doi: 10.7690/bgzdh.2014.08.018

一种改进的 TIC6xDSP 系统的快速自举方式

刘歆浏，周建平，李健，田瑞娟
（中国兵器工业第五八研究所军品部，四川 绵阳 621000）

摘要：为提高 DSP 的自举速度和程序运行效率，提出一种改进的 TIC6xDSP 系统的快速自举方式。以 TMS320C6455 为例，详细分析 TIC6xDSP 自举的基本原理、硬件接口设计、Flash 烧写固化以及软件实现全过程。针对现有解决方案存在的弊端，从加载模式设计、引导程序编写以及目标代码整合 3 方面进行改进。结果表明：该方法能极大地提高 DSP 的自举速度和程序运行效率，降低程序员对 DSP 的开发难度。

关键词：TIC6xDSP；自举；Flash；二级下载
中图分类号：TP311　　**文献标志码：**A

An Improved TIC6xDSP System Fast Boot Mode

Liu Xinliu, Zhou Jianping, Li Jian, Tian Ruijuan
(*Department of Military Products*, *No*. 58 *Research Institute of China Ordnance Industries*, *Mianyang* 621000, *China*)

Abstract: For accelerating DSP boot speed and improving program running efficiency, put forward an improved TIC6xDSP system fast boot mode. Taking TMS320C6455 as example, the whole process of TIC6xDSP's boot is analyzed, including fundamental principle, hardware and interface design, flash burn and software design. Meanwhile, aiming at the problem of existing solutions, 3 aspects are proposed for improving, such as loading mode design, boot programming and integration of destination code. The results show that the method can greatly accelerate DSP boot speed and improve program running efficiency, and reduce difficulties for DSP research and development.

Keywords: TIC6xDSP; boot; Flash; secondary loading

0　引言

随着信息技术的飞速发展，现今的高速 DSP 内存不再基于 Flash 结构，而是采用存取速度更快的 RAM 结构。在 DSP 掉电和重启后，其内部 RAM 中的程序和数据会全部丢失。完成 DSP 的软件开发，之后，最终的系统将要脱离仿真器独立运行。自举(bootloader)就是指系统加电或复位时，DSP 通过某种方式将存储在 Flash 的用户代码搬移到片内缓存或片外 SDRAM 中自动运行完成系统的自启动。

TI 的 DSP 有 3x、5x、6x 等多个系列，不同系列 DSP 的自举方法也不尽相同。如 TIC3x 系列 DSP 采用引导表，由固化在 DSP 内部的引导程序实现程序的自举；而 TIC6x 系列 DSP，由于硬件复杂度和主频更高，用户代码更大，则采用的是一种全新的、更为复杂的自举方法。笔者以 TMS320C6455 为例，分析 TIC6xDSP 系统的自举过程，并针对现有自举存在的不足进行了改进，提出了一种更为快速和便捷的自举方式。

1　自举的基本原理

TiC6xDSP 处理器的启动模式主要有 No boot、EMIF ROM/Flash boot 和 Host boot 3 种[1]。第一种模式是不加载数据，CPU 直接从存储器地址 0 处开始执行。第三种模式则是通过外围主机接口初始化 DSP 必要存储器空间并控制 DSP 启动。笔者主要对第二种模式的启动过程及加载方法进行分析。

在基于 Flash 模式的启动中，Flash 中的应用程序需要被自动加载到 DSP 的 RAM 中高速地运行。由于 C6455 系统的复杂性使应用程序比较庞大，而且没有片内 Flash，所以无法像 TiC2xDSP 那样将程序到片内 Flash 中，并在启动的时候单级引导就完成自举。而是通过扩展一个更大外部 Flash 芯片，通过开发二级引导完成自举。

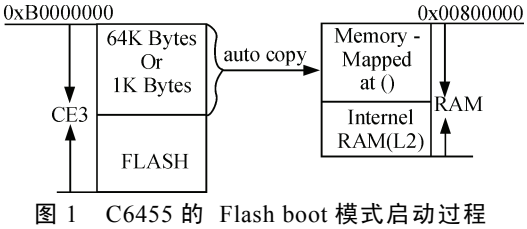

图 1　C6455 的 Flash boot 模式启动过程

如图 1 所示，EMIF ROM/Flash boot 模式中，DMA/EDMA 控制器自动将程序从位于 CE3 空间的 Flash 数据拷贝到地址 0(片内 RAM)开始的存储空间，然后跳转到地址 0 处执行，此为第一级的引导。系统 EMIF 接口自动将 8 位或者 16 位宽度的数据合并为 32 位数据。自动拷贝的程序根据处理器类型有所不同：C620x/C670x 为 64 K 字节数据，而

收稿日期：2014-02-28；**修回日期：**2014-03-28
作者简介：刘歆浏(1983—)，男，四川人，硕士，工程师，从事嵌入式软件及数字图像处理研究。

C621x/C671x/C64x 为 1 K 字节。当用户程序长度小于 64 K/1 K 时，无需二级 Bootloader；大于 64 K/1 K 的时候，二级引导程序就保存在 64 K/1 K 里。

2　C6455 自举设计与实现

2.1　硬件设计

为了满足将来用户程序的存储空间需要，本系统选用的 Flash 芯片是 AMD 公司的 AM29LV033C，32 Mbit 的总存储容量，采用 BYTE 模式[2]。C6455 扩展存储接口(EMIF)的 CE3 地址空间对应 Flash。

硬件电路设计如图 2 所示。

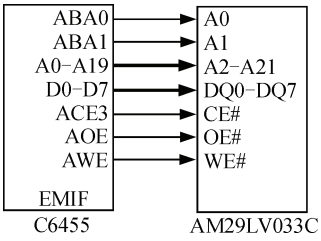

图 2　硬件电路设计

把 CE3 配置成 8 bit 异步接口，ABA[0:1]用作地址最低 2 位，共 22 位地址线，满足 AM29LV033C 的寻址空间。Flash 系统无需进行分页处理，极大地降低了 Flash 读写的软件和硬件开发难度。

2.2　二级下载实现

DSP 系统加电后，RESET 信号为低，芯片复位。在 RESET 信号上升沿处，锁存 BOOTMODE 信号，以决定芯片的存储器映射方式、地址 0 处的存储器类型以及复位后芯片的自举模式，复位结束后，芯片从存储器的 0 地址开始执行指令。BOOTMODE 信号设置为[0,1,0,0]，为 Flash boot 模式(通过对 DSP 芯片 4 个引脚的电平控制实现)。

二级 bootloader[3]的实现主要是对二级引导程序的开发，包括 2 个方面：1) 链接命令文件的编写；2) bootloader 代码的编写。

链接命令文件(*.cmd 文件)是链接器链接时的输入文件，是目标硬件系统存储空间分配及代码定位的描述性文件，决定各代码段和数据段在目标存储器中的绝对位置。

开发 Bootloader 代码主要包括 3 个步骤：1) 通过配置 EMIF 用来访问外部存储器；2) 将 ROM 中被初始化后的块移植到运行地址处，其中移植的地址需要和通过链接命令文件分配的地址一致；3) 调用_c_int00()完成启动。

2.3　程序固化

目标代码和 Bootloader 代码在完成编写后都需要烧写到 Flash 指定的空间里面。程序固化通用流程如图 3 所示。

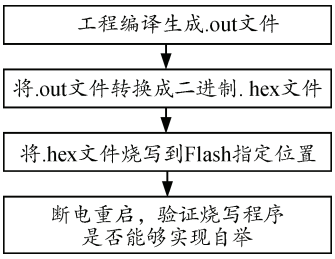

图 3　固化通用流程

用户程序经 CCS 编译连接生成目标文件*.out 文件为 COFF 格式，但 Flash 不支持这种文件格式，必须将该文件转换成 Flash 接受的 Hex 文件。COFF/Hex 的转化比较复杂，一般采用 TI 提供的工具实现。对于 C6x 系统，TI 提供了一个 hex6x 的应用软件，这样.out 文件就按照 COFF/Hex 转换协议转换到了 Hex 文件。

对于 Flash 烧写主要有 2 种方法：1) 自己编写 Flash 烧写代码；2) 采用外部工具进行烧写。最常用的方法是通过 Ti 公司提供的一套 FLASHBURN 的外部插件烧写。首先清洗掉 Flash 上的数据，然后将 BOOTLOADER 代码和目标代码分别烧写到 Flash 指定的地址上。

3　对现有方法的改进

从上节对现有方法的研究和分析可以发现，TI 公司针对 C6x 的解决方案虽然可靠，但是在实际的操作过程中却存在诸多不便。主要问题包括自举速度慢、Flash 烧写过程繁琐和不可控，以及引导程序编写复杂和不可控等问题。笔者针对这些问题，从 3 个方面进行了改进和提升。

3.1　分散加载

对于 C6x 来说，复杂系统的用户代码将会极大。现有二级引导都是采用了集中加载方式[4-5]，造成引导程序执行效率低、DSP 自举速度慢和片内资源紧张等问题。

笔者提出并采取了一种分散加载的新方法。如图 4 所示，主要是对二级引导程序的优化和修正，将目标代码的各种不同段定位到不同的物理地址上。进行二级引导的时候，分别进行代码的加载。主要的方法是将目标程序的执行代码定位到 DSP 的片内 IRAN 中，提高软件的执行效率；把变量段定位到执行速度稍慢的外部 RAM 中，减少片内 IRAM 的存储压力，以满足较大程序对内存的要求。

同时，对于目标预分配但无需初始化的数据段(本系统中，笔者定义为.my_data)，只定位地址，不进行段搬移，提高了 Flash 加载速度。

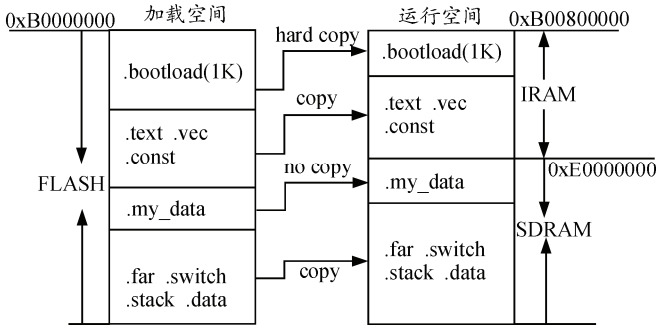

图 4 分散加载示意图

3.2 基于 C 语言的 bootloader 编写

Bootloader 的代码编写，也就是所谓的一级引导程序，系统自动加载的 1 K 代码。为了节省代码资源，现有系统大部分都采用了基于汇编的编写方式。但是汇编代码可改性、可读性不强，目标代码的一点变动都会造成引导代码的重复开发。并且由于采用了分散加载方式，一级引导程序的编写难度将更大。笔者通过研究，基于 C 语言完成了 bootloader 代码的编写。流程如下：

1) *.cmd 中给 bootloader 分配空间。

```
MEMORY{
    BOOT :  o= 00800000h l = 00000400h //分配空间
……  }
SECTIONS{
    .bootload  >  BOOT  //定义位段
……  }
```

2) 建立 bootloader 的源文件(boot.c)，并将其定位到分配的位段。

```
#progma CODE_SECTION(boot , ".bootload");
```

3) 建立 C 语言运行环境，并初始化 EMIF 接口。

```
extern far void c_init00(void);//C 语言入口地址
……
*(volatile init*)0x700000A0 = 0x00000380;
                                    //EMIFA_AWCC
*(volatile init*)0x70000084 = 0x00240120;
                                  //EMIF CE3 config
```

4) 根据分散加载的各个位段的地址，分别进行数据拷贝。

5) 完成拷贝后，调用 c_int00()，完成二级引导。

3.3 目标代码和程序固化的整合

针对 C6x 系列 DSP，由于 CCS 没有提供完整而且便捷的烧写方案，需要多个外部软件和插件配合，或者重新开发烧写程序并运行，才能完成烧写任务。程序的固化过程过于繁杂和不可控，在工程和研发项目中，给程序员带来了极大的不便。

笔者将目标代码和 Flash 烧写的整合，只需要在目标工程内通过一个全局变量完成状态切换，即可完成程序的固话。主要流程如下：

1) 建立烧写代码源文件和烧写主函数：void programer(void)；

2) 烧写代码分配空间分配到外部 SDRAM 的预留空间；

3) 建立状态标志全局变量 FLASH_WRITE，初始化默认状态为 0；

4) main()函数编写烧写入口程序：

```
If(FLASH_WRITE == 1)
{ rogramer();}
```

完成代码编写后，进行固化的步骤如图 5 所示。

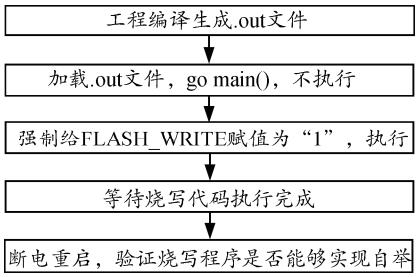

图 5 固化步骤

采用以上方式，程序的烧写和程序的调试只需通过一个全局变量就能完成自动切换。极大地提高了软件开发的效率，简化了 DSP 自举的复杂度。

4 结束语

笔者从多个方面对 DSP 自举进行了改进，为 C6x 系列自举所涉及的程序固化、导引代码的编写给出了一套完整的解决方案。实际使用结果证明：该方法极大地提高了 DSP 自举速度和程序运行的效率，降低了程序员开发 DSP 的难度。

参考文献：

[1] 张红. 基于 DM642 的二级 Bootloader 设计与实现[J]. 科技通报, 2012, 28(6): 197-201.

[2] 陈代媛. C6000 外部 FLASH 在线编程引导技术[J]. 电视技术, 2009, 49(5): 86-88.

[3] 韩艳芬, 吴援明, 王斯瑶, 等. 一种二次 Bootloader 升级和回退的设计与实现[J]. 计算机技术与发展, 2009, 19(10): 89-95.

[4] 王静, 陈伟, 刘志东, 等. DSP 系统中电磁兼容问题的技术研究[J]. 兵工自动化, 2013, 32(1): 36-37.

[5] 伍微, 王礼亮, 刘小汇. C6000 系列 DSP 带加解密的 bootloader 研究与应用[J]. 舰船电子工程, 2005, 25(2): 59-64.