基于 Blackfin 嵌入式系统的 U-boot 分析与调试

引言

Boot loader 代码是 CPU 芯片复位后进入操作系统之前执行的一段代码，主要用于完成硬件启动到操作系统启动的过渡，从而为操作系统提供基本的运行环境。Boot loader 代码与 CPU 的内核结构、具体型号、应用系统配置及操作系统有关，其功能类似于 PC 机的 BIOS 程序。

Blackfin DSP 是美国模拟器件公司与 Intel 联合开发的第 4 代 DSP 产品，是专为通信和互联网应用而设计的通用 DSP 芯片，适合处理互联网中的大量图像、声音、文本和数据流，以及汽车电子中的可视系统、宽带无线系统、消费类多媒体电子、数字摄像机、多通道 VoIP、安全和监督、机顶盒和视频电话会议等。 本文对基于 Blackfin 561 微处理器构建的嵌入式开发板 EZKIT561 的 U-boot 第一和第二阶段的具体工作流程进行了分析，画出了各阶段的流程图，同时在 U-boot 第一阶段代码中加入 LED 指示程序来跟踪第一阶段的执行情况；而在 U-boot 第二阶段，则在代码的相应位置添加了向串口的打印信息，以跟踪 U-boot 在此阶段的执行情况。

1 Blackfin DSP 简介

ADI 公司推出的 Blackfin 处理器是专为嵌人式音频、视频、通信计算要求和功耗约束条件而设计的新型 16～32 位嵌入式处理器。Blackfin 处理器由 ADI 和 Intel 联合开发，主要基于微信号架构(MSA)。它将一个 32 位 RISC 型指令集和双 16 位乘法累加(MAC)信号处理功能与通用型微控制器所具有的易用性组合在一起。这种处理特征使得 Blackfin 处理器在信号处理和控制处理应用中均能发挥上佳作用，因而在许多场合可免除增设单独异类处理器。

2 Boot loader 及 U-boot 简介

2．1 Boot loader 简介

Boot loader 是用于初始化目标板硬件，可给嵌入式操作系统提供板上硬件资源信息，并进行装载、引导嵌人式操作系统运行的固件。最终，Boot Loader 会把操作系统内核映像加载到 RAM 中，并将系统控制权传递给它。

大多数 Boot Loader 都包含两种不同的操作模式： "启动加载"模式和"下载模式"。

启动加载(Boot loading)模式也称"自主"(Autonomous)模式。即 Boot Loader 从目标机上的某个固态存储设备上将操作系统加载到 RAM 中运行。这种模式是 Boot Loader 的正常工作模式，因此在嵌入式产品发布的时侯，Boot Loader 显然必须工作在这种模式下。

在下载(Downloading)模式下，目标机上的 Boot Loader 将通过串口连接或网络连接等手段从主机(Host)下载内核映像和根文件系统映像等。从主机下载的文件通常先被 Boot Loader 保存到目标机的 RAM，然后再被 Boot Loader 写到目标机的 FLASH 等固态存储设备中。Boot Loader 的这种模式通常在第一次安装内核与根文件系统时使用；此外，以后的系统更新也会使用 Boot Loader 的这种模式。工作于这种模式下的 Boot Loader 通常都会向它的终端用户提供一个简单的命令行接口。

2．2 U-boot 简介

U-Boot(全称 U niversal Boot Loader)是遵循 GPL 条款的开放源码项目。其源码目录和编译形式与 Linux 内核相似。事实上，不少 U-Boot 源码就是相应的 Linux 内核源程序的简化，尤其是一些设备的驱动程序，这一点从 U-Boot 源码的注释中就能体现。目前支持的目标操作系统有 OpenBSD，NetBSD，FreeBSD，4．4BSD，Linux，SVR4，Esix，Solaris，Irix，VxWorks，LynxOS，pSOS，QNX，RTEMS，ARTOS；这是 U-Boot 中 Universal 的一层含义，另外一层含义是 U-Boot 除了支持 PowerPC 系列处理器外，还能支持 Black-fin、MIPS、x86、ARM、NIOS、XScale 等诸多处理器。上述两个特点正是 U-Boot 项目的开发目标，即支持尽可能多的嵌入式处理器和嵌人式操作系统。U-Boot 的主要目录结构如表 1 所列。

3 基于 Blackfin DSP 的 U-boot 运行分析

大多数 Boot loader 都分为 stagel 和 stage2 两大部分，U-boot 也是如此。

3．1 U-boot 的 stagel 阶段

依赖于 CPU 体系结构的代码(比如设备初始化代码等)，通常都放在 stage1 中，该代码可用 blackfin DSP 汇编语言来实现，以达到短小精悍的目的。实际操作可在位于 U-boot 1．1．3＼cpu＼bf533 中的 Start．S 和 Startl．S 文件中实现，而且是从 Start．S 开始运行，此阶段的程序流程图如图 1 所示。

表1　U-boot主要目录结构

目录	相关内容
board	目标版相关文件，主要包含SDRAM、FLASH驱动
common	独立于处理器体系结构的通用代码，如内存大小探测与故障检测
cpu	与处理器相关的文件。如mpc8xx子目录下含串口、网口、LCD驱动及中断初始化等文件
driver	通用设备驱动，如CFI FLASH驱动（目前对INTEL FLASH支持较好）
doc	U-Boot的说明文档
examples	可在U-Boot下运行的示例程序；如hello_world.c,timer.c
include	U-Boot头文件；尤其configs子目录下与目标板相关的配置头文件是移植过程中经常要修改的文件
lib_xxx	处理器体系相关的文件，如lib_ppc, lib_arm目录分别包含与PowerPC、ARM体系结构相关的文件
net	与网络功能相关的文件目录，如bootp,nfs,tftp
post	上电自检文件目录。尚有待于进一步完善
rtc	RTC驱动程序
tools	用于创建U-Boot S-RECORD和BIN镜像文件的工具

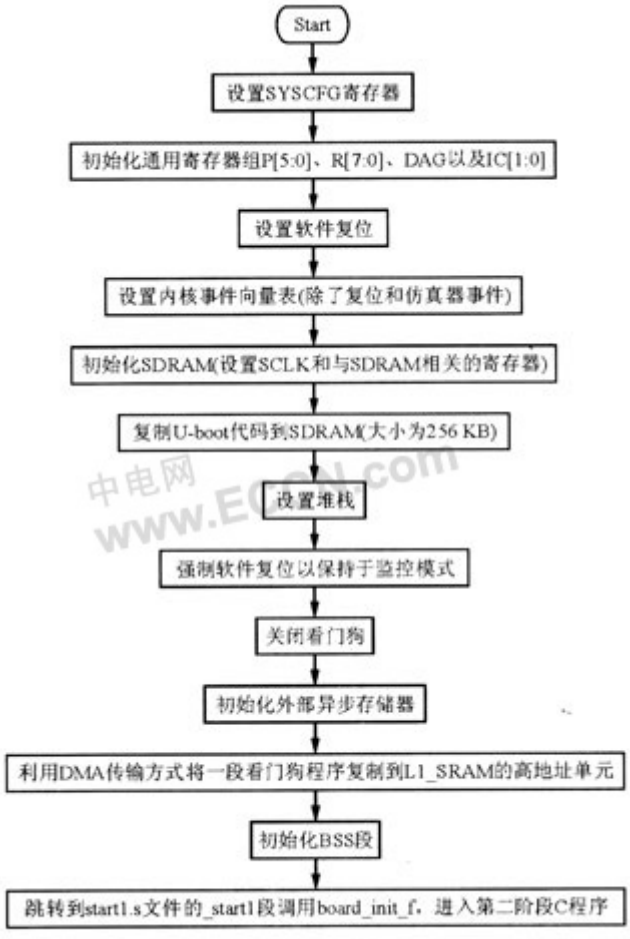

图1　第一阶段程序流程图

stage1 的步骤以执行的先后主要包括硬件设备初始化、为加载 Boot Loader 的 stage 2 准备 RAM 空间、拷贝 Boot Loader 的 stage2 到 RAM 空间、设置堆栈、跳转到 stag2 的 C 入口等。

3．2 U-boot 的 stage2 阶段

通常 stage2 可用 C 语言来实现更复杂的功能，该代码具有更好的可读性和可移植性。Stage2 实现的主要功能包括初始化本阶段要使用到的硬件设备，检测系统内存映射(memory map)，将 ker-nel 映像和根文件系统映像从 flash 上读到 RAM 空间中，为内核设置启动参数，以及调用内核等。

而基于 ADSP-BF561 EZKIT-Lite 评估板的 U-boot 在该阶段的启动时，首先在第一阶段结束后，调用＼U-boot 1．1．3＼1ib_blackfin＼board．c 文件中的 board_init_f()函数并执行。

然后再调用 board．c 文件中的 board_init_r()函数并按先后顺序执行，其流程图如图 2 所示。

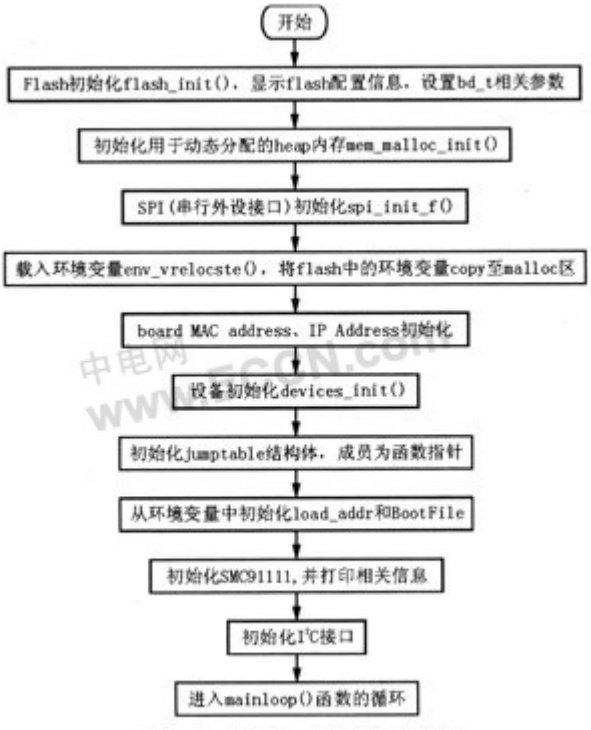

图2 board_init_r () 函数流程图

之后，再在 board_init_r 函数的最后调用＼U-boot 1．1．3＼common＼main．c 中的 main_loop()函数。在执行过程中，系统会首先对自动启动内核进行倒计时，倒计时的时间

由环境变量 bootdelay 的设定值决定。由于先前已经对串口进行了初始化，所以会在 windows 超级终端打印"ezkit：>"，这样，mainloop()函数的执行将产生两条分支：一是等待 u-boot 的自启动命令执行，即执行 bootcmd 环境变量所设定的自动运行的命令(比如 setenvbootcmd bootm 0x2000 0000)，引导 flash 特定地址中的嵌入式操作系统；二是在 u-boot 的自启动命令执行前按下任意键，以进入 u-boot 的命令行。在此状态下可以查看和修改环境变量、下载更新 U-boot 和内核镜像文件、对 flash 进行擦写操作或通过命令启动操作系统(如 bootm 0x2000 0000)；

4 基于评估板的 U-boot 启动跟踪调试

4．1 第一阶段跟踪调试

由于 U-boot 的启动过程分为两个阶段，第一阶段在串口初始化之前无法获得字符串提示信息。这样，第一阶段的运行过程似乎就没办法掌握．但是，EZKIT561 开发板提供 16 个用户可编程的 LED，所以就可以通过这 16 个 LED 来了解 u-boot 在第一阶段的具体执行过程，即在 U-boot 第一阶段的几个不同的代码处添加 LED 指示程序。

ADSP-BF561 有 48 个双向通用可编程 I / O 引脚。这些可编程引脚具有实现 SPI 接口的特殊功能。每一个可编程引脚均能通过操作一系列的标志控制寄存器、标志状态寄存器和标志中断寄存器来进行独立控制。由于一共有 48 个通用可编程 I / O 引脚，所以可将以上寄存器分成三组，每组可对 16 个通用可编程 I / O 引脚进行操作。

通过参考 EZKIT56 1 原理图可知，LED 5～20 与 PF 32～47 pins 相连，可以跟踪堆栈配置。设计时可使用以下寄存器进行控制。

(1)FIO2_DIR 寄存器

这是一个 16 位寄存器，若将其中的某一位设置为 1，那么相应的 PF 引脚就可作为输出；反之，则为输入。其相关设置代码如下：

```
p0.l = lo (FIO2_DIR) ;
p0.h = hi (FIO2_DIR) ;
r0.l = 0xFFFF; /*此处将PF32—PF47 pin设置为1，即为输出*/
w [p0] = r0;
ssync;
```

(2)FIO2_FLAG_D

这也是一个 16 位寄存器，对其写操作时，可指定相应的 PF 引脚状态；而当进行读操作时，则返回相应的 PF 引脚的值。它的每一位都控制着一个 LED 灯。其相关设置代码如下：

```
p0.l = lo (FIO2_FLAG_D) ;
p0.h = hi (FIO2_FLAG_D) ;
r0.l = 0xXXXX;/*此处对具体LED进行控制*/

w [p0]=r0;
ssync;
```

添加的跟踪堆栈配置程序的流程图如图 3 所示。修改代码后即可在 U-boot 文件夹路径下依次输入以下命令：make clean、make mrproper、make ezkit561 config 和 make，然后再利用 bfin-u．clinux-objcopy 将生成的 U-boot．bin 转换为 U-boot．hex，最后通过 VDSP＋＋开发环境中 TOOL 下的 flashprogrammer 将 u-boot．hex 烧写到 flash 中，同时进行复位操作以观察 LED 的变化。

图3 堆栈配置跟踪程序流程图

本设计希望在堆栈配置前使 LED 13、LED 14 亮，其它 LED 灭，持续时间为 1 s；而在堆栈分配之后使 LED 11、LED 12 亮，其它 LED 灭，持续时间为 1 s。其实际的观察结果是，在复位之后，LED 13、LED 14 持续亮 1 s，接着 LED 11、LED 12 持续亮 1 秒，可见其完全达到了预期目标。

4．2 第二阶段跟踪调试

第二阶段是在进人 C 函数之后，就进行串口的初始化。之后，便可通过向串口打印信息来实时跟踪所启动的执行流程，以了解程序目前执行的具体部分或运行到哪一个阶段出现了问题。

下面以打印串口初始化完成信息为例。首先在 U-boot 第二阶段找到串口初始化的代码，即 U-boot / lib-blackfin / board．c 文件的 serial．init()函数，然后在此函数之后添加 printf("serial initializationis ok!＼n")以实现打印。其程序代码如下：

```
……
void board_init_f (ulong bootflag)
{
……
init_baudrate () ; /* initialze baudrate settings
```

```
*/
    serial_init () ;      /* serial communications
setup */
    console_init_f () ;
    printf ("serial initialization is ok! \n");
```

修改代码后的编译和下载步骤如前所示，调试时使用的串行通信软件是 windows 自带的"超级终端"程序，所选择的"每秒位数" (即波特率)为 57600，传输文件使用的通信协议为 Kermit 协议。配置好超级终端后，按下开发板上的复位键．终端便可显示出系统启动过程的相关信息。系统复位后，第一行显示的是"serial initializa-tion is ok!"。这便是自行添加的打印语句，其主要功能是提示串口初始化已完成。

由于第二阶段可以通过串口打印信息，且在相关的每一阶段均可添加相关的 printf 句来实现打印提示信息，所以跟踪及调试都比较容易。

5 结束语

通过分析基于 Blackfin 561 微处理器构建的嵌入式开发板 EZKIT561 的 U-boot 代码，以期对 Boot Loader 的启动过程有一个比较深入的理解，文章还通过一些调试方法对其运行阶段进行跟踪，以便对将来在开发板上的 ucLinux 移植和进一步的视频编解码工作进行准备。