

文章编号:1001-9081(2006)07-1727-03

基于 arms3c44b0x 处理器的 Bootloader 设计和实现

胡修林,余凯军

(华中科技大学 电子与信息工程系,湖北 武汉 430074)

(catchingyu@gmail.com)

摘 要:在不同安全域之间传输文件,只能是低保密级别安全域的文件向高保密级别安全域传输。介绍了基于 USB2.0 总线的单向数据传输系统的实现原理及技术,系统地阐述了如何采用 USB2.0 集成芯片 CY7C68013 与光耦、双口 RAM 结合来实现计算机单向数据传输的方案,通过实验验证该系统实现了单向、可靠、快速的数据传输。

关键词:启动程序;arm;可移植性;嵌入式系统

中图分类号:TP303 **文献标识码:**A

Design and implementation of bootloader based on arms3c44b0x

HU Xiu-lin, YU Kai-jun

(Department of Electronics and Information Engineering, Huazhong University of Science and Technology, Wuhan Hubei 430074, China)

Abstract: When transferring files between the different security domains, files are required to be only transferred from low security domain to high security domain. An unilateral transferring system was developed based on USB2.0 bus. It used USB2.0 interface device CY7C68013, DAUL-PORT RAM, and optical couplers insulation. The experiments verify that this system realizes unilateral, reliable and high-speed transmission.

Key words: bootloader; arm; flexibility; embedded system

启动程序(Bootloader)是与硬件系统高度相关的一段程序。在嵌入式系统的应用中,它起着举足轻重的作用。系统每次加电复位以后,处理器将首先执行 Bootloader。它把系统的软硬件环境设置在一个合适的状态,以便操作系统内核和用户的应用程序运行。其功能类似于 PC 机上的 BIOS。

笔者在参与一款手持导航设备的研发中,为该设备设计实现了 Bootloader。

1 Bootloader 设计概述

Bootloader 一般要实现的功能如下:1)初始化 CPU、存储器、存储器配置寄存器以及串口等;2)激活指令/数据 Cache、建立堆栈指针、建立启动参数区、构造参数结构和标识列表;3)通过上电自检,识别存在哪些设备,并报告异常,提供对电源管理中休眠/恢复的支持;4)跳转到内核起始处,启动系统^[1]。这是 Bootloader 需实现的基本功能。操作系统内核是与硬件无关的代码,它在不同的硬件平台上运行都必须附加一段启动代码。Bootloader 就承担这部分启动工作,保证操作系统能够在不同的平台上顺利运行。但是硬件的多样性导致了一个标准、通用的 Bootloader 的编写比较困难^[2]。本文提出将 Bootloader 划分为初始化部分和可重用部分,移植的时候只需修改初始化部分代码。初始化部分在操作系统运行后可被覆盖,节省了系统的内存资源,增强了系统的可移植性。

市面上存在一些源代码公开的 Bootloader,它们主要是针对 Linux 操作系统的,如 redboot, Uboot 等。这些代码对 Bootloader 的编写都有着重要的参考价值。但是这些启动代码比较复杂,又是专门为 Linux 开发的,移植起来并不容易。本文所描述的 Bootloader 侧重于代码的可移植性和通用性,

并已被用作一款国产操作系统的启动代码。

2 Bootloader 的设计与实现

2.1 整体设计与模块划分

从整体上划分,笔者设计的 Bootloader 分为两个部分:可重用部分和初始化部分(如图 1)。我们将后续程序需要的服务以接口的方式提供出来,并将这部分代码及其需要的数据集合一起称为可重用部分。后续程序运行起来之后,初始化部分被覆盖,而可重用部分仍然存在。例如,后续的应用程序需要启动某一个应用程序映像(在 Bootloader 看来,任何非二进制格式的映像都是纯数据文件)。可以调用 Bootloader 的读取纯数据文件内容的接口,将其读取出来,然后加载到系统中并运行。

依据这种划分,我们将“初始化部分”定义为 Bootloader 启动第一个映像(操作系统映像)前需要执行的,而在后续程序运行中通过接口又不会使用到的代码和数据部分。“初始化部分”在后续程序运行时可将其覆盖。如果后续程序不需要使用可重用部分,也可将其覆盖。如果都被覆盖,那么映像文件开始运行起来后,RAM 中就不存在 Bootloader 的代码,Bootloader 完全把 arms3c44b0x 处理器交给操作系统和应用程序。需要注意的是,可重用部分和初始化部分之间的界限并不是绝对的。可能在某个系统中,其后续的程序不需要某个接口提供的服务,则这个接口相关的代码就可以被划分到初始化部分中;而在另一个系统中,后续的程序需要使用这个功能,则这个接口就必须被划分到可重用部分。

图 1 中初始化模块完成上电自检和硬件初始化工作,命令处理模块完成交互方式中各种命令的执行和处理,通信模

收稿日期:2006-01-23

作者简介:胡修林(1955-),河南滑县人,教授,博士生导师,主要研究方向:现代通信系统与通信网、多媒体通信、卫星通信;余凯军(1982-),湖北武汉人,硕士研究生,主要研究方向:嵌入式系统、现代通信系统与通信网。

块主要实现与 GDB(调试工具)或用户进行信息交互时的各种通信协议,通信设备驱动模块完成串口或网口的底层设备驱动,解压缩模块用来对镜像文件进行压缩和解压缩处理。可重用部分包括接口模块、Flash 管理模块和 Flash 编程模块。

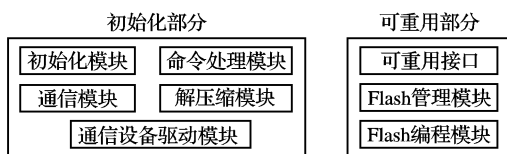


图1 Bootloader 模块划分

2.2 设计与实现

2.2.1 镜像下载与启动

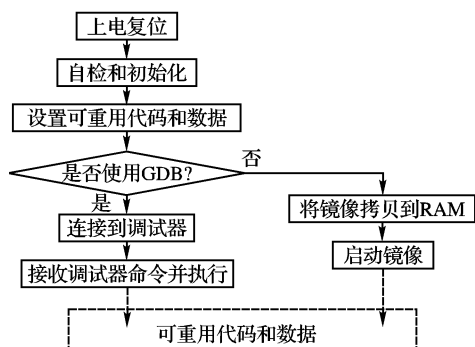


图2 上电启动流程

如图2所示,上电复位和初始化设置以后,用户可以通过触摸屏、键盘按键、硬件跳线或硬件开关来选择是否进入交互方式。如果用户选择进入交互方式,则 Bootloader 等待与调试器(GDB)的通讯连接。连接成功以后,目标机上的 Bootloader 将通过串口或网络等通信手段从主机下载文件,如下载内核镜像和根文件系统镜像等。如果选择启动后续映像程序,则按照指定的映像程序描述信息采用对应的方式启动。除了用户输入选择以外,系统还可以是配置选择。这个配置文件是存储在 Flash 上的一个纯数据文件。

启动镜像有两种方式,一是直接在 Flash 中启动映像开始执行。在这种情况下,映像程序可以把自己的部分代码拷贝到 RAM 中执行,以提高执行的速度。Bootloader 一般直接采用一条 jump 指令跳转到映像的入口点。这种情况下,要求 ROM 中的映像的运行地址必须同这时处理器(arms3c44b0x)地址空间中见到的 ROM 地址一致,这种一致性由用户在连接时指定其正确的运行地址及装载地址,在固化时指定对应的 Flash 地址来保证。二是 Bootloader 先将 Flash 中固化的映像拷贝到 RAM 中,然后使用 jump 指令跳转到其入口开始运行。这时候,Flash 中的固化地址和映像的连接定位基本上没有什么关系。

2.2.2 硬件初始化与自检

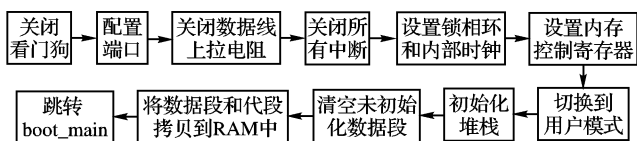


图3 初始化流程

如图3所示,这一段代码是用汇编完成的,其后的代码都是用 C 语言编写的。这样的设计有利于 Bootloader 的移植。跳转到 boot_main 最简单的实现方法是直接把 boot_main() 函数的起始地址作为执行映像的入口点。但是这样做会有两个缺点:一是无法通过 boot_main() 函数传递函数参数;二是无法处理 boot_main() 函数返回的情况。笔者在实现的过程中

采用了弹簧床的概念。实现代码如下:

```
.text
.globl_trampoline
_trampoline
bl boot_main
b _trampoline
```

这样,在转到 C 语言编写的 boot_main 函数之前,可以通过寄存器 R0 ~ R3 来传递参数,参数返回使用寄存器 R0 和 R1。当 boot_main 函数返回的时候,实际上是重新再次调用 boot_main 并执行。程序也可以根据需要,在 boot_main 函数返回后做相应的后续处理(如通知用户等)。这样就完全克服了直接跳转的两个缺点。

硬件初始化实现的过程中特别要注意异常服务堆栈的初始化(如合理设置其大小)。图4描述了异常服务堆栈在系统 RAM 空间中的分配情况。异常(如中断)服务子程序使用该堆栈。当有大量的数据传输采用 FIQ 或 IRQ 中断方式时^[3],FIQ 或 IRQ 的堆栈要足够大。否则系统的内存就会产生越界,严重的时候会导致系统崩溃。Bootloader 程序中使用的堆栈被安排在内存的底部,在 Bootloader 运行完毕以后,这部分内存可以被后续应用(操作系统和应用程序)继续使用。

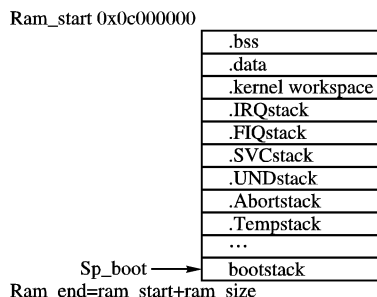


图4 内存分配

上电自检部分可以根据具体应用的需要做相应的修改。笔者设计的 Bootloader 应用在一款手持导航设备中。应用中,上电自检部分须完成系统内存映射的检测、对 FPGA 接口的配置、对 IC 卡的设置和身份验证、GPRS 模块的自检以及天线射频部分的检测。自检完成以后手持机把自检信息通过串口或者 LCD 液晶显示屏通知用户。所谓的系统内存映射检测是指 Bootloader 须知道整个物理地址空间中哪些地址是有效的地址单元。ARMS3c44b0x 处理器把从 0x0c000000 到 0x10000000 开始的 64M 空间分配用于 RAM。虽然 ARMS3c44b0x 预留了一大段足够的地址空间给系统的 RAM,但是在搭建具体的嵌入式系统时却不一定实现全部的地址空间。如笔者在实际应用中就只采用了 16M 的 RAM。因此,Bootloader 在自检的时候应该检测整个系统的内存映射情况,即它必须知道系统预留的全部 RAM 空间的哪些被真正映射到实际的 RAM 地址单元,哪些是未使用状态。检测内存映射可在每个页面头两个字节的位置任意写两个值(如 0x55 或 0xaa),然后马上读回。如果读数与预期的一样,则说明该地址有效。如果不一样,则说明该地址未对应实际物理空间。

2.2.3 镜像管理

Bootloader 能够查询 Flash 空间的使用情况(如已用块的起始地址、大小,未用块的分布情况及其大小),Flash 中镜像的情况(镜像的名字,镜像的大小,镜像的类型:可执行、纯数据,镜像的创建时间等),并可删除镜像等。这些信息通过一个目录来管理,这个目录中的每个目录项对应着一个镜像(和纯数据块),GDB 和应用程序均可以查询这个目录。Image Directory 一般位于第一块 Flash 的最后一个扇区,描述

Flash 中 image 的信息。每一个 image 的描述信息都在其对应的结构 struct image_desc 中, Image Directory 就是由一个个数据类型 struct image_desc 构成。

```
struct image_desc {
    T_UBYTE name[16];           //the name of image
    T_UWORD flash_base;         //the addr of the image
    T_UWORD ram_base;
    T_UWORD size;               //the size of image (byte)
    T_UWORD entry_point;        //the entry point of image,
    T_UBYTE image_type;         //the type of image,
    T_UWORD datetime;          //the date and time of image
    T_UWORD desc_cksum;         //the checksum of name[16],
};
```

只要系统内固化了 Bootloader, 并且 Bootloader 启动过, Flash 中就至少有两个 image。一个是固化在 Flash 上的 Boot image; 另外一个 Image Directory。对应这两个 image, 在 Image Directory 内就会有二个 image_desc。一个用来描述 Bootloader, 一个用来描述 Image Directory。当通过 Bootloader 在 Flash 中固化了多个 image 时, Image Directory 内就会有多个 image_desc。

2.2.4 通信协议

Bootloader 可通过串口和网络同运行于主机的 GDB 建立连接。笔者在实现通信协议(串口协议或网络协议)的时候做了相应的简化。如要接受一个字符'a'时, 其 ASCII 码值为 97(0x61)。则无论在宿主机还是在目标机上都是以二进制串 01100001 保存。但是在通讯传送时, 则将'a'转化成 0x6 和 0x1, 而 0x6 和 0x1 对应的 ASCII 码值分别是 54(00110110) 和 49(00110001)。然后把这两个 ASCII 码分别传送。这样, 所有的数据的 ASCII 码值就肯定在 0x0 到 0xf 之间。其他的

字符就可以作为控制字符, 从而简化通信协议。当然, 付出的代价是传送的数据量增大了一倍。

3 Bootloader 性能分析

3.1 可移植性

通过修改初始化代码、通讯驱动程序和 Flash 硬件编程代码, Bootloader 可以移到各种类型的处理器和目标板上, 具有良好的可移植性。

3.2 时间性能

在交互方式下, Bootloader 的固化程序的速度主要受限于两个方面。一方面受限于通讯的速度: 在使用串口通讯时, 受限于串口的波特率; 在使用网络通讯时, 受限于当前的网络阻塞状态以及主机端、目标机端网口的通讯能力。另一方面, 固化程序的速度还受限于 Flash 芯片的硬件编程响应速度。与编程所使用的缓冲区的大小基本上没有关系。

4 结语

笔者设计的 Bootloader 具有良好的可移植性和健壮性, 只须有针对性地修改部分代码, 该 Bootloader 就可轻松地移植到各种类似的嵌入式平台上。硬件平台资源的任何变动和异常, 程序都会通过串口或人机界面通知用户。该 Bootloader 已应用在一款手持导航设备中, 运行稳定。

参考文献:

- [1] 马学文, 朱明日, 程小辉. 嵌入式系统中 Bootloader 的设计与实现[J]. 计算机工程, 2005, 31(7): 96-98.
- [2] 田泽. 嵌入式系统开发与应用教程[M]. 北京: 北京航空航天大学出版社, 2005. 14-15.
- [3] SamSung electronics. Arms3c44b0x datasheet[Z], 2001.

(上接第 1714 页)

讯 Agent 发出请求异地库协作匹配指令, 并传递相应的目标匹配对象的描述信息, 通讯 Agent 负责向其他同源数据库发出协作匹配请求, 所有符合条件的匹配代理都会收到请求, 匹配代理₂ 到匹配代理_n 的通讯 Agent 在响应请求后, 分别在数据库₂ 到数据库_n 中进行匹配, 并向匹配代理₁ 返回结果, 通讯 Agent 选择匹配等级最高的匹配信息提交用户作选择。

2.4 模糊匹配的处理

一般说来, 模型系统双向匹配过程中, 往往难得到精确匹配解(即完全满足用户所列全部要求的解), 而实际上人们往往只想要模糊匹配解(只要在某种程度上能满足用户所列主要要求的解, 即可满足用户大多数的匹配要求)。对此, 可引入关键匹配项和匹配等级的概念来解决。

关键匹配项 它是用户在提出申请时对匹配对象所列出的最基本要求条件。显然, 只要该项不满足时, 就可拒绝与之匹配。

匹配等级 它是将用户要求的双向匹配条件的符合程度, 按照双方相互满足要求的个数占全部满足要求个数的比例而分为匹配等级数 10%~100%。若匹配等级数为 100% 时, 则表明当前匹配是最精确匹配(即双方均可彼此百分之百地吻合对方的匹配条件); 若匹配等级为 10%, 则表明当前匹配是最粗略匹配(即“对方只能满足本方匹配要求条件的十分之一”)。

双向匹配模型系统通过感知 Agent 获得用户对模糊匹配的接受程度信息(即关键匹配项和匹配等级等), 传给协调 Agent 并存入数据库, 用以指导后续匹配处理。自然, 在匹配过程中, 匹配等级高的包容了匹配等级低的。在求解过程中, 要力求匹配等级最高的解(注意, 等级的设定不会影响最优解的求得)。

3 结语

首先, 本文基于多 Agent 技术的双向智能自动匹配系统模型, 进行网络信息整合与模糊匹配, 克服传统人工或半人工双向匹配的缺点, 实现双向匹配的自动智能处理。其次, 它可应用于多数存在双向选择的问题中, 典型的有求职招聘匹配、男女择偶匹配和研究生与导师的选择匹配等。笔者正在利用 AgentBuilder^[5,6] 集成开发平台研究如何用高效并行算法实现一个以该模型为基础的求职招聘匹配系统, 并将利用 KQML 表示各 Agent 间的通讯原语, 以实现它们之间的通讯^[7]。从实验初期看, 此模型是可行的。

参考文献:

- [1] 杨鹤标, 陈华, 徐向英. 基于 Agent 的智能检索技术在类库管理系统中的应用研究[J]. 计算机工程与科学, 2005, 27(3): 74-76.
- [2] 蔡俊. 多 Agent 技术在网络教学交互模型设计中的研究[J]. 计算机科学, 2004, 31(增刊): 324-327.
- [3] 李陶深. 人工智能[M]. 重庆: 重庆大学出版社, 2002.
- [4] 彭志平, 李绍平. 面向 Agent 与面向对象的软件技术[J]. 计算机工程与科学, 2005, 27(3): 77-79.
- [5] User-guideraboutagentbuilder[EB/OL]. <http://www.agentbuilder.com>, 2005.
- [6] Reference-manual about agentbuilder[EB/OL]. <http://www.agentbuilder.com>, 2005.
- [7] 张谦, 俞集辉. Agent 技术在 Web 数据仓库结构中的应用研究[J]. 计算机科学, 2005, 32(6): 79-81.
- [8] 高刚毅, 金勤, 陈海波. 基于多 Agent 结构地理信息服务研究[J]. 计算机应用与软件, 2005, 22(8): 60-62.