A Brief Introduction Das U-Boot

A.K.A U-Boot

Presented By: Rick Miles
Melbourne Linux Users Group - 31 Oct. 2016

This presentation will cover:

- What is U-Boot
- Building U-Boot
- Installing U-Boot to an SD Card
- The U-Boot boot process
- Commands and Variables in U-Boot
- Using Boot scripts with U-Boot
- Booting via TFTP with U-Boot

Caveat!

There always is one somewhere

I have 2 Lemaker Banana Pro’s They use an Allwinner A-20 dual core (2 x Cortex-A7) processor. Allwinner A7 series processors use the machine descriptor (mach) sunxi.

As such my presentation may seem “sunxi” oriented but should serve as a basic U-Boot introduction relevant to other ARM CPU’s and Boards.

What is U-Boot?

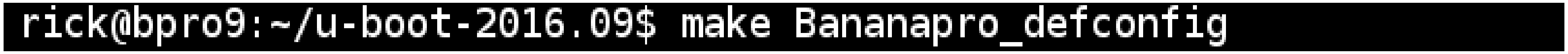
- **Das U-Boot** is an open source, primary boot loader used in embedded devices to package the instructions to boot the device's operating system kernel.
- **U-Boot** provides out-of-the-box support for hundreds of embedded boards and a wide variety of CPUs including PowerPC, ARM, XScale, MIPS, Coldfire, NIOS, Microblaze, and x86.
- The user interface to **U-Boot** consists of a command line interrupter, much like a Linux shell prompt

What is U-Boot? (cont)

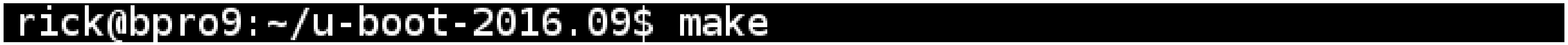
- **U-Boot** uses commands similar to the BASH shell to manipulate environment variables.
- **U-Boot** supports TFTP (Trivial FTP), a stripped down FTP. So that user authentication is not required for downloading images into the board's RAM.

Building U-Boot

U-Boot can be either cross compiled or built natively. The source contains headers and include files for all supported devices. Two commands are required to create the U-Boot binary for a Lemaker Banana Pro.

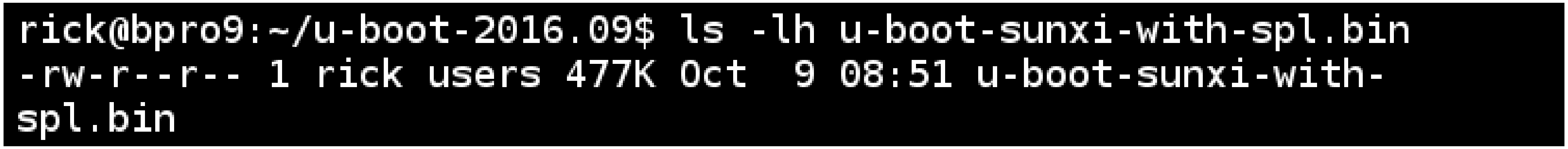


```
rick@bpro9:~/u-boot-2016.09$ make Bananapro_defconfig
```



```
rick@bpro9:~/u-boot-2016.09$ make
```

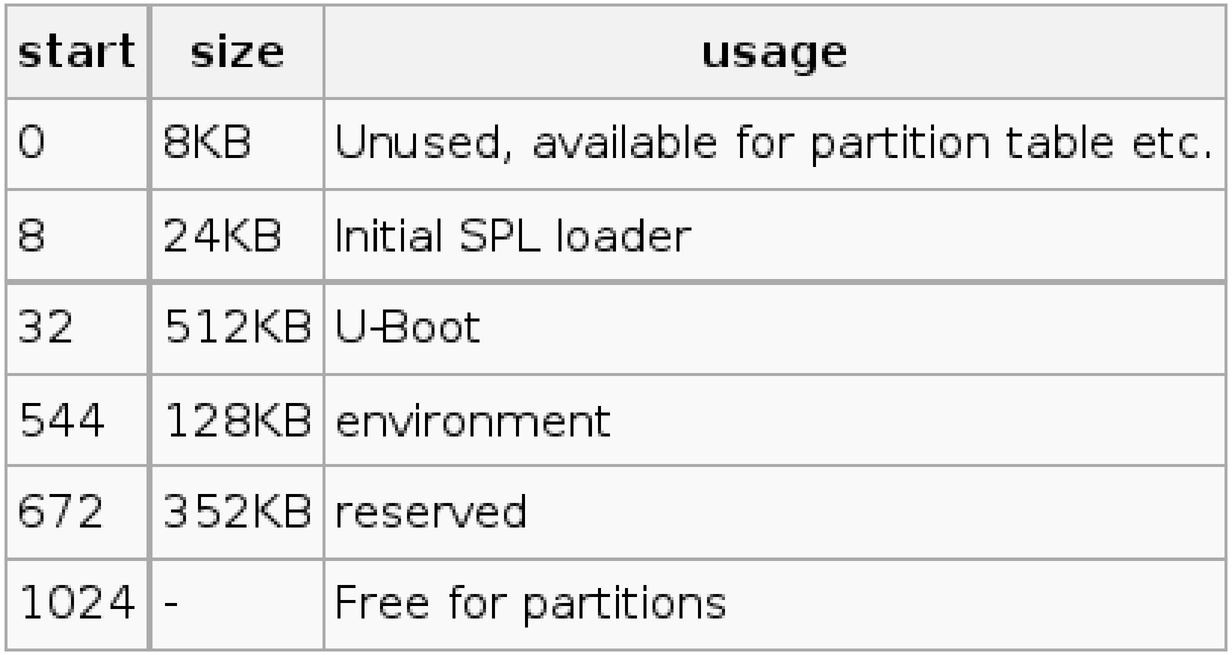
In this instance a file is created comprising U-Boot and SPL.bin (Secondary Program Loader).



```
rick@bpro9:~/u-boot-2016.09$ ls -lh u-boot-sunxi-with-spl.bin
-rw-r--r-- 1 rick users 477K Oct  9 08:51 u-boot-sunxi-with-
spl.bin
```

Installing U-Boot on an SD Card

U-Boot is installed at the beginning of a SD Card and before any partitions. Vendors will hard code processors to find the SPL.bin in a given location

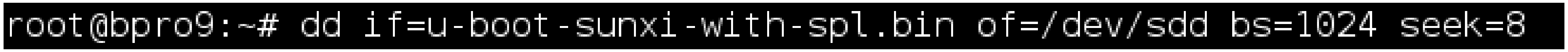


start	size	usage
0	8KB	Unused, available for partition table etc.
8	24KB	Initial SPL loader
32	512KB	U-Boot
544	128KB	environment
672	352KB	reserved
1024	-	Free for partitions

http://linux-sunxi.org/Bootable_SD_card

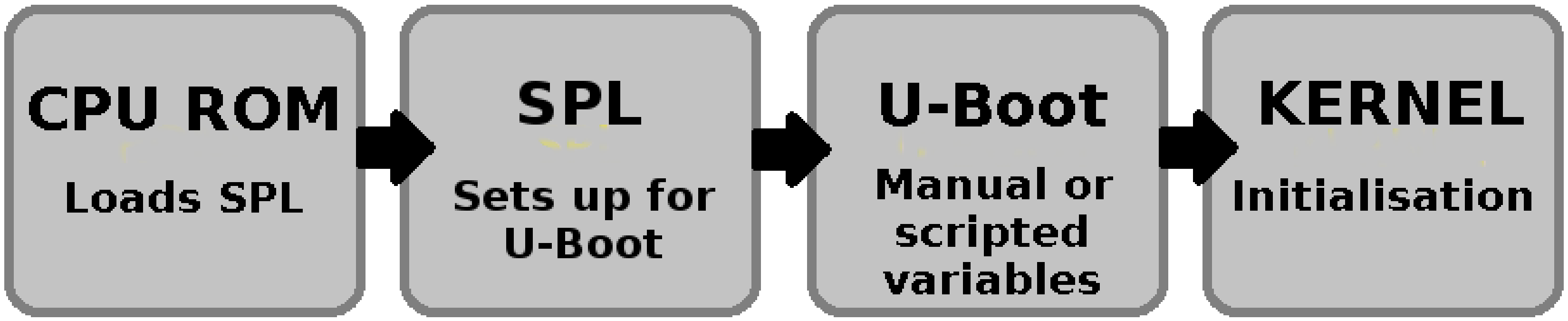
Installing U-Boot on an SD Card

The command ***dd*** is used to transfer the SPL binary and U-Boot to an SD Card beginning at the 8th sector.



```
root@bpro9:~# dd if=u-boot-sunxi-with-spl.bin of=/dev/sdd bs=1024 seek=8
```

The U-Boot Boot Process

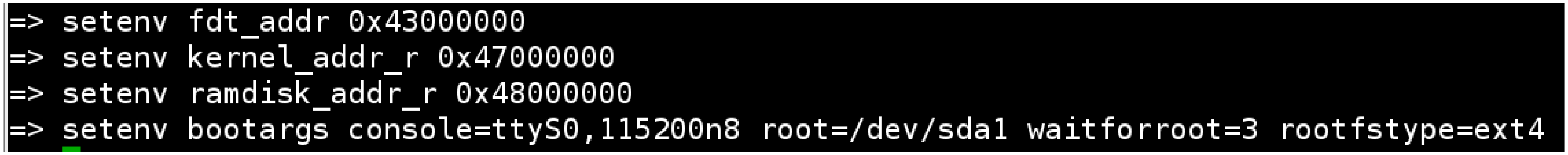


1) ROM does essential initalisations, checks for SPL and then loads it, if it is present, on the SD Card into SRAM (Static RAM).
2) SPL continues initilisation, prepares for and then loads U-Boot into RAM.
3) U-Boot continues setup according to U-boot default environmetal values, variables and commands provided in a boot script and/or variables and commands provided in real time via comman line.
4) Kernel is loaded and system boots into runtime environment.

Commands and Variables

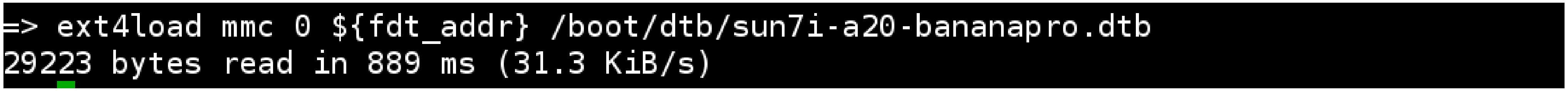
It would not be possible to cover the range of commands available in U-Boot. Instead I will provide some practical examples I use.

Below are four commands setting up variables. The first three provide locations in RAM for the DT (Device Tree) blob, kernel and ramdisk (initrd image). The fourth line supplies variables for a U-Boot reserved variable *bootargs.*



```
=> setenv fdt_addr 0x43000000
=> setenv kernel_addr_r 0x47000000
=> setenv ramdisk_addr_r 0x48000000
=> setenv bootargs console=ttyS0,115200n8 root=/dev/sda1 waitforroot=3 rootfstype=ext4
```

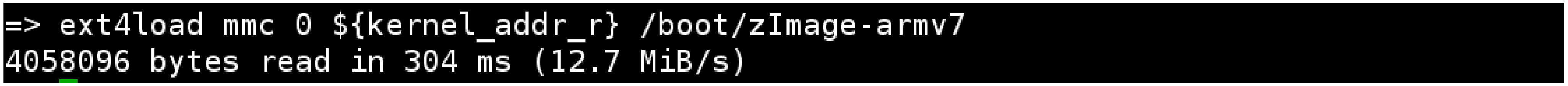
Next is a command to load the variable *fdt_addr from the first partition on the sd card.*



```
=> ext4load mmc 0 ${fdt_addr} /boot/dtb/sun7i-a20-bananapro.dtb
29223 bytes read in 889 ms (31.3 KiB/s)
```

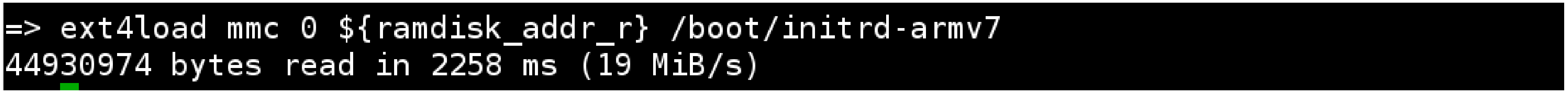
Commands and Variables (cont.)

This command will load the kernel to RAM.



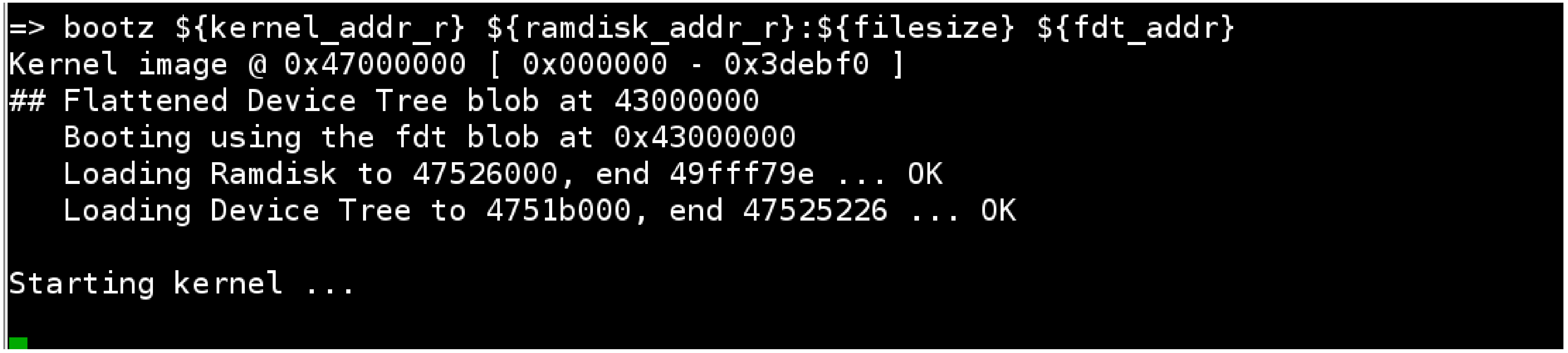
```
=> ext4load mmc 0 ${kernel_addr_r} /boot/zImage-armv7
4058096 bytes read in 304 ms (12.7 MiB/s)
```

The following command will load the initrd image to RAM.



```
=> ext4load mmc 0 ${ramdisk_addr_r} /boot/initrd-armv7
44930974 bytes read in 2258 ms (19 MiB/s)
```

In this final line the U-Boot command *bootz* is used to boot the kernel (zImage) with the ramdisk and fdt being passed to it.

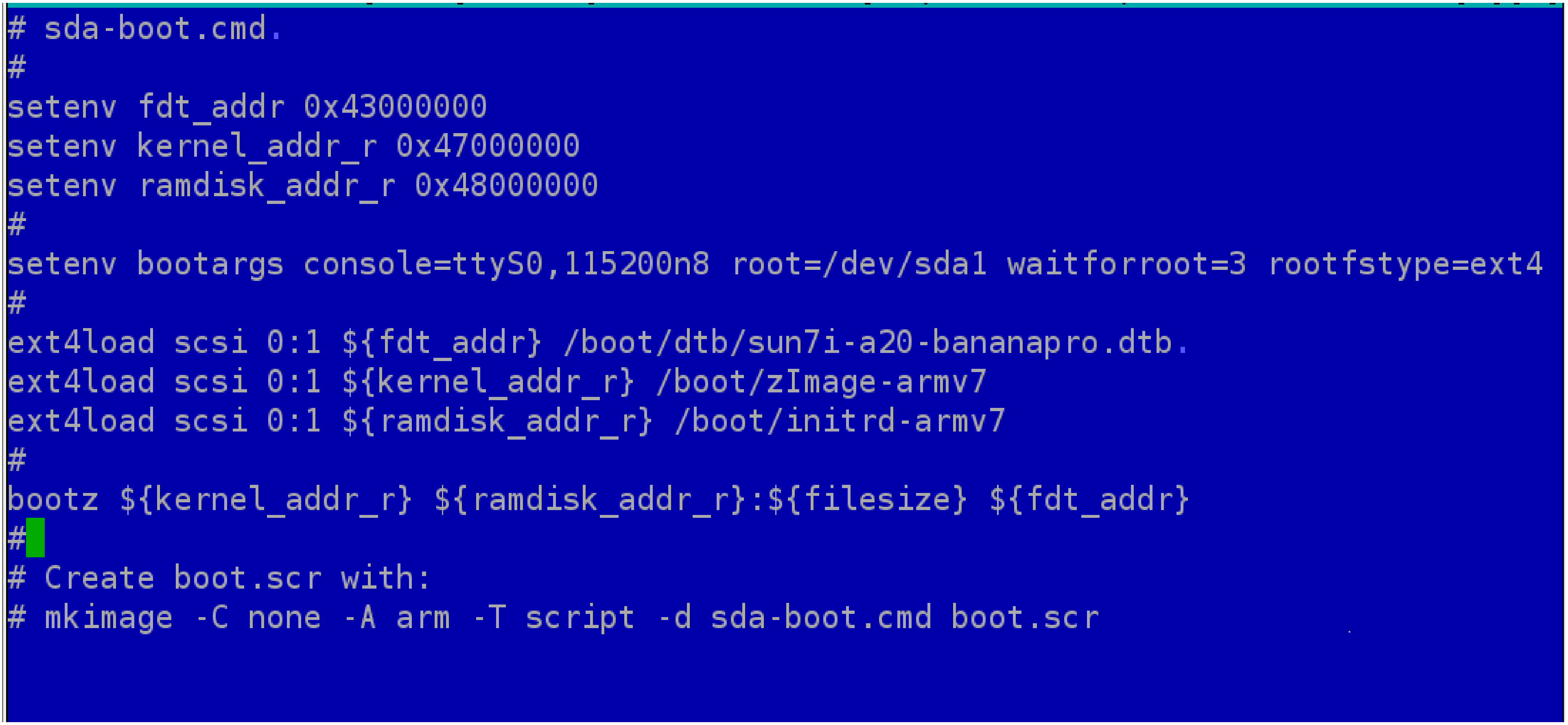


```
=> bootz ${kernel_addr_r} ${ramdisk_addr_r}:${filesize} ${fdt_addr}
Kernel image @ 0x47000000 [ 0x000000 - 0x3debf0 ]
## Flattened Device Tree blob at 43000000
   Booting using the fdt blob at 0x43000000
   Loading Ramdisk to 47526000, end 49fff79e ... OK
   Loading Device Tree to 4751b000, end 47525226 ... OK

Starting kernel ...
```

Using boot scripts with U-Boot

U-Boot commands can be put together in a text file and then the text files used to create a boot.scr. U-boot will look for the script in the root or /boot directory of the first partition on the SD Card, If not found it will look in any SATA disk present and finally in any USB storage device present

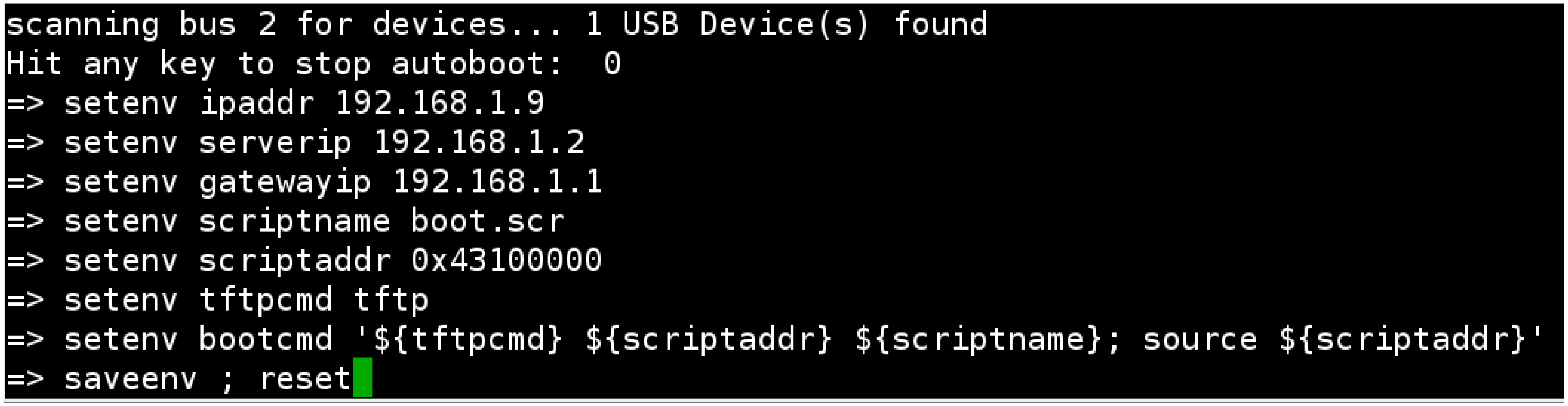


```
# sda-boot.cmd.
#
setenv fdt_addr 0x43000000
setenv kernel_addr_r 0x47000000
setenv ramdisk_addr_r 0x48000000
#
setenv bootargs console=ttyS0,115200n8 root=/dev/sda1 waitforroot=3 rootfstype=ext4
#
ext4load scsi 0:1 ${fdt_addr} /boot/dtb/sun7i-a20-bananapro.dtb.
ext4load scsi 0:1 ${kernel_addr_r} /boot/zImage-armv7
ext4load scsi 0:1 ${ramdisk_addr_r} /boot/initrd-armv7
#
bootz ${kernel_addr_r} ${ramdisk_addr_r}:${filesize} ${fdt_addr}
#
# Create boot.scr with:
# mkimage -C none -A arm -T script -d sda-boot.cmd boot.scr
```

Booting via TFTP with U-Boot

Here is how I set up to boot into a Slackware install on a BananaPro using a kernel, Initrd image and DTB located on another computer set up as a TFTP server.

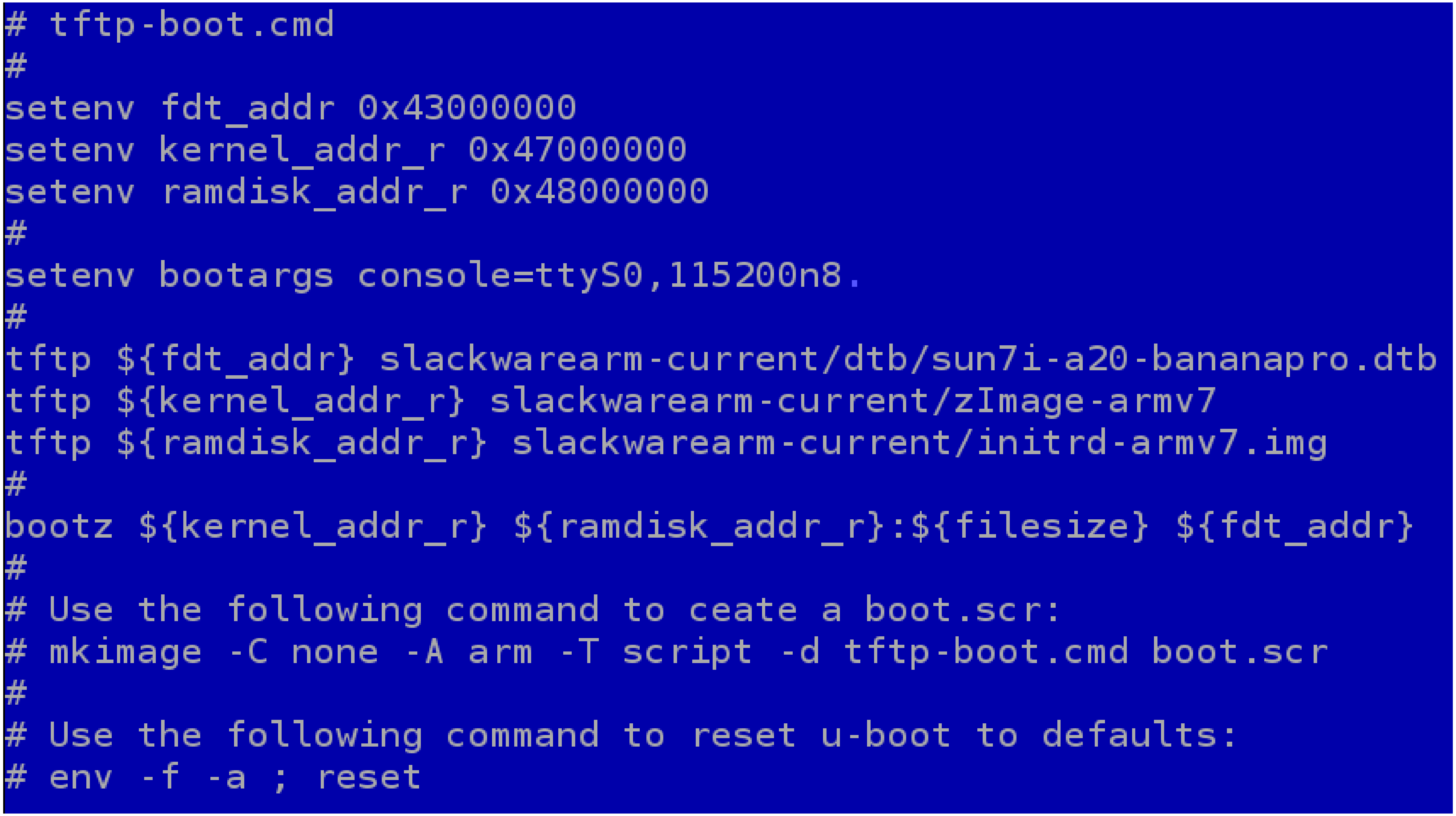
U-Boot environmental settings on the BananaPro’s SD Card are default. After booting into a U-Boot command prompt I use the following commands to set up for booting via TFTP from the server at 192.168.1.2.



```
scanning bus 2 for devices... 1 USB Device(s) found
Hit any key to stop autoboot:  0
=> setenv ipaddr 192.168.1.9
=> setenv serverip 192.168.1.2
=> setenv gatewayip 192.168.1.1
=> setenv scriptname boot.scr
=> setenv scriptaddr 0x43100000
=> setenv tftpcmd tftp
=> setenv bootcmd '${tftpcmd} ${scriptaddr} ${scriptname}; source ${scriptaddr}'
=> saveenv ; reset
```

Booting via TFTP with U-Boot (cont.)

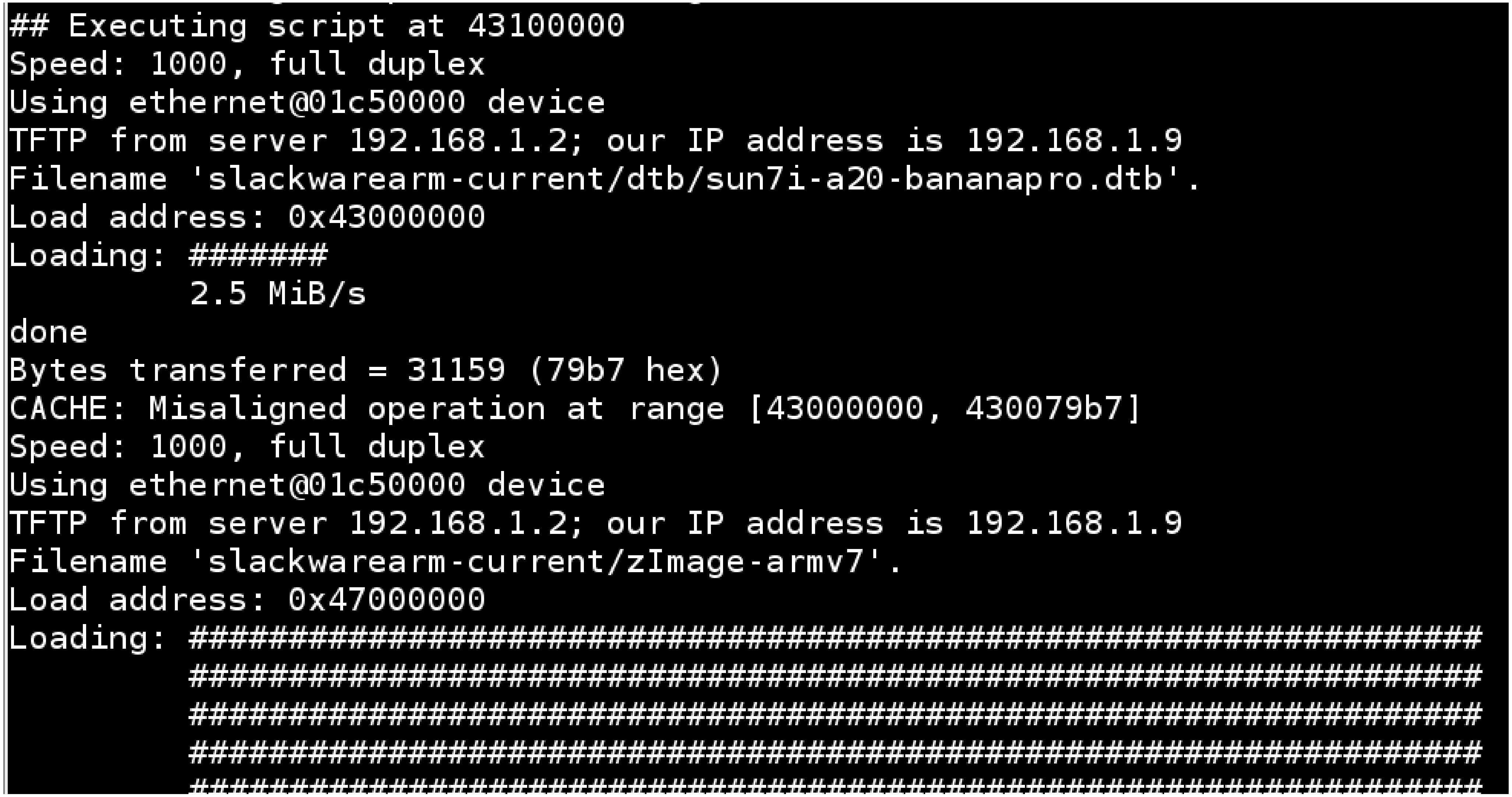
Below the tftp-boot.cmd I use to create a boot.scr that is copied into /tftpboot on the server, 192.168.1.2



```
# tftp-boot.cmd
#
setenv fdt_addr 0x43000000
setenv kernel_addr_r 0x47000000
setenv ramdisk_addr_r 0x48000000
#
setenv bootargs console=ttyS0,115200n8.
#
tftp ${fdt_addr} slackwarearm-current/dtb/sun7i-a20-bananapro.dtb
tftp ${kernel_addr_r} slackwarearm-current/zImage-armv7
tftp ${ramdisk_addr_r} slackwarearm-current/initrd-armv7.img
#
bootz ${kernel_addr_r} ${ramdisk_addr_r}:${filesize} ${fdt_addr}
#
# Use the following command to ceate a boot.scr:
# mkimage -C none -A arm -T script -d tftp-boot.cmd boot.scr
#
# Use the following command to reset u-boot to defaults:
# env -f -a ; reset
```

Booting via TFTP with U-Boot (cont.)

Here’s a bit of screenshot of a TFTP boot in progress. Note that the bananapro dtb download is complete and the kernel download has commenced



```
## Executing script at 43100000
Speed: 1000, full duplex
Using ethernet@01c50000 device
TFTP from server 192.168.1.2; our IP address is 192.168.1.9
Filename 'slackwarearm-current/dtb/sun7i-a20-bananapro.dtb'.
Load address: 0x43000000
Loading: #######
         2.5 MiB/s
done
Bytes transferred = 31159 (79b7 hex)
CACHE: Misaligned operation at range [43000000, 430079b7]
Speed: 1000, full duplex
Using ethernet@01c50000 device
TFTP from server 192.168.1.2; our IP address is 192.168.1.9
Filename 'slackwarearm-current/zImage-armv7'.
Load address: 0x47000000
Loading: #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
```

That’s about it for a lightening introduction. If there’s still some time left I’d like to demonstrate U-boot using a USB To RS232 Serial Adapter between my BananaPro and netbook.

If there’s not enough time may I thank you for your kind attention.

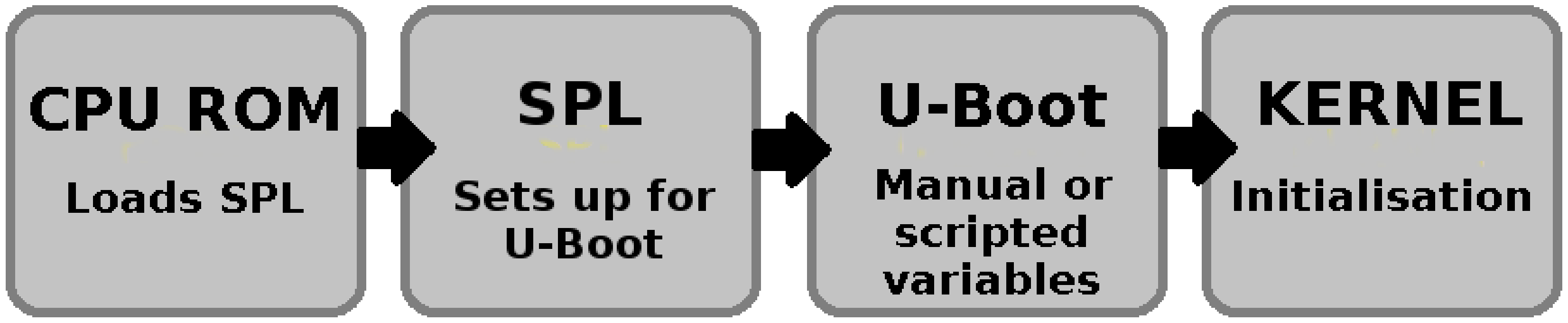
References and further reading:

Keep in mind that U-Boot has a bi-monthly release cycle and documentation found on the web may be out of date. However, I found the following very helpful.

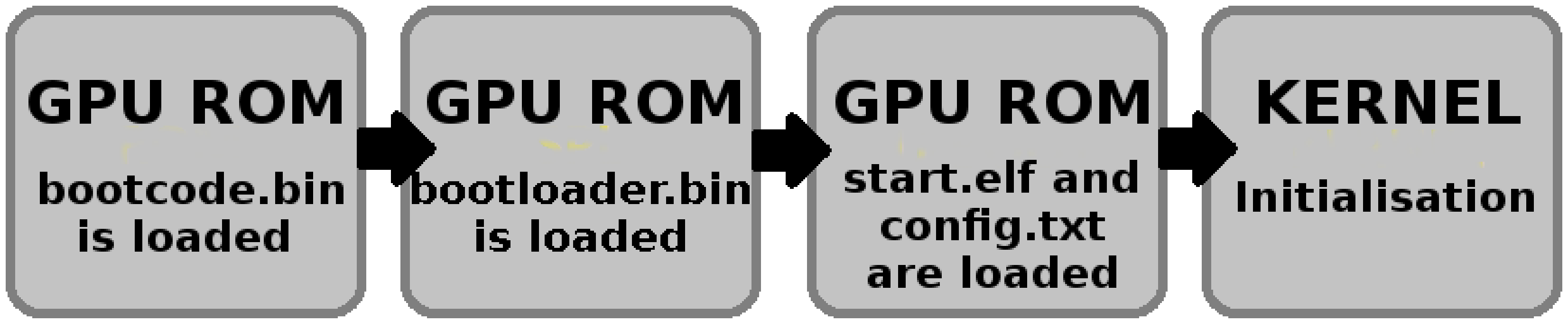
- U-Boot/Documentation: http://www.denx.de/wiki/U-Boot
- https://wiki.debian.org/InstallingDebianOn/Allwinner
- http://processors.wiki.ti.com/index.php/Booting_Linux_kernel_using_U-Boot
- http://linux-sunxi.org
- http://slackware.uk/slackwarearm/slackwarearm-current/INSTALL_BANANAPI.TXT

Addendum: Comparison of Boot Processes

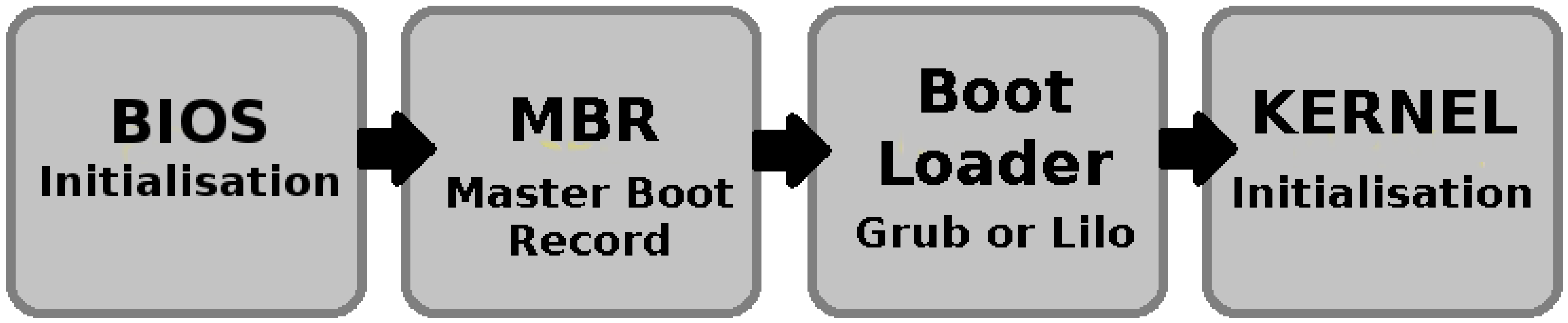
U-Boot Boot Process



RaspberryPi A and B Boot Process



IBM X86 Boot Process


BIOS
Initialisation
MBR
Master Boot Record
Boot Loader
Grub or Lilo
KERNEL
Initialisation