其它虚拟 CPU 或 I/O 的中断通过向量中断将中断消息传递给超特权代码（如 VMM），向量中断服务程序将相应的外部中断消息插入到相应虚拟化中断队列的尾部；

（2）硬件通过判断中断队列的头尾指针检测每个虚拟化中断队列是否有外部

中断报文来处理。如果有，则产生虚拟机间中断或者虚拟 I/O 中断并使 CPU 陷入到特权模式来处理机间中断和 I/O 中断消息。

上述 I/O 到虚拟 CPU，以及虚拟 CPU 到虚拟 CPU 的中断消息传递过程分别如图 2.5 和图 2.6 所示：

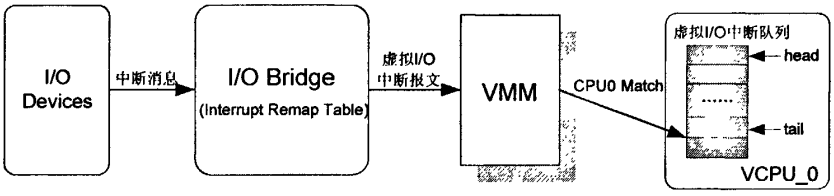


图 2.5 I/O 至 VCPU 中断的传递

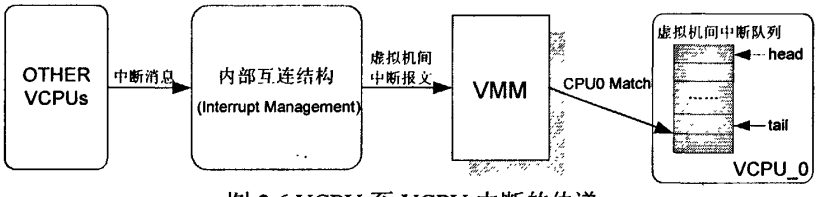


图 2.6 VCPU 至 VCPU 中断的传递

2.4 本章小节

多核多线程处理器中断系统设计重点包括实现精确中断、实现对吞吐量计算的支持，以及实现高效的外部中断机制等。而中断系统的虚拟化设计则需要从硬件上解决执行模式扩展、硬件接口虚拟化和外部中断虚拟化的问题。本章分析了上述设计需求和存在的问题，并简要总结了解决这些问题的常用方法。

第三章 多核多线程 X 处理器虚拟化中断系统模型

X 处理器是一款自主研发的多核多线程处理器。出于应用的考虑，该处理器需要进行虚拟化扩展，以从硬件上支持虚拟化应用。针对 X 处理器的特点和需求，本章在前文研究的基础上提出了适用于 X 处理器的中断系统模型，定义了 X 处理器的软件接口，并且描述了 X 处理器中断控制和中断处理机制。

3.1 X 处理器中断系统模型

3.1.1 处理器的运行模式

在多核多线程 X 处理器中，每一个线程都具备独立的拥有独立的寄存器堆栈、状态和控制寄存器，在逻辑上都可视为一个完整的 CPU。为了使处理器从硬件上支持虚拟化应用，X 处理器对其中所有逻辑 CPU 的运行模式进行虚拟化扩展，由传统处理器中的“用户－特权”二级模式扩展成为“用户－特权－超特权”三级模式，并将这些逻辑 CPU 视为独立的单位分配给虚拟机使用。处理器的每一个逻辑 CPU 都始终运行在其中一个特定的特权模式下，这样用户程序代码、部分客户操作系统的特权代码以及 VMM 的代码都可以在 CPU 上直接执行（图 3.1）。

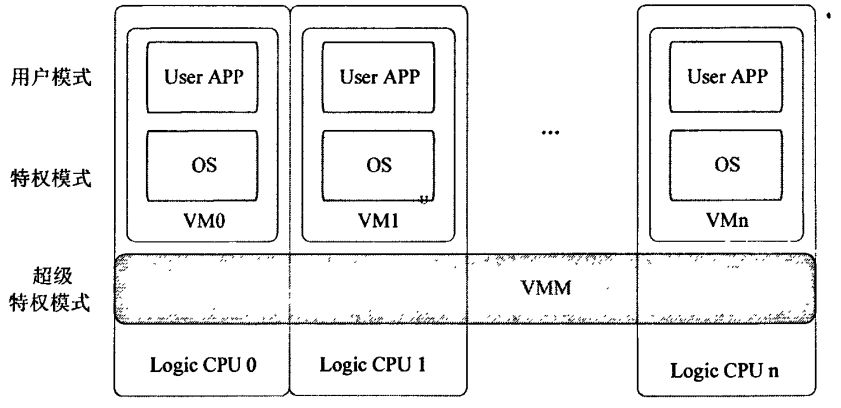


图 3.1 X 处理器执行模式的设定

X 处理器的各种运行模式的作用如下：

- 用户模式

具有一般的执行权限，用于执行用户程序代码。用户模式下执行的指令对处理器资源的访问是受限的，对于 CPU 大多状态/控制寄存器都没有访问权限。

- 特权模式

具备操作系统的执行权限，用于执行虚拟机中客户操作系统的部分特权指令。操作系统特权模式的指令只能访问到 VMM 给虚拟机分配的硬件资源，比如分配

给虚拟机的某个逻辑 CPU 的控制/状态控制寄存器，对于其它硬件资源，或者更高特权的状态/控制寄存器无法访问。

- 超特权模式

具备最高的执行权限，用于执行 VMM 软件的特权指令。这些指令操作或者为虚拟机中的 Guest OS 提供底层虚拟 CPU 的接口服务，或者负责管理分配实际的硬件资源。

3.1.2 中断模型

X 处理器的中断系统为处理器执行模式的转换提供自陷接口，并根据每个逻辑 CPU 当前的执行模式进行权限控制和中断处理。X 处理器的中断系统模型如图 3.2 所示：

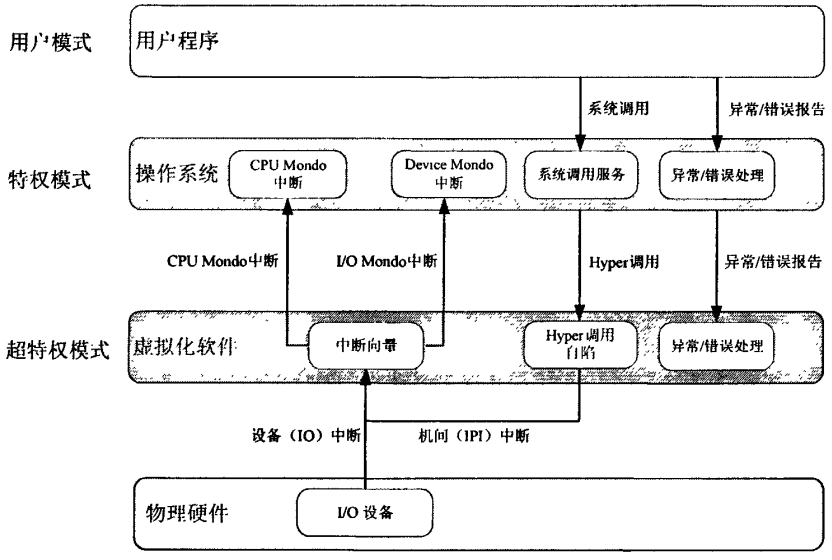


图 3.2 X 处理器中断系统模型

X 处理器中断模型主要包括：

（1）异常和错误：由指令执行过程中产生的异常或错误引起的中断，例如浮点的除 0 异常、TLB 失效等。

（2）自陷：主要由自陷指令引起的中断，用于满足用户模式到特权模式的系统调用和以及特权模式到超特权模式的 Hyper 调用的需要。

（3）外部 I/O 中断：实际物理 I/O 设备的中断并不直接送到 VM 进行处理，而是交由硬件完 I/O 中断重映射之后，通过外部向量中断接口传递到 VMM 进行处理。VMM 为每个 VM 维护单独的外部 I/O 中断队列，Guest OS 通过特殊的队列中断接口完成 I/O 中断消息的处理。

（4）机间（IPI）中断：X 处理器中各 CPU 之间的中断请求。机间中断的处

理与 I/O 中断的处理过程类似，所不同的是机间中断源不是来自外部设备，而是其它的硬件线程。

（5）其它中断：如定时器中断、软件中断、复位中断等。

从中断来源看，X 处理器的中断可以分为 CPU 内部中断及外部中断。内部中断是指 CPU 在执行过程中产生的中断，如指令异常、自陷、软中断、定时器中断等，该类中断的响应和处理一般由核内的中断处理部件完成；外部中断是指外部 I/O 设备产生的 I/O 中断、以及来自于其它 CPU 的机间中断，外部中断的处理需要核外 I/O 桥逻辑和核间互连控制逻辑的传递。

按照处理中断的 CPU 运行模式，X 处理器的中断还可以分为特权模式的中断和超特权模式的中断。前者一般指是由客户操作系统代码处理的中断，需要传递到 CPU 的特权模式下处理；而后者需要 VMM 层次的软件才有权进行处理，需要 CPU 切换到超特权模式。

3.2 X 处理器中断系统的软件接口

X 处理器中断系统的软件接口包括处理器的中断等级、中断类型与优先级、中断向量表等。

3.2.1 中断等级

中断等级规定了处理器在不同特权级别的运行模式下所允许的最大中断嵌套层数。X 处理器中为每一逻辑 CPU 设定了 6 个中断等级。其中，0 级用于用户模式下代码的正常执行，其它中断等级用于不用权限的代码处理各种异常和中断事件。X 处理器中断等级的具体分配情况如表 3.1 所示：

表 3.1 X 处理器中断等级的分配

中断等级	相关的执行模式
0	用户模式
1-2	特权模式
3-5	超特权模式

根据表 3.1，CPU 在特权模式下运行时最多只支持 2 层中断嵌套，在超特权模式下最多支持 5 层嵌套。对此做出以下约定：如果一个 CPU 运行在特权模式并且中断嵌套层数已经等于 2 级时再发生中断，就可能会引起 CPU 切换到超特权模式运行；如果 CPU 已经运行在超特权模式下并且中断嵌套的层数已经达到 5 级，此时如果再发生中断，CPU 将进入一种调试模式。后续的小节将对 X 处理器中断处理与 CPU 运行模式切换之间的关系进行详细的阐述。

3.2.2 中断类型和中断优先级

中断系统另一个重要的软件接口是中断类型的定义和中断优先级的约定。在X处理器中，一共定义了88种类型的中断。这些中断类型的分类以及中断处理的优先级由高到低的顺序排列如下：

1. 复位中断；
2. I/O中断与机间中断；
3. 除法指令异常；
4. Load Miss和其它长延迟指令的异常；
5. 浮点流水线中一般的指令异常；
6. 整型流水线及存储流水线中的一般指令异常；
7. TLB reloads中断。

优先处理I/O中断和机间中断，可以满足I/O设备和机间通信实时性的要求；优先处理长延迟操作的产生异常，可以尽可能早的恢复执行长延迟操作，从而达到隐藏延迟的目的。

由于核内的中断系统被多个线程共享，因此中断系统还必须约定不同线程中断事件的处理优先级。在X处理器中，约定优先处理编号较小线程的中断请求，即按照线程0到线程7的优先顺序来处理各线程的可能的中断事件。

3.2.3 中断向量表

出于虚拟化和安全性的考虑，X处理器为每个逻辑CPU定义了三类中断向量表，分别用于特权模式、超特权模式和调试模式代码的中断处理。这些中断向量表的结构组织如表3.2、表3.3和表3.4所示。为了提高中断响应速度，中断向量表的每一项大小均为32字节，用于存放相应中断处理代码的前8条指令。特权模式的中断向量表被设计成2个部分，分别用于用户模式到特权模式以及特权模式到特权模式传递的中断处理。

表3.2 特权中断向量表的组织

中断等级	软中断类型	硬件中断类型	偏移地址	表项内容
IL=0	—	00016—07F16	016—FE016	硬件中断
	—	08016—0FF16	100016—1FE016	寄存器窗口中断
	016—7F16	10016—17F16	200016—2FE016	特权模式处理的软中断
IL=1	—	00016—07F16	300016—3FE016	硬件中断
	—	08016—0FF16	400016—4FE016	寄存器窗口中断
	016—7F16	10016—17F16	500016—5FE016	特权模式处理的软中断

表 3.3 超特权中断向量表的组织

软中断类型	硬件中断类型	偏移地址	表项内容
—	00016—07F16	016—FE016	硬件中断
—	08016—0FF16	100016—1FE016	寄存器窗口有关的中断
016—7F16	10016—17F16	200016—2FE016	超特权至超特权模式的软中断
8016—FF16	18016—1FF16	300016—3FE016	其它模式至超特权模式的软中断

表 3.4 复位/调试模式中断向量表的组织

硬件中断类型	偏移地址	表项内容
1	2016	上电复位
2	4016	Watch-dog 复位
3	6016	外部初始化复位
4	8016	软件初始化复位
5	A016	复位调式模式下的异常处理

不同运行模式下中断向量表基地址的形成也是不同的，CPU 特权模式和超特权模式中断向量表基地址由每个 CPU 相应向量表的基地址寄存器来保存，对于复位/调试模式的中断向量表，由于该模式是一种资源访问受限的执行模式，因此其中断向量表的基地址是约定的常数。

CPU 不同运行模式中断向量表入口地址的形成方式如图 3.3 所示：

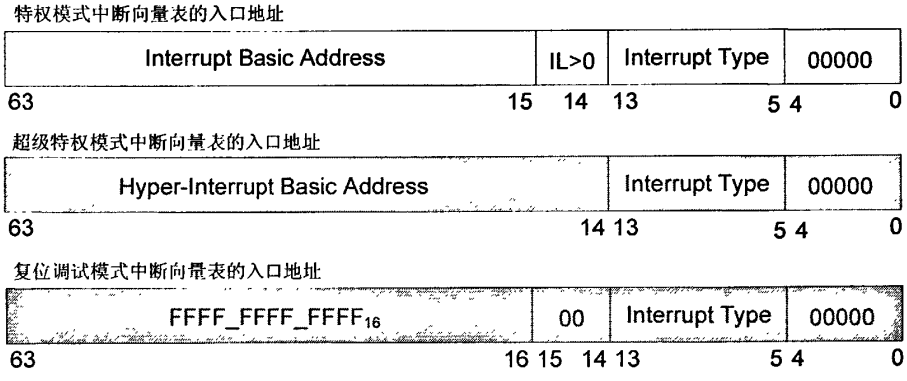


图 3.3 中断向量表入口地址的形成

3.3 X 处理器的中断处理机制

3.3.1 中断与处理器运行模式的切换

在 X 处理器中，除了直接修改逻辑 CPU 状态寄存器之外，中断处理也会引起 CPU 运行模式发生变化。中断将使得一个 CPU 从用户模式进入特权模式或超特权模式，或者从特权模式进入超特权模式。根据约定，X 处理器中断处理所导致的

CPU 运行模式的切换取决于中断类型、CPU 当前中断等级、以及 CPU 当前运行模式三个因素。X 处理器中断处理引起 CPU 运行模式切换的条件如图 3.4 所示：

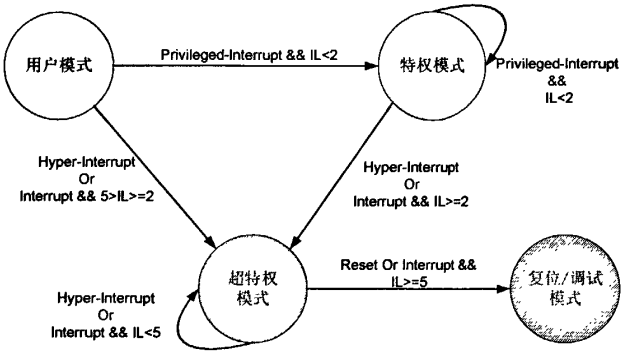


图 3.4 CPU 执行模式的切换

不同类型的中断可能需要由不同特权级别的代码来处理。当 CPU 运行在用户模式时，如果发生一个可以由操作系统特权代码处理的中断，则 CPU 会被中断系统切换到特权模式运行，以便处理该中断。同样，如果在用户模式或特权模式下，CPU 接收到一个需要由 VMM 代码处理的中断请求，则中断系统会直接将 CPU 切换到超特权模式进行中断处理。

中断导致的 CPU 运行模式的切换还取决于中断发生时 CPU 的中断等级。如果在中断等级等于 2 时发生一个需要特权代码处理的中断，则 CPU 会从用户模式直接切换到超特权模式（在 CPU 中断等级小于 2 时，同样一个中断会被传递到特权模式处理）。每个 CPU 还设置了一个特殊的复位/调试模式，用于处理 CPU 复位、致命错误、调式等事件。超特权模式下，如果在某一 CPU 已经达到 X 处理器所支持的最大中断嵌套层数（5 级）的情况下再发生一次中断，则 CPU 进入复位调式模式进行特殊处理。

3.3.2 中断处理流程

X 处理器中断系统的中断处理流程如图 3.5 所示：

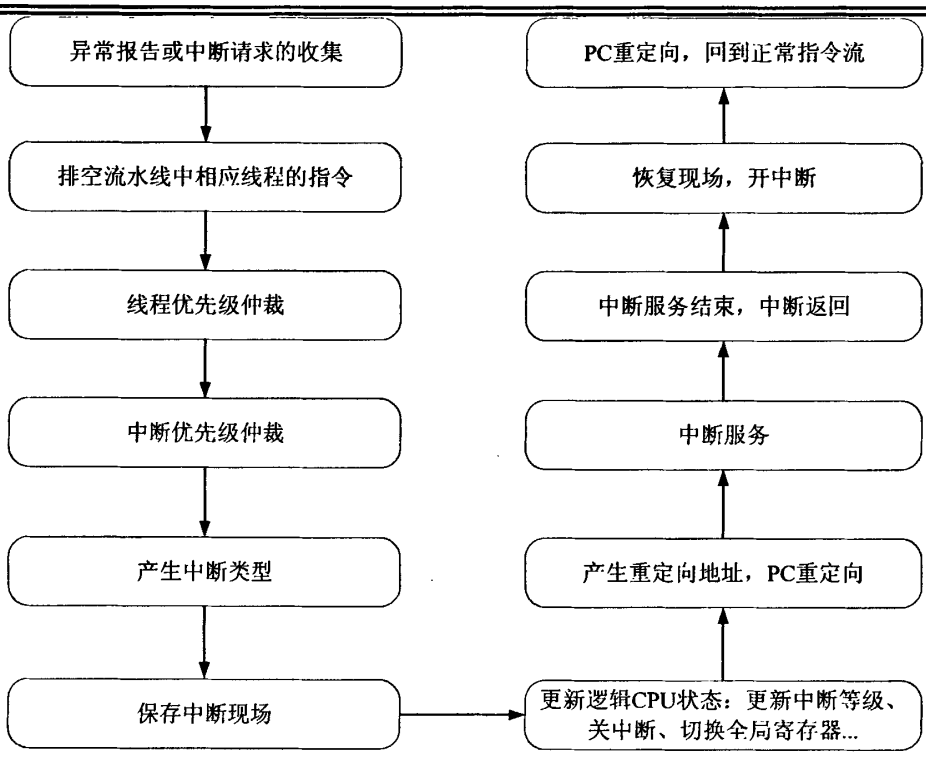


图 3.5 X 处理器中断处理流程

X 处理器中断处理的主要步骤包括：

1、异常、中断请求的收集

X 处理器中断系统对所有中断请求采用集中方式进行处理。对于指令异常等同步中断，中断系统收集来自各流水站的异常报告，并站存到同一站对所有中断请求进行处理。对于外部中断等异步中断，X 处理器的中断系统将异步中断请求插入到流水线中的某一站中，和同步中断请求进行统一的处理。

2、中断仲裁

由于 X 处理器核内有多条执行流水线，并且线程以细粒度的方式调度占用这些流水线，所以处理器的中断系统同一时间会收到来自于各线程的中断请求；即使对于同一个线程而言，也可能有多个不同来源的中断请求同时报告给中断系统。因此 X 处理器的中断系统既要进行线程仲裁，也要对同线程进行中断优先级仲裁。

3、现场保存和恢复

X 处理器采用了全局寄存器的思想[36]，处理器核为每个线程都设置一定数量全局寄存器组。发生中断时，程序数据可以依靠全局寄存器组指针的更新和全局寄存器组的切换来完成，这种做法可以减少保存和恢复中断现场的时间。此外，X 处理器的中断系统还维护一个独立的中断堆栈用于保存中断发生时处理器中各线程的处理器状态。堆栈的大小根据 X 处理器核线程的个数以及处理器支持的最大

中断嵌套层数而定。在中断现场保存的过程中，处理器的中断系统根据线程编号和中断嵌套等级产生访问地址，并将需要保存的 CPU 状态按照约定的组织保存到堆栈的对应地址中；中断恢复过程中，中断系统根据相同的地址从中断堆栈中读取先前保存的现场用于 CPU 状态的恢复。

4、中断服务程序入口地址的产生

X 处理器中断系统硬件为每个 CPU 维护着特权和超特权两个中断向量表的基地址。中断系统根据向量表的基地址和中断类型产生重定向地址，即中断服务程序第一条指令的地址。

中断处理的流水化可以提高中断处理的效率。在 X 处理器中，中断处理的上述过程被划分成多站来完成，中断处理的流水站划分和每一站所完成的操作如图 3.6 所示：

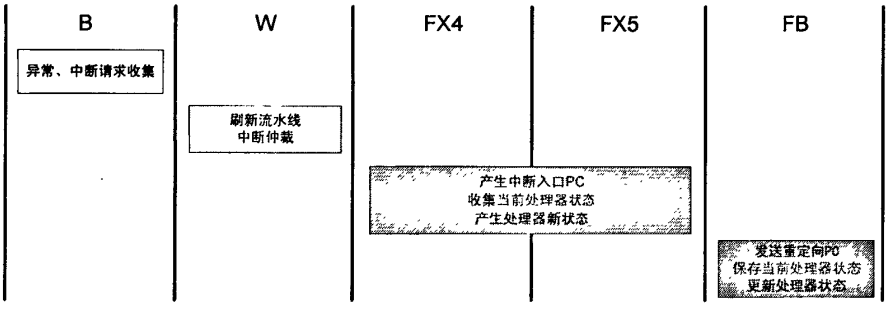


图 3.6 X 处理器中断处理流水线

3.3.3 中断处理的控制

X 处理器的中断处理受到多个控制寄存器的控制，如表 3.5 所示：

表 3.5 中断处理相关的控制寄存器

寄存器	作用
PSTATE	CPU 特权状态控制寄存器
HPSTATE	CPU 超特权状态控制寄存器
FPRS	浮点寄存器状态寄存器
FSR	浮点状态寄存器
PIL	处理器中断等级寄存器，设置处理器响应的最高外部中断等级
IL	当前处理器的中断等级

上述寄存器每个线程一套，它们对中断处理的控制作用具体如下：

1. PSTATE 寄存器的中断使能域和 PIL 寄存器控制着是否产生外部中断和相应等级的软件中断。

2. FPRS 寄存器的浮点部件使能域、PSTATE 寄存器的浮点部件使能域，以

及 FSR 寄存器的中断屏蔽域决定着浮点异常是否产生中断。

3. IL 寄存器包含了当前嵌套的中断等级，决定着一个中断是否会引起 CPU 运行模式的切换。

4. 对于在特权模式处理的中断，PSTATE 寄存器的中断大小端格式域决定着中断处理过程中的一般的数据访问使用大端还是小端的字节顺序。对于超特权模式的中断服务程序，X 处理器约定只能使用大端字节序。

对于不同情况中，上述控制寄存器的作用会有所不同，如表 3.6 所示：

表 3.6 不同执行模式中断的控制方式

中断类型	说明
用户模式/特权模式至特权模式的中断	由 PSTATE.IE、PIL、IL 的控制
超特权模式至特权模式的中断	不由 PSTATE.IE 和 IL 的控制
其它模式至超特权模式的中断	不可屏蔽，立即处理
超特权模式至超特权模式的中断	由 PSTATE.IE 的控制

3.3.4 外部中断的处理

为了实现外部中断的虚拟化，X 处理器中断系统在硬件上为每个线程维护了两个虚拟化队列，分别用于保存和处理 I/O 中断和机间中断请求。同时，还为每个线程设置了一个外部中断向量，以检测和接收 I/O 中断和机间中断请求。I/O 中断和机间中断的处理过程基本类似，要经历传递到超特权模式，再传递到特权模式的过程，这一过程是由不同级别的软件和硬件共同完成的。

3.3.4.1 I/O 中断的传递和处理

I/O 中断请求的处理过程包括以下几个步骤（图 3.7）：

1、由 I/O 桥完成外部 I/O 中断消息的重映射，产生虚拟化的 I/O 中断报文，并向目标 CPU 发送外部中断标识。

2、目标 CPU 接收到中断标识后更新中断向量，并产生向量中断，陷入到超特权的向量中断服务程序代码中。该中断服务程序将内部 I/O 中断报文插入到 I/O 中断队列尾部。

3、目标 CPU 对 I/O 中断队列头尾指针进行比较以检测是否有未处理的 I/O 中断报文。如果队列非空，则表示存在未处理的 I/O 中断报文，从而陷入到特权模式中产生 I/O 队列中断，该中断的服务程序从 I/O 中断队列头部取出中断报文，并根据中断类型设置软中断向量。

4、处理器根据软件中断向量的设置产生不同类型和级别的中断，这些中断服务程序对不同类型的 I/O 中断进行处理。

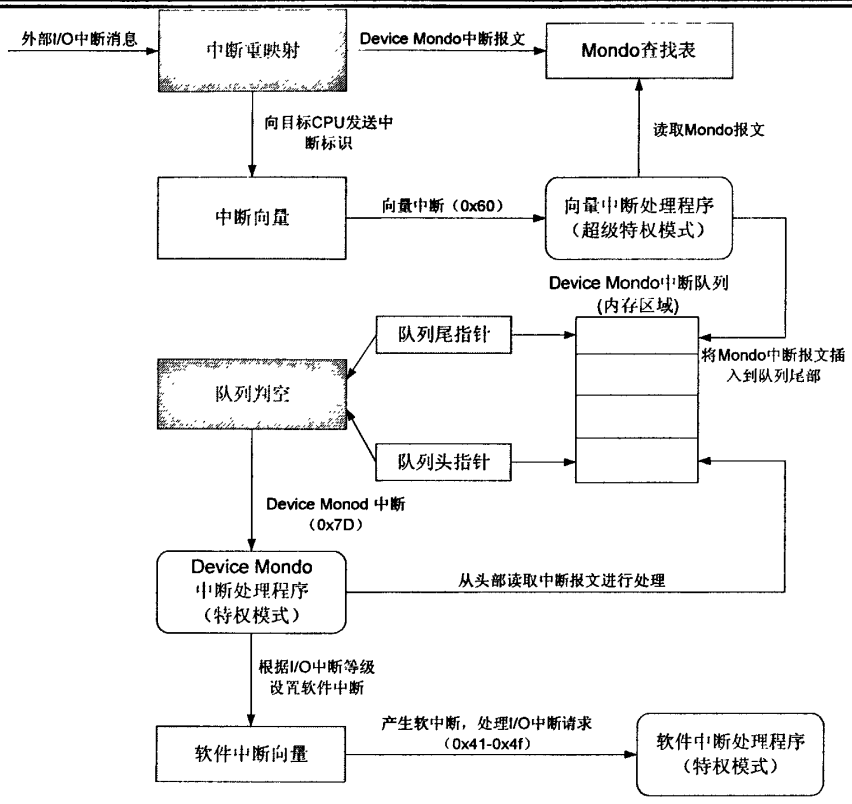


图 3.7 I/O 消息中断处理过程

3.3.4.2 机间中断的产生和处理

在 X 处理器中，每个逻辑 CPU 通过写机间中断寄存器向其它 CPU 发送机间中断请求。机间寄存器包含了目标 CPU 的编号以及中断标识。处理器核的互连接口根据机间中断寄存器的内容形成特殊的机间中断消息，送往目标 CPU。

机间中断消息的传递和 I/O 中断消息传递过程类似，并且两者使用同一个外部中断接口。每个 CPU 的外部中断接口根据中断请求中包含的 CPU 编号和中断标识设置相应 CPU 的外部中断向量，并产生向量中断。然后向量中断的处理程序将外部中断消息插入虚拟化机间中断队列尾部。中断处理逻辑通过比较中断队列的指针来判断是否存在未处理的机间中断消息。如果有，则会陷入到特权模式中产生 CPU Mondo 中断来处理该中断。

3.4 本章小结

本章主要提出了 X 处理器中断的基本模型，包括中断系统的软件接口，以及中断系统的处理、控制机制。该模型可以很好的适应 X 处理器多核多线程的结构特点，并满足 X 处理器引入硬件辅助虚拟化设计的需求。

第四章 X 处理器中断部件的设计与实现

前文分析了多核多线程结构处理器中断系统的虚拟化设计需要解决的问题以及基本解决方法，提出了多核多线程 X 处理器中断模型。本章将根据 X 处理器中断系统的设计目标，设计实现 X 处理器核内的中断部件。

4.1 设计目标

中断部件是 X 处理器核内的主要功能部件之一，主要功能是收集核内各部件的异常、自陷请求以及核外的 I/O、机间中断请求，完成中断响应和中断现场切换，进行中断服务程序的转移。中断部件所完成的操作包括：排空流水线内中断线程的指令、进行中断优先级仲裁、保存中断现场、产生中断向量表的入口地址、中断返回时恢复中断现场等。

X 处理器中断部件的设计目标包括：

1、实现多线程的中断处理

完成处理器核内所有线程的中断处理工作，具体包括：

（1）收集异常、错误、中断请求信号

中断部件处理的对象包括核内各部件产生的异常、错误处理、自陷请求、软件中断、定时器中断请求，以及来自核外的 I/O 设备中断和机间中断请求。中断部件收集上述中断请求，以便进行中断处理。

（2）控制排空流水线

中断部件根据中断请求的线程号，产生该线程的排空信号，控制各执行部件的流水线排空该线程的后续指令，以防止后续指令修改线程的处理器状态。

（3）中断仲裁

X 处理器中断部件要完成的中断仲裁操作包括线程仲裁和中断优先级仲裁。中断系统约定优先响应编号较小线程的中断，并优先响应同一线程高优先级的中断请求。

（4）保存和恢复中断现场

中断部件根据仲裁结果将被响应线程的处理器状态保存到中断堆栈中，并在执行中断返回指令时读出堆栈中保存的中断现场，恢复线程的处理器状态。程序数据现场保存通过更新全局寄存器指针来完成，此时会切换到新的全局寄存器组用于中断服务程序的中断处理。

（5）产生重定向指令地址

在整个中断处理和中断返回的过程中，一共包括两次指令流的重定向。一次

是从正常程序的指令流定向到中断服务程序的指令流，另一次是中断服务程序的指令流返回到正常程序的指令流。这两次重定向的指令地址都由中断部件产生，并送往指令部件进行取指。

2、实现软件中断

中断部件维护各线程的软中断向量，根据软件中断向量产生软件中断请求，并实现软件中断屏蔽。

3、实现定时器中断

X 处理器为每个线程设定了三个定时器，分别用于产生特权、超特权执行模式和系统同步的定时器中断。中断部件负责维护定时器的更新和设定，以及定时器中断请求的产生。

4.2 总体结构

根据 X 处理器中断系统的设计目标，我们设计中断部件的总体结构如图 4.1 所示：

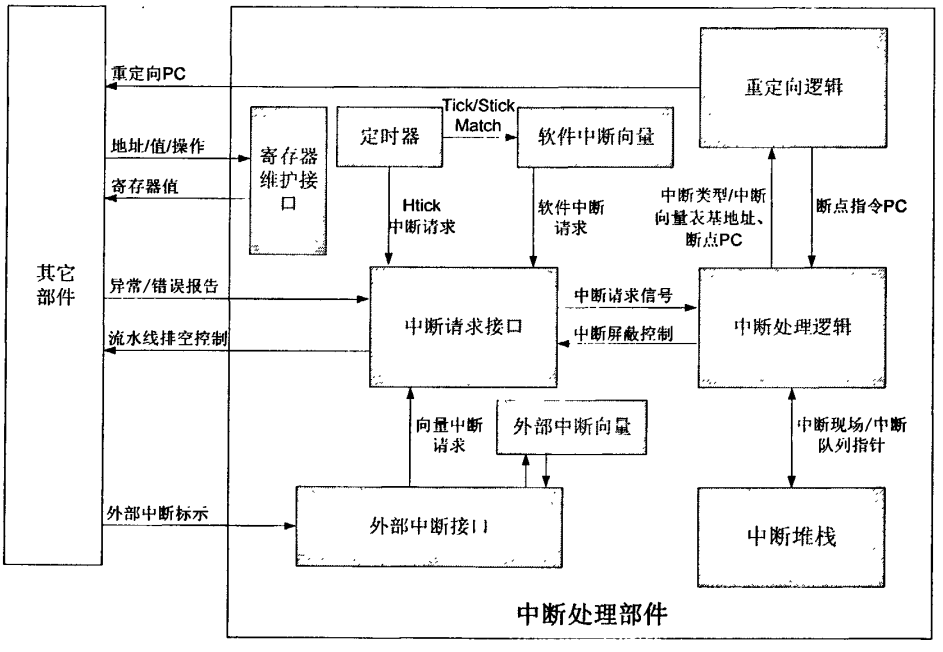


图 4.1 中断处理部件的总体结构

根据中断部件的功能，整个中断处理部件可以划分为中断处理、定时器与软中断、外部中断接口和寄存器维护接口四个功能单元：

- 中断处理单元

中断处理是中断部件主要的功能单元。它负责完成与中断处理相关的功能，包括收集中断报告、产生中断请求、控制流水线的排空、中断仲裁、保存与恢复

中断现场、产生重定向 PC 等。中断处理功能单元包括中断请求接口、中断处理逻辑、重定向逻辑、以及中断堆栈等模块。

- 定时器与软中断单元

定时器与软件中断部分维护了各线程的定时器以及软件中断向量，负责设置定时器和软件中断向量，以及产生定时器中断和软件中断请求的。该功能单元包括了定时器逻辑、软件中断向量逻辑等模块。

- 外部中断接口单元

外部中断接口单元实现了外部 I/O 与线程的 I/O 中断接口以及其它处理器核之间的机间中断接口。该单元通过维护一个统一外部中断向量来产生向量中断报告。

- 寄存器维护接口单元

寄存器维护接口单元是中断部件内体系结构寄存器的读写接口，负责对寄存器的访问地址进行译码。该部分根据访问地址和访问类型控制其它模块读写相应寄存器。

4.3 模块设计与实现

4.3.1 中断处理单元的设计与实现

中断处理单元是中断部件的主体，该功能单元内模块和模块间的接口关系如图 4.2 所示：

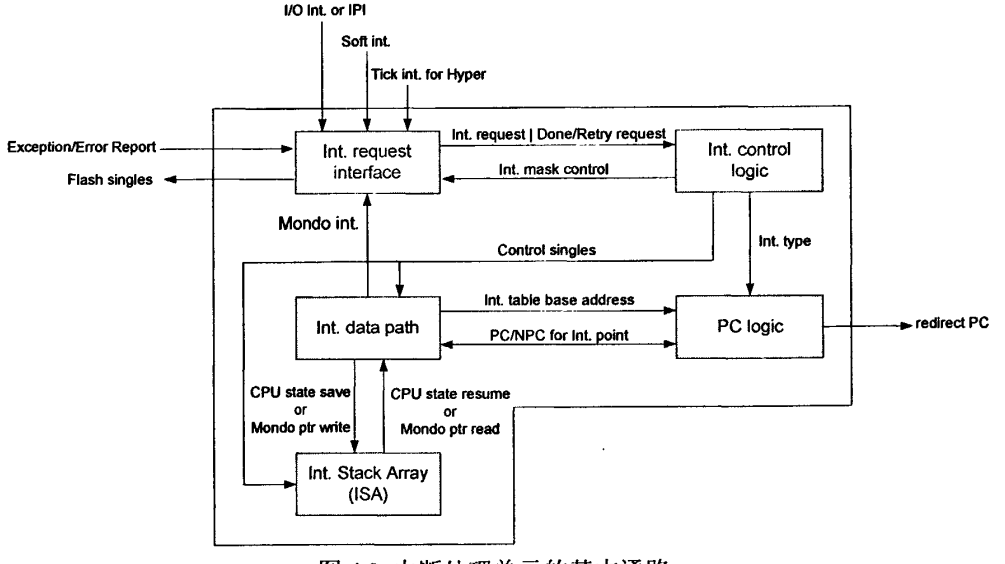


图 4.2 中断处理单元的基本通路

该通路包含的主要模块的功能如表 4.1 所示：

表 4.1 中断处理通路的基本组成

模块	功能说明
Int_ReqInterface	接收异常/错误报告和中断请求，控制流水线刷新
Int_ControlLogic	中断响应和中断返回的主要控制逻辑，它还负责控制中断数据通路和中断堆栈的读写
Int_DataPath	中断处理数据通路，维护部分中断处理相关的处理器状态寄存器
Int_PCLogic	产生中断程序入口 PC
ISA	中断堆栈，保存线程中断前的状态以及中断队列的头尾指针，该模块采用大小为 32x152-bit 的全定制的 RAM 实现

各部件异常/错误报告、定时器/软中断报告、以及外部中断报告统一到中断处理部件的中断请求接口模块（Int_ReqInterface）。中断请求接口模块将与指令流执行同步发生的异常/错误报告站存到流水线的写回站（W 站）；将与指令流异步的中断报告，如外部中断、定时器中断等插入到流水线的执行站（E 站）或 W 站进行统一处理。中断请求接口根据中断控制模块送来的中断控制信号进行中断屏蔽控制，并产生中断请求，送往中断控制逻辑。如果断请求接口接收到是中断返回请求，则该请求也会同时送往中断控制模块。

中断控制模块收到来自于各线程的所有中断请求之后，对这些中断请求进行归类，并进行线程仲裁和优先级仲裁。一旦有中断请求被选中，中断控制模块就会重新更新相应线程的中断等级，控制中断处理数据模块（Int_DataPath）和中断堆栈（中断堆栈）完成该线程处理器状态的保存和更新。中断控制模块会根据仲裁结果产生中断类型标识并更新当前中断等级，送往重定向模块（Int_PCLogic）用于产生中断程序入口指令地址。如果中断控制模块优先响应中断返回请求，则它会控制中断处理数据通路和中断堆栈完成该线程的处理器状态恢复工作。

中断数据通路负责维护核内各 CPU 状态寄存器、中断向量表基地址寄存器、以及中断堆栈的部分数据读写通路。在中断响应的过程中，该模块在中断控制模块相关信号的控制下，将中断线程的各处理器状态/控制寄存器送往中断堆栈进行保存。中断数据通路选出相应逻辑 CPU 和相应执行模式的中断向量表基地址送往重定向模块用于产生中断程序入口 PC。在中断返回的过程中，该模块还负责接收中断堆栈读出的中断现场，恢复各状态/控制寄存器，并将被中断指令的地址送往重定向模块。

重定向模块在发生中断时将被中断指令的地址传递到中断处理数据模块以保存至中断堆栈，并根据中断处理数据通路送来的中断向量表基地址和中断控制模块送来的中断类型以及中断线程当前的中断等级形成中断服务程序入口指令地址，送往指令部件。

中断堆栈由一个全定制实现的 RAM 和相关逻辑组成，用于保存各线程中断现

场和各线程的中断队列（CPU Mondo 中断队列和 Device Mondo 中断队列）的头尾指针。该模块的读写信号由中断控制模块产生，读写地址和写数据由中断处理数据通路产生。

4.3.1.1 中断请求接口的设计

中断请求接口接收并处理来自各功能部件的异常、错误报告、定时器、软中断向量、外部中断接口的向量中断请求，以及中断返回指令 Done/Retry 所产生的中断返回请求。

该模块主要包括中断请求的产生以及流水线排空信号的产生两部分功能逻辑，模块的主要通路如图 4.3 所示：

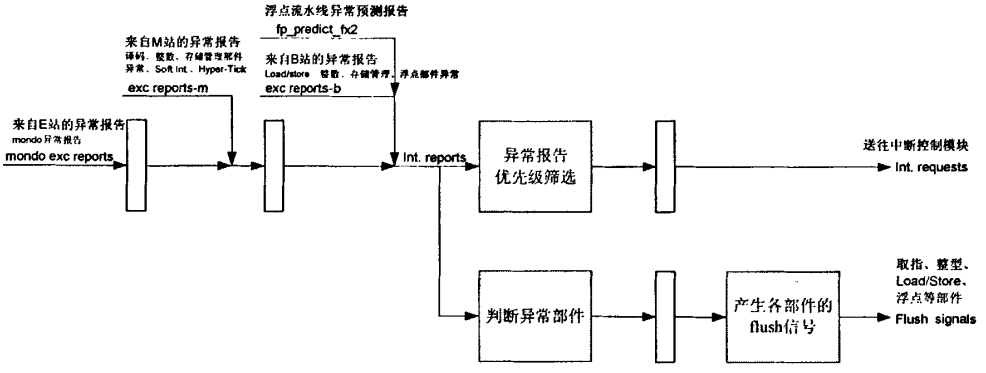


图 4.3 中断请求接口模块的主要通路

1、中断报告的收集和中断请求的产生

中断报告的收集和中断请求的产生的流水线结构如图 4.3 中的上半部分所示。整个过程可以分为以下几个阶段：

- 中断报告的收集

中断报告可以分为同步报告和异步报告两类。同步报告一般为异常/错误报告，它们来自于核内的各功能部件。这些报告一般由特定的指令引起，并与指令流执行同步产生。同步报告有可能同时来自于多个流水线站，包括存储站（M 站）、旁路站（B 站）。异步的中断报告（包括 Mondo 队列中断、软中断、定时器中断事件等），一般由异步事件产生。中断请求接口也将这些异步中断报告插入到流水线中进行统一处理。其中，Mondo 队列中断被插入到执行站（E 站）、软中断和超特权模式的定时器中断报告被插入到 M 站。中断请求接口将上述报告保存到站间寄存器中，并传递到 B 站进行统一处理。

- 异常的识别与判断

异常的识别与判断是指，不同的异常报告可能会产生同一种中断；同一异常报告在不同的处理器状态下可能会导致不同的情况（该异常产生或不产生中断）；有些部件发出了异常报告，但异常是否产生中断请求还取决于中断屏蔽控制。中

断请求接口会在 B 站对上述情况进行判断，以使在多个报告中筛选出真正会产生中断请求的那个异常报告。

- 优先级处理

异常报告处理的另外一项工作是对部分类型的异常报告进行简单的优先级处理。根据中断优先级等级，可以在发生中断嵌套时屏蔽低优先级异常报告。在这一过程中，中断请求接口将异常按优先级分成 4 组，其屏蔽信号由中断控制模块产生，分别为：(1) kill_exc_lt_6_：异常的优先级等级小于 6；(2) kill_exc_gt_6_lt_7_：异常的优先级等级大于 6 小于 7；(3) kill_exc_ge_7_lt_11_：异常的优先级等级大于等于 7 小于 11；(4) kill_exc_b_：异常的优先级等级大于 11。当某些信号有效时，相关优先级组的异常报告就会被处理，并产生中断请求。

- 产生中断请求

在经过上述处理后，模块根据没有被丢弃的异常报告来产生相应的中断请求。每种类型的中断都有独立的中断请求信号。中断请求接口模块会将这些中断请求信号和中断线程编号送往中断处理模块，进行进一步的处理。

2、控制流水线排空

中断请求接口在处理各异常、中断报告的同时，会根据这些报告判断当前各功能部件是否发生了异常，并向所有功能部件的执行流水线发出针对被中断线程的排空（Flush）信号。

中断请求接口在旁路站（B 站）通过对自各部件的异常报告信号来判断该部件是否产生异常。中断请求模块在写回站（W 站）对各部件异常判断结果进行汇总，并根据异常的来源、异常所在线程，产生各部件的刷新（flush）信号用于排空流水线中相应线程的后续指令。

中断处理部件在 M 站向各部件发出控制信号，控制排空各流水线中同一线程的指令。中断请求接口模块会在 M 站判断各部件处于该站的指令是否处于被中断线程。如果是，则排空该部件在 M 站的指令。这样就减少了中断处理部件到流水线各站的旁路，从而降低了各部件的互连复杂度。

4.3.1.2 中断控制模块的设计

中断控制模块是中断处理部件中主要的控制通路，负责进行中断响应、中断返回的控制，以及中断堆栈和中断处理数据通路访问的控制。

1、中断控制寄存器

中断控制模块实现了与中断控制相关的寄存器。每个线程都拥有独立的一套状态/控制寄存器，这些寄存器的功能如表 4.2 所示：

表 4.2 中断控制寄存器

控制寄存器	功能说明
IL[2..0]	中断等级寄存器，其值代表当前 CPU 所处的中断等级
PIL[3..0]	处理器中断优先级寄存器，用来屏蔽低于 PIL 值的软中断
GL[2..0]	全局寄存器指针寄存器，用于指出当前正在使用的全局寄存器组

中断等级寄存器根据发生的事件类型（中断响应或者中断返回）进行递增或递减；处理器中断优先级寄存器由特权寄存器写指令负责更新；全局寄存器组指针属于中断现场的一部分，在发生中断时需要保存到中断堆栈中，它在中断响应事件中由自递增逻辑负责更新，在中断返回中由中断现场恢复逻辑进行恢复。

2、中断请求的处理

中断请求处理相关逻辑主要实现保存、仲裁中断请求，产生中断类型标识等功能，其主要的逻辑通路如图 4.4 所示。为了减少关键路径长度，中断请求处理的过程被划分为两站完成。

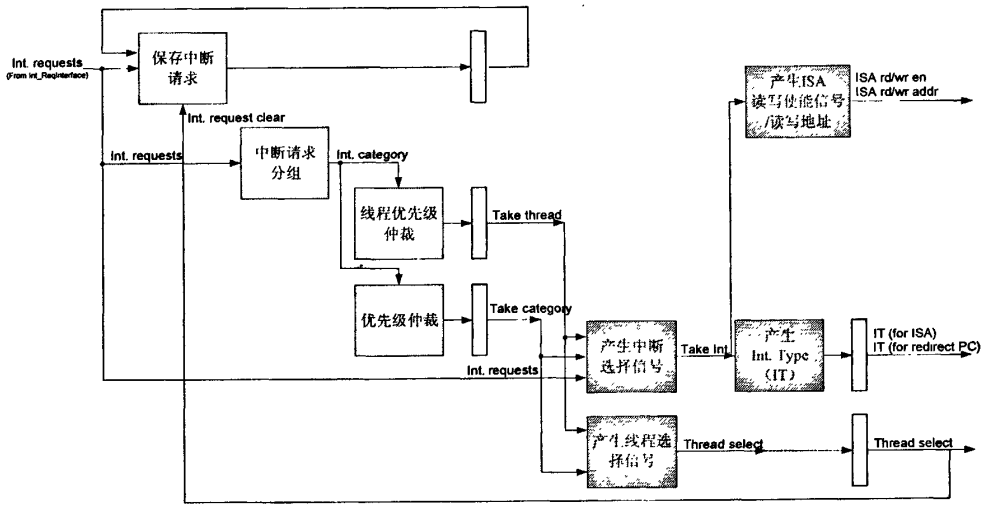


图 4.4 中断请求处理逻辑通路

1、第一站：中断仲裁

- 保存中断请求

中断控制模块保存来自中断请求接口的各类中断或中断返回请求。请求保存逻辑将各线程所有的中断请求和返回请求保存到寄存器中，一直到相应线程获得并完成中断仲裁之后才会被清除。

- 中断请求分类

中断请求分类逻辑对所有中断、中断返回请求进行分组，一共分类复位（Reset）、异步（Disrupt）、除法（Divide）异常、长延迟（Long）、浮点（FP）、中断返回（Done & Retry）、写指令 TLB、其它等 8 组。后续处理逻辑将根据各线

程有效的组别信号进行线程优先级仲裁和中断优先级仲裁。

- 线程优先级仲裁

线程优先级仲裁逻辑按照小号优先的顺序按组对来自各线程的中断请求进行线程仲裁，以浮点类型中断请求的线程仲裁为例，其仲裁逻辑如图 4.5 所示。仲裁产生的线程选择信号将用于产生中断选择信号以及中断请求的清除信号。

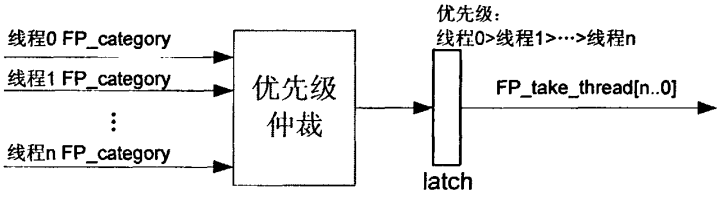


图 4.5 线程优先级仲裁示意图

- 优先级仲裁

中断优先级仲裁逻辑按组对各线程的中断请求进行组间的优先级仲裁，各类中断、中断返回请求优先响应的顺序为：复位中断请求 > 异步中断请求 > 除法异常中断请求 > 长延迟操作中断请求 > 浮点中断请求 > 中断返回请求 > 写 ITLB 中断请求 > 其它中断请求。如果某一组中断请求获得了仲裁，那么该组中断请求响应信号将会有效。

2、第二站：中断类型标识的产生

- 中断类型标识的产生

中断类型标识产生逻辑根据中断请求分组、线程仲裁与优先级仲裁的处理结果产生线程选择信号和中断类型标识。线程选择信号用于指出哪一个线程的中断得到了响应。中断类型标识被送往中断处理数据通路用于更新中断类型寄存器，并作为中断现场保存到中断堆栈当中。同时，中断类型标识还被送往重定向模块用于产生中断服务程序入口指令地址。

- 中断堆栈读写信号和读写地址的产生

在中断响应过程中，中断控制模块产生中断堆栈的写使能信号和写地址用于中断现场的保存；在中断返回过程中，中断控制模块产生中断堆栈的读使能信号和读地址用于读出原来保存的中断现场，用以被中断线程处理器状态的恢复。

4.3.1.3 重定向模块的设计

重定向模块负责在中断响应和返回的过程产生用于指令流重定向的指令地址。这个指令地址可能是中断程序入口的指令地址或者被中断指令的 PC 或 NPC，具体取决于当前处理事件是中断响应还是中断返回。重定向模块的另外两个作用是在中断发生时获取指令部件中发生中断指令的 PC 值用于保存，以及浮点异常预测时将被预测要发生异常的指令地址发送到取指部件中进行重新取指。

1、重定向指令 PC 的产生

产生重定向指令 PC 的相关通路如图 4.6 所示：

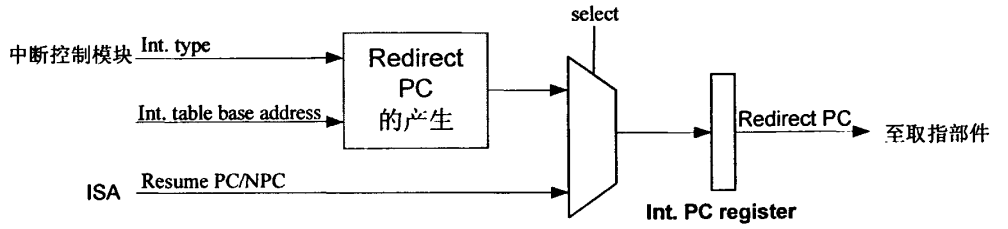
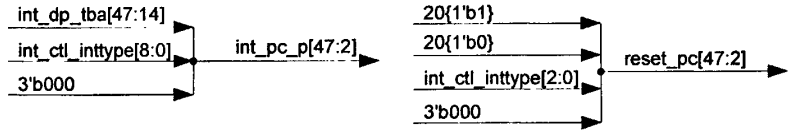


图 4.6 重定向 PC 产生逻辑通路

不同的处理器运行模式有不同的中断向量表，不同中断向量表入口地址的形成方式也不同，重定向逻辑会根据中断发生后的线程所需切换到的运行模式（即处理该中断的 CPU 执行模式）选择中断向量入口地址的形成方式。中断向量表入口地址的形成逻辑如图 4.7 所示：



(a) (超级)特权模式中断向量表入口指令地址　(b) 复位/调试模式中断向量表入口指令地址

图 4.7 中断向量表入口地址的形成

2、指令流同步控制

重定向模块为每个线程维护了一个用于与当前流水线 W、B 站的指令同步的 PC 和 NPC 寄存器。一旦发生中断，相关逻辑就会将该 PC/NPC 送往中断处理数据模块，通过该模块相关数据通路保存至中断堆栈。

通常作为中断现场被保存的中断指令 PC 来自于流水线，这么做的原因是中断处理部件不能在指令流发生重定向时得知指令流的变化。这种设计需要设计专门的指令流水线来保存正在执行指令的 PC，带来了额外的硬件开销，并且增大了部件之间互连的复杂度。为了解决这个问题，我们在 X 处理器中断部件中进行了指令流同步控制的设计(如图 4.8)[37]。重定向模块为每个 CPU 维护了一个副本 PC 寄存器，并采用一个专门的指令流控制逻辑来维护这些 PC 寄存器，使其始终与执行流水线中的指令流同步，并指向流水线 W 站的指令。指令流控制逻辑的作用就是在指令顺序执行时计算后续指令的地址、在分支跳转及中断处理引起指令流重定向的同时负责更改副本 PC 寄存器中的指令流。

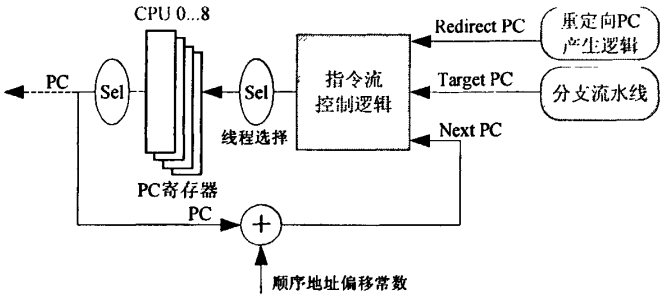


图 4.8 中断部件中的指令流同步控制[37]

3、浮点异常预测时的指令重取

整数和浮点流水线的不等长设计会给精确中断带来潜在的异常冲突。紧跟浮点指令的之后的整数指令可能会在浮点指令检测到异常之前就更改处理器状态，从而导致浮点流水线无法实现精确中断。

在 X 处理器中，我们采用浮点异常预测机制实现精确中断，浮点部件在流水线的某一站对浮点指令进行一次快速、精确的预测，来判断一个浮点操作是否会引发异常。浮点的异常预测在 M 站完成，并在 B 站将预测结果报告给中断处理部件。如果接收到的预测报告指出被预测的指令将会引发异常，那么中断处理部件就会向各执行流水线发出排空信号，清空同一线程所有的后续指令。而被预测的浮点指令将会继续执行，直到在 FW 站被检测到真正发生了异常，再次报告给中断处理单元，中断处理单元会产生一次中断（图 4.9）[37]。

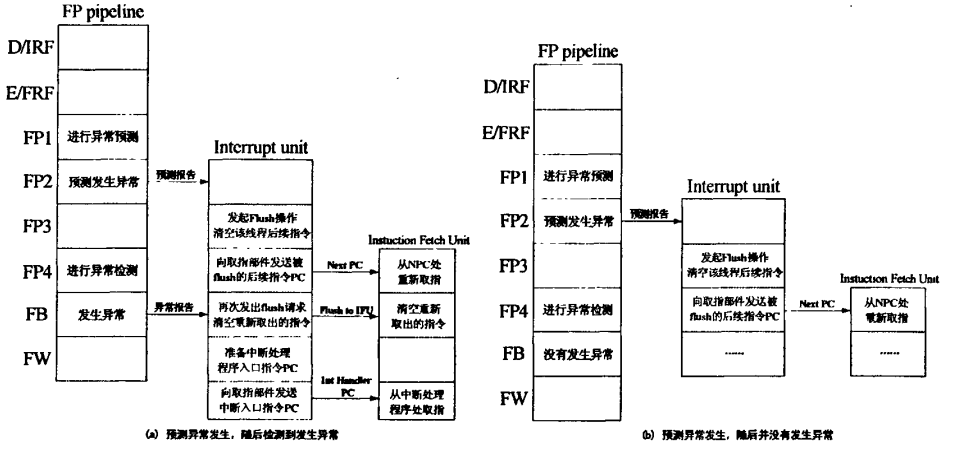


图 4.9 带有指令重取机制的浮点异常预测 [37]

在预测到浮点异常时中断控制逻辑会控制流水线对同一线程的后续指令进行排空，从而消除流水线不等长所带来的潜在异常冲突。同时，为了减少被预测线程的性能损失（当被预测的异常没有发生时，被预测线程的后续指令已被排空）我们加入了指令重取机制，使得浮点流水线在检测到异常并排空相关线程的同时，重定向逻辑就立即向取指部件发送被排空线程的后续指令 PC。这样，取指部件就

可以在被预测指令继续执行的同时，尽可能早的对该线程重新取指。如果被预测的指令实际没有发生异常，发射部件就可以直接发射被排空的后续指令，从而隐藏了重新取指的时间开销。如果浮点流水线的异常预测是正确的，那么只需要由中断控制逻辑再次向取指部件发出排空请求，清空重新取出的指令，并由重定向逻辑重新送出中断处理程序的入口指令 PC，使得该线程进入中断处理程序当中。

4.3.1.4 中断处理数据模块的设计

中断处理数据模块包含中断处理部件主要的数据通路，它维护了各线程中断处理相关的大部分处理器状态/控制寄存器，实现了访问中断堆栈的数据接口和 I/O、机间队列中断的硬件检测机制。

1、中断处理相关的处理器状态/控制寄存器

中断处理数据模块为每个线程维护的中断处理相关的寄存器如表 4.3 所示。该模块实现了这些寄存器的更新通路以及读通路。

表 4.3 中断控制寄存器

控制寄存器	功能说明
PSTATE	特权状态寄存器，用于 CPU 运行的控制
HPSTATE	超特权状态寄存器，超特权模式下 CPU 运行的额外控制
IT[8:0]	中断类型寄存器，存放代表获得中断仲裁的中断类型标识
IPC[47:2]	中断指令 PC，保存引起中断的指令地址
INPC[47:2]	中断指令 NPC，保存引起中断的指令之后一条指令的地址
IBA[47:15]	特权模式中断向量基地址寄存器，用于特权模式的中断处理
HIBA[47:14]	超特权模式中断向量基地址寄存器，用于该模式的中断处理

2、中断堆栈的读写接口

中断处理数据通路还实现了中断堆栈读写访问接口。该模块在写中断堆栈操作时按约定的数据结构组织好要写入中断堆栈的数据；在读访问时将从中断堆栈中读出的数据进行缓冲并分离各数据域，用于中断现场的恢复或者进行虚拟化外部队列中断的检测。

中断处理数据模块对中断堆栈的操作是一种“先读后写”的访问，即在相关逻辑写某一项之前，会先将该项读出，更新其中部分域后再将该项写回到中断堆栈中。中断处理数据模块读写中断堆栈的情况如下：

- 中断的读操作

读出中断堆栈的一项，用于：

（1）读指令读出中断堆栈中保存的中断现场，用于中断服务；

（2）恢复中断现场；

（3）读出其中 Mondo 队列的头/尾指针进行 Mondo 队列中断的检测；

- 中断堆栈的写操作

写中断堆栈的操作分为以下三种情况：

（1）写指令更新某一 CPU 的 Mondo 中断队列的头/尾指针；

（2）写指令更新保存在其中某个中断现场的某一些域；

（3）发生中断时，将中断现场相关的状态/控制寄存器保存到中断堆栈中。

3、Mondo 队列中断的检测

根据中断堆栈写操作的过程，模块在更新某线程的一个 Mondo 队列指针时，会将该线程队列的头尾指针一起读出。更新后指针在写回中断堆栈之前会由相应的 Mondo 队列中断检测逻辑进行检测，判断是否产生 Mondo 队列中断报告。相应的判断逻辑如图 4.10 所示：

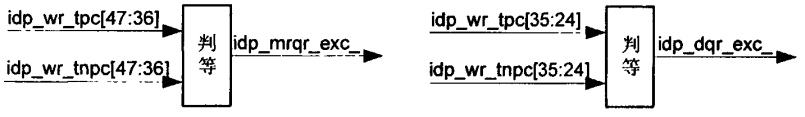


图 4.10 Mondo 队列中断的判断

信号 idp_mrqr_exc_指出了 cpu_mondo 队列是否为空；信号 idp_dqr_exc_指出了 dev_mondo 队列是否为空；具体是哪个 CPU 的队列取决于当前中断堆栈的访问地址。

4.3.1.5 中断堆栈数据结构的组织

中断堆栈是一个大小为 64x152 的 RAM，采用全定制的方式实现。中断堆栈的作用包括以下两个：（1）保存每个线程的中断现场；（2）保存每个线程的虚拟化中断队列的头尾指针。考虑到处理器最大支持 6 级嵌套中断嵌套，所以中断堆栈的前 48 项用于保存 8 个线程的中断现场状态，之后 8 项用于保存 Mondo 中断队列指针，最后的 8 项未被使用。

在发生中断时需要保存到中断堆栈的寄存器包括——PSTATE、HPSTATE、IT、IPC、INPC、GL、IL 以及来自执行部件和 Load/Store 部件的相应状态寄存器，这些寄存器被中断处理数据模块按照与中断堆栈约定组成 152-bit 的写数据写入到中断堆栈中。属于中断现场的各寄存器以及这些寄存器在保存至中断堆栈时的数据组织见表 4.4：

表 4.4 中断堆栈数据结构组织

中断现场的数据结构组织		
位置	相关寄存器	描述
151：136	—	ecc 校验码
135	—	ipc 虚拟地址越界标示
134	—	inpc 虚拟地址越界标示
133	—	inpc 非顺序标示
132：131	GL	全局寄存器组指针
130：123	CCR	CCR 寄存器，来自于整数部件
122：115	ASI	ASI 寄存器，来自于 Load/Store 部件
114：104	PSTATE/HPSTATE	PSTATE、HPSTATE 寄存器的所有有效域
103：101	CWP	CWP 寄存器，来自于整数部件
100：92	IT	中断类型寄存器
91：46	IPC	发生中断时正在执行的指令 PC
45：0	INPC	发生中断时指令的 NPC
中断队列指针的数据结构组织		
151：92	—	—
91：80	cpu_mondo_head	CPU Mondo 队列头指针
79：68	dev_mondo_head	Device Mondo 队列头指针
67：46	—	—
45：34	cpu_mondo_tail	CPU Mondo 队列尾指针
33：22	dev_mondo_tail	Device Mondo 队列尾指针

4.3.2 外部中断接口单元的设计与实现

外部中断接口实现了外部 I/O 设备中断以及机间中断的接收逻辑，并维护了一个外部中断向量，以记录 I/O 设备或其它处理器核送来的中断报告。

1、外部中断向量的维护

外部中断接口为每一个逻辑维护了一个长度为 64-bit 的外部中断向量寄存（IVR）用于记录该线程收到的外部 I/O 中断报告和机间中断报告。处理器核接口部件对来自外部的中断报文进行预处理后，将报文的类型和中断的标识送到中断处理部件的外部中断接口。

外部中断向量 IVR 的维护通路如图 4.11 所示：

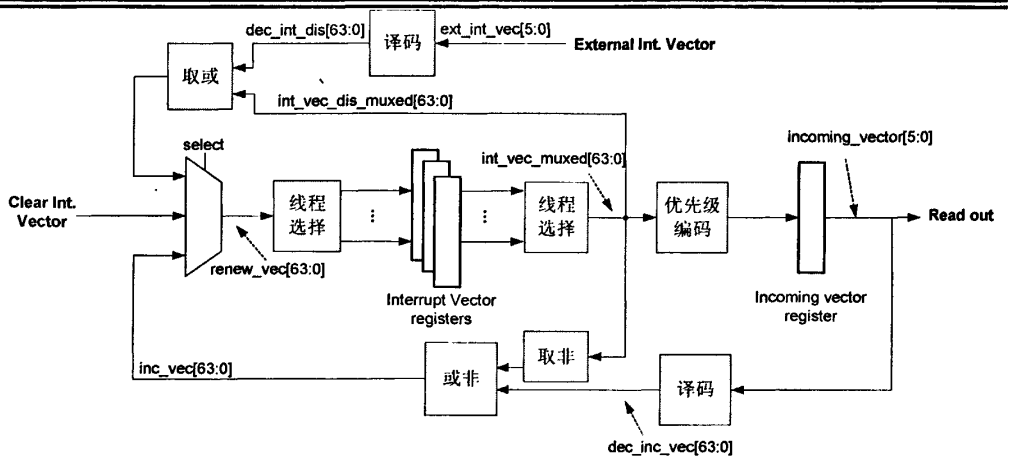


图 4.11 外部中断向量及其维护通路

IVR 寄存器的每一位都代表了一类外部中断，中断向量本身隐含了优先级，高位所代表的外部中断类型比低位代表的中断类型具有更高的中断优先级。当某一线程的 IVR 寄存器其中的一位被置 1 时，表示该线程收到了一个中断请求，中断请求的中断号即为该位的位编号。IVR 的更新将导致向量中断（中断类型：0x60）的产生。外部中断接口相关逻辑会对 IVR 进行优先级编码，选出具有最高优先级的中断请求，将其中断号保存到中断标识寄存器（IncVR）中，供向量中断服务程序读出并进行处理。

更新某一线程的 IVR 被有以下三种情况：

（1）一个新的 I/O/机间中断请求到来时，更新逻辑将对 6-bit 的外部中断标识进行译码，并运用或运算将该中断请求记录到 IVR 中；

（2）指令写 IVR 寄存器，通过掩码的方式清除 IVR 的部分位；

（3）向量中断服务程序读出 IncVR，同时硬件会清除 IVR 中代表 IncVR 内中断类型的相应位。

上述三种情况更新逻辑使用的逻辑表达式分别为：

（1） IVR ＝ IVR | dec_int_dis[63:0]

（2） IVR ＝ (～write_data[63:0]) & IVR

（3） IVR ＝ ～((～IVR) | dec_inc_vec[63:0])

其中，dec_int_dis[63:0]为外部中断标识的译码结果，write_data[63:0]为要更新 IVR 的掩码，dec_inc_vec[63:0]为最高优先级外部中断号的译码结果。

2、向量中断的检测

向量中断的检测通路如图 4.12 所示：

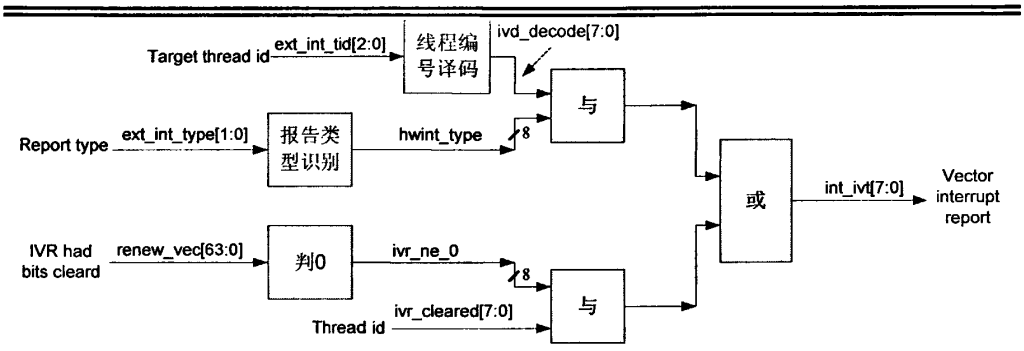


图 4.12 向量中断的检测

向量中断的检测逻辑会在以下两个时机可能产生向量中断报告：

（1）有新的外部报告到来时。此时相关逻辑会对报告类型进行识别，如果是外部中断报告，则会根据报告的目标线程的编号产生该线程的向量中断报告；

（2）相应线程的 IVR 被更新时。检测逻辑会判断更新后的 IVR 是否为空，如果不是，则说明 IVR 中还有存在要处理的外部中断报告，则检测逻辑会继续产生该线程的向量中断报告。

4.3.3 其它功能单元的设计与实现

中断部件还包括定时器和软中断单元、寄存器管理接口单元等。本小节主要对定时器模块和软中断模块的设计做简要的介绍。

4.3.3.1 定时器的设计

为了满足处理器不同运行模式软件的需要，X 处理器中断部件为每个 CPU 实现了三套定时器机——用户定时器、系统定时器、超特权定时器。分别用于用户模式、特权模式和超特权模式下的定时器中断。

三类定时器使用同一个 63 位的计数器。由于同一时间只能有一个逻辑 CPU 占用流水线，所以该计数器也被核内 CPU 所共享。计数器的格式如图 4.13 所示：

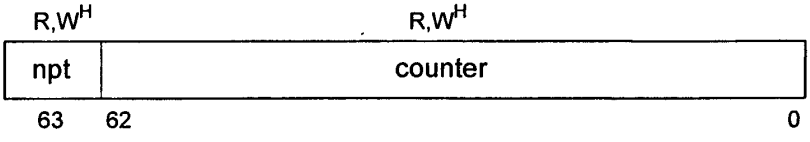


图 4.13 定时器计数器格式

计数器的更新通路如图 4.14 所示。指令写计数器的操作只能在超特权执行模式下进行，特权和超特权模式指令都可以读取该寄存器的值。但是，为了保证安全性，用户模式是否能够读取计数器的值取决于超特权软件对 npt 域的设置。

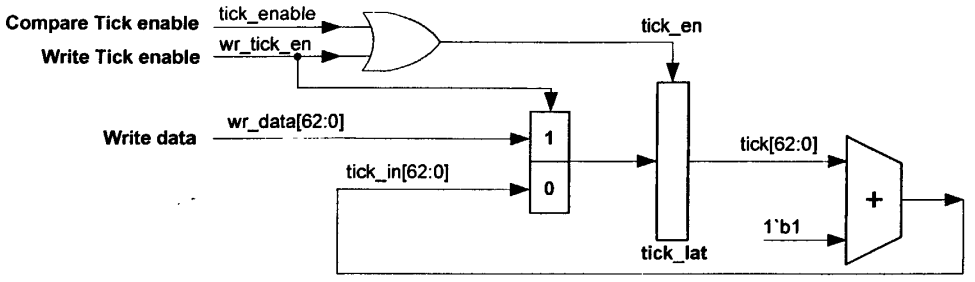


图 4.14 定时器的计数器通路

CPU 为每一类定时器分别设置了不同的比较寄存器分别用于用户、特权和超特权模式定时器的比较。这三个寄存器的格式如图 4.15 所示，其中 int_dis 域为定时器中断使能域。

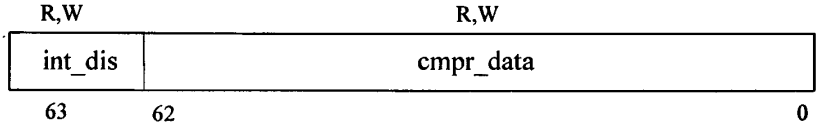


图 4.15 定时器比较寄存器格式

不同的比较寄存器的 cmpr_data 域由不同运行模式的代码进行设置。如果定时器中断使能有效，那么当计数器与某一比较寄存器预先设定的值相等时就会产生相应类型的定时器中断报告。定时器中断的检测逻辑如图 4.16 所示：

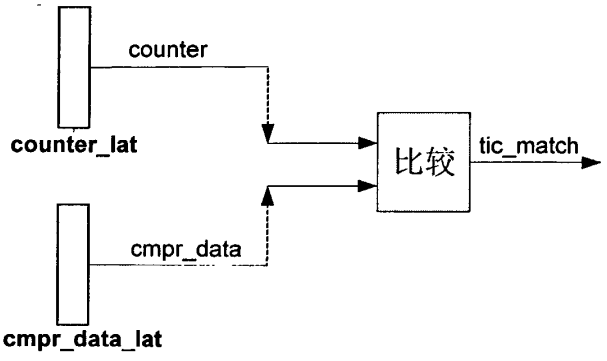


图 4.16 定时器与 CMPR 的比较通路

由于 TICK 的计数器被核内 8 个线程共享，因此 TICK 寄存器每个周期只能与一个线程的一个比较寄存器进行比较。同一个定时器比较寄存器参与比较的时间间隔为 32 个周期，因此实际上比较寄存器的最低 5 位不参与计数器的比较。

定时器中断的产生逻辑根据不同类型定时器匹配信号 tic_match、stic_match、htic_match 以及相应的定时器中断使能信号判断是否产生定时器中断。用户定时器中断和系统定时器中断通过设置软中断向量中的不同的向量位来产生，而超特权定时器中断则通过设置超特权定时器中断寄存器来产生，该寄存器只有一位有效位，超特权定时器匹配时，硬件会设置该寄存器的值。

4.3.3.2 软中断向量的设计

软中断向量模块为核内的每个线程维护了一个具有 15 个中断等级的软中断向量。特权和超特权模式的指令通过设置该向量不同的向量位来产生不同等级的软中断。另外，用户定和系统定时器也通过软中断向量来实现定时器中断。

软件中断向量寄存器（SINT）是为了能够产生软件中断（interrupt_level_*n* 中断）而特别设置的。寄存器各个域的含义如表 4.5 所示。软件中断向量还包括两个辅助的伪寄存器：SINT_clear 和 SINT_set，分别用于设定和清除 SINT 相应类型的软件中断位。如果指令向 SINT_set 写入数据则会将写入的数据与 SINT 的数据进行一个 OR 操作，相应位置位，而向 SINT_clear 写入将会清除相应位。

表 4.5 软件中断向量寄存器包含的有效域

数据域	位置	描述
TM	0	tic_match，定时器计数器与用户定时器比较寄存器的 cmpr_data 域相等时置位，从而产生 interrupt_level_14 中断
INT_LEVEL	15:1	向第 n 位写入会产生 interrupt_level_n+1 的中断
SM	16	stic_match，定时器计数器与系统定时器比较寄存器的 cmpr_data 相等时置位，从而产生 nterrupt_level_14 中断

软中断的产生受到 PIL 寄存器和 PSTATE 的中断屏蔽域的控制，PIL 寄存器用来屏蔽优先级小于 PIL 值的 interrutp_levle_n 中断。PSTATE 的中断屏蔽域）用于屏蔽所有软件中断和外部中断。

4.4 本章小结

本章根据 X 处理器的中断模型设计实现了 X 处理器核内的中断部件。X 处理中断处理部件是 X 处理器核中负责中断处理的重要部件。该部件主要包括中断处理单元、外部中断接口单元、定时器与软中断单元等几大功能单元。设计通过中断过程的分站处理器来减少关键路径长度；利用指令流同步设计来减少硬件开销；利用指令重取的浮点异常预测来实现高性能的精确中断。

第五章 X 处理器中断部件的测试与验证

随着集成电路进入到千万级门的规模，集成电路设计的验证已经成为芯片设计的瓶颈[38]。研究表明，74%的芯片失败的原因在于设计存在功能上错误[39]。验证是制造正确芯片所必须的过程，在集成电路整体设计流程中也比其它工作占用了更多的时间和资源。因此，必须有高效的验证方法将设计中出错可能性降低到最小，确保投片成功。本章将重点论述中断部件模拟验证的方案，并给出模拟验证和逻辑综合结果。

5.1 模拟验证

5.1.1 验证方法和验证方案

功能验证是处理器设计验证一个重要方面。目前的功能验证方法主要包括模拟验证、仿真验证和形式验证。其中，模拟验证是将激励信号施加于设计，在观察输出结果的同时进行计算，并判断计算的结果是否与预期一致；仿真验证的原理与模拟验证类似，它将模拟验证中的激励生成、结果监视和覆盖率判断集成为测试基准，通过 FPGA 等硬件手段实现；形式验证则是使用某种语言和逻辑构造系统的数据模型，运用严格的数学推理来证明设计的正确性。

模拟验证是较传统的验证方法。对于对芯片/模块级设计而言，初始的验证方式均采用功能模拟的方式进行。模拟验证的目的包括：验证设计描述的正确性、判断设计的逻辑功能是否符合设计要求、判断设计有无违反设计规则等[40]。

一个简单的模拟验证模型如图 5.1 所示：

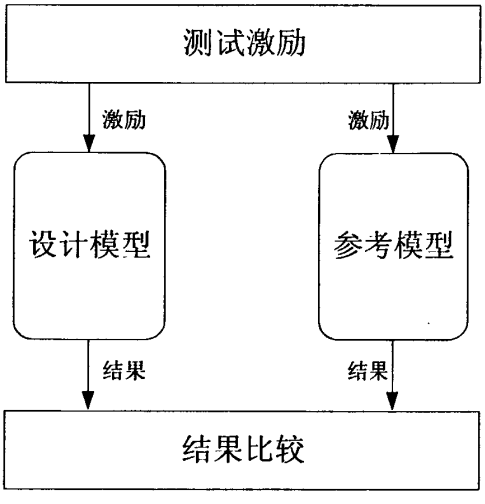


图 5.1 模拟验证的基本模型

从过程上来看，模拟验证的过程包括以下三个方面：编写测试激励、模拟运行、结果比较三个步骤。验证平台将编写好的测试激励输入被验证设计的同时，还设立了一个用于对比的参考模型。验证平台对两个模型的输出结果进行比较，如果结果不相同，则表示设计模型中可能存在错误。输入的测试激励可以用 HDL 语言行为级描述，也可以用 C 或 C++等高级编程语言实现。虽然模拟验证的正确性并不能证明设计是完全正确的，但通过标准测试向量的测试，各种边界、特殊数据和有选择的随机数据的测试，可以发现设计中存在的绝大多数问题。

X 处理器中断部件的模拟验证采用的是 down-top 的验证策略。根据被验证设计规模的大小，整个模拟验证过程分为：模块级验证、部件级验证和系统级验证。各阶段的工作和作用分别如下：

- 模块级模拟验证：验证中断部件中各逻辑模块功能的正确性，保证所设计的模块准确实现所要求的功能；
- 单元级验证：将中断部件中完成同一功能的多个基本模块按照一定的连接关系组成更高层次的功能单元，验证各功能单元执行相应操作的正确性；
- 系统级验证：将各功能单元组成中断部件并置于整个处理器设计模型中，测试中断部件与其它部件连接是否正确，以及系统各部件运行是否正确。

5.1.2 各层次的模拟验证

5.1.2.1 模块级验证

1、验证平台

模块级验证平台的结构如图 5.2 所示，验证平台由 Verilog-HDL 语言描述，包含了用于测试设计模块的测试激励以及用于对比被测模块输出结果的预期结果。模块级验证的测试激励采用手写生成，测试激励加载在被测的模块上。测试平台将被测模块输出信号的结果与预期的结果进行比较，并将比较结果输出。模块级的功能验证通过观察被测模块的内部信号、输入输出信号的波形或者观察比较结果的方式来完成。

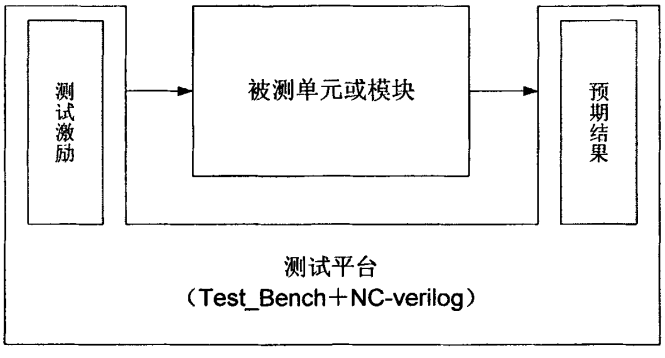


图 5.2 模块级测试平台

2、验证内容

模块级验证主要对中断部件各功能单元的子模块进行测试和验证。在验证模块时，我们根据具体需要编写每一组测试激励，这些激励信号按照约定的时序产生并输入到设计模块中。由于中断部件大部分模块都属于控制逻辑，接口信号的类型和个数比较复杂，所以中断部件模块级验证的测试激励基本用手工方法生成，每一组激励用于测试模块的一种功能。

中断部件的主要模块的验证内容包括：

- 中断请求接口模块的验证

中断请求接口模块的验证目标为验证该模块收集和产生中断请求信号的功能以及控制流水线排空的功能是否正确。测试激励按照约定的时序不断向模块输入各类异常、错误信号以及一些异步中断信号，并观察相应的中断请求信号是否按照约定的时序输出，以及用于各部件流水线排空的信号是否正常产生。

- 中断控制模块的验证

中断控制模块的验证主要包括三个方面：相关处理器控制/状态寄存器的读写是否正常；中断处理功能是否正确；保存中断现场、恢复中断现场的相关控制信号产生以及存储体的读写地址是否正确。我们对模块的各内部寄存器施加读写使能信号和写值来判断寄存器维护通路的设计是否正确；通过向模块同时输入各线程的各类中断请求信号的组合判断模块线程仲裁和中断优先级仲裁功能是否正确，以及是否能在约定的时间产生正确的中断类型标识；通过输入特定的中断请求和中断返回请求判断模块对中断现场和中断现场恢复过程的控制是否正确。

- 重定向模块的验证

通过输入特定的中断类型标识以及中断向量表基地址来验证重定向指令 PC 的产生是否正确，通过外部激励模拟中断保存和中断返回的过程验证中断点处指令 PC 是否能够正确保存和恢复，通过输入浮点异常预测报告，判断需要重取指令的 PC 是否能够按约定的时序产生。

- 外部中断接口模块的验证

测试平台按一定的时序输入不同的外部中断报文，判断外部中断向量的更新是否正确，以及该向量是否能够正常产生向量中断；Test_bench 模拟读 IncVR 寄存器，判断 IncVR 读出的值是否与预期相同，并验证 INR 寄存器的更新是否正确；此外，测试平台还模拟置、清 IVR 寄存器，以验证相关读写通路是否正确。

- 其它模块的功能验证，包括中断数据处理通路、定时器模块、软中断模块以及寄存器维护接口模块的功能验证等。

3、验证结果

以外部中断接口模块的验证为例说明模块级验证的结果判断。图 5.3 所示为外

部中断接口中断向量产生功能的一项测试。在第 1 个周期，测试激励向线程 0 的接口输入外部中断报文类型和有效信号，以及报文中包含的中断标识。通过波形可以看出，第 2 个周期外部中断接口模块根据报文类型识别了外部中断标识，并在第 3 个周期产生向量中断报告、在第 4 第 5 周期分别正确更新了中断向量 IVR 和最高优先级中断标识 Inc_VR 寄存器。波形的结果符合外部中断接口模块预期的功能和时序要求。

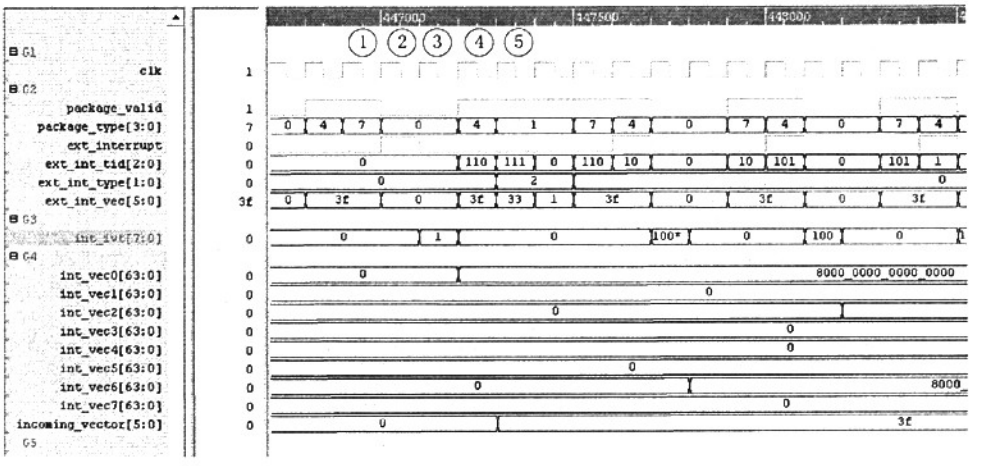


图 5.3 外部中断接口模块测试波形

5.1.2.2 单元级验证

单元级验证将中断处理模块中各子模块按功能和连接关系组成更大规模的功能单元进行正确性验证。单元级验证的目的包括以下两个：（1）验证各个单元功能的正确性；（2）验证各单元的互连接口及时序是否正确。中断部件单元级的验证以各类事务为单位进行。

1、单元级验证的内容

单元级验证平台的构成与模块级验证的验证平台相似。其测试激励以任务的形式来描述，每一个测试任务包含一个或多个测试输入，用于验证功能单元某一种事务的操作或某一种功能。根据中断部件功能单元的划分，中断处理部件的单元级验证主要分为以下几个部分：

- 中断部件内部寄存器读写的验证

测试模型由寄存器管理接口与实现体系结构寄存器的各子模块构成。外部激励模拟 X 处理器执行流水线的行为将读写地址和读写控制以及写数据输入到被测模型。测试平台按照一定顺序先写后读中断处理部件中各模块的寄存器，并判断读出的数据是否与写入的数据一致。

- 中断处理单元的验证

将中断处理接口、中断控制逻辑、中断数据通路以及重定向模块组成被测模

型。测试激励采用任务的形式描述，首先对中断部件中的各状态/控制寄存器进行初始化，接着通过输入不同的异常、中断报告组成的激励，验证中断处理流水线在中断响应和返回的过程中是否能按照预期的时序产生重定向 PC。

为了验证中断优先级仲裁的准确性、必须使用大量的测试激励组合来完成验证。各类异常、中断报告的信号激励采用随机的策略生成、test_bench 在将激励输入到被测模型的同时，也采用行为级描述算法计算出优先得到仲裁的中断类型的预期重定向 PC，并对结果进行对比。另外，中断堆栈的访问行为也采用行为级描述进行模拟。

- 定时器中断和软件中断单元的验证

设置定时器比较值，验证定时器模块是否能产生定时器中断报告，以及中断处理部件是否能产生正确的中断类型；设置软件中断寄存器，验证中断处理部件能否正确产生软件中断标识。

- 外部中断接口单元的验证

通过模拟核接口逻辑向中断处理部件的外部中断接口发送外部中断报文内容，判断中断处理通路能否正确产生向量中断。通过设置 Mondo 队列的尾指针，判断中断处理通路能否正确产生 Mondo 中断标识。单元级的验证的结果由 test_bench 通过调用系统函数输出，部件级的验证主要通过观察验证平台输出的验证结果报告来判断验证是否完成。

2、验证结果

以定时器中断和软件中断的单元级验证为例来说明 X 处理器中断部件单元级验证的过程和验证结果。

对于超特权模式的定时器中断的操作，我们验证了以下事务：

表 5.1 超特权模式定时器相关测试

测试序号	测试内容
1	超特权定时器中断的产生
2	定时器计数器的读访问
3	定时器计数器的写访问

上述测试的测试结果的波形分别如图 5.4、图 5.5 和图 5.6 所示：

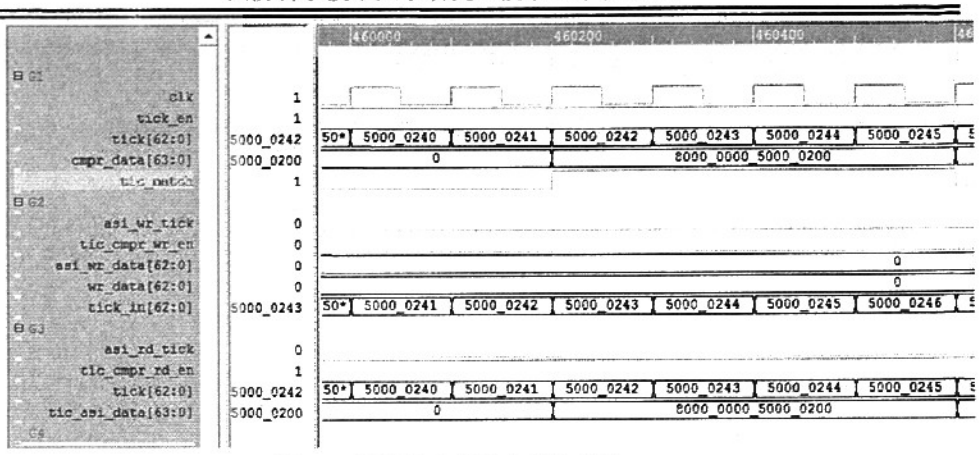


图 5.4 超特权定时器中断的产生

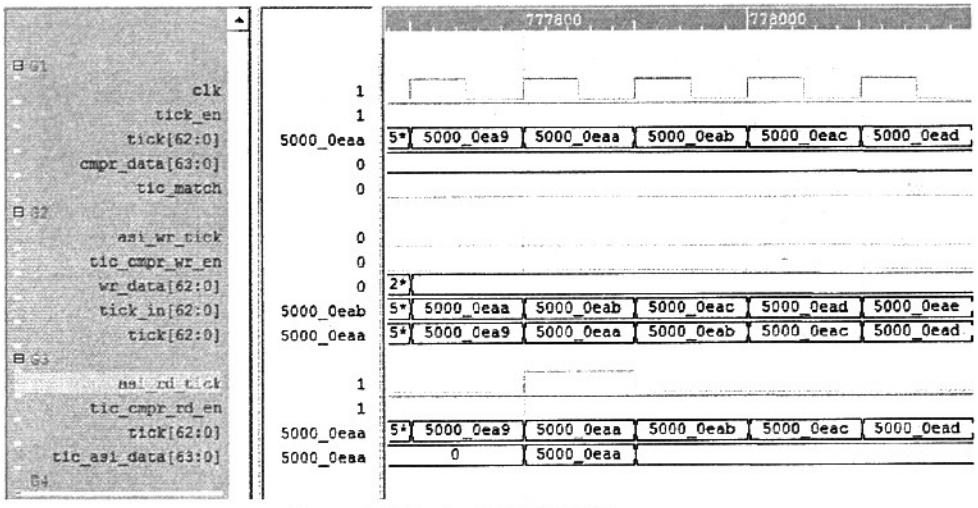


图 5.5 超特权定时器的读访问

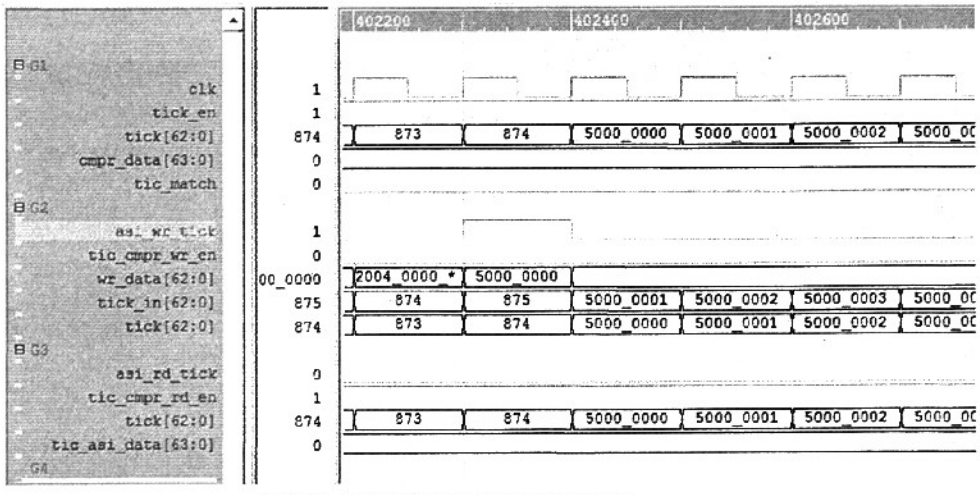


图 5.6 超特权定时器中断的写访问

对软件中断的相关操作，我们验证了以下事务：

表 5.2 软件中断向量的相关测试

测试序号	测试内容
1	产生不同等级的软件中断
2	读软件中断向量寄存器
3	写软件中断向量寄存器
4	设置软件中断向量位
5	清软件中断向量位

验证波形如图 5.7、图 5.8、图 5.9、图 5.10 和图 5.11 所示：

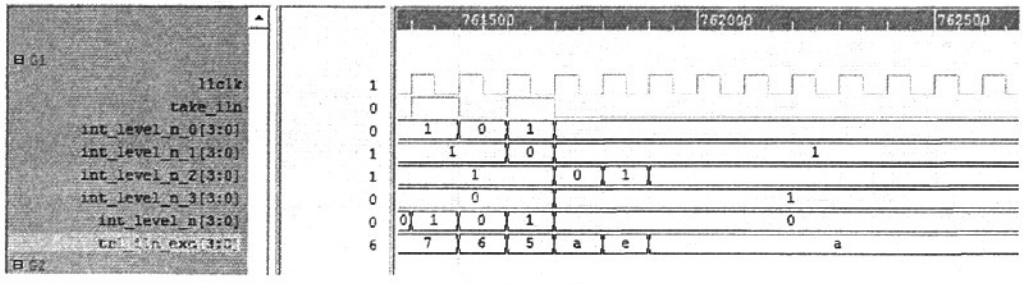


图 5.7 软件中断的产生

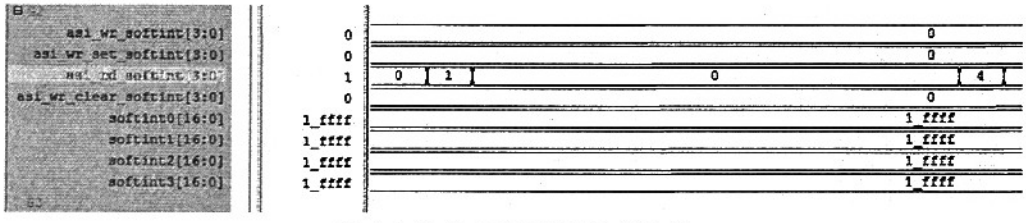


图 5.8 软件中断寄存器的读访问

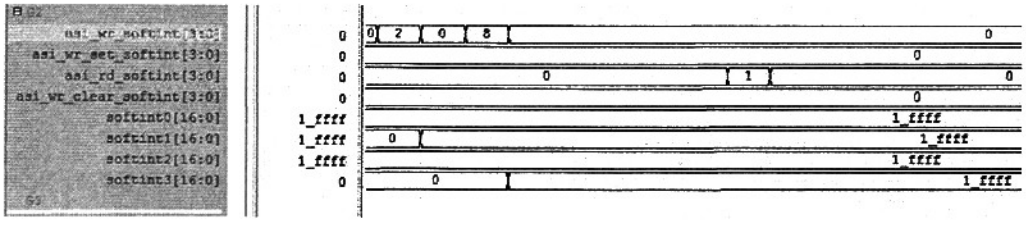


图 5.9 软件中断寄存器的写访问

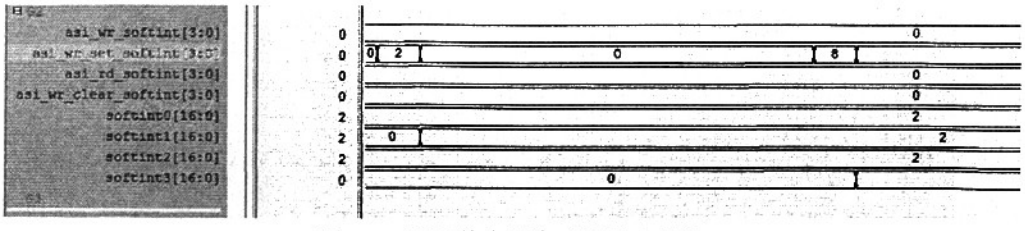


图 5.10 置软件中断寄存器的向量位

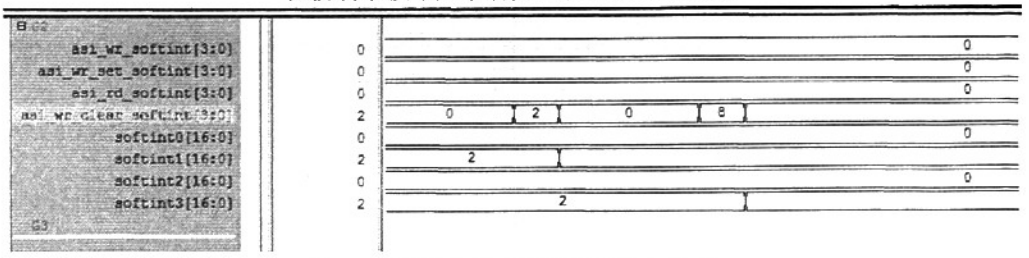


图 5.11 清除软件中断寄存器的向量位

5.1.2.3 系统级验证

系统级验证的目的是检验指令执行的正确性，并进一步验证各功能部件实现功能的正确性。

1、验证平台

X 处理器的系统级验证基于的验证平台如图 5.12 所示：。

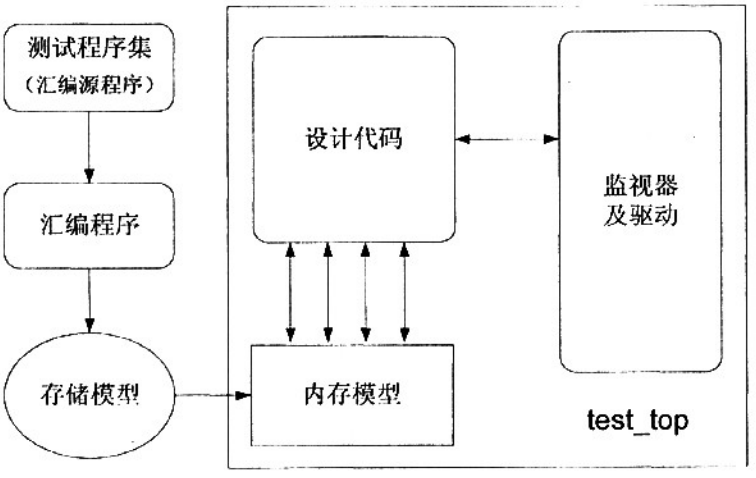


图 5.12 X 处理器系统级验证平台

从结构上看，X 处理器系统级验证平台包含以下几个部分：

- 处理器设计代码
- 验证代码和测试激励代码
- 控制模拟、测试验证的脚本和建立测试激励的工具

X 处理器系统级验证平台利用控制模拟、测试验证的脚本和建立测试激励的工具将 X 处理器全芯片的设计代码、验证代码及测试代码和 EDA 工具整合成一个完整的运行系统。测试激励主要由汇编语言书写，通过各种脚本工具与描述文件来说明测试激励的组成，使用仿真模拟软件(NC-verilog 或者 vcs)加载测试激励以完成系统级的软件模拟测试。

为了充分验证 X 处理器中的每一个部件，X 处理器验证平台为每一个功能部件专门编写了测试集，用于验证每一部件的各种功能或操作，验证平台的设计人员针对各部件的具体功能和操作，编写了多个测试程序组。作为平台的使用者，

也可以通过了解 X 处理器指令集的汇编语法，参照现有的测试激励程序编写自己所需的测试激励。平台的测试激励使用的是高级汇编(HLA: High Level Assembly)，在很多测试中特别是多线程测试中使用了函数调用。使用者可以充分利用已有的测试程序快速的构造自己所需的测试。

2、测试内容和测试结果

验证中断处理部件功能的测试集一共包含 79 个测试激励，被分 4 组，分别用于测试一般中断、快速中断、异步中断以及调试模式下中断处理的功能正确性。每一组包含测试激励及其测试功能如表 5.3 所示：

表 5.3 中断部件系统级测试激励

组名	测试激励个数	说明
int_long	20	一般中断处理的测试
int_halt	10	调试模式下中断处理测试
int_fast	29	快速中断处理测试
int_disrupting	20	异步中断处理测试

验证平台最终产生的验证报告如图 5.13 所示。从报告中可以看出，4 组 79 个测试激励的测试均已通过。

```
Summary
================================================================================================
             Group:Total | PASS | FAIL |            Cycles |        Time |   C/S |
------------------------------------------------------------------------------------------------
          int_long:   20 |   20 |    0 |        6445834.00 |    62662.63 |102.87 |
          int_halt:   10 |   10 |    0 |         115118.00 |      960.10 |119.90 |
          int_fast:   29 |   29 |    0 |        4656339.50 |    35596.27 |130.81 |
      int_disrupting: 20 |   20 |    0 |        2258105.00 |    17548.60 |128.68 |
------------------------------------------------------------------------------------------------
               ALL:   79 |   79 |    0 |       13475396.50 |   116767.60 |115.40 |
================================================================================================

Total Diags  :                79
Total Passed :                79
Total Unknown:                 0
Total Unfini :                 0
Total Fail   :                 0
Total Cycles :       13475396.50
Total Time   :         116767.60
Average C/S  :            115.40
================================================================================================
```

图 5.13 中断部件系统级验证报告

5.2 逻辑综合

逻辑综合的任务是将寄存器传输级描述的电路行为和功能自动转换为门级结构网表描述。逻辑综合是根据用户的约束将用硬件描述语言描述的设计转化为目标工艺库中门表述的过程，是专用集成电路 ASIC 设计中的重要环节。

我们采用 Synopsys 公司的 Design Compiler 作为综合工具，基于某公司提供的工艺库，对中断部件的设计进行逻辑综合。工艺库的线宽为 65nm，参考时钟周期设置为 1ns，并采用零线负载模型。在该情况下得到的中断部件综合结果如表 5.4 所示：

表 5.4 中断部件逻辑综合结果

参数	值
工艺库线宽（nm）	65
综合条件	WORST
线负载模型	ZeroWireload
关键路径长度（ns）	0.86
单元数	189
面积（μm^2）	214906.98

综合结果显示，中断部件的逻辑划分合理，主频可以达到 1GHZ 以上，时序已经满足 X 处理器的设计要求。

5.3 本章小结

本章采用 down-top 的策略，按照模块级、单元级和系统级的层次顺序对中断部件的设计进行了大量的测试验证。验证的结果表明，中断部件符合 X 处理器中断系统设计方案的要求，实现功能正确。随后，我们使用 65nm 的工艺库对部件进行逻辑综合。综合结果显示，该部件的逻辑划分合理，时序上满足 X 处理器的设计要求。

第六章 结束语

多核多线程技术以其高度的线程并行性和较大的处理吞吐量等优点得到越来越广泛的应用。为了进一步提高多核多线程处理器的性能，以及运行虚拟化应用的效率，必须对中断系统进行重新设计和改进，并从硬件层次加入虚拟化支持。本课题的主要工作围绕多核多线程 X 处理器中断系统以及虚拟化的设计展开，设计并实现了适用于 X 处理器的虚拟化中断系统，解决了 X 处理器的多核多线程中断处理需求和虚拟化的需求。论文的研究工作对于 X 处理器的设计实现起到了重要的作用。

本文的主要工作及贡献在于：

（1）回顾了中断和中断处理的一般原理，提出了多核多线程架构下虚拟化中断系统的设计要求。文章对以往的中断处理技术进行了分析和总结，并讨论了多核多线程中断处理过程以及中断系统虚拟化中存在各类问题的解决方案。

（2）针对 X 处理器的设计要求，提出了 X 处理器中断系统的虚拟化方案和中断系统模型。阐述了 X 处理器的中断系统的设计实现，该系统支持多核多线程的中断处理，并从硬件上支持虚拟化应用。

（3）鉴于中断处理部件验证的复杂性，采用了自底向上的方法，进行有层次的模拟验证，验证了中断处理部件的正确性，并给出了逻辑综合结果，为 X 处理器的投片起到了一定的作用。

在本文撰写的同时，基于 X 处理器中断系统的实现及优化技术的相关研究工作正在不断深入。在已有研究成果的基础上，继续深入开展课题研究很有意义。

综合考虑，在下一步工作中需进一步研究的问题包括：

（1）X 处理器中断系统虚拟化效率的评估。由于 X 处理器目前仍然处于 RTL 级代码设计验证阶段，而且基于 X 处理器的虚拟化应用也是处于开发阶段，因此目前尚无法对 X 处理器中断系统虚拟化的实际工作效率做出有效的评估。随着 X 处理器设计项目总体进度的不断推进，有必要对现有 X 处理器中断虚拟化效率进行模拟测试，发现其中存在的性能瓶颈并做出相应的改进。

（2）中断处理性能的进一步优化。现有的多核多线程 X 处理器往往通过快速线程切换的技术来隐藏中断处理过程中长延迟操作，没有充分考虑到单线程的执行效率。可以从中断系统本身的机制出发，在现场保存和现场恢复等方面做进一步的优化和改进，从而在保证多核多线程处理器大吞吐量和线程级并行性的同时，提高单线程的性能。

通过本课题的研究和实践，可以看出多核多线程技术因其自身的诸多技术优势处于微处理器发展的前沿，多核多线程处理器的虚拟化实现技术因而也备受关

注。随着多核多线程结构的不断发展、虚拟化技术的不断进步，多核多线程处理器的虚拟化设计必然会成为一个极具发展潜力的研究方向。

致 谢

在我的课题和硕士论文完成之际，谨向在我攻读硕士学位的过程中曾经指导过我的老师、关心过我的朋友、关怀过我的领导、所有帮助过我的人们和我亲爱的家人致以崇高的敬意和深深的谢意！

首先要感谢我的导师张民选教授。衷心感谢张老师在百忙之中时刻关心我的学习情况和课题研究进展，并在生活上给予我细心的关怀。他宽广深厚的专业知识和严谨求实的治学态度使我受益匪浅。张老师不仅对我的学习和研究进行了指导，而且还教会我很多做人的道理以及工作的经验。他踏实的工作作风、平易近人的态度和积极的进取精度，深深感染了我，激励我奋发向上，以不辜负张老师的期望与信任。

邢座程教授、高军老师和唐玉星老师在课题研究期间给予了我无私的关怀和真诚的帮助。从论文的选题、实验的进展都始终给予我细心的指导和不懈的支持。以他们广博的专业知识、严谨的工作态度，对我的研究和工作给予了很多直接的指导。谨向三位老师致以深深的谢意！

感谢“高性能 X 处理器”项目组的所有老师，他们为我的工作提出了很多的建议和帮助，时刻关心着我的研究和工作的进度和困难，给予热忱指导。项目组中浓郁的学术氛围，让我不仅学到了知识，还学到了一个独立学者所必需的意志和勇气。能够参与 X 处理器的设计，是我一生重要的一次学术历练和人生经历，所学到的一切都将成为我一生中最宝贵的财富。

冯超超、胡文敏、周海亮、隋兵才等几位师兄在课题的不同阶段分别给予我很大的指导和帮助，使我的课题工作得以顺利进行。在此，我谨向他们表示衷心的感谢！

感谢在项目组中一起愉快合作的程汉强、姚云茂、郑勇、黄安文、魏永成、李秋亮、文良勇等诸位同学，正是由于他们的帮助和支持，我才能克服一个个困难和疑惑，直至本文的顺利完成。

感谢我的室友黎渊、黄冕、王县和同学杨强、刘宗福、吴庆，作为为最亲密的同学和最真诚的朋友，他们给了我极大的鼓励。

感谢计算机学院硕士研究生队的所有队领导对我的支持和帮助，感谢 2006 级的所有同学，与他们在一起让我度过了快乐的研究生时光。

感谢我的家人，每当我遇到困难或是难以坚持的时候，是他们的关心、鼓励和理解支撑着我，激励着我。他们是我取得任何进步的精神支柱和动力源泉。

再次对所有给予我支持和帮助，关心和鼓励我不断进取的人致以我最诚挚的谢意！

参考文献

[1] L. Hammond, B. A. Nayfeh, K. Olukotun. A Single-Chip Multi-Processor. IEEE Computer. 1997, 30(9): 79-85

[2] L. Hammond, B. Hubbert, etc. The Stanford Hydra CMP. IEEE Micro. 2000,20(2)

[3] D. Tullsen, S. Eggers, H. Levy. Simultaneous Multithreading: Maximizing On-Chip Parallelism. Intl. Computer Architecture. 1995: 392-403

[4] L. Spracklen, S. G. Abraham, etc. Chip Multithreading: Opportunities and Challenges. Proceedings of HPCA-11, 2005

[5] Ana Sonia Leon, Brian Langley, Jinuk Luke Shin. The UltraSPARC T1 Processor: CMT Reliability. on Conference 2006, IEEE Custom Integrated Circuits. 2006.9: 555-562

[6] John L. Henney, David A. Patterson. Computer Architecture: A Quantitative Approach (Fourth Edition). Publishing House of Electronics Industry. 2007

[7] 刘近光, 梁满贵. 多核多线程处理器的发展及其软件系统架构. 微处理机. 2007.2: 1-3

[8] 黄鹏. CMT: 处理器吞吐量倍增的秘诀. 程序员. 2006.3: 122-124

[9] Greg Goth. Virtualization: Old Technology Offers Huge New Potential. IEEE Distributed Systems Online. 2003,8(2): 78-82

[10] R.P. Goldberg. Survey of Virtual Machine Research. Computer. 1974.6: 34-45

[11] VMware Inc. Building Virtual Infrastructure with VMware VirtualCenter. White paper V00014-20001205, 2004. www.vmware.com/pdf/viwp.pdf

[12] Microsoft Corp. Virtual PC Technical Overview. www.microsoft.com/ windowsxp/ virtualpc

[13] VMware Inc. Virtualization Overview. www.vmware.com/solutions/

[14] P. Barham, etc. Xen and the Art of Virtualization. Proc. 19th ACM Symp. Operating Systems Principles, ACM Press, 2003: 164-177

[15] VMware Inc. Understanding Full Virtualization, Paravirtualization, and Hardware Assist. www.vmware.com

[16] Microsoft Corp. Windows Server Virtualization：Windows Server 2008 的关键功能. www.microsoft.com/china/windowsserver2008/

[17] Rich Uhlig, Gil Neiger, Dion Rodgers, etc. Intel virtualization technology. Computer. 2005.5:48-56

[18] Intel Corp. Intel Virtualization Technology Specification for the IA-32 Architecture. www.intel.com/technology/vt/

[19] Intel Corp. Intel Virtualization Technology Specification for the Intel Itanium

Architecture. www.intel.com/technology/vt

[20] Intel Corp. Intel Virtualization Technology for Directed I/O. www.intel.com/technology /vt

[21] AMD Inc. Live Migration with AMD-V Extended Migration Technology. 2007

[22] Sun Microsystems, Inc. Sun4v and LDom-an introduction. www.sun.com

[23] W. J. Armstrong, R. L. Arndt, etc. Advanced Virtualization Capabilities of POWER5 Systems. IBM J. RES. & DEV. 2005.7: 523-532

[24] 陆志才. 微型计算机组成原理. 高等教育出版社. 2003

[25] Sun Microsystems Inc. UltraSPARC Architecture 2007: One Architecture-Multiple Innovative Implementations. 2007.8. www.sun.com

[26] Intel corp. Intel Architecture Software Developer's Maual Volume 3: System Programming. 1999

[27] Eckert R. Communication between Computers and Perheral Device-An Analogy. ACM SIGCSE Bulletion. 1990.9

[28] 周明德. 微型计算机系统原理及应用(第四版). 清华大学出版社. 2002.6

[29] J. E. Smith, A. R. Pleszkun. Implementation of precise interrupts in pipelined processors. Proc 12th Annual Intenrational Symposium on ComputerA rchitecture (I SCA'85). 1985: 36-44

[30] Harry Dwyer, H. C. Torng. An Out-of-Order Superscalar Processor with Speculative Execution and Fast, Precise Interrupts. Microarchitecture, MICRO 1992, 25 (8): 272-282

[31] Jaleel A, Jacob B. Improving the precise interrupt mechanism of software-managed TLB miss handlers. LNCS 2228: HiPC 2001. 2001 : 284-293

[32] Harsh Sharangpani, Ken Arora, Itanium Processor Microarchitecuture. 2000.9

[33] Jean J. UC/OS-I I, the real time kernel(2nd edition). R&D Technical Books, 2002

[34] 席晨，张盛兵，等. 基于备份缓冲区的精确中断研究与实现. 计算机工程与应用，2007, 43 (6): 95-98

[35] 许新任，陈进. 高性能 DSP 中断处理技术. 计算机工程. 2004.10(30): 176-177

[36] David L. Weaver, Tom Germond. The SPARC Architecture Manual, Version 9. SPARC International, Inc. 1994

[37] 祝帅君，高军，张民选等. 多线程处理器精确中断机制的分析与设计. 第十二届计算机工程与工艺全国学术年会论文集(NCCET'08). 2008.8: 139-142

[38] 阳柳，胡建国，李鑫. 高性能微处理器的全芯片验证. 第九届计算机工程与工艺全国学术年会论文集. 2005.8: 369-371

[39] Janick Bergeron 著，张春等译. 编写测试平台-HDL 模型的功能验证 (第二

版). 电子工业出版社. 2006.8

[40] 吕涛，李华伟，李晓维等. 基于模拟的验证技术在 CPU 设计中的应用. 同济大学学报. 2002,30(10): 1257 -1261

作者在学期间取得的学术成果

[1] 祝帅君，高军，张民选，谢胡. 多线程处理器精确中断机制的分析与设计. 第十二届计算机工程与工艺全国学术年会论文集. 2008.8: 139-142