

# HHBF533/HHBF561 U-Boot & $\mu$ Clinux 用户手册\*

Version 0.3.0

Meihui Fan

范美辉

<[hhbf-support@hhcn.com](mailto:hhbf-support@hhcn.com)>

Copyright © 2005 HHTech. Co., Ltd. †  
All rights reserved.

华恒科技 版权所有

2005年6月7日

---

\*如无特别说明，本文档为HHBF533/HHBF561 通用。

†<http://www.hhcn.com>

## 目录

<b>1</b>	<b>U-Boot 简介</b>	<b>4</b>
<b>2</b>	<b>映象安装指示</b>	<b>4</b>
2.1	第一次的映象烧写	5
2.2	准备映象文件	9
2.3	通过串口烧写映象	10
2.4	通过以太网口烧写映象	11
2.5	典型的存储空间映射	13
<b>3</b>	<b>U-Boot 的使用</b>	<b>13</b>
3.1	U-Boot 引导Linux 映象	13
3.2	监听器命令	13
3.3	U-Boot 的配置	16
<b>4</b>	<b>为HHBF533/HHBF561 创建U-Boot 映象</b>	<b>18</b>
4.1	在SDRAM 中调试Linux	18
<b>5</b>	<b>为HHBF533/HHBF561 创建<math>\mu</math>Clinux 内核映象</b>	<b>19</b>
<b>6</b>	<b>关于<math>\mu</math>Clinux 中一些驱动程序的使用说明</b>	<b>19</b>
6.1	BF561 CORE B 的二级引导驱动	19
6.2	以太网驱动和网络应用服务	23
6.3	音频驱动	23
6.3.1	简单的音频输入输出演示	24
6.3.2	音频录 / 放演示	24
6.4	视频驱动	24
6.4.1	简单的视频输入输出演示	24
6.4.2	用于视频编解码演示的应用程序	25
6.4.3	Frame Buffer 驱动	25
6.4.4	V4L2 驱动	26
6.5	片内SRAM 的测试程序	26
6.6	IPC 测试程序	26
<b>7</b>	<b>内核和应用程序调试</b>	<b>27</b>
<b>8</b>	<b>常见问题与解答</b>	<b>27</b>

## List of Figures

1	VDSP 中典型的寄存器初始化设置(HHBF561)	6
2	PC 主机上串口终端的设置	7
3	用Kermit 协议串口发送文件	8

4	把映像烧写到FLASH 中 . . . . .	10
5	U-Boot 引导Linux 映像之一 . . . . .	14
6	U-Boot 引导Linux 映像之二 . . . . .	15
7	μClinux Menuconfig . . . . .	20
8	μClinux Vendor & Linux Kernel Menuconfig . . . . .	21
9	μClibc Menuconfig . . . . .	22

## List of Tables

1	U-Boot 期典型的寄存器初始化值 . . . . .	5
2	HHBF533 的U-Boot 期存储映射 . . . . .	12
3	HHBF561 的U-Boot 期存储映射 . . . . .	12

## 1 U-Boot 简介

“U-Boot”是一个通用启动加载器固件的项目，项目的所有源代码都遵循公共通用许可证(GPL)协议。

U-Boot 是给嵌入式板片使用的启动加载器，最初是基于PowerPC 和ARM 系列处理器的。我们把它移植到HHBF533/HHBF561 开发板上。它可以被烧入到启动ROM 中，并用于检测硬件或者下载和运行应用程序。

U-Boot 的开发与Linux 紧密相关，事实上，U-Boot 借用了Linux 源码树中的部分代码。在U-Boot 中有一些公共的头文件，当然也提供了一些专门的代码以支持Linux 系统映象的引导。

为了更好和更容易地配置、使用和扩展U-Boot 的功能，你需要阅读并注意本文提到的一些细节。比如：所有的监听命令都使用了同样的调用接口，因而你可以非常方便地加上新的命令。此外，对于一些很少用到的代码（如硬件测试实用工具），你也许不需要也不喜欢把它们加到监听器中，于是你可以动态地加载和运行它们。

以下是U-Boot 的一些特性：

- ✿ 通过RS-232 串行接口或者以太网口与主机连接通信。
- ✿ 提供了基于RS-232 的命令行接口。
- ✿ 可以使用TFTP 协议或者Kermit 协议来下载映象。
- ✿ 支持压缩的映象文件格式的下载和FLASH 加载。
- ✿ FLASH 映象管理系统支持多映象格式。
- ✿ 可以把U-Boot 期的习惯配置存储于FLASH 中。
- ✿ 支持执行引导期脚本命令。

更多关于U-Boot 的信息请参考：<http://u-boot.sourceforge.net> 和U-Boot 源码树中的‘README’ & ‘ReleaseNotes’ 文件。

## 2 映象安装指示

可以有两种方法把一个准备好的U-Boot 或 $\mu$ Clinux 的ROM 映象下载到板载内存SDRAM 中。第一种是利用Kermit 协议通过UART1 串口下载，另外一种是利用TFTP 协议通过以太网接口下载。其中串口下载比较慢，但是更为健壮。

当你一开始得到/购到一块裸的开发板时，你需要学习使用ADI 的Visual DSP++（简记为‘VDSP’）仿真软件来通过JTAG 接口下载RAM 版的U-Boot 映象，然后再利用运行的这一RAM 版U-Boot 下载和烧写ROM 版U-Boot 映象到板载FLASH 闪存中。

## 2.1 第一次的映象烧写

1. 从我们的分发光盘中得到<sup>①</sup>或者自己构建（请参考第4节）RAM版和ROM版U-Boot映象，并且确定RAM版是ELF格式而ROM版是裸二进制格式。
2. 把开发板和ADI仿真器用JTAG接口连接，并且与主机用RS-232串口连接。
3. 运行ICE Test实用工具，并运行VDSP软件检测BF533/BF561开发板，如图1。
4. 在VDSP中下载ELF格式RAM版U-Boot映象。
5. 手工给一些寄存器初始化，比如：‘PLL\_CTL’，‘UART\_DLL’，‘EBIU\_SDGCTL’，‘EBIU\_SDBCTL’，‘EBIU\_SDRRC’等等。表1和图1是典型的寄存器设置。

第1表: U-Boot期典型的寄存器初始化值

寄存器名	系统复位值	HHBF533 的设置	HHBF561(CORE A) 的设置
EBIU_AMBCTL0	0xffc2ffc2	0x7bb07bb0	0x7bb07bb0
EBIU_AMBCTL1	0xffc2ffc2	0x99b37bb0	0x99b37bb0
EBIU_AMGCTL	0x00f2	0x00f8	0x00f8
EBIU_SDGCTL	0xe0088849	0x0091998d	0x0091998d
EBIU_SDBCTL	0x00000000	0x00000013	0x00000015
EBIU_SDRRC	0x081a	0x074a	0x074a
UART_DLL	0x0000	0x0012	0x0012 <sup>②</sup>
PLL_CTL	0x1400	0x1400	0x2800
PC	0xffa00000	0xffa00000	0xffa00000

**注意：**EBIU系列寄存器设置之后读出来的值可能跟设置值不太一致。

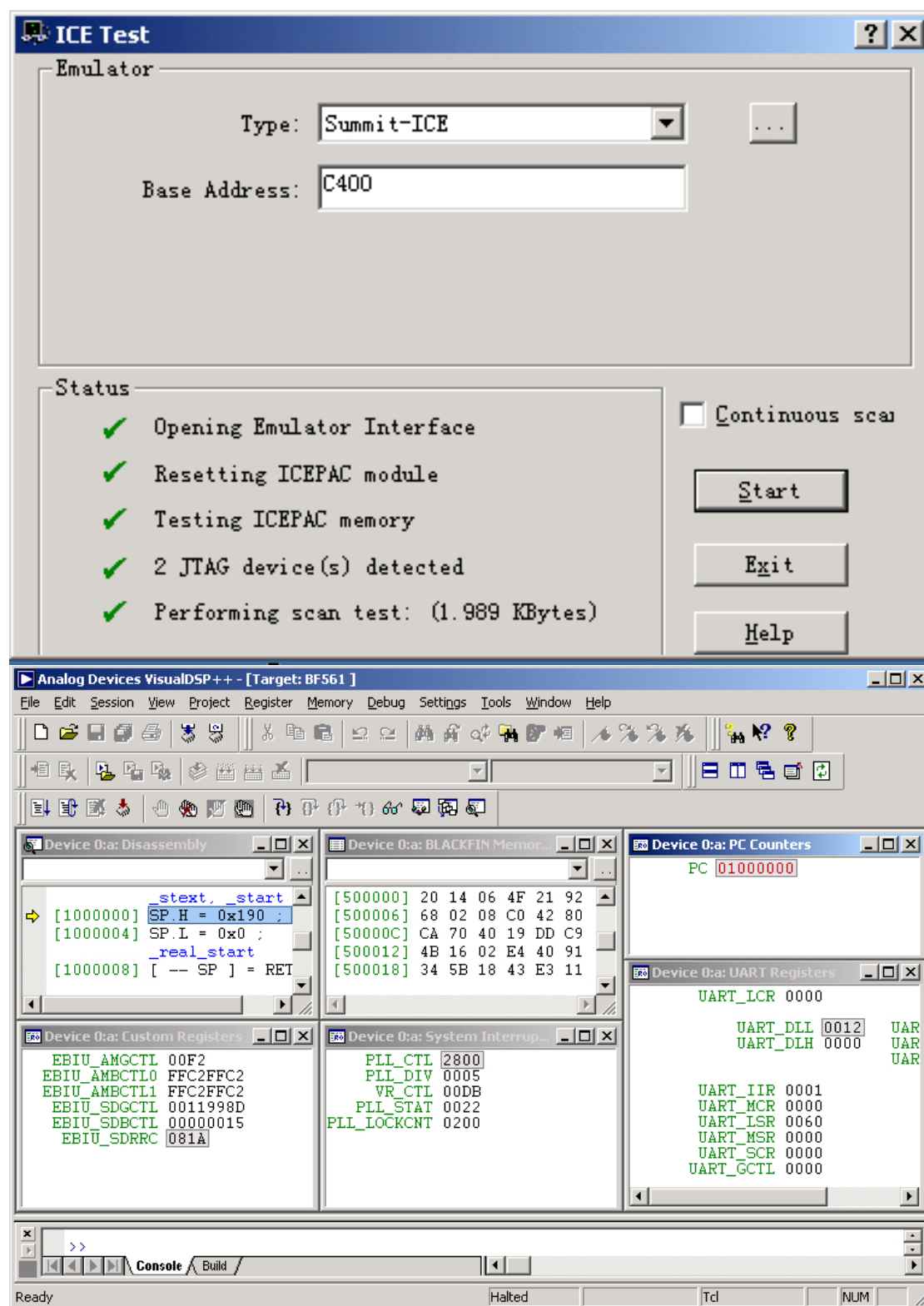
6. 修改PC计数器，使它指向RAM版U-Boot映象的入口点（典型地是0x10000000）。
7. 运行一个串口终端（典型地在MS Windows中是Hyper Terminal，在GNU/Linux中是minicom），连接到相应串口并设置波特率、校验位和检测位（典型地是：‘115200 8N1’，如图2和图2）。
8. 在VDSP中按下<F5>键使BF533/BF561开始运行U-Boot映象。<sup>③</sup>
9. 在运行的U-Boot中通过串口（或网口）下载ROM版U-Boot映象，并烧写到板载FLASH闪存中：典型地，首先在U-Boot提示符后启动串口下载程序‘loadb’，后面用一个参数指定下载的内存地址：

```
BOOT> loadb 10000000
```

之后在主机的串口终端中通过Kermit协议发送ROM版U-Boot映象，如图3和图3。

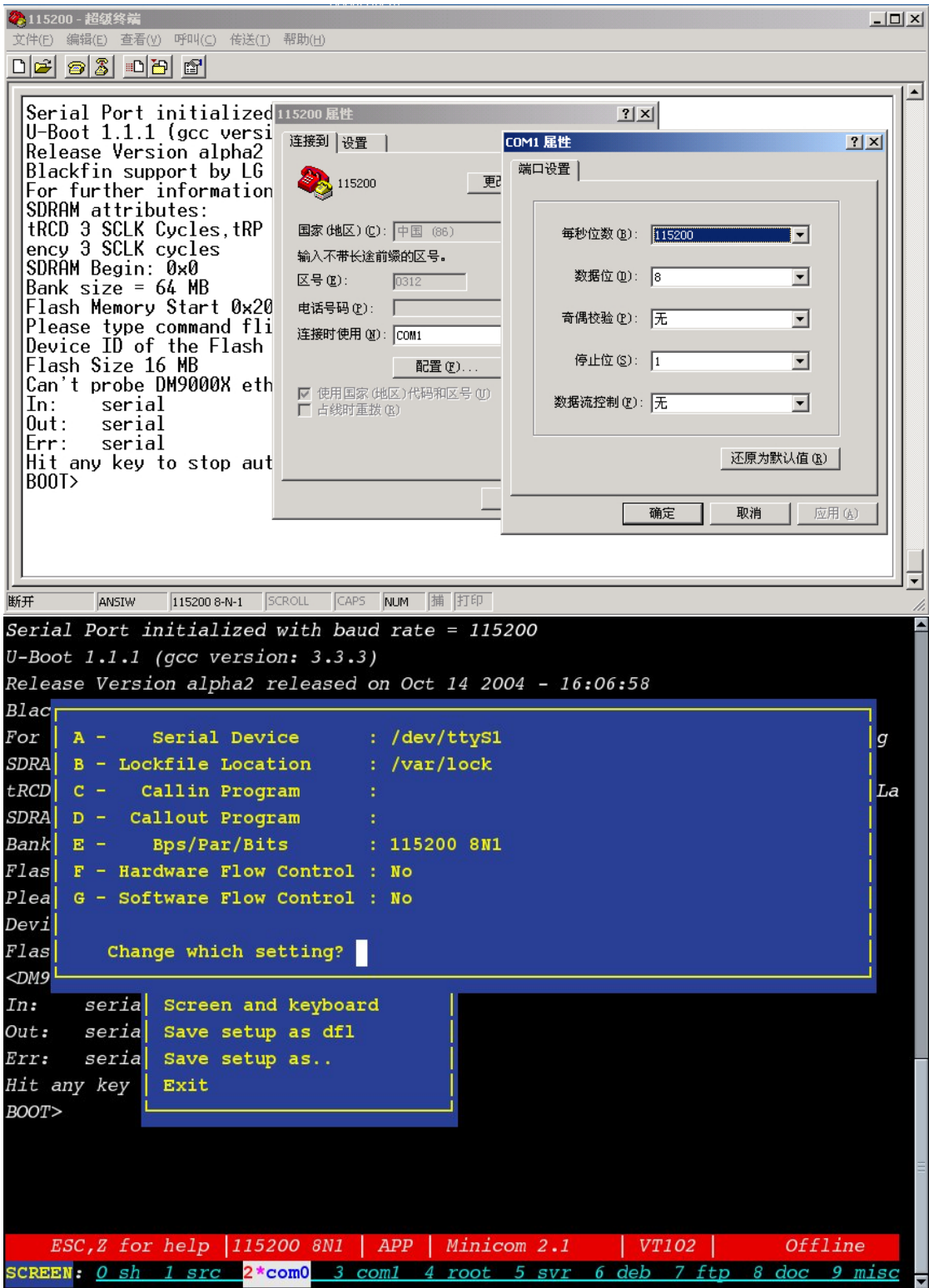
<sup>①</sup>在分发包解开的目录‘images’目录下有我们预先编译制作好的各种映象文件。

<sup>②</sup>在BF561中，如果运行后串口终端没有反应，则在VDSP中按下Shift + <F5>挂起CPU，若PC Counter指向指令：STI R2；则再按下<F5>继续让CPU运转，这下应该好了！这也是BF561在仿真器下的“不良反应”。

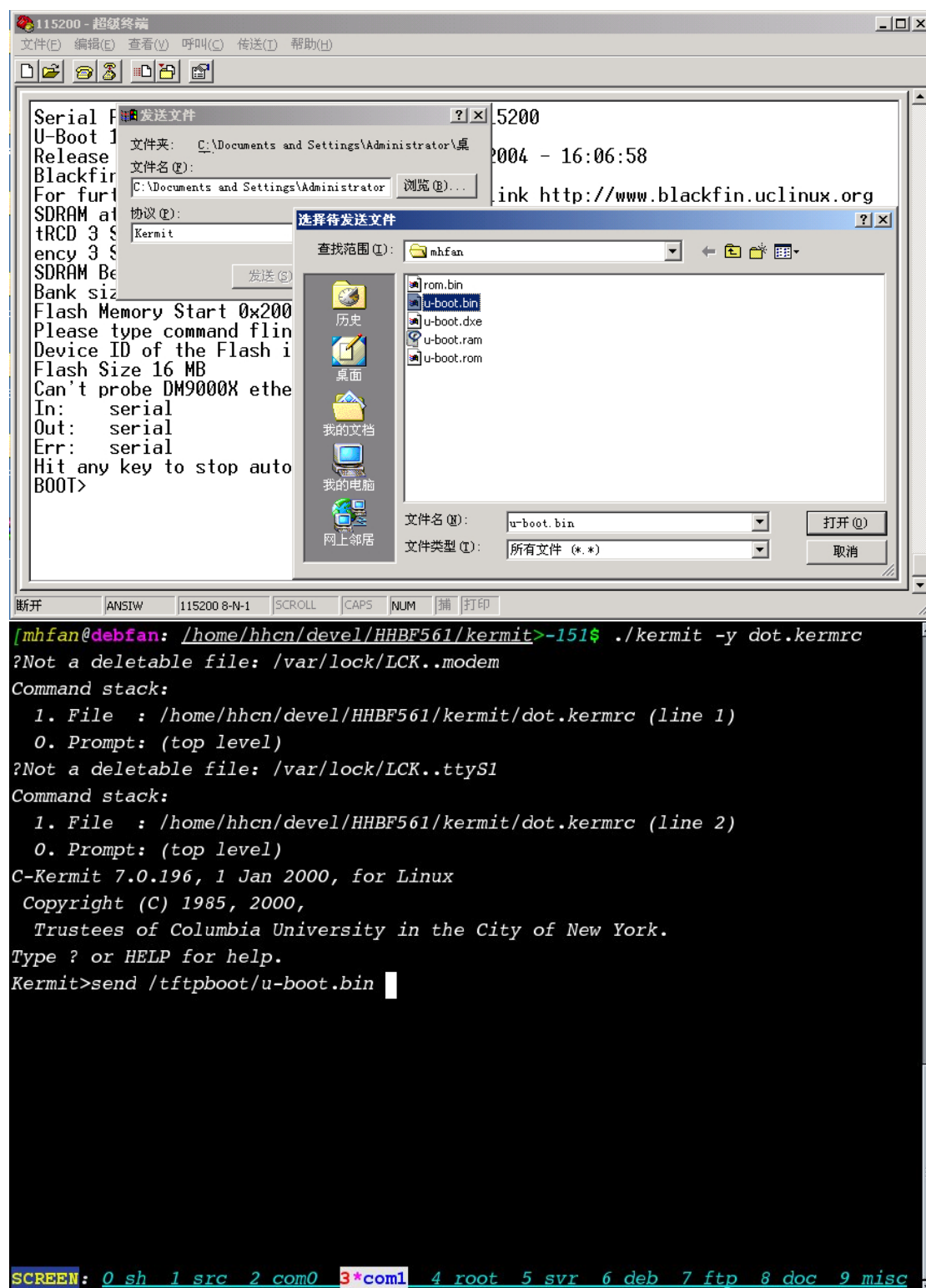


第 1 图: VDSP 中典型的寄存器初始化设置(HHBF561)

注意: 对于HHBF533 需要做相应的修改, 详见表1 HHBF533 一列。



第 2 图: PC 主机上串口终端的设置



第 3 图: 用Kermit 协议串口发送文件



传输完成后，把映象烧写到FLASH中，典型地：

```
BOOT> fl 200000000 1000000 20000
```

关于下载和烧写映象的更详细的信息，请参考本节的最后两个小节，2.3 和2.4 以及第3 节。

## 2.2 准备映象文件

如果你的闪存足够大，不需要压缩映象或者组合U-Boot 和 $\mu$ Clinux 映象，那么你可以跳过这一节；否则你必须学习使用‘mkimage’ 实用工具来从原始映象生成压缩映象或多映象组合。

‘mkimage’ 工具是随着U-Boot 一起分发的，你可以在U-Boot 源码树的‘tools’目录中找到它。

通常为了节省闪存空间，你需要压缩 $\mu$ Clinux 映象，下面是你需要在U-Boot 源码树根目录中做的事：

```
$ gzip -9 -c /path/to/linux.bin > linux-bin.gz
$ gzip -9 -c /path/to/image.dxe > image-dxe.gz
$ ./tools/mkimage -A blackfin -O uClinux -T kernel -C gzip \
  -a 1000 -e 1000 -n "Blackfin uClinux System" \
  -d linux-bin.gz zImage.bin
$ ./tools/mkimage -A blackfin -O linux -T kernel -C gzip \
  -a 1000000 -e 1000000 -n "Blackfin uClinux System" \
  -d image-dxe.gz zImage.dxe
```

最后生成的gzip 压缩格式的 $\mu$ Clinux 映象命名为‘zImage.bin’ 和‘zImage.dxe’。

**注意：** 你需要指定 $\mu$ Clinux 分发版中构建的‘linux.bin’ 或‘image.dxe’ 文件的实际路径。

不带参数调用‘mkimage’ 会打印出它的用法：

```
$ ./tools/mkimage
```

实际上，如果仅需要生成 $\mu$ Clinux 的gzip 压缩映象，你不必这么麻烦；只需要在U-Boot 根目录或者 $\mu$ Clinux 根目录或者Linux 内核根目录中运行下面的命令即可：

```
$ make zImage.bin
$ make zImage.dxe
```

最后生成的‘zImage.bin’ 和‘zImage.dxe’ 放在当前目录下，并且在/tftpboot 目录下有一份拷贝。

当前版本的分发包中，只要在‘uClinux-dist’ 目录下敲‘make’ 就能自动完成所有这一切了。

## 2.3 通过串口烧写映像

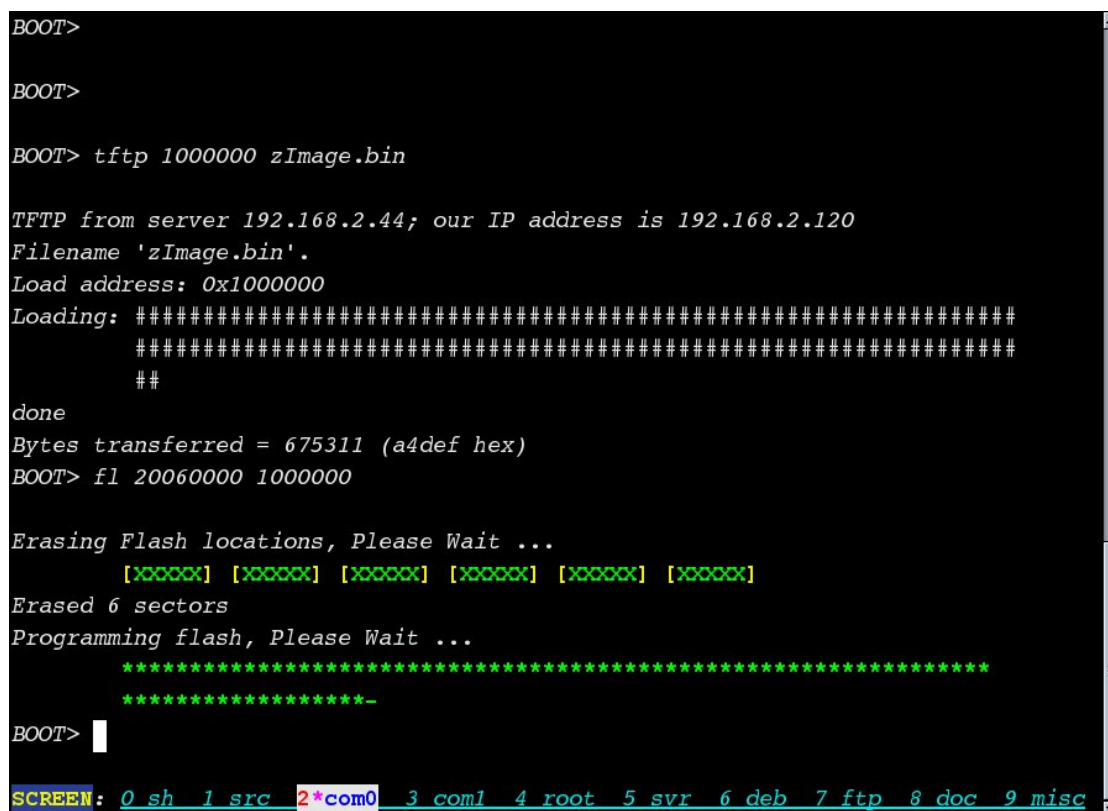
1. 把HHBF533/HHBF561 开发板的UART1 串口与PC 主机的串口正确连接。
2. 运行一个串口终端（典型地在MS Windows 中是Hyper Terminal，在GNU/Linux 中是minicom），连接到相应串口并设置波特率、校验位和检测位（典型地是：‘115200 8N1’，如图2 和图2）。
3. 在U-Boot 提示符后启动串口下载程序，后面用一个参数指定下载的内存地址，典型地：

```
BOOT> loadb 1000000
```

4. 在主机的串口终端中通过Kermit 协议发送映像（典型地， $\mu$ Clinux 映像‘zImage.bin’或者‘zImage.dxe’），参考图3 和图3。
5. 传输完成后，把映像烧写到板载FLASH 闪存中，比如：

```
BOOT> fl 20060000 1000000
```

如图4。



```
BOOT>
BOOT>
BOOT> tftp 1000000 zImage.bin

TFTP from server 192.168.2.44; our IP address is 192.168.2.120
Filename 'zImage.bin'.
Load address: 0x1000000
Loading: #####
#####
##
done
Bytes transferred = 675311 (a4def hex)
BOOT> fl 20060000 1000000

Erasing Flash locations, Please Wait ...
[XXXXX] [XXXXX] [XXXXX] [XXXXX] [XXXXX] [XXXXX]
Erased 6 sectors
Programming flash, Please Wait ...
*****
*****-
BOOT>
SCREEN: 0 sh 1 src 2*com0 3 com1 4 root 5 svr 6 deb 7 ftp 8 doc 9 misc
```

第 4 图: 把映像烧写到FLASH 中

**注意：**‘f1’还有第三个参数可以指定烧写映象的大小，如果没有指定则从环境变量中读入映象的大小（这是由下载的程序‘loadb’或‘tftp’设置的）。

## 2.4 通过以太网口烧写映象

1. 把所需的映象拷贝到主机TFTP服务器根目录中，典型地：

```
$ cp -f /path/to/u-boot.bin /tftpboot/  
$ cp -f /path/to/zImage.bin /tftpboot/  
$ cp -f /path/to/zImage.dxe /tftpboot/
```

**注意：**你需要映（‘u-boot.bin’和‘zImage.bin’或‘zImage.dxe’）象的正确路径！

**注意：**通常你不必做这一步，用‘make’构建映象的时候自动帮你完成了这一步。

2. 确定你主机上的TFTP服务器正确运行了：

```
$ /etc/init.d/tftpd restart
```

3. 把HHBF533/HHBF561开发板的UART1串口与PC主机的串口正确连接。
4. 把HHBF533/HHBF561开发板的网口与PC主机的网口直接用用于双机互连的双绞线连接。
5. 设置主机的IP地址，用于做开发板的目标TFTP服务器(默认为192.168.2.44)，典型地：

```
$ ifconfig eth0 192.168.2.44
```

6. 运行一个串口终端（典型地在MS Windows中是Hyper Terminal，在GNU/Linux中是minicom），连接到相应串口并设置波特率、校验位和检测位(典型地是：‘115200 8N1’，如图2和图2)。
7. 在U-Boot提示符后启动网口下载程序‘tftp’，后面的参数用于指定下载的内存地址和准备下载的映象文件名，典型地：

```
BOOT> tftp 1000000 zImage.bin  
BOOT> tftp 1000000 zImage.dxe
```

8. 传输完成后，把映象烧写到板载FLASH闪存中，比如：

```
BOOT> f1 20060000 1000000
```

**注意：**‘f1’还有第三个参数可以指定烧写映象的大小，如果没有指定则从环境变量中读入映象的大小（这是由下载的程序‘loadb’或‘tftp’设置的）。

第 2 表: HHBF533 的U-Boot 期存储映射

SDRAM(总共32 MB)		
0x00000000 - 0x00000fff	保留	
0x00001000 - 0x00ffffff	µClinux 运行代码	
0x01000000 - 0x015fffff	映象下载缓存	
0x01600000 - 0x01f1bfff	保留	
0x01f1c000 - 0x01f3bfff	U-Boot 栈空间	
0x01f3c000 - 0x01f3ffff	U-Boot 全局数据(环境变量)	
0x01f40000 - 0x01f7ffff	U-Boot 堆空间	
0x01f80000 - 0x01ffffff	U-Boot 运行代码	
FLASH(总共2 MB)		
0x20000000 - 0x2002ffff	U-Boot 二进制映象	
0x20030000 - 0x2003ffff	U-Boot 环境配置	
0x20040000 - 0x201fffff	µClinux 二进制映象	

第 3 表: HHBF561 的U-Boot 期存储映射

SDRAM(总共64 MB)		
0x00000000 - 0x00000fff	保留	
0x00001000 - 0x00ffffff	µClinux 运行代码	
0x01000000 - 0x015fffff	映象下载缓存	
0x01600000 - 0x01f1bfff	保留	
0x01f1c000 - 0x01f3bfff	U-Boot 栈空间	
0x01f3c000 - 0x01f3ffff	U-Boot 全局数据(环境变量)	
0x01f40000 - 0x01f7ffff	U-Boot 堆空间	
0x01f80000 - 0x01ffffff	U-Boot 运行代码	
0x02000000 - 0x03ffffff	保留给CORE B 使用	
FLASH(总共16 MB)		
0x20000000 - 0x2003ffff	U-Boot 二进制映象	
0x20040000 - 0x2005ffff	U-Boot 环境配置	
0x20060000 - 0x207fffff	µClinux 二进制映象	
0x20800000 - 0x20ffffff	保留给CORE B 使用	

## 2.5 典型的存储空间映射

## 3 U-Boot 的使用

开发板上电复位即从板载FLASH 闪存中运行U-Boot 映像，U-Boot 运行后即计数等待用户指示：如果用户没有动作，则计数结束后U-Boot 自动从板载FLASH 中引导 $\mu$ Clinux 映像(见第3.1 节)；如果在计数过程中用户按下‘y’ 键，则U-Boot 从默认的TFTP 服务器(典型地，192.168.2.44)中下载新的 $\mu$ Clinux 映像，烧写到FLASH 中并引导该映像；如果在计数过程中用户按下任意的其它键(除‘y’ 键)，则U-Boot 进入监听器程序等待用户输入命令，串口终端上出现U-Boot 提示符：BOOT> 。

### 3.1 U-Boot 引导Linux 映像

### 3.2 监听器命令

在U-Boot 环境下输入‘help’ 可以列出所有已实现并且编译进当前运行的U-Boot 映像中的命令。

如果输入‘help <command>’ 则可以显示该命令的详细描述和用法。

比如：

```
BOOT> help
BOOT> help fl
```

下面是所有已实现的监听器命令：

```
BOOT> help
```

```
?      - alias for 'help'
askenv - get environment variables from stdin
autoscr - run script from memory
base    - print or set address offset
bdinfo  - print Board Info structure
bootelf - Boot from an ELF image in memory
bootm   - boot application image from memory
bootp   - boot image via network using BootP/TFTP protocol
bootvx  - Boot vxWorks from an ELF image
cmp     - memory compare
coninfo - print console devices and informations
cp      - memory copy
crc32   - checksum calculation
dcache  - enable or disable data cache
echo    - echo args to console
```

第 5 图: U-Boot 引导Linux 映象之一

```

## Booting image at 20060000 ...
Image Name:   Blackfin uClinux System
Image Type:   Blackfin uClinux Kernel Image (gzip compressed)
Data Size:    675247 Bytes = 659.4 kB
Load Address: 00001000
Entry Point:  00001000
Verifying Checksum ... OK
Uncompressing Kernel Image ... OK
Linux version 2.6.2-uc0 (mhfan@debfan) (gcc version 3.3.3) #948 Mon Oct 18 12:04
BF533/561 Blackfin support (C) 2004 Analog Devices, Inc.
uClinux/Blackfin
Blackfin BF561 support by HHTech (www.hhcn.com)
Memory map:
  text = 0x001000-0x0ee000
  data = 0x0ee000-0x1bd3f0
  bss  = 0x1bd3f0-0x1bd3f0
  rootfs = 0x108ff0-0x1bd3f0
  stack= 0x1000000-0x1000000
On node 0 totalpages: 4096
  DMA zone: 0 pages, LIFO batch:1
  Normal zone: 4096 pages, LIFO batch:1
  HighMem zone: 0 pages, LIFO batch:1
Instruction Cache enabled
Building zonelist for node : 0
SCREEN: 0 sh 1 src 2*com0 3 com1 4 root 5 svr 6 deb 7 ftp 8 doc 9 misc
Building zonelist for node : 0
Kernel command line:
Configuring Blackfin Priority Driven Interrupts
PID hash table entries: 16 (order 4: 128 bytes)
Physical pages: 1000
Memory available: 14400k/15432k RAM, 0k/0k ROM (237k kernel code, 207k data)
Blackfin Scratchpad data SRAM: 4 KB
Calibrating delay loop... 107.31 BogoMIPS
Dentry cache hash table entries: 2048 (order: 1, 8192 bytes)
Inode-cache hash table entries: 1024 (order: 0, 4096 bytes)
Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
POSIX conformance testing by UNIFIX
NET: Registered protocol family 16
BlackFin BF561 serial driver version 1.00
ttyS0 at irq = 35 is a builtin BlackFin BF561 UART
Enabling Serial UART Interrupts
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
<DM9000> HHCN-eth0 I/O: 2c000300, VID: 90000a46, MAC: 18:60:19:07:1a:f0
Using noop io scheduler
uclinux[mtd]: RAM probe address=0x108ff0 size=0xffff4bc00
Creating 1 MTD partitions on "RAM":
0x00000000-0xffff4bc00 : "EXT2fs"
uclinux[mtd]: set EXT2fs to be root filesystem
NET: Registered protocol family 2
SCREEN: 0 sh 1 src 2*com0 3 com1 4 root 5 svr 6 deb 7 ftp 8 doc 9 misc

```

第 6 图: U-Boot 引导Linux 映像之二

```
NET: Registered protocol family 2
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 1024 bind 2048)
NET: Registered protocol family 1
NET: Registered protocol family 17
VFS: Mounted root (ext2 filesystem) readonly.
Freeing unused kernel memory: 36k freed (0xe4000 - 0xec000)

Blackfin 533 development board
HHTech, Co., Ltd. www.hhcn.com
Sat, 18 Sep 2004 02:51:56 +0800

Welcome to:

      _/_/_/_/_
     /_/_/_/_/_ \
    /_/_/_/_/_ \
   /_/_/_/_/_ \
  /_/_/_/_/_ \
 /_/_/_/_/_ \
/_/_/_/_/_ \

For further information see:
http://www.uclinux.org/
http://blackfin.uclinux.org/
SCREEN: 0 sh 1 src 2 *com0 3 com1 4 root 5 svr 6 deb 7 ftp 8 doc 9 misc

Blackfin 533 development board
HHTech, Co., Ltd. www.hhcn.com
Sat, 18 Sep 2004 02:51:56 +0800

Welcome to:

      _/_/_/_/_
     /_/_/_/_/_ \
    /_/_/_/_/_ \
   /_/_/_/_/_ \
  /_/_/_/_/_ \
 /_/_/_/_/_ \
/_/_/_/_/_ \

For further information see:
http://www.uclinux.org/
http://blackfin.uclinux.org/

BusyBox v0.60.5 (2004.10.14-09:45+0000) Built-in shell (msh)
Enter 'help' for a list of built-in commands.

# 
SCREEN: 0 sh 1 src 2 *com0 3 com1 4 root 5 svr 6 deb 7 ftp 8 doc 9 misc
```

```
erase - erase FLASH memory
fl - flush a file to FLASH memory
flinfo - print FLASH memory information
go - start application at address 'addr'
help - print online help
icache - enable or disable instruction cache
iminfo - print header information for application image
in - read data from an IO port
irqinfo - print information about IRQs
itest - return true/false on integer compare
loadb - load binary file over serial line (kermit mode)
loop - infinite loop on address range
md - memory display
mm - memory modify (auto-incrementing)
mtest - simple RAM test
mw - memory write (fill)
nm - memory modify (constant address)
oc - over clocking
out - write datum to IO port
ping - send ICMP ECHO_REQUEST to network host
printenv - print environment variables
protect - enable or disable FLASH write protection
rarpboot - boot image via network using RARP/TFTP protocol
reset - Perform RESET of the CPU
run - run commands in an environment variable
saveenv - save environment variables to persistent storage
setenv - set environment variables
sleep - delay execution for some time
tftpboot - boot image via network using TFTP protocol
version - print monitor version
BOOT>
```

### 3.3 U-Boot 的配置

U-Boot 支持用户根据实际的使用情况通过环境变量做一些动态的设置和调节，并且可以把这些设置‘永久’保存到FLASH 闪存中。

环境变量的通过使用命令‘setenv’来设置，‘printenv’打印出所有已设置的环境变量，为了把变量设置保存到FLASH 中可以使用命令‘saveenv’。调用‘setenv’时只指定变量名而未指定变量值则将从环境中删除这一变量。只要你还没有保存环境变量，你都是工作在一个临时的环境变量的内存拷贝中。如果你不小心刷除了环境变量所保存的FLASH 区域，不用担心，你还有一个默认的环境变量设置。

典型的使用如下：



```
BOOT> help setenv
```

```
setenv name value ...
```

```
- set environment variable 'name' to 'value ...'
```

```
setenv name
```

```
- delete environment variable 'name'
```

```
BOOT> printenv
```

```
bootcmd=run linuxrun
```

```
bootdelay=30
```

```
baudrate=115200
```

```
ipaddr=192.168.2.120
```

```
serverip=192.168.2.44
```

```
gatewayip=192.168.2.254
```

```
netmask=255.255.255.0
```

```
hostname=HHBF533
```

```
netdev=eth0
```

```
netretry=yes
```

```
uboot=0x20000000
```

```
linux=0x20060000
```

```
loadaddr=0x01000000
```

```
bootfile=zImage.bin
```

```
tftp_boot=tftp;bootm;echo
```

```
serial_boot=loadb;bootm;echo
```

```
menucmd=tftp;fl linux;bootm;echo
```

```
autoload=yes
```

```
autostart=no
```

```
linuxrun=oc 495;bootm 0x20060000;echo
```

```
stdin=serial
```

```
stdout=serial
```

```
stderr=serial
```

```
Environment size: 448/131068 bytes
```

```
BOOT> saveenv
```

```
Saving Environment to Flash...
```

```
Un-Protected 1 sectors
```

```
Erasing Flash... [XXXXXX]
```

```
Erased 1 sectors
```

```
Writing to Flash... *****
```

```
done
```

```
Protected 1 sectors
```

```
BOOT>
```

## 4 为HHBF533/HHBF561 创建U-Boot 映像

第一次创建U-Boot 映像时，你需要在U-Boot 源码树根目录中运行如下命令：

```
$ cd <ROOT_OF_U-BOOT>
$ make mrproper
$ make hhbf533_config
```

**注意：** 请用实际的U-Boot 源码树根目录路径来替换<ROOT\_OF\_U-BOOT>。

**注意：** 如果是HHBF561 开发板分发包请使用‘make hhbf561\_config’来配置。

然后，可以用下面的命令创建ROM 版U-Boot 映像：

```
$ make clean
$ make all
```

或者用‘make rom’，甚至直接敲‘make’就可以了。

下面的命令创建RAM 版U-Boot 映像：

```
$ make clean
$ make ram
```

下面的命令创建能在Visual DSP 中正确加载执行的RAM 版U-Boot 映像：

```
$ make clean
$ make dxe
```

**注意：** 新生成的‘u-boot’是ELF 格式映像，‘u-boot.bin’是相应的裸二进制映像，‘u-boot.ram’是可以在SDRAM 中调试的映像，‘u-boot.dxe’是用于Visual DSP 加载执行的映像。

**注意：** RAM 版U-Boot 映像的默认加载地址(或者说入口点、正文基地址)是在0x10000000。

这在你需要改进U-Boot 或者要添加功能时很有用：你可以先编译成RAM 版调试或测试；类似于后面要说到的在SDRAM 中调试Linux。

### 4.1 在SDRAM 中调试Linux

为了调试Linux 的方便你可以把μClinux 映像下载到板载SDRAM 中，并直接在U-Boot 中运行该映像，而不需要烧写到FLASH 中。

典型地，首先在U-Boot 中通过串口或网口下载μClinux 映像(请参考第2.3 节和第2.4 节)；然后在U-Boot 提示符后用‘go’命令从入口点运行映像：

```
BOOT> go 1000
```

**注意：** 典型的μClinux 映像入口点是在0x1000，所以你需要把映像下载到该地址！

除此之外，你可以用‘bootelf’命令来加载运行GNU ELF 格式的映像；这在你需要把一些代码或数据放置到片内SRAM 中时很有用！

## 5 为HHBF533/HHBF561 创建 $\mu$ Clinux 内核映象

如果你想要调整、修改 $\mu$ Clinux 的默认设置，你需要在 $\mu$ Clinux 根目录下运行：

```
$ cd <ROOT_OF_UCLINUX>
$ make menuconfig
```

如果你只需要修改Linux 内核的配置，那么你可以：

```
$ cd <ROOT_OF_UCLINUX>
$ make linux_menuconfig
```

或者：

```
$ cd <ROOT_OF_LINUX_KERNEL>
$ make -C menuconfig
```

**注意：** 请用实际的 $\mu$ Clinux 源码树根目录路径来替换<ROOT\_OF\_UCLINUX>；用实际使用的Linux 内核根目录路径来替换<ROOT\_OF\_LINUX\_KERNEL>。

如图7 8 8。

如果你想调整 $\mu$ Clibc 库的配置，你还需要运行：

```
$ cd <ROOT_OF_UCLINUX>
$ make -C uClibc menuconfig
```

如图9。

为了创建 $\mu$ Clinux 映象，你需要运行：

```
$ make
```

## 6 关于 $\mu$ Clinux 中一些驱动程序的使用说明

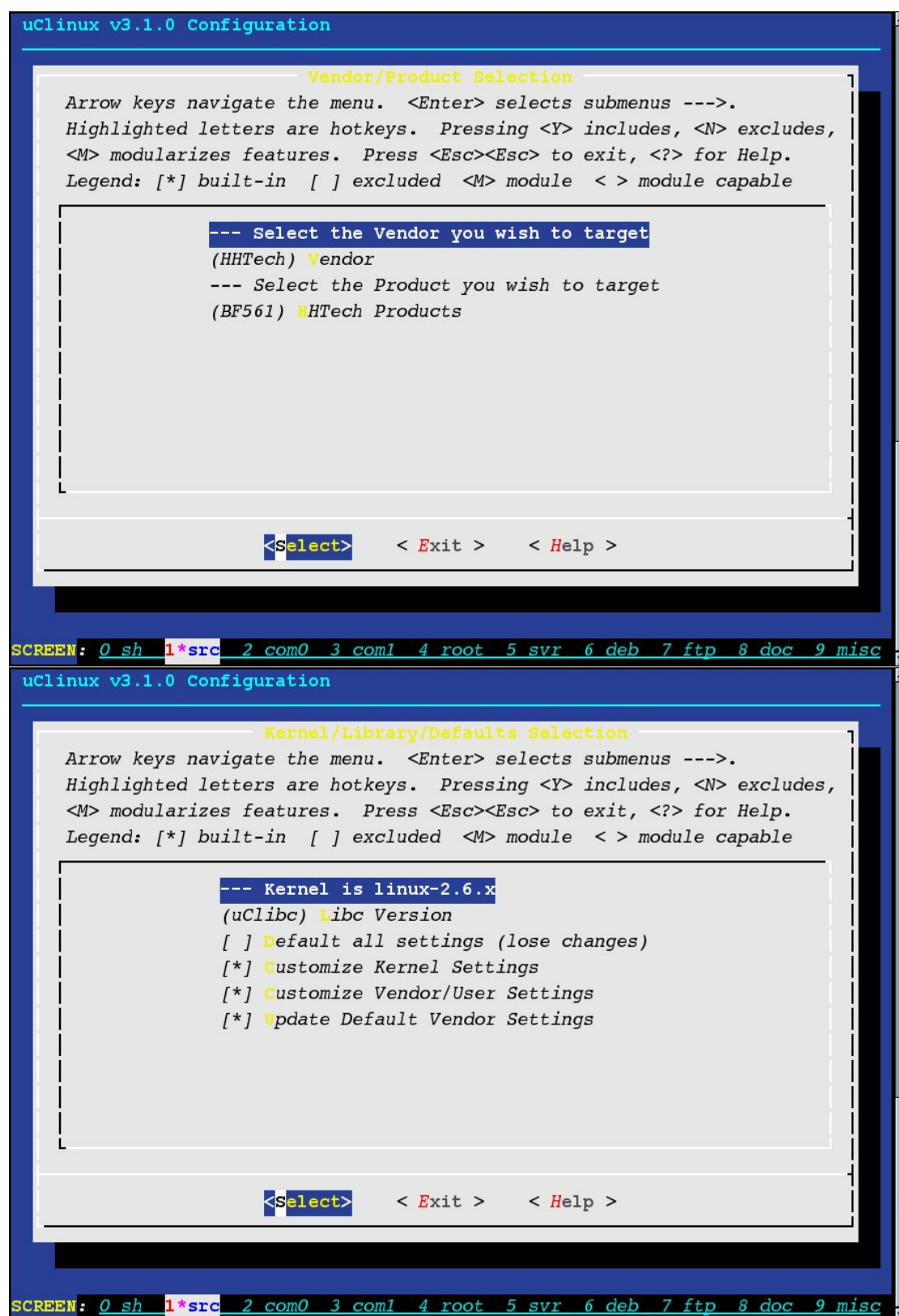
### 6.1 BF561 CORE B 的二级引导驱动

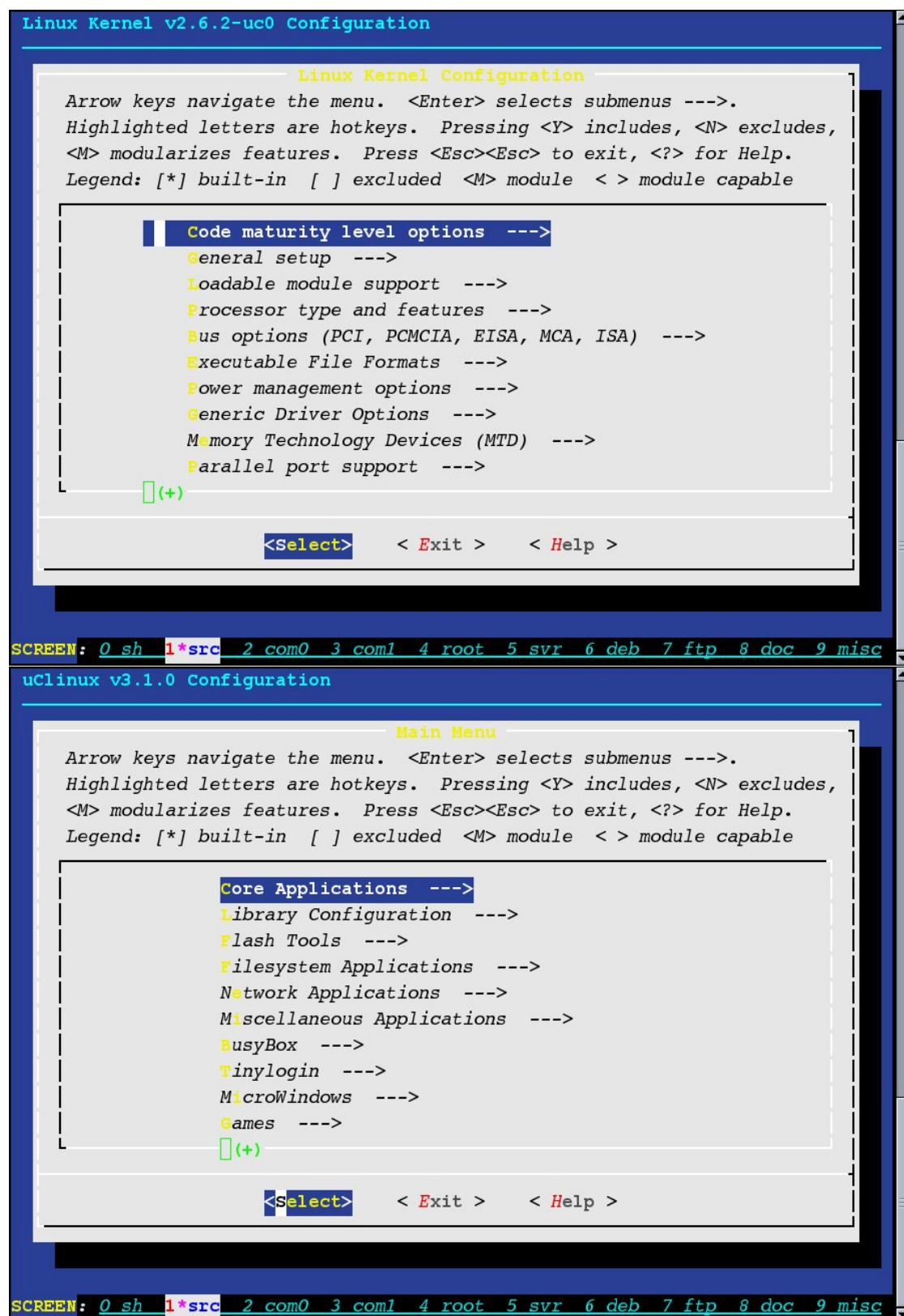
**注意：** 本节仅限于HHBF561 开发板。

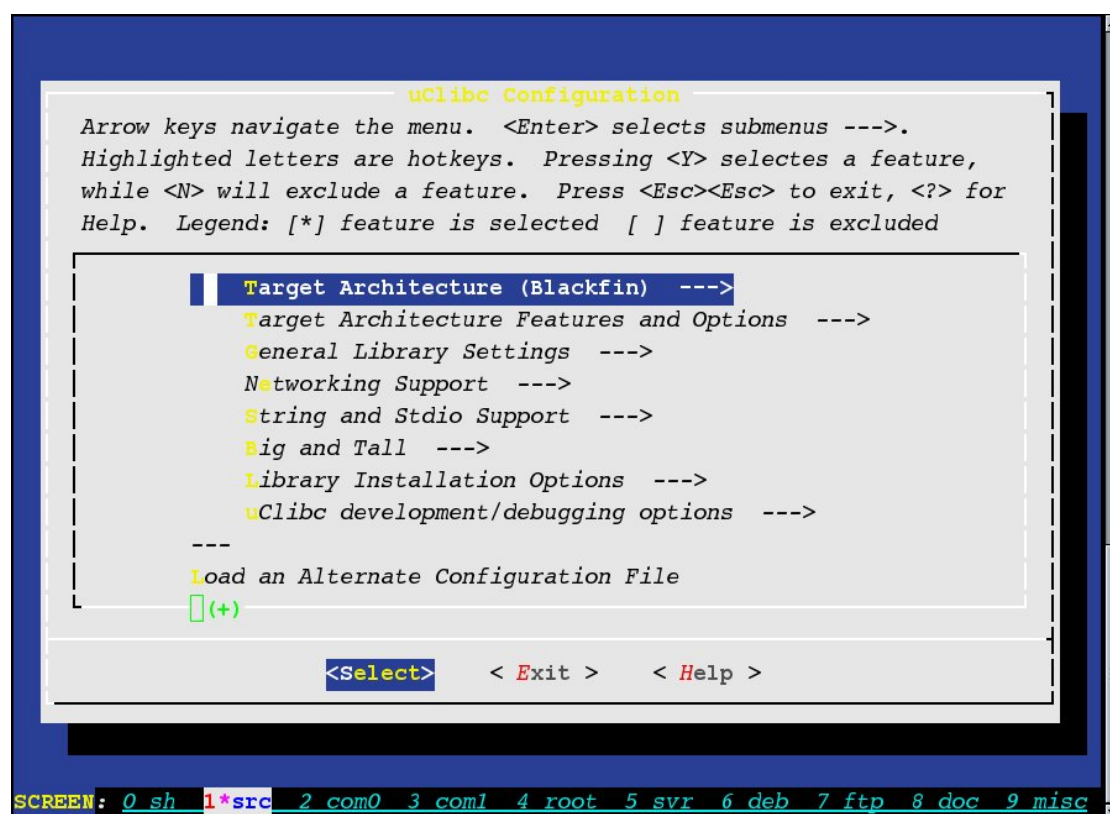
为了使用这个驱动，你必须在Linux 内核配置菜单中选中它，使之能编译进内核中。具体是在‘Processor type and features’子菜单中的‘BF561 CORE B second stage loader’项。

重新编译内核，并用之运行之后有两个用户接口：

**/proc/coreb/load**：用于从FLASH 中加载代码到适当位置。







第 9 图: μClibc Menuconfig

**/proc/coreb/hold** : 用于启动CORE B, 开始执行代码<sup>④</sup>。

**注意：**当前的二级引导驱动中已被限定为只能从0x20800000 地址之后的FLASH 中加载代码；如果有额外的需求，则需要修改源代码（linux-2.6.x/arch/bfinnommu/coreb/）了。

例如，在 $\mu$ Clinux 启动运行之后：

```
HHBF> -# echo 0x20a00000 > /proc/coreb/load
Loading code for CORE B from 0x20A00000 ... Done!
HHBF> -# echo 0 > /proc/coreb/hold
CORE B is running!
this is from L1!
this is from L2!
this is from L3!
HHBF> -#
```

你可以用以下命令查看CORE B 的状态：

```
HHBF> -# cat /proc/coreb/load
DXE base: 0x20A00000
DXE skip: 0
HHBF> -# cat /proc/coreb/hold
SICA_SYSCR: 0x0020
SICB_SYSCR: 0x0020
HHBF> -#
```

## 6.2 以太网驱动和网络应用服务

关于以太网驱动，目前已测试过的应用程序包括inetd, telnetd, ftpd, telnet, nfsmount, socket, flood ping等。

这些程序的使用都很简单，这里不再赘述。需要注意的是，nfsmount 需要加上‘-o nolock’ 选项才能正常工作，如：

```
HHBF> -# mount -o nolock 192.168.2.240:/tmp /mnt
```

## 6.3 音频驱动

为了能正确编译音频驱动，你需要在Linux 内核配置菜单的‘Sound’ 子菜单中选中‘Sound card support’, ‘1836 Audio support for BF53x’, ‘Advanced Linux Sound Architecture’, ‘OSS Mixer API’, ‘OSS PCM (digital audio) API’。

---

<sup>④</sup>此时COREB 的入口点是：0xFF600000

### 6.3.1 简单的音频输入输出演示

这个音频I/O 演示简单地把音频输入接口采集的数据不经处理地送到音频输出接口（即talktrough 的工作模式）：

```
HHBF> -# echo 1 > /proc/asound/card0/talktrough
```

关闭talktrough 模式：

```
HHBF> -# echo 0 > /proc/asound/card0/talktrough
```

### 6.3.2 音频录 / 放演示

用于录 / 放音频的应用程序分别是‘ossrec’和‘ossplay’，需要在 $\mu$ Clinux 的‘Customize Vendor/User Setting’配置菜单中‘Blackfin test programs’子菜单‘Audio test program’项来使之正确编译安装。

在 $\mu$ Clinux 启动进入SHELL 提示符之后：

```
HHBF> -# mount -o nolock 192.168.2.240:/tmp /mnt
HHBF> -# ossrec hhbfa.raw
HHBF> -# ossplay foo.wav
```

不带参数地运行‘ossrec’和‘ossplay’可以得到它们的用法。

## 6.4 视频驱动

为了编译视频驱动，你需要在Linux 内核配置菜单的‘Multimedia devices’子菜单中选中‘Video For Linux’和‘HHBF board video support’项（它实际上还依赖选择了‘I2C’驱动中的一些选项：‘I2C bit-banging interfaces’& ‘HHBF533/561 I2C support’），还有‘Video For Linux on HHBF’子菜单中的‘ADV7170/ADV7171 DAC chip’和‘Philips SAA7113H ADC chip’。

### 6.4.1 简单的视频输入输出演示

这个视频I/O 演示简单地把视频输入接口采集的数据不经处理地送到视频输出接口（即talktrough 的工作模式），

需要在Linux 内核配置菜单的‘Multimedia devices’子菜单中选中‘Video For Linux’和‘Video For Linux on HHBF’子菜单中的‘Video talktrough mode’。



### 6.4.2 用于视频编解码演示的应用程序

用于视频编解码演示（测试）的应用程序（包括两个开发板应用程序和两个PC 机应用程序）实际上都是网络（socket 方式连接）通信程序；它们可以通过选中 $\mu$ Clinux 的‘Customize Vendor/User Setting’ 配置菜单中‘Blackfin test programs’子菜单‘Video Input/Output test utilities’ 项来使之正确编译安装。

#### 视频编码演示

假设你已经把视频编码器代码烧写到0x20800000中了，在 $\mu$ Clinux 启动进入SHELL 提示符之后：

[HHBF561 board]	[PC host]
HHBF> -# echo 0x20800000 > /proc/coreb/load	\$
HHBF> -# vsvr	\$ cd <ROOT_OF_UCLINUX>/user/video
	\$ ./hvc1t

然后在PC 主机上会新生成一个‘test/5.xvid’ 其中的内容是CORE B 做视频采集并进行MPEG4 编码后经由CORE A 发送出来的码流。

#### 视频解码演示

假设你已经把视频解码器代码烧写到0x20900000中了，在 $\mu$ Clinux 启动进入SHELL 提示符之后：

[HHBF561 board]	[PC host]
HHBF> -# echo 0x20900000 > /proc/coreb/load	\$ cd <ROOT_OF_UCLINUX>/user/video
HHBF> -# vc1t	\$ ./hvsvr

这里，PC 主机把一个码流文件（‘test/vstream.dat’）的内容发送给CORE A，再提交给CORE B 解码输出到视频输出接口。

### 6.4.3 Frame Buffer 驱动

为了使用Frame Buffer 驱动，你必须在内核配置菜单的‘Graphic Support’ 子菜单中选中‘Support for frame buffer devices’ 和‘HHBF frame buffer support’。

Frame Buffer 驱动的应用程序接口是‘/dev/fb\*’（其中，‘\*’ 表示第几个Frame Buffer 设备，相应于从设备号；一般系统中只有一个Frame Buffer 设备，所以实际的接口是‘/dev/fb0’）。

‘/dev/fb0’ 实际上是“显存”的裸映射，因此读写Frame Buffer 的设备文件就相当于读写显存了。

```
HHBF> -# mount -o nolock 192.168.2.240:/tmp /mnt
HHBF> -# cat /dev/fb0 > /mnt/hhbffb.raw
HHBF> -# cat /mnt/test.raw > /dev/fb0
```

由于显存快照是像素对应的‘裸’文件，为了能正确的转换和显示，还需要两个辅助程序：bmp2raw 和raw2bmp，用于在裸图像文件和位图文件格式之间转换。你可以在内核源码树的linux-2.6.x/drivers/video/hhbffb/test 目录中找到它们。

```
$ bmp2raw test.bmp test.raw
$ raw2bmp hhbffb.raw hhbffb.bmp
```

**注意：**这两个辅助程序的实现很简单，基本上就是添加 / 删除BMP 图像文件格式的文件头，这里处理的是RGB 24 的像素。包括Frame Buffer 驱动，当前都只实现了RGB 24 像素的处理。

通过这些操作你可以把一幅720x576 像素的BMP 图像刷到视频输出显式上；也可以反向地截取视频输出屏的图像内容存成一幅720x576 像素的BMP 文件。

#### 6.4.4 V4L2 驱动

仍在实现中.....

### 6.5 片内SRAM 的测试程序

在 $\mu$ Clinux 的内核配置菜单‘Processor type and features’子菜单中选‘On-chip SRAM test driver’项，并重新编译。

为了能够使用片内SRAM，我们必须使用ELF 格式的内核映像文件——编译后在Linux 内核根目录下的‘vmlinux’即是。

在U-Boot 中下载‘vmlinux’后，必须用‘bootelf’引导。假设你已经把‘vmlinux’拷到PC 机TFTP 服务的根目录下：

```
BOOT> tftp 10000000 vmlinux; bootelf
```

正确运行 $\mu$ Clinux 后在‘/proc’文件系统中会多出一个接口‘sram’，用于测试 / 演示片内SRAM 的使用：

```
HHBF>-# cat /proc/sram
```

之后系统打印的信息能让你确认是否能够正常使用片内SRAM。

### 6.6 IPC 测试程序

感谢南京天朗电子的陶猛提供了部分测试代码。

#### POSIX threads 测试

把‘<ROOT\_OF\_UCLINUX>/user/hhtech/pthtest’中的代码编译安装进 $\mu$ Clinux 的ROMFS 中，系统运行之后执行这些代码，你应该能得到你想要的结果。

另外， $\mu$ Clinux 中本来就有一个多进程测试程序，你可以在‘Customize Vendor/User Setting’配置菜单的‘Miscellaneous Applications’子菜单中选‘pThreads threaddemos’。

#### System V IPC 测试

把‘<ROOT\_OF\_UCLINUX>/user/hhtech/mqtest’中的代码编译安装进 $\mu$ Clinux 的ROMFS 中，系统运行之后执行这些代码，你应该能得到你想要的结果。

## **7 内核和应用程序调试**

.....

## **8 常见问题与解答**

正在收集中.....