LINUX-26-BSP

Kontron Embedded Linux 2.6 User Manual

Manual ID: 29331, Rev. Index 07

May 2007

The product described in this manual is in compliance with all applied CE standards.

1Copyright

Copyright © 2005-2007 Kontron modular Computers.

Kontron Modular Computers makes no representations or warranties with respect to the contents or use of this manual, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose.

Kontron Modular Computers makes no representations or warranties with respect to this embedded Linux package, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose.

Permission is granted to make and distribute verbatim copies of this manual provided that the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this documentation under the conditions for verbatim copying, provided also that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this documentation into another language, under the above conditions for modified versions.

Intel and Pentium are a trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

SuSE is a registered trademark of SuSE GmbH

RedHat is a registered trademark of RedHat

All other trademarks, registered trademarks, and trade names are the property of their respective owners.

2Revision History

Manual/Product Title:	**LINUX-26-BSP-PPC**	
Manual ID Number:	**29331**	
Revision Index	**Brief Description of Changes**	**Date of Issue**
01	Initial Issue	March 2005
02	Added chapter about cross development	April 2005
03	Some additional clarifications / improvements	May 2005
04	Added subchapter about compiler options and possible influence on performance Added chapter describing the GPL root file system (fits to filesystem.0102 or newer)	June 2005
05	Added chapter about User Project creating, development, etc., UDEV description chapter, Xscale specific appendix, grammar and style corrections. Included PDR 06025237: solution for problem with shared libraries symbols when cross debugging.	May 2006
07	Added information valid for BSP EB8347	May 2007

© 2007 Kontron Modular Computers

3Contents

1 Copyright....2
2 Revision History....3
3 Contents....4
4 Introduction....6
5 Components of the Kontron Embedded Linux....6
6 Legend....6
7 System Requirements....7
7.1 Development Host....7
7.1.1 Hardware....7
7.1.2 Software....7
7.2 Target....7
7.2.1 Hardware....7
7.2.2 Software....7
8 Preparing the Host Development Environment....8
8.1 Embedded Linux BSP Installation....8
8.1.1 Contents of the Kontron Linux CD-ROM....8
8.1.2 BSP Package Installation....9
9 Preparing the Development Target....11
9.1 NetBootLoader....11
9.1.1 NetBootLoader Documentation....12
9.1.2 Prerequisites for Using Network Capabilities....12
10 My First Linux Project....14
10.1 Project Initialisation....14
10.2 Setting the development environment....15
10.3 Linux-Kernel build....15
10.4 Compile the BSP components....17
10.5 Install the BSP components....17
10.6 How to build own applications using the BSP....18
10.7 How to modify files provided in the BSP....18
11 The Linux Kernel....19
11.1 Kernel Configuration....19
11.1.1 Processor Setup for the Kontron Target CPUs....19
11.1.2 Kernel Command Line....19
11.1.3 Compiling Kernel....22
11.2 Network Download....23
11.3 Booting....25
12 Root File System Strategies....26
12.1 Onboard FLASH (INITRD/RAMFS)....26
12.2 CompactFlash, IBM Microdrive™....26
12.3 IDE-HDD....27
12.4 SCSI Drives....27
12.5 M-Systems Disk-On-Chip....27
12.6 Remote File System on an NFS Server....27
13 Setting Up a Diskless Client....28
13.1 Kernel Configuration and Command Line....28
13.2 NFS Server Setup....28
13.3 Setting Up the Root File System....28
13.4 Booting from NFS Root File System....29
14 The Initial Ramdisk INITRD....30
14.1 Introduction....30
14.2 How to Create or Modify an INITRD....30
14.2.1 Creating a New INITRD Image....30

14.2.2 Inspecting and Modifying an Existing INITRD Image....32
14.3 Adding an INITRD to the Kernel....33
14.3.1 Kernel Configuration....33
14.3.2 Kernel Command Line....33
14.3.3 Building a Kernel image including INITRD....33
14.4 Downloading....33
14.5 Booting....34
15 INITRAMFS....35
15.1.1 Kernel Configuration....35
15.1.2 Kernel Command Line....35
15.1.3 Building a Kernel image with root filesystem in RAMFS....35
15.2 Downloading and Booting....35
16 Root File System on a Mass Storage Device (IDE/SCSI/CompactFlash)....36
16.1 Required Kernel Configuration for CompactFlash....36
16.2 Required Kernel Configuration for IDE....36
16.3 PMC260 or CP360 Required Kernel Configuration....36
16.4 The Chicken/Egg Problem....37
16.4.1 Preparation of the Root File System on the Development Host....37
16.4.2 Preparation of the Root File System on the Target....37
17 Root File System on an M-Systems Disk-On-Chip....38
17.1 General....38
17.2 Kernel Configuration....38
17.3 Creating Partitions....38
17.4 Preparation of the Root File System on the Disc-On-Chip....39
18 Cross Development....39
18.1 Cross Compilation....39
18.2 Debugging Tools gdb and gdbserver....41
18.3 A Sample Debugging Session....41
18.4 The .gdbinit File....44
18.5 Graphical Debugging....45
18.6 Execution Performance....48
18.7 PowerPC e500 Specific Optimization....48
19 Linux GPL Root File System....50
19.1 Application List and Command Reference....50
19.2 Directory structure....50
19.3 Linux System Configuration Files....51
19.4 Device files....51
20 Additional Libraries in directory addl_libs....53
21 The LM-Sensors package (optional)....54
22 Appendix A: Notes on RPM....55
23 Appendix B: Configuring Minicom....57
24 Appendix C: XScale IXP42x based eBrain specific considerations....58
24.1 Using built in processor Ethernet Interfaces....58
24.1.1 Downloading the software....58
24.1.2 Compiling the software modules....59
24.1.3 Using the created modules....59
24.2 ARM XScale Linux Kernel....61
24.2.1 Kernel compiling....61
24.2.2 Kernel Command Line....61
24.2.3 INITRD with ARM Linux Kernel....62
24.3 NFS mounted Root file system....62

 © 2007 Kontron Modular Computers

4Introduction

This manual does not describe the Linux operating system. Its usage requires some Linux experience with:

- basics of the Linux operating system
- GNU C compiler
- installing software on a Linux system
- system administration
- Linux kernel configuration and compilation

5Components of the Kontron Embedded Linux

The Kontron Linux CD-ROM contains the following components for each particular supported hardware:

- a customized Linux kernel
- a GNU cross-compiler tool chain
- a Linux root file system with the most important Linux tools
- drivers, function libraries, tools, demo programs for the Kontron specific onboard components such as E^{2}PROMs, LEDs, Watchdog, etc.

The purpose of this package is to provide all the basic functionality to write Linux applications for Kontron boards.

It is not required to purchase an embedded Linux distribution for this target architecture in order to write applications.

6Legend

Throughout this manual the following typographical conventions and abbreviations for special elements are used.

Abbreviations:

Term	Abbreviation
The user name	usr
user prompt at the development host (in directory dir)	usr@x86host:/dir >
The root prompt	root@x86host:/dir #
The user prompt in the target system	usr@target: #
The root prompt in the target system	root@target: #

Special Fonts:

Term	Example
User entries or program outputs visible on a command shell	`root@x86host:/home/usr # YaST`
Name of a file or directory	***lin-bsp-xxx***
Placeholder for a variable value	<version>
Placeholder for the system architecture (will be “ppc”, “arm”, “x86” in the real world)	<architecture>

7 System Requirements

7.1 Development Host

7.1.1 Hardware

The following documentation assumes a development host which matches the following specifications:

- Intel® Pentium® III processor 800 MHz or higher
- minimum RAM: 128MB
- 500 MB available disk space
- network connection via Ethernet

7.1.2 Software

A desktop Linux must be installed on the development host. The installation should contain Linux development, Network server components (ftp, telnet, nfs) and a serial terminal program (e.g. minicom).

We recommend SuSE 9.0 or newer.

Known limitations or specific handling instructions for particular distributions are documented within the ***doc*** directory on the Kontron Linux CD-ROM.

7.2 Target

7.2.1 Hardware

Kontron embedded Linux BSPs are available for a wide variety of boards, e.g.

PowerPC:

- E²Brain: EB8245, EB8540, EB8541, EB860, EB8347
- CompactPCI: CP320, CP321, CP322, CP620
- VME: VMP1, VMP2, VMP3

Arm/Xscale:

- E²Brain: EB425, EB420

7.2.2 Software

The Kontron embedded Linux BSP contains all software components required for the target.

© 2007 Kontron Modular Computers

8 Preparing the Host Development Environment

8.1 Embedded Linux BSP Installation

8.1.1 Contents of the Kontron Linux CD-ROM

The Kontron Linux CD-ROM which is provided together with system components is always structured following the various hardware architectures supported, e.g.:

- **noarch**
  packages which are not architecture dependent.
- **ppc_40x**
  packages which are specific for PowerPC 40x architectures, e.g. Kontron EB405
- **ppc_6xx**
  packages which are specific for PowerPC 6xx architectures, e.g. Kontron EB8245, VMP1/2, CP320/1
- **ppc_e500**
  packages which are specific for PowerPC e500 architectures, e.g. Kontron EB8540, EB8541, VMP3
- **ppc_e300**
  packages which are specific for PowerPC e300 architectures, e.g. Kontron EB8347
- **i386 / i686**
  packages which are specific for Intel architectures

Each directory contains sub directories in the following format:

- **bsp-<boardtype>-[kernelver].<index>**
  board support package containing tools, libraries and sample applications.
- **filesystem.<index>**
  Linux root file system
- **toolchain.<index>**
  gcc cross-toolchain
- **[kernelver]lin-drv-<boardtype>.<index>**
  driver packages for peripharal boards
  (contains own documentation which is not described in this manual)

Legend:

- **[kernelver]**

  optional version of the Linux kernel contained or supported e.g.

  - 2.6.13 (contains Kernel 2.6.13)
  - lin26 supports Kernel 2.6.xx
  - lin independent of a particular kernel version
- **<index>**
  Kontron internal release number, e.g. 0101

 © 2007 Kontron Modular Computers

- **<boardtype>**
  board name, e.g. eb8540, cp620, vmp1

In addition to the Driver- and BSP-related directories there are the following directories on the CDROM

- **doc**
  This directory contains additional documentation, e.g. notes on particular distributions.

- **GPLSOURCES**
  This directory contains sources of packages, which are contained in the BSP's or drivers in binary format.

The contents of ***GPLSOURCES*** are not required for Linux development as described in this manual. These sources are provided under the GPL without any warranty or support.

8.1.2 BSP Package Installation

Three packages from the Kontron CD-ROM must be installed on the development host:

- the BSP component: **bsp-<boardtype>-<kernelver>.<index>**
- the root file system: **filesystem.<index>**
- the cross toolchain: **toolchain.<index>**

All components can be found inside the directory for the particular architecture family (as described above). All packages are delivered in RPM format and additionally as a tarball.

a) RPM Package Installation

The RPM package is installed as root user with the following command:

```
usr@x86host:~ > su
Password: xxxxx
root@x86host: # rpm -ivh <filename of rpm package>
```

e.g.

```
root@x86host: # rpm -ihv bsp-cp620-kom-2.6.10-0100.rpm
root@x86host: # rpm -ihv crosstool-powerpc-6xx-linux-gnu.noarch.rpm
root@x86host: # rpm -ihv rootfs-ppc-6xx-kom-0102.noarch.rpm
```

The RPM package format provides the advantage that all of the installed packages are managed in a database. So it's always possible to keep track of all installed packages.

To view all the Kontron packages installed on the system, enter:

```
root@x86host: # rpm -qa | grep kom
bsp-eb8540-kom-2.6.10-0100
bsp-eb405-kom-2.6.10-0100
```

Information about individual packages may be obtained via, ***rpm –qi***, e.g.:

```
root@x86host: # rpm -qi bsp-eb8540-kom
```

To determine where the package files were installed, enter:

```
root@x86host: # rpm -ql bsp-eb8540-kom
```

This will call up a list of all files belonging to this package.

Information for Advanced Users

To learn more about the advantages of the RPM package format, refer to “Appendix A: Notes on RPM”

After the Kontron RPM packages are installed on the development host the following directories exist in the ***/opt*** directory.

```
/opt/
|-- crosstool
|   `-- <architecture>-linux-gnu/…                        GNU cross-compiler
                                                           toolchain
`-- [kernelver]_bsp_kom                                    The BSP
    |-- bsp_[boardname]
    |   |-- 3rd_party_drivers                              Optional - other parties
                                                           software
    |   |-- lm_sensors                                     Optional – the Lm-sensors
                                                           source package
    |   |-- linux                                          Linux kernel source
    |   |-- scripts                                        useful scripts
    |   |-- apps                                           target applications
    |   |-- inc                                            target header files
    |   |-- lib                                            API library
    |   |-- addl_libs                                      Additional libraries
                                                           (e.g. sysfsutils, pciutils)
    |       `-- src                                        sources (all sources are
                                                           provided)
    `-- rootfs_<architecture>_[subarch].[version]          the linux root file system
        |-- docs                                           command reference
        `--                                                binary image of the root file
root_filesystem_<architecture>_[subarch]_linux.tar.bz2    system
```

b)Tarball Installation

The RPM installation is the preferred method. Tarballs are provided for Linux distributions not supporting RPM.

Tarballs are extracted with tar commands:

- ***tar –xvjf <filename>*** for archives with the extension ***tar.bz2*** or
- ***tar –xvzf <filename>*** for archives with the extension ***tar.gz*** or ***tgz***

Refer to the Linux man pages for further information on how to extract tarballs.

In principle, tarballs may be extracted to any location. But the provided scripts and Makefiles will only work if the tars are extracted to the same locations as described above in paragraph RPM installation.

 © 2007 Kontron Modular Computers

9Preparing the Development Target

Refer to the hardware manual for the Kontron CPU board for guidance on how to set it up for booting. In particular, the chapter about the NetBootLoader provides important information regarding the downloading and starting of the Linux kernel.

Kontron boards are typically equipped with the NetBootLoader, a boot loader which has integrated TCP/IP support for downloading files and remote login via telnet. On some boards, especially custom specific designs, there may be a different boot loader, e.g. U-Boot. This manual is only applicable for NetBootLoader equipped CPUs.

For the initial communication with the target, a serial interface must be established between the development host and the serial term connection of the Kontron PowerPC board using a null modem cable.

The next step is to set up a serial terminal program on the development host. The minicom program is used throughout this manual. To set up minicom, refer to "Appendix B: Configuring Minicom" or refer to the manual for minicom.

This serial communication will represent the target's system console for the NetBootLoader and the Linux kernel.

NetBootLoader and Linux kernel are also capable of handling consoles logged in via telnet, but, for the initial steps, this serial connection is mandatory.

To verify that the serial console works, enter the boot loader command mode.

Now start the program minicom (if not already running).

```
> minicom
```

9.1NetBootLoader

Press the RESET button on the front panel. The following prompt will appear.

```
+------------------------------------------------------------------+
|  NetBootLoader: Kontron BootstrapLoader with Network support      |
+------------------------------------------------------------------+
|  Press the Abort button, type 'abort' or login over telnet to     |
|  cancel a running Boot Wait Time (LED blinks)                     |
|  Type 'help' or '?' to get online help                            |
+------------------------------------------------------------------+
|  This Product uses the Real Time Operating System ECOS from RedHat |
|  http://www.redhat.com                                            |
+------------------------------------------------------------------+

NetBtLd>
```

After a certain time (typically 5 seconds) the NetBootLoader will boot the installed operating system kernel if a valid start key of a kernel image is found in the onboard flash.

 © 2007 Kontron Modular Computers

To cause the system to remain at the NetBootLoader command prompt, enter:

```
NetBtLd> abort
```

Alternatively, press the abort button on the front panel.

FDT (**F**lattened **D**evice **T**ree)

BSPs based on the *powerpc* architecture specification of the Linux kernel require a boot loader capable of passing a flattened device tree to the Linux kernel. This is the case for all Kontron PowerPC BSPs designed for Kernel 2.6.17 or newer.

If a Linux kernel is designed for FDT, is obvious by looking into the ARCH specification. All Kernels, which have ARCH=powerpc, are using FDT.

On the Kontron NetBootLoader use the command "fdt 1" to switch on FDT support.

```
NetBtLd> fdt 1

Enabling creation of flattened device tree. Boot image from address 0.
```

Command “fdt_show” print actually generated **F**lattened **D**evice **T**ree.

9.1.1NetBootLoader Documentation

Refer to the board's hardware manual documentation for details.

A summary of how to use the NetBootLoader for downloading a Linux kernel to the Kontron target is provided here.

Hint: The command “*help*” displays a help menu for all implemented commands.

```
NetBtLd> help
```

9.1.2Prerequisites for Using Network Capabilities

The following actions must be completed prior to using the network.

- a network cable (10 / 100 Mbit/s) must be attached to the PowerPC’s primary Ethernet port.
- an FTP server must be running on the development host (see the appropriate documentation for inetd.conf, ftpd, in.ftpd for details on how to install and configure an ftp server)
- the network parameters must be assigned to the target.

  This is done with the command “*net*”:

  ```
  NetBtLd> help net
  Syntax:

    Display/change network settings:
      net [<ip_addr> | -netmask <netmask> | -gw <gateway> | -f]
        <ip_addr>, <netmask>, <gateway>: aa.bb.cc.dd
        -f force CRC update
  ```

If the IP address, netmask or gateway is not known, request the system administrator to provide them. This needs to be done only once for the target. These parameters are stored in a serial flash EPROM. Once these parameters have been changed the system must be restarted.

Pinging the target from the development host is a way to check the correctness of the IP connection and parameters. In addition, a telnet connection from the development host to the target should now be possible. The procedure how to download a kernel via FTP is described in detail later in this manual.

A user must be created on the development host that is allowed to login via ftp from the Kontron target. For the following chapters it assumed, that a user with the name “targetuser” and the password “targetpasswd” have been created. The home directory of this user will be /home/targetuser.

To be able to store kernel images for the target, all users should have write access to /home/targetuser:

```
usr@x86host:~ > cd /home
usr@x86host:~ > su
Password:
root@x86host: # chmod 777 targetuser/
root@x86host: #
```


© 2007 Kontron Modular Computers

10 My First Linux Project

10.1 Project Initialisation

If the BSP was installed using the RPM-package (recommended), it is now installed somewhere under the ***/opt*** directory (chapter 8.1.2 BSP Package Installation). All installed files are owned by root. But for the daily development work, it is highly recommended to work under a standard user account. For this reason a local project directory has to be created inside the user's home area.

This is done by invoking the script ***initproject.sh*** (provided within the BSP installation directory) with the desired working directory as a parameter.

```
usr@x86host: > /opt/<linux
version>_bsp_dir/bsp_<boardname>/scripts/initproject.sh <work_dir>
```

example:

```
usr@x86host: > cd ~
usr@x86host: > mkdir bspwork
usr@x86host: > cd bspwork
usr@x86host: ~/bspwork >
/opt/linux26_bsp_dir/bsp_eb42x/scripts/initproject.sh eb42x_project

INIT Kontron BSP Project
-----------------------------------------------------------------

BSP installation directory = /opt/linux26_bsp_kom/bsp_eb42x
Project directory = /home/usr/bspwork/eb42x_project
initproject finished !!!
usr@x86host: ~/bspwork>
```

This script generates the following entries under the directory <***work_dir***>

- ***documentation*** - the online documentation in HTML
- ***scripts*** - some useful scripts (described later)
- ***modules*** – source code and build environment for all Kernel modules provided within the BSP
- ***lib*** - source code and build environment for the board library provided within the BSP
- ***apps*** - source code and build environment for all sample applications provided within the BSP
- ***inc*** - header files
- ***Makefile*** - the main ***Makefile*** (steps down into ***modules***, ***lib***, ***apps*** and calls the subordinate ***Makefiles*** there)
- ***addl_libs*** – additional libraries and headers, which are required for compiling particular applications (containing pciutils and sysfsutils)
- ***3rd_party_drivers*** – (optional) directory containing other parties software
- ***lm_sensors<-version>*** – (optional) directory containing the Lm_sensors source package for HW-monitoring

The created directory structure is based on symbolic links to the original files within the installed BSP. This has the advantage, that creating such a project environment requires a very small amount of disk space, and as a 2nd effect, that the BSP-provided files can not be unintentionally modified.

 © 2007 Kontron Modular Computers

10.2 Setting the development environment

Besides the directory structure, ***initproject.sh*** will generate a script called ***environment.sh***, which sets up all required environment variables for the build process. This means, that it configures the Kernel source location, and the compiler environment.

The ***environment.sh*** script must be sourced before compiling any component of the BSP. Additionally, this environment script can be used for setting up the environment within own application projects.

So, use the following command as the next step:

```
usr@x86host: > cd <work_dir>
usr@x86host: > source scripts/environment.sh
```

example:

```
usr@x86host: > cd ~/bspwork/eb42x_project/
usr@x86host: ~/bspwork/eb42x_project/ > source scripts/environment.sh
Setting KOMKERNELDIR to /home/usr/bspwork/eb42x_project/linux
Setting KOMINSTALLDIR to /home/usr/bspwork/eb42x_project/boot
Kernel Version = 2.6.13-rc6
usr@x86host: ~/bspwork/eb42x_project/>
```

After this the system is prepared for compilation.

10.3 Linux-Kernel build

The Kontron embedded Linux contains a Linux kernel 2.6.x adapted to work with the particular target hardware.

The Linux kernel is free software covered by the GNU General Public License.
The kernel provided by the Kontron embedded Linux has some extensions to the standard kernel to support the specific hardware features of the Kontron boards. However, all these modifications to the vanilla kernel are done in a way which conforms to the open source conventions of the Linux kernel, and all of these modifications will be made available to the open source community.

a) Set environment for cross development

Please dont forget to source the environment.sh script, as mentioned above.

```
~/bspwork/sample_project > source scripts/environment.sh
Setting KOMKERNELDIR to /home/<username>/bspwork/sample_project/linux
Setting KOMINSTALLDIR to /home/<username>/bspwork/sample_project/boot
Kernel Version = <kernel version>
```

b) Load default configuration for the board

The Linux kernel provides a default configuration for each Kontron board supported. All default configurations can be found under the directory ***arch/<architecture>/configs***.

The filename is typically **<boardtype>_defconfig**. To load a default configuration, enter:

```
usr@x86host: ~/bspwork/sample_project/linux > make <boardtype>_defconfig
```

example:

```
usr@x86host: ~/bspwork/linux > make eb8540_defconfig
```

c) Modify kernel configuration

The kernel is already configured correctly for the Kontron board supported. For most applications this default configuration will be correct. At least for the first tests it is highly recommended to build the kernel with this default configuration.

Information for Advanced Users
This kernel supports the standard kernel configuration mechanisms
make menuconfig and ***make xconfig.***
However, the various configuration options of a Linux kernel will not be described in this document.

d) Build kernel
Call the following command to build dependencies and the kernel image for Kernel 2.6:

```
~/bspwork/sample_project/linux > make
```

 © 2007 Kontron Modular Computers

10.4 Compile the BSP components

Before any build, dont forget to source the environment.sh script.

```
~/bspwork/sample_project > source scripts/environment.sh
Setting KOMKERNELDIR to /home/<username>/bspwork/sample_project/linux
Setting KOMINSTALLDIR to /home/<username>/bspwork/sample_project/boot
Kernel Version = <kernel version>
```

After this the system is prepared for compilation.

To compile all components just call in ***<work_dir>***

```
usr@x86host: > make
```

The particular components of the BSP can also be built separately, with the commands

```
usr@x86host: > make kernelmodules
```

Compile all kernel modules, provided in the BSP

```
usr@x86host: > make userlib
```

Create the board library (which is required to build the demo applications)

```
usr@x86host: > make demoapps
```

Build the demo applications

Optionally (not available in all BSP's)

```
usr@x86host: > make lm_sensors
```

Build all the user space applications, contained in the lm-sensors package.

Optionally (not available in all BSP's)

```
usr@x86host: > make thirdparty
```

Build additional 3rd-party drivers, which are provided within the BSP. This component only exists, if the board is equipped with particular HW components, for which 3rd-party drivers (not integrated in the kernel) are required. If this rule doesnt exist, all HW-components are fully supported within kernel and the Kontron board drivers.

Finally, to clean up everything, call

```
usr@x86host: > make clean
```

10.5 Install the BSP components

The building process, described in the previous chapter, generates various binary exectubles, kernel modules, and shared libraries. To execute these on the target, they have to be copied to the desired location in the target root file system. There are various ways to generate a target root file system (please refer to the according chapters below); so there is no possibility to do that fully automatically within the Makefile.

But, just by calling

```
usr@x86host: > make install
```

All generated output files are installed into the sub directory boot within the current working directory, already in the correct directory hierarchy, as you could also find in the target root fs. So it is up to the user to copy the whole tree under the directory <boot>, or only particular files, to the target root fs manually. How this can be done, is described in later chapters.

10.6 How to build own applications using the BSP

Building the demo programs provided with the BSP will give a first look&feel of the available functionality. However, the main aim of the BSP is, to provide facilities for writing own applications by using the BSP API.

This is achieved with the following files:

- **scripts/environment.sh** - this script sets up the build environment (Kernel source dir and compiler-configuration) and shall be sourced in every build environment
- **lib/bin/lib<boardname>.a** - this library contains the API as described in on-line manual, and should be always linked against user applications
- **inc** - contains all header-files, which come with the BSP

Additionally, the ***Makefile*** in the apps directory, and the demo program sources shall be considered as templates for building own applications

10.7 How to modify files provided in the BSP

After initialisation the ***<work_dir>*** contains only symbolic links to the original BSP files. Because the original files are owned by root, it is not possible to modify any of them in the project directory.

However, there is a script ***make_private.sh*** provided. It replaces the symbolic link to a BSP source file with local copy of this file. Then, it is possible to modify the file (without damaging the installed BSP).
example:

```
usr@x86host: > cd apps/src
usr@x86host: > ../../scripts/make_private.sh wd_demo.c
```

This replaces the ***wd_demo.c*** symbolic link with a local copy of the file, and now it can be modified. The original file, however, is still available in the BSP install dir, for this reason at any time a new and clean project copy of the BSP can be generated by calling ***initproject.sh*** again.

 © 2007 Kontron Modular Computers

11 The Linux Kernel

11.1 Kernel Configuration

This chapter explains how the kernel can be customized for the particular needs of an application. If this is being done for the first time, then it is better to proceed with “10.3 Linux-Kernel build” and return here only after a kernel has been compiled and downloaded to the target.

Please note also that this document only shows a few particular items of the kernel configuration which may be useful for Kontron boards. This document is not intended to describe all the configuration features of the Linux 2.6 kernel.

There are several ways to enter the kernel configuration:

```
make menuconfig   (menu driven text mode, requires Ncurses)
make xconfig      (graphical user interface, requires X and TCL/TK)
make gconfig      (graphical user interface, requires GTK-Lib)
```

Independent of the type of configuration GUI chosen, always the same configuration options will appear.

11.1.1 Processor Setup for the Kontron Target CPUs

For each board supported in a BSP, a default setting is provided. The way of how this default setting is loaded is described in the previous chapter.

This default setting already provides a correct set up for the devices onboard the target system, e.g. an E²Brain module with the appropriate carrier, or a CompactPCI or VME-Board.

This means, that the following onboard devices are already preconfigured correctly:

- CPU type and options
- console (typically serial console, optional graphics controller)
- network drivers
- serial drivers
- onboard mass storage devices, e.g. CompactFlash

However, there are kernel options which must be set depending on the application. The most important are mentioned in the following paragraphs.

11.1.2 Kernel Command Line

The Kontron NetBootLoader can pass a kernel command line to the Linux kernel by the ***cbl*** command.

Calling help on the NetBootLoader command prompt will display this message:

```
  Display all valid bootlines
    cbl
  Delete a bootline
    cbl n -
  Change/Overwrite a bootline
    cbl n <args>
    n may be c or 0..3
```

Refer to the hardware manual for a detailed description of the ***cbl*** command.

This feature is not available for bootloaders other than the Kontron NetBootLoader, e.g. the U-Boot.

The kernel concatenates the

- common command line (***cbl c*** command on NetBootLoader)
- the image specific command line (***cbl [imagenumber]*** command on NetBootLoader)

and uses the resulting string as its kernel command line

Alternatively, the kernel command line can be configured and compiled into the kernel. This will be the only solution to provide a command line, if using the U-Boot loader; or it might be desired to hardcode the commandline into the kernel for any reason.

The kernel command line is configured in the menu:

```
Platform options -->
      Default bootloader kernel arguments
```

The compiled-in command line is only active, if no command line is passed by the boot loader. The bootloader command line will always overwrite the compiled-in command-line.
For XScale based boards please refer to Appendix C: XScale IXP42x based eBrain specific considerations.

The resulting command line is always visible on the kernel startup, as shown in the following example:

```
loaded at:      00000100 00116298
relocated to:   00800000 00916198
board data at: 00914030 00914098
relocated to:   00805290 008052F8
zimage at:      00805B19 00913A95
avail ram:      00917000 08000000

Linux/PPC load: console=ttyS0,9600
ip=193.102.136.44:193.102.136.43:193.102.136.43:255.255.255.0:eb8347:eth1
root=nfs nfsroot=/nfsroot/claugi/eb8347
Uncompressing Linux...done.
Now booting the kernel
Linux version 2.6.13-rc6 (claugi@claugisuse10) (gcc version 3.4.1) #3 Tue Mar
28 15:19:09 CEST 2006
Built 1 zonelists
Kernel command line: console=ttyS0,9600
ip=193.102.136.44:193.102.136.43:193.102.136.43:255.255.255.0:eb8347:eth1
root=nfs nfsroot=/nfsroot/claugi/eb8347
```

The kernel command line should include at least the root device specification (*root=*) and console device specification (*console=*). Depending on the requirements, other options may also be included in this command line (e.g. for diskless systems the *nfsroot=* option).

 © 2007 Kontron Modular Computers

A complete list of kernel command line options can be found in the Linux kernel sources subdirectory *Documentation* in the file: **kernel-parameters.txt**

 © 2007 Kontron Modular Computers

Here is a typical example of a command line set up with the NetBootLoader ***cbl*** command:

```
NetBtLd> cbl
Bootline common:
console=ttyS0,9600

Bootline 0:
root=/dev/nfs
ip=193.102.136.40:193.102.136.43:193.102.136.1:255.255.255.0:atc82
70:eth0 nfsroot=/nfsroot/claugi/atc8270

Bootline 1:
root=/dev/hdc1

Bootline 2:
root=/dev/hda1

Bootline 3:
root=/dev/ram0
```

Comments:

console=ttyS0,9600 in common bootline:

This ensures, that for all Kernel images commonly a serial console with 9600 Bd on ttyS0 is supported

root=/dev/hdc1, root=/dev/hda1:

Configuration of a mass storage device (CompactFlash, IDE) for the root fs

root=/dev/ram0:

Root file system is INITRD

From kernel 2.6.17 the root filestem can be includet to the kernel image, in this case the “root” option can not be set.

root=/dev/nfs ip=193.102.136.40:193.102.136.43:193.102.136.1:255.255.255.0:atc82 70:eth0 nfsroot=/nfsroot/claugi/atc8270:

This is a configuration for root fs via NFS. The whole IP-configuration is done in the command line.

All of the above examples are described more in detail in the chapter 12 “Root File System Strategies”

11.1.3Compiling Kernel

Generate a pure Linux kernel image for Kontron CPU:

```
> make
```

If ***<architecture>*** is defined as "powerpc", “make” can also generate a kernel image including the root filesystem in RAMFS filesystem. Enabling and usage of the RAMFS is described later in this manual.

After a successful build process the generated file(s) can be found in arch/powerpc/boot:

arch/powerpc/boot /vmlinux.bin
arch/powerpc/boot /vmlinux.gz

 © 2007 Kontron Modular Computers

For all other architectures, to generate a Linux kernel image including an initial RAM disk invoke:

```
> make zImage.initrd
```

For now, only pure kernel images without the initial RAM disk should be created. Creating and using the INITRD is described later in this manual.

The generated file has the following properties:

- it contains the Linux kernel in a compressed form
- it contains code for unpacking and relocating the compressed kernel
- it contains code detecting whether an INITRD image is present.
  In this case, the INITRD image is unpacked and mounted as root file system.
- memory addresses are located properly for the Kontron target

The generated file(s) are found in arch/<architecture>/boot/images:

arch/<architecture>/boot/images/zImage.kom
pure kernel image

or

arch/<architecture>/boot/images/zImage.initrd.kom
kernel image including an initial ramdisk

The generated kernel image can now be downloaded to the Kontron target.

11.2 Network Download

Ensure that the system is set up in accordance with chapter “9.1.2 Prerequisites for Using Network Capabilities”.

First, the zImage.kom or zImage.initrd.kom must be copied to a location where the target can access it via FTP. After compiling the kernel, enter

```
> cp arch/<architecture>/boot/images/zImage.kom /home/targetuser
```

on the development host’s command line.

Now reset the target; when the prompt appears, immediately enter ‘abort’ or press the abort button:

```
NetBtLd> abort
```

Now an FTP connection to the development host can be opened with the command:

```
NetBtLd> login [hosts ip-address] <username> <password>
```

The response should be similar to this:

```
NetBtld> login <server IP address> targetuser targetpasswd
220 develop-host FTP server (Version 6.5/OpenBSD, linux port 0.3.2) ready.
331 Password required for targetuser.
230- Have a lot of fun...
230 User targetuser logged in.
Successfully logged in
```

This ftp client supports the commands *put, get, ls, pwd, bye*. To download a Linux kernel zImage enter the command:

```
NetBtLd> get zImage.kom
```

The response should be similar to this:

```
200 Type set to I.
200 STRU F ok.
200 MODE S ok.
200 PORT command successful.
150 Opening BINARY mode data connection for 'zImage.kom' (561924 bytes).
226 Transfer complete.
Data loaded, count: 561924
```

Now the kernel image is downloaded to the target system.

 © 2007 Kontron Modular Computers

The next step is to program it to the onboard flash with the command *lf*:

```
NetBtLd> lf
```

When the prompt appears, the system must be rebooted with:

```
NetBtLd> rs
```

This kernel image is programmed into a flash ROM. This means that it can never be lost, even without any batteries. It can only be deleted by a new download. The image is secured with a CRC so that the boot loader can verify the integrity of the flash contents at every system startup before unpacking and starting the kernel image.

For quick testing start the image directly from RAM without flashing it. In this case, use the command ***run***, just enter:

```
NetBtLd> run
```

The kernel image starts immediately. After the next reboot the image is lost and must be downloaded from the server again.

11.3Booting

If the command ***run*** has been entered, the kernel will immediately boot.

If the target was flashed with ***lf*** , followed by a system reset (command ***rs***, or pressing of the reset button), an LED on the front panel (CompactPCI or VME) or on the carrier (E²Brain) will flash for about 5 seconds. After this time the kernel will be booted unless the abort button is pressed or the ***abort*** command is entered on the command line.

On the minicom screen, an output similar to the following will appear:

```
NetBtLd
run

loaded at:     00000100 00122298
relocated to:  00800000 00922198
board data at: 0092002C 00920098
relocated to:  008052F0 0080535C
zimage at:     00805AE9 0091F4AA
avail ram:     00923000 10000000

Linux/PPC load: root=/dev/nfs rw console=ttyS0,9600
nfsroot=192.168.3.13:/home/exp
ip=192.168.3.139:192.168.3.13:192.168.3.2:255.255.255.0:ebrain8540:eth2:off

Uncompressing Linux...done.
Now booting the kernel
Memory CAM mapping: CAM0=256Mb, CAM1=0Mb, CAM2=0Mb residual: 0Mb
Linux version 2.6.10-rc3 (claugi@acerclaugi) (gcc version 3.4.3) #2 Wed Mar 9
09:00:14 CET 2005

...
and so on
```

At this point Linux kernel is now running on the target.

However, without a file system it is not possible to execute any kind of a Linux application on this system. The next step is to install a root file system.

12 Root File System Strategies

The Kontron embedded Linux supports various alternatives for mounting a root file system.

It is up to the user to decide which root file system strategy is best suited to the application requirements whereby the following must be considered:

- available capacities
- solid-state or mechanical drive
- removable or fixed
- form factor
- access speed
- price
- read-only or read-write

Possible root file system strategeies are described in the following chapters.

12.1 Onboard FLASH (INITRD/RAMFS)

The Kontron target CPUs have a minimum of 4 MB onboard flash which can be shared between the Linux kernel and a root file system in a ramdisk.image.gz format. Kernel and INITRD/RAMFS are downloaded to the onboard flash in one step.

Advantages**:**

- no external storage media necessary
- easy download
- non-volatile

Restrictions:

- read-only
- space restrictions, kernel and file system must fit in the onboard flash
- no remote debugging facilities

For detailed information refer to “14 The Initial Ramdisk INITRD” or “15 RAMFS”

12.2 CompactFlash, IBM Microdrive™

Many Kontron CPUs, e.g. the Kontron CP620 PowerPC and all E²Brain carriers, have a CompactFlash / IBM Microdrive™ compatible slot. CompactFlash or IBM Microdrives™ are available in capacities between 32 MB and 1 GB.

Advantages:

- broad range of memory capacities
- adapters for the development host available, easy download and software installation
- removable

Restrictions:

- restricted lifetime

  Flash memories have a restricted lifetime if they are used for frequent write operations. If the CompactFlash has an internal wear leveling algorithm the lifetime of this device will be much higher. This should be taken into consideration when purchasing a CompactFlash device.

  This restriction does not apply to the IBM Microdrives™.

 © 2007 Kontron Modular Computers

12.3 IDE-HDD

Some Kontron targets can be equipped with an IDE-HDD. Some have the IDE controller and connector on the baseboard (E²Brain), some others provide open interfaces for IDE extensions (e.g. the PMC-Slot on CompactPCI boards which can be equipped with the Kontron PMC-HDD1 extension). IDE-HDDs provide a cost-effective way to add substantial mass storage capacity.

Advantages:

- broad range of memory capacities
- cost effective
- fast data transfer via DMA

Restrictions:

- not suitable for applications where mechanical drives are not feasible

12.4 SCSI Drives

Kontron boards with a PMC slot can be optionally equipped with a SCSI adapter, e.g. Kontron PMC260.

Advantages:

- broad range of memory capacities
- cost effective
- fast data transfer via DMA

Restrictions:

- not suitable for applications where mechanical drives are not feasible

12.5 M-Systems Disk-On-Chip

Some Kontron PowerPC CPUs, e.g. CP320, CP321, VMP1 and VMP2, support adding an M-Systems Disk-On-Chip to the onboard DIP socket.

Advantages:

- emulates a block device
- partitions can be modified with the standard Linux tools fdisk, mke2fs,
- driver integrated wear levelling for increasing the flash lifetime

Restrictions:

- compatibility problems

  Not all Disk-On-Chip devices are compatible with all Kontron PowerPC CPUs. Contact the Support department for further information regarding the compatibility of Disk-On-Chip devices.

For detailed information refer to “17 Root File System on an M-Systems Disk-On-Chip”

12.6 Remote File System on an NFS Server

Besides the physical mass storage devices described above, a kind of virtual mass storage device can be mounted via NFS.

Advantages**:**

- no external storage media necessary
- very convenient during the debugging phase of an application
- remote debugging is possible
- modifications on the root file system performed on the NFS server are immediately visible for the target (no specific download necessary)

Restrictions:

- not a stand alone solution, server is always required for booting

For detailed information refer to “12.6 ys required for booting ”.

13Setting Up a Diskless Client

A very convenient way for application cross development is to setup the Kontron target as a diskless client which mounts the root file system via NFS on the development host.

All application programs which are cross-compiled on the development host are immediately available on the target system. No download or target reboot is required. Additionally, the remote debugging facility of the **gdb** debugger (with or without the graphical frontend **ddd**) can be used.

13.1Kernel Configuration and Command Line

A description of the required Kernel Configuration and command line can be found in the Linux kernel sources subdirectory *Documentation* in the file: ***nfsroot.txt***.

13.2NFS Server Setup

Set up the development host as an NFS server. Refer to the distribution’s manuals for information on how to setup an NFS server.

Then add an entry in the ***/etc/exports*** file to make a directory for the target that is mountable at system start up. Refer to the manual page of ***exports*** file (man exports) for details on the specific options. Note that the option "no_root_squash" is important in order to enable the target to mount this directory as the root directory.

A line like this must be added to the ***/etc/exports*** file:

```
/nfsroot/<board_type>
*(rw,no_root_squash,no_all_squash,mapping=identity,anonuid=-2,anongid=-2)
```

Remember that in the configuration file this should be one line (not two as above).

13.3Setting Up the Root File System

Now the root file system in ***/nfsroot/<boardtype>*** is required.

For this reason it is necessary to unpack the root file system from the Kontron CD-ROM into this directory.
First, login as root; create the root directory for this archive; switch to this directory; and unpack the root file system archive from the CD-ROM:

 © 2007 Kontron Modular Computers

```
usr@x86host:~ > su
Password: xxxxx
root@x86host: # cd /
root@x86host: # mkdir nfsroot
root@x86host: # cd nfsroot
root@x86host:/nfsroot # mkdir <boardtype>
root@x86host:/nfsroot # cd <boardtype>
root@x86host:/nfsroot/<boardtype> # tar --preserve-permissions -xvjf
/opt/linux26_bsp_kom/rootfs_<architecture>_<subarchitecture>.<version>/
root_filesystem_<architecture>_<subarchitecture>_linux.tar.bz2
```

13.4 Booting from NFS Root File System

After performing the above steps, the target should be rebooted. It will now be able to mount the root file system via NFS. It should start into a Linux prompt with messages similar to the following:

```
IP-Config: Complete:
      device=eth2, addr=193.102.136.44, mask=255.255.255.0, gw=193.102.136.1,
     host=ebrain8540, domain=, nis-domain=(none),
     bootserver=193.102.136.40, rootserver=193.102.136.40, rootpath=
Looking up port of RPC 100003/2 on 193.102.136.40
eth2: Full Duplex
eth2: Speed 100BT
eth2: Link is up
Looking up port of RPC 100005/1 on 193.102.136.40
VFS: Mounted root (nfs filesystem).
Mounted devfs on /dev
Freeing unused kernel memory: 96k init

Please press Enter to activate this console.


BusyBox v1.00 (2005.02.23-16:12+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

Processing /etc/profile... Done

/ #
```


© 2007 Kontron Modular Computers

14The Initial Ramdisk INITRD

14.1Introduction

INITRD provides the capability to load a small root file system together with the kernel into the target's onboard flash. The kernel loader code detects the packed INITRD image if present and mounts it as the root file system.

INITRD is principally designed to allow system start up to occur in two phases: first, the kernel comes up with a minimum set of built-in drivers, and second, additional modules are loaded from INITRD.

Depending on the size of the application the INITRD may be sufficient to hold the root file system and the application program. This is an important feature especially for embedded systems. In this case, a Linux application may be downloaded and run without any additional external storage components.

For additional information about the INITRD, refer to the file: ***Documentation/initrd.txt*** in the kernel source tree.

14.2How to Create or Modify an INITRD

This chapter describes, how to create an INITRD in the ramdisk.image.gz format, which can be added to the kernel and programmed to the onboard flash then.

Prerequisites:

a)The kernel of the development host must support the loopback device. This means that the *mount –o loop* option must be supported.

b)Logged in as super user.
In this special case it is necessary to work as the root user, because all the system files of the root file system need to be owned by the root.

14.2.1Creating a New INITRD Image

RAM disk image space is restricted. The available space inside such a RAM disk image is defined when it is created. To define a bigger (or smaller) RAM disk image, or, to use a different kind of file system, a new INITRD image must be created.

First, create an empty image. This is done with the commands:

```
root@x86host: # dd if=/dev/zero of=ramdisk.image bs=1024 count=16384
16384+0 records in
16384+0 records out
root@x86host: # /sbin/mkfs.ext2 ramdisk.image
mke2fs 1.38 (30-Jun-2005)
ramdisk.image is not a block special device.
Proceed anyway? (y,n) y
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
4096 inodes, 16384 blocks
819 blocks (5.00%) reserved for the super user
First data block=1
2 block groups
8192 blocks per group, 8192 fragments per group
2048 inodes per group
Superblock backups stored on blocks:
        8193
```

```
Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 39 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override
```

The command ***dd*** creates an empty file with a size matching the space, which shall be available for the ramdisk (in the above example it is 16384*1024=16 MB).

The mkfs.ext2 command then creates an ext2 file system within this file. However, any other file system type could be used alternatively, if desired. Refer to the documentation for *dd*, *mkfs* for more details on the arguments for these commands.

Now, create a temporary directory for the mount point and then mount this image to this mount point

```
root@x86host: /home/usr/bsp_work/ramdisk_work # mkdir mnttmp
root@x86host: /home/usr/bsp_work/ramdisk_work # mount -o loop ramdisk.image
mnttmp
```

Entering the mount directory and inspecting the contents will provide the following results:

```
root@x86host: /home/usr/bsp_work/ramdisk_work # cd mnttmp/
root@x86host: /home/usr/bsp_work/ramdisk_work/mnttmp # ls
.  ..  lost+found
root@x86host: /home/usr/bsp_work/ramdisk_work/mnttmp #
```

And the diskfree-command ***df*** will show the amount of available space in the ramdisk

```
root@x86host: /home/usr/bsp_work/ramdisk_work/mnttmp # df -h
…
/home/usr/bsp_work/ramdisk_work/ramdisk.image
       16M   13K   15M   1% /home/usr/bsp_work/ramdisk_work/mnttmp
```

Now, the contents of the Linux GPL root file system can be unpacked to this directory

```
root@x86host: /home/usr/bsp_work/ramdisk_work/mnttmp # tar --preserve-
permissions -xvjf
/opt/linux26_bsp_kom/rootfs_<architecture>_<subarchitecture>.<version>/
root_filesystem_<architecture>_<subarchitecture>_linux.tar.bz2
```

e.g.

```
root@x86host: /home/usr/bsp_work/ramdisk_work/mnttmp # tar -xvjf
/opt/linux26_bsp_kom/rootfs_ppc_e300.0100/root_filesystem_ppc_e300_linux.tar.
bz2
```

Now, this ramdisk.image has to be unmounted, and then compressed with ***gzip***. This is the format, which is required to add it as an INITRD to the Kernel.

```
root@x86host: /home/usr/bsp_work/ramdisk_work/mnttmp # cd ..
root@x86host: /home/usr/bsp_work/ramdisk_work # umount mnttmp
root@x86host: /home/usr/bsp_work/ramdisk_work # gzip ramdisk.image
```

To be able to add ***ramdisk.image.gz*** to the kernel, it has to be copied to the appropriate location.

```
root@x86host: /home/usr/bsp_work/ramdisk_work # cp ramdisk.image.gz
<kerneldirectory>/arch/<architecture>/boot/images/ramdisk.image.gz
```

For XScale based boards please refer to Appendix C: XScale IXP42x based eBrain specific considerations.

© 2007 Kontron Modular Computers

14.2.2 Inspecting and Modifying an Existing INITRD Image

The previous chapter showed, how a new INITRD image can be created and filled with the contents of the provided GPL root file system. A common task for an application programmer will be to modify this existing image by adding own files, e.g. self-developped applications or remove unused components.

To do this, the existing ***ramdisk.image.gz*** first has to be unzipped.

```
root@x86host: /home/usr/bsp_work/ramdisk_work # gunzip <ramdisk.image.gz
>ramdisk.imagenew
```

Now it can be mounted via the loop device.

```
root@x86host: /home/usr/bsp_work/ramdisk_work # mkdir mntloop
root@x86host: /home/usr/bsp_work/ramdisk_work # mount -t auto -o loop
ramdisk.imagenew mntloop
```

Now the RAM disk image is mounted to the directory mntloop. Enter the command to list its contents:

```
root@x86host: /home/usr/bsp_work/ramdisk_work # ls -l mntloop
```

and the output looks like this:

```
root@x86host: /home/usr/bsp_work/ramdisk_work # ls -l mntloop/
total 20
drwxr-xr-x  10 root root  1024 2005-03-08 18:17 .
drwxr-xr-x   3 root root   136 2005-03-09 10:48 ..
drwxr-xr-x   2 root 1000  1024 2005-03-08 17:36 bin
drwxr-xr-x   2 root 1000  1024 2005-02-23 12:32 dev
drwxr-xr-x   3 root 1000  1024 2005-02-23 12:32 etc
drwxr-xr-x   2 root 1000  1024 2005-03-08 18:08 lib
lrwxrwxrwx   1 root root    11 2005-03-08 18:17 linuxrc -> bin/busybox
drwx------   2 root root 12288 2005-03-08 18:16 lost+found
drwxr-xr-x   2 root 1000  1024 2005-02-23 12:32 proc
drwxr-xr-x   2 root 1000  1024 2005-03-08 17:40 sbin
drwxr-xr-x   4 root 1000  1024 2005-02-23 17:23 usr
root@x86host: /home/usr/bsp_work/ramdisk_work #
```

The entire root file system, which was hidden previously, is visible now. It is possible to inspect or modify the contents of such a file system. With the tool **df -h** it is also possible to check the free space on the mounted RAM disk image.

```
root@x86host: /home/usr/bsp_work/ramdisk_work # df -h
Filesystem            Size  Used Avail Use% Mounted on
. . .
/home/usr/bsp_work/ramdisk_work/ramdisk.imagenew
                      3.4M  3.2M  3.0K 100% /home/usr/bsp_work/ramdisk_
work/mntloop
```

The file system may be modified according to the application requirements, e.g. add applications which where cross-compiled for this hardware.

Once the file system has been modified it must be converted back to an updated ***ramdisk.image.gz*** file and stored back into the kernel. This is achieved with the commands:

```
root@x86host: /home/usr/bsp_work/ramdisk_work # umount mntloop
root@x86host: /home/usr/bsp_work/ramdisk_work # gzip ramdisk.imagenew
root@x86host: /home/usr/bsp_work/ramdisk_work # cp ramdisk.imagenew.gz
        <kerneldirectory>/arch/<architecture>/boot/images/ramdisk.image.gz
```

For XScale based boards please refer to Appendix C: XScale IXP42x based eBrain specific considerations.

 © 2007 Kontron Modular Computers

14.3Adding an INITRD to the Kernel

The previous chapters explained, how an INITRD is generated. Now this INITRD shall be bound to the Kernel image in a way, that it can be programmed to the onboard flash and booted together with the Kernel.

To achieve this, the following has to be done.

14.3.1Kernel Configuration

For INITRD support, the following options must be set in the Kernel configuration:

```
Device Drivers
      Block devices
            <*> RAM disk support
            (16)  Default number of RAM disks (NEW)
            (4096)      Default RAM disk size
             [*] Initial RAM disk (initrd) support
```

Please note that the “Default RAM disk size” option refers to the size of INITRD image before compression. Example: *if the 16MB large has been created, the “Default RAM disk size” should be configured for at least 16MB INITRD image*. If the RAM disk size is too small, an error will be generated at kernel start.

The INITRD is also formatted similar to a mass storage to one of supported in Linux file system types, e.g. ext2, minix. For this reason also file system support for the chosen type must be enabled in the kernel statically.

14.3.2Kernel Command Line

In the kernel command line the root device must be configured as:

```
root=/dev/ram0
```

14.3.3Building a Kernel image including INITRD

To create a kernel image including this image as an INITRD, just call:

```
usr@x86host:~/bsp_work/linux> make zImage.initrd
```

This will generate a file ***zImage.initrd.kom*** in the subdirectory *arch/<architecture>/boot/images*.

For XScale based boards please refer to Appendix C: XScale IXP42x based eBrain specific considerations.

14.4Downloading

Download the newly created kernel image as described in chapter “11.2 Network Download”.

© 2007 Kontron Modular Computers

14.5Booting

Connect the minicom terminal and boot the system by pressing the reset button.

The early kernel startup messages now contain a line "*initrd at:*" which was not present when the kernel was booted without an initrd, e.g.:

```
run

loaded at:     00000100 00261298
relocated to:  00800000 00A61198
board data at: 00A5F02C 00A5F098
relocated to:  008052D8 00805344
zimage at:     00805AD1 00920EFE
initrd at:     00921000 00A5E81C
avail ram:     00A62000 10000000

Linux/PPC load: root=/dev/ram0 console=ttyS0,9600
ip=193.102.136.44:193.102.136.43:193.102.136.1:255.255.255.0:eb8540_initrd:et
h2:off
Uncompressing Linux...done.
Now booting the kernel
```

Linux will come up with the command line prompt:

```
RAMDISK: Compressed image found at block 0
VFS: Mounted root (ext2 filesystem) readonly.
Mounted devfs on /dev
Freeing unused kernel memory: 104k init

Please press Enter to activate this console. eth2: Full Duplex
eth2: Speed 100BT
eth2: Link is up

BusyBox v1.00 (2005.02.23-16:12+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.
Processing /etc/profile... Done

/ # uname -a
Linux eb8540_initrd 2.6.10-rc3 #3 Wed Mar 9 09:59:34 CET 2005 ppc unknown
unknown GNU/Linux
/ #
```

 © 2007 Kontron Modular Computers

15INITRAMFS

RAMFS is a very simple filesystem that exports Linux disk caching mechanisms as a dynamically resizable ram-based filesystem. It is very similar mechanism to INITRD but much simpler in use, because it does not require the root privileges to build kernel image with root filesystem included.

More information is available in the kernel documentation in file "*Documentation/filesystems/ramfs-rootfs-initramfs.txt"*

15.1.1Kernel Configuration

The *initramfs* is configured in kernel menu configuration in **General setup** section. The menu entry **Initramfs source files** must contain a valid path to the root filesystem.

```
()   Local version - append to kernel release
[*]  Automatically append version information to the version string
[*]  Support for paging of anonymous memory (swap)
[*]  System V IPC
[ ]  POSIX Message Queues
[ ]  BSD Process Accounting
[*]  Sysctl support
[ ]  Auditing support
[*]  Kernel .config support
[*]    Enable access to .config through /proc/config.gz
[ ]  Kernel->user space relay support (formerly relayfs)
(/home/root) Initramfs source file(s)
(0)    User ID to map to 0 (user root) (NEW)
(0)    Group ID to map to 0 (group root) (NEW)
[ ]  Optimize for size (Look out for broken compilers!)
[*]  Configure standard kernel features (for small systems)  --->
```

15.1.2Kernel Command Line

In the kernel command line the root device must not (!) be configured.

15.1.3Building a Kernel image with root filesystem in RAMFS

To create a kernel image including root filesystem in RAMFS, just call:

```
usr@x86host:~/bsp_work/linux> make
```

This will generate a file ***vmlinux.gz*** or ***vmlinux.bin*** in the subdirectory *arch/<architecture>/boot*

15.2Downloading and Booting

Download the newly created kernel image as described in chapter "11.2 Network Download" and in chapter "11.3 Booting".

16Root File System on a Mass Storage Device (IDE/SCSI/CompactFlash)

Some Kontron targets can be equipped with an IDE-HDD. Some have the IDE controller and connector on the carier (E²Brain), others provide open interfaces for IDE extensions (e.g. the PMC slot on CompactPCI boards which can be equipped with the Kontron PMC-HDD1 extension). IDE-HDD provides a cost-effective way to add substantial mass storage capacity.

Kontron boards with a PMC slot can be optionally be equipped with a SCSI adapter (e.g. Kontron PMC260).

Many Kontron targets are equipped with a CompactFlash socket onboard.

16.1Required Kernel Configuration for CompactFlash

For all Kontron boards with a CompactFlash socket onboard, this controller is already activated in the kernel default configuration of the BSP.

16.2Required Kernel Configuration for IDE

For all Kontron boards with IDE controller onboard, this controller is already activated in the kernel default configuration of the BSP.

For third-party controllers it may be required to enable the appropriate drivers in the kernel configuration.

16.3PMC260 or CP360 Required Kernel Configuration

For the Kontron SCSI controller PMC260 or CP360 please configure the kernel as follows.
Be sure to enable SCSI support in the kernel configuration and select the appropriate SCSI low-level driver. In the case of a PMC260 SCSI module this would be:

```
SCSI support  --->
  <*> SCSI support
  <*>   SCSI disk support (NEW)
  SCSI low-level drivers  --->
    <*> SYM53C8XX SCSI support (NEW)
```

After generating and downloading the kernel the SCSI hard drive should be detected by the Linux kernel. During the boot process an output similar to this should be displayed:

```
SCSI subsystem driver Revision: 1.00
sym53c8xx: at PCI bus 0, device 20, function 0
sym53c8xx: setting PCI_COMMAND_PARITY...(fix-up)
sym53c8xx: 53c895 detected
sym53c895-0: rev 0x1 on pci bus 0 device 20 function 0 irq 3
sym53c895-0: NCR clock is 40037KHz
sym53c895-0: ID 7, Fast-40, Parity Checking
sym53c895-0: on-chip RAM at 0x80204000
sym53c895-0: restart (scsi reset).
sym53c895-0: Downloading SCSI SCRIPTS.
sym53c895-0: SCSI bus mode change from 80 to 80.
sym53c895-0: restart (scsi mode change).
sym53c895-0: Downloading SCSI SCRIPTS.
scsi0 : sym53c8xx - version 1.6b
  Vendor: QUANTUM   Model: VIKING II 4.5WLS  Rev: 5520
  Type:   Direct-Access                      ANSI SCSI revision: 02
```

16.4The Chicken/Egg Problem

Imagine an embedded Linux target and an unformatted hard drive connected to it. How is it possible to install a Linux root file system to the hard drive?

16.4.1Preparation of the Root File System on the Development Host

In case of a removable media (CompactFlash) one possible procedure is as follows:

- attach the empty CompactFlash to the development host using a CompactFlash card reader (USB, PCMCIA)
- create a file system on the CompactFlash as required

  With some file systems endianess problems can arise. In particular if the mkfs.xxx program is called on a little-endian machine, and then an attempt is made to mount this file system on a big-endian machine. However, most file systems can handle this. Still it may be necessary to determine this empirically.

- mount the CompactFlash on the development host
- unpack the root file system which is provided in the BSP to the CompactFlash

```
root@x86host: # mount <compact_flash_device> /mnttmp
root@x86host: # cd /mnttmp
root@x86host: # tar --preserve-permissions -xvjf
/opt/linux26_bsp_kom/rootfs_<arch>_<subarch>.<version>/
root_filesystem_<arch>_<subarch>_linux.tar.bz2
```

- configure the kernel command line with ***root=/dev/<compact-flash-device>***, download this new kernel and reboot

16.4.2Preparation of the Root File System on the Target

For non-removable devices or to totally exclude endianess problems, the following method is also possible:

- attach the IDE or SCSI disk to the target

boot the system as described in “12.6 ys required for booting ”

- create a file system on the disk with a mkfs.xxx program
- mount the drive
- unpack the root file system which is provided in the BSP to the CompactFlash

```
root@target: # mount <disk_device> /mnt
root@target: # cd /mnt
root@target: # tar --preserve-permissions -xvjf
[path_to_rootfs_package]/root_filesystem_<arch>_<subarch>_linux.tar.bz2
```

- configure the kernel command line with ***root=/dev/<disk-device>***; download this new kernel and reboot

17Root File System on an M-Systems Disk-On-Chip

17.1General

The Kontron PowerPC boards CP320, CP321, VMP1, and VMP2 are equipped with an onboard socket for the M-Systems Disk-On-Chip. This device is supported by the Memory Technology Device (MTD) support which is integrated within the Linux kernel. The Linux MTD drivers are still subject to frequent modifications. The official MTD site is:

http://www.linux-mtd.infradead.org

This chapter includes some excerpts from the excellent “how to” written by Vipin Malik:

vipin@embeddedLinuxWorks.com

The original document can be found under:

ftp://ftp.uk.linux.org/pub/people/dwmw2/mtd/cvs/mtd/mtd-jffs-HOWTO.txt

For detailed information and useful links, refer to the above document.

The Linux kernel source provided by Kontron already contains a version of the MTD driver which works correctly with the Kontron PowerPC CPUs and the Disk-On-Chip.

17.2Kernel Configuration

For all Kontron boards with a Disk-On-Chip socket onboard, the support for this device is already activated in the provided Kernel default configuration.

17.3Creating Partitions

In typical applications the Disk-On-Chip will be used very often to store the root file system. To achieve this, first an appropriate file system must be installed on the Disk-On-Chip.

The following command line when run on the target system will return the contents (partitions) of the installed DOC device:

```
root@target: # ls /dev/nftla*
/dev/nftla   /dev/nftla1
```

The example results show two files ***/dev/nftla*** and ***/dev/nftla1***. The ***nftla*** file represents whole DOC device, the ***nftla1*** file represents the partition already existing on the device.

If there is no partition on the device, there will be only the *disc* file in the result.

Use the **fdisk** utility to remove any existing partitions and create the new one as needed. The **fdisk** utility should be invoked with:

```
root@target: # fdisk /dev/nftla
```

For details about usage of the **fdisk** utility, see its manual page.

After the partition is created and changes are written to the disc, the partition must be formatted with the requested file system type.

For example:

Formatting the partition with **ext2** filesystem:

```
root@target: # mkfs.ext2 /dev/nftla1
mke2fs 1.36 (05-Feb-2005)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
6000 inodes, 23996 blocks
1199 blocks (0.00%) reserved for the super user
```

```
First data block=1
3 block groups
8192 blocks per group, 8192 fragments per group
2000 inodes per group
Superblock backups stored on blocks:
        8193

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 23 mounts or
2.11367e-314 days, whichever comes first.  Use tune2fs -c or -i to override.
```

17.4Preparation of the Root File System on the Disc-On-Chip

To prepare the DOC for the root file system installation perform the following:

•plug the Disc-On-Chip device into the board

boot the system as described in "12.6 ys required for booting "

•mount the drive

•copy the root file system package which is provided in the BSP to the exported via NFS directory mounted as a root file system by the target board

•unpack the root file system which is provided in the BSP to the Disc-On-Chip

```
root@target: # mount /dev/nftla1 /mnt
root@target: # cd /mnt
root@target: # tar --preserve-permissions -xvjf
[path_to_rootfs_package]/root_filesystem_<arch>_<subarch>_linux.tar.bz2
```

•umount the Disk-On-Chip device

•configure the kernel command line with ***root=/dev/nftla1***; download this new kernel and reboot

18Cross Development

For application development and debugging, the PowerPC target will typically mount the root file system via NFS. There are no specific requirements for the root file system in this case, but, generally speaking, usually it is not necessary to focus on keeping it very small.

18.1Cross Compilation

For cross compilation of user space applications the same cross development tool chain should be used as for Linux kernel. As a base for cross compilation, the Makefile provided in the target directory of the BSP may be used.

In this case, to recompile all user space applications and kernel modules enter the following:

```
usr@x86host:> cd target
usr@x86host:> source ../scripts/SETENV.sh
usr@x86host:> make
```

Each of the BSP applications can be compiled also by hand. A good example would be the 'primes' application:

```
usr@x86host:> mkdir temp
usr@x86host:> cd temp
usr@x86host:> source ../scripts/SETENV.sh
usr@x86host:> "$CROSS_COMPILE"gcc -g ../src/primes.c
```

In this case the ‘a.out’ binary should appear in *temp* directory. Note that ‘primes.c’ is a very simple application that does not need specific header files or libraries. For other applications included in the BSP, compilation is a little more complicated.

The ‘-g’ option in the command line causes the gcc to produce the debugging information in the binary file ‘a.out’. Without this information debugging is impossible.

The binary file can be transferred to the target file system. In case of an NFS based file system, it can just be copied into an exported directory:

```
usr@x86host:> cp a.out /nfsroot/<boardtype>/primes
```

Now it can be executed on target system:

```
Please press Enter to activate this console.


BusyBox v1.00 (2005.02.23-14:30+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.


Processing /etc/profile... Done

/ # ./primes
        This program lists all primes <= n

Input n: 10


        Primes <= 10:

2
3
5
7
/ #
```

 © 2007 Kontron Modular Computers

18.2Debugging Tools gdb and gdbserver

The most commonly used tool for debugging under Linux is the gdb debugger.

In the cross-development environment, gdb is divided into two parts:

- gdbserver

  runs on the PowerPC target and controls the execution of the program being debugged

- gdb

  runs on the host and provides a command interface and access to symbol information and source code

Both gdb and gdbserver are free software covered by the GNU General Public License. To make things easier, the program binaries are included in the BSP package. These binaries are configured and tested for x86-host to PowerPC-target cross debugging.

- gdbserver is located under: bsp_<boardtype>/target/bin/gdbserver
- gdb is located under: bsp_<boardtype>/host/bin/powerpc-linux-gdb

Note that this version of the gdb is specifically configured for cross debugging. To avoid confusion with the native x86 gdb, this version's file name is 'powerpc-linux-gnu-gdb'.

gdbserver should be copied to the file system on the target machine and 'powerpc-linux-gnu-gdb' should be stored on the x86 host machine.

18.3A Sample Debugging Session

Before trying to start the debugging session, the binary of an application suitable for debugging should be copied to the target file system. To compile applications with debugging information included, option "-g" should be passed on to gcc compiler. Applications built by Makefile provided in the BSP already have debugging information included.

For example the 'primes' application is to be debugged.

At the terminal of target machine call:

```
usr@x86host:> /<gdbserver path>/gdbserver aaa.bbb.ccc.ddd:1234
/<primes_app_path>/primes
Process /<primes_app_path>/primes created; pid = 148
```

where:

- aaa.bbb.ccc.ddd

  is the IP address of the development host

- 1234

  is the network port number on which the server will wait for a connection from the client; this can be any available port number on the target

On the development host gdb must be started and connected to the above gdbserver:

```
usr@x86host:> ./powerpc-linux-gnu-gdb
GNU gdb 6.3
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=powerpc-linux-
gnu"
.
```

```
(gdb) target remote aaa.bbb.ccc.ddd:1234
Remote debugging using aaa.bbb.ccc.ddd:1234
0x3000f1b0 in ?? ()
(gdb)
```

where:

- aaa.bbb.ccc.eee

  is the IP address of the PowerPC target

- 1234

  is the same port number as chosen in the gdbserver command line argument

When cross-debugging the gdb on the development host should be able to find the absolute and non-absolute shared library symbol files. The proper search paths are represented by gdb internal variables:

- solib-search-path - the search path for loading non-absolute shared library symbol files
- solib-absolute-prefix - prefix for loading absolute shared library symbol files

These paths are configured in following way:

```
(gdb) set solib-search-path /dev/null
(gdb) set solib-absolute-prefix /<path to the target libraries>/
(gdb)
```

In case if the root filesystem is exported by NFS server the path to the target libraries may be set as **/<path to the exported root filesystem>/lib**.

However if the target root filesystem is not exported by NFS server (ex. The root filesystem is installed on the CF device), this path may point to the target libraries included in the cross-development toolchain: **/opt/crosstool/<architecture>-linux-gnu/gcc-<gcc version>-glibc-<glibc version>/<architecture>-linux-gnu/lib**

Now load the symbols for the file to debug:

```
(gdb) sym bin/primes
Reading symbols from bin/primes...done.
```

Set breakpoints on line 28 and line 33 of the program:

```
(gdb) break 28
Breakpoint 1 at 0x10000414: file .../primes.c, line 28.
(gdb) break 33
Breakpoint 2 at 0x10000460: file .../primes.c, line 33.
```

Now execute the program, until it reaches the first breakpoint:

```
(gdb) c
Continuing.

Breakpoint 1, main ()
    at .../primes.c:28
28          printf( "\tThis program lists all primes <= n\n\n" );
(gdb)
```

Continue the program:

```
(gdb) c
Continuing.
```

Now the prompt does not appear immediately. The reason for this is that the program prompts for input. This is visible in the target command line:

```
usr@target: # /bin/gdbserver aaa.bbb.ccc.ddd:1234 ./primes
Process ./primes created; pid = 151
Listening on port 1234
```

```
Remote debugging from host aaa.bbb.ccc.ddd
        This program lists all primes <= n

Input n:

        Primes <= 2:

2
```


© 2007 Kontron Modular Computers

After entering a number, the command proceeds until it reaches the breakpoint in line 33. On the development host, the gdb command prompt will appear again:

```
Breakpoint 2, main ()
    at .../primes.c:33
33          for ( possible_prime = 2; possible_prime <= n;
(gdb)
```

Now the basic startup procedure of a remote debugging session should be clear. More detailed information can be found in the gdb documentation.

18.4The .gdbinit File

At this point a .gdbinit file should be created.

When gdb is started, it automatically executes commands from its "init files". These files are named `.gdbinit' on Linux systems. During startup, gdb does the following:

- reads the init file (if any) in the home directory
- processes command line options and operands
- reads the init file (if any) in the current working directory
- reads command files specified by the `-x' option

In this case, specific properties for the remote cross debugging needs to be configured. It is recommended to create a “.gdbinit” file in the directory from which gdb is called.

For example:

```
echo "PowerPC remote debugging configured"
directory src
set endian big endian
target remote aaa.bbb.ccc.eee:1234
set solib-search-path /dev/null
set solib-absolute-prefix /<path to the target shared libraries>/lib
```

With a “.gdbinit” file in this format, the connection to the remote target is automatically performed when gdb is called.

Now the start of the debug session as described in the previous chapter is performed as follows on the target command line:

```
usr@target: # /bin/gdbserver aaa.bbb.ccc.ddd:1234 /bin/primes
Process /<primes_application_path>/primes created; pid = 148
```

Command on the development host command line:

```
usr@x86host:> powerpc-linux-gnu-gdb
GNU gdb 6.3
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=powerpc-linux-
gnu".
"PowerPC remote debugging configured"The target is assumed to be big endian
0x3000f1b0 in ?? ()
(gdb)
```

The target will be immediately connected. This is visible on the target shell with the message:

```
Remote debugging using aaa.bbb.ccc.ddd:1234
```

and debugging can begin.

18.5Graphical Debugging

Migrating from the gdb command line interface to the ddd GUI is an easy step, because ddd is just a graphical front end for the gdb. The low-level debugging task is still performed by gdb.

Requirements:

- command line remote debugging with gdb, as described in the previous chapter, must work correctly
- ddd debugger is installed on the development host (ddd is contained in most of desktop distributions e.g. SuSE, Fedora, Red Hat)
- there is a file ".gdbinit" in the current directory as described above; this is not mandatory, but more convenient

The debugging can be started as follows:

On the target command line it is the same as described in the previous chapter:

```
usr@target :~/target # /bin/gdbserver aaa.bbb.ccc.ddd:1234 /bin/primes
Process /bin/primes created; pid = 148
```

On the development host, now start ddd instead of the powerpc-linux-gdb. But ddd must know that it should use the powerpc-linux-gdb as the inferior debugger.

ddd is a program that is indifferent to the hardware specific details. It always uses an underlying command line debugger to perform the hardware relevant job.

The command line to start ddd is:

```
usr@x86host:> ddd --debugger powerpc-linux-gdb bin/primes&
ddd --debugger <path to debugger>/powerpc-linux-gnu-gdb ./bin/primes
```

This will open a screen similar to this:

On the first occasion that ddd is started for cross-debugging, go to the menu

Edit - GDB Settings.

Move to the items:

architecture of target

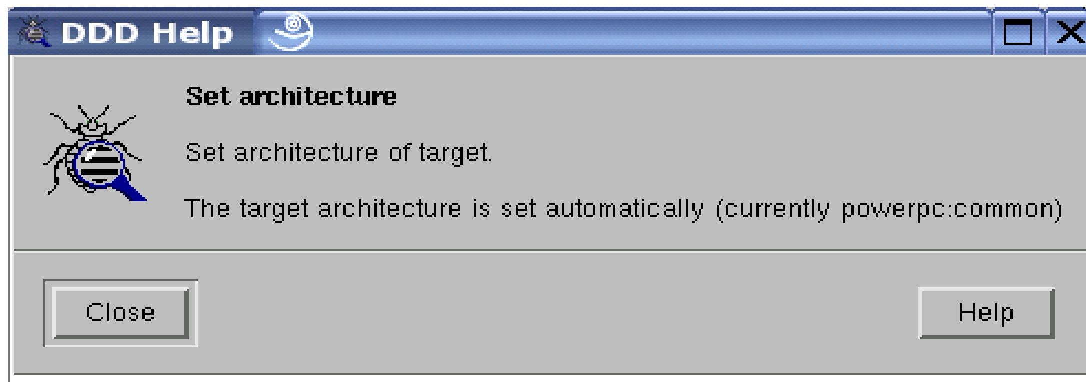
and

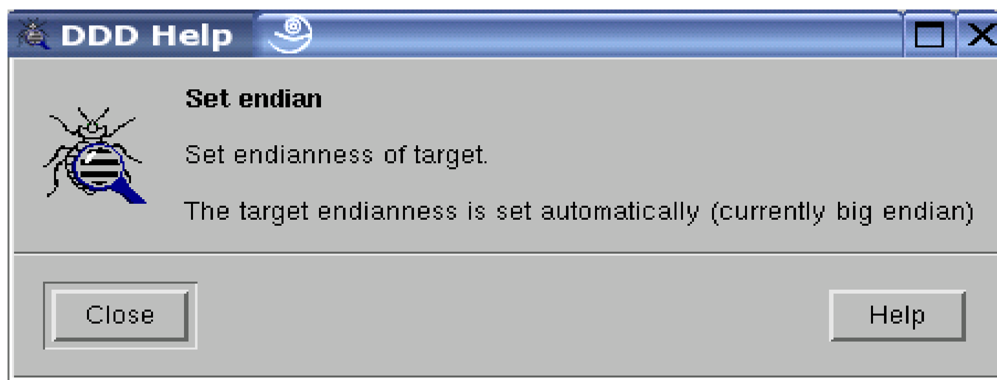
endianness of target.

Normally, these two items are set to auto per default. The target architecture should be detected correctly in this mode. In any case, this should be verified by clicking on the “***?***” sign.

 © 2007 Kontron Modular Computers

The output should look as follows:





If this is not the case, the architecture **powerpc:common** and **endianness: big endian** will need to be manually configured here.

After exiting this menu, choose:

Edit - Save Options

to make the modifications permanent.

Now debugging with ddd can be performed. For help with ddd refer to the ddd online Help or the manuals available for ddd.

18.6Execution Performance

The right selection of GCC compilation options has a very high impact on the performance.

Note that some of the options may force the compiler to omit implementing some checking/safe features to output code.

Here are list of options that should be used to achieve good performance:

OPTION	DESCRIPTION
-O3	Turns on optimization
-fomit-frame-pointer	Removes the frame pointer for all functions which might make debugging more difficult
-funroll-loops	Unroll loops when number of interations may be determined during compilation
-fforce-mem	Force memory operands to be copied into registers before doing arithmetic on them
-falign-loops=n	Align start of loop to next power of ‘n’, skipping up to ‘n’ bytes
-falign-functions=n	Align start of function to next power of ‘n’ skipping up to ‘n’ bytes
-falign-jumps=n	Align start of places to jump into to next power of ‘n’ skipping up to ‘n’ bytes
-ffast-math	Causes __FAST_MATH__ macro to be defined This may result in incorrect output for programs that depend on an exact implementation of IEEE and ISO rules.
-msoft-float (for FPU-less machines)	Generate output containing library calls for floating point

A complete description of these features may be found in the manual for the gcc compiler.

18.7PowerPC e500 Specific Optimization

For cross development, the tool chain contains the Freescale provided ‘libfsl_e500.a’ library. This library replaces some of glibc math library calls providing code that executes much faster on with the e500 core (especially double precision). The library is included in the cross-compilation tool chain for the e500 in a path that fits to the default library search path.

The following information is provided by Freescale regarding this library.

```
*************************************************************
Library Math e500 version 1.0 ESS3
Build 06
readme.txt
Copyright Motorola, Inc. 2004
Motorola Confidential Proprietary
*************************************************************
1.1. Contents

libfsl_e500.a - static ELF library for arithmetic and mathematical functions

1.2  Usage Example

When using GNU tools add "-lfsl_e500" to the command line,before linking
with the GNU libgcc or math libraries:

gcc $(CFLAGS) -L$(LDIR) -o myapp myapp.c -lfsl_e500  -lm

1.3. Functions supported

1.3.1 The following arithmetic functions are supported:

add (__adddf3)
sub (__subdf3)
multiply (__muldf3)
divide (__divdf3)
negate (__negdf2)
less than (__ltdf2)
equal to (__eqdf2)
greater than (__gtdf2)
less or equal (__ledf2)
not equal to (__nedf2)
greater or equal (__gedf2)
low-level general comparison (__cmpdf2)
low-level unordered comparison (__unorddf2)

data type conversions:
float to double
signed int to double
unsigned int todouble
signed long long to double
unsigned long long to double
double to float
double to signed int
double to unsigned int
double to signed long long
double to unsigned long long

1.3.2 The following math functions are supported:

floating absolute value (fabs)
square root (sqrt)
sine (sin)
cosine (cos)
nearest integer rounding (ceil, floor)
exponent (exp)
logarithm (log, log10)
power (pow)
arctangents (atan, atan2)
fractional and integer decomposition (modf)
floating point remainder (fmod)
```

Better performance with an e500 core can also be achieved by including ‘-Wa,-me500’ compiler options to the build process.

19Linux GPL Root File System

The Linux GPL root file system package includes many common utilities in a single, small, root directory tree. It provides most of the utilities usually found in GNU coreutils, util-linux, etc.

The previous chapters regarding root file system strategies already described how to install this root file system on the target. This chapter provides a brief overview about the contents of this root file system.

The GPL root file system is provided on the CD-ROM in a ready-to-use binary format for every supported architecture family. The binary format ensures that the Linux programs are already compiled with the correct toolchain, and use the correct runtime libraries.

Information for Advanced Users

Additionally, the sources of this GPL root file system are enclosed on the CD-ROM under the directory GPLSOURCES. The sources are delivered under the GPL without any warranties or support.

19.1Application List and Command Reference

Detailed help on all Linux commands can be found in the file system installation directory ***/opt/linux26_bsp_kom/rootfs_<architecture>_<subarchitecture>.<version>/doc/.*** The cover sheet *index.html* within this document provides information about all available Linux programs and links to the command references of the particular commands.

19.2Directory structure

One of the final operations performed by the Linux kernel during system boot-up is the mounting of the root file system. The root file system is an essential component of all Linux systems.

Each top-level directory in the root file system has a specific purpose:

/bin	commands required during boot-up and the tools for use by normal users
/sbin	like /bin, but the commands are not intended for normal users, although they may use them if necessary and allowed
/etc	configuration files
/root	the home directory for root user
/lib	shared libraries needed by the applications
/lib/modules	loadable kernel modules; in subdirectory fitting to Kernel version (***uname –r***)
/dev	device files
/proc	mount points for the proc file system.
/usr	secondary hierarchy containing most applications useful to most users
/var	variables data store by demons and utilities
/home	root of the user home directories

 © 2007 Kontron Modular Computers

19.3Linux System Configuration Files

The **/etc** directory contains configuration files. They are described below:

/etc/rc.d	scripts or directories including scripts running at startup or when changing the runlevel
/etc/passwd	user passwords database The format is documented in the 'passwd' manual page (**man passwd**).
/etc/fstab	lists the file systems to be mounted automatically at startup (**man fstab**)
/etc/group	similar to /etc/passwd, but describes groups instead of users Refer to the group manual page for more information (**man group**).
/etc/inittab	configuration file for init This file format is described within the busybox man page.
/etc/inet.conf	configuration file for inetd processes
/etc/mtab	list of currently mounted file systems
/etc/shadow	shadow password file on systems with shadow password software installed
/etc/profile	files executed at login or startup time by shell These files allow the system administrator to set global defaults for all users. Refer to the manual pages for the respective shells.

19.4 Device files

In the Linux operating system communication with every device is realized by using "device files". These are special files in the */dev/* directory. The device file has following characteristics:

-Device type (ex. Block, character, pipe)
-major number
-minor number.

Over the time there were different strategies to handle them. Initially the device files were created statically during the system installation. However there were some problems regarding this approach. With the time the */dev/* file had more and more files, and only a small number of them were really used in the system. However if the device used in the system wasn't popular, or was some special device, the proper device file had to be created specially. Also the hot plugged devices caused lot of problems with static */dev/* contents.

At some point the pseudo filesystem *devfs* has been developed and included in the Linux kernel. The *devfs* controlled the device files creating during kernel boot-up and drivers loading. So the */dev* directory contained only the files of the initialized devices. However this approach (or solution) was difficult to maintain in the kernel. Therefore this mechanism has been removed from it.

The latest solution is *udev*. This mechanism allows Linux users to have a dynamic */dev* directory and it provides the ability to have persistent device names. It uses *sysfs* and ***/sbin/hotplug*** and runs entirely in userspace.
When populating */dev*, *udev* decides which nodes to include, and how to name them, by reading a series of rule files included in */etc/udev/rules.d/* directory.

Default *udev* rules are stored in */etc/udev/rules.d/udev.rules*. This file should not be modified. It contains default rules for all Linux systems. The rules files are parsed in lexical order. *udev* will stop processing rules as soon as it finds a matching rule in a *.rules file for the new item of hardware that has been detected. Therefore it is important that own rules get processed before the *udev* defaults, otherwise they will not take effect.
Details concerning *udev* configuration and operating may be found on *udev* project Internet site: http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.htm

The Linux GPL Root Filesystem starting from version 0103 uses the *udev* solution for device files handling. It contains the common Linux *udev* rules as well as the rules specific for the Kontron E²Brain boards. The specific rules are put in the file */etc/udev/rules.d/kontron.rules*.

20Additional Libraries in directory addl_libs

The chapter 10 My First Linux Project mentioned, that the script ***initproject.sh*** creates a directory name ***addl_libs***. This additional library directory contains libraries, which are derived from other open source projects, e.g. pciutils and sysfsutils.

These libraries can be used within own applications, if the following precautions are met:

- The library versions match with the versions, which are provided together with the Kontron GPL root file system.
  If you are using a different kind of root file system, e.g. a root fs, which was provided by a commercial embedded Linux distribution, it might happen, that these libraries and headers are not compatible with this root file system. In this situation the libraries and headers of this distribution shall be used instead
- The lm_sensors package within this BSP also links against the libsysfs within this addl_libs directory.
  If you are using a different kind of root file system, e.g. a root fs, which was provided by a commercial embedded Linux distribution, you might run into compatibility problems. In this case the libraries and headers of this distribution should be used instead.

Information for Advanced Users

For more informations regarding the libraries contained in addl_libs, and for their usage, please refer to the project home pages of sysfsutils and pciutils, or look into the Kontron GPL root fs source package (where this packages are contained as full tarballs)

© 2007 Kontron Modular Computers

21The LM-Sensors package (optional)

Some BSP's provide the LM-Sensors package for system monitoring. This package is integrated into the BSP in a way, that it can be easily cross-compiled for the particular supported Kontron HW (see 10.4 Compile the BSP components).

And, the kernel is already configured for LM-Sensors support for all devices contained on this board.

However, LM-Sensors is a mighty package with a rich feature set.To create own applications using the LM-Sensors API, please refer to the online documentation contained within the lm_sensors source tree within the project working directory, or look into the LM-Sensors homepage at http://secure.netroedge.com/~lm78/index.html

22Appendix A: Notes on RPM

RPM stands for the Red Hat Package Manager. This is an open packaging system available for anyone to use. RPM packages can be easily installed and tracked and the source can be easily reconstructed. It also maintains a database of all packages and their corresponding files that can be used for verifying packages and to obtain information about files and packages.

The following is a quick primer for Red Hat's packaging system, RPM. To the RPM it is necessary to be the root user. In general, normal usage of the **rpm** command can be summarized as follows:

Package Installation and Removal:

To install a package, issue the command:

rpm -ivh [filename]

To upgrade a package, type:

rpm -Uvh [filename]

To remove a package, type:

rpm -e [package name]

In order to upgrade or install some packages, additional flags may be required. Use these commands only if it is understood what they mean and how they can affect the system.

Some examples are: **--force** (which will overwrite files that are owned by other packages), and **--nodeps** (which will perform the installation even if the package needs other packages that are not present).

Querying Packages:

To determine if a package is installed, type:

rpm -q [package name]

To obtain information about an installed package, type:

rpm -qi [package name]

To list the files belonging to a package, type:

rpm -ql [package name]

To determine a package in which a file is located, type:

rpm -qf [path-to-filename]

One can usually join various query commands together. For instance, **rpm -qil** will give information and list all the files in the package.

To view the contents of an RPM that is not installed, add **p** to the query line:

rpm -qilp somepackage.1.1-4.i386.rpm

This will list the information and the files contained in the file: somepackage.1.1-4.i386.rpm.

Verification:

RPM can also be used to see what files on the system may have changed from their initial settings. To check all installed packages, type:

rpm –Va

This will summon a list of all files that have changed since the packages were installed. This can be a large number of files. To see only what packages have changed so that they can be verified individually, type:

rpm -Va | awk "{print $2}" | xargs rpm -qf | sort -u &> /tmp/file1

Then look in the file **/tmp/file1** for the list of packages that have changed:

More Advanced Usage:

More advanced usage procedures can be found in the documentation for RPM and in the documentation on the RPM web site:

www.rpm.org

In addition, "*Maximum RPM",* by Edward C. Bailey, is another great source for detailed information about the many powerful options available with RPM:

www.redhat.com/support/books/max-rpm/max-rpm-html/index.html

23Appendix B: Configuring Minicom

To configure minicom, follow these steps:

a) Log in as root, and run:

 # minicom -s

The minicom configuration menu appears:

b) Select serial port set up and set the options for the target

A - Serial Device : *dev/ttySx (select the port of the host)*
B - Lockfile Location : */var/lock*
E - Bps/Par/Bits : *9600 8N1*
F - Hardware Flow Control : No
G - Software Flow Control : No

Press Enter to return to the main configuration menu.

d) Select modem and dialling

Erase the values for:

A - Init string,
B - Reset string, and
K - Hang-up string.

Press Enter to return to the main configuration menu.

e) Save values and exit

Select Save as dfl to save these as the default settings.
Select Exit from Minicom.

f) Go back to normal user mode

Now leave the root login. From a users command prompt enter

 # minicom

NOTE!

Usage of minicom may require certain permissions. On some distributions it might be necessary to be a member of the **uucp** group to access serial ports. Refer to the manual of the Linux distribution or query the system administrator.

24Appendix C: XScale IXP42x based eBrain specific considerations

24.1Using built in processor Ethernet Interfaces

The E²Brain family of Kontron embedded computers includes a few boards built on the Intel IXP42x processors (XScale). These are E²Brain 425 and E²Brain 420.
The XScale embedded processors base on the ARM 5 processor core and they include a lot of additional functional blocks.

All network processors in the Intel IXP4XX product line have a unique distributed processing architecture that speeds development for a range of applications. Each processor combines a high-performance Intel XScale core with additional Network Processor Engines (NPE) to achieve wire-speed packet processing performance.

The NPEs functional blocks require special software modules to operate. This software is available on Intel Internet site to download. Unfortunately the building of required software modules is not trivial. It requires some additional activities to compile it against the 2.6.x Linux kernel versions.

24.1.1 Downloading the software

The required Intel software is available at:
http://www.intel.com/design/network/products/npfamily/ixp400_archives.htm

These are the required packages:

- Intel Hardware Access Software ver. 2.0 (***IPL_ixp400AccessLibrary-2_0.zip***)
- NPE microcode ver. 2.0 (***IPL_ixp400NpeLibrary-2_0_5.zip***)

Unfortunately the packages above are not suitable for Linux kernel 2.6.x, but there is a solution to allow their proper compilation: the SnapGear Embedded Linux patch.
The SnapGear patch for the Intel Access Library may be downloaded from:
http://www.snapgear.org/snapgear/downloads.html by using the link on this site named:
"***SnapGear IXP400 Access Library patch [shar]***"

This patch is valid only for Intel software version 2.0!

The downloaded file should be:

- ***snapgear-modules-20051115.sh***

 © 2007 Kontron Modular Computers

24.1.2 Compiling the software modules

In fact the SnapGear patch is intended to be used with SnapGear Embedded Linux. But it is also possible to compile the software outside of the SnapGear distribution. By doing this, two additional files are required:

- ***autoconf.h*** – header file containing the configuration of Intel Hardware Access Library
- ***.config*** – containing configuration for build environment

The compilation requires several steps:

1. create the SnapGear build environment
2. uncompress Intel software
3. patch the Intel software with SnapGear patches
4. include the provided ***autoconf.h*** and ***.config*** files into build environment
5. configure the kernel sources for ARM XScale architecture
6. compile.

These steps look simple, but a lot of mistakes may happen when performing them. Therefore the Kontron BSP for E²Brain 42x contains a proper structure and a script to build the software modules successfully.

In the installed BSP under directory ***3rd_party_drivers/*** the following files can be found:

- README.txt – short description where to download the packages and how to compile the software
- ***config.example*** – configuration for build environment
- ***autoconf.h*** – configuration for Intel Hardware Access Library

In the ***scripts/*** subdirectory of the BSP the script ***make3rd_party.sh*** can be found. This script performs all steps listed above. The only required activity from the user is to download the required software packages and put them into the ***3rd_party_drivers/*** directory of the BSP. Note that the downloaded packages may be copied to the project directory if it has been created (as described in chapter 10. My First Linux Project) – this is the preferred way.

After the packages are put in the ***3rd_party_drivers/*** directory, the ***make3rd_party.sh*** script may be invoked from the project directory with the project directory path as a parameter e.g.:

```
usr@x86host: /bsp/project> sh ./scripts/make3rdparty.sh
/home/usr/bsp/project/
```

The build process requires the **configured** kernel sources!

24.1.3 Using the created modules

After invoking the script ***make3rd_party.sh*** three files have been created:

- ***ixp400.ko*** – the Intel Hardware Access Library kernel module
- ***ixp400_eth.ko*** – the Intel Ethernet ports kernel module
- ***IxNpeMicrocode.dat*** – the microcode for NPE functional blocks

These files must be downloaded to the target filesystem root directory (“/”).
Using these modules requires additional activities on the target root filesystem:

```
# mknod /dev/ixNpe c 241 0
# insmod ixp400.ko
```

```
# cat IxNpeMicrocode.dat > /dev/ixNpe
# insmod ixp400_eth.ko
```

These commands have been integrated into the starting scripts of theLinux GPL Root Filesystem. These scripts assume that the modules listed above exist in the root directory of the target root filesystem.

Now the Ethernet interfaces are started and running but they are not configured.
Before using them, they should be configured using ***ifconfig*** linux tool.
The command

```
~ # ifconfig -a
dummy0    Link encap:Ethernet  HWaddr F6:03:38:D4:27:EB
          BROADCAST NOARP  MTU:1500  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)

eth0      Link encap:Ethernet  HWaddr 00:02:B3:01:01:01
          BROADCAST MULTICAST  MTU:1500  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:256
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

eth1      Link encap:Ethernet  HWaddr 00:02:B3:02:02:02
          BROADCAST MULTICAST  MTU:1500  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:256
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

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)

~ #
```

shows all available interfaces.

However they do not have the IP addresses assigned. Following example will configure the ETH0 interface with IP address 192.168.3.139:

```
~ # ifconfig eth0 192.168.3.139 up
~ # ifconfig -a
dummy0    Link encap:Ethernet  HWaddr F6:03:38:D4:27:EB
          BROADCAST NOARP  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
```

 © 2007 Kontron Modular Computers

```
          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)

eth0      Link encap:Ethernet  HWaddr 00:02:B3:01:01:01
          inet addr:192.168.3.139  Bcast:192.168.3.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:256
          RX bytes:60 (60.0 b)  TX bytes:0 (0.0 b)

eth1      Link encap:Ethernet  HWaddr 00:02:B3:02:02:02
          BROADCAST MULTICAST  MTU:1500  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:256
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

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)

~ #
```

Now the interface eth0 is configured and connection to the network should work. This can be verified using e.g. ***ping*** tool:

```
~ # ping 192.168.3.13
PING 192.168.3.13 (192.168.3.13): 56 data bytes
64 bytes from 192.168.3.13: icmp_seq=0 ttl=64 time=5.2 ms
64 bytes from 192.168.3.13: icmp_seq=1 ttl=64 time=0.3 ms
64 bytes from 192.168.3.13: icmp_seq=2 ttl=64 time=0.4 ms
64 bytes from 192.168.3.13: icmp_seq=3 ttl=64 time=0.3 ms
```

24.2ARM XScale Linux Kernel

Linux kernel supports various processor architectures. These architectures differ in the set of options presented to the user. Therefore some of the kernel configuration menus may differ between kernel configured ex. for PPC and ARM processors. Also the details regarding kernel compilation process may differ in both cases.

24.2.1Kernel compiling

1. The result of ARM XScale Linux kernel compilation is put in the *<arch>/arm/boot/* directory.
2. The kernel is always compiled to the *zImage* file.
3. The ARM Linux kernel compiling is always performed by *make* (without target parameters) call in Kernel source directory. The created *zImage* type (only kernel, kernel with initial ramdisk) depends on the sources configuration.

24.2.2Kernel Command Line

The kernel command line for ARM architecture is configured in the menu:

```
Boot options -->
     () Default kernel command string
```

24.2.3 INITRD with ARM Linux Kernel

The chapter 14 The Initial Ramdisk INITRD describes how to create the initial ramdisk. However when creating the initial ramdisk for ARM architecture following remarks must be taken into account:

1. The ramdisk image name should be *ramdisk* instead of *ramdisk.image*
2. The created ramdisk image should be copied to: *<arch>/arm/boot/compressed* directory

24.3 NFS mounted Root file system

The E[2]Brain 425/420 include two Ethernet Interfaces on board. They are provided by XScale IXP425 processor. To use these interface proper kernel module drivers must be loaded into memory. Because of the license issues these drivers can not be compiled into the kernel.

Unfortunately to mount the Root file system, the kernel must have access to the network during boot process. This requirement may not be fulfilled without additional PCI Ethernet card.

The E[2]Brain 42x evaluation kit contains the Ethernet PCI extension card. The default configuration for the kernel provided within the E[2]Brain 42x BSP already contains the proper drivers.

Chapter 12.6 ys required for booting guards through the NFS server setup and kernel command line configuration.

 © 2007 Kontron Modular Computers