

基于 MIPS 的 Au1500 NC 板上的启动程序 U-Boot 设计

周涛, 刘有源

(武汉理工大学, 湖北 武汉 430063)

摘要: U-Boot 是一种功能强大的、开源的、嵌入式系统启动软件(Bootloader), 移植 U-Boot 相比自己重新开发一套启动程序可以节省大量的精力且后期的维护也更为方便。文中介绍了 U-Boot 在 Au1500 NC 板上的移植方法, 涉及对 U-Boot 的架构及与 MIPS 体系结构相关代码的分析, Flash 驱动程序的添加和以太网驱动程序的修改。通过移植, 最后成功实现了对 Linux 操作系统的引导, 从而为后期的进一步开发带来了极大的方便。

关键词: 嵌入式; U-Boot; Au1500; MIPS

中图分类号: TP311.54

文献标识码: A

文章编号: 1673-629X(2006)08-0094-03

Design of U-Boot on Au1500 NC Board Based on MIPS

ZHOU Tao, LIU You-yuan

(Wuhan University of Technology, Wuhan 430063, China)

Abstract: U-Boot is a powerful, open source bootloader used in embedded system. Compared with developing a new bootloader, porting U-Boot is more efficient and more convenient for maintenance. Introduces the porting of U-Boot on Au1500 NC board, including the analysis of the code relative to the MIPS architecture, the add and modifying of drivers. The porting boots the Linux system successfully. This is convenient for the further development.

Key words: embedded; U-Boot; Au1500; MIPS

0 引言

Au1500 是 AMD 公司针对网络设备市场设计的一款高性能、低功耗、高集成度的嵌入式处理器。它采用 MIPS 内核, 主频最高可以达到 500MHz^[1]。MIPS 内核在嵌入式领域有着广泛的应用, 它的指令严格按照五级流水线来执行, 通过一个协处理器来控制高速缓存(Cache)、异常、中断以及内存管理单元(MMU)的行为^[2]。U-Boot 则是一种开源且功能强大的启动引导程序, 其中 U 是指 Universal, 因而 U-Boot 的目标是成为一种通用的 bootloader, 它提供了很多有用的命令, 并对 Linux 提供了很好的支持。U-Boot 对 ARM 和 PowerPC 体系结构支持得较为完善, 而对 MIPS 结构的支持相对要弱很多。文中下面所介绍的就是在一款采用 Au1500 针对网络计算的开发板(NC 板)上移植 U-Boot 的方法, 所用的 U-Boot 版本为 1.1.2。

1 NC 板硬件组成及移植思路

本 NC 板主要硬件组成如下:

CPU, AMD Au1500 processor; 内存, HY57V561620CT

-H, 32MB x 2; Flash, SST 39vf160, 1M x 16 bit; 网络物理层, DAVIDCOM DM9161 x 2; IDE, ITE IT8212。其组成框图如图 1 所示。

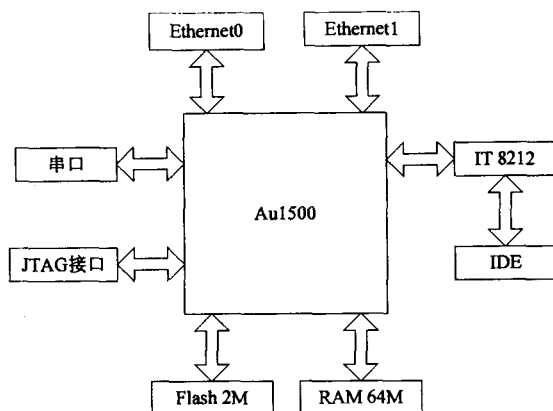


图 1 NC 板硬件组成框图

U-Boot 移植总的来说有三部分: 一是启动部分的修改, 这一块大部分是用汇编语言编写的, 主要是修改内存控制器参数(因为使用的内存芯片和 U-Boot 支持的不一样); 二是添加 flash 驱动; 三是修改网卡驱动, 增加对网卡(MII, Media Independent Interface)接口的支持。

先说一下编译工具, 编译 U-Boot 最好用的工具链是 ELDK, 网上可以下载, 注意的一点是它的工具链分大端(MIPS_4Kc)和小端(MIPS_4Kcle)两个版本, 不能混用, 这里使用的是小端版本。

收稿日期: 2005-11-22

作者简介: 周涛(1980-), 男, 湖北武汉人, 硕士研究生, 研究方向为嵌入式系统、驱动技术; 刘有源, 教授, 硕士生导师, 研究方向为人工智能、机器人技术、嵌入式系统。

2 启动部分的移植

这个移植是在 dbau1x00 开发板的基础上进行的,这也是 MIPS 体系结构中 U-Boot 目前唯一支持的一款板子。打开 `/cpu/mips/config.mk` 文件,将其中的 `-EB` 都改为 `-EL`,也就是让编译器将代码编译为小端格式。打开 `/board/dbau1x00/memsetup.s`,这个文件主要是对内存进行初始化的,不同板子内存型号不一样,这里要进行一些调整。这里代码的结构并不复杂,主要是对一些寄存器附各种参数值。先将“`beq zero, t1, big-endian`”改为“`beq zero, t1, little-endian`”,即设置 CPU 进入小端模式,可以看到,在小端模式的设置上,CPU、编译器参数、编译器版本的选择都必须一致才行。

下面说一下 MIPS 的启动过程,这个过程还是比较特殊的。

它开始于 `_start(/cpu/mips/start.s)`,其起始地址为 `0xbfc00000`(在 flash 的开始处),这个是 MIPS 结构的统一规定,它处于既不使用缓存(cache)也不使用内存管理单元(MMU)的 `kseg1` 段,CPU 启动时这些部件都没有初始化,无法使用,而只有在 `kseg1` 段,没有这些部件的支持程序也能运行^[3]。

然后 U-Boot 关掉不用的中断,cache 等,对 MIPS 内核做了些基本的初始化(Cache,内存控制器等)后就设置临时堆栈,跳转至 `board_init_f(/lib.mips/board.c)`,第一次跳到 C 程序,规划内存布局(包括 U-Boot 将要拷贝到内存的什么位置,一些全局量的布局及堆栈指针的位置等),跳回 `start.s`,将整个 U-Boot 从 flash 拷贝到 RAM,设置正式堆栈指针,最后跳转到内存中继续剩余的初始化工作(`/lib.mips/board.c` 中的 `board_init_r`,第二次跳到 C 程序)。

再来看对 SDRAM 相关寄存器参数的修改,这里只是针对自己使用的内存修改一些值,并不改动程序结构。现在程序是在 flash 中运行的,后面会转到 RAM 中运行,使用内存之前必须对内存控制器进行初始化。内存的起始地址位于 `0x80000000` 处,这个地址属于 `kseg0` 段,在 cache 初始化之后就可以使用了,可以不使用 MMU。

打开 `/board/dbau1x00/dbau1x00.c`,这里定义的两个函数都是在 `/lib.mips/board.c` 的 `board_init_f` 中被调用的。其中, `initdram` 是返回板子的内存大小,这里是直接返回,只要根据板子的内存大小修改即可。

3 Flash 驱动的添加

下面再来添加 flash 的驱动。NC 板使用 `sst` 的 `39VF160`,是 16 位的,2MB。U-Boot 中 flash 的驱动编写有一套比较固定的格式,主要是实现几个函数,函数名是统一的,以便上层调用。

① `flash_init`: U-Boot 中有一个 `flash_info_t` 结构用以描述 flash。 `flash_init` 函数主要就是根据 flash 的型号初始化 `flash_info` 中的一些量。此外, U-Boot 还为 flash 提

供了写保护功能,写 flash 前都要先检查写保护是否打开,如果打开了,就要先关闭写保护,写完后再次打开写保护。这只是软件意义上的保护, `flash_info_t` 中有一个 `protect` 变量用于描述写保护。

② `flash_erase`: flash 的擦除函数,擦除指定的若干个扇区,flash 在写之前都要先擦除再写。具体的擦除过程按照手册的要求编写就行了,总体上是先向指定的地址写入指定的数据序列,再检测 `DQ6` 位的变化。

③ `write_word`: 在指定地址写一个字的数据。

④ `flash_print_info`: 命令 `flinfo` 要调用的函数,打印 flash 的相关信息。

⑤ `write_buff`: 利用 `write_word` 实现在指定地址写入任意长度的数据,实现数据从 RAM 到 flash 的复制。

以 `saveenv` 命令的执行为例,它最后是调用 `/common/env-flash.c` 中的 `saveenv` 函数, `saveenv` 对 flash 的擦写是调用 `/common/flash.c` 中的 `flash_write` 函数,它最终又是调用了驱动程序中的 `write_buff` 函数。因此, `/common/flash.c` 可以看做是对 flash 操作的一个统一接口。

4 网络驱动的修改

网络物理层芯片使用的是 `DM9161`,它支持 MII 接口。原有的驱动写的并不完整,无法直接使用。`9161` 完成物理层的工作, `Au1500` 完成媒介访问控制层(MAC, Media Access Control)的工作,驱动可以看成是数据链路层,通过 MAC 间接控制物理层^[4]。

原驱动只实现了发送、接收、初始化和注册这 4 个函数,其中注册函数(`au1x00_enet_initialize`)是初始化 `eth_device` 结构,在 U-Boot 的启动过程中注册函数就会被执行。原驱动没有实现对物理层芯片的设置,不支持 MII 接口,因此,下面就来添加对 `DM9161` 的初始化。主要就是通过 MII 接口对 `9161` 的内部寄存器进行设置。

首先在文件开头添加“`include <miiphy.h>`”。再实现 `au1500_miiphy_read` 和 `au1500_miiphy_write` 这两个对 mii 接口读写的基本函数。下面再来编写其它函数。

(1) `au1500_mii_init`: 对 `9161` 的初始化,这里手动设置网络的模式为 100M,全双工, `9161` 也支持 `Autonegotiation` 模式,即自适应。

(2) `dm9161_isphy_connected`: 通过读取 `9161` 的 id 号并进行比对,来判断 `9161` 是否可以正常工作。

(3) `dm9161_link_status`: 判断网络是否真正的连通,为了保证判断的准确性,对 `9161` 的 `BMSR` 寄存器连续读取了两次,如果网络连通,再将 `9161` 中保存的网络状态(网速 10M, 100M, 全双工, 半双工)读出并存入 `Au1500` 的相应寄存器中。以上两个函数有利于对网络的调试。

(4) `au1500_halt`: 通过停止 `Au1500` 的各 FIFO 通道的工作来暂停网络的数据传输。在对网络进行初始化时可以先暂停网络,再进行各种设置。

最后在网络的初始化函数 `au1x00_init` 中添加对

au1500_mii_init 的调用。

Au1500 各有 4 个 DMA 收发 FIFO,一共是 8 个通道,收发过程对 4 个通道是轮流使用的。

U-Boot 实现了一个简单的 TCP/IP 协议栈,整个 U-Boot 是不使用中断的,因此对于数据包的接收 U-Boot 采用轮循的办法。U-Boot 没有实现 TCP 协议,只有 UDP 协议,这已经足够使用了。

5 对 /include/configs/dbau1x00.h 的修改

这个文件是整个 U-Boot 的配置文件,U-Boot 的很多功能和参数都可以在这里进行调节。下面只举出其中的一部分作为例子来说明。

* CFG_MALLOC_LEN: malloc 区域可用的空间大小,这里设为 128 * 1024,即 128kB;

* CFG_HZ: CPU 的主频,设为 396000000;

* CFG_SDRAM_BASE: 内存基址,设为 0x80000000;

* CFG_MAX_FLASH_BANKS: 可以看成是 flash 的片数,设为 1,只用了一片 flash;

* PHYS_FLASH_1: 第一片 flash 的起始地址,为 0xbfc00000;

* CFG_ENV_IS_IN_FLASH: 将 CFG_ENV_IS_NOWHERE 改为 CFG_ENV_IS_IN_FLASH,其值设为 1,这样使 U-Boot 支持从 flash 读取环境变量;

* CFG_ENV_ADDR: 环境变量保存在 flash 中的起始地址,为 0xbfc80000;

* CFG_ENV_SIZE: 保存环境变量所占空间,为 0x10000;

* CFG_DCACHE_SIZE, CFG_ICACHE_SIZE, CFG_CACHELINE_SIZE: 对 cache 大小的设置。

6 U-Boot 对 Linux 的引导

在调试阶段,可以通过 tftp 将内核上传到指定的内存地址。之后可以用 go 命令执行上传的文件,这是针对纯二进制文件,go 命令直接跳转到指定地址执行上传的文件。U-Boot 还带有一个 mkimage 工具,它可以在文件开头加一个 U-Boot 可以识别的文件头,这样使用起来更

加方便。对 Linux 的引导是采取的这种方法。这种文件只能用 bootm 命令启动。

首先获得没有压缩的 Linux 内核纯二进制文件^[5],用 gzip 对其压缩,再用 mkimage 打包。打包时注意 -a 和 -e 这两个选项,-a 后面跟的是加载地址(load address),这里的加载地址并不是指的 tftp 的上载地址,bootm 执行后上传的文件首先被解压缩,-a 后面跟的地址即是解压地址。由于解压需要较大的空间,解压地址和 RAM 的起始地址之间应该留有足够的空间。-e 后面跟的是解压之后的跳转进入地址(entry point)。

文中使用的 mkimage 命令如下:

```
mkimage - AMIPS - O linux - T kernel - C gzip - a
0x82000000 - e 0x81000000 \ - n "Linux Kernel Image" - d linux.
bin. gz uImage
```

再说一下 bootargs 环境变量的设置,以笔者为例:

```
setenv bootargs root = /dev/nfs rw nfsroot = 191.168.
123.146:/home/au1500 nfsaddr = 191.168.123.152:191.
168.123.146
```

这里是用 nfs 的方式加载文件系统,nfsaddr 中前一个地址是传给内核的板子的 IP,后一个是 nfs 服务器的 IP 地址。

7 结束语

通过移植,U-Boot 可以正常运行,一方面借助其强大的功能方便了后续的开发工作,另一方面,以后的 U-Boot 维护工作也可以借助开源社区的力量来完成。

参考文献:

- [1] Advanced Micro Devices, Inc. AMD Alchemy Solutions Au-1500 Processor Data Book [EB/OL]. <http://www.amd.com>, 2003.
- [2] MIPS Technologies, Inc. MIPS32 Architecture For Programmers [EB/OL]. <http://www.mips.com>, 2001.
- [3] Sweetman D. MIPS 处理器设计透视 [M]. 赵俊良等译. 北京:北京航空航天大学出版社, 2005.
- [4] Stevens R. TCP/IP 详解 [M]. 谢希仁等译. 北京:机械工业出版社, 2004.
- [5] MIPS Technologies, Inc. Linux Kernel Archives [EB/OL]. <http://www.linux-mips.org>, 2005.

(上接第 93 页)

这种分配方案使其具有占用 RAM 和 ROM 空间小的特点,从而能够极大地节省开发成本,丰富人机交互信息。因此可以得到结论:此方案可以有效地提高 $\mu C/GUI$ 系统的实时性,使之可以带来更大的经济效益,提高产品的性能。

参考文献:

- [1] 罗 蕾. 嵌入式实时操作系统及应用开发 [M]. 北京:北京航空航天大学出版社, 2005. 23-26.

- [2] 谭浩强. C 程序设计 (第 2 版) [M]. 北京:清华大学出版社, 2000. 101-108.
- [3] 探砂工作室. 嵌入式系统开发圣经 [M]. 北京:中国青年出版社, 2002. 139-149.
- [4] Mercer C W. An Introduction to Real-Time Operating Systems: Scheduling Theory [Z]. School of Computer Science, Carnegie Mellon University, 1992.
- [5] Bruyninckx H, Leuven K U. Real-Time and Embedded Guide [J]. Mechanical Engineering, 2001, 56: 5-9.