Use U-Boot

U-Boot Main Commands

setenv this command is used to set variables
saveenv this command saves variables previously set in the environment permanent storage space
printenv this command print the current variables

The `help` command show a brief summary of the built-in commands of U-Boot. Here is a selection of useful commands:

```
U-Boot> help
?       - alias for 'help'
boot    - 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
erase   - erase FLASH memory
fatinfo - print information about filesystem
fdt     - flattened device tree utility commands
flinfo  - print FLASH memory information
go      - start application at address 'addr'
help    - print command description/usage
md      - memory display
mm      - memory modify (auto-incrementing address)
mmc     - MMC sub system
mmcinfo - display MMC info
nand    - NAND sub-system
ping    - send ICMP ECHO_REQUEST to network host
printenv- print environment variables
protect - enable or disable FLASH write protection
reset   - Perform RESET of the CPU
run     - run commands in an environment variable
saveenv - save environment variables to persistent storage
setenv  - set environment variables
sf      - SPI flash sub-system
tftpboot- boot image via network using TFTP protocol
version - print monitor, compiler and linker version
U-Boot>
```

Refer to the U-Boot manual page for the command line interface.

U-Boot script capability

You can create script or complex variables, which prevents you to type commands. Here is a summary of several variables built to make a network loading of linux easier :

```
setenv boot_addr 0x21400000
setenv linux 'tftp $(boot_addr) linux-2.6.x.img'
setenv ramdisk_addr 0x21100000
setenv ramdisk 'tftp $(ramdisk_addr) sam9-ramdisk.gz'
setenv go 'run linux; run ramdisk; bootm $(boot_addr)'
saveenv
```

The `setenv linux 'tftp $(boot_addr) linux-2.6.x.img'` line is equivalent of typing `tftp 0x21400000 linux-2.6.x.img` but combined with others and stored in flash, it allows you to save time, and automate. For executing a Linux kernel bootup, using this snippet, simply type `run go`

Boot pre-defined variables and command:

bootcmd when set, this variable content is executed automatically after the boot delay. It enables the U-Boot autoboot mode
bootargs this variable it used as an exchange area to pass information to the main application started by U-Boot (Linux kernel for instance)
bootm this command executes an application generated by the *mkimage* tool

Load Linux with U-Boot on AT91 boards

This section describes the loading of a Linux kernel and its root file system. Keep in mind useful U-Boot commands to setup your U-Boot behavior.

Preparing linux image (optional)

If you want to use an uImage file with U-Boot, you can use the *mkimage* tool which encapsulates kernel image with header information, CRC32 checksum, etc.

mkimage comes in source code with U-Boot distribution and it is built during U-Boot compilation (u-boot-source-dir/tools/mkimage).

See U-Boot README file for more information.

Command to generate an uncompressed uImage file (5) :

```
mkimage -A arm -O linux -C none -T kernel -a 20008000 -e 20008000 -n linux-2.6 -d arch/arm/boot/Image uImage
```

Commands to generate a compressed uImage file (6) :

```
mkimage -A arm -O linux -C none -T kernel -a 20008000 -e 20008000 -n linux-2.6 -d arch/arm/boot/zImage uImage
```

Preparing Kernel DTB image

For latest Linux kernel, it supports the Device Tree Binary which describes the hardware in a binary file.

U-Boot can load both the DTB and kernel. The only change is running `bootm` or `bootz` with two arguments:

```
bootm 0x22000000 - 0x21000000
```
or
```
bootz 0x22000000 - 0x21000000
```
First argument is the address in memory of the Linux kernel, second one is the address of the DTB binary.

Loading through network

On a development system, it is useful to get the kernel and root file system through the network. U-Boot provides support for loading binaries from a remote host on the network using the TFTP protocol.

To manage to use TFTP with U-Boot, you will have to configure a TFTP server on your host machine. Check your distribution manual or Internet resources to configure a Linux or Windows TFTP server on your host:

- U-Boot documentation on a Linux host
- another TFTP configuration reference

On the U-Boot side, you will have to setup the networking parameters:

1. setup an Ethernet address (MAC address)
   Check this U-Boot network BuildRootFAQ entry to choose a proper MAC address.
   `setenv ethaddr 3e:36:65:ba:6f:be`
2. setup IP parameters
   - the board ip address
     `setenv ipaddr 10.159.245.180`
   - the server ip address where the TFTP server is running
     `setenv serverip 10.159.245.186`
3. saving Environment to flash
   `saveenv`
4. if Ethernet Phy has not been detected during former bootup, reset the board to reload U-Boot : the Ethernet address and Phy initialization shall be ok, now
5. download the Linux *uImage* and the root file system to a ram location using the U-Boot `tftp` command (Cf. U-Boot script capability chapter).
6. launch Linux issuing a `bootm` or `boot` command.

ℹ If the board has both emac and gmac, you can use following to choose which one to use:

```
setenv ethact macb0,gmacb0
setenv ethprime gmacb0
```

Build U-Boot from sources

To build the binary found above, you will have to go through the following steps.

Getting U-Boot sources

Dedicated page on U-Boot wiki: http://www.denx.de/wiki/U-Boot/SourceCode

You can easily download U-Boot source code from Linux4SAM GitHub U-Boot repository:

- clone the Linux4sam GitHub U-Boot repository
- `$ git clone git://github.com/linux4sam/u-boot-at91.git`
- `Cloning into 'u-boot-at91'...`
- `remote: Counting objects: 219350, done.`
- `remote: Compressing objects: 100% (40142/40142), done.`
- `remote: Total 219350 (delta 175755), reused 219350 (delta 175755)`

- `Receiving objects: 100% (219350/219350), 56.01 MiB | 1.24 MiB/s, done.`
- `Resolving deltas: 100% (175755/175755), done.`
- `$ cd u-boot-at91`

- The source code has been taken from the *master* branch which is pointing to the latest branch we use. If you want to use the other branch, you can list them and use one of them by doing:
- `$ git branch -r`
- `origin/HEAD -> origin/master`
- `origin/master`
- `origin/u-boot-2012.10-at91`
- `origin/u-boot-2013.07-at91`
- `origin/u-boot-2014.07-at91`
- `origin/uboot_5series_1.x`
- `$ git checkout origin/u-boot-2014.07-at91 -b u-boot-2014.07-at91`
- `Branch u-boot-2014.07-at91 set up to track remote branch u-boot-2014.07-at91 from origin.`
- `Switched to a new branch 'u-boot-2014.07-at91'`

Choosing where the U-Boot environment resides

Above, we talked about the location of the U-Boot environment.

☞Go to the top-level `boards.cfg` file to find the exact target when invoking *make*.

```
$ grep  -e "at91sam" -e "sama5" boards.cfg | awk '{print $7}'
at91sam9260ek_dataflash_cs0
at91sam9260ek_dataflash_cs1
at91sam9260ek_nandflash
at91sam9g20ek_2mmc_nandflash
at91sam9g20ek_dataflash_cs0
at91sam9g20ek_dataflash_cs1
at91sam9g20ek_mmc
at91sam9g20ek_nandflash
at91sam9xeek_dataflash_cs0
at91sam9xeek_dataflash_cs1
at91sam9xeek_nandflash
at91sam9261ek_dataflash_cs0
at91sam9261ek_dataflash_cs3
at91sam9261ek_nandflash
at91sam9g10ek_dataflash_cs0
at91sam9g10ek_dataflash_cs3
at91sam9g10ek_nandflash
at91sam9263ek_dataflash
at91sam9263ek_dataflash_cs0
at91sam9263ek_nandflash
at91sam9263ek_norflash
at91sam9263ek_norflash_boot
at91sam9m10g45ek_mmc
at91sam9m10g45ek_nandflash
at91sam9n12ek_mmc
at91sam9n12ek_nandflash
at91sam9n12ek_spiflash
at91sam9rlek_mmc
at91sam9rlek_dataflash
at91sam9rlek_nandflash
```

```
at91sam9x5ek_dataflash
at91sam9x5ek_mmc
at91sam9x5ek_nandflash
at91sam9x5ek_spiflash
sama5d3_xplained_mmc
sama5d3_xplained_nandflash
sama5d3xek_mmc
sama5d3xek_nandflash
sama5d3xek_spiflash
sama5d4ek_mmc
sama5d4ek_nandflash
sama5d4ek_spiflash
```

Here is the way to fit the location of the environment to your needs:

To put environment variables in serial flash:

```
   make sama5d3xek_serialflash_config
```

To put environment variables in nandflash (default):

```
   make sama5d3xek_nandflash_config
```

To put environment variables in SD/MMC card:

```
   make sama5d3xek_sdcard_config
```

Cross-compiling U-Boot

Once the AT91 U-Boot sources available, cross-compile U-Boot is made in two steps : configuration and compiling. Check the Configuration chapter in U-Boot reference manual.

Here are the building steps for the SAMA5D3x-EK board:

```
make distclean
make sama5d3xek_nandflash_config
make CROSS_COMPILE=<path_to_cross-compiler/cross-compiler-prefix->
```

path_to_cross-compiler is only needed if it is not in your `PATH`.
Usually *cross-compiler-prefix-* looks like *arm-linux-* or *arm-elf-*

The result of these operations is a fresh U-Boot binary called `u-boot.bin` corresponding to the binary ELF file `u-boot`.

- `u-boot.bin` is the file you should store on the board
- `u-boot` is the ELF format binary file you may use to debug U-Boot through a JTag link for instance.