浙江大学
ZHEJIANG UNIVERSITY

嵌入式系统

第7课 Boot Loader程序设计

王总辉
zhwang@zju.edu.cn

http://course.zju.edu.cn

提纲

- 嵌入式系统启动流程
- Boot Loader基本概念
- Boot Loader典型结构
- Boot Loader典型案例

提纲

- 嵌入式系统启动流程
- Boot Loader基本概念
- Boot Loader典型结构
- Boot Loader典型案例

嵌入式系统启动流程

- 硬件加电
- 引导加载程序
  - Boot代码、Boot Loader等
- 操作系统内核，如Linux 内核
  - 根据特定的目标嵌入式硬件系统，定制的内核及启动参数
- 加载文件系统
  - 包括根文件系统以及建立于Flash内存设备上的文件系统
- 运行用户程序
  - 用户编写的完成特定功能的程序
  - 一些用户程序运行在一个嵌入式图形用户界面（GUI）上，常用的嵌入式GUI包括：Micro Windows 和MiniGUI等

Boot Loader

- 嵌入式系统中的OS启动加载程序
- 引导加载程序
  - 包括固化在固件(firmware)中的boot代码(可选)，和 Boot Loader两大部分
  - 是系统加电后运行的第一段软件代码
- 相对于操作系统内核来说，它是一个硬件抽象层

PC 机中引导加载程序（1）

- BIOS基本输入输出系统
- IBM兼容计算机中启动时调用的固件代码
- 基本功能
  - 为存储在其它介质中的软件程序做准备工作
  - 使它们能够正常地装载，执行并接管计算机的控制权
  - 这个过程被称为启动
- 两部分组成
  - BIOS(其本质就是一段固件程序)
  - 位于硬盘 MBR 中的 OS Boot Loader（如LILO 和 GRUB 等）
- BIOS ：硬件管理
  - 系统BIOS
    - 用来管理设备的基本子程序
    - 一般都装在主板的ROM中
  - 适配器上的BIOS
    - 硬盘、网卡、显卡等的BIOS都在自己的控制卡上
- OS Boot Loader：操作系统管理

PC 机中引导加载程序（2）

- 流程
  - 系统加电启动时，系统BIOS负责检测并启动加载控制卡上的BIOS
  - BIOS 在完成硬件检测和资源分配后，将硬盘 MBR 中的 Boot Loader 读到系统的 RAM 中，然后将控制权交给 OS Boot Loader
  - Boot Loader 的主要运行任务就是将内核映象从硬盘上读到 RAM 中，然后跳转到内核的入口点去运行，即开始启动操作系统。
- BIOS常驻系统内存的高端(C0000-FFFFF)
- BIOS扩充方法
  - 直接修改，然后固化
  - 软件方法，TSR技术
- BIOS目前逐步扩展到用固件实现的UEFI
  - UEFI，全称为Unified Extensible Firmware Interface，即“统一的可扩展固件接口”，是BIOS的一种升级替代方案，相当于一个微型操作系统，支持文件系统，运行应用程序。

嵌入式系统中引导加载程序

- 没有类似BIOS的固件程序
  - 有的嵌入式CPU也会内嵌一段短小的启动程序
- 系统的加载启动任务就完全由Boot Loader来完成
  - 如ARM7TDMI中，系统在上电或复位时从地址 0x00000000 处开始执行
  - 这个地址是Boot Loader程序

提纲

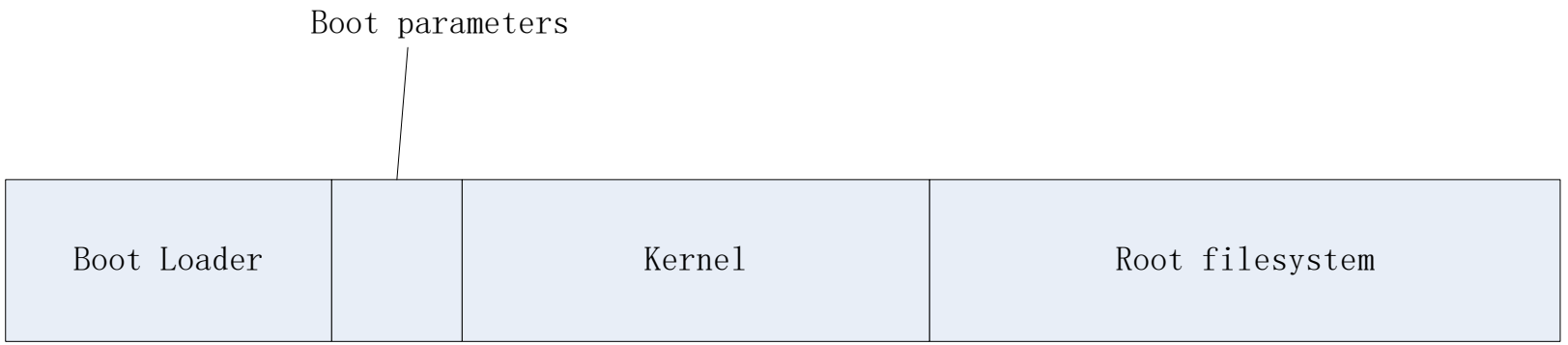
- 嵌入式系统启动流程
- Boot Loader基本概念
- Boot Loader典型结构
- Boot Loader典型案例

Boot Loader概念

- 在操作系统内核运行之前运行的一段小程序
- 功能
  - 初始化硬件设备
  - 建立内存空间的映射图
  - 调整系统的软硬件环境，以便操作系统内核启动
- 一般不通用
  - 依赖于处理器架构
    - CPU体系结构：ARM、MIPS、DSP、x86等
  - 依赖于具体的板级配置
    - 板级设备的配置：不同厂家的处理器芯片、不同的外设
  - 不同的CPU有不同的Boot Loader
  - 一些Boot Loader还支持多种CPU

Boot Loader存储

- 嵌入式系统没有BIOS，系统上电或复位后从0x00000000开始执行
- 嵌入式系统通常有固态存储设备(比如：ROM、EEPROM 或 FLASH 等)被映射到0地址上
- Boot Loader一般存储在0地址
- 系统加电后，CPU将首先执行 Boot Loader 程序



Boot Loader输入/输出

- 可用来控制Boot Loader的设备或机制
- 调试方法
  - 在Boot Loader阶段，显示设备不可用
  - 因此字符方式与用户进行交互
  - 主机和目标机之间一般通过串口建立连接
  - 通常需要与Host主机相连，Host作为TTY终端
  - 输出打印信息到串口，从串口读取用户控制命令

Boot Loader操作模式 (1/3)

- 启动加载模式(Boot Loading)
  - 自主（Autonomous）模式，是Boot Loader的正常工作模式
  - 流程
    - 从目标机上某个固态存储设备上将OS加载到RAM
    - 准备好内核运行所需的环境和参数
    - 在RAM运行操作系统内核

Boot Loader操作模式（2/3）

- 下载模式(Downloading)
  - 用户干预进入下载模式，在控制台打印提示信息，等待用户输入
    - 如用户不干预，则进入正常启动模式，即调用操作系统内核
  - 可通过串口连接或网络连接等通信手段从主机（Host）下载文件
    - 可以下载内核映像、根文件系统映像、Boot Loader自身
  - 通常在第一次安装内核与根文件系统时被使用
  - 系统更新也会使用 Boot Loader 的这种工作模式
  - 流程
    - 从主机下载的文件首先被 Boot Loader保存到目标机的 RAM 中
    - 被 Boot Loader写到目标机上的FLASH类固态存储设备中，或者直接在RAM中运行

Boot Loader操作模式（3/3）

- 通用boot Loader一般同时支持两种工作模式
  - 例如Blob或U-Boot
  - 允许用户在这两种工作模式之间进行切换
- Blob 在启动时处于正常的启动加载模式，但是它会延时10秒等待终端用户按下任意键而将blob切换到下载模式。如10秒内没有用户按键，则blob继续启动Linux内核

Boot Loader文件传输设备及协议

- 控制信息传输
  - 目标机Boot Loader 与主机之间：串口
  - 传输协议：xmodem / ymodem / zmodem 协议
  - 简单、通用
- 大型文件，如
  - 串口传输的
  - 以太网，TF
    - 主机提供
    - 通过以太
    - 通用，易
  - USB
    - 简单、易

1、xmodem是最早的协议之一，一种由几乎所有通讯程序支持的文件传送协议，每次传送128个字节信息块；
ymodem和zmodem都是它的改进协议，
2、ymodem传送1024字节长的信息块，快于xmodem并且可送多个文件；
3、zmodem速度快于ymodem和xmodem，且可以更好地在断开后恢复传输。

提纲

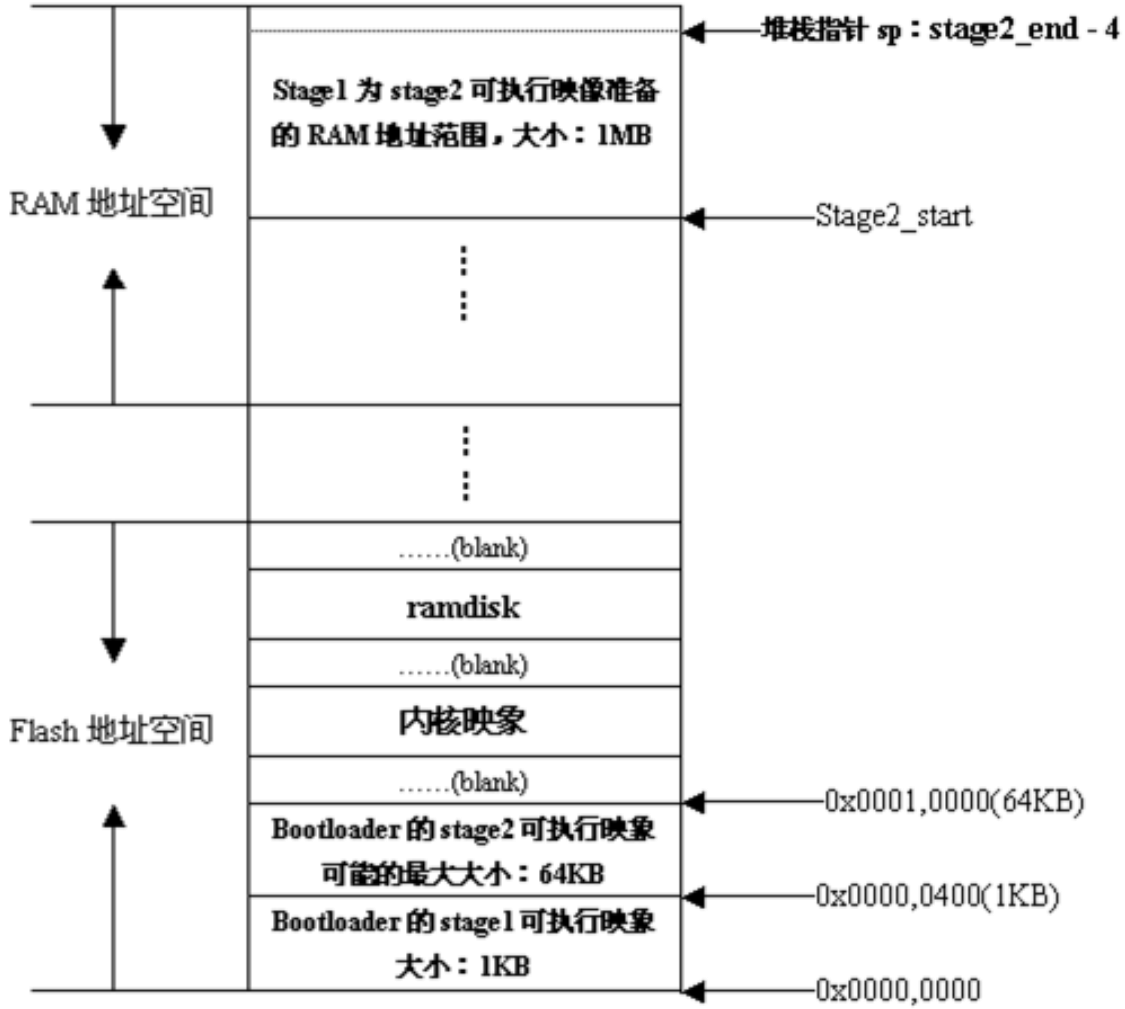
- 嵌入式系统启动流程
- Boot Loader基本概念
- Boot Loader典型结构
- Boot Loader典型案例

Boot Loader 基本架构

- Boot Loader 的启动过程是单阶段（Single Stage）还是多阶段（Multi-Stage）
- 多阶段的 Boot Loader
  - 提供更为复杂的功能，以及更好的可移植性
- Boot Loader的生命周期
  - 初始化硬件,如设置UART(至少设置一个),检测存储器等
  - 设置启动参数,告诉内核硬件的信息,如用哪个启动界面,波特率.
  - 跳转到操作系统的首地址.
  - 消亡
- 从固态存储设备上启动的 Boot Loader 大多都是 2 阶段的启动过程
  - 启动过程可以分为 stage 1和 stage 2 两部分

两阶段Boot Loader

- stage 1
  - 汇编
  - 简单的硬件初始化
- stage 2
  - C语言
  - 复制数据
  - 设置启动参数
  - 串口通信等功能



Boot Loader主要任务

- 假定
  - 内核映像与根文件系统映像都被加载到 RAM 中运行
  - 另外选择XIP
    - 直接在 ROM 或 Flash 这样的固态存储设备中直接运行
- stage1 通常包括以下步骤
  - 硬件设备初始化
  - 为加载 Boot Loader 的 stage2 准备 RAM 空间
  - 拷贝 Boot Loader 的 stage2 到 RAM 空间中
  - 设置好堆栈
  - 跳转到 stage2 的 C 入口点
- Boot Loader 的 stage2 通常包括以下步骤
  - 初始化本阶段要使用到的硬件设备
  - 检测系统内存映射(memory map)
  - 将 kernel 映像和根文件系统映像从 flash 上读到 RAM 空间中
  - 为内核设置启动参数
  - 调用内核

stage1—基本的硬件初始化

- 目的
  - 为 stage2 的执行以及随后的 kernel 的执行准备好一些基本的硬件环境
- （1）屏蔽所有的中断
  - 为中断提供服务通常是 OS 设备驱动程序的责任，Boot Loader 的执行全过程中可以不必响应任何中断
  - 中断屏蔽可以通过写 CPU 的中断屏蔽寄存器或状态寄存器（如 ARM 的 CPSR 寄存器）来完成
- （2） 设置 CPU 的速度和时钟频率。
- （3）RAM 初始化
  - 包括正确地设置系统的内存控制器的功能寄存器以及各内存库控制寄存器等。
- （4）初始化 LED
  - 通过 GPIO 来驱动 LED，其目的是表明系统的状态是 OK 还是 Error
  - 如板子上没有LED，那么也可以通过初始化 UART 向串口打印 Boot Loader 的 Logo 字符信息
- （5）关闭 CPU 内部指令 / 数据 cache

为加载 stage2 准备 RAM 空间

- 通常把 stage2 加载到 RAM 空间中来执行
- stage2 通常是 C 语言执行代码，考虑堆栈空间
- 空间大小最好是 memory page 大小(通常是 4KB)的倍数
- 一般1M RAM 空间已经足够，地址范围可以任意安排
  - 如blob就将stage2可执行映像从系统 RAM 起始地址0xc0200000开始的1M空间内执行
  - 将stage2安排到RAM空间的最顶1MB也是一种值得推荐的方法。
  - stage2_end＝stage2_start＋stage2_size
- 对所安排的地址范围进行测试
  - 必须确保所安排的地址范围可读写的 RAM 空间
  - 测试方法可以采用类似于 blob 的方法
    - 以 memory page 为被测试单位，测试每个 page 开始的两个字是否是可读写的

拷贝 stage2 到 RAM 中

- 拷贝时要确定两点
  - （1） stage2 的可执行映象在固态存储设备的存放起始地址和终止地址
  - （2） RAM 空间的起始地址。

设置堆栈指针 SP

- 通常把 SP 的值设置为(stage2_end-4)
  - 1MB 的 RAM 空间的最顶端(堆栈向下生长)
- 在设置堆栈指针 SP 之前，也可以关闭 LED 灯，以提示用户准备跳转到 stage2

跳转到 stage2 的 C 入口点

- 可以跳转到 Boot Loader的stage2去执行
- 如在 ARM系统中，这可以通过修改 PC 寄存器为合适的地址来实现

Stage2

- stage2 的代码通常用 C 语言来实现
- 代码可读性和可移植性
- 不能使用 glibc 库中的任何支持函数
  - Why?
- trampoline(弹簧床)编程方式
  - 用汇编语言写一段trampoline 小程序，并将这段 trampoline 小程序来作为 stage2 可执行映象的执行入口点
  - 在trampoline汇编小程序中用CPU跳转指令跳入main()函数中去执行
  - 当main()函数返回时，CPU 行路径显然再次回到trampoline程序
  - 用 trampoline小程序来作为main()函数的外部包裹(external wrapper)
- Why not use main
  - （1）无法传递函数参数
  - （2）无法处理函数返回值

blob trampoline 程序示例

```
.text
.globl _trampoline
_trampoline:
bl main
/* if main ever returns we just call it again */
b _trampoline
```

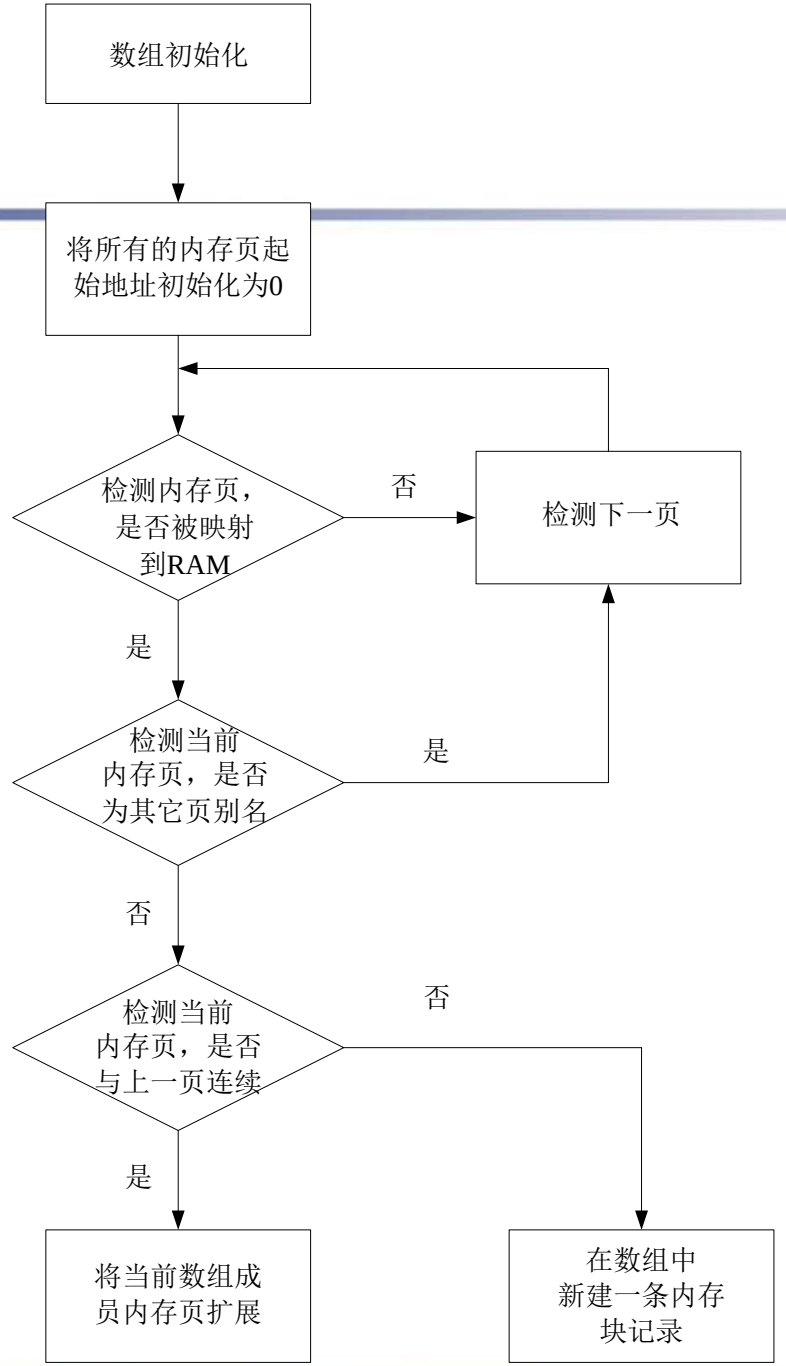
Stage2—初始化本阶段要使用的硬件设备

- （1）初始化至少一个串口，以便终端用户进行 I/O 输出信息
  - 在初始化这些设备之前，也可以重新把LED灯点亮，以表明我们已经进入main()函数执行
  - 设备初始化完成后，可以输出一些打印信息，程序名字字符串、版本号等
- （2）初始化计时器等
- （3）使用Volatile
  - 避免编译器优化
    - 编译器为了提高程序运行速度，可能对变量进行优化，会将其放在缓存中
  - 缺陷是变量的内容可能被误改，从而与内存中实际值不一致
  - 采用volatile 关键字可以避免上述问题

检测系统的内存映射（memory map）

- 在 4GB 物理地址空间中哪些地址范围被分配用来寻址系统的RAM 单元
  - 如SA-1100 中，从 0xC0000000 开始的 512M空间被用作系统的 RAM 空间
  - 在Samsung S3C44B0X 中，从0x0c00,0000 到 0x1000,0000 之间的 64M 地址空间被用作系统的 RAM 地址空间
- 嵌入式系统往往只把 CPU 预留的全部 RAM 地址空间中的一部分映射到 RAM 单元上，而让剩下的那部分预留 RAM 地址空间处于未使用状态
- Boot Loader的stage2必须检测整个系统的内存映射情况
  - 必须知道 CPU 预留的全部 RAM 地址空间中的哪些被真正映射到 RAM 地址单元，哪些是处于 "unused" 状态的

检测流程


数组初始化
将所有的内存页起始地址初始化为0
检测内存页，是否被映射到RAM
否
检测下一页
是
检测当前内存页，是否为其它页别名
是
否
检测当前内存页，是否与上一页连续
否
是
将当前数组成员内存页扩展
在数组中新建一条内存块记录

加载内核映像和根文件系统映像

- 规划内存占用的布局
  - 内核映像所占用的内存范围
  - 根文件系统所占用的内存范围
- 从 Flash 上拷贝内核映像到RAM上

设置内核的启动参数

- Linux 2.4.x 以后的内核都期望以标记列表(tagged list)的形式来传递启动参数
- 启动参数标记列表以标记 ATAG_CORE 开始，以标记 ATAG_NONE 结束
- 每个标记由标识被传递参数的 tag_header 结构以及随后的参数值数据结构来组成
- 在嵌入式 Linux 系统中，通常需要由 Boot Loader 设置的常见启动参数有：ATAG_CORE、ATAG_MEM、ATAG_CMDLINE、ATAG_RAMDISK、ATAG_INITRD等

调用内核

- 直接跳转到内核的第一条指令处
- 在跳转时，下列条件要满足
  - （1）CPU 寄存器的设置
    - R0＝0；@R1＝机器类型 ID；@R2＝启动参数标记列表在 RAM 中起始基地址
  - （2）CPU 模式
    - 必须禁止中断（IRQs和FIQs）；
    - CPU 必须 SVC 模式；
  - （3）Cache 和 MMU 的设置
    - MMU 必须关闭；
    - 指令 Cache 可以打开也可以关闭；
    - 数据 Cache 必须关闭

串口终端的调试作用

- 调试手段：打印信息到串口终端
- 串口终端显示乱码或根本没有显示
  - boot loader 对串口的初始化设置不正确
  - 运行在 host 端的终端仿真程序对串口的设置不正确，这包括：波特率、奇偶校验、数据位和停止位等方面的设置
- Bootloader的运行过程中可以正确向串口输出信息，但bootloader 启动内核后无法看到内核启动输出信息
  - 确认是否在编译内核时配置了对串口的支持
  - 确认bootloader对串口的设置与内核对串口的设置一样
  - 确认bootloader所用的内核基地址与内核映像在编译时所用的运行基地址一致

提纲

- 嵌入式系统启动流程
- Boot Loader基本概念
- Boot Loader典型结构
- Boot Loader典型案例

Boot Loader基本设计

- WinCE专用Boot Loader
- Boot Strap
- U-Boot
  - U-Boot简介
  - U-Boot特点
  - U-Boot的代码结构
- vivi简介
- Redboot简介

WinCE专用Boot Loader

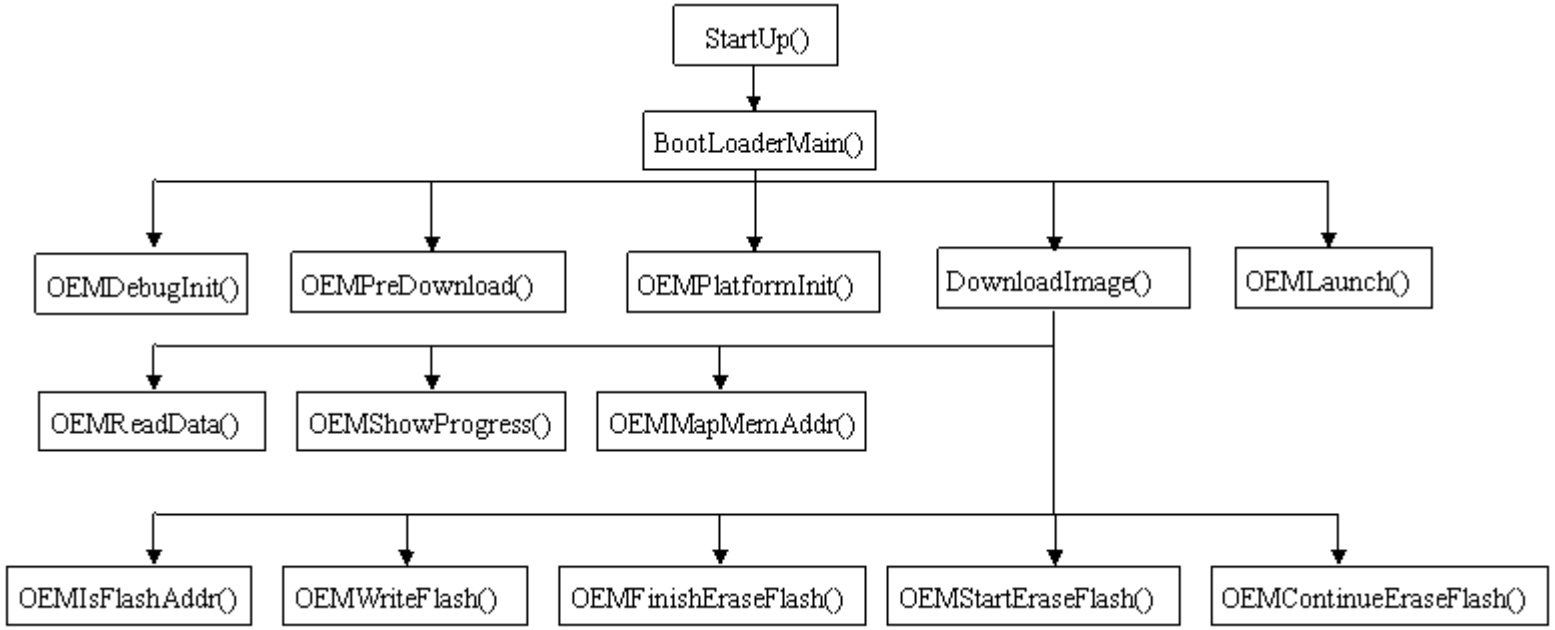
- 2009年2月16日Windows Mobile 6.5
  - 内核windows CE 5.x
  - 以后不用Windows Mobile ，用Windows phones
- X86的ROM Boot Loader
  - 又叫Rom Boot
  - 存放在Flash/EEPROM中，也就是原来BIOS的位置
  - 上电后CPU到固定地址执行代码，也就是执行了Rom Boot包含的代码
  - 对整个硬件系统进行初始化和检测
  - 支持通过网卡从远程机器上下载nk.bin或者从本地IDE/ATA 硬盘的活动分区中寻找nk.bin文件加载
  - 优点：引导并且加载速度快，而且它自身完成了所有的操作，这样就不用BIOS、MSDOS，更不用Loadcepc了
  - 缺点：需要CE开发者读懂它的源码并修改
  - CE提供了Rom Boot的所有源码

X86的BIOS Boot Loader

- BIOS Boot Loader只是不需要MSDOS操作系统，它仍然需要BIOS和FAT文件系统
- 系统上电后BIOS执行完硬件初始化和配置后，BIOS检查引导设备的启动顺序，如果引导设备是硬盘、CF卡、DOC（Disk-On-Chip）一类的存储设备，那么就加载这些存储器上的主引导扇区（Master Boot Sector）中的实模式代码到内存，然后执行这些代码
- 代码被称为主引导记录（MBR）
- MBR首先在分区表（同样位于主引导扇区）中寻找活动分区，如果存在活动分区，那么加载位于这个活动分区的第一个扇区上的代码到内存，然后执行这些代码。这里提到的活动分区的第一个扇区被称为引导扇区（Boot Sector）。
- 引导扇区上的代码的功能是找到并且加载BIOS Boot Loader，BIOS Boot Loader再加载nk.bin。
- 对于BIOS Boot Loader，CE提供了Setupdisk.144和Bootdisk.144两个文件

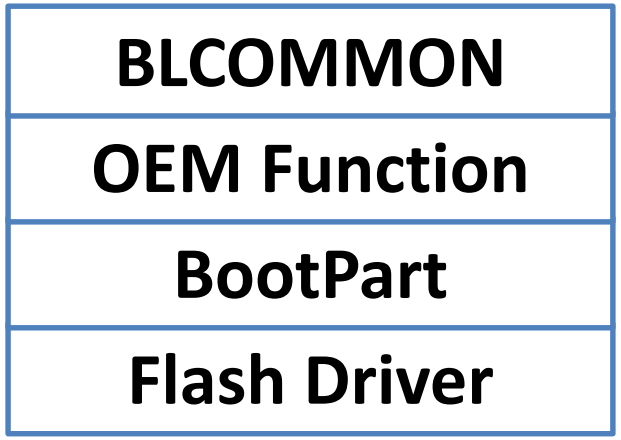
MSDOS＋Loadcepc

- 非常简单
- BIOS Boot Loader和MSDOS＋Loadcepc两种方式差不多
- 在MSDOS启动后再执行loadcepc.exe，让loadcepc加载nk.bin到内存后再把CPU控制权交给CE内核程序

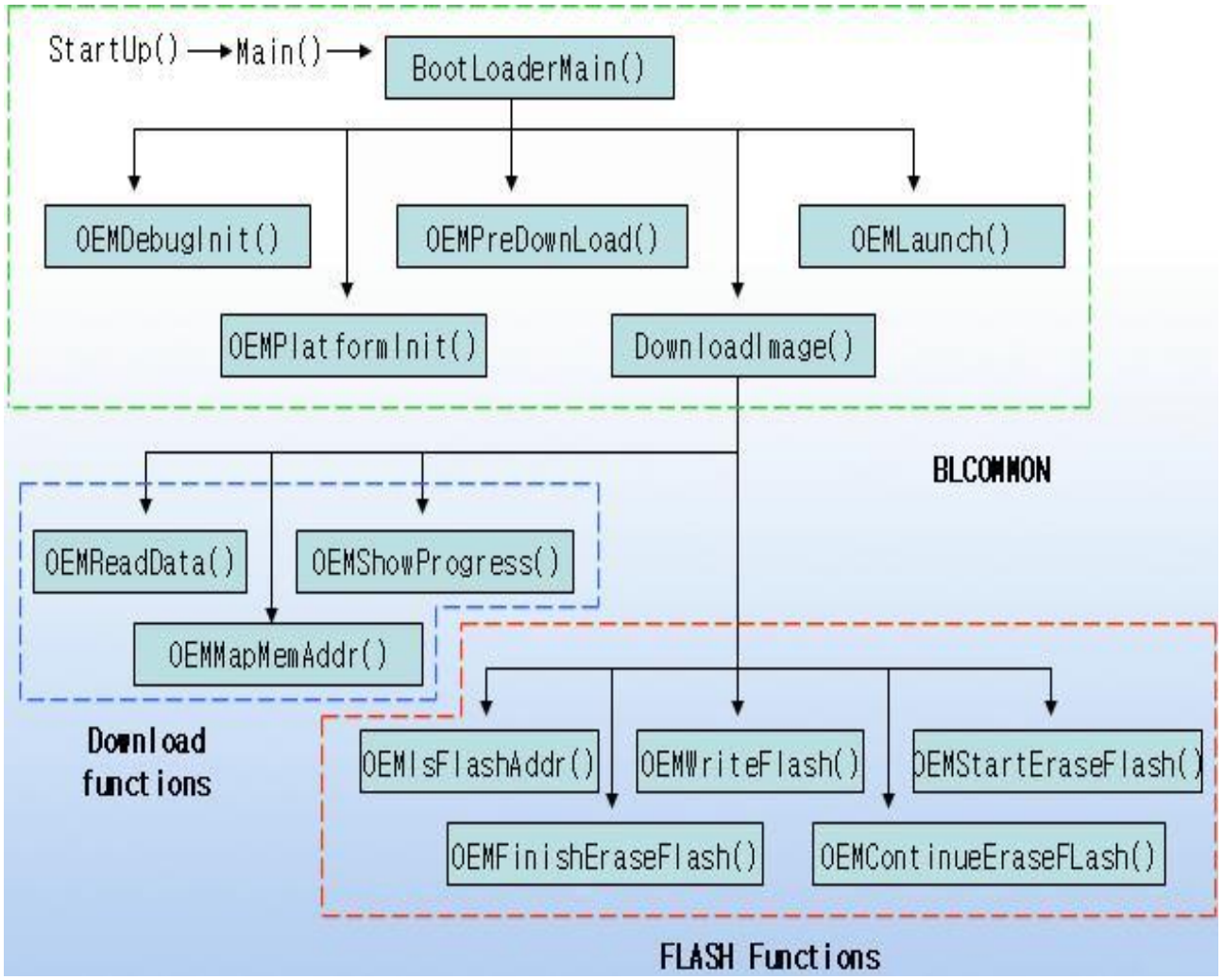


非X86的EBOOT

- Ethernet Boot loader，基于网络的Bootloader
- Platform Builder集成开发环境提供了对Eboot开发的支持库
- 可以加载WinCE image，还可以通过网络配合Platform Builder下载WinCE image进行调试
- 组成
  - BLCOMMON、OEM 代码、Eboot、存储管理和网络驱动



- Eboot流程
- 初始化MCU
  - 包括初始化MCU的相关寄存器、中断、看门狗、系统时钟、内存和MMU
- 调用BootloaderMain
  - 依次调用以下几个函数，OEMDebugInit、OEMPlatformInit、OEMPreDownload、OEMLaunch
  - 这些函数必须由EBOOT的代码来实现。
- 最终跳转到OAL.exe的StartUp处
  - 启动WinCE操作系统



Boot Strap

- 一段固化在处理器内部ROM之中的一段代码
  - 用户可以通过片外的管脚配置是否启动Boot Strap
  - 如果用户选择启动Boot Strap
    - 在系统上电复位时，固化在片上ROM中的代码将获得控制权
    - 这段代码将初始化片内的一些硬件设备，比如外部存储器接口控制器，通信接口等等
    - 完成这些初始化后，Boot Strap将通过这个通信接口等待调试主机发送来的命令，这些命令包括从调试主机下载一段映象到外部存储器的某个特定地址，或者是将外部存储器的内容通过通信接口上传到调试主机，从外部存储器的某个地址开始运行等等
- 某些厂商推出的嵌入式微处理器中（如飞思卡尔公司推出的龙珠328系列、新龙珠系列）提供了Boot Strap功能

- Boot Strap模式
  - Boot Loader在Boot strap模式下被加载到Flash上
- Bootstrap模式
  - 容许开发者通过UART对系统进行初始化
  - 容许开发者通过UART将程序下载到系统RAM中
  - Boot strap可以接收命令，并且运行事先储存在系统内存上的程序
  - Boot strap支持对内存和寄存器的读写操作

Boot Strap 操作流程

- 将目标板上的Boot Strap pin 打开
- 上电后，Boot Strap模式将UART1和UART2设置为自动波特率模式
- 同时将UART1和UART2设为8bit长、无校验位、1个停止位
- 等待用户输入a或A，一旦接收到上述字符，则可以设定波特率并返回冒号
- 通过Boot Strap record语句将CPU芯片内部寄存器，初始化为目标寄存器

U-Boot简介

- 德国DENX软件工程中心的Wolfgang Denk
- 全称Universal Boot Loader
  - 遵循GPL条款的开放源码项目
  - http://sourceforge.net/projects/U-Boot
  - 从FADSROM、8xxROM 、PPCBOOT逐步发展演化而来
  - 其源码目录、编译形式与Linux内核很相似
  - 源码就是相应Linux内核源程序的简化，尤其是设备驱动程序
- 支持
  - 嵌入式Linux/NetBSD/VxWorks/QNX/RTEMS/ARTOS/LynxOS
  - PowerPC、MIPS、x86、ARM 、Nios、XScale等处理器
- 对PowerPC系列处理器/对Linux的支持最完善

U-Boot特点（1）

- 支持SCC/FEC以太网、OOTP/TFTP引导、IP和MAC的预置功能
- 支持在线读写Flash、DOC、IDE、USB、EEPROM、RTC
- 支持串口下载S-Record格式文件和二进制格式文件，并把ELF32格式文件转换为S-Record格式文件，并执行
- 识别二进制格式的Linux内核映像文件，对Linux引导有特别的支持

1、Kermit包大小是随数据的整体结构和线路质量的变化而改变（1k-128）
2、当线路出现瞬时中断时，Kermit能迅速恢复数据同步传输
3、 Kermit支持重复字符的压缩处理

嵌入式系统可执行文件格式（1）

ELF文件格式

Executable and linking format(ELF)文件是Linux系统下的一种常用、可移植目标文件(object file)格式。有可重定位文件、可执行文件和共享目标文件三种形式。

S-record文件格式

S-Record文件遵循Motorola制定的格式规范，是一种标准的、可打印格式的文件，适用于在计算机平台间传送。由多条记录组成的，每条记录是5个字段组成的ASCII字符串。

嵌入式系统可执行文件格式（2）

bin文件格式

bin文件是二进制文件，内部没有地址标记。用编程器烧写时，从00开始；下载运行则下载到编译时的地址。

HEX文件格式

Intel HEX文件是记录文本行的ASCII文本文件，每一行是一个HEX记录，每条记录有五个域，由十六进制数组成的机器码或者数据常量；经常被用于将程序或数据传输存储到ROM、EPROM。大多数编程器和模拟器使用HEX文件。

U-Boot特点（2）

- 单任务软件运行环境，U-Boot可以动态加载和运行独立的应用程序
- 监控命令集：读写I/O、内存、寄存器、内存、外设测试功能等
- 脚本语言支持（类似bash脚本）
- 支持WatchDog、LCD logo和状态指示功能等
- 支持MTD和文件系统
- 支持中断
- 详细的开发文档

代码结构分析（1）

- Board：和一些已有开发板相关的文件，比如Makefile和u-boot.lds等都和具体硬件有关。
- common：与体系结构无关的文件，实现各种命令的C文件。
- cpu：CPU相关文件，其子目录都是以所支持的CPU为名，比如arm926ejs、mips、mpc8260和nios等，每个子目录中都包括cpu.c和interrupt.c，start.S。
- disk：disk驱动的分区处理代码。
- doc：文档。
- drivers：通用设备驱动程序。

代码结构分析（2）

- dtt：数字温度测量器或传感器的驱动。
- examples：一些独立运行的应用程序例子。
- fs:支持文件系统的文件，U-Boot现在支持cramfs、fat、fdos、jffs2和registerfs。
- include：头文件，还有对各种硬件平台支持的汇编文件，系统的配置文件和对文件系统支持的文件。
- net：与网络有关的代码，BOOTP协议、TFTP协议、RARP协议和NFS文件系统的实现。
- lib_xxx：与处理器体系结构相关的文件，如lib_mips目录与MIPS体系结构相关，lib_arm目录与ARM相关。
- tools：创建S-Record格式文件 和U-Bootimages的工具。

移植方法

- 先用硬件调试器 BDI2000创建目标板初始运行环境，将U-Boot镜像文件U-Boot.bin下载到目标板RAM中的指定位置，然后，用BDI2000进行跟踪调试
  - 好处:不用将U-Boot镜像文件烧写到Flash中去
  - 弊端:对移植开发人员的移植调试技能要求较高，BDI2000的配置文件较为复杂
- 用BDI2000先将U-Boot镜像文件烧写到Flash中去，然后用GDB和BDI2000调试
  - 所用的BDI2000配置文件较为简单，调试过程与U-Boot移植后运行过程相吻合
  - U-Boot先从Flash中运行，再重载至RAM 中相应位置，并从那里正式投入运行
  - 需要不断烧写Flash

vivi简介

- vivi来自韩国，由mizi公司开发维护，但是现在目前停止开发。
- vivi适用于arm9处理器，它是三星官方板SMDK2410采用的Boot Loader。通过修改之后可以支持S3C2440等处理器。
- vivi支持两种工作模式：启动加载模式和下载模式。

vivi特点

- vivi最主要的特点就是代码小巧，有利于移植新的处理器
- vivi的软件架构和配置方法类似Linux风格，对于有过编译Linux内核经验的读者，vivi更容易上手。
- 支持网卡、USB接口，LCD驱动，MTD，支持yaffs文件系统固化等功能。

vivi代码结构分析（1）

- arch：包括了所有vivi支持的目标板的子目录，官方原版只包括s3c2410目录，修改后可包括s3c2440目录等等。
- drivers：其中包括了引导内核需要的设备的驱动程序，主要是mtd和serial两个目录，分别是MTD设备驱动和串口驱动。MTD目录下分map、nand和nor三个目录。
- init：这个目录只有main.c和version.c两个文件。和普通的C程序一样，vivi将从main函数开始执行。

vivi代码结构分析（2）

- lib：平台公共接口代码
- include：头文件的公共目录。比如s3c24xx.h定义了这块处理器的一些寄存器。Platform/ smdk24xx.h定义了与开发板相关的资源配置参数，只需要修改这个文件就可以配置目标板的参数，如波特率、引导参数、物理内存映射等。
- vivi分两阶段启动，第一阶段代码采用汇编，位于arch/s3c2410/head.S，第二阶段代码用C语言编写，位于init/main.c。

Redboot简介

- Redhat公司随eCos发布的一个BOOT方案
- 开源项目
- 支持
  - ARM，MIPS，MN10300，PowerPC， Renesas SHx，v850，x86
- 使用X-modem或Y-modem协议经由串口下载，也可以经由以太网口通过BOOTP/DHCP服务获得IP参数，使用TFTP方式下载程序映像文件，常用于调试支持和系统初始化（Flash下载更新和网络启动）
- Redboot可以通过串口和以太网口与GDB进行通信，调试应用程序，甚至能中断被GDB运行的应用程序
- Redboot为管理FLASH映像，映像下载，Redboot配置以及其他如串口、以太网口提供了一个交互式命令行接口，自动启动后，REDBOOT用来从TFTP服务器或者从Flash下载映像文件加载系统的引导脚本文件保存在Flash上

□谢　谢！