

# OpenRISC

## User Manual

Edition: April 2008



Vision Systems GmbH

Tel: +49 40 528 401 0

Fax: +49 40 528 401 99

Web: [www.visionsystems.de](http://www.visionsystems.de)

Support: [service@visionsystems.de](mailto:service@visionsystems.de)

---

The software described in this manual is furnished under a license agreement and may be used only in accordance with the terms of that agreement.

## **Copyright Notice**

Copyright © 2008 Vision Systems. All rights reserved. Reproduction without permission is prohibited.

## **Trademarks**

VScom is a trademark of Vision Systems GmbH. All other trademarks and brands are property of their rightful owners.

## **Disclaimer**

Vision Systems reserves the right to make changes and improvements to its product without providing notice.

Vision Systems provides this document “as is,” without warranty of any kind, either expressed or implied, including, but not limited to, its particular purpose. Vision Systems reserves the right to make improvements and/or changes to this manual, or to the products and/or the programs described in this manual, at any time.

Information provided in this manual is intended to be accurate and reliable. However, Vision Systems assumes no responsibility for its use, or for any infringements on the rights of third parties that may result from its use.

This product might include unintentional technical or typographical errors. Changes are periodically made to the information herein to correct such errors, and these changes are incorporated into new editions of the publication.

## Contents

<b>1. Introduction</b>	<b>7</b>
<b>2. Getting Started</b>	<b>8</b>
2.1. Connect to OpenRISC via Serial Link	8
2.2. Configure Network	8
2.3. Start Programming	9
<b>3. Hardware Configuration</b>	<b>11</b>
3.1. Power Supply	11
3.2. Network	11
3.2.1. WLAN Antenna	11
3.2.2. WLAN Configuration	11
3.2.3. Ethernet	11
3.3. RS422/485 Electrical Configuration	12
3.3.1. Termination Resistors	12
3.3.2. BIAS Function	12
<b>4. Software Configuration</b>	<b>13</b>
4.1. Swapping and Logging	13
4.2. Activating and Deactivating Services	14
4.3. Base Image	14
4.3.1. Program Overview	14
4.3.2. GCC	15
4.3.3. Netcat	15
4.3.4. Socat	15
4.4. Complete Image	15
4.4.1. Program Overview	15
4.4.2. Samba	16
4.4.3. Web Server (Apache2.2)	16
4.4.4. Mail Server (courier)	17
4.4.5. NTP	18
4.4.6. sreddird	18
4.4.7. WindowMaker	19
4.4.8. Connecting to X-Window	19
4.4.9. AbiWord	20
<b>5. Network Services and Tools Provided by OpenRISC</b>	<b>21</b>
5.1. wireless-tools	21
5.2. Bluetooth Support	22
5.3. Telnet	23
5.4. SSH	23
5.5. FTP	24
5.6. Other Possible Services	24
5.6.1. Samba	24
5.6.2. NFS	24
<b>6. Software Development</b>	<b>25</b>
6.1. Environment	25

6.2. Linux Kernel . . . . .	26
6.3. Setup Shared Source Directory . . . . .	27
6.4. Debugging . . . . .	28
6.5. OpenRISC Hardware API . . . . .	29
6.5.1. Digital I/O . . . . .	29
6.5.2. Buzzer . . . . .	30
6.5.3. LEDs . . . . .	30
6.5.4. Serial Interfaces . . . . .	31
<b>7. /proc-Extensions for the OpenRISC</b>	<b>33</b>
<b>8. BIOS</b>	<b>34</b>
8.1. System Console . . . . .	35
8.2. Boot Priority . . . . .	35
8.3. Configure Network Shares . . . . .	36
8.4. Configure Network Parameter . . . . .	37
8.5. Configure Miscellaneous Parameter . . . . .	38
8.6. Edit Bootscript . . . . .	39
8.7. BIOS Update . . . . .	40
8.8. View Board Information . . . . .	41
8.9. Hardware Test . . . . .	41
8.10. Default Parameter . . . . .	41
8.11. Exit . . . . .	41
<b>9. Debian Installation</b>	<b>42</b>
9.1. Preparing Boot Image Files . . . . .	42
9.2. Starting Debian Installer . . . . .	43
9.3. Choosing CD-ROM . . . . .	44
9.4. Load Installer Components from CD . . . . .	44
9.5. Network Configuration . . . . .	44
9.6. Partition Disks . . . . .	45
9.7. Setting Passwords . . . . .	46
9.8. Install the Base System . . . . .	46
9.9. Configure the Package Manager . . . . .	46
9.10. Software Selection . . . . .	46
9.11. Finish the Installation . . . . .	47
<b>A. Debian Maintenance Notes</b>	<b>48</b>
A.1. Debian Package Management . . . . .	48
A.2. Keep a Track on Disk Usage . . . . .	49
<b>B. Hardware Test Utility: hwtest</b>	<b>50</b>
B.1. Network Test . . . . .	51
B.2. MiniPCI Test . . . . .	51
B.3. Serial Test . . . . .	51
B.4. GPIO Test . . . . .	52
B.5. CompactFlash Test . . . . .	52
B.6. I2C and RTC Test . . . . .	53
B.7. EPLD Test . . . . .	53
B.8. USB Test . . . . .	53

B.9. Build Notes . . . . .	54
<b>C. VS Image Tool: VSImgTool</b>	<b>55</b>
<b>D. Eclipse</b>	<b>56</b>
D.1. Installation Notes . . . . .	56
D.2. Debugging . . . . .	57
<b>E. DVDs Content</b>	<b>59</b>
<b>Index</b>	<b>60</b>

## List of Figures

1. Compilation example . . . . .	10
2. Kernel Configuration Menu . . . . .	27
3. Insight: target selection . . . . .	29
4. BIOS: main menu . . . . .	34
5. BIOS Prompt for Bootlist . . . . .	34
6. BIOS: System Console . . . . .	35
7. BIOS: Boot Priority . . . . .	35
8. BIOS: Configure Network Shares . . . . .	36
9. BIOS: Configure Network Parameter . . . . .	37
10. BIOS: Configure Miscellaneous Parameter . . . . .	38
11. BIOS: Edit Bootscript . . . . .	39
12. BIOS: Update . . . . .	40
13. BIOS: Update (Images) . . . . .	40
14. BIOS: Update (Reboot System) . . . . .	40
15. BIOS: View Board Information . . . . .	41
16. Windows Share . . . . .	43
17. Bootlist . . . . .	43
18. Low Memory Mode . . . . .	44
19. Detect and mount CD-ROM . . . . .	44
20. Network configuration . . . . .	44
21. Partition Overview . . . . .	45
22. Ext2 Partition Settings . . . . .	45
23. Software Selection . . . . .	46
24. Continue without Boot Loader . . . . .	47
25. Eclipse debug: Main tab . . . . .	57
26. Eclipse debug: Debugger tab . . . . .	58
27. Eclipse debug: Commands tab . . . . .	58

## List of Tables

1. RS422/485 Jumper Configuration . . . . .	12
2. Images for 512Mb and 1Gb CompactFlash Cards . . . . .	13
3. Enable/disable services . . . . .	14
4. Tool chain items . . . . .	25

5.	BIOS: hwtest parameters . . . . .	41
6.	Common parameters and test modules . . . . .	50
7.	Customized null-modem cable layout . . . . .	51
8.	GPIO Pin Connections . . . . .	52
9.	VSIImgTool call parameters . . . . .	55

## 1. Introduction

The OpenRISC is an ARM9-based RISC industrial embedded computer. The great variety of interfaces like LAN, CF, USB, I<sup>2</sup>C, serial interface and digital I/O makes it easy to connect various industrial devices to the OpenRISC.

Compact dimensions and DIN Rail mounting capability make the OpenRISC to a space saving and flexible mounting industrial computer. It is feasible to be installed even in space limited environments.

Due to RISC based architecture the OpenRISC has very small power consumption (6,5 Watt), so fanless heat dissipation is possible. Working in an extended temperature range from -10°C up to 65°C the OpenRISC can be used under harsh industrial conditions. Therefore the OpenRISC is downright designed for industrial automation.

The embedded computer runs full-featured Debian GNU/Linux on ARM operating system. With Debian's repository database it is easy to install and update the free software on the OpenRISC. The OpenRISC is capable to act directly as software development host, Web, Mail, Print and Database server or as desktop computer with X11 window manager and many more.

See the list of features below:

- ARM9 32-bit RISC CPU, 166MHz
- 64MB SDRAM on board
- 4MB Flash Disk on board
- 2 x RS232/RS422/RS485 serial ports
- 8 independent digital I/O channels
- 1 x CF-Slot in True IDE mode (accepts Microdrives)
- 2 x USB 2.0 as Host
- MiniPCI-slot for expansion with WLAN, GPS etc.
- 2 x Ethernet interfaces for redundant networking or routing functions
- I<sup>2</sup>C bus with max. 330kHz clock
- RTC
- Ready-to-Run Debian Linux for ARM operating system
- DIN-Rail and wall-mount installation
- Robust, fanless design
- Wide temperature range -10 to 65°C
- Buzzer, Watch Dog Timer

## 2. Getting Started

### 2.1. Connect to OpenRISC via Serial Link

Connect the OpenRISC to the serial port of your PC and start a terminal software (Hyperterminal, ZOC<sup>1</sup> etc) with 115200,8,n,1 settings. Insert a CF-card with one of the preinstalled systems (refer to Section 4). Power your OpenRISC according to Section 3.1. You'll see Linux booting. After the boot procedure you'll be asked to log in. Two users are already added to the system: the super user (root) and an ordinary user (user). For the super user enter:

```
Debian login: root
Password: linux
```

For the ordinary user enter:

```
Debian login: user
Password: user
```

### 2.2. Configure Network

Now you can configure network interfaces by editing `/etc/networking/interfaces`. The IP addresses for `eth0`, `eth1` and `ra0` are by default statically assigned (see `interfaces` file below).

```
# This file describes the network interfaces available on your system
# and how to activate them.  For more information, see interfaces(5).
# The loopback network interface
auto lo
iface lo inet loopback
# The primary network interface
auto eth0
iface eth0 inet static
address 192.168.254.254
netmask 255.255.255.0
# The secondary network interface
auto eth1
iface eth1 inet static
address 192.168.253.254
netmask 255.255.255.0
#wireless interface
auto ra0
iface ra0 inet static
address 192.168.127.254
netmask 255.255.255.0
pre-up ifconfig ra0 up
pre-up iwconfig ra0 mode ad-hoc essid subpc channel 7
post-up echo BLUE > /proc/vsopenrisc/leds
post-down echo blue > /proc/vsopenrisc/leds
```

---

<sup>1</sup>[www.emtec.com](http://www.emtec.com)



`post-up` and `post-down` directives switch blue LED on and off signaling that wireless interface is up or down. The wireless interface `ra0` will be configured with the `wireless-tools` (see Section 5.1) distribution.

### 2.3. Start Programming

Connect to the OpenRISC either via serial link or network and login as `user`. For the introduction some examples were prepared and placed under `/home/user/examples`. This folder contains following files:

- `ioctls.c` - LEDs, buzzer, reset push button and digital IO usage examples
- `ioctls2.c` - UART and network usage examples
- `rawsrv.c` - raw server application that transfers data from network to serial interface and vice versa
- `vsopenrisc.h` - OpenRISC hardware API header file
- `Makefile` - the makefile to produce examples. Following targets can be created:
  - `all` - creates `ioctls`, `ioctls2` and `rawsrv` executables
  - `doc` - creates doxygen documentation in `doc` folder
  - `clean` - deletes executables
  - `distclean` - executes `clean` and in addition removes `doc` folder
- `vsopenrisc.doxyfile` - doxygen configuration file

Execute `make` and you'll get three above mentioned executables (see Figure 1).

You can start with an `ioctls` example that reads and changes the LEDs (Power LED, WLAN LED etc.), reads the reset push button status and digital IO registers:

```
./ioctls
```

After that you'll see your LEDs blinking and explaining outputs on your terminal. For further information about software development for the OpenRISC refer to Section 6.

```
user@debian:~/examples$ make
gcc -O0 -g3 -Wall ioctls.c -o ioctls
gcc -O0 -g3 -Wall ioctls2.c -o ioctls2
gcc -O0 -g3 -Wall rawsrv.c -o rawsrv
user@debian:~/examples$ ls -l
total 350
-rw-r--r-- 1 user user 349 Aug 27 14:32 CMakeLists.txt
-rw-r--r-- 1 user user 392 Aug 27 14:57 Makefile
-rwxr-xr-x 1 user user 68577 Aug 27 17:12 ioctls
-rw-r--r-- 1 user user 8865 Aug 27 14:54 ioctls.c
-rwxr-xr-x 1 user user 8568 Aug 27 17:12 ioctls.strip
-rwxr-xr-x 1 user user 91036 Aug 27 17:12 ioctls2
-rw-r--r-- 1 user user 3097 Aug 14 14:01 ioctls2.c
-rwxr-xr-x 1 user user 5352 Aug 27 17:12 ioctls2.strip
-rwxr-xr-x 1 user user 93286 Aug 27 17:12 rawsrv
-rw-r--r-- 1 user user 5682 Aug 14 14:01 rawsrv.c
-rwxr-xr-x 1 user user 8720 Aug 27 17:12 rawsrv.strip
-rw-r--r-- 1 user user 50619 Aug 14 14:01 vsopenrisc.doxyfile
-rw-r--r-- 1 user user 2243 Aug 27 14:50 vsopenrisc.h
user@debian:~/examples$
```

Figure 1: Compilation example

## 3. Hardware Configuration

### 3.1. Power Supply

The OpenRISC device is powered by a single 9-30V power supply. A suitable power supply adapter is part of the packaging. Connect the cable to the power jack at the right side of OpenRISC, and put the adapter into the socket. The Power LED on OpenRISC (red) will light. You can connect a power supply of your choice, providing the technical requirements are met.

### 3.2. Network

The OpenRISC can use all three network interfaces (Ethernet and WLAN) independently. The default IP addresses are:

- 192.168.254.254 for eth0 (first network interface)
- 192.168.253.254 for eth1 (second network interface)
- 192.168.127.254 for ra0 (wireless network interface)

#### 3.2.1. WLAN Antenna

The connector used for the WLAN Antenna is known as SMA-Reverse. This is a standard type to allow for simple connection of different equipment. Just fit the supplied antenna by carefully screwing it to the connector. You are free to connect a cable and a different antenna of your choice, as long as it is designed for WLAN.

#### 3.2.2. WLAN Configuration

WLAN supports 802.11b/g standard with 54Mbit/s. When the interface goes up, the blue LED will light, when the interface goes down, the blue LED goes dark.

#### 3.2.3. Ethernet

The connectors for Ethernet are the usual RJ45. Simply connect it to your switch or hub. When the connect is done the Link LED on RJ45 (yellow) will light. When data traffic occurs on the network, this LED will blink. It depends on your network whether a 100Mbit or a 10Mbit connect will be established. A 100Mbit net causes the Speed LED on RJ45 (green) to light, otherwise it will remain dark.

### 3.3. RS422/485 Electrical Configuration

In typical RS 422 and RS 485 installations certain electric conditions have to be configured. Simply connecting cables is not enough to fulfil the specifications of RS 422 and RS 485.

For ease of installations the OpenRISC provides these functions for often used parameters. They are activated by placing certain jumpers (see Table 1), internal of the OpenRISC. There is one block of jumpers for each serial port (JP5 for Port 1 and JP6 for Port 2). Place a connection cap to activate the function.

**Warning:** All jumpers are unconnected by default. This is important for use in RS232 mode. Never close any jumper, otherwise communication errors or damage of devices is possible.

Pins	Function of Signals
1-2	Place 120 $\Omega$ to terminate Tx+/- (Data+/- in RS 485 2-wire)
3-4 5-6	Add BIASing function to Tx+/- (mostly required for RS 485 2-wire modes)
7-8	Place 120 $\Omega$ to terminate Rx+/-
9-10	Add BIASing function to Rx+/-
11-12	Add BIASing function to Rx+/-

Table 1: RS422/485 Jumper Configuration

#### 3.3.1. Termination Resistors

The use of long communication lines in RS 422 and RS 485 mode require the installation of termination resistors. These must match the impedance of the cable. Typical cables in Twisted-Pair configuration have an impedance around 120 $\Omega$ . In RS 422 this resistor has to be placed at the far end from the sender, in RS 485 the typical configuration requires one resistor at each end of the cable.

#### 3.3.2. BIAS Function

RS 485 requires a BIAS option for the communication lines. This will guarantee stable electrical levels on the cables, even at times when no station is transmitting data. Without BIAS there will be noise on the cable, and sometimes receivers can not detect the first characters of a beginning communication.

## 4. Software Configuration

The OpenRISC comes with a preinstalled Debian GNU/Linux on ARM<sup>2</sup> operating system. The customer can choose between two preconfigured images: base image and complete image. The latter image is based on the first image and includes all services and tools provided by it. The base image (see Section 4.3) provides the necessary tools and services to start with application development. The complete image (see Section 4.4) provides various services such as web and mail server, Samba server, etc. For office tasks it provides a X-Server with graphical desktop manager and some office software. Table 2 shows free and used space for each image:

Image	Used	Free
Base	213Mb	173Mb
Complete	411Mb	393Mb

Table 2: Images for 512Mb and 1Gb CompactFlash Cards

These images are to find on the DVD "Software" and can be copied to the CF card via `vsimgtool` utility (see Appendix C) under MS Windows or via `dd`.

### 4.1. Swapping and Logging

Due to the fact that the flash memory has a finite number of erase-write cycles it is very important to reduce them. Many applications use logging for information, recovery and debugging purposes, this can follow to frequent flash usage. There are several possibilities to avoid this:

1. use external HDD attached via USB and redirect swapping and logging to it
2. disable swapping (`swapoff -a`) and logging where it is possible
3. redirect the log stream via network

To redirect logged messages that need syslog edit `/etc/syslog.conf` and change all file destinations to network destinations (see the example). Change

```
*.=info;*.=notice;*.=warn;\nauth,authpriv.none;\ncron,daemon.none;\nmail,news.none -/var/log/messages
```

to

```
*.=info;*.=notice;*.=warn;\nauth,authpriv.none;\ncron,daemon.none;\nmail,news.none @192.168.254.84
```

for further information refer to the `syslog.conf` manpage. To receive the log messages under Linux you can use your existing syslog utility, for Windows you can use Kiwi Syslog Daemon<sup>3</sup> or any other Syslog daemon.

---

<sup>2</sup><http://www.debian.org/ports/arm/>

<sup>3</sup>[www.kiwisyslog.com](http://www.kiwisyslog.com)

## 4.2. Activating and Deactivating Services

Many services will be started as daemons at system startup and hence reduce the amount of free memory and increase the boot time. You can use one of the following options to deactivate the unused services at startup or start them only on demand:

- rename links under `/etc/rc2.d`<sup>4</sup> (see Table 3).
- start the service over `inetd`<sup>5</sup>
- start manually via `/etc/init.d`

Disabled name	Enabled name
K09apache2	S91apache2
K77ntp	S23ntp
K80courier-authdaemon	S20courier-authdaemon
K80courier-mta	S20courier-mta
K80courier-pop	S20courier-pop
K80samba	S20samba
K80sqwebmail	S20sqwebmail
K84openvpn	S16openvpn

Table 3: Enable/disable services

## 4.3. Base Image

The base image contains a minimal set of programs and libraries. Despite of its small size, it contains a development environment consisting of the gcc tool chain and vim-tiny text editor.

### 4.3.1. Program Overview

The base image provides among others the following utilities:

- Development
  - gcc-3.3
  - vim-tiny
- Network
  - ssh (server and client)
  - telnet (server and client)
  - vsftpd (server and client)
  - netcat
  - socat

---

<sup>4</sup>For detailed information about filenames see `/etc/rc2.d/README` file

<sup>5</sup>see `man inetd`

### 4.3.2. GCC

GCC<sup>6</sup> development is a part of the GNU Project, aiming to improve the compiler used in the GNU system including the GNU/Linux variant. The GCC development effort uses an open development environment and supports many other platforms in order to foster a world-class optimizing compiler, to attract a larger team of developers, to ensure that GCC and the GNU system work on multiple architectures and diverse environments, and to more thoroughly test and extend the features of GCC.

### 4.3.3. Netcat

Netcat<sup>7</sup> is a featured networking utility which reads and writes data across network connections, using the TCP/IP protocol. It is designed to be a reliable "back-end" tool that can be used directly or easily driven by other programs and scripts. At the same time, it is a feature-rich network debugging and exploration tool, since it can create almost any kind of connection you would need and has several interesting built-in capabilities.

### 4.3.4. Socat

socat<sup>8</sup> is a relay for bidirectional data transfer between two independent data channels. Each of these data channels may be a file, pipe, device (serial line etc. or a pseudo terminal), a socket (UNIX, IP4, IP6 - raw, UDP, TCP), an SSL socket, proxy CONNECT connection, a file descriptor (stdin etc.), the GNU line editor (readline), a program, or a combination of two of these. These modes include generation of "listening" sockets, named pipes, and pseudo terminals.

## 4.4. Complete Image

The complete image is designed to use the OpenRISC as a server and/or desktop system to accomplish such tasks as mail server, web server, resource sharing and office tasks. For the latter WindowMaker is installed to provide graphical desktop on the OpenRISC.

### 4.4.1. Program Overview

In addition to the programs provided by the base image, following programs are contained in the complete image:

- Network:
  - Samba client and server
  - Apache2 web server
  - Courier mail server
  - NTP client and server

---

<sup>6</sup><http://gcc.gnu.org/gccmission.html>

<sup>7</sup><http://netcat.sourceforge.net/>

<sup>8</sup><http://www.dest-unreach.org/socat/>

- sreddird RFC2217 server
- Desktop:
  - WindowMaker desktop
  - xdm server
  - AbiWord
  - dillo web browser

#### 4.4.2. Samba

Samba<sup>9</sup> is an Open Source/Free Software suite that has, since 1992, provided file and print services to all manner of SMB/CIFS clients, including the numerous versions of Microsoft Windows operating systems. Samba is freely available under the GNU General Public License.

To share an extra directory for users create this directory:

```
mkdir /samba
Edit /etc/samba/smb.conf:
[global]
workgroup = debian
netbios name = debianserver
server string = %h server (Samba %v)
log file = /var/log/samba/log.%m
max log size = 1000
syslog = 0
[SAMBA]
path=/samba
browseable=yes
writeable=yes
valid users = user
admin users = debian
```

Now you need to restart the samba to take the new changes effect:

```
/etc/init.d/samba restart
```

#### 4.4.3. Web Server (Apache2.2)

The Apache HTTP Server Project<sup>10</sup> is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows NT. The goal of this project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards. Apache has been the most popular web server on the Internet since April 1996.

In addition to the Apache2 packages the openssl package is installed to enable a SSL certificate creation. Apache2 is already equipped with SSL certificate and is ready to accept https connections needed for example to configure mail server via webadmin tool (see 4.4.4) . To create your own certificate or to install the official certificate take a look at the following paragraph.

---

<sup>9</sup><http://us3.samba.org/samba/>

<sup>10</sup><http://httpd.apache.org/>



**How to Configure SSL?** To enable https connections, a SSL certificate must be created and then registered by the Apache2 web server:

1. `mkdir /etc/apache2/ssl/`
2. `openssl req $@ -new -x509 -days 365 -nodes -out /etc/apache2/ssl/apache.pem -keyout /etc/apache2/apache.pem`
3. `chmod 600 /etc/apache2/ssl/apache.pem`
4. `cp /etc/apache2/sites-available/default /etc/apache2/sites-available/ssl`
5. `ln -s /etc/apache2/sites-available/ssl /etc/apache2/sites-enabled/ssl`
6. ssl file must be changed:  
`NameVirtualHost *:443`  
`<VirtualHost *:443>`  
`SSLEngine On`  
`SSLCertificateFile /etc/apache2/ssl/apache.pem`  
`SSLCertificateKeyFile /etc/apache2/apache.pem`  
`...`  
`</VirtualHost>`
7. to activate port 443 for https-Queries edit `/etc/apache2/ports.conf`. Add  
`Listen 443`
8. activate SSL module:  
`a2enmod ssl`
9. restart Apache:  
`apache2ctl restart`

#### 4.4.4. Mail Server (courier)

The Courier mail transfer agent (MTA)<sup>11</sup> is an integrated mail/groupware server based on open commodity protocols, such as ESMTP, IMAP, POP3, LDAP, SSL and HTTP. Courier provides ESMTP, IMAP, POP3, webmail and mailing list services within a single, consistent, framework. Individual components can be enabled or disabled at will. Courier now implements basic web-based calendaring and scheduling services integrated in the webmail module. Advanced groupware calendaring services will follow soon.

The mail server can be conveniently configured via web interface using courierwebadmin tool. It can be started with the following URL

`https://mail-server-address/cgi-bin/courierwebadmin`.

Execute webadmin-tool, setup server name and local domains under *Mail server name and local domains* menu. Then create a user:

```
adduser --ingroup users testuser
```

To create user's maildir-folder structure login as testuser and execute following commands:

```
cd /home/testuser
maildirmake Maildir
```

Execute webmail-tool and log on. From now on one can send and receive e-mails.

---

<sup>11</sup><http://www.courier-mta.org/>

#### 4.4.5. NTP

NTP<sup>12</sup> is a protocol designed to synchronize the clocks of computers over a network to a common timebase (usually UTC).

**Configuration Client** For client configuration edit `/etc/ntp.conf` as follows:

```
### client:/etc/ntp.conf #####
driftfile /var/lib/ntp/ntp.drift
# NTP-server in LAN
server 192.168.1.1
# Grant access from other NTP-server
restrict 192.168.1.1
# Grant access from localhost (ntpq -p)
restrict 127.0.0.1
# deny access for other peers
restrict default notrust nomodify nopeer
#####
```

Then restart `ntpd`.

**Configuration Server** For server configuration edit `/etc/ntp.conf` as follows:

```
### server:/etc/ntp.conf #####
driftfile /var/lib/ntp/ntp.drift
# NTP-server
server ptbtime1.ptb.de
server ptbtime2.ptb.de
# Grant access from other NTP-server
restrict ptbtime1.ptb.de
restrict ptbtime2.ptb.de
# Grant access from localhost (ntpq -p)
restrict 127.0.0.1
# grant access for local network
restrict 192.168.1.0 mask 255.255.255.0
# deny access for other peers
restrict default notrust nomodify nopeer
#####
```

Then restart `ntpd`.

#### 4.4.6. soredird

Soredird<sup>13</sup> is a serial port redirector that is compliant with the RFC 2217<sup>14</sup> "Telnet Com Port Control Option" protocol. This protocol lets you share a serial port through the network.

---

<sup>12</sup><http://www.ntp.org/>

<sup>13</sup><http://freshmeat.net/projects/soredird/>

<sup>14</sup><http://rfc.net/rfc2217.html>

**Configuration** `sredird` runs only under `inetd`. To configure `inetd` two files must be edited:

- the following line must be appended to `/etc/inetd.conf` to share `/dev/modem`:  
`sredir stream tcp nowait root /usr/sbin/tcpd /usr/sbin/sredird 5 /dev/modem /var/lock/LCK..modem`
- `sredir` service must be enabled in `/etc/services`:  
`sredir 705015/tcp`

Following clients could be used with `sredird`:

- Serial/IP COM Port Redirector (<http://www.tacticalsoftware.com/products/serialip.htm>) for Windows-Clients
- cyclades-serial-client (<http://freshmeat.net/projects/cyclades-serial-client/>) for Linux-Clients

#### 4.4.7. WindowMaker

Window Maker<sup>16</sup> is an X11 window manager originally designed to provide integration support for the GNUstep Desktop Environment. In every way possible, it reproduces the elegant look and feel of the NEXTSTEP[tm] user interface. It is fast, feature rich, easy to configure, and easy to use. It is also free software, with contributions being made by programmers from around the world. Window Maker includes compatibility options which allow it to work with other popular desktop environments, namely GNOME and KDE, and comes with a powerful GUI configuration editor, called WPrefs, which removes the need to edit text-based config files by hand

#### 4.4.8. Connecting to X-Window

The `xdm` manager<sup>17</sup> lets you connect to the X-Server. Following configuration issues should be considered to enable a remote access to the OpenRISC:

- `#* #any host can get a login window`  
in `/etc/X11/xdm/Xaccess` should be uncommented
- `DisplayManager.requestPort: 0`  
in `/etc/X11/xdm/xdm-config` must be replaced with  
`! DisplayManager.requestPort: 0`

Xming<sup>18</sup> can be used to connect to the X-Server from MS Windows. After installation, execute XLaunch and configure the proper IP address. The configuration can also be saved. The saved XML-file can look as follows:

```
<?xml version="1.0"?>
<XLaunch xmlns="http://www.straightrunning.com/XmingNotes"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.straightrunning.com/XmingNotes XLaunch.xsd"
```

---

<sup>15</sup>port 7050 is an example port number

<sup>16</sup><http://www.windowmaker.info/>

<sup>17</sup>[http://en.wikipedia.org/wiki/X\\_display\\_manager](http://en.wikipedia.org/wiki/X_display_manager)

<sup>18</sup><http://sourceforge.net/projects/xming>

```
WindowMode="Windowed"  
ClientMode="XDMCP"  
XDMCPHost="192.168.1.66"  
Display="0"  
Clipboard="true"/>
```

### 4.4.9. AbiWord

AbiWord<sup>19</sup> is a free word processing program, similar to Microsoft® Word. It is suitable for a wide variety of word processing tasks.

---

<sup>19</sup><http://www.abisource.com/>

## 5. Network Services and Tools Provided by OpenRISC

The OpenRISC can be accessed via Ethernet for remote usage and file sharing. For this purpose there are several services such as telnet, ssh and ftp installed and preconfigured. For WLAN configuration wireless-tools are included in the distribution.

### 5.1. wireless-tools

The Wireless Extension (WE) is a generic API allowing a driver to expose to the user space configuration and statistics specific to common Wireless LANs. The beauty of it is that a single set of tool can support all the variations of Wireless LANs, regardless of their type (as long as the driver support Wireless Extension). Another advantage is these parameters may be changed on the fly without restarting the driver (or Linux).

The Wireless Tools<sup>20</sup> (WT) is a set of tools allowing to manipulate the Wireless Extensions. They use a textual interface and are rather crude, but aim to support the full Wireless Extension. There are many other tools you can use with Wireless Extensions, however Wireless Tools is the reference implementation.

- **iwconfig** manipulate the basic wireless parameters
- **iwlist** allow to initiate scanning and list frequencies, bit-rates, encryption keys...
- **iwspy** allow to get per node link quality
- **iwpriv** allow to manipulate the Wireless Extensions specific to a driver (private)
- **ifrename** allow to name interfaces based on various static criteria

Here are some examples for the configuration of the wireless settings:

Example I: Config STA to link with AP which is OPEN/NONE(Authentication/Encryption)

1. `iwconfig ra0 mode managed`
2. `iwconfig ra0 key open`
3. `iwconfig ra0 key off`
4. `iwconfig ra0 essid "AP's SSID"`

Example II: Config STA to link with AP which is SHARED/WEP(Authentication/Encryption)

1. `iwconfig ra0 mode managed`
2. `iwconfig ra0 key restricted`
3. `iwconfig ra0 Key [1] "s:AP's wep key"`
4. `iwconfig ra0 key [1]`
5. `iwconfig ra0 essid "AP's SSID"`

Example III: Config STA to create/link as adhoc mode

1. `iwconfig ra0 mode ad-hoc`
2. `iwconfig ra0 key off`
3. `iwconfig ra0 essid "AP's SSID"`

Example IV: Config STA to link with AP which is WPAPSK/TKIP(Authentication/Encryption)

1. `iwconfig ra0 mode managed`
2. `iwpriv ra0 set AuthMode=WPAPSK`
3. `iwpriv ra0 set EncrypType=TKIP`
4. `iwpriv ra0 set WPAPSK="AP's wpa-preshared key"`
5. `iwconfig ra0 essid "AP's SSID"`

---

<sup>20</sup>[http://www.hpl.hp.com/personal/Jean\\_Tourrilhes/Linux/Tools.html](http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html)

Example V: Config STA to link with AP which is WPAPSK/AES(Authentication/Encryption)

1. `iwconfig ra0 mode managed`
2. `iwpriv ra0 set AuthMode=WPAPSK`
3. `iwpriv ra0 set EncrypType=AES`
4. `iwpriv ra0 set WPAPSK="AP's wpa-preshaed key"`
5. `iwconfig ra0 essid "AP's SSID"`

All examples mentioned above can be applied to the `/etc/network/interfaces` configuration file to be available on reboot. To do this insert `iwconfig`, `iwpriv` calls after `pre-up ifconfig ra0 up` (see Section 2.2).

### 5.2. Bluetooth Support

The Bluetooth support is already integrated in the kernel. To use a USB Bluetooth adapter you'll need to install some additional software:

```
apt-get install bluetooth
```

To connect to the Bluetooth network access point (NAP) execute:

1. `hcitool scan`  
you'll get the list of available Bluetooth devices with their addresses
2. select the needed one and connect to it  
`pand -c <bdaddr>`
3. check `/var/log/daemon.log` to verify that the connection is established or execute  
`ifconfig -a`  
if you can see `bnep0` interface the connection is definitely established
4. set up the `bnep0` interface. For example:  
`ifconfig bnep0 192.168.10.1`
5. assuming the NAP has an address 192.168.10.2 try to ping it:  
`ping 192.168.10.2`

To set up the NAP execute:

1. edit `/etc/bluetooth/hcid.conf` and set the security option to auto:  
# Security Manager mode  
# none - Security manager disabled  
# auto - Use local PIN for incoming connections  
# user - Always ask user for a PIN  
#  
`security auto;`
2. execute `/etc/init.d/bluetooth restart` to activate the new configuration
3. `hciconfig hci0 piscan lm master`  
to enable authentication (required by MS Windows) execute:  
`hciconfig hci0 piscan auth lm master`  
the default passkey configured in `/etc/bluetooth/hcid.conf` is "1234"
4. `pand --listen --role NAP`
5. after the client has connected, the `bnep0` must be also set up

To connect to the OpenRISC from MS Windows, use the on-board tools or the software supplied with the Bluetooth device. For the on-board tools:

1. select “Join a Personal Area Network”
2. search for the OpenRISC
3. select it and try to connect. Enter “1234” as passkey
4. configure the network interface

### 5.3. Telnet

The telnet console can be accessed with following command:

```
telnet ip address
```

This connection can be used only for normal users, not for the super user. To execute super user commands login as user and then execute `su`. For further information see:

```
man telnetd or man issue.net
```

### 5.4. SSH

To access the OpenRISC via SSH from Linux execute:

```
ssh ip address
```

To access OpenRISC via SSH from Windows you need a ssh-client such as PuTTY<sup>21</sup>. To exchange files several tools could be used:

- scp (Linux) - secure copy tool
- pscp (Windows) - secure copy tool included in PuTTY distribution
- WinSCP<sup>22</sup> (Windows) - secure copy tool with graphical interface

For further information see:

```
man sshd
```

---

<sup>21</sup><http://www.chiark.greenend.org.uk/~sgtatham/putty/>

<sup>22</sup><http://winscp.net/eng/index.php>

## 5.5. FTP

OpenRISC provides vsftpd<sup>23</sup> as FTP-server. The files to be shared must be placed in `/home/ftp`. To access the files execute:

```
ftp ip address
```

The FTP-server is configured for normal user accounts created on the OpenRISC, so login as user “user” with a password “user”. For access via graphical FTP-client following programs can be used:

- FileZilla (<http://www.filezilla.de/>)
- Firefox with FireFTP add-on (<http://fireftp.mozdev.org/help.html>)

For further information see:

```
man vsftpd
```

## 5.6. Other Possible Services

Other services such as Samba (see Section 4.4.2) or NFS could also be installed to share resources.

### 5.6.1. Samba

To install Samba execute:

```
apt-get install samba
```

To access your home directory from Windows, you must register the user in Samba:

```
smbpasswd -a user
```

For further information see:

```
man samba
```

### 5.6.2. NFS

To install the NFS-server execute:

```
apt-get install nfs-kernel-server
```

For further information see:

```
man nfsd
```

---

<sup>23</sup><http://vsftpd.beasts.org/>



## 6. Software Development

### 6.1. Environment

You can start programming directly on the OpenRISC. To modify files you can use vi editor. For more convenience you can also use a PC with Linux running on it. This can be directly installed Linux or Linux running in VMWare<sup>24</sup> Software for Windows users. For the compilation of your own applications outside of the OpenRISC you'll need a cross-compiler for the ARM platform. You find one on the DVD "Software" in the archive `toolchain_gcc.3.3.6_libc.2.3.6.tar.bz2`. Please decompress this archive into the `/` directory. To achieve this execute:

```
su
mount /dev/cdrom /mnt
cd /
tar -xvzf /mnt/development/toolchain_gcc.3.3.6_libc.2.3.6.tar.bz2
```

Following utilities prepent with `arm-linux-` will be available after extracting the files from the archive (refer to Table 4).

Name	Description
ar	Creates and updates static library files
as	Assembler
g++	C++ compiler
cpp	C preprocessor
gcc	C compiler
gdb	Debugger
insight	Frontend for gdb
ld	Linker
nm	Lists symbols from object files
objcopy	Copies and translates object files
objdump	Displays information about object files
ranlib	Generates indexes to archives (static libraries)
readelf	Displays information about ELF files

Table 4: Tool chain items

For an Integrated Development Environment following programs can be used:

- Eclipse ([www.eclipse.org](http://www.eclipse.org)). See Appendix D for installation and configuration notes
- Vim<sup>25</sup> ([www.vim.org](http://www.vim.org))
- many more

For ease of creating Makefiles the CMake<sup>26</sup> utility can be used. For that purpose the `CMakeLists.txt` file was added to the examples directory. After installing CMake on your OpenRISC directly or on your development host execute:

---

<sup>24</sup>The free VMWare Player can be downloaded under <http://www.vmware.com/>

<sup>25</sup>A good tutorial to start programming with Vim <http://heather.cs.ucdavis.edu/~matloff/ProgEdit/ProgEdit.html>

<sup>26</sup>[www.cmake.org](http://www.cmake.org)

```
cd /home/user/examples
cmake .
```

After that the Makefile is created. This Makefile has the same targets as the original one.

## 6.2. Linux Kernel

The OpenRISC uses 2.4.32 Linux kernel. The source code of this kernel can be obtained both from DVD "Debian" and from Subversion<sup>27</sup> repository at <http://svn.visionsystems.de/>. For the kernel archive on DVD execute:

```
su
mount /dev/cdrom /mnt
exit
cd /home/user/projects
tar -xvjf /mnt/development/linux-2.4.32-ks8695.tar.bz2
```

To check out the repository execute:

```
svn co svn://svn.visionsystems.de/linux-2.4.32-ks8695/trunk linux-2.4.32-ks8695/trunk
```

After extracting the kernel sources from the archive or checking out the repository execute

```
cd linux-2.4.32-ks8695/trunk
```

to get to the kernels source tree, where the main **Makefile** is placed.

To configure the kernel via text based graphical user interface Ncurses<sup>28</sup> library must be installed in the development version with all needed headers. After that execute:

```
make menuconfig
```

you'll see the following menu as shown in Figure 2. Change kernel configuration according to your needs and save the configuration. To compile the kernel for the first time execute:

```
make dep
```

```
make zImage
```

the next time you only need to execute

```
make zImage
```

The compiled kernel image **zImage** will be placed under

```
arch/arm/boot/zImage
```

To start the OpenRISC with new kernel do the following steps:

1. enter BIOS
2. enter "System Console"
3. configure IP address according to your network
4. execute `nc -l -p 5000 > /var/zImage` this will start Netcat listening on the port 5000

---

<sup>27</sup><http://subversion.tigris.org/>

<sup>28</sup><http://www.gnu.org/software/ncurses/ncurses.html>

5. on your development host execute `netcat "IP of the OpenRISC" 5000 < arch/arm/boot/zImage`
6. after the data transmission mount your CF and copy new kernel
7. `mount /dev/hda1 /mnt`
8. `cp /var/zImage /mnt/boot`
9. `umount /mnt`
10. reboot the OpenRISC

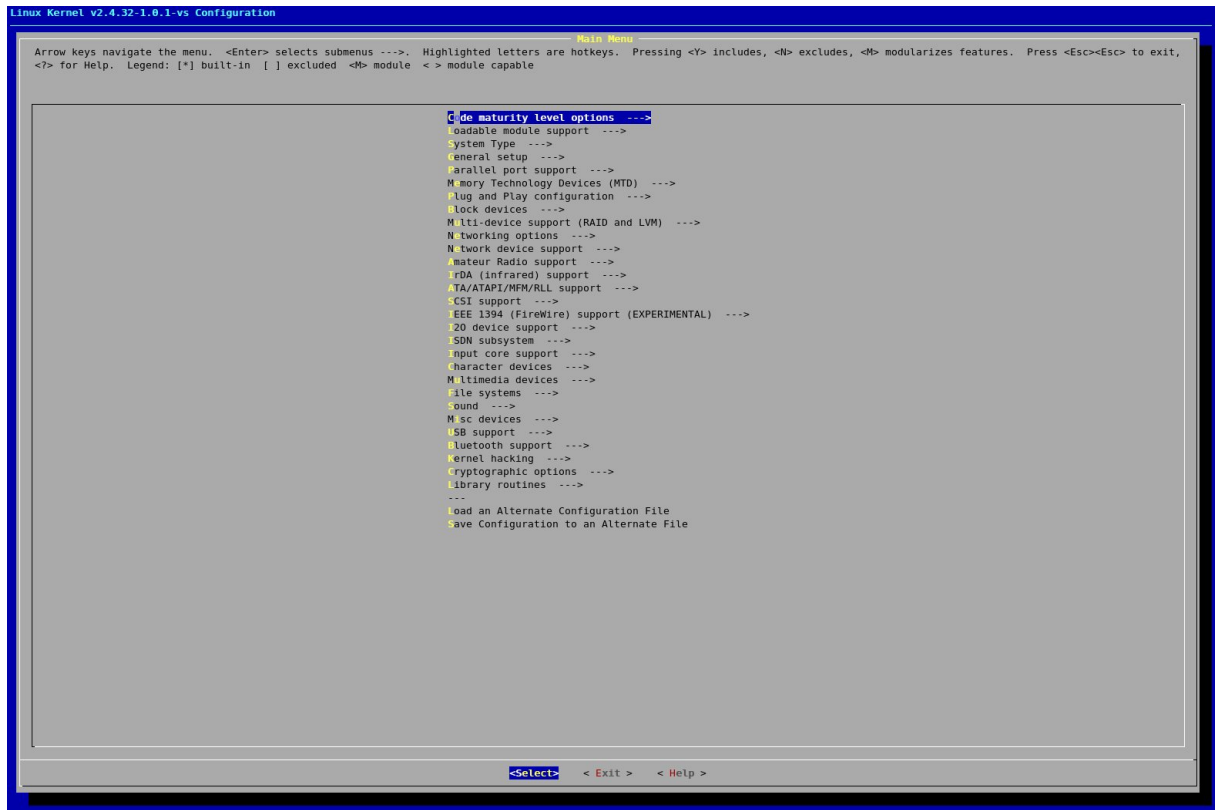


Figure 2: Kernel Configuration Menu

### 6.3. Setup Shared Source Directory

To avoid permanent copying the compiled files, you can share the examples folder on your PC via Samba<sup>29</sup>. Assuming your PC has the IP address 192.168.254.253 and a valid user like “user” from group “user” registered in the Samba-server do:

- On the PC side
  - `cd /home/user`
  - `mkdir openrisc_examples`
  - copy all files from `examples` folder to `openrisc_examples`

<sup>29</sup>Please make sure to install `smbfs` package: `apt-get install smbfs`

- edit as `su /etc/samba/smb.conf`. Add the `[openrisc]` share:  
    `[openrisc]`  
    `comment=OpenRISC programming examples`  
    `path=/home/user/openrisc_examples`  
    `browsable=yes`  
    `writable=yes`  
    `valid users=user`
- `/etc/init.d/samba restart`
- On the OpenRISC
  - login as user
  - `mkdir openrisc_examples`
  - `su`
  - `mount -t smbfs -o username=user,uid=user,gid=user //192.168.254.253/openrisc /home/user/openrisc_examples`
  - enter password
  - `exit`
  - `cd openrisc_examples`

Now you can edit and compile programs on the PC and execute them on the OpenRISC.

## 6.4. Debugging

You have the possibility to debug your own applications with the `gdbserver`<sup>30</sup> on the target. To debug your application, start the server with the following command:

```
gdbserver :9000 /home/user/examples/ioctls
```

Make a connection to the server with the insight debugger:

```
arm-linux-insight ioctls
```

Go to the menu **Target Settings** (see Figure 3) and enter your destination data. Than you can get connected to the target with **Run\Connect to Target**. The rest of the debugging is up to you.

You can also use Eclipse to debug your application (see Appendix D.2) .

---

<sup>30</sup>Please make sure to install `gdb` package: `apt-get install gdb`

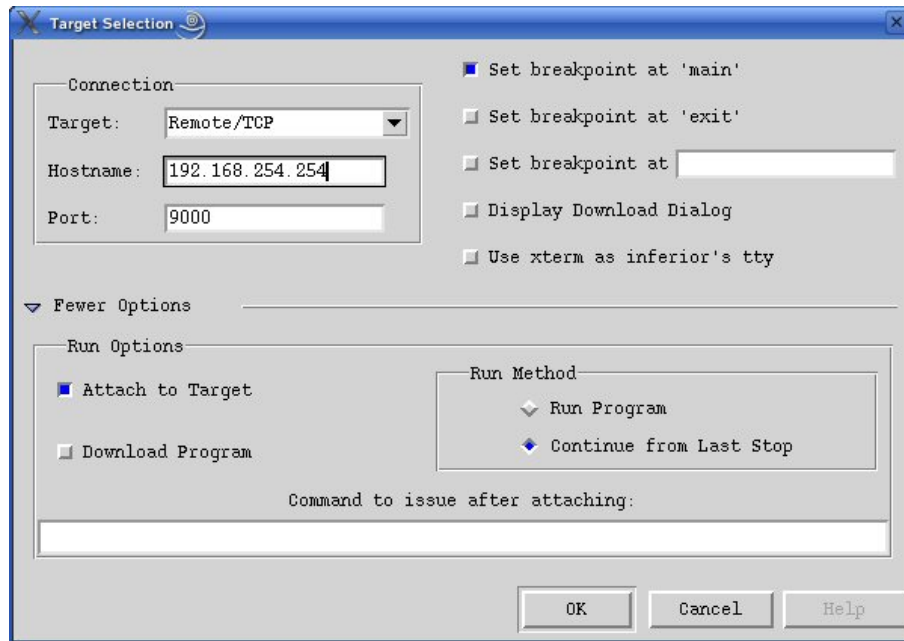


Figure 3: Insight: target selection

## 6.5. OpenRISC Hardware API

Such hardware as digital I/O, buzzer, serial interface will be controlled via IOCTL commands. These commands are defined in the `vsopenrisc.h` header file. Almost all IOCTL commands are reflected in the `/proc`-filesystem (see Section 7).

To control the hardware the `/dev/gpio` device must be opened:

Listing 1: Open `/dev/gpio`

```
int fd;

fd = open("/dev/gpio", O_RDWR);
if (fd < 0)
    exit(-1);
```

Some examples for the OpenRISC hardware are provided in the `Vscom/examples/ioctls.c` and `Vscom/examples/ioctls2.c`. For the usage in real applications see the source code of the Hardware Test Utility (`Vscom/hwtest` in the Linux kernel source provided on the DVD "Software").

### 6.5.1. Digital I/O

The OpenRISC provides 8 digital input/output channels. The data direction for each channel can be independently set to input (bit value 0) or output (bit value 1). An interrupt for an input channel can also be independently enabled (bit value 1 enables interrupt, bit value 0 disables interrupt for the desired channel). The physical driver operates with 32mA for both high and low level. For digital IO usage four registers are provided:

- data register - reflected in `/proc/vsopenrisc/gpio_data`

- data direction register - reflected in `/proc/vsopenrisc/gpio_ctrl`
- interrupt mask register - reflected in `/proc/vsopenrisc/gpio_irqmask`
- status register (read only) - reflected in `/proc/vsopenrisc/gpio_change`

In addition the interrupts will be counted for each pin.

Following IOCTL commands are defined in `vsopenrisc.h` to control digital I/O:

- `GPIO_CMD_GET/GPIO_CMD_SET` - set/get data register value
- `GPIO_CMD_GET_CTRL/GPIO_CMD_SET_CTRL` - set/get data direction register value
- `GPIO_CMD_GET_IRQMASK/GPIO_CMD_SET_IRQMASK` - set/get interrupt mask register value
- `GPIO_CMD_GET_CHANGE` - get status register value
- `GPIO_CMD_GET_CHANGES` - get interrupt count (reflected in `/proc/vsopenrisc/gpio_changes`)

The digital I/O driver also supports `select()` and `poll()` calls. To use this functionality the interrupt must be enabled for the desired pins.

### 6.5.2. Buzzer

The OpenRISC provides a buzzer for acoustic signaling. You can manipulate it via the IOCTL-calls or via the Proc-Filesystem. Following IOCTL commands are defined in the `vsopenrisc.h` to control the buzzer:

- `GPIO_CMD_GET_BUZZER/GPIO_CMD_SET_BUZZER` - turn on/off the buzzer (reflected in `/proc/vsopenrisc/buzzer`)
- `GPIO_CMD_GET_BUZZER_FRQ/GPIO_CMD_SET_BUZZER_FRQ` - get/set the modulated signal delay. So that the buzzer will be turned on for the delay specified and the it will be turned off for the same delay and so on till this mode will be turned off (reflected in `/proc/vsopenrisc/buzzer_frq`)

### 6.5.3. LEDs

The OpenRISC provides three configurable LEDs:

- red LED (power LED)
- blue LED (can be used to signal WLAN link)
- green LED (user LED)

Following IOCTL commands are defined in the `vsopenrisc.h` to control the LEDs:

- `GPIO_CMD_GET_LEDS/GPIO_CMD_SET_LEDS` - get/set LED combination (reflected in `/proc/vsopenrisc/leds`)
- `GPIO_CMD_SET_LED_POWER` - turn on/off the power LED
- `GPIO_CMD_SET_LED_BLUE` - turn on/off the blue LED
- `GPIO_CMD_SET_LED_GREEN` - turn on/off the green LED

#### 6.5.4. Serial Interfaces

**RS232/422/485 modes** The serial ports can operate in one of the three modes RS232, RS422 or RS485. These modes will be controlled through the EPLD circuit. Following commands are be used to switch the modes:

- TI0CGEPLD - get EPLD mode
- TI0CSEPLD - set EPLD mode

These commands use the following structure to store the desired parameters:

```
struct epld_struct
{
    unsigned long port;
    unsigned long reg_shift;
    unsigned long value;
};
```

The field `value` can have one of the following values:

- EPLD\_RS232 - RS232 mode
- EPLD\_RS422 - RS422 mode
- EPLD\_RS485\_ART\_4W - RS485 mode 4-wire transmit control by ART
- EPLD\_RS485\_ART\_2W - RS485 mode 2-wire direction control by ART
- EPLD\_RS485\_ART\_ECHO - RS485 mode 2-wire direction control by ART with echo
- EPLD\_RS485\_RTS\_4W - RS485 mode 4-wire transmit control by RTS
- EPLD\_RS485\_RTS\_2W - RS485 mode 2-wire direction control by RTS
- EPLD\_RS485\_RTS\_ECHO - RS485 mode 2-wire direction control by RTS with echo
- EPLD\_PORTOFF - shutdown the port

**Baud rate generation interface** The OpenRISC provides full support for the 16C950 UART baud rate generation. This allows the user to use the serial interfaces with arbitrary speeds up to 3,6Mbit/s. To use this capability the current baud rate must be set to B38400 and the custom divisor to the negative value of the desired baud rate. In the source code below the baud rate will be set to 500000bit/s, so the custom divisor was set to -500000 (see the lines 26-39) and the current baud rate to 38400 (see the lines 41-66).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/time.h>
5 #include <sys/ioctl.h>
6 #include <sys/stat.h>
7 #include <unistd.h>
8 #include <termios.h>
9 #include <fcntl.h>
10 #include <linux/serial.h>
11 #include <string.h>
12
13 int main (int argc, char **argv)
14 {
15     int fd, ret;
```

```
16  struct termios ser_termios;
17  struct serial_struct ss_st;
18
19  fd = open("/dev/ttyS1", (O_RDWR | O_NOCTTY));
20  if (fd < 0)
21  {
22      perror("open");
23      return -1;
24  }
25
26  // set custom divisor to -500000 to achieve 500000 bit/s
27  if (ioctl(fd, TIOCGSERIAL, &ss_st)<0)
28  {
29      perror("TIOCGSERIAL");
30      return -1;
31  }
32  ss_st.custom_divisor = -500000;
33  ss_st.flags |= ASYNC_SPD_CUST;
34
35  if (ioctl(fd, TIOCSSERIAL, &ss_st)<0)
36  {
37      perror("TIOCSSERIAL");
38      return -1;
39  }
40
41  // set baud rate to 38400 bit/s to activate custom divisor
42  ret = tcgetattr(fd, &ser_termios);
43  if (ret < 0)
44  {
45      perror("getattr");
46      return -1;
47  }
48  ...
49  ret = cfsetispeed(&ser_termios, B38400);
50  if (ret < 0)
51  {
52      perror("ispeed");
53      return -1;
54  }
55  ret = cfsetospeed(&ser_termios, B38400);
56  if (ret < 0)
57  {
58      perror("ospeed");
59      return -1;
60  }
61  ret = tcsetattr(fd, TCSANOW, &ser_termios);
62  if (ret < 0)
63  {
64      perror("setattr");
65      return -1;
66  }
67  ...
68  close(fd);
69  return 0;
70 }
```

Every time the baud rate will be changed a special baud rate generation function will be invoked in the serial driver (`drivers/char/serial.c`) namely `serial16C950_get_divisors()`. This routine calculates appropriate values for the clock prescaler register (CPR), times clock register (TCR) and 16-bit divisor supported by 16C950 UART. If the routine couldn't find the appropriate values the baud rate 9600bit/s will be set. To disable this functionality comment the following definition in `drivers/char/serial.c`:

```
#define ENABLE_16C950_BAUD_GENERATION_FEATURES
```



## 7. /proc-Extensions for the OpenRISC

In the `/proc/vsopenrisc` directory reside several files to manipulate the OpenRISC hardware:

- `epld_ttys*` - configure serial driver e.g. rs232, rs422, rs485. For exact values execute `echo /proc/vsopenrisc/epld_ttys1`  
For detailed description see [6.5.4](#)
- `gpio_*` - set and read digital IO channels. For detailed description see [6.5.1](#)
- `leds` - set and read LEDs values. For detailed description see [6.5.3](#)
- `buzzer`, `buzzer_freq` - configure the buzzer. For detailed description see [6.5.2](#)
- `reset` - reboot the OpenRISC. To execute the hardware reset  
`echo 1 > /proc/vsopenrisc/reset`

Examples:

<code>echo 1 &gt; /proc/vsopenrisc/buzzer</code>	turn buzzer on
<code>echo 0 &gt; /proc/vsopenrisc/buzzer</code>	turn buzzer off
<code>echo 0x000001f4 &gt; /proc/vsopenrisc/buzzer_freq</code>	this will activate the buzzer for 500ms and then deactivate it for 500ms and so on
<code>cat /proc/vsopenrisc/leds</code>	get the current LED status
<code>echo GREEN &gt; /proc/vsopenrisc/leds</code>	turn the green LED on
<code>echo green &gt; /proc/vsopenrisc/leds</code>	turn the green LED off
<code>echo rs422 &gt; /proc/vsopenrisc/epld_ttyS2</code>	sets the mode of the second port to rs422

## 8. BIOS

BIOS (Basic Input Output System) lets you configure your OpenRISC e.g. configure how to boot, set up date, time and so on. To get into the BIOS, press 'ESC' during the system start and you'll enter the following menu (see Figure 4).

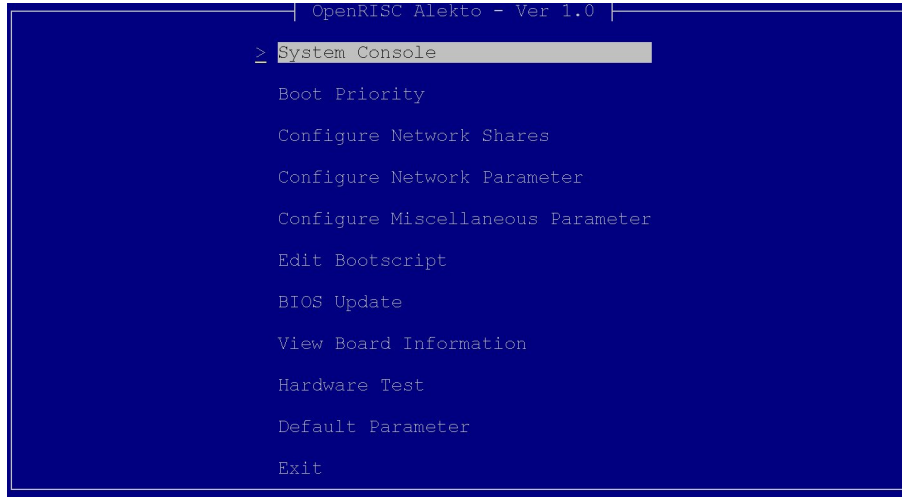


Figure 4: BIOS: main menu

To connect to the BIOS via Telnet press reset push button for a while till the following output appears (see Figure 5). Now you can connect to the BIOS via network by using the IP address specified under "Configure Network Parameter" (see 8.4).

Press 'Esc' to enter setup or 'Tab' for a bootlist

Figure 5: BIOS Prompt for Bootlist

## 8.1. System Console

Choosing this menu item, you'll get to the system console. To return to the main menu press 'Ctrl+D' or execute `exit` (see Figure 6). You can use the console to mount CompactFlash, copy the kernel (see 9.11), partition the disk and so on.

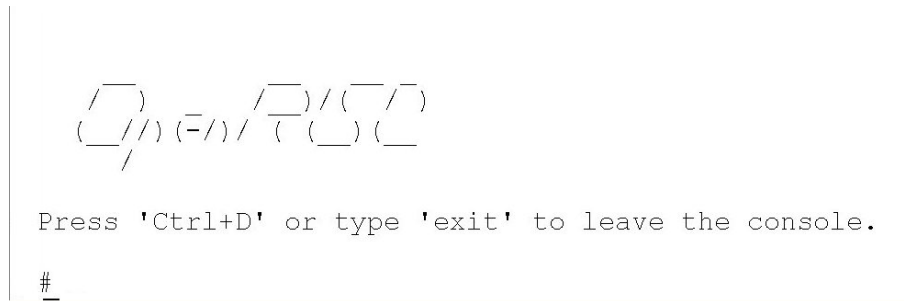


Figure 6: BIOS: System Console

## 8.2. Boot Priority

In this menu item you can change the boot priority (see Figure 7). You can choose between following sources:

- CompactFlash
- USB
- Network (Windows share, Samba)

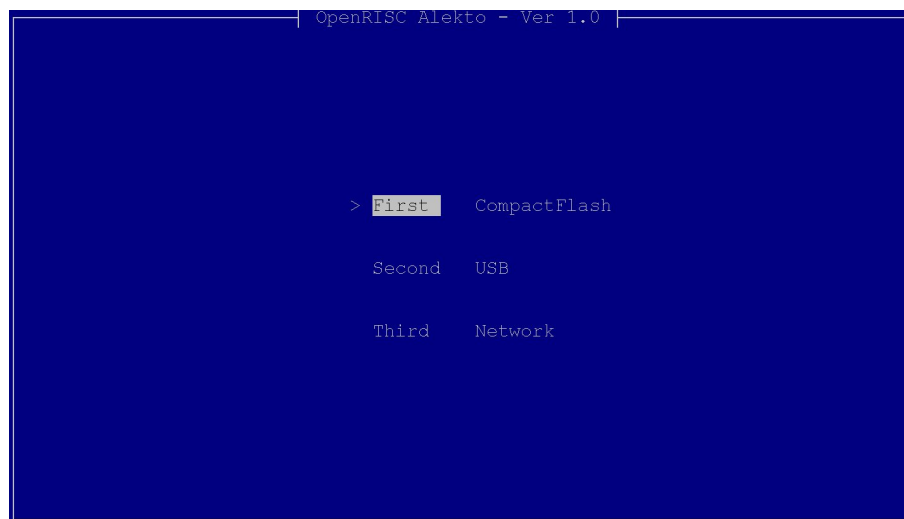


Figure 7: BIOS: Boot Priority

### 8.3. Configure Network Shares

Here you can configure a Windows share (SMB) to boot from (see Figure 8). To use this feature, you should set up following parameters:

- Type
- Server IP address
- Samba directory name
- User name
- Password

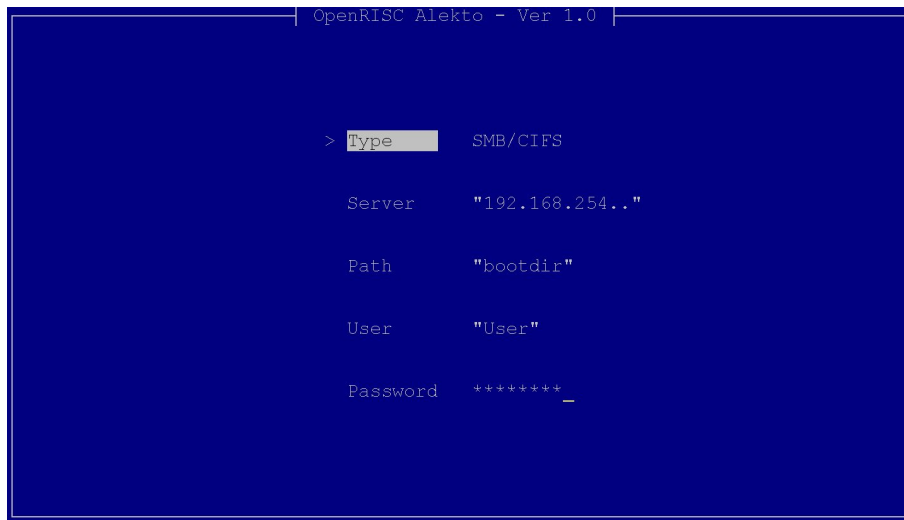


Figure 8: BIOS: Configure Network Shares

#### 8.4. Configure Network Parameter

Here you can configure the network parameters (see Figure 9). You can choose between getting IP Address via DHCP or to assigning it statically. For the latter you should configure following parameters:

- IP Address
- Netmask
- Broadcast
- Gateway



Figure 9: BIOS: Configure Network Parameter

### 8.5. Configure Miscellaneous Parameter

Following parameters can be set up here (see Figure 10):

- Date
- Time
- Start-Timeout - the time in seconds during that the BIOS or bootlist could be accessed
- Terminal Type (Console/Telnet) - emulation type to choose by terminal software such as HyperTerminal, ZOC etc while connecting via serial link or telnet. The OpenRISC can emulate three types of terminals:
  - Linux
  - ANSI
  - VT100
- Password - BIOS protecting password

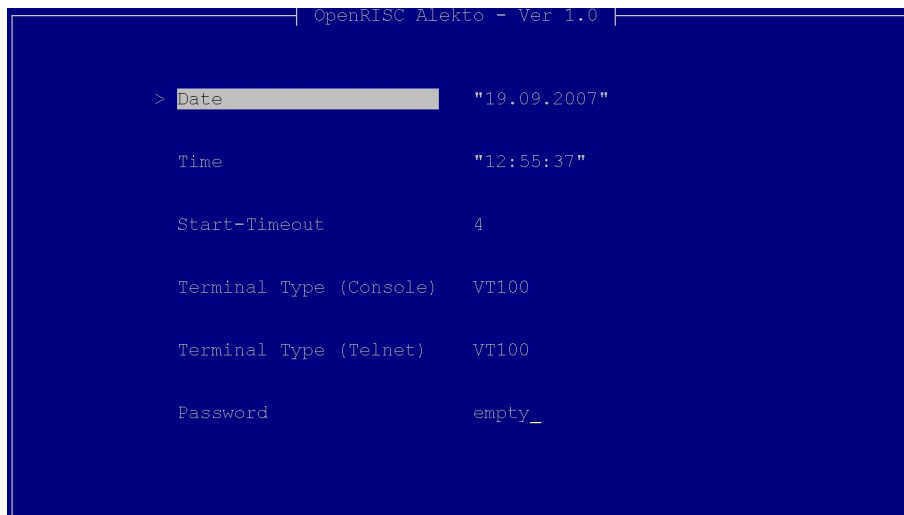


Figure 10: BIOS: Configure Miscellaneous Parameter



## 8.7. BIOS Update

Here you can update the BIOS to a newer version via sending a file with your terminal software.

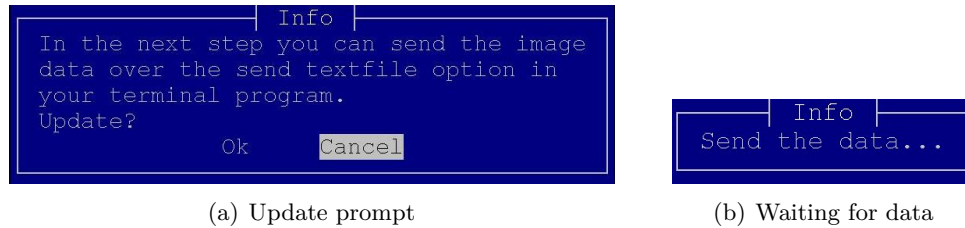


Figure 12: BIOS: Update

In the dialog show in Figure 12(a) choose “Ok”. “Send the data...” prompt appears (see Figure 12(b)). Select the appropriate \*.b64 file and send it via the “Send Text File” functionality of your terminal software.

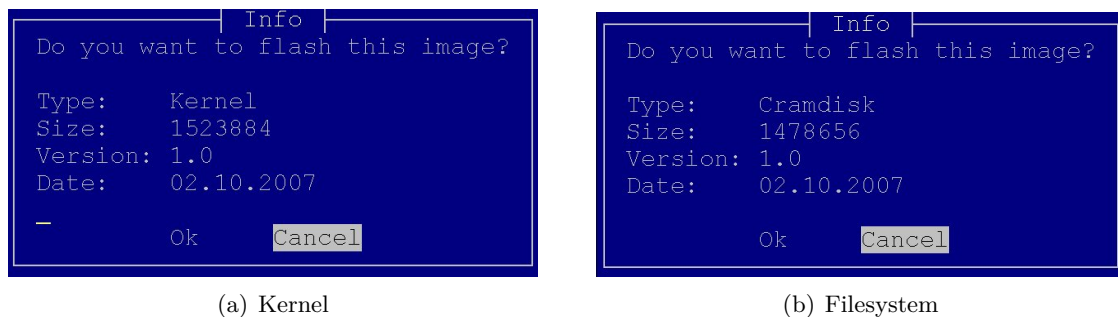


Figure 13: BIOS: Update (Images)

The \*.b64 file consists of two files:

- Kernel - Linux kernel image (zImage)
- Cramdisk - filesystem image

After transmission you can choose between installing both kernel (see Figure 13(a)) and filesystem image (see Figure 13(b)) or only one of them.

**Warning:** do not power off or reset the OpenRISC during the flashing of the images!!!

The last step is to reboot the system (see Figure 14).

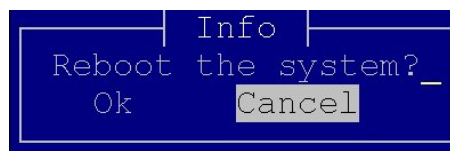


Figure 14: BIOS: Update (Reboot System)



## 8.8. View Board Information

Shows some board information (see Figure 15)

```
| OpenRISC Alekto - Ver 1.0 |
Hardware Revision  1.0
Serialnumber       1234567
Production Date    20.02.2006
BIOS Id            1
MAC 1              00:10:20:30:40:50
MAC 2              00:11:22:33:44:55
Flash Size 1       2
Flash Size 2       0
Memory Size        16
```

Figure 15: BIOS: View Board Information

## 8.9. Hardware Test

The `hwtest` program will be invoked for one cycle with the parameters listed in Table 5. For detailed information about `hwtest` refer to Appendix B.

size	4096	modem	
cycles	1	sero	/dev/ttyS1
confirm		seri	/dev/ttyS2
serial		io	
net		buz	
gpio		led	
mpci		sleep	1
cf		testrtc	
i2c		eplddev	/dev/ttyS1
epld		eplddev	/dev/ttyS2
usb		usbmntdev	/dev/sda1

Table 5: BIOS: hwtest parameters

## 8.10. Default Parameter

Sets all parameters to factory settings.

## 8.11. Exit

Leaves BIOS. If any parameters were changed, you'll be asked to save the changes.

## 9. Debian Installation

The installation of the Debian Etch 4.0r0<sup>31</sup> will be described in this section. Your system should meet the following requirements:

- Serial Terminal (Emulation) at 115200bps, 8N1
- Installation source: CD/DVD-ROM on USB or Windows share (Samba)
- Installation target: CompactFlash card or USB Mass Storage device<sup>32</sup>

Due to the fact that OpenRISC is not officially supported by arm-debian port there is no suitable kernel in the distribution. So you have to provide boot image files to the OpenRISC to be able to start the Debian installer. To get these files unpack the kernel source `development\linux-2.4.32-ks8695.tar.bz2` from DVD "Software". The files `zImage`, `initrd.gz` and `kparam` are stored under `linux-2.4.32-ks8695/trunk/Vscom/installation`.

### 9.1. Preparing Boot Image Files

To install Debian you need three files to be placed in the root directory of your USB mass storage device. These files are:

- `init.rd` - RAM disk image with Debian Installer
- `zImage` - Linux kernel
- `kparam` - Kernel parameter like `mem=0x3b00000 root=/dev/ram` describing that the kernel should use 59Mb of memory and Linux should boot from RAM disk

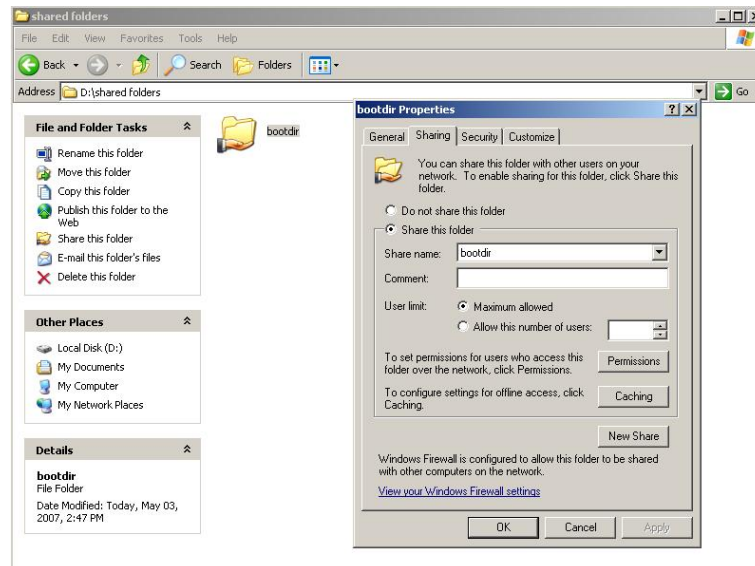
Alternatively you can use a Windows share (Samba) (see Figure 16). It can be made in four steps:

1. create shared folder. For example `D:\shared folders\bootdir` (see Figure 16(a))
2. copy the image files to this folder (see Figure 16(b))
3. enter BIOS and set the Network Parameter (see 8.4)
4. configure your BIOS to access the network share directory (see 8.3)

---

<sup>31</sup>For detailed information about Debian Installation visit <http://www.debian.org/releases/stable/arm/>

<sup>32</sup>512 MB or more recommended



(a) Shared Folder Properties

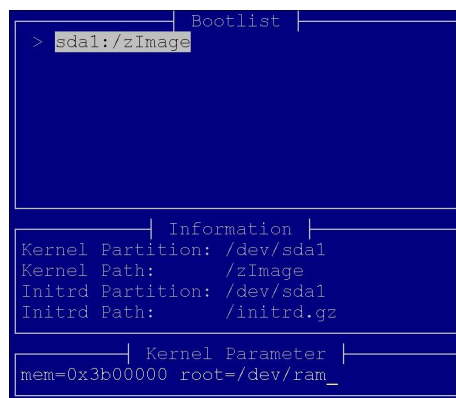


(b) Boot Image Files

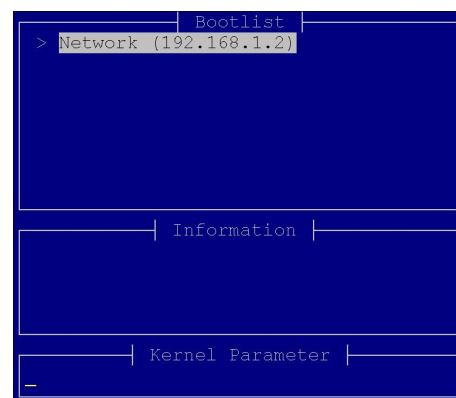
Figure 16: Windows Share

## 9.2. Starting Debian Installer

Exit BIOS and save changes if required. Press 'Tab' to access bootlist and choose USB Mass Storage Device (see Figure 17(a)) or Network (see Figure 17(b)). The Installer will be put into low memory mode (see Figure 18). Press 'Enter' and choose your country in the next dialog.



(a) Mass storage



(b) Windows share

Figure 17: Bootlist

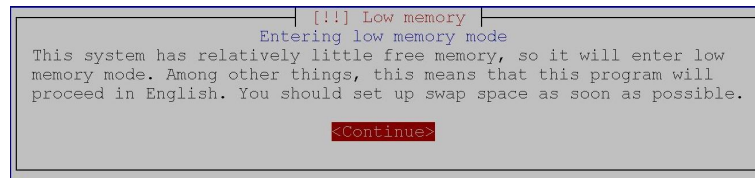


Figure 18: Low Memory Mode

### 9.3. Choosing CD-ROM

Debian Installer will try to find a device driver for your USB CD-ROM. This will fail. To select the device manually answer with 'No' (see Figure 19). In the second dialog say 'Yes' to manually select your CD-ROM module. In the third dialog select 'none'. In the fourth dialog press 'Enter' to select `/dev/cdrom` device file to access the CD-ROM. The Debian Installer will now scan your CD-ROM.

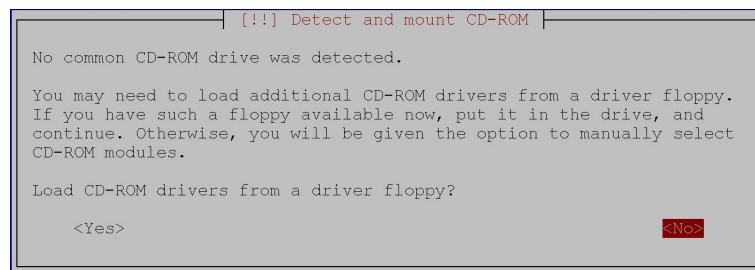


Figure 19: Detect and mount CD-ROM

### 9.4. Load Installer Components from CD

First you'll see a dialog with the statement that the kernel modules were not found. It is due to the issue that all needed modules were compiled into the kernel. So answer with 'Yes'. Additional components will be loaded.

### 9.5. Network Configuration

In the next dialog the network configuration will be executed (see Figure 20). Choose the network interface to which the Ethernet cable is connected to. In the next two dialogs enter host and domain names.

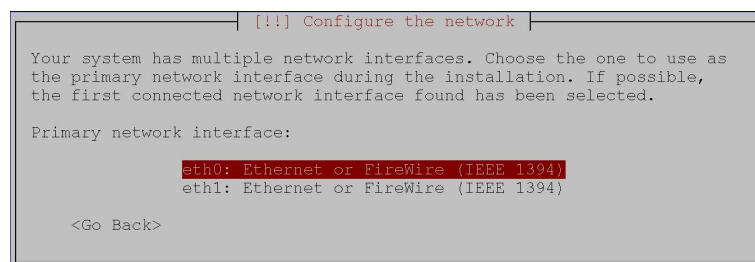


Figure 20: Network configuration

## 9.6. Partition Disks

After scanning your CompactFlash, you'll be asked to continue with partitioning. Answer with 'Yes'. In the second dialog select 'Manual'. In the third dialog (see Figure 21) you can partition your CF. It is recommended to create two partitions (for 512Mb CF-Card):

1. ext2 with 455Mb
2. swap with 64Mb

Following settings should be changed for the first partition (see Figure 22):

- Use as: Ext2 filesystem<sup>33</sup>
- Format the partition: yes, format it
- Mount point: /

After doing this select 'Finish partitioning and write changes to disk'. In the next dialog select 'Yes' to format both partitions.

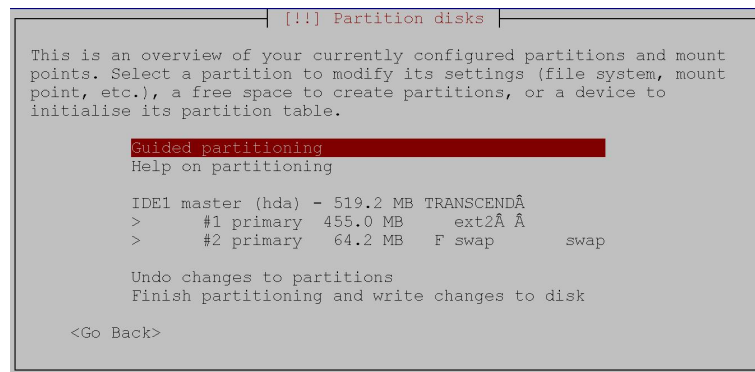


Figure 21: Partition Overview

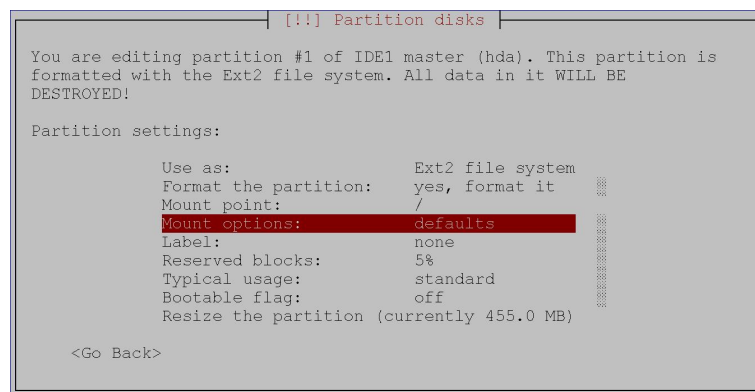


Figure 22: Ext2 Partition Settings

---

<sup>33</sup><http://e2fsprogs.sourceforge.net/ext2intro.html>

## 9.7. Setting Passwords

After partitioning the disks and setting up the clock, you'll be asked to set a password for 'root' and create a new user.

## 9.8. Install the Base System

As the OpenRISC is not officially supported by Debian, there is no kernel shipped with the distribution. Answer with 'Yes', when you'll be asked to install without a kernel.

## 9.9. Configure the Package Manager

You can configure the Debian Installer to use a network mirror. If you answer with 'Yes', you'll get a list of mirrors to choose from. After configuring the package manager, you'll be asked to participate in the package usage survey.

## 9.10. Software Selection

In this dialog you can select which component groups will be installed on your system (see Figure 23).

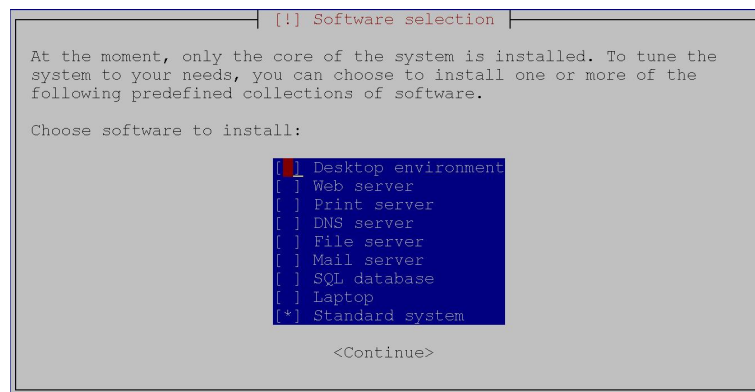


Figure 23: Software Selection

### 9.11. Finish the Installation

Due to the lack of the official support by Debian there is no boot loader (see Figure 24). After reboot, enter BIOS and go to the console. Mount the first partition:

```
mount /dev/hda1 /mnt
```

Copy zImage either from your host or USB mass storage device into the /boot directory and execute:

```
echo "mem=59M root=/dev/hda1" > /boot/kparam
```

Exit BIOS and enter the bootlist by pressing 'Tab'. You can now boot from /dev/hda1.

To enable the console port edit /etc/inittab, so that tty1-tty6 are commented and console is configured for 115200bps:

```
...
# Note that on most Debian systems tty7 is used by the X Window System,
# so if you want to add more getty's go ahead but skip tty7 if you run X.
#
#1:2345:respawn:/sbin/getty 38400 tty1
#2:23:respawn:/sbin/getty 38400 tty2
#3:23:respawn:/sbin/getty 38400 tty3
#4:23:respawn:/sbin/getty 38400 tty4
#5:23:respawn:/sbin/getty 38400 tty5
#6:23:respawn:/sbin/getty 38400 tty6
# Example how to put a getty on a serial line (for a terminal)
#
T0:2345:respawn:/sbin/getty -L console 115200 linux
#T1:23:respawn:/sbin/getty -L ttyS1 9600 vt100
...
```

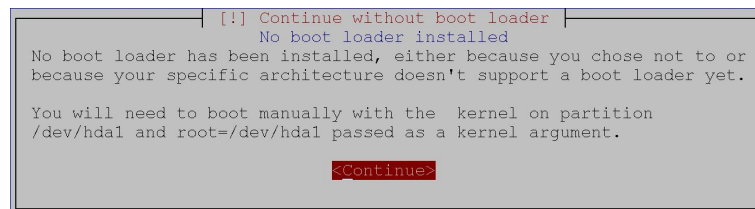


Figure 24: Continue without Boot Loader

## A. Debian Maintenance Notes

### A.1. Debian Package Management

Debian uses following utilities for the package management<sup>34</sup> :

- `dpkg` - the main package management program.
- APT - the Advanced Package Tool. It provides the `apt-get` program. `apt-get` allows a simple way to retrieve and install packages from multiple sources using the command line. Unlike `dpkg`, `apt-get` does not understand `.deb` files, it works with the packages proper name and can only install `.deb` archives from a source specified in `/etc/apt/sources.list`. `apt-get` will call `dpkg` directly after downloading the `.deb` archives from the configured sources.
- `aptitude` - a package manager for Debian GNU/Linux systems that provides a frontend to the apt package management infrastructure. `aptitude` is a text-based interface using the curses library, it can be used to perform management tasks in a fast and easy way.

To find a package execute:

```
apt-cache search pkg name
```

To view info such as version, dependencies, installed size etc. execute:

```
apt-cache show pkg name
```

To install packages execute:

```
apt-get install pkg1 pkg2 ... (su rights needed)
```

To update the list of package known by your system execute:

```
apt-get update (su rights needed)
```

To upgrade all the packages on your system

```
apt-get upgrade (su rights needed)
```

To remove packages from your system execute:

```
apt-get remove pkg1 pkg2 ... (su rights needed)
```

To install a package that is not contained in the repository download the `*.deb` file and execute:

```
dpkg -i pkg file name (su rights needed)
```

One of the on-line repositories is already configured, to use the enclosed DVD "Debian" uncomment the first repository entry:

```
# deb cdrom:[Debian GNU/Linux 4.0 r0 _Etch_ - Official arm DVD Binary-1 20070407-18:08]/ etch contrib  
main  
deb http://security.debian.org/ etch/updates main contrib  
deb-src http://security.debian.org/ etch/updates main contrib  
deb http://ftp.de.debian.org/debian stable main
```

---

<sup>34</sup>for detailed information visit <http://www.debian.org/doc/FAQ/ch-pkgtools.en.html>



## A.2. Keep a Track on Disk Usage

To get the list of all installed packages with its installed sizes execute:

```
dpkg-query -W -f'${Package}\t${Installed-Size}\t${Status}\n' | awk '/installed/ { print $2 "\t" $1 }'
```

To estimate the file space usage execute:

```
du -h
```

To get a HTML output execute:

```
durep -w /tmp/web/
```

For detailed information see the manpages for `du` and `durep`.

To estimate free disk space execute:

```
df -h
```

## B. Hardware Test Utility: hwtest

The test utility for the OpenRISC hardware is provided in both BIOS and preconfigured Debian CF-images and consists of the following test modules:

- Network
- MiniPCI
- Serial
- GPIO
- CompactFlash
- I2C and RTC
- EPLD
- USB

Three modes can be chosen for the test execution :

1. userless - executes all tests and shows the statistics at the end
2. fully interactive - the user has to acknowledge each test
3. half interactive - all of the automatic tests such as Network, MiniPCI, Serial, EPLD, USB and CompactFlash will be executed without user acknowledgment but GPIO and I2C would ask for the acknowledgment

The parameters can be specified in the command line or stored in the configuration file<sup>35</sup>. Without any parameter, the program will not execute any test. For the list of common parameters see Table 6.

<code>--cfg &lt;file name&gt;</code>	configuration file
<code>--cycles &lt;number&gt;</code>	number of test cycles. Default: infinite
<code>--size &lt;bytes&gt;</code>	test file size in bytes. Default: 1048576 bytes
<code>--mode &lt;mode&gt;</code>	<code>uless</code> for userless, <code>fint</code> for fully interactive, <code>hint</code> for half interactive. Default: <code>uless</code>
<code>--failstop</code>	stop testing after the first failure
<code>--verbose</code>	verbose output
<code>--net &lt;params&gt;</code>	network test module with its parameters
<code>--mpci &lt;params&gt;</code>	MiniPCI test module with its parameters
<code>--serial &lt;params&gt;</code>	serial test module with its parameters
<code>--gpio &lt;params&gt;</code>	GPIO test module with its parameters
<code>--cf &lt;params&gt;</code>	CompactFlash test module with its parameters
<code>--i2c &lt;params&gt;</code>	I2C and RTC test module with its parameters
<code>--epld &lt;params&gt;</code>	EPLD test module with its parameters
<code>--usb &lt;params&gt;</code>	USB test module with its parameters

Table 6: Common parameters and test modules

For detailed information about options execute:

---

<sup>35</sup>To use configuration file execute `hwtest --cfg=hwtest.conf`

```
hwtest --help
```

### B.1. Network Test

Two interfaces must be connected with each other for the network test (either with a patch or crossover cable). A test file will be sent as raw ethernet packets. To see the packet content use the `--verbose` flag. The size of the ethernet packet (in bytes) can be defined with `--nblock` option. The delays between two packets will be defined with the `--ndelay` option (default 100000 microseconds).

Usage example:

```
hwtest --cycles=1 --size=4096 --net --verbose
```

executes the network test with a 4096 byte test file in verbose mode

### B.2. MiniPCI Test

During the test `/proc/bus/pci/devices` will be searched for the presence of the WLAN card. If the WLAN card is found the name of the chip will be printed. After that, the WLAN environment will be searched for available participants.

Usage example:

```
hwtest --cycles=1 --mpci
```

executes the MiniPCI test

### B.3. Serial Test

To test the serial port you will need a special null-modem cable. Following pinout will be used (see Table 7):

TX	↔	RX
RX	↔	TX
RTS	↔	CTS, RI
CTS, RI	↔	RTS
DTR	↔	DSR, CD
DSR, CD	↔	DTR

Table 7: Customized null-modem cable layout

The test file will be transferred from the first interface to the other and vice versa. This test can be also used to simultaneously test the serial and USB interfaces. To do this you must connect the USB port with the serial one using a USB-to-Serial adapter based on the FTDI chip. The serial test module has its own parameters `--seri` and `--sero` to configure the interfaces. With the `--sblock` parameter, the serial write block size can be configured. By default it is set to 64 bytes. The `--rtscts` option enables hardware handshake. The `--modem` option enables the test of the modem status pins.

Usage examples:

`hwtest --cycles=1 --size=4096 --serial --seri=/dev/ttyS1 --sero=/dev/ttyS2`  
executes the serial test for on-board serial interfaces

`hwtest --cycles=1 --size=4096 --serial --sblock=200 --rtscts`  
executes the serial test for default on-board serial interfaces (`/dev/ttyS1` and `/dev/ttyS2`) using hardware handshake and write block size of 200 bytes

`hwtest --cycles=1 --size=4096 --serial --seri=/dev/ttyS1 --sero=/dev/ttyUSB0`  
executes the serial test using a USB-to-Serial adapter based on the FTDI chip

## B.4. GPIO Test

The GPIO test module consists of following tests:

- IO test (will be activated with `--io` option)
- `poll()`, `select()` and interrupt functionality test (will be activated with `--poll` option)
- buzzer test (will be activated with `--buz` option)
- LEDs test (will be activated with `--led` option)

The GPIO pins must be connected with each other for the IO test (using 4,7k resistors for example) (see Table 8):

pin 1	↔	pin 2
pin 3	↔	pin 4
pin 5	↔	pin 6
pin 7	↔	pin 8

Table 8: GPIO Pin Connections

Usage examples:

`hwtest --cycles=1 --gpio --io --poll --buz --led --mode=fint`  
executes the complete GPIO test in full interactive mode

`hwtest --cycles=1 --gpio --io`  
executes only the IO test in userless mode

## B.5. CompactFlash Test

The CompactFlash test must be started from the BIOS or USB mass storage device, because the CF card must be unmounted. A test file will be written to the CF card during the test and then compared with the original one. After that, the test file will be removed from the CF card.

Usage example:

`hwtest --cycles=1 --size=4096 --cf`  
executes the CompactFlash test with 4096 bytes big test file

## B.6. I2C and RTC Test

A time stamp will be read from the Real Time Clock using I2C bus during the RTC test . The second time stamp will be read after the delay in seconds specified by `--sleep` option (default value is 1 second) and then compared. The test can be activated/deactivated with `--testrtc` and `--testlcd` options.

The LCD display (EAT123A-I2C) must be connected to the OpenRISC for the I2C test. The program connects to the display and "O.K." string must be visible on the display.

Usage examples:

```
hwtest --cycles=1 --i2c --testrtc --testlcd
executes the RTC and LCD tests
```

```
hwtest --cycles=1 --i2c --testrtc
executes the RTC test only
```

```
hwtest --cycles=1 --i2c --testlcd
executes the LCD test only
```

## B.7. EPLD Test

Each UART has its own EPLD to switch between RS232, RS422 and RS485 modes. Current configuration will be acquired and then switched to RS232 or RS422 depending on what the current configuration was. The EPLDs will be chosen using serial port devices configured in `--eplddev` option.

Example:

```
hwtest --cycles=1 --epld --eplddev=/dev/ttyS1 --eplddev=/dev/ttyS2
executes the EPLD test for both serial ports
```

## B.8. USB Test

For USB test USB mass storage device should be connected to the USB port. The device must be properly detected and mounted. `--usbmntdev` option defines which device to mount.

Usage examples:

```
hwtest --cycles=1 --usb --usbmntdev=/dev/sda1
executes the USB test for the first detected mass storage device
```

```
hwtest --cycles=1 --usb --usbmntdev=/dev/sda1 --usbmntdev=/dev/sdb1
executes the USB test for the first and the second detected mass storage devices
```

## B.9. Build Notes

hwtest can be built either with GNU make or CMake. Both `Makefile` and `CMakeLists.txt` are supplied. The command line parameter parser was generated using GNU gengetopt<sup>36</sup>. To be able to generate the parser for changed options in `hwtest.ggo` one has to install GNU gengetopt tool (included in the hwtest distribution as tar.gz file).

---

<sup>36</sup><http://www.gnu.org/software/gengetopt/>

## C. VS Image Tool: VSImgTool

VSImgTool<sup>37</sup> was developed to make/copy images from/to the flash drives connected to a Windows host. This utility can make the whole image or only part of it. Refer to Table 9 for parameters.

<code>--list</code>	list available flash drives with its size in bytes
<code>source destination</code>	copy image from source to destination
<code>source destination custom_size</code>	copy only part of the image from source to destination

Table 9: VSImgTool call parameters

Usage example:

1. Insert CF card in your card reader and execute  
`vsimgtool --list`  
you should see the similar output:  
`>vsimgtool.exe --list`  
VSImgTool 1.0  
Available devices:  
PhysicalDrive2 Size: 512483328 bytes
2. to copy the base image to the CF card on the PhysicalDrive2 execute  
`vsimgtool.exe 09102007_etch_base.bin PhysicalDrive2`  
you should see the following output:  
`>vsimgtool.exe 09102007_etch_base.bin PhysicalDrive2`  
VSImgTool 1.0  
Image source: 09102007\_etch\_base.bin Size: 512483328 bytes  
Removable destination: PhysicalDrive2 Size: 512483328 bytes  
The data on the destination will be lost. Are you sure you want to continue?  
[yes/no] >
3. answer with "yes" and the copying process will begin
4. at the end of the copying process you should see the elapsed time and the size of the burned/-copied image

To make an image from your CF card execute:

```
vsimgtool drive_name image.bin
```

where *drive\_name* is the name showed by `vsimgtool --list`.

For 256Mb and 512Mb CF cards there are two custom sizes prepared:

```
vsimgtool PhysicalDrive2 image.bin 256  
will copy exactly  $244 \cdot 1024 \cdot 1024 = 255852544$  bytes
```

```
vsimgtool PhysicalDrive2 image.bin 512  
will copy exactly  $488 \cdot 1024 \cdot 1024 = 511705088$  bytes
```

```
vsimgtool PhysicalDrive2 image.bin 519192576  
will copy exactly 519192576 bytes
```

---

<sup>37</sup>you must have administrator privileges to properly execute VSImgTool!

## D. Eclipse

### D.1. Installation Notes

Eclipse is an open source community whose projects are focused on building an extensible development platform, runtimes and application frameworks for building, deploying and managing software across the entire software life cycle. In Debian run:

```
apt-get install eclipse
```

to install it. Additionally you'll need to download and install Java 2 SDK or Java 2 JRE from Sun Microsystems<sup>38</sup>. It can happen that Eclipse doesn't find Java binary, in this case execute:

```
eclipse -vm java
```

To get working with C projects you'll need the CDT plug-in from Zylin AS<sup>39</sup> that also enables Eclipse to debug remote targets. To install CDT plug-in do the following:

1. delete previous CDT and Zylin Embedded CDT directories from `eclipse\features` and `eclipse\plugins` directory
2. download `embeddedcdt4.0-yyyyymmdd.zip` and `zylincdt4.0-yyyyymmdd.zip` files (yyyyymmdd are placeholders for the version) and unzip them to your Eclipse folder

If you'd like to checkout Kernel or examples from our Subversion repository, use doxygen etc. you could also find these plug-ins as useful:

- Subclipse (<http://subclipse.tigris.org/>)
- Eclox (<http://home.gna.org/eclox/>)
- Gengetopt Eclipse (<http://ggoeclipse.sourceforge.net/>)

---

<sup>38</sup><http://java.sun.com/javase/downloads/index.jsp>

<sup>39</sup><http://www.zylin.com/embeddedcdt.html>



## D.2. Debugging

Assuming your OpenRISC has IP address 192.168.1.66. Start your `ioctls` executable on the OpenRISC with following command:

```
gdbserver :9000 ioctls
```

On the PC side:

- open `Vscom/examples` project (the `.project` and `.cdtproject` file are already created)
- Select `Run->Debug`
- Double click `Embedded debug (Native)`. After that you'll get new configuration entry
- select the project corresponding to yours (see Figure 25)
- select `C/C++ Application` to `ioctls`
- choose `Debugger` tab (see Figure 26)
- GDB debugger: `arm-linux-gdb`
- choose `Commands` tab (see Figure 27)
- 'Initialize' commands: `target remote 192.168.1.66:9000`
- 'Run' commands: `c`
- Apply
- Debug

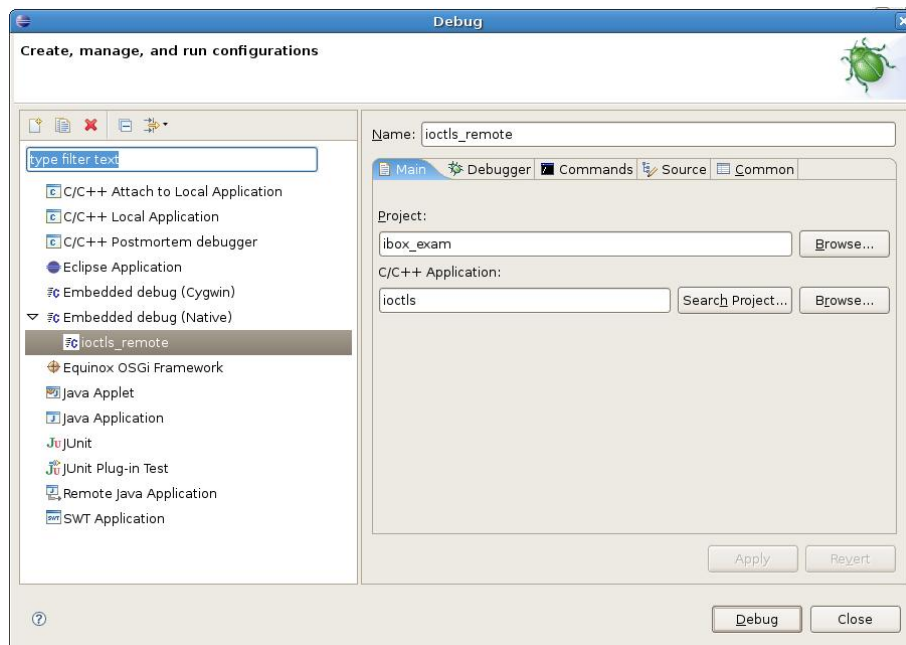


Figure 25: Eclipse debug: Main tab

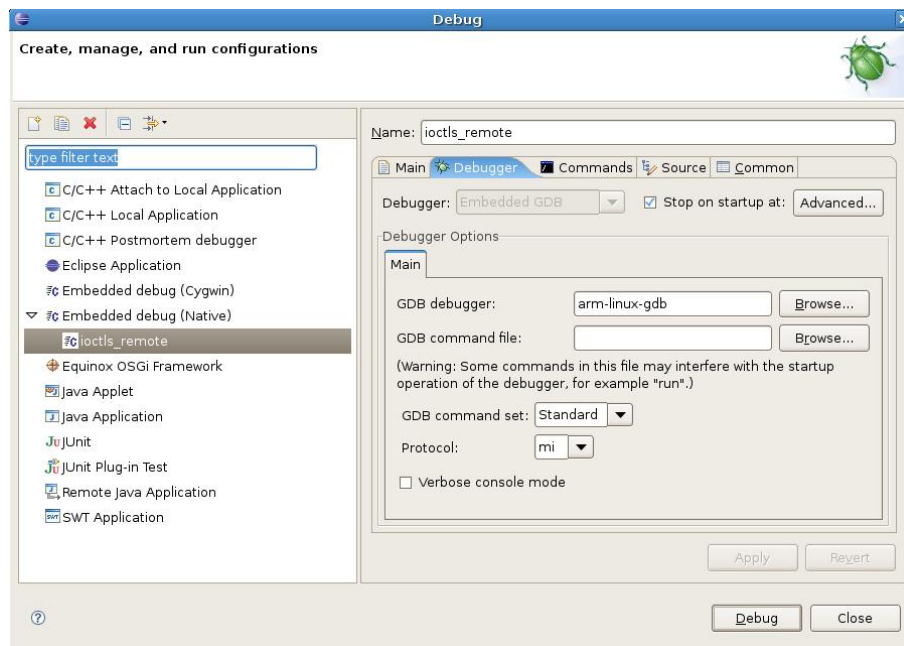


Figure 26: Eclipse debug: Debugger tab

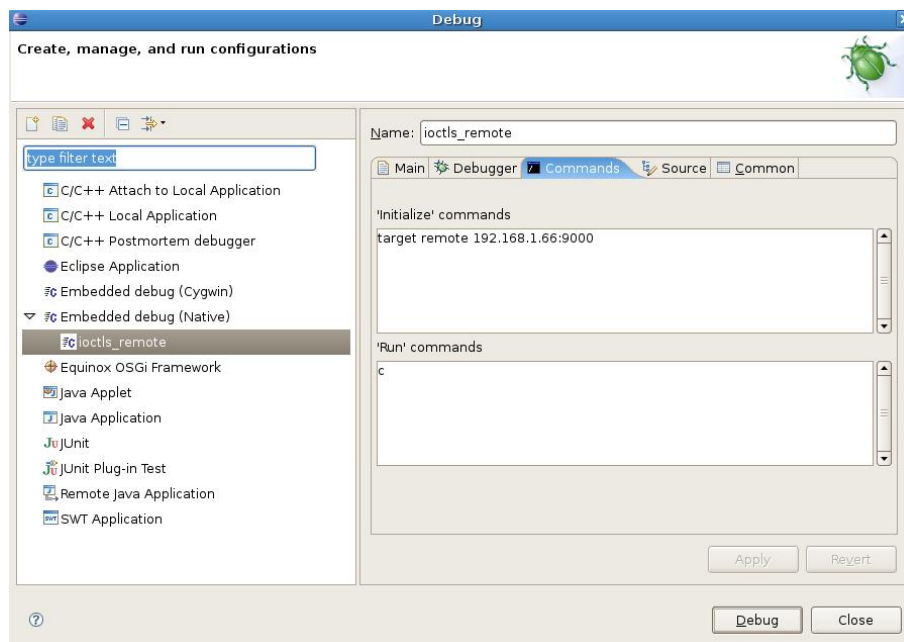


Figure 27: Eclipse debug: Commands tab

## E. DVDs Content

The OpenRISC will be shipped with two DVDs:

- DVD "Debian" is the Debian-40r0 for ARM distribution
- DVD "Software" includes documentation, development tools and sources:
  - development: arm-linux toolchain and kernel source
  - documentation: manuals and data sheets
  - images: CF images with preinstalled Debian r4.0 for ARM Linux
  - tools: various Windows tools useful for administrating and maintaining the OpenRISC

## Index

/proc-extensions, 33

AbiWord, 20

Apache, 16

apt, 48

BIOS, 34

Bluetooth, 22

buzzer, 30

CDT, 56

CMake, 25

courier, 17

dd, 13

Eclipse, 25, 56

Eclox, 56

EPLD\_PORTOFF, 31

EPLD\_RS232, 31

EPLD\_RS422, 31

EPLD\_RS485\_ART\_2W, 31

EPLD\_RS485\_ART\_4W, 31

EPLD\_RS485\_ART\_ECHO, 31

EPLD\_RS485\_RTS\_2W, 31

EPLD\_RS485\_RTS\_4W, 31

EPLD\_RS485\_RTS\_ECHO, 31

FTP, 24

gcc, 15

gdbserver, 28, 57

gengetopt, 54, 56

GPIO\_CMD\_GET, 30

GPIO\_CMD\_GET\_BUZZER, 30

GPIO\_CMD\_GET\_BUZZER\_FRQ, 30

GPIO\_CMD\_GET\_CHANGE, 30

GPIO\_CMD\_GET\_CHANGES, 30

GPIO\_CMD\_GET\_CTRL, 30

GPIO\_CMD\_GET\_IRQMASK, 30

GPIO\_CMD\_GET\_LEDS, 30

GPIO\_CMD\_SET, 30

GPIO\_CMD\_SET\_BUZZER, 30

GPIO\_CMD\_SET\_BUZZER\_FRQ, 30

GPIO\_CMD\_SET\_CTRL, 30

GPIO\_CMD\_SET\_IRQMASK, 30

GPIO\_CMD\_SET\_LED\_BLUE, 30

GPIO\_CMD\_SET\_LED\_GREEN, 30

GPIO\_CMD\_SET\_LED\_POWER, 30

GPIO\_CMD\_SET\_LEDS, 30

GUI, 19

HTTP, 17

hwtest, 41, 50

IMAP, 17

Insight, 28

ioctl.c, 29

ioctl2.c, 29

Java, 56

JRE, 56

LDAP, 17

LEDs, 30

Mail Server, 17

NAP, 22

netcat, 15

NFS, 24

NTP, 18

poll(), 30

POP3, 17

Samba, 24, 27, 36

select(), 30

socat, 15

sredird, 18

SSH, 23

SSL, 17

Subclipse, 56

telnet, 23

TIOCEPLD, 31

TIOCSEPLD, 31

Vim, 25

vsimgtool, 13, 55

vsopenrisc.h, 29

WindowMaker, 19

wireless-tools, 21

X-Server, 19

X-Window, 19

X11, 19

Xming, 19