AJ010
User's Guide

Version 2.04

March 4, 2005

Atmark Techno Co., Ltd.
http://www.atmark-techno.com/

Armadillo Official Web-Site
http://armadillo.atmark-techno.com/

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1. About This Manual

This manual provides the following information necessary for using the Armadillo-J.

- Rewriting Flash Memory
- Basic Operation
- Kernel and Userland Building
- Application Development

We hope the information contained in this document will help you get the best functionality out of the Armadillo-J.

1.2. Typographical Conventions

The following font conventions are used in this document.

Table 1-1 Fonts

Font	Description
Font in text	Text
`[PC ~]$ ls`	Prompt and user input character strings Command names Directory and file names

1.3. Conventions in Command Input Examples

The command input examples contained in this manual are based on the assumed execution environment associated with the respective display prompt. The directory part "/" will differ depending on the current directory. The home directory of each user is represented by "~".

Table 1-2 Relationship between Prompt and Execution Environment

Prompt	Command Execution Environment
`[PC /]#`	Executed by privileged user on work PC
`[PC /]$`	Executed by general user on work PC
`[AJ /]#`	Executed by privileged user on Armadillo-J
`[AJ /]$`	Executed by general user on Armadillo-J

1.4. Trademarks

Armadillo is a registered trademark of Atmark Techno, Inc.
Other company and product names in this document are either trademarks or registered trademarks of their respective company or organization.

1.5. Acknowledgements

The software used in the Armadillo-J consists of Free Software and Open Source Software. They are the achievements of many developers from around the world. We would like to take this opportunity to thank all these developers.

uClinux is supported by the achievements of D. Jeff Dionne, Greg Ungerer, David McCullough and all of the people participating in the uClinux development list.

uClibc and Busybox have been developed and are maintained by Eric Andersen.

2. Safety Precautions

2.1. Safety Precautions

Before using the Armadillo-J, read the following safety precautions carefully to assure correct use.

This product uses the semiconductor components designed for generic electronics equipment such as office automation equipment, communications equipment, measurement equipment and machine tools. Do not incorporate this product into devices such as medical equipment, traffic control systems, combustion control systems, safety equipment, etc. which can directly threaten human life or pose a hazard to the body or property due to malfunction or failure. Moreover, those products incorporating semiconductor components can be caused to malfunction or fail due to foreign noise or surge. To ensure there will be no risk to life, the body or property even in the event of malfunction or failure, be sure to take all possible measures in the safety system design, such as using protection circuits like limit switches or fuse breakers, or system multiplexing.

2.2. Operational Precautions

To avoid degradation, damage, malfunction or fire, the following operational precautions must be observed when handling the product.

- Power-On:
  While the Armadillo-J and peripheral circuits are turned on, be sure not to connect or disconnect any expansion I/O connectors.

- Static Electricity:
  The Armadillo-J incorporates CMOS devices. Until it is used, store it safely in the provided antistatic package.

- Interfaces:
  Do not connect signals other than specified to each interface (external I/O, RS232C or Ethernet). Use caution in polarity and input/output direction

- Impact/Vibration:
  Do not apply an excessive impact such as a drop or collision.
  Do not put the product on anything vibrating and/or rotating. Do not apply strong vibration or centrifugal force.

- Excessive High/Low Temperatures and High Humidity:
  Do not use the product in environments where it would be subject to excessive high/low temperatures or high humidity.

- Dust:
  Do not use the product in dusty areas.

2.3. Software Related Precautions

The software and documentation contained in this product are provided "AS IS" without warranty of any kind including any warranty of merchantability or fitness for a particular purpose, reliability, or accuracy. Furthermore, Atmark Techno does not guarantee any outcomes resulting from the use of this product.

3. Overview of Armadillo-J

At only half the size of a credit card, the Armadillo-J is an Ethernet-enabled ultra compact network computer board employing a 32-bit ARM processor (NetSilicon NS7520).

3.1. Features

- Linux:
  The Armadillo-J uses Linux (μCLinux Kernel 2.4.22 based) as its standard OS, providing a wide range of software resources and proven operational stability. Additionally, utilizing the GNU development environment facilitates a smooth development process.

- Network (Ethernet):
  The Armadillo-J supports 10Base-T/100Base-Tx based communication.

- Serial Port:
  The Armadillo-J provides a RS-232C (D-sub 9) compliant serial port for connection with serial devices at a data rate of 600bps to 230,400bps.

- General Purpose Parallel I/O (GPIO)
  The Armadillo-J provides a general purpose parallel I/O (5-pin) to control external devices

3.2. Functions and Application Examples

3.2.1. Device Control via Network

The Armadillo-J has the following functionality at shipment:

- Serial – Ethernet Conversion
- GPIO Control via Network

These functions can be used "as is" for device control via network.
For more information, refer to the Startup Guide.

3.2.2. Linux Terminal

The Armadillo-J can be used as a standard Linux terminal by writing the base image contained in the supplied CD to the Armadillo-J.
The following applications run on the base image:

- telnet
- ftp server
- Web server

For more information, refer to Section 5, "Rewriting Flash Memory" and Section 6, "Getting Started."

3.2.3. Development of Original Embedded Devices

Being able to use the GNU development environment and open source code, a developer can easily develop an original embedded device by customizing the Armadillo-J. For more information, refer to Section 10, "Creating Your Own Application."

4. Before Getting Started

4.1. Preparations

Please make the following preparations before using the Armadillo-J.

- PC
  One Linux or Windows PC that has one or more serial ports.

- Serial Cross Cable
  One D-Sub9-pin (male-male) cable for cross connection.

- Development Kit, Supplied CD-ROM (hereafter called the "supplied CD")
  The supplied CD contains various manuals and source code for the Armadillo-J.

- Serial Console Software
  A serial console program such as minicom or Tera Term (Linux software is contained in the supplied CD in the "tools" directory).

- AC Adapter
  A DC8 to 48V output AC adapter (power consumption of the Armadillo-J is 1.2W).

4.2. Connections

Connect the Armadillo-J and the PC using a serial cross cable as shown in Figure 4-1.

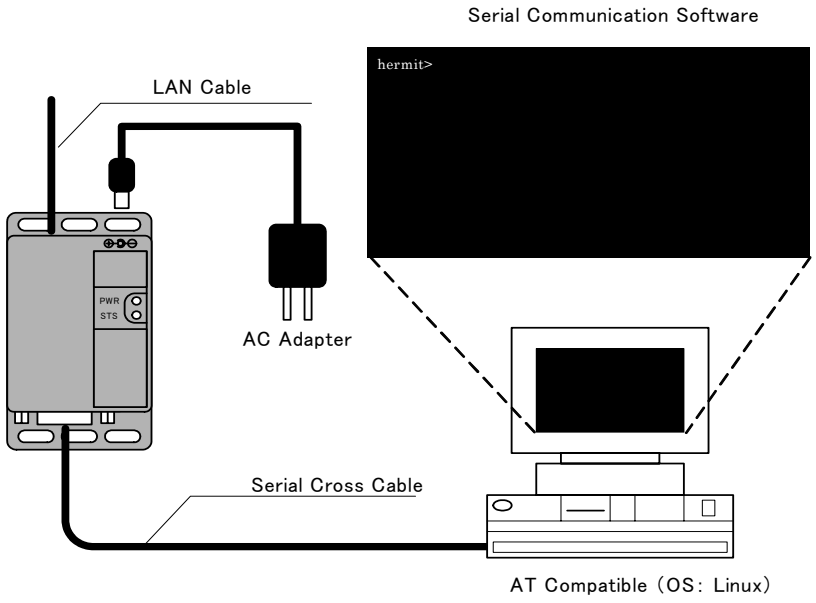


Figure 4-1 Connecting Armadillo-J and Work PC

5. Rewriting Flash Memory

The functionality of the Armadillo-J can be altered by rewriting the flash memory. This chapter explains how to rewrite the flash memory by giving an example of writing a "base image" which allows the Armadillo-J to function as a Linux terminal.

If the downloading of the image fails for any reason, the Armadillo-J may not boot normally. Be careful of the following points when performing a rewriting.

- Do not power off the Armadillo-J.
- Do not disconnect the serial cable connecting the Armadillo-J to the Work PC.

If the Armadillo-J cannot be booted normally, refer to Section 12.1, "The Armadillo-J Doesn't Boot Properly."

5.1. Installing the Downloader

Install the downloader (hermit) on the PC. The downloader is used to rewrite the Armadillo-J flash memory.

1) Linux:
Expand the `tools/hermit-1.3-armadillo.tgz` file contained in the supplied CD. This must be done by a user with root privileges.

```
[PC ~]# tar xzvf hermit-1.3-armadillo.tgz -C /
```

Figure 5-1 Example of Expand Command

The supplied CD also contains rpm (Red Hat package) and deb (Debian package) versions. Select the one suitable for your OS.

2) Windows:
Expand Hermit host for Win32 (`tools/hermit-1.3-armadillo-4_win32.zip`) contained in the supplied CD to an appropriate folder.

5.2. Rewriting Procedure

Rewrite the flash memory according to the following procedure.

5.2.1. Setting Jumper Pins

Before power-on, set jumper pins as follows:

- JP1: Set to "Short"
- JP2&3: Set according to user environment (if no connection is made to GPIO, set to "Short").
- JP4: Set to "Short"

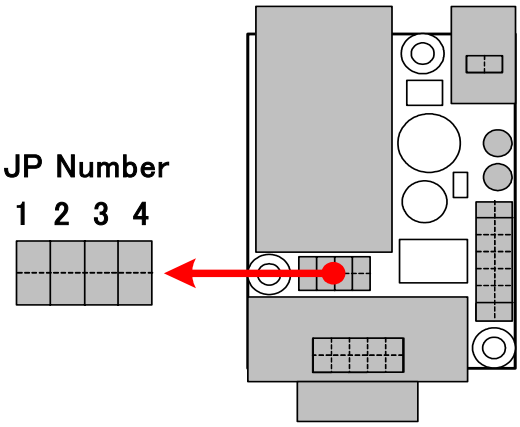


Figure 5-2 Jumper Position

Table 5-1 Action Corresponding to Each Jumper Setting

Setting	Short (Close)	Open
JP1	Connects network module to TXD/RXD of the serial interface.	Disconnects network module from TXD/RXD of the serial interface.
JP2	Connects network module to CTS/RTS of the serial interface.	Disconnects network module from CTS/RTS of the serial interface.
JP3	Connects network module to DTR/DSR/DCD of the serial interface.	Disconnects network module from DTR/DSR/DCD of the serial interface.
JP4	Boots in rewriting mode	Boots in normal mode

5.2.2. Downloading a Rewriting Image

1) Linux:
Activate a terminal on the Linux PC and then enter the hermit command.
A base image (`base.img`) is specified as the file name in Figure 5-3.

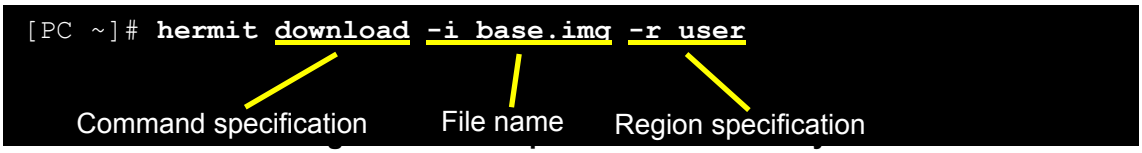


Figure 5-3 Command Input Example

If the serial port on the work PC is something other than `ttyS0`, add the `[--port "port name"]` option.

When the rewriting completes, `serial : completed xxxxxxxx (xxxxxxxx)` will be displayed on the screen.
Once it has completed, set JP4 to Open and reboot the Armadillo-J to activate the base image.

2) Windows:
Start Hermit host for Win32 which was expanded in Section 5.1, "Installing a Downloader".

Set the parameters according to Table 5-2.

Table 5-2 Example of Parameters When Using Hermit for WIN32

Parameter	Description	Setting Value
Port	Name of a port connecting to the Armadillo-J	COM1 (when using COM1)
Input file	Name of a target file for rewriting	base.img (include path name)
Region	Specification of a writing area	user (fixed)

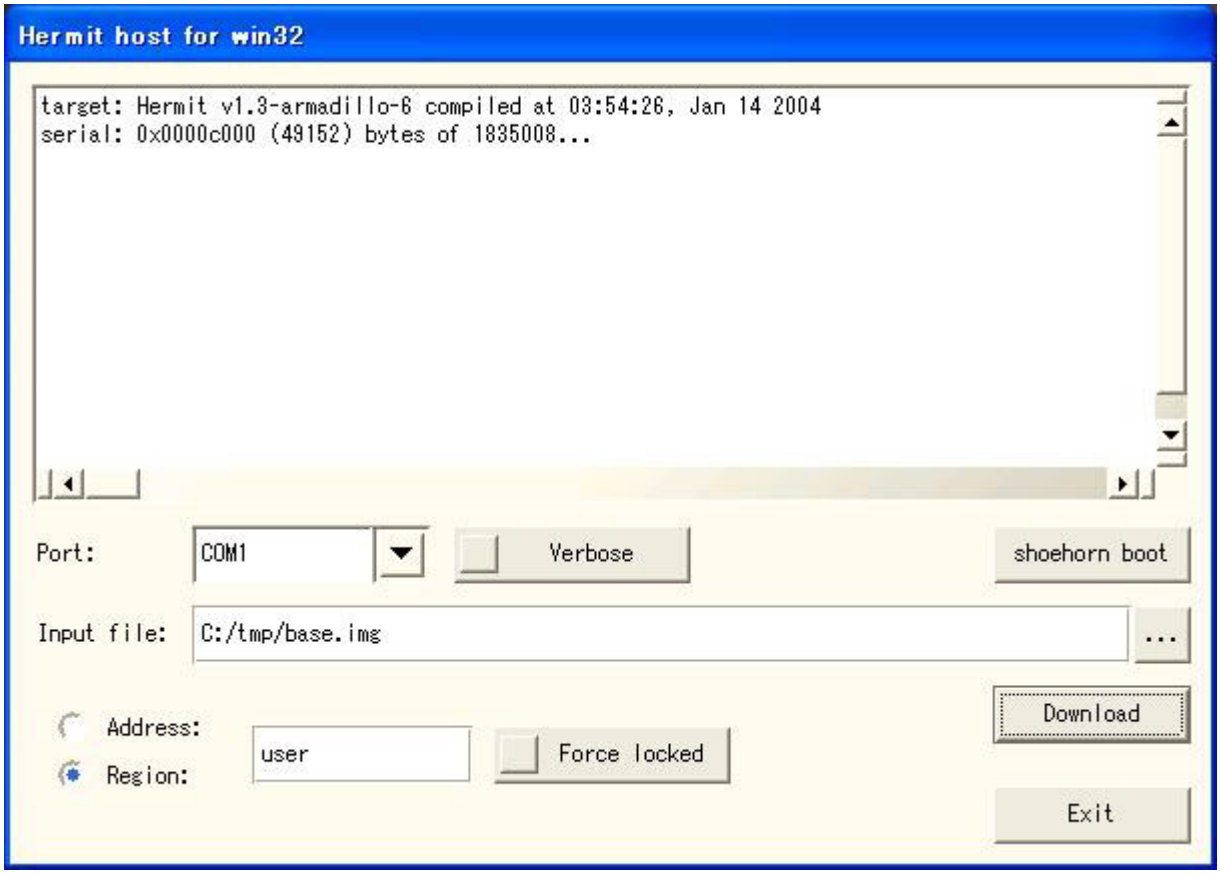


Figure 5-4 Example of Downloading a Rewriting Image

When the rewriting completes, "serial : completed xxxxxxxx (xxxxxxxx)" will be displayed on the screen. Once it has completed, set JP4 to Open and reboot the Armadillo-J to activate the base image.

6. Getting Started

The supplied CD contains three different flash memory images in the `image` directory. They each have a different functionality.

- Recovery Image (`recover.img`)
  This is the image already written to the flash memory at the time of purchase. It provides Serial-Ethernet Conversion, GPIO Control via Network and so on. You can restore the Armadillo-J to its default state by writing this image to the flash memory. For information on how to use this image, refer to the Startup Guide.

- Base Image (`base.img`)
  This image is used as a basis for application development. It provides telnet, ftp and Web server functionality. You can log into the Armadillo-J via serial port or network.

- JFFS2 Image (`jffs2.img`)
  This image provides a file system that allows the user to save all modified files. It has the same functionality as the base image.

This section describes how to use the base image. For the base image memory map and command list, refer to 13.2, "About The Base Image."

6.1. Before Booting

If the base image is not yet written to the flash memory, write the base image to the flash memory first according to the procedure described in Chapter 5, "Rewriting Flash Memory."

Connect the Armadillo-J to the PC using a serial cable and start the serial console program. Serial communication settings are shown in Table 6-1.

Table 6-1 Serial Communication Settings

Parameter	Setting
Download Rate	115,200bps
Data Length	8bit
Stop Bit	1bit
Parity	None
Flow Control	None

6.2. Booting

To boot the base image, set JP4 to Open and turn the Armadillo-J on.
When the Armadillo-J successfully boots, the following startup log is generated.

```
Copying        kernel...done.
Linux version 2.4.22-uc0-aj1 (somebody@colinux) (gcc version 2.95.3 20010315
(release)(ColdFire patches - 20010318 from
http://fiddes.net/coldfire/)(uClinux XIP and shared lib patches from
http://www.snapgear.com/)) #1 Tue Jun 8 05:13:34 UTC 2004
Processor: ARM/VLSI ARM 7 TDMI revision 0
Architecture: NET+ARM
fixup_netarm: Kernel memory start 0x00000000 end 0x000ea000
On node 0 totalpages: 2048
zone(0): 0 pages.
zone(1): 2048 pages.
zone(2): 0 pages.
Kernel command line: root=/dev/rom0
setup_timer : T2 CTL = D0000008
setting up timer IRQ
Calibrating delay loop... 8.93 BogoMIPS
Memory: 8MB = 8MB total
Memory: 6288KB available (728K code, 1072K data, 36K init)
Dentry cache hash table entries: 1024 (order: 1, 8192 bytes)
Inode cache hash table entries: 512 (order: 0, 4096 bytes)
Mount cache hash table entries: 512 (order: 0, 4096 bytes)
Buffer cache hash table entries: 1024 (order: 0, 4096 bytes)
Page-cache hash table entries: 2048 (order: 1, 8192 bytes)
POSIX conformance testing by UNIFIX
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
Starting kswapd
Net+ARM serial driver version 0.2 (2002-02-27) with CONSOLE enabled
ttyS00 at 0x0001 (irq = 15) is a NetARM
ttyS01 at 0x0002 (irq = 13) is a NetARM
ns7520port: port driver, (C) 2004 Atmark Techno, Inc.
Software Watchdog Timer: 0.05, timer margin: 60 sec
NS7520 Ethernet Driver Initialized
uclinux[mtd]: RAM probe address=0xe8b98 size=0xda000
uclinux[mtd]: root filesystem index=0
Initializing Armadillo-J MTD mappings
 Amd/Fujitsu Extended Query Table v1.0 at 0x0040
number of CFI chips: 1
cfi_cmdset_0002: Disabling fast programming due to code brokenness.
Creating 7 MTD partitions on "Flash":
0x00000000-0x00020000 : "Flash/Reserved"
0x00020000-0x00040000 : "Flash/Hermit"
0x00040000-0x00200000 : "Flash/Image"
0x00040000-0x000b0000 : "Flash/Kernel"
0x000b0000-0x00200000 : "Flash/User"
0x001e0000-0x001f0000 : "Flash/Backup"
0x001f0000-0x00200000 : "Flash/Config"
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 512 bind 512)
VFS: Mounted root (romfs filesystem) readonly.
init started:  BusyBox v0.60.5 (2004.06.08-05:15+0000) multi-call binary
Mounting proc: done
Mounting var: done
Mounting /etc/config: done
Populating /etc/config: FLATFSD: created 6 configuration files (502 bytes)
Populating /var: done
Mounting /home/guest/pub: done
```

```
Setting hostname: done
Setting up interface lo: done
Running local start scripts.
Starting flatfsd: done
Setting up network: Starting DHCP for interface :
ns7520_eth: PHY (0x13, 0x78e2) = LXT971A detected
ns7520_eth: link mode 100 Mbps full duplex (auto)
done
Starting inetd: done
Starting thttpd: done
Setting local time: done
Starting ledctrl: done

aj login:
```

Figure 6-1 Startup Log

The base image supports the following two types of users.

Table 6-2 User Name and Password for Log into the Serial Console

User Name	Password	Privilege
root	root	Super User
guest	guest	General User

6.3. Directory Structure

The directory structure is shown in Table 6-3.

Table 6-3 Structure of Directory

Directory Name	Write	Description	File System
/bin	×	Applications	romfs
/dev		Device nodes	
/etc		System setting	
/etc/config	o	System settings (saved)	ramfs
/etc/default	×	System settings restoration	romfs
/lib		Common library	
/mnt		Mount point	
/proc		Process data	
/root		Root home directory	
/sbin		System management command	
/usr		Common user data	
/home		User home directory	
/home/guest/pub	□	ftp data transmission and receive	ramfs
/tmp		Temporary storage	
/var		Modified data	

Table 6-4 Writing Attributes

Mark	Description
×	Writing is not allowed.
□	Writing is allowed. Data is not saved at power OFF/ON.
o	Writing is allowed. With flatfsd, data is saved even at power OFF/ON.

6.4. Saving Data

In the base image, the `/etc/config` directory is used to save the system setting files.
This directory uses flatfsd, which enables modified data to be saved even after power off. flatfsd is executed when the base image is booted.

- Saving data:
  When the USR1 signal is sent to flatfsd, the data in the `/etc/config` directory is written to the setting data area in the flash memory. The USR1 signal can be sent to the flatfsd by entering the command `killall -USR1 flatfsd`. The setting data area in the flash memory is limited to 64kbytes. Therefore, data exceeding this size cannot be saved.

- Restoring data:
  When the command `flatfsd -r` is entered, the flatfsd reads data from the setting data area in the flash memory and copies it to the `/etc/config` directory. In the base image the `flatfsd -r` command is executed during system startup, thus setting data is automatically restored. If the data written in the setting data area of the flash memory is not correct, copy the content of the `/etc/default` directory to `/etc/config`.

If a system data file in the `/etc/config` directory (i.e. network settings) has been updated, enter the command `killall -USR1 flatfsd` to save the data. This command must be executed by a user with root privileges.

```
[Armadillo-J ~]# killall -USR1 flatfsd
[Armadillo-J ~]#
```

Figure 6-2 Example of Save Data Command (killall -USR1 flatfsd)

6.5. Exiting

The Armadillo-J can be shutdown by turning off the power. To modify the content of the `/etc/config` directory for system settings and retain it at the next boot, issue the `killall -USR1 flatfsd` command before shutting the Armadillo-J down.

6.6. Network Settings

Network settings can be modified by editing the `/etc/config/network` file within the Armadillo-J. Be sure to log into the Armadillo-J with root privileges when performing this task. `vi` can be used as the editor for modification.

6.6.1. Using a Fixed IP Address

An example of settings for specifying a fixed IP address is shown in Table 6-5.

Table 6-5 Details of Network Setting

Parameter	Setting Value
IP Address	192.168.10.10
Netmask	255.255.255.0
Broadcast Address	192.168.10.255
Default Gateway	192.168.10.1

```
#!/bin/sh

PATH=/bin:/sbin:/usr/bin:/usr/sbin

ifconfig eth0 192.168.10.10 netmask 255.255.255.0 \
          broadcast 192.168.10.255 up
route add default gw 192.168.10.1 eth0
```

Figure 6-3 Example of Network Settings (Fixed IP Address)

6.6.2. Using DHCP

An example of the network settings for obtaining an IP address using DHCP is shown in Figure 6-4.

```
#!/bin/sh

PATH=/bin:/sbin:/usr/bin:/usr/sbin

/bin/dhcpcd &
```

Figure 6-4 Example for Network Setting (by DHCP)

To retain the modified settings even after power-off, you need to issue the `killall -USR1 flatfsd` command before exiting.

6.7. telnet Login

You can log into the system with the following user name and password. Root login is not allowed. If you need root privileges for a task, login to the system as guest first and then change to root with the `su` command.

Table 6-6 User Name and Password for telnet Login

User Name	Password
guest	guest

6.8. File Transfer

The Armadillo-J supports FTP file transfer. Login to the system with the following user name and password. The home directory is `/home/guest`. Go to the `/home/guest/pub` directory to upload data.

Table 6-7 User Name and Password for FTP

User Name	Password
guest	guest

6.9. Web Server

A small HTTP server called thttpd is run which allows the user to browse the Armadillo-J from a Web browser.

Data directory: `/home/www`

URL: `http://(IP address of the Armadillo-J)/` (Example: `http://192.168.0.100/`)

7. Preparing Development Environment

The Armadillo-J allows for development on Linux or Windows.
If developing on Windows you will need coLinux, a utility that creates a Linux environment in Windows. For information on how to install coLinux, refer to 13.1. "Building a Development Environment in Windows."

7.1. Installing the Tool Chain

From the supplied CD, `execute cross-dev/arm-elf-tools-20030314.sh`. Be sure to execute this with root privileges. The `arm-elf-tools-20030314.sh` is an installer for the development tool chain that includes a compiler, binutils and uClibc libraries. The development tool chain is installed at `usr/local/bin`.

```
[PC ~]# sh ./arm-elf-tools-20030314.sh
```

Figure 7-1 Example of Expanding Development Tool Chain

7.2. Configuring Environment Variables

To make the Development Tool Chain easier to use, you will need to add the directory that contains the executables to the PATH environment variable. The method for doing this differs dependent on type of shell. For more information, refer to your shell manual.
The following example is in bash.

```
[PC ~]$ export PATH="$PATH:/usr/local/bin"
[PC ~]$ echo $PATH
/usr/bin:/bin:/usr/bin/X11:/usr/sbin:/sbin:/usr/local/bin
[PC ~]$
```

Figure 7-2 Setting the PATH Environment Variable (bash)

7.3. Preparing Source Code

At uClinux.org, all source code for the Kernel, libraries and applications are collectively distributed as the Full Source Distribution under the file name `uClinux-dist-YYYYMMDD.tar.gz`. The Armadillo-J uses uClinux-dist with some modifications. The uClinux-dist for the Armadillo-J is contained in the dist directory on the supplied CD under the file name `uClinux-dist.tar.gz`.
`uClinux-dist.tar.gz` can be expanded to anywhere appropriate. For convenience, here it is expanded to `~/`.

```
[PC ~]$ gzip -cd uClinux-dist.tar.gz | tar xvf -
[PC ~]$ ls
uClinux-dist
[PC ~]$
```

Figure 7-3 Example of Expanding Source Code

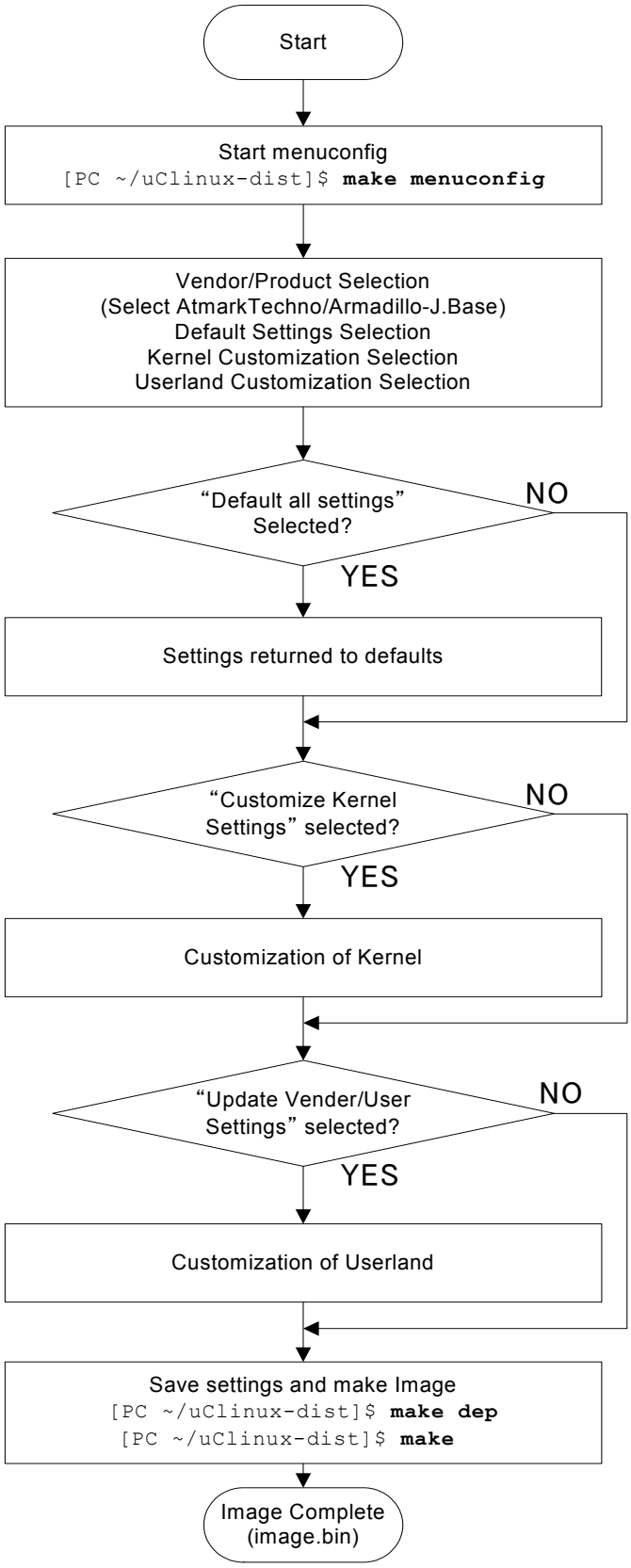
8. Image Creation with uClinux-dist

This section shows how to create an image file for writing to flash memory using the installed development tool chain and source code. uClinux-dist is designed to allow customization in the same way as the Linux Kernel build-system. This will be specifically easy for developers who have experience in compiling a Linux Kernel. Here, `menuconfig` is used for the example.

8.1. Work Flow

Figure 8-1 shows the work flow for creating an image file.

Figure 8-1 Work Flow for Creating an Image File



The work process can be roughly divided into three areas; restoring defaults, kernel and userland.

If you select to default all settings, both the kernel and userland are returned to default. In addition, all work done before selecting the default all settings is discarded. Therefore, take care when selecting this.

8.2. Menuconfig

As with the Linux kernel, the uClinux-dist build system allows the use of `config`, `menuconfig` and `xconfig`. Here, uClinux-dist is built using `menuconfig`.

First go to the uClinux-dist directory expanded in 7.3, "Preparing Source Code" and then enter the `make menuconfig` command.

Example 8-1 Executing the make menuconfig command

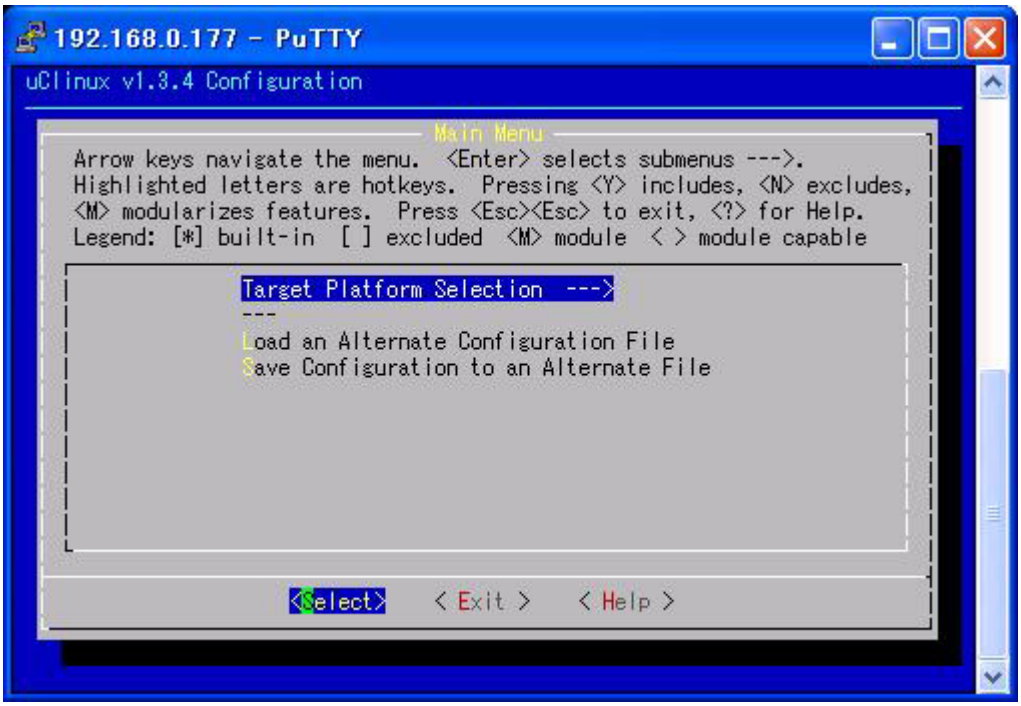
```
[PC ~]$ cd uClinux-dist
[PC ~/uClinux-dist]$ make menuconfig
```

In make `menuconfig`, a screen control program that requires the ncurses library is compiled at the first execution. The ncurses library must therefore be pre-installed. Dependant on the operating system, not only the ncurses library package but also the development package for this library may also be required.

8.2.1. Main Menu

When the above command is entered, a screen as shown in Figure 8-2 appears. This is the uClinux-dist main screen.

Figure 8-2 uClinux v3.1.0 Configuration Main Menu

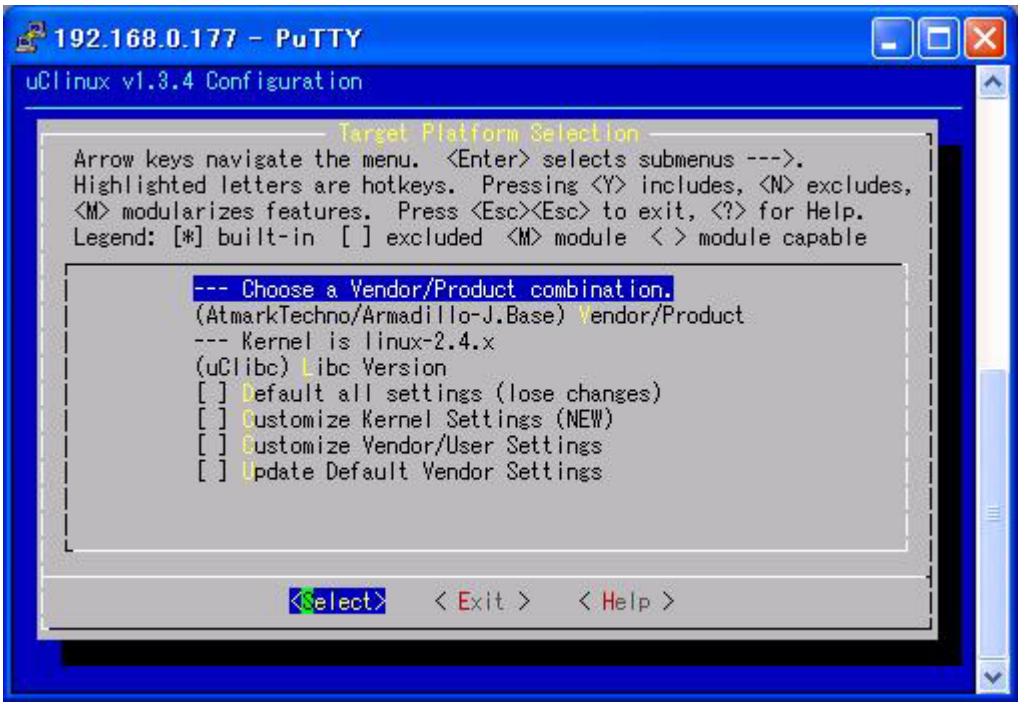


In `menuconfig`, menu selection is made using the arrow keys on the keyboard. The "`--->`" mark at the right side of a menu option means that you can move to that menu. When selecting a menu, make sure that the "`<select>`" at the bottom of the screen is highlighted before pressing the enter key.

8.2.2. Target Platform Selection Menu

From the main menu select `Target Platform Selection` to move to that submenu screen.

Figure 8-3 Target Platform Selection



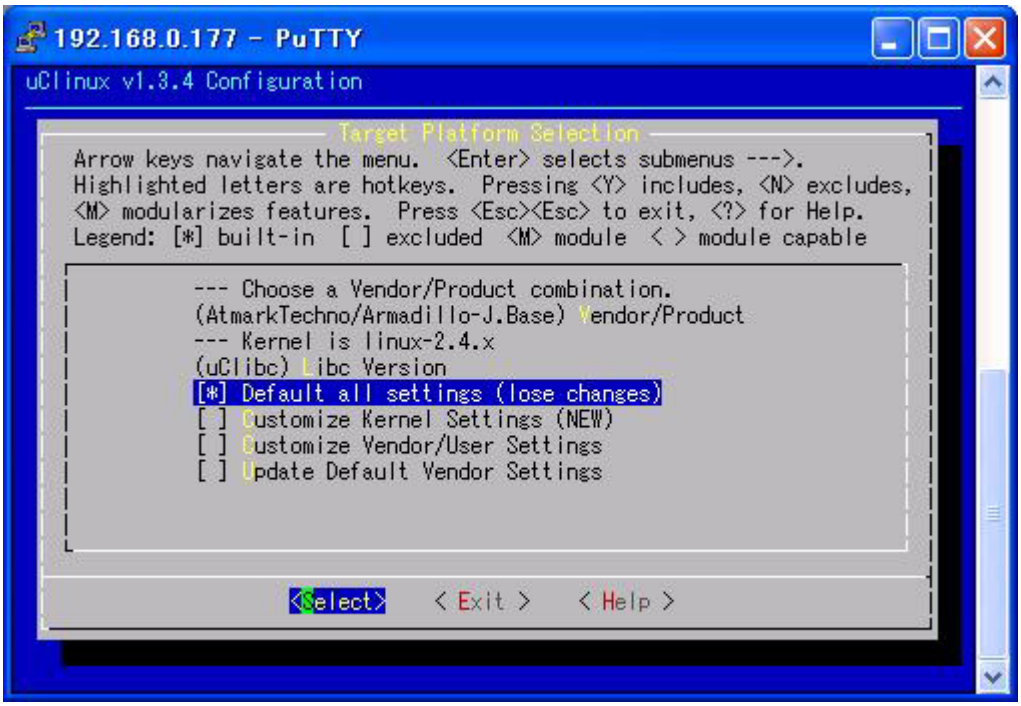
Place the cursor on `(AtmarkTechno/Armadillo-J.Recover) Vendor/Product` and press the enter key to select other images. Here, the base image that runs as a Linux terminal will be created.
From the list, select `Armadillo-J.Base`.

Figure 8-4 Target Platform Selection

Select `<Exit>` with the arrow keys to return to the previous screen.

Then select `Default all settings (lose changes)`. The `[ ]` mark at the right side of the menu represents the state of selection. Move the cursor up or down with the arrow keys and make a selection with the space bar. When the selection is made, the "*" mark will appear in the brackets.

Figure 8-5 Selecting Default all setting



After selecting `Default all settings (lose changes)`, select `<Exit>` to return to the main menu. When you select `<Exit>` in the main menu, you will be asked if you want to save the settings. Then select `<Yes>` to save the settings.

A setting log appears on the screen and you will then return to the prompt.

8.3. Building

Actual compiling or creating an image file is called "building". The `make` command is used for the build. At the prompt, enter the following shown in Example 8-2.

Example 8-2 Build Command

```
[PC ~/uClinux-dist]$ make dep; make
```

The first `make dep` command is used to resolve the dependency of the Linux kernel. The command will be familiar to those with experience compiling Linux Kernels up to version 2.4.

The next make command performs the entire build processes and finally creates an image file that can be written to the flash memory. If the build process is successfully completed, an `image.bin` file will be created under the `uClinux-dist/images/` directory.

For information on how to write the created image file to the Armadillo-J, refer to Section 5. "Rewriting Flash Memory."

In this manual the image file refers to a binary file that combines the uClinux kernel and userland. Writing this binary file to the flash memory with Hermit will make uClinux bootable on the Armadillo-J.

9. uClinux-dist Configuration

This chapter describes how to create a customized image using the uClinux-dist menu.

9.1. Changing Kernel Settings

To customize the Kernel, select `Customize Kernel Settings` from the Target Platform Selection of uClinux Configuration. From this menu exit the uClinux v1.3.4 Configuration main menu to display the Linux Kernel Configuration menu screen. The Linux Kernel Configuration itself is the same as the main version of Linux.

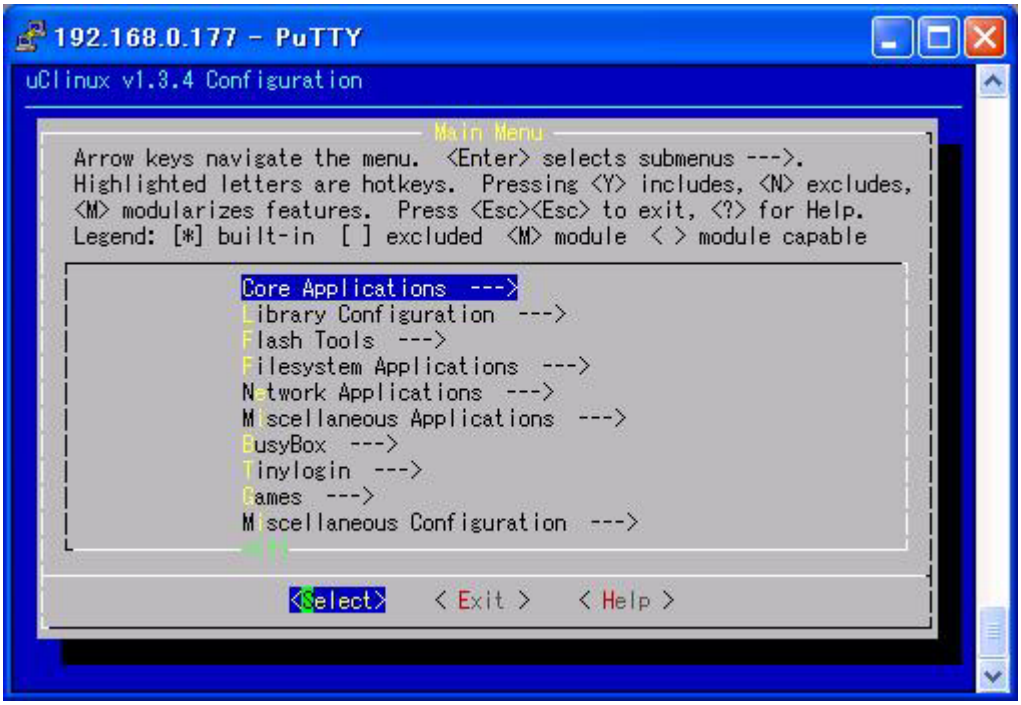
When customization of the Kernel is completed, select `<Exit>`. When asked whether or not to save the settings, select **`Yes`**.

If you do not need to customize userland, execute a build. For information on building, refer to section 8.3, "Building."

9.2. Customizing Userland Settings

To customize userland, select `Customize Vendor/User Settings` from the Target Platform Selection menu of uClinux Configuration. From this menu exit the uClinux v1.3.4 Configuration main menu to display the userland main menu.

Figure 9-1 Userland Main Menu



The userland customization screen allows you to add applications provided by the uClinux Full Source Distribution to the image, or to customize the applications to be added.

uClinux-dist has a collection of more than 150 applications. In the uClinux Configuration menu they are classified into several categories.

- Core Applications

  Core Applications contains the basic applications necessary for running as a system. This section allows selection of `init` for system initialization and `login` for user authentication.

- Library Configuration

  Library Configuration allows selection of libraries required by applications.

- Flash Tools

  Flash Tools have a selection of applications associated with flash memory. In Armadillo-J.Base a network update application called `netflash` is selected in this category.

- Filesystem Applications

  Filesystem Applications have a selection of applications associated with the file system. In Armadillo-J.Base `flatfsd` is selected. Others included are `mount`, `fdisk`, ext2 file system, reiser file system and Samba.

- Network Applications

  Network Applications have a selection of applications associated with networking. In addition to `dhcpcd-new`, `ftpd`, `ifconfig`, `inetd` and `thttpd` used in Armadillo-J.Base, `ppp` and a wireless network utility are also included.

- Miscellaneous Applications

  Miscellaneous Applications include other applications not belonging to the above categories. Generic Unix-based commands (`cp`, `ls`, `rm`, etc.), editors, audio-related programs, and script language interpreters are also included.

- Busybox

  Busybox can be customized here. Busybox is a single command having multiple command functions and has a good track record in embedded Linux systems. Since Busybox provides many customization functions, it is classified into a separate section.

- Tinylogin

  The Tinylogin application also provides multiple command functions associated with authentication such as `login`, `passwd` and `getty`. Since it provides many customization functions, it is classified into a separate section.

- MicroWindows

  MicroWindows is a graphical window environment targeted towards embedded equipment. It is well suited to the development of LCD-based equipment.

- Games

These are games. No further explanation needed.

- Miscellaneous Configuration

  Numerous configuration options are contained here.

- Debug Builds

  Contains numerous debug options. This category is used when debugging applications during development.

There is no guarantee that every application will run on all architectures. However, in most cases, they can be run with minor modifications (at source code level or with the Makefile).

10. Creating Your Own Application

This section shows how to create the Hello World application and run it on the Armadillo-J. Compiling is performed outside of the uClinux-dist package (this is called OTC: Out of Tree Compile).

Even if compiling is performed outside of uClinux-dist, since it uses the uClinux-dist supplied libraries and the Makefile, a uClinux-dist directory previously built for the Armadillo-J is required. If not yet built, refer to Chapter 8, "Image Creation with uCLinux" to build it.

Please observe the following precautions when running your own application on Armadillo-J.

When developing an application for controlling the I/O or serial ports, be sure to properly set jumper pins, I/O port related registers and devices connecting to an I/O port. Otherwise, this can cause malfunction or failure of equipment when running the Armadillo-J. For information on jumper pins and I/O port related registers, refer to the Hardware manual.

10.1. Creating a Directory

The directory to be used can be made anywhere. Here, `~/hello` is used.

Example 10-1 Preparing a Directory for Hello World

```
[PC ~]$ mkdir hello
[PC ~]$
```

10.2. Creating a Makefile

A Makefile sample template is available in `sample/hello/Makefile` on the supplied CD. Copy this file.

Example 10-2 Hello World Makefile

```
ifndef ROOTDIR
ROOTDIR=../uClinux-dist ----------------------------------  ①
endif
ROMFSDIR  = $(ROOTDIR)/romfs
ROMFSINST = romfs-inst.sh
PATH      := $(PATH):$(ROOTDIR)/tools

UCLINUX_BUILD_USER = 1 -----------------------------------  ②
include $(ROOTDIR)/.config
LIBCDIR = $(CONFIG_LIBCDIR)
include $(ROOTDIR)/config.arch


EXEC = hello ---------------------------------------------  ③
OBJS = hello.o -------------------------------------------  ④


all: $(EXEC)

$(EXEC): $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS)

clean:
        -rm -f $(EXEC) *.elf *.gdb *.o

romfs:
        $(ROMFSINST) /bin/$(EXEC)

%.o: %.c
        $(CC) -c $(CFLAGS) -o $@ $< ------------------------------------  ⑤
```

① If `ROOTDIR` is not specified, it is assumed that the `uClinux-dist` directory exists in parallel to the directory you are currently in. If the `uClinux-dist` directory does not exist there, specify it with `ROOTDIR`.
② Define `UCLINUX_BUILD_USER` to select the Userland compiler option.
③ Set the executable file name to be created in the variable `EXEC`. Here, it has been set to `hello`.
④ Specify the object file to be used for the above executable file. If you need to specify multiple files, separate them with a space.
⑤ This is a pattern rule for converting C-source code into object code of the same name. For more information, refer to the Make manual.

10.3. C Source Code

The C code used in Hello World is shown below.

Example 10-3 Hello World Source Code

```
#include <stdio.h>

int main(int argc, char * argv[])
{
        printf("Hello World!\n");
        return 0;
}
```

This is the standard Hello World that can be found in any C textbook. Save this as `hello.c`. It is compiled into `hello.o` according to the Makefile.

10.4. Compiling

Do a `make` as shown in the following example. If the `make` completes successfully, three files are created including `hello.o`, `hello.gdb` and `hello`. `hello` is the executable file.

Example 10-4 Execution of make

```
[PC ~/hello]$ make
[PC ~/hello]$ ls
Makefile  hello  hello.c  hello.gdb  hello.o
[PC ~/hello]$
```

10.5. Installing to the ROMFS Directory

The user area for an image file is created from the `uClinux-dist/romfs` directory. So, in order to include your own application in the image to be written to the flash memory, it needs to be copied to the `romfs` directory.

If, like this time, there is only one generated executable file, the copying is easy. However, it is not so easy when generating several executable files and copying the configuration and data files. To cope with this, the Makefile template provides a `romfs` target.

Example 10-5 Installing Hello World to the ROMFS Directory

```
[PC ~/hello]$ make romfs
romfs-inst.sh /bin/hello
[PC ~/hello]$ ls ../uClinux-dist/romfs/bin/hello
../uClinux-dist/romfs/bin/hello
[PC ~/hello]$
```

10.6. Creating/Executing the Image File

Finally, go to the `uClinux-dist` directory and create the image file.

By specifying the image target with the `make` command, after the userland image file (`romfs.img`) based on the `romfs` directory is created, it is combined with the kernel image file (`linux.bin`) to form the one image file (`image.bin`) that is to be written into the flash memory.

Example 10-6 Creating an Image File

```
[PC ~/hello]$ cd ../uClinux-dist
[PC ~/uClinux-dist]$ make image
    :
    :
    :
[PC ~/uClinux-dist]$ ls images
image.bin  image.bin.cksum  romfs.img
[PC ~/uClinux-dist]$
```

Download the created image file to the Armadillo-J and then execute `hello`. For information on how to download the file, refer to Chapter 5, "Rewriting Flash Memory."

Example 10-7 Execution

```
[AJ ~]# hello
Hello World!
[AJ ~]#
```

11. Flat Binary Format

This chapter describes the Flat Binary Format that is one of uClinux's features. The size of the executable binary files represents a critical problem in the embedded systems targeted by uClinux. The ELF used by generic Linux provides a format rich in flexibility but the size is too large. So, most uClinux adopt a new binary format similar to the traditional "a.out" format.

This section first describes the features of the Flat Binary Format and then provides information on how to compress executable files and change the stack size.

11.1. Features of the Flat Binary Format

The Flat Binary Format has the following features.

- Simplicity
  This simple format design contributes to the execution speed and size of the binary file. Although it has less flexibility than ELF, it can be said that it is a necessary tradeoff for embedded systems.

- Compression
  The Flat Binary Format is a compressible format. There are two compression types, whole file compression and data area only compression. The executable file will be uncompressed when it is loaded, so the activation speed is slow compared to a non-compressed executable file. Since there is no difference between it and a non-compressed file if once it is activated, the format is suitable for programs such as a residential processes which do not repeatedly activate and shutdown.

- Stack Size Field
  The Flat Binary Format has a stack size field that can be changed without re-compiling. In CPUs without a MMU, since it is difficult to extend the stack area dynamically, it has a stack area with fixed stack size. This field can be changed with the `flthdr` tool. Furthermore, it is also possible to specify its stack size at compiling.

- XIP (eXecute In Place)
  The Flat Binary Format is also compatible with XIP. XIP is an abbreviation for "eXecute In Place" (spot execution), and generally refers to the ability of executable files to run directly on ROM where they are stored, without being copied to RAM.

11.2. Compressing an Executable File

The following example shows how to compress the Hello World program created in the previous chapter. The latter part of this section describes how to specify compression when compiling.

11.2.1. Compressing a Compiled Binary File

Here, `flthdr` is used to compress a compiled binary file. `flthdr` is a program that is used to edit or view Flat Binary Format files.

Example 11-1 shows an executable file created in a standard compiling.

Example 11-1 Normal Flat Binary Format

```
[PC ~/hello]$ make
[PC ~/hello]$ flthdr hello
hello
    Magic:        bFLT
    Rev:          4
    Entry:        0x50
    Data Start:   0x3e60
    Data End:     0x4bf0
    BSS End:      0x6bf0
    Stack Size:   0x1000
    Reloc Start:  0x4bf0
    Reloc Count:  0x53
    Flags:        0x1  ( Load-to-Ram )
[PC ~/hello]$
```

Then compress it with `flthdr`.

Example 11-2 Compressed Flat binary Format

```
[PC ~/hello]$ flthdr -z hello ----------------------------  ①
zflat hello --> hello
[PC ~/hello]$ flthdr hello -------------------------------  ②
hello
    Magic:        bFLT
    Rev:          4
    Entry:        0x50
    Data Start:   0x3e60
    Data End:     0x4bf0
    BSS End:      0x6bf0
    Stack Size:   0x1000
    Reloc Start:  0x4bf0
    Reloc Count:  0x53
    Flags:        0x5  ( Load-to-Ram Gzip-Compressed ) ----  ③
[PC ~/hello]$
```

① Pass the compression option `-z` to `flthdr`.
② Display the header of the executable file created with the `flthdr` command.
③ The Gzip-Compressed flag can be seen.

11.2.2. Compression at Compiling

The `FLTFLAGS` environment variable is used for compression at compiling. The following is an example with Hello World.

Example 11-3 Specifying Compression with FLTFLAGS

```
[PC ~/hello]$ make FLTFLAGS=-z  -------------------------  ①
[PC ~/hello]$ flthdr hello  -------^--------------------  ②
hello
    Magic:         bFLT
    Rev:           4
    Entry:         0x50
    Data Start:    0x3e60
    Data End:      0x4bf0
    BSS End:       0x6bf0
    Stack Size:    0x1000
    Reloc Start:   0x4bf0
    Reloc Count:   0x53
    Flags:         0x5  ( Load-to-Ram Gzip-Compressed ) ----  ③
[PC ~/hello]$
```

① Execute make while setting the `FLTFLAGS` environment variable to `-z`.
② Display the header of the executable file created with the `flthdr` command.
③ The Gzip-Compressed flag can be seen.

11.3. Specifying Stack Size

This section introduces two methods of specifying stack size.

11.3.1. Changing the Stack Size of a Compiled Binary File

The stack size is specified with `flthdr -s`. While dependant on architecture, it seems the default stack size value is mostly 4096.

Example 11-4 Changing Stack Size

```
[PC ~/hello]$ flthdr -s 8192 ----------------------------  ①
[PC ~/hello]$ flthdr hello -------------------------------  ②
hello
    Magic:         bFLT
    Rev:           4
    Entry:         0x50
    Data Start:    0x3e60
    Data End:      0x4bf0
    BSS End:       0x6bf0
    Stack Size:    0x2000 ---------------------------------  ③
    Reloc Start:   0x4bf0
    Reloc Count:   0x53
    Flags:         0x1  ( Load-to-Ram )
[PC ~/hello]$
```

① Pass the change stack size option `-s` and the stack size to `flthdr`.
② Display the header of the executable file created with the `flthdr` command.
③ The change to 8192bytes can be seen.

11.3.2. Specifying Stack Size at Compiling

The `FLTFLAGS` environment variable is used to specify stack size at compiling. The following shows an example with Hello World.

Example 11-5 Specifying Stack Size with FLTFLAGS

```
[PC ~/hello]$ make FLTFLAGS='-s 8192' --------------------  ①
[PC ~/hello]$ flthdr hello -------------------------------  ②
hello
    Magic:         bFLT
    Rev:           4
    Entry:         0x50
    Data Start:    0x3e60
    Data End:      0x4bf0
    BSS End:       0x6bf0
    Stack Size:    0x2000 ---------------------------------  ③
    Reloc Start:   0x4bf0
    Reloc Count:   0x53
    Flags:         0x1  ( Load-to-Ram )
[PC ~/hello]$
```

① Execute make while setting the `FLTFLAGS` environment variable to `-s 8192`.
② Display the header of the executable file created with the `flthdr` command.
③ The change to 8192bytes can be seen.

12. Troubleshooting

12.1. The Armadillo-J Doesn't Boot Properly

If the Armadillo-J does not startup properly as a result of writing an improper image file to the flash memory, rewrite the `image/base.img` image contained in the supplied CD to the flash memory following the procedure described in Chapter 5, "Rewriting Flash Memory."

12.2. Restoring Default Settings

You can restore default settings by writing the `image/recover.img` image contained in the supplied CD to the flash memory following the procedure described in Chapter 5, "Rewriting Flash Memory."

13. Appendix

13.1. Building a Development Environment in Windows

A cross-development environment for the Armadillo-J can be built on Windows by utilizing coLinux (http://www.colinux.org/), a form of Linux that can be run on a Windows system. Windows XP and Windows2000 are supported..

13.1.1. Installing coLinux

1) Execute `coLinux-0.6.1.exe` contained in the `colinux` directory of the supplied CD.
2) Specify `c:\colinux` as the install folder. All other settings can be left at their defaults.

Note: If another directory is specified as the install folder, you will need to edit the file prepared under the following procedure (`default.colinux.xml`) and change the directory name as appropriate.

13.1.2. Preparing Files for Building Environment

From the `colinux` directory on the supplied CD, prepare the following files and decompress them into the coLinux install folder (`c:\colinux`).

- `root_fs.lzh` (root file system)
- `swap_device_256M.lzh` (swap file system)
- `home_fs_2G.lzh` (/home-mounted file system)
- `default.colinux.xml.lzh` (device data settings file

Note: The value in the file names `swap_device_…`, `home_fs_…` represents the file size after decompression. Other sizes are also available. Choose and decompress the file size deemed appropriate.

Note: Dependent on the decompression software, decompression can fail. LHA Utility 32, Ver1.46 (http://www.lhut32.com/index.shtml) has been verified to work successfully.

13.1.3. Running coLinux

1) Open a DOS prompt and go to the install folder (`c:\colinx`).
2) Enter the command string `colinux-daemon.exe -c default.colinux.xml`.
3) A `colinux login:` prompt is displayed following the startup log. Login as "root".

13.1.4. Network Settings

coLinux has a separate IP address to Windows and as it accesses the network via Windows some configuration within Windows is required.

Several methods are available including router connection and bridge connection. The following explains how to setup a bridge connection.

(Windows XP)

1) From the control panel, open "Network Connections".
2) Right-click an externally connected network and open "Properties".
3) Open the "Advanced" tab and enable Internet connection sharing.

(Windows2000)
1) From Control Panel, open "Network and Dial-up Connections".
2) Right-click an externally connected network and open "Properties".
3) Open the "Sharing" tab and enable the Internet connection sharing".

Then, execute the following command to enable the network settings in coLinux.

Example 13-1 Network Setting Command

```
colinux:~# /etc/init.d/networking restart
Reconfiguring network interfaces: done.
colinux:~#
```

For Router Connections, as the network address of 192.168.0.0/24 is automatically used, if the network address of external connection is the same 192.168.0.0/24, the connection will fail. In this case please change the network address of the external connection.

If the network address of the external connection cannot be changed, refer to 13.1.8 "Windows Network Setup Under Special Circumstances".

13.1.5. Creating a coLinux User

In the coLinux screen enter the command as shown in Example 13-2 to create a user. Choose an appropriate password.

Example 13-2 Adding User "somebody"

```
colinux:~# adduser somebody
Adding user somebody...
Adding new group somebody (1000).
Adding new user somebody (1000) with group somebody.
Creating home directory /home/somebody.
Copying files from /etc/skel
Enter new UNIX password:
```

13.1.6. File Sharing between Windows and coLinux

This is a method for exchanging files between coLinux and Windows uses Windows' shared folder. In the coLinux screen execute a `smbmount` command as shown in Example 13-3 and enter the password for the shared folder.

Example 13-3 Windows IP Address: 192.168.0.100, Shared Folder Name: "shared"

```
colinux:~# mkdir /mnt/smb
colinux:~# smbmount //192.168.0.100/shared /mnt/smb
212: session request to 192.168.0.100 failed (Called name not present)
212: session request to 192 failed (Called name not present)
Password:
```

If the user name is different from that on the Windows side, specify it with a command option. For more information, execute "`man smbmount`" and refer the help.

After this, data in the windows' shared folder "shared" and that in the coLinux directory `/mnt/smb` will be the same.

13.1.7. Introducing a Cross Development Environment

Build a cross-development environment in coLinux by following Chapter 7, "Preparing Development Environment."

All files necessary for the building of the environment can be accessed from coLinux through the shared folder as described above.

Armadillo-J development can now be carried out from Windows. The following sections provide instructions for special cases.

13.1.8. Windows Network Setup Under Special Circumstances

These network settings are to be used when the external network connection as an address of 192.168.0.0/24.

(Windows XP)

"Bridge Connection"

1) From the control panel, open "Network Connections".
2) Select both the external network connection and the network connection with the device name "TAP-Win32 adapter".
3) From the "Advanced" menu, select "Bridge Connection".

(Windows2000)

In this method, a network address other than 192.168.0.0/24 is used for the private network in Windows2000. Here, the network address of 192.168.1.0/24 is used.

1) From the control panel, open "Network and Dialup Connections".
2) Right-click the externally connected network and choose to disable it.
3) Right-click the externally connected network to open "Properties".
4) On the "General" tab, select "Internet Protocol (TCP/IP)" and press the "Properties" button.
5) Select "Use the following IP address" and enter "192.168.100.100".
6) Open the "Share" tab and enable Internet connection sharing.
7) Right-click the network connection with device name "TAP-Win32 adapter" and open "Properties".
8) Select "Internet Protocol (TCP/IP) on the "General" tab and press the "Properties" button.
9) Select "Use the following IP address" and enter 192.168.1.1.
10) Right-click the externally connected network and open "Properties".
11) Select "Internet Protocol (TCP/IP) on the "General" tab and press the "Properties" button.
12) Return the IP address settings to their original state.
13) Right-click the externally connected network and choose to enable it.

13.1.9. coLinux Network Configuration

After coLinux is installed, DHCP is used by default. In an environment where a DHCP server is not running, a fixed IP address must be set.

Network settings can be viewed using the `ifconfig` command.

Example 13-4 Executing a ifconfig command

```
colinux:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:43:4F:4E:45:30
          inet addr:192.168.0.151  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:189 errors:0 dropped:0 overruns:0 frame:0
          TX packets:115 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:24472 (23.8 KiB)  TX bytes:9776 (9.5 KiB)
          Interrupt:2

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

colinux:~#
```

If the IP address of the eth0 device is not displayed, a fixed IP address must be set. The IP address should be set to match the "TAP-Win32 adapter" network connection for "Router Connections" or to match the external network connection for "Bridge Connections".

The following describes how to the change settings to those shown in Table 13-1.

Table 13-1 Network Settings

Parameter	Setting
IP Address	192.168.1.100
Net Mask	255.255.255.0
Gateway	192.168.1.1
DNS Server	192.168.1.1

1) Edit `/etc/network/interfaces` in coLinux as shown in Example 13-5.

Example 13-5 Example of Editing a /etc/network/interfaces File

```
auto lo eth0
iface lo inetloopback
iface eth0 inet static
  address 192.168.1.100
  gateway 192.168.1.1
  netmask 255.255.255.0
```

2) Edit the `/etc/resolv.conf` in coLinux as shown in Example 13-6.

Example 13-6 Example of Editing a /etc/resole.conf File

```
nameserver 192.168.1.1
```

3) Execute the following command to update the network settings as edited.

Example 13-7 A Command to Reconfigure Network Settings

```
colinux:~# /etc/init.d/networking restart
Reconfiguring network interfaces: done.
colinux:~#
```

13.2. About The Base Image

13.2.1. Memory Map of The Base Image

Flash memory devices can be checked in the following section of the boot log.

```
Initializing Armadillo-J MTD mappings
flash memory device = M29W160EB detected
 Amd/Fujitsu Extended Query Table v1.0 at 0x0040
number of CFI chips: 1
cfi_cmdset_0002: Disabling fast programming due to code brokenness.
```

Table 13-2 Memory Map (Flash Memory Devices: AM29LV160DB, MBM29LV160BE)

Address	Contents of Flash Memory	Memory Size	Description
0x02000000 0x0201ffff	Reserved (Rewriting not allowed)	128kB	
0x02020000 0x0203ffff	Bootloader (hermit)	128kB	
0x02040000 0x021effff	kernel userland	Approx. 1.7MB	`base.img` Image
0x021f0000 0x021fffff	`/etc/config` (Rewritable)	64kB	

Note: Only the kernel and userland are copied to RAM before uCLinux boots.

Table 13-3 Memory Map (Flash Memory Device: M29W160EB)

Address	Contents of Flash Memory	Memory Size	Description
0x02000000 0x0200ffff	Board Parameter （Rewriting not allowed）	64kB	
0x02010000	kernel	Approx.1.9MB	base.img Image
	userland		
0x021effff			
0x021f0000 0x021fffff	/etc/config (Rewritable)	64kB	

Note: Only the kernel and userland are copied to RAM before uCLinux boots.

Table 13-4 Memory Map (RAM)

Address	Contents of RAM	File System	Description
0x08000000	kernel		base.img Image Copied from flash Memory before startup of uCLinux
	Userland	romfs	
	/var	ramfs	
	/etc/config	ramfs	
	/home/guest/pub	ramfs	

13.2.2. Command List

Table 13-5 Command List

Command Name	Description
addgroup	Add a group.
adduser	Add a user.
Cat	Output files continuously.
Chgrp	Change group ownership of a file.
chmod	Change access right of a file.
chown	Change ownership and group of a file.
Cp	Copy a file.
delgroup	Delete a group.
deluser	Delete a user.
echo	Display one line of text.
false	Return a "1" representing a termination status of "failure" where nothing occurs after execution
flatfsd	flat file system daemon
ftpd	ftp daemon
inetd	Inet daemon
kill	Send a signal to a process.
ls	Display a directory list.
mkdir	Create a directory.
more	Filter for viewing a file
mount	Mount file system
netflash	Update the onboard flash image via network
passwd	Change a password
ping	Send a ICMP ECHO_REQUEST packet to a network host
Ps	Display state of processes
route	Display and set an IP routing table
Sh	Shell
Su	Obtain super user rights
sulogin	Login in single mode
telnetd	telnet daemon
test	Check a file format and compare values
thttpd	Daemon to provide Web server functionality
tinylogin	Tool suit for login and user management
true	Return a "0" representing a termination status of "success" where nothing occurs after execution
umount	Unmount a file system
watchdog	Watchdog daemon
vi	Text editor

13.2.3. Startup Daemon List

Table 13-6 Startup Daemon List

Start Daemon	Description
flatfsd	Saves data to the flash memory
inetd	Provide various network service interfaces (telnet, ftp, etc.)
thttpd	Web server functionality

13.3. Original Device Driver Specifications

13.3.1. Parallel Port Driver

Parameters of device nodes corresponding to a parallel port (CON2) are shown in Table 13-6.

Table 13-7 Parallel Port List

Type	Major No.	Minor No.	Node Name (/dev/xxx)	Device Name
Character Device	210	0	padr0	Port A Data Register CH0 (Pin.7)
		1	padr1	Port A Data Register CH1 (Pin.5)
		2	padr2	Port A Data Register CH2 (Pin.6)
		5	padr5	Port A Data Register CH5 (Pin.3)
		6	padr6	Port A Data Register CH6 (Pin.4)
		8	padr	Port A Data Register All CH(8bit)
		16	paddr0	Port A Data Direction Register CH0 (Pin.7)
		17	paddr1	Port A Data Direction Register CH1 (Pin.5)
		18	paddr2	Port A Data Direction Register CH2 (Pin.6)
		21	paddr5	Port A Data Direction Register CH5 (Pin.3)
		22	paddr6	Port A Data Direction Register CH6 (Pin.4)

- Data Type

padr 0,1,2,5,6 (each CH): unsigned char (8bit) 0x00 / 0x01
padr (All CH): unsigned char (8bit) 0x00～0xff
paddr 0,1,2,5,6 (each CH): unsigned char (8bit) 0x00 / 0x01 / 0x02

The mode of each parallel port pin can be set with paddr (0: input / 1: output / 2: serial) and data writing/reading with padr.

padr 0,1,2,5,6 and paddr 0,1,2,5,6 can read and write to each CH and padr can read and write to all CH (8bit) simultaneously. CH0 corresponds to the lowest-order bit, CH7 to the highest-order bit, and padr (all-CH) to all bits. (As CH3, 4 and 7 are fixed serial mode, their values will not be affected by writing).

Example 13-8 Sample Program of Parallel Port Operation

```
#include <fcntl.h>
#include <stdio.h>

int main (void)
{
	int fd_ddr, fd_dr;
	unsigned char val;

	// Sets CH0 Direction to write only and open
	fd_ddr = open ("/dev/paddr0", O_WRONLY);
	if (fd_ddr < 0) {
		fprintf (stderr, "Open error.\n");
		return -1;
	}
	// Sets CH0 to read/write and open
	fd_dr = open ("/dev/padr0", O_RDWR);
	if (fd_dr < 0) {
		fprintf (stderr, "Open error.\n");
		return -1;
	}

	val = 1;
	write (fd_ddr, &val, sizeof(unsigned char)); //CH0 to output
	val = 1;
	write (fd_dr, &val, sizeof(unsigned char)); // Outputs High to CH0

	val = 0;
	write (fd_ddr, &val, sizeof(unsigned char)); // CH0 to input
	read (fd_dr, &val, sizeof(unsigned char)); // CH0 read to val
	printf ("padr0: %d\n", val); // Display val

	close (fd_ddr);
	close (fd_dr);

	return 0;
}
```

13.3.2. LED Control Driver

The parameters of the device node for controlling the Status LED (D4) are as follows.

Table 13-8 LED Device Node Parameters

Type	Major No.	Minor No.	Node Name (/dev/xxx)
Character Device	240	0	ajled

The LED is controlled by opening the device node and writing the following data.

Table 13-9 Written Data and Corresponding State

Data	Size	State
"OFF"	3 Bytes	Off
"ON"	2 Bytes	On

Also, the current state of the LED can be acquired by reading the device node.
The relationship between the acquired value and the LED state is as follows.

Table 13-10 Read Data and Corresponding State

Data	Size	State
"OFF"	3 Bytes	Off
"ON"	2 Bytes	On

Sample source code for the control of the LED is shown below.

Example 13-9 Sample Program LED control

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

int main( void )
{
      int fd = 0;
      char status[4]={0,0,0,0};

      if( ( fd = open( "/dev/ajled", O_RDWR ) ) < 0 ){
            fprintf( stderr, "Open error \n" );
            return -1;
      }

      /* Light LED */
      write( fd, "ON", 2 );

      /* Acquire current state of LED */
      read( fd, status, 4 );
      printf( "current status : %s\n", status );

      sleep( 1 );

      /*Turn LED off */
      write( fd, "OFF", 3 );

      /* Acquire current state of LED */
      read(fd, status, 4 );
      printf( "current status : %s\n", status );

      close( fd );
      return 0;
}
```

13.4.URL List

Figure 13-1 URL List

URL	Summary
http://armadillo.atmark-techno.com/	Armadillo Official Site
http://www.digi.com/	Digi Official Site
http://www.uclinux.org/	uClinux Official Site
http://www.uclibc.org/	uClibc Official Site
http://www.colinux.org/	coLinux Official Site
http://www.busybox.net/	BusyBox Official Site
http://tinylogin.busybox.net/	TinyLogin Official Site
http://www.net-snmp.org/	ucd-snmp Official Site
http://hp.vector.co.jp/authors/VA002416/	TeraTerm Official Site
http://alioth.debian.org/projects/minicom/	Minicom Official Site
http://www.beyondlogic.org/	Description of binary-flat
http://www.lhut32.com/index.shtml	LHA Utility 32

Revision History

Ver.	Date	Description
1.00	2003.12.24	• Initial release
1.01	2004.1.6	• Correction to a description in "6.5. Network Settings"
1.02	2004.1.14	• Additional descriptions of "6.2.2. Features of /etc/config Directory" and restoration procedure after damage to Config area. • "6.7. ftp Login", login user was changed to "guest only". • Additional description of recommended environment/distribution to "7.1. Building a Development Environment". • Addition of "8.2. To return "Image" to the default state (Hermit for WIN32)". • Additional description of "A.3. Changing Stack Size". • Additional command descriptions to "Table 13-5 Command List (addgroup, adduser, delgroup, deluser, passwd, su, sulogin). • Deletion of commands ((fsck.minix, free, ln, mkfs.minix) from "Table 13-5 Command List • Correction to a description regarding data updating to the flash memory in "6.2.2. Features of /etc/config Directory", "6.4. Termination Method" and "6.5.2. Using DHCP" • Correction to a typographical error
1.03	2004.1.22	• Correction to a typographical error
1.04	2004.2.26	• Correction to a description in "5.2.2. Downloading Rewriting Image". • Addition of "6.9. watchdog timer function" and "6.10. SNTP function". • Addition of "7.2. Building a development environment on Windows" • Addition of a command (msntp) to "Table 13-5 Command List • Addition of "Digi" and "cygwin Official site" to "Appendix.C URL List". • Correction to a typographical error
1.05	2004.3.9	• Addition of a "Windows method" to Section 5.2.2 "Downloading a Rewriting Image".
1.06	2004.4.14	• Correction to revision history, linking to the appropriate text. • Addition of " Parallel Port Driver"
2.00	2004.6.10	• Full-fledged revision
2.01	2004.9.3	• Revision addressing the exclusion of the msntp application • Correction of a typographical error.
2.02	2004.10.5	• Addition of "LED Control Driver".
2.03	2004.12.29	• Company address updated
2.04	2005.3.4	• Revisions made to accompany changes to the memory map

Atmark Techno, Inc.
AFT Building 6F, North 5 East 2, Chuo-ku, Sapporo, Hokkaido 060-0035
TEL: 011-207-6550 FAX: 011-207-6570