



Linux[®] and VxWorks[®] Resource Guide

Broadcom Confidential

REVISION HISTORY

<i>Revision</i>	<i>Date</i>	<i>Change Description</i>
47xx-RG00-R	12/05/02	Initial Release

Broadcom Confidential

Broadcom Corporation
P.O. Box 57013
16215 Alton Parkway
Irvine, California 92619-7013
© Copyright 2002 by Broadcom Corporation
All rights reserved
Printed in the U.S.A.

Broadcom[®], the pulse logo, and ILine are trademarks of Broadcom Corporation and/or its subsidiaries in the United States and certain other countries. MIPS32 is a trademark of MIPS Technologies, Inc. VxWorks, Wind River Systems, Tornado and WindView are registered trademarks and Crosswind and WindSh are trademarks of Wind River Systems, Inc. MIPS32 is a trademark of MIPS Technologies, Inc. Intersil and Prism are registered trademarks of Intersil Corporation. Intel is a registered trademark of Intel Corporation. Linux is a registered trademark of Linus Torvalds. GoAhead is a registered trademark of GoAhead Software, Inc. Procomm Plus is a registered trademark of Symantec Corporation. Red Hat is a registered trademark of Red Hat Inc. Win32 is a registered trademark of Microsoft Corporation. Lucent is a registered trademark of Lucent Technologies. Fujitsu is a registered trademark of Fujitsu, Ltd. St is a registered trademark of STMicroelectronics. All other trademarks are the property of their respective owners.

TABLE OF CONTENTS

Introduction	1
Supported Reference Platforms	1
BCM94710AP	1
BCM4702	2
Reference Software Assumptions	2
GPIO Pins	2
Support for Reading/Writing GPIO Register	3
Interrupt Map	4
PCI_INTB Signal	4
SDRAM Configuration	4
SDRAM Initialization Control Register	5
SDRAM Configuration Register	6
SDRAM Refresh Control Register	6
Flash Configuration	7
Flash with Linux	7
Tested Flash Devices	7
UART Removal	8
Release Packages	8
BCM947XX Linux Packages	10
What You Get	11
Additional Tools	11
BCM430x 802.11b Wireless Support	12
BSP Package	12
Residential Gateway Package	13
Adding Additional Services	13
BSP Package	13
Residential Gateway Package	13
Incorporating Binaries into the File System	13
Building the Linux Image	13
Pruning the C Libraries	13
Statistics Reporting	14
PCI Device Support	14

Tulip Example Driver	14
PCMCIA Device Support	14
Intersil® Example Driver	14
Third-Party Device Drivers	15
Building the Driver	15
Copying the Driver to the File System	15
Invoking The Driver	15
Debugging Linux-Based Code.....	16
User-Level Applications.....	17
brctl.....	17
BusyBox	18
httpd	18
iptables	19
netconf.....	19
nvrAm	21
rc	21
udhcp.....	22
Additional Information.....	22
Software Packaging Received.....	23
Required Tools	24
Installing the Package Source	24
Preparation Checklist	24
UART and SIO	24
DRAM.....	24
NVRAM	25
Interrupts	25
Integrating with THG.....	25
Circumventing the NAT	25
Building the VxWorks Image	26
Debugging with CrossWind	26
Modifying CONFIG.H	26
Preparing Source Files.....	27
Choosing Internal or External UART	27
Tornado Target Server	27



Debugging the Code	28
Statistics Reporting	28
Accessing NVRAM	29
BCM430x 802.11b Wireless Support	29
Configuring the BCM430x	30
HTML-Based Wireless Configuration	30
PCI Device Support	30
PCMCIA Device Support	31
PMON Boot Loader	33
PMON Environment Variables	34
set	34
unset	34
nvserial	35
Programming PMON Into Flash	35
Connecting a Serial Monitor to a BCM47xx-based Platform	36
When PMON Boots	37
Erasing the NVRAM Area	37
Loading an OS Image Using PMON	37
Adding a Header	38
Preparing visionICE for Use with the BCM4710	39
Preparing Your Code For Debugging With VisionCLICK	39
Configuring the EJTAG Connection	39
Configuring the IP Address Parameters	40
Debugging with a Terminal Application	40
Verifying the Connection	41
Upgrading the visionICE Firmware	41
Main Firmware Upgrade	41
Custom Flash Algorithm Firmware Upgrade	41
Updating a visionCLICK Installation to Support the BCM4710	42
Register Configuration Files	42
Creating a Project File	42
Writing BCM4710 Register Descriptions to the visionICE	43
Programming a Flash Device	44
Getting Ready	44
Notes on Big-Endian Support	44

Using the SWAP Utility.....	44
Preparing a Flash Binary Image.....	45
Writing to the Flash Device.....	45
Basic Debugging	46
Debugging a Linux Image	46
NVRAM	47
NVRAM Protection	47
Accessing NVRAM Information	47
Getting an Initial NVRAM Environment	48
Reading/Writing NVRAM Environment Variables.....	48
NVRAM Format	48
NVRAM Header.....	48
NVRAM Environment Variables	49

Broadcom Confidential



LIST OF TABLES

Table 1: GPIO Functions	3
Table 2: Linux and VxWorks Interrupt Map	4
Table 3: SDRAM Configuration	4
Table 4: SDRAM Initialization Control Register	5
Table 5: SDRAM Configuration Register	6
Table 6: SDRAM Refresh Control Register	6
Table 7: Serial Monitor Configuration	36
Table 8: EJTAG Connections	39
Table 9: Register Configuration Files	42
Table 10: Flash Device Configurations	44
Table 11: Flash Programming Algorithms	45
Table 12: Flash Algorithm Start Addresses	46
Table 13: NVRAM Environment Variables—General Parameters	49
Table 14: NVRAM Environment Variables—Interface Parameters	50
Table 15: NVRAM Environment Variables—Firmware Upgrade Parameters	50
Table 16: NVRAM Environment Variables—TCP/IP Parameters	50
Table 17: NVRAM Environment Variables—Firewall Parameters	50
Table 18: NVRAM Environment Variables—LAN Filters Parameters	51
Table 19: NVRAM Environment Variables—Port Forward Parameters	51
Table 20: NVRAM Environment Variables—DHCP Server Parameters	51
Table 21: NVRAM Environment Variables—Web Server Parameters	51
Table 22: NVRAM Environment Variables—PPPOE Parameters	52
Table 23: NVRAM Environment Variables—802.11b Parameters	52
Table 24: NVRAM Environment Variables—Restore Defaults Parameters	53

Broadcom Confidential



INTRODUCTION

The BCM47xx line of ICs provide a system-on-a-chip (SOC) targeted at embedded residential gateways and wireless access points. To facilitate development of these applications, Broadcom will also supply collateral such as residential gateway reference code, access point reference code, build tools and pretested platform binaries that can be used with one of our reference platforms. The BCM47xx Resource Guide introduces customers to the packages and tools that are included in a release.

Refer to either the LinuxReadme.txt or the VxReadme.txt that was included with the source-code distribution. It contains release-specific information.

SUPPORTED REFERENCE PLATFORMS

Several sections of this document refer to BCM47xx-based reference platforms. Two Broadcom-supplied reference platforms are supported by this document: the BCM94710AP and the BCM94710D.

BCM94710AP

The BCM94710AP is a small-form-factor design with the following features:

- 4 switched 10/100-Mbit LAN Ethernet ports
- 1 10/100 Mbit WAN Ethernet port
- 1 HPNA LAN port
- 4-MB Flash memory
- 16-MB RAM
- 2 serial RS-232 connectors
- 1 EJTAG connector
- 2 USB ports for use as a slave interface
- 2 Mini PCI slots for additional functionality (such as 802.11b)

The BCM94710AP is intended to demonstrate a final product based on the BCM47xx. When combined with a BCM94301MP 802.11b Mini PCI adapter, the BCM94710AP can implement an 802.11b Access Point or an 802.11b Residential Gateway.

Note

In addition to 802.11b support, Broadcom offers 54g™, which is fully compliant with the current 802.11g draft specification.



BCM94710D

The BCM94710D is centered around an ATX form factor with the following features:

- 1 10/100-Mbit LAN Ethernet port
- 1 10/100-Mbit WAN Ethernet port
- 1 HPNA LAN port
- 4-MB Flash memory
- 8-MB RAM
- Dual serial 9-pin RS-232 connectors
- 2 PCI slots
- 1 PCMCIA (16-bit) slot¹
- 1 EJTAG connector

The BCM94710D is intended as a development platform. A vendor can add additional hardware via PCI or PCMCIA slots, perform kernel debugging using the dual serial ports and provide additional SW services with the expanded 8 MB of RAM.

BCM4702

The BCM4702 is a lower-cost version of the BCM4710 that does not contain an HPNA interface. This allows for deployment of a residential gateway or access point (AP) product that does not include HPNA functionality.

For the most part, the Linux[®] and VxWorks[®] packages will work correctly with either a BCM4710 or a BCM4702. The only changes that need to be made are:

- When embedding default NVRAM variables into the PMON binary, a different NVRAM default file should be used. See “Creating a Default NVRAM Image” on page 35 for more information on the 4702_nvdata.txt file.
- The Linux kernel must be instructed to not load the HPNA device driver. This can be done by properly setting the kernel_mods NVRAM variable. See “NVRAM Environment Variables” on page 49.

REFERENCE SOFTWARE ASSUMPTIONS

GPIO PINS

The reference software makes certain assumptions about how the GPIO pins on the BCM47xx are connected. Table 1 shows the GPIO pins and their functions for each of the supported reference platforms. N/A indicates that the gpio pin is not used by the reference software for the given platform. N/C indicates not connected, but still may be configured for other purposes. If a gpio pin is needed for your design, use one of the currently unused pins. If a currently defined pin must be used, you must make changes to the reference software.

1. The PCMCIA slot can be configured for either 3.3V or 5V operation. For 5V, ensure that a zero-ohm resistor is present on R34 of the board. For 3.3V, ensure that a zero-ohm resistor is present on R48 of the board. A resistor must be present on one or the other, not both, for proper PCMCIA operation.

Table 1: GPIO Functions

GPIO Pin	BCM94710AP	BCM94710D
gpio0	cardbus_reset#	pcmcia_wp
gpio1	euart_int1/2 (16551)	euart_int1/2
gpio2	spi_cs# (bcm5325)	mii0 ephy_reset
gpio3	spi_clk (bcm5325)	eide_irqr
gpio4	spi_do (bcm5325)	N/A
gpio5	spi_di (bcm5325)	pcmcia_stschg
gpio6	wan_reset (bcm5221)	mii1 ephy_reset
gpio7	N/C	pcmcia_reset
gpio8	N/A	N/A

When using the BCM5325 (connections shown above) in reverse MII mode, the boardtype NVRAM parameter must be set to "bcm94710ap."² For more information, see "NVRAM Environment Variables" on page 49.

SUPPORT FOR READING/WRITING GPIO REGISTER

The GPIO pins can be read from or written to in user space (Linux release only). An example of a user-level application (gpio.c) to use this kernel GPIO driver is below:

```
#include <stdio.h>
#include <stdlib.h>
int
main(int argc, char **argv)
{
    unsigned int val = 0;
    if (argc >= 2) {
        val = strtoul(argv[1], NULL, 0);
        fwrite(&val, 4, 1, stdout);
    } else {
        fread(&val, 4, 1, stdin);
        printf("0x%08x\n", val);
    }
    return 0;
}
```

To read the GPIO values:

```
gpio < /dev/gpio/in
```

To write the GPIO values:

```
gpio 0xff > /dev/gpio/out
```

Basically, there are now two entries in /dev/gpio that use file operations for reading/writing values.

You could extend the sample gpio.c file to periodically read the value of the GPIO hooked up to your reset button. If you also link in the nvrn libraries, you can restore the defaults when the reset button is pressed.

2. The correct NVRAM configuration file to start with is /bootrom/ap_nvdata.txt.

INTERRUPT MAP

Both the Linux and VxWorks drivers use a common interrupt map. See Table 2.

Table 2: Linux and VxWorks Interrupt Map

<i>Interrupt</i>	<i>Function</i>
0	extif
1	enet0
2	enet1
3	ILine™
4	pci

PCI_INTB SIGNAL

The PCI_INTB signal is not supported by the driver. Using this signal causes the BCM47XX software builds to fail. All PCI devices must share the PCI_INTA signal.

To force PCI_INTA-only interrupts on the BCM94710D platform, place a jumper on J45, connecting the two pins nearest the R127 silkscreen label. The BCM94710AP requires no jumpers.

SDRAM CONFIGURATION

The default BCM47xx SDRAM configuration assumes 16 Mbytes (4 Mbit × 16 bits × 2 units) with an 8-bit address column and 32-bit interface. Table 3 indicates all the possible configurations supported by the BCM47xx.

Table 3: SDRAM Configuration

<i>Config Number</i>	<i>SDRAM UNIT CAPACITY</i>	<i>SRAM WIDTH</i>	<i>SDRAM Unit(s) USED</i>	<i>Total Memory</i>	<i>Makefile Flag</i>	<i>SDRAM_INIT Value</i>
1	1 Mbit	16-bit	1	2 Mbytes	MEM1MX16_KM	0x0009
2	2 Mbit	8-bit	2	4 Mbytes	MEM2MX8X2_KM	0x0809
3	4 Mbit	16-bit	1	8 Mbytes	MEM4MX16_KM	0x0019
4	8 Mbit	8-bit	2	16 Mbytes	MEM8MX8X2_KM	0x0819
5	8 Mbit	16-bit	1	16 Mbytes	MEM8MX16_KM	0x0829
6	1 Mbit	16-bit	2	4 Mbytes	MEM1MX16X2_KM	0x0409
7	2 Mbit	8-bit	4	8 Mbytes	MEM2MX8X4_KM	0x0c09
8	2 Mbit	32-bit	1	8 Mbytes	MEM2MX32_KM	0x0439
9	4 Mbit	16-bit	2	16 Mbytes	MEM4MX16X2_KM	0x0419
10	8 Mbit	8-bit	4	32 Mbytes	MEM8MX8X4_KM	0x0c19
11	4 Mbit	32-bit	1	16 Mbytes	MEM4MX32_KM	0x0429
12	8 Mbit	16-bit	2	32 Mbytes	MEM8MX16X2_KM	0x0c29



12/05/02

Each configuration is numbered (for reference purposes only) and defined by a makefile flag. This makefile flag is used to recompile PMON to ensure that the SDRAM controller is initialized correctly upon bootup. For VxWorks, this flag has to be added to the EXTRA_DEFINE statement in the root makefile. For Linux, add this flag to the CFLAGS statement in the GNU Makefile located in the PMON root directory. The SDRAM_INIT column value is used later. See “SDRAM Initialization Control Register” in the next section.

Example: If there is a 4M x 16 x 2 configuration that describes 2 banks of 4 Mbytes of memory with a 16-bit data path, making it an 8-Mbyte (4M/unit x 2 banks/unit) SDRAM with 16-bit-wide data. So, this is really a 64-Mbit, 1-unit SDRAM with a 16-bit interface. From Table 3, one needs to build the driver with the memory configuration flag of MEM8MX16_KM.

In addition to possible Makefile flag define modifications already listed, there are three registers used to configure how the SDRAM is accessed by the BCM47xx processor:

- SDRAM Initialization Control
- SDRAM Configuration
- SDRAM Refresh Control

The initialization values for these registers (used by PMON on bootup) are found in the following files located in the /bootrom directory: ap_nvdata.txt, dev_nvdata.txt, and 4702_nvdata.txt. Each of the initialization values have an identifier string (in the xx_nvdata.txt files) as follows: sdram_init, sdram_config, and sdram_refresh. Table 4 through Table 6 allow the user to arrive at the correct initialization values depending on the SDRAM used.

SDRAM Initialization Control Register

The two-byte SDRAM Initialization Control register is both readable and writable and contains the value 0x0 at device reset. This register is used to generate the initialization sequence required by the external SDRAM. All the possible values are listed for any given SDRAM configuration in Table 3.

Table 4: SDRAM Initialization Control Register

Field	Name	Description
0	CBRCmd	Writing a 1 to this field causes the SDRAM controller to generate a refresh cycle to the external SDRAM. Writing a 1 to this field toggles the field. Writing a 0 has no effect.
1	PRECmd	Precharge Command. Writing a 1 to this field causes the SDRAM controller to generate a precharge cycle to the external SDRAM. Writing a 1 to this field toggles the field. Writing a 0 has no effect.
2	MRSCmd	Mode Register Select Command. Writing a 1 to this field causes the SDRAM controller to generate a mode register select cycle to the external SDRAM. Writing a 1 to this field toggles the field. Writing a 0 has no effect.
3	SDRAMCtrlEn	SDRAM Control Enable. Writing a 1 to this field enables access to the external SDRAM. Writing a 0 to this field disables access to the external SDRAM.
5:4	BitSize	Bit Size. The value of this field specifies the size of the external SDRAM. Along with the contents of the 32BitMem and 9BitColumn fields, this value is used to select how the memory address is divided into bank select(s), row address, and column address. 11: Special: This value is used for 64 Mb configurations using a 32-bit interface and 8-bit column addresses. If this value is used, the 32BITMEM field must be set to 1 and the 9BITCOLUMN field must be set to 0. 10: 128-Megabit SDRAM 01: 64-Megabit SDRAM 00: 16-Megabit SDRAM
6	Reserved	-

Table 4: SDRAM Initialization Control Register (Cont.)

Field	Name	Description
7	SoftReset	Soft Reset. Writing a 1 to this field causes the registers in the SDRAM controller to be reset.
8	SelfRefresh	Self Refresh. Writing a 1 to this field causes the SDRAM to go into self-refresh mode.
9	PowerDown	Power Down. Writing a 1 to this field causes the SDRAM to go into power down mode.
10	32BitMem	32-Bit Memory. Indicates 32-bit SDRAM interface.
11	9BitColumn	9-Bit Column. Indicates that SDRAM utilizes 9-bit columns.

SDRAM Configuration Register

The two-byte SDRAM Configuration register is both readable and writable and contains the value 0x0 at device reset. This register is used to initialize the mode register of the external SDRAM. The current SDRAM controller only supports CAS latency of 3 and full-page burst.

Table 5: SDRAM Configuration Register

Field	Name	Description
1:0	BurstLen	Burst Length. This value determines the burst length of access to the external SDRAM. 00: full page burst 01: burst of 8 10: burst of 4 11: burst of 2
2	FastMem	Fast Memory. Writing a 1 to this field sets the CAS latency of the external SDRAM to 2. Writing a 0 to this field sets the CAS latency to 3.

SDRAM Refresh Control Register

The two-byte SDRAM Refresh Control register is both readable and writable and contains the value 0x0 at device reset. This register is used to control the refresh rate of the external SDRAM.

Table 6: SDRAM Refresh Control Register

Field	Name	Description
7:0	RefreshPeriod	Refresh Period. This value determines the refresh rate. The frequency of the refresh request can be calculated with the following formula: Refresh rate = 16 x 1/f x refresh period, where f is the clock frequency.
15	RefreshEnable	Refresh Enable. Writing a 1 to this field enables the SDRAM controller to issue refresh commands to the external SDRAM at intervals based on the refresh period.

Example: To see how the initialization values were arrived at for the standard BCM94710AP configuration. The values are obtained from /bootrom/ap_nvdata.txt as follows:

```
s dram_init=0x0419: Initialize for 32-bit SDRAM interface, 64 Megabit SDRAM, enable access to external SDRAM, and allow SDRAM controller to generate refresh cycle.
s dram_config=0x0000: Full page burst, CAS latency of 3.
s dram_refresh=0x8040: Enable SDRAM controller to issue refresh commands, set refresh period to 64.
```



FLASH CONFIGURATION

Flash with Linux

Both the BCM94710AP and BCM94710D platforms are configured with 4 megabytes (MB) of Flash memory. To allow boards with only 2 megabytes of Flash, the following procedure must be followed for Linux images only³:

- 1 Change the "trx" line in /src/Makefile from "-b 768k" to "-b 640k". The new line should read:
"trx -o \$@ \$(VMLINUZ) -b 640k \$(RELEASEDIR)/image/target/cramfs."
- 2 Remove the file /target/usr/sbin/ppoeed.
- 3 Remove the file /target/usr/sbin/upnp.
- 4 Remove the file /target/lib/modules/2.4.5/kernel/drivers/il/il.o.
- 5 Change the line "CONFIG_IL=m" to "CONFIG_IL=n" in /src/linux/linux/.config.
- 6 Go to the /src/linux/linux directory and enter "make dep."
- 7 Go to the /src directory and enter "make."
- 8 Reprogram the Flash with the new PMON. See "Programming PMON Into Flash" on page 35.
- 9 Reset the platform, and interrupt the booting process to enter the following:
PMON> set kernel_args "root=/dev/mtdblock2 noinitrd console=ttyS0,115200 flash=2M"
PMON> set nvram
- 10 Reboot the platform. Load the new image into Flash or boot over the ethernet connection.

To verify the correct configuration changes for the 2-MB Flash device, the following sequence should be printed out during bootup:

```
Creating 4 MTD partitions on "Physically mapped flash":
0x00000000-0x00040000: "pmon"
0x00040000-0x001e0000: "linux"
0x000e0000-0x001e0000: "rootfs"
0x001e0000-0x00200000: "nvram" flash device: 200000 at 1fc00000
```

Tested Flash Devices

The following flash devices have been tested with the BCM94710AP platform:

- Intel® - 28F320JA
- AMD - AM29LV320DT/B
- Toshiba - TC58FV/T321, TC58FVB321
- ST® Micro - M29W320DT/B
- Fujitsu® - MBM29LV320BE, MBM29LV320TE

3. Note that the Residential Gateway package as currently shipped does not fit in 2 megabytes of Flash. This configuration is only recommended for APs and not routers. Much of the router functionality in the Gateway package must be stripped off (that is, pppOE, iLine, and WAN support, and so on) to fit the image onto the 2 MB of Flash.

UART REMOVAL

This section covers a hardware platform requiring the exclusion of the UART 16550 chip. One hardware change is required: the GPIO1 pin (on the 47xx chip) has to be tied to ground through a 1-k Ω resistor. When using a Linux operating system, no software changes should be necessary.

However, VxWorks users have to implement the following code modification to `/target/config/bcm47xx/bsp/pciConfigLib.c` (code to be added is in bold type):

```
void platform_init()
{
char *board, *cf;
uint32 pci_clock, req_sb, req_pci, cpu_clock;
char *wd;

/* if there is no uart then clear the gpointmask */
if (NUM_TTY == 0)
W_REG(&extif->gpointmask, 0);

/* Read the watchdog variable (set in ms) */
.....
.....
}

```

Once the VxWorks code is modified, rebuild the image and try booting without the UART present.

RELEASE PACKAGES

Depending on the nature of the project and the type of engagement with Broadcom, the release package can have different parameters including:

- Linux or VxWorks code base
- Support for bridging only or full residential gateway support
- A binary-only package or a combination of source and object files to provide for customer differentiation

Because the release packages can vary, some sections of the resource guide may not be applicable to a particular project.

RESIDENTIAL GATEWAY FEATURES

The residential gateway packages for both Linux and VxWorks are intended to be turnkey solutions for home use. They provide a comprehensive list of features.

General Residential Gateway Features

- Configuration of WAN interface
 - Static configuration
 - DHCP client
 - PPPoE
- Multiple LAN interfaces
 - Ethernet
 - HPNA 2.0
 - 802.11b
 - USB (Linux only)
- DHCP server on LAN interfaces
- NAT/PAT
- WAN DNS
- WAN WINS
- Application Layer Gateway (ALG) support for popular Internet applications
 - NetMeeting
 - FTP
 - HTTP
- UPNP IGD 1.0
- HTTP server hosting Web-based configuration pages
- Multiple firmware upgrade mechanisms
 - Remote firmware upgrade over WAN
 - Local firmware upgrade using HTTP POST method
 - Local firmware upgrade using TFTP at boot time
- IPsec and PPTP pass through
- PPPoE client
 - PPPoE MTU/MRU (Linux)
 - PPPoE connect on demand (Linux only)
 - PPPoE keep alive (Linux)
- Support for add-on adapters
 - PCI/Mini PCI
 - PCMCIA
- Integrated SW bridge for all LAN interfaces
- SW filtering of packets based on:
 - LAN MAC address
 - IP filtering
 - TCP/UDP port filtering
- Configuration of static routes

- LAN port forwarding and DMZ
- Connection event logging with Network Time Protocol (NTP) support (Linux only)
- Restore to default factory configuration
- WAN statistics
- WAN blocking
- Remote login
- WAN MAC address cloning

802.11 Features

- Full configuration of interface
 - SSID
 - Channel
 - Open/Closed networks
 - Rate
 - Country code
- Configuration of WEP
 - WEP 64
 - WEP 128
- Shared key authentication
- 802.11 MIB counters
- MAC address filtering
- DS magic packet
- AP-only option
- WECA parameters selection

BCM947XX LINUX PACKAGES

The Linux package can be either a bridge or a residential gateway implementation.

The bridge package includes the following features:

- Packet bridging between all LAN interfaces (Ethernet, HPNA and 802.11)
- Command shell accessible through the serial UART
- Basic UNIX command-line utilities
- iptables package to provide basic routing services
- Support for BCM430x-based mini-PCI wireless 802.11b wireless adapters

In addition to the bridge features, the residential gateway package includes:

- Dynamic and static IP address support
- PPPoE connections on the WAN port
- Built-in firewall to protect local PCs from outside intrusion
 - Stateful Packet Inspection (SPI) and IP masquerading
- Configuration through any networked PC's Web browser
 - Remote upgrade and remote management



- DHCP server (for LAN PCs) or DHCP client (on the WAN port)
- Standard Internet applications compatibility
- Blocking of specific LAN users Internet access
- IP packet filtering
- TCP/UDP port forwarding and DMZ capability

WHAT YOU GET

Both the bridge and residential gateway packages include:

- General readme file that describes the tools used in building the package, how to install the tools, and so on
- Source to the Linux kernel in the /src/linux/linux directory
- Source for all of the Broadcom-supplied components in the root directory:
 - /src/et - contains source for the Ethernet driver (binary only for residential gateway package)
 - /src/ll - contains source for the HPNA 2.0 driver (binary only for residential gateway package)
 - /src/shared - contains source code shared among many of the Broadcom-supplied components
 - /src/include - contains header files for many of the Broadcom-supplied components
 - /src/router - contains the remaining bridge/router modules
- /image - contains pre-compiled binaries for the package. The image is a zipped flat binary image with the appropriate TRX header attached. The TRX header allows you to use the PMON boot loader to upload the image to your BCM47xx-based design. If you are not using the PMON boot loader, you can build a new image and use one of the intermediate file formats (ELF, self-booting, flat binary, and so on) for your purposes.
- /doc - contains this document as well as other supporting documentation
- /tools - contains things necessary to build the Linux image, including nvserial, trx, and so on.
- /target - root file system of the router
- /bootrom - contains source and a binary file for the PMON boot loader

The residential gateway package also includes source code for the services necessary to implement a full residential gateway application.

Note

Depending on the package, some or all of the source directories can be omitted and replaced with corresponding binary files.

ADDITIONAL TOOLS

A separate tools package contains the compilers, libraries, and other tools necessary to build the Linux image for the BCM4710. The compiler and libraries are based on gcc 3.0 and use the glibc version 2.2.3. This tools package is available on your customer page at <http://hnsupport.broadcom.com>. Refer to the README.TXT file in the root directory of the installation package for information on installing and configuring these tools.

LINUX KERNEL

The Linux image is based on freely available Linux kernel source code. The specific version of the Linux kernel used is 2.45.

Because of memory and Flash constraints, certain kernel services are not compiled when the BCM94710 Linux image is created. These services are not necessary to provide the desired residential gateway functionality. To add extra residential gateway functionality, it may be necessary to modify the kernel build to include some necessary services. To do this, perform the following:

- 1 Create a separate kernel .config file. This is normally done by executing the make config command in the /src/linux/linux directory.
- 2 Rename the resulting .config file to something like myconfig.
- 3 Copy the myconfig file to the /src/linux/linux/arch/mips directory.
- 4 Copy /src/Makefile to /src/Makefile.bak.
- 5 Open /src/Makefile with in a text editor.
- 6 Change all instances of the defconfig-bcm947xx string to myconfig.

Now, the build process for the residential gateway (see below) uses the custom config file instead of the default Broadcom config file.

BCM430X 802.11B WIRELESS SUPPORT

BSP Package

In the BSP release, the 802.11b interface is configured as eth4. The iwconfig suite of tools are provided to allow for configuration of various 802.11b parameters such as SSID, channel number, and data rate. Check the main pages on the Linux build machine or consult online Web pages for information on the usage of iwconfig.

Additional configuration can be done using the iwpriv function. Iwpriv, like iwconfig, is a community-developed configuration tool for wireless adapters. Iwpriv allows for vendor-specific configuration of wireless parameters. Below is a list of supported iwpriv options:

- **hide:** set to 1 for closed network or 0 for an open network. When a network is closed, it is hidden from active scans.
- **hidden:** returns 1 if network is closed, 0 if open
- **setauth:** set to 1 for shared key authentication or 0 for open authentication
- **getauth:** returns 1 if in shared key authentication mode or 0 if in open authentication mode
- **setbcn:** sets the beacon interval (in milliseconds)
- **getbcn:** returns the beacon interval (in milliseconds)
- **setdtim:** sets the DTIM period (in milliseconds)
- **getdtim:** returns the DTIM period (in milliseconds)

In addition to iwconfig and iwpriv, configuration of the 802.11b interface can be done through a combination settings of NVRAM variables and the wlconfig tool. The command nvramp can be used to set a NVRAM variable, get the value of an NVRAM variable, or show the value of all NVRAM variables. Enter the command nvramp at the console prompt for usage information. See Table 23: "NVRAM Environment Variables—802.11b Parameters," on page 52 for information on the 802.11b-specific NVRAM parameters.

The wlconfig tool reads all of the 802.11b-specific NVRAM variables and then program the 802.11b interface accordingly. Reference the file target/etc/init.d/rcS for examples of how to use the nvramp and wlconfig commands.

Residential Gateway Package

The residential gateway package provides for full configuration of the wireless interface through the provided http server. Specifically, the `src/router/www/broadcom/wireless.asp` file contains the http code that sets/gets the appropriate nvram variables that control the wireless configuration.

ADDING ADDITIONAL SERVICES

BSP Package

The `/etc/init.d/rcS` target file is parsed to start services when the bridge package kernel is booted. If you want to start additional services when the kernel boots, you must add commands to this startup script to invoke them.

Residential Gateway Package

The starting/stopping of services in the residential gateway package is controlled by the `/src/router/rc/rc.c` file. If there are additional entry points for services that you want to start or stop on the residential gateway, you must add them to this file. Specifically, calls to stop your services should be added to the STOP case in the main `loop()` function of `src/router/rc/rc.c`. Calls to start your services should be added to the START case.

Incorporating Binaries into the File System

Include the binaries for services in the file system. You can integrate the compiling of your services into the main residential gateway build by editing the makefile that is located in the `/src` directory of the distribution.

As the image is being built, the file system that will support the Linux kernel is created in the `src/router/mipsel-install` directory. The binaries and object files that you want to include in the file system must be copied to the appropriate place in the `/src/router/mipsel-install` directory structure.

BUILDING THE LINUX IMAGE

The top-level of the distribution contains a makefile. After all of the tools have been properly installed, you can simply type `make` in the `/src` directory, and the PMON boot loader and the Linux image are created. The PMON boot loader binary is located in the `/bootrom` directory and the Linux binary is located in the `/bin` directory.

The Linux kernel is called `linux.trx`. It contains both the Linux kernel and the root file system, and already contains the TRX header necessary for use with the PMON TFTP function.

The PMON binary does not have the necessary EST header necessary for programming with the visionICE because it must first have a default NVRAM image embedded in it before the EST header can be added. See "PMON Boot Loader" on page 33 for information about embedding NVRAM information in a PMON image. See "Using visionCLICK/visionICE" on page 39 for details for adding EST headers so that the PMON image can be burned to Flash using visionICE.

PRUNING THE C LIBRARIES

The standard C library, `glibc`, is included with both the BSP and the residential gateway source-code distribution. For the BSP, a complete `glibc` with all of the modules linked is included. This is done because it is unknown what additional services will be added to the final product BSP release. Leaving `glibc` whole ensures that the drivers and/or applications will have the necessary library support.

However, it is often the case that the final product only uses a small subset of the glibc functionality. In this case, you may want to remove some of the extra object files of glibc to save space. The Linux release will include this functionality.

The final step in the building of either the Linux BSP or the Linux residential gateway is a pruning of the glibc library. This is done by a script called `generate_library`. To ensure that the final glibc is optimally pruned, any additional applications or libraries that need to be included with the Linux build should be present in the output directory when the final step of the makefile is invoked. At this time, the `generate_library` script will be invoked and glibc will be pruned.

STATISTICS REPORTING

The BCM4710 device keeps a variety of statistics on all of the various interfaces. The residential gateway image can be configured to periodically take these statistics, as well as other operating parameters of the residential gateway, and upload them to a statistics server. Here, information for a number of residential gateways deployed externally can be coalesced for analysis.

The server where the statistics are stored is specified by the `stats_server` NVRAM variable. If this variable is unset, then stats collection does not occur. See "NVRAM" on page 47 for information on setting/unsetting NVRAM variables.

As an example, included in the router release is a Perl script, `src/router/misc/stats.pl`. Refer to this script as an example of how to gather these statistics.

PCI DEVICE SUPPORT

Tulip Example Driver

The BCM94710D platform has two PCI slots in which PCI devices can be inserted. Included in the Linux BSP release is a *tulip* Ethernet driver. The tulip driver is automatically compiled as a module and supports a number of Ethernet adapters including the Netgear FA310TX.

To invoke the driver simply type the command

```
insmod tulip
```

at the command prompt.

PCMCIA DEVICE SUPPORT

Intersil® Example Driver

The BCM94710D platform has a single PCMCIA slot in which a PCMCIA device can be inserted. Included in the Linux BSP release is a driver for an Intersil-based 802.11b network adapter. This driver has been tested with Lucent® Orinoco 802.11b adapters.

The `/etc/init.d/rcS` target file controls the starting of services in the Linux BSP build. If you would like to start the driver for the Intersil adapter, locate the following lines near the end of the rcS file and uncomment them:

```
# This is how you would add an Orinoco PCMCIA 802.11b card to the bridge
# /sbin/cardmgr
# brctl addif br0 eth3
# ifconfig eth3 0.0.0.0
```

In the /sbin directory on the Linux BSP file system, utilities such as dump_cis and pack_cis are included. These can be used in conjunction with the Intersil driver or used in development of additional PCMCIA drivers.

THIRD-PARTY DEVICE DRIVERS

To use a third-party device driver, four things need to be done:

- The driver needs to be added to the build
- When compiled, the -m4710a0kern option must be used
- The driver needs to be copied to the target 4710 file system
- The driver needs to be invoked

Building the Driver

The recommend way for compiling third-party device drivers outside of the kernel tree is to go into the kernel tree and type *make modules*. This will descend into each subdirectory, printing out status in the form:

```
make -C drivers/net CFLAGS="..."
```

Ensure that the CFLAGS have at least the following defined. (There may be more defines depending on the module being built.)

```
-DMODULE -D__KERNEL__ -DLINUX -G 0 -mno-abicalls -fno-pic -mlong-calls -fno-common
```

Also, add:

```
-I<path to Linux>/include
```

so that the compiler can find the necessary kernel headers.

Copying the Driver to the File System

For the Linux kernel running on the BCM4710 platform to find the driver, it must be copied to the /target/lib/modules directory before the file system is compressed.

Invoking The Driver

For a Broadcom-supplied router package, add the module name (minus the .o extension) to the NVRAM variable kernel_mods. If no kernel_mods NVRAM variable is specified, the modules et and il will be assumed. So to add a module called foo, you would set the kernel_mods NVRAM variable to et il foo. The module is then loaded during system initialization.

For a Broadcom-supplied BSP package, the procedure is somewhat different. Add the line

```
insmod foo
```

to the /target/etc/init.d/rcS script. This file is installed on the target Linux system as /etc/init.d/rcS and is parsed at system startup.



DEBUGGING LINUX-BASED CODE

The Linux BSP and Residential Gateway images support the use of the gdb debugger to debug kernel code. To use gdb for debugging kernel components, the following must be true:

- You must have a PC running Red Hat® Linux (6.2 - 7.1)⁴ that acts as a build machine and runs the gdb debugger. You also need to run a remote terminal using a terminal program, such as HyperTerminal for Windows or minicom for Linux. If your main build PC has two serial ports, it can be used to run both the gdb application and the terminal application. If it does not have two serial ports, use a second machine to run the terminal software. See “Connecting a Serial Monitor to a BCM47xx-based Platform” on page 36 for information on configuring the remote terminal.
- Install the build tools on your Linux PC. Follow the README.TXT that came with the tools release for information on how to install the build tools.
- The Flash must contain a linux.trx image that boots and functions correctly. Follow the instructions in the section titled “Loading an OS Image Using PMON” on page 37 to load a working OS image if one is not already present on the platform.
- The platform must have two serial ports: one to be used for a remote console and one for the serial gdb debugging. The Broadcom BCM94710D reference platform meets these requirements.
- The kernel must be recompiled to support gdb debugging.

To recompile the Linux kernel to support gdb debugging:

- 1 Change to the src/linux/linux directory in the distribution.
- 2 Enter the command make menuconfig. This will bring up the kernel configuration menu.
- 3 Scroll to Kernel Hacking section and press **Enter**. In the subsequent screen, make sure that only the **Are you using a crosscompiler, Remote GDB kernel debugging, and Magic SysRq key** options are set.
- 4 Exit and save from the menuconfig screens.
- 5 In the src/linux/linux directory, enter the command make zimage. This will create two separate files: src/linux/linux/vmlinux and src/linux/linux/arch/mips/bcm47xx/brcm-boards/compressed/piggy. The vmlinux file is an uncompressed kernel image with full debug information. This file is used by the mipsel-linux-gdb tool to obtain debug information, but it is never transferred to the BCM47xx-based platform. The piggy file is a compressed image without debug information that is transferred to the BCM47xx-based platform. The debugger is able to correlate execution addresses in the piggy file with source code information embedded in the vmlinux file. This allows for source-level remote debugging without eating up excessive RAM on the target platform.

To transfer the piggy image to the BCM47xx-based platform:

- 1 Ensure that the PC with the piggy image has its Ethernet interface statically configured to the IP address 192.168.1.100 with a netmask of 255.255.255.0
- 2 Ensure that the Ethernet interface on the PC is connected to the LAN interface on the BCM47xx-based platform.
- 3 Ensure that a terminal program (such as HyperTerminal) is running on the PC connected to the appropriate serial port on the BCM47xx-based platform. See “Connecting a Serial Monitor to a BCM47xx-based Platform” on page 36 for more information.
- 4 Reset the BCM47xx-based platform and press **Ctrl+C** in the terminal program to get to a PMON> prompt.
- 5 Enter the set command to get a list of all of the current NVRAM variables. Take note of the dl_ram_addr variable value. This is the location in memory where an image received via TFTP will be stored. This address is used later in the procedure.
- 6 Enter the command load -B. This tells PMON to accept an image via TFTP and store it in RAM.

4. More recent Red Hat Linux distributions (for example, 7.2, 7.3, and 8.0) have been used for compiling/linking/debugging. However, the most recent reference platform is still Red Hat 7.1.

- 7 On the PC, change to the `src/linux/linux/arch/mips/bcm47xx/brcm-boards/compressed` directory and use the TFTP program to send the new piggy image to the platform. See "Using the TFTP Application" on page 34 for more information.
- 8 At the `PMON>` prompt, enter the command `call <dl_ram_address>` where `<dl_ram_address>` is the value of that NVRAM variable. This will cause the kernel to boot. Note that there is not any output in the remote terminal window until gdb has attached to the kernel.

Connect the Linux build machine to the lower serial port on the BCM94710D reference platform with a standard 9-pin serial cable. In the main kernel directory (`src/linux/linux`) on the Linux build machine, enter the command `mipsel-linux-gdb -nw -b 115200 vmlinux`. At the gdb prompt, type `target remote /dev/ttyS0`. (`ttyS0` corresponds to COM1: on most systems. If you are using a different port for the gdb connection, the tty device should be changed accordingly. For instance, if you were using COM2 then the device would be `ttyS1`.)

After entering the command `target remote /dev/ttyS0`, there is a warning about shared library handles. Ignore this warning. You should now be at a gdb prompt.

From here, use any of the gdb facilities to set breakpoints, view memory, and so on. You can use the `continue` command to get the kernel running. If you need to enter the debugger again, send the `break` command followed by `D` in the terminal application. (The method for issuing the `break` command is different for each terminal application. In `minicom`, it is **Ctrl+A, F**. In `Teraterm` it is **Alt+B**. Refer to the serial terminal documentation for more information)

For more information on using gdb, consult the gdb man pages.

USER-LEVEL APPLICATIONS

In addition to the kernel, certain Linux user-level applications are provided. The following sections give some details about these applications.

brctl

(Included in both the bridge and residential gateway packages)

From the `brctl` man page:

The `brctl` is used to set up, maintain, and inspect the ethernet bridge configuration in the linux kernel.

An Ethernet bridge is a device commonly used to connect different networks of Ethernet together, so that these Ethernets appear as one Ethernet to the participants.

Each of the Ethernets being connected corresponds to one physical interface in the bridge. These individual Ethernets are bundled into one bigger (logical) Ethernet, this bigger Ethernet corresponds to the bridge network interface.

Usage:

To bridge together `eth0` and `eth1` into a new logical interface `br0`, do:

```
brctl addbr br0
brctl addif br0 eth0
brctl addif br0 eth1
```

See the `brctl` man page for a more detailed list of options.

Package: busybox
Version: 0.60.0
Maintainer: Lineo
License: GPL
Homepage: <http://busybox.lineo.com>
Description:

BusyBox

(Included in both the bridge and residential gateway packages)

From the BusyBox Web site at www.busybox.net:

BusyBox combines tiny versions of many common UNIX utilities into a single small executable. It provides minimalist replacements for most of the utilities 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.

BusyBox has been written with size-optimization and limited resources in mind. It is also extremely modular so you can easily include or exclude commands (or features) at compile time. This makes it easy to customize your embedded systems. To create a working system, just add /dev, /etc, and a kernel.

Usage:

Edit busybox/Config.h to suit your requirements. The BusyBox binary and the symlinks it generates, as well as a minimal /dev directory, should be all that is required to get Linux to a shell prompt.

httpd

(Included in the residential gateway package)

Version: 1.0
Maintainer: Broadcom Corporation
License:
Homepage:
Description:

A modified combination of Jef Poskanzer's micro_httpd and mini_httpd, httpd is a minimal Web server daemon that calls hooks to handle various types of HTTP/1.0 GET requests. The basic build defines a NULL-terminated list of MIME handlers of the form:

```
{ <pattern>, <mime_type>, <mime_handler>, <auth_handler> }
```

See basic.c for a default list that handles most common requests. More complicated parsing of dynamic HTML files can also be implemented. The basic build provides do_file() as an example of handling static HTML files.

httpd has been expanded to support a limited subset of the GoAhead[®] Web server (<http://www.goahead.com>) functionality. This allows Web pages to call predefined Embedded Javascript (ej) routines and allows support routines to be written to function with the GoAhead Web server, but still link against httpd. This section of httpd.h deals with GoAhead compatibility:

```
/* GoAhead 2.1 compatibility */
typedef FILE * webs_t;
typedef char char_t;
#define T(s) (s)
#define websWrite(wp, fmt, args...) ({ int ret = fprintf(wp, fmt, ##
args); fflush(wp); ret; })
#define websError(wp, code, msg, args...) fprintf(wp, msg, ## args)
#define websHeader(wp) fputs("<html>", wp)
#define websFooter(wp) fputs("</html>", wp)
#define websDone(wp, code) fflush(wp)
#define websGetVar(wp, var, default) (get_cgi(var)?: default)
extern int ejArgs(int argc, char_t **argv, char_t *fmt,...);
```

Any GoAhead functionality not listed above is not supported by httpd. For information on Embedded Javascript, refer to the GoAhead Web site.

The source code to support the calls made by the Embedded Javascript can be found in the `src/router/shared/broadcom.c` file. To run httpd, start it in the root directory of the HTML files.

iptables

(Included in both the bridge and residential gateway packages)

Version: 1.2.2

Maintainer: Netfilter Project

License: GPL

Homepage: <http://www.netfilter.org>

Description:

The iptables is the user interface to the Linux 2.4 packet mangling infrastructure (netfilter). Netfilter encompasses NAT (including IP masquerading), packet filtering (firewalling), and generalized packet mangling.

Usage:

For embedded needs, it is strongly recommended that netconf be used instead. netconf provides a C API to the kernel networking stack with a much simpler and reduced syntax.

Otherwise, see the iptables man page for detailed information on usage.

netconf

(Included in the residential gateway package)

Version:

Maintainer: Broadcom Corporation

License:

Homepage:

Description:

The netconf is a C API to the Linux 2.4 kernel networking stack. netconf provides hooks for accessing Ethernet interfaces, IP routes, NAT, and packet filter tables.

Usage:

See the netconf header file netconf.h for API definitions and the rc package for usage examples. The netconf binary provides sample ifconfig, route, and fw commands implemented using the netconf API. Type ifconfig -h and route -h and fw for usage information.

Examples:**From the command line:**

```
ifconfig br0 192.168.1.1 netmask 255.255.255.0 broadcast 192.168.1.255
route add net 192.168.1.0 netmask 255.255.255.0 br0
fw
```

From C:

```
netconf_if_t interface;

memset(&interface, 0, sizeof(interface));
interface.flags = IFF_UP | IFF_RUNNING | IFF_BROADCAST;
strncpy(interface.name, "br0", sizeof(interface.name));
inet_aton("192.168.1.1", &interface.ipaddr);
inet_aton("255.255.255.0", &interface.netmask);
inet_aton("192.168.1.255", &interface.broadcast);
netconf_set_interface(&interface);
```

The following is a list of netconf API calls with short descriptions.

```
netconf_get_interface() - get parameters for an interface or all
interfaces
netconf_set_interface() - set parameters for an interface or list of
interfaces
```

```
where parameters are things like;
struct in_addr ipaddr; /* IP address */
struct in_addr netmask; /* IP network mask */
struct in_addr broadcast; /* IP broadcast address */
struct in_addr netaddr; /* IP network address */
struct ether_addr mac; /* MAC address */
```

```
netconf_get_routes() - get a list of the routes bound to an interface or list of interfaces
netconf_add_routes() - add a route or list of routes
netconf_del_routes() - delete a route or list of routes
netconf_get_fw() - get a list of the current firewall entries
netconf_reset_fw() - reset the firewall
netconf_reset_fw() - resets the firewall to a sane state.
netconf_add_nat() - add a NAT entry or list of NAT entries
netconf_del_nat() - delete a NAT entry or list of NAT entries
netconf_get_nat() - get a list of the current NAT entries
netconf_add_filter() - add a filter entry or list of filter entries
netconf_del_filter() - delete a filter entry or list of filter entries
```

nvram

(Included in the residential gateway package)

Version:

Maintainer: Broadcom Corporation

License:

Homepage:

Description:

The nvr

am package is a C API to the NVRAM portion of flash memory. It provides persistent storage for ASCII text in the form of name=value pairs. See the nvram header file `bcmnvr`am.h for API definitions and the rc package for usage examples. The nvram binary provides access to the NVRAM API from the command line and supports `get`, `set`, `unset`, and `show` commands.

Examples:

From the command line:

```
nvr
```

am set foo=bar
nvram get foo
nvram show
nvram unset foo

From C:

```
nvr
```

am_set("foo", "bar");
nvram_get("foo");
nvram_getall(buf, sizeof(buf));
nvram_unset("foo");
nvram_init(void);**rc**

(Included in the residential gateway package)

Version:

Maintainer: Broadcom Corporation

License:

Homepage:

Description

The rc program is the master control daemon for the reference design. It runs as process 1 (*init*) and is responsible for starting and stopping all processes. To add a new process to the system initialization and shutdown sequences, edit `main_loop()` in `rc.c`. The state of rc can be controlled from the command line using the rc utility or by sending signals to process 1 from C. For example:

From the command line:

```
rc start  
rc stop  
rc restart
```

From C:

```
kill(1, SIGUSR1);
kill(1, SIGUSR2);
kill(1, SIGHUP);
```

udhcp

(Included in the residential gateway package)

Version: 0.9.5

Maintainer: Lineo

License: GPL

Homepage: <http://opensource.lineo.com>

Description:

The udhcp package provides minimal DHCP server and client functionality in a (optionally) single binary. Both are highly configurable through the use of command-line options and configuration files. See the README for more information.

Usage:

A sample udhpc script is located in /samples/sample.script. Install the script and point the client at it before starting the client. A sample udhcpd configuration file is located in /samples/udhcpd.conf. Install the configuration file in the location defined in dhcpcd.h before starting the server.

Example:

From the command line:

```
udhpc -i eth1 -s /etc/sample.script
udhcpd
```

ADDITIONAL INFORMATION

Below are links to Web sites that contain additional information about the tools used in the Linux builds.

GNU Compiler Suite

- <http://gcc.gnu.org/onlinedocs/gcc-3.0/gcc.html>
- <http://www.gnu.org/manual/gas-2.9.1/as.html>
- <http://www.gnu.org/manual/binutils-2.10.1/binutils.html>

Linux Reference Books

- Linux Application Development
 - http://www.amazon.com/exec/obidos/ASIN/0201308215/qid=1005950723/sr=8-3/ref=sr_8_7_3/002-0510203-9712032
- Linux Device Drivers 2nd Edition
 - http://www.amazon.com/exec/obidos/ASIN/0596000081/qid=1005950745/sr=1-1/ref=sr_1_14_1/002-0510203-9712032
- Understanding the LINUX Kernel: From I/O Ports to Process Management
 - http://www.amazon.com/exec/obidos/ASIN/0596000022/ref=pd_sim_books/103-8405895-4051809

BCM47xx VxWORKS PACKAGES

There are two VxWorks support packages for the BCM47xx: a BSP that only performs a bridging function and a BSP that implements a residential gateway application. In either case, the package provides device drivers for all of the hardware peripherals that are supported by the BCM47xx. Additionally, each package contains support for Broadcom BCM430x-based 802.11b wireless solutions. This allows these packages to act as either 802.11b access points or wireless routers.

The BSP bridge package provides source and libraries that support the BCM47xx silicon in a bridging application. This package is also the recommended starting place to integrate BCM47xx support into the VxWorks-based application.

The BSP residential gateway package adds the following:

- Broadcom-provided libraries for the HTTP interface used for residential gateway configuration
- DHCP server and client libraries
- Network Address Translation (NAT) libraries
- Libraries to provide bridging functionality on the LAN interfaces

The residential gateway support relies on the Tornado[®] Home Gateway (THG) v1.0 packages from WindRiver Systems[®]. Broadcom has made additions to the stock THG code to provide additional functionality.

SOFTWARE PACKAGING RECEIVED

For both the BSP bridge and BSP residential gateway package, you receive:

- A general readme file, VxReadme.txt, that describes the contents of the package in the /bcm47xx directory
- Source for all of the Broadcom-supplied components in the /bcm47xx directory:
 - /bcm47xx/bsp/et- contains source for the Ethernet driver (binary only for residential gateway package)
 - /bcm47xx/bsp/il - contains source for the HPNA 2.0 driver (binary only for residential gateway package)
 - /bcm47xx/bsp/shared - contains source code shared among many of the Broadcom-supplied components
 - /bcm47xx/bsp/include - contains header files for many of the Broadcom-supplied components
 - /bcm47xx/bsp/bridge - contains the remaining bridge modules for the BSP package
- /bin - contains pre-compiled binaries for either the BSP package or the residential gateway package. The resulting image, vxbsp.trx, is ready to be uploaded to the BCM47xx-based platform using TFTP and PMON. See "PMON Boot Loader" on page 33 for more details.
- /doc - contains this document
- /tools - contains tools necessary to prepare the images for use on a BCM47xx-based platform. The tools are SWAP.EXE, ADDHDR.EXE, NVSERIAL.EXE and TRX.EXE. Use of these tools is covered later in this guide.
- /tools/visionice - contains additional files necessary to properly use a visionICE debugger with the BCM47xx
- /bootrom - contains a PMON boot loader binary file as well as supporting files necessary for customization

The following files should be noted:

- /bcm47xx/vxbsp.h is a header file for target board. All board-specific information is defined here. This includes the definition of serial interface, timer, the I/O device addresses, the interrupt vector/level, the meaning of device register bits, the clock definitions, and so on.
- /bcm47xx/sysLib.c file contains target specific, system-dependent sysLib routines. These routines provide board-level interface on which VxWorks and applications can be built in a system-independent way.
- /bcm47xx/sysALib.s file is assembler source of target-specific, system-dependent routines. All BSP specific assembly code should be placed here.

- /bcm47xx/config.h file contains definitions specific to the BCM47xx CPU board.
- /bcm47xx/romInit.s file is assembly source for initializing code that is the entry point for VxWorks boot ROM-based versions of VxWorks.

REQUIRED TOOLS

The BSP residential gateway package requires the prior installation of Tornado 2.1 tools from WindRiver. The BSP bridge requires either Tornado 2.0 or Tornado 2.1 to be installed. Refer to the documentation provided by WindRiver on how to install the Tornado environment.

In addition, a license for the THG package from WindRiver is required to provide residential gateway support in your product. Refer to the WindRiver Web site at <http://www.windriver.com> for more information.

If you are using the Tornado 2.0 environment for development, the bcm47xx/bsp/Makefile needs to be modified. The line that reads:

```
CPU = MIPS32
```

must be changed to read:

```
CPU = RC32364
```

Also, the RC32364BSP that supports the MIPS32™ processor family and the Tornado 2.0 Cumulative Patch 4 must be installed in addition to the base Tornado 2.0 installation. Contact WindRiver to obtain the necessary tools.

INSTALLING THE PACKAGE SOURCE

The package source should be installed into the existing Tornado 2.1 installation. To do this, copy the entire /bcm47xx directory in the package distribution to the <Tornado>/target/config directory of the Tornado 2.1 installation.

PREPARATION CHECKLIST

The following configuration checklist needs to be covered to have a successful VxWorks BSP port for the BCM47xx targets.

UART and SIO

The default is set to 1 for a serial device. You can change this to 0 if no support is needed or > 1 if more than one serial devices are there. The number of serial and UART channels (N_SIO_CHANNELS and N_UART_CHANNELS) must also be appropriately changed in *vxbsp.h*. This BSP assumes that an external NS16550 UART is available at address 0xbf800000, with the clock set to 13.5 MHz. The UART TTY driver is included in the BSP (ns16550SioBE.c). If a different UART is used, you must provide the appropriate device driver.

DRAM

By default, the BSP is built for BCM47xx-based system with 4 MB of SDRAM. The amount of DRAM that the BSP expects is defined by the LOCAL_MEM_SIZE variable in the *config.h* file. For 4 MB, the value is set to 0x00400000. This variable must match the amount of DRAM in the target system. For example, if the target system uses 8 MB of SDRAM, set the LOCAL_MEM_SIZE variable to 0x00800000.

The BSP queries the NVRAM to determine the appropriate initialization values for the SDRAM initialization registers. The appropriate values must be present in the NVRAM before the VxWorks target is executed. See “PMON Boot Loader” on page 33 for details on how to embed the appropriate NVRAM data into the PMON image or for information on how to create a NVRAM image to be programmed via the visionICE debugger.

NVRAM

A valid NVRAM must be present for the VxWorks image to function properly. If the PMON bootloader is used in the design, it places its default NVRAM image into the Flash when it first boots. If PMON is not used, a default NVRAM image can be made with the NVSERIAL.EXE tool and programmed into Flash with the visionICE. See “PMON Boot Loader” on page 33 for more information on the use of NVSERIAL.EXE.

Interrupts

Under VxWorks interrupts are assigned a unique ID typically ranging from 0-255. The ISR is then bound by creating the interrupt vector and then calling the *intConnect* routine. This BSP assigns a range of interrupt numbers for MIPS interrupt 0 and for the external i/f interrupt. These interrupt lines call a handler sharedInt0() that determines which devices are currently enabled on interrupt 0 and calls the appropriate handler for each device.

The interrupt map is programmable on a BCM47xx device. There can be multiple sources of interrupt for mips0, which is defined by the SBINTVEC register. The interrupts mapped to MIPS interrupts 1-4 are also programmable. The interrupt mappings can be altered by changing the setting of the following two defines from vxbsp.h. Avoid this whenever possible.

```
#define DEF_SBINTVEC 0x20
#define DEF_SBIPFLAG 0x6020104
```

Note



VxWorks has a per-task CP0 default status register, which requires turning on any interrupt (at the MIPS level) that the system uses. Ensure that no device is generating an interrupt before the default status register value is set. Do this in sysHwlnit().

The iLine and two Ethernet cores each have a timer function. If one of these cores is not being used then the aux timer function can be associated with it through a call to `set_auxtimer(uint32 base_addr_of_core)`. By default there is no aux clock support in this BSP.

INTEGRATING WITH THG

Circumventing the NAT

All packets that go through the VxWorks THG module go through the network address translation (NAT) code. This is true not only for packets that are to be routed over the WAN interface but also for packets that are to be bridged to another local LAN interface. This may not be desirable for additional services that you can add for your product.

The function:

```
extern void natPassThruListAdd(u_long ipaddr, u_long netmask);
```

which can be used to instruct the THG code to **not** route packets with a particular IP address through the NAT code. An example would be:

```
natPassThruListAdd(inet_addr("10.0.0.100"), inet_addr("255.255.255.0"));
```

After an address is added to the pass-through list, it bypasses the NAT.

An additional function:

```
extern void natPassThruListDelete(u_long ipaddr, u_long netmask);
```

can be used to remove a particular address from the pass-through list.

BUILDING THE VxWORKS IMAGE

The output of a make command is the vxWorks.st image. With this image, VxWorks native debugging is operational and can be used to test and debug network and other facilities. The target also includes the generic VxWorks shell.

To build vxWorks.st, open a DOS command shell. Go to the <Tornado>\host\x86-win32\bin directory and run the torVars.bat batch file to setup the necessary environment variables. The BCM47xx BSP code should have been put into the <Tornado>\target\config\bcm47xx directory. Go to that directory and type make.

The standard build uses -g to include all symbols in the image for use by the VxWorks shell. You can remove this in your final build. The standard build also includes -DDBG (in bcm47xx.mk), which enables some debug output. Again you can remove this in your final build.

By default the build produces a big-endian, DMA-enabled target. Change these with the compile flag settings in the bcm47xx.mk file.

DEBUGGING WITH CROSSWIND

The CrossWind™ debugger from WindRiver can be used to debug the VxWorks kernel as well as any applications that you bundle on your platform. The supplied VxWorks reference source supports debugging over the following interfaces:

- The Ethernet interface
- The BCM47xx internal UART interface
- The external UART interface on the BCM94710D reference platform (or compatible external UART on a customer-designed platform)

By default, CrossWind debugging uses the Ethernet 0 port for debugging purposes. This support is already within the BSP so no changes are necessary. When using the Ethernet port for debugging, only task-level debugging can occur. To perform system-level debugging, one of the serial UART interfaces must be used.

Modifying CONFIG.H

To use the serial port for both task- and system-level debugging, the line in the bcm47xx/bsp/CONFIG.H file that reads:

```
#undef SERIAL_SYS_MODE_DEBUG
```

must be changed to:

```
#define SERIAL_SYS_MODE_DEBUG
```

Preparing Source Files

For files to do source-level debugging must be compiled with the `-g` option so that debug symbols are included. Additionally, it is desirable to turn off optimizations during the compile so that local variables exist in the output code. To do this, compile with the `-O0` option. This preparation can be achieved by adding the following line to the Makefile:

```
ADDED_CFLAGS = -g -O0
```

Choosing Internal or External UART

If you enable serial debugging, the code will attempt to make a connection over an external UART like the ones used in either the BCM94710AP or the BCM94710D reference platforms. If you do not have an external UART device or your external UART device is not compatible with the NS16550, then enable the platform to use the UART device internal to the BCM4710 device by doing the following:

- 1 Disable the Ethernet 0 port.

This is necessary because pins to drive the internal UART are multiplexed with the MII pins for the Ethernet 0 port. To disable the port, use PMON to change the `et0phyaddr` variable to the value 31. See "PMON Boot Loader" on page 33 for more information on viewing and setting NVRAM variables.

- 2 Change the line in the `bcm47xx/bsp/CONFIG.H` file that reads:

```
#define INCLUDE_ET0_END
```

to read

```
#undef INCLUDE_ET0_END
```

- 3 Alter the bootline to read:

```
et(1,0) e= ET0_IP_ADDRESS /*
```

from

```
et(0,0) e= ET0_IP_ADDRESS */
```

- 4 Add `-DINT_UART` flag to the `EXTRA_DEFINES` section of the Makefile in the `bcm47xx/bsp` directory.

On the hardware side:

- 1 Lift pin 6 (RXDV{2}) on the BCM5222 chip.
- 2 Connect SIN (U20) and SOUT (Y22) of the BCM4710 UART interface to the appropriate pins on an external RS-232 driver.
- 3 Connect a PC running terminal software to the RS-232 driver.

When the system is rebooted, an internal UART device will be used for the console as well as for CrossWind debugging.

Tornado Target Server

The target server is a service that runs on the PC that has the Tornado 2.x tools installed. The target server is required to use the following tools:

- WindView[®], a software profiler application from WindRiver
- CrossWind, a gdb-based debugger from WindRiver
- WindSh[™] remote CLI

To configure the Tornado target server:

- 1 Start the Tornado 2.x application by double-clicking the icon.
- 2 On the menu bar, click the **Tools** Menu item followed by **Target Server** and **Configure**.



- 3 At the **Configure Target Servers** dialog, enter the following information:
 - Description: BCM4710
 - Check the **Add description to menu** option.
 - Set **Available Back Ends** to wdbserial for serial or wdrpc for Enet.
 - Set **Serial Port** to the port number (COMx) the host is using to connect to the target.
 - Set **Speed** to 115200.

Note For debugging over Enet, Serial port and Speed options do not exist.



- Set the **Target Name/IP Address** to BCM47xx. For debugging over Enet, this needs to be the Target IP address for example, 10.19.13.181 and so on.
- 4 From the **Target Server Properties** menu, select **Core Files and Symbols**. Select the **File** option and inside the edit box to the right put the vxWorks image location where it was built. For example, if building the BSP-ONLY for the BCM47xx, the path should be something like `c:\Tornado\Target\Config\Bcm47xx\VxWorks.st`.
 - 5 From the **Target Server Properties** menu, select **Memory Cache Size** and change from the default of 1 MB to 8 MB (8000 KB).
 - 6 Click **OK** at the **Configure Target Servers** menu to save this configuration.

To start the Tornado target server:

- 1 To run the target server, from the menu bar click the **Tools** menu followed by **Target Server** and **BCM47xx**.
- 2 When the target server has launched successfully, select the **BCM47xx** target from the menu within the Tornado application. A virtual console should also have been launch with the target server.

From this point, one should be able to run any of the configured Tornado tools such as the Tornado Browser, WindSh, Crosswind, or WindView. Refer to the *Tornado 2.x User's Guide* for more complete details on running each tool.

Debugging the Code

Follow the instructions in "Building the VxWorks image" on page 26 to create a vxWorks.st image. This image should be written to the platform using either visionICE or the PMON TFTP facilities.

Connect a terminal to the serial port and reboot the platform. When choosing the serial debugging technique, you should see PMON boot, and then the VxWorks kernel boot. The last line on the terminal will read:

```
WDB READY
```

Close the terminal application and start the CrossWind debugger.

STATISTICS REPORTING

The BCM4710 device keeps a variety of statistics on all of the various interfaces. The residential gateway image can be configured to periodically take these statistics, as well as other operating parameters of the residential gateway, and upload them to a statistics server. Here, information for a number of residential gateways deployed externally can be coalesced for analysis.

The server where the statistics are stored is specified by the `stats_server` NVRAM variable. If this variable is not set, stats collection does not occur. See "Accessing NVRAM" on page 29 for information on setting and unsetting NVRAM variables.

Additionally, the following lines of code must be uncommented out before the stats task starts:

```
/* initiate the router stats task */
if (taskSpawn("stats", 200, 0, (32*1024), (FUNCPTR)report_stats, 0,
0, 0, 0, 0, 0, 0, 0, 0) == ERROR)
printf("could not start vx stats task\n");
```

As an example, included in the router release is a Perl script, *src/router/misc/stats.pl*. Refer to this script as an example of how to gather these statistics.

ACCESSING NVRAM

A set of functions are provided that allow for NVRAM manipulation. These functions are *nvram_set()*, *nvram_unset()*, *nvram_get()*, *nvram_getall()*, *nvram_show()* and *nvram_commit()*.

int nvram_set(const char *name, const char *value)

Sets the named variable to the specified value. Change must be committed to NVRAM by using the *nvram_commit()* function. Returns 0 if successful, nonzero otherwise.

int nvram_unset(const char *name)

Unsets the named variable. Change must be committed to NVRAM by using the *nvram_commit()* function. Returns 0 if successful, nonzero otherwise.

char *nvram_get(const char *name)

Returns a pointer to a buffer containing the named NVRAM variable's string value. If the variable is not found, returns NULL.

int nvram_getall(char *buf, int count)

Copies name and value for all NVRAM variables into given buf. Size of buffer is specified by count parameter. Returns 0 if operation successful, -1 if given buffer is too small.

void nvram_show(void)

Dumps all NVRAM variables and their values to the console.

int nvram_commit(void)

Commits recent variable changes to NVRAM. Returns 0 if successful, nonzero otherwise.

BCM430X 802.11B WIRELESS SUPPORT

Both the bridge and residential gateway packages include objects to support Broadcom 802.11b BCM430x wireless chips. Both packages share a common configuration interface.

Configuring the BCM430x

The behavior of the BCM430x interface is controlled through a set of NVRAM variables. These variables control such functions as data rate, channel selection, station ID, and encryption method. For a complete list of NVRAM variables related to wireless operation, see Table 23: “NVRAM Environment Variables—802.11b Parameters,” on page 52. For information on NVRAM support routines, see “Accessing NVRAM” on page 29.

Once all of the desired NVRAM variables have been correctly set, the `wlconfig()` function should be called to correctly configure the interface. The format of this call is:

```
wlconfig("wl0")
```

where `wl0` is the name of the wireless interface. Once called, this function will read all of the associated NVRAM variables and reconfigure the wireless interface with the new parameters. The changes will take effect immediately.

HTML-Based Wireless Configuration

For the residential gateway release, wireless configuration can be done through the HTML interface. When applied, the HTML will set the NVRAM variables and use the `wlconfig()` function to reconfigure the interface.

PCI DEVICE SUPPORT

The BCM94710D platform has two PCI slots in which PCI devices can be inserted. Included in the release is an Ethernet driver for PCI Ethernet adapters using the Intel® 82559 controller (like the Intel Pro /100+ 10/100 Ethernet PCI Network Adapter.)

The driver, however, is not built automatically. To enable building and linking of the 82559 driver, the following changes need to be made:

target/config/bcm47xx/bsp/config.h:

Add the following lines:

```
#define INCLUDE_PCI_INTEL_END
#define ET_INTEL_IP_ADDRESS "10.0.0.1" /* This can be any ip address */
```

target/config/bcm47xx/bsp/configNet.h:

Add the following lines:

```
#ifdef INCLUDE_PCI_INTEL_END
#define ET_PCI_LOAD_FUNC fei82557EndLoad
#define ET_PCI_LOAD_STRING ""
IMPORT END_OBJ* ET_PCI_LOAD_FUNC (char*, void*);
#endif
```

Inside `endDevTbl`, add an entry:

```
#ifdef INCLUDE_PCI_INTEL_END
{0, ET_PCI_LOAD_FUNC, ET_PCI_LOAD_STRING, 0,
 NULL, FALSE},
#endif
```

This should be the first entry in the `endDevTbl` table.



target/config/bcm47xx/bsp/sysLib.c:

Add following after calling platform_init inside sysHwInit2 function:

```
#define MAPPED_ADDR 0x8000000
#define PCI_CFG_47XX(d, f, o) (0xAC000000 | (1 << (16+d)) | (f << 8) | o)
/* assume pci card is in slot 2 */
*(unsigned long*)(PCI_CFG_47XX(2, 0, 0x04)) = 0xffffffff; DELAY(1);
*(unsigned long*)(PCI_CFG_47XX(2, 0, 0x10)) = MAPPED_ADDR; DELAY(1);
*(unsigned long*)(PCI_CFG_47XX(2, 0, 0x3c)) = 0x00000101; DELAY(1);
```

target/config/bcm47xx/bsp/vxbsp.h:

Add the INT_LVL_IORQ4 bit into BCM47XX_SR definition to enable pci interrupt.

target/config/bcm47xx/Makefile:

Add fei82557End.c when building the library.

target/config/all/usrConfig.c

Add the following lines in the usrRoot() routine just before the line that reads #ifdef INCLUDE_USER_APPL to attach the driver to the IP stack:

```
#ifdef INCLUDE_PCI_INTEL_END
ipAttach(0, "fei");
ifMaskSet("fei0", 0xFFFFFFFF00);
ifAddrSet("fei0", ET_INTEL_IP_ADDRESS);
#endif
```

Additionally, refer to the VxReadme.txt file included with the release on changes that need to be made to the WindRiver-supplied fei82557End.c and fei82557End.h files.

Note

An Intel® 82559-based PCI driver must be present in the second PCI slot on the BCM94710D reference platform when the VxWorks image that supports this card is booted. The second slot is the rightmost slot when looking at the platform from the front (serial connectors facing you). If the card is not present, the kernel hits an exception and halts. Also, the J45 jumper must be set to use PCI_INT_A. To set this, put a jumper between the two pins closest to R127 on the BMC94710D board.

PCMCIA DEVICE SUPPORT

The BCM94710D platform has a single PCMCIA slot in which a PCMCIA 16-bit device can be inserted. Included in the release is driver for 802.11b wireless LAN cards utilizing the Intersil Prism® chip. Specifically, the driver has been tested with the Lucent PCMCIA Orinoco wireless LAN PC Card.

The driver, however, is not built automatically. To enable building and linking of the Intersil Prism driver, change the following:

target/config/bsp/bcm47xx/config.h:

Add the following lines:

```
#define INCLUDE_PCMCIA
#define INCLUDE_INTERSIL_END
#define ET_ISL_IP_ADDRESS "10.0.0.12" /* This can be any ip address */
```



target/config/bcm47xx/bsp/configNet.h:

Add the following lines:

```
#ifdef INCLUDE_INTERSIL_END
#define ET_PCI_LOAD_FUNC    isl_end_load
#define ET_PCI_LOAD_STRING ""
IMPORT END_OBJ* ET_ISL_LOAD_FUNC (char*, void*);
#endif
```

Inside endDevTbl, add an entry:

```
#ifdef INCLUDE_INTERSIL_END
{0, ET_ISL_LOAD_FUNC, ET_ISL_LOAD_STRING, 0,
 NULL, FALSE},
#endif
```

target/config/bcm47xx/Makefile:

Add *pcmciaLib.c* *pccardLib.c* *cisLib.c* *cisShow.c* *pccardShow.c* *icicShow.c* *icic.c* *islEnd.c* when building the library.

target/config/all/usrConfig.c:

Add the following lines in the *usrRoot()* routine just before the line that reads `#ifdef INCLUDE_USER_APPL` to attach the driver to the IP stack:

```
#ifdef INCLUDE_INTERSIL_END
ipAttach(0, "isl");
ifMaskSet("isl0", 0xFFFFFFFF0);
ifAddrSet("isl0", ET_ISL_IP_ADDRESS);
#endif
```

Also, locate the lines:

```
#ifdef INCLUDE_SHOW_ROUTINES
pcmciaShowInit (); /* install PCMCIA show routines */
#endif /* INCLUDE_SHOW_ROUTINES */
```

and comment them out.

target/src/drv/pcmcia/pcmciaLib.c:

Remove the following line:

```
sysIntEnablePIC (pAdapter->intLevel);
```

target/src/drv/pcmcia/cisLib.c:

In *cisConfigGet* function, add:

```
#if (_BYTE_ORDER != _BIG_ENDIAN)
pCard->regBase = LONGSWAP(pCard->regBase);
pCard->regMask = LONGSWAP(pCard->regMask);
#endif
```

right after:

```
pCard->regBase = base.l;
pCard->regMask = mask.l;
```

In *cisConfigregSet* function, add:

```
#if (_BYTE_ORDER == _BIG_ENDIAN)
pReg = (char *)((int)pReg ^ 3);
#endif
```

12/05/02

right after:

```
pReg = (char *)memWin.start + ....
```

target/src/drv/pcmcia/cisLib.c:

Add the following lines:

```
#define PCMCIA_ICIC0x03
#define PCCARD_LAN_INTERSIL0x04
```

target/src/config/usrExtra.c:

Add the line:

```
#include "../../h/drv/pcmcia/pcmcialib.h"
```

before the line:

```
#include "../../src/config/usrPCMCIA.C"
```

target/h/drv/pcmcia/pccardLib.h:

Add the following lines:

```
typedef struct islResource
{
    PCCARD_RESOURCE resource;
    intintVector;
} ISL_RESOURCE;
```

Additionally, refer to the VxReadme.txt file included with the release on changes that need to be made to the WindRiver-supplied pccardLib.c file.

The source code for the Intersil Prism End device driver is located at target/src/drv/end/islEnd.c. The accompanying header files are located at target/h/drv/end: islreg.h, if_ieee80211.h, and islwavelan_ieee.h.

PMON BOOT LOADER

PMON is an open-source MIPS boot loader ported for the BCM47xx. PMON has three main purposes:

- Performs some hardware diagnostics on power-up
- Acts as a system bootloader on power-up
- Upgrades the OS and application images in the Flash

PMON also supports a serial command-line console. Using a terminal application on your PC, such as HyperTerminal, you can connect to the BCM47xx-based platform to monitor the boot process, monitor the self diagnostics, and perform upgrade functions.

USING THE TFTP APPLICATION

PMON utilizes the TFTP protocol so that images can be sent to the BCM47xx-based platform. These images may be updated OS images or updated PMON images. The TFTP application, running on either Windows or Linux⁵, is used to transfer these images.

In a Windows environment, the TFTP command is entered on a single line:

```
tftp -i <IP Address> <image>
```

<IP Address> is the address of the BCM47xx-based platform. This is usually 192.168.1.1. <image> is the name of the image to be sent. This may be something like `PMON.BIN` or `LINUX.TRX`.

In Linux, a TFTP session will look like this:

```
# tftp <IP Address>
tftp> binary
tftp> put <image>
```

PMON ENVIRONMENT VARIABLES

PMON has several variables which are maintained in the NVRAM portion of the Flash. For more information on the NVRAM section and its use, see "NVRAM" on page 47 of this document. PMON provides commands to manipulate these environment variables as well as updating the NVRAM with new variable values.

set

The set command has three functions:

- To view the entire list of environment variables
- To add a new variable or modify an existing one
- To write the current set of environment variables to the NVRAM

To view the list of current environment variables, issue the `set` command without any parameters.

To add a new environment variable or to modify an existing one, use the format `set <varname> <value>`. For example, to change the `localip` parameter to `10.0.0.1`, issue the command `set localip 10.0.0.1`.

To write the current set of environment variables to the NVRAM, issue the command `set nvram`.

unset

The unset command does exactly what it sounds like it does. It removes an environment variable. To make the changes to the environment variables permanent, issue the `set nvram` command.

5. Broad has found that the TFTP application installed with RedHat 7.x does not work properly. Obtain a Linux TFTP application from some other source, like an open source repository or an earlier RedHat 6.x distribution.

CREATING A DEFAULT NVRAM IMAGE

The NVRAM contains customized variables for a specific board and should *never* be lost, even if there are problems with the Flash in the field. These variables include the MAC address for all of the interfaces, the serial number of the platform, any manufacturing lot numbers, and so on. At boot time, PMON performs a CRC check of this NVRAM configuration area and if this area is invalid or does not have any data, the default NVRAM image is loaded. A default NVRAM image is embedded in the PMON binary itself and is located at offset 0x400 in the image. The length of this area is limited to 8 KB.

If no default NVRAM image is embedded in a PMON binary and the PMON binary is programmed into a BCM47xx-based platform, PMON creates a generic NVRAM image. All generic NVRAM images have the same MAC and IP addresses, so boards programmed this way conflict with each other. These values can be changed from the PMON serial command line using the *set* command described above.

nvserial

Along with the source-code distribution for the BCM47xx is the received NVSERIAL⁶ utility (in the */tools* directory). This utility is used to embed a default NVRAM image in the PMON boot loader binary file.

Most of the default NVRAM information is stored in a plain text file. Along with the distribution are the following files in the */bootrom* directory: *AP_NVDATA.TXT*, *DEV_NVDATA.TXT*, and *4702_NVDATA.TXT*. These files are very similar but have some minor changes with respect to the SDRAM configuration.

The *xx_NVDATA.TXT* files are plain text ASCII files that list the NVRAM environment variables default values. These files should be edited so that the variables match the particulars of the BCM47xx-based platforms. Most importantly, the *sdram_init*, *sdram_config*, and *sdram_refresh* variables **must** be configured to match the memory configuration of your particular platform. For more information, see “SDRAM Configuration” on page 4. You must also modify the MAC address variables for the Ethernet and HPNA interfaces to be unique to the board being programmed.

After modifying the *NVDATA.TXT* information, you can use the *NVSERIAL* utility to embed a default NVRAM in the PMON binary. To do this, enter the command:

```
nvserial -i <input file> -o <output file> -b 0x400 -c 0x2000 -s 42 nvdata.txt
```

This puts 8 KB of NVRAM data at a 1-KB offset with the serial number set to 42 in the default NVRAM data.

It is possible to enter whatever you want for the PMON input and output file and specify a different filename instead of *NVDATA.TXT*. The package includes two (of three) files, *AP_NVDATA.TXT*, and *DEV_NVDATA.TXT*, that correspond to values used by the two reference platforms. *4702_NVDATA.TXT* is included to point users to the proper format for this file given the lack of HPNA interface on the 4702 chip.

PROGRAMMING PMON INTO FLASH

After you have a PMON, you can then embed a default NVRAM image as described above. You are then ready to program the resulting images into Flash.

For programming the PMON image into the target Flash, one could use a standard Data I/O programmer or WindRiver VisionICE tool. See “Programming a Flash Device” on page 44 for information about how to do the actual Flash programming. To use the Vision ICE tool, access the EJTAG port of the BCM47xx in the target board. After the device is programmed, remove the vision ICE pod from the target board, and recycle power for PMON to boot up.

6. Currently only an .exe file that can be run in a Windows environment, not Linux.

Additionally, you can program a new version of PMON into Flash using the TFTP application with the old PMON image. To do this, perform the following steps:

- 1 Follow the instructions in the section “Creating a Default NVRAM Image” on page 35 to embed default NVRAM variables in the new PMON image.
- 2 Ensure that the PC with the new PMON image has its Ethernet interface statically configured to the IP address 192.168.1.100 with a netmask of 255.255.255.0.
- 3 Ensure that the Ethernet interface on the PC is connected to the LAN interface on the BCM47xx-based platform.
- 4 Ensure that a terminal program (such as HyperTerminal) is running on the PC connected to the appropriate serial port on the BCM47xx-based platform. See “Connecting a Serial Monitor to a BCM47xx-based Platform” on page 36 for more information.
- 5 Reset the BCM47xx-based platform and press **Ctrl+C** in the terminal program to get to a PMON> prompt.
- 6 Enter the *set* command to get a list of all of the current NVRAM variables. Take note of the *dl_ram_addr* variable value. This is the location in memory where an image received via TFTP will be stored. This address is used later in the procedure.
- 7 Enter the command *load -B*. This tells PMON to accept an image via TFTP and store it in RAM.
- 8 On the PC, use the TFTP program to send the new PMON image to the platform. See “Using the TFTP Application” on page 34 for more information.
- 9 After the TFTP transfer is completed, the number of transferred bytes is displayed. This byte count will be used in the next step.
- 10 The new PMON image must now be copied into Flash. Enter the command *cp2fl <dl_ram_addr> bfc00000 <byte count>* where *<dl_ram_address>* is the value of that NVRAM variable and *<byte count>* is the number of bytes reported in the TFTP transfer.
- 11 Enter the command *nvErase bfff8000 8000* to erase the NVRAM variables. Upon reset, the default NVRAM variables will be copied from the new PMON image into the NVRAM area.
- 12 Reset the BCM47xx-based platform to boot the new PMON image.

CONNECTING A SERIAL MONITOR TO A BCM47XX-BASED PLATFORM

It is possible to connect to the BCM47xx-based platform through the serial port. For the BCM94710D platform, you only need a null modem serial cable (female DB-9 plug at one end, male DB-9 at other). Whereas the BCM94710AP requires a serial cable (same DB-9 plug configuration) in addition to a custom cable. This custom cable can be manufactured with a female DB-9 plug, a short length of 9- or 10-wire ribbon cable (< 4 inches), and a standard 10-pin ribbon plug. The ribbon cable is attached to the ribbon plug in a standard manner. The ribbon plug, in turn, is connected to the J8 10-pin header (serial interface on the BCM94710AP). This leaves only the mapping of connections between the ribbon plug (J8) and the DB-9 plug. See Table 7.

Table 7: Serial Monitor Configuration

RS-232 Signal	J8 (10-pin header on BCM94710AP)	DB-9 PLUG
RXD	5	2
TXD	3	3
GROUND	9	5

Once the serial connection is made, invoke a serial terminal application (such as minicom, HyperTerminal, Procomm Plus®, TeraTerm) on the PC. Configure the connection to use these parameters:

- 115K baud



- 8 bits
- 1 stop bit
- no parity
- no flow control

After resetting the BCM47xx-based platform, the output is seen as the PMON boot loader executes.

WHEN PMON BOOTS

The PMON boots in the following steps:

- 1 PMON performs self-diagnostics on the SDRAM, Ethernet, and HPNA functions. If there is an error, the appropriate error message is displayed on the serial console and the booting sequence continues.
- 2 PMON waits for three seconds for a new OS image to be loaded into Flash via the LAN Ethernet port.
- 3 PMON tries to boot a compressed OS image stored in the Flash location specified by the `os_flash_addr` environment variable.
- 4 If the OS boot fails, PMON waits for a new OS image on the LAN Ethernet port.

ERASING THE NVRAM AREA

To erase an existing NVRAM image in Flash and start over from scratch, invoke the `nvErase` command in PMON using the following steps:

- 1 Connect a serial monitor to the BCM47xx-based platform as described above.
- 2 As PMOM boots, enter **Ctrl+C** to enter the PMOM prompt.
- 3 Enter the command `nvErase bffe0000 8000` to erase the NVRAM in Flash.
- 4 Reboot the system.

When the system starts, PMON writes a new default NVRAM image into Flash.

LOADING AN OS IMAGE USING PMON

After power-on reset, PMON waits 3 seconds, listening on the LAN Ethernet port to see if a new OS image is transferred. If no image is received, or an image is received with a corrupt or invalid header, PMON tries to boot the OS already located in Flash.

Note

The environment variable, `boot_wait` controls, whether or not PMON waits for an OS image upgrade at power-up. If you want quick boot times and use other means (such as visionICE) to program the Flash, this variable can be set to OFF. But, if `boot_wait` is set to OFF, OS image updates at power-up are no longer possible.

Adding a Header

A header for the binary images must be added using the TRX utility before sending it to PMON for download into the target Flash. PMON checks the header ID, image length, and CRC for every downloaded image. If a header is not found, the downloaded OS image is not saved into the flash and is not auto-booted.

Note A TRX header does not have to be added to the PMON binary itself.



To add a header to a compressed OS image:

```
trx -o os_trx.bin os.bin
```

where

os.bin = original untagged image (input)

os_trx.bin = tagged/added image (output)

TRX can also create a single file out of multiple images. For instance, both the Linux kernel and the Linux root file system can be concatenated into a single image that can be uploaded to the platform using PMON. The following is an example:

```
trx -o linux.trx vmlinux -b 768k rootfs
```

The resulting image has the TRX header at offset 0, immediately followed by the Linux kernel. At an offset of 768 KB into the image, the root filesystem is located. When any image is uploaded via PMON, it begins at the 256 KB offset in Flash. In this example, the Linux file system ultimately resides at the 1MB (768 + 256) boundary in Flash.

To transfer the file to the BCM47xx-based platform, use the TFTP application. TFTP comes standard on both Windows and Linux installations and must be invoked from a command-line prompt. The format for the command is:

```
"tftp -I <IP address> PUT <OS image file>
```

In Linux, enter the following at the prompt:

```
tftp <IP address>
tftp> binary
tftp> put
(file) <OS image file>
```

The IP address of the BCM47xx-based design is defined by the *localip* environment variable. After entering the TFTP command, the image is downloaded to the target RAM starting at the location specified by the *dl_ram_addr* PMON environment variable. PMON then immediately saves this image into the Flash at the location specified by the *os_flash_addr* variable. It then copies or decompresses the image from Flash back into RAM and boot it. PMON assumes the maximum length of the OS image is 3 MB.

USING VISIONCLICK/VISIONICE

The combination of visionCLICK software along with a visionICE hardware debugger provides for a powerful debugging platform to be used with BCM4710-based designs.

The visionCLICK/visionICE tools are available from WindRiver systems. However, some additional files are required to fully utilize these tools with a BCM4710-based hardware design. These additional files are provided along with the source code that is outlined in the Source Code License Agreement (SCLA).

This document provides guidelines in preparing the visionICE environment for a BCM4710-based platform and describes some additional uses beyond normal debugging. Before continuing, refer to the instructions from WindRiver on how to install the visionCLICK software on a Windows PC.

PREPARING VISIONICE FOR USE WITH THE BCM4710

After installing the visionCLICK software on your computer, ensure that you can communicate with the visionICE unit. Communication between the visionCLICK software and the visionICE unit can be through traditional computer ports (COM, LPT) or through an Ethernet connection. The Ethernet connection is recommended, because it provides faster communication with visionICE and remote debugging of hardware platforms.

PREPARING YOUR CODE FOR DEBUGGING WITH VISIONCLICK

When compiling code (PMON or residential gateway) with the tools provided with a Tornado 2.x distribution from WindRiver, then no special steps need to be done. When compiling with the Linux tools provided in the Linux BSP or residential gateway distribution from Broadcom, then the default debugging information format must be changed. Normally, the `-g` option is used to include debugging information. With the Linux tools, this must be changed to `-gdwarf` to use dwarf debugging information that the visionCLICK software can correctly understand.

CONFIGURING THE EJTAG CONNECTION

The VisionIce JTAG connector has fourteen (14) pins whereas the EJTAG block (J10) has only twelve (12). A twelve-pin jumper block may be needed to mount the VisionIce connector onto the BCM94710AP. Note that pins 13-14 of the VisionIce adapter remain unconnected. The pinout for the EJTAG block (J10) is described in the following table.

Table 8: EJTAG Connections

J10 Pin number	EJTAG SIGNAL
1	JTAG_TRST_L
3	JTAG_TDI
5	JTAG_TDO
7	JTAG_TMS
9	JTAG_TCK
11	SW1 RESET ^a
2, 4, 6, 8, 10, 12	GROUND

a. Pin 11 is connected to the hardware reset switch (SW1) of the BCM94710AP.

CONFIGURING THE IP ADDRESS PARAMETERS

For visionCLICK to use the network (Ethernet) connection to communicate with visionICE, visionICE must first be assigned an IP address.

Note



Assigning an IP address using DHCP is not supported! Follow the steps below to program a static IP address into the visionICE.

Connect a serial cable from the visionICE to the serial port on the computer (usually the COM1 port). The visionICE hardware should include an appropriate serial cable. After making the connection, open the terminal software (HyperTerminal for Windows) and make the following settings:

- 8 data bits
- stop bit
- no parity
- no flow control
- 9600 baud

At the *NET>* prompt, configure the IP parameters of the visionICE following these steps:

- 1 Enter the command *ethsetup*.
- 2 From the resulting menu, choose **2. Modify Basic IP Parameters**.
- 3 Choose **1. Enter the IP Manually**. Enter the IP address to be used for the visionICE.
- 4 Choose **1. Enter the Netmask Manually**. Enter the netmask to be used in hexadecimal format. For example, if the netmask is 255.255.255.0, enter 0xfffff00.
- 5 If you access the visionICE through a router, choose **Y** from the *Should this system use a default gateway* prompt. Then enter the default gateway.
- 6 Next choose **8. Save parameters** and then **9. Exit setup mode**.
- 7 Enter the reset command to reset the ICE so that it uses these IP parameters

You should now be able to ping the visionICE at the IP address you setup. If you cannot ping the visionICE, repeat the above steps and verify the entered values.

DEBUGGING WITH A TERMINAL APPLICATION

Use the visionCLICK software for debugging the platform. You can enter some basic visionICE commands with a terminal application. From the *NET>* prompt, enter the *bkm* command. This allows you to enter commands like *dc* to check the version and *cf* to configure the project. To enter the BKM mode, the visionICE must be connected to the EJTAG port on the BCM94710 platform. Make sure pin 1 on the EJTAG connection is connected to pin 1 on the BCM94710 platform.

When entering the *bkm* command, a different prompt displays, either *BKM>* or *ERR>*. Either of these prompts indicates that you are no longer in the *NET>* configuration mode. The difference is that the *BKM>* prompt is displayed when the visionICE has made a connection with the target hardware (for example, the BCM94710AP). The *ERR>* prompt is displayed when there is no connection.

At the *BKM>* or *ERR>* prompt, you can enter the *he* command to get a list of options. For more information, refer to the documentation that accompanied your visionICE.

VERIFYING THE CONNECTION

After all of the IP parameters have been properly set through the serial port, we are now ready to test the network connection between visionCLICK and the visionICE hardware. To do this, perform the following steps:

- 1 Open the visionCLICK software. When installed, visionCLICK puts a shortcut on the Windows start menu.
- 2 On the menu that appears, choose the **Configure** option.
- 3 Click the **Communications** tab.
- 4 In the Normal Port/Rate section, change from *LPT1* to *Net*.
- 5 Enter the IP address that was assigned to the visionICE and click **Connect**.
- 6 In the **Warning: Configuration Conflict** dialog box, choose the **No** option. You can create a new project file later.
The status line should show *Connected...* in the status section near the top of the dialog box. You can also open the Terminal window in visionCLICK to issue commands directly to the visionICE.
- 7 Click **Ok** to close the configuration window. Click **No** on the next dialog box so you can create a new project later.

Note



By default, visionCLICK tries to initialize the target hardware to be debugged. It is not necessary to have the hardware connected to visionICE. If you see a dialog box saying that it is unable to make a connection to the target hardware, ignore it. Check the *Do not display this error window* option to avoid seeing this in the future.

UPGRADING THE VISIONICE FIRMWARE

Main Firmware Upgrade

The visionICE firmware may or may not be compatible with the BCM4710. To check the firmware level, use the *dc* (display configuration) command in the Terminal window within visionCLICK. You will see output similar to:

```
>ERR>dc
Firm Rev= vn 1.8i
```

If the visionICE firmware revision is before 1.8i, you *must* upgrade your visionICE before using it with your BCM4710-based design. Contact WindRiver support to obtain the latest firmware. To upgrade the visionICE unit:

- 1 Ensure there is Ethernet communication between visionCLICK and the visionICE unit.
- 2 From the visionCLICK menu, choose **Update visionICE Firmware** in the Tools menu.
- 3 Choose the visionICE firmware binary file to upgrade to.

The next dialog box warns that the register configuration information can be lost during the upgrade. Because this is an initial upgrade, the register information is probably not relevant to the project, so you do not have to save it. If, however, you have used this visionICE on other projects and want to save the internal settings, then proceed to do so.

- 4 Choose **Yes** to confirm the update and to start the upgrade.

To verify that the upgrade was complete, go back to the Terminal window and enter the *dc* command. Check that the firmware revision matches the upgrade.

Custom Flash Algorithm Firmware Upgrade

Broadcom also distributes a custom Flash firmware. This firmware contains additional algorithms that are useful with BCM4710-based designs. To update the custom firmware flash:

- 1 In the visionCLICK application, choose **Update visionICE Firmware** in the Tools menu.
- 2 Select the **usrflh_ICE2_MIPS.bin** file in the /tools/visionice directory.
- 3 Power cycle the visionICE after the new firmware is loaded.
- 4 Reconnect to the visionICE and check the flash versions in the terminal window from within visionCLICK. To do this, type the *tf version* command in the terminal window. The TF USR driver(s) version should be *C5.7a*.

This new firmware provides support for the AMD29LV32XXT (2048 x 16) 1 device. This is used with the specified Flash when the BCM4710 is operating in little-endian mode. When using this Flash device, but operating the BCM4710 in big-endian mode, use the BROADCOM 320DT (2048 x 16) 1 Device Flash driver instead.

UPDATING A VISIONCLICK INSTALLATION TO SUPPORT THE BCM4710

To properly use the visionCLICK/visionICE tools with a BCM4710-based platform, several of the files that were originally distributed with visionCLICK must be upgraded with Broadcom-supplied versions. (Also, if you have an older visionCLICK installation, these files may not exist at all on your computer.) Broadcom has created a batch file that performs the file updates. The batch file renames any existing BCM4710-specific visionCLICK files to save them and then copies over new files. To perform the update, run the UPDATE.BAT file in the /tools/visionice directory.

After you have made the update, if the visionCLICK software is already running, exit and restart the visionCLICK application for the BCM4710-specific changes to take effect.

Register Configuration Files

To support the reference platforms, a group of register configurations files have been provided. These files tell the visionICE information about the registers within the various cores of the BCM4710 as well as certain operating parameters of the platform, like SDRAM size/type and endianness.

Table 9 shows the correct register configuration file to use for a given platform configuration (note that both the D and AP platforms use the same file).

Table 9: Register Configuration Files

<i>Reference Platform</i>	<i>Big Endian</i>	<i>Little Endian</i>
BCM94710D	BCM4710D_BE.REG	BCM4710D_LE.REG
BCM94710AP	BCM4710D_BE.REG	BCM4710D_LE.REG

Creating a Project File

If you already have a project file for the BCM4710, then you can skip this step. Otherwise, create a project file that tells the visionCLICK software about certain parameters of your BCM4710 project, including where to find the configuration files and what ICE is used. To create a new project file:

- 1 Start the visionCLICK software and choose **Open Project Files** in the File menu.
- 2 On the Project Configuration tab, choose the **New** button to create a new project (this button is located in the lower-left hand corner of the dialog box). Name the project appropriately (for example, BCM4710).
- 3 Fill in several of the fields associated with this new project. To modify any of the fields in the project configuration dialog box, right-click in the field to bring up the context menu. These fields are:
 - **Description:** Enter a textual description of the project you are creating (for example, BCM4710-based Hardware Platform)

- **Emulator Register Configuration File:** Enter the correct register configuration file from the table above. If you are using your own custom register configuration file, you can enter it here. Remember that these files are located in \ESTII\REGFILES\MIPS directory.
 - **Symbol File:** Points to any symbol files you have for the image you are running. For now, leave blank.
 - **Download File:** For now, leave blank.
 - **Source Paths:** Points to the path(s) of the source file for the current symbol file. This is used for source-level debugging. For now, leave blank.
 - **Reset Symbol:** The symbol to jump to when a reset occurs. For now, clear out this entry.
 - **Microprocessor:** The microprocessor you are using. In this case, the BCM4710. The BCM4710 is under the MIPS category when you right-click in this area.
 - **Target Control:** Set to visionICE.
 - **RTOS Debugging:** Choose the target environment you are going to be debugging. Currently, only VxWorks is supported for this option. If using VxWorks, you can see many objects related to the kernel. If you are using Linux, leave it on the VxWorks setting.
 - **Event System:** Choose the **None** option.
 - **Frequency:** Leave as default value.
 - **Text Editor:** Specify a text editor to be used.
 - **Make/BAT File:** Specifies a make or batch file to build your target image allowing builds from within the visionCLICK IDE. For now, leave the default.
 - **ToolBar File:** If you have created a custom toolbar file, you can specify it here. For now, leave blank.
 - **Auto-Playback:** Specifies a file to play back at startup of visionICE. For now, leave blank.
 - **Sim Register File:** For now, leave blank.
 - **Starting Stack Address (highest):** For now, set to 0xFFFFFFFF.
 - **Ending Stack Address (lowest):** For now, set to 0x00000000.
- 4 After completing all the fields of the new project, click the **Save** button at the bottom of the dialog box.
 - 5 Click the **OK** button to close the dialog box. If it asks to activate this project, choose **Yes**. The newly created project is now active.

Writing BCM4710 Register Descriptions to the visionICE

After you have a valid project file for the BCM4710, you need to write the register configuration to the visionICE. This only needs to be done once, because the register configuration is stored in non-volatile memory in the visionICE. To do this, perform the following steps:

- 1 Choose the **Open Project Files...** option from the File menu.
- 2 On the **Project Configuration** tab, locate the **Active Project**, which should be set to the BCM4710.PRJ file just created.
- 3 Right-click in the **Emulator Register Configuration File** and choose the **Load Registers Configuration** option.

A dialog box asks to save the current register configuration. Generally, choose **No**, unless there is a register configuration for some other project that you want to preserve. This writes the register groups, all of the registers for each core, and the bit definitions for these registers into the visionICE's non-volatile memory.

After all of the register information is written, an error dialog box pops up that says something like `>ERR>CF REGFILE BCM4710_D.REG=162518`. Ignore this error.
- 4 Close the Project Configuration menu by clicking the **OK** button. If asked to make this project active, choose **Yes**.
- 5 In the terminal window, issue the `in` command to initialize the BCM4710-based board with the register configuration information just loaded into the visionICE. The Flash device is ready for programming.

PROGRAMMING A FLASH DEVICE

The visionCLICK software allows programming of the Flash device on the BCM4710-based design. This is especially useful for a new board back with a completely blank Flash part. Using the visionCLICK/visionICE gets a Flash image up and running quickly.

Getting Ready

Before programming a Flash device, it is necessary to initialize the visionICE and target platform. To do this, enter the *in* command in the terminal window. The prompt should change to *BKM>*, indicating background debug mode.

Notes on Big-Endian Support

The BCM4710 can support both big-endian and little-endian configurations. Due to the unique architecture for supporting these modes, many Flash programming algorithms cannot function properly in big-endian mode. Broadcom has contracted with WindRiver Systems to develop custom Flash programming algorithms that can operate in big-endian mode. These custom algorithms exist as a firmware upgrade file for the visionICE. See the instructions above about how to install these custom algorithms.

Table 10 shows the Flash device configurations for which there are custom algorithms.

Table 10: Flash Device Configurations

<i>Device</i>	<i>Configuration</i>	<i>Big-Endian Driver</i>	<i>Little-Endian Driver</i>
AMD29LV32XXT	(2048 x 16) 1 device	BROADCOM 320DT (2048 x 16) 1 Device	AMD29LV32XXT (2048 x 16) 1 device

If the BCM4710-based design is operating in little-endian mode, any of the Flash algorithms provided by WindRiver should function properly. If the design operates in big-endian mode and the Flash configuration is not supported by a customer driver, there remains the following choices:

- Contact WindRiver and have them develop a custom driver for the particular Flash configuration. Broadcom can assist in this effort.
- Design a jumper on the BCM4710-based design so that the endian mode can be changed. With this option, it is possible to program the Flash image in little-endian mode and then switch back to big-endian mode for normal operating mode. See "Using the SWAP Utility" for more information on this option.
- Install pre-programmed Flash devices with a boot loader, such as PMON, already installed. The PMON boot loader (provided in the distribution package) allows programming Flash images through the Ethernet port and supports both big-endian and little-endian configurations. See "PMON Boot Loader" on page 33 for more information on programming Flash images using PMON.

Using the SWAP Utility

If needed, to program a big-endian Flash image using the visionICE while in little-endian mode, the images must first be run through the SWAP utility. This utility is provided in the distribution in the /tools/bin directory. The format of the command is *swap <filename>*. The output has *_bin* embedded in the output file. For example, to swap the pmon.bin file:

```
swap pmon.bin
```

The output file is called *pmon_swap.bin*. Use this swapped file in the following section, "Preparing a Flash Binary Image".

PREPARING A FLASH BINARY IMAGE

If the image that you are writing to the Flash is an OS image booted by the PMON boot loader, you must add a header that tells PMON the size of the OS image and contains a checksum to validate the image. Without this header, PMON cannot boot the OS image. See “PMON Boot Loader” on page 33 for more information on how to use the TRX utility to do this.

For a Flash image to be programmed using the visionICE, it must first contain another header defined by WindRiver that instructs the visionCLICK software the size of the image and where in Flash it should be programmed. Broadcom has developed a program called ADDHDR (for both Win32[®] and Linux environments) that adds this header to a flat binary file. (Refer to the documentation on your specific build environment to obtain details on how to create a flat binary image.)

Note The ADDHDR program should be run after any other utilities, such as TRX or NVSERIAL, have been run.



To use ADDHDR, simply issue the command:

```
ADDHDR <filename>
```

where the filename is the flat binary file to prepend to the header. The resulting output file has an _EST suffix before any filename extension. This _EST file is used to program the Flash from within visionCLICK.

WRITING TO THE FLASH DEVICE

After the binary image(s) is prepared for programming, bring up the Flash programming dialog box by choosing the **Program Flash Device** option from the Tools menu. It is here that you set the parameters of the programming task. Below are the various fields and suggested values:

- **Flash Card or PC Host File Name and Path:** Select the flat binary with header to program into the Flash device. To select the image, click the **Select...** button. This displays a dialog box that contains recent images that were used as well as Browse and Remove buttons and a field to specify the Bias of the image. The ADDHDR.EXE utility always adds a header that instructs visionCLICK to program the image to the base of the Flash address space, 0x1FC00000. To program a particular image (like the file system for a Linux environment) into a different location in Flash, put the relative offset in the bias field. For example, to program an image at 0x1FC80000, enter 0x80000 into the bias field. See Table 11 below for a list of bias offsets for particular files. After selecting the appropriate filename and entering the appropriate bias, click the **OK** button to continue.
- **Programming Algorithm:** Select the Flash device that is being used in the design. Table 11 shows Flash algorithms that can be used for various reference designs. Some of these algorithms are part of the Broadcom-supplied custom Flash algorithm upgrade. See “Custom Flash Algorithm Firmware Upgrade” on page 41 for instructions on how to update to the newer Flash algorithms.

Table 11: Flash Programming Algorithms

Reference Board	Flash Device	Big-Endian Driver	Little-Endian Driver
BCM94710AP	AMD29LV320XXT	BROADCOM 320DT (2048 x 16) 1 Device	AMD29LV32XXT (2048 x 16) 1 Device
BCM94710D	Intel 28F320J3	N/A*	INTEL 28F320Jx (2048 x 16) 1 Device

* No big-endian driver available. See “Notes on Big-Endian Support” on page 44. Also, check the **Initialize Target Prior To Erase and/or Program** check box.

- **Device or Sector Base and End Address:** The base address for a Flash device connected to a BCM4710 device is 0x1FC00000. The visionCLICK software can erase the entire flash before programming by choosing the **Erase All** option, or just a portion of the Flash can be erased. For BCM94710AP/D boards that contain BSP or residential gateway images (either VxWorks or Linux), the last 128K of memory should NOT be erased, because it contains the NVRAM for residential gateway environment variables. To leave this portion of the Flash intact, ensure that the **Erase to 0x** option is checked and that the value BFFE0000 is entered into the adjacent field.
- **Available RAM for Workspace for Flash Algorithm:** The Start Address should be set to A0000000 and the Bytes of Target RAM Required should be set to 8000.

Table 12: Flash Algorithm Start Addresses

<i>Flash File</i>	<i>Bias Offset</i>
PMON Boot Loader	0x00000000
VxWorks Compressed Image	0x00040000
Linux Compressed Image	0x00040000

After all the parameters have been set in the Flash programming dialog box, the image can be programmed into the Flash device. At the bottom of the dialog box are three options: Erase and Program, Program Only, and Erase Only. To erase a Flash device, choose Erase Only. To only program (without a prior erase), choose Program Only. The Erase and Program button performs an erase on the Flash prior to programming the image specified.

When programming a single image, use the Erase and Program button. Use the other options when programming multiple images to different locations in Flash at a single sitting. For example, to program a PMON image to the base of Flash and program a VxWorks image at offset 0x40000:

- 1 Point to the PMON image with a bias of 0 and choose **Erase and Program**.
- 2 Point to the VxWorks image with a bias of 0x40000 and then choose **Program Only**.

It is relatively slow to erase the Flash. The fastest option is to do a single erase and multiple programs.

BASIC DEBUGGING

Debugging a Linux Image

Before debugging a Linux image, disable the watchdog timer. In the Linux platform, the watchdog timer is set up so that if there is no processor activity for a specified period of time, the platform is reset. If some unknown bug causes the residential gateway to lock up, it automatically resets and allows the user to continue to use the device, sometimes without ever noticing anything wrong.

This is desirable in the field, but causes problems with debugging. When the visionICE tries to take control of the processor, the watchdog timer thinks that the platform has crashed and causes it to reset, terminating the link between the visionICE and the platform. There are two ways to disable the watchdog timer: through the PMON boot loader or at the Linux prompt. (The Linux BSP does not contain the necessary tools to disable the watchdog timer.)

If you have a Linux residential gateway package:

- 1 Connect a PC running a terminal application to the serial connector on the BCM94710 platform.
The serial parameters are: 115000 baud, 8 bits, 1 stop bit, no parity, no flow control.
- 2 Reset the platform and wait for the prompt *Please press Enter to activate the console*.
- 3 Press the **<Enter>** or **<Return>** key on your PC.

- 4 At the terminal prompt, enter the command `nvruntime unsetenv watchdog`.
- 5 Reset the BCM4710-based platform.

From the PMON boot loader prompt:

- 1 Connect a PC running a terminal applications to the serial connector on the BCM94710 platform.
The serial parameters are: 115000 baud, 8 bits, 1 stop bit, no parity, no flow control.
- 2 Reset the platform. When the message "Downloading from Ethernet, ^C to abort" appears, press **Ctrl+C**.
- 3 At the `PMON>` prompt, enter the command `unset watchdog`, then `set nvruntime`.
- 4 Reset the BCM4710-based platform.

The watchdog timer is now disabled and the visionICE can be used to debug the Linux kernel.

NVRAM

In VxWorks and Linux designs based around the BCM4710, the last 32 KB of Flash are designated as non-volatile RAM (NVRAM) space. This NVRAM is used as a storage space for variables that are used by the PMON boot loader as well as the BSP or residential gateway image.

NVRAM PROTECTION

The NVRAM section of the Flash should be preserved, even if new images are being programmed into the board. This is because the NVRAM contains vital information such as the MAC addresses for each of the interfaces, as well as the serial number and any other vendor-specific identifying information, such as lot number or manufacture date. This information, if lost, can be difficult or even impossible to reconstruct. Even if this information is easily retrievable (perhaps from the boot loader itself), other information such as the current residential gateway IP configuration and any statically forwarded ports should be preserved if at all possible. Loss of this user-configured information can lead to an undesirable user experience.

ACCESSING NVRAM INFORMATION

To maintain NVRAM integrity and ensure that all clients of the NVRAM data (PMON, VxWorks, Linux) can properly access data stored in the NVRAM, the NVRAM memory block must NOT be modified directly. Direct modification of the data in the NVRAM section can result in an unusable NVRAM.



Do not write NVRAM data directly.

To ensure that all clients of the NVRAM information are in sync, each client is compiled with shared NVRAM access libraries. These libraries perform tasks like reading NVRAM environment variables, changing or creating new variables, calculating checksums, and writing NVRAM to Flash.

GETTING AN INITIAL NVRAM ENVIRONMENT

A default NVRAM image can be inserted into a PMON boot loader file. For instructions on how this is done, see “PMON Boot Loader” on page 33.

If a PMON image with default NVRAM is booted on a board, it checks to see if the NVRAM is either missing entirely or is corrupted. If either is true, the default NVRAM is loaded into the NVRAM section of the Flash and is subsequently used by PMON and any OS image loaded into the Flash.

READING/WRITING NVRAM ENVIRONMENT VARIABLES

In general, NVRAM environment variables are read/written to by any of the NVRAM clients. For instance, if the Linux residential gateway is running and a configuration change was made that altered the IP address to be used on the WAN port, the Linux code would write this new information to the NVRAM memory. Also, if some information like a static route entry is no longer needed, the Linux code removes this variable from the NVRAM space. (All of this NVRAM manipulation is done using the NVRAM access libraries described above.)

Viewing the NVRAM variables or creating new ones can be done with the PMON boot loader.

- 1 Connect a serial client to the BCM4710-based board and reset it. The PMON boot information displays.
- 2 Quickly press **Ctrl+C** to break the PMON boot process and get to the PMON prompt.
- 3 Enter the *set* command to see all of the current environment variables.

For more information on PMON NVRAM manipulation commands, see “PMON Environment Variables” on page 34.

NVRAM FORMAT

NVRAM Header

The first 40 bytes of the Flash designated as the start of NVRAM is the NVRAM header. The format of the NVRAM header is as follows:

```
struct nvram_header {
    unsigned long magic;
    unsigned long len;
    unsigned long crc_ver_init;
    unsigned long config_refresh;
    unsigned long reserved;
};
```

The definition for each of the structure members is:

- **Magic:** Unique identifier marking the beginning of the NVRAM data sector. The value of this identifier is 0x48534C46, or “FLSH” in ASCII.
- **Len:** The total length of the NVRAM including the NVRAM header and all of the NVRAM environment strings.
- **Crc_ver_init:**
 - Bits 0–7: NVRAM CRC.
 - Bits 8–15: NVRAM version.
 - Bits 16–27: Value to be written into the SDRAMInit register.
 - Bits 29–31: reserved.

- **Config_refresh:**
 - Bits 0–15: Value to be written into the SDRAM Configuration register.
 - Bits 16–32: Value to be written into the SDRAM Refresh Control register.

NVRAM Environment Variables

The NVRAM environment variables are located directly after the NVRAM header. The environment variables are represented as a series of NULL-terminated strings. The strings are in the format

```
variable=value
```

The end of the strings is delimited by two consecutive NULL bytes. Table 13 through Table 24 list supported variables and their meanings (note that the same list is included in every /bootrom/xx_nvdata.txt file).

Table 13: NVRAM Environment Variables—General Parameters

Parameter	Description
boardtype	Describes the hardware platform. Possible values include bcm94710ap and bcm94710dev.
boardnum	Serial number of board. Used to generate unique MAC addresses for both the WAN and LAN interfaces.
clkfreq	bcm4710 clock frequency in Mhz (default value is 125 Mhz, only other acceptable value is 100 Mhz)
sdram_init	The value to be written to the SDRAM Initialization Control register in the SDRAM core of the BCM4710
sdram_config	The value to be written to the SDRAM Configuration register in the SDRAM core of the BCM4710
sdram_refresh	The value to be written to the SDRAM Refresh Control register in the SDRAM core of the BCM4710
il0macaddr	The MAC address to be used for the HPNA 2.0 port. The address is given in the format 00:11:22:33:44:55.
et0macaddr	The MAC address to be used for the LAN Ethernet port. The address is given in the format 00:11:22:33:44:55
et0phyaddr	MII Address of the physical layer device connected to the LAN Ethernet port on the BCM4710 platform. For 94710R4 reference designs, this should be set to 30.
et0mcdport	MII MCD port of LAN Ethernet port.
et1macaddr	The MAC address to be used for the WAN Ethernet port. The address is given in the format 00:11:22:33:44:55.
et1phyaddr	MII Address of the physical layer device connected to the WAN Ethernet port on the BCM4710 platform. For BCM94710R4 reference designs, this should be set to 5.
et1mcdport	MII MCD port of WAN Ethernet port.
dl_ram_addr	The location in RAM where a downloaded image (to PMON using tftp) is stored.
os_ram_addr	The location in RAM where a compressed OS image, located in Flash, is decompressed.
os_flash_addr	The location in Flash where a compressed OS image is located. On boot, PMON will look for a valid OS image at this location and decompress and execute it.
lan_ipaddr	The IP address to be used on the LAN ports. For the BCM94710R4 reference design, this is 192.168.1.1.
lan_netmask	The subnet to be used on the LAN ports. For the BCM94710R4 reference design, this should be set to 255.255.255.0.
scratch	Scratch location in RAM where a compressed OS image in Flash is copied before it is decompressed.
boot_wait	Whether PMON should wait for an OS image download upon power-up. Set to <i>on</i> to wait for 3 seconds on powerup and set to <i>off</i> to immediately boot OS image on power-up.
watchdog	Interval, in milliseconds, of the system watchdog timer
kernel_mods	A list of Linux device modules to load (for Linux builds only). Default is et il wl. The et (Ethernet) module should always be loaded. If using a BCM4702, the il (HPNA) module should NOT be loaded. And if using a BCM4301-based 802.11b solution, the wl module should be loaded.

Table 14: NVRAM Environment Variables—Interface Parameters

Parameter	Description
lan_ifname	Linux LAN interface device name - typically <i>br0</i>
lan_ifnames	Linux LAN interface enslaved device names - typically <i>eth0 eth2</i>
wan_ifname	Linux WAN interface device name - typically <i>eth1</i>
wan_ifnames	Linux WAN interface enslaved device names - typically blank

Table 15: NVRAM Environment Variables—Firmware Upgrade Parameters

Parameter	Description
sw_version	The current platform software revision
sw_server	URL of the upgrade server

Table 16: NVRAM Environment Variables—TCP/IP Parameters

Parameter	Description
lan_proto	Type of IP addresses to be used on the LAN interface. Can be either <i>static</i> or <i>dhcp</i> .
lan_ipaddr	Default IP address on the LAN interface, typically <i>192.168.1.1</i>
lan_netmask	IP netmask on the LAN interface, typically <i>255.255.255.0</i>
wan_proto	Type of IP addresses to be used on the WAN interface. Can be either <i>static</i> or <i>dhcp</i> .
wan_ipaddr	Default IP address on the WAN interface. If wan_proto was set to DHCP, this should be set to <i>0.0.0.0</i> .
wan_netmask	IP netmask on the WAN interface. If wan_proto was set to DHCP, this should be set to <i>0.0.0.0</i> .
wan_gateway	IP gateway to be used on the WAN interface. If wan_proto was set to DHCP, this should be set to <i>0.0.0.0</i> .
wan_dns	IP address of the DNS server to be used on the WAN interface. If wan_proto was set to DHCP, this field should be left blank.
wan_wins	IP address of the WINS server to be used on the WAN interface. If wan_proto was set to DHCP, this field should be left blank.
wan_hostname	IP host name to be used on the WAN interface. If wan_proto was set to DHCP, this field should be left blank.
wan_domain	IP domain name to be used on the WAN interface. If wan_proto was set to DHCP, this field should be left blank.

Table 17: NVRAM Environment Variables—Firewall Parameters

Parameter	Description
fw_disable	Flag to enable/disable the firewall protection. Set to 0 to keep firewall disabled and set to 1 to disable the firewall.



Table 18: NVRAM Environment Variables—LAN Filters Parameters

Parameter	Description
filter_ip	A list of ranges, separated by spaces, of IP addresses to be filtered on the LAN interfaces. For example, <code>192.168.1.100-192.168.1.101 192.168.1.150-192.168.1.150</code>
filter_tcp	Lists IP address and TCP port ranges that the residential gateway should block. For example, to block ports 80-81 on 192.168.1.100: <code>filter_tcp=192.168.1.100:80-01</code> . To block ports 80-81 on ALL LAN clients: <code>filter_tcp=*:80-81</code>
filter_udp	This variable has the same syntax as the filter_tcp variable. The difference is that it pertains to UDP ports instead of TCP ports.
filter_mac	A list of MAC addresses, separated by commas, to be filtered on the LAN interfaces.

Table 19: NVRAM Environment Variables—Port Forward Parameters

Parameter	Description
forward_tcp	Indicates ranges of TCP port numbers and to where they should be forwarded on the LAN. For instance, to forward ports 80-81 to IP address 192.168.1.100 on the LAN, ports 80-81: <code>80-81>192.168.1.1:80-81</code> . Additional TCP port forwards can be appended, separated by a space character.
forward_udp	Indicates ranges of UDP port numbers and to where they should be forwarded on the LAN. For instance, to forward ports 80-81 to IP address 192.168.1.100 on the LAN, ports 80-81: <code>80-81>192.168.1.1:80-81</code> . Additional UDP port forwards can be appended, separated by a space character.

Table 20: NVRAM Environment Variables—DHCP Server Parameters

Parameter	Description
dhcp_start	Beginning of range of LAN IP addresses to be handed out by the DHCP server.
dhcp_end	End of range of LAN IP addresses to be handed out by the DHCP server.

Table 21: NVRAM Environment Variables—Web Server Parameters

Parameter	Description
http_username	Username required to access configuration Web pages. This field can be left blank if desired.
http_passwd	Default password required to access the configurations Web pages, typically set to <code>admin</code>
http_lanport	Default port number used to access configuration Web pages on the LAN interface, typically set to <code>80</code>
http_wanport	Default port number used to access configuration Web pages from the WAN interface, typically set to <code>8080</code>

Table 22: NVRAM Environment Variables—PPPOE Parameters

Parameter	Description
pppoe_username	Login name to be used when making a PPPOE connection.
pppoe_passwd	Password to be used when making a PPPOE connection.
pppoe_idletime	Timeout value, in seconds, after which the PPPOE connection will be terminated if there has been no activity.
pppoe_keepalive	Set to 1 to automatically reconnect when a PPPoE link goes down.
pppoe_demand	Set to 1 to only bring up the PPPoE link when it is necessary. (Note: This functionality is not currently enabled.)
pppoe_mru	Maximum size of PPPOE receive packets
pppoe_mtu	Maximum size of PPPOE transmit packets

Table 23: NVRAM Environment Variables—802.11b Parameters

Parameter	Description
wl_ssid	Sets the network name (SSID) to be used.
d11b_channel	Sets the channel to be used when creating networks.
wl_country	Sets the country code. See 802.11b specification for values.
d11b_rate	Sets the rate to be used, in bps. So 5.5Mbps = 5500000.
wl_closed	Network will be closed if set to 1, open if set to 0. If network is closed, it will not respond to active scans by clients and will be effectively hidden.
d11b_frag	Sets the fragmentation threshold.
d11b_rts	Sets the RTS threshold.
d11b_dtim	Sets the wakeup interval for clients in power-save mode.
d11b_bcn	Sets the beacon interval for the AP (in milliseconds.)
wl_wep	Set to <i>off</i> to disable WEP data encryption. Set to <i>on</i> to enable WEP encryption and require a key. Set to <i>restricted</i> to enable WEP and drop all unencrypted packets.
wl_auth	Set to <i>Open System</i> for no authentication. Set to <i>Shared Key</i> to enable authentication.
wl_key	Set to 1-4 to select the current key in use.
wl_key1 - wl_key4	The 4 available keys. Only one is in use at a time, as determined by the <i>wl_key</i> index variable. These keys must be either: <ul style="list-style-type: none"> • 5 ASCII or 10 hexadecimal characters for 64-bit key • 12 ASCII or 26 hexadecimal characters for a 128-bit key
wl_mac	Sets the MAC addresses in the allow/deny list. Each individual address is a string in the form 00:11:22:33:44:55. Multiple addresses can be stored in this variable separated by a space.
wl_macmode	Indicates whether the addresses in the <i>wl_mac</i> list are “allow” or “deny”. When the list is set to “allow,” then only MAC addresses that are in the list are allowed to associate. When the list is set to “deny,” every MAC address is able to associate except ones in the list.
d11b_rateset	Set to “default” for support of (1,2) or set to “all” for support of (1,2,5.5,11)
d11b_plcphdr	Sets the PLCP header type. Valid entries are “long” and “short.”

Table 24: NVRAM Environment Variables—Restore Defaults Parameters

Parameter	Description
restore_defaults	Set to restore to factory defaults on the next reset—0 if no restore required, 1 to perform restore.

Broadcom Confidential

Broadcom Confidential

Broadcom Corporation

16215 Alton Parkway
P.O. Box 57013
Irvine, California 92619-7013
Phone: 949-450-8700
Fax: 949-450-8710

Broadcom[®] Corporation reserves the right to make changes without further notice to any products or data herein to improve reliability, function, or design. Information furnished by Broadcom Corporation is believed to be accurate and reliable. However, Broadcom Corporation does not assume any liability arising out of the application or use of this information, nor the application or use of any product or circuit described herein, neither does it convey any license under its patent rights nor the rights of others.