

# BlueCat Linux Board Support Guide

---

DOC-0482-01

*For Intel IQ80310 Boards*

Release 4.0

Product names mentioned in *BlueCat Linux Target Support Guide for Intel IQ80310 Boards* are trademarks of their respective manufacturers and are used here for identification purposes only.

Copyright ©1987-2001, 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 Target Support Guide for Intel IQ80310 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

<b>CHAPTER 1</b>	<b>OVERVIEW .....</b>	<b>1</b>
<b>CHAPTER 2</b>	<b>DOWNLOADING AND BOOTING BLUECAT LINUX ON THE TARGET .....</b>	<b>3</b>
	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
	Downloading a BlueCat Linux System into Flash .....	6
	Booting a Demo System from Flash .....	7
	Booting a Demo System over a Network .....	8
<b>CHAPTER 3</b>	<b>KERNEL CONFIGURATION OPTIONS .....</b>	<b>9</b>
<b>CHAPTER 4</b>	<b>SUPPORTED DEMO SYSTEMS.....</b>	<b>27</b>
	Demo Systems .....	27
	developer Demo System .....	27
	osloader Demo System .....	28
	showcase Demo System .....	28
	Using Selected RPM Packages .....	28
	Using BusyBox .....	28
	Creating a BlueCat Linux System for BusyBox .....	28
	Booting BusyBox Images from a Network .....	30
	Using BusyBox Utilities .....	32
	Using TinyLogin RPM Package .....	33
	Creating a BlueCat Linux System for TinyLogin .....	33

Booting the TinyLogin Images from a Network .....	36
Using TinyLogin .....	36
Using GNU Zebra RPM Package .....	37
Creating a BlueCat Linux System for Zebra .....	38
Booting the Zebra Images from a Network .....	41
Using Zebra .....	41

---

<b>CHAPTER 5</b>	<b>SUPPORTED DEVICE DRIVERS .....</b>	<b>45</b>
	Using Intel PRO/1000 Ethernet Driver .....	45
	Flash Support .....	47
	Configuring Custom Flash Chips .....	47
	PCI Support .....	48
	Using Drivers for External PCI Devices with BlueCat Linux for IQ80310 Boards .....	48
	Configuring PCI Buses .....	48
	Possible Incompatibilities between BlueCat Linux for IQ80310 and Existing PCI Device Drivers .....	50

---

<b>CHAPTER 6</b>	<b>BLUECAT LINUX-SPECIFIC APIS .....</b>	<b>53</b>
	Advanced MMU/Cache Management Services .....	53
	Overview .....	53
	Advanced MMU/Cache Management Service Reference .....	54
	LED Display Control .....	56
	Application Accelerator Unit Services .....	56
	Overview .....	57
	Application Accelerator Unit Services Reference .....	57
	PCI DMA Transfers Services .....	58
	PCI DMA Transfers Services Overview .....	59
	PCI DMA Transfers Services Reference .....	59
	Messaging Unit Support Services .....	61
	Messaging Unit Support Services Overview .....	61
	Messaging Unit Support Reference .....	61
	Message Registers Services .....	61
	Doorbell Registers Services .....	62
	Circular Queues Services .....	62
	Index Registers Service .....	63
	Performance Monitoring Services .....	63
	Performance Monitoring Services Overview .....	63

---

Performance Monitoring Services Reference .....	64
Hints for Xscale-Specific Optimizations .....	65
<b>CHAPTER 7</b>	
<b>DEFECT FIXES AND KNOWN PROBLEMS.....</b>	<b>67</b>
IQ80310 Target Board Problems and Limitations .....	68



The *BlueCat Linux Board Support Guide for Intel IQ80310 Evaluation Boards* (BSG) provides information about the BlueCat Linux Board Support Package (BSP) for Intel 80310 I/O processor chipset with Intel Xscale microarchitecture (ARM architecture compliant). This BSP includes support for the Intel 80200 processor based on the Intel Xscale microarchitecture and the Intel 80312 I/O companion chip.

Throughout this Board Support Guide (BSG), the BSP is referred to as the “IQ80310,” “iq80310,” or simply as the “target board.”

- **Chapter 1** is an overview of the BSG’s individual chapters.
- **Chapter 2** describes the download and boot procedure for BlueCat Linux on the target IQ80310 board, using the `showcase` demo system as an example.
- **Chapter 3** details configuration of the prebuilt BlueCat Linux kernel contained in the iq80310 BSP.
- **Chapter 4** in this guide lists the BlueCat Linux demo systems supported by the iq80310 BSP
- **Chapter 5** lists the device drivers in the iq80310 BSP.
- **Chapter 6** describes BlueCat Linux-specific Application Programming Interfaces (APIs) for the IQ80310 board.



# *Downloading and Booting BlueCat Linux on the Target*

This chapter provides instructions for downloading a BlueCat Linux 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 to downloading and booting a BlueCat Linux system on the user's target board. Scenarios that use demo systems included in the BlueCat Linux distribution are presented. As such, the user is assumed to be familiar with the target board hardware and the manufacturer's documentation for it. The user must also have an understanding of system administration for the particular cross development host before BlueCat Linux and the iq80310 Board Support Package (BSP) are installed.

Before installing and booting BlueCat Linux on an IQ80310 target board, it is assumed that the BlueCat Linux Xscale core and the iq80310 BSP have been installed on the cross development host. The user must:

1. Install the BlueCat Linux Xscale Core on the cross development host, as described in the "Installing the Default Configuration" section in Chapter 1, "Introduction and Installation" of the *BlueCat Linux User's Guide*.
2. Install the iq80310 BSP on the cross development host as described in the "Installing Target Board Support" section of Chapter 1 in the *BlueCat Linux User's Guide*.
3. Activate support for the IQ80310 target board as detailed in the "Activating Support for a Target Board" section of Chapter 1 in the *BlueCat Linux User's Guide*.

## Downloading and Booting Overview

The procedure for downloading and booting a BlueCat Linux system on the IQ80310 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 the BlueCat OS loader demo system. The OS loader demo includes the `i_osloader` and `osloader` downloadable images.

`osloader` is the image that is configured with the base functionality of the BlueCat Linux OS loader. This includes the ability to download BlueCat Linux 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 two serial ports. The first port (the `J9` connector) is used by the RedBoot firmware. The second port (the `J10` connector) is used by the BlueCat Linux embedded system.

Before using the board, these ports need to be connected to the development host. It is recommended to connect the `J9` connector to `COM1` on the host, and the `J10` connector to `COM2` on the host. Use the serial cables shipped with the IQ80310 board to connect the ports.

The serial port settings on the host must be as follows:

- The serial port connected to the target `J9` port has a baud rate of 115200
- The serial port connected to the `J10` target port has a baud rate of 9600

Throughout this chapter, the terminal window connected to the J9 connector is referred to as the *RedBoot console*, and the terminal window connected to the J10 connector is referred to as the *BlueCat Linux console*.

## 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 IQ80310 board is the J18 connector. The user must use this port to connect the IQ80310 or a 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 false
Use BOOTP for network configuration: false false
Local IP address: 127.0.0.1 target_board_IP
Default server IP address: 127.0.0.1 development_host_IP
GDB connection port: 0 0
Network debug at boot time: false false
Update RedBoot non-volatile configuration - are you sure (y/n)? y
...Unlock from 0x007c0000-0x007e0000: .
...Erase from 0x007c0000-0x007e0000: .
...Program from 0xa0013018-0xa0013418 at 0x007c0000: .
...Lock from 0x007c0000-0x007e0000:
```

where *target\_board\_IP* is the IP address of the target, and *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 Linux embedded system can be downloaded into target flash memory using the BlueCat Linux OS loader. Refer to the *BlueCat Linux User's Guide* for details on BlueCat Linux OS loader.

These instructions are applicable to any of the demo systems. Specifically, this chapter uses the `showcase` demo system as an example.

The following figure shows how the flash memory on IQ80310 boards is partitioned for BlueCat Linux:

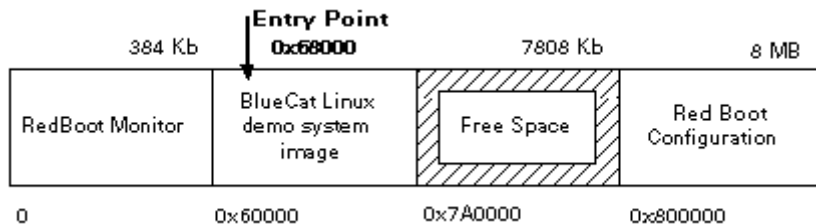


Figure 2-1: Partitioning Flash Memory

Use the following procedure to download `showcase` onto the target board:

1. Copy the `i_osloader.srec` file from the `BLUECAT_PREFIX/demo/osloader` directory to the `/tftpboot` directory on the host.
2. Copy the `showcase.kdi` file from the `BLUECAT_PREFIX/demo/showcase` 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 i_osloader.srec
RedBoot> go
```

These commands load the `i_osloader` system to RAM and start it. As a result, the BlueCat OS loader prompt appears on the BlueCat console.

5. At the BlueCat Linux OS loader prompt type the following commands:

```
> set IF eth0
> set IP target_board_IP
> set HOST development_host_IP
> set FILE tftp showcase.kdi
> exec flash_fdisk /dev/mtdchar0 3-61
> flash /dev/mtdchar1 erase
> reset
```

where *target\_board\_IP* is the target IP address and *development\_host\_IP* is the IP address of the host.

After these commands have been performed, the `showcase` demo system is programmed into flash and can be booted as described below.

---

## Booting a Demo System from Flash

The following procedure is used to boot the showcase demo system installed into the target board flash memory. For details on how to install a demo system into flash, refer to the “Downloading a BlueCat Linux System into Flash” section.

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

```
RedBoot> go 0x00068000
```

This command starts the `showcase` demo programmed into flash.

The IQ80310 board can be configured to start a demo system programmed into flash automatically upon board power-up. Use the following command to prepare the board to boot BlueCat Linux from flash automatically:

```
RedBoot> fconfig
Run script at boot: false true
Boot script:
Enter script, terminate with empty line
>> go 0x68000
>> ENTER
Boot script timeout (1000ms resolution): 0 2
Use BOOTP for network configuration: false ENTER
Local IP address: 172.17.3.11 ENTER
Default server IP address: 172.17.0.1 ENTER
GDB connection port: 0 ENTER
Network debug at boot time: false ENTER
Update RedBoot non-volatile configuration - are you sure (y/n)? Y
... Unlock from 0x007c0000-0x007e0000: .
... Erase from 0x007c0000-0x007e0000: .
```

```
... Program from 0xa0013018-0xa0013418 at 0x007c0000: .  
... Lock from 0x007c0000-0x007e0000: .
```

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

---

**NOTE:** In the **Boot script timeout** field, some boards may require a longer timeout before executing the script.

---

---

## Booting a Demo System over a Network

The following demonstrates booting the `showcase` demo system over a network.

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

```
RedBoot> load osloader.srec  
RedBoot> go
```

These commands start the `osloader` demo system from RAM and the BlueCat Linux OS loader prompt appears on the BlueCat Linux console.

5. At the BlueCat Linux 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 the target IP address and `development_host_IP` is the IP address of the host.

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

---

# *Kernel Configuration Options*

The iq80310 BSP comes with a default BlueCat Linux kernel. This kernel has a number of configuration options. These options are detailed in the tables listed in Table 3-1: Kernel Configuration Options below

**Table 3-1: Kernel Configuration Options**

Table Number and Configuration Parameter
Table 3-2: Code Maturity Level Options
Table 3-3: Loadable Module Support
Table 3-4: System and Processor Type
Table 3-5: IQ80310-Specific Options
Table 3-6: General Setup
Table 3-7: Parallel Port Support
Table 3-8: Memory Technology Devices
Table 3-9: Plug and Play Configuration
Table 3-10: Block Devices
Table 3-11: Multi-Device Support (RAID and LVM)
Table 3-12: Networking Options
Table 3-13: QoS and/or Fair Queueing
Table 3-14: Network Device Support
Table 3-15: ARCnet Devices
Table 3-16: Ethernet (10 or 100 MBit)
Table 3-17: Ethernet (1000 Mbit)
Table 3-18: Wireless LAN (non-Ham Radio)

**Table 3-1: Kernel Configuration Options (Continued)**

<b>Table Number and Configuration Parameter</b>
Table 3-19: Token Ring Devices
Table 3-20: Wan Interfaces
Table 3-21: Amateur Radio Support
Table 3-22: IrDA (infrared) Support
Table 3-23: ATA/IDE/MFM/RLL Support
Table 3-24: SCSI Support
Table 3-25: IEEE 1394 (FireWire) Support
Table 3-26: I2O Support
Table 3-27: ISDN Subsystem
Table 3-28: Input Core Support
Table 3-29: Character Devices
Table 3-30: I2C Support
Table 3-31: Mice
Table 3-32: Watchdog Cards
Table 3-33: Floppy Tape Device Driver, Ftape
Table 3-34: Multimedia Devices
Table 3-35: File Systems
Table 3-36: Network File Systems
Table 3-37: Partition Types
Table 3-38: Sound Support
Table 3-39: USB Support
Table 3-40: Kernel Hacking
Table 3-41: Modular Advanced Power Management
Table 3-42: LinuxWorks 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 incomplete code/drivers

**Table 3-3: Loadable Module Support**

Option	Value	Description
CONFIG_MODULES	Y	Enables loadable module support
CONFIG_MODVERSIONS	Y	Sets version information on all symbols for modules
CONFIG_KMOD	Y	Kernel module loader

**Table 3-4: System and Processor Type**

Option	Value	Description
CONFIG_ARCH_IQ80310	Y	ARM system type

**Table 3-5: IQ80310-Specific Options**

Option	Value	Description
CONFIG_IQ80310_RAMSIZE	32	System memory size in Mb
CONFIG_IQ80310_PCI_MASK_ROUTING	N	Mask PCI interrupts by routing them to host
CONFIG_IQ80310_AAU	Y	Support for Application Accelerator Unit
CONFIG_IQ80310_DMA	Y	Support for DMA transfers between PCI and local memory
CONFIG_IQ80310_MU	Y	Support for Messaging Unit (MU)
CONFIG_IQ80310_PMON	Y	Support for Performance Monitor Unit

Table 3-6: General Setup

Option	Value	Description
CONFIG_ANGELBOOT	N	Load kernel using Angel Debug Monitor
CONFIG_BLUECAT_IGNORE_PRINTK	N	BlueCat Linux ignore printk
CONFIG_BLUECAT_THUMB	N	BlueCat Linux kernel support for THUMB binaries
CONFIG_BLUECAT_LOADER	N	BlueCat Linux OS loader
CONFIG_BLUECAT_SMALL_FOOTPRINT	N	BlueCat Linux small memory footprint
CONFIG_PCI	Y	PCI Support
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 benchmark
CONFIG_SYSVIPC	N	System V IPC
CONFIG_BSD_PROCESS_ACCT	N	BSD Process Accounting
CONFIG_SYSCTL	N	Systl 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**

Option	Value	Description
CONFIG_MTD	Y	Memory Technology Device (MTD) support
CONFIG_MTD_DEBUG	N	Debugging
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
CONFIG_MTD_PMC551	N	Ramix PMC551 PCI Mezzanine RAM card support
CONFIG_MTD_SLRAM	N	Uncached system RAM
CONFIG_MTD_RAM	N	Support for RAM chips in bus mapping
CONFIG_MTD_ROM	N	Support for ROM chips in bus mapping
CONFIG_MTD_MTDRAM	N	Test driver using RAM
CONFIG_MTD_CFI	Y	Common Flash Interface (CFI) support
CONFIG_MTD_CFI_ADV_OPTIONS	N	CFI advanced configuration options
CONFIG_MTD_CFI_PROBE_CFI	N	Automatic chip detection
CONFIG_MTD_284800B_CFI	N	Support for Intel E28F800B flash device
CONFIG_MTD_28F016S_CFI	N	Support for Intel E28F016S flash device
CONFIG_MTD_28F320_CFI	N	Support for Intel 28F320 flash device
CONFIG_MTD_SM732X8_CFI	N	Support for SMT SM732x8 flash device
CONFIG_MTD_CFI_INTELEXT_CFI	N	CFI support for Intel/Sharp Extended Commands
CONFIG_MTD_CFI_AMDSTD_CFI	N	CFI support for AMD/Fujitsu Standard Commands

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

Option	Value	Description
CONFIG_MTD_CFI_INTELSTD_CFI	N	CFI support for Intel/Sharp Standard commands
CONFIG_MTD_AMDSTD	N	AMD compatible flash chip support (non-CFI)
CONFIG_MTD_SHARP	N	Pre-CFI Sharp chip support
CONFIG_MTD_PHYSMAP_CFI	N	CFI flash device in physical memory map
CONFIG_MTD_NORA_CFI	N	CFI flash device mapped on Nora
CONFIG_MTD_PNC2000_CFI	N	CFI flash device mapped on Photron PNC-2000
CONFIG_MTD_RPXLITE_CFI	N	CFI flash device mapped on RPX Lite or CLLF
CONFIG_MTD_SC520CDP_CFI	N	CFI flash device mapped on AMD SC520 CDP
CONFIG_MTD_SBC_MEDIAGX_CFI_INTELNEXT	N	CFI flash device mapped on Arcom SBC-MediaGX
CONFIG_MTD_ELAN_104NC_CFI_INTELNEXT	N	CFI flash device mapped on Arcom ELAN-104NC
CONFIG_MTD_SA1100_CFI	N	CFI flash device mapped on StrongARM SA11x0
CONFIG_MTD_DC21285_CFI	N	CFI flash device mapped on DC21285 Footbridge
CONFIG_MTD_IQ80310	Y	Flash chip mapping on IQ80310 board
CONFIG_MTD_IQ80310_PART	:	Partitions layout
CONFIG_MTD_CSTM_CFI_JEDEC_CFI	N	CFI and JEDEC flash device mapping on custom board
CONFIG_MTD_JEDEC	N	JEDEC device support
CONFIG_MTD_NAND	N	NAND device support
CONFIG_MTD_CHAR	Y	Direct character 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: Plug and Play Configuration**

Option	Value	Description
CONFIG_PNP	N	Plug and Play support

**Table 3-10: 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 ( <code>initrd</code> ) support
CONFIG_BLUECAT_RFS	Y	BlueCat Linux RFS support

**Table 3-11: Multi-Device Support (RAID and LVM)**

Option	Value	Description
CONFIG_MD	N	Multiple device support (RAID and LVM)

Table 3-12: 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 auto-configuration
CONFIG_NET_IPIP	N	IP: Tunneling
CONFIG_NET_IPGRE	N	IP: GRE tunnels over IP
CONFIG_INET_ECN	N	TCP Explicit Congestion Notification support
CONFIG_SYN_COOKIES	N	IP: TCP <code>syncookie</code> support (Not enabled per default)
CONFIG_IPV6	N	The IPv6 protocol (Experimental)
CONFIG_KHTTPD	N	Kernel <code>httpd</code> acceleration (Experimental)
CONFIG_ATM	N	Asynchronous Transfer Mode (Experimental)
CONFIG_IPX	N	The IPX protocol
CONFIG_ATALK	N	Appletalk protocol support
CONFIG_DECNET	N	DECnet support
CONFIG_BRIDGE	N	802.1d Ethernet bridging support
CONFIG_X25	N	CCITT X.25 Packet Layer (Experimental)
CONFIG_LAPB	N	LAPB Data Link Layer (Experimental)
CONFIG_LLC	N	802.2 LLC (Experimental)
CONFIG_NET_DIVERT	N	Frame diverter (Experimental)
CONFIG_ECONET	N	Acorn Econet/AUN protocols (Experimental)

**Table 3-12: Networking Options (Continued)**

Option	Value	Description
CONFIG_WAN_ROUTER	N	WAN router
CONFIG_NET_FASTROUTE	N	Fast switching
CONFIG_NET_HW_FLOWCONTROL	N	Forwarding between two high speed interfaces

**Table 3-13: QoS and/or Fair Queueing**

Control	Value	Description
CONFIG_NET_SCHED	N	QoS and/or Fair Queueing (Experimental)

**Table 3-14: Network Device Support**

Option	Value	Description
CONFIG_NETDEVICES	Y	Network device 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 driver support
CONFIG_NET_SB1000	N	General Instruments Surfboard 1000
CONFIG_FDDI	N	FDDI driver support
CONFIG_HIPPI	N	HIPPI driver support (Experimental)
CONFIG_PPP	N	PPP (point-to-point) support
CONGIG_SLIP	N	SLIP (serial line) support
CONFIG_NET_FC	N	Fibre Channel driver support
CONFIG_RCPCI	N	Red Creed Hardware VPN (Experimental)
CONFIG_SHAPER	N	Traffic Shaper (Experimental)

**Table 3-15: ARCnet Devices**

Option	Value	Description
CONFIG_ARCNET	N	ARCnet support

**Table 3-16: Ethernet (10 or 100 MBit)**

Option	Value	Description
CONFIG_NET_ETHERNET	N	Ethernet (10 or 100 MBit)
CONFIG_NET_VENDOR_3COM	N	3COM cards
CONFIG_NET_VENDOR_SMC	N	Western Digital/SMC cards
CONFIG_NET_VENDOR_RACAL	N	Racal-Interlan (Micom) NI cards
CONFIG_ATI700	N	AT1700/1720 support (Experimental)
CONFIG_DEPCA	N	DEPCA, DE10x, DE200, DE201, DE202, DE422 support
CONFIG_HP100	N	HP 10/100VG PCLAN (ISA, EISA, PCI) support
CONFIG_NET_PCI	Y	EISA, VLB, PCI and on-board controllers
CONFIG_PCNET32	N	AMD PCnet32 support
CONFIG_ADAPTEC_STARFIRE	N	Adaptec Starfire support (Experimental)
CONFIG_APRICOT	N	Apricot Xen-II on board Ethernet
CONFIG_TULIP	N	DECchip Tulip (dc21x4x) PCI support
CONFIG_DE4X5	N	Generic DECchip & DIGITAL EtherWORKS PCI/EISA
CONFIG_DGRS	N	Digi Intl. RightSwitch SE-Xsupport
CONFIG_DM9102	N	DM9102 PCI Fast Ethernet Adapter support (Experimental)
CONFIG_EEPRO100	Y	EtherExpress PRO/100 support
CONFIG_EEPRO100_PM	N	Enable Power Management (Experimental)
CONFIG_NATSEMI	N	National Semiconductor DP83810 series PCI Ethernet support

**Table 3-16: Ethernet (10 or 100 MBit) (Continued)**

Option	Value	Description
CONFIG_NE2K_PCI	N	PCI NE2000 and clones support
CONFIG_8139TO0	N	RealTek RTL-8139 Fast Ethernet adapter support
CONFIG_RTL8129	N	RealTek 8129 (not 8019/8029/8139) support (Experimental)
CONFIG_SIS900	N	SiS 900/7016 PCI Fast Ethernet Adapter support
CONFIG_EPIC100	N	SMC EtherPower II
CONFIG_SUNDANCE	N	Sundance Alta support
CONFIG_TLAN	N	TI ThunderLAN support
CONFIG_VIA_RHINE	N	VIA Rhine support
CONFIG_WINBOND_840	N	Winbond W89c840 Ethernet support
CONFIG_NET_POCKET	N	Pocket and portable adaptors
CONFIG_HAPPYMEAL	N	Sun Happy Meal 10/100baseT PCI support

**Table 3-17: Ethernet (1000 Mbit)**

Option	Value	Description
CONFIG_ACENIC	N	Alteon AceNIC/3Com 3C985/ NetGear GA620 Gigabit 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

**Table 3-18: Wireless LAN (non-Ham Radio)**

Option	Value	Description
CONFIG_NET_RADIO	N	Wireless LAN (non-ham radio)

**Table 3-19: Token Ring Devices**

Option	Value	Description
CONFIG_TR	N	Token Ring driver support

**Table 3-20: Wan Interfaces**

Option	Value	Description
CONFIG_WAN	N	Wan interfaces support

**Table 3-21: Amateur Radio Support**

Option	Value	Description
CONFIG_HAMRADIO	N	Amateur radio support

**Table 3-22: IrDA (infrared) Support**

Option	Value	Description
CONFIG_IRDA	N	IrDA subsystem support

**Table 3-23: ATA/IDE/MFM/RLL Support**

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

---

**Table 3-24: SCSI Support**

Option	Value	Description
CONFIG_SCSI	N	SCSI support

**Table 3-25: IEEE 1394 (FireWire) Support**

Option	Value	Description
CONFIG_IEEE1394	N	IEEE 1394 (FireWire) support (Experimental)

**Table 3-26: I2O Support**

Option	Value	Description
CONFIG_I2O	N	I2O support

**Table 3-27: ISDN Subsystem**

Option	Value	Description
CONFIG_ISDN	N	ISDN support

**Table 3-28: Input Core Support**

Option	Value	Description
CONFIG_INPUT	N	Input core support

**Table 3-29: 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	Non-standard serial port support
CONFIG_UNIX98_PTYS	Y	Unix98 PTY support
CONFIG_UNIX98_PTY_COUNT	32	Maximum number of Unix98 PTYs in use (0-2048)
CONFIG_LED_IQ80310	N	IQ80310 LED control

**Table 3-30: I2C Support**

Option	Value	Description
CONFIG_I2C	N	I2C support

**Table 3-31: Mice**

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

**Table 3-32: Watchdog Cards**

Option	Value	Description
CONFIG_WATCHDOG	N	Watchdog Timer support

---

**Table 3-33: Floppy Tape Device Driver, Ftape**

Option	Value	Description
CONFIG_FTAPE	N	Ftape (QIC-80/Travan) support

**Table 3-34: Multimedia Devices**

Option	Value	Description
CONFIG_VIDEO_DEV	N	Video for Linux

**Table 3-35: File Systems**

Option	Value	Description
CONFIG_QUOTA	N	Quota support
CONFIG_AUTOFS_FS	N	Kernel automounter support
CONFIG_AUTOFS4_FS	N	Kernel automounter v4 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_EFS_FS	N	EFS file system support (read-only) (Experimental)
CONFIG_JFFS_FS	Y	Journalling Flash File System (JFFS) support (Experimental)
CONFIG_JFFS_FS_VERBOSE	N	JFFS debugging verbosity (0 = quiet, 3 = noisy)

**Table 3-35: File Systems (Continued)**

Option	Value	Description
CONFIG_JFFS2_FS	N	Journalling Flash File System v2 (JFFS2) support (Experimental)
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_CRAMFS	N	Compressed ROM file system support
CONFIG_RAMFS	N	Simple RAM-based file system support
CONFIG_FAT_FS	N	DOS FAT file system support
CONFIG_ISO9660_FS	N	ISO 9660 CD-ROM file system support
CONFIG_MINIX_FS	N	Minix file system support
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_ROMFS_FS	N	ROM file system support
CONFIG_EXT2_FS	Y	Second extended file system support
CONFIG_SYSV_FS	N	System V and Coherent file system support (read-only)
CONFIG_UDF_FS	N	UDF file system support (read-only)
CONFIG_UFS_FS	N	UFS file system support

**Table 3-36: Network File Systems**

Option	Value	Description
CONFIG_CODA_FS	N	Coda file system support (advanced network file system)
CONFIG_NFS_FS	Y	NFS file system support
CONFIG_NFS_V3	N	Provide NFSv3 client support
CONFIG_NFSD	N	NFS server support

---

**Table 3-36: Network File Systems (Continued)**

Option	Value	Description
CONFIG_SMB_FS	N	SMB file system support (to mount WfW shares, etc.)
CONFIG_NCP_FS	N	NCP file system support (to mount NetWare volumes)

**Table 3-37: Partition Types**

Option	Value	Description
CONFIG_PARTITION_ADVANCED	N	Advanced partition selection

**Table 3-38: Sound Support**

Option	Value	Description
CONFIG_SOUND	N	Sound support

**Table 3-39: USB Support**

Option	Value	Description
CONFIG_USB	N	Support for USB

**Table 3-40: Kernel Hacking**

Option	Value	Description
CONFIG_FRAME_POINTER	Y	Compile kernel with frame pointer
CONFIG_DEBUG_ERRORS	N	Verbose kernel error messages
CONFIG_DEBUG_USER	N	Verbose user fault messages
CONFIG_DEBUG_INFO	N	Includes debugging information in kernel binary
CONFIG_MAGIC_SYSRQ	N	Magic SysRq key

**Table 3-40: Kernel Hacking (Continued)**

Option	Value	Description
CONFIG_BLUECAT_KDBG	N	Include <code>kdbg</code> kernel debugger
CONFIG_DEBUG_LL	N	Kernel low-level debugging functions

**Table 3-41: Modular Advanced Power Management**

Option	Value	Description
CONFIG_BLUECAT_APM	N	Modular Advanced Power Management support

**Table 3-42: LynuxWorks Messenger Support**

Option	Value	Description
CONFIG_BLUECAT_IOPMAN	N	Enables LynuxWorks IOP Manager support
CONFIG_BLUECAT_MSNG	N	Enables Messenger Support

This chapter provides information about the BlueCat Linux demo systems supported in the iq80310 Board Support Package (BSP).

---

## Demo Systems

The following table shows the demo systems supported by the iq80310 Board Support Package (BSP).

Table 4-1: Demo Systems Supported by iq80310 BSP

Demo System	Default Boot Devices Supported	ROM Requirements	RAM Requirements
developer	<b>Flash</b> <b>Network</b> using the OS loader	3511 KB	15360 KB
osloader	<b>Flash</b> <b>Network</b> using the OS loader <b>Network</b> using RedBoot	741 KB	4096 KB
showcase	<b>Flash</b> <b>Network</b> using OS loader	2628 KB	13312 KB

### developer Demo System

Various functionalities are combined in one BlueCat Linux system: `developer` demonstrates the use of an FTP client, the bash shell, simple networking, remote debugging with GDB, and the VisualLynux program.

Refer to Chapter 4 of the *BlueCat Linux User's Guide* for a complete description of `developer` and how to use it.

## osloader Demo System

`osloader` is the BlueCat Linux OS loader system used to boot the 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.

---

## Using Selected RPM Packages

This section describes how to use selected RPM packages that are frequently deployed in the embedded systems environment.

### Using BusyBox

The BusyBox RPM package combines tiny versions of many common UNIX utilities into a single small executable. It provides minimalist replacements for most utilities found usually 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 counterparts; however, the options included provide the expected functionality and behave much like their GNU correlates.

### Creating a BlueCat Linux System for BusyBox

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

1. Create a new directory by typing

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

2. Set up the BlueCat Linux kernel configuration using the standard kernel configuration tools.

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

- Pick a kernel configuration after invoking `xconfig`, save the configuration, and exit the kernel configuration program.
- Copy the kernel configuration file to the `$BLUECAT_PREFIX/demo/busybox` directory.

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

---

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

---

- Create a BlueCat Linux Kernel Downloadable Image, `busybox.kernel`.

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

- Create a specification file (`busybox.spec`) with 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 755 /etc/rc.d/rc.sysinit
chmod 755 /bin /sbin
# End of File
```

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

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

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

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

::respawn:/bin/sh
```

---

**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.

---

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

```
#!/bin/sh

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

10. 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 root file system images. All references to the `src` directory need to be removed from the `developer` demo Makefile: edit out the phrase `cd src;` from the `this:` and the `clean:` fields. Change the line `KDI_NAME = developer` to `KDI_NAME = busybox` and then run the `make all` command.

---

## Booting BusyBox Images from a Network

Use the following procedure to boot the BlueCat Linux with BusyBox utility from a network using the BlueCat Linux OS loader. Refer to Chapter 2, “Downloading and Booting BlueCat Linux on the Target” for details on the 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
> boot
```

where *target\_board\_IP* is the IP address of the target, and *development\_host\_IP* is the IP address of the host.

## 2. The following screen output appears:

```
Uncompressing Linux..... done, booting the
kernel.
Linux version 2.4.2-1 (bin@build1) (gcc version 2.9-xscale-010413) #78
pTN aWG 24 17:14:40 MSD 2001
Processor: Intel XScale-80200 revision 0
Architecture: IQ80310 Evaluation Board
On node 0 totalpages: 8192
zone(0): 8192 pages.
zone(1): 0 pages.
zone(2): 0 pages.
Kernel command line: root=101
Calibrating delay loop... 722.53 BogoMIPS
Memory: 32MB = 32MB total
Memory: 30176KB available (850K code, 180K data, 52K init)
Dentry-cache hash table entries: 4096 (order: 3, 32768 bytes)
Buffer-cache hash table entries: 1024 (order: 0, 4096 bytes)
Page-cache hash table entries: 8192 (order: 3, 32768 bytes)
Inode-cache hash table entries: 2048 (order: 2, 16384 bytes)
POSIX conformance testing by UNIFIX
PCI BIOS init
  got res[90011000:9001103f] for resource 1 of PCI device 8086:1209
  got res[88100000:8811ffff] for resource 2 of PCI device 8086:1209
  got res[88120000:88120fff] for resource 0 of PCI device 8086:1209
PCI: Bus 2, bridge: PCI device 1011:0026
  IO window: 90011000-90011fff
  MEM window: 88100000-881fffff
PCI enable device: (PCI device 1011:0026)
  cmd reg 0x347
PCI enable device: (PCI device 8086:1209)
  cmd reg 0x147
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Starting kswapd v1.8
pty: 256 Unix98 ptys configured
block: queued sectors max/low 19986kB/6662kB, 64 slots per queue
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
Serial driver version 5.02 (2000-08-09) with MANY_PORTS SHARE_IRQ
SERIAL_PCI enabled
ttyS00 at 0xfe800000 (irq = 2) is a 16550A
ttyS01 at 0xfe810000 (irq = 3) is a 16550A
eepr0100.c:v1.09j-t 9/29/99 Donald Becker
http://cesdis.gsfc.nasa.gov/linux/drivers/eepr0100.html
eepr0100.c: $Revision: 1.36 $ 2000/11/17 Modified by Andrey V. Savochkin
<saw@saw.sw.com.sg> and others
eth0: Invalid EEPROM checksum 0xe472, check settings before activating
this device!
eth0: OEM i82557/i82558 10/100 Ethernet, 00:80:4D:46:22:B8, IRQ 1.
  Board assembly ffffff-255, Physical connectors present: RJ45 BNC AUI
MII
  Primary interface chip unknown-15 PHY #31.
  Secondary interface chip i82555.
  General self-test: passed.
  Serial sub-system self-test: passed.
```

```
Internal registers self-test: passed.
ROM checksum self-test: passed (0xbd8681d).
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 2048 bind 2048)
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
NetWinder Floating Point Emulator V0.95 (c) 1998-1999 Rebel.com
RAMDISK: Compressed image found at block 2622912
Freeing BlueCat RFS memory: 913K
VFS: Mounted root (ext2 filesystem).
Freeing init memory: 52K
serial console detected. Disabling virtual terminals.
init started: BusyBox v0.52pre (2001.08.13-21:46+0000) multi-call
binary

hush -- the humble shell v0.01 (testing)
```

## Using BusyBox Utilities

This section provides examples of how to use 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:/ect/rc.d/rc.sysinit

::respawn:/bin/sh
```
- `chmod`

```
/ # chmod a-x /sbin/reboot
/ # ls -la /sbin/reboot
-rw-r--r--  1 0      0                7720 Aug 24 2001 /sbin/reboot
/ # chmod 755 /sbin/reboot
/ # ls -la /sbin/reboot
-rwxr-xr-x  1 0      0                7720 Aug 24 2001 /sbin/reboot
```
- `echo`

```
/ # echo !!!!!!!!!!
!!!!!!!!!!
```
- `date`

```
/ # date
Fri Aug 24 19:00:05 UTC 2001
```

- `uname`

```

/ # uname -a
Linux (none) 2.4.2-1 #78 pTN aWG 24 17:14:40 MSD 2001 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 00:30:4D: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 the use of the utility.

## Creating a BlueCat Linux System for TinyLogin

Use the following procedure to create a BlueCat Linux image for TinyLogin:

1. Create a new directory:

```

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

```

2. Set up the BlueCat Linux kernel configuration by using the standard kernel configuration tools.

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

3. Pick a kernel configuration after invoking `xconfig`, save the configuration, and exit the kernel configuration program.
4. Copy the kernel configuration file to the `$BLUECAT_PREFIX/demo/tinylogin` directory.

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

---

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

---

5. Create a BlueCat Linux Kernel Downloadable Image (KDI) (`tinylogin.kernel`):

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

6. Create the specification 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/ttyl

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` symbolic links point to `/usr/bin/tinylogin`. This allows the user to change his/her password by simply typing `passwd`.

---

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

```

none /proc proc
none /dev/pts devpts

```

8. 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

```

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

```

console
tty1

```

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

```

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

```

11. Create the `local/shadow` file:

```

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

```

12. 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
```

13. Create a root file system image (`tinylogin.rfs`) by entering:

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

---

**NOTE:** The Makefile for the `developer` demo system can be used to produce the TinyLogin kernel and root file system images. All references to the `src` directory need to be removed from the `developer` demo Makefile: edit out the phrase `cd src;` from the `this` and the `clean` fields. Change the line `KDI_NAME = developer` to `KDI_NAME = tinylogin` and then run the `make all` command.

---

## Booting the TinyLogin Images from a Network

Use the following procedure to boot BlueCat Linux with the TinyLogin utility from a network using the BlueCat Linux OS loader. Refer to Chapter 2, “Downloading and Booting BlueCat Linux on the Target” for details.

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
```

The TinyLogin utility is loaded onto the target board and started automatically.

## Using TinyLogin

This section provides 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#

```

## Using GNU Zebra RPM Package

GNU Zebra is a GPL software package that manages a TCP/IP-based routing protocol. It takes a multi-server and multi-thread approach to resolve the complexity of the internet. GNU Zebra supports BGP4, BGP4+, OSPFv2, OSPFv3, RIPv1, RIPv2, and RIPng.

GNU Zebra is intended to be used as a Route Server and a Route Reflector. It is not a toolkit; it provides full routing power under a new architecture. GNU Zebra is unique in design in that it has a process for each protocol.

## Creating a BlueCat Linux System for Zebra

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

1. Create a new directory:

```
BlueCat:$ mkdir -p $BLUECAT_PREFIX/demo/zebra/local
```

2. Set up the BlueCat Linux kernel configuration by using the standard kernel configuration tools.

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

3. Pick a kernel configuration after invoking `xconfig`, save the configuration, and exit the kernel configuration program.

4. Copy the kernel configuration file to the `$BLUECAT_PREFIX/demo/zebra` directory.

```
BlueCat:$ cp .config
$BLUECAT_PREFIX/demo/zebra/zebra.config
```

---

**NOTE:** In the kernel configuration **Networking options** menu, the following options must be set to **Y**.

```
CONFIG_NETLINK=Y
CONFIG_RNETLINK=Y
```

By default, Zebra is configured to communicate with the kernel via the netlink socket.

---

5. Create the BlueCat Linux KDI (`zebra.kernel`).

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

6. Create a root file system specification file (`zebra.spec`) with 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
# Standard 16550 serial driver device
mknod /dev/ttyS0 c 4 64
mknod /dev/ttyS1 c 4 65
```

```

mkdir -p /lib/security
mkdir -p /usr/lib
mkdir /bin
mkdir /sbin
mkdir -p /etc/rc.d
mkdir -p /etc/pam.d
mkdir -p /etc/xinetd.d
mkdir -p /etc/zebra
mkdir /proc
mkdir /tmp
mkdir -p /usr/bin
mkdir -p /usr/sbin
mkdir -p /var/run
mkdir -p /usr/libexec

mkdir -p /var/log/zebra

mkdir /root

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

chmod 0666 /dev/ptmx

cp ./local/fstab ./local/passwd ./local/inittab ./local/mtab /etc
cp ./local/other /etc/pam.d
cp ./local/rc.sysinit /etc/rc.d
cp ./local/hosts /etc
cp ./local/protocols /etc
cp ./local/resolv.conf /etc
cp ${BLUECAT_PREFIX}/etc/pwdb.conf /etc
cp ${BLUECAT_PREFIX}/etc/nsswitch.conf /etc
cp ${BLUECAT_PREFIX}/etc/services /etc

cp ${BLUECAT_PREFIX}/etc/security /etc

cp ./local/shadow /etc
cp ./local/pam.d /etc
cp ./local/xinetd.d/* /etc/xinetd.d
cp ./local/zebra.conf /etc/zebra/

cp ${BLUECAT_PREFIX}/lib/libnss_files-*.so /lib
cp ${BLUECAT_PREFIX}/lib/libnss_dns-*.so /lib
cp ${BLUECAT_PREFIX}/lib/libpwdb.so /lib
cp ${BLUECAT_PREFIX}/lib/security /lib

cp ./local/empty /var/log/wtmp

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

cp ${BLUECAT_PREFIX}/lib/security/pam_permit.so /lib/security

cp ${BLUECAT_PREFIX}/etc/xinetd.conf /etc

cp ${BLUECAT_PREFIX}/usr/bin/telnet /usr/bin

cp ${BLUECAT_PREFIX}/etc/shells /etc
chmod 644 /etc/shells

cp ${BLUECAT_PREFIX}/etc/group /etc

#

```

```
# General Binaries
#
lcd ${BLUECAT_PREFIX}/bin
cp ping mount bash cat ls hostname ps /bin
cp login /bin
ln -s /bin/bash /bin/sh

cp ${BLUECAT_PREFIX}/usr/bin/vtysch           /usr/bin

# internet services utils
cp ${BLUECAT_PREFIX}/usr/sbin/xinetd         /usr/sbin
cp ${BLUECAT_PREFIX}/usr/sbin/in.telnetd     /usr/sbin
cp ${BLUECAT_PREFIX}/usr/sbin/zebra         /usr/sbin

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

chmod 755 /bin /sbin /usr/bin /usr/sbin

# End of File
```

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

```
id:1:initdefault:

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

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

```
#!/bin/sh

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

mount -a
xinetd -stayalive -reuse

hostname myhostname

zebra -d
```

9. Create the `local/zebra.conf` file with the following contents:

```
!
! zebra configuration file
!
hostname Router
password zebra
enable password zebra
!
! Interface's description.
!
interface lo
ip address 127.0.0.1/8

interface eth0
ip address 172.17.3.11/16

!
! Static default route.
!
ip route 213.24.0.0 255.255.0.0 172.17.0.1
```

---

```
log stdout
```

---

**NOTE:** This configuration file sets the password to **zebra**. The user has to enter this password when connecting to Zebra or changing the Zebra configuration mode by entering the **enable** command at the command prompt.

---

10. Copy the `fstab`, `passwd`, `mtab`, `other`, `hosts`, `protocols`, `resolv.conf`, `shadow`, `pam.d/*`, `xinetd.d/*`, and `empty` files from the `$BLUECAT_PREFIX/demo/developer/local` directory to the `$BLUECAT_PREFIX/demo/zebra/local` directory.

11. Create a root file system image (`zebra.rfs`) by entering:

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

---

**NOTE:** The Makefile for the `developer` demo system can be used to produce the Zebra kernel and RFS images. All references to the `src` directory need to be removed from the `developer` demo Makefile: edit out the phrase `cd src;` from the `this` and the `clean` fields. Change the line `KDI_NAME = developer` to `KDI_NAME = zebra` and then run the `make all` command.

---

## Booting the Zebra Images from a Network

Use the following procedure to boot the BlueCat Linux image with Zebra over a network using the BlueCat Linux OS loader. Refer to Chapter 2, “Downloading and Booting BlueCat Linux on the Target” for details about OS loader.

At the OS loader prompt (`>`), type the following commands:

```
> set IF eth0
> set IP target_board_IP
> set HOST development_host_IP
> set CMD ramdisk_size=8192
> set KERNEL tftp zebra.kernel
> set RFS tftp zebra.rfs
> boot
```

The Zebra utility is loaded onto the target and automatically started.

## Using Zebra

This section provides an example of using the Zebra utility.

```
myhostname login: root
bash-2.04# ifconfig
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
          UP BROADCAST RUNNING 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
          Interrupt:1

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:3904  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

bash-2.04# ping -c 2 172.17.0.1
PING 172.17.0.1 (172.17.0.1) from 172.17.3.11 : 56(84) bytes of data.
Warning: time of day goes back, taking countermeasures.
Warning: time of day goes back, taking countermeasures.
64 bytes from 172.17.0.1: icmp_seq=0 ttl=255 time=0 usec
64 bytes from 172.17.0.1: icmp_seq=1 ttl=255 time=650 usec

--- 172.17.0.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.000/0.325/0.650/0.325 ms

bash-2.04# ping -c 3 213.24.253.87
PING 213.24.253.87 (213.24.253.87) from 172.17.3.11 : 56(84) bytes of
data.
64 bytes from 213.24.253.87: icmp_seq=0 ttl=254 time=902 usec
Warning: time of day goes back, taking countermeasures.
64 bytes from 213.24.253.87: icmp_seq=1 ttl=254 time=1.821 msec
64 bytes from 213.24.253.87: icmp_seq=2 ttl=254 time=898 usec

--- 213.24.253.87 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.898/1.207/1.821/0.434 ms

bash-2.04# telnet localhost 2601
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Hello, this is zebra (version 0.91a).
Copyright 1996-2001 Kunihiro Ishiguro.

User Access Verification

Password: zebra
Router> enable
Password: zebra
Router# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       B - BGP, > - selected route, * - FIB route

C>* 127.0.0.0/8 is directly connected, lo
```

```
C>* 172.17.0.0/16 is directly connected, eth0
S>* 213.24.0.0/16 [1/0] via 172.17.0.1, eth0
Router#
```



The following table shows the device drivers supported by the iq80310 BSP.

**Table 5-1: The Device Drivers Supported by the iq80310 BSP**

Hardware Device	Device Drivers	Location in Source Tree	Kernel Configuration Options	Notes
<b>UART</b> Two 16550-compatible devices	serial.c	drivers/char	CONFIG_SERIAL CONFIG_SERIAL_CONSOLE	
<b>LED Display</b>	led_iq80310.c led_iq80310.h	drivers/char include/linux	CONFIG_LED_IQ80310	
<b>Ethernet</b> Intel 82559 Adapter	eeepro100.c	drivers/net	CONFIG_EEPRO100	
<b>NOR Flash</b>	iq80310.c	drivers/mtd	CONFIG_MTD_IQ80310 CONFIG_MTD_IQ80310_PART	
<b>Gbit Ethernet</b> Intel PRO/1000		\$BLUECAT_PREFIX/ usr/src/ e1000-3.0.10/src		The driver is installed as a Linux kernel module.

## Using Intel PRO/1000 Ethernet Driver

This section explains how to set up and use the Intel PRO/1000 Ethernet driver with the BlueCat Linux kernel. The **showcase** demo system is used to illustrate the procedure. Because this driver is not part of the statically linked BlueCat Linux kernel, additional steps are necessary.

To enable support for the Intel PRO/1000 Ethernet driver:

1. Set up the BlueCat Linux environment.

```
$ . SETUP.sh iq80310
```

2. To enable support for kernel modules, run the following commands:

```
BlueCat:$ cd "$BLUECAT_PREFIX/demo/showcase"  
BlueCat:$ make xconfig
```

and turn on the **Enable loadable module support** option in the **Loadable module support** submenu.

3. Rebuild the kernel by entering the following:

```
BlueCat:$ cd "$BLUECAT_PREFIX/demo/showcase"  
BlueCat:$ make kernel
```

4. Rebuild the driver for the Intel PRO/1000 adapter by typing:

```
BlueCat:$ cd "$BLUECAT_PREFIX/usr/src/  
e1000-3.0.10/src"  
BlueCat:$ make clean  
BlueCat:$ make  
BlueCat:$ make install
```

5. To include the driver into the `showcase` root file system, update the `showcase.spec` file as follows:

```
BlueCat:$ cd "$BLUECAT_PREFIX/demo/showcase"  
  
BlueCat:$ echo `mkdir -p /lib/modules' >>\`  
showcase.spec  
  
BlueCat:$ echo `cp ${BLUECAT_PREFIX}/lib/modules/  
2.4.2-1-iq80310/net/e1000.o /lib/modules/'\  
>> showcase.spec  
  
BlueCat:$ echo `cp ${BLUECAT_PREFIX}/sbin/  
insmod /sbin/' >> showcase.spec
```

6. Rebuild the `showcase` embedded system

```
BlueCat:$ make rootfs kdi
```

After these steps have been performed, the Intel PRO/1000 Ethernet driver can be activated by running a `showcase` demo system with the following commands:

```
bash# insmod /lib/modules/e1000.o  
  
Intel(R) PRO/1000 Network Driver - version 3.0.10  
Copyright (c) 1999 -2001 Intel Corporation  
  
Intel(R) PRO/1000 Network Connection  
eth1: Mem:0x80000000 IRQ:7 Speed:1000 Mbps Duplex:Full
```

```
bash# ifconfig eth1 board_IP_address
```

where *board\_IP\_address* is the IP address of the secondary Ethernet interface.

---

## Flash Support

The BlueCat Linux Flash/JFFS management subsystem is extended with support for the specific flash devices on the IQ80310 board. This provides a standard set of flash management features available in BlueCat Linux. For a detailed description of flash memory support and the Journalling Flash File System (JFFS), refer to Chapter 6 of the *BlueCat Linux User's Guide*.

### Configuring Custom Flash Chips

The following files in the BlueCat Linux kernel must be updated to support custom flash chips:

- `include/asm-arm/arch-iq80310/hardware.h`

Change the following lines

```
#define FLASH_START          HDW_FLASH1_START
#define FLASH_BASE           HDW_FLASH1_BASE
#define FLASH_SIZE           HDW_FLASH1_SIZE
```

to

```
#define FLASH_START          physical_address
#define FLASH_BASE           virtual_address
#define FLASH_SIZE           size
```

where

*physical\_address*      An address of the custom flash chip on the internal bus

*virtual\_address*      A virtual address to which the custom flash chip is mapped (If the chip size is less than 8 Mb, then the HDW\_FLASH1\_BASE value can be used.)

*size*                    Size of the custom flash

The values used in the iq80310 BSP are:

*physical\_address*      0x90800000

*virtual\_address*        0xFF800000

```
size 0x00800000
```

- `drivers/mtd/ig80310.c`

Change the following line

```
buswidth: 1,
```

to

```
buswidth: bus_width
```

where `bus_width` is the width of the flash bus.

If the custom flash chips do not support the CFI autodetection, then a detection module must be written for the device. For information on writing the detection module, refer to Chapter 6 in the *BlueCat Linux User's Guide*.

---

## PCI Support

This section describes specificities of PCI support in BlueCat Linux for the IQ80310 board.

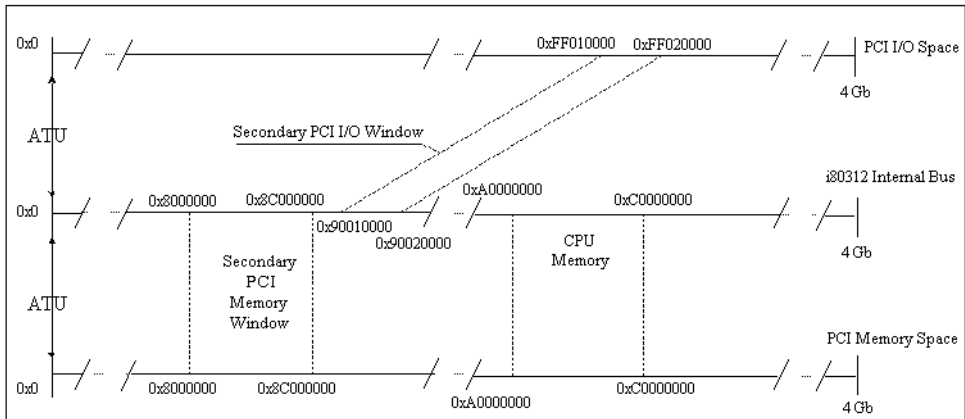
### Using Drivers for External PCI Devices with BlueCat Linux for IQ80310 Boards

#### Configuring PCI Buses

Although the RedBoot monitor configures the PCI buses, BlueCat Linux does not use the existing PCI configuration, and recognizes PCI by itself.

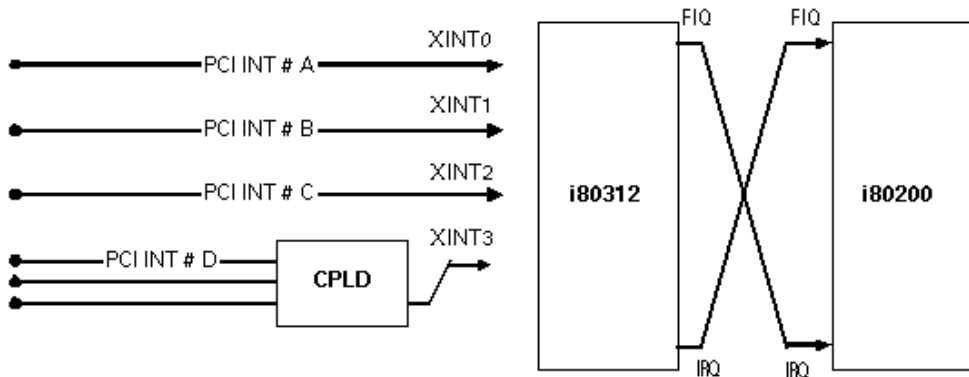
At startup, BlueCat Linux fully initializes the PCI bus and configures resources for devices on it. Also, BlueCat Linux configures memory windows for the secondary PCI bus. The primary PCI bus is not accessible under BlueCat Linux.

The following figure illustrates the configuration of PCI memory windows set up by BlueCat Linux:



**Figure 5-1: Mapping of Addresses Between the PCI and Internal Buses**

The IQ80310 board supports all the PCI interrupt lines for the secondary PCI bus, but the PIC INT#D interrupt line is routed through CPLD.



**Figure 5-2: Interrupt Routing on the IQ80310 Board**

## Possible Incompatibilities between BlueCat Linux for IQ80310 and Existing PCI Device Drivers

In general, BlueCat Linux for IQ80310 supports all correct Linux PCI device drivers.

The following is a list of possible errors in PCI device drivers that may affect the functionality of the drivers on IQ80310:

- Invalid access to the PCI memory space

The drivers must use `readb`, `readw`, `readl`, (and the `write*` counterparts) to access the PCI memory space. References to the PCI memory space as to a regular memory can cause incorrect behavior of the drivers, because I/O transactions on the i80200 processor require additional delays. The address argument to the `read*` and `write*` functions must be obtained by calling the `ioremap` function with a bus address as an argument.

The following example demonstrates the usage of these functions:

```
Adapter->HardwareVirtualAddress =
    ioremap(pci_resource_start(pdev, BAR_),
           pci_rep_len(pdev, BAR_0));

...

#define E1000_READ_REG(reg)
    ((Adapter->MacType >= MAC_LIVENGOOD) ?
     readl(&((PE1000_REGISTERS)Adapter->HardwareVirtualAddress->reg)) :
     readl(&((POLD_REGISTERS)Adapter->HardwareVirtualAddress->reg)))
```

- Time-consuming operations in interrupt handlers

Some Linux device drivers (not just PCI drivers) perform much of the data processing in the interrupt handlers instead of scheduling it to a dedicated thread or a bottom-half handler. Such a practice lowers the responsiveness of the Linux operating system in general, but it has additional impact on IQ80310 because PCI INT#A-INT#C interrupts cannot be selectively masked in hardware.

The trick to masking problems in certain cases is performed by routing PCI interrupts from the secondary PCI bus to the primary PCI bus. This acts effectively as masking. Unexpected interrupts on the primary PCI bus is the disadvantage of this approach. Such masking behavior can be enabled by adding the `CONFIG_IQ80310_PCI_MASK_ROUTING` kernel configuration option.

- Incorrect allocation of the memory for PCI bus-mastering transfers

The `pci_alloc_consistent` function must be used to allocate the memory for PCI bus-mastering transfers. Some older drivers (written primarily for the x86 architecture) still use the `__get_free_pages` function instead. This may lead to the cache consistency problems (the `pci_alloc_consistent` function always allocates non-cached memory).

The following example shows usage of `pci_alloc_consistent`:

```
Adapter->TxDescriptors =  
    pci_alloc_consistent(pdev, size, & Adapter->TxDescDMA);
```

- PCI resources allocation

PCI resources are automatically allocated the by the kernel at startup. If the BlueCat Linux kernel cannot allocate a resource for a device, then `pci_resource_start(pdev, rsc)` for this resource is 0.

The following example shows how to check if a resource has been allocated by the kernel:

```
ctrlptr_phys = pci_resource_start(ACCESS_FBINFO(pci_dev), 0);  
  
...  
  
if (! ctrlptr_phys)  
    printk(KERN_ERR "matroxfb: control registers are"  
           "not available, matroxfb disabled\n");
```



This chapter describes the BlueCat Linux Application Interface specific to the IQ80310 target board.

---

## Advanced MMU/Cache Management Services

The BlueCat Linux kernel for IQ80310 provides a set of advanced MMU/Cache management services for the Intel 80200 processor. These are declared in the `$BLUECAT_PREFIX/usr/src/linux/include/asm/i80200.h` file.

### 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, say for a device driver for a sound device that provides audio data to be played, this might be the only way to meet 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 run as fast as possible on the 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 Service Reference

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

- `void i80300_dcache_lock(unsigned int add, int lines)`

Locks 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 ensures that 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)`

Locks 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 ensures execution from the instruction code. The contents of the area are not pushed out of the cache by the 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 no longer needed.

- `void i80200_itlb_lock(unsigned int addr)`

Translates and locks an ITLB entry.

The locked ITLB entry ensures that the instruction fetch from the page at the specified virtual address goes 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 ensures that the accesses to the page at the specified virtual address 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 ITLB configuration is no longer needed.

- `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 no longer needed.

- `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 are performed in the program order without write combining operations in the write buffer, regardless of the values of the page attributes.

---

## LED Display Control

BlueCat Linux for IQ80310 has a device driver for the on-board LED display. The driver is configured using the `CONFIG_LED_IQ80310` kernel configuration option, which can be enabled in the **Character Devices** submenu of the `make xconfig` interface. The driver provides an `ioctl()` interface for the user-space applications, defined in the `$(BLUECAT_PREFIX)/usr/src/linux/include/linux/led_iq80310.h` file.

The driver is accompanied by the `$(BLUECAT_PREFIX)/bin/iq80310_led` utility that provides control of the LED display from the shell command line or shell scripts. The utility can be used to control the LED display on the target. The utility syntax is as follows:

```
iq80310_led value
```

where `value` is a numerical value to be displayed on the LED. For example:

```
# iq80310_led 1.2
# iq80310_led 55
```

Please note that the `iq80310_led` utility requires that an LED driver special device file (`/dev/iq80310_led`) is created in the root file system. The major number of this device is `233`. For example, the special `/dev/iq80310_led` file can be created by the following command:

```
mknod /dev/iq80310_led c 233 0
```

---

## Application Accelerator Unit Services

The BlueCat Linux kernel for IQ80310 boards provides a set of Application Accelerator Unit support services for the Intel 80312 I/O chip, declared in the `$(BLUECAT_PREFIX)/usr/src/linux/include/asm/arch/iop310-aau.h` file.

## 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 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 the 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 IQ80310 provides a set of PCI DMA transfer services for the Intel 80312 I/O chip. These are declared in the `BLUECAT_PREFIX/usr/src/linux/include/asm/arch/iop310-dma.h` file.

## PCI DMA Transfers Services Overview

The Intel 80312 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 control over a DMA channel that can be one of the following:

`IOP310_DMA_P0`: PCI Primary 1

`IOP310_DMA_P1`: PCI Primary 2

`IOP310_DMA_S0`: PCI Secondary 1

---

**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 address [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 incomplete. This service is used when DMA channel errors occur.

- `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 IQ80310 provides a set of Messaging Unit support services for the Intel 80312 I/O chip. These are declared in the `$BLUECAT_PREFIX/usr/src/linux/include/asm/arch/iop310-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 registers.

- `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 the inbound free queue and the callback function to handle the inbound messages are the parameters of this service.

- `int mu_cq_enable(u32 mu_context)`



## Performance Monitoring Services Reference

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

---

**NOTE:** The PMU can be used by only one user.

---

- `int pmon_claim(void)`  
Acquires the ownership of the PMU services.
- `int pmon_release(int claim)`  
Releases the PMU services. This service must be called with the identifier returned by `pmon_claim()`.
- `int pmon_start(int claim, int mode)`  
Starts the PMU operation. `mode` specifies what statistics to capture. `mode` can be one of the following:

**Table 6-1: PMU Modes**

Mode	Description
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 80312 I/O Companion Chip)
I80312_PMU_MODE4	Secondary PCI bus and internal agents (External Masters 3..5 and Intel 80312 I/O Companion Chip)
I80312_PMU_MODE5	Intel 80312 I/O companion chip internal bus, DMA Channels and Application Accelerator
I80312_PMU_MODE6	Intel 80312 I/O companion chip internal bus, PATU, SATU and Intel 80200 processor
I80312_PMU_MODE7	Intel 80312 I/O companion chip internal bus, Primary PCI bus, Secondary PCI bus and Secondary PCI agents (External Masters 0..5 & Intel 80312 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    PECR[NUM_OF_PECR]; /* Programmable Event
                               Counter Register 1 -14
                               */
    u32    PECR_CNT[NUM_OF_PECR]; /* Overflow counter for PECR1-14 */
} pmon_res_t;
```

---

## Hints for Xscale-Specific Optimizations

A recommended optimization technique is that if the cache hit rate is high for the 32 KB data cache, the overall performance increases considerably. It is recommended that the user consider optimizing custom applications and device drivers by ensuring that data is moved within 32 KB blocks whenever possible.



---

CHAPTER 7 *Defect Fixes and Known Problems*

The following table shows the defects fixed in this release of BlueCat Linux:

**Table 7-1: Defect Fixes in BlueCat Linux**

Platform	Subcomponent	ID	Summary
All	BlueCat Linux Misc	16057	<b>Ctrl-C, Ctrl-Z</b> , etc. do not work from the shell.
Windows NT/2000 Cross	BlueCat Installation	16130	Windows install does not installing on non-C drives.
All	BlueCat Linux OS loader	16358	The BLOSH <b>ntar</b> command hangs the system.
All	BlueCat Misc	16684	<code>make menuconfig</code> does not work on a Windows host.
All	BlueCat Linux Misc	17308	Certain sequence of file updates sometimes causes FFS to crash after reboot.

## **IQ80310 Target Board Problems and Limitations**

- This release of BlueCat Linux supports binary compatibility with Revision D of the IQ80310 board. However, only very limited testing has been accomplished on Revision D targets. The currently shipping version of BlueCat Linux for IQ80310 has been extensively tested for and supports Revision F of the board.
- The IQ80310 board lacks some functionality for controlling the interrupt signal for the PCI INT#A-INT#C interrupt lines. Specifically, selective masking for these interrupts is not supported in the hardware. This limitation forces the handlers for the PCI INT#A-INT#C interrupts to be executed as if they were registered with the SA\_INTERRUPT flag. This lowers the responsiveness of the BlueCat Linux kernel.

However, if the board is inserted into a passive backpane, the IQ80310 board's ability to route PCI interrupts to a host can be used to mask them, without side effects or performance impact. Support for such a masking method is implemented in the BlueCat Linux kernel and can be enabled with the `CONFIG_IQ80310_PCI_MASK_ROUTING` kernel configuration option.

To change the `CONFIG_IQ80310_PCI_MASK_ROUTING` using the `make xconfig` interface, the user must select the Mask PCI interrupts by routing them to host item from the **IQ80310-Specific Options** submenu of the **System Type** menu.

- When booting a BlueCat Linux demo system on the IQ80310 board using the RedBoot monitor, the kernel may sporadically crash on initialization. After resetting the board, the same demo system usually starts successfully. This problem does not exist when BlueCat Linux is booted using the BlueCat OS loader.
- The speed of memory transfers on the IQ80310 board is about 20-40 Mb/s, however, according to the board's documentation, the speed should be 69 Mb/s in the worst case.
- All demo systems (see Chapter 4 of the Board Support Guide) for the IQ80310 board are preconfigured to use 32 MB of RAM. The demo systems can be configured for other RAM sizes by setting the `CONFIG_IQ80310_RAMSIZE` kernel configuration option to the desired size in megabytes. Autodetection of the size of RAM is also supported, and can be enabled by setting the `CONFIG_IQ80310_RAMSIZE` option to 0.

To change `CONFIG_IQ80310_RAMSIZE` using the `make xconfig` interface, the user must select the **Support memory size (in MB)** item from the **IQ80310-Specific Options System Type** submenu of the **Select Type** menu.

- If `mkrootfs` is terminated (either by error or by a signal), it tries to clean all of its temporary files before exiting. However, due to certain features of the Cygwin environment, 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.
- The `tc1x` RPM package is not included in the Windows-hosted distribution.
- On Windows hosts some file permissions (including `r` and `s`) always have the default values. To set permissions different from the default ones, the `chmod` command should be used in the `.spec` file.

