嵌入式系统
An Introduction to Embedded System

第四课 嵌入式系统的BootLoader技术

教师：蔡铭 cm@zju.edu.cn

浙江大学计算机学院人工智能研究所
航天科技—浙江大学基础软件研发中心

课程大纲

- Bootloader程序的基本概念
- Bootloader典型框架结构
- S3C2410 Bootloader代码分析

Boot Loader程序基本概念

- Boot Loader是在系统启动时激活，在操作系统内核运行之前运行的一段程序
  - 初始化硬件设备和建立内存空间的映射图
  - 将系统的软硬件环境带到一个合适的状态，以便为最终调用操作系统内核准备好正确的环境

PC机的Boot Loader——BIOS（1/2）

- BIOS——Basic Input Output System

PC机的Boot Loader——BIOS（2/2）

- BIOS——Basic Input Output System

ROM

PROM

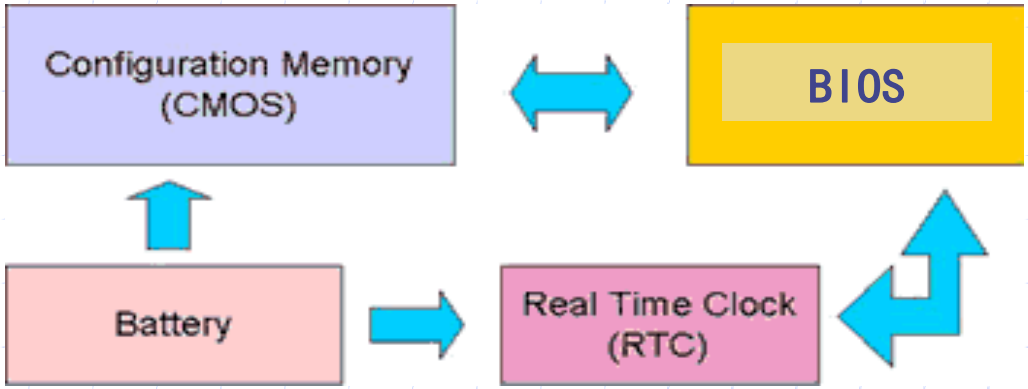
EPROM

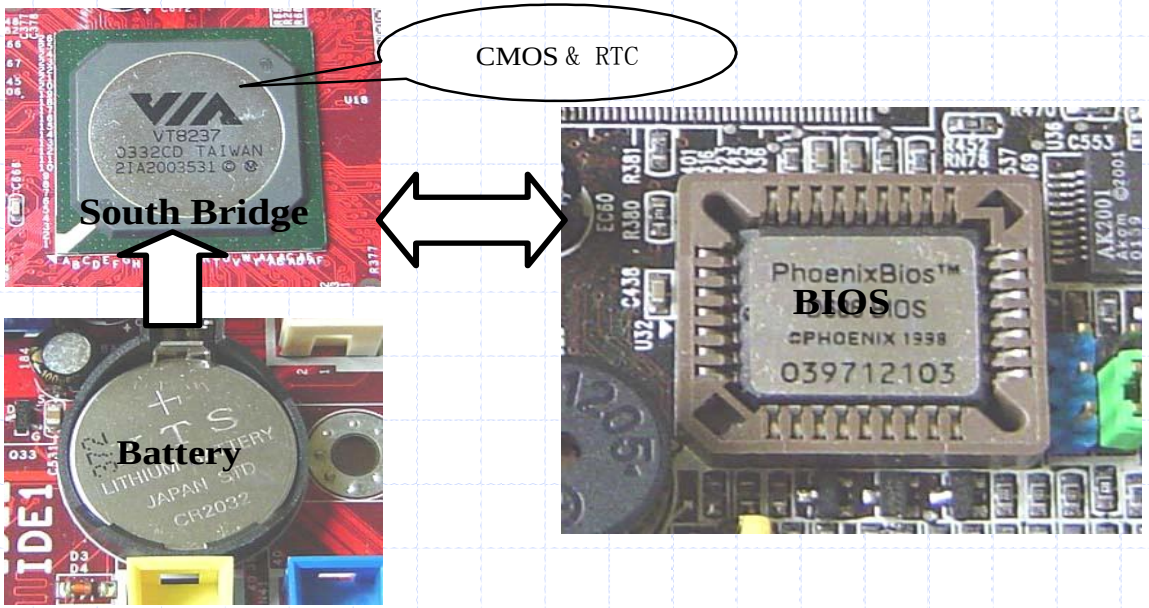
EEPROM

BIOS与CMOS的关系（1/2）

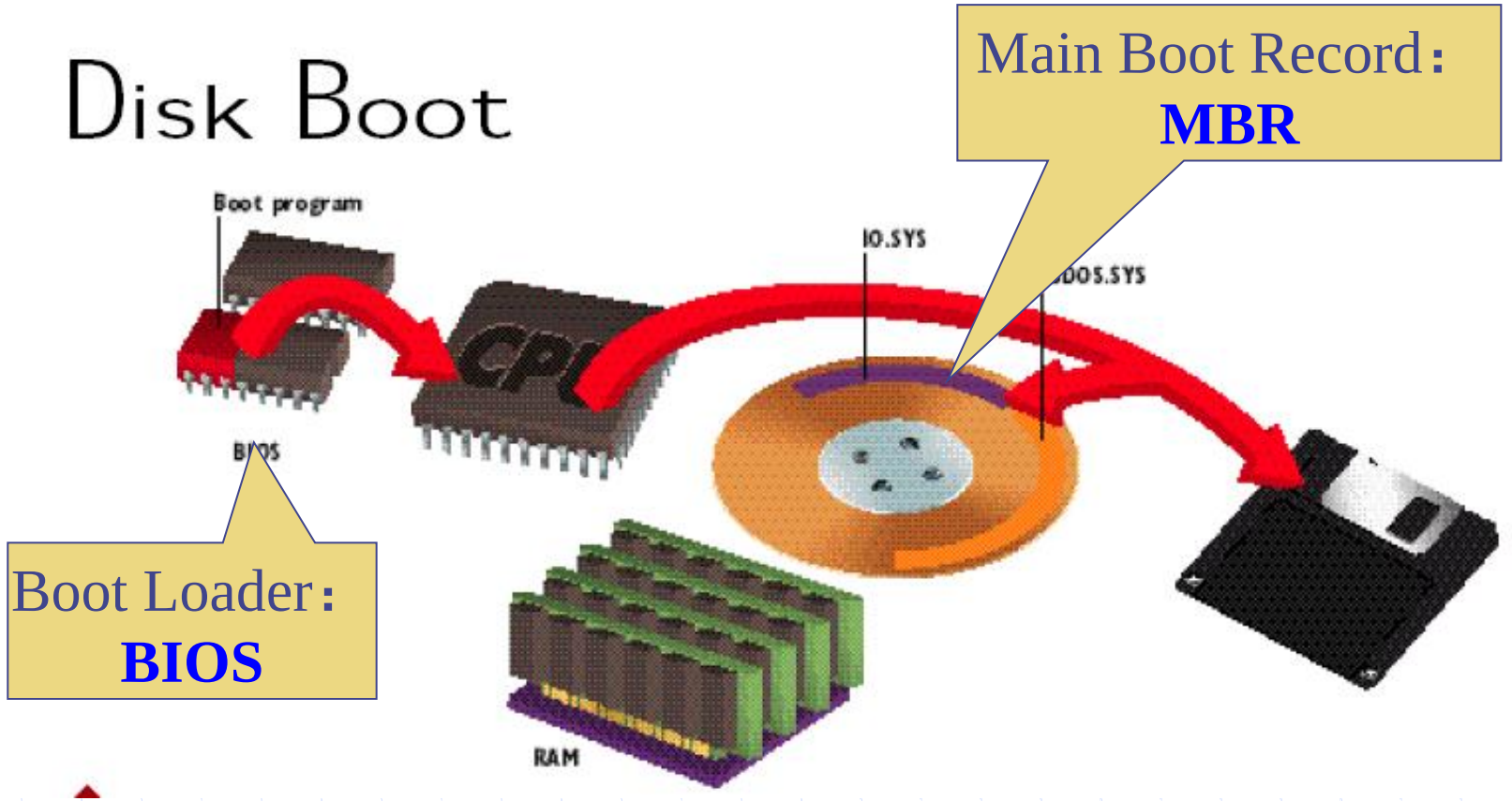
- BIOS：是一个用于系统自检、初始化、中断处理及系统参数设置的程序。
- CMOS ：是主机板上一块特殊的RAM芯片，用于存储系统参数。
- 通过BIOS设置程序对CMOS参数进行设置。

BIOS与CMOS的关系（2/2）





PC机的引导装载简介（1/2）



BIOS+MBR启动方式

PC机的引导装载简介（2/2）

PC机是通过BIOS来启动机器:

1、当PC机加电之后，BIOS启动相应的程序完成机器自检POST（Power-On Self Test）；

2、寻找可以引导的驱动器，即启动盘IPL（Initial Program Load Device）；

3、若找到合法的引导扇区（以0xAA55结束），那么就会将引导扇区的内容（共512字节）装载到内存0x0000:7C00处；

4、BIOS把控制权限交给系统引导扇区的程序。

从PC机启动扇区程序

- MBR示例—《自己动手写操作系统》

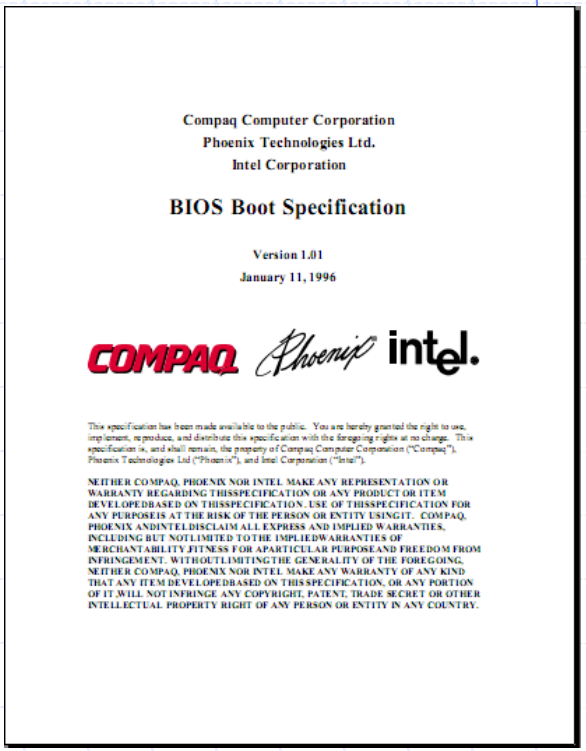
```
1     org 07c00h        ; 告诉编译器程序加载到7c00处
2     mov ax, cs
3     mov ds, ax
4     mov es, ax
5     call  DispStr     ; 调用显示字符串例程
6     jmp $       ; 无限循环
7   DispStr:
8     mov ax, BootMessage
9     mov bp, ax        ; ES:BP = 串地址
10    mov cx, 16        ; CX = 串长度
11    mov ax, 01301h    ; AH = 13,  AL = 01h
12    mov bx, 000ch     ; 页号为0(BH = 0) 黑底红字(BL = 0Ch,高亮)
13    mov dl, 0
14    int 10h       ; 10h 号中断
15    ret
16  BootMessage:    db  "Hello, OS world!"
17  times   510-($-$$)  db  0 ; 填充剩下的空间，使生成
18  dw  0xaa55        ; 结束标志
```

BIOS中断服务

- 10H 屏幕显示
- 11H 设备检验
- 12H 测定存储器容量
- 13H 磁盘I/O
- 14H 串行通讯口I/O
- 15H 盒式磁带I/O
- 16H 键盘输入
- 17H 打印机输出
- 18H BASIC入口代码
- 19H 引导装入程序

BIOS Boot Specification

- **BIOS Boot Specification**
  - ✓ 用于描述BIOS识别各类IPL(Initial Program Load)设备，并进行引导设备优先级排序、启动加载等功能。
- **BIOS启动时采用实模式（real mode）寻址，寻址空间为16位。**
- **BIOS启动程序采用汇编语言编写，这使得编写BIOS程序变得比较复杂。**



Compaq Computer Corporation
Phoenix Technologies Ltd.
Intel Corporation

BIOS Boot Specification

Version 1.01
January 11, 1996

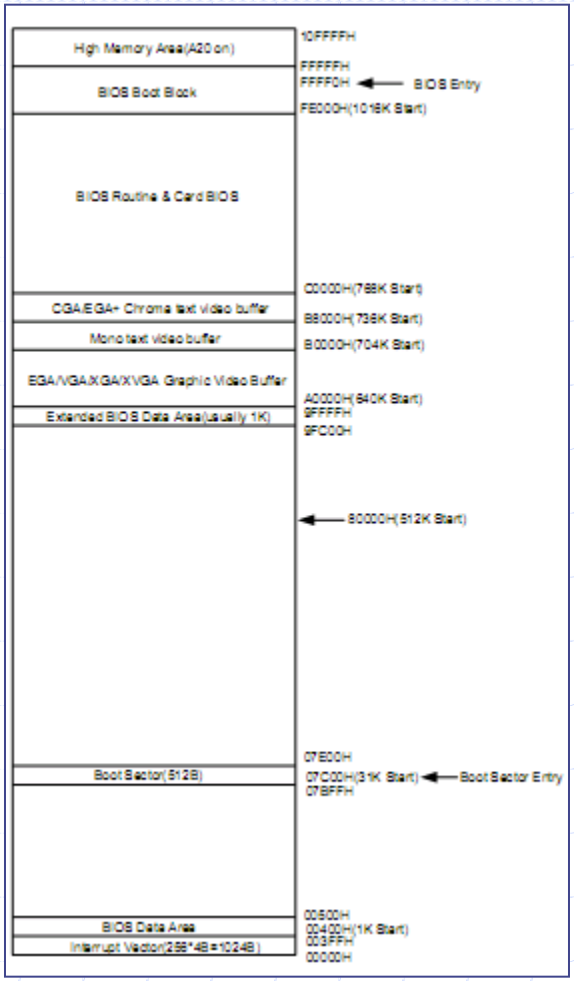
COMPAQ Phoenix intel

This specification has been made available to the public. You are hereby granted the right to use, implement, reproduce, and distribute this specification with the foregoing rights at no charge. This specification is, and shall remain, the property of Compaq Computer Corporation ("Compaq"), Phoenix Technologies Ltd ("Phoenix"), and Intel Corporation ("Intel").

NEITHER COMPAQ, PHOENIX NOR INTEL MAKE ANY REPRESENTATION OR WARRANTY REGARDING THISSPECIFICATION OR ANY PRODUCT OR ITEM DEVELOPEDBASED ON THISSPECIFICATION. USE OF THISSPECIFICATION FOR ANY PURPOSEIS AT THE RISK OF THE PERSON OR ENTITY USINGIT. COMPAQ, PHOENIX ANDINTEL DISCLAIM ALL EXPRESS AND IMPLIED WARRANTIES, INCLUDING BUT NOTLIMITED TOTHE IMPLIEDWARRANTIES OF MERCHANTABILITY,FITNESS FOR APARTICULAR PURPOSEAND FREEDOM FROM INFRINGEMENT. WITHOUTLIMITINGTHE GENERALITY OF THE FOREGOING, NEITHER COMPAQ, PHOENIX NOR INTEL MAKE ANY WARRANTY OF ANY KIND THAT ANY ITEM DEVELOPEDBASED ON THISSPECIFICATION, OR ANY PORTION OF IT,WILL NOT INFRINGE ANY COPYRIGHT, PATENT, TRADE SECRET OR OTHER INTELLECTUAL PROPERTY RIGHT OF ANY PERSON OR ENTITY IN ANY COUNTRY.

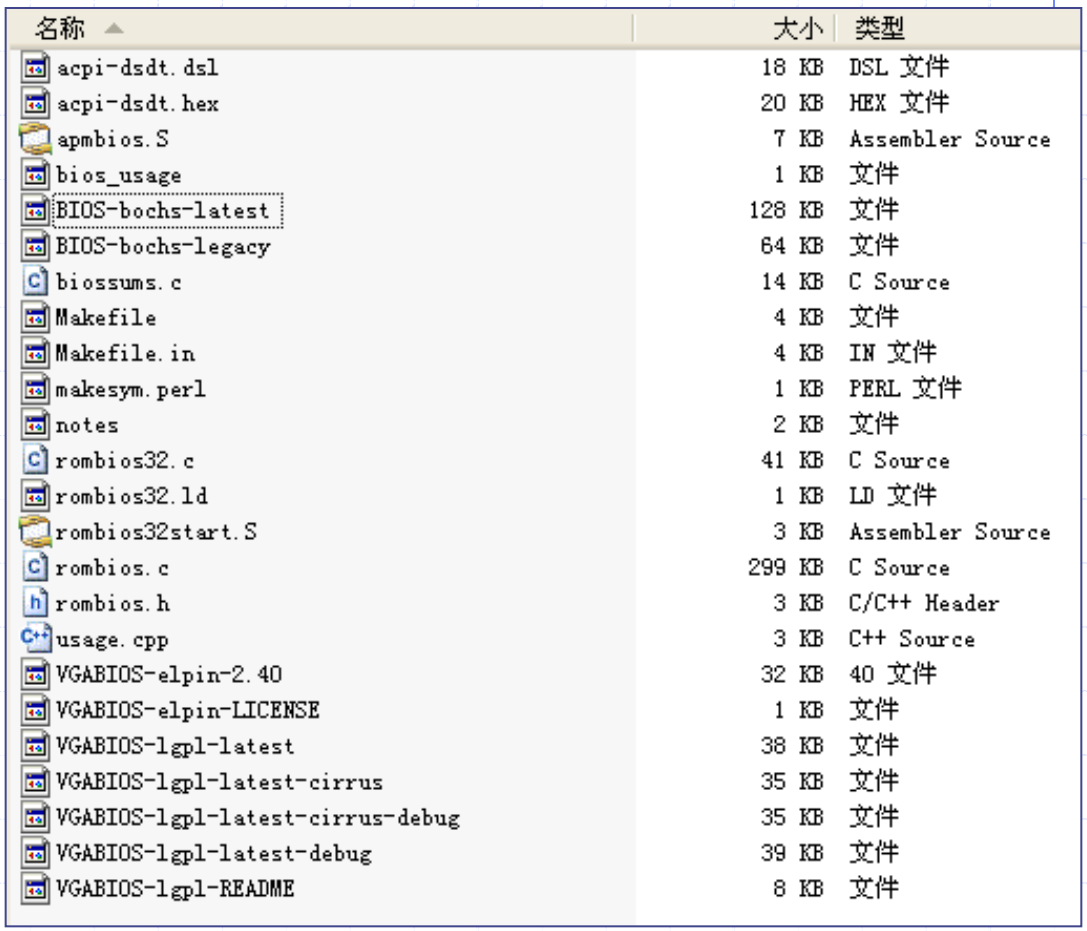
BIOS—内存分布图

- BIOS内存分布图
  - BIOS Entry
    - ✓0xFFFF0h
  - BIOS Boot Block
    - ✓0xFe000h ～ 0xFFFFFh
  - （Extended）BIOS Data Area
  - Boot Sector
    - ✓0x7c00h
  - Interrupt Vector
    - ✓0x0000h ～ 0x3ffh



BIOS—构成分析

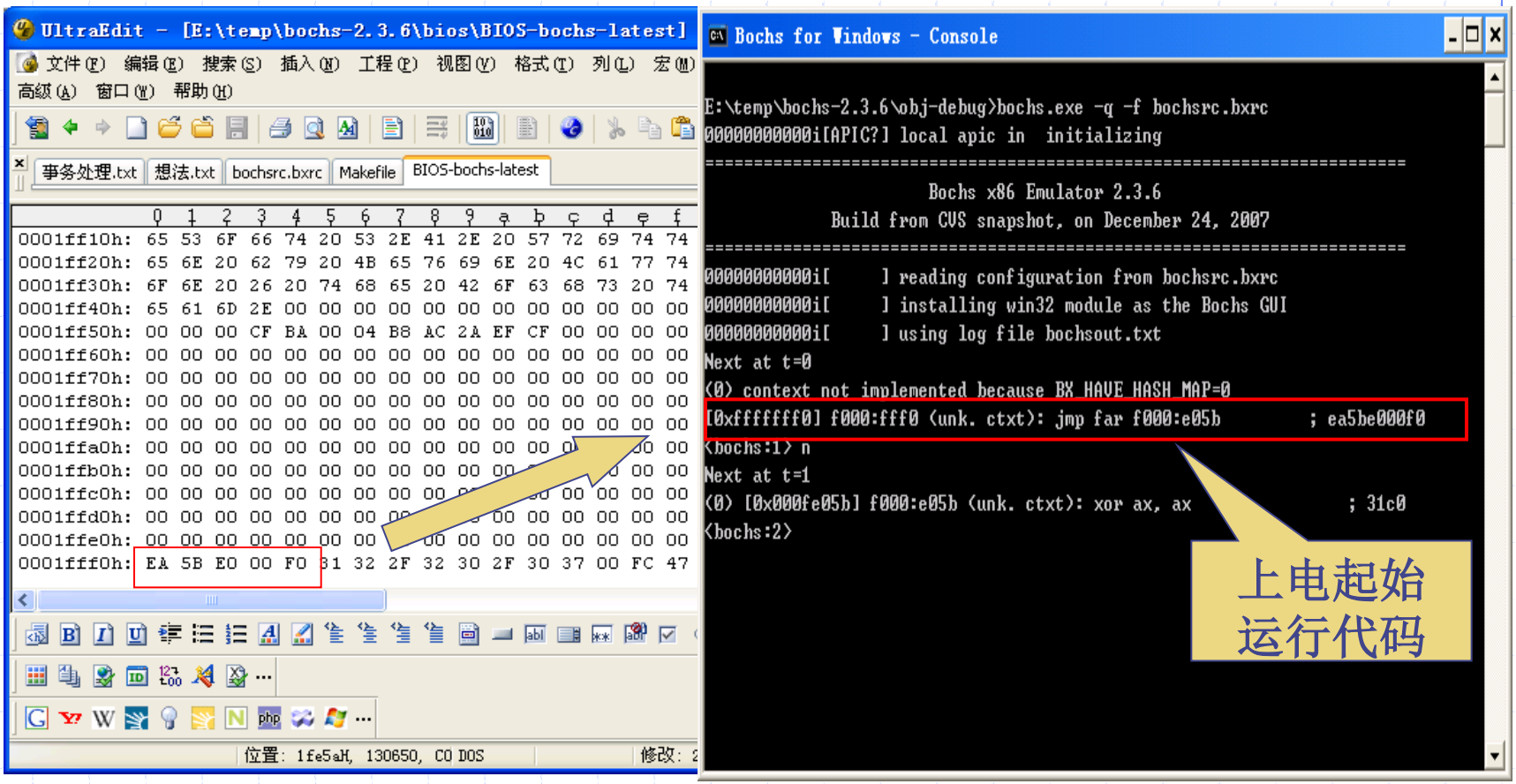
- BIOS映像文件
  - ✓ BIOS-bochs-latest
- 源代码
  - ✓ C程序：rombios.c、rombios32.c
  - ✓ 汇编代码：rombios32start.S
- Makefile
- BIOS映像生成工具
  - ✓ biossums.c



名称	大小	类型
acpi-dsdt.dsl	18 KB	DSL 文件
acpi-dsdt.hex	20 KB	HEX 文件
apmbios.S	7 KB	Assembler Source
bios_usage	1 KB	文件
BIOS-bochs-latest	128 KB	文件
BIOS-bochs-legacy	64 KB	文件
biossums.c	14 KB	C Source
Makefile	4 KB	文件
Makefile.in	4 KB	IN 文件
makesym.perl	1 KB	PERL 文件
notes	2 KB	文件
rombios32.c	41 KB	C Source
rombios32.ld	1 KB	LD 文件
rombios32start.S	3 KB	Assembler Source
rombios.c	299 KB	C Source
rombios.h	3 KB	C/C++ Header
usage.cpp	3 KB	C++ Source
VGABIOS-elpin-2.40	32 KB	40 文件
VGABIOS-elpin-LICENSE	1 KB	文件
VGABIOS-lgpl-latest	38 KB	文件
VGABIOS-lgpl-latest-cirrus	35 KB	文件
VGABIOS-lgpl-latest-cirrus-debug	35 KB	文件
VGABIOS-lgpl-latest-debug	39 KB	文件
VGABIOS-lgpl-README	8 KB	文件

BIOS—映像文件分析

- BIOS-bochs-latest



BIOS模拟－rombios.c分析

- 16位实模式程序
  - ✓ Power-up Entry Point（系统上电入口）：0xfff0
  - ✓ POST Entry Point（系统检测入口）：0xe05b
  - ✓ INT 19H（系统引导入口）：0xe6f2 -> int19_relocated -> int19_function -> int19_load_done
  - ✓ INT 18H：int18_handler
  - ✓ Jump to the boot vector

下一代BIOS——UEFI (1/3)

- BIOS的局限性
  - ✓16位模式：寻址能力低下，不能支持2TB大硬盘
  - ✓扩展能力问题：驱动空间固定为128KB
  - ✓汇编语言：开发困难
  - ✓安全性不足：简单密码保护，无法监控其安全
  - ✓界面不够人性化：文字界面，无图形化操作界面
  - ✓……

下一代BIOS——UEFI （2/3)

- UEFI(Unified Extensible Firmware Interface，统一可扩展固件接口)发展历程

业界BIOS演进

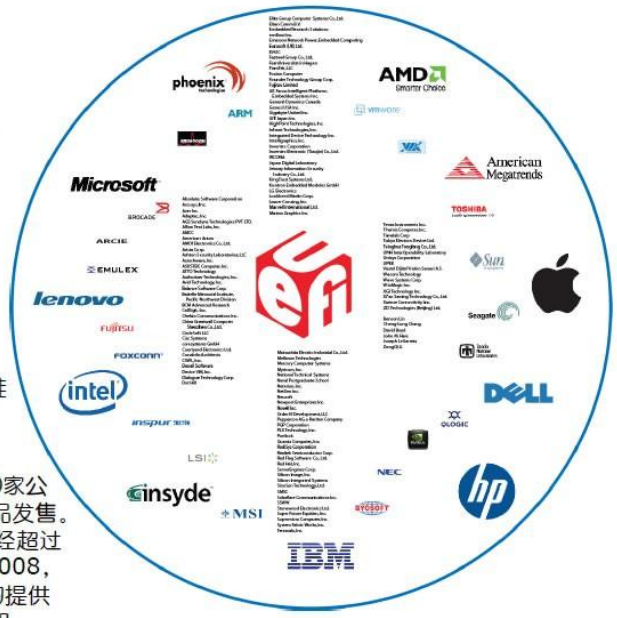
之前 所有平台的BIOS都是私有的

2000 英特尔推出可扩展固件接口(EFI)规范并提供遵照free BSD条款的示例实现

2004 **tianocore.org**，EFI开源社区启动

2005 11家公司联合创办**统一可扩展固件接口(UEFI)**论坛以推进EFI规范的标准化

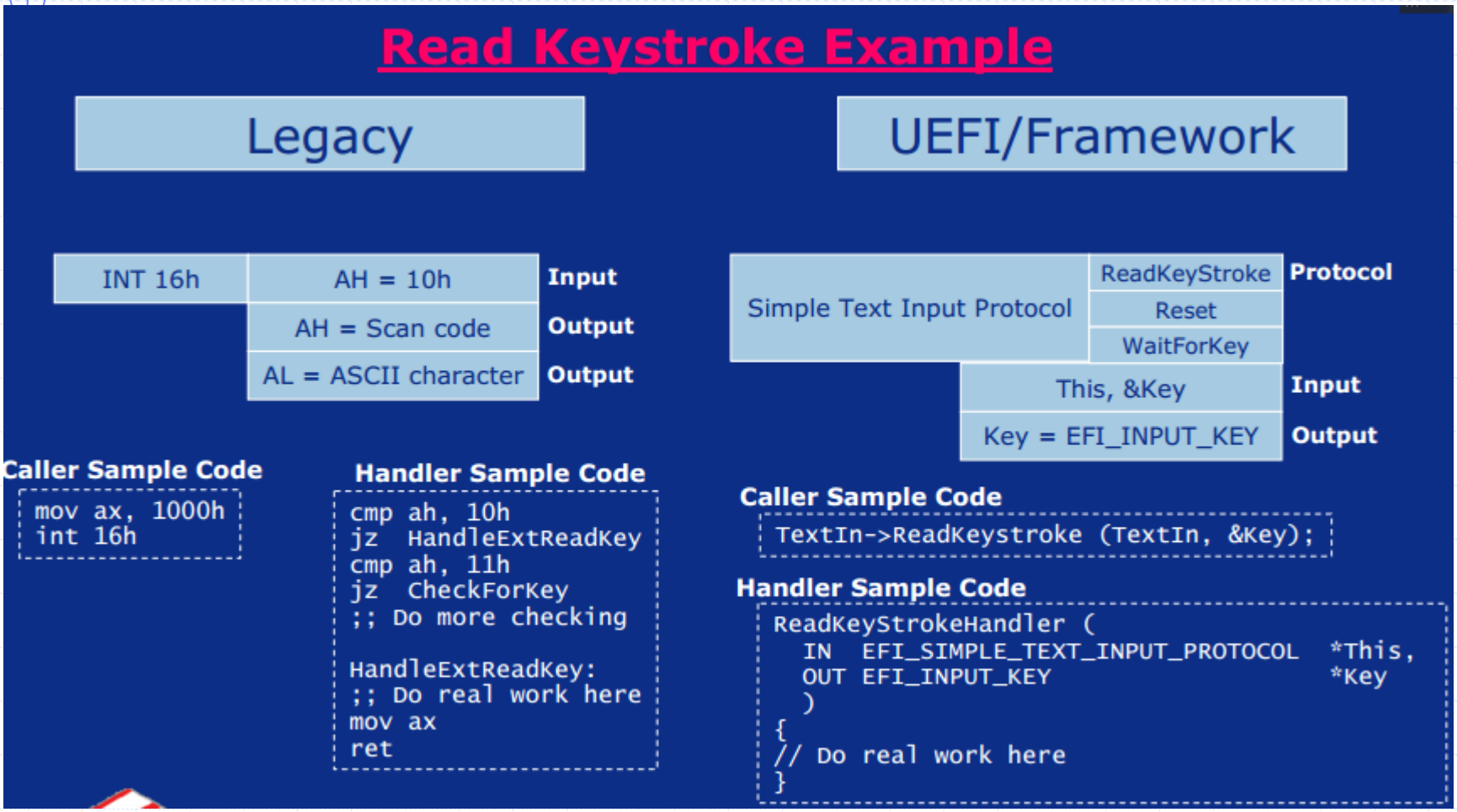
2011 UEFI论坛茁壮成长，已有170家公司加入！主要的MNC均有产品发售。IA架构中使用UEFI的平台已经超过50%，Microsoft* Server 2008，Vista* 和Win7* 操作系统均提供64位UEFI的支持，Redhat*和Novell* OS也提供对UEFI的支持



下一代BIOS——UEFI (3/3)

- UEFI的优势
  - ✓ 32/64位寻址模式，支持2T硬盘，驱动空间大大提高
  - ✓ 模块化、C语言风格、动态链接形式系统结构
  - ✓ Driver/protocol替代中断/硬件端口操作，启动速度快
  - ✓ 采用Flat mode替代X86实模式
  - ✓ 采用Removable Binary Drivers替代二进制code
  - ✓ 提供UEFI shell命令
  - ✓ ……

UEFI特点解析 (1/2)

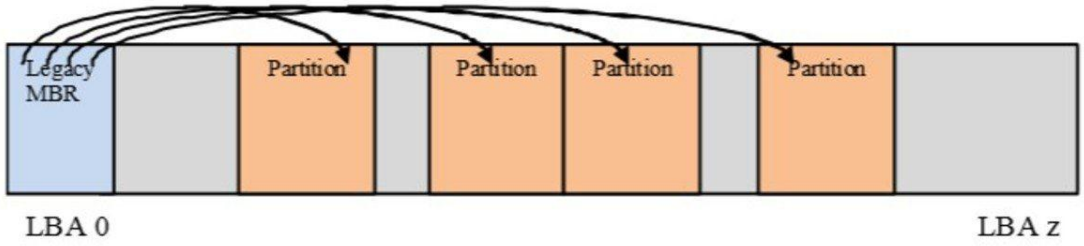


- Driver/protocol

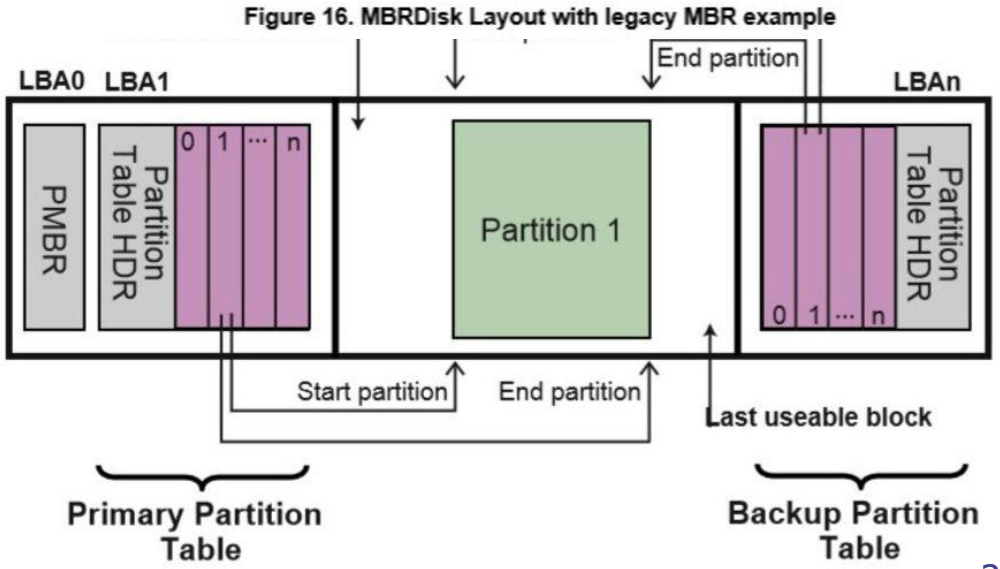
UEFI特点解析 (2/2)

- UEFI引导方式

BIOS+MBR



UEFI+GPT



嵌入式系统的Boot Loader程序（1/2）

- 在嵌入式系统中没有BIOS/UEFI那样的固件程序，因此，整个系统的加载启动任务完全由BootLoader来完成。
- 基于ARM内核的嵌入式系统，加电或复位的地址为0x00000000，Boot Loader程序的入口就安排在该地址上。

嵌入式系统的Boot Loader程序（2/2）

- Boot Loader的实现依赖于硬件环境
  - CPU体系结构：ARM、PPC、x86、MIPS
  - 板级设备：时钟、FLASH、通讯端口
  - Boot Loader的入口地址
  - Boot Loader与开发机的通讯机制
    - 串口
    - 网络
- 专用Boot Loader
- 通用Boot Loader：U-boot、Redboot

由Boot Loader启动操作系统的方式

- Flash启动方式

Bootloader—参数区－操作系统内核－文件系统

- 硬盘启动方式
  - 在硬盘主引导区放置bootloader
  - 从文件系统中引导操作系统
- 网络启动方式
  - Bootloader放置在EPROM或Flash中
  - 通过以太网远程下载操作系统内核或文件系统
  - 开发板不需配置大的存储介质

Boot Loader的功能种类

- 简单Bootloader
  - 只具有系统引导功能
- 具有监控功能（Monitor）的Bootloader
  - 调试支持
  - 内存读写
  - Flash烧写
  - 网络下载
  - 环境变量配置

开放源码的Boot Loader程序

名称	支持体系结构	是否Monitor
LILO	X86	否
GRUB	X86	否
BLOB	ARM	否
Etherboot	X86	否
U—boot	X86、ARM、PPC	是
RedBoot	X86、ARM、PPC	是
VIVI	ARM	是

课程大纲

- Bootloader程序的基本概念
- Bootloader典型框架结构
- S3C2410 Bootloader代码分析

Boot Loader的典型框架结构

- Boot Loader的启动过程通常是多阶段的
  - 提供复杂的功能：突破引导扇区512字节限制
  - 提高代码移植性：高阶段代码采用高级语言
  - 提高运行速度　：高阶段代码在内存中执行
- 大多数Boot Loader可分为阶段1和阶段2两大部分
  - 阶段1：实现依赖于CPU体系结构的代码
  - 阶段2：实现一些复杂的功能

Boot Loader阶段1介绍（1/4）

- Boot Loader的阶段1通常包括以下步骤：
- 1）硬件设备初始化
  - 屏蔽所有的中断
  - 设置CPU的速度和时钟频率
  - RAM初始化
  - 初始化LED
  - 关闭CPU内部指令 / 数据Cache

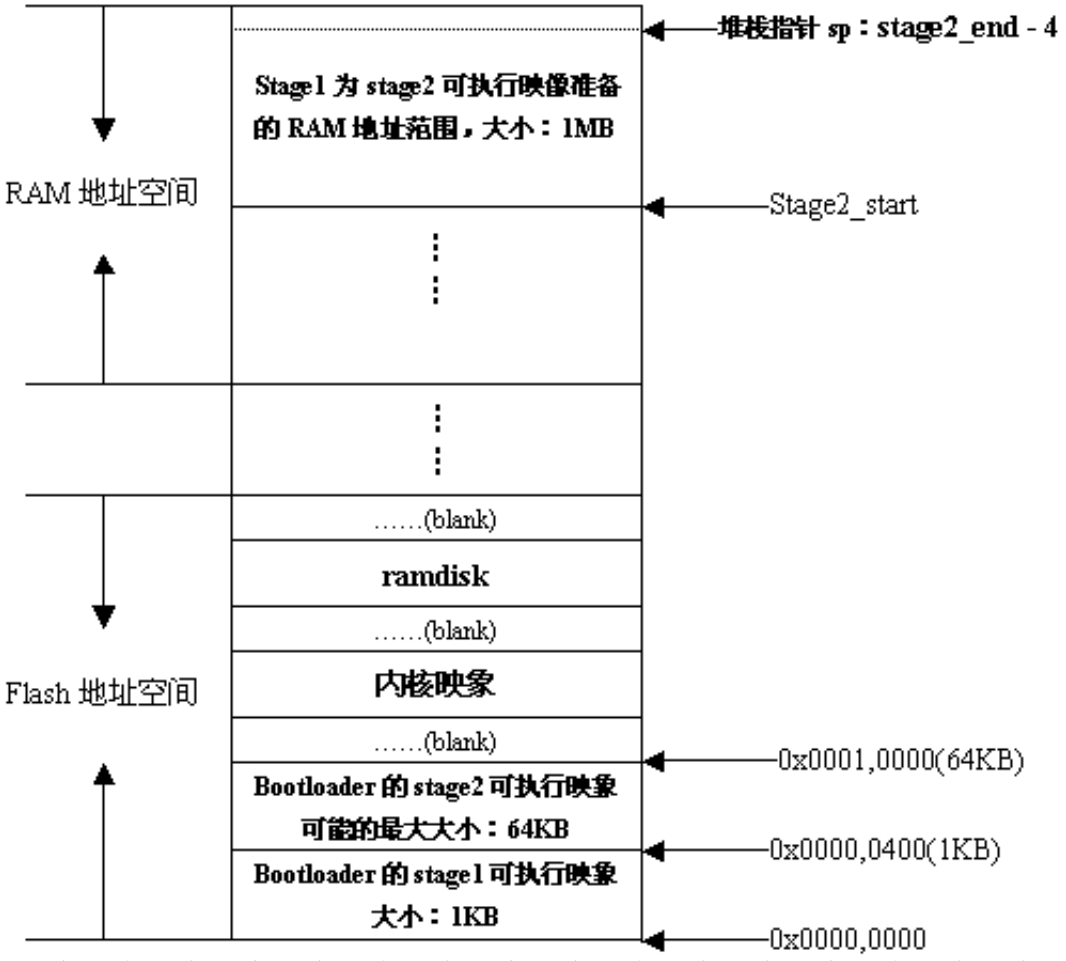
Boot Loader阶段1介绍（2/4）

- 2）为加载阶段2准备RAM空间
  - 除了阶段2可执行映象的大小外，还必须把堆栈空间也考虑进来
  - 必须确保所安排的地址范围的确是可读写的RAM空间
  - 内存区域有效性检测方法
    - 保存指定内存区域
    - 写入预定数据
    - 读入数据并比较
    - 恢复内存数据

Boot Loader阶段1介绍（3/4）

- 3）拷贝阶段2代码到RAM中
- 4）设置堆栈指针sp
- 5）跳转到阶段2的C语言入口点
- Boot Loader 的阶段1执行完成后的RAM空间布局，如下图：

Boot Loader阶段1介绍（4/4）



Boot Loader阶段2介绍（1/8）

- 1）初始化本阶段要使用到的硬件设备
  - 初始化至少一个串口，以便和终端用户进行I/O输出信息
  - 初始化计时器等

Boot Loader阶段2介绍（2/8）

- 2）检测系统的内存映射
  - 内存映射的描述
  - 可以用如下数据结构来描述RAM地址空间中的一段连续的地址范围：

```
typedef struct memory_area_struct {
        u32 start; /* 内存空间的基址 */
        u32 size;  /* 内存空间的大小 */
        int used;
} memory_area_t;
```

  - 内存映射的检测

Boot Loader阶段2介绍（3/8）

- 连续内存区域探测——X86系统内存探测算法举例

```
for (p = (char *)0x100000; (int)p < 0x40000000; p += delta)
{
    for (ix = 0; ix < N_TIMES; ix++)  /* 保存内存原有信息 */
   {   temp[ix] = *((int *)p + ix);
       *((int *)p + ix) = TEST_PATTERN;  /*TEST_PATTERN
       0x12345678*/
    }
    cacheFlush (DATA_CACHE, p, 4 * sizeof(int));
    if (*(int *)p != TEST_PATTERN)    /* 测试内存单元有效性 */
       p -= delta;
    for (ix = 0; ix < N_TIMES; ix++)   /* 恢复内存原有信息 */
       *((int *)p + ix) = temp[ix];
}
```

Boot Loader阶段2介绍（4/8）

- 非连续内存区域探测——具有空洞内存的探测

```
from 内存区域低端 to 内存区域最高端，以页为单位进行内存探测
{
    对当前内存进行检测；

    if (当前内存页状态 与 前一内存页状态一致)
        合并内存状态情况；
    else
        记录当前内存区域状态。
}
```

Boot Loader阶段2介绍（5/8）

- 3）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
      - MEM_START + 0X8000
    - 根文件系统所占用的内存范围
      - MEM_START + 0X00100000
  - 从Flash上拷贝
    - While循环

Boot Loader阶段2介绍（6/8）

- 4）设置内核的启动参数
  - 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
  - 嵌入式Linux系统中，通常需要由Boot Loader设置。常见启动参数有：ATAG_CORE、ATAG_MEM、ATAG_CMDLINE、ATAG_RAMDISK、ATAG_INITRD，以ATAG_NONE结束。

Boot Loader阶段2介绍（7/8）

- 例：设置ATAG_MEM的代码如下：

```
params->hdr.tag = ATAG_MEM;
params->hdr.size = tag_size(tag_mem32);
params->u.mem.start = memory_map[i].start;
params->u.mem.size = memory_map[i].size;
params = tag_next(params);
```

指针params是一个struct tag类型的指针。宏tag_next()将以指向当前标记的指针为参数，计算出当前标记的下一个标记的起始地址。

Boot Loader阶段2介绍（8/8）

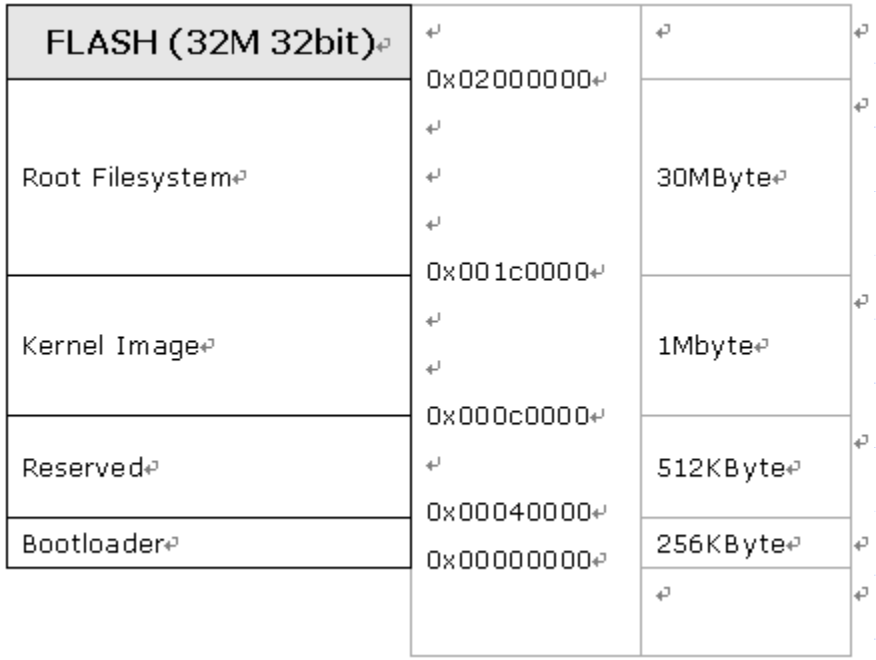
- 5）调用内核
  - CPU寄存器的设置：
    - R0＝0；
    - R1＝机器类型ID；关于机器类型号，可以参见：
    - linux/arch/arm/tools/mach-types。
    - R2＝启动参数标记列表在RAM中起始基地址；
  - CPU 模式：
    - 必须禁止中断（IRQs和FIQs）；
    - CPU必须SVC模式；
  - Cache和MMU的设置：
    - MMU必须关闭；
    - 指令Cache可以打开也可以关闭；
    - 数据Cache必须关闭。

课程大纲

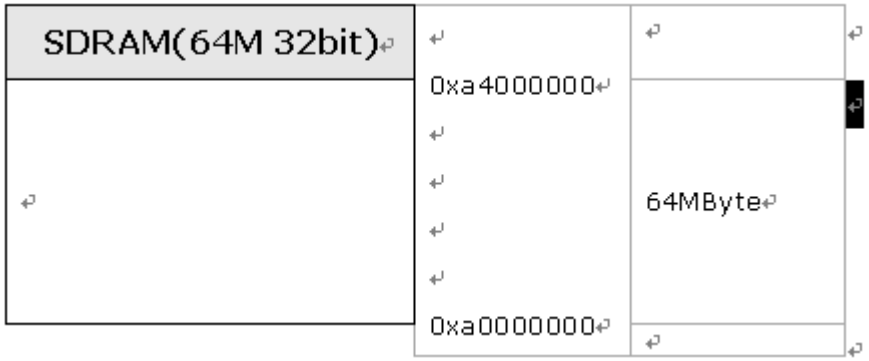
- Bootloader程序的基本概念
- Bootloader典型框架结构
- S3C2410 Bootloader代码分析

S3C2410 内存地址映像

- Flash内存映像



- SDRAM内存映像



S3C2410 中断向量设置

Address	Exception	Mode on entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	*Reserved*	*Reserved*
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

S3C2410 Bootloader—vivi

- vivi是韩国Mizi公司开发的一种bootloader，适用于ARM9处理器，支持S3C2410处理器。

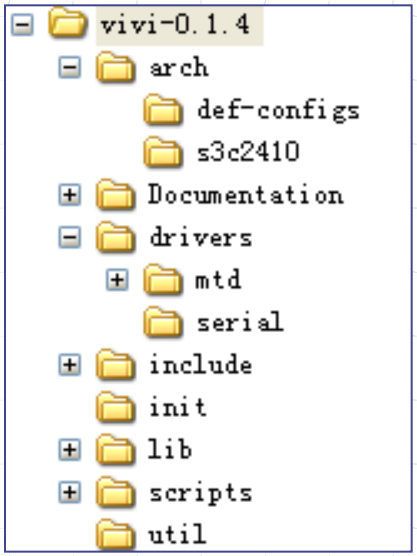
- vivi有两种工作模式：
  - 启动加载模式
  - 下载模式：提供命令行接口

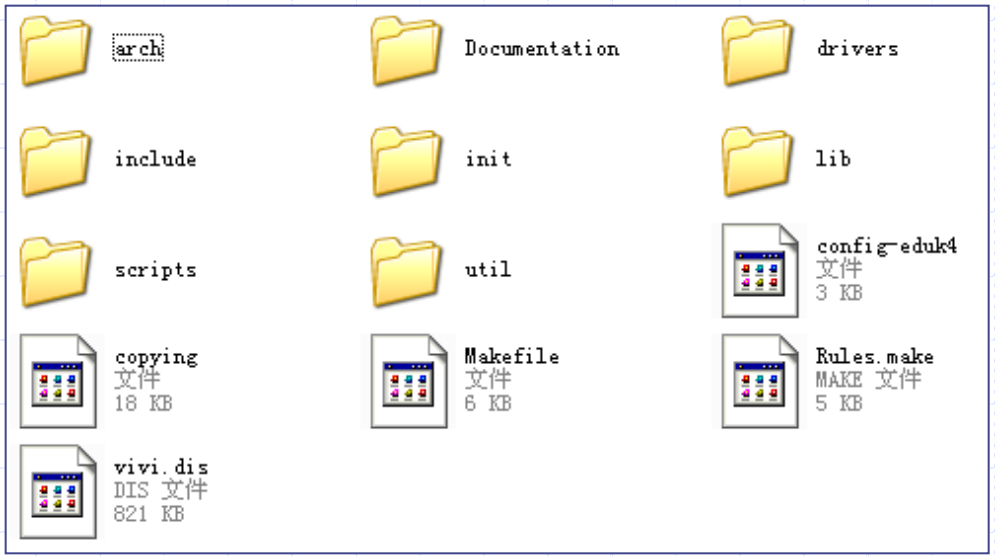
vivi命令举例

命令名称	用途	举例
help	命令使用帮助	help part
rest	系统复位	rest
boot	引导操作系统	boot
part	MTD设备分区操作	part add partname
load	装载文件	load flash vivi x
param	设置bootloader参数	param show
mem	内存管理	mem read 0x20000

vivi源代码分析—目录结构

- 代码目录结构分析





vivi源代码分析—阶段1

- Bootloader阶段1代码分析
  - 禁止看门狗、中断
  - 初始化时钟、内存
  - 设置LED、GPIO
  - 拷贝vivi至内存
    - 初始化nand flash
    - 设置堆栈寄存器、入口参数
    - 调用C函数将vivi由nand flash拷贝至内存
  - 跳转至内存执行
  - 设置堆栈寄存器、入口参数
  - 调用C函数main

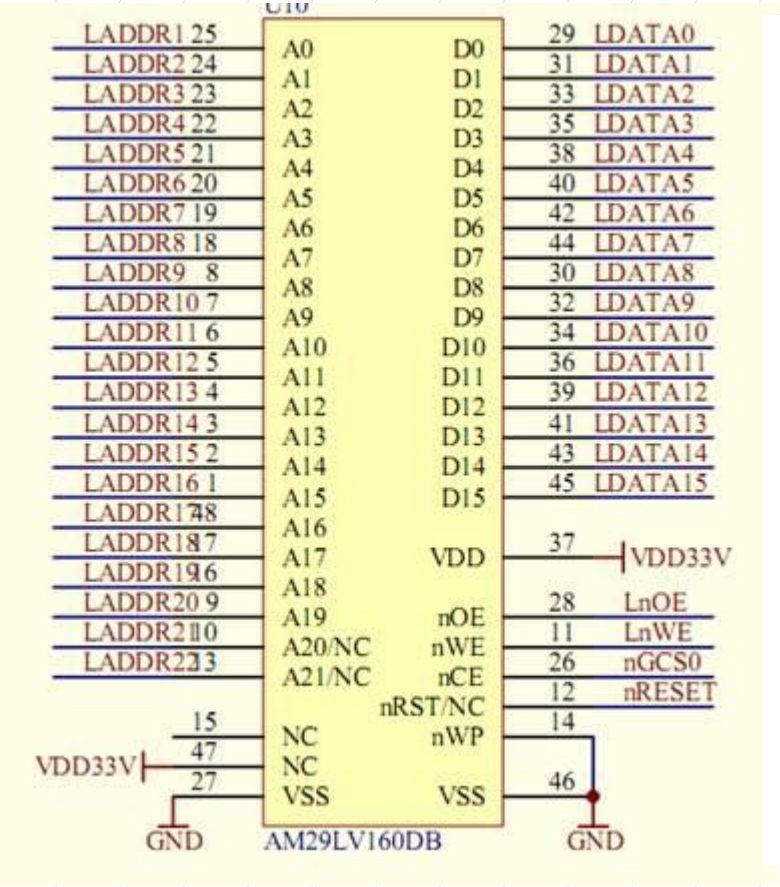
vivi源代码分析一阶段2

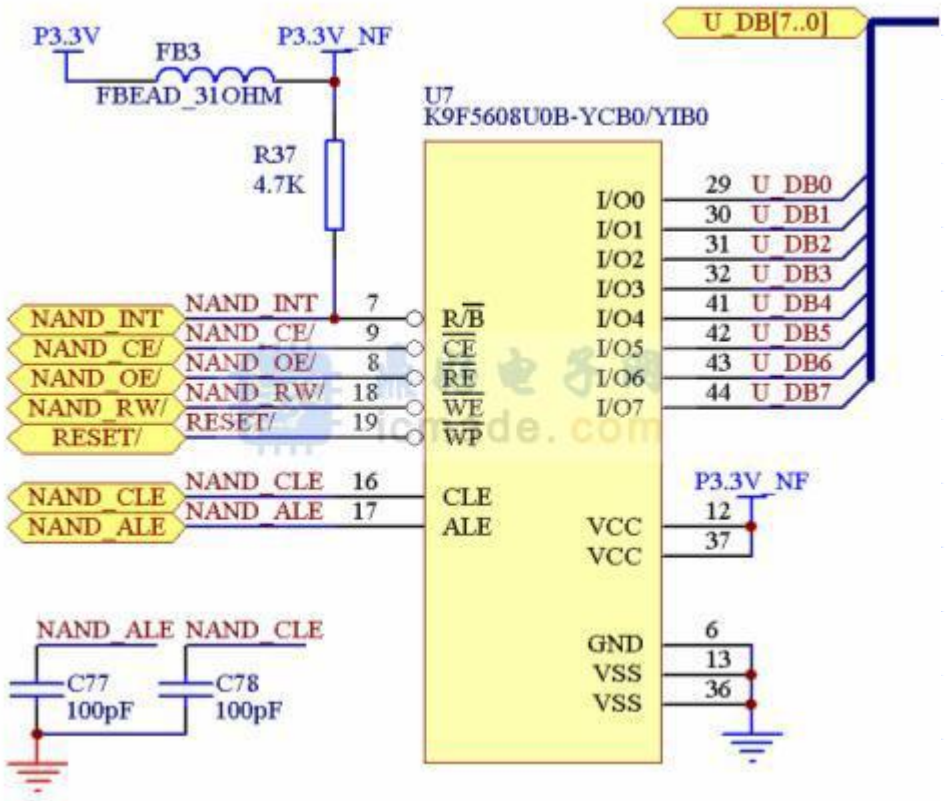
- Bootloader阶段2代码分析
  - 打印系统启动标识
  - 初始化CACHE、MMU、Heap
  - 初始化MTD设备
  - 初始化系统参数
  - 命令装载
  - 启动shell 或 启动操作系统

嵌入式系统代码执行方式

- 完全映射 (fully shadowed)：将代码从非易失存储器复制到RAM中运行。
- 按需分页 (demand paging)：复制部分代码到RAM中，如果访问位于虚存中但不在物理RAM中产生页错误，再将代码装载到RAM中。
- 片上执行(eXecute In Place, XIP)：系统启动时代码直接在非易失性存储(NOR Flash、ROM)中执行。

NOR Flash VS NAND Flash

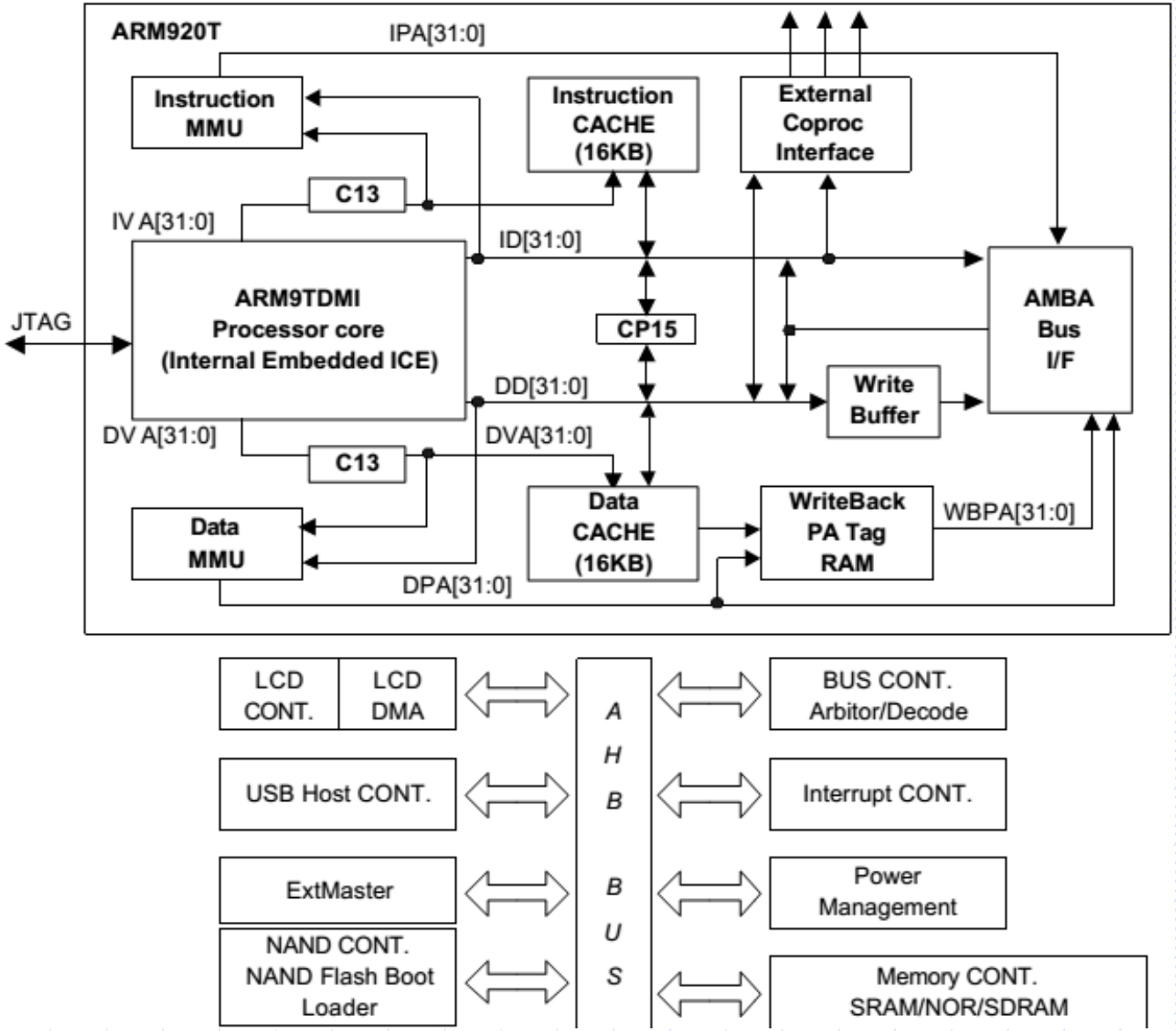




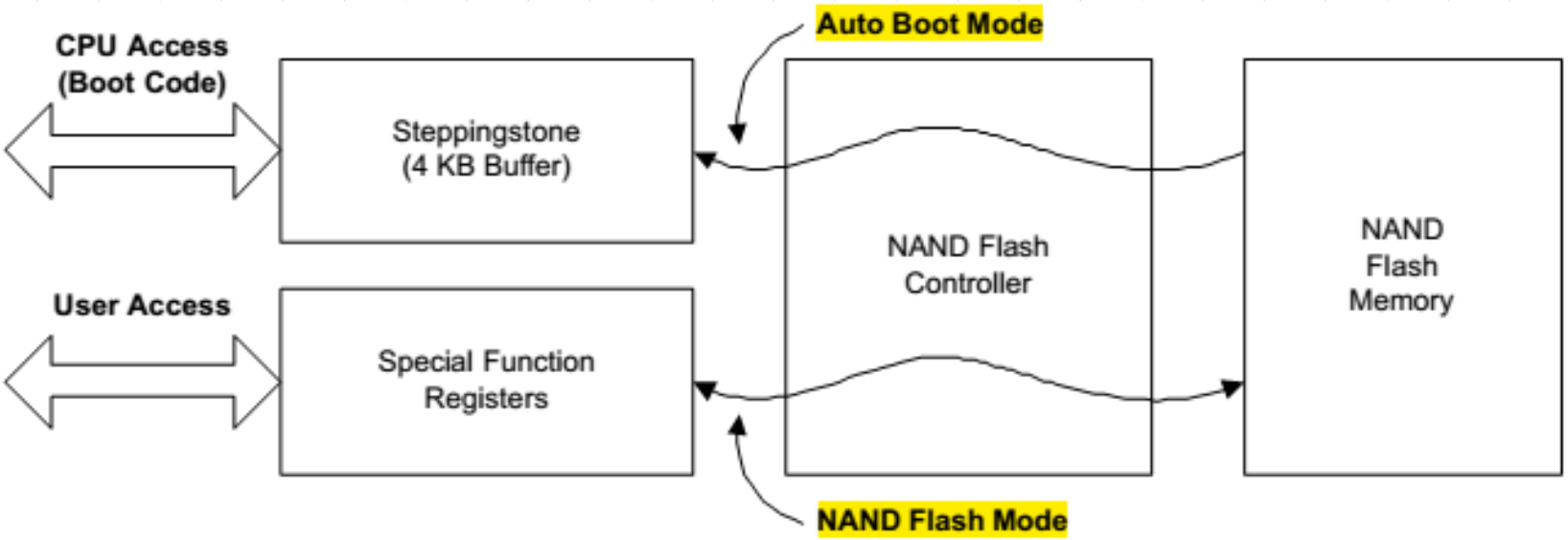
NOR FLASH(XIP)

NAND FLASH(非XIP)

S3C2410 Block Diagram

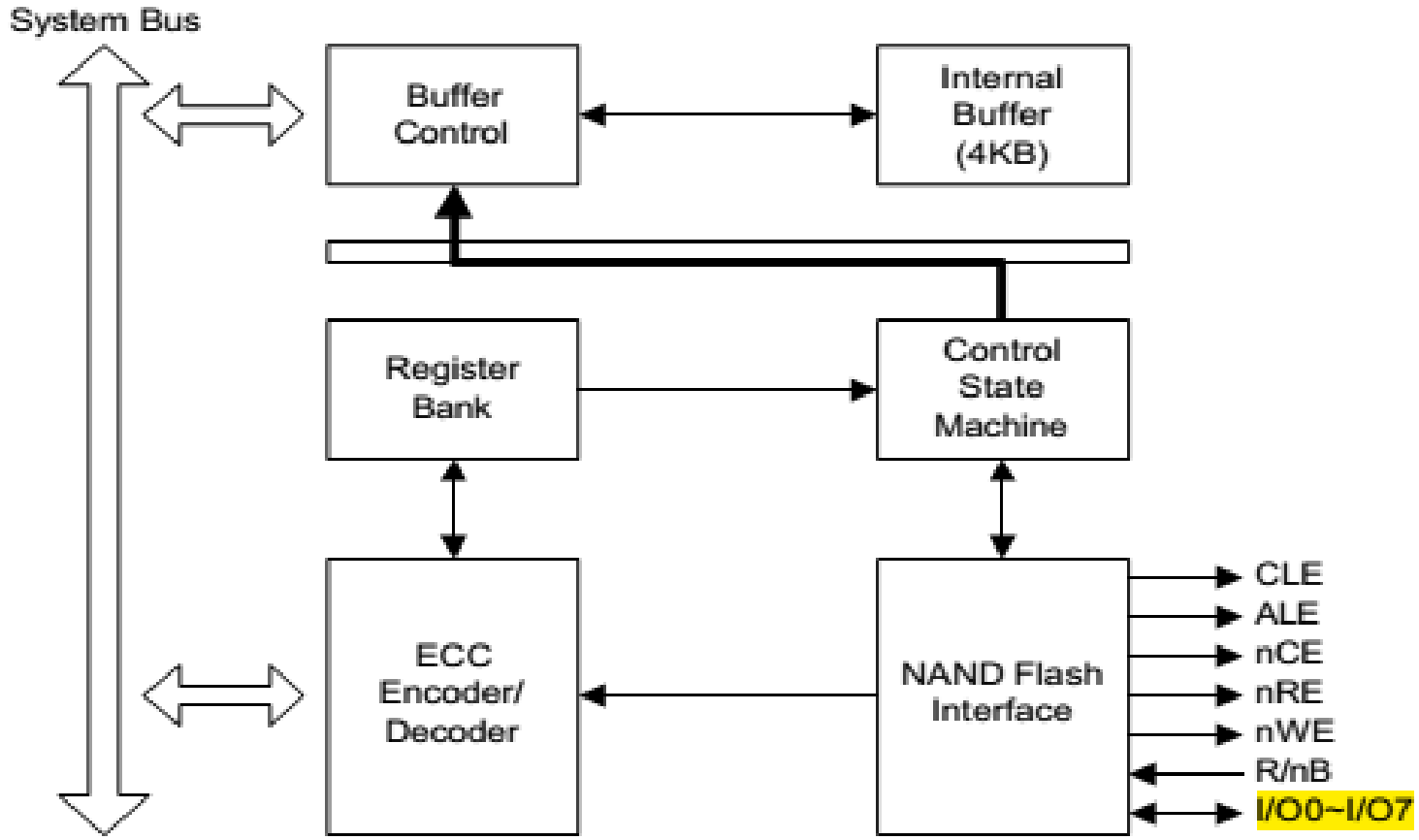


S3C2410 NAND Flash Operation Scheme



- Auto boot mode : The boot code is transferred into Steppingstone after reset. After the transfer, the boot code will be executed on the Steppingstone.

S3C2410 NAND Flash Block Diagram



S3C2410 NAND Flash Special Registers

Register Name	Address (B. Endian)	Address (L. Endian)	Acc. Unit	Read/ Write	Function
NAND Flash					
NFCONF	0x4E000000	←	W	R/W	NAND Flash Configuration
NFCMD	0x4E000004				NAND Flash Command
NFADDR	0x4E000008				NAND Flash Address
NFDATA	0x4E00000C				NAND Flash Data
NFSTAT	0x4E000010			R	NAND Flash Operation Status
NFECC	0x4E000014			R/W	NAND Flash ECC

谢 谢！