RT Embedded http://www.kontronn.com

MIPS架构计算机平台的支持固件研究

杜振龙[1]，沙光侠[1]，李晓丽[1]，王庆川[2]，沈钢纲[1]

(1. 南京工业大学 电子与信息工程学院，江苏 南京 210009；2. 南京百敖软件有限公司，江苏 南京 210061)

摘要：通用可扩展固件接口UEFI定义了操作系统和硬件之间的接口规范，为嵌入式系统和计算机平台的支持固件开发提供了框架.龙芯系列CPU是32/64位MIPS架构国产CPU，UEFI目前不支持MIPS架构处理器，因此亟需研究MIPS架构计算机平台的支持固件.分析利用EDKⅡ在构建龙芯支持固件出现的引导模块无法定位和例外中断向量地址入口不符合规范等问题，通过在固件文件卷零向量填充SecCore的映像起始地址使固件系统正确加载SecCore模块，利用PAD文件以填零方式使例外处理向量位于固定地址.以龙芯为例研发支持MIPS架构的计算机平台固件系统，使配备龙芯系列处理器的计算机系统能正常驱动.

关键词：MIPS；龙芯；UEFI；固件

中图分类号：TP391.4　　**文献标识码**：A

Investigation of support firmware for computer platform with MIPS architecture

DU Zhen-long[1]，SHA Guang-xia[1]，LI Xiao-li[1]，WANG Qing-chuan[2]，SHEN Gang-gang[1]

(1. College of Electronic and Information Engineering，Nanjing University of Technology，Nanjing 210009，China；2. Nanjing Byosoft Company Limited，Nanjing 210061，China)

Abstract：The interface specifications between operation system and hardware are defined by unified extensible firmware interface (UEFI)，which provides a framework of support firmware development for embedded system and compute platform. The Loongson series CPU belongs to 32/64 MIPS architecture processor，UEFI does not support the MIPS architecture processor at present，so to develop a support firmware for MIPS architecture processor is urgent. The problems of unable to locate the initial module and inconsistency of exceptional interruption vector address with the specification，which would appear when EDKⅡ was used to construct loongson support firmware were analyzed. By means of filling SecCore mapping start address into the zero-vector of the firmware file volume，the firmware system was loaded properly with SecCore module，and by using PAD file，locate at fixed address in the form of zero filling. Taking Loongson as an example，the firmware system for MIPS architecture processor was developed，making the computer system with Loongson series processor to be able to normally driven.

Key words：MIPS；Loongson；UEFI；firmware

龙芯系列CPU是中国科学院计算所自主研发的通用32/64位CPU[1-2]，采用类MIPS精简指令集，且支持MIPSⅢ指令集和一些专用指令.龙芯系列CPU的发展需要依赖上、下游的产业链的支持，因此，如何扩展龙芯处理器的应用领域，增加龙芯产品的市场空间，已经成为国内众多IT企业共同关注的问题.龙芯CPU架构、指令集与X86 CPU完全不同，传统的固件不能直接应于龙芯CPU，因此研究MIPS架构计算机平台的支持固件对配备龙芯处理器的计算机系统驱动具有重要的现实意义.

EDK(EFI development kit)[3-4]是遵循UEFI[5]规范的框架核心代码实现，是开发EFI驱动、应用程序的软件架构平台[6].EDKⅡ是在EDK的基础上进行功能拓展和优化，形成了功能更全面、更安全的BIOS开发框架平台.EDKⅡ支持X86之外的硬件，支持在Linux操作系统使用GCC作为开发环境[7]进行平台的固件研发.

收稿日期：2013-03-01

基金项目：教育部高等学校博士点基金(20113221120003)，江苏省六大人才高峰基金(2012-WLW-023)，江苏省高校自然科学基金(09KJB520006，11KJD520007)

作者简介：杜振龙(1971-)，男，陕西韩城人，博士，副教授.

RT Embedded http://www.kontronn.com

目前 EDK Ⅱ 支持 X86 和 ARM 架构 CPU，尚不支持 MIPS 架构 CPU，因此不能直接编译出可运行在龙芯平台的 UEFI 固件和 EFI 执行文件，需要利用 EDK Ⅱ 构建支持 MIPS 架构处理器的固件系统. 研发固件系统需要确定合适的开发工具链，目前较多使用 VS 2005(Visual Studio. NET 2005)作为工具链. VS 2005 可编译出 MIPS 32 指令集目标代码，但龙芯的指令集和 MIPS32 的指令集并不完全兼容，因此要编译出龙芯处理器的支持固件还需要对编译器做相应的修改. 本文选择在 Linux 环境下使用 MIPS-ELF-GCC 编译器编译适合龙芯处理器的固件系统. 支持 MIPS 架构计算机平台的 UEFI 固件系统须有正确的 EFI 可执行文件，在加电后定位至 SecCore 模块，使例外处理函数位于映像文件的固定位置. 只有这些问题得到解决，才能研发出能驱动 MIPS 架构计算机平台的固件系统.

1 UEFI 固件文件系统分析

为了在有限的存储器空间有效地存储固件，并有利于文件的组织与访问，UEFI 在 PI(platform initialization)规范中定义了固件文件系统 firmware file system(FFS). 固件文件系统是 MIPS 架构计算机平台支持固件的开发基础，本节主要分析 UEFI 固件文件系统的组成和特点.

1.1 UEFI 固件文件系统组成

固件设备 firmware device(FD)是物理的存储空间，用来保存固件数据和代码. 通常的存储固件设备是 Flash ROM. 通过 EDK Ⅱ 编译生成的最终烧录在 Flash 的二进制文件称为固件镜像文件(后缀名为 FD)，它由若干个固件卷 firmware volume (FV)文件按顺序排列组成，固件卷是逻辑上的固件镜像文件. 固件卷可设置独自的文件系统，通过固件卷头结构体中 GUID 成员去识别. 固件卷以固件文件系统方式保存固件文件 firmware file (FF). 固件文件由固件文件头结构 firmware file header 和固件文件段 firmware file section (Section)组成，固件文件段是固件文件系统的最小存储单元[3]. 图 1 给出了固件卷、固件文件和固件文件段与固件文件系统的关系.

1.2 UEFI 固件文件类型

EDK Ⅱ 把项目依赖的源程序和库模块编译为后缀名为 efi 的 UEFI 固件文件，把 EFI 可执行文件包装在 EFI_SECTION_PE 文件段内，和其他相关的文件段组成固件文件. 多个固件文件再组成固件卷，由多个固件卷再形成固件镜像文件. 需要指出的是，根据 EDK Ⅱ 编译规范，固件文件系统需要保证固件文件段、固件卷数据对齐，所以需要在固件文件头结构、固件文件段头结构、固件卷头结构之间按所需进行 PAD 文件(数据为 0 的文件)填充，同时进行校验和检测[5].

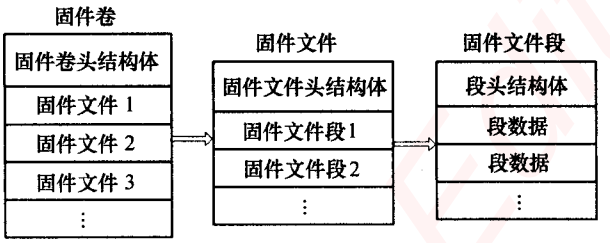

图 1 固件卷、固件文件和固件文件段的关系

Fig. 1 Relationship among firmware volume, firmware file, and its section

为了便于管理和使用更多类型的固件文件，UEFI 允许固件卷定义单独的文件系统. 组成固件文件的常用的固件文件类型如表 1 所示.

表 1 固件文件类型

Tab. 1 Firmware file type

文件类型	功能描述	GUID 值
EFI_FV_FILETYPE_RAW	原始二进制数据文件	0x01
EFI_FV_FILETYPE_FREEFORM	原始的段数据文件	0x02
EFI_FV_FILETYPE_SECURITY_CORE	SEC 阶段核心固件文件	0x03
EFI_FV_FILETYPE_PEI_CORE	PEI 阶段核心固件文件	0x04
EFI_FV_FILETYPE_DXE_CORE	DXE 阶段核心固件文件	0x05
EFI_FV_FILETYPE_PEIM	PEI 阶段驱动文件	0x06
EFI_FV_FILETYPE_DRIVER	DXE 阶段驱动文件	0x07
EFI_FV_FILETYPE_APPLICATION	应用程序文件	0x09
EFI_FV_FILETYPE_SMM	SMM 驱动文件	0x0A
EFI_FV_FILETYPE_FFS_PAD	PAD 类型文件	0xF0

本文用到的固件文件类型有 SEC 阶段核心固件文件、PAD 文件、PEI 阶段驱动文件等. SEC 阶段核心固件文件 SecCore 是开机后首先要执行的固件文件，PAD 文件用于填充固件文件系统中多余的空间，保证固件文件段、固件卷在固件镜像文件中的位置符合要求.

2 利用 EDK Ⅱ 开发 MIPS 架构计算机平台支持固件的问题分析

本节基于固件镜像文件的静态结构(flash lay-

RT Embedded http://www.kontronn.com

out)分析 EDK Ⅱ 直接编译生成的 UEFI 固件不能使用的原因.

静态结构是固件最终在 Flash 中的静态物理结构,它同平台的特性以及运行期间的调度流程相关,通过 EDK Ⅱ 在编译时按预先设定组织生成. MIPS 架构计算机平台设计的静态结构如图 2 所示.

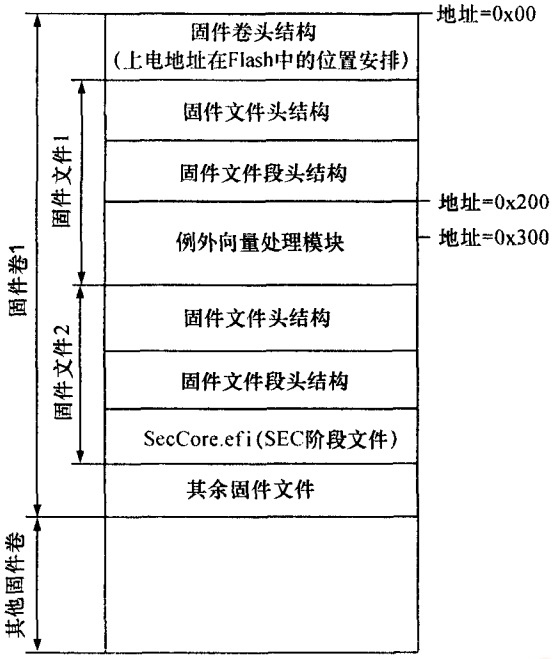

图 2 固件镜像文件静态结构

Fig. 2 Structure of firmware

设计固件镜像文件的静态结构需要考虑平台上电地址的复位向量 reset vector 映射在 Flash 的位置从而使得系统开机后能首先执行,CPU 的例外向量(exception vector)如何映射在 Flash 中的 0x200 或 0x380 固定位置处.

2.1 SecCore 模块位置

计算机平台刚上电时,内存和内存控制器都处于未初始化状态;CPU 不能从内存读取要执行的代码,EDK Ⅱ 编译生成的 BIOS 文件在初始阶段只能在非易失性介质中运行.

本文实验所用的龙芯 2F-6003 平台 Flash 芯片的物理地址是 0x1FC00000,虚拟地址为 0xBFC00000[2]. CPU 上电后从虚拟地址为 0xBFC00000 处读取指令执行. 而根据 EDK Ⅱ 编译规则,此处保存的是 UEFI 固件文件系统的结构信息. 所以系统开机后无法找到 SEC 阶段模块 SecCore.

2.2 例外向量位置

MIPS 架构处理器把异常和中断服务地址称为例外向量[9]. 龙芯 2F 处理器的冷重置、软重置和非屏蔽中断例外向量地址是专用的冷重置例外向量地址,这个地址既不通过 Cache 进行存取,也不能进行地址映射. 而其他所有例外向量地址的形式都可由基地址加上向量偏移地址计算得到[2].

表 2 例外向量地址

Tab. 2 Exceptional vector address

BEV 位	例外类型	例外向量地址
BEV = 0	冷重置/ 不可屏蔽中断	0xFFFFFFFF BFC00000
	TLB 重填 (EXL=0)	0xFFFFFFFF 80000000
	XTLB 重填 (EXL=0)	0xFFFFFFFF 80000000
BEV = 1	TLB 重填 (EXL=0)	0xFFFFFFFF BFC00200
	XTLB 重填 (EXL=0)	0xFFFFFFFF BFC00200
	其他	0xFFFFFFFF BFC00380

龙芯 2F 处理器通过设置状态寄存器的 BEV 位设定当前的异常处理模式. 平台刚上电时,状态寄存器中的 BEV 位设置为 1,即系统处于启动时模式,例外向量地址位于既不能通过 Cache 进行存取,也不能进行地址映射. 当 BEV 位设置为 0 时,进入正常操作模式(如运行在操作系统下),此时例外向量地址可通过 Cache 访问. 表 2 列出了龙芯 2F 处理器中的例外向量地址.

由表 2 可知,龙芯处理器的例外向量在 UEFI 固件镜像文件的位置要求严格固定,即位于 Flash 的 0x200 或 0x380 位置. MIPS 处理器在遇到中断、系统调用等异常处理时,会从 0x200 或 0x380 位置读取异常处理函数. 而 EDK Ⅱ 正常编译生成的固件镜像文件无法确保例外处理函数位于固定地址,从而使不能调用例外向量服务.

3 MIPS 架构计算机平台支持固件设计

MIPS 架构计算机平台支持固件设计包括 SEC 模块定位、例外向量位置设置及固件系统编译构建.

3.1 SEC 阶段 SecCore 模块定位

SEC 阶段 SecCore 模块定位的目标是使平台上电后可找到 SecCore 模块并被首先执行.

固件卷文件头结构的构成如下所示:

```
Typedef struct {
UINT8 ZeroVector[16]; //零向量
EFI_GUID FileSystemGuid; //文件系统GUID
UINT64 FvLength; //长度
UINT32 Signature; //签名
EFI_FVB_ATTRIBUTES Attributes;//属性
UINT16 HeaderLength; //头结构长度
UINT16 Checksum; //校验和
UINT16 ExtHeaderOffset;
```

RT Embedded http://www.kontronn.com

UINT8 Reserved[1]；//保留

UINT8 Revision；

EFI_FV_BLOCK_MAP_ENTRY BlockMap[]；

} EFI_FIRMWARE_VOLUME_HEADER

其中 ZeroVector 是为兼容非 X86 平台预留的 16 字节数组. 本文用该域存储一条跳转指令，直接跳转至 SecCore 模块的入口地址，从而使系统加电后可从 0XBFC00000 地址加载到 SecCore 模块.

龙芯 SecCore 引导定位需从固件镜像文件中确定 SecCore 模块的偏移地址 Offset，把偏移地址放为绝对跳转指令的目标地址，并把该指令写入固件卷头结构体零向量的前 4 个字节，且需保证整个固件卷的校验和正确.

龙芯 SecCore 引导定位首先从固件镜像文件读取固件卷 FdFile，在 FdFile 中定位至为 EFI_FV_FI-LETYPE_SECURITY_CORE 的固件文件位置. 在固件文件中搜索 SecCore 的 TeSection 位置，定位至 TeSection 后，计算出其距固件镜像文件起始地址的 Offset 值. 龙芯 2F 系列 CPU 的程序指令占 4 个字节，通过(Offset >> 2—1)计算出 SecCore 首地址和 SecCore 起始地址的距离，并把该距离值设为绝对跳转指令 bal 的目标地址，把指令写入固件镜像文件开始 4 个字节，最后在 15 和 16 字节填充计算的校验和，保证固件卷头结构体的总校验和正确. 算法流程如图 3 所示，本文设计了 RecalculateVector 工具实现图 3 算法.

图 4 是编译生成的固件二进制文件中零向量部分的内容显示. 直线标注 63 01 00 10 即为跳转指令 bal 0x163 的二进制显示，其中 0x163 是 SecCore 首条指令相对起始地址的偏移地址，第 15、16 字节的 9D EE 是计算所得的校验和.

3.2 例外向量位置设置

龙芯处理器的例外处理函数在固件镜像文件的位置要求严格固定，即位于固件镜像文件的 0x200 或 0x380 位置. MIPS 处理器在遇到中断、系统调用等异常处理时，会从 0x200 或 0x380 位置读取异常处理函数[9]. 传统的 MIPS 固件系统，把异常处理函数作为独立的模块编译至固件文件的合适地址空间，然后在两个例外入口地址处存放跳转至例外处理模块入口的指令. 由于 EDK Ⅱ 编译方式及 UEFI 固件文件不能使用上述方式. 本文使用 PAD 类型固件文件以填充方式解决龙芯 UEFI 固件例外向量位置的设置问题.

为了不改变 EDK Ⅱ 的核心架构，本文没有通过

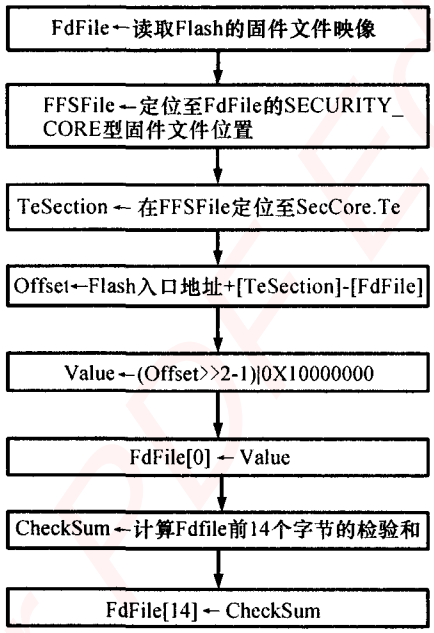

图 3 龙芯 SecCore 引导定位

Fig. 3 SecCore location of Loongson

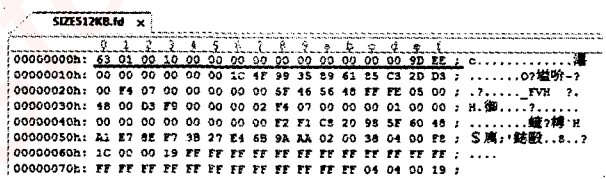

图 4 填充零向量后的固件卷头结构

Fig. 4 Firmware volume header filled with zerovector

修改 GenFv 工具(EDK Ⅱ 的标准工具，用于把函数模块组织到一起，封装成固件卷结构)的方法设置例外向量位置，而是采用修改例外处理函数的编译链接脚本 Ld. Script，把入口地址链接到例外处理向量所在的(0x200，0x380)位置，同时根据 EDK Ⅱ 编译规范，设计预编译模块 GenPad，用 PAD 类型固件文件填充在例外处理模块与固件卷头结构之间的空隙，使编译后例外处理函数位于固件镜像文件的 0x200 或 0x380 位置.

例外向量位置设置的主要流程为计算(0x200，0x380)位置的例外处理向量与固件文件系统的卷头结构、文件头和固件文件段头结构之间的空隙 ZeroPadding，如下式所示：

$$\text{ZeroPadding} = \text{Min}(0x200, 0x380) - \text{sizeof}(\text{固件卷头}) - \text{sizeof}(\text{固件文件头}) - \text{sizeof}(\text{固件文件段头}) \quad (1)$$

式中：sizeof 函数用于计算变量、结构体所需空间大小. 式(1)用(0x200、0x380)的最小地址减去固件卷头结构、固件文件头结构和固件文件段头结构所占

RT Embedded http://www.kontronn.com

向量位置设置如图 5 所示.

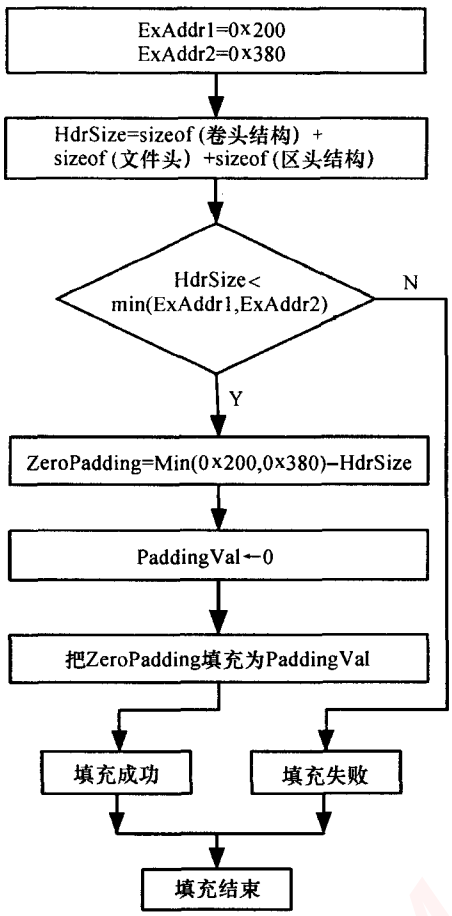

图 5　例外向量位置设置

Fig. 5　Location setting of exceptional vector

本节设计了 GenPad 模块以 PAD 文件填零方式使例外向量位于 0x200 或 0x380 地址，解决了例外向量的地址设置问题.

3.3　MIPS 架构计算机平台支持固件编译

本文设计了 RecalculateVector 工具使平台上电后可找到 SecCore 模块首先执行，及 GenPad 工具用于填充在例外处理函数和固件文件系统头结构之间的空隙，使编译后例外处理函数位于指定位置. 修改 EDK Ⅱ 编译脚本 BUILD. SH 使得两个工具在 EDK Ⅱ 的标准编译流程开始之前完成编译，在 Flash 的描述文件中加入对生成的异常处理函数模块文件的引用.

图 6 的编译流程中，首先编译生成 RecalculateVector 和 GenPad 模块. GenPad 计算固件文件系统和例外中断模块之间的空间并生成 PAD 二进制文件，然后编译处理例外向量中断模块. 在 PAD 文件、例外向量处理模块编译完成后，用 EDK Ⅱ 的编译工具把例外处理模块、PAD 二进制文件及例外向用 RecalculateVector 工具在编译的固件镜像文件中找出 EFI_FV_FILETYPE_SECURITY_CORE 型的固件文件，按照 SecCore 模块位置设置方法进行处理，形成最终的 BIOS 文件.

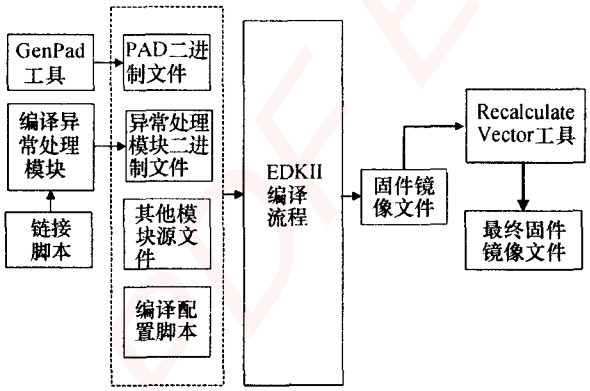

图 6　编译模块流程

Fig. 6　Flowchart of firmware compilling

4　MIPS 平台 UEFI 固件执行

MIPS 架构 UEFI 固件和传统 X86 固件涵盖的主要阶段相同，在操作系统启动之前包括安全保护阶段 SEC(security)、EFI 环境初始化阶段 PEI (Pre EFI initialization environment)、驱动环境执行阶段 DXE(driver execution environment)和引导设备选择阶段 BDS(boot device selection)共 4 个阶段，处理了系统从加电、处理器初始化、硬件初始化、启动路径、系统策略、用户界面配置，直至进入 OS 等过程[8].

(1) SEC 阶段. 由于龙芯 CPU 不支持在内存初始化之前把 CPU 的 Cache 缓存配置为 Cache As Ram，因而不能建立 C 语言运行环境的堆栈. 与 X86 平台相比，龙芯 UEFI 的 SEC 阶段执行流程主要是把 PEI 阶段的芯片初始化、平台初始化等功能由 SEC 阶段的 SecCore 模块完成.

SecCore 模块主要任务是 CPU 的基本初始化，Cache、TLB 和主板南北桥芯片初始化、内存初始化等，之后构建 C 语言代码的执行环境. SecCore 模块同时创建与平台相关的 HOB 信息表. SecCore 的工作流程如图 7 所示.

(2) DXE 阶段. DXE 阶段主要执行系统初始工作. DXE 环境包括 DXE 基本服务、DXE 分发器和 DXE 驱动程序. DXE 基本服务基于 DXE 架构协议服务和 PEI 阶段传来的 HOB 列表生成 EFI 启动服务、EFI 运行时服务和 DXE 服务的全部组件. DXE 分发器是 DXE 基本服务的组件，负责发现和按依

RT Embedded http://www.kontronn.com

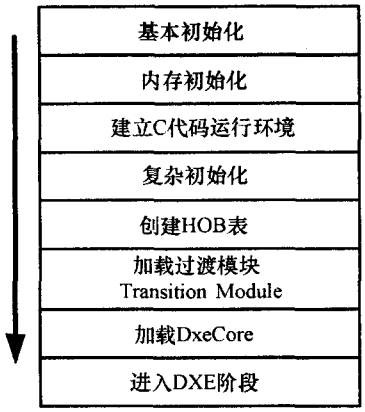

图 7　SecCore 工作流程

Fig. 7　SecCore flowchart

赖顺序执行 DXE 驱动以完成设备初始化.

DXE 阶段显卡的初始化通过执行显卡 BIOS 中的 X86 二进制代码来实现. 在 MIPS CPU 平台上,不能直接运行 X86 架构的代码,因此本文为 DXE 阶段移植了 X86 模拟器解释 X86 二进制代码,实现显卡初始化[8].

(3) BDS 阶段. BDS 阶段由 BdsDxe 模块产生,在 DXE 阶段被加载. BDS 阶段可进入图形界面进行系统管理或配置启动选项,在 BDS 的最后阶段 UEFI 固件有选择地把控制权交给操作系统.

经过 SEC、DXE、BDS 三个阶段后,操作系统被加载运行,进入 EFI 实时运行阶段. 在实时运行阶段,EFI 固件除了运行时服务(runtime service)和运行时驱动(runtime driver)驻留于内存,其他内存被操作系统收回. 通过以上流程,MIPS 架构的 UEFI 固件系统被正常加载引导,正常驱动了计算机系统.

5　结论

传统的 X86 平台支持固件已有较多研究,但 MIPS 架构的 UEFI 支持固件研究目前尚处于初始阶段. 龙芯 CPU 是新一代国产 CPU,属于 MIPS 架构处理器,其 UEFI 支持固件存在如何定位 SecCore 和例外处理函数如何加载等问题.

本文根据固件文件系统的组织和运行特点,通过在固件文件卷的零向量保存 SecCore 的起始映像地址使 SecCore 模块在开机时被加载执行,利用 PAD 文件填充固件文件卷首结构和例外向量地址空间的空隙使系统能正常调用例外向量. 所提方法在不改变 EDK Ⅱ 核心架构的情况下实现对 MIPS 架构处理器的支持.

本文开发了支持 MIPS 架构的计算机平台固件系统,并在龙芯 2F CPU 处理器上进行了实验和测试. 实验结果表明本文所提方法能够成生成支持 MIPS 架构的固件系统,并能运行于龙芯系列 CPU. 由于所提方法基于 UEFI 架构,具有通用、易维护、易扩展等性能,对龙芯产业的延伸、扩展具有一定意义.

致谢:本文得到南京大学软件新技术国家重点实验室开放基金(KFKT2008B15)和东南大学计算机网络和信息集成教育部重点实验室(K93-9-2010-04)的资助,在此表示感谢.

参考文献:

[1] HU Wei-wu, ZHANG Fu-xin, LI Zu-song. Micro-architecture of the Godson-2 processor [J]. Journal of Computer Science and Techology, 2005, 20(2): 243-249.

[2] 中国科学院计算技术研究所. 龙芯 2F 处理器数据手册 [Z]. 2006.

[3] Unified EFI Inc. Unified extensible firmware interface specification version2. 3 [EB/OL]. [2013-01-10]. https://www.tianocore. org/.

[4] 王晓箴,于　磊,刘宝旭. 基于 EDK2 平台的数据备份与恢复技术 [J]. 计算机工程,2011,37(15):262-264.

[5] 周　洁,谢智勇,余　涵,等. 基于 UEFI 的国产计算机平台的 BIOS 研究 [J]. 计算机工程,2011,37(增刊):355-358.

[6] 唐　笛. UEFI BIOS 电源管理研究和应用 [D]. 上海:上海交通大学,2011.

[7] 唐文彬,祝跃飞,陈嘉勇. 统一可扩展固件接口攻击方法研究 [J]. 计算机工程,2012,38(13):99-111.

[8] LO L T, WATSON G R, MINNICH R G. FreeVGA: Architecture independent video graphics initialization for LinuxBIOS [C]//Proceeding of USENIX Annual Technical Conference. Anaheim: [s. n.], 2005.

[9] SWEETMAN D, 赵俊良,张福新,等. MIPS 处理器设计视 [M]. 北京:北京航空航天大学出版社,2005.

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计
30. 总线协议中的 CRC 及其在 SATA 通信技术中的应用
31. 基于 FPGA 的 SATA 硬盘加解密控制器设计
32. Modbus 协议在串口通讯中的研究及应用

Creator Master PDF Editor

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植
15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通

Created in Master PDF Editor

12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解
23. Linux 桌面环境下内存去重技术的研究与实现
24. 掌握 Android 7.0 新增特性 Quick Settings

Windows CE：

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究
4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究

PowerPC：

1. Freescale MPC8536 开发板原理图

Created with Master PDF Editor

2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置
8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植
11. 基于 PowerPC 处理器 SMP 系统的 UBoot 移植

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植
15. CortexA8 平台的μC-OSⅡ及 LwIP 协议栈的移植与实现

Hardware：

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用

Created by Master PDF Editor

6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计
10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现
12. 基于 UEFI Shell 的 PreOS Application 的开发与研究
13. 基于 UEFI 固件的恶意代码防范技术研究

Created in Master PDF Editor