

申请上海交通大学工程硕士学位论文

## UEFI BIOS 电源管理研究和应用

学校代码： 10248  
作者姓名： 唐笛  
学 号： 1080372035  
第一导师： 胡飞  
第二导师： 蔺杰  
学科专业： 嵌入式系统  
答辩日期： 2011 年 3 月 17 日

上海交通大学软件学院

A Dissertation Submitted to Shanghai Jiao Tong University  
for Master Degree of Engineering

**THE RESEARCH FOR UEFI BIOS POWER  
MANAGEMENT AND APPLICATION**

University Code:	10248
Author:	Di Tang
Student ID:	1080372035
Mentor 1:	Hu Fei
Mentor 2:	Lin Jie
Field:	Software Engineering
Date of Oral Defense:	03/17/2011

School of Software  
Shanghai Jiaotong University  
Dec, 2010

## 上海交通大学

### 学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，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，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。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。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

唐笛

日期：2011 年 3 月 17 日

## 上海交通大学 学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本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和汇编本学位论文。


保密 ☐，在\_\_\_\_年解密后适用本授权书。

本学位论文属于

不保密 ☒。

(请在以上方框内打“√”)

学位论文作者签名: 唐笛

指导教师签名: 

日期: 2011 年 3 月 17 日

日期: 2011 年 3 月 17 日

## UEFI BIOS 电源管理研究和应用

### 摘 要

UEFI (Unified Extensible Firmware Interface)由于支持新技术的应用和使用模式的创新,并且它相对于原有的 BIOS(Basic Input Output System)系统更加开放和标准化,因此得到了英特尔、微软、超微等公司的大力推广。但是,目前 UEFI 的发展还未解决下面两个问题:一方面,由于 UEFI 和传统的 BIOS 不支持其自身环境下的电源管理功能,另一方面 UEFI 环境或是传统 BIOS 环境和操作系统环境之间的切换时间较长。因此它给一些应用的推广带来了技术上的困难。本文结合软硬件两方面技术的特点,为开放的创新应用模式构建技术平台,主要实现两个目标:一是研究如何在 UEFI 环境下实现系统级别的电源管理;二是在系统睡眠唤醒后,不直接进入操作系统,而是通过重新创建并进入 UEFI 环境(S3 UEFI)来支持新的应用模式。

本文的研究使用了英特尔的笔记本开发电路板用作开发验证平台,完成了如下主要工作任务。

1) 实现了系统睡眠唤醒后重新构建 UEFI 环境,进入到 UEFI Shell。而在该环境工作结束后,可以退出 UEFI Shell,并继续原来的操作系统唤醒过程,回到操作系统后各设备均能正常使用。

2) 在该 S3 UEFI 环境下实现了网络协议栈功能,并且可以运行 UEFI Shell 下的应用程序,实现了 FTP 下载数据到磁盘的应用模式。

3) 在该环境下实现了系统电源管理功能,通过利用软件控制处理器、PCI 设备、磁盘设备等,结合网络下载特定的使用模式,优化了系统的电源管理功能。

4) 测试该电路板包括直流电源转换器在内的各主要部件的功耗分布,从而了解系统各部件功耗分布在整体功耗中的比重。

5) 比较了 S3 UEFI 环境和不同操作系统,例如 Win7, WinXP, Linux 下的功耗。在相同的使用模式下,对比整体系统功耗,通过数据表明本研究的系统整体功耗低于操作系统下的功耗。

实验结果表明,本研究的功耗低于其他操作系统至少在 17%以上,并且可以在 UEFI

环境和操作系统环境之间快速切换，从而创造新的使用模式。这是由于在商业模式上，该使用模式主要由电脑品牌商控制，从而不依赖于特定的操作系统。例如：可以在 UEFI 环境下做病毒扫描，网络下载和备份等应用，从而达到优于操作系统下的电源管理功能，使得品牌电脑厂商可以做到产品差异化。

由于 UEFI 是开放的工业标准，从而可以做到不同厂家软件的兼容性，鼓励更多的应用在该环境下运行。

**关键词** UEFI, BIOS, 电源管理, ACPI, Shell

## THE RESEARCH FOR UEFI BIOS POWER MANAGEMENT AND APPLICATION

### ABSTRACT

Since UEFI (Unified Extensible Firmware Interface) supports the application of new technology and promotes new usage model; and it is open and standardized compared with legacy closed BIOS (Basic Input Output System) system, so it gets support and promotion by many famous companies such as Intel, Microsoft, AMD, etc. But it has not solved the following 2 problems. First, UEFI and legacy BIOS do not support power management function under their own environment. Second, the switch time between UEFI/BIOS environment and operating system is too long. So the two problems brings obstacle to promote some new technology. The paper will combine the technology features of both hardware and software to create an innovational technology platform for new usage model. The two purposes of the paper are, 1) to research the methods on how to implementing system level power management feature under UEFI environment; 2) when the system is waken up from sleep state, to implement to go to the new created UEFI environment for support new usage model instead of back to operating system..

The paper uses the laptop customer reference board of Intel as the development and verification platform. The main accomplishes are as follows. 1) It creates new UEFI environment after system is waken up from sleep state and runs into UEFI shell. After the work is done, it can exit the UEFI shell to continue previous boot progress. When system returns back to operating system, all devices work normally. 2) It implements the network stack under S3 UEFI status and can run FTP application to download data into the hard disk. 3) It implements the power management function to control devices by using software, such as processor, PCI devices, hard disk, etc. It optimizes the system power consumption under the specific network download usage model of the UEFI environment. 4) It requires measuring power consumption distribution of various components, including DC-DC converter on the motherboard to understand the device power consumption to system total power consumption ratio of various devices. 5) It compares total system power consumption of the S3 UEFI environment to other operating systems, such as the power consumption

under Win7, WinXP and Linux at the same usage model. It provides data to prove that the total system power consumption of the UEFI environment is the lowest. The experiment result shows that the system power consumption based on the paper is at least 17% lower than other operating systems. And it also can switch software environment between UEFI and operating system in short time. So it can bring a new usage model for user. Because that this usage model can enable the business model that the brand computer companies can create new usage model independent of specific operating system. For example, under UEFI it can scan virus, download data from network, and backup local data to server. So it can get better power result to make the computer company to distinguish itself among other companies in the market. The open industry standard feature of UEFI, could support different companies' software, and is easy to promote more applications under the UEFI environment.

**Keywords:** UEFI, BIOS, Power Management, ACPI, Shell



## 目 录

1 绪论 .....	1
1.1 论文研究的目的和意义 .....	1
1.2 国内外技术发展现状与趋势 .....	2
1.3 研究目标与关键技术介绍 .....	3
1.3.1 研究目标介绍 .....	3
1.3.2 关键技术介绍 .....	4
1.4 论文结构安排 .....	5
2 相关背景知识与技术介绍 .....	6
2.1 传统 BIOS 技术介绍 .....	6
2.2 UEFI BIOS 标准介绍 .....	7
2.2.1 UEFI 概况 .....	7
2.2.2 PI 标准介绍 .....	9
2.3 ACPI 电源管理概念简介 .....	11
2.3.1 ACPI 各状态定义 .....	12
2.3.2 ACPI 标准中的 ASL 和 AML 简介 .....	13
2.4 硬件电源管理原理 .....	14
2.5 软件电源管理理论 .....	14
2.6 操作系统电源管理架构介绍 .....	15
2.7 本章小结 .....	16
3 系统总体设计 .....	17
3.1 睡眠唤醒后的 UEFI BIOS 环境创建 .....	17
3.2 网络下载使用模式整体设计 .....	18
3.3 UEFI BIOS 电源管理和网络下载流程 .....	19
3.4 系统和部件功耗测量方法 .....	20

3.5 本章小结 .....	21
4 系统关键技术与实现 .....	23
4.1 睡眠唤醒后的 UEFI BIOS 重建技术 .....	23
4.2 网络下载应用模式实现 .....	27
4.3 设备电源管理实现 .....	31
4.3.1 处理器的电源管理 .....	31
4.3.2 定时器的电源管理 .....	34
4.3.3 PCI 设备的电源管理 .....	35
4.3.4 PCI Express 设备的电源管理 .....	38
4.3.5 ATA 设备的电源管理 .....	39
4.3.6 USB 设备的电源管理 .....	40
4.3.7 内存的电源管理 .....	40
4.3.8 主板时钟发生器和锁相环的电源管理 .....	40
4.3.9 系统 IC 内部时钟的电源管理 .....	42
4.4 本章小结 .....	42
5 系统测量和数据分析 .....	43
5.1 测试平台和测量工具介绍 .....	43
5.2 测量的目的，原理和方法 .....	46
5.2.1 系统主要功率分布和不同使用模式下整体功耗比较 .....	47
5.2.2 电压转换器效率分析 .....	50
5.2.3 网络下载模式不同操作系统与本文模式下功耗对比 .....	51
5.3 本章小结 .....	54
6 总 结 .....	55
6.1 工作心得体会 .....	55
6.2 项目总结 .....	56
6.3 项目后续改进和展望 .....	58

---

参考文献 .....	60
附 录 .....	63
电路板三种模式下功耗分布详细数据 .....	63
致 谢 .....	72
攻读学位期间发表的学术论文目录 .....	73

# 1 绪论

## 1.1 论文研究的目的和意义

目前的台式机,笔记本和大部分服务器都广泛采用了基于英特尔公司(Intel)的 x86 架构。随着 x86 架构的普及和发展,用户对系统级别的电源管理提出了更高的要求。在移动计算领域,例如,基于 Intel Atom 架构的上网本和智能手机,用户提出了希望延长电池的使用时间、减少系统发热量、和降低散热器的噪音等要求。在家用领域,用户希望在晚间半价电费的时间内,能以低功耗的方式下载电影和歌曲。在服务器领域,用户希望降低服务器空闲时的功耗,一方面降低系统自身运行所需功耗;另一方面降低系统发热量从而减少散热所需功耗,从而节约电费来降低运营成本。在当今世界能源消耗日趋紧张,各国提倡低碳减排的背景下,在不影响用户的正常使用情况下,同时降低计算机的功耗,无疑节省了大量的能源,也降低了二氧化碳的排放量。

虽然在操作系统内实现了电源管理,但目前在系统进入操作系统前的 BIOS(Basic Input Output System) 环境中,却并没有实现电源管理功能。而随着 BIOS 技术的标准化发展,越来越多的厂商希望在该环境下开发更多的应用模式。例如,随着网络技术的发展,计算机病毒已经成为一个热点问题。计算机病毒的防护通常由杀毒软件在操作系统下运行,通过比较病毒特征数据库,找到病毒并将之删除。虽然杀毒软件的功能很强大,但是用户常常会碰到病毒无法删除的情况。这是因为病毒已经运行在操作系统内,具有很好的反侦测和反删除功能。而要解决这一问题,可以在系统启动时进入到 BIOS(Basic Input Output System)环境下完成杀毒。在该环境下,由于运行环境不同导致病毒不能运行,杀毒程序可以容易的侦测和删除病毒体。

以上想法需要应用模式长时间运行在 BIOS 环境下,但由于目前 BIOS 环境缺乏电源管理功能,不能满足前文中用户对电源管理的要求。为了支持 BIOS 环境下该特定应用模式,需要用户在操作系统和 BIOS 环境下进行切换。目前在两个环境下切换,只能重新启动计算机,然而计算机启动往往需要数分钟、甚至十多分钟的时间(依据启动应用程序的数目不同而不同),导致用户使用不方便。

因此,本论文将重点研究这两方面的问题。首先,是如何在 BIOS 环境下实现电源管理功能,并比较 BIOS 环境和操作系统环境下电源管理的优劣。其次,是找到一种技术方法,使 BIOS 环境和操作系统间切换变得更快捷,并且对用户更

透明、乃至不需要用户的操作。

## 1.2 国内外技术发展现状与趋势

在目前的电源管理研究中，分为软件和硬件两方面的研究。

软件层面从编译器、体系结构、操作系统等层次展开研究。编译器可以用更省电的指令来优化程序，例如：移位操作代替乘法。体系结构可以降低流水线深度，从而减少跳转导致的流水线刷新，以降低无用消耗<sup>[1]</sup>。当硬件系统被制造后，电源管理主要通过操作系统内的系统软件管理实现。在操作系统电源管理研究中，可以通过电源管理策略来实现，例如超时策略、预测策略、动态电压、动态频率管理等<sup>[2][3]</sup>。例如，在微软 Win7 的移动设备电池生命解决方案中<sup>[4]</sup>，它结合微软新一代的电源管理功能和要求，指导设备制造商和应用程序开发商，说明如何管理设备电源、如何设置电源管理用户策略、并指出了操作系统中如何实现这些功能。

硬件层面则提供了电源管理能力以配合操作系统实现电源管理。在电路级别板上，硬件实现了多路径供电的结构，可以在不同的睡眠状态下提供不同的供电分路，可以将更多的设备或设备功能在不使用时关闭。在部件级别上，硬件通过在空闲状态下关闭设备的部分功能，达到省电的目的。例如，硬件内可以通过动态关闭时钟(Clock Gating)来省电。虽然可以通过在空闲状态关闭设备，来达到省电的目的，但是当系统需要使用设备时，会导致额外的延时，而且恢复设备电源的动作也需要消耗功耗<sup>[5]</sup>。

因此，系统需要在功耗和性能间找到平衡点。而功耗和延时是电源管理的 2 个衡量指标。

在目前基于 x86 架构的计算机系统中，操作系统级别的电源管理都采用了 ACPI (Advanced Configuration and Power Interface)<sup>[6][7]</sup>标准，作为电源管理的实现标准。ACPI 电源管理分为 BIOS(Basic Input Output System)和操作系统两个部分。BIOS 提供伪代码 AML(ACPI Machine Language)和一些静态表格信息，来抽象底层的硬件设备，使操作系统有统一的操作接口。操作系统分析 AML 伪代码和这些表格，然后调用其自身的驱动程序，根据用户设置的策略来控制整个系统的电源管理。例如，ACPI 中对处理器定义的 P state，即是对动态电压和动态频率控制的实现。BIOS 提供 AML 代码，来说明每个 P state 能力和如何操作处理器内部的寄存器。操作系统则根据不同的负载情况和用户策略，来选择使用哪一个 P State。

虽然目前的操作系统在电源管理方面已经做得非常优秀。但是，在缺乏操

作系统的情况下，目前的 BIOS 包括最新的 UEFI(Unified Extensible Firmware Interface)<sup>[8]</sup>架构中，并不提供对 AML 代码的解析，也没有实现直接的硬件操作来管理系统电源。因此，在目前的 BIOS 环境中，不能提供电源管理，从而导致运行在 BIOS 中的程序一直处于高功耗状态。而导致这一现状的根本原因是：一方面，在于原有的 BIOS 环境封闭不利于创新，另一方面，是缺乏长时间运行在 BIOS 下的应用模。如果仅在启动阶段实现电源控制来省电，效果并不明显（时间太短），而且将带来启动时间延长的副作用。但如果新的应用模式，导致可以长时间运行在 BIOS 环境下，则对 BIOS 环境下实现电源管理提出了新的要求。

在目前主要的 BIOS 提供商中，例如国外或台湾地区的 AMI, Phoenix, Insyde 公司、以及国内信息产业部投资的百熬软件，都没有提供在 BIOS 阶段完善的电源管理功能。而一些 ISV(Independent Software Vendor)厂商和 OEM (Original Equipment Manufacture) 厂商，也希望由 BIOS 厂商实现通用的电源管理。这样，他们自己可以把精力放在更有商业价值的应用创新中，而不是陷入电源管理的技术问题之中。

### 1.3 研究目标与关键技术介绍

#### 1.3.1 研究目标介绍

本课题的研究目标是选用目前 Intel 最新的笔记本平台 Calpella 作为硬件开发板，其主要部件由 Core i5/i3 处理器，5 系列芯片组<sup>[9]</sup>，CK505 时钟发生器<sup>[10]</sup>，网卡和一些电压调节器<sup>[11]</sup>组成。该课题涉及到软件和硬件的知识。

为了支持电源管理，以及 BIOS 和操作系统环境之间快速切换，需要在 UEFI 环境下开发代码以实现目标。因此从大的研究目标看，需要做到以下几点：

- 1) 系统目标功耗达到与常用操作系统可比较的级别，力求做到功耗低于 Win7 下的功耗；
- 2) 在系统 S3 睡眠<sup>[6][7]</sup>唤醒后，进入到 UEFI 环境；
- 3) 在该环境中网络可以使用，例如 FTP 下载<sup>[12][13][14]</sup>；
- 4) 不用的设备进入到低功耗状态，而需要使用的设备在空闲时进入低功耗状态，在使用时唤醒；
- 5) 可以在 UEFI 环境和操作系统间多次快速睡眠唤醒，不需要重新启动计算机

系统；

- 6) 测量电路板上各主要部件在空闲和网络下载模式下的功耗分布；
- 7) 提出对目前 UEFI BIOS 架构对于电源管理方面的改进意见。

### 1.3.2 关键技术介绍

表 1-1 各硬件电源管理功能简述

Table 1-1 Brief functional description for device power management

硬件	电源管理方法
处理器	P state 和 C state 的电源管理 <sup>[1][6][7][15]</sup> 。  P State 可以降低处理器频率和电压,最深的 C State 会关闭处理器 Cache,把处理器内部状态存入 SRAM 中后, 关闭处理器运算处理单元
PCI/PCI Express	将 PCI <sup>[16]</sup> 设备放入低功耗的 D3 hot 状态, 启动 PCI Express 设备的链路层的动态电源管理 <sup>[17][18][19]</sup>
USB	USB 控制器和集线器上的端口设为休眠状态 <sup>[20]</sup>
SATA	根据动态电源管理, 将长时间不用的 SATA 硬盘放入到空闲或则休眠状态来省电 <sup>[22]</sup> , 但是需要分配大的内存来延长硬盘的休眠时间
网络设备	使用模式中需要一直使用网络, 但是在家庭的实际使用模式下, 考虑到一般使用 ADSL 等设备作为高速网络连接, 可以将以太网的速度从千兆降低到 10 兆或百兆
系统中断源 时钟	增大时钟的中断间隔, 使处理器可以更长的处于低功耗的 C State 状态。
电路板上时 钟和 PLL	电路板上许多设备处于非活动状态, 例如显示设备, 可以关闭某些设备的时钟和 PLL <sup>[23][24]</sup>

该课题涉及到 UEFI 系统软件和硬件两大知识领域, 且需要保证操作系统和 UEFI BIOS 能安全快速切换, 列举关键技术如下:

- 1) 实现对 S3 睡眠唤醒后、重新构建 UEFI BIOS 环境, 目前市场上没有商业实现版本, 需要实现在 Intel 32 位模式和 64 位模式间的切换<sup>[15]</sup>。
- 2) 实现对不同设备的电源管理, 并且在进入操作系统前能恢复其状态, 使所有设备在回到操作系统后仍然能正常使用, 需要对不同的操作系统测量和解决问题。



- 3) 为了实现电源管理，需要完成的测量工作有：
- a) 交流-直流转换器效率分析和能耗损失<sup>[25]</sup>；虽然软件目前不能控制其转换效率，但是可以得出其功耗，作为数据分析；
  - b) 各个部件的耗电量测量<sup>[25]</sup>。
- 4) 在 UEFI BIOS 中 S3 唤醒后的新环境中，实现网络协议栈功能<sup>[12][13][14]</sup>。在 UEFI 协议中已有完整的网络功能，例如 IP，TCP 协议等，需要开发 FTP 应用程序。
- 5) 代码量增大可能导致存储 Flash 空间不够用，需要提供压缩方法。可以利用目前 UEFI 中提供的无损压缩、解压缩逻辑，对 S3 路径中使用到模块压缩。
- 6) UEFI BIOS 环境数据与操作系统的交互。操作系统中由于磁盘缓存<sup>[26]</sup>的存在，可能导致在 UEFI S3 环境中写入的数据，不能在回到操作系统后显示出来。需要在操作系统中告之使其磁盘缓存无效，而使操作系统重新读取硬盘数据。

## 1.4 论文结构安排

论文的各部分安排如下：

第一章是绪论，对课题背景、研究目的、方法和最新国内外技术作了概括介绍。

第二章是相关知识和技术介绍，是对课题所涉及的国内外最新知识领域的说明和相关概念的介绍。

第三章是系统整体设计，描述了系统的两个关键技术，即 UEFI S3 BIOS 环境的创建和设备电源管理软件流程。

第四章是系统关键技术实现，分别详细的描述了 UEFI S3 环境创建，FTP 下载和设备电源管理功能的实现。

第五章是系统测试和验证，描述了如何搭建测试平台，如何设计测试方案，以及如何获得和分析数据。

第六章是总结，指出了本文项目的不足和改进，并分析了本课题应用的前景。

附录列出了测试得到的各部件详细的测试数据作为参考，由于数据列表太长故未将其放在第五章中。

最后是参考资料列表和致谢。



## 2 相关背景知识与技术介绍

本章首先介绍传统 BIOS 的作用和不足，UEFI 出现的原因，并列出了 UEFI 的优点。其次介绍了 PI 标准，它是 UEFI 接口的实现标准。接下来介绍 ACPI 的相关术语以及它在系统电源管理中的作用。然后简单介绍电源管理在硬件实现上的理论以及目前软件实现上的理论。最后介绍操作系统如何分层实现电源管理功能。本章的主要作用在于介绍本文后续涉及到的相关知识、背景和技术现状。

### 2.1 传统 BIOS 技术介绍

在 IBM 兼容机即 X86 体系架构中，BIOS 全称是基本输入输出系统（Basic Input Output System），是一种和硬件关系密切的系统软件。它一般存储在非挥发存储单元中，例如 Flash，EEPROM 等，作用是引导系统启动，识别和配置硬件，配置系统属性等。在 X86 系统中，启动的第一条指令即由 BIOS 提供，并且 BIOS 能访问和控制所有的硬件，特别是和主板上的硬件密切相关。在早期基于 16 位的操作系统中，例如在 MSDOS、Win95、Win98 和 WinMe 中，操作系统在运行时刻会使用到 BIOS 提供的软件接口来访问硬件。例如，通过软中断调用，Int 13 访问磁盘，Int 10 访问显卡等。但是在后续的 32 位系统以及 64 位系统中，操作系统在启动后，会利用自己的驱动程序来访问硬件。因此，目前在传统的 BIOS 中，BIOS 只在启动期间为操作系统提供服务（注：不包括 ACPI 等新的规范中定义的解释性代码方式）。

由于传统的 BIOS 是基于 16 位系统设计，并且代码运行在系统 1M 以下的地址空间。随着计算机硬件的发展，其不足越来越明显。例如，服务器常用到多种外接插卡硬件设备，而许多插卡上带有存储在 Option ROM 上的程序，在启动时 BIOS 会调用这些程序以初始化插卡设备。但是，由于位于 1M 以下空间越来越紧张，常常在不同的外设卡之间出现冲突。另一个缺点是，现有架构不符合当代计算机技术的发展趋势，当英特尔公司推出了安腾 64 位机的系统架构时，传统的 BIOS 架构已经无法满足其需求，从而推出了 Extensible Firmware Interface(EFI) 标准。而且传统的 BIOS 由于缺乏统一的工业标准，各个厂商只是利用了一些事

实准则作为标准，因此不容易鼓励创新。

## 2.2 UEFI BIOS 标准介绍

### 2.2.1 UEFI 概况

为了解决以上提到的传统 BIOS 问题，英特尔公司在 2000 年推出了 Extensible Firmware Interface(EFI)标准<sup>[8][27]</sup>，其特点是：

- 1) 操作系统无关性；
- 2) 文档化的标准接口，且用 128 位的 GUID 保证接口唯一性；
- 3) 模块化，方便二进制模块插入；
- 4) 提供标准化的系统服务；
- 5) 提供指令集无关性代码的支持，即提供解释执行功能；
- 6) 友好的支持高级语言，编程和接口有一定面向对象的思想；
- 7) 扩展性好，支持包括移动手持设备，笔记本，服务器；
- 8) 提供启动管理器且在启动阶段有丰富的接口调用，在运行期间也提供有限的服务。

英特尔公司在推出 EFI 后，又联合业界其他公司，例如：微软、惠普(HP)、凤凰(Phoenix)、包括其传统竞争对手 AMD，成立了 Unified Extensible Firmware Interface (UEFI) 标准组织，于 2005 年推出了 UEFI 标准，目前发展到 UEFI2.3 标准。

由于 UEFI 标准只是提供了接口、与实现无关，UEFI 组织后续又推出了基于实现的 Platform Initialization (PI) 标准<sup>[28]</sup>。到 2010 年，已经有多个操作系统版本支持 UEFI 标准，例如 Win7，Win2008 和多个 Linux 版本，多家 OEM 厂商已经将 UEFI BIOS 应用到多种产品之中。由于 UEFI 软件架构的开放性，越来越多的 OEM 厂商，独立 BIOS 厂商(Independent BIOS Vendor, IBV)以及一些软件厂商(Independent Software Vendor, ISV)，希望在这个架构中开发一些应用模式创新的产品出来。例如，惠普最新推出的 Quick Look 3 产品，即是在不用启动到操作系统前，就能察看日程安排的 UEFI 创新应用。而其他一些厂商希望在 UEFI 中进行杀毒，因为这样的环境更封闭，更有利于安全性。

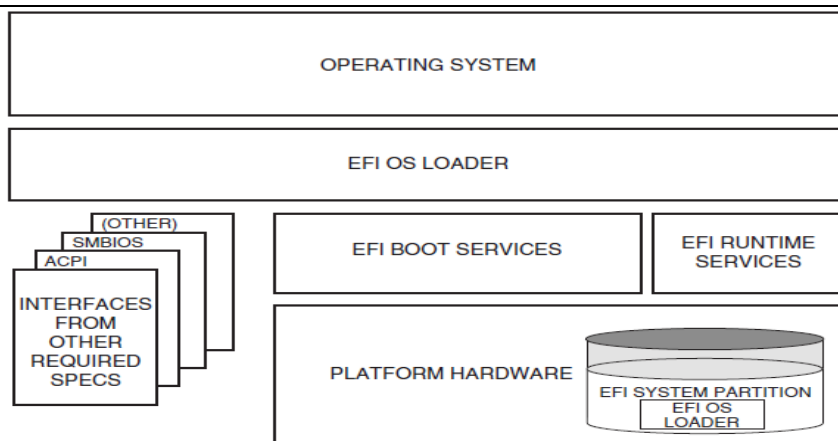


图 2-1 UEFI 概况

Figure 2-1 UEFI Overview

如图 2-1 所示，UEFI 接口可以直接管理硬件，并在规范中对标准硬件的操作抽象成标准的接口协议(Protocol)。UEFI 中的内核提供了基本的操作，分为启动服务(Boot Service)和运行时刻服务(Runtime Service)。其他模块则利用内核提供的服务，来产生接口协议或调用其他接口协议。UEFI 的操作系统加载器则利用 UEFI 提供的服务来装载操作系统。并且 UEFI 通过静态表格的方式提供了工业标准的访问，例如 ACPI，SMBIOS。由于现有操作系统还没有广泛支持运行时刻服务，所以现阶段运行时刻服务提供的功能非常有限。而启动阶段提供的服务功能接口协议非常丰富，包括对解压缩、网络连接、USB 设备驱动、硬盘驱动、显示等的支持。

UEFI 的一个特点是其仅支持单线程，只支持时钟中断，因此硬件设备的输入输出操作是采用轮流循环的方式访问。因此，该架构导致输入输出设备的吞吐量低于操作系统，操作系统的输入输出操作方式请参考文献[26]。但是 UEFI 结构更简单，更方便调试。

图 2-2 是 UEFI 的启动流程。在固件初始化好基本的硬件环境后，例如，处理器保护模式、内存等，开始加载 UEFI 内核，然后内核加载各个 UEFI 模块。当各个模块加载完成，用户需要启动到 UEFI 操作系统时，再装载 UEFI 启动装载器。UEFI 启动装载器负责加载操作系统。

从 UEFI 标准得知，UEFI 定义了标准的接口，并没有限制其如何实现。因此各个厂商可以灵活的开发自己的 UEFI 模块，然后可以发布二进制代码给 IBV 或则 OEM，使其可以集成到 UEFI BIOS 中而不用担心泄漏源代码，这保证了工业界的互操作性。

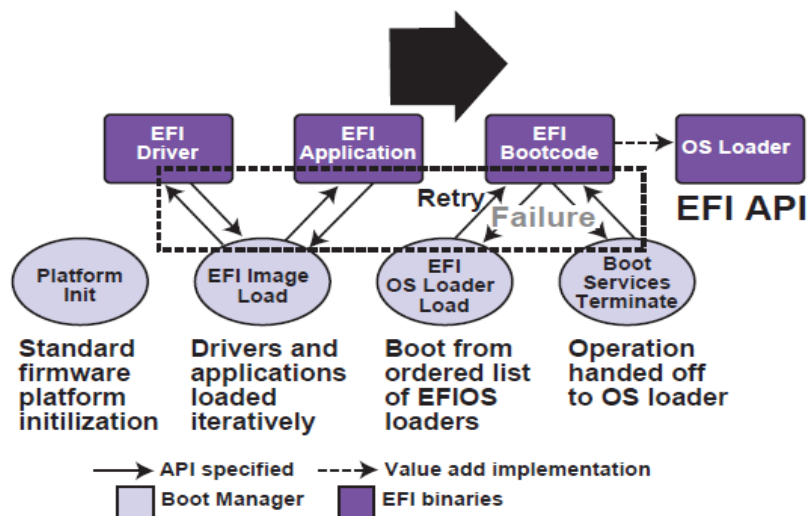


图 2-2 UEFI 启动流程

Figure 2-2 UEFI Boot Sequence

### 2.2.2 PI 标准介绍

从 UEFI 标准得知，UEFI 定义了标准的接口，但是并没有限制其如何实现。因此各个厂商可以灵活的开发自己的 UEFI 模块，然后可以发布二进制代码给 IBV 或则 OEM，使其可以集成到 UEFI BIOS 中而不用担心泄漏源代码。这保证了工业界的互操作性

UEFI 标准仅定义了接口，而 PI(Platform Initialization)标准<sup>[28]</sup>定义了更详细的实现。它定义了系统启动、非挥发性数据存储、SMM 模式、以及启动阶段的划分。事实上 PI 标准是在 UEFI 标准后推出，基于英特尔公司对 UEFI 接口实现的代码抽象出来的。

PI 标准定义了 SEC，PEI，DXE 和 BDS 各阶段，以及为了兼容传统 BIOS 的特殊的 CSM(Compatibility Support Module)阶段，以及它们之间的联系。

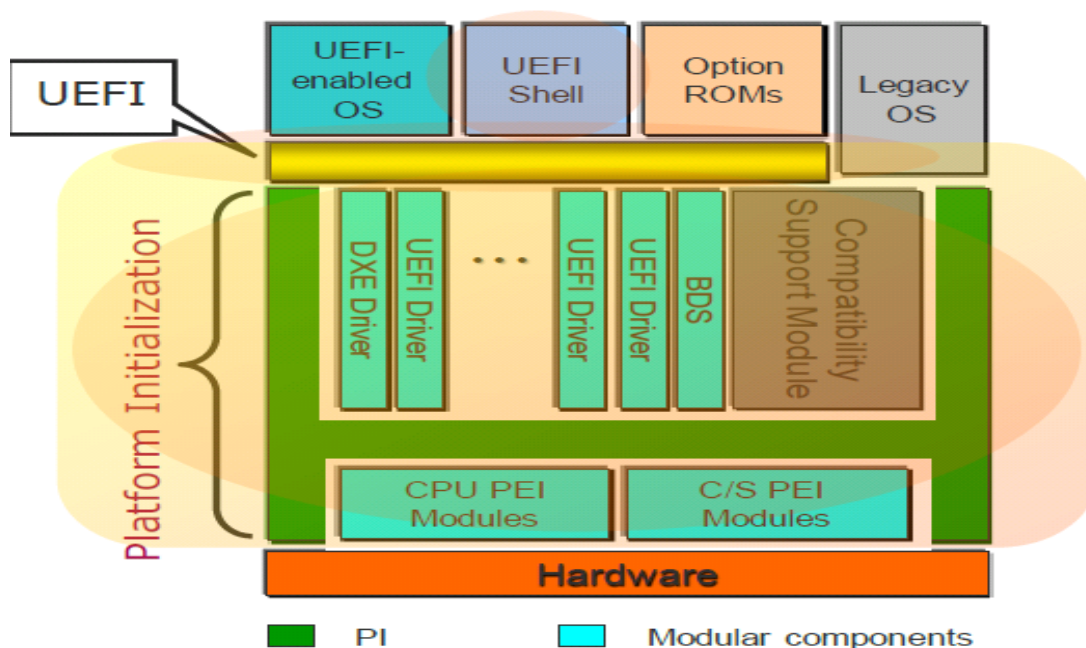


图 2-3 PI 和 UEFI 关系

Figure 2-3 The Relationship Between PI and UEFI

从图 2-3 可以看到，PI 内核提供了一个像字母 H 的框架，可以插入包含 PEI，DXE，BDS 和 CSM 等各阶段的软件模块。由这些模块和 PI 一起提供了 UEFI 接口，以支持 UEFI 标准的操作系统。为了兼容传统操作系统，CSM 模块将利用 PI 的接口转换成传统 BIOS 的标准，例如 INT 软中断调用。

PI 执行阶段分为如下几个阶段

- 1) SEC: 安全阶段(Security)，由于它是系统的第一条指令开始的地方，所以它被认为是可以信赖的阶段，并因此得名。它主要负责早期处理器的初始化，例如：加载处理器微代码(Microcode)、将缓存(Cache)设定为特殊模式用作存储单元、并转到下一 PEI 阶段。
- 2) PEI: Pre-EFI 阶段，主要工作在于初始化必要硬件，以保证系统内存可以使用，并将 PEI 自身加载到内存中。此外还判断启动路径，例如：是正常启动(S0)、还是休眠唤醒启动(S3)。
- 3) DXE: 驱动执行环境阶段(Driver Execution Environment)，在该阶段加载系统指定的 UEFI 模块，执行主要的硬件初始化动作，并生成各种符合 UEFI 规范的接口。
- 4) BDS: 启动设备选择阶段(Boot Device Selection)，用作系统策略的设定、用户配置界面的呈现、启动设备的选择。

5) Runtime: 运行时阶段, 通常已退出 UEFI BIOS 环境, 进入到操作系统阶段, 但是操作系统可以调用一些基本的 UEFI 系统服务来辅助工作, 但这需要操作系统支持 UEFI 标准。

### 2.3 ACPI 电源管理概念简介

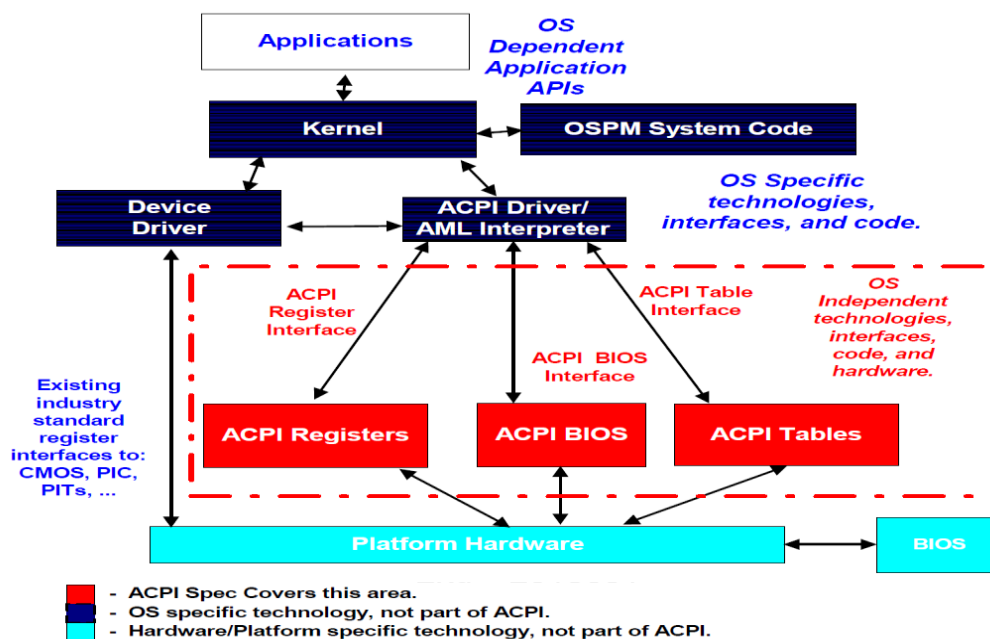


图 2-4 ACPI , 硬件和操作系统关系

Figure 2-4 The Relationship among ACPI , Hardware and Operating System

ACPI(Advanced Configuration and Power Interface)标准定义了电源管理的软硬件接口和功能。操作系统、驱动程序、BIOS、以及硬件根据这个标准一起协作完成复杂的电源管理功能, 如图 2-4。其中硬件提供标准的寄存器接口; BIOS 提供 ACPI 定义的表格和 AML 代码; 操作系统则解释 AML 代码、访问寄存器、分析表格数据、调用驱动程序; 驱动程序则负责各硬件状态的管理、保存和恢复等动作<sup>[6][7]</sup>。



### 2.3.1 ACPI 各状态定义

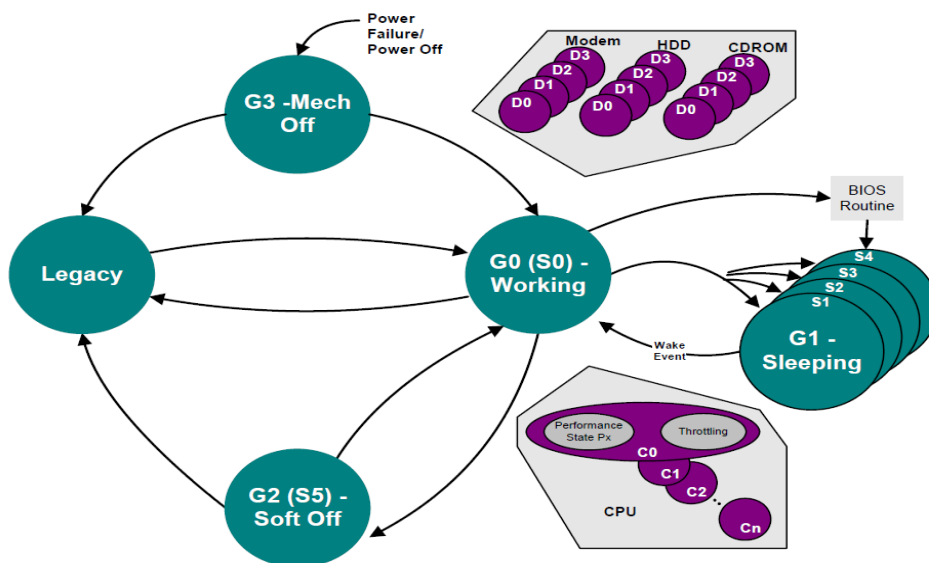


图 2-5 全局电源状态转换

Figure 2-5 Global Power State Change

由于 ACPI 的规范繁琐和复杂，且需要在了解其他工业标准和硬件芯片后才能深入了解其整个流程。为了更易于理解 ACPI，首先应该理解 ACPI 规范中定义的各种电源管理状态及其特点，如图 2-5。ACPI 定义了全局系统状态(G state)，分别为：

- G3: 机械关闭，即无任何交流电供给系统，此时系统仅靠纽扣电池供电，维持日期等内容。
- G2: 软关机状态，也即 S5 状态，即在有电源线插入供电的情况下关闭系统，系统原先的内容不保存，需要在启动后重新初始化。系统在这个状态下可以通过网卡或电源键等外设唤醒系统。
- G1: 睡眠状态，系统的部分硬件没有掉电，部分内容得以保存，例如内存信息不会丢失。因此在恢复到工作状态时恢复时间比较短。
- G0: 工作状态，系统处于工作中，外设的电源可以动态管理，例如可以把设备放入低功耗状态。

ACPI 定义了睡眠状态 (S State)，分别为：

- 1) S0: 工作状态，等同于 G0。
- 2) S1: 系统无任何信息丢失，包括处理器和芯片。
- 3) S2: 类似于 S1，但是处理器的信息会丢失，包括 Cache 内容<sup>[29]</sup>。
- 4) S3: 系统除了内存系统外，其他部分信息丢失，例如处理器信息，芯片信息

等。

- 5) S4: 系统进入休眠状态, 所有系统信息在保存到非挥发性存储设备后, 例如硬盘, 系统所有信息丢失。
- 6) S5: 即软关机状态。等同于 G2。

值得注意的是, 在现有的计算机系统中, 一般都支持从睡眠态中唤醒。如果系统在睡眠前被设定为可以唤醒, 那么支持唤醒的外设是由专门的供电电路来提供电源的。但是系统状态信息会丢失。

处理器电源状态定义为 C State, 分别为

- 1) C0: 处理器执行指令;
- 2) C1: 即处理器处于停顿状态, 在英特尔架构实现中, 其对应于 HLT 指令<sup>[15]</sup>;
- 3) C2: 和 C1 类似, 可以省更多的电, 但是恢复时间更长;
- 4) C3: 处理器的 Cache 信息保存, 但是不再执行窥探动作, 操作系统需要负责维护 Cache 得一致性。

值得注意的是, 在处理器省电技术发展初期 ACPI 规范中定义的 C state 和硬件规范中的 C state 是一一对应的。但是, 现在英特尔架构的 C state 已经发展到 C6/C7<sup>[1]</sup>。因此现在 ACPI 中的 C state 变成了接口定义, 它和处理器手册中的 C state 在实现的时候有对应的关联关系。

如果处理器处于 C0 工作状态, 还可以分为 P state, 即性能状态。P0 是处理器处于最高的性能和功耗, 对应于最高的频率和电压。Pn (n=1, 2, 3, ....) 是处理器的频率和电压逐渐降低, 性能和功耗也逐渐降低。其中英特尔处理器中的 Enhanced Speed Step 技术, 即是对 ACPI 中 P state 的实现。

设备的电源状态是 D state, 分别为

- 1) D0: 设备处于完全供电状态, 无状态信息丢失;
- 2) D1/D2: 设备部分信息丢失, D2 比 D1 更省电, 但需要更多的恢复时间, 值得注意的是, 不是所有设备都是实现了这两个状态;
- 3) D3: 设备电源关闭, 信息丢失。值得注意的是在实现中常常把 D3 分为 D3 hot 和 D3 cold。D3 hot 是系统仍有电源供应, D3 cold 是系统无电源供应。

### 2.3.2 ACPI 标准中的 ASL 和 AML 简介

ASL 全称是 ACPI Source Language, 它是 ACPI 代码开发人员的文本方式开发语言, 其特点是格式紧凑, 难于看懂, 可以定义访问硬件的方法。通过 ASL 编译器, 可以得到 AML(ACPI Machine Language)编码。AML 代码被操作系统解释后, 可以执行硬件操作。一般来说, ASL 代码中主要功能是系统和设备的电



源管理。它还可以提供封装接口，实现硬件相关性大的内容，而驱动程序实现操作系统相关内容，这样方便驱动程序移植性，例如集成显卡开发。ASL 代码的错误一般会导致系统不稳定、蓝屏、休眠后不能唤醒等系统错误，而且不易调试，因此是 BIOS 开发领域的重点内容。

BIOS 启动时提供 AML 和其他符合 ACPI 规范的表格，操作系统将利用它们得到系统信息、管理硬件、并开展系统电源管理。

## 2.4 硬件电源管理原理

电源管理中的最终实现是要靠硬件的管理。虽然笔记本电源是交流电，但是有一个交流转直流的适配器，因此电路板供电模式属于直流供电，功耗计算公式十分简单，为：

$$P = V \times I \quad (2-1)$$

P 为功耗，单位是瓦特；V 为电压，单位是伏特；I 为电流，单位是安培。

而根据 CMOS 电路的供电特点，CMOS 电路的绝大部分功耗是时钟信号导致的动态功耗。其动态功耗计算公式是（2-2）：

$$P = C \times f \times V^2 \quad (2-2)$$

P 为功耗，单位是瓦特；C 为负载电容，单位是法拉；f 为电路时钟频率，单位是赫兹，V 为电压，单位是伏特。该公式的推导请见参考文献[25]。

因此对设备的电源管理主要是通过降低设备电压或则频率，或是空闲时关闭时钟通道（Clock Gating）来实现。

为了得到系统功耗的分布，需要测量直流转直流，以及交流转直流的电压调节器效率，公式如下：

$$N = P_{out} / P_{in} \quad (2-3)$$

$$P_{VR} = P_{in} - P_{out} \quad (2-4)$$

$P_{out}$  是输出功率， $P_{in}$  是输入功率， $P_{VR}$  是消耗在电压调节器上的功耗。

## 2.5 软件电源管理理论

现有有关电源管理控制策略的各种论文和操作系统的实现中，其电源管理理

论主要是通过空闲的情况下关闭设备、或是将设备放入到低功耗状态来实现的。其主要探讨的范围：是如何检测设备的空闲状态和何时关闭设备，同时考虑到系统的延迟和负载情况来作决定。见参考文献[30]和[31]。

现有的一些国内外学术论文中讨论了操作系统的电源管理模型，主要分为：动态功耗管理(Dynamic Power Management)策略和动态电压调节(Dynamic Voltage Scaling)策略。其中前者主要是超时、预测、随机化策略，后者目前是根据负载调节电压来降低系统功耗。在现在实际的运用中，操作系统主要利用超时和预测策略来作电源管理，例如 Windows XP、Win7 可以设置各种设备的空闲时间，到时间后会把显示器或硬盘放入低功耗状态。而英特尔处理器 Enhanced Speed Step 技术就是对电压的动态调整，并且需要电压控制器来支持步进式的电压调整<sup>[2][11]</sup>，即：处理器通过发送代码给电压控制器，电压控制器根据代码调节供给处理器的电压。

## 2.6 操作系统电源管理架构介绍

现在的操作系统中从应用层，内核和驱动程序中提供了电源管理的功能。在微软的 Windows 驱动模型中(WDM)<sup>[32]</sup>，驱动程序分为总线驱动程序和设备驱动程序。在系统要进入到睡眠状态时，系统会发送 IRP(I/O Request Packet)来查询设备是否支持该状态，以及在该状态下对应的设备电源状态。

如图 2-6 所示，如果所有的设备驱动程序允许睡眠，那么系统会按照总线关系，先给子节点设备驱动程序 A 和 B 同时发送睡眠请求 IRP。当 A 和 B 处理好自己的任务后，例如：保存 A 和 B 设备的状态，以便唤醒后恢复现场，它们各自将收到的 IRP 向下层驱动程序传递。只有在总线驱动程序收到 A 和 B 的 IRP 包后，它才能保存总线设备状态，设置硬件到低功耗状态，最后完成 IRP。因此从设备关系来讲是从上到下。同理，当系统唤醒后，会按照先总线驱动、然后设备驱动的 IRP 传递顺序恢复现场，因此从设备关系来讲是从下到上。

如果是标准的 PNP 设备而且带有驱动程序，例如标准 PCI 设备、USB 设备等，这些设备并不需要 UEFI BIOS 来恢复其状态。现在的操作系统自身便拥有丰富的设备驱动程序，因此简化了 BIOS 对睡眠恢复的工作。

此外操作系统也会根据用户策略，例如：笔记本电脑可以设置性能优先或是节点优先，通过驱动程序来管理各个设备。在微软的 Win 7 移动电池寿命解决方案中，便列举了 Win 7 是如何管理各个设备的电源，来达到延长电脑使用时间的目的<sup>[4]</sup>。

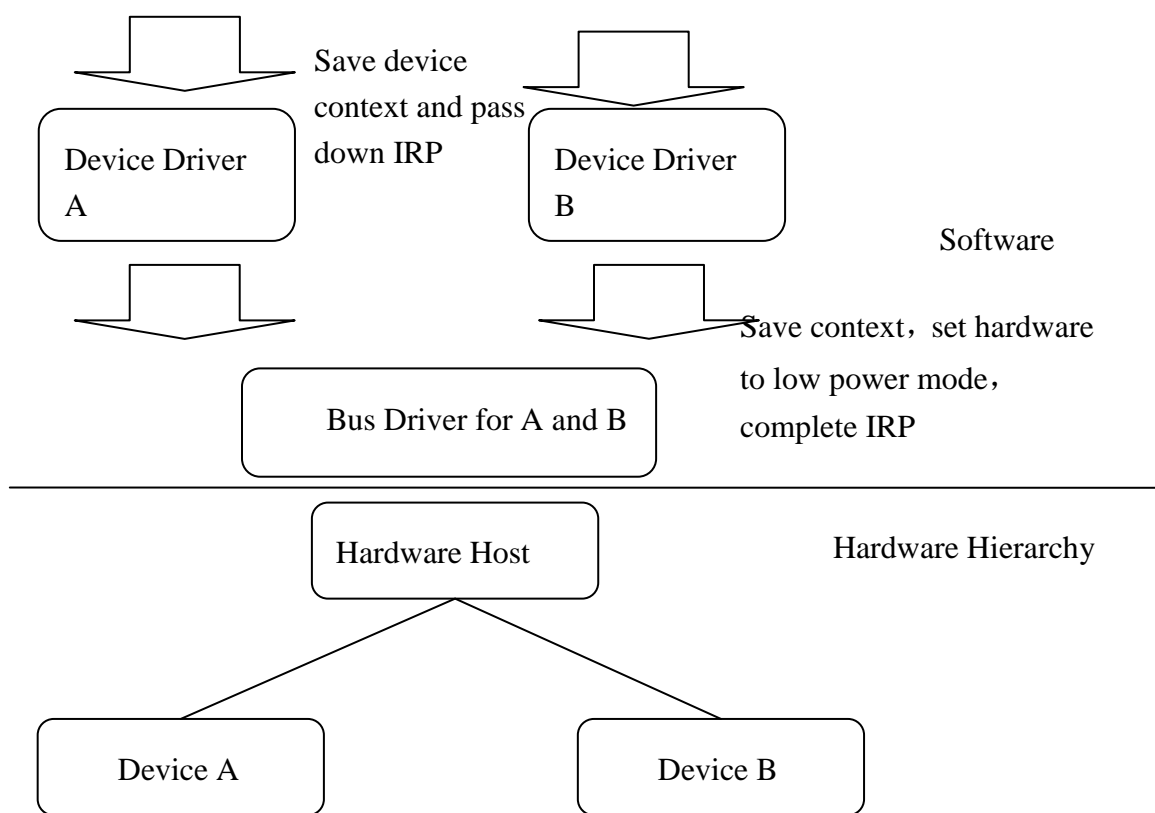


图 2-6 系统进入睡眠时电源 IRP 传递示意图

Figure 2-6 Power IRP Example When System in Sleep State

## 2.7 本章小结

本章介绍了 BIOS 的功能，UEFI 的特点，以及从不同角度介绍了相关电源管理的功能。例如从软件理论和硬件理论的角度来分析，从 BIOS 和操作系统的不同作用来分析，以及介绍了作为操作系统和 BIOS 电源管理协作工作的接口的 ACPI。

在后续各章节中会涉及到本章介绍的相关术语和概念，详细的功能性描述和细节请读者参考相关的参考资料。

### 3 系统总体设计

论文 1.3 节给出的研究目标，即在 UEFI BIOS 环境下对系统作电源管理，而该电源管理是针对特定的使用模式，论文选择一个典型的网络下载作为研究模式。而为了减少 BIOS 环境和操作系统相互切换的延时，则选择在睡眠唤醒时刻，作 BIOS 环境和操作系统环境的切换，即：需要在 S3 唤醒时，重新创建 BIOS 环境。因此系统总体设计由三部分组成，根据进入时间，可分为 3.1、3.2 和 3.3 三节。本章从总体上介绍系统方案，而系统实现细节在下一章进行介绍。

#### 3.1 睡眠唤醒后的 UEFI BIOS 环境创建

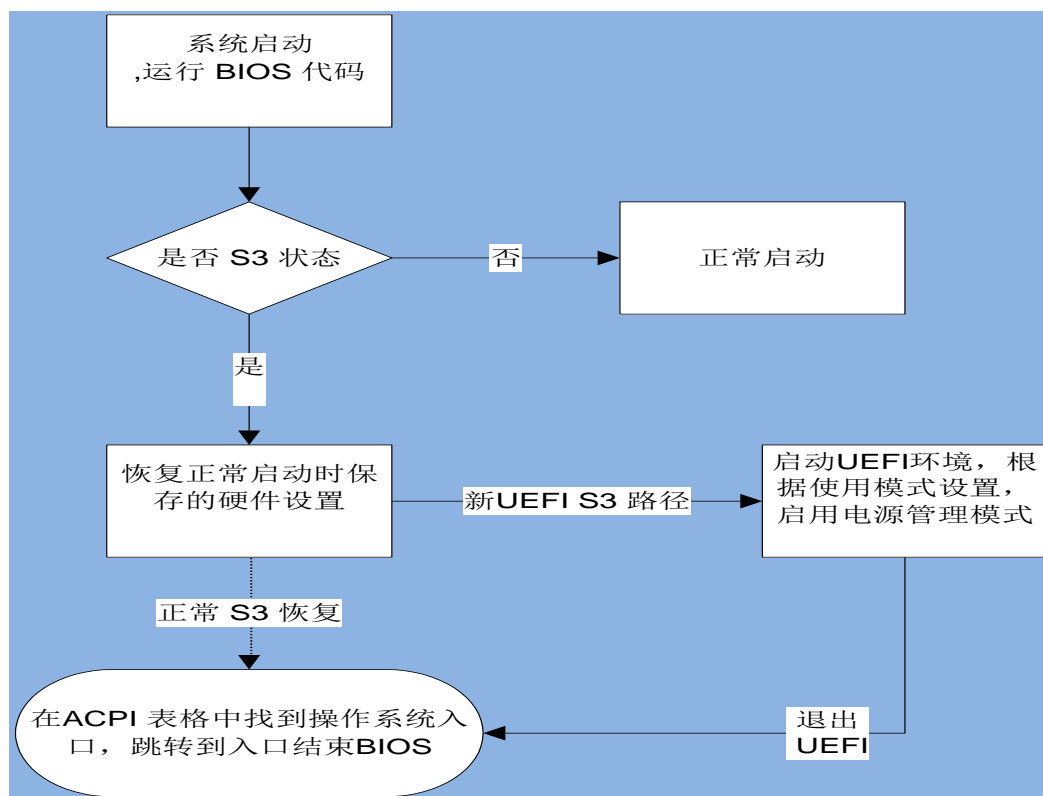


图 3-1 新 UEFI S3 路径

Figure 3-1 New UEFI S3 Path

当计算机在睡眠后，进入 ACPI 定义的 S3 状态。在 S3 状态时，处理器和所有 I/O 设备均处于掉电状态，仅内存模块处于自刷新状态，即保存着操作系统睡

眠前的状态。而支持唤醒的设备有辅助供电电路以提供唤醒功能。

如图 3-1 所示, 当计算机在睡眠状态 (S3) 时被外部事件唤醒, 例如电源按键, 键盘或鼠标动作等, 主板首先恢复各路电源供电路径, 然后时钟单元恢复工作, 最后处理器在收到重启信号后, 开始读取映射到 Flash 或 ROM 中的 BIOS(Basic Input Output System)代码并执行。即 BIOS 获得系统控制权, 开始了睡眠恢复流程。

传统的 BIOS 将重新初始化包括处理器在内的各个硬件, 并将内存从自刷新状态恢复到工作状态。即通过在 ACPI 标准中定义的一种表格, 找到操作系统入口地址, 然后跳转到该地址, 将系统控制权交给操作系统。接下来操作系统将执行其自身的睡眠恢复动作, 例如通过驱动程序恢复各硬件睡眠前保存的状态, 应用程序得到事件通知, 最后显示器点亮鼠标键盘工作, 让用户感觉到睡眠前的工作重新恢复执行。整个系统恢复过程所花时间一般在数秒内。

不同于传统的恢复过程, 本文中 UEFI BIOS 在初始化硬件后, 并不立刻把控制权交回到操作系统, 而是重新创建 UEFI BIOS 运行环境, 执行 UEFI BIOS 中指定程序。在该程序完成或用户自行中止后, 再将控制权交给操作系统。用户可以在操作系统环境下, 通过睡眠和唤醒实现快速切换到这个新的 BIOS 环境中的目标, 并且在该环境下可以实现电源管理功能。即可以根据使用模式来选择性的关闭一些外设, 或是只使用处理器某一单核, 而将其他核放到低功耗状态。

### 3.2 网络下载使用模式整体设计

由于本文中研究重点不是网络, 而是在使用局域网 FTP 下载的特定使用模式下, 测量系统各主要部件的功耗分布。并在相同的使用模式下, 对比操作系统和基于本研究结果的系统功耗, 因此仅需要实现网络的 FTP 下载功能。

总体设计上, 网络协议栈借鉴已有的 UEFI 标准网络协议栈, 实现 FTP 下载功能。网络相关内容见参考文献[12][13][14][26]和 UEFI 规范[8]。

协议栈从下往上的驱动程序分别为:

- 1) UEFI 千兆以太网驱动程序: 提供以太网的访问, 负责数据输出输入。其主要任务是收发以太网包, 对于 MAC 地址来判断是否属于自身, 然后去掉/加入包头包尾的信息和上层驱动程序交换;
- 2) UEFI 网络管理驱动程序: 该程序主要是封装接口, 是对网络驱动程序的抽象, 方便移植;

- 3) UEFI IP 驱动程序：负责 IP 数据的封装解析；
- 4) UEFI TCP 驱动程序：负责 TCP 数据的封装解析；
- 5) FTP 客户端应用程序：负责用户界面，数据存储，FTP 协议的实现等，调用 TCP 程序提供的接口。

### 3.3 UEFI BIOS 电源管理和网络下载流程

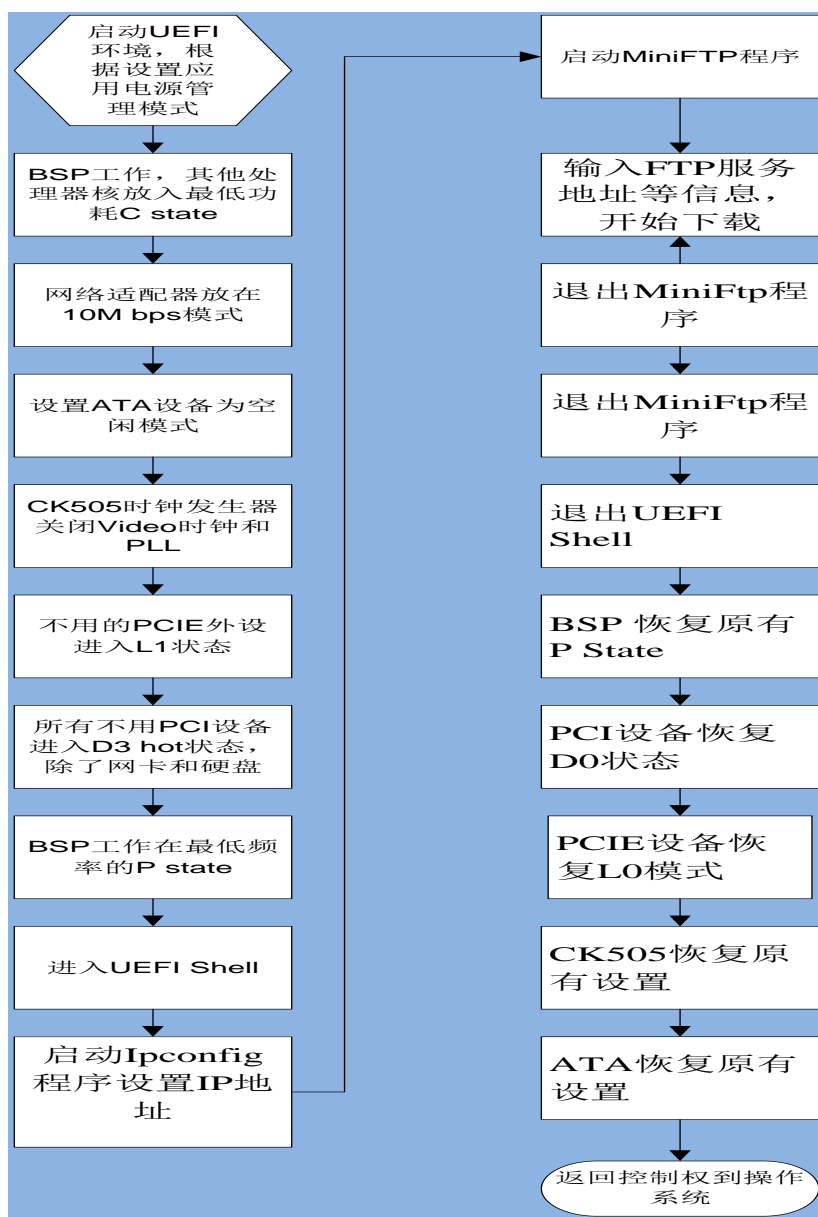


图 3-2 电源管理和网络下载流程

Figure 3-2 Power Management and Network Download Flow



图3-2是对电源管理流程的示意图。它是对图3-1中“启动UEFI环境，根据设置应用电源管理模式”部分的展开。描述了在S3 BIOS环境下，如何对各部件做电源管理，如何进入UEFI Shell，以及启动FTP下载的整体流程。

首先系统在 S3 恢复流程中进入到 UEFI 环境，然后启动包括处理器在内的各设备电源管理功能。因为后续要做网络下载，所以需要用到网络适配器和硬盘两种外设，而其他的包括显卡在内的设备都进入到低功耗状态。注意在实际工作中，显示器是不工作的，用户仅能通过电源按键或键盘退出该环境。而在开发环境中，将控制台操作例如显示和输入通过串口实现。

接下来进入到 UEFI Shell 环境，可以访问硬盘上的 UEFI 应用程序（也可以放在 Firmware Flash 中，放在硬盘仅是为了程序修改方便，不用重新烧录 Flash）。然后运行 IpConfig.efi 程序设置本地 IP 地址、网关和掩码。接下来启动 MiniFtp.efi 应用程序，输入局域网的 FTP 服务器地址连接成功后，就能执行标准的 FTP 命令，或是输入 Help 查询帮助。选择一个有若干 Giga Bytes 的文件下载，就可以在这种模式下测量系统整体和各部件功耗了。为了模拟实际中国家庭用户的宽带使用环境，在服务器端将速度限制在 400KB/s 内。

下载结束或用户中止下载后，退出 MiniFtp.efi 程序，然后退出 UEFI Shell，系统将恢复各硬件电源管理前的状态。然后退出 UEFI 环境，执行正常的 S3 睡眠恢复流程，重新回到操作系统中。

而返回到操作系统有两个难点。一是保证所有设备在 BIOS 阶段进行电源管理操作后还能正常工作。二是由于在 BIOS 环境下进行硬盘读写，但是在操作系统中的硬盘驱动程序，实现了对硬盘读写的缓存以提高读写速度；所以在返回到操作系统后，要使硬盘缓存无效，否则将导致硬盘数据不一致，使数据丢失或使硬盘数据损坏。以上需要在操作系统中利用系统 API 或是编写驱动程序来实现，请参见[26]和[29]中对磁盘缓存的描述。

### 3.4 系统和部件功耗测量方法

英特尔公司 Capella 笔记本开发板上，对每个主要 IC 的供电输入端都串联了一个高精度的精密电阻。通过使用精密仪器测试每个精密电阻两端的电压差，可以算出供电电流。然后再测量供电端的对地电压，就可以算出每个供电路径的输入功率，见公式（2-1）。像 CPU 这样的大型 IC 有许多个不同电压的供电路径，需要将各个供电路径的功耗相加，得到该 IC 的功耗。这样可以算出部件级别的功耗。

而对于系统整体功耗，需要使用单独测量交流电的测量仪器。而在主板的笔记本电源插口处有一个精密电阻可以测量出系统输入的直流功耗。 所以通过一次这样的全面测量，可以得到电路板的总体交流功耗、电路板总体直流功耗、和各部件的功耗。这里将其称为在某种应用模式下的一次功耗分布测量。

而对于本文的功耗对比，需要测量在网络下载使用模式下，不同下载速度的功耗分布，以及系统空闲状态下的功耗分布。除此之外，还需要测量在不同操作系统下不同使用模式的总体功耗，作电源管理性能对比。以上可以看出，本研究需要多大量的测量工作。

而为了测量分布在主板正面和反面多达近 140 个的测量点，需要对主板进行焊接将近 140 条 70 对测量线。这些都是作者自己动手焊接完成。焊接好的电路板如图 3-3 所示。

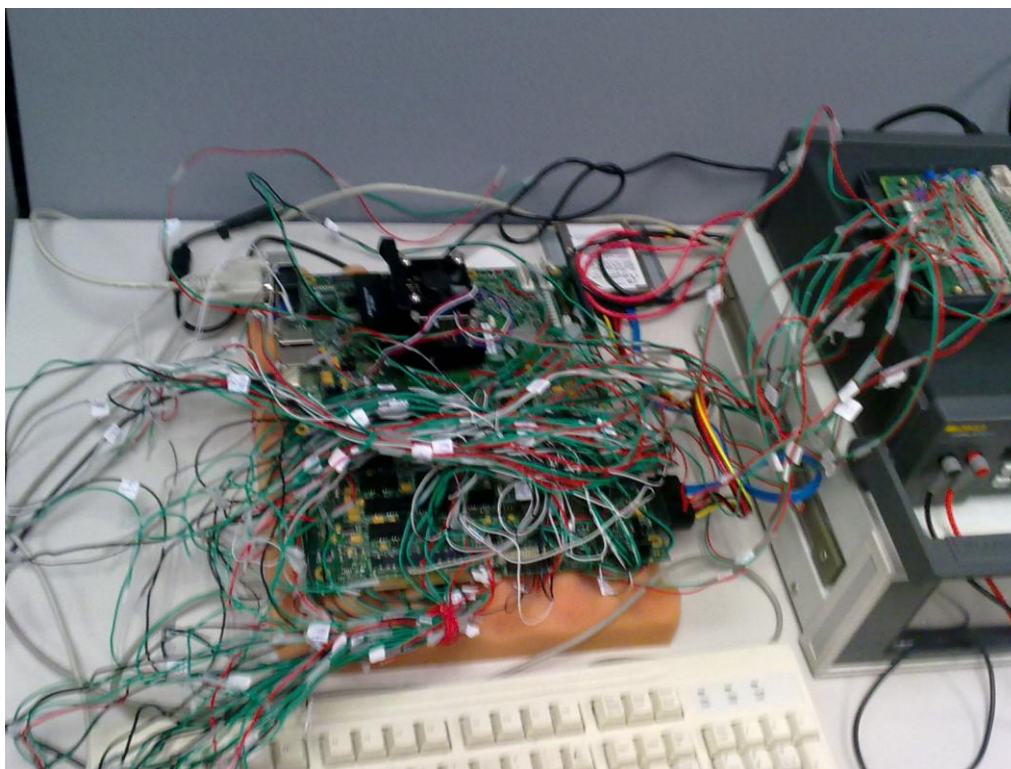


图 3-3 测量电路板实物图 比例:84%

Figure 3-3 The Photo of the Measurement Motherboard Ratio:84%

### 3.5 本章小结

本章介绍了本文的整体流程。首先是如何在 S3 唤醒过程中重建 BIOS 环境。然后介绍了本文中如何使用网络以及涉及到的网络协议层。接下来介绍如何在



FTP 下载的应用模式下进行电源管理，并展示了流程图.最后介绍本文如何测量功耗数据。下一章将对本章的内容展开，并作更详细的介绍。

## 4 系统关键技术与实现

论文涉及 UEFI BIOS 基础、网络协议栈、FTP 下载、软件系统级别电源管理、硬件基础知识和对各种设备的电源管理等多方面内容。由于该项目属于研究探索性质，因此重点在于在创建 S3 UEFI BIOS 环境, 在该环境下进行系统级别电源管理和得出电路板各主要部件功耗分布。

限于时间和经费等限制条件，对于用户易用性未做优化，一些命令或操作目前仍需要手工完成。本章将重点介绍 S3 UEFI 重建、网络下载的软件协议栈和应用程序、以及各设备的电源管理接口和实现。

### 4.1 睡眠唤醒后的 UEFI BIOS 重建技术

在 3-1 节中描述了如何在现有技术的基础上创建 S3 UEFI 环境的概况。本节将详细介绍如何具体实现其功能。

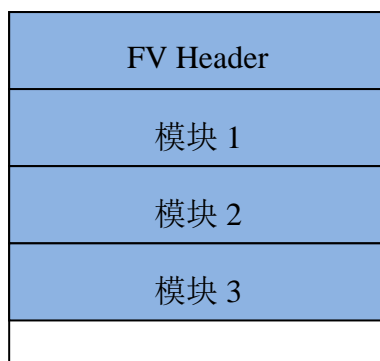


图 4-1 Firmware Volume 格式

Figure 4-1 The Format of Firmware Volume

Firmware Volume 是 UEFI 中定义的一种数据存放格式，其中可以包含代码模块或数据，如图 4-1 所示。而一个物理 Flash 部件中，存放了由多个 FV (Firmware Volume) 组成的 FD (Firmware Device)，即最终烧录到 Flash 的一个二进制文件中。把一个 FD 分为多个 FV 的目的，是为了在做 Flash 恢复时 (Recovery)，可以仅更新 DXE 的 FV 而不改变 PEI FV。这样做是为了更新时的安全。因为即使更新失

败，也还有机会重做一次或数次。

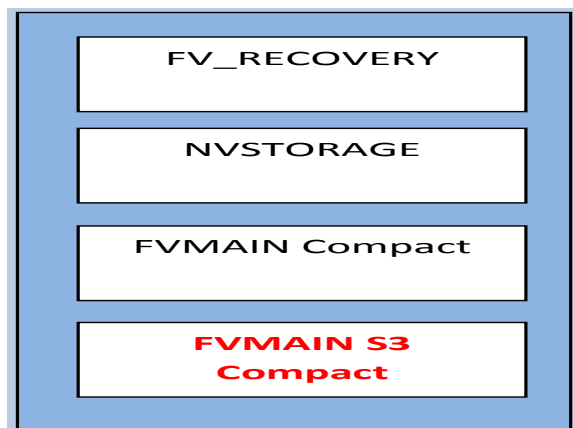


图 4-2 Firmware Device 格式

Figure 4-2 The Format of Firmware Device

在 UEFI 系统中根据启动阶段，分为 SEC/PEI 阶段的 FV(Firmware Volume) 和 DXE/BDS 阶段的 FV。在图 4-2 中，SEC/PEI 阶段的 FV 是 FV\_RECOVERY，由于需要在 Flash 上直接运行(Execution in Place)，因此不能在此阶段压缩。DXE/BDS/RunTime 阶段的 FV 是 FVMAINCompact，由于可以加载到内存中执行，因此可以被压缩以节省存储空间。此外还有一个单独的 NVSTORAGE FV 用来存储系统和用户配置数据。

而为了创建额外的 S3 BIOS 环境，需要加入一个新的 FV，在图 4-2 中名字是 FVMAINS3Compact。由于它可以在内存中运行，因此可以被压缩以节省存储空间。该 FV 中存放在 S3 BIOS 环境下需要加载的模块，这些模块大部分都和正常启动时完全相同，只有少许模块需要少量修改。使用单独的 S3 FV 是为了尽量减小对正常阶段模块的影响，提高开发速度和降低复杂性。

为了支持 S3 FV 的加载，需要在正常启动时将 S3 FV 加载到可用内存高端，然后将该段内存（用作代码存放）标记为保留，并且需要再分配一段内存用作数据访问的堆栈。因此需要在原有的 FVMainCompact FV 中加入一新的模块 S3FVLoader，该模块功能如图 4-3 所示。其详细工作工程如下：

在正常启动过程中，DXE 内核载入 S3FVLoader 模块。该模块会分配名字为 DXE\_S3\_VARIABLE\_SET 的变量，其类型是 ACPI None Volatile（ACPI NV），即为预留类型，操作系统不会使用该内存。然后将内存变量的指针保存到 Flash 中，以方便在 S3 启动时从 Flash 上得到该指针。而该变量用来保存一些全局信息，例如下面的 S3 FV 在内存中的地址和大小，可用来存贮数据和作为栈的内存地址等。

接下来将 DXE 内核模块加载到高端内存，类型仍为 ACPI NV。并且从 Flash 中找到压缩的 S3 FV，然后解压缩到高端内存，标记类型为 ACPI NV。并且还需要分配 64M 的内存作为堆栈。

如果是 64 位系统，还需要重新定位模块 PpiNeededByDxeFileName。该模块为 64 位 UEFI DXE 内核所需。

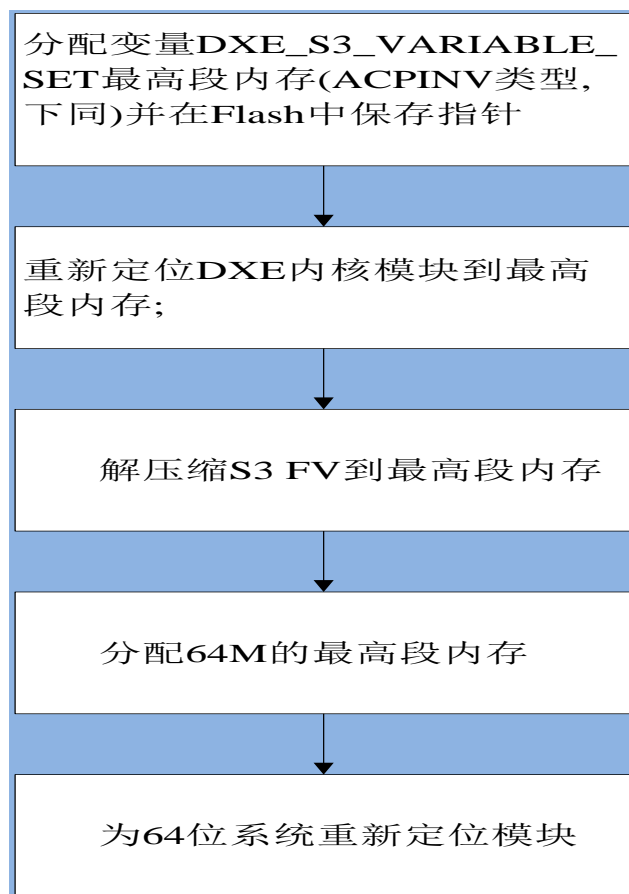


图 4-3 S3FVLoader 工作流程

Figure 4-3 S3FVLoader Work Flow

在以上内存分配的步骤中，需要更新 DXE\_S3\_VARIABLE\_SET 全局变量里面的一些信息来保存状态，例如，分配内存的起始地址和大小。并且使用的内存类型是 ACPI NV，即该内存是操作系统不能使用的内存，专门为 UEFI BIOS 使用预留。

这个阶段的主要目的是将 S3 FV 解压缩到内存中，并分配好 S3 阶段运行时所需要的内存。一方面确保这些内存不会被操作系统使用，另一方面将这些信息保存在预留的内存中。

如图 4-4 所示，系统从 S3 睡眠状态唤醒后，仍由 BIOS 代码接管系统控制权。

BIOS 首先调度 PEI 内核和模块，然后在初始化部分硬件后，根据某些寄存器状态判断系统是 S3 唤醒或是开机启动 (S0)。如果是从 S3 唤醒，将恢复启动阶段保存的硬件状态，然后找到操作系统入口地址，跳转到该地址。此后系统将由操作系统控制，BIOS 阶段结束。

而为了在唤醒阶段进入 BIOS 环境，需要在系统将控制权交给操作系统前保护现场；然后从 Flash 中得到 DXE\_S3\_VARIABLE\_SET 变量的指针，从而可以在内存中找到 S3 FV 位置和运行时可用堆栈等信息。

接着到已经解压缩的 S3 FV 位置，将该 FV 作为普通得 DXE FV 装载。系统开始建立 DXE 环境。然后系统像正常启动一样，加载 S3 FV 中的各个模块以确保所有基本的 UEFI 被建立。接下来执行 S3 FV 中的 BDS 阶段，此后找到可以启动的程序，例如 UEFI Shell，从而进入到 UEFI Shell。此时基本的 UEFI 环境建立。

此刻可以根据应用模式，加载不同的管理程序和模块，例如播放音乐、加载网络协议栈、执行备份程序等。并且可以在此时应用系统电源管理策略，将不使用的设备放置到低功耗状态。

例如杀毒时可以将 USB 设备，显示器，各外接卡，网卡等设备保存状态后，放到低功耗状态，而将硬盘和处理器处于最优性能状态。在杀毒结束后，将各进入低电源管理模式下的设备恢复到正常工作模式。

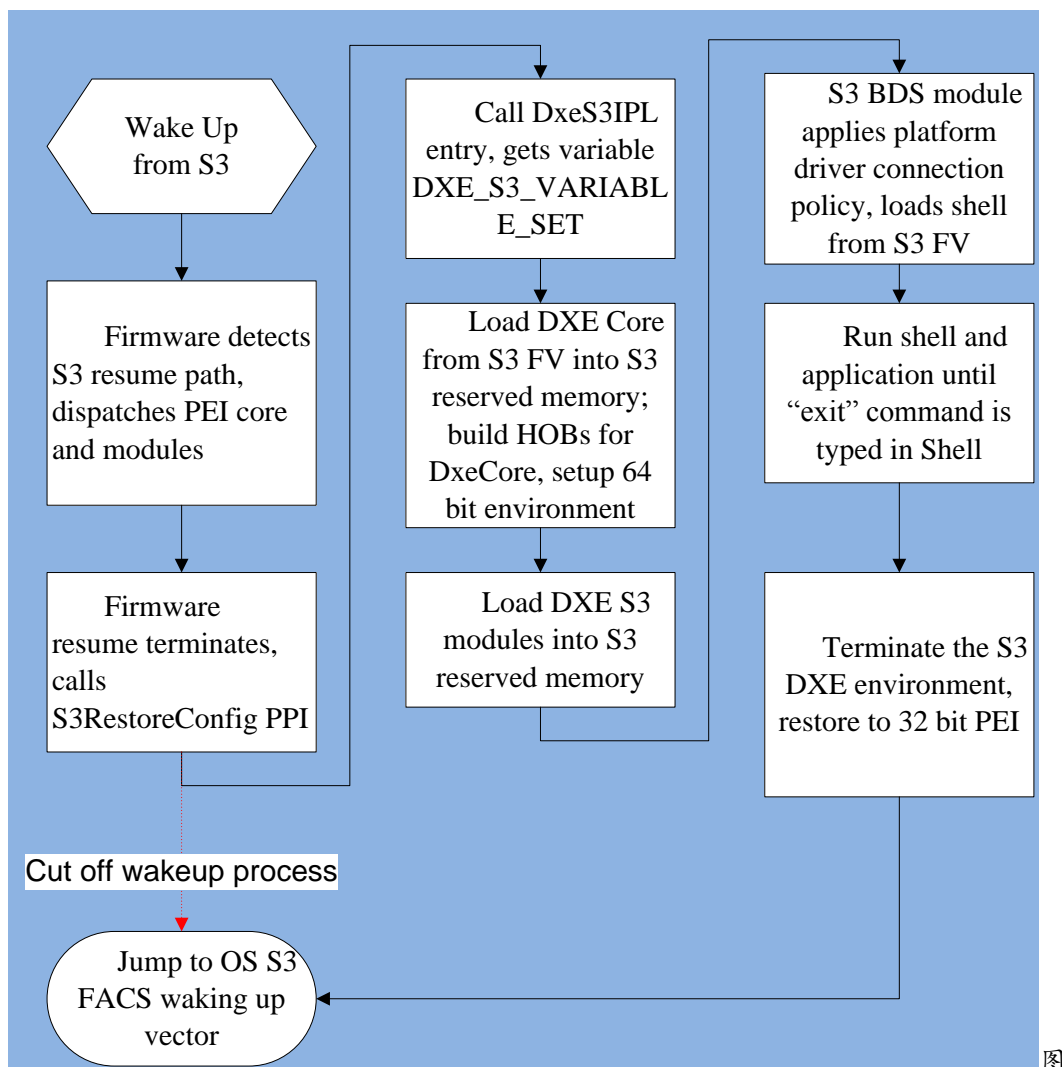
而在网络备份应用中，将网络设备、硬盘放到最优性能状态，而多核处理器只需要一个核处于低频工作模式，而其他核进入到 ACPI 定义中的 C State 状态，可以节省更多功耗。

在所有模式中，策略的制定都可以在操作系统中执行，并且将结果保存到 Flash 中，在 S3 BIOS 启动后，通过读取 Flash 中信息来获得策略。并且一些复杂操作可以分为两部分：一部分仍然可以在操作系统中完成，例如杀毒软件的更新；而杀毒的过程耗时长，可以进入到 BIOS 环境中完成，并且在 BIOS 环境中杀毒更安全。

当系统完成了用户安排的任务或被用户中断任务，可以退出 UEFI Shell，将进入到 Shell 前的状态恢复。并且确保各硬件状态恢复到进入 Shell 前的状态。然后继续加载操作系统。根据策略，“退出”这一命令的发出可以是电源按键，或是时钟超时，或是鼠标键盘，甚至是通过串口发出的命令等事件或动作。

此后 BIOS 将从 DXE 工作模式切换到 32 位 PEI 模式，然后寻找 ACPI 规范中定义的 FACS 表格，接着找到操作系统入口地址，跳转并结束 BIOS 环境。此后操作系统将接管系统。而操作系统并不知道 BIOS 已经作了一些磁盘操作，因此需要

在操作系统内部调用一些 API 使磁盘缓存无效，以便保持数据一致性。



4-4 睡眠唤醒 UEFI 环境创建/退出流程

Figure 4-4 S3 Resume UEFI Environment Creation and Exit Flow

## 4.2 网络下载应用模式实现

如前文所诉，UEFI S3 环境下的应用模式主要有杀毒、网络下载。本论文主要针对网络下载的应用模式做电源管理优化。因为杀毒应用模式只需要使用硬盘、处理器和内存；而网络下载需要使用硬盘、网络适配器、处理器和内存。因此针对网络下载模式进行研究，可以覆盖更广的范围。

参见图 3-2 中的进入 UEFI Shell 后的流程，首先需要启动 IpConfig.efi 应用

```
graph TD
    MiniFtp.efi[MiniFtp.efi]
    subgraph " "
        direction TB
        I1["<<interface>>  
EFI_TCP4_PROTOCOL"]
        I2["<<interface>>  
EFI_IP4_PROTOCOL"]
        I3["<<interface>>  
EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL"]
        I4["<<interface>>  
EFI_SIMPLE_NETWORK_PROTOCOL"]
        I5["<<interface>>  
EFI_NETWORK_INTERFACE_IDENTIFIER_PROTOCOL"]
        I6["<<interface>>  
EFI_PCI_IO_PROTOCOL"]
        I1 --> I2
        I2 --> I3
        I3 --> I4
        I4 --> I5
        I5 --> I6
    end
    I6 --> NA[Network Adapter]
    subgraph " "
        direction TB
        I7["<<interface>>  
EFI_PLATFORM_POWER_POLICY_PROTOCOL"]
        CPU[CPU]
        I7 -.-> CPU
    end
    subgraph " "
        direction TB
        I8["<<interface>>  
EFI_SIMPLE_FILE_SYSTEM_PROTOCOL"]
        I9["<<interface>>  
EFI_DISK_IO_PROTOCOL"]
        I10["<<interface>>  
EFI_BLOCK_IO_PROTOCOL"]
        I11["<<interface>>  
EFI_PCI_IO_PROTOCOL"]
        I8 --> I9
        I9 --> I10
        I10 --> I11
        I11 --> SATA[SATA Device]
    end
    FAT[FAT Driver] -.-> I9
    ATA[ATA Device Driver] -.-> I10
    PCI[PCI BUS Driver] -.-> I11
```

Figure 4-5 The Interface Stack Used by MiniFTP Application

28



想的实现，考虑了 BIOS 环境面向过程的特点，同时也减小了 C++ 语言不必要的时间和空间上的开销，加快了启动速度和节省了 Flash 上的存储空间。

图 4-5 中左边是网络协议栈。最下面是千兆网络适配器硬件，负责网络传输物理包的传输。硬件上一层是 PCI 总线为枚举到的网卡所安装的 PCI IO Protocol，该层 Protocol 支持对硬件接口的操作，例如 PCI 配置空间访问、I/O 读写封装为函数调用，方便在各种架构移植，例如 IA32、X64 或者 IA64。

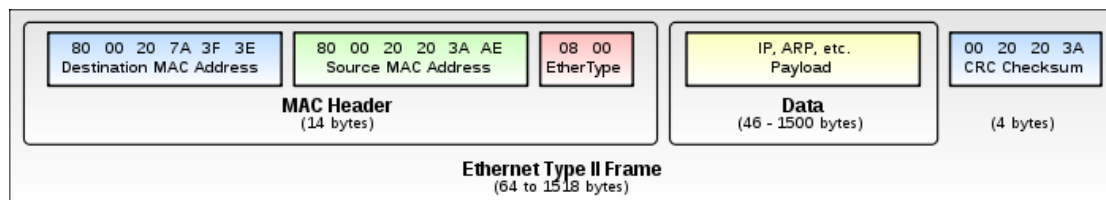


图 4-6 以太网包格式<sup>[36]</sup>

Figure 4-6 Ethernet Frame Format<sup>[36]</sup>

在 PCI 驱动上层的是真正的网卡驱动程序，它将使用 PCI IO Protocol 来操作网卡。网卡接收和发送的物理包格式如图 4-6 所示。该格式是 DIX 格式，由英特尔、DEC、Xerox 等公司主导，也是本论文试验中实际所使用的格式，称为 Ethernet Type II，和其对应的是 IEEE802.3 格式。网卡将管理发送和接收两个硬件 FIFO 缓存，接收时用来存放和本网卡 MAC 地址一样的数据包，发送时将网卡自己的 MAC 地址打包进去。网卡驱动程序将会从这些包中提炼出目的地址、源地址、包类型等信息，并检查包的完整性。然后根据类型(Ether Type)对数据区域做相应处理。接收时网卡硬件会将 FIFO 缓存中内容，通过 DMA 方式放到指定的系统内存地址。发送时网卡硬件会到系统内存中获取数据然后打包发送。网卡驱动抽象出了 Network Interface 的 Protocol。

在网卡驱动上两层是 UEFI 规范中定义的软件实现的网络管理层。具体解释请参见 UEFI 规范<sup>[8]</sup>。再上面一层为网络协议中定义的 IP 层，它主要解释从图 4-6 中以太网包里面提取的数据，并可以得到目的地和源地址的 IP 信息。在 IP 层上是 TCP 层，实现 TCP 协议。其中 IP 层和 TCP 层有 IPv4 和 IPv6 两个版本，将根据用户配置动态加载模块。而 MiniFTP 程序将使用网络协议层中的 TCP 协议实现 FTP 协议，例如，作 FTP 网络传输和用户校验等操作。MiniFTP 作为协议栈的最高层，将负责为网络协议层分配和释放数据内存，而放在内存中的数据将在网络协议栈和磁盘访问栈中交换。



普通代码	电源管理代码
<pre> T = 0; While (T &lt; TIMEOUT) {     Staus = GetIoStatus ();     if (Status = EFI_SUCCESS) {         break;     }     T += DELAY_1;     Stall (DELAY_1); } If (T == TIMEOUT) {     Status = EFI_TIMEOUT } </pre>	<pre> T = 0; TimerPeriod = GetTimerPeriod(); While (T &lt; TIMEOUT) {     Staus = GetIoStatus ();     if (Status = EFI_SUCCESS) {         break;     }     T += TimerPeriod;     PowerPolicy-&gt;CpuIdle (); } If (T == TIMEOUT) {     Status = EFI_TIMEOUT } </pre>

图 4-7 I/O 处理伪代码比较

Figure 4-7 Faked Code Comparison for I/O Processing

图 4-5 中间的 Protocol 负责对处理器进行电源管理,它提供抽象的电源管理接口,在等待 I/O 设备完成任务时,提供处理器空闲等待时的节电处理。由于在 UEFI 架构中,并不提供多线程任务处理,也不提供通用的中断处理,因此是通过轮循的方式,来判断 I/O 操作是否完成。

图 4-7 左侧是常见的 UEFI BIOS 中的 I/O 处理等待伪代码。在 While 循环中,查询时间计数 T 是否超过 TIMEOUT 值,是则表示 I/O 操作超时,命令失败。GetIoStatus 将查询 I/O 设备操作是否已经完成,一般通过查询寄存器状态位可得到。如果 I/O 任务完成,那么退出循环,否则调用系统 Stall 函数等待。而 Stall 函数的实现是不停的对 Timer 时钟查询,直到 DELAY\_1 的时间到达。因此普通的 UEFI BIOS 代码将是通过处理器不停的轮循 I/O 设备和计数器状态,来控制 I/O 操作是否完成。由于处理器一直在运行,所以处理器的功耗在 I/O 操作时,

大部分被浪费掉了。而图 4-7 右图中，虽然基本流程和左图相同，不同的是其调用 `EFI_PLATFORM_POWER_POLICY` protocol 提供的 `CpuIdle` 函数。该函数将处理器放入省电的 C State，处理器停止工作，然后被系统的周期性时钟中断唤醒，间隔是 `TimerPeriod`。因此 `CpuIdle` 函数的值是从系统周期性中断时钟获得。该实现使得处理器大部分时间处于低功耗状态，达到省电的目的。注意图右是简单处理的伪代码，实际代码比这个复杂，甚至可以动态的调整中断时钟周期。

图 4-5 中右边是磁盘控制，文件系统协议栈。从下到上分别是 PCI 总线枚举到的 SATA 设备控制器（抽象出 `PCI I/O Protocol`），SATA 总线枚举到的磁盘设备（抽象出 `BLOCK I/O Protocol`），以及磁盘上的文件系统(FAT 格式)。因此该协议栈的作用是提供磁盘读写能力。而 MiniFTP 直接通过类似于操作系统中的 `open`, `read`, `write`, `close` 等文件操作，直接向磁盘写入从网络中得到的数据或是从磁盘中读数据向网络传输。

综合来说，MiniFTP 应用程序通过调用不同的 `Protocol`，达到在磁盘和网络间数据传输的目的，并且通过电源管理接口优化 I/O 操作等待时的功耗，达到在不影响系统使用 and 性能的前提下，省电的目的。

### 4.3 设备电源管理实现

电源管理中有一对矛盾的参数，分别为功耗和延迟。即如果系统省电效果好，需要关闭更多的功能部件，那么当系统需要恢复到工作状态时，会导致更多的恢复时间：即延迟。因此需要根据工作特点和用户要求，在省电和延迟之间取得平衡。本节将根据 MiniFTP 的应用模式和使用特点，来实现相关设备的电源管理，即：网络和磁盘处于工作状态，处理器处于工作状态，但是不需要强大的处理能力；而其他外设处于非工作状态，则可以放入低功耗状态。

根据本节电源管理得出的数据，请参见第五章的表 5-3。在表 5-3 中列出了系统主要部件实现电源管理功能前后的功耗对比。

#### 4.3.1 处理器的电源管理

处理器的电源管理在硬件支持上分为工作状态的 P State 和空闲状态的 C State。

在英特尔处理器电源管理 P State 技术的实现上，P State 又称为 Enhanced Intel SpeedStep Technology (EIST)。其原理是处理器根据性能和电源管理，支持多

个工作频率和电压点。软件可以通过 Model Specific Register(MSR)访问来控制处理器频率。而硬件根据所设定的频率来调整电压。根据公式（2-2）可知，降低 CMOS 器件的频率和电压，都将降低动态功耗，达到省电的目的。

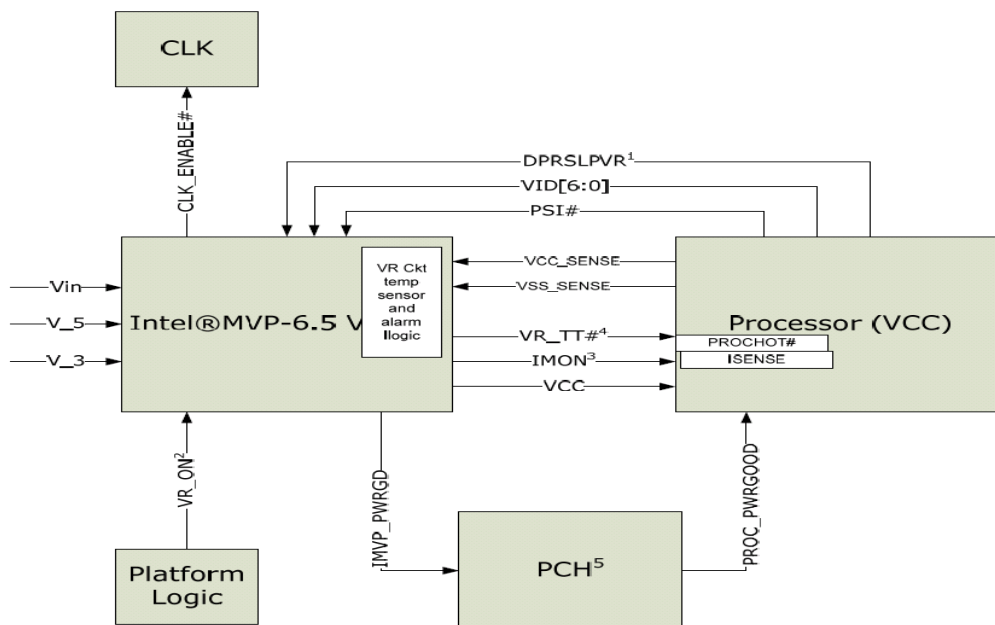


图 4-8 处理器核心电压调整器接口模块图

Figure 4-8 CPU Core VR Interface Block Diagram

图 4-8 是处理器 EIST 电压控制技术实现中，处理器和电压调整器之间的硬件接口示意图。Intel MVP-6.5V 是电压调整器<sup>[11]</sup>，负责将 3.3V 和 5V 的电压转变为处理器可以使用的核心电压 VCC。而 VCC 电压的大小可以通过处理器输出总线的 VID[6:0] 调节。例如，VID[6:0]=0：可以把 VCC 设置在 1.5V；VID[6:0]=0x40：可以把 VCC 设置在 0.7V；VID[6:0]=0x7F：把 VCC 设置在 0V。注意该机制只能调节处理器的核心电压 VCC，处理器还有其他供电路径将在数据测量时讨论。

$$\text{处理器内部的频率} = \text{倍频参数} \times \text{外部输入频率} \quad (4-1)$$

处理器频率的调整，是通过改变处理器内部的倍频来实现的。而倍频的改变可以通过访问 MSR 寄存器实现。软件可以在任何时候控制 EIST。如果前面的操作正在进行，那么硬件将自动等待前面的操作完成后，再执行新的操作。如果目

标频率大于现有频率，VCC 通过 VID[6:0]分步逐次提升，直到对应的电压，然后 PLL 锁定到目标频率。如果目标频率小于现有频率，PLL 首先锁定到目标频率，然后 VCC 通过 VID[6:0]分步降低到对应的电压。由于 EIST 操作点的转变延时很小，每秒可做许多次转变。

在 C State 管理上，本次研究的 Capella 平台上所用的 i3 处理器有表 4-1 描述的几个状态。

表 4-1 处理器 C State 功能简述

Table 4-1 Brief description of Processor C State

状态	描述
C0	活动模式，处理器工作
C1	AutoHALT 状态.
C1E	AutoHALT 状态，并且处理器工作在最低的电压和最低的频率
C3	写回 L1 数据和代码 Cache，L2 Cache 到 L3 Cache，时钟关闭
C6	执行单元保存架构状态，然后核心电压掉电

由于目前的处理器已经到了多核时代（在软件逻辑上，超线程也认为是单独的核），但是在 UEFI 环境下主要是做 I/O 操作，并不需要处理器强大的处理能力。因此，需要把不用的核称作 Application Processor (AP) 放入 C State；且不可以被中断唤醒；使用的核称作 Boot Strap Processor (BSP)，在不用时也放入 C State，但是该 BSP 可以通过中断唤醒。这样处理器耗电最小。由于 MiniFTP 应用中对处理器性能要求不高，所有 AP 和 BSP 均可设置在最省电的 P State 运行。即通过修改所有逻辑内核的某一特定 MSR 实现。

图 4-9 是所有 AP 的 C State 控制汇编代码片断，汇编代码编写格式见参考文献[37]。由于 AP 在唤醒时需要运行系统指令，例如 monitor, mwait, rdmsr 等，因此只有使用汇编代码才能执行这类控制。在未进行电源管理前，所有 AP 在初始化后运行 GoToSleep 处代码，只调用了 HLT 指令，相当于在 C1 状态。而 GoToCSate 代码是新加入的 C State 控制代码。其通过 monitor/mwait 指令实现 C State 控制，选择最省电的 C State(该处理器是 C6)。具体用法请参见参考文献[15]。

BSP 的 C State 控制将通过访问某一特殊的 I/O 地址实现，将其封装在 EFI\_POWER\_POLICY Protocol 中的 CpuIdle 函数内，可以被其他驱动程序调用，例如图 4-7 所示。即 BSP 在等待 I/O 设备完成工作时进入 C State，因此，为了更好的在系统级别省电，需要判断 MiniFTP 应用中所有的较长时间的 I/O 等待环节，在其等待循环中加入 CpuIdle 函数。在本研究中，在主要的 TCP 状态轮循函

数 SyncTcpPoll 和 SATA I/O 状态轮循函数中，加入了 CpuIdle。

由于 BSP 可以通过中断唤醒，但是 UEFI 环境下仅有定时器中断，而在 UEFI 下系统大多处于等待 I/O 完成或是空闲的情况，因此可以增大定时器中断时间间隔来达到目的。

```
GoToCSate::
    cli
    mov     ecx, 01Bh           ; support monitor instruction?
    rdmsr
    bt      eax, 8
    jnc     next
    jmp     GoToSleep

next::
    mov     edi, esi
    add     edi, MonitorAddress
    mov     eax, dword ptr [edi]
    mov     ecx, 0
    mov     edx, 0
    DB      0Fh, 1, 0C8h        ;monitor
    mov     eax, 20h            ; choose deepest C
    mov     ecx, 1              ; State
    DB      0Fh, 1, 0C9h        ;mwait
    cmp     dword ptr [edi], 1  ;monitor address modified
    jz      GoToSleep           ;exit C state
    jmp     GoToCSate

GoToSleep::
    cli
    hlt
    jmp     $-2
```

图 4-9 AP C state 控制汇编代码片段

Figure 4-9 AP C State control assembly code segment

### 4.3.2 定时器的电源管理

目前 UEFI 和 BIOS 系统中主要使用 8254 定时器。在英特尔南桥 I/O 芯片中，8254 时钟频率为 14.31818MHz，其最大支持 16 位位宽，作为系统时钟的 Timer0 运行在第三种模式下。因此，最大系统时钟中断间隔为 4.58 毫秒。而为了更多的降低功耗，根据 I/O 操作的特点，希望时钟中断间隔能达到 50 毫秒以上。这样，处理器可以更长时间待在更省电的 C State。所以 8254 作为系统时钟发生器不能满足该要求。

为此，本研究选择了 HPET 时钟，并且新写了 HPET 模块的代码。它可以配置为 32 位或是 64 位位宽，并且每个最小时钟周期是 69.841279 纳秒。因此精度更高，溢出时间更长<sup>[9][23][24]</sup>。图 4-10 是 HPET 中断周期设定示意代码，在代码注释中注明了如何设置的步骤。

目前许多操作系统已经支持 HPET，因此减少了本研究的工作量，不用再回到操作系统前，在 2 个不同时钟源间切换。

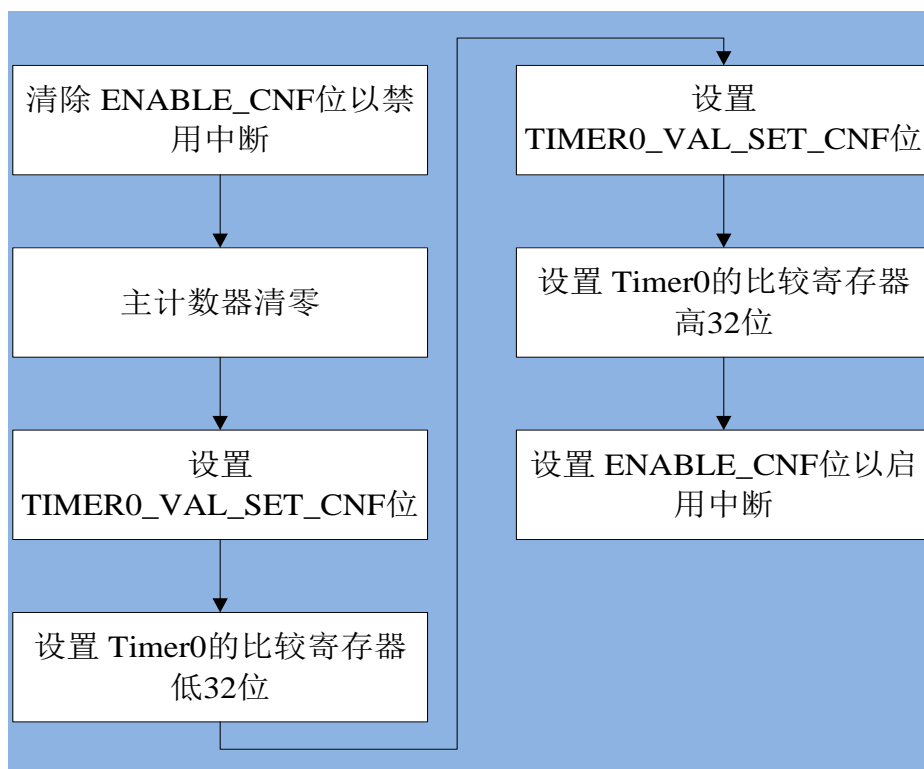


图 4-10 HPET 周期设定流程

Figure 4-10 HPET Period Setting Process

### 4.3.3 PCI 设备的电源管理

在现代的 x86 架构中，PCI<sup>[16][17]</sup>总线是系统连接的总线，扮演沟通 I/O 设备、处理器和内存的角色。虽然随着 PCI Express 总线的出现取代了 PCI 总线，但是其软件操作架构仍然和 PCI 总线兼容。

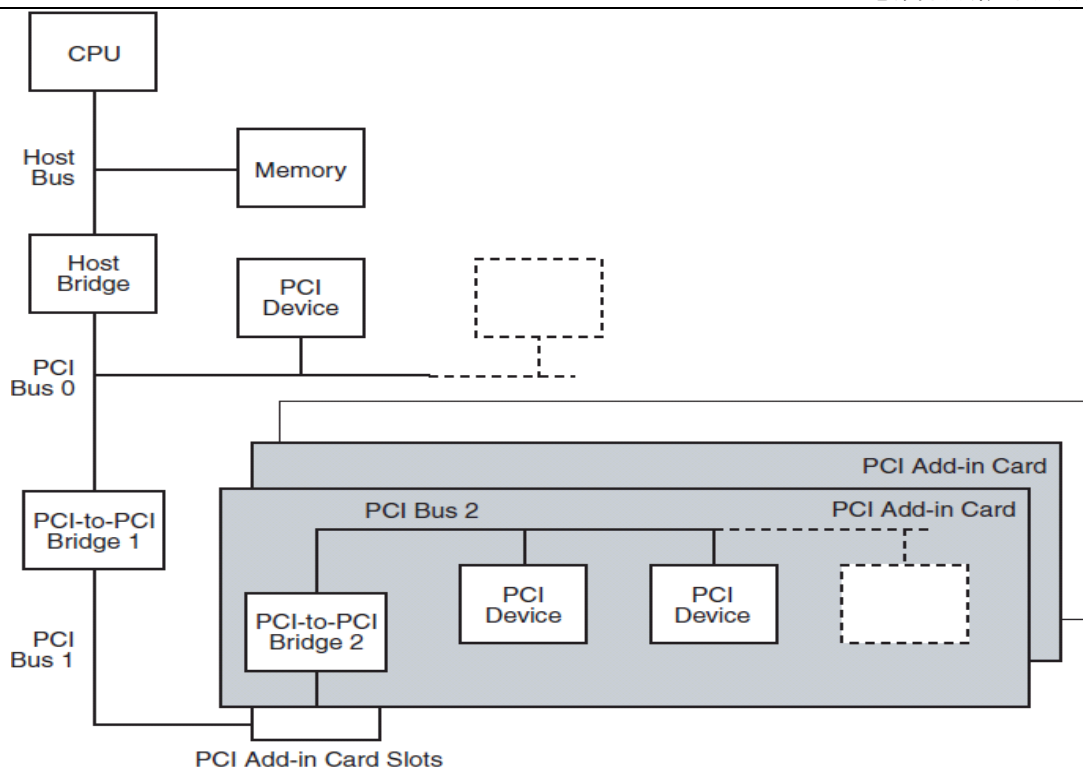


图 4-11 PCI 总线和桥架构<sup>[38]</sup>

Figure 4-11 PCI Bus and Bridge Architecture<sup>[38]</sup>

如图 4-11 所示，在系统枚举 PCI 设备时，将从 Host Bridge 开始，并默认将其作为总线 0。当检测到一个 PCI 设备并判断出其类型为 PCI 桥<sup>[38]</sup>时，按照深度优先<sup>[39][40]</sup>的算法来枚举其下面的树状结构，并为其分配总线号码和资源。在一个 PCI 主桥下面，可以有 256 个总线。在遍历结束后，所有的 PCI 桥得以分配到总线号码、I/O 资源和内存映射 I/O 资源，这样一组 PCI 总线层次被配置好。访问 PCI 设备时，将根据设备分配好的总线号码、设备号码(最大 32 个)、功能号码(最大 8 个)、以及提供的寄存器位置，可以读写 PCI 设备的配置空间。

通过访问配置空间固定位置的状态寄存器后，可以得到该 PCI 设备是否支持扩展能力。如果支持，将通过访问位于配置空间 0x34 位置(标准 PCI 设备和桥设备，CardBus 是 0x14)得到扩展能力模块的头指针。然后按照图 4-12 的方式访问，直到 Next Item 项为 0：表示链表结束。而对于每次访问到的项，将比较其 ID 是否为电源管理扩展能力标志(0x1：表示是该项包括的是 PCI 电源管理寄存器)。这样就可以通过标准的 PCI 电源管理接口，设置 PCI 设备的电源状态。



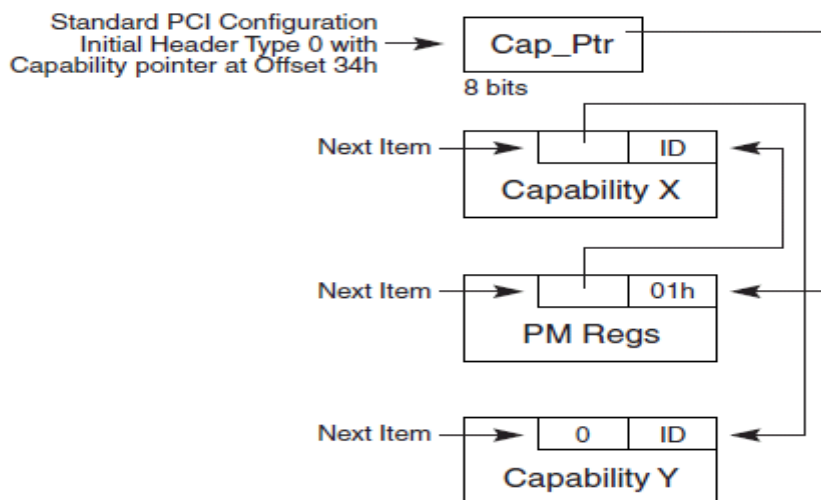


图 4-12 扩展能力链表

Figure 4-12 Capabilities Link List

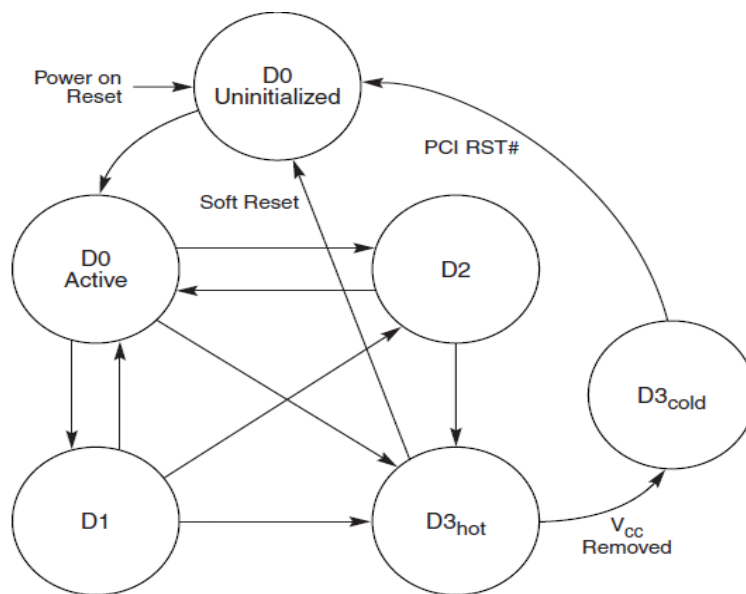


图 4-13 PCI 功能电源管理状态转换

Figure 4-13 PCI Function Power Management State Transitions

PCI 设备的电源状态称为 D State，其描述如下。其相互转换见图 4-13。

D0: 设备工作在工作模式，设备必须支持该状态；

D1: 设备的内存映射和 I/O 空间不能访问，设备处于睡眠状态，可选；

D2: 设备比 D1 更省电但恢复时需时更长，可选；

D3(hot): 设备比 D2 更省电，但恢复时需时更长。设备的状态可以被保存或不被

保存，可以由硬件设计决定，但是需要状态位说明，设备必须支持该状态；

D3(cold): 设备的主要供电路径 VCC 断电，但辅助电源 3.3V Aux 仍供电工作，设备此时仅支持唤醒功能，而且电源管理状态得以保存，设备默认支持该状态；

因此，PCI 电源管理控制可以通过访问 PCI 的配置空间，找到电源管理扩展能力区域。在设置时需要考虑树状父子关系<sup>[39][40]</sup>，先把总线上的所有设备进入到 D3 hot 状态后，总线自己才能进入到总线的 B3 状态<sup>[17]</sup>。例如，显卡设备不用时，可以放入 D3 hot 态省电。但是在 MiniFTP 使用模式下，存储硬盘和网卡不能进入到 D3 hot 状态。

在退出 UEFI S3 BIOS 环境时，需要恢复各 PCI 设备到 D0 状态。可以通过 Software Reset 将其恢复到 D0 状态。然后根据电源管理寄存器中 DSI (Device Specific Initialization) 位，判别该设备是否需要额外的初始化。如果是则表示该 PCI 设备需要重新配置。而此时 BIOS 环境已经退出，将交给操作系统管理设备。因此不用重新配置，而是通过 Function Level Reset 来使 PCI 设备恢复到默认值。操作系统中的驱动程序将负责恢复 PCI 配置空间和内存映射空间在睡眠前保存的值，或是重新初始化硬件。注意：在实验中发现，如果在退出时不做 Function Level Reset，那么有的设备在恢复到操作系统后将不能正常使用，甚至系统都不能回到操作系统而是挂起。

#### 4.3.4 PCI Express 设备的电源管理

PCI Express<sup>[18][19]</sup>总线是在 PCI 总线基础上发展起来的总线结构，目前在 x86 架构中已经普及，特别是显卡更是从 PCI Express 的高速传输中获益。PCI Express 不同于 PCI，它是串行总线，从下到上分为物理层、数据链路层和传输层。在软件层面上，PCI Express 和 PCI 设备兼容。PCI 规范中，单个设备的配置空间大小是 256 字节，而 PCI Express 扩展到 4096 个字节。对于 PCI Express 配置空间的访问，在前 256 字节区域可以通过传统的 0xCF8/0xCFC I/O 端口方式访问。它还提供了按照内存映射方式访问的方法。因此 PCI Express 独特的电源管理寄存器，可以通过后一种方式访问。PCI Express 除了支持 PCI 设备的电源管理特性外，还提供了更多的支持，主要体现在对物理层的电源管理特性上。

在表 4-2 中，列出了物理层链路电源管理的状态。表中列出开/关的项表示硬件设计既可以让该硬件处于打开、也可以处于关闭的状态，由该设备设计人员决定。L0 是工作状态，L0s 也是工作状态，但是不传输数据包。L1 是睡眠状态。

表 4-2 PCI Express 链路电源管理

Table 4-2 PCI Express Link Power Management

	软件可控	支持 ASPM	时钟	主电源	内部PLL	辅助电源
L0	是(D0)	是	开	开	开	开/关
L0s	否	是	开	开	开	开/关
L1	是(D1 – D3 hot)	是	开/关	开	开/关	开/关
L2	是	否	关	关	关	开
L3	否	否	关	关	关	关

L0, L0s 和 L1 支持 ASPM (Active State Power Management)，即硬件可以根据目前工作负载，来自行决定将链路放入何种状态。L2 是主电源断电，但是副主电源工作的状态，主要支持唤醒。L3 是设备断电时状态。

根据 PCI Express 特点，本电路板的硬件特性以及 MiniFTP 的应用特性，研究中将不使用的 PCI Express 链路层放到 L2 状态，以节省更多的功耗。而将使用的 PCI Express 设备，例如 DMI 总线（用于处理器和 I/O 芯片通信的总线，物理层借鉴于 PCI Express），研究中开启了 ASPM 电源管理以节省 PCI Express 设备功耗。这样在 PCI Express 物理层处于空闲状态时，硬件会自行管理进入到更低功耗状态。而当侦测到需要做数据传输时，恢复到 L0 状态。

ASPM 和链路层关闭的操作比较简单，按照类似于 PCI 设备找电源管理扩展能力的方法，找到 PCI Express 扩展能力 ID(0x10)，然后再在其偏移量 0x10 的位置找到控制寄存器 Link Control Register 便能控制。BIT0:1 是 ASPM 管理，BIT3 是 Link Disable。

#### 4.3.5 ATA 设备的电源管理

ATA 设备<sup>[22]</sup>在不用硬盘时，通过发送 0xE2/0xE0 等命令，使其进入省电模式，操作系统中空闲状态关闭硬盘便是此类操作。该操作比较简单，通过标准 IDE 寄存器接口向硬盘发送命令。

但是在发送命令后，由于硬盘机械式操作的特点，硬盘内部停止旋转。所

以, 对于后续硬盘的访问有明显的延迟, 在研究中根据命令的不同延迟在数秒到十几秒间。所以, 在 MiniFTP 的应用中, 可以通过分配更大的内存空间, 来存储网络下载的数据, 这样可以增加硬盘设备的休眠时间。从而节省功耗。

#### 4.3.6 USB 设备的电源管理

将 USB<sup>[20]</sup>各端口、包括 Hub 端口在内, 设置其为 suspend 模式, 达到省电目的<sup>[21]</sup>。同样, 需要考虑树状结构关系。在本论文的实验和实现中, 考虑到 USB 外设的多样性, 并且 USB 设备不处在项目规划的关键路径中。因此, 在本阶段的后续实验中没有插入 USB 外设, 不做其端口功耗的测量。

#### 4.3.7 内存的电源管理

SDRAM 类型的内存可以通过进入自刷新状态达到省电的目的。但是处于该状态下, 系统将不能进行内存单元的读写, 而只能保持数据。这也是内存单元在系统处于 ACPI S3 状态时所对应的状态。

由于内存在正常的使用中一直处于访问状态, 因此, 不能将其放入到更省电的自刷新状态。由于该平台提供的内存电源管理能力有限, 因此这里在软件层面上电源管理不作处理。幸运的是, 当前的内存控制单元集成在处理器内部, 当处理器处在省电的 C State 时, 处理器硬件内部的电源管理控制器将通过空闲时间计数器, 自动控制内存在工作状态和自刷新状态。这个过程不需要 x86 架构的软件干预。

注: 但是目前服务器对内存的电源管理提供更多的硬件支持控制策略, 可以参考英特尔公司基于 Sandybridge 架构的服务器内存管理相关文档。

#### 4.3.8 主板时钟发生器和锁相环的电源管理

由于在当前计算机主板上, 各设备的运行频率不同, 因此由主板时钟发生器统一提供主板上各设备不同频率的时钟。但是在该 MiniFTP 使用模式下, 一些设备没有使用。因此, 和这类未使用设备相关的时钟发生器和相关的锁相环电路(PLL)可以被关闭<sup>[10][25]</sup>。当恢复到操作系统前时, 才重新打开。而该操作通过 SMBus 总线发送命令操作。

由于某些设备之间的时钟或是 PLL 共用, 所以只要有一个设备在使用, 就不能关闭。因此, 还需要启用下节提到的动态 Clock Gating 功能。

其管理流程如图 4-14 所示。在关闭未用时钟和 PLL 框内，根据系统的电路图和 CK505 文档，Capella 电路中使用到 CK505 提供的如下 5 个时钟输出。

- 1) SRC0: PCH 96MHz, 连接 PLL2, 用作 48MHz USB, SIO 芯片和 24M HDA bit;
- 2) SRC1: PCH 100MHz SATA, 连接 PLL3, SATA;
- 3) SRC2: PCH 100MHz DMI, 共享连接 PLL3;
- 4) CPU0: 133MHz Ref, 不能停止否则处理器不工作, 控制位是 Buffer[4].BIT2
- 5) REF: PCH 14.318MHz REFCLK14IN. 不能停止否则系统时钟不能工作, 控制位是 Buffer[2].BIT7。

由于 PLL4 未使用, 25MHz 未使用, 以及 PCH 96MHz 未使用。因此, 可以通过 SMBUS 总线访问暂时关闭。在退出电源管理模式时, 恢复保存的原始值。

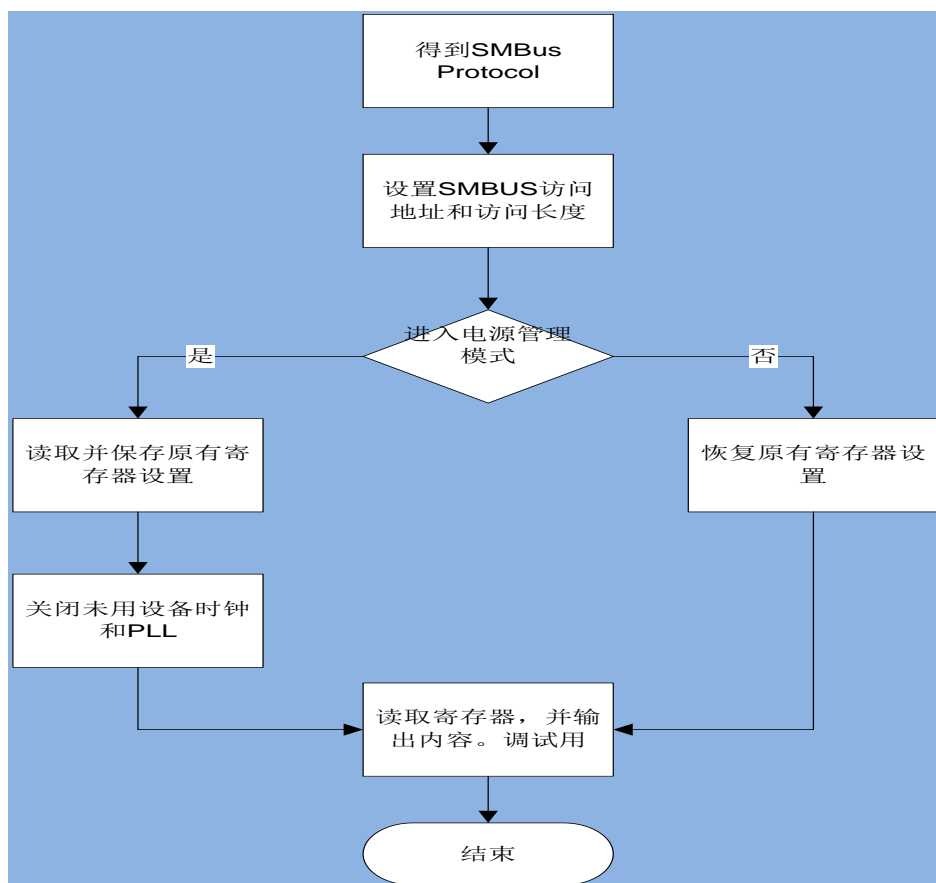


图 4-14 时钟和 PLL 电源管理

Figure 4-14 Clock and PLL Power Management

#### 4.3.9 系统 IC 内部时钟的电源管理

Platform Controller Hub 内部有多种 I/O 设备，而这些 I/O 设备大多数时间处于空闲状态。可以启用这些部件的动态 Clock Gating<sup>[5][25]</sup>功能。其作用是通过硬件来保证，在部件存在一段时间不使用时，硬件可以自动切断其时钟，从而达到省电的目的。

通过访问 PCH 内部 RCBA 内存 I/O 映射空间，可以访问 Clock Gating 寄存器。

#### 4.4 本章小结

本章主要利用流程图和逻辑说明的方式，详细的介绍了本文中各个技术细节的实现。在电源管理小节中，涉及到计算机系统许多部件的电源管理工业标准，限于篇幅只对其中最关键的步骤进行了说明。

下一章将介绍如何测量系统的整体功耗和部件功耗分布，以及对比实现本文技术前后的系统功耗。

## 5 系统测量和数据分析

本章介绍了论文采用的不同测量工具、测量方法和测量平台。并说明了如何利用测量数据来获得系统各部件的功耗分布情况。本章还在同一使用模式下，对比了采用本文技术前后，在各个主要操作系统下的系统整体功耗。可以看出采用本文技术后，可以得到更低的整体功耗。

### 5.1 测试平台和测量工具介绍

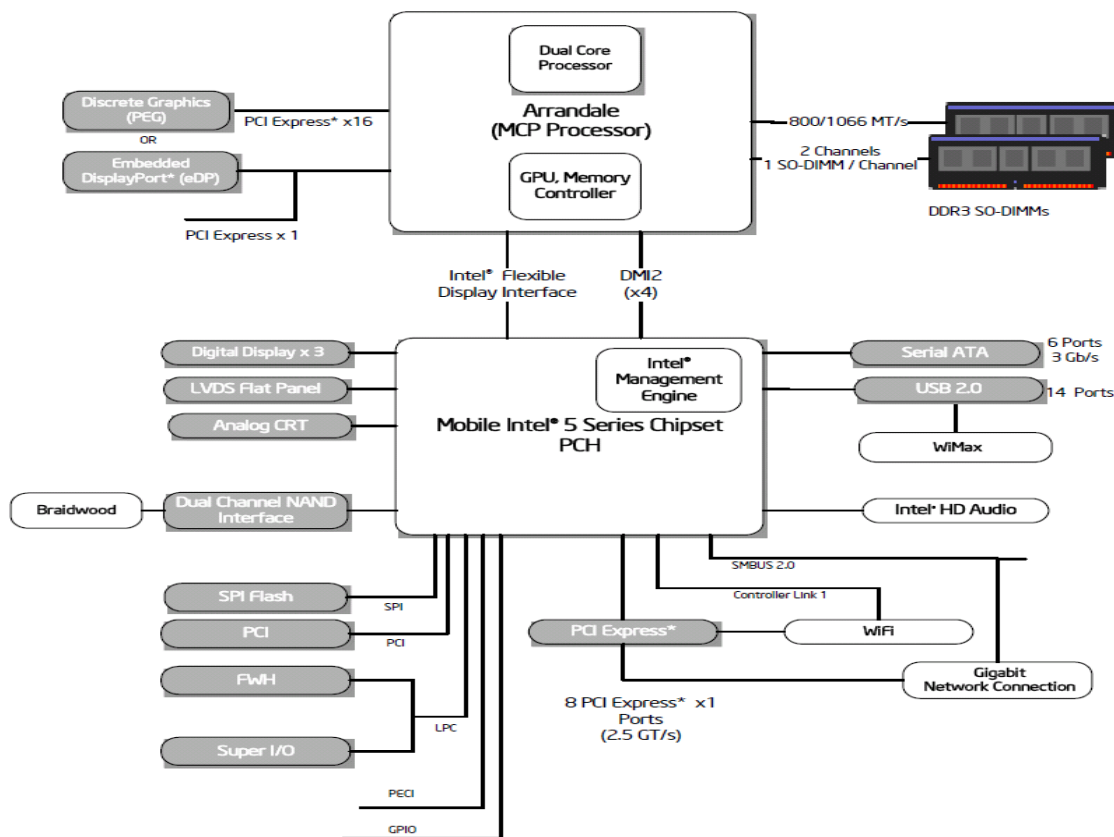


图 5-1 测试平台概况

Figure 5-1 Test Platform Overview

本课题采用英特尔笔记本 Calpella 客户参考开发平台。其电路板主要集成电路部件为 Arrandale (Core i3) 处理器、支持集成显卡，5 系列 I/O 控制芯片 (Ibex Peak)，82577 千兆以太网物理层控制器，多个直流电压转换器，Super I/O 芯



片, CK505 时钟发生器, SPI Flash 等<sup>[9][10][11]</sup>。而外插设备有内存条, SATA 硬盘等。该电路板支持基于 JTAG 的 ITP XDP3 工具调试 UEFI BIOS 代码, 并且支持串口控制台作为输入输出。如图 5-1 所示。

由于该电路板在主要部件的供电路径串联了精密电阻, 方便测量各主要集成部件耗电量。但是, 由于现在集成电路内部集成了多种复杂功能和支持多种类型输入输出电压, 常常包含多路供电路径, 因此测量点多。而且由于有些测量点在电路板背面, 为了方便测量, 需要对测量点焊接导线方便测量, 通过电路图可以得知测量点约 70 多个, 需要焊接上百根导线。焊接好后的电路板如图 3-3。

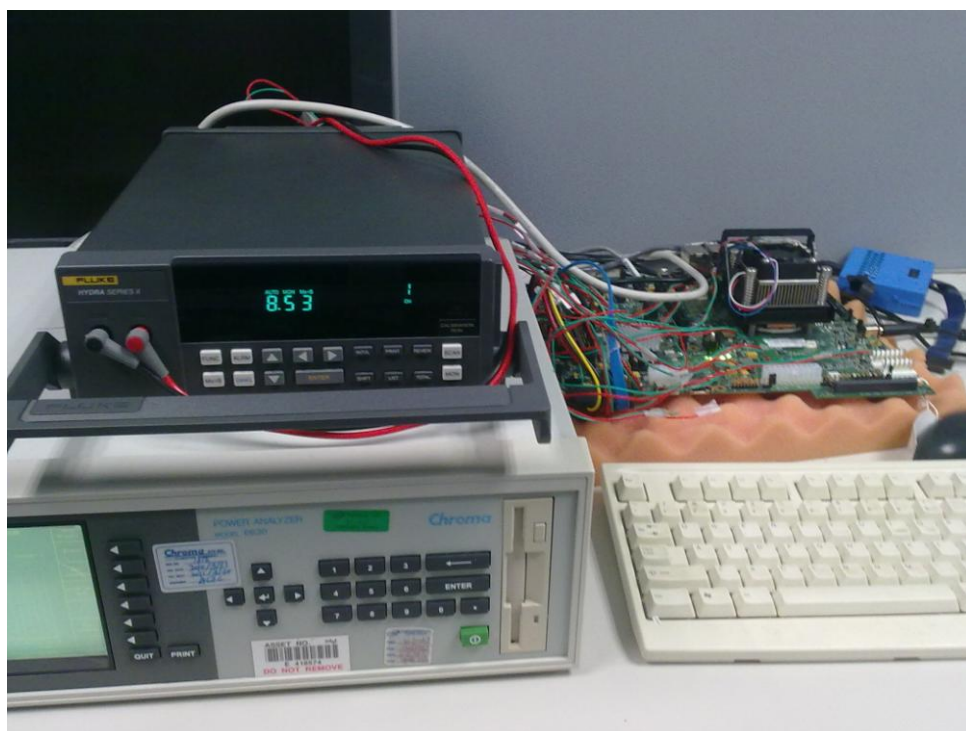


图 5-2 测试平台和测试设备 比例 85%

Figure 5-2 Test Platform and Measurement Instruments Ratio 85%

图 5-2 是本研究所用的测试设备和调试工具。图左上黑色的仪器是 Fluke 公司的 HYDRA Series 2 数据采集器。它主要用来测试电路板上精密电阻两端的电压差。它提供一共 20 路输入, 可以同时测量 20 路电压。其直流电压测量精度可以达到  $1\mu\text{V}$ , 即  $0.001\text{mV}$ 。一方面它可以满足精密小电阻的测量要求, 另一方面它可以同时测量 20 路数据, 减少了线缆测试安装的工作量和测试时间。该仪器主要用来间接测试直流功率。而下面的白色仪器是 Chroma 公司的 6630 电源分析仪。它主要用来测量交流功耗, 即包括笔记本适配器在内的系统输入总功耗。它的优点是可以记录长达 2 个小时的平均交流功耗, 可以直接在屏幕上显示。而图右远端的蓝色盒子是英特尔公司的 ITP XDP3 调试工具, 它基于 JTAG 接口,

提供源代码级别的 BIOS 调试，处理器内部寄存器设置，I/O 设置，物理内存访问等功能。

下面将介绍系统主要 IC 的电源信号情况，一是处理器；二是 I/O 芯片，即传统意义上的南桥芯片。

表 5-1 是处理器（Arrandale 工程样品，对应产品即移动 Core i3 处理器）的电源信号，这些电源信号提供了处理器工作所需要的功率。这些电源信号的电压值大小不同，并且使用目的不同。例如，为了提高内部信号的抗干扰能力，将模拟供电和数字供电路径分开，所以需要单独提供 PLL 供电，以减小 PLL 工作时对处理器核心的干扰。而表中各 VTT 电压提供了对 I/O 门电路的驱动能力，用作上拉电压的功能。VCC 电压值是浮动的，其具体值由 VID 信号控制，请参见 4.3.1。

表 5-1 处理器电源信号说明

Table 5-1 Processor Power Signal Description

信号名称	描述
VCC(Processor Core)	处理器核心电压，提供处理器内核的供电。其电压值由信号 VID[] 动态控制，范围在 0-1.4V
VAXG (Graphics core)	提供集成图像控制器核心电压，最大值 1.45 伏
VCCPLL(Power rail for filters and PLLs)	提供锁相环电压，典型电压值在 1.8 伏
VDDQ(DDR3 power rail)	提供内存控制器工作电压，典型电压值在 1.5 伏
V1.1S_VCCTT	提供处理器的 I/O 引脚电压，典型电压值在 1.05 伏
V1.1S_VCCTTA_QPI	提供处理器的 QPI 总线引脚电压，典型电压值在 1.05 伏
V1.1S_VCC_FDI	提供处理器和 I/O 芯片间的图像数据通道(Flexible Display Interface)电压，典型电压值在 1.05 伏
V1.1S_VCC_PEG_DMI	提供处理器和 I/O 芯片间的 DMI(Direct Media Interface)数据通道电压，典型电压值在 1.05 伏
V1.1S_VCCTTA_DDR	提供处理器内部内存控制器 I/O 引脚电压，典型电压值在 1.05 伏
V1.1S_VCC_SA	提供处理器内部系统部件(System Agent)电压，即 PCI Express，中断控制器等，典型电压值在 1.05 伏

表 5-2 提供了 Platform Controller Hub(PCH)的各个供电路径，标为 1.1 伏的电源信号，可以在 1.05 和 1.1 伏之间在生产时选定。对于笔记本平台来说，其选定为 1.05 伏。而所有的 PLL 电路的电源都单独隔离出来，以减少对数字电路的干扰。

值得注意的是，由于涉及到芯片内部供电电路，属于商业机密，因此无法得

到详细的资料描述，只能得到简介，所以以下描述并不十分精确。但是这并不影响对该芯片功耗的计算。

表 5-2 PCH 电源信号说明

Table 5-2 PCH Power Signal Description

信号名称	描述
V1.1S_PCH_VCC	PCH 核心电压供电，1.05 伏
V1.1S_PCH_VCCDPLL_EXP	DMI PLL 供电，1.05 伏
V1.1S_VCC_EXP	PCI Bridge 和 DMI 供电，1.05 伏
V3.3S_VCCA3GBG	内部总线 3GBG 模拟电路供电，3.3 伏
VCCAFDI_VRM	内部 FDI 总线模拟电路供电，1.8 伏
V1.1S_VCCDPLL_FDI	FDI 数字总线供电，1.05 伏
V3.3S_CRT_VCCA_DAC	内部模拟显示控制器电路供电，3.3 伏
V3.3S_VCCA_LVD	内部 LVDS 模拟电路供电，3.3 伏
V1.8S_VCCTX_LVD	LVDS 数字传输电路供电，1.8 伏
V3.3S_VCC_GIO	内部高速 GIO 数字电路供电，3.3 伏
V1.5S_1.8S_VCCADMI_VRM	DMI PLL 模拟电路供电，1.8 伏
V1.1S_VCC_DMI	DMI 数字供电，1.05 伏
V_NVRAM_VCCPNAND	NVRAM 和 NAND 电路供电，1.8 伏
V3.3M_VCCPEP	无描述，3.3 伏
V1.1M_VCCAUX	辅助电路供电，1.05 伏
V1.1M_VCCPEPW	和 ME 有关供电，1.05 伏
VCCA_DPLL_L	显示适配器 PLL 模拟供电电路，1.05 伏
V1.1S_SSCVCC	无描述，1.05 伏
V3.3A_VCCPSUS	内部睡眠状态电路供电，3.3 伏
V3.3S_VCCPCORE	GPIO 核心电路供电，3.3 伏
V1.1S_VTT	外部 I/O 驱动电路供电，1.05 伏
V1.1S_VCCUSBCORE	USB 总线核心电路供电，1.1 伏
V3.3A_VCCPUSB	USB 总线核心电路供电，3.3 伏
V3.3S_VCCPPCI	PCI 总线核心电路供电，3.3 伏
V1.1S_VCC_SATA	SATA 核心电路供电，1.05 伏
VCCPLLVRM	SATA PLL 供电，1.8 伏
V3.3A_1.5A_VCCPAZSUS	声卡解码器模拟电路供电，3.3 伏

## 5.2 测量的目的，原理和方法

本文所涉及到的实验测量有两个主要目的。一是通过测量得出电路板上主要功率分布情况，从而选出功耗较高的设备，通过软件来做电源管理或则提出对硬件的改进意见。二是对比本方案和和商业操作系统下的电路板整体功耗，希望通过本方案，达到整体功耗低于操作系统环境下的结果。

由于电路板上都是直流供电，测量原理如下公式。

$$\text{功耗} = \text{电压} \times \text{电流} \quad (5-1)$$

$$\text{电流} = \text{精密电阻电压差} / \text{精密电阻电压阻值} \quad (5-2)$$

$$\text{功耗} = (\text{精密电阻电压差} / \text{精密电阻电压阻值}) \times \text{测量点电压} \quad (5-3)$$

而每个集成电路芯片的功耗等于所有电源信号路径上的功耗和。这样可以通过测量每一路供电路径上的功耗，算出系统主要 IC 部件的功耗。除此之外，主板上存在许多 DC-DC 电压转换器，可以通过如上方法测出输入和输出功耗。那么，电压转换器的功耗等于输入功耗减去输出功耗。而对于 AC-DC 电压转换器，需要通过专有仪器测量 AC 功耗、以及输入到电路板的直流功耗，得出笔记本电源适配器的损耗功耗。通过以上测量，可以得到电路板主要集成电路电路功耗和电压转换器功耗的分布数据。根据这些数据，便可以指导如何运用软件控制主要耗电部件，来达到省电的目的。而且还可以通过分析电压转换器的转换效率，指导硬件器件的选择和设计。

### 5.2.1 系统主要功率分布和不同使用模式下整体功耗比较

为了算出个部件的功率，特别是针对处理器和 I/O 控制器这样复杂的 IC，需要测量多路供电路径。由于测试点多，因此表格较长。并且实验测试了三种状态下的系统功耗分布，将 3 个表格放在附录 A 中，供有兴趣的读者参考。正文中只对系统各部件的结果进行分析和对比。

表 5-3 列出了具体的测试数据。它测试并对比了三种状态下，系统的功耗分布情况。分别是空闲状态，电源管理空闲状态，电源管理 FTP 下载状态。

#### 1) 空闲状态

系统未作任何电源管理优化处理，运行在普通的 UEFI Shell 环境处于空闲状态下所做的测量。在该状态下，处理器散热风扇正常运行。但是由于风扇供电电路不含精密电阻，无法测量该部件的单独功耗。风扇数据的获得是通过热拔插风扇，测量直流输入功耗前后所得差值获得。

本状态将作为基准状态，和其他状态进行对比。

表 5-3 系统电源分布-单位瓦特

Table 5-3 System Power Distribution-Unit Watt

部件名称	空闲状态	电源管理空闲状态	电源管理 FTP 下载
AC 输入功率	21.100	9.700	10.300
DC 输入功率	18.047	7.970	8.347
交流电源适配器	3.053	1.730	1.953
处理器风扇	0.800	0.000	0.000
其他部件	1.277	1.327	1.227
Processor	8.999	2.752	2.881
DDR3 DIMM	0.159	0.068	0.068
SATA	1.060	1.036	1.036
LAN 82577	0.292	0.201	0.319
Super I/O	0.033	0.033	0.033
PCH	1.893	0.992	1.093
CK505	0.244	0.206	0.223
EC	0.033	0.033	0.033
PS/2 port	0.074	0.074	0.074
IMVP6.5 VR	0.206	0.060	0.038
DDR3 VR	0.123	0.049	0.041
System VR	0.521	0.283	0.288
V1.1 VR	2.203	0.788	0.907
V1.8 VR	0.130	0.069	0.085

注1. AC 输入功率: 测量的是包括电源适配器在内的交流输入功耗, 测量值是通过 Chroma 6630 测量半小时以上, 获得平均值;

注2. DC 输入功率: 是电路板的输入 12V 直流电的功率, 不包含电源适配器。因此 AC 输入功率减去 DC 输入功率, 得到的是交流适配器自身的功率;

注3. 其他部件的功率: 是通过 DC 输入功率减去包括风扇在内的所有表中实际测量部件的功率。由于受限于测量方法, 实验时分次单独测量, 测试设备的瞬时显示波动, 和 220V 交流电压的波动以及测量设备的分辨率等因素, 所有的误差在这里累积。因此, 该数值只能做大概参考。其实际的意义表示电路板中未测量的其他所有上拉电阻、电容、电感、晶体管、和其他小部件 IC 的功率和。

注4. Super I/O、EC(Embedded Controller)、PS/2 端口: 通过测量发现其功率非常小。因此, 未实现其电源管理功能。

注5. VR 的功率: 是其输入值减去输出值获得, 是其电压转换工程中的损耗



值，不包含 VR 自己的工作消耗功率。

注6. 电源管理 FTP 下载状态中的硬盘功耗：是空闲时的功耗

注7. DC 输入功耗：是动态瞬时值，AC 功耗：是长时间平均值。

## 2) 电源管理空闲状态

系统运行在 S3 睡眠唤醒后的 Shell 空闲状态下，此刻已运行完所有本文第四章中描述的设备电源管理功能，包括处理器进入 P/C State，PCI 进入 D3hot 状态，关闭未用 clock 输出等。

和未应用电源管理的空闲状态比较，该状态节省了一半以上的功耗。处理器的功耗以及和处理器相关的电压转换器的功耗大幅下降。由于 PS/2、EC、Super I/O 没有实现电源管理功能，因此，它们的功耗值不变。

## 3) 电源管理 FTP 下载状态

处在该状态下和电源管理空闲状态相比，其网络处于活动状态。因此网络的功耗以及处理器对网络数据的处理都有所提高。值得注意的是，在表中硬盘的功耗：表示的是硬盘处于空闲状态的功耗，而不是加上硬盘读写的平均功耗。这是因为局限于 DC 功耗测量工具只能显示瞬时值，而不能做均值处理。但是 AC 的整体功耗：是包括硬盘读写功耗在内的长时间平均值。

值得注意的是 IMVP6.5 VR 和 DDR3 VR 的功耗值反而下降，这是由于电压转换器效率提高的关系、以及电压转换器的动态负载决定，请读者参考相关电压转换器文档。

图 5-3 比较了这三种状态下各部件功耗值，并且给出了直观的系统功耗分布。可以看到在这三种状态下处理器、VR、电源适配器的功耗都有明显的区别。而处理器在实现了电源管理功能后，即使在网络下载的应用中，功耗仍得到大幅下降。

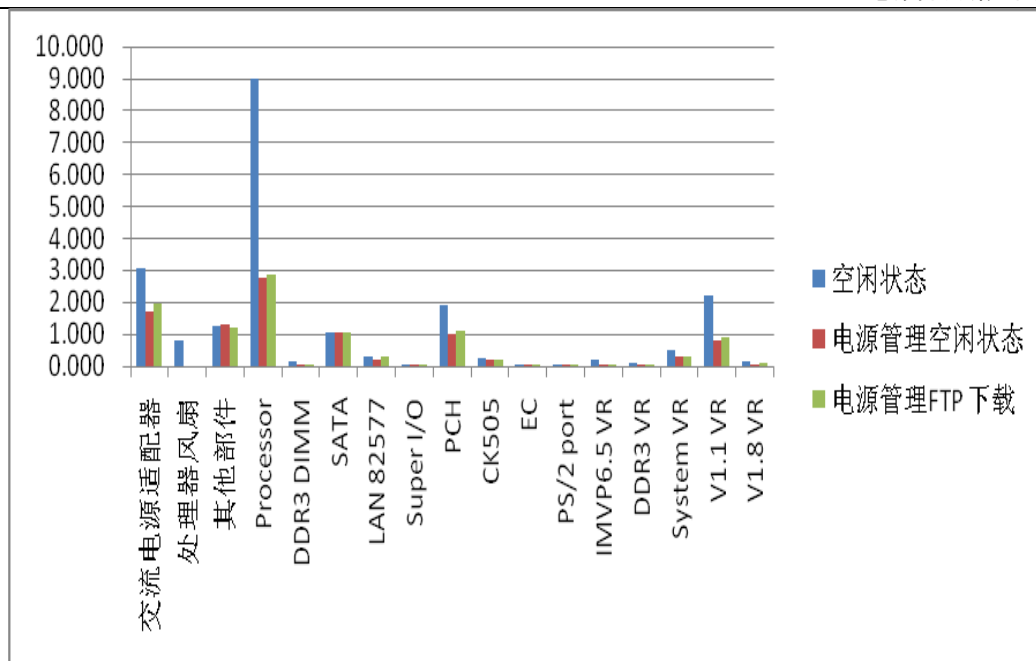


图 5-3 三种状态部件功耗比较 单位:瓦特

Figure 5-3 Component Power Consumption among the 3 States Unit: Watts

## 5.2.2 电压转换器效率分析

表 5-4 电压转换器效率比较

Table 5-4 Voltage Regulator Efficiency Comparison

VR 名称	空闲状态	电源管理空闲状态	电源管理 FTP 下载
IMVP6.5 VR	83.2%	26.9%	53.8%
DDR3 VR	86.9%	90.3%	91.8%
System VR	91.9%	93.2%	93.1%
V1.1 VR	75.9%	73.9%	72.7%
V1.8 VR	87.1%	89.9%	87.7%

表 5-4 列出了各个电压转换器的效率，即由输出功率/输入功率计算得出。表中值反映了电压转换器在直流-直流电压转换过程中的效率。值越大表示转换效果越好，无效损失越低。一般来说，实际中较好的电压转换器效率一般在 80-95%这个区间，效率越高、成本相映越高。

从表中可以看到，为处理器供电的 IMVP 电压转换器在电源管理状态下，其转换效率较低。这是由于在电路板设计时，一般把最优效率调整在常用输出功率



值附近。而在电源管理状态下由于输出功率较低，因此远离常用输出功率，使得转换效率低。电源转换器概念和设计见参考文献[41]。

虽然该效率不由软件控制，但是希望在以后的电压转换器设计中，能提供一种低功耗模式的软件控制或自适应模式，来提高其转换效率，以便更好的降低系统功耗。

### 5.2.3 网络下载模式不同操作系统与本文模式下功耗对比

表 5-5 不同系统在相同使用模式下的交流功耗比较

Table 5-5 Different System AC Power Consumption Comparison under Same Usage Model

单位：瓦特	空闲	100KB/s	200KB/s	300KB/s	400KB/s	注释
Win7	11.7	11.8	12	12	12	1 分钟后显示器关闭 *1
WinXP	12.7	13.4	13.5	13.6	13.8	1 分钟后显示器关闭 *1
Ubuntu 9.04	20.8	N/A	N/A	N/A	N/A	11 分钟后显示器关闭，无网卡驱动程序
EFI Shell	21.0	N/A	N/A	N/A	N/A	风扇运转且有显示
UEFI 电源管理模式	9.7	10	10.2	N/A	10.4	*2

注释：

1. 在 WinXP 和 Win7 测试过程进入测试状态前，都处于长时间的稳定状态，且无后台程序运行，操作系统未安装除驱动程序外的任何程序；
2. 通过服务器端口调整下载速度限制，最大平均下载速度不能达到 300KB/s。而去掉服务器端下载限制，最大下载速度可以达到 410.8KB/s，网卡被限制在 10Mbps 的速度；
3. 下载过程测试的数度是长时间的交流系统功耗平均值，下载文件大小约 4G 字节，测试时间在 1 小时以上；
4. 测试中发现同一模式下的测试数据重复测量会有细微不同：0.1 瓦，可能由交流电电压波动影响测量工具、或测试系统造成。

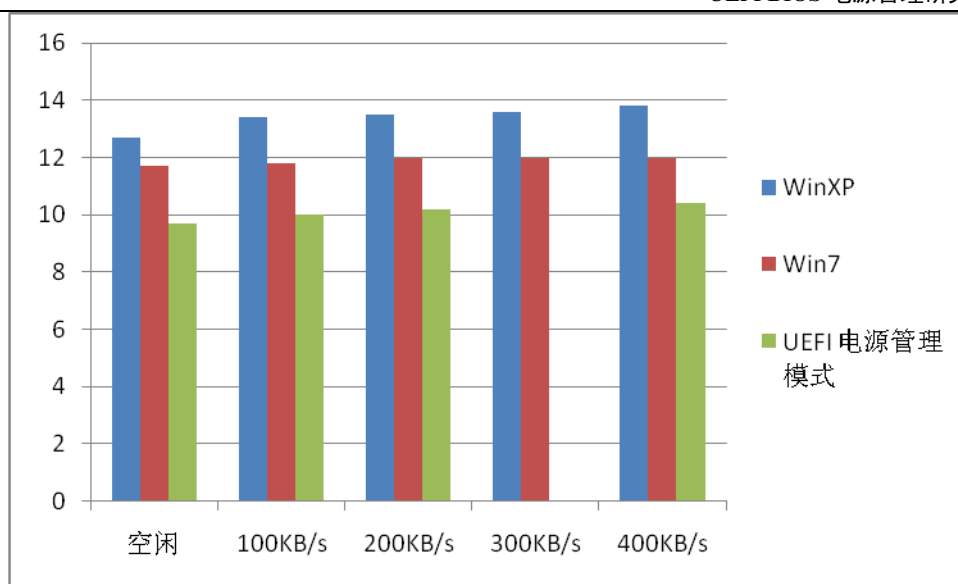


图 5-4 不同系统在相同使用模式下的交流功耗比较 单位:瓦特

Figure 5-4 Different System AC Power Consumption Comparison under Same Usage Model

在表 5-5 中, 分别列出了三种操作系统(Win 7, Win XP 和 Ubuntu 9.04)、两种 UEFI Shell 模式下(普通和电源管理模式)、不同的网络下载速度下, 系统整体交流平均功耗。

UEFI Shell 普通模式是正常启动到 UEFI Shell 下测量, 而 UEFI 电源管理模式即是通过实现本文前面提到的电源管理技术, 并且启动到 S3 DXE Shell 下的一种模式环境。

在该环境下, 通过模拟中国普通家庭的网络带宽, 即一般为 1M/2M/4M bps, 相应的下载速率上限为 100/200/400 KB/S, 测量不同速率下的 FTP 网络下载。而希望在该应用模式下, 获得优于操作系统下的整体电源功耗。

从表 5-5 和图 5-4 中可以看到, 通过实现本文的电源管理模式, 可以达到优于操作系统下的电源功耗。相对于最省电的普通操作系统 Win7, 本模式可以节省大约 15-18%, 而相对于其他操作系统则可以节省更多。值得注意的是, 15-18%的数据是在“干净”的 Win7 下获得, 而通常用户的操作系统都有后台程序运行, 例如: 杀毒软件、更新程序等。因此, 在用户的实际应用下, 往往可以达到更佳的效果。

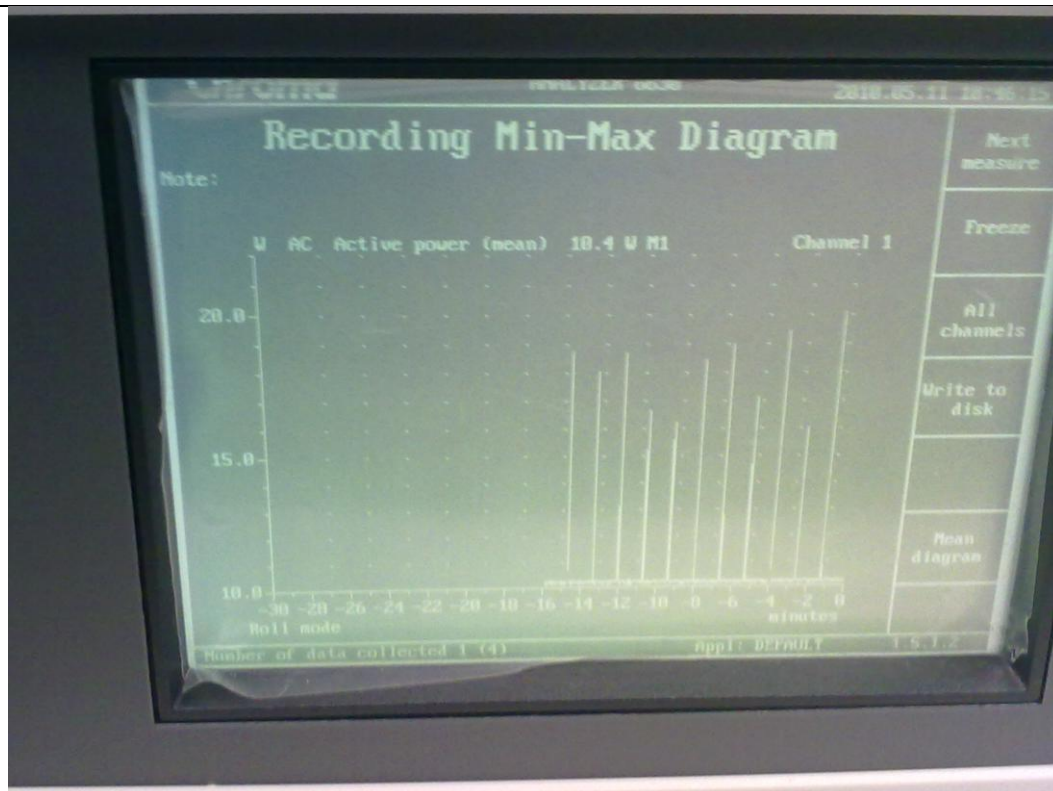


图 5-5 UEFI 电源管理模式实际测量值效果 比例 80%

Figure 5-5 UEFI Power Management Mode Real Measured Data Effect Ratio 80%

在 UEFI 电源管理模式下获得的数据是平均值,但实际的情况如图 5-5 所示。在图 5-5 中,每隔一段时间有一个脉冲,表示较高的系统功耗,它是由于在硬盘读写时形成。而在网络下载模式应用中,下载得到的数据暂时保存在内存缓存中。当下载的数据累计到一定数据时,例如:本文实现 32M 字节的缓存,FTP 应用程序将所有的数据通过文件系统的服务写入到硬盘。此时会导致大量的 I/O 操作,使处理器处于忙碌的工作状态,同时磁盘处于读写状态。因此,系统处于高功耗状态,瞬时值达到 20W 以上,但该过程持续时间较短。

因此,途中看到的高矮不一的脉冲,实际上的值是相近的,只是由于其持续时间短,不能被该测量设备捕捉。但是在写操作完成后,FTP 应用程序在处理器等待网络数据时,将其大部分时间放入到睡眠状态,而磁盘主动放入电源管理状态。因此,该状态下系统的功耗低于 10W。通过长时间的测量,可以得到系统平均功耗,该计算由测量设备完成。

如图 5-5 中的平均值表示系统 15 分钟左右的平均功耗是 10.4W。

### 5.3 本章小结

本章分析了具体的测试方法和数据，并给出了整理后各部件和整体功耗的数据，其中详细的数据和测量点请读者参见附录。本章的数据是本文技术实现的整体反映，对下一步研究和工作重点有指导意义。其中在系统整体功耗分布中，读者可以看到，系统中各电压转换器损失的功耗在低功耗时所占比重较大，这样无论系统部件的各硬件和软件如何降低功耗，在系统空闲时都无法显著降低系统整体功耗。这需要主板设计人员综合考虑转换效率和成本，设计出在低功耗模式下效率更高的电压转换器。

## 6 总 结

本论文的研究课题属于企业的研究探索性项目，其目的在于证明该方向的可行性。因此，其注重利用现有技术，在商业应用模式上有所创新。本研究利用 UEFI 开发方便快捷的特点，新颖的创建了睡眠唤醒后基于 UEFI BIOS 环境，来启动特定的应用模式。基于该模式，比较 UEFI 环境和操作系统环境下的系统功耗，达到功耗和操作系统可比较、或更低的效果。

### 6.1 工作心得体会

我在该项目中承担了所有的实际工作，并负责该项目时间管理，即在给定的时间内汇报最终数据结果。

我所做的主要工作有代码流程设计，休眠唤醒后 BIOS 功能的实现，网络下载方法的整合和实现，查阅电路图来设计测量方案和寻找测量点，动手焊接测量导线和不同情况下的数据测量等。并在项目后期转交给其他同事做下一步规划工作。由于自己对该平台软硬件方面的了解和有实际的操作系统电源管理的工作经验，并且有现有的平台启动 BIOS 代码，在软件设计方面能独立工作，克服困难。其中最有挑战性的工作是测量点的焊接和测量工作，需要自己焊接上百根测量线，并且由于设备的限制，需要分组测量。这些工作需要耐心，细心和动手能力。而其中最高兴的两件事一是看到自己焊接好密密麻麻，花花绿绿测量导线的电路板能正常启动到操作系统，二是改方案到达并好于预想的结果。

其中的工作重点首先是建立一个单独 Firmware Volume 来存放和 S3 相关的模块，其次是加入网络下载的功能，然后根据测量结果寻找功耗大的部件，针对不同部件分别优化，例如处理器作 C State, P State 管理，在网络下载等待 I/O 操作完成前的循环查询中将处理器放入 C State。最后是前期和后期的测量占用许多时间。

除了技术问题外，还有一个问题是测试设备的缺乏。由于我们属于软件部门，对功耗的测量仪器仅有例如万用表这样的简单设备，这是远远不够的。在领导和其他部门的帮助下，在其他部门借用到了所需的设备。

由于时间紧迫，我利用了学校项目管理课程中学到的知识，利用 Project 2007 来估算和分配项目时间，规划关键路径。项目总结情况在下一节介绍。

## 6.2 项目总结

通过应用操作系统中共有的电源管理方法，研究成果达到了节省整体系统功耗的目的。除此之外，本研究成果还具有在操作系统中所不具备的以下优点。

- 1) 应用了基于电路板相关信息而动态关闭时钟信号（该信息在目前的操作系统中还未实现，因为依赖于电路板连线和设备选定）的方法，进一步节省了功耗；
- 2) 由于 UEFI 环境简单是单线程的特点，和操作系统相比有更少的后台任务，因此处理器使用率更低从而更省电；
- 3) 由于基于特殊的应用模式，因此提前可以知道什么设备使用，什么设备不使用，可以将不用的设备放入更低的功耗状态。

对于应用模式的选择，取决于实际实现中 BIOS 或 OEM 厂商的定位。例如：可以在操作系统中，通过应用程序展示给用户图形界面，在配置菜单中让用户禁用该功能或则选择应用模式；例如：网络 MP3 播放、网络下载、或是杀毒、数据远程备份等。而网络地址可以提前配置默认值、或可以作修改。当所有配置完成后，系统进入 S3 睡眠模式然后马上自动唤醒，然后按照用户的配置来工作。例如，开始播放 MP3。而当用户按下按键或是设置时间结束，系统退出该环境，而恢复原来在操作系统中的任务（磁盘操作会有所影响，所以需要在两个环境中同步）。

本项目虽然在技术上涉及 UEFI BIOS 软件开发、笔记本电路硬件、操作系统、和网络传输等领域，跨度大、综合度高，项目风险较高，但是本项目充分利用了软件工程管理中迭代的开发模式，把整体功能分解为一个个单一的功能，按照功



能点的方式一个个进行了完成。而对于技术难度高，并且处于非关键路径上的功能，则放在下一阶段完成，见表 6-1。

表 6-1 项目管理任务关系示意

Table 6-1 Project Management Task Relationship Demo

任务编号	任务名称	依赖关系	资源
1	UEFI S3 环境的实现	无	研究平台 BIOS 代码
2	网络协议栈和 FTP 下载程序开发	无	网络协议栈代码
3	整合代码	1, 2	
4	测量点寻找和电路板焊接	无	焊接设备，电路图，布线图
5	测量获得无电源管理各部件和系统功耗数据	3, 4	测量设备
6	数据分析并识别关键电源管理部件	5	
7	处理器电源管理	6	
8	PCI/PCIE 电源管理	6	
9	时钟/PLL 电源管理	6	
10	硬盘电源管理	6	
11	测量获得电源管理各部件和系统功耗数据	7, 8, 9, 10	测量设备
12	各部件代码修改，性能和功耗调整	11	
13	测量得到最终数据	12	测量设备

例如：USB 总线的电源管理功能，虽然进行了研究，但受限于时间和优先级排序，在代码中尚未实现，因为系统即使没有 USB 设备，仍然可以正常工作。而对于通过测量得到的实际数据分析，一些耗电量本身很小的设备，例如 Super I/O、PS/2 口等，则在现阶段忽略其电源管理功能。但是对于功耗大的设备，例如处理器，则尽可能的实现该设备全面的电源管理功能。例如，在网络应用模式



中，对协议栈中主要的等待 I/O 结束循环，试探性将处理器放入短暂的 C State 状态，通过测试设备找到关键代码位置，即在该位置将处理器放入 C State 状态不会带来明显的性能损失，并且可以降低系统功耗。表 6-1 是本项目的项目管理示意表格，反映了各个任务的依赖关系。

### 6.3 项目后续改进和展望

本系统在实现时侧重于快速的获取数据，因此某些设备的电源管理功能没有启动，还有待完成的功能如下：

- 1) Super I/O 芯片的电源管理；
- 2) Embedded Controller 芯片的电源管理；
- 3) USB 总线电源管理功能的实现，包括通过 USB 唤醒系统后的处理；
- 4) 优化网络协议栈的缓存使用，减少内存拷贝；编写操作系统应用程序，作为系统策略控制以及和用户交互；
- 5) 能够动态预测提前唤醒硬盘以减小访问延时从而提高数据读写性能，使得硬盘可以进入更深的电源管理状态；
- 6) 建立设备电源管理的依赖关系；
- 7) 建议硬件电路板提供效率更高的电压转换器，且电压转换器具有低功耗状态控制信号，以切换其工作模式提高转换效率；

其中建立设备电源管理的依赖关系，是指在软件层建立树状数据结构，使得父节点在进入睡眠电源管理前，满足所有子节点已经睡眠；而在唤醒时，满足在唤醒子节点前保证父节点已经唤醒。

除了功能外，还可以用面向对象的思想来设计电源管理模块，使得配置更加灵活，更有利于平台移植。可以开发出一个统一的软件接口和模型，以方便工业界的互操作应用。而当前在嵌入式电源管理领域，已经有人开展面向对象技术的研究<sup>[42]</sup>。

本项目基于日渐普及的 UEFI 技术，以独特的应用模式作为切入点，融合了平台系统软件和硬件技术，构建了一个开放的平台，鼓励其他厂商开发创新应用。

---

相信在该技术逐渐成熟后，通过商业推广，可以得到日益广泛的应用。

## 参考文献

- [1] Jose Allarey, Varghese George, Sanjeev Jahagirdar, Power-Management Enhancements in the 45 nm Intel Core Microarchitecture[J], Intel Technology Journal, 2008, Volume 12, Issue 3, 1-12
- [2] 赵霞, 陈向群, 郭耀等, 操作系统电源管理研究进展[J], 计算机研究与发展 ISSN 1000—1239 / CN 11-1777, 2008, 2-5.
- [3] Luca Benini, Alessandro Bogliolo, Giovanni De Micheli, A Survey of Design Techniques for System-Level Dynamic Power Management[J], IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, 2000, VOL 8, NO. 3, 11-18.
- [4] Mobile Battery Life Solutions for Windows 7: A Guide for Mobile Platform Professionals[R], Microsoft, 2009, 13-38.
- [5] Luca Benini, Giovanni De Micheli, Dynamic Power Management of Electronic Circuits and Systems[C], Stanford University, 2008, 2-4.
- [6] Advanced Configuration and Power Interface Specification 3.0b[S], <http://www.acpi.info>, 2006, .
- [7] Jerzy Kolinski, Ram Chary, Andrew Henroid, etc, Building the Power-Efficient PC: A Developer's Guide to ACPI Power Management[M], Intel Press, 2001, 19-32.
- [8] Unified Extensible Firmware Interface Specification Version 2.2[S], <http://www.uefi.org>, 2008.
- [9] Intel 5 Series Chipset and Intel 3400 Series Chipset External Design Specification (EDS) Revision 2.0[S], <http://www.intel.com>, 2009.
- [10] CK505 Clock Synthesizer Specification Revision 1.0[S], Intel, 2006.
- [11] Intel MVP 6.5 Mobile Processor and Mobile Chipset Voltage Regulation Specification Revision 1.25[S], Intel, 2005.
- [12] Behrouz A. Forouzan, Sophia Chung Fegan, TCP/IP Protocol Suite Second Edition[M], 清华大学出版社, 2004.

- [13] RFC959 File Transfer Protocol[S], <http://www.w3.org/Protocols/rfc959/>, 1985.
- [14] Andrew S. Tanenbaum, Computer Networks, Fourth Edition[M], Prentice Hall, 2003.
- [15] Intel 64 and IA-32 Architectures Software Developer's Manual Volume1-3[M], <http://www.intel.com>, 2009.
- [16] PCI Local Bus Specification Revision 3.0[S], <http://www.pcisig.com>, 2002.
- [17] PCI Bus Power Management Interface Specification Revision 1.2[S], <http://www.pcisig.com>, 2004.
- [18] PCI Express Base Specification Revision 2.0[S], <http://www.pcisig.com>, 2006.
- [19] Adam Wilen, Justin Schade, Ron Thornburg, Introduction to PCI Express[M], Intel Press, 2003.
- [20] Universal Serial Bus Specification Revision 2.0[S], <http://www.usb.org>, 2000.
- [21] Power Management of USB Host Controllers[G], Microsoft, August 30, 2004, 4-10.
- [22] Information Technology -AT Attachment with Packet Interface – 6 (ATA/ATAPI-6)[S], <http://www.t13.org>, 2002.
- [23] IA-PC HPET (High Precision Event Timers) Specification Revision 1.0a[S], <http://www.intel.com>, 2004.
- [24] Guidelines for Providing Multimedia Timer Support[R], Microsoft, 2002.
- [25] Donald A. Neame, Microelectronics Circuit Analysis and Design Third Edition[M], 清华大学出版社, 2007.
- [26] Lubomir F. Bic, Alan C. Shaw, Operating System Principles[M], 清华大学出版社, 2004.
- [27] Vincent Zimmer, Michael Rothman, Robert Hale, Beyond BIOS[M], Intel Press, 2006.
- [28] VOLUME 1 - 5: Platform Initialization Specification Pre-EFI Initialization Core Interface Version 1.2[S], <http://www.uefi.org>, 2009.
- [29] John L. Hennessy, David A. Patterson, Computer Architecture A Quantitative Approach, Fourth Edition[M], Morgan Kaufmann publications, 2006.
- [30] Yung-Hsiang Lu, Tajana hnwziC, Giovanni De Micheli, Software Controlled

- Power Management[J], IEEE, 1999, 1-3.
- [31] Dinesh Ramanathan, Sandy Irani, Rajesh Gupta, Latency Effects of System Level Power Management Algorithms[J], IEEE, 2000.
- [32] Walter Oney, Programming the Microsoft Windows Driver Model 2nd edition[M], Microsoft, 2003.
- [33] UEFI Shell 源代码网址[K],  
<http://sourceforge.net/apps/mediawiki/tianocore/index.php?title=Efi-shell>.
- [34] Michael Rothman, Tim Lewis, Harnessing the UEFI Shell[M], Intel Press, 2010.
- [35] UEFI Shell Specification 2.0[S], <http://www.uefi.org>, 2008.
- [36] Ethernet Frame Format[K], [http://en.wikipedia.org/wiki/Ethernet\\_II\\_framing](http://en.wikipedia.org/wiki/Ethernet_II_framing).
- [37] MASM Programmer's Guide version 6.1[G], Microsoft, 1996.
- [38] PCI-to-PCI Bridge Architecture Specification Revision 1.2[S],  
<http://www.pcisig.com>, 2003.
- [39] Jon Kleinberg, Eva Tardos, Algorithm Design[M], 清华大学出版社, 2006.
- [40] 严蔚敏, 吴伟民, 数据结构 (C 语言版) [M], 清华大学出版社, 1997.
- [41] Switch Mode Power Supply Reference Manual Rev3B[G], On Semiconductor, 2007.
- [42] Ankur Agarwal, Saeed Rajput, A. S. Pandya, Power Management System for Embedded RTOS: An Object Oriented Approach[J], IEEE, 2006.

## 附 录

### 电路板三种模式下功耗分布详细数据

表 1, 表 2 和表 3 提供了各部件详细的测量方法和数据报告。本文第五章中数值来源于下列表项中。而由于测量值较多, 计算复杂, 所以将这 3 张表格放在附录 A 中作为参考。

表 1 系统电源分布-EFI Shell 空闲状态

Table 1 System Power Distribution-EFI Shell Idle State

部件名称	电阻值 (mΩ)	电阻电压降 (mV)	测量电 压(V)	功耗(W)	注释
AC 输入功率				21.100	*1
DC 输入功率	7	10.05	12.57	18.047	*2
交流电源适配器				3.053	*3
处理器风扇				0.800	*4
其他部件				1.277	*5
Processor				8.999	*6
VCC_CORE	2	1.32	0.8364	0.552	
VCC_CORE	2	1.12	0.8362	0.468	
VGFX_CORE	2	0.23	0.9773	0.112	
V1.8S_VCCSFR	2	0.71	1.7884	0.635	
V1.5_VCCDDQ	2	0.77	1.5114	0.582	
V1.1S_VCCTT	2	4.72	1.0592	2.500	
V1.1S_VCCTTA_QPI	2	3.95	1.0517	2.077	
V1.1S_VCC_FDI	2	0.25	1.0593	0.132	
V1.1S_VCC_PEG_DMI	2	0.62	1.0521	0.326	
V1.1S_VCCTTA_DDR	2	1.16	1.0593	0.614	
V1.1S_VCC_SA	2	1.9	1.0521	0.999	
DDR3 DIMM				0.159	*6

续表 1

VDD	2	0.21	1.5122	0.159	
VDDSPD	22	0	3.3027	0.000	
SATA				1.060	*6
V3.3S	2	0	3.3037	0.000	
V5S	2	0.43	4.928	1.060	
LAN 82577				0.292	*6
V3.3M	22	1.22	3.3032	0.183	
V1.1_LAN_M	22	2.32	1.0321	0.109	
Super I/O				0.033	*6
V3.3S	2	0.02	3.3029	0.033	
PCH				1.893	*6
V1.1S_PCH_VCC	2	0.63	1.0442	0.329	
V1.1S_PCH_VCCDPLL_EXP	22	0.69	1.0447	0.033	
V1.1S_VCC_EXP	2	0.86	1.0442	0.449	
V3.3S_VCCA3GBG	22	0	3.3034	0.000	
VCCAFDI_VRM	22	0.86	1.764	0.069	
V1.1S_VCCDPLL_FDI	22	0.51	1.0448	0.024	
V3.3S_CRT_VCCA_DAC	22	1.38	3.309	0.208	
V3.3S_VCCA_LVD	22	0.43	3.3025	0.065	
V1.8S_VCCTX_LVD	22	0	1.7777	0.000	
V3.3S_VCC_GIO	22	0.49	3.3025	0.074	
V1.5S_1.8S_VCCADMI_VRM	22	0.82	1.7639	0.066	
V1.1S_VCC_DMI	22	0	1.0617	0.000	
V_NVRAM_VCCPNAND	22	0.01	1.7777	0.001	
V3.3M_VCCPEP	22	0.02	3.2943	0.003	
V1.1M_VCCAUX	22	0.99	1.0458	0.047	
V1.1M_VCCEPW	2	0.24	1.0476	0.126	
VCCA_DPLL_L	22	1.1	1.0471	0.052	
V1.1S_SSCVCC	22	3.2	1.0443	0.152	
V3.3A_VCCPSUS	22	0.04	3.3053	0.006	



续表 1

V3.3S_VCCPCORE	22	0.01	3.3032	0.002	
V1.1S_VTT	22	0	1.0618	0.000	
V1.1S_VCCUSBCORE	22	0.11	1.045	0.005	
V3.3A_VCCPUSB	22	0.17	3.3056	0.026	
V3.3S_VCCPPCI	2	0	3.3034	0.000	
V1.1S_VCC_SATA	2	0.18	1.0436	0.094	
VCCPLLVRM	22	0.79	1.7645	0.063	
V3.3A_1.5A_VCCPAZSUS	22	0.01	3.3053	0.002	
CK505				0.244	*6
V3.3S	2	0.14	3.3036	0.231	
VDDIO_CLK	2	0.03	0.8171	0.012	
EC				0.033	*6
V3.3A_KBC	2	0.02	3.3053	0.033	
PS/2 port				0.074	*6
V5_PS2	2	0.03	4.941	0.074	
IMVP6.5 VR				0.206	*3
Input 1	10	0.45	4.935	0.222	*1
Output 1	2	1.32	0.8364	0.552	*2
Input 2	2	0.16	12.551	1.004	*1
Output 2	2	1.12	0.8362	0.468	*2
DDR3 VR				0.123	*3
Input	2	0.15	12.547	0.941	*1
Output	2	1.08	1.5142	0.818	*2
System VR				0.521	*3
Input 1	2	0.49	12.544	3.073	*1
Output 1	2	1.71	3.311	2.831	*2
Input 2	2	0.53	12.544	3.324	*1
Output 2	2	1.23	4.952	3.045	*2
V1.1 VR				2.203	*3
Input 1	2	1.17	12.543	7.338	*1
Output 1	2	10.36	1.0742	5.564	*2
Input 2	2	0.29	12.541	1.818	*1
Output 2	2	2.64	1.0519	1.389	*2
V1.8 VR				0.130	*3
Input 1	2	0.61	3.3051	1.008	*1
Output 1	2	0.98	1.7913	0.878	*2

- 1.电压调节器的输入功率；
- 2.电压调节器的输出功率；
- 3.电压调节器自身功耗，即输入功率减去输出功率；
- 4.处理器风扇的功耗。由于无精密电阻测量，采取拔除该风扇电源的方式测量；
- 5.其他部件表示，从 DC 输入功率减去以下所有主要部件而剩下的系统功耗。包括处理器、内存条、SATA、LAN、Super I/O、PCH、CK505、EC、PS/2、和表中 5 个 VR；
- 6.单独部件的功耗是其各路电源信号路径功耗的累积和。

表 2 系统电源分布-Power Management EFI Shell 空闲状态

Table 2 System Power Distribution- Power Management EFI Shell Idle State

部件名称	电阻值 (mΩ)	电阻电压降 (mV)	测量电 压(V)	功耗 (W)	注释
AC 输入功率				9.700	*1
DC 输入功率	7	4.44	12.566	7.970	*2
交流电源适配器				1.730	*3
处理器风扇				0.000	*4
其他部件				1.327	*5
Processor				2.752	*6
VCC_CORE	2	0.05	0.8879	0.022	
VCC_CORE	2	0	0.8879	0.000	
VGFX_CORE	2	0.3	0.9767	0.147	
V1.8S_VCCSFR	2	0.49	1.7893	0.438	
V1.5_VCCDDQ	2	0.47	1.5101	0.355	
V1.1S_VCCTT	2	0.95	1.0529	0.500	
V1.1S_VCCTTA_QPI	2	0.4	1.0522	0.210	
V1.1S_VCC_FDI	2	0.04	1.0528	0.021	
V1.1S_VCC_PEG_DMI	2	0.43	1.0528	0.226	
V1.1S_VCCTTA_DDR	2	0.06	1.0523	0.032	
V1.1S_VCC_SA	2	1.52	1.0528	0.800	
DDR3 DIMM				0.068	*6
VDD	2	0.09	1.5106	0.068	
VDDSPD	22	0	3.3025	0.000	
SATA				1.036	*6

续表 2

V3.3S	2	0	3.3035	0.000	
V5S	2	0.42	4.932	1.036	
LAN 82577				0.201	*6
V3.3M	22	0.96	3.3033	0.144	
V1.1_LAN_M	22	1.21	1.0401	0.057	
Super I/O				0.033	*6
V3.3S	2	0.02	3.3027	0.033	
PCH				0.992	*6
V1.1S_PCH_VCC	2	0.39	1.0442	0.204	
V1.1S_PCH_VCCDPLL_EXP	22	0.68	1.0447	0.032	
V1.1S_VCC_EXP	2	0.15	1.0442	0.078	
V3.3S_VCCA3GBG	22	0	3.3034	0.000	
VCCAFDI_VRM	22	0	1.764	0.000	
V1.1S_VCCDPLL_FDI	22	0.03	1.0448	0.001	
V3.3S_CRT_VCCA_DAC	22	0.03	3.309	0.005	
V3.3S_VCCA_LVD	22	0.34	3.3025	0.051	
V1.8S_VCCTX_LVD	22	0	1.7777	0.000	
V3.3S_VCC_GIO	22	0.39	3.3025	0.059	
V1.5S_1.8S_VCCADMI_VRM	22	0.82	1.7639	0.066	
V1.1S_VCC_DMI	22	0	1.0617	0.000	
V_NVRAM_VCCPNAND	22	0.01	1.7777	0.001	
V3.3M_VCCPEP	22	0.02	3.2943	0.003	
V1.1M_VCCAUX	22	0.83	1.0458	0.039	
V1.1M_VCCPEW	2	0.21	1.0476	0.110	
VCCA_DPLL_L	22	0.05	1.0471	0.002	
V1.1S_SSCVCC	22	3.13	1.0443	0.149	
V3.3A_VCCPSUS	22	0.04	3.3053	0.006	
V3.3S_VCCPCORE	22	0.01	3.3032	0.002	
V1.1S_VTT	22	0	1.0528	0.000	
V1.1S_VCCUSBCORE	22	0.03	1.0469	0.001	
V3.3A_VCCPUSB	22	0.17	3.3054	0.026	
V3.3S_VCCPPCI	2	0	3.3034	0.000	
V1.1S_VCC_SATA	2	0.18	1.0456	0.094	
VCCPLLVRM	22	0.79	1.772	0.064	
V3.3A_1.5A_VCCPAZSUS	22	0	3.3052	0.000	
CK505				0.206	*6

续表 2

V3.3S	2	0.12	3.3037	0.198	
VDDIO_CLK	2	0.02	0.8172	0.008	
EC				0.033	*6
V3.3A_KBC	2	0.02	3.3053	0.033	
PS/2 port				0.074	*6
V5_PS2	2	0.03	4.945	0.074	
IMVP6.5 VR				0.060	*3
Input 1	10	0.04	4.939	0.020	*1
Output 1	2	0.05	0.8879	0.022	*2
Input 2	2	0.01	12.559	0.063	*1
Output 2	2	0	0.8879	0.000	*2
DDR3 VR				0.049	*3
Input	2	0.08	12.556	0.502	*1
Output	2	0.6	1.5118	0.454	*2
System VR				0.283	*3
Input 1	2	0.41	12.555	2.574	*1
Output 1	2	1.45	3.3101	2.400	*2
Input 2	2	0.25	12.555	1.569	*1
Output 2	2	0.59	4.949	1.460	*2
V1.1 VR				0.788	*3
Input 1	2	0.33	12.554	2.071	*1
Output 1	2	2.85	1.0567	1.506	*2
Input 2	2	0.15	12.553	0.941	*1
Output 2	2	1.37	1.0504	0.720	*2
V1.8 VR				0.069	*3
Input 1	2	0.41	3.3052	0.678	*1
Output 1	2	0.68	1.7913	0.609	*2

- 1.电压调节器的输入功率；
- 2.电压调节器的输出功率；
- 3.电压调节器自身功耗，即输入功率减去输出功率；
- 4.处理器风扇的功耗。该模式下风扇不运行；
- 5.其他部件，表示从 DC 输入功率减去以下所有主要部件而剩下的系统功耗，包括处理器、内存条、SATA、LAN、Super I/O、PCH、CK505、EC、PS/2、和表中 5 个 VR；
- 6.单独部件的功耗是其各路电源信号路径功耗的累积和。

表 3 系统电源分布-FTP 400KB/S 下载状态

Table 3 System Power Distribution- FTP 400KB/S Download State

部件名称	电阻值 (mΩ)	电阻电压 降(mV)	测量电 压(V)	功耗(W)	注释
AC 输入功率				10.300	*1
DC 输入功率	7	4.65	12.566	8.347	*2
交流电源适配器				1.953	*3
处理器风扇				0.000	*4
其他部件				1.227	*5
Processor				2.881	*6
VCC_CORE	2	0.09	0.8879	0.040	
VCC_CORE	2	0.01	0.8879	0.004	
VGFX_CORE	2	0.3	0.9767	0.147	
V1.8S_VCCSFR	2	0.49	1.7893	0.438	
V1.5_VCCDDQ	2	0.48	1.5101	0.362	
V1.1S_VCCTT	2	1.02	1.0529	0.537	
V1.1S_VCCTTA_QPI	2	0.49	1.0522	0.258	
V1.1S_VCC_FDI	2	0.04	1.0528	0.021	
V1.1S_VCC_PEG_DMI	2	0.43	1.0528	0.226	
V1.1S_VCCTTA_DDR	2	0.07	1.0523	0.037	
V1.1S_VCC_SA	2	1.54	1.0528	0.811	
DDR3 DIMM				0.068	*6
VDD	2	0.09	1.5106	0.068	
VDDSPD	22	0	3.3025	0.000	
SATA				1.036	*6
V3.3S	2	0	3.3035	0.000	
V5S	2	0.42	4.932	1.036	
LAN 82577				0.319	*6
V3.3M	22	1.33	3.3033	0.200	
V1.1_LAN_M	22	2.53	1.0401	0.120	
Super I/O				0.033	*6
V3.3S	2	0.02	3.3027	0.033	
PCH				1.093	*6
V1.1S_PCH_VCC	2	0.39	1.0442	0.204	
V1.1S_PCH_VCCDPLL_EXP	22	0.68	1.0447	0.032	

续表 3

V1.1S_VCC_EXP	2	0.32	1.0442	0.167	
V3.3S_VCCA3GBG	22	0	3.3034	0.000	
VCCA_FDI_VRM	22	0	1.764	0.000	
V1.1S_VCCDPLL_FDI	22	0.03	1.0448	0.001	
V3.3S_CRT_VCCA_DAC	22	0.03	3.309	0.005	
V3.3S_VCCA_LVD	22	0.34	3.3025	0.051	
V1.8S_VCCTX_LVD	22	0	1.7777	0.000	
V3.3S_VCC_GIO	22	0.39	3.3025	0.059	
V1.5S_1.8S_VCCADMI_VRM	22	0.82	1.7639	0.066	
V1.1S_VCC_DMI	22	0	1.0617	0.000	
V_NVRAM_VCCPNAND	22	0.01	1.7777	0.001	
V3.3M_VCCPEP	22	0.02	3.2943	0.003	
V1.1M_VCCAUX	22	1.06	1.0458	0.050	
V1.1M_VCCEPW	2	0.21	1.0476	0.110	
VCCA_DPLL_L	22	0.05	1.0471	0.002	
V1.1S_SSCVCC	22	3.16	1.0443	0.150	
V3.3A_VCCPSUS	22	0.04	3.3053	0.006	
V3.3S_VCCPCORE	22	0.01	3.3032	0.002	
V1.1S_VTT	22	0	1.0528	0.000	
V1.1S_VCCUSBCORE	22	0.03	1.0469	0.001	
V3.3A_VCCPUSB	22	0.17	3.3054	0.026	
V3.3S_VCCPPCI	2	0	3.3034	0.000	
V1.1S_VCC_SATA	2	0.18	1.0456	0.094	
VCCPLLVRM	22	0.79	1.772	0.064	
V3.3A_1.5A_VCCPAZSUS	22	0	3.3052	0.000	
CK505				0.223	*6
V3.3S	2	0.13	3.3037	0.215	
VDDIO_CLK	2	0.02	0.8172	0.008	

续表 3

EC				0.033	*6
V3.3A_KBC	2	0.02	3.3053	0.033	
PS/2 port				0.074	*6
V5_PS2	2	0.03	4.945	0.074	
IMVP6.5 VR				0.038	*3
Input 1	10	0.04	4.939	0.020	*1
Output 1	2	0.09	0.8879	0.040	*2
Input 2	2	0.01	12.559	0.063	*1
Output 2	2	0.01	0.8879	0.004	*2
DDR3 VR				0.041	*3
Input	2	0.08	12.556	0.502	*1
Output	2	0.61	1.5118	0.461	*2
System VR				0.288	*3
Input 1	2	0.42	12.555	2.637	*1
Output 1	2	1.47	3.3101	2.433	*2
Input 2	2	0.25	12.555	1.569	*1
Output 2	2	0.6	4.949	1.485	*2
V1.1 VR				0.907	*3
Input 1	2	0.35	12.554	2.197	*1
Output 1	2	2.92	1.0567	1.543	*2
Input 2	2	0.18	12.553	1.130	*1
Output 2	2	1.67	1.0504	0.877	*2
V1.8 VR				0.085	*3
Input 1	2	0.42	3.3052	0.694	*1
Output 1	2	0.68	1.7913	0.609	*2

- 1.电压调节器的输入功率；
- 2.电压调节器的输出功率；
- 3.电压调节器自身功耗，即输入功率减去输出功率；
- 4.处理器风扇的功耗。该模式下风扇不运行；
- 5.其他部件，表示从 DC 输入功率减去以下所有主要部件而剩下的系统功耗，包括处理器、内存条、SATA、LAN、Super I/O、PCH、CK505、EC、PS/2、和表中 5 个 VR；
- 6.单独部件的功耗是其各路电源信号路径功耗的累积和。



## 致 谢

首先感谢我的导师胡飞教授，在论文的写作阶段给予我细心的指导和关怀。当我一次次发出小论文，大论文等初稿时，常常能在 2-4 天内收到胡飞老师的回信。在信中是认真的修改意见和鼓励，由于在职读书在时间上常利用节假日来写作论文，而这时胡飞老师还不忘附上一句节日的问候，令我非常感动。胡飞老师严谨的治学态度和学识，令我受益匪浅。

同时在研究生课程学习期间，无论寒冬还是酷暑，所有的老师都在周末或是节日，和我们一起坚持在教室上课，不仅给予我们知识和方法，还用自己的热情引导我们对知识的渴望，作为一名学生我在心底向你们说声谢谢。你们是黄林鹏、饶若楠，姜丽红，王赓，梁阿磊，王赓，沈备军，李健，张艳红等老师。除此之外，管理学生事务的李力和郑国英老师，在日常学生和论文管理的工作中也做的非常认真负责，即时帮助我们了解学校和学院的情况。

在研究生学习期间，我认识了许多同学。每个人身上都有独特之处，而共同之处在于都能承受工作、学习、甚至家庭的压力，放弃周末来学校学习。我们相互鼓励和帮助，度过了一年半近于无休的生活。在学习时的项目中，大家相互合作和探讨。生活上，大家对各自的工作中的问题相互探讨，不仅学习了知识，还更了解整个软件和电子行业的情况。例如邢璐、伍盛、曾斌、连进、江圣祺、梁业飞、王万均、杨远、宗爱军、叶文虎、周小永、万象、沈文婷等人。

本项目的完成，得到了公司领导和同事的帮助，特别是移动事业部和服务器事业部两个实验室的大力帮助。例如：测试仪器的借用和指导如何使用这些设备等。在此向他们表示感谢。

最后还要感谢我的家人，特别是我的妻子给了我最大的支持。她承担了所有照顾小孩的责任和几乎全部的家务，使我有时间全身心投入学习。并且她在我困难和疲倦的时候，总是给我鼓励，使我能坚持下去，度过紧张的研究生生活，战胜了自己，取得了收获。

## 攻读学位期间发表的学术论文目录

- [1] 唐笛. 计算机睡眠 (S3) 唤醒重建 UEFI BIOS 环境技术, 上海交通大学软件学院网站公示, 2010.09