

分类号: \_\_\_\_\_

密级: \_\_\_\_\_

U D C: \_\_\_\_\_

编号: \_\_\_\_\_

## 工学硕士学位论文

# Linux 环境下 DUET 平台引导程序 设计与实现

硕士研究生 : 桂 坤

指导老师 : 顾国昌 教授

学位级别 : 工学硕士

学科、专业 : 计算机应用技术

所在单位 : 计算机科学与技术学院

论文提交日期 : 2010 年 1 月

论文答辩日期 : 2010 年 3 月

学位授予单位 : 哈尔滨工程大学



Classified Index:

U.D.C:

# **Design and Implementation of DUET Loader under Linux Environment**

Candidate: Gui, Kun

Supervisor: Prof. Gu Guochang

Academic Degree Applied for: Master of Engineering

Speciality: Computer Applied Technology

Date of Submission: January, 2010

Date of Oral Examination: March, 2010

University: Harbin Engineering University

# 哈尔滨工程大学

## 学位论文原创性声明

本人郑重声明：本论文的所有工作，是在导师的指导下，由作者本人独立完成的。有关观点、方法、数据和文献的引用已在文中指出，并与参考文献相对应。除文中已注明引用的内容外，本论文不包含任何其他个人或集体已经公开发表的作品成果。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。

作者（签字）：



日期：

2010年3月15日

# 哈尔滨工程大学

## 学位论文授权使用声明

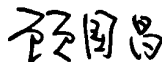
本人完全了解学校保护知识产权的有关规定，即研究生在校攻读学位期间论文工作的知识产权属于哈尔滨工程大学。哈尔滨工程大学有权保留并向国家有关部门或机构送交论文的复印件。本人允许哈尔滨工程大学将论文的部分或全部内容编入有关数据库进行检索，可采用影印、缩印或扫描等复制手段保存和汇编本学位论文，可以公布论文的全部内容。同时本人保证毕业后结合学位论文研究课题再撰写的论文一律注明作者第一署名为哈尔滨工程大学。涉密学位论文待解密后适用本声明。

本论文（☐在授予学位后即可 ☒在授予学位 12 个月后 ☐解密后）由哈尔滨工程大学送交有关部门进行保存、汇编等。

作者（签字）：



导师（签字）：



日期：

2010年3月18日

2010年3月18日

## 摘 要

Unified Extensible Firmware Interface (UEFI) 是由 Intel 提出的下一代计算机固件接口标准。旨在糅合现代软件工程思想以及设计方法, 构建出一个灵活、健壮、易扩展、可重用、方便开发和维护的固件开发框架。Intel 在 UEFI 框架的基础上, 通过开源社区提供了一套 UEFI 开发包 (EDK, EDK2) 及多种 UEFI 模拟平台的实现样例。DUET 平台包含在 UEFI 开发包中, 是一种不同于其它运行在操作系统层次上模拟平台的 UEFI 模拟平台, 其明显的特点是再封装传统 BIOS 产生的 ACPI 表中的各种函数及系统调用、形成 UEFI Framework 所需要的各种系统接口, 从而运行在实际硬件平台之上。针对其特点, 该模拟平台的设计和实现充分的重用了 EDK2 Core 中的 Library 以及 Module, 以便在最大程度上简化代码和工作量。

本文主要研究了 DUET 平台引导程序的功能, 设计思路及实现方法。讨论了 DUET 平台和基于 UEFI Framework 的平台之间的区别与联系。然后展开讨论了基于 MBR 和 GPT 的 Boot Sector 的设计、如何实现相关工具将 Boot Sector 写入到启动介质的相关扇区、同时提出了如何编写 Shell Script 及编译工具编译整个工程。

本文深入描述了从传统 BIOS 启动完成之后跳转到 DUET 平台引导程序执行启动介质的 Boot Sector、跳转保护模式、解压缩需要执行的 DXE 镜像, 并准备 DXE 镜像所执行的 Memory、Stack 并以 HOB 的方式传递给 DXE Entrypoint。分析了整个 DUET 平台的内存映射、以及 DxeIPL 和 EFI Loader 的实现方式及如何重用传统 BIOS 提供的系统调用。

**关键词:** UEFI; EDK2; DUET; 引导程序; Boot Sector

## Abstract

Unified Extensible Firmware Interface(UEFI) is the next generation standard of computer firmware raised by Intel. Mixed modern software engineering idea and design method, establish a flexible, robust, easily extensible, reusable, convenience firmware development framework for development & maintain. Based on the basis of UEFI framework, Intel provides a suit of UEFI Extensible Development Kit(EDK, EDK2) and couples of UEFI emulation platform sample implementation according to open source forum. DUET platform included by EDK & EDK2, is an UEFI emulation platform different with which running over operation system level platforms, obvious feature is re-encapsulate the system calls and functions generated by legacy BIOS, to form UEFI Framework system interfaces for satisfy the requirement so as to run on real platform. To point against this feature, design and implementation of the emulation platform full reuse Library and Modules of EDK2 Core to simplify the code and workload in the whole hog.

This thesis mainly focus on researching the functionality、 design layout and implementation method. It discussed the differences and relationship of DUET platform with UEFI Framework and then Boot Sector based on MBR or GPT, the tool for write Boot Sector into specify sector in the boot media, and simply for write Shell script and Build Tools to compile the whole project.

Thesis also described how to jump to DUET Loader after legacy BIOS finished to execute code in Boot Sector, switch into Protected Mode, decompress DXE image and prepare the Memory, Stack then passed to DXE Entrypoint as HOB required by DXE image for execute. Analyzed the memory layout of DUET Platform and the method of DxeIPL and EFI Loader's implementation, how to reuse the system calls of Legacy BIOS according to ACPI table.

**Key words:** UEFI; EDK2; DUET; Loader; Boot Sector

# 目 录

第 1 章 绪论	1
1.1 课题研究背景	1
1.2 国内外研究现状	2
1.3 课题主要研究内容	2
1.4 论文组织结构	4
第 2 章 DUET 平台分析	6
2.1 UEFI Framework	6
2.2 DUET Framework	7
2.3 启动流程	8
2.3.1 UEFI 启动流程	9
2.3.2 DUET 启动流程	10
2.4 DUET 与 UEFI Firmware 的异同	12
2.5 本章小结	13
第 3 章 DUET 平台引导程序的设计	15
3.1 磁盘分区表	15
3.1.1 Main Boot Record	16
3.1.2 Guided Partition Table	18
3.2 Lzma 压缩	20
3.3 GnuGenBootSector	22
3.4 Build Shell Script	22
3.4.1 Post Build	22
3.4.2 Create Boot Disk	23
3.5 ACPI	24
3.6 平台模块划分	28
3.7 内存映射	29
3.8 本章小结	32

第 4 章 DUET 平台引导程序的实现 .....	34
4.1 Boot Sector .....	34
4.1.1 BIOS Parameter Block 的实现 .....	34
4.1.2 查找 EfiLdr .....	35
4.1.3 查找 EfiVar .....	36
4.2 EFI Loader .....	37
4.2.1 跳转保护模式 .....	37
4.2.2 中断向量表 .....	39
4.2.3 执行 EFI Loader .....	40
4.3 DxeIPL .....	45
4.4 实验结果与分析 .....	48
4.4.1 实验环境及工具 .....	48
4.4.2 过程及结果分析 .....	48
4.5 本章小结 .....	50
结论 .....	51
参考文献 .....	53
攻读硕士学位期间发表的论文和取得的科研成果 .....	55
致谢 .....	56

# 第 1 章 绪论

## 1.1 课题研究背景

Unified Extensible Firmware Interface (UEFI) 是下一代计算机固件接口标准<sup>[1]</sup>。UEFI 采用面向对象的方法进行设计, 用模块化的方法组织驱动及应用程序, 模块间以标准的协议进行通信。UEFI 旨在替代 BIOS, 成为能够利用高级语言以及现代软件工程方法的下一代固件接口。

DUET (Developer's UEFI Emulation) 平台的是 Intel 为了方便 UEFI 开发人员进行 UEFI 驱动程序及应用程序开发和调试而开发的一种基于硬件环境的 UEFI 模拟平台。基于 UEFI 平台架构的 DUET 平台提供了一种在传统 BIOS 环境下可以进行引导、加载的 UEFI 环境。其可以在安装有传统 BIOS 的基于 IA32 或者 IA64 架构的实际平台上运行, 也可以在 Qemu、Bochs、VmWare 等虚拟机上运行。从而使得基于 UEFI BIOS 的驱动和应用程序的开发可以不必使用具有调试和烧写接口的主板平台, 大大的方便了 UEFI 驱动及应用程序的开发和调试工作。

而引导程序则是 DUET 平台最基本也是架构最重要的部分。负责在传统 BIOS 生命周期结束之时, 接管整个系统平台, 加载引导介质中位于 MBR 上针对 DUET 平台所需的代码并运行。为 UEFI 环境的建立起到一个承上启下的关键作用。图 1.1 为 DUET 平台的一个整体概况。

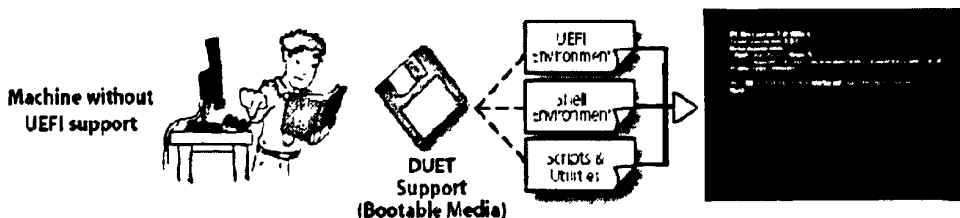


图 1.1 DUET 平台整体概况

可以看出, DUET 平台运行在没有 UEFI 环境的硬件平台上, 通过启动



引导介质（包括软盘、U 盘、硬盘等）构建出一个 UEFI 的环境、一个可与用户进行交互的 Shell，以及一些脚本和工具。

## 1.2 国内外研究现状

目前针对UEFI框架的模拟平台，直接在硬件平台上运行的只有Intel的DUET平台。Intel公司还存在类似运行于操作系统平台之上的模拟器，主要是Unix环境下的Unix Simulator和基于Microsoft Windows平台的NT32 Simulator。此类模拟器由于运行在操作系统之上，对很多硬件相关的行为只能是模拟和仿真，比如NT32 Simulator中对文件系统的操作在底层实际上是调用Windows API完成的。这在很大程度上限制了此类模拟器的功能，对于平台相关硬件驱动的开发就不可能在此类模拟器上运行。

而DUET平台则完全不同，其完全运行在硬件平台上，所有的操作都是基于UEFI框架来进行的。开发人员在开发时，在DUET上进行调试和测试，然后就可以直接将最后调试通过的UEFI驱动或应用程序在实际运行UEFI BIOS的平台上加载和运行。

## 1.3 课题主要研究内容

### 1. Boot Sector结构的研究

Boot Sector 也就是硬盘的第一个扇区，它由 MBR（Master Boot Record）、DPT（Disk Partition Table）和 Boot Record ID 三部分组成。MBR 又称主引导记录占用 Boot Sector 的前 446 个字节（0~0x1BD），存放系统主引导程序（它负责从活动分区中装载并运行系统引导程序）。DPT 即主分区表占用64个字节（0x1BE to 0x1FD），记录了磁盘的基本分区信息。主分区表分为四个分区项，每项 16 字节，分别记录了每个主分区的信息（因此最多可以有四个主分区）。Boot Record ID 即引导区标记占用两个字节（0x1FE and 0x1FF），对于合法引导区，它等于 0xAA55，这是判别引导区是否合法的标志。由于DUET平台还需要还支持GPT格式的磁盘介质（包括用于引导和数据存取），因此，还需要研究GPT格式分区时Boot Sector的写法。

Boot Sector结构的研究有助于DUET平台引导程序的编写和可引导磁盘相关工具的开发。Boot Sector存放传统BIOS生命周期结束之后进入DUET平台所需要执行的第一段代码,也是DUET平台引导程序的第一段代码。

## 2. 传统BIOS系统接口及ACPI表

在DUET平台中,为了减少工作量和加快平台的启动、运行速度,大量使用了如INT 10h、INT 13h、INT 15h等传统BIOS的中断调用。同时,将传统BIOS同OS的接口—ACPI表作为DUET平台大多数系统信息数据的来源,如启动设备选择阶段的硬盘可启动存储介质信息。对于引导程序来说,不仅在引导程序运行时通过BIOS系统接口调用来完成如读写软盘、显示字符等功能,还要需要在DUET平台运行初期阶段从传统BIOS中读出ACPI表并将其转换成符合UEFI标准文档规范的格式以便DUET平台运行时使用。

## 3. 实模式、保护模式以及Big-实模式

最早的Intel系列的CPU只存在一种操作模式,即现在所说的实模式(Real Mode)。在Intel推出80286之后,为了增强CPU的处理能力,同时也为了适应当时的软件开发需求,Intel提出了保护模式(Protected Mode)。保护模式相比实模式有很多优点。包括最大可访问4GB内存空间、支持虚拟内存、支持地址映射、内存保护和分段保护、更灵活的寻址方式以及多任务支持等。

在DUET平台中,相比传统BIOS的最大不同在于,要尽早跳入保护模式,从而可以使用保护模式的各种特性。这有利于平台程序开发的灵活性,也使得程序员可以开发出更复杂,功能更强大的BIOS应用程序。但在DUET平台中,引导程序最初运行在实模式下通过INT 13h调用读取需要执行的DUET镜像文件并对其进行解压缩操作。当镜像文件过大时,往往超过实模式下所需的1M的地址空间。因此,需要研究Big-实模式来满足项目需求。

## 4. DUET平台框架

该部分主要研究DUET平台的启动流程及内存映射。从而保证引导程序在引导时将各个模块加载到相应的地址空间。DUET平台在代码镜像上主要

由4部分组成: EfiLoader、DxeIpl.z、DxeMain.z、EfiMain.z, z文件代表使用Lzma算法压缩。由于这些镜像文件在EFI Loader中解压缩, 虽然此时已经进入保护模式, 可以使用4G的地址空间(针对IA32)。但此时从引导介质上读取镜像到指定的地址空间, 如果镜像文件过大会覆盖掉引导程序所在的代码, 会导致引导失败。因此需要在编译和压缩时都首先考虑代码的大小问题。

## 5. 调试及测试方法的研究

由于DUET平台引导程序本身的特殊性, 在其启动初期, 不可能通过嵌入GDB Stub等方式实现软调试。因此其调试需要在可调试的虚拟机(如Bochs)中进行。但DUET平台本身运行在实际硬件平台上, 在虚拟机中进行的调试往往会受虚拟机自带的传统BIOS的影响(如不符合标准BIOS文档规范)。在这种情况下, 需要采用带有硬件调试接口的硬件平台通过Intel ITP(一种基于特殊硬件的调试工具)进行调试。对调试而言, 至少需要掌握在虚拟机中进行代码调试和使用ITP进行代码调试两种调试方法。

为了软件的健壮性和规范性。DUET平台采用UEFI SCT(标准测试用例)进行测试, 需要对如何使用UEFI SCT进行一定的研究。

## 1.4 论文组织结构

本文对Linux环境下DUET平台引导程序的设计及实现进行了详细的说明和描述。讨论了引导程序的研究背景、面临的问题及研究意义, 深入分析DUET平台同UEFI架构的平台的异同, 并根据DUET平台自身的特点对引导程序进行了设计及实现, 讨论了在设计及实现中遇到的问题及解决方案, 并描述了在开发过程中如何进行调试及测试。全文共分4个章节进行展开。

第一章为绪论, 主要讨论课题的研究背景、研究内容、国内外研究现状。针对UEFI模拟器的发展背景, 提出了研究DUET平台的必要性, 明确了引导程序的地位。

第二章深入分析了DUET平台。主要从其Framework、启动流程、以及烧写到Flash中的镜像文件所包含模块的异同3个方面进行展开。在各个小节中针对DUET平台特点, 以及其同基于UEFI的平台的异同点提出了引导

程序设计的基本思路。

第三章详细介绍了 DUET 平台引导程序的设计思路。分析了基于 DPT、GPT 的磁盘分区格式、以及两种磁盘分区格式在 Boot Sector 编写时的不同点及要求。并从实模式下寻址 1M 地址空间结合传统 BIOS 标准文档中 1M 以下地址空间的内存分配分析了 DUET 镜像文件的地址空间。根据地址空间范围提出了应对镜像文件进行压缩并引出 Lzma 压缩的设计思路。本章还针对 Linux 环境的特点引出了 GnuGenBootSector 工具以及应用此工具后镜像文件编译、生成相关脚本应用程序的设计方法并给出了相关代码。最后结合 DUET 平台自身特点给出了 DUET 平台引导程序在实现过程中应重用传统 BIOS 的 ACPI 表。本章还结合 DUET 平台启动流程及镜像文件特点提出了平台模块划分的思路。

第四章详细描述了 DUET 平台引导程序核心模块的实现。按平台运行时各模块运行时间顺序进行展开。DUET 平台引导程序主要由 Boot Sector、DxeIPL、EFI Loader 共 3 个核心模块组成。分别介绍了各自模块的实现方法,并给出了相关的实现代码。最后结合各种实验平台对 DUET 平台进行了实验,并给出了相关的实验结果。

## 第 2 章 DUET 平台分析

本章深入分析了 DUET 平台。主要从其 Framework、启动流程、以及烧写到 Flash 中的镜像文件所包含模块的异同 3 个方面进行展开。在各个小节中针对 DUET 平台特点, 以及其同基于 UEFI 的平台的异同点提出了引导程序设计的基本思路。

### 2.1 UEFI Framework

传统 BIOS 的规范自从 PC/AT 时代起就一成不变, 它已经无法很好地适用于日趋复杂的现代计算机体系, 并阻碍了引入新的平台技术。传统 BIOS 规范的缺陷包括:

- 1) 仅能使用汇编语言编程, 极大地限制了 BIOS 程序开发的效率, 使得 BIOS 开发成为少数人能够胜任的工作;
- 2) 由各自为政的 BIOS 开发商开发, 相互之间缺乏兼容性与互操作性;
- 3) 16 位实模式的 BIOS 程序极大地浪费了 IA32 架构所提供的资源, 更难以应用于 X64 架构。

可扩展固件接口规范 (Extensible Firmware Interface Specification) 定义操作系统与系统固件之间的开放接口。这些接口提供了平台相关信息、启动服务例程以及操作系统运行时服务例程。操作系统装载器与操作系统可通过接口调用这些服务例程。EFI 规范是一个公开的纯接口定义, 它不依赖于某个特定的 BIOS 制造商或某个特定的 BIOS 实现。它仅定义了平台固件必须实现的接口, 以及操作系统可能使用的一系列接口与数据结构, 其实现的方式与细节均取决于此规范的实现者。EFI 规范还定义了固件驱动程序模型, 使得所有遵循此模型开发的固件驱动程序能够互相协作<sup>[2]</sup>。

UEFI Framework 给出了基于可扩展固件接口规范实现平台的层次结构。其中, 整个平台运行于 Hardware 之上, SEC 及 PEI 阶段对 CPU 以及 Chipset 初始化的模块位于整个 Green H 的下端, 即图 2.1 中所示的 CPU Modules 以及 C/S Modules。在 Green H 的上半部分, 是固件中运行的各个模块的驱动 (EFI

Driver), 如 SCSI、Keyboard、USB、Hard Disk 等。这些驱动模块由 DXE 阶段的核心模块 DXE Core Module 根据 UEFI 标准文档的规定进行加载、运行。为了兼容传统 BIOS, 整个 Framework 中设计了一个 CSM (Compatibility Support Module) 提供对基于传统 BIOS 的操作系统的支持。Green H 的顶端是由各个驱动模块按 UEFI 标准文档导出的可扩展固件接口。兼容 EFI 的操作系统 (如 Windows 7、FC7)、Pre-boot 环境下的工具如 Shell、SMBIOS 查看器、以及一些外插卡上的 Option ROM 等都运行在基于可扩展固件接口的基础之上。

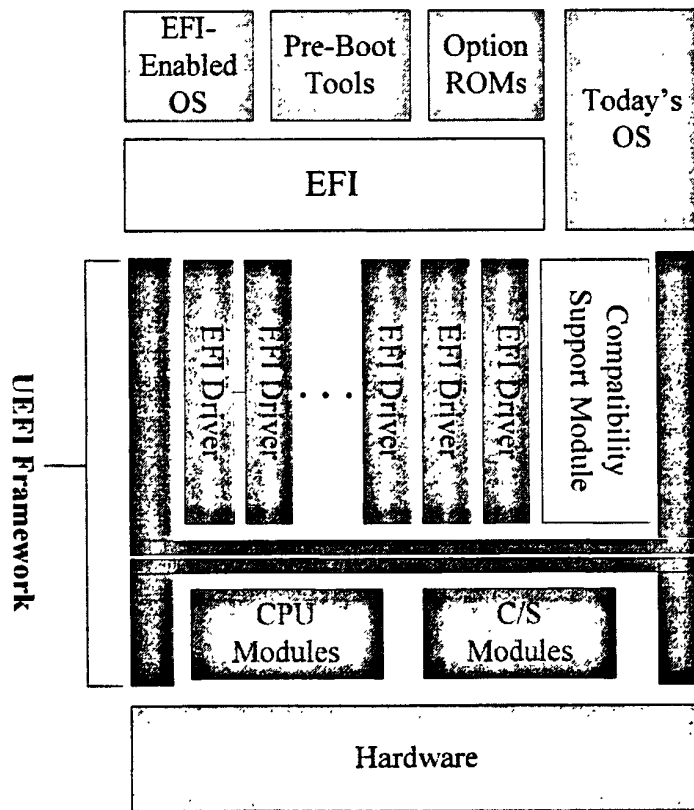


图 2.1 UEFI Framework 结构图

## 2.2 DUET Framework

DUET Framework 从结构上类似 UEFI Framework, 针对平台特点进行了部分修改, 如图 2.2 所示。

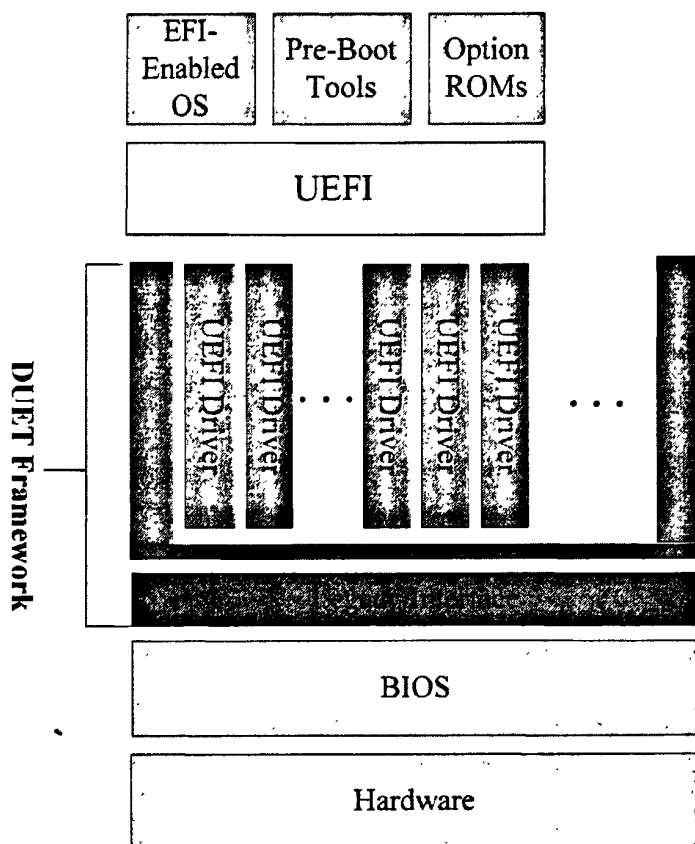


图 2.2 DUET Framework 结构图

从图 2.2 和图 2.1 对比可以看出，DUET 模拟平台和 UEFI Framework 的平台的主要区别在于 DUET 平台建立在 BIOS 提供的 Legacy Interface 基础之上，因此也没有 Green H 中的下半部分（CPU Module 和 Chipset Module）。DUET 模拟平台没有 CSM 模块和对传统 OS 的支持，因为其本身基于传统 BIOS 之上，不需要 CSM 模块去提供对传统 BIOS 的兼容。整个框架的其余部分 DUET 模拟平台和基于 UEFI Framework 的平台基本相同。

对于 DUET 平台引导程序而言，根据此图可以看出引导程序及平台其它模块应充分重用传统 BIOS 接口，从而减少项目开发时间、提高项目开发质量。

## 2.3 启动流程

本小节分别介绍 UEFI Framework BIOS 和 DUET Framework BIOS 的启

动流程，并结合 DUET 平台的特点分析两者的不同之处。

### 2.3.1 UEFI 启动流程

对于一台基于标准 UEFI Framework 的机器来说，其一个生命周期可以分为 SEC<sup>[3]</sup>、PEI<sup>[3]</sup>、DXE<sup>[4]</sup>、BDS、TSL、RT 以及 AL 阶段<sup>[5]</sup>。如图 2.3 所示。

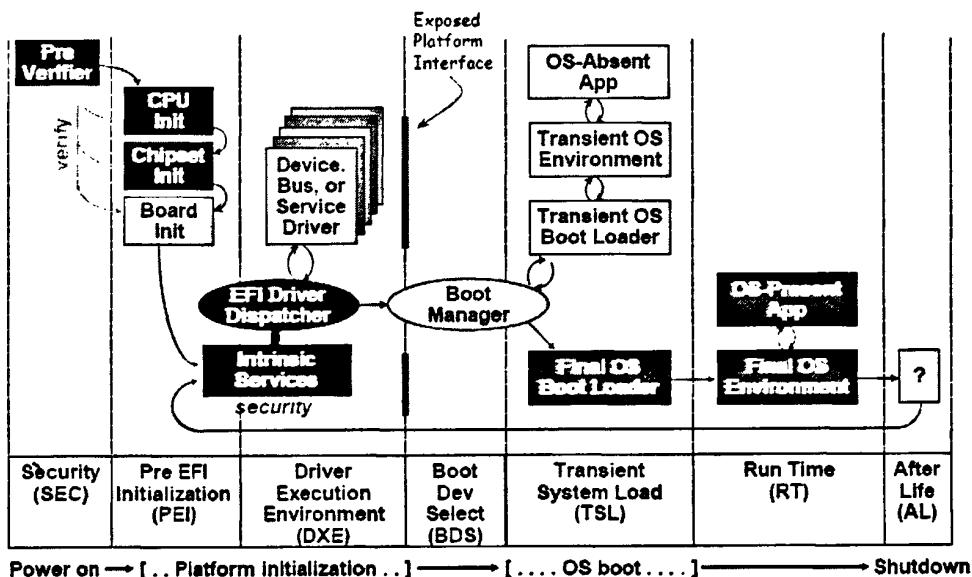


图 2.3 UEFI Framework BIOS 启动流程图

SEC (Security) 阶段主要完成对固件镜像文件的校验，在目前大多数基于 UEFI Framework 的平台来说，都没有实现 SEC 阶段。一般在可信计算平台中会考虑实现此阶段。

PEI (Pre EFI Initialization) 阶段用于对平台进行简单的初始化，一般要求 PEI 阶段的运行时间尽可能短。只完成对 CPU 的初始化以及南北桥芯片组的初始化。同时对内存进行简单的初始化，然后跳入 DXE 阶段。在 PEI 阶段的初期，由于 EDK2 绝大部分模块基于 C 语言编写，运行时需要一定的内存空间作为堆栈来使用。因此采用 CAR<sup>[6]</sup> (Cache As ROM) 技术将 CPU 中的 Cache 作为内存来使用，当内存完成初始化后，再将应用程序重加载到内存中运行。PEI 阶段初始化后得到的硬件信息以 HOB (Hand Off Block) 的形式传递到下一个阶段—DXE。

DXE (Driver Execution Environment) 阶段是整个基于 UEFI Framework



平台中执行时间最长，运行模块最多，也相对最重要的阶段。在此阶段中，DXE 的核心模块 DXE Dispatcher 会根据不断的扫描固件中的各个模块，并为这些模块加载相应的驱动，类似 Windows Driver Model<sup>[7]</sup>。DXE Dispatcher 是一个相对通用的模块，在 IA32、X64、IPF 不同 CPU 架构的硬件平台中，都可以使用相同的代码而只需要在编译工程文件时更改相应的配置。在 DXE 阶段中，硬件平台的绝大部分驱动如设备 SCSI、Keyboard、USB、Hard Disk 以及 PCI 设备的枚举及加载等都会完成<sup>[8]</sup>。并会根据加载的这些驱动得到的硬件信息生成 ACPI 表、SMBIOS 表。

BDS (Boot Device Select) 阶段是一个相对比较简单阶段。在此阶段只做启动设备的选择，并根据所选择的启动设备跳转到相关启动设备上的引导代码进行执行。

由于启动流程之后的阶段如 OS Boot 以及 Run Time 相对比较简单而且和 DUET 平台关系不大，在此不进行介绍。

### 2.3.2 DUET 启动流程

DUET 平台的启动流程和基于 UEFI Framework 的平台启动流程有一定区别，主要体现在整个启动流程的初始阶段。启动流程如图 2.4 所示。

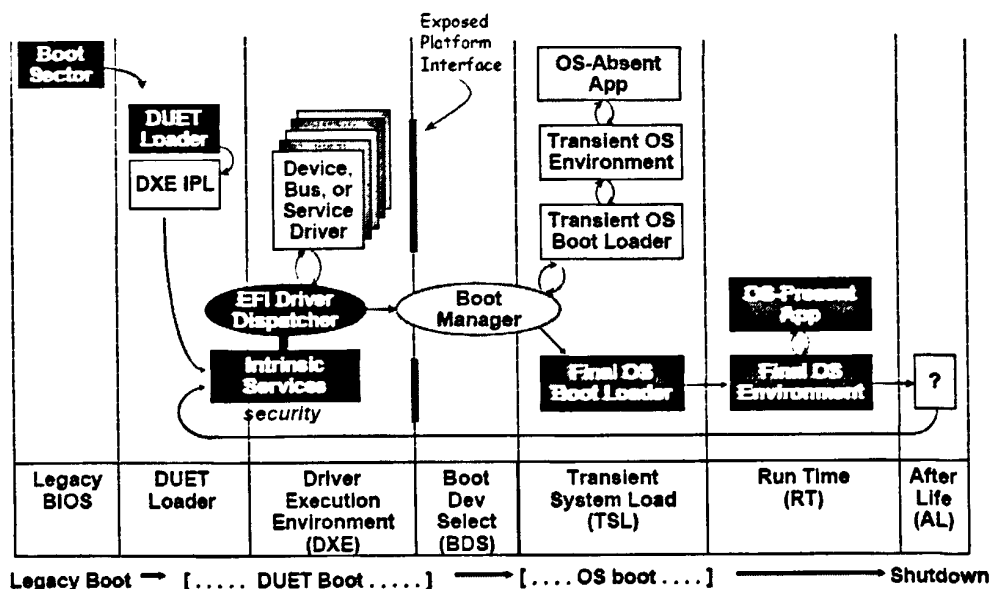


图 2.4 DUET 启动流程图

从以上启动流程可以看出, DUET 平台经过 Boot Sector 引导后, 由 DUET Loader 和 DXE IPL 阶段完成 DXE 阶段运行所必须的数据准备后, 直接跳入 DXE 阶段。而没有一般平台所应经历的 SEC 阶段, 以及 CPU、Chipset、以及 Board 的初始化。对于 SEC 阶段而言, DUET 平台本身并不需要。对于 PEI 阶段, 基于 UEFI Framework 的平台在完成 CPU、Chipset 以及 Board 简单的初始化对 DUET 平台也不需要, 因为传统 BIOS 已经完成了相关初始化使得这些硬件已经直接可用。在初始化过程中所得到的硬件平台信息 DUET 平台都可以从传统 BIOS 的 ACPI 表中得到。其余阶段和基于 UEFI Framework 的平台一致。对于 DUET 平台的启动流程, 根据实际代码的组织情况如图 2.5 所示。

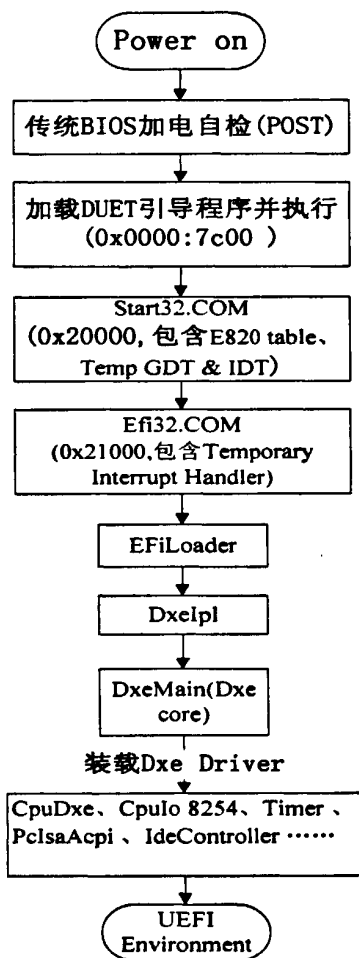


图 2.5 DUET 平台启动流程

以一台安装传统 BIOS 的机器为例。系统开机后，BIOS 首先进行加电自检操作（POST）、完成各个硬件的简单初始化，并生成将于操作系统进行交互的相关数据<sup>[9]</sup>。然后检查启动项中的存储介质中的 0 磁道 1 扇区（Boot Sector）中 0x1FE 处是否为 0xAA55，判断该 Boot Sector 是否可引导<sup>[10]</sup>。如果可引导，将其内容加载到 0x0000:7c00 处，然后 CPU 跳转到该地址进行执行<sup>[11]</sup>。

在跳转到 0x0000:7c00 后，CPU 执行的代码也就是 DUET 引导程序所开始进行的工作。首先其简单分析 FAT 文件表，寻找 EfiLdr 镜像，并判断该启动介质上是否有 EfiVar.bin 文件存在。EfiVar.bin 文件用于存储平台的一些常用 Variable，第一次引导时会创建并写入，以后再启动时直接读取该文件的内容从而加速系统引导。而后系统将 EfiLdr 镜像的第一个扇区的内容，也就是 start.com 的内容装载到内存 0x2000:0000，并跳转到该地址执行。

## 2.4 DUET 与 UEFI Firmware 的异同

为了更进一步的分析 DUET 平台与基于 UEFI Framework 的平台之间的差别，转换视角从生成的固件中所包含的各个模块以及所遵循的 UEFI/PI（Platform Initialization）文档的差异来进行比较和分析。在此，仅考虑 DUET 平台和基于 UEFI Framework 的平台都包含的部分。比较结果如表 2.1 所示。

表 2.1 DUET 与 UEFI Firmware 的异同

Item	UEFI Firmware	DUET
UEFI Spec	Yes	Yes
Framework/PI Spec	Yes(PEI/DXE/SMM)	Yes(Only DXE)
Platform	Platform Specific Initialization	No Initialization
Variable	Real variable on flash	#1 Fake variable on FS #2 Fake FVB on FS
Timer	8254 + ACPI Timer	#1 8254 #2 8254 + ACPI Timer
Metronome	ACPI Timer	#1 Port 61 #2 ACPI Timer

续表 2.1 DUET 与 UEFI Firmware 的异同

Item	UEFI Firmware	DUET
Reset	CF9 Reset	# 1 KBC Reset #2 ACPI Reset
Capsule	Yes	Not Support
CPU MP	Yes	Not Support
PCI	Full enumeration(Device detect and Resource Assignment )	Device detection only, no resource assignment.
USB	Legacy USB	No Legacy USB support
ATA	ATA/AHCI	ATA PIO2 mode, no AHCI
ISA	Super IO specific	Generic PCAT -2 COM + PS2 KB + PS2 Mouse + 2 Floppy
Video	#1 VESA GOP #2 Native GOP	#1 VESA GOP #2 VGA #3 Native GOP

总体而言，DUET 和基于 UEFI Framework 的平台 Firmware 并没有太大的差异。最显著的区别在于 DUET 平台只存在 DXE 阶段，没有一般平台固件所需要的 PEI 和 SMM 阶段。由于 DUET 平台运行在实际硬件平台之上，其本质意义上来说是一个模拟平台，不应该去操作 Flash，因此在实际平台上存在的 Variable 在 DUET 平台上基于文件系统的文件对其进行模拟。一般而言，Variable 都存储在引导介质的一个名为 Efivar.bin 的文件中。在 DUET 平台引导程序的设计和实现过程中应注意 DUET 平台没有 PEI 阶段。而 DUET 平台的其它模块则应根据表中所列举的内容进行实现。

## 2.5 本章小结

本章比较和分析了 DUET 平台与基于 UEFI Framework 的平台两者之间的相同点和不同点。并根据两者之间的不同点提出了 DUET 平台引导程序在设计和实现时应注意的问题。主要包括：

- 1). DUET 平台引导程序应充分重用传统 BIOS 所提供的系统接口调用从而简化引导程序的代码;
- 2). DUET 平台本身运行在传统 BIOS 基础之上, 引导程序中不必对 CPU、芯片组进行 UEFI Framework 平台中 PEI 阶段所做的初始化工作;
- 3). DUET 平台引导程序应使用传统 BIOS 生成的 ACPI 表来获得当前硬件平台相关信息。

## 第3章 DUET 平台引导程序的设计

本章详细介绍了 DUET 平台引导程序的设计思路。分析了基于 DPT、GPT 的磁盘分区格式、以及两种磁盘分区格式在 Boot Sector 编写时的不同点及要求。并从实模式下寻址 1M 地址空间结合传统 BIOS 标准文档中 1M 以下地址空间的内存分配分析了 DUET 镜像文件的地址空间。根据地址空间范围提出了应对镜像文件进行压缩并引出 Lzma 压缩的设计思路。本章还针对 Linux 环境的特点引出了 GnuGenBootSector 工具以及应用此工具后镜像文件编译、生成相关脚本应用程序的设计方法并给出了相关代码。最后结合 DUET 平台自身特点给出了 DUET 平台引导程序在实现过程中应重用传统 BIOS 的 ACPI 表。本章还结合 DUET 平台启动流程及镜像文件特点提出了平台模块划分的思路。

### 3.1 磁盘分区表

一般而言,可启动磁盘引导扇区位于 0 柱面、0 磁道、1 扇区。对于一个以主引导记录(Main Boot Record-MBR)格式组织的可启动磁盘引导扇区而言,其由 Boot Loader、分区表、以及 Magic Number 共 3 部分组成<sup>[12]</sup>,如图 3.1 所示。在 DUET 平台引导程序中,不仅要考虑基于 MBR 格式的可启动磁盘引导扇区。还要考虑兼容 EFI 的磁盘分区格式 Guided Partition Table(GPT)。我们将对两种磁盘分区结构分别进行讨论,并给出各自相关的 Boot Sector 的数据结构及设计思路。

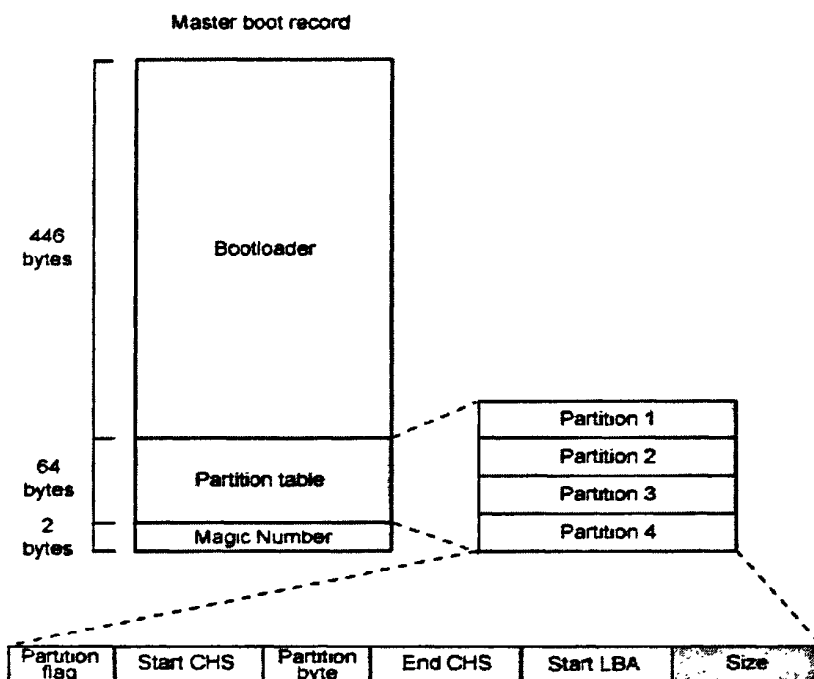


图 3.1 Boot Sector 结构图

### 3.1.1 Main Boot Record

MBR (Main Boot Record) 占用 446 字节，存放系统主引导程序和引导参数。其中的引导程序主要作用是：检查 DPT 表是否正确；如果系统硬件自检通过后，激活标志分区上的操作系统，并把控制权限交给启动程序。MBR 可以由 Fdisk、Com 等命令产生，不依赖于任何操作系统。磁盘引导程序是可以修改的，这样就可以实现多个操作系统以及文件系统的并存。图 3.2 为一个硬盘的逻辑结构图。

MBR 的作用主要包括如下：

- 1) 复制自身到一段低地址内存空间中，为 OS 引导代码准备所需空间；
- 2) 分析分区表找到活动 (active) 分区；
- 3) 将活动分区的第一个扇区加载到 0x0000:7c00 处；
- 4) 检查、验证该扇区是否为可引导扇区；
- 5) 通过跳转到 0x0000:7c00 处将控制权交给 OS 引导代码；
- 6) OS 引导代码开始工作。

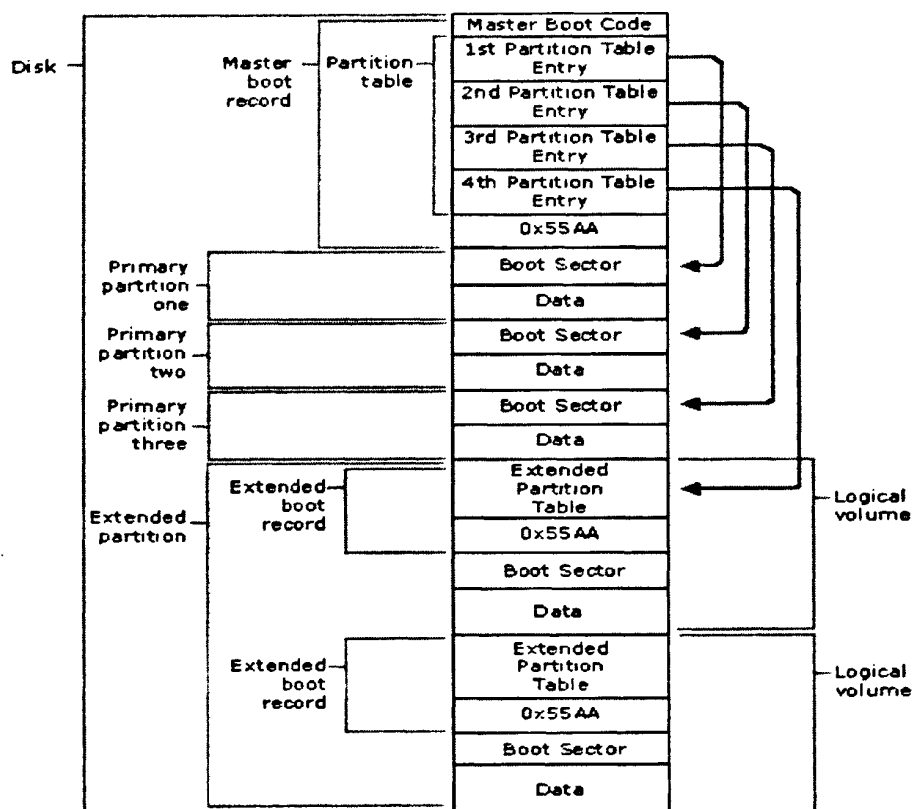


图 3.2 硬盘逻辑结构图

MBR 的数据结构表如表 3.1 所示。

表 3.1 MBR 数据结构表

Address			Description	Size in bytes
Hex	Oct	Dec		
0000	0000	0	Code Area	440(max. 446)
01B8	0670	440	Optional Disk signature	4
01BC	0674	444	Usually Nulls; 0x0000	2
01BE	0676	446	Table of primary partitions (Four 16-byte entries)	64
01FE	0776	510	55h MBR signature;	2
01FF	0777	511	AAh 0xAA55	
MBR, total size: 446 + 64 + 2 =				512

硬盘分区表一般位于 0 磁头、1 扇区，而硬盘主分区表则一般位于 0 柱



面、0 磁道、1 扇区。由主分区表可以推算出其余分区表的位置。分区表占其所在扇区 447 ~ 510 之间的 64 个字节。要判定此 64 字节的数据是否为分区表，则只需要看其后紧邻的两个字节（511 ~ 512）是否为"55AA"，若是，则为分区表。DPT 一般可以标识 4 个分区，每个分区占 16 个字节。如果分区数大于 4，则需要使用扩展分区的格式，在此不予讨论。对于一个分区表，其信息结构如表 3.2 所示。

表 3.2 分区信息格式表

Offset	Length(byte)	Field	Description
0	1	Boot Indicator	00 - inactive; 80 - active
1	1	Starting Head	N/A
2	1	Starting Sector	Bit 0~5 sector number Bit 6~7 upper two bits of Starting Cylinder
3	1	Starting Cylinder	10-bit number, with a max value of 1023
4	1	System ID	Defines the volume type
5	1	Ending Head	N/A
6	1	Ending Sector	Bit 0~5 sector number Bit 6~7 upper two bits of Ending Cylinder
7	1	Ending Cylinder	10-bit number, with a max value of 1023
8	4	Relative Sectors	The offset from the beginning of the disk to the beginning of the volume, counting by sectors
12	4	Total Sectors	The total number of sectors in the volume

### 3.1.2 Guided Partition Table

Guided Partition Table (GPT) 是一种由计算机中的可扩展固件接口 (EFI) 使用的磁盘分区架构。与主启动记录 (MBR) 分区方法相比，GPT 具有更多的优点，因为它允许每个磁盘有多达 128 个分区，支持高达 18EB 字节的卷大小，允许将主磁盘分区表和备份磁盘分区表用于冗余，还支持唯一的磁盘和分区 ID (GUID)。

与支持最大卷为 2 TB (terabytes) 并且每个磁盘最多有 4 个主分区 (或 3 个主分区, 1 个扩展分区和无限制的逻辑驱动器) 的主启动记录 (MBR) 磁盘分区的样式相比, GUID 分区表 (GPT) 磁盘分区样式支持最大卷为 18 EB (exabytes) 并且每磁盘最多有 128 个分区。与 MBR 分区的磁盘不同, 至关重要的平台操作数据位于分区, 而不是位于非分区或隐藏扇区。另外, GPT 分区磁盘有多余的主要及备份分区表来提高分区数据结构的完整性。GPT 的结构如图 3.3 所示。

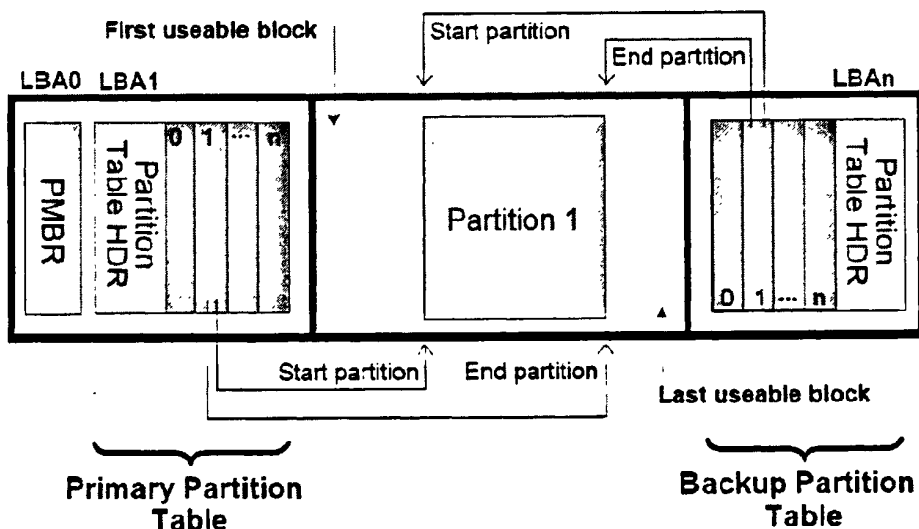


图 3.3 GPT 结构图

GPT 和 MBR 的区别:

1. 64-bit 逻辑块编址;
2. 支持多分区;
3. 通过主分区表和备用分区表实现冗余;
4. 使用版本号和尺寸大小域方便扩展;
5. 使用 CRC32 校验数据完整性;
6. 使用 GUID 唯一标识每一个数据分区;
7. 使用 GUID 及相关参数定义分区类型;
8. 每一个分区由 36 个 Unicode 字符的分区名标识。

## 3.2 Lzma 压缩

Lzma 是 7zip 开源压缩软件中最常用的一种压缩算法，是运用“字典模型”的经典的 LZ 系列算法进行改良、优化后的新版本。Lzma 压缩算法的主要特征是：高压缩比，压缩速率高<sup>[13]</sup>。

DUET 平台通过调用传统 BIOS 提供的 INT 13 中断读写存储镜像文件的磁盘介质。这一特性决定了其可寻址的地址空间限定在了实模式的 1M 地址空间下。而对于传统 BIOS 来说，BDA 和 EBDA 部分之间的地址空间范围是可被用来存储和操作数据的。DUET 平台选定了该区域存放整个平台的镜像文件。

根据 BIOS 标准文档，该区域地址空间范围在 0x500 ~ 0x9F800 之间，而其中从 0x7C00 开始的 512 个字节用于存放 Boot Sector。因此，对 DUET 平台而言，镜像文件的大小是受到限制的。再考虑 0x10000 ~ 0x15000 这段地址空间，用来存放 EfiLdr 文件。以及 0x15000 ~ 0x20000 地址空间用来存放 EfiVar.bin 文件。一般而言，镜像文件存放在 0x20000 ~ 0x9F800 之间 510k 大小的地址空间对于 DUET 平台和 BIOS 标准文档来说都是可行的。

由于直接编译生成的镜像文件一般远大于 510k，需要进行压缩处理。同时，在引导程序加载镜像文件时需要进行解压缩操作。由于 EDK2 平台本身实现有现成的压缩库——TianoCompress。在实际的使用过程中发现，该压缩算法的压缩率不能满足 GCC 编译器所编译出来的镜像文件。在对 GCC 编译器进行了编译选项的优化后依然不能满足小于 510k 大小的需要。因此，舍弃了 EDK2 平台自带的采取一种压缩率更高的压缩算法—Lzma。在使用 Lzma 压缩后，镜像文件压缩前后大小和 TianoCompress 对比如表 3.3 所示。

表 3.3 镜像文件压缩效果对比

编译器	原始尺寸	Tiano 压缩	Lzma 压缩	压缩比提高
MSFT	640 K	253 K	216 K	10.67%
UNIXGCC	2.0 M	427 K	305 K	28.57%
ELFGCC	3.1 M	316 K	221 K	30.06%

从上表可以看出，经过 Lzma 压缩算法压缩后的镜像文件已经完全可以放入 0x20000 ~ 0x9F800 的地址空间中。对于模拟平台来说，往往还需要加

载用户用来调试的模块，在经过压缩后，有约 200k 的空间可供用户使用，能够满足各种用户开发的各种模块的调试。

在实际的 GCC 编译器镜像文件尺寸优化过程中，采取了如下步骤。以一个精简了部分不常用模块的 DUET 平台为例。

其编译器编译及链接选项如下：

```
*_UNIXGCC_IA32_CC_FLAGS = -O2 -fshort-wchar -fno-strict-aliasing -Wall
-Werror -Wno-missing-braces -Wno-array-bounds -c -include AutoGen.h -m32
-malign-double -freorder-blocks -freorder-blocks-and-partition
```

### 1. 移除-O2, -O1

原因：-O2 将会增加 128K 的尺寸 FV 镜像文件尺寸大小，-O1 将会增加 64K 的尺寸 FV 镜像文件尺寸大小。对于 GCC 编译器来说，其编译选项进行的优化主要是针对运行速度，虽然 GCC 手册中申明-Os 编译选项会减少编译后的镜像文件的尺寸<sup>[14]</sup>，但在 DUET 平台中，经过实际测试，不论是对单个模块还是整个 FV 镜像文件，其都不能起到减小尺寸的作用。

### 2. 移除链接选项中的 export-all-symbols

原因：移除后镜像文件的大小将会减小大约 192K。因为 PE32+ 镜像文件在 UNIXGCC 中使用动态链接的格式，其需要导出动态链接库所需要的符号表。但对于 DUET 平台的模块来说，仅仅需要一个模块的入口函数，并且这个入口函数并不会被其他的模块所链接，因此，移除此选项不会对整个镜像文件的运行有影响。实际上，这个编译选项也是 UNIXGCC 和 ELFGCC 编译出来的镜像文件的尺寸大小存在差距的原因之一。

### 3. 在链接选项中增加-s

原因：增加-s 链接选项后，镜像文件的尺寸大小将会减少大约 190K。-s 选项主要用来移除镜像文件中不必要的段，如.comment 段。

### 4. 移除编译选项中的-g

该选项主要用来产生在镜像文件中产生 debug 时所需要的信息。由于 DUET 平台本身就没有打开此编译选项，此处不给出尺寸相关数据。

### 5. 移除编译选项中的-falign-functions -falign-jumps -falign-loops

对于 DuetPkg/Cpu.efi 镜像文件来说，可节约 1K 的空间。

Lzma 压缩算法在 DUET 平台中将被分成两部分，其中压缩算法放入

Build Tool 中, 用于在生成镜像文件时进行压缩操作。解压缩部分则参考 TianDecompress 抽象成代码库, 在引导程序执行时调用相关接口函数执行解压缩操作。

### 3.3 GnuGenBootSector

GnuGenBootSector 是一个用于在 Linux 环境下将 Boot Sector 写入到指定磁盘分区的应用程序。EDK2 中本身有针对 Windows 平台的 GenBootSector 工具, 但 Windows 平台的 GenBootSector 工具其本身不具备跨平台性, 在程序的代码中核心部分都是基于 Windows API 的调用。因此, 需要根据 Linux 平台本身的特点重新编写此工具。

应用程序会根据指定磁盘分区自身的特点判断是否需要处理整个磁盘主引导扇区。一般而言, 如果仅仅写 MBR 到磁盘主引导扇区, 应用程序不会去更新硬盘分区表。而如果需要处理整个磁盘主引导扇区, 则是需要将 MBR 写入到磁盘的第一个分区, 并在当该分区不是活动分区时将其设置为活动分区。

### 3.4 Build Shell Script

在整个工程中, 通过 Linux Shell 脚本操作编译后的文件。主要功能是压缩镜像文件, 生成 EfiLdr, 以及将 Boot Sector 及引导程序写入启动介质。

#### 3.4.1 Post Build

Post Build 的主要功能是生成需要写入到启动介质的 EfiLdr 文件。除去参数解析, 环境变量的设定等 Shell 脚本程序片段外, 核心的程序片段主要包括 2 部分, 压缩镜像文件和生成 EfiLdr。

压缩镜像文件脚本程序片段如下:

```
echo Compressing DUETEFIMainFv.FV ...
$BASETOOLS_DIR/LzmaCompress -e -o $BUILD_DIR/FV/DUETEFIMAINFV.z
$BUILD_DIR/FV/DUETEFIMAINFV.Fv
echo Compressing DxeMain.efi ...
```

```
$BASETOOLS_DIR/LzmaCompress -e -o $BUILD_DIR/FV/DxeMain.z
$BUILD_DIR/$PROCESSOR/DxeCore.efi
echo Compressing DxeIpl.efi ...
$BASETOOLS_DIR/LzmaCompress -e -o $BUILD_DIR/FV/DxeIpl.z
$BUILD_DIR/$PROCESSOR/DxeIpl.efi
```

其中, LzmaCompress 是移植开源 Lzma 压缩算法到 EDK2 平台上的压缩程序。遵循 EDK2 压缩/解压缩算法接口。将 DUETEFIMAINFV.Fv、DxeCore.efi、DxeIpl.efi 文件压缩成 DUETEFIMAINFV.z、DxeMain.z、以及 DxeIpl.z 文件。在压缩完成后, 在通过 EfiLdrImage 应用程序生成 EfiLdr。

以下程序片段以 IA32 平台为例说明如何通过应用生成 EfiLdr。首先将 EfiLoader.efi、DxeIpl.z、DxeMain.z 以及 DUETEFIMAINFV.z 文件通过 EfiLdrImage 应用程序生成 EfiLdr32 文件。然后使用 Linux Shell 下 cat 命令<sup>[15]</sup>将 start.com、efi32.com2 以及刚生成的 EfiLdr32 文件结合起来生成 EfiLdr。

```
if [ $PROCESSOR = IA32 ]
then
$BASETOOLS_DIR/EfiLdrImage -o $BUILD_DIR/FV/EfiLdr32
$BUILD_DIR/$PROCESSOR/EfiLoader.efi $BUILD_DIR/FV/DxeIpl.z
$BUILD_DIR/FV/DxeMain.z $BUILD_DIR/FV/DUETEFIMAINFV.z
cat $BOOTSECTOR_BIN_DIR/start.com $BOOTSECTOR_BIN_DIR/efi32.com2
$BUILD_DIR/FV/EfiLdr32 > $BUILD_DIR/FV/EfiLdr
fi
```

以下是一个简单的 Post Build 命令。

```
./PostBuild IA32
```

### 3.4.2 Create Boot Disk

Create Boot Disk Shell 脚本的主要功能是根据输入参数判断所要写入的磁盘介质是软盘、硬盘、U 盘。然后根据磁盘介质的类型, 写入到磁盘对应的位置。本文以最简单的软盘为例。

#### 1. 格式化软盘

```
## Format floppy disk

umount $EFI_BOOT_MEDIA

mkfs.msdos $EFI_BOOT_DEVICE

mount $EFI_BOOT_DEVICE $EFI_BOOT_MEDIA

2. 生成 Boot Sector

echo Create boot sector ...

$BASETOOLS_DIR/GnuGenBootSector -i $EFI_BOOT_DEVICE -o FDBs.com

$BASETOOLS_DIR/BootSectImage -g FDBs.com

$BOOTSECTOR_BIN_DIR/bootsect.com -f

$BASETOOLS_DIR/GnuGenBootSector -o $EFI_BOOT_DEVICE -i

BOOTSECTOR_BIN_DIR/bootsect.com

3. 将 EfiLdr 以及 DUET 平台的 Shell 复制到软盘中

cp $BUILD_DIR/FV/EfiLdr $EFI_BOOT_MEDIA

mkdir -p $EFI_BOOT_MEDIA/efi

mkdir -p $EFI_BOOT_MEDIA/efi/boot

cp $WORKSPACE/EdkShellBinPkg/MinimumShell/ia32/Shell.efi

$EFI_BOOT_MEDIA/efi/boot/bootia32.efi
```

对于 U 盘和硬盘来说, 创建一个 DUET 的可引导磁盘步骤相对更复杂一些。由于硬盘一般有多个分区, 在创建启动分区时, 需要读分区表, 判断活动分区, 然后写 MBR。以下是一个简单的创建 DUET 可引导磁盘的 Shell 命令, 以软盘为例, 假设格式化为 FAT12 文件系统。

```
CreateBootDisk floppy /media/floppy0 /dev/fd0 FAT12
```

### 3.5 ACPI

ACPI (Advanced Configuration and Power Interface) 高级管理配置与电源接口<sup>[16]</sup>, 是一种 BIOS 向操作系统传递信息并进行交互的接口规范。其主要用途包括:

- 1) 操作系统获得硬件信息: 如中断表和 PCI 设备的硬件拓扑图;
- 2) 操作系统获取硬件状态: 如 LID/Battery 状态;

3) 操作系统控制和操作硬件：如用户在特定一段时间过后关闭显示器或硬盘；

4) 通过电源管理的相关规范提供最优的电源管理和控制；

ACPI 在一个平台中从硬件和软件的角度来看其作用域如图 3.4 所示<sup>[16]</sup>。

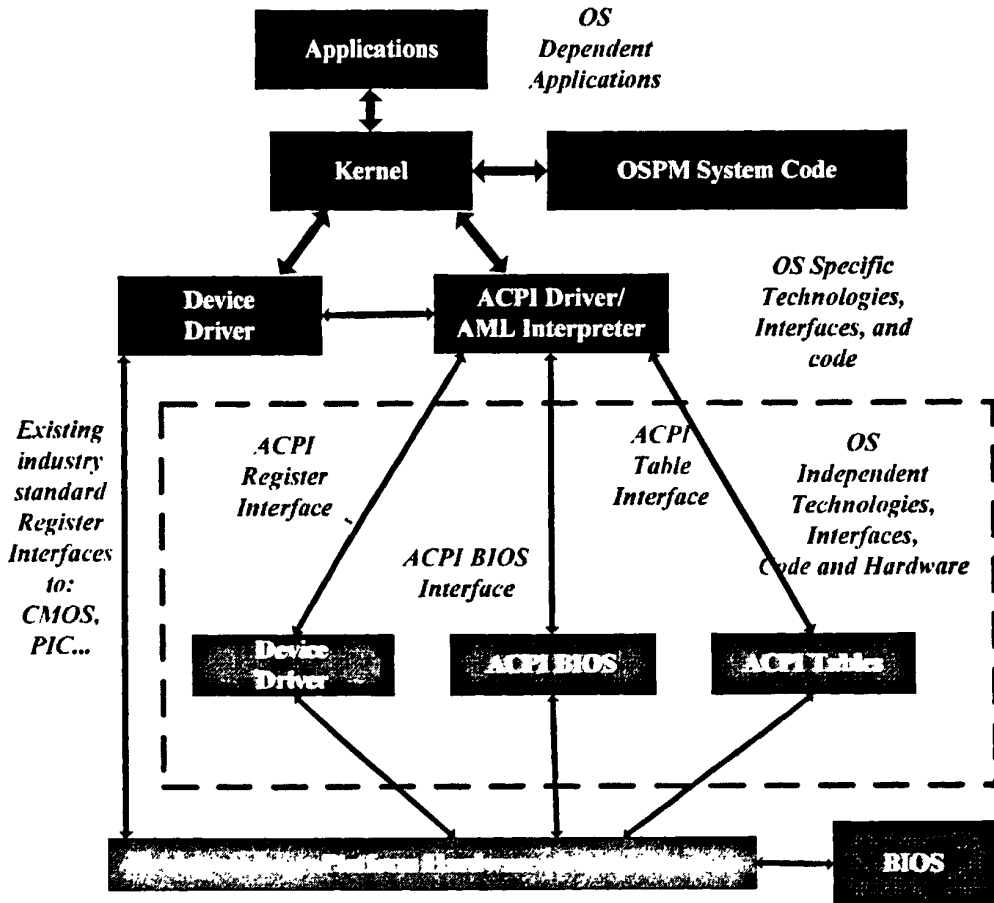


图 3.4 ACPI 在系统中的作用域

从 ACPI 标准文档得知，ACPI 结构一般有多个表组成，一般有如 RSDP, RSDT, FADT, FACS 等表格。在 DUET 平台引导程序中，完全遵照 ACPI 标准文档进行代码实现。包含的表具体如下所示。

- 1) Root System Description Pointer (RSDP)
- 2) Extended System Description Table (RSDT)
- 3) Fixed ACPI Description Table
- 4) Firmware ACPI Control Structure



- 5) Differentiated System Description Table
- 6) Secondary System Description Table
- 7) Multiple APCI Description Table
- 8) Smart Battery Table
- 9) Extended System Description Table
- 10) Embedded Controller Boot Resources Table
- 11) System Locality Distance Information Table
- 12) System Resource Affinity Table

Root System Description Pointer 结构位于系统的内存空间, 由 BIOS 建立, 这个结构中有 Root System Description Table 的地址, RSDT 表中有指向其他表的指针, 这些表向操作系统提供基本系统实现和配置信息。RSDP 以及 RSDT 之间的关系如图 3.5 所示。

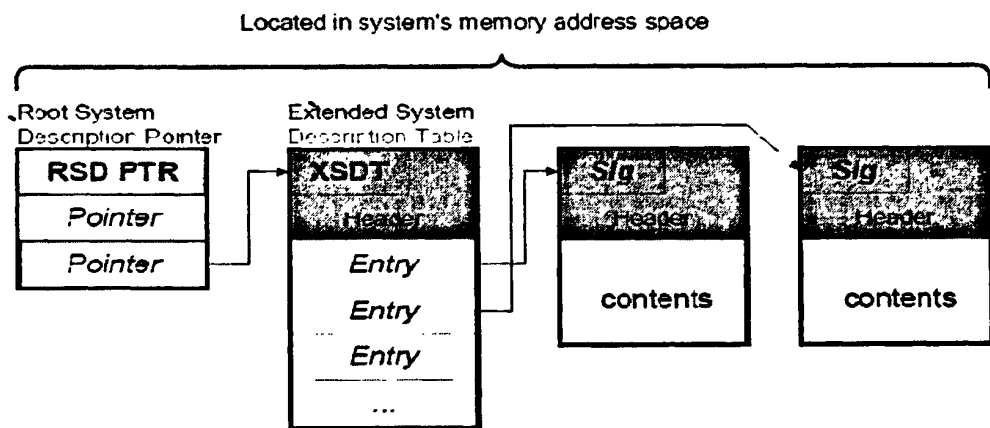


图 3.5 Root System 描述指针及表

RSDP 结构在 DUET 平台中根据定义如下。

```

typedef struct {
    UINT8      Signature[8];
    UINT8      Checksum;
    UINT8      OemId[6];
    UINT8      Revision;
    UINT32     RsdAddress;
}
    
```

```

    UINT32      Length;

    UINT64      XsdtAddress;

    UINT8       ExtendedChecksum;

} RSDP_TABLE;
    
```

所有的描述表都以标识头开始。描述表的基本目的是向操作系统提供实现厂家不同的 ACPI 实现的细节，还提供操作系统需要的直接控制硬件的信息。RSDT 指向内存中的其它 ACPI 表，通常其指向的第一个表为 Fixed ACPI Description Table (FADT)。在 FADT 表中存放着大量固定长度的 Entry，它们用来描述特定硬件的 ACPI 特征<sup>[17]</sup>。FADT 表始终指向差异化的 DSDT 表，在 DSDT 表中，包含着各种各样的系统特征信息。结构关系如图 3.6 所示。

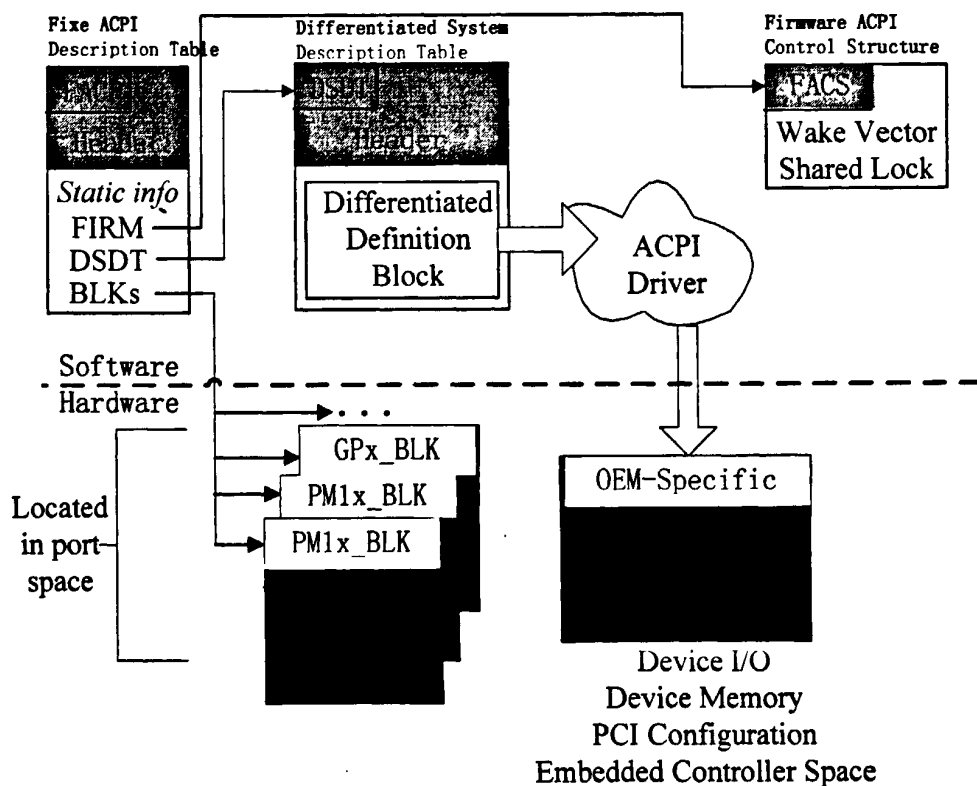


图 3.6 描述表数据结构

DUET 平台引导程序中，寻找 RSDP 的代码片段如下。

```
#define ACPI_RSD_PTR 0x2052545020445352LL

VOID *
FindAcpiRsdPtr( VOID )
{
    UINTN          Address;
    UINTN          Index;

    // Seach 0x0e0000 - 0x0ffff for RSD Ptr
    for (Address = 0xe0000; Address < 0xffff; Address += 0x10) {
        if (*(UINT64 *)(Address) == ACPI_RSD_PTR) {
            return (VOID *)Address;
        }
    }

    return NULL;
}
```

可以看出，在内存地址范围 0x0e0000 ~ 0x0ffff 中寻找 RSDP。对于标准 BIOS 来说，其遵循 ACPI 标准文档规范，RSDP 都会位于此段地址范围内。

### 3.6 平台模块划分

平台模块主要分为 SEC、Dxe Driver、UEFI Driver、以及 Runtime Driver 四种类型。其中 EfiLoader 归于了 SEC 类型，但其并非 UEFI 标准文档规范中所指的 SEC 类型，只是因为其运行在整个平台最初始阶段。具体如表 3 所示。

表 3.4 平台模块详细划分及描述

名字	模块类型	CPU 架构	描述
EfiLoader	SEC	IA32/x64	处理引导扇区镜像
DxeIpl	DXE DRIVER	IA32/x64	产生 HOB, 准备 DXE 阶段需要的 PPIs, 并跳转到 DXE core 的入口地址
PlatformBds	DXE DRIVER	IA32/x64	引导设备选择模块
DataHubGen	DXE DRIVER	IA32/x64	为平台产生 cache, memory data 记录

续表 3.4 平台模块详细划分及描述

名字	模块类型	CPU 架构	描述
PciBusNo-Enumeration	UEFI DRIVER	IA32/x64	为 PCI 总线产生 driver binding protocol 并负责初始化和管理工作 PCI 设备及其数据结构。
PciRootBridge-Enumerate	DXE DRIVER	IA32/x64	初始化 PciRootBridge, 产生 EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL
FsVariable	RUNTIME DRIVER	IA32/x64	在 SimpleFileSystem 的基础上提供模拟的 variable 服务
CpuDxe	DXE DRIVER	IA32/x64	产生并实现 EFI_CPU_ARCH_PROTOCOL
KbcResetDxe	DXE DRIVER	IA32/x64	产生并实现 EFI_RESET_ARCH_PROTOCOL_GUID for
LegacyBiosThunk	DXE DRIVER	IA32/x64	产生并实现 EFI_LEGACY_BIOS_PROTOCOL
IdeController	UEFI DRIVER	IA32/x64	产生并实现 EFI_IDE_CONTROLLER_INIT_PROTOCOL
PcIsaAcpi	UEFI DRIVER	IA32/x64	产生并实现 EFI_ISA_ACPI_PROTOCOL_GUID
8254Timer	DXE DRIVER	IA32/x64	安装 8254 Timer 的中断处理协议 (ISA IRQ0)
LegacyMetronome	DXE DRIVER	IA32/x64	配置并实现 ACPI timer, 产生 EFI_METRONOME_ARCH_PROTOCOL

### 3.7 内存映射

DUET 平台主要由 4 个 EFI 镜像组成。EFI 镜像是可执行的 EFI 程序的静态存储形式。EFI 镜像通常被编译连接成可重定位的代码, 因此能够被装载到系统内存的任意位置。

#### 1. EfiLoader

该镜像文件在引导扇区程序执行完后被加载到内存中, 主要用于解压缩

以下的另外 3 个镜像 (DxeIpl.z、DxeMain.z、EfiMian.z) 并将其加载到内存中的相应位置。EfiLoader 会装载 BFV, 重载 DxeCore 和 DxeIpl, 然后跳转倒 DxeIpl 的入口地址。具体定义见表 3.5。

表 3.5 EfiLoader Module Definitions

Module Name	EfiLoader
Module Type	SEC
Supported Architectures	IA32/x64
GUID	40985DA3-4CD4-47f9-B447-F74D0419921F
Version	1.00
Package Dependencies	MdePkg
Entry Point	EfiLoader

## 2. DxeIpl.z

DxeIpl 为 EFI 框架中 Dxe (Driver execute phase) 阶段之前加载用于建立 Dxe 环境的程序。采用 Lzma 压缩算法压缩, 在引导程序结束后, 将会被 EfiLoader 解压缩并重新加载。DxeIpl 主要用于从收集传统 BIOS 的信息并将其以 HOB (Hand of Block) 的形式传递给其后的 DXE 阶段使用, 运行完成后跳转到 DxeCore 的入口地址。其模块定义如表 3.6 所示。

表 3.6 DxeIPL 模块定义

Module Name	DxeIpl
Module Type	PEIM
Supported Architectures	IA32/x64
GUID	6098E08C-0E54-42ac-9142-47DCAA5C8F77
Version	1.00
Package Dependencies	MdePkg
Entry Point	DxeInit

其需要收集的信息主要包括 Cpu、栈、Bfv、BfvResource、

MemoryAllocation、DxeCore、MemoryTypeInfo、Acpi、Acpi20、Smbios、FlushInstructionCache、PeCoffLoader、MemoryDescriptor、PciExpress 等。类似一张整个系统的状态信息描述表。

### 3. DxeMain.z

该镜像包含 DxeMain 模块，同样采用 Lzma 算法压缩。在 DxeIpl 模块运行结束后，接受其传递过来的 HOB 并运行。

### 4. EfiMain.z

该镜像包含了平台所有的 DXE Driver。主要有由 EfiMain.fv 压缩而成。

以上 4 个镜像都会被装载到低于 1M 地址的内存中，其在内存中的映像如表 3.6 所示。

表 3.6 DUET 平台各模块内存地址分布表

Address Start	Address End	Description(For 32-bit Platform)
0x0	0x400	IVT
0x400	0x500	BDA
0x7c00	0x10000	BootSector
0x10000	0x15000	EfiLdr(EfiLdr relocate by efiXX.COM))
0x15000	0x20000	Efivar.bin (Load by StartXX.COM)
0x20000	0x21000	StartXX.COM (E820 table, GDT, IDT)
0x21000	0x22000	EfiXX.COM (Temporary Interrupt Handler)
0x22000	0x86000	EfiLdr.efi + DxeIpl.Z + DxeMain.Z + BFV.Z
0x86000	0x90000	MemoryFreeUnder1M (For legacy driver DMA)
0x90000	0x9F800	Temporary 4G PageTable for X64 (6 page)
0x9F800	0xA0000	EBDA
0xA0000	0xC0000	VGA
0xC0000	0xE0000	OPROM
0xE0000	0x100000(1M)	FIRMEWARE
0x100000(1M)	0x200000	Temporary Stack (1M)

1M 以上地址空间结构如图所示。描述了 ACPI NVS、ACPI Reclaim、HOB、

NvFV、FtwFV、DxeCore、DxeIPL 等在内存中的结构。

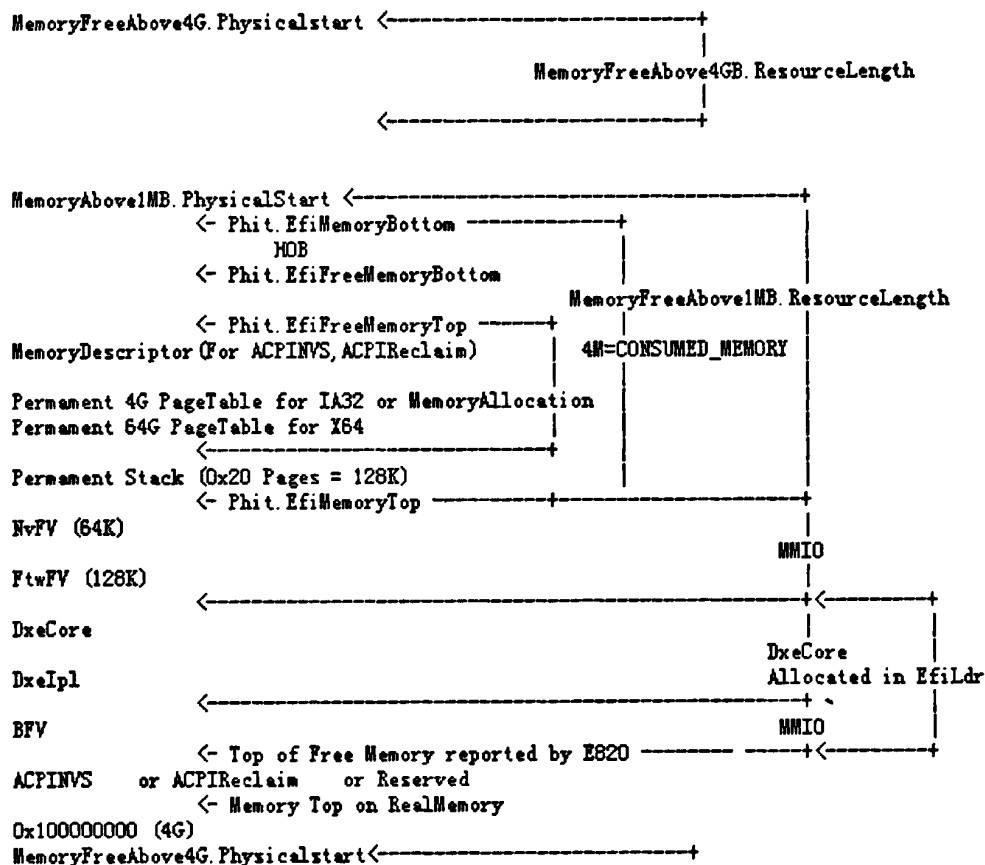


图 3.7 DUET 平台 1M 以上地址空间模块分布图

### 3.8 本章小结

本章从 Boot Sector 程序编写时应考虑的磁盘分区表格式进行展开，研究了 MBR 以及 GPT 格式的磁盘分区表，并给出了相应的数据结构。结合多种编译环境和编译器分析了镜像文件编译后的尺寸并给出了 DUET 平台应采取 Lzma 压缩算法进行压缩的结论。针对 Linux 环境下 Boot Sector 写入可引导磁盘介质的特点，给出了 GnuGenBootSector 工具的相关设计思路。同时，给出了 Linux 环境下编译脚本程序及可引导磁盘生成脚本程序的设计及简单实现。结合 ACPI 标准文档和 BIOS 标准文档，给出了 DUET 平台中应如何查找、使用传统 BIOS 生成的 ACPI 表。通过总结 DUET 平台的特点及其引导程序的特点，提出平台模块划分设计。最后，结合 DUET 平台特点，给出了

整个 DUET 平台的内存映射结构，包括实模式下 1M 地址空间的分配以及跳转到保护模式后 1M 以上地址空间的分配情况。



## 第 4 章 DUET 平台引导程序的实现

本章描述 DUET 平台引导程序的实现, 以平台引导程序运行的先后次序分为 Boot Sector、DxeIPL、EFI Loader 三个部分。并分析了在实际硬件平台上运行的结果。

### 4.1 Boot Sector

Boot Sector 是整个 DUET 平台引导程序运行的第一段代码。下面从功能层面上介绍 Boot Sector 在 DUET 平台引导程序中的实现。在 EDK2 代码库中, 位于 DuetPkg/BootSector/bootsect.S 的代码文件给出了具体实现, 在此给出重要功能的实现代码片段。

#### 4.1.1 BIOS Parameter Block 的实现

对于 Boot Sector, 其中 BIOS Parameter Block (BPB) 数据在代码中仅填写其中的 OemId、SystemId 两个域, 其它域由 GnuGenBootSector 工具在将 Boot Sector 写入到可引导磁盘介质时写入。但在 Boot Sector 的代码编写中应为 BPB 数据预留空间。一般而言, 对 FAT12、FAT16、FAT32 以及 NTFS 格式的磁盘分区, 其 BPB 数据结构均不相同, 在此, 以最简单的 FAT12 格式为例进行说明。FAT12 格式的磁盘分区 BPB 数据结构如表 4.1 所示。

表 4.1 FAT12 BPB Data Structure

Field	Unit	Data	Description	Length	
OemId	.ascii	"INTEL "	# OemId	- 8 bytes	
SectorSize	.word	0	# Sector Size	- 16 bits	
SectorsPerCluster	.byte	0	# Sector Per Cluster	- 8 bits	
ReservedSectors	.word	0	# Reserved Sectors	- 16 bits	
NoFats	.byte	0	# Number of FATs	- 8 bits	
RootEntries	.word	0	# Root Entries	- 16 bits	

续表 4.1 FAT12 BPB Data Structure

Field	Unit	Data	Description	Length	
Sectors	.word	0	# Number of Sectors	- 16 bits	
Media	.byte	0	# Media	- 8 bits	- ignored
SectorsPerFat	.word	0	# Sectors Per FAT	- 16 bits	
SectorsPerTrack	.word	0	# Sectors Per Track	- 16 bits	- ignored
Heads	.word	0	# Heads	- 16 bits	- ignored
HiddenSectors	.long	0	# Hidden Sectors	- 32 bits	- ignored
LargeSectors	.long	0	# Large Sectors	- 32 bits	
PhysicalDrive	.byte	0	# PhysicalDriveNumber	- 8 bits	- ignored
CurrentHead	.byte	0	# Current Head	- 8 bits	
Signature	.byte	0	# Signature	- 8 bits	- ignored
VolId	.ascii	" "	# Volume Serial Number	- 4 bytes	
FatLabel	.ascii	" "	# Label	-11 bytes	
SystemId	.ascii	"FAT12 "	# SystemId	- 8 bytes	

通过 BIOS INT 13h ah=8 调用来获得 BPB 相关数据<sup>[18]</sup>。代码如下所示。

```

movb  $8, %ah          # ah = 8 - Get Drive Parameters Function
movb  %dl, PhysicalDrive(%bp) # BBS defines that BIOS would pass the booting
                                driver number to the loader through DL

int    $0x13           # Get Drive Parameters
xorw   %ax, %ax        # ax = 0
movb   %dh, %al        # al = dh
incb   %al             # MaxHead = al + 1
pushw  %ax             # 0000:7bfe = MaxHead
movb   %cl, %al        # al = cl
andb   $0x3f, %al      # MaxSector = al & 0x3f
pushw  %ax             # 0000:7bfc = MaxSector

```

#### 4.1.2 查找 EfiLdr

对于分区表,由于事先并不知道该 Boot Sector 会被放到哪种文件系统中。不同的文件系统分区表的描述是不相同的,因此,Boot Sector 实现中采取先留空再根据启动介质实际文件系统的不同来写入的方法。Boot Loader 的主要任务(针对 Floppy 启动而言)是调用 BIOS INT 13h 中断读出 EfiLdr。部分代码如下。

FindEFILDR:

```

    cmpl    $LOADER_FILENAME_PART1, (%di)    # Compare to "EFIL"
    jne     FindVARSTORE
    cmpl    $LOADER_FILENAME_PART2, 4(%di)
    jne     FindVARSTORE
    cmpl    $LOADER_FILENAME_PART3, 7(%di)
    jne     FindVARSTORE
    movw    26(%di), %bx                      # bx = Start Cluster for EFILDR
    testw   %dx, %dx
    je      FindNext                          # Efivar.bin is not loaded
    jmp     FoundAll

```

FindNext: # go to next find

```

    addw    $FAT_DIRECTORY_ENTRY_SIZE, %di    # Increment di
    subw    $FAT_DIRECTORY_ENTRY_SIZE, %cx    # Decrement cx
    jne     FindEFILDR

```

#### 4.1.3 查找 EfiVar

在上一节中代码可以看出,当找到 EfiLdr 后,都会跳转到查找 EfiVar 的代码片段—FindVARSTORE。在 DUET 平台中,由于其本身并不操作硬件 Flash,为了加快平台的启动速度。会在平台第一次启动后,将一些平台配置的数据(如启动顺序)存储在启动磁盘介质上一个名为 EfiVar.bin 文件中。当平台再次启动时,直接从此文件中查找平台配置数据从而加快启动速度。查找 EfiVar.bin 文件的代码如下所示。

FindVARSTORE:

```
## if the file is not loader file, see if it's "EFIVAR  BIN"
cmpl $0x56494645, (%di)      # Compare to "EFIV"
jne  FindNext
cmpl $0x20205241, 4(%di)     # Compare to "AR  "
jne  FindNext
cmpl $0x4e494220, 7(%di)     # Compare to " BIN"
jne  FindNext
movw %di, %dx                # dx = Offset of Start Cluster for Efivar.bin
addw $26, %dx
testw %bx, %bx
je   FindNext                # Efldr is not loaded
jmp  FoundAll
```

可以看出，查找程序实际上就是简单的比较文件名，当文件名同 EfiVar.bin 相同时，就认为查找成功，并将文件所在位置保留起来，供 EfiLdr 加载后使用。

## 4.2 EFI Loader

### 4.2.1 跳转保护模式

当找到 EfiLdr 并将其加载到内存中的 0x2000:0000 位置后，首先开始执行的是 Start32.com（假设 CPU 架构为 IA32）。该程序的主要功能是完成实模式向保护模式的跳转。从实模式到保护模式的跳转主要有一下几个步骤：

#### 1. 准备 GDT 及 IDT。

由于准备 GDT 和 IDT 的步骤一般都比较通用，在此不详细介绍。仅给出一个简单的以 GNU 汇编语言描述的空描述符<sup>[19]</sup>。

```
# null descriptor
.equ  NULL_SEL, .-GDT_BASE
.word 0          # limit 15:0
.word 0          # base 15:0
```

```
.byte 0      # base 23:16
.byte 0      # type
.byte 0      # limit 19:16, flags
.byte 0      # base 31:24
```

## 2. 打开 A20。

8086 采用 SEG:OFFSET 的模式进行分段，所以其表示的最大内存为 0xFFFF:FFFF 即 0x10FFFEF。但 20 位的地址线只能寻址到 1MB,从 80286 开始，可以访问 1MB 以上的地址<sup>[20]</sup>。为了访问 1MB 以上的内存空间，需要将 A20 打开<sup>[21]</sup>。代码如下：

```
# Enable A20 Gate
movw $0x2401, %ax      # Enable A20 Gate
int $0x15
jnc A20GateEnabled     # Jump if it succeeded
```

当传统 BIOS 中不支持 INT 15h ax=0x2041 时，将手动打开 A20。简而言之，就是向 8042 控制器的控制端口 0x060 写入命令 0x0df。其代码如下。

```
.equ  DELAY_PORT, 0x0ed      # Port to use for 1uS delay
.equ  KBD_CONTROL_PORT, 0x060  # 8042 control port
.equ  KBD_STATUS_PORT, 0x064   # 8042 status port
.equ  WRITE_DATA_PORT_CMD, 0x0d1 # 8042 command to write data port
.equ  ENABLE_A20_CMD, 0x0df    # 8042 command to enable A20
call  Empty8042InputBuffer     # Empty the Input Buffer on the 8042 controller
jnz   Timeout8042             # Jump if the 8042 timed out
outw  %ax, $DELAY_PORT        # Delay 1 uS
movb  $WRITE_DATA_PORT_CMD, %al # 8042 cmd to write output port
outb  %al, $KBD_STATUS_PORT    # Send command to the 8042
call  Empty8042InputBuffer     # Empty the Input Buffer on the 8042 controller
jnz   Timeout8042             # Jump if the 8042 timed out
movb  $ENABLE_A20_CMD, %al     # gate address bit 20 on
outb  %al, $KBD_CONTROL_PORT  # Send command to 8042
call  Empty8042InputBuffer     # Empty the Input Buffer on the 8042 controller
```

```
movw $25, %cx          # Delay 25us for the command to complete on 8042
```

Delay25uS:

```
    outw    %ax, $DELAY_PORT      # Delay 1 uS
```

```
    loop    Delay25uS
```

3. 加载 GDT、IDT，置 cr0 的 PE 位<sup>[22]</sup>。

```
cli
```

```
lgdt    gdt
```

```
.byte    0x66
```

```
lidt    idtr
```

```
movl    %cr0, %eax
```

```
orb     $1, %al
```

```
movl    %eax, %cr0
```

4. 跳转。

```
jmp 0x00021000
```

此处跳转地址为 0x21000 即跳转到 DuetPkg/BootSector/Efi32.S 继续执行。具体各个模块的地址分布会在下一节 DUET 平台内存映射中进行详细介绍。

#### 4.2.2 中断向量表

在 DUET 平台引导程序中，会重用传统 BIOS 中的中断向量。简而言之，就是将传统 BIOS 的中断向量在 DUET 平台引导程序中再重新加载一遍。DUET 平台引导程序在此基础上做了一些简单的中断异常处理。代码如下。

LOOP\_1: # loop through all IDT exception handlers and initialize to default handler

```
movw    %bx, (%edi)          # write bits 15..0 of offset
```

```
movw    $0x20, 2(%edi)       # SYS_CODE_SEL from GDT
```

```
movw    $(0x0e00 | 0x8000), 4(%edi) # type = 386 interrupt gate, present
```

```
movw    %ax, 6(%edi)         # write bits 31..16 of offset
```

```
addl    $8, %edi             # move up to next descriptor
```

```
addw    DEFAULT_HANDLER_SIZE, %bx # move to next entry point
```

```
loopl   LOOP_1    # loop back through again until all descriptors are initialized
```

### 4.2.3 执行 EFI Loader

在执行 EFI Loader 之前，应找到 EFI Loader 在内存中的镜像文件基地址以及入口点。代码如下。

```
movl $0x22000, %esi      # esi = 22000
movl 0x14(%esi), %eax    # eax = [22014]
addl %eax, %esi          # esi = 22000 + [22014] = Base of EFILDR.C
movl 0x3c(%esi), %ebp    # [22000 + [22014] + 3c] = Image Header for EFILDR.C
addl %esi, %ebp
movl 0x34(%ebp), %edi    # edi = [[22000 + [22014] + 3c] + 30] = ImageBase
movl 0x28(%ebp), %eax    # eax = [[22000 + [22014] + 3c] + 24] = EntryPoint
addl %edi, %eax          # eax = ImageBase + EntryPoint
movl %eax, EfiLdrOffset # Modify far jump instruction for correct entry point
```

接着执行 EfiLdr 的入口函数。该入口函数需要传统 BIOS E820 表中的内存映射表作为参数。首先 DUET 在引导的时候同 OS 的引导相同，必须知道系统物理内存的容量以及分布。因为 DUET 平台没有 PEI 阶段，没有也不必要对内存进行初始化，这样可以既减少 DUET 平台的工作量，又能够加快 DUET 平台的启动速度，这是 DUET 平台在基于传统 BIOS 平台之上运行的一个优势。

在得到了 E820 表的内容后，能够有效地使用和管理这些物理内存。传统的 BIOS 在 INT 15 中提供了三个子中断来获得系统内存大小，分别示 88h, E801h, E802h。88h 方法现在已经不在使用，因为它最大能够识别 64M 的内存。而现在的 PC 机，内存大小一般都在 256M 以上。所以只能用扩展功能即 E802h。中断 E802h 可以获得被安装在主机上的 RAM，以及被 BIOS 所保留的物理内存范围的内存映射。表 4.2 是一个通过 E802h 获得的内存分配表。

表 4.2 传统 BIOS 中 INT 15 EAX=E820 的内存分配表

Description	Address Start	Address End	Usage
BIOS-e820	0x0000000000000000	0x000000000000A000	usable
BIOS-e820	0x000000000000F000	0x0000000000010000	reserved
BIOS-e820	0x0000000000010000	0x000000000001FFF000	usable
BIOS-e820	0x000000000001FFF000	0x000000000001FFF300	ACPI NVS
BIOS-e820	0x000000000001FFF300	0x000000000002000000	ACPI data
BIOS-e820	0x00000000FFFF0000	0x0000000010000000	reserved

以下是查找 E820 并将其写入到 Memory Map 中的一段代码。

MemMapLoop:

```

movl $0xe820, %eax
movl $20, %ecx
movl $0x534d4150, %edx # SMAP
int $0x15
jc MemMapDone
addl $20, %edi
cmpl $0, %ebx
je MemMapDone
jmp MemMapLoop

```

MemMapDone:

```

leal MemoryMap, %eax
subl    %eax, %edi          # Get the address of the memory map
movl    %edi, MemoryMapSize # Save the size of the memory map
xorl    %ebx, %ebx
movw    %cs, %bx           # BX=segment
shll    $4, %ebx           # BX="linear" address of segment base
leal    GDT_BASE(%ebx), %eax # EAX=PHYSICAL address of gdt
movl    %eax, (gdt + 2)     # Put address of gdt into the gdt
leal    IDT_BASE(%ebx), %eax # EAX=PHYSICAL address of idt

```



```

movl    %eax, (idtr + 2)          # Put address of idtr into the idtr
leal    MemoryMapSize(%ebx), %edx # Physical base address
addl    $0x1000, %ebx             # Source of EFI32
movl    %ebx, JUMP+2
addl    $0x1000, %ebx
movl    %ebx, %esi                # Source of EFILDR32
    
```

其中, memory map 的地址在通过查找 E820 表查找出来后, 作为 EfiLdr 的入口函数的参数。下图为在 Bochs 中进行调试时, EfiLdr 在执行时打印出的调试信息。在图 4.1 中, 可以看到在获得内存映射的基地址后, DUET 引导程序将会解压缩镜像文件。

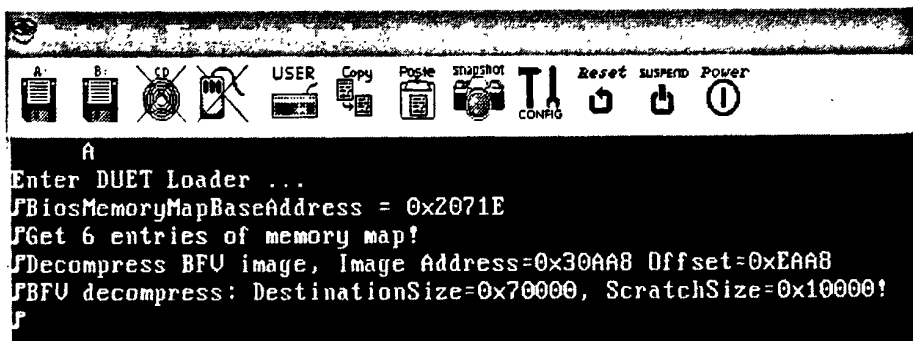


图 4.1 DUET 中通过 E820 表获得的内存映射表基地址

由于镜像文件的压缩采用了上文中提到的Lzma压缩算法, 在此解压缩也必须使用Lzma解压缩算法。为了使用方便, 将其抽象成2个函数, 其函数原型如下。其中LzmaUefiDecompressGetInfo()函数用来获得将要解压缩的镜像文件解压缩后所需要的空间大小, 以及解压缩过程中所需要的缓存大小。LzmaUefiDecompress()函数根据调用LzmaUefiDecompressGetInfo()函数后得到的信息进行解压缩。

RETURN\_STATUS

EFIAPI

LzmaUefiDecompressGetInfo (

IN CONST VOID \*Source,

IN UINT32 SourceSize,

```

    OUT UINT32      *DestinationSize,
    OUT UINT32      *ScratchSize
);
RETURN_STATUS
EFIAPI
LzmaUefiDecompress (
    IN CONST VOID   *Source,
    IN OUT VOID     *Destination,
    IN OUT VOID     *Scratch
);

```

在 EFI Loader 执行过程中，会将 Memory Map 的信息转换成符合 UEFI 标准文档规范的 EFI 格式的内存描述表。图 4.2 是 DUET 平台运行后的内存信息截图。

EFI Memory Descriptors			
Type = 00000007	Start = 00000000	NumberOfPages = 0000009F	
Type = 00000000	Start = 0009F000	NumberOfPages = 00000001	
Type = 00000000	Start = 000E8000	NumberOfPages = 00000018	
Type = 00000007	Start = 00100000	NumberOfPages = 00001E45	
Type = 00000009	Start = 01FF0000	NumberOfPages = 00000010	
Type = 00000000	Start = FFFC0000	NumberOfPages = 00000040	
Type = 00000006	Start = 01F60000	NumberOfPages = 00000090	
Type = 00000005	Start = 01F5B000	NumberOfPages = 00000005	
Type = 00000006	Start = 01F5A000	NumberOfPages = 00000001	
Type = 00000005	Start = 01F47000	NumberOfPages = 00000013	
Type = 00000006	Start = 01F45000	NumberOfPages = 00000002	

图 4.2 DUET 平台运行时内存信息

在整个 EFI Loader 执行完后，会跳入到 DxeIPL 的入口地址继续执行。其中 DxeIPL 阶段所需 EFI Loader 阶段所得到的平台信息以 Hand Off Block (HOB) 的数据结构传入到 DxeIPL 阶段。HOB 数据结构定义如下。

```

typedef struct _EFILDRHANDOFF {
    UINTN                      MemDescCount;
    EFI_MEMORY_DESCRIPTOR     *MemDesc;
    VOID                      *BfvBase;
    UINTN                      BfvSize;
};

```

```

VOID                *DxeIplImageBase;
UINTN               DxeIplImageSize;
VOID                *DxeCoreImageBase;
UINTN               DxeCoreImageSize;
VOID                *DxeCoreEntryPoint;

} EFILDRHANDOFF;
    
```

在 EFI Loader 的最后阶段，跳转到 DxeIPL 的代码如下。

```

if (DxeIplImage.EntryPoint != NULL) {
    Handoff.MemDescCount    = NumberOfMemoryMapEntries;
    Handoff.MemDesc         = EfiMemoryDescriptor;
    Handoff.BfvBase         = (VOID *) (UINTN) BfvBase;
    Handoff.BfvSize         = BfvPageNumber * EFI_PAGE_SIZE;
    Handoff.DxeIplImageBase = (VOID *) (UINTN) DxeIplImage.ImageBasePage;
    Handoff.DxeIplImageSize = DxeIplImage.NoPages * EFI_PAGE_SIZE;
    Handoff.DxeCoreImageBase = (VOID *) (UINTN) DxeCoreImage.ImageBasePage;
    Handoff.DxeCoreImageSize = DxeCoreImage.NoPages * EFI_PAGE_SIZE;
    Handoff.DxeCoreEntryPoint = (VOID *) (UINTN) DxeCoreImage.EntryPoint;
    EfiMainEntrypoint = (EFI_MAIN_ENTRYPOINT) (UINTN) DxeIplImage.EntryPoint;
    EfiMainEntrypoint (&Handoff);
}
    
```

DxeIPL 运行时需要有 Memory、Bfv、DxeIplImage 以及 DxeCoreImage 的相关信息都通过 HOB 传递到 DxeIPL 镜像文件的入口函数。其中一些 HOB 的信息如图 4.3 所示。

```

DxeIpl Handoff Information:
Handoff.BfvBase = 00000000001F60000
BfvLength = 0000000000090000
Handoff.DxeIplImageBase = 00000000001F5A000
DxeIplImageSize = 0000000000006000
Handoff.DxeCoreImageBase = 00000000001F45000
DxeCoreImageSize = 0000000000015000
    
```

图 4.3 Content of Hand Off Block for DxeIPL

### 4.3 DxeIPL

在 DxeIPL 阶段，需要完成如下工作。

1. 生成 CPU HOB 信息
2. 生成 BFV HOB 信息
3. 生成内存 HOB 信息
4. 生成非易失存储设备 (NV, 如 Flash) 信息
5. 准备 DxeCore 运行需要的内存 HOB 信息
6. 获得 ACPI 表、SMBIOS 表等相关信息
7. 跳转 DxeCore

图 4.4 为在 DxeIpl 执行过程中，根据函数调用的先后顺序打印出的执行信息，反应了 DxeIPL 的执行流程。

```
Enter DxeIpl ...
Prepare Cpu HOB information ...
Prepare BFV HOB information ...
Prepare Memory HOB information ...
Prepare NV Storage information ...
NV Storage Base=0x7F14000F
Stack Top=0x7F14000, Stack Bottom=0x7EF4000F
Prepare DxeCore memory Hob ...
F
AcpiTable=0xFA5D0 SMBIOS Table=0x0 MPS Table=0x0F
HobStart=0x2F14000F
Memory Top=0x7F14000, Bottom=0x2F14000F
Free Memory Top=0x7EF1000, Bottom=0x2F15058F
Nv Base=0x7F14000, Length=0x20000F
```

图 4.4 DxeIPL 运行信息

对于各个 HOB 信息的生成，其流程大体一致，在第 3 章中也有对如何获得 ACPI 表信息进行了描述。因此，在此不一一介绍。仅简单介绍 CPU HOB 信息的生成过程。简而言之，整个 DxeIPL 的执行过程，就是在填充如下一个名为 HOB\_TEMPLATE 的数据结构，由于此数据结构过于庞大，在此仅给出其部分数据项。因为此数据结构中有数据项在程序编译时就可以对其进行初始化，所以将其定义为一个模版结构，避免被重复初始化。

```
typedef struct {
    EFI_HOB_HANDOFF_INFO_TABLE    Phit;
    EFI_HOB_FIRMWARE_VOLUME       Bfv;
    EFI_HOB_RESOURCE_DESCRIPTOR    BfvResource;
    EFI_HOB_CPU                   Cpu;
    TABLE_HOB                    Acpi;
    TABLE_HOB                    Acpi20;
    TABLE_HOB                    Smbios;
    TABLE_HOB                    Mps;
    MEMORY_DESC_HOB               MemoryDescriptor;
    ACPI_DESCRIPTION_HOB          AcpiInfo;
    EFI_HOB_RESOURCE_DESCRIPTOR    NvStorageFvResource;
    FVB_HOB                       NvStorageFvb;
    FVB_HOB                       NvStorage;
    ....
} HOB_TEMPLATE;
```

对于CPU HOB而言，其数据结构定义如下。在信息生成的过程中，仅需要填充其SizeOfMemorySpace域即可。该域标识了处理器可寻址的最大物理内存地址。

```
typedef struct {
    EFI_HOB_GENERIC_HEADER  Header;
    UINT8                   SizeOfMemorySpace;
    UINT8                   SizeOfIoSpace;
    UINT8                   Reserved[6];
} EFI_HOB_CPU;
```

在引导程序中生成CPU HOB信息的代码如下。其主要是调用CpuID指令获得CPU的信息，从而得到处理器可寻址的最大内存地址，并将其写入HOB\_TEMPLATE中。

```

VOID PrepareHobCpu ( VOID )
{
    EFI_CPUID_REGISTER      Reg;

    UINT8                    CpuMemoryAddrBitNumber;

    // Create a CPU hand-off information

    CpuMemoryAddrBitNumber = 36;

    AsmCpuid (EFI_CPUID_EXTENDED_FUNCTION,
               &Reg.RegEax, &Reg.RegEbx, &Reg.RegEcx, &Reg.RegEdx);

    if (Reg.RegEax >= CPUID_EXTENDED_ADD_SIZE) {
        AsmCpuid (CPUID_EXTENDED_ADD_SIZE,
                   &Reg.RegEax, &Reg.RegEbx, &Reg.RegEcx, &Reg.RegEdx);

        CpuMemoryAddrBitNumber = (UINT8)(UINTN)(Reg.RegEax & 0xFF);
    }

    gHob->Cpu.SizeOfMemorySpace = CpuMemoryAddrBitNumber;
}
    
```

图4.5为DxeIPL阶段运行结束后，HOB\_TEMPLATE数据结构被填充完成，其所包含的信息。

```

Hob Info
Phit.EfiMemoryTop      = 000000007F21000
Phit.EfiMemoryBottom   = 0000000002F21000
Phit.EfiFreeMemoryTop  = 0000000007EFE000
Phit.EfiFreeMemoryBottom = 0000000002F22058
Bfv                    = 0000000007F6D000 Length = 0000000000083000
NoStorageFvb           = 0000000007F21000 Length = 000000000010000
NoFtwFvb               = 0000000007F31000 Length = 000000000020000
BfvResource            = 0000000007F6D000 Length = 0000000000083000
NoStorageFvbResource   = 0000000007F21000 Length = 000000000010000
NoStorage               = 0000000007F21048 Length = 000000000004000
NoFtwFvbResource        = 0000000007F31000 Length = 000000000020000
NoFtwWorking            = 0000000007F31048 Length = 000000000002000
NoFtwSpare              = 0000000007F33048 Length = 000000000010000
Stack                  = 0000000007F01000 Length = 000000000020000
PageTable               = 0000000007EFF000
MemoryFreeUnder1MB      = 0000000000086000
MemoryFreeUnder1MBLength = 0000000000019800
MemoryAbove1MB          = 0000000000100000 MemoryAbove1MBLength = 0000000007E21000
MemoryAbove4GB          = 0000000000000000 MemoryAbove4GBLength = 0000000000000000
DxeCore                 = 0000000007F51000 DxeCoreLength = 000000000001C000
MemoryAllocation        = 0000000007EFE000 MemoryLength = 0000000000083000
    
```

图 4.5 DxeIPL HOB 信息

此HOB信息产生后，即为整个DUET平台引导程序的工作结束，将跳入

到DxeCore镜像文件的入口地址执行DUET平台DXE Core阶段的代码。

## 4.4 实验结果与分析

### 4.4.1 实验环境及工具

工具包括如下。

1. 虚拟机: QEMU 0.9.0 For Windows

Bochs 2.3.7

VmWare WorkStation 6.0.4

2. 硬件: ThinkPad T61。

3. 镜像生成相关:

1) EDK2相关代码, 版本R10156。

<https://edk2.svn.sourceforge.net/svnroot/edk2/trunk/edk2>

2) 操作系统Ubuntu8.10, GCC4.4.3。

### 4.4.2 过程及结果分析

生成DUET平台镜像的过程如下。假设在开发机已经导出EDK2代码库到/Edk2\_Tree目录下, 编译IA32 CPU架构的DUET平台。

1. 准备编译工具。此处可参考DUET平台ReadMe文件, 不做详细说明。

2. 编译DUET平台。

1) 进入Edk2\_Tree运行edksetup.sh BaseTools。

2) 执行命令“build -p DuetPkg/DuetPkg.dsc -a IA32 -t UNIXGCC”。

3) 进入Edk2\_Tree/DuetPkg目录。

4) 执行命令./PostBuild.sh IA32。

5) 执行命令./CreateBootDisk.sh floppy /media/floppy0 /dev/fd0 FAT12

经过以上步骤后, 会得到一个含有DUET镜像文件的可引导软盘。可以在虚拟机或者硬件平台上对其进行测试。由于所有虚拟机的运行结果一致, 在此仅给出在QEMU中启动后截图如图4.6。

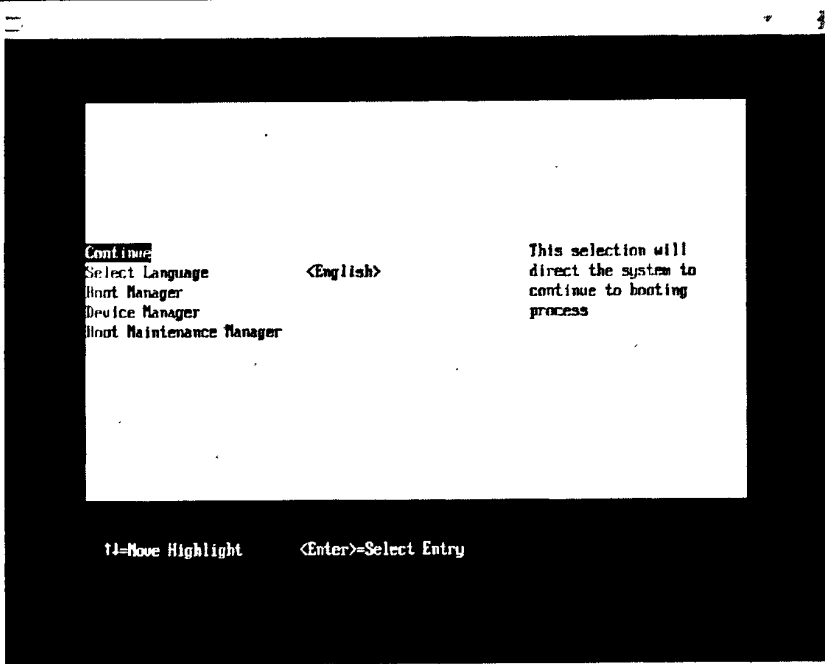


图 4.6 QEMU 中运行 DUET 平台镜像

在 IBM T61 中运行后截图如图 4.7 所示。

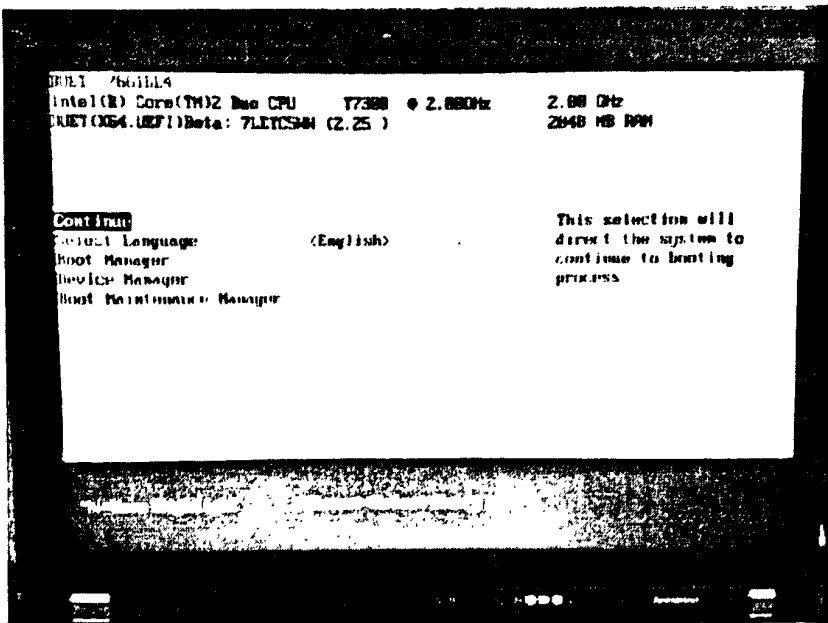


图 4.7 ThinkPad T61 中运行 DUET 平台镜像

可以看出，DUET平台引导程序在Linux环境下通过编译并成功引导 DUET平台运行在虚拟机以及硬件平台上，达到了预期目标。



## 4.5 本章小结

本章详细描述了 DUET 平台引导程序核心模块的实现。按平台运行时各模块运行时间顺序进行展开。分为 Boot Sector、DxeIPL、EFI Loader 共 3 个部分。分别介绍了各自模块中核心部分如 BPB、查找 EfiLdr、查找 EfiVar、跳转保护模式的实现方法、并给出了相关的实现代码。最后结合各种实验平台对 DUET 平台进行了测试、并给出了相关的测试结果。

## 结 论

通过近半年的努力,本文在阅读了大量关于新一代 BIOS 平台和传统 BIOS 平台的相关文献的基础上,深入分析和研究了一种在 Linux 环境下基于 Intel 提出的新一代的 BIOS 平台引导程序的设计目标和实现方法,提出了一种在传统 BIOS 平台环境下进行 UEFI 驱动和应用程序调试及开发的途径,起到了一种衔接传统 BIOS 和新一代 BIOS 的承上启下的桥梁作用。具有很强的创新性和实用价值。

完成了设计文档中所提出的 DUET 平台引导程序的的设计和编码工作,代码质量达到了 M4 标准。主要工作包括 UEFI Framework 以及 DUET Framework 的研究,Boot Sector 的设计及编写,DxeIPL 及 EFI Loader 模块的研究及编写,项目相关编译工具及脚本的设计及编写,以及在各种虚拟机及硬件平台上的测试及调试。

项目进行过程中遇到了许多不同方面的问题,包括编码和设计方面都遇到了一定的困难。由于 DUET 平台引导程序基于 Linux 平台进行编译,为了更大程度的兼用 GCC 编译器,采用不常用的 GNU 汇编语言进行开发,资料稀少增大了项目开发的难度。而在软件调试的过程中,由于 Bochs、Vmware、QEMU 等虚拟机软件所使用的传统 BIOS 并不都完全遵循 BIOS 标准文档规范,经常遇到无法预估的调试和运行问题,在硬件平台上基于 Intel ITP 硬件调试器的调试也会遇到硬件断点无法停下、无法步进等较难解决的问题,但都在不断的探索和测试中一一解决。多平台的测试也提高了代码的质量、增强了软件的健壮性。

为了兼容 EDK2 开发包不断更新的版本,项目采取了定期更新和测试重用的 EDK2 开发包模块,在一定程度上降低了风险。由于 DUET 平台要兼容 IA32 和 X64 两种 CPU 架构,不同的汇编语言指令集也增加了本文的工作难度。

总体而言,此项目的从设计到开发达到了预期的目标,通过了UEFI SCT测试平台的测试,同时兼容IA32和X64两种CPU架构,不仅可以稳定的运行在基于传统BIOS的硬件平台上,也可以运行在包括Vmware、QEMU、Bochs、VBox等多种虚拟机平台上。DUET平台引导程序项目代码及相关编译工具及脚本程序已包含在EDK2开发包并通过开源方式以BSD license发布在<http://www.tianocore.org> 站点。自DUET平台发布以来,获得了BIOS软件开发工程师及研究人员的肯定,并被广泛用于BIOS硬件驱动的开发及调试中。同时,也得到了各方面的反馈意见。根据反馈和意见,对DUET平台引导程序相关部分进行了修改和测试。无论是从代码的规范程度还是文档的完善程度而言,都达到了用户需求书中的要求。当然,由于开发周期短,开发时间紧,程序必然在各方面存在一定的问题。将在程序在被使用的过程中根据用户的反馈信息,对程序进行更全面的完善和修改,以使其更稳定、可靠。

## 参考文献

- [1] Intel,Microsoft,ed. Unified Extensible Firmware Interface Specification. V2.3. 2009:12-15P
- [2] EFI 接口 BIOS 驱动体系的设计、实现与应用. 石浚菁. 2006 年 3 月.
- [3] Intel,Microsoft,Apple,ed. UEFI Platform Initialization Specification-Pre- EFI Initialization Core Interface. Intel Corporation. V1.2. 2008. 132-133P
- [4] Intel,Microsoft,Apple,ed. UEFI Platform Initialization Specification—Driver Execution Environment Core Interface. Intel Corporation. V1.2. 2008. 56-60P
- [5] Vincent Zimmer, Michael Rothman, Robert Hale. Beyond BIOS: Implementing the Unified Extensible Firmware Interface with Intel's Framework. Intel Press. 2006. 70-71P
- [6] Ying Hailu, Li-Ta Lo, Gregory R. Watson, Ronald G. CAR: Using Cache as RAM in Linux BIOS.2000. 1-3P
- [7] Oney, Walter. Programming the Microsoft Windows Driver Model. Microsoft Press. 2005:35-40P
- [8] Intel Corporation. EDKII Module Development Environment Package Library Specification. 2009:12-40P
- [9] 毛德操,胡希明. Linux 内核源代码情景分析(上、下). 浙江大学出版社,2001:38-45 页
- [10] 孙国斌. 使用 Linux LILO 实现操作系统多重引导. 计算机应用研究. 2001 年第 8 期:149-150 页
- [11] 王 静,刘夏伟. 基于 Linux 的嵌入式系统的启动设计. 电子科技 2004 年第 6 期(总第 177 期): 3-6 页
- [12] 邓剑,杨晓非,廖俊卿. FAT 文件系统原理及实现.计算机与数字工程. 2005 年第 33 卷第 9 期:105-108 页
- [13] 于振生,李蓉等. LZMA 压缩算法在固件下载升级中的应用.电子测量技

术.2006年10月,29卷第5期:136页

- [14] Arthur Griffith. GCC: The Complete Reference. McGraw-Hill Osborne Media.1 edition.2002: 397-422P
- [15] 倪莹,蔡杰,李莹. 基于 GCC 的嵌入式系统开发环境构造. 计算机工程与应用.2004. 32.
- [16] 牟树波. ACPI 标准的实现技术.中国科学院计算技术研究所学位论文.1998:32-33 页
- [17] HP,Intel,Microsoft,ed. Advanced Configuration and Power Interface Specification. V3.0. 2004. 20-21P, 99-100P
- [18] Compaq, Phoenix, Intel. BIOS Boot Specification. V1.01. 1996. 19-21P
- [19] W. RICHARD STEVENS. Advanced Programming in the UNIX Environment. ADDISON-WESLEY Press.2005:225-226P.
- [20] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture. Intel Corporation. 2008:35-48P
- [21] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2: Instruction Set Reference. Intel Corporation. 2008:125-133P
- [22] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3: System Programming Guide. Intel Corporation. 2008:385-401P

## 攻读硕士学位期间发表的论文和取得的科研成果

- [1] 吴艳霞, 桂坤, 朱东亮. XX 军工项目. VxWorks 文本显示系统. 2008-2009
- [2] 吴艳霞, 桂坤, 王吉发. Intel 大学合作计划项目. Pre-Boot 环境下基于云安全的漏洞扫描引擎. 2009-2010

## 致 谢

在我的硕士论文完成之际，谨向在我攻读硕士学位期间所有帮助、指导、关心过我的老师、同学、朋友和亲人致以最真挚的感谢！

首先我要衷心感谢顾国昌教授、吴艳霞老师。本课题的选题、设计到论文的撰写的整个过程，两位老师给予了孜孜不倦的指导，多次询问进度，指导我的开发思路，精心点拨。正是两位老师在整个毕业设计过程中的耐心指导，才使我顺利完成了毕业设计和本论文。两位老师渊博的知识、严谨的治学态度和认真负责的工作作风都深深地影响着我。在此，我向两位老师致以最诚挚的谢意！

感谢 Intel 亚太研发有限公司 Tiano Team 的 Huang Qing、Chen Stanley、Lu Ken、Long Qin、Gao Penny、Gao Liming 等在工作以及生活中的指导与帮助，他们实事求是的工作作风、一丝不苟的工作态度深深的影响了我。特别要感谢他们在我毕业设计进行的过程中不厌其烦的指导和无私的帮助，才使得我顺利的完成了毕业设计。

感谢一起学习和生活的同学纪佳程、刘江、张喆、张强、周阳、朱东亮、林志强，与他们在一起攻克了诸多项目难题，除了自身能力得到提高外，还收获了珍贵的友谊，与他们在一起共同学习和生活的日子为我的研究生生活带来了许多快乐和回忆。

最后，再次向所有曾经帮助、指导和关心过我的老师、同学以及我的亲友表达最诚挚的谢意。谢谢大家！