版权声明

摘要

随着集成电路工艺的进步和计算机技术的发展，计算机系统的功耗逐步成为制约计算机技术进一步发展的瓶颈。为了解决计算机系统的功耗问题，各种功耗管理技术应运而生，成为学术界、工业界的共同研究热点。

北大众志 PKUnity-3 系统芯片是北大众志"超 K 计划"面向单芯片个人计算机、便携式移动计算终端、安全微工作站等应用领域而提出的系统性解决方案，该系列芯片采用了多种功耗管理技术，具有低功耗、高性能、低成本等优点。

本文的主要工作是充分利用 PKUnity-3 系统芯片硬件在功耗管理方面提供的基本支持，在 Linux 操作系统之中设计并实现面向 PKUnity-3 系统芯片平台的功耗管理机制：系统级动态功耗管理机制和时钟管理机制。其中，系统级动态功耗管理机制涉及系统中的所有设备，具有处理复杂、难于调试等特点，是本文工作的重点和难点。该机制具有：工作（Working）、空闲（Idle）、睡眠（Sleep）、休眠（Hibernation）和关机（Off）五种运行模式，能够支持在 PKUnity-3 系统芯片平台之上进行有效的系统级动态功耗管理。时钟管理机制则直接关系到整个计算机系统的正确、高效运行，是本文工作的重要内容，主要包括操作系统时钟初始化、动态频率设置、时钟门控和内核接口四方面内容。该机制的实现为系统高效运转和调试提供了有力支持。在上述主要工作的基础之上，本文还设计了一种面向 PKUnity-3 系统芯片平台的功耗管理实施方案，并以系统级动态功耗管理作为该实施方案的应用实例，在 PKUnity-3 系统芯片平台上实现了系统级的动态功耗管理。

综上，本文面向 PKUnity-3 系统芯片平台，在 Linux 操作系统之中设计并实现了包含系统级动态功耗管理机制和时钟管理机制两部分内容的操作系统功耗管理机制。此外，本文还设计了一种面向 PKUnity-3 系统芯片平台的功耗管理实施方案，并以系统级动态功耗管理为例，在北大众志计算机中实现了系统级动态功耗管理。目前，该系统级动态功耗管理已经成功应用于北大众志 A25 计算机之中，能够有效降低该款计算机系统空闲时的功耗。

关键词： PKUnity-3 系统芯片，系统级动态功耗管理，时钟管理

Design and Implementation of Operating System Power Management Mechanisms for PKUnity-3 Soc

He Jun (Computer Architecture)
Directed by Wang Keyi

Abstract

With the development of computer technology and the advancement in the integrated circuit technique, the computer performance has been boosted for decades. However meanwhile, the performance has been largely constrained by the power consumption. To conquer the bottleneck, effective power management techniques have become a popular topic in both the academia and industry.

The PKUnity-3 Soc characterized by low power-consuming, high-performance and efficiency, etc, is personalized chip which is oriented to the PC, portable mobile computing terminal and micro workstation and so on. Furthermore, it is totally MPRC-made (Micro Processor Research and Development center of Peking University) and an important part of the "Super K plan".

In this thesis, we focus on the systematic level dynamic power management mechanism and the clock management mechanism based on the PKUnity-3 Soc hardware's support. The goal is to implement the mechanisms in Linux operating system kernel and serve the PKUnity-3 Soc platform. More specifically, the systematic level dynamic power management mechanism consists of 5 modes, the Working, Idle, Sleep, Hibernation and the Off, and the clock management mechanism mainly includes: the initialization of operating system clock, the dynamic frequency setting up, the clock gating and the kernel interfaces. The systematic level dynamic power management mechanism makes the power management in OS possible on the platform of PKUnity-3 Soc, while clock management mechanism guarantees the high–efficient running and debugging of the computer system.

Finally, we propose an administration plan based on the PKUnity-3 Soc aiming at lowering down the systematic level dynamic power consumption and the plan has been successfully applied to the A25 computer produced by the MPRC.

Keywords: Pkunity-3 Soc, Systematic level dynamic power management, Clock management

目录

图目录

表目录

第1章 绪论

1.1 研究背景

随着集成电路工艺的进步和计算机技术的发展，计算机系统的性能不断增强。各种类型的计算机广泛应用于人们生产、生活的各个方面，为经济社会的发展进步发挥了重要的作用。但是，在计算机系统性能显著提高的同时，也带了另一个显著问题——系统功耗问题。当前，计算机系统功耗已经成为制约计算机技术进一步发展的瓶颈，功耗与计算机性能、成本一起，成为计算机设计、制造时必须考虑的重要因素[1][2][3]。

通常情况下，计算机系统功耗是指计算机系统运行过程中，硬件电路所消耗的功率，单位为瓦特。功耗对计算机发展的制约作用，主要表现在以下几个方面：

- 降低计算机系统可靠性。对更高计算性能的追求，一直是推动计算机科学与技术不断发展进步的动力。为了获取更高的计算性能，人们不断提高硬件芯片集成度和运行频率。但是，芯片集成度和运行频率的提高必然导致更高的系统功耗，产生更多的热量。如果产生的热量得不到有效的处理，计算机系统的运行温度将不可避免地升高。已有研究表明，随着集成电路温度的升高，将会产生诸如硅片连线故障、封装故障、电学参数偏移、电迁移等一系列问题，进而导致电路工作的可靠性不断下降。一般来说，衬底温度每升高 10℃，系统可靠性下降 50%，同时性能损失 3%[4]。
- 增加计算机散热成本。作为一种成熟的工业产品，计算机系统的成本一直是计算机生产厂家和设计人员必须考虑的重要方面。但是，面对市场对更高计算性能的需求以及随之而来的功耗问题，计算机设计人员只能采用成本更高的散热工艺和装置，导致计算机系统整体成本不断增加。以计算机处理器的散热为例，为了使处理器在获得更高性能的同时，兼具很高的系统稳定性，设计人员往往会从两方面来对处理器进行散热：采用更好的芯片封装工艺（如将芯片由塑料封装改为成本更高但散热效果更好的陶瓷封装），和采用更好的散热装置（由风冷散热改为水冷散热，甚至液氮散热等）。

- 限制计算机性能的提升。由于计算机系统的散热成本不可能无限制的增加，因此计算机散热能力在一定程度上受到限制。在计算机硬件散热能力一定的条件下，为了保证系统的稳定性，计算机厂家不得不以牺牲硬件工作频率的方式来降低计算机系统的功耗，因此限制了处理器性能的进一步提高。
- 限制移动、便携式计算设备的使用。首先，在移动、便携式计算设备中，电池是主要电源供应来源。用户对更长续航时间和更高计算性能的需求，对电池技术的发展提出了挑战。而过去的 30 年中，电池容量仅增长了 2~4 倍，电池寿命仅延长了 40%，远远低于同时期计算机性能和功耗的增长速度[5]。其次，由于移动、便携式计算设备（如手机、上网本等）往往具有体积较小，易于携带的特点，体积较大的散热装置将难以安装在设备之上，进而对设备的散热提出了更高的要求。因此，功耗问题将不可避免的限制移动、便携式计算设备的广泛使用。

为了解决计算机系统的功耗问题，大量管理和控制计算机系统中功率消耗的功耗管理技术应运而生。目前，功耗管理技术是计算机学术界和工业界共同关注的研究热点，取得了一定的研究成果。

PKUnity-3 系统芯片是北大众志面向单芯片个人计算机、便携式移动计算终端、安全微工作站等应用领域而提出的系统性解决方案[6]。由于这些应用领域对功耗极为敏感，因此最大程度降低系统芯片运行功耗成为 PKUnity-3 系统芯片设计的重要目标。

为达此目标，PKUnity-3 系统芯片运用了多种功耗管理技术。空闲模式（Idle）、睡眠模式（Sleep）是 PKUnity-3 系统芯片硬件基于时钟门控技术实现的系统级低功耗模式，为操作系统和应用软件实现系统级的动态功耗管理提供了基本支持。此外，PKUnity-3 系统芯片硬件还提供了对系统芯片上主要硬件设备进行分频比设置和时钟门控的软件接口，允许软件根据实际需要（如系统芯片调试、功耗管理等），在系统运行过程中动态调整硬件设备的运行频率和门控其时钟输入。

为了充分利用 PKUnity-3 系统芯片对软件功耗管理提供的各种技术，本文试图在 Linux 操作系统内核之中实现较为完整的、基于 PKUnity-3 系统芯片功耗管理模块硬件的功耗管理机制，以便为系统功耗管理提供有力支持，进而达到对系

统进行有效功耗管理的目的。

1.2 研究平台

本文研究工作的硬件平台是北大众志 PKUnity-3 系统芯片及位于其中的功耗管理模块，软件平台是 PKUnity-MUSE 软件系统。本节将对本文工作的软、硬件平台做简要介绍。

1.2.1 PKUnity-3 系统芯片

PKUnity-3 系统芯片[7]是北大众志"超 K 计划"面向单芯片个人计算机、便携式移动计算终端、安全微工作站等应用领域而提出的系统性解决方案，具有低功耗、高性能、低成本等优点。目前，PKUnity-3 系统芯片包括：PKUnity-3(SK)和 PKUnity-3(65) 两款型号。

PKUnity-3(SK)芯片是北大众志"超 K 计划"的第一代系统芯片。该芯片是目前国际上为数极少的可提供"单芯片个人计算机"主板解决方案的系统芯片，CPU 典型工作频率为 700MHz。PKUnity-3(SK)系统芯片结构如图 1-1 所示。

从图 1-1 中可以看出，PKUnity-3(SK)系统芯片采用使用北大众志自主指令系统标准的 UniCore-2 CPU，支持 32 位定点处理、64 位浮点处理和 2D/3D 增强指令处理。该芯片中实现的多媒体和显示子系统 UniGFX，集成了 GPU 的核心功能并具有专用部件对 MPEG-1/2/4 和 H.264 等视频编解码进行加速，以满足迅速增长的多媒体处理需求。在存储方面，该芯片集成了桌面计算机中主流的存储控制设备，包括 DDR2 SDRAM 控制器、SATA 硬盘控制器、NAND Flash 控制器和 MMC/SD 控制器等，构成了芯片内部的多层次存储控制子系统。此外，该芯片中还集成了 PCI 2.2 主控制器、10M/100M/1G 以太网控制器、USB 2.0 控制器、静态存储控制器、AC'97 控制器、串口控制器和 PS/2 键盘鼠标控制器等南北桥芯片组中提供的功能部件，以提供日常应用所需的设备接口。该芯片的晶体管数量大约为 5500 万（根据晶片的面积换算），封装后尺寸为 35 毫米×35 毫米，管脚数量 976 个。

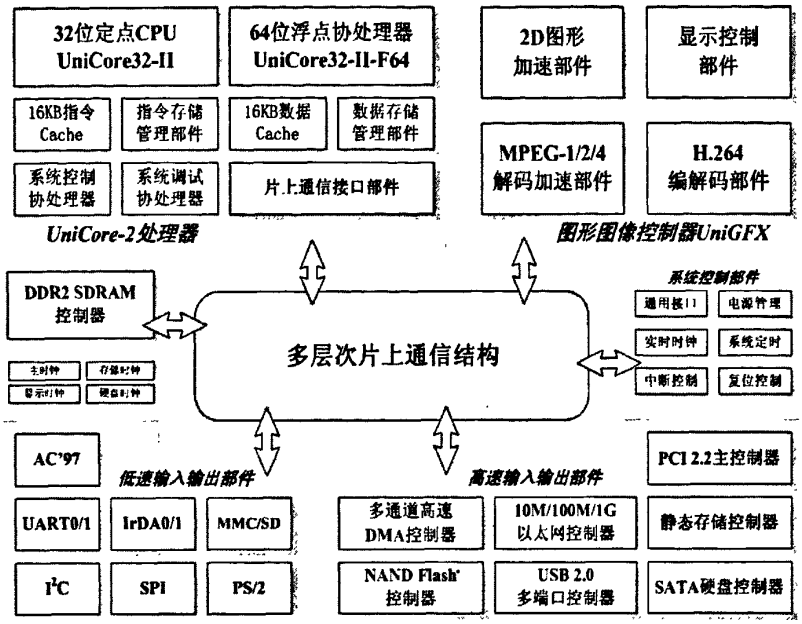

图 1-1 PKUnity-3(SK)系统芯片结构图

PKUnity-3(65)是北大众志"超 K 计划"继 PKUnity-3(SK)芯片之后推出的第二代系统芯片。PKUnity-3(65)采用 65 纳米工艺，集成了增强型 UniCore-2 CPU 和新一代 2D/3D 图形加速部件、高带宽显示控制部件、多格式的视频编解码部件，以及众多其它外设控制器。由于采用了新的制造工艺和图形加速部件，系统典型工作频率和图形显示效果获得了大幅提高，CPU 典型工作频率达到 1.2GHz。

1.2.2功耗管理模块硬件

功耗管理模块硬件（Power Manager，PM）[4][8][9]是挂接在 PKUnity-3 系统芯片低速总线 APB 上的从设备，简称 PM 模块。

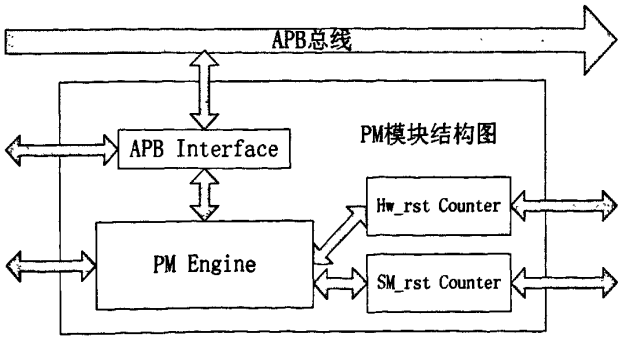

图 1-2 PKUnity-3 系统芯片功耗管理模块结构图

PKUnity-3 系统芯片功耗管理模块硬件结构如图 1-2 所示。

从图 1-2 中可以看出，PM 模块共有四个子模块：APB 总线接口（APB Interface）、控制状态机（PM Engine）、硬件复位计数器（HW_rst Counter）、睡眠复位计数器（SM_rst Counter）。其中，APB 总线接口是 APB 总线与 PM 模块进行数据交换的接口，包含所有与功耗管理模块相关的寄存器；控制状态机负责控制系统中各时钟的上电复位时序、锁相回路重配置时序和系统进入不同功耗模式的时序；硬件复位计数器负责保证系统硬件复位信号结束时所有时钟均正确产生并到达各功能模块；睡眠复位计数器负责保证系统睡眠复位信号结束时所有时钟均正确产生并到达各功能模块。

功耗管理模块硬件的主要功能有：

1. 管理锁相回路（Phase Locked Loop，PLL）。负责 PLL 打开和关闭，设置 PLL 输出端的频率。PLL 是否关闭由功耗管理模块硬件通过自身运行状态进行判别，不向软件提供关闭 PLL 的编程模式。软件可通过配置睡眠模式下 PLL 的操作方式来间接控制睡眠模式下 PLL 的打开与关闭。PLL 的频率设置可由软件配置相应寄存器实现，PLL 配置信号由功耗管理模块硬件产生。

2. 管理系统芯片中各主要模块的输入时钟分频比。功耗管理模块在 PLL 与各主要模块之间添加了分频器和时钟门控逻辑，软件可通过操作相应的分频比寄存器和时钟门控寄存器实现对系统芯片中各主要模块的输入时钟进行管理。

3. 管理系统芯片的运行模式。功耗管理模块支持 3 种系统级运行模式：运行（Run）、空闲（Idle）和睡眠（Sleep）。a) 运行模式下，系统中所有模块均处于运行状态；b) 空闲模式下，CPU 模块停止运行，CPU 模块输入时钟停止。在空闲模式下，必须保证 GPIO、中断控制器、RTC、OST 和功耗管理模块自身正常运行。软件可通过配置功耗管理模块中的相应寄存器使系统芯片进入空闲模式，任意硬件中断将使系统退出空闲模式；c) 睡眠模式下，除 GPIO、RTC 和功耗管理模块自身在 RTC 时钟驱动下继续运行外，其余所有模块输入时钟均被关闭。通过配置功耗管理模块中的一些寄存器使系统进入睡眠模式，但是在系统进入睡眠模式之前，软件必须设置使系统芯片退出此模式的唤醒源。在 PKUnity-3(SK)芯片中，退出此模式后将会产生一个全局复位信号，使系统芯片复位。而在 PKUnity-3(65)中，退出睡眠模式时，是否产生全局复位可由软件通

过配置功耗管理模块相应寄存器进行决定。

1.2.3 MUSE 软件系统

MUSE（MPRC UniCore32 Software Environment）是运行在 PKUnity-3 系统芯片平台上的软件系统，如图 1-3 所示。

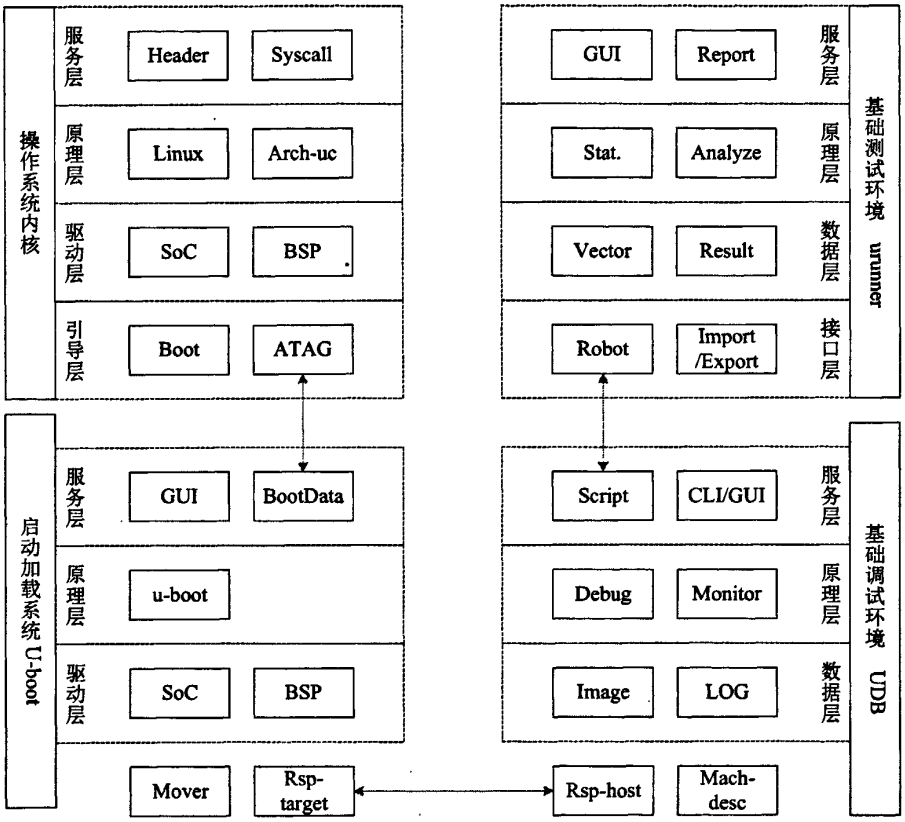

图 1-3 MUSE 软件系统

从图 1-3 可以看出，MUSE 软件系统由四个部分组成，分别是操作系统内核 Linux-uc，启动加载系统 U-Boot，基础调试环境 UDB 和基础测试环境 U-Runner。

Linux-uc 是由标准 Linux 内核移植而来的、在 PKUnity-3 系统芯片上运行的操作系统内核，当前版本是 2.6.32。它分为四个层次，引导层引导操作系统启动、驱动层包含各个模块的驱动程序、原理层是基于 Linux 内核机制的实现、服务层包含系统调用等 Linux 内核服务的实现。

U-Boot 包括四个部分：服务层提供图形交互界面，为 Linux 内核提供启动参数；原理层即为 U-Boot 机制的实现；驱动层是启动时必要模块的驱动程序；

通信层在调试加载等工作时与主机端通信。

UDB 是运行在调试主机之上，用于对 PKUnity-3 系统芯片平台进行远程调试的基础调试环境。它通过最底层的通信层协议和目标机端的 U-Boot 系统进行通信，从而控制目标机端的程序运行，完成调试工作。在通信层协议的上层是数据层，数据包括运行所需要加载的内核二进制镜像，和运行以后所输出的调试信息（以 log 的形式保留下来）。原理层即为调试功能的实现。最上层是服务层，script 和 GUI 等都在此层实现。

U-Runer 包括四个部分，接口层用于与 UDB 通信，以运行 script 的方式将数据批量通过调试环境加载到目标机上运行测试；数据层的功能是对需要测试的文件或数据进行管理；原理层是对测试文件的分析和归类；最上层的是服务层，即为用户交互层，用户的输入为数据文件和命令，能得到的返回结果是多种形式的结果报告。

MUSE 软件环境之上构建有 PKUnity-3 应用软件系统，从而形成完整的软件环境，支持 PKUnity-3 系统平台的多种应用。

1.3 研究动机及内容

本文最根本的研究动机是：通过在操作系统内核之中设计、实现功耗管理机制，以便能够充分利用 PKUnity-3 系统芯片功耗管理模块的硬件特性，为 PKUnity-3 系统芯片平台的功耗管理和软硬件调试提供支持，达到降低系统运行功耗和提高系统芯片调试效率的目的。

在本文工作完成之前，尽管 PKUnity-3 系统芯片功耗管理模块为软件提供了系统级低功耗模式、时钟频率设置和时钟门控等基本支持，但是由于操作系统内核之中缺乏相应的功耗管理机制，导致硬件的特性没有得到充分的使用。

比如功耗管理模块硬件提供了支持系统级动态功耗管理的空闲和睡眠运行模式，但是由于操作系统中相应功耗管理机制的缺失，导致系统级动态功耗管理无法在 PKUnity-3 系统芯片平台上有效实现，进而导致当系统负载较轻、甚至空闲时存在功耗大量浪费的现象。

又比如在系统芯片的调试过程中，常常需要在不同时钟频率对系统或系统硬件模块进行性能和功能调试。由于操作系统中缺乏直接操作功耗管理模块硬件的

时钟频率设置机制和时钟门控机制，导致时钟频率的修改和门控的设置只能在系统加载程序 U-boot 完成。但是，U-boot 程序中并无向外提供修改时钟频率和门控配置的接口，因此为了实现不同时钟频率下的系统芯片调试，需要频繁的修改、编译U-boot代码，需要反复烧录保存U-boot二进制代码的Flash（在PKUnity-3(SK)平台中，U-boot 二进制代码保存在 NOR Flash 之中，而在 PKUnity-3(65)中，U-boot 二进制代码则保存在 SPI NOR Flash 之上），然后重新引导并启动操作系统。由于修改、编译 U-boot 代码、烧录 Flash 芯片和重新引导启动操作系统是一个极其繁冗的过程，因此不可避免的会降低 PKUnity-3 系统芯片的调试效率。

本文主要工作及贡献如下：

1.设计并实现了系统级动态功耗管理机制。系统级动态功耗管理机制具有涉及设备众多、处理复杂、难于调试等特点，是本文工作的重点和难点。本文在参考、借鉴操作系统之中主流的系统级动态功耗管理机制的基础之上，结合 PKUnity-3 系统芯片提供的硬件支持，本文设计并实现了支持工作（Working）、空闲（Idle）、睡眠（Sleep）、休眠（Hibernation）和关机（Off）五种运行模式的系统级动态功耗管理机制，为系统级动态功耗管理提供了的有力支持。

2.设计并实现了时钟管理机制。时钟管理机制直接关系到整个计算机系统的正确、高效运行，是本文工作的重要内容。本文设计并实现的时钟管理机制包含操作系统时钟初始化、时钟频率设置和时钟门控三方面内容。操作系统时钟初始化是指操作系统根据系统芯片的硬件设置，初始化并维护系统中各个时钟的频率数值，并向应用程序和驱动程序提供相应查询接口，以便应用程序和驱动程序根据查询到的时钟频率数值而采取正确的行为；时钟频率设置和时钟门控这两种机制的实现既方便了系统芯片的调试工作，提高了调试效率，同时也为根据系统运行载而进行动态变频、动态时钟门控的功耗管理技术提供了基本支持。

3. 设计了一种面向 PKUnity-3 系统芯片平台的功耗管理实施方案，并以系统级动态功耗管理作为该实施方案的应用实例，在 PKUnity-3 系统芯片平台上实现了一种系统级的动态功耗管理，有效地减少了系统空闲时所消耗的能量。目前，系统级动态功耗管理已在北大众志 A25 计算机产品中得到实际应用。应用结果表明，该系统级动态功耗管理能够有效降低 A25 计算机空闲时的功耗，符合国家最新发布的计算机产品节能认证规范（CQC3114-2009）。

1.4 本文组织结构

本文的组织结构如下：

第一章首先从功耗管理的重要性和 PKUnity-3 系统芯片项目需求两方面介绍了本文的研究背景，然后介绍了本文工作的软、硬件平台，最后在上述内容之上明确了本文的研究动机和主要工作内容。

第二章分别从功耗的组成、功耗管理的主要技术、当前主流的工业标准和 Linux 内核中的功耗管理技术四个方面介绍了本文的相关工作，指出了 Linux 内核功耗管理技术对不兼容 APM 和 ACPI 标准的平台支持不足的问题，明确了本文工作的基本技术路线。

第三章详细讨论了面向 PKUnity-3 系统芯片的操作系统系统级动态功耗管理机制的设计与实现。在借鉴 APM 和 ACPI 标准的基础之上，结合自身平台特点提出并实现了包含工作(Working)、空闲(Idle)、睡眠(Sleep)、休眠(Hibernation)和关机(Off)五种运行模式的系统级动态功耗管理机制，为第五章面向 PKUnity-3 系统芯片平台的系统级动态功耗管理提供了支持。

第四章设计并实现了操作系统功耗管理机制中的时钟管理机制。时钟管理机制实现了包括操作系统时钟初始化、动态频率设置、时钟门控在内的时钟管理功能，并分别向内核驱动程序和用户空间提供了查询、控制系统时钟频率和门控时钟的接口。

第五章在第三、四章工作的基础之上，首先设计了一种面向 PKUnity-3 系统芯片平台的功耗管理实施方案，然后以系统级动态功耗管理作为该实施方案的应用实例，实现了一种面向 PKUnity-3 系统芯片平台的系统级动态功耗管理，并进行了相应的功耗评测。

第六章对本文的工作进行了总结并对未来工作进行了展望。

第2章 相关工作

本章主要介绍本文的相关工作。首先介绍计算机系统功耗的组成、然后介绍常见的功耗管理技术，接下来介绍工业界主流的功耗管理标准，最后介绍 Linux 操作系统中实现的各种功耗管理技术。

2.1 功耗的组成

通常来讲，计算机系统功耗是指计算机运行过程中，整个硬件电路所消耗的功率。

从宏观结构上看，计算机系统功耗是计算机系统中各功能部件功耗之和。由于计算机系统中各功能模块的划分、各功能部件的种类和数量的确定是在计算机系统级设计时确定的，因此，系统层级的设计对计算机系统的功耗具有直接而根本的影响。

从微观电路上讲，系统电路功耗主要由静态功耗、动态功耗两部分组成[10]。静态功耗是指器件处于静默状态、输入信号未激活情况下消耗的功率，也称之为稳态功耗或待机功耗。静态功耗主要包括：静态直流功耗，指由泄漏电流引起的静态功耗；漏电流功耗，指亚阈值电流和反向偏压电流引起的静态功耗。动态功耗是指电路正常工作时消耗的功率，也称工作功耗。动态功耗主要包括：翻转功耗（Switching Power），指当逻辑门输出逻辑值发生改变时，电源对输出电容进行充放电所消耗的功率；短路功耗，指由于器件输入信号波形并非具有零的上升和下降时间（即输入波形不是理想的方波），而导致在输入信号上升或下降过程中存在短时的直流通路，进而消耗的那部分功率。通常，寄存器传输层级、电路层级的设计对系统的电路功耗具有重要影响。

2.2 功耗管理

功耗管理泛指管理和控制计算机系统中功率消耗的技术和方法。由于计算机系统设计本身涉及多个层次，因此作为系统设计重要目标之一的低功耗属性也贯穿于计算机系统设计的各个层次之中[11]。本节将主要从系统层级、寄存器传输层

级和电路层级三个层次简要回顾功耗管理的相关技术和方法。

2.2.1 系统层级

系统层级的功耗管理技术主要有：软硬件功能划分、存储器优化、指令集优化、动态变频技术、可变电压和多电压技术、动态功耗管理、总线低功耗设计等。

● 软硬件功能划分

软硬件功能划分[12][13]是指从系统功能的抽象描述着手，把系统功能分解为软件和硬件两大部分分别加以实现。比如，对于一个具体的功能任务，可通过：1)编写、运行软件程序和 2）设计、采用专用硬件电路两种方式加以实现。系统设计者可通过比较这两种实现方式所消耗的功耗而得到一个低功耗实现方案。软硬件划分处于系统设计的初始阶段，能够更大程度的节省系统功耗。

● 存储器优化

存储器是计算机系统的重要组成部分，不同的存储结构和组织方法对计算机系统的性能和功耗有重要影响。面向低功耗的存储器优化研究，充分利用了不同存储器件（如 SRAM，DRAM，Cache 等）在功耗方面的差异，通过采用采用了不同的存储结构和组织方法来尽可能地减少存储器所消耗的功耗[14][15][16]。

● 指令集优化

指令集优化主要涉及四个方面。1) 指令集提取[17]。对于确定的处理器，其每条指令的功耗是一定的。对于某一确定的系统功能，通常可采用不同的指令集加以实现。指令集提取的目的就是选择功耗最小的指令集来实现某一确定的系统功能，进而降低计算机完成该功能时消耗的能量；2) 指令长度的确定[18]。不同的指令长度（如固定长度 16 位、32 位，或可变长度）将会直接影响代码密度和代码执行时对存储器的访问次数。通过选择合理的指令长度，可以尽可能减少代码执行时的存储器访问次数，进而减少功耗；3) 指令编码优化[19][20]。通过对运行于计算机系统之上的典型程序的指令相关性进行分析，优化指令的编码，以使总线上的信号翻转次数最少，进而减少系统的功耗；4) 指令的压缩[21]。通过将指令进行压缩后存储在存储器之中，可有效减少程序执行过程中的存储器访问次数，进而降低系统功耗。

● 动态变频技术

时钟频率与电路的翻转功耗成线性关系。因此，降低频率可降低系统的功耗。但是，时钟频率降低将会使系统性能下降，故在实际使用中需要对功耗和性能之间进行折中[22]。

● 可变电压和多电压技术

可变电压和多电压技术[23][24]分别从时间和空间上对输入电压进行变换以达到降低功耗的目的。系统输入电压对系统功耗特别是电路动态功耗具有至关重要的影响，降低输入电压是降低系统功耗的重要方法。但是降低输入电压将导致电路性能降低，因此降低电压时必须在功耗和性能之间进行折中。可变电压技术根据系统不同的工作状态对系统性能的不同要求，动态的改变电压以实现系统功耗的降低。但是，电压转换过程中产生的额外功耗和引入的延迟时间是制约可变电压技术的关键因素。多电压技术则是根据系统电路中不同组成部分对性能的要求不同，而对不同电路部分采用不同的电压，进而达到降低系统功耗的目的。不同电压信号之间的转化代价是多电压技术的关键因素。

● 动态功耗管理

动态功耗管理[25][26][27][28][29][30]是一种使系统或系统单元在负载较轻或不工作时进入具有较低功耗模式的控制技术。根据动态功耗管理的对象所处层级的不同，可将动态功耗管理分为系统级动态功耗管理和设备级动态功耗管理。动态功耗管理需要软、硬件系统协同配合才能实现。其中，软件负责确定系统进入低功耗模式的时机和完成相应的准备工作，硬件则利用可变电压技术或时钟门控技术为系统低功耗模式提供基本支持。

● 总线低功耗设计

总线低功耗设计主要包括：减小总线上信号的电压变化幅度、总线的分段隔离和总线数据编码[31][32]等技术。

2.2.2 寄存器传输层级

寄存器传输层级功耗管理技术主要有时钟门控技术、预计算逻辑、状态机的状态分配优化、工艺映射优化、门尺寸优化等。

● 时钟门控技术

由于时钟树上的时钟缓冲器自身或其内部节点具有极高的翻转频率，因此时

钟网络消耗的功率在电路的动态功耗中占有相当大的比例。为了降低因时钟而导致的这部分功耗，时钟门控技术[33][34]在电路处于空闲或做无用运算时关断其时钟信号，从而有效减少了时钟驱动的功耗。时钟门控技术既可作用于某一局部电路或模块，也可作用于整个电路。时钟门控技术作用的范围越大，减少的功耗越多。

● 预计算逻辑

预计算逻辑[35]的基本思想是提前一个时钟周期计算电路中某些重要的逻辑输出值，然后根据这些计算的结果来减少电路内部的翻转次数。

● 状态机的状态分配优化

对采用硬件电路实现的有限状态机，将其中的相关性强的状态分配汉明距离近的状态编码，可以减少状态转换引起的电路翻转。

● 工艺映射优化

工艺映射[36]是把逻辑表达式或布尔网络映射到目标库中的门单元的过程。在工艺映射过程中，将活动因子大的节点尽量隐藏于门单元内部，可减少其电容负载，进而降低功耗。

● 门尺寸优化

门尺寸优化[37][38]通过缩减位于非关键路径之上的门的尺寸，以达到减小面积和功耗的目的。

2.2.3 电路层级

电路层级的功耗管理主要有：时钟树生成优化和晶体管尺寸优化。

● 时钟树生成优化

由于时钟信号的翻转频率通常较高，因此时钟树的功耗在系统总体功耗中占有相当大的比例。在保证时序约束的前提下，对时钟信号的组织结构和驱动方式进行优化选择，可减少时钟信号网络所消耗的功耗。

● 晶体管尺寸优化

晶体管尺寸优化的方法与门尺寸优化的方法类似。由于电路级设计获得了布局布线后的物理信息，晶体管尺寸的优化可进一步减小电路的功耗。

2.3 主流工业标准

根据上节的介绍可知，功耗管理涉及计算机系统设计的各个层次和方面。但是，无论具体的功耗管理技术处于何种层次，均可根据其实现的时机，将其归为：1) 在设计时间实现，如软硬件功能划分、时序优化、门尺寸优化等；2) 在运行时间实现，如时钟门控技术、可变电压技术、动态变频、动态功耗管理等技术。

对于设计时间实现的功耗管理技术，主要由硬件具体实现，软件几乎不再需要做任何工作。对于运行时间实现的功耗管理技术，软件则往往扮演了至关重要的角色，发挥了核心作用，故又称之为软件功耗管理技术。但是，通常情况下，软件功耗管理技术需要硬件提供相应的支持才能实现。在桌面计算机和移动便携式计算设备（主要基于 X86 架构）之中，当前普遍使用了基于动态功耗管理技术的功耗管理方案。主流的功耗管理方案有：基于 APM（Advanced Power Management）标准的功耗管理方案和基于 ACPI（Advanced Configuration and Power Interface）标准的功耗管理方案。

基于 APM 标准或 ACPI 标准的功耗管理方案以系统级的动态功耗管理为主要内容，其基本思路是：将系统的运行状态划分为若干具有不同功耗的运行模式，当系统负载较轻或处于空闲时，使系统进入具有较低功耗的模式，从而达到节省系统功耗的目的。本节将对 APM 标准和 ACPI 标准进行简要的介绍。

2.3.1 APM 功耗管理标准

APM（Advanced Power Management）标准[39]由 Intel 和 Microsoft 于 1992 年 1 月首次提出，其初衷是使运行在 IBM 兼容个人计算机（IBM Compatible Personal Computer）上的操作系统和 BIOS 能够相互配合工作，进而达到功耗管理的目的。目前，APM 标准最新版本是 1996 年 2 月发布的 V1.2。

APM 标准定义了五种系统级运行模式：全速运行（Full On），APM 使能（APM Enabled），APM 待机（APM Standby），APM 挂起（APM Suspend），关机（Off）。不同运行模式的主要区别在于：1) 该模式下系统的整体性能和功耗；2) 系统返回工作模式（默认为 APM 使能模式）的延时（latency）。其中，全速运行是系统性能最高、功耗最大的运行模式，其余模式的性能和功耗依次递减。

此外，APM 标准还定义了与特定硬件相关的功耗管理软件（即 BIOS 软件）与操作系统之间进行交互的接口，规定了应用程序、操作系统和 APM BIOS 之间如何分工协作。基于 APM 标准的功耗管理方案如图 2-1 所示。

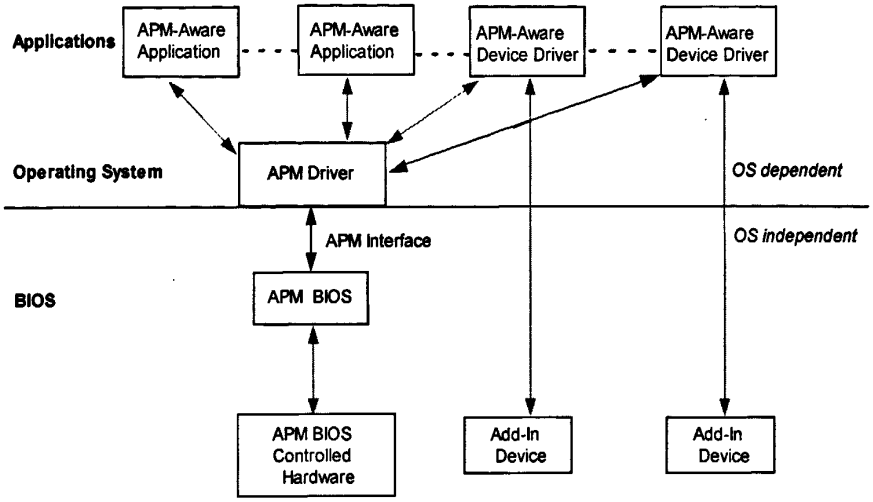

图 2-1 基于 APM 标准的功耗管理方案

2.3.2 ACPI 功耗管理标准

ACPI（Advanced Configuration and Power Interface）标准[40]由 Hewlett-Packard，Intel，Microsoft，Phoenix 和 Toshiba 共同提出。其设计初衷是为了替代 APM 功耗管理标准，为计算机工业界提供一种面向操作系统的、稳定（robust）的主板设备配置和功耗管理通用接口。ACPI 标准于 1996 年 12 月推出了 V1.0 版，当前最新版本是 2010 年 4 月发布的 V4.0a 版。

ACPI 标准将系统的全局运行状态划分为五种系统级运行模式：G0 Working 模式、G1 Sleeping 模式、G2 Soft Off 模式、G3 Mechanical Off 模式和 Legacy 模式。其中，对于 G1 Sleeping 模式，ACPI 标准又将其进一步细化为 S1 至 S5 五个子模式。

与 APM 标准类似，ACPI 标准定义了软件和硬件之间的通用接口，以及软件和硬件之间的交互行为。基于 ACPI 标准的功耗管理方案如图 2-2 所示。

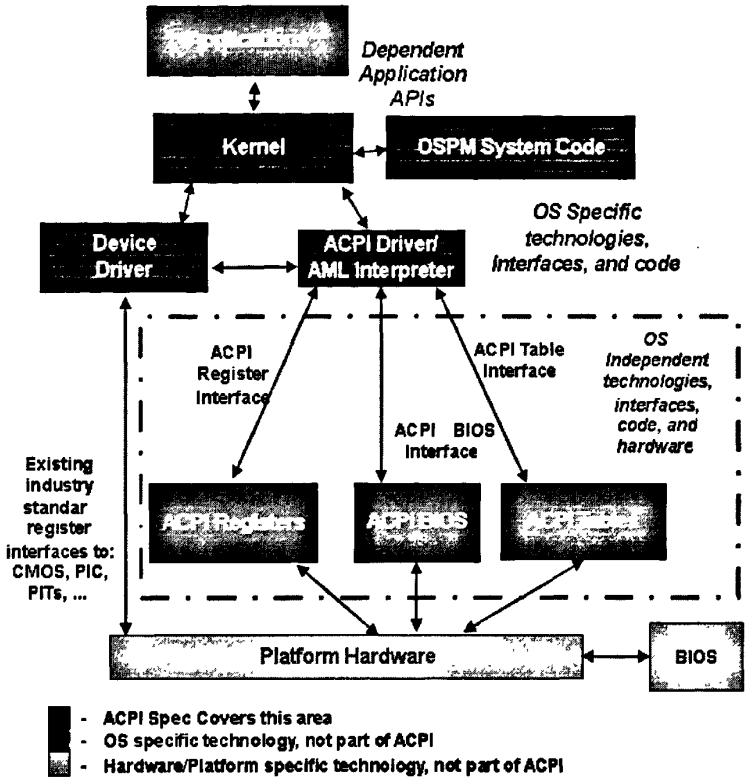

图 2-2 基于 ACPI 标准的功耗管理方案

作为 APM 标准的替代者，ACPI 标准主要是为了克服 APM 标准的以下不足：

1. 兼容性：APM 标准将功耗管理的所有核心功能（如功耗管理的策略和直接控制系统模式转换的代码）均放在 APM BIOS 中实现，造成了对 BIOS 的过度依赖。由于 BIOS 可能由不同的厂家实现，且不同的 BIOS 实现可能采用不同的功耗管理策略和机制，故可能造成了 BIOS 与操作系统不兼容的情况发生。

2. 可扩展性：APM 标准对较新的硬件支持不足，如 USB 设备和 1394 设备。

3. 功耗管理的有效性：APM 标准将功耗管理的核心代码放置在 BIOS 中实现。但由于 BIOS 在计算机系统中所处的层次较低，不能较好的获取功耗管理所需要的必要信息（如系统负载，应用程序运行特性等）。此外，由于 APM 不能对较新的硬件提供较好的功耗管理支持。这些不足将导致 APM 不能对计算机系统进行有效的功耗管理。

4. BIOS 开发复杂性：在实际应用中，BIOS 代码通常受到会存储、运行环境的限制。将复杂的功耗管理代码放置在 BIOS 中无疑将会增加 BIOS 的复杂性和代码量，进而增加了 BIOS 的开发难度。

5. 对主板设备的配置支持不足：APM 标准主要是为了支持软件系统级动态功耗管理而提出的，因此对主板设备的初始配置支持不是太好。但在实际应用中，对这一功能具有强烈的需求。

为此，ACPI 标准特意将功耗管理的核心代码从 BIOS 移入操作系统之中，同时还增加了对主板设备的配置支持。但是，基于 ACPI 标准的功耗管理方案仍然严重依赖于 BIOS 实现。

2.4 Linux 内核中的功耗管理技术

Linux 操作系统内核[41][42]是一个由 C 语言和汇编语言编写、符合 POSIX 标准的类 Unix 操作系统内核。由于它开放源码、支持多任务、具有较高的性能，并且可以免费下载使用，因此获得了开发者和计算机厂商的青睐。目前，Linux 操作系统内核在大型计算机、桌面个人计算机、移动便携式设备之中得到广泛运用。

当前，Linux 内核中已经实现的功耗管理技术主要有[43][44][45][46][47][48][49]：基于 APM 标准或 ACPI 标准的系统级动态功耗管理机制、基于 ACPI 标准的 CPU 动态变频变压技术。

2.4.1 系统级动态功耗管理机制

基于 APM 标准和 ACPI 标准实现的 Linux 系统级动态功耗管理机制严格按照 APM 和 ACPI 标准中的相关规定，完成并实现了相应标准中操作系统负责实现的相关功能。基于 APM 和 ACPI 标准的 Linux 系统级动态功耗管理分别如图 2-3、图 2-4 所示。

从图 2-3 中可以看出，Linux 将 APM BIOS 抽象为 APM BIOS 字符设备，应用程序、内核系统级动态功耗管理机制和设备驱动程序均通过 APM BIOS 字符设备提供的接口与 APM BIOS 进行通信。整个功耗管理方案中，APM BIOS 处于核心地位，内核系统级动态功耗管理机制和设备驱动程序协助 APM BIOS 完成系统运行模式的切换。

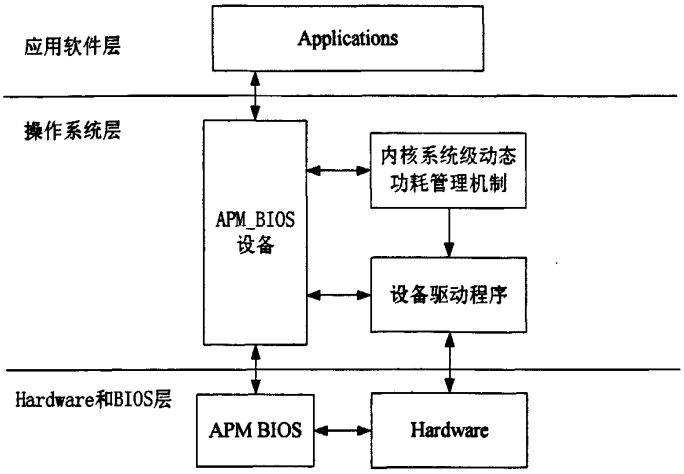

图 2-3 基于 APM 标准的 Linux 系统级动态功耗管理

从图 2-4 中可以看出，基于 ACPI 标准的 Linux 系统级动态功耗管理中，操作系统的系统级动态功耗管理机制在整个功耗管理中处于主导地位，ACPI 驱动负责与 ACPI BIOS 和硬件通信，协助操作系统中的系统级动态功耗管理机制完成系统模式的切换。

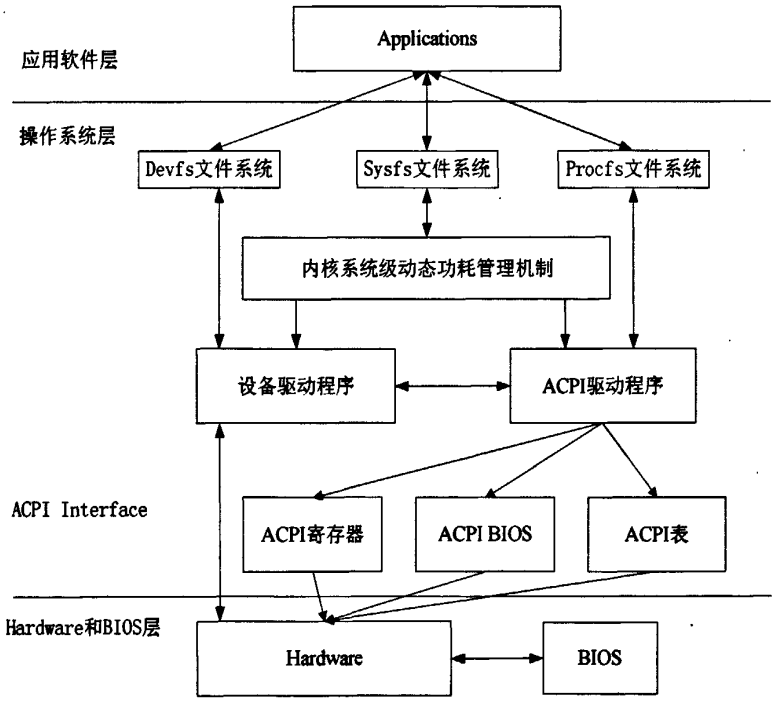

图 2-4 基于 ACPI 标准的 Linux 系统级动态功耗管理

2.4.2CPU 动态变频变压技术

动态变频技术和动态变压技术均是有效的功耗管理技术，其基本思想是在系

统负载较轻、对系统性能要求不高的情况下，动态降低 CPU 的运行频率和系统的电源电压。但是，对于 CPU 而言，仅仅依靠降低其运行频率并不能有效增加其单位能量所能执行的指令数目（Instructions per energy consumption）。为此，在实际使用中，CPU 动态变频技术和动态变压技术常常需要结合使用，以提高系统能量利用效率。

目前，Intel 公司在其设计、生产的 CPU 中运用了 Enhanced SpeedStep 技术，AMD 公司则在其产品中使用了 PowerNow 和 Cool&Quiet 技术，为软件实现 CPU 动态变频变压技术提供了支持。在 Intel 和 AMD 公司的引领下，许多其它平台的 CPU 厂商如 PowerPC、ARM、SPARC 和 SuperH 等处理器中也在其产品中提供了类似的支持。

为了利用处理器提供的动态变频变压支持，Linux 内核开发人员在内核之中新增加了 cpufreq 子系统。cpufreq 子系统的结构如图 2-5 所示。

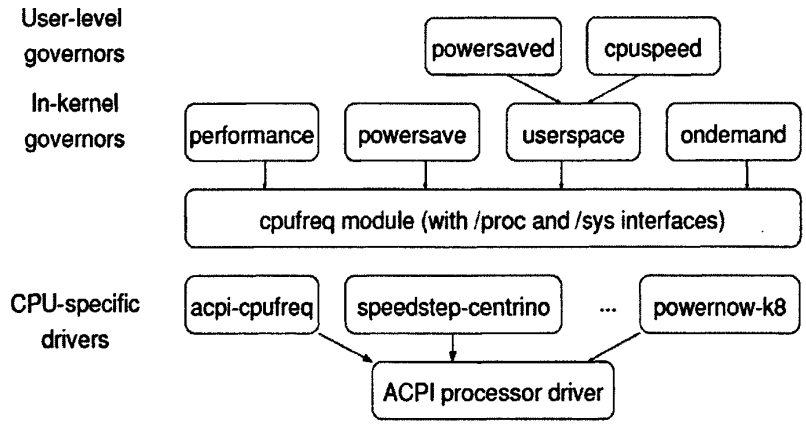

图 2-5 Linux cpufreq 子系统结构

从图 2-5 中可以看出，cpufreq 子系统在设计上主要分为以下三个层次：

- cpufreq 模块（cpufreq module）：实现了与具体 CPU 无关的通用 CPU 变频变压机制。
- CPU 相关的驱动（CPU-specific drivers）：与具体 CPU 相关的驱动程序，负责对具体 CPU 进行变频变压操作，向 cpufreq module 层提供底层支持。通常，CPU 驱动程序由相关的 CPU 生产厂商提供。
- 管理程序（governors）：利用 cpufreq 模块提供的机制，根据系统负载和一定的策略，决定系统合适的目标运行频率。

这种分层的设计带来的好处是：使得管理程序和与 CPU 相关的变频驱动程序的开发可以相互独立进行，并在最大限度上实现代码重用，内核开发人员在编写和试验新的管理程序时不会再陷入到某款特定 CPU 的变频技术的硬件实现细节中去，而 CPU 生产厂商在向 Linux 内核中添加支持其特定的 CPU 变频技术的代码时只需提供一个相对来说简单了许多的驱动程序，而不必考虑在各种不同的应用场景中如何选择合适的运行频率这些复杂的问题。

2.4.3 Linux 内核对不兼容 APM 和 ACPI 标准的平台的支持

由于 Linux 中的系统级动态功耗管理机制和变频变压技术均依赖于 APM 标准和（或）ACPI 标准，因此，对于不兼容 APM 或 ACPI 标准的平台而言，Linux 内核并不能提供直接而有效的功耗管理支持。

但是，这些不兼容 APM 和 ACPI 标准的平台对在 Linux 中实现功耗管理的相关支持有着强烈的需求。以嵌入式计算机为例，由于自身原因，嵌入式计算机通常没有与 APM 和 ACPI 标准兼容的硬件和 BIOS 软件，但是由于它们的应用领域对功耗极为敏感，因此这些计算机上常常在硬件层面上对功耗管理提供了支持，同时希望在 Linux 内核之中能够利用这些硬件支持，以便在该平台上实现类似于兼容 APM 和（或）ACPI 平台上相应的功耗管理功能。

为了达到对不兼容 APM 和 ACPI 标准的平台提供功耗管理支持的目的，Linux 内核预留了相关接口，允许各个平台根据需要添加相关的代码，进而在 Linux 中实现对该平台的功耗管理支持。

此外，除 Linux 内核自身为不兼容 APM 或 ACPI 标准的计算机系统提供基本的基本支持之外，第三方开发人员也发起了面向嵌入式系统功耗管理的 DPM（Dynamic Power Management）项目[50]。

DPM 项目是 IBM Austin Research Lab 和 MontaVista Software 于 2002 年发起了开源项目，期望能够为运行 Linux 操作系统的嵌入式计算机添加动态功耗管理功能。但是，DPM 至今仍不成熟，没有在嵌入式计算机上得到广泛应用。当前，DPM 项目的最新代码是发布于 2006 年 4 月的针对 Linux2.6.16 内核的补丁包。基于 DPM 的功耗管理系统如图 2-6 所示。

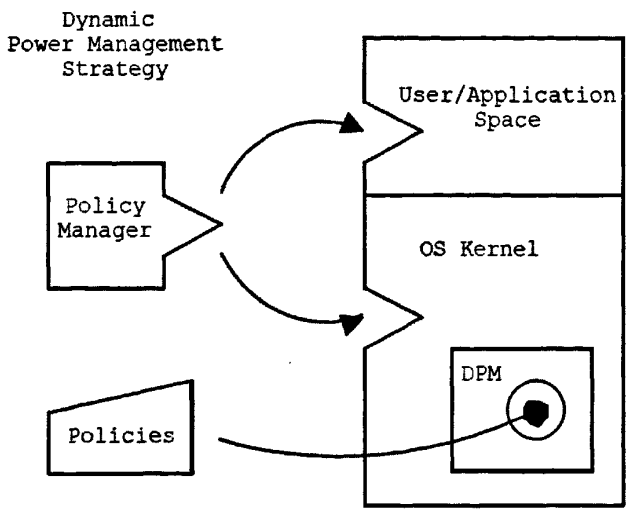

图 2-6 基于 DPM 的功耗管理系统

根据上述分析，由于 PKUnity-3 系统芯片平台并未实现与 APM 和 ACPI 标准兼容的功耗管理模块硬件和 BIOS 软件，且 DPM 项目仍不成熟（故不能将其移植到 PKUnity-3 系统芯片平台之上），因此，为了实现 PKUnity-3 系统芯片平台上的功耗管理，只有利用 Linux 内核提供的基本框架，结合 PKUnity-3 系统芯片自身特点，在 Linux 内核之中设计与实现面向 PKUnity-3 系统芯片平台的功耗管理的相应机制。

2.5 本章小结

功耗是计算机的重要属性，功耗管理是计算机领域当前的研究热点。本章首先介绍了功耗的组成和常见的功耗管理技术，然后简要介绍了工业界主流的功耗管理标准(APM 标准和 ACPI 标准)，并分析了它们的不足之处，最后介绍了 Linux 操作系统中的各种功耗管理技术。本章内容为本文后续各章提供了理论背景和设计参考。

第3章 系统级动态功耗管理机制设计与实现

在本文中，系统级动态功耗管理是指：将计算机系统的全局运行状态划分为工作模式和多个具有不同功耗、性能的低功耗模式，以便当系统负载较低、对性能要求不高时，使系统转入具有较低功耗和性能的低功耗模式，从而达到节省系统功耗的目的。系统级动态功耗管理包含两大部分：系统级动态功耗管理机制（负责按照系统级动态功耗管理策略的指示，完成工作模式与低功耗模式之间的切换工作）和系统级动态功耗管理策略（即系统级动态功耗管理机制的使用策略，决定系统何时转入何种低功耗模式），涉及功耗管理模块硬件、操作系统和应用软件多个层次。

由于系统级动态功耗管理机制需要直接操作功耗管理模块硬件、维护好整个系统运行的软、硬件上下文（包括硬件寄存器和操作系统自身数据结构等内容），因此在操作系统之中实现系统级动态功耗管理的机制常常更加有效。

基于这一思路，本文在操作系统内核——Linux-uc 之中设计并实现了系统级的动态功耗管理机制，为面向 PKUnity-3 系统芯片的系统级动态功耗管理提供了支持。

3.1 任务特点

操作系统系统级动态功耗管理机制是指将系统级动态功耗管理中的机制部分置于操作系统之中实现。操作系统系统级动态功耗管理机制的主要工作包括：对于系统所支持每一种低功耗模式，完成进入该模式前（本文称这一阶段为挂起阶段）的准备工作，退出该模式（本文称之为恢复阶段）时的恢复工作。由于不同低功耗模式的特点不同，因此进入、退出该模式时所需完成的准备和恢复工作也不相同。系统工作模式和低功耗模式的转换示意图如图 3-1 所示。

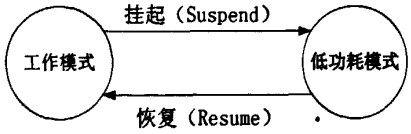

图 3-1 工作模式和低功耗模式之间的转换

但由于系统级动态功耗管理机制涉及整个计算机系统中所有硬件设备和软件上下文，因此具有涉及面广、处理复杂、难于调试等特点。本章将在接下来的内容中，首先结合PKUnity-3系统芯片硬件特性，明确PKUnity-3系统芯片平台需要支持的系统运行模式，然后对系统需要支持的各运行模式分别进行分析、设计与实现，并对各模式设计、实现时需要重点考虑的难点进行总结。

此外，由于操作系统系统级动态功耗管理机制仅仅实现了系统级动态功耗管理的机制部分，因此它必须同功耗管理的策略部分才能实现系统级的动态功耗管理。基于本文实现的系统级动态功耗管理机制的系统级动态功耗管理参见第五章相关内容。

3.2 系统支持的运行模式

系统支持的低功耗模式是系统级动态功耗管理的工作基础。本节将在对比APM标准和ACPI标准支持的运行模式的基础之上，结合PKUnity-3系统芯片功耗管理模块硬件提供的支持，确定面向PKUnity-3系统芯片平台的系统级动态功耗管理要支持的系统运行模式。

3.2.1 APM标准和ACPI标准的系统运行模式对比

在本文第2.3节中，对APM和ACPI功耗管理标准[39][40]支持的系统运行模式进行了较为详细的介绍。APM和ACPI标准支持的系统运行模式的对应关系见表3-1。

从表3-1中可以看出，APM和ACPI标准均将其运行模式划分为三大类：工作模式（APM Enabled和ACPI G0）、低功耗模式（APM Standby、APM Suspend和ACPI S1~S4）和关机模式（APM Off和ACPI G2~G3）。其中，APM Enabled基本等价于ACPI G0，APM Standby基本等价于ACPI S1，APM Suspend基本等价于ACPI S2~S4，APM Off则基本等价于ACPI的G2、G3。

尽管APM标准和ACPI标准都对各自定义的系统运行模式进行了较为详细的描述，但是它们并没有要求支持它们的所有计算机系统必须实现其标准定义的全部运行模式，而且它们对各自运行模式的具体实现细节也没有做硬性的规定。因此，不同平台可以根据实际需要和具体的硬件特性选择性的进行实现。

表 3-1 APM、ACPI 系统运行模式对应关系

APM	ACPI		特性描述
Full On	——		1.属于工作模式 2.系统中没有进行功耗管理，系统中所有设备均开启
APM Enabled	G0 Working		1.属于工作模式 2.系统中存在功耗管理 3.CPU 时钟根据功耗管理需要而降频或停止 4.系统中设备根据功耗管理需要而进行相应管理
APM Standby	G1	S1	1.系统可能没有工作 2.属于具有较低功耗的模式，节省了部分功耗 3.CPU 被降频或停止 4.系统运行中的状态和参数不会丢失 5.系统中大部分设备处于低功耗状态 6.系统恢复到工作状态迅速 7.Resume 事件或定时器事件将使系统返回到工作模式 8.为了使系统返回到工作模式可能需要用户参与 9.处于此模式时，中断请求能够得到正常处理。为此，系统可能会暂时唤醒 CPU，当中断处理完毕后，系统可能再次进入此模式
APM Suspend		S2	1.系统没有工作 2.CPU 运行状态和 Cache 内容会丢失 3.系统中大部分设备处于低功耗状态 4.为了使系统返回到工作模式可能需要用户参与 5.当唤醒事件导致系统退出此模式时，系统将复位（Reset），即 CPU 将从处理器唤醒向量（Reset Vector）处开始执行
		S3	1.系统没有工作 2.系统除内存以外的运行状态均会丢失（包括 CPU、Cache、各外设控制器状态），系统需要保存丢失的信息至内存之中 3.系统中大部分设备处于低功耗状态 4.为了使系统返回到工作模式可能需要用户参与 5.当唤醒事件导致系统退出此状态时，系统将复位（Reset），即 CPU 将从处理器唤醒向量（Reset Vector）处开始执行
		S4	1.系统没有工作，系统电源关闭 2.系统所有运行状态均会丢失，系统需要将丢失信息保存至非易失性存储设备之中 3.为了使系统返回到工作模式需要用户参与 4.系统再次通电后，系统将从非易失性存储设备之中读取保存的上下文，并将系统恢复到进入此状态之前的状态
Off	G2/S5		1.系统没有工作，系统功耗几乎为 0 2.系统所有运行状态均会丢失，但系统不将丢失信息保存至非易失性存储设备之中 3.为了使系统返回到工作模式需要用户参与 4.系统再次通电后，系统将重新启动并初始化
	G3 Mechanical Off		1.系统停止工作，系统电源关闭 2.系统所有运行状态均会丢失 3.系统再次通电后，系统将重新启动并初始化

通常，由于工作模式和关机模式不需要硬件提供特殊支持，且无需软件做任何修改，所以所有的系统级动态功耗管理系统均支持这两种运行模式；但是对于低功耗模式，则由于必须依赖硬件提供的支持，系统级动态功耗管理系统往往仅实现了标准规定的低功耗模式中的一个子集。

3.2.2 系统支持的运行模式

PKUnity-3 系统芯片功耗管理硬件支持运行（Run）、空闲（Idle）和睡眠（Sleep）三种运行模式。结合功耗管理模块硬件支持，借鉴 APM 标准和 ACPI 标准支持的系统运行模式，面向 PKUnity-3 系统芯片的系统级动态功耗管理机制支持的系统运行模式定义如表 3-2 所示。

表 3-2 面向 PKUnity-3 系统芯片的系统级动态功耗管理支持的运行模式

系统模式	特性描述
工作模式（Working）	1.系统中唯一的工作模式 2.系统全速运行 3.系统上电，操作系统完成启动之后系统即处于此模式
空闲模式（Idle）	1.属于低功耗模式，节省功耗较少，但系统恢复到工作模式迅速 2.CPU 时钟停止 3.系统运行中的状态和参数不会丢失 4.系统中仅 CPU 停止工作，处于低功耗状态 5.系统唤醒后将返回到工作模式
睡眠（Sleep）	1.属于低功耗模式，节省功耗较多，系统恢复到工作模式时间较长 2.CPU 和系统中大部分功能模块的时钟停止 3.系统运行中 CPU 和各功能模块的状态和参数丢失，故需要保存至内存中 4.系统中 CPU 和大部分功能模块停止工作，处于低功耗状态 5.系统唤醒后将返回到工作模式
休眠（Hibernation）	1.属于低功耗模式 2.系统电源关闭，功耗为 0，系统恢复到工作模式时间很长 3.系统运行中的状态和参数丢失，故需要保存至非易失性存储设备之中 4.为了使系统返回到工作模式需要重新开机 5.系统再次通电后，系统将从非易失性存储设备之中读取保存的上下文，并将系统恢复到进入此模式之前的状态
关机（Off）	1.系统停止工作，系统电源关闭 2.系统所有运行状态均会丢失 3.系统再次通电后，系统将重新启动并初始化，进入到工作模式 4.系统关机之后即处于此模式

其中，工作（Working）模式为系统上电、操作系统完成启动后系统所处的

模式，为系统唯一的工作模式，关机（Off）模式为用户关闭计算机并切断电源后系统所处模式。这两个模式无需操作系统对其进行任何特殊处理即已实现。

空闲（Idle）、睡眠（Sleep）和休眠（Hibernation）模式为系统的低功耗模式，需要操作系统利用硬件提供的基本支持加以实现。低功耗模式与对应的硬件物理实现关系见表 3-3。

表 3-3 面向 PKUnity-3 系统芯片的系统低功耗模式的物理实现

模式名	硬件物理实现
空闲（Idle）	1.利用功耗管理模块硬件提供的空闲模式加以实现； 2.以任意硬件中断作为唤醒事件。
睡眠（Sleep）	1.利用功耗管理模块硬件提供的睡眠模式加以实现； 2.需将内存降频，并设置为自刷新状态，同时关闭系统 PLL 输出； 3.以用户“按下电源键”事件作为唤醒事件； 4.需 U-boot 提供支持。
休眠（Hibernation）	1.利用关闭硬件电源加以实现； 2.以用户“重新开机”事件作为唤醒事件。

各系统运行各模式之间的转换关系如图 3-2 所示。由图可知，系统中所有的模式转换均在 PKUnity 工作模式和其他模式之间产生。

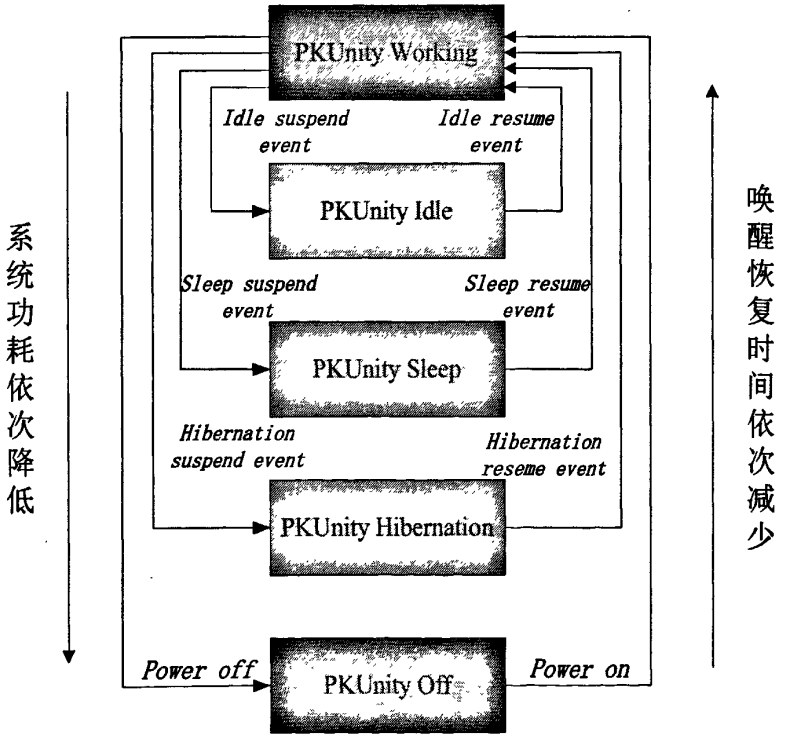

图 3-2 面向 PKUnity-3 系统芯片的系统运行模式转换关系

3.3 空闲模式

根据表 3-2、表 3-3 中对空闲模式的描述可知，在系统处于空闲模式时，仅 CPU 硬件由于其输入时钟停止而不再工作，系统中其余硬件模块仍然在其自身输入时钟的驱动之下正常工作。此时，PKUnity-3 系统芯片中的功耗管理模块硬件监测系统中其余模块发出的硬件中断。一旦监测到硬件中断发生，CPU 时钟将恢复输入，系统将返回到工作模式。此时，中断请求将会在工作模式下得到处理。

空闲模式分析如下：

1. 空闲模式下，CPU 时钟停止，故软件程序的执行亦将停止。由于此时 CPU 仍然供电，因此 CPU 内部寄存器、Cache、及其内部的硬件运行状态均不会丢失，故一旦 CPU 时钟输入恢复，CPU 硬件将会在原先基础之上继续工作，软件将会恢复继续执行。因此，仅从 CPU 角度讲，系统进、出空闲模式对系统中运行的软件程序是透明的。

2. 系统中除 CPU 之外，还有多个其它外设与 CPU 并行工作，它们通过中断、DMA 或轮询方式与 CPU 协同。对于采用中断和 DMA 方式与 CPU 通信的外设，由于系统中仅 CPU 时钟停止，故它们自身的运行状态不受影响。当它们需要发出中断请求，请求 CPU 给予配合时，系统将被唤醒并返回到工作模式，它们的中断请求将得到处理。因此，空闲模式对于采用中断或 DMA 方式的外设来讲，也是透明的。对于采用轮询方式与 CPU 协同的外设，由于 PKUnity-3 系统芯片中仅集成了一个 CPU 核，故在同一时刻系统中仅会有一个控制流正在执行，若 CPU 正在执行进、出空闲模式的控制流，则此时系统就一定不会去轮询某个外设，因此，空闲模式不会对采用轮询方式与 CPU 通信的外设产生影响。

3. 系统中其余模块在空闲模式期间上下文信息不会丢失，不会受到影响。

综上，系统进出空闲模式对系统中各模块的影响较小，无需保存、恢复系统软硬件上下文。为此，在操作系统中实现空闲模式，只需实现进入空闲模式的相关代码即可。

在 Linux-uc 中，进入空闲模式由 puv3_cpu_do_idle 函数实现，其流程如图 3-3 所示。

值得注意的是，由于空闲模式下仅停止了 CPU 的输入时钟，且系统每次进

出空闲模式时，系统处于空闲模式下的持续时间较短（空闲模式可由任意硬件中断唤醒，故其处于空闲模式的持续时间不超过系统中相邻两次中断之间的间隔时间），因此其所节省的功耗较少。但是，也正是由于空闲模式仅停止了 CPU 时钟输入，对系统软硬件影响较小，故其进入、恢复时间也相应较短。

空闲模式在系统空闲时间持续较短，且系统空闲、繁忙状态交替频繁的情况下较为有效。

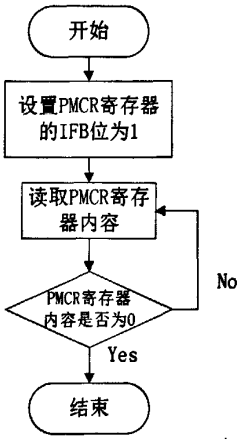

图 3-3 puv3_cpu_do_idle 函数流程

3.4 睡眠模式

睡眠模式是一个节省功耗较多的低功耗模式，本节将首先分析、介绍睡眠模式的特点，然后给出睡眠模式在 Linux-uc 内核中的设计与实现。

3.4.1 睡眠模式分析

根据表 3-2、表 3-3 中对睡眠模式的描述可知，当系统处于睡眠模式时，系统中 GPIO、RTC 和功耗管理模块自身在 RTC 时钟驱动之下继续工作，内存被降频并处于自刷新状态，系统 PLL 停止输出，系统中的其他模块均停止工作。当功耗管理模块硬件监测到用户“按下电源键”事件时，功耗管理模块硬件将会唤醒系统，同时发出全局的复位信号。此时，由于复位，CPU 将而从 BIOS 所在的地址空间取指令执行，而系统除内存和非易失性存储设备中内容以外的全部运行信息将会全部丢失。

睡眠模式分析如下：

1. 系统退出睡眠模式时，CPU 将复位，导致 CPU 中的系统运行上下文全部丢失，故应在系统进入睡眠模式之前保存、退出睡眠模式之后恢复这部分信息。由于睡眠模式下，内存处于自刷新状态，内存内容不会丢失，因此可将 CPU 运行的上下文信息保存到内存之中。需要保存的 CPU 运行的上下文信息包括：定点处理器核中的通用寄存器、程序状态寄存器(PSR)、程序计数器(PC)、链接寄存器(LR)、堆栈指针寄存器(SP)，浮点协处理器中的通用寄存器和控制寄存器、系统控制协处理器中的控制寄存器（如 MMU、Cache 使能寄存器、页表基址寄存器等）等内容。

2. 系统退出睡眠模式时，Cache 将复位，Cache 中的内容也会因为全局复位而丢失。因此，需要在系统进入睡眠模式之前将 Cache 中的内容同步到内存之中，以避免错误。

3. 系统退出睡眠模式时，系统中各外设将被复位，各外设的运行状态及其中的寄存器内容也会丢失。为了保证系统退出睡眠模式后，各外设能够恢复之前状态并继续正常工作，必须在进入睡眠模式之前停止各外设的工作，然后再保存各设备的运行上下文信息。

4. 对于系统中的其他模块，其运行的上下文信息也会丢失。因此，应根据需要在进入睡眠模式前和退出睡眠模式后进行相应的保存和恢复工作。

5. 系统退出睡眠模式后，CPU 因复位而从 U-boot 处开始取指令执行。为了使 CPU 能够跳转到执行系统恢复工作的代码处继续执行，必须在系统进入睡眠模式之前，将执行恢复工作的代码的首条指令的地址保存在进、出睡眠模式前后内容不变的变量或寄存器之中，同时在 U-boot 代码中添加相应的判断逻辑，以便睡眠模式唤醒后，CPU 能够从 U-boot 代码跳转到系统执行恢复工作的代码处执行。注意，此处保存的恢复代码首条指令地址应为物理地址，因为 CPU 复位后，将运行在实地址模式。

综上，由于睡眠模式下大部分硬件模块的运行上下文信息均会丢失，对系统软硬件的运行产生了较大影响，因此为了保证系统退出睡眠模式后依然能够继续正常运行，操作系统需要在进入和退出睡眠模式时做大量的工作，系统进入、退出此模式所需时间较长。但是，也正是睡眠模式下仅少量模块仍在工作，因此，此模式节省功耗较多。睡眠模式适用于系统中空闲时间持续较长的情况下使用。

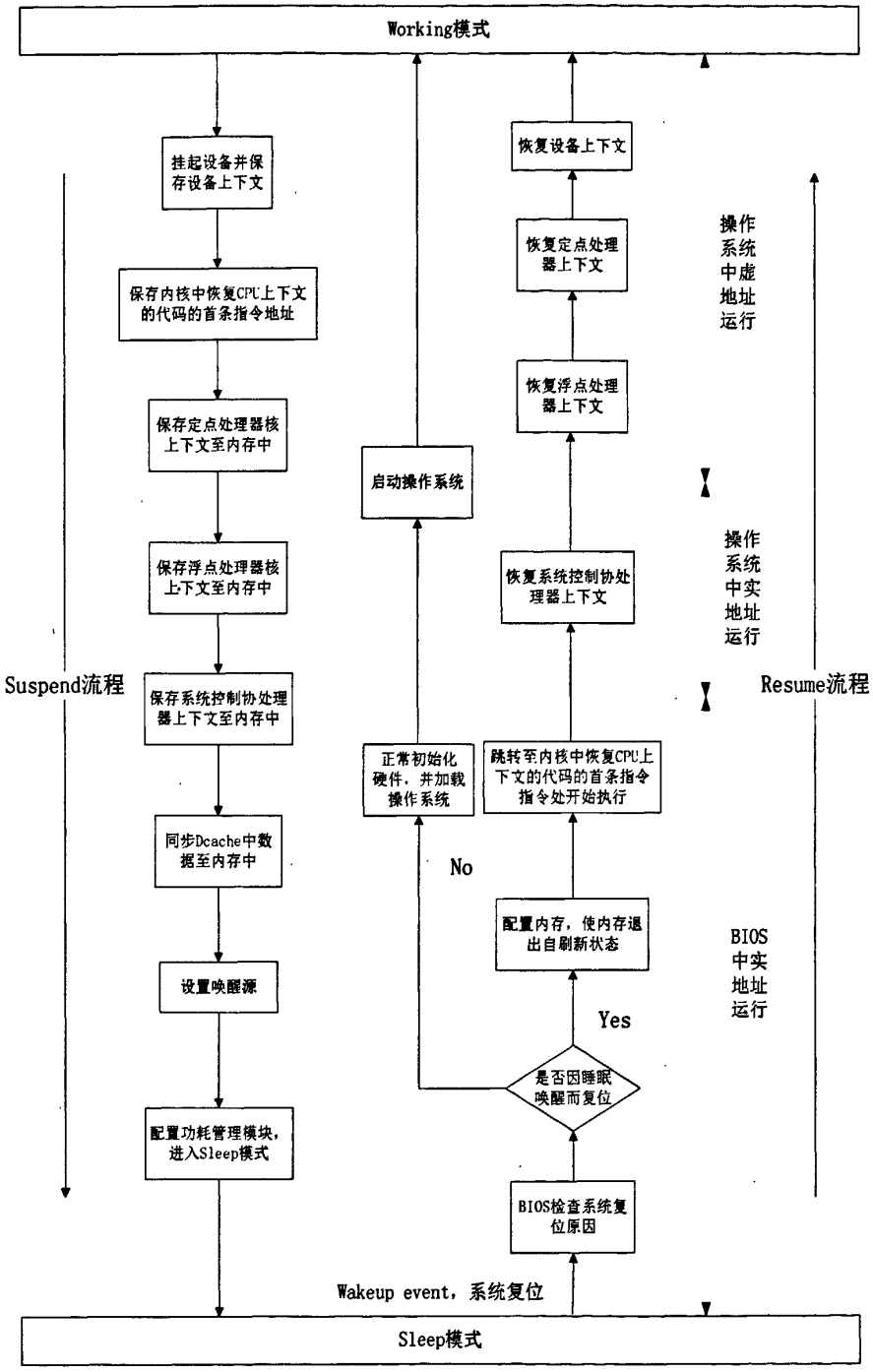

图 3-4 进入、退出睡眠模式流程

结合上述对睡眠模式的分析，系统进入和退出睡眠模式需要完成的工作流程如图 3-4 所示。

3.4.2 睡眠模式设计与实现

如本文 2.4 节所述，Linux 为不兼容 APM 和 ACPI 标准的平台提供了基本的功耗管理支持，允许不同平台结合自身实际添加相应代码，进而实现相应的功耗管理功能。Linux 内核中进入和退出睡眠模式的处理流程[47]如图 3-5、图 3-6 所示。

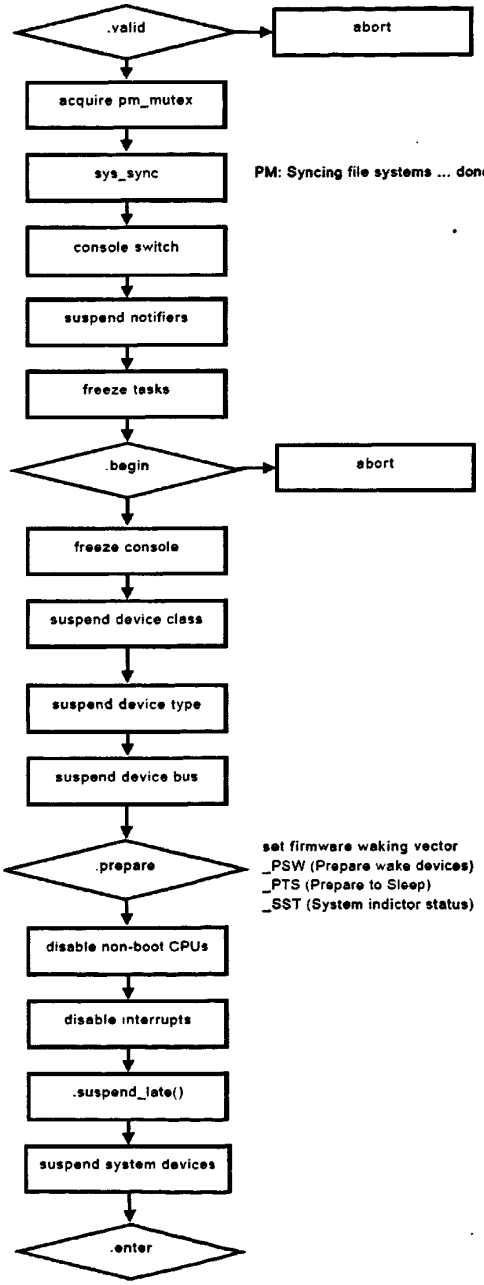

图 3-5 Linux 中进入睡眠模式的框架

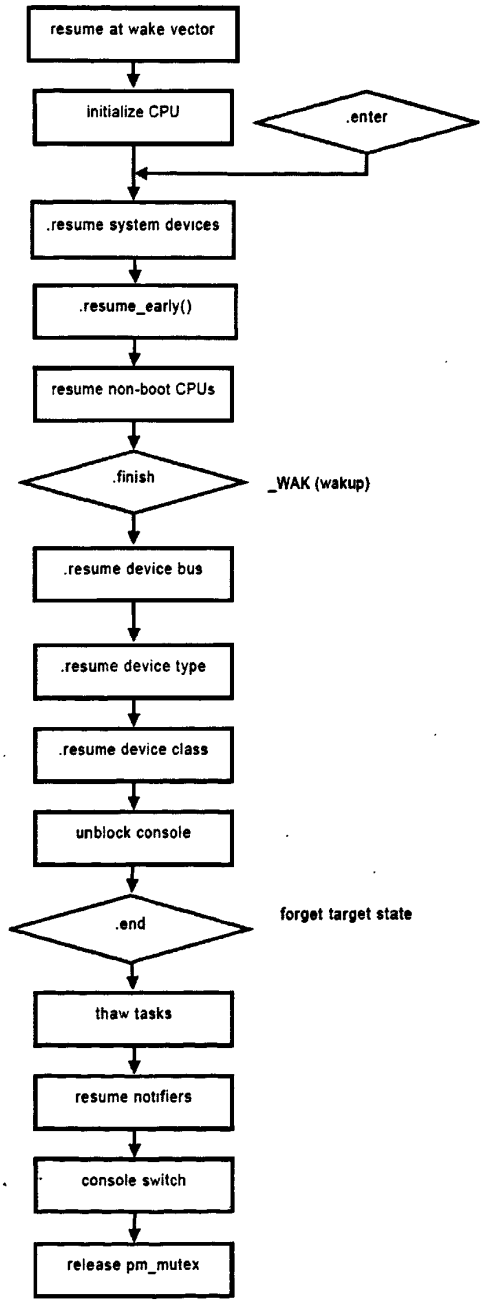

图 3-6 Linux 中退出睡眠模式的框架

从图 3-5、图 3-6 中可以看出，Linux 内核实现了睡眠模式中的与平台无关的通用框架，并为不同平台的睡眠模式保留了相关的函数接口（如图中的.enter，.valid，.prepare，.finish 等函数），允许不同平台根据自身特点实现睡眠模式。函数接口主要包含两个部分：1）与整个平台相关的功耗管理函数结构体 platform_suspend_ops 和相关的底层恢复函数；2）与各个设备相关的驱动程序自身的 suspend 和 resume 函数。其中，platform_suspend_ops 结构体和相关的底层恢复函数提供了系统进入和退出睡眠模式时，需要平台自身完成的必要操作。而各个设备驱动程序提供的 suspend 和 resume 函数则分别用于：在进入睡眠模式前挂起该设备，并保存相应的设备状态；退出睡眠模式后，恢复设备状态，并使其恢复运行。本文将主要讨论 platform_suspend_ops 和相关的底层恢复函数的设计与实现，而设备驱动程序自身的 suspend 和 resume 函数则由于涉及的设备较多，且不同设备处理方式各异，因此其设计与实现由各个驱动程序的开发人员负责，本文不做过多的讨论。

结合实际需要，在 PKUnity-3 系统芯片平台中定义 platform_suspend_ops 类型的结构体变量 puv3_pm_ops，并由其提供相应的函数操作。其具体定义如下：

```
struct platform_suspend_ops puv3_pm_ops = {
    .valid = puv3_pm_valid,
    .enter = puv3_pm_enter,
    .prepare = puv3_pm_prepare,
    .finish = puv3_pm_finish
}
```

其中，puv3_pm_valid 函数负责检查系统是否欲进入睡眠模式，若系统欲进入的模式不是睡眠模式，此函数将会返回错误信息，并终止系统进入睡眠模式的流程；puv3_pm_prepare 函数负责在系统进入睡眠模式前，将系统退出睡眠模式后，执行恢复操作的函数的首条指令地址（即函数 puv3_cpu_resume 的首条指令地址）的物理地址保存到功耗管理模块硬件的特殊寄存器中，以便系统唤醒复位后，能够从 U-boot 跳转到该地址处执行恢复处理器上下文的相关代码；puv3_pm_finish 函数负责在系统退出睡眠模式后，完成一些扫尾工作。

puv3_pm_enter 函数较为复杂，主要完成：1) 进入睡眠模式前的特殊硬件和

CPU 上下文的保存等工作；2) 配置功耗管理模块硬件和内存控制器硬件，使系统硬件进入睡眠模式，内存进入自刷新状态。其主要流程如图 3-7 所示。其中 puv3_cpu_save 函数和 puv3_cpu_restore 函数用于保存、恢复系统睡眠过程中特殊的硬件配置信息（这类硬件没有专门的驱动程序，因此也没用相应的 suspend 和 resume 函数）。puv3_cpu_suspend 函数完成系统进入睡眠模式前最后的准备工作，定点处理器、浮点和系统控制协处理器上下文均被保存到内存栈之中。

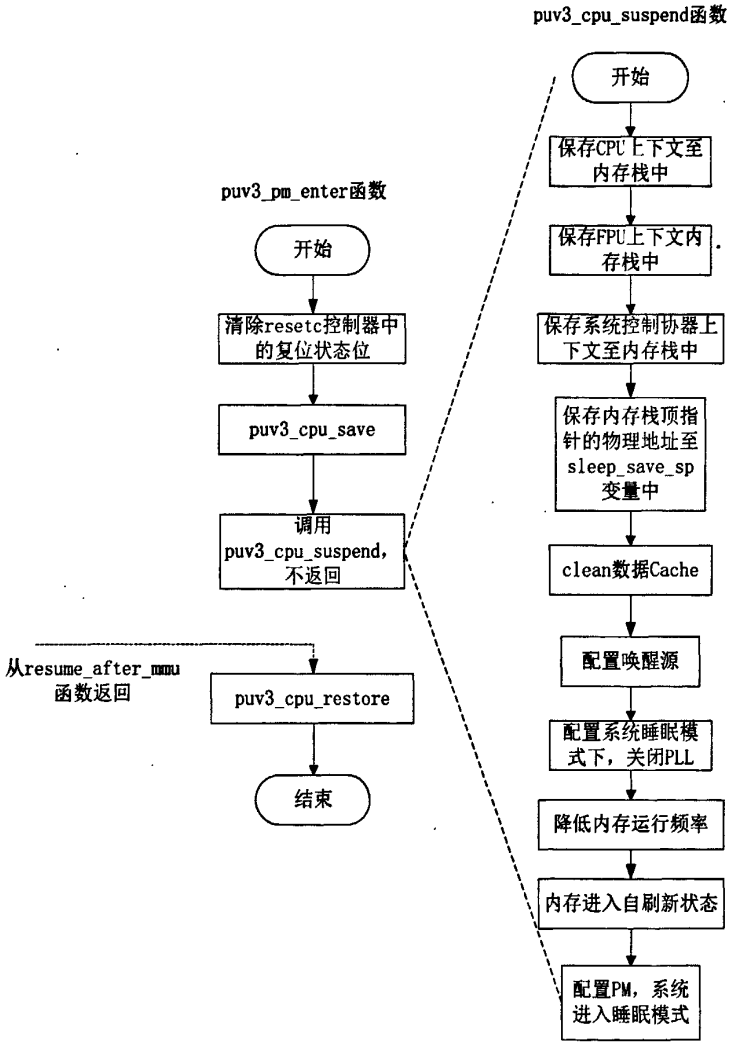

图 3-7 puv3_pm_enter 函数的处理流程

puv3_pm_ops 变量将在操作系统内核初始化过程中，由 puv3_pm_init 函数调用 suspend_set_ops 函数将其作为系统默认的平台功耗管理操作函数。

系统退出睡眠模式时，系统将进行全局复位，CPU 将跳转到 U-boot 处取指令执行。因此，在 U-boot 程序中必须完成相应的判断工作，以使 CPU 能够跳转

到相应的恢复程序（即 puv3_cpu_resume 函数）处执行。U-boot 中的工作流程如图 3-8 所示。

puv3_cpu_resume 函数是系统退出睡眠模式时，执行恢复工作的首个函数，其主要工作为恢复系统控制协处理器、浮点协处理器和定点 CPU 的上下文，其主要流程如图 3-9 所示。

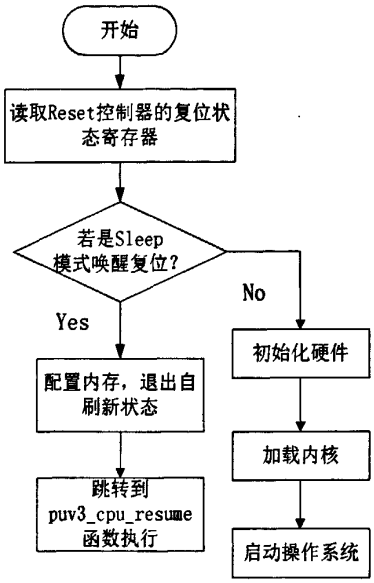

图 3-8 U-boot 程序的处理流程

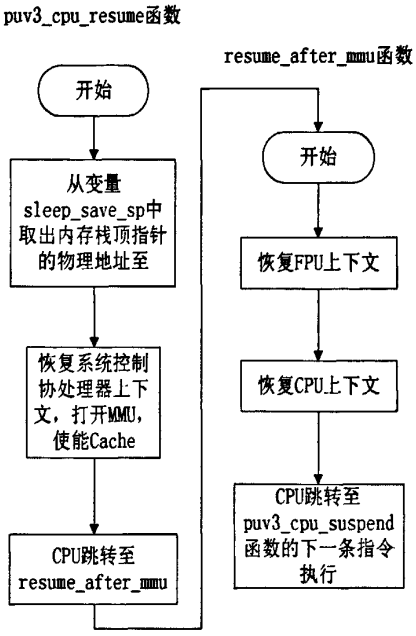

图 3-9 puv3_cpu_resume 函数的处理流程

3.4.3 睡眠模式设计实现难点

从图 3-4 中可以看出，睡眠模式的处理流程极其复杂，且涉及系统中几乎所有的硬件设备和软件上下文环境。睡眠模式的主要难点在于：

1.处理流程长，如果整个流程中的任意一个操作处理不当，将导致整个睡眠功能失效。因此，在睡眠模式的设计与实现中，必须仔细设计并实现每一个操作，并采用增量开发的方式，逐步确保整个流程的正确性。

2.涉及设备多，包含计算机系统中几乎所有的设备，如 CPU、FPU、系统控制协处理器、PCI 控制器、MMU、Cache、USB 控制器、PS2 键盘控制器、DE、网卡控制器、Nand 控制器等设备。对于在内核中具有专用驱动程序的硬件设备，睡眠模式需要在相应的驱动程序中设计并实现设备上下文保存和恢复工作；对于没有专用驱动程序的设备，则需要在睡眠过程中选择合适的时间点进行处理。在

对这些设备的处理过程中，若任意一个设备处理不当，均会导致整个睡眠功能的失效。因此，在睡眠模式的设计、实现过程中，也采用了增量的开发方式，即先保证系统运行所需的最小设备集能够正确实现睡眠功能，再逐渐增加其他外围设备的睡眠支持。

3.需要妥善处理 MMU、Cache 使能和关闭情况下的代码执行。在 PKUnity-3 系统芯片中，MMU、Cache 的使能由系统控制协处理器进行控制。在进入睡眠模式以前，MMU 和 Cache 均处于使能状态；但是当退出睡眠模式时，系统芯片将被全局复位，MMU 和 Cache 将因复位而关闭，CPU 将按物理地址进行寻址，并从 U-boot 处开始执行，而 Cache 中的所有数据将会丢失。因此，1）必须在进入睡眠模式前同步 Cache，保存系统控制协处理器内容至内存中；2）同时在退出睡眠模式后，首先将 CPU 从 U-boot 中跳转到操作系统执行恢复工作的首条指令处，在 MMU、Cache 关闭的情况下恢复系统控制协处理器上下文。当系统控制协处理器上下文恢复之后，MMU 和 Cache 均将使能，CPU 将按虚拟地址进行寻址，然后执行恢复 CPU、FPU 等其余的恢复工作。

4.需要解决内存处于自刷新状态下的 CPU 访存问题。为了尽可能地使系统在睡眠模式下具有较低的功耗，本文在系统进入睡眠模式前首先使内存处于自刷新状态，然后再配置系统芯片硬件进入睡眠模式。但是，当内存进入睡眠模式后 CPU 将不能再进行访存操作，因此，此时若再执行配置系统芯片硬件，将会使系统崩溃。为了解决这一问题，本文通过将配置系统芯片硬件进入睡眠模式的代码放置在指令 Cache 之中，解决了这一难题。

5.调试困难。作为内核通常使用的调试手段——通过打印语句向显示屏幕或 Uart 串口输出调试信息，则由于睡眠模式自身特点，在睡眠模式处理流程中的某个阶段不能使用，为了解决这个问题，本文在实际调试中，通过使用 GPIO 控制主板 LED 灯的方式来输出调试信息（赋予 LED 灯闪烁频率和开、关等状态以一定的语义信息，进而达到输出调试信息的目的）。

3.5 休眠模式

休眠模式是系统中节省功耗最多的模式，在此模式下，系统功耗几乎为 0。休眠模式除可应用于动态功耗管理外，还可以在加速系统启动等方面发挥作用。

本节将首先分析休眠模式的特点，然后给出休眠模式在 Linux-uc 中的设计与实现。

3.5.1 休眠模式分析

根据表 3-2、表 3-3 中对休眠模式的描述可知，当系统处于休眠模式时，系统电源停止供电，位于 CPU、FPU、系统控制协处理器、Cache、各外设控制器和内存中的系统运行信息将会全部丢失。因此，为了保证退出休眠模式后，系统能够返回到之前状态继续运行，操作系统必须在进入休眠模式之前，将系统该时刻的所有易失性的运行信息以快照(snapshot)的方式保存至非易失性存储设备——硬盘之中。当系统退出休眠模式（即用户重新接通系统电源并开机）时，操作系统将会在完成对系统软硬件初始化工作之后，检查硬盘中是否存在有效的系统快照。若存在有效快照，操作系统将会根据快照，将系统恢复到进入休眠模式前的状态。

休眠模式分析如下：

1. CPU 停止供电，导致 CPU 中的系统运行上下文全部丢失，故应在系统进入休眠模式之前保存这部分信息。由于休眠模式下，内存内容也会丢失，但硬盘中的数据不会丢失。因此一种可行的方案是先将 CPU 中的系统运行信息保存至内存中，然后将这部分信息连同内存中的其它数据保存至硬盘之中。在休眠模式下，需要保存的 CPU 运行的上下文信息与休眠模式下需要保存的信息相同，主要包括定点、浮点协处理器中相关寄存器的内容。当系统退出休眠模式时，系统将根据这些保存的上下文恢复 CPU 上下文。

2. Cache 停止供电，导致 Cache 中内容丢失。为此，需要在进入休眠模式前，将 Cache 中的内容同步到内存中去。

3. 外设停止供电，导致外设的运行状态和上下文信息丢失，故需要在进入休眠模式前挂起设备，并将设备的上下文信息首先保存至内存中，然后连同内存内容一起保存在硬盘之中。当系统退出休眠模式时，系统需要恢复设备的上下文信息，并使设备恢复运行。

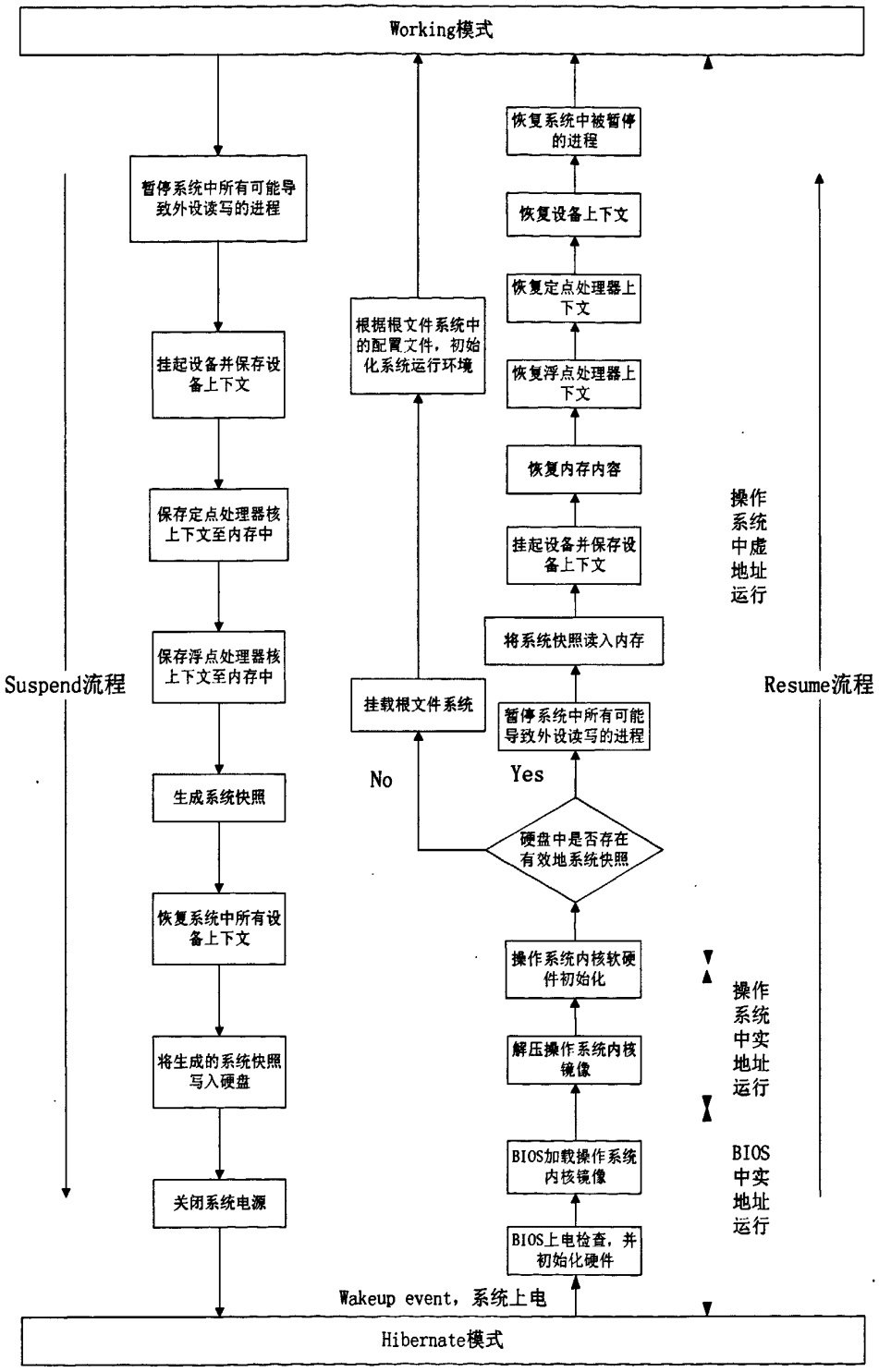

图 3-10 进入、退出休眠模式流程

4. 内存停止供电，内存内容丢失。因此，在进入休眠模式前，需要在系统空闲内存中，首先生成包含 CPU、外设上下文和内存本身有效数据的系统运行快照，然后再将快照保存至硬盘之中。当系统退出休眠模式时，将利用硬盘保存

的快照首先恢复内存中的有效数据，然后恢复 CPU 上下文和外设的上下文。

5. 除需要在快照中保存系统相关的易失性上下文信息外，操作系统还必须保证快照中保存的系统上下文和硬盘中其它数据的一致性（主要是快照和硬盘中的文件系统信息的一致性）。为此，操作系统必须防止系统快照生成之后，系统再对硬盘数据进行修改（如在硬盘之上创建、删除和修改数据，当然向硬盘写入系统快照的操作除外），以避免硬盘中的数据与系统快照中的数据因不一致而发生冲突。为了防止快照生成后，系统对硬盘进行不必要的操作进而修改硬盘数据，一种最为可行的方案是：在系统生成快照之前，将系统中所有可能引起硬盘操作的进程全部挂起。这样，当系统生成快照之后，由于所有可能引起硬盘操作的进程均暂停运行，故除系统向硬盘写入快照之外，硬盘数据不会再被第三方修改，从而达到了保证快照和硬盘数据一致性。

根据上述对休眠模式的分析，系统进入和退出休眠模式需要完成的工作流程如图 3-10 所示。

3.5.2 休眠模式设计与实现

与其它低功耗模式相比，休眠模式所需硬件支持较少，因此在 Linux 内核中对休眠模式的支持相对较好，但是仍然需要不同平台根据自身特点添加相关代码以实现休眠功能。Linux 内核中进入和退出休眠模式的通用流程分别如图 3-11、图 3-12 所示。

从图 3-11 中可以看出，Linux 内核在进入休眠模式的主要步骤为：冷冻进程（freeze tasks，即挂起可能引起外设读写的进程，包括所有用户进程和部分内核线程如 pdflush 线程等），挂起设备和系统设备（suspend devices 和 suspend system devices，即依照设备的拓扑顺序依次调用各设备驱动的 suspend 函数，保存设备上下文），调用 swsusp_arch_suspend 函数（完成系统快照的生成工作），和写入快照（write snapshot，即将系统快照写入硬盘）。

swsusp_arch_suspend 函数是进入休眠模式时需要实现的最重要函数之一，其主要功能包括：1) 保存定点、浮点处理器上下文；2) 调用 swsusp_save 函数，将内存有效内容保存至空闲内存之中，从而在系统空闲内存之中生成系统快照。swsusp_arch_suspend 函数的主要流程如图 3-13 所示。

为了保存定点和浮点处理器的上下文，定义结构体类型 swsusp_arch regs：

```
struct swsusp_arch_regs {
    struct cpu_context_save cpu;
    struct fpu_context_save fpu;
}
```

其中，cpu_context_save 和 fpu_context_save 分别定义为：

```
struct cpu_context_save {
    unsigned int save[29]; //保存 r4~r25, fp, sp, pc
}
struct fpu_context_save{
    unsigned int save[33];
}
```

swsusp_save 函数首先计算系统中保存有有效数据的物理内存的页框数，然后将保存有有效数据的物理内存页内容依次拷贝至空闲物理内存页中。其中，保存有效内容的物理内存页由内存位图 orig_bm 标记，而空闲物理内存页则由内存位图 copy_bm 标记。当 swsusp_save 函数执行成功后，系统空闲内存中则生成了一个系统的运行快照。

当系统快照生成之后，内核会调用 swsusp_write 函数将快照写入硬盘。快照的保存方式利用了 Linux 虚拟存储系统中的 swap 机制，快照被保存至位于硬盘之中的 swap 交换空间。swsup_write 函数首先将保存在系统空闲物理内存页框（由 copy_bm 位图标记）中的系统快照写入交换空间的数据区（data pages），然后将系统快照中各物理内存页的原始位置（由 orig_bm 位图标记）写入交换空间的元数据区（meta pages），最后向交换空间头部写入相应的控制信息。

在系统快照被写入硬盘之后，内核将调用 power_down 函数关闭系统电源。power_down 函数最终调用 gpio_set_value 函数关闭系统电源（在 PKUnity-3 系统芯片平台之上，通过 GPIO 管脚输出控制系统电源）。

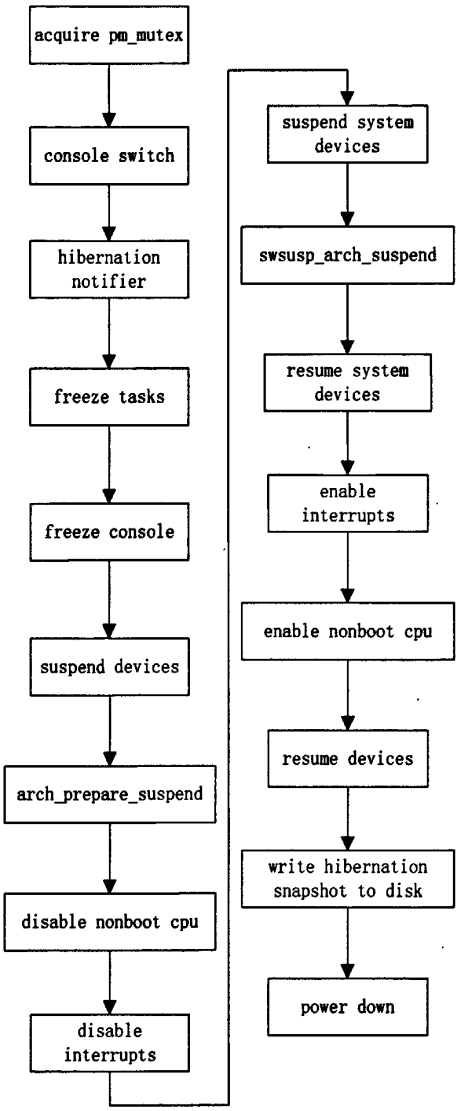

图 3-11 Linux 中进入休眠模式框架

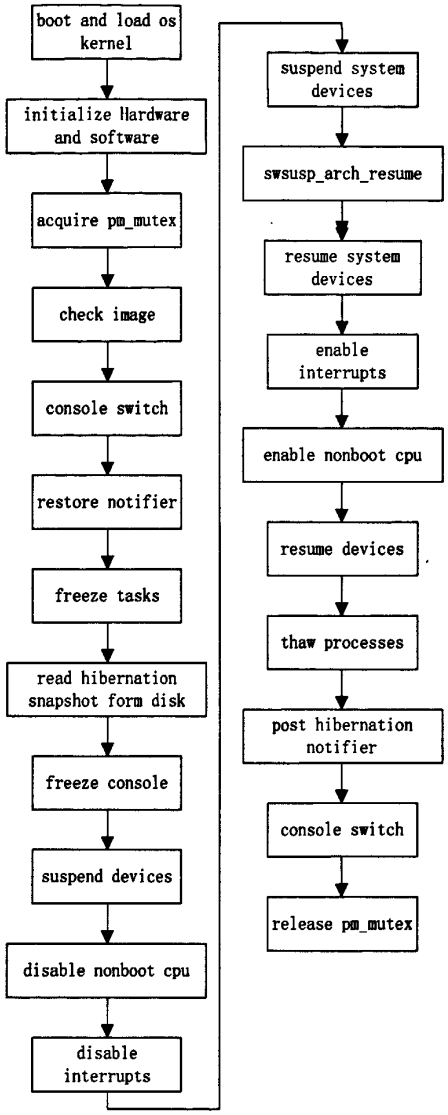

图 3-12 Linux 中退出休眠模式框架

从图 3-12 中可以看出，退出休眠模式的主要步骤包括：检查镜像（check image，即检查硬盘中是否存在有效的系统快照），挂起进程，读入快照（read snapshot），挂起设备和系统设备，调用 swsup_arch_resume 函数，唤醒设备和系统设备（resume system devices 和 resume deveces），解冻进程（thaw_processes）。

swsusp_arch_resume 函数是退出休眠模式过程中需要实现的最重要函数之一，其主要功能是根据系统快照恢复内存、定点、浮点处理器上下文。

swsusp_arch_resume 函数的主要流程如图 3-14 所示。

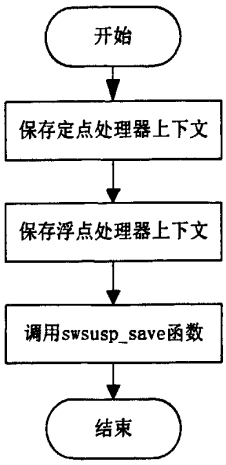

图 3-13 swsup_arch_suspend 函数流程

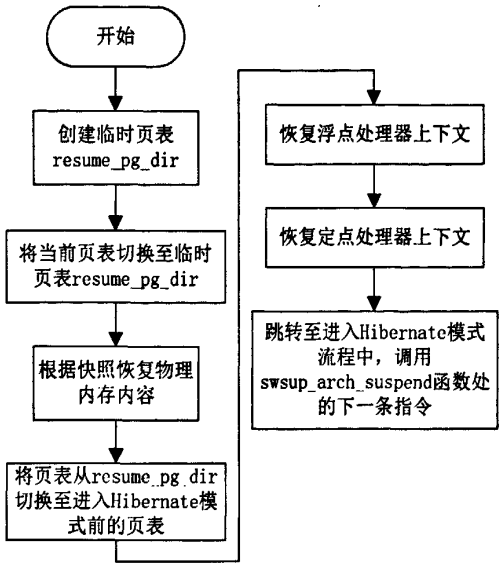

图 3-14 swsup_arch_resume 函数流程

3.5.3 休眠模式设计实现难点

从图 3-10 中可以看出，休眠模式与睡眠模式一样，也具有极其复杂的处理流程，也涉及到系统中几乎所有的硬件设备和软件上下文环境。除具有睡眠模式所具有的处理流程长、涉及设备多、调试困难等特点外，休眠模式的主要难点还体现在：

1.需要在 Linux-uc 之中提供 Swap 机制支持。休眠模式保存、恢复快照是基于 Swap 机制而实现的，但是在本文工作以前，PKUnity-3 系统芯片平台并不支持 Swap 机制，因此本文在 Linux-uc 中增加了面向 PKUnity-3 系统芯片平台的 Swap 机制支持。

2.进入休眠模式前的旧系统和退出休眠模式后的新系统使用物理内存冲突的问题。在休眠模式实际应用中，进入休眠模式前的旧系统和退出休眠模式后的新系统不可避免的会分配使用相同的物理内存，从而引起物理内存冲突。如果不考虑物理内存冲突，直接用保存在快照中的系统上下文覆盖新系统所使用的物理内存，则不可避免会导致系统崩溃。

为了解决这一问题，在休眠模式的设计与实现中，将系统运行过程中使用的物理内存分为两类：1）操作系统可执行代码静态使用的物理内存数据；2）其它有效数据使用的物理内存。

对于前者，由于休眠前的旧系统和休眠后的新系统均采用相同的操作系统内

核，故其在休眠前、后无需保存和恢复。

而对于后者，则将其进一步划分为两类：1）冲突的物理内存和 2）不冲突的物理内存。对于旧系统快照中不冲突的物理内存，在休眠后恢复系统时使其直接覆盖相应的物理内存；而冲突的物理内存，则将其中数据暂时保存在空闲的不冲突的物理内存之中，然后在恢复系统快照的最后一个阶段进行恢复。最后阶段的冲突物理内存恢复过程如下：1）在空闲且不冲突的物理内存中创建临时页表，并让 MMU 使用临时页表进行虚、实地址转换；2）将暂时保存在空闲物理内存中的、冲突的物理内存数据恢复至其应存储的本来位置；3）将页表切换回旧系统的页表，让 MMU 按旧系统页表进行虚、实地址转换。

3.6 本章小结

本章首先通过对比、借鉴主流工业标准，并结合硬件平台自身特点，提出了 PKUnity-3 系统芯片平台支持的系统运行模式（工作、空闲、睡眠、休眠和关机五种模式），然后根据实际需要分别讨论了各种系统运行模式的设计与实现。此外，本章还对睡眠和休眠模式设计与实现中需要解决的难点问题进行了讨论。本章内容完成了系统级动态功耗管理中的机制部分内容，为第五章的系统级动态功耗管理方案的提出提供了支持和保障。

第4章时钟管理机制设计与实现

在现代计算机系统中，计算机中各组成部件在时钟驱动下正常运转并协调工作，时钟频率的设置和通断对整个计算机系统的性能、功耗有着至关重要的影响。对系统中时钟树的管理和维护是操作系统功耗管理机制的重要组成部分，亦是本文工作的主要工作之一。本章将讨论面向PKUnity-3系统芯片平台的操作系统时钟管理机制的设计与实现。

4.1 PKUnity-3系统芯片中的时钟树结构

在数字集成电路中，时钟树是保证系统正常工作的心脏与脉络，负责将时钟源（时钟树的源头）产生的时钟送达时钟域内所有寄存器[9]。时钟源通常由锁相回路（PLL）产生，其频率由PLL决定。

在PKUnity-3(SK)系统芯片中，共有4棵时钟树，分别对应4个时钟域（主时钟域、DDR2 SDRAM时钟域、显示部件时钟域和SATA时钟域），每棵时钟树均以相应的PLL为时钟源。其中，DDR2 SDRAM时钟域、显示部件时钟域和SATA时钟域的时钟树结构相对较为简单，而主时钟域较为复杂，其结构如图4-1所示。

由图4-1可以看出，主时钟域对应的时钟树采用了多级分频的架构。PLL产生的时钟经过一级分频后产生CPU时钟mclk，经门控单元后输出至CPU；第二级，mclk时钟再次分频得到bclk64（64位总线时钟）和bclk32（32位总线时钟）；最后一级，bclk64分频得到2D图像加速模块时钟ge_clk；bclk32分频得到视频解码模块MME时钟ve_clk，H264编解码模块时钟he_clk、hd_clk，以及Nand Flash控制器时钟nand_clk。

在PKUnity-3(65)系统芯片中同样有4棵时钟树，分别对应于时钟域SYS、DDR、显示部件和PHY。其中后三者结构较为简单，SYS时钟域对应的时钟树结构如图4-2所示。

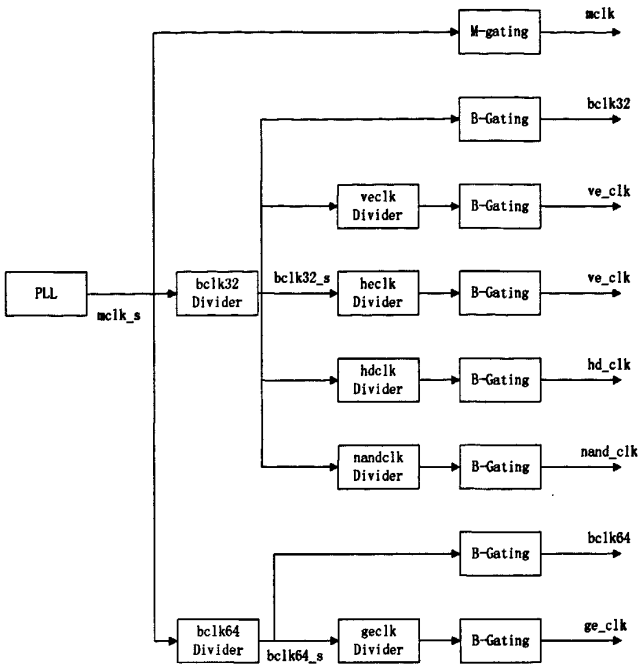

图 4-1 PKUnity-3(SK)主时钟树结构

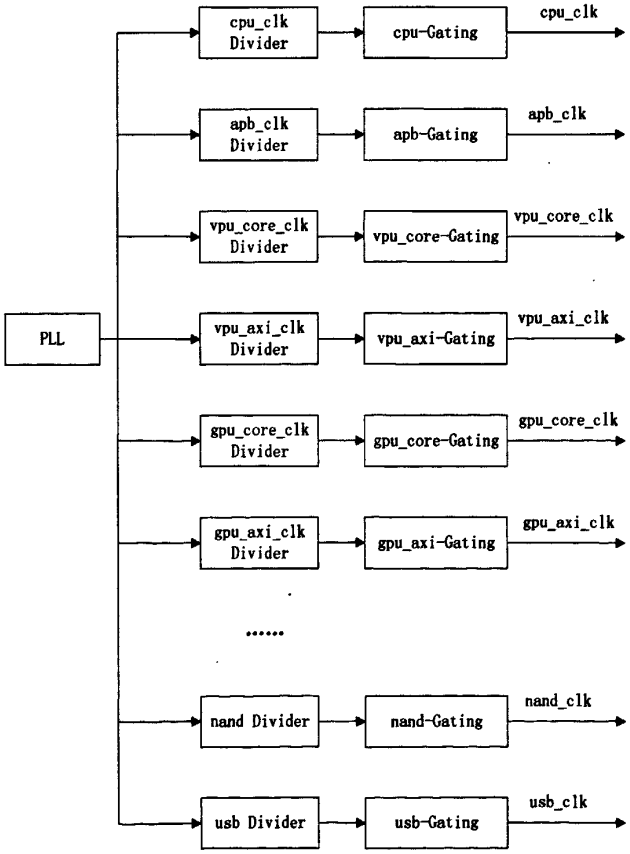

图 4-2 PKUnity-3(65)主时钟树结构

从图 4-2 中可以看出，PKUnity-3(65)的 SYS 时钟树采用了单级分频的并行结构。其时钟树上的所有时钟均由 PLL 单级分频得到。同 PKUnity-3(SK)的时钟树结构相比， PKUnity-3(65)时钟树结构缩短了时钟路径，降低了路径延迟，有利于时钟树的平衡和抑制发生在始终能够网络上的 OCV 效应。

4.2 操作系统内核时钟管理

操作系统内核时钟管理主要包括：时钟初始化，时钟频率设置，时钟门控和内核接口四部分内容。 本节将首先明确内核时钟管理的主要任务需求，然后将从系统时钟的表示、注册和初始化、时钟的主要操作、内核接口几方面详细讨论内核时钟管理的设计与实现。

4.2.1 主要任务需求

面向 PKUnity-3 系统芯片平台的操作系统时钟管理主要需求如下：

1. 为各硬件模块的初始化程序（主要指该模块的驱动程序中的初始化函数）提供支持。在硬件模块初始化过程中，各模块初始化程序需要获取该模块自身的输入时钟频率，以便根据时钟频率设置该模块的其他硬件参数。

2. 为各硬件模块驱动程序提供修改、设置其硬件模块自身输入时钟频率的功能。1）通常，硬件模块可以在多个时钟频率下正常运行，但为了获取更佳的性能（或别的指标参数），驱动程序常常希望将该模块的时钟频率设置为某一目标值；2）一些设备控制器（如 MMC 控制器和 Nand 控制器）常常需要根据其实际连接的设备动态设置时钟参数，因此存在动态设置该模块时钟输入频率的现实需求。

3. 为用户或应用程序提供查询、设置、门控系统中各硬件模块输入时钟频率的功能。在系统运行过程中，用户或应用程序常常希望查询获知系统中各模块的运行频率，而在系统的研发阶段，开发人员需要在不同时钟频率下对系统进行调试和评测。为了满足上述应用需求，故需要操作系统内核向用户或应用程序提供接口，实现对系统中各时钟的查询、设置和门控功能。

4.2.2时钟的表示

尽管在系统芯片中的各个时钟呈现树状拓扑结构，但是根据操作系统时钟管理的需求，操作系统并不需要显式的维护时钟的这种树状组织结构，因此我们采用了最简单的链表结构来表示系统中的各个时钟。

为了表示系统中的每一个时钟，定义结构体变量（该变量的类型为 struct clk）进行表示。结构体 clk 定义如下：

```
struct clk{
    struct list_head node; //维护链表所需的相关数据结构
    unsigned long rate; //该时钟的频率
    const char *name; //该时钟的名字
}
```

如 PKUnity-3(65)中 SYS 时钟域的 PLL 可表示如下：

```
struct clk clk_sys_clk = {
    .name = "SYS_CLK"
}
```

对于系统中的所有时钟，定义链表型变量 clocks 进行组织，定义如下：

```
LIST_HEAD(clocks);
```

由于链表 clocks 表示了系统中所有的时钟，为了避免系统中并发访问、修改该链表，特定义互斥锁 clocks_mutex。该锁定义如下：

```
DEFINE_MUTEX(clocks_mutex);
```

注意，LIST_HEAD 和 DEFINE_MUTEX 均是 Linux 内核中的宏，其中前者用于初始化一个链表，后者用于初始化一个互斥锁。

4.2.3时钟的注册和初始化

系统中时钟的初始化工作由操作系统在其启动过程中的 arch_init 阶段完成，具体负责执行系统时钟初始化的函数是 clk_init 函数。clk_init 函数的主要流程分为两步：1）按时钟树的层级顺序，依次初始化系统中表示各时钟的数据结构；2）将系统中的各个时钟注册到链表 clocks 之中。

对各个时钟的初始化分别由相应的函数完成。以 PKUnity-3(65)中的 SYS 时钟域 PLL 和 CPU 时钟初始化为例，说明如下。

SYS 时钟域 PLL 和 CPU 时钟分别由变量 clk_sys_clk 和 clk_cpu_clk 表示(其类型均为 struct clk)，相应初始化函数为 clk_sys_init 和 clk_cpu_init。clk_sys_init 和 clk_cpu_init 函数流程图如图 4-3、图 4-4 所示。

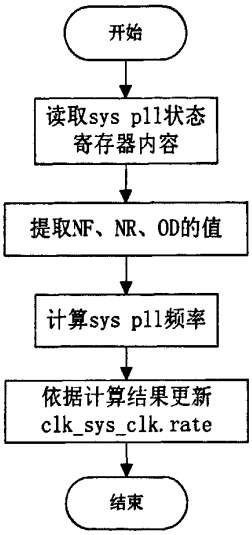

图 4-3 PKUnity-3(65) clk_sys_init 函数流程

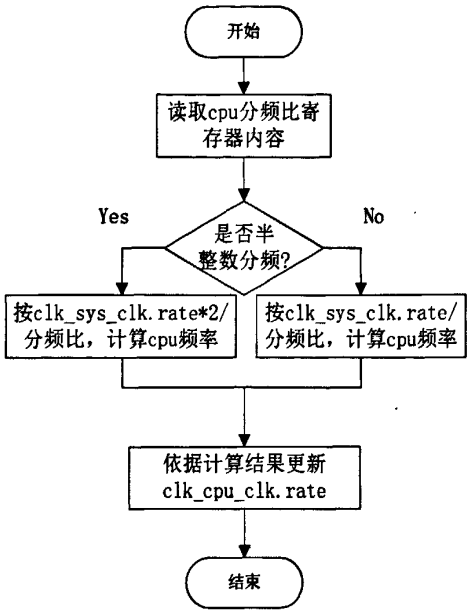

图 4-4 PKUnity-3(65) clk_cpu_init 函数流程

注意图 4-3 中的 NF、NR、OD 分别表示 Reference divider、Feedback divider 和 Output divider。

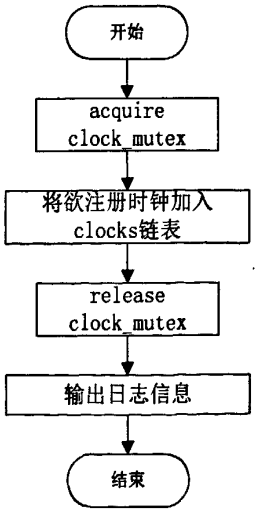

图 4-5 clk_register 函数流程

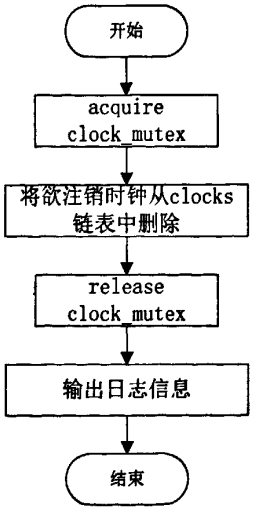

图 4-6 clk_unregister 函数流程

时钟注册由 clk_register 函数完成，相应的时钟注销函数为 clk_unregister。clk_register 函数和 clk_unregister 函数的流程如图 4-5、图 4-6 所示。

4.2.4面向内核的时钟操作接口

面向内核的时钟操作主要有注册、查询、门控和频率设置。注册操作已经在 4.2.3 小节中进行介绍，本节主要介绍剩余几类操作的设计与实现。

时钟的查询操作函数主要有：clk_get 和 clk_get_rate 函数。其中 clk_get 负责根据指定的时钟名（字符串）查询 clocks 链表，并返回查找到的表示该时钟的 clk 结构体类型变量的指针。clk_get_rate 函数以表示时钟的 clk 类型变量为参数，并以表示该时钟的 clk 类型变量的 rate 域作为函数返回值。

时钟门控的相关函数为：clk_enable 和 clk_disable 函数，分别负责使能和禁止该时钟输入。对时钟进行门控需修改功耗管理模块硬件的时钟门控寄存器。如使能和禁止 nand 模块的输入时钟，分别通过在相应函数中执行语句实现：

```
PM_PCGR &= ~PM_PCGR_NAND;
PM_PCGR |= PM_PCGR_NAND;
```

时钟频率设置由 clk_set_rate 函数实现。由于不同时钟的频率设置方法不同，因此在 clk_set_rate 函数中并不直接进行时钟的频率设置操作，而是调用具体时钟自身的频率设置函数。

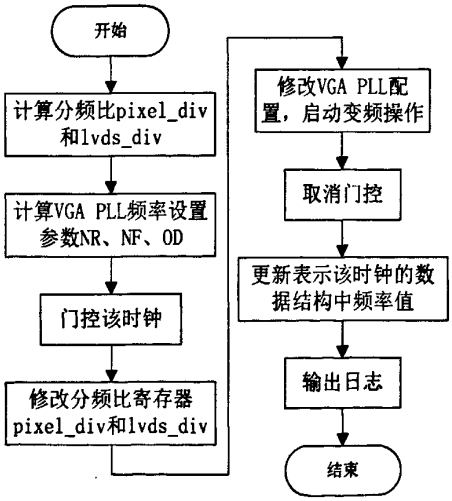

图 4-7 clk_set_lvds 函数流程

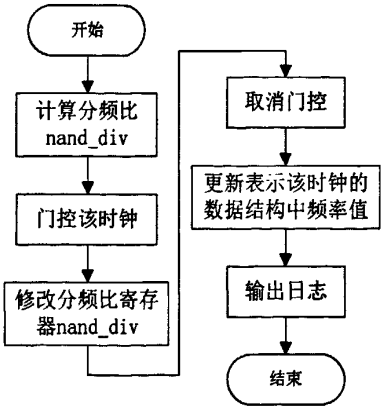

图 4-8 clk_set_nand_rate 函数流程

如在 PKUnity-3(65)中，lvds 的频率设置由 clk_set_lvds_rate 函数完成，而 nand

的频率设置由 clk_set_nand_rate 函数完成。其它时钟的频率设置与 lvds 和 nand 频率设置流程类似。

clk_set_lvds_rate 函数和 clk_set_nand_rate 函数的主要流程如图 4-7、图 4-8 所示。

4.2.5面向用户空间的内核接口

面向用户空间的内核接口是 PKUnity-3 系统芯片操作系统时钟管理机制向用户和应用程序提供的查询、设置、门控系统运行频率的统一接口，由通过在 procfs 中添加相应文件实现。

procfs[41]是 Linux 向用户或应用程序提供的一个特殊文件系统，通过将系统运行过程中的内核信息抽象为一个个文件，用户或应用程序可以像操作文件一样读取、修改操作系统内核的相关数据。通常，procfs 挂载在系统根目录的 proc 目录之上（即/proc 目录），包含有进程运行信息、内核中其他系统运行信息、内核自身信息等内容。

面向用户空间的内核接口需要完成两项功能：1）时钟频率查询功能，即能直接向用户或应用程序提供某一模块的运行频率；2）时钟配置、门控功能，即能向用户或应用程序提供功耗管理模块相关寄存器内容的读写配置功能。为了完成这两项功能，本文在 procfs 的根目录下创建了包含 clk_value 和 clk_setting 子目录的 clk_rate 目录。面向用户空间的内核接口的目录组织如图 4-9 所示。

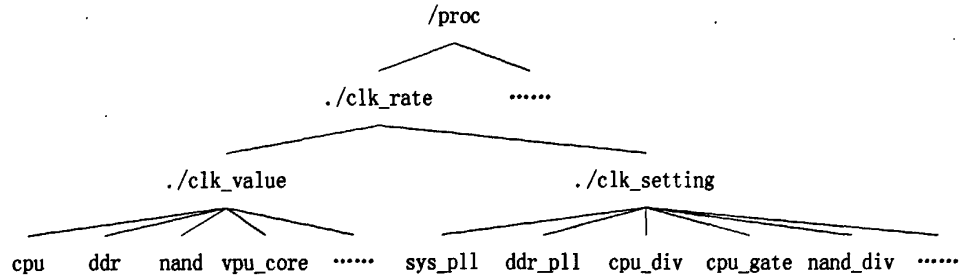

图 4-9 内核接口在/proc 目录下的组织结构

其中 clk_value 目录下包含名称为 cpu、ddr、nand 等的特殊文件，用户和应用程序通过读取这些文件的内容即可获知相应模块的运行频率，clk_value 目录下的所有文件的访问权限均为只读；clk_setting 子目录下包含 sys_pll、ddr_pll、

vga_pll、cpu_div、cpu_gate 等特殊文件，用户或应用程序通过这些即可直接读写功耗管理模块硬件相应寄存器的内容，进而实现系统时钟的管理和配置，此目录下的所有文件权限为系统用户可读写，其余用户只读。

为了实现时钟管理的内核接口，除了需要按照图 4-9 中的组织结构，调用 procfs 提供的相关接口函数创建相应的文件之外，还需要为所有文件创建相应的读写函数。对于 clk_value 子目录下的文件，由于其只具有只读权限，因此本文仅统一实现了该子目录下所有文件的读函数 proc_read_clk_value；而对于 clk_setting 目录下的所有文件，则统一实现了该子目录下所有文件的读写函数 proc_read_reg 和 proc_write_reg。

4.3 本章小结

本章完成了操作系统功耗管理机制中的重要组成部分——时钟管理机制的设计与实现。本章首先介绍了 PKUnity-3 系统芯片中的时钟树结构，明确了操作系统时钟管理的主要任务需求，然后结合实际，设计并实现了表示时钟的数据结构、相关操作函数和内核接口。本章工作对 PKUnity-3 系统芯片操作系统的调试、评测和正常运行具有重要意义。

第5章 功耗管理实施方案

本文在第三章和第四章中设计并实现了包含系统级动态功耗管理机制和时钟管理机制的操作系统功耗管理机制。本章将首先以已实现的操作系统功耗管理机制为基础，设计一种面向 PKUnity-3 系统芯片平台的功耗管理实施方案；然后以系统级动态功耗管理为例，利用第三章实现的系统级动态功耗管理机制，并结合本章给出的实施方案，实现面向 PKUnity-3 系统芯片平台的系统级动态功耗管理并给出相关的功耗评测数据。

5.1 面向 PKUnity-3 系统芯片平台的功耗管理实施方案

根据不同代码在功耗管理中扮演的角色不同，功耗管理可以被划分为两大部分内容：功耗管理机制和功耗管理策略。其中，前者负责解决“需要提供什么样的功耗管理功能”，如在本文中，提供了包含五种运行模式的系统级动态功耗管理功能、时钟门控和时钟频率动态设置功能；而后者负责“如何使用这些提供的功耗管理功能”，以便使功耗管理机制提供的各项功能得到有效的利用。

结合硬件平台的特点和实际需要，一种面向 PKUnity-3 系统芯片平台的功耗管理实施方案如图 5-1 所示。

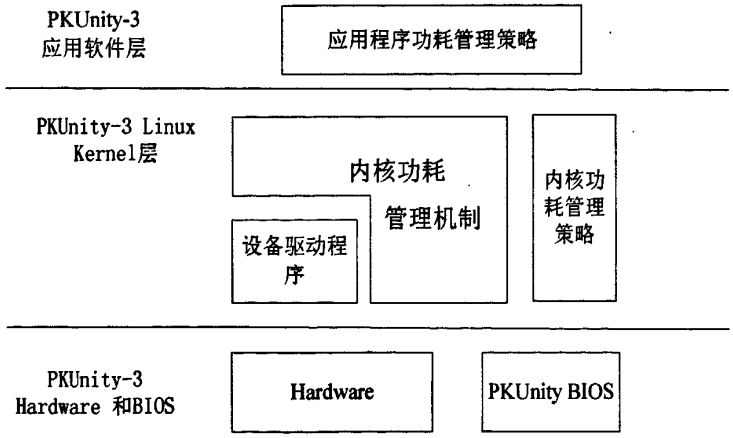

图 5-1 一种面向 PKUnity-3 系统芯片平台的功耗管理实施方案

从图5-1中可以看出，该实施方案分为：PKUnity-3 硬件和 BIOS 层、PKUnity-3 操作系统层和 PKUnity-3 应用软件层三个层次。其中，功耗管理的机制部分全部

位于操作系统内核之中，而管理策略则根据实际需要放置于应用软件层或操作系统层内。硬件和 BIOS 为操作系统功耗管理机制的实现提供基本的硬件支持。

在该实施方案的实际应用中，内核或应用软件层中负责功耗管理策略的软件监听系统的运行负载情况，根据自身实现预定管理策略来决定何时使用功耗管理机制提供的何种功能。比如，可在操作系统内核中创建一些线程，用于监视系统中某些设备的活动情况，当这些线程发现其所监视的设备处于空闲状态（或利用率不高时）时，这些线程可利用本文第四章时钟管理机制中的时钟门控机制（或时钟频率动态设置机制），门控其所监视设备的时钟输入（或降低该设备的输入时钟频率），进而达到节省该设备所消耗功耗的目的；又如，可在应用软件层中创建监视系统负载的后台进程，当该进程发现系统长时间处于空闲状态时，可以利用本文第三章提供的系统级动态功耗管理机制，使系统转入具有较低功耗的运行模式，从而达到节省系统总体功耗的目的。

该实施方案的设计体现了机制和策略相分离的原则。即将功耗管理中最核心的功耗管理机制放置在操作系统之中，作为操作系统的一部分加以实现，功耗管理机制的设计与实现无需考虑功耗管理的策略。而功耗管理策略则根据实际需要可以在应用软件和操作系统内部实现，但不必考虑功耗管理机制的实现细节，体现了一定的灵活性。此外，将包含系统级动态功耗管理机制在内的功耗管理机制放置在操作系统中实现，还避免了对 BIOS 软件的不必要依赖，避免了因 BIOS 和操作系统内核版本问题而影响系统中功耗管理系统不能正常工作情况的发生。

5.2 系统级动态功耗管理

APM 和 ACPI 是当前工业界主流的功耗管理标准。由于系统级动态功耗管理是 APM 和 ACPI 的主要内容，因此系统级动态功耗管理在大量计算机中得到了广泛应用。

本节将以系统级动态功耗管理作为上节提出的功耗管理实施方案的具体应用实例，进行阐述。作为功耗管理重要组成部分，系统级动态功耗管理也可以分为系统级动态功耗管理机制和和系统级动态功耗管理策略两部分内容。其中，由于系统级动态功耗管理机制部分已经在本文第三章进行了论述，因此本节将主要讨论系统级动态功耗管理的策略部分，并给出系统级动态功耗管理在实际计算机

中的应用效果。

5.2.1系统级动态功耗管理的工作原理

在系统级动态功耗管理的实际应用中，操作系统内核或应用软件层中负责功耗管理策略的软件运行于系统的工作模式之下。它们监听系统运行负载情况，根据自身实现的预定管理策略来决定何时将系统由工作模式转入何种低功耗模式之中。当功耗管理策略软件决定了系统需要转入某个低功耗模式时，它们会触发操作系统内核中的系统级动态功耗管理机制完成相应的操作，使系统转入该低功耗模式。操作系统中的系统级动态功耗管理机制接收到功耗管理策略软件的相关指示之后，将会首先完成进入该模式前的准备工作，然后配置硬件，使硬件电路转入相应模式下的低功耗状态。当系统硬件转入低功耗状态之后，功耗管理策略软件将停止运行，而系统硬件则会等待唤醒事件（Wakeup Event，即使系统退出低功耗模式的事件）的发生。若系统硬件监测到唤醒事件已经发生，硬件逻辑自身将会使硬件电路恢复到运行状态。此时，操作系统中的系统级动态功耗管理机制将会完成退出该低功耗模式的恢复工作。当恢复工作完成之后，系统即恢复到之前的工作状态，此时功耗管理策略软件将再次工作，等待系统下次模式切换的时机。

5.2.2系统级动态功耗管理策略

所谓系统级动态功耗管理策略是指对系统级动态功耗管理机制的使用方法，即决定系统何时转入何种低功耗模式的方式方法。如第三章所述，在PKUnity-3系统芯片平台之上支持空闲，睡眠，休眠三种低功耗模式，不同的低功耗模式具有不同的特点，适合不同的使用场合，因此系统级动态功耗策略将紧密结合各模式的特点加以制定。

●空闲模式

空闲模式是一种节省功耗较少、但模式切换时间较快的低功耗模式。由于空闲模式可由系统中任意硬件中断唤醒，因此系统每一次进出空闲模式的过程中，系统真正处于空闲模式的时间较短（不超过系统中相邻两次硬件中断的时间间隔）。空闲模式的这一特点决定了空闲模式适合于系统空闲持续时间较短，且系

统空闲、繁忙交替频繁的情况使用。

结合空闲模式的这一特点，本文在位于操作系统内核层中的空闲进程（Idle process）中使用了系统的空闲模式，即每当操作系统因空闲而调用空闲进程时，空闲进程将调用 puv3_cpu_do_idle 函数，使整个系统进入空闲低功耗模式。

●睡眠模式

睡眠模式节省功耗较多，但模式切换时间相对较长的低功耗模式。考虑到睡眠模式的这一特点，睡眠模式通常适合于系统中空闲时间较长的情况下使用。为了较好的使用睡眠模式，本文采用了如下的使用策略：

1. 在应用软件层中实现管理策略。在应用软件层中实现有利于用户根据实际情况随时修改功耗管理策略，使系统的功耗管理具有较大的灵活性；

2. 采用了当前普遍使用的超时（Time-out）算法，即若当前系统处于空闲状态的时间超过实现预定的阈值（如默认为 3 分钟），使系统进入睡眠模式；

3. 通过检测系统所有输入设备（/dev/inpus 目录下的所有设备）是否有新的输入事件（输入事件将会导致相应的设备产生新的中断）产生来决定系统是否空闲。

● 休眠模式

休眠模式节省功耗最多，但模式切换时间最长的低功耗模式，适合系统空闲时间很长的情况下使用。考虑到休眠模式的特点，本文将进入休眠模式的时机留给计算机系统的使用者决定，即当用户将长时间不使用计算机，但又愿意丢失当前系统运行状态的情况下，使系统进入休眠模式。

综上，面向 PKUnity-3 系统芯片平台的系统级动态功耗管理方案如表 5-1 所示。

表 5-1 面向 PKUnity-3 系统芯片的系统级动态功耗管理策略

模式名称	进入模式事件（Suspend Event）	退出模式事件（Resume Event）
空闲（Idle）	系统调用空闲进程	任意硬件中断
睡眠（Sleep）	系统无任何输入的持续时间超过指定阈值	用户按下计算机电源键
休眠（Hibernation）	用户主动发起	用户重新开启计算机

5.2.3功耗评测

为了验证系统级动态功耗管理的有效性，本文在北大众志安全微工作站

（A25-18110 计算机）上实现了系统级的动态功耗管理，并进行了测试。

北大众志安全微工作站（A25-18110 计算机）采用了 PKUnity-3(SK)系统芯片，运行频率为 500MHz，运行于该计算机上的操作系统内核采用 Linux-uc 2.6.32，内核配置为该型计算机的默认配置。

采用五次测量取平均值的方法，该型款计算机的系统级动态功耗管理测试结果如表 5-2 所示，功耗单位为瓦特（W）。注意，表中安全微工作站的功耗是包含显示屏和电源适配器在内的整机功耗。

表 5-2 系统级动态功耗管理测试结果

系统模式	安全微工作站
工作（Working）	17.0W
空闲（Idle）	16.4W
睡眠（Sleep）	3.7W
休眠（Hibernation）	约为 0
关机（Off）	0

从表 5-2 中可以看出，与工作模式相比，空闲模式节省功耗较少，这是因为：1）空闲模式本身特点。空闲模式仅门控了 CPU 的输入时钟，故理论上，该模式所能节省的功耗就少；2）空闲模式的使用特点。由于空闲模式是在多任务环境中的空闲进程中加以使用的，而在多任务环境下，系统中将存在较多的硬件中断，因此每次系统真正处于空闲模式的持续时间较短。在这种情况下，测量的空闲模式下的功耗，从严格意义上讲，并不是真正的空闲模式下的功耗，而是空闲模式与工作模式频繁交替而产生的一个合成功耗，故这种情况下，空闲模式下的功耗测量值较其真实值高，因此测试结果反映出的空闲模式下节省的功耗较其实际节省的功耗偏少。

睡眠模式节省功耗较多。但是，由于测试结果中包含电源适配器功耗，故睡眠模式下计算机的真实功耗应比表中结果更低。

休眠模式下，由于通过 GPIO 切断了系统电源，但电源适配器（或台式机开关电源）仍然接通电源，因此该模式下功耗为约为 0。而关机模式则由于完全切断电源，功耗为 0。

从测试结果来看，北大众志安全微工作站上的系统级动态功耗管理十分有效，符合国家 2009 年发布的计算机节能产品认证标准。

5.3 本章小结

本章首先基于第三章、第四章的系统级动态功耗管理机制和时钟管理机制，设计了一种面向 PKUnity-3 系统芯片平台的功耗管理实施方案；然后以系统级动态功耗管理作为该实施方案的应用实例，实现了面向 PKUnity-3 系统芯片平台的系统级动态功耗管理。实验结果表明，系统级动态功耗管理可以正确、有效地运用于基于北大众志 PKUnity-3 系统芯片的计算机之中。

第6章 总结与展望

随着计算机芯片集成度和计算能力的迅速提升，功耗成为制约计算机技术进一步发展、应用的瓶颈因素。为了有效降低系统功耗，功耗管理技术成为学术界和工业界共同的研究热点。

本文以面向 PKUnity-3 系统芯片平台的功耗管理为主题，结合 PKUnity-3 系统芯片平台特点，并借鉴其他平台的相关实现，在 Linux-uc 操作系统内核中实现了较为完整的、包含系统级动态功耗管理机制和时钟管理机制两部分内容的操作系统功耗管理机制，为系统级动态功耗管理和系统的调试、运行提供了有力支持。在设计、实现操作系统功耗管理机制的基础之上，本文给出了一种面向 PKUnity-3 系统芯片平台的功耗管理实施方案，并以系统级动态功耗管理作为该实施方案的应用实例，实现了面向 PKUnity-3 系统芯片平台的系统级动态功耗管理。目前，该系统级动态功耗管理已成功应用于北大众志 A25 计算机之中。

为了完成本文的研究任务，本文首先介绍了功耗的组成和已有的主要功耗管理技术，然后分析了工业界主流的功耗管理标准（APM 和 ACPI）和 Linux 内核中的现有的功耗管理技术，明确了它们的软、硬件依赖和不足之处，为实现面向 PKUnity-3 系统芯片的功耗管理提供了借鉴和参考。在对相关工作有了较为全面的了解之后，本文依据 PKUnity-3 系统芯片平台的硬件特性，分别设计并实现了系统级动态功耗管理机制（支持工作、空闲、睡眠、休眠和关机五种运行模式）和时钟管理机制（包含操作系统时钟初始化、动态时钟频率设置、时钟门控等内容）。最后，根据机制与策略相分离的软件设计原则，本文设计了一种面向 PKUNity-3 系统芯片平台的功耗管理实施方案。此外，为了验证该实施方案的可行性，本文以系统级动态功耗管理为例在基于北大众志 PKUnity-3 系统芯片的计算机中实现了系统级动态功耗管理。

本文工作表明，PKUnity-3 系统芯片硬件（主要指其中的功耗管理模块）为软件功耗管理提供了较好的支持。但是，在 PKUnity-3 系统芯片的后续研发中，如果能够在以下几个方面有所改进，将会使系统功耗管理更为有效：

1. 系统睡眠模式下，各硬件模块的电源供给问题。在 PKUnity-3(SK)和

PKUnity-3(65)芯片中，功耗管理模块提供的睡眠模式是通过门控各硬件模块输入时钟实现的。但是在实际使用此模式的过程中，由于软件会在进、出此模式时保存、恢复设备的上下文，因此，即使硬件在睡眠模式下关闭各硬件模块的电源输入，软件依然能够保证系统的正确运行。故如果能够采用关闭模块电源的方式实现睡眠模式，将使系统在睡眠模式下具有更小的功耗。

2. CPU 动态变频变压技术的支持。目前，CPU 动态变频变压技术已经成为一种有效的功耗管理技术并得到实际应用。理论、实践均表明动态变频技术只有和动态变压技术相结合，才能有效的降低系统功耗。当前，PKUnity-3(SK)和 PKUnity-3(65)芯片仅对动态变频具有基本支持，如果在后续芯片研发中添加动态变压支持，并结合软件的功耗管理，将使系统功耗管理变得更为有效。

目前，PKUnity-3(SK)芯片应经成功运用于北大众志系列计算机产品之中，而 PKUnity-3(65)芯片的研发也已进入了最后的收尾阶段。相信，随着研发的继续进行，面向 PKUnity-3 系统芯片的软、硬件功耗管理技术将会更加完善和有效。希望本文能够为实验室下一步的工作起到抛砖引玉的效果。

参考文献

[1].J. L. Hennessy, D. A. Patterson. "Computer Architecture: A Quantitative Approach". *Morgan Kaufmann*, 2006, 4th edition, pp.17~19

[2].A. S. Tanenbaum. "Modern Operating Systems". *Pearson/Prentice Hall*, 2001, 3rd edition, pp.363~371

[3].T. Mudge. "Power: A First-Class Architectural Design Constraint". *IEEE Computer*, 2001, vol.34, no.4, pp.52~58

[4].杨锟. "PKUnity-3(SK)系统芯片功耗管理部件的设计与实现". *北京大学硕士研究生学位论文*, 2009

[5].J.M. Rabaey and M. Pedram, E. Kluwer. "Low-Power Design Methodologies". 1996

[6].北京大学微处理器研究开发中心. "北大众志 CPU 及系统芯片的研究进展与展望". 2009

[7].北京大学微处理器研究开发中心. "PKUnity-3 系统芯片用户手册". 2010

[8].北京大学微处理器研究开发中心. "PKUnity-3 系统芯片 PM 控制器用户手册". 2010.

[9].刘洋. "PKUnity-3 系统芯片功耗管理部件设计与实现". *北京大学硕士研究生学位论文*, 2010

[10]. C. Watts. "Dynamic Management of Energy Consumption in Embedded Systems". *Information Quarterly*, 2003, vol.2, no.3

[11].T. Pering, T. Burd, R. Brodersen. "The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms". *Proceedings of International Symposium on Low Power Electronics and Design*, 1998, pp.76~81

[12].W. Fomaciari, P. Gubian, D. Sciuto. "Power estimation of embedded system: a hardware /software codesign approach". *IEEE transactions on Very Large Scale Integration (VLSI) Systems*, 1998, vol.6, no.1, pp.266~275

[13].B. Dave, G. Lakshminarayana, N. Jha. "COSYN: hardware-software co-synthesis for heterogeneous distributed embedded systems". *IEEE transactions on Very Large Scale Integration(VLSI) Systems*, 1999, vol.7, no.1, pp. 92~104

[14].P. P. Ranjan, N. D. Dutt. "Low-power memory mapping through reducing address bus activity". *IEEE transactions on Very Large Scale Integration(VLSI) Systems*, 1999, vol.7,

no.3, pp.309~320

[15].C. Geboty. "Low energy memoyr and register allocation using network flow". *Proceedings of Design Automation Conference(DAC)*, 1997, pp.435~440

[16]. M. Kamble, K. Ghose. "analytical energy dissipation models for low power caches". *Proceedings of International Symposium on Low Power Electronics and Design*, 1997, pp.143~148

[17]. W. E. Dougherty, D. J. Pursley, D. E. Thomas. "Instruction subsetting: trading power for Programmability". *Proceedings of VLSI'98 System Level Design*, 1998, pp.42~47

[18].D. G. Ruimy. "Micro risc architecture for the wireless market". *IEEE Micro*, 1999, vol.19, pp. 30~37

[19]. L. Benini, D. M. Giovanni, A. Macii. "Reducing power consumption of dedicated processors through instruction set encoding". *Proceedings of the 8th Great Lakes Symposium on VLSI IEEE*. Lafayette, 1998, pp.8~12

[20].K. Sunghwan, K. Jigong. "Opcode encoding for low-power instruction fetch". *Electronics Letters*, 1999, vol.35, pp.1064~1065

[21].L. Benini, D. M. Giovanni, A. Macii, et al. "Selective instruction compression for memory energy reduction in embedded systems". *Proceedings of Low Power Electronics and Design*, 1999, pp.206~211

[22].C. Poellabauer, L. Singleton and K. Schwan. "Feedback-Based Dynamic Frequency Scaling for Memory-Bound Real-Time Applications. *Proceedings of the 11th Real-Time and Embedded Technology and Applications Symposium(RTAS)*, 2005

[23]. W. T. Shiue. "Low-Power Scheduling with Resources Operating at Multiple Voltages". *Analog and Digital Signal Processing*, 2000, vol.47, no.6, pp.536~543

[24].A. Varma, B. Ganesh, M. Sen, et al. "A Control-Theoretic Approach to Dynamic Voltage Scaling". *Proceedings of International Conference on Compiles, Architectures, and Synthesis for Embedded Systems(CASES)*, 2003, pp.255~266

[25]. L. Benini, R. Hodgson, R. Siegel. "System-level power estimation and optimization". *Proceedings of Low power Electronics and Design*, 1998, pp.173~178

[26].Y. H. Lu, T. Simunic, G. D. Micheli. "Sotfware controlled power management". *Proceedings of 7th Hardware/Software Codesign(CODES)*, 1999, pp.157~161

[27]. Q. Qiu, Q. Wu, M. Pedram. "Stochastic modeling of a power managed system: construction and optimization". *Proceedings of Low Power Electronics and Design*, 1999, pp.194~199

[28]. L. Benini, G. D. Micheli, E. Macii, et al. "Sysetm-level power optimization of special purse applications: the beach solution". *Proceedings of Low Power Electronics and Design*, 1997, pp.24~29

[29]. M. Li, X. Wu, R. Yao, et al. "Q-DPm: An Efficient Model-Free Dynamic Power Management Technique". *Proceedings of Design, Automation and Test in Europe(DATE)*, 2005, pp.526~527

[30]. C. Gniady, Y. Hu, Y. H. Lu. "Program Counter Based Techniques for Dynamic Power Management". *Proceedings of International Symposium on High Performance Computer Architecture*, 2004, pp.24~25

[31]. S. Ramprasad, N. R. Shanbhag, I. N. Hajj. "A coding framework for low-power address and data busses". *IEEE transactions on Very Large Scale Integration(VLSI) Systems*, 1999, vol.7, no.2, pp.212~221

[32]. S. Ramprasad, N. R. Shanbhag, I. N. Hajj. "Signal coding for low power: fundamental limits and practical realizations". *IEEE transactions on Very Large Scale Integration(VLSI) Systems*, 1999, vol.46, no.7, pp.923~929

[33]. Q. Wu, M. Pedram, W. Wu. "Clock-gating and its application to low power design of sequential circuits". *Proceedings of Custom Integrated Circuits Conference*, 1997, pp.479~482

[34]. L. Benini, G. D. Micheli, E. Macii, et al. "Symbolic synthesis of clock-gating logic for power optimization of control-oriented synchronous networks". *Proceedings of European Design and Test Conference*, 1997, pp.514~520

[35]. J. Monteiro, S. Devadas, A. Ghosh. "Sequential logic optimization for low power using input-disabling precomputation architectures". *IEEE Transaction on CAD of IC and Systems*, 1998, vol.17, no.3, pp.279~284

[36]. C. Yeh, C. C. Chang, J. S. Wang. "Technology mapping for low power". *Proceedings of Design Automation Conference(DAC)*, 1999, pp.145~148

[37]. J. Kim, J. Yang, S. Y. Hwang. "Path sensitization and gate sizing approach to low power optimization. *IEEE electronic Letter Online*, 1998, vol.34, no.7, pp.619~620

[38].H. R. Lin, T. T. Hwang. "On determining sensitization criterion in an iterative gate sizing process". *IEEE Transaction on CAD of IC and Systems*, 1999, vol.18, no.2, pp.231~238

[39].Intel Corporation, Microsoft Corporation. "Advanced Power Management BIOS Interface Specification". 1996, Revision 1.2

[40].Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., Toshiba Corporation. "Advanced Configuration and Power Interface Specification". 2010, Revision 4.0a

[41].D. Bovet, M. Cesati. "Understanding the Linux Kernel, Second Edition". *O'Reilly & Associates*, Inc, 3rd edition

[42].J. Corbet, A. Rubini, K. H. Greg. "Linux Device Drivers". *O'Reilly Media,Inc*, 3rd Edition

[43]. P. Mochel. "Linux Kernel Power Management". *Proceedings of the Linux Symposium*, 2003

[44].A. L. Brown. "The State of ACPI in the Linux Kernel". *Proceedings of the Linux Symposium*, 2004

[45].L. Brown, A. Keshavamurthy, D. S. H. Li, et, al. "ACPI in Linux, Architecture, Advance, and Challenges". *Proceedings of the Linux Symposium*, 2005

[46].L. Brown. "ACPI in Linux – Myths vs. Reality". *Proceedings of the Linux Symposium*, 2007

[47].A. L. Brown, R. J. Wysocki. "Suspend-to-RAM in Linux". *Proceedings of the Linux Symposium*, 2008

[48]. V. Pallipadi, A. Belay, et, al. "cpuidle – Do nothing, efficiently...". *Proceedings of the Linux Symposium*, 2007

[49].V. Pallipadi, A. Starikovskiy. "The Ondemand Governor, Past, Present, and Future". *Proceedings of the Linux Symposium*, 2006

[50]. IBM, Montavista Software. "Dynamic Power Management for Embedded Systems". 2002, Version1.1

致谢

光阴荏苒，如白驹过隙，宝贵的研究生生活即将结束。值此论文完成之际，谨向给予我指导、关心和帮助的各位老师、同学、朋友、亲人致以最衷心的感谢！

首先我要感谢我的导师王克义教授。王老师学识渊博、治学严谨、为人谦和、乐于助人，无论在学术科研上，还是日常生活中，都给予了我宝贵的指导和热情的帮助，使我受益匪浅。王老师您辛苦了，感谢王老师长期以来的默默付出，您是我人生学习的榜样！

感谢微处理器研发中心主任程旭教授。程老师心系国家，学高身正，为中国芯的设计、研发而忘我工作。能够成为 MPRC 一员，在程老师带领下从事中国芯的研发工作，是我终身的荣幸！师恩如海，程老师对我的关心、帮助、理解和支持我将终身铭记！

感谢我的指导教师管雪涛老师。进入实验室以来，我一直在管老师的直接带领下从事科研工作。管老师对中国芯事业的热爱、对科研工作的一丝不苟深深地感染了我，我将在今后的工作、生活中牢记管老师的教诲，努力工作、踏实做人！

感谢佟冬副教授、刘锋副教授、刘先华老师、陆俊林老师、冯毅老师、杨春老师、易江芳老师、李险峰老师、王箫音老师、郑衍松老师、黄侃老师。感谢您们在我研究生阶段给予的无私指导、帮助和理解！滴水之恩，当涌泉相报，我将永远铭记您们的帮助！

感谢中心行政办公室吴丽娜老师、邱文老师、张天童老师和张芳老师。您们通过辛勤的劳动为我们创造了一个良好的学习、科研环境，谢谢您们！

感谢高海斌师兄、苏永刚师兄、张吉豫师姐、王晶师姐、刘姝师姐、夏虞斌师兄、钮艳师姐、黄涛师兄、李皓师兄、刘军涛师兄、钟祺师兄、郑穗欣师兄、贾宁师兄、刘丹师姐、陆小凤师姐、杨阳师姐，感谢胡栋梁、赵野、刘洋、许经纬、华远志、迮雪峰、徐安华、吴栋霞、储小伟、王新平、李凌达、赵兴鹏、易昕，感谢您们对我的关心、帮助、鼓励和支持！

感谢谭明星、冯迹、党向磊和费洪亮，感谢我的女友和家人！

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明： 所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名：何俊　日期：2011 年 6 月 7 日

学位论文使用授权说明

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

- 按照学校要求提交学位论文的印刷本和电子版本；
- 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；
- 学校可以采用影印、缩印、数字化或其它复制手段保存论文；
- 因某种特殊原因需要延迟发布学位论文电子版，授权学校□一年/□两年/□三年以后，在校园网上全文发布。

（保密论文在解密后遵守此规定）

论文作者签名：何俊　导师签名：

日期：2011 年 6 月 7 日