Loading Images with U-Boot

U-Boot is the bootloader used on the majority of EMAC ARM-based systems. It is loaded by the processor boot-ROM or low-level bootstrap code and performs device initialization followed by loading and starting the OS. EMAC has made modifications to U-Boot to add additional features when necessary. While generally used to load Linux, EMAC uses U-Boot to load Windows CE and other operating systems.

This section describes basic usage of U-Boot focusing on the process of loading and programming OS images using U-Boot. More information can be found on the U-Boot homepage at http://www.denx.de/wiki/U-Boot (http://www.denx.de/wiki/U-Boot)

Accessing the U-Boot Console

By default U-Boot is setup to automatically boot the OS after a set timeout (generally one second). The following steps will interrupt the boot process and start the U-Boot prompt:

1. Setup a serial connection with the target board using the settings specified for the hardware. Refer to the EMAC OE Getting Started Guide or other documentation for more information.
2. After a serial connection has been established, reboot the board.
3. When initial boot messages are printed to the serial terminal, press Enter in the serial terminal application. This should bring up the U-Boot prompt as seen below:

```
U-Boot 2009.06-rc1-svn1357 (Jul 08 2010 - 13:31:57)

EMAC Inc. SOM-9M10/G45M

DRAM:  128 MB
NAND:  256 MiB
DataFlash:AT45DB321
Nb pages:   8192
Page Size:    528
Size= 4325376 bytes
Logical address: 0xC0000000
Area 0: C0000000 to C00041FF (RO) Bootstrap
Area 1: C0004200 to C00083FF      Environment
Area 2: C0008400 to C0041FFF (RO) U-Boot
Area 3: C0042000 to C0251FFF      Kernel
Area 4: C0252000 to C041FFFF      FS
In:    serial
Out:   serial
Err:   serial
Net:   macb0
macb0: Starting autonegotiation...
macb0: Autonegotiation complete
macb0: link up, 100Mbps full-duplex (lpa: 0x45e1)
Hit any key to stop autoboot:  0
U-Boot>
```

4. U-Boot commands can be entered from this prompt. To see all available commands, type `help`. Individual command help can be accessed by running using the command name as an argument to the `help` command, `help printenv` for example. Available commands may differ between systems depending on implemented features.

The U-Boot Environment

The U-Boot environment is stored in a pre-determined area of non-volatile storage. This is generally the same device that stores the U-Boot image and Linux Kernel. On systems that use raw NAND flash as the primary storage device, there is generally secondary storage device used for low-level boot images such as U-Boot and the environment due to the complexities of bad-block handling with NAND flash. Devices using NOR flash typically use a portion of the NOR flash to store the bootloader and kernel. The `flinfo` command can provide an idea of the storage layout on the device, see below for an example:

```
U-Boot> flinfo
DataFlash:AT45DB321
Nb pages:   8192
Page Size:    528
Size= 4325376 bytes
Logical address: 0xC0000000
Area 0: C0000000 to C00041FF (RO) Bootstrap
Area 1: C0004200 to C00083FF      Environment
Area 2: C0008400 to C0041FFF (RO) U-Boot
Area 3: C0042000 to C0251FFF      Kernel
Area 4: C0252000 to C041FFFF      FS
```

Standard Environment Variables

U-Boot uses a set of environmental variables to control the operation of the system. There are standard variables that are used for specific purposes by the system. The user can also add and combine variables to customize the system operation. The `printenv` command is used to view the current environment, as shown below:

```
U-Boot> printenv
baudrate=115200
stdin=serial
stdout=serial
stderr=serial
ethact=macb0
ethaddr=00:50:C2:C9:2B:D6
bootdelay=1
bootargs=console=ttyS0,115200 root=/dev/mtdblock0 mtdparts=atmel_nand:128M(root),-(aux) ro rootfstype
bootcmd=cp.b 0xC0042000 0x70000000 0x210000; bootm 0x70000000
kernel_name=uImage-2.6.30
rootfs_name=rootfs.jffs2
filesize=130B194
fileaddr=70000000
ipaddr=10.0.2.24
serverip=10.0.2.60

Environment size: 771/16892 bytes
```

Table 1 below defines many of the variables above.

Table 1. Standard U-Boot Environmental Variables

Variable name	Purpose
baudrate	Serial console baud rate.
stdin	The device to use for the console standard input.
stdout	The device to use for the console standard output.
stderr	The device to use for the console standard error output.
ethact	Defines the Ethernet device.
ethaddr	The MAC address.*
bootdelay	The number of seconds to wait after loading before running the command stored in bootcmd.
bootargs	Boot argument string to be passed to Linux kernel on boot.

bootcmd	Command(s) to run after loading. This is generally used to load and boot the OS.
filesize	Automatically set to the size of the last file loaded to the system.
fileaddr	Automatically set to the logical address used for the last file loaded to the system.
ipaddr	The static IP address to use for network communications.
serverip	The IP address of the TFTP server on the local network to use for file transfer.

*: Once the ethaddr variable has been set, it cannot be erased without clearing the entire environment.

Accessing Variables

To access the value of a variable, use the variable name enclosed in ${}. For example, echo ${bootcmd} will print the current value of the bootcmd variable. One common use for this is storing variables that refer to the value of another variable, such as a filename.

Setting Variables

The setenv command is used to set the value of a new or existing variable. For example, the following code would set the ipaddr variable to a value of 192.168.2.1:

```
U-boot> setenv ipaddr 192.168.2.1
```

When U-Boot loads it makes a copy of the current environment into RAM. Using setenv only changes the volatile copy of the environment but does not commit the changes to flash. If a change needs to be committed to non-volatile memory, the saveenv command must be used to save the current environment. This will allow the changes to be preserved for the next time that U-Boot is loaded.

Transferring Files

Before programming an image to a board, it must be transferred to the system RAM in U-Boot. While this may be done using storage devices such as USB or SD cards, TFTP is the most common and efficient method. This requires that you have a TFTP server accessible on the local network. TFTP server setup is beyond the scope of this document. This section explains the process required to transfer a file using TFTP.

Configuration

In order to load a file using TFTP, U-Boot must be configured to access the local network. Static networking configuration is recommended, though DHCP can be used on some boards. Typically, the IP address and TFTP server IP address are the only settings that need to be defined. The netmask and broadcast will be determined from this information, and no default gateway is required if the server is on the same subnetwork as the board. Before continuing, determine a valid static network address for your local network; contact your IT department for more information if required. The example below shows how to set the IP address of the board to 192.168.2.2 and the TFTP server IP address to 192.168.2.1:

```
U-Boot> setenv ipaddr 192.168.2.2
U-Boot> setenv serverip 192.168.2.1
U-Boot> saveenv
Saving Environment to dataflash...
U-Boot> reset
```

Once this has been done, attempt to ping the TFTP server to test the network connection as illustrated below:

```
U-Boot> ping 192.168.2.1
macb0: link up, 100Mbps full-duplex (lpa: 0x45e1)
Using macb0 device
host 192.168.2.1 is alive
```

Loading to RAM

In order to load the image to RAM, you will need to know the physical address of the RAM device on the system. Refer to the documentation for your board if you are unsure of this. The standard addresses for some common EMAC systems are shown in Table 2 below.

Table 2. RAM Physical Addresses

Product	RAM Start Address
SoM-9260M	0x20000000
SoM-9G20M	0x20000000
SoM-9G45M	0x70000000
SoM-9M10M	0x70000000

Before transferring a file to the system, make sure that it does not exceed the size of the available RAM.

The `tftp` U-Boot command is used to transfer files to the system. The command requires two arguments: the address to load the file to and the filename of the image on the TFTP server. The example below demonstrates loading an image named uImage-2.6.30 to the RAM on an SoM-9G45M:

```
U-Boot> tftp 0x70000000 uImage-2.6.30
macb0: link up, 100Mbps full-duplex (lpa: 0x45e1)
Using macb0 device
TFTP from server 192.168.2.1; our IP address is 192.168.2.2
Filename 'uImage-2.6.30'.
Load address: 0x70000000
Loading: #################################################################
         #################################################################
         #################################################################
         #################################################################
         ############################
done
Bytes transferred = 1472456 (1677c8 hex)
U-Boot> printenv filesize
filesize=1677C8
```

Note that the `filesize` variable has been automatically updated to the size of the `uImage-2.6.30` file that was loaded.

Executing from RAM

In some situations, it is advantageous to execute the image directly from RAM after loading with TFTP rather than saving to flash. This is especially helpful when testing new Linux kernel images. Furthermore, the boot command can be set such that the image is automatically downloaded and executed on each boot, making testing more efficient.

After loading a bootable image to RAM, you can execute it directly using the `bootm` command. For example, after loading the kernel image above, running `bootm 0x70000000` would boot the board using the new image without saving the image to flash.

To update the `bootcmd` variable to download the image on each boot, simply replace the command used to load the image from flash with the TFTP download command. The following example illustrates this process on an SoM-9G45M module.

```
U-Boot> printenv bootcmd
bootcmd=cp.b 0xC0042000 0x70000000 0x210000; bootm 0x70000000
U-Boot> setenv bootcmd 'tftp 0x70000000 uImage-2.6.30; bootm 0x70000000'
U-Boot> saveenv
Saving Environment to dataflash...
```

Copying to Flash

The process of copying an image to flash memory differs depending on what type of flash storage device is available on the system. Many boards have more than one flash device, such as serial dataflash and NAND flash. This section explains the process of storing an image to non-volatile storage from RAM.

NOR Flash

A combination of the `erase` and `cp` commands are used to store an image to a NOR flash device. NOR flash is directly memory-mapped to the system at a physical address. Also, each image (U-Boot, bootstrap, kernel, filesystem) must be stored in the correct offset for the system to operate correctly. Refer to the documentation for your hardware for more information on the correct address ranges to use. The `flinfo` command can be used to display the addressing of available flash devices on the system.

After loading an image to RAM, the flash device should be erased and then the image should be copied to the appropriate offset in the flash. The example below illustrates the commands used to program a new kernel image to an SoM-9260M module. Note the use of the `protect off all` command required to unprotect the flash on some systems.

```
U-Boot> protect off all
U-Boot> tftp 0x20000000 ${kernel_name}
U-Boot> erase 0x10100000 0x103fffff
U-Boot> cp.b 0x20000000 0x10100000 ${filesize}
```

Serial DataFlash

Serial DataFlash devices are used on many systems that employ NAND flash as the primary storage device. On these systems, low-level boot code such as the bootstrap, U-Boot, and OS kernel are primarily stored on the DataFlash device. Serial DataFlash is an SPI-based NOR flash device. As with NOR flash, the `flinfo` command can be used to determine the memory addressing layout of the DataFlash device. The `erase` command does not support the DataFlash devices and should not be used before programming an image. The `cp` command is capable of copying an image from RAM to DataFlash.

The example below illustrates the commands used to copy a kernel image to the DataFlash device on an SoM-9G45M module.

```
U-Boot> tftpboot 0x70000000 ${kernel_name}
U-Boot> cp.b 0x70000000 0xC0042000 ${filesize}
```

NAND Flash

Generally, NAND flash is used to store only the root filesystem and auxiliary storage partitions of the OS. Storing an image to NAND flash under U-Boot uses a different set of commands than NOR or DataFlash devices. See `help nand` for more information on the available commands for examining and manipulating NAND flash devices. To gain information on what NAND devices are available on the system, use the command `nand info`. Programming a NAND flash device is much faster than programming most other flash devices.

The example below illustrates the process of loading the root JFFS2 filesystem onto an SoM-9G45M device. Note that the device is erased prior to programming it.

```
U-Boot> nand erase
U-Boot> tftp 0x70000000 ${rootfs_name}
U-Boot> nand write.jffs2 0x70000000 0x0 ${filesize}
```

Scripting

U-Boot also includes a scripting feature that allows an script file with U-Boot commands to be loaded and executed. This can make the task of programming multiple systems much more efficient. The `mkimage` utility is required on a Linux development system in order to create a valid U-Boot image from a script. This section gives a brief example of a U-Boot script for programming a Linux kernel and filesystem.

Making a text file with all of the desired commands is the first step in creating a U-Boot script. The example below is a script used by EMAC to program images to SoM-9260M modules.

```
echo ===== Setting Environment =====
setenv bootdelay 1
setenv rev sl013-aon-30010
setenv bootargs 'console=ttyS3,115200 root=/dev/mtdblock3 rootfstype=jffs2'
setenv kernel_name ${rev}-uImage
setenv rootfs_name sl013-aon-emac-image-SOM9260M.jffs2
saveenv
echo ===== Loading Kernel =====
protect off all
tftpboot 0x20000000 ${kernel_name}
erase 0x10100000 0x103fffff
cp.b 0x20000000 0x10100000 ${filesize}
setenv kernelsize ${filesize}
echo ===== Loading Root Filesystem =====
tftpboot 0x20000000 ${rootfs_name}
erase 0x10400000 0x11ffffff
cp.b 0x20000000 0x10400000 ${filesize}
setenv bootcmd 'protect off all;cp.b 0x10100000 0x20000000 ${kernelsize};bootm 0x20000000'
saveenv
echo ===== DONE! =====
reset
```

Once the file has been created and saved, the `mkimage` utility must be used to create a U-Boot script image. EMAC uses a shell script to aid in this process as listed below:

script_mkimage.sh

```
#!/bin/sh

if [ ! -f "$1" ] || [ $# -lt 2 ]
then
        echo "Usage: $0 SCRIPTNAME OUTPUTNAME"
        exit 1
fi

mkimage -T script -C none -A arm -n 'SoM9260 Load Script' -d $1 $2
```

Note that this script assumes that the `mkimage` command has been installed in the user's PATH. The script should be run with the name of the existing text file as the first argument and the desired output filename as the second argument:

```
./script_mkimage.sh sl013-aon-30010-flash-script sl013-aon-30010-flash-script.img
```

Once an image file has been created, it can be transferred to the board and executed as shown below:

```
U-Boot> tftp 0x20000000 sl013-aon-30010-flash-script.img
U-Boot> autoscr 0x20000000
```

Depending on the version of U-Boot used on the system, U-Boot may print a message recommending use of the `source` command rather than `autoscr`.

U-Boot will execute all commands in the script line-by-line until the script has finished, at which point the U-Boot prompt will be returned to the user. In the example script given above, the script is terminated by a `reset` command, so the script will never exit to the U-Boot prompt.

» esdk » getting_started » install » import » qt » install » getting_started » eclipse » linux_start » uboot_image_loading

---