

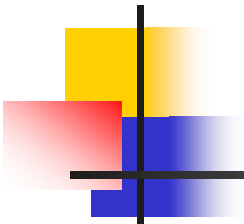
嵌入式Linux学习七步曲

Sailor_forever(扬帆)

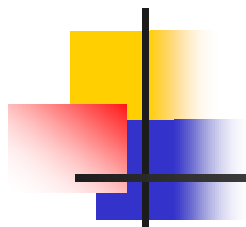
自由传播 版权所有 翻版必究



八一卦-我是who

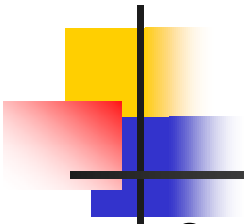
- 
-
- n 目前就职于通信行业某外企研发中心
 - n 参与校园招聘和社会招聘的技术面试工作
 - n 5年嵌入式软件开发经验，擅长嵌入式Linux开发；
 - n 接触的软硬件平台包括ARM，DSP，PowerPC，uC/OS-II，Linux，VxWorks及OSE

八一卦-我是who



- n 嵌入式Linux七步曲 学习群 交流讨论 资源共享
- n 群号 107900817
- n 7steps2linux@gmail.com
- n http://blog.csdn.net/sailor_8318

嵌入式水平小调查



n 0—3个月

n 3—6 个月

n 1年左右

n 2年以上

n 多少人参加过系列交流会？

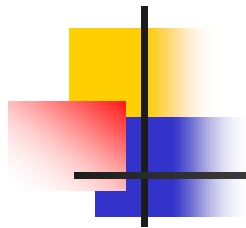


嵌入式Linux学习七步曲

- 
- 1 Linux主机开发环境
 - 2 嵌入式Linux交叉开发环境
 - 3 **Linux系统bootloader移植**
 - 4 Linux的内核移植
 - 5 Linux的内核及驱动编程
 - 6 文件系统制作
 - 7 Linux的高级应用编程

Grammy for Linux





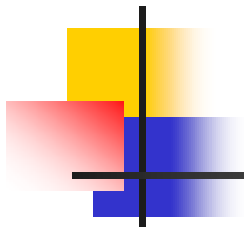
STAR Volunteer

- n 嵌入式Linux交叉开发环境
 - n 乔伟康, NFS服务器搭建
 - n 刘红保, TFTP 服务器搭建
 - n 卢佳孟, 驱动模块调试
 - n 王晓宇, GDB和GDB Server交叉调试



STAR Volunteer





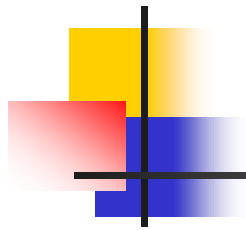
Volunteer Task

n 宗旨

- n 鼓励大家实际的参与嵌入式Linux的开发
- n 自己解决动手解决问题
- n 总结记录、分享
- n 形成知识库
- n **采用统一的模板**，争取成为系列交流会的特色项目
- n 扩大BUPT BES的影响力，创造品牌

n 运作

- n 下次交流会之前完成上次的总结文档
- n **Share**给大家，提建议意见
- n 每次交流会**颁奖鼓励**
- n 最终将评出 **STAR Volunteer**



Volunteer Task

n Logo

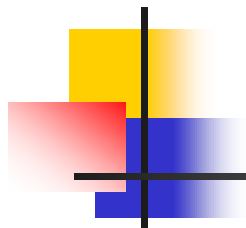
《嵌入式 Linux 学习七步曲》
BUPT BES 系列交流会 Volunteer Task



Key To Success

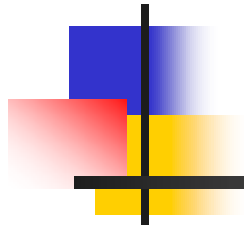
- n Google、Baidu
- n 理论 + 实践（开发板）
- n 勤于思考，善于总结
- n 多上相关技术论坛，他山之石可以攻玉
- n 良好的文档撰写习惯
- n Passion！





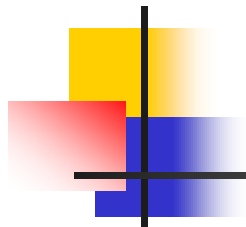
CHAPTER

3 Linux系统 bootloader移植



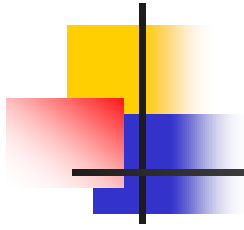
主要内容

- 
- 1 **Bootloader 基本介绍**
 - 2 **U-boot 介绍**
 - 3 **U-boot 移植过程**
 - 4 **U-boot 如何启动内核**

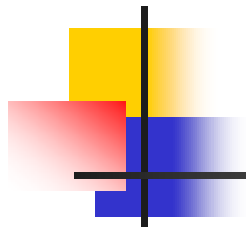


Bootloader基本介绍

- n 何谓Boot Loader
- n Boot Loader所支持的CPU和板级配置
- n Boot Loader的烧录和存储
- n Boot Loader的操作模式
- n Boot Loader与主机之间的传输协议
- n Boot Loader的通用执行流程



X86是如何启动的？



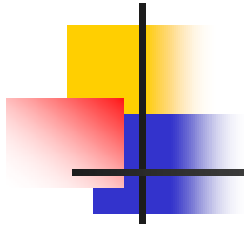
何谓Boot Loader

n Boot

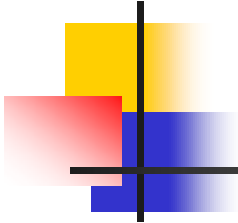
- n 严重依赖于硬件环境
- n CPU及板级资源的初始化
- n 为操作系统启动建立基本的环境

n Loader

- n 拷贝内核至RAM中
- n 检查校验甚至是解压缩
- n 跳转到内核入口
- n 完成使命



Bootloader群雄争霸？



Boot Loader所支持的CPU和板级配置

- n 每种不同的CPU体系结构都有特定的Boot Loader
- n 有些Boot Loader也支持多种体系结构的CPU
- n Boot Loader也依赖于板级配置

Bootloader	CPU	OS
Grub	X86	Linux, windows
LiLo	X86	Linux
U-booot	ARM, PowerPC, MIPS	Linux, VxWorks, PSOS
Redboot	ARM	eCos
ViVi	三星 ARM	Linux
BootRom	PowerPC	VxWorks
Blob	Intel PXA 系列	Linux



Boot Loader的烧录和存储

n 启动媒介

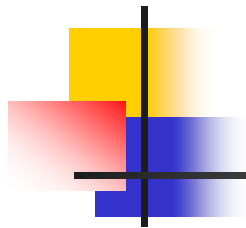
- n Nor Flash
- n Nand Flash
- n EEPROM
- n RAM
- n 内部ROM
- n 上电采样决定启动媒介

n 启动地址

- n ARM系列0
- n PowerPC系列大多数0或者0xfff00000及0xFFFF FFFC
- n MIPS 0x1FC0 0000

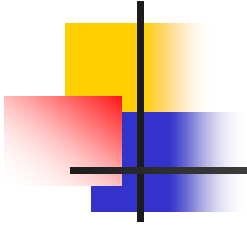
n 烧录方式

- n 固化的loader, 引导加载
- n JTAG仿真器
- n 外部controller



Boot Loader的操作模式

- n 主机和目标机之间一般通过串口建立连接
- n 下载**Downloading**模式
 - n 下载内核映像和根文件系统映像
 - n 暂存**RAM**中
 - n 最终存在**Flash**中
 - n 也可刷新**bootloader**本身
 - n 适用于开发阶段
- n 启动加载(**Boot loading**)模式
 - n 自主模式**bootstrap**
 - n 无用户干预
 - n 自动拷贝内核及文件系统
 - n 适用于产品发布阶段



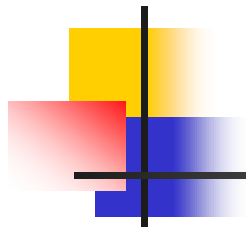
Boot Loader与主机之间的传输协议

n 串口

- n kermi / xmodem / ymodem
- n 速率有限
- n 适用于前期开发
- n 相关工具如hyperterminal, DNW, putty, SecureCRT, minicom, kermi

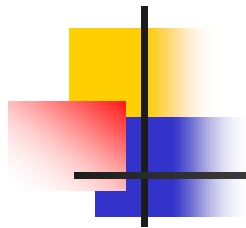
n 网口

- n TFTP (Trivial File Transfer Protocol)
- n NFS (下载, 启动及挂载)
- n 需要网络支持, 速率快



Boot Loader的通用执行流程

- n Stage1 + stage2划分原则
 - n **Flash + RAM**
 - n **汇编 + C**
 - n 体系结构相关 + 体系结构无关



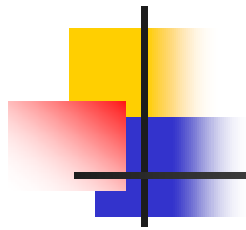
Boot Loader的通用执行流程

n Stage1特点

- n 汇编实现，短小精悍
- n 相对寻址，位置无关
- n Flash中运行

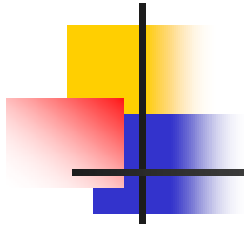
n Stage1 功能

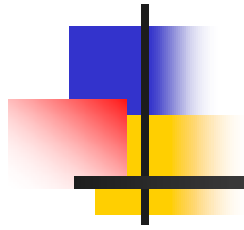
- n 复位向量
- n CPU核心寄存器初始化
- n **RAM**初始化，检查
- n 拷贝代码至RAM中
- n 设置堆栈
- n 跳转到C入口



Boot Loader的通用执行流程

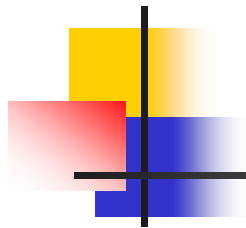
- n Stage2 特点
 - n RAM中运行，速度快
 - n C实现
 - n 更好的可读性和可移植性
- n Stage2 功能
 - n 初始化CPU非核心寄存器
 - n 相关外设初始化
 - n 等待用户输入
 - n 若无输入则自动运行bootcmd指定的命令，一般为拷贝内核和文件系统至RAM中，然后跳转到内核入口





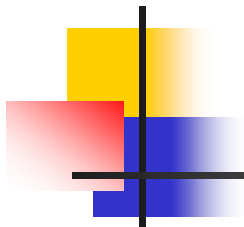
主要内容

- 
- 1 Bootloader 基本介绍
 - 2 **U-boot 介绍**
 - 3 U-boot 移植过程
 - 4 U-boot 如何启动内核



U-boot介绍

- n U-boot概述
- n U-boot源代码目录结构
- n U-boot支持的主要功能
- n U-boot命令介绍
- n U-boot环境变量



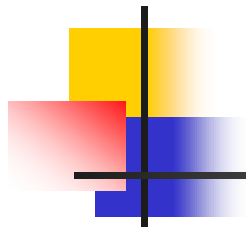
U-boot概述

- n 开源bootloader
- n 支持多种CPU和OS，应用最为广泛
- n 研究者众多，方便易用。就业方面非常有优势
- n 可配置可裁剪
- n 可移植性非常好
- n 代码架构值得学习借鉴



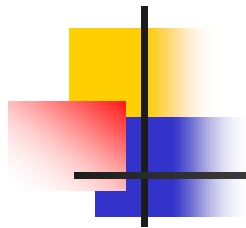
U-boot源代码目录结构

- |-- **board** 平台依赖，存放板级相关的目录文件
- |-- **common** 通用多功能函数的实现
- |-- **cpu** 平台依赖，存放 **cpu** 相关的目录文件
- |-- **disk** 通用。硬盘接口程序
- |-- **doc** 文档
- |-- **drivers** 通用的设备驱动程序，如以太网接口驱动
- |-- **dtb**
- |-- **examples** 应用例子
- |-- **fs** 通用存放文件系统的程序
- |-- **include** 头文件和开发板配置文件，所有开发板配置文件放在其 **configs** 里
- |-- **lib_arm** 平台依赖，存放 **arm** 架构通用文件
- |-- **lib_generic** 通用的库函数
- |-- **lib_ppc** 平台依赖，存放 **ppc** 架构通用文件
- |-- **net** 存放网络的程序
- |-- **post** 存放上电自检程序
- |-- **rtc** rtc 的驱动程序
- |-- **tools** 工具



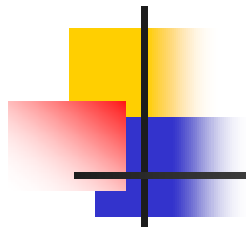
U-boot支持的主要功能

- n 支持多种CPU和OS
- n 多种启动类型及启动参数
- n 环境变量多种存储方式
- n 多种通信下载机制
- n 各种系统命令及用户自定义命令
- n 脚本执行
- n 常见的设备驱动
- n 存储器自动探测检查
- n 各种上电自己程序



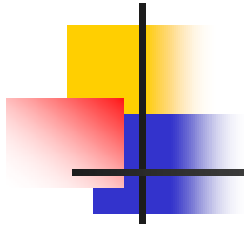
U-boot命令介绍

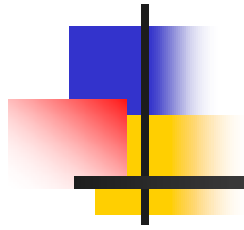
- n 板级配置信息
 - n Bdinfo, flinfo
- n 存储器相关
 - n Cp, md, cmp, mw
- n Flash擦写
 - n Cp, erase, protect
- n 环境变量
 - n Printenv, saveenv , setenv , run
- n 下载启动
 - n loadb, nfs , tftp , dhcp, ping
 - n Bootm, go



U-boot环境变量

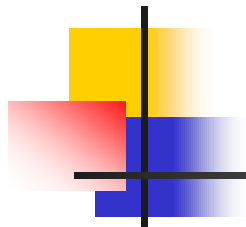
- n 存储位置
 - n Flash, ram, nvram, eeprom
- n 板级配置信息
 - n Baudrate, ethaddr
- n 启动行为
 - n Bootdelay, bootcmd, bootargs
- n 网络参数
 - n ipaddr, serverip, gatewayip, dnsip, netmask, hostname, rootpath, bootfile
- n 下载
 - n Loadaddr, filesize, fileaddr
- n 用户自定义
 - n 组合命令, 脚本等等





主要内容

- 
- 1 Bootloader 基本介绍
 - 2 U-boot 介绍
 - 3 **U-boot 移植过程**
 - 4 U-boot 如何启动内核



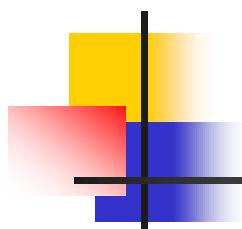
U-boot移植过程

- n 移植的类型、级别
- n 移植相关文件
- n 输入信息为什
- n 移植过程
- n U-boot启动流程



移植的类型、级别

- n 全新的CPU架构
 - n Almost impossible just with your hand
- n 新的CPU类型
 - n 可参照现有的CPU类型，稍简单
- n 新的板级配置
 - n 最常见的移植类型
 - n 参照现有的同CPU的板子
 - n 厂家提供demo板
 - n 输入为HW DS



移植相关文件

- |-- **board** 平台依赖，存放板级相关的目录文件
- |-- **common** 通用多功能函数的实现
- |-- **cpu** 平台依赖，存放 **cpu** 相关的目录文件
- |-- **drivers** 通用的设备驱动程序，如以太网接口驱动
- |-- **include** 头文件和开发板配置文件，所有开发板配置文件放在其 **configs** 里
- |-- **post** 存放上电自检程序

```
sea@sea-dev:~/linux_dev/u-boot-1.3.3/board/smdk2410$ tree -L 1
```

```
.  
|-- Makefile  
|-- config.mk  
|-- flash.c  
|-- lowlevel_init.S  
|-- smdk2410.c  
`-- u-boot.lds
```



移植相关文件

```
sea@sea-dev:~/linux_dev/u-boot-1.3.3/cpu/arm920t/s3c24x0$ tree -L 1
```

```
.  
|-- Makefile  
|-- i2c.c  
|-- interrupts.c  
|-- nand.c  
|-- serial.c  
|-- speed.c  
|-- usb.c  
|-- usb_ohci.c  
`-- usb_ohci.h
```

```
sea@sea-dev:~/linux_dev/u-boot-1.3.3/cpu/arm920t$ tree -L 1
```

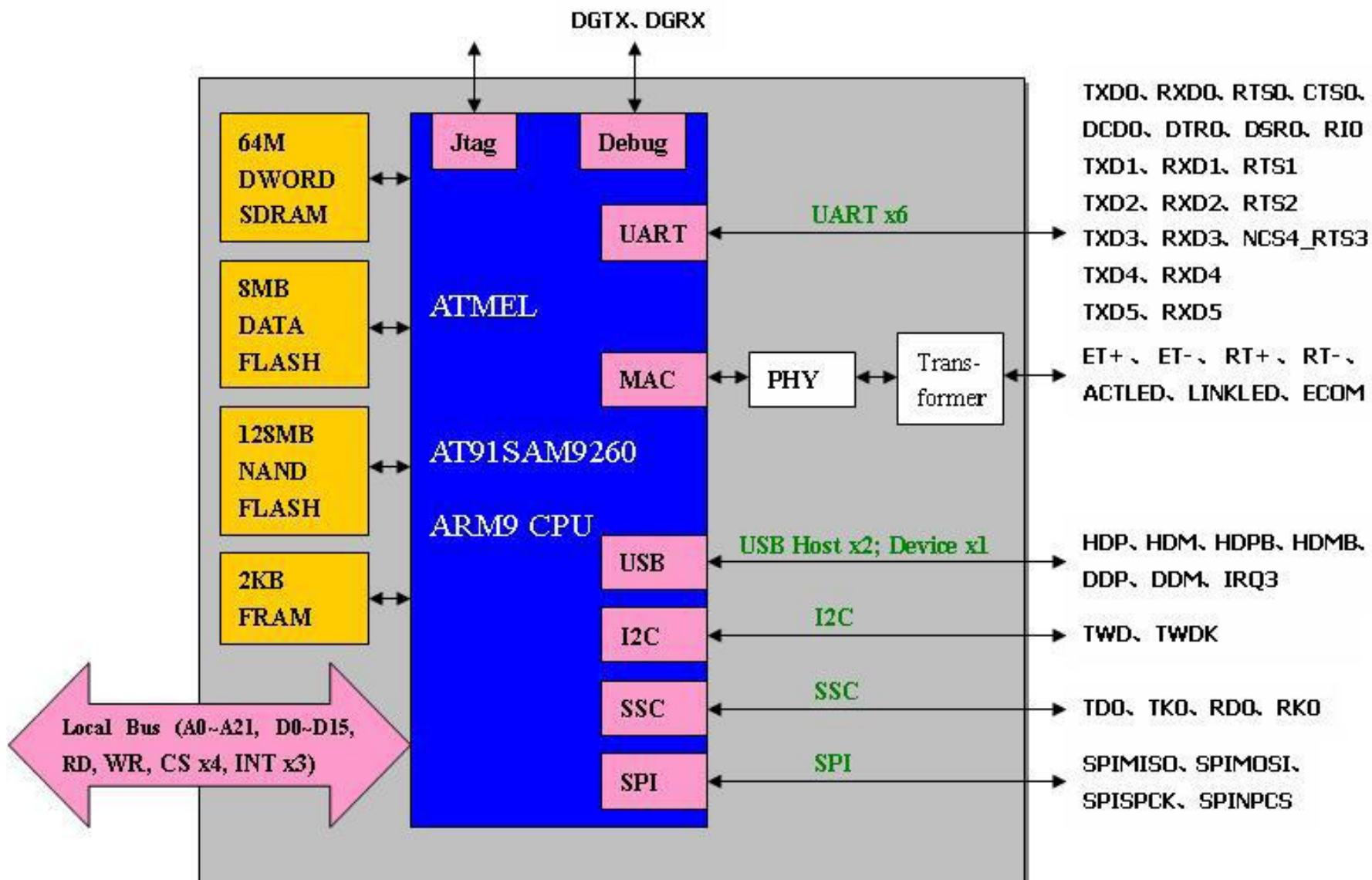
```
.  
|-- Makefile  
|-- config.mk  
|-- cpu.c  
|-- interrupts.c  
|-- s3c24x0  
`-- start.S
```



输入信息为什

- n CPU
- n 时钟
- n 启动方式
- n memory
 - n Nor Flash
 - n Nand Flash
 - n SDRAM
 - n 大小
 - n 位宽
- n IIC EEPROM, RTC
- n 波特率
- n 哪个串口
- n 哪个网口

输入信息为什

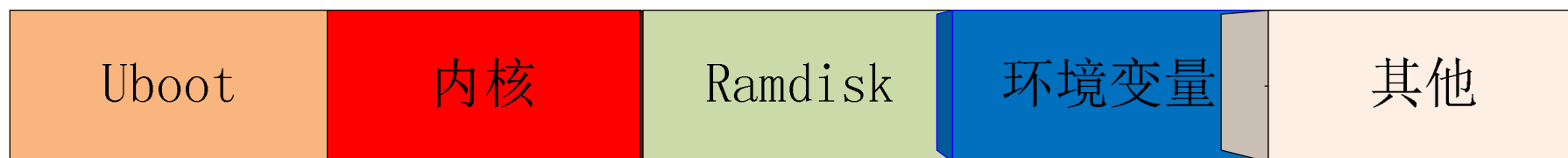


输入信息为什

n SDRAM内存布局



n Flash内存布局





移植过程

n CPU

- n 对应的ARM架构下创建CPU目录，如\cpu\arm920t\cpu_type，拷贝参考CPU的文件或者自己创建

n board

- n 在\board目录下创建对应的board目录，如\board\board-type，拷贝参考CPU的文件或者自己创建
- n 修改config.mk中的text_base链接地址
- n 修改flash驱动、SDRAM驱动，及board-type.c中的相关驱动
- n 修改链接脚本u-boot.lds中的start.s文件的路径

n Config头文件

- n 在\include\configs下创建对应的头文件board-type.h
- n 根据板子配置修改相关宏定义

n 主makefile

- n board-type_config : unconfig
- n @\$(MKCONFIG) \$(@:_config=) arm arm920t board-type
NULL s3c24x0

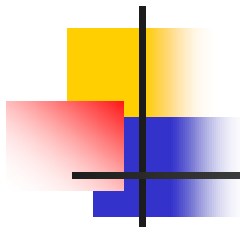


移植过程

n \board\board-type\u-boot.lds

```
OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
/*OUTPUT_FORMAT("elf32-arm", "elf32-arm", "elf32-arm")*/
OUTPUT_ARCH(arm)
ENTRY(_start)
SECTIONS
{
    . = 0x00000000;

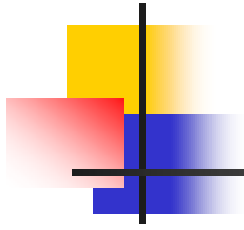
    . = ALIGN(4);
    .text :
    {
        cpu/arm920t/start.o (.text)
        *(.text)
    }
}
```



移植过程

n \board\board-type\u-boot.lds

```
. = ALIGN(4);  
.rodata : { *(.rodata) }  
. = ALIGN(4);  
.data : { *(.data) }  
. = ALIGN(4);  
.got : { *(.got) }  
  
. = .;  
__u_boot_cmd_start = .;  
.u_boot_cmd : { *(.u_boot_cmd) }  
__u_boot_cmd_end = .;  
. = ALIGN(4);  
__bss_start = .;  
.bss (NOLOAD) : { *(.bss) }  
_end = .;  
}
```



移植过程

- n Make distclean
- n Make board-type_config
- n Make depend
- n make



U-boot的启动流程-stage1

n \cpu\arm920t\start.s

n 复位

```
.globl _start
_start: b start_code
        ldr pc, _undefined_instruction
        ldr pc, _software_interrupt
        ldr pc, _prefetch_abort
        ldr pc, _data_abort
        ldr pc, _not_used
        ldr pc, _irq
        ldr pc, _fiq

_undefined_instruction: .word undefined_instruction
_software_interrupt:   .word software_interrupt
_prefetch_abort:      .word prefetch_abort
_data_abort:          .word data_abort
_not_used:             .word not_used
_irq:                 .word irq
_fiq:                 .word fiq
```



U-boot的启动流程-stage1

n 核心初始化

n MMU, SDRAM, 主频

```
start_code:
/*
 * set the cpu to SVC32 mode
 */
mrs    r0,cpsr
bic    r0,r0,#0x1f
orr    r0,r0,#0xd3
msr    cpsr,r0

/*
 * we do sys-critical inits only at reboot,
 * not when booting from ram!
 */
#ifdef CONFIG_SKIP_LOWLEVEL_INIT
    bl    cpu_init_crit
#endif
```

U-boot的启动流程-stage1

n 代码重定位

```
#ifndef CONFIG_AT91RM9200
#ifndef CONFIG_SKIP_RELOCATE_UBOOT
relocate:                                /* relocate U-Boot to RAM */
    adr    r0, _start                    /* r0 <- current position of code */
    ldr    r1, _TEXT_BASE                /* test if we run from flash or RAM */
    cmp    r0, r1                        /* don't reloc during debug */
    beq    stack_setup

    ldr    r2, _armboot_start
    ldr    r3, _bss_start
    sub    r2, r3, r2                    /* r2 <- size of armboot */
    add    r2, r0, r2                    /* r2 <- source end address */

copy_loop:
    ldmbia r0!, {r3-r10}                /* copy from source address [r0] */
    stmbia r1!, {r3-r10}                /* copy to target address [r1] */
    cmp    r0, r2                        /* until source end addreee [r2] */
    ble    copy_loop
#endif /* CONFIG_SKIP_RELOCATE_UBOOT */
#endif
```


U-boot的启动流程-stage1

n 建立堆栈、清除BSS、准备跳转至RAM

```
/* Set up the stack */
stack_setup:
    ldr    r0, _TEXT_BASE          /* upper 128 KiB: relocated uboot */
    sub    r0, r0, #CFG_MALLOC_LEN /* malloc area */
    sub    r0, r0, #CFG_GBL_DATA_SIZE /* bdfinfo */
    sub    sp, r0, #12             /* leave 3 words for abort-stack */

clear_bss:
    ldr    r0, bss_start          /* find start of bss segment */
    ldr    r1, bss_end            /* stop here */
    mov    r2, #0x00000000         /* clear */

clbss_1: str    r2, [r0]            /* clear loop... */
    add    r0, r0, #4
    cmp    r0, r1
    ble    clbss_1

    ldr    pc, start_armboot
start_armboot:    .word start_armboot
```



U-boot的启动流程-stage2

n lib_arm/board.c start_armboot

n 初始化序列

```
for (init_fnc_ptr = init_sequence; *init_fnc_ptr; ++init_fnc_ptr) {  
    if ((*init_fnc_ptr)() != 0) {  
        hang();  
    }  
}
```

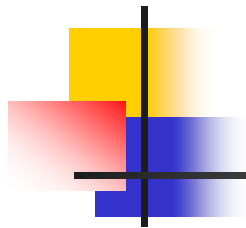
```
init_fnc_t *init_sequence[] = {  
    cpu_init, /* basic cpu dependent setup */  
#if defined(CONFIG_SKIP_RELOCATE_UBOOT)  
    reloc_init, /* Set the relocation done flag, must  
                do this AFTER cpu_init(), but as soon  
                as possible */  
#endif  
    board_init, /* basic board dependent setup */  
    interrupt_init, /* set up exceptions */  
    env_init, /* initialize environment */  
};
```



U-boot的启动流程-stage2

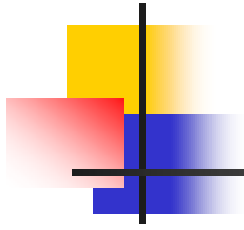
n 初始化序列

```
init_baudrate,          /* initialize baudrate settings */  
serial_init,           /* serial communications setup */  
console_init_f,        /* stage 1 init of console */  
display_banner,        /* say that we are here */  
#if defined(CONFIG_DISPLAY_CPUINFO)  
    print_cpuinfo,        /* display cpu info (and speed) */  
#endif  
#if defined(CONFIG_DISPLAY_BOARDINFO)  
    checkboard,          /* display board info */  
#endif  
#if defined(CONFIG_HARD_I2C) || defined(CONFIG_SOFT_I2C)  
    init_func_i2c,  
#endif  
    dram_init,           /* configure available RAM banks */  
    display_dram_config,  
    NULL,  
};
```



U-boot的启动流程-stage2

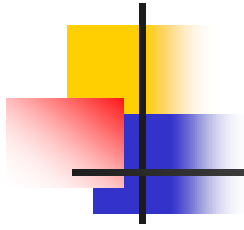
- n 特定的初始化
 - n flash_init
 - n mem_malloc_init
 - n nand_init
 - n env_relocate
 - n enable_interrupts
 - n eth_initialize
- n 主循环
 - n main_loop
 - n common/main.c



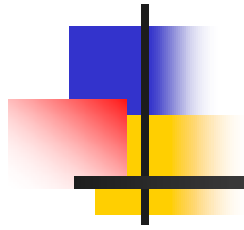
Volunteer task

- n 分析U-boot在ARM平台下的启动流程
- n 要求
 - n 说明U-boot版本和ARM处理器类型
 - n 从反汇编的角度解释关键跳转点和重定位代码
 - n 关键代码需要给出注释
 - n 最好有一个总的流程图

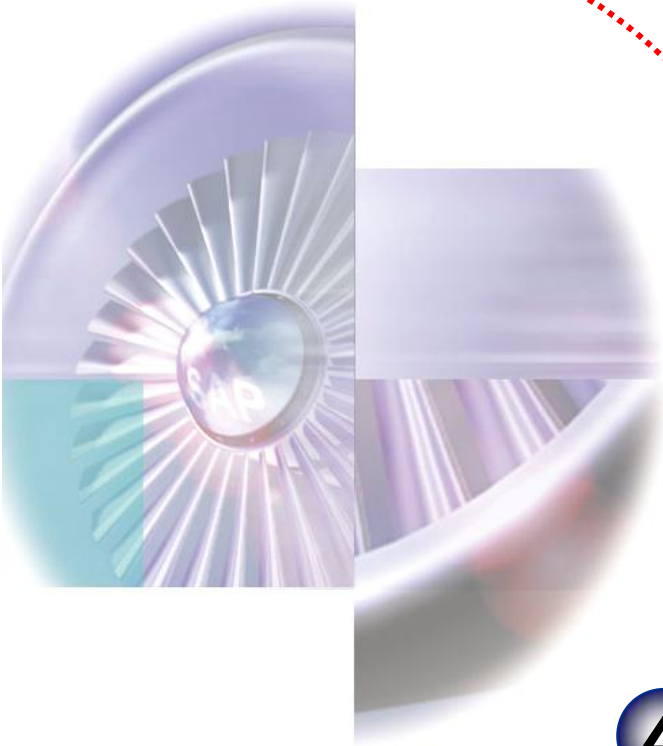
Self Learning, summary, share!

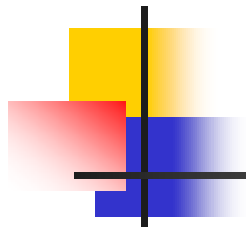


Q & A ?



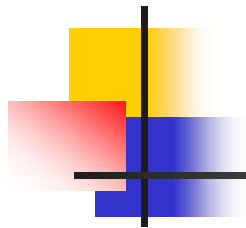
主要内容

- 
- 1 Bootloader 基本介绍
 - 2 U-boot 介绍
 - 3 U-boot 移植过程
 - 4 U-boot 如何启动内核



U-boot如何启动内核

- n 启动命令
- n 启动参数
- n Mkimage工具
- n Bootm运行流程
- n 如何制作ulmage映像



启动命令

n Go

- n Common\cmd_boot.c

- n 需要定制调整

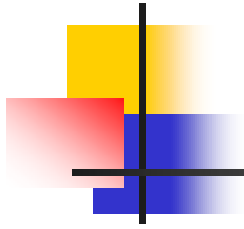
- n 添加内核启动入口参数及命令行参数

n bootm

- n 标准的u-boot启动命令

- n 支持多种类型的映像

- n 支持解压缩及**CRC**校验



启动参数

n ramdisk

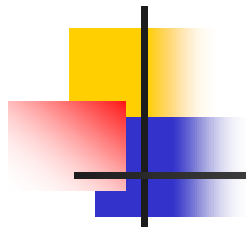
n root=/dev/ram init=/linuxrc
initrd=0x21000000,0x00500000
console=ttySAC0,115200

n Flash

n root=/dev/mtdblockx noinitrd
console=ttySAC0,115200

n nfs

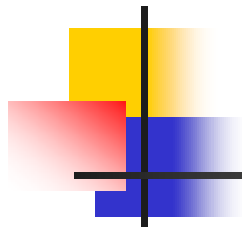
n root=/dev/nfs rw
nfsroot=\$serverip:\$root_path
ip=\$ipaddr:\$serverip:\$gatewayip:\$netmask:
\$hostname:\$netdev:off
console=ttySAC0,115200



Mkimage工具

- n Image header
- n Toos\mkimage.c生成的工具

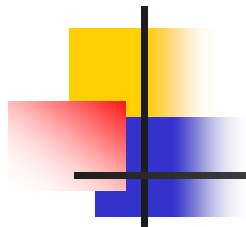
```
typedef struct image_header {  
    uint32_t ih_magic; /* Image Header Magic Number */  
    uint32_t ih_hcrc; /* Image Header CRC Checksum */  
    uint32_t ih_time; /* Image Creation Timestamp */  
    uint32_t ih_size; /* Image Data Size */  
    uint32_t ih_load; /* Data Load Address */  
    uint32_t ih_ep; /* Entry Point Address */  
    uint32_t ih_dcrc; /* Image Data CRC Checksum */  
    uint8_t ih_os; /* Operating System */  
    uint8_t ih_arch; /* CPU architecture */  
    uint8_t ih_type; /* Image Type */  
    uint8_t ih_comp; /* Compression Type */  
    uint8_t ih_name[IH_NMLEN]; /* Image Name */  
} image_header_t;
```



Mkimage工具

n 参数意义

- A ==> set architecture to 'arch'
- O ==> set operating system to 'os'
- T ==> set image type to 'type' “kemel 或是 ramdisk”
- C ==> set compression type 'comp'
- a ==> set load address to 'addr' (hex)
- e ==> set entry point to 'ep' (hex) (此为内核启动时的跳转入口)
- n ==> set image name to 'name'
- d ==> use image data from 'datafile'
- x ==> set XIP (execute in place, 即不进行文件的拷贝, 在当前位置执行)



Mkimage工具

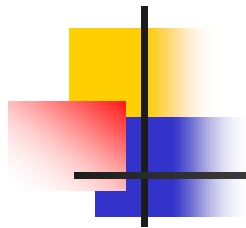
n ARM实例

- A arm ----- 架构是 arm
- O linux ----- 操作系统是 linux
- T kernel ----- 类型是 kernel
- C none/bzip/gzip ----- 压缩类型
- a 20008000 ---- image 的载入地址(hex), 通常为 0xX00008000
- e 200080XX---- 内核的入口地址(hex), XX 为 0x40 或者 0x00
- n linux-XXX --- image 的名字, 任意
- d nameXXX --- 无头信息的 image 文件名, 你的源内核文件
- uImageXXX ---- 加了头信息之后的 image 文件名, 任意取



Bootm运行流程

- n 压缩是指编译出来的内核在mkimage处理前是否经过压缩，而非内核自身是否解压缩
- n Bootm loadaddr
- n 非压缩
 - n -c = none
 - n Loadaddr = -a，无需搬动，但-e = -a + 0x40
 - n Loadaddr != -a，把去掉header后的内核拷贝到-a指定的地址，此时-e = -a
- n 压缩内核
 - n -c = gzip等
 - n 需要解压缩
 - n Loadaddr != -a， Loadaddr = -a + maxsizeof(kernel)
 - n -a = -e



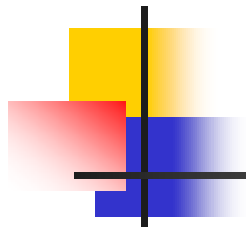
Mkimage如何制作内核映像

n 影响因子

- n -e, 内核的入口地址是否与-a相同
- n Loadaddr, 即将内核加载到RAM中运行的地址, 决定是否搬运或解压内核
- n c, 内核是否经过gzip压缩过, 决定了是搬运还是解压
- n 内核本身为非压缩的Image或zImage

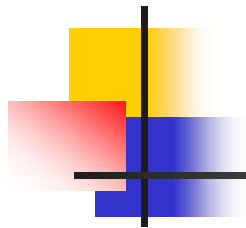
n 排列组合

- n $2^4 = 16$ 种



非压缩的Image

- n ramdiskaddr= 0x21100000 统一
- n -a=-e = 0x20008000 , -c=none,
loadaddr= 0x20f00000
 - n 此法主要由于内核太大, 导致loadaddr做了一定的修正
- n -a= 0x20008000 , -e = 0x20008040, -
c=none, loadaddr=0x20008000
 - n 此法无需拷贝, 看样子可以, 但非压缩内核对启动入口地址有对齐要求
 - n 对于非压缩的Image内核, mkimage之前不压缩的话, 内核印象较大, 此法不常用



非压缩的Image

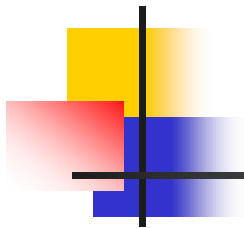
n -a=-e = 0x20008000 , -c=gzip,
loadaddr= 0x21000000

n -c=gzip压缩内核必须解压，只有这种情况成功；其他解压覆盖或者-e入口不对



压缩的zImage内核

- n ramdiskaddr= 0x21100000 统一
- n -a=-e = 0x20008000 , -c=none,
loadaddr= 0x21000000
- n -a= 0x20008000 , -e = 0x20008040, -
c=none, loadaddr=0x20008000
- n -a=-e = 0x20008000 , -c=gzip,
loadaddr= 0x21000000
 - n -c=gzip压缩内核必须解压, 只有这种情况成功; 其他解压覆盖或者-e入口不对
 - n zImage已经压缩过一次了, 一般无需再压缩, 此法不常用



最优解

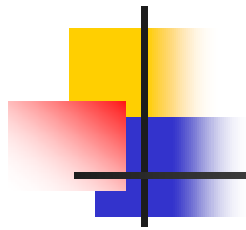
n 非压缩的Image内核

n -a=-e = 0x20008000 , -c=gzip, loadaddr=0x21000000

n 压缩的zImage内核

n a=-e = 0x20008000 , -c=none, loadaddr=0x21000000

n -a= 0x20008000 , -e = 0x20008040, -c=none, loadaddr=0x20008000



Volunteer task

- n U-boot如何启动ARM Linux内核
- n 要求
 - n 给出mkimage工具的使用方法
 - n 分析bootm启动ARM Linux内核的流程，最好有流程图
 - n 分析mkimage参数的各种关系，给出16种方案的测试实例

Self Learning, summary, share!

