Embedded Linux Workshop on Blueboard-AT91

B. Vasu Dev
vasu@easyarm.com

Bootloader

BOOT Sequence of BB_AT91

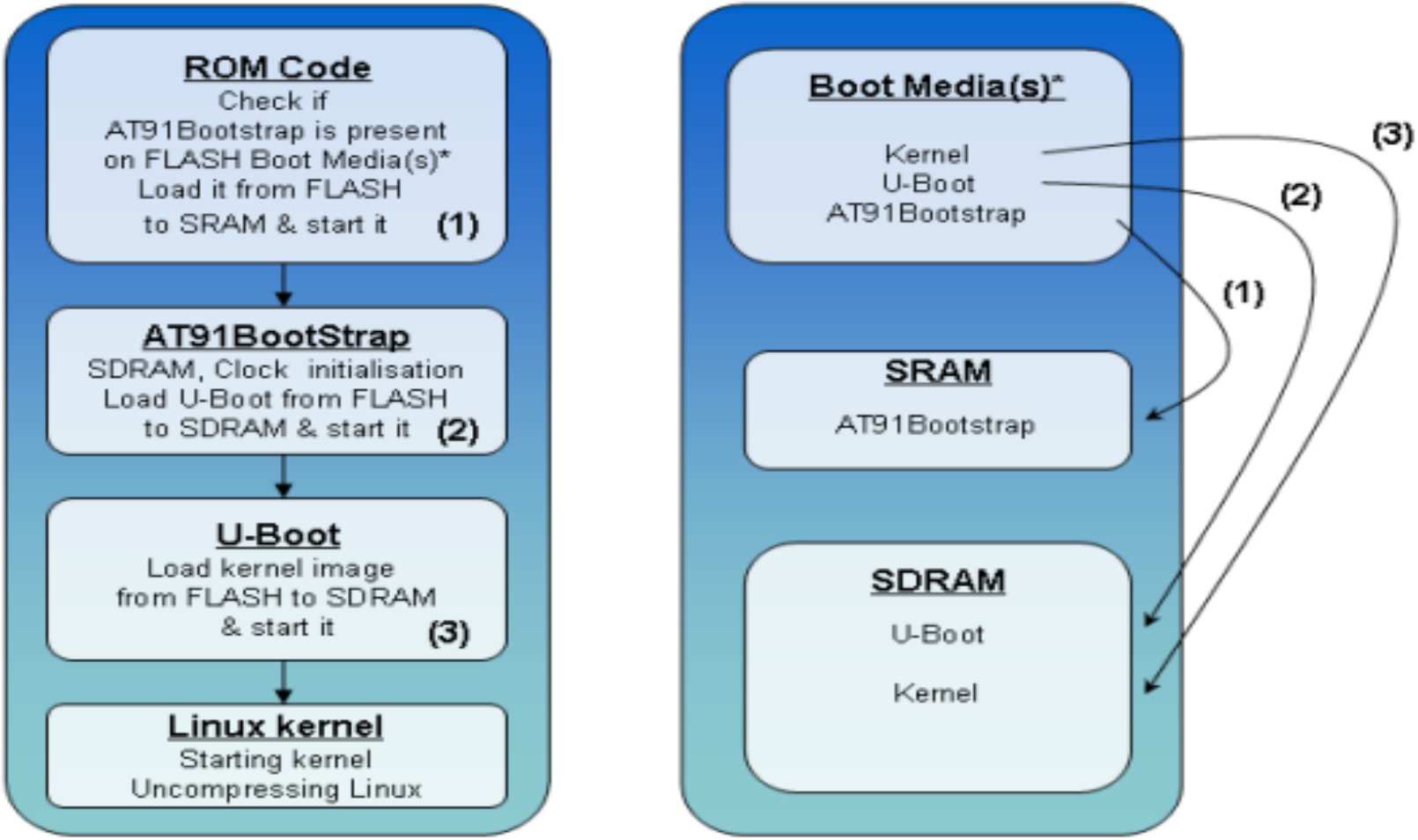
Boot Sequence:

* BMS Pin selects (AT91RM9200 Manual, 10.3.2.4, p.127)
  - o high: internal ROM (see [below])
  - o low: external 16bit non-volatile memory (see Manual)

* when using Internal ROM (AT91RM9200 Manual, 7. p.87)
  - o Sequence:
    1. try to load and execute tiny program from SPI DataFlash
    2. try to load and execute tiny program from TWI EEPROM
    3. try to load and execute tiny program from from 8bit parallel non-volatile memory
    4. otherwise use "DBGU serial interface" and "USB Device Port" to let someone (Xmodem or DFU protocol) load and execute a tiny program.

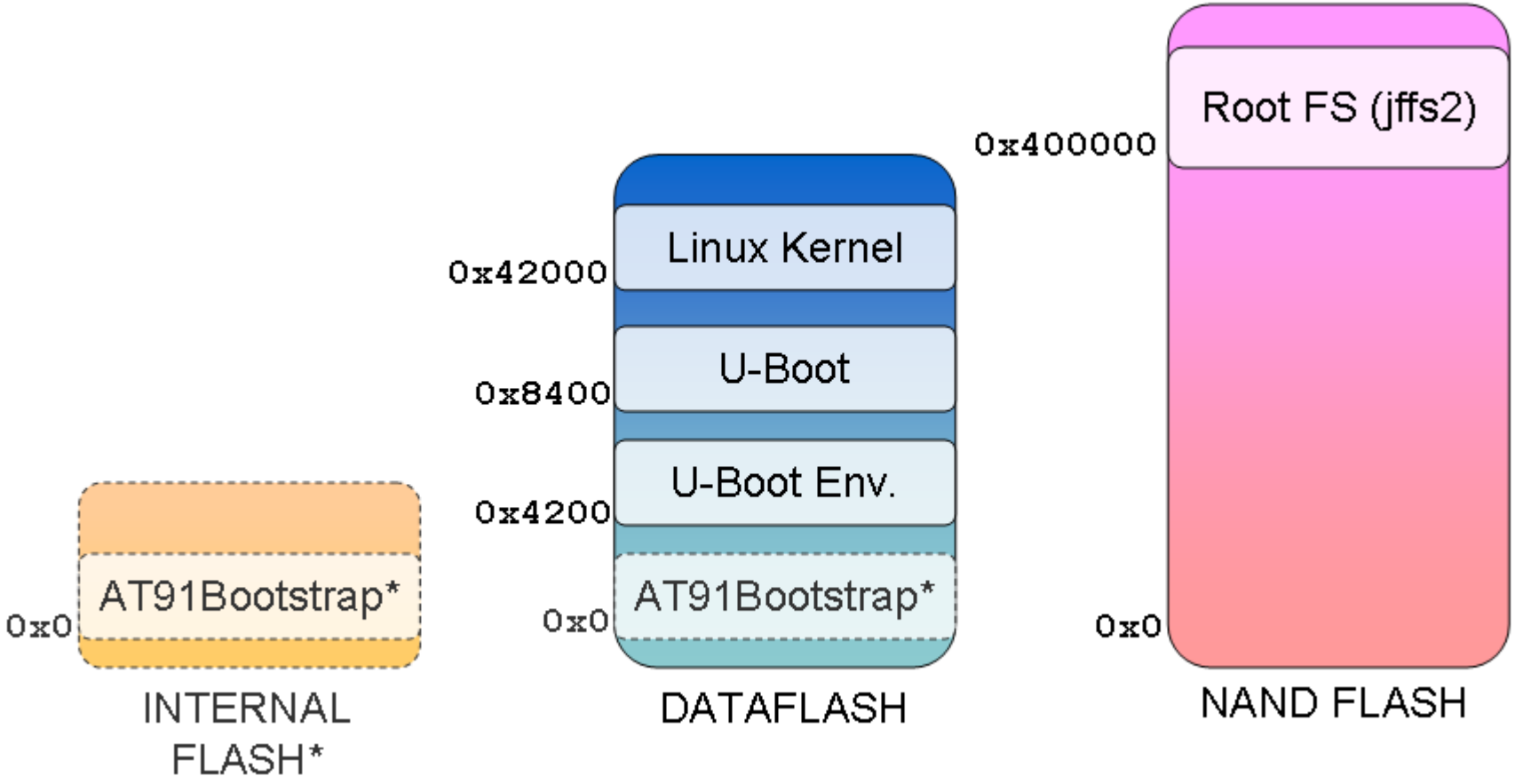
  - o tiny program means: program may be up to 12KByte (16K-4K)
  - o the program is loaded into internal sram and immediately started there.
  - o the tiny program can be
    - + a standalone application
    - + on BB_AT91 it is Darrel Loader

BOOT Sequence of BB_AT91

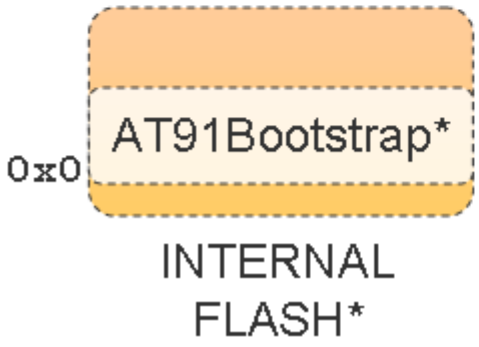


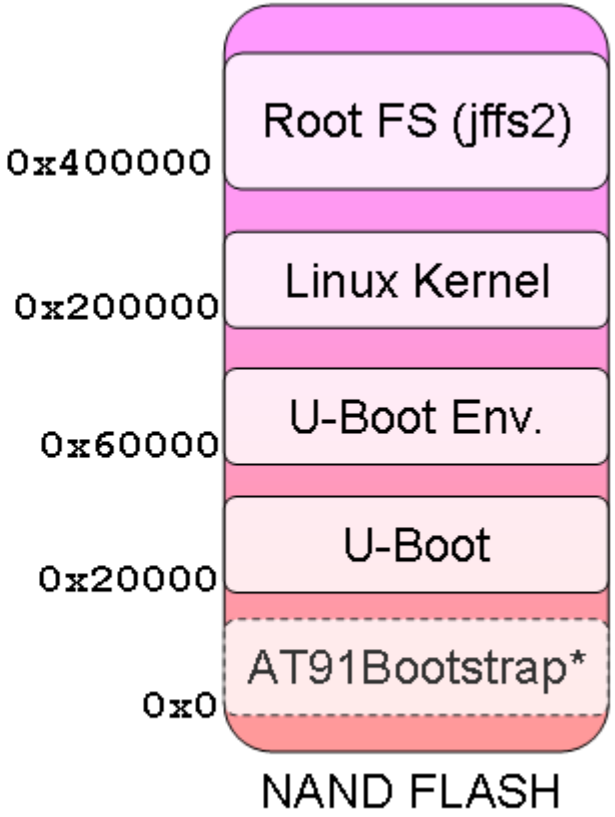
(*) Depends on AT91SAM9xxx FLASH capabilities (DataFlash, Nand Flash, SDCard...).
Please check your product datasheet;

Data Flash Layout Example



NAND Flash Layout Example





Bootloader

- The bootloader is a piece of code responsible for
  - Basic hardware initialization
  - Loading of an application binary, usually an operating system kernel, from flash storage, from the network, or from another type of non-volatile storage.
  - Possibly uncompression of the application binary
  - Execution of the application
- Besides these basic functions, most bootloaders provide a shell with various commands implementing different operations.
  - Loading of data from storage or network, memory inspection, hardware diagnostics and testing, etc.

Bootloaders

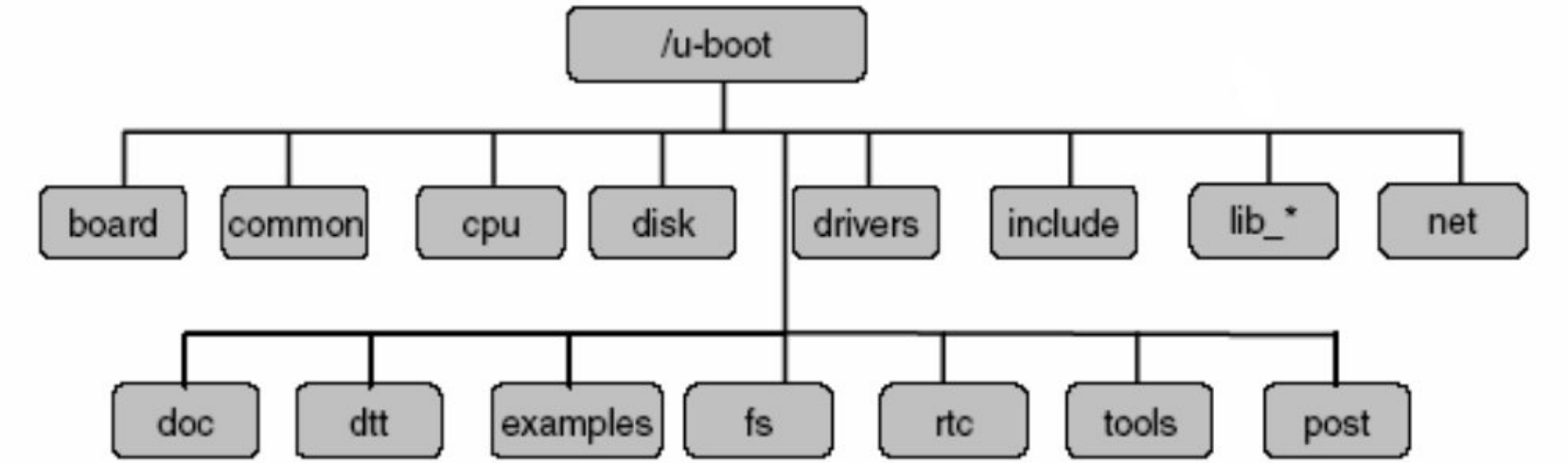
- There are several open-source generic bootloaders.
  Here are the most popular ones:
  - U-Boot, the universal bootloader by Denx
    The most used on ARM, also used on PPC, MIPS, x86, m68k, NIOS, etc. Clearly the most widely used community solution.
    http://www.denx.de/wiki/U-Boot
  - RedBoot, based on RedHat eCos
    http://sources.redhat.com/redboot/
- There are also a lot of other open-source or proprietary bootloaders, often architecture-specific

U-Boot

U-Boot is a typical free software project

- Freely available at http://www.denx.de/wiki/U-Boot
- Documentation available at http://www.denx.de/wiki/U-Boot/Documentation
- The latest development source code is available in a Git repository: http://git.denx.de/cgi-bin/gitweb.cgi?p=u-boot.git;a=summary
- Development and discussions happen around an open mailing-list http://lists.denx.de/pipermail/u-boot/
- Since the end of 2008, it follows a fixed-interval release schedule. Every two months, a new version is released. Versions are named YYYY.MM.

U-Boot Structure



U-Boot Porting

Create a new set of directories and files to mimic the existing configuration.

The necessary directories and files are:

u-boot/board/<boardname>

u-boot/include/configs/<boardname>.h

u-boot/Makefile

Then copy files, and make changes to Makefile and platform specific files to retarget the code for board.

include, cpu, lib_arch, and board are the directories dependent upon a particular architecture and the board. All other directories remain the same for any architecture.

U-Boot Porting

The include directory:

The include directory contains all header files globally used by u-boot. In this directory we need to generate or change following files for a specific cpu and board:

1) Configs/<boardname>.h

This file includes hardware configuration for memory map and peripherals. It contains the chip configuration, flash boot configuration, nor and NAND flash configuration, SDRAM memory configuration, serial configuration, u-boot configuration, I2C configuration, Linux configuration, Network and Ethernet configuration etc.

2) <core>.h (for example arm920t.h)

This file contains the processor specific definitions required for u-boot to access the CPU. This includes the register definitions.

U-Boot Porting

The cpu directory:

The cpu directory contains initialization routines for internal peripherals.

1) <core>/cpu.c

This file contains the cpu specific code. It performs operations like read and write to control register, reset cpu, I/D cache enable/disable, Setup the stacks for IRQ and FIQ etc.

2) <core>/interrupt.c

This file contains the function related to interrupt and timer. Like enable interrupts, disable interrupts, timer related operations etc.

3) <core>/start.S

This file contains startup code for ARM920T cpu core.

U-Boot Porting

The lib_<arch> directory :

4) board.c

This file performs the following functions:

* memory map initialization
  - Logical memory map configuration
  - dynamic memory allocation routines(malloc)
* u-boot peripheral initialization

5) <arch>linux.c for example armlinux.c

This file contains Implementation of bootm command, that is used to places the kernel image into RAM. This command passes the architecture number and boot parameters to the kernel and starts the kernel execution by giving a control to kernel image.

6) div0.c

This file contains division-by zero exception handler

U-Boot Porting

The board directory:

An individual directory for each board configuration supported by u-boot is found in the board directory. This sub-directory includes board specific configuration routines.

7) **< boardname >/flash.c**
This file contains the function related to flash memory like flash Initialization, Reset flash, print flash information, Erase flash, Write to flash

8) **< boardname >/ < boardname >_emac.c**
This file initializes EMAC and performs EMAC related operations like receive packet and send packet.

9) **< boardname >/ < boardname >.c**
This file contains functions like board initialization routine, Initialize Timer configuration register, Serial Port initialization, NAND flash initialization etc.

10) < boardname >/ soc.h
This file contains peripheral base addresses, Interrupt event ids etc.

11) **< boardname >/ platform.S**
This file contains the PLL, power domain initialization, SDRAM Memory configuration register like SDRAM control register, refresh rate control register, initialization or SDRAM etc.

U-Boot Porting

Modify the top level makefile to specify the new configuration.
All the configurations are listed under various titles.

```
<boardname>_config : unconfig
    @./mkconfig $(@:_config=) <architecture> <core> <boardname>
```

e.g, For ECB_AT91 board,

```
ecb_at91_config :       unconfig
      @$(MKCONFIG) $(@:_config=) arm arm920t ecb_at91
```

U-Boot Compiling

- Get the source code from the website, and uncompress it
- The `include/configs/` directory contains one configuration file for each supported board
  - It defines the CPU type, the peripherals and their configuration, the memory mapping, the U-Boot features that should be compiled in, etc.
  - It is a simple .h file that sets pre-processor constants. See the `README` file for the documentation of these constants.
- Assuming that your board is already supported by U-Boot, there should be one file corresponding to your board, for example `include/configs/ecb_at91.h`.

U-Boot Compiling

- U-Boot must be configured before being compiled
  - `make BOARDNAME_config`
  - Where `BOARDNAME` is the name of the configuration file in `include/configs/`, without the `.h`
  - **`make ecb_at91_config`**

  Make sure that the cross-compiler is available in `PATH`

  `export PATH=<CROSS_COMPILER_PATH>:$PATH`
- Compile U-Boot, by specifying the cross-compiler prefix.

  `make CROSS_COMPILE=<CROSS_CROSS_COMPILER_PREFIX>`

U-Boot Loading on BB_AT91

- On sucess of compilation of U-Boot u-boot.bin is generated which need to loaded on the target.
  - Reset the target and press enter key as soon you see the darrel loader screen.
  - Select option 2 'Target will wait for u-boot.bin on X-Modem'
  - Transfer the u-boot.bin from host 'minicom / hyperterminal'
  - Press Enter at end, which writes the data in flash.

Running U-Boot

- Connect the target to the host through a serial console
- Power-up the board. On the serial console, you will see something like:

```
U-Boot 1.1.6 (Aug  3 2004 - 17:31:20)
   RAM Configuration:
   Bank #0: 00000000  8 MB
   Flash:  4 MB
   In:     serial
   Out:    serial
   Err:    serial
   u-boot #
```

- The U-Boot shell offers a set of commands. We will study the most important ones, see the documentation for a complete reference or the `help` command.

U-Boot Commands

U-Boot Basic Command Set (1/4)

Information Commands

- **bdinfo** - print Board Info structure ¨
- **coninfo** - print console devices and informations
- **flinfo** - print FLASH memory information
- **iminfo** - print header information for application image
- **imls** - list all images found in flash
- **help** - print online help

Memory Commands

- **base** - print or set address offset
- **crc32** - checksum calculation
- **cmp** - memory compare
- **cp** - memory copy
- **md** - memory display
- **mm** - memory modify (auto-incrementing)
- **mtest** - simple RAM test
- **mw** - memory write (fill)
- **nm** - memory modify (constant address)

Flash Memory Commands

- cp - memory copy (program flash)
- flinfo - print FLASH memory information
- erase - erase FLASH memory
- protect - enable or disable FLASH write protection

Execution Control Commands

- autoscr - run script from memory
- bootm - boot application image from memory
- bootelf - Boot from an ELF image in memory
- bootvx - Boot vxWorks from an ELF image
- go - start application at address 'addr'

U-Boot Commands

U-Boot Basic Command Set (2/4)

Network Commands

- **bootp** - boot image via network using BOOTP/TFTP protocol
- **cdp** - Perform Cisco Discovery Protocol network configuration
- **dhcp** - invoke DHCP client to obtain IP/boot params
- **loadb** - load binary file over serial line (kermit mode)
- **loads** - load S-Record file over serial line
- **nfs** - boot image via network using NFS protocol
- **ping** - send ICMP ECHO_REQUEST to network host
- **rarpboot-** boot image via network using RARP/TFTP protocol
- **tftpboot-** boot image via network using TFTP protocol

Environment Variables Commands

- printenv- print environment variables
- saveenv - save environment variables to persistent storage
- askenv - get environment variables from stdin
- setenv - set environment variables
- run - run commands in an environment variable
- bootd - boot default, i.e., run 'bootcmd'

U-Boot Commands

U-Boot Basic Command Set (3/4)

Filesystem Support (FAT, cramfs, JFFS2, Reiser)

- **chpart** - change active partition
- **fsinfo** - print information about filesystems
- **fsload** - load binary file from a filesystem image
- **ls** - list files in a directory (default /)
- fatinfo - print information about filesystem
- **fatls** - list files in a directory (default /)
- **fatload** - load binary file from a dos filesystem
- **nand** - NAND flash sub-system
- **reiserls-** list files in a directory (default /)
- **reiserload-** load binary file from a Reiser filesystem

- *Special Commands*
  - i2c - I2C sub-system
  - doc - Disk-On-Chip sub-system
  - dtt - Digital Thermometer and Themostat
  - eeprom - EEPROM sub-system
  - fpga - FPGA sub-system
  - ide - IDE sub-system
  - kgdb - enter gdb remote debug mode
  - diskboot- boot from IDE device
  - icache - enable or disable instruction cache
  - dcache - enable or disable data cache
  - diag - perform board diagnostics (*POST* code)
  - log - manipulate logbuffer
  - pci - list and access PCI Configuraton Space
  - regdump - register dump commands
  - usb - USB sub-system
  - sspi - SPI utility commands

U-Boot Commands

U-Boot Basic Command Set (4/4)

Miscellaneous Commands

- **bmp** - manipulate BMP image data ¨
- **date** - get/set/reset date & time ¨
- **echo** - echo args to console ¨
- **exit** - exit script ¨
- **kbd** - read keyboard status ¨
- **in** - read data from an IO port ¨
- **out** - write datum to IO port ¨
- **reset** - Perform RESET of the CPU ¨
- **sleep** - delay execution for some time ¨
- **test** - minimal test like /bin/sh ¨
- **version** - print monitor version ¨
- **wd** - check and set watchdog ¨
- **?** - alias for 'help' ¨

U-Boot Env Variables

U-Boot takes all the configurable data from the environment variables.

The most frequently used variables are:

1. bootcmd : Command executed to boot the kernel

2. bootargs : Boot arguments passed to the kernel

3. bootdelay: Delay before the kernel booting once uboot starts

4. ipaddr : IP address of the target

5. serverip : IP address of the server

6. filename : file name used while tftp download

U-Boot Env Variables

To boot from Kernel on Flash and Rootfs on SD Card:

```
ecb_at91> setenv bootargs 'mem=32M rootfstype=ext3 root=/dev/mmcblk0p1
console=ttyS0,115200n8 rootdelay=2'

ecb_at91> setenv bootcmd 'bootm 0xC0021840'

ecb_at91> saveenv
```

To boot from Kernel over TFTP and Rootfs over NFS:

```
ecb_at91> setenv bootargs 'mem=32M root=/dev/nfs nfsroot=192.168.0.128:/srv/nfsroot/rootfs
ip=192.168.0.135:192.168.0.128:192.168.0.1:255.255.255.0::eth0: rootdelay=2'

ecb_at91> setenv bootcmd 'tftp:bootm 0x20200000'

ecb_at91> saveenv
```

TFTP SETUP

1. Remove any other version of tftpd
   - **sudo apt-get remove tftpd atftpd**
2. Install the tftpd-hpa package
   - **sudo apt-get install tftpd-hpa**
3. Configure the tftp server
   - **vi /etc/default/tftpd-hpa**
   - Change RUN_DAEMON=”no” to RUN_DAEMON=”yes”
   - Change the default location /var/lib/tftpboot to **/tftpboot**
4. Create the TFTP directory in root
   - **sudo mkdir /tftpboot**
   - **sudo chmod 777 /tftpboot**
5. Restart the TFTP Server

   sudo /etc/init.d/tftpd-hpa restart

NFS SERVER SETUP

1. Install the NFS Server packages

```
sudo apt-get install nfs-kernel-server nfs-common portmap
```

2. While portmap installation prompt appears for "loopback" option set it to **no**. OR change it later

```
sudo dpkg-reconfigure portmap; sudo /etc/init.d/portmap restart
```

3. Configure the NFS Server

```
sudo mkdir /srv/nfsroot; sudo chmod 777 /srv/nfsroot
```

```
sudo vi /etc/exports
```

Add this line to specify the shared folder

```
/srv/nfsroot    *(rw,sync,no_subtree_check)
```

4. Restart the NFS Server

```
sudo /etc/init.d/nfs-kernel-server restart
```

5. Test: `mkdir nfstest; sudo mount localhost:/nfsroot  nfstest`