9.5 U-Boot

As I said earlier, U-Boot is a richly documented bootloader. The *README* file included with the package, for example, covers the use of U-Boot extensively. Among other things, it discusses the package's source code layout, the available build options, U-Boot's command set, and the typical environment variables used in U-Boot. In the following, I will cover the essential aspects of U-Boot and provide practical examples of its use. An in-depth discussion of U-Boot would, however, require a book of its own. For this reason, I encourage you to print a copy of the *README* provided with U-Boot and have a look at the other documentation written by the project maintainer.

9.5.1 Compiling and Installing

Start by downloading and extracting the latest version of U-Boot in your *${PRJROOT}/bootldr* directory. As of this writing, the latest U-Boot version is 0.2.0. Once extracted, move to the package's directory:

```
$ cd ${PRJROOT}/bootldr/u-boot-0.2.0
```

Physical RAM and Flash Location

The board used in the following explanations has 16 MB of RAM and 8 MB of flash. The RAM is mapped from address 0x0000000 to address 0x00FFFFFF, and the flash is mapped from address 0x40000000 to address 0x407FFFFF. The documentation provided with U-Boot discusses its use of the physical memory of targets.

Before you can build U-Boot, you need to configure it for your target. The package includes a number of preset configurations for quite a few boards. So, a configuration may very well exist for your target already. Look at the *README* file to see if your board is supported. For each supported board, U-Boot's Makefile includes a `*BOARD_NAME*_config` target, which is used to configure U-Boot's build for the designated board. The configuration target for the TQM860L board I use for my control module, for example, is `TQM860L_config`. Once you have determined the proper Makefile target to use, configure U-Boot's build process:

```
$ make TQM860L_config
```

Now, build U-Boot:

```
$ make CROSS_COMPILE=powerpc-linux-
```

In addition to generating bootloader images, the build process will compile a few tools to be used on the host for conditioning binary images before downloading them off to the target to a running U-Boot. Table 9-2 lists the files generated during U-Boot's compilation.

Table 9-2. Files generated during U-Boot's compilation

Filename	Description
System.map	The symbol map
u-boot	U-Boot in ELF binary format
u-boot.bin	U-Boot raw binary image that can be written to the boot storage device
u-boot.srec	U-Boot image in Motorola's S-Record format

You can now download the U-Boot image onto your target's boot storage device using the appropriate procedure. If you already have U-Boot, or one its ancestors (PPCBoot or ARMBoot) installed on your target, you can use the installed copy to update U-Boot to a new version, as we shall see in Section 9.5.10. If you have another bootloader installed, follow the procedures described in that bootloader's documentation for updating bootloaders. Finally, if you have no bootloader whatsoever installed on your target, you need to use a hardware programming device, such as a flash programmer or a BDM debugger, to copy U-Boot to your target.

Whichever method you use to copy the actual bootloader image to your target, make a copy of the relevant bootloader images to your *${PRJROOT}/images* directory. For my control module for example, I copy the images as follows:

```
$ cp System.map ${PRJROOT}/images/u-boot.System.map-0.2.0
$ cp u-boot.bin ${PRJROOT}/images/u-boot.bin-0.2.0
$ cp u-boot.srec ${PRJROOT}/images/u-boot.srec-0.2.0
```

If you intend to debug U-Boot itself, copy the ELF binary also:

```
$ cp u-boot ${PRJROOT}/images/u-boot-0.2.0
```

Finally, install the host tool generated by the U-Boot build:

```
$ cp tools/mkimage ${PREFIX}/bin
```

9.5.2 Booting with U-Boot

Once U-Boot is properly installed on your target, you can boot it while being connected to the target through a serial line and using a terminal emulator to interface with the target. As I said in Chapter 4, not all terminal emulators interact cleanly with all bootloaders. In the case of U-Boot, avoid using *minicom* for file transfers, since problems may occur during such transfers.

Here is a sample boot output for my control module:

```
U-Boot 0.2.0 (Jan 27 2003 - 20:20:21)

CPU:   XPC860xxZPnnD3 at 80 MHz: 4 kB I-Cache 4 kB D-Cache FEC present
Board: TQM860LDB0A3-T80.201
DRAM:  16 MB
FLASH:  8 MB
In:    serial
Out:   serial
Err:   serial
Net:   SCC ETHERNET, FEC ETHERNET
PCMCIA:   No Card found
Hit any key to stop autoboot:  5
```

As you can see, U-Boot prints version information and then provides some detail regarding the hardware it is running on. As soon as it boots, a five second timer starts ticking at the last output line. If you do not press a key during those five seconds, U-Boot boots its default configuration. By pressing a key, you get a prompt:

```
=>
```

One of the first things you probably want to try is obtaining help from U-Boot:

```
=> help
askenv  - get environment variables from stdin
autoscr - run script from memory
base    - print or set address offset
bdinfo  - print Board Info structure
bootm   - boot application image from memory
bootp   - boot image via network using BootP/TFTP protocol
bootd   - boot default, i.e., run 'bootcmd'
cmp     - memory compare
coninfo - print console devices and informations
cp      - memory copy
crc32   - checksum calculation
date    - get/set/reset date & time
dhcp    - invoke DHCP client to obtain IP/boot params
diskboot- boot from IDE device
echo    - echo args to console
erase   - erase FLASH memory
flinfo  - print FLASH memory information
go      - start application at address 'addr'
help    - print online help
ide     - IDE sub-system
iminfo  - print header information for application image
loadb   - load binary file over serial line (kermit mode)
loads   - load S-Record file over serial line
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)
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
               and env variables ipaddr and serverip
version - print monitor version
?       - alias for 'help'
```

As you can see, U-Boot has a lot of commands. Fortunately, U-Boot also provides per-command help:

```
=> help cp
cp [.b, .w, .l] source target count
    - copy memory
```

When U-Boot appends the `[.b, .w, .l]` expression to a command, this means that you need to append one of the indicated strings to the command to invoke the desired version of the command. In the case of *cp*, for example, there are three versions, *cp.b*, *cp.w*, and *cp.l*, for copying bytes, words, and longs, respectively.

U-Boot is strict in its argument parsing. It expects most values to be provided in hexadecimal form. In the case of the *cp* command, for example, this means that the source address, the target address, and the byte count must be provided in hexadecimal values. You don't need to prepend or append those values with any sort of special characters, such as "0x" or "h". If your source address is 0x40000000, for example, simply type `40000000`.

U-Boot accepts any unique subset of characters that starts a command name. If you want to use the *erase* command, for example, you can type just the first three letters, *era*, since *erase* is the only command to start with those three first letters. On the other hand, you can't type *lo* and expect U-Boot to understand it, since there are three commands that start with those letters: *loadb*, *loads*, and *loop*.

9.5.3 Using U-Boot's Environment Variables

Once U-Boot is up and running, you can configure it by setting the appropriate environment variables. The use of U-Boot environment variables is very similar to the use of environment variables in Unix shells, such as *bash*. To view the current values of the environment variables on your target, use the *printenv* command. Here is a subset of the environment variables found on my control module:

```
=> printenv
bootdelay=5
baudrate=115200
loads_echo=1
serial#= ...
ethaddr=00:D0:93:00:05:E3
netmask=255.255.255.0
ipaddr=192.168.172.10
serverip=192.168.172.50
clocks_in_mhz=1
stdin=serial
stdout=serial
stderr=serial

Environment size: 791/16380 bytes
```

Each environment variable has a different meaning. Some environment variables, such as `bootdelay`, `serial#`, or `ipaddr`, have predetermined uses that are recognized by U-Boot itself. See the *README* file for a complete discussion of U-Boot's environment variables and their meanings.

As with Unix shells, you can add environment variables in U-Boot. To do so, you must use the *setenv* command. Here is an example session where I add a few environment variables to my control module (the third command must be entered as a single line, even though it appears broken on the page):

```
=> setenv rootpath /home/karim/ctrl-rootfs
=> setenv kernel_addr 40100000
=> setenv nfscmd setenv bootargs root=/dev/nfs rw nfsroot=\$(serverip):\$(rootpath)
   ip=\$(ipaddr):\$(serverip):\$(gatewayip):\$(netmask):\$(hostname)::off panic=1\;
   bootm \$(kernel_addr)
=> setenv bootcmd run nfscmd
```

In this case, I set U-Boot to boot from the kernel found at 0x40100000 and to mount its root filesystem using NFS. Notice that I used the `\` character to tell U-Boot that the character following `\` should not be interpreted as a special character. This is how the `nfscmd` looks like, for example, after U-Boot has read it:

```
=> printenv nfscmd
nfs2cmd=setenv bootargs root=/dev/nfs rw nfsroot=$(serverip):$(rootpath)
ip=$(ipaddr):$(serverip):$(gatewayip):$(netmask):$(hostname)::off panic=1;bootm
$(kernel_addr)
```

The *setenv* command adds the environment variables to the current session only. Hence, if you reset the system, any environment variable you set only with *setenv* will be lost. For the environment variables to survive reboots, they must be saved to flash. This is done using the *saveenv* command:

```
=> saveenv
Saving Enviroment to Flash...
Un-Protected 1 sectors
Erasing Flash...
. done
Erased 1 sectors
Writing to Flash... done
Protected 1 sectors
```

Be careful when using *saveenv*, since it will save all the environment variables currently defined, even those you may have been using temporarily. Before using *saveenv*, use *printenv* to take a look at the currently defined environment variables to make sure you are saving only the necessary variables. If you want to delete a variable, simply use *setenv* on the variable without providing any values. Here's an example:

```
=> setenv RAMDisk_addr 40500000
=> printenv RAMDisk_addr
RAMDisk_addr=40500000
=> setenv RAMDisk_addr
=> printenv RAMDisk_addr
```

```
## Error: "RAMDisk_addr" not defined
```

Note that the = character is not treated as a special character by *setenv*. In fact, it is seen as another character in the string making up the environment variable, as we saw earlier in this section. The following command, for example, is flawed (notice the extra = displayed by *printenv* in comparison to the same *printenv* shown in the previous capture):

```
=> setenv RAMDisk_addr = 40500000
=> printenv RAMDisk_addr
RAMDisk_addr=  = 40500000
```

9.5.4 Creating Boot Scripts

U-Boot environment variables can be used to create boot scripts. Such boot scripts are actually environment variables containing a set of U-Boot command sequences. By using a combination of the *run* command and the ; (semicolon) operator, you can make U-Boot run boot scripts. The environment variables I set in the previous section, for instance, are actually part of a boot script, `nfscmd`.

The key to the way the script I provided in the previous section works is the `bootcmd` environment variable. This variable is recognized by U-Boot as the script to run automatically when the system is booted. I set this variable as `run nfscmd`. In other words, U-Boot should run the `nfscmd` script to boot the system. In turn, this environment variable is a set of commands of its own. First, it sets the `bootargs` environment variable, which U-Boot passes to the kernel as its boot parameters, and then uses the *bootm* command to boot the kernel located at `$(kernel_addr)`. The semicolon separates commands. The use of the `$(`*`VAR_NAME`*`)` operator tells U-Boot to replace the entire string with the value of the *`VAR_NAME`* environment variable. Hence, when `nfscmd` runs, `$(kernel_addr)` is replaced by `40100000`, which is the value I set earlier. In the same way, `$(rootpath)` is replaced by `/home/karim/ctrl-rootfs`, and the rest of the environment variables included in `nfscmd` are replaced by their respective values.

Though it would have been possible to set `bootcmd` to contain the entire boot script instead of using `run nfscmd`, it would have been much harder then to specify alternative boot scripts at the boot command line. By using the *run* command in the `bootcmd` script, multiple boot scripts can coexist within U-Boot's environment variables. You can then change the system's default boot using:

```
=> setenv bootcmd run  OTHER_BOOT_SCRIPT
```

Or you can run boot scripts directly from the command line without changing the value of the `bootcmd` environment variable:

```
=> run  OTHER_BOOT_SCRIPT
```

Scripts are a very useful feature of U-Boot and you should use them whenever you need to automate a certain task in U-Boot.

9.5.5 Preparing Binary Images

Since the raw flash is not structured like a filesystem and does not contain any sort of file headers, binary images downloaded to the target must carry headers for U-Boot to recognize their content and understand how to load them. The *mkimage* utility we installed earlier was packaged with U-Boot for this purpose. It adds the information U-Boot needs to binary images while attaching a checksum for verification purposes.

While the use of image headers is not a technical requirement for a bootloader, such headers are very convenient both during development and in the field. Hence, their use by U-Boot.

To see the typical use of *mkimage*, type the command without any parameters:

```
$ mkimage
Usage: mkimage -l image
          -l =  => list image header information
       mkimage -A arch -O os -T type -C comp -a addr -e ep -n name
       -d data_file[:data_file...] image
          -A =  => set architecture to 'arch'
          -O =  => set operating system to 'os'
          -T =  => set image type to 'type'
          -C =  => set compression type 'comp'
          -a =  => set load address to 'addr' (hex)
          -e =  => set entry point to 'ep' (hex)
          -n =  => set image name to 'name'
          -d =  => use image data from 'datafile'
          -x =  => set XIP (execute in place)
```

For example here is how I create a U-Boot image of the 2.4.18 kernel I compiled for my control module:

```
$ cd ${PRJROOT}/images
$ mkimage -n '2.4.18 Control Module' \
> -A ppc -O linux -T kernel -C gzip -a 00000000 -e 00000000 \
> -d vmlinux-2.4.18.gz vmlinux-2.4.18.img
Image Name:   2.4.18 Control Module
Created:      Wed Feb  5 14:19:08 2003
Image Type:   PowerPC Linux Kernel Image (gzip compressed)
Data Size:    530790 Bytes = 518.35 kB = 0.51 MB
```

```
Load Address: 0x00000000
Entry Point:  0x00000000
```

The command takes quite a few flags, but their meaning is easily understood by looking at the usage message provided by *mkimage*. Note that the name of the image, provided in the *-n* option, cannot be more than 32 characters. Any excess characters will be ignored by *mkimage*. The rest of the command line tells *mkimage* that this is a gzipped PPC Linux kernel image that should be loaded at address 0x00000000 and started from that same address. The image being provided in input is *vmlinux-2.4.18.gz* and the U-Boot-formatted image will be output to *vmlinux-2.4.18.img*.

RAM disk images can be processed in a similar fashion:

```
$ mkimage -n 'RAM disk' \
> -A ppc -O linux -T ramdisk -C gzip \
> -d initrd.bin initrd.boot
Image Name:   RAM disk
Created:      Wed Feb  5 14:20:35 2003
Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
Data Size:    470488 Bytes = 459.46 kB = 0.45 MB
Load Address: 0x00000000
Entry Point:  0x00000000
```

In this case, the number of parameters is shorter, since we don't need to specify start and load addresses. Note that the image type has changed to `ramdisk`.

We can also create a `multi`-type image that combines both the kernel image and a RAM disk. In that case, the files included are listed sequentially using a colon separator:

```
$ mkimage -n '2.4.18 Ctrl and Initrd' \
> -A ppc -O linux -T multi -C gzip -a 00000000 -e 00000000 \
> -d vmlinux-2.4.18.gz:initrd.bin \
> vmlinux-2.4.18-initrd.img
Image Name:   2.4.18 Ctrl and Initrd
Created:      Wed Feb  5 14:23:29 2003
Image Type:   PowerPC Linux Multi-File Image (gzip compressed)
Data Size:    1001292 Bytes = 977.82 kB = 0.95 MB
Load Address: 0x00000000
Entry Point:  0x00000000
Contents:
   Image 0:   530790 Bytes =  518 kB = 0 MB
   Image 1:   470488 Bytes =  459 kB = 0 MB
```

Once you have prepared an image with *mkimage*, it is ready to be used by U-Boot and can be downloaded to the target. As we'll see below, U-Boot can receive binary images in a number of different ways. One way is to use images formatted in Motorola's S-Record format. If you intend to use this format, you need to further process the images generated by *mkimage* by converting them to the S-Record format. Here is an example conversion of the `multi`-type image generated above:

```
$ powerpc-linux-objcopy -I binary -O srec \
> vmlinux-2.4.18-initrd.img vmlinux-2.4.18-initrd.srec
```

9.5.6 Booting Using BOOTP/DHCP, TFTP, and NFS

If you have properly configured a server to provide the target with DHCP, TFTP, and NFS services, as I explained earlier, you can boot your target remotely. Back from U-Boot's prompt on my control module, here is how I boot my target remotely, for example:

```
=> bootp
BOOTP broadcast 1
DHCP client bound to address 192.168.172.10
ARP broadcast 1
TFTP from server 192.168.172.50; our IP address is 192.168.172.10

Filename '/home/karim/vmlinux-2.4.18.img'.
Load address: 0x100000
Loading: ####################################################  ...
done
Bytes transferred = 530854 (819a6 hex)
```

The *bootp* command issues a request that is answered by the DHCP server. Using the DHCP server's answer, U-Boot contacts the TFTP server and obtains the *vmlinux-2.4.18.img* image file, which it places at address 0x00100000 in RAM. You can verify the image's header information using the *iminfo* command:

```
=> imi 00100000

## Checking Image at 00100000 ...
   Image Name:   2.4.18 Control Module
   Created:      2003-02-05  19:19:08 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    530790 Bytes = 518.3 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
```

As you can see, the information printed out by *iminfo* on the target is very similar to that printed out on the host by *mkinfo*. The OK string reported for the checksum means that the image has been downloaded properly and that we can boot it:

```
=> bootm 00100000
## Booting image at 00100000 ...
   Image Name:   2.4.18 Control Module
   Created:      2003-02-05  19:19:08 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    530790 Bytes = 518.3 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
Linux version 2.4.18 (karim@Teotihuacan) (gcc version 2.95.3 20010315  ...
On node 0 totalpages: 4096
zone(0): 4096 pages.
zone(1): 0 pages.
zone(2): 0 pages.
Kernel command line: root=/dev/nfs rw nfsroot=  ...
Decrementer Frequency: 5000000
Calibrating delay loop... 79.66 BogoMIPS
 ...
VFS: Cannot open root device "" or 02:00
Please append a correct "root=" boot option
Kernel panic: VFS: Unable to mount root fs on 02:00
 <0>Rebooting in 180 seconds..
```

In this case, the kernel panics because it is unable to find any root filesystem. To solve this problem, we must use the environment variables to create a boot script for passing appropriate boot options to the kernel. The following commands create a new boot script, `bootpnfs`, and modify the special `bootcmd` script, as we did in Section 9.5.3, in order for the system to boot using BOOTP/DHCP, TFTP, and NFS:

```
=> setenv bootpnfs bootp\; setenv kernel_addr 00100000\; run nfscmd
=> printenv bootpnfs
bootpnfs=bootp; setenv kernel_addr 00100000; run nfscmd
=> setenv bootcmd run bootpnfs
=> printenv bootcmd
bootcmd=run bootpnfs
```

In this case, the `bootpnfs` script automatically executes the *bootp* instruction we used earlier in this section to obtain a kernel from the TFTP server. It then uses the `nfscmd` script we created in Section 9.5.3 to boot this kernel. The value of `kernel_addr` is changed so that the `nfscmd` script would use the kernel loaded using TFTP, not the one located at 40100000.

If you use the *boot* command now, U-Boot will boot entirely from the network. It will download the kernel through TFTP and mount its root filesystem on NFS. If you would like to save the environment variables we just set, use the *saveenv* command before rebooting the system, otherwise, you will have set the same variables again at the next reboot.

9.5.7 Downloading Binary Images to Flash

Booting from the network is fine for early development and testing. For production use, the target must have its kernel stored in flash. As we will see shortly, there a few ways to copy a kernel from the host to the target and store it to flash. Before you can copy any kernel image, however, you must first choose a flash region to store it and erase the flash region for the incoming kernel. In the case of my control module, I store the default kernel between 0x40100000 and 0x401FFFFF. Hence, from U-Boot's prompt, I erase this region:

```
=> erase 40100000 401FFFFF
Erase Flash from 0x40100000 to 0x401fffff
........ done
Erased 8 sectors
```

The simplest way to install a kernel in the target's flash is to first download it into RAM and then copy it to the flash. You can use the *tftpboot* command to download a kernel from the host to RAM:

```
=> tftpboot 00100000 /home/karim/vmlinux-2.4.18.img
ARP broadcast 1
TFTP from server 192.168.172.50; our IP address is 192.168.172.10
Filename '/home/karim/vmlinux-2.4.18.img'.
Load address: 0x100000
Loading: ####################################################  ...
done
Bytes transferred = 530854 (819a6 hex)
```

When *tftpboot* is run, it adds the `filesize` environment variable to the existing environment variables and sets it to the size of the file downloaded:

```
=> printenv filesize
filesize=819a6
```

You can use this environment variable in subsequent commands to avoid typing in the file size by hand. Don't forget to erase this environment variable before saving the environment variables, or it, too, will be saved.

In addition to *tftpboot*, you can use the *loadb* command to download images to the target:

```
=> loadb 00100000
## Ready for binary (kermit) download ...
```

At this point, U-Boot suspends and you must use the terminal emulator on the host to send the image file to the target. In this case, U-Boot expects to download the data according to the kermit binary protocol, and you must therefore use *kermit* to download a binary image to U-Boot. Once the transfer is done, U-Boot will output:

```
## Total Size      = 0x000819a6 = 530854 Bytes
## Start Addr      = 0x00100000
```

Here, too, U-Boot will set the `filesize` environment variable to the size of the file downloaded. As we did earlier, you may want to use the *iminfo* command to verify that the image has been properly downloaded.

Once the image is in RAM, you can copy it to flash:

```
=> cp.b 00100000 40100000 $(filesize)
Copy to Flash... done
=> imi 40100000

## Checking Image at 40100000 ...
   Image Name:   2.4.18 Control Module
   Created:      2003-02-05  19:19:08 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    530790 Bytes = 518.3 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
```

Alternatively, instead of downloading the image to RAM first using *tfptboot* or *loadb* and then writing it to flash, you can download the image directly to flash using *loads*. In this case, the host sends the image to the target in S-Record format. In comparison to the two previous methods, however, downloading an S-Record file is extremely slow. In most cases, it is preferable to use *tftpboot* or *loadb* instead.[6]

[6] The *loadb* command and, by default, the *tftpboot* command can't be used to download directly to flash. Though U-Boot can be configured at compile time to allow direct flash download using *tftpboot*, direct flash download using *loadb* is not supported.

To download S-Record files, you will need to use the *cu* terminal emulator to transfer them to the target, because the other terminal emulators don't interact properly with U-Boot when downloading this sort of file. When connected through *cu*, use the following commands:

```
=> loads 40100000
## Ready for S-Record download ...
~>vmlinux-2.4.18.srec
1 2 3 4 5 6 7 8 9 10 11 12 13 14  ...
 ...
 ...176 33177 33178 33179 33180 33181
[file transfer complete]
[connected]

## First Load Addr = 0x40100000
## Last  Load Addr = 0x401819A5
## Total Size      = 0x000819A6 = 530854 Bytes
## Start Addr      = 0x00000000
```

The ~> string shown here is actually part of the input you have to type. It is actually the *cu* command used to initiate a file download.

As before, you can verify the image once it's in memory:

```
=> imi 40100000

## Checking Image at 40100000 ...
   Image Name:   2.4.18 Control Module

   Created:      2003-02-05  19:19:08 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    530790 Bytes = 518.3 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
```

Every time you want to load a new image to flash, you have to start back at the *erase* command shown in the beginning of this section.

9.5.8 Booting Using a RAM Disk

The first step in booting from a RAM disk is to download the RAM disk from the host and install it on the target's flash. Many of the commands are the same as those shown and explained in previous sections. Here is how I do this for my control module:

```
=> tftpboot 00100000 /home/karim/initrd.boot
ARP broadcast 1
TFTP from server 192.168.172.50; our IP address is 192.168.172.10
TFTP from server 192.168.172.50; our IP address is 192.168.172.10
Filename '/home/karim/initrd.boot'.
Load address: 0x100000
Loading: ####################################################  ...
done
Bytes transferred = 470552 (72e18 hex)
=> imi 00100000

## Checking Image at 00100000 ...
   Image Name:   RAM disk
   Created:      2003-02-05  19:20:35 UTC
   Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
   Data Size:    470488 Bytes = 459.5 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
=> printenv filesize
filesize=72e18
=> imi 40200000

## Checking Image at 40200000 ...
   Bad Magic Number
=> erase 40200000 402FFFFF
Erase Flash from 0x40200000 to 0x402fffff
........ done
Erased 8 sectors
=> cp.b 00100000 40200000 $(filesize)
Copy to Flash... done
=> imi 40200000

## Checking Image at 40200000 ...
   Image Name:   RAM disk
   Created:      2003-02-05  19:20:35 UTC
   Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
   Data Size:    470488 Bytes = 459.5 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
```

Since I had already installed a kernel, I can boot the kernel available in flash with the RAM disk I just installed:

```
=> bootm 40100000 40200000
## Booting image at 40100000 ...
   Image Name:   2.4.18 Control Module
   Created:      2003-02-05  19:19:08 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    530790 Bytes = 518.3 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
## Loading RAMDisk Image at 40200000 ...
   Image Name:   RAM disk
   Created:      2003-02-05  19:20:35 UTC
   Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
   Data Size:    470488 Bytes = 459.5 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Loading Ramdisk to 00f2c000, end 00f9edd8 ... OK
Linux version 2.4.18 (karim@Teotihuacan) (gcc version 2.95.3 20010  ...
On node 0 totalpages: 4096
zone(0): 4096 pages.
zone(1): 0 pages.
zone(2): 0 pages.
Kernel command line:
Decrementer Frequency: 5000000
Calibrating delay loop... 79.66 BogoMIPS
 ...
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
 ...
RAMDISK: Compressed image found at block 0
 ...
VFS: Mounted root (ext2 filesystem).
 ...
```

Here, too, we can use environment variables to automate the booting process. Also, instead of using separate images for the kernel and the RAM disk, we could use a single image containing both, such as the one we created in Section 9.5.5. As I said earlier, U-Boot is a very flexible bootloader with many possible configurations. Though we cannot hope to cover all its possibilities here, feel free to experiment with U-Boot to obtain the setup that suits you best.

9.5.9 Booting from CompactFlash Devices

Before booting a kernel from a CF card using U-Boot, you need to properly partition and populate the CF card. Use *pdisk* or *fdisk* to partition the CF device, depending on your host. Since U-Boot does not recognize any disk filesystem, you will need to create a few small partitions to hold raw binary images and one large partition to hold your root filesystem will need to create a few small partitions to hold raw binary images and one large partition to hold your root filesystem, as I explained in Chapter 7.

For my control module, for example, I used a 32 MB CF card on which I created three partitions using *fdisk*: two 2 MB partitions to hold one stable kernel and one experimental kernel, and one 30 MB partition to hold my root filesystem. To copy the kernels to their respective partitions, I used the *dd* command:

```
# dd if=vmlinux-2.4.18.img of=/dev/sda1
1036+1 records in
1036+1 records out

# dd if=vmlinux-2.4.18-preempt.img of=/dev/sda2
1040+1 records in
1040+1 records out
```

I formatted */dev/sda3* using *mke2fs*, mounted it on */mnt/cf*, and copied the root filesystem to it using the techniques described in the previous chapter.

After I inserted the CF card in the PCMCIA port using a CF-to-PCMCIA adapter, here was the output of U-Boot at startup:

```
U-Boot 0.2.0 (Jan 27 2003 - 20:20:21)

CPU:   XPC860xxZPnnD3 at 80 MHz: 4 kB I-Cache 4 kB D-Cache FEC present
Board: TQM860LDB0A3-T80.201
DRAM:  16 MB
FLASH:  8 MB
In:    serial
Out:   serial
Err:   serial
Net:   SCC ETHERNET, FEC ETHERNET
PCMCIA: 3.3V card found: SunDisk SDP 5/3 0.6
            Fixed Disk Card
            IDE interface
            [silicon] [unique] [single] [sleep] [standby] [idle] [low power]
Bus 0: OK
  Device 0: Model: SanDisk SDCFB-32 Firm: vde 1.10 Ser#: 163194D0310
            Type: Removable Hard Disk
            Capacity: 30.6 MB = 0.0 GB (62720 x 512)
Hit any key to stop autoboot:  5
```

U-Boot identifies the storage device at startup. U-Boot provides a wide range of *ide* commands for manipulating IDE storage devices. You can see these commands by typing the *help* command:

```
=> help ide
ide reset - reset IDE controller
ide info  - show available IDE devices
ide device [dev] - show or set current device
ide part [dev] - print partition table of one or all IDE devices
ide read  addr blk# cnt
ide write addr blk# cnt - read/write `cnt' blocks starting at block `blk#'
    to/from memory address `addr'
```

We can further use U-Boot's command line to get more information regarding the device:

```
=> ide part

Partition Map for IDE device 0  --   Partition Type: DOS

Partition     Start Sector     Num Sectors     Type
    1                   62            4154      83
    2                 4216            4154      83
    3                 8370           54312      83
```

This command reads the partition table of the CF device and prints it out. In this case, the partition printed out by U-Boot fits the description provided earlier.

Loading a kernel image from one of the partitions on the CF device is done using the *diskboot* command. This command takes two arguments: the address where the kernel is to be loaded and a partition identifier. The latter is a concatenation of the device number and the partition number on that device separated by a colon. This is how I load the kernel image found on partition 1 of device 0 to address 0x00400000:

```
=> diskboot 00400000 0:1

Loading from IDE device 0, partition 1: Name: hda1
  Type: U-Boot
   Image Name:   2.4.18 Control Module
   Created:      2003-02-05  19:19:08 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    530790 Bytes = 518.3 kB
   Load Address: 00000000
   Entry Point:  00000000
=> imi 00400000

## Checking Image at 00400000 ...
   Image Name:   2.4.18 Control Module
   Created:      2003-02-05  19:19:08 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    530790 Bytes = 518.3 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
```

Once the kernel is loaded, you can use the *bootm* command to boot that kernel. This can also be automated by setting the `autostart` environment variable to `yes`. In that case, *diskboot* will automatically boot the kernel it loads:

```
=> setenv autostart yes
=> disk 00400000 0:1

Loading from IDE device 0, partition 1: Name: hda1
  Type: U-Boot
   Image Name:   2.4.18 Control Module
   Created:      2003-02-05  19:19:08 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    530790 Bytes = 518.3 kB
   Load Address: 00000000
   Entry Point:  00000000
Automatic boot of image at addr 0x00400000 ...
## Booting image at 00400000 ...
   Image Name:   2.4.18 Control Module
   Created:      2003-02-05  19:19:08 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    530790 Bytes = 518.3 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
Linux version 2.4.18 (karim@Teotihuacan) (gcc version 2.95.3  ...
On node 0 totalpages: 4096
 ...
```

As we did in Section 9.5.3 and Section 9.5.6, you can script the bootup from the CF device by setting the appropriate U-Boot environment variables. Also, if you wish, you can write to the disk directly from U-Boot using the *ide write* command. Have a look at the *help* output and the documentation for more information regarding the use of U-Boot's IDE capabilities.

9.5.10 Updating U-Boot

U-Boot is like any other open source project; it continues to evolve over time as contributions are made and bug fixes are integrated to the codebase. Consequently, you may feel the need to update your target's firmware version. Fortunately, because U-Boot runs from RAM, it can be used to update itself. Essentially, we have to download a new version to the target, erase the old firmware version, and copy the new version over it.

There are obvious dangers to this operation, because a mistake or a power failure will render the target unbootable. Hence, utmost caution must be used when carrying out the following steps. Make sure you have a copy of the original bootloader you are about to replace so that you can at least fall back to a known working version. Also, seriously consider avoiding the replacement of your firmware if you have no hardware means to reprogram the target's flash if the upgrade fails. If you do not have access to a BDM debugger or a flash programmer, for example, there is a great risk that you will be left with a broken system if one of the following steps fails. Dealing with buggy software is one thing; ending up with unusable hardware is another.

Once you have taken the necessary precautions, download the U-Boot image into RAM using TFTP:

```
=> tftp 00100000 /home/karim/u-boot.bin-0.2.0
ARP broadcast 1
TFTP from server 192.168.172.50; our IP address is 192.168.172.10
Filename '/home/karim/u-boot.bin-0.2.0'.
Load address: 0x100000
Loading: #################################
done
Bytes transferred = 166532 (28a84 hex)
```

If you do not have a TFTP server set up, you can also use the terminal emulator to send the image:

```
=> loadb 00100000
## Ready for binary (kermit) download ...

## Start Addr      = 0x00100000
```

Unlike other images we have downloaded to the target, you cannot use the *imi* command to check the image, since the U-Boot image downloaded was not packaged on the host using the *mkimage* command. You can, however, use *crc32* before and after copying the image to flash to verify proper copying.

Now, unprotect the flash region where U-Boot is located so you can erase it (in this case, U-Boot occupies the flash region from 0x40000000 to 0x4003FFFF):

```
=> protect off 40000000 4003FFFF
Un-Protected 5 sectors
```

Erase the previous bootloader image:

```
=> erase 40000000 4003FFFF
Erase Flash from 0x40000000 to 0x4003ffff
... done
Erased 5 sectors
```

Copy the new bootloader to its final destination:

```
=> cp.b 00100000 40000000 $(filesize)
Copy to Flash... done
```

Erase the `filesize` environment variable set during the download:

```
=> setenv filesize
```

Save the environment variables:

```
=> saveenv
Saving Enviroment to Flash...
Un-Protected 1 sectors
Erasing Flash...
. done
Erased 1 sectors
Writing to Flash... done
Protected 1 sectors
```

At this stage, the new bootloader image has been installed and is ready to be used. Until you issue the *reset* command, however, you can still use the old U-Boot currently running to fix any problems that may have occurred during the update. Once you are satisfied that every step of the update has gone through cleanly, you can go ahead and restart the system:

```
=> reset


U-Boot 0.2.0 (Jan 27 2003 - 20:20:21)

CPU:   XPC860xxZPnnD3 at 80 MHz: 4 kB I-Cache 4 kB D-Cache FEC present
Board: TQM860LDB0A3-T80.201
DRAM:  16 MB
FLASH:  8 MB
In:    serial
Out:   serial
Err:   serial
Net:   SCC ETHERNET, FEC ETHERNET
PCMCIA:   No Card found
Hit any key to stop autoboot:  5
```

If you can see the U-Boot boot message again, U-Boot has been successfully updated. Otherwise, there has been a problem with the replacement of the firmware and you need to reprogram the flash device using the appropriate hardware tools.

Sometimes, kernel images that used to boot with the older bootloader version will fail to boot with newer versions. When upgrading from a PPCBoot version prior to 1.0.5 to Version 1.0.5 or later, for example, kernels prior to 2.4.5-pre5 may fail to boot. In that case, the reason behind the problem is in the way U-Boot passes the clock speed to the kernel. Prior to kernel 2.4.5-pre5, kernels expected to receive the speed in MHz, while later kernels expect to receive the speed in Hz. To this end, PPCBoot 1.0.5 passes the clock speed to kernels in Hz. Kernels that expect to receive it in MHz, however, fail to boot. In practice, the boot process will start as it would normally, but the system will freeze right after U-Boot finishes uncompressing the images for startup. You will, therefore, see something like:

```
   ...
   Entry Point:  00000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
```

Nothing will be output after that, and there will be no responses to any input from the terminal. To solve the problem, you need to tell the newer version of U-Boot to keep passing the clock speed in MHz to the older kernels. This is done by setting the `clocks_in_mhz` environment variable to 1:

```
=> setenv clocks_in_mhz 1
=> saveenv
```

Though this sort of problem does not occur for every upgrade, changes in the kernel sometimes require significant changes to the tools that interface with it. Given that such problems are difficult to figure out if you are not involved in the actual development of each project, I strongly encourage you to keep in touch with the rest of the U-Boot users by subscribing to the U-Boot users mailing list from the project's web site and to read announcements of new versions carefully.

5 Comments etutorials Login

Sort by Best Share Favorite

Join the discussion…

Arun • 8 months ago
Compact and clear explanation .. Thanks !!
• Reply • Share ›

guest • 10 months ago
very very...useful....
• Reply • Share ›

Guest • a year ago
Great examples. Really helpful. Thanks for posting!
• Reply • Share ›

QD • 2 years ago
Thanks! So helpful!
• Reply • Share ›

Khajapasha • 2 years ago
Good Notes And It Is Very Help
• Reply • Share ›

ALSO ON **ETUTORIALS** WHAT'S THIS?

NAI Extension :: Appendix A: Mobile IP Extensions :: Mobile vpn. delivering advanced …

3.6 Internet Sharing (Software Base Stations) :: Chapter 3. Making the Wi-Fi Connection :: Mac …

1 comment • a year ago

mobile applications developers — This is a great inspiring article.You put really very helpful informal am pretty much pleased with your good work.tion. …

1 comment • a year ago

Benjamin181 — This is a nice post in an interesting line of content.Thanks for sharing this article.

Subscribe Add Disqus to your site