华清远见
FARSIGHT
嵌入式培训专家

嵌入式Linux移植和Uboot

华清远见：曾宏安

内容提纲

- Bootloader
  - Bootloader的作用
  - 常用Bootloader介绍
- U-boot
  - U-boot 介绍
  - U-boot 启动流程及代码分析
  - 在U-boot 中添加命令

什么是Bootloader

- Bootloader是硬件启动时执行的引导程序，是运行操作系统的前提；
- 是在操作系统内核或用户应用程序运行之前运行的一段代码。
- 在嵌入式系统中，整个系统的初始化和加载任务一般由Bootloader来完成。

Bootloader的特点

- Bootloader运行通常分为两个阶段。
- Bootloader独立于操作系统。
- Bootloader不仅依赖于CPU的体系结构，而且依赖于嵌入式系统板级设备的配置。

Bootloader的执行模式

- **自启动模式：**该模式下，Bootloader按照预先设定的命令自动运行，整个过程并没有用户的介入。

- **交互模式：**改模式下，Bootloader通过串口和PC端通信，接收用户的命令。

一些bootloader介绍

Bootloader	Monitor	描述	X86	ARM	PowerPC
LILO	否	Linux磁盘引导程序	是	否	否
GRUB	否	GNU的LILO替代程序	是	否	否
Loading	否	从DOS引导linux	是	否	否
ROLO	否	从ROM引导linux而不需要BOIS	是	否	否
Etherboot	否	通过以太网卡启动linux系统的固件	是	否	否
LinuxBOIS	否	完全替代BUIS的linux引导程序	是	否	否
BLOB	是	LART等硬件平台的引导程序	否	是	否
U-boot	是	通用引导程序	是	是	是
RedBoot	是	基于eCos的引导程序	是	是	是
Vivi	是	Mizi公司针对于三星的arm cpu设计的引导程序			

ARM Bootloaders

- U-Boot是ARM bootloader标准
  - armboot 加入到 ppcboot 形成了 u-boot
  - 支持arm720, arm920, arm926, CortexA8, sa1100, xscale
  - http://armboot.sourceforge.net/
- Blob
  - blob是LART工程使用的bootloader
  - 移植到多个ARM平台上
  - http://www.lart.tudelft.nl/lartware/blob
- Redboot

内容提纲

- Bootloader
  - Bootloader的作用
  - 常用Bootloader介绍
- U-boot
  - U-boot 介绍
  - U-boot 启动流程及代码分析
  - 在U-boot 中添加命令

U-boot介绍

- u-boot(Universal Boot Loader)是德国DENX小组开发的用于多种嵌入式CPU的bootloader程序。遵循GPL条款。

- 8xxROM 、PPCBOOT、Armboot逐步发展演化而来；

- U-boot的特点
  - **代码结构清晰、易于移植（见程序结构）**
  - **支持多种处理器体系结构（见程序结构cpu目录）**
  - **支持众多参考板（目前官方包中有200多种，见程序结构board目录）**
  - **命令丰富、有监控功能**
  - **支持网络协议、USB、SD等多种协议和设备**
  - **支持文件系统**
  - **更新较活跃，使用者多，有助于解决问题**

U-boot介绍

半导体厂商		嵌入式处理器		参考板
Samsung	→	**S3C2410**	→	**SMDK2410**

uboot —移植→ **FS2410**

U-Boot目录结构

- 平台相关
  - board, cpu, lib_arm, include…
- 平台无关
  - common, net, fs, drivers…
- 工具和文档
  - tools, doc

U-Boot目录结构

board	Board dependent files, RPXlite(mpc8xx), smdk2410(arm920t), sc520_cdp(x86) …
cpu	CPU specific files, mpc8xx, ppc4xx, arm920t, cortexA8, xscale, i386
lib_ppc	Files generic to PowerPC architecture
lib_arm	Files generic to ARM architecture
lib_i386	Files generic to X86 architecture
include	Header Files and board configs

U-Boot目录结构

common	Misc functions
lib_generic	Generic library functions
net	Networking code
fs	File System Code
post	Power On Self Test
drivers	Common used device drivers
disk	Hard disk interface code
rtc	Real Time Clock drivers
examples	Example code

U-boot命令介绍

- 命令分类
  - 环境设置、数据传输、存贮器操作及其他

- **printenv** 打印环境变量。
  - Uboot> printenv

    baudrate=115200

    ipaddr=192.168.1.1

    ethaddr=12:34:56:78:9A:BC

    serverip=192.168.1.5

U-boot命令介绍

- **setenv** 设置新的变量
  - Uboot> setenv myboard FS2410
    Uboot> printenv
    baudrate=115200
    ipaddr=192.168.1.1
    ethaddr=12:34:56:78:9A:BC
    serverip=192.168.1.5
    myboard=FS2410
    Environment size: 102/8188 bytes
- **saveenv**
  - 将当前定义的所有变量的值存入flash中。

U-boot命令介绍

- **tftp** 通过网络下载程序
  - Uboot> tftp 33000000 zImage

    Uboot> go 33000000
- **loadb** 通过串口Kermit协议下载二进制数据。

U-boot命令介绍

- md 显示内存区的内容。
- mm 修改内存，地址自动递增。
- nm 修改内存，地址不自动递增。
- mw 填充内存。
- mtest 测试内存。
- cp 拷贝一块内存到另一块。
- cmp 比较两块内存区。

- mw 0x32000000 ff 0x10000

U-boot命令介绍

- protect 写保护操作
  - protect on 1:0-3(就是对第一块FLASH的0-3扇区进行保护)
  - protect off 1:0-3取消写保护

- erase 擦除扇区
  - erase: 删除FLASH的扇区
  - erase 1:0-2(就是对每一块FLASH的0-2扇区进行删除)

U-boot命令介绍

- nand info: 显示NAND 设备
- nand device [dev]:显示或设置当前设备
- nand bad – 显示坏块
- nand read[.jffs2[s]] addr off size
- nand write[.jffs2] addr off size
- nand erase [clean] [off size]
- nand read.oob addr off size
- nand write.oob addr off size

U-boot命令介绍

- go 执行内存中的二进制代码，一个简单的跳转到指定地址

- bootm 执行内存中的二进制代码
  - 要求二进制代码为指定格式的。通常为mkimage处理过的二进制文件。

- bootp 通过网络启动，需要提前设置好硬件地址。

U-Boot 启动过程介绍

- 第一阶段代码在Flash中运行，用汇编实现
  - 设置CPSR、关闭看门狗、屏蔽中断
  - 关MMU、关闭I/D cache、内存初始化
  - 自搬运、设置堆栈、清空BSS段

- cpu/arm920t/start.S

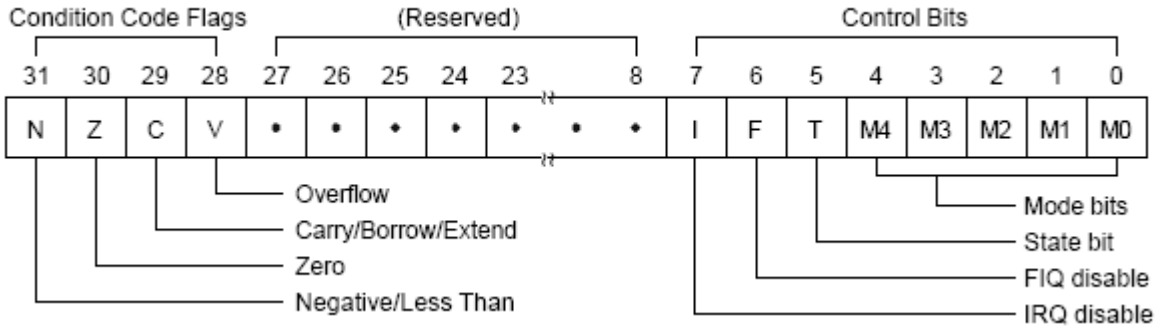
cpu/arm920t/start.S

- 异常向量表

```
.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
```

复位

```
start_code:
    mrs  r0, cpsr
    bic  r0, r0, #0x1f    ; 位清零
    orr  r0, r0, #0xd3    ; 逻辑或0xd3= 1101 0011
    msr  cpsr, r0
```



模式位含义

M[4:0]	Mode	Visible THUMB state registers	Visible ARM state registers
10000	User	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR
10001	FIQ	R7..R0, LR_fiq, SP_fiq PC, CPSR, SPSR_fiq	R7..R0, R14_fiq..R8_fiq, PC, CPSR, SPSR_fiq
10010	IRQ	R7..R0, LR_irq, SP_irq PC, CPSR, SPSR_irq	R12..R0, R14_irq, R13_irq, PC, CPSR, SPSR_irq
10011	Supervisor	R7..R0, LR_svc, SP_svc, PC, CPSR, SPSR_svc	R12..R0, R14_svc, R13_svc, PC, CPSR, SPSR_svc
10111	Abort	R7..R0, LR_abt, SP_abt, PC, CPSR, SPSR_abt	R12..R0, R14_abt, R13_abt, PC, CPSR, SPSR_abt
11011	Undefined	R7..R0 LR_und, SP_und, PC, CPSR, SPSR_und	R12..R0, R14_und, R13_und, PC, CPSR
11111	System	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR

关闭看门狗

```
ldr     r0, =pWTCON
mov     r1, #0x0
str     r1, [r0]
```

Register	Address	R/W	Description	Reset Value
WTCON	0x53000000	R/W	Watchdog timer control register	0x8021

关闭中断

```
mov   r1, #0xffffffff
ldr   r0,  =INTMSK
str   r1,  [r0]
```

Register	Address	R/W	Description	Reset Value
INTMSK	0X4A000008	R/W	Determine which interrupt source is masked. The masked interrupt source will not be serviced. 0 = Interrupt service is available. 1 = Interrupt service is masked.	0xFFFFFFFF

刷新指令/数据缓存

```
mov   r0, #0
mcr   p15, 0, r0, c7, c7, 0  /* flush v3/v4 cache */
mcr   p15, 0, r0, c8, c7, 0  /* flush v4 TLB */
```

Function	Data	Instruction
Invalidate ICache & DCache	SBZ	MCR p15,0,Rd,c7,c7,0
Invalidate TLB(s)	SBZ	MCR p15,0,Rd,c8,c7,0

禁用MMU和缓存

```
mrc     p15, 0, r0, c1, c0, 0
bic     r0, r0, #0x00002300   @ clear bits 13, 9:8 (--V- --RS)
bic     r0, r0, #0x00000087   @ clear bits 7, 2:0 (B--- -CAM)
orr     r0, r0, #0x00000002   @ set bit 1 (A) Align
orr     r0, r0, #0x00001000   @ set bit 12 (I) I-Cache
mcr     p15, 0, r0, c1, c0, 0
```

设置RAM

- board/samsung/smdk2410/lowlevel_init.S

```
mov  ip, lr
bl  lowlevel_init
mov  lr, ip
mov  pc, lr
```

重定位(Relocate)的概念

- 加载地址和链接地址要求一致
  - cpu/arm920t/u-boot.lds

- 将u-boot从Flash中搬运到内存中

重定位的实现

```
1.  relocate:                       /* relocate U-Boot to RAM */
2.      adr   r0, _start    /*current address of code */
3.      ldr   r1, _TEXT_BASE /* test if we run from flash or ram*/
4.      cmp  r0, r1                 /* don't reloc during debug */
5.      beq  stack_setup
6.      ldr  r2, _armboot_start
7.      ldr  r3, _armboot_end
8.      sub  r2, r3, r2             /* r2 <- size of armboot */
9.      add  r2, r0, r2             /* r2 <- source end address */
10. copy_loop:
11.     ldmia  r0!, {r3-r10}        /* copy from source address [r0] */
12.     stmia  r1!, {r3-r10}        /* copy to   target address [r1] */
13.     cmp  r0, r2                 /* until source end addreee [r2] */
14. ble copy_loop
```

设置堆栈

```
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   /* bdinfo*/
#ifdef CONFIG_USE_IRQ
    sub  r0, r0,
#(CONFIG_STACKSIZE_IRQ+CONFIG_STACKSIZE_FIQ)
#endif
    sub  sp,  r0,  #12      /* leave 3 words for abort-stack    */
```

清空BSS段、跳转到C函数

```
clear_bss:
    ldr   r0, _bss_start   /*   find start of bss segment    */
    ldr   r1, _bss_end     /*   stop here    */
    mov   r2, #0x00000000      /*    clear    */
clbss_l:
    str   r2, [r0]   /*    clear loop...      */
    add   r0, r0, #4
    cmp   r0, r1
    ble   clbss_l
    ldr   pc, _start_armboot
_start_armboot:          .word start_armboot
```

第二阶段代码

- lib_arm/board.c

```
void start_armboot (void)
{
        ……
        init_fnc_ptr = init_sequence;
        for (; *init_fnc_ptr; ++init_fnc_ptr) {
            if ((*init_fnc_ptr)() != 0) { hang(); } }
        ……
        env_relocate();
        ……
        for (;;) {
        main_loop ();  }
}
```

在u-boot中添加命令

} cmd_tbl_t include/command.h

```
struct cmd_tbl_s {
    char *name;
    int maxargs;
    int repeatable;
    int (*cmd)(struct cmd_tbl_s *, int, int, char *[]);
    char *usage;
    char *help;
};
```

添加命令hello

} common/cmd_hello.c

```
#include <common.h>
#incude <command.h>

int do_hello(cmd_tbl_t *cmdtp, int flag, int argc, char *argv[])
{
    printf("Hello World\n");
    return 0;
}
```

添加命令hello

- U_BOOT_CMD(
  hello, CFG_MAXARGS, 1, do_help,
  “hello --- brief info for hello\n”,
  “hello --- detailed info for hello\n”);

```
#define U_BOOT_CMD(name,maxargs,rep,cmd,usage,help) \
cmd_tbl_t __u_boot_cmd_##name Struct_Section = {#name,
    maxargs, rep, cmd, usage, help}
```

- common/Makefile
  COBJS-y += cmd_hello.o

总结

- Bootloader是搭建系统平台的第一步
- 汇编、C、开发板(原理图、芯片手册)

结束

华清远见