基于AT91RM9200的BootLoader设计与实现

王 燚,栾英姿
(西安电子科技大学,陕西 西安 710071)

摘 要:BootLoader的编写在嵌入式系统软件开发中非常重要。目前已有一些通用的BootLoader,但如何根据特定的嵌入式系统平台,开发相应的BootLoader是一个重点和难点。基于AT91RM9200硬件平台,介绍了BootLoader开发要完成的主要任务和实现方法,并对经过实践验证的启动代码进行了详细的分析,设计与实现了具有良好可移植性的简单灵巧的嵌入式系统的启动程序。

关键词:BootLoader;AT91RM9200;ARM;启动代码
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2009)03-0189-04

Design and Realization of BootLoader Based on AT91RM9200

WANG Yi, LUAN Ying-zi
(Xidian University, Xi'an 710071, China)

Abstract: BootLoader programming is very essential in the software development of embedded system. Currently, there are some common BootLoaders, but how to develop and port it on the basis of special hardware platform is an important and difficult question. Based on AT91RM9200, illustrates the important task and realization method of BootLoader, it also analyzes the well-tested startup code and designs a smart startup code.

Key words: BootLoader; AT91RM9200; ARM; startup code

0 引 言

嵌入式系统的BootLoader,类似于PC中的BIOS,是系统加电/复位后运行的第一段程序。通过这段程序,可以初始化硬件设备、建立内存空间的映射等,使系统软硬件进入到合适的状态[1]。BootLoader完成基本软硬件环境初始化后,对于在有操作系统的情况,将启动操作系统,启动内存管理、任务调度,加载驱动程序等,最后执行应用程序或等待用户命令;对于没有操作系统的情况则直接执行应用程序或等待用户命令。

嵌入式系统中常见的BootLoader有以下几种:U-Boot、VIVI、Blob、RedBoot、ARMboot和DIY(自己制作)方式[2]。其中U-Boot、VIVI、Blob、RedBoot和ARMboot等成熟工具移植起来简单快捷,但由于它们是面向大部分硬件的工具,所以在功能上要满足大部分硬件的需求,这导致它们的代码量较大,而且它们在使用上不够灵活,比如在这些工具上添加自己的特有功能相对比较困难,因为必须熟悉该代码的组织关系,以及了解它的配置编译等文件。用DIY方式自己编写针对目标板的BootLoader不但代码量小,同时灵活性很大,最重要的是将来容易维护。文中就是在无操作系统情况下,采用DIY方式设计开发了一个既满足应用要求又简单灵巧的BootLoader。

1 硬件基础和开发环境

文中所设计的BootLoader针对的目标板的微控制器(MCU)是ATMEL公司AT91系列的AT91RM9200。AT91RM9200是一款有代表性的ARM芯片,它基于ARM920T内核,支持重映射,内部集成有16k字节的SRAM和128k字节的ROM。

本目标板与所设计的BootLoader相关的资源配置如下:

(1)片内16k字节的SRAM,起始地址为0x200000;

(2)板上16M字节,型号为IS42S32400B的SDRAM,起始地址为0x20000000;

(3)板上16M字节,型号为MT28F128J3的Flash,起始地址为0x10000000。

收稿日期:2008-07-02
基金项目:军事科研项目
作者简介:王 燚(1983-),男,硕士研究生,主要从事无线数据传输及嵌入式系统在通信中的应用的研究;栾英姿,硕士生导师,副教授,研究方向为多载波码分多址中的关键技术。

开发环境:开发工具采用 ARM 公司的 ADS1.2,开发主机通过仿真器连接目标板以建立交叉开发调试环境。

2 启动方式的选择

AT91RM9200 芯片提供了灵活多样的启动方式。当系统上电复位后,CPU 将从复位地址 0x0 处开始取指令,大多以微处理器为核心的嵌入式系统通常都将某种类型的固态存储设备映射到复位地址上。AT91RM9200 根据上电复位后其 BMS(Boot Mode Select,启动模式选择)引脚状态决定器件将内部还是外部存储器映射到复位地址处[3]。若 BMS 为高电平,则将内部存储器域 1 映射至内部存储器域 0,即由内部 ROM 启动;若 BMS 为低电平,则将外部存储器域 0 映射至内部存储器域 0,即由外部存储器域 0 启动。

内部 ROM 固化的引导程序包含 BootLoader 和 BootUploader 两大部分。先激活 BootLoader,依次查找连接在 SPI 上的 DataFlash、连接在两线接口(TWI)上的 EEPROM 或连接在外部总线接口(EBI)上的 8 位存储器件上是否有合法的程序。如果检测到有效中断向量序列,代码将载入内部 SRAM,如果没有检测到有效序列,便引导 Uploader 启动,它将调试单元串行端口(DBGU)或 USB 器件端口初始化,然后通过 USB 的 DFU 协议或 DBGU 的 XMODEM 协议将代码载入内部 SRAM[4]。由于本芯片支持地址重映射,当上述载入内部 SRAM 的代码下载完成后,将内部 SRAM 的地址由 0x00200000 重新映射到 0x00000000,将内部 ROM 地址由 0x00000000 映射到 0x00100000,然后跳转到地址 0x00000000 处的内部 SRAM 开始执行。

这种从内部 ROM 启动的方式有一些软硬件限制需要注意[3]:

·载入的代码大小要小于内置于产品中 SRAM 的大小。

·在 TWI 总线上连接的 EEPROM 地址必须为 0。

·有效代码由片外存储器(DataFlash,EEPROM)的 0x00000000 地址处下载到内部 SRAM 地址 0x00000000(重新映射后)。

·下载代码必须位置独立或连接在地址 0x00000000 处。

·DataFlash 必须连接在 SPI 的 NPCS0 上。

·8 位并行 Flash 必须连接到 EBI 的 NCS0 上(适用于集成了 EBI 的器件)。

文中采用的是 BMS 为低电平,将连接到 EBI 的 NCS0 上的 Flash 即外部存储器域 0 映射至内部存储器域 0,也就是从 Flash 启动。

3 启动流程的设计

BootLoader 是依赖于硬件实现的,特别是在嵌入式系统中。不同体系结构需求的 BootLoader 是不同的,除了体系结构,BootLoader 还依赖于具体的嵌入式板级设备的配置,而且根据实现的功能不同,其复杂程度也各不相同,因而不易编出统一的启动代码,但主要功能及其基本原理是相通的,下面将结合所设计的启动流程来分析其主要的功能。

3.1 建立中断向量表

ARM 通常要求异常向量表必须放置在从 0 地址开始、连续 8×4 字节的空间内。每当一个中断发生后,ARM 处理器便强制把 PC 指针指向异常向量表中对应中断类型的地址值。因为每个中断只占据向量表中一个字的存储空间,所以只能放置一条 ARM 指令,包含简单的跳转指令,使程序跳转到存储器里放置异常中断服务程序的地方,再执行具体的中断处理。相关的程序代码如下:

```
VectorTable
    LDR PC, ResetAddr;          复位,地址 0x0
    LDR PC, UndefinedAddr;      未定义指令,地址 0x4
    LDR PC, SWIAddr;            软件中断,地址 0x8
    LDR PC, PrefetchAddr;       预取中止,地址 0xC
    LDR PC, DataAbortAddr;      数据中止,地址 0x10
    NOP;                        保留,地址 0x14
    LDR PC,[PC,# -0xF20];       IRQ 中断,地址 0x18
    LDR PC,[PC,# -0xF20];       FIQ 中断,地址 0x1C
ResetAddr       DCD InitReset
UndefinedAddr   DCD Undefined
SWIAddr         DCD SoftwareInterrupt
PrefetchAddr    DCD PrefetchAbort
DataAbortAddr   DCD DataAbort
```

比如程序复位,将跳转到 0x0 处,执行 LDR PC,ResetAddr 这条指令,而 ResetAddr 又通过后面的一条伪指令 ResetAddr DCD InitReset 得到了定义,则执行的结果是使 PC 跳到 InitReset 标号处执行。使用 LDR 指令而不使用 B 指令跳转的原因是 LDR 指令可以全地址范围跳转,而 B 指令不行。

3.2 初始化堆栈

ARM 微处理器支持 7 种工作模式,工作模式位 M[4:0]是程序状态寄存器 CPSR 的低五位,用来标识或设置处理器的工作模式,如表 1 所示。

每种模式都有一个专门的堆栈寄存器 SP,保存其堆栈指针。初始化堆栈必须确定两点:第一,堆栈的位置,这决定堆栈在内存里从何处开始;第二,堆栈的大小,大小如果不合理会导致栈溢出,使系统不稳定。初始化堆栈时先初始化每种模式下的 SP,使其指向该运

行模式的堆栈空间,然后改变状态寄存器 CPSR 的状态位,使处理器切换到相应的模式。

表 1　ARM 微处理器 7 种工作模式

处理器工作模式	说明	M[4:0]
用户模式(usr)	正常的程序工作模式	10000
快速中断模式(fiq)	用于高速数据传输或通道处理	10001
外部中断模式(irq)	用于通用的中断处理	10010
管理模式(svc)	操作系统使用的保护模式	10011
中止模式(abt)	用于虚拟存储及存储保护	10111
未定义指令模式(und)	支持硬件协处理器的软件仿真	11011
系统模式(sys)	运行具有特权的操作系统任务	11111

下面以设置 IRQ 模式堆栈的相关代码为例来说明堆栈的设置:

```
IRQ_STACK_SIZE    EQU 3*8*4;栈大小设置
TOP_STACK         EQU RAM_LIMIT;栈顶定义
ARM_MODE_IRQ      EQU 0x12
I_BIT             EQU 0x80
F_BIT             EQU 0x40
;设置外部中断模式及其堆栈
ldr r0,= TOP_STACK
msr CPSR_c, #ARM_MODE_IRQ:OR:I_BIT:OR:F_BIT
mov sp, r0
```

按照以上步骤可以设置其它模式及其堆栈,系统需要初始化哪些堆栈取决于用户使用了哪些中断,并且系统需要处理哪些异常类型。

3.3　初始化硬件

软件运行离不开硬件,因此 BootLoader 必须对硬件进行初始化。硬件的初始化主要通过配置特殊控制寄存器来完成[5],包括以下几个部分:

·关看门狗。

·屏蔽所有中断。中断屏蔽可以通过写 CPU 的中断屏蔽寄存器或状态寄存器(ARM 的 CPSR 寄存器)来完成。

·初始化 PLL 和时钟。一般情况下,系统启动时 CPU 会自动选择慢速访问外部存储器,以适应大多数设备,并且在启动时禁止了部分时钟信号。因此,要根据自己的需求对时钟进行适当的配置,以达到期望的速度,并且使能相应的外设时钟以便访问对应的外设。基于 AT91RM9200 芯片的系统刚启动时,默认选择的是工作在 32.768kHz 下的慢时钟模式,此时主振荡器及 PLL 关闭以节约功耗,因此为了提高性能和使用相应的外设,必须使能主振荡器及 PLL 并进行相应的设置,使时钟信号的频率满足相应的要求。

·初始化存储系统和地址重映射。可以使用专用的特殊功能寄存器来控制外部存储器(主要是 Flash、SDRAM)的读/写操作,通过对该组特殊功能寄存器编程,可以设定外部数据总线宽度、访问周期、存储器类型、时序和定时的控制信号等参数。又由于代码在 RAM 中运行时,有明显的速度优势,而且变量可以动态配置,因此常将中断向量表放到内部 SRAM 中,并通过重映射将内部 SRAM 映射到 0x0 地址处,使得出现异常时 ARM 从内部 SRAM 中取得向量。

中断向量表复制部分的代码如下:

```
mov r8, #SRAM_BASE ;定位向量在片内 SRAM 中地址
add r9, pc, #-(8+.-VectorTable) ;定位读取相对值地址
ldmia r9!, {r0-r7} ;读取 8 个向量
stmia r8!, {r0-r7} ;保存在片内 SRAM
ldmia r9!, {r0-r4} ;读取 5 个绝对地址
stmia r8!, {r0-r4} ;保存在片内 SRAM
```

将中断向量表复制到 SRAM 后,通过向存储控制器(MC)的重映射控制寄存器(MC_RCR)的重映射命令位 RCB 写 1 即可实现重映射将内部 SRAM 映射到 0x0 地址。这样出现异常时将从内部 SRAM 中取得向量,提高了中断响应速度。

3.4　初始化应用程序执行环境

一个可执行程序的映像由 RO、RW、ZI 三个段组成,其中 RO 为只读代码段;RW 是已初始化的全局变量;ZI 是未初始化的全局变量。映像一开始总是存储在 ROM 或 Flash 里面的,其 RO 段是可读的,在运行的时候不可以改变,所以 RO 部分既可以驻留在 ROM 或 Flash 里执行,也可以转移到速度更快的 RAM 中执行;而 RW 段是可读/写的,所以在运行时必须被装载到 RAM 里,并将 ZI 段清零,以保证程序可以正确运行。

所谓应用程序执行环境的初始化,就是完成必要的从 ROM 或 Flash 到 RAM 的数据传输和内容清零。复制前后代码和数据段分布如图 1 所示。

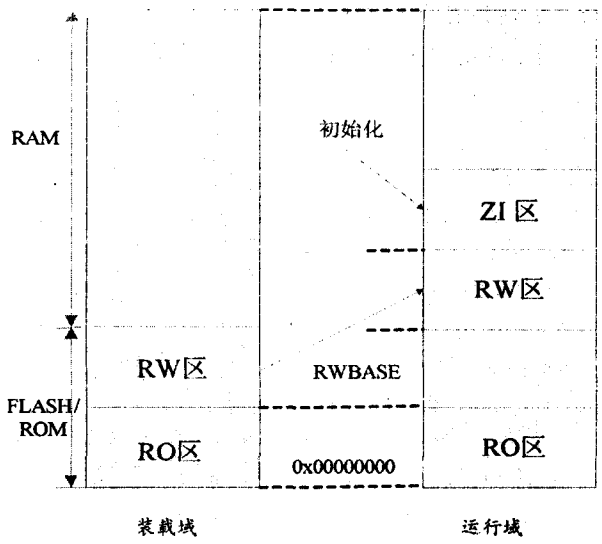

图 1　装载域和运行域的地址映射

下面是在 ADS 下，一种常用的实现模型：

```
IMPORT |Image$$RO$$Base| ;RO 区开始地址
IMPORT |Image$$RO$$Limit| ;RO 区末地址后面的地址即 RW 数据源起始地址
IMPORT |Image$$RW$$Base| ;RW 区在 RAM 里的执行区起始地址
IMPORT |Image$$RW$$Limit| ;RW 区末地址后面的地址
IMPORT |Image$$ZI$$Base| ;ZI 区在 RAM 里面的起始地址
IMPORT |Image$$ZI$$Limit| ;ZI 区在 RAM 里面结束地址后面的一个地址
LDR r0,=|Image$$RO$$Limit|
LDR r1,=|Image$$RW$$Base|
LDR r3,=|Image$$ZI$$Base|
CMP r0,r1
BEQ %F1
0 CMP r1,r3
LDRCC r2,[r0],#4
STRCC r2,[r1],#4
BCC %B0
1 LDR r1,=|Image$$ZI$$Limit|
MOV r2,#0
2 CMP r3,r1
STRCC r2,[r3],#4
BCC %B2
```

程序实现了 RW 数据的拷贝和 ZI 区域的清零功能。程序先把 ROM 里以|Image$$RO$$Limt|开始的 RW 初始数据拷贝到 RAM 里面|Image$$RW$$Base|开始的地址，当 RAM 这边的目标地址到达|Image$$ZI$$Base|后就表示 RW 区的结束和 ZI 区的开始，接下去就对这片 ZI 区进行清零操作，直到遇到结束地址|Image$$ZI$$Limit| 为止。

3.5 跳转到主应用程序

当系统初始化工作完成之后，就需要把程序流程转入主应用程序。

最简单的一种情况是直接跳到自定义的主函数，函数名由用户定义，例如：

```
IMPORT Main
BL Main
```

在 ARM ADS 环境中，还另外提供了一套系统级的呼叫机制。

```
IMPORT__main
BL__main
```

其中__main()是编译系统提供的一个函数，负责完成库函数的初始化和初始化应用程序执行环境，最后自动跳转到 main()函数。这种情况下用户程序的主函数名必须是 main。

至此，系统软硬件已进入到合适的状态，用户可以进行主应用程序的开发了。

4 结束语

在嵌入式系统软件设计中，BootLoader 的编写至关重要。性能良好的 BootLoader，可以大大提高系统的实时性和稳定性。文中开发的 BootLoader 已成功应用于一款无线通信设备，所介绍的原理和设计与实现的方法，对设计和移植到其他类型的嵌入式系统有一定的参考价值。

参考文献：

[1] 万永波，张根宝. 基于 ARM 的嵌入式系统 Bootloader 启动流程分析[J]. 微计算机信息，2005，21(11—2)：90－92.

[2] 李亚锋，欧文盛. ARM 嵌入式 Linux 系统开发从入门到精通[M]. 北京：清华大学出版社，2007：48－84.

[3] ATMEL Corporation. ARM920T－based Microcontroller AT 91 RM9200 Guide[M]. [s.l.]：[s.n.]，2006：83－94.

[4] 潘 浩，马艳敏. Bootloader 在 AT91RM9200 系统中的实现[J]. 微计算机信息，2007，23(1－2)：168－170.

[5] 王宇行. ARM 程序分析与设计[M]. 北京：北京航空航天大学出版社，2008：213－232.

(上接第 157 页)

[3] Friedman N, Nachman I, Pe'er D. Learning Bayesian Network Structure from Massive Datasets: The "Sparse Candidate" Algorithm[C]//In: Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI). [s. l.]: Morgan Kaufmann, 1999：206－215.

[4] Tsamardinos I, Aliferis C F, Statnikov A. Time and Sample Efficient Discovery of Markov Blankets and Direct Causal Relations[C]//In: The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2003). [s.l.]: ACM, 2003：673－678.

[5] Tsamardinos I, Brown L E, Aliferis C F. The Max－Min Hill－Climbing Bayesian Network Structure Learning Algorithm [J]. Machine Learning, 2006, 65：31－78.

[6] 杨庆平. 基于贝叶斯网络的基因调控网络构建算法的研究[D]. 哈尔滨：哈尔滨工业大学，2006.

[7] 张连文，郭海鹏. 贝叶斯网引论[M]，北京：科学出版社，2006：186－188.

[8] 张剑飞. 贝叶斯网络学习方法和算法研究[D]. 长春：东北师范大学，2005.