

ARM9 – Linux Kernel

Pradip Selokar*, P. T. Karule**

* Department of Electronics & Communication Engineering, Shri Ramdeobaba College of Engineering & Management, Nagpur, India

** Department of Electronics Engineering, Yashwantrao Chavan College of Engineering, Nagpur, India

Article Info

Article history:

Received Nov 3, 2015

Revised Jan 18, 2016

Accepted Feb 11, 2016

Keyword:

Linux Kernel

Network File System (NFS)

Root File System (RFS)

U-Boot

ABSTRACT

ARM9 supports the Linux Kernel. On a development system it is advantageous to load the Root File System (RFS) through Network File System (NFS). Several pieces of software are involved to boot a linux kernel on SAM9 products. First is the ROM code which is in charge to check if a valid application is present on supported media (FLASH, DATAFLASH, NANDFLASH, and SDCARD). The boot sequence of linux for SAM is done in several steps as given below. Figure 1 gives the linux boot sequence.

1. Boot Program - Check if a valid application is present in FLASH and if it is the case download it into internal SRAM.
2. AT91Bootstrap - In charge of hardware configuration, download U-Boot binary from FLASH to SDRAM, start the bootloader
3. U-Boot - The bootloader, in charge of download kernel binaries from FLASH, network, USB key, etc. Start the kernel.
4. Linux kernel - The operating system kernel.
5. Root File system - Contains applications which are executed on the target, using the OS kernel services.

*Copyright © 2016 Institute of Advanced Engineering and Science.
All rights reserved.*

Corresponding Author:

Pradip Selokar,

Department of Electronics & Communication Engineering,

Shri Ramdeobaba College of Engineering & Management, Katol Road, Gittikhadan, Nagpur – 440013,

India

Email: pradip.selokar@gmail.com

1. INTRODUCTION

ARM9 supports the Linux Kernel. The firmware for ARM9 has been developed in Embedded C language. Therefore the first and foremost task is to load Embedded Linux with GCC compiler on ARM9.

Several pieces of software are involved to boot a linux kernel on SAM9 products. First is the ROM code which is in charge to check if a valid application is present on supported media (FLASH, DATAFLASH, NANDFLASH, and SDCARD). The boot sequence of linux for SAM is done in several steps as given below in Figure 1.

The actual linux boot up procedure can be understood from following steps

1. Processor comes out of reset and branches to the ROM startup code.
2. The ROM startup code initializes the CPU and memory controller, performing only minimal initialization of on-chip devices, such as the console serial port to provide boot diagnostic messages. It also sets up the memory map for the kernel to use in a format that is consistent across platforms, and then jumps to the boot loader.
3. The boot loader decompresses the kernel into RAM, and jumps to it.
4. The kernel sets up the caches, initializes each of the hardware devices via the init function in each driver, mounts the root file system and execs the init process, which is the ultimate parent of all user mode processes, typically /sbin/initd.

Executing the first program linked against the shared C runtime library (often `init`) causes the shared runtime library to be loaded. In a typical Linux system, `init` reads `/etc/inittab` to execute the appropriate run control script from `/etc/rc.d`, which executes the start scripts to initialize networking and other system services.

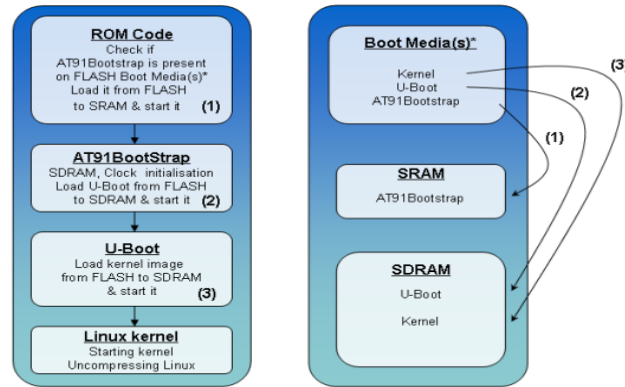


Figure 1. Boot Sequence of Linux for SAM (Smart ARM Microcontroller)

2. AT91BOOTSTRAP

AT91Bootstrap is a first step bootloader providing a set of algorithms to manage hardware initialization (GPIO, Clock, SDRAM, etc), to download your main application from specified FLASH media to main memory and to start it. AT91Bootstrap can be easily configured using a higher level protocol.

AT91Bootstrap integrates several sets of algorithms:

- Device initialization such as clock speed configuration, PIO settings, etc.
- Peripheral drivers such as PIO, PMC, SDRAMC, etc.
- Physical media algorithms such as Data Flash, NAND Flash, Parallel Flash, etc.
- File System drivers such as JFFS2, FAT, etc.
- Compression and Cipher algorithms

The bootloader performs the processor initialization (PLLs, PIOs, SDRAMC, SPI), loads UBoot from Data Flash sectors to SDRAM and then jumps to it.

2.1. Load AT91Bootstrap on SAM9 board

This section describes How to load AT91Bootstrap into the boot media with SAM-BA (Smart ARM based Microcontrollers-Boot Assistant). Following are the steps.

- Jumper (J7) must be opened to boot from on chip Boot ROM (Boot Mode Select BMS = 1) as shown in Figure 2.

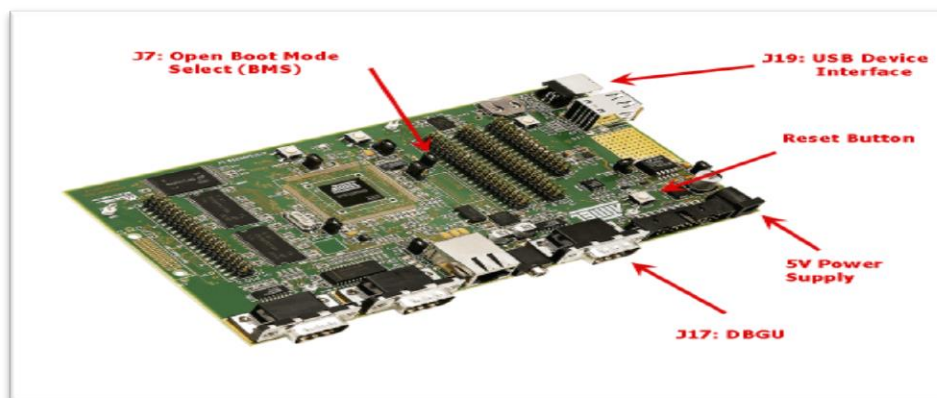


Figure 2. AT91SAM9260 Evaluation Kit

- Power up the board
- Verify that the USB connection is established (ATMEL AT91SAM9260-EK Test Board appears in taskbar notification area)
- Start SAM-BA GUI Application
- Select the board in the drop-down menu and choose the USB Connection (Figure 3)



Figure 3. SAM-BA Board Selection Dialogue Box

- Eventually plug back a jumper to access the media on which u-boot must be loaded to
- In the main SAM-BA window (Figure 4)
 - Choose the proper *media tab* (Data Flash, Nand Flash, etc.) in the SAM-BA GUI interface
 - Initialize the media choosing the *Enable* action in the *Scripts* rolling menu and press *Execute*
 - Choose *Send boot file*, press *Execute*
 - Select the at91bootstrap binary file and press *Open* ; the media is written down
 - Close SAM-BA, remove the USB cable

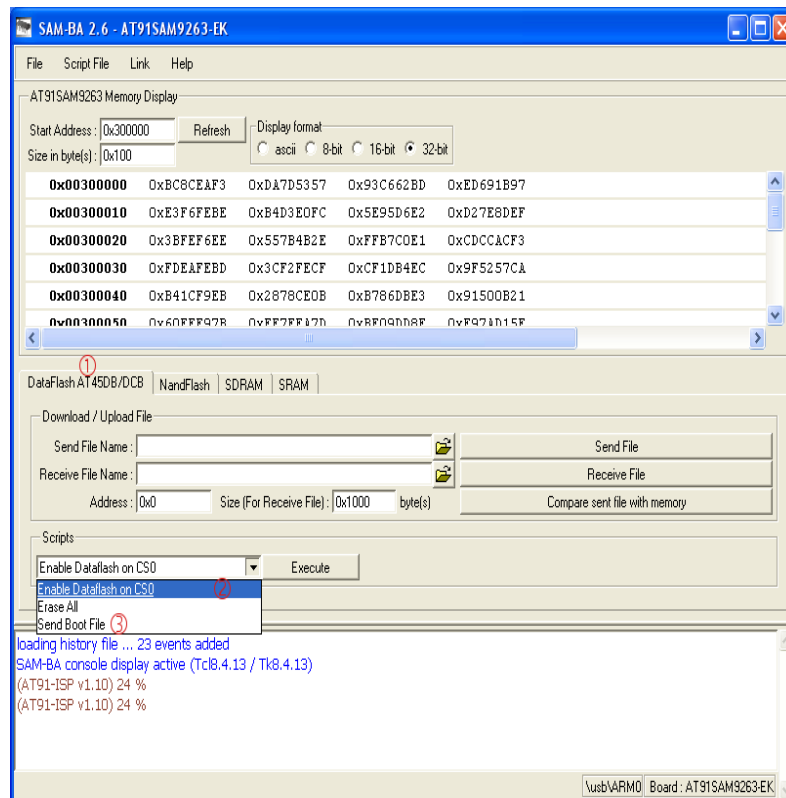


Figure 4. Main SAM-BA Window

2.2. Boot Strategies

AT91 chips embed a boot ROM code. It is enabled depending on BMS (Boot Mode Select) pin state on reset. This ROM code scans the contents of different media like SPI DATAFLASH, NAND FLASH or SDCARD to determine if a valid application is available then it download the application into SAM9 internal SRAM and run it. To determine if a valid application is present the ROM code checks the eight ARM exception vectors.

If no application is available then SAM-BA application is executed. It waits for transactions either on the USB device, or on the DBGU serial port, Then the SAM-BA tool can be used to program FLASH or EEPROM present on your board.

3. U – BOOT

U-boot takes place in the Linux demo as a second stage bootloader. It is responsible of configuring main interfaces and launching a Linux system. The u-boot environment is a little read/write persistent space that stores variables needed by the bootloader to configure itself properly and to adapt to its environment (network configuration, boot arguments, storage location, etc.). It is located in the same media that it has booted from. Loading U-Boot on AT91SAM boards is similar to that of loading AT91Bootstrap.

3.1. Use U – Boot

Using a terminal software (MINICOM on Linux and Hyper Terminal on Windows) on a host system, one can connect to u-boot through the DBGU serial interface. Serial communication parameters are 115200 8-N-1 i.e.

Table 1. Serial Communication Parameters

| | |
|--------------|--------|
| Baud Rate | 115200 |
| Data | 8 bits |
| Parity | None |
| Stop | 1 bit |
| Flow Control | None |

The “minicom” window appears on a Linux terminal (konsole) by applying the command minicom. To set the serial communication parameters, the command minicom –s needs to be applied. After applying this command the window as shown in Figure 5 appears. In this window you should go to “serial port setup”.

Once entered in serial port setup, the window as shown in Figure 6 appears. Set serial communication parameters as shown in Figure 6. In this window capital letter alphabets are there adjacent to each parameter. To set a particular parameter, press a corresponding alphabet key from the keyboard i.e. ‘A’ for Serial Device, ‘E’ for Bps/Par/Bits, ‘F’ for Hardware Flow Control etc. Hardware Flow Control is set to NO because there are no handshaking type of signals.

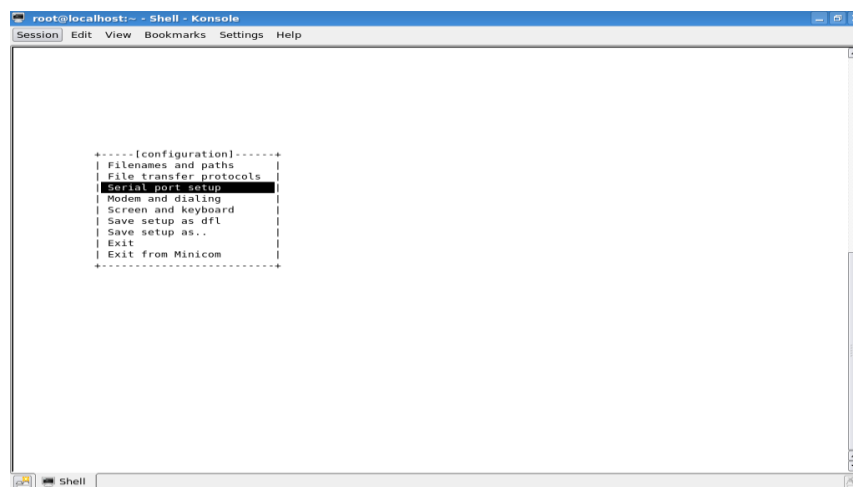


Figure 5. Minicom window

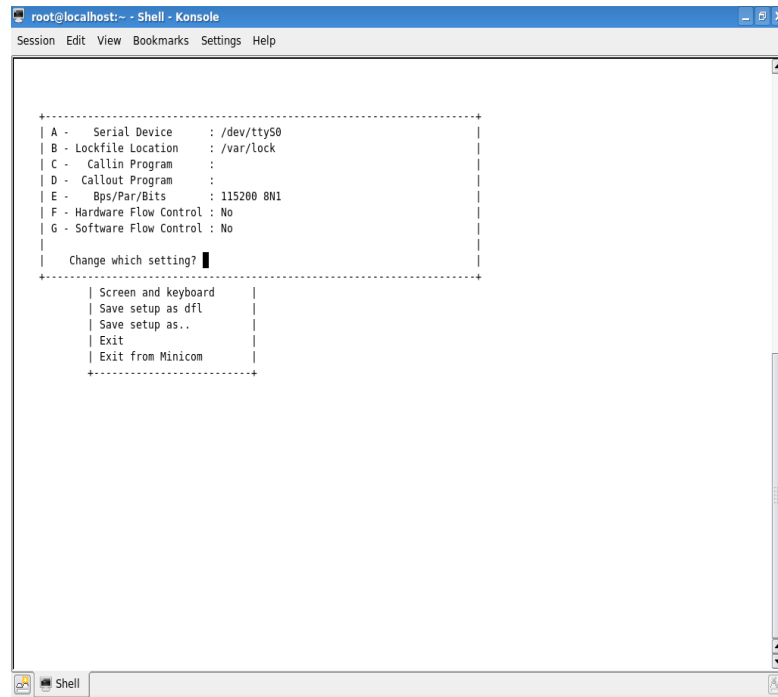


Figure 6. Serial Port Setup in Minicom

3.2. 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

Following are some useful U-Boot commands

- **boot** - boot default, i.e., run 'bootcmd'
- **bootm** - boot application image from memory
- **erase** - erase FLASH memory
- **ftpbboot** - boot image via network using TFTP (Trivial File Transfer Protocol) protocol
- **erase** - erase FLASH memory
- **flinfo** - print FLASH memory information

3.3. 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 0x21400000 linux-2.6.25.img
setenv ramdisk_addr 0x21100000
setenv ramdisk tftp 0x21100000 sam9-ramdisk.gz'
setenv go run linux
run ramdisk
bootm 0x21400000
saveenv
```

4. LOADING LINUX WITH U-BOOT ON AT91 BOARDS

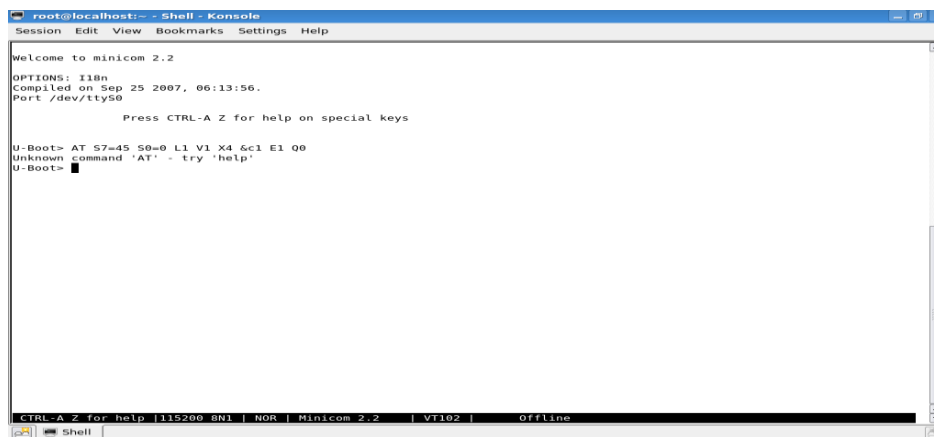
U-Boot does not support normal linux kernel images like zImage or Image (arch/arm/boot/), you have to create an uImage file with the mkimage tool which encapsulates kernel image with header information, CRC32 checksum, etc. mkimage comes in source code with U-Boot distribution and it is built during U-Boot compilation (u-boot-source-dir/tools/mkimage).

4.1. Loading Linux through Network (RFS through NFS)

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. On the u-boot side, you will have to setup the networking parameters as given in the following five steps:

1. Setup an Ethernet address (MAC address): `setenv ethaddr 2e:48:27:cf:7d:ec`
2. Setup IP parameters
 - the board IP address: `setenv ipaddr 172.16.180.220`
 - the server IP address where the TFTP server is running: `setenv serverip 172.16.180.229`
3. Saving Environment to flash: `saveenv`
4. Download the Linux `ulmage` and the root file system to a RAM location using the u-boot tftp command (U-Boot script file can be used for this purpose).
5. Launch Linux issuing a bootm or boot command (Figure 9).

Once the U-boot is set on the AT91SAM9260 board it is ready for loading linux kernel through network. The Root File System (RFS) is located on the host machine and using Network File System it is to be loaded on the board's RAM i.e. it can be said as RFS through NFS. To launch the linux on a AT91SAM9260 board ready with U-boot, connect the serial (RS-232) cable between host machine and the board (COM0/ttyS0 port of host machine and RS-232 port of the board). When connected and board is powered up, in the minicom shell the window as shown in Figure 7 appears. Now press the RESET button (Refer Figure 2) on the board, it will show memory details of the board (RAM size, NAND Flash, Data Flash, Data Flash page size, number of pages, logical memory addresses etc.) as shown in Figure 8.



```

root@localhost:~# Shell - Konsole
Session Edit View Bookmarks Settings Help

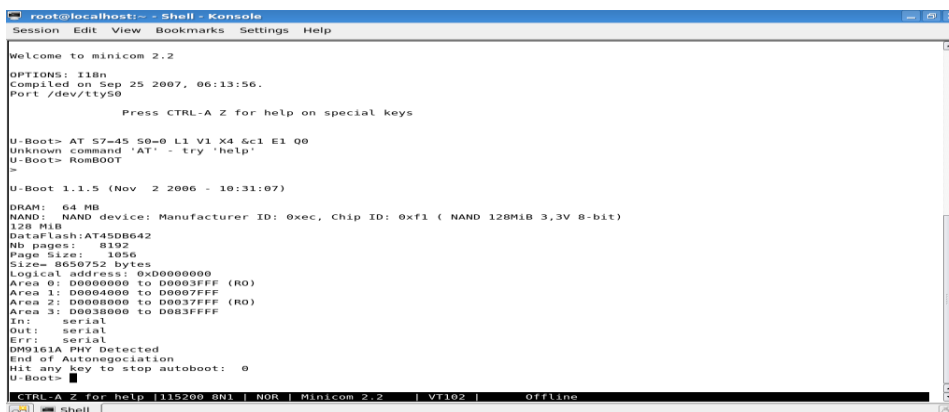
Welcome to minicom 2.2
OPTIONS: I18n
Compiled on Sep 25 2007, 06:13:56.
Port /dev/ttyS0

Press CTRL-A Z for help on special keys

U-Boot> AT S7=45 S0=0 L1 V1 X4 &c1 E1 00
Unknown command 'AT' - try 'help'
U-Boot>
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.2 | VT102 | Offline
Shell

```

Figure 7. U-Boot Window



```

root@localhost:~# Shell - Konsole
Session Edit View Bookmarks Settings Help

Welcome to minicom 2.2
OPTIONS: I18n
Compiled on Sep 25 2007, 06:13:56.
Port /dev/ttyS0

Press CTRL-A Z for help on special keys

U-Boot> AT S7=45 S0=0 L1 V1 X4 &c1 E1 00
Unknown command 'AT' - try 'help'
U-Boot> RomBOOT
=
U-Boot 1.1.5 (Nov 2 2006 - 10:31:07)

DRAM: 64 MB
NAND: NAND device: Manufacturer ID: 0xec, Chip ID: 0xf1 ( NAND 128MiB 3,3V 8-bit)
128 KiB
DataFlash: AT45DB642
Nb pages: 8192
Page Size: 1056
Size= 8656752 bytes
Logical address: 0x00000000
Area 0: D0000000 to D0003FFF (RO)
Area 1: D0004000 to D0007FFF (RO)
Area 2: D0008000 to D0037FFF (RO)
Area 3: D0038000 to D003FFFF
In: serial
Out: serial
Err: serial
DM9161A PHY Detected
End of Autonegotiation
Hit any key to stop autoboot: 0
U-Boot>
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.2 | VT102 | Offline
Shell

```

Figure 8. Memory Details of the AT91SAM9260 Board

5. LINUX KERNEL

5.1. Get and Patch the Linux Kernel

1. <http://www.kernel.org> is the primary site for Linux Kernel source.
2. Identify on which linux kernel version the experimental patches will apply & download the corresponding Linux kernel.
<http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.25.tar.bz2>
`tar xyz linux-2.6.25.tar.bz2`
`cd linux-2.6.25`
3. Download the AT91 Maintainer's patch and apply it.
 We get patch at <http://maxim.org.za/AT91RM9200/2.6/2.6.25-at91.patch.gz>
 The command to apply this patch is
`zcat 2.6.25-at91.patch.gz | patch -p1`
4. Take the experimental patch set and apply it on top of the AT91 one
<ftp://www.linux4sam.org/pub/linux/2.6.25-at91-exp.patch.gz>
 command to apply patch is
`zcat linux-2.6.25-at91-exp.diff.gz | patch -p1`

5.2. Configure and Build the Linux Kernel

Now configure the Linux Kernel according to your hardware. First identify your kernel revision, your board and then, download the corresponding configuration file. Default configuration files are provided on Internet at http://www.linux4sam.org/twiki/pub/Linux4SAM/LinuxKernel/at91sam9260_defconfig. Apply following commands

```
cd linux-2.6.25
cp at91sam9260_defconfig .config
make ARCH=arm oldconfig
```

At this step you can modify the default configuration. Apply the following command

```
make ARCH=arm menuconfig
```

And then finally build the Linux Kernel image using command

```
make ARCH=arm CROSS_COMPILE=<path_to_cross-compiler/cross-compiler-prefix->
```

Finally when ready with the linux kernel patched properly, one can launch the linux to the AT91SAM9260 board. To launch the linux with the command 'boot', it is necessary to initially reset the board, and arrive at a stage as shown in Figure 8.

At this stage type the command 'boot' and press ENTER, as shown in Figure 9.

```
root@localhost:~# Shell - Konsole
Session Edit View Bookmarks Settings Help
Nb pages: 8192
Page Size: 1056
Size= 0050752 bytes
Logical address: 0x00000000
Area 0: 00000000 to 0003FFFF (RO)
Area 1: 00040000 to 0007FFFF
Area 2: 00080000 to 0003FFFF (RO)
Area 3: 00038000 to 0003FFFF
In: serial
Out: serial
Err: serial
DM9161A PHY Detected
End of Autonegotiation
Hit any key to step autoboot: 0
U-Boot> boot
TFTP from server 172.29.51.112; our IP address is 172.29.48.156
Filename 'linux'.
Load address: 0x21400000
Loading: #####
done
Bytes transferred = 1367072 (14dc20 hex)
# Booting image at 21400000 ...
Image Name: linux-2.6
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 1367008 Bytes = 1.3 MB
Load Address: 20008000
Entry Point: 20008000
Verifying Checksum ... OK
OK
Starting kernel ...
Uncompressing Linux.....
CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.2 | VT102 | Offline
Shell
```

Figure 9. Launching Linux to the board

As soon as the linux is launched to the board from host machine, it will show the host machine's IP address as 'TFTP from server 172.16.180.229, AT91SAM9260 board's IP address as 'our IP address is 172.16.180.220, it also shows boot address as 'Load address: 0x21400000' (Refer to Figure 9). These addresses are as per I have set earlier in section 3.4.1 (Loading Linux through Network). Also from Figure 9, it shows the Filename as 'linux' which is actually a 'ulmage' file of Linux Kernel, created using the steps mentioned in section 3.5 (LINUX KERNEL). I have renamed the file 'ulmage' as 'linux' for simplicity.

6. CONCLUSION

In this paper I have explained how to load Linux operating system on ARM9 through network, i.e. RFS through NFS.

Once the development of the firmware is over the Linux kernel and the root file system can be burnt completely to the NAND Flash for permanent usage.

As a future scope, ARM9 can communicate with the ARM7 using CAN bus. ARM7 further communicates serially with the general purpose microcontrollers. Thus serial data can be brought up over an Ethernet using ARM9.

REFERENCES

- [1] Jinxue Zhang, Ming Zhang, "Research and Design of Embedded Tank Car Monitoring System Based on ARM9", IEEE Computer Society, 2009 Second International Symposium on Computational Intelligence and Design.
- [2] Shaoke Chen and Shaojun Jiang, "Design of Embedded Network Interface Controller Based on ARM9 and ARMLinux", Applied Computing, Computer Science, and Advanced Communication First International Conference on Future Computer and Communication, FCC 2009, Wuhan, China, June 6-7, 2009. Proceedings – Springerlink.
- [3] Mike Meyerstein, Inhyok Cha and Yogendra Shah, "Security Aspects of Smart Cards vs. Embedded Security in Machine-to-Machine (M2M) Advanced Mobile Network Applications", Security and Privacy in Mobile Information and Communication Systems. First International ICST Conference, MobiSec 2009, Turin, Italy, June 3-5, 2009, Revised Selected Papers – Springerlink.
- [4] Ou Qingyu, Huang Kai and Wu Xiaoping, "Research on the Embedded Security Architecture Based on the Control Flow Security", IEEE Computer Society : 2009 Second International Workshop on Computer Science and Engineering.
- [5] Chandrasekaran, S. Rajendran, J. Annamalai, "Data Driven Security Alarm Model for Embedded Applications", Computing, Communication and Networking, 2008. ICCCN 2008. International Conference – IEEE xplore digital library.
- [6] Guy Gogniat, Tilman Wolf and others, "Reconfigurable hardware for high-security/high-performance embedded systems: the SAFES perspective", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Volume 16, Issue 2 (February 2008).
- [7] Ted Huffmire, Brett Botherton and others, "Managing Security in FPGA based embedded Systems", November-December 2008 (vol. 25 no. 6), *IEEE CS digital Library*.
- [8] R. Vaslin, G. Gogniat, J.-P. Diguët, R. Tessier, and W. Burlinson. A security approach for off-chip memory in embedded microprocessor systems. *Elsevier Journal of Microelectronics and Microprocessors*, 2008.
- [9] T. Eisenbarth, T. Guneyesu, C. Paar, A.-R. Sadeghi, D. Schellekens, and M. Wolf. *Reconfigurable trusted computing in hardware*. In Proceedings of the ACM Workshop on Scalable Trusted Computing, November 2007.
- [10] R. Elbaz, L. Torres, G. Sassatelli, P. Guillemain, M. Bardouillet, and A. Martinez. *A parallelized way to provide data encryption and integrity checking on a processor-memory bus*. In Proceedings of the IEEE/ ACM International Design Automation Conference, July 2006.
- [11] T. Burlinson, W. University of South Brittany, "Reconfigurable Security Support for Embedded Systems ", HICSS '06. Proceedings of the 39th Annual Hawaii International Conference on System Sciences, 2006 – IEEE xplore digital library.
- [12] C. Yan, B. Rogers, D. Engländer, Y. Solihin, and M. Prvulovic. *Improving cost, performance, and security of memory encryption and authentication*. In Proceedings of the International Symposium on Computer Architecture, June 2006.
- [13] Z Baruch and others, 2006 *IEEE International Conference on Automation, Quality and Testing, Robotics, Embedded System for network flow identification*.

BIOGRAPHY OF AUTHOR



Pradip Selokar, Department of Electronics & Communication Engineering, Shri Ramdeobaba College of Engineering & Management, Nagpur – 440013, Maharashtra, India.