
板级支持包与系统引导

主要内容

- 嵌入式软件运行流程
- 版级支持包 BSP
- Bootloader

嵌入式软件运行流程

基于多任务操作系统的嵌入式软件的主要运行流程

嵌入式软件运行流程-I

- 上电复位、板级初始化阶段
  - 嵌入式系统上电复位后完成板级初始化工作。
  - 板级初始化程序具有完全的硬件特性，一般采用汇编语言实现。不同的嵌入式系统，板级初始化时要完成的工作具有一定的特殊性，但以下工作一般是必须完成的：
    - CPU中堆栈指针寄存器的初始化。
    - BSS段（Block Storage Space表示未被初始化的数据）的初始化。
    - CPU芯片级的初始化：中断控制器、内存等的初始化。

嵌入式软件运行流程-2

- 系统引导/升级阶段
  - 根据需要分别进入系统软件引导阶段或系统升级阶段。
  - 软件可通过测试通信端口数据或判断特定开关的方式分别进入不同阶段。

嵌入式软件运行流程-3

- 系统引导阶段，系统引导有几种情况：
  - 将系统软件从NOR Flash中读取出来加载到RAM中运行：这种方式可以解决成本及Flash速度比RAM慢的问题。软件可压缩存储在Flash中。
  - 不需将软件引导到RAM中而是让其直接在NorFlash上运行，进入系统初始化阶段。
  - 将软件从外存（如NandFlash、CF卡、MMC等）中读取出来加载到RAM中运行：这种方式的成本更低。

嵌入式软件运行流程-4

- 系统升级阶段
  - 进入系统升级阶段后系统可通过网络进行远程升级或通过串口进行本地升级。
  - 远程升级一般支持TFTP、FTP、HTTP等方式。
  - 本地升级可通过Console口使用超级终端或特定的升级软件进行。

嵌入式软件运行流程-5

- 系统初始化阶段
  - 在该阶段进行操作系统等系统软件各功能部分必需的初始化工作，如根据系统配置初始化数据空间、初始化系统所需的接口和外设等。
  - 系统初始化阶段需要按特定顺序进行，如首先完成内核的初始化，然后完成网络、文件系统等的初始化，最后完成中间件等的初始化工作。

嵌入式软件运行流程-6

- 应用初始化阶段
  - 在该阶段进行应用任务的创建，信号量、消息队列的创建和与应用相关的其它初始化工作。
- 多任务应用运行阶段
  - 各种初始化工作完成后，系统进入多任务状态，操作系统按照已确定的算法进行任务的调度，各应用任务分别完成特定的功能。

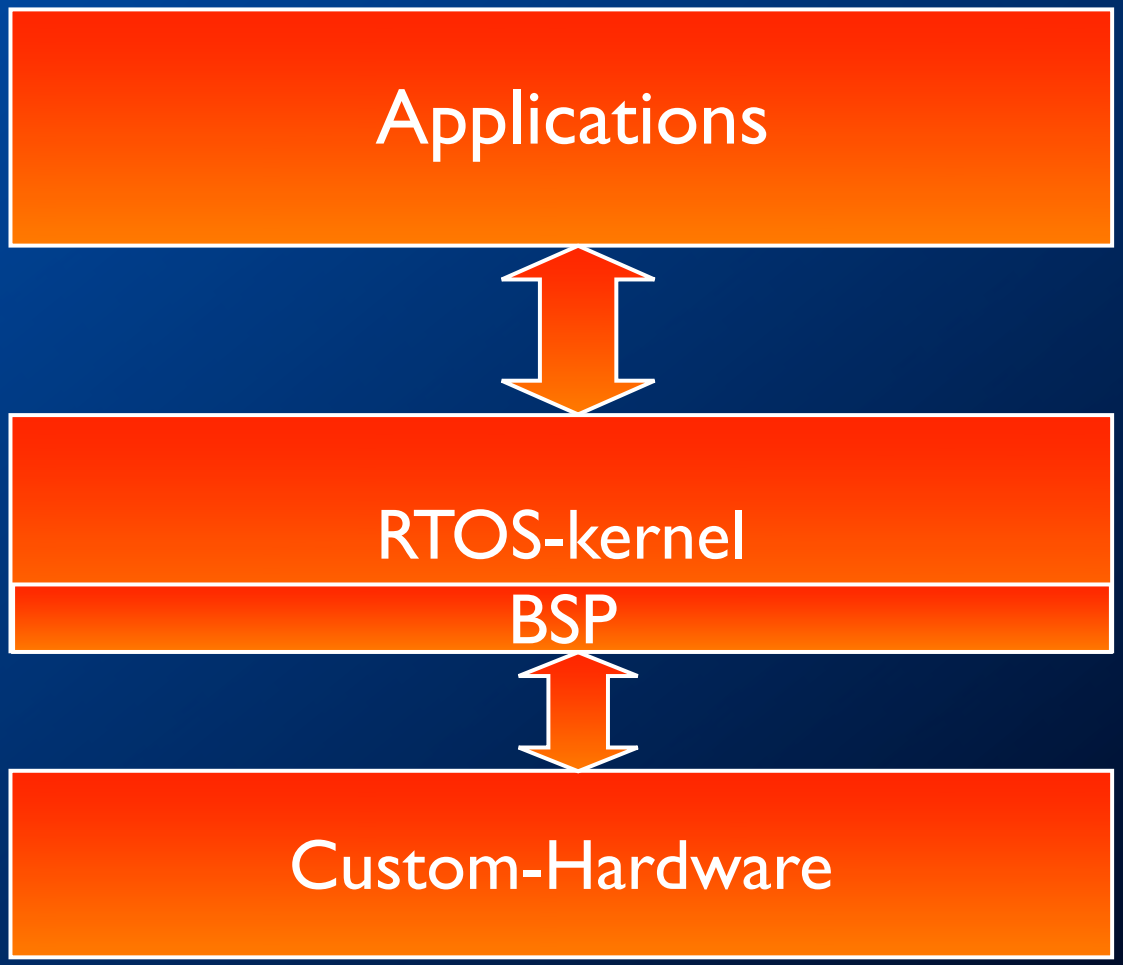
主要内容

- 嵌入式软件运行流程
- 版级支持包 BSP
- Bootloader

BSP

- BSP全称“板级支持包”（Board Support Packages），说的简单一点，就是一段启动代码，和计算机主板的BIOS差不多，但提供的功能区别就相差很大。

BSP在嵌入式系统中所处的位置



BSP中的驱动程序

- 驱动程序的基本功能
  - 对设备初始化和释放
  - 对设备进行管理
  - 读取应用程序传送给设备文件的数据，并回送应用程序的请求数据
  - 检测和处理设备出现的错误

BSP和BIOS的区别

- BIOS主要是负责在电脑开启时检测、初始化系统设备（设置栈指针，中断分配，内存初始化..）、装入操作系统并调度操作系统向硬件发出的指令。
- BSP是和操作系统绑在一起运行，尽管BSP的开始部分和BIOS所做的工作类似，但是 BSP还包含和系统有关的基本驱动 。
- BIOS程序是用户不能更改，编译编程的，只能对参数进行修改设置，但是程序员还可以编程修改BSP，在BSP中任意添加一些和系统无关的驱动或程序，甚至可以把上层开发的统统放到BSP中 。

不同系统中的BSP

- 一个嵌入式操作系统针对不同的CPU，会有不同的BSP。
- 即使同一种CPU，由于外设的一点差别BSP相应的部分也不一样 。

BSP的特点与功能

- 硬件相关性
  - 因为嵌入式实时系统的硬件环境具有应用相关性，所以，作为高层软件与硬件之间的接口，BSP必须为操作系统提供操作和控制具体硬件的方法。
- 操作系统相关性
  - 不同的操作系统具有各自的软件层次结构， 因此，不同的操作系统具有特定的硬件接口形式。

嵌入式系统初始化以及BSP的功能

- 嵌入式系统的初始化过程是一个同时包括硬件初始化和软件初始化的过程；而操作系统启动以前的初始化操作是BSP的主要功能之一
- 初始化过程总可以抽象为三个主要环境，按照自底向上、从硬件到软件的次序依次为：
  - 片级初始化
  - 板级初始化
  - 系统级初始化

初始化过程

- 片级初始化：
  - 主要完成CPU的初始化
    - 设置CPU的核心寄存器和控制寄存器
    - CPU核心工作模式
    - CPU的局部总线模式等
  - 片级初始化把CPU从上电时的缺省状态逐步设置成为系统所要求的工作状态
  - 这是一个纯硬件的初始化过程

初始化过程（续I）

- 板级初始化：
  - 完成CPU以外的其他硬件设备的初始化
  - 同时还要设置某些软件的数据结构和参数，为随后的系统级初始化和应用程序的运行建立硬件和软件环境
  - 这是一个同时包含软硬件两部分在内的初始化过程

初始化过程（续2）

- 系统级初始化：
  - 这是一个以软件初始化为主的过程，主要进行操作系统初始化
  - BSP将控制转交给操作系统，由操作系统进行余下的初始化操作：
    - 包括加载和初始化与硬件无关的设备驱动程序
    - 建立系统内存区
    - 加载并初始化其他系统软件模块（如网络系统、文件系统等）
  - 最后，操作系统创建应用程序环境并将控制转交给应用程序的入口

主要内容

- 嵌入式软件运行流程
- 版级支持包 BSP
- Bootloader

RTOS的引导模式

- 操作系统引导概念：将操作系统装入内存并开始执行的过程。
- 按时间效率和空间效率不同的要求，分为两种模式：
  - 需要BootLoader的引导模式：节省空间，牺牲时间，适用于硬件成本低，运行速度快，但启动速度相对慢
  - 不需要BootLoader的引导模式：时间效率高，系统快速启动，直接在NOR flash或ROM系列非易失性存储介质中运行，但不满足运行速度的要求。

BootLoader

- 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

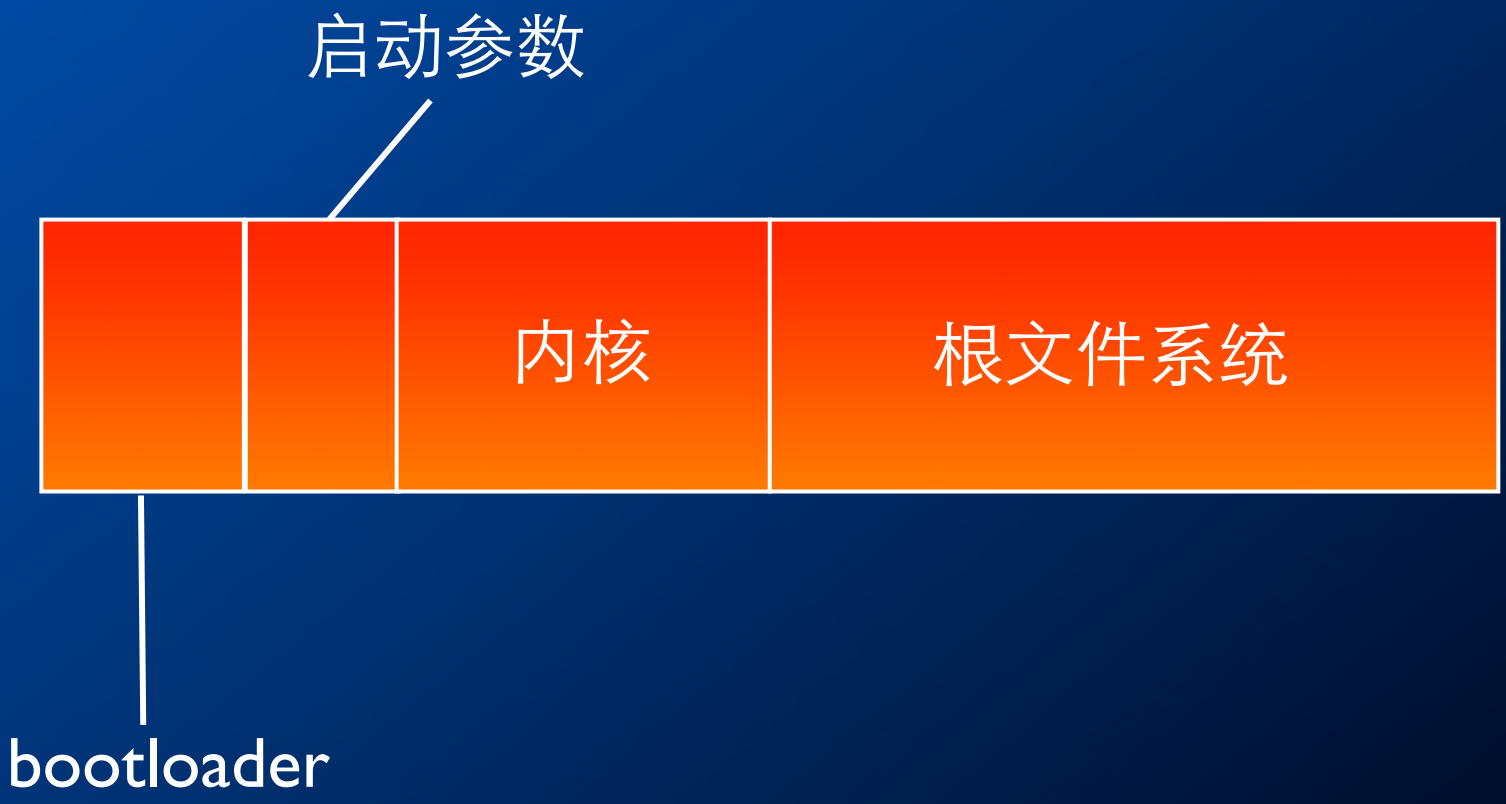
PC 机中的引导加载程序

- 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）
- 流程
  - 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。

嵌入式系统中引导加载程序

- 没 BIOS 那样的固件程序
  - 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
  - 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

固态存储设备的典型空间分配结构



用来控制 Boot Loader 的设备或机制

- 主机和目标机之间一般通过串口建立连接
- Boot Loader执行时通常会通过串口进行 I/O
  - 如输出打印信息到串口，从串口读取用户控制字符等

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
  - 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
- BOOTLOADER一般分为2部分
  - 汇编部分执行简单的硬件初始化
  - C语言部分负责复制数据,设置启动参数,串口通信等功能
- BOOTLOADER的生命周期
  - 1. 初始化硬件,如设置UART(至少设置一个),检测存储器等
  - 2. 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.
  - 3. 跳转到操作系统的首地址.
  - 4. 消亡

Boot Loader 的操作模式 (Operation Mode)

- 两种不同的操作模式
  - 启动加载模式
    - 自主（Autonomous）模式
    - 从目标机上的某个固态存储设备上将操作系统加载到 RAM 中运行
    - Boot Loader 的正常工作模式
  - 下载模式
    - 通过串口连接或网络连接等通信手段从主机（Host）下载文件
      - 如：下载内核映像和根文件系统映像等。
    - 从主机下载的文件通常首先被 Boot Loader 保存到目标机的 RAM 中，然后再被 BootLoader 写到目标机上的FLASH 类固态存储设备中。
    - 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
    - 通常都会向它的终端用户提供一个简单的命令行接口
- Blob 或 U-Boot 等功能强大的 Boot Loader 通常同时支持这两种工作模式
  - 允许用户在这两种工作模式之间进行切换
    - 如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

BootLoader 与主机之间进行文件传输所用的通信设备及协议

- 通常目标机上的 Boot Loader 通过串口与主机之间进行文件传输
- 传输协议
  - 通常是 xmodem／ymodem／zmodem 协议中的一种
- 可通过以太网连接并借助 TFTP 协议来下载文件
  - 串口传输的速度是有限的
  - 主机提供 TFTP 服务

Boot Loader 的主要任务

- 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

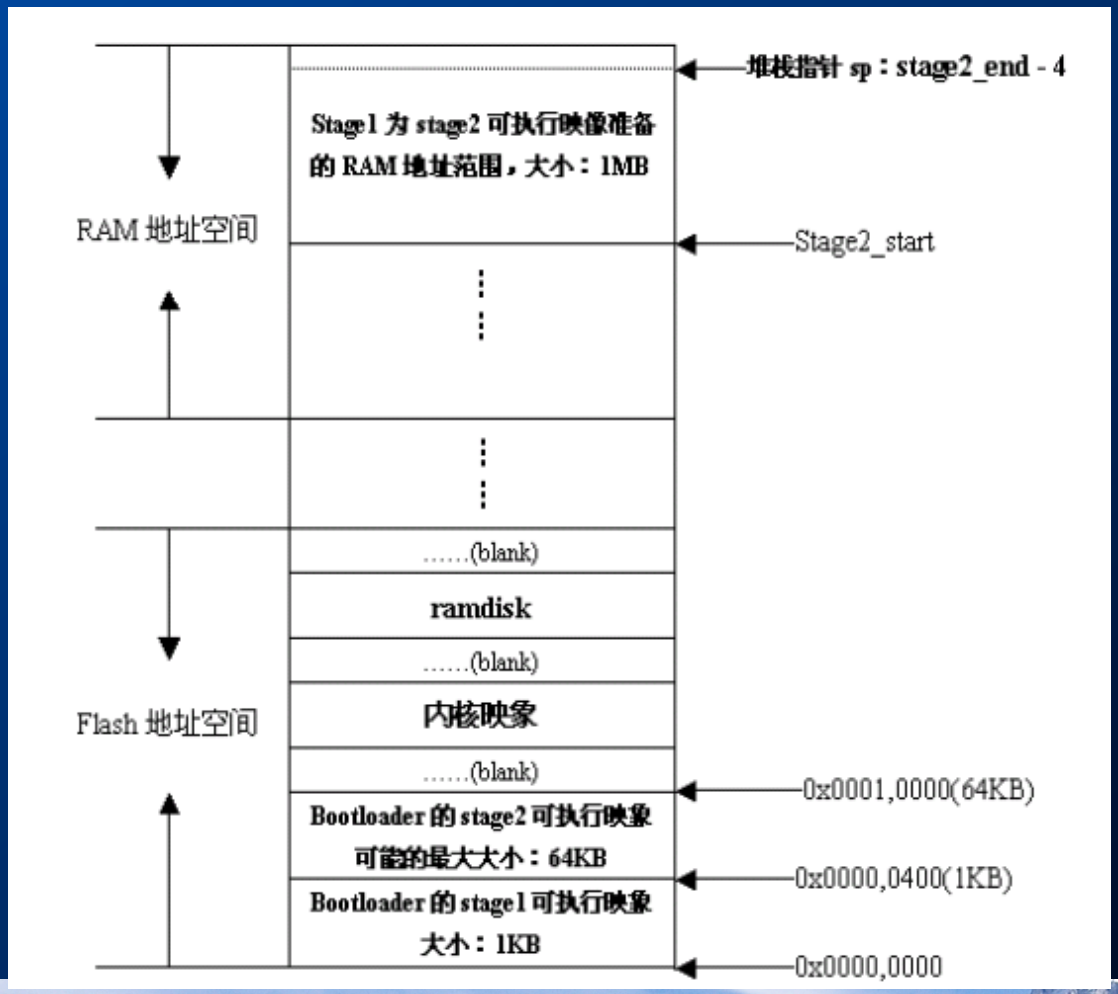
拷贝 stage2 到 RAM 中

- 拷贝时要确定两点
  - 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
  - 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，以提示用户我们准备跳转到 stage2

Boot Loader 的Stage2可执行映像被拷贝到内存后内存布局情况



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 执行路径显然再次回到我们的 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 directly
  - 1)无法传递函数参数；2)无法处理函数返回

Stage2 初始化本阶段要使用到的硬件设备

1. 初始化至少一个串口，以便终端用户进行 I/O 输出信息
2. 初始化计时器等

- 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

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

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

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

BootLoader的调试

- 从交叉调试的技术实现途径及应用场合看，分为：
  - 硬件调试：调试方便，价格昂贵
  - 源码软件调试：点灯或串口输出
    - 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

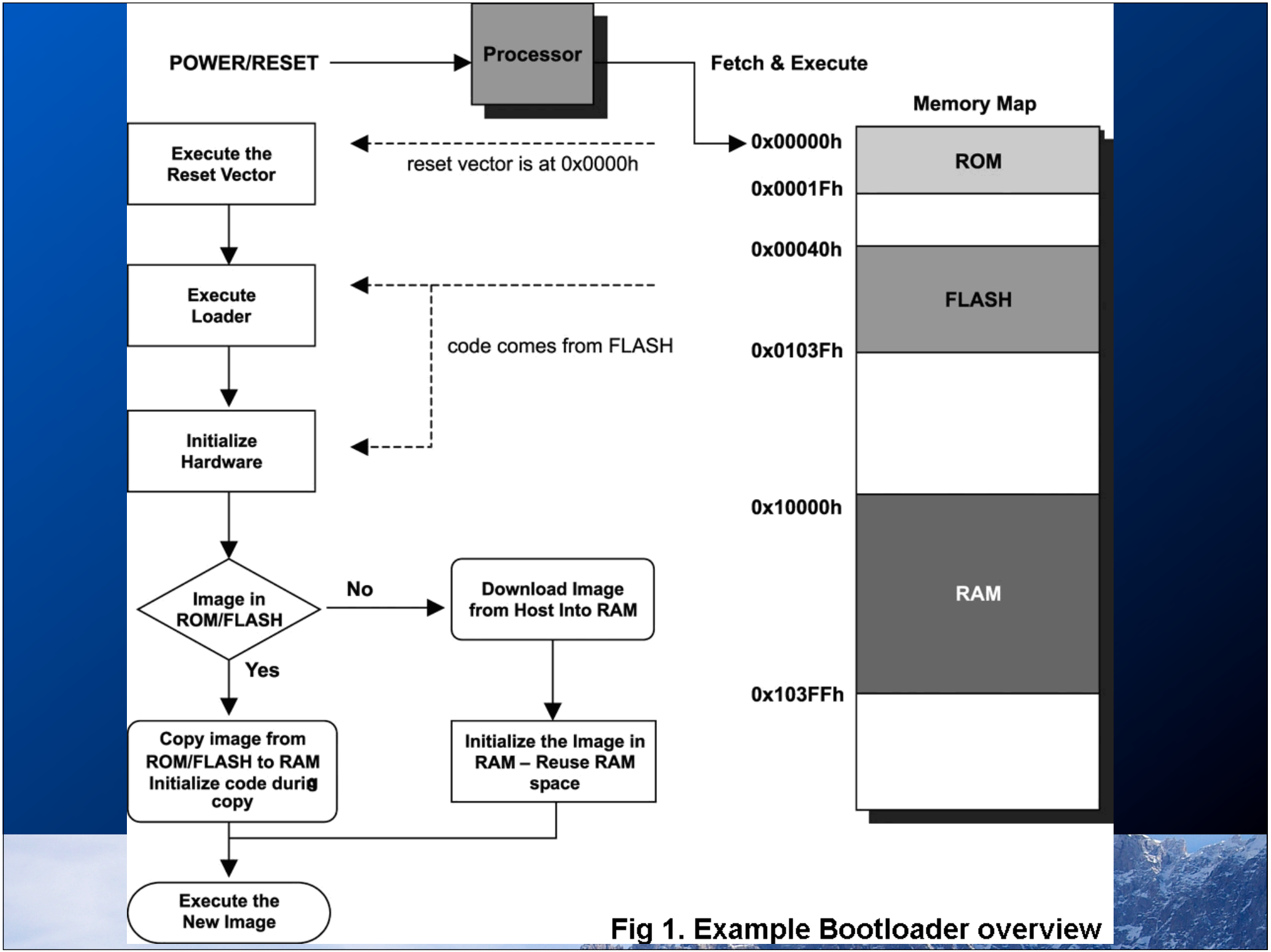
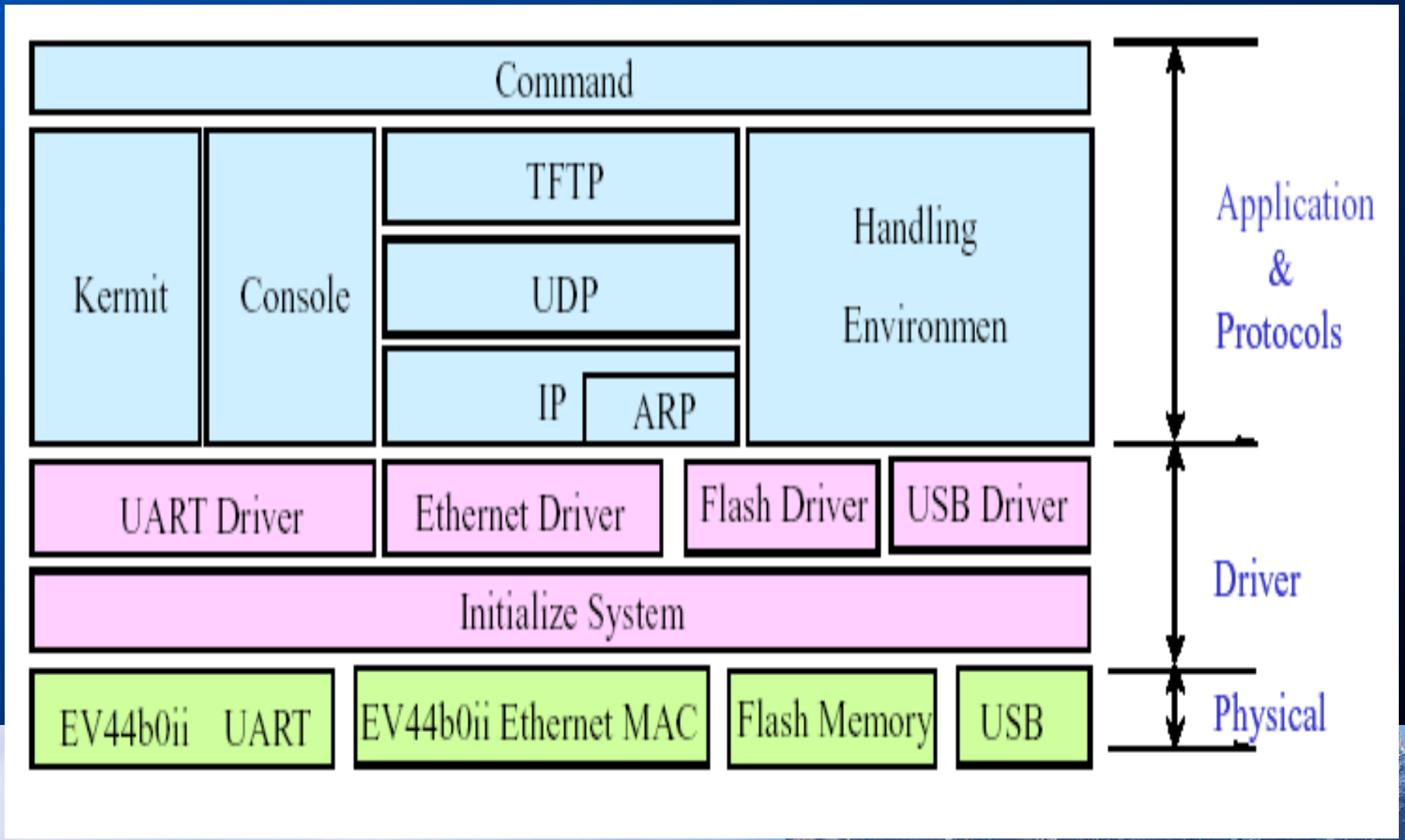


Fig 1. Example Bootloader overview

U-boot

- 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的体系结构


Command
Kermit
Console
TFTP
UDP
IP
ARP
Handling
Environmen
Application
&
Protocols
UART Driver
Ethernet Driver
Flash Driver
USB Driver
Initialize System
Driver
EV44b0ii UART
EV44b0ii Ethernet MAC
Flash Memory
USB
Physical

Redboot

- Redhat公司随eCos发布的一个BOOT方案
- 开源项目
- 支持
  - ARM，MIPS，PowerPC，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

vivi

- vivi是由韩国Mizi公司开发的一种Bootloader
- 适合于ARM9处理器
- 源代码可以在http://www.mizi.com网站下载

SportSummit2009

SPORTSUMMIT USER CONFERENCE

PARIS, FRANCE. CONFERENCE: SEPT. 18-20

Merci pour votre attention

Peter Keates

Fondateur, StartYourDoc
email votremail@votreadresse.com
Tél : +33 (0)1 01 01 01 01
www.startyourdoc.com