User Manual of
U-Boot for HHBF533/HHBF561

Version 0.1

Meihui Fan
范 美 辉
<mhfan@ustc.edu>

October 18, 2004

[*]http://www.hhcn.com

Contents

List of Tables

1 Overview

The “U-Boot” Universal Bootloader project provides firmware with full source code under GPL.

U-Boot is a boot loader for Embedded boards, originally based on PowerPC and ARM processors, we port it to our HHBF533/HHBF561 development boards, which can be installed in a boot ROM and used to initialize and test the hardware or to download and run application code.

The development of U-Boot is closely related to Linux: some parts of the source code originate in the Linux source tree, we have some header files in common, and special provision has been made to support booting of Linux images.

Some attention has been paid to make this software easily configurable and extendable. For instance, all monitor commands are implemented with the same call interface, so that it’s very easy to add new commands. Also, instead of permanently adding rarely used code (for instance hardware test utilities) to the monitor, you can load and run it dynamically. Some of the features are:

- Host connectivity via RS-232 serial port or Ethernet
- Command line interface via RS-232
- Image downloads via TFTP, or Kermit protocol
- Support for compressed images (download and flash load)
- Flash Image System for managing multiple flash images
- Flash stored configuration
- Boot time script execution

For more information about U-Boot, refer to http://u-boot.sourceforge.net and read the “README” file in the U-Boot source tree.

2 Image Install Instructions

There are two ways to download a prepaired U-Boot or Linux ROM image into the onboard SDRAM, and programming to the flash. One way is through the UART1 serial port with Kermit protocol, the other is via Ethernet with TFTP protocol. The former is slower but more sophisticated.

When you get a ‘raw’ development board, you must learn to using the ADI VDSP to download the RAM U-Boot image through the ADI Emulator, then using the running RAM U-Boot to download and programming a ROM U-Boot into the onboard flash. That is the beginning.

2.1 The First Time

1. Get or Build(refer to section 4) a RAM and a ROM U-Boot image, make sure the RAM version is in ELF binary format and the ROM version is a raw binary image.
2. Connect the development board to the ADI Emulator via JTAG, and to a host via RS-232 serial port.
3. Running the ICE Test utility.
4. Running the VDSP to detect the BF533/BF561 development board.
5. Download the ELF binary format RAM U-Boot image.
6. Modify the PC Counter register value to the entry point of the RAM U-Boot image, typically 0x1000000.
7. Run a serial terminal(Typically HyperTerminal in MS Windows, minicom in GNU/Linux) connected to the corresponding port, and configure baudrate, parity bit, and bits correctly(typically 115200 8N1).
8. Press <F5> to run U-Boot image.
9. Using the running U-Boot to download and programming a ROM U-Boot image to the onboard flash. At first, download the image, typically

   ```
   BOOT> loadb 1000000
   ```

   After this, send a ROM U-Boot image by Kermit protocol from the host terminal.

   When transfered successfully, programming to the flash, typically

   ```
   BOOT> fl 20000000 1000000 20000
   ```

   Please refer to the last two subsections of this section(2.3, 2.4) and section 3 for more details.

2.2 Prepairing an Image for Download

If you needn't to compress or composit your binary images both U-Boot and μClinux, you can jump over this section. Otherwise you must learn to use the 'mkimage' utility to generate a compressed or multiple image.

The 'mkimage' is distributed with the U-Boot, you can find it in 'tools' directory under the root directory of the U-Boot source tree.

Typically, we need compress the μClinux raw binary image(the following commands all running in the root directory of the U-Boot source tree) for reducing the flash using:

```
$ gzip -9 -c /path/to/linux.bin > linux-bin.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
```

That generate a gzip compressed μClinux image named with ‘zImage.bin’.

NOTE: Please specify the real path of the builded linux.bin of your got μClinux distribution.

Run mkimage with no arguments to show usage:

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

2.3 Programming Through the Serial Port

1. Connect UART1 on the HHBF533/HHBF561 development board to the PC serial port.
2. Running and configuring a host serial terminal(Typically Hyper Terminal in MS Windows, or minicom on GNU/Linux) on the corresponding port in ‘115200 8N1’ configuration.
3. Execute the ‘loadb’ command with a argument specifying loading address after the U-Boot prompt. e.g.:

   ```
   BOOT> loadb 1000000
   ```

4. Send a binary image(typically u-boot.bin or zImage.bin) in the host terminal by Kermit protocol.
5. Programming the image into the onboard flash from loading address. e.g.:

   ```
   BOOT> fl 20000000 1000000
   ```

   NOTE: If didn’t specify image-size argument, the command will automatically detect it from the environment variable which set after download transfer successfully.

2.4 Programming Through the Ethernet

1. Copy the image into the TFTP server root directory, typically

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

2. Check the TFTP server daemon is correctly running.

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

3. Connect UART1 on the HHBF533/HHBF561 development board to the PC serial port.

4. Running and configuring a host serial terminal(Typically Hyper Terminal in MS Windows, or minicom on GNU/Linux) on the corresponding port in ‘115200 8N1’ configuration.

5. Execute the ‘tftp’ command with arguments specifying loading address and image file name to download a binary image file after the U-Boot prompt. Typically,

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

6. Programming the image into the onboard flash from loading address. e.g.:

```
BOOT> fl 20000000 1000000
```

   NOTE: If didn’t specify image-size argument, the command will automatically detect it from the environment variable which set after download transfer successfully.

2.5 Typicall Memory Map

Table 1: HHBF533 Memory Map in U-Boot Stage

SDRAM(Total 32 MB)	
`0x00000000 - 0x00000fff`	Reserved
`0x00001000 - 0x00ffffff`	μClinux Running Usage
`0x01000000 - 0x015fffff`	Reserved for Downloading
`0x01600000 - 0x018fffff`	U-Boot Running Usage
`0x01900000 - 0x01ffffff`	Reserved
FLASH(Total 2 MB)	
`0x20000000 - 0x2002ffff`	U-Boot Binary Image
`0x20030000 - 0x2003ffff`	U-Boot Environment Storage
`0x20040000 - 0x201fffff`	μClinux Binary Image

Table 2: HHBF561 Memory Map in U-Boot Stage

SDRAM(Total 64 MB)	
`0x00000000 - 0x00000fff`	Reserved
`0x00001000 - 0x00ffffff`	μClinux Running Usage
`0x01000000 - 0x015fffff`	Reserved for Downloading
`0x01600000 - 0x018fffff`	U-Boot Running Usage
`0x01900000 - 0x03ffffff`	Reserved
FLASH(Total 16 MB)	
`0x20000000 - 0x2003ffff`	U-Boot Binary Image
`0x20040000 - 0x2005ffff`	U-Boot Environment Storage
`0x20060000 - 0x207fffff`	μClinux Binary Image
`0x20800000 - 0x20ffffff`	Reserved

3 Using U-Boot

3.1 Monitor Commands

Type “help” for showing all the monitor commands implementation and compiled in.
Type “help <command>” for detailed description of a special monitor command.
e.g.:

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

All implemented command show as following:

```
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
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>
```

3.2 Configuring U-Boot

U-Boot supports user configuration using Environment Variables which can be made persistent by saving to Flash memory.

Environment Variables are set using ”setenv”, printed using ”printenv”, and saved to Flash using ”saveenv”. Using ”setenv” without a value can be used to delete a variable from the environment. As long as you don’t save the environment you are working with an in-memory copy. In case the Flash area containing the environment is erased by accident, a default environment is provided.

4 Build U-Boot for HHBF533/HHBF561

When building the U-Boot in the first time, you must first run the following commands in the root directory of the U-Boot source tree:

```
$ cd <ROOT_OF_U-BOOT>
$ make clean
$ make mrproper
$ make ezkit533_config
```

NOTE: Please replace the <ROOT_OF_U-BOOT> with the real path of the root directory of the U-Boot source tree.

NOTE: If you got HHBF561, you could also do ‘make ezkit533_config’!

Then to build a ROM U-Boot, typing:

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

to build a RAM U-Boot, typing:

```
$ make clean
$ make u-boot.ram
```

NOTE: The default loading address(entry point or text base address) of the RAM U-Boot image is at 0x1000000.

5 Frequently Asked Questions

Collecting ......