ARM-Based Product Application Group

AT91RM9200DK U-Boot User Manual

Reference: ABP-STD-BOOT-003

Version: V1.0

Release Date: 10-Jul-2003

Revision History

Revision	Date	Author	Comments
V1.0	24-Jun-2003	NLe	Creation.

Date	10-Jul-2003	Name	AT91RM9200DK U-Boot User Manual
Version	V1.0	Reference	ABP-STD-BOOT-003

Table of Contents

Date	10-Jul-2003	Name	AT91RM9200DK U-Boot User Manual
Version	V1.0	Reference	ABP-STD-BOOT-003

1 - Scope

This document aims at describing the bootloader embedded in the AT91RM9200DK development kit. It gives a general overview of U-Boot and a detailed description of the different commands to use.

It is mainly destined to those who want to know how using 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 - Overview

3.1 - Description

The U-Boot program is a standalone software application that initializes the AT91RM9200 processor and then jumps to the main application. If the user sends a character to the system console during the bootstrap, U-Boot enters interactive mode. The user is prompted to enter commands on the command line. The commands allow the processor to read/modify memory areas, set environment variables, download data from the serial port or from the Ethernet interface and execute code routines.

The U-Boot program is completely standalone. In particular, it is independent of the operating system and not linked to the main application software. However, it is able to launch the latter by executing an absolute branch operation.

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.

3.2 - AT91RM9200 Development Kit Setup

To start the U-Boot software:

- Connect a PC running a terminal emulator such as Kermit or HyperTerminal to the Serial Debug port of the AT91RM9200DK development board. The default port parameters are 115200 bps (can be changed later), 8 bits, one stop, no parity, no flow control. The PC acts as the system console.
- Power-up the board.

The U-Boot starts and supplies a prompt on the terminal emulator.

It is recommended that a network connection is in place when U-Boot is used. In this case, the network should be connected to the Ethernet interface of the AT91RM9200DK development board.

3.3 - Autoboot Mode and Interactive Mode

When the AT91RM9200-based system is powered up, a banner is displayed on the system console. The user is then prompted to send a character to the system console to enter the interactive mode. If, after a delay controlled by the bootdelay environment variable, no character is received, U-Boot continues in autoboot mode by attempting to execute the command defined by the bootcmd environment variable. Typically, this variable contains a “go” command that performs a jump and gives control to the main application software.

Note that if the bootdelay variable is not defined or does not contain a relevant value, U-Boot does not attempt the autoboot and will remain in interactive mode. The same fall-back in interactive mode occurs if the bootcmd variable is undefined.

3.4 - Environment Variables

U-Boot has the ability to define and maintain environment variables in non-volatile memory. An environment variable may contain any character string. Some variables have a reserved name which directly impacts the behavior of the system. An example of such a reserved variable is ipaddr. This variable contains the IP address of the local host in the quad-dotted notation. For complete information on the reserved environment variables, refer to Section 6.1.

Other variables are user-defined. Their name can be almost any alphanumeric string and they may contain any character string for the user's convenience.

The list of currently defined environment variables can be displayed on the console with the printenv command. The example below shows a typical display following the printenv command:

```
Uboot> printenv
baudrate=115200
ipaddr=192.168.1.1
ethaddr=12:34:56:78:9A:BC
serverip=192.168.1.5
Environment size: 80/8188 bytes
```

A new variable can be defined by the setenv command. The example below defines a new variable:

```
Uboot> setenv myboard AT91RM9200DK
Uboot> printenv
baudrate=115200
ipaddr=192.168.1.1
ethaddr=12:34:56:78:9A:BC
serverip=192.168.1.5
myboard=AT91RM9200DK
Environment size: 102/8188 bytes
```

Variables are created and modified in regular system memory (RAM). This means that a power outage would alter the existence and value of a new variable. A Flash sector is dedicated to store all the environment variables and their values in a non-volatile area. The saveenv command transfers all the currently defined variables and their values into the Flash memory.

The memory space used to store environment variables is 8 kilobytes. This space is used to store the variable names and values. The total length of variable names and values should not exceed 8 kilobytes.

4 - Data Download

U-Boot offers two options to download data into the local memory: via the serial port of the console using the Kermit protocol or via a tftp network service.

4.1 - Download via the serial port

The loadb command enables download of binary data via the serial port interface used by the console.

loadb uses the Kermit protocol to transfer data with a good compromise between reliability and bandwidth usage. Kermit is a protocol supported by most terminal emulation tools (including C-Kermit, Minicom and Microsoft's HyperTerminal).

- The example below shows how to transfer data from the PC connected on the console port to the AT91RM9200DK memory at address 0x20000000:

```
Uboot> loadb 20000000
## Ready for binary (Kermit) download ...
```

- Instruct the terminal emulation tools to send the required file using the Kermit protocol.
- For information on how to send files using the Kermit protocol, refer to the documentation of the terminal emulation tool.

Note: For C-Kermit users: "set prefixing all" under the C-Kermit prompt must be set to ensure proper transmission of all packets.

- When transmission is complete, the U-Boot command line prompt re-appears.

4.2 - Download via the Ethernet Interface

Network downloading is one of the most powerful capabilities of U-Boot. It provides the system with the capability to download binary data from a tftp server connected to the network via the Ethernet interface of the AT91RM9200 processor. In particular, the downloaded data can be the main application software itself, thus economizing a considerable amount of Flash memory.

Before downloading any data from the network, some reserved environment variables must be set so as to configure the user system. The configuration can be done either manually, or automatically by co-operating with a remote bootp service. The operations described below demonstrate how to configure the network manually. For information about the automated bootp procedure, refer to Section 5.5.

- Define the Ethernet hardware address of the AT91RM9200 device. This is done by setting the reserved environment variable ethaddr to a text string reflecting this address. The hardware address is given by using common 6-byte notation, as illustrated below:

```
Uboot> setenv ethaddr 12:34:56:78:9A:BC
```

Note: The hardware address given above is only for purposes of illustration. It may belong to another organization. The user should ensure that the hardware addresses used are his own property.

- Define an IP address for the Ethernet interface. This is done by setting the reserved environment variable ipaddr. The value of ipaddr should be a text string reflecting the IP address in quad-dotted notation. This is illustrated below:

  ```
  Uboot> setenv ipaddr 192.168.1.1
  ```

- Specify the IP address of the tftp server by setting the reserved environment variable serverip as illustrated below:

  ```
  Uboot> setenv serverip 192.168.1.254
  ```

- The system is now able to connect the tftp service (assuming it is available) on the server 192.168.1.254 to download data. This can be done with the following command:

  ```
  Uboot> tftp 20000000 application.bin
  ```

The command above assumes that there is a tftp service active on host 192.168.1.254, and that there is a file named application.bin located in the tftp root directory of this host.

The command tells U-Boot to download the file with the tftp protocol and to store it at address 0x20000000.

```
Uboot> flinfo
Bank # 1: Atmel: AT49BV1614 (16Mbit)
  Size: 2 MB in 40 Sectors
  Sector Start Addresses:
10000000 (RO)  10002000 (RO)  10004000      10006000      10008000
1000A000       1000C000       1000E000 (RO) 10010000 (RO) 10018000 (RO)
10020000       10030000       10040000      10050000      10060000
10070000       10080000       10090000      100A0000      100B0000
100C0000       100D0000       100E0000      100F0000      10100000
10110000       10120000       10130000      10140000      10150000
10160000       10170000       10180000      10190000      101A0000
101B0000       101C0000       101D0000      101E0000      101F0000
```

When data needs to be written into a Flash area, the sectors concerned must be erased before the write can be performed. This is done with the erase command.

For complete information on Flash-specific operation, refer to Section 6.2.

5.3 - DataFlash Features

There is no specific command to access a DataFlash. Indeed, it is accessed exactly as a flash device that is to say as a memory with a physical address.

When starting, U-Boot checks if DataFlash devices are connected on SPI Chip Select 0 and 3. If it is the case, a virtual address is attributed to the corresponding DataFlash:

- 0xC0000000 for DataFlash connected on SPI Chip Select 0
- 0xD0000000 for DataFlash connected on SPI Chip Select 3

Here is an example of U-Boot starting with two DataFlash devices connected.

```
U-Boot 0.3.0 (Jun 11 2003 - 14:33:35)

U-Boot code: 21F00000 -> 21F13C0C  BSS: -> 21F1EF44
DRAM Configuration:
Bank #0: 20000000 32 MB
Atmel: AT49BV1614 (16Mbit)
Flash:  2 MB
DataFlash:AT45DB642
Nb pages:   8192
Page Size:   1056
Size= 8650752 bytes
Logical address: 0xC0000000
DataFlash:AT45DB161
Nb pages:   4096
Page Size:    528
Size= 2162688 bytes
Logical address: 0xD0000000
Uboot>
```

Once virtual address is attributed, the DataFlash is accessed by using the generic memory operations (cp, md...) with its corresponding virtual address.

It is possible to have informations about DataFlash by using the flinfo command:

```
Uboot> flinfo
DataFlash:AT45DB642
Nb pages:   8192
Page Size:   1056
Size= 8650752 bytes
Logical address: 0xC0000000
DataFlash:AT45DB161
Nb pages:   4096
Page Size:    528
Size= 2162688 bytes
Logical address: 0xD0000000

Bank # 1: Atmel: AT49BV1614 (16Mbit)
  Size: 2 MB in 40 Sectors
  Sector Start Addresses:
    10000000 (RO) 10002000 (RO) 10004000 (RO) 10006000      10008000
    1000A000      1000C000      1000E000 (RO) 10010000 (RO) 10018000 (RO)
    10020000      10030000      10040000      10050000      10060000
    10070000      10080000      10090000      100A0000      100B0000
    100C0000      100D0000      100E0000      100F0000      10100000
    10110000      10120000      10130000      10140000      10150000
    10160000      10170000      10180000      10190000      101A0000
    101B0000      101C0000      101D0000      101E0000      101F0000
Uboot>
```

5.4 - Advanced Features

5.4.1 - Scripting

In the context of U-Boot, scripting refers to the capability of executing several consecutive commands as if they were entered in sequence on the command line. The script program is stored in an environment variable.

Scripting is supported by assigning an environment variable to a string which contains a list of commands separated by semi-colons. The contents of the variable can be subsequently executed consecutively with the run command. The example below illustrates scripting. It implements a routine which downloads a new application from the tftp server and writes it to Flash. The first line assigns the variable flashit to a list of commands. The saveenv command is then executed to save the flashit variable into the non-volatile environment area. The flashit command is then executed and the Flash is updated. Note that this script assumes that the Flash memory is unprotected against write.

```
Uboot> setenv flashit tftp 20000000 mycode.bin\; erase 10020000 1002FFFF\;
cp.b 20000000 10020000 8000
Uboot> saveenv
Uboot> run flashit
```

Note that the reserved variable bootcmd is a special case of scripting. If this variable is defined, its contents are automatically executed in autoboot mode.

5.4.2 - Code Execution

U-Boot has the ability to execute binary code located in memory. Two commands are provided to execute the binary: go and bootm.

Go command

go executes a simple branch to the specified address. If required, command line parameters can be passed to the launched application. The usage of go is simple, but no checks are made as to the suitability of the target binary. Additionally, a binary started by go is always executed in place.

Bootm command

bootm expects the binary code to be specially formatted. Such a format is produced by mkimage with the -T kernel or -T standalone option. Before the actual execution, bootm checks the suitability of the code by checking a magic number and a checksum. Then the executable section is copied into RAM at the address pointed to by loadaddr and a jump is performed there.

Note that bootm supports the execution of compressed code produced by mkimage with the -C gzip option. In this case, the compressed code is de-compressed while it is copied to RAM.

Please refer to the U-Boot Developper Manual for more informations on mkimage.

5.5 - Automatic Host Configuration

U-Boot supports automatic configuration via the bootp protocol. With this protocol, U-Boot can be assigned several parameters, such as its IP address or its host name, dynamically. Its application code can also be downloaded from a server and does not have to reside locally in a non-volatile memory. Another advantage is the ease of maintenance of the application software. When a new firmware release is available, it can be copied to a server, identified by the bootp protocol, and the update is effective as soon as the host is rebooted.

The only prerequisite for starting the bootp operation is the assignment of an Ethernet hardware address to the AT91RM9200DK development kit. When the bootp starts, the host sends a network broadcast requesting the response of a bootp service. If a bootp service is available and recognizes the hardware address of the requesting host, the server is able to provide the client host with its configuration parameters.

Starting the bootp operation with U-Boot is easy. Once the Ethernet hardware address is assigned, the operation consists of typing the bootp command. All the configurations regarding which parameters can be assigned to which values by bootp are made on the server side. Refer to the bootp server documentation for more information.

6 - U-Boot Reference

6.1 - Reserved Environment Variables

baudrate

This variable is used to select the baud rate of the system console through which U-Boot interacts with the user.

baudrate is always defined. If not specified, it defaults to 115200. The supported values are 9600, 19200, 38400, 57600, and 115200.

bootargs

This variable is application-specific. It is used as an exchange area to pass information to the main application which is started by U-Boot. It can be filled with any data.

It is the responsibility of the main application to implement the routines which will access bootargs and interpret its contents.

bootcmd

If defined, this variable is executed automatically when entering the autoboot mode. In an operational context, it should contain the list of commands which launch the main application.

If bootcmd is left undefined, U-Boot never enters the autoboot mode and remains in interactive mode.

As any other script, bootcmd can be executed with the run command for test purposes.

bootfile

This variable is set by the bootp service to the name of the boot file to be loaded from the tftp service.

ethaddr

This variable is used to specify the MAC address of the Ethernet interface of the AT91RM9200 device. The address must be in the format of six hex bytes separated by a column.

When the variable is first defined, the Ethernet MAC address is immediately programmed into the Ethernet interface. If the ethaddr is set in the non-volatile environment, the Ethernet MAC address is programmed when U-Boot starts.

filesize

This variable is set by the bootp service to the size (in hex) of the boot file to be loaded from the tftp service.

hostname

This variable can be set to a character string which reflects the name of the local host. It can be either set manually or by the bootp service.

ipaddr

The ipaddr variable defines the IP address assigned to the first network interface. This address is mandatory for U-Boot to exchange data through the network such as for tftp downloads.

ipaddr can be either specified manually, or automatically assigned by the bootp protocol.

The format of the address is the popular quad-dotted notation composed of four decimal bytes separated by dots.

loadaddr

Several commands can be used to download data into the host's memory. Such commands include loadb and tftp. If the destination address is not specified explicitly in the command line, the downloaded data will be stored at the address defined by loadaddr.

When undefined, a default of 0x21000000 is assumed.

serverip

This variable specifies the IP address of the tftp server from which U-Boot downloads data. It can be either defined manually or automatically assigned by the bootp protocol.

The tftp server IP address must be specified in quad-dotted notation composed of four decimal bytes separated by dots.

6.2 - Commands

?

Alias for the help command.

bootd

Boot default. Short for 'run bootcmd'.

bootm [<address> [<parameters list>]]

Bootm is used to execute an application located in the host’s memory. This application must be in the specific image format generated by the mkimage tool with the -T kernel or -T standalone option. For more information about mkimage, please refer to U-Boot Developer Manual.

The optional <address> parameter of the bootm command is used to specify the location of the application in the host's memory. This value can be overridden by the <load address> parameter given to mkimage when the executable application was built. If none of these addresses is given, the value of the reserved variable loadaddr is used as a default.

If extra parameters are given after <address>, these are passed to the application as its argc and argv global variables.

If the application was designed so that it returns, the user is prompted with a new command line when the application is finished.

The main difference between the go and bootm commands is that go expects a pure binary code while bootcmd expects an mkimage-formatted executable.

bootp [<address>]

This command establishes the host's bootp client connection. The only pre-requisite needed to initiate the bootp connection is the hardware address of the local host's Ethernet interface. If the connection is successful, the bootp client can get configuration information from the bootp server, and stores this information in environment variables.

The bootp command of U-Boot has the ability to set the following environment variables:

- bootfile
- filesize
- hostname

- ipaddr
- netmask
- serverip

For more information about these variables, refer to the corresponding entry in Section 6.1

If the automatic configuration defines a tftp server to download a boot file, the tftp connection is established and the file download is attempted. If the optional <address> parameter is given, the boot file is stored at this location. If this address remains unspecified, the value of loadaddr is used as a default.

Note that the boot file cannot be downloaded into Flash memory. To program Flash memory, the file must be first downloaded into regular RAM, and then copied into Flash with the cp command.

cp[.<size>] <source> <destination> <length>

This command is used to copy data between two memory areas. The optional size parameter specifies the type of data to be moved at each iteration. Supported values are l for long 32-bit words, w for 16-bit half-words, and b for bytes. If the size is not specified, the last used size is used, l being the first default.

The <length> parameter specifies the total number of bytes to be copied.

Misaligned copies are not allowed.

Note that cp is the only command which permits Flash memory to be written. For such an operation to be successful, make sure that the Flash area to be programmed has been erased (all bits at 1). See the erase command below.

cp command is also the only command which permits DataFlash memory to be written.

cmp [.<size>] <area1> <area2> <length>

cmp compares the values of two memory areas. It returns a message giving the result. In case of mismatch, a summary is given.

echo [<string> [...]]

echo is mainly used in scripts. It simply sends the argument strings to the console.

erase <start> <stop>

This command erases an area of Flash memory. This operation is mandatory before any write to the are can be performed.

<start> and <stop> are the first and last addresses of the area to be erased. They must match Flash sector boundaries. Please refer to the datasheet of the Flash memory to obtain the sector boundaries and to the flinfo command below.

For the erase to be successful, it is necessary to have the area unprotected (see the protect command).

flinfo

Gives information about the Flash memory. More specifically, it display the start address of all sectors and the protection status. Device type and vendor information are also provided.

It gives also informations about the DataFlash memories. It displays the DataFlash type, the number of pages, the page size, the DataFlash size and the associated virtual address.

go <address> [<parameter list>]

This command executes an absolute branch to <address>. A common usage of the go command is to start the main software application.

If the code initiated by the go command returns, control is given back to U-Boot.

Note that the <address> parameter is mandatory.

It is the responsibility of the user to make sure that there is a valid executable code starting from <address>.

If the parameter list is given, it is passed to the application in its argc and argv global variables.

The main difference between the go and bootm commands is that go expects a pure binary code while bootcmd expects an mkimage-formatted executable.

help

Displays an online help summary of all available commands. A question mark ? is an alias for the help command.

loadb [<address> [<baudrate>]]

Any binary file can be downloaded through the console's serial port with the loadb command. loadb uses the Kermit protocol developed by University of Columbia.

When loadb is run, the console operation through the serial port is suspended and U-Boot waits for a valid Kermit communication. The user then should then instruct the terminal emulator to send a file in Kermit format.

If <address> is specified, the received file will be stored at this address. Otherwise, the value of loadaddr is used as the default load address.

Note that the boot file cannot be downloaded into Flash memory. To program Flash memory, the file must be first downloaded into RAM, and then copied into Flash with the cp command.

Specifying <baudrate> switches the serial port speed temporarily for the Kermit transfer. When the transfer is finished, the original console speed is restored.

md[.<size>] [<address> [<length>]]

md dumps areas of memory. The <size> parameter defines the width of the read transaction of the memory and the display format. l stands for long 32-bit words, w for 16-bit half-words, and b stands for bytes. If <size> is not specified, the last used value is assumed, l being the default.

If specified, <address> defines the address of the first data to display. If unspecified, the dump address is the sum of the last <address> + <length>. Note that <address> must be specified at least once since it has no default value.

If specified, <length> gives the number of words to be dumped. Note that this is not necessarily the number of bytes, but rather the number of data items of the specified size. The default is 64.

mm[.<size>] [<address>]

nm[.<size>] [<address>]

mm and nm provide the user with the possibility of modifying areas of memory manually. The address of the first memory location to be modified can be specified by the

<address> argument. If <address> remains unspecified, the last used address is retained.

The width of the word to be modified at each iteration can be specified by the <size> argument. l means that modifications will occur on 32-bit long words, w for 16-bit half-words and b for bytes.

When the mm command is used, the user is prompted to enter the new value at the specified address. Once entered, the address is incremented by a value defined by <size> (4 for l, 2 for w and 1 for b) and the same iteration restarts. When the user has finished data modification, he can exit by typing Ctrl-C.

nm has the same behavior except the address is not auto-incremented between modifications. This is convenient to program several values in sequence in the same location.

Note that neither mm nor nm can modify the contents of Flash and DataFlash memories. Only cp can do that.

mw[.<size>] <address> <value> [<length>]

mw fills <length> words of data in memory with <value>. The start address is <address>, and the size of a word (and of <value>) is defined by <size>. l specifies a 32-bit long word, w a 16-bit half-word, b a byte.

If <size> is unspecified, the last size is retained, defaulting to l.

If <length> is unspecified, 1 is assumed.

nm[.<size>] [<address>]

See mm.

printenv

Displays the value of all defined environment variables.

protect <on | off> <start> <stop>

Flash memory areas can be soft protected against unwanted corruption or erasure by setting a protection flag. This operation is performed with the protect on command.

When an erase or a write needs to be performed, protect off removes the protection.

<start> and <stop> should contains the first and last addresses of the area to protect/unprotect.

run <envvar>

The run command implements the scripting capability of U-Boot. The <envvar> parameter is expected to be an environment variable that contains a U-Boot script. This script is a list of commands to be executed, separated by semi-colons.

saveenv

Saves all environment variables in a non-volatile memory area.

setenv <variable> <value>

setenv creates new environment variables, and can modify their value.

tftpboot [<address> <filename>]

tftp [<address> <filename>]

tftp is an alias for tftpboot. This command is used to download data from a tftp server. To operate successfully, it is required that the serverip variable be set and reflect the IP address of an host where a tftp service is active.

<address> and <filename> are optional arguments. <filename> specifies the name of the file to be downloaded as it is known by the tftp server, including the directory path. <address> should be set to the address (hex) where to store the file on the local host.

If <address> and <filename> are not specified, the file name defaults to the contents of the reserved variable bootfile, and the load address defaults to the contents of the reserved variable loadaddr.

Note that the boot file cannot be downloaded into Flash or DataFlash memory. To program Flash or DataFlash memory, the file must be first downloaded into regular RAM, and then copied into Flash or DataFlash with the cp command.

version

Displays the version number of U-Boot.