

BlueCat Linux Board Support Guide

BlueCat Linux 4.0
DOC-0564-00

for Intel IQ80321 Boards

Product names mentioned in *BlueCat Linux Board Support Guide for Intel IQ80321 Boards* are trademarks of their respective manufacturers and are used here for identification purposes only.

Copyright ©1987 - 2003, LynuxWorks, Inc. All rights reserved.
U.S. Patents 5,469,571; 5,594,903

Printed in the United States of America.

All rights reserved. No part of *BlueCat Linux Board Support Guide for Intel IQ80321 Boards* may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photographic, magnetic, or otherwise, without the prior written permission of LynuxWorks, Inc.

LynuxWorks, Inc. makes no representations, express or implied, with respect to this documentation or the software it describes, including (with no limitation) any implied warranties of utility or fitness for any particular purpose; all such warranties are expressly disclaimed. Neither LynuxWorks, Inc., nor its distributors, nor its dealers shall be liable for any indirect, incidental, or consequential damages under any circumstances.

(The exclusion of implied warranties may not apply in all cases under some statutes, and thus the above exclusion may not apply. This warranty provides the purchaser with specific legal rights. There may be other purchaser rights which vary from state to state within the United States of America.)

Contents

CHAPTER 1	OVERVIEW	1
CHAPTER 2	DOWNLOADING AND BOOTING BLUECAT LINUX ON THE TARGET	3
	Prerequisites	3
	Downloading and Booting Overview	4
	Setting up Hardware	4
	Connecting Target Board Serial Ports to Host	4
	Connecting Target Board Ethernet Card to Host	5
	Setting up the RedBoot Firmware	5
	Installing LynuxWorks Boot Loader into Flash	6
	Setting up the LynuxWorks Boot Loader Firmware	7
	Downloading a BlueCat Linux System into Flash	8
	Downloading a BlueCat Linux System into Flash using RedBoot	8
	Downloading a BlueCat Linux System into Flash using Boot Loader ..	10
	Booting a BlueCat Linux System from Flash	10
	Booting a BlueCat Linux System from Flash using RedBoot	10
	Booting a BlueCat Linux System from Flash using Boot Loader	11
	Booting a BlueCat Linux System from a Network	12
	Booting a BlueCat Linux System from a Network using RedBoot	12
	Booting a BlueCat Linux System from a Network using Boot Loader ..	13
	Booting a BlueCat Linux System from a Network using the OS Loader	13
CHAPTER 3	KERNEL CONFIGURATION OPTIONS	15
CHAPTER 4	SUPPORTED DEMO SYSTEMS.....	39
	Demo Systems	39

developer Demo System	39
osloader Demo System	40
showcase Demo System	40
zebra Demo System	40
Using Selected RPM Packages	45
Using BusyBox RPM Package	45
Creating a BlueCat Linux System for BusyBox	46
Booting the BusyBox Images from a Network	47
Using BusyBox Utilities	48
Using TinyLogin RPM Package	49
Creating a BlueCat Linux System for TinyLogin	50
Booting the TinyLogin Images from a Network	52
Using TinyLogin Utility	53

CHAPTER 5 SUPPORTED DEVICE DRIVERS 55

CHAPTER 6 BLUECAT LINUX SPECIFIC APIS 57

Advanced MMU/Cache Management Services	57
Advanced MMU/Cache Management Services Overview	57
Advanced MMU/Cache Management Services Reference	58
Application Accelerator Unit Services	60
Application Accelerator Unit Services Overview	60
Application Accelerator Unit Services Reference	60
PCI DMA Transfers Services	62
PCI DMA Transfers Services Overview	62
PCI DMA Transfers Services Reference	62
Messaging Unit Support Services	64
Messaging Unit Support Services Overview	64
Messaging Unit Support Reference	65
Message Registers Services	65
Doorbell Registers Services	65
Circular Queues Services	66
Index Registers Service	66
Performance Monitoring Services	67
Overview	67
Performance Monitoring Services Reference	67

CHAPTER 7	KNOWN PROBLEMS AND LIMITATIONS	69
	IQ80321 Target Board Problems and Limitations	69

Preface

Typographical Conventions

The typefaces used in this manual, summarized below, emphasize important concepts. All references to file names and commands are case-sensitive and should be typed accurately.

Kind of Text

Examples

Body text; *italicized* for emphasis, new terms, and book titles

Refer to the *LynxOS User's Guide*.

Environment variables, file names, functions, methods, options, parameter names, path names, commands, and computer data

```
ls
-l
myprog.c
/dev/null
```

Commands that need to be highlighted within body text, or commands that must be typed as is by the user are **bolded**.

```
login: myname
# cd /usr/home
```

Text that represents a variable, such as a file name or a value that must be entered by the user

```
cat <filename>
mv <file1> <file2>
```

Blocks of text that appear on the display screen after entering instructions or commands

```
Loading file /tftpboot/shell.kdi
into 0x4000
.....
File loaded. Size is 1314816
Copyright 2002 LynuxWorks, Inc.
All rights reserved.
```

```
LynxOS (ppc) created Mon Jan 17
17:50:22 GMT 2002
user name:
```

Special Notes

The following notations highlight any key points and cautionary notes that may appear in this manual.

NOTE: These callouts note important or useful points in the text.



CAUTION! Used for situations that present minor hazards that may interfere with or threaten equipment/performance.

Technical Support

LynuxWorks Technical Support is available Monday through Friday (holidays excluded) between 8:00 AM and 5:00 PM Pacific Time (U.S. Headquarters) or between 9:00 AM and 6:00 PM Central European Time (Europe).

LynuxWorks U.S. Headquarters

email support@lnxw.com

phone (408) 979-3940
(800) 327-5969

fax (408) 979-3945

LynuxWorks Europe

email tech_europe@lnxw.com

phone (+33) 1 30 85 06 00

fax (+33) 1 30 85 06 06

World Wide Web

website <http://www.lynuxworks.com>

customer support <http://www.lynuxworks.com/support/custhelp.php3>

This *BlueCat Linux Board Support Guide for Intel IQ80321 Boards* (BSG) provides information about the BlueCat Linux Board Support Package (BSP) for the Intel 80321 I/O processor with Intel XScale microarchitecture.

Throughout this Board Support Guide (BSG), the BSP is referred to as the “iq80321”. The target board is referred to as the “IQ80321.”

The chapters of this BSG provide the information listed below:

- *Chapter 1* is an overview of this BSG’s individual chapters.
- *Chapter 2* describes BlueCat Linux downloading and booting procedures for the IQ80321 target board, using the BlueCat Linux `showcase` and `developer` demo systems as examples.
- *Chapter 3* provides configuration information about the iq80321 BSP’s default BlueCat Linux kernel.
- *Chapter 4* summarizes BlueCat Linux demo systems supported by the iq80321 BSP.
- *Chapter 5* provides a list of iq80321 BSP-supported device drivers, with important information about each of them.
- *Chapter 6* describes BlueCat Linux-specific Application Programming Interfaces (APIs) for the IQ80321 board.
- *Chapter 7* describes known limitations and workarounds of this release.

Downloading and Booting BlueCat Linux on the Target

This chapter provides instructions for downloading a BlueCat Linux demo system from a cross development host onto the target, and then booting the demo system on the target board.

Prerequisites

This document is a guide for downloading and booting BlueCat Linux systems onto the user's target board. Scenarios that use demo systems included in the BlueCat Linux distribution are presented. As such, a basic familiarity with the target board hardware and operation is required before using this guide. The user must also have an understanding of system administration for the particular cross development host on which BlueCat Linux Core and the BSP is installed. It is assumed that the user has the manufacturer's documentation for the target board as well as system administration reference material for the cross development host.

Before downloading and booting BlueCat Linux on the target board, it is assumed that the default BlueCat Linux ARM configuration and the iq80321 BSP have been installed on the cross development host. This means that the user must:

1. Install the BlueCat Linux XScale Core onto the cross development host, as described in the “Installing the Default Configuration” section in Chapter 1, “Installation” of the *BlueCat Linux User's Guide*.
2. Install the iq80321 BSP onto the cross development host as detailed in the “Installing Target Board Support” section of Chapter 1, “Installation” in the *BlueCat Linux User's Guide*.
3. Activate support for the iq80321 BSP as detailed in the “Activating Support for a Target Board” section of Chapter 1, “Installation” in the *BlueCat Linux User's Guide*.

Downloading and Booting Overview

The procedure for downloading and booting a BlueCat Linux system on the IQ80321 target consists of the following main steps:

- Setting up hardware
- Downloading and booting a BlueCat Linux system from target Flash memory or a network.

Downloading and booting a BlueCat Linux system can be performed using either of the two firmware monitors:

- RedBoot
 - LynuxWorks Boot Loader
- or
- BlueCat Linux OS loader.

The LynuxWorks Boot Loader is a firmware intended to act as a bootloader for BlueCat Linux and LynxOS. Boot Loader is able to load BlueCat Linux embedded systems from Flash or over a network and to program them to Flash.

The BlueCat Linux OS loader demo system includes the `i_osloader` and `osloader` downloadable images. `osloader` is the image with the base functionality of the BlueCat OS loader configured in. This includes the ability to download BlueCat images from a TFTP host, execute them in RAM, and other important features. `i_osloader` is extended with support for the Journalling Flash File System (JFFS) and can thus be used to download a desired BlueCat Linux custom or demo system into the target board's Flash memory.

Please refer to Chapter 3, “Downloading and Booting BlueCat Linux” in the *BlueCat Linux User's Guide* for a discussion of the OS loader.

Setting up Hardware

Connecting Target Board Serial Ports to Host

The target board has one serial port. This port (the J1G1 connector) is used by the RedBoot and LynuxWorks Boot Loader firmware as well as by BlueCat Linux.

Before using the board, this port needs to be connected to the development host. It is recommended to connect the J1G1 connector to COM1 on the host. Use the serial cable shipped with the IQ80321 board to connect the port.

The serial port connected to the target J1G1 port must be configured at a baud rate of 115200.

Throughout this chapter, the terminal window connected to the serial connector is referred to as the *Boot Loader* console or the *RedBoot* console, depending on the context.

Connecting Target Board Ethernet Card to Host

The Ethernet port on the target board is used to provide a standard network connection for the board, and, in particular, to load BlueCat Linux embedded systems onto the board over a network.

The Ethernet port on the IQ80321 board is the J1F1 connector. The user must use this port to connect the IQ80321 to the LAN.

It also required to set up networking on the host system. In particular, the user must choose a unique IP address for the development host as well as for the target board.

These addresses are referred to as *development_host_IP* and *target_board_IP* respectively. For more information on how to set up networking on the host, please refer to the host operating system documentation.

TFTP must be enabled on the host. Refer to “Setting Up a TFTP Server” in Chapter 3 of the *BlueCat Linux User's Guide* for more information.

Setting up the RedBoot Firmware

Use the following procedure to set up the RedBoot firmware options for BlueCat Linux operations:

1. Reset the target board.
2. At the RedBoot console, enter:

```
RedBoot> fconfig
Run script at boot: false Enter
Use BOOTP for network configuration: false Enter
Local IP address: target_board_IP
Default server IP address: development_host_IP
Console baud rate: 115200 Enter
```

```
DNS server IP address: Enter
GDB connection port: 9000 Enter
Force console for special debug messages: false Enter
Network debug at boot time: false Enter
Update RedBoot non-volatile configuration - are you sure (y/n)? y
... Unlock from 0xf07c0000-0xf07c1000: .
... Erase from 0xf07c0000-0xf07c1000: .
... Program from 0x01fd2000-0x01fd3000 at 0xf07c0000: .
... Lock from 0xf07c0000-0xf07c1000: .
RedBoot>
```

where *target_board_IP* is the IP address of the target, and
development_host_IP is the IP address of the development host.

Installing LynuxWorks Boot Loader into Flash

To install LynuxWorks Boot Loader onto the IQ80321 board, the on-board Flash device must be programmed with the Boot Loader image (\$BLUECAT_PREFIX/boot/iq80321.bin).

The IQ80321 board has a documented mechanism for Flash programming. The CD-ROM coming with the board includes an MS-DOS utility (Flash Recovery Utility, FRU) to program the on-board Flash using a PC. The board must be inserted into a motherboard's PCI slot. The utility runs on the PC from a floppy and programs the Flash of the IQ80321 board.

Perform the following steps to install LynuxWorks Boot Loader onto the IQ80321 board:

1. Prepare a system floppy disk with FRU as described in the *FRU 6.0 Reference Manual* (FRU60Manual.pdf) that is available on the CD-ROM shipped with the target board.
2. Copy the Boot Loader image (\$BLUECAT_PREFIX/boot/iq80321.bin) to the floppy.

NOTE: The Boot Loader image may not fit on the same floppy with the Flash Recovery Utility. You can download the Boot Loader image from a separate floppy using the Flash Recovery Utility.

3. Set the S7E1-2 and S7E1-3 jumpers to ON.

4. Install the target board in the PCI slot of a PC computer.

NOTE: The PC motherboard must support either 3.3V PCI or PCI-X. The IQ80321 board will not plug into motherboards equipped with a 5V PCI.

5. Boot the host PC from the system disk created in step 1.
6. Type `FRU.EXE` and press **Enter**.
7. Using the utility, program the LynuxWorks Boot Loader image into Flash as described in the *FRU 6.0 Reference Manual* (`FRU60Manual.pdf`).
If the Boot Loader image resides on a separate floppy, insert the floppy and browse and select the Boot Loader image from the Flash Recovery Utility.
8. Turn off the host PC computer, remove the IQ80321 board, install it back to the PCI backplane, set the S7E1-2 and S7E1-3 jumpers to OFF, connect the serial cable to the J1G1 connector and turn on the board.
9. Reset the board. At this time, the Boot Loader prompt must appear on the serial console.

Setting up the LynuxWorks Boot Loader Firmware

Use the following procedure to set up the LynuxWorks Boot Loader firmware for BlueCat Linux operations:

1. Reset the target board.
2. At the LynuxWorks Boot Loader console, enter the following commands:

```
iq80321> set autoboot 0
iq80321> set boot_tftp_host_ip development_host_IP
iq80321> set boot_tftp_client_ip target_board_IP
iq80321> set flash_tftp_host_ip \
development_host_IP
iq80321> set flash_tftp_client_ip target_board_IP
iq80321> save
```

where *target_board_IP* is the IP address of the target,
development_host_IP is the IP address of the development host.

3. Reset the target board.

Downloading a BlueCat Linux System into Flash

This section provides instructions on how a BlueCat embedded system can be downloaded into the target Flash memory using the RedBoot firmware (via the BlueCat Linux OS loader) and the LynuxWorks Boot Loader firmware. Refer also to the *BlueCat Linux User's Guide* for additional details about the BlueCat Linux OS loader.

Downloading a BlueCat Linux System into Flash using RedBoot

Specifically, these instructions are applicable to any of the demo systems. This chapter uses the `osloader` demo system as an example.

The following figure shows how the Flash memory on the IQ80321 board is partitioned for BlueCat Linux if the RedBoot monitor is used.

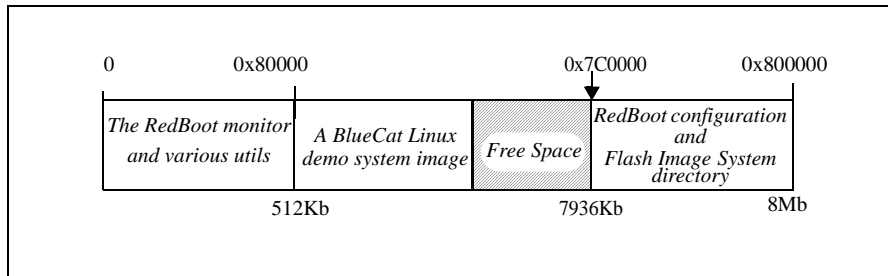


Figure 2-1: Partitioning of the Flash Memory

The following figure shows how the Flash memory on the IQ80321 board is partitioned for BlueCat Linux if the LynuxWorks Boot Loader firmware is used.

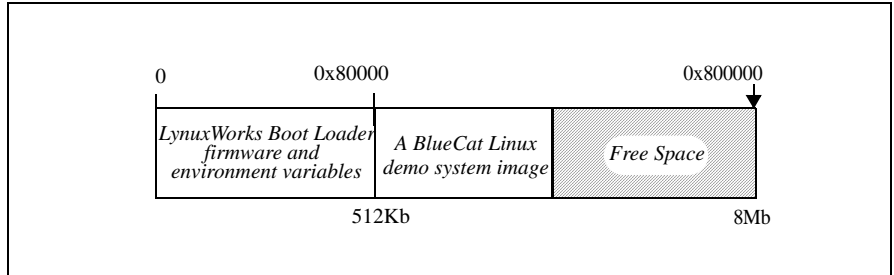


Figure 2-2: Partitioning of the Flash Memory

To download the `osloader` demo system into the target board, the user must follow the steps below:

1. Copy the `i_osloader.kdi` file from the `$BLUECAT_PREFIX/demo/osloader` directory to the `/tftpboot` directory on the development host.
2. Copy the `osloader.kdi` file from the `$BLUECAT_PREFIX/demo/osloader` directory to the `/tftpboot` directory on the development host.
3. Reset the target board.
4. At the RedBoot console, enter the following commands:

```
RedBoot> load -r -b 0x100000 i_osloader.kdi
RedBoot> go 0x100000
```

These commands load the `i_osloader` system to RAM and start it. As a result, the BlueCat OS Loader prompt will appear in the BlueCat console.

5. At the BlueCat OS Loader prompt, type the following commands:

```
> set IF eth0
> set IP target_board_IP
> set HOST development_host_IP
> set FILE tftp osloader.kdi
> exec flash_fdisk /dev/mtdchar0 4-15
> flash /dev/mtdchar1 erase
> reset
```

where *target_board_IP* is an IP address of the target,
development_host_IP is an IP address of the development host.

After these commands have been performed, the `osloader` demo is programmed into Flash and can be booted as described in “Booting a BlueCat Linux System from Flash” section below.

Downloading a BlueCat Linux System into Flash using LynuxWorks Boot Loader

The following procedure is used to download `developer` into the primary Flash of the target board using LynuxWorks Boot Loader:

1. Copy the `developer.kdi` file from the `$BLUECAT_PREFIX/demo/developer` directory to the `/tftpboot` directory on the development host.
2. Reset the target board.
3. At the LynuxWorks Boot Loader console, enter the following commands:

```
iq80321> set flash_device tftp
iq80321> set flash_tftp_file developer.kdi
iq80321> set flash_target flash0
iq80321> set flash_offset 0x80000
iq80321> flash
```

After these commands have been performed, the `developer` demo system is programmed into the primary Flash and can be booted as described in “Booting a BlueCat Linux System from Flash” below.

Booting a BlueCat Linux System from Flash

Booting a BlueCat Linux System from Flash using RedBoot

The following procedure is used to boot the `osloader` demo installed into the Flash memory. For a detailed information on how to install the demo system to Flash, refer to the “Downloading a BlueCat Linux System into Flash using RedBoot” section.

1. Reset the target board.
2. At the RedBoot console, type the following:

```
RedBoot> go 0xf0080000
```

This command will start the `osloader` demo system programmed into Flash.

The IQ80321 board can be configured to start a demo system programmed into Flash automatically at the board power-up. Use the following command to prepare the IQ80321 board to boot BlueCat Linux from Flash automatically:

```
RedBoot> fconfig
Run script at boot: true
Boot script: Enter
Enter script, terminate with empty line
>> go 0xf0080000
>> Enter
Boot script timeout (1000ms resolution): 2
Use BOOTP for network configuration: false
Console baud rate: 115200 Enter
DNS server IP address: Enter
GDB connection port: 9000 Enter
Force console for special debug messages: false Enter
Network debug at boot time: false Enter
Update RedBoot non-volatile configuration - are you sure (y/n)? Y
... Unlock from 0xf07c0000-0xf07c1000: .
... Erase from 0xf07c0000-0xf07c1000: .
... Program from 0x01dde000-0x01ddf000 at 0xf07c0000: .
... Lock from 0xf07c0000-0xf07c1000: .
RedBoot>
```

As a result, the demo system programmed into Flash will be started by the RedBoot monitor automatically on board power-up.

Booting a BlueCat Linux System from Flash using LynuxWorks Boot Loader

The following procedure is used to boot a demo installed into the primary Flash memory. For a detailed information on how to install the demo system to Flash, refer to “Downloading a BlueCat Linux System into Flash using LynuxWorks Boot Loader” on page 10.

1. Reset the target board.
2. At the LynuxWorks Boot Loader console, type the following:

```
iq80321> set boot_device flash0
iq80321> set boot_flash_offset 0x80000
```

```
iq80321> set boot_os BlueCat
iq80321> boot
```

These commands will start the demo system programmed into the primary Flash at offset 0x80000.

The IQ80321 board can be configured to start a demo system programmed into Flash automatically at the board power-up. Use the following commands to prepare the IQ80321 board to boot BlueCat Linux from primary Flash automatically:

```
iq80321> set boot_device flash0
iq80321> set boot_flash_offset 0x80000
iq80321> set boot_os BlueCat
iq80321> set autoboot 1
iq80321> save
```

As a result, the demo system programmed into primary Flash will be started by the Boot Loader monitor automatically on board power-up.

Booting a BlueCat Linux System from a Network

A BlueCat Linux demo system can be booted from a network using either the RedBoot firmware or LynuxWorks Boot Loader firmware.

Booting a BlueCat Linux System from a Network using RedBoot

The following example demonstrates booting the `showcase` demo system over a network using the RedBoot firmware.

1. Copy the `showcase.kdi` file from the `$(BLUECAT_PREFIX)/demo/showcase` directory to the `/tftpboot` directory on the development host.
2. Reset the target board.
3. At the RedBoot console, enter the following commands:

```
RedBoot> load -r -b 0x100000 showcase.kdi
RedBoot> go 0x100000
```

These commands load the `showcase` demo system from a network onto the target board and then automatically start it.

Booting a BlueCat Linux System from a Network using LynuxWorks Boot Loader

The following demonstrates booting the `developer` demo system over a network using the LynuxWorks Boot Loader firmware.

1. Copy the `developer.kdi` file from the `BLUECAT_PREFIX/demo/developer` directory to the `/tftpboot` directory on the cross development host.
2. Reset the target board.
3. At the LynuxWorks Boot Loader console, enter the following commands:

```
iq80321> set boot_device tftp
iq80321> set boot_tftp_file developer.kdi
iq80321> boot
```

These commands load the `developer` demo system from a network onto the target board and then automatically start it.

Booting a BlueCat Linux System from a Network using the OS Loader

The following demonstrates booting the `developer` demo system over a network using the OS loader.

1. Copy the `osloader.srec` file from the `BLUECAT_PREFIX/demo/osloader` directory to the `/tftpboot` directory on the development host.
2. Copy the `showcase.kernel` and `showcase.rfs` files from the `BLUECAT_PREFIX/demo/showcase` directory to the `/tftpboot` directory on the cross development host.
3. Reset the target board.
4. At the LynuxWorks Boot Loader console, enter the following commands:

```
iq80321> set boot_device tftp
iq80321> set boot_tftp_file osloader.kdi
```

```
iq80321> set boot_os BlueCat
iq80321> boot
```

These commands start the `osloader` demo system from RAM. As a result, the BlueCat OS Loader prompt will appear in the BlueCat console.

5. At the BlueCat OS Loader prompt, enter the following commands:

```
> set IF eth0
> set IP target_board_IP
> set HOST development_host_IP
> set KERNEL tftp showcase.kernel
> set RFS tftp showcase.rfs
> set CMD ramdisk_size=16384
> boot
```

where `target_board_IP` is an IP address of the target, and `development_host_IP` is an IP address of the development host.

These commands load the `showcase` demo system from a network onto the target board and then automatically start it.

Kernel Configuration Options

The iq80321 BSP comes with a default BlueCat Linux kernel. This kernel has a number of configuration options. This chapter details these options in the tables listed in Table 3-1: "BlueCat Linux Default Configuration for the iq80321 BSP Distribution" below.

Table 3-1: BlueCat Linux Default Configuration for the iq80321 BSP Distribution

Table Number and Configuration Parameter
Table 3-2: "Code Maturity Level Options"
Table 3-3: "Loadable Module Support"
Table 3-4: "System Type"
Table 3-5: "IQ80321-Specific Options"
Table 3-6: "General Setup"
Table 3-7: "Parallel Port Support"
Table 3-8: "Memory Technology Devices (MTD)"
Table 3-9: "RAM/ROM/Flash Chip Drivers"
Table 3-10: "Mapping Drivers For Chip Access"
Table 3-11: "Self-contained MTD Device Drivers"
Table 3-12: "NAND Flash Device Driver"
Table 3-13: "Plug and Play Configuration"
Table 3-14: "Block Devices"
Table 3-15: "Multidevice Support (RAID and LVM)"
Table 3-16: "Networking Options"
Table 3-17: "QoS and/or Fair Queueing"

Table 3-1: BlueCat Linux Default Configuration for the iq80321 BSP Distribution (Continued)

Table Number and Configuration Parameter
Table 3-18: "Network Device Support"
Table 3-19: "Arcnet Devices"
Table 3-20: "Ethernet (10 or 100Mbit)"
Table 3-21: "Ethernet (1000 Mbit)"
Table 3-22: "Wireless LAN (Non-hamradio)"
Table 3-23: "Token Ring Devices"
Table 3-24: "WAN Interfaces"
Table 3-25: "Amateur Radio Support"
Table 3-26: "IrDA (Infrared) Support"
Table 3-27: "ATA/IDE/MFM/RLL Support"
Table 3-28: "SCSI Support"
Table 3-29: "IEEE 1394 (FireWire) Support (Experimental)"
Table 3-30: "I2O Device Support"
Table 3-31: "ISDN Subsystem"
Table 3-32: "Input Core Support"
Table 3-33: "Character Devices"
Table 3-34: "Serial Drivers"
Table 3-35: "I2C Support"
Table 3-36: "L3 Serial Bus Support"
Table 3-37: "Mice"
Table 3-38: "Joysticks"
Table 3-39: "Watchdog Cards"
Table 3-40: "Ftape, the Floppy Tape Device Driver"
Table 3-41: "Multimedia Devices"
Table 3-42: "File System"
Table 3-43: "Network File Systems"

Table 3-1: BlueCat Linux Default Configuration for the iq80321 BSP Distribution (Continued)

Table Number and Configuration Parameter
Table 3-44: "Partition Types"
Table 3-45: "Sound"
Table 3-46: "Multimedia Capabilities Port Drivers"
Table 3-47: "USB Support"
Table 3-48: "USB Serial Converter Support"
Table 3-49: "Bluetooth Support"
Table 3-50: "Kernel hacking"
Table 3-51: "Modular Advanced Power Management"
Table 3-52: "Messenger Support"

Table 3-2: Code Maturity Level Options

Option	Value	Description
CONFIG_EXPERIMENTAL	Y	Prompt for development and/or incomplete code/drivers
CONFIG_OBSOLETE	N	Prompt for obsolete code/drivers

Table 3-3: Loadable Module Support

Option	Value	Description
CONFIG_MODULES	Y	Enable loadable module support
CONFIG_MODVERSIONS	Y	Set version information on all module symbols
CONFIG_KMOD	Y	Kernel module loader

Table 3-4: System Type

Option	Value	Description
CONFIG_ARCH_IQ80321	Y	ARM system type

Table 3-5: IQ80321-Specific Options

Option	Value	Description
CONFIG_IQ80321_AAU	Y	Support for Application Accelerator Unit (AAU)
CONFIG_IQ80321_DMA	Y	Support for DMA transfers between PCI and local memory
CONFIG_IQ80321_MU	Y	Support for Messaging Unit (MU)
CONFIG_IQ80321_PMON	Y	Support for Performance Monitor Unit (PMU)

Table 3-6: General Setup

Option	Value	Description
CONFIG_ANGELBOOT	N	Load kernel using Angel Debug Monitor
CONFIG_BLUECAT_IGNORE_PRINTK	N	BlueCat Ignore printk
CONFIG_BLUECAT_THUMB	N	BlueCat Kernel support for THUMB binaries
CONFIG_BLUECAT_LOADER	N	BlueCat OS Loader
CONFIG_BLUECAT_SMALL_FOOTPRINT	N	BlueCat small memory footprint
CONFIG_PCI_NAMES	N	PCI device name database
CONFIG_HOTPLUG	N	Support hot-pluggable devices
CONFIG_NET	Y	Networking support
CONFIG_BLUECAT_MEMSIZE	N	Memory sizing benchmarks

Table 3-6: General Setup (Continued)

Option	Value	Description
CONFIG_SYSVIPIC	Y	System V IPC
CONFIG_BSD_PROCESS_ACCT	N	BSD Process Accounting
CONFIG_SYSCTL	N	Sysctl support
CONFIG_FPE_NWFPE	Y	NWFPE math emulation
CONFIG_FPE_FASTFPE	N	FastFPE math emulation (Experimental)
CONFIG_KCORE_ELF	Y	Kernel core (/proc/kcore) format
CONFIG_BINFMT_AOUT	N	Kernel support for a.out binaries
CONFIG_BINFMT_ELF	Y	Kernel support for ELF binaries
CONFIG_BINFMT_MISC	N	Kernel support for MISC binaries
CONFIG_PM	N	Power Management support (Experimental)
CONFIG_ARTHUR	N	RISC OS personality
CONFIG_ALIGNMENT_TRAP	Y	Kernel-mode alignment trap handler

Table 3-7: Parallel Port Support

Option	Value	Description
CONFIG_PARPORT	N	Parallel port support

Table 3-8: Memory Technology Devices (MTD)

Option	Value	Description
CONFIG_MTD	Y	Memory Technology Device (MTD) support
CONFIG_MTD_DEBUG	N	Debugging
CONFIG_MTD_PARTITIONS	Y	MTD partitioning support
CONFIG_MTD_REDBOOT_PARTS	N	RedBoot partition table parsing

Table 3-8: Memory Technology Devices (MTD) (Continued)

Option	Value	Description
CONFIG_MTD_BOOTLDR_PARTS	N	Compaq bootldr partition table parsing
CONFIG_MTD_AFS_PARTS	N	ARM Firmware Suite partition parsing
CONFIG_MTD_CHAR	Y	Direct char device access to MTD devices
CONFIG_MTD_BLOCK	Y	Caching block device access to MTD devices
CONFIG_FTL	N	FTL (Flash Translation Layer) support
CONFIG_NFTL	N	NFTL (NAND Flash Translation Layer) support

Table 3-9: RAM/ROM/Flash Chip Drivers

Option	Value	Description
CONFIG_MTD_CFI	Y	Detect Flash chips by Common Flash Interface (CFI) probe
CONFIG_MTD_JEDEC_PROBE	N	Detect non-CFI AMD/JEDEC-compatible Flash chips
CONFIG_MTD_CFI_ADV_OPTIONS	N	Flash chip driver advanced configuration options
CONFIG_MTD_CFI_INTELEXT	Y	Support for Intel/Sharp Flash chips
CONFIG_MTD_CFI_AMDSTD	N	Support for AMD/Fujitsu Flash chips
CONFIG_MTD_RAM	N	Support for RAM chips in bus mapping
CONFIG_MTD_ROM	N	Support for ROM chips in bus mapping
CONFIG_MTD_ABSENT	N	Support for absent chips in bus mapping
CONFIG_MTD_OBSOLETE_CHIPS	N	Older (theoretically obsoleted now) drivers for non-CFI chips

Table 3-10: Mapping Drivers For Chip Access

Option	Value	Description
CONFIG_MTD_PHYSMAP	N	CFI Flash device in physical memory map
CONFIG_MTD_NORA	N	CFI Flash device mapped on Nora
CONFIG_MTD_ARM_INTEGRATOR	N	CFI Flash device mapped on ARM Integrator/P720T
CONFIG_MTD_CDB89712	N	Cirrus CDB89712 evaluation board mappings
CONFIG_MTD_PCI	N	PCI MTD driver
CONFIG_MTD_INTEL_CS	Y	Flash chip mapping on the IQ80321 board
CONFIG_MTD_INTEL_CS_PART	:	Partitions layout

Table 3-11: Self-contained MTD Device Drivers

Option	Value	Description
CONFIG_MTD_PMC551	N	Ramix PMC551 PCI Mezzanine RAM card support
CONFIG_MTD_SDRAM	N	Uncached system RAM
CONFIG_MTD_MTDRAW	N	Test driver using RAM
CONFIG_MTD_BLKMTD	N	MTD emulation using block device
CONFIG_MTD_DOC1000	N	M-Systems Disk-On-Chip 1000
CONFIG_MTD_DOC2000	N	M-Systems Disk-On-Chip 2000 and Millennium
CONFIG_MTD_DOC2001	N	M-Systems Disk-On-Chip Millennium-only alternative driver (see help)

Table 3-12: NAND Flash Device Driver

Option	Value	Description
CONFIG_MTD_NAND	N	NAND Device Support

Table 3-13: Plug and Play Configuration

Option	Value	Description
CONFIG_PNP	N	Plug and Play support

Table 3-14: Block Devices

Option	Value	Description
CONFIG_BLK_DEV_FD	N	Normal PC floppy disk support
CONFIG_BLK_CPQ_DA	N	Compaq SMART2 support
CONFIG_BLK_CPQ_CISS_DA	N	Compaq Smart Array 5xxx support
CONFIG_BLK_DEV_DAC960	N	Mylex DAC960/DAC1100 PCI RAID Controller support
CONFIG_BLK_DEV_LOOP	N	Loopback device support
CONFIG_BLK_DEV_NBD	N	Network block device support
CONFIG_BLK_DEV_RAM	Y	RAM disk support
CONFIG_BLK_DEV_RAM_SIZE	28472	Default RAM disk size
CONFIG_BLK_DEV_INITRD	N	Initial RAM disk (initrd) support
CONFIG_BLUECAT_RFS	Y	BlueCat RFS support

Table 3-15: Multidevice Support (RAID and LVM)

Option	Value	Description
CONFIG_MD	N	Multiple devices driver support (RAID and LVM)
CONFIG_MD_LINEAR	N	Linear (append) mode
CONFIG_MD_RAID0	N	RAID-0 (striping) mode
CONFIG_MD_RAID1	N	RAID-1 (mirroring) mode
CONFIG_MD_RAID5	N	RAID-4/RAID-5 mode
CONFIG_MD_MULTIPATH	N	Multipath I/O support

Table 3-16: Networking Options

Option	Value	Description
CONFIG_PACKET	N	Packet socket
CONFIG_NETLINK	N	Kernel/User netlink socket
CONFIG_NETFILTER	N	Network packet filtering (replaces ipchains)
CONFIG_FILTER	N	Socket Filtering
CONFIG_UNIX	Y	Unix domain sockets
CONFIG_INET	Y	TCP/IP networking
CONFIG_IP_MULTICAST	N	IP: multicasting
CONFIG_IP_ADVANCED_ROUTER	N	IP: advanced router
CONFIG_IP_PNP	N	IP: kernel level autoconfiguration
CONFIG_NET_IPIP	N	IP: tunneling
CONFIG_NET_IPGRE	N	IP: GRE tunnels over IP
CONFIG_INET_ECN	N	IP: TCP Explicit Congestion Notification support
CONFIG_SYN_COOKIES	N	IP: TCP syncookie support (disabled per default)

Table 3-16: Networking Options (Continued)

Option	Value	Description
CONFIG_IPV6	N	The IPv6 protocol (Experimental)
CONFIG_KHTTPD	N	Kernel httpd acceleration (Experimental)
CONFIG_ATM	N	Asynchronous Transfer Mode (ATM) (Experimental)
CONFIG_IPX	N	The IPX protocol
CONFIG_ATAALK	N	Appletalk protocol support
CONFIG_DECNET	N	DECnet Support
CONFIG_BRIDGE	N	802.1d Ethernet Bridging
CONFIG_X25	N	CCITT X.25 Packet Layer (Experimental)
CONFIG_LAPB	N	LAPB Data Link Driver (Experimental)
CONFIG_LLC	N	802.2 LLC (Experimental)
CONFIG_NET_DIVERT	N	Frame Diverter (Experimental)
CONFIG_ECONET	N	Acorn Econet/AUN protocols (Experimental)
CONFIG_WAN_ROUTER	N	WAN router
CONFIG_NET_FASTROUTE	N	Fast switching (read help!)
CONFIG_NET_HW_FLOWCONTROL	N	Forwarding between high speed interfaces

Table 3-17: QoS and/or Fair Queueing

Option	Value	Description
CONFIG_NET_SCHED	N	QoS and/or fair queueing

Table 3-18: Network Device Support

Option	Value	Description
CONFIG_NETDEVICES	Y	Network device support?

Table 3-19: Arcnet Devices

Option	Value	Description
CONFIG_ARCNET	N	ARCnet support
CONFIG_DUMMY	N	Dummy net driver support
CONFIG_BONDING	N	Bonding driver support
CONFIG_EQUALIZER	N	EQL (serial line load balancing) support
CONFIG_TUN	N	Universal TUN/TAP device driver support

Table 3-20: Ethernet (10 or 100Mbit)

Option	Value	Description
CONFIG_NET_ETHERNET	N	Ethernet (10 or 100Mbit)

Table 3-21: Ethernet (1000 Mbit)

Option	Value	Description
CONFIG_ACENIC	N	Alteon AceNIC/3Com 3C985/NetGear GA620 Gigabit support
CONFIG_DL2K	N	D-Link DL2000-based Gigabit Ethernet support
CONFIG_NS83820	N	National Semiconduct DP83820 support
CONFIG_HAMACHI	N	Packet Engines Hamachi GNIC-II support
CONFIG_YELLOWFIN	N	Packet Engines Yellowfin Gigabit-NIC support (Experimental)
CONFIG_SK98LIN	N	SysKonnect SK-98xx support
CONFIG_E1000	Y	Intel PRO/1000 support
CONFIG_FDDI	N	FDDI driver support
CONFIG_HIPPI	N	HIPPI driver support (Experimental)

Table 3-21: Ethernet (1000 Mbit) (Continued)

Option	Value	Description
CONFIG_PPP	N	PPP (point-to-point protocol) support
CONFIG_SLIP	N	SLIP (serial line) support

Table 3-22: Wireless LAN (Non-hamradio)

Option	Value	Description
CONFIG_NET_RADIO	N	Wireless LAN (non-hamradio)

Table 3-23: Token Ring Devices

Option	Value	Description
CONFIG_TR	N	Token Ring driver support
CONFIG_NET_FC	N	Fibre Channel driver support
CONFIG_RCPCI	N	Red Creek Hardware VPN (Experimental)
CONFIG_SHAPER	N	Traffic Shaper (Experimental)

Table 3-24: WAN Interfaces

Option	Value	Description
CONFIG_WAN	N	Wan interfaces support

Table 3-25: Amateur Radio Support

Option	Value	Description
CONFIG_HAMRADIO	N	Amateur Radio support

Table 3-26: IrDA (Infrared) Support

Option	Value	Description
CONFIG_IRDA	N	IrDA subsystem support

Table 3-27: ATA/IDE/MFM/RLL Support

Option	Value	Description
CONFIG_IDE	N	ATA/IDE/MFM/RLL support

Table 3-28: SCSI Support

Option	Value	Description
CONFIG_SCSI	N	SCSI support?

Table 3-29: IEEE 1394 (FireWire) Support (Experimental)

Option	Value	Description
CONFIG_IEEE1394	N	IEEE 1394 (Firewire) Support (Experimental)

Table 3-30: I2O Device Support

Option	Value	Description
CONFIG_I2O	N	I2O support

Table 3-31: ISDN Subsystem

Option	Value	Description
CONFIG_ISDN	N	ISDN support

Table 3-32: Input Core Support

Option	Value	Description
CONFIG_INPUT	N	Input core support
CONFIG_INPUT_MOUSEDEV_SCREEN_X	1024	Horizontal screen resolution
CONFIG_INPUT_MOUSEDEV_SCREEN_Y	768	Vertical screen resolution

Table 3-33: Character Devices

Option	Value	Description
CONFIG_VT	N	Virtual terminal
CONFIG_SERIAL	Y	Standard/generic (8250/16550 and compatible UARTs) serial support
CONFIG_SERIAL_CONSOLE	Y	Support for console on serial port
CONFIG_SERIAL_EXTENDED	N	Extended dumb serial driver options
CONFIG_SERIAL_NONSTANDARD	N	Nonstandard serial port support

Table 3-34: Serial Drivers

Option	Value	Description
CONFIG_SERIAL_8250	N	8250/16550 and compatible serial support (Experimental)

Table 3-35: I2C Support

Option	Value	Description
CONFIG_I2C	Y	I2C support
CONFIG_I2C_ALGOBIT	N	I2C bit-banging interfaces
CONFIG_I2C_ALGOPCF	N	I2C PCF 8584 interfaces
CONFIG_I2C_ALGO_80321	Y	Intel 80321 I2C algorithm
CONFIG_I2C_ADAP_80321	Y	Intel 80321 I2C adapter
CONFIG_I2C_CHARDEV	N	I2C device interface
CONFIG_I2C_PROC	N	I2C /proc interface (required for hardware sensors)

Table 3-36: L3 Serial Bus Support

Option	Value	Description
CONFIG_L3	N	L3 support

Table 3-37: Mice

Option	Value	Description
CONFIG_BUSMOUSE	N	Bus Mouse Support
CONFIG_MOUSE	N	Mouse Support (not serial and bus mice)

Table 3-38: Joysticks

Option	Value	Description
CONFIG_QIC02_TAPE	N	QIC-02 tape support

Table 3-39: Watchdog Cards

Option	Value	Description
CONFIG_WATCHDOG	N	Watchdog Timer Support
CONFIG_INTEL_RNG	N	Intel i8x0 Random Number Generator support
CONFIG_NVRAM	N	/dev/nvram support
CONFIG_RTC	N	Enhanced Real Time Clock Support
CONFIG_RTC_DS1307	N	DS1307 Real Time Clock
CONFIG_DTLK	N	Double Talk PC internal speech card support
CONFIG_R3964	N	Siemens R3964 line discipline
CONFIG_APPLICOM	N	Applicom intelligent fieldbus card support

:

Table 3-40: Ftape, the Floppy Tape Device Driver

Option	Value	Description
CONFIG_FTAPE	N	Ftape (QIC-80/Travan) support
CONFIG_AGP	N	/dev/agpgart (AGP Support)
CONFIG_DRM	N	Direct Rendering Manager (XFree86 DRI support)
CONFIG_MWAVE	N	ACP Modem (Mwave) support

Table 3-41: Multimedia Devices

Option	Value	Description
CONFIG_VIDEO_DEV	N	Video For Linux

Table 3-42: File System

Option	Value	Description
CONFIG_QUOTA	N	Quota support
CONFIG_AUTOFS_FS	N	Kernel automounter support
CONFIG_AUTOFS4_FS	N	Kernel automounter version 4 support (also supports v3)
CONFIG_REISERFS_FS	N	Reiserfs support
CONFIG_ADFS_FS	N	ADFS file system support
CONFIG_AFFS_FS	N	Amiga FFS file system support (Experimental)
CONFIG_HFS_FS	N	Apple Macintosh file system support (Experimental)
CONFIG_BFS_FS	N	BFS file system support (Experimental)
CONFIG_CMS_FS	N	CMS file system support (Experimental)
CONFIG_EXT3_FS	N	Ext3 journalling file system support (Experimental)
CONFIG_FAT_FS	N	DOS FAT fs support
CONFIG_EFS_FS	N	EFS file system support (read only) (Experimental)
CONFIG_JFFS_FS	Y	Journalling Flash File System (JFFS) support
CONFIG_JFFS_FS_VERBOSE	0	JFFS debugging verbosity (0 = quiet, 3 = noisy)
CONFIG_JFFS_PROC_FS	N	JFFS stats available in /proc filesystem
CONFIG_JFFS2_FS	N	Journalling Flash File System v2 (JFFS2) support
CONFIG_CRAMFS	N	Compressed ROM file system support

Table 3-42: File System (Continued)

Option	Value	Description
CONFIG_TMPFS	N	Virtual memory file system support (former shm fs)
CONFIG_RAMFS	N	Simple RAM-based file system support
CONFIG_ISO9660_FS	N	ISO 9660 CDROM file system support
CONFIG_MINIX_FS	N	Minix fs support
CONFIG_FREEVXFS_FS	N	FreeVxFS file system support (VERITAS VxFS(TM) compatible)
CONFIG_NTFS_FS	N	NTFS file system support (read only)
CONFIG_HPFS_FS	N	OS/2 HPFS file system support
CONFIG_PROC_FS	Y	/proc file system support
CONFIG_DEVFS_FS	N	/dev file system support (Experimental)
CONFIG_DEVPTS_FS	Y	/dev/pts file system for Unix98 PTYs
CONFIG_QNX4FS_FS	N	QNX4 file system support (read only) (Experimental)
CONFIG_ROMFS_FS	N	ROM file system support
CONFIG_EXT2_FS	Y	Second extended fs support
CONFIG_SYSV_FS	N	System V/Xenix/V7/Coherent file system support
CONFIG_UDF_FS	N	UDF file system support (read only)
CONFIG_UFS_FS	N	UFS file system support (read only)

Table 3-43: Network File Systems

Option	Value	Description
CONFIG_CODA_FS	N	Coda file system support (advanced network fs)
CONFIG_INTERMEZZO_FS	N	InterMezzo file system support (Experimental, replicating fs)
CONFIG_NFS_FS	Y	NFS file system support
CONFIG_NFS_V3	N	Provide NFSv3 client support
CONFIG_NFSD	N	NFS server support
CONFIG_SMB_FS	N	SMB file system support (to mount Windows shares etc.)
CONFIG_NCP_FS	N	NCP file system support (to mount NetWare volumes)

Table 3-44: Partition Types

Option	Value	Description
CONFIG_PARTITION_ADVANCED	Y	Advanced partition selection
CONFIG_ACORN_PARTITION	N	Acorn partition support
CONFIG_OSF_PARTITION	N	Alpha OSF partition support
CONFIG_AMIGA_PARTITION	N	Amiga partition table support
CONFIG_ATARI_PARTITION	N	Atari partition table support
CONFIG_MAC_PARTITION	N	Macintosh partition map support
CONFIG_MSDOS_PARTITION	N	PC BIOS (MSDOS partition tables)
CONFIG_LDM_PARTITION	N	Windows Logical Manager (Dynamic Disk) support
CONFIG_SGI_PARTITION	N	SGI partition support
CONFIG_ULTRIX_PARTITION	N	Ultrix partition table support
CONFIG_SUN_PARTITION	N	Sun partition table support

Table 3-45: Sound

Option	Value	Description
CONFIG_SOUND	N	Sound support

Table 3-46: Multimedia Capabilities Port Drivers

Option	Value	Description
CONFIG_MCP	N	Multimedia drivers
CONFIG_MCP_UCB1200_TS	N	Touchscreen interface support

Table 3-47: USB Support

Option	Value	Description
CONFIG_USB	N	Support for USB
CONFIG_USB_STORAGE_DEBUG	N	USB Mass Storage verbose debug
CONFIG_USB_STORAGE_DATAFAB	N	Datafab MDCFE-B Compact Flash Reader
CONFIG_USB_STORAGE_FREECOM	N	Freecom USB/ATAPI Bridge support
CONFIG_USB_STORAGE_ISD200	N	ISD-200 USB/ATA Bridge support
CONFIG_USB_STORAGE_JUMPSHOT	N	Lexar Jumpshot Compact Flash Reader
CONFIG_USB_STORAGE_DPCM	N	Microtech CompactFlash/SmartMedia reader
CONFIG_USB_STORAGE_HP8200e	N	HP CD-Writer 82xx support
CONFIG_USB_STORAGE_SDDR09	N	SanDisk SDDR-09 (and other SmartMedia) support

Table 3-48: USB Serial Converter Support

Option	Value	Description
CONFIG_USB_SERIAL_GENERIC	N	USB Generic Serial Driver
CONFIG_USB_SERIAL_BELKIN	N	USB Belkin and Peracom Single Port Serial Driver (Experimental)
CONFIG_USB_SERIAL_WHITEHEAT	N	USB ConnectTech WhiteHEAT Serial Driver (Experimental)
CONFIG_USB_SERIAL_DIGI_ACCELEPORT	N	USB Digi International AccelePort USB Serial Driver
CONFIG_USB_SERIAL_EMPEG	N	USB Empeg empeg-car Mark I/II Driver (Experimental)
CONFIG_USB_SERIAL_FTDI_SIO	N	USB FTDI Single Port Serial Driver (Experimental)
CONFIG_USB_SERIAL_VISOR	N	USB Handspring Visor /Palm m50x/Sony Clie Driver
CONFIG_USB_SERIAL_IR	N	USB IR Dongle Serial Driver (Experimental)
CONFIG_USB_SERIAL_EDGEPORT	N	USB Inside Out Edgeport Serial Driver (Experimental)
CONFIG_USB_SERIAL_KEYSPAN_PDA	N	USB Keyspan PDA Single Port Serial Driver (Experimental)
CONFIG_USB_SERIAL_KEYSPAN	N	USB Keyspan USA-xxx Serial Driver (Experimental)
CONFIG_USB_SERIAL_KEYSPAN_USA28	N	USB Keyspan USA-28 Firmware
CONFIG_USB_SERIAL_KEYSPAN_USA28X	N	USB Keyspan USA-28X Firmware
CONFIG_USB_SERIAL_KEYSPAN_USA28XA	N	USB Keyspan USA-28XA Firmware
CONFIG_USB_SERIAL_KEYSPAN_USA28XB	N	USB Keyspan USA-28XB Firmware
CONFIG_USB_SERIAL_KEYSPAN_USA19	N	USB Keyspan USA-19Firmware
CONFIG_USB_SERIAL_KEYSPAN_USA18X	N	USB Keyspan USA-18X Firmware
CONFIG_USB_SERIAL_KEYSPAN_USA19W	N	USB Keyspan USA-19W Firmware
CONFIG_USB_SERIAL_KEYSPAN_USA49W	N	USB Keyspan USA-49W Firmware

Table 3-48: USB Serial Converter Support

Option	Value	Description
CONFIG_USB_SERIAL_MCT_U232	N	USB MCT Single Port Serial Driver (Experimental)
CONFIG_USB_SERIAL_PL2303	N	USB Prolific 2303 Single Port Serial Driver (Experimental)
CONFIG_USB_SERIAL_CYBERJACK	N	USB REINER SCT cyberJack pinpad/e-com chipcard reader (Experimental)
CONFIG_USB_SERIAL_XIRCOM	N	USB Xircom/ Entegra Single Port Serial Driver (Experimental)
CONFIG_USB_SERIAL_OMNINET	N	USB ZyXEL omni.net LCD Plus Drive (Experimental)

Table 3-49: Bluetooth Support

Option	Value	Description
CONFIG_BLUEZ	N	Bluetooth subsystem support

Table 3-50: Kernel hacking

Option	Value	Description
CONFIG_NO_FRAME_POINTER	Y	Compile kernel without frame pointer
CONFIG_DEBUG_ERRORS	N	Verbose kernel error messages
CONFIG_DEBUG_USER	N	Verbose user fault messages
CONFIG_DEBUG_INFO	N	Include debugging information in kernel binary
CONFIG_DEBUG_SLAB	N	Debug memory allocations
CONFIG_MAGIC_SYSRQ	N	Magic SysRq key
CONFIG_BLUECAT_KDBG	N	Include kdbg kernel debugger

Table 3-50: Kernel hacking

Option	Value	Description
CONFIG_DEBUG_SPINLOCK	N	Spinlock debugging
CONFIG_DEBUG_LL	N	Kernel low-level debugging functions

Table 3-51: Modular Advanced Power Management

Option	Value	Description
CONFIG_BLUECAT_APM	N	MAPM support

Table 3-52: Messenger Support

Option	Value	Description
CONFIG_BLUECAT_IOPMAN	N	Enable IOP Manager support
CONFIG_BLUECAT_MSNG	N	Enable Messenger support

This chapter provides information about BlueCat Linux demo systems supported by the iq80321 Board Support Package (BSP).

Demo Systems

Table 4-1 lists the demo systems supported in the iq80321 BSP distribution, the boot devices supported by each demo system, and their respective RAM and ROM requirements.

Table 4-1: Demo Systems Supported by iq80321 BSP

Demo	Boot Devices Supported by Default	ROM Requirements	RAM Requirements
developer	Network (using LynuxWorks Boot Loader) Network (using RedBoot)	3582 KB	17920 KB
osloader	Flash (using LynuxWorks Boot Loader) Network (using LynuxWorks Boot Loader) Network (using RedBoot)	784 KB	6656 KB
showcase	Flash (using LynuxWorks Boot Loader) Network (using LynuxWorks Boot Loader) Network (using RedBoot)	2709 KB	16384 KB
zebra	Network (using LynuxWorks Boot Loader) Network (using RedBoot)	3606 KB	18432 KB

developer Demo System

The `developer` demo system is a package consisting of the functionalities of the `shell`, `ftp`, `ping`, `gdb`, and `v1_demo` systems. Refer to Chapter 4 of the *BlueCat*

Linux User's Guide for descriptions of developer and its component demo systems.

osloader Demo System

`osloader` is the BlueCat OS Loader system used to boot a BlueCat Linux system on the target board. Refer to Chapter 4 of the *BlueCat Linux User's Guide* for details.

showcase Demo System

The `showcase` demo system starts and configures the Apache HTTP daemon turning the target board into a Web server. Refer to Chapter 4 of the *BlueCat Linux User's Guide* for details.

zebra Demo System

The `zebra` demo system demonstrates the Zebra routing mechanism. Since the `zebra` demo system is specific to the IQ80321 board its description is not included in the *BlueCat Linux User's Guide*.

SYNOPSIS

The Zebra Route Server functionality demonstration.

REQUIREMENTS

Storage	Tiny
RAM	Small
Network	Yes
Disk	None
Special	Three machines: Demo Machine (IQ80321 target) and two test machines (Test1 and Test2) are connected by a twisted-pair cable to an Ethernet HUB. Although Test1 Machine and Test2 Machine are in the same physical network, they belong to different IP subnets. Demo Machine is running BlueCat Linux. Test1 Machine and Test2 Machine are running Red Hat Linux 7.1.

Kernel Option

On all machines the `CONFIG_IP_MULTICAST` and `CONFIG_NETLINK` options must be set to `Y`.

DESCRIPTION

GNU Zebra is a free software distributed under GNU Generic Public License that manages a TCP/IP-based routing services with routing protocol support such as BGP4, BGP4+, OSPFv2, OSPFv3, RIPv1, RIPv2, and RIPng.

Routing is a process by which a host with multiple network connection decides where to deliver the IP packets that it has received. Each host keeps a special list of routing rules called a routing table. This table contains rows which typically contain at least three fields: a destination address, the name of the interface where the packet is to be routed, and optionally the IP address of another machine that carries the packet on its next step through the network. The routing process is very simple. The incoming packet is received, the destination address is examined, and then it is compared with each entry in the table. The entry that best matches the address is selected, and the packet is forwarded to the specified interface. If the gateway field is filled, then the packet is forwarded to that host via the specified interface. The destination address is otherwise assumed to be on the network supported by the interface.

There are two types of routing: static (manual) routing and dynamic routing. A static routing “algorithm” is simply a definition of the routing table established by the network administrator in order to allow routing on the network. The routing table never changes without a manual update by the administrator. A dynamic routing is a technique developed to automatically adjust routing tables in the event of network failures. The most common dynamic routing protocols are Routing Information Protocol (RIP) and Open Shortest Path First (OSPF). RIP is very common on small networks. OSPF is capable at handling large network configurations, and it is suited to environments where there is a large number of possible paths through the network.

With Zebra installed, the machine exchanges routing information with other routers using routing protocols.

Traditional routing software is made as a one process program which provides all of the routing protocol functionalities. Zebra takes a different approach. It is made from a collection of several daemons that work together to build a routing table. There is no need for these daemons to be running on the same machine. There may be several protocol-specific routing daemons and Zebra that acts as the kernel routing manager. It is easy to add a new routing protocol daemons to the entire routing system without affecting any other software by running only the protocol

daemon associated with routing protocols in use. Thus, the user may run a specific daemon and send routing reports to a central routing console.

A `zebra` demo system is intended to demonstrate advantages of Zebra's modularity in configuring a network using the RIP and OSPF routing protocols. Although only these two popular protocols are included in `zebra`, the demo system can easily be extended with support of any routing protocol from a number of protocols supported in the Zebra software. To do such an extension, the user must create configuration files for desired routing protocols and then update the demo spec file to include the daemons from the Zebra software and created configuration files. For more information on how to create the configuration files, the user is referred to the Zebra documentation in the `$BLUECAT_PREFIX/cdt/doc/zebra_trg-0.91a` directory.

In the `zebra` demo, two network interfaces are necessary on Demo Machine to support Test1 Machine and Test2 Machine that belong to different subnets. Demo Machine has two Ethernet interfaces via a built-in Intel Gigabit Ethernet adapter and via a plug-in Intel Ethernet Express Pro adapter. The following table shows networks settings used in the `zebra` demo system.

Machine Name	IP Address	Subnet Mask
Demo Machine		
eth0	172.17.3.9	255.255.0.0
eth1	172.21.0.1	255.255.0.0
Test1 Machine	172.17.0.5	255.255.0.0
Test2 Machine	172.21.0.2	255.255.0.0

The following procedure describes necessary steps to install, configure, run, and test the `zebra` demo system.

1. On Demo Machine, bring up the network interface for Ethernet Device 2 manually using the `ifconfig` command:

```
bash# ifconfig eth1 172.21.0.1 netmask\  
255.255.0.0
```

2. On Demo Machine, start the RIP and OSPF network protocols by running the `ripd` and `ospfd` daemons:

```
bash# ripd -d  
bash# ospfd -d
```

3. Make sure that both Ethernet Device 1 and Ethernet Device 2 on Demo Machine have been started by entering the `ifconfig -a` command:

```
bash# ifconfig -a

eth0      Link encap:Ethernet  HWaddr 00:02:B3:28:26:17
          inet addr:172.17.3.9  Bcast:172.17.255.255 Mask:255.255.0.0
          inet6 addr: fe80::202:b3ff:fe28:2617/10 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:54 errors:0 dropped:0 overruns:0 frame:0
          TX packets:40 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:29

eth1      Link encap:Ethernet  HWaddr 00:07:E9:03:38:68
          inet addr:172.21.0.1  Bcast:172.21.255.255 Mask:255.255.0.0
          inet6 addr: fe80::207:e9ff:fe03:3868/10 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:42 errors:0 dropped:0 overruns:0 frame:0
          TX packets:18 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:27 Memory:80000000-80020000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          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

sit0      Link encap:IPv6-in-IPv4
          NOARP  MTU:1480  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
```

4. On the Test1 and Test2 machines, install the `zebra` RPM from the Red Hat Linux 7.1 distribution if it has not been installed already:

```
bash# rpm -i zebra-0.91a-3.i386.rpm
```

5. On Test1 Machine and Test2 Machine, configure `zebra` by creating the `zebra.conf`, `ripd.conf`, and `ospfd.conf` configuration files in the `/etc/zebra` directory with the following contents:

Configuration File	Test 1 Machine	Test 2 Machine
<code>zebra.conf</code>	<pre>password zebra enable password zebra interface lo ip address 127.0.0.1/8 interface eth0 ip address 172.17.0.5/16 log stdout</pre>	<pre>password zebra enable password zebra interface lo ip address 127.0.0.1/8 interface eth0 ip address 172.21.0.2/16 log syslog</pre>
<code>ripd.conf</code>	<pre>hostname ripd password zebra enable password zebra log syslog interface eth0 interface eth1 router rip network 172.20.0.0/16 network 172.17.0.0/16 redistribute connected</pre>	<pre>hostname ripd password zebra enable password zebra log syslog interface eth0 router rip network 172.21.0.0/16</pre>
<code>ospfd.conf</code>	<pre>hostname ospfd password zebra enable password zebra router ospf network 172.17.0.0/16 area 0 network 172.20.0.0/16 area 2 redistribute connected log syslog</pre>	<pre>hostname ospfd password zebra enable password zebra router ospf network 172.21.0.0/16 area 1 log syslog</pre>

6. On Test1 Machine and Test2 Machine, start `zebra` by typing the following command:

```
bash# zebra -d
```

To demonstrate the RIP or OSPF routing protocols, a respective daemon must be started on a Test Machine:

7. To check the RIP protocol, start the `ripd` daemon on a Test machine:

```
bash# ripd -d
```

or, to check the OSPF protocol, start the `ospfd` daemon on a Test machine:

```
bash# ospfd -d
```

After starting either daemon, a Test Machine exchanges information using a respective protocol to create and update the IP routing table. This

allows the user to get access to a Test Machine that belongs to another IP subnet. No additional efforts are required from the user to access another subnet.

8. Use the `ping` utility to check a network connectivity. To `ping` from Test1 Machine to Test2 Machine, type the following:

```
bash# ping 172.21.0.2
```

9. To examine the IP routing tables on the Test machines, use the `netstat` utility:

- On Test 1 Machine:

```
bash# netstat -nr
```

Destination	Gateway	Genmask	Flags	MSS Window	irrtt	Iface
172.17.0.0	0.0.0.0	255.255.0.0	U	40 0	0	eth0
172.20.0.0	0.0.0.0	255.255.0.0	U	40 0	0	eth1
172.21.0.0	172.17.3.9	255.255.0.0	UG	40 0	0	eth0
127.0.0.0	0.0.0.0	255.0.0.0	U	40 0	0	lo
0.0.0.0	172.17.0.1	0.0.0.0	UG	40 0	0	eth0

The above example shows that the IP address 172.21.0.0 and the respective gateway 172.17.3.9 exist in the table.

- On Test 2 Machine:

```
bash# netstat -nr
```

Destination	Gateway	Genmask	Flags	MSSWindow	irrtt	Iface
172.17.0.0	172.21.0.1	255.255.0.0	UG	40 0	0	eth0
172.20.0.0	172.21.0.1	255.255.0.0	UG	40 0	0	eth0
172.21.0.0	0.0.0.0	255.255.0.0	U	40 0	0	eth0

The above example shows that the IP address 172.17.0.0 and the respective gateway 172.21.0.1 exist in the table.

Using Selected RPM Packages

This section provides a description on how to use selected RPM packages that are frequently deployed in the embedded systems environment.

Using BusyBox RPM Package

The BusyBox RPM package combines tiny versions of many common UNIX utilities into a single small executable. It provides minimalist replacements for most of the utilities that can be usually found in `fileutils`, `shellutils`,

findutils, textutils, grep, gzip, tar, etc. BusyBox provides a fairly complete POSIX environment for any small or embedded system.

The utilities in BusyBox generally have fewer options than their full-featured GNU cousins. However, the options that are included provide the expected functionality and behave very much like their GNU counterparts.

This section describes the steps necessary for creating and booting a BlueCat Linux system containing BusyBox and demonstrates use of the BusyBox utilities.

Creating a BlueCat Linux System for BusyBox

To create a BlueCat Linux image for BusyBox, perform the following steps:

1. Create a new directory by typing:

```
BlueCat:$ mkdir -p \  
$BLUECAT_PREFIX/demo/busybox/local
```

2. Set up the BlueCat Linux kernel configuration by using the standard kernel configuration tools and copy kernel configuration file to the \$BLUECAT_PREFIX/demo/busybox directory. For instance, type the following commands:

```
BlueCat:$ cd $BLUECAT_PREFIX/usr/src/linux  
BlueCat:$ make xconfig
```

Select **Save and Exit** to update the .config file, then type the following commands:

```
BlueCat:$ cp .config \  
$BLUECAT_PREFIX/demo/busybox/busybox.config
```

NOTE: The kernel config file for the developer demo (\$BLUECAT_PREFIX/demo/developer/developer.config) is also recommended as a starting point.

3. Create the BlueCat kernel downloadable image (busybox.kernel):

```
BlueCat:$ cd $BLUECAT_PREFIX/demo/busybox  
BlueCat:$ mkkernel ./busybox.config \  
./busybox.kernel ./busybox.disk
```

4. Create a spec file (busybox.spec) that contains the following minimal directives:

```

strip on

mkdir /dev
mknod /dev/console c 5 1

mkdir /lib
mkdir -p /usr/lib
mkdir /bin
mkdir /sbin
mkdir -p /etc/rc.d
mkdir /proc

cp ./local/fstab ./local/inittab /etc
cp ./local/rc.sysinit /etc/rc.d

lcd ${BLUECAT_PREFIX}/sbin
cp reboot busybox /sbin

ln -s /sbin/busybox /sbin/init
ln -s /sbin/busybox /sbin/ifconfig
ln -s /sbin/busybox /sbin/route
ln -s /sbin/busybox /bin/mount
ln -s /sbin/busybox /bin/sh
ln -s /sbin/busybox /bin/ping

chmod 711 /etc/rc.d/rc.sysinit
chmod 755 /bin /sbin
cp ${BLUECAT_PREFIX}/lib/libnss_files-*.so /lib
# End of File

```

5. Create the local/fstab file with the following contents:

```
proc /proc proc defaults 0 0
```

NOTE: The first two fields in every record of the inittab file are ignored by the BusyBox init, so they must be empty. For example, the line 1:12345:respawn:/bin/sh is not valid.

6. Create the local/inittab file with the following contents:

```

# System initialization.
::sysinit:/etc/rc.d/rc.sysinit

::respawn:/bin/sh

```

7. Create the local/rc.sysinit file with the following contents:

```

#!/bin/sh

PATH=/bin:/sbin:/usr/bin:/usr/sbin
export PATH
mount -a

```

8. Create a root file system image (`busybox.rfs`) by entering the following command:

```
BlueCat:$ mkrootfs -lv ./busybox.spec ./busybox.rfs
```

NOTE: The Makefile for the developer demo system can be used to produce the BusyBox kernel and RFS images. To do so, modify the Makefile as follows:

1. Change the line `KDI_NAME = developer` to `KDI_NAME = busybox`.
2. Comment out the following lines:

```
# IS_NEED_REBUILD_SRC=$(shell if ! make -q -C src; then echo this ; fi)
# cd src; make all
```

3. Change the following lines:

```
clean      :
            rm -f *.rfs *.tar *.kernel *.disk *.kdi *.srec; cd src; make clean
```

to read as follows:

```
clean      :
            rm -f *.rfs *.tar *.kernel *.disk *.kdi *.srec
```

4. Run the `make all` command.
5. If the following message appears:

```
make: circular busybox <- busybox dependency dropped.
make: *** No rule to make target '.rfs', needed by 'busybox'. stop.
```

edit the Makefile to delete the following line:

```
KDI_NAME = busybox
```

Then retype the line in. There may be trailing characters on this line that cause errors in the Makefile.

Booting the BusyBox Images from a Network

To boot the BlueCat Linux with the BusyBox utility from a network using the BlueCat Linux OS Loader, perform the steps listed below. Refer to Chapter 2, “Downloading and Booting BlueCat Linux on the Target” for additional details about the BlueCat Linux OS Loader.

1. At the OS Loader prompt, type the following commands:

```
> set IF eth0
> set IP target_board_IP
> set HOST development_host_IP
```

```

> set KERNEL tftp busybox.kernel
> set RFS tftp busybox.rfs
> set CMD ramdisk_size=28472
> boot

```

where *target_board_IP* is an IP address of the target, and *development_host_IP* is an IP address of the development host.

As a result, the BusyBox utility has been loaded onto the target board and then automatically started.

Using BusyBox Utilities

This section provides the examples of using the BusyBox utilities. Entering a command from the following list results in the respective output:

- `ls`

```

/ # ls /
bin          etc          lost+found  sbin
dev          lib          proc        usr

```

- `cat`

```

/ # cat /etc/inittab
# System initialization.
::sysinit:/etc/rc.d/rc.sysinit

::respawn:/bin/sh

```

- `chmod`

```

/ # chmod a-x /sbin/reboot
/ # ls -la /sbin/reboot
-rw-r--r--  1 0      0              7836 Mar  7  2003 /sbin/reboot
/ # chmod 755 /sbin/reboot
/ # ls -la /sbin/reboot
-rwxr-xr-x  1 0      0              7836 Mar  7  2003 /sbin/reboot

```

- `echo`

```

/ # echo !!!!!!!!!!!
!!!!!!!!!!!!

```

- `date`

```

/ # date
Thu Jan  1 00:01:57 UTC 1970

```

- `uname`

```

/ # uname -a

```

```
Linux (none) 2.4.10-1 #8 Fri Mar 7 12:33:09 MSK 2003 armv5l unknown
```

- `mount`

```
/ # mount
/dev/root on / type ext2 (rw)
proc on /proc type proc (rw)
```
- `ifconfig`

```
/ # ifconfig eth0 172.17.3.11
/ # ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:80:4D:46:22:B8
          inet addr:172.17.3.11 Bcast:172.17.255.255 Mask:255.255.0.0
          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:100
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
          Interrupt:1

/ # ping 172.17.0.1
PING 172.17.0.1 (172.17.0.1): 56 data bytes
64 bytes from 172.17.0.1: icmp_seq=0 ttl=255 time=1.0 ms
64 bytes from 172.17.0.1: icmp_seq=1 ttl=255 time=0.4 ms
64 bytes from 172.17.0.1: icmp_seq=2 ttl=255 time=0.4 ms
64 bytes from 172.17.0.1: icmp_seq=3 ttl=255 time=0.4 ms

--- 172.17.0.1 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.5/1.0 ms
```

Using TinyLogin RPM Package

The TinyLogin RPM package is a suite of tiny Unix utilities for handling logging into, being authenticated by, changing one's password for, and otherwise maintaining users and groups on an embedded system. It also provides shadow password support to enhance system security.

This section describes the steps necessary for creating and booting a BlueCat Linux system containing TinyLogin and demonstrates use of the TinyLogin utility.

Creating a BlueCat Linux System for TinyLogin

To create a BlueCat Linux image for TinyLogin, perform the following steps:

1. Create a new directory by typing:

```
BlueCat:~$ mkdir -p \
$BLUECAT_PREFIX/demo/tinylogin/local
```

2. Set up the BlueCat Linux kernel configuration by using the standard kernel configuration tools and copy kernel configuration file to the

`$BLUECAT_PREFIX/demo/tinylogin` directory. For instance, type the following commands:

```
BlueCat:$ cd $BLUECAT_PREFIX/usr/src/linux
```

```
BlueCat:$ make xconfig
```

Select **Save and Exit** to update the `.config` file, then type the following commands:

```
BlueCat:$ cp .config \
$BLUECAT_PREFIX/demo/tinylogin/tinylogin.config
```

NOTE: The kernel config file for the developer demo (`$BLUECAT_PREFIX/demo/developer/developer.config`) is also recommended as a starting point.

3. Create the BlueCat kernel downloadable image (`tinylogin.kernel`):

```
BlueCat:$ cd $BLUECAT_PREFIX/demo/tinylogin
```

```
BlueCat:$ mkkernel ./tinylogin.config \
./tinylogin.kernel ./tinylogin.disk
```

4. Create a spec file (`tinylogin.spec`) that contains the following minimal directives:

```
strip on

mkdir /dev
mknod /dev/console c 5 1
ln -s /dev/console /dev/tty
ln -s /dev/console /dev/tty1

mkdir /bin
mkdir /sbin
mkdir -p /etc/rc.d
mkdir /proc
mkdir /tmp
mkdir -p /usr/bin

mkdir /root

mkdir /dev/pts
mknod /dev/ptmx c 5 2

chmod 0666 /dev/ptmx

cp ./local/fstab ./local/passwd ./local/inittab /etc
cp ./local/securetty ./local/shadow /etc
cp ./local/rc.sysinit /etc/rc.d
cp ${BLUECAT_PREFIX}/etc/shells /etc
chmod 644 /etc/shells
cp ${BLUECAT_PREFIX}/etc/group /etc
```

```
lcd ${BLUECAT_PREFIX}/sbin
cp reboot init mingetty /sbin

cp ${BLUECAT_PREFIX}/usr/bin/tinylogin /usr/bin
ln -s /usr/bin/tinylogin /usr/bin/passwd
ln -s /usr/bin/tinylogin /bin/login

lcd ${BLUECAT_PREFIX}/bin
cp mount bash ls cat hostname /bin
ln -s /bin/bash /bin/sh

chmod 711 /etc/rc.d/rc.sysinit

chmod 755 /bin /sbin /usr/bin

chmod 04755 /usr/bin/tinylogin
# End of File
```

NOTE: In this spec file the `/bin/login` and `/usr/bin/passwd` symlinks point to `/usr/bin/tinylogin`. This allows the user to change the user's password simply by typing `passwd`.

5. Create the local `/fstab` file with the following contents:

```
none /proc proc
none /dev/pts devpts
```

6. Create the local `/inittab` file with the following contents:

```
id:1:initdefault:

# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit

1:12345:respawn:/sbin/mingetty tty1
```

7. Create the local `/securetty` file with the following contents:

```
console
tty1
```

8. Create the local `/passwd` file with the following contents:

```
root:x:0:0:/root:/:/bin/bash
guest:x:500:10:/:/bin/bash
```

9. Create the local `/shadow` file:

```
root::10942:0:99999:7:::
guest::500:10:99999:7:::
```

10. Create the local `/rc.sysinit` file with the following contents:

```
#!/bin/sh

PATH=/bin:/sbin:/usr/bin:/usr/sbin
export PATH
```

```
mount -a
hostname myhostname
```

11. Create a root file system image (`tinylogin.rfs`) by entering the following command:

```
BlueCat:$ mkrootfs -lv ./tinylogin.spec \
./tinylogin.rfs
```

NOTE: The Makefile for the developer demo system can be used to produce the TinyLogin kernel and RFS images. To do so, modify the Makefile as follows:

1. Change the line `KDI_NAME = developer` to `KDI_NAME = tinylogin`.
2. Comment out the following lines:

```
# IS_NEED_REBUILD_SRC=$(shell if ! make -q -C src; then echo this ; fi)
# cd src; make all
```

3. Change the following lines:

```
clean      :
            rm -f *.rfs *.tar *.kernel *.disk *.kdi *.srec; cd src; make clean
```

to read as follows:

```
clean      :
            rm -f *.rfs *.tar *.kernel *.disk *.kdi *.srec
```

4. Run the `make all` command.
5. If the following message appears:

```
make: circular tinylogin <- tinylogin dependency dropped.
make: *** No rule to make target `rfs', needed by `tinylogin'. stop.
```

edit the Makefile to delete the following line:

```
KDI_NAME = busybox
```

Then retype the line in. There may be trailing characters on this line that cause errors in the Makefile.

Booting the TinyLogin Images from a Network

To boot the BlueCat Linux with the TinyLogin utility from a network using the BlueCat Linux OS Loader, perform the steps listed below. Refer to Chapter 2, “Downloading and Booting BlueCat Linux on the Target” for additional details about the BlueCat Linux OS Loader.

1. At the OS Loader prompt, type the following commands:

```
> set IF eth0
> set IP target_board_IP
> set HOST development_host_IP
> set KERNEL tftp tinylogin.kernel
> set RFS tftp tinylogin.rfs
> boot
```

where *target_board_IP* is an IP address of the target, and *development_host_IP* is an IP address of the development host.

As a result, the TinyLogin utility has been loaded onto the target board and then automatically started.

Using TinyLogin Utility

This section provides the examples of using the TinyLogin utility:

- Changing the guest password:

```
myhostname login: guest
bash-2.04$ passwd
Changing password for guest
Enter the new password (minimum of 5, maximum of 8 characters)
Please use a combination of upper and lower case letters and numbers.
Enter new password: new_guest_password
Re-enter new password: new_guest_password
passwd[13]: password for `guest' changed by user `guest'
Password changed.
bash-2.04$ exit
myhostname login: guest
Password: new_guest_password
bash-2.04$ exit
```

- Changing the root password:

```
myhostname login: root
login[16]: root login on `console'

bash-2.04# passwd
```

```
Changing password for root
Enter the new password (minimum of 5, maximum of 8 characters)

Please use a combination of upper and lower case letters and numbers.
Enter new password: new_root_password
Re-enter new password: new_root_password
passwd[17]: password for `root' changed by user `root'
Password changed.
bash-2.04# exit
myhostname login: root
Password: new_root_password
login[18]: root login on `console'

bash-2.04# exit
```

- Getting the root permissions:

```
myhostname login: guest
Password: guest_password
bash-2.04$ tinylogin su
Password:
login[17]: root login on `console'

bash-2.04#
```

CHAPTER 5 *Supported Device Drivers*

The following table lists the device drivers supported by the iq80321 and details important information about them.

Table 5-1: Device Drivers Supported by the iq80321 BSP

Hardware Device	Device Drivers	Location in Source Tree	Kernel Configuration Options
UART 1 serial UART (16C550)	<code>serial.c</code>	<code>drivers/char</code>	<code>CONFIG_SERIAL</code> <code>CONFIG_SERIAL_CONSOLE</code>
Ethernet Intel 82544 Giga Ethernet	<code>e1000*.c</code>	<code>drivers/net/e1000</code>	<code>CONFIG_E1000</code>
Ethernet Intel 82559 10/100 Ethernet interface in the PCI slot	<code>eeepro100.c</code>	<code>drivers/net</code>	<code>CONFIG_EEPRO100</code>
I ² C 2 I ² C Serial Interfaces integrated in 80321	<code>i2c-algo-80321.c</code> <code>i2c-adap-80321.c</code>	<code>drivers/i2c</code>	<code>CONFIG_I2C_ALGO_80321</code> <code>CONFIG_I2C_ADAP_80321</code>
Flash 8 MB on board	<code>arm_flash*.c</code>	<code>drivers/mtd/maps</code>	<code>CONFIG_MTD_IQ80321</code>

This chapter describes the BlueCat Linux Application Programming Interface specific for the 80200 processor.

Advanced MMU/Cache Management Services

The BlueCat Linux kernel for IQ80321 provides a set of advanced MMU/cache management services for the Intel XScale 80321 processor. These are declared in the `$BLUECAT_PREFIX/usr/src/linux/include/asm/i80200.h` file.

Advanced MMU/Cache Management Services Overview

The purpose of the advanced MMU/cache management services is to provide an access to Intel i80200 cache control capabilities. The kernel components, mostly device drivers, can use these services to greatly increase performance of a selected piece of critical code. Locking the MMU/cache for a particular purpose effectively allocates the processor execution resources for a particular task at the expense of the other tasks in the system. What follows are the most common examples of how the MMU/cache management can solve various problems:

- A device driver can lock its interrupt handler code into the instruction cache to provide more prompt hardware interrupt response. In some cases, for instance a device driver for a sound device that provides audio data to be played, this might be the only way to be able to meet a hardware device events response requirements.
- If a driver implements mission critical calculations to be completed as fast as possible, locking the code and data into the respective caches combined with the interrupts/scheduling disabled can provide execution

environment that fully utilizes the processor, thus effectively making the code to run as fast as possible on this hardware.

- Advanced MMU/cache management provides means for general tasks targeting the overall optimization of the Linux kernel performance and real-time characteristics.

Advanced MMU/Cache Management Services Reference

What follows is the list of services available for the kernel components:

- `void i80200_dcache_lock(unsigned int addr, int lines)`

Lock a number of lines at the specified virtual address into the data cache.

This service can be used to lock a data area into the data cache. This will ensure the accesses to the locked area are cached. The contents of the area are not pushed out of the cache by accessing other addresses.

- `void i80200_icache_lock(unsigned int addr, int lines)`

Lock a number of lines at the specified virtual address into the instruction cache.

This service can be used to lock a piece of code into the instruction cache. This will ensure execution from the instruction cache. The contents of the area are not pushed out of the cache by execution code from other addresses.

- `void i80200_dcache_unlock(void)`

Unlocks the data cache.

This service invalidates all locks on the data cache, allowing the cache to operate in normal mode with hardware line replacement algorithm. The service can be used to restore the data cache normal operating mode when the advanced cache configuration is not needed anymore.

- `void i80200_icache_unlock(void)`

Unlocks the instruction cache.

This service invalidates all locks on the data cache, allowing the cache to operate in normal mode with hardware line replacement algorithm. The service can be used to restore the instruction cache normal operating mode when the advanced cache configuration is not needed anymore.

- `void i80200_itlb_lock(unsigned int addr)`

Translates and locks an ITLB entry.

The locked ITLB entry will ensure that the instruction fetch from the page at the specified virtual address will go straight through the ITLB translation mechanism, without overhead for the page table tablewalk.

- `void i80200_dtlb_lock(unsigned int addr)`

Translates and locks an DTLB entry.

The locked DTLB entry will ensure that the accesses to the page at the specified virtual address will go straight through the DTLB translation mechanism, without overhead for the page table tablewalk.

- `void i80200_itlb_unlock(void)`

Unlocks the ITLB.

This service invalidates all locks on the instruction TLB, allowing the TLB to operate in normal mode with hardware entry replacement algorithm. The service can be used to restore the ITLB normal operating mode when the advanced TLB configuration is not needed anymore.

- `void i80200_dtlb_unlock(void)`

Unlocks the DTLB.

This service invalidates all locks on the data TLB, allowing the TLB to operate in normal mode with hardware entry replacement algorithm. The service can be used to restore the DTLB normal operating mode when the advanced TLB configuration is not needed anymore.

- `void i80200_md_attr(unsigned int attr)`

Sets attributes for mini data cache.

`attr` defines mini data cache attributes as specified in the Auxiliary Control Register description in the Processor Manual. `attr` can be one of the following:

0 - Write back, Read allocate

1 - Write back, Read/Write allocate

2 - Write through, Read allocate

3 - Unpredictable.

- `void i80200_wb_coalescing(int enable)`

Enables or disables write coalescing.

This service allows to globally disable write coalescing. That is, all writes to an external memory will be performed in the program order without write combining operations in the write buffer, regardless the values of the page attributes.

Application Accelerator Unit Services

The BlueCat Linux kernel for IQ80321 provides a set of Application Accelerator Unit support services for the Intel 80321 I/O chip. These are declared in the `$BLUECAT_PREFIX/usr/src/linux/include/asm/arch/iop80321-aau.h` file.

Application Accelerator Unit Services Overview

The Application Accelerator Unit (AAU) provides hardware acceleration of XOR functions that are commonly used in RAID algorithms. XOR functions can process up to 8 blocks of source data and store the result in the processor memory. The AAU can also be used to transfer data blocks in local memory without using CPU.

The purpose of the Application Accelerator Unit (AAU) services is to provide the necessary functions to utilize the AAU.

Application Accelerator Unit Services Reference

The following is the list of services available for the kernel components:

NOTE: The AAU API can be used by several clients simultaneously.

- `int aau_request(u32 *aau_context, const char *device_id)`
Gets a control over the AAU and initializes internal structures. The AAU context is passed to other AAU API services.
- `int aau_queue_buffer(u32 aau_context, aau_head_t *listhead)`
Creates an AAU buffer chain from user scattered gather list (SGL),

places the created chain into the processing queue and starts the AAU operation. The user creates and initializes a SGL, the header of which is passed to `aau_queue_buffer()` as an argument. The format of SGL is as follows:

```

/* hardware descriptor */
typedef struct _aau_desc
{
    u32 NDA;                /* next descriptor address [READONLY] */
    u32 SAR[AAU_SAR_GROUP]; /* src addr */
    u32 DAR;                /* destination addr */
    u32 BC;                 /* byte count */
    u32 DC;                 /* descriptor control */
    u32 SARE[AAU_SAR_GROUP]; /* extended src addr */
} aau_desc_t;

/* user SGL format */
typedef struct _aau_sgl
{
    aau_desc_t    aau_desc; /* AAU HW Desc */
    u32           status;   /* status of SGL [READONLY] */
    struct _aau_sgl *next; /* pointer to next SG [READONLY] */
    void          *dest;   /* destination addr */
    void          *src[AAU_SAR_GROUP]; /* source addr[4] */
    void          *ext_src[AAU_SAR_GROUP]; /* ext src addr[4] */
    u32           total_src; /* total number of source */
} aau_sgl_t;

/* header for user SGL */
typedef struct _aau_head
{
    u32           total; /* total descriptors allocated */
    u32           status; /* SGL status */
    aau_sgl_t    *list; /* ptr to head of list */
    aau_callback_t callback; /* callback func ptr */
} aau_head_t;

```

If `callback` is not specified the service waits until the SGL has been processed and returns to the caller. If `callback` is specified the service returns to the caller immediately.

- `int aau_suspend(u32 aau_context)`
Suspends the AAU operation.
- `int aau_resume(u32 aau_context)`
Resumes the AAU operation suspended by `aau_suspend()`.
- `int aau_free(u32 aau_context)`
Notifies that the AAU context is not needed any longer.
- `aau_sgl_t * aau_get_buffer(u32 aau_context, int num_buf)`

Allocates a SGL of AAU descriptors. The size of the SGL is passed as an argument to `aau_get_buffer()`.

- `void aau_return_buffer(u32 aau_context, aau_sgl_t *list)`

Frees the AAU SGL.

- `int aau_memcpy(void *dest, void *src, u32 size)`

Copies a memory region using the AAU. This service is called to copy memory regions inside a cache memory area without using CPU.

PCI DMA Transfers Services

The BlueCat Linux kernel for IQ80321 provides a set of PCI DMA transfer services for the Intel 80321 I/O chip. These are declared in the `BLUECAT_PREFIX/usr/src/linux/include/asm/arch/iop80321-dma.h` file.

PCI DMA Transfers Services Overview

The Intel 80321 companion chip contains a DMA controller. The DMA controller is designed to perform high-speed memory transfer between the PCI bus and local memory. Three DMA channels are available: two channels for the primary PCI bus and one channel for the secondary PCI bus. The maximum throughput DMA transfer speed that can be achieved is 528 Mb/s. Data transfer using the DMA controller can be performed without loading CPU.

The purpose of the PCI DMA services is to provide functions to utilize the DMA controller.

PCI DMA Transfers Services Reference

The following is the list of services available for the kernel components:

- `int dma_request(dmach_t channel, const char *device_id)`

Requests a control over a DMA channel that can be one of the following:

`IOP310_DMA_S0`: PCI Secondary 1

IOP310_DMA_SI: PCI Secondary 2

NOTE: The same DMA channel can be used by several clients simultaneously.

If the channel has been allocated, the service returns the channel number that must be passed to other DMA services. Otherwise, the service returns a negative value.

- `int dma_queue_buffer(dmach_t channel, dma_sghead_t *listhead)`

Creates a DMA buffer chain from the user SGL, places this chain into the processing queue and starts the DMA operation. The user creates and initializes a SGL, the header of which is passed to `dma_queue_buffer()` as an argument. The format of SGL is as follows:

```

/*
 * Scattered Gather DMA List for user
 */
typedef struct _dma_desc
{
    u32  NDAR;      /* next descriptor adress [READONLY] */
    u32  PDAR;      /* PCI address */
    u32  PUADR;     /* upper PCI address */
    u32  LADR;      /* local address */
    u32  BC;        /* byte count */
    u32  DC;        /* descriptor control */
} dma_desc_t;

typedef struct _dma_sgl
{
    dma_desc_t      dma_desc; /* DMA descriptor */
    u32             status;   /* descriptor status [READONLY] */
    void            *data;    /* local data virt addr */
    struct _dma_sgl *next;    /* next descriptor [READONLY] */
} dma_sgl_t;

/* dma sgl head */
typedef struct _dma_head
{
    u32             total;    /* total elements in SGL */
    u32             status;   /* status of sgl */
    u32             mode;     /* read or write mode */
    dma_sgl_t       *list;    /* pointer to list */
    dma_callback_t  callback; /* callback function */
} dma_head_t;

```

If `callback` is not specified, the service waits until the SGL has been processed and returns to the caller. Otherwise, the service returns to the caller immediately calling `callback` when the SGL has been processed.

- `dma_sgl_t * dma_get_buffer(dmach_t channel, int buf_num)`

Allocates a SGL of DMA descriptors. The size of the SGL is passed to `dma_get_buffer` as a parameter.

- `void dma_return_buffer(dmach_t channel,
 dma_sgl_t *list)`

Frees the DMA SGL.

- `int dma_suspend(dmach_t channel)`

Suspends all DMA transfers via the channel `channel`.

- `int dma_resume(dmach_t channel)`

Resumes DMA transfers suspended by `dma_suspend()`.

- `int dma_flush_all(dmach_t channel)`

Stops all operation with queued buffer chains via the DMA channel and marks partially processed buffer chains as uncompleted. This service is used when DMA channel errors have occurred.

- `void dma_free(dmach_t channel)`

Notifies the DMA API that the channel is not used any longer.

Messaging Unit Support Services

The BlueCat Linux kernel for IQ80321 provides a set of Messaging Unit support services for the Intel 80321 I/O chip. These are declared in the `$BLUECAT_PREFIX/usr/src/linux/include/asm/arch/iop80321-mu.h` file.

Messaging Unit Support Services Overview

The Messaging Unit (MU) provides a mechanism to perform data transfer between the primary PCI bus and the 80200 CPU. Both ends are notified on the data arrival.

The following messaging mechanisms are implemented:

- Message Registers
- Doorbell Registers
- Circular Queues
- Index Registers

All mechanisms can be used simultaneously.

Messaging Unit Support Reference

The following is the list of services available for the kernel components.

Message Registers Services

The Message Registers mechanism is supported by Message Registers services.

The Message Registers is composed of four 32-bit registers: 2 inbound registers and 2 outbound registers.

- `int mu_msg_request(u32 *mu_context)`
Acquires the ownership of the Message Register services.
- `int mu_msg_set_callback(u32 mu_context, u8 reg, mu_msg_cb_t func)`
Sets up the callback function for the first, second or both inbound message registers.
- `int mu_msg_post(u32 mu_context, u32 val, u8 reg)`
Posts a message in the specified outbound message register.
- `int mu_msg_free(u32 mu_context, u8 mode)`
Releases the Message Register service.

Doorbell Registers Services

The Doorbell Registers mechanism is supported by Doorbell Registers services.

The Doorbell Registers is composed of one inbound and one outbound register.

- `int mu_db_request(u32 *mu_context)`
Acquires the ownership of the Doorbell Registers services.
- `int mu_db_set_callback(u32 mu_context, mu_db_cb_t func)`
Sets up the callback function for the inbound doorbell message register.
- `void mu_db_ring(u32 mu_context, u32 mask)`
Sets the bits in the outbound doorbell message register using `mask`.

- `int mu_db_free(u32 mu_context)`

Releases the Doorbell Registers service.

Circular Queues Services

The Circular Queues mechanism is supported by Circular Queues services. The Circular Queues is composed of 4 circular queues: inbound post queue, inbound free queue, outbound post queue, outbound free queue. These queues are used to pass messages.

- `int mu_cq_request(u32 *mu_context, u32 q_size)`

Acquires the ownership of the Circular Queues services.

- `int mu_cq_inbound_init(u32 mu_context,
 mfa_list_t *list, u32 size,
 mu_cq_cb_t func)`

Initializes inbound queues. A list of free message frames to be put in inbound free queue and the callback function to handle the inbound messages are the parameter of this service.

- `int mu_cq_enable(u32 mu_context)`

Enables the Circular Queues mechanism operation.

- `u32 mu_cq_get_frame(u32 mu_context)`

Obtains the address of an outbound free message frame.

- `int mu_cq_post_frame(u32 mu_context, u32 mfa)`

Tries to put the message frame into the outbound message queue.

- `int mu_cq_free(u32 mu_context)`

Releases the Circular Queues services.

Index Registers Service

The Index Register mechanism is supported by Index Register services. The following Index Registers Service are available:

- `int mu_ir_request(u32 *mu_context)`

Acquires the ownership of the Index Register services.

Start the PMU operation. mode specifies what statistics to capture. mode can be one of the following:

I80312_PMU_MODE0	Performance Monitoring Disabled
I80312_PMU_MODE1	Primary PCI bus and internal agents (Bridge, DMA Ch0, DMA Ch1, PATU)
I80312_PMU_MODE2	Secondary PCI bus and internal agents (Bridge, DMA Ch0, DMA Ch1, PATU)
I80312_PMU_MODE3	Secondary PCI bus and internal agents (External Masters 0..2 and Intel 80321 I/O Companion Chip)
I80312_PMU_MODE4	Secondary PCI bus and internal agents (External Masters 3..5 and Intel 80321 I/O Companion Chip)
I80312_PMU_MODE5	Intel 80321 I/O companion chip internal bus, DMA Channels and Application Accelerator
I80312_PMU_MODE6	Intel 80321 I/O companion chip internal bus, ATU, and Intel 80200 processor
I80312_PMU_MODE7	Intel 80321 I/O companion chip internal bus, Primary PCI bus, Secondary PCI bus and Secondary PCI agents (External Masters 0..5 & Intel 80321 I/O companion chip)

- `int pmon_stop(pmon_res_t *result)`

Stops the PMU operation and returns the results. The format of the result is as follows:

```
typedef struct _pmon_result
{
    u32    GTMR; /* Global Timer Mode Register */
    u32    ESR; /* Event Select Register */
    u32    EMISR; /* Event Monitoring Interrupt Status Register */
    u32    GTSR; /* Global Time Stamp Register */
    u32    TIMEROVERFLOW; /* Time Stamp overflow count */
    u32    PECCR[NUM_OF_PECCR]; /* Programmable Event Counter Register 1 -14 */
    u32    PECCR_CNT[NUM_OF_PECCR]; /* Overflow counter for PECCR1-14 */
} pmon_res_t;
```

IQ80321 Target Board Problems and Limitations

The following are known problems and limitations of this release:

- Kernel debugger (KDBG) is supported but has not been explicitly tested. This is due to a limitation of the IQ80321 hardware, which provides only a single serial port. Since debugging of BlueCat over a serial line requires a dedicated serial port on the target, testing of the kernel debugger on the IQ80321 is not possible.
- For the same reason, debugging of user applications over a serial line is supported but not explicitly tested.
- If a BlueCat embedded system is loaded to the target using RedBoot, stability problems are possible. Such stability problems are identified as either inability of BlueCat to boot or BlueCat crashing at seemingly random points of execution. This problem is believed to be due to RedBoot not shutting down the Ethernet controller properly prior to passing control to the OS image being booted. As a result, the Ethernet DMA continues to function and may corrupt RAM contents with Ethernet packets (such as broadcast traffic) coming from the network. As a workaround, use LynuxWorks Boot Loader for booting BlueCat from the network. Boot Loader takes special measures to shut down DMA-capable devices such as Ethernet prior to passing control to a loaded OS image.
- The IQ80321 board might fail to boot up if there is a 33MHz PCI card installed into the on-board PCI slot. To solve this problem, set the S7E1-6 and S7E1-7 switches to the ON position.
- If the S7E1-6 and S7E1-7 switches are in the ON position and there is no PCI card plugged into the on-board PCI slot, LynuxWorks Boot Loader

might unexpectedly hang or crash. It is highly recommended to use the default switches settings whenever it is possible.

- Use the following command in order to use Ethernet in the `i_osloader` demo system:

```
make -f Makefile.i xconfig
```

Enable your network card, then type the following command:

```
make -f Makefile.i all
```

The demo now will have a correct Ethernet configuration and can be used to boot other BlueCat Linux demos over the network.

- If `mkrootfs` is terminated (either by an error or by a signal), it tries to clean all its temporary files before exiting. However, due to certain features of the Cygwin execution environment, such temporary files can remain uncleaned in the `/tmp` directory on a Windows host. It is recommended that the `/tmp` directory be regularly checked and cleaned.
- Debugging of multithreaded applications via GDB is not supported.
- The `tc1x` RPM package is not included in the Windows-hosted distribution.
- On Windows hosts, some file permissions (including `r` and `s`) always have default values. To set permissions different from the default values, the `chmod` command should be used in the `.spec` file.