多核 DSP 的 BootLoader 程序的实现

DSP 芯片的 BootLoader 程序用于实现用户程序上电自举，它有多种工作方式。上电自举就是将用户存放在片外的非易失性、慢速的存储器中的程序装载到片内易失的、高速的存储空间中，以保证用户程序在 DSP 核内的高速运行。

多核 DSP 是指由多个独立的 DSP 子核集成的 DSP 芯片，且所有 DSP 子核共享一套片外总线。由于每个 DSP 子核内部都有其自身独立的掩模 BootLoader 程序，当 DSP 芯片上电或复位时　所有 DSP 子核都将自行启动自身独立的 BootLoader 程序，实现用户程序的上电自举。所以，多核 DSP 的 BootLoader 程序的实现方法与单核 DSP 的 BootLoader 程序的实现方法有较大的差异。为此，本文立足于实践，以双核 DSP-TMS320VC5421 的 1 6 位并行 E P R O M的 BootLoader 程序的工作方式为例，详细阐述了多核 DSP 的 BootLoader 程序的实现方法。

BootLoader 程序简介

BootLoader 程序的四种工作方式

一般的 DSP 都采用常见的 BootLoader 程序工作方式来实现用户程序的上电自举：

1. 处理器通信口(主端口)HPI 方式--通过 DSP 芯片与 P C 机或 DSP 芯片与其它 DSP 芯片之间的主机通信端口实现上电自举；
2. 8 位或 1 6 位并行 EPROM 方式--通过 DSP 内核的 D M A 通道实现上电自举；

3. 8 位或 1 6 位并行 I/O 方式--通过 DSP 芯片的片外并行 I/O 接口实现上电自举；
4. 8 位或 1 6 位串行口方式--通过 DSP 芯片的串行端口实现上电自举。

在以上四种工作方式中，最常用的是 1 6 位并行 EPROM 方式。即在 DSP 芯片上电或复位时，通过 D M A 通道将存储在核外 EPROM 中的程序以 1 6 位形式存储到核内的程序空间中。

16 位并行 EPROM 方式的 Boot 表

各种方式的 BootLoader 程序都有其固定格式的 Boot 表，用来实现用户程序的上电自举。 1 6 位并行 EPROM 方式的 Boot 表如表 1 所示：

表项 1：存放 BootLoader 程序工作方式控制字，用于 DSP 芯片上电或复位时确认该 Boot 表是否为 1 6 位并行 E P R O M工作方式的 Boot 表。该表项内容为 10AAH，表示 DSP 内核认为该 Boot 表是 1 6 位并行 E P R O M工作方式的 BootLoader 程序的 Boot 表；否则 DSP 内核认为该 Boot 表不是 1 6 位并行 E P R O M的方式的 Boot 表；

表项 2：存放 DSP 特殊寄存器 SWWSR 在上电或复位时被赋予的初始化数值；

表项 3:存放 DSP 特殊寄存器 B S C R在上电或复位时被赋予的初始化数值；

表项 4：存放用户程序将要被存放在 DSP 核内程序空间的页地址；

表项 5：存放用户程序将要被存放到 DSP 核内程序空间的页内偏移地址；

表项 6：开始依次存放用户程序第m段代码的长度N。用户程序第m段代码将要被存放到 DSP 核内程序空间的页地址，用户程序第m段代码将要被存放到 DSP 核内程序空间的页内偏移地址，用户程序第m段代码的第 1 个字，第 2 个字，……，第N个字；Boot 表的最后表项存放 Boot 表结束字 0 0 0 0 H，表示 Boot 表到此结束。因此 DSP 内核要实现 BootLoader 程序，在上电复位后首先要申请到片外数据、地址总线的控制权，然后再根据 Boot 表完成用户程序上电自举过程。

表 1 16 位并行 EPROM 方式的 Boot 表

10AAH
初始化 SWWSR
初始化 BSCR
XPC 的入口地址
入口地址
第一段大小
第一段目的地址 XPC
第一段目的地址
代码字 1
·
·
代码字 *N*
第 *m* 段大小
第 *m* 段目的地址 XPC
第 *m* 段目的地址
代码字 1
·
代码字 *N*
0000H

16 位并行 EPROM 方式 Boot 表的生成

所有 BootLoader 程序所需的 Boot 表的数据结构都是通过执行包含－v 5 4 8 参数的链接命令和Hex 5 0 0 转换命令的程序形成的。在链接过程中确定用户程序和数据的存放地址，在Hex 5 0 0 转换过程中定义BootLoader 程序的工作方式和用户程序执行的入口地址等。

为了生成 1 6 位并行ＥＰＲＯＭ方式的 Boot 表　首先，在链接程序时必须设置－v 5 4 8 选项；然后使用ＴＩ公司 DSP 开发工具自带的ＨＥＸ５００．ＥＸＥ文件，根据用户的ＣＯＦＦ格式的代码生成 Boot 表中的相应内容。

HEX500.EXE 可执行文件一般使用以下几种参数

（1）*.out ：用户的 COFF 格式的程序；

（2）-e ：确定用户程序的入口点；

（3）-a：以 ASCII 形式，根据用户的.ｏｕｔ文件输出对应的ＨＥＸ文件；

（4）－Boot 实现用户程序的装载；

（5）－bootorg ：确定生成哪种形式的 Boot 表；

（6）－ｍｅｍｗｉｄｔｈ:确定引导方式的位数；

（7）－Ｏ ＊．ｈｅｘ:输出的ＨＥＸ文件的名称。

例如：

```
hex500 ti.out /  根据 ti.out 文件生成 Boot 表  /

-e 0x4000 /  用户程序的入口点为 0x4000  /
-a /  以 ASCII 形式输出ＨＥＸ文件  /
－Boot / 装载用户的程序ｔｉ．ｏｕｔ  /

－Bootｏｒｇ ＰＡＲＡＬＬＥＬ

/  生成并行ＥＰＲＯＭ方式的 Boot 表  /
－ｍｅｍｗｉｄｔｈ １６ /  生成１６位的 Boot 表  /

－ｏ ｔｉ．ｈｅｘ /  生成的ＨＥＸ文件名为ｔｉ．ｈｅｘ  /
```

执行完该 HEX500.EXE 命令后，系统会创建一个文件名为ｔｉ．ｈｅｘ的ＡＳＣＩＩ文件，然后用户根据ｔｉ．ｈｅｘ文件内容对ＥＰＲＯＭ进行编程就能产生上述的１６位并行ＥＰＲＯＭ工作方式的 Boot 表。

多核 DSP 的 BootLoader 程序的实现

目前ＴＩ公司已经不再局限于生产单核 DSP。为了提高用户程序运行的效率，ＴＩ公司又推出了 2 核、4 核等多核 DSP。在实现多核 DSP 上电自举时，每一个子核都需要申请片外总线的控制权。对于单核 DSP 而言，只有一个 DSP 内核，对应一个 BootLoader 程序，DSP 核可以永远拥有片外总线的控制权。但对于多核 DSP 而言，由于只有一套片外总线，所以片外总线的控制权不允许也不可能永远被其中的某一个 DSP 子核所拥有。因此，多核 DSP 需要片外总线仲裁机制，以避免片外总线冲突。

下面以双核 DSP-ＴＭＳ３２０ＶＣ５４２１的１６位并行ＥＰＲＯＭ方式的 BootLoader 程序实现过程为例，详细阐述多核 DSP 的 BootLoader 程序的实现。

２．１ TMS320VC5421 结构简介

TMS320VC5421１６位定点双核 DSP，它集中了早期ＴＭＳ３２０Ｃ５４Ｘ系列 DSP 的优点，并提供了许多新的功能。其内部结构与ＴＭＳ３２０Ｃ５４Ｘ系列的其它款式 DSP 有很大的不同，其简单结构框图如图 1 所示。

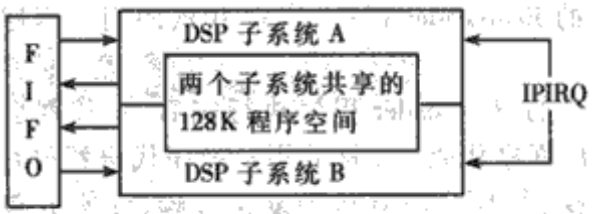

图 1 TMS320VC5421 的系统结构简图

由于每个 DSP 子核的工作频率是１００ＭＨｚ，所以它的工作速率可达到２００ＭＩＰＳ，且它的每一个 DSP 子核都具备单核 DSP(如ＴＭＳ３２０ＶＣ５４０２)的所有特性。

２． 2 TMS320VC5421 的１６位并行ＥＰＲＯＭ工作方式的 BootLoader 程序的选择

TMS320VC5421 的两个 DSP 子核在 DSP 芯片上电或复位时，能否启动各自的 BootLoader 程序以完成上电自举功能，是由每个子核自身的 XIO 和ＧＰＩＯ０／ＲＯＭＥＮ两个管脚决定的。在 DSP 芯片上电或复位时，每个 DSP 子核自动检测自身的ＸＩＯ和ＧＰＩＯ０／ＲＯＭＥＮ两个管脚，如果对应的ＸＩＯ和ＧＰＩＯ０／ＲＯＭＥＮ两个管脚都为高电平　则启动自身的 BootLoader 程序完成用户程序的上电自举。

每个 DSP 子核启动 BootLoader 程序后，采用哪一种 BootLoader 程序的工作方式是由各自的ＧＰＩＯ１管脚的状态和各自以ＤＭＡ方式从核外数据空间０００００Ｈ地址单元读入的数据决定的：检测ＧＰＩＯ１管脚，如果ＧＰＩＯ１管脚为高电平，则采用串行口ＥＥＰＲＯＭ的 BootLoader 工作方式，否则采用并行ＥＰＲＯＭ的 BootLoader 工作方式。若 DSP 子核的ＤＭＡ通道读入核外数据空间００００Ｈ单元中的数据为１０ＡＡＨ，则采用１６位并行ＥＰＲＯＭ的 BootLoader 工作方式；若读入的数据为ｘｘ０８Ｈ或ｘｘＡＡＨ　则采用 8 位并行ＥＰＲＯＭ的 BootLoader 工作方式。否则将重新判断ＧＰＩＯ１管脚的电平，进入死循环。

２． 3　TMS320VC5421 的 BootLoader 程序片外总线冲突的解决

DSP 核的 BootLoader 程序总是在 DSP 核上电或复位时启动，且一启动 BootLoader 程序，对应的 DSP 核就要申请核外的总线控制权。因此为了避免多核 DSP 的各个 DSP 子核启动 BootLoader 程序时引起的片外总线冲突，可通过控制每个 DSP 子核的复位过程，使每个 DSP 子核在不同的时间内启动自身的 BootLoader 程序来解决片外总线冲突的问题。

为了实现两个 DSP 子核复位过程的分离，应采用如图 2 所示的 DSP 子核复位过程控制方法。

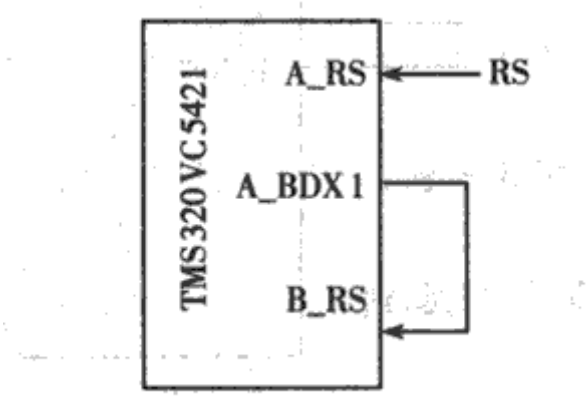

图 2 DSP 子核复位过程控制图

由于TMS320VC5421中A核拥有倍频的锁相环电路，所以首先复位A核，启动A核的BootLoader程序，实现A核的用户程序上电自举。然后再由A核的用户程序控制B核的复位过程，启动B核的BootLoader程序，实现B核的用户程序上电自举。

在A核的BootLoader程序执行完后，A核就会执行自身的用户程序代码。A核的用户程序代码释放片外总线的控制权，并且控制B核的复位管脚，促使B核启动自身的BootLoader程序。如果此时A核中的用户代码又申请片外总线控制权或正在使用片外总线，就会造成片外总线冲突。解决此冲突的办法有如下两个：

粗略估计B核的BootLoader程序执行时间，在A核的有效程序代码前加一个延迟程序。

在A核的有效程序代码前加入一个死循环程序，当B核BootLoader程序执行完后，B核通知A核，A核就跳出这个死循环程序，开始执行自己的有效代码。

2．4 TMS320VC5421的16位并行EPROM工作方式的BootLoader程序的编程实现

首先设计一个简单的电路图，如图3所示。在DSP的A__XF和B__XF两个管脚分别连接一个发光二极管，A核以2Hz的频率点亮发光二极管，B核以10Hz的频率点亮发光二极管。将128K的FLASH SST39VF400A 分成两页，每页为64K。FLASH的页的选择由TMS320VC5421的A__BDXO管脚控制。当A__BDX0为低电平，即FLASH的A16地址线为低电平时，选中FLASH的第一页，由FLASH的A0～A15地址线选择页内地址，用于存放A核的16位并行EPROM工作方式的Boot表。当A__BDX0为高电平，即FLASH的A16地址线为高电平时，选中FLASH的第二页，由FLASH的A0～A15地址线选择页内地址，用于存放B核的16位并行EPROM工作方式的Boot表。

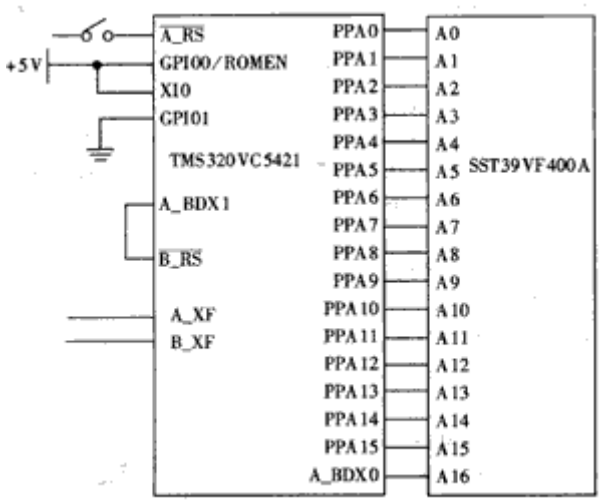

图 3 16 位 EPROM 并行引导装载实现电路简图

CPU_A 和 CPU_B 的程序流程图分别如图 4 和图 5 所示。

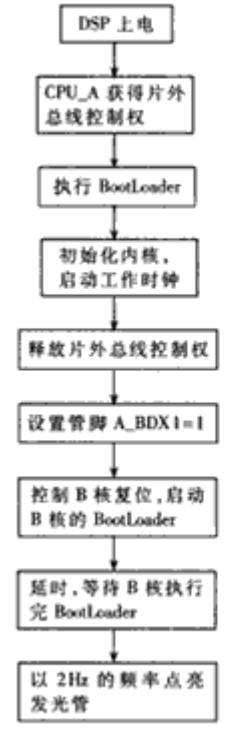

图 4 CPU_A 的程序流程图

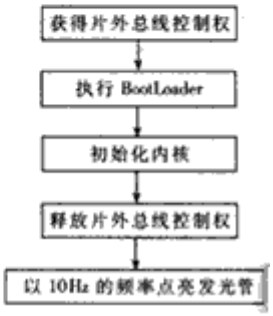

图 5 CPU_B 的程序流程图

1 片外总线冲突的解决

估算 B 核执行 BootLoader 程序所需的时间后，在 A 核的用户有效程序之前，加一段延迟程序。

延迟的时间计算如下：

TMS320VC5421DSP 的 D M A 通道从片外数据空间读取一个字到片内数据空间，需要 7 个指令周期时间。

统计用户程序大小　将对应 Boot 表中的所有段的大小相加　Ｎ１＋Ｎ２＋．．．＝Ｎ。

延迟的时间为Ｎ×７＝７Ｎ个指令周期。

由上面所述的方法可知，只需在开始执行Ａ核的有效程序之前加一段延迟７Ｎ个指令周期的代码即可。

（２）生成 Boot 表

对 CPU_A 来说，以Ａ核程序流程图建立一个项目Ａｔｉ．ｍｓｋ。产生Ａｔｉ．ｏｕｔ文件后，进入该目录的ＤＯＳ环境，键入：

hex500 Ati. out－ａ－ｅ 0x4000h－Boot－Bootｏｒｇ PARALLEL－ｍｅｍｗｉｄｔｈ１６－ｒｏｍｗｉｄｔｈ １６－ｏ Ａｔｉ．ｈｅｘ

生成Ａ核的１６位并行ＥＰＲＯＭ工作方式的 Boot 表。

对ＣＰＵ__Ｂ来说，同样以Ｂ核程序流程图建立一个项目Ｂｔｉ．ｍｓｋ。产生Ｂｔｉ．ｏｕｔ文件后，进入该目录的ＤＯＳ环境，键入：

hex500 Bti.out-a -e0x4000h－Boot－Bootorg PARALLEL－ｍｅｍｗｉｄｔｈ１６－romwidth １６－ｏ Bti.hex

生成Ｂ核的１６位并行ＥＰＲＯＭ工作方式的 Boot 表。

在实现双核 DSP 的上电自举后，Ａ核和Ｂ核的用户程序将会被存放在核内程序空间的不同页面上。如从ＤＭＡ的角度观看：Ａ核的用户程序将被存放在Ａ核的程序空间的第 0 页上；Ｂ核的用户程序将被存放在Ｂ核的程序空间的第 2 页上。因此Ａ核的 Boot 表不需要修改，而Ｂ核的 Boot 表中的所有存放页地址的表项中的内容要更改为 2 。

（３）FLASH 编程实现

根据 FLASH 芯片的控制时序，编写一个简单的 DSP 程序，用于将Ａ核的 Boot 表写入 FLASH 的低６４Ｋ，将Ｂ核的 Boot 表写入 FLASH 的高６４Ｋ。

上电试验结果

将电路上电后，Ａ核控制的发光二极管开始闪烁，Ｂ核控制的发光二极管也开始闪烁，且Ａ核发光二极管闪烁频率要低于Ｂ核发光二极管闪烁频率。由此现象可得出　Ａ核与Ｂ核的 BootLoader 实现成功，未产生片外总线冲突；Ａ核以２Ｈｚ的频率点亮发光二极管，Ｂ核以１０Ｈｚ的频率点亮发光二极管。

附件：DSP Boot Loader

初学 DSP 时最头疼的事就是 DSP 的 bootload 问题，以前学 51 时只要把程序写好编译通过后就可以用烧写器直接将*.hex 文件烧进单片机运行。但 DSP 内部不带 FLASH RAM，它必须在复位期间将外部的程序加载到内部 RAM 之后才能运行。这有点像 PC 的体系结构，PC 机中的引导加载程序由 BIOS(其本质就是一段固件程序)和位于硬盘 MBR 中的 OS Boot Loader（比如，LILO 和 GRUB 等）一起组成。BIOS 在完成硬件检测和资源分配后，将硬盘 MBR 中的 Boot Loader 读到系统的 RAM 中，然后将控制权交给 OS Boot Loader。Boot Loader 的主要运行任务就是将内核映象从硬盘上读到 RAM 中，然后跳转到内核的入口点去运行，也即开始启动操作系统。我刚开始时就被这 bootload 搞的心焦如焚，仿真好的程序却不能脱机运行，最好还是老老实实看 TI 的 DSP datasheet，经过一番周折总算搞定！为此我想把我的调试心得写出来与大家共享，一方面是避免初学者走弯路，另一方面是借此抛砖引玉望方家多斧正^_^。

简单地说，bootloader 就是在用户程序运行之前的一段小程序，通过这段小程序初始化硬件设备、建立内存空间的映射图，最终调用用户程序。这段小程序其实已固化在 DSP 芯片 TMS320VC33 内部 ROM 中,在 TMS320C3X datasheet 中有源码及功能流程图，我们只需知道它的功能流程即可。从流程图我们可以知道在 DSP 上电复位时只要 DSP 芯片引脚'MCBL/MP'为高电平，TMS320VC33 就开始自动执行固化在内部 ROM 中的 bootloader 程序，然后根据外部中断引脚'INT3～INT0'判断用户程序的加载起始地址。

管脚	电平	存储空间	
INT0	0	0x001000	
INT1	0	0x400000	外部 FLASH
INT2	0	0xfff000	
INT3	0	串口	

复位对应的管脚即会从对应空间读取用户程序执行

在我的系统应用中，我将程序存储在外部 FLASH 芯片 39VF040 中，它在系统中的起始地址为：0X400000,所以只要 DSP 复位时外部中断引脚'INT1'为低电平，bootloader 程序就开始读取外部 FLASH 芯片 39VF040 中的用户程序并加载到 DSP 内部 RAM 中，加载完毕后就自动跳到用户程序的入口地址开始运行用户程序。从 bootloader 的流程图还可以知道，bootloadr 加载用户程序时是有一定格式要求的，即存储在用户外部 FLASH 程序空间的数据结构的格式要求如下：

WORD 0 ：用户外部 FLASH 芯片数据宽度，如 8，16，32 位等
WORD 1 ：控制字，用来写入 TMS320VC33 的总线控制寄存器
WORD 2 ：数据块大小
WORD 3 ：当前上载数据块将要装载到 DSP 内部 RAM 中的目标地址
WORD 4～N ：用户程序内容

这个排布编译器会实现。用户只要设置而已。

用户的程序分为多个数据块（因为 DSP 开发软件生成的目标文件是 COFF 格式），每块

数据块起始都包含一个程序头，每个又包含两个内容：1、当前数据块大小，即 32 位格式的数据量。2、当前数据块在 DSP 内部 RAM 存储的起始地址。程序头之后就是用户的程序内容。

讲到这里问题的关键就出来了：怎样产生这样的程序块呢？程序内容应该为哪种格式*.hex、*.bin、*.out？这也是我当初最头疼的问题。

用 TI 公司的 DSP 开发软件 Code Composer 建立一个项目文件后，要做的第一件事就是编写*.cmd 命令文件，命令文件有两个：一个是链接命令文件，另一个是 boot 引导表格式生成命令文件。

链接命令文件作用是分配各个程序段在 DSP 内部 RAM 中的存储位置，链接命令文件必须和项目文件名相同。例如项目 online.mak 的链接命令文件 online.cmd 如下所示：

```
-c //ROM 初始化
-o online.out //产生 online.out 可执行文件
-m online.map //产生 online.map 映象文件
online.obj //链接的目标文件
-l rts30.lib //链入 TMS320C3X 运行支持库

MEMORY
{
 VECS: org=0x809fc1 len=0x3f //定义矢量的起始地址及空间的长度
 RAM0: org=0x809800 len=0x7c1 //定义堆栈起始地址及空间的长度
 RAM2: org=0x800000 len=0x8000 //定义用户程序数据空间
}
SECTIONS
{
 "vectors": load=VECS //将中断向量块安排在 VECS 空间
 .text:   load=RAM2 //将程序代码、常量、变量等数据块安排在 RAM2 空间
 .cinit:   load=RAM2
 .const:   load=RAM2
 .bss:   load=RAM2
 .stack:   load=RAM0 //将堆栈块安排在 RAM0 空间
}
```

建立链接命令文件后，开发软件在对用户程序汇编链接生成目标文件的过程中，就会按照链接命令文件对输出的 COFF 格式的数据块自动选择存储器地址。

boot 引导表格式生成命令文件，这个文件名可以随意取，例如可取名为 hex.cmd。先头讲到 TMS320VC33 的 bootloader 程序加载用户程序是有一定格式，boot 引导表格式生成命令文件就是将用户的目标文件转换成符合要求的格式，举例如下：hex.cmd

```
online.out //要进行转换的目标文件
-map hex.map //生成映象文件
-boot //产生装载程序
-image //输出文件去掉地址映象
-i //建立 INTEL 十六进制文件的输出
-memwidth 8 //用户程序存储器的数据宽度
```

```
-cg 10e8h //总线控制字
-e 00803f86h    //程序装载完成后的执行地址即为_c_int00 的地址，可查看 online.map 文件*/

ROMS //ROM 的映射范围，及用户程序在外部 FLASH 中的地址空间
{
      FLASH1: org=0,len=10000h,romwidth=8,files={online1.hex}
      FLASH2: org=10000h,len=10000h,romwidth=8,files={online2.hex}
}
```

写好格式转换命令文件后，在 WINDOWS 的命令提示符工具下用'cd'命令进入用户程序目标文件所在目录，然后运行 hex30 hex.cmd 即可根据格式命令文件 hex.cmd 的要求产生两个 hex 格式的文件 online1.hex 和 online2.hex。其中 hex30 是 DSP 开发软件自带的程序。因为 Intel 格式中除了数据之外还有起始符、字节个数、起始地址、类型以及校验位等各种信息，并非纯粹数据的 HEX 格式表示。所以还不能直接烧进 FLASH 芯片，同样在命令提示符下运行 hexbin 程序，将 online1.hex 和 online2.hex 分别转换成 online1.bin 和 online2.bin，就可得到纯粹数据格式的烧写文件，好了，只要把刚才两个文件烧进 FLASH 芯片，重启系统，OK!终于可以脱机运行程序了.