# 嵌入式系统引导程序详探

陈海军,申卫昌,史 颖
(西北大学 计算机科学系,陕西 西安 710069)

**摘 要**:在32位嵌入式系统应用程序开发中的主要技术难点在于系统启动程序的编写,因此文中详细论述了在ARM7-S344BOX基础上开发嵌入式系统时启动程序的原理及实现的方法。最后介绍了一种轻量级的、便于移植的启动程序的实现。经调试运行,该启动程序运行良好。

**关键词**:BootLoader;I/O;存储映像;内核;异常向量
**中图分类号**:TP311.1 **文献标识码**:A **文章编号**:1005-3751(2006)01-0123-03

# Research of Boot Program in Embedded System

CHEN Hai-jun, SHEN Wei-chang, SHI Ying
(Dept. of Computer Sci. and Tech., Northwest Univ., Xi'an 710069, China)

**Abstract**: The main difficulty of application development in the 32bit embedded system is lying on the Bootloader program. To this end, describe the theory and implementation of Bootloader in an embedded system based on ARM7-S344BOX system. Finally, introduces a kind of Bootloader which is lightweight and easy transplanted.

**Key words**: BootLoader; I/O; memory map; kernel; exception vector

## 1 BootLoader概述

在嵌入式操作系统运行起来之前需要一段程序来初始化CPU、内存控制器、看门狗等硬件设备,然后配置存储映射,建立系统的内存空间映射图,从而将系统软硬件设置到合适状态,最后将操作系统内核解压加载到RAM中,并且跳转到操作系统的起始代码部分去执行,这段程序称为BootLoader,其作用类似于PC上的BIOS。文中分析了BootLoader的引导过程。

## 2 BootLoader分析

### 2.1 BootLoader工作简介

BootLoader对硬件依赖性比较强,即使是基于同一种CPU构造的不同的硬件系统,通常也需要修改BootLoader。为了提高BootLoader的可重用性和可移植性,一般将BootLoader程序分为两个阶段,将与硬件密切相关,不便于移植的部分放在第一个阶段,主要以汇编语言编写。将与硬件无关,比较容易移植的部分放到第二个阶段,以C语言编写,以提高程序的通用性和可读性[1~3]。

● 第一个阶段将被映射到CPU启动代码处,这个阶段的程序主要做的工作是:

①设置异常向量表,为启动过程中用到的中断和异常做准备;

②初始化CPU、IO端口、看门狗等硬件设备,将其置于一个合适的状态;

③配置存储控制器,为BootLoader的第二个阶段的代码准备内存空间;

④设置堆栈,为运行接下来的第二个阶段的C语言程序做准备;

⑤将BootLoader的第二个阶段的代码复制的内存空间中;

⑥跳转到第二个阶段的C语言入口。

● 第二个阶段程序主要做的工作是:

①初始化本阶段要用到的硬件设备(例如UART口等);

②配置存储映像,建立系统的内存空间映射图;

③解压并加载操作系统内核到内存中;

④设置操作系统内核启动参数;

⑤跳转到操作系统内核入口。

### 2.2 硬件平台简介

文中分析的硬件平台的CPU是三星公司生产的S344BOX(下面如不特殊注明的话,都指的是44BOX的硬件系统),它的core是ARM7TDMI,在片内集成了很多外围接口,如LCD控制器、串行接口、I2C总线、PWM接口和JTAG接口等。但在其上没有存储管理单元(MMU),这导致在其上运行的操作系统不能采用虚拟存储、内存地

收稿日期:2005-04-03
作者简介:陈海军(1974—),男,内蒙古五原人,硕士研究生,研究方向为嵌入式系统、计算机应用技术;申卫昌,副教授,研究方向为并行计算、嵌入式系统。

址转换和存储保护等存储管理机制[4~6]。文中分析的 BootLoader 引导的操作系统是 μCLinux。

### 2.3 实例分析

(1)下面首先来分析一下 BootLoader 的第一个阶段。

①设置异常向量表,为启动过程中用到的中断和异常做准备。

当系统复位的时候,nReset 信号被置为高电平,此时 CPU 将当前的 PC 值和 CPSR 的值写入 R14_svc 和 SPSR_svc,再将 CPSR 的 M[0]~M[4]置为 10011,即 Supervisor 模式,同时将 I 位和 F 位置为一,将 T 位清零,强制 PC=0x00000000(使 CPU 从地址为 0 的地方取指令),并将 CPU 置为 ARM 状态。因此要在以地址 0 处开始的地方放置异常向量表,其中第一条是复位跳转语句,让其进入硬件初始化程序。异常中断向量表是 BootLoader 和 μCLinux 操作系统内核发生联系的关键之处,当操作系统发生异常时,它将进入异常中断向量表读取相应的指令表项进行处理,因为 μCLinux 已经在内存中 0x0C000000 处建立了自己的二级异常中断向量表,所以在这里除了复位中断以外的其它异常中断可以简单地跳转到相应的二级异常中断向量表处[7,8]。

```
;_start /* 此处表明为 BootLoader 开始的地方 */
0x00000000 B Reset /* 复位的时候跳转到 Reset 处开始执行 */
0x00000004 Ldr pc,=0x0C000004/* 未定义指令异常 */
0x00000008 Ldr pc,=0x0C000008/* 自陷指令异常 */
0x0000000C Ldr pc,=0x0C00000C/* 预取指令失败异常 */
0x00000010 Ldr pc,=0x0C000010 /* 预取数据失败异常 */
0x00000014 B /* 保留字 */
0x00000018 Ldr pc,=0x0C000018 /* 中断请求异常 */
0x0000001C Ld pc,=0x0C00001C /* 快速中断异常 */
```

②复位代码初始化 CPU、IO 端口、看门狗等硬件设备,将其置于一个合适的状态。

在系统初始化的过程中,必须关闭看门狗的定时器。同时由于 S344BOX 有 71 个多功能复用的 IO 端口,从 A~G 分为 7 组,必须将其根据硬件设备配置到合适的状态,接着关闭中断,然后将 CPU 的运行频率设置正确[1,7,8]。

```
;Reset
    /* 关闭看门狗时钟寄存器 */
    ldr r0,=WTCON
    ldr r1,=0x0
    str r1,[r0]
    /* 初始化 IO 端口,共 71 个端口,从 A~G 分为 7 组 */
    /* Port A */
    ldr r1,=PCONA
    ldr r0,=0x1ff
    str r0,[r1]
    ...
/* 将外中断设为上升沿触发 */
    ldr r1,=EXTINT
    ldr r0,=0x44444444
    str r0,[r1]
/* 关闭中断 */
    ldr r1,=INTMSK
    ldr r0,=0x03fffeff
    str r0,[r1]
/* 设置为非矢量中断模式 */
    ldr r1, =INTCON
    ldr r0, =0x05
    str r0,[r1]
/* 清除以前的所有中断 */
    ldr r1,=I_ISPC
    ldr r0,=0xffffffff
    str r0,[r1]
```

在设置 CPU 频率之前,需要设置锁相环输出稳定时间计数器,起码要大于 208μs,即每次更改锁相环的主频时,必须经过一定时间后才能达到稳定,稳定时间>计数器值×1/Fin (Fin 为外部晶振频率)

```
    ldr r1,=LOCKTIME
    ldrb r0,=0xfff
    strb r0,[r1]
```

根据硬件条件设置 CPU 主频,锁相环输出频率 Fpllo=(m * Fin)/(p* $2^s$),m=(MDIV+8),p=(PDIV+2),s=SDIV,其中 MDIV 是 PLLCON 的 12~19 位,PDIV 是 PLLCON 的 4~9 位,SDIV 是 PLLCON 的 0~1 位,按照 s344b0 的手册,这些位的设置必须满足以下条件:

a. Fpllo 必须大于 20MHz,小于 66MHz;

b. Fpllo×$2^s$ 必须小于 170MHz;

c. 与此同时,s 应该尽可能大;

d. (Fin/p)的值应该大于 1MHz,小于 2MHz。

这里使用的硬件是 64MHz 的 S344BOX,晶振频率 Fin 为 8MHz,所以设置 PLLCON 为 0x38021,即 MDIV=0x38=56,PDIV=2,SDIV=1,这样主频 Fpllo=(56+8)×8/((2+2)×$2^1$)=64

```
    ldr r1,=PLLCON
    ldr r0,=0x38021
    str r0,[r1]
```

③配置存储控制器,为 BootLoader 的第二个阶段的代码准备内存空间。

需要配置数据存储格式是小/大尾格式,配置每个 bank 的数据宽度。

```
    ldr r0,=MEMORY_CONFIG
    ldmia r0,{r1-r13}
    ldr r0,=0x01c80000
    stmia r0,{r1-r13}
```

④设置堆栈,为运行接下来的第二个阶段的 C 语言程序做准备。

```
    mrs r0,cpsr
    bic r0,r0,#0x1f
    orr r1,r0,#0xdb
```

```
msr cpsr_cxsf,r1
adr sp,irqstack
mrs r0,cpsr
bic r0,r0,#0x1f
orr r1,r0,#0xd7
msr cpsr_cxsf,r1
adr sp,irqstack
... ...
mrs r0,cpsr
bic r0,r0,#0x80
msr cpsr,r0
```

⑤将 BootLoader 的第二个阶段的代码复制的内存空间中。

其中 r0 中存放的是代码在 flash 中存放的地址,r1 中存放的是在 ram 中的目的地址,r2 中存放的是代码在 flash 中的结束地址。

```
copy_loop:
    ldmia r0!,{r3-r10}
    stmia r1!,{r3-r10}
    cmp r0,r2
    ble copy_loop
```

⑥ 跳转到第二个阶段的 C 语言入口。

为了防止从 C 语言主函数返回,采用了蹦床形式(trampoline)的程序结构。

```
_trampoline:
  Bl main
  B_trampoline
```

2)第二个阶段程序主要做的工作是:

①初始化本阶段要用到的硬件设备(UART 口等)。

由于 UART 口的时钟产生信号是 CPU 内部时钟经过 16 分频以后的信号再被一个计数器进行分频,因此该计数器的值应该设为(CPUCLK/(baud×16))-1 取整的值。

```
rUFCON0=0x0; //禁用 FIFO
rUMCON0=0x0; //不使用流控
rULCON0=0x3; //无奇偶校验,1 位停止位,8 位数据位
rUCON0=0x245;//rx 脉冲触发,tx=电平触发,采用中断或轮询方式
rUBRDIV0=((int)(CPUCLK/16./baud + 0.5)-1);
```

②配置存储映像,建立系统的内存空间映射图。

```
for(addr=MEMORY_START;addr<MEMORY_END;addr+=TEST_BLOCK_SIZE){
    if(testram(addr)==0){
      if(*(u32 *)addr!=0){
        if(memory_map[i].used)
          i++;
        continue;
    }
    if(memory_map[i].used==0){
        memory_map[i].start=addr;
        memory_map[i].len=TEST_BLOCK_SIZE;
        memory_map[i].used=1;
    }else{
        memory_map[i].len+=TEST_BLOCK_SIZE;
    }
  }else{
    if(memory_map[i].used==1)
        i++;
    }
  }
```

③解压并加载操作系统内核到内存中。

```
while(n--)
  *dest++=*src++;
```

④设置操作系统内核启动参数。

```
params=(struct tag *)BOOT_PARAMS;
params->hdr.tag=ATAG_CORE;
params->hdr.size=tag_size(tag_core);
params->u.core.flags=0;
params->u.core.pagesize=0;
params->u.core.rootdev=0;
params=tag_next(params);
```

⑤跳转到操作系统内核入口。

直接调用操作系统入口处函数。

```
Mov pc,#KERNEL_RAM_BASE
```

## 3　平台和实例的优劣比较

Samsung 公司的 S344BOX 微控制处理器为手持设备和一般应用提供了一种高性价比和高性能的方案,相对于其它以 ARM7TDMI 为内核的芯片来说,S344BOX 芯片具有价格低廉、接口丰富和在国内缓用最为广泛的特点[4,5],因此,这里的硬件平台选择了 S344BOX。启动程序 BootLoader 的移植主要参考了 BLOB-2.0.5,与其它通用 BootLoader 如 RedBoot,u-Boot 等比较起来,它的特点是程序可读性较好,可移植性较高。

## 4　结束语

经过调试运行,该启动程序运行良好,可以正常加载 μCLinux 的操作系统。文中介绍的 BootLoader 系统可移植性较强,经过对底层的简单修改就可以应用于其它的 ARM 硬件系统。

**参考文献:**

[1] 陈　渝,李　明,杨　畔.源码开放的嵌入式系统软件分析与实践——基于 SkyEye 和 ARM 开发平台[M].北京:北京航空航天大学出版社,2004.

[2] 王利明.Arm linux 启动分析[EB/OL].http://www.codelphi.com/faye/archive/2004/10/21/26320.aspx,2004.

[3] 詹荣开.嵌入式系统 Boot Loader 技术内幕[EB/OL].http://www.cnarm.com/Article_Print.asp?ArticleID=44,2004.

(下转第 128 页)

随着节点负载的变化，均衡器对节点权值不断进行动态调整。

在自适应负载均衡算法的设计中，笔者周期性地从各个节点中收集以下参数：

- CPU 利用率 $Cpu(N_i)\%$
- 内存利用率 $Memory(N_i)\%$
- 当前网络流量 $T(N_i)$
- 磁盘 I/O 访问率 $Io(N_i)\%$
- 响应时间 $Rt(N_i)$
- 进程总数 $Pr(N_i)$

将这些参数作为计算公式的因子。结合每个节点当前的权值，可以计算出新权值的大小。动态权值目的是要正确反映节点负载的状况，以预测节点将来可能的负载变化。对于不同类型的系统应用，各个参数的重要程度也有所不同。为了方便在系统运行过程中针对不同的应用对各个参数的比例进行适当调整，为每一个参数设定一个常量系数 $\pi_i$，用来表示各个负载参数的重要程度，其中 $\Sigma\pi_i = 1$。因此，任何一个节点 $Ni$ 的权值公式就可以描述为：

$$Load(N_i) = \pi_1 * Cpu(N_i)\% + \pi_2 * Memory(N_i)\% + \pi_3 * T(N_i) + \pi_4 * Io(N_i)\% + \pi_5 * Rt(N_i) + \pi_6 * Pr(N_i)$$

另外，虽然很短的周期可以更确切地反映各个节点的负载，但是很频繁地采集（如 1 秒 1 次或者多次）会给均衡器和节点带来负担，也可能增加不必要的网络负荷。另外，由于采集器是在采集时刻进行负载计算的，经实验证明，均衡器反映出来各个节点的负载信息会出现剧烈的抖动，均衡器无法准确捕捉节点真实的负载变化趋势。因此解决这些问题，一方面要适当地调整采集负载信息的周期，一般在 5 ~ 10 秒；另一方面，可以使用移动平均线或者是滑动窗口来避免抖动，使得均衡器收集到的负载信息表现为平滑曲线，这样在动态反馈机制的调整效果上就会比较好。

均衡器的动态权值采集程序周期性地运行，若缺省权值不为零，则可查询该节点的各负载参数，并计算出动态权值 $Load(N_i)$。引入以下权值计算公式，结合节点的初始权值和采集的动态权值来计算最终的权值结果。

$$W(N_i) = A * IW(N_i) + B * (Load(N_i) - IW(N_i))1/4$$

在公式中，如果动态权值恰好等于初始权值，最终权值不变，则说明系统的负载状况刚好达到理想状况，等于初始权值 $W(N_i)$。如果动态权值计算结果高于初始权值，最终权值变高，则说明系统负载很轻，均衡器将会增加分配给该节点的任务比率。如果动态权值低于初始权值，最终权值变低，说明系统开始处于重载状况，均衡器将会减少对该节点分配的任务。在实际使用中，若发现所有节点的权值都小于它们的 $W(N_i)$，则说明当前集群处于超载状态，这时需要加入新的节点到集群中来处理部分负载；反之，若所有节点的权值大大高于 $W(N_i)$，则说明当前系统的负载都比较轻。

## 3 结 论

校园网格在高校教学科研中的重要性已经初见端倪，负载均衡技术的研究在计算机领域也是方兴未艾，一种优良的负载均衡算法的设计往往与系统的体系结构、网络流量特性、流量模型有关，它一般只在某些特殊的应用环境下才能发挥最大效用。因此在应用时可根据实际系统的访问特性结合系统中各服务器站点的响应能力，把不同的算法策略和技术结合起来使用，使整个校园网格的服务器集群系统的性能得以充分发挥。

**参考文献：**


[1] Foster I, Kesselman C, Tuecke S. The anatomy of the Grid: Enabling scalable virtual organizations[J]. International Journal Supercomputing Application, 2001,15(3):200 - 222.

[2] Foster I, Kesselman C. 开放网格服务体系结构. 网格计算（第 2 版）[M]. 金 海，等译. 北京：电子工业出版社，2004.125 - 139.

[3] Dias D M, Kish W, Mukherjee R, et al. A Scalable and Highly Available Web Servers[A]. Proc of 41st IEEE Computer Society Intl. Conf, Computer Conference, IEEE International (COMPCON)[C]. [s.l.]:[s.n.],1997.85 - 92.

[4] Schroeder T, Goddard S, Ramamurthy B. Scalable Web Server Clustering Technologies[J]. IEEE Network, 2000, 14(3):38 - 45.

[5] Cardellini V, Colajanni M. Dynamic load balancing on Web - server systems[J]. IEEE Internet Computing, 1999, 3(3):28 - 39.


（上接第 125 页）


[4] 马忠梅，李善平，康 慨，等. ARM 嵌入式处理器结构与应用基础[M]. 北京：北京航空航天大学出版社，2002.

[5] Furber S. ARM Soc 体系结构[M]. 田 泽，于敦山，盛世铭译. 北京：北京航空航天大学出版社，2002.

[6] 毛德操，胡希明. 嵌入式系统—采用公开源代码和 StrongARM/Xscale 处理器[M]. 杭州：浙江大学出版社，2003.

[7] UMSUNG 公司. S344BOX 手册[DB/OL]. http://www.hyesco.com/download/ARM/s3c44box.rar,2004.

[8] ARM 公司. ARM Software Development Toolkit Version 2.0 - Programming Techniques[DB/OL]. http://infoeng.ee.ic.ac.uk/gac1/Architecture/Progtech.pdf,2004.