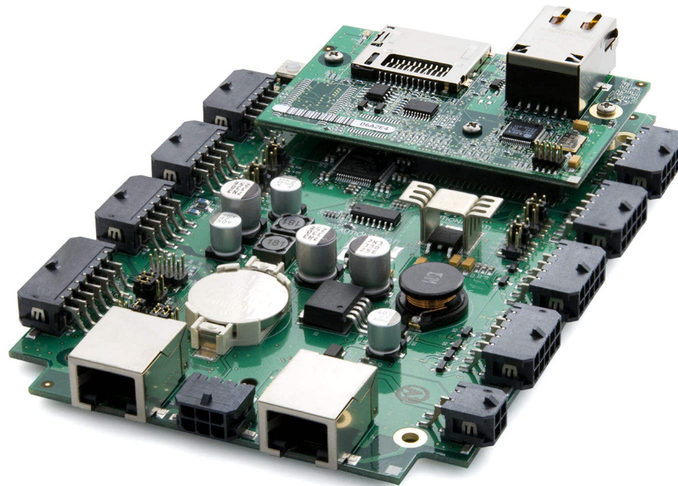
RABBIT™

PRODUCT MANUAL



BL4S200

C-Programmable Single-Board Computer with Networking

User’s Manual

019–0171_F

BL4S200 User's Manual

Part Number 019-0171_F • Printed in U.S.A.

©2008–2013 Digi International Inc. • All rights reserved.

Digi International reserves the right to make changes and
improvements to its products without providing notice.

Trademarks

Rabbit, RabbitCore, and Dynamic C are registered trademarks of Digi International Inc.

RabbitNet is a trademark of Digi International Inc.

The latest revision of this manual is available on the Rabbit Web site, www.digi.com, for free, unregistered download.

Digi International Inc.

www.digi.com

TABLE OF CONTENTS

Chapter 1. Introduction 6

1.1 BL4S200 Description 6
1.2 BL4S200 Features....6
1.3 Development and Evaluation Tools....8
1.3.1 Tool Kit....8
1.3.2 Software....9
1.3.3 Optional Add-Ons....9
1.4 RabbitNet Peripheral Cards....10
1.5 CE Compliance....11
1.5.1 Design Guidelines....12
1.5.2 Interfacing the BL4S200 to Other Devices....12
1.6 Wi-Fi Certifications (BL5S220 Model only)....13
1.6.1 FCC Part 15 Class B....13
1.6.2 Industry Canada Labeling....14
1.6.3 Europe....15

Chapter 2. Getting Started 16

2.1 Preparing the BL4S200 for Development....16
2.2 BL4S200 Connections....17
2.2.1 Hardware Reset....18
2.3 Installing Dynamic C....19
2.4 Starting Dynamic C....20
2.5 Run a Sample Program....20
2.5.1 Troubleshooting....20
2.6 Run a Wi-Fi Sample Program (BL5S220 only)....21
2.7 Run a ZigBee Sample Program (BL4S230 only)....22
2.8 Where Do I Go From Here?....23

Chapter 3. Subsystems 24

3.1 BL4S200 Pinouts....25
3.1.1 Connectors....26
3.2 Digital I/O....27
3.2.1 Configurable I/O....27
3.2.2 High-Current Digital Outputs....34
3.3 Serial Communication....36
3.3.1 RS-232....36
3.3.2 RS-485....36
3.3.3 Programming Port....38
3.3.4 Ethernet Port....39
3.4 A/D Converter Inputs....40
3.4.1 A/D Converter Calibration....42
3.5 D/A Converter Outputs....43
3.5.1 D/A Converter Calibration....44
3.6 Analog Reference Voltages Circuit....45
3.7 USB Programming Cable....46
3.7.1 Changing Between Program Mode and Run Mode....46

3.8 Other Hardware....47
3.8.1 Clock Doubler....47
3.8.2 Spectrum Spreader....48
3.9 Memory....49
3.9.1 SRAM....49
3.9.2 Flash Memory....49
3.9.3 VBAT RAM Memory....49
3.9.4 *microSD*™ Cards....49

Chapter 4. Software 51
4.1 Running Dynamic C....51
4.1.1 Upgrading Dynamic C....53
4.1.2 Add-On Modules....53
4.2 Sample Programs....54
4.2.1 Digital I/O....55
4.2.2 Serial Communication....60
4.2.3 A/D Converter Inputs....62
4.2.4 D/A Converter Outputs....64
4.2.5 Use of *microSD*™ Cards with BL4S200 Model....66
4.2.6 Real-Time Clock....66
4.2.7 TCP/IP Sample Programs....66
4.3 BL4S200 Libraries....67
4.4 BL4S200 Function Calls....68
4.4.1 Board Initialization....68
4.4.2 Digital I/O....69
4.4.3 High-Current Outputs....92
4.4.4 Rabbit RIO Interrupt Handlers....104
4.4.5 Serial Communication....108
4.4.6 A/D Converter Inputs....110
4.4.7 D/A Converter Outputs....123
4.4.8 SRAM Use....131

Chapter 5. Using the Ethernet TCP/IP Features 132
5.1 TCP/IP Connections....132
5.2 TCP/IP Sample Programs....134
5.2.1 How to Set IP Addresses in the Sample Programs....134
5.2.2 How to Set Up your Computer for Direct Connect....135
5.2.3 Run the **PINGME.C** Demo....136
5.2.4 Running More Demo Programs With a Direct Connection....137
5.3 Where Do I Go From Here?....137

Chapter 6. Using the Wi-Fi Features 138
6.1 Introduction to Wi-Fi....138
6.1.1 Infrastructure Mode....138
6.1.2 Ad-Hoc Mode....139
6.1.3 Additional Information....139
6.2 Running Wi-Fi Sample Programs....140
6.2.1 Wi-Fi Setup....141
6.2.2 What Else You Will Need....142
6.2.3 Configuration Information....143
6.2.4 Wi-Fi Sample Programs....146
6.2.5 RCM5400W Sample Programs....151
6.3 Dynamic C Wi-Fi Configurations....154
6.3.1 Configuring TCP/IP at Compile Time....154
6.3.2 Configuring TCP/IP at Run Time....158
6.3.3 Other Key Function Calls....158
6.4 Where Do I Go From Here?....159

Chapter 7. Using the ZigBee Features 160
7.1 Introduction to the ZigBee Protocol 160
7.2 ZigBee Sample Programs 161
7.2.1 Setting Up the Digi XBee USB Coordinator 162
7.2.2 Setting up Sample Programs 164
7.3 Dynamic C Function Calls 167
7.4 Where Do I Go From Here? 167

Appendix A. Specifications 168
A.1 Electrical and Mechanical Specifications 169
A.1.1 Exclusion Zone 173
A.1.2 Headers 173
A.2 Conformal Coating 174
A.3 Jumper Configurations 175
A.4 Use of Rabbit Microprocessor Parallel Ports 177

Appendix B. Power Supply 178
B.1 Power Supplies 178
B.1.1 Power for Analog Circuits 179
B.2 Batteries and External Battery Connections 179
B.2.1 Replacing the Backup Battery 180
B.3 Power to Peripheral Cards 181

Appendix C. Demonstration Board 182
C.1 Connecting Demonstration Board 183
C.2 Demonstration Board Features 184
C.2.1 Pinout 184
C.2.2 Configuration 184

Appendix D. Rabbit RIO Resource Allocation 186
D.1 Configurable I/O Pin Associations 187
D.2 High-Current Output Pin Associations 188
D.3 Interpreting Error Codes 188

Appendix E. RabbitNet 190
E.1 General RabbitNet Description 190
E.1.1 RabbitNet Connections 190
E.1.2 RabbitNet Peripheral Cards 191
E.2 Physical Implementation 192
E.2.1 Control and Routing 192
E.3 Function Calls 193
E.3.1 Status Byte 203

Appendix F. Additional Configuration Instructions 204
F.1 XBee Module Firmware Downloads 204
F.1.1 Dynamic C v. 10.44 and Later 204
F.2 Digi® XBee USB Configuration 205
F.2.1 Additional Reference Information 206
F.2.2 Update Digi® XBee USB Firmware 208

Index 209

Schematics 213

1. Introduction

The BL4S200 series of high-performance, C-programmable single-board computers offers built-in digital and analog I/O combined with Ethernet, Wi-Fi, or ZigBee network connectivity in a compact form factor. The BL4S200 single-board computers are ideal for both discrete manufacturing and process-control applications.

A Rabbit® 4000 or Rabbit® 5000 microprocessor provides fast data processing. A removable flash memory option supports a full directory file structures to maximize remote access control and programmability. The I/O can be expanded with RabbitNet peripheral cards.

1.1 BL4S200 Description

Throughout this manual, the term BL4S200 refers to the complete series of BL4S200 single-board computers unless other production models are referred to specifically.

The BL4S200 is an advanced single-board computer that incorporates the powerful Rabbit 4000 or Rabbit 5000 microprocessor, flash memory options, static RAM, digital I/O ports, A/D converter inputs, D/A converter outputs, RS-232/RS-485 serial ports, and Ethernet, Wi-Fi, or ZigBee network connectivity.

1.2 BL4S200 Features

- Rabbit® 4000 or Rabbit® 5000 microprocessor operating at up to 73.73 MHz.
- Industry-standard Micro-Fit® polarized positive-locking connectors.
- 512KB SRAM and 512KB/1MB flash memory options.
- 40 digital I/O: 32 protected digital I/O individually software-configurable as inputs or sinking outputs, and 8 high-current digital outputs software-configurable as sinking or sourcing.
- Advanced input capabilities including event counting, event capture, and quadrature decoders that may be set up on most I/O pins.
- Independent PWM and PPM capability on most I/O pins and all high-current outputs.
- 10 analog channels: eight 11-bit A/D converter inputs, two 12-bit D/A converter 0–10 V or ±10 V buffered outputs.
- Ethernet, Wi-Fi, or ZigBee network connectivity.
- Up to 5 serial ports:
    - ▶ Up to three serial ports (one 5-wire RS-232 or two 3-wire RS-232, one RS-485).

- ▶ Two RabbitNet™ expansion ports multiplexed from one serial port.
- ▶ One serial port dedicated to programming/debugging.

- Battery-backed real-time clock.
- Watchdog supervisor.

Four BL4S200 models are available. Their standard features are summarized in Table 1.

Table 1. BL4S200 Models

Feature	BL4S200	BL4S210	BL5S220	BL4S230
Microprocessor	Rabbit® 4000 running at 58.98 MHz		Rabbit® 5000 running at 73.73 MHz	Rabbit® 4000 running at 29.49 MHz
Program Execution SRAM	512KB	—	512KB	—
Data SRAM	512KB	512KB	512KB	512KB
Flash Memory (program)	512KB (serial flash)	512KB (parallel flash)	512KB (parallel flash)	512KB (parallel flash)
Flash Memory (data storage)	supports *microSD™ Card* 128MB–1GB	—	1MB (serial flash)	
Network Interface	10/100Base-T, 3 LEDs	10Base-T, 2 LEDs	Wi-Fi (802.11b/g)	ZigBee 2007 (802.15.4)
RabbitCore Module Used	RCM4310	RCM4010	RCM5400W	RCM4510W (ZB)

Note that the BL5S220 model is named as such to reflect that it uses a Rabbit 5000 micro-processor.

BL4S200 single-board computers consist of a main board with a RabbitCore module. Refer to the RabbitCore module manuals, available on the Web site, for more information on the RabbitCore modules, including their schematics.

BL4S200 single-board computers are programmed over a standard PC USB port through a programming cable supplied with the Tool Kit. The BL4S200 and BL5S220 models may also be programmed remotely using the Remote Program Update library with Dynamic C v. 10.54 or later. See Application Note AN421, ***Remote Program Update***, for more information.

> **NOTE:** BL4S200 Series single-board computers cannot be programmed via the RabbitLink.

Appendix A provides detailed specifications.

Visit the Web site for up-to-date information about additional add-ons and features as they become available. The Web site also has the latest revision of this user’s manual.

1.3 Development and Evaluation Tools

1.3.1 Tool Kit

A Tool Kit contains the hardware essentials you will need to use your own BL4S200 single-board computer. These items are supplied in the Tool Kit.

- ***Getting Started*** instructions.
- ***Dynamic C*** CD-ROM, with complete product documentation on disk.
- USB programming cable, used to connect your PC USB port to the BL4S200.
- Universal AC adapter, 12 V DC, 1 A (includes Canada/Japan/U.S., Australia/N.Z., U.K., and European style plugs).
- Stand-offs to serve as legs for the BL4S200 board during development.
- Demonstration Board with pushbutton switches and LEDs. The Demonstration Board can be hooked up to the BL4S200 to demonstrate the I/O and capabilities of the BL4S200.
- CAT 5/6 Ethernet crossover cable.
- Cable assemblies with Micro-Fit® connectors.
- ***Rabbit 4000 Processor Easy Reference*** and ***Rabbit 5000 Processor Easy Reference*** posters.
- Screwdriver.
- Registration card.

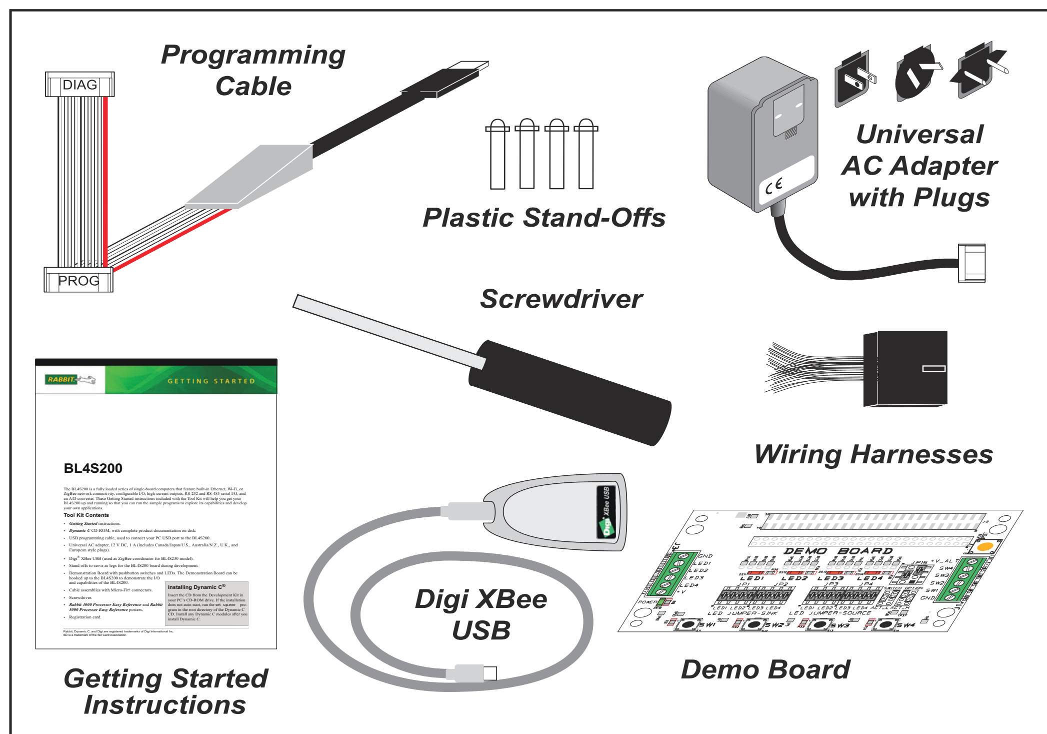


Figure 1. BL4S200 Tool Kit

1.3.2 Software

The BL4S200 is programmed using version 10.42 or later of Rabbit's Dynamic C. A compatible version is included on the Tool Kit CD-ROM. This version of Dynamic C includes the popular µC/OS-II real-time operating system, point-to-point protocol (PPP), FAT file system, RabbitWeb, and the Rabbit Embedded Security Pack featuring the Secure Sockets Layer (SSL) and a specific Advanced Encryption Standard (AES) library.

In addition to the Web-based technical support included at no extra charge, a one-year telephone-based technical support subscription is also available for purchase. Visit our Web site at www.digi.com for further information and complete documentation, or contact your Rabbit sales representative or authorized distributor

1.3.3 Optional Add-Ons

Rabbit has available a Mesh Network Add-On Kit and additional tools and parts to help you to make your own wiring assemblies with the friction-lock connectors.

- Mesh Network Add-On Kit (Part No. 101-1272)
  - ▶ Digi® XBee USB (used as ZigBee coordinator)
  - ▶ XBee Series 2 RF module
  - ▶ RF Interface module

  The XBee Series 2 RF module is installed on the RF Interface module, which can be connected via an RS-232 serial connection to a Windows PC for setup. The Mesh Network Add-On Kit enables you to explore the wireless capabilities of the BL4S230 model that offers a ZigBee network interface.
- Connector Cable Assemblies (Part No. 151-0153)—Two 2 × 5 friction-lock connectors (3 mm pitch) assembled with wiring harness.
- Crimp tool (Part No. 998-0013) to secure wire in crimp terminals.

Visit our Web site at www.digi.com or contact your Rabbit sales representative or authorized distributor for further information.

1.4 RabbitNet Peripheral Cards

RabbitNet™ is an SPI serial protocol that uses a robust RS-422 differential signalling interface (twisted-pair differential signaling) to run at a fast 1 Megabit per second serial rate. BL4S200 single-board computers have two RabbitNet ports, each of which can support one peripheral card. Distances between a master processor unit and peripheral cards can be up to 10 m or 33 ft.

The following low-cost peripheral cards are currently available.

- Digital I/O
- A/D converter
- D/A converter
- Relay card
- Display/Keypad interface

Appendix E provides additional information on RabbitNet peripheral cards and the RabbitNet protocol. Visit our Web site for up-to-date information about additional add-ons and features as they become available.

1.5 CE Compliance

Equipment is generally divided into two classes.

CLASS A	CLASS B
Digital equipment meant for light industrial use	Digital equipment meant for home use
Less restrictive emissions requirement: less than 40 dB µV/m at 10 m (40 dB relative to 1 µV/m) or 300 µV/m	More restrictive emissions requirement: 30 dB µV/m at 10 m or 100 µV/m

These limits apply over the range of 30–230 MHz. The limits are 7 dB higher for frequencies above 230 MHz. Although the test range goes to 1 GHz, the emissions from Rabbit-based systems at frequencies above 300 MHz are generally well below background noise levels.

The BL4S200 single-board computer has been tested and was found to be in conformity with the following applicable immunity and emission standards. The BL4S210, BL5S220, and BL4S230 single-board computers are also CE qualified as they are sub-versions of the BL4S200 single-board computer. Boards that are CE-compliant have the CE mark.

Immunity

The BL4S200 series of single-board computers meets the following EN55024/1998 immunity standards.

- EN61000-4-3 (Radiated Immunity)
- EN61000-4-4 (EFT)
- EN61000-4-6 (Conducted Immunity)

Additional shielding or filtering may be required for a heavy industrial environment.

Emissions

The BL4S200 series of single-board computers meets the following emission standards.

- EN55022:1998 Class B
- FCC Part 15 Class B

Your results may vary, depending on your application, so additional shielding or filtering may be needed to maintain the Class B emission qualification.

1.5.1 Design Guidelines

Note the following requirements for incorporating the BL4S200 series of single-board computers into your application to comply with CE requirements.

General

- The power supply provided with the Tool Kit is for development purposes only. It is the customer's responsibility to provide a CE-compliant power supply for the end-product application.
- When connecting the BL4S200 single-board computer to outdoor cables, the customer is responsible for providing CE-approved surge/lighting protection.
- Rabbit recommends placing digital I/O or analog cables that are 3 m or longer in a metal conduit to assist in maintaining CE compliance and to conform to good cable design practices.
- When installing or servicing the BL4S200, it is the responsibility of the end-user to use proper ESD precautions to prevent ESD damage to the BL4S200.

Safety

- All inputs and outputs to and from the BL4S200 series of single-board computers must not be connected to voltages exceeding SELV levels (42.4 V AC peak, or 60 V DC).
- The lithium backup battery circuit on the BL4S200 single-board computer has been designed to protect the battery from hazardous conditions such as reverse charging and excessive current flows. Do not disable the safety features of the design.

1.5.2 Interfacing the BL4S200 to Other Devices

Since the BL4S200 series of single-board computers is designed to be connected to other devices, good EMC practices should be followed to ensure compliance. CE compliance is ultimately the responsibility of the integrator. Additional information, tips, and technical assistance are available from your authorized Rabbit distributor, and are also available on our Web site at www.digi.com.

1.6 Wi-Fi Certifications (BL5S220 Model only)

The systems integrator and the end-user are ultimately responsible for the channel range and power limits complying with the regulatory requirements of the country where the end device will be used. Dynamic C function calls and sample programs illustrate how this is achieved by selecting the country or region, which sets the channel range and power limits automatically. See Section 6.2.4.1 for additional information and sample programs demonstrating how to configure an end device to meet the regulatory channel range and power limit requirements.

Only RCM5400W modules bearing the FCC certification are certified for use in Wi-Fi enabled end devices associated with the BL5S220 model, and any applications must have been compiled using Dynamic C v. 10.40 or later. The certification is valid only for RCM5400W modules equipped with the dipole antenna that is included with the modules, or a detachable antenna with a 60 cm coaxial cable (Digi International part number 29000105). Changes or modifications to this equipment not expressly approved by Digi International may void the user's authority to operate this equipment.

In the event that these conditions cannot be met, then the FCC certification is no longer considered valid and the FCC ID can not be used on the final product. In these circumstances, the systems integrator or end-user will be responsible for re-evaluating the end device (including the transmitter) and obtaining a separate FCC certification.

> **NOTE:** Any regulatory certification is voided if the RF shield on the RCM5400W module is removed.

1.6.1 FCC Part 15 Class B

The RCM5400W RabbitCore module has been tested and found to comply with the limits for Class B digital devices pursuant to Part 15 Subpart B, of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential environment. This equipment generates, uses, and can radiate radio frequency energy, and if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try and correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and the receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

Labeling Requirements (FCC 15.19)

FCC ID: VCB-E59C4472

This device complies with Part 15 of FCC rules. Operation is subject to the following two conditions:

(1) this device may not cause harmful interference, and

(2) this device must accept any interference received, including interference that may cause undesired operation.

If the FCC identification number is not visible when the module is installed inside another device, then the outside of the device into which the module is installed must also display a label referring to the enclosed module or the device must be capable of displaying the FCC identification number electronically. This exterior label can use wording such as the following: “Contains Transmitter Module FCC ID: VCB-E59C4472” or “Contains FCC ID: VCB-E59C4472.” Any similar wording that expresses the same meaning may be used.

The following caption must be included with documentation for any device incorporating the RCM5400W RabbitCore module.

Caution — Exposure to Radio-Frequency Radiation.

To comply with FCC RF exposure compliance requirements, for mobile configurations, a separation distance of at least 20 cm must be maintained between the antenna of this device and all persons.

This device must not be co-located or operating in conjunction with any other antenna or transmitter.

1.6.2 Industry Canada Labeling

7143A-E59C4472

This Class B digital apparatus complies with Canadian standard ICES-003.

Cet appareil numérique de la classe B est conforme à la norme NMB-003 du Canada.

1.6.3 Europe

The marking shall include as a minimum:

- the name of the manufacturer or his trademark;
- the type designation;
- equipment classification, (see below).

Receiver Class	Risk Assessment of Receiver Performance
1	Highly reliable SRD communication media, e.g., serving human life inherent systems (may result in a physical risk to a person).
2	Medium reliable SRD communication media, e.g., causing inconvenience to persons that cannot be overcome by other means.
3	Standard reliable SRD communication media,e.g., inconvenience to persons that can simply be overcome by other means.

NOTE: Manufacturers are recommended to declare the classification of their devices in accordance with Table 2 and EN 300 440-2 [5] clause 4.2, as relevant. In particular, where an SRD that may have inherent safety of human life implications, manufacturers and users should pay particular attention to the potential for interference from other systems operating in the same or adjacent bands.

Regulatory Marking

The equipment shall be marked, where applicable, in accordance with CEPT/ERC Recommendation 70-03 or Directive 1999/5/EC, whichever is applicable. Where this is not applicable, the equipment shall be marked in accordance with the National Regulatory requirements.

2. GETTING STARTED

Chapter 2 explains how to connect the programming cable and power supply to the BL4S200.

2.1 Preparing the BL4S200 for Development

Position the BL4S200 as shown below in Figure 2. Attach the four stand-offs supplied with the Tool Kit in the holes at the corners as shown.

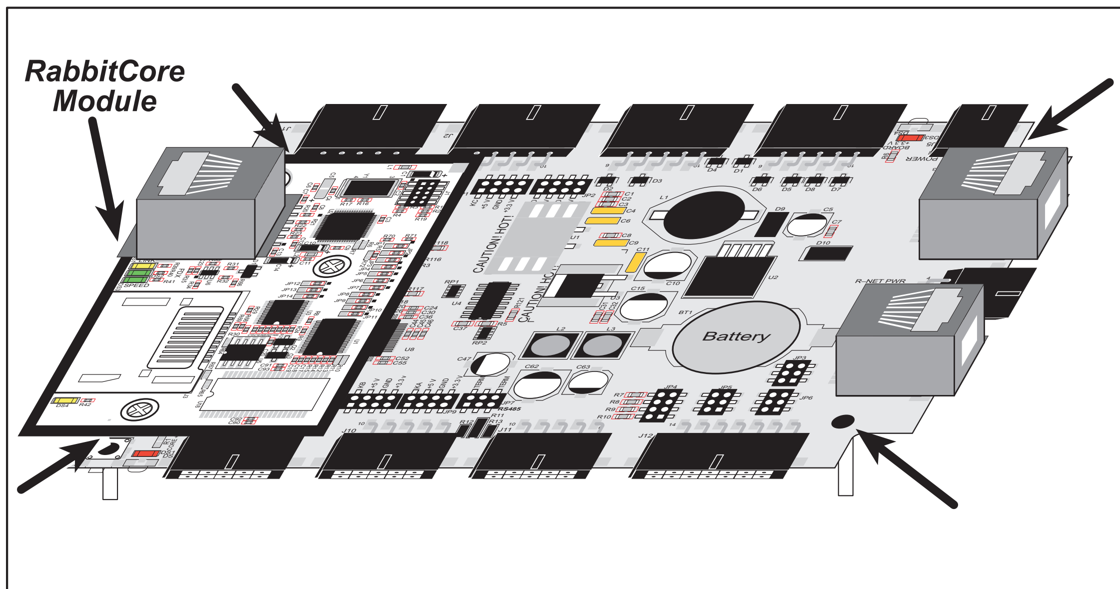


Figure 2. Attach Stand-Offs to BL4S200 Board

The stand-offs facilitate handling the BL4S200 during development, and protect the bottom of the printed circuit board against scratches or short circuits while you are working with the BL4S200.

> **NOTE:** If you ever need to remove the RabbitCore module, take care to keep the BL4S200 main boards and their corresponding RabbitCore modules paired since the RabbitCore modules store calibration constants specific to the BL4S200 main board to which they are plugged in. If you use a RabbitCore module from a different model in the BL4S200 series, your specific BL4S200 model may no longer operate as designed.

2.2 BL4S200 Connections

1. Connect the programming cable to download programs from your PC and to program and debug the BL4S200.

Connect the 10-pin **PROG** connector of the programming cable to header J1 on the BL4S200's RabbitCore module (the programming header is labeled J2 on the BL5S220 and BL4S230 models). Ensure that the colored edge lines up with pin 1 as shown. (Do not use the **DIAG** connector, which is used for monitoring only.) Connect the other end of the programming cable to an available USB port on your PC or workstation.

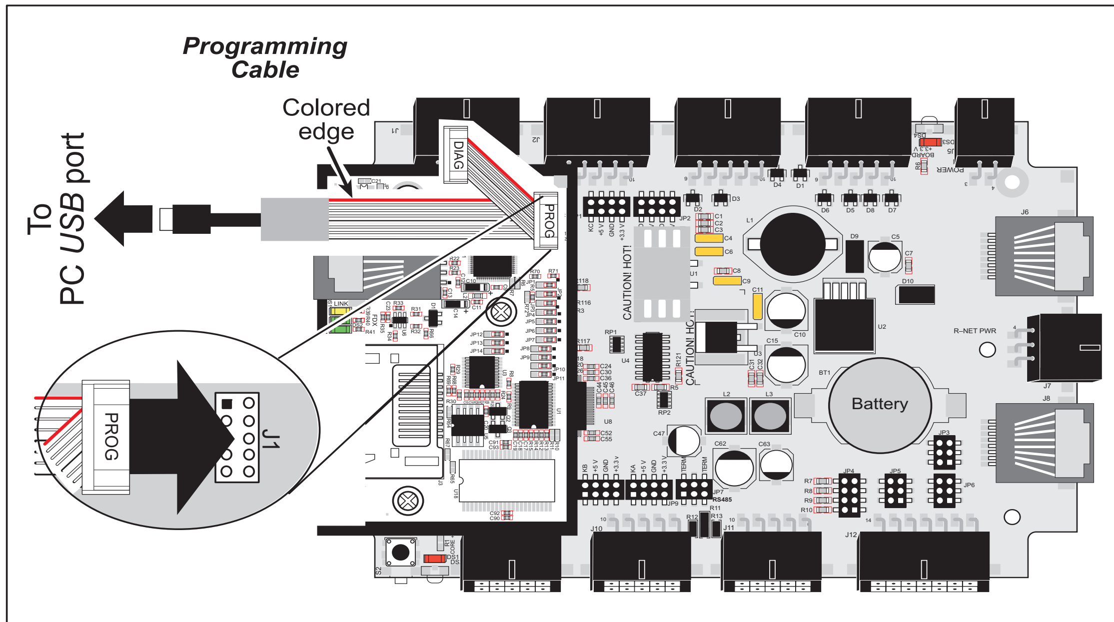


Figure 3. Programming Cable Connections

> **NOTE:** Never disconnect the programming cable by pulling on the ribbon cable. Carefully pull on the connector to remove it from the header.

Connect the other end of the programming cable to an available USB port on your PC or workstation.

Your PC should recognize the new USB hardware, and the LEDs in the shrink-wrapped area of the USB programming cable will flash — if you get an error message, you will have to install USB drivers. Drivers for Windows XP are available in the Dynamic C **`Drivers\Rabbit USB Programming Cable\WinXP_2K`** folder — double-click **`DPInst.exe`** to install the USB drivers. Drivers for other operating systems are available online at www.ftdichip.com/Drivers/VCP.htm.

2. Connect the power supply to header J5 on the BL4S200 as shown in Figure 4. Be sure to match the latch mechanism with the top of the connector to header J5 on the BL4S200 as shown. The Micro-Fit® connector will only fit one way.

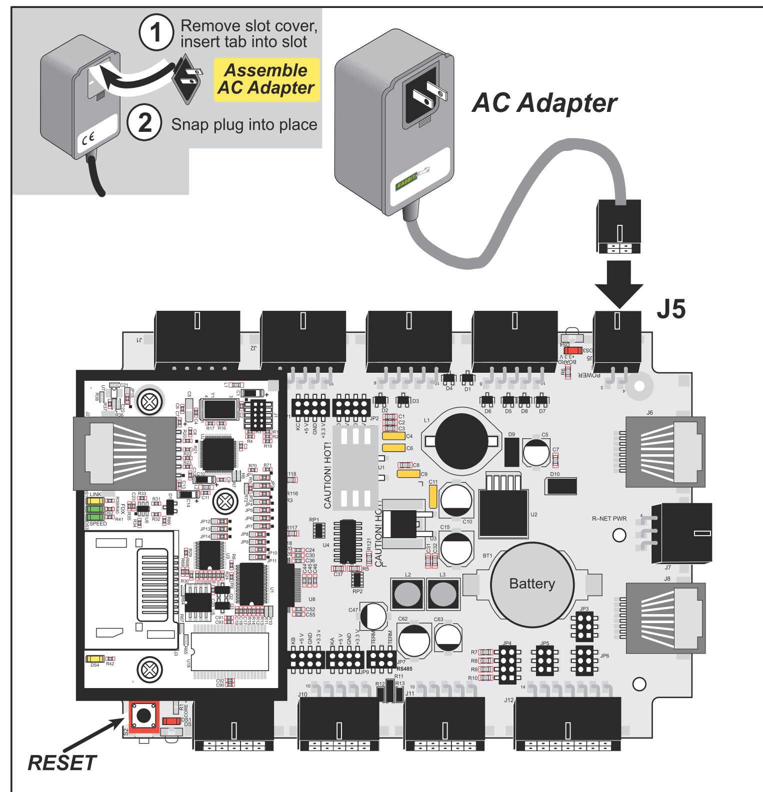


Figure 4. Power Supply Connections

3. Apply power.

Once all the other connections have been made, you may connect power to the BL4S200.

First, prepare the AC adapter for the country where it will be used by selecting the plug. The Tool Kit presently includes Canada/Japan/U.S., Australia/N.Z., U.K., and European style plugs. Snap in the top of the plug assembly into the slot at the top of the AC adapter as shown in Figure 4, then press down on the spring-loaded clip below the plug assembly to allow the plug assembly to click into place. Release the clip to secure the plug assembly in the AC adapter.

Plug in the AC adapter. The red LED next to the power connector at J5 should light up. The BL4S200 is now ready to be used.

> **CAUTION:** Unplug the power supply while you make or otherwise work with the connections to the headers. This will protect your BL4S200 from inadvertent shorts or power spikes.

2.2.1 Hardware Reset

A hardware reset is done by unplugging the power supply, then plugging it back in, or by pressing the **RESET** button located just below the RabbitCore module.

2.3 Installing Dynamic C

If you have not yet installed Dynamic C version 10.42 (or a later version), do so now by inserting the Dynamic C CD from the BL4S200 Tool Kit in your PC's CD-ROM drive. If autorun is enabled, the CD installation will begin automatically.

If autorun is disabled or the installation does not start, use the Windows **Start | Run** menu or Windows Disk Explorer to launch **`setup.exe`** from the root folder of the CD-ROM.

The installation program will guide you through the installation process. Most steps of the process are self-explanatory.

> **NOTE:** If you have an earlier version of Dynamic C already installed, the default installation of the later version will be in a different folder, and a separate icon will appear on your desktop.

The online documentation is installed along with Dynamic C, and an icon for the documentation menu is placed on the workstation's desktop. Double-click this icon to reach the menu. If the icon is missing, create a new desktop icon that points to **default.htm** in the **docs** folder, found in the Dynamic C installation folder. The latest versions of all documents are always available for free, unregistered download from our Web sites as well.

The ***Dynamic C User's Manual*** provides detailed instructions for the installation of Dynamic C and any future upgrades.

Once your installation is complete, you will have up to three icons on your PC desktop. One icon is for Dynamic C, one opens the documentation menu, and the third is for the Rabbit Field Utility, a tool used to download precompiled software to a target system.

If you have purchased any of the optional Dynamic C modules, install them after installing Dynamic C. The modules may be installed in any order. You must install the modules in the same directory where Dynamic C was installed.

2.4 Starting Dynamic C

Once the BL4S200 is connected to your PC and to a power source, start Dynamic C by double-clicking on the Dynamic C icon on your desktop or in your **Start** menu. Select **Store Program in Flash** on the "Compiler" tab in the Dynamic C **Options > Project Options** menu. Then click on the "Communications" tab and verify that **Use USB to Serial Converter** is selected to support the USB programming cable. Click **OK**.

You may have to select the COM port assigned to the USB programming cable on your PC. In Dynamic C, select **Options > Project Options**, then select this COM port on the "Communications" tab, then click **OK**. You may type the COM port number followed by **Enter** on your computer keyboard if the COM port number is outside the range on the dropdown menu.

2.5 Run a Sample Program

You are now ready to test your set-up by running a sample program.

Use the **File** menu to open the sample program **`PONG.C`**, which is in the Dynamic C **`SAMPLES`** folder. Press function key **F9** to compile and run the program. The **STDIO** window will open on your PC and will display a small square bouncing around in a box.

This program shows that the CPU is working. The sample program described in Section 5.2.3, "Run the PINGME.C Demo," tests the TCP/IP portion of the board.

2.5.1 Troubleshooting

If you receive the message **`No Rabbit Processor Detected`**, the programming cable may be connected to the wrong COM port, a connection may be faulty, or the target system may not be powered up. First, check to see that the red power LED next to header J5 is lit. If the LED is lit, check both ends of the programming cable to ensure that it is firmly plugged into the PC and the programming header on the BL4S200 with the marked (colored) edge of the programming cable towards pin 1 of the programming header. Ensure that the module is firmly and correctly installed in its connectors on the BL4S200 board.

If Dynamic C appears to compile the BIOS successfully, but you then receive a communication error message when you compile and load a sample program, it is possible that your PC cannot handle the higher program-loading baud rate. Try changing the maximum download rate to a slower baud rate as follows.

- Locate the **Serial Options** dialog on the "Communications" tab in the Dynamic C **Options > Project Options** menu. Select a slower Max download baud rate. Click **OK** to save.

If a program compiles and loads, but then loses target communication before you can begin debugging, it is possible that your PC cannot handle the default debugging baud rate. Try lowering the debugging baud rate as follows.

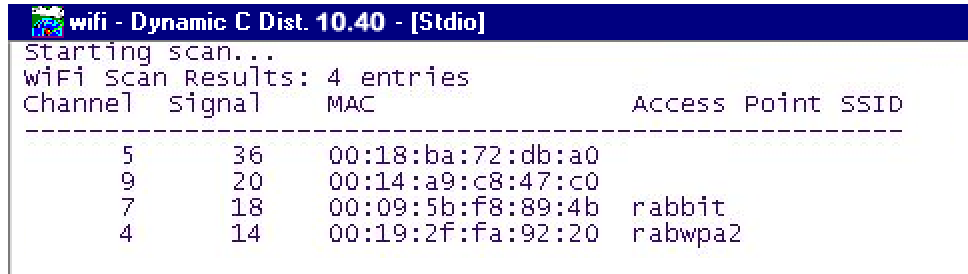
- Locate the **Serial Options** dialog on the "Communications" tab in the Dynamic C **Options > Project Options** menu. Choose a lower debug baud rate. Click **OK** to save.

Press **<Ctrl-Y>** to force Dynamic C to recompile the BIOS. You should receive a **`Bios compiled successfully`** message once this step is completed successfully.

2.6 Run a Wi-Fi Sample Program (BL5S220 only)

Find the **`WIFISCAN.C`** sample program in the Dynamic C **`Samples\WiFi`** folder, open it with the **File** menu, then compile and run the sample program by pressing **F9**.

The Dynamic C **STDIO** window will display **`Starting scan...`**, and will display a list of access points/ad-hoc hosts as shown here.



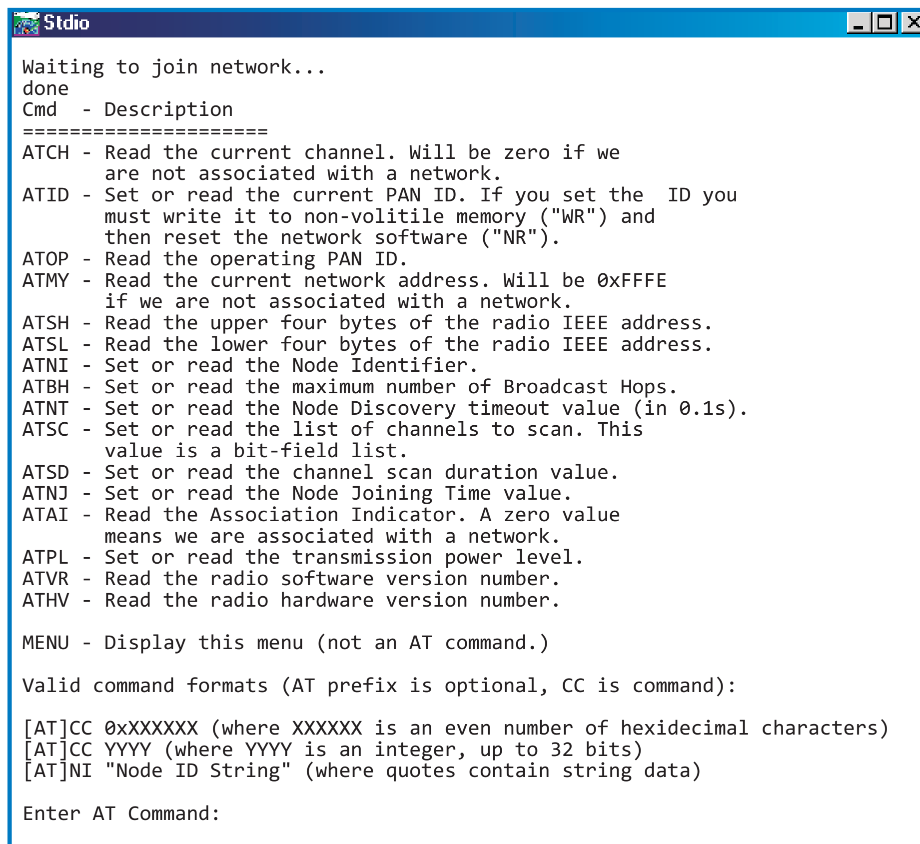
The following fields are shown in the Dynamic C **STDIO** window.

- Channel—the channel the access point is on (1–11).
- Signal—the signal strength of the access point.
- MAC—the hardware (MAC) address of access point.
- Access Point SSID—the SSID the access point is using.

2.7 Run a ZigBee Sample Program (BL4S230 only)

This section explains how to run a sample program in which the BL4S230 is used in its default setup as a router and the Digi® XBee USB is used as the ZigBee coordinator.

1. Connect the Digi® XBee USB acting as a ZigBee coordinator to an available USB port on your PC or workstation. Your PC should recognize the new USB hardware.
2. Find the file **`AT_INTERACTIVE.C`**, which is in the Dynamic C **`SAMPLES\XBee`** folder. To run the program, open it with the **File** menu, then compile and run it by pressing **F9**. The Dynamic C **STDIO** window will open to display a list of AT commands. Type **`MENU`** to redisplay the menu of commands.



Appendix F provides additional configuration information if you experience conflicts while doing development simultaneously with more than one ZigBee coordinator, or if you wish to upload new firmware.

2.8 Where Do I Go From Here?

NOTE: If you purchased your BL4S200 through a distributor or Rabbit partner, contact the distributor or partner first for technical support.

If there are any problems at this point:

- Use the Dynamic C **Help** menu to get further assistance with Dynamic C.
- Check the Rabbit Technical Bulletin Board and forums at www.digi.com/support/ and at www.digi.com/support/forums/.
- Use the Technical Support e-mail form at www.digi.com/support/.

If the sample program ran fine, you are now ready to go on to explore other BL4S200 features and develop your own applications.

When you start to develop your application, run **USERBLOCK_READ_WRITE.C** in the **SAMPLES\UserBlock** folder to save the factory calibration constants before you run any other sample programs in case you inadvertently write over them while running another sample program.

Chapter 3, "Subsystems," provides a description of the BL4S200's features, Chapter 4, "Software," describes the Dynamic C software libraries and introduces some sample programs, and Chapter 5, "Using the Ethernet TCP/IP Features," explains the TCP/IP features.

3. Subsystems

Chapter 3 describes the principal subsystems for the BL4S200.

- Digital I/O
- Serial Communication
- A/D Converter Inputs
- D/A Converter Outputs
- Analog Reference Voltages Circuit
- Memory

Figure 5 shows these Rabbit-based subsystems designed into the BL4S200.

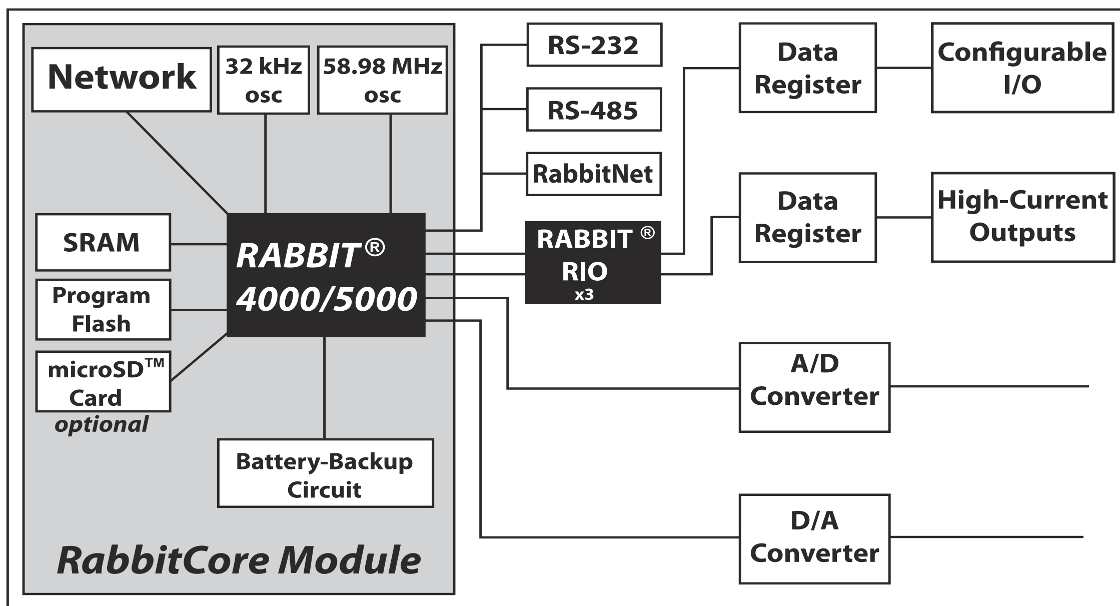


Figure 5. BL4S200 Subsystems

3.1 BL4S200 Pinouts

The BL4S200 pinouts are shown in Figure 6.

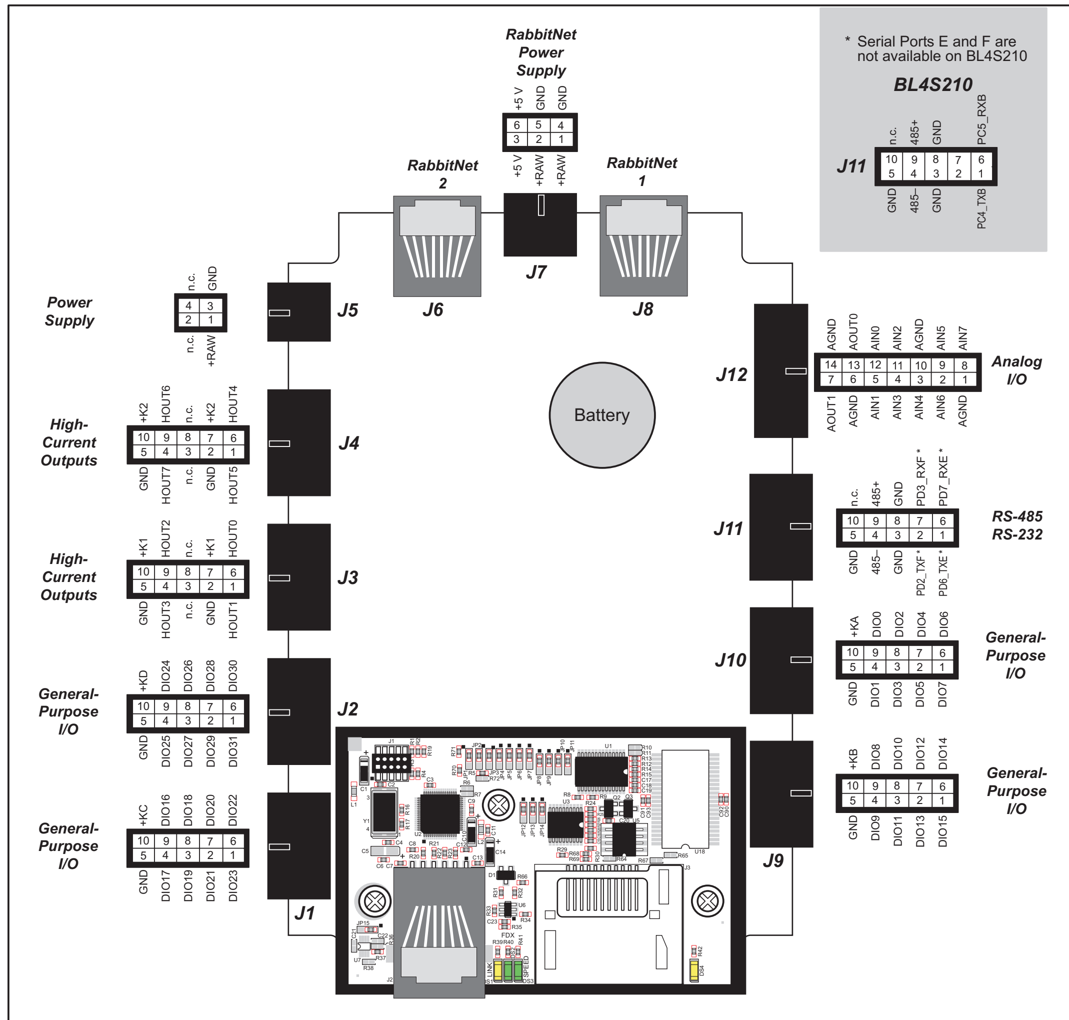


Figure 6. BL4S200 Pinouts

3.1.1 Connectors

Standard BL4S200 models are equipped with seven polarized 2 × 5 Micro-Fit® connectors (J1–J4 and J9–J11), one polarized 2 × 7 Micro-Fit® connector (J12), and one polarized 2 × 3 connector at J7 to supply power (DCIN and +5 V) to up to two RabbitNet peripheral expansion boards. The polarized 2 × 2 Micro-Fit® connector at J5 is for the main power supply connections.

The RJ-45 jacks at J6 and J8 labeled ***RabbitNet*** are serial I/O expansion ports for use with RabbitNet peripheral expansion boards. The ***RabbitNet*** jacks do ***not*** support Ethernet connections. Be careful to make your Ethernet connection to the ***Ethernet*** jack on the RabbitCore module (note that the wireless BL5S220 and BL4S230 models do not have an Ethernet port).

Table 2 lists Molex connector part numbers for the crimp terminals, and housings needed to assemble male Micro-Fit® connector assemblies for use with their female counterparts on the BL4S200.

Table 2. Male Micro-Fit® Connector Parts

Micro-Fit® Connector	Used with BL4S200 connectors	Molex Housing Part Number	Molex Crimp Terminals
3 mm 2 × 2	J5	0430250400	0430300001 (bronze contacts) 0430300007 (tin/brass contacts)
3 mm 2 × 3	J7	0430250600	
3 mm 2 × 5	J1–J4, J9–J11	0430251000	
3 mm 2 × 7	J12	0430251400	

3.2 Digital I/O

3.2.1 Configurable I/O

3.2.1.1 Digital Inputs

The BL4S200 has 32 configurable I/O, DIO0–DIO31, each of which may be configured individually in software as either digital inputs or as sinking digital outputs. By default, a configurable I/O channel is a digital input, but may be set as a sinking digital output by using the `setDigOut()` function call. The inputs are factory-configured to be pulled up to +5 V, but they can also be pulled up to +K or DCIN, or pulled down to 0 V in banks by changing a jumper as shown in Figure 7.

CAUTION: Do not simultaneously jumper more than one setting on a particular jumper header (JP9, JP8, JP1, and JP2) when configuring a bank of configurable I/O.

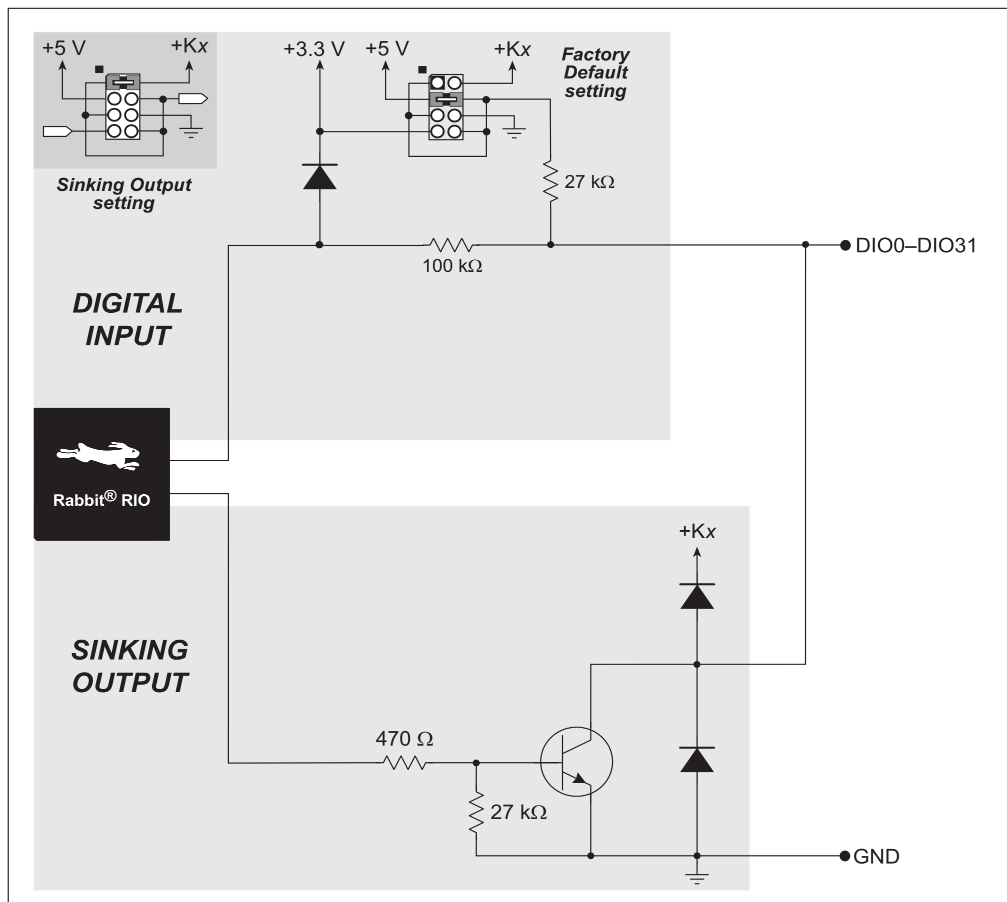


Figure 7. BL4S200 Configurable I/O DIO0–DIO31

Table 3 lists the banks of configurable I/O and summarizes the jumper settings.

Table 3. Banks of BL4S200 Digital Inputs

Digital Inputs	Configuration Header	Pins Jumpered	Pulled Up/Pulled Down
DIO0–DIO7	JP9	1–2	Inputs pulled up to +K*x*
DIO8–DIO15	JP8	3–4	Inputs pulled up to +5 V
DIO16–DIO23	JP1	5–6	Inputs pulled down to GND
DIO24–DIO31	JP2	7–8	Inputs pulled up to + 3.3 V

The actual switching threshold is approximately 1.40 V. Anything below this value is a logic 0, and anything above 1.90 V is a logic 1. The configurable I/O are each fully protected over a range of 0 V to +36 V, and can handle short spikes from -5 V to +40 V.

NOTE: If the inputs are pulled up to +K*x*, the voltage range over which the digital inputs are protected changes to 5 V - K*x* to +36 V.

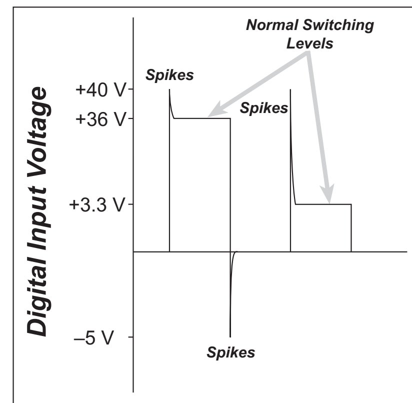
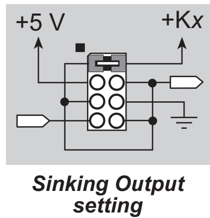


Figure 8. BL4S200 Digital Input Protected Range

CAUTION: Do not allow the voltage on a configurable I/O pin to exceed +K*x* to avoid damaging the input.

3.2.1.2 Sinking Digital Outputs

When you configure a configurable I/O pin as a sinking output, be sure to connect an external voltage source up to 36 V DC across the corresponding +K*x* and GND on connector J1, J2, J9, or J10, and set the pullup jumper on the corresponding JP1/JP2/JP8/JP9 header to +K*x*.



Sinking Output setting

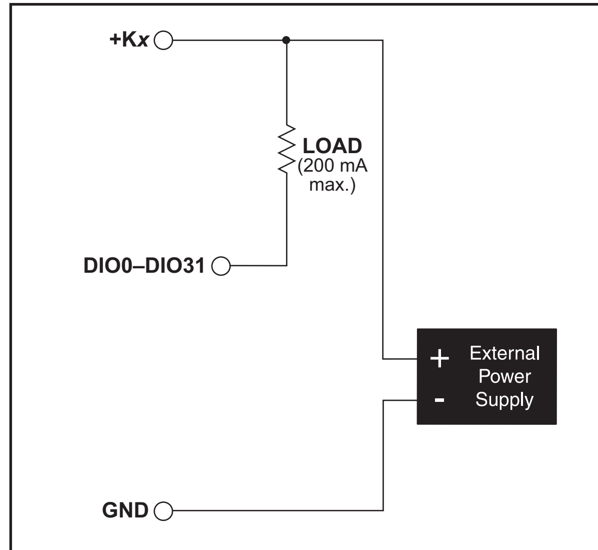


Figure 9. Load and +K Power Supply Connections for Sinking Digital Output

Table 4 lists the banks of configurable I/O and the corresponding +K*x*.

Table 4. BL4S200 Sinking Outputs

Digital Inputs	+K*x*	Configuration Header	Pins Jumpered	Pulled Up/Pulled Down
DIO0–DIO7	KA on J10	JP9	1–2	I/O pulled up to +K*x*
DIO8–DIO15	KB on J9	JP8	3–4	Do not use these options for a sinking output.
DIO16–DIO23	KC on J1	JP1	5–6	
DIO24–DIO31	KD on J2	JP2	7–8	

CAUTION: Do not simultaneously jumper more than one setting on a particular jumper header (JP9, JP8, JP1, and JP2) when configuring a bank of configurable I/O.

CAUTION: Do not allow the voltage on a configurable I/O pin to exceed +K*x* to avoid damaging the input.

3.2.1.3 Configurable I/O Special Uses

Individual configurable I/O pins may be used for interrupts, input capture, as quadrature decoders, or as PWM outputs. The use of these channels for PWM, interrupts, input capture, and as quadrature decoders is described in the ***Rabbit RIO User's Manual***.

Blocks of configurable I/O pins are associated with counters/timers on the three Rabbit RIO chips that support them. Table 5 provides complete details for these associations.

Table 5. Counter/Timer Associations for BL4S200 Configurable I/O Pins

Configurable I/O Pin(s)	Counter/Timer Blocks	RIO Chip Index
DIO0–DIO3	4 (outputs) 5 (inputs)	0 (U8)
DIO4–DIO7	0 (outputs) 1 (inputs)	1 (U7)
DIO8–DIO11	2 (outputs) 3 (inputs)	1 (U7)
DIO12–DIO15	4 (outputs) 5 (inputs)	1 (U7)
DIO16–DIO17	0 (I/O)	2 (U9)
DIO18–DIO19	1 (I/O)	2 (U9)
DIO20–DIO21	2 (I/O)	2 (U9)
DIO22–DIO23	3 (I/O)	2 (U9)
DIO24–DIO25	4 (I/O)	2 (U9)
DIO26–DIO27	5 (I/O)	2 (U9)
DIO28	6 (output) 7 (input)	0 (U8)
DIO29	6 (output) 7 (input)	1 (U7)
DIO30	6 (input only)	2 (U9)
DIO31	7 (input only)	2 (U9)

Configurable I/O pins DIO30 and DIO31 fully support all input-associated special uses such as interrupts and input captures, but otherwise they are limited to function only as regular digital I/O pins because their outputs are latch-driven since sufficient Rabbit RIO resources are not available to support their use for specialized outputs.

Appendix D provides further details on the blocks and pins associated with each Rabbit RIO chip to facilitate configuring each block consistently and to identify misconfigured pins when a software function call returns a *Mode Conflict* error code.

Keep the following guidelines in mind when selecting special uses for the remaining configurable I/O pins.

- Interrupts, event counters, and input capture are available on any configurable I/O pin.
- Each Quadrature Decoder channel requires at least two configurable I/O pins associated with the same counter/timer block; three configurable I/O pins associated with the same counter/timer block are needed if you need indexing.
- When using configurable I/O pins for PWM outputs, they can only share the same RIO block if they are using the same period or frequency. Depending on the pin(s) selected, from one to four PWM outputs could operate based on the same counter block. Remember to set the corresponding jumper (Table 4) so that the I/O for that bank are pulled up to the selected voltage. The output voltage swing will be from 0 to the voltage you selected\.

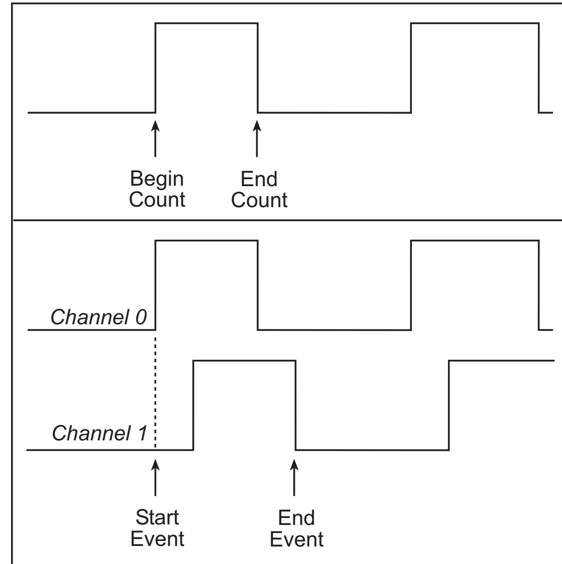
  The sample program **`PWM.C`** in the **`DIO`** subdirectory in **`SAMPLES\BLxS2xx`** shows how to set up and use the PWM outputs.
- Configurable I/O have their own set of function calls. These function calls will only work with configurable I/O. High-current outputs have their own function calls that end with **`_H`**.

See Appendix D for additional information about the Rabbit RIO pin associations and how to select which special functionality to best apply to a particular pin.

Interrupt, Counter, and Event Capture Setup

External interrupts on the BL4S200 configurable I/O pins are configured using the **`setExtInterrupt()`** function call. The interrupt can be set up to occur on a rising edge, a falling edge, or either edge.

An input channel may be set up to count events, with the count incrementing or decrementing, using the rising edge, falling edge, or either edge as triggers to start/end the count. This feature is configured using the **`setCounter()`** function call.



A more extensive use of the timing abilities of the BL4S200 configurable I/O can be realized through the event capture function call, **`setCapture()`**. Here the count of a particular clock cycle is noted at the start of the event and at the end of the event so that the time between them can be determined. This can be set up on one or two configurable I/O channels. The event counter can be reset with the **`resetCounter()`** function call.

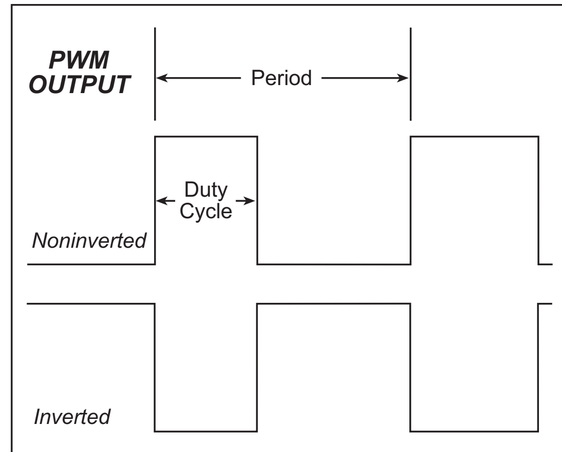
The counter readings can be obtained via the **`getBegin()`** or **`getEnd()`** function calls.

PWM/PPM Outputs Setup

A PWM output is described as *noninverted* when it starts high, remains high for a duty cycle that is a fraction of the period, then goes low for the remainder of the period.

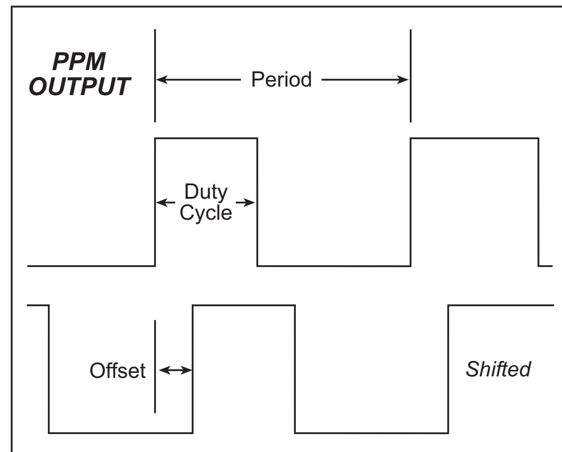
Similarly, an *inverted* PWM output starts low, remains low for a duty cycle that is a fraction of the period, then goes high for the remainder of the period.

A PWM output is normally set up to start when triggered by an event, and may be set up so that the leading and trailing edges of several PWM outputs are aligned as long as the all the PWM outputs are on the same block of a particular Rabbit RIO chip.



A PPM ouput is similar to a PWM output, except it is *shifted* by an *offset* relative to the event that triggered the start of the PPM output.

A PPM output is either inverted or noninverted, based on whether it starts high or low, and may be set up so that their leading and trailing edges of several PPM outputs are aligned as long as the all the PPM outputs are on the same block of a particular Rabbit RIO chip



PWM and PPM outputs on the BL4S200 configurable I/O are configured using the **`setPWM()`** and **`setPPM()`** function calls. PWM and PPM outputs on the BL4S200 high-current outputs are configured using the **`setPWM_H()`** and **`setPPM_H()`** function calls.

3.2.2 High-Current Digital Outputs

The BL4S200 has eight high-current digital outputs, HOUT0–HOUT7, which can each sink or source up to 2 A. Figure 10 shows a wiring diagram for using the digital outputs in either a sinking or a sourcing configuration.

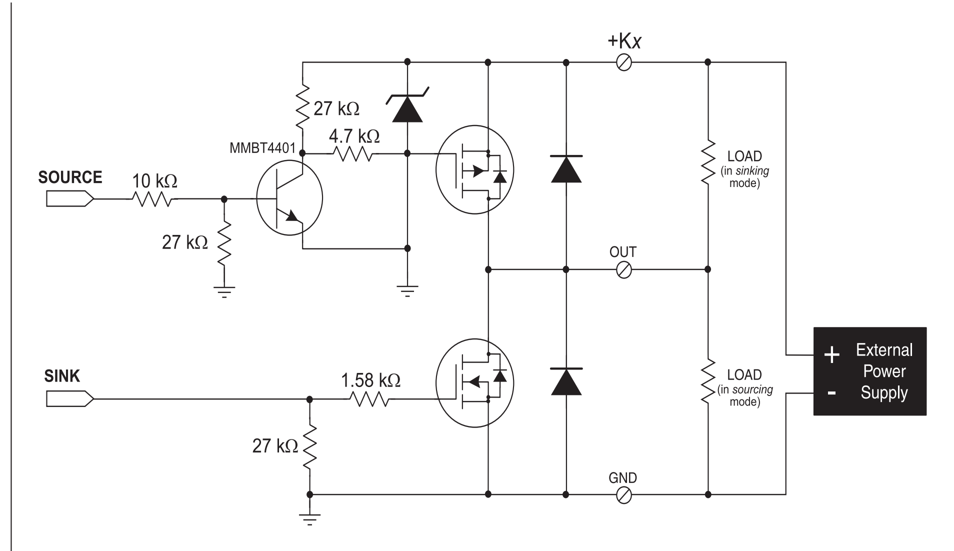


Figure 10. BL4S200 High-Current Outputs

All the digital outputs sink and source actively. They can be used as high-side drivers, low-side drivers, or as an H-bridge driver. When the BL4S200 is first powered up or reset, all the outputs are disabled, that is, at a high-impedance tristate.

Each bank of four high-current output has its own +K supply, as shown in Table 6. When wiring the high-current outputs, keep the distance to the power supply as short as possible.

Table 6. BL4S200 High-Current Outputs

High-Current Outputs	+K*x*	Connector
HOUT0–HOUT3	K1	J3
HOUT4–HOUT7	K2	J4

For the H bridge, which is shown in Figure 11, Ka and Kb ***should be the same***. This is most easily accomplished by using outputs from the same bank on one connector.

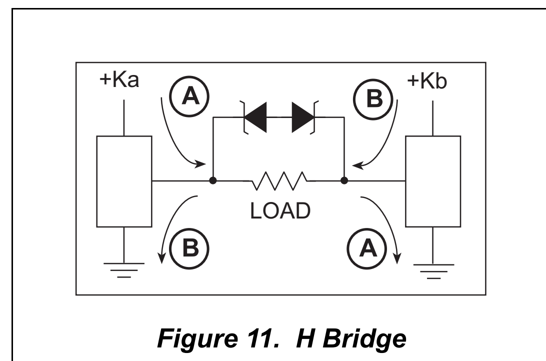


Figure 11. H Bridge

High-current outputs have their own function calls for control (**digOut_H()** and **digOutTriState_H()**) and to set up the PWM and PPM outputs. All function calls that work with high-current outputs end with **_H** — do not confuse these function calls with their configurable I/O counterparts. The **digOutConfig_H()** function call configures the high-current outputs as two state outputs with either sinking or sourcing drivers. The **digOutTriStateConfig_H()** function call configures the high-current outputs as tristate drivers with both sinking and sourcing capability.

3.3 Serial Communication

The BL4S200 has up to three serial communication ports, one RS-485 channel, and either one RS-232 serial channel (with RTS/CTS) or two RS-232 (3-wire) channels. Table 7 summarizes the serial ports.

Table 7. Serial Communication Configurations

BL4S200 Model	Serial Port			
	B	C	E	F
BL4S200	—	RS-485	RS-232 (PD6/PD7)	RS-232 (PD2/PD3)
BL4S210	RS-232	RS-485	—	—
BL5S220	—	RS-485	RS-232 (PD6/PD7)	RS-232 (PD2/PD3)
BL4S230	—	RS-485	RS-232 (PD6/PD7)	RS-232 (PD2/PD3)

Two RabbitNet™ expansion ports are multiplexed from Serial Port D. The BL4S200 also has one CMOS serial channel that serves as the programming port.

All three serial ports operate in an asynchronous mode. An asynchronous port can handle 7 or 8 data bits. A 9th bit address scheme, where an additional bit is sent to mark the first byte of a message, is also supported. Serial Port A, the programming port, can be operated alternately in the clocked serial mode. In this mode, a clock line synchronously clocks the data in or out. Either of the two communicating devices can supply the clock. The BL4S200 boards supports standard asynchronous baud rates from 3.7 Mbps to 9.2 Mbps, depending on the frequency the Rabbit microprocessor on a particular model is operating at.

3.3.1 RS-232

The BL4S200 RS-232 serial communication is supported by an RS-232 transceiver. This transceiver provides the voltage output, slew rate, and input voltage immunity required to meet the RS-232 serial communication protocol. Basically, the chip translates the Rabbit microprocessor's CMOS signals to RS-232 signal levels. Note that the polarity is reversed in an RS-232 circuit so that a +3.3 V output becomes approximately -10 V and 0 V is output as +10 V. The RS-232 transceiver also provides the proper line loading for reliable communication.

RS-232 can be used effectively at the BL4S200's maximum baud rate for distances of up to 15 m.

3.3.2 RS-485

The BL4S200 has one two-wire RS-485 serial channel, which is connected to Serial Port C through an RS-485 transceiver. This port operates in a half-duplex communication mode, which requires directional control on the communication line.

The BL4S200 can be used in an RS-485 multidrop network. Connect the 485+ to 485+ and 485– to 485– using single twisted-pair wires (nonstranded, tinned) as shown in Figure 12. Note that a common ground is recommended.

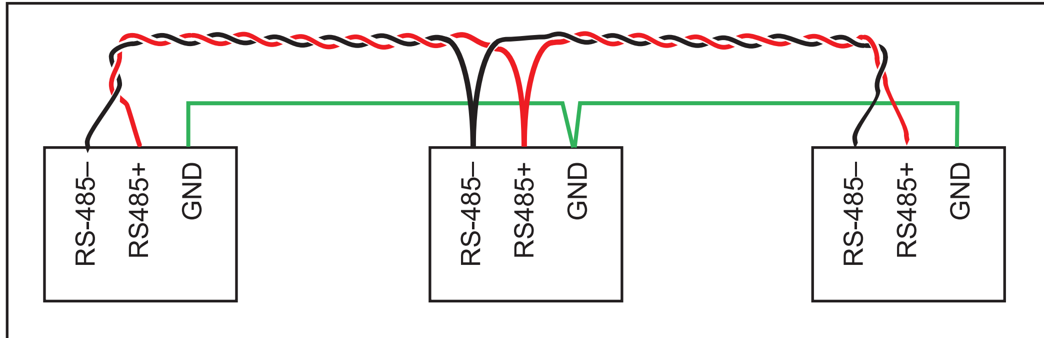


Figure 12. BL4S200 Multidrop Network

The BL4S200 comes with a 220 Ω termination resistor and two 681 Ω bias resistors installed and enabled with jumpers across pins 1–2 and 5–6 on header JP7, as shown in Figure 13.

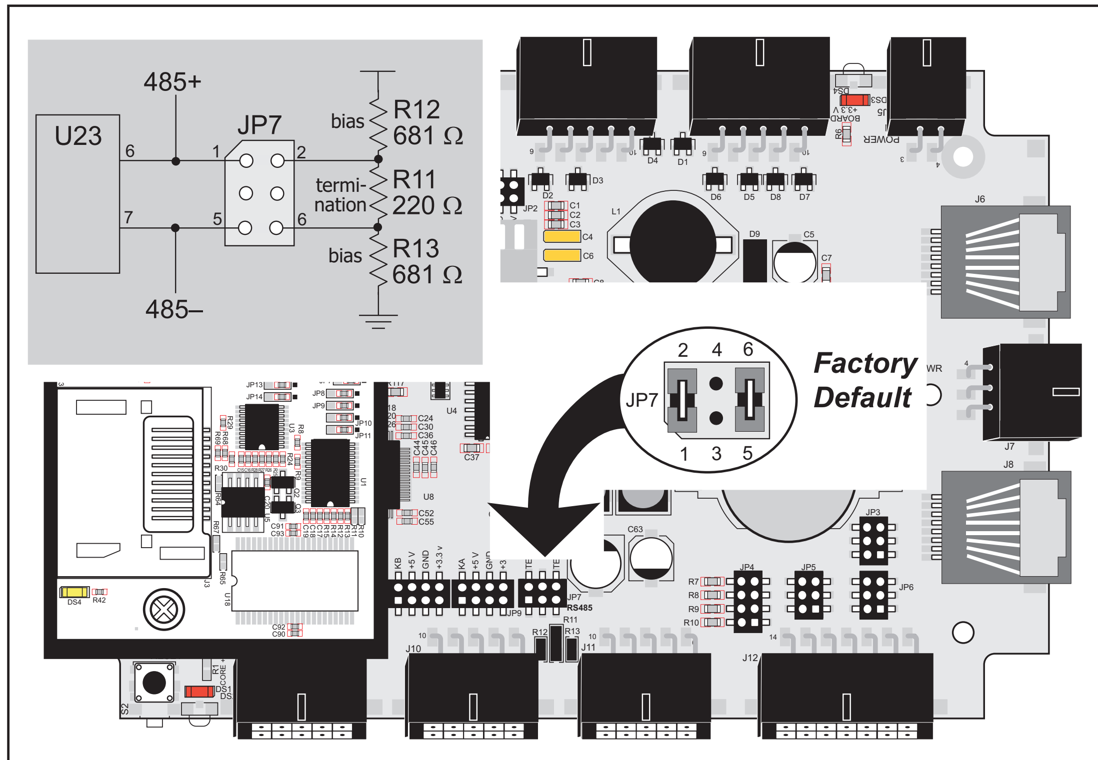


Figure 13. RS-485 Termination and Bias Resistors

For best performance, the bias and termination resistors in a multidrop network should only be enabled on both end nodes of the network. Disable the termination and bias resistors on any intervening BL4S200 units in the network by removing both jumpers from header JP6.

> **TIP:** Save the jumpers for possible future use by "parking" them across pins 1–3 and 4–6 of header JP7. Pins 3 and 4 are not otherwise connected to the BL4S200.

3.3.3 Programming Port

The RabbitCore module on the BL4S200 has a 10-pin programming header. The programming port uses the Rabbit 4000 or Rabbit 5000 Serial Port A for communication, and is used for the following operations.

- Programming/debugging
- Cloning

The programming port is used to start the BL4S200 in a mode where the BL4S200 will download a program from the port and then execute the program. The programming port transmits information to and from a PC while a program is being debugged.

The Rabbit 4000 or Rabbit 5000 startup-mode pins (SMODE0, SMODE1) are presented to the programming port so that an externally connected device can force the BL4S200 to start up in an external bootstrap mode. The BL4S200 can be reset from the programming port via the **/EXT_RSTIN** line.

The Rabbit microprocessor status pin is also presented to the programming port. The status pin is an output that can be used to send a general digital signal.

> **NOTE:** Refer to the ***Rabbit 4000 Microprocessor User's Manual*** and the ***Rabbit 5000 Microprocessor User's Manual*** for more information related to the bootstrap mode.

3.3.4 Ethernet Port

Figure 14 shows the pinout for the Ethernet port (J2 on the BL4S200 modules that support Ethernet networking). Note that there are two standards for numbering the pins on this connector—the convention used here, and numbering in reverse to that shown. Regardless of the numbering convention followed, the pin positions relative to the spring tab position (located at the bottom of the RJ-45 jack in Figure 14) are always absolute, and the RJ-45 connector will work properly with off-the-shelf Ethernet cables.

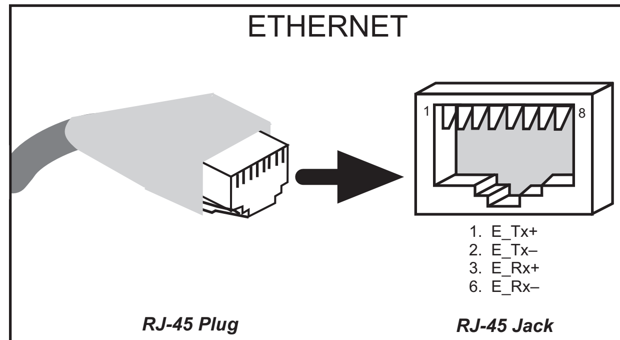


Figure 14. RJ-45 Ethernet Port Pinout

Three LEDs are placed next to the RJ-45 Ethernet jack on the BL4S200 model, one to indicate Ethernet link/activity (**LINK/ACT**), one to indicate when the BL4S200 is connected to a functioning 100Base-T network (**SPEED**), and one (**FDX/COL**) to indicate that the current connection is in full-duplex mode (steady on) or that a half-duplex connection is experiencing collisions (blinks).

Two LEDs are placed next to the RJ-45 Ethernet jack on the BL4S210 model, one to indicate an Ethernet link (**LNK**) and one to indicate Ethernet activity (**ACT**).

The RJ-45 connector is shielded to minimize EMI effects to/from the Ethernet signals.

3.4 A/D Converter Inputs

The single A/D converter chip used in the BL4S200 has a resolution of 12 bits (11 bits for the value and one bit for the polarity). The A/D converter chip has a programmable amplifier. Each external input has circuitry that provides scaling and filtering. All 8 external inputs are scaled and filtered to provide the user with an input impedance of 1 MΩ and a variety of single-ended unipolar, single-ended bipolar, and differential bipolar ranges as shown in Table 8.

Figure 15 shows a pair of A/D converter input circuits. The resistors form an approx. 10:1 attenuator, and the capacitors filter noise pulses from the A/D converter inputs.

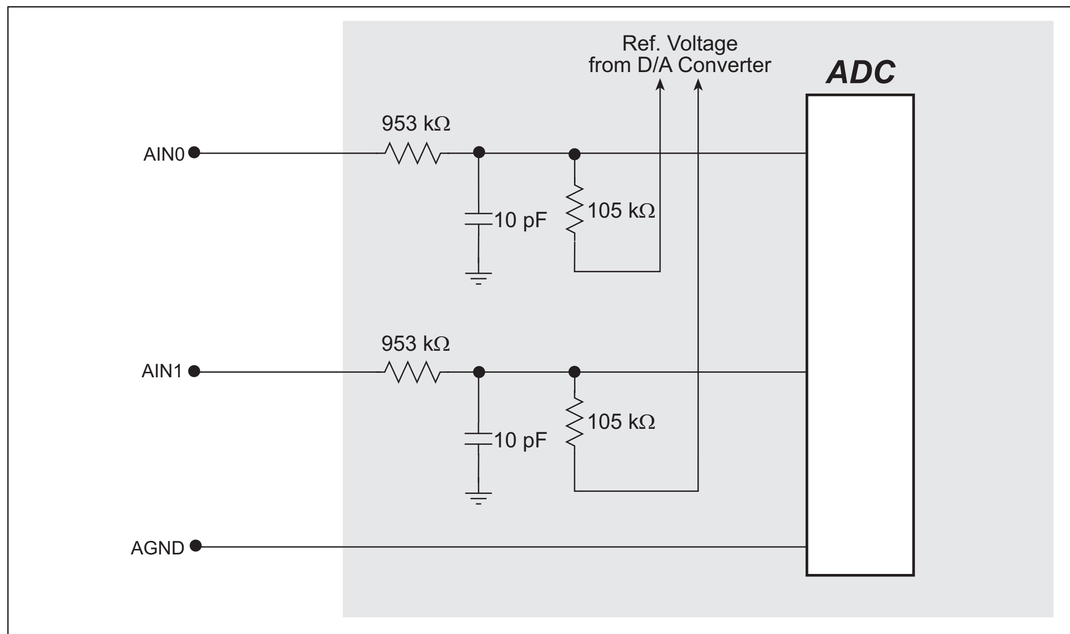


Figure 15. Buffered A/D Converter Inputs

The A/D converter chip can only accept positive voltages. By pairing the analog inputs and setting the reference voltage from the D/A converter, single-ended unipolar, single-ended bipolar, differential bipolar, or current (4–20 mA on channels 0–3 only) measurements are possible, and can be configured for each channel or channel pair with the **`opmode`** parameter in the **`anaInConfig()`** software function call. Adjacent A/D converter inputs AIN4–AIN7are paired to make bipolar measurements. The available voltage ranges are listed in Table 8.

Table 8. A/D Converter Input Voltage Ranges

Amplifier Gain	Voltage Range		
	Single-Ended Unipolar	Single-Ended Bipolar	Differential Bipolar
1	0–20 V	±10 V	± 20 V
2	0–10 V	±5 V	± 10 V
4	0–5 V	±2.5 V	± 5 V
5	0–4 V	±2 V	± 4 V
8*	0–2.5 V	±1.25 V	± 2.5 V
10	0–2 V	±1 V	± 2 V
16	0–1.25 V	±0.625 V	± 1.25 V
20	0–1 V	±0.5 V	± 1 V

* 4–20 mA operation is available with an amplifier gain of 8

In the differential mode, each individual channel is limited to half the total voltage—for example, the range for a gain code of 1 is ±20 V, but each channel is limited to ±10 V.

When using channels AIN0–AIN3 for current measurements, remember to set the corresponding jumper(s) on header JP4. The current measurements are realized by actually measuring the voltage drop across a 100 Ω resistor. You may substitute a different resistor value as shown in Figure 16.

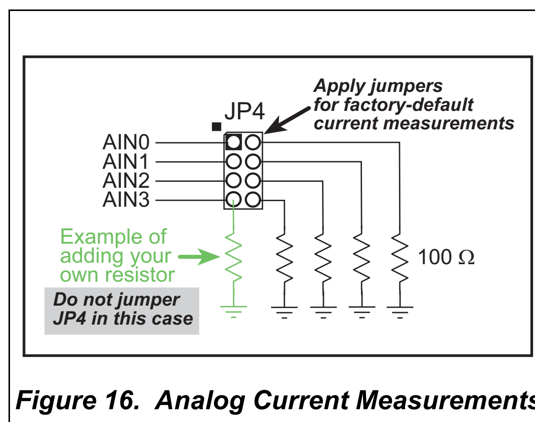


Figure 16. Analog Current Measurements

CAUTION: If you are using a supply voltage of +10 V DC for the 4–20 mA current measurements, do not exceed 500 Ω for these resistors.

The A/D converter inputs are factory-calibrated, and the calibration constants are stored in the user block.

When you start to develop your application, run **`USERBLOCK_READ_WRITE.C`** in the **`SAMPLES\UserBlock`** folder to save the factory calibration constants in case you inadvertently write over them while running the sample programs.

3.4.1 A/D Converter Calibration

To get the best results form the A/D converter, it is necessary to calibrate each mode (single-ended, differential, and current) for each of its gains. It is imperative that you calibrate each of the A/D converter inputs in the same manner as they are to be used in the application. For example, if you will be performing floating differential measurements or differential measurements using a common analog ground, then calibrate the A/D converter in the corresponding manner. The calibration table in software only holds calibration constants based on mode, channel, and gain. ***Other factors affecting the calibration must be taken into account by calibrating using the same mode and gain setup as in the intended use.***

Sample programs are provided to illustrate how to read and calibrate the various A/D inputs for the three operating modes.

Mode	Read	Calibrate
Single-Ended, unipolar	**AD_RD_SE_UNIPOLAR.C**	**ADC_CAL_SE_UNIPOLAR.C**
Single-Ended, bipolar	**AD_RD_SE_BIPOLAR.C**	**ADC_CAL_SE_BIPOLAR.C**
Differential, bipolar	**AD_RD_DIFF.C**	**ADC_CAL_DIFF.C**
Milli-Amp	**AD_RD_MA.C**	**ADC_CAL_MA.C**

These sample programs are found in the **ADC** subdirectory in **SAMPLES\BLxS2xx**. See Section 4.2.3 for more information on these sample programs and how to use them.

3.5 D/A Converter Outputs

The two D/A converter outputs are buffered and scaled to provide an output from 0 V to +10 V (12-bit resolution) or ±10 V (11-bit resolution, one bit used for polarity). The selection is made via jumpers on header JP3 for AOUT0 and via JP6 for AOUT1. There is also the option to select either D/A converter output as a 4–20 mA current output via jumpers on header JP5. Figure 17 shows the D/A converter outputs.

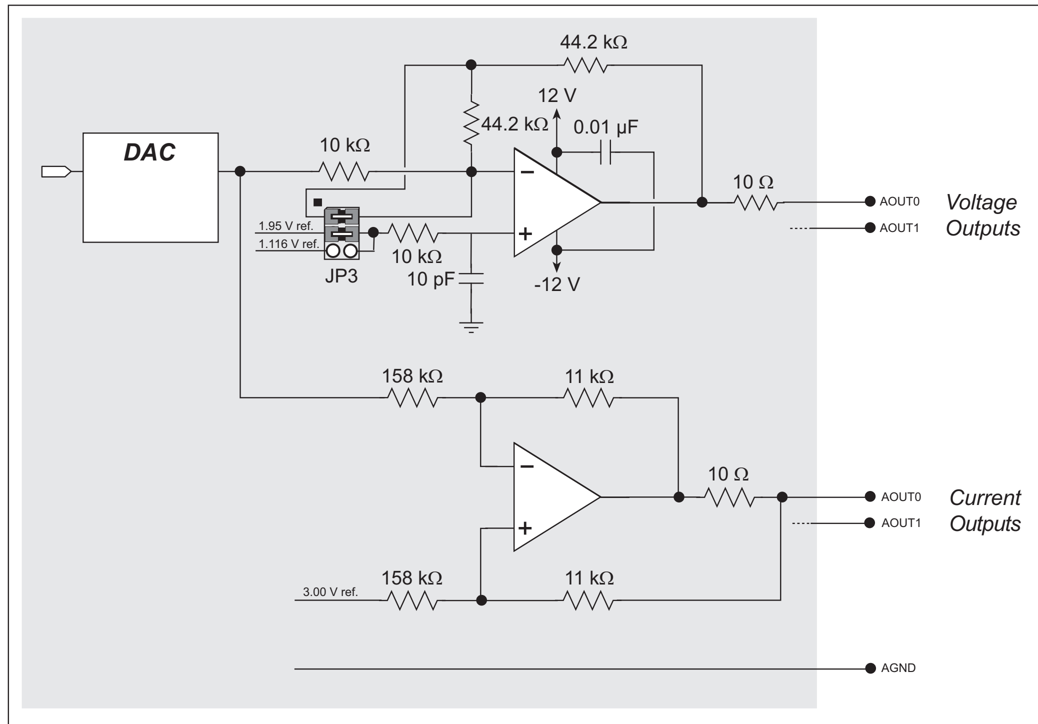


Figure 17. D/A Converter Outputs

Table 9 summarizes the jumper settings to configure each D/A converter output. Note that the software configuration requires both channels to be configured the same way.

Table 9. D/A Converter Jumper Configurations

D/A Converter Output		JP3	JP5	JP6
AOUT0	0 to +10 V (default)	1–2 3–4	1–3	—
	±10 V	5–6	1–3	—
	4–20 mA	—	3–5	—
AOUT1	0 to +10 V (default)	—	2–4	1–2 3–4
	±10 V	—	2–4	5–6
	4–20 mA	—	4–6	—

To stay within the maximum power dissipation of the D/A converter circuit, the maximum D/A converter output current is 10 mA per channel for the voltage outputs. If you are using the current outputs, keep the resistance driven by a current output channel above 1 kΩ to stay within the voltage compliance capability of the op-amp output circuit.

As Figure 17 shows, both the voltage and the current outputs for a particular channel are driven by the same output on the D/A converter chip. As a result, either the **anaOutVolts()** or the **anaOutmAmps()** function calls will set both the voltage and the current outputs corresponding to a particular channel. Pay attention to which function call you use to set the D/A converter output — setting a current level using voltage function calls will not produce a correctly calibrated output.

The D/A converter outputs are factory-calibrated and the calibration constants are stored in the user block.

3.5.1 D/A Converter Calibration

To get the best results form the D/A converter, it is necessary to calibrate each mode (unipolar, bipolar, and current) that you intend to use. It is imperative that you calibrate each of the D/A converter outputs in the same manner as they are to be used in the application. The calibration table in software only holds calibration constants based on unipolar, bipolar, and voltage or current operation. ***Other factors affecting the calibration must be taken into account by calibrating using the same mode and voltage/current setup as in the intended use.***

Sample programs are provided to illustrate how to calibrate the various D/A outputs for the three operating modes.

Mode	Calibrate
Voltage	**DAC_CAL_VOLTS.C**
Current	**DAC_CAL_MA.C**

These sample programs are found in the **DAC** subdirectory in **SAMPLES\BLxS2xx**. See Section 4.2.4 for more information on these sample programs and how to use them.

3.6 Analog Reference Voltages Circuit

Figure 18 shows the analog voltage reference circuit.

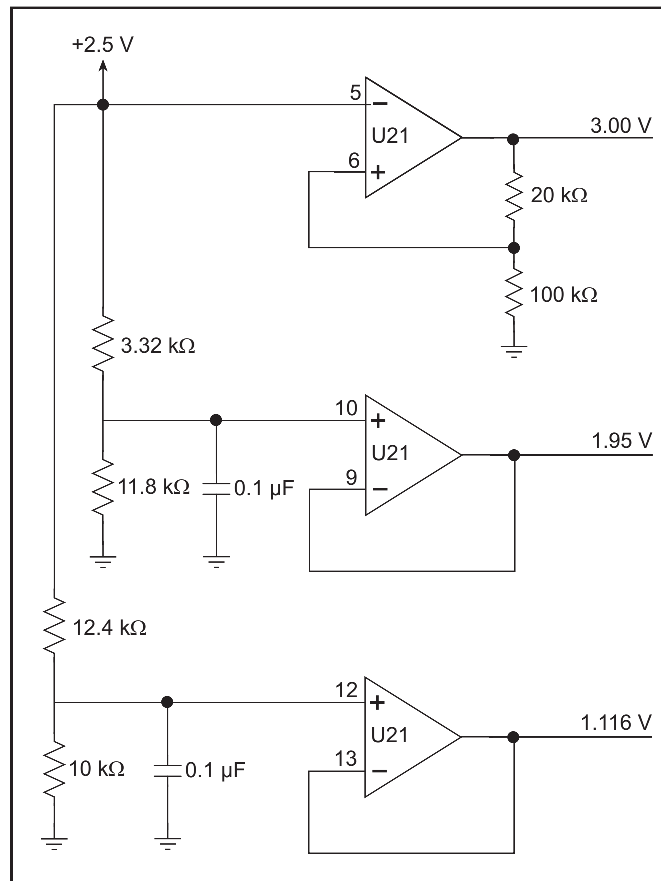


Figure 18. Analog Reference Voltages

The A/D converter chip supplies the 2.5 V reference voltage, which is then amplified and buffered to provide the 3.00 V, 1.95 V, and 1.116 V reference voltages used by the digital output circuits.

The D/A converter chip provides the reference voltages for the digital inputs to provide single-ended unipolar or differential measurements [0 V], or to provide single-ended bipolar measurements [V = (voltage range) ÷ 9]. Because the D/A converter chip operation is configured by the **`anaOutConfig()`** function, it is important to run the **`anaOutConfig()`** function before running **`anaInConfig()`** if you plan to use the analog outputs to ensure that the reference voltages are established first before the analog inputs are configured.

3.7 USB Programming Cable

The USB programming cable is used to connect the serial programming port of the BL4S200 to a PC USB port. The programming cable converts the voltage levels used by the PC USB port to the CMOS voltage levels used by the Rabbit microprocessor.

When the **PROG** connector on the programming cable is connected to the programming header on the BL4S200's RabbitCore module, programs can be downloaded and debugged over the serial interface.

The **DIAG** connector of the programming cable may be used on the programming header on the BL4S200's RabbitCore module with the BL4S200 operating in the Run Mode. This allows the programming port to be used as a regular serial port.

3.7.1 Changing Between Program Mode and Run Mode

The BL4S200 is automatically in Program Mode when the **PROG** connector on the programming cable is attached, and is automatically in Run Mode when no programming cable is attached. When the Rabbit microprocessor is reset, the operating mode is determined by the status of the SMODE pins. When the programming cable's **PROG** connector is attached, the SMODE pins are pulled high, placing the Rabbit microprocessor in the Program Mode. When the programming cable's **PROG** connector is not attached, the SMODE pins are pulled low, causing the Rabbit microprocessor to operate in the Run Mode.

Figure 19. BL4S200 Program Mode and Run Mode Setup

A program "runs" in either mode, but can only be downloaded and debugged when the BL4S200 is in the Program Mode.

Refer to the ***Rabbit 4000 Microprocessor User's Manual*** and the ***Rabbit 5000 Microprocessor User's Manual*** for more information on the programming port and the programming cable.

3.8 Other Hardware

3.8.1 Clock Doubler

The BL4S200 Ethernet models and the Wi-Fi model (BL4S200, BL4S210, and BL5S220) take advantage of the Rabbit microprocessor's internal clock doubler. A built-in clock doubler allows half-frequency crystals to be used to reduce radiated emissions. The clock doubler on the BL4S230 is disabled by default.

The clock doubler may be disabled on the BL4S200 Ethernet models (BL4S200 and BL4S210)if the higher clock speeds are not required. Disabling the clock doubler will reduce power consumption and further reduce radiated emissions. The clock doubler is disabled with a simple configuration macro as shown below.

1. Select the "Defines" tab from the Dynamic C **Options > Project Options** menu.
2. Add the line **`CLOCK_DOUBLED=0`** to always disable the clock doubler.

   The clock doubler is enabled by default, and usually no entry is needed. If you need to specify that the clock doubler is always enabled, add the line **`CLOCK_DOUBLED=1`** to always enable the clock doubler.
3. Click **OK** to save the macro. The clock doubler will now remain off whenever you are in the project file where you defined the macro.

> **NOTE:** Do not disable the clock doubler on the Wi-Fi model (BL5S220) since Wi-Fi operations depend highly on the CPU resources.

3.8.2 Spectrum Spreader

The Rabbit microprocessors features a spectrum spreader, which help to mitigate EMI problems. By default, the spectrum spreader is on automatically, but it may also be turned off or set to a stronger setting. The means for doing so is through a simple configuration macro as shown below.

1. Select the “Defines” tab from the Dynamic C **Options > Project Options** menu.
2. Normal spreading is the default, and usually no entry is needed. If you need to specify normal spreading, add the line

   ```
   ENABLE_SPREADER=1
   ```

   For strong spreading, add the line

   ```
   ENABLE_SPREADER=2
   ```

   To disable the spectrum spreader, add the line

   ```
   ENABLE_SPREADER=0
   ```

   NOTE: The strong spectrum-spreading setting is not recommended since it may limit the maximum clock speed or the maximum baud rate. It is unlikely that the strong setting will be used in a real application.

3. Click **OK** to save the macro. The spectrum spreader will be set according to the macro value whenever a program is compiled using this project file.

NOTE: The Rabbit RIO is driven by a 16Mhz Spread Spectrum Clock

Refer to the ***Rabbit 4000 Microprocessor User’s Manual*** and the ***Rabbit 5000 Microprocessor User’s Manual*** for more information on the spectrum-spreading settings and the maximum clock speed.

3.9 Memory

3.9.1 SRAM

The RabbitCore modules used with the BL4S200 boards all have 512 KB of data SRAM. The RabbitCore modules on the BL4S200 and BL5S220 boards also have 512 KB and 1 MB of fast program execution SRAM.

3.9.2 Flash Memory

The RabbitCore modules used with the BL4S200 boards have 512 KB or 1 MB of flash memory.

> **NOTE:** Rabbit recommends that any customer applications should not be constrained by the sector size of the flash memory since it may be necessary to change the sector size in the future.

Writing to arbitrary flash memory addresses at run time is also discouraged. Instead, define a "user block" area to store persistent data. The functions **`writeUserBlock()`** and **`readUserBlock()`** are provided for this.

3.9.3 VBAT RAM Memory

The tamper detection feature of the Rabbit microprocessor can be used to detect any attempt to enter the bootstrap mode. When such an attempt is detected, the VBAT RAM memory in the Rabbit microprocessor is erased. The serial bootloader on the RabbitCore module on the BL4S200 model uses the bootstrap mode to load the SRAM, which erases the VBAT RAM memory on any reset, and so it cannot be used on this model for tamper detection.

3.9.4 *microSD™* Cards

The RabbitCore module on the BL4S200 model supports a removable *microSD™ Card* up to 1 GB to store data and Web pages. The *microSD™ Card* is particularly suitable for mass-storage applications, but is generally unsuitable for direct program execution.

Unlike other flash devices, the *microSD™ Card* has some intelligence, which facilitates working with it. You do not have to worry about erased pages. All *microSD™ Cards* support 512-byte reads and writes, and handle any necessary pre-erasing internally.

Figure 20 shows how to insert or remove the *microSD™ Card*. The card is designed to fit easily only one way — do ***not*** bend the card or force it into the slot. While you remove or insert the card, take care to avoid touching the electrical contacts on the bottom of the card to prevent electrostatic discharge damage to the card and to keep any moisture or other contaminants off the contacts. You will sense a soft click once the card is completely inserted. To remove it, gently press the card towards the middle of the RabbitCore module on the BL4S200 model — you will sense a soft click and the card will be ready to be removed. Do not attempt to pull the card from the socket before pressing it in — otherwise the ejection mechanism will get damaged. The ejection mechanism is spring-loaded, and will partially eject the card when used correctly.

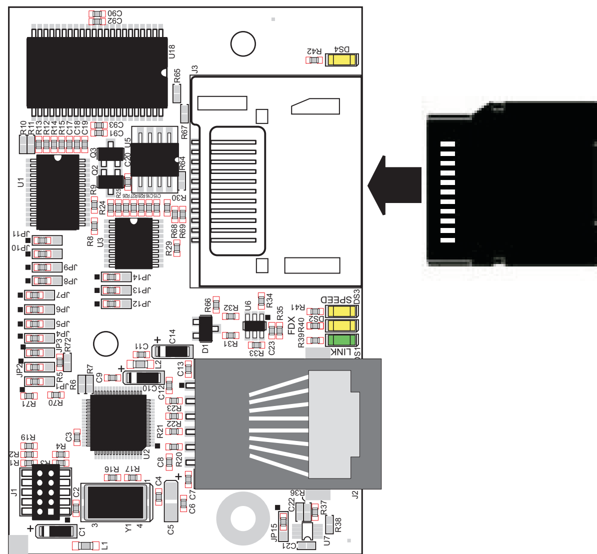


Figure 20. Insertion/Removal of microSD Card

NOTE: When using the Dynamic C FAT file system, do ***not*** remove or insert the *microSD™ Card* while LED DS4 above the *microSD™ Card* is on to indicate that the *microSD™ Card* is mounted. The LED will go off when the *microSD™ Card* is unmounted, indicating that it is safe to remove it.

Rabbit recommends that you use the *microSD™ Card* holder at connector J3 on the RabbitCore module only for the *microSD™ Card* since other devices are not supported. Be careful to remove and insert the card as shown, and be careful ***not*** to insert any foreign objects, which may short out the contacts and lead to the destruction of your card.

It is possible to hot-swap *microSD™ Card*s without removing power from the BL4S200. The file system must be closed before the cards can be hot-swapped. The chip selects associated with the card must be set to their inactive state, and read/write operations addressed to the *microSD™ Card* port cannot be allowed to occur. These operations can be initiated in software by sensing an external switch actuated by the user, and the card can then be removed and replaced with a different one. Once the application program detects a new card, the file system can be opened. These steps allow the *microSD™ Card* to be installed or removed without affecting either the program, which continues to run on the RCM4300 module, or the data stored on the card. The Dynamic C FAT file system will handle this overhead automatically when you unmount the *microSD™ Card*. LED DS4 above the *microSD™ Card* is used by the FAT file system to show when the media is mounted.

Standard Windows SD Card readers may be used to read the *microSD™ Card* formatted by the Dynamic C FAT file system with the BL4S200 as long as it has not been partitioned. SD Card adapters have a sliding switch along the left side that may be moved down to write-protect the *microSD™ Card* while it is being used with an SD Card reader.

Sample programs in the **`SAMPLES\BLxS2xx\SD_Flash`** folder illustrate the use of the *microSD™ Card*s.

4. SOFTWARE

Dynamic C is an integrated development system for writing embedded software. It runs on an IBM-compatible PC and is designed for use with single-board computers and other devices based on the Rabbit microprocessor.

Chapter 4 provides the libraries, function calls, and sample programs related to the BL4S200.

4.1 Running Dynamic C

You have a choice of doing your software development in the flash memory or in the static RAM included on the BL4S200. The flash memory and SRAM options are selected with the **Options > Project Options > Compiler** menu.

The advantage of working in RAM is to save wear on the flash memory, which is limited to about 100,000 write cycles. The disadvantage is that the code and data might not both fit in RAM.

> **NOTE:** On the BL4S210, an application can be developed in RAM, but cannot run standalone from RAM after the programming cable is disconnected. Standalone applications can only run from flash memory.

> **NOTE:** Do not depend on the flash memory sector size or type. Due to the volatility of the flash memory market, the BL4S200 and Dynamic C were designed to accommodate flash devices with various sector sizes.

Developing software with Dynamic C is simple. Users can write, compile, and test C and assembly code without leaving the Dynamic C development environment. Debugging occurs while the application runs on the target. Alternatively, users can compile a program to an image file for later loading. Dynamic C runs on PCs under Windows NT and later—see Rabbit's Technical Note TN257, *Running Dynamic C® With Windows Vista®*, for additional information if you are using a Dynamic C under Windows Vista. Programs can be downloaded at baud rates of up to 460,800 bps after the program compiles.

Dynamic C has a number of standard features:

- Full-feature source and/or assembly-level debugger, no in-circuit emulator required.
- Royalty-free TCP/IP stack with source code and most common protocols.
- Hundreds of functions in source-code libraries and sample programs:
    - Exceptionally fast support for floating-point arithmetic and transcendental functions.
    - RS-232 and RS-485 serial communication.
    - Analog and digital I/O drivers.
    - I^2C, SPI, GPS, file system.
    - LCD display and keypad drivers.
- Powerful language extensions for cooperative or preemptive multitasking
- Loader utility program to load binary images into Rabbit targets in the absence of Dynamic C.
- Provision for customers to create their own source code libraries and augment on-line help by creating “function description” block comments using a special format for library functions.
- Standard debugging features:
    - Breakpoints—Set breakpoints that can disable interrupts.
    - Single-stepping—Step into or over functions at a source or machine code level, µC/OS-II aware.
    - Code disassembly—The disassembly window displays addresses, opcodes, mnemonics, and machine cycle times. Switch between debugging at machine-code level and source-code level by simply opening or closing the disassembly window.
    - Watch expressions—Watch expressions are compiled when defined, so complex expressions including function calls may be placed into watch expressions. Watch expressions can be updated with or without stopping program execution.
    - Register window—All processor registers and flags are displayed. The contents of general registers may be modified in the window by the user.
    - Stack window—shows the contents of the top of the stack.
    - Hex memory dump—displays the contents of memory at any address.
    - **STDIO** window—`printf` outputs to this window and keyboard input on the host PC can be detected for debugging purposes. `printf` output may also be sent to a serial port or file.

4.1.1 Upgrading Dynamic C

4.1.1.1 Patches and Updates

Dynamic C patches that focus on bug fixes and updates are available from time to time. Check the Web site at www.digi.com/support/ for the latest patches, workarounds, and updates.

The default installation of a patch or update is to install the file in a directory (folder) different from that of the original Dynamic C installation. Rabbit recommends using a different directory so that you can verify the operation of the patch or update without over-writing the existing Dynamic C installation. If you have made any changes to the BIOS or to libraries, or if you have programs in the old directory (folder), make these same changes to the BIOS or libraries in the new directory containing the patch. Do ***not*** simply copy over an entire file since you may overwrite an update; of course, you may copy over any programs you have written. Once you are sure the new patch or update works entirely to your satisfaction, you may retire the existing installation, but keep it available to handle legacy applications.

4.1.2 Add-On Modules

Starting with Dynamic C version 10.40, Dynamic C includes the popular µC/OS-II real-time operating system, point-to-point protocol (PPP), FAT file system, RabbitWeb, and other select libraries. Starting with Dynamic C version 10.56, Dynamic C includes the Rabbit Embedded Security Pack featuring the Secure Sockets Layer (SSL) and a specific Advanced Encryption Standard (AES) library.

In addition to the Web-based technical support included at no extra charge, a one-year telephone-based technical support subscription is also available for purchase.

Visit our Web site at www.digi.com for further information and complete documentation.

4.2 Sample Programs

Sample programs are provided in the Dynamic C **`Samples`** folder. The sample program **`PONG.C`** demonstrates the output to the **STDIO** window.

The various directories in the **`Samples`** folder contain specific sample programs that illustrate the use of the corresponding Dynamic C libraries.

The **`SAMPLES\BLxS2xx`** folder provides sample programs specific to the BL4S200. Each sample program has comments that describe the purpose and function of the program. Follow the instructions at the beginning of the sample program.

To run a sample program, open it with the **File** menu (if it is not still open), then compile and run it by pressing **F9**. The BL4S200 must be in **Program** mode (see Section 3.7, “USB Programming Cable,”) and must be connected to a PC using the programming cable as described in Section 2.2, “BL4S200 Connections.” See Appendix C for information on the power-supply connections to the Demonstration Board.

Complete information on Dynamic C is provided in the ***Dynamic C User’s Manual***. TCP/IP specific functions are described in the ***Dynamic C TCP/IP User’s Manual***. Information on using the TCP/IP features and sample programs is provided in Section 5, “Using the Ethernet TCP/IP Features.”

4.2.1 Digital I/O

The following sample programs are found in the **SAMPLES\BLxS2xx\DIO** subdirectory.

Figure 21 shows the signal connections for the sample programs that illustrate the use of the digital inputs.

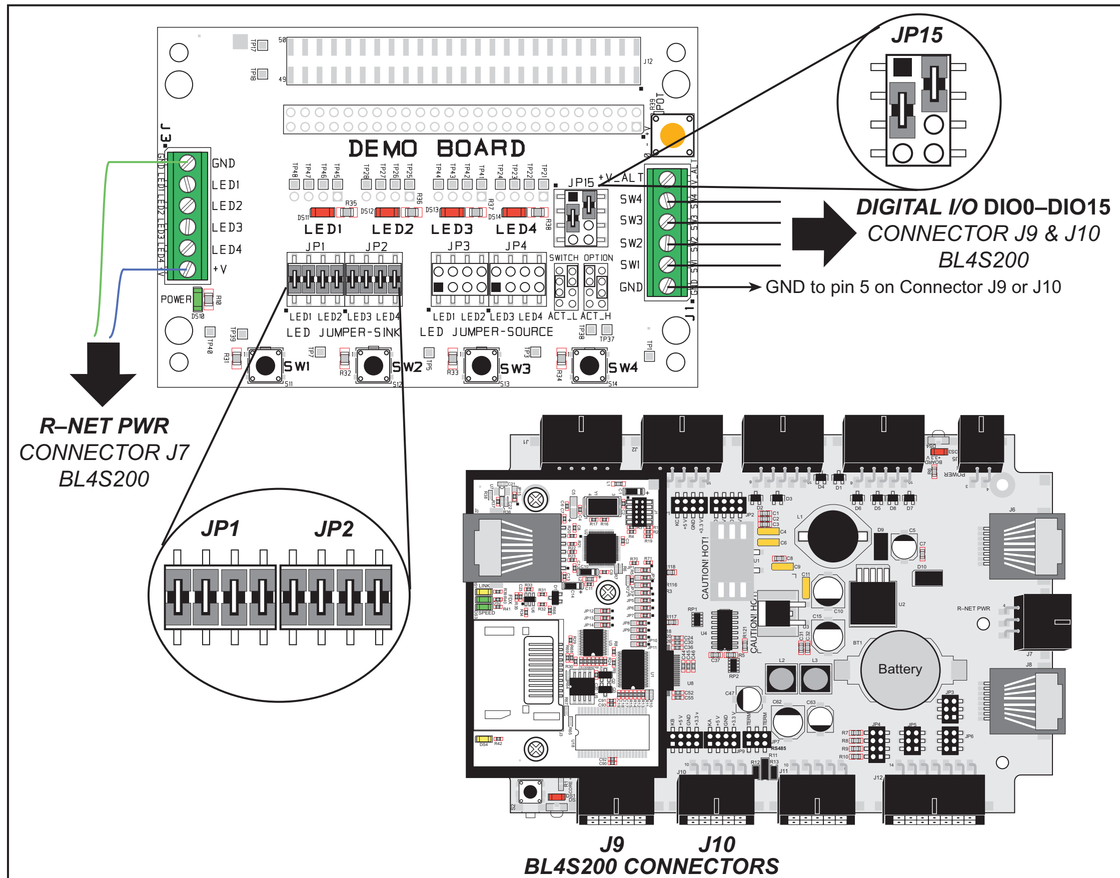


Figure 21. Digital Inputs Signal Connections

- **DIGIN.C**—Demonstrates the use of the digital inputs. Using the Demonstration Board, you can see an input channel toggle from HIGH to LOW in the Dynamic C **STDIO** window when you press a pushbutton on the Demonstration Board.
- **DIGIN_BANK.C**—Demonstrates the use of **digInBank()** to read digital inputs. Using the Demonstration Board, you can see an input channel toggle from HIGH to LOW in the Dynamic C **STDIO** window when you press a pushbutton on the Demonstration Board.

Figure 22 shows the signal connections for the sample programs that illustrate the use of the digital outputs.

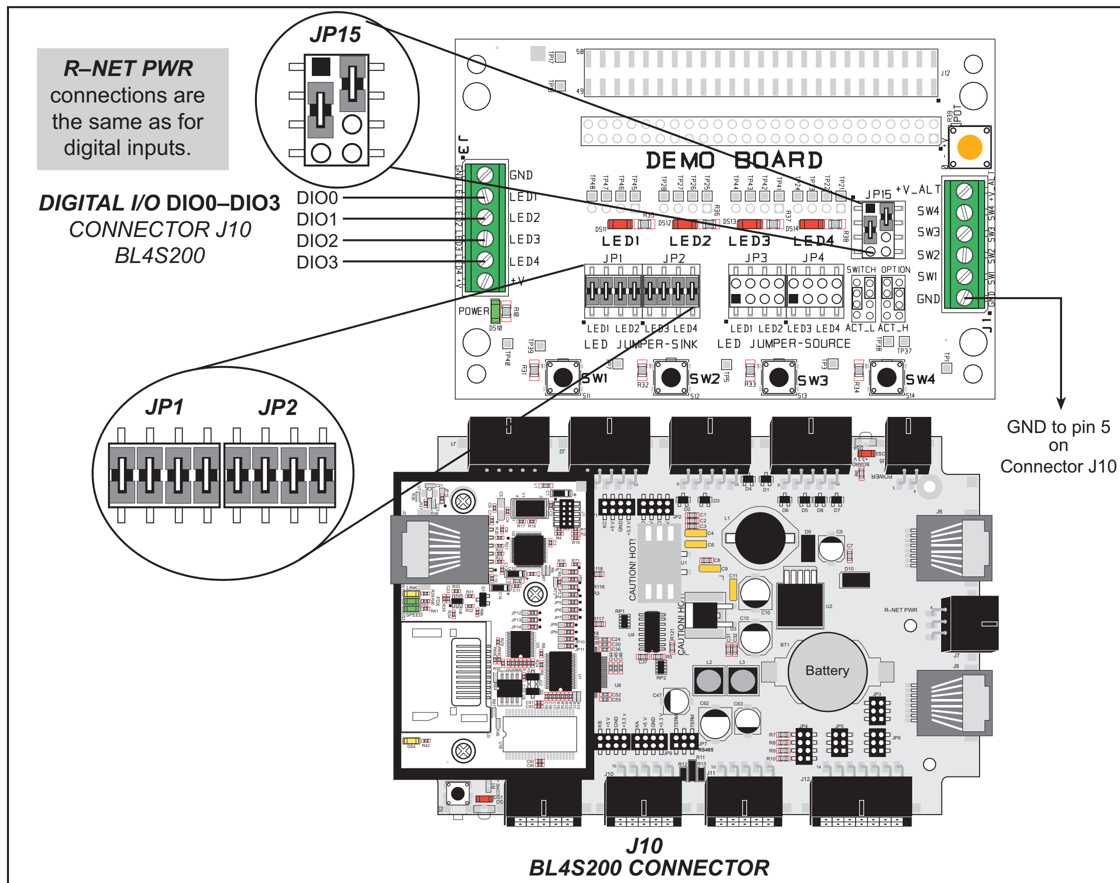


Figure 22. Digital Outputs Signal Connections

- **`DIGOUT.C`**—Demonstrates the use of the configurable I/O sinking outputs. Using the Demonstration Board, you can see an LED toggle on/off via a sinking output.
- **`DIGOUT_BANK.C`**—Demonstrates the use of **`digOutBank()`** to control the configurable I/O sinking outputs. Using the Demonstration Board, you can see an LED toggle on/off via a sinking output.

Figure 23 shows the signal connections for the sample program that illustrates the use of the high-current outputs. Note that the regular power-supply connection is substituted by HOUT0, which operates in the sourcing mode to supply power for this sample program.

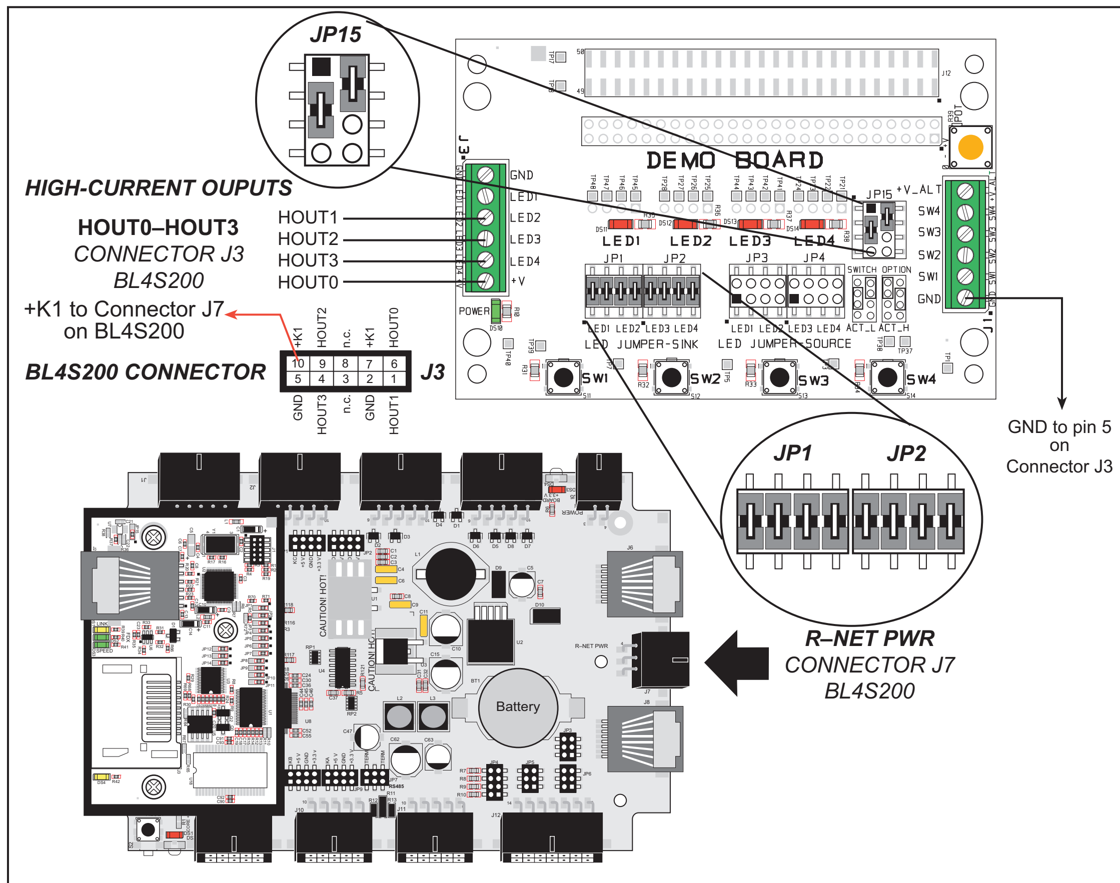


Figure 23. High-Current Outputs Signal Connections

- **`HIGH_CURRENT_IO.C`**—Demonstrates the use of the high-current outputs configured as either sinking or sourcing outputs. High-current output HOUT0 is configured for sourcing to provide power to the Demonstration Board. Outputs HOUT1 and HOUT2 are configured to demonstrate tristate operation to toggle the LEDs on the Demonstration Board. Output HOUT3 is configured as a sinking output to toggle an LED on the Demonstration Board.

- **`INTERRUPTS.C`**—Demonstrates the use of the Rabbit RIO interrupt service capabilities. Set up the Demonstration Board as shown in Figure 21 with DIO0 connected to SW1.

  The sample program sets up two interrupt sources, an external interrupt tied to pushbutton switch SW1, and a rollover interrupt tied to a timer that is producing a PWM output. The Dynamic C **STDIO** window will show a count of rollovers that have occurred since the PWM signal was started. The window will also display *Button Pressed* each time the pushbutton switch is pressed. Each time the button is pressed, the timeout timer that removes the message is reset, so you can keep the message on the screen indefinitely by pressing the button repeatedly.

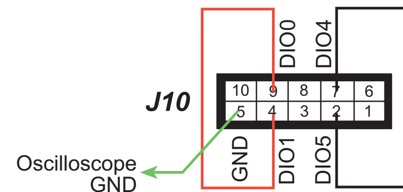
- **`PPM.C`**—Demonstrates the use of four PPM channels on the configurable I/O pins DIO0, DIO2, DIO4, and DIO6 on connector J10. The PPM signals are set for a frequency of 200 Hz, with the duty cycle adjustable from 0 to 100% and an offset adjustable from 0 to 100% by the user. These pins can be connected to an oscilloscope to view the waveform being generated. The overall frequency can be adjusted in the **`#define PPM_FREQ`** line. Follow these instructions when running this sample program.

  1. Verify that the jumper on header JP9 is in the default position across pins 3–4 for +5 V pullup.
  2. Connect the oscilloscope probe to the configurable I/O pins on connector J10. Remember to connect the oscilloscope ground to GND on connector J10.

  Change the duty cycle and offsets for a given PPM channel via the Dynamic C **STDIO** window and watch the change in waveforms on the oscilloscope. Signals on DIO0 (PPM00) and DIO2 (PPM01) will all be synchronized with each other as they share the same overall counter block that sets the cycle frequency. The same is true for PPM signals on DIO4 (PPM02) and DIO6 (PPM03). The two blocks may have a phase shift from each other, but will run at the same frequency.

- **`PULSE_CAPTURE.C`**—Demonstrates the use of two input capture inputs tied to PPM channels on the configurable digital I/O pins on connector J10. The input capture feature allows the begin and end positions of a pulse to be measured in a given time window. We take advantage of the counter synchronization feature of the underlying Rabbit RIO chip to create capture windows and pulse modulation windows that are synchronized. This guarantees that we always catch the begin edge first on a quickly repeating waveform. This was done to create an interactive element to this sample program, but capturing real-world repetitive signals will usually not have this advantage. Refer to Section 3.2.1.3 for more information on how to use the input capture.feature. Follow these instructions when running this sample program.

  1. Connect digital I/O pins DIO0 and DIO1 together.
  2. Connect digital I/O pins DIO4 and DIO5 together.
  3. Connect the oscilloscope ground to GND on connector J10.
  4. Use the oscilloscope probes on the DIO0 and the DIO1 pair or the DIO4 and DIO5 pair to view the PPM signals.



  Once the connections have been made, compile and run this sample program. Change the offset and duty cycle for a given PPM channel via the Dynamic C **STDIO** window and watch the change to the begin and end counts measured on the input capture inputs. The PPM frequency can be changed in the **`#define PPM_FREQ`** line.

- **`PWM.C`**—Demonstrates the use of the eight PWM channels on configurable I/O pins DIO0–DIO7. The PWM signals are set for a frequency of 200 Hz with the duty cycle adjustable from 0 to 100% by the user. These pins can be connected to an oscilloscope to view the waveform being generated. The overall frequency can be adjusted in the **`#define PWM_FREQ`** line. Follow these instructions when running this sample program.

  1. Verify that the jumper on header JP9 is in the default position across pins 3–4 for +5 V pullup.
  2. Connect the oscilloscope probe to the configurable I/O pins on connector J10. Remember to connect the oscilloscope ground to GND on connector J10.

- **`QUADRATURE_DECODER.C`**—Demonstrates the use of quadrature decoders on the BL4S200. See Figure 24 for hookup instructions of configurable I/O pins DIO0–DIO6 on connector J10 with the Demonstration Board.

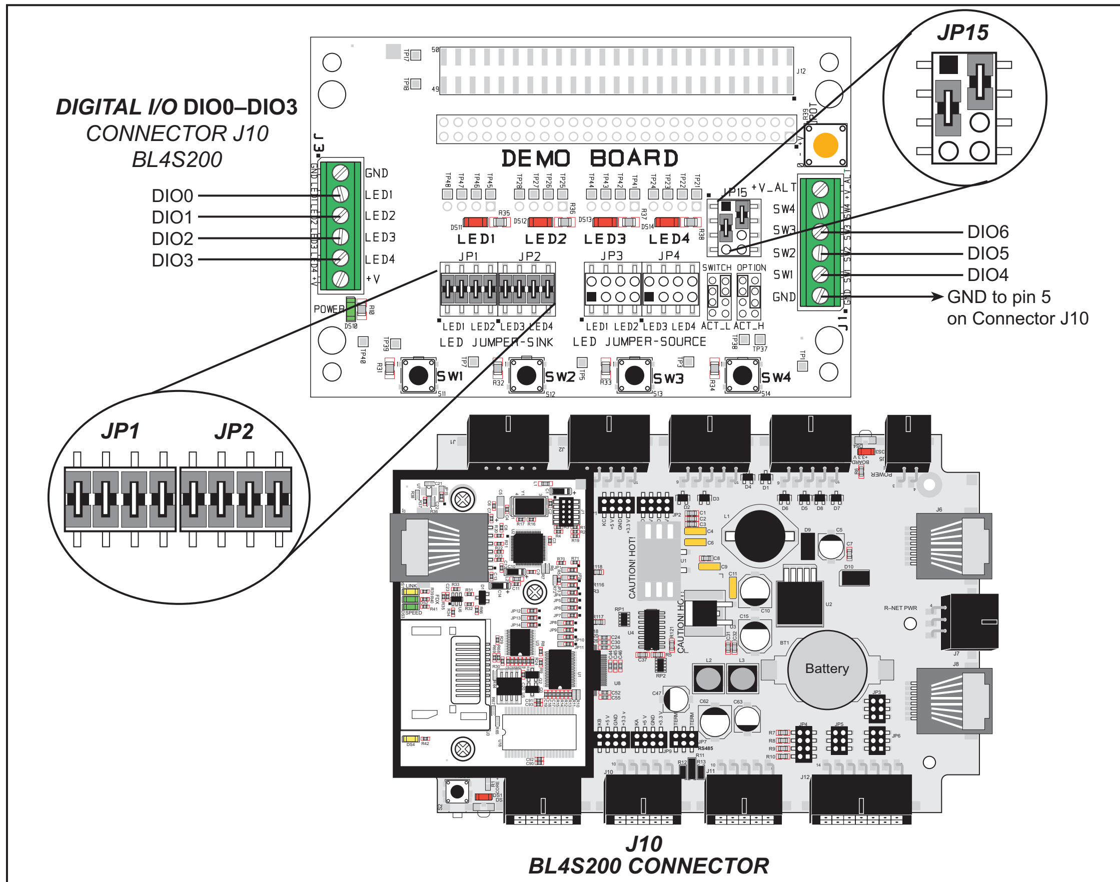


Figure 24. Quadrature Decoder Signal Connections

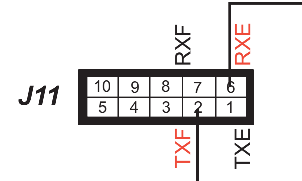
Once the connections have been made, compile and run this sample program. Press button SW1 on the Demonstration Board to decrement the quadrature counter, or press button SW2 on the Demonstration Board to increment the quadrature counter. Press button SW3 on the Demonstration Board to reset the quadrature counter.

4.2.2 Serial Communication

The following sample programs are found in the **`SAMPLES\BLxS2xx\RS232`** subdirectory.

- **`PARITY.C`**—This sample program repeatedly sends byte values 0–127 from Serial Port F to Serial Port E. The program switches between generating parity and not generating parity on Serial Port F. Serial Port E will always be checking parity, so parity errors should occur during every other sequence. The results are displayed in the Dynamic C **STDIO** window.

  Connect TxF (pin 2 on connector J11) to RxE (pin 6 on connector J11) before compiling and running this sample program.

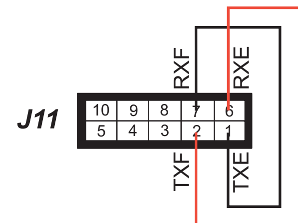
  

  The BL4S210 model only has one RS-232 serial port available, and so this sample program cannot be run on the BL4S210 model.

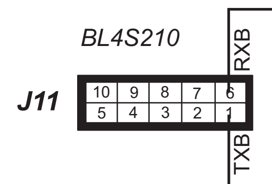
  NOTE: For the sequence that does yield parity errors, the errors won't occur for each byte received. This is because certain byte patterns along with the stop bit will appear to generate the correct parity for the UART.

- **`SIMPLE3WIRE.C`**—This program demonstrates basic RS-232 serial communication using the Dynamic C **STDIO** window. Follow these instructions before running this sample program.

  BL4S200, BL5S220, BL4S230 models—Connect TxE (pin 1 on connector J11) to RxF (pin 7 on connector J11), then connect TxF (pin 2 on connector J11) to RxE (pin 6 on connector J11) before compiling and running this sample program.

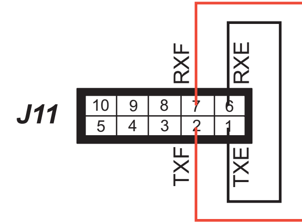
  BL4S210—Connect TxB (pin 1 on connector J11) to RxB (pin 6 on connector J11) before compiling and running this sample program.

- **SIMPLE5WIRE.C**—This program demonstrates 5-wire RS-232 serial communication using the Dynamic C **STDIO** window. Follow these instructions before running this sample program.

  The BL4S210 model only has one RS-232 serial port available, and so this sample program cannot be run on the BL4S210 model.

  Before you compile and run this sample program on any of the other BL4S200 models, connect TxE (pin 1 on connector J11) to RxE (pin 6 on connector J11), then connect TxF (pin 2 on connector J11) to RxF (pin 7 on connector J11).

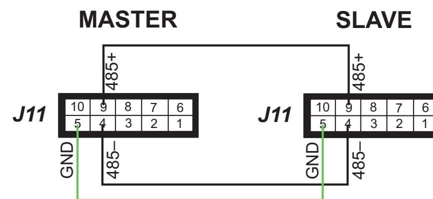


  TxF and RxF become the flow control RTS and CTS. To test flow control, disconnect RTS from CTS while running this program. Characters should stop printing in the Dynamic C **STDIO** window and should resume when RTS and CTS are connected again

The following sample programs are found in the **SAMPLES\BLxS2xx\RS485** subdirectory.

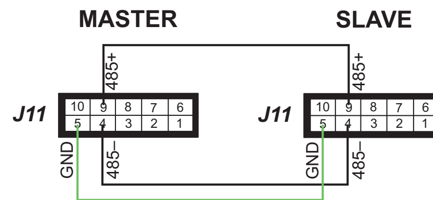
- **MASTER.C**—This program demonstrates a simple RS-485 transmission of lower case letters to a slave. The slave will send back converted upper case letters back to the master BL4S200 and display them in the **STDIO** window. Use **SLAVE.C** to program the slave. Make the following connections between the master and slave:

  485+ to 485+ (pin 9 on connector J11)
  485- to 485- (pin 4 on connector J11)
  GND to GND (pin 5 on connector J11)



- **SLAVE.C**—This program demonstrates a simple RS-485 transmission of lower case letters to a slave. The slave will send back converted upper case letters back to the master BL4S200 and display them in the **STDIO** window. Use **MASTER.C** to program the master BL4S200. Make the following connections between the master and slave:

  485+ to 485+ (pin 9 on connector J11)
  485- to 485- (pin 4 on connector J11)
  GND to GND (pin 5 on connector J11)



4.2.3 A/D Converter Inputs

The following sample programs are found in the **SAMPLES\BLxS2xx\ADC** subdirectory. You will need a separate power supply and a multimeter to use with these sample programs.

> **NOTE:** The calibration sample programs will overwrite the calibration constants set at the factory. Before you run these sample programs, run **USERBLOCK_READ_WRITE.C** in the **SAMPLES\UserBlock** folder to save the factory calibration constants in case you inadvertently write over them while running other sample programs.

- **ADC_CAL_DIFF.C**—Demonstrates how to recalibrate a differential A/D converter channel using two measured voltages to generate two coefficients, gain and offset, which are rewritten into the user block. The voltage that is being monitored is displayed continuously.

  Once you compile and run this sample program, connect the power supply across a differential channel pair, then follow the instructions in the Dynamic C **STDIO** window.

- **ADC_CAL_MA.C**—Demonstrates how to recalibrate a milli-amp A/D converter channel using two measured currents to generate two coefficients, gain and offset, which are rewritten into the reserved user block. The current that is being monitored is displayed continuously.

  Before you compile and run this sample program, jumper pins 1–2, 3–4, 5–6, and 7–8 on header JP4. Then connect a current meter in series with the power supply connected to one of pins AIN0–AIN3 and AGND, then compile and run the sample program, and follow the instructions in the Dynamic C **STDIO** window.

- **ADC_CAL_SE_BIPOLAR.C**—Demonstrates how to recalibrate a single-ended bipolar A/D converter channel using two measured voltages to generate two coefficients, gain and offset, which are rewritten into the reserved user block. The voltage that is being monitored is displayed continuously.

  Before you compile and run this sample program, connect the power supply (which should be OFF) to one of pins AIN0–AIN3 and AGND, then compile and run the sample program, and follow the instructions in the Dynamic C **STDIO** window.

- **ADC_CAL_SE_UNIPOLAR.C**—Demonstrates how to recalibrate a single-ended unipolar A/D converter channel using two measured voltages to generate two coefficients, gain and offset, which are rewritten into the reserved user block. The voltage that is being monitored is displayed continuously.

  Before you compile and run this sample program, connect the power supply (which should be OFF) between the pin (AIN0–AIN7) of the channel you are calibrating and AGND, then compile and run the sample program, and follow the instructions in the Dynamic C **STDIO** window.

- **ADC_RD_CALDATA.C**—Demonstrates how to display the two calibration coefficients, gain and offset, in the Dynamic C **STDIO** window for each channel and mode of operation.

- **`AD_RD_DIFF.C`**—Demonstrates how to read and display voltage and equivalent values for a differential A/D converter channel using calibration coefficients previously stored in the user block. The user selects to display either the raw data or the voltage equivalent.

  Once you compile and run this sample program, connect the power supply across a differential channel pair, then follow the instructions in the Dynamic C **STDIO** window.

- **`AD_RD_MA.C`**—Demonstrates how to read and display voltage and equivalent values for a milli-amp A/D converter channel using calibration coefficients previously stored in the user block. The user selects to display either the raw data or the current equivalent.

  Before you compile and run this sample program, jumper pins 1–2, 3–4, 5–6, and 7–8 on header JP4. Then connect a current meter in series with the power supply connected to one of pins AIN0–AIN3 and AGND, then compile and run the sample program, and follow the instructions in the Dynamic C **STDIO** window as you vary the output from the power supply.

- **`AD_RD_SE_AVERAGING.C`**—Demonstrates how to read and display the voltage of all single-ended analog input channels using a sliding window. The voltage is calculated from coefficients read from the reserved user block.

  Before you compile and run this sample program, connect the power supply (which should be OFF) between a pin (AIN0–AIN7) and AGND, then compile and run the sample program, and follow the instructions in the Dynamic C **STDIO** window. The voltage readings will be displayed for all the channels measured to that point.

- **`AD_RD_SE_BIPOLAR.C`**—Demonstrates how to read and display the voltage of all single-ended A/D converter channels using calibration coefficients previously stored in the user block.

  Before you compile and run this sample program, connect the power supply (which should be OFF) between a pin (AIN0–AIN7) and AGND, then compile and run the sample program, and follow the instructions in the Dynamic C **STDIO** window. Reverse the power supply connections to get a negative voltage reading.

- **`AD_RD_SE_UNIPOLAR.C`**—Demonstrates how to read and display the voltage of all single-ended A/D converter channels using calibration coefficients previously stored in the user block.

  Before you compile and run this sample program, connect the power supply (which should be OFF) between a pin (AIN0–AIN7) and AGND, then compile and run the sample program, and follow the instructions in the Dynamic C **STDIO** window.

4.2.4 D/A Converter Outputs

The following sample programs are found in the **SAMPLES\BLxS2xx\DAC** subdirectory.

> **NOTE:** The calibration sample programs will overwrite the calibration constants set at the factory.

- **DAC_CAL_MA.C**—Demonstrates how to recalibrate a D/A converter channel using a measured current to generate calibration constants, which are written into the reserved user block.

  Before you compile and run this sample program, connect pins 3–5 and 4–6 on header JP5, and verify that jumpers are in place across pins 1–2 and 3–4 on both headers JP3 (AOUT0) and JP6 (AOUT1). Now compile and run the sample program, and follow the instructions in the Dynamic C **STDIO** window.

- **DAC_CAL_VOLTS.C**—Demonstrates how to recalibrate a D/A converter channel using a measured voltage to generate calibration constants, which are written into the reserved user block.

  Before you compile and run this sample program, connect pins 1–3 and 2–4 on header JP5, then connect pins 1–2 and 3–4 on both headers JP3 and JP6 (unipolar), or connect pins 5–6 on both headers JP3 and JP6 (bipolar). Now connect a voltmeter across one of the D/A converter outputs, then compile and run the sample program, and follow the instructions in the Dynamic C **STDIO** window.

- **DAC_MA_ASYNC.C**—Demonstrates how to output a current that can be read with an ammeter. The output current is computed with using the calibration constants that are stored in the reserved user block.

  The D/A converter circuit is set up for asynchronous operation, which updates the D/A converter output at the time it's being written via the **anaOut()** or **anaOutmAmps()** function calls.

  Before you compile and run this sample program, connect pins 3–5 and 4–6 on header JP5, and verify that jumpers are in place across pins 1–2 and 3–4 on both headers JP3 (AOUT0) and JP6 (AOUT1). Now set up an ammeter in series with the D/A converter output and a resistor from 50 Ω to 400 Ω, then compile and run the sample program, and follow the instructions in the Dynamic C **STDIO** window.

- **DAC_MA_SYNC.C**—Demonstrates how to output a current that can be read with an ammeter. The output current is computed using the calibration constants that are stored in the reserved user block.

  The D/A converter circuit is set up for synchronous operation, which updates the D/A converter output when the **anaOutStrobe()** function call executes. The outputs will be updated with values previously written via the **anaOut()** or **anaOutmAmps()** function calls.

  Before you compile and run this sample program, connect pins 3–5 and 4–6 on header JP5, and verify that jumpers are in place across pins 1–2 and 3–4 on both headers JP3 (AOUT0) and JP6 (AOUT1). Now set up an ammeter in series with the D/A converter

output and a resistor from 50 Ω to 400 Ω, then compile and run the sample program, and follow the instructions in the Dynamic C **STDIO** window.

- **`DAC_RD_CALDATA.C`**—Demonstrates how to display the calibration coefficients, gain and offset, in the Dynamic C **STDIO** window for each channel and mode of operation.
- **`DAC_VOLT_ASYNC.C`**—Demonstrates how to output a voltage that can be read with a voltmeter. The output voltage is computed with using the calibration constants that are stored in the reserved user block.

  The D/A converter circuit is set up for asynchronous operation, which updates the D/A converter output at the time it's being written via the **`anaOut()`** or **`anaOutVolts()`** function calls.

  Before you compile and run this sample program, connect pins 1–3 and 2–4 on header JP5, then connect pins 1–2 and 3–4 on both headers JP3 and JP6 (unipolar), or connect pins 5–6 on both headers JP3 and JP6 (bipolar). Now connect a voltmeter across one of the D/A converter outputs, then compile and run the sample program, and follow the instructions in the Dynamic C **STDIO** window.
- **`DAC_VOLT_SYNC.C`**—Demonstrates how to output a voltage that can be read with a voltmeter. The output voltage is computed using the calibration constants that are stored in the reserved user block.

  The D/A converter circuit is set up for synchronous operation, which updates the D/A converter output when the **`anaOutStrobe()`** function call executes. The outputs will be updated with values previously written via the **`anaOut()`** or **`anaOutVolts()`** function calls.

  Before you compile and run this sample program, connect pins 1–3 and 2–4 on header JP5, then connect pins 1–2 and 3–4 on both headers JP3 and JP6 (unipolar), or connect pins 5–6 on both headers JP3 and JP6 (bipolar). Now connect a voltmeter across one of the D/A converter outputs, then compile and run the sample program, and follow the instructions in the Dynamic C **STDIO** window.

4.2.5 Use of *microSD*™ Cards with BL4S200 Model

The following sample program can be found in the **SAMPLES\BLxS2xx\SD_Flash** folder.

- **SDFLASH_INSPECT.c**—This program is a utility for inspecting the contents of a *microSD™ Card*. It provides examples of both reading and writing pages or sectors to the a *microSD™ Card*. When the sample program starts running, it attempts to initialize the *microSD™ Card* on Serial Port B. The following five commands are displayed in the Dynamic C **STDIO** window if a *microSD™ Card* is found:

  p — print out the contents of a specified page on the *microSD™ Card*

  r — print out the contents of a range of pages on the *microSD™ Card*

  c — clear (set to zero) all of the bytes in a specified page

  f — sets all bytes on the specified page to the given value

  t — write user-specified text to a selected page

  The sample program prints out a single line for a page if all bytes in the page are set to the same value. Otherwise it prints a hex/ASCII dump of the page.

  This utility works with the *microSD™ Card* at its lowest level, and writing to pages will likely make the *microSD™ Card* unreadable by a PC. For PC compatibility, you must use the Dynamic C FAT file system module, which allows you to work with files on the *microSD™ Card* in a way that they will be PC-compatible.

4.2.6 Real-Time Clock

If you plan to use the real-time clock functionality in your application, you will need to set the real-time clock. You may set the real-time clock using the **SETRTCKB.C** sample program from the Dynamic C **SAMPLES\RTCLOCK** folder. The **RTC_TEST.C** sample program in the Dynamic C **SAMPLES\RTCLOCK** folder provides additional examples of how to read and set the real-time clock

4.2.7 TCP/IP Sample Programs

TCP/IP sample programs are described in Chapter 5.

4.3 BL4S200 Libraries

Two library directories provide libraries of function calls that are used to develop applications for the BL4S200.

- **BLxS2xx**—libraries associated with features specific to the BL4S200. The functions in the **BLxS2xx.LIB** library are described in Section 4.4, “BL4S200 Function Calls.”
- **RN_CFG_BLS2xx.LIB**—used to configure the BL4S200 for use with RabbitNet peripheral boards.
- **TCPIP**—libraries specific to using TCP/IP functions on the BL4S200. Further information about TCP/IP is provided in Chapter 5, “Using the Ethernet TCP/IP Features.”

4.4 BL4S200 Function Calls

4.4.1 Board Initialization

---

brdInit

```
void brdInit (void);
```

FUNCTION DESCRIPTION

Call this function at the beginning of your program. This function initializes the system I/O ports.

The ports are initialized according to Table A-3 in Appendix A.

4.4.2 Digital I/O

`setDigIn`

```
int setDigIn(int channel);
```

FUNCTION DESCRIPTION

Sets a configurable I/O channel to be a general digital input.

PARAMETERS

`channel`	configurable I/O channel to be set as an input, 0–31 (pins DIO0–DIO31)

RETURN VALUE

0 — success.

`-EINVAL` — invalid parameter value.

SEE ALSO

`brdInit, digIn, digInBank`

`digIn`

```
int digIn(int channel);
```

FUNCTION DESCRIPTION

Reads the state of a channel set to any form of digital input functionality.

PARAMETERS

`channel`	configurable I/O channel set as an input, 0–31 (pins DIO0–DIO31)

RETURN VALUE

The logic state of the specified channel.

0 — logic low
1 — logic high

`-EINVAL` — channel value is out of range.

`-EPERM`:— pin functionality does not permit this operation.

SEE ALSO

`brdInit, setDigIn, digInBank`

digInBank

```
int digInBank(int bank);
```

FUNCTION DESCRIPTION

Reads the state of the 32 configurable I/O channels.

PARAMETER

bank digital input bank pins:

0 — DIO0–DIO7
1 — DIO8–DIO15
2 — DIO16–DIO23
3 — DIO24–DIO31

RETURN VALUE

Data read from the bank of digital inputs.

Data Bits		Bank 0	Bank 1	Bank 2	Bank 3
LSB	D0	DIO0	DIO8	DIO16	DIO24
	D1	DIO1	DIO9	DIO17	DIO25
	D2	DIO2	DIO10	DIO18	DIO26
	D3	DIO3	DIO11	DIO19	DIO27
	D4	DIO4	DIO12	DIO20	DIO28
	D5	DIO5	DIO13	DIO21	DIO29
	D6	DIO6	DIO14	DIO22	DIO30
MSB	D7	DIO7	DIO15	DIO23	DIO31

-EINVAL — invalid parameter value.

-EPERM — pin functionality does not permit this operation.

SEE ALSO

brdInit, digIn, setDigIn

setExtInterrupt

```
int setExtInterrupt(int channel, char edge, int handle);
```

FUNCTION DESCRIPTION

Sets the specified channel to be an interrupt. The interrupt can be configured as a rising edge, falling edge, or either edge.

PARAMETERS

`channel`	input channel to be configured as an interrupt channel
`edge`	macro to set edge of the interrupt: **`BL_IRQ_RISE`** — interrupt event on rising edge **`BL_IRQ_FALL`** — interrupt event on falling edge **`BL_IRQ_BOTH`** — interrupt events on both edges
`handle`	handle for the ISR handler to service this interrupt

RETURN VALUE

0 — success.

`-EINVAL` — invalid parameter value.

`-EPERM` — pin type does not permit this function.

`-EACCES` — resource needed by this function is not available.

`-EFAULT` — internal data fault detected.

positive number — Mode Conflict — the positive number is a bitmap that corresponds to the pins on a particular block of a RIO chip that have not been configured to support this function call. Appendix D provides the details of the pin and block associations to allow you to identify the channels that need to be reconfigured to support this function call.

SEE ALSO

`brdInit, digIn, setDigIn`

setDecoder

```
int setDecoder(int channel_a, int channel_b, int channel_index,
   char index_polarity);
```

FUNCTION DESCRIPTION

Sets up Quadrature Decoder functionality on the specified channels. The Quadrature Decoder may optionally use an index channel.

PARAMETERS

`channel_a` channel to use as Input A (also known as in-phase or I)

`channel_b` channel to use as Input B (also known as quadrature or Q)

`channel_index` channel to use as index input (-1 if not used)

NOTE: The Quadrature Decoder count may still be reset by existing or new synch signals set up on the same block of a particular RIO chip.

`index_polarity` polarity of the index channel
(not used when channel_index set to -1)l.

0 — index on low level
non-zero — index on high level

RETURN VALUE

0 — success.

`-EINVAL` — invalid parameter value.

`-EACCESS`— resource needed by this function is not available.

SEE ALSO

`brdInit, getCounter, resetCounter`

setCounter

```
int setCounter(int channel, int mode, int edge, word options);
```

FUNCTION DESCRIPTION

Sets up the channel as a counter input, with selectable modes and edge settings. The counter will increment or decrement on each selected edge event. Use **`getCounter()`** to read the current count and use **`resetCounter()`** to force a reset of the counter.

PARAMETERS

`channel` channel to use for the up count input

`mode` macro to set the mode of the counter:

`BL_UP_COUNT` — continuous up count mode

`BL_DOWN_COUNT` — up/down count mode (uses 2 pins)

`BL_MATCH_ENABLE` — continuous up count mode with count stopping on any match event

`edge` edge setting macro for the up count event:

`BL_EDGE_RISE` — up count on rising edge

`BL_EDGE_FALL` — up count on falling edge

`BL_EDGE_BOTH` — up count on either edge

`options` options based on mode (N/A if the continuous up mode is selected):

`BL_EDGE_RISE` — down count on rising edge

`BL_EDGE_FALL` — down count on falling edge

`BL_EDGE_BOTH` — down count on either edge

If the up/down mode is selected, **`options`** has down count pin event edge settings (these settings cannot be on be same pin as the up count). The low 5 bits are the channel number for the down count input

If the stop on match mode is selected, **`options`** has the match count to stop at (other match registers on the block are set to max.).

setCounter (continued)

RETURN VALUE

0 — success.

-EINVAL — invalid parameter value or pin use.

-EPERM — pin type does not permit this function.

-EACCESS— resource needed by this function is not available.

-EFAULT — internal data fault detected.

positive number — Mode Conflict — the positive number is a bitmap that corresponds to the pins on a particular block of a RIO chip that have not been configured to support this function call. Appendix D provides the details of the pin and block associations to allow you to identify the channels that need to be reconfigured to support this function call.

SEE ALSO

brdInit, getCounter, resetCounter

setCapture

```
int setCapture(int channel, int mode, int edge, word options);
```

FUNCTION DESCRIPTION

Sets up the channel as an event capture input, with selectable modes and edge settings. The counter will run from a gated main or prescaled clock signal based on the run criteria of the selected mode, and begin/end events can be set to capture the count at the time of these events. Optionally, a second channel can be set (which shares the same RIO channel input block as **`channel`**) for two-signal begin/end event detection. Use **`getBegin()`** and **`getEnd()`** to read the captured count values and use **`resetCounter()`** to force a reset of the counter.

PARAMETERS

`channel` channel to use for the begin event input for all modes except **`BL_CNT_TIL_END`**, then it specifies the end event input.

`mode` mode macro for the counter/timer:

`BL_CNT_RUN` — continuous count mode

`BL_CNT_BEGIN_END` — start count on begin event, continue to count until end event detected

`BL_CNT_TIL_END` — count until end event detected

`BL_CNT_ON_BEGIN` — count while begin signal is active

NOTE: If an end event occurs before the begin event, the count will begin then end immediately on the begin event, and the end count will be 1. The begin count will be 0 or 1 based on the edge that triggered the event (0 = rising, 1 = falling).

`edge` edge/state macro setting for the begin event for all modes except **`BL_CNT_TIL_END`**, then it specifies the end event:

`BL_EVENT_RISE` — begin event on rising edge
`BL_EVENT_FALL` — begin event on falling edge
`BL_EVENT_BOTH` — begin event on any edge

The following two settings are only for the **`ON_BEGIN`** mode:

`BL_BEGIN_HIGH` — begin active while signal is high
`BL_BEGIN_LOW` — begin active while signal is low

setCapture (continued)

options options based on mode:

BL_CNT_TIL_END — begin input and edge can be selected
all others modes — end input and edge can be selected.

For all modes, the prescale clock and save limit flags can be used (OR in).

For input and edge selection, use:

low 5 bits for channel to use for begin/end input
BL_SAME_CHANNEL — begin and end both from same channel
BL_EVENT_RISE — begin/end event on rising edge
BL_EVENT_FALL — begin/end event on falling edge
BL_EVENT_BOTH — begin/end event on any edge

For clock and limit options use:

BL_PRESCALE — use prescaled clock
BL_SAVE_LIMIT — save current limit register value (otherwise limit set to 0xFFFF)

RETURN VALUE

0 — success.

-EINVAL — invalid parameter value.

-EPERM — pin type does not permit this function.

-EACCESS— resource needed by this function is not available.

-EFAULT — internal data fault detected.

positive number — Mode Conflict — the positive number is a bitmap that corresponds to the pins on a particular block of a RIO chip that have not been configured to support this function call. Appendix D provides the details of the pin and block associations to allow you to identify the channels that need to be reconfigured to support this function call.

SEE ALSO

brdInit, getBegin, getEnd, getCounter, resetCounter, setLimit

getCounter

```
int getCounter(int channel, word *count);
```

FUNCTION DESCRIPTION

Reads the current count of the counter register within the counter block hosting the given channel.

PARAMETERS

`channel`	a channel that uses the desired counter block
`count`	pointer to word variable to place count register reading

RETURN VALUE

0 — success.

`-EINVAL` — invalid parameter value.

SEE ALSO

`brdInit, setCounter, setDecoder, setCapture, resetCounter`

getBegin

```
int getBegin(int channel, word *begin);
```

FUNCTION DESCRIPTION

Reads the current value of the begin register within the counter block hosting the given channel.

PARAMETERS

`channel`	a channel that uses the desired counter block
`begin`	pointer to word variable to place begin register reading

RETURN VALUE

0 — success.

`-EINVAL` — invalid parameter value.

SEE ALSO

`brdInit, setCapture, resetCounter`

getEnd

```
int getEnd(int channel, word *end);
```

FUNCTION DESCRIPTION

Reads the current value of the end register within the counter block hosting the given channel.

PARAMETERS

`channel`	a channel that uses the desired counter block
`begin`	pointer to word variable to place end register reading

RETURN VALUE

0 — success.

`-EINVAL` — invalid parameter value.

SEE ALSO

`brdInit, setCapture, resetCounter`

resetCounter

```
int resetCounter(int channel);
```

FUNCTION DESCRIPTION

Resets the current count of the counter register within the counter block hosting the given channel. The active block is determined by the function the configurable I/O channel is set up to perform.

PARAMETER

`channel`	a channel that uses the desired counter block

RETURN VALUE

0 — success.

`-EINVAL` — invalid parameter value.

SEE ALSO

`brdInit, getCounter, setDecoder`

setLimit

```
int setLimit(int channel, word limit);
```

FUNCTION DESCRIPTION

Sets the value of the limit register within the counter block hosting the given channel. This new value will take effect on the next counter overflow or by resetting the counter via the **`resetCounter()`** function call.

PARAMETERS

`channel`	a channel that uses the desired counter block
`limit`	new value for the limit register

RETURN VALUE

0 — success.

`-EINVAL` — invalid parameter value.

SEE ALSO

`brdInit, setCapture, resetCounter`

setSync

```
int setSync(int channel, int source, int edge);
```

FUNCTION DESCRIPTION

Sets the synch for the block the channel is associated with.

Note that when more than one block is synchronized to the same synch signal (global or external), each block has its own independent edge-detection circuit. These circuits will synch to the edge within plus or minus one count of the block’s current clock source (main or prescale). This means synchronized blocks may have a small offset when compared to each other.

PARAMETERS

`channel` channel that is on the block that will have its synch set

`source` source of the synch signal.

-1 to use the RIO chip's Global Synch signal *or*
input-capable channel to use as an external synch signal

`edge` edge of the synch signal.

`BL_EDGE_RISE` — synchronize event on rising edge
`BL_EDGE_FALL` — synchronize event on falling edge
`BL_EDGE_BOTH` — synchronize events on both edges
`0` — disable the synch on this block (if the source of the external synch is given, it will be set to a digital input)

RETURN VALUE

0 — success.

`-EINVAL` — invalid parameter value.

`-EPERM` — pin type does not permit this function.

`-EACCES` — resource needed by this function is not available.

`-EFAULT` — internal data fault detected.

SEE ALSO

`brdInit`

globalSync

```
int globalSync(void);
```

FUNCTION DESCRIPTION

Sends a single pulse to the global synch inputs of all RIO chips.

+Note that when more than one block is synchronized to the same synch signal (global or external), each block has its own independent edge-detection circuit. These circuits will synch to the edge within plus or minus one count of the block's current clock source (main or prescale). This means synchronized blocks may have a small offset when compared to each other.

RETURN VALUE

0 — success.

-EPERM — **brdInit()** was not run before calling this function.

SEE ALSO

brdInit

setDigOut

```
int setDigOut(int channel, int state);
```

FUNCTION DESCRIPTION

Configures the output channel as a simple digital output. The output state of the channel is also initialized to logic 0 or logic 1 based on the **state** parameter. The digOut function should be used to control the output state after configuration as it is more efficient. This function is non-reentrant.

PARAMETERS

channel — digital output channel, 0–31 (DIO0–DIO31)

state — set output to one of the following states:

0 — connects the load to GND
1 — puts the output in a high-impedance tristate

RETURN VALUE

0 — success.

-EINVAL — invalid parameter value.

SEE ALSO

brdInit, digOut, digOutBank

digOut

```
void digOut(int channel, int state);
```

FUNCTION DESCRIPTION

Sets the state of a configurable I/O channel configured as a sinking digital output to a logic 0 or a logic 1. This function will only allow control of pins that are configured by the **setDigOut()** function call to be a sinking digital output.

PARAMETERS

channel digital output channel, 0–31 (DIO0–DIO31).

state set output to one of the following states:

0 — connects the load to GND
1 — puts the output in a high-impedance tristate.

RETURN VALUE

0 — success.

-EINVAL — invalid parameter value.

-EPERM — pin function was not set up as a digital output

SEE ALSO

brdInit, setDigOut, digOutBank

digOutBank

```
int digOutBank(char bank, char data);
```

FUNCTION DESCRIPTION

Sets the state (logic 0 or logic 1) of a bank of 8 digital output pins within one of 4 banks to the states contained in the **`data`** parameter. This function only updates the channels that are configured to be sinking digital outputs by the **`setDigOut()`** function call. Channels configured for other functionality will not be affected.

PARAMETERS

`bank` digital output bank pins:

0 — DIO0–DIO7
1 — DIO8–DIO15
2 — DIO16–DIO23
3 — DIO24–DIO31

`data` data value to be written to the specified digital output bank; the data format and bitwise value are as follows:

Data Bits		Bank 0	Bank 1	Bank 2	Bank 3
LSB	D0	DIO0	DIO8	DIO16	DIO24
	D1	DIO1	DIO9	DIO17	DIO25
	D2	DIO2	DIO10	DIO18	DIO26
	D3	DIO3	DIO11	DIO19	DIO27
	D4	DIO4	DIO12	DIO20	DIO28
	D5	DIO5	DIO13	DIO21	DIO29
	D6	DIO6	DIO14	DIO22	DIO30
MSB	D7	DIO7	DIO15	DIO23	DIO31

Bitwise value:

0 — connects the load to GND
1 — puts the output in a high-impedance tristate.

RETURN VALUE

0 — success.

`-EINVAL` — invalid parameter value or board not initialized.

SEE ALSO

`brdInit, digOut, setDigOut`

setPWM

```
int setPWM(int channel, float frequency, float duty,
   char invert, char bind);
```

FUNCTION DESCRIPTION

Sets up a PWM output on the selected configurable I/O channel with the specified frequency and duty cycle. The PWM output can be inverted. The PWM channel duty cycle can be bound to a PWM/PPM on another channel on the same RIO block so that they share an edge.

NOTE: Configurable I/O channels DIO30 and DIO31 do not support PWM/PPM functionality, and cannot be used with this function call.

PARAMETERS

channel configurable I/O channel being set up for PWM, 0–29 (DIO0–DIO29)

frequency PWM frequency in Hz (should be from 2 Hz to 50 kHz); use -1 to preserve the existing frequency on the RIO block

duty PWM duty cycle (should be from 0 to 100%); use -1 and **bind** parameter to use bound edge to set the duty cycle (a duty cycle above 100.0% will be set to 100.0%)

invert whether the PWM output is inverted; the PWM output normally starts with the output high and inverted starts with the output low.

0 — noninverted
1 — inverted

bind use **BL_BIND_LEAD** or **BL_BIND_TRAIL** to enable binding for the leading edge of the PWM output on this channel to another PWM or PPM output on a channel hosted by same RIO chip and block. Bindings allow PWM and PPM outputs to align their leading and trailing edges.

setPWM (continued)

RETURN VALUE

0 — success.

-EINVAL — invalid parameter value.

-EPERM — pin type does not permit this function.

-EACCES — resource needed by this function is not available.

-EFAULT — internal data fault detected.

positive number — Mode Conflict — the positive number is a bitmap that corresponds to the pins on a particular block of a RIO chip that have not been configured to support this function call. Appendix D provides the details of the pin and block associations to allow you to identify the channels that need to be reconfigured to support this function call.

SEE ALSO

brdInit, setFreq, setDuty, setToggle, setSync, pulseDisable

setPPM

```
int setPPM(int channel, float frequency, float offset,
   float duty, char invert, char bind_offset, char bind_duty);
```

FUNCTION DESCRIPTION

Sets up a PPM output on the selected configurable I/O channel with the specified frequency and duty cycle. The PPM output of the PPM can be inverted. The offset and duty of the PPM can be bound to a PWM/PPM on another channel on the same RIO block so that they share an edge.

NOTE: Configurable I/O channels DIO30 and DIO31 do not support PWM/PPM functionality, and cannot be used with this function call.

PARAMETERS

`channel` configurable I/O channel being set up for PPM, 0–29 (DIO0–DIO29)

`frequency` PPM frequency in Hz (should be from 2 Hz to 50 kHz); use -1 to preserve the existing frequency on the RIO block

`offset` PPM offset (should be from 0 to 100%); use -1 and **`bind_offset`** parameter to use bound edge to set the offset (an offset above 100.0% will be set to 100.0%)

NOTE: A zero offset will produce the smallest offset possible, which is one count. If you must have a zero offset, use **`setPWM()`** instead of **`setPPM()`**.

`duty` PPM duty cycle (should be from 0 to 100%); use -1 and **`bind_duty`** parameter to use bound edge to set the duty cycle (a PPM duty cycle above 100.0% will be set to 100.0%)

NOTE: PPM will not wrap around the PPM period. If **`offset`** is set to 25%, the 75 to 100% duty cycle will have the same effect as **`offset`** = 25%, **`duty`** = 75%. The same waveform as a wrapped PPM can be created using an inverted PPM

`invert` whether the PPM output is inverted; the PPM output normally starts with the output low, goes high at the offset, and stays high for the remainder of the duty cycle; inverted will start with the output high, goes low at the offset, and stays low for the duration of the duty cycle.

0 — noninverted
1 — inverted

`bind_offset` use **`BL_BIND_LEAD`** or **`BL_BIND_TRAIL`** to enable binding for the leading edge of the PPM signal to another PWM or PPM output on a channel hosted by same RIO chip and block. Bindings allow PWM and PPM outputs to align their leading and trailing edges.

setPPM (continued)

bind_duty use **BL_BIND_LEAD** or **BL_BIND_TRAIL** to enable binding for the trailing edge of the PPM signal to another PWM or PPM output on a channel hosted by same RIO chip and block

RETURN VALUE

0 — success.

-EINVAL — invalid parameter value.

-EPERM — pin type does not permit this function.

-EACCES — resource needed by this function is not available.

-EFAULT — internal data fault detected.

positive number — Mode Conflict — the positive number is a bitmap that corresponds to the pins on a particular block of a RIO chip that have not been configured to support this function call. Appendix D provides the details of the pin and block associations to allow you to identify the channels that need to be reconfigured to support this function call.

SEE ALSO

brdInit, setFreq, setOffset, setDuty, setToggle, setSync, pulseDisable

setFreq

```
int setFreq(int channel, float frequency);
```

FUNCTION DESCRIPTION

Sets the frequency of all the PWM or PPM outputs on the same block as the channel. Will preserve the duty cycle and offset percentages for all of the channels on the same block.

This function call is for the configurable I/O channels only.

Repeated calls to this function by itself may cause the duty cycle and offset values to drift. If this drift is of concern, call **`setOffset()`** and **`setDuty()`** to reset the duty cycle and offset to the desired value.

NOTE: Configurable I/O channels DIO30 and DIO31 do not support PWM/PPM functionality, and cannot be used with this function call.

PARAMETERS

`channel`	all channels on the same RIO chip and block as this channel (0–29, DIO0–DIO29) will have their frequency set. Duty cycle and offset percentages will be maintained.
`frequency`	frequency of the PWM and PPM outputs (should be from 2 Hz to 50 kHz)

RETURN VALUE

0 — success.

`-EINVAL` — invalid parameter value.

SEE ALSO

`brdInit, setPWM, setPPM, setOffset, setDuty, setToggle, setSync`

setDuty

```
int setDuty(int channel, float duty);
```

FUNCTION DESCRIPTION

Sets the duty cycle of the PWM or PPM output on a configurable I/O channel. Will affect any PWM/PPM that has been bound to this channel's PWM/PPM.

NOTE: Configurable I/O channels DIO30 and DIO31 do not support PWM/PPM functionality, and cannot be used with this function call.

PARAMETERS

`channel`	channel that is getting its duty cycle set, 0–29 (DIO0–DIO29)
`duty`	duty cycle of the PWM/PPM output (should be from 0 to 100%, a duty cycle above 100.0% will be set to 100.0%)

RETURN VALUE

0 — success.

`-EINVAL` — invalid parameter value.

`-EPERM` — channel function does not permit this operation.

SEE ALSO

`brdInit, setPWM, setPPM, setOffset, setFreq, setToggle, setSync`

setOffset

```
int setOffset(int channel, float offset);
```

FUNCTION DESCRIPTION

Sets the offset of a PPM output on a configurable I/O channel. Will affect any PWM/PPM output that has been bound to this channel's PPM signal.

NOTE: Configurable I/O channels DIO30 and DIO31 do not support PWM/PPM functionality, and cannot be used with this function call.

PARAMETERS

channel	channel that is getting its offset set, 0–29 (DIO0–DIO29)
duty	PPM offset (should be from 0 to 100%, an offset above 100.0% will be set to 100.0%)

NOTE: A zero offset will produce the smallest offset possible, which is one count. If you must have a zero offset, use **`setPWM()`** instead of **`setOffset()`**.

RETURN VALUE

0 — success.

`-EINVAL` — invalid parameter value.

`-EPERM` — channel function does not permit this operation.

SEE ALSO

`brdInit, setPWM, setPPM, setFreq, setDuty, setToggle, setSync`

pulseDisable

```
int pulseDisable(int channel, int state);
```

FUNCTION DESCRIPTION

Disables a PWM/PPM output and sets the output to **`state`**. The pin can be restored to the same PWM/PPM operation as before by calling **`pulseEnable()`**.

PARAMETERS

`channel`	channel that is getting its PWM/PPM disabled, 0–29 (DIO0–DIO29)
`state`	state that the digital output will be set to (0 or 1)

RETURN VALUE

0 — success.

`-EINVAL` — invalid parameter value.

`-EPERM` — channel function does not permit this operation.

SEE ALSO

`brdInit, setPWM, setPPM, pulseEnable`

pulseEnable

```
int pulseEnable(int channel);
```

FUNCTION DESCRIPTION

Enables a disabled PWM/PPM output. The pin is restored to the same PWM/PPM operation it had before being disabled.

PARAMETER

`channel`	channel that is getting its PWM/PPM enabled, 0–29 (DIO0–DIO29)

RETURN VALUE

0 — success.

`-EINVAL` — invalid parameter value.

`-EPERM` — channel function does not permit this operation.

SEE ALSO

`brdInit, setPWM, setPPM, pulseDisable`

4.4.3 High-Current Outputs

digOutConfig_H

`int digOutConfig_H(char configuration);`

FUNCTION DESCRIPTION

Sets the configuration of a high-current output to be a sinking or sourcing type output. Upon configuration, the output will be set initially to a high-impedance tristate.

NOTE: Configuring a given output channel for tristate operation using the **`digOutTriStateConfig()`** function call will temporarily override the configuration set by the **`digOutConfig_H()`** function call as long as it is kept a tristate channel. This configuration can also be overridden by setting the channel as a PWM or PPM output.

PARAMETER

`configuration` configuration byte to configure output channels HOUT0–HOUT7 as sinking or sourcing outputs.

Each bit corresponds to one of the following high-current outputs.

Bit 7 = high-current output channel HOUT7
Bit 6 = high-current output channel HOUT6
Bit 5 = high-current output channel HOUT5
Bit 4 = high-current output channel HOUT4
Bit 3 = high-current output channel HOUT3
Bit 2 = high-current output channel HOUT2
Bit 1 = high-current output channel HOUT1
Bit 0 = high-current output channel HOUT0

The high-current outputs are configured to be sinking or sourcing outputs by setting the corresponding bit to 0 or 1: 0 = sinking, 1 = sourcing.

EXAMPLE

```
configuration = 0x26; // 0 0 1 0 0 1 1 0
                      // HOUT7-HOUT6 = Sinking
                      // HOUT5 = Sourcing
                      // HOUT4-HOUT3 = Sinking
                      // HOUT2-HOUT1 = Sourcing
                      // HOUT0 = Sinking
```

RETURN VALUE

0 — success.

`-EINVAL` — board initialization not performed.

SEE ALSO

`brdInit, digOut_H, digOutTriStateConfig_H`

digOut_H

```
int digOut_H(int channel, int state);
```

FUNCTION DESCRIPTION

Sets the state of the selected high-current output channel to a logic 0, logic 1, or high-impedance tristate output.

PARAMETERS

`channel` high-current output pins 0 to 7 (HOUT0–HOUT7)

`state` sets a given channel to one of the following output states depending on how the output was configured by the **`digOutConfig_H()`** function call.

Sinking configuration:

0 — connects the load to GND
1 — puts the output in a high-impedance tristate.

Sourcing configuration:

0 — connects the load in a high-impedance tristate
1 — connects the load to +K1 or + K2.

RETURN VALUE

0 — success.

`-EINVAL` — if not configured correctly or invalid parameter.

SEE ALSO

`brdInit, digOutConfig_H`

digOutTriStateConfig_H

`int digOutTriStateConfig_H(char configuration);`

FUNCTION DESCRIPTION

Allows configuration of a high-current output to be a tristate type output. Upon configuration, the output will be initially set to a high-impedance state.

NOTE: Configuring a given output channel for tristate operation using the **`digOutTriStateConfig()`** function call will temporarily override the configuration set by the **`digOutConfig_H()`** function call as long as it is kept a tristate channel. This configuration can also be overridden by setting the channel as a PWM or PPM output.

PARAMETER

`configuration` configuration byte to configure output channels HOUT0–HOUT7 as tristate outputs.

Each bit corresponds to one of the following high-current outputs.

Bit 7 = high-current output channel HOUT7
Bit 6 = high-current output channel HOUT6
Bit 5 = high-current output channel HOUT5
Bit 4 = high-current output channel HOUT4
Bit 3 = high-current output channel HOUT3
Bit 2 = high-current output channel HOUT2
Bit 1 = high-current output channel HOUT1
Bit 0 = high-current output channel HOUT0

The high-current outputs are configured to be tristate outputs by setting the corresponding bit to 0 or 1: 0 = disable tristate operation, 1 = enable tristate operation.

EXAMPLE

```
configuration = 0x59; // 0 1 0 1 1 0 0 1
                      // HOUT7 = Tristate disabled
                      // HOUT6 = Tristate enabled
                      // HOUT5 = Tristate disabled
                      // HOUT4-HOUT3 = Tristate enabledg
                      // HOUT2-HOUT1 = Tristate disabled
                      // HOUT0 = Tristate enabled
```

RETURN VALUE

0 — success.

`-EINVAL` — board initialization not performed.

SEE ALSO

`brdInit, digOutTriState_H, digOutConfig_H`

digOutTriState_H

```
int digOutTriState_H(int channel, int state)
```

FUNCTION DESCRIPTION

Sets the state of the high-current output channel to a logic 0, logic 1, or high-impedance tristate.

PARAMETERS

channel high-current output pins 0 to 7 (HOUT0–HOUT7)

state sets a given channel to one of the following output states as long as it has been enable as a tristate output by the **digOutTriStateConfig_H()** function call.

Tristate configuration:

0 — connects the load to GND
1 — connects the load to +K1 or + K2.
2 — puts the output in a high-impedance tristate

RETURN VALUE

0 — success.

-EINVAL — invalid parameter or channel not configured for tristate.

SEE ALSO

brdInit, digOutTriStateConfig_H

setPWM_H

```
int setPWM_H(int channel, float frequency, float duty,
   char mode, char bind);
```

FUNCTION DESCRIPTION

Sets up a PWM output on the selected high-current output channel with the specified frequency and duty cycle. The PWM output can be set to any of its three states during either phase of the PWM signal. The PWM channel duty cycle can be bound to a PWM/PPM on another channel on the same RIO chip block so that they share an edge.

PARAMETERS

channel high-current output channel being set up for PWM, 0–7 (HOUT0–HOUT7)

frequency PWM frequency in Hz (should be from 2 Hz to 50 kHz); use -1 to preserve the existing frequency on the RIO block

duty PWM duty cycle (should be from 0 to 100%); use -1 and **bind** parameter to use bound edge to set the duty cycle (a duty cycle above 100.0% will be set to 100.0%)

mode sets the normal or begin state and the pulsed state of the PWM output using these macros:

- **HCPWM_TRI_LOW** — normally tristated and pulsed to sinking
- **HCPWM_TRI_HIGH** — normally tristated and pulsed to sourcing
- **HCPWM_LOW_HIGH** — normally sinking and pulsed to sourcing
- **HCPWM_HIGH_LOW** — normally sourcing and pulsed to sinking
- **HCPWM_LOW_TRI** — normally sinking and pulsed to tristate
- **HCPWM_HIGH_TRI** — normally sourcing and pulsed to tristate

bind use **BL_BIND_LEAD** or **BL_BIND_TRAIL** to enable binding for the leading edge of the PWM output on this channel to another PWM or PPM output on a channel hosted by same RIO chip and block. Bindings allow PWM and PPM outputs to align their leading and trailing edges.

RETURN VALUE

0 — success.

-EINVAL — invalid parameter value.

-EPERM — pin type does not permit this function.

-EACCES — resource needed by this function is not available.

-EFAULT — internal data fault detected.

positive number — Mode Conflict — the positive number is a bitmap that corresponds to the pins on a particular block of a RIO chip that have not been configured to support this function call. Appendix D provides the details of the pin and block associations to allow you to identify the channels that need to be reconfigured to support this function call.

SEE ALSO

brdInit, setFreq_H, setDuty_H, setToggle_H, setSync_H

setPPM_H

```
int setPPM_H(int channel, float frequency, float offset,
   float duty, char mode, char bind_offset, char bind_duty);
```

FUNCTION DESCRIPTION

Sets up a PPM output on the selected high-current output channel with the specified frequency, offset, and duty cycle. The PPM output can be set to any of its three states during either phase of the PPM signal. The offset and duty of the PPM can be bound to a PWM/PPM on another channel on the same RIO block so that they share an edge.

PARAMETERS

channel high-current output channel being set up for PPM, 0–7 (HOUT0–HOUT7)

frequency PPM frequency in Hz (should be from 2 Hz to 50 kHz); use -1 to preserve the existing frequency on the RIO block

offset PPM offset (should be from 0 to 100%); use -1 and **bind_offset** parameter to use bound edge to set the offset (an offset above 100.0% will be set to 100.0%)

NOTE: A zero offset will produce the smallest offset possible, which is one count. If you must have a zero offset, use **setPWM_H()** instead of **setPPM_H()**.

duty PPM duty cycle (should be from 0 to 100%); use -1 and **bind_duty** parameter to use bound edge to set the duty cycle (a PPM duty cycle above 100.0% will be set to 100.0%)

set to 25%, duty in range 75-100% will have the same effect

NOTE: PPM will not wrap around the PPM period. If **offset** is set to 25%, the 75 to 100% duty cycle will have the same effect as **offset** = 25%, **duty** = 75%. The same waveform as a wrapped PPM can be created using an inverted PPM

mode sets the normal or begin state and the pulsed state of the PPM output using these macros:

- **HCPWM_TRI_LOW** — normally tristated and pulsed to sinking
- **HCPWM_TRI_HIGH** — normally tristated and pulsed to sourcing
- **HCPWM_LOW_HIGH** — normally sinking and pulsed to sourcing
- **HCPWM_HIGH_LOW** — normally sourcing and pulsed to sinking
- **HCPWM_LOW_TRI** — normally sinking and pulsed to tristate
- **HCPWM_HIGH_TRI** — normally sourcing and pulsed to tristate

bind_offset use **BL_BIND_LEAD** or **BL_BIND_TRAIL** to enable binding for the leading edge of the PPM signal to another PWM or PPM output on a channel hosted by same RIO chip and block. Bindings allow PWM and PPM outputs to align their leading and trailing edges.

setPPM_H (continued)

bind_duty use **BL_BIND_LEAD** or **BL_BIND_TRAIL** to enable binding for the trailing edge of the PPM signal to another PWM or PPM output on a channel hosted by same RIO chip and block

RETURN VALUE

0 — success.

-EINVAL — invalid parameter value.

-EPERM — pin type does not permit this function.

-EACCES — resource needed by this function is not available.

-EFAULT — internal data fault detected.

positive number — Mode Conflict — the positive number is a bitmap that corresponds to the pins on a particular block of a RIO chip that have not been configured to support this function call. Appendix D provides the details of the pin and block associations to allow you to identify the channels that need to be reconfigured to support this function call.

SEE ALSO

brdInit, setFreq_H, setOffset_H, setDuty_H, setToggle_H, setSync_H

setFreq_H

```
int setFreq_H(int channel, float frequency);
```

FUNCTION DESCRIPTION

Sets the frequency of all the PWM or PPM outputs on the same block as the channel. Will preserve the duty cycle and offset percentages for all of the channels on the same block.

This function call is for the high-current output channels only.

Repeated calls to this function by itself may cause the duty cycle and offset values to drift. If this drift is of concern, call **setOffset_H()** and **setDuty_H()** to reset the duty cycle and offset to the desired value.

PARAMETERS

channel	all channels on the same RIO chip and block as this channel (0–7, HOUT0–HOUT7) will have their frequency set. Duty cycle and offset percentages will be maintained.
frequency	frequency of the PWM and PPM outputs (should be from 2 Hz to 50 kHz)

RETURN VALUE

0 — success.

-EINVAL — invalid parameter value.

SEE ALSO

brdInit, setPWM_H, setPPM_H, setOffset_H, setDuty_H, setToggle_H, setSync_H

setDuty_H

```
int setDuty_H(int channel, float duty);
```

FUNCTION DESCRIPTION

Sets the duty cycle of the PWM or PPM output on a high-current output channel. Will affect any PWM/PPM that has been bound to this channel's PWM/PPM.

PARAMETERS

channel channel that is getting its duty cycle set, 0–7 (HOUT0–HOUT7)

duty duty cycle of the PWM/PPM output (should be from 0 to 100%, a duty cycle above 100.0% will be set to 100.0%)

RETURN VALUE

0 — success.

-EINVAL — invalid parameter value.

-EPERM — channel function does not permit this operation.

SEE ALSO

brdInit, setPWM_H, setPPM_H, setOffset_H, setFreq_H, setToggle_H, setSync_H

setOffset_H

```
int setOffset_H(int channel, float offset);
```

FUNCTION DESCRIPTION

Sets the offset of a PPM output on a high-current output channel. Will affect any PWM/PPM output that has been bound to this channel's PPM signal.

PARAMETERS

channel	channel that is getting its offset set, 0–7 (HOUT0–HOUT7)
duty	PPM offset (should be from 0 to 100%, an offset above 100.0% will be set to 100.0%)

NOTE: A zero offset will produce the smallest offset possible, which is one count. If you must have a zero offset, use **setPWM_H()** instead of **setOffset_H()**.

RETURN VALUE

0 — success.

-EINVAL — invalid parameter value.

-EPERM — channel function does not permit this operation.

SEE ALSO

brdInit, setPWM_H, setPPM_H, setFreq_H, setDuty_H, setToggle_H, setSync_H

setSync_H

```
int setSync_H(int channel, int edge);
```

FUNCTION DESCRIPTION

Enables or disables the global synch for the block the high-current output channel is associated with.

PARAMETERS

channel channel that is on the block that will have its synch set

edge edge of the synch signal (0 will disable the synch).

BL_EDGE_RISE — synchronize event on rising edge
BL_EDGE_FALL — synchronize event on falling edge
BL_EDGE_BOTH — synchronize events on both edges

RETURN VALUE

0 — success.

-EINVAL — invalid parameter value.

SEE ALSO

brdInit, setPWM_H, setPPM_H

4.4.4 Rabbit RIO Interrupt Handlers

addISR

```
int addISR(int channel, int io, int ier, void (*handler)());
```

FUNCTION DESCRIPTION

Adds an interrupt handler for the interrupts specified in the **ier** parameter for the given RIO block hosting the given configurable I/O pin. The interrupt service routine (ISR) is always disabled when created. Call **enableISR()** to enable the interrupt service routine. The ISR handler function is responsible for clearing the interrupt(s) within the hosting RIO block when called.

PARAMETERS

channel configurable I/O channel to bind to ISR, 0–31

io **BL_INPUT_BLOCK** for input block
BL_OUTPUT_BLOCK for output block

ier bit mask of interrupt(s) this handler services

- **BL_IER_DQE** — decrement/quadrature/end
- **BL_IER_IIB** — increment/inphase/begin
- **BL_IER_ROLL_D** — counter rollover on decrement
- **BL_IER_ROLL_I** — counter rollover on increment
- **BL_IER_MATCH3** — Match 3 condition
- **BL_IER_MATCH2** — Match 2 condition
- **BL_IER_MATCH1** — Match 1 condition
- **BL_IER_MATCH0** — Match 0 condition

handler pointer to the interrupt service function

RETURN VALUE

Success — returns the handler ID number (**0..RSB_MAX_ISR-1**).

-EINVAL— Invalid parameter given.

-ENOSPC — No more room in ISR table (increase **RSB_MAX_ISR**).

SEE ALSO

addISR_H, tickISR, enableISR, setIER

addISR_H

```
int addISR_H(int channel, int ier, void (*handler)());
```

FUNCTION DESCRIPTION

Adds an interrupt handler for the interrupts specified in the **ier** parameter for the given RIO block hosting the given high-current output pin. The interrupt service routine (ISR) is always disabled when created. Call **enableISR** to enable the ISR. The ISR handler given is responsible for clearing the interrupt(s) within the hosting RIO block.

PARAMETERS

channel high-current output channel to bind to ISR, 0–7

ier bit mask of interrupt(s) this handler services:

- **BL_IER_DQE** — decrement/quadrature/end
- **BL_IER_IIB** — increment/inphase/begin
- **BL_IER_ROLL_D** — counter rollover on decrement
- **BL_IER_ROLL_I** — counter rollover on increment
- **BL_IER_MATCH3** — Match 3 condition
- **BL_IER_MATCH2** — Match 2 condition
- **BL_IER_MATCH1** — Match 1 condition
- **BL_IER_MATCH0** — Match 0 condition

handler pointer to the interrupt service function

RETURN VALUE

Success — returns the handler ID number (**0..RSB_MAX_ISR-1**).

-EINVAL— Invalid parameter given.

-ENOSPC — No more room in ISR table (increase **RSB_MAX_ISR**).

SEE ALSO

addISR, tickISR, enableISR, setIER

setIER

```
int setIER(int isr_handle, int ier);
```

FUNCTION DESCRIPTION

Sets the Interrupt Enable Register (IER) mask for an interrupt handler. Note that the interrupt handler must be currently disabled to set the IER value. Disabling the ISR can be done by calling **`enableISR()`** with a zero for the **`enable`** parameter.

PARAMETERS

`isr_handle`	index to the desired ISR
`ier`	bit mask of interrupts this handler services (bit positions match RIO Interrupt Enable and Status registers)

RETURN VALUE

0 — success

`-EINVAL`— Invalid parameter given.

`-EPERM` — Handler is enabled, can't change IER.

SEE ALSO

`addISR, addISR_H, enableISR, tickISR`

enableISR

```
int enableISR(int isr_handle, int enable)
```

FUNCTION DESCRIPTION

Enables or disables an interrupt handler.

PARAMETERS

isr_handle	index to the desired ISR
enable	non-zero enables the ISR, zero disables the ISR

RETURN VALUE

0 — success.

-EINVAL— invalid parameter given.

SEE ALSO

addISR, addISR_H, setIER, tickISR

tickISR

```
void tickISR(void)
```

FUNCTION DESCRIPTION

Polls the RIO chip(s) for ISR events if interrupts are not being used. Any enabled ISR events will be passed to the appropriate ISR handler.

RETURN VALUE

None.

SEE ALSO

addISR, addISR_H, enableISR, setIER

4.4.5 Serial Communication

Library files included with Dynamic C provide a full range of serial communications support. The **RS232.LIB** library provides a set of circular-buffer-based serial functions. The **PACKET.LIB** library provides packet-based serial functions where packets can be delimited by the 9th bit, by transmission gaps, or with user-defined special characters. Both libraries provide blocking functions, which do not return until they are finished transmitting or receiving, and nonblocking functions, which must be called repeatedly until they are finished. For more information, see the ***Dynamic C User's Manual*** and Technical Note 213, ***Rabbit Serial Port Software***.

Use the following function calls with the BL4S200.

serMode

```
int serMode(int mode);
```

FUNCTION DESCRIPTION

This function call sets the serial interfaces used by your application program. Call this function after executing **serXopen()** and before using any other serial port function calls.

PARAMETER

mode the defined serial port configuration

Mode	BL4S200, BL5S220, and BL4S230 Models Only		
	Serial Port C	Serial Port E	Serial Port F
0	RS-485	RS-232, 3-wire	RS-232, 3-wire
1	RS-485	RS-232, 5-wire	RTS/CTS

The mode parameter has no effect on the BL4S210 model, which is configured in hardware for RS-485 on Serial Port C and one 3-wire RS-232 channel on Serial Port B.

RETURN VALUE

0 if valid mode selected, **-EINVAL** if not.

SEE ALSO

ser485Tx, ser485Rx

ser485Tx

```
void ser485Tx(void);
```

FUNCTION DESCRIPTION

Enables the RS-485 transmitter. **serMode()** must be executed before running this function call.

NOTE: Transmitted data are echoed back into the receive data buffer. The echoed data could be used to identify when to disable the transmitter by using one of the following methods.

Byte mode—disables the transmitter after the byte that is transmitted is detected in the receive data buffer.

Block data mode—disable the transmitter after the same number of bytes transmitted are detected in the receive data buffer.

RETURN VALUE

None.

SEE ALSO

brdInit, serMode, ser485En

ser485Rx

```
void ser485Rx(void);
```

FUNCTION DESCRIPTION

Disables the RS-485 transmitter. This puts you in listen mode, which allows you to receive data from the RS-485 interface. **serMode()** must be executed before running this function call.

RETURN VALUE

None.

SEE ALSO

brdInit, serMode, ser485Tx

4.4.6 A/D Converter Inputs

anaInConfig

```
void anaInConfig(int channel, int opmode);
```

FUNCTION DESCRIPTION

Configures an A/D converter input channel for a given mode of operation. This function must be called before accessing the A/D converter chip.

The configuration of the A/D converter is complicated because channels AIN0–AIN3 are offset independently, but channels AIN4–AIN7 are biased in pairs. When configured for the differential mode, the A/D converter will return differential readings for all the channel pairs indicated below, with calibration constants reducing the effect of any bias differences.

AIN0 — biased by D/A converter internal channel 2
AIN1 — biased by D/A converter internal channel 3

AIN2 — biased by D/A converter internal channel 4
AIN3 — biased by D/A converter internal channel 5

AIN4 — biased by D/A converter internal channel 6
AIN5 — biased by D/A converter internal channel 6

AIN6 — biased by D/A converter internal channel 7
AIN7 — biased by D/A converter internal channel 7

When the differential mode is selected, this function call configures both the selected channel and its differential mate. The differential mode will always configure pairs. For all of the pairs indicated above, both or neither will be configured for the differential mode, depending on the mode selected for the channel being configured.

The AIN4–AIN7 pairs are allowed to be configured as paired differential mode or as either unipolar or bipolar single-ended, but because the AIN4–AIN7 pairs share a D/A converter bias channel, if a pair has mismatched configurations, they will incur extra delays as the common D/A converter offset switches with reads from each. The same is true if they are both bipolar single-ended, but are read with different gains

NOTE: If you plan to configure the D/A converter chip using **`anaOutConfig`**, you must call **`anaOutConfig()`** before executing **`anaInConfig()`**. This is because the A/D converter uses internal channels 2–7 on the D/A converter chip to bias the A/D converter input circuit.

anaInConfig (continued)

PARAMETERS

channel analog input channel, 0–7 (AIN0–AIN7)

opmode selects the mode of operation for the A/D converter channel pair. The values are as follows:

- **SE0_MODE** — single-ended unipolar (0–20 V)
- **SE1_MODE** — single-ended bipolar (±10 V)
- **DIFF_MODE** — differential bipolar (±20 V)
- **mAMP_MODE** — 4–20 mA operation

RETURN VALUE:

0 — success.

BL_SPIBUSY — SPI port busy.

-EINVAL — invalid parameter.

SEE ALSO

brdInit, anaInCalib, anaIn, anaInVolts, anaInmAmps, anaInDiff

anaInCalib

```
int anaInCalib(int channel, int opmode, int gaincode,
    int value1, float volts1, int value2, float volts2);
```

FUNCTION DESCRIPTION

Calibrates the response of a given A/D converter channel as a linear function using the two conversion points provided. Gain and offset constants are calculated and placed into flash memory.

NOTE: The 10 and 90% points of the maximum voltage range are recommended when calibrating a channel.

PARAMETERS

`channel` analog input channel number (0 to 7) corresponding to AIN0–AIN7

`channel`	Single-Ended	Differential	4–20 mA
0	+AIN0	+AIN0 -AIN1	+AIN0
1	+AIN1	—	+AIN1
2	+AIN2	+AIN2 -AIN3	+AIN2
3	+AIN3	—	+AIN3
4	+AIN4	+AIN4 -AIN5	
5	+AIN5	—	
6	+AIN6	+AIN6 -AIN7	
7	+AIN7	—	

`opmode` the mode of operation for the specified channel. Use one of the following macros to set the mode for the channel being configured.

`SE0_mode` = single-ended unipolar (0–20 V)
`SE1_mode` = single-ended bipolar (±10 V)
`DIFF_MODE` = differential bipolar (±20 V)
`mAMP_mode` = 4–20 mA

gaincode the gain code of 0 to 7 (use a gain code of 4 for 4–20 mA operation)

Gain Code	Macro	Voltage Range		
		Single-Ended Unipolar	Single-Ended Bipolar	Differential Bipolar
0	GAIN_X1	0–20 V	±10 V	±20 V
1	GAIN_X2	0–10 V	±5 V	±10 V
2	GAIN_X4	0–5 V	±2.5 V	±5 V
3	GAIN_X5	0–4 V	±2 V	±4 V
4	GAIN_X8	0–2.5 V	±1.25 V	±2.5 V
5	GAIN_X10	0–2 V	±1 V	±2 V
6	GAIN_X16	0–1.25 V	—	±1.25 V
7	GAIN_X20	0–1 V	—	±1 V

value1 the first A/D converter value

volts1 the voltage corresponding to the first A/D converter value

value2 the second A/D converter value

volts2 the voltage corresponding to the second A/D converter value

RETURN VALUE

0 — success.

-EINVAL — invalid parameter.

-ERR_ANA_CALIB — error writing calibration constants.

SEE ALSO

brdInit, anaInConfig, anaIn, anaInmAmps, anaInDiff, anaInVolts

anaIn

```
int anaIn(int channel, int gaincode);
```

FUNCTION DESCRIPTION

Reads the state of an A/D converter input channel. If the access is for an A/D converter single-ended bipolar channel and the gain code for the given channel has changed from the previous cycle, the user block in the flash memory will be read to get the calibration constants for the new gain value.

PARAMETERS

channel analog input channel number (0 to 7) corresponding to AIN0–AIN7

channel	Single-Ended	Differential	4–20 mA
0	+AIN0	+AIN0 -AIN1	+AIN0
1	+AIN1	—	+AIN1
2	+AIN2	+AIN2 -AIN3	+AIN2
3	+AIN3	—	+AIN3
4	+AIN4	+AIN4 -AIN5	
5	+AIN5	—	
6	+AIN6	+AIN6 -AIN7	
7	+AIN7	—	

gaincode the gain code of 0 to 7 (use a gain code of 4 for 4–20 mA operation)

Gain Code	Macro	Voltage Range		
		Single-Ended Unipolar	Single-Ended Bipolar	Differential Bipolar
0	GAIN_X1	0–20 V	±10 V	± 20 V
1	GAIN_X2	0–10 V	±5 V	± 10 V
2	GAIN_X4	0–5 V	±2.5 V	± 5 V
3	GAIN_X5	0–4 V	±2 V	± 4 V
4	GAIN_X8	0–2.5 V	±1.25 V	± 2.5 V
5	GAIN_X10	0–2 V	±1 V	± 2 V
6	GAIN_X16	0–1.25 V	—	± 1.25 V
7	GAIN_X20	0–1 V	—	± 1 V

RETURN VALUE

A value corresponding to the voltage on the analog input channel:

0–2047 for 11-bit A/D conversions,
or a value of **BL_ERRCODESTART** or less to indicate an error condition:

A/D converter operation errors (will not create run-time error):

BL_SPIBUSY
BL_TIMEOUT
BL_OVERFLOW
BL_WRONG_MODE

System errors (can create run-time error unless disabled):

-ERR_ANA_INVAL — invalid parameter value.

SEE ALSO

brdInit, anaInConfig, anaInCalib, anaInmAmps, anaInDiff, anaInVolts

anaInVolts

```
float anaInVolts(int channel, int gaincode);
```

FUNCTION DESCRIPTION

Reads the state of a single-ended A/D converter input channel and uses the previously set calibration constants to convert it to volts. The voltage ranges given in the table below are nominal ranges that will be returned. However, values outside these ranges can often be seen before the return of a **BL_OVERFLOW** error.

If the gain code for a given channel has changed from the previous cycle, the following code accesses will occur.

1. The user block will be read to get the calibration constants for the new gain value.
2. The D/A converter will be written to bias the A/D converter input circuit for proper operation. (The D/A converter access only applies for the single-ended bipolar A/D converter operation.)

PARAMETERS

channel analog input channel number (0 to 7) corresponding to AIN0–AIN7

gaincode the gain code of 0 to 7; the table below applies for single-ended modes only

Gain Code	Macro	Voltage Range	
		Single-Ended Unipolar	Single-Ended Bipolar
0	**GAIN_X1**	0–20 V	±10 V
1	**GAIN_X2**	0–10 V	±5 V
2	**GAIN_X4**	0–5 V	±2.5 V
3	**GAIN_X5**	0–4 V	±2 V
4	**GAIN_X8**	0–2.5 V	±1.25 V
5	**GAIN_X10**	0–2 V	±1 V
6	**GAIN_X16**	0–1.25 V	—
7	**GAIN_X20**	0–1 V	—

RETURN VALUE

A voltage on the analog input channel, or a value of **BL_ERRCODESTART** or less to indicate an error condition:

A/D converter operation errors (will not create run-time error):

BL_NOT_CAL — A/D converter is not calibrated for this channel/gain.
BL_OVERFLOW — A/D converter overflow.
BL_SPIBUSY — shared SPI port is already in use.
BL_TIMEOUT — A/D converter timeout.
BL_WRONG_MODE — A/D converter is in wrong mode (run **anaInConfig()**).

System errors (can create run-time error unless disabled):

-ERR_ANA_CALIB — fault detected in reading calibration facto.r
-ERR_ANA_INVAL — invalid parameter value.

SEE ALSO

brdInit, anaInConfig, anaIn, anaInmAmps, anaInDiff, anaInCalib

anaInDiff

```
float anaInDiff(int channel, int gaincode);
```

FUNCTION DESCRIPTION

Reads the state of a differential A/D converter input channel and uses the previously set calibration constants to convert it to volts. Voltage ranges given in the table below are the nominal ranges that will be returned. However, values outside these ranges can often be seen before the return of a **BL_OVERFLOW** error.

If the gain code for a given channel has changed from the previous cycle, the user block will be read to get the calibration constants for the new gain value.

PARAMETERS

channel the analog input channel number (0, 2, 4, 6) as shown below

channel	Differential Inputs
0	+AIN0 -AIN1
2	+AIN2 -AIN3
4	+AIN4 -AIN5
6	+AIN6 -AIN7

gaincode the gain code of 0 to 7

Gain Code	Macro	Actual Gain	Differential Voltage Range	Actual Voltage Range
0	GAIN_X1	×1	± 20 V	± 10 V
1	GAIN_X2	×2	± 10 V	± 5 V
2	GAIN_X4	×4	± 5 V	± 2.5 V
3	GAIN_X5	×5	± 4 V	± 2 V
4	GAIN_X8	×8	± 2.5 V	± 1.25 V
5	GAIN_X10	×10	± 2 V	± 1 V
6	GAIN_X16	×16	± 1.25 V	± 0.625 V
7	GAIN_X20	×20	± 1 V	± 0,5 V

anaInDiff (continued)

RETURN VALUE

A voltage on the analog input channel, or a value of **BL_ERRCODESTART** or less to indicate an error condition:

A/D converter operation errors (will not create run-time error):

BL_NOT_CAL — A/D converter is not calibrated for this channel/gain.
BL_OVERFLOW — A/D converter overflow.
BL_SPIBUSY — shared SPI port is already in use.
BL_TIMEOUT — A/D converter timeout.
BL_WRONG_MODE — A/D converter is in wrong mode (run **anaInConfig()**).

System errors (can create run-time error unless disabled):

-ERR_ANA_CALIB — fault detected in reading calibration factor.
-ERR_ANA_INVAL — invalid parameter value.

SEE ALSO

brdInit, anaInConfig, anaIn, anaInmAmps, anaInVolts, anaInCalib

anaInmAmps

```
float anaInmAmps(int channel);
```

FUNCTION DESCRIPTION

Reads the state of a single-ended A/D converter input channel and uses the previously set calibration constants to convert it to a floating-point current value in milli amps. The nominal range is 0 mA to 20 mA, although it is possible to receive values outside this range before a **BL_OVERFLOW** error is returned.

PARAMETER

channel A/D converter input channel (0–3 corresponding to AIN0–AIN3)

RETURN VALUE

A current value corresponding to the current on the analog input channel, or a value of **BL_ERRCODESTART** or less to indicate an error condition:

A/D converter operation errors (will not create run-time error):

BL_NOT_CAL — A/D converter is not calibrated for this channel/gain.
BL_OVERFLOW — A/D converter overflow.
BL_SPIBUSY — shared SPI port is already in use.
BL_TIMEOUT — A/D converter timeou.t
BL_WRONG_MODE — A/D converter is in wrong mode (run **anaInConfig()**).

System errors (can create run-time error unless disabled):

-ERR_ANA_CALIB — fault detected in reading calibration factor.
-ERR_ANA_INVAL — invalid parameter value.

SEE ALSO

brdInit, anaInConfig, anaIn, anaInDiff, anaInVolts, anaInCalib

anaInDriver

```
int anaInDriver(char cmd);
```

FUNCTION DESCRIPTION

Low-level driver to read the ADS7870 A/D converter chip.

PARAMETER

cmd The **cmd** parameter contains a gain code and channel code, and the MSB is set high for direct-mode access. The format is as follows:

D7	D6–D4	D3–D0
1	**gain_code**	**channel_code**

Use the following calculation and tables to determine **cmd**:

```
cmd = 0x80 | (gain_code<<4) + channel_code
```

gain_code	Multiplier
0	1
1	2
2	4
3	5
4	8
5	10
6	16
7	20

channel_code	Differential Input Lines	channel_code	Single-Ended Input Lines	mA Input Lines
0	+AIN0 -AIN1	8	+AIN0	+AIN0
1	+AIN2 -AIN3	9	+AIN1	+AIN1
2	+AIN4 -AIN5	10	+AIN2	+AIN2
3	+AIN6 -AIN7	11	+AIN3	+AIN3
4	Reserved	12	+AIN4	Reserved
5	Reserved	13	+AIN5	Reserved
6	Reserved	14	+AIN6	Reserved
7	Reserved	15	+AIN7	Reserved

anaInDriver (continued)

The BL4S200 boards were designed to extend the A/D converter input circuit configurations, which is done by the **anaInConfig()** function call. The following table maps the BL4S200 A/D converter configurations to the A/D converter **channel_code** listed above:

BL4S200 A/D Converter Input	channel_code
Differential	0–4
Single-Ended Unipolar	8–15
Single-Ended Bipolar	8–15
4–20 mA	8–11

RETURN VALUE

A value corresponding to the voltage on the analog input channel, which will be either in the range [-20480,2047], or an error code of **BL_ERRCODESTART** or less as follows:

BL_SPIBUSY
BL_TIMEOUT
BL_OVERFLOW

4.4.7 D/A Converter Outputs

anaOutConfig

```
int anaOutConfig(char polarity, int mode);
```

FUNCTION DESCRIPTION

Configures the D/A converter chip for a given output voltage range, 0–10 V or ±10 V, and loads the calibration data for use by the D/A converter function calls. This function must be called before accessing any of the D/A converter channels.

NOTE: If you are using the analog outputs, you must configure the D/A converter chip using the **`anaOutConfig()`** function before executing **`anaInConfig()`** to configure the A/D converter chip. This is because the A/D converter chip uses internal channels 2–7 on the D/A converter chip to bias the A/D converter input circuit, and the correct configuration of the A/D converter would be affected if the D/A converter configuration was changed later.

PARAMETERS

`polarity` sets the output configuration polarity as follows:

`DAC_UNIPOLAR` (0) = unipolar operation. (0–10V and 4–20 mA)
`DAC_BIPOLAR` (1) = bipolar operation. (±10V and 4–20 mA)

NOTE: This parameter has no effect when the D/A converter is configured for 4–20 mA channels.

`mode` the mode of operation:

0 = asynchronous—an output is updated at the time data are written to the given channel

1 = synchronous—all outputs are updated with data previously written when the **`anaOutStrobe()`** function is executed.

RETURN VALUE

0 — success.

`BL_SPIBUSY` — SPI port busy.

`-EINVAL` — invalid configuration parameter.

`-ERR_ANA_CALIB` — error reading calibration constants.

SEE ALSO

`brdInit, anaOut, anaOutmAmps, anaOutStrobe, anaOutConfig, anaOutCalib`

anaOutStrobe

```
int anaOutStrobe(int channels);
```

FUNCTION DESCRIPTION

Outputs the previously written value of each channel indicated by the input parameter.

This function is only useful when the D/A converter is configured for synchronous mode operation because each channel is updated immediately in the asynchronous mode when a value is written to it. It is called internally by **`anaInConfig()`** to strobe the D/A converter offsets when the D/A converter is in the synchronous mode, but its normal use in programs should only be to strobe external D/A converter channels 0 and 1.

PARAMETER

`channels` bitmap of channels to be strobed.

1 — Channel 0
2 — Channel 1
3 — Channels 0 and 1

RETURN VALUE

0 — success.

`BL_SPIBUSY` — SPI port busy.

SEE ALSO

`brdInit, anaOut, anaOutmAmps, anaOutConfig, anaOutCalib`

anaOutPwrOff

```
int anaOutPwrOff(BL_POWER_T mode);
```

FUNCTION DESCRIPTION

This function enables or disables the BL4S200 power supply that is used to power the D/A converter voltage or current output circuits.

CAUTION: Do not call this function until you have configured both D/A converter channels to the desired voltage or current operation. Unconfigured D/A converter channels will be set to approx. 0 V or 4 mA.

PARAMETER

mode D/A converter power-off mode.

BL_HIGH_Z (0) — high output impedance
BL_OHM100 (1) — 100 kΩ to GND
BL_OHM2_5 (2) — 2.5 kΩ to GND

RETURN VALUE

0 — success.

-EINVAL — invalid parameter.

SEE ALSO

anaOut, anaOutVolts, anaOutmAmps

anaOutCalib

```
int anaOutCalib(int channel, int calib_index, int value1,
   float volts1, int value2, float volts2);
```

FUNCTION DESCRIPTION

Calibrates the response of a given D/A converter channel as a linear function with using two conversion points provided by the user. Gain and offset constants are calculated and written to the user block in flash memory for use by the D/A converter function calls.

NOTE: The 10 and 90% points of the maximum voltage range are recommended when calibrating a channel.

PARAMETERS

`channel` the D/A converter output channel (0–1) corresponding to AOUT0–AOUT1

`calib_index` index used to go to the proper location in the lookup table for writing the calibration data

0 = 0–10 V calibration data
1 = ±10 V calibration data
2 = 4–20 mA calibration data (unipolar configuration)

`value1` the first D/A converter value (0–4095)

`volts1` the voltage or current corresponding to the first D/A converter value (0–10 V, ±10 V or 4– 20 mA)

`value2` the second D/A converter value (0–4095)

`volts2` the voltage or current corresponding to the second D/A converter value (0–10 V, ±10 V or 4– 20 mA)

RETURN VALUE

0 — success.

`-EINVAL` — invalid parameter.

`-ERR_ANA_CALIB` — error writing calibration constants.

SEE ALSO

`brdInit, anaOut, anaOutVolts, anaOutmAmps, anaOutStrobe, anaOutConfig`

anaOut

```
void anaOut(int ch, int rawdata);
```

FUNCTION DESCRIPTION

Sets the voltage of a D/A converter output channel.

PARAMETERS

`ch`	the D/A converter output channel (0–1) corresponding to AOUT0–AOUT1
`rawdata`	data value corresponding to the voltage desired on the output channel (0–4095). If a value larger than 4095 is given, the channel will be set to maximum (4095).

RETURN VALUE

0 — success.

`BL_SPIBUSY` — SPI port busy.

SEE ALSO

`anaOutDriver, anaOutVolts, anaOutCalib`

anaOutVolts

```
void anaOutVolts(int ch, float voltage);
```

FUNCTION DESCRIPTION

Sets the voltage of a D/A converter output channel by using the previously set calibration constants to calculate the correct data values.

PARAMETERS

ch the D/A converter output channel (0–1) corresponding to AOUT0–AOUT1

voltage the voltage desired on the output channel

RETURN VALUE

0 — success.

BL_SPIBUSY — SPI port busy.

-ERR_ANA_INVAL — invalid config parameter.

-ERR_ANA_CALIB — error reading calibration data.

SEE ALSO

brdInit, anaOut, anaOutStrobe, anaOutConfig, anaOutCalib

anaOutmAmps

```
void anaOutmAmps(int ch, float current);
```

FUNCTION DESCRIPTION

Sets the current of a D/A converter output channel by using the previously set calibration constants to calculate the correct data values.

PARAMETERS

`ch`	the D/A converter output channel (0–1) corresponding to AOUT0–AOUT1
`current`	the current desired on the output channel (valid range is 4–20 mA)

RETURN VALUE

0 — success.

BL_SPIBUSY — SPI port busy.

`-ERR_ANA_CALIB` — error reading calibration data.

SEE ALSO

`brdInit, anaOut, anaOutVolts, anaOutStrobe, anaOutConfig, anaOutCalib`

anaOutDriver

`int anaOutDriver(unsigned int cmd)`

FUNCTION DESCRIPTION

Low-level driver to read the DAC128S085 D/A converter chip. It handles writing the **`rawdata`** output value to the D/A converter chip.

The synch/asynch D/A converter mode is critical for determining whether a strobe needs to follow **`anaOut in _bias_adc()`**, so any mode change is detected here, not relying on that mode to only be changed through the high-level **`anaOutConfig()`** function call.

PARAMETER

`cmd` The **`cmd`** parameter format is as follows:

D15–D12	D11–D0
`channel` (0–7)	**`rawdata value`** (0–4095)

Use the following calculation and tables to determine **`cmd`**:

```
cmd = (channel<<12) | rawdata value
```

RETURN VALUE

0 — success.

`BL_SPIBUSY` — SPI port busy.

4.4.8 SRAM Use

The BL4S200 model and some memory variations described in Table 1 have a battery-backed data SRAM and a program-execution SRAM. Dynamic C provides the **`protected`** keyword to identify variables that are to be placed into the battery-backed SRAM. The compiler generates code that maintains two copies of each protected variable in the battery-backed SRAM. The compiler also generates a flag to indicate which copy of the protected variable is valid at the current time. This flag is also stored in the battery-backed SRAM. When a protected variable is updated, the "inactive" copy is modified, and is made "active" only when the update is 100% complete. This assures the integrity of the data in case a reset or a power failure occurs during the update process. At power-on the application program uses the active copy of the variable pointed to by its associated flag.

The sample code below shows how a protected variable is defined and how its value can be restored.

```
protected nf_device nandFlash;
int main() {
   ...
   _sysIsSoftReset();    // restore any protected variables
```

The **`bbram`** keyword may also be used instead if there is a need to store a variable in battery-backed SRAM without affecting the performance of the application program. Data integrity is ***not*** assured when a reset or power failure occurs during the update process.

Additional information on **`bbram`** and **`protected`** variables is available in the ***Dynamic C User's Manual***.

5. Using the Ethernet TCP/IP Features

Chapter 5 discusses using the Ethernet TCP/IP features on the BL4S200 boards. Ethernet is ***not*** available on BL5S220 and BL4S230 models, which have wireless network interfaces.

5.1 TCP/IP Connections

Before proceeding you will need to have the following items.

- If you don't have Ethernet access, you will need at least a 10Base-T Ethernet card (available from your favorite computer supplier) installed in a PC.
- Two RJ-45 straight-through CAT 5/6 Ethernet cables and a hub, or an RJ-45 crossover CAT 5/6 Ethernet cable.

The CAT 5/6 Ethernet cables and Ethernet hub are available from Rabbit in a TCP/IP tool kit. More information is available at www.digi.com.

1. Connect the AC adapter and the programming cable as shown in Chapter 2, "Getting Started."
2. Ethernet Connections

If you do not have access to an Ethernet network, use a crossover CAT 5/6 Ethernet cable to connect the BL4S200 to a PC that at least has a 10Base-T Ethernet card.

If you have Ethernet access, use a straight-through CAT 5/6 Ethernet cable to establish an Ethernet connection to the BL4S200 from an Ethernet hub. These connections are shown in Figure 25.

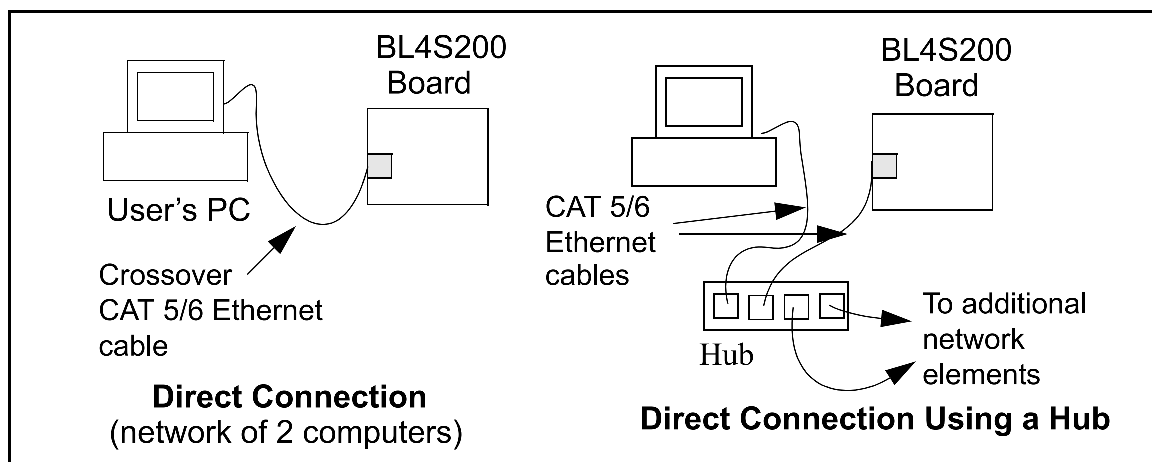


Figure 25. Ethernet Connections

The PC running Dynamic C through the serial programming port on the BL4S200 does not need to be the PC with the Ethernet card.

3. Apply Power

Plug in the AC adapter. The BL4S200 is now ready to be used.

> **NOTE:** A hardware RESET is accomplished by unplugging the AC adapter, then plugging it back in, or by momentarily grounding the board reset input at pin 9 on screw terminal header J2.

When working with the BL4S200, the green **LNK** light is on when a program is running and the board is properly connected either to an Ethernet hub or to an active Ethernet card. The orange **ACT** light flashes each time a packet is received.

5.2 TCP/IP Sample Programs

We have provided a number of sample programs demonstrating various uses of TCP/IP for networking embedded systems. These programs require that you connect your PC and the BL4S200 together on the same network. This network can be a local private network (preferred for initial experimentation and debugging), or a connection via the Internet.

5.2.1 How to Set IP Addresses in the Sample Programs

With the introduction of Dynamic C 7.30 we have taken steps to make it easier to run many of our sample programs. You will see a **TCPCONFIG** macro. This macro tells Dynamic C to select your configuration from a list of default configurations. You will have three choices when you encounter a sample program with the **TCPCONFIG** macro.

1. You can replace the **TCPCONFIG** macro with individual **MY_IP_ADDRESS**, **MY_NETMASK**, **MY_GATEWAY**, and **MY_NAMESERVER** macros in each program.
2. You can leave **TCPCONFIG** at the usual default of 1, which will set the IP configurations to **10.10.6.100**, the netmask to **255.255.255.0**, and the nameserver and gateway to **10.10.6.1**. If you would like to change the default values, for example, to use an IP address of **10.1.1.2** for the BL4S200 board, and **10.1.1.1** for your PC, you can edit the values in the section that directly follows the "General Configuration" comment in the **TCP_CONFIG.LIB** library. You will find this library in the **LIB\TCPIP** directory.
3. You can create a **CUSTOM_CONFIG.LIB** library and use a **TCPCONFIG** value greater than 100. Instructions for doing this are at the beginning of the **TCP_CONFIG.LIB** library in the **LIB\TCPIP** directory.

There are some other "standard" configurations for **TCPCONFIG** that let you select different features such as DHCP. Their values are documented at the top of the **TCP_CONFIG.LIB** library in the **LIB\TCPIP** directory. More information is available in the ***Dynamic C TCP/IP User's Manual***.

5.2.2 How to Set Up your Computer for Direct Connect

Follow these instructions to set up your PC or notebook. Check with your administrator if you are unable to change the settings as described here since you may need administrator privileges. The instructions are specifically for Windows 2000, but the interface is similar for other versions of Windows.

> **TIP:** If you are using a PC that is already on a network, you will disconnect the PC from that network to run these sample programs. Write down the existing settings before changing them to facilitate restoring them when you are finished with the sample programs and reconnect your PC to the network.

1. Go to the control panel (**Start > Settings > Control Panel**), and then double-click the Network icon.
2. Select the network interface card used for the Ethernet interface you intend to use (e.g., **TCP/IP Xircom Credit Card Network Adapter**) and click on the “Properties” button. Depending on which version of Windows your PC is running, you may have to select the “Local Area Connection” first, and then click on the “Properties” button to bring up the Ethernet interface dialog. Then “Configure” your interface card for a “10Base-T Half-Duplex” or an “Auto-Negotiation” connection on the “Advanced” tab.

   > **NOTE:** Your network interface card will likely have a different name.

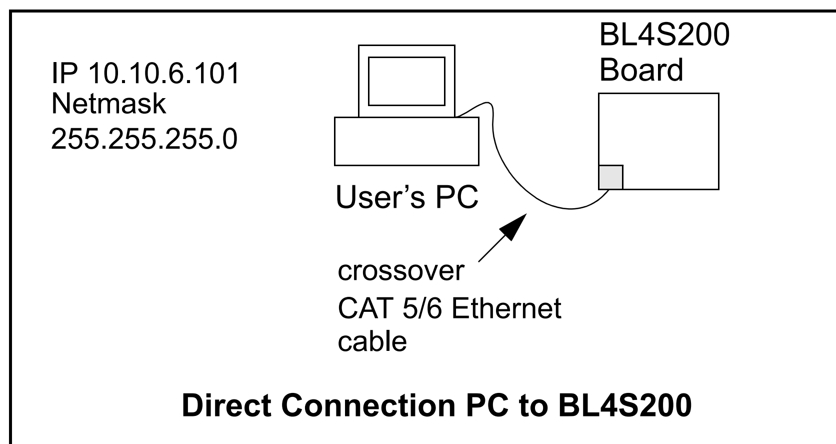
3. Now select the **IP Address** tab, and check **Specify an IP Address**, or select TCP/IP and click on “Properties” to assign an IP address to your computer (this will disable “obtain an IP address automatically”):

   IP Address : 10.10.6.101

   Netmask : 255.255.255.0

   Default gateway : 10.10.6.1

4. Click **<OK>** or **<Close>** to exit the various dialog boxes.



Direct Connection PC to BL4S200

5.2.3 Run the `PINGME.C` Demo

Connect the crossover cable from your computer's Ethernet port to the BL4S200's RJ-45 Ethernet connector. Open this sample program from the **`SAMPLES\TCPIP\ICMP`** folder, compile the program, and start it running under Dynamic C. When the program starts running, the green **LNK** light on the BL4S200 should be on to indicate an Ethernet connection is made. (Note: If the **LNK** light does not light, you may not have a crossover cable, or if you are using a hub perhaps the power is off on the hub.)

The next step is to ping the board from your PC. This can be done by bringing up the MS-DOS window and running the ping program:

```
ping 10.10.6.100
```

or by **Start** > **Run**

and typing the command

```
ping 10.10.6.100
```

Notice that the orange **ACT** light flashes on the BL4S200 while the ping is taking place, and indicates the transfer of data. The ping routine will ping the board four times and write a summary message on the screen describing the operation.

5.2.4 Running More Demo Programs With a Direct Connection

The program **`SSI.C`** (**`SAMPLES\BLxS2xx\TCPIP\`**) demonstrates how to make the BL4S200 a Web server. This program allows you to turn the LEDs on an attached Demonstration Board from the Tool Kit on and off from a remote Web browser. The LEDs on the Demonstration Board match the ones on the Web page. Follow the instructions included with the sample program. As long as you have not modified the **`TCPCONFIG 1`** macro in the sample program, enter the following server address in your Web browser to bring up the Web page served by the sample program.

http://10.10.6.100.

Otherwise use the TCP/IP settings you entered in the **`TCP_CONFIG.LIB`** library.

The sample program **`RWEB DIGITAL OUTPUTS.C`** (**`SAMPLES\BLxS2xx\TCPIP\`**) demonstrates using the **`digOut()`** function call to control configurable I/O sinking outputs to toggle LEDs on and off on the Demonstration Board from your Web browser.

The sample program **`TELNET.C`** (**`SAMPLES\BLxS2xx\TCPIP\`**) allows you to communicate with the BL4S200 using the Telnet protocol. This program takes anything that comes in on a port and sends it out Serial Port E (BL4S200) or Serial Port B (BL4S210). It uses a digital input to indicate that the TCP/IP connection should be closed and a digital output to toggle an LED to indicate that there is an active connection.

Follow the instructions included with the sample program. Run the Telnet program on your PC (**Start > Run `telnet 10.10.6.100`**). As long as you have not modified the **`TCPCONFIG 1`** macro in the sample program, the IP address is 10.10.6.100 as shown; otherwise use the TCP/IP settings you entered in the **`TCP_CONFIG.LIB`** library. Each character you type will be printed in Dynamic C's **STDIO** window, indicating that the board is receiving the characters typed via TCP/IP.

5.3 Where Do I Go From Here?

> **NOTE:** If you purchased your BL4S200 through a distributor or Rabbit partner, contact the distributor or partner first for technical support.

If there are any problems at this point:

- Use the Dynamic C **Help** menu to get further assistance with Dynamic C.
- Check the Rabbit Technical Bulletin Board and forums at www.digi.com/support/ and at www.rabbit.com/forums/.
- Use the Technical Support e-mail form at www.rabbit.com/support/questionSubmit.shtml.

If the sample programs ran fine, you are now ready to go on.

Additional sample programs are described in the ***Dynamic C TCP/IP User’s Manual***.

Refer to the ***Dynamic C TCP/IP User’s Manual*** to develop your own applications. ***An Introduction to TCP/IP*** provides background information on TCP/IP, and is available on the Web site.

6. Using the Wi-Fi Features

Chapter 6 discusses using the TCP/IP Wi-Fi features on the BL5S220 board. This networking feature is ***not*** available on other BL4S200 models, which have other network interfaces.

6.1 Introduction to Wi-Fi

Wi-Fi, a popular name for 802.11b/g, refers to the underlying technology for wireless local area networks (WLAN) based on the IEEE 802.11 suite of specifications conforming to standards defined by IEEE. IEEE 802.11b describes the media access and link layer control for a 2.4 GHz implementation, which can communicate at a top bit-rate of 11 Mbits/s. Other standards describe a faster implementation (54 Mbits/s) in the 2.4 GHz band (802.11g). The adoption of 802.11 has been fast because it's easy to use and the performance is comparable to wire-based LANs. Things look pretty much like a wireless LAN.

Wi-Fi (802.11b/g) is the most common and cost-effective implementation currently available. This is the implementation that is used with the BL5S220. A variety of Wi-Fi hardware exists, from wireless access points (WAPs), various Wi-Fi access devices with PCI, PCMCIA, CompactFlash, USB and SD/MMC interfaces, and Wi-Fi devices such as Web-based cameras and print servers.

802.11b/g can operate in one of two modes—in a managed-access mode (BSS), called an infrastructure mode, or an unmanaged mode (IBSS), called the ad-hoc mode. The 802.11 standard describes the details of how devices access each other in either of these modes.

6.1.1 Infrastructure Mode

The infrastructure mode requires an access point to manage devices that want to communicate with each other. An access point is identified with a channel and service set identifier (SSID) that it uses to communicate. Typically, an access point also acts as a gateway to a wired network, either an Ethernet or WAN (DSL/cable modem). Most access points can also act as a DHCP server, and provide IP, DNS, and gateway functions.

When a device wants to join an access point, it will typically scan each channel and look for a desired SSID for the access point. An empty-string SSID (" ") will associate the device with the first SSID that matches its capabilities.

Once the access point is discovered, the device will logically join the access point and announce itself. Once joined, the device can transmit and receive data packets much like an Ethernet-based MAC. Being in a joined state is akin to having link status in a 10/100Base-T network.

802.11b/g interface cards implement all of the 802.11b/g low-level configurations in firmware. In fact, the 802.11b/g default configuration is often sufficient for a device to join an access point automatically, which it can do once enabled. Commands issued to the chip set in the interface allow a host program to override the default configurations and execute functions implemented on the interface cards, for example, scanning for hosts and access points.

6.1.2 Ad-Hoc Mode

In the ad-hoc mode, each device can set a channel number and an SSID to communicate with. If devices are operating on the same channel and SSID, they can talk with each other, much like they would on a wired LAN such as an Ethernet. This works fine for a few devices that are statically configured to talk to each other, and no access point is needed.

6.1.3 Additional Information

802.11 Wireless Networking; published by O'Reilly Media, provides further information about 802.11b wireless networks.

6.2 Running Wi-Fi Sample Programs

In order to run the sample programs discussed in this chapter and elsewhere in this manual,

1. Your module must be installed on the BL5S220 motherboard.
2. Dynamic C must be installed and running on your PC.
3. The programming cable must connect the programming header on the module to your PC.
4. Power must be applied to the BL5S220.

Refer to Chapter 2, “Getting Started,” if you need further information on these steps.

To run a sample program, open it with the **File** menu, then compile and run it by pressing **F9**.

Each sample program has comments that describe the purpose and function of the program. Follow the instructions at the beginning of the sample program.

Complete information on Dynamic C is provided in the ***Dynamic C User’s Manual***.

6.2.1 Wi-Fi Setup

Figure 26 shows how your development setup might look once you're ready to proceed.

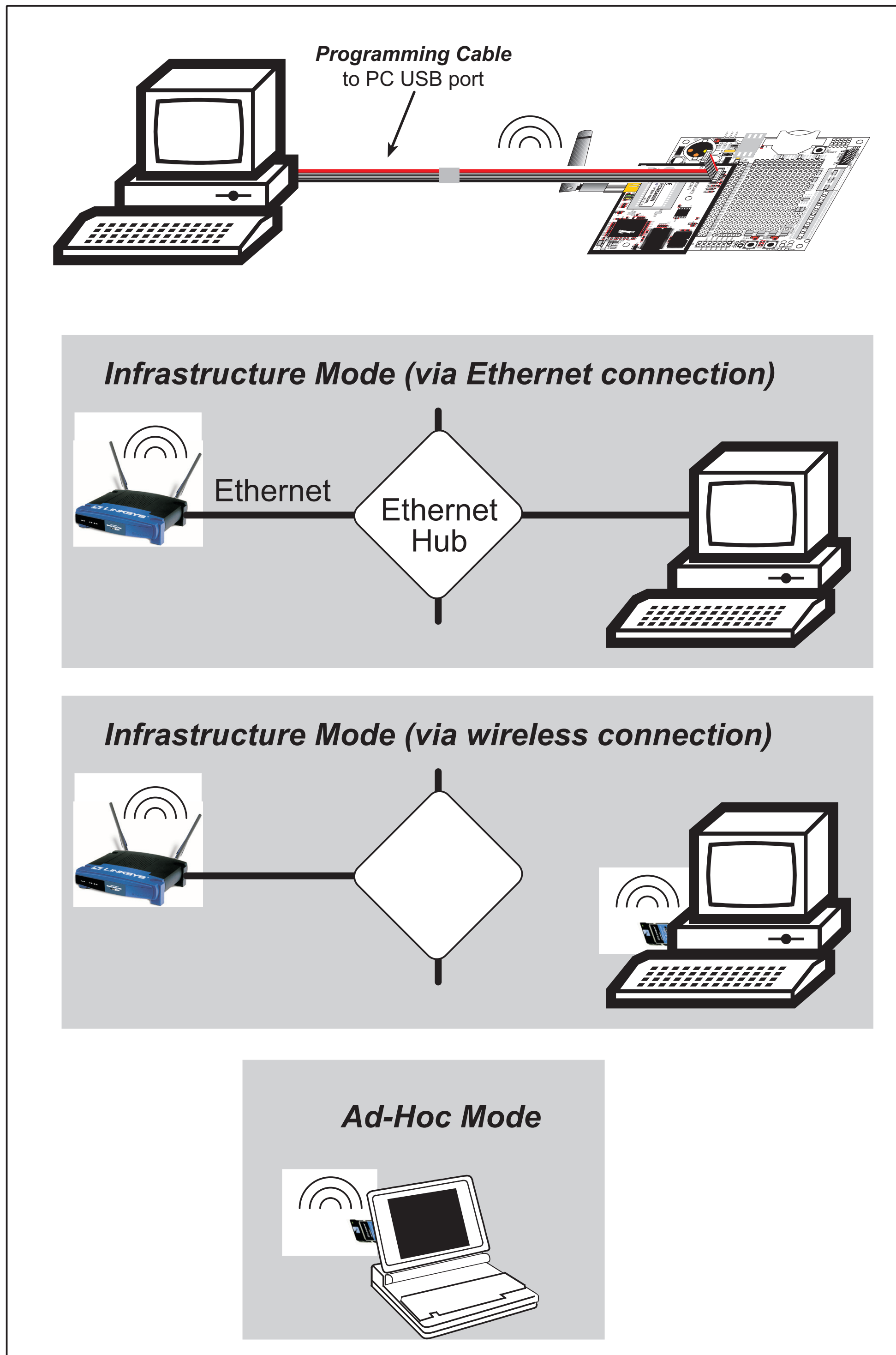


Figure 26. Wi-Fi Host Setup

6.2.2 What Else You Will Need

Besides what is supplied with the BL4S200 Tool Kit, you will need a PC with an available USB port to program the BL5S220. You will need either an access point for an existing Wi-Fi network that you are allowed to access and have a PC or notebook connected to that network (infrastructure mode), or you will need at least a PDA or PC with Wi-Fi to use the ad-hoc mode.

6.2.3 Configuration Information

6.2.3.1 Network/Wi-Fi Configuration

Any device placed on an Ethernet-based Internet Protocol (IP) network must have its own IP address. IP addresses are 32-bit numbers that uniquely identify a device. Besides the IP address, we also need a netmask, which is a 32-bit number that tells the TCP/IP stack what part of the IP address identifies the local network the device lives on.

The sample programs configure the BL5S220 with a default **TCPCONFIG** macro from the **LIB\Rabbit4000\TCPIP\TCP_CONFIG.LIB** library. This macro allows specific IP address, netmask, gateway, and Wi-Fi parameters to be set at compile time. Change the network settings to configure your BL5S220 with your own Ethernet settings only if that is necessary to run the sample programs; you will likely need to change some of the Wi-Fi settings.

- Network Parameters

  These lines contain the IP address, netmask, nameserver, and gateway parameters.

```
#define _PRIMARY_STATIC_IP "10.10.6.100"
#define _PRIMARY_NETMASK   "255.255.255.0"
#define MY_NAMESERVER      "10.10.6.1"
#define MY_GATEWAY         "10.10.6.1"
```

There are similar macros defined for the various Wi-Fi settings as explained in Section 6.3.1.

The Wi-Fi configurations are contained within **TCPCONFIG 1** (no DHCP) and **TCPCONFIG 5** (with DHCP, used primarily with infrastructure mode). You will need to **#define TCPCONFIG 1** or **#define TCPCONFIG 5** at the beginning of your program.

> **NOTE: TCPCONFIG 0** is not supported for Wi-Fi applications.

There are some other "standard" configurations for **TCPCONFIG**. Their values are documented in the **LIB\Rabbit4000\TCPIP\TCP_CONFIG.LIB** library. More information is available in the ***Dynamic C TCP/IP User's Manual***.

6.2.3.2 PC/Laptop/PDA Configuration

This section shows how to configure your PC or notebook to run the sample programs. Here we're mainly interested in the PC or notebook that will be communicating wirelessly, which is not necessarily the PC that is being used to compile and run the sample program on the BL5S220.

This following instructions provide configuration information for the three possible Wi-Fi setups shown in Figure 26. Start by going to the control panel (**Start > Settings > Control Panel**) and click on **Network Connections**. Check with your administrator if you are unable to change the settings as described here since you may need administrator privileges.

When you are using an access point with your setup in the infrastructure mode, you will also have to set the IP address and netmask (e.g., 10.10.6.99 and 255.255.255.0) for the access point. Check the documentation for the access point for information on how to do this.

Infrastructure Mode (via Ethernet connection)

1. Go to the **Local Area Connection** to select the network interface card used you intend to use (e.g., **TCP/IP Xircom Credit Card Network Adapter**) and click on the "Properties" button. Depending on which version of Windows your PC is running, you may have to select the "Local Area Connection" first, and then click on the "Properties" button to bring up the Ethernet interface dialog. Then "configure" your interface card for an "Auto-Negotiation" or "10Base-T Half-Duplex" connection on the "Advanced" tab.

   NOTE: Your network interface card will likely have a different name.

2. Now select the **IP Address** tab, and check **Specify an IP Address**, or select TCP/IP and click on "Properties" to fill in the following fields:

   IP Address : 10.10.6.101

   Netmask : 255.255.255.0

   Default gateway : 10.10.6.1

   TIP: If you are using a PC that is already on a network, you will disconnect the PC from that network to run these sample programs. Write down the existing settings before changing them so that you can restore them easily when you are finished with the sample programs.

   The IP address and netmask need to be set regardless of whether you will be using the ad-hoc mode or the infrastructure mode.

3. Click **<OK>** or **<Close>** to exit the various dialog boxes.

Infrastructure Mode (via wireless connection)

Set the IP address and netmask for your wireless-enabled PC or notebook as described in Step 2 for **Infrastructure Mode (via Ethernet connection)** by clicking on **Network Connections**, then on **Local Area Connection**. Now click on **Wireless Network Connection** to select the wireless network you will be connecting to. Once a sample program is running, you will be able to select the network from a list of available networks. You will have to set your wireless network name with the access point SSID macro for the infrastructure mode as explained in Section 6.3, “Dynamic C Wi-Fi Configurations.”

Ad-Hoc Mode

Set the IP address and netmask for your wireless-enabled PC or notebook as described in Step 2 for **Infrastructure Mode (via Ethernet connection)** by clicking on **Network Connections**, then on **Local Area Connection**. Now click on **Wireless Network Connection** to select the wireless network you will be connecting to. Once a sample program is running, you will be able to select the network from a list of available networks. You will have set your wireless network name with the Wi-Fi channel macros for the ad-hoc mode as explained in Section 6.3, “Dynamic C Wi-Fi Configurations.”

Once the PC or notebook is set up, we're ready to communicate. You can use Telnet or a Web browser such as Internet Explorer, which come with most Windows installations, to use the network interface, and you can use HyperTerminal to view the serial port when these are called for in some of the later sample programs.

Now we're ready to run the sample programs in the Dynamic C **`Samples\TCPIP\WiFi`** folder. The sample programs should run as is in most cases.

6.2.4 Wi-Fi Sample Programs

The sample programs in Section 6.2.4.1 show how to set up the country- or region-specific attributes, but do not show the basic setup of a wireless network. The sample programs in Section 6.2.4.2 show the setup and operation of a wireless network — the **`WIFISCAN.C`** sample program is ideal to demonstrate that the BL5S220 has been hooked up correctly and that the Wi-Fi setup is correct so that an access point can be found.

6.2.4.1 Wi-Fi Operating Region Configuration

The country or region you select will automatically set the power and channel requirements to operate the BL5S220. The following three options are available.

1. *Country or region is set at compile time.* This option is ideal when the end device is intended to be sold and used only in a single region. If the end device is to be deployed across multiple regions, this method would require an application image to be created for each region. This option is the only approved option for the BL5S220 in Japan.
2. *Country or region is set via the 802.11d feature of the access point.* This option uses beacons from an access point to configure the BL5S220 country or region automatically. The end user is responsible for enabling 802.11d on the access point and then selecting the correct country to be broadcast in the beacon packets.

   NOTE: This option sets the power limit for BL5S220 to the maximum level permitted in the region or the capability of the BL5S220, whichever is less. Since the beacons are being sent continuously, the **`ifconfig IFS_WIFI_TX_POWER`** function cannot be used with this option.

3. *Country or region is set at run time.* This is a convenient option when the end devices will be deployed in multiple regions. A serial user interface would allow the BL5S220 to be configured via a Web page. Systems integrators would still have to make sure the end devices operate within the regulatory requirements of the country or region where the units are being deployed.

These options may be used alone or in any combination. The three sample programs in the Dynamic C **`Samples\WiFi\Regulatory`** folder illustrate the use of these three options.

- **`REGION_COMPILETIME.C`**—demonstrates how you can set up your BL5S220-based system at compile time to operate in a given country or region to meet power and channel requirements.

  The country or region you select will automatically set the power and channel requirements to operate the BL5S220. Rabbit recommends that you check the regulations for the country where your system incorporating the BL5S220 will be deployed for any

other requirements. *Any attempt to operate a device outside the allowed channel range or power limits will void your regulatory approval to operate the device in that country.*

Before you compile and run this sample program, uncomment the **`#define IFC_ WIFI_REGION`** line corresponding to the region where your system will be deployed. The Americas region will be used by default if one of these lines is not uncommented. Now compile and run this sample program. The Dynamic C **STDIO** window will display the region you selected.

The sample program also allows you to set up the TCP/IP configuration, and set the IP address and SSID as shown in the sample code below.

```
#define TCPCONFIG 1
#define _PRIMARY_STATIC_IP "10.10.6.170"
#define IFC_WIFI_SSID "rabbitTest"
```

- **`REGION_MULTI_DOMAIN.C`**—demonstrates how the multi-domain options from the access point can be used to configure your BL5S220-based system to meet regional regulations. The sample program includes pings to indicate that the BL5S220-based system has successfully received country information from your access point.

  The country or region you select will automatically set the power and channel requirements to operate the BL5S220. Rabbit recommends that you check the regulations for the country where your system incorporating the BL5S220 will be deployed for any other requirements.

  Before you compile and run this sample program, verify that the access point has the 802.11d option enabled and is set for the correct region or country. Check the TCP/IP configuration parameters, the IP address, and the SSID in the macros, which are reproduced below.

  ```
  #define TCPCONFIG 1
  #define WIFI_REGION_VERBOSE
  #define _PRIMARY_STATIC_IP "10.10.6.170"
  #define IFC_WIFI_SSID "rabbitTest"
  ```

  Now compile and run this sample program. The **`#define WIFI_REGION_VERBOSE`** macro will display the channel and power limit settings. The Dynamic C **STDIO** window will then display a menu that allows you to complete the configuration of the user interface.

- **`REGION_RUNTIME_PING.C`**—demonstrates how the region or country can be set at run time to configure your BL5S220-based system to meet regional regulations. The sample program also shows how to save and retrieve the region setting from nonvolatile memory. Once the region/country is set, this sample program sends pings using the limits you set.

  The country or region you select will automatically set the power and channel requirements to operate the BL5S220. Digi International recommends that you check the regulations for the country where your system incorporating the BL5S220 will be deployed for any other requirements.

Before you compile and run this sample program, check the TCP/IP configuration parameters, the IP address, and the SSID in the macros, which are reproduced below.

```
#define TCPCONFIG 1
// #define WIFI_REGION_VERBOSE
#define PING_WHO "10.10.6.1"
#define _PRIMARY_STATIC_IP "10.10.6.170"
#define IFC_WIFI_SSID "rabbitTest"
```

Now compile and run this sample program. Uncomment the **#define WIFI_REGION_VERBOSE** macro to display the channel and power limit settings. The Dynamic C **STDIO** window will then display a menu that allows you to complete the configuration of the user interface.

6.2.4.2 Wi-Fi Operation

- **WIFIDHCPORSTATIC.C**—demonstrates the runtime selection of a static IP configuration or DHCP. The **SAMPLES\TCPIP\DHCP.C** sample program provides further examples of using DHCP with your application.

  Before you compile and run this sample program, check the TCP/IP configuration parameters, the IP address, and the SSID in the macros, which are reproduced below.

  ```
  #define USE_DHCP
  #define TCPCONFIG 1
  #define _PRIMARY_STATIC_IP "10.10.6.100"
  #define IFC_WIFI_SSID "rabbitTest"
  ```

  Modify the values to match your network. You may also need to modify the values for **MY_GATEWAY** if you are not pinging from the local subnet.

  Now press **F9** to compile and run the sample program. When prompted in the Dynamic C **STDIO** window, type 's' for static configuration or 'd' for DHCP.

- **WIFIMULTIPLEAPS.C**—demonstrates changing access points using WEP keys. You will need two access points to run this sample program. The access points should be isolated or on separate networks.

  The sample program associates the RabbitCore module with the first access point (**AP_0** defined below) with WEP key **KEY0** (defined below). After associating, the sample program waits for a predefined time period, and then pings the Ethernet address of the access point (**AP_ADDRESS_0**). The sample program then associates with the second access point and pings its Ethernet address (**AP_1**, **KEY1**, **AP_ADDRESS_1**), and then switches back and forth between the two access points.

  When changing access points, first bring the **IF_WIFI0** interface down by calling **ifdown(IF_WIFI0)**. Next, change the SSID and key(s) using **ifconfig()** calls. Finally, bring the **IF_WIFI0** interface back up by calling **ifup(IF_WIFI0)**. Note that the sample program checks for status while waiting for the interface to come up or down.

  Before you compile and run this sample program, check the TCP/IP configuration parameters, the IP address, and the SSID in the macros, which are reproduced below.

  ```
  #define TCPCONFIG 1
  #define IFC_WIFI_ENCRYPTION IFPARAM_WIFI_ENCR_WEP
  ```

You do not need to configure the SSID of your network since that is done from the access point names.

Now configure the access to the two access points.

```
// First Access Point
#define AP_0 "test1"
#define AP_0_LEN strlen(AP_0)
#define MY_ADDRESS_0 "10.10.6.250" // use this static IP when connected to AP 0
#define PING_ADDRESS_0 "10.10.6.1" // address on AP 0 to ping
#define KEY_0 "0123456789abcdef0123456789"

// Second Access Point
#define AP_1 "test2"
#define AP_1_LEN strlen(AP_1)
#define MY_ADDRESS_1 "10.10.0.99" // use this static IP when connected to AP 1
#define PING_ADDRESS_1 "10.10.0.50"// address on AP 1 to ping
#define KEY_1 "0123456789abcdef0123456789"

#define IFC_WIFI_SSID AP_0
#define _PRIMARY_STATIC_IP MY_ADDRESS_0
```

Modify the access point names and keys to match your access points and network.

- **WIFIPINGYOU.C**—sends out a series of pings to a RabbitCore module on an ad-hoc Wi-Fi network.

  This sample program uses some predefined macros. The first macro specifies the default TCP/IP configuration from the Dynamic C **LIB\Rabbit4000\TCPIP\TCP_CONFIG.LIB** library.

  ```
  #define TCPCONFIG 1
  ```

  Use the next macro unchanged as long as you have only one BL5S220. Otherwise use this macro unchanged for the first BL5S220.

  ```
  #define NODE 1
  ```

  Then change the macro to **#define NODE 2** before you compile and run this sample program on the second BL5S220.

  The next macros assign an SSID name and a channel number to the Wi-Fi network.

  ```
  #define IFC_WIFI_SSID  "rab-hoc"
  #define IFC_WIFI_OWNCHANNEL "5"
  ```

  Finally, IP addresses are assigned to the RabbitCore modules.

  ```
  #define IPADDR_1  "10.10.8.1"
  #define IPADDR_2  "10.10.8.2"
  ```

  As long as you have only one BL5S220, the Dynamic C **STDIO** window will display the pings sent out by the module. You may set up a Wi-Fi enabled laptop with the IP address in **IPADDR_2** to get the pings.

  If you have two BL5S220 boards, they will ping each other, and the Dynamic C **STDIO** window will display the pings.

- **`WIFISCAN.C`**—initializes the BL5S220 and scans for other Wi-Fi devices that are operating in either the ad-hoc mode or through access points in the infrastructure mode. No network parameter settings are needed since the BL5S220 does not actually join an 802.11 network. This program outputs the results of the scan to the Dynamic C **STDIO** window.
- **`WIFISCANASSOCIATE.C`**— demostrates how to scan Wi-Fi channels for SSIDs using **`ifconfig IFS_WIFI_SCAN`**. This takes a while to complete, so **`ifconfig()`** calls a callback function when it is done. The callback function is specified using **`ifconfig IFS_WIFI_SCAN`**.

  Before you run this sample program, configure the Dynamic C **`TCP_CONFIG.LIB`** library and your **`TCPCONFIG`** macro.

  1. Use macro definitions in the “Defines” tab in the Dynamic C **Options > Project Options** menu to modify any parameter settings.

     If you are not using DHCP, set the IP parameters to values appropriate to your network.

     ```
     _PRIMARY_STATIC_IP = "10.10.6.100"
     _PRIMARY_NETMASK = "255.255.255.0"
     MY_NAMESERVER = "10.10.6.1"
     MY_GATEWAY = "10.10.6.1"
     ```

     Set **`IFS_WIFI_SSID`** to an appropriate value. To connect to a specific BSS, set **`IFS_WIFI_SSID`** to the SSID of your access point as a C-style string, for example,

     ```
     IFS_WIFI_SSID = "My Access Point"
     ```

     or use an empty string, **`""`**, to associate with the strongest BSS available.

     Alternatively, you may create your own **`CUSTOM_CONFIG.LIB`** library modeled on the Dynamic C **`TCP_CONFIG.LIB`** library. Then use a **`TCPCONFIG`** macro greater than or equal to 100, which will invoke your **`CUSTOM_CONFIG.LIB`** library to be used. Remember to add the **`CUSTOM_CONFIG.LIB`** library to **`LIB.DIR`**.
  2. If you are using DHCP, change the definition of the **`TCPCONFIG`** macro to 5. The default value of 1 indicates Wi-Fi with a static IP address.

  Now compile and run the sample program. Follow the menu options displayed in the Dynamic C **STDIO** window.

  ```
  s - scan for BSS's,
  a - scan and associate
  m - dump MAC state information
  t - dump tx information
  ```

  Note that **`ifconfig IFS_WIFI_SCAN`** function calls do not return data directly since the scan takes a fair amount of time. Instead, callback functions are used. The callback function is passed to **`ifconfig()`** as the only parameter to **`IFS_WIFI_SCAN`**.

  ```
  ifconfig(IF_WIFI0, IFS_WIFI_SCAN, scan_callback, IFS_END);
  ```

The data passed to the callback function are ephemeral since another scan may occur. Thus, the data need to be used (or copied) during the callback function.

While waiting for user input, it is important to keep the network alive by calling **tcp_tick(NULL)** regularly.

6.2.5 RCM5400W Sample Programs

The following sample programs are in the Dynamic C **SAMPLES\RCM5400W\TCPIP** folder.

- **PASSPHRASE.C**—This program demonstrates how to perform the CPU-intensive process of converting an ASCII passphrase into a WPA PSK hex key.

  For security reasons, the mapping function is deliberately designed to be CPU-intensive in order to make a dictionary-based attack more difficult. It can take on the order of 40 seconds to perform the 4096 iterations on the BL5S220. Since this may be an unacceptable amount of time to “block” the application program, a method is provided to split up the computation.

  As you compile and run this sample program, there is no network activity. You will only notice that some information is printed out in the Dynamic C **STDIO** window.

- **PINGLED_STATS.C**—This program is similar to **PINGLED.C**, but it also displays receiver/transmitter statistics in the Dynamic C **STDIO** window.

  Before you compile and run this sample program, change **PING_WHO** to the host you want to ping. You may modify **PING_DELAY** define to change the amount of time in milliseconds between the outgoing pings.

  Modify the value in the **MOVING_AVERAGE** macro to change the moving average filtering of the statistics. Also review the **GATHER_INTERVAL** and **GRAPHICAL** macros, which affect the number of samples to gather and create a bar graph display instead of a numeric display.

  Uncomment the **VERBOSE** define to see the incoming ping replies.

- **PINGLED_WPA_PSK.C**—This program demonstrates the use of WPA PSK (Wi-Fi Protected Access with Pre-Shared Key). WPA is a more secure replacement for WEP. The implementation in the sample program supports use of the TKIP (Temporal Key Integrity Protocol) cypher suite.

  The sample program uses macros to configure the access point for WPA PSK, specify the TKIP cypher suite, assign the access point SSID, and set the passphrase.

```
#define WIFI_USE_WPA  // Bring in WPA support
#define IFC_WIFI_ENCRYPTION IFPARAM_WIFI_ENCR_TKIP // Define cypher suite

#define IFC_WIFI_SSID "parvati"

#define IFC_WIFI_WPA_PSK_PASSPHRASE "now is the time"
```

The next macro specifies a suitable pre-shared key to use instead of the passphrase. The key may be entered either as 64 hexadecimal digits or as an ASCII string of up to 63 characters.

```
#define IFC_WIFI_WPA_PSK_HEXSTR
```

TIP: There is a good chance of typos since the key is long. First, enter the key in this sample program macro, then copy and paste it to your access point. This ensures that both the BL5S220 and the access point have the same key.

TIP: For an initial test, it may be easier to use the 64 hex digit form of the key rather than the ASCII passphrase. A passphrase requires considerable computation effort, which delays the startup of the sample program by about 30 seconds.

Change **`PING_WHO`** to the host you want to ping. You may modify **`PING_DELAY`** to change the amount of time in milliseconds between the outgoing pings.

Uncomment the **`VERBOSE`** define to see the incoming ping replies.

Once you have compiled the sample program and it is running, LED DS2 will flash when a ping is sent, and LED DS3 will flash when a ping is received.

- **PINGLED_WPA2_CCMP.C**—This sample program is an extension of **PINGLED.C**. It demonstrates the use of WPA2 PSK (Wi-Fi Protected Access with Pre-Shared Key).). WPA is a more secure replacement for WEP. The implementation in the sample program uses the Advanced Encryption Standard (AES) based algorithm, also known as the CCMP (Counter Mode with Cipher Block Chaining Message Authentication Code Protocol) cypher suite.

  Apart from the configuration of **WPA2_CCMP** at the top of the sample program, the rest of the code is identical to the case without WPA2 PSK. Indeed, most of the TCP/IP sample programs should work with WPA2 CCMP simply by using the same configuration settings.

  Configure your access point for WPA2 PSK before you run this sample program. Specify the CCMP cypher suite, and enter a suitable pre-shared key. The key may be entered either as 64 hexadecimal digits or as an ASCII string of up to 63 characters.

  TIP: There is a good chance of typos since the key is long. First, enter the key in this sample program macro, then copy and paste it to your access point. This ensures that both the BL5S220 and the access point have the same key.

  TIP: For an initial test, it may be easier to use the 64 hex digit form of the key rather than the ASCII passphrase. A passphrase requires considerable computation effort, which delays the startup of the sample program by about 30 seconds.

  Now change **PING_WHO** to the address of the host you want to ping.

  You may modify the **PING_DELAY** define to change the amount of time (in milliseconds) between the outgoing pings.

  Uncomment the VERBOSE define to see the incoming ping replies.

  Finally, compile and run this sample program. LED DS2 will flash when a ping is sent. LED DS3 will flash when a ping is received.

6.3 Dynamic C Wi-Fi Configurations

Rabbit has implemented a packet driver for the BL5S220 that functions much like an Ethernet driver for the Dynamic C implementation of the TCP/IP protocol stack. In addition to functioning like an Ethernet packet driver, this driver implements a function call to access the functions implemented on the 802.11b/g interface, and to mask channels that are not available in the region where the BL5S220 will be used.

The Wi-Fi interface may be used either at compile time using macro statements or at run time with the **ifconfig()** function call from the Dynamic C **LIB\Rabbit4000\TCPIP\NET.LIB** library.

6.3.1 Configuring TCP/IP at Compile Time

Digi International has made it easy for you to set up the parameter configuration using already-defined **TCPCONFIG** macros from the Dynamic C **LIB\Rabbit4000\TCPIP\TCP_CONFIG.LIB** library at the beginning of your program as in the example below.

```
#define TCPCONFIG 1
```

There are two **TCPCONFIG** macros specifically set up for Wi-Fi applications with the BL5S220. (**TCPCONFIG 0** is not supported for Wi-Fi applications.)

TCPCONFIG 1	No DHCP
TCPCONFIG 5	DHCP enabled

These default IP address, netmask, nameserver, and gateway network parameters are set up for the **TCPCONFIG** macros.

```
#define _PRIMARY_STATIC_IP "10.10.6.100"
#define _PRIMARY_NETMASK   "255.255.255.0"
#define MY_NAMESERVER      "10.10.6.1"
#define MY_GATEWAY         "10.10.6.1"
```

The use of quotation marks in the examples described in this chapter is important since the absence of quotation marks will be flagged with warning messages when encrypted libraries are used.

Wi-Fi Parameters

- Access Point SSID—**IFC_WIFI_SSID**. This is the only mandatory parameter. Define the **IFC_WIFI_SSID** macro to a string for the SSID of the access point in the infrastructure (BSS) mode, or the SSID of the ad-hoc network in the ad-hoc (IBSS) mode.

  The default is shown below.

  ```
  #define IFC_WIFI_SSID "rabbitTest"
  ```

- Mode—**IFC_WIFI_MODE** determines the mode: **IFPARAM_WIFI_INFRASTRUCT** for the infrastructure mode, or **IFPARAM_WIFI_ADHOC** for the ad-hoc mode.

  The default is shown below.

  ```
  #define IFC_WIFI_MODE IFPARAM_WIFI_INFRASTRUCT
  ```

- Your Own Channel—**IFC_WIFI_CHANNEL** determines the channel on which to operate. Define it to a *string,* not an integer.

  The default is shown below.

  ```
  #define IFC_WIFI_CHANNEL 0
  ```

  The default **0** means that any valid channel may be used by the requested SSID. This parameter is mandatory when creating an ad-hoc network. While it is optional for the infrastructure mode, it is usually best left at the default **0**.

  Note that there are restrictions on which channels may be used in certain countries. These are provided in the ***RabbitCore RCM5400 User's Manual*** for some countries.

- Region/Country—**IFC_WIFI_REGION** sets the channel range and maximum power limit to match the region selected. The ***RabbitCore RCM5400 User's Manual*** lists the regions that are supported and their corresponding macros.

  The region selected *must* match the region where the BL5S220 will be used.

  The default is shown below.

  ```
  #define IFC_WIFI_REGION IFPARAM_WIFI_REGION_AMERICAS
  ```

- Disable/enable encryption—**IFC_WIFI_ENCRYPTION** indicates whether or not encryption is enabled.

  The default (encryption disabled) is shown below.

  ```
  #define IFC_WIFI_ENCRYPTION IFPARAM_WIFI_ENCR_NONE
  ```

  The following encryption options are available.

  - **IFPARAM_WIFI_ENCR_NONE** — no encryption is used.
  - **IFPARAM_WIFI_ENCR_ANY** — any type of encryption is used.
  - **IFPARAM_WIFI_ENCR_WEP** — use WEP encryption. You will need to define at least one WEP key (see below).
  - **IFPARAM_WIFI_ENCR_TKIP** — use TKIP or WPA encryption. You will need to define a passphrase or a key for TKIP encryption, as well as define the **WIFI_USE_WPA** macro (see below).
  - **IFPARAM_WIFI_ENCR_CCMP** — use CCMP or WPA2 encryption. You will need to define at least one WEP key (see below).

- There are four encryption keys (**0**, **1**, **2**, **3**) associated with the **IFC_WIFI_WEP_KEYNUM** macro (default **0**). One or more of the following additional macros must be defined as well. The default is for the keys to remain undefined.

  ```
  IFC_WIFI_WEP_KEY0_BIN        IFC_WIFI_WEP_KEY0_HEXSTR
  IFC_WIFI_WEP_KEY1_BIN        IFC_WIFI_WEP_KEY1_HEXSTR
  IFC_WIFI_WEP_KEY2_BIN        IFC_WIFI_WEP_KEY2_HEXSTR
  IFC_WIFI_WEP_KEY3_BIN        IFC_WIFI_WEP_KEY3_HEXSTR
  ```

These macros specify the WEP keys to use for WEP encryption. These keys can be either 40-bit or 104-bit (i.e., 5 bytes or 13 bytes). They must be defined as a comma-separated list of byte values.

Note that you do not necessarily need to define all four WEP keys. You may typically just define one key, but make sure it matches the key used on all other devices, and set **IFC_WIFI_WEP_KEYNUM** to point to the correct key.

If both **IFC_WIFI_WEP_KEY#_BIN** and **IFC_WIFI_WEP_KEY#_HEXSTR** are defined for a particular key, the hex version will be used.

- Use WPA encryption.

  The following macro must also be used to compile WPA functionality into the Wi-Fi driver. This is necessary to enable TKIP encryption.

```
#define WIFI_USE_WPA
```

- Set WPA passphrase—**IFC_WIFI_WPA_PSK_PASSPHRASE** is a string that matches the passphrase on your access point. It may also point to a variable.

  Define an ASCII passphrase here, from 1 to 63 characters long. An example is shown below.

```
#define IFC_WIFI_WPA_PSK_PASSPHRASE "now is the time"
```

  If possible, you should use **IFC_WIFI_WPA_PSK_HEXSTR** instead of **IFC_WIFI_WPA_PSK_PASSPHRASE** to set the key.

- Set WPA hexadecimal key—**IFC_WIFI_WPA_PSK_HEXSTR** is a string of hexadecimal digits that matches the 256-bit (64-byte) hexadecimal key used by your access point.

  Specify a 64 hexadecimal digit (256 bits) key here. This key will be used and will override any passphrase set with the **IFC_WIFI_WPA_PSK_PASSPHRASE** macro. The example hex key shown below

```
#define IFC_WIFI_WPA_PSK_HEXSTR \
   "57A12204B7B350C4A86A507A8AF23C0E81D0319F4C4C4AE83CE3299EFE1FCD27"
```

  is valid for the SSID **"rabbitTest"** and the passphrase **"now is the time"**.

  Using a passphrase is rather slow. It takes a Rabbit 5000 more than 20 seconds to generate the actual 256-bit key from the passphrase. If you use a passphrase and **#define WIFI_VERBOSE_PASSPHRASE**, the Wi-Fi library will helpfully print out the hex key corresponding to that passphrase and SSID.

- Authentication algorithm—**IFC_WIFI_AUTHENTICATION** can be used to specify the authentication modes used.

  The default shown below allows enables both open-system authentication and shared-key authentication.

```
#define IFPARAM_WIFI_AUTH_ANY
```

The following authentication options are available.

- **`IFPARAM_WIFI_AUTH_OPEN`** — only use open authentication.
- **`IFPARAM_WIFI_AUTH_SHAREDKEY`** — only use shared-key authentication (useful for WEP only).
- **`IFPARAM_WIFI_WPA_PSK`** — use WPA preshared-key authentication (useful for TKIP and CCMP only).

- Fragmentation threshold—**`IFC_WIFI_FRAG_THRESHOLD`** sets the fragmentation threshold. Frames (or packets) that are larger than this threshold are split into multiple fragments. This can be useful on busy or noisy networks. The value can be between **`256`** and **`2346`**.

  The default, **`0`**, means no fragmentation.

```
#define IFC_WIFI_FRAG_THRESHOLD 0
```

- RTS threshold—**`IFC_WIFI_RTS_THRESHOLD`** sets the RTS threshold, the frame size at which the RTS/CTS mechanism is used. This is sometimes useful on busy or noisy networks. Its range is **`0`** to **`2347`**.

  The default, **`2347`**, means no RTS/CTS.

```
#define IFC_WIFI_RTS_THRESHOLD 2347
```

Examples are available within Dynamic C. Select "Function Lookup" from the **Help** menu, or press **<ctrl-H>**. Type "TCPCONFIG" in the Function Search field, and hit **<Enter>**. Scroll down to the section on "Wi-Fi Configuration." The ***Dynamic C TCP/IP User's Manual***.(Volume 1) provides additional information about these macros and Wi-Fi.

It is also possible to redefine any of the above parameters dynamically using the **`ifconfig()`** function call. Macros for alternative Wi-Fi configurations are provided with the **`ifconfig()`** function call, and may be used to change the above default macros or configurations.

6.3.2 Configuring TCP/IP at Run Time

There is one basic function call used to configure Wi-Fi and other network settings — **`ifconfig()`**. See the ***Dynamic C TCP/IP User's Manual, Volume 1*** for more information about this function call.

6.3.3 Other Key Function Calls

Remember to call **`sock_init()`** after all the Wi-Fi parameters have been defined. The Wi-Fi interface will be up automatically as long as you configured Dynamic C at compile time with one of the **`TCPCONFIG`** macros. Otherwise the Wi-Fi interface is neither up nor down, and must be brought up explicitly by calling either **`ifup(IF_WIFI0)`** or **`ifconfig(IF_WIFI0,…)`**. You must bring the interface down when you configure Dynamic C at run time before modifying any parameters that require the interface to be down (see Section 6.3.2) by calling **`ifdown(IF_WIFI0)`**. Then bring the interface back up.

Finally, no radio transmission occurs until you call **`tcp_tick(NULL)`**.

Instead of executing the above sequence based on **`sock_init()`**, you could use **`sock_init_or_exit(1)`** as a debugging tool to transmit packets (ARP, DHCP, association, and authentication) while bringing up the interface and to get the IP address.

6.4 Where Do I Go From Here?

NOTE: If you purchased your BL5S220 through a distributor or through a Rabbit partner, contact the distributor or partner first for technical support.

If there are any problems at this point:

- Use the Dynamic C **Help** menu to get further assistance with Dynamic C.
- Check the Rabbit Technical Bulletin Board and forums at www.rabbit.com/support/bb/ and at www.rabbit,com/forums/.
- Use the Technical Support e-mail form at www.rabbit.com/support/.

If the sample programs ran fine, you are now ready to go on.

An Introduction to TCP/IP, ***An Introduction to Wi-Fi***, and the ***Dynamic C TCP/IP User's Manual***.provide background and reference information on TCP/IP, and are available on the CD and on our Web site.

7. Using the ZigBee Features

Chapter 7 discusses using the ZigBee features on the BL4S230 board. This networking feature is ***not*** available on other BL4S200 models, which have other network interfaces.

7.1 Introduction to the ZigBee Protocol

The ZigBee PRO specification was ratified in April, 2007, and covers high-level communication protocols for small, low-power digital modems based on the IEEE 802.15.4 standard for wireless personal area networks (WPANs). The XBee RF module used by the BL4S230 operates in the 2.4 GHz industrial, scientific, and medical (ISM) radio band in most jurisdictions worldwide.

The ZigBee protocol is ideal for embedded-system applications that are characterized by low data rates and low power consumption. A network of devices using the ZigBee protocol works via a self-organizing mesh network that can be used for industrial control, embedded sensors, data collection, smoke and intruder warning, and building automation. The power consumption of the individual device could be met for a year or longer using the originally installed battery.

A ZigBee device can be set up in one of three ways.

- As a *coordinator*: The coordinator serves as the root of the network tree. Each network can only have one coordinator. The coordinator stores information about the network and provides the repository for security keys.
- As a *router*. Routers pass data from other devices.
- As an *end device*. End devices contain just enough functionality to talk to their parent node (either the coordinator or a router), and cannot relay data from other devices.

The XBee RF module used by the BL4S230 presently supports using the BL4S230 in a mesh network. RCM4510W modules on the BL4S230 are preconfigured with ZB router firmware; coordinator firmware is included in the Dynamic C installation along with a sample program to allow you to download the coordinator firmware.

The firmware used with the XBee RF modules on the BL4S230 is based on the API command set.

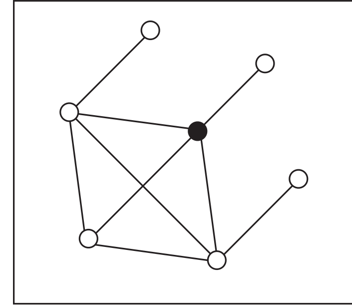


Figure 27. Mesh Network

An Introduction to ZigBee provides background information on the ZigBee protocol, and is available on the CD and on our Web site.

7.2 ZigBee Sample Programs

In order to run the sample programs discussed in this chapter and elsewhere in this manual,

1. Dynamic C must be installed and running on your PC.
2. The programming cable must connect the programming header on the RabbitCore module to your PC.
3. Power must be applied to the BL4S230.
4. The Digi® XBee USB used as the ZigBee coordinator must be connected to an available USB port on your PC if you are exercising the ZigBee protocol.

Refer to Chapter 2, “Getting Started,” if you need further information on these steps.

> **NOTE:** The Digi XBee USB device is an optional accessory and is not a part of the standard BL4S200 Tool Kit. See section F.2 Digi® XBee USB Configuration for more information on the Digi XBee USB device.

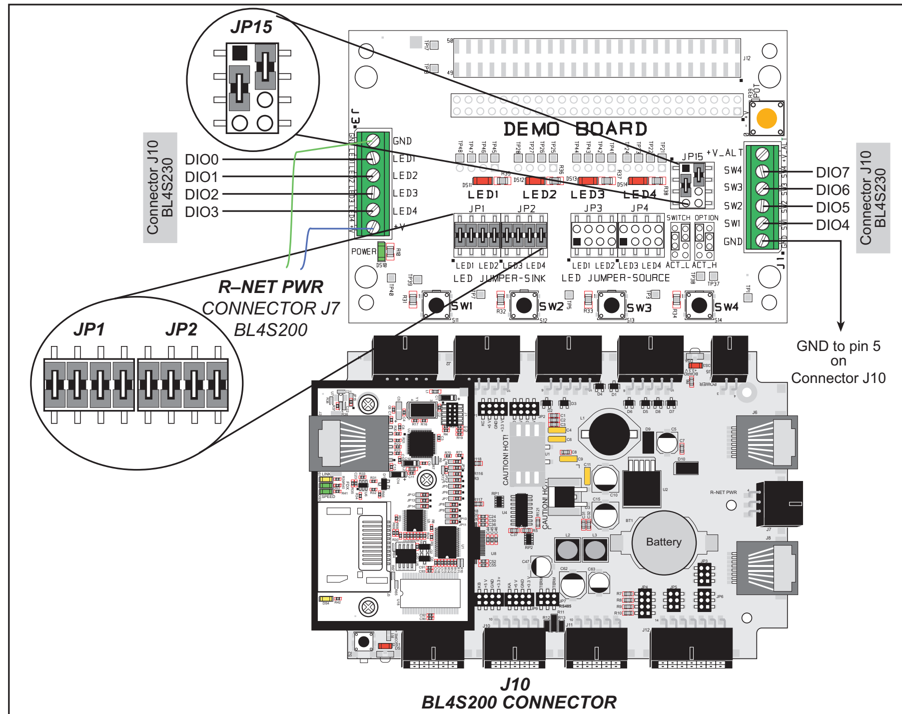
To run a sample program, open it with the **File** menu (if it is not still open), then compile and run it by pressing **F9**.

Each sample program has comments that describe the purpose and function of the program. Follow the instructions at the beginning of the sample program.

The sample programs in the Dynamic C **SAMPLES\XBee** folders illustrate the use of the ZigBee function calls.

7.2.1 Setting Up the Digi XBee USB Coordinator

1. Connect the Digi® XBee USB acting as a ZigBee coordinator to an available USB port on your PC or workstation. Your PC should recognize the new USB hardware.
2. Connect the Demonstration Board to the BL4S230 as shown below.



3. Compile and run the **XBEE_GPIO_SERVER.C** sample program in the Dynamic C **SAMPLES\BLxS2xx\XBee** folder.
4. Open the ZigBee Utility by double-clicking **XBEE_GPIO_GUI.exe** in the Dynamic C **Utilities\XBee GPIO GUI** folder — if you have problems launching the ZigBee Utility, install a .Net Framework by double-clicking **dotnetfx.exe** in the Dynamic C **Utilities\dotnetfx** folder. You may add a shortcut to the ZigBee Utility on your desktop.

5. Confirm the following hardware setup is displayed on the “PC Settings” tab.

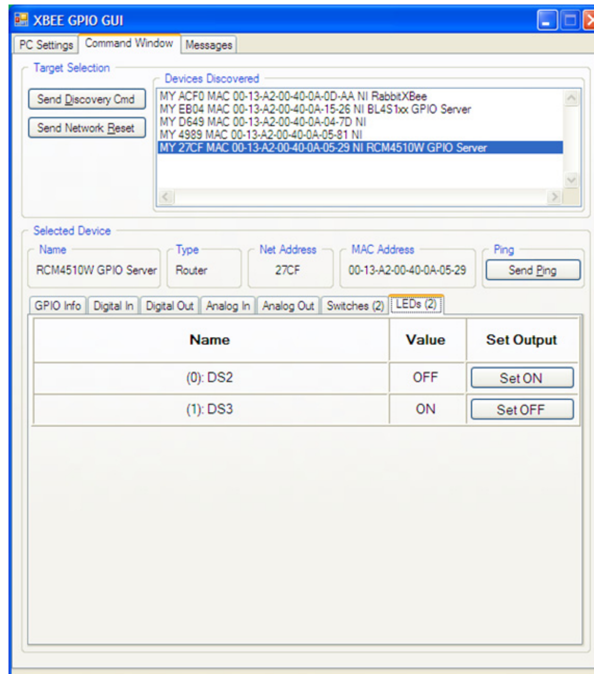
- 115200 baud
- Hardware flow control
- 8 data bits
- No parity
- 1 stop bit

Now select the COM port the Digi® XBee USB is connected to, and click the “Open Com Port” button. The message “Radio Found” is displayed to indicate that you selected the correct COM port. The ZigBee parameters (firmware version, operating channel, PAN ID) for the Digi® XBee USB will be displayed in the “Radio Parameters” box. Go to **Control Panel > System > Hardware > Device Manager > Ports** on your PC if you need help in identifying the USB COM port.

6. Any ZigBee devices discovered will be displayed in the “Devices Discovered” window to the right.

   If the utility times out and no ZigBee devices are displayed, you will have to reconfigure the Digi® XBee USB and recompile the sample program once you make sure the BL4S230 is powered up. The timeout may occur if you are doing development simultaneously with more than one ZigBee coordinator. Appendix F explains the steps to reconfigure the Digi® XBee USB.

7. Select a device with your mouse pointer and click on the selected device to select that device. This device will now be displayed in the “Selected Device” area.



8. You are now ready to interface with the BL4S230 via the ZigBee protocol. Try pinging the selected device by clicking the “Send Ping” button.

7.2.2 Setting up Sample Programs

The sample programs are set up so that the BL4S230 you are using is a ZigBee router or coordinator. Uncomment the line corresponding to the role the BL4S230 will have once it is running the sample program. The default in the sample programs is for the BL4S230 to be a router.

```
// Set XBEE_ROLE to NODE_TYPE_COORD, NODE_TYPE_ROUTER or NODE_TYPE_ENDDEV
// to match your XBee's firmware.
#define XBEE_ROLE   NODE_TYPE_ROUTER
```

NOTE: Remember that the firmware loaded to the XBee RF module is different depending on whether the BL4S230 is a router (default) or a coordinator. See Appendix F, “Additional Configuration Instructions,” for information on how to download firmware to the BL4S230 to set it up as a coordinator or to resume its original configuration as a router.

There are several macros that may be changed to facilitate your setup. The macros can be included as part of the program code, or they may be put into the Program Options “Defines” on the “Defines” tab in the **Options > Program Options** menu.

Channel mask — defaults to 0x1FFE, i.e., all 16 possible channels via the macro in the Dynamic C **LIB\Rabbit4000\XBee\XBee_Firmware\XBEE_API.LIB** library. If you want to limit the channels used, all devices on your network should use the same channel mask.

```
#define DEFAULT_CHANNELS XBEE_DEFAULT_CHANNELS
```

Extended PAN ID — the 64-bit network ID. Defaults to **DEFAULT_PANID** if set in the Dynamic C **LIB\Rabbit4000\XBee\XBEE_API.LIB** library, otherwise defaults to **0x0123456789abcdef** to match the default used on the Digi® XBee USB.

If set to **0x00**, tells coordinators to "select a random extended PAN ID," and tells routers and end devices to "join any network."

Change the extended PAN ID if you are developing simultaneously with more than one ZigBee coordinator.

```
#define DEFAULT_EXTPANID "0x0123456789abcdef"
```

Node ID — the ID of your particular node via the macro in the Dynamic C **LIB\Rabbit4000\XBee\XBee_Firmware\XBEE_API.LIB** library. Each node should have a unique identifier.

```
#define NODEID_STR "RabbitXBee"
```

The XBee sample programs in the Dynamic C **SAMPLES\XBee** folder illustrate the use of the XBee function calls.

- **AT_INTERACTIVE.C**—This sample program shows how to set up and use AT commands with the XBee RF module.

  The program will print out a list of AT commands in the Dynamic C **STDIO** window. You may type in either "ATxx" or just the "xx" part of the command.

  - Use just the AT command to read any of the values.
  - Use [AT]xx yyyy (where the y is an integer up to 32 bits) to set any of the "set or read" values. (Note that this works for NI, the *node identifier*, where the data will be a Node ID.string in quotes — [AT]NI "Node ID string" where the quotes contain the string data)
  - Type "menu" to redisplay the menu of commands.
  - Press **F4** to exit and close the **STDIO** window.

- **AT_RUNONCE.C**—This sample program uses many of the most important and useful AT commands. Several commands can either set a parameter or read it. This sample program simply reads the parameters and displays the results.

  Compile and run this sample program. The program will display the results in the Dynamic C **STDIO** window.

The XBee sample program in the Dynamic C **SAMPLES\BLxS2xx\XBee** folder illustrates the use of the XBee function calls.

- **XBEE_GPIO_SERVER.C**—This sample program shows how to set up and use endpoints and clusters. It is meant to be run with the Windows GUI client (installed in Dynamic C's **Utilities** directory) and a Digi USB XBee coordinator or with the GPIO client sample program (**SAMPLES/XBEE/XBEE_GPIO_CLIENT.C**) running on an RCM4510W RabbitCore module or on a single-board computer with an XBee RF module.

  Connect the BL4S230 to the Demonstration Board as explained in Section 7.2.1. Then compile and run this sample program on the BL4S230. Run the Windows GUI client on your PC. Configure the GUI client (**XBEE_GPIO_GUI.exe** in the Dynamic C **Utilities\XBee GPIO GUI** folder) to connect to the Digi USB XBee coordinator and scan for devices. Make sure the BL4S230 and the Digi USB XBee coordinator are configured with the same PAN ID.

  If you run the **XBEE_GPIO_CLIENT.C** sample program on another board with an XBee RF module, set the PAN IDs to match between the client and the server sample programs.

  Now select the GPIO server and use the GUI interface on the PC, or the command-line client on another XBee-equipped board to view the server's inputs and change its outputs.

7.3 Dynamic C Function Calls

Function calls for use with the XBee RF modules are in the Dynamic C **`LIB\Rabbit4000\XBee\XBEE_API.LIB`** library. These ZigBee specific function calls are described in ***An Introduction to ZigBee***, which is included in the online documentation set.

7.4 Where Do I Go From Here?

> **NOTE:** If you purchased your BL4S230 through a distributor or through a Rabbit partner, contact the distributor or partner first for technical support.

If there are any problems at this point:

- Use the Dynamic C **Help** menu to get further assistance with Dynamic C.
- Check the Rabbit Technical Bulletin Board and forums at www.rabbit.com/support/bb/ and at www.rabbit.com/forums/.
- Use the Technical Support e-mail form at www.rabbit.com/support/.

If the sample programs ran fine, you are now ready to go on.

An Introduction to ZigBee provides background information on the ZigBee protocol, and is available on the CD and on our Web site.

Digi's ***XBee™ Series 2 OEM RF Modules*** provides complete information for the XBee RF module used on the BL4S230, provides background information on the ZigBee protocol, and is available at ftp1.digi.com/support/documentation/90000976_a.pdf.

Appendix A. Specifications

Appendix A provides the specifications for the BL4S200 and describes the conformal coating.

A.1 Electrical and Mechanical Specifications

Figures A-1(a) and A-1(b) show the mechanical dimensions for the BL4S200.

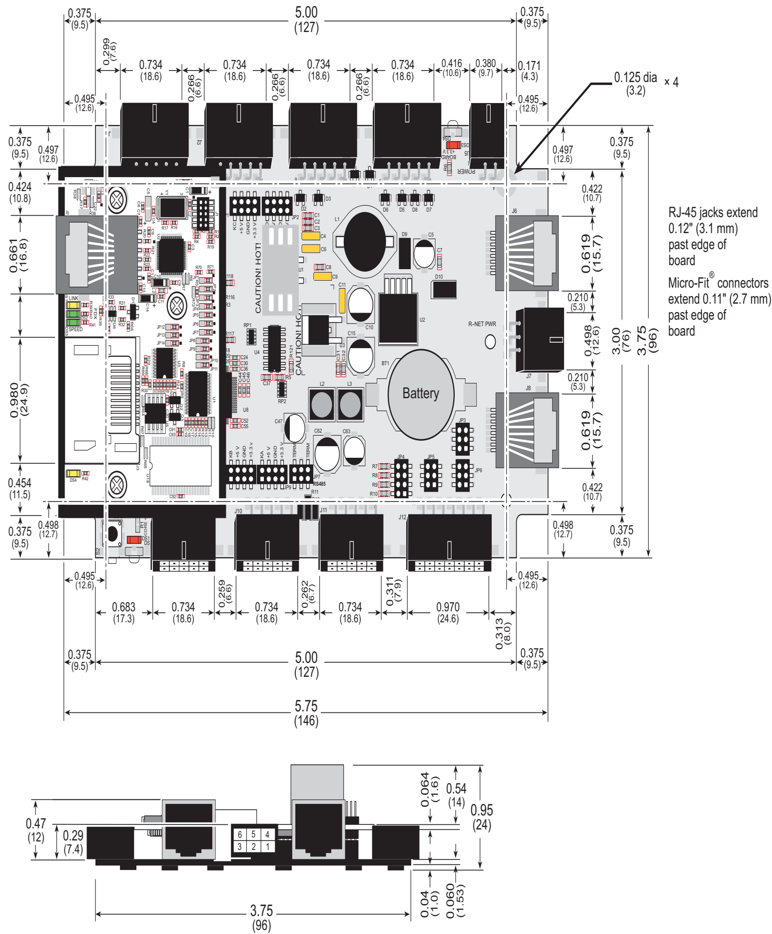


Figure A-1(a). BL4S200/BL4S210 Dimensions

NOTE: All measurements are in inches followed by millimeters enclosed in parentheses. All dimensions have a manufacturing tolerance of ±0.01" (0.25 mm).

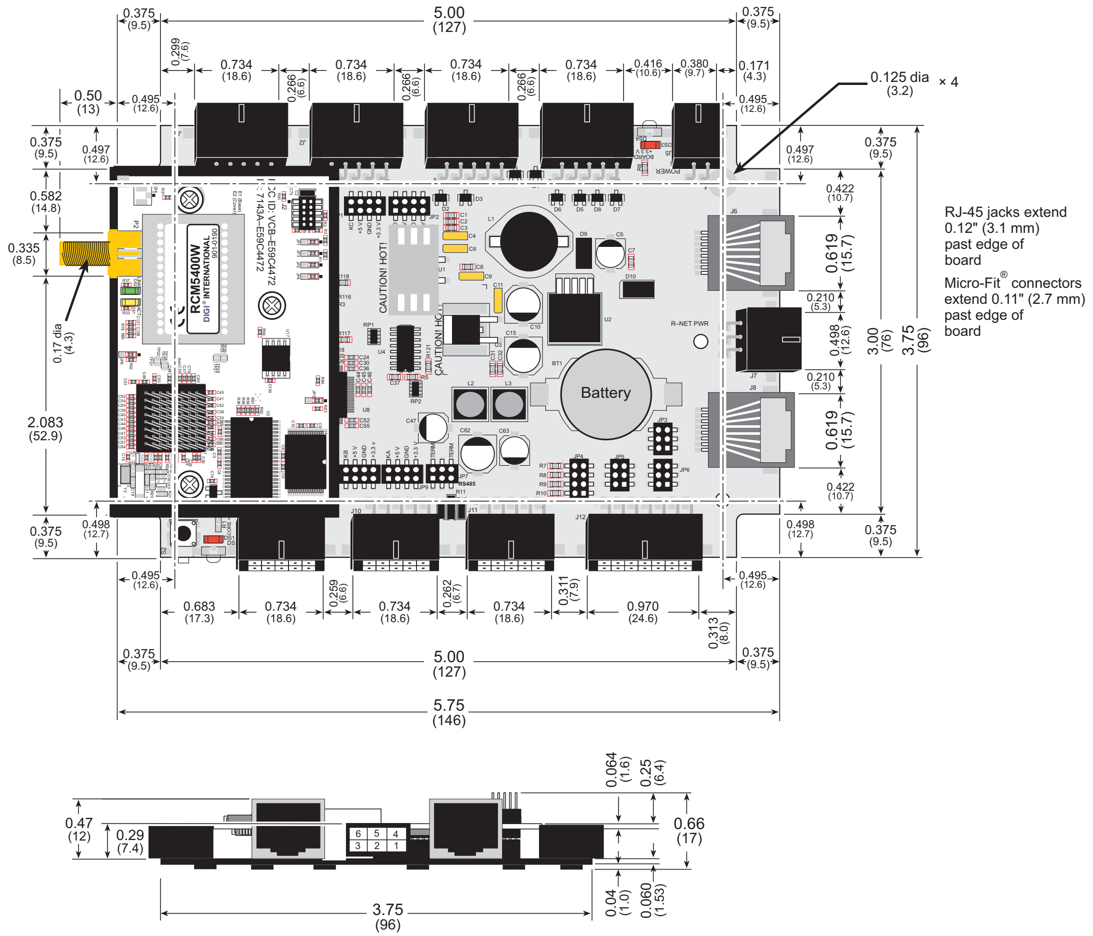


Figure A-1(b). BL5S220/BL4S230 Dimensions

NOTE: All measurements are in inches followed by millimeters enclosed in parentheses. All dimensions have a manufacturing tolerance of ±0.01" (0.25 mm).

Table A-1 lists the electrical, mechanical, and environmental specifications for the BL4S200.

Table A-1. BL4S200 Specifications

Feature	BL4S200	BL4S210	BL5S220	BL4S230
Microprocessor	Rabbit 4000® at 58.98 MHz		Rabbit® 5000 at 73.73 MHz	Rabbit 4000® at 29.49 MHz
Network Interface	10/100Base-T, 3 LEDs	10Base-T, 2 LEDs	Wi-Fi (802.11b/g)	ZigBee 2007 (802.15.4)
Flash Memory (program)	1MB (serial flash)	512KB (parallel flash)	512KB (parallel flash)	512KB (parallel flash)
Flash Memory (data storage)	supports *microSD™ Card* 128MB–1GB	—	1MB (serial flash)	
Program Execution SRAM	512KB	—	512KB	—
Data SRAM	512KB	512KB	512KB	512KB
Backup Battery	Renata CR2032 or equivalent 3 V lithium coin type, 235 mA·h standard, socket-mounted			
Configurable I/O	32 individually software-configurable I/O channels may be configured as digital inputs 0–36 V DC, switching threshold 1.4 V/1.9 V typical, or as sinking digital outputs up to 40 V, 200 mA each			
High-Current Digital Outputs	8 outputs individually software-configurable as sinking or sourcing, +40 V DC, 2 A max. per channel			
Analog Inputs	Eight 11-bit res. channels, software-selectable ranges unipolar: 1, 2, 2.5, 5, 10, 20 V DC; bipolar ± 1, ±2, ±5, ±10 V DC: 4 channels can be hardware-configured for 4–20 mA; 1 MΩ input impedance, up to 4,100 samples/s			
Analog Outputs	Two 12-bit res. channels, buffered, 0–10 V DC, ±10 VDC, and 4–20 mA, update rate 12 kHz			

Table A-1. BL4S200 Specifications (continued)

Feature	BL4S200	BL4S210	BL5S220	BL4S230
Serial Ports	5 serial ports: • one RS-485 • two RS-232 or one RS-232 (with CTS/RTS) • one clocked serial port multiplexed to two RS-422 SPI master ports • one serial port dedicated for programming/debug	4 serial ports: • one RS-485 • one RS-232 (no CTS/RTS)) • one clocked serial port multiplexed to two RS-422 SPI master ports • one serial port dedicated for programming/debug	5 serial ports: • one RS-485 • two RS-232 or one RS-232 (with CTS/RTS) • one clocked serial port multiplexed to two RS-422 SPI master ports • one serial port dedicated for programming/debug	
Serial Rate	Max. asynchronous rate = 250kbps, Max. synchronous rate = 1 MB/s			
Hardware Connectors	RJ-45 connectors: two RabbitNet™ Micro-Fit® connectors: seven polarized 2 × 5 with 3 mm pitch one polarized 2 × 7 with 3 mm pitch one polarized 2 × 2 with 3 mm pitch one polarized 2 × 3 with 3 mm pitch Programming port: 2 × 5 IDC, 1.27 mm pitch			
Network Connectors	One RJ-45 Ethernet		One RP-SMA antenna	—
Real-Time Clock	Yes			
Timers	Ten 8-bit timers (6 cascadable, 3 reserved for internal peripherals), one 10-bit timer with 2 match registers			
Watchdog/Supervisor	Yes			
Power	9–36 V DC, 4.5 W max.		9–36 V DC, 9.0 W max.	9–36 V DC, 4.5 W max.
Operating Temperature	-20°C to +85°C (–40°C to +85°C without *microSD™ Card*)	0 to +70°C	-30°C to +75°C	-40°C to +85°C
Humidity	5–95%, noncondensing			
Board Size	3.75" × 5.75" × 0.95" (96 mm × 146 mm × 24 mm)		3.75" × 5.75" × 0.66" (96 mm × 146 mm × 17 mm)	

A.1.1 Exclusion Zone

It is recommended that you allow for an “exclusion zone” of 0.25" (6 mm) around the BL4S200 in all directions when the BL4S200 is incorporated into an assembly that includes other components. This “exclusion zone” that you keep free of other components and boards will allow for sufficient air flow, and will help to minimize any electrical or EMI interference between adjacent boards. An “exclusion zone” of 0.12" (3 mm) is recommended below the BL4S200. Figure A-2 shows this “exclusion zone.”

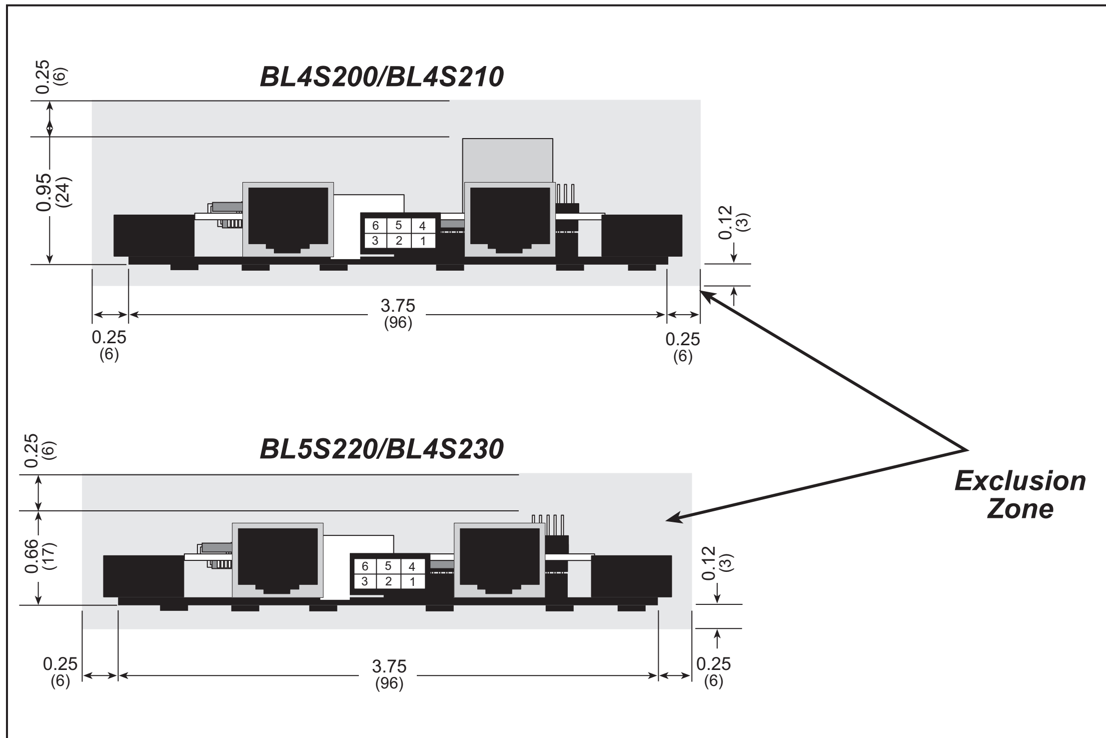


Figure A-2. BL4S200 “Exclusion Zone”

A.1.2 Headers

The BL4S200 has 3 mm Micro-Fit® connectors at J1, J2, J3, J9, J10, J11, and J12 for physical connection to other boards via wiring harnesses. There are 3 mm Micro-Fit® connectors at J5 and J7 for power-supply connections.

A.2 Conformal Coating

The areas around the crystal oscillator and the battery backup circuit on the BL4S200 modules based on the Rabbit 4000 microprocessor have had the Dow Corning silicone-based 1-2620 conformal coating applied. The conformal coating protects these high-impedance circuits from the effects of moisture and contaminants over time. Refer to the individual RabbitCore module's User's Manual for additional information on where the conformal coating was applied.

Any components in the conformally coated area may be replaced using standard soldering procedures for surface-mounted components. A new conformal coating should then be applied to offer continuing protection against the effects of moisture and contaminants.

> **NOTE:** For more information on conformal coatings, refer to Technical Note 303, ***Conformal Coatings***.

A.3 Jumper Configurations

Figure A-3 shows the header locations used to configure the various BL4S200 options via jumpers.

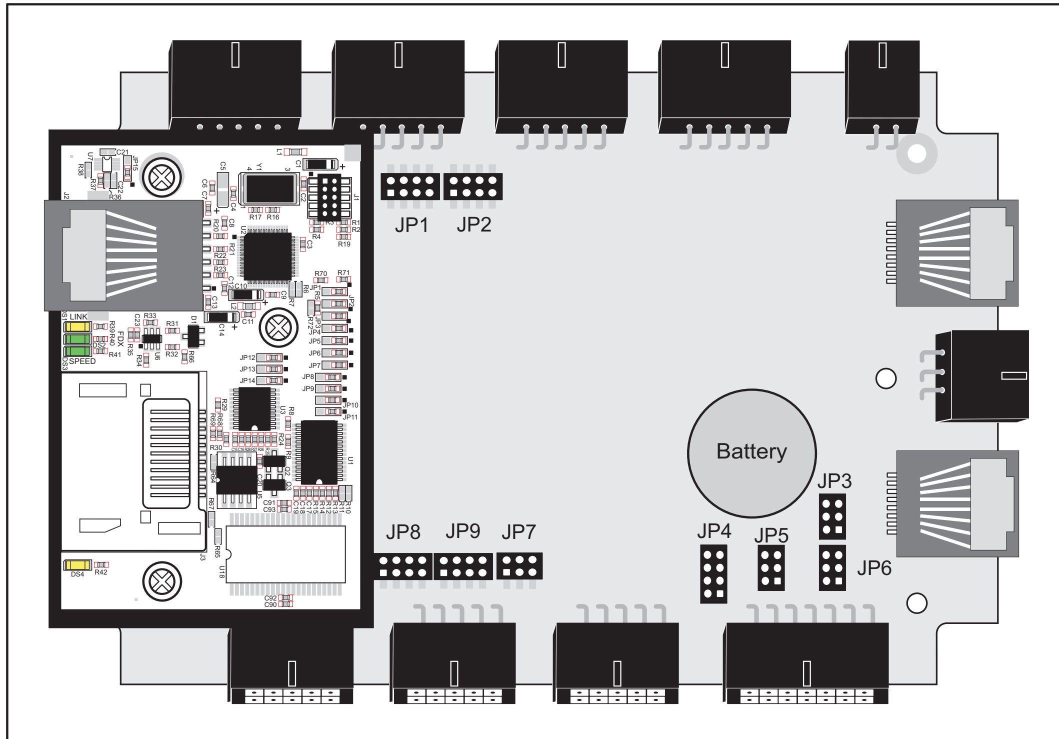


Figure A-3. Location of BL4S200 Configurable Positions

Table A-2 lists the configuration options.

Table A-2. BL4S200 Jumper Configurations

Header	Description	Pins Connected		Factory Default
JP1	DIO16–DIO23	1–2	Inputs pulled up to +KC	
		3–4	Inputs pulled up to +5 V	×
		5–6	Inputs pulled down to GND	
		7–8	Inputs pulled up to +3.3 V	
JP2	DIO24–DIO31	1–2	Inputs pulled up to +KD	
		3–4	Inputs pulled up to +5 V	×
		5–6	Inputs pulled down to GND	
		7–8	Inputs pulled up to +3.3 V	

Table A-2. BL4S200 Jumper Configurations

Header	Description	Pins Connected		Factory Default
JP3	AOUT0	1–2 3–4	0 to +10 V D/A converter output	×
		5–6	±10 V D/A convert output	
JP4	A/D Converter Voltage/Current Measurement Options	None	Voltage Option	×
		1–2	AIN0 4–20 mA option	
		3–4	AIN1 4–20 mA option	
		5–6	AIN2 4–20 mA option	
		7–8	AIN3 4–20 mA option	
JP5	AOUT0	1–3	D/A converter voltage output	×
		3–5	D/A converter current output	
	AOUT1	2–4	D/A converter voltage output	×
		4–6	D/A converter current output	
JP6	AOUT1	1–2 3–4	0 to +10 V D/A converter output	×
		5–6	±10 V D/A convert output	
JP7	RS-485 Bias and Termination Resistors	1–2 5–6	Bias and termination resistors connected	×
		1–3 4–6	Bias and termination resistors ***not*** connected*	
JP8	DIO8–DIO15	1–2	Inputs pulled up to +KB	
		3–4	Inputs pulled up to +5 V	×
		5–6	Inputs pulled down to GND	
		7–8	Inputs pulled up to +3.3 V	
JP9	DIO0–DIO7	1–2	Inputs pulled up to +KA	
		3–4	Inputs pulled up to +5 V	×
		5–6	Inputs pulled down to GND	
		7–8	Inputs pulled up to +3.3 V	

* Although pins 1–3 and 4–6 of header JP7 are shown “jumpered” for the termination and bias resistors ***not*** connected, pins 3 and 4 are not actually connected to anything, and this configuration is a “parking” configuration for the jumpers so that they will be readily available should you need to enable the termination and bias resistors in the future.

A.4 Use of Rabbit Microprocessor Parallel Ports

Table A-3 lists the Rabbit microprocessor parallel ports and their use in the BL4S200 boards.

Table A-3. Use of Rabbit Microprocessor Parallel Ports

Port	I/O	Signal		Initial State
PA0–PA7	I/O	ID0–ID7		Pulled up
PB0	Input	ADC busy (BL4S210)		Pulled up
PB1	Input	Not connected		Pulled up
PB2–PB7	Output	IA0–IA5		High
PC0	Output	TXD SPI	Serial Port D	Inactive high
PC1	Input	RXD SPI		Pulled up
PC2	Output	TXC RS-485	Serial Port C	Inactive high
PC3	Input	RXC RS-485		Pulled up
PC4	Output	TXB RS-232 (BL4S210)	Serial Port B	Inactive high
PC5	Input	RXB RS-232 (BL4S210)		Pulled up
PC6	Output	TXA Programming Port	Serial Port A	Low
PC7	Input	RXA Programming Port		Pulled up
PD0–PD1	Output/ENET	Not connected		
PD2	Output	TXF RS-232 (except BL4S210)	Serial Port F	Low
PD3	Output	RXF RS-232 (except BL4S210)		Low
PD4–PD5	Output	Not connected		Low
PD6	Output	TXE RS-232 (except BL4S210)	Serial Port E	Low
PD7	Output	RXE RS-232 (except BL4S210)		Low
PE0	Output	RIO I/O enable		Inactive high
PE1	Input	RIO interrupt input		Pulled up
PE2	Output	RIO Global Synch		Low
PE3	Output	SPI SCLKD		Inactive high
PE4	Input	ADC busy		Pulled up
PE5	Input	Not connected		Pulled up
PE6–PE7	Output	Not connected		Low

APPENDIX B. POWER SUPPLY

Appendix B describes the power circuitry provided on the BL4S200.

B.1 Power Supplies

Power is supplied to the BL4S200 boards via the Micro-Fit® connector at J5. The BL4S200 is protected against reverse polarity by a diode at D10 as shown in Figure B-1.

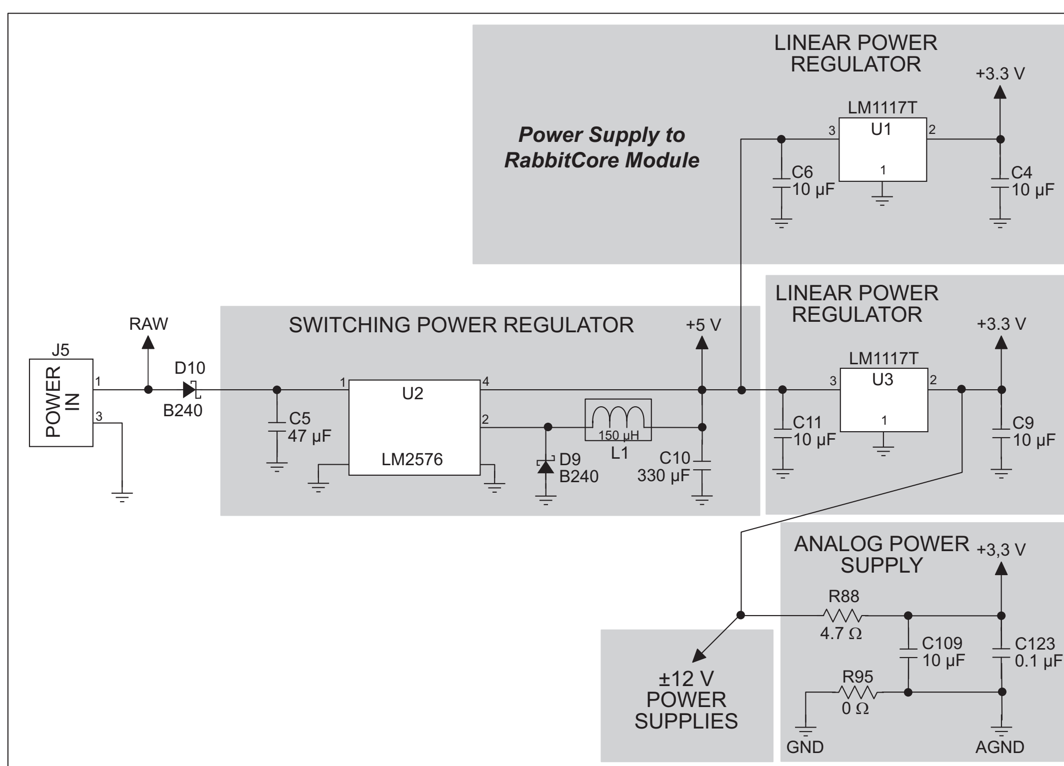


Figure B-1. BL4S200 Power Supply

The input voltage range is from 9 V to 36 V. A switching power regulator is used to provide +5 V for the BL4S200 logic circuits. In turn, the regulated +5 V DC power supply is used to drive two regulated +3.3 V power supplies and ±12 V power supplies used by the op-amps driving the outputs. A separate +3.3 V power supply is provided for the RabbitCore module to ensure adequate capacity for its circuits.

The digital ground and the analog ground share a single split ground plane on the board, with the analog ground connected at a single point to the digital ground by a 0 Ω resistor

(R95). This is done to minimize digital noise in the analog circuits and to eliminate the possibility of ground loops. External connections to analog ground are made on a Micro-Fit® connector at J12.

B.1.1 Power for Analog Circuits

Power to the analog circuits is provided by way of a one-stage low-pass filter, which isolates the analog section from digital noise generated by the other components. The analog +3.3 V supply powers the D/A converter, and is not accessible to the user. The A/D converter is powered by the regulated +3.3 V supply, and supplies the +2.5 V reference violate from which the 1.116 V, 1.95 V, and 3.00 V reference voltages for the D/A converter output circuits are derived.

> **NOTE:** The +12 V power supplies used to drive the analog outputs only have sufficient current output in their design to supply the analog output channels. Do not use these voltage supplies for other applications.

B.2 Batteries and External Battery Connections

The SRAM and the real-time clock on the BL4S200 modules have battery backup. Power to the SRAM and the real-time clock (VRAM) is provided by two different sources, depending on whether the main part of the BL4S200 is powered or not. When the BL4S200 is powered normally, and the +3.3 V supply is within operating limits, the SRAM and the real-time clock are powered from the +3.3 V supply. If power to the board is lost or falls below 2.93 V, the VRAM and real-time clock power will come from the battery. The reset generator circuit controls the source of power by way of its **/RESET** output signal.

A replaceable 235 mA·h lithium battery provides power to the real-time clock and SRAM when external power is removed from the circuit board. The drain on the battery is typically less than 10 µA when there is no external power applied to the BL4S200, and so the expected shelf life of the battery is

$$\frac{235\ \text{mA}\cdot\text{h}}{10\ \mu\text{A}} = 2.7\ \text{years.}$$

The actual battery life in your application will depend on the current drawn by components not on the BL4S200 and on the storage capacity of the battery. The BL4S200 does not drain the battery while it is powered up normally.

B.2.1 Replacing the Backup Battery

The battery is user-replaceable, and is fitted in a battery holder. To replace the battery, lift up on the spring clip and slide out the old battery. Use only a Renata CR2032 or equivalent replacement battery, and insert it into the battery holder with the + side facing up.

> **NOTE:** The SRAM contents and the real-time clock settings will be lost if the battery is replaced with no power applied to the BL4S200. Exercise care if you replace the battery while external power is applied to the BL4S200.

CAUTION: There is an explosion danger if the battery is short-circuited, recharged, or replaced incorrectly. Replace the battery only with the same type or an equivalent type recommended by the battery manufacturer. Dispose of used batteries according to the battery manufacturer's instructions.

Cycle the main power off/on after you install a backup battery for the first time, and whenever you replace the battery. This step will minimize the current drawn by the real-time clock oscillator circuit from the backup battery should the BL4S200 experience a loss of main power.

> **NOTE:** Remember to cycle the main power off/on any time the RabbitCore module is removed from the BL4S200 main board since that is where the backup battery is located.

Rabbit's Technical Note TN235, ***External 32.768 kHz Oscillator Circuits***, provides additional information about the current draw by the real-time clock oscillator circuit.

B.3 Power to Peripheral Cards

DCIN and Vcc are available on Micro-Fit® connector J7 to power RabbitNet peripheral boards that may be used with the BL4S200.

Keep in mind that the BL4S200 draws 377 mA from the Vcc supply, and that the diode at D10 (shown in Figure B-1) can handle at most 2 A at V_{RAW}, so that leaves the remaining current capacity to be shared among the DCIN and Vcc pins on Micro-Fit® connector J7. Table B-1 lists the available current at DCIN based on the current drawn at Vcc.

Table B-1. DCIN Current Available at J7 (in mA)
Based on Power Supply and Vcc (= 5 V) Current Used at J7

V_{RAW} Power Supply Input at J2 (V)	Current at J7 with Vcc = 5 V						
	100 mA	200 mA	300 mA	400 mA	500 mA	600 mA	623 mA
8.0	545	450	355	260	164	69	47
8.5	574	484	395	306	216	127	107
9.0	599	515	431	347	263	178	159
10	641	566	490	415	340	265	248
12	703	641	579	517	455	393	378
18	805	764	723	682	642	601	591
24	855	824	794	763	733	703	696
30	884	860	836	811	787	763	750
40	913	895	877	859	841	823	819

For example, if the raw power supply input is 12 V, and the Vcc supply at J7 draws 200 mA, 641 mA will be available for DCIN.

APPENDIX C. DEMONSTRATION BOARD

Appendix C explains how to use the Demonstration Board with the BL4S200 sample programs.

C.1 Connecting Demonstration Board

Before running sample programs based on the Demonstration Board, you will have to connect the Demonstration Board from the BL4S200 Tool Kit to the BL4S200 board. Proceed as follows.

1. Use the 6-position connector to bare leads wiring harness included in the BL4S200 Tool Kit to connect screw-terminal header J3 on the Demonstration Board to connector J7 on the BL4S200. The connections are shown in Figure C-1, with the green wire to GND and the blue wire to +V.
2. Make sure that your BL4S200 is connected to your PC via the programming cable and that the power supply is connected to the BL4S200 and plugged in as described in Chapter 2, "Getting Started."

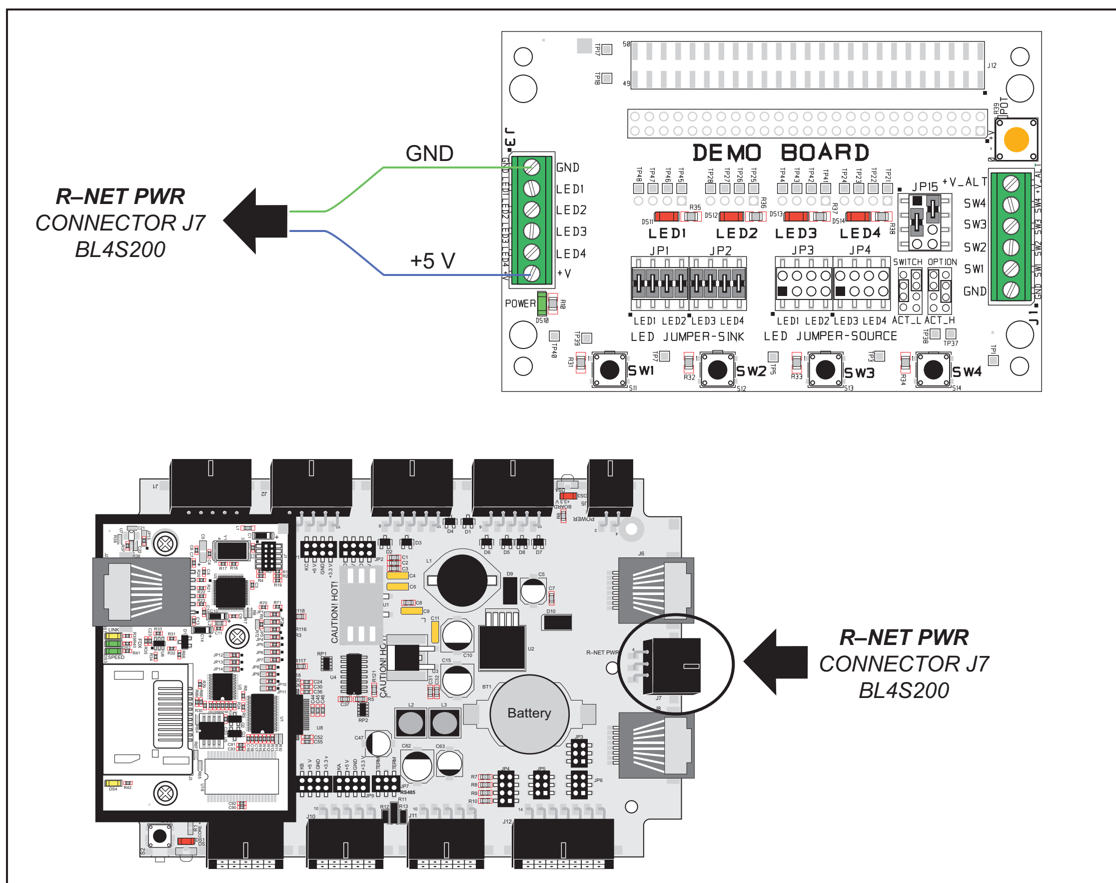


Figure C-1. Power Supply Connections Between BL24S60 and Demonstration Board

CAUTION: If you are using your own power supply with the Demonstration Board, note that the maximum power supply input voltage the Demonstration Board can handle is + 12 V DC. Do not use a higher power supply voltage.

C.2 Demonstration Board Features

The Demonstration Board can be used to illustrate I/O activity via LEDs and pushbutton switches.

C.2.1 Pinout

Figure C-2 shows the pinouts for the input signals on screw-terminal header J1 and the outputs on screw-terminal header J3.

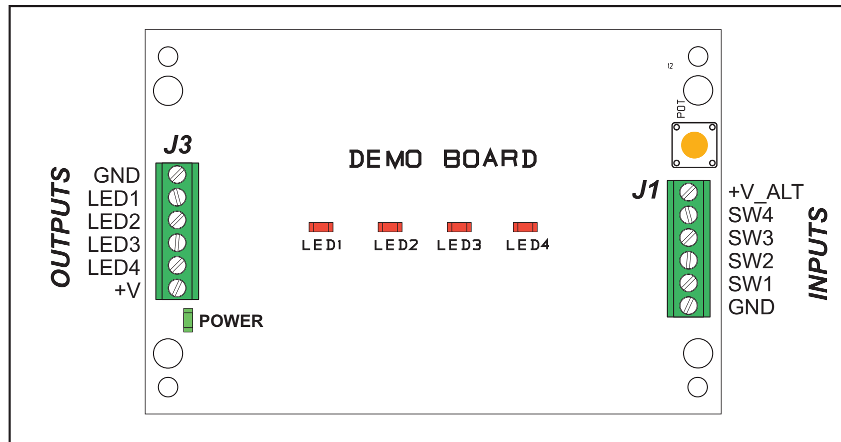


Figure C-2. Demonstration Board Pinout

C.2.2 Configuration

The pushbutton switches may be configured active high or active low via jumper settings on header JP15.

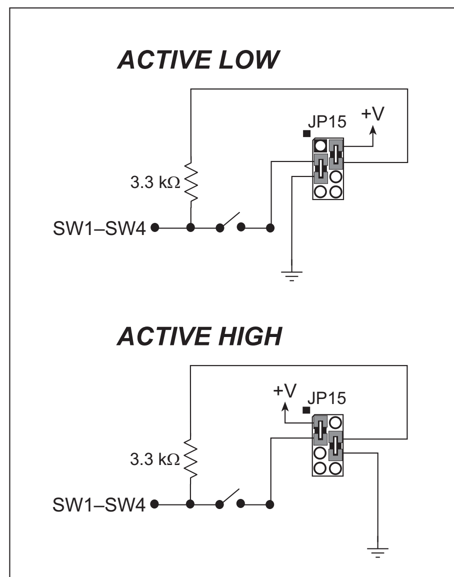


Figure C-3. Pushbutton Switch Configuration

The four LED output indicators can be configured as sinking outputs or as sourcing outputs via jumpers on headers JP1–JP4 as shown in Figure C-4.

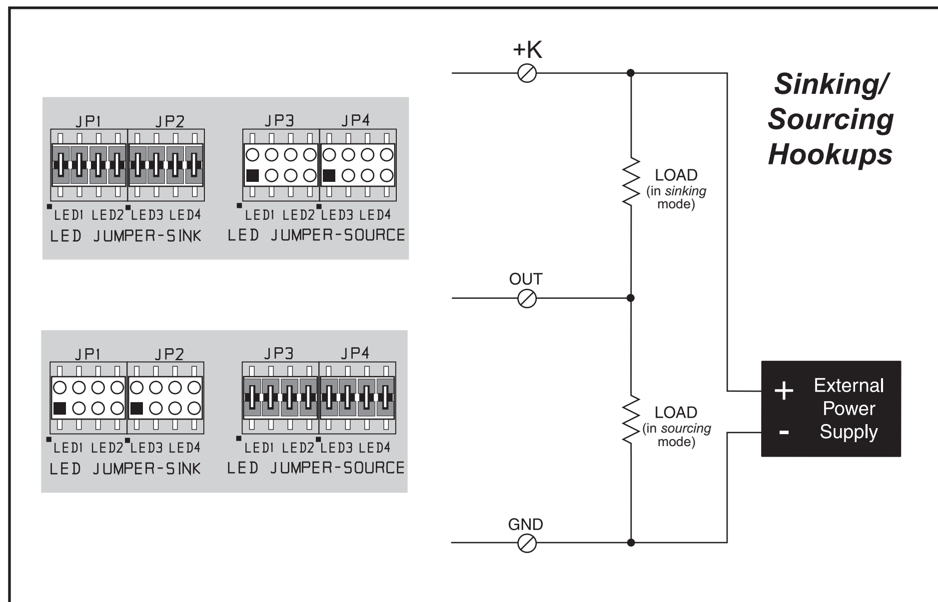


Figure C-4. LED Output Indicators Sinking or Sourcing Configuration

The power supply voltage input at +V on screw-terminal header J3 is available as +V_ALT on screw-terminal header J1. There is a potentiometer immediately above the +V_ALT location to allow you to reduce the voltage from the +V originally input.

Figure C-5 shows the location of the adjustable output voltage and the potentiometer.

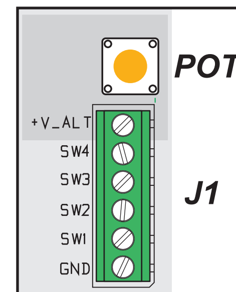


Figure C-5. Location of Adjustable Output Voltage

Appendix D. Rabbit RIO Resource Allocation

Appendix D provides the pin and block associations on the Rabbit RIO chips with their corresponding I/O on the BL4S200 boards. The main shared resource within the RIO chips are the counter/timer blocks — each RIO chip has eight counter/timer blocks. The BL4S200 boards have three RIO chips, which gives a total of 24 blocks. A given block is defined by both the RIO number and the block number. The tables in this appendix provide a quick reference of which block is used by each input and/or output pin on the BL4S200 board.

D.1 Configurable I/O Pin Associations

Table D-1. Configurable I/O Pin Associations

<table>
<tr><th rowspan="2">I/O Pin</th><th rowspan="2">RIO Chip</th><th colspan="2">Inputs</th><th colspan="2">Outputs</th></tr>
<tr><th>Block</th><th>Pin</th><th>Block</th><th>Pin</th></tr>
<tr><td>DIO0</td><td rowspan="4">0</td><td rowspan="4">5</td><td>0</td><td rowspan="4">4</td><td>0</td></tr>
<tr><td>DIO1</td><td>1</td><td>1</td></tr>
<tr><td>DIO2</td><td>2</td><td>2</td></tr>
<tr><td>DIO3</td><td>3</td><td>3</td></tr>
<tr><td>DIO4</td><td rowspan="12">1</td><td rowspan="4">1</td><td>0</td><td rowspan="4">0</td><td>0</td></tr>
<tr><td>DIO5</td><td>1</td><td>1</td></tr>
<tr><td>DIO6</td><td>2</td><td>2</td></tr>
<tr><td>DIO7</td><td>3</td><td>3</td></tr>
<tr><td>DIO8</td><td rowspan="4">3</td><td>0</td><td rowspan="4">2</td><td>0</td></tr>
<tr><td>DIO9</td><td>1</td><td>1</td></tr>
<tr><td>DIO10</td><td>2</td><td>2</td></tr>
<tr><td>DIO11</td><td>3</td><td>3</td></tr>
<tr><td>DIO12</td><td rowspan="4">5</td><td>0</td><td rowspan="4">4</td><td>0</td></tr>
<tr><td>DIO13</td><td>1</td><td>1</td></tr>
<tr><td>DIO14</td><td>2</td><td>2</td></tr>
<tr><td>DIO15</td><td>3</td><td>3</td></tr>
<tr><td>DIO16</td><td rowspan="12">2</td><td rowspan="2">0</td><td>0</td><td rowspan="2">0</td><td>2</td></tr>
<tr><td>DIO17</td><td>1</td><td>3</td></tr>
<tr><td>DIO18</td><td rowspan="2">1</td><td>0</td><td rowspan="2">1</td><td>2</td></tr>
<tr><td>DIO19</td><td>1</td><td>3</td></tr>
<tr><td>DIO20</td><td rowspan="2">2</td><td>0</td><td rowspan="2">2</td><td>2</td></tr>
<tr><td>DIO21</td><td>1</td><td>3</td></tr>
<tr><td>DIO22</td><td rowspan="2">3</td><td>0</td><td rowspan="2">3</td><td>2</td></tr>
<tr><td>DIO23</td><td>1</td><td>3</td></tr>
<tr><td>DIO24</td><td rowspan="2">4</td><td>0</td><td rowspan="2">4</td><td>2</td></tr>
<tr><td>DIO25</td><td>1</td><td>3</td></tr>
<tr><td>DIO26</td><td rowspan="2">5</td><td>0</td><td rowspan="2">5</td><td>2</td></tr>
<tr><td>DIO27</td><td>1</td><td>3</td></tr>
<tr><td>DIO28</td><td>0</td><td>7</td><td>0</td><td>6</td><td>0</td></tr>
<tr><td>DIO29</td><td>1</td><td>7</td><td>0</td><td>6</td><td>0</td></tr>
<tr><td>DIO30</td><td rowspan="2">2</td><td>6</td><td>0</td><td rowspan="2" colspan="2">Not controlled by any RIO chip</td></tr>
<tr><td>DIO31</td><td>7</td><td>0</td></tr>
</table>

D.2 High-Current Output Pin Associations

Table D-2. High-Current Output Pin Associations

High-Current Output Pin	RIO Chip	Block	Sinking	Sourcing
			Pin	Pin
HOUT0	0	0	0	1
HOUT1			2	3
HOUT2		1	0	1
HOUT3			2	3
HOUT4		2	0	1
HOUT5			2	3
HOUT6		3	0	1
HOUT7			2	3

D.3 Interpreting Error Codes

Some BL4S200 function calls may return a *Mode Conflict* error code. This error code is a 4-bit value that identifies other pins using the same counter/timer block on a particular RIO chip that require this block to be in a mode that conflicts with the functionality that has already been requested — the additional functionality requested cannot be supported. The error code also helps you identify the other pins whose functionality needs to change to possibly allow the latest function call to succeed.

The bit values in the *Mode Conflict* error codes have the following meanings.

- Bits [7:4] don’t matter, will always be zero
- Bit 3 — Pin 3 of this block has a mode conflict
- Bit 2 — Pin 2 of this block has a mode conflict
- Bit 1 — Pin 1 of this block has a mode conflict
- Bit 0 — Pin 0 of this block has a mode conflict

By looking at the tables in this appendix, you can identify the other pins that share the RIO counter/timer block with the pin(s) that returned the *Mode Conflict* error code. For example, if you already configured DIO8 and DIO9 as Quadrature Decoder inputs, then try to set DIO11 as a counter input, the function call will return a *Mode Conflict* error code of 3.

This error code is a 4-bit value that identifies other pins other pins using the same counter/timer block that conflict with the requested function. In this case, 3 is 0011, which indicates that pin 1 and pin 2 of the block used by DIO11 have the conflicts — they are using the counter/timer in a way that conflicts with setting DIO11 as a counter input. Looking at Table D-1, you find DIO11 is using block 3 on RIO chip 1, and pin 0 and pin 1 of this block are used by DIO8 and DIO9. Therefore you cannot use DIO11 as a counter input unless you remove the Quadrature Decoder inputs from this block. This illustrates how the *Mode Conflict* error code can be used to identify the pin functions that cannot mix together on the same RIO block.

The tables in this appendix are useful for both finding the cause of mode conflicts, and for planning which pins to use for which functions to avoid conflicts in the first place. Note that pins DIO30 and DIO31 do not have their output functionality controlled by a RIO chip. This means that you cannot assign PWM or PPM functions to these two pins. On the other hand, both of these pins have a counter block that is not shared with any other pins on their inputs, which makes these pins ideal for use as counter or capture inputs.

Notice that there is a pattern to the block sharing of certain configurable I/O pins. The first 16 configurable I/O pins, DIO0—DIO15, have blocks shared across four inputs or four outputs. These are the only pins that can support functions such as Quadrature Decoder inputs with an independent index-based reset. The next group of 12 configurable I/O pins (DIO16–DIO27) share blocks among their configurable I/O pairs, bringing both the input and output functionality of these pins into the same block. This allows PWM or PPM outputs that can be used with an external synchronization signal. It would also allow synchronization of a pulse capture response to a PWM-based output pulse. The last 4 configurable I/O pins have nonshared RIO blocks available for both the input and output functionality, making these pins ideal for single-pin functions requiring a counter/timer.

Table D-3 shows all counter/timer modes of the RIO block and which functions can use the given modes. The use of synch signals is allowed with all the functions, but does affect the timer/counter so it may have an adverse affect on functions marked with * or #.

Table D-3. RIO Counter/Timer Block Mode Summary

	Up Count	Count Until Match	Up/Down Count	Free-Running Timer	Count Until End	Count from Begin to End	Count While Begin Is Active
Digital Input	×	×	×	×	×	×	×
Digital Output	×	×	×	×	×	×	×
Event Counter Input	*	*	*				
Event Capture Input				#	*	*	*
Quad. Decoder Input			*				
Ext. Interrupt Input	×	×	×	×	×	×	×
External Synch Input	×	×	×	×	×	×	×
PWM/PPM Output				#			

× — I/O are compatible with the given mode, and can work with any other function using that mode.
* — I/O cannot share the block with any other * or # marked function without possible conflicts.
# — I/O can generally share the timer, but will be affected by settings of the limit value (value at which the timer rolls over) or resetting of the counter, either directly or through synch signals.

NOTE: The Rabbit RIO is driven by a 16Mhz Spread Spectrum Clock.

Appendix E. RabbitNet

E.1 General RabbitNet Description

RabbitNet is a high-speed synchronous protocol developed by Rabbit to connect peripheral cards to a master and to allow them to communicate with each other.

E.1.1 RabbitNet Connections

All RabbitNet connections are made point to point. A RabbitNet master port can only be connected directly to a peripheral card, and the number of peripheral cards is limited by the number of available RabbitNet ports on the master.

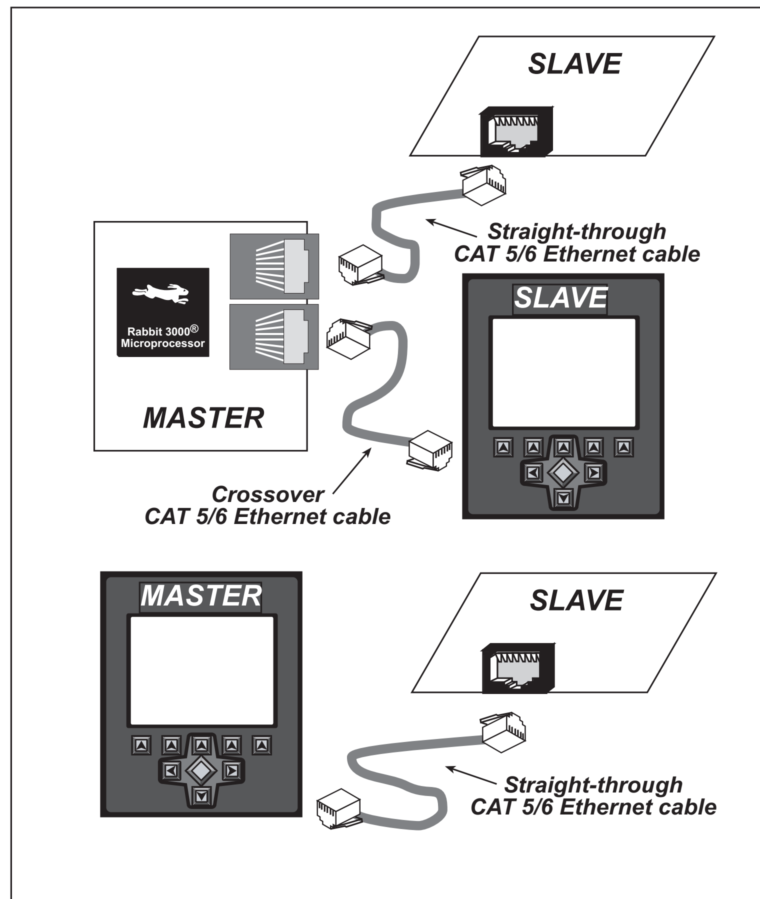


Figure 1. Connecting Peripheral Cards to a Master

Use a straight-through CAT 5/6 Ethernet cable to connect the master to slave peripheral cards, unless you are using a device such as the OP7200 that could be used either as a master or a slave. In this case you would use a crossover CAT 5/6 Ethernet cable to connect an OP7200 that is being used as a slave.

Distances between a master unit and peripheral cards can be up to 10 m or 33 ft.

E.1.2 RabbitNet Peripheral Cards

- Digital I/O

  24 inputs, 16 push/pull outputs, 4 channels of 10-bit A/D conversion with ranges of 0 to 10 V, 0 to 1 V, and -0.25 to +0.25 V. The following connectors are used:

  Signal = 0.1" friction-lock connectors
  Power = 0.156" friction-lock connectors
  RabbitNet = RJ-45 connector

- A/D converter

  8 channels of programmable-gain 12-bit A/D conversion, configurable as current measurement and differential-input pairs. 2.5 V reference voltage is available on the connector. The following connectors are used:

  Signal = 0.1" friction-lock connectors
  Power = 0.156" friction-lock connectors
  RabbitNet = RJ-45 connector

- D/A converter

  8 channels of 0–10 V 12-bit D/A conversion. The following connectors are used:

  Signal = 0.1" friction-lock connectors
  Power = 0.156" friction-lock connectors
  RabbitNet = RJ-45 connector

- Display/Keypad interface

  Allows you to connect your own keypad with up to 64 keys and one character liquid crystal display from 1 × 8 to 4 × 40 characters with or without backlight using standard 1 × 16 or 2 × 8 connectors. The following connectors are used:

  Signal = 0.1" headers or sockets
  Power = 0.156" friction-lock connectors
  RabbitNet = RJ-45 connector

- Relay card

  6 relays rated at 250 V AC, 1200 V·A or 100 V DC up to 240 W. The following connectors are used:

  Relay contacts = screw-terminal connectors
  Power = 0.156" friction-lock connectors
  RabbitNet = RJ-45 connector

Visit our Web site for up-to-date information about additional cards and features as they become available. The Web site also has the latest revision of this user's manual.

E.2 Physical Implementation

There are four signaling functions associated with a RabbitNet connection. From the master's point of view, the transmit function carries information and commands to the peripheral card. The receive function is used to read back information sent to the master by the peripheral card. A clock is used to synchronize data going between the two devices at high speed. The master is the source of this clock. A slave select (SS) function originates at the master, and when detected by a peripheral card causes it to become selected and respond to commands received from the master.

The signals themselves are differential RS-422, which are series-terminated at the source. With this type of termination, the maximum frequency is limited by the round-trip delay time of the cable. Although a peripheral card could theoretically be up to 45 m (150 ft) from the master for a data rate of 1 MHz, Rabbit recommends a practical limit of 10 m (33 ft).

Connections between peripheral cards and masters are done using standard 8-conductor CAT 5/6 Ethernet cables. Masters and peripheral cards are equipped with RJ-45 8-pin female connectors. The cables may be swapped end for end without affecting functionality.

E.2.1 Control and Routing

Control starts at the master when the master asserts the slave select signal (SS). Then it simultaneously sends a serial command and clock. The first byte of a command contains the address of the peripheral card if more than one peripheral card is connected.

A peripheral card assumes it is selected as soon as it receives the select signal. For direct master-to-peripheral-card connections, this is as soon as the master asserts the select signal. The connection is established once the select signal reaches the addressed slave. At this point communication between the master and the selected peripheral card is established, and data can flow in both directions simultaneously. The connection is maintained so long as the master asserts the select signal.

E.3 Function Calls

The function calls described in this section are used with all RabbitNet peripheral cards, and are available in the **RNET.LIB** library in the Dynamic C **RABBITNET** folder.

---

rn_init

```
int rn_init(char portflag, char servicetype);
```

FUNCTION DESCRIPTION

Resets, initializes, or disables a specified RabbitNet port on the master single-board computer. During initialization, the network is enumerated and relevant tables are filled in. If the port is already initialized, calling this function forces a re-enumeration of all devices on that port.

Call this function first before using other RabbitNet functions.

PARAMETERS

portflag bit that represents a RabbitNet port on the master single-board computer (from 0 to the maximum number of ports). A set bit requires a service. If **portflag** = 0x03, both RabbitNet ports 0 and 1 will need to be serviced.

servicetype enables or disables each RabbitNet port as set by the port flags.

0 = disable port
1 = enable port

RETURN VALUE

0

rn_device

```
int rn_device(char pna);
```

FUNCTION DESCRIPTION

Returns an address index to device information from a given physical node address. This function will check device information to determine that the peripheral card is connected to a master.

PARAMETER

pna	the physical node address, indicated as a byte. 7,6—2-bit binary representation of the port number on the master 5,4,3—Level 1 router downstream port 2,1,0—Level 2 router downstream port

RETURN VALUE

Pointer to device information. -1 indicates that the peripheral card either cannot be identified or is not connected to the master.

SEE ALSO

rn_find

rn_find

```
int rn_find(rn_search *srch);
```

FUNCTION DESCRIPTION

Locates the first active device that matches the search criteria.

PARAMETER

srch search criteria structure **rn_search**:

```
unsigned int flags;     // status flags see MATCH macros below
unsigned int ports;     // port bitmask
char productid;         // product id
char productrev;        // product rev
char coderev;           // code rev
long serialnum;         // serial number
```

Use a maximum of 3 macros for the search criteria:

```
RN_MATCH_PORT          // match port bitmask
RN_MATCH_PNA           // match physical node address
RN_MATCH_HANDLE        // match instance (reg 3)
RN_MATCH_PRDID         // match id/version (reg 1)
RN_MATCH_PRDREV        // match product revision
RN_MATCH_CODEREV       // match code revision
RN_MATCH_SN            // match serial number
```

For example:

```
rn_search newdev;
newdev.flags = RN_MATCH_PORT|RN_MATCH_SN;
newdev.ports = 0x03; //search ports 0 and 1
newdev.serialnum = E3446C01L;
handle = rn_find(&newdev);
```

RETURN VALUE

Returns the handle of the first device matching the criteria. 0 indicates no such devices were found.

SEE ALSO

rn_device

rn_echo

```
int rn_echo(int handle, char sendecho, char *recdata);
```

FUNCTION DESCRIPTION

The peripheral card sends back the character the master sent. This function will check device information to determine that the peripheral card is connected to a master.

PARAMETERS

handle	address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.
sendecho	character to echo back.
recdata	pointer to the return address of the character from the device.

RETURN VALUE

The status byte from the previous command. -1 means that device information indicates the peripheral card is not connected to the master.

rn_write

```
int rn_write(int handle, int regno, char *data, int datalen);
```

FUNCTION DESCRIPTION

Writes a string to the specified device and register. Waits for results. This function will check device information to determine that the peripheral card is connected to a master.

PARAMETERS

handle	address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.
regno	command register number as designated by each device.
data	pointer to the address of the string to write to the device.
datalen	number of bytes to write (0–15).

NOTE: A data length of 0 will transmit the one-byte command register number.

RETURN VALUE

The status byte from the previous command. -1 means that device information indicates the peripheral card is not connected to the master, and -2 means that the data length was greater than 15.

SEE ALSO

rn_read

rn_read

```
int rn_read(int handle, int regno, char *recdata, int datalen);
```

FUNCTION DESCRIPTION

Reads a string from the specified device and register. Waits for results. This function will check device information to determine that the peripheral card is connected to a master.

PARAMETERS

handle address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.

regno command register number as designated by each device.

recdata pointer to the address of the string to read from the device.

datalen number of bytes to read (0–15).

NOTE: A data length of 0 will transmit the one-byte command register number.

RETURN VALUE

The status byte from the previous command. -1 means that device information indicates the peripheral card is not connected to the master, and -2 means that the data length was greater than 15.

SEE ALSO

rn_write

rn_reset

```
int rn_reset(int handle, int resettype);
```

FUNCTION DESCRIPTION

Sends a reset sequence to the specified peripheral card. The reset takes approximately 25 ms before the peripheral card will once again execute the application. Allow 1.5 seconds after the reset has completed before accessing the peripheral card. This function will check peripheral card information to determine that the peripheral card is connected to a master.

PARAMETERS

handle	address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.
resettype	describes the type of reset. 0 = hard reset—equivalent to power-up. All logic is reset. 1 = soft reset—only the microprocessor logic is reset.

RETURN VALUE

The status byte from the previous command. -1 means that device information indicates the peripheral card is not connected to the master.

rn_sw_wdt

```
int rn_sw_wdt(int handle, float timeout);
```

FUNCTION DESCRIPTION

Sets software watchdog timeout period. Call this function prior to enabling the software watchdog timer. This function will check device information to determine that the peripheral card is connected to a master.

PARAMETERS

handle	address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.
timeout	timeout period from 0.025 to 6.375 seconds in increments of 0.025 seconds. Entering a zero value will disable the software watchdog timer.

RETURN VALUE

The status byte from the previous command. -1 means that device information indicates the peripheral card is not connected to the master.

rn_enable_wdt

```
int rn_enable_wdt(int handle, int wdttype);
```

FUNCTION DESCRIPTION

Enables the hardware and/or software watchdog timers on a peripheral card. The software on the peripheral card will keep the hardware watchdog timer updated, but will hard reset if the time expires. The hardware watchdog cannot be disabled except by a hard reset on the peripheral card. The software watchdog timer must be updated by software on the master. The peripheral card will soft reset if the timeout set by **`rn_sw_wdt()`** expires. This function will check device information to determine that the peripheral card is connected to a master.

PARAMETERS

`handle` address index to device information. Use **`rn_device()`** or **`rn_find()`** to establish the handle.

`wdttype` 0 enables both hardware and software watchdog timers
1 enables hardware watchdog timer
2 enables software watchdog timer

RETURN VALUE

The status byte from the previous command. -1 means that device information indicates the peripheral card is not connected to the master.

SEE ALSO

`rn_hitwd, rn_sw_wdt`

rn_hitwd

```
int rn_hitwd(int handle, char *count);
```

FUNCTION DESCRIPTION

Hits software watchdog. Set the timeout period and enable the software watchdog prior to using this function. This function will check device information to determine that the peripheral card is connected to a master.

PARAMETERS

handle	address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.
count	pointer to return the present count of the software watchdog timer. The equivalent time left in seconds can be determined from **count** × 0.025 seconds.

RETURN VALUE

The status byte from the previous command. -1 means that device information indicates the peripheral card is not connected to the master.

SEE ALSO

rn_enable_wdt, rn_sw_wdt

rn_rst_status

```
int rn_rst_status(int handle, char *retdata);
```

FUNCTION DESCRIPTION

Reads the status of which reset occurred and whether any watchdogs are enabled.

PARAMETERS

handle address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.

retdata pointer to the return address of the communication byte. A set bit indicates which error occurred. This register is cleared when read.

7—HW reset has occurred
6—SW reset has occurred
5—HW watchdog enabled
4—SW watchdog enabled
3,2,1,0—Reserved

RETURN VALUE

The status byte from the previous command.

rn_comm_status

```
int rn_comm_status(int handle, char *retdata);
```

FUNCTION DESCRIPTION

If the communication error bit is set in the status byte, use this function call to determine the error.

PARAMETERS

handle address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.

retdata pointer to the return address of the communication byte. A set bit indicates which error occurred. This register is cleared when read.

7—Data available and waiting to be processed MOSI (master out, slave in)
6—Write collision MISO (master in, slave out)
5—Overrun MOSI (master out, slave in)
4—Mode fault, device detected hardware fault
3—Data compare error detected by device
2,1,0—Reserved

RETURN VALUE

The status byte from the previous command.

E.3.1 Status Byte

Unless otherwise specified, functions returning a status byte will have the following format for each designated bit.

7	6	5	4	3	2	1	0	
×	×							00 = Reserved 01 = Ready 10 = Busy 11 = Device not connected
		×						0 = Device 1 = Router
			×					0 = No error 1 = Communication error*
				×				Reserved for individual peripheral cards
					×			Reserved for individual peripheral cards
						×		0 = Last command accepted 1 = Last command unexecuted
							×	0 = Not expired 1 = HW or SW watchdog timer expired†

* Use the function **`rn_comm_status()`** to determine which error occurred.

† Use the function **`rn_rst_status()`** to determine which timer expired.

APPENDIX F. ADDITIONAL CONFIGURATION INSTRUCTIONS

Appendix F provides information on how to find the latest firmware for the XBee RF module and the Digi® XBee USB used as the ZigBee coordinator, and how to install the firmware.

F.1 XBee Module Firmware Downloads

By default, the BL4S230 is shipped from the factory with firmware to operate as a router in a mesh network. You will need to run the **MODEMFWLOAD.C** sample program in the Dynamic C **SAMPLES\XBEE** folder to download the firmware needed to operate the BL4S230 as a coordinator.

> **NOTE:** You can verify the firmware version by running the **AT_INTERACTIVE.C** sample program in the Dynamic C **SAMPLES\XBEE** folder and by entering the command ATVR **<Enter>** to get the version number displayed in the Dynamic C **STDIO** window.

> **CAUTION:** Different firmware versions are likely to interact with the Dynamic C libraries in different ways. Rabbit has tested the firmware associated with a particular version of Dynamic C for correct operation, and only this version is included on the Dynamic C CD-ROM — do not use any other firmware versions with that version of Dynamic C.

Once you have successfully loaded the firmware, compile and run another sample program to make sure the **MODEMFWLOAD.C** sample program does not inadvertently reload (or partially reload) the firmware.

If you are uploading firmware because you upgraded to a more recent Dynamic C release, remember to recompile your applications using the new version of Dynamic C once you have uploaded the new firmware.

F.1.1 Dynamic C v. 10.44 and Later

Encrypted libraries have been created within Dynamic C for the firmware. The three libraries are in the **LIB\Rabbit4000\XBee\XBee_Firmware\ZigBee** folder.

- A Dynamic C library of the type **XB24-ZB_21**….**.LIB** is used for a coordinator BL4S230.
- A Dynamic C library of the type **XB24-ZB_23**….**.LIB** is used for a router BL4S230.

Make the following modifications to the **MODEMFWLOAD.C** sample program.before you run it according to whether you will be using the BL4S230 as a coordinator or a router.

- Select the XBee role macro according to whether the BL4S230 is being used as a coordinator or a router.

```
#define XBEE_ROLE NODE_TYPE_COORD
#define XBEE_ROLE NODE_TYPE_ROUTER
```

- Some Rabbit boards use the ZNet 2.5 protocol. The BL4S230 uses the ZB protocol. Make sure the **#define** statement calls for the ZB protocol.

```
#define XBEE_PROTOCOL XBEE_PROTOCOL_ZB
```

F.2 Digi® XBee USB Configuration

The Digi XBee USB device is an optional accessory and is available as a part of the Mesh Networking Add on Kit (101-1272), or for separate purchase (101-1286). It is not a part of the standard BL4S200 Tool Kit.

You may experience difficulty when you use the ZigBee sample programs and the Digi® XBee USB with the default settings if you are working simultaneously with more than one ZigBee coordinator.

Section 7.2.2 explains how to set up the BL4S230 configuration patterns for the sample programs via macros in the Dynamic C **LIB\Rabbit4000\XBee\XBEE_API.LIB** library folder.

Channel mask — defaults to 0x1FFE, i.e., all 16 possible channels via the macro in the Dynamic C **LIB\Rabbit4000\XBee\XBEE_API.LIB** library.

```
#define DEFAULT_CHANNELS ZB_DEFAULT_CHANNELS
```

For example, to limit the channels to three channels, the macro would read as follows.

```
#define DEFAULT_CHANNELS 0x000E
```

Extended PAN ID — the 64-bit network ID. Defaults to **DEFAULT_PANID** if set in the Dynamic C **LIB\Rabbit4000\XBee\XBEE_API.LIB** library, otherwise defaults to **0x0123456789abcdef** to match the default used on the Digi® XBee USB.

If set to **0x00**, tells coordinators to “select a random extended PAN ID,” and tells routers and end devices to “join any network.”

Change the extended PAN ID if you are developing simultaneously with more than one ZigBee coordinator.

```
#define DEFAULT_EXTPANID "0x0123456789abcdef"
```

The same configurations must then be applied to the Digi® XBee USB via Digi’s X-CTU utility. If you have not previously used this utility, install it from the Dynamic C **Utilities\X-CTU** folder by double-clicking **Setup_x-ctu.exe**,

Continue the following steps with the Digi® XBee USB connected to your PC's USB port. Since the ZigBee Utility **XBEE_GPIO_GUI.exe** will conflict with X-CTU, first close the ZigBee Utility if it is running.

1. Start X-CTU from the desktop icon and set the "PC Settings" tab to **115200** baud, **HARDWARE** flow control, **8** data bits, parity **NONE**, **1** stop bit.
2. On the "PC Settings" tab, check the "Enable API" box under "Host Setup."
3. On the "PC Settings" tab, select the "USB Serial Port" corresponding to the USB serial port the Digi® XBee USB is connected to and click "Test/Query." You should see a response showing the Modem Type (XB 24-B) and the firmware version. Click **OK**.

   Note that several USB serial ports could be listed. If you select a serial port without the Digi® XBee USB connected, the X-CTU response to "Test/Query." will be "communication with modem … OK," but the modem type will be unknown, and the firmware version will be blank.

   If you a get a message that X-CTU is unable to open the COM port, verify that you selected the COM port with the "USB Serial Port," then try unplugging the Digi® XBee USB from the USB slot and plugging it back in. Now click "Test/Query" again.
4. Under the "Modem Configuration" tab click the "Read" button. X-CTU will now display the networking and I/O parameters for the Digi® XBee USB being used as the ZigBee coordinator.

   Modem: XBEE XB24-ZB

   Function Set: ZIGBEE COORDINATOR API (do not select other settings)

   Version: the version of the firmware included with the version of Dynamic C you are using (should be of the type **21**...)
5. Now change the networking parameters to match the parameters in the Dynamic C **LIB\Rabbit4000\XBee\XBEE_API.LIB** library.

   Networking

   (D) CH - Operating Channel — this is the operating channel you could see when you ran the **AT_INTERACTIVE.C** sample program in the Dynamic C **SAMPLES\XBEE** folder by entering the command ATCH <**Enter**>. This channel information cannot be changed from the X-CTU utility.

   (0123456789ABCDEF) ID - Extended Pan ID — set the new extended PAN ID that follows 0x to match the **DEFAULT_EXTPANID** macro.

   (1FFE) SC - Scan Channels - set the new value for the channels to scan, E, for example, to match the new setting in the macro.

   ```
   #define DEFAULT_CHANNELS 0x000E
   ```
6. Finish by clicking the "Write" button.

F.2.1 Additional Reference Information

Check Digi's Web site for the latest information and documentation on the XBee Series 2 module, the X-CTU utility, and the Digi® XBee USB. Note that the XBee™ and the XBee PRO™ RF modules are presently not compatible with the XBee Series 2 module used

with the BL4S230, but the general documentation about ZigBee and the use of AT commands for the XBee™ and the XBee PRO™ RF modules is relevant.

F.2.2 Update Digi® XBee USB Firmware

The firmware version used by the Digi® XBee USB must correspond to the firmware version installed on the BL4S230. If you have updated the BL4S230 firmware (or you have a need to re-install the firmware on the Digi® XBee USB), the corresponding firmware for the Digi® XBee USB is in the Dynamic C **`Utilities\X-CTU\MODEMFW`** folder.

- Remember to record the extended PAN ID, NI, and other parameters you are using.
- Firmware of the type **`XB24-ZB_21`**….**`zip`** is used for the Digi® XBee USB coordinator.

> **CAUTION:** Different firmware versions are likely to interact with the Dynamic C libraries in different ways. Rabbit has tested the firmware associated with a particular version of Dynamic C for correct operation, and only this version is included on the Dynamic C CD-ROM — do not use any other firmware versions with that version of Dynamic C.

1. Start X-CTU from the desktop icon and set the "PC Settings" tab to **115200** baud, **HARDWARE** flow control, **8** data bits, parity **NONE**, **1** stop bit.
2. On the "PC Settings" tab, check the "Enable API" box under "Host Setup."
3. On the "PC Settings" tab, select the "USB Serial Port" and click "Test/Query." You should see a response showing the Modem Type (XB 24-B) and the firmware version. Click **OK**.

   Note that several USB serial ports could be listed. If you select a serial port without the Digi® XBee USB connected, the X-CTU response to "Test/Query." will be "communication with modem … OK," but the modem type will be unknown, and the firmware version will be blank.

   If you a get a message that X-CTU is unable to open the COM port, verify that you selected the COM port with the "USB Serial Port," then try unplugging the Digi® XBee USB from the USB slot and plugging it back in. Now click "Test/Query" again.
4. Under the "Modem Configuration" tab click the "Read" button. X-CTU will now display the networking and I/O parameters for the Digi® XBee USB.

   Modem: XBEE XB24-ZB

   Function Set: ZIGBEE COORDINATOR API (do not select other settings)

   Version: the version of the firmware included with the version of Dynamic C you are using
5. Under the "Modem Configuration" tab click the "Download new versions…" button, select "File," and browse to the **`Utilities\X-CTU\MODEMFW`** subfolder, then click "Open" when you have selected the firmware. (Do *not* select "Web," which will allow you to find the file on a Web site.) Remember to select firmware of the type **`XB24-ZB_21`**….**`zip`** that is used for a coordinator.

   The X-CTU utility will display an Update Summary box. Click "OK," and then click "Done."
6. Click the "Read" button, select XB24-B as the Modem type; select ZIGBEE COORDINATOR API as the Function Set, and 21… as the Version, then click "Write."
7. When the process is complete set the PANID, NI, and other parameters to the values you were using before the firmware was upgraded.

INDEX

A

A/D converter 40
  buffered inputs 40
  calibration 42
  calibration constants 41
  current-measurement setup 41
  function calls
    anaIn() 114
    anaInCalib() 112
    anaInConfig() 110
    anaInDiff() 118
    anaInDriver() 121
    anaInmAmps() 120
    anaInVolts() 116
analog I/O
  reference voltage circuit 45
  reference voltages 45
analog inputs *See* A/D converter
analog outputs *See* D/A converter
antenna
  extension 13

B

battery backup
  battery life 179
  use of battery-backed SRAM 131
battery connections 179
board initialization
  function calls 68
    brdInit() 68

C

CE compliance 11
  design guidelines 12
certifications 13
  Europe 15
  FCC 13
  Industry Canada 14
  labeling requirements 14
clock doubler 47
configurable I/O
  capture setup 32
  counter setup 32
  function calls
    getBegin() 32
    getEnd() 32
    resetCounter() 32
    setCapture() 32
    setCounter() 32
    setExtInterrupt() 32
    setPPM() 33
    setPWM() 33
  interrupts setup 32
  pin associations 187
  PWM/PPM setup 33
configuration
  BL4S200
    4–20 mA current 41
    configurable I/O inputs . 28
    configurable I/O outputs 29
    high-current outputs 34
    RS-485 termination and bias resistors 37
  Digi® XBee USB (ZigBee coordinator) 205
conformal coating 174
connections
  Ethernet cable 132
connectivity tools
  Connectivity Kit 9
  crimp tool 9
  Micro-Fit® connector parts .. 26

D

D/A converter 43
  calibration 44
  calibration constants 44
  function calls
    anaOut() 127
    anaOutCalib() 126
    anaOutConfig 123
    anaOutDriver() 130
    anaOutmAmps() 129
    anaOutPwrOff() 125
    anaOutStrobe() 124
    anaOutVolts() 128
Demonstration Board 8, 182
  configuration options 184
    LED outputs 185
    output voltage 185
    pushbutton switches 184
  hookup instructions 183
  maximum power-supply voltage 183
  pinout 184
  power supply connections 183
  wire assembly 8
Digi® XBee USB (ZigBee coordinator)
  configuration 205
  uploading new firmware . 208
digital I/O
  function calls
    digIn() 69
    digInBank() 70
    digOut() 82
    digOutBank() 83
    getBegin() 77
    getCounter() 77
    getEnd() 78
    globalSync() 81
    pulseDisable() 91
    pulseEnable() 91
    resetCounter() 78
    setCapture() 75
    setCounter() 73
    setDecoder() 72
    setDigIn() 69
    setDigOut() 81
    setDuty() 89
    setExtInterrupt() 71
    setFreq() 88
    setLimit() 79
    setOffset() 90
    setPPM() 86
    setPWM() 84
    setSync() 80
digital inputs

pullup/pulldown configuration 28, 29, 34
switching threshold 28
digital outputs 34
PWM/PPM setup 33
sinking or sourcing 34
dimensions
BL4S200 main board 169
Dynamic C 9, 51, 52
add-on modules 19, 53
installation 19
basic instructions 51
battery-backed SRAM 131
debugging features 52
installation 19
protected variables 131
Rabbit Embedded Security Pack 9, 53
standard features
debugging 52
starting 20
telephone-based technical support 9, 53
upgrades and patches 53

E

error codes
Mode Conflict 188
Ethernet cables 132
Ethernet connections 132
steps 132
Ethernet port 39
pinout 39
exclusion zone 173

F

features 6
firmware download
Digi® XBee USB 208
firmware updates 208
XBee module 204
coordinator vs. end device/ router 205
firmware updates . 204, 208
flash memory
lifetime write cycles 51
flash memory addresses
user blocks 49

H

high-current outputs
function calls
digOut_H() 93
digOutConfig_H() 92
digOutTriState_H() 95
digOutTriStateConfig_H() 94
setDuty_H() 101
setFreq_H() 100
setOffset_H() 102
setPPM_H() 33, 98
setPWM_H() 33, 96
setSync_H() 103
pin associations 188
PWM/PPM setup 33

I

interrupt handlers
function calls
addISR_H() 105
addISR() 104
enableISR() 107
setIER() 106
tickISR() 107
IP addresses
how to set 134
how to set PC IP address . 135

J

jumper configurations 175
JP1 (digital input DIO00–DIO07 pullup/pulldown configuration) 175
JP2 (digital input DIO08–DIO15 pullup/pulldown configuration) 175, 176
JP3 (digital input DIN16–DIN19 pullup/pulldown configuration) .. 28, 29, 34, 176
JP4 (digital input DIN20–DIN23 pullup/pulldown configuration) .. 28, 29, 30, 34
JP5 (digital input DIN24–DIN31 pullup/pulldown configuration) .. 28, 29, 34, 176
JP6 (A/D converter voltage/ current measurement options) 41, 176
JP7 (RS-485 bias and termination resistors) 37, 176
jumper locations 175

L

labeling requirements 14

M

memory 49
flash memory configurations 49
SRAM configuration for different sizes 49
Micro-Fit® connectors
parts 26
microSD™ Card
adapter 50
use with SD Card reader ... 50
Mode Conflict
error codes 188
models 7
BL4S200 7
BL4S210 7
BL4S230 7
BL5S220 7

O

operating region configuration . 146
options 9
Mesh Network Add-On Kit 9

P

pin associations
configurable I/O 187
high-current outputs 188
pinout
BL4S200 headers 25
Demonstration Board 184
Ethernet port 39
power management 178
power supply 178
battery backup 179
connections 17
RabbitNet peripheral cards ... 181
switching voltage regulator ... 178
Program Mode 46
programming
flash vs. RAM 51
programming cable 8
programming port 38
Remote Program Update 7
programming cable 8
connections 17
PROG connector 46

programming port 38

R

Rabbit microprocessor
parallel ports 177
tamper detection 49
VBAT RAM memory 49
RabbitNet 10
Ethernet cables to connect peripheral cards 190, 191
function calls
rn_comm_status() 202
rn_device() 194
rn_echo() 196
rn_enable_wdt() 199
rn_find() 195
rn_hitwd() 200
rn_init() 193
rn_read() 197
rn_reset() 198
rn_rst_status() 201
rn_sw_wdt() 198
rn_write() 196
general description 190
peripheral cards 191
A/D converter 191
D/A converter 191
digital I/O 191
display/keypad interface ... 191
relay card 191
physical implementation . 192
RabbitNet peripheral cards ... 10
connection to master 190, 191
power from BL4S200 181
power-supply connections 181
real-time clock
how to set 66
reset
hardware 18
RIO pin/block associations
cinfigurable I/O 187
high-current outputs 188
RS-232 36
RS-485 36
RS-485 network 37
termination and bias resistors 37
Run Mode 46

S

sample programs 54
A/D converter
AD_CAL_ALL.C 42
AD_CALDIFF_CH.C ... 42
AD_RD_DIFF.C 63
AD_RD_MA.C 63
AD_RD_SE_BIPOLAR.C 63
AD_RD_SE_UNIPOLAR.C 63
AD_RDVOLT_ALL.C . 42
ADC_CAL_DIFF.C 62
ADC_CAL_MA.C .. 44, 62
ADC_CAL_SE_BIPOLAR.C 62
ADC_CAL_SE_UNIPOLAR.C 62
ADC_RD_CALDATA.C .. 62
D/A converter
DAC_CAL_MA.C 64
DAC_CAL_VOLTS.C .. 64
DAC_MA_ASYNC.C .. 64
DAC_MA_SYNC.C 64
DAC_RD_CALDATA.C .. 65
DAC_VOLT_ASYNC.C .. 65
DAC_VOLT_SYNC.C . 65
digital I/O
DIGIN_BANK.C 55
DIGIN.C 55
DIGOUT_BANK.C 56
DIGOUT.C 56
HIGH_CURRENT_IO.C .. 57
INTERRUPTS.C 57
PPM.C 58
PULSE_CAPTURE.C .. 58
PWM.C 58
QUADRATURE_DECODER.C 59
how to set IP address 134
microSD™ Card
SDFLASH_INSPECT.C 66
PC/notebook configuration ... 144
PONG.C 20
real-time clock
RTC_TEST.C 66
SETRTCKB.C 66
save/retrieve calibration constants 23, 41, 62
serial communication
MASTER.C 61
PARITY.C 60
SIMPLE3WIRE.C 60
SIMPLE5WIRE.C 61
SLAVE.C 61
TCP_CONFIG.LIB 143
TCP/IP 66, 134
PINGME.C 136
SSI.C 137
TELNET.C 137
USERBLOCK_READ_WRITE.C 23, 41, 62
Wi-Fi
PINGLED_STATS.C .. 151
PINGLED_WPA_PSK.C .. 151
PINGLED_WPA2_CCMP.C 153
PINGLED.C 151
WIFI_SCAN.C 146, 150
WIFI_SCANASSOCIATE.C 150
WIFIDHCPORTSTATIC.C 148
WIFIMULTIPLEAPS.C ... 148
WIFIPINGYOU.C 149
Wi-Fi configuration macros .. 143
Wi-Fi network configuration 143
Wi-Fi regulatory setup
operating region configuration 146
REGION_COMPILETIME.C 146
REGION_MULTI_DOMAIN.C 147
REGION_RUNTIME_PING.C 147
XBee module
AT_INTERACTIVE.C 22, 165, 204, 206
AT_RUNONCE.C 165
MODEMFWLOAD.C 204, 205
XBEE_GPIO_SERVER.C 162, 166
ZigBee 161
ZigBee setup 161
SD Card adapter 50
serial communication 36
function calls
ser485Rx() 109
ser485Tx() 109
serMode() 108

programming port 38
RS-232 description 36
RS-485 description 36
RS-485 network 37
RS-485 termination and bias
resistors 37
serial ports
Ethernet port 39
setup 17
power supply connections . 17
software 9
libraries 67
BL4S200 67
BLxS2xx.LIB 67
PACKET.LIB 108
RN_CFG_BLS2xx.LIB 67
RNET.LIB 193
RS232.LIB 108
TCP_CONFIG.LIB 154
TCP/IP 67
Mode Conflict
error codes 188
RIO pin/block assocations
187, 188
sample programs 54
PONG.C 20
Wi-Fi configuration at com-
pile time 154
configuration macros ... 154
access point SSID 154
authentication 156
channel 155
enable/disable encryp-
tion 155
encryption keys 155
fragmentation threshold
157
mode 154
other macros 157
region/country 155
RTS threshold 157
select encryption key 155
set WPA hex key 156
set WPA passphrase 156
WPA encryption 156
network configuration . 154
TCPCONFIG macro ... 154
Wi-Fi configuration at run
time 158
specifications
BL4S200
dimensions 169
electrical 171
exclusion zone 173
headers 173
temperature 171
spectrum spreader
settings 48
status byte 203
subsystems 24

T

tamper detection 49
TCP/IP connections 132
10Base-T Ethernet card .. 132
additional resources 137
Ethernet hub 132
steps 132
Tool Kit 8
AC adapter 8
DC power supply 8
Demonstration Board 8
Dynamic C software 8
programming cable 8
software 8
User's Manual 8
wire assembly 8

U

user block
function calls
readUserBlock() 49
writeUserBlock() 49
save/retrieve calibration con-
stants 23, 41, 62

V

VBAT RAM memory 49

W

Wi-Fi
additional resources 159
bring interface down 158
bring interface up 158
function calls
ifconfig() 154, 158
ifconfig(IF_WIFI0,…) 158
ifdown(IF_WIFI0) 158
ifup(IF_WIFI0) 158
sock_init_or_exit(1) 158
sock_init() 158
tcp_tick(NULL) 158
sample programs 146

X

XBee modem
function calls 167
XBee module
additional resources 167
firmware download 204

Z

ZigBee protocol
coordinator 160
end device 160
introduction 160
mesh network 161
router 160

SCHEMATICS

090-0267 BL4S200 Schematic

www.digi.com/documentation/schemat/090-0267.pdf

090-0227 RCM4000 Schematic

www.digi.com/documentation/schemat/090-0227.pdf

090-0229 RCM4300 Schematic

www.digi.com/documentation/schemat/090-0229.pdf

090-0266 RCM5400W Schematic

www.digi.com/documentation/schemat/090-0266.pdf

090-0246 RCM4500W Schematic

www.digi.com/documentation/schemat/090-0246.pdf

090-0252 USB Programming Cable Schematic

www.digi.com/documentation/schemat/090-0252.pdf

090-0272 Rabbit Demonstration Board

www.digi.com/documentation/schemat/090-0272.pdf

You may use the URL information provided above to access the latest schematics directly.