U-Boot beagleboneblack Documentation

Abstract

About this Document

The documentation is written in reStructuredText and converted into a pdf document. Some parts of this document are created automatically out of the log files from the tbot build process.

This document is generated for the beagleboneblack with U-Boot version

U-Boot 2017.09-00396-g6ca43a5 (Sep 30 2017 - 07:16:03 +0200)

Introduction

This document describes how to use the firmware U-Boot and the operating system Linux in Embedded Power Architecture®, ARM and MIPS Systems.

There are many steps along the way, and it is nearly impossible to cover them all in depth, but we will try to provide all necessary information to get an embedded system running from scratch. This includes all the tools you will probably need to configure, build and run U-Boot and Linux.

First, we describe how to install the Cross Development Tools Embedded Linux Development Kit which you probably need - at least when you use a standard x86 PC running Linux or a Sun Solaris 2.6 system as build environment.

Then we describe what needs to be done to connect to the serial console port of your target: you will have to configure a terminal emulation program like cu or kermit.

In most cases you will want to load images into your target using ethernet; for this purpose you need TFTP and DHCP / BOOTP servers. A short description of their configuration is given.

A description follows of what needs to be done to configure and build the U-Boot for the beagleboneblack board, and how to install it and get it working on that board.

The configuration, building and installing of Linux in an embedded configuration is the next step. We use SELF, our Simple Embedded Linux Framework, to demonstrate how to set up both a development system (with the root filesystem mounted over NFS) and an embedded target configuration (running from a ramdisk image based on busybox).

This document does not describe what needs to be done to port U-Boot or Linux to a new hardware platform. Instead, it is silently assumed that your board is already supported by U-Boot and Linux.

Disclaimer

U-Boot

Current Versions

Das U-Boot (or just "U-Boot" for short) is Open Source Firmware for Embedded Power Architecture®, ARM, MIPS, x86 and other processors. The U-Boot project is hosted by DENX, where you can also find the project home page: http://www.denx.de/wiki/U-Boot/

The current version of the U-Boot source code can be retrieved from the DENX "git" repository.

You can browse the "git" repositories at http://git.denx.de/

The trees can be accessed through the git, HTTP, and rsync protocols. For example you can use one of the following commands to create a local clone of one of the source trees:

```
git clone git://git.denx.de/u-boot.git u-boot/
git clone http://git.denx.de/u-boot.git u-boot/
git clone rsync://git.denx.de/u-boot.git u-boot/
```

For details please see here.

Official releases of U-Boot are also available through FTP. Compressed tar archives can downloaded from the directory ftp://ftp.denx.de/pub/u-boot/.

Get U-Boot code for the beagleboneblack

define some PATH variables

Export our workdirectory:

```
$ export TBOT_BASEDIR=/work/hs/tbot
$
```

and cd into it

```
$ cd $TBOT_BASEDIR
$
```

clone the U-Boot code

Now we simply clone the U-Boots source code with git:

```
$ git clone  git://git.denx.de/u-boot u-boot-am335x_evm
Cloning into 'u-boot-am335x_evm'...
remote: Counting objects: 501780, done.
remote: Compressing objects: 100% (85102/85102), done.
remote: Total 501780 (delta 412960), reused 497673 (delta 409125)
Receiving objects: 100% (501780/501780), 102.72 MiB | 1.13 MiB/s, done.
Resolving deltas: 100% (412960/412960), done.
Checking connectivity... done.
$
```

cd into it

```
$ cd u-boot-am335x_evm
$
```

checkout the branch you want to test

```
$ git checkout master
Already on 'master'
Your branch is up-to-date with 'origin/master'.
$
```

just for the records, print some info of the branch

```
$ git describe --tags
v2017.09-396-g6ca43a5
$
```

setup toolchain

This depends on the toolchain you use.

```
$ export ARCH=arm
$ printenv PATH | grep --color=never /home/hs/.buildman-toolchains/gcc-4.9.0-nolibc/arm-unknown-linux-gnueabi/bin
$ export PATH=/home/hs/.buildman-toolchains/gcc-4.9.0-nolibc/arm-unknown-linux-gnueabi/bin:$PATH
$ export CROSS_COMPILE=arm-unknown-linux-gnueabi-
$
```

If you have no toolchain installed, may you try buildman (see U-Boot code tools/buildman) to fetch a toolchain:

```
cd /path/to/u-boot
PATH=$PATH:`pwd`/tools/buildman
buildman --fetch-arch arm
```

compile U-Boot for the beagleboneblack

Add the path to the dtc command to your PATH variable

```
$ printenv PATH | grep --color=never /work/tbot2go/tbot/dtc
$ export PATH=/work/tbot2go/tbot/dtc:$PATH
$
```

clean the source code

```
$ make mrproper
$
```

configure source for the beagleboneblack

```
$ make am335x_evm_defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/zconf.lex.c
  SHIPPED scripts/kconfig/zconf.hash.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
$
```

Now compile it

```
$ make -s DTC_FLAGS="-S 0xb000" all
*** Your GCC is older than 6.0 and will not be supported starting in v2018.01.
arch/arm/dts/am335x-pxm50.dtb: Warning (unit_address_format): Node /ocp/i2c@4819c000/egalax_ts@04 unit name should not have\
  leading 0s
==================== WARNING ======================
This board uses CONFIG_DM_I2C_COMPAT. Please remove
(possibly in a subsequent patch in your series)
before sending patches to the mailing list.
====================================================
$
```

after U-Boot is compiled, copy the resulting binaries we need later to our tftpboot directory.

```
$ scp  u-boot.bin pi@192.168.1.110:/srv/tftpboot//beagleboneblack/tbot/u-boot.bin
pi@192.168.1.110's password:
u-boot.bin\
  100%  494KB 493.8KB/s   00:00
$
$ scp  u-boot.img pi@192.168.1.110:/srv/tftpboot//beagleboneblack/tbot/u-boot.img
pi@192.168.1.110's password:
u-boot.img\
  100%  716KB 715.7KB/s   00:00
$
$ scp  MLO pi@192.168.1.110:/srv/tftpboot//beagleboneblack/tbot/MLO
pi@192.168.1.110's password:
MLO\
  100%   92KB  92.4KB/s   00:00
$
```

We also copy the u-boot.dtb file to our tftp directory, as we do some testing with it later.

```
$ scp  u-boot.dtb pi@192.168.1.110:/srv/tftpboot//beagleboneblack/tbot/u-boot.dtb
pi@192.168.1.110's password:
u-boot.dtb\
  100%   44KB  44.0KB/s   00:00
$
```

U-Boot installation

install U-Boot

for this example, we install the new U-Boot on the SD card, as we use SD card bootmode.

```
=> print tbot_upd_uboot load_uboot upd_uboot
tbot_upd_uboot=run load_uboot;run upd_uboot
load_uboot=tftp ${load_addr_r} ${ubfile}
upd_uboot=fatwrite mmc 1:1 ${load_addr_r} u-boot.img ${filesize}
=>
```

tfpt the new u-boot image into ram and write it to the sd card.

```
=> run tbot_upd_uboot
link up on port 0, speed 100, full duplex
Using ethernet@4a100000 device
TFTP from server 192.168.3.1; our IP address is 192.168.3.20
Filename 'beagleboneblack/tbot/u-boot.img'.
Load address: 0x81000000
Loading: *■#################################################################
       #################################################################
       ##############
       1.1 MiB/s
done
Bytes transferred = 732904 (b2ee8 hex)
writing u-boot.img
732904 bytes written
=>
```

install SPL

for this example, we install the new SPL on the SD card, as we use SD card bootmode.

```
=> print tbot_upd_spl load_mlo upd_mlo
tbot_upd_spl=run load_mlo;run upd_mlo
load_mlo=tftp ${load_addr_r} ${mlofile}
upd_mlo=fatwrite mmc 1:1 ${load_addr_r} mlo ${filesize}
=>
```

tfpt the new SPL image into ram and write it to the sd card.

```
=> run tbot_upd_spl
link up on port 0, speed 100, full duplex
Using ethernet@4a100000 device
TFTP from server 192.168.3.1; our IP address is 192.168.3.20
Filename 'beagleboneblack/tbot/MLO'.
Load address: 0x81000000
Loading: *■###################
       1 MiB/s
done
Bytes transferred = 94652 (171bc hex)
writing mlo
94652 bytes written
=>
```

Tool Installation

U-Boot uses a special image format when loading the Linux kernel or ramdisk or other images. This image contains (among other things) information about the time of creation, operating system, compression type, image type, image name and CRC32 checksums.

The tool mkimage is used to create such images or to display the information they contain. When using the ELDK, the mkimage command is already included with the other ELDK tools.

If you don't use the ELDK then you should install mkimage in some directory that is in your command search PATH, for instance:

```
$ cp tools/mkimage /usr/local/bin/
```

mkimage is readily available in several distributions; for example, in Ubuntu it is part of the u-boot-tools package, so it can be installed with:

```
$ sudo apt-get install u-boot-tools
```

In Fedora the package name is uboot-tools, and the command to install it is:

```
$ sudo dnf install uboot-tools
```

Finally, if you're building with OpenEmbedded or Yocto Project, you would want to add the u-boot-fw-utils recipe to your image.

U-Boot Command Line Interface

The following section describes the most important commands available in U-Boot. Please note that U-Boot is highly configurable, so not all of these commands may be available in the configuration of U-Boot installed on your hardware, or additional commands may exist. You can use the help command to print a list of all available commands for your configuration.

For most commands, you do not need to type in the full command name; instead it is sufficient to type a few characters. For instance, help can be abbreviated as h.

The behaviour of some commands depends on the configuration of U-Boot and on the definition of some variables in your U-Boot environment.

Almost all U-Boot commands expect numbers to be entered in hexadecimal input format. (Exception: for historical reasons, the sleep command takes its argument in decimal input format.)

Be careful not to use edit keys besides 'Backspace', as hidden characters in things like environment variables can be very difficult to find.

Information Commands

bdinfo - print Board Info structure

```
=> help bdinfo
bdinfo - print Board Info structure

Usage:
bdinfo
=>
```

The bdinfo command (bdi) prints the information that U-Boot passes about the board such as memory addresses and sizes, clock frequencies, MAC address, etc. This information is mainly needed to be passed to the Linux kernel.

```
=> bdi
arch_number = 0x00000E05
boot_params = 0x80000100
DRAM bank   = 0x00000000
-> start    = 0x80000000
-> size     = 0x20000000
baudrate    = 115200 bps
TLB addr    = 0x9FFF0000
relocaddr   = 0x9FF4E000
reloc off   = 0x1F74E000
irq_sp      = 0x9DF21EC0
sp start    = 0x9DF21EB0
Early malloc usage: 184 / 400
fdt_blob = 9df21ed8
=>
```

coninfo - print console devices and informations

```
=> help conin
coninfo - print console devices and information

Usage:
coninfo
=>
```

The coninfo command (conin) displays information about the available console I/O devices.

```
=> conin
List of available devices:
serial@44e09000 00000007 IO stdin stdout stderr
serial   00000003 IO
=>
```

The output contains the device name, flags, and the current usage. For example, the output

```
serial@44e09000 00000003 IO stdin stdout stderr
```

means that the serial device provides input (flag 'I') and output (flag 'O') functionality and is currently assigned to the 3 standard I/O streams stdin, stdout and stderr.

flinfo - print FLASH memory information

```
=> help flinfo
Unknown command 'flinfo' - try 'help' without arguments for list of all known commands

=>
```

The command flinfo (fli) can be used to get information about the available flash memory (see Flash Memory Commands below).

```
=> flinfo
Unknown command 'flinfo' - try 'help'
=>
```

help - print online help

```
=> help help
help - print command description/usage

Usage:
help
      - print brief description of all commands
help command ...
      - print detailed usage of 'command'
=>
```

The help command (h or ?) prints online help. Without any arguments, it prints a list of all U-Boot commands that are available in your configuration of U-Boot. You can get detailed information for a specific command by typing its name as argument to the help command:

```
=> help printenv tftp
printenv - print environment variables

Usage:
printenv [-a]
    - print [all] values of all environment variables
printenv name ...
    - print value of environment variable 'name'
tftpboot - boot image via network using TFTP protocol

Usage:
tftpboot [loadAddress] [[hostIPaddr:]bootfilename]
=>
```

Memory Commands

base - print or set address offset

```
=> help base
base - print or set address offset

Usage:
base
    - print address offset for memory commands
base off
    - set address offset for memory commands to 'off'
=>
```

You can use the base command (ba) to print or set a "base address" that is used as the address offset for all subsequent memory commands; the default value of the base address is 0, so all addresses you enter are used unmodified. However, when you repeatedly have to access a certain memory region (like the internal memory of some embedded Power Architecture® processors) it can be very convenient to set the base address to the start of this area and then use only the offsets:

ToDo

crc32 - checksum calculation

The crc32 command (crc) can be used to calculate a CRC32 checksum over a range of memory:

```
=> crc 0x80000004 0x3fc a6d53e40
=>
```

When used with 3 arguments, the command stores the calculated checksum at the given address:

```
=> crc 0x80000004 0x3fc 0x80000000 a6d53e40
=> md 0x80000000 4
80000000: 403ed5a6 b9070000 38000000 38070000    ..>@.......8...8
=>
```

As you can see, the CRC32 checksum was not only printed, but also stored at address passed in the 3th argument.

cmp - memory compare

```
=> help cmp
cmp - memory compare

Usage:
cmp [.b, .w, .l] addr1 addr2 count
=>
```

With the cmp command you can test whether the contents of two memory areas are identical or not. The command will test either the whole area as specified by the 3rd (length) argument, or stop at the first difference.

```
=> cmp 0x80000000 0x80100000L 40000
Total of 262144 word(s) were the same
=> md 0x80000000 0xc
80000000: a555037f 7d9aebcf 1a1e2529 dfb0e082    ..U....})%......
80000010: 0d838660 e04e09df 62a2cc8a def1601a    `.....N....b.`..
80000020: 65b8b965 fb0fa81b c202c517 7fc93243    e..e........C2..
=> md 0x80100000L 0xc
80100000: a555037f 7d9aebcf 1a1e2529 dfb0e082    ..U....})%......
80100010: 0d838660 e04e09df 62a2cc8a def1601a    `.....N....b.`..
80100020: 65b8b965 fb0fa81b c202c517 7fc93243    e..e........C2..
=>
```

Like most memory commands the :redtext:cmp` can access the memory in different sizes: as 32 bit (long word), 16 bit (word) or 8 bit (byte) data. If invoked just as cmp the default size (32 bit or long words) is used; the same can be selected explicitly by typing cmp.l instead. If you want to access memory as 16 bit or word data, you can use the variant cmp.w instead; and to access memory as 8 bit or byte data please use cmp.b.

Please note that the count argument specifies the number of data items to process, i. e. the number of long words or words or bytes to compare.

```
=> cmp.l 0x80000000 0x80100000L 40000
Total of 262144 word(s) were the same
=> cmp.w 0x80000000 0x80100000L 80000
Total of 524288 halfword(s) were the same
=> cmp.b 0x80000000 0x80100000L 100000
Total of 1048576 byte(s) were the same
=>
```

cp - memory copy

```
=> help cp
cp - memory copy

Usage:
cp [.b, .w, .l] source target count
=>
```

The cp command is used to copy memory areas.

```
=> cp 0x80000000 0x80100000L 10000
=>
```

The cp command understands the type extensions .l, .w and .b :

```
=> cp.l 0x80000000 0x80100000L 10000
=> cp.w 0x80000000 0x80100000L 20000
=> cp.b 0x80000000 0x80100000L 40000
=>
```

md - memory display

```
=> help md
md - memory display

Usage:
md [.b, .w, .l] address [# of objects]
=>
```

The md command can be used to display memory contents both as hexadecimal and ASCII data.

```
=> md 0x80000000
80000000: a555037f 7d9aebcf 1a1e2529 dfb0e082    ..U....})%......
80000010: 0d838660 e04e09df 62a2cc8a def1601a    `.....N....b.`..
80000020: 65b8b965 fb0fa81b c202c517 7fc93243    e..e........C2..
80000030: d5246ead 92d57185 768d30c9 cecb8383    .n$..q...0.v....
80000040: 4cc9e97b 852c5e63 9cf20e84 4f946122    {..Lc^,....."a.O
80000050: 98649544 0a9a9788 526ca383 caa717c9    D.d.......lR....
80000060: 40833929 97efd07d 93b179f9 faa3c3d2    )9.@}....y......
80000070: d9aadf23 272e8f4c 8686240a 2230043a    #...L..'.$..:.0"
80000080: 3939631e 3eb909cb 369e094f b2bb9795    .c99...>O..6....
80000090: e64395c4 553b38e3 1e68ac97 ffdd5ff3    ..C..8;U..h.._..
800000a0: 16c77f90 12bf8209 3b22f187 d922fba9    ..........";..".
800000b0: 0f8a8ebc 3bee36c1 3d432085 04eb523f    .....6.;. C=?R..
800000c0: 8525a54d bb52b1ea 1bd58cc6 210481d9    M.%...R........!
800000d0: ef186528 dc261323 17e5f5bf d57f5029    (e..#.&.....)P..
800000e0: 6bd14e6d f5289e33 07fcfa87 e9fef934    mN.k3.(.....4...
800000f0: 1f4bdda6 2ef1e31d 2d7a05b2 2d21b481    ..K.......z-..!-
=>
```

This command can also be used with the type extensions .l, .w and .b :

```
=> md.w 0x80000000
80000000: 037f a555 ebcf 7d9a 2529 1a1e e082 dfb0    ..U....})%......
80000010: 8660 0d83 09df e04e cc8a 62a2 601a def1    `.....N....b.`..
80000020: b965 65b8 a81b fb0f c517 c202 3243 7fc9    e..e........C2..
80000030: 6ead d524 7185 92d5 30c9 768d 8383 cecb    .n$..q...0.v....
80000040: e97b 4cc9 5e63 852c 0e84 9cf2 6122 4f94    {..Lc^,....."a.O
80000050: 9544 9864 9788 0a9a a383 526c 17c9 caa7    D.d.......lR....
80000060: 3929 4083 d07d 97ef 79f9 93b1 c3d2 faa3    )9.@}....y......
80000070: df23 d9aa 8f4c 272e 240a 8686 043a 2230    #...L..'.$..:.0"
=> md.b 0x80000000
80000000: 7f 03 55 a5 cf eb 9a 7d 29 25 1e 1a 82 e0 b0 df    ..U....})%......
80000010: 60 86 83 0d df 09 4e e0 8a cc a2 62 1a 60 f1 de    `.....N....b.`..
80000020: 65 b9 b8 65 1b a8 0f fb 17 c5 02 c2 43 32 c9 7f    e..e........C2..
80000030: ad 6e 24 d5 85 71 d5 92 c9 30 8d 76 83 83 cb ce    .n$..q...0.v....
=>
```

The last displayed memory address and the value of the count argument are remembered, so when you enter md again without arguments it will automatically continue at the next address, and use the same count again.

```
=> md.b 0x80000000 0x20
80000000: 7f 03 55 a5 cf eb 9a 7d 29 25 1e 1a 82 e0 b0 df    ..U....})%......
80000010: 60 86 83 0d df 09 4e e0 8a cc a2 62 1a 60 f1 de    `.....N....b.`..
=> md.w 0x80000000
80000000: 037f a555 ebcf 7d9a 2529 1a1e e082 dfb0    ..U....})%......
80000010: 8660 0d83 09df e04e cc8a 62a2 601a def1    `.....N....b.`..
80000020: b965 65b8 a81b fb0f c517 c202 3243 7fc9    e..e........C2..
80000030: 6ead d524 7185 92d5 30c9 768d 8383 cecb    .n$..q...0.v....
=> md 0x80000000
80000000: a555037f 7d9aebcf 1a1e2529 dfb0e082    ..U....})%......
80000010: 0d838660 e04e09df 62a2cc8a def1601a    `.....N....b.`..
80000020: 65b8b965 fb0fa81b c202c517 7fc93243    e..e........C2..
80000030: d5246ead 92d57185 768d30c9 cecb8383    .n$..q...0.v....
80000040: 4cc9e97b 852c5e63 9cf20e84 4f946122    {..Lc^,....."a.O
80000050: 98649544 0a9a9788 526ca383 caa717c9    D.d.......lR....
80000060: 40833929 97efd07d 93b179f9 faa3c3d2    )9.@}....y......
80000070: d9aadf23 272e8f4c 8686240a 2230043a    #...L..'.$..:.0"
=>
```

mm - memory modify (auto-incrementing)

```
=> help mm
mm - memory modify (auto-incrementing address)

Usage:
mm [.b, .w, .l] address
=>
```

The mm command is a method to interactively modify memory contents. It will display the address and current contents and then prompt for user input. If you enter a legal hexadecimal number, this new value will be written to the address. Then the next address will be prompted. If you don't enter any value and just press ENTER, then the contents of this address will remain unchanged. The command stops as soon as you enter any data that is not a hex number (like .):

```
=> mm 0x80000000
80000000: a555037f ? 0
80000004: 7d9aebcf ? 0xaabbccdd
80000008: 1a1e2529 ? 0x01234567
8000000c: dfb0e082 ? .
=> md 0x80000000 10
80000000: 00000000 aabbccdd 01234567 dfb0e082    ........gE#.....
80000010: 0d838660 e04e09df 62a2cc8a def1601a    `.....N....b.`..
80000020: 65b8b965 fb0fa81b c202c517 7fc93243    e..e........C2..
80000030: d5246ead 92d57185 768d30c9 cecb8383    .n$..q...0.v....
=>
```

Again this command can be used with the type extensions .l, .w and .b :

```
=> mm.w 0x80000000
80000000: 0000 ? 0x0101
80000002: 0000 ? 0x0202
80000004: ccdd ? 0x4321
80000006: aabb ? 0x8765
80000008: 4567 ? .
=> md 0x80000000 10
80000000: 02020101 87654321 01234567 dfb0e082    ....!Ce.gE#.....
80000010: 0d838660 e04e09df 62a2cc8a def1601a    `.....N....b.`..
80000020: 65b8b965 fb0fa81b c202c517 7fc93243    e..e........C2..
80000030: d5246ead 92d57185 768d30c9 cecb8383    .n$..q...0.v....
=>
=> mm.b 0x80000000
80000000: 01 ? 0x48
80000001: 01 ? 0x65
80000002: 02 ? 0x6c
80000003: 02 ? 0x6c
80000004: 21 ? 0x6f
80000005: 43 ? 0x20
80000006: 65 ? 0x20
80000007: 87 ? 0x20
80000008: 67 ? .
=> md 0x80000000 10
80000000: 6c6c6548 2020206f 01234567 dfb0e082    Hello   gE#.....
80000010: 0d838660 e04e09df 62a2cc8a def1601a    `.....N....b.`..
80000020: 65b8b965 fb0fa81b c202c517 7fc93243    e..e........C2..
80000030: d5246ead 92d57185 768d30c9 cecb8383    .n$..q...0.v....
=>
```

mw - memory write (fill)

```
=> help mw
mw - memory write (fill)

Usage:
mw [.b, .w, .l] address value [count]
=>
```

The mw command is a way to initialize (fill) memory with some value. When called without a count argument, the value will be written only to the specified address. When used with a count value, the entire memory area will be initialized with this value:

```
=> md 0x80000000 0x10
80000000: 6c6c6548 2020206f 01234567 dfb0e082    Hello   gE#.....
80000010: 0d838660 e04e09df 62a2cc8a def1601a    `.....N....b.`..
80000020: 65b8b965 fb0fa81b c202c517 7fc93243    e..e........C2..
80000030: d5246ead 92d57185 768d30c9 cecb8383    .n$..q...0.v....
=> mw 0x80000000 0xaabbccdd
=> md 0x80000000 0x10
80000000: aabbccdd 2020206f 01234567 dfb0e082    ....o   gE#.....
80000010: 0d838660 e04e09df 62a2cc8a def1601a    `.....N....b.`..
80000020: 65b8b965 fb0fa81b c202c517 7fc93243    e..e........C2..
80000030: d5246ead 92d57185 768d30c9 cecb8383    .n$..q...0.v....
=> mw 0x80000000 0 6
=> md 0x80000000 0x10
80000000: 00000000 00000000 00000000 00000000    ................
80000010: 00000000 00000000 62a2cc8a def1601a    ...........b.`..
80000020: 65b8b965 fb0fa81b c202c517 7fc93243    e..e........C2..
80000030: d5246ead 92d57185 768d30c9 cecb8383    .n$..q...0.v....
=>
```

This is another command that accepts the type extensions .l, .w and .b :

```
=> mw.w 0x80000004L 0x1155 6
=> md 0x80000000 0x10
80000000: 00000000 11551155 11551155 11551155    ....U.U.U.U.U.U.
80000010: 00000000 00000000 62a2cc8a def1601a    ...........b.`..
80000020: 65b8b965 fb0fa81b c202c517 7fc93243    e..e........C2..
80000030: d5246ead 92d57185 768d30c9 cecb8383    .n$..q...0.v....
=> mw.b 0x80000007L 0xff 7
=> md 0x80000000 0x10
80000000: 00000000 ff551155 ffffffff 1155ffff    ....U.U.......U.
80000010: 00000000 00000000 62a2cc8a def1601a    ...........b.`..
80000020: 65b8b965 fb0fa81b c202c517 7fc93243    e..e........C2..
80000030: d5246ead 92d57185 768d30c9 cecb8383    .n$..q...0.v....
=>
```

nm - memory modify (constant address)

```
=> help nm
nm - memory modify (constant address)

Usage:
nm [.b, .w, .l] address
=>
```

The nm command (non-incrementing memory modify) can be used to interactively write different data several times to the same address. This can be useful for instance to access and modify device registers:

```
=> nm 0x80000000
80000000: 00000000 ? 0x48
80000000: 00000048 ? 0x65
80000000: 00000065 ? 0x6c
80000000: 0000006c ? 0x6c
80000000: 0000006c ? 0x6f
80000000: 0000006f ? .
=> md 0x80000000 10
80000000: 0000006f ff551155 ffffffff 1155ffff    o...U.U.......U.
80000010: 00000000 00000000 62a2cc8a def1601a    ..........b.`..
80000020: 65b8b965 fb0fa81b c202c517 7fc93243    e..e........C2..
80000030: d5246ead 92d57185 768d30c9 cecb8383    .n$..q...0.v....
=>
```

The nm command also accepts the type .l, .w and .b

loop - infinite loop on address range

```
=> help loop
loop - infinite loop on address range

Usage:
loop [.b, .w, .l] address number_of_objects
=>
```

The loop command reads in a tight loop from a range of memory. This is intended as a special form of a memory test, since this command tries to read the memory as fast as possible.

This command will never terminate. There is no way to stop it but to reset the board!

Execution Control Commands

source - run script from memory

```
=> help source
source - run script from memory

Usage:
source [addr]
      - run script starting at addr
      - A valid image header must be present
For FIT format uImage addr must include subimage
unit name in the form of addr:<subimg_uname>
=>
```

With the source command you can run "shell" scripts under U-Boot: You create a U-Boot script image by simply writing the commands you want to run into a text file; then you will have to use the mkimage tool to convert this text file into a U-Boot image (using the image type script).

This image can be loaded like any other image file, and with source you can run the commands in such an image. For instance, the following text file:

```
$ cat source_example.txt
echo
echo Network Configuration:
echo ----------------------
echo Target:
printenv ipaddr hostname
echo
echo Server:
printenv serverip rootpath
echo
$
```

can be converted into a U-Boot script image using the mkimage command like this:

```
$ mkimage -A ppc -O linux -T script -C none -a 0 -e 0 -n "autoscr example script" -d /work/tbot2go/tbo\
> t//source_example.txt /srv/tftpboot//beagleboneblack/tbot/source_example.scr
Image Name:   autoscr example script
Created:      Sat Sep 30 05:24:22 2017
Image Type:   PowerPC Linux Script (uncompressed)
Data Size:    157 Bytes = 0.15 kB = 0.00 MB
Load Address: 00000000
Entry Point:  00000000
Contents:
   Image 0: 149 Bytes = 0.15 kB = 0.00 MB
$
```

Now you can load and execute this script image in U-Boot:

```
=> tftp 0x80000000 /srv/tftpboot//beagleboneblack/tbot/random
link up on port 0, speed 100, full duplex
Using ethernet@4a100000 device
TFTP from server 192.168.3.1; our IP address is 192.168.3.20
Filename '/srv/tftpboot//beagleboneblack/tbot/random'.
Load address: 0x80000000
Loading: *■#################################################################
       #################################################################
       #################################################################
       ##########
       1 MiB/s
done
Bytes transferred = 1048576 (100000 hex)
=>
=> imi 0x80000000
```

```
## Checking Image at 80000000 ...
   Legacy image found
   Image Name:   autoscr example script
   Created:      2017-09-30   5:24:22 UTC
   Image Type:   PowerPC Linux Script (uncompressed)
   Data Size:    157 Bytes = 157 Bytes
   Load Address: 00000000
   Entry Point:  00000000
   Contents:
      Image 0: 149 Bytes = 149 Bytes
   Verifying Checksum ... OK
=> source 0x80000000
## Executing script at 80000000

Network Configuration:
----------------------
Target:
ipaddr=192.168.3.20
hostname=bbb

Server:
serverip=192.168.3.1
rootpath=/work/tbot2go/tbot/nfs/bbb

=>
```

bootm - boot application image from memory

```
=> help bootm
bootm - boot application image from memory

Usage:
bootm [addr [arg ...]]
    - boot application image stored in memory
      passing arguments 'arg ...'; when booting a Linux kernel,
      'arg' can be the address of an initrd image
      When booting a Linux kernel which requires a flat device-tree
      a third argument is required which is the address of the
      device-tree blob. To boot that kernel without an initrd image,
      use a '-' for the second argument. If you do not pass a third
      a bd_info struct will be passed instead

For the new multi component uImage format (FIT) addresses
      must be extended to include component or configuration unit name:
      addr:<subimg_uname> - direct component image specification
      addr#<conf_uname>   - configuration specification
      Use iminfo command to get the list of existing component
      images and configurations.

Sub-commands to do part of the bootm sequence.  The sub-commands must be
issued in the order below (it's ok to not issue all sub-commands):
      start [addr [arg ...]]
      loados  - load OS image
      ramdisk - relocate initrd, set env initrd_start/initrd_end
      fdt     - relocate flat device tree
      cmdline - OS specific command line processing/setup
      bdt     - OS specific bd_t processing
      prep    - OS specific prep before relocation or go
      go      - start OS
=>
```

The bootm command is used to start operating system images. From the image header it gets information about the type of the operating system, the file compression method used (if any), the load and entry point addresses, etc. The command will then load the image to the required memory address, uncompressing it on the fly if necessary. Depending on the OS it will pass the required boot arguments and start the OS at it's entry point.

The first argument to bootm is the memory address (in RAM, ROM or flash memory) where the image is stored, followed by optional arguments that depend on the OS.

Linux requires the flattened device tree blob to be passed at boot time, and bootm expects its third argument to be the address of the blob in memory. Second argument to bootm depends on whether an initrd initial ramdisk image is to be used. If the kernel should be booted without the initial ramdisk, the second argument should be given as "-", otherwise it is interpreted as the start address of initrd (in RAM, ROM or flash memory).

To boot a Linux kernel image without a initrd ramdisk image, the following command can be used:

```
=> bootm ${kernel_addr} - ${fdt_addr}
```

If a ramdisk image shall be used, you can type:

```
=> bootm ${kernel_addr} ${ramdisk_addr} ${fdt_addr}
```

Both examples of course imply that the variables used are set to correct addresses for a kernel, fdt blob and a initrd ramdisk image.

When booting images that have been loaded to RAM (for instance using TFTP download) you have to be careful that the locations where the (compressed) images were stored do not overlap with the memory needed to load the uncompressed kernel. For instance, if you load a ramdisk image at a location in low memory, it may be overwritten when the Linux kernel gets loaded. This will cause undefined system crashes.

go - start application at address 'addr'

```
=> help go
go - start application at address 'addr'

Usage:
go addr [arg ...]
    - start application at address 'addr'
      passing 'arg' as arguments
=>
```

U-Boot has support for so-called standalone applications. These are programs that do not require the complex environment of an operating system to run. Instead they can be loaded and executed by U-Boot directly, utilizing U-Boot's service functions like console I/O or malloc() and free().

This can be used to dynamically load and run special extensions to U-Boot like special hardware test routines or bootstrap code to load an OS image from some filesystem.

The go command is used to start such standalone applications. The optional arguments are passed to the application without modification.

TODO For more information see 5.12. U-Boot Standalone Applications.

Download Commands

bootp - boot image via network using BOOTP/TFTP protocol

```
=> help bootp
bootp - boot image via network using BOOTP/TFTP protocol

Usage:
bootp [loadAddress] [[hostIPaddr:]bootfilename]
=>
```

dhcp - invoke DHCP client to obtain IP/boot params

```
=> help dhcp
dhcp - boot image via network using DHCP/TFTP protocol

Usage:
dhcp [loadAddress] [[hostIPaddr:]bootfilename]
=>
```

loadb - load binary file over serial line (kermit mode)

```
=> help loadb
loadb - load binary file over serial line (kermit mode)

Usage:
loadb [ off ] [ baud ]
    - load binary file over serial line with offset 'off' and baudrate 'baud'
=>
```

With kermit you can download binary data via the serial line.

Make sure you use the following settings in kermit.

```
set carrier-watch off
(/work/tbot2go/tbot/) C-Kermit>set handshake none
(/work/tbot2go/tbot/) C-Kermit>set flow-control none
(/work/tbot2go/tbot/) C-Kermit>robust
(/work/tbot2go/tbot/) C-Kermit>set file type bin
(/work/tbot2go/tbot/) C-Kermit>set file name lit
(/work/tbot2go/tbot/) C-Kermit>set rec pack 100
(/work/tbot2go/tbot/) C-Kermit>set send pack 100
(/work/tbot2go/tbot/) C-Kermit>set window 5
(/work/tbot2go/tbot/) C-Kermit>
```

If you have problems with downloading, may you set the values

```
set rec pack
set send pack
```

to smaller values.

Now for example download u-boot.img.

```
=> loadb 80000000
## Ready for binary (kermit) download to 0x80000000 at 115200 bps...

(Back at raspberrypitbot2go)
----------------------------------------------------
(/work/tbot2go/tbot/) C-Kermit>
(/work/tbot2go/tbot/) C-Kermit>send /protocol=kermit /srv/tftpboot//beagleboneblack/tbot/u-boot.img
C-Kermit 9.0.302 OPEN SOURCE:, 20 Aug 2011, raspberrypitbot2go [192.168.3.1]

   Current Directory: /work/tbot2go/tbot
Communication Parity: none/ttyUSB0
 CommunicRTT/Timeout: 01 / 02
             SENDING:  => /tftpboot//beagleboneblack/tbot/u-boot.img
           File Type: BINARY
           File Size: 732904
        Percent Done: 100 //////////////////////////////////////////////////
:06  ...10...20...30...40...50...60...70...80...90..100                                                    :01
        Elapsed Time: 00:02:33
76Transfer Rate, CPS: 4704                                                                                   0
550     Window Slots: 1 of 1                                                                              %767
332      Packet Type: B                                                                                2105674
3108    Packet Count: 53006                                                                               1226

hecksum errorr Count: 84                                                                                   (r
esend)
        Last Message: SUCCESS.  Files: 1, Bytes: 732904, 4787 CPS


(/work/tbot2go/tbot/) C-Kermit>connect
Connecting to /dev/ttyUSB0, speed 115200
 Escape character: Ctrl-\ (ASCII 28, FS): enabled
Type the escape character followed by C to get back,
or followed by ? to see other options.
----------------------------------------------------
CACHE: Misaligned operation at range [80000000, 800b2ee8]
```

```
## Total Size      = 0x000b2ee8 = 732904 Bytes
## Start Addr      = 0x80000000
=>


=> imi 80000000

## Checking Image at 80000000 ...
   FIT image found
   FIT description: Firmware image with one or more FDT blobs
   Created:         2017-09-30   5:16:48 UTC
    Image 0 (firmware@1)
     Description:  U-Boot 2017.09-00396-g6ca43a5 for am335x board
     Created:      2017-09-30   5:16:48 UTC
     Type:         Firmware
     Compression:  uncompressed
     Data Start:   unavailable
     Data Size:    unavailable
     Architecture: ARM
     Load Address: 0x80800000
    Image 1 (fdt@1)
     Description:  am335x-evm
     Created:      2017-09-30   5:16:48 UTC
     Type:         Firmware
     Compression:  uncompressed
     Data Start:   unavailable
     Data Size:    unavailable
     Architecture: ARM
     Load Address: unavailable
    Image 2 (fdt@2)
     Description:  am335x-bone
     Created:      2017-09-30   5:16:48 UTC
     Type:         Firmware
     Compression:  uncompressed
     Data Start:   unavailable
     Data Size:    unavailable
     Architecture: ARM
     Load Address: unavailable
    Image 3 (fdt@3)
     Description:  am335x-boneblack
     Created:      2017-09-30   5:16:48 UTC
     Type:         Firmware
     Compression:  uncompressed
     Data Start:   unavailable
     Data Size:    unavailable
     Architecture: ARM
     Load Address: unavailable
    Image 4 (fdt@4)
     Description:  am335x-evmsk
     Created:      2017-09-30   5:16:48 UTC
     Type:         Firmware
     Compression:  uncompressed
     Data Start:   unavailable
     Data Size:    unavailable
     Architecture: ARM
     Load Address: unavailable
    Image 5 (fdt@5)
     Description:  am335x-bonegreen
     Created:      2017-09-30   5:16:48 UTC
     Type:         Firmware
     Compression:  uncompressed
     Data Start:   unavailable
     Data Size:    unavailable
     Architecture: ARM
     Load Address: unavailable
    Image 6 (fdt@6)
     Description:  am335x-icev2
     Created:      2017-09-30   5:16:48 UTC
     Type:         Firmware
     Compression:  uncompressed
     Data Start:   unavailable
     Data Size:    unavailable
     Architecture: ARM
     Load Address: unavailable
    Default Configuration: 'conf@1'
    Configuration 0 (conf@1)
     Description:  am335x-evm
     Kernel:       unavailable
```

```
     FDT:          fdt@1
    Configuration 1 (conf@2)
     Description:  am335x-bone
     Kernel:       unavailable
     FDT:          fdt@2
    Configuration 2 (conf@3)
     Description:  am335x-boneblack
     Kernel:       unavailable
     FDT:          fdt@3
    Configuration 3 (conf@4)
     Description:  am335x-evmsk
     Kernel:       unavailable
     FDT:          fdt@4
    Configuration 4 (conf@5)
     Description:  am335x-bonegreen
     Kernel:       unavailable
     FDT:          fdt@5
    Configuration 5 (conf@6)
     Description:  am335x-icev2
     Kernel:       unavailable
     FDT:          fdt@6
## Checking hash(es) for FIT Image at 80000000 ...
   Hash(es) for Image 0 (firmware@1):  error!
Can't get image data/size for '' hash node in 'firmware@1' image node
Bad hash in FIT image!
=>
```

loads - load S-Record file over serial line

```
=> help loads
loads - load S-Record file over serial line

Usage:
loads [ off ]
    - load S-Record file over serial line with offset 'off'
=>
```

rarpboot- boot image via network using RARP/TFTP protocol

```
=> help rarp
Unknown command 'rarp' - try 'help' without arguments for list of all known commands

=>
```

tftpboot- boot image via network using TFTP protocol

```
=> help tftp
tftpboot - boot image via network using TFTP protocol

Usage:
tftpboot [loadAddress] [[hostIPaddr:]bootfilename]
=>
```

Environment Variables Commands

printenv- print environment variables

```
=> help printenv
printenv - print environment variables

Usage:
printenv [-a]
    - print [all] values of all environment variables
printenv name ...
    - print value of environment variable 'name'
=>
```

The printenv command prints one, several or all variables of the U-Boot environment. When arguments are given, these are interpreted as the names of environment variables which will be printed with their values:

```
=> printenv ipaddr hostname netmask
ipaddr=192.168.3.20
hostname=bbb
netmask=255.255.255.0
=>
```

Without arguments, printenv prints all a list with all variables in the environment and their values, plus some statistics about the current usage and the total size of the memory available for the environment.

```
=> printenv
addcon=setenv bootargs ${bootargs} console=${console}
addip=setenv bootargs ${bootargs} ip=${ipaddr}:${serverip}:${gatewayip}:${netmask}:${hostname}:${netdev}::off panic=1
addmisc=setenv bootargs ${bootargs} loglevel=8
addmtd=setenv bootargs ${bootargs} ${mtdparts}
arch=arm
args_mmc=run finduuid;setenv bootargs console=${console} ${optargs} root=PARTUUID=${uuid} rw rootfstype=${mmcrootfstype}
baudrate=115200
board=am335x
board_name=A335BNLT
board_rev=00C0
board_serial=414BBBK0180
boot_a_script=load ${devtype} ${devnum}:${distro_bootpart} ${scriptaddr} ${prefix}${script}; source ${scriptaddr}
boot_efi_binary=load ${devtype} ${devnum}:${distro_bootpart} ${kernel_addr_r} efi/boot/bootarm.efi; if fdt addr\
  ${fdt_addr_r}; then bootefi ${kernel_addr_r} ${fdt_addr_r};else bootefi ${kernel_addr_r} ${fdtcontroladdr};fi
boot_extlinux=sysboot ${devtype} ${devnum}:${distro_bootpart} any ${scriptaddr} ${prefix}extlinux/extlinux.conf
boot_fdt=try
boot_fit=0
boot_net_usb_start=usb start
boot_prefixes=/ /boot/
boot_script_dhcp=boot.scr.uimg
boot_scripts=boot.scr.uimg boot.scr
boot_targets=mmc0 legacy_mmc0 mmc1 legacy_mmc1 nand0 pxe dhcp
bootcmd=if test ${boot_fit} -eq 1; then run update_to_fit;fi;run findfdt; run init_console; run envboot; run distro_bootcmd
bootcmd_dhcp=run boot_net_usb_start; if dhcp ${scriptaddr} ${boot_script_dhcp}; then source ${scriptaddr}; fi;setenv\
  efi_fdtfile ${fdtfile}; if test -z "${fdtfile}" -a -n "${soc}"; then setenv efi_fdtfile ${soc}-${board}${boardver}.dtb; fi;
  setenv efi_old_vci ${bootp_vci};setenv efi_old_arch ${bootp_arch};setenv bootp_vci PXEClient:Arch:00010:UNDI:003000;setenv\
  bootp_arch 0xa;if dhcp ${kernel_addr_r}; then tftpboot ${fdt_addr_r} dtb/${efi_fdtfile};if fdt addr ${fdt_addr_r}; then\
  bootefi ${kernel_addr_r} ${fdt_addr_r}; else bootefi ${kernel_addr_r} ${fdtcontroladdr};fi;fi;setenv bootp_vci\
  ${efi_old_vci};setenv bootp_arch ${efi_old_arch};setenv efi_fdtfile;setenv efi_old_arch;setenv efi_old_vci;
bootcmd_legacy_mmc0=setenv mmcdev 0; setenv bootpart 0:2 ; run mmcboot
bootcmd_legacy_mmc1=setenv mmcdev 1; setenv bootpart 1:2 ; run mmcboot
bootcmd_mmc0=setenv devnum 0; run mmc_boot
bootcmd_mmc1=setenv devnum 1; run mmc_boot
bootcmd_nand=run nandboot
bootcmd_pxe=run boot_net_usb_start; dhcp; if pxe get; then pxe boot; fi
bootcount=6
bootdelay=2
bootdir=/boot
bootenvfile=uEnv.txt
bootfile=beagleboneblack/tbot/zImage
```

```
bootm_size=0x10000000
bootpart=0:2
bootscript=echo Running bootscript from mmc${mmcdev} ...; source ${loadaddr}
cmp_addr_r=82000000
cmp_mlo=tftp ${cmp_addr_r} ${mlofile};cmp.b ${load_addr_r} ${cmp_addr_r} ${filesize}
cmp_uboot=tftp ${cmp_addr_r} ${ubfile};cmp.b ${load_addr_r} ${cmp_addr_r} ${filesize}
console=ttyS0,115200n8
cpu=armv7
dfu_alt_info_emmc=rawemmc raw 0 3751936
dfu_alt_info_mmc=boot part 0 1;rootfs part 0 2;MLO fat 0 1;MLO.raw raw 0x100 0x100;u-boot.img.raw raw 0x300 0x1000;u-env.raw\
  raw 0x1300 0x200;spl-os-args.raw raw 0x1500 0x200;spl-os-image.raw raw 0x1700 0x6900;spl-os-args fat 0 1;spl-os-image fat 0
  1;u-boot.img fat 0 1;uEnv.txt fat 0 1
dfu_alt_info_nand=SPL part 0 1;SPL.backup1 part 0 2;SPL.backup2 part 0 3;SPL.backup3 part 0 4;u-boot part 0 5;u-boot-spl-os\
  part 0 6;kernel part 0 8;rootfs part 0 9
dfu_alt_info_ram=kernel ram 0x80200000 0x4000000;fdt ram 0x80f80000 0x80000;ramdisk ram 0x81000000 0x4000000
distro_bootcmd=for target in ${boot_targets}; do run bootcmd_${target}; done
efi_dtb_prefixes=/ /dtb/ /dtb/current/
envboot=mmc dev ${mmcdev}; if mmc rescan; then echo SD/MMC found on device ${mmcdev};if run loadbootscript; then run\
  bootscript;else if run loadbootenv; then echo Loaded env from ${bootenvfile};run importbootenv;fi;if test -n $uenvcmd; then
  echo Running uenvcmd ...;run uenvcmd;fi;fi;fi;
eth1addr=6c:ec:eb:83:40:33
ethaddr=6c:ec:eb:83:40:31
fdt_addr_r=0x88000000
fdtaddr=0x88000000
fdtcontroladdr=9df21ed8
fdtfile=beagleboneblack/tbot/am335x-boneblack.dtb
filesize=b2ee8
findfdt=if test $board_name = A335BONE; then setenv fdtfile am335x-bone.dtb; fi; if test $board_name = A335BNLT; then setenv\
  fdtfile am335x-boneblack.dtb; fi; if test $board_name = BBBW; then setenv fdtfile am335x-boneblack-wireless.dtb; fi; if tes
  $board_name = BBG1; then setenv fdtfile am335x-bonegreen.dtb; fi; if test $board_name = BBGW; then setenv fdtfile am335x-\
  bonegreen-wireless.dtb; fi; if test $board_name = BBBL; then setenv fdtfile am335x-boneblue.dtb; fi; if test $board_name =\
  A33515BB; then setenv fdtfile am335x-evm.dtb; fi; if test $board_name = A335X_SK; then setenv fdtfile am335x-evmsk.dtb; fi;
  if test $board_name = A335_ICE; then setenv fdtfile am335x-icev2.dtb; fi; if test $fdtfile = undefined; then echo WARNING:\
  Could not determine device tree to use; fi;
finduuid=part uuid mmc ${bootpart} uuid
fit_bootfile=fitImage
fit_loadaddr=0x87000000
hostname=bbb
importbootenv=echo Importing environment from mmc${mmcdev} ...; env import -t ${loadaddr} ${filesize}
init_console=if test $board_name = A335_ICE; then setenv console ttyO3,115200n8;else setenv console ttyO0,115200n8;fi;
ipaddr=192.168.3.20
kernel_addr_r=0x82000000
load_addr_r=81000000
load_efi_dtb=load ${devtype} ${devnum}:${distro_bootpart} ${fdt_addr_r} ${prefix}${efi_fdtfile}
load_mlo=tftp ${load_addr_r} ${mlofile}
load_uboot=tftp ${load_addr_r} ${ubfile}
loadaddr=0x82000000
loadbootenv=fatload mmc ${mmcdev} ${loadaddr} ${bootenvfile}
loadbootscript=load mmc ${mmcdev} ${loadaddr} boot.scr
loadfdt=load ${devtype} ${bootpart} ${fdtaddr} ${bootdir}/${fdtfile}
loadfit=run args_mmc; bootm ${loadaddr}#${fdtfile};
loadimage=load ${devtype} ${bootpart} ${loadaddr} ${bootdir}/${bootfile}
loadramdisk=load mmc ${mmcdev} ${rdaddr} ramdisk.gz
mlofile=beagleboneblack/tbot/MLO
mmc_boot=if mmc dev ${devnum}; then setenv devtype mmc; run scan_dev_for_boot_part; fi
mmc_mmc=run mmcloadk; run mmcloadfdt;run args_mmc;bootz ${loadaddr} - ${fdtaddr}
mmcboot=mmc dev ${mmcdev}; setenv devnum ${mmcdev}; setenv devtype mmc; if mmc rescan; then echo SD/MMC found on device\
  ${mmcdev};if run loadimage; then if test ${boot_fit} -eq 1; then run loadfit; else run mmcloados;fi;fi;fi;
mmcdev=0
mmcloadfdt=ext2load mmc 0:2 ${fdtaddr} /boot/am335x-boneblack.dtb
mmcloadk=ext2load mmc 0:2 ${loadaddr} /boot/zImage
mmcloados=run args_mmc; if test ${boot_fdt} = yes || test ${boot_fdt} = try; then if run loadfdt; then bootz ${loadaddr} -\
  ${fdtaddr}; else if test ${boot_fdt} = try; then bootz; else echo WARN: Cannot load the DT; fi; fi; else bootz; fi;
mmcrootfstype=ext4 rootwait
mtdids=nand0=nand.0
mtdparts=mtdparts=nand.0:128k(NAND.SPL),128k(NAND.SPL.backup1),128k(NAND.SPL.backup2),128k(NAND.SPL.backup3),256k(NAND.u\
  -boot-spl-os),1m(NAND.u-boot),128k(NAND.u-boot-env),128k(NAND.u-boot-env.backup1),8m(NAND.kernel),-(NAND.file-system)
nandargs=setenv bootargs console=${console} ${optargs} root=${nandroot} rootfstype=${nandrootfstype}
nandboot=echo Booting from nand ...; run nandargs; nand read ${fdtaddr} NAND.u-boot-spl-os; nand read ${loadaddr}\
  NAND.kernel; bootz ${loadaddr} - ${fdtaddr}
nandroot=ubi0:rootfs rw ubi.mtd=NAND.file-system,2048
nandrootfstype=ubifs rootwait=1
net_nfs=run netloadimage; run netloadfdt;run nfsargs addcon addip addmtd addmisc;bootz ${loadaddr} - ${fdtaddr}
netargs=setenv bootargs console=${console} ${optargs} root=/dev/nfs nfsroot=${serverip}:${rootpath},${nfsopts} rw ip=dhcp
netboot=echo Booting from network ...; setenv autoload no; dhcp; run netloadimage; run netloadfdt; run netargs; bootz\
  ${loadaddr} - ${fdtaddr}
netdev=eth0
netloadfdt=tftp ${fdtaddr} ${fdtfile}
```

```
netloadimage=tftp ${loadaddr} ${bootfile}
netmask=255.255.255.0
netmmcboot=echo Booting from network ... with mmcargs ...; setenv autoload no; run netloadimage; run netloadfdt; run\
  args_mmc; bootz ${loadaddr} - ${fdtaddr}
nfsargs=setenv bootargs ${bootargs} root=/dev/nfs rw nfsroot=${serverip}:${rootpath},${nfsopts}
nfsopts=nfsvers=3 nolock rw
partitions=uuid_disk=${uuid_gpt_disk};name=rootfs,start=2MiB,size=-,uuid=${uuid_gpt_rootfs}
pxefile_addr_r=0x80100000
ramargs=setenv bootargs console=${console} ${optargs} root=${ramroot} rootfstype=${ramrootfstype}
ramboot=echo Booting from ramdisk ...; run ramargs; bootz ${loadaddr} ${rdaddr} ${fdtaddr}
ramdisk_addr_r=0x88080000
ramroot=/dev/ram0 rw
ramrootfstype=ext2
rdaddr=0x88080000
rootpath=/work/tbot2go/tbot/nfs/bbb
scan_dev_for_boot=echo Scanning ${devtype} ${devnum}:${distro_bootpart}...; for prefix in ${boot_prefixes}; do run\
  scan_dev_for_extlinux; run scan_dev_for_scripts; done;run scan_dev_for_efi;
scan_dev_for_boot_part=part list ${devtype} ${devnum} -bootable devplist; env exists devplist || setenv devplist 1; for\
  distro_bootpart in ${devplist}; do if fstype ${devtype} ${devnum}:${distro_bootpart} bootfstype; then run scan_dev_for_boot
  fi; done
scan_dev_for_efi=setenv efi_fdtfile ${fdtfile}; if test -z "${fdtfile}" -a -n "${soc}"; then setenv efi_fdtfile\
  ${soc}-${board}${boardver}.dtb; fi; for prefix in ${efi_dtb_prefixes}; do if test -e ${devtype} ${devnum}:${distro_bootpart
  ${prefix}${efi_fdtfile}; then run load_efi_dtb; fi;done;if test -e ${devtype} ${devnum}:${distro_bootpart}\
  efi/boot/bootarm.efi; then echo Found EFI removable media binary efi/boot/bootarm.efi; run boot_efi_binary; echo EFI LOAD\
  FAILED: continuing...; fi; setenv efi_fdtfile
scan_dev_for_extlinux=if test -e ${devtype} ${devnum}:${distro_bootpart} ${prefix}extlinux/extlinux.conf; then echo Found\
  ${prefix}extlinux/extlinux.conf; run boot_extlinux; echo SCRIPT FAILED: continuing...; fi
scan_dev_for_scripts=for script in ${boot_scripts}; do if test -e ${devtype} ${devnum}:${distro_bootpart} ${prefix}${script};
  then echo Found U-Boot script ${prefix}${script}; run boot_a_script; echo SCRIPT FAILED: continuing...; fi; done
scriptaddr=0x80000000
serverip=192.168.3.1
soc=am33xx
spiargs=setenv bootargs console=${console} ${optargs} root=${spiroot} rootfstype=${spirootfstype}
spiboot=echo Booting from spi ...; run spiargs; sf probe ${spibusno}:0; sf read ${loadaddr} ${spisrcaddr} ${spiimgsize};\
  bootz ${loadaddr}
spibusno=0
spiimgsize=0x362000
spiroot=/dev/mtdblock4 rw
spirootfstype=jffs2
spisrcaddr=0xe0000
static_ip=${ipaddr}:${serverip}:${gatewayip}:${netmask}:${hostname}::off
stderr=serial@44e09000
stdin=serial@44e09000
stdout=serial@44e09000
tbot_cmp_spl=run cmp_mlo
tbot_cmp_uboot=run cmp_uboot
tbot_upd_spl=run load_mlo;run upd_mlo
tbot_upd_uboot=run load_uboot;run upd_uboot
ubfile=beagleboneblack/tbot/u-boot.img
upd_mlo=fatwrite mmc 1:1 ${load_addr_r} mlo ${filesize}
upd_uboot=fatwrite mmc 1:1 ${load_addr_r} u-boot.img ${filesize}
update_to_fit=setenv loadaddr ${fit_loadaddr}; setenv bootfile ${fit_bootfile}
usb_boot=usb start; if usb dev ${devnum}; then setenv devtype usb; run scan_dev_for_boot_part; fi
vendor=ti
ver=U-Boot 2017.09-00396-g6ca43a5 (Sep 30 2017 - 07:16:03 +0200)

Environment size: 11032/131068 bytes
=>
```

saveenv - save environment variables to persistent storage

```
=> help saveenv
saveenv - save environment variables to persistent storage

Usage:
saveenv
=>
```

All changes you make to the U-Boot environment are made in RAM only. They are lost as soon as you reboot the system. If you want to make your changes permanent you have to use the saveenv command to write a copy of the environment settings to persistent storage, from where they are automatically loaded during startup:

```
=> saveenv
Saving Environment to FAT...
writing uboot.env
FAT: Misaligned buffer address (9df01d58)
done
=>
```

setenv - set environment variables

```
=> help setenv
setenv - set environment variables

Usage:
setenv [-f] name value ...
    - [forcibly] set environment variable 'name' to 'value ...'
setenv [-f] name
    - [forcibly] delete environment variable 'name'
=>
```

To modify the U-Boot environment you have to use the setenv command. When called with exactly one argument, it will delete any variable of that name from U-Boot's environment, if such a variable exists. Any storage occupied for such a variable will be automatically reclaimed:

```
=> setenv foo This is an example value.
=> printenv foo
foo=This is an example value.
=> setenv foo
=> printenv foo
## Error: "foo" not defined
=>
```

When called with more arguments, the first one will again be the name of the variable, and all following arguments will (concatenated by single space characters) form the value that gets stored for this variable. New variables will be automatically created, existing ones overwritten.

```
=> printenv bar
## Error: "bar" not defined
=> setenv bar This is a new example.
=> printenv bar
bar=This is a new example.
=> setenv bar
=>
```

Remember standard shell quoting rules when the value of a variable shall contain characters that have a special meaning to the command line parser (like the $ character that is used for variable substitution or the semicolon which separates commands). Use the backslash () character to escape such special characters, or enclose the whole phrase in apstrophes ('). Use "${name}" for variable expansion.

```
=> setenv cons_opts 'console=tty0 console=ttyS0,\${baudrate}'
=> printenv cons_opts
cons_opts=console=tty0 console=ttyS0,${baudrate}
=> setenv cons_opts
=>
```

There is no restriction on the characters that can be used in a variable name except the restrictions imposed by the command line parser (like using backslash for quoting, space and tab characters to separate arguments, or semicolon and newline to separate commands). Even strange input like =-/|()+= is a perfectly legal variable name in U-Boot.

A common mistake is to write

```
setenv name=value
```

instead of

```
setenv name value
```

There will be no error message, which lets you believe everything went OK, but it didn't: instead of setting the variable name to the value value you tried to delete a variable with the name name=value - this is probably not what you intended! Always remember that name and value have to be separated by space and/or tab characters!

Flattened Device Tree support

U-Boot is capable of quite comprehensive handling of the flattened device tree blob, implemented by the fdt family of commands:

```
=> help fdt
fdt - flattened device tree utility commands

Usage:
fdt addr [-c]  <addr> [<length>]   - Set the [control] fdt location to <addr>
fdt apply <addr>                    - Apply overlay to the DT
fdt move   <fdt> <newaddr> <length> - Copy the fdt to <addr> and make it active
fdt resize [<extrasize>]            - Resize fdt to size + padding to 4k addr + some optional <extrasize> if needed
fdt print  <path> [<prop>]          - Recursive print starting at <path>
fdt list   <path> [<prop>]          - Print one level starting at <path>
fdt get value <var> <path> <prop>   - Get <property> and store in <var>
fdt get name <var> <path> <index>   - Get name of node <index> and store in <var>
fdt get addr <var> <path> <prop>    - Get start address of <property> and store in <var>
fdt get size <var> <path> [<prop>]  - Get size of [<property>] or num nodes and store in <var>
fdt set    <path> <prop> [<val>]    - Set <property> [to <val>]
fdt mknode <path> <node>            - Create a new node after <path>
fdt rm     <path> [<prop>]          - Delete the node or <property>
fdt header                          - Display header info
fdt bootcpu <id>                    - Set boot cpuid
fdt memory <addr> <size>            - Add/Update memory node
fdt rsvmem print                    - Show current mem reserves
fdt rsvmem add <addr> <size>        - Add a mem reserve
fdt rsvmem delete <index>           - Delete a mem reserves
fdt chosen [<start> <end>]          - Add/update the /chosen branch in the tree
                                        <start>/<end> - initrd start/end addr
NOTE: Dereference aliases by omitting the leading '/', e.g. fdt print ethernet0.
=>
```

fdt addr - select FDT to work on

First, the blob that is to be operated on should be stored in memory, and U-Boot has to be informed about its location by the fdt addr command. Once this command has been issued, all subsequent fdt handling commands will use the blob stored at the given address. This address can be changed later on by issuing fdt addr or fdt move command. Here's how to load the blob into memory and tell U-Boot its location:

```
=> setenv fdt_addr_r 0x80000000
=> tftpb ${fdt_addr_r} beagleboneblack/tbot/u-boot.dtb
link up on port 0, speed 100, full duplex
Using ethernet@4a100000 device
TFTP from server 192.168.3.1; our IP address is 192.168.3.20
Filename 'beagleboneblack/tbot/u-boot.dtb'.
Load address: 0x80000000
Loading: *■##########
       862.3 KiB/s
done
Bytes transferred = 45056 (b000 hex)
=>
=> fdt addr ${fdt_addr_r}
=>
```

fdt list - print one level

Having selected the device tree stored in the blob just loaded, we can inspect its contents. As an FDT usually is quite extensive, it is easier to get information about the structure by looking at selected levels rather than full hierarchies. fdt list allows us to do exactly this. Let's have a look at the hierarchy one level below the cpus node:

```
=> fdt list /cpus
cpus {
	#address-cells = <0x00000001>;
	#size-cells = <0x00000000>;
	cpu@0 {
	};
};
=>
```

fdt print - recursive print

To print a complete subtree we use fdt print. In comparison to the previous example it is obvious that the whole subtree is printed:

```
=> fdt print /cpus
cpus {
	#address-cells = <0x00000001>;
	#size-cells = <0x00000000>;
	cpu@0 {
		compatible = "arm,cortex-a8";
		device_type = "cpu";
		reg = <0x00000000>;
		operating-points = <0x000afc80 0x00139b88 0x000927c0 0x0012b128 0x0007a120 0x00112a88 0x00043238 0x00112a88>;
		voltage-tolerance = <0x00000002>;
		clocks = <0x00000002>;
		clock-names = "cpu";
		clock-latency = <0x000493e0>;
		cpu0-supply = <0x00000003>;
	};
};
=>
```

fdt mknode - create new nodes

fdt mknode can be used to attach a new node to the tree. We will use the fdt list command to verify that the new node has been created and that it is empty:

```
=> fdt list /
/ {
	#address-cells = <0x00000001>;
	#size-cells = <0x00000001>;
	compatible = "ti,am335x-evm", "ti,am33xx";
	interrupt-parent = <0x00000001>;
	model = "TI AM335x EVM";
	chosen {
	};
	aliases {
	};
	memory {
	};
	cpus {
	};
	pmu {
	};
	soc {
	};
	ocp {
	};
	fixedregulator@0 {
	};
	fixedregulator@1 {
	};
	fixedregulator@2 {
	};
	matrix_keypad@0 {
	};
	volume_keys@0 {
```

```
        };
        backlight {
        };
        panel {
        };
        sound {
        };
};
=> fdt mknode / testnode
=> fdt list /
/ {
        #address-cells = <0x00000001>;
        #size-cells = <0x00000001>;
        compatible = "ti,am335x-evm", "ti,am33xx";
        interrupt-parent = <0x00000001>;
        model = "TI AM335x EVM";
        testnode {
        };
        chosen {
        };
        aliases {
        };
        memory {
        };
        cpus {
        };
        pmu {
        };
        soc {
        };
        ocp {
        };
        fixedregulator@0 {
        };
        fixedregulator@1 {
        };
        fixedregulator@2 {
        };
        matrix_keypad@0 {
        };
        volume_keys@0 {
        };
        backlight {
        };
        panel {
        };
        sound {
        };
};
=> fdt list /testnode
testnode {
};
=>
```

fdt set - set node properties

Now, let's create a property at the newly created node; again we'll use fdt list for verification:

```
=> fdt set /testnode testprop
=> fdt set /testnode testprop testvalue
=> fdt list /testnode
testnode {
        testprop = "testvalue";
};
=>
```

fdt rm - remove nodes or properties

The fdt rm command is used to remove nodes and properties. Let's delete the test property created in the previous paragraph and verify the results:

```
=> fdt rm /testnode testprop
=> fdt list /testnode
testnode {
};
=> fdt rm /testnode
=> fdt list /
/ {
      #address-cells = <0x00000001>;
      #size-cells = <0x00000001>;
      compatible = "ti,am335x-evm", "ti,am33xx";
      interrupt-parent = <0x00000001>;
      model = "TI AM335x EVM";
      chosen {
      };
      aliases {
      };
      memory {
      };
      cpus {
      };
      pmu {
      };
      soc {
      };
      ocp {
      };
      fixedregulator@0 {
      };
      fixedregulator@1 {
      };
      fixedregulator@2 {
      };
      matrix_keypad@0 {
      };
      volume_keys@0 {
      };
      backlight {
      };
      panel {
      };
      sound {
      };
};
=>
```

fdt move - move FDT blob to new address

To move the blob from one memory location to another we will use the fdt move command. Besides moving the blob, it makes the new address the "active" one - similar to fdt addr:

```
=> fdt move ${fdt_addr_r} 0x8000b000L b000
=>
=> fdt list /
/ {
      #address-cells = <0x00000001>;
      #size-cells = <0x00000001>;
      compatible = "ti,am335x-evm", "ti,am33xx";
      interrupt-parent = <0x00000001>;
      model = "TI AM335x EVM";
      chosen {
      };
      aliases {
      };
      memory {
      };
      cpus {
      };
      pmu {
```

```
	};
	soc {
	};
	ocp {
	};
	fixedregulator@0 {
	};
	fixedregulator@1 {
	};
	fixedregulator@2 {
	};
	matrix_keypad@0 {
	};
	volume_keys@0 {
	};
	backlight {
	};
	panel {
	};
	sound {
	};
};
=> fdt mknod / foobar
=> fdt list /
/ {
	#address-cells = <0x00000001>;
	#size-cells = <0x00000001>;
	compatible = "ti,am335x-evm", "ti,am33xx";
	interrupt-parent = <0x00000001>;
	model = "TI AM335x EVM";
	foobar {
	};
	chosen {
	};
	aliases {
	};
	memory {
	};
	cpus {
	};
	pmu {
	};
	soc {
	};
	ocp {
	};
	fixedregulator@0 {
	};
	fixedregulator@1 {
	};
	fixedregulator@2 {
	};
	matrix_keypad@0 {
	};
	volume_keys@0 {
	};
	backlight {
	};
	panel {
	};
	sound {
	};
};
=>
=> fdt addr ${fdt_addr_r}
=> fdt list /
/ {
	#address-cells = <0x00000001>;
	#size-cells = <0x00000001>;
	compatible = "ti,am335x-evm", "ti,am33xx";
	interrupt-parent = <0x00000001>;
	model = "TI AM335x EVM";
	chosen {
	};
	aliases {
	};
	memory {
	};
```

```
        cpus {
        };
        pmu {
        };
        soc {
        };
        ocp {
        };
        fixedregulator@0 {
        };
        fixedregulator@1 {
        };
        fixedregulator@2 {
        };
        matrix_keypad@0 {
        };
        volume_keys@0 {
        };
        backlight {
        };
        panel {
        };
        sound {
        };
};
=>
```

fdt chosen - fixup dynamic info

One of the modifications made by U-Boot to the blob before passing it to the kernel is the addition of the /chosen node. Linux 2.6 Documentation/powerpc/booting-without-of.txt says that this node is used to store "some variable environment information, like the arguments, or the default input/output devices." To force U-Boot to add the /chosen node to the current blob, fdt chosen command can be used. Let's now verify its operation:

```
=> fdt list /
/ {
        #address-cells = <0x00000001>;
        #size-cells = <0x00000001>;
        compatible = "ti,am335x-evm", "ti,am33xx";
        interrupt-parent = <0x00000001>;
        model = "TI AM335x EVM";
        chosen {
        };
        aliases {
        };
        memory {
        };
        cpus {
        };
        pmu {
        };
        soc {
        };
        ocp {
        };
        fixedregulator@0 {
        };
        fixedregulator@1 {
        };
        fixedregulator@2 {
        };
        matrix_keypad@0 {
        };
        volume_keys@0 {
        };
        backlight {
        };
        panel {
        };
        sound {
```

```
	};
};
=> fdt chosen
=> fdt list /
/ {
	#address-cells = <0x00000001>;
	#size-cells = <0x00000001>;
	compatible = "ti,am335x-evm", "ti,am33xx";
	interrupt-parent = <0x00000001>;
	model = "TI AM335x EVM";
	chosen {
	};
	aliases {
	};
	memory {
	};
	cpus {
	};
	pmu {
	};
	soc {
	};
	ocp {
	};
	fixedregulator@0 {
	};
	fixedregulator@1 {
	};
	fixedregulator@2 {
	};
	matrix_keypad@0 {
	};
	volume_keys@0 {
	};
	backlight {
	};
	panel {
	};
	sound {
	};
};
=> fdt list /chosen
chosen {
	stdout-path = "/ocp/serial@44e09000";
	tick-timer = "/ocp/timer@48040000";
};
=>
```

Note: fdt boardsetup performs board-specific blob updates, most commonly setting clock frequencies, etc. Discovering its operation is left as an excercise for the reader.

Special Commands

i2c - I2C sub-system

```
=> help i2c
i2c - I2C sub-system

Usage:
i2c bus [muxtype:muxaddr:muxchannel] - show I2C bus info
crc32 chip address[.0, .1, .2] count - compute CRC32 checksum
i2c dev [dev] - show or set current I2C bus
i2c loop chip address[.0, .1, .2] [# of objects] - looping read of device
i2c md chip address[.0, .1, .2] [# of objects] - read from I2C device
i2c mm chip address[.0, .1, .2] - write to I2C device (auto-incrementing)
i2c mw chip address[.0, .1, .2] value [count] - write to I2C device (fill)
i2c nm chip address[.0, .1, .2] - write to I2C device (constant address)
i2c probe [address] - test for and show device(s) on the I2C bus
i2c read chip address[.0, .1, .2] length memaddress - read to memory
i2c write memaddress chip address[.0, .1, .2] length [-s] - write memory
          to I2C; the -s option selects bulk write in a single transaction
i2c flags chip [flags] - set or get chip flags
i2c olen chip [offset_length] - set or get chip offset length
i2c reset - re-init the I2C Controller
i2c speed [speed] - show or set I2C bus speed
=>
```

Miscellaneous Commands

echo - echo args to console

```
=> help echo
echo - echo args to console

Usage:
echo [args..]
    - echo args to console; \c suppresses newline
=>
```

The echo command echoes the arguments to the console:

```
=> echo The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
=>
```

reset - Perform RESET of the CPU

```
=> help reset
reset - Perform RESET of the CPU

Usage:
reset
=>
```

The reset command reboots the system.

```
=> reset

resetting ...

U-Boot SPL 2017.09-00396-g6ca43a5 (Sep 30 2017 - 07:16:03)
Trying to boot from MMC2
*** Warning - bad CRC, using default environment

reading u-boot.img
reading u-boot.img
reading u-boot.img
reading u-boot.img


U-Boot 2017.09-00396-g6ca43a5 (Sep 30 2017 - 07:16:03 +0200)

CPU  : AM335X-GP rev 2.1
Model: TI AM335x BeagleBone Black
DRAM:  512 MiB
NAND:  0 MiB
MMC:   OMAP SD/MMC: 0, OMAP SD/MMC: 1
*** Warning - bad CRC, using default environment

ERROR: No USB device found

at drivers/usb/gadget/ether.c:2709/usb_ether_init()
<ethaddr> not set. Validating first E-fuse MAC
Net:   CACHE: Misaligned operation at range [9df2f580, 9df2f624]
eth0: ethernet@4a100000
Hit any key to stop autoboot:  2 ■■■ 0
=>
```

sleep - delay execution for some time

```
=> help sleep
sleep - delay execution for some time

Usage:
sleep N
    - delay execution for N seconds (N is _decimal_ and can be
      fractional)
=>
```

The sleep command pauses execution for the number of seconds given as the argument:

```
=> sleep 5
=>
```

version - print monitor version

```
=> help version
version - print monitor, compiler and linker version

Usage:
version
=>
```

You can print the version and build date of the U-Boot image running on your system using the version command (short: vers):

```
=> version
U-Boot 2017.09-00396-g6ca43a5 (Sep 30 2017 - 07:16:03 +0200)

arm-unknown-linux-gnueabi-gcc (GCC) 4.9.0
GNU ld (GNU Binutils) 2.24
=>
```

? - alias for 'help'

You can use ? as a short form for the help command (see description above).

U-Boot Environment Variables

The U-Boot environment is a block of memory that is kept on persistent storage and copied to RAM when U-Boot starts. It is used to store environment variables which can be used to configure the system. The environment is protected by a CRC32 checksum.

This section lists the most important environment variables, some of which have a special meaning to U-Boot. You can use these variables to configure the behaviour of U-Boot to your liking.

- autoload: if set to "no" (or any string beginning with 'n'), the rarpb, bootp or dhcp commands will perform only a configuration lookup from the BOOTP / DHCP server, but not try to load any image using TFTP.
- autostart: if set to "yes", an image loaded using the rarpb, bootp, dhcp, tftp, disk, or docb commands will be automatically started (by internally calling the bootm command).
- baudrate: a decimal number that selects the console baudrate (in bps). Only a predefined list of baudrate settings is available. When you change the baudrate (using the "setenv baudrate ..." command), U-Boot will switch the baudrate of the console terminal and wait for a newline which must be entered with the new speed setting. This is to make sure you can actually type at the new speed. If this fails, you have to reset the board (which will operate at the old speed since you were not able to saveenv the new settings.) If no "baudrate" variable is defined, the default baudrate of 115200 is used.
- bootargs: The contents of this variable are passed to the Linux kernel as boot arguments (aka "command line").
- bootcmd: This variable defines a command string that is automatically executed when the initial countdown is not interrupted. This command is only executed when the variable bootdelay is also defined!
- bootdelay: After reset, U-Boot will wait this number of seconds before it executes the contents of the bootcmd variable. During this time a countdown is printed, which can be interrupted by pressing any key. Set this variable to 0 boot without delay. Be careful: depending on the contents of your bootcmd variable, this can prevent you from entering interactive commands again forever! Set this variable to -1 to disable autoboot.
- bootfile: name of the default image to load with TFTP
- ethaddr: Ethernet MAC address for first/only ethernet interface (= eth0 in Linux). This variable can be set only once (usually during manufacturing of the board). U-Boot refuses to delete or overwrite this variable once it has been set.
- eth1addr: Ethernet MAC address for second ethernet interface (= eth1 in Linux).
- eth2addr: Ethernet MAC address for third ethernet interface (= eth2 in Linux).

  ...

- initrd_high: used to restrict positioning of initrd ramdisk images: If this variable is not set, initrd images will be copied to the highest possible address in RAM; this is usually what you want since it allows for maximum initrd size. If for some reason you want to make sure that the initrd image is loaded below the CFG_BOOTMAPSZ limit, you can set this environment variable to a value of "no" or "off" or "0". Alternatively, you can set it to a maximum upper address to use (U-Boot will still check that it does not overwrite the U-Boot stack and data). For instance, when you have a system with 16 MB RAM, and want to reserve 4 MB from use by Linux, you can do this by adding "mem=12M" to the value of the "bootargs" variable. However, now you must make sure that the initrd image is placed in the first 12 MB as well - this can be done with

```
=> setenv initrd_high 00c00000
```

Setting initrd_high to the highest possible address in your system (0xFFFFFFFF) prevents U-Boot from copying the image to RAM at all. This allows for faster boot times, but requires a Linux kernel with zero-copy ramdisk support.

- ipaddr: IP address; needed for tftp command
- loadaddr: Default load address for commands like tftp or loads.
- loads_echo: If set to 1, all characters received during a serial download (using the loads command) are echoed back. This might be needed by some terminal emulations (like cu), but may as well just take time on others.
- mtdparts: This variable (usually defined using the mtdparts command) allows to share a common MTD partition scheme between U-Boot and the Linux kernel.
- pram: If the "Protected RAM" feature is enabled in your board's configuration, this variable can be defined to enable the reservation of such "protected RAM", i. e. RAM which is not overwritten by U-Boot. Define this variable to hold the number of kB you want to reserve for pRAM. Note that the board info structure will still show the full amount of RAM. If pRAM is reserved, a new environment variable "mem" will automatically be defined to hold the amount of remaining RAM in a form that can be passed as boot argument to Linux, for instance like that:

```
=> setenv bootargs ${bootargs} mem=\${mem}
=> saveenv
```

This way you can tell Linux not to use this memory, either, which results in a memory region that will not be affected by reboots.

- serverip: TFTP server IP address; needed for tftp command.
- serial#: contains hardware identification information such as type string and/or serial number. This variable can be set only once (usually during manufacturing of the board). U-Boot refuses to delete or overwrite this variable once it hass been set.
- silent: If the configuration option CONFIG_SILENT_CONSOLE has been enabled for your board, setting this variable to any value will suppress all console messages. Please see doc/README.silent for details.
- 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.

The following environment variables may be used and automatically updated by the network boot commands (bootp, dhcp, or tftp), depending the information provided by your boot server:

- bootfile: see above
- dnsip: IP address of your Domain Name Server
- gatewayip: IP address of the Gateway (Router) to use
- hostname: Target hostname
- ipaddr: see above
- netmask: Subnet Mask
- rootpath: Pathname of the root filesystem on the NFS server
- serverip: see above
- filesize: Size (as hex number in bytes) of the file downloaded using the last bootp, dhcp, or tftp command.

U-Boot Scripting Capabilities

U-Boot allows to store commands or command sequences in a plain text file. Using the mkimage tool you can then convert this file into a script image which can be executed using U-Boot's source command, see section source - run script from memory.

Hint: maximum flexibility can be achieved if you are using the Hush shell as command interpreter in U-Boot, see How the Command Line Parsing Works

How the Command Line Parsing Works

There are two different command line parsers available with U-Boot: the old "simple" one, and the much more powerful "hush" shell:

Old, simple command line parser

- supports environment variables (through setenv / saveenv commands)
- several commands on one line, separated by ';'
- **variable substitution using "... ${_variablename_} ..." syntax**

  NOTE: Older versions of U-Boot used "$(...)" for variable substitution. Support for this syntax is still present in current versions, but will be removed soon. Please use "${...}" instead, which has the additional benefit that your environment definitions are compatible with the Hush shell, too.
- special characters ('$', ';') can be escaped by prefixing with '', for example:

```
setenv bootcmd bootm \${address}
```

- You can also escape text by enclosing in single apostrophes, for example:

```
setenv addip 'setenv bootargs ${bootargs} ip=${ipaddr}:${serverip}:${gatewayip}:${netmask}:${hostname}:${netdev}:off'
```

Hush shell

- similar to Bourne shell, with control structures like if...then...else...fi, for...do...done, while...do...done, until...do...done, ...
- supports environment ("global") variables (through setenv / saveenv commands) and local shell variables (through standard shell syntax name=value); only environment variables can be used with the run command, especially as the variable to run (i. e. the first argument).
- In the current implementation, the local variables space and global environment variables space are separated. Local variables are those you define by simply typing like name=value. To access a local variable later on, you have to write '$name' or '${name}'; to execute the contents of a variable directly you can type '$name' at the command prompt. Note that local variables can only be used for simple commands, not for compound commands etc.

- Global environment variables are those you can set and print using setenv and printenv. To run a command stored in such a variable, you need to use the run command, and you must not use the '$' sign to access them.
- To store commands and special characters in a variable, use single quotation marks surrounding the whole text of the variable, instead of the backslashes before semicolons and special symbols.
- Be careful when using the hash ('#') character - like with a "real" Bourne shell it is the comment character, so you have to escape it when you use it in the value of a variable.

Examples:

```
setenv bootcmd bootm \$address
setenv addip 'setenv bootargs $bootargs ip=$ipaddr:$serverip:$gatewayip:$netmask:$hostname:$netdev:off'
```

Hush shell scripts

Here are a few examples for the use of the advanced capabilities of the hush shell in U-Boot environment variables or scripts:

```
=> setenv check 'if imi $addr; then echo Image OK; else echo Image corrupted!!; fi'
=> print check
check=if imi $addr; then echo Image OK; else echo Image corrupted!!; fi
=> addr=0x80000000 ; run check

## Checking Image at 80000000 ...
   FIT image found
Bad FIT image format!
Image corrupted!!
=>
=> addr=0x80100000L ; run check

## Checking Image at 80100000 ...
   Legacy image found
   Image Name:   autoscr example script
   Created:      2017-09-30   5:24:22 UTC
   Image Type:   PowerPC Linux Script (uncompressed)
   Data Size:    157 Bytes = 157 Bytes
   Load Address: 00000000
   Entry Point:  00000000
   Contents:
      Image 0: 149 Bytes = 149 Bytes
   Verifying Checksum ... OK
Image OK
=>
```

Instead of "echo Image OK" there could be a command (sequence) to boot or otherwise deal with the correct image; instead of the "echo Image corrupted!!" there could be a command (sequence) to (load and) boot an alternative image, etc.

For Example:

```
=> addr1=0x80000000
=> addr2=0x80100000L
=> bootm $addr1 || bootm $addr2 || tftp 0x80000000 beagleboneblack/tbot/source_example.scr && imi 0x80000000
Wrong Image Format for bootm command
ERROR: can't get kernel image!
bootm - boot application image from memory

Usage:
bootm [addr [arg ...]]
    - boot application image stored in memory
      passing arguments 'arg ...'; when booting a Linux kernel,
      'arg' can be the address of an initrd image
      When booting a Linux kernel which requires a flat device-tree
      a third argument is required which is the address of the
```

```
      device-tree blob. To boot that kernel without an initrd image,
      use a '-' for the second argument. If you do not pass a third
      a bd_info struct will be passed instead

For the new multi component uImage format (FIT) addresses
      must be extended to include component or configuration unit name:
      addr:<subimg_uname> - direct component image specification
      addr#<conf_uname>   - configuration specification
      Use iminfo command to get the list of existing component
      images and configurations.

Sub-commands to do part of the bootm sequence.  The sub-commands must be
issued in the order below (it's ok to not issue all sub-commands):
      start [addr [arg ...]]
      loados  - load OS image
      ramdisk - relocate initrd, set env initrd_start/initrd_end
      fdt     - relocate flat device tree
      cmdline - OS specific command line processing/setup
      bdt     - OS specific bd_t processing
      prep    - OS specific prep before relocation or go
      go      - start OS
link up on port 0, speed 100, full duplex
Using ethernet@4a100000 device
TFTP from server 192.168.3.1; our IP address is 192.168.3.20
Filename 'beagleboneblack/tbot/source_example.scr'.
Load address: 0x80000000
Loading: *■#
       43 KiB/s
done
Bytes transferred = 221 (dd hex)

## Checking Image at 80000000 ...
   Legacy image found
   Image Name:   autoscr example script
   Created:      2017-09-30   5:24:22 UTC
   Image Type:   PowerPC Linux Script (uncompressed)
   Data Size:    157 Bytes = 157 Bytes
   Load Address: 00000000
   Entry Point:  00000000
   Contents:
      Image 0: 149 Bytes = 149 Bytes
   Verifying Checksum ... OK
=>
```

This will check if the image at address "addr1" is ok and boot it; if the image is not ok, the alternative image at address "addr2" will be checked and booted if it is found to be OK. If both images are missing or corrupted, a new image will be loaded over TFTP and checked with imi.

General rules

1. If a command line (or an environment variable executed by a run command) contains several commands separated by semicolons, and one of these commands fails, the remaining commands will still be executed.
2. If you execute several variables with one call to run (i. e. calling run with a list of variables as arguments), any failing command will cause run to terminate, i. e. the remaining variables are not executed.

U-Boot pytest suite

Read more of how to setup test/py in U-Boot source code:

http://git.denx.de/?p=u-boot.git;a=blob;f=test/py/README.md

Test py results

```
$ PATH=/home/hs/data/Entwicklung/test_py_scripts/hook-scripts:$PATH;PYTHONPATH=/work/hs/tb
ot/u-boot-am335x_evm;./test/py/test.py --bd am335x_evm -s --build-dir .
+u-boot-test-flash am335x_evm na
board type  am335x_evm
No flashing supported yet
========================================================================================== test session starts ==========
================================================================================
platform linux2 -- Python 2.7.8, pytest-3.2.1, py-1.4.34, pluggy-0.4.0
rootdir: $TBOT_BASEDIR/u-boot-am335x_evm/test/py, inifile: pytest.ini
collecting 0 items
collecting 1 item
[...]
Unknown command 'non_existent_cmd' - try 'help'
=>
test/py/tests/test_unknown_cmd.py .
test/py/tests/test_ut.py ss
test/py/tests/test_vboot.py s

================================================================================ 74 passed, 20 skipped in 12.71 seconds
================================================================================
$
```

links