U-Boot v2009 Reference Manual

90001129_B

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)
☎ +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*).

```
U-Boot 2009.08 - DUB-1.0  - (Mar 17 2010 - 18:01:12) - GCC 4.3.2
for ConnectCore Wi-i.MX51 on a JSK Development Board

I2C:   ready
NAND:  512 MB
DRAM:  512 MB
MMC:   FSL_ESDHC: 0, FSL_ESDHC: 1
In:    serial
Out:   serial
Err:   serial
Net:   FEC0 [PRIME], smc911x-0
Hit any key to stop autoboot:  0
CCWMX51 #
```

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 will see a list similar to this one:

```
CCWMX51 # help
?       - alias for 'help'
autoscr - DEPRECATED - use "source" command instead
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'
bootm   - boot application image from memory
bootp   - boot image via network using BOOTP/TFTP protocol
cmp     - memory compare
coninfo - print console devices and information
cp      - memory copy
crc32   - checksum calculation
date    - get/set/reset date & time
dboot   - Digi modules boot commands
dcache  - enable or disable data cache
dhcp    - boot image via network using DHCP/TFTP protocol
echo    - echo args to console
envreset- Sets environment variables to default setting
erase_pt- Erases the partition
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.
fuse    - Fuse sub system
go      - start application at address 'addr'
help    - print online help
i2c     - I2C sub-system
icache  - enable or disable instruction cache
iminfo  - print header information for application image
imxtract- extract a part of a multi-image
intnvram- displays or modifies NVRAM contents like IP or partition table
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
md      - memory display
mii     - MII utility commands
mm      - memory modify (auto-incrementing address)
mmc     - MMC sub system
mmcinfo - mmcinfo <dev num>-- display MMC info
mtest   - simple RAM read/write 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
pmic    - pmic register access
printenv- print environment variables
printenv_dynamic- Prints all dynamic variables
rarpboot- boot image via network using RARP/TFTP protocol
reboot  - Perform RESET of the CPU
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
source  - run script from memory
sspi    - SPI utility commands
tftpboot- boot image via network using TFTP protocol
update  - Digi modules update commands
usb     - USB sub-system
usbboot – boot from USB device
version - print monitor version
CCWMX51 #
```

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'*
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 these 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 a number or a 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.

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.
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.
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).
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).

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 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 section 9.2).
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'*.

4.2.8 E-Fuses/OTP bits

The i.MX51, i.MX53, and i.MX28 processors have a special feature: the E-Fuses (OTP or One Time Programmable bits on i.MX28). E-Fuses are electrically **one-time erasable** fuses that allow the user to store some bits of information, like a MAC address or an IMEI number.

On the i.MX51/i.MX53, storage space is divided into 4 banks (0..3), each of them containing rows (for the detailed fuse map, see Chapter 6 in the i.MX51/i.MX53 documentation.) Each row is 1 byte. To read or write fuses, you have to select the appropriate row of a bank. Only one row is readable or writable at a time.

On the i.MX28, storage space is divided into 32-bit words, with a total number of 40 (for the detailed OTP map, see Chapter 20 On-Chip OTP of the i.MX28 documentation).

It is possible to write only ones. Writing a 1 means erasing a fuse. Writing zeros is **not** possible because erased fuses cannot be restored. For example: if the IMEI's first byte was set previously to 0x10, it's possible to change it to 0x30, but not to 0x00.

Fuse rows (or OTP words) may be locked by setting a single fuse called 'fuse lock' (see Table 6-1 in Chapter 6 in the i.MX51/i.MX53 documentation, or see the word LOCK at offset 0x10 in the i.MX28 documentation). If a certain fuse row (or OTP word) is locked, no further writes are allowed to that row (or word).

To read, write and lock the E-Fuses, use the following commands:

For i.MX51/i.MX53

Command	Description
iim read <bank> <row>	Read out a fuse row
iim blow <bank> <row> <value>	Writes an hex value into a fuse row

where <*bank*> is the bank index (0..3) and <*row*> is the row index (starting at 0 in each bank).

For i.MX28

Command	Description
otp dump	Dump all OTP bits
otp read <addr> [count]	Read 'count' OTP registers (32-bits wide) starting at 'addr'
otp blow <addr> <value>	Blow OTP register at 'addr' with 'value'
otp lock <addr>	Lock OTP register at 'addr'

Be extremely careful when using this command as erased fuses cannot be recovered.
An incorrect programming of fuses might lead to a non-working module.
Not all fuses at available for customer use. Please read the CPU manual to understand which fuses you are free to use.

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 Chapter 6 to learn how to create a bootscript that automatically executes at start up.

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
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.

5.4.2 Network related variables

The following variables are related to network:

Variable	Description
ethaddr	MAC address of the FIRST wired target's Ethernet interface
eth1addr	MAC address of the SECOND wired target's Ethernet interface*
wlanaddr	MAC address of the target's WLAN interface*
btaddr	MAC address of Bluetooth adapter*
ipaddr	IP address of the FIRST wired target's Ethernet interface
ipaddr1	IP address of the SECOND wired target's Ethernet interface*
ipaddr_wlan	IP address of the target's WLAN interface*
netmask	Subnet mask of the FIRST wired target's Ethernet interface
netmask1	Subnet mask of the SECOND wired target's Ethernet interface*
netmask_wlan	Subnet mask of the target's WLAN interface*
dhcp	Whether DHCP is enabled on the FIRST wired target's Ethernet interface
dhcp1	Whether DHCP is enabled on the SECOND wired target's Ethernet interface*
dhcp_wlan	Whether DHCP is enabled on the target's WLAN interface*
ntpserverip	NTP server IP address (for getting the date/time).
gatewayip	IP address used as network gateway.
serverip	IP address of the host PC (for remote connections like TFTP transfers).
dnsip1	IP of DNS server 1
dnsip2	IP of DNS server 2
ethprime	Contains the name of the network interface to be used as PRIMARY interface
ethact	Contains the name of the currently active network interface
hostname	Contains the hostname of the device

* Not all modules have a wireless interface or a second wired network interface or a Bluetooth adapter.

5.4.3 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.

5.4.4 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 allows you to have different boot scripts, depending on the key pressed during boot. Therefore, you can boot two different kernels, such as a dual Linux/Windows CE or two versions of the same OS.

When the two keys are pressed during boot, both are detected as pressed, and the script **key12** is launched. If this variable is not defined, the scripts at variables **key1** and **key2** will be run (in this order).

If the **key1**, **key2** or **key12** variables do not exist, or if no key is pressed, the normal **bootcmd** is executed.

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 12 for information about U-Boot development.

5.4.5 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.

5.4.6 Hostname

The environment variable *hostname* contains the name by which a device can be referred to in an IPv6 enabled network. Although U-Boot does not support IPv6, having this variable in the NVRAM allows the operating system to use its value to establish the hostname when the kernel has booted.

Although the user can set the *hostname* variable to anything they want, if the variable does not previously exist its value defaults to a combination of the platform name plus the last three bytes of the first wired MAC address (variable *ethaddr*).

For example, for a ConnectCore Wi-i.MX51 platform with a MAC 12:34:56:78:9A:BC, the hostname would default to a value of **CCWMX51-789ABC**.

The auto-generated *hostname* variable is **NOT** automatically saved in NVRAM. The user must save the auto-generated value or set and save a value of their own.

It is the responsibility of the operating system to retrieve the value of the *hostname* variable from the NVRAM and set it appropriately in the system so that it is valid on the network.

6. Network interfaces

6.1 Number of interfaces

U-Boot is able to issue Ethernet communication using several interfaces. The supported network interfaces are detected by U-Boot at start and their names are printed in the start messages:

```
U-Boot 2009.08 - DUB-1.0  - (Mar 17 2010 - 18:01:12) - GCC 4.3.2
for ConnectCore Wi-i.MX51 on a JSK Development Board

I2C:   ready
NAND:  512 MB
DRAM:  512 MB
MMC:   FSL_ESDHC: 0, FSL_ESDHC: 1
In:    serial
Out:   serial
Err:   serial
Net:   FEC0 [PRIME], smc911x-0
Hit any key to stop autoboot:  0
CCWMX51 #
```

The network interfaces are given the name of the driver plus an index number starting at 0. In the example above two network interfaces have been detected: **FEC0** and **smc911x-0**.

Although U-Boot can route communications through several network interfaces it can only handle one IP address (the one specified at variable 'ipaddr').

Notice that U-Boot may or may not support all the available network interfaces, depending on configuration options. For example, the platform *ccwmx51js* only has support for the FEC0 interface, by default. To add support for the second wired Ethernet interface you need to enter the configuration tool and enable it. Please refer to your OS development environment for information about U-Boot configuration.

6.2 Primary network interface

By default, U-Boot will route Ethernet communications through the interface specified at environment variable **ethprime**. Such interface will be marked as [PRIME] in the boot messages. If this fails, U-Boot will try with the following available network interface (using the same IP address specified in 'ipaddr').

6.3 Active network interface

U-Boot automatically sets the variable **ethact** to the name of the Ethernet interface that is currently active. This variable can be changed on hot to force U-Boot to use a different network interface.

This variable is not persistent between resets. U-Boot will set this variable to the interface that becomes active after reset.

7. Bootscript

The bootscript is a 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.

7.1 Bootscript process

The bootscript works in the following way:

1. U-Boot checks the variable **loadbootsc**. If set to "no", it continues its 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 the script is cancelled.
5. When the bootscript has been fully executed (or cancelled) U-Boot continues normal execution.

7.2 Creating a bootscript

To create a bootscript file:

1. Create a plain text file with the sequence of U-Boot commands. 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 Core Wi-MX51 platform, go to the U-Boot directory and execute:

```
$   tools/mkimage -T script -n "Bootscript" -C none -d myscript.txt
    ccwmx51js-bootscript
```

7.2.1 Creating a bootscript in Windows

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

7.2.1.1 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.

7.2.1.2 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*.

7.3 Configuration for launching the bootscript

Once the bootscript file has been created two more steps are needed to allow the target to 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 all steps have been completed, power up the target. It will connect to the host PC and will download the bootscript to RAM; execute it, and continue booting as usual.

7.4 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 section 9.2 for more information).

8. Boot commands

8.1 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, USB, MMC, flash) into RAM
- Jumping to the first instruction of the image in RAM

8.2 Reading images into RAM

8.2.1 From Ethernet

The most common way to boot an image during development is by transferring it using TFTP over the Ethernet interface. This can be done 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.

8.2.2 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* is given as a number starting at zero (0, 1, 2...) and *partition* is given as a number starting at 1 (1, 2, 3...). If no partition is specified, partition number 1 is assumed.

8.2.3 From SD/MMC card

If the target has an MMC or HSMMC (High Speed MMC) interface U-Boot can also read from it. The SD/MMC card must contain a partition formatted in the FAT file system.

Make sure the SD/MMC card is partitioned before formatting. Windows systems allow formatting of a card without partitions, but this won't work in U-Boot.

To read an image from an SD/MMC card's partition formatted in FAT, enter:

```
#   mmc rescan <dev>
#   fatload 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.

*For slow SD/MMC cards, a special variable **slowmmc** has been created. If set to **yes** the MMC subsystem will do special delays to be able to communicate with such cards.*

Due to a limitation with FAT implementation in U-Boot, when using SD/MMC cards it is recommended that the kernel image is stored in one partition which does not have many files, and have the root file system in a different partition.

8.2.4 From SATA disk

Some platforms, like the ConnectCore for i.MX53, support SATA drives also. To boot from a SATA disk it must first be formatted either in FAT, ext2, or ext3 file system.

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

```
#   sata init
#   fatload sata <dev>[:partition] <loadAddress> <bootfilename>
```

If the disk is formatted in ext2/ext3:

```
#   sata init
#   ext2load sata <dev>[:partition] <loadAddress> <bootfilename>
```

This command reads file *bootfilename* from device *dev*, partition *partition* of the SATA disk into the RAM address *loadAddress*. *Device* is given as a number starting at zero (0, 1, 2...) and *partition* is given as a number starting at 1 (1, 2, 3...). If no partition is specified, partition number 1 is assumed.

8.2.5 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*.

8.3 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.

8.4 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:

```
#   help dboot
dboot - Digi modules boot commands

Usage:
dboot <os> [source] [extra-args...]
 Description: Boots <os> via <source>
 Arguments:
   - os:          a partition name or one of the reserved names:
                  linux|android|wce|netos|eboot
   - [source]:    tftp (default)|flash|nfs|usb|mmc|hsmmc|sata|ram
   - [extra-args]: extra arguments depending on 'source'

      source=tftp|nfs -> [filename]
       - filename: file to transfer (required if using a partition name)

      source=usb|mmc|hsmmc|sata -> [device:part filesystem] [filename]
                                   [rootfspart]
       - device:part: number of device and partition
       - filesystem: fat|vfat|ext2|ext3
       - filename: file to transfer
       - rootfspart: root parameter to pass to the kernel command line.
                     If omitted uses the one at variable usb_rpart|
                     mmc_rpart|sata_rpart.

      source=ram -> [image_address] [initrd_address] [initrd_max_size]
       - image_address: address of image in RAM (default: linuxloadaddr,
           netosloadaddr, etc)
       - initrd_address: address of initrd image (default: loadaddr_initrd)
       - initrd_max_size: max. allowed ramdisk size (in kB) to pass to
           the kernel (default: kernel default)

If <os> is 'wce' the following bootargs are possible:
    cleanhive
```

where

- *OS* is either **linux, android, wce, eboot, netos** or any partition name that holds an OS.
- *source* is either **flash, tftp, nfs, usb, mmc, hsmmc**, or **sata**.
- *dev[:partition]* is the device index (only for USB, MMC media and SATA) 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**, **vfat**, **ext2**, or **ext3** (only for USB, MMC media and SATA) and must match the file system of the partition that holds the image to boot. If not provided, FAT is assumed.
- *filename* is the name of the kernel image file to download and boot. If not provided, the filename is taken from ‘kimg’ variable for Linux, 'aimg' for Android, or ‘wimg’ for Windows CE.
- *rootfs_partition* can be:
    - *The name of the partition to use as root file system if booting from Flash.*
    - *The name of the device to mount as root file system if booting from USB, MMC,* HSMMC, or SATA.

- Other available options are (only for WindowsCE kernels):
  - *cleanhive*: Cleans the WindowsCE registry on boot.

*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.*

*For slow SD/MMC cards, a special variable **slowmmc** has been created. If set to **yes** the MMC subsystem will do special delays to be able to communicate with such cards.*

USB media is not supported in ConnectCore Wi-i.MX53 and ConnectCore i.MX53 platforms.

Due to a limitation with FAT implementation in U-Boot, when using SD/MMC cards it is recommended that the kernel image is stored in one partition which does not have many files, and the root file system in a different partition.

To boot from NFS it is required that the target IP is added to the /etc/hosts file in the computer serving the NFS.

8.4.1 Boot examples

Boot default Windows CE image stored in Flash:

```
#   dboot wce flash
```

Boot Windows CE image for ConnectCore Wi-MX51 from TFTP cleaning the registry:

```
#   dboot wce tftp wce-CCXMX51 cleanhive
```

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

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

Boot Linux image stored in Flash and use partition with name "MyRootFS" for mounting the root file system:

```
#   dboot linux flash MyRootFS
```

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

8.5 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, you want to automatically boot a WinCE image from TFTP server, set **bootcmd** as follows:

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

Or, if you want to automatically boot a Linux image from flash, set **bootcmd** as follows:

```
#   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.

9. 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 this data or data becomes corrupted, the information can be restored from the mirrored data.

9.1 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.

9.1.1 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/Android-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 • UBIFS • CRAMFS • SQUASHFS • INITRD

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

9.1.2 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
NrNr | 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 |     24 MiB | WinCE-Kernel |       |
 3 | Filesys |  25856 KiB | 498432 KiB | Filesystem   | ExFAT |
```

You can add, modify, or delete partitions step-by-step; a 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 can also be reset to default values. In this case, because the partition table differs according to the target's OS, you must 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.

9.2 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.

The syntax of the **intnvram** command is as follows:

```
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 one parameter or a set of parameters. Parameters are grouped in blocks. The following is a complete list of parameters with the possible values some of them can take:

```
  params for "set" or "print" can be
     module    [producttype=] [serialnr=] [revision=] [patchlevel=]
               [ethaddr1=] [ethaddr2=] [ethaddr3=] [btaddr1=]
     network   [gateway=] [dns1=] [dns2=] [server=] [netconsole=] [ip1=]
               [netmask1=] [dhcp1=] [ip2=] [netmask2=] [dhcp2]
               [ip3=] [netmask3=] [dhcp3=]
     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=]
     dualboot  [avail0=] [avail1=] [verified0=] [verified1=] [boot_part=]
               [last_updated=] [boot_attempts=] [mode_peer=]
               [guarantee_perboot=]

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,NET+OS-Loader,User defined,
                   Wireless Calibration
```

```
  partition type:  U-Boot,NVRAM,FPGA,Linux-Kernel,WinCE-EBoot,WinCE-Kernel,
                   Net+OS-Kernel,Filesystem,WinCE-Registry,Unknown,
                   Splash-Screen,NET+OS-Loader,NET+OS-NVRAM
  flag_fs_type:    None,JFFS2,CRAMFS,INITRD,FlashFX,Unknown,YAFFS,
                   ExFAT,SQUASHFS,ROMFS
```

Specify the group of the parameter before the parameter itself. For example, to print the module IP for the first 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.

9.2.1 Mappings of variables

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

U-Boot variable	NVRAM parameter	Blocked for 'setenv'
ethaddr	ethaddr1	X
wlanaddr	ethaddr2	X
eth1addr	ethaddr3	X
btaddr	btaddr1	X
netmask	netmask1	
netmask_wlan	netmask2	

netmask1	netmask3	
ipaddr	ip1	
ipaddr_wlan	ip2	
ipaddr1	ip3	
dhcp	dhcp1	
dhcp_wlan	dhcp2	
dhcp1	dhcp3	
dnsip	dns1	
dnsip2	dns2	
serverip	server	
gatewayip	gateway	

10. Firmware update commands

10.1 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:

1. Reading image from media (Ethernet, USB, MMC, SATA) into RAM memory
2. Erasing the flash that is to be updated
3. Copying the image from RAM into flash

10.2 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 8.2 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.

10.3 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.

The following is the syntax for **update**:

```
#   help update
update - Digi modules update commands

Usage:
update <partition> [source] [extra-args...]
 Description: updates flash <partition> via <source>
 Arguments:
   - partition:    a partition name or one of the reserved names:
                   uboot|linux|android|rootfs|userfs|androidfs|eboot|wce|
                   wcez|netos|netos_loader|splash
   - [source]:     tftp (default)|nfs|usb|mmc|hsmmc|sata|ram
   - [extra-args]: extra arguments depending on 'source'

      source=tftp|nfs -> [filename]
       - filename: file to transfer (required if using a partition name)

      source=usb|mmc|hsmmc|sata -> [device:part filesystem] [filename]
       - device:part: number of device and partition
       - filesystem: fat|vfat|ext2|ext3
       - filename: file to transfer

      source=ram -> <image_address> <image_size>
       - image_address: address of image in RAM
       - image_size: size of image in RAM
```

- *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, SATA 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, SATA 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
    - ***aimg***: for the Android 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
    - ***arimg***: for the Android root file system image
    - ***simg***: for the Splash screen image
    - ***fimg***: for the FPGA 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.*

To boot from NFS it is needed so that the target IP is added to the /etc/hosts file in the computer serving the NFS.

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

```
#   update splash mmc 0:2 fat 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
```

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

To overcome this problem a special TFTP on-the-fly update mechanism has been created and is used only when updating from TFTP media.

10.3.2 TFTP on-the-fly update mechanism

This update mechanism solves the RAM limitation problem with firmware updates. It first erases the target flash partition and then transfers a chunk of the image file via TFTP. It writes the chunk into the flash memory, verifies it and then continues to transfer the next chunk.

The on-the-fly mechanism is disabled by default and can be enabled by setting the environment variable **otf-update** to **yes**.

```
#   setenv otf-update yes
#   saveenv
```

This mechanism is only recommended to write large image files (bigger than 32MB or than half the available RAM).

On-the-fly update mechanism is automatically disabled when updating U-Boot partition (to prevent partial update due to a network cut).

To manually disable the mechanism, simply set the variable to **no**:

```
#   setenv otf-update no
#   saveenv
```

11. Customize U-Boot

11.1 Overview

U-Boot has several 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 are as follows:

11.2 Silent Console

The target does not display any output when the console is set to silent mode. This option is disabled by default. To enable it, you must configure U-Boot using your OS configuration tool and rebuild it.

When silent console is enabled messages will be printed depending on the value of the environment variable 'silent' (yes|no).

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 recovery possibility is using the keyX environment variables (see section 5.4.4) to set the 'silent' variable to 'no':

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

In this case, the sequence to recover from silent mode would be 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 or a user button on the JumpStart board. This method can also be enabled in the configuration tool.

If a recover method is not defined you won't be able to access U-Boot prompt and you will need to recover it.
Please refer to Chapter *14 Recovering a device* for more information.

The console can be switched to silent mode after reboot by typing:

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

After reboot, you can recover from silent mode by pressing the user key (or setting the GPIO to the defined level) shortly after the target starts to boot.

11.3 Video interface

11.3.1 Initialize video interface

To initialize the video interface in U-Boot, the variable 'video' must be set to a value like this:

```
#   setenv video displayfb:<DISPLAYTYPE>[@[<DISPLAYNAME>][<RESOLUTION>]]
#   saveenv
```

where

- <DISPLAYTYPE> is the type of display: VGA, HDMI, LCD
- <DISPLAYNAME> is the name of the connected display (only for LCD displays)
- <RESOLUTION> is the display resolution (not needed for LCD displays, as those have fixed resolution).

If the hardware is capable of a second display variable 'video2' must be set with the video settings of the second display. Some platforms support LVDS video bus through a bridge, which needs to be configured by means of the environment variable 'ldb'.

To simplify things, the command **'video'** allows you to interactively set the 'video', 'video2' and 'ldb' variables. A menu will present the available displays and resolutions (where applicable):

```
#   video
Video interface
  1) Video 1
  2) Video 2
  3) LVDS bridge
  4) Primary display
  5) Cancel
Select interface: 1

Displays/video mode
  1) Disabled
  2) VGA
  3) HDMI
  4) LCD display
  5) Cancel
Select displays/video mode: 4

Type of LCD
  1) Parallel
  2) LVDS
  3) Cancel
Select LCD type: 1

LCD display
  1) LQ070Y3DG3B (800x480 JumpStart Kit display)
  2) LQ064V3DG01 (640x480)
  3) LQ121K1LG11 (1280x800)
  4) LQ106K1LA05 (1280x768)
  5) LQ104V1DG62 (640x480)
  6) custom1 configuration
  7) custom2 configuration
  8) custom1 configuration in NVRAM
  9) custom2 configuration in NVRAM
  a) Cancel
Select LCD display: 1

Setting variable:
  video=displayfb:LCD@LQ070Y3DG3B
Remember to save the configuration with 'saveenv'

#   saveenv
```

*Remember to save the changes with **saveenv** after configuring the display with the **video** command.*

Refer to your CPU hardware reference manual for details about the configuration of the LVDS Bridge, if you want to select this display option.

11.3.2 Custom LCD displays

Apart from the built-in displays and resolutions, a user can select two custom configuration options (**custom1** or **custom2**) to instruct the OS to boot with a custom video configuration. For LCDs, it means to use a custom LCD display. For video modes (VGA/HDMI) it means to configure other settings like resolution, refresh frequency, etc.

Refer to your OS documentation for instructions on configuring a custom display in source code.

For LCD displays, there are two additional options: the **custom3/custom4 configuration in NVRAM**. While the previous custom1 and custom2 configurations were hard-coded (which means, the LCD settings are established in source code and compiled with the kernel), the *custom3_nv* and *custom4_nv* settings are stored in the NVRAM, so you can modify these at runtime.

To select and change the *custom3_nv* or *custom4_nv* LCD settings, run the **'video'** command, select the LCD interface then the **'custom3** or **custom4 configuration in NVRAM'** option. Now, U-Boot will ask for the LCD configuration parameters:

```
LCD display
  1) LQ070Y3DG3B (800x480 JumpStart Kit display)
  2) LQ064V3DG01 (640x480)
  3) LQ121K1LG11 (1280x800)
  4) LQ106K1LA05 (1280x768)
  5) LQ104V1DG62 (640x480)
  6) custom1 configuration
  7) custom2 configuration
  8) custom3 configuration in NVRAM
  9) custom4 configuration in NVRAM
  a) Cancel
Select LCD display: 8
  Refresh rate (60): 60
  X-resolution (800): 800
  Y-resolution (480): 480
  Pixel clock (44000): 44000
  Left margin (0): 0
  Right margin (50): 50
  Upper margin (25): 25
  Lower margin (10): 10
  H-sync (128): 128
  V-sync (10): 10
  Sync (? for help) (4): 4
  V-Mode (? for help) (0): 0
  Flags (0): 0
  Backlight function enabled (0): 0

Windows CE specific settings:
  Pixel Data Offset [0]:
  Pixel Clock Up [0]:
  Pixel Clock Down [0]:

Setting variable:
  video=displayfb:LCD@custom3_nv
Remember to save the configuration with 'saveenv'

#   saveenv
```

The values of the previous example are the display settings of the LQ070Y3DG3B.

For 'Sync' and 'V-Mode' parameters, you can type a '?' to get help about the valid input values.

11.3.3 Selecting the primary video interface

In platforms that support two video interfaces, you may use the variable **fbprimary** to determine which video interface will be the primary interface.

Use the **video** command to set this variable from its menu:

```
#   video

Video interface
  1) Video 1
  2) Video 2
  3) LVDS bridge
  4) Primary display
  5) Cancel
Select interface: 4

Primary display
  1) Video 1
  2) Video 2
  3) Cancel
Select primary display: 1

video=displayfb:LCD@custom3_nv
video2=displayfb:disabled
fbprimary=video
#   saveenv
```

The variable **fbprimary** can take the values:

- **video**, in which case display1 will be primary and display2 will be secondary.
- **video2**, in which case display1 will be secondary and display2 will be primary.

By default, the **fbprimary** variable is not set and display1 is considered the primary.

11.4 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.

The usage of splash screen support in U-Boot is discouraged and will be deprecated in the future.
Instead, it is recommended to use whatever mechanism the Operating System has, to display a splash image on the frame buffer soon during the system boot process.

In particular, the ConnectCard for i.MX28 platform doesn't support splash screen in U-Boot.

11.4.1 Creating a splash partition

Use the command **flpart** and option **p** to print the partition table and check if there is already a splash partition in your flash. If there isn't you will need to create one to hold the splash screen image (you may need to reduce some existing partition):

```
#  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
```

11.4.2 Uploading a splash image

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

Uploading is done with the 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
```

where *Splash* is how we named the partition. Or, if the variable **simg** contains the BMP filename, you can simply execute:

```
# update splash
```

and the keyword *splash* will auto-select the first partition of type Splash-Screen.

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.

11.4.3 Initialize video interface

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

```
#   setenv videoinit yes
#   saveenv
```

Additionally, the variable *video* must be initialized with the selected display, as described in section *11.3. Video interface*.

The frame buffer is, by default, located 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).

11.5 Dual boot mechanism

11.5.1 Overview

Dual boot is a mechanism that allows you to store an alternate system **that will be booted if the current system turns invalid or unbootable** for whatever reason within Flash.

The dual boot mechanism should not be confused with the ability to boot multiple different operating systems which is explained in chapter 12. Boot different operating systems.

When the mechanism detects that a system partition is not booting correctly it switches automatically to boot the alternate one.

IMPORTANT NOTES

- The dual boot mechanism is not enabled by default. Refer to your OS documentation for information about configuring U-Boot for enabling dual boot.
- The dual boot mechanism is devised to work when booting from flash memory only.
- Enabling dual boot requires you to modify your flash partition table, maintain two systems, program actions from user space and usually properly handle your watchdog timer from your OS.

11.5.2 Enabling Dual boot mechanism

Run your OS configuration tool and enable the dual boot mechanism on the U-Boot configuration section. You can select other options like the dual boot mode and the usage of the watchdog (explained in detail further in this chapter). Save the settings, rebuild U-Boot, and update the U-Boot firmware into your platform.

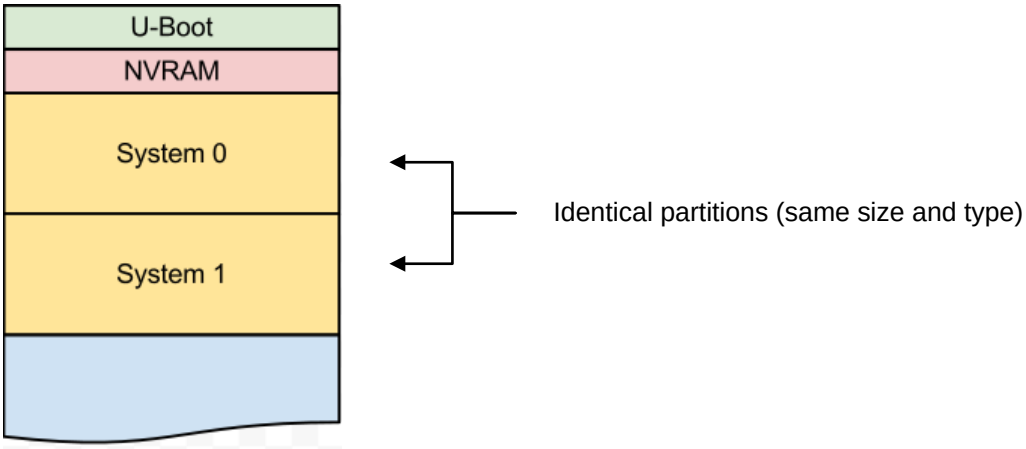
11.5.3 Dual boot modes

Dual boot can work in two modes (selectable at build time):

- Peer
- Rescue

11.5.3.1 Peer mode

Dual boot **peer** mode requires you to have duplicated identical system partitions that will be used to store two exact copies of the system.



The term *system partition* means as many partitions as needed to have a bootable OS, for example:

- For Linux or Android the system is formed by two partitions: Kernel and RootFS. This means we need to duplicate these two partitions in order to have two systems: Kernel0+RootFS0 and Kernel1+Rootfs1 (four partitions in total).
- For WindowsCE the system is only formed by one partition: Kernel. This means that we need to duplicated this partition to have two systems: Kernel0 and Kernel1 (two partitions in total).

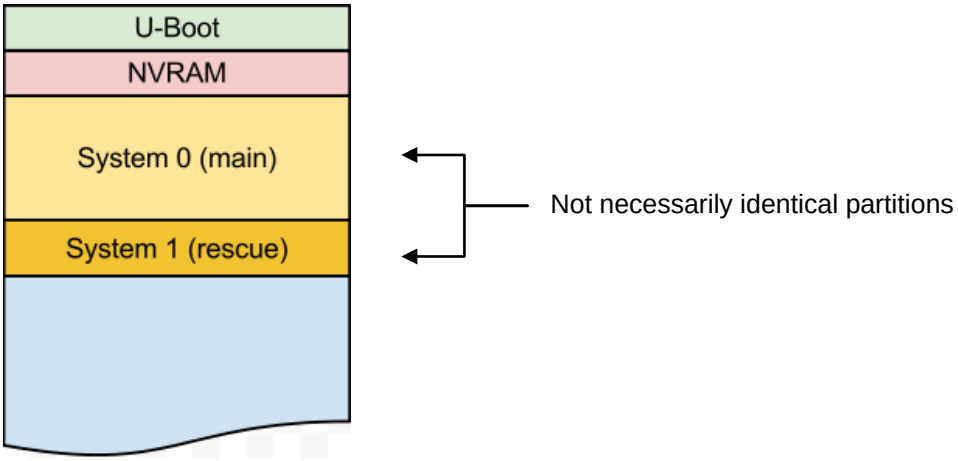
*Once you have flashed and booted a U-Boot with dual boot support you can run the **flpart** command and **reset** your flash partition table to your OS defaults. Doing this will automatically create the necessary duplicate system partitions.*

11.5.3.2 Rescue mode

The reason for utilizing dual boot **rescue** mode is to have an secondary rescue system partition that holds a well-verified bootable system that can serve for restoring the main system if it becomes defective.

Note that the rescue system does not necessarily need to have the complete functionality designed for the device. Instead, it has to have the necessary tools to allow for main system recovery.

As in peer mode, we need to have additional partition(s) to hold a secondary system, but these secondary partition(s) don't need to be identical.



*Once you have flashed and booted a U-Boot with dual boot support you can run the **flpart** command and **reset** your flash partition table to your OS defaults. Doing this will automatically create the necessary duplicate system partitions.*

11.5.4 Detecting a corrupt system

A system may become unstable for different reasons. The dual boot mechanism must automatically detect such a case and instead boot the alternate system. This is achieved by using a watchdog timer and allowing users to decide when they consider a system as being fully booted.

11.5.4.1 Marking a system as valid

The dual boot mechanism maintains a counter called *boot_attempts* with the number of attempts a system has been tried to boot. Every time the system boots, this counter increments automatically. If it reaches the established maximum number of attempts without the system having been marked as valid, the dual boot mechanism will mark that system as unavailable and will switch to boot the alternate system in the flash.

Marking a system as valid is the responsibility of the user. It is up to the user to decide under which conditions a system is considered valid. The simplest way to do this is by marking the system as valid right after the system starts (for example, in a script that runs after boot). However, the user may decide to delay this to the point where a certain application has launched and started different services.

Depending on the selected dual boot guarantee mode, the way to mark a system as valid varies:

Guarantee mode	Description	How to mark the system as valid
Per boot	The system must be verified to be working (valid) on every single boot.	The *boot_attempts* counter works as a kind of watchdog timer, and the way to mark the system as valid is to reset the boot_attempts counter.
After update	The system must be verified to be working (valid) only after it has been updated with new firmware. Once marked valid, the boot_attempts counter is no longer incremented and the dual boot mechanism will assume the system will always boot correcly.	In this mode, a 'verified' flag for each system is stored in the NVRAM partition. The way to mark the system as valid is to set the system's verified flag in the NVRAM

If a system is not able to reach this point (defined by the user) for example, because the kernel hangs during the boot process, the *boot_attempts* counter will not be reset (or the verified flag will not be set), meaning an unsuccessful boot attempt occurred.

The way to mark a system as valid is hardware and OS dependent. Please refer to your OS documentation for information about how to mark a system as valid in your system.

11.5.4.2 Maximum boot attempts and switch to alternate system

The dual boot mechanism will try to boot a system a configurable number of times (three attempts by default). If the *boot_attempts* counter has not been reset by the OS in any of these attempts and it reaches the maximum number of attempts, on the next boot cycle U-Boot will switch and try the alternate system instead.

11.5.5 Start up guarantee modes

The dual boot mechanism can be configured to guarantee a valid system in two ways:

- Per boot
- After update

11.5.5.1 Per boot start up guarantee mode

When **per boot** start up guarantee is selected, the *boot_attempts* counter is incremented on every attempt to boot the system. This model is important because it guarantees that the device boots every time to a state where the system is usable (as defined by the user).

This is the recommended mode if the platform has persistent RAM of any kind available to store the *boot_attempts* counter.

11.5.5.2 After update start up guarantee mode

When **after update** start up guarantee is selected, the *boot_attempts* counter is incremented on every attempt to boot the system until the system has been marked as valid. A *verified* flag must be set to mark a system as valid in this mode. Once this mark has been made, the system is assumed to always be valid and the dual boot mechanism is not used again until the system partition is updated with new firmware, in which case the *verified* flag is automatically reset.

This model only guarantees that a system boots the first time after new firmware has been put into the flash.

This is the recommended mode if NVRAM needs to be used for persistent data storage, to prevent wear of the flash (in *per boot* mode the *boot_attempts* counter must be changed on every boot).

11.5.6 Watchdog to reset a corrupt system

By default U-Boot won't enable the watchdog timer. This means that you will have to manually reset your device in case of a failed boot attempt.

Enabling the watchdog for the dual boot mechanism is a configurable option. If you enable it, U-Boot will program the watchdog timer of your platform with the value specified in variable **dualb_wdt_timeout** (in seconds) and will start it right before booting the OS.

IMPORTANT NOTES:

- The target will reset in the time specified in the U-Boot variable **dualb_wdt_timeout**, unless the OS takes control of the watchdog timer (either disabling it, or periodically kicking it before it expires).

- Make sure you set the U-Boot variable **dualb_wdt_timeout** to a time big enough for your system to safely reach the point where you are marking the system as valid (reset of the *boot_attempts* counter or setting of *verified* flag) and taking control of the watchdog.
- Enabling the watchdog is the only way the device can automatically reset and switch to boot the alternate system after the specified number of attempts. Otherwise, a manual reset is required.

11.5.7 Update process in U-Boot

When using the **update** command in U-Boot with the partition key reserved names (linux, wce, rootfs) the following behavior will apply:

11.5.7.1 Update on peer mode

U-Boot will boot the partition that was updated most recently (the *current partition*). Updates will take place over the alternate partition so that if an update process crashes, the current system is not affected. The partition to update will be automatically selected by U-Boot.

If the update process completes correctly, the updated partition will become the *current partition* for the next boot attempt.

11.5.7.2 Update on rescue mode

In rescue mode there is only one main system partition and updates take place over this partition.

When specifying the exact name of the partition (as opposed to the abbreviated key name) that particular partition will be updated.

11.5.8 Additional notes on dual boot

- The user can specify the maximum number of boot attempts to try before toggling to boot the alternate system by means of the environment variable *dualb_retries*. Notice that in some platforms, the implementation of the *boot_attempts* may already fix a limit (for example in ConnectCard for i.MX28 two bits are used for the counter, fixing the limit on 3 attempts).
- When a system has failed to boot the number of times specified in *dualb_retries* this system is marked as invalid, so that it is never tried again (until a new firmware is written).
- Before booting the OS, U-Boot prints a message to the console with the attempt number and the maximum number of retries. If U-Boot toggles to the alternate system a message is also printed.
- A configuration option allows the user to choose what to do when the alternate system also does not succeed in fully booting. The default action would be to retry this system forever but you can instead call a *no_system_panic* function that simply prints a message to the console or other hardware specific action (like blinking a LED, for example).

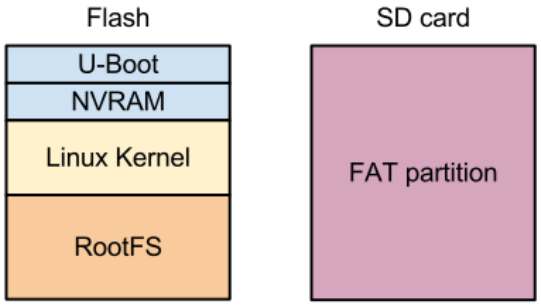
12. Boot different operating systems

U-Boot allows for several different customizations, one of which is the possibility to boot multiple different operating systems. This can serve for demo or operating purposes.

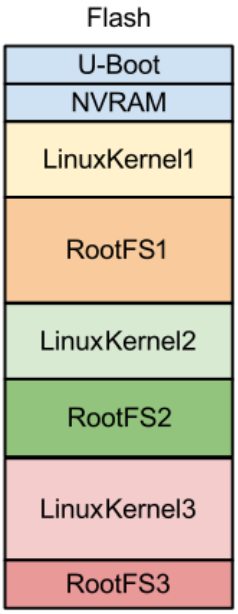
12.1 Media requirements

In order to boot different operating systems, the OS images must be in the appropriate media.

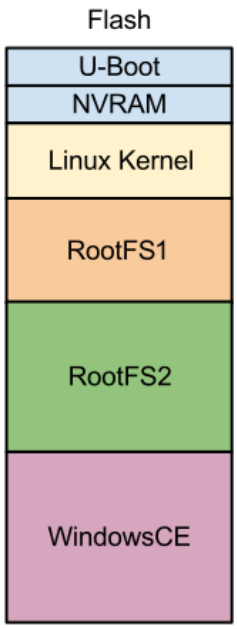
Example 1: if you wish to have Linux on the Flash and WindowsCE on an MMC card, you must make sure that the Flash is partitioned to have a Kernel and RootFS partition and that the MMC card is formatted in FAT:



Example 2: if you wish to have three different Linux systems on the flash you will need to create a partition table in flash that contains three kernel partitions, plus three rootfs partitions:



Example 3: if you wish to have two different Linux systems on the flash with the same kernel but different root file systems, plus a WindowsCE system, you will need to create a partition table that contains one kernel partition, two root file system partitions and one WindowsCE partition.



For instructions on how to partition the flash memory see *9.1.2. Changing the partition table*.

Once the various media options are prepared you will need to transfer the different firmware images to the respective media (see Chapter *10 Firmware update commands* for more information).

12.2 Booting the different operating systems

12.2.1 Boot commands

To manually boot each system, you can use the **dboot** command with appropriate parameters.

For **example1** (described in the previous section):

- **dboot linux flash** would boot the Linux system on the flash
- **dboot wce mmc** would boot the WindowsCE system on the SD card

For **example2** (described in the previous section):

- **dboot LinuxKernel1 flash RootFS1** would boot the first Linux system on the flash
- **dboot LinuxKernel2 flash RootFS2** would boot the first Linux system on the flash
- **dboot LinuxKernel3 flash RootFS3** would boot the first Linux system on the flash

For **example3** (described in the previous section):

- **dboot linux flash RootFS1** would boot the Linux system on the flash using RootFS1 as root file system partition
- **dboot linux flash RootFS2** would boot the Linux system on the flash using RootFS2 as root file system partition
- **dboot wce flash** would boot the WindowsCE system on the flash

*For more information on the **dboot** command see 8.4. Direct booting or run **help dboot**.*

12.2.2 Choosing the operating system to boot

By default, U-Boot will run the command stored at the variable **bootcmd**. You must set this variable to the boot command that boots your desired default system.

To boot a different system than the default one you have the following options:

- **Manual boot**: stops the autoboot process and manually rusn the boot command to boot a different system.
- **User keys**: as described in *5.4.4. User keys*, you can program variables **key1**, **key2**, and **key12** to run different commands when those keys are pressed (only on JumpStart development boards that have such user keys).

13. 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.

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.

14. Recovering a device

There may be occasions where a custom boot loader is built and then updated into flash. If this custom boot loader is not able to boot, or if the boot loader is accidentally erased from the Flash, there are still methods to recover your platform.

14.1 Using Digi J-Link

The Digi J-Link Debugger is a hardware debugger that can be used to load a boot loader to your target's RAM.

Please refer to your OS documentation for additional help to recover a device via JTAG.

14.2 Loading a boot loader via USB

ConnectCard for i.MX28 platforms are able to load a boot loader image using a USB cable and a special program called **sb_loader**.

The cable must be a USB A type connector to Micro-USB B connector.

Connect the USB A type to a USB host port of your computer and the Micro-USB B connector to the development board. Notice that the USB provides 5V power to the board and module.

The **sb_loader** program is located at the toolchain path, typically at:

/usr/local/DigiEL-X.Y/x-tools/arm-unknown-linux-gnueabi/bin/

You need to run this application passing as argument the boot loader image you want to load. Notice that in order to have access to the USB port, you may be required to run the program as root:

```
$   sudo /usr/local/DigiEL-X.Y/x-tools/arm-unknown-linux-
    gnueabi/bin/sb_loader u-boot-ccardwmx28js-ivt.sb
    Waiting for USB device attach...
```

At this point, you must force the CPU to boot in USB recovery mode:

- In development board V1 (PCB S/N 30013792-01) maintain pressed the USB0 BOOT button on the development board and reset the target.
- In development board V2 (PCB S/N 30013792-02) set the boot mode jumpers to USB mode (refer to the development board Hardware Reference or schematics) and then reset the target.

The boot loader image should be loaded from USB to the RAM and executed. Once in the U-Boot prompt you can update the flash with a valid boot loader image.

14.3 Loading a boot loader from SD/MMC card

Digi platforms may be able to boot a boot loader from an SD/MMC or microSD. This is specially useful to recover targets whose boot loader in the Flash memory has been corrupted.

To make an SD/MMC card bootable we need to raw write the boot loader image to the card sectors at the position (or with the needed metadata) that the CPU expects. The following topics explain how to prepare a bootable card in a Linux or Windows host PC. The method is different depending on the target platform.

14.3.1 ConnectCore for i.MX51 and ConnectCore for i.MX53 platforms

14.3.1.1 Preparing the SD/MMC card (on a Linux host PC)

Insert the SD/MMC card (or microSD card) into your host PC card reader socket and check the identifier assigned by Linux to the card. You can usually check this by running the **dmesg** command and checking the last lines, right after inserting the card:

```
$   dmesg
[117953.837337] sdg: detected capacity change from 7969177600 to 0
[118258.629614] sd 10:0:0:0: [sdg] 15564800 512-byte logical blocks: (7.96
GB/7.42 GiB)
[118258.631335] sd 10:0:0:0: [sdg] No Caching mode page present
[118258.631340] sd 10:0:0:0: [sdg] Assuming drive cache: write through
[118258.633952] sd 10:0:0:0: [sdg] No Caching mode page present
[118258.633957] sd 10:0:0:0: [sdg] Assuming drive cache: write through
[118258.636662]  sdg: unknown partition table
```

In the example above the card was identified as **sdg**.

Do not mount the SD/MMC card. If your system automatically mounted it, umount all partitions before continuing.
Also notice that after the process, existing partitions might become unusuable.

Locate the U-Boot image file (for example /tmp/u-boot-ccwmx53js.bin) and run the following commands to raw write it to the SD/MMC card (you may need root privileges):

```
$   sudo dd if=/tmp/u-boot-ccwmx53js.bin of=/dev/sdg bs=512 seek=2 skip=2
781+1 records in
781+1 records out
400196 bytes (400 kB) copied, 0.451786 s, 886 kB/s
$   sync
```

where **/tmp/u-boot-ccwmx53js.bin** must be substituted with the path to your U-Boot image and **/dev/sdg** must be substituted with the device assigned by Linux to your SD/MMC card.

The command **sync** simply makes sure all the data is effectively transferred to the card.

After this, you can remove the card and proceed to chapter *14.3.3. Preparing the hardware platform* for further instructions.

14.3.1.2 Preparing the SD/MMC card (on a Windows host PC)

Download Windows version of **dd** program from *http://www.chrysocome.net/dd*. Install the **dd.exe** executable in C:\Windows, so that it can be accessed from anywhere.

Insert the SD/MMC card (or microSD card) into your host PC card reader. Windows will detect the new media. If the card is not formatted, a warning message will appear that the disk cannot be used unless you format it. **DO NOT FORMAT THE CARD**. An unpartitioned card is needed for the Windows version of **dd** program to be able to raw write the card.

Open a **Command Prompt** and run the following command to discover the identifier given by Windows to the SD/MMC card:

```
$   dd --list
C:\Temp>dd --list
rawwrite dd for windows version 0.5.
Written by John Newbigin <jn@it.swin.edu.au>
This program is covered by the GPL.  See copying.txt for details
Win32 Available Volume Information
\\.\Volume{6e4f119c-7aa7-11e1-a70b-806e6f6e6963}\
  link to \\?\Device\HarddiskVolume2
  fixed media
  Not mounted

\\.\Volume{e83ab88c-5e1d-11e2-8506-0004768fc2d3}\
  link to \\?\Device\HarddiskVolume3
  removeable media
  Mounted on \\.\e:

\\.\Volume{6e4f119b-7aa7-11e1-a70b-806e6f6e6963}\
  link to \\?\Device\HarddiskVolume1
  fixed media
  Mounted on \\.\c:

\\.\Volume{6e4f119f-7aa7-11e1-a70b-806e6f6e6963}\
  link to \\?\Device\Floppy0
  removeable media
  Mounted on \\.\a:

\\.\Volume{94e38262-7c99-11e1-bacd-806e6f6e6963}\
  link to \\?\Device\CdRom0
  CD-ROM
  Mounted on \\.\d:


NT Block Device Objects
\\?\Device\CdRom0
  size is 2147483647 bytes
\\?\Device\Floppy0
\\?\Device\Harddisk0\Partition0
  link to \\?\Device\Harddisk0\DR0
  Fixed hard disk media. Block size = 512
```

```
  size is 80000000000 bytes
\\?\Device\Harddisk0\Partition1
  link to \\?\Device\HarddiskVolume1
\\?\Device\Harddisk0\Partition2
  link to \\?\Device\HarddiskVolume2
\\?\Device\Harddisk1\Partition0
  link to \\?\Device\Harddisk1\DR1
  Removable media other than floppy. Block size = 512
  size is 7969177600 bytes
\\?\Device\Harddisk1\Partition1
  link to \\?\Device\HarddiskVolume3
  Removable media other than floppy. Block size = 512
  size is 7969177600 bytes

Virtual input devices
 /dev/zero   (null data)
 /dev/random (pseudo-random data)
 -           (standard input)

Virtual output devices
 -           (standard output)
781+1 records in
781+1 records out
400196 bytes (400 kB) copied, 0.451786 s, 886 kB/s
```

Locate a **Removable media other than floppy** whose size matches the one of your SD/MMC card. In the example, marked in yellow, our 8Gb card identifier is ***\\?\Device\Harddisk1\Partition0*** (ignore additional partitions on the same card).

If the card is already partitioned, the **dd** program might fail, so run the following command to remove any existing partition table in the card:

```
$   dd if=C:\Temp\u-boot-ccwmx53js.bin of=\\?\Device\Harddisk1\Partition0
rawwrite dd for windows version 0.5.
Written by John Newbigin <jn@it.swin.edu.au>
This program is covered by the GPL.  See copying.txt for details
Error writing file: 5 Access is denied
2+0 records in
1+0 records out
```

Despite the error, the command will erase any existing partition table so that you can proceed to write the boot loader.

Now run the following command to write the boot loader

```
$   dd if=C:\Temp\u-boot-ccwmx53js.bin of=\\?\Device\Harddisk1\Partition0
    bs=512 seek=2 skip=2
rawwrite dd for windows version 0.5.
Written by John Newbigin <jn@it.swin.edu.au>
This program is covered by the GPL.  See copying.txt for details
Error reading file: 87 The parameter is incorrect
790+1 records in
790+1 records out
```

The additional parameters place the boot loader two 512 bytes sectors after the start of the card, allowing you to create a partition table after the card is bootable (explained in chapter *14.3.4. Creating partitions in the bootable SD/MMC card*).

After this, you can remove the card and proceed to chapter *14.3.3. Preparing the hardware platform* for further instructions.

14.3.2 ConnectCard for i.MX28

i.MX28 CPUs have a ROM loader that expects an SB boot image file that it can load. Apart of that the media where this image resides must contain Boot Control Blocks (BCB) which allow multiple

copies of the firmware to be stored on the media. For this reason, a special formatting of the media is needed to make it bootable.

14.3.2.1 Preparing the SD/MMC card (on a Linux host PC only)

Manually doing the special format required by i.MX28 CPU can be a lengthy process. For this reason, Digi Embedded Linux distribution provides a script to generate a bootable SD/MMC card. The script is located under the scripts folder on the DEL installation folder (***/usr/local/DigiEL-X.Y/scripts/mk_mx28_sd***).

Insert the SD/MMC card (or microSD card) into your host PC card reader socket and check the identifier assigned by Linux to the card. You can usually check this by running the **dmesg** command and checking the last lines, right after inserting the card:

```
$   dmesg
[117953.837337] sdg: detected capacity change from 7969177600 to 0
[118258.629614] sd 10:0:0:0: [sdg] 15564800 512-byte logical blocks: (7.96
GB/7.42 GiB)
[118258.631335] sd 10:0:0:0: [sdg] No Caching mode page present
[118258.631340] sd 10:0:0:0: [sdg] Assuming drive cache: write through
[118258.633952] sd 10:0:0:0: [sdg] No Caching mode page present
[118258.633957] sd 10:0:0:0: [sdg] Assuming drive cache: write through
[118258.636662]  sdg: unknown partition table
```

In the example above the card was identified as **sdg**.

Do not mount the SD/MMC card. If your system automatically mounted it, umount all partitions before continuing.
Also notice that after the process, existing partitions might become unusable.

Locate the U-Boot image file (for example /tmp/u-boot-ccardwmx28js-ivt.sb) and run the script to raw write it to the SD/MMC card (you may need root privileges):

```
$   cd /usr/local/DigiEL-X.Y/scripts
$   ./mk_mx28_sd -o -b /tmp/u-boot-ccardwmx28js-ivt.sb /dev/sdg
Insert the specified device (/dev/sdc) now, if you have not already done so.


This script requires the use of 'sudo' and erases the content of the
specified device (/dev/sdc)
Are you sure you want to continue? (yes/no):
yes

Command (m for help): Building a new DOS disklabel with disk identifier
0x0bc9bb7e.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by
w(rite)

Command (m for help): Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): Partition number (1-4, default 1): First sector (2048-
7716863, default 2048): Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-7716863, default 7716863):
Command (m for help): Selected partition 1
Hex code (type L to list codes): Changed system type of partition 1 to b
(W95 FAT32)

Command (m for help): Partition type:
   p   primary (1 primary, 0 extended, 3 free)
   e   extended
```

```
Select (default p): Partition number (1-4, default 2): First sector (67584-
7716863, default 67584): Using default value 67584
Last sector, +sectors or +size{K,M,G} (67584-7716863, default 7716863):
Command (m for help): Partition number (1-4): Hex code (type L to list
codes): Changed system type of partition 2 to 53 (OnTrack DM6 Aux3)

Command (m for help): Partition type:
   p   primary (2 primary, 0 extended, 2 free)
   e   extended
Select (default p): Partition number (1-4, default 3): First sector (133120-
7716863, default 133120): Using default value 133120
Last sector, +sectors or +size{K,M,G} (133120-7716863, default 7716863):
Using default value 7716863

Command (m for help): Partition number (1-4): Hex code (type L to list
codes):
Command (m for help):
Disk /dev/sdc: 3951 MB, 3951034368 bytes
122 heads, 62 sectors/track, 1020 cylinders, total 7716864 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x0bc9bb7e

   Device Boot      Start         End      Blocks   Id  System
/dev/sdc1            2048       67583       32768    b  W95 FAT32
/dev/sdc2           67584      133119       32768   53  OnTrack DM6 Aux3
/dev/sdc3          133120     7716863     3791872   83  Linux

Command (m for help): The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for additional
information.
Syncing disks.

Installing boot stream /tftpboot/u-boot-ccardwmx28js-ivt.sb on /dev/sdc2...
0+1 records in
1+0 records out
512 bytes (512 B) copied, 0.672998 s, 0.8 kB/s
722+1 records in
723+0 records out
370176 bytes (370 kB) copied, 0.165805 s, 2.2 MB/s
...finished installing boot stream on /dev/sdc2.

0
Omitting rootfs install.


Done! Plug the SD/MMC card into the i.MX board and power-on.
```

where **/tmp/u-boot-ccardwmx28js-ivt.sb** must be substituted with the path to your U-Boot image and **/dev/sdg** must be substituted with the device assigned by Linux to your SD/MMC card.

After this, you can remove the card and proceed to chapter *14.3.3. Preparing the hardware platform* for further instructions.

14.3.3 Preparing the hardware platform

Although the JumpStart kit is by default configured to boot from the flash memory in the module, jumpers or microswitches on the development board allow you to establish a different configuration to force the module to boot from the SD/MMC or microSD card.

Please refer to the Hardware Reference Manual of your JumpStart kit for instructions about setting the needed jumpers/microswitches configuration to force boot from SD/MMC or microSD card.

Once the needed hardware configuration is done, insert the card in the JumpStart socket and power up the module. U-Boot should then boot from the SD/MMC card, allowing you to recover the boot loader in the flash.

14.3.4 Creating partitions in the bootable SD/MMC card

The bootable card created in the previous paragraphs has no partitions and it's not formatted. We can create different partitions with different formats (FAT, ext2, ext3,…) while keeping the card bootable. For this we can use any partition manager software.

The only caveat to have in mind when creating a partition table is to place the first partition at an offset of at least 1MiB after the start of the media, to avoid overwriting the boot loader that is at the first sectors.

Due to the special formatting of bootable SD/MMC card for i.MX28 based platforms, these cards cannot hold a partition table and can only be used for booting.