

U-Boot Reference Manual

90000852_C

© Digi International Inc. 2007. All Rights Reserved.

The Digi logo is a registered trademark of Digi International, Inc.

All other trademarks mentioned in this document are the property of their respective owners.

Information in this document is subject to change without notice and does not represent a commitment on the part of Digi International. Digi provides this document "as is," without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of fitness or merchantability for a particular purpose. Digi may make improvements and/or changes in this manual or in the product(s) and/or the program(s) described in this manual at any time.

This product could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes may be incorporated in new editions of the publication.

Digi International Inc.

11001 Bren Road East

Minnetonka, MN 55343 (USA)

The second secon

www.digiembedded.com

Contents

	Conventions used in this manual	5
2	2 Acronyms and Abbreviations	6
3	3 Introduction	7
	3.1 What is a boot loader?	
	3.2 What is U-Boot?	7
	3.3 U-Boot Features	
	3.3.1 Customizable footprint	
	3.3.3 Variables	8
	3.3.4 Kernel images downloadable via Ethernet and USB	
	3.3.5 Numbers assumed in hexadecimal format	
4	`	
-	4.1 Overview	
	4.1 Overview	
	4.2.1 Information commands	
	4.2.2 Network commands	
	4.2.3 USB commands 4.2.4 Memory commands	
	4.2.5 Serial port commands	
	4.2.6 I2C commands	
	4.2.7 Environment variable commands	
5	5 Environment variables	14
	5.1 Overview	
	5.2 Simple and recursive variables	14
	5.2 Simple and recursive variables5.3 Scripts	14 15
	5.2 Simple and recursive variables	14 15 16
	 5.2 Simple and recursive variables 5.3 Scripts 5.4 System variables 5.4.1 Common system variables 5.4.2 Dynamic variables 	
	 5.2 Simple and recursive variables 5.3 Scripts 5.4 System variables 5.4.1 Common system variables 5.4.2 Dynamic variables 5.4.3 Digi custom variables 	
	 5.2 Simple and recursive variables 5.3 Scripts 5.4 System variables 5.4.1 Common system variables 5.4.2 Dynamic variables 5.4.3 Digi custom variables 5.4.4 Protected variables 	
6	 5.2 Simple and recursive variables. 5.3 Scripts. 5.4 System variables. 5.4.1 Common system variables. 5.4.2 Dynamic variables. 5.4.3 Digi custom variables. 5.4.4 Protected variables. 6 Boot commands. 	
6	 5.2 Simple and recursive variables	
6	 5.2 Simple and recursive variables. 5.3 Scripts. 5.4 System variables. 5.4.1 Common system variables. 5.4.2 Dynamic variables. 5.4.3 Digi custom variables. 5.4.4 Protected variables. 6 Boot commands. 	
6	 5.2 Simple and recursive variables	
6	 5.2 Simple and recursive variables	
6	 5.2 Simple and recursive variables	
6	 5.2 Simple and recursive variables	
6	 5.2 Simple and recursive variables. 5.3 Scripts. 5.4 System variables. 5.4.1 Common system variables. 5.4.2 Dynamic variables. 5.4.3 Digi custom variables. 5.4.4 Protected variables. 5.4.4 Protected variables. 6.1 Overview. 6.2 Reading images into RAM. 6.2.1 From Ethernet. 6.2.2 From USB. 6.3 Booting images in RAM. 6.4 Direct booting. 	
6	 5.2 Simple and recursive variables	
6	 5.2 Simple and recursive variables	
6	 5.2 Simple and recursive variables	
6	 5.2 Simple and recursive variables	

8 Firmware update commands	
8.1 Overview 8.2 Updating flash with images in RAM	
8.3 Direct updating 8.3.1 Update limits	
9 U-Boot development	29
Index	

1 Conventions used in this manual

This list shows the typographical conventions used in this guide:

Style	In text, to introduce new terms	
Style	In text, for command and variable names.	
Style	In examples, to show the text that should be typed literally by the user.	
Style	In text and syntax discussions, to display command variables.	
#	A prompt that indicates the action is performed in the target device.	
\$	A prompt that indicates the action is performed in the host computer.	
<field></field>	A mandatory field that must be replaced with a value.	
[field]	An optional field.	
[a b c]	A field that can take one of several values.	

This manual also uses these frames and symbols:



This is a warning. It helps solve or to avoid common mistakes or problems



This is a hint. It contains useful information about a topic

		6		ñ	
	æ			П	
- 1		-	-	,	
1	-	7	-		
			-		

\$ This is a host computer session \$ Bold text indicates what must be input

mi	5
44	

#

#

This is a target session Bold text indicates what must be input

2 Acronyms and Abbreviations

BIOS	Basic Input Output System
CPU	Central Processing Unit
FAT	File Allocation Table
I2C	Inter-Integrated Circuit
MBR	Master Boot Record
NVRAM	Non Volatile RAM
OS	Operating System
PC	Personal Computer
RAM	Random Access Memory
TFTP	Trivial File Transfer Protocol
USB	Universal Serial Bus

3 Introduction

3.1 What is a boot loader?

Microprocessors can execute only code that exists in memory (either ROM or RAM), while operating systems normally reside in large-capacity devices such as hard disks, CD-ROMs, USB disks, network servers, and other permanent storage media.

When the processor is powered on, the memory does not hold an operating system, so special software is needed to bring the OS into memory from the media on which it resides. This software is normally a small piece of code called the *boot loader*. On a desktop PC, the boot loader resides on the master boot record (MBR) of the hard drive and is executed after the PC's *basic input output system* (BIOS) performs system initialization tasks.

In an embedded system, the boot loader's role is more complicated because these systems rarely have a BIOS to perform initial system configuration. Although the low-level initialization of the microprocessor, memory controllers, and other board-specific hardware varies from board to board and CPU to CPU, it must be performed before an OS can execute.

At a minimum, a boot loader for an embedded system performs these functions:

- Initializing the hardware, especially the memory controller
- Providing boot parameters for the OS
- Starting the OS

Most boot loaders provide features that simplify developing and updating firmware; for example:

- Reading and writing arbitrary memory locations
- Uploading new binary images to the board's RAM from mass storage devices
- Copying binary images from RAM into flash

3.2 What is U-Boot?

U-Boot is an open-source, cross-platform boot loader that provides out-of-box support for hundreds of embedded boards and many CPUs, including PowerPC, ARM, XScale, MIPS, Coldfire, NIOS, Microblaze, and x86.

For more information about the U-Boot project see <u>http://sourceforge.net/projects/u-boot/</u> and <u>http://www.denx.de/wiki/DULG/Manual</u>.

3.3 U-Boot Features

3.3.1 Customizable footprint

U-Boot is highly customizable to provide both a rich feature set and a small binary footprint.

3.3.2 Monitor

U-Boot has a command shell (also called a monitor) for working with U-Boot commands to create a customized boot process.

3.3.3 Variables

U-Boot uses environment variables that can be read or written to and from non-volatile media. Use these variables to create scripts of commands (executed one after the other) and to configure the boot process.

3.3.4 Kernel images downloadable via Ethernet and USB

Because U-Boot can download a kernel image using either Ethernet or USB, no flash programming is needed to test a new kernel. This prevents the deterioration of flash caused by repeated flash erases and writes.

3.3.5 Numbers assumed in hexadecimal format

Numbers used by U-Boot are always considered to be in hexadecimal format. For example, U-Boot understands number 30100000 as 0x30100000.

3.4 The boot process

After power-up or reset, the processor loads the U-Boot boot loader in several steps.

- The processor does these steps:
 - Executes a primary bootstrap that configures the interrupt and exception vectors, clocks, and SDRAM
 - Decompresses the U-Boot code from flash to RAM
 - Passes execution control to the U-Boot
- U-Boot does these steps:
 - Configures the Ethernet MAC address, flash, and, serial console
 - Loads the settings stored as environment variables in non-volatile memory
 - After a few seconds (a programmable length of time), automatically boots the pre-installed kernel

To stop the automatic booting (*autoboot*) of the pre-installed kernel, send a character to the serial port by pressing a key from the serial console connected to the target. If U-Boot is stopped, it displays a command line console (also called *monitor*).

```
U-Boot 1.1.4 (Apr 20 2007 - 21:47:39) DUB-RevA
      for Digi ConnectCore Wi-9C on Development Board
b.
      DRAM:
             64 MB
      NAND:
             128 MiB
      In:
             serial
      Out:
             serial
             serial
      Err:
             NS9360 @ 154.828800MHz
      CPU:
      Strap: 0x03
      SPI ID:2007/02/21, V1_4, CC9C/CCW9C, SDRAM 64MByte, CL2, 7.8us, LE
      FPGA: wifi.ncd, 2007/01/25, 17:49:41, V2.01
      Hit any key to stop autoboot: 0
```

4 U-Boot commands

4.1 Overview

U-Boot has a set of built-in commands for booting the system, managing memory, and updating an embedded system's firmware. Custom built-in commands can be created by modifying U-Boot source code.

4.2 Built-in commands

For a complete list and brief descriptions of the built-in commands, at the U-Boot monitor prompt, enter either of these commands:

- help
- ?

#

help

A list of commands and help text like this is displayed:



# 1	nerp	,	
?		-	alias for 'help'
			run script from memory
base	2	-	print or set address offset
			print Board Info structure
boot		-	boot default, i.e., run 'bootcmd'
			boot default, i.e., run 'bootcmd'
boot	elf	-	Boot from an ELF image in memory
boot	m	-	boot application image from memory
boot	p	-	boot image via network using BootP/TFTP protocol
boot	vx	-	Boot vxWorks from an ELF image
cmp		-	memory compare
coni	nfo	-	print console devices and information
ср		-	memory copy
crc3	2	-	checksum calculation get/set/reset date & time
date	<u>:</u>	-	get/set/reset date & time
dboo	t	-	Digi ConnectCore modules boot commands
dcac	he	-	enable or disable data cache
dhcp)	-	invoke DHCP client to obtain IP/boot params
echo)	-	echo args to console
envr	reset	2-	Sets environment variables to default setting
fati	nfo	-	print information about filesystem
fatl	oad	-	load binary file from a dos filesystem
fatl	S	-	list files in a directory (default /)
flpa	rt	-	displays or modifies the partition table.
			start application at address 'addr'
help)	-	print online help
icac	he	-	enable or disable instruction cache
icrc	:32	-	checksum calculation
iloo	p	-	infinite loop on address range
imd		-	i2c memory display
imin	fo	-	print header information for application image
imm		-	i2c memory modify (auto-incrementing)
imw		-	memory write (fill)
inm		-	memory modify (constant address)
intn	ivran	n–	displays or modifies NVRAM contents like IP or partition table
ipro	be	-	probe to discover valid I2C chip addresses
ites	t	-	return true/false on integer compare
load	lb	-	load binary file over serial line (kermit mode)
load	ls	-	load S-Record file over serial line
load	ly	-	load binary file over serial line (ymodem mode)
loop)	-	infinite loop on address range
md		-	memory display
mm		-	memory modify (auto-incrementing)

```
mtest
       - simple RAM test
       - memory write (fill)
mw
       - NAND sub-system
nand
      - boot from NAND device
nboot
       - boot image via network using NFS protocol
nfs
nm
       - memory modify (constant address)
ping
       - send ICMP ECHO_REQUEST to network host
printenv- print environment variables
printenv_dynamic- Prints all dynamic variables
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
       - delay execution for some time
sleep
       - synchronize RTC via network
sntp
tftpboot- boot image via network using TFTP protocol
update - Digi ConnectCore modules update commands
       - USB sub-system
usb
usbboot - boot from USB device
version - print monitor version
```

The available commands vary according to the capabilities of the hardware platform.

For more information about a command, enter:

help command name

For example:

#



help run run var [...] - run the commands in the environment variable(s) 'var'



As the first characters of a command are entered, U-Boot searches its list of built-in commands until it finds a match. For example, entering save or sav or even sa, causes U-Boot to execute the saveenv command.

U-Boot needs enough characters to be entered to determine the command to execute. For example, if loa is entered, U-Boot cannot tell whether to execute loadb, loads or loady, and an Unknown command message is displayed.

4.2.1 Information commands

Commands that information about the development board, devices, memory, etc., include:

Command	Description
bdinfo	Prints board info structure.
coninfo	Prints console devices and information.
date [MMDDhhmm[[CC]YY][.ss]]	Gets, sets, or resets system date/time.
fatinfo <i><interface< i="">> <<i>dev</i>[:<i>part</i>]></interface<></i>	Prints information about the file system from <i>dev</i> on <i>interface</i> .
iminfo addr [addr]	Prints header information for the application image starting at the <i>addr</i> address in memory, including verification of the image contents (magic number, header, and payload checksums). Works only for Linux kernel images.
nand bad	Shows NAND bad blocks.
nand info	Shows available NAND devices.
version	Displays U-Boot version and timestamp.

4.2.2 Network commands

Network-related commands include:

Command	Description
bootp [<i>loadAddress</i>] [<i>bootFilename</i>]	Boots the image over the network using the BootP/TFTP protocol. If no argument is given, bootp takes the values from the <i>loadaddr</i> and <i>bootfile</i> environment variables.
dhcp	Requests an IP address from a DHCP server, set in the serverip system variable. If the autoload variable is set to yes, also transfers the file to which the bootfile environment variable points to the loadAddress RAM memory address by TFTP.
nfs [<i>loadAddress</i>] [<i>host ip addr.bootfilename</i>]	Using NFS, transfers image <i>bootfilename</i> into the RAM address <i>loadAddress</i> .
ping < <i>pingAddress</i> >	Pings the IP address passed as parameter. If the other end responds, this message is displayed: host < <i>pingAddress</i> > is alive.
rarpboot [<i>loadAddress</i>] [<i>bootfilename</i>]	Using RARP/TFTP, transfers image into the RAM address <i>loadAddress</i> .
sntp	Gets the date and time from the NTP server to which the <i>ntpserverip</i> environment variable points
tftpboot [loadAddress] [bootfilename]	Using FTP, transfers image <i>bootfilename</i> into the RAM address <i>loadAddress</i> .



If the *autostart* variable is set to 'yes', all commands (except *ping*) boot the transferred image by calling the *bootm* command. *bootm* does not work for WinCE images. If working with a WinCE image file, either set the *autostart* variable to 'no' or delete it before executing these network commands.

4.2.3 USB commands

To access the USB subsystem, use the **usb** command, followed by its operations:

Command	Description
usb reset	Resets (rescans) USB controller.
usb stop [f]	Stops USB [f]=force stop.
usb tree	Shows USB device tree.
usb info [<i>dev</i>]	Shows available USB devices.
usb storage	Shows details of USB storage devices.
usb dev [<i>dev</i>]	Shows or sets current USB storage device.
usb part [<i>dev</i>]	Prints the partition table of one (<i>dev</i>) or all USB storage devices.
usb read addr blk# cnt	Reads <i>cnt</i> blocks starting at block <i>blk#</i> to RAM address <i>addr.</i>
fatload usb < <i>dev</i> [: <i>part</i>]> < <i>addr</i> > < <i>filename</i> >	Reads <i>filename</i> image from partition <i>part</i> of USB device <i>dev</i> into the RAM memory address <i>addr</i> . If <i>part</i> is not specified, partition 1 is assumed.
usbboot	Boots from USB device.

4.2.4 Memory commands

These commands manage RAM memory:

Command	Description
cmp[.b, .w, .l] addr1 addr2 count	Compares memory contents from address <i>addr</i> to <i>addr2</i> for as many <i>count</i> bytes, words, or long words.
cp[.b, .w, .l] source target count	Copies memory contents from address <i>source</i> to <i>target</i> for as many <i>count</i> bytes, words, or long words.
go <i>addr</i> [<i>arg</i>]	Starts the application at address <i>addr</i> passing <i>arg</i> as arguments.
md[.b, .w, .l] address [# of objects]	Displays memory contents at address <i>addr</i> for as many <i>[#of objects]</i> bytes, words, or long words.
mm[.b, .w, .l] address	Modifies locations of memory, beginning at <i>address</i> , which gets auto-incremented.
mw[.b, .w, .l] address value [count]	Writes <i>value</i> into <i>address</i> for as many <i>count</i> bytes, words, or long words.
nm[.b, .w, .l] address	Modifies a fixed location of memory.
nand read addr off size	Copies memory contents from flash address <i>off</i> to RAM address <i>addr</i> for as many <i>size</i> bytes (only for NAND flash memories).
nand write addr off size	Copies memory contents from RAM address <i>addr</i> to flash address <i>off</i> for as many <i>size</i> bytes (NAND flash memories only).
nand erase [<i>off size</i>]	Erases <i>size</i> bytes from address <i>off</i> . Erases entire device if no parameters are specified (NAND flash memories only). U-Boot skips bad blocks and shows their addresses.
nand dump[.oob] off	Dumps NAND page at address <i>off</i> with optional out-of-band data (only for NAND flash memories).

Boots image from NAND device *dev* at offset *off* (transferring it first to RAM *address*).

4.2.5 Serial port commands

Use these commands to work with the serial line:

Command	Description
loadb [<i>off</i>] [<i>baud</i>]	Loads binary file over serial line with offset <i>off</i> and baud rate <i>baud</i> (Kermit mode).
loads [<i>off</i>]	Loads S-Record file over the serial line with offset off.
loady [<i>off</i>] [<i>baud</i>]	Loads binary file over the serial line with offset <i>off</i> and baud rate <i>baud</i> (Ymodem mode).

4.2.6 I2C commands

These commands interface with the I2C interface:

Command	Description
iloop chip <i>address</i> [.0, .1, .2] [# of objects]	Loops, reading a set of I2C addresses.
<pre>imd chip address[.0, .1, .2] [# of objects]</pre>	Displays I2C memory.
imm chip address[.0, .1, .2]	Modifies I2C memory with an auto-incremented address.
imw address[.0, .1, .2] value [count]	Fills an I2C memory range with value.
inm chip address[.0, .1, .2]	Modifies memory, reads and keeps an address.
iprobe	Discovers valid I2C chip addresses.
itest [.b, .w, .l, .s] [*] <i>value1<op></op></i> [*] <i>value2</i>	Returns TRUE/FALSE on integer compare.

4.2.7 Environment variable commands

To read, write, and save environment variables, use these commands:

Command	Description
printenv [name]	If no variable is given as argument, prints all U-Boot environment variables.
	If a list of variable names is passed, prints only those variables.
printenv_dynamic	Prints all dynamic variables.
envreset	Overwrites all current variables values to factory default values.
	Does not reset the <i>wlanaddr or ethaddr</i> variables or any other persistent settings stored in NVRAM (see topic 7.1).
saveenv	Writes the current variable values to non-volatile memory (NVRAM).
setenv name [value]	If no value is given, the variable is deleted. If the variable is dynamic, it is reset to the default value.
	If a value is given, sets variable <i>name</i> to value <i>value</i> .

5 Environment variables

5.1 Overview

U-Boot uses environment variables to tailor its operation. The environment variables configure settings such as the baud rate of the serial connection, the seconds to wait before auto boot, the default boot command, and so on.

These variables must be stored in either non-volatile memory (NVRAM) such as an EEPROM or a protected flash partition.

The factory default variables and their values also are stored in the U-Boot binary image itself. This allows recovering the variables and their values at any time with the **envreset** command.

Environment variables are stored as strings (case sensitive). Custom variables can be created as long as there is enough space in the NVRAM.

5.2 Simple and recursive variables

Simple variables have a name and a value (given as a string):



setenv myNumber 123456
printenv myNumber
myNumber=123456

To expand simple variables, enclose them in braces and prefix a dollar sign:



```
# setenv myNumber 123456
# setenv var This is my number: ${myNumber}
# printenv var
var=This is my number: 123456
```

Recursive variables (or scripts) contain one or more variables within their own value. The inner variables are not expanded in the new variable. Instead, they are expanded when the recursive variable is run as a command, as shown here:



```
# setenv dumpaddr md.b \${addr} \${bytes}
# printenv dumpaddr
dumpaddr=md.b ${addr} ${bytes}
# setenv addr 2C000
# setenv bytes 5
# run dumpaddr
0002c000: 00 00 00 00 .....
```

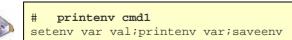
To prevent variables from being expanded into other variables' values, use the back slash \ before \$.

5.3 Scripts

In U-Boot, a script is made up of variables that contain a set of commands that are executed one after another.

Consider this variable:

run cmd1



Running this script with **run cmd1** creates the **var** variable with value **val**, prints the value **val** to the console, and saves the variables to either the EEPROM or flash partition dedicated to variables.



#

```
var=val
Saving Environment to Flash...
Un-Protected 1 sectors
Erasing Flash...
. done
Erased 1 sectors
Writing to Flash... done
Protected 1 sectors
```

Separate the commands in a script with semicolons (;). As with recursive variables, semicolons must be preceded by a back-slash sign to prevent them from being interpreted as the termination of the first command itself.

To save cmd1, enter:



#

#

setenv cmd1 setenv var val\;printenv var\;saveenv

For running commands stored in variables, use the **run** command and its variables separated by spaces:



S	etenv	cmd1	setenv	var	val
ຣ	etenv	cmd2	printer	ıv va	ar
S	etenv	cmd3	saveen	7	
r	un cmo	11 cmd	12 cmd3		

5.4 System variables

U-Boot uses several built-in system variables:

5.4.1 Common system variables

Variable	Description
autoload	If set to no (or any string beginning with n), the rarpboot , bootp , or dhcp command performs a configuration lookup from the BOOTP / DHCP server but does not try to load any image using TFTP.
autostart	If set to yes , an image loaded using the rarpboot , bootp , dhcp or tftpboot commands is automatically started (by internally calling the bootm command).
baudrate	The baud rate of the serial connection.
bootcmd	Defines a command string that is automatically executed when the initial countdown is not interrupted.
	Executed only when the bootdelay variable is also defined.
bootdelay	Seconds to wait before running the automatic boot process in bootcmd .
bootfile	Name of the default image to load with TFTP.
dhcp	If set to on , enables the DHCP client to obtain a dynamic IP for the Ethernet interface.
dhcp_wlan	For modules with a WLAN interface, if set to on , enables the DHCP client to obtain a dynamic IP for the WLAN interface.
dnsip	IP address of the primary DNS server,
dnsip2	IP address of the secondary DNS server.
fileaddr	The RAM address where the last file transferred by TFTP was placed.
filesize	The size of the last file transferred by TFTP or USB.
gatewayip	IP address used as network gateway.
ipaddr	IP address of the target's Ethernet interface.
ipaddr_wlan	IP address of the target's WLAN interface (for modules that have it).
netmask	Subnet mask of Ethernet interface.
netmask_wlan	Subnet mask of WLAN interface (for modules that have it).
ntpserverip	NTP server IP address (for getting the date/time).
stdin	Standard input system.
stdout	Standard output system.
stderr	Standard error output system.
serverip	IP address of the host PC (for remote connections like TFTP transfers)
verify	If set to n or no , disables the checksum calculation over the complete image in the bootm command to trade speed for safety in the boot process. Note that the header checksum is still verified.

5.4.2 Dynamic variables

Depending on the module, the partitioning information, and so on, U-Boot generates some variables "on the fly" if they do not already exist in U-Boot.

These variables can be overwritten with **setenv** thus becoming standard U-Boot variables. Dynamic variables which are not set with **setenv** also exist (they are automatically created), but they cannot be printed with **printenv**.

Some of these variables are OS-specific for different OS implementations (Linux, Windows CE, NET+OS). They provide special functionality for the OS running in the platform.



For more information, see the boot loader development chapter of the development kit's documentation.

5.4.3 Digi custom variables

The development board in the kit may have two user buttons. If it does, U-Boot can detect which one is pressed when it starts.

Pressing either key when the boot loader is starting, executes the **key1** or **key2** variable before the **bootcmd**. This allows for different boot scripts, depending on the key pressed during boot, for booting two different kernels, such as a dual Linux/Windows CE or two versions of the same OS.

If the key1 and key2 variables do not exist, the normal bootcmd is executed.

When the two keys are pressed during boot, both are detected as pressed, and both scripts are launched. The script in variable **key1** is always executed before the one in variable **key2**.



Detection of user keys can be disabled for customized hardware where these keys do not exist. This requires reconfiguring and recompiling U-Boot. See chapter 9 for information about U-Boot development.

5.4.4 Protected variables

Several variables are of great relevance for the system and are stored in a protected section of NVRAM.

Some of these protected variables are, for example, the serial number of the module and the MAC addresses of the network interfaces, which are programmed during production and normally should not be changed.

6 Boot commands

6.1 Overview

U-Boot runs code placed in RAM, although it also can read from other media. The boot process normally takes place in two steps:

- Reading the OS image from media (Ethernet, flash, USB) into RAM
- Jumping to the first instruction of the image in RAM

6.2 Reading images into RAM

6.2.1 From Ethernet

The most common way to boot an image during development is by transferring it using TFTP over the Ethernet interface. To do this, use the **tftpboot** command, passing:

- The address of RAM in which to place the image (*loadAddress*)
- The image file name (*bootfilename*)



tftpboot <loadAddress> <bootfilename>

The TFTP transfer occurs between the **serverip** address (host) and the **ipaddr** address (target). The host must be running a TFTP server and have **bootfilename** archive placed in the TFTP-exposed directory.

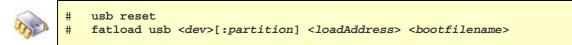
For Linux kernel images, if the **autostart** variable is set to **yes**, this command directly boots the kernel after downloading it.

6.2.2 From USB

#

Another way to boot an image is by reading it from a USB flash storage device. The USB disk must be formatted in FAT file system.

To read an image from a USB flash disk, enter:



This command reads file **bootfilename** from device **dev**, partition **partition** of the USB flash disk into the RAM address **loadAddress**. Device and **partition** are given as a number (0, 1, 2...).

If no partition is specified, partition 1 is assumed.

6.2.3 From flash

For standalone booting, the device can read the image from flash, avoiding dependency on any external hardware.

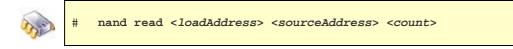
In targets with NOR flash memories, do this with memory commands:



cp.[b/w/l] <sourceAddress> <loadAddress> <count>

This command copies *count* bytes, words, or long words (depending on the suffix used -: b, w, I - from *sourceAddress* into *loadAddress*.

In targets with NAND flash memories, the special NAND commands must be used:



This command copies count bytes from sourceAddress into loadAddress.

6.3 Booting images in RAM

After the image is transferred to RAM, it can be booted it in either of two ways, depending on the OS:

For Windows CE images:



go <loadAddress>

For Linux images:



bootm <loadAddress>

where *loadAddress* (in both cases) is the address in RAM at which the image resides.



Windows CE images must be compiled with the information about the address in RAM from which they will be booted. For example, if a WinCE kernel is compiled with a boot address of 0x2C0000, it can be transferred to a different address, but the system can boot only from the compiled-in address.

6.4 Direct booting

To simplify the boot process, Digi's U-Boot version includes the **dboot** built-in command, which reads the OS image from the media and runs it from RAM in a single step.

The syntax for the **dboot** command is:



dboot <os> <media>

where

#

- os is linux, wce, eboot, netos or any partition name.
- media is flash, tftp or usb.



If the **dhcp** variable is set to **yes** or **on**, the command first gets an IP address from the DHCP server to which the **serverip** variable points.

For example, to boot linux from flash, execute:



#

dboot linux flash

To boot a partition from flash, a valid partition name must be provided. To print the partitions table, use the **flpart** command (for more information about this command see topic 7.2). Then execute the command with the selected partition name:

#	flpart							
Nr	Name	Start		Size		Туре	FS	Flags
0	U-Boot		0	768	KiB	U-Boot	None	fixed
1	NVRAM	768 K	ib	512	KiB	NVRAM	None	fixed
2	Kernel	1280 K	ίв	3	MiB	Linux-Kernel	None	
3	RootFS-JFFS2	4352 K	iв	16	MiB	Filesystem	JFFS2	rootfs
4	User-JFFS2	20736 K	iв	12	MiB	Filesystem	JFFS2	
5	Kernel-2	33024 K	ib	2	MiB	Linux-Kernel	None	
#	dboot Kernel-2 f	lash						

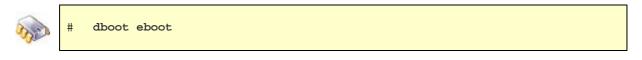


Partition names are case sensitive, so *kernel-2* is a partition name different from *Kernel-2*.

6.4.1 Direct booting with Microsoft Platform Builder / Visual Studio

The **eboot** program is used to interconnect the target to the Windows CE development system (Platform Builder or Visual Studio). **Eboot** sends **BOOTME** messages to the development system program. These two programs talk to each other to transfer and debug the Windows CE kernel.

This is the command for booting with Platform Builder / Visual Studio:

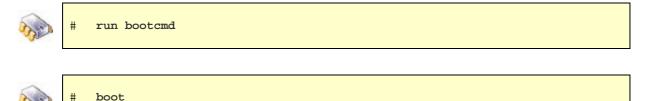


This command reads the **eboot** image from flash into a specific RAM address and jumps to this image for execution.

6.5 Automatic booting

If U-Boot is not interrupted after the delay established in **bootdelay**, the automatic boot process takes place. Automatic booting consists of running what is specified in the **bootcmd** environment variable.

In other words, automatic booting has the same effect as doing either of the next two examples:



If, for example, to automatically boot a WinCE image from TFTP server, set **bootcmd** like this:



#

setenv bootcmd dboot wce tftp

Or, to automatically boot a Linux image from flash, set **bootcmd** like this:



setenv bootcmd dboot linux flash



If *bootdelay* is set to 0, the autoboot happens immediately after U-Boot starts. To stop the process and enter the monitor, press a key as soon as the first U-Boot output lines appear.

7 Using NVRAM

An embedded OS requires some persistent settings; for example, MAC address, IP address, Internet gateway, flash partition table, and U-Boot environment variables. Some of these are changed only in production and others only during custom setup.

These settings must be stored in non-volatile memory (NVRAM) so they are not lost when the target is powered off.

- For modules that have an I2C EEPROM (such as the ConnectCore 9P family), NVRAM is the EEPROM memory.
- For modules that do not have I2C EEPROM, a flash partition is reserved for this purpose.

The contents are protected by a CRC32 checksum. They also are mirrored to either a different I2C location or a second flash partition. In this way, if anything goes wrong or data becomes corrupted, the good image is taken and the bad one is automatically repaired when booting U-Boot or running the **intvram** command.

7.1 The intrvram command

Protected variables stored in NVRAM can be read, modified, erased or stored with the **intnvram** command.

Changes made to NVRAM with the **intnvram** command are kept in RAM. U-Boot writes the changes to NVRAM only when the **saveenv** command or **intnvram save** command is executed.



Executing an *envreset* resets U-Boot environment variables and saves them to NVRAM.

The syntax of the intnvram command is:

```
Usage: intnvram help|print <params>|printall|repair|reset|save|set <params>
           : prints this
 help
           : prints selected parameters.
 print
            E.g.: print module mac serialnr
 printall : prints complete contents and metainfo
 repair
          : Repairs the contents. If one image is
             bad, the good one is copied onto it.
            If both are good or bad, nothing happens.
           : resets everything to factory default values.
 reset
 save
          : saves the parameters
           : sets parameters.
 set
```

For help with this command, enter intnvram help.

To print the complete contents of the NVRAM settings, enter intnvram printall.

Either one parameter or a set of parameters can be set or printed. Parameters are grouped in blocks. This is the complete parameters list with the possible values some of them can take:



parama for "got"	or "print" can be
module	[producttype=] [serialnr=] [revision=] [patchlevel=]
liiodure	
	[ethaddr1=] [ethaddr2=]
network	[gateway=] [dns1=] [dns2=] [server=] [netconsole=] [ip1=]
	[netmask1=] [dhcp1=] [ip2=] [netmask2=] [dhcp2=]
partition	[add] [del] [select=] [name=] [chip=] [start=] [size=]
	[type=] [flag_fixed=] [flag_readonly=]
	[flag_fs_mount_readonly=] [flag_fs_root=] [flag_fs_type=]
	[flag_fs_version=]
os	[add] [del] [select=] [type=] [start=] [size=]
Params trailed w	with '=' require a value in the set command. In the print
command, '=' must	tn't be used.
Possible Values a	are
os type:	None,Critical,OS-Meta,U-Boot,Linux,EBoot,WinCE,Net+OS,
	Unknown, Application
partition type	: U-Boot, NVRAM, FPGA, Linux-Kernel, WinCE-EBoot, WinCE-Kernel,
	Net+OS-Kernel, Filesystem, WinCE-Registry, Unknown,
	Splash-Screen
flag_fs_type:	None, JFFS2, CRAMFS, INITRD, FlashFX, Unknown, YAFFS
Examples:	
-	module ethaddr1 serialnr : prints mac address and
F1110	serial number
intnyram print	partition select=0 name select=1 name : prints first and
Inchvian pillic	second partition
	name
intruram get n	etwork ip1=192.168.42.30 : changes the IP address
Inchivian Set In	SCWOLK IPI-172.100.12.30 · CHANYES CHE IF AUULESS

Specify the group of the parameter before the parameter itself. For example, to print the module's MAC address, execute:



#

intnvram print module ethaddr1 ethaddr1=00:40:9D:2E:92:D4

For printing different parameters of a block, the block must be used only once. For example, to print the module IP and subnet mask of Ethernet interface, execute:



```
#
   intnvram print network ip1 ethaddr1
ip1=192.168.42.30
ethaddr1=00:40:9D:2E:92:D4
```

To set a parameter a valid value must be provided, as shown here:



intnvram set network ip1=192.168.42.80

To access a partition parameter, address the specific partition with the parameter **select**=*n*, where *n* is the index to the partition. This example prints the names of partitions 1 and 2:



#

intnvram print partition select=0 name select=1 name name=U-Boot name=NVRAM

7.1.1 Mappings of variables

Some of the protected variables in NVRAM are mapped to U-Boot environment variables. Therefore, modifying them with **intnvram** command is the same as doing so with **setenv** command. For security reasons, however, some variables cannot be modified with the **setenv** command.

This table lists the mapped variables:

U-Boot variable	NVRAM parameter	Blocked for 'setenv'
ethaddr	ethaddr1	Х
wlanaddr	ethaddr2	Х
netmask	netmask1	
netmask_wlan	netmask2	
ipaddr	ip1	
ipaddr_wlan	ip2	
dnsip	dns1	
dnsip2	dns2	
dhcp	dhcp1	
dhcp_wlan	dhcp2	
serverip	server	
gatewayip	gateway	

7.2 The flpart command

To print, modify, or restore the partitions table, use the **flpart** command. This U-Boot command requires no arguments; the partitions table is created using a menu of options.

7.2.1 A partition table entry

A partition table entry contains these fields:

Field	Description
Number	Index of partition in the table
Name	Name of the partition
Chip	Index of flash chip (normally, only one)
Start	Physical start address of the partition (in hex)
Size	Size of the partition (in hex)
Туре	Partition type (what it will contain)
	U-Boot
	• NVRAM
	• FPGA
	Linux-Kernel
	WinCE-EBoot
	WinCE-Kernel
	Net+OS-Kernel
	Filesystem
	WinCE-Registry
	Unknown
FS	File system that the partition contains:
	• None
	• JFFS2
	CRAMFS
	• INITRD
	• FlashFX
	YAFFS
	Unknown
Flags	Flags (non-exclusive):
	• read-only
	mount read-only
	rootfs

7.2.2 Changing the partition table

To modify the partition table, use the **flpart** command in U-Boot:

~	#	flpart							
min	Nr	Name	Start		Size		Туре	FS	Flags
S. S	0 1 2 3 4 5 Comr	Name U-Boot NVRAM FPGA EBoot Registry Kernel mands: a) Append pa d) Delete pa m) Modify pa	768 1 1280 1 2304 1 3328 1 4352 1 artition artition	0 KiB KiB KiB KiB	768 512 1 1 1	KiB KiB MiB MiB MiB MiB	Type U-Boot NVRAM FPGA WinCE-EBoot WinCE-Registry WinCE-Kernel	None None None None None None	flags fixed fixed fixed
	I C) Print par) Reset par A) Quit (? for help	ctition ta						

Partitions are added, modified, or deleted step-by-step; the command prompts for the necessary information.

P

Start and Size values can be given as hexadecimal numbers (prefixed with $\mathbf{0x}$) or as decimal numbers followed with \mathbf{k} (for KiB) or \mathbf{m} (for MiB).

The partition table also can be reset to the default values. In this case, because the partition table differs according to the target's OS, the desired OS can be selected.



Changes take effect only after quitting 'flpart' and saving the changes.

When the size or start address of a partition has been changed, it is always necessary to erase it and write a new image to it.

8 Firmware update commands

8.1 Overview

The boot loader, kernel, and other data stored in flash form the firmware of the device. Because U-Boot can write any part of flash, its flash commands can be used to reprogram (update) any part of the firmware. This includes the boot loader itself.

The update process normally takes place in three steps:

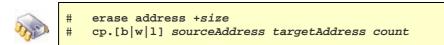
- Reading image from media (Ethernet, USB) into RAM memory
- Erasing the flash that is to be updated
- Copying the image from RAM into flash

8.2 Updating flash with images in RAM

Flash memory must be updated with images located in RAM memory. Images are moved to RAM using either Ethernet or USB (see section 6.2 for more information).

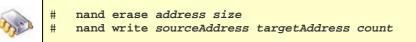
To erase flash and copy the images from RAM to flash, use these commands:

• For NOR flash memory:



The first command erases *size* bytes beginning at *address*. The second command copies *count* bytes, words or long words (depending on the suffix used: b, w, I) from *sourceAddress* into *targetAddress*.

For NAND flash memory:



The first command erases *size* bytes beginning at *address*. The second command copies *count* bytes from *sourceAddress* into *targetAddress*.



The erasure of the flash comprises whole erase-blocks. The *address* and *size* parameters must be multiples of the erase-blocks of the flash memory. See the module's flash datasheet for the erase-block size.

8.3 Direct updating

Digi's U-Boot version includes the built-in **update** command. This command copies the image from the media to RAM, erases the flash size needed for the image, and moves the image from RAM into flash in a single step, simplifying the update process.

Here is the syntax for update:

```
# help update
update partition source [file]
- updates 'partition' via 'source'
values for 'partition': uboot, linux, rootfs, userfs, eboot, wce
or any partition name
values for 'source': tftp, usb
values for 'file': the file to be used for updating
```



If the **dhcp** variable is set to **yes** or **on**, the command first gets an IP address from the DHCP server pointed to by the **serverip** variable.

8.3.1 Update limits

The **update** command in U-Boot transfers files to RAM, erases the flash partition, and writes the files from RAM into flash memory.

The file that is transferred is copied to a specific physical address in RAM; therefore, the maximum length of the file to update is:

Update file size limit = Total RAM memory - RAM offset where the file was loaded

As a general rule, U-Boot does not allow updating a flash partition with a file size that exceeds the available RAM memory. This means that, for example, if a module has 32MB RAM and 64MB flash and the file for updating a partition is 35MB, U-Boot will not do it.

This limitation is due to the RAM memory size, as U-Boot first needs to transfer the file to RAM before copying it to flash.



For updating partitions with files larger than the available RAM memory, see your OS-specific update flash tool.

9 U-Boot development

U-Boot is an open source project. Sources are freely distributed, and can be modified to meet requirements for a boot loader.

The project sources are ready to be installed and compiled in a Linux environment. If a Linux machine is not available for development, install the *Cygwin X-Tools* software (<u>http://www.cygwin.com</u>). The *X-Tools* provide a Unix-like development environment for Windows, based on *Cygwin* and the GNU toolchain, to cross-compile the boot loader.

For information about installing the U-Boot sources, modifying platform-specific sources, and recompiling the boot loader, see your development kit documentation. Procedures may vary according to hardware platform and OS.

Index

Α

autoload system variable	16
automatic booting	
autostart system variable	
available commands	10
	40
and hardware platform	10
available RAM	
and update limits	28

В

baudrate system variable	16
bdinfo command	11
BIOS	6, 7
boot loader	
defined	7
boot process	8
bootcmd system variable	16
bootdelay system variable	16
booting images in RAM	19
bootm command	
and Windows CE images	11
bootp command	11
built-in commands	
creating	9

С

cmp command	12
commands	
boot	18
built-in	9
environment variables	13
firmware update	27
flpart	24
information	11
intnvram	22
memory	12
network	
serial port	13
type-ahead	10
update	28
USB	12

coninfo command	11
conventions in this manual	5
cp command	12
creating built-in commands	9
custom variables	17

D

date command	11
dhcp command	11
dhcp system variable	16
dhcp wlan system variable	16
Digi custom variables	17
direct booting	20
dnsip system variable	16
dnsip2 system variable	16
dynamic variables	17

Ε

environment variable commands13
environment variables
simple and recursive14
envreset command13

F

FAT	6
fatinfo command	11
fileaddr system variable	16
filesize system variable	16
firmware	
update commands	27
flash	
updating with images in RAM	27
flpart command	24

G

gatewayip system variable	16
go command	12

Н

hardware platform
and available commands10

I

I2C	6
I2C commands	13
iloop command	13
imd command	13
iminfo command	11
imm command	13
imw command	13
information commands	11
inm command	13
intnvram	22
ipaddr system variable	16
ipaddr wlan system variable	
iprobe command	13
itest command	13

L

loadb command	13
loads command	13
loady command	13

Μ

MBR 6
md command12
memory
comparing12
copying12
copying contents from flash to RAM address12
copying contents from RAM to flash address12
displaying contents12
dumping NAND page12
erasing12
modifying a fixed location 12
modifying contents12
writing values to 12
memory commands 12
mm command12
mw command 12

Ν

nand bad command	. 11
nand dump command	.12

nand erase command 12
nand info command11
nand read command12
nand write command12
nboot command 12, 13
nboot memory
booting image from NAND device 13
netmask system variable16
netmask wlan system variable16
network commands11
nfs command11
nm command12
non-volatile memory (NVRAM)
and persistent settings22
ntpserverip system variable
NVRAM6

Ρ

partition table	
entries, contents of	25
modifying	26
resetting	26
persistent settings	22
ping command	11
printenv command	13
protected variables	17

R

rarpboot command	11
reading images into RAM	
recursive variables	14

S

saveenv command	13
scripts	15
serial port commands	13
serverip system variable	16
setenv command	13
simple variables	14
sntp command	11
stderr system variable	16
stdin system variable	16
stdout system variable	16
system variables	16

Т

TFTP	6
tftpboot command	11

U

U-Boot	
boot process	8
built-in commands	9
command shell (monitor)	7
commands	9
described	7
development	29
hexadecimal assumed for numbers.	8
source code, modifying	9
variables	8
update command	28
update limits	
and available RAM	28

updating flash	27
USB	6
USB commands	12

V

17
17
13, 17
13, 14
15
24
17
14
14
16
16
11