针对嵌入式系统 BootLoader 的中断介入调试技术的研究

Research on Debugging Techniques for Bootloader by Interruption

刘 科 雷跃明 (重庆大学软件学院 重庆 400044)

摘 要: 系统引导程序(BootLoader)是嵌入式系统开发的关键之一。长期以来它的开发都依赖于昂贵的硬件调试器以及低效率的打印调试方法。本文在分析了 BootLoader 程序的结构与特征以及时间中断的相关细节后,提出以时间中断介入调试的纯软件调试解决方案。该方案不仅适用于 BootLoader 程序的调试,也同样可以运用到单片机程序调试中。

关键词: 引导程序 调试 时间中断 存软件

1 引言

嵌入式 Linux 系统从软件的角度看通常可以分为四个层次:

引导程序,即 boot loader 、Linux 内核 、文件系统、用户程序。

BootLoader 是系统启动上电后运行的第一段软件代码,它承担着硬件初始化、内核启动、建立内存空间的映射图等重要任务。由于严重依赖于硬件,致使不可能建立统一通用的引导程序。随着嵌入式软硬件的飞速发展,BootLoader 所承担的功能越来越多,对其的扩展和二次开发也越来越频繁,而目前调试手段仍然以 BDI2000 等硬件调试器为主。硬件调试器虽然功能强大,但价格昂贵,而且在对引导程序进行功能扩展的时候并不一定需要硬件调试器。本文在分析了 BootLoader 的程序特征和时间中断介入调试的可行性后,提出了一种针对 BootLoader 的利用时间中断介入调试过程的纯软件调试解决方案。

2 多阶段 BootLoader 分析与调试策略

2.1 多阶段 BootLoader

多阶段的 BootLoader 能提供更为复杂的功能和更好的可移植性。目前 BootLoader 多以两阶段结构为主,其中第一阶段提供了必要的硬件初试化,第二阶段提供了复杂的功能扩展。两阶段启动方法,一是提高了移植性和扩展性;另一方面是因为 ARM 处理器冷启动时装载到片内 SRAM 中执行的代码尺寸需小于片内 SRAM 的大小(多为 16KB),这部分空间对于实现一个功能完善的引导程序显得太小,因此将引导程序分成两个阶段。第一阶段的代码被自动装载到片内 SRAM 中执行,初始化片外 SDRAM,然后第二阶段的代码就可以被装载到片外 SDRAM 中执行,由第二阶段完成以后的工作。

第一阶段代码只是做出必要的硬件初始化操作,具有代码短小,逻辑简单且以汇编实现完成的特点,并且这部分的代码主要作用与硬件,可以通过硬件所反映出来的情况进行查错,所以在第一阶段里没有必要辅助调试方法。

2.2 第二阶段的分析

2.2.1 第二阶段的数据

(1) 数据的分类与存放

嵌入式 BootLoder 第二阶段的代码是以 C 语言为主且很少运用到浮点型变量和负值,所以其中所涉及的数据对象可以分为以下三类:

值变量,如: int a = 10;

指针变量,如:short * b;

常量地址宏,如:

#define DMA (* (unsigned *)0x30)

结构体类型和数组都是以上的集合形式。

第二阶段的代码用 gnu 工具进行编译后生成的目标文件缺省为 elf 格式，其中的可写的变量存放于：

- .text 段，包含程序的指令代码
- .data 段，包含以非 0 值初始化了的静态变量和全局变量
- .bss 段，清零后的静态变量和全局变量
- .common 段，未初试化的静态变量和全局变量符号
- 动态存放在堆栈中局部变量的地址（遵守 ATPCS 规则）

（2）数据对象的描述与信息输出

C 语言中在对数据对象进行索引是从地址开始，然后根据特征和类型得到所需要的值，所以通过一个数据来完全描述以上分析的数据对象：

```
enum TYPE{MAC=0, CHAR=1, SHORT=2, INT=4};
struct Data_Info {
    char* name; /* 变量名 */
    intaddress; /* 变量存放地址值 */
    TYPE type;/* 变量类型 */
    int level;/* 变量级别 */
};
```

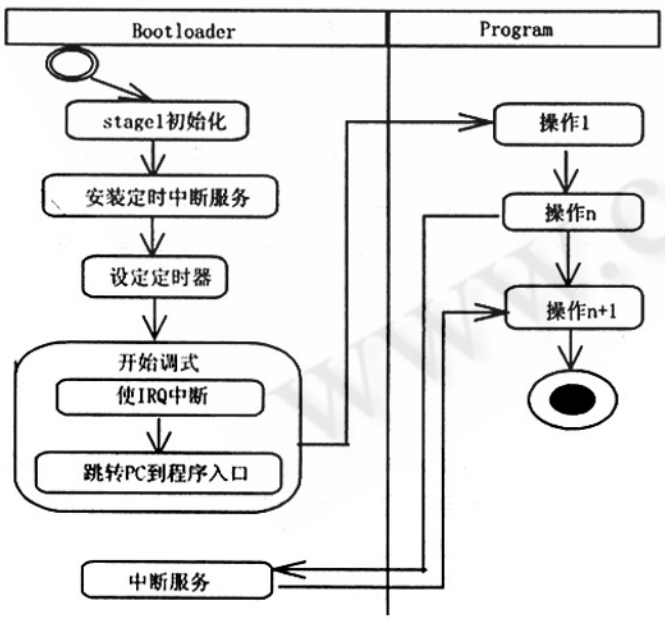

图 1　动态反映方法活动图

如果是值变量，则 level 为 0；如果是 1 级指针，则 level 为 1；如果是 n 级指针，则 level 为 n。32 位嵌入式系统地址都是 32 位，所以不管是指针还是值变量都能通过一个 int 型变量（address）来保存所涉及的地址值。

通过以上结构对数据进行描述后，可以非常详细的输出数据对象的信息：

```
void Get_DATA_Info(struct* Data){
if(Data->level==0)
将 address 转换为相应类型的地址指针并输出值;
else{
输出 name、address、level;
Data->level=Data->level-1;
Data->address=*((int*)Data->address);
Get_DATA_Info(Data);
}
}
```

3　定时中断动态反映方法的分析

由上面的分析可以看出，可以通过一种通用的数据结构来描述数据对象，并且输出数据对象的地址、值、名称等信息。如果可以通过一种方法将数据的这些信息在程序执行的过程中动态的反映出来，这样就可以达到动态观测程序以辅助调试的目的了。

3.1　定时中断动态反映方法的设计

linux 内核中采用了一种叫做分时调度的进程调度策略，其基本原理就是将时间划分为时间片，然后通过调度器分配给进程一定的时间片去运行，当进程所分得的时间完毕后，不管程是否完成，它都将放弃对 CPU 的占有权。将这一原理运用在这里后，需要调试的程序被看成一个需要运行的进程，分配给它一定的时间去运行，当运行时间完毕后对其内部的数据进行动态反

映。引导程序是直接运行在机器上的，没有构件在任何的操作系统上，所以不需要考虑进程优先级、竞态等复杂的情况。由此对于定时中断动态反映方法有如下设计，如图 1 显示。

(1) 安装定时中断服务程序

以 ARM9TDMI 为例，它提供了 5 个 16 位的定时器，每个定时器对应了一个 IRQ(Interrupt Request)中断源，在这里我们只需要对其中的一个设置相应的中断处理服务程序即可。在中断服务程序所需要处理的任务就是对开始建立的描述数据对象的通用数据结构进行输出。

(2) 设置定时器

定时器需要设定四个值：定时器输入时钟频率、定时器检查频率、定时器初始值、定时器检测值。

设定好 timer 后，定时器以设定的输入时钟频率递减定时器的初始值，每次减 1，然后设定的检查频率比较初始值和检测值，如果相等则出发 timer 中断，调用相应的中断服务程序。

(3) 打开中断并开始运行被调试程序

将 cprs 寄存器的第八位清零，使能 irq 中断；打开所设定的 timer 的中断掩码，是中断控制器能够处理该中断；将 pc 移到被调试程序的入口地址。

3.2 定时中断动态反映方法的可行性分析

3.2.1 CPU 频率

ARM9TDMI CPU 中有 3 个时钟信号源：

FCLK，对应 CPU 的时钟信号源、HCLK，对应 AHB(Advanced High performance Bus)总线时钟信号源、PCLK，对应到 APB(Advanced Peripheral Bus)总线时钟信号源，定时器是在 APB 上。

CPU 在 normal 模式下面时，FCLK = Mpll clock，Mpll clock 可以通过计算得到：

$Mpll = (m \times Fin) / (p \times 2^s)$

m = (MDIV + 8)，MDIV(Main Divider Control)

P = (PDIV + 2)，PDIV(Pre - divider Control)

S = SDIV，SDIV(Post Divider Control)

HCLK 和 PCLK 与 FCLK 的比例如表 1 所示。可以看出，PCLK ≤ FCLK，即 PCLK 总小于 CPU 时钟频率。

3.2.2 中断间隔

PCLK 是 Timer 的信号源，我们通过设置每个 Timer 相应的 Prescaler 和 Clock Divider 把 PCLK 转换成输入时钟信号传送给各个 Timer 的逻辑控制单元(Control Logic)，事实上每个 Timer 都有一个称为输入时钟频率(Timer input clock Frequency)的参数，这个频率就是通过 PCLK，Prescaler 和 Clock Divider 确定下来的，每个 Timer 的逻辑控制单元就是以这个频率在工作。下面给出输入时钟频率的公式：

表 1　HCLK 和 PCLK 与 FCLK 的比例

FCLK:HCLK:PCLK
1:1:1
1:1:2
1:2:2
1:2:3
1:4:4

TICF (Timer Input Click Frequency) = PCLK/(Ps + 1)C

Ps(Pr escaler) $\in \{x | 0 \leq x \leq 255\}$

C(Clock Divider) $\in \{2,4,8,16\}$

$\because$ PCLK ≤ FCLK

$\therefore$ TICF = PCLK/(Ps + 1)C

≤ FCLK/(Ps + 1)C

≤ PCLK/2

可以看出时间中断的频率最大不能超过 FCLK / 2，其时间间隔至少是 2 / FCLK(2 个时钟周期)。如果将初始值设为 counter = FCLK / 2，对比值设为 compare = FCLK / 2 - 1，那么时间中断在每 2 / FCLK 的间隔内产生一次。

3.2.3 时间问题

(1) 指令流水线

ARM9TDMI CPU 采用的是五级流水线：

取指令(fetch)、译码(decode)、执行(execute)、缓冲/数据(buffer/data)及 回写(write - back)

不用流水线，则 5 条指令需要 5 × 5 个时钟周期，采用了 5 级流水线后，每个时钟周期能完成一条指令，但一条指令仍然需要 5 个时钟周期来完成，所以有 5

个时钟周期的延时。

(2) 时间的不确定性

时间中断间隔的最小值是 2 / FCLK(2 个时钟周期),在不考虑总线耗时和延时的理想情况下每 1 个时钟周期就能完成一条指令,当用每 2 个时钟周期间隔的时钟中断来反映程序内部数据的变化情况时,只能反映出程序内部数据以 2 个周期为单位时间的状况。如果对同一地址进行多次写操作,那么会出现结果被覆盖不能反映的情况,如图 2 所示。

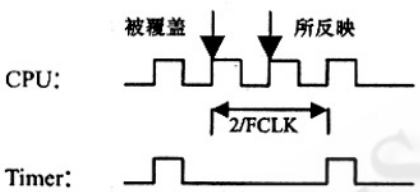

图 2

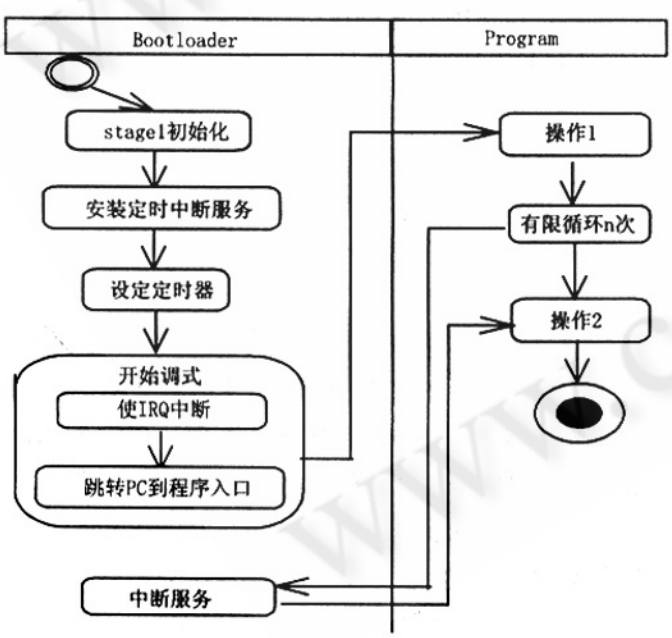

图 3　改进后动态反映方法活动图

并且,指令流水线如果遇到跳转、寄存器锁定等情况时,指令流水线会被扰乱以至于会出现延时,再加上总线的延时、外部设备等待响应等情况,致使无法精确的计算时间中断出现的准确位置,这样在输出变量信息的时候就会出现遗漏和产生多余信息。

3.2.4 解决办法

程序在运行的过程中是一个变化的过程,在整个过程中程序数据对象的变化非常频繁,如果能把程序进入可能存在 bug 的代码区域后某一时间点程序内数据对象的情况反映出来,就像设置断点一样,这样同样能够达到调试的目的了。解决的方法就是插入循环语句,使程序在此停留足够长的时间,以至于必然发生时间中断,并且能使程序停止运行直到时间中断服务程序执行完后在往前执行,如图 3 示。

4 结论

以中断介入调试的方法与用 BDI2000 等硬件调试器来调试程序的方法相比存在一些不足:

- 中断介入调试方法是被动调试方法,只能输出信息,缺乏交互性。
- 由于本身是运用中断来调试程序,没有考虑中断重入的情况,所以不能调试 IRQ 中断服务程序。

但中断介入调试技术是以纯软件的方式构建的一种调试方法,它的最终目标能为嵌入式系统 BootLoader 程序以及类似的单片机程序提供一种详实、低成本、多方位的调试方法。虽然中断介入调试存在一定的不足,但是可以看出完全可以运用中断介入方法来动态反映程序内部运行情况从而达到调试程序的这一目的。

参考文献

1 杜春雷, ARM 体系结构与编程[M], 北京:清华大学出版社, 2003.

2 Samsung Corporation. S3C2410B User's Manual Rev1.0 [Z]. 2002.

3 Arm Limited. ARM Architecture Reference Manual [Z]. http://www arm. com.

4 ARM Ltd.. ARM DAI 0107I Document [Z]. 2002.

5 康保祥, RISC 体系结构及其实现技术, 北京: 科学出版社, 1996 - 01.