论文题目一种在 UEFI BIOS Shell 下除错工具的设计与实现

专业学位类别	工　程　硕　士
学　　号	200992231181
作 者 姓 名	张　琼
指 导 教 师	段贵多　博　士

分类号 _____ 密级 _____

UDC [注1] _____

学 位 论 文

一种在 UEFI BIOS Shell 下除错工具的设计与实现

（题名和副题名）

张 琼

（作者姓名）

指导教师 段贵多 博 士

电子科技大学 成 都

贾慧鹏 高 工

上海宇烨电子科技有限公司 上 海

（姓名、职称、单位名称）

申请学位级别 硕士 专业学位类别 工程硕士

工程领域名称 软 件 工 程

提交论文日期 2012.09 论文答辩日期 2012.11

学位授予单位和日期 电子科技大学 2012 年 12 月 30 日

答辩委员会主席 _____

评阅人 _____

注 1：注明《国际十进分类法 UDC》的类号。

DESIGN AND IMPLEMENTATION ON THE UEFI BIOS SHELL UNDER DEBUGGING TOOLS

A Thesis Submitted to

University of Electronic Science and Technology of China

Major: Software Engineering

Author: Zhang Qiong

Advisor: Duan Guiduo

School : School of Computer Science & Engineering

独 创 性 声 明

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

作者签名：_____ 日期： 年 月 日

论文使用授权

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

作者签名：_____ 导师签名：_____

日期： 年 月 日

摘要

随着个人计算机的需求量不断增加，计算机设计和制造厂家也不断增加，使得各厂家之间竞争更加激烈，为了加快出货速度，各厂家不断改进研发或维修测试除错时间来加快出货速度。计算机除错工具控制的关键——底层硬件资源的访问是整个计算机系统开发的核心技术。实现计算机底层硬件资源访问共有三个方法，分别是计算机访问端口的简单可操作性、除错工具内嵌 BIOS 或外部调用的方法和计算机系统在还没进入操作系统前进行硬件的除错技术。

本文主要通过对计算机底层硬件设备的访问和 BIOS 的研究，提出了一种创新的方法，即在计算机开机后在 UEFI BIOS Shell 环境下进行功能除错的方法，与现存的除错方式相比，它具有下述显著特点：不借助操作系统，进行访问计算机底层硬件资源情况的软件方法。这种方法实现了计算机底层硬件资源在不需要借助任何操作系统的情况下完成信息访问，使用户更加节省系统操作时间：如 PCI 设备的配置寄存器，键盘控制器，SMBus 等，并且提供了一种可视化的用户接口。为更深层次的了解系统硬件的工作原理，开发或调试计算机系统的软硬件或计算机系统硬件问题的解决提供了更加便捷的除错方式，节约了计算机生产测试环节中的时间和外存治具费用，使在 Shell 中执行除错提供了一种新的方法，同时，它也将是下一个技术发展趋势，本论文第八章进行与现有方式对比的详细性能分析。

关键词：测试除错，UEFI BIOS，BIOS Shell，系统管理总线，PCI 设备

ABSTRACT

With the increasing demand for personal computer, the computer manufacturer increases constantly too, make the competition between manufacturers more intense, in the constant improvement and development or maintenance test debug time to speed up the delivery. The computer system control key - -hardware resources access is the entire computer system development core technology. There are three methods to realize accessing computer hardware resource, respectively is visual access interface, firmware, realization method and prior to operating system hardware access technology.

In this thesis,mainly through tools to access of the computer hardware equipment, puts forward a functional debug method under UEFI BIOS Shell environment when computer power on, realize without the help of operating system, to access the computer hardware resources situation of software method. This method realized the computer development or maintenance testers in do not need to operating system under the premise of deeper levels of access to low-level hardware resources: such as PCI device configuration register, the keyboard controller, etc., and provides a visual user interface. For the development of the computer system, in-depth understanding of the working principle of the computer hardware, and then to solve the design and maintenance of the problems appeared in the process to provide a convenient, save the computer production testing manpower and peripheral storage fixture cost, make Shell in perform debug provides a new method, at the same time, it also will be a generated technology development trend，This thesis chapter 8 with existing way contrast detailed performance analysis..

KEY WORDS: Testing Debug，UEFI BIOS, BIOS Shell，SMBus, PCI Function

目录

第一章 引言

1.1 研究背景与意义

近几年随着我国经济的对外开放，经济水平不断提升，特别是在 IT 信息产业的需求不断加大，个人计算机发展也非常迅速，人们对原先的桌上型电脑，现在追求的是小而轻的笔记型电脑，各行各业对计算机需求，使得计算机技术快速发展并向电脑市场提出更高的要求，同时由于市场需求也有更多的电脑生产商加入生产的团队，来争取各自的市场份额。与此计算机市场需求相适应，各种利用计算机的进化系统纷纷出现，如普通的计算机已经从又大重的“砖头机”进化成小而美的上网本，发展到现在的超级本，计算机被不停的实现创新，以满足市场对计算机需求在外观、性能、重量、价格。在目前生活中物廉价美的说法已经行不通了，由于上面已经提到的人们经济水平的提高，大家都希望优越的生活，所以只要舒适、方便、可行，贵一点计算机用户也是非常乐意投资的，如苹果电脑很贵，但需求还是很大，就因为计算机设计商抓住了消费者的心理，设计出了别人没有的品质和外观，导致它在计算机市场中始终产能和利润双丰收。

综合需求和生产的竞争，计算机生产商对生产和售后服务成本就越加关心，在品质不变的情况下追求低价格材料和廉价劳动力，在材料成本已经压缩到一个极致的情况下，大家都聚焦在人力成本上，借着各个生产作业标准来规范测试作业, 使测试作业有所依据及标准化，并确保测试的施行皆能依据质量方针判定标准，对品保政策予以明确管制来提高品质减少返工测试时间，确立产品质量的测试标准化及可追溯性[7]。

针对在提高品质的情况下缩短测试除错时间，所以本文的研究对研发和测试有了更好的思路，它在研究 UEFI BIOS Shell 下的计算机硬件除错实现方式就是为了服务计算机系统底层开发或计算机电脑维修测试而设计的可视化系统，方便电脑生产商测试除错，加快了电脑的研发和测试，从而缩短生产商交货时间，提高竞争力。在竞争如此激烈的情况下，就是与时间赛跑，所谓谁能先占有市场，攒取第一桶金。

软件开发人员对各种底层硬件资源分配情况进行数据访问后进行得到计算机硬件的除错信息，譬如可以查看计算机内存信息、PCI 总线上的设备信息、CPU 的配置寄存器、CMOS 信息、USB 设备信息等，有时还可以利用外部的设备或治具来测

试 RJ45、Card Reader、Wireless 等设备的运行功能是否正常。通常情况下，除错程序都是借用 DOS 或 Windows 操作系统进行访问查看计算机设备状态或信息，碰到操作系统出现崩溃或存放操作系统的硬盘损坏，除错系统将无法执行功能，同时需要更多的测试时间。系统开发者遇到这种情形下时通常会不知所措或需要大量的等待时间。在整个测试过程中 OS 是放在硬盘里作为测试治具来被使用的，这样治具的损坏不仅花费公司时间而且浪费公司的成本。

本文设计实现在 UEFI BIOS Shell 环境下进行的除错方式不仅可以内嵌集成 BIOS，而且也可以调用外存程序，除错用户在计算机开机后进入 Shell 模式后可以直接操作和访问计算机系统设备信息，这种方法方便了硬件维修人员的除错工作，同时也摆脱了除错工具对操作系统的依赖性。本文还实现了输入输出界面的可视化，为计算机除错用户提供了人机互动的接口。这样的实现节省了测试治具的损坏，同时加快了测试时间，提高了设计人员和工厂测试人员的工作效率。

1.2 硬件测试除错现状

目前，在计算机生产领域，研发人员和工程测试人员，对计算机底层硬件设备和核心寄存器的访问实现方式，主要可以通过以下两种途径：

1．依赖操作系统

这样的访问方式除错工具的运行需要借助操作系统环境，需在 DOS 或者 Windows 环境下工作，它具有完全依赖操作系统性，当操系统出现故障或系统用户没有足够权限，工具就不能访问到相关的计算机设备信息。

2．依赖 CMOS 设置菜单

这样的访问方式是不需要执行除错工具的，用户可以直接通过 CMOS 菜单查询计算机的设备和用户信息，它的信息或功能是 BIOS 开发商已经固化好的，所以系统访问功能不能随意被普通用户增加。

BIOS 充当着计算机硬体与操作系统之间的一个接口。它提供了访问系统硬体的先决条件并使其能够创建高层操作系统，完成你所要实现的操作请求。对于控制计算机的硬体设定、电源开关或者重新启动等等的系统功能，BIOS 都是值得依赖的。如下图 1-1 表示：

层号#	层号说明
0	硬件
1	系统 BIOS
2	操作系统
3	应用程序

图 1-1 软硬件与固件 BIOS 的关系

操作系统和应用程序是放在硬盘里进行除错的，所以应对每天成百上千的读取和插拔，造成治具硬盘接口不断的受损，而且在除错过程中也易断掉非法关机，导致硬盘 0 磁道和其他坏道后硬盘无法使用而报废。

1.3 论文的主要工作

1.3.1 研究目标

实现工具在 UEFI BIOS Shell 环境下除错的方式，此工具可以内嵌到 BIOS Shell 内部或调用存放在外部存储器里，直接访问计算机设备信息和工作状态，使用户不依赖操作系统情况下完成除错任务。

1.3.2 研究内容

除错工具在 BIOS Shell 环境中实现的过程中，实现了 BIOS 的固件内嵌功能，如何访问计算机内存和 CMOS 内存信息、PCI 和 SMBus 上的硬件设备信息和设备使用状态等，对信息进行解码后传送给通过用户操作界面，完成在 Shell 下的计算机硬件除错方式。

以下几个方面概述了论文除错方式的主要研究内容：

1. 在研究 BIOS ROM 文件格式和加载文件的技术要求上，调试 BIOS Shell 模式的除错工具，并最终实现将硬件除错工具内嵌 BIOS ROM 存储芯片中。研究除错工具开发和调试并将程序编译汇整成 BIOS ROM 能认识的文件格式，然后使用 EDK 工具环境进行模拟测试，同时完成了在 UEFI BIOS 启动过程中进入 Shell 模式进行硬件除错的实现是作者负责的主要工作内容。

2. 通过对CPU寄存器的访问改写，BIOS环境下实现了访问4G 内存地址空间，利用切换保护模式与实模式状态来突破实现在 DOS 环境下不能最大访问到计算机4G 的系统内存。设计并调试 CPU 的大模式与实模式间的切换实现访问 4G 内存是作者负责的具体工作。

3. 研究了访问计算机内存信息、PCI 和 SMBus 总线上的设备信息、CPU 的配置寄存器、CMOS RAM 信息和对计算机底层设备的程序设计作出流程图。对 CPU、PCI 配置寄存器读写、PCI 和 SMBus 总线上的计算机设备信息和运行状态读写和计算机系统内存和 CMOS 内存的读写访问控制是作者负责的主要工作。

4. 在论文中根据软件设计规范，完成软件设计和调试并给出 Shell 下除错工具方式的对比。对主功能模块、子功能模块、调用模块和用户界面模块作出了具体的软件测试及实现方法。通过模块间的调用完成整个计算机设备的信息和使用状态访问，最后对除错工具在各种环境下的对比和模块演示工作是作者负责的主要任务。

1.4 论文章节安排

第一章前言，介绍了个人计算机行业的不断发展及对计算机的需求，构成计算机制造业迅速发展，造成行业间的竞争非常激烈，所以作者看到了如何节约人工成本，看到了在研发和制造过程中的测试环节节省时间，在不借助 OS 环境下进行除错，阐述了在 BIOS Shell 下也有内部和外部命令，可以对计算机硬件进行测试除错的方法。同时安排了论文研究的顺序和内容，为后面章节的研究内容做了简单的总结。

第二章概述了传统 BIOS 的发展历程、UEFI BIOS 的发展趋势、UEFI BIOS 的固件架构及 PCI 设备的软件访问方式。

第三章介绍了在 UEFI BIOS Shell 中进行硬件测试除错工具的需求分析研究研究，提出了除错工具内嵌和存放外部设备的两种除错需求并行对整个计算机硬件除错需求。

第四章介绍了在 UEFI BIOS Shell 中进行硬件测试除错工具的研究设计方案轮廓，对实现 UEFI 各个模块的工作原理和功能进行了简要的描述。

第五章介绍了在 BIOS Shell 环境下用户交互界面和主调用模块的实现，具体描述了系统实现的框架流程，使用户模块的实现更简单化。

第六章介绍了在 BIOS Shell 各功能模块通过程序软件实现的设计实现。主要包括各模块的软件控制原理、模块功能要求以及用户界面接口设计等，并完成功能模

块设计框架中子模块的调用实现。

第七章详细介绍了除错工具各功能模块实现后，如何在 UEFI BIOS 中完成内嵌的应用和使用方法。

第八章详细介绍了除错工具在 UEFI Shell 中除错的意义和调试的方法，同时给出了除错工具的设计结果和其他环境除错做了分析对比。

论文最后第九章针对除错工具在 UEFI Shell 环境下执行的优势和系统技术实现做了结论，对论文计算机生产商除错功能在 UEFI Shell 中除错方式的实现和发展趋势作出了肯定，证明了 Shell 下除错方式是迫于企业的生产改革，节省人力和物料等费用，提高在各 ODM 中的竞争力。

第二章　BIOS 的发展历程和技术特点

本章对BIOS在国内外的发展历程进行了简要叙述，它从传统的BIOS发展到目前UEFI BIOS的发展趋势，同时比较出了传统BIOS和UEFI BIOS的优缺点，接着对UEFI技术特点进行了整体框架的深入分析，对UEFI各个启动模块作了较为透彻的描述。在此分析中，指出了本论文研究内容在UEFI架构中的位置，对计算机生产商的重要性，从而确定了论文选题的可行性及在计算机生产设计或测试维修中的研究意义。

2.1 BIOS 的简介

2.1.1 BIOS 的基本概念

通常讲的计算机的输入（Input）和输出（Output）设备包含了连接在计算机上的所有设备，顾名思义 BIOS 的功能非常强大，被集成在主板的存储芯片中，已经从开始的只读内存芯片（ROM）发展到可擦写内存芯片（EEPROM），它里面存放着程序代码，功能管理着计算机的基本输入输出系统（Basic Input Output System），可想而知 BIOS 是整块主板的核心部分，它管理着主板上所有的设备。BIOS 芯片现在一般都是可供 ODM（生产商）或 End User（终端用户）进行升级刷新的可擦除存储芯片。它里面的程序代码除了存放着计算机基本输入输出程序外，还包括 POST（Power On Self Test）程序和计算机的系统信息等，业界综合 BIOS 的以上介于软件（Software）和硬件（Hardware）之间，所以在学习讨论中也称之为固件（Firmware）。

2.1.2 BIOS 的基本功能

计算机主板性能的强弱主要由 BIOS 的管理功能的强弱来决定，BIOS 的管理功能主要包括：

1. POST 开机自检

当我们按下电源开关时，电源就开始向主板和其它设备供电，此时电压还是不稳定，主板控制芯片组会向 CPU 发出一个 RESET 信号，电源开始稳定供电后，芯片组便撤去 RESET 信号，CPU 马上就从地址 FFFF0H 处开始执行指令，CPU 取的

第一条指令（一条跳转指令），跳到系统 BIOS 中真正的启动代码处(系统板上 8KBROMBIOS 程序)，运行 POST 程式（Power On Self Test 程式）开始自检。通常自检主要包括对 640KB 基本内存、ROM 主板、显卡系统和键盘控制器等。如果在自检过程中发现问题，系统程序会暂停运行且在屏幕上给出提示信息，无法在屏幕上显示出来可通过喇叭发声来报告错误情况，声音长短和次数代表了错误的类型。

2. CMOS 系统设置程序

CMOS RAM 存储芯片中记录着计算机的硬件信息和用户的设置信息。硬件信息是 BIOS 程序开发商设定的；用户信息计算机用户可以根据自己的喜好进行设定，这些信息资料全部存放在计算机主板上的一颗 CMOS RAM 芯片中的，用户可以通过热键进行读取或修改。POST 自检时会对计算机设备复位然后从 RAM 中读取硬件配置信息进行状态设置，有时设置不正确会导致计算机系统瘫痪而无法使用；另外 BIOS 在把主控权交给操作系统前，它会读取访问 CMOS RAM 中的用户信息，根据用户的设定进行读取访问，最常见的就是启动项，用户可以实际需要设定光驱还是硬盘为操作系统第一启动项。

3. BIOS 操作系统自检程序

计算机 POST 自检完成后，会根据 CMOS RAM 中的信息选择操作系统启动项，按照保存的启动顺序进行搜寻，当第一启动项失效无法启动时系统跳转到第二启动项，以此类推。顺利找到操作系统的话 BIOS 就会把系统控制权交给引导记录；如果没有发现启动操作系统，系统就会在屏幕上提示没有启动设备或者缺少相应的文件。

4. BIOS 中断服务程序

用程序软件功能与计算机硬件之间实施衔接其实就是所说的 BIOS 中断服务程序，它是一个位于计算机系统中硬件和软件的可供用户调用执行的可编程接口。如：硬盘和键盘等其它外设管理中断，它们都可以在 Windows 或 DOS 操作系统中实现调用或访问，而且 BIOS 中断程序也可以通过 Int13、Int5 等中断来实现调用服务。

为了让操作系统和应用软件能够在计算机上顺利运行，BIOS 提供了一个 OS 可以与硬体对话的服务标准，然后 OS 再将这些服务标准提供给应用软件，并执行其所具有的其他功能。这一服务标准可以从图 2-1 所表示的视图中看到各个层面之间的关系。

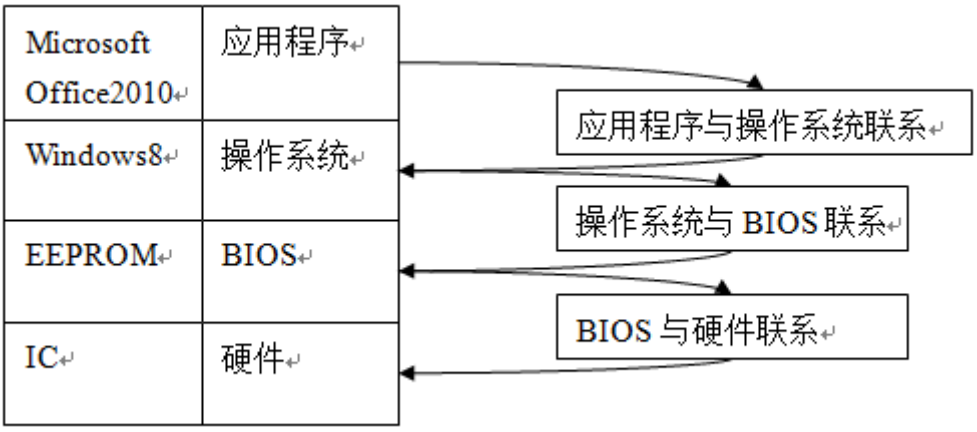


图 2-1 操作系统与应用软件运行图

2.2 UEFI 国内外发展趋势和概况

2.2.1 UEFI BIOS 的发展状态

BIOS 在计算机不断的更新换代中除了不断扩充自己的装备已适应计算机系统的需求，而且也在从传统的 BIOS 到目前的 UEFI BIOS 变革中。当然传统的 BIOS 为计算机的发展作出了举足轻重的作用，承担着计算机内部硬件设备和用户外部设备的管理，在与计算操作系统间发挥着承上启下的链接作用。BIOS 的程序内容被烧录在一颗内存芯片后，它就集成了软件和硬件的功能。在计算机 POST 自检时，除了初始化计算机内部系统，还要对用户使用的外部设备进行初始化管理。传统的 BIOS 是由 16 位代码的汇编语言编写的，它对计算机系统内存的 1M 的地址空间内划分了好几个模块进行管理，包括预先调用和存放一些设备和用户处理的信息，以加快计算机的执行速度，数据量太大是通过定义高位内存来进行存放调用。长期以来一直保留着各种寄存器参数的调用方法和 16 位的实模式。即使 CPU 经过很多次的更新换代，访问模式也从实模式进阶到大实模式，但还是需要 BIOS 系统的管理调用。这种传统的计算机系统调用方式，导致新款 CPU 不管进阶到更高级的双核甚至四核以上都要加入这一调用方式来符合系统兼容，这样大大降低了 CPU 和整个计算系统的性能。传统 BIOS 以上特性导致计算机设计者不仅对 BIOS 有着代码难以记忆、复杂的编写原则、用户操作体验不佳和降低 CPU 和计算机系统性能的抱怨。Intel 酝酿出的一个改革新方案 EFI(可扩展固件接口)计划导致老旧的传统 BIOS 面对一场新的革命，EFI 计划可以用来优化管理日后 IT 业的发展需求。

传统的 BIOS 代码是采用汇编语言编写，面对软硬件快速的发展升级，对于计算机软硬件不断更新的新需求，已经无法满足市场需求，在短暂的 TAT（Time Around Time）交期中显得力不从心。Intel 在 2000 年的时候向业界展示了 BIOS 的新一代接口程序 EFI，并将 EFI 技术运用在当时较新的服务器平台，从而推出了一种能适应未来计算机系统可持续发展的 BIOS 的替代方案。而新的 EFI 管理模式采用了 C 语言参数进行系统数据的传送和访问、构建信息动态链接和系统模块化的布局形式比传统的 BIOS 更容易实现编码调试。另外，EFI 驱动程序为了保证在不同 CPU 架构上的兼容性，EFI 字节编写的源程序可以不用工作在系统 CPU 上的编码组成。在传统 BIOS 中需要程序员的丰富设计经验、除了界定 BIOS 的地址区域没有固定的文件来规定，Intel 根据传统 BIOS 的这个事实标准来制定统一了一个可扩展和可扩展的固件接口规范。

EFI 很像一个被简化的操作系统，从内部功能来看，它的功能是介于计算机操作系统和计算机硬件设备之间实现的。传统 BIOS 的界面是单调的纯文本方式显示，而 EFI 的界面是以彩色图形方式显示，EFI 内部集成了一个高分辨率的图形驱动器，同时计算机用户可以通过计算机输入设备如用鼠标进行点击操作。EFI 与传统 BIOS 的另一个显著差异是用 C 高级语言进行编码，这个不仅摆脱了传统 BIOS 16 位汇编语言的复杂难学，而且 C 高级语言在全球的程序设计领域已经非常广泛，这样就意味着更多的 C 高级语言软件工程师去步入 EFI BIOS 的开发领域，非常有利于 BIOS 操作平台的技术快速发展和实际应用。目前，EFI 平台的应用已经从服务器（Server）技术领域扩展到个人计算机领域，如苹果、惠普、Dell 等世界知名计算机生产商都以逐步开始向新一代 EFI BIOS 发展应用。同时，EFI 的技术平台也慢慢在向个人家庭家用设备和消费电子领域扩张，电子操作用户可以借用 EFI 不依赖操作系统的特性，进行直接信息功能处理，如开机直接上网浏览网页或进入云计算等，这样不仅提供了方便，而且提高了产品的运行速度，满足市场需求。

毫无疑问，EFI 在计算机和消费产品上的发展和应用，它的发展前景让更多哟用户见到了 EFI 技术的方便性，经过微软、惠普、AMD 等世界知名企业的联盟进行开发和应用，同时也把 EFI 更名为 UEFI，制定管理统一可扩展固件接口规范，并把 UEFI 主要源代码共享在了网站上，供 UEFI 爱好者学习和开发。

为了更好的开发和管理 UEFI 规范，这样就形成了开发和管理的组织，同时还负责 UEFI 规范的推广，和推广 UEFI 规范标准，让开发商根据统一的标准开发应用，它就是 UEFI 联盟。这个国际性组织是在 2005 年由 HP（惠普）、Microsoft（微软）、Intel(英特尔)等厂商共同筹建成立的，该组织并自定详细的章程，根据技术发展的需

要制定每年共同研讨 UEFI 技术。为了使 IT 业更快的关注和采用，联盟组织成立了四个工作小组进行不同方式的市场调查，调查内容包括了丰富多样的培训和推动，使 IT 行业很快明白了这个新的技术。目前，UEFI 联盟旗下已经有 86 家世界知名企业成员，参加联盟成员企业秉承着不断为 UEFI 技术的应用、推广和开发贡献做努力，各自承担相应的义务和权责，使 UEFI 技术形成了一种循环的产业链结构如图 2-2，紧密协作，到应用成收集资料进行技术改进后再推广应用。

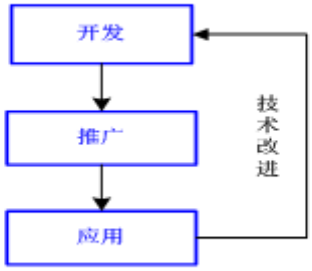


图 2-2 UEFI 企业联盟成员分工流程

经过 UEFI 在企业联盟的共同开发到使用过程中，发布了两个最新的 UEFI 管理规范。其中计算机的操作系统或设备驱动程序与计算机系统固件之间的接口有 UEFI 2.3 规范来定义；PI(平台初始化规范)v1.2 是用来提供固件组件与设备生产商包括设备固件和设备管理信息资料等能相互操作和访问规范。

UEFI 企业联盟技术大会研讨也从先前的美国本部向其他国家举行，中国南京也曾在 2007 年举办过。这样不仅使联盟组织区域扩大化，而且使国内外更多生产或设计厂商了解 UEFI 技术，了解 UEFI 联盟及他们的职责和任务，为后面推动 UEFI 技术的发展奠定了基石。

由于技术大会的推动，在全世界范围内掀起了一场 UEFI BIOS 逐步取代传统 BIOS 的革命，使得让 UEFI 更通用性，自从 Windows 2008 Server 和 Windows7 的出现，这些过去需要通过 BIOS 来完成硬件控制的已经慢慢由 UEFI 来实现完成，在 Windows8 刚刚露出来，相信在未来开放应用将会更加广泛，大家对基于 UEFI 的各项技术应用来满足社会需求也会更加成熟。

2.2.2 UEFI 的局限性和安全隐患

科技就像一把双刃剑，有好的也有它的局限性，UEFI技术也拥有着两面性特点。

它在为UEFI技术应用领域不仅提供给普通用户操作界面的视觉化功能，而且给UEFI用户提供访问接口的简单易懂、模块化的结构层次更加方便记忆操作，但在提供方便的同时也带来了一些局限性和安全隐患，如以下几点：

1．黑客侵入风险，由于UEFI可以不借用操作系统方式直接进行实现用户访问，TCP/IP、UDP等协议和数据链路成可以完全在DXE阶段执行生效，用户可以在没有杀毒软件或防火墙的情况下直接访问Internet功能，方便用户操作使用的同时给黑客侵入计算机系统创造了更加方便的途径；

2．接口保留（Reserved）风险，UEFI规范定义了很多扩展性接口，其中保留的可供用户开放使用，但同时使黑客也可以轻而易举的瞄准这些接口，不需要通过大量的反编译等流程，直接可以查看源代码核心进行乘虚而入，使得系统侵入风险性比传统的BIOS更高；

3．加载模块繁琐，传统的BIOS模块简单，作为新技术的UEFI内部被划分了四个主要部分，每个部分都有自己单独的核心内容控制，即使功能重复相互也不会共享资源，比如DXE和PEI模块里都有分发器负责，但都是用自己单独的分发器来操作。对于用户使用的个人计算机来说，它的操作要求非常简单，分的太细反而市操作平台复杂化，对于大型的计算机而言，这么多的模块自己执行完然后再交个下一个模块操作，这样每个阶段的操作验证使系统需要很多等待的时间，必然导致计算机开机速度没有传统的BIOS快。

4．开发人员专业素质降低， C 高级语言不仅比汇编语言易懂好记，而且在国内外的发展已经非常成熟，这样开发 UEFI 不需要传统 BIOS 那么专业素质高，开发者不用再需要记忆很多的汇编等代码，同时 UEFI 也提供了一个调试端口供用户除错或程序调用，这种直接在 DXE 阶段开放 UEFI Shell 端口给 UEFI 系统增加了 BIOS SMM(System Mangement Mode)的攻击的可能性[20]，严重时时造成系统瘫痪或破损坏导致无法正常使用。

2.2.3 UEFI BIOS 与传统 BIOS 的区别

Basic Input Output System 最初产生于 20 世纪 80 年代，应用于 IBM 的 PC/XT 以及 PC/AT 系统。BIOS 被固化于主板上的 ROM 中，如 EEPROM 等。计算机加电启动后首先运行 BIOS 代码，由 BIOS 代码接管并负责计算机硬件的识别，检测以及初始化工作。BIOS 抽象出了不同计算机平台的硬件组织，使得应用程序能够运行于基于 X86 架构的多种 PC 之上。

由于硬件设备种类和功能的发展，现代计算机系统中通常将 BIOS 代码分离在

主板的 ROM 以及若干存在于硬件适配器的 Opt ion Rom 中，主板 BIOS 包含基本的硬件识别，检测代码，并负责加载 Option Rom 中的特定硬件的驱动代码，这些 Option Rom 中的代码可以不用电脑生产商而由硬件生产厂商自行开发。

由于传统 BIOS 采用了软件抽象的方法向应用程序提供了统一的访问系统资源的接口，所以被计算机生产商广泛的采纳引用。但是随着硬件的发展，即便是使用 Option Rom 的方法非常方便，但传统 BIOS 也开始面临一系列新的问题：

第一，传统 BIOS 定义了一组与 OS 无关的接口，但它却不是平台独立的，它依赖于中断来组织 BIOS 的软件结构，也因此依赖于 X86 的中断模型。也就是说，它的软件接口并不是完全硬件独立的。

第二，传统 BIOS 运行于实模式(Real Mode)下，寻址能力为 1M。虽然理论上和技术上可以在实模式下拥有更大的寻址空间，但这些技术往往采用特殊技巧，使得系统的耦合度大幅增加。可选 ROM 的地址空间也局限于 1MB，大大的限制了 Option ROM 中驱动程序的扩展能力。1985 年 Intel 发布了 32 位 CPU，而目前的 CPU 总线已到达 64 位，传统 BIOS 的寻址能力对此造成了严重的浪费。

第三，传统 BIOS 采用汇编语言，软件工程实施与管理的效率较低。工程师往往采用大量的汇编技巧，扩展性和可维护性较差。然而硬件的发展要求 BIOS 具有较高的扩展性和可维护性。

在此发展背景下，Intel 开发和规范了 EFI（Extensible Firmware Interface）BIOS 系统架构的统一，这样传统得 BIOS 功能就被可扩展固件接口协议来替代，EFI 已成为工业联盟标准来开发使用，它定义了平台固件与计算机操作系统或用户设备之间的相互访问规范，这个规范制定的操作平台信息，用户可以根据实际要求启动实时服务，是固件和操作系统新的接口方式。它健全的平台信息和请求服务使得用户程序在 UEFI 系统启动时完全载入，它为预先载入用户程序和操作系统的执行提供了全新的思路模型。UEFI 与传统 BIOS 的优点是提供用户设备可扩展的环境，清晰的模块化结构和提供了统一的操作规范。UEFI BIOS 采用了 C 语言编写，能够在启动时实现丰富的功能扩展[19]，它相当于一个简单的操作系统，在开机程序进入 POST 未把系统主控权交给操作系统前，用户可以选择 UEFI BIOS 内嵌的 Shell 环境进行系统操作访问，它可以在 DXE 阶段加入了用户程序一起载入到 Shell 环境进行调用操作，它实现的功能可以是 Internet 的访问，也可以在此环境下进行设备信息访问、软件工具的调试和硬件设备的除错。虽然目前计算机生产商正在逐步使用新的 UEFI BIOS 开发计算机系统，但 UEFI BIOS 离正式完全取代传统的 BIOS 仍需要一段时间去磨合和改善，基于 DXE 阶段 UEFI Shell 开发的应用程序仍然局限于小

规模的群体中，如何推广这种发展前景巨大的底层应用开发环境，使生产商节省开发和测试时间，其意义是显而易见的。

2.3 UEFI 的架构分析和设备访问

2.3.1 UEFI 架构平台

UEFI是操作系统与平台固件之间的平台信息的数据表和启动时以及启动后的服务规范定义的软件接口。启动服务的功能是调用图像控制器，系统上的所有设备的初始化和系统端口访问操作平台的建立。

UEFI内核还内嵌了一个操作窗口，类似Windows操作系统中内嵌了一个Command(命令)窗口一样，可供用户在窗口中直接执行程序，这个内嵌的窗口就是本论文研究的重点，叫Shell窗口。它可以不借助操作系统直接进入Shell模式进行运行用户指定的程序。EFI管理器也可以用在装载操作系统，在这个环境下安装系统就不需要传统的启动盘安装，免除了用户安装系统时碰到的光驱无法启动或没有光驱等困扰。实现UEFI固件接口统一的方法，它完全可以替代传统的BIOS，而且功能比传统BIOS更加强大，它是用C高级语言模块化和层次化编写而成的，使UEFI的内容更加容易分析理解，固件架构非常Framework，UEFI整体框架中提供了本文所研究的UEFI Shell模块，UEFI整体结构图如图2-2所示：

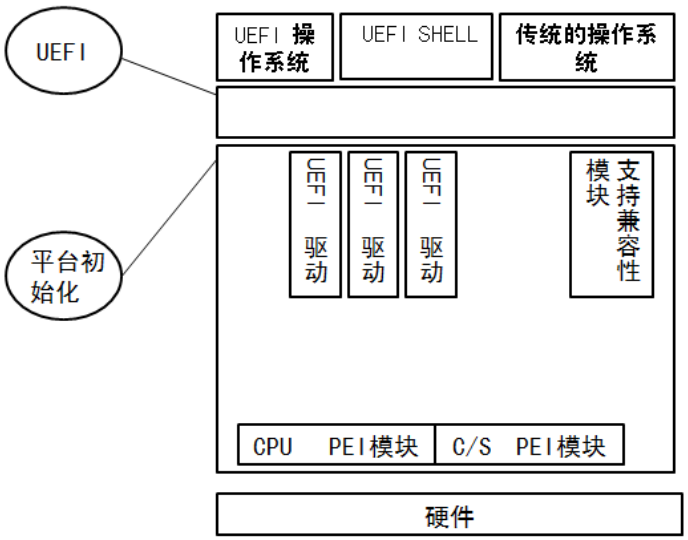


图 2-2 UEFI 的框架结构图

在UEFI框架下，计算机设备硬件是通过调用PAL和SAL来控制，它本身不能直接对硬件进行操作。UEFI的运行时服务(Runtime Service) 和引导服务(BootService)都通过调用PAL和SAL的相应功能[20]。UEFI自带了一些可以供API直接调试调用的系统工具软件，在DOS中我们称为内部命令，在UEFI中可以直接调用，不用定义盘符路径，同时UEFI中也有称为DOS中的外部命令，也就把开发好的UEFI程序不内嵌BIOS中，存放在外存设备中供在Shell模式中调用。总的来说，UEFI在平台固件模型的框架里提供各模块间的互相访问，隔离或关闭系统硬件物层。

其他模块间在UEFI结构中的关系如图2-3所示，主系统在经历了PAL和SAL模块后，会运行一些必须的服务程序，以便能将EFI BIOS完成的系统参数传递给OS，这里有运行时服务和引导服务。同时也会有一些EFI API和OS loader，在第四章的时候会介绍如何在Shell底下分析这些EFI API。

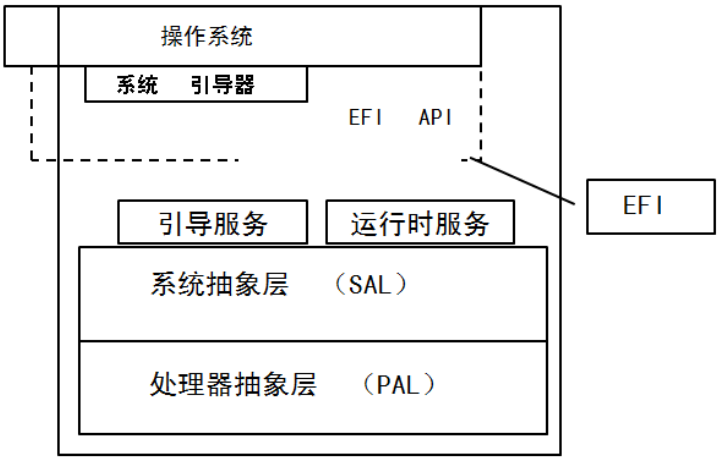


图 2-3 UEFI 与其他模块的关系

2.3.2 PCI 设备的软件访问方式

软件的配置周期是有 PCI 总线规范来定义，用户可以通过两个运行机制来实现 PCI 设备的软件访问。不管 PCI 设备在计算机的发展中有多快，目前主板采用方式都是配置机制 1 方式访问，另外一个配置机制 2 的目的是可以向下兼容访问 PCI 设备。在配置机制 1 中，它又占用了两个单元的双字 IO 端口，这两个端口有共同的特点是：它们都是寄存器并且这个寄存器都是可以读写的。第一个端口的地址是从 0CF8H 开始的，第二端口的地址是从 0CFCH 开始的[2]，它们分别叫做配置地址寄存器和配置数据寄存器，地址寄存器里存放着总线编号、读写寄存器编号和 PCI 设

备编号，这样就可以对数据寄存器进行读写操作了。这样一来访问配置周期也符合了 PCI 总线的要求，程序先将需访问的设备编号地址作为目标地址写入配置地址寄存器，寄存器记入设备编号后同时转化成 PCI 设备号，然后对数据寄存器进行数据读写，完成对 PCI 设备的检测，方便后面测试除错工具在 Shell 环境下读取 PCI 设备的工作信息，对 PCI 设备进行信息交换的来验证设备在 PCI 总线上是否堪用。

通过对 PCI 设备的软件访问方式，可以在 UEFI Shell 中顺利获得设备信息或设备的使用功能情况等，当然计算机架构中多个设备都是可以挂在 PCI 总线上的，所以我们是根据不同的地址总线去寻址那个设备在使用，但前提是设备使用电源没有短路，计算机有硬件保护系统，一旦短路将无法使用软件方式访问，只能借用外部设备对计算机设备进行检测，那是因为PCI总线不要一个PCI设备短路造成其他PCI设备全部损坏。在工作电源正常情况下，PCI 设备是可以通过软件的方式告诉具体哪个硬件设备或设备号在 PCI 总线上功能的好坏，这样提供了软件快速除错硬件的可能性。

2.4 UEFI BIOS 过程阶段分析

传统BIOS的开机POST比较繁琐且慢长，UEFI利用软件工程的设计原理，把计算机系统的启动过程作为一个项目来划分成四个区域模块（SEC、PEI、DXE、和BDS），整个平台的初始化有各模块间的衔接来完成，方便了系统对模块的访问和调用，模块及模块间的功能和运行流程如图2-4所示，每个阶段的具体内容是：

第一个阶段是安全阶段(Security，SEC)，它一般在主板中需要硬件支持，负责计算机软件和硬件的安全保护。通电不开机状态下程序就已经先执行的，除了确保UEFI平台的启动固件程序是完整的，而且还监控着系统上的所有设备安全，也就是通电后非法移除任一系统组件或系统设备时都会有报警提示。UEFI体系提供了扩展接口功能，这样就可以支持可能在今后科技发展中被新发现及增加的新的产品功能。

C高级语言被作为UEFI BIOS编写格式的基本开发语言，使得C代码可以在UEFI的系统上或UEFI Shell里实现执行的特别设计任务。这个基本执行原则是第二（PEI）和三（DXE）阶段功能划分的依据，为顺利进入PEI阶段做初始化前的检查。

第二个阶段是PEI初始阶段（Pre-EFI Initialization），起初PEI阶段里的代码大部分是用机器的汇编语言编码的，它主要任务是初始化内部系统和系统设备中的设备寄存器等，初始化内存和利用DMA（Direct Memory Access）实现快速访问内存和数据传送，系统保持初始化状态到DXE执行环境。在PEI阶段里，程序一旦在内存中内

存中发现状态信息资源，它就马上把信息轮廓描述出来，为程序主控权移交到下一个DXE阶段做初始化。PEI里的状态信息被初始化后，里面的运行代码将不能再被调用，因为它到下一个DXE阶段后，程序运行过程不能逆向访问执行，即是一个单向的执行过程，PEI主控权交给DXE阶段之后，PEI阶段里的执行功能消失，DXE操作环境得到了真强，它同时具有了齐全的设备管理功能。所以PEI的主要工作任务是：

1．基本的芯片组（Chipset）初始化
2．内存大小检查
3．BIOS内容备份
4．睡眠（S3，Suspend）复位
5．切换寄存器从堆栈（Stack）到内存（Memory）
6．预备到下一个阶段，启动DXEIPL（DXE Initial Program Loader）

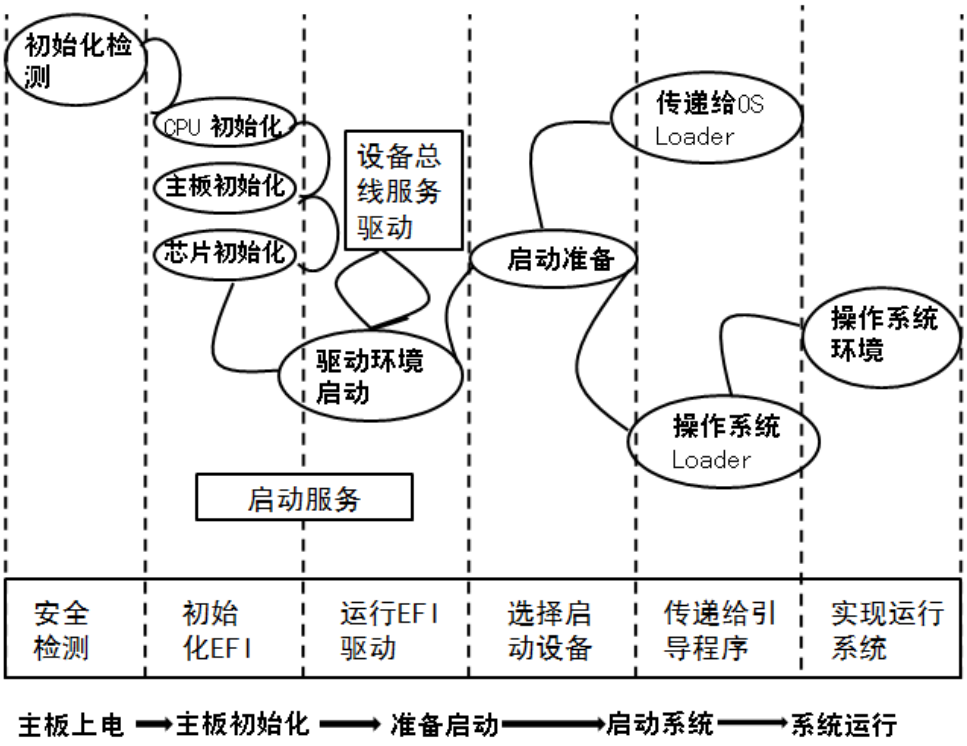


图 2-4 UEFI 与 Tiano 系统框架

第三个阶段是DXE阶段，这个阶段的功能是对系统中的设备驱动进行初始化动作。DXE的过程里面包括几个部分，第一个是DXE的内核，其中包括DXE分派程序，实时运行服务和DXE服务程序，第二个是DXE的硬件驱动程序[26]。DXE分派程序按照预定义的规则负责发现和运行DXE驱动程序，DXE不仅提供对用户运行硬件设备、系统内部调用和设备控制器的初始化，而且也完成计算机系统和硬件，包括CPU处

理器，芯片组和其他设备组件的初始化， DXE阶段主要负责用标准的UEFI驱动来初始化平台，并且提供启动操作系统所需的各种启动或者实时服务。DXE阶段的最后会将控制权交给启动设备选择BDS(Boot Device Selection)阶段，BDS会启动选择程序，启动设备的执行选择顺序是按照用户在CMOS中设定的顺序进行逐一执行寻找可启动的操作系统。

第四个阶段是BDS阶段，这是UEFI 启动的最后一个阶段，UEFI在这个阶段把控制权转交给被启动的操作系统。在BDS阶段，一般的实现方式是向最终用户展示用户界面，在用户界面可以自由修改和选择启动选项，业界OEM和ODM可以修改和定制DXE的启动方式和适应的操作系统[3]。

2.5 本章小结

本章通过对 UEFI 和传统 BIOS 的比较，总结概括了 UEFI BIOS 的可编程性，C 高级语言更能方便程序员解读或编写，同时更进一步对 UEFI 架构的介绍，使 UEFI 轮廓更简单化，在选择启动设备前加载作者需调试的 EFI 驱动等程序，这样方便在进入 Shell 后各个 PCI 设备的正确调用，把 UEFI 解剖成四个主要阶段，为后面在 BIOS Shell 中内嵌硬件除错程序的方式奠定了设计方案的可执行性。

第三章　UEFI BIOS Shell 中硬件除错方式的需求分析

本章根据在计算机 UEFI 和生产系统除错系统现状的背景下，提供了一种基于 UEFI BIOS Shell 中进行硬件访问除错的内部需求，并给出了除错工具在 UEFI Shell 环境下的可行性方案，进一步确定了利用 UEFI BIOS Shell（以下简称 Shell）实现内嵌程序和在 Shell 中执行外存设备上的除错工具的两种需求分析。

3.1 Shell 中除错工具设计使用性能的可行性分析

3.1.1 除错工具的运行时间分析

BIOS Shell 里面内嵌硬件除错工具执行是比较方便，Shell 环境中执行外存设备上的程序同样又方便又节约成本。目前的除错环境都是在 OS 环境下进行除错，用每天生产测试 500 台电脑实验，收集到存放放数据的存储设备比放 OS 的存储设备使用寿命会多 6~9 个月，而因经常拔插硬盘测试治具而损坏的每月平均就有 3000 元人民币左右。

在测试时间上面，目前测试 OS 需加载很多的应用程序，所以开机一般在 2 分钟左右，而 BIOS Shell 15 秒就可以进行执行除错程序，所以每个工位测试一片板子就可以节省 1 分 45 秒时间，拿作者所在的计算机生产厂商测试除错流程举例如图 3-1，里面有 5 次功能测试可以节省 8 分 45 秒，算作人力成本每片板人力成本就可节省 1.89 元人民币，这样一天就可以节省 945 元人民币。

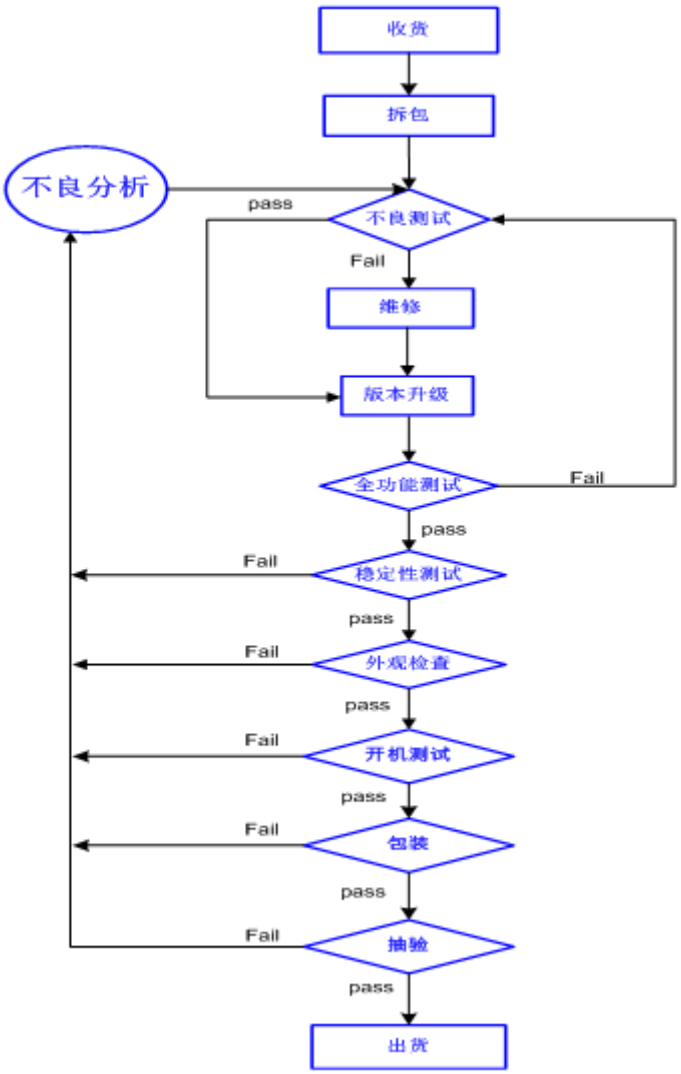


图 3-1 计算机生产商除错测试流程

为了实现这个除错性能需求，除错工具需要在 15 秒内实现对计算机底层硬件设备的访问， 以满足测试除错过程中的需求[6][7]。

3.1.2 除错工具的调用环境分析

除错工具调用的性能需求主要有以下两点：

1．内嵌方式需求

常用的除错工具可以进行内嵌调用执行，方便计算机生产过程中经常调用节省时间，也可以供计算机使用者对硬件的功能或性能进行检测。

2．外存方式调用需求

不常用的除错工具，只是计算机生产过程中进行除错使用的可以存放在计算机的外存设备里，进行对计算机硬件的除错。

3.2 Shell 中除错工具设计功能需求的可行性分析

3.2.1 BIOS 下除错工具的设计分析

事先设计安装硬件除错工具在现有的 BIOS 里面是一种方便快捷，与操作系统不相关的实现方法。它要求设计界面友好，使用可靠，除错工具生成的执行模块数据要小，尽可能减少在 BIOS 存储芯片里占用地址和数据空间；同时除错工具的加载不能影响 BIOS 其它功能段的地址和数据，从而影响除错工具的除错功能。依据这种情况，选择了下面的设计方法：

1. 程序开发使用 C 高级语言

除错程序利用 C 高级语言来编码的优点是压缩产生的文件代码小且执行速度较快，缺点是需要大量的 C 语言代码实现功能。

2. 软件健壮性的研究

访问硬件时，软件要求是在没有任何保护措施环境下执行的，这样加大了对系统硬件的风险。比如，用户要访问一个已经损坏或不稳定的硬件设备，可能会导致系统死机或电脑强制重新启动等故障。这样就需要对软件健壮性保障，在执行过程中发现硬件有问题，用超时保护或用户输入确认来实现对硬件的访问。

3. BIOS 集成方案举例

用主板上的 BIOS ROM 生产厂家为 Award 为例，准备好 BIOS 芯片和符合的 BIOS 烧录器与电脑连接，把 ISA 参数加在 Award 公司已经提供的 CBROM 工具上，然后利用此工具和 ISA ROM 的方式把全部的工作模块打包压缩到 BIOS 芯片中。把压缩好的 BIOS 取下放到需调试的主板上，系统开机后 BIOS 释放压缩的规则也是按照 ISA ROM 的工作方式处理的，然后调用执行 BIOS 代码，不影响 BIOS 其它地址数据的情况下加载除错工具到 BIOS 模块中，保证了计算机硬件除错的有效性。

4.不内嵌除错工具需求

系统针对不内嵌的工具需存放在外部存储设备，和内嵌的唯一区别是启动后到 Shell 模式后需手动运行，但依然不依赖 OS 进行除错。

3.2.2 除错工具的软件编程要求

计算机硬件的除错工具是预先写在系统 BIOS 固件中的，存储芯片大小限制对硬件除错工具的长度和大小也有所限制，所有最后编码生成的软件代码长度需尽量的小巧，但是必须在能保证硬件除错工具能准确的除错下，对于已经存在的硬件中断功能，设计时尽量多调用来减少 BIOS 的大小；本硬件除错工具主要采用 C 语言编写，非常便于移植，同时为了节省代码空间和提高运行效率，小部分核心代码和反复使用的函数采用汇编语言编写，基本上都能符合以上 BIOS 的开发要求。

3.2.3 除错工具的格式处理

计算机除错工具在编码定义时必须遵守 Option ROM 的编码规范，这个规范定义了所有加载过程中 Option ROM 对所有程序代码的规定和调用，在 BIOS 中实现内嵌时硬件除错工具开发也一定也严格按照 Option ROM 基本规范，保证除错工具的正常调用。

1. 处理 ROM 格式头

模拟 Option ROM 的格式头处理可以在 Option ROM 的格式头里添加，设计和编写硬件除错工具文件后进行调试，然后取消硬件除错工具中的操作系统格式头。

2. ROM 长度处理

计算机硬件除错工具的格式头长度大小要遵循 512 字节的整数倍，因为 Option ROM 的大小单都是以 512 个字节。硬件除错工具程序设计编码好后需要自己检查格式头的长度大小。其方法是满足对除错工具的长度要求，即为 512 字节的整数倍，并且如果除错工具的格式头长度大小不满这一大小的要求，设计者可以全部用零填写来满足以 512 字节大小为一个长度单位的特性[5]。

3. 计算校验和

计算校验和的方式是累加除错工具的长度大小，包含除错工具的格式头和长度大小，查看校验位个位和十位上的数值，校验结果全为 0 的话设计者不用在除错工具后填写计算机的反码，如果不为 0 要在除错工具上追加此部分代码，利用这种增加反码的方式可以保证除错工具的校验累加值为 0。

4. 软件开发环境

软件开发环境需要以下工具：MASM 6.11 工具包、Intel EDK2010 工具包、Intel UEFI Shell1.0.shell2.0、微软 VS2008 带 64 位编译器。

3.2.4 除错工具的系统框图

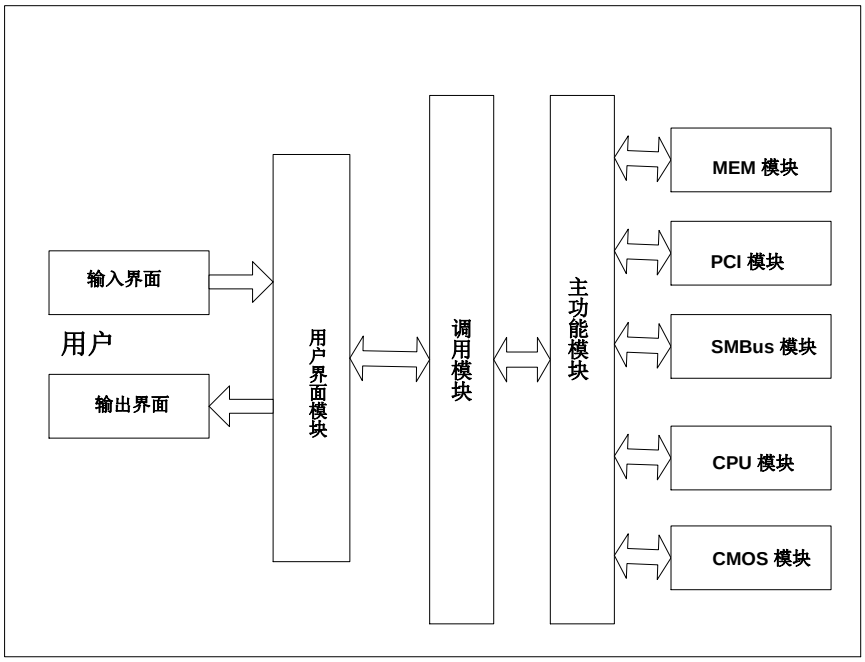


图 3-2 Shell 中硬件除错工具的系统框图

如图 3-2 所示，用户输入信息后可以通过用户界面模块处理后到调用模块，调用模块执行主功能模块，然后主调用模块调用相应的子功能模块，子功能模块处理后再把测试除错结果原路返回输出给用户。这样用户界面输入信息后传递给可以得到除错的信息结果回馈，主功能模块收到用户信息后调用相对应的子功能模块，如内存模块，实现对计算机系统内存的读写测试；PCI 模块，实现对 PCI 总线上的设备功能测试；SMBus 模块，实现对系统管理总线上的设备功能测试；CPU 模块，实现对 CPU 的运算控制等功能的测试；CMOS 模块，实现对计算机和用户信息存储功能的测试，包括 RTC 功能测试。各个模块间都是收到信息后处理，处理后回馈给调用模块，最终把信息情况传递到用户界面，完成计算机设备的除错测试。

通过除错工具的调用方式得出几个重点:第一,除错工具是预先装在 UEFI BIOS 里的，使得运行环境脱离里固件和操作系统的联系，访问底层设备资源的方式也不依赖操作系统，在系统开机后可以直接进入 BIOS Shell 环境实现除错方式。第二，计算机系统协议的多样性，对于计算机底层硬件的访问方法也不同，通过不同类型的指令来实现底层硬件的读取或改写。本文章节中实现测试除错工具预装在 UEFI BIOS 内，开机后通过进入 BIOS Shell 模式进行对计算机系统或设备进行访问读取，并通过可视化的交互式界面告诉除错用户系统信息资料、硬件设备的使用情况和工

作状态等。

Shell 中硬件除错工具的系统框架图，整个框架系统分别由除错工具内嵌 UEFI BIOS 固件中和 Shell 环境下进行计算机硬件除错工具，这两个大的功能模块实现组成对实现测试工具的除错功能。除错工具两种不同实现方式，第一内嵌 BIOS 固件中，第二作为 Shell 环境下工具。内嵌 BIOS 的好处是不需要额外的启动设备，启动速度快，使用方便，同时也需增加 BIOS 大小的缺点；作为 Shell 环境工具好处是可以增加非常多的用户界面信息，大小没限制，缺点是启动速度稍慢一些，同时需要外部 Shell 的支持环境。

3.3 本章小结

通过对用户需求和技术需求的分析，对除错工具在 UEFI BIOS Shell 环境下的整体设计原理提供了更详细的介绍，特别是常用的除错工具进行内嵌，不常用的工具存放在外部存储设备，这样不仅节省计算机生产过程中的时间，以达到公司除错要求，也方便了计算机购买者对计算机主要设备的功能或性能检测。

第四章　UEFI BIOS Shell 中硬件除错方式的总体方案设计

4.1 Shell 中硬件除错方式的系统工作原理

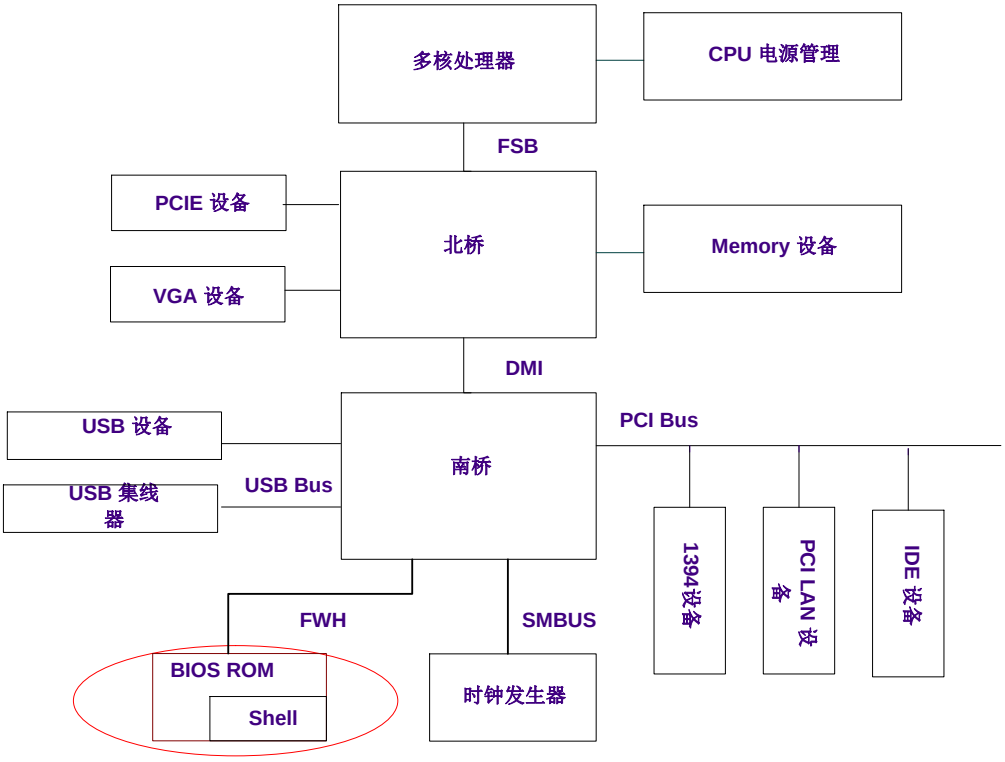


图 4-1 Shell 中硬件除错方式的系统工作框图

通过上图 4-1 可以看出，BIOS 已经自带了 Shell 环境，除错工具可以通过 BIOS 自检时进入该运行环境，调用事先设计好的内嵌或外部调用除错工具。计算机各种外围设备都是通过其所在的 PCI（Peripheral Component Interconnect）控制器与北桥相连的设备要求运行速度，如 New Card、Wireless 等设备；与南桥相连的设备运行速度要求较低，如 1394、RJ45 设备等。所以，访问或读取 PCI 总线上的控制器就可以得到这是系统中的哪个 PCI 设备，包括得到它的系统资源等信息，以及运行状态，甚至可以左右其工作状态和模式。在系统架构中连接在北桥上的硬件设备还有内存设备和显示设备[18]。对于那些运行速度超高的可以连接在系统前端总线

---Front Side Bus，FSB，CPU 和现在一些高档的显示设备都会连接在这个总线上，以改善或提高计算机系统的处理和显示效果；南桥上则连接着运行速度要求不高的设备，本文研究的 BIOS 就是连接在南桥下，CPU 通过访问它所在的 PCI 控制器来访问此类设备，例如 USB 设备、声卡、硬盘、1394 设备、网卡等。

以计算机系统工作原理的框架图作为基础，总结后得出预安装在 UEFI BIOS Shell 内的硬件除错工具的实现可分为两大步骤。第一步是在 EDK 环境下调试好除错工具性能，实现访问底层硬件的设备信息或功能状态，包括访问 IO 资源---计算机系统内存和 CMOS 内存相关数据信息、PCI 寄存器---与 PCI 总线相关的、MSR 核心寄存器---与 CPU 相关的；第二步把除错工具内嵌在固件内，实现不依赖操作系统情况，直接在 UEFI BIOS Shell 环境下执行除错工具。

预安装在 Shell 下的程序可以选择比较常用的，如检测 HDD、RAM、CPU 等，不光可以供笔记本生产商设计或者测试用，而且可以给终端用户做简单的电脑设备性能检测，其他设备功能，例如检测 Audio、Camera、Keyboard 等比较专业的测试程序可以放在周边外接的存储设备里面，可以在 Shell 环境下指定其它盘符直接调用执行，这样 Shell 下进行除错功能不内嵌也可以执行，方便计算机生产商调用，也解决了后续研究中碰到的 BIOS 大小限制问题。

4.2 Shell 中硬件测试除错工具的系统流程设计

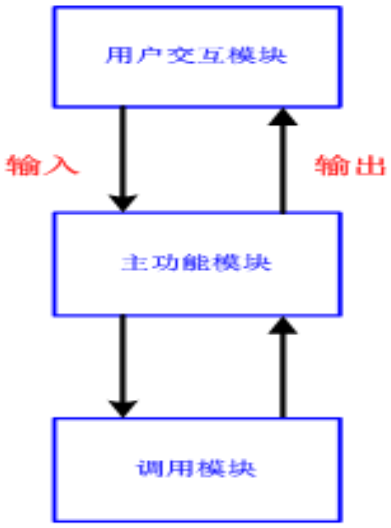


图 4-2 Shell 中硬件除错工具的系统流程设计框图

硬件除错工具的实现，共由三大功能模块组成：第一，主功能模块（Main Function Module）第二，调用模块（Handle Module）第三，用户界面模块（GUI）。

主功能模块，主要任务是负责把用户需请求信息进行译码后把传递给调用模块，然后调用模块把指令交给相应的个子功能模块处理，比如 CMOS 访问模块、PCI 底层访问模块、系统内存访问模块等。调用模块把用户操作请求分配给这些子功能模后再等待相应子功能模块的回馈，然后将得到的结果传输到用户界面模块，使用户得到计算机硬件信息等资料，达到人机互动方式。

调用模块起着桥梁连接的作用，它的任务是控制主功能模块与用户界面模块间通过通用接口模块进行信息的互相传递。当它收到用户传递过来的数据后，按照接口的定义原则进行解码后传输给主功能模块，最后将主功能模块得到的数据信息转送给用户界面。

用户界面模块的功能是显示用户界面，它负责自动扫描这个界面用户输入的请求信息，接收并传送给主功能模块，然后把主功能模块反馈的信息提示给使用者，再等待扫描用户的再次请求。

固件内嵌 UEFI BIOS Shell 内的实现就是，把硬件除错工具内嵌在 UEFI BIOS 中，用 ISA 模式把代码压缩到可选择性 ROM（Option ROM）中，将 ROM 芯片装入计算机系统主板上，然后上电开机自我检测过程中进入 Shell 模式进行执行，并且在除错工具执行结束后再次返回到 BIOS 的 Shell 模式。

4.3 BIOS 下除错工具的加载过程流程

测试除错工具在计算机的开机过程中的所有加载过程如图 4-3 所示。

通过各计算机模块在 Shell 中的调用和执行，最终完成除错工具对计算机底层设备的读取和回馈。这一小节重点描述了硬件除错工具在 BIOS 加电自检过程中的调用流程。

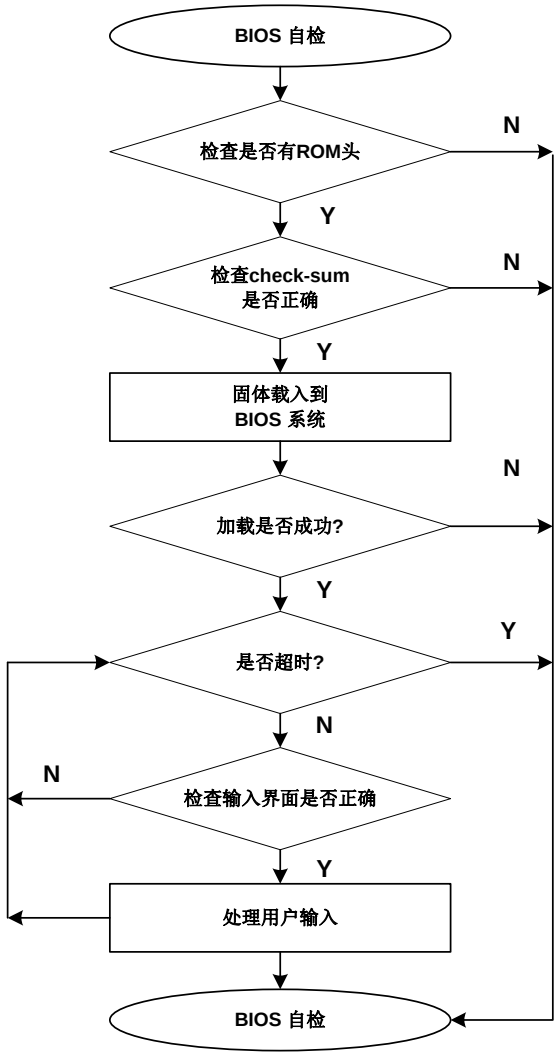


图 4-3 除错工具在 BIOS 中的加载流程图

1. Option ROM 表头的扫描

如果除错工具存在，BIOS 会根据表头标志扫描到标志是 0AA55H 的这个系统，因为 BIOS 在上电开机自检的过程当中会进行测试除错工具表头的扫描工作。

2. BIOS Checksum 的检查

当 BIOS 检测到硬件除错工具在指定的 ROM 位置时，会按照数据结构的定义，马上调用和读取除错工具的头格式，整个除错工具的文件长度也会被 BIOS 会自动计算出，然后数据从 0H 顺序开始硬件除错工具的偏移量处读取获得，累加单位大小为字节，最后计算出一个累加值。这个累加值除了可以用软件实现，也可以通过对 BIOS 烧录器来实现，如果 Checksum 的结果不是 0 的话，就提示设计者这个除错工具的源代码出现损坏或损坏，用 BIOS 机器累加的话还有一种可能是存储芯片硬

件损坏，BIOS 就会认为这个 Option ROM 头格式是无效的，计算机扫描程序会自动退出。

3. BIOS 执行除错工具程序

当 BIOS 读到可用的准备好的除错工具之后，其会自动调用除错工具处在偏移量 3H 处的初始化向量，后将控制权交给测试除错工具。

4. 加载除错工具的方法

计算机完成正常的初始化确保系统和设备都正常后，BIOS 在进入操作系统前把控制权交给硬件除错工具执行，除错工具中的代码被逐条运行，运行前先要确认有没有足够的系统内存可以分配给存放除错工具代码，如果系统资源不足就把主控权还给 BIOS 继续下一步动作，如果系统资源可以满足除错工具的大小，那就开始执行除错工具代码。

5. 执行除错工具的流程

除错工具里内嵌了一个定时器功能，在工具初始化后首先启动该功能，如果在规定的时间范围内用户没有输入信息或超时那程序就会在用户模块中提醒。当用户正常输入请求信息后，除错工具就会扫描到用户资料，进行译码后调用相应的模块以获得正确的请求回馈，然后退出除错工具，除错工具的具体执行流程图如图 4-4。

图 4-4 除错工具的执行流程图

4.4 本章小结

本章对 Shell 中硬件除错的方法进行了设计方案分析，该方案给出了除错工具在 UEFI Shell 下工作的原理，最后在应用上实现了人机互动，同时也点到了除错工具如果很多无法完全内嵌到 BIOS 的情况下，也可以存储到外部存储设备，在 Shell 下实现直接调用，所以除错工具不仅在 Shell 下内嵌而且可以在外存储，这样更加增强了除错工具在 Shell 下执行的可能性。

其中除错工具用户交互、主调用模块和主功能模块实现在第五和六章介绍，第七章介绍了除错工具内嵌的实现。

第五章　用户交互和主调用模块的实现

用户和电脑的交互主要通过用户交互模块来完成，这个模块的设计主要考虑界面友好和使用的便利性，使用者可以通过用户界面来浏览，检查，和更新硬件设备的寄存器，从而实现自己的除错目的。用户交互模块和系统主功能模块的桥梁是主调用模块，这个模块传递参数，协调两个模块的工作。用户交互模块和主调用模块的实现方式将在本章节中详细介绍。

5.1 用户交互模块的逻辑图

UEFI Shell 除错工具的用户交互部分全部由用户交互模块完成，这个模块提供一个用户直观可视化，可交互的操作界面，当除错工具成功加载后，会有一个循环查询机制等待用户的输入，在用户输入后将执行的结果反馈给用户。

用户交互模块有两个主要的接口，第一个接口是跟用户相关性的，输入输出功能，即提供给用户输入的界面和程序输出的显示结果画面。第二个接口是针对主调用的，把用户输入的数据按照程序规定的格式传递给主调用模块，在主调用模块完成数据处理后将结果返回给用户交互模块。

用户交互模块的实现如下图 5-1 所示：

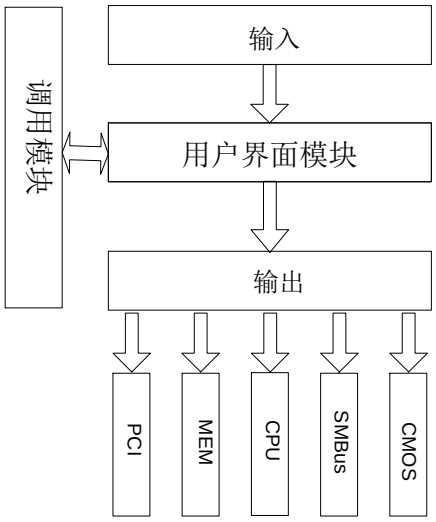


图 5-1 用户交互模块逻辑图

用户交互模块的用户相关接口主要包括输入处理接口和输出处理接口。输入处理接口通过捕获键盘信息获得用户的输入数据，将输入数据按照特定格式转化后传递给用户界面；输出显示接口负责处理从用户界面返回的数据，并根据所要求的显示格式在屏幕上进行输出[17] 。

5.1.1 用户交互模块的实现方式

1. 用户交互模块中输入捕获和输出显示的处理

用户交互模块设计的主要方面就是捕获输入数据和在屏幕上输出最终执行结果。采用 BIOS 中断的方式捕获用户输入数据，用直接写屏的方式处理屏幕显示。。

1）用户输入数据的捕获

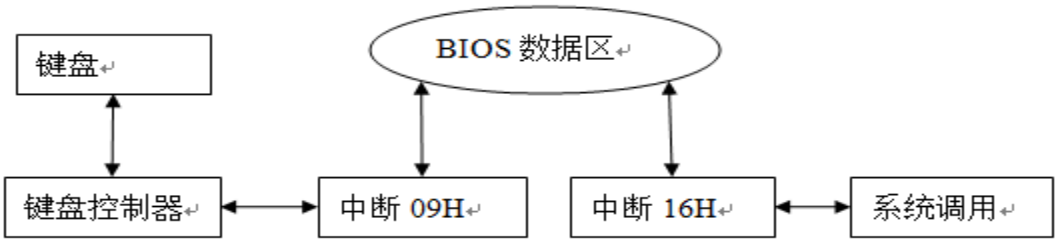


图 5-2 系统按键响应流程图

当有键盘上有键被按下的时候，键盘控制器 8042 将按键的系统扫描码放入自己的输出缓冲区里面，同时发出系统的中断请求，系统 9 号中断从键盘输出缓冲区读到数据，然后将数据转化为通用的 ascii，放到 BIOS 数据区的键盘缓冲区段。

用户交互模块使用 BIOS 的 16 号中断的键盘服务程序来访问键盘缓冲区，首先使用 1 号功能检查是否有按键动作发生，有动作发生时使用 0 号功能调用来获得按键动作的键值，这样就确认了按下的是哪个键，从而交给上层程序来判断如何处理，如果有功能键，譬如 FN 或者组合键的时候，可以用其他 16 号中断调用的辅助功能来实现。

2）输出显示的处理

本工具提供了用户交互的操作，用户在输入数据后程序会处理然后将处理结果显示到屏幕上，显示到屏幕的方法有很多种，通过系统中断或者通过 BIOS 中断都可以做到，在此选择直接写屏幕的方法，因为直接对内存操作，所以动作快，而且不依赖于 BIOS 和任何操作系统。

为了便于用户在交互的时候使用鼠标和键盘定位查询和修改的数据，同时也减

少图形模式对于图像像素的计算和不同硬件设备的不同编程需求，本设计采用文本显示模式。显示字符的格式是 80 行 25 列，这也是 DOS 操作系统的主流显示方式，很多程序的开发也是基于这个模式。

文本显示的缓冲区位于物理内存的 B8000H 的开始位置，第一个地址存放的是屏幕左上角的字符的属性，根据业界的标准定义，属性包括字符的背景色，前景色和高亮选项。第二个地址存放的是显示字符的 ASCII 编码。从屏幕的左上角到右下角，都依照上述两个字节决定一个字符属性的方式排列。所以在本章节的直接写屏幕的技术中，只要根据屏幕的位置计算出在内存的哪个地址，就可以通过改相应地址值的方式实现直接写屏幕的目的。

5.2 主调用模块的逻辑图

主调用模块主要起到桥梁的作用，它是用户交互界面和主功能实现模块的中间人，是一个中间模块，主要在数据上跟用户界面模块和主功能模块进行交互[28]，用户在用户交互界面进行了数据的输入，主调用模块就负责将用户的输入数据进行转化，转化成主功能模块可以识别的数据，同时也接受主功能模块传回的数据，显示给最终用户，这个章节给出主调用模块的逻辑图和建议使用的部分数据结构。

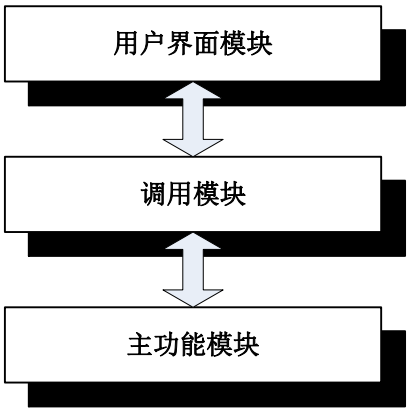


图 5-3 调用模块系统框图

主调用模块是用户交互界面和主功能模块的桥梁，起到数据衔接的作用，主调用模块的数据来自于上行和下行两个方面，第一方面是用户交互界面，用户交互界面有按键动作的时候，主调用模块启动，第一步进行消息解析，如果消息格式不正确或者误判或者包含本程序不能识别的信息，主调用模块将信息反馈给用户交互界面，从而用户有了进一步纠正输入的机会，第二方面的数据来自于主功能模块，用户交互模块对收到来自于主功能模块的数据进行响应和解析，按照特定的格式将数

据传给用户交互界面，由用户交互模块显示在输出设备上。

5.2.1 主调用模块的实现方式

主调用模组主要的工作是数据转换，架设用户交互模块和主功能模块之间通讯的桥梁，使这两个模块能够协同工作。

提供一个实例说明，以 PCI 设备作为参考。

1，PCI 设备的数据结构

```
typedef struct _PCI_BASE_TYPE
{
    char PCI_ComanyS_String[16];
    uint32 PCI_DID;
    uint32 PCI_VID:
    uint8 PCI_BUS_NUMBER;
    uint8 PCI_BUS_DEVICE;
    uint8 PCI_BUS_FUNCTION;

}PCI_BASE_TYPE, *PCI_BASE_TYPE;
```

从总线，设备到功能，对于给定的值能够唯一的确定这个设备。

2．用户交互模块数据结构

```
typedef struct _PCI_GUI_TYPE
{
    PCI_BASE_TYPE    PCI_Specific_Device
    Uint32 GUI_POSITION;
    Uint32 GUI_FLAG;
}PCI_GUI_TYPE, *PCI_GUI_TYPE;
```

用户交互模块的数据结构体现了对于用户输入信息的处理和准备写屏幕前的准备工作。针对 PCI 设备需要访问到的数据结构，此数据由程序在有键盘动作按下后进行提交，它对应某个确定的 PCI 设备，GUI_POSITON 记录了用户进行交互操作时对应屏幕的位置，譬如屏幕左上角的相对位置，同时 GUI_FLAG 里面记录上屏幕控制信息。

3，主功能模块数据结构

```
typedef struct _PCI_F_STRUCTURE
{
    PCI_BASE_TYPE    PCI_Specific_Device
    Uint32 FUNC_FLAG;
}PCI_F_STRUCTURE, *PCI_F_STRUCTURE;
```

主功能模块是处理硬件信息的主要工作模块，利用它的数据结构达到对于子功能模块的控制，其中 PCI_BASE_TYPE 指向特定的 PCI 设备，FUNC_FLAG 主要保存主控制段的控制情况[8]。

当用户交互模块有数据输入时，主调用模块启动用户输入模块的程序，开始解析用户的输入请求，同时将 PCI 设备的地址记录到主功能模块，同时开始转化两个结构体里面的控制信息，起到桥梁的作用，是两个模块能更好的协同工作。

5.3 本章小结

综上所述，这个章节集中介绍了用户交互模块和主调用模块，提供了系统实现所需的框架图，简化了用户交互模块和主调用模块的工作流程。接下来的第六章将着重阐述各个子功能模块的实现方式和所使用到的方法，更方便在 UEFI Shell 下除错工具的实现。

第六章 主功能模块的实现

本章节介绍了主功能模块的实现过程，使除错工具的开发有了更详细的实现框架分析，在主功能实现的过程中同时也实现了对子功能模块的软件实现做了较为详尽的叙述，并举例了常用子功能模块的软件工作原理，使用软件进行各模块的除错访问设计，使用系统框架图分析了主功能模块的实现方式。

6.1 主功能模块的系统框图

在计算机除错流程中，对各种计算机硬件的访问功能是由主功能模块实现的，用户可以在调用模块里设计自己的除错需求，主功能模块接到用户调用模块的指令后，把收到的用户除错信息分配给相关的子功能模块执行，子功能模块把执行好的除错结果返回给调用模块,这样各模块间的相互配合执行来完成用户除错功能实现。

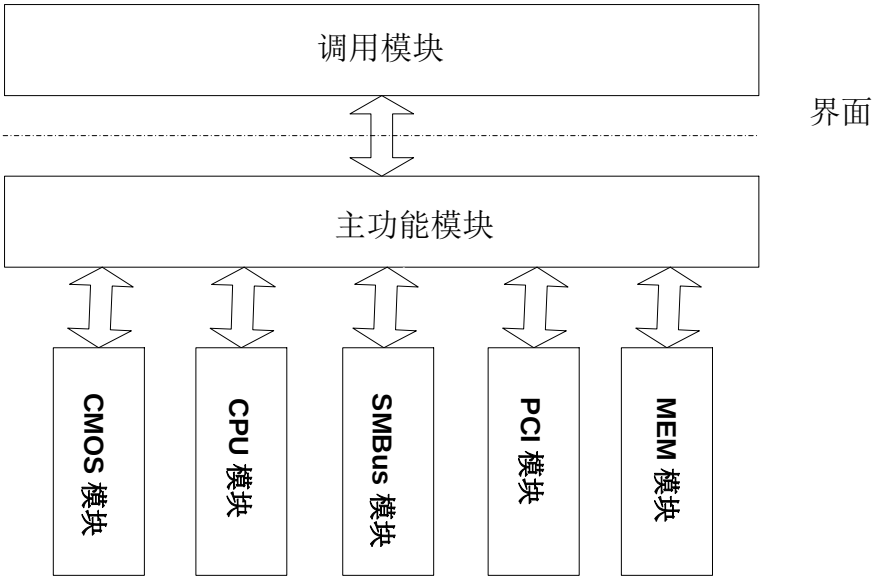


图 6-1 主功能模块系统框图

主功能模块在整个系统中的功能有两部分组成，它的系统模块框架如图 6-1 所示。主功能模块的第一部分是功能是实现子功能模块的除错信息访问，它收到用户除错的指令后，可以进入相对应的子功能模块，其工作任务是对硬件进行数据等信息访问，最后对访问的结果进行信息回馈；主功能模块的另外一个部分就是外部接

口的访问，负责在调用模块间建立桥梁联系，对调用模块获得的用户除错指令信息进行解码分析，然后根据实际功能要求派送给相对应的子功能模块执行处理。在整个系统框架中，主功能模块和调用模块的系统功能实现在前面的调用模块的章节中已经做了叙述，本章节介绍另外一部分子功能模块的设计实现。

6.2 各个子功能模块的功能的实现

主功能模块系统功能流程图在前面作了介绍后，这个单元主要针对其下的各个子功能模块的设计实现作了进一步的分析，在硬件除错软件设计中，硬件设备信息都是通过子功能模块与主功能模块的信息交换实现的，所以在整个论文除错设计的实现有着举足轻重的作用，所以下面将对主功能模块下每一个子功能模块的交互端口的访问，顺利实现除错工具在子功能模块中的设计应用。

6.2.1 内存（Memory）访问模块的实现

用户通过调用模块的请求，触发内存模块的访问，并把访问地址空间内的数据内容返回给调用模块。所谓内存访问就是用程序指令来控制使用数据，除错程序执行前数据已经存放在计算机的内存中，程序是通过地址来访问的，C 高级语言除了通过地址访问还可以通过调用变量名来实现获得内存数据。

1. 访问功能要求

调用模块发出请求给内存模块处理，它的功能要求是处理内存的读和写操作，并把获得的结果回馈给调用模块。它访问内存的宽度有三种方式：字节、单字和双字。该模块内存访问地址可以从 0H 到 4GH。

2. 内存模块的实现原理

根据内存模块访问的要求，内存访问功能实现的最大地址寻址范围可以到 4GH（00000000H~0FFFFFFFH）。但从内存访问模式的基本介绍得知，在实模式下只能允许访问到 1M 大小的地址空间，因此只有通过软件的方式进入大实模式或保护模式后才能访问到 4GH 的地址空间。本硬件除错设计是采用大实模式的方法访问的，其用软件加载方式实现的具体流程图如 6-2。

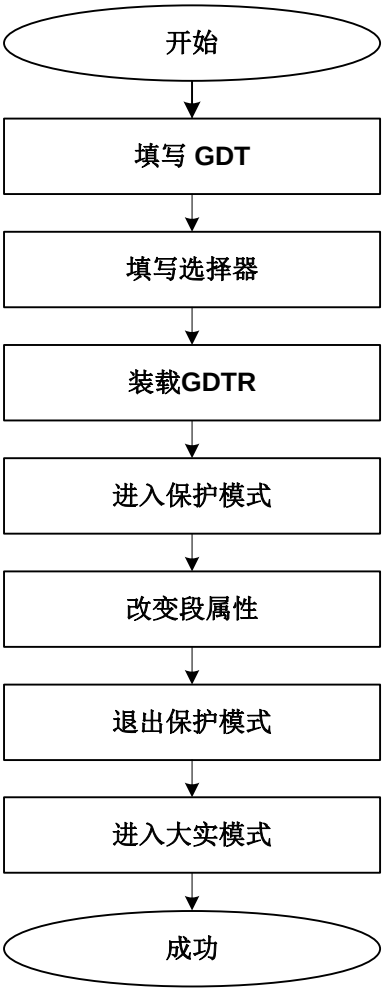


图 6-2 大实模式软件加载过程图

1）数据结构的准备

书写定义全局描述符表格，在这个表格里填写了在保护模式环境下能调用的描述符，各种段的属性详细定义在描述符里。

记入段选择子的重点是一定要与描述符表的位置一一应对，描述符表的地址是 8 个字节的，那么它的地址范围一定要 8 的整数倍，这种编写对应方式程序才能被正确的调用。

2）装载描述符表的准备

CPU 的全局描述符表寄存器里面记入着全局描述符表地址，这个寄存器里的全局描述符表地址要在进入保护模式之前装载进来。GDTR 寄存器中的值是通过执行计算机特定的 LGDT 程序指令把描述符的表头地址填写进来，完成后把全局描述符表的起始地址传送给 CPU，段选择子可以通过这个地址调用到正确的内容。

3）进入保护模式

CR0 寄存器在 CPU 里有一个特殊的功能，第 0 位的内容表示此计算机 CPU 的工作模式。它里面的数值 0 代表是系统在保护模式下工作；数值 1 代表在系统保护模式下工作。如果系统要进入保护模式下工作，就必须把 CR0 寄存器的第 0 位内容写成 1，然后运行跳转指令动作，这个动作其实是为了清除 CPU 的执行任务序列和加载段描述符，确定 CPU 在系统中已经进入到保护模式。

4）段属性的改变

将预先定义好的段的选择子的值赋给指定的改变段，这样就建立了新的段属性，如果段的大小被程序改成 4G，那么段的属性内容马上会生效。通过改变段的属性来决定程序是否可以访问 4G 的内存空间，此子功能模块有字、字节和双字的读、写函数共六个。

5）进入大实模式，退出保护模式

在步骤 3 已经研究到，对 CPU 的 CR0 寄存器的第 0 位进行改写，然后运行跳转指令动作，这样 CPU 工作模式就进入到了实模式，唯一区别是与以前的实模式寻址范围不一样，因为被改写后实模式的段属性值也被改变，所以经过再次切换后，其实这个段的属性值已经具备保护模式的功能，所以这个段的大小也已经具备可以寻址 4G 的内存空间。

3. 设计内存模块的访问

在大实模式工作环境下，通过定义数据结构体来实现软件访问、如何实现加载功能。包括全局描述符表的数据结构、段描述符结构体和全局描述符表。

1） 定义段选择子的方法

```
REAL_DATA_SEL     EQU     10h             ; Real Data Select (64kB)
FLAT_DATA_SEL     EQU     08h             ; Flat Data Select (0-4GB)
```

2） 定义段描述符的方法

```
Descriptor              STRUC
Descriptor      ENDS

Attribute               db      0               ; Attribute(7..0)
LimitLow                dw      0           ; Limit Low Bit (15...0)
BaseHi                  db      0           ; Base High Bit(31..24)
LimitMid                db      0           ; Limit Middile Bit(19..16)
BaseLow                 dw      0           ; Base Low Bit (15...0)
```

```
BaseMid          db      0          ; Base Middile bit (23...16)
LimitMid         db      0          ; Limit Middile Bit(19..16)
```

3） 定义全局描述符表的数据结构方法

```
5RealData     Descriptor   <0FFFFh,00000h,000h,092h,000h,000h>
1GDTPointer                LABEL     FWORD
6Null         Descriptor   <00000h,00000h,000h,000h,000h,000h>
2GDTLimit                  dw        (GDTTableEnd-GDTTableStart-1)[29]
3GDTBaseAddress            dd
4GDTTableStart             LABEL     WORD
7FlatData     Descriptor   <0FFFFh,00000h,000h,093h,08Fh,000h>
GDTTableEnd                LABEL     WORD
GDTLimit                   dw
GDTPointer                 LABEL     FWORD
GDTBaseAddress             dd
```

4） 定义如何装载全局描述符表方法

```
mov          GDTBaseAddress,eax          ; 存放数据结果
shl          eax,4
add          eax,ebx
mov          ax,ds                       ; 实模式
Lgdt         fword ptr GDTPointer
lea          bx,GDTTableStart            ; 溢出值
```

4，定义内存模块的用户输入接口访问方法

```
[PMEM]                                   ; 内存访问模式
```

WB, 89;

WB -- > W (Write) 代表写模式，还有 R（Read）代表读模式；B (Byte) 代表信息是按照字节方式写入的，还可以通过 W（Word）双字节写模式和 D（Double）四字节写模式，读操作模式也是按照这样的规范即 RB/RW/RD。

89 -- > 代表写操作的起始地址，即 00000089H。

分号表示每行操作命令的结束标记，同一行不同项目的指令用逗号分界，大小

写字母都可以（通常软件编程用小写）。如读操作的定义可以用以下表示：

RD, 7890;

-- > 代表读取内存信息，起始地址是 00007890H，是按照四字节模式显示。.

WW, 68, 1234;

-- > 代表写入内存信息，起始地址是 00000068H，是按照双字节模式写入值是 1234H。

5. 访问内存模块的有关事项

1） 4G 内存段属性的访问

程序一直可以访问 4G 内存，已经退出保护模式时也一样可以访问，即使这个段的属性值已经是 0，也不用改变段的属性值，程序可以通过 32Bit 的偏移量来直接访问 4G 内存空间。

2）ES 和 DS 寄存器的保护

ES 和 DS 寄存器不仅可以在保护模式和实模式下都可以使用，为了软件开发人员在编写程序时不要把两种模式混淆使用，所以在保护模式下，程序员对于 4G 内存空间的访问最好用段寄存器 FS 和 GS，把 ES 和 DS 保护起来不使用。

3）段的概念和物理地址的转换

段的概念是在保护模式下可以通过对存储块的调整改变内存的大小，其中内存的大小和占用情况可以用段描述符来描述，逻辑地址有描述符的段偏移地址和段选择子的组合组成。分段管理机制再把它分到对应的线性地址，实际的物理地址通过分页管理器把线性地址转化而成；反之段里不存在分页管理器的话，那这个实际的物理地址就是线性地址直接对应地址，不用进行地址转化直接访问。

4）段属性的参数

段属性（Attributes）、段基地址（Base Address）和段界限（Limit）三部分构成了段的属性参数。

在保护模式下，线性空间的开始地址是由段基地址定义的。它的地址长度是 32Bit，因为基地址的寻址长度和基地址长度是一样的，所以这个访问模式和实模式环境下的不同，段边界地址一定要 16 的整数倍，它起始地址可以在 32Bit 线性地址空间内的任何一个段，任何一个地址位置开始。

在保护模式下，段界限定义段的大小，有 20 位地址显示段界限的长度，它的大小单位可以用 4K 字节来表示。单位大小是以段属性中一位 G 来标记，我们称这位为颗粒大小位。G 位是 1 时，代表段界限是 4K 为单位，20 位对应的段界限寻址范围从 4K 到 4G 字节，它的增量大小是以 4K 为一个单位；G 位是 0 时，代表段界限

是以字节为单位，20 位对应的段界限寻址范围从 1 到 1M 字节，同时它是以 1 字节为增量单位的。

线性地址范围大小是段界限和基地址的范围定义的。扩大段的容量大小是可以通过增加段界限来实现的。经过大量的程序证明，对普通数据段实现扩展到 4G 内存空间使用这种调整增加的方法是非常有效的，而且返回实模式时段的属性还是保持不变的。

6.2.2 访问底层 PCI 设备模块的实现

主功能模块通过调用模块把指令请求发给底层 PCI 模块处理，然后返回处理的结果，这样用户就可以得到 PCI 上相关的设备信息。

1. 访问功能要求

这个功能模块实现对 PCI 设备的读写操作访问，负责各种 PCI 设备的信息的读写，它可以访问到 256 个字节的 PCI 配置寄存器空间，访问方式一般有三种方式：双字节访问、字访问和字节访问。主功能模块可以通过它的操作读取得到 PCI 设备的功能信息，然后用户界面模块就可以获得主功能模块所得到的回馈，进行判断 PCI 设备的使用状态等。

2. 实现访问底层 PCI 设备模块的工作原理

1） 配置数据（Data）和地址（Address）端口（0CFCH 和 0CF8H）

配置端口有数据和地址两种，通过对这两种端口的操作访问，完成了访问底层 PCI 设备配置信息的访问，当访问请求信号由 PCI 设备发到 PCI 主桥时，把配置数据和配置地址的数据进行破译，从而使 PCI 主桥自动分配到是访问 PCI 上的哪个设备对这两个端口的数据进行解码工作，从而将访问请求分配到哪个 PCI 设备上，实现对 PCI 设备的寄存器访问和功能调用。

2） 半字节对齐要求的配置地址端口（0CF8H）

在访问 PCI 设备过程中配置寄存器一定要以半字节方式排列的，换句话讲访问时要保证清空最后两位在索引寄存器中的地址，这种输入为空的硬件解码特性是由 PCI 设备要求严格遵守编写规范的。如果要访问的设备配置寄存器不是半个字节对齐的方式编写的系统将不知道如何读取执行，所以这一特性要在使用数据配置寄存器是完全遵守，这样只要通过访问地址寄存器的地址就可以得到数据寄存器上的正确资料。

3） 访问 PCI 相关的基地址寄存器的实现

PCI 设备上有些配置寄存器是受保护的，所以那些配置寄存器上的信息是不能

够直接或完全进行读取访问的。需要按照不同的作业方式对那些特殊的配置寄存器进行读取访问，譬如说 010H~013H 基地址寄存器就是通过这种方式实现的，如果用户想获得该 PCI 设备的实际基地址，那系统第一步需先对基地址寄存器通过索引寄存器进行地址组合解码，然后在数据寄存器里写入 0FFFFFFFFH，这样 PCI 主桥会在 010H~~013H 寄存器里写入该设备分配到的基地址，最后实现对该 PCI 设备的读写操作。

3，设计底层 PCI 设备模块的案例

在下面列举了软件方式对访问 PCI 配置寄存器使用的方法。

1） 访问一般寄存器的实现方式

寄存器的内容访问方式：1 代表是 PCI 设备写总线号码；2 代表是 PCI 设备编号；3 代表是 PCI 设备功能号；4 代表是 PCI 设备配置寄存器的偏移量。

```
mov eax, 61100803H                ；Coding PCI device data1234
mov dx, 0AC6h                     ；Transferring the address data to the port
mov dx, 0B34h                     ；Transferring the data register to the port
out     dx, eax                   ；Out put the data to address data
in      ax, dx                    ；Get the result
```

2） 访问特殊配置寄存器的实现方式

特殊配置寄存器的访问目的：1 代表是 PCI 设备读总线号码，2 代表是 PCI 设备的设备编号，3 代表是 PCI 设备功能号；10H 代表是 PCI 设备配置寄存器的偏移量。当配置寄存器的内容全是 F 数值时，系统会强制写到配置数据端口，这样配置寄存器 10H~~13H 的地址空间会被主桥得到数据信息回重写分配给 10H~~13H 的地址数据。

```
mov eax, 61100210H                ；Coading PCI device data 12310
mov dx, 0AC6h                     ；Transferring the address data to port
mov dx, 0B34h                     ；Transferring the data register to the port
out     dx, eax                   ；Out put the data to address data
out   dx, 0FFFFFFFFh              ；All F data to return 10h and 13h
in      ax, dx                    ；Get the result
```

4. 设计相关用户自定义界面输入接口的底层 PCI 模块实现方式

[PCI]

解释：空间访问 PCI 界面配置信息

RW, 1, 2, 3, 4;

RW 代表是读取 PCI 界面配置信息，并且配置信息显示模式是以字读模式的，读取显示模式有三种方式， 分别是按双字读模式（RD）、字读模式（RW）和字节读模式。

1 代表是 PCI 设备总线号码

2 代表是 PCI 设备设备号码

3 代表是 PCI 设备功能号码

4 代表是访问 PCI 设备的起始地址空间

6.2.3 实现访问 CPU 核心寄存器模块的方式

相关 CPU 核心寄存器的访问请求有底层 CPU 核心寄存器模块完成实现，然后把 CPU 核心寄存器的处理结果值并返回到该处理模块。

1. 访问功能要求

访问CPU的关键寄存器的内容有底层CPU核心寄存器模块完成，它包括对CPU核心寄存器的读写操作，不但要具备 CPU 的关键寄存器的访问功能，而且绝对不能通过访问而造成计算机系统问题，功能模块得到 CPU 核心寄存器的访问结果后，然后负责把信息资料传送到用户界面模块上，这样的访问功能实现了除错信息的可读性。

2. 软件实现底层 CPU 关键寄存器模块的设计原理

逻辑运算和系统控制是 CPU 的主要功能，CPU 是计算机的大脑，BIOS 是计算机的神经系统，可想而知 CPU 在计算机系统的重要性，而且它的运算速度是最快的。它可以对计算机上的各个组件部分进行控制和部署，大量的计算机信息都存放在 CPU 内部的寄存器里。

1） CPU 内部寄存器种分类

用 8086/8088 CPU 举例，CPU 内部寄存器有以下几个类别：

IP，BP -→两个指针寄存器

AX，BX，CX，DX -→四个通用寄存器

SI，DI -→两个地址寄存器

Flag -→一个标志寄存器

CS，DS，ES-→三个段寄存器

SS，SP -→两个堆栈寄存器

CPU 从 80386 开始需要注意的是添加了两个段寄存器分别是 GS 和 FS，变成有五个段寄存器组成，新增的两个段寄存器被称为段选择子，它的功能比其它普通的

寄存器更多，同时其它的寄存器大小也增加到了 32 位长度。

2） 访问 CPU MSR（Model Special Register）寄存器实现方式

CPU 的所有信息都存放在 MSR 里面，它包括我们通常一直衡量 CPU 速度的频率和倍频，还有一些 CPU 的工作条件如 CPU 平台特性、CPU 的工作时间和 CPU 睡眠功能等。

MSR 寄存器是长度是 64 位，它的访问是通过 CPU 的 ECX 寄存器的索引来实现的。EDX 寄存器里存放着读写的结果。高 32 位信息存放在 EDX 里；低 32 位信息存放在 EAX 的组合里面。RDMSR 和 WRMSR 是访问 CPU 信息的特殊指令功能，即写 MSR 的信息（WRMSR）和读取 MSR 的信息（RDMSR）。

CPU 的 MSR 信息被用户输入访问时，调用模块获得子功能模块的信息后，把信息译码成 CPU 能读懂的格式传给 CPU 执行，然后把运行的结果回馈给调用模块。这里需要注意的是不是所有的 CPU MSR 信息可以被直接调用访问，它们的信息访问要通过特殊的流程和访问条件才能被读取，因为那些 MSR 信息有很强的依赖性。

3. 设计底层 CPU 核心寄存器模块的实现方式

```
Call special_do_input                ; call special request
Mov ecx，input                       ;Put request data to input
JC RETURN
JMP RETURN
RDMSR                                ; read MSR information
RETRN
ERROR
```

标志寄存器的 Carry 位可以表示子功能模块的调用结果。EDX 值表示调用成功；EAX 值表示调用错误。子功能模块把调用的结果返回给调用模块，用户界面再通过传递等到出错信息。

4. 设计 CPU 模块相关的自定义用户输入界面接口实现

[CPUMSR]

CPUMSR 的指令功能是 CPU MSR 信息的访问。

RB, 8;

RB 代表的意思是读取 CPU MSR 的信息，并且信息显示是以 64 位显示模式的，这里的原因是 CPU 的相关特性的寄存器全以 64 位结构的，因此仅仅应用其中的一种显示模式，未来的研究中功能可以扩展到 128bit，256bit。

8 代表的意思是访问的 CPU MSR 起始地址。

6.2.4 访问底层 SMBus 总线设备模块的实现方式

访问 SMBus 总线设备的用户请求主要有底层 SMBus 总线设备模块来处理，然后将完成后结果回馈给调用模块。

1. 访问功能要求

访问 SMBus 总线设备的信息访问，它的功能主要是对设备信息的字节模式读写操作动作完成，使调用模块可以获得访问的结果，然后调用模块和用户界面模块接口间进行信息传递，这样用户可以清楚的看到 SMBus 总线上的设备信息情况。

2. 实现软件访问底层 SMBus 总线设备模块的工作方法

SMBus 传输中的 8 组寄存器是有 SMBus 控制器的芯片组提供完成的，在 SMBus 总线协议的规定下，这 8 组寄存器在整个传输系统中是必须保证被提供支持的，这 8 组寄存器每个大小是 1 个字节，它们的地址范围可以被映射到处理系统的接口上，但映射的地址不能与其它的内容地址相叠加。控制器的芯片组也规定了这 8 组寄存器的基地址内容，但是寄存器的功能作用是由 SMBus 总线规定的，通过定义和生成不同的子模块来完成用户对软件编程接口的访问和控制。

系统管理总线基寄存器 IO base + 00 的定义：

表 6-1 SMBus 基寄存器

Bit	Function
4	*Failed*
3	*Bus Error*
2	*Device Error*
1	*INTR*
0	*Host Busy*

系统管理总线 IO base + 02 的定义：

表 6-2 SMbus 寄存器二

Bit	Function	
6	*Start*	
4 ~ 2	*SMBus*	*Command Protocol*
	010	*Byte Date Read/ Write*
	011	*Word Read/ Write*
	101	*Block Data Read/ Write*

系统管理总线寄存器 IO base + 03 的定义：

表 6-3 SMBus 寄存器三

Bit	Function
0 ~ 7	*Byte (Word) Offset*

系统管理总线寄存器 IO base + 04 的定义：

表 6-4 SMBus 寄存器四

Bit	Function
7 ~ 1	*Device ID*
0	*Read / Write*

系统管理总线寄存器 IO base + 05 的定义：

表 6-5 SMBus 寄存器五

Bit	Function
0~ 7	*Data 0*

系统管理总线寄存器 IO base + 06 的定义：

表 6-6 SMBus 寄存器六

Bit	Function
0~ 7	*Data 1*

系统管理总线 IO base + 07 的定义：

表 6-7 SMBus 寄存器七的定义

Bit	Function
0~ 7	*SMBUS Block Data*

访问 SMbus 设备信息本硬件除错工具是通过利用读写字节和读写字的方式来操作的，并完成回馈设备信息结果。

3. 访问底层 SMBus 模块的流程图实现

系统管理总线读字节程序的运行流程具体如 6-3 图所示。

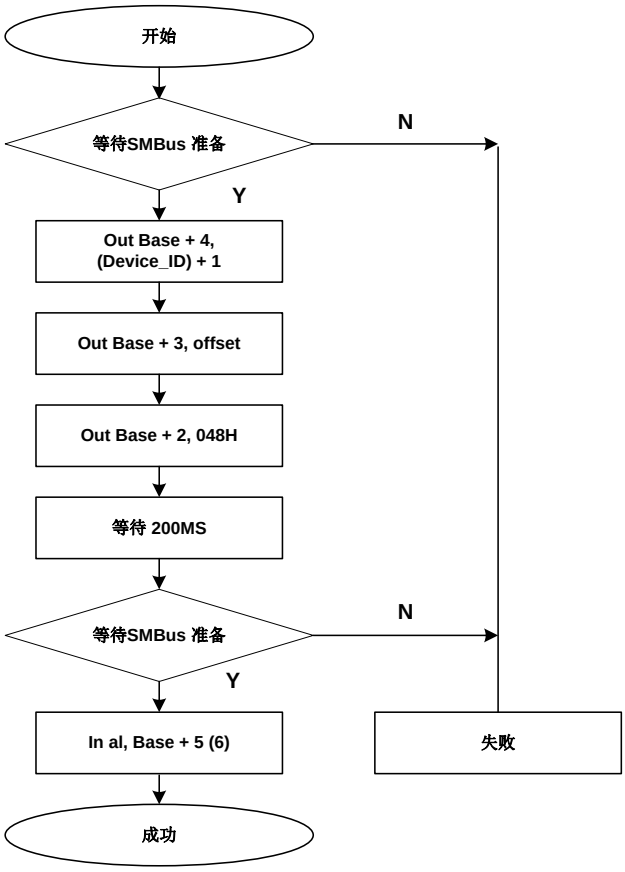


图 6-3 SMBus 的读字节流程图

4. 访问系统管理总线模块相关的用户自定义传输端口实现

[SMBUS]

功能是完成访问系统管理总线的存储区域地址内容

RD, 1, 2;

RD 代表是读取系统管理总线上的信息是通过双字显示模式来完成的，它的读取功能一样支持前面其它功能模块介绍过的字节、字和双字显示模式。

1 代表是要读取的系统管理总线的设备号码信息。

2 代表是要读取的系统管理总线设备存储空间的偏移量。

6.2.5 访问底层 CMOS 模块的实现方式

程序处理得到的信息内容再回馈到调用模块，是访问 CMOS 的需求用户是通过调用模块让底层 CMOS 模块来实现完成的。

1. 访问功能要求

CMOS 的信息内容是通过 CMOS 功能模块负责完成，它的访问操作包含字节模式的读写操作，使调用模块可以获得访问的结果，然后调用模块和用户界面模块接口间进行信息传递，这样用户可以清楚的看到 CMOS 中存放的所有信息。

2. 软件访问底层 CMOS 模块的工作原理

1） CMOS 的简介

CMOS 是 Complementary Metal Oxide Semiconductor 英语第一个字母的缩写，断电后它的信息需要有一块电池来进行保护[20]。我们通常称的 CMOS 其实是在计算机主板上一块可读写的的存储芯片（EEPROM）,它和前面介绍过的 BIOS 一样都存放着计算机启动所需调用的数据。数据包括计算机的系统时间、当前系统的硬件配置和计算机用户对系统参数的设定，CMOS 存储芯片虽然可以擦写，用户读写访问非常方便，但缺点是用户设定的信息在计算机断电后会全部消失，这种断电后就消除“记忆”的特性，计算机制造者就一般就在主板上安装一片钮扣电池来克服断电后的麻烦，这样信息资料在计算机关机状态中也不会丢失或初始化，那片钮扣电池通常我们称为 CMOS 电池。CMOS 电池是否有电最常见的方法是当每次开关机你的计算机时间总不是当前时间，那说明电池已经没电需更换了。

作为存储器功能的 CMOS 存储芯片而言，它本身的功能只是保存计算机的信息数据，所以我们可以通过专门的程序对计算机 CMOS 中各种信息进行访问。同时计算机生产厂商研发时看到 CMOS 和 BIOS 都具有存储功能，所以有的就把那片 CMOS RAM 的芯片位置设计在了 BIOS 芯片中，这样不仅方便了计算机设计的简洁性，而且也节约了材料成本，间接提高了计算机运行速度。当计算机开机时用户按定义好的热键（通常台式机是“Del”键；笔记本是“F2”或“F10”键）进行设置 CMOS 内容的动作称为 BIOS 设置，严格意义上讲 BIOS 设置是错误的，其实是 CMOS 设置，而 BIOS 不是一般计算机用户所能设置的，是通过 BIOS 程序员设置改写的。

2） 存储 CMOS 内容的存储结构定义

存储 CMOS 内容的芯片大小最初是 64 个字节，后来由于计算机的不断发展，硬件功能的不断强大和计算机用户的个性化设置也不断增多，这样导致需求计算机系统存放硬件初始化数据的空间地址要更加大，所以现在的 CMOS RAM 空间大小一般是 128 个字节，乃至 256 个字节。不过现在有些商用机和消费机电脑的功能需求保存数据信息更多，所以甚至经过计算机不停的发展过程中把存储芯片大小扩展到了 2K 字节，具体对 CMOS 存储芯片每个端口的作用和端口清单如下表 6-7 所示。

表 6-7 CMOS 端口清单

Location	Length	Description
00h – 0Fh	16 bytes	实时时钟数据
10h – 2Fh	32 bytes	ISA 配置数据
30h – 3Fh	16 bytes	AMIBIOS-特殊配置数据
40h – 7Fh	64 bytes	扩展 CMOS 存储空间，预先存放许多主板芯片的配置信息

在计算机设计架构中 Intel 架构属于市场主流，CMOS RAM 的存储空间大小一般都是 256 个字节，本除错设计也是进行对 256 字节大小进行研究访问，在未来的计算机实际测试调用中也可以支持扩充更多的访问空间[27]。CMOS RAM 中硬件配置信息和操作人员设定的存储空间位置都是有计算机厂商发布的，所以具体每台计算机的存储 RAM 规格同样可以参照芯片说明书，通过规格说明书可以具体获得存储芯片的内部存储信息结构和每个相对应的数据项功能。

3） 访问 CMOS 存储芯片的工作原理

CMOS 的存储区域是在独立与系统的编码空间之外的[8]，它不是在计算机系统内存的位置，计算机 CMOS 存储芯片的存储空间与一般的系统物理端口不一样，所以访问 CMOS 存储芯片的内容不能够像其它系统端口一样通过 IN 和 OUT 指令来调用访问，访问这个特殊的空间计算机系统提供了两个端口进行 CMOS 存储空间的访问问，它们分别是 70H 和 71H 操作端口。其中地址索引端是 70H；数据端口是 71H，通过相互配合完成访问 CMOS 存储芯片的地址空间，具体读写端口如表 6-8 说明。

表 6-8 CMOS 读写端口

Data Area	I/O port (Read/Write)	Size	Description
CMOS RAM data	070h & 071h	64 Byte	存储时钟，开机自检和系统数据

具体系统操作的时候，用户需要先读写数据的地址输出到地址索引端口 70H 上，系统进行破译收到的 CMOS 区域数据，当用户需要运行 CMOS 数据读取操作，71H 端口上就会有程序从索引端口地址访问地址的值传送过来的数据信息，进行数据读取；如果用户需执行 CMOS 数据的写动作，那么系统会把写入数据端口 71H 上值写回到实际的 CMOS 地址中去，其实用户实际写入数据的地址就是 70H 索引端口

地址。

用户在访问 CMOS RAM 地址还需要注意的问题是，用户输入给索引端口 70H 上的值只有第 0~~6 位才会被系统解码，其中第 7 位是不解码的，它的具体功能是实现关闭和打开 NMI（Non-Maskable Interrupt）中断，所以根据这个运行机制，系统利用索引端口 70H 和数据端口 71H 的互相调用来实现访问到 128 个字节的寻址范围[9]。如果碰到计算机系统平台较落后的 CMOS 存储空间大小只有 64 个字节，那么用户可以对超过 64 个字节的地址空间部分，系统会自动绕回运行寻址范围 40H 到 7FH 的空间，把地址空间对应到 00H 到 3FH 上去；如果操作用户访问 CMOS 内存地址 128 到 256 字节数据空间，那用户可以使用系统中的另外两个端口即索引端口 72H 和数据端口 73H。

具体访问 CMOS 存储空间的流程如如下图 6-4 模块表示，通过索引端口和数据端口的读写来实现所有 CMOS 内存信息的读写访问。

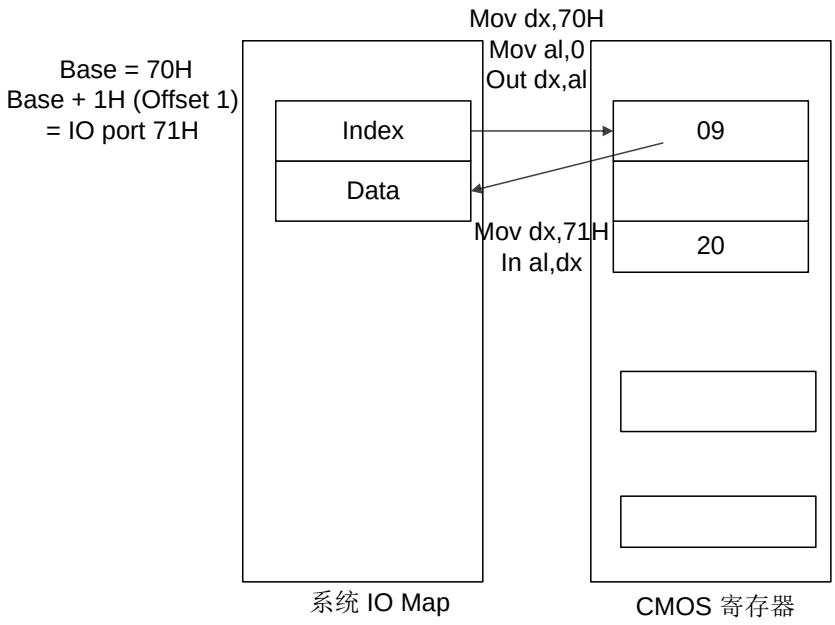


图 6-4 访问 CMOS 存储空间流程图

3. 访问底层 CMOS 模块的软件实现

本除错程序中 CMOS 存储空间的访问设计如下，用户也可以根据具体计算机系统中存储芯片的说明书进行结构调整。

outportb (0x70, address);

outportb 代表是用户执行读命令

0x70 代表索引端口

address 代表是需要操作的地址

char result

result=inportb(0x71);

result 代表的是回馈给用户模块的数据内容

inportb 代表是用户执行写命令

0x71 代表是操作的是数据端口

Get_CMOS 是 CMOS 存储芯片 0~127 字节的数据访问

4. CMOS 模块相关的用户界面接口定义

[CMOS]

访问 CMOS 存储芯片中的用户或系统配置信息

RD, 1;

RD 代表系统读取 CMOS 存储芯片的功能，并且信息的显示模式是按照双字显示模式的。用户同样可以选择字节/字/双字的显示模式对 CMOS 存储芯片空间地址的访问。

1 代表是用户需要读取 CMOS 存储信息空间地址的偏移位置。

6.3 本章小结

通过对主功能模块对用户进行除错的功能模块的访问调用，用软件实现相关子功能模块的调用，子功能模块经过执行得到的数据传送到调用模块，调用模块得到的设备信息有主功能模块统一处理，最终使用户模块可以得到计算机系统中各设备的基本信息和运行状，最终实现下面章节中的计算机系统整体功能检测。

第七章 UEFI BIOS 中内嵌除错工具的实现

预安装的固件包括所有的实现功能，包括用户交互和主调用模块，主功能模块都在预安装的固件里面得以实现。本章主要叙述了除错设计的一个重要性技术实现，就是实现在 UEFI BIOS 内预安装固件的方式，然后在 BIOS Shell 环境下进行除错。具体实现方式，实现对计算机系统设备及用户设备的信息访问和功能调用做了较完善的叙述：首先是计算机开机后 BIOS 对硬件设备的初始化；其次是如何实现在可选择 ROM 中加载硬件除错工具的方法；后叙述实现硬件除错工具在 BIOS Shell 环境中对计算机系统和用户设备的内外部调用功能实现。

7.1 BIOS 对计算机硬件设备初始化实现

系统在上电开机的过程中除了要复位和初始化计算机内部寄存器和内部工作电压外，还要对计算机底层硬件设备包括用户设备驱动和工作电压等初始化和复位，最终获得对所有硬件的管理权。计算机 BIOS 控制存储芯片上的信息是可以被 CPU 直接读取访问的，所有 BIOS 的功能不仅初始化和复位所有硬件设备，且要在完成此动作和对检测初始化的硬件明细产生硬件设备表，根据计算机系统的要求预先调用一些操作系统或用户环境需执行的信息内容，在生成系统管理表后检验用户启动设备是否工作正常，最后把把系统主控权交付给操作系统来完成其他功能的实现。BIOS 在硬件初始化的过程中，必须严格按照硬件原有的属特性进行设定，就是为了确保硬件工作的准确无误[21]。

硬件设备可分为两种情况的初始化：

1. 在 BIOS 源代码的添加或修改过程中，开发者可以根据 BIOS 存储芯片的规格说明和硬件厂商提供的硬件说明书上查询到控制芯片的相关初始化流程和配置寄存器等信息，可以直接把读取到的信息增加进来，使硬件设备得到正确的初始化，是针对没有 Option ROM 的硬件设备而言的。

2. BIOS 开发者不可能都完全预见到未来计算机硬件设备的发展，有些用户定制的设备或特殊要求的设备硬件信息资料所有使得 BIOS 对这些计算机设备不能完全初始化。对这些特殊设备的硬件初始化方式 BIOS 只能采用实时检测这些设备是否自带 BIOS 控制芯片，如果存在的话利用设备本身 BIOS 来实现对此设备的复位

和初始化。

计算机在上电开机后，BIOS 会对所有的硬件设备包括用户自己添加插在主板的 PCI、USB 等总线上的设备进行自动侦测，对发现的硬件设备中自带有 Option ROM 的进行复位检测，通过正常校验后把硬件板卡自带的 BIOS 芯片地址内容映射到计算机系统特殊指定内存空间的地址上，对这个设备的初始化和复位，对用户设备初始化完成后把主控权交换给系统 BIOS 进行下一个初始化动作，所以在内嵌过程中也读懂每一个段功能。

7.2 硬件在 Option ROM 方式中除错工具的开发使用

依据前面计算机 BIOS 所述可以得出结论，计算机硬件的初始化，可以由 BIOS 通过加载硬件 Option ROM 的特定方式来实现。下文将介绍开发者想要实现加载自己的除错工具，如何利用 BIOS 加载硬件 Option ROM 的方法，从而达到了除错工具预先加载到 Option ROM 后，启动加载后的 BIOS 进入 BIOS Shell 模式进行模拟测试硬件或检测客户反应的功能不良的硬件。

7.2.1 除错工具的 BIOS Shell 集成

通过格式处理完的除错工具已经符合了 Option ROM 的编码原则，设计者可以通过 EDK 的环境进行除错工具对计算机设备硬件的信息访问，为程序内嵌后做最后的调试，完成后通过 ROM 烧录器和应用程序对除错程序进行内嵌，成为 BIOS 固件的一部分，为后面硬件除错工具在 BIOS Shell 环境中运行奠定了基础。

7.3 实现处理 Option ROM 固件格式的方法

ROM 格式的严格定义规定对于实现设计硬件除错工具的加载以及判断过程的具有非常重要的作用，定义可选择性 ROM 的格式如表 7-1 所填写。

表 7-1 定义 Option ROM 的格式头

Offset	Length	Value	Description
3H	4H	Varies	初始化向量
0H	2H	AA55H	签名
7H	13H	Varies	保留
2H	1H	Varies	数据长度

7.3.1 解析栏位定义的格式

1. Length 是长度单位项，数据结构占用的长度。

2. Description 是说明项，解释数据结构的功能。

3. Value 是值项，定义数据的结构值。

4. Offset 是偏移量项，从 ROM 起始位置开始计算到数据结构项位置的地址差，它的偏移地址是数据结构项的[22]。

7.3.2 解析数据表项的应用方法

1. 第一个数据项描述的是初始化向量位，它一般保存着一条计算机跳转指令，是 ROM 主程序第一个起始调用处，它的大小是 4 个字节。

2. 第二个数据项描述的是签名项位，签名数的偏移量是 0H，它里面的数值固定定是 AA55H，如果签名项和与 ROM 的偏移位置的值来判断系统是否正常，对比值不一样就会马上退出 ROM 验证，只有一样的情况下，系统才会进入下一个校验环节。

3. 第三个数据项描述是数据保留位，这一项是计算机生产（OEM）或设计商（ODM）在设计时输入供应者的信息，BIOS 校验范围不包括此项，所以里面的内容也可以为空，它的大小是 13 个字节。

4. 第四个数据项描述的是数据长度位，长度的统计单位是以 512 字节作为一个单元的，所以数据项是 2H 可开始的偏移量时，ROM 的大小会比 1H 多增加 512 个字节即 1024 个字节大小[23]。

7.4 本章小结

本章节通过用软件程序来模拟硬件加载 Option ROM 之过程，表示测试除错工

具被成功装载在 BIOS ROM 中。其中最重要应该注意的地方是 ROM 的一些规范，了解了编码规则可以方便除错工具的正确编写和调用。如果不内嵌的话把除错工具存放在外部设备里直接在 Shell 模式下调用就可以，内嵌的完成可以满足常用功能的需求，可以供电脑购买或使用者对本电脑进行功能测试。设计者可根据用户要求安排除错工具的运行环境，顺利完成内嵌或外部存储工具对计算机功能检测。

第八章　系统整体功能检测与结果分析

8.1 除错器功能测试的意义

除错器开发完成以后需要进行全面的测试，用于检测针对预先设定功能的完成度，进行试错测试以提高除错器的纠错能力，这对于一个功能全面，使用可靠的产品是非常有意义的。检测的过程基本分为单项模块测试和整体模块测试，单项模块测试侧重于检测单项模块的功能性，用预先输入的数据检测输出，整体模块测试用来检测模块之间的配合度。

8.1.1 白盒测试

针对于程序的源代码，在各个功能模块相互调用的过程中，程序员自己设定条件，把预订参数直接输入到预订模块之中，然后查看输出结果。具体测试用例和参数将在测试流程章节详细介绍，在计算机生产商的实际运用中，一般白盒测试是程序开发者在调试使用。

8.1.2 黑盒测试

使用最终开发的结果，用户对各个模块进行检测，跟不同操作系统下的工具进行对比查看是否输出正确。具体测试用例和参数将在测试流程章节详细介绍，在计算机生产商的实际运用中，一般黑盒测试是产品工程师进行对程序开发者已经开放好的除错工具进行调试使用。

8.1.3 测试开发平台的选择

开发环境：Intel UDK2010，微软 VS2008 带 64 位开发包

实验环境：主机：Dell E4310　Win7 OS，UEFI Shell2.0，Shell1.0, USB Key 2.0GB

8.2 除错器的功能测试框图

针对除错器的每个功能模块都要设计相应的测试用例，好的测试方法应该涵盖功能模块地方每一个接口和数据传输，下图给出本测试用例的测试框图。

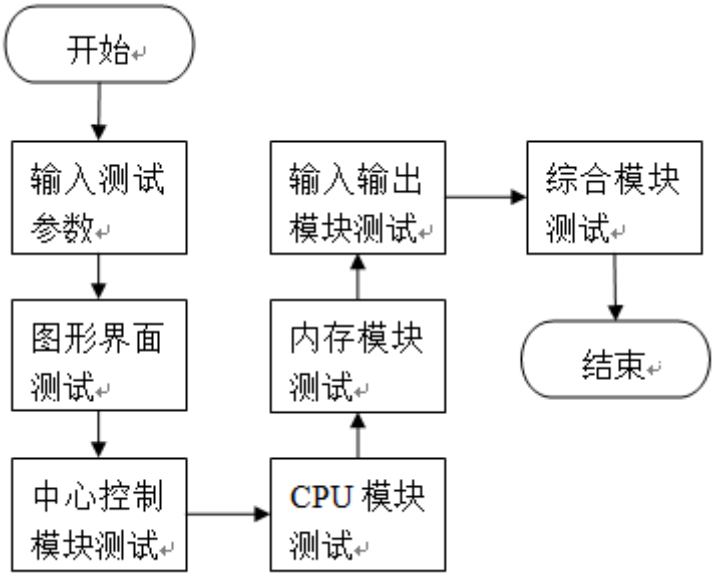


图 8-1 除错工具系统测试流程图

下面对测试流程图的每个阶段进行介绍。

8.2.1 输入参数模块测试

该工具支持图形交互界面和命令行界面，输入参数测试是针对于命令行界面进行测试，首先要针对所有在开发说明书里面列出的参数进行测试，看看是否所有的参数都有相应的输出，输出的结果是否正确，其次对于不支持的参数进行试错，查看是否有出错提示，设计的理念是系统应该对所有的输入有所回应，对的参数返回系统的运行结果，错的参数系统应该返回给用户的提示，以便于用户进行修改再输入。无论用户输入任何参数，系统都不应该出现没有反应，死机，系统重新启动等错误。

输入参数模块测试结果如下表 8-1 所示：

表 8-1 硬件除错模块测试结果

	tool.efi	tool.efi –h	tool.efi –cpu	tool.efi–mem	tool.efi –io	tool.efi –xx
功能划分	图形加载	帮助模块	CPU 模块	内存模块	IO 访问模块	错误输入举例
希望输出	加载图形界面	显示帮助信息	加载 CPU 模块，响应用户输入	加载内存模块，响应用户输入	加载 IO 模块，响应用户输入	显示帮助
二级参数简介	用户交互	无	CPU MSR 地址输入	内存地址输入	IO 端口号码输入	无
实际结果	通过	通过	通过	通过	通过	通过

8.2.2 用户界面模块测试

针对系统提供的用户界面进行模拟用户操作的测试，包括所有系统定义的键盘和鼠标动作，系统都应该有相应的反馈，在系统不希望用户输入或者输出的屏幕界面，系统应该将所有的输入功能禁用，鼠标的点击和键盘的输入事件应该没有回应，对于用用户的输入进行边界测试，使用超过和未达到系统需求的数值来测试系统的健壮性，对于数值的输入进行十进制和十六进制的交替输入，查看输出的状态。以下图示为用户界面测试的流程框图。

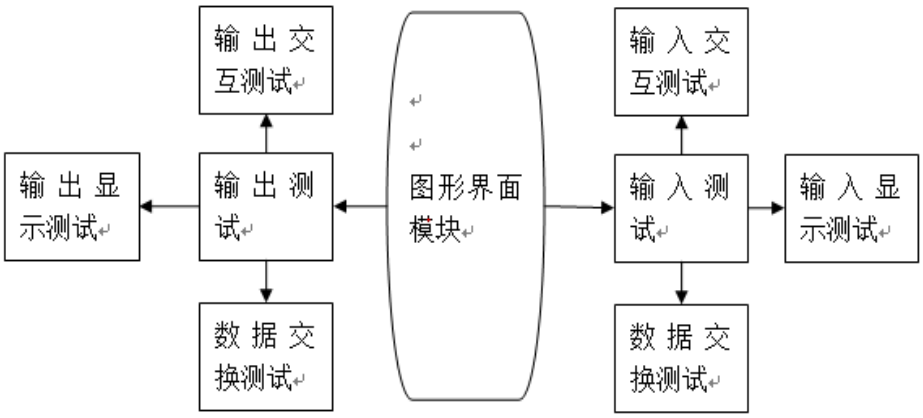


图 8-2 用户界面模块测试流程图

用具界面模块测试的快捷键用下表8-2：

表 8-2 除错模块快捷键清单

	F1	F2	F3	F4	F5	F6	F11
期望输出	帮助信息	CPU访问的人机交互界面	内存访问的人机交互界面	端口访问的人机交互界面	图形界面的命令行操作模式	文件操作模式	退出界面
结果	通过	通过	通过	通过	通过	通过	通过

用户界面模块除错测试输入的识别如表8-3：

表 8-3 测试除错输入识别

	数字	字母	符号	鼠标单击	鼠标双击
期望	10进制和16进制正确识别	16进制的识别，关键字的识别	关键字的识别	选中时间，选中单元颜色加深	对于选中单元出现编辑窗口
测试界面	除了F1	除了F1	除了F1	全部	除了F1
结果	通过	通过	通过	通过	通过

8.2.3 主控制模块测试

主控制模块负责进行各个模块之间的调用，这个模块的测试主要涉及参数的传递。因为这个模块是用户不可见的，只能使用白盒测试的方式。几个数据传输的模型定义如表8-4所示主控制模块数据传输参数：

表 8-4 数据传输模型定义

功能	定义	返回
内存值的读写	Bool OpMemory(adr,type,type,value)	0 或者 1
CPU 寄存器的读写	Bool OpMSR(adr, type, type,value)	0 或者 1
输入输出端口的读写	Boot OpIO(adr, type, type,value)	0 或者 1

8.2.4 各个主功能模块测试

主功能模块是除错工具访问硬件的主要部分，各个硬件的访问都通过主功能模块来实现，对于主功能模块的测试主要考虑两点，第一是功能性的实现，即每个功能模块都要实现预定义的功能，第二是调用的实现，即主功能模块和控制模块之间调用的准确性。

1. 输入输出模块的测试。主要测试每个输入输出端口是否能够读到和写入希望的数值，对于超过支持的端口，系统函数是否有正确的出错提示。

在除错过程中，用端口来调试结果如表 8-5。

表 8-5 端口调试结果对比

	0号端口	70端口	71端口	65535端口	65536端口
读入希望	FF	30H	55H	FF	FF
写入结果对比	无效	30H	55H	无效	无效

2. 内存模块测试。内存的读和写是否能够实现，另外内存的访问设计到字节，字，双字方式，这些工作方式是否正常。

在除错过程中，对内存进行调试结果如表 16。

表 8-6 内存读写结果

	内存地址100H	内存地址1024K	内存地址1024M	内存地址4G
读入	45H	44H	67H	FFH
写入	55H	55H	66H	无效
结果	通过	通过	通过	通过

3. CPU 模块测试。CPU 的寄存器返回值是否正确，关键寄存器 MSR 的访问是否正常，对于超过支持范围的寄存器是否有反馈机制。测试用例分析关键 MSR 寄存器的值是否正确。

8.2.5 模块整合性测试

通过分模块的测试，可以判定各个模块工作正常，这个单元主要进行模块间的整合测试，通过综合测试观察各个模块相互配合的能力和程序的兼容性，健壮性是否符合设计要求。下面是几个需要考虑的点。

1. 各个功能模块的交叉测试。参考上章节的测试用例。

2. 系统的兼容性测试，例如 shell1.0，shell2.0.，使用 Intel 发布的 shell 来观察不同的结果。

3. 系统的可靠性测试，例如多人多机，不同的 BIOS 厂商直接测试。在实验室环境下分别使用 Intel，Phoenix，AMI 的 BIOS 查看结果。

8.2.6 除错工具开发注意事项

通过测试，发现不少问题，有些可问题以在开发中避免，有些问题在开发的过程中应该随时注意，减少出现错误的机会。

1．应用程序里面库的调用。

1） 尽量使用系统现成的库函数。

2） 多使用指针调用以减少程序大小。

2．模块的功能实现

1） 注意各个模块之间数据的匹配关系，特别是函数类型的一致性。

2） C 语言本身的问题，例如野指针，内存泄露，变量的初始化，边界问题等。

3） 功能模块的硬件访问原理，要依照规范实现。

8.3 除错器实验结果分析

本除错器是在 UEFI 环境下访问硬件的编程实现。对于 UEFI 下面协议的创建和使用进行了透彻的分析，理解和应用，进而实现对硬件的访问和控制。

8.3.1 实验结果演示

1） 除错器的加载

除错器可以手动执行 EFI 文件加载，也可以通过开始脚本实现自动调用，类似目前生产测试中的批处理文件（autoexec.bat），程序可以根据用户的设定自动进行除错测试，然后把测试结果反馈到用户界面上或生成文件（Error Log）供程序自动文件。下面是如何通过批处理加载的图 8-3 所示。

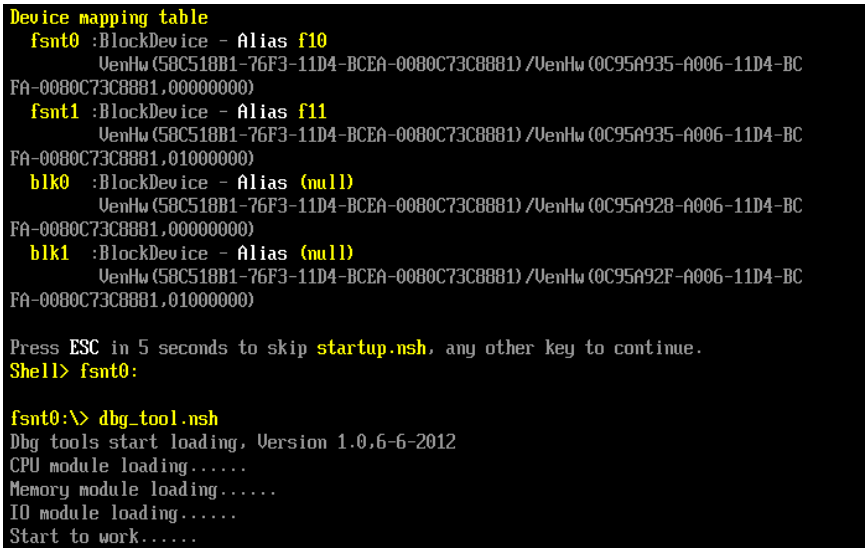


图 8-3 除错工具批处理加载图

2）除错器的运行界面

除错器可以工作在命令行模式，也可以工作在用户交互模式。

下面是命令行模式运行的图 8-4，用户可以手动进行除错，以节省批处理执行时间。

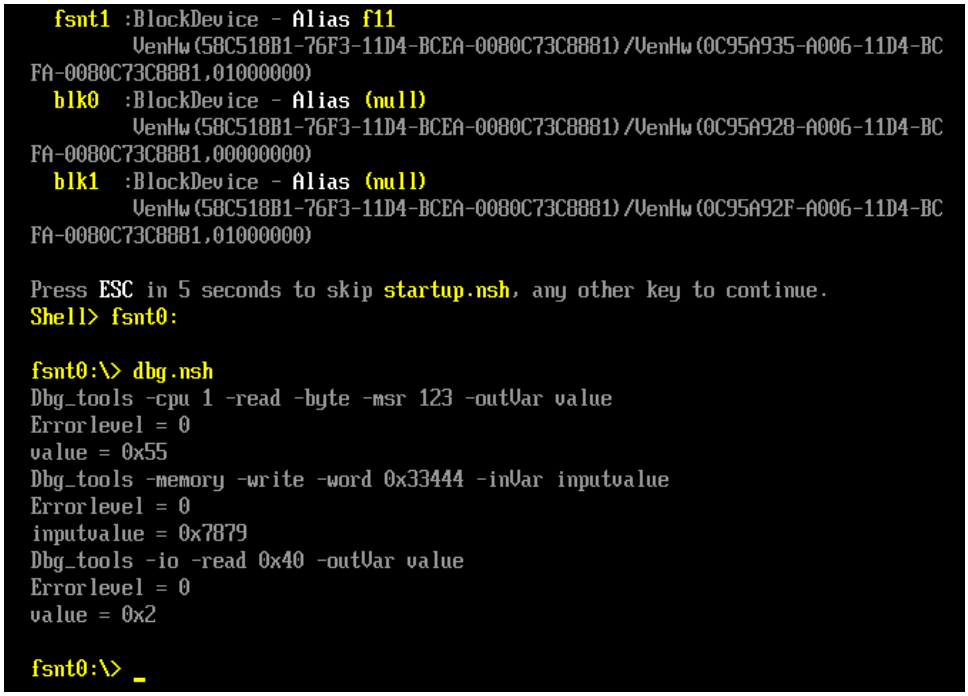


图 8-4 Shell 命令行模式图

优点，查看方便，直观，有错误返回提示，可以方便用户进行交互，如图 8-5。

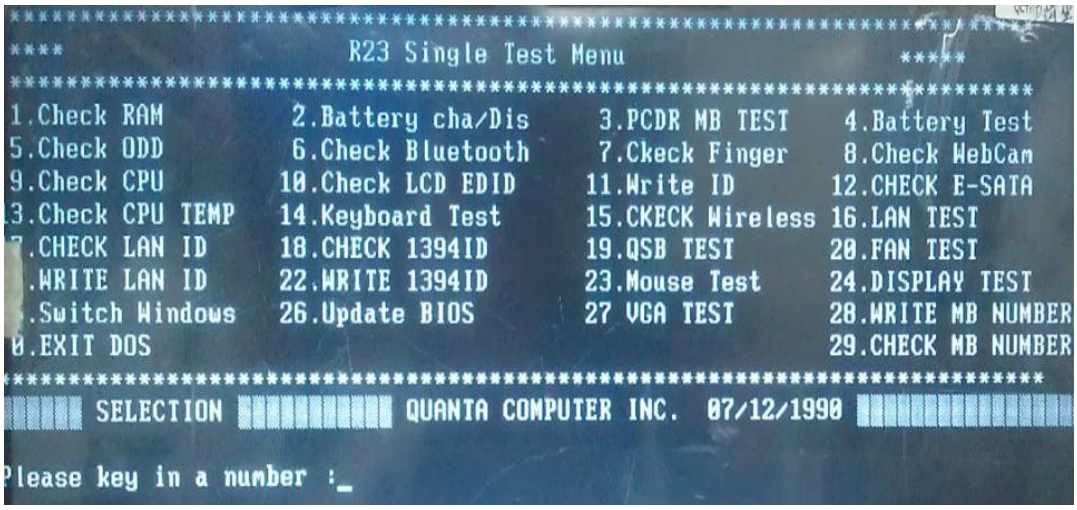


图 8-5　除错工具交互窗口

通过用户交互，可以将执行的结果在反馈给用户，除错结果PASS如图8-6所示；除错结果Fail的如图8-7所示。用户可以选择需除错的硬件设备，这样除错工具经过调用测试后反馈给用户除错的结果是PASS还是Fail。

图 8-6　除错工具 PASS 窗口

图 8-7　除错工具 Fail 窗口

8.3.2 实验结果分析

1）Shell 下面现存的除错工具

查看内存如图8-8：

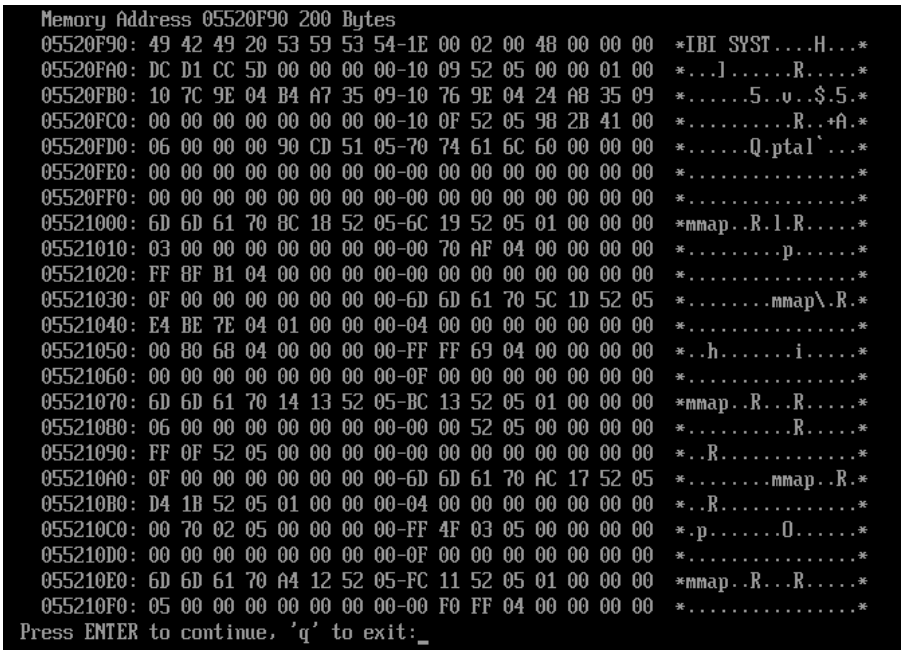


图 8-8 用户查看内存信息图

2）OS 系统里面的除错方式

主要通过系统资源管理器实现，如下图 8-9 所示

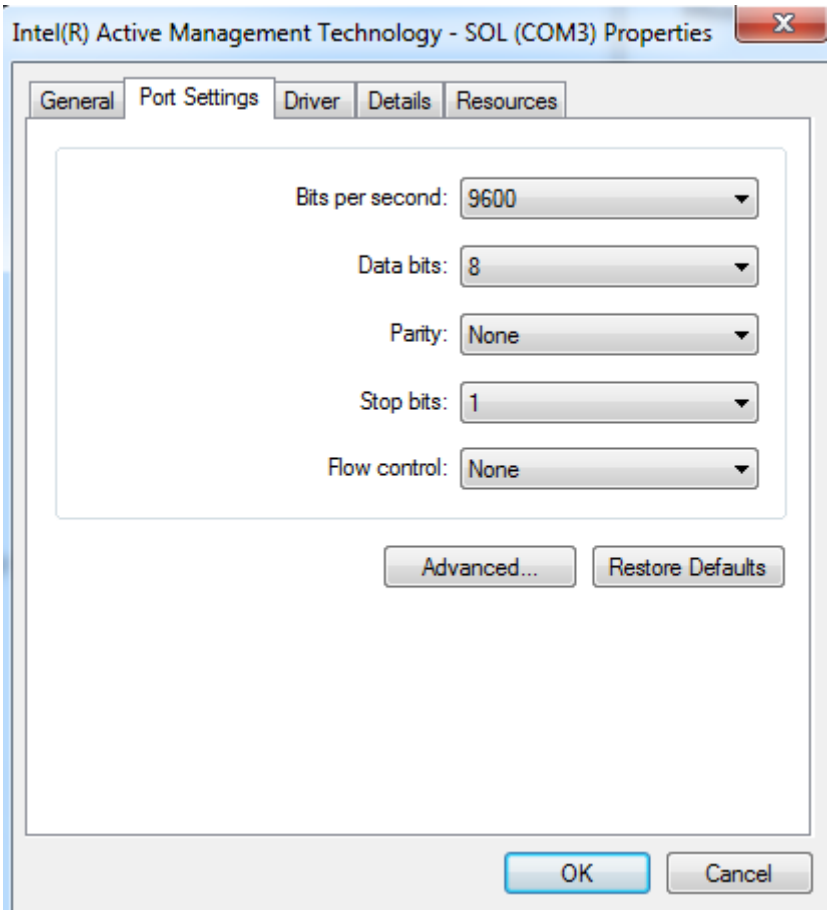


图 8-9 OS 下系统资源查看图

3）除错工具启动时间对比

从上面表格可以看出本除错工具在启动时间上非常具有优势，只比传统的 DOS 工具启动时间多一些，这是因为 UEFI 的 BIOS 启动的时间会略长，跟本工具没有关系。各个除错工具的执行时间比较如表 8-7。

表 8-7 除错工具运行时间对照表

	时间
DOS 工具	7’5
Shell 现存工具	14’5
OS系统工具	66’80
本除错工具	14’6

从除错工具运行时间对照表里可以看出本除错工具的测试时间是 14.6 秒，符合性能需求中的运行时间在 15 秒以内。

4）除错工具系统依赖性对比

从如上表格可以看出，本工具具有功能覆盖率高和界面友好的特点，因为BIOS会逐渐完全摒弃传统BIOS而切换到UEFI BIOS，所以本工具具有无可比拟的优点和不可替代性，具体比较如表8-8就可以体现出。

表 8-8 除错工具的优势对照表

	传统BIOS	UEFI BIOS	系统依赖	功能覆盖率	界面友好
DOS工具	Y	N	Y	N	Y
Shell现存工具	N	Y	N	N	N
OS系统工具	Y	Y	Y	N	Y
本除错工具	N	Y	N	Y	Y

从除错工具的优势对照表中可以看出本除错工具满足性能需求中的调用环境要求。

8.4 本章小结

本章通过用在Shell下对除错工具的功能检测和结果分析，同时也对在其它OS环境下执行作出了时间比对，更加说明了Shell下进行除错的可行性。

第九章 结论

论文主要介绍了一种在 UEFI BIOS Shell 下的除错方式，作者在实际的工作中碰到现有除错方式的落后，无法满足现在社会竞争中提倡的品质要好，交货要快，价格要便宜，介于公司的现状和目前计算机 BIOS 的现状这种除错方式会对企业减少测试时间和降低材料成本带来非常大的帮助。硬件测试除错的实现和系统工作原理，其中主要涉及的技术：内嵌 BIOS 的方法、Shell 环境的调试、计算机系统和设备的访问以及除错用户的设计。相信此除错方式的实现，对作者公司乃至其他计算机生产或售后维修单位有着更广阔的发展空间。使硬件测试除错在电脑生产过程中有很大的帮助，不仅缩短了操作人员测试的时间，而且也节省了治具硬盘费用。

但作者能力水平所限，虽然在 BIOS Shell 下的除错方式得到了较好的验证，但大量的除错还有待开发，所以从长远的计算机除错发展来看，后面有几点需要完善：

1．除错工具的通用性

随着计算机各种技术的飞速发展，每个电脑客户都会根据市场的要求，都会自己设计或定制不同的计算机配置，这样功能的设备复杂多样化，特别由于各种 Cost Down 的原因，材料生产厂商也不停的更好，导致相同功能产品也有不同的生产商生产。目前的工具只能根据设备的厂商来开发的，所以除错的单一性，其他产品或不同厂商就无法进行除错访问。所以在后续的开发中借鉴其他产品的规格书开发通用性的产品工具。可以集成在一个用户界面中，使用户界面更加强大，方便除错应用者大规模导入使用。

2．增加 UEFI 网络测试

在除错的过程中，发行 Shell 下的网络功能也很强大，在后续的生产或维修除错中可以利用这个功能在成本和速度上更上一层楼。可以通过网络进行操作系统的预装，或下载程序进行除错，包括制造厂的一些 Burn In 程序网络优化，这样比现有的磁盘安装或测试速度更快。就使用户界面更加友好，使用更加广泛。

3．优化用户界面

本设计主要采用人机交互的方式进行对计算机信息或计算机设备的访问，在大规模的应用中，靠人手动操作除错不是太方便，目前的工具只能提供给维修员根据客户的不良描述进行单测，如果没有客户描述单侧计算机上所有的功能耗时将更加多，所以后续还因优化用户界面的操作，当第一项的除错工具通用后用户界面可以

改成自动全功能检测，碰到功能问题的可以产生 Log 档案进行记录，这样用户也方便处理。甚至等上面第二项的网络功能强大后可以把数据通过网络传到 Shopfloor 中。方便电脑生产商对产品品质的管控。

随着计算机应用技术的发展，计算机生产商的制造技术也不断的发展进步，有一套先进完善的动作，在保证品质的情况下又能节约生产成本是企业生存的命脉。而且完善的流程会给更多的客人点头赞扬，客户的满意无形会带来更多的订单。

作者在实现硬件除错工具内嵌的同时，使终端用户都可以在购置新电脑时，不用借助在 OS 下的软件就可以在 UEFI Shell 下直接进行硬件检测。也提出了因 BIOS 大小的限制无法实现大容量内嵌时，也可以存放在外存设备里，同样可以在 Shell 模式下调用节省 OS 启动时间，所以这个系统将有广阔的应用前景，使电脑生产商看到更好的生产效率和经济效益。

致 谢

首先，作者要向学校的指导导师段贵多表示真挚的感谢。感谢她能够在工作的同时抽出时间一直关心着作者的课题以及论文的进展情况，在作者遇到困难和问题的时候能够及时的给予指导和帮助，帮助作者在工作之余解决了一些研究中碰到的问题，为作者提供了学习时间。而在其中，段贵多导师认真的工作态度以及灵活解决问题的方式给作者留下了极其深刻的印象，给作者日后的工作和学习都树立了一个很好的榜样。

其次，还要感谢企业方导师贾慧鹏，他具有解决设计调试问题的丰富经验，对于调试的思路和方法都有独特的见解，在作者课题论证过程中给予了许多建设性的意见和想法，并在课题开展过程中给予了很多帮助，为作者提供了独立的工作环境，更够让作者这次研究顺利结束提供了有力的保障，在今后更长的调试并导入中也是必不可少的。

最后，我要感谢我的家人，没有她们对我无微不至的关心，承担家庭所有的琐事，没有她们提供给我足够的学习时间和空间，我就不可能全身心地投入到学习和工作中来。

谢谢！

参考文献

[1] UEFI Inc. UEFI Spec 2.1 Copyright 2006, 2007 Unified EFI, Inc. 2007

[2] Vincent Zimmer, Michael Rothman, Robert Hale. Beyond BIOS.2006

[3]Intel Corporation.EFI Shell User's Guide.Copyright2000-2005 Intel Corporation.2005

[4]Intel Corporation.EFI Shell Getting Started Guide.Copyright 1999-2005 Intel Corporation.2005

[5]Intel Corporation, EFI Developer Kit (EDK) Getting Started Guide.Copyright 2005 Intel Corporation.2005

[6]Quanta RMA.Service Plan.Quanta Document Center.2001:P2-8

[7]Quanta RMA.TOP Testing Procedure.Quanta Document Center.2001:P9-16

[8]贾慧鹏.BIOS中内嵌硬件资源浏览器的设计与实现.上海交通大学硕士论文.2008:P7-39

[9] Michael Abrash.图形程序开发人员指南（前导工作室译）.机械工业出版社.1998:P172-201

[10]Marshall P.Cline,Greg A.Lomow.C++FAQs.Addison-Wesley.1995:P46-78

[11]Douglas E.Comer,David L.Stevens,Internetworking With TCP/IP.Vol iii.Prentice-Hall.1996：P56-57

[12]Michael A. Cusumano, Richard W. Sellby.微软的秘密（程化等译）.北京大学出版社.1995:P65-76

[13]Ivar Jacobson，Martin Griss.Software Reuse.世界图书出版社.1997:P201-202

[14]Geoffery James.编程之道（郭海等译）.清华大学出版社.1999:P67-68

[15]David J.Kruglinski,Scot Wingo.Programming Visual C++.北京希望电子出版社.1999:P50-71

[16]林锐，蔡文立.微机科学可视化系统设计.西安电子科技大学出版社.1996:P19-21

[17]林锐，戴玉宏.图形用户界面设计与技术.西安电子科技大学出版社.1997:P28-22

[18]林锐.支持协同工作交互式三维图形软件开发系统与可视化平台.浙江大学博士论文.2000:P3-4

[19]Steve Maguire，Writing Clean Code（姜静波等译）.电子工业出版社.1993:P113-116

[20]BIOS之家.初识UEFI结构.www.bios.net.cn.2008

[21] Roger S.Pressman.Software Engineeering:A Pracitioner's Approach(Fourth Edition).McGraw-Hill.1997

[22] Ronald J.Norman.Object-Oriented Systems Analysis And Design.Prentice-Hall.1996

[23]傲博.EFI/UEFI入门：All for Beginner. www.aub.org.cn. 2008

[24]余超志,朱泽民.新一代 BIOS EFI CSSBIOS 技术研究[J].科技信息(学术版).2006(5):P14-15.

[25]潘登.EFI 结构分析与 Driver 开发[D].湖南国防科学技术大学研究生院.2004:P378

[26]Unifled EFI Inc.Shell command reference manual revision 2.0. www.tianocore.org. 2008

[27]Unified EFI Inc.Unifled extensible firmware interface specification version 2.3. www.tianocore.org. 2009

[28]吴松青,王典洪.基于 UFEI 的 Application 和 Driver 的分析与开发[J].计算机应用与软件.2007, 24(2):P98-100.

[29]Unified EFI Inc.EFI developer kit (EDK) getting started guide version 0.41. www.tianocore.org. 2005-01-14

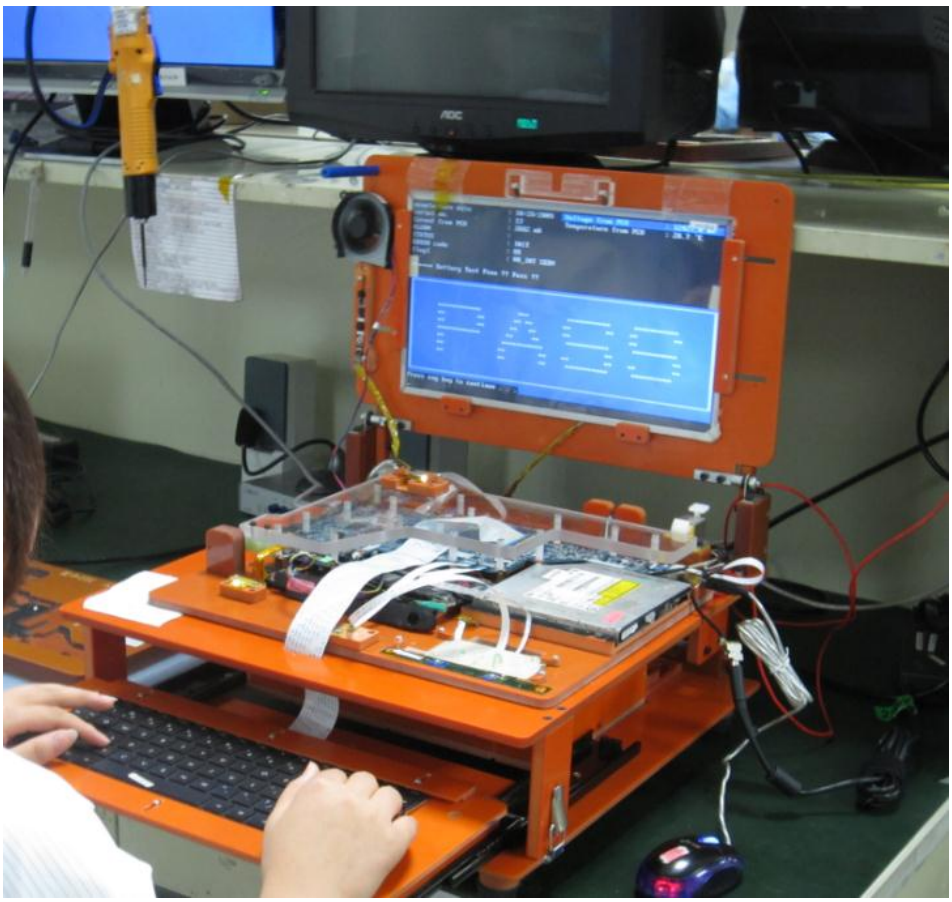
[30]Unified EFI Inc.EFI driver writer's guide version 0.9. www.tianocore.org. 2004-07-20.

[31]Unified EFI Inc. EFI driver library specification. www.tianocore.org. 2007

[32]倪兴荣. 基于 UEFI 技术的 API 性能分析设计与实现.南京信息工程大学硕士论文. 2010:P9-17

附录

附录[1] 系统实现测试环境



图中作者的测试环境，可以对笔记型主板进行全功能测试，或模拟客人故障现象进行除错测试。

附录[2] 系统实现测试平台

图中作者的测试平台采用了目前计算机生产商的测试平台，除错工具内嵌或在UEFI BIOS Shell 环境下调用存储在硬盘上的除错工具进行功能除错测试。

附录[3] PCI 设备除错工具源代码

```
/* Debug and link all the functions    */
#include <stdio.h>
/* read 32 port */
unsigned long inportl(int portid)
{
    unsigned long dwret;
    asm mov dx,portid;
    asm lea bx,dwret;
```

```
    __emit__(
    0x66,0x50,          /* push EAX */
    0x66,0xED,          /* in EAX,DX */
    0x66,0x89,0x07, /* mov [BX],EAX */
    0x66,0x58);         /* pop EAX */
    return dwret;
}
/* write 32 port */
void outportl(int portid, unsigned long dwval)
{
    asm mov dx,portid;
        asm lea bx,dwval;
    __emit__(
        0x66,0x50,          /* push EAX */
        0x66,0x8B,0x07,/* mov EAX,[BX] */
        0x66,0xEF,          /* out DX,EAX */
        0x66,0x58);         /* pop EAX */
    return;
}

/* show PCI devices */
unsigned long show_pci(unsigned busNo, unsigned deviceNo, unsigned funNo,
unsigned regNo)
{
    unsigned long config_reg_value = 0x80000000;
    config_reg_value += ((unsigned long)busNo) << 16;
    config_reg_value += ((unsigned long)deviceNo) << 11;
    config_reg_value += ((unsigned long)funNo) << 8;
    config_reg_value += regNo;
    outportl(0xcf8, config_reg_value);
    return inportl(0xcfc);
}
```

```
int main(void)
{
    unsigned short i = 1;
    unsigned short sym;
    unsigned short bus_no;
    unsigned short device_no;
    unsigned short fun_no;
    unsigned short reg_no = 0x00;
    unsigned long reg_ret;
    unsigned long vendor_id;
    unsigned long device_id;
    unsigned char act;
    unsigned long shift1 = 0x00ffffff;
    unsigned long shift2 = 0x0000ffff;
    unsigned long shift3 = 0x000000ff;
    /* choose functions */
    while(1)
    {
        printf ("Please select the functions:\n");
        printf ("\n");
        printf ("1. Show PCI devices which are exist\n");
        printf ("2. Show PCI configaration space\n");
        printf ("Other key: Back to Dos\n");
        act = getch();
        if(act == '1')
        {
            i = 0;
            reg_no = 0x0;
            printf ("Listing all the PCI device, please wait...\n ");
            printf ("bus_no\tdevice_no\tfun_no\tvendor_id\tdevice_id\n");
            for(bus_no=0; bus_no<255; bus_no++)
```

```
                {
                    for(device_no=0; device_no<32; device_no++)
                    {
                        for(fun_no=0; fun_no<8; fun_no++)
                        {
                            /* shift show_pci() function */
                            reg_ret = show_pci(bus_no, device_no, fun_no, reg_no);
                            /* if register is true */
                            if(reg_ret != 0xffffffff)
                            {
                                i++;
                                vendor_id = reg_ret & 0x0000ffff;
                                device_id = (reg_ret >> 16) & 0x0000ffff;

    printf("    %02X\t    %02X\t\t    %X\t    %X\t\t    %X\n",
bus_no,device_no,fun_no,(unsigned)vendor_id,(unsigned)device_id);
                                if(i%20 == 0)
                                {
                                    printf ("Press any key to continue...\n");
                                    getch();
                                    printf
("bus_no\tdevice_no\tfun_no\tvendor_id\tdevice_id\n");
                                }
                            }
                        }
                    }
                }
                else if(act == '2')
                {
                    i = 0;    sym = 0;
                    printf ("\nPlease enter the bus No. (Range is 0 ~ 255):\t");
```

```
scanf    ("%X", &bus_no);
if(bus_no>255)
    {
        printf ("Please note the bus No. range is 0~255\n");
        printf ("Please enter the bus No. (Range is 0 ~ 255):\t");
        scanf    ("%X", &bus_no);
    }
printf ("\nPlease enter the device No. (Range is 0 ~ 32):\t");
scanf    ("%X", &device_no);
if(device_no>32)
{
    printf ("Please note the device No. range is 0~32\n");
    printf ("Please enter the devive No. (Range is 0 ~ 255):\t");
    scanf    ("%X", &device_no);
}
printf ("Please enter the function No. (Range is 0 ~ 7):\t");
scanf    ("%X", &fun_no);
if(fun_no>7)
{
    printf ("Please note the function No. range is 0~7\n");
    printf ("Please enter the function No. (Range is 0 ~ 7:\t");
    scanf    ("%X", &fun_no);
}
printf ("Bus %d    Device %d Function %d,    the configuration space of
it is as follows:\n", bus_no, device_no, fun_no);
printf ("      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F\n");
printf ("%02X ", sym);
/* show PCI configuration space */
for(reg_no=0x0; reg_no<=0xFF; reg_no+=0x4)
{
    reg_ret = show_pci(bus_no, device_no, fun_no, reg_no);
    printf ("%02X ", reg_ret >> 24);
```

```
                printf ("%02X ", (reg_ret & shift1) >> 16);
                printf ("%02X ", (reg_ret & shift2) >> 8);
                printf ("%02X ", (reg_ret & shift3));
                i += 4;
                if((i%16 == 0) && (sym < 15))
                {
                    printf("\n");
                    ++sym;
                    printf ("%02X ", sym);
                }
            }
        }
        else break;
    }
    return 0;
}
```