ARM-Based Product Application Group

# AT91RM9200DK U-Boot Developper Manual

**Reference: ABP-STD-BOOT-004**

**Version: V1.1**

**Release Date: 1-Sep-2003**

## Revision History

| Revision | Date | Author | Comments |
|---|---|---|---|
| V1.0 | 25-Jun-2003 | NLe | Creation. |
| V1.1 | 1-Sep-2003 | NLe | Add upgrade Thunderboot to U-Boot paragraph.<br>Modifiy Commands Directory names |

# Table of Contents

| Date | 1-Sep-2003 | Name | AT91RM9200DK U-Boot Developper Manual |
|---|---|---|---|
| Version | V1.1 | Reference | ABP-STD-BOOT-004 |

## 1 - Scope

This document aims at describing the bootloader embedded in the AT91RM9200DK development kit. It gives an U-Boot description on a developper point of view. It presents:

- U-Boot organization.
- U-Boot compilation.
- the different solutions to upgrade U-Boot or to flash U-Boot from scratch.
- U-Boot associated tools.

This document is a complement to the U-Boot User Manual and is destined to programmers who have to make some modifications on U-Boot.

## 2 - Features

The main features of the U-Boot software for the AT91RM9200 series ARM Based Products are:

- Standalone primary bootstrap
- Small footprint
- OS-independent
- Auto-boot and interactive modes
- Command line interface
- Non-volatile environment variables
- Flash programming capability
- DataFlash programming capability (only available in latest Open Source download)
- Download through serial interface (Kermit protocol)
- Download through Ethernet (tftp)
- Integrated bootp
- Scripting capability

## 3 - Software Package Description

U-Boot is a project supported by the Open Source community. It is distributed under the GPL license. See the source code for credits and other licensing information.

The sources proposed are:

- An U-Boot version containing all source codes.

Note: This version is not the latest version available on the SourceForge CVS. If important changes have to be made, please recover the latest source code on the SourceForge website at URL : http://sourceforge.net/projects/u-boot/

- A Boot version containing primary bootstrap and de-compression source code.
- A Loader version containing all source code allowing to restart U-Boot from scratch.

The AT91RM9200DK U-Boot software release is made of three different parts:

- A Source directory containing U-Boot, Boot and Loader sources.
- A Binary Files directory containing all binary files to flash an U-Boot version without any compilation phase.
- A Documentation directory containing a User manual and a Developper Manual.

# 4 - Getting Started with U-Boot Solution

## 4.1 - U-Boot Organization

To optimize resource use and efficiency, U-Boot is made up of three parts:

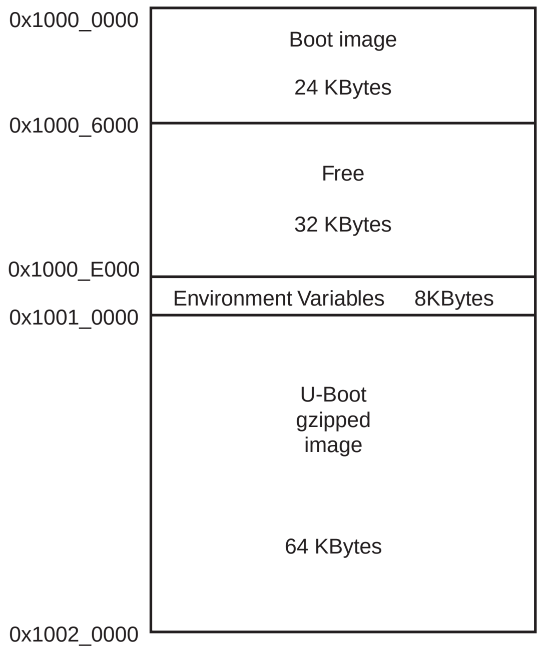
- A primary bootstrap.
- A de-compression executable.
- A gzip-compressed binary executable which is called the U-Boot Image.

The primary bootstrap is concatenated to the de-compression executable into one single file called the boot image and must reside in the AT91RM9200DK Flash on NCS0 at address 0x1000_0000.

The primary bootstrap is a simple assembly code routine that sets up the system and puts it into an operational state. More specifically, it starts the high-speed clocks of the CPU and programs the on-chip memory controller to define the memory layout. When these steps are completed, the primary bootstrap gives control to the de-compression executable.

The de-compression executable is an optimized version of the gunzip program. It decompresses the gzip-compressed image of U-Boot into RAM, and jumps to it. The location of the binary executable in RAM is defined at link time.

**Figure 1.** Flash Mapping



## 4.2 - Building ATMEL U-Boot solution

### 4.2.1 - Compiling U-Boot

Building U-Boot is normally not necessary since it is already loaded into Flash memory in the target boards in the factory and is operational as is.

However, specific requirements may need U-Boot to be modified and recompiled. This section describes how to proceed on a PC running Linux® RedHat7.2 or later. It is assumed that the arm-linux-gcc cross compiler is installed.

- The main executable is built with:

```
> cd AT91RM9200-U-Boot
> make at91rm9200dk_config
> make all
```

The above steps produce the U-Boot code in several formats, including elf and pure binary formats.

The U-Boot gzipped flashable image named u-boot.gz is built with:

```
> gzip -c u-boot.bin > u-boot.gz
```

### 4.2.2 - Compiling Boot image

The Boot contains primary bootstrap and de-compression source code necessary to start U-Boot.

Building Boot image is normally not necessary since it is already loaded into Flash memory in the target boards in the factory and is operational as is.

However, specific requirements may need Boot image to be modified and recompiled. This section describes how to proceed on a PC running Linux® RedHat7.2 or later. It is assumed that the arm-linux-gcc cross compiler is installed.

- Build the primary bootstrap:

```
> cd AT91RM9200-Boot
> make
```

The above steps produce the Boot code named boot.bin in pure binary format.

### 4.2.3 - Compiling Loader image

The Loader allows the user to launch a new U-boot application when U-Boot has been erased from flash.

Building Loader image is normally not necessary since a binary file has already been compiled and is in the Binary files directory and is operational as is.

However, specific requirements may need Loader image to be modified and recompiled. This section describes how to proceed on a PC running Linux® RedHat7.2 or later. It is assumed that the arm-linux-gcc cross compiler is installed.

- Build the loader image:

```
> cd AT91RM9200-Loader
> make
```

The above steps produce the Loader code named loader.bin in pure binary format.

## 4.3 - Flash U-Boot solution on your board

### 4.3.1 - Upgrading Thunderboot to U-Boot

This paragraph aims to describe how to upgrade the "Thunderboot / Zooboot" boot program with the latest boot program version "U-boot". See the user documentation for more information on U-Boot features.

*Hardware & Software requirements:*

- A serial cable (null modem)
- An ethernet cable (Not mandatory)
- Hyperterminal running on your PC application (1115200,8,N,1)

The Kermit protocol with a hyperterminal session will be used to download the specific_file in the SDRAM, then it will be copied in the Flash.

*How to proceed:*

The first file to load is the boot image “**boot.bin**”.

```
AT91RM9200-DK> loadb 20000000
## Ready for binary (Kermit) download …
## Start Addr =0x20000000
```

At this step the boot image is loaded in SDRAM. The next command copies the SDRAM (0x20000000) to flash(0x10000000).

```
AT91RM9200-DK>protect off 10000000 10005fff
AT91RM9200-DK> erase 10000000 10005fff
Erase Flash from 0x10000000 to 0x10005fff...
Erasing sector 0 ... ok.
Erasing sector 1 ... ok.
Erasing sector 2 ... ok.
done.
Erased 3 sectors.
AT91RM9200-DK>cp.b 20000000 10000000 5FFF
Copy to flash… done.
AT91RM9200-DK>protect on 10000000 10005FFF
Protected 3 sectors
AT91RM9200-DK>
```

Once Primary Bootstrap is loaded, U-Boot gzipped image must be copied into Flash.

The principle is exactly the same as for the primary bootstrap. The file to load is the uboot gzipped image “**u-boot.gz**”.

```
AT91RM9200-DK> loadb 20000000
## Ready for binary (Kermit) download …
## Start Addr =0x20000000
AT91RM9200-DK> protect off 10010000 1001ffff
AT91RM9200-DK> erase 10010000 1001ffff
Erase Flash from 0x10010000 to 0x1001ffff...
```

```
Erasing sector 8 ... ok.
Erasing sector 9 ... ok.
done.
Erased 2 sectors.
AT91RM9200-DK>cp.b 20000000 10010000 FFFF
Copy to flash… done.
AT91RM9200-DK>protect on 10010000 1001FFFF
Protected 2 sectors
AT91RM9200-DK>
```

Reboot your board, you have upgraded your board with U-Boot.

### 4.3.2 - U-Boot Upgrade

This paragraph deals with people who want to replace their U-Boot version embedded in flash.

Once the gzipped U-Boot image is available, it can be transferred to the host with loadb or tftp and copied into the Flash to replace the old version via the cp command. This process is illustrated below:

```
Uboot> tftp 20000000 u-boot.gz
or
Uboot> loadb 20000000 u-boot.gz
...
Uboot> protect off 10010000 1001ffff
Uboot> erase 10010000 1001ffff
Uboot> cp.b 20000000 10010000 ffff
Uboot> protect on 10010000 1001ffff
```

Note: It is very important to protect the AT91RM9200DK development board against power outages during Flash programming.

### 4.3.3 - Flashing U-Boot from scratch

This paragraph deals with people who crashed their previous U-Boot flashed version.

It is mandatory to remove temporarily the R159 resistor on the AT91RM9200DK development kit to set the on-chip boot mode (BMS low during the reset) to boot on the embedded Boot ROM Program.

*Hardware and software requirements*

- A serial cable (Null modem)
- An Ethernet cable (not mandatory)

*Loader*

- Connect the serial port of your PC to the “SERIAL DEBUG PORT” on the AT91RM9200DK development board.
- Launch an hyperterminal application on your PC (115200, 8, N, 1)

The Boot Program waits for any transaction and downloads a piece of code into the internal SRAM via Xmodem protocol for the DBGU. After the end of the download, it branches to the application entry point at the first address of the SRAM.

- On hyperterminal, send the file “loader.bin” with the Xmodem protocol.

At the end of Xmodem, the “Loader” application starts a new xmodem protocol.

- On hyperterminal, send the file “u-boot.bin” with the Xmodem protocol.

*Commands to flash*

At the end of Xmodem, hyperterminal displays a prompt (Uboot>).

Enter the following commands:

```
Uboot> protect off all
Un-protect Flash Bank # 1
Uboot> erase all
Erasing sector 1   … ok
...
Erasing sector 39 … ok
Done
```

At this step the flash is completely erased. The next step is to load the boot image in SDRAM and to copy it in flash. This is done through the serial port (Kermit protocol on hyperterminal). The first file to load is the boot image “boot.bin”.

```
Uboot> loadb 20000000
## Ready for binary (Kermit) download …
## Start Addr          =0x20000000
```

At this step the boot image is loaded in SDRAM. The next command copies the SDRAM (20000000) to flash(10000000).

```
Uboot>cp.b 20000000 10000000 5FFF
Copy to flash… done.
Uboot>protect on 10000000 10005FFF
Protected 3 sectors
Uboot>
```

Once Primary Bootstrap is loaded, U-Boot gzipped image must be copied into Flash. The principle is exactly the same as for the primary bootstrap. The file to load is the u-boot gzipped image “u-boot.gz”.

```
Uboot> loadb 20000000
## Ready for binary (Kermit) download …
## Start Addr          =0x20000000
Uboot>cp.b 20000000 10010000 FFFF
Copy to flash… done.
Uboot>protect on 10010000 1001FFFF
Protected 2 sectors
Uboot>
```

U-boot is flashed. Set the R159 resistor on the AT91RM9200DK development kit to boot on flash.

# 5 - U-Boot Associated Tools : mkimage

## 5.1 - mkimage

mkimage is a command line tool that is supplied with the U-Boot distribution. It is provided in source code under the GPL license. It was designed to be compiled by the gcc GNU compiler and to run in an x86 Linux operating environment.

The synopsis of mkimage is:

```
> mkimage [<options>] <image file>
```

mkimage is a utility that builds binary images compatible with U-Boot. Basically, mkimage takes a data file and adds a data description header. The data file can be a CPU-executable binary, or raw data. The header contains several fields, the more important being a magic number describing the type of data, and a CRC.

To build the header correctly, mkimage needs information about the type of data it will append to the header. This information is given through command line options. The supported options are described below:

**-d <data file>**

This option is used to specify the file that contains the data put in the image file after the header. The -d option must always be used except when the -l option is used.

**-T <data type>**

Sets the type field. This type tells which kind of data the image contains. The supported types are:

- standalone: the data contains an OS-independent executable.
- kernel: the data contains the kernel of an OS.
- ramdisk: the data contains a ramdisk for Linux OS.

**-a <load address>**

Sets the load address. This field is applicable when the image file is an executable that will be run by U-Boot with a bootm command. The load address tells bootm into which location to copy the executable code before it jumps to it. If the load address is not explicitly given to mkimage, U-Boot will use the contents of the loadaddr environment variable instead.

The load address must be expressed in hexadecimal.

**-C <compression>**

mkimage has the ability to compress the data section. U-Boot can decompress the data at load time or run time. The supported keywords for the -C option are:

- none: no compression is used.
- gzip: the popular GNU gzip algorithm is used to compress the data.

**-e <entry point>**

Sets the entry point for an executable. This field is applicable only for images which contain an executable code. This field tells U-Boot that it must jump to the entry point address to execute the image correctly.

**-l**

This option must be used alone. It reads the specified image and lists header information.

Other options for mkimage exist, but their operation is irrelevant in the context of their operation with U-Boot, or they are not fully tested.

## 5.2 - Bootm command example

That paragrah gives an example of how to use the bootm command.

```
($U-BOOT-PATH)/tools > ./mkimage -A arm -O linux -T kernel -C gzip -a
0x20008000 -e 0x20008000 -d linux.bin.gz uImage /tftpboot/
```

This command builds a compressed image of the linux kernel and places it in the tftp-boot directory. This image is downloadable in SDRAM and is executable by bootm.

```
($U-BOOT-PATH)/tools > ./mkimage -A arm -O linux -T ramdisk -C gzip -a
0x21100000 -e 0x21100000 -d ramdisk.bin ramdisk /tftpboot/
```

This command builds a compressed image of a linux ramdisk and places it in the tftp-boot directory. This image is downloadable in SDRAM.

**Example**

```
Uboot> tftp 20008000 uImage
Uboot> tftp 21100000 ramdisk
Uboot> setenv bootargs root=/dev/ram rw initrd=0x21100000,
60000000 ramdisk_size=15360 console=ttyS0,115200 mem=32M
Uboot> saveenv
Uboot> bootm 20008000 21100000
```

The second bootm argument indicates the ramdisk address. bootm will de-compress the kernel and the ramdisk with the gzip algorithm into the memory located at address 0x20008000 for the kernel and at address 0x21100000 for the ramdisk. bootm will then jump to 0x20008000 defined as the entry point. Linux arguments are passed to the kernel via the bootargs variable.