

分类号_____

学校代码 **10487**

学号 **M201176120**

密级_____

华中科技大学

硕士学位论文

基于 HP-UX11iV3 操作系统

BIOS 的设计与实现

学位申请人 王 伟

学 科 专 业：软件工程

指 导 教 师：沈 刚 教授

答 辩 日 期：2014.1.6

**A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree for the Master of Engineering**

**Design and Implementation of BIOS
Based on HP-UX11iV3 Operating System**

Candidate : Wang Wei

Major : Software Engineering

Supervisor : Prof. Shen Gang

Huazhong University of Science and Technology

Wuhan 430074, P. R. China

January, 2014

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，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：

日期： 年 月 日

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，即：学校有权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。本人授权华中科技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。

本论文属于 ☐ 保密， 在_____年解密后适用本授权书。
☐ 不保密。

（请在以上方框内打“√”）

学位论文作者签名：

日期： 年 月 日

指导教师签名：

日期： 年 月 日

摘要

BIOS 是一段被固化到计算机的主板 ROM 芯片上的代码，记录着计算机最为重要基本输入和输出的操作以及系统需要设置的信息。统一可扩展固件接口即 UEFI(Unified Extensible Firmware Interface)，是一种详细描述输入输出类型接口的标准。作为 BIOS 的上层应用，操作系统是用户和计算机之间的中间层，将计算机硬件和其他的软件的接口联系起来。在用汇编语言编写 BIOS 时代，无论是 PC 机还是服务器在设计 BIOS 时候，所有的功能都是最大可能的满足所有操作系统的所有要求。随着操作系统的发展，特别是在 UEFI 技术日益成熟以后，针对专门操作系统设计特殊的 BIOS 成为了一个新的课题。

惠普的 HP-UX11iV3 操作系统，是 HP 公司的完全 64 位的操作系统环境，可为具有极高要求的应用提供极大的扩展能力和极高的系统性能。在 EDKII 开源开发环境下，针对 HP-UX11iV3 系统在企业级应用的一些特点，结合 UEFI 规范化、模块化与可扩展的特点。在 BIOS 设计中，加入惠普操作系统驱动，减少非必需加载的驱动，加快启动时间，可以为 HP-UX11iV3 系统设计出一款专门的 BIOS，满足其性能方面的需求。

在硬件平台加载 UEFI 镜像文件时，由于加入惠普操作系统驱动，可以在系统进入 shell 时，准确的加载 HP-UX11iV3 操作系统。采用 C 语言编写的 UEFI，在 BIOS 设计的时候能够在启动时实现丰富的自定制功能扩展。根据实际需求，添加或者删除相应的驱动，满足启动时对硬件驱动最基本的要求。这样的设计可以更稳定、更快的加载 HP-UX11iV3 系统，促进其在企业级别上更好的应用。

关键词：基本输入输出系统 操作系统 统一的可扩展固件接口

HP-UX11iV3 系统

Abstract

BIOS is a piece of the computer code that is solidified into the ROM chip on the motherboard, it records the most important input and output procedures as well as the information needed to set for the system. UEFI (Unified Extensible Firmware Interface) is a detailed description of the standard type of input and output interface. As an intermediate layer between the user and the computer, operating system links the interfaces between hardwares and other softwares. Written in assembly language BIOS age, all the functions are supposed to meet all the demands of operating systems when it comes to PC as well as when the servers desgins the BIOS. With the development of the operating system, especially after the UEFI technology gradually matures, designing a special BIOS for the operating system has become a new topic.

Hewlett-Packard's HP-UX11iV3 operating system is HP's full 64 -bit operating system environment that can provide great scalability and high systematical performance to high demanding applications. In EDKII open source development environment, based on some features adopted by HP-UX11iV3 systems, combining the features of standardization, modularity and scalability, when we design BIOS, add HP operating systematic drive and reduce the load of the drive which is unnecessary, thus reducing the start duration. As a result, we can develop a unique BIOS for HP-UX11iV3 system to meet its performance needs.

Due to adding the HP operating system drive, when loading UEFI on the hardware platform, it can accurately load HP-UX11iV3 operating system when the system enters the shell. Using UEFI coded by C language, we can help the system acquire affluent self-made function extensions in design of BIOS, and add or delete the corresponding drives to meet the fundamental requirements software drive needs when the computer starts. This kind of design can load HP-UX11iV3 more stably and quickly, promoting a better application at the enterprise level.

Key words: Basic Input OutPut System Operating System

Unified Extensible Firmware Interface HP-UX11iV3 Syetem

目 录

摘 要.....	I
Abstract.....	II
1 绪论	
1.1 研究背景与意义	(1)
1.2 国内外研究现状	(2)
1.3 论文研究内容与组织结构	(4)
2 关键技术分析	
2.1 EDKII 介绍	(6)
2.2 UEFI 驱动模型	(8)
2.3 本章小结	(17)
3 UEFI BIOS 需求分析	
3.1 系统背景	(18)
3.2 UEFI BIOS 需求.....	(18)
3.3 基于操作系统需求	(21)
3.4 本章小结	(23)
4 UEFI BIOS 设计	
4.1 BIM 文件的设计	(24)
4.2 模块描述类文件设计	(26)
4.3 二进制驱动文件设计	(28)
4.4 RSE 算法设计	(29)
4.5 本章小结	(32)

5 UEFI BIOS 实现与测试

5.1 UEFI BIOS 实现.....(33)

5.2 测试结果(43)

5.3 本章小结(46)

6 总结与展望

6.1 全文总结(47)

6.2 展望(47)

致 谢.....(49)

参考文献.....(50)

1 绪论

1.1 研究背景与意义

作为连接操作系统和系统硬件体系之间的桥梁，BIOS 自诞生以来伴随着主板经历了三十多年的风风雨雨，一直是计算机中不可缺少的一部分，为计算机的发展做出了重要的贡献^[1]。但随着最近计算机技术的飞速发展，BIOS 自身技术的发展远远落后于计算机的发展，现有的 BIOS 突显出了它的局限性。只有少数几家公司或厂家如 Phoenix、AMI、Insyde、Byosoft 等掌握 BIOS 的开发技术，其开发技术也严格保密，这也严重制约着计算机的进一步发展^[2]。

1988 年，当时的英特尔、微软、惠普和其他一些公司开启了超越 BIOS 的计划 IBI (the Intel Boot Initiative)，后来这一计划被称为 EFI (Extensible Firmware Interface)，即可扩展固件接口。EFI 把现代计算机的软件架构概念引入固件程序设计，它允许用 C 等高级语言来开发固件，提供对硬件的适当抽象，并具有良好的可扩展性。为了统一 EFI 标准和规范，由英特尔、微软、惠普等全球著名的计算机软、硬件厂商于 2005 年发起成立了国际 UEFI (Unified Extensible Firmware Interface) 联盟，即统一可扩展固件接口联盟^{[3][4]}。

UEFI 的提出，解决了 BIOS 技术上的局限性，并且作为一个国际开放标准，能够有更多的企业和组织参与到 UEFI 的研究与开发中，让每个 IT 成员都接触 UEFI 技术，极大的推动了 UEFI 技术的发展^[5]。传统的 BIOS 程序都是采用的汇编语言和机器语言来编写，新的计算机技术的发展，对于 BIOS 功能要求更高，传统 BIOS 显得无能为力。而 UEFI 使用了模块化、动态链接和 C 语言风格的堆栈参数传递方式，使其在功能实现上更加简单，扩展功能更加强大^[6]。

目前，Intel 开放了 UEFI 平台的源代码，UEFI 的规范标准为计算机硬件和操作系统的相应接口规定了一个新的标准规范模型。这个规范是由一系列的数据表组成，包含了平台相关信息，以及对加载操作系统和启动实时服务的请求。它们一起为启动加载操作系统和导入应用程序准备相应的环境搭建了一个完整的框架^[7]。UEFI BIOS 类似一个操作系统，能够控制所有的硬件资源^[8]。在开机程序进入驱动程序执

行阶段时,通过Shell的应用接口,可以在加载操作系统之前在DXE阶段对网络进行应用、启动的相关信息进行检测、还可以调试相关的软硬件^[9]。

计算机的发展,也带来了操作系统技术的不断革新,为了满足不同的应用需要,各种不同的操作系统应运而生。HP-UX11iV3 系统就是应用在企业级大型服务器上的一款新型操作系统^[10]。HP-UX 操作系统的初期版本只能够支持惠普整机的个人计算机和惠普 9000 系列的工作站,这些计算机都是基于 Motorola 68000 系列处理器的。在惠普公司自主研发和发布 PA-RISC 这款处理器后,HP-UX 操作系统成为了 HP9000 的 700 系列工作站和 800 系列服务器的主要操作系统。Intel 公司发布安腾 Itanium 处理器后,惠普公司推出的 HP-UX 的操作系统也成为支持该处理器架构的主要操作系统^[11]。

HP-UX 是首个内置逻辑卷管理(LVM)功能的操作系统,也是第一个提供文件系统访问控制列表(ACL)的 UNIX 系统。2008 年 HP-UX11i 操作系统被评为具备关键业务虚拟化、可感受性能、高可用及出色管理性的领先 UNIX 操作系统^[12]。

新型的操作系统功能强而且针对于专门的应用环境,但是传统的 BIOS 严重的限制了新型操作系统的将其技术上的优势发挥到极致。针对专门的操作系统定制特定的 BIOS 显得尤为重要。随着 UEFI 技术的发展,也使得这种应用成为了可能。

1.2 国内外研究现状

当前,正是传统 BIOS 技术向着新一代 UEFI BIOS 规范的过渡过程中,UEFI 是有 Intel 联合业界采用开发方式共同制定推出的一种在从服务器、桌面电脑、笔记本电脑到嵌入式系统中替代传统 BIOS 的一种升级方案^[13]。UEFI 已经得到大规模应用。统计表明,超过半数的新一代 Intel 和 AMD 的个人电脑和服务器的 CPU 都支持 UEFI^[14]。微软发表声明:“由于 64 位硬件的普及,UEFI 所带来的功能,以及向 UEFI 固件的快速转移,微软已经选择首先在 UEFI 系统上实现全部与固件相关的新特性。微软将以个案审查的方式评估在旧的 BIOS 系统上支持新功能所需架构工作的可能性^[15]。”惠普公司声明:“自 2008 年开始,所有新设计的惠普笔记本电脑都支持 UEFI 架构。惠普动能服务器、多功能打印机/扫描机和智能存储阵列目前已经使用 UEFI

技术。桌面电脑与工作站已决定采用同一源程序库开发 UEFI^[16]。” IBM 也在 System x 架构服务器上全面使用 UEFI 技术^[17]。

2008 年 4 月，ARM 加入 UEFI 论坛，以支持 OEM 厂商使用 UEFI 开发技术基于 ARM 处理器的解决方案。Intel 公司发表声明：“通过业界多年对 UEFI 的支持和发展，已经在 UEFI 周围形成了一个生态系统。Intel 支持产业界从支持 UEFI 标准，延伸到开发、生产以 Intel 产品和 UEFI 固件为基础的解决方案^[18]。”同时，所有业界领先的 BIOS 供应商都向客户和目标市场提供基于 UEFI 固件的解决方案。从这可以看出，新一代的 UEFI BIOS 必将成为一个核心技术的热点^[19]。

UEFI 联盟目前有了八十多家企业成员，组织约定定期召开 UEFI 技术大会。这些企业成员一共分为三类：推广者、参与者和使用者^[20]。推广者主要负责 UEFI 项目的宣传和推广，参与者负责 UEFI 的相关研究工作，使用者是 UEFI 成果的享有者，在 UEFI 成熟技术成果的基础上，将自己的特制的技术与之结合，应用到具体的项目上。每个成员通过对行业推广，促进软、硬件行业更快的了解、认识并采用 UEFI 标准^[21]。

UEFI 的发展，带来了 BIOS 技术上翻天覆地的变化。对个人电脑来说，优势体现不是很明显。但是对于企业用户来说，优势巨大。比如，传统的 BIOS 只支持 4 个主分区，当工作需要 100 个主分区时，传统的 BIOS 就遇到瓶颈了。但是 UEFI 支持 128 个主分区，UEFI 的研究和应用潜力巨大^[22]。

目前国内外对于 UEFI 的研究，一方面停留在通用性的层面，即设计的 UEFI BIOS 能够满足加载不同的操作系统的要求^[23]。比如 PC 上写入的 BIOS Image 需要支持常见的 Windows 操作系统（如 win7、win8、winXP）、Linux 操作系统（如 Red Hat Linux、CentOS、Fedora）等。UEFI 的通用性设计，能够最大限度的兼容不同的操作系统^[24]。但是它同样带来了一个问题，无法结合特定的操作系统发挥出操作系统在加载前和加载后的优势。特别是在企业级的 UNIX 操作系统上，兼容性的设计，会使得所设计的 UEFI BIOS 复杂度大大增加，在启动中也发挥不了 UEFI 能快速准确加载操作系统的特点。在启动过程中，如果相关硬件驱动发生问题，无法加载操作系统的时候，这系统的排错和纠错的能力也大大减弱^[25]。

另一方面，国内外的企业对于 UEFI 的研究从 PC 领域向着其他电子领域发展，UEFI 在复杂的嵌入式领域的应用也越来越多^[26]。UEFI 的应用变得越来越广泛，更多的企业和组织参与到 UEFI 的研究设计 and 应用中来。

相比 UEFI 技术，操作系统技术也在不断革新，各种操作系统不断升级更新。HP-UX，全称为 Hewlett Packard UniX，它属于 UNIX 的一个变种，基于 SystemV，主要是应用在惠普 9000 系列的服务器上的操作系统，支持 PA-RISC 处理器（惠普公司）、Itanium 处理器（英特尔公司）的 CPU 芯片。许多大型公司采用 HP9000 的服务器管理企业的资源和电子商务的业务，提供的远程办公业务和虚拟主机得到了广泛使用^[27]。在这些服务器上都用的是 HP-UX 系列的操作系统。HP-UX 操作系统还随着用户需求的变化而不断的升级优化，仅 HP-UX11i 操作系统就发布了 V1、V2、V3 三个版本^[28]。

操作系统技术的不断进步，使得不同的用户根据使用目的对操作系统的要求不尽相同，操作系统的定制将会越来越广泛。设计一款 BIOS 兼容所有系统显得越来越力不从心。根据操作系统的特点，定制出专门的 UEFI BIOS 将成为一种趋势。随着 UEFI 技术的发展，根据其模块化和可扩展的特点，针对特定操作系统的设计 UEFI BIOS 将会得到更多的重视，在这个领域进行研究的企业、组织和个人也将会越来越多。

1.3 论文研究内容与组织结构

本文针对 HP-UX11iV3 操作系统，设计出一款 UEFI BIOS，并产生一个 BIOS Image。研究内容主要有：

（1）HP-UX11iV3 操作系统特点研究。HP-UX11iV3 操作系统继承 HP 公司的传统，支持各项 UNIX 标准^[29]。根据 X/OpenSingleUNIX 规范上的说明，HP-UX11iV3 是 UNIX95 标志产品。而且 HP-UX 的 64 位版本是根据 LP-64 数据模型标准编制的。分析 HP-UX11iV3 操作系统的各个特点，能够使后面的 UEFI 驱动选择和设计更具有针对性。

（2）UEFI 的基本框架和驱动模型。UEFI 描述了一个平台的可编程接口。这个

平台包括主板、芯片组、中央处理器（CPU）和其它组件。UEFI 中允许执行在操作系统加载之前（Pre-OS）进行运行的程序（agent）。Pre-OS 程序可以是操作系统的加载器、诊断程序和其他为了执行和互操作的系统应用程序，其中包括 UEFI 驱动程序和应用程序。UEFI 针对依赖它进行互操作的驱动程序和应用程序提供了清晰的纯接口规范^[30]。

（3）UEFI 的启动流程。UEFI 的启动主要分为 Security Core 阶段(SEC)、Pre EFI Initialization 阶段(PEI)、Driver Execution Environment 阶段(DXE)、Boot Device Select 阶段（BDS）、Transient System Load 阶段（TSL）、Run Time 阶段（RT）、After Lift 阶段（AL）^[31]。

（4）BIM 文件的设计和 RSE 算法设计。BIM 文件将记录 BIOS 文件所需要每一个驱动和 EDKII 下的一些配置环境。RSE 算法将 BIM 文件记录的所有驱动，在 EDKII 的 Repository 里寻找出一个 BIOS 驱动的最优算法。

（5）UEFI BIOS Image 的设计。UEFI BIOS 里面所需要的驱动，为分别设计成 *.inf 文件和 *.efi 文件。*.inf 文件记录着每个驱动模块的一些信息，*.efi 文件为这个驱动模块所对应的二进制文件。UEFI BIOS Image 将设计为 *.fd 文件，该 fd 文件为所有驱动对应的 *.efi 文件在 EDKII 环境下 build 生成。

全文一共分为六章

第一章为绪论，介绍了 BIOS 技术和操作系统的发展，本文的研究背景以及意义。

第二章介绍设计 UEFI BIOS 中用到的一些关键性技术。

第三章是结合 UEFI 的设计规范和 HP-UX11iV3 操作系统的特点，对系统设计进行需求分析。

第四章是 UEFI BIOS 设计，介绍了 BIM 文件的设计、INF 文件设计、DEC 文件设计和 DSC 文件设计，设计 RSE 算法去寻找最佳解决方案。

第五章是在 EDKII 环境下，通过 Assembly 去生成 BIOS Image 并测试得到 UEFI Image，验证可用性和稳定性。

第六章是总结，总结了论文的核心内容，提出了其中的不足之处，并对以后的工作进行了展望。

2 关键技术分析

本章主要介绍设计 BIOS Image 过程中的相关技术。包括 EDKII 的开发环境，UEFI 的驱动模型和基本框架。

2.1 EDKII 介绍

EDK (EFI Development Kit) 是一个包含一系列基本的底层库函数和接口的 UEFI 规范标准的框架接口。它包含了每个启动阶段，安全检测阶段 (SEC)、EFI 初始化准备阶段 (PEI)、驱动程序执行环境阶段 (DXE)、启动设备选择阶段 (BDS) 的具体实现。

几年来，UEFI 联盟在 EDK 的基础上进行了功能的优化和增强，形成了功能更大强大的开发框架 EDKII。它在编译控制优化、元数据控制、C 类库的增强等方面进行了扩展^[32]。在模块设计方面，给出了相应的 INF 文件规范^[33]，在设计 INF 文件时还需要遵守 DEC 文件、DSC 文件的相应规范。

用户可以很容易的设计出符合自己要求并且规范的驱动和应用程序模块。并且在用户个性配置的方面，EDKII 实现了 PCD 机制，用户不需要去修改具体的代码实现只需要在 DSC、DEC 文件中设置一个 PCD 的 Token 就可以在编译控制阶段带来很多方便^[34]。

EDKII 中各个模块框架主要包括：BaseTools、Conf、MdeModulePkg、MdePkg、FatPkg、ShellPkg、DuetPkg、NetworkPkg、Nt32Pkg、UnixPkg、UefiCpuPkg、EdkCompatibilityPkg、OptionRomPkg、。此外还包括一个设置环境变量的 edksetup.bat 文件。表 2-1 描述了每个包的功能。

表 2-1 EDKII 各个 Pkg 的功能

模块	描述
BaseTools	包含代码编译所需要的二进制编译工具和编译环境的配置文件
Conf	保存编译环境信息、编译目标路径以及编译器参数
MdeModulePkg	提供跨平台的底层库示例的模块，例如 MdePkg
MdePkg	EDKII 环境下 IA32、X64、IPF 和 EBC 平台的底层库函数、工业标准和协议
FatPkg	包含针对不同 CPU 架构的原始 FAT 驱动
ShellPkg	提供一个平台通用的 UEFI Shell 应用程序开发环境
DuetPkg	提供基于传统 BIOS 的 Runtime 环境的支持库
NetworkPkg	网络驱动模型
Nt32Pkg	在 windows 操作系统下通过加载 32 位模拟器来提供 Runtime 的运行环境
UnixPkg	在 linux 操作系统下通过加载 32 位模拟器来提供 Runtime
UefiCpuPkg	CPU 驱动模型
EdkCompatibilityPkg	提供兼容传统 BIOS 和 EDK 的协议和库
OptionRomPkg	针对不同 CPU 和 PCI 的实例程序
edksetup.bat	该文件提供配置 EDKII 的运行环境, 包括工作区的配置和编译工具使用环境的配置

其中每个 Pkg 都保持了紧密的联系，例如 MdeModulePkg 依赖于 MdePkg，这是由于 MdePkg 中包含了 MdeModulePkg 的使用原型。主要的依赖关系如图 2-1 所示。

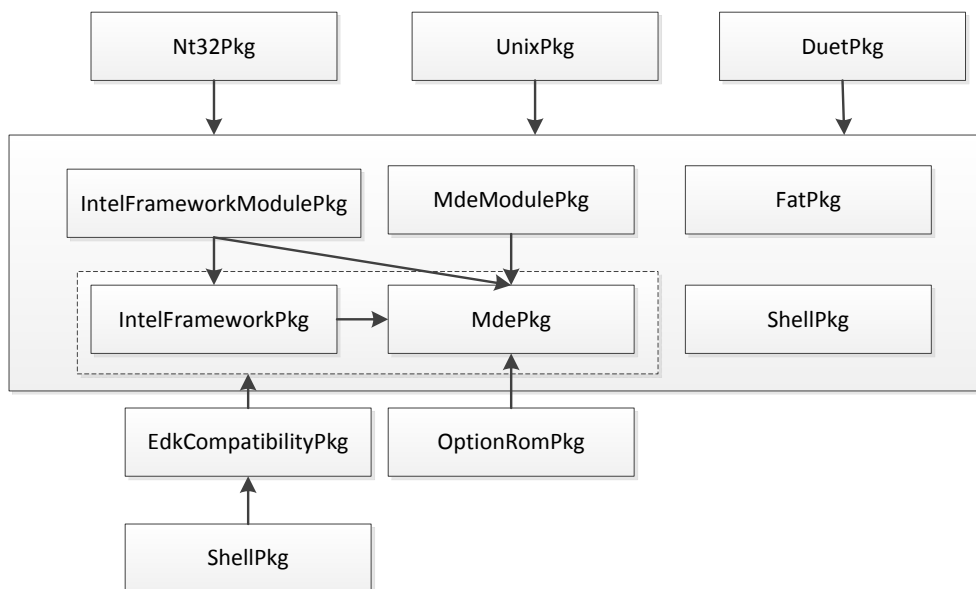


图 2-1 各个 Pkg 的依赖关系

2.2 UEFI 驱动模型

对于UEFI的开发, 首选需要研究UEFI的驱动模型和基本框架, 有针对性的对每个阶段进行设计。

2.2.1 UEFI 驱动模型

UEFI提供了一种驱动模型, 这个模型遵循如PCI和USB等在当今工业标准总线以及未来体系结构的设备。UEFI驱动模型的目的是简化设备驱动程序的设计和实现, 并减少其可执行映像的大小。设备驱动程序必须在用于加载驱动程序的映像句柄上产生驱动绑定协议, 然后等待系统固件把该驱动程序连接到控制器。一旦连接成功, 设备驱动程序就负责在这个控制器的设备句柄上产生一个协议, 这个协议用于抽象控制器支持的I/O操作^[35]。总线驱动程序执行完全一样的任务。此外, 总线驱动程序还负责查找总线上的任何子控制器, 并且为每个检测到的子控制器创建一个设备句柄^[36]。

在任何给定的平台上, 固件服务、总线驱动程序和设备驱动程序的组合很可能由包括OEM、IBV和IHV这样广泛的各种供应商来提供。这些来自不同供应商的不同组件需要协同工作来为IO设备产生一种协议, 该协议用于启动UEFI兼容的操作系统。

传统BIOS系统, 在启动时服务器的许多驱动都要被激活, 在UEFI驱动模型下, 它只允许激活启动必需的设备子集^[37]。随着更高的启动时间要求, 我可以把那些需要激活的设备的驱动策略放入操作系统中, 在操作系统启动之前加载驱动模型。这样可以使系统的快速启动成为了可能^[38]。下面分几部分介绍UEFI的驱动模型:

(1) 驱动程序初始化

驱动程序的映像文件必须从某种类型的媒介载入, 这包括ROM、Flash、硬盘、软盘、USB或者SATA的CD-ROM/DVD, 甚至是一个网络连接^[39]。当驱动程序的映像已经被发现以后, 他就会启动服务(LoadImage())加载到系统内存中。LoadImage()函数把一个COFF/PE格式的映像加载到系统内存中, 系统为驱动程序创建一个句柄。此时, 驱动程序并没有被启动, 它只是在停留在内存中等待启动。遵循UEFI 驱动模型的驱动程序必需在其映像句柄上安装驱动程序绑定协议(Driver Binding Protocol)的实例。它可以有选择地安装驱动配置协议(Driver Configuration Protocol)、驱动诊断

协议(Driver Diagnostics Protocol)或者组件名称协议(Component Name Protocol)。此外,如果一个驱动程序希望可以被卸载,它可以有选择地更新已加载映像协议(Loaded Image Protocol)来提供自己的Unload()函数^[40]。最后,如果一个驱动程序在启动服务ExitBootServices()被调用时需要执行任何特定的操作,那么它可以有选择地创建一个支持通知功能的事件,启动ExitBootServices()服务将触发该事件。一个包含驱动绑定协议实例的映像句柄被称为驱动映像句柄。

(2) 主机总线控制器

驱动程序在其入口函数处不允许接触任何硬件。因此,虽然驱动程序被载入和启动,但是它们都在等待被通知去管理系统中的一个或多个控制器。一个像UEFI启动管理程序这样的平台组件负责管理驱动程序和控制器之间的连接。然而,甚至在第一个连接被建立之前,为了能够管理驱动程序,一些初始集合的控制器也必须被准备好。这个初始集合的控制器被称为主机总线控制器(Host Bus Controllers)。主机总线控制器提供的I/O抽象通过在UEFI驱动模型的范围之外的固件组件产生。用于主机总线控制器的驱动程序句柄和它们的I/O抽象必须通过平台上的核心固件产生,或者通过一个不符合UEFI驱动模型的UEFI驱动程序来产生。

(3) 设备驱动程序

设备驱动程序不允许创建任何新的设备句柄。相反,它只在一个现有的设备句柄上安装其它的协议接口。设备驱动程序最常见的类型是负责把I/O抽象挂接到总线驱动程序创建好的设备句柄上。该I/O抽象可用于启动一个UEFI兼容的操作系统。这些I/O抽象的例子可能包括简单文本输出(Simple Text Output)、简单输入(Simple Input)、块I/O(Block I/O)和简单网络协议(Simple Network Protocol)。接到设备句柄的设备驱动程序必须已经在自己的映像句柄上安装上一个驱动程序绑定协议(Driver Binding Protocol)。驱动程序绑定协议包含三个函数:Supported()、Start()和Stop()。Supported()函数检测驱动程序是否支持一个给定的控制器。如果一个驱动的Supported()函数通过了检测,那么驱动程序能够通过调用驱动程序的Start()函数被连接到控制器。Start()函数决定了实际上添加了什么样的附加I/O协议到一个设备句柄上。在这个例子中,Block I/O协议被安装。为了提供对称性,驱动程序绑定协议也

使用Stop()函数来迫使驱动程序终止管理驱动句柄。这将引起设备驱动程序卸载那些Start()安装的所有协议接口。

UEFI驱动程序绑定协议的Support()、Start()和Stop()函数对于用新的启动服务OpenProtocol()来获得协议接口和用新的启动服务CloseProtocol()来释放协议接口来说是必需的。OpenProtocol()和CloseProtocol()会更新由系统固件维护的句柄数据库，以跟踪哪些驱动程序正在使用哪些协议接口。句柄数据库中的信息能被用于检索关于驱动程序和控制器的信息。新的启动服务OpenProtocolInformation()能被用于获取当前正在使用一个特定协议接口的组件列表。

(4) 总线驱动程序

从UEFI驱动模型的角度来看，总线驱动和设备驱动几乎是相同的。唯一的差异的是总线驱动会为在自己总线上发现的子控制器创建新的设备句柄。所以，总线驱动比设备驱动稍微复杂一些，但反过来这简化了设备驱动的设计和实现。总线驱动主要有两种类型。第一种是在第一次调用Start()时为所有的子控制器创建句柄。第二种总线类型非常有效的支持快速启动能力。它允许每次创建一些甚至是一个子句柄。总线用很长的时间去枚举所有的子控制器（如SCSI总线），这导致在平台启动时花费非常多的时间。图2-2是总线控制器的树结构调用Start()之前和之后的情况。

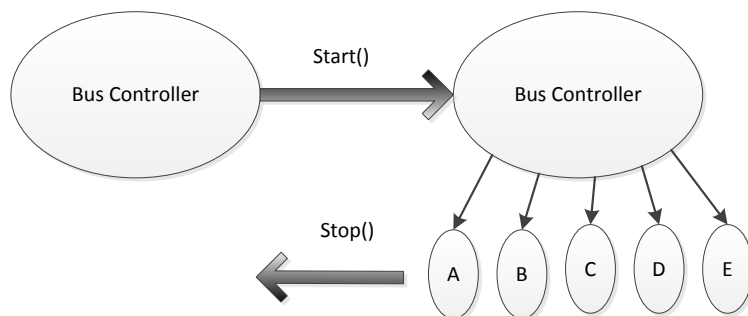


图 2-2 连接总线驱动程序

连接总线控制器节点的虚线表示到总线控制器的父控制器的一个连接。如果总线控制器是主机总线控制器，它没有父控制器。节点A、B、C、D和E表示总线控制器的子控制器。总线驱动支持为每一个Start()的调用创建一个子控制器，可能会选择最先创建C，然后E，之后是A，B和D。驱动绑定协议的Supported()、Start()和Stop()功能具有足够的灵活性来允许这种类型的行为。

（5）平台组件

在UEFI驱动模型下，从平台里内的控制器连接和断开驱动的动作是在平台固件的控制下的。这作为UEFI启动管理的一部分被典型地实现，但是其实现是有必要的。平台固件可以使用新的启动服务ConnectController()和DisconnectController()来判断哪些控制器已经被驱动，哪些还没有。如果平台希望运行系统诊断或者安装一个操作系统，则它可能会选择连接驱动到所有可能的启动服务。如果一个平台希望启动一个预安装的操作系统，它可以选择只连接驱动到启动操作系统需要的设备。UEFI驱动模型通过新启动服务ConnectController()和DisconnectController()支持所有这些操作模式。此外，因为负责启动平台的平台组件必须根据控制台设备和启动选项的设备路径来工作，所以涉及UEFI驱动模型的所有服务和协议都根据记住的设备路径进行了优化。

平台也可以选择产生一个可选协议，称之为平台驱动重载协议(Platform Driver Override Protocol)。这个协议与特定总线驱动重载协议(Bus Specific Driver Override Protocol)相似，但它拥有更高的优先级。当决定哪个驱动被连接到哪个控制器的时候，这个协议赋予平台固件最高的优先级。平台驱动重载协议被挂接到系统中的一个句柄上，如果系统总存在该协议，那么新启动服务ConnectController()将会充分利用它。

（6）热插拔事件

过去，在预启动环境中系统固件不处理热插拔事件。但是，随着如USB这样的总线的出现，用户可在任何时候添加和移除设备，确认有必要描述在UEFI驱动模型里这些总线的类型是重要的。对这些事件的支持需要依赖支持设备热插拔总线的驱动程序来提供。为了这些总线的类型，一些平台管理将移到总线驱动中。例如，当一个键盘被添加到平台上的USB总线，用户希望该键盘被激活。一个USB总线驱动检测热插入事件并为键盘设备创建子句柄。但是，如果驱动不被连接到控制器，键盘将不会变成活动的输入设备，除非调用ConnectController()。要使键盘驱动可用，需要USB总线驱动在热插入事件发生的时候调用ConnectController()。此外，USB总线驱动将在热拔出事件发生时调用DisconnectController()。

设备驱动也受热插拔事件的影响。就USB来说，设备能够在没有任何通知的情

况下被移除。这意味着USB设备驱动的Stop()功能必须将这些不再存在于系统中的设备驱动关闭。因此,任何多余的I/O请求必须被清空掉而不能实际地去访问设备硬件。

总的来说,添加支持热插拔事件将大大增加总线驱动和设备驱动的复杂性。是否增加这种支持是驱动编写人员根据具体的情况来决定,所以必须权衡预启动环境中的功能需求造成的额外复杂性和驱动规模。

2.2.2 UEFI 基本架构

UEFI描述了一个平台的可编程接口。这个平台包括主板、芯片组、中央处理器和其它组件。UEFI中允许执行在操作系统加载之前(Pre-OS)进行运行的程序。Pre-OS程序可以是操作系统的加载器、诊断程序和其他为执行和互操作的系统应用程序,其中包括UEFI驱动程序和应用程序。UEFI针对依赖它进行互操作的驱动程序和应用程序提供了清晰的纯接口规范^[41]。UEFI的基本架构包括:基于UEFI固件管理的对象、EFI系统表、句柄数据库、协议、UEFI映像、事件和任务优先级。

(1) 基于UEFI固件管理的对象。

UEFI通过提供服务来实现几种不同类型的对象的管理。一些UEFI驱动程序可能需要访问环境变量,但是大部分驱动程序不需要这样做。少有UEFI 驱动程序需要使用单调计数器(Monotonic Counter)、看门狗定时器(Watchdog Timer)和实时时钟(Real-time Clock)。EFI系统表是最重要的数据结构,因为它用于访问UEFI提供的所有服务和附加的数据结构,而这些附加的数据结构用于描述平台的配置。

(2) EFI系统表

EFI系统表是UEFI中最重要数据结构。EFI系统表的指针作为入口函数的一部分被传递到每一个驱动程序和应用程序。从这个数据结构中,可执行的UEFI映像就能访问系统配置信息和足够的UEFI服务:UEFI启动服务、UEFI运行时服务、协议服务。UEFI启动服务和UEFI运行时服务的访问分别通过EFI启动服务表和EFI运行时服务表来进行。

这两个表都是EFI系统表里的数据段。每张表所能提供的服务质量和种类对于

UEFI规范的每个修订版本来说都是固定的。UEFI启动服务和UEFI运行时服务在UEFI规范中定义。协议服务是一组相关的功能和数据字段，它们被命名为全局唯一标识符（GUID，Globally Unique Identifier）。协议服务用于为诸如控制台、磁盘和网络这样一类的设备提供软件抽象，但是它们可被用于扩展平台可用的普通服务的数量。

协议是UEFI固件扩展功能的机制。UEFI规范定义了超过 30 种不同的协议，各种各样的UEFI固件的实现，UEFI驱动程序可以产生更多的协议来扩展平台功能。

（3）句柄数据库

句柄数据库是被称为句柄和协议的对象所组成的。句柄是一个或者多个协议所组成的集合，而协议是带GUID字符来命名的一个数据结构。协议的数据结构可能是空的，可能包含数据字段，可能包含服务，或者可能既包含服务又包含数据字段。在UEFI初始化过程中，系统固件、UEFI驱动程序和UEFI应用程序对句柄进行创建，然后把一个或者多个协议安装在句柄上。句柄数据库里的信息是全局的，并且可被任何可执行的UEFI映像访问。

句柄数据库是一个由基于UEFI的固件来维护的对象的中心资料储存库。句柄数据库是一个UEFI句柄列表，而每一个UEFI句柄通过一个唯一的句柄号来识别，这个句柄号通过系统固件来维护。

一个句柄号给句柄数据库的入口提供一个“关键字”。句柄数据库的每一个入口都是一个或多个协议的集合。由GUID来命名的被安装在一个UEFI句柄上的协议的类型决定了句柄的类型。

一个UEFI句柄可以描述这几个组件：像EFI驱动程序和EFI应用程序这样一类的可执行程序、像网络控制器和磁盘驱动器这样一类的设备、像EFI还原和EBC虚拟机这样一类的EFI服务。

图 2-3 显示了句柄数据库的一部分。除了句柄和协议之外，一个对象的列表与每一个协议联系起来。该列表被用于跟踪哪个程序使用哪种协议。这一信息决定了UEFI驱动程序的操作，因为这个信息可以允许UEFI驱动程序在没有资源冲突的情况下被安全地加载、启动、停止和卸载。

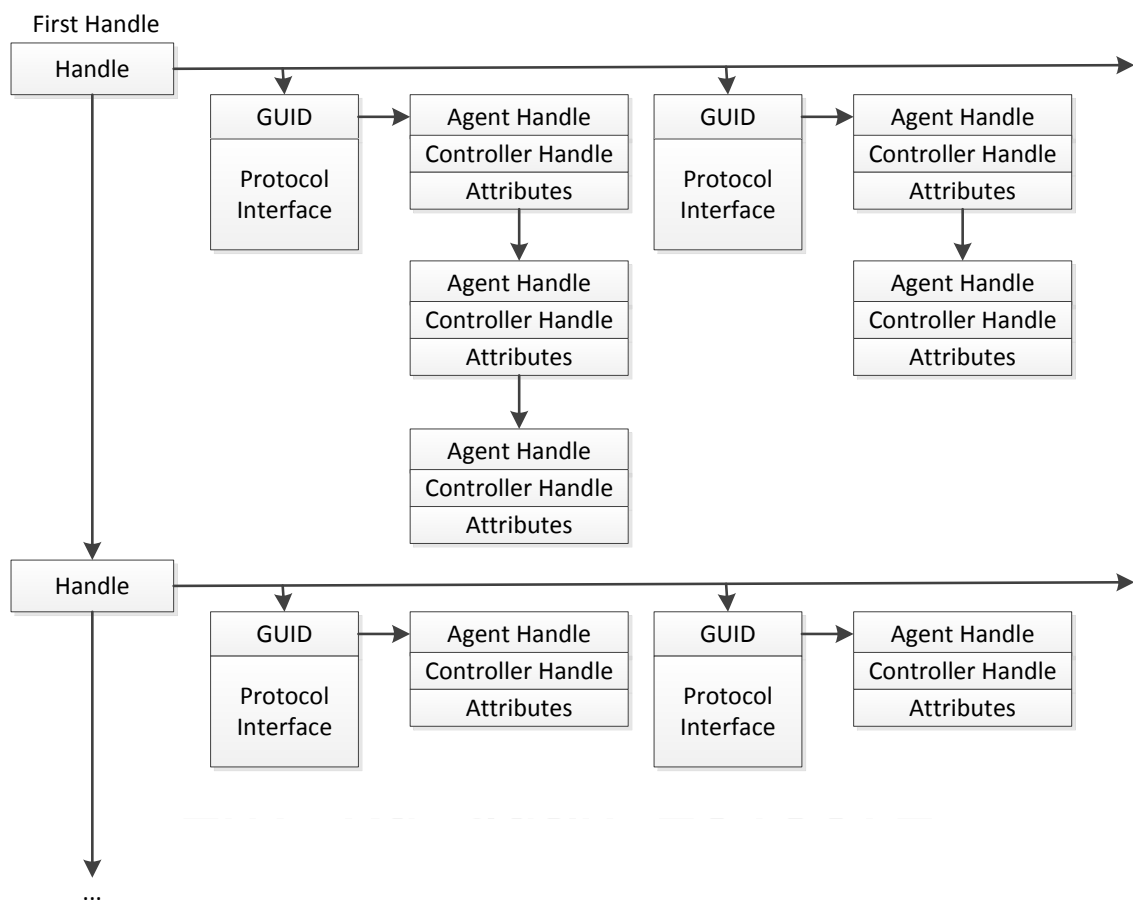


图 2-3 句柄数据库

(4) 协议

UEFI的扩展性在很大程度上依赖于协议。UEFI驱动程序往往会与UEFI协议混淆。虽然它们的关系很密切，但是他们确实是不同的。一个UEFI驱动程序是一个安装了各种协议的多种句柄来完成工作的可执行映像。UEFI协议是一组有函数指针和数据结构块，或者由规范定义的API构成的集合。每个规范必须定义一个GUID。这个GUID号是协议真正的名字^[42]。例如：

```
#define EFI_COMPONENT_NAME_PROTOCOL_GUID \
{0xba62bc2c, 0xd8d0, 0x425c, 0x86, 0x7a, 0xb4, 0x99, 0x17, 0xeb, 0x9d, 0x3c}
```

图 2-4 显示了由UEFI驱动程序产生的句柄数据库中的一个单一的句柄和协议。这个协议有一个GUID和一个协议接口结构组成。通常，产生一个协议接口的UEFI驱动程序还需要维护额外的私有数据段。协议接口结构本身仅仅包括协议函数的指

针。协议函数实际上包含在UEFI驱动程序里。一个UEFI驱动程序可以产生一个或多个协议，这取决于驱动程序的复杂程度。

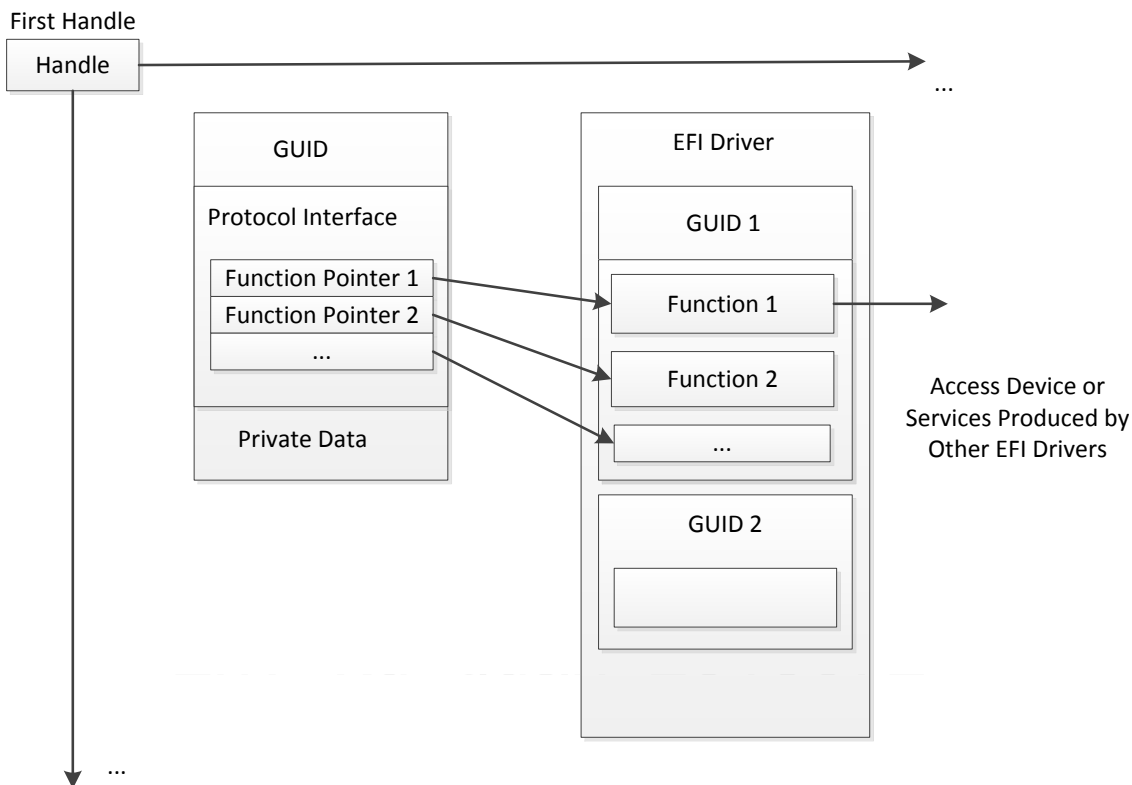


图 2-4 协议结构

UEFI的可扩展性允许每个平台的开发者设计和添加特定的协议。通过这些协议，能够扩充UEFI的能力，并提供对专有设备和接口的访问，而这些接口和UEFI体系的其余部分保持一致^[43]。任何UEFI代码都能在启动期间使用协议。因为协议以一个GUID来命名，所以其它的协议不具有相同的标识号码。当创建一个新的协议时，就给它定义一个新的GUID。

(5) UEFI映像

所有的UEFI映像都包含一个PE/COFF头部，该PE/COFF头部定义了处理器类型和映像类型。代码的目标处理器可以是IA32 架构，X64 架构，IPF架构和EBC架构。映像类型为UEFI应用程序、UEFI启动服务驱动程序、UEFI运行时驱动程序。系统支持的UEFI映像存储单元有：系统ROM或者系统Flash、PCI上的扩展ROM、媒体设备，如SATA或USB硬盘、DVD等、局域网络启动服务器^[44]。

（6）事件和任务优先级

事件是通过UEFI服务进行管理的另外一种类型对象。事件是可以被创建和销毁的，一个事件也能够处于等待状态或者触发状态。由于UEFI不支持中断操作，取而代之的是对驱动程序的轮询。图 2-5 是UEFI支持的不同类型的事件以及这些事件相互之间的关系。

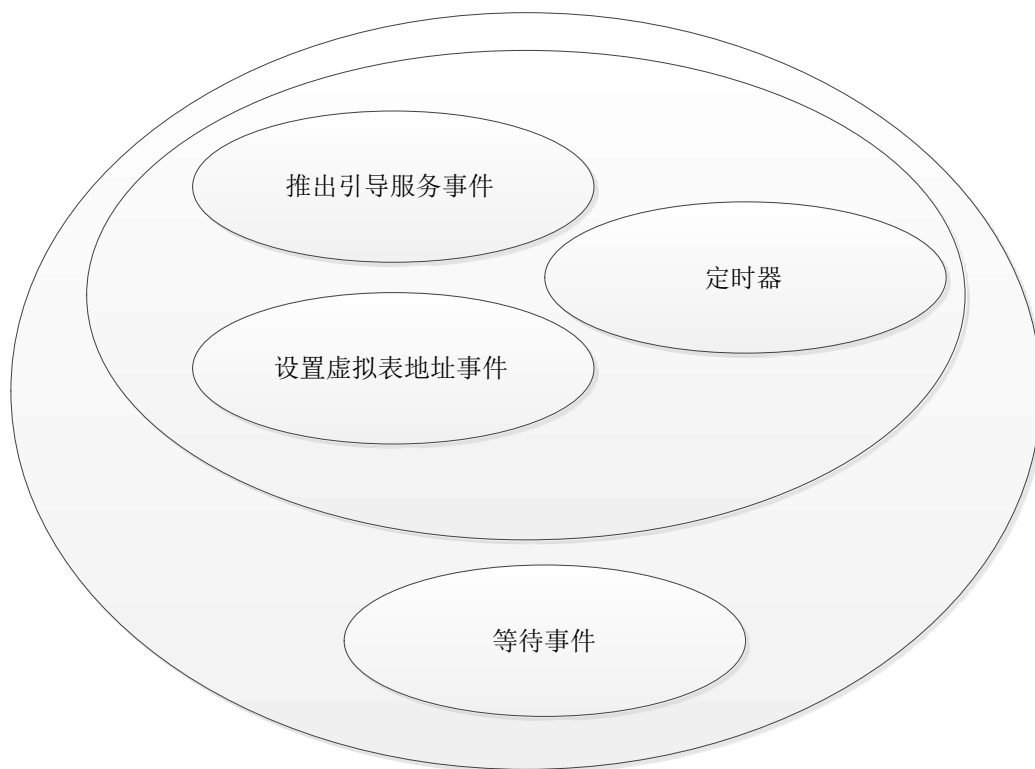


图 2-5 各种事件的类型和相互关系

每个事件都与事件的任务优先级、通知函数和通知的上下文三个要素相关联。当事件的状态为被检查或者事件的状态为正在被等待时，等待事件的通知函数被执行。每次当事件从等待状态转移到信号触发状态时，信号事件的通知函数都会被执行。每当通知函数被执行时，通知的上下文就会被传递给这个通知函数。TPL即事件的任务优先级（Task Priority Level）是通知函数被执行的优先级。表 2-2 列出了四种目前被定义的TPL级别。一个符合TPL列表添加条件的例子可能包括在TPL_NOTIFY和TPL_HIGH_LEVEL之间的一系列的“中断 TPL”，这些“中断”是为了在UEFI内提供中断驱动的I/O支持^[45]。

表 2-2 UEFI 中定义的任务优先级

任务优先级	描述
TPL_APPLICATION	UEFI 映像执行的优先级
TPL_CALLBACK	大多数通知函数的优先级
TPL_NOTIFY	大多数 I/O 操作执行的优先级
TPL_HIGH_LEVEL	UEFI 中支持的计时器中端的优先级

对于优先权，只有在多个事情同时处于信号触发状态时才使用这种机制。应用程序首先执行注册为高优先级的通知函数，这样拥有较高优先级的通知函数能够中断目前正在执行的较低优先级的通知函数。

2.3 本章小结

BIOS Image 设计中有很多的关键技术要点，EDKII 的开发环境为整个 BIOS Image 的开发提供了标准规范的框架接口。UEFI 的驱动模型和基本架构为设计 UEFI 驱动提供了理论依据。

3 UEFI BIOS 需求分析

3.1 系统背景

随着计算机技术的发展，UEFI 技术广泛的应用到各种电子领域。Phoenix、AMI、Insyde、Byosoft 等 BIOS 公司应用最新的 UEFI 技术，定制出满足不同应用需求的 UEFI 产品。华硕、技嘉等主板厂商将这些定制的 UEFI 产品固化到最新的主板上，推向市场，UEFI 技术得到了空前的发展。本文提出的针对特定的操作系统，设计出与之匹配，并能更快、更安全准确加载操作系统的 UEFI 产品。将 UEFI 新技术与操作系统的具体应用结合起来，为 UEFI 技术的发展，提供了另外一种发展方向。

本文为专门操作系统设计 UEFI 的思想能够为研究 UEFI 技术的企业提供一种新的研究思路。设计和生成 UEFI BIOS Image，也可以供主板厂商直接固化到主板内，将对应的产品推向各种需求不同的市场。所以在 BIOS 设计时，需要满足 UEFI 设计的基本需求，也要结合操作系统的具体特点，满足其特定需求。

3.2 UEFI BIOS 需求

作为一个开放的业界标准接口，UEFI在系统固件和操作系统之间定义了一个抽象的编程接口，就像操作系统的API一样。UEFI的规范并没有限制这个接口的具体实现，标准的UEFI接口可以有多种不同架构的实现，它们对外都表现为相同的接口。对于UEFI设计了实现方案，取名为Platform Innovation Framework for EFI，简称Framework。

PI (Platform Initialization) 即平台初始化规范建立了固件内部接口架构以及固件和平台硬件间接口，而后者使得平台硬件驱动程序具有模块化和互操作性。Framework实现了UEFI和PI规范。

图 3-1 按照系统架构由低到高的层次，介绍组成UEFI系统的各个模块之间的相互关系；平台硬件、PI、UEFI和操作系统的相互联系和各自的作用。

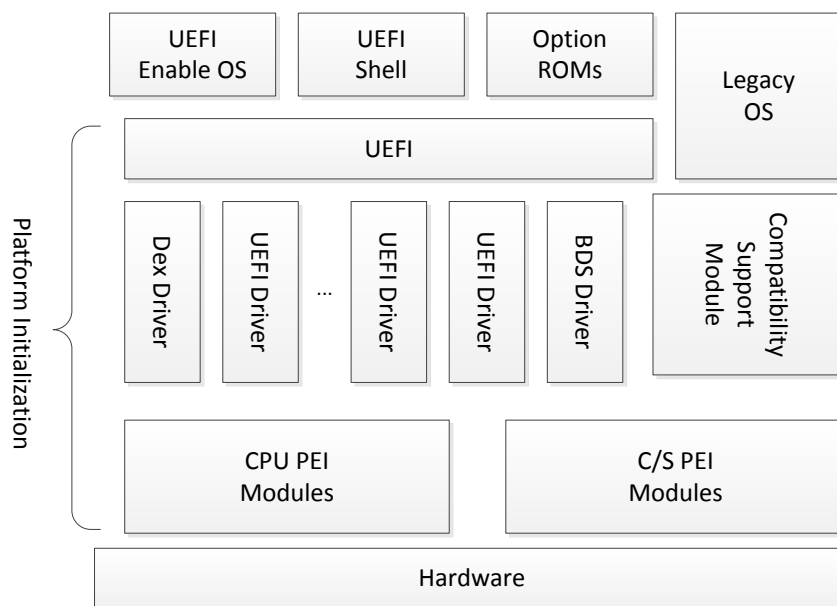


图 3-1 UEFI BIOS 的层次结构

UEFI的层次结构主要由 4 层组成：UEFI初始化准备层（PEI，Pre-EFI Initialization）、固件驱动执行环境层（DXE，Driver Execution Environment）、UEFI接口层、UEFI应用层。

在PEI层将启动最少量的必备硬件资源，这些系统启动所必需的硬件资源是可以满足启动最基本的固件驱动，如CPU、Flash驱动等。PEI层会把硬件资源的启动信息传递给固件驱动执行环境层。DXE层是将初始化所有硬件，为上层接口中实现所有UEFI的规范定义各种服务，它在UEFI的层次结构中起着很重要的作用。UEFI接口层，是固件对外的接口，提供标准UEFI所规定的所有协议与接口。UEFI应用层，它基于UEFI接口的调用，可以实现在进入操作系统前的应用，完成操作系统的加载；在这个阶段，也可以完成对一些兼容性模块的加载。它提供了与传统BIOS完全一致的接口，来支持传统的操作系统以及基于传统BIOS中端调用的应用程序。UEFI各个层次结构之间主要关系如下：

PI和UEFI架构组成了UEFI BIOS的层次结构的主体，它们往下连接了平台硬件，往上连接了操作系统和其它应用。

(1) PI主要由UEFI初始化准备层（PEI）和固件驱动执行层（DXE）这两层组成PEI层初始化系统，并提供最少量的内存；而DXE层提供可支持C语言代码的DXE驱动程序底层架构。基于Green H架构（实际上是PEI基础和DXE基础），PI使得DXE

驱动程序标准化和模块化，从而具有很强的互操作性。PI架构是UEFI得以实现的基础，推动了固件组件提供者之间的互操作能力。

(2) UEFI接口就是UEFI本身，它仅提供接口。UEFI接口层是一个接口规范，它的意思是统一可扩展固件接口，是新一代的BIOS，用于启动服务，具有良好的灵活性和兼容性。Framework不仅实现了PI架构，还实现了UEFI架构。

(3) UEFI的应用层很有特点。在操作系统启动之前支持某些底层应用，甚至可以直接应用UEFI来完成控制任务而不需要操作系统，这在传统BIOS时代是非常困难的。使用UEFI，可以在操作系统启动之前就欣赏一部好莱坞大片，或者运行一些嵌入式应用。对UEFI进行嵌入式设计，具有启动快速、代码容量小、廉价等优势。通过UEFI，甚至还可以实现初级的操作系统功能，如上网，文件管理等功能。

根据 UEFI 每个层次的特点，可以总结出 UEFI、PI 和 Framework 在设计过程中的每项主要目标。

1) UEFI 的设计目标主要有如下几点：

- (1) 提供统一的可扩展固件接口和接口规范；
- (2) 在平台固件和操作系统之间定义抽象的编程接口；
- (3) 可支持高级语言，比如 C 语言；
- (4) 提供系统启动管理；

2) PI的设计目标主要有如下几点：

- (1) 提供平台初始化服务；
- (2) 提供固件内部接口规范，以及固件和平台硬件间接口规范；
- (3) 提供模块化组件，比如PEI组件、DXE驱动程序等；
- (4) 提供UEFI实现的平台基础。

3) Framework的设计目标主要有如下几点：

- (1) 提供UEFI规范和PI规范的实现；
- (2) 基于驱动程序的设计。

PI规范包括PEI核心接口、DXE核心接口、共享结构单元、系统管理模式芯片接口和常用接口标准模块等内容。

3.3 基于操作系统需求

3.3.1 HP-UX11iV3 操作系统特点分析

自从惠普公司推出高端服务器以来，HP-UX 操作系统被广泛应用到各种高端服务器上。HP-UX 操作系统也在可靠性、安全性和分区功能等特点上进行了优化，以满足各类服务器需求。

可靠性主要体现在单个系统质量和出现故障后的纠错能力，多个系统之间故障的切换，以及对于问题的跟踪监视能力。HP-UX11i 系统提供高度的可用性作为多系统集群，全域负载管理系统来管理和优化操作系统的系统性能，并且与临时的增加容量机制配合使用，提供系统使用高峰时的负载处理能力。HP-UX 在开发过程中一直重视集成着安全性能。

从 HP-UX11i 版的操作系统开始，加入“可信任”模式。到 HP-UX11iV2 系统时，增加了基于内核的入侵检测、强随机数生成、堆栈缓冲溢出保护、安全分区、基于角色的访问控制、访问控制白名单，以及各种基于开放源代码的安全工具，安全性能得到了很大的提高。HP-UX 系统的虚拟化分区技术包括了基于硬件分区、软件分区、虚拟机和操作系统虚拟分区等各种技术。

HP-UX11iV3 除了具备以上特点以外，还具有以下几种扩展功能：灵活的容量配置、稳定的可用性和简洁的管理，同时不断的提高操作系统的性能。

(1) 灵活的容量配置：企业级的应用对于服务器的容量有着很高的要求。HP-UX11iV3 可以对容量进行灵活的配置，这样可以解决目前企业在使用中所面对的问题。而且在以后大数据时代，当数据指数级增长时，能后按照需求来增加负载和其容量。这就使得用户可以将更多的精力和时间投入到业务上，不需要被技术能力限制服务器的使用。它能够支持 1 亿 ZB 的存储容量，使用下一代最先进的大数据存储技术使数据管理更加轻松。内存的自动调整和处理器的不断优化，可以使得在断电停机的情况下来增大服务器的容量。

(2) 稳定的可用性：该操作系统通过内存的热交换、增强处理器的性能和使用 I/O 卡使得性能得到了很大的提升。利用动态根磁盘在线修补 HP-UX 软件。

HP-UX11iV3 有着加密卷和文件系统的功能，可以在休眠时保护数据，这是 HP-UX11iV3 操作系统增加的一个很重要的安全性功能，也是行业内第一个使用了 UNIX 加密卷和文件系统的系统。

(3) 简洁的管理：当操作系统的功能繁多时，使用者就需要为了保持系统的竞争力投入很多的时间。HP-UX11iV3 提供自行修复和简化管理功能，内核具有自动调整功能，这样子在使用过程中可以进行无形的调整，当发现存储设备和 I/O 通道时，就进行自动的配置。HP-UX 系统有一个系统管理的主页，为使用者提供了一个简化的 UI。V3 和 V2 提供的新的操作系统管理功能能够提高系统的可用性，利用软件助手能够使操作系统管理更加的简化。

(4) 性能的不断提升：HP-UX11iV3 与 HP-UX11iV2 相比，操作系统的性能平均提高了 30%。当考虑 HP-UX11iV2 在重要的业界基准测试中创造的世界纪录时，这就说明了许多问题。成千上万个 HP-UX11i 的应用不需要重新进行编译或者修改，就能够直接在 V3 上运行，这样性能的提高可以被充分的应用。HP 操作系统强大的二进制兼容性功能，可以为投资提供保护，来满足客户的需求。目前，HP-UX11iV2 在业界的系统基准测试中处于领先的地位，测试包含了数据仓库、企业资源规划、Java 应用和联机事务处理。

(5) 操作环境：HP-UX11iV3 采用模块化的软件，简化了定制的安装和更新。HP-UX11iV3 OE 在保持集成和测试优势的同时，将软件组件分成多个产品类别，使用户可以更加轻松可靠地更新软件组件。

本文主要研究的是UEFI的发展，对系统加载操作系统的速度提升和稳定性的作用，结合HP-UX11iV3 系统的企业级应用和技术上良好的发展应用前景，使得本文非常有研究意义。

3.3.2 基于 HP-UX11iV3 的操作系统需求

由于本文是基于HP-UX11iV3 的操作系统设计，因此在设计出的UEFI BIOS要充分考虑到操作系统的特点，满足其可用需求，性能需求。

(1) 可用需求

HP-UX11iV3 操作系统是一款UNIX系统，在进入引导和加载操作系统的BDS

（Boot Device Selection）阶段后，需要保证UEFI能够找到该系统，并准备加载这款操作系统，需要在BIOS Image的全局设计中加入支持HP操作系统的PCD。

（2）性能需求

对于HP-UX11iV3 操作系统主要是应用在企业大型服务器上，对于加载该系统的速度存在着很高的要求，可以去掉一些不必要的硬件，来提高系统的速。在加载UEFI的时候，当系统进入DXE阶段，需要去发现和执行的硬件越少，这个阶段所花的时间将会越短。

3.4 本章小结

本章主要介绍了UEFI BIOS在设计中的功能需求，并结合HP-UX11iV3 操作系统的特点，对适用于该系统的BIOS设计提出了可用需求、性能需求。

4 UEFI BIOS 设计

UEFI BIOS 主要由二进制镜像管理文件 BIM (Binary Image Management) 来管理每个硬件的设备、外部连接设备和驱动文件, 模块描述类文件来记录驱动文件信息、包定义信息和平台描述信息。通过设计 RSE (Repository Search Engine) 算法去寻找驱动的解决方案。总体设计框架如图 4-1 所示。

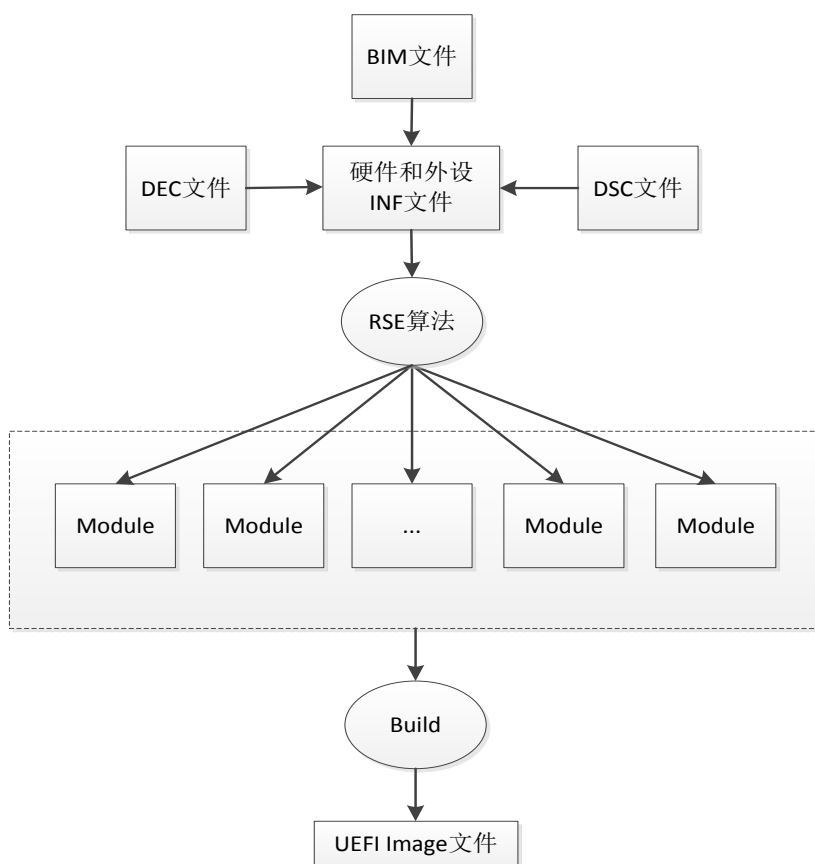


图 4-1 系统设计框架

4.1 BIM 文件的设计

二进制镜像管理文件 BIM (Binary Image Management) 文件来记录每个硬件设备和外设以及所需要的驱动。一个 BIM 文件包含有用户配置设备的结果, 如果这个配置信息完成, 它还会记录生成 File.fd (Bios Image) 文件所需要的一些系统配置文件和 Firmware 装配操作中需要产生的 DSC/FDF 文件。BIM 文件的设计需要遵

循 EDKII 的 BIM 文件的规则。

BIM 文件是一个 INI 格式的文件，将包含以下的信息：

(1) 平台的名字 (Platform Name)

给这个 BIM 文件的工程设置一个有别于其他工程的名字。

(2) 平台的版本号 (Platform Version)

在 build UEFI BIOS 的时候需要借助于 EDKII 下的 Repository 即 Pkg 的包，需要设定这个 Repository 的版本号，本文所用到的是 1.0。

(3) 固件供应商 (Firmware Vendor)

生成好的 BIOS Image 需要借助一个硬件平台去测试 Image 是否能启动操作系统，这个就是要设置这个硬件的供应商。本文用到的是 Inforce9450 这个硬件平台。所以把 Firmware Vendor 设置成 Inforce9450 Reference Platform。

(4) 固件所创建二进制文件的数量 (Firmware Build Number)

Build 完二进制文件后需要产生的 Image 的个数。本文只需要产生一个。

(5) 固件所创建二进制文件的时间 (Firmware Build Date)

这是在 build 完 image 之后，记录的 build 时间。

(6) 固件最后修改时间 (Firmware Last Modification Date)

最后一次修改 BIM 文件的时间。

(7) EDKII 包的路径 (Repository Path)

EDKII 下所有包 Pkg 的集合，就是所有驱动存放的路径。

(8) 工具箱的路径 (Toolset Path)

需要借助打包好了的 build.exe, assembly.exe 进行 image 的 build，这个路径就是这些工具的路径。

(9) 平台必需的 INF 文件 (Platform Profile INF)

使用 Inforce 平台的所必需的 INF 文件。

(10) 硬件、外设列表 (Bill of Materials is a list of Hardware&Peripheral)

BOM 下所记录的是 BIOS image 所支持的硬件设备和外部连接设备的所有集合。

(11) 模块解决方案 (Module Solution)

一组解决方案，这组方案能包含满足所需要支持的硬件设备和外部连接设备的所有驱动文件。

（12）模块设置（Module Settings）

所有的解决方案的 Modules 中，对 Module 的一些作用需要进行相应的设置，这些功能或者端口是以 PCD 的形式存放在模块的 INF 文件中，给这些 PCD 相应的 TRUE/FALSE。

（13）基本功能设置结果（Answers to all configuration settings/question）

对一些启动端口，或者 BIOS Image 在启动过程中是否产生一些信息进行一些设置。

（14）UEFI 的驱动（UEFI Drivers）

加入一些如同 shell、Usb 负载之类的 UEFI 驱动，使 BIOS Image 在启动时，在加载 OS 之前能加载这些应用。

4.2 模块描述类文件设计

EDKII 下对 BIOS Image 的设计和管理主要通过 file.inf、file.dec、file.dsc、file.fdf、file.fd 文件来描述和实现的。其中 file.fd 文件为最终生成的 BIOS Image 文件。

4.2.1 INF 文件设计

EDKII 下的 INF（EDKII Module Information File）文件描述了一个功能模块的属性、它是按什么来编码的、它能提供什么功能、它依赖于哪些模块、模块架构的具体项等。INF 文件是在平台在模块构建编译、链接时，使用到 PCDs（Platform Configuration Database items）的过程中使用的。EDKII 下 INF 模块文件可能在一个 Pkg 的子目录下，头文件定义的模块库必须被放置在 Include 文件夹里。每个模块文件可能有一个或者多个 INF 文件，可以使用 Assembly 工具来生成 Images。为了更加方便和规范的使用 INF 文件，INF 文件的设计如下：

（1）基本定义。包括 INF 文件的版本、驱动名称、文件的 GUID 号、驱动的类型。

（2）包文件信息。借助 EDKII 下的包，应用到的 Protocol、Guid、Ppi 在哪些包里面会有定义。

(3) Protocol、Guid、Ppi 的需要和提供情况。

(4) 该 INF 文件对应的二进制文件的路径。

4.2.2 DEC 文件设计

file.dec 文件：DEC (EDKII Package Declaration File) 文件支持 EDKII 模块构建，该文件用于定义特定信息，将不同的 EDKII 模块之间共享。使用 DEC 文件去解析 DSC 文件和 INF 文件，并且为最后创建 BIOS Image 产生临时文件 AutoGen.c 和 AutoGen.h。DEC 文件描述的内容有定义的模块处于 EDKII 下的哪个目录、这个工作区的环境变量等。每个群体模块的集合必须在一个独立的工作区目录里面。DEC 文件的设计如下：

(1) 基本信息。包括 DEC 文件的版本，DEC 包文件名称，Guid 号。

(2) Protocol、Guid、Ppi 定义。将所有用到这个包里的 Protocol、Guid、Ppi 给予一个初值并定义一个全球唯一标示符。

(3) PCD 的定义。给一些起到特定作用的 PCD 定义其初值或者使用范围。

4.2.3 DSC 文件和 FDF 文件设计

file.dsc、file.fdf 文件：EDKII 创建 BIOS Image 的时候（即 Assembly 过程）必须兼容 EDK 的方法。为了使用 EDKII 的功能模块和 EDKII 的 Assembly 工具，DSC (EDKII Platform Description File) 文件和 FDF (EDKII Flash Definition File) 文件在创建 Image 的过程中必须被使用。EDKII 的工具使用基于文本文件 INI 来描述组件 (Components)、平台 (Platforms) 和固件 (Firmware) 的，但是 EDKII 的 DSC 文件和 EDK 的 DSC 文件一样，采用新的实用程序来处理这些文件。DSC 文件是平台元数据描述文件，这个文件描述了哪些模块拿来被创建，或者被组装成新的模块。FDF 文件用来定义 Firmware 模块目录和二进制镜像文件的布局，还有文件的更新信息和 PCI 选项。DSC 文件的设计如下：

(1) 基本信息。包括平台版本，平台名，平台的 Guid 号，build 完后目录，支持的硬件架构。

(2) 库文件。Build 该平台所需要的原始库文件。

FDF 文件设计如下:

- (1) 基本信息。Image 的名字, ROM 的大小。
- (2) 地址信息。不同模块的寻址信息。
- (3) 驱动文件信息。按照启动顺序排列的二进制文件。

4.3 二进制驱动文件设计

在 EDKII 的体系中, 代码存在着 Source code (源代码) 和 Binary Code (编译后的二进制文件) 这两种存在方式。为了 UEFI 组织为了减少维护代码的花费, 这个花费是与暴露出来的代码数量是成正比的, 采用了许多 Binary Code 的代码的形式, 减少了 Source Code。在由 C 语言设计的一些源代码和 Source INF 文件经过 Binary build 之后, 会被编译成 Binary INF 文件和对应的二进制文件。file.efi 就是编译后生成的一种最常用的二进制文件。下面以 Usb Keyboard 的驱动来描述一下 Usb Keyboard 的二进制驱动的产生过程。

Usb Keyboard 的驱动源代码里 KeyBoard.c、Keyboard.h、UsbKbDxe.inf (Source) 文件。KeyBoard.c 里是用 C 语言格式, 编写的 Usb Keyboard 的驱动源代码; Keyboard.h 里定义了该驱动所用的头文件; UsbKbDxe.inf (Source) 文件按照 INF 文件的格式, 记录了编译这个文件所依赖的一些源文件, 依赖的一些 dec 文件, 和一些 UEFI 库文件以及 Protocol、Ppi、Guid 的使用和提供信息。UEFI 平台下, Source Code build 成 Binary Code 需要借助第三方的 build 工具, 如 VS2005、VS2008 等。使用 Python 脚本语言调用编译工具, 在 build 命令后可以加上一系列的参数来具体的描述 build 的一些操作。-a 表示所选的 cpu 架构, -b 表示是否采用 DEBUG 模式, -t 表示采用哪种编译工具, -p 表示需要编译的 DSC 文件的路径, -m 表示需要编译的 INF 文件路径。在输入命令 build -a IA32 -b DEBUG -t VS2008x86 -p DSCPATH -m INFPATH 后, 系统会调用 VS2008 对 UsbKbDxe.inf (Source) 文件进行编译, 编译完成后, 会生成 UsbKbDxe.efi 文件和 UsbKbDxe.inf (Binary) 文件。UsbKbDxe.efi 是一个二进制文件, 它按照 EFI 固件文件系统规范, 对驱动模块的接口信息进行一个记录。UsbKbDxe.inf (Binary) 文件不会再记录源文件和 DEC 文件的信息, 只保留了 Protocol、Ppi、Guid 的信息。

4.4 RSE 算法设计

Repository Search Engine (RSE) 算法是为了满足一组指定的平台需求，就是确定了硬件和外部设备之后，根据硬件和外部设备的要求，生成一组驱动模块 (Module Solution)，使这组模块能够驱动这些硬件和外设。然后用这些 Module Solution 去得到创建 BIOS Image 的所必需的中间文件---FDF 文件和 DSC 文件。

在设置端口启动信息的时候，有一些 TRUE/FALSE 的问题需要进行设置，这些问题在 Module 模型存放方式是 PCD，在 Find Module Solution 的时候先要对这些 Question 进行设置，即给这些全局的 PCD 赋值。在寻找最终的 Module Solution 的时候，找到的 INF 驱动文件的组合，需要上面的 PCD 的值和这些 Questions 的值保持一致，将是 RSE 的第一步。

第二步，在[BillofMaterials]下，列举出了所有的硬件 (Hardwares) 和外部设备 (Peripherals)。每个 Hardware 的 INF 文件里，都会有一项[HardwareDevice]，该项表示加载这个硬件设备，需要有以后硬件驱动的支持。

每个 Peripheral 的 INF 文件里每个文件都有 [PeripheralDevice]项，这项表示，支持该外部设备，需要以下外设驱动的支持。以 E620AtomProcessor CPU 的 INF 文件为例，如图 4-2 所示。

```
[UserExtensions.Intel."HardwareDevices".IA32]
##
# Processor Devices
IntelCpu (0x06, 0x00, 0x06, 0x02, 0x01) # All E6xx processors have the same CPUID information
Acpi (PNP0C04, 0) # Floating Point Coprocessor
VenHw(0F4724BB-6720-4036-A099-2F532713F4EA) | gIntelTokenSpaceGuid.E6xxSystemManagementModeEnable
##
# Uncore Devices
Pci (0x8086, 0x4115, 0x06, 0x00, 0x00) # Host bridge
Pci (0x8086, 0x4108, 0x03, 0x00, 0x00) # Graphics Device
Pci (0x8086, 0x4108, 0x03, 0x00, 0x00) # Graphics Device
Pci (0x8086, 0x8184, 0x06, 0x04, 0x00) # PCIe Root Port
Pci (0x8086, 0x8185, 0x06, 0x04, 0x00) # PCIe Root Port
Pci (0x8086, 0x8180, 0x06, 0x04, 0x00) # PCIe Root Port
Pci (0x8086, 0x8181, 0x06, 0x04, 0x00) # PCIe Root Port
Pci (0x8086, 0x811B, 0x04, 0x03, 0x00) # HD Audio
Pci (0x8086, 0x8186, 0x06, 0x01, 0x00) # PCI to ISA bridge
# - SMBUS controller base
# - GPIO base
Acpi (PNP0A03, 0) # ACPI info for Host Bridge
Acpi (PNP0A08, 0) # PCIe Root Port 0
Acpi (PNP0A08, 1) # PCIe Root Port 1
Acpi (PNP0A08, 2) # PCIe Root Port 2
Acpi (PNP0A08, 3) # PCIe Root Port 3
Acpi (PNP0003, 0) # I/O Apic
Acpi (PNP0C0F, 0) # PCI Interrupt LNKA
Acpi (PNP0C0F, 1) # PCI Interrupt LNKB
Acpi (PNP0C0F, 2) # PCI Interrupt LNKC
Acpi (PNP0C0F, 3) # PCI Interrupt LNKD
Acpi (PNP0C0F, 4) # PCI Interrupt LNKE
Acpi (PNP0C0F, 5) # PCI Interrupt LNKF
Acpi (PNP0C0F, 6) # PCI Interrupt LNKG
```

图 4-2 E620AtomProcessor.inf 里[HardwareDevice]的相关内容

从图中可以看出，支持 E620 系列的 CPU 需要处理器驱动 (Processor Devices)、非核心的设备驱动和一些 GPIO 口的驱动。只有找到所有能支持这些驱动的 Module 文件，该 CPU 才能正常的工作。图 4-3 是外设 UsbKeyboard.inf 文件里 [PeripheralDevice] 的内容。

```
[Packages]
MdePkg_1E73767F-8F52-4603-AEB4-F29B510B6766_1.03/MdePkg.dec
HardwarePkg_F9D480FC-75A5-4ac3-B0A4-9D16B61C5263_0.10/HardwarePkg.dec

[UserExtensions.Intel."PeripheralDevices"]
UsbClass (0xFFFF, 0xFFFF, 0x03, 0x01, 0x01) # USB Keyboard

[UserExtensions.TianoCore."ExtraFiles"]
UsbKeyboardProperties.uni
UsbKeyboard.png
```

图 4-3 UsbKeyboard.inf 文件里 [PeripheralDevice] 的内容

从图中可以看到支持 Usb 键盘驱动，只需要找到相应的 module，支持 UsbClass (0xFFFF, 0xFFFF, 0x03, 0x01, 0x01)。经过查找，找到了这个 Module: MdeModulePkg_BA0D78D6-2CAF-414b-BD4D-B6762A894288_0.92\Bus\Usb\Usb Kbdxe\UsbKbdxe\RELEASE\UsbKbdxe.inf 可以支持 UsbClass (0xFFFF, 0xFFFF, 0x03, 0x01, 0x01)。所以在最后得到的 Module Solution 中，这个 module 需要被加进去。

除了 [HardwareDevice] 和 [PeripheralDevice] 这两项外，还有 [Guid]、[Ppi]、[Protocol] 这几项。[Guid] 是全局唯一标识符，用 128bit 来标明实体的唯一名称。Guid 的值的格式可以是 8-4-4-12 或者 C 语言数据结构的格式。[Protocol] 是用 Guid 形式表示的一个 API。[Ppi] 是用 Guid 命名的 PEIM-to-PEIM 接口。这些 Guid、Protocol、Ppi 也是对硬件端口的一些设定。

存在着以下几种赋值的情况：

(1) 直接赋值 TRUE/FALSE，如网卡驱动 IScsiServer.inf 文件中 IO 端口赋值 gEfiScsiIoProtocolGuid | TRUE

(2) 该端口依赖于其他个或者多个端口的值，如网卡驱动 IScsiServer.inf 文件中 DiskIO 端口赋值: gEfiDiskIoProtocolGuid | ((gHardwarePkgTokenSpaceGuid.OsBootRecoverySupport &

$(0x01) \neq 0x00) \text{ AND } (g\text{HardwarePkgTokenSpaceGuid.BlockDeviceProtocolSupport} \geq 1) \text{ AND } (g\text{HardwarePkgTokenSpaceGuid.BlockDeviceProtocolSupport} \leq 2)$

(3) 在 Guid、Protocol、Ppi 后面有些会加上 PRODUCE 或者 CONSUMES 标识。如果在 1.inf 文件的里 [Protocol] 下存在 WsOriginal_Protocol_1 ##PRODUCES, 表示在使用到 1.inf 文件驱动时, 可以同时产生这个 WsOriginal_Protocol_1 这个 Protocol; 如果在 2.inf 文件的里 [Protocol] 下存在 WsOriginal_Protocol_1 ##CONSUMES, 表示在使用到 2.inf 文件驱动时, 需要消费 WsOriginal_Protocol_1 这个 Protocol, 这就需要在这组 Module Solutions 中存在一个 INF 文件去产生一个 WsOriginal_Protocol_1。也就是说, 如果 2.inf 作为驱动文件在 Module Solutions 中是必需存在的, 那么 1.inf 作为 2.inf 的 Protocol 提供者, 也必需存在于这组 Module Solutions 中。

寻找 Module Solution 的过程, 就是将所有需要加载的硬件和外设驱动上的 [HardwareDevice]、[PeripheralDevice]、[Guid]、[Protocol]、[Ppi] 的信息都分别集中存在一个 list 上, 组成一个 [AllHardwareDevices]、[AllPeripheralDevices]、[AllGuids]、[AllProtocols]、[AllPpis] 的组合, 然后去寻找相应的 Module 组合。

具体的实现是:

(1) 在所设计的 BIM 文件的 [BillOfMaterials] 里拿到所有的硬件设备和外部设备。得到 HardwareBOM 和 PeripheralBOM。

(2) 由于 EDKII 的 Pkg 包 Repository 很大, 在得到 HardwareBOM 和 PeripheralBOM 后, 在 Repository 里循环遍历一边, 找到所有支持这些 BOM 的 module 组合 Module1s。

(3) 根据 Guid 的值或者所得到的表达式的值, 再在 Repository 里遍历一边, 得到满足 Guid 条件的 Module2s 集合。

(4) 同样满足 Protocol 和 Ppi 条件的 Module 组合分别为 Module3s, Module4s。

(5) 将 Module1s, Module2s, Module3s, Module4s 融合到一起, 去掉里面重复的 Module, 这样子得到的 Module solution, 就将是满足 BIM 文件硬件设备和外部设备的所有 Modules。

4.5 本章小结

本章主要为 BIOS Image 设计出了对应的管理文件，包括二进制管理的 BIM 文件，和鼠标键盘等外设的 INF 文件。并设计 RSE 算法，为后面 UEFI BIOS 实现做好铺垫。

5 UEFI BIOS 实现与测试

UEFI BIOS 的实现，主要是在 VS2008 编译器的命令行平台，对 BIM 文件进行解析，并调用 RSE 算法，找到所有的模块解决方案，最终将得到所有的模块通过 Build 工具，生成最后的 Image 文件。将生成的 Image 连接到硬件平台上，接上显示器或者通过串口终端就可以看到加载操作系统的情况。

5.1 UEFI BIOS 实现

5.1.1 BIM 文件实现

根据上一章的设计，在 BIM 文件开始，设定一个[Globals]的项，里面需要记录文件的 type，这个 type 使用的是 Guid 格式，全局唯一的标识符。由于我们采用的 Repository 版本为 1.0，所以 FileVersion 为 1.0。ReadOnly 可设置成只读（TRUE）、或者可修改（FALSE），我们由于会根据实际测试情况进行修改 BIM 文件，所以 ReadOnly 设置成 FALSE。

设计一个[Paths]项，这个项的所有路径都设置成相对路径，在 Build 的过程中，加入工作区的路径就可以找到相应的路径，这样便于这些路径的统一更改。定义 Image 文件的输出路径 OutputPath=OUTPUT/，以前 Build 过程 image 产生的文件存放在 HistoryPath=History/、Build 过程中产生临时文件 DSC 和 FDF 的路径 Temp=temp/。

[Platform]项是记录的平台一些信息，由于本文使用的是 Inforce9450 的硬件平台平台，所以 FirmwareVendo=Inforce 9450 Reference Platform，FirmwareBuildDate 和 FirmwareModificationDate 是 Build 过程中，产生的时间信息，在 Build 完成后，会自动记录。平台必需的 INF 文件被记录在 CrownBayPkg 下，所以 PlatformProfileByINF=CrownBayPkg_46C1F476-A85E-49a8-B258-DD4396B87FEF_0.2\PlatformProfile\Inforce9450.inf。Firmware 的版本号 FirmwareVersion=1.0，image 的创建数量 FirmwareBuildNumber=1。

[Tool]项有一个 ToolPath 制定使用默认的 Build 和 Assembly 的目录 ToolPath=

华中科技大学硕士学位论文

ToolPath = C:\Program Files (x86)\Intel\UEFI Binary Management Suite\0.1.0.57569\Bin\Win32。在这个目录下会有 EDKII 平台已经打包的 build.exe 和 assembly.exe 文件。

[Repository]项记录的是, EDKII 的 Repository 包的路径, 这个路径也是下面 Module、硬件和外部设备的工作区。Module、硬件和外设在 BIM 文件的记录方式是相对路径, 加上 Repository 的路径就构成了一个绝对路径。把这个路径设置成 RepositoryPath = C:\Users\Public\Repository\UBMS_Pilot。

[BillOfMaterials]这项所记录的就是所有硬件和外设, 由于 Inforce9450 的硬件平台, 已经确定了 CPU, 内存, 闪存等一些硬件型号, 所以在设计的时候, 硬件设备的选择, 可以根据 Inforce9450 的型号来进行确定。外设在为 HP-UX11iV3 操作系统设计的时候, 为了加快启动速度, 可以进行最简化的定制。USB 和 SATA 的键盘、鼠标、HD、CD 都选择一个, 这里就选择比较常用的 USB 型号的外设。串口的控制连接着屏幕, 也可以从中打印一些输出信息, 所以必须加入 UartHeader 和 UartConsole 两个串口的驱动。显卡等设备使用时需要用到 PCI 的插槽, 所以 PCI 插槽驱动必须加入到 BIM 文件中。UEFI 启动的时候, 就支持网络操作, Network 驱动也将会被加入到 BIM 文件中。

[GlobalSettings]这一项是对启动时的一些界面进行的设置, 这些需要给相应的 PCD 的 Guid 进行赋值。例如:

gEfiHpBootSupportTokenSpaceGuid.PcdHpBootSupportEnabled = TRUE 表示进入 UEFI Shell 界面后, 可以加载 HP 的操作系统。

gEfiUefiShellBootSupportTokenSpaceGuid.PcdUefiShellBootSupportEnabled = TRUE 表示支持 UEFI 在启动时可以进入 Shell 界面。

gEfiMdeModulePkgTokenSpaceGuid.PcdVideoHorizontalResolution = 0x00000320

gEfiMdeModulePkgTokenSpaceGuid.PcdVideoVerticalResolution = 0x00000258

表示进入 UEFI 界面后, 水平和垂直的高度。采用的是 16 进制的表示方法。

[ModuleSolution]在 RSE (Repository Search engine) 后, 会把所有的 Module solution 加入到对应的项目中。

华中科技大学硕士学位论文

综合上面的设计情况，最终得到 BIM 文件 Project.bim 如图 5-1 和 5-2 所示。

```
Project.bim
1  [Globals]
2  FileType = 1C560B40-5939-42b6-8CBB-AF72BEA7FC3E
3  FileVersion = 1.0
4  FileInstance = 35cb8545-519e-4fb8-b381-46fa7cba99e3
5  Name = Project
6  ReadOnly = TRUE
7
8  [Paths]
9  OutputPath = OUTFUT/
10 HistoryPath = History/
11 TempPath = Temp/
12
13 [Platform]
14 FirmwareVendor = Infineon 9450 Reference Platform
15 FirmwareBuildDate = Fri DEC 20 19:34:05 2013
16 FirmwareModificationDate = Fri DEC 20 14:39:24 2013
17 PlatformProfileByInf = CrownBayPkg_46C1F476-A85E-49a8-B258-DD4396B87FEF_0.2\PlatformProfile\Inforce9450.inf
18 FirmwareVersion = 1.0
19 FirmwareBuildNumber = 1
20
21 [Tool]
22 ToolPath = C:\Program Files (x86)\Intel\UEFI Binary Management Suite\0.1.0.57569\Bin\Win32
23
24 [Repository]
25 RepositoryPath = C:\Users\Public\Repository\UBMS_Pilot
26
27 [ModuleSettings]
28
29 [GlobalSettings]
```

图 5-1 工程的 BIM 文件（1）

```
29 [GlobalSettings]
30 gEfiHlpBootSupportTokenSpaceGuid.PodHpBootSupportEnabled = TRUE
31 gEfiBinaryDistributionSupportPkgTokenSpaceGuid.PodPreferSizeOrSpeed = FALSE
32 gEfiUefiShellBootSupportTokenSpaceGuid.PodUefiShellBootSupportEnabled = TRUE
33 gHardwarePkgTokenSpaceGuid.UnicodeAnnotationString = L""
34 gHardwarePkgTokenSpaceGuid.OsBootRecoverySupport = 0x01
35 gHardwarePkgTokenSpaceGuid.BlockDeviceProtocolSupport = 0x02
36 gHardwarePkgTokenSpaceGuid.EfiBlockIoSecurityCommandSupport = FALSE
37 gEfiTunnelCreekTokenSpaceGuid.PodTcdDeviceEnables = 0x0000006f
38 gEfiMdeModulePkgTokenSpaceGuid.PodVideoHorizontalResolution = 0x000000320
39 gEfiMdeModulePkgTokenSpaceGuid.PodVideoVerticalResolution = 0x000000258
40 gEfiMdePkgTokenSpaceGuid.PodUefiVariableDefaultPlatformLang = {0x65, 0x6e, 0x2d, 0x55, 0x53, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}
41
42 [DefaultGlobalSettings]
43
44 [BillOfMaterials]
45 BomModuleInf_1 = NuvotonHardwarePkg_F1658FE8-7570-45b2-BAAD-305CF5CCECF4_0.10\W83627DHG-PT\W83627dhg-pt.inf
46 BomModuleInf_2 = IntelHardwarePkg_64260E85-B9F0-4fb7-BE5E-0A64607E903A_0.10\Cpu\E6xx\E680AtomProcessor.inf
47 BomModuleInf_3 = SstHardwarePkg_A5037D8A-B134-4312-989D-90CA7891BB94_0.10\Sst25vf016b\Sst25vf016bSpiFlash.inf
48 BomModuleInf_4 = AsixHardwarePkg_2C391FDA-21A7-4fbb-8D3C-FC74DC6BD64_0.10\UsbAx88772BNet\UsbAx88772BNet.inf
49 BomModuleInf_5 = HardwarePkg_F9D480FC-75A5-4ac3-B0A4-9D16B61CS263_0.10\Hardware\DDR2Memory\DDR2Memory.inf
50 BomModuleInf_6 = IntelHardwarePkg_64260E85-B9F0-4fb7-BE5E-0A64607E903A_0.10\Chipset\EG20T\Eg20t.inf
51 BomModuleInf_7 = HardwarePkg_F9D480FC-75A5-4ac3-B0A4-9D16B61CS263_0.10\Peripherals\SdMmc\SdMmc.inf
52 BomModuleInf_8 = HardwarePkg_F9D480FC-75A5-4ac3-B0A4-9D16B61CS263_0.10\Peripherals\Usb\UsbHardDrive\UsbHardDrive.inf
53 BomModuleInf_9 = HardwarePkg_F9D480FC-75A5-4ac3-B0A4-9D16B61CS263_0.10\Peripherals\Usb\UsbMouse\UsbMouse.inf
54 BomModuleInf_10 = HardwarePkg_F9D480FC-75A5-4ac3-B0A4-9D16B61CS263_0.10\Peripherals\Usb\UsbCdDvdDrive\UsbCdDvdDrive.inf
55 BomModuleInf_11 = HardwarePkg_F9D480FC-75A5-4ac3-B0A4-9D16B61CS263_0.10\Peripherals\PciAdapter\PciNetworkAdapter\PciNetworkAdapter.inf
56 BomModuleInf_12 = HardwarePkg_F9D480FC-75A5-4ac3-B0A4-9D16B61CS263_0.10\Peripherals\Usb\UsbFlashDrive\UsbFlashDrive.inf
57 BomModuleInf_13 = HardwarePkg_F9D480FC-75A5-4ac3-B0A4-9D16B61CS263_0.10\Peripherals\Uart\UartConsole\UartConsole.inf
58 BomModuleInf_14 = HardwarePkg_F9D480FC-75A5-4ac3-B0A4-9D16B61CS263_0.10\Peripherals\Usb\UsbKeyboard\UsbKeyboard.inf
59 BomModuleInf_15 = HardwarePkg_F9D480FC-75A5-4ac3-B0A4-9D16B61CS263_0.10\Peripherals\Monitor\Monitor.inf
```

图 5-2 工程的 BIM 文件（2）

5.1.2 模块描述类文件实现

由于在 BIM 文件中加入了 USB 口键盘、USB 口鼠标、DVD、串口等，所实现的驱动文件很多。键盘作为一个典型的计算机设备，下面以 UsbKbDxe.inf 为例论述 INF 文件的设计。

在[Defines]项里，我们用到的 INF 版本为 0x00010005，文件名设计为 UsbKbDxe，

利用 Guid 生成器生成一个 Guid 号。该文件的类型为 UEFI_DRIVER, 本文还涉及到 UEFI_Application 类型。

在[Sources]项里, 由于 USB 的键盘用到的用 C 语言驱动设计在 EfiKey.c、EfiKey.h、KeyBoard.c、ComponentName.c、Keyboard.h 里, 需要在这个项目下列出这些文件, 为 Source Build 时提供源文件。

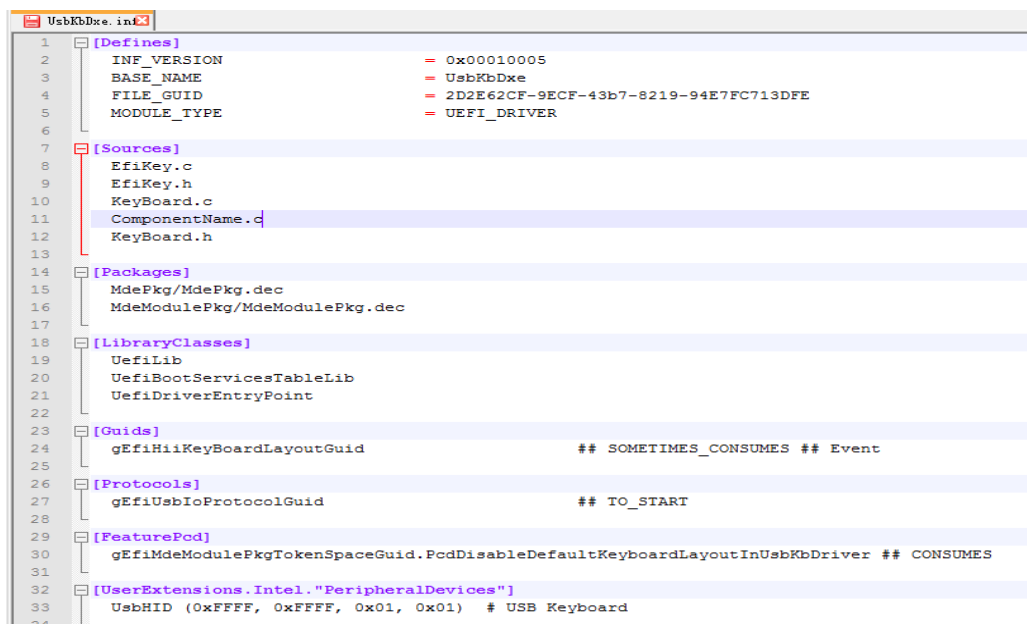
在[Packages]项里, 列举出了 Protocol、Guid 所定义在的 DEC 文件。这些协议都定义在 MdePkg.dec 文件和 MdeModule.dec 文件中。

在[LibraryClasses]项里需要指出, USB 键盘需要使用的库文件, 包括 UefiLib、UefiBootServicesTableLib、UefiDriverEntryPoint 三个库文件。

[Guids]、[Protocols]和[Pcd]里列举出该文件需要消耗的 Guid 或者 Protocol, 能够产生哪些 Guid 和 Protocol。使得在寻找模块解决方案的时候, 能够根据消耗和产生情况找到最合适的模块解决方案。USB 键盘的 INF 文件需要消耗 gEfiHiiKeyboardLayoutGuid 这个 Guid, 能够产生 gEfiUsbIoProtocolGuid 这个 Protocol 协议。

[UserExtensions.Intel."PeripheralDevices"]项里记录这个外设的一个特定 ID。USB 键盘的 ID 为: UsbHID (0xFFFF, 0xFFFF, 0x01, 0x01)。

设计好的 UsbKbDxe.inf 文件如图 5-3 所示



```
1 [Defines]
2   INF_VERSION           = 0x00010005
3   BASE_NAME             = UsbKbDxe
4   FILE_GUID             = 2D2E62CF-9ECF-43b7-8219-94E7FC713DFE
5   MODULE_TYPE           = UEFI_DRIVER
6
7 [Sources]
8   EfiKey.c
9   EfiKey.h
10  Keyboard.c
11  ComponentName.c
12  Keyboard.h
13
14 [Packages]
15   MdePkg/MdePkg.dec
16   MdeModulePkg/MdeModulePkg.dec
17
18 [LibraryClasses]
19   UefiLib
20   UefiBootServicesTableLib
21   UefiDriverEntryPoint
22
23 [Guids]
24   gEfiHiiKeyboardLayoutGuid  ## SOMETIMES_CONSUMES ## Event
25
26 [Protocols]
27   gEfiUsbIoProtocolGuid      ## TO_START
28
29 [FeaturePcd]
30   gEfiMdeModulePkgTokenSpaceGuid.PcdDisableDefaultKeyboardLayoutInUsbKbDriver ## CONSUMES
31
32 [UserExtensions.Intel."PeripheralDevices"]
33   UsbHID (0xFFFF, 0xFFFF, 0x01, 0x01) # USB Keyboard
34
```

图 5-3 UsbKbDxe.inf 文件

华中科技大学硕士学位论文

DEC 文件、DSC 文件和 FDF 文件按照上一章的设计，相应得到的结果如图 5-4，图 5-5，图 5-6 所示。

```
MdeModulePkg.dec
1 [Defines]
2   DEC_SPECIFICATION           = 0x00010005
3   PACKAGE_NAME                = MdeModulePkg
4   PACKAGE_GUID                = BA0D78D6-2CAF-414b-BD4D-B6762A894288
5   PACKAGE_VERSION             = 0.92
6
7 [Includes]
8   Include
9
10 [LibraryClasses]
11   IpIoLib|Include/Library/IpIoLib.h
12   NetLib|Include/Library/NetLib.h
13   UdpIoLib|Include/Library/UdpIoLib.h
14   TcpIoLib|Include/Library/TcpIoLib.h
15
16 [Guids]
17   gEfiMdeModulePkgTokenSpaceGuid = { 0xA1AFF049, 0xFDEB, 0x442a, { 0xB3, 0x20, 0x13, 0xAB, 0x4C, 0xB7, 0x2B, 0xBC }}
18   gPcdDataBaseHobGuid           = { 0xEA296D92, 0x0B69, 0x423C, { 0x8C, 0x28, 0x33, 0xB4, 0xE0, 0xA9, 0x12, 0x68 }}
19   gEfiIfrTianoGuid             = { 0xf0b1735, 0x87a0, 0x4193, { 0xb2, 0x66, 0x53, 0x8c, 0x38, 0xaf, 0x48, 0xce }}
20
21 [Ppis]
22   gPeiAtaControllerPpiGuid      = { 0xa45e60d1, 0xc719, 0x44aa, { 0xb0, 0x7a, 0xaa, 0x77, 0x7f, 0x85, 0x90, 0x6d }}
23   gPeiUsbHostControllerPpiGuid = { 0x652B38A9, 0x77F4, 0x453F, { 0x89, 0xD5, 0xE7, 0xBD, 0xC3, 0x52, 0xFC, 0x53 }}
24   gPeiUsb2HostControllerPpiGuid = { 0xfedd6305, 0xe2d7, 0x4ed5, { 0x9f, 0xaa, 0xda, 0x8, 0xe, 0x33, 0x6c, 0x22 }}
25
26 [PcdsFeatureFlag]
27   gEfiMdeModulePkgTokenSpaceGuid.PcdSupportUpdateCapsuleReset|FALSE|BOOLEAN|0x0001001d
28   gEfiMdeModulePkgTokenSpaceGuid.PcdPeiFullPcdDatabaseEnable|TRUE|BOOLEAN|0x00010020
29   gEfiMdeModulePkgTokenSpaceGuid.PcdDevicePathSupportDevicePathToText|TRUE|BOOLEAN|0x00010037
```

图 5-4 MdeModule.dec 文件

```
Assembly.dsc
1 # DO NOT EDIT
2 # FILE auto-generated
3
4 [Defines]
5   PLATFORM_NAME                = Assembly
6   PLATFORM_GUID                = 35cb8545-519e-4fb8-b381-46fa7cba99e3
7   PLATFORM_VERSION             = 1.0
8   DSC_SPECIFICATION            = 0x00010016
9   SUPPORTED_ARCHITECTURES      = IA32
10  BUILD_TARGETS                 = DEBUG|RELEASE
11  SKUID_IDENTIFIER              = DEFAULT
12  OUTPUT_DIRECTORY              = C:\Users\tiano\AppData\Local\Temp\USMS
13  FIX_LOAD_TOP_MEMORY_ADDRESS   = 0
14  VPD_TOOL_GUID                 = 8C3D856A-9BE6-468E-850A-24F7A8D38E08
15
16 #####
17 #
18 # SKU Identification section - list of all SKU IDs supported by this Platform.
19 #
20 #
21 #####
22 [Skuids]
23   0|DEFAULT                    # The entry: 0|DEFAULT is reserved and always required.
24
25 [PcdsDynamicExHii]
26   gEfiIntelFrameworkModulePkgTokenSpaceGuid.PcdPlatformBootTimeOut|L"Timeout"|gEfiGlobalVariableGuid|0x0|0x0001
27
28
29 [PcdsDynamicExVpd]
30
31
32 [PcdsDynamicExDefault]
33   gEfiMdeModulePkgTokenSpaceGuid.PcdConOutRow|0x00000019
34   gEfiMdeModulePkgTokenSpaceGuid.PcdConOutColumn|0x00000050
35   gEfiMdeModulePkgTokenSpaceGuid.PcdVideoHorizontalResolution|0x000000320
36   gEfiMdeModulePkgTokenSpaceGuid.PcdVideoVerticalResolution|0x000000258
37
```

图 5-5 Assembly.dsc 文件

```

1  [FD.Assembly]
2  BaseAddress = 0xFFC00000|gIntelE6xxRuTokenSpaceGuid.PcdFlashAreaBase
3  Size = 0x00400000|gIntelE6xxRuTokenSpaceGuid.PcdFlashAreaSize
4  ErasePolarity = 1
5  BlockSize = 0x00010000
6  NumBlocks = 0x00000040
7
8  SET gIntelE6xxRuTokenSpaceGuid.PcdFlashNvStorageBase = 0xFFFF70000
9  SET gIntelE6xxRuTokenSpaceGuid.PcdFlashNvStorageSize = 0x00040000
10 SET gIntelE6xxRuTokenSpaceGuid.PcdBiosImageBase = 0xFFC00000
11 SET gIntelE6xxRuTokenSpaceGuid.PcdBiosImageSize = 0x00400000
12
13 0x00000000|0x00300000
14 gCrownBayTokenSpaceGuid.PcdFlashPayloadBase|gCrownBayTokenSpaceGuid.PcdFlashPayloadSize
15 FV = PAY_LOAD
16
17 0x00300000|0x00060000
18 gCrownBayTokenSpaceGuid.PcdFlashFvMainBase|gCrownBayTokenSpaceGuid.PcdFlashFvMainSize
19 FV = FVMAIN_COMPACT
20
21 0x00360000|0x00010000
22 gCrownBayTokenSpaceGuid.PcdFlashFvRecovery2Base|gCrownBayTokenSpaceGuid.PcdFlashFvRecovery2Size
23 FV = FVRECOVERY2
24
25 0x00370000|0x0001E000
26 gEfiMdeModulePkgTokenSpaceGuid.PcdFlashNvStorageVariableBase|gEfiMdeModulePkgTokenSpaceGuid.PcdFlashNvStorageVariableSize
27 DATA = {
28  ## This is the EFI_FIRMWARE_VOLUME_HEADER
29  # ZeroVector []
30  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
31  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
32  # FileSystemGuid: gEfiSystemNvDataFvGuid =
33  # ( 0xFFFF12B8D, 0x7696, 0x4C8B, { 0xA9, 0x85, 0x27, 0x47, 0x07, 0x5B, 0x4F, 0x50 } )
34  0x8D, 0x2B, 0xF1, 0xFF, 0x96, 0x76, 0x8B, 0x4C,
35  0xA9, 0x85, 0x27, 0x47, 0x07, 0x5B, 0x4F, 0x50,
36  # FvLength
37  0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00,
38  #Signature "_FVH" #Attributes

```

图 5-6 Assembly.fdf 文件

5.1.3 模块解决方案结果

根据得到的 Project.bim 文件，在 Eclipse+Python 的开发环境里，调用 FindModuleSolution (self) 函数，在包含所有驱动文件的 Repository 里寻找模块解决方案，按照设计的 RSE 算法，最终得到的模块解决方案如图 5-7 所示。得到这个 Module Solution 就可以去生成 BIOS Image 了。

```

27 [ModuleSettings]
28 ModuleInf_1 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\AcpiPlatformDxe\AcpiPlatformDxe\RELEASE\AcpiPlatform.inf
29 ModuleInf_2 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\AcpiSupportOnAcpiTableAndAcpiSdtThunk\AcpiSupportOnAcpiTableAndAcpiSdtThunk\RELEASE\AcpiSupp
30 ModuleInf_3 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\Acpi\AcpiTableDxe\AcpiTableDxe\RELEASE\AcpiTableDxe.inf
31 ModuleInf_4 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\AcpiTablesDxe\AcpiTablesDxe\RELEASE\AcpiTablesDxe.inf
32 ModuleInf_5 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Bus\Ata\AtaAtapiPassThru\AtaAtapiPassThru\RELEASE\AtaAtapiPassThruDxe.inf
33 ModuleInf_6 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Bus\Ata\AtaBusDxe\AtaBusDxe\RELEASE\AtaBusDxe.inf
34 ModuleInf_7 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\Bds\Bds\RELEASE\Bds.inf
35 ModuleInf_8 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\CapsulePei\CapsulePei\RELEASE\CapsulePei.inf
36 ModuleInf_9 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\CapsuleRuntimeDxe\CapsuleRuntimeDxe\RELEASE\CapsuleRuntimeDxe.inf
37 ModuleInf_10 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\Disk\CdExpressPei\CdExpressPei\RELEASE\CdExpressPei.inf
38 ModuleInf_11 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\Console\ConPlatformDxe\ConPlatformDxe\RELEASE\ConPlatformDxe.inf
39 ModuleInf_12 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\Console\ConSplitterDxe\ConSplitterDxe\RELEASE\ConSplitterDxe.inf
40 ModuleInf_13 = IA32FamilyCpuPkg_7dbe088f-2e1a-475c-b006-55632c2a5489_0.4\CpuArchDxe\CpuArchDxe\RELEASE\CpuArchDxe.inf
41 ModuleInf_14 = UefiCpuPkg_2171df9b-0d39-45aa-ac37-2de190010d23_0.2\CpuIo2Dxe\CpuIo2Dxe\RELEASE\CpuIo2Dxe.inf
42 ModuleInf_15 = UefiCpuPkg_2171df9b-0d39-45aa-ac37-2de190010d23_0.2\CpuIo2Smm\CpuIo2Smm\RELEASE\CpuIo2Smm.inf
43 ModuleInf_16 = UefiCpuPkg_2171df9b-0d39-45aa-ac37-2de190010d23_0.2\CpuIoPei\CpuIoPei\RELEASE\CpuIoPei.inf
44 ModuleInf_17 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\Restricted\CpuMicrocode\CpuMicrocode\RELEASE\CpuMicrocode.inf
45 ModuleInf_18 = IA32FamilyCpuPkg_7dbe088f-2e1a-475c-b006-55632c2a5489_0.4\CpuMpDxe\CpuMpDxe\RELEASE\CpuMpDxe.inf
46 ModuleInf_19 = IA32FamilyCpuPkg_7dbe088f-2e1a-475c-b006-55632c2a5489_0.4\CpuPei\CpuPei\RELEASE\CpuPei.inf
47 ModuleInf_20 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\IncompatiblePciDeviceSupportDxe\IncompatiblePciDeviceSupportDxe\RELEASE\CrownBayIncompatibl
48 ModuleInf_21 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\StatusCodeHandlerPei\StatusCodeHandlerPei\RELEASE\CrownBayStatusCodeHandlerPei.inf
49 ModuleInf_22 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\DevicePathDxe\DevicePathDxe\RELEASE\DevicePathDxe.inf
50 ModuleInf_23 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\Disk\DiskIoDxe\DiskIoDxe\RELEASE\DiskIoDxe.inf
51 ModuleInf_24 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Core\Dxe\DxeMain\RELEASE\DxeCore.inf
52 ModuleInf_25 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Core\DxeIplPeim\DxeIpl\RELEASE\DxeIpl.inf
53 ModuleInf_26 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\EbcDxe\EbcDxe\RELEASE\EbcDxe.inf
54 ModuleInf_27 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Bus\Pci\EhciDxe\EhciDxe\RELEASE\EhciDxe.inf
55 ModuleInf_28 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Bus\Pci\EhciPei\EhciPei\RELEASE\EhciPei.inf
56 ModuleInf_29 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\Disk\UnicodeCollation\EnglishDxe\EnglishDxe\RELEASE\EnglishDxe.inf
57 ModuleInf_30 = FatPkg_8EA68A2C-99CB-4332-85C6-DD5864EAA674_0.2\EnhancedFatDxe\Fat\RELEASE\Fat.inf
58 ModuleInf_31 = FatPkg_8EA68A2C-99CB-4332-85C6-DD5864EAA674_0.2\FatPei\FatPei\RELEASE\FatPei.inf
59 ModuleInf_32 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\FaultTolerantWriteDxe\FaultTolerantWriteDxe\RELEASE\FaultTolerantWriteDxe.inf
60 ModuleInf_33 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\FvbRuntimeDxe\FvbRuntimeDxe\RELEASE\FvbRuntimeDxe.inf
61 ModuleInf_34 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\FvbRuntimeDxe\FvbSmm\RELEASE\FvbSmm.inf
62 ModuleInf_35 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\FvbRuntimeDxe\FvbSmmDxe\RELEASE\FvbSmmDxe.inf
63 ModuleInf_36 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\Console\GraphicsConsoleDxe\GraphicsConsoleDxe\RELEASE\GraphicsConsoleDxe.inf
64 ModuleInf_37 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\HiDatabaseDxe\HiDatabaseDxe\RELEASE\HiDatabaseDxe.inf
65 ModuleInf_38 = TopcliffPkg_2B827A37-D5D8-4053-9925-92CA344EE965_0.1\IohSerialDxe\SerialDxe\RELEASE\IohSerialDxe.inf
66 ModuleInf_39 = IntelFrameworkModulePkg_88894582-7553-4822-B484-624E24B6DEC7_0.92\Bus\Isa\IsaBusDxe\IsaBusDxe\RELEASE\IsaBusDxe.inf

```



```

67 ModuleInf_40 = IntelFrameworkModulePkg_88894582-7553-4822-B484-624E24B6DECF_0.92\Bus\Isa\IsaSerialDxe\IsaSerialDxe\RELEASE\IsaSerialDxe.inf
68 ModuleInf_41 = PcAtChipsetPkg_B728689A-52D3-4B8C-AE89-2CE5514CC6DC_0.2\6259InterruptControllerDxe\8259\RELEASE\Legacy8259.inf
69 ModuleInf_42 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\LoadFileOnFv2\LoadFileOnFv2\RELEASE\LoadFileOnFv2.inf
70 ModuleInf_43 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Logo\Logo.inf
71 ModuleInf_44 = TunnelCreekPkg_28DEC17-6C75-448F-87DC-BDE4BD579919_0.2\MemoryInitPei\MemoryInitPei\RELEASE\MemoryInitPei.inf
72 ModuleInf_45 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\Metronome\Metronome\RELEASE\Metronome.inf
73 ModuleInf_46 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\MonotonicCounterRuntimeDxe\MonotonicCounterRuntimeDxe\RELEASE\Monoton
74 ModuleInf_47 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\MemoryTest\NullMemoryTestDxe\RELEASE\NullMemoryTest
75 ModuleInf_48 = TopcliffPkg_2B827A37-D5D8-4053-9925-92CA344EE965_0.1\OhciDxe\OhciDxe\RELEASE\OhciDxe.inf
76 ModuleInf_49 = TopcliffPkg_2B827A37-D5D8-4053-9925-92CA344EE965_0.1\OhciPei\OhciPei\RELEASE\OhciPei.inf
77 ModuleInf_50 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\Disk\PartitionDxe\PartitionDxe\RELEASE\PartitionDxe.inf
78 ModuleInf_51 = PcAtChipsetPkg_B728689A-52D3-4B8C-AE89-2CE5514CC6DC_0.2\PcatRealTimeClockRuntimeDxe\PcatRealTimeClockRuntimeDxe\RELEASE\PcRtc.inf
79 ModuleInf_52 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\PcatSingleSegmentPciCfg2Pei\PcatSingleSegmentPciCfg2Pei\RELEASE\PcatS
80 ModuleInf_53 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\PCD\Dxe\Fcd\RELEASE\FcdDxe.inf
81 ModuleInf_54 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\PCD\Pei\Fcd\RELEASE\FcdPeim.inf
82 ModuleInf_55 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Bus\Pci\PciBusDxe\PciBusDxe\RELEASE\PciBusDxe.inf
83 ModuleInf_56 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\PciPlatformDxe\PciPlatformDxe\RELEASE\PciPlatformDxe.inf
84 ModuleInf_57 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Core\Pei\PeiMain\RELEASE\PeiCore.inf
85 ModuleInf_58 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\Variable\Pei\VariablePei\RELEASE\PeiVariable.inf
86 ModuleInf_59 = IA32FamilyCpuPkg_7dbe088f-2e1a-475c-b006-55632c2a5489_0.4\PiSmmCommunication\PiSmmCommunicationSmm\RELEASE\PiSmmCommunicationSmm.inf
87 ModuleInf_60 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Core\PiSmmCore\PiSmmCore\RELEASE\PiSmmCore.inf
88 ModuleInf_61 = IA32FamilyCpuPkg_7dbe088f-2e1a-475c-b006-55632c2a5489_0.4\PiSmmCpuDxeSmm\PiSmmCpuDxeSmm\RELEASE\PiSmmCpuDxeSmm.inf
89 ModuleInf_62 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Core\PiSmmCore\PiSmmIpi\RELEASE\PiSmmIpi.inf
90 ModuleInf_63 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\PlatformPei\PlatformPei\RELEASE\PlatformPeim.inf
91 ModuleInf_64 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\SmmPowerManagement\AcpiTables\PowerManagementAcpiTablesDxe\RELEASE\PowerManagemen
92 ModuleInf_65 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\ReportStatusCodeRouter\Pei\ReportStatusCodeRouterPei\RELEASE\ReportSt
93 ModuleInf_66 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\ReportStatusCodeRouter\RuntimeDxe\ReportStatusCodeRouterRuntimeDxe\RE
94 ModuleInf_67 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\ReportStatusCodeRouter\Smm\ReportStatusCodeRouterSmm\RELEASE\ReportSt
95 ModuleInf_68 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Core\RuntimeDxe\RuntimeDxe\RELEASE\RuntimeDxe.inf
96 ModuleInf_69 = TopcliffPkg_2B827A37-D5D8-4053-9925-92CA344EE965_0.1\SDControllerDxe\SDControllerDxe\RELEASE\SDController.inf
97 ModuleInf_70 = TopcliffPkg_2B827A37-D5D8-4053-9925-92CA344EE965_0.1\SDMediaDeviceDxe\SDMediaDeviceDxe\RELEASE\SDMediaDevice.inf
98 ModuleInf_71 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Bus\Scsi\ScsiBusDxe\ScsiBusDxe\RELEASE\ScsiBus.inf
99 ModuleInf_72 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Bus\Scsi\ScsiDiskDxe\ScsiDiskDxe\RELEASE\ScsiDisk.inf
100 ModuleInf_73 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\SecCore\SecCore\RELEASE\SecCore.inf
101 ModuleInf_74 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\SecurityStubDxe\SecurityStubDxe\RELEASE\SecurityStubDxe.inf
102 ModuleInf_75 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\SetupDxe\SetupDxe\RELEASE\SetupDxe.inf
103 ModuleInf_76 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\SmbiosDxe\SmbiosDxe\RELEASE\SmbiosDxe.inf
104 ModuleInf_77 = TunnelCreekPkg_28DEC17-6C75-448F-87DC-BDE4BD579919_0.2\SmmAccessDxe\SmmAccessDxe\RELEASE\SmmAccessDxe.inf
105 ModuleInf_78 = TunnelCreekPkg_28DEC17-6C75-448F-87DC-BDE4BD579919_0.2\SmmControlDxe\SmmControlDxe\RELEASE\SmmControlDxe.inf
106 ModuleInf_79 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\SmmPlatformHandler\SmmPlatformHandler\RELEASE\SmmPlatformHandler.inf
107 ModuleInf_80 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\SmmPowerManagement\SmmPowerManagement\RELEASE\SmmPowerManagement.inf
108 ModuleInf_81 = TunnelCreekPkg_28DEC17-6C75-448F-87DC-BDE4BD579919_0.2\SpiRuntimeDxe\SpiRuntimeDxe\RELEASE\SpiRuntimeDxe.inf
109 ModuleInf_82 = TunnelCreekPkg_28DEC17-6C75-448F-87DC-BDE4BD579919_0.2\SpiRuntimeDxe\SpiSmm\RELEASE\SpiSmm.inf
110 ModuleInf_83 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\SpiDeviceDxe\SpiSstDeviceDxe\RELEASE\SpiSstDeviceDxe.inf
111 ModuleInf_84 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\SpiDeviceDxe\SpiSstDeviceSmm\RELEASE\SpiSstDeviceSmm.inf
112 ModuleInf_85 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\StatusCodeHandler\RuntimeDxe\StatusCodeHandlerRuntimeDxe\RELEASE\Status
113 ModuleInf_86 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\StatusCodeHandler\Smm\StatusCodeHandlerSmm\RELEASE\StatusCodeHandlerSmm
114 ModuleInf_87 = TunnelCreekPkg_28DEC17-6C75-448F-87DC-BDE4BD579919_0.2\TCInitDxe\TCInitDxe\RELEASE\TCInitDxe.inf
115 ModuleInf_88 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\RestrictedTCMicrocode\TCMicrocode.inf
116 ModuleInf_89 = TunnelCreekPkg_28DEC17-6C75-448F-87DC-BDE4BD579919_0.2\ResetSystemRuntimeDxe\ResetSystemRuntimeDxe\RELEASE\TCResetSystemRuntimeDxe.inf
117 ModuleInf_90 = TunnelCreekPkg_28DEC17-6C75-448F-87DC-BDE4BD579919_0.2\Smm\TCsmmDispatcher\RELEASE\TCsmmDispatcher.inf
118 ModuleInf_91 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\Console\TerminalDxe\TerminalDxe\RELEASE\TerminalDxe.inf
119 ModuleInf_92 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Bus\Usb\UsbBotPei\UsbBotPei\RELEASE\UsbBotPei.inf
120 ModuleInf_93 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Bus\Usb\UsbBusDxe\UsbBusDxe\RELEASE\UsbBusDxe.inf
121 ModuleInf_94 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Bus\Usb\UsbBusPei\UsbBusPei\RELEASE\UsbBusPei.inf
122 ModuleInf_95 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Bus\Usb\UsbKbDxe\UsbKbDxe\RELEASE\UsbKbDxe.inf
123 ModuleInf_96 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Bus\Usb\UsbMassStorageDxe\UsbMassStorageDxe\RELEASE\UsbMassStorageDxe.inf
124 ModuleInf_97 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Bus\Usb\UsbMouseDxe\UsbMouseDxe\RELEASE\UsbMouseDxe.inf
125 ModuleInf_98 = TopcliffPkg_2B827A37-D5D8-4053-9925-92CA344EE965_0.1\UsbPei\UsbPei\RELEASE\UsbPei.inf
126 ModuleInf_99 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\Variable\RuntimeDxe\VariableRuntimeDxe\RELEASE\VariableRuntimeDxe.inf
127 ModuleInf_100 = IntelFrameworkModulePkg_88894582-7553-4822-B484-624E24B6DECF_0.92\Universal\Console\VgaClassDxe\VgaClassDxe\RELEASE\VgaClassDxe.inf
128 ModuleInf_101 = IntelFrameworkModulePkg_88894582-7553-4822-B484-624E24B6DECF_0.92\Bus\Pci\VgaMiniPortDxe\VgaMiniPortDxe\RELEASE\VgaMiniPort.inf
129 ModuleInf_102 = MdeModulePkg_BA0D78D6-2CAF-414B-BD4D-B6762A894288_0.92\Universal\WatchdogTimerDxe\WatchdogTimer\RELEASE\WatchdogTimer.inf
130 ModuleInf_103 = NuvotonHardwarePkg_F1658FE8-7570-45b2-BAAD-305CF5CCECF_0.10\Winbond83627dhgDxe\Winbond83627dhgDxe\RELEASE\Winbond83627dhgDxe.inf
131 ModuleInf_104 = CrownBayPkg_46C1F476-A85E-49A8-B258-DD4396B87FEF_0.2\GopBinary\GopBinary

```

图 5-7 得到的模块解决方案

5.1.4 BIOS Image 实现

在得到所有的 Module Solution 之后，在 Repository 里，所有的 Module INF 文件会有一个二进制的 EFI 文件与之对应。Assembly 的过程就是将这些 Module 的二进制文件按照 EDKII 下的 Build 规则，组合在一起，得到最终的 FD 文件，这个 FD 文件就是得到 BIOS Image。图 5-8 为 Build BIOS Image 的过程示意图。

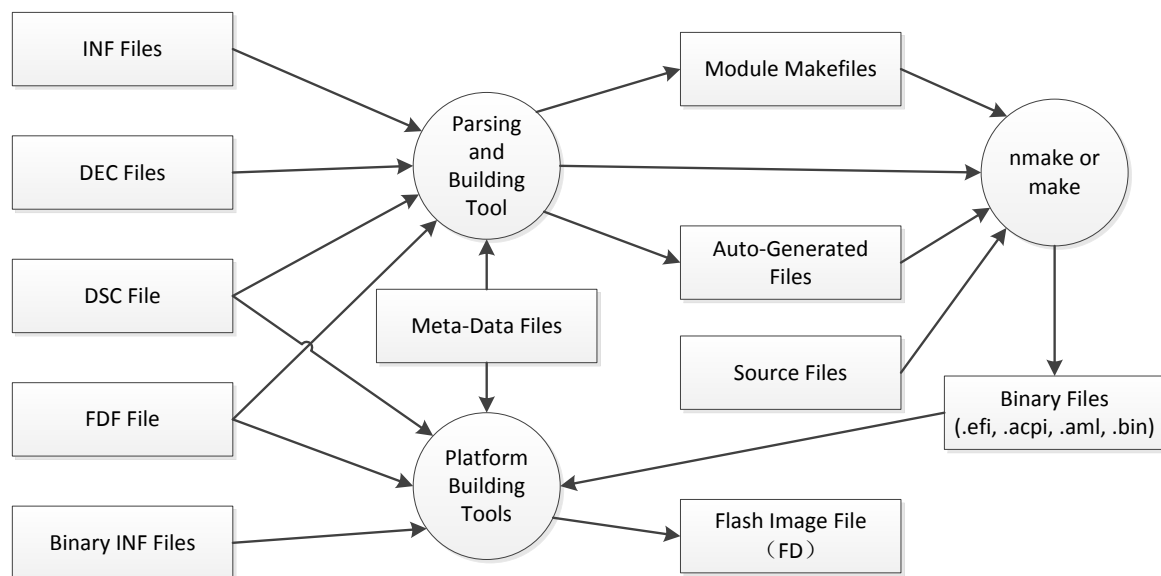


图 5-8 Build BIOS Image 示意图

从图中可以看出 EDKII 的 Build 过程主要由三个阶段组成：

(1) 预处理和 AutoGen 阶段：主要是解析元数据 meta-data 文件，UCS-2LE 编码文件和 VFR 文件，生成一些 C 源文件和 Makefile 文件。在 AutoGen 阶段，Build 工具的第一个工作就是寻找 target.txt 文件。在 target.txt 文件里需要设置 Build 目标的一些配置，包括需要 Build 的目标 DSC 文件、Build 目标的 CPU 的架构、和 Build 将要借助的编译工具。如果构建工具发现模块的描述文件在当前目录(INF 文件)，它将去单独 Build 该模块而不是 Build 整个文件系统。当 Build 工具得到 target.txt 文件,它开始解析平台描述文件(DSC)。在 DSC 文件里，Build 工具将定位 DSC 文件里所有 INF 文件的所有模块和库，以及其他平台的设置(包括 DEC 指定默认设置的 PCDs 所使用的模块和库，PCDs 的值不会在 DSC 文件中指定，而是在相应的 INF 文件中进行设置)。下一步在 AutoGen 阶段是生成 Build 模块文件所需要的一些临时文件，这些文件包括：AutoGen.h、AutoGen.c、Project.depex 和 Makefile。每个模块中发现了 DSC 文件将生成 Makefile。一旦所有的 Makefile 文件都生成了，Build 工具会调用 nmake 去解析 Makefile 文件。

(2) Build、Make 阶段：该阶段主要是将源文件处理成 EFI 文件。这一阶段是以 Build 所必需的库文件开始的。库文件包括 EDK 的一些组件和 EDKII 的 Module。这个阶段的输出是被链接成的 PE32+/COFF 文件，他们就是将要生成 EFI Image 的头文件。Build 的规则是记录在 build_rule.txt 文件里，它里面规定了不同的输入格式文

件，会如何通过相应的 Command 被编译的。这些输入文件包括[C-Code-File]、[C-Header-File]、[Assembly-Code-File.COMMON.COMMON]、[Assembly-Code-File.COMMON.IPF]、[Visual-Form-Representation-File]、[Object-File]、[Static-Library-File]、[Static-Library-File.USER_DEFINED]、[Dynamic-Library-File]、[Dependency-Expression-File]、[Acpi-Source-Language-File]、[C-Code-File.AcpiTable]、[Acpi-Table-Code-File]、[Masm16-Code-File]、[Microcode-File.USER_DEFINED, Microcode-File.Microcode]、[Microcode-Binary-File]、[EFI-Image-File]、[Unicode-Text-File]、[Efi-Image.UEFI_OPTIONROM]、[Visual-Form-Representation-File.UEFI_HII]、[Hii-Binary-Package.UEFI_HII]。图 5-9 是 C-Code-File 定义在 build_rule.txt 里的内容。

```
[C-Code-File]
<InputFile>
  ?.c
  ?.C
  ?.cc
  ?.CC
  ?.cpp
  ?.Cxx
  ?.CXX

<ExtraDependency>
  $(MAKE_FILE)

<OutputFile>
  $(OUTPUT_DIR) (+) ${s_dir} (+) ${s_base}.obj

<Command.MSFT, Command.INTEL>
  "$(CC)" /Fo${dst} $(CC_FLAGS) $(INC) ${src}

<Command.GCC, Command.RVCT>
  # For RVCTCYGWIN CC_FLAGS must be first to work around pathing issues
  "$(CC)" $(CC_FLAGS) -o ${dst} $(INC) ${src}
  "$(SYMRENAME)" $(SYMRENAME_FLAGS) ${dst}

<Command.ARMGCC, Command.ARMLINUXGCC>
  "$(CC)" $(CC_FLAGS) -o ${dst} $(INC) ${src}
```

图 5-9 C-Code-File 在 build_rule.txt 里的设计

最终 Image 文件的构成是被 EFI 架构协议所认可的一些文件，这些类型的文件是 EFI 可执行映像文件(.efi)所支持的。包括 ACPI 机器语言文件(ACPI machine language file, *.aml)、ACPI 表文件(ACPI table file, *.acpi)、实模式可执行文件(real mode executable file, *.com)和微码二进制文件(microcode binary file, *.bin)。

(3) Build Image Gen 阶段：这个阶段只要是将 EFI 支持的二进制文件处理成 FD 的二进制 Image 文件，即 UEFI BIOS 文件。一些在 FDF 文件中被定义或者使用的 PCDs

都在平台的 DSC 文件中进行了赋值。所以在 Build BIOS Image 的时候, 至少需要一个含有 PCDs 值的 DSC 文件和一个对 PCDs 进行排列规则进行规定的 FDF 文件。如果有一个 PCD 在 FDF 文件中用到, 但是没有在 DSC 文件中定义, Build 过程将会被终止。图 5-10 所描述的是 FD 文件的生成过程, 这就是典型的 UEFI Firmware 的层属关系。从最初的源文件, 通过编译器, 组装器, 链接器等生成了 PE32 文件, PE32 文件通过 GenFw generate firmware 步骤生成了 Firmware Image; 然后逐层封装, 得到了 EFI section、FFS、FV 文件、最终得到 FD 文件, 这一过程是 GenFd, 生成 firmware device

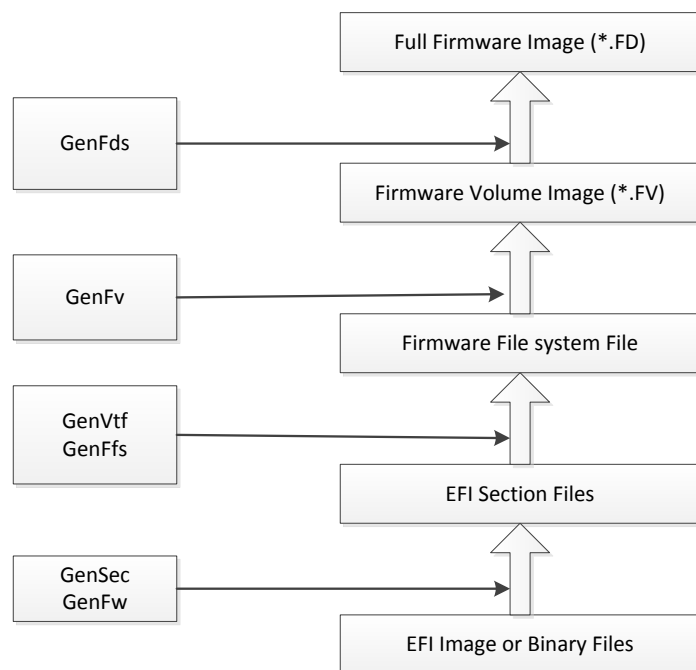


图 5-10 生成 Fd 文件的过程

从上至下的看。FD 是最终得到的 image。FD 是 Firmware Device, 是可以启动的 flash, 平台的最终完整的 Binary Image。它是一组二进制模块, 和一些其他组件, 如变量空间, Payload, 还有其他非 UEFI, 非 PI 的部件构成, 包含多个 FV 文件。Firmware Volume, 是 Firmware 存储的文件级别的接口。FV 与 FD 是一个多对多的关系, 一个 Fv 文件可以有多个 Flash Device, 一个 FD 文件中可以有多个 FV 文件。FV 可以用来支持其他存储方式, 例如磁盘分区, 网络设备等。在所有的情况下, FV 都是要遵循一种 Binary File System 文件系统格式的。在 FV 中, 所有的 Module 都以文件的形式存储。某些 Module 是在本地上的, 连接在固定地址, 从 ROM 中执行; 还有一种是执行时要加载到内存里;

还有一些 Module 可以在没有内存的情况下从 ROM 执行，或者在有内存的情况从内存执行。文件本身都有着内部定义的 binary 格式。这些格式允许实现安全，压缩，签名等应用。使用 binary，EFI 格式的文件，生成 EFI 的 FFlash Image。

我们将得到的所有的模块解决方案，调出 VS2008 编译工具的命令平台，在设置完环境变量之后，输入 `build -p Assembly.dsc -D SYMBOLIC_DEBUG=TRUE -D LOGGING=TRUE -b DEBUG -a IA32 -n 1 -t VS2008x86 -j Images\BinaryBuild\log\log.txt` 命令，在经历 PE32+/Coff 文件、EFI 文件、FV 文件这几个阶段之后，最终得到 Project.FD 文件，这个文件大小为 2M。Build 的日志文件会被记录在工作区的 log 文件夹的 log.txt 里。

5.2 测试结果

前面已经总结出来整个 BIOS Image 的开发流程。在得到了 Project.fd 文件后，本节主要对所得到的 UEFI Image 进行了测试。在装有 win7 系统的 PC 开发机里，采用 dediprog 软件，通过 deliprog 工具将 BIOS Image 下载 Inforce9450 的 Flash 里面，进行加载操作系统的可用性测试。还进行了多次测试来测试 UEFI 加载系统的稳定性。

5.2.1 硬件环境

本文的硬件平台采用的是 Inforce9450 的硬件平台，图 5-11 为 Inforce9450 的结构图。

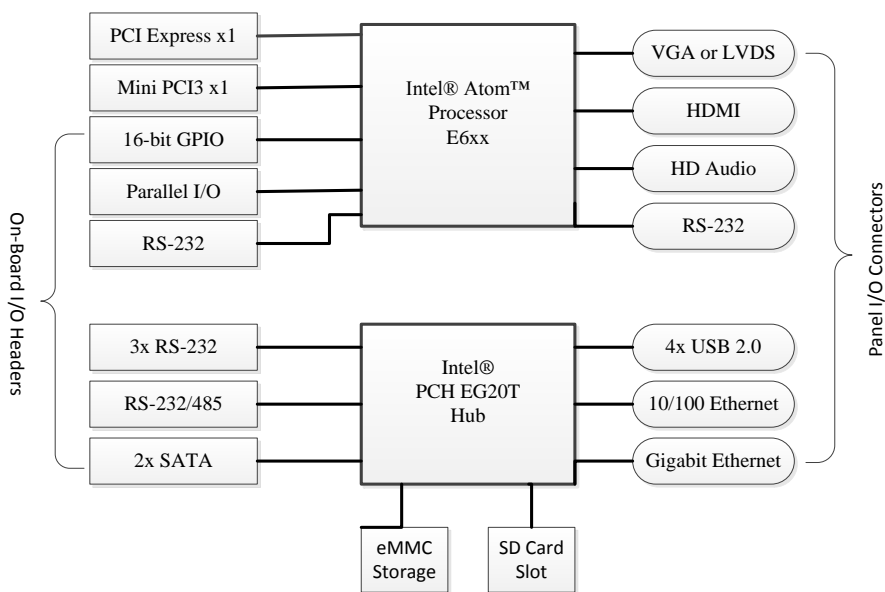


图 5-11 Inforce9450 的结构图

相关的参数如下：

- (1) 处理器：Intel® Atom™ Processor E6xx, 1.6Ghz
- (2) 芯片组：Intel® Platform Controller Hub EG20T
- (3) 内存：Up to 2GB DDR2 Memory, 800MT/s
- (4) 存储接口：2 个 SATA 端口（每个端口 150Mb/s）、SD 卡和 eMMC（可选）
- (5) 网络接口：Gigabit Ethernet with Wol、10/100 Ethernet with Wol、Wireless LAN Support on Mini-card slot
- (6) 音频和视频接口：VGA or LVDS connectors、HDMI connector、HD Audio connectors
- (7) I/O 端口：4 个 USB2.0、并行 I/O 口、16 位的 GPIO 口、6 个 RS-232 出口（一个可以当成 RS-485）、1 个 PCI 插槽
- (8) 电源：+12V

5.2.2 测试结果

将 HP-UX11iV3 系统制作一个 U 盘的镜像文件。借助 dediprog 烧写 BIOS 工具，将得到的 BIOS Image Project.fd 烧写到 Inforce9450 的硬件平台中，并将操作系统启动 U 盘插到 Inforce9450 的相应接口。给平台上电，等 UEFI 进入 Shell 界面，可以看到如图 5-12 的 Shell 画面。

```
1182 Loading driver at 0x0003DFEE000 EntryPoint=0x0003DFEE260 Shell.efi
1183 InstallProtocolInterface: BC62157E-3E33-4FEC-9920-2D3B36D750DF 3E32BA90
1184 Starting boot image [Shell] ...
1185 Starting boot image [Shell] ...
1186 PROGRESS CODE: V3058001 IO
1187 InstallProtocolInterface: 387477C2-69C7-11D2-8E39-00A0C969723B 3E322F14
1188 InstallProtocolInterface: 752F3136-4E16-4FDC-A22A-E5F46812F4CA 3E323410
1189 InstallProtocolInterface: 6302D008-7F9B-4F30-87AC-60C9FEF5DA4E 3E0A3090
1190 UEFI Interactive Shell v2.0. UEFI v2.31 (EDK II, 0x00010000). Revision 1.02
1191 Mapping table
1192 Error. No mapping found
1193 Press ESC in 5 seconds to skip startup.
1194 2.0 Shell>
```

图 5-12 Shell 界面

输入 exit，进入的 UEFI 的应用工具界面，有一个“EFI USB Device”的选项，选择该选项，就可以 boot 进入到 U 盘所存放的 HP-UX11iV3 操作系统，如图 5-13、5-14

所示。选择 Boot 功能后，就可以进入到 HP-UX11iV3 操作系统界面。

```
1 exit
2 PDB = o:\build\Build\Repository\ShellPkg_9FB7587C-93F7-40a7-9C04-FD7BA94EE646_0.50\Application\Shell\Shell\DEBUG_LOG\Ia32\Shell.pdb
3 Image Return Status = Success
4 MinnowBoardIntel(R) Atom(TM) CPU E640 @ 1.00GHz1.00 GHzEDK II Build 65536 on Jan 2 20141024 MB Shell
5 Type Pages Pages Pages
6 =====
7 09 00000040 00000010 00000040
8 0A 00000040 00000029 00000040
9 00 00000040 00000018 00000040
10 06 00000060 0000001B 00000060
11 05 00000060 0000001E 00000060
12 Loading boot image [EFI USB Device] ...
13 Loading boot image [EFI USB Device] ...
```

图 5-13 UEFI 应用工具界面

```
1 Loading boot image [EFI USB Device] ...
2 Loading boot image [EFI USB Device] ...
3 PROGRESS CODE: V3058000 IO
4 FSOpen: Open '\EFI\BOOT\BOOTIA32.EFI' Success
5 PROGRESS CODE: V3058000 IO
6 FSOpen: Open '\EFI\BOOT\BOOTIA32.EFI' Success
7 InstallProtocolInterface: 5B1B31A1-9562-11D2-8E3F-00A0C969723B 3E5637A8
8 Loading driver at 0x0003DBC6000 EntryPoint=0x0003DBC6400
9 InstallProtocolInterface: BC62157E-3E33-4FEC-9920-2D3B36D750DF 3DEAB310
10 Starting boot image [EFI USB Device] ...
11 Starting boot image [EFI USB Device] ...
12 PROGRESS CODE: V3058001 IO
13 Welcome to GRUB!
14
15 GNU GRUB version 2.00
16
17 Use the and keys to select which entry is highlighted.
18 Press enter to boot the selected OS, 'e' to edit the commands
19 before booting or 'c' for a command-line.
20 boot
21 The highlighted entry will be executed automatically in 10s.
22 The highlighted entry will be executed automatically in 9s.
23 The highlighted entry will be executed automatically in 8s.
24 The highlighted entry will be executed automatically in 7s.
25 The highlighted entry will be executed automatically in 6s.
```

图 5-14 启动 U 盘镜像文件

在一次系统加载之后，还需要进行多次的测试来测试系统的稳定性，表 5-1 是进行 10 组测试的数据，该数据表明烧写该 Image 的系统能够较快和稳定的进入加载 HP-UX11iV3 系统。

表 5-1 十次测试结果

试验组数	能否加载 HP-UX11iV3 系统	加载时间 (s)
第一组	能	56
第二组	能	61
第三组	能	58
第四组	能	62
第五组	能	72
第六组	能	53
第七组	能	60
第八组	能	62
第九组	能	63
第十组	能	68

5.3 本章小结

本章对 BIM 文件和模块描述类文件进行了设计，针对性的选择了硬件平台和外部设备。将 BIOS Image 烧写到 Inforce9450 的硬件平台，能够快速准确的加载 HP-UX11iV3 操作系统，实现了本文所设计的目标。

6 总结与展望

6.1 全文总结

BIOS 和操作系统计算机密不可分的两个部分，随着 UEFI 的发展，BIOS 的功能愈发的强大，在加载操作系统之前能完成更多的应用。也能更快的加载系统进入操作系统。针对单独的操作系统，进行 BIOS 设计，是随着 UEFI 发展在这个领域的一种尝试。结合操作系统的特点，和使用时的一些具体环境，在加载 BIOS 的时候，有选择性的加载驱动。这样不仅能更快的启动计算机设备，进入到操作系统控制的状态，更能提高硬件资源的利用率，减少硬件设备故障对整个计算机的影响。本文是基于 HP-UX11iV3 这款系统的特点，针对性的设计出来一款 BIOS 文件，将 HP-UX11iV3 的优势和 UEFI 快速发展所带来技术特点结合起来。

本文对 HP-UX11iV3 操作系统的特点进行了详细描述，叙述了 UEFI 的设计原理和启动流程。同时还根据 EDKII 的背景，对 UEFI BIOS Image 的管理进行了 BIM 文件的设计，在实现过程中采取了 RSE 算法来寻找 Module Solution，并使用 VS2008 编译工具，按照 FDF 文件的 DSC 文件所设计的规则，对得到的 Module Solution 进行 Build，得到最终的 BIOS Image。把得到的 BIOS Image 烧写到 Inforce9450 的硬件平台中，借助 Shell 工具，加载 HP-UX11iV3 操作系统，更快速，准确的进入操作系统，实现本文所设计的目标。本文创新点包括：（1）针对专门的操作系统，定制出专门的 BIOS，将操作系统的优势发挥出来（2）采用 RSE 算法，更加智能的寻找 Module Solution。

6.2 展望

虽然把得到的 BIOS Image 烧写到 Inforce9450 的硬件平台，能更快，更准确去进入到 HP-UX11iV3 操作系统。满足本文的设计目标，但是，HP-UX11iV3 操作系统是企业级别的应用，一般是在大型服务器上应用。由于时间较短，经济能力有限，对在大型服务器上的设计和测试工作做得还不够，同时在 Find Module Solution

的算法还有许多需要改进的方面。(1) 在 BIOS 设计的时候, 考虑到大型服务器级的设计时, 对 CPU、硬盘、Flash 等硬件设备驱动的设计将需要更加具有针对性。(2) RSE 算法只是针对一个简单的硬件平台进行的设计, 企业级服务对于外部设备驱动的需求有着自己更高的需求, 如一个平台上需要更多的相同的 USB 扩展的驱动。这样子就将突显出了 RSE 算法的局限性。

针对单独操作系统的 BIOS 设计是一个尝试的开始。最后, 希望本文能对 UEFI 发展做出贡献,

致 谢

研究生的学习生涯会随着论文的结束而结束。在研究生期间，无论是在学校学习还是在企业实习，有许多帮助过我和指导过我的人，在这里我想对他们表示真诚的感谢。

首先，我要衷心感谢我的导师沈刚老师，感谢沈老师在研究生期间对我的关系和指导。沈老师学识渊博，治学严谨，虽然工作繁忙，但还是能经常给予我细致的指导，让我学到不少的知识，并且能力能够不断提高。而他创新性的学术思维、严谨的治学研究态度更值得我终身学习。

同时，还要感谢在 Intel 实习期间，帮助过我的各位同事。他们在我的论文的研究方向上给我的帮助和技术指导，才使得此论文能够完成。还有实验室的每一位同学，与他们在一起学习生活可以使我保持一个开朗乐观积极向上的心态，而这种心态伴随了我整个研究生生涯。

最后要感谢我的家人，在我研究生期间给我的关心和爱护，是我能持续进步的最大动力。

参考文献

- [1] 胡藉. 面向下一代 PC 体系结构的主板 BIOS 研究与实现: [硕士学位论文]. 南京: 南京航空航天大学图书馆, 2005
- [2] 韩山秀, 樊晓桢, 张盛兵. BIOS 的设计与实现. 微电子学与计算机, 2005(12): 15-17
- [3] 华东. 基于 UEFI 技术的 BIOS 系统分析及其 API 性能测试研究: [硕士学位论文]. 西安: 西安电子科技大学图书馆, 2012
- [4] 石浚菁. EFI 接口 BIOS 驱动体系的设计、实现与应用: [硕士学位论文]. 南京: 南京航空航天大学图书馆, 2006
- [5] 余超志, 朱泽民. 新一代 BIOS_EFI_CSSBIOS 技术研究. 科技信息(学术版), 2006(5): 14-15
- [6] 王宇飞. 基于 UEFI 的底层 API 的性能分析及其功能测试的研究与设计: [硕士学位论文]. 长春: 吉林大学图书馆, 2010
- [7] 洪蕾. UEFI 的颠覆之旅. 中国计算机报, 2007(50): 1-3
- [8] 倪光南. UEFI BIOS 是软件业的蓝海. UEFI 技术大会, 2007
- [9] 邢卓媛. 基于 UEFI 的网络协议栈的研究与改进: [硕士学位论文]. 上海: 华东师范大学图书馆, 2011
- [10] 齐书阳. HP-UX 不只是操作系统. 软件世界, 2008(2): 23-25
- [11] 郭涛. HP-UX 确定六年发展路线图. 中国计算机报, 2008(8): 30-32
- [12] 王卫兵. HP-UNIX 操作系统维护技巧. 金融电子化, 2003(7): 20-22
- [13] Murray, Matthew. UEFI Session, ExtremeTech. com Vol. (), pp. n/a, 2010
- [14] 张朝华. 基于 EFI/Tiano 的协处理器模型的设计与实现: [硕士学位论文]. 上海交通大学图书馆, 2007
- [15] 崔莹, 辛晓晨, 沈钢钢. 基于 UEFI 的嵌入式驱动程序的开发研究. 计算机工程与设计, 2010, 31(10): 2384-2387
- [16] 李振华. 基于 USBKEY 的 EFI 可信引导的设计与实现: [硕士学位论文]. 北

京交通大学图书馆, 2008

- [17] 万象. 基于 UEFI 系统的 LINUX 通用应用平台的设计与实现: [硕士学位论文]. 上海: 上海交通大学图书馆, 2012
- [18] 冯成. HP-UX11. 0 内涵谈. 每周电脑报, 1997(8): 16-18
- [19] 邱忠乐. 基于 PC 主板下一代 BIOS 的研究与开发: [硕士学位论文]. 南京: 南京航空航天大学图书馆, 2005
- [20] 吴广, 何宗键. 基于 UEFI Shell 的 Pre-OS Application 的开发与研究. 科技信息, 2010(1): 15-17
- [21] Intel Corporation. EDKII user manual Revision 1. 0. The United States: Intel Corporation, 2011: 10-12
- [22] BIOS 支架. 初识 UEFI 结构. www.bios.net.cn, 2008
- [23] Intel Corporation. Intel 64 and IA-32 Architectures Software Develop's Manual. The United States: Intel Corporation, 2009: 10-778
- [24] UEFI Forum. UEFI Specification. Version 2. 3, 2011
- [25] 任超. EFI 技术的实现与应用研究: [硕士学位论文]. 上海: 上海交通大学图书馆, 2005
- [26] 王兴欣. 基于 EFI 和多核体系的软件运用架构: [硕士学位论文]. 成都: 电子科技大学图书馆, 2011
- [27] 唐文彬, 陈熹, 陈嘉勇. UEFI Bootkit 模型与分析. 计算机科学, 2012(4): 8-10
- [28] 陈文钦. BIOS 研发技术剖析. 第 1 版. 北京: 清华大学出版社, 2001: 30-44
- [29] Intel Corporation. Intel 64 and IA-32 Architectures Software Develop's Manual. The United States: Intel Corporation, 2009: 10-778
- [30] Intel Corporation. EDK II Extended INF File Specification Revision. Version 1. 24, 2013
- [31] Intel Corporation. EDK II Package Declaration (DEC) File Specification Revision. Version 1. 24, 2013
- [32] Intel Corporation. EDK II Platform Description (DSC) File Specification Revision. Version 1. 24, 2013
- [33] Intel Corporation. EDK II Flash Description (FDF) File Specification Revision.

Version 1. 24, 2013

- [34] Intel Corporation. Driver Writer's Guide for UEFI 2.0. Revision 0. 95, 2009: 9-51
- [35] Intel Corporation. EDKII Platform Configuration Database Infrastructure Description. Revision 0. 55, 2009: 11-13
- [36] 吴松青, 王典洪. UEFI 的 Application 的分析与开发. 计算机应用与软件, 2007(2): 45-47
- [37] Intel Corporation. UEFI 2. 1 Porting Guide, 2006
- [38] 傲博. EFI/UEFI 入门: All for Beginner. www.aub.org.cn, 2008
- [39] 周伟东. 基于 EFI BIOS 的计算机网络接入认证系统的研究与实现: [硕士学位论文]. 西安: 西安电子科技大学图书馆, 2008
- [40] Intel Corporation. Intel UEFI Packaging Tool. Revision 040, 2013
- [41] Intel Corporation. EDK II Build Specification Revision. Version 1. 24, 2013
- [42] Mark Lutz. Programming Python, 3rd Edition. O'Reilly, August 2006
- [43] 潘登. EFI 结构分析与 Driver 开发: [硕士学位论文]. 长沙: 国防科技大学图书馆, 2004
- [44] 马栋. 基于 HP-UX 11i 操作系统的中间业务系统设计与实现: [硕士学位论文]. 南昌大学图书馆, 2007
- [45] Intel, HP, Microsoft, Advanced Configuration and Power Interface Specification. Revision 4. 0, 2009: 5-496