U-Boot Reference Manual

90000852_K

Digi International Inc.

11001 Bren Road East

Minnetonka, MN 55343 (USA)

☎ +1 877 912-3444 or +1 952 912-3444

http://www.digi.com

Contents

1. Conventions used in this manual

This list shows the typographical conventions used in this guide:

Style	Used for file and directory names, variables in commands, URLs and new terms.
`Style`	In examples, to show the contents of files, the output from commands, the C code. Variables to be replaced with actual values are shown in italics.
`Style`	Variable's names and commands. In examples, to show the text that should be typed literally by the user.
#	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>	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

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

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

```
This is an excerpt from a file
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
MII	Media Independent Interface
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 doesn't 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 *http://sourceforge.net/projects/u-boot/* and *http://www.denx.de/wiki/DULG/Manual*.

3.3 Features of U-Boot

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) in which you work 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 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

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 length of time you can program), 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*).

```
UU-Boot 1.1.6 (Aug 28 2009 - 14:03:27 - GCC 4.3.2) DUB-RevF3
for ConnectCore 9M 2443 on Development Board

DRAM:  64 MB
NAND:  128 MB
CPU:   S3C2443@534MHz
       Fclk = 534MHz, Hclk = 133MHz, Pclk = 66MHz
Autoscript from TFTP... [not available]
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. By modifying U-Boot source code, you can create your own built-in commands.

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
- ?

You see a list like this one

```
#   help
?       - alias for 'help'
autoscr - run script from memory
base    - print or set address offset
bdinfo  - print Board Info structure
boot    - boot default, i.e., run 'bootcmd'
bootd   - boot default, i.e., run 'bootcmd'
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
clock   - Set Processor Clock
cmp     - memory compare
coninfo - print console devices and information
cp      - memory copy
crc32   - checksum calculation
date    - get/set/reset date & time
dboot   - Digi ConnectCore modules boot commands
dcache  - enable or disable data cache
dhcp   - invoke DHCP client to obtain IP/boot params
echo    - echo args to console
envreset- Sets environment variables to default setting
fatinfo - print information about filesystem
fatload - load binary file from a dos filesystem
fatls   - list files in a directory (default /)
flpart  - displays or modifies the partition table.
fsinfo - print information about filesystems
fsload - load binary file from a filesystem image
go      - start application at address 'addr'
help    - print online help
icache  - enable or disable instruction cache
icrc32  - checksum calculation
iloop   - infinite loop on address range
imd     - i2c memory display
iminfo  - print header information for application image
imm     - i2c memory modify (auto-incrementing)
imw     - memory write (fill)
inm     - memory modify (constant address)
intnvram- displays or modifies NVRAM contents like IP or partition table
iprobe  - probe to discover valid I2C chip addresses
itest  - return true/false on integer compare
loadb   - load binary file over serial line (kermit mode)
loads   - load S-Record file over serial line
loady   - load binary file over serial line (ymodem mode)
loop    - infinite loop on address range
```

```
ls      - list files in a directory (default /)
md       - memory display
mm       - memory modify (auto-incrementing)
mtest    - simple RAM test
mw       - memory write (fill)
nand     - NAND sub-system
nboot    - boot from NAND device
nfs     - boot image via network using NFS protocol
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
sleep    - delay execution for some time
sntp    - synchronize RTC via network
tftpboot- boot image via network using TFTP protocol
update   - Digi ConnectCore modules update commands
usb      - USB sub-system
usbboot - boot from USB device
version - print monitor version
```

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

For more information about a command, enter:

help *command name*

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

*As you enter the first letters of a command, U-Boot searches its list of built-in commands until it finds a match. For example, if you enter **save** or **sav** or even **sa**, U-Boot executes the **saveenv** command.*

*You need to enter enough letters for U-Boot to determine the command to execute. For example, if you enter **loa** U-Boot cannot tell whether to execute **loadb**, **loads** or **loady**, and you get an 'Unknown command' message.*

4.2.1 Information commands

To get information, use these commands:

Command	Description
bdinfo	Prints board info structure.
coninfo	Prints console devices and information.
date [MMDDhhmm[[CC]YY][.ss]]	Gets / sets / resets system date/time.
fatinfo <interface> <dev[:part]>	Prints information about the file system from *'dev'* on *'interface.'*
flinfo [bank]	Prints information about the flash memory banks.
fsinfo	Prints information about file systems.
iminfo [addr ...]	Prints header information for the application image starting at the *'addr'* address in memory, including verification of the image contents (magic number, header, and payload checksums). This command works only for Linux kernel images.

nand bad	Shows NAND bad blocks.
nand info	Shows available NAND devices.
mii info <addr>	Prints MII PHY info
version	Displays U-Boot version and timestamp.

4.2.2 MII commands

To access the Ethernet PHY use this commands:

Command	Description
mii device	Lists available devices.
mii device <device name>	Set current device.
mii read <addr > <reg>	Reads register *'reg'* from MII PHY *'addr'*.
mii write <addr > <reg> <data>	Writes *'data'* to register *'reg'* at MII PHY *'addr'*.
mii dump <addr > <reg>	Displays data of register *'reg'* from MII PHY *'addr'*.

The parameter addr and reg can be range e.g. 2-7.

The command mii dump is only usable for register 0-5.

4.2.3 Network commands

This table shows the network-related commands:

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

Netconsole is not supported, because of poor performance.

If the *autostart* variable is set to 'yes', all these commands (except *ping* and *sntp*) boot the transferred image by calling the *bootm* command.

***bootm* does not work for WinCE images. If you are working with a WinCE image file, either set the *autostart* variable to 'no' or delete it before executing these network commands.**

4.2.4 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 [dev]	Shows available USB devices
usb storage	Shows details of USB storage devices
usb dev [dev]	Shows or set current USB storage device
usb part [dev]	Prints the partition table of one or all USB storage devices
usb read addr blk# cnt	Reads *'cnt'* blocks starting at block *'blk#'* to RAM address *'addr'*
fatload usb <dev[:part]> <addr> <filename>	Reads *'filename'* image from FAT partition *'part'* of USB device *'dev'* into the RAM memory address *'addr'*. If *part* is not specified, partition 1 is assumed.
ext2load usb <dev[:part]> <addr> <filename>	Reads *'filename'* image from EXT2/3 partition *'part'* of USB device *'dev'* into the RAM memory address *'addr'*. If *part* is not specified, partition 1 is assumed.

4.2.5 Memory commands

These commands manage RAM memory:

Command	Description
cmp[.b, .w, .l] addr1 addr2 count	Compares memory contents from address *'addr1'* to *'addr2'* for as many *'count'* bytes, words, or long words.
cp[.b, .w, .l] source target count	Copies memory contents from address *'source'* to *'target'* for as many *'count'* bytes, words, or long words.
dcache [on|off]	Turns data cache on or off.
eeprom read <addr> <off> <cnt>	Copies *'cnt'* bytes memory contents from eeprom offset *'off'* to RAM address *'addr'*.
eeprom write <addr> <off> <cnt>	Copies *'cnt'* bytes memory contents from RAM address *'addr'* to eeprom offset *'off'*.
erase_pt <name>	Erases the partition *'name'*. With flpart the *'name'* can be found.
go addr [arg ...]	Starts the application at address *'addr'* passing 'arg' as arguments.

imls	Prints information about all images found at sector boundaries in flash.
icache [on|off]	Turns instruction cache on or off.
md[.b, .w, .l] <address> [# of objects]	Displays the contents of the memory at address *'addr'* for as many *'[#of objects]'* bytes, words, or long words.
mm[.b, .w, .l] <address>	Lets you modify locations of memory, beginning at *'address,' which* gets auto-incremented.
mw[.b, .w, .l] <address> <value > [count]	Writes *'value'* into *'address'* for as many *'count'* bytes, words, or long words.
nm[.b, .w, .l] address	Lets you modify a fixed location of memory.
nand[.jffs2] read <addr> <off> <size>	Copies the memory contents from flash address *'off'* to RAM address *'addr'* for as many *'size'* bytes (only for NAND flash memories). Bad block management is used, when using .jffs2. The bad block management detects bad blocks and skips them.
nand[.jffs2] write <addr> <off> <size>	Copies the memory contents from RAM address *'addr'* to flash address *'off'* for as many *'size'* bytes (NAND flash memories only). Bad block management is used, when using .jffs2. The bad block management detects bad blocks and skips them.
nand erase [off size]	Erases *'size'* bytes from address *'off'*. Erases the 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 *'off'* with optional out-of-band data (only for NAND flash memories).
nand markbad <off>	Marks block at *'off'* as bad.
nand unmarkbad <off>	Erases bad block at *'off'*.
nboot address dev [off]	Boots image from NAND device *dev* at offset *off* (transferring it first to RAM *address*).
protect [on|off] ...	Protects/unprotects NOR sector(s).

The eeprom command uses default eeprom configured by CFG_I2C_EEPROM_ADDR.

When writing your own boot macro, make sure data cache and instruction cache are turned off right before booting the OS.

4.2.6 Serial port commands

Use these commands to work with the serial line:

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

4.2.7 I2C commands

These commands interface with the I2C interface:

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

4.2.8 Environment variables 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 *'wlanaddr' or 'ethaddr'* variables or any other persistent settings stored in NVRAM (see topic 8.2).
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 *'name´* to value *'value'*.

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. In this way, you can recover 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 00    .....
```

You must use the back slash '\' before '$' to prevent variables from being expanded into other variables' values.

5.3 Scripts

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

Consider this variable:

```
#   printenv cmd1
setenv var val;printenv var;saveenv
```

If you were to run this script, with **run cmd1** the **var** variable would be created with **val** value, the value would be printed to the console, and the variables would be saved to either the EEPROM or flash partition dedicated to variables.

```
#   run cmd1
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 semi-colons. (;). As with recursive variables, this sign must be preceded by a back-slash sign or it is considered the termination of the first command itself.

This is how you would save **cmd1**:

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

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

```
#   setenv cmd1 setenv var val
#   setenv cmd2 printenv var
#   setenv cmd3 saveenv
#   run cmd1 cmd2 cmd3
```

See how to create a bootscript that automatically executes at start at topic 6.

5.4 System variables

U-Boot uses several built-in 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.
filesize	Contains the size of the last file transferred by TFTP or USB.
fileaddr	The RAM address where the last file transferred by TFTP was placed.
ntpserverip	NTP server IP address (for getting the date/time).
stdin	Standard input system.
stdout	Standard output system.
stderr	Standard error output system.

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.
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).
gatewayip	IP address used as network gateway.
serverip	IP address of the host PC (for remote connections like TFTP transfers).

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 your development kit's documentation.

5.4.3 User keys

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.

If you press either key when the boot loader is starting, the *key1* or *key2* variable is executed before the **bootcmd**. This lets you have different boot scripts, depending on the key pressed during boot, so you can boot 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**.

You can disable detection of user keys for customized hardware where these keys don't exist. To do so, you need to reconfigure and recompile U-Boot. See chapter 11 for information about U-Boot development.

For the Digi Connect ME 9210 only user key 2 is enabled by default.

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. Bootscript

The bootscript is an script that is automatically executed when the boot loader starts, and before the OS auto boot process.

The bootscript allows the user to execute a set of predefined U-Boot commands automatically before proceeding with normal OS boot. This is especially useful for production environments and targets which don't have an available serial port for showing the U-Boot monitor.

6.1 Bootscript process

The bootscript works in the following way:

1. U-Boot checks the variable **loadbootsc**. If set to "no", it continues normal execution.
2. If the variable **loadbootsc** is set to "yes" (factory default value) U-Boot tries to download the bootscript file with the filename stored in variable **bootscript** from the TFTP server IP address defined at variable **serverip** (by default 192.168.42.1).

   The default value of the **bootscript** variable is *<platformname>-boot*script.
3. If the bootscript file is successfully downloaded, it is executed.
4. If any of the commands in the bootscript fails, the rest of script is cancelled.
5. When the bootscript has been fully executed (or cancelled) U-Boot continues normal execution.

6.2 Creating a bootscript

To create a bootscript file:

1. Create a plain text file with the sequence of U-Boot commands. Usually, it is recommended that the last command sets the variable **loadbootsc** to "**no**", to avoid the bootscript from executing a second time.

   For example, create a file called *myscript.txt* with the following contents:

```
setenv company digi
setenv bootdelay 1
printenv company
setenv loadbootsc no
saveenv
```

   This script creates a new variable called **company** with value **digi** and sets the bootdelay to one second. Finally it sets the variable **loadbootsc** to "**no**" so that U-Boot doesn't try to execute the bootscript in the future, and saves the changes.

2. Execute the *mkimage* tool (provided with U-Boot) with the file above as input file. Syntax is:

```
mkimage -T script -n "Bootscript" -C none -d <input_file> <output_file>
```

   The name of the output file must be in the form ***<platformname>-boot*script**, where <platformname> must be replaced with your target's platform name.

   For example, to create the bootscript from the text file above and for a Connect ME 9210 platform, go to the U-Boot directory and execute:

```
$   tools/mkimage -T script -n "Bootscript" -C none -d myscript.txt
    cme9210js-bootscript
```

6.2.1 Creating a bootscript in Windows

There are some important caveats when producing a bootscript in a Windows host PC.

6.2.1.1 Final carriage return

When creating the plain text file with the sequence of commands, make sure that the last command in the sequence contains a final carriage return (in other words, a blank line at the end of the file). Otherwise, the last command in the sequence may not execute.

6.2.1.2 Windows line-end characters

If developing U-Boot in a Windows host (using Cygwin), the plaintext file with the list of commands might contain incorrect line-end characters. If this is the case, the plaintext file must be converted first to UNIX line-end format before generating the bootscript. This can be done using the dos2unix application over the file, before calling *mkimage*:

```
$   dos2unix.exe myscript.txt
```

The prebuilt *mkimage* tool is not included with the *Cygwin* development environment. This tool is built the first time you compile U-Boot under *Cygwin*.

6.3 Configuration for launching the bootscript

Once the bootscript file has been created two more steps are needed to let the target run the bootscript at start.

1. Place the bootscript file into the TFTP exposed folder, so that the target is able to find it when it boots.
2. The U-Boot variable **serverip** of the target must point to the host PC with the TFTP server. You have two options:
    a. Connect to the target's U-Boot monitor and set the **serverip** variable to the IP of your host PC.
    b. If you don't have access to the U-Boot monitor or simply don't want to have any user interaction with the target (for example in a production environment), configure the host PC Ethernet card's IP to the factory default IP address stored in variable **serverip**, which is **192.168.42.1**.

Once done with all the steps, power up the target and it will connect to the host PC, will download the bootscript to RAM, execute it, and continue booting as usual.

6.4 Bootscript restrictions

The Digi U-Boot command **flpart** (for partitioning the Flash) is a menu-driven program, which expects key presses for different user selections. This command may not work in a bootscript. For repartitioning the Flash, use the command **intnvram** instead (refer to chapter 8.2 for more information).

7. Boot commands

7.1 Overview

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

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

7.2 Reading images into RAM

7.2.1 From Ethernet

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

- The address of RAM in which to place the image
- The image file name

```
#   tftpboot <loadAddress> <bootfilename>
```

The TFTP transfer takes place 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.

7.2.2 From USB

Another way to boot an image is by reading it from a USB flash storage device. The USB disk must be formatted either in FAT, ext2, or ext3 file system.

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

```
#   usb reset
#   fatload usb <dev>[:partition] <loadAddress> <bootfilename>
```

If the flash disk is formatted in ext2/ext3:

```
#   usb reset
#   ext2load usb <dev>[:partition] <loadAddress> <bootfilename>
```

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.

7.2.3 From MMC

If the target has an MMC or HSMMC (High Speed MMC) interface U-Boot can also read from it. The MMC card must be formatted either in FAT, ext2, or ext3 file system.

To read an image from an MMC card formatted in FAT, enter:

```
#   mmcinit
#   fatload mmc <dev>[:partition] <loadAddress> <bootfilename>
```

If the card is formatted in ext2/ext3:

```
#   mmcinit
#   ext2load mmc <dev>[:partition] <loadAddress> <bootfilename>
```

This command reads file *bootfilename* from device *dev*, partition *partition* of the MMC card 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.

7.2.4 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, l - from *sourceAddress* into *loadAddress*.

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

```
#   nand read <loadAddress> <sourceAddress> <count>
```

This command copies *count* bytes from *sourceAddress* into *loadAddress*.

7.3 Booting images in RAM

After the image is transferred to RAM, you can boot 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.

7.4 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> <dev>[:partition] <filesystem> <bootfilename>
```

where

- *os* is either **linux, wce** or **netos**.
- *media* is either **flash**, **tftp, nfs**, **usb**, **mmc**, or **hsmmc**.
- *dev[:partition]* is the device index (only for USB, MMC, HSMMC media) starting at 0 and the partition number (starting at 1) where the image to boot resides. If not provided, device 0 and partition 1 are assumed
- *filesystem* is either **fat**, **ext2**, or **ext3** (only for USB, MMC, HSMMC media) and must match the file system of the partition that holds the image to boot. If not provided, FAT is assumed.
- *bootfilename* is the name of the kernel image file to download and boot. If not provided, the filename is taken from 'kimg' variable for Linux, or 'wimg' for Windows CE

*If booting from a network media (tftp, nfs) and the **dhcp** variable is set to **yes** or **on**, the command first gets an IP address from a DHCP.*

7.4.1 Boot examples

Boot default Windows CE image stored in a USB disk with partition 1 formatted in FAT, plugged in the USB host device 0:

```
#   dboot wce usb
```

Boot default Linux image stored in an SD card with partition 3 formatted in ext3, plugged in the USB host device 0:

```
#   dboot linux mmc 0:3 ext3
```

Refer to your OS user manual for further instructions on booting your kernel image.

7.5 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:

```
#   run bootcmd
```

```
#   boot
```

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

```
#   setenv bootcmd dboot wce tftp
```

Or, if you want 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.

8. 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. You change some of these 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 you power the target off.

A partition called NVRAM on the flash memory is used to store these settings. The contents are protected by a CRC32 checksum and they are also mirrored to a different location in the partition. This way, if anything goes wrong reading these data or data becomes corrupted, the information can be restored from the mirrored data.

8.1 The 'flpart' command

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

8.1.1 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)
Type	Partition type (what it will contain) • U-Boot • NVRAM • FPGA • Linux-Kernel • WinCE-EBoot • WinCE-Kernel • Net+OS-Kernel • Net+OS-Loader • Net+OS-NVRAM • File system • WinCE-Registry • Splash-Screen • Unknown
FS	File system that the partition contains: • YAFFS • JFFS2 • CRAMFS • SQUASHFS • INITRD

	• FlashFX • Unknown
Flags	Flags (non-exclusive): • read-only • mount read-only • rootfs

8.1.2 Changing the partition table

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

```
#  flpart
Commands:
   a) Append partition
   d) Delete partition
   m) Modify partition
   p) Print partition table
   r) Reset partition table
   q) Quit
Cmd (? for help)> p
Nr | Name      |  Start    | Size      | Type            | FS      | Flags
----------------------------------------------------------------------
 0 | U-Boot    |         0 |   768 KiB | U-Boot          | None    | fixed
 1 | NVRAM     |   768 KiB |   256 KiB | NVRAM           | None    | fixed
 2 | FPGA      |     1 MiB |     1 MiB | FPGA            | None    | fixed
 3 | EBoot     |     2 MiB |     1 MiB | WinCE-EBoot     | None    |
 4 | Registry  |     3 MiB |     1 MiB | WinCE-Registry  | None    |
 5 | Kernel    |     4 MiB |    20 MiB | WinCE-Kernel    | None    |
 6 | FFX       |    24 MiB |     2 MiB | Filesystem      | FlashFX |
```

You add, modify, or delete partitions step-by-step; the command prompts you for the necessary information.

*Start and Size values can be given as hexadecimal numbers (prefixed with **0x**) or as decimal numbers followed with **k** (for KiB) or **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, you select the OS you want.

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.

8.2 The 'intnvram' command

Most of the variables stored in NVRAM can be read with the **printenv** command, modified or erased with the **setenv** command, stored with the **saveenv** command, and reset with the **envreset** command. There are, however, protected variables in the NVRAM which are read-only. These are, for example, the MAC address of the module, the serial number, the boot and NVRAM partitions, the wireless calibration data, etc.

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

DO NOT USE THE intnvram COMMAND UNLESS YOU ARE COMPLETELY SURE OF WHAT YOU ARE DOING. INCORRECT USE CAN DESTROY SENSITIVE NVRAM DATA FOREVER AND MAKE THE MODULE UNUSABLE

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

Here is the syntax of the **intnvram** command:

```
Usage: intnvram help|print <params>|printall|repair|reset|save|set <params>

  help     : prints this
  print    : prints selected parameters.
             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.
  reset    : resets everything to factory default values.
  save     : saves the parameters
  set      : sets parameters.
```

For help with this command, enter **intnvram help**.

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

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

```
  params for "set" or "print" can be
     module    [producttype=] [serialnr=] [revision=] [patchlevel=]
               [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 with '=' require a value in the set command. In the print
command, '=' mustn't be used.

Possible Values 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
```

Specify the group of the parameter before the parameter itself. For example, to print the module IP for the wired Ethernet interface, execute:

```
#   intnvram print network ip1
ip1=192.168.42.30
```

For printing different parameters of a block, the block must be used only once. For example, to print the module's MAC address and serial number, execute:

```
#   intnvram print module ethaddr1 serialnr
ethaddr1=00:40:9D:2E:92:D4
serialnr=0700-94000329A
```

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

```
#   intnvram set module serialnr=REVA-6_001
```

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
```

The 'reset' command will completely erase the MAC addresses and the wireless calibration data of your module. Do not use the 'reset' command unless you have a backup file with the wireless calibration data for future restoration.

8.2.1 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	X
wlanaddr	ethaddr2	X
netmask	netmask1	
netmask_wlan	netmask2	
ipaddr	ip1	
ipaddr_wlan	ip2	
dnsip	dns1	
dnsip2	dns2	
dhcp	dhcp1	
dhcp2	dhcp2	
serverip	server	
gatewayip	gateway	

9. Firmware update commands

9.1 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, MMC) into RAM memory
- Erasing the flash that is to be updated
- Copying the image from RAM into flash

9.2 Updating flash with images in RAM

Flash memory must be updated with images located in RAM memory. You can move images to RAM using either Ethernet, USB, or MMC (see section 7.2 for more information).

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

- For NOR flash memory:

```
#   erase address +size
#   cp.[b|w|l] sourceAddress targetAddress count
```

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, l) from *sourceAddress* into *targetAddress*.

- For NAND flash memory:

```
#   nand erase address size
#   nand write sourceAddress targetAddress count
```

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 your module's flash datasheet for the erase-block size.

9.3 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 [device:part filesystem] [file]]
  - updates 'partition' via 'source'
    values for 'partition': uboot, linux, rootfs, userfs, eboot, wce, wcez,
                          netos, netos_loader, splash, or any partition name
    values for 'source': tftp, nfs, usb, mmc, hsmmc
    'device:part': number of device and partition, for 'usb', 'mmc', 'hsmmc'
                   sources
    values for 'filesystem': fat|vfat, ext2|ext3
    values for 'file'  : the file to be used for updating
```

- *source* is the place to take the image from. If not provided, tftp is assumed.
- *dev[:partition]* is the device index (only for USB, MMC, HSMMC media) starting at 0, and the partition number (starting at 1) where the image to update resides. If not provided, device 0 and partition 1 are assumed
- *filesystem* is either **fat**, **ext2**, or **ext3** (only for USB, MMC, HSMMC media). If not provided, FAT is assumed.
- *file* is the name of the image to download and update. If not provided, the filename is taken from one of the following U-Boot environment variables (depending on the partition to be updated):
  - ***kimg***: for the Linux kernel image
  - ***wimg***: for the Windows CE kernel image
  - ***nimg***: for the NET+OS kernel image
  - ***uimg***: for the boot loader image
  - ***usrimg***: for the user image
  - ***rimg***: for the Linux root file system image
  - ***simg***: for the Splash screen image
  - ***fimg***: for the FPGA image
  - ***nloader***: for the Windows CE kernel image

*If updating from a network media (tftp, nfs) and the **dhcp** variable is set to **yes** or **on**, the command first gets an IP address from a DHCP server.*

For example, to update the Splash screen partition using a file called ‘mylogo.bmp’ that resides on the second partition (formatted in ext3) of an SD card which is plugged in the first MMC device (index 0), the update command would be:

```
#   update splash mmc 0:2 ext3 mylogo.bmp
```

To update the boot loader from the TFTP exposed folder with the default name image stored in variable **uimg**, the update command would be:

```
#   update uboot
```

9.3.1 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 let you update a flash partition with a file whose size exceeds the available RAM memory. This means that, for example, if you have a module with 32MB RAM and 64MB flash and you want to update a partition with a file that is 35MB, U-Boot will not do it.

Note that 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.

10. Customize U-Boot

10.1 Overview

U-Boot has a lot of functionalities, which can only be enabled before compiling U-Boot. To configure U-Boot options and customize the boot loader, refer to your OS specific user manual.

Description of some available configurations follow:

10.2 JTAG Console

The JTAG console uses the JTAG interface instead of the serial line, which is used by the default console. The Requirements for the JTAG console are a debugger and a host application to communicate with the Direct Communication Channel. The BDI2000 supports input/output and the Segger jlink supports output only.

This configuration does not work together with silent mode.

Before you can start using JTAG-Console you have to configure your debugger for using the Direct Communication Channel. Using a BDI200 you have to add the following line in section *[TARGET]* to your *BDI configuration file*:

```
DCC        7
```

After that type:

```
$   telnet <bdi2000 ip-address> 7
    Connected to bdi2000.
    Escape character is '^]'.
```

Using a Segger jlink the version 3.87i or higher of jlink-software is required. Start the jlinkcommander and type:

```
$   term
```

Only stdout can be used with the Segger jlink.

After starting your target you can switch output and input independently to JTAG console by typing

```
#   setenv stdout jtag
#   setenv stdin jtag
#   setenv stderr jtag
#   saveenv
```

After reboot the JTAG console is used by default, if you have stored the environment.

10.3 Silent Console

The target does not display any output when the console is set to silent mode. Before using a boot loader with silent console, you should first define a way to recover from it. Otherwise you will not be able to disable it in the future.

The first possibility is using the key environments variables (see topic 5.4.3):

```
#   setenv key1 setenv silent no\;saveenv
#   saveenv
```

The sequence to recover from silent mode is the following:

1. Keep Key1 pressed while the target is booting
2. After a short time (about 4 seconds) press the reset button
3. The target now boots with output on the console

The second possibility to recover from silent mode is using a gpio. To use this functionality you have to define the gpio number and the level of the gpio, which signals the console to leave the silent mode. The number of the gpios can be found in the hardware reference.

You will need the JTAG interface to flash the firmware, when the function to recover from silent mode is undefined.

The console is switch to silent mode after reboot by typing:

```
#   setenv silent yes
#   saveenv
```

After reboot you can recover from silent mode, by setting the gpio to the defined level short after the target starts to boot or when the target end to boot. The second point is only reached, when autoboot is not used or the execution fails.

10.4 Splash screen support

The U-Boot splash screen support allows the user to display a picture at boot time on an LCD or monitor that is connected to the target. The picture is read from a partition in the flash.

Some platforms have splash support already enabled in U-Boot and some have not. To determine if your platform has splash support enabled, check if the variable *video* exists:

```
#   printenv_dynamic
```

If it exists and has some value, there should also be a partition for the splash screen image. Check it with command **flpart** and option **p** to print the partition table. If there is no splash screen partition, refer to chapter 8.1 to create one. If there is, you can go to chapter 10.4.3 to see how to upload a splash image to the flash partition.

10.4.1 Enable splash screen support in U-Boot

Go to your OS configuration tool for U-Boot and enable the Display support and splash screen support. You will need to select the displays you want to support (more than one can be supported):

Display	resolution
Sharp LQ057Q3DC12I	320x240
Sharp LQ064V3DG01	640x480
VGA monitor	640x480
EDT 8-inch QVGA	320x240
Sharp LQ070Y3DG3B1	800x480

Note that, depending on your platform, some displays might not be available.

Save the changes and recompile U-Boot. Then update the U-Boot image in your target.

10.4.2 Creating a splash partition

A partition is needed to hold the splash screen image. If your partition table doesn't have one, you need to create it with the **flpart** command:

```
#   flpart
Nr | Name          |  Start      | Size         | Type            | FS     |
Flags
---------------------------------------------------------------------
 0 | U-Boot         |     0      |768 KiB |U-Boot        |        | fixed
 1 | NVRAM          |   768 KiB |512 KiB |NVRAM         |        | fixed
 2 | Kernel         | 1280 KiB |   3 MiB |Linux-Kernel |        |
 3 | RootFS-JFFS2   | 4352 KiB |  16 MiB |Filesystem    | JFFS2 | rootfs
 4 | User-JFFS2     |20736 KiB |  11 MiB | Filesystem   | JFFS2 |
Commands:
   a) Append partition
   d) Delete partition
   m) Modify partition
   p) Print partition table
   r) Reset partition table
   q) Quit
Cmd (? for help)> a
Last partition 4 had already maximum size.
  Size  (in MiB, 0 for auto, 11 MiB max) (11 MiB): 10 MiB
Adding partition # 5
  Name  (): Splash
  Chip  (0): 0
  Start (in MiB, 0 for auto) (0): 0
   --> Set to 31 MiB
  Size  (in MiB, 0 for auto, 1 MiB max) (0): 1MiB
  Partition Types
Partition Type (U-Boot, ? for help)> s
  Fixed  (n): n
  Readonly  (n): n
Partition 5 added
Cmd (? for help)> p
Nr | Name          |  Start     | Size   | Type            | FS     | Flags
---------------------------------------------------------------------
 0 | U-Boot        |     0      |768 KiB |U-Boot        |        | fixed
 1 | NVRAM         |   768 KiB |512 KiB |NVRAM         |        | fixed
 2 | Kernel        | 1280 KiB |   3 MiB |Linux-Kernel |        |
 3 | RootFS-JFFS2 | 4352 KiB |  16 MiB |Filesystem    | JFFS2 | rootfs
```

```
 4 | User-JFFS2   |20736 KiB | 10 MiB |Filesystem    | JFFS2 |
 5 | Splash       |    31 MiB |  1 MiB |Splash-Screen |       |
Cmd (? for help)>q
Partition table has been modified. Save? (y): y
Writing Parameters to NVRAM
```

10.4.3 Uploading a splash image

After the partition is created the splash image has to be flashed. Only 8-bit BMP images can be used. Images must match the display resolution.

Uploading is done by following command (in the example, the bitmap file *digi.bmp* is used):

```
# update Splash tftp digi.bmp
TFTP from server 192.168.42.1; our IP address is 192.168.42.30
Filename 'digi.bmp'.
Load address: 0x200000
Loading: ######
done
Bytes transferred = 77878 (13036 hex)
Calculated checksum = 0x2a13f7f
Erasing:   complete
Writing:   complete
Verifying: complete
Update successful
```

Or, if the variable **simg** contains the BMP filename, you can simply execute:

```
# update splash
```

A splash image takes about 75 KiB for QVGA resolution and 300 KiB for VGA resolution.

If the splash image does not match the resolution of the display or it is not 8-bit pallet bitmap, U-Boot will print an error message and will not show the splash screen.

10.4.4 Initialize video interface

To initialize the video interface in U-Boot, the following variables must be set:

```
#   setenv videoinit yes
#   setenv video displayfb:DISPLAYNAME
#   saveenv
```

where DISPLAYNAME is the name of the connected display:

- LQ057Q3DC12I
- LQ064V3DG01
- VGA
- EDT28

- LQ070Y3DG3B1

In order to change the selected display without recompiling U-Boot, each display must have support enabled. See in topic 10.4.1.

The default location of the frame buffer is at the end of U-Boot in RAM. The location can be changed by setting:

```
#   setenv fb_base <address>
```

The default is also used if the passed address for the frame buffer is inside of a protected area (for example if the address points to the U-Boot code).

Platforms cc9m2443 and ccw9m2443 always initialize the video interface, regardless of the value of 'videoinit' variable or the existence of a splash partition.

11. U-Boot development

U-Boot is an open source project. Sources are freely distributed, and you can modify them to meet your requirements for a boot loader.

The project sources are ready to be installed and compiled in a Linux environment. If you do not have a Linux machine for development, you can install the *Cygwin X-Tools* software (http://www.cygwin.com). 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.

12. Troubleshooting

12.1 Work with FIMs on NS921x based modules

Modules using NS921x processors can use the FIMs in the processor as UART and/or SD controller.

When using the FIMs, instruction cache must be enabled, otherwise they may malfunction.

Instruction cache is enabled by default in U-Boot, so do not disable it when using the FIMs.

12.2 Problems booting from NFS

When booting or updating directly from NFS, sometimes U-Boot can experience problems mounting or unmounting the NFS server. To work around this problem, edit the file **/etc/hosts** of the host PC serving the NFS path and add a line with the IP address of the target.

```
192.168.42.40      mytarget
```

12.3 USB memory sticks

U-Boot might experience problems detecting certain USB memory sticks due to different response timings.