RABBIT

PRODUCT MANUAL

Coyote (BL2500)

C-Programmable Single-Board Computer with Ethernet

User's Manual

019–0120_M

BL2500 User's Manual

Part Number 019-0120 • Printed in U.S.A.

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

Trademarks

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

Rabbit 3000, RabbitCore, and RabbitNet are trademarks of Digi International Inc.

The latest revision of this manual is available on the Rabbit Web site, www.rabbit.com.

Digi International Inc.

www.rabbit.com

TABLE OF CONTENTS

1. INTRODUCTION

The Coyote single-board computer gives OEM designers extremely low-cost embedded control for high-volume applications. Two standard models—one with Ethernet, one without—feature the Rabbit® 3000 microprocessor running at 29.4 MHz, with standard 256K flash and 128K SRAM. These compact boards are rich with the I/O (including one A/D input and two D/A outputs) designers need for embedded control and monitoring applications, and the Coyote's compact board size of 3.95" × 3.95" (100 × 100 mm) is easily mountable in standard 100 mm DIN rail trays.

Customized BL2500 models can be manufactured in volume in OEM versions to user-specified configurations. Pin-compatible RabbitCore modules allow multiple configurations of the Coyote with Ethernet and memory options.

1.1 Features

- Rabbit 3000® microprocessor operating at 29.4 MHz (option for 44.2 MHz with 10/100Base-T Ethernet interface)
- 128K SRAM and 256K flash memory standard, optional 512K SRAM/512K flash
- 24 digital I/O: 9 protected and filtered digital inputs, 7 high-speed protected but unfiltered digital inputs, and 8 digital outputs sinking up to 200 mA at up to 36 V DC
- one 8-bit analog input channel
- two 9-bit PWM analog output channels
- six serial ports, including RabbitNet™ expansion ports
- one 10/100-compatible RJ-45 Ethernet port with standard 10Base-T interface (optional 10/100Base-T interface)
- 4 user-programmable LEDs.
- battery-backed real-time clock.
- watchdog supervisor.
- onboard backup battery for real-time clock and SRAM

Two BL2500 models are available. Their standard features are summarized in Table 1.

Table 1. BL2500 Models

Feature	BL2500	BL2510
Microprocessor	Rabbit 3000® running at 29.4 MHz	
Flash Memory	256K*	
Static RAM	128K*	
Ethernet Connections	Yes	No
RabbitCore Module Used	RCM3010	RCM3110
A/D Converter Input	Yes	Yes

* 512K options available

The BL2500 consists of a main board with a RabbitCore module. Refer to the RabbitCore module manuals, available on Rabbit's Web site, for more information on the RabbitCore modules, including their schematics.

Appendix A provides detailed specifications.

Visit our 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.

1.1.1 OEM Versions

The BL2500 and BL2510 models are also available in OEM versions as the OEM2500 and the OEM2510 (minimum quantity 500) where certain features have been removed or eliminated:

- fewer digital inputs—only 16 digital I/O, with 8 protected and filtered digital inputs and 8 digital outputs sinking up to 200 mA at up to 36 V DC (no header J12)
- no backup battery
- no RabbitNet™ hardware—no RS-422/multiplexer chips, no RabbitNet RJ-45 jacks, no RabbitNet™ power connectors (headers J7 and J8)

1.2 Development and Evaluation Tools

1.2.1 Development Kit

A Development Kit contains the hardware essentials you will need to use your BL2500/OEM2500. The items in the Development Kit and their use are as follows.

- BL2500 single-board computer.
- ***Getting Started*** instructions.
- ***Dynamic C*** CD-ROM, with complete product documentation on disk.
- Programming cable, used to connect your PC serial port to the BL2500.
- 12 V AC adapter, used to power the BL2500. An AC adapter is supplied with development kits sold in the North American market. If you are using your own power supply, it must provide 8 to 40 V DC.
- Demonstration Board with pushbutton switches and LEDs. The Demonstration Board can be hooked up to the BL2500 to demonstrate the I/O.
- Parts to build your own wire assemblies: wire, twenty-five 0.1" crimp terminals; ten 0.156" crimp terminals; 1 × 2, 1 × 4, and 1 × 10 friction-lock connectors.
- Nylon machine screws to serve as legs for the BL2500 board during development.
- ***Rabbit 3000 Processor Easy Reference*** poster.
- Registration card.

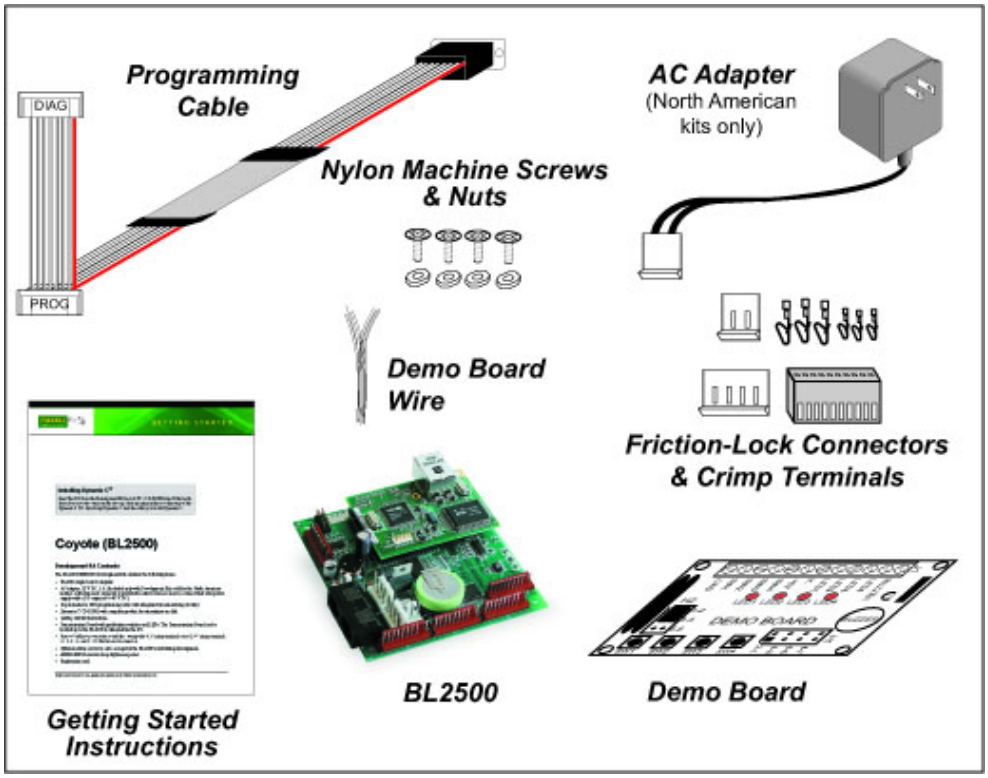


Figure 1. BL2500/OEM2500 Development Kit

1.2.2 Software

The Coyote is programmed using version 7.33 or later of Rabbit's Dynamic C. A compatible version is included on the Development Kit CD-ROM. Web-based technical support is included at no extra charge. Dynamic C v. 9.60 includes the popular µC/OS-II real-time operating system, point-to-point protocol (PPP), FAT file system, RabbitWeb, and other select libraries that were previously sold as individual Dynamic C modules.

Rabbit also offers for purchase 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.rabbit.com for further information and complete documentation, or contact your Rabbit sales representative or authorized distributor.

1.2.3 Connectivity Tools

Rabbit also has available additional tools and parts to allow you to make your own wiring assemblies in quantity to interface with the friction-lock connectors on the Coyote.

- Connectivity Kit (Part No. 101-0581)—Six 1 × 10 friction-lock connectors (0.1" pitch) with sixty 0.1" crimp terminals; and two 1 × 4 friction-lock connectors (0.156" pitch) and two 1 × 2 friction-lock connectors (0.156" pitch) with fifteen 0.156" crimp terminals. Each kit contains sufficient parts to interface with one Coyote board (some parts may be left over).
- Crimp tool (Part No. 998-0013) to secure wire in crimp terminals.

Table 3 in Chapter 3 provides information on specific friction-lock connectors and crimp terminals to be used with the various headers on the BL2500. Contact your authorized Rabbit distributor or your sales representative for more information.

1.2.4 DIN Rail Mounting

The Coyote may be mounted in 100 mm DIN rail trays as shown in Figure 2.

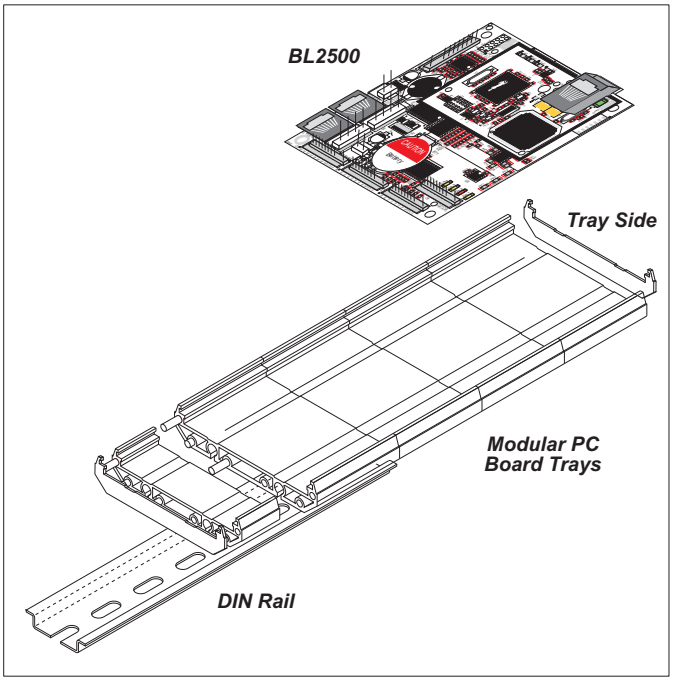


Figure 2. Mounting Coyote in DIN Rail Trays

DIN rail trays are typically mounted on DIN rails with “feet.” Table 2 lists Phoenix Contact part numbers for the DIN rail trays, rails, and feet. The tray side elements are used to keep the Coyote in place once it is inserted in a DIN rail tray, and the feet are used to mount the plastic tray on a DIN rail.

Table 2. Phoenix Contact DIN Rail Mounting Components

DIN Rail Mounting Component	Phoenix Contact Part Description	Phoenix Contact Part Number
Trays	UM 100-PROFIL cm*	19 59 87 4
Tray Side Elements	UM 108-SE	29 59 47 6
Foot Elements	UM 108-FE	29 59 46 3

* Length of DIN rail tray in cm

NOTE: Other major suppliers besides Phoenix Contact also offer DIN rail mounting hardware. Note that the width of the plastic tray should be 100 mm (3.95") since that is the width of the Coyote. 108 mm plastic trays may be used with spacers.

1.3 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. The Coyote has 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 D 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.

1.4 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 BL2500 single-board computer has been tested and was found to be in conformity with the following applicable immunity and emission standards. The BL2510 and OEM single-board computers are also CE qualified as they are sub-versions of the BL2500 single-board computer. Boards that are CE-compliant have the CE mark.

NOTE: Earlier versions of the BL2500 that do not have the CE mark are ***not*** CE-compliant.

Immunity

The BL2500 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 BL2500 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.

1.4.1 Design Guidelines

Note the following requirements for incorporating the BL2500 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 BL2500 single-board computer to outdoor cables, the customer is responsible for providing CE-approved surge/lightning 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 BL2500, it is the responsibility of the end-user to use proper ESD precautions to prevent ESD damage to the BL2500.

Safety

- All inputs and outputs to and from the BL2500 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 BL2500 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.

1.4.2 Interfacing the BL2500 to Other Devices

Since the BL2500 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.rabbit.com.

2. GETTING STARTED

Chapter 2 explains how to connect the programming cable and power supply to the BL2500.

2.1 Preparing the BL2500 for Development

Position the BL2500 as shown below in Figure 3. Attach the four nylon 4-40 × ¼ machine screws and nuts supplied with the Development Kit in the holes at the corners as shown.

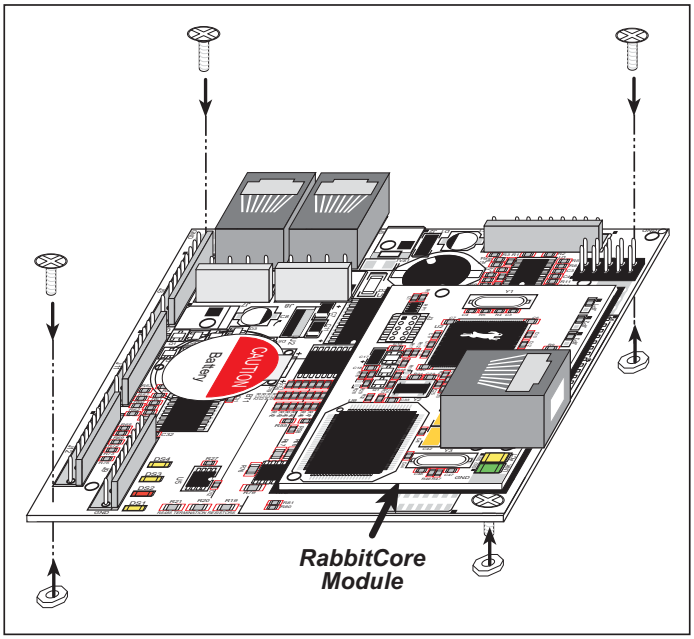


Figure 3. Attach Nylon Screws to BL2500 Board

NOTE: You will have to remove the RabbitCore module to install one screw under the module. When replacing the RabbitCore module, it is important that you line up the pins on the module exactly with the corresponding pins on the BL2500. The header pins may become bent or damaged if the pin alignment is offset, and the module will not work. Permanent electrical damage may also result if a misaligned module is powered up.

The nylon screws serve as standoffs to facilitate handling the BL2500 during development, and protect the bottom of the printed circuit board against scratches or short circuits while you are working with the BL2500.

2.2 BL2500 Connections

1. Connect the programming cable to download programs from your PC and to program and debug the BL2500.

   NOTE: Use only the programming cable that has a red shrink wrap around the RS-232 level converter (Part No. 20-101-0513). If you are using a BL2500 with the optional 10/100Base-T Ethernet interface, you will need the programming cable that has a blue shrink wrap around the RS-232 level converter (Part No. 20-101-0542). Other Rabbit programming cables might not be voltage-compatible or their connector sizes may be different.

Connect the 10-pin **PROG** connector of the programming cable to header J3 on the BL2500's RabbitCore module. Ensure that the colored edge lines up with pin 1 as shown. There is a small dot on the circuit board next to pin 1 of header J3. (Do not use the **DIAG** connector, which is used for monitoring only.) Connect the other end of the programming cable to a COM port on your PC. Make a note of the port to which you connect the cable, as Dynamic C will need to have this parameter configured. Note that COM1 on the PC is the default COM port used by Dynamic C.

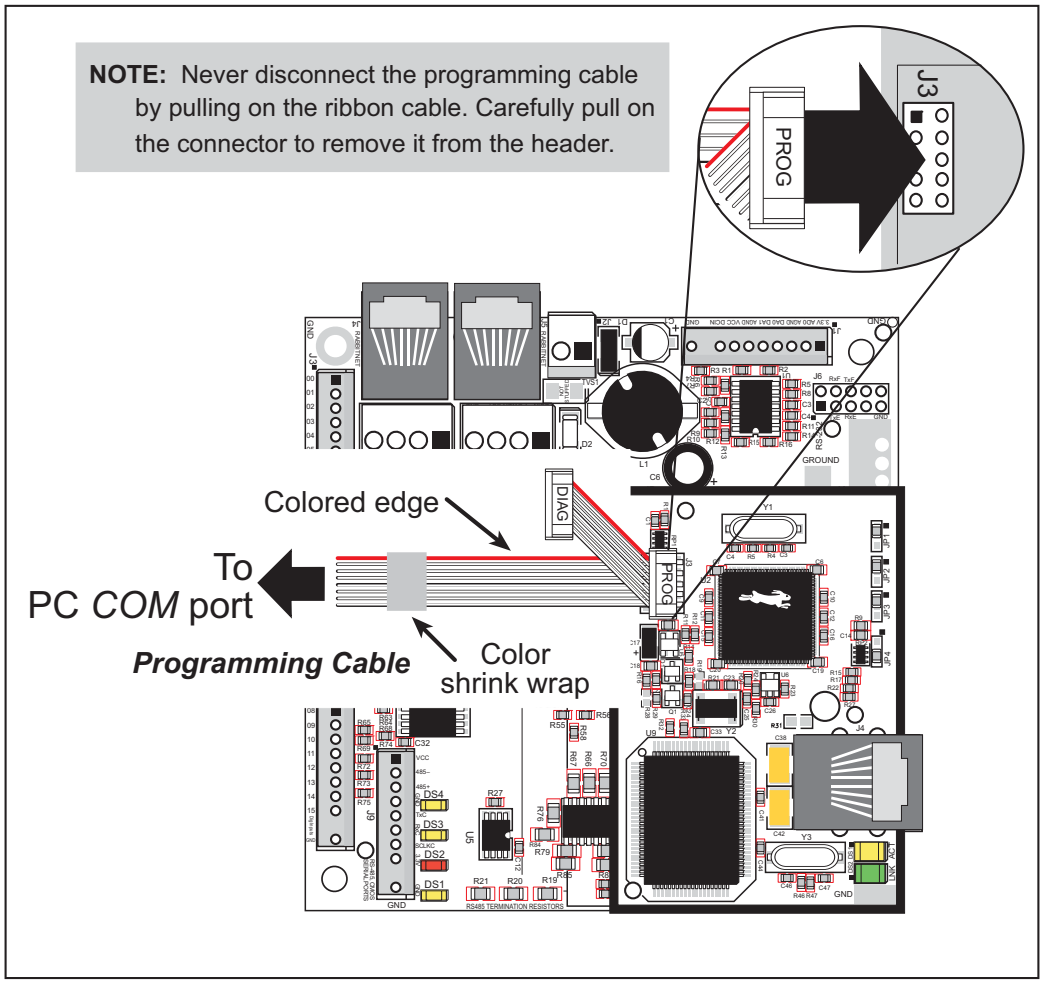


Figure 4. Programming Cable Connections

2. When all other connections have been made, you can connect power to the BL2500.

   Connect the AC adapter to header J2 on the BL2500 as shown in Figure 5. Match the friction lock tab on the friction-lock connector to the back of header J2 on the BL2500 as shown. The friction-lock connector will only fit one way.

   Development Kits sold outside North America include a friction lock friction-lock connector that may be connected to header J2 on the BL2500. Connect the leads from your power supply to the friction-lock connector to preserve the polarity indicated in Figure 5. The power supply should deliver 8 V–40 V DC at 500 mA.

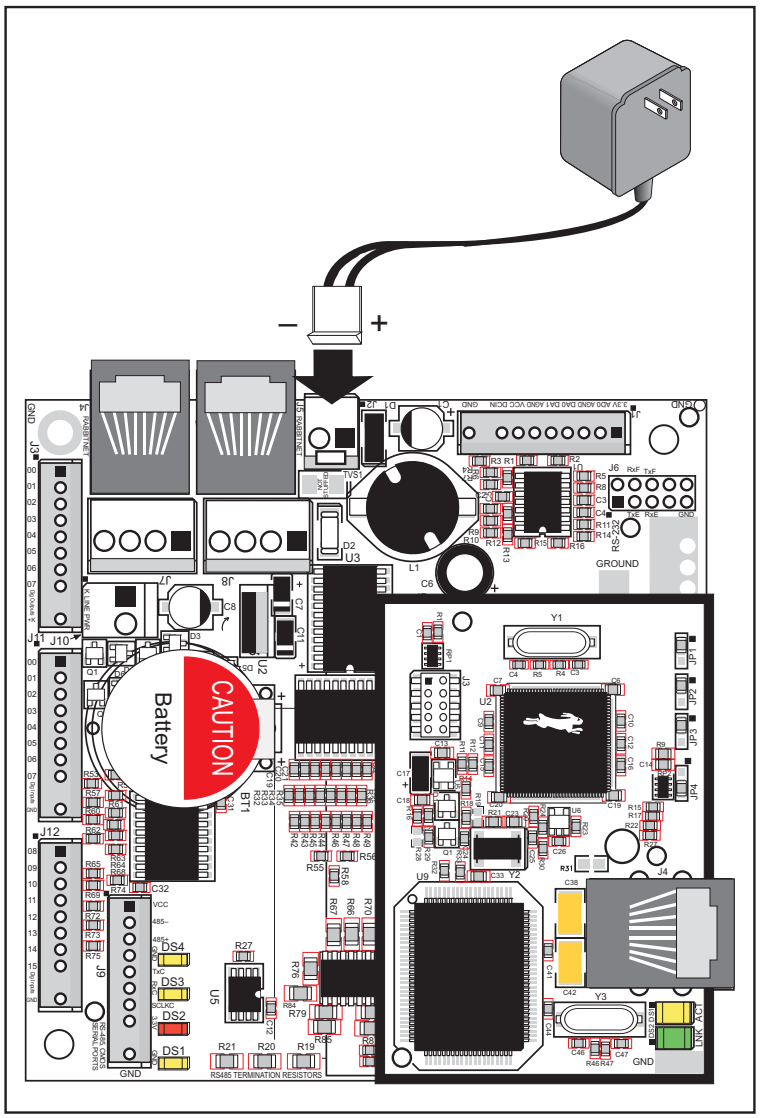


Figure 5. Power Supply Connections

3. Apply power.

Plug in the AC adapter.

CAUTION: Unplug the power supply while you make or otherwise work with the connections to the headers. This will protect your BL2500 from inadvertent shorts or power spikes.

2.2.1 Hardware Reset

A hardware reset is done by unplugging the AC adapter, then plugging it back in, or by shorting out the reset pads on the back of the BL2500 (see Figure 6).

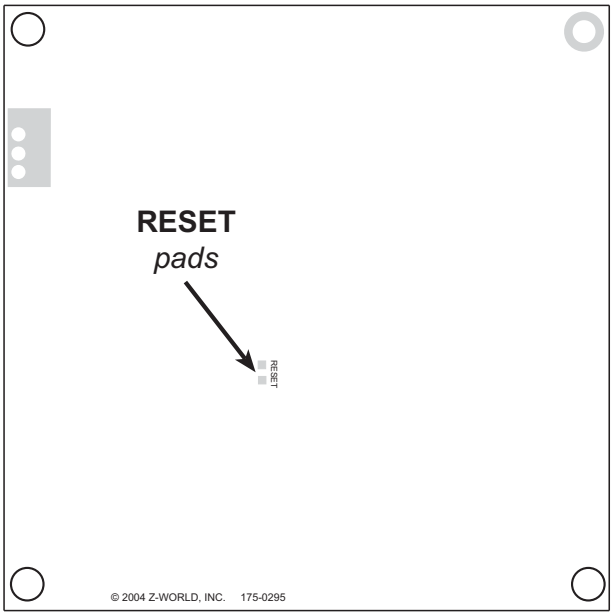


Figure 6. Location of RESET Pads

2.3 Installing Dynamic C

If you have not yet installed Dynamic C version 7.33 (or a later version), do so now by inserting the Dynamic C CD from the BL2500/OEM2500 Development Kit in your PC's CD-ROM drive. The CD will auto-install unless you have disabled auto-install on your PC.

If the CD does not auto-install, click **Start > Run** from the Windows **Start** button and browse for the Dynamic C **setup.exe** file on your CD drive. Click **OK** to begin the installation once you have selected the **setup.exe** file.

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.

> **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.

2.4 Starting Dynamic C

Once the BL2500 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.

Dynamic C defaults to using the serial port on your PC that you specified during installation. If the port setting is correct, Dynamic C should detect the BL2500 and go through a sequence of steps to cold-boot the BL2500 and to compile the BIOS. (Some versions of Dynamic C will not do the initial BIOS compile and load until the first time you compile a program.)

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 both ends of the programming cable to ensure that it is firmly plugged into the PC and the programming port.

If there are no faults with the hardware, select a different COM port within Dynamic C. From the **Options** menu, select **Communications**. Select another COM port from the list, then click OK. Press **<Ctrl-Y>** to force Dynamic C to recompile the BIOS. If Dynamic C still reports it is unable to locate the target system, repeat the above steps until you locate the active COM port. You should receive a **`Bios compiled successfully`** message once this step is completed successfully.

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 in the Dynamic C **Options > Communications** menu. Select a slower Max download baud rate.

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 in the Dynamic C **Options > Communications** menu. Choose a lower debug baud rate.

2.5 PONG.C

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

Find the file **PONG.C**, which is in the Dynamic C **SAMPLES** folder. To run the program, open it with the **File** menu (if it is not still open), compile it using the **Compile** menu, and then run it by selecting **Run** in the **Run** menu. The **STDIO** window will open on the 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.

2.6 Where Do I Go From Here?

> **NOTE:** If you purchased your BL2500 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.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 program ran fine, you are now ready to go on to explore other BL2500 features and develop your own applications.

Chapter 3, "Subsystems," provides a description of the BL2500's features, Chapter 4, "Software," describes the Dynamic C software libraries and introduces some sample programs. Chapter 5, "Using the TCP/IP Features," explains the TCP/IP features.

2.7 Using the Coyote In High-Vibration Environments

If you plan to use your Coyote in a high-vibration environment, the RabbitCore module may be secured more solidly to a swage on the Coyote main board using a 2-56 × ¼" machine screw as shown in Figure 7.

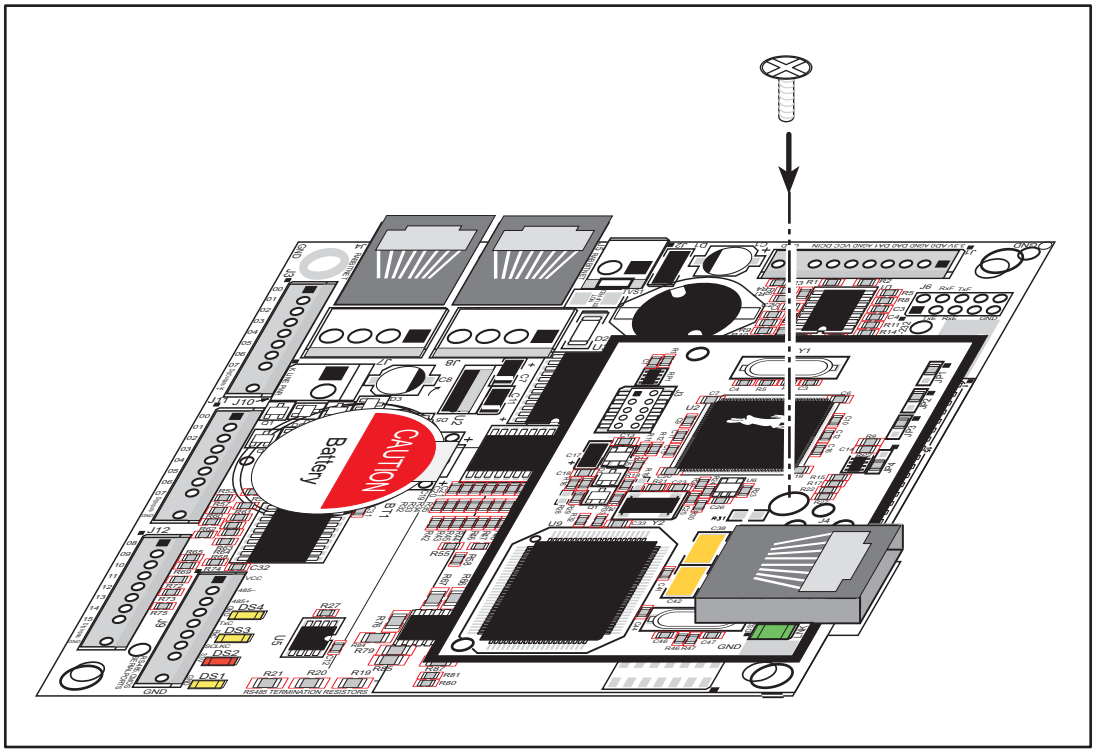


Figure 7. Secure RabbitCore Module to Coyote for High-Vibration Environments

3. SUBSYSTEMS

Chapter 3 describes the principal subsystems for the Coyote.

- Digital I/O
- Analog Features
- Serial Communication
- Memory

Figure 8 shows these Rabbit-based subsystems designed into the Coyote.

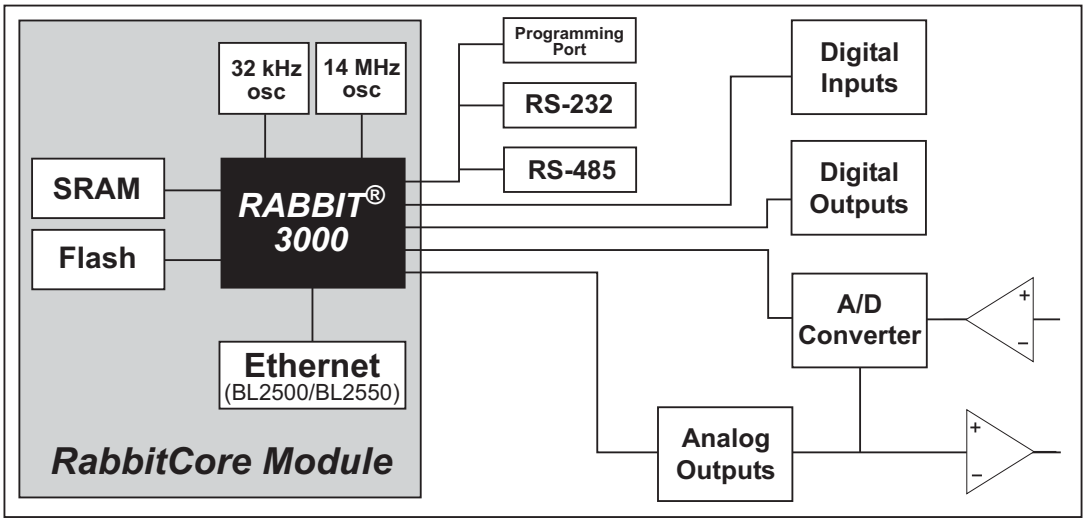


Figure 8. Coyote Subsystems

The memory and microprocessor are located on the RabbitCore module. If you have more than one Coyote or other Rabbit products built around RabbitCore modules, take care not to swap the RabbitCore modules since they contain system ID block information and calibration constants that are unique to the board they were originally installed on. It is a good idea to save the calibration constants should you need to replace a RabbitCore module in the future. See Section 4.2.6, "Using System Information from the RabbitCore Module," for more information.

3.1 Coyote Pinouts

The Coyote pinouts are shown in Figure 9.

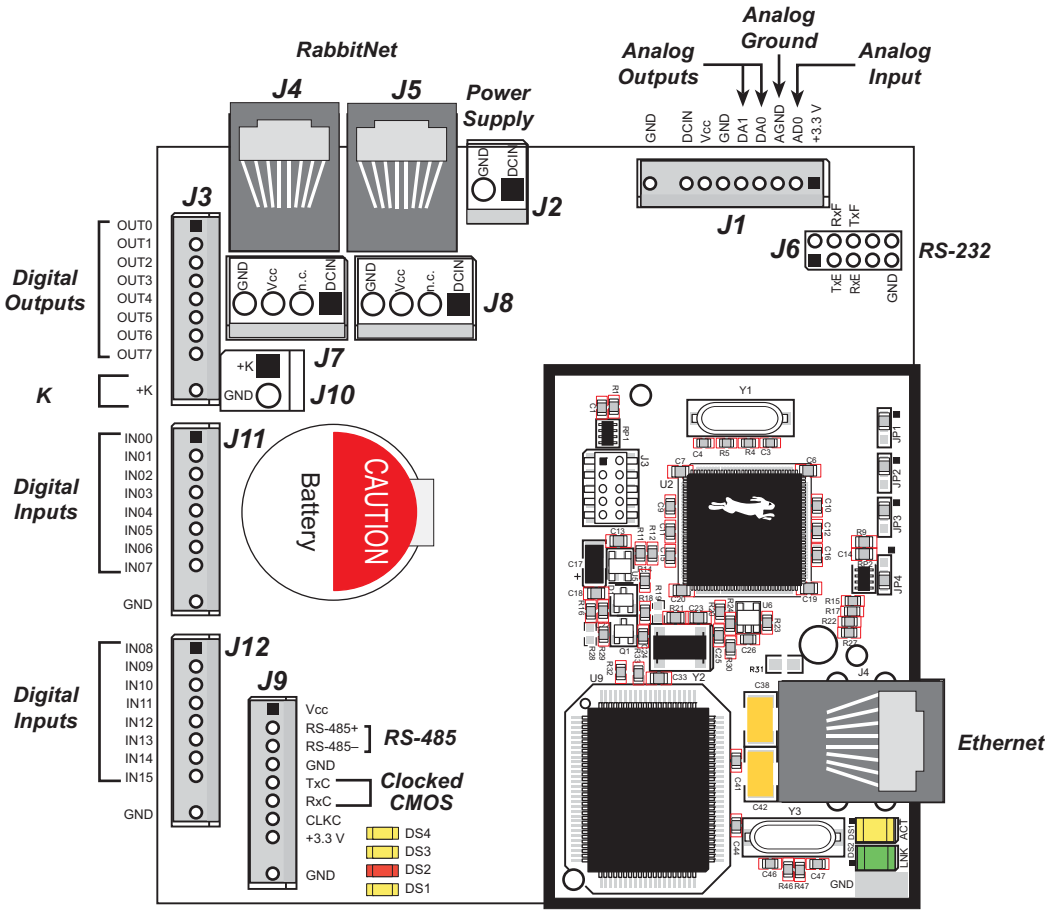


Figure 9. Coyote Pinouts

3.1.1 Headers

Standard Coyote models are equipped with five 1 × 10 friction-lock connector terminals (J1, J3, J9, J11, and J12) where pin 9 is removed to polarize the connector terminals, a 2 × 5 RS-232 signal header, a 2 × 5 programming header, and an RJ-45 Ethernet jack on the RabbitCore module.

The RJ-45 jacks at J4 and J5 labeled ***RabbitNet*** are serial I/O expansion ports for use with digital I/O and analog I/O boards currently being developed. The ***RabbitNet*** jacks do ***not*** support Ethernet connections. Be careful to connect your Ethernet cable to the jack labeled ***Ethernet***.

Two 4-pin 0.156" friction-lock connector terminals at J7 and J8 are installed to supply power (DCIN and +5 V) to the peripheral cards currently being developed for use with the RabbitNet. Two 2-pin 0.156" friction-lock connector terminals at J2 and J10 are for power supply and +K connections.

Table 3 lists Molex connector part numbers for the crimp terminals, housings, and polarizing keys needed to assemble female friction-lock connector assemblies for use with their male counterparts on the BL2500.

Table 3. Female Friction-Lock Connector Parts

Friction-Lock Connector	Used with BL2500 Headers	Molex Housing Part Number	Molex Crimp Terminals	Molex Polarizing Keys
0.1" 1 × 10	J1, J3, J9, J11, J12	22-01-2107	08-50-0113	15-04-9209
0.156" 1 × 4	J7, J8	09-50-3041	08-50-0108	15-04-0219
0.156" 1 × 2	J2, J10	09-50-3021		

3.2 Indicators

3.2.1 LEDs

The Coyote's RabbitCore module has two LEDs next to the RJ-45 Ethernet jack, one to indicate an Ethernet link (**LNK**) and one to indicate Ethernet activity (**ACT**).

User-programmable LEDs driven by the Rabbit 3000

- DS1—PB6 (yellow),
- DS2—PB7 (red),
- DS3—PA7 (yellow), and
- DS4—PA6 (yellow)

are also provided.

3.3 Digital I/O

3.3.1 Digital Inputs

The Coyote has 16 digital inputs, IN00–IN15. IN00–IN13 and IN15 are each protected over a range of –36 V to +36 V, and IN14 is protected over a range of –36 V to +5 V. The inputs are factory-configured to be pulled up to +3.3 V; IN00–IN07 can also be pulled up to +K or they can be pulled down to 0 V by changing a surface-mounted 0 Ω resistor. Figure 10 shows a sample digital input circuit. IN00-IN07 and IN15 are protected against noise spikes by a low-pass filter composed of a 22 kΩ series resistor and a 10 nF capacitor.

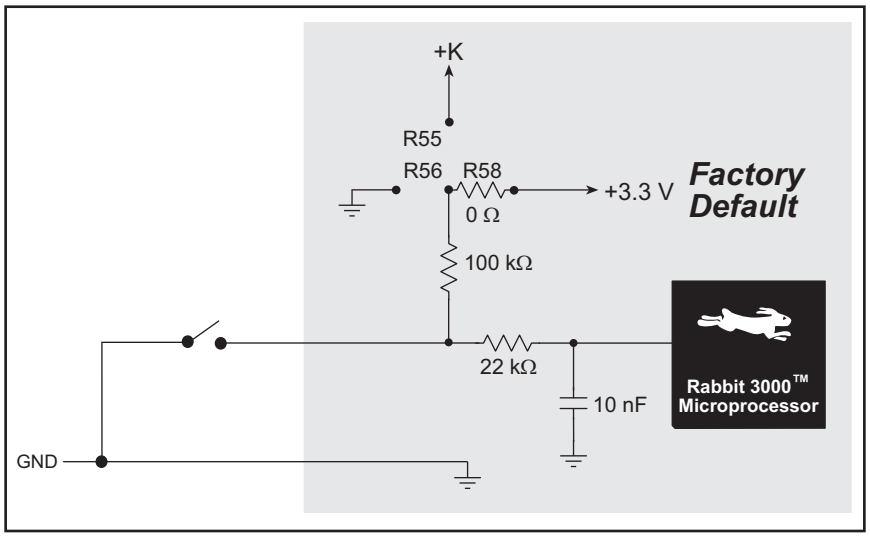


Figure 10. Coyote Digital Inputs [Pulled Up—Factory Default]

Coyote series boards can be made to order in volume with the digital inputs pulled up to +K or pulled down to 0 V. Contact your authorized Rabbit distributor or your sales representative at for more information.

The actual switching threshold between a zero and a one is between 0.9 V and 2.3 V for all 16 inputs.

IN00–IN13 and IN15 are each fully protected over a range of -36 V to +36 V, and can handle short spikes of ±40 V. IN14 is protected over a range of -36 V to +5 V.

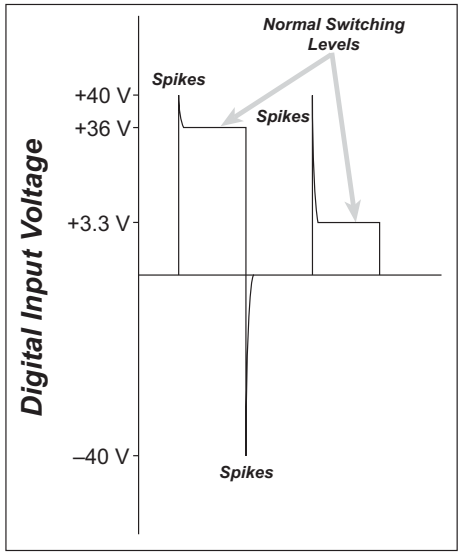


Figure 11. Coyote Digital Input Protected Range

3.3.2 Digital Outputs

The Coyote has eight digital outputs, OUT0–OUT7, each of which can sink up to 200 mA. Figure 12 shows a wiring diagram for using the digital outputs in a sinking configuration.

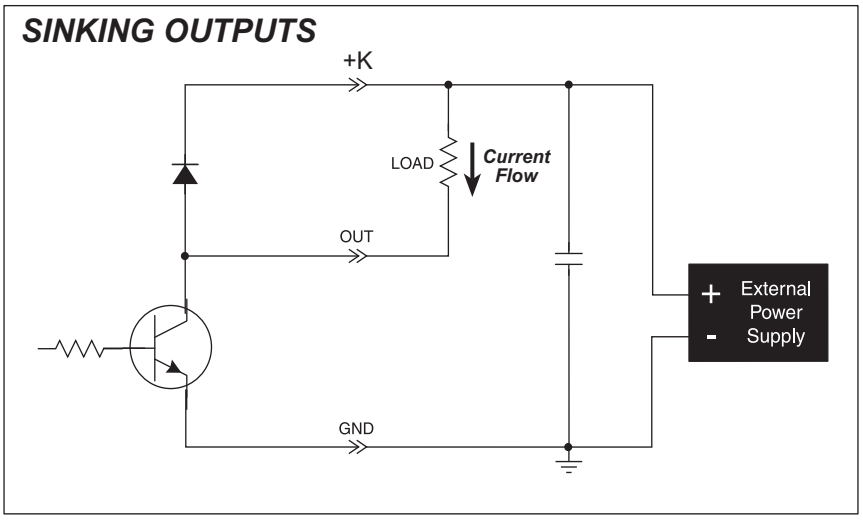


Figure 12. Coyote Digital Outputs

+K is an externally supplied voltage of 3.3–40 V DC, and should be capable of delivering all the load currents. Although a connection to a +K supply is not absolutely ***required*** with sinking outputs, it is ***highly recommended*** to protect against current spikes when driving inductive loads such as relays and solenoids.

Connect the positive +K supply to pin 1 of friction-lock connector terminal J10 and the negative side of the supply to pin 2 of friction-lock connector terminal J10. A friction-lock connector is recommended to connect this supply because the +K inputs are ***not*** protected against reverse polarity, and serious damage to the Coyote may result if you connect this supply backwards.

3.4 Analog Features

3.4.1 A/D Converter

The A/D converter, shown in Figure 13, compares the DA0 voltage to AD0, the voltage presented to the A/D converter. DA0 therefore cannot be used for the D/A converter when the A/D converter is being used.

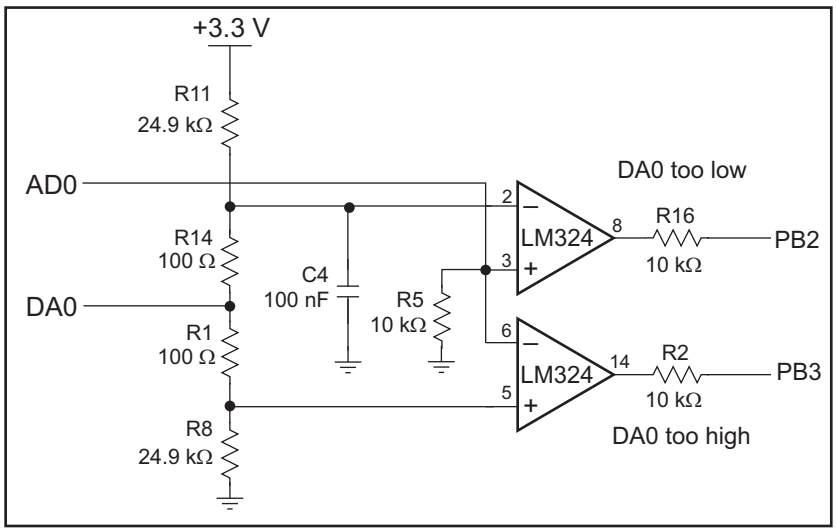


Figure 13. Schematic Diagram of A/D Converter

The A/D converter programs DA0 using a successive-approximation binary search until DA0 equals the A/D converter input voltage. That programmed DA0 voltage is then reported as the A/D converter value.

The A/D converter transforms the voltage at DA0 into a 13.2 mV window around DA0. Because the A/D converter circuit uses a 13.2 mV window, the accuracy is ±6.6 mV. DA0 can range from 0.1 V to 3.1 V, which represents 227 steps of 13.2 mV. This represents an accuracy of approximately 8 bits. Since the D/A converter is able to change the DA0 output in 3.22 mV steps, there are 930 steps over the range from 0.1 V to 3.1 V. This represents a resolution of more than 9 bits.

For example, if DA0 is 1.650 V, the window in the A/D converter would be 1.643 V to 1.657 V. If AD0 > 1.657 V, PB2 would read high and PB3 would read low. If 1.643 V < AD0 < 1.657 V, PB2 would read low and PB3 would read low. This is the case when the A/D input is exactly the same as DA0. If AD0 < 1.643 V, PB2 would read low and PB3 would read high. The A/D converter input, AD0, is the same as DA0 only when both PB2 and PB3 are low.

PB3 can be imagined to be a “DA0 voltage is too high” indicator. If DA0 is larger than the analog voltage presented at AD0, then PB3 will be true (high). If this happens, the program will need to reduce the DA0 voltage.

PB2 can be imagined to be a “DA0 voltage is too low” indicator. If DA0 is smaller than the analog voltage presented at AD0, then PB2 will be true (high). If this happens, the program will need to raise the DA0 voltage.

The A/D converter has no reference voltage. There is a relative accuracy between measurements, but no absolute accuracy without calibration. This is because the +3.3 V supply can vary ±5%, the pulse-width modulated outputs might not reach the full 0 V and 3.3 V rails out of the Rabbit 3000 microprocessor, and the gain resistors used in the circuit have a 1% tolerance. For these reasons, each Coyote needs to be calibrated individually, with the constants held in software, to be able to rely on an absolute accuracy. The Coyote has this calibration support.

An A/D conversion takes less than 100 ms with a 29.4 MHz Coyote.

3.4.2 D/A Converters

Two D/A converter outputs, DA0 and DA1, are supplied on the Coyote. These are shown in Figure 14.

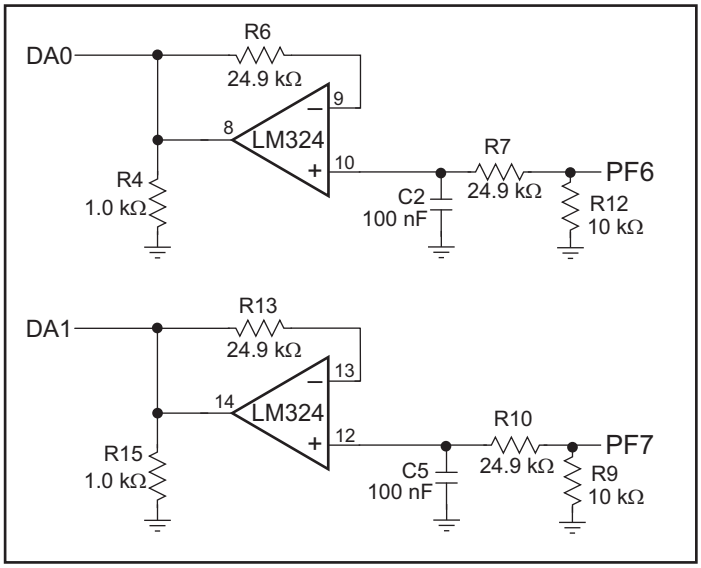


Figure 14. Schematic Diagram of D/A Converters

The D/A converters have no reference voltage. Although they may be fairly accurate from one programmed voltage to the next, they do not have absolute accuracy. This is because the +3.3 V supply can change ±5%, the PWM outputs might not achieve the full 0 V and 3.3 V rail out of the processor, and the gain resistors in the circuit have a 1% tolerance. The D/A converters therefore need individual calibration, with the calibration constants held in software before absolute accuracy can be relied on. The Coyote has such calibration.

Note that DA0 is used to provide a reference voltage for the A/D converter and is unavailable for D/A conversion when the A/D converter is being used.

Pulse-width modulation (PWM) is used for the D/A conversion. The digital signal, which is either 0 V or 3.3 V, will be a train of pulses. This means that if the signal is taken to be usually at 0 V (or ground), the pulses will be some 3.3 V pulses of varying width. The voltage will be 0 V for a given time, then jump to 3.3 V for a given time, then back to ground for a given time, then back to 3.3 V, and so on. A hardware filter that consists of a resistor and capacitor averages the 3.3 V signal and the 0 V signal over time. Therefore, if the time that the signal is at 3.3 V is equal to the time the signal is 0 V, the duty cycle will be 50%, and the average signal will be 1.65 V. If the time at 3.3 V is only 25% of the time, then the average voltage will be 0.825 V. Thus, the software needs to only vary the time the signal is at 3.3 V with respect to the time the signal is at 0 V to achieve any desired voltage between 0 and 3.3 V. It is very easy to do pulse-width modulation with the Rabbit 3000 microprocessor because the chip's architecture includes an advanced PWM feature.

3.4.2.1 DA0 and DA1

The RC networks supporting DA0 and DA1 converts pulse-width modulated signals to an analog voltage between 0 V and 3.3 V. A digital signal that varies with time is fed from PF6 or PF7. The resolution of the DA0 or DA1 output depends on the smallest increment of time to change the on/off time (the time between 3.3 V and 0 V). The Coyote uses the Rabbit 3000's Port F control registers to clock out the signal at a timer timeout. The dedicated PWM hardware has 10 bits of resolution, and so that the voltage can be varied in 1/1024 increments. The resolution is thus about 3 mV (3.3 V/1024).

R6 and R13 are present solely to balance the op amp input current bias. R4 and R15 help to achieve a voltage close to ground for a 0% duty cycle.

A design constraint dictates how fast the PWM hardware must run. The hardware filter has a resistor-capacitor filter that averages the 0 V and 3.3 V values. Its effect is to smooth out the digital pulse train. It cannot be perfect, and so there will be some ripple in the output voltage. The maximum signal decay between pulses will occur when DA1 is set to 1.65 V. This means the pulse train will have a 50% duty cycle. The maximum signal decay will be

$$1.65\ \text{V} \times \left[1 - e^{\left(\frac{-t}{RC}\right)} \right]$$

where RC = 2.5 ms, and t is the pulse on or off time (not the length of the total cycle).

The PWM hardware is driven at the Rabbit 3000 frequency divided by 2. The frequency achievable with a 29.4 MHz clock is (29.4 MHz/2)/1024 = 14.3 kHz. Since the Rabbit 3000 PWM spreader enhances the frequency fourfold, the effective frequency becomes 57.4 kHz. This is a period of 1/f = 17.4 µs. For a 50% duty cycle, half of the period will be high (8.7 µs at 3.3 V), and half will be low (8.7 µs at 0 V). Thus, a 29.4 MHz Coyote has t = 8.7 µs.

Based on the standard capacitor discharge formula, this means that the maximum voltage change will be

$$1.65\ \text{V} \times \left[1 - e^{\left(\frac{-8.7\ \mu\text{s}}{2.5\ \text{ms}}\right)}\right] = 5.73\ \text{mV}$$

This is a ripple of approximately 6 mV peak-to-peak.

Table 4 lists typical uncalibrated DA0 or DA1 voltages measured for various duty cycle values with a load larger than 1 MΩ.

Table 4. Typical Uncalibrated DA0 or DA1 Voltages for Various Duty Cycles

Duty Cycle (%)	Voltage (V)	Programmed Count
0	0.086	1
50	1.628	512
100	3.244	1023

The full D/A converter voltage range of 0–3.3 V cannot be realized because of the voltage tolerances associated with the voltage regulator, the Rabbit 3000 PWM output, and the op-amp rail. The circuit can achieve an actual voltage range of 0.1–3.3 V.

It is important to remember that the DA0 or DA1 output voltage will not be realized instantaneously after programming in a value. There is a settling time because of the RC time constant (24.9 kΩ × 100 nF), which is 2.5 ms. For example, the voltage at any given time is

$$V = V_P - (V_P - V_{DA})e^{(-t/RC)}$$

where V is the voltage at time t, V_P is the programmed voltage, V_{DA} is the last DA0 or DA1 output voltage from the D/A converter, and RC is the time constant (2.5 ms). The settling will be within 99.326% (or within about 22 mV for a 3.3 V change in voltage) after five time constants, or 12.5 ms. Six time constants, 15 ms, will allow settling to within 99.75% (or to within about 8 mV for a 3.3 V change in voltage). Seven time constants, 17.5 ms, will allow settling to within 99.91% (or to within about 3 mV for a 3.3 V change in voltage).

An LM324 op amp, which can comfortably source 10 mA throughout the D/A converter range, drives the D/A converter output. If the output voltage is above 1 V, the D/A converter can comfortably sink 10 mA. Below 1 V, the D/A converter can only sink a maximum of 100 µA.

To summarize, DA0 and DA1 are factory-calibrated, with the calibration constants stored in flash memory. DA0 and DA1 can be programmed with a resolution of 3 mV and a peak-to-peak ripple of 6 mV over the range from 0.1 to 3.1 V. The settling time to within 3 mV is 17.5 ms.

3.5 Serial Communication

The Coyote has two RS-232 serial ports, which can be configured as one RS-232 serial channel (with RTS/CTS) or as two RS-232 (3-wire) channels. The Coyote also has one RS-485 serial channel, one clocked CMOS serial channel, and two SPI serial ports with RS-422. There is also a CMOS serial channel that serves as the programming/debug port.

Table 5. Coyote Serial Port Configuration

Serial Port	Use	Header
A	Programming Port	J3 (RabbitCore module)
B	RabbitNet SPI (RS-422)	J4/J5
C	Clocked CMOS	J9
D	RS-485	J9
E	RS-232	J6
F	RS-232	J6

The RS-232 and RS-485 serial ports operate in an asynchronous mode up to the baud rate of the system clock divided by 8. 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. The CMOS serial channel and the two RS-422 SPI ports can also be operated 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 for the clocked CMOS channel. As the master, the Coyote must supply the clock for the SPI ports.

The Coyote boards use all six serial ports. Serial Port A is used in the clocked serial mode to provide cold-boot, download, and emulation functions. Serial Port B is multiplexed between the two SPI RS-422 RabbitNet ports, SPI_1 and SPI_2. Clocked Serial Port C is available as a basic CMOS voltage-level serial port. Serial Port D is used for RS-485 communication, and Serial Ports E and F are used for RS-232 communication.

3.5.1 RS-232

The Coyote 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 3000's CMOS/TTL 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 -6 V and 0 V is output as +6 V. The RS-232 transceiver also provides the proper line loading for reliable communication.

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

RS-232 flow control on an RS-232 port is initiated in software using the **serXflowcontrolOn** function call from **RS232.LIB**, where **X** is the serial port (E or F). The locations of the flow control lines are specified using a set of five macros.

SERX_RTS_PORT—Data register for the parallel port that the RTS line is on (e.g., PGDR).

SERX_RTS_SHADOW—Shadow register for the RTS line's parallel port (e.g., PGDRShadow).

SERX_RTS_BIT—The bit number for the RTS line.

SERX_CTS_PORT—Data register for the parallel port that the CTS line is on (e.g., PCDRShadow).

SERX_CTS_BIT—The bit number for the CTS line.

Standard 3-wire RS-232 communication using Serial Ports E and F is illustrated in the following sample code.

```
#define EINBUFSIZE  15
#define EOUTBUFSIZE 15

#define FINBUFSIZE  15
#define FOUTBUFSIZE 15

#ifndef _232BAUD
#define _232BAUD 115200
#endif

main(){
    serEopen(_232BAUD);
    serFopen(_232BAUD);
    serEwrFlush();
    serErdFlush();
    serFwrFlush();
    serFrdFlush();
}
```

3.5.2 RS-485

The Coyote has one RS-485 serial channel, which is connected to the Rabbit 3000 Serial Port D through an RS-485 transceiver. The half-duplex communication uses PA4 to control the transmit enable on the communication line. Using this scheme a strict master/slave relationship must exist between devices to insure that no two devices attempt to drive the bus simultaneously.

Serial Port D is configured in software for RS-485 as follows.

```
#define ser485open serDopen
#define ser485close serDclose
#define ser485wrFlush serDwrFlush
#define ser485rdFlush serDrdFlush
#define ser485putc serDputc
#define ser485getc serDgetc

#define DINBUFSIZE  15
#define DOUTBUFSIZE 15

#ifndef _485BAUD
#define _485BAUD 115200
#endif
```

The configuration shown above is based on circular buffers. RS-485 configuration may also be done using functions from the **PACKET.LIB** library.

The Coyote can be used in an RS-485 multidrop network spanning up to 1200 m (4000 ft), and there can be as many as 32 attached devices. Connect the 485+ to 485+ and 485– to 485– using single twisted-pair wires as shown in Figure 15. Note that a common ground is recommended.

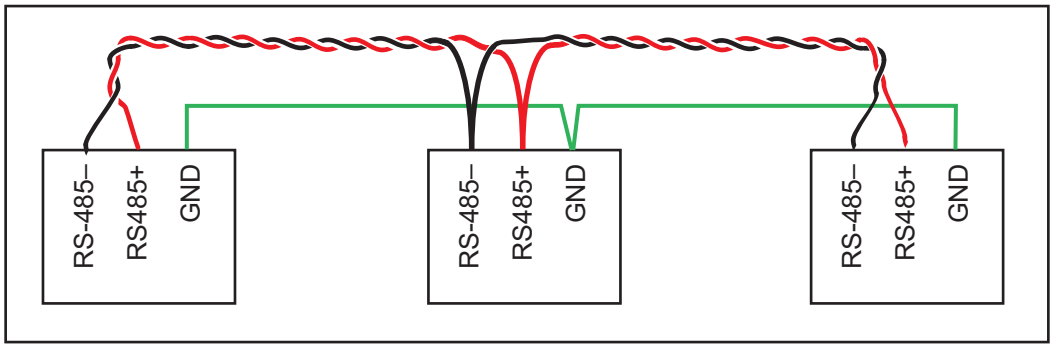


Figure 15. Coyote Multidrop Network

The Coyote comes with a 220 Ω termination resistor and two 681 Ω bias resistors installed and enabled.

The load these bias and termination resistors present to the RS-485 transceiver limits the number of Coyotes in a multidrop network to one master and nine slaves, unless the bias and termination resistors are removed. When using more than 10 Coyotes in a multidrop network, or when you need the full common-mode immunity per the RS-485 specification, leave the 681 Ω bias resistors in place on the master Coyote, and leave the 220 Ω termination resistors in place on the Coyote at each end of the network.

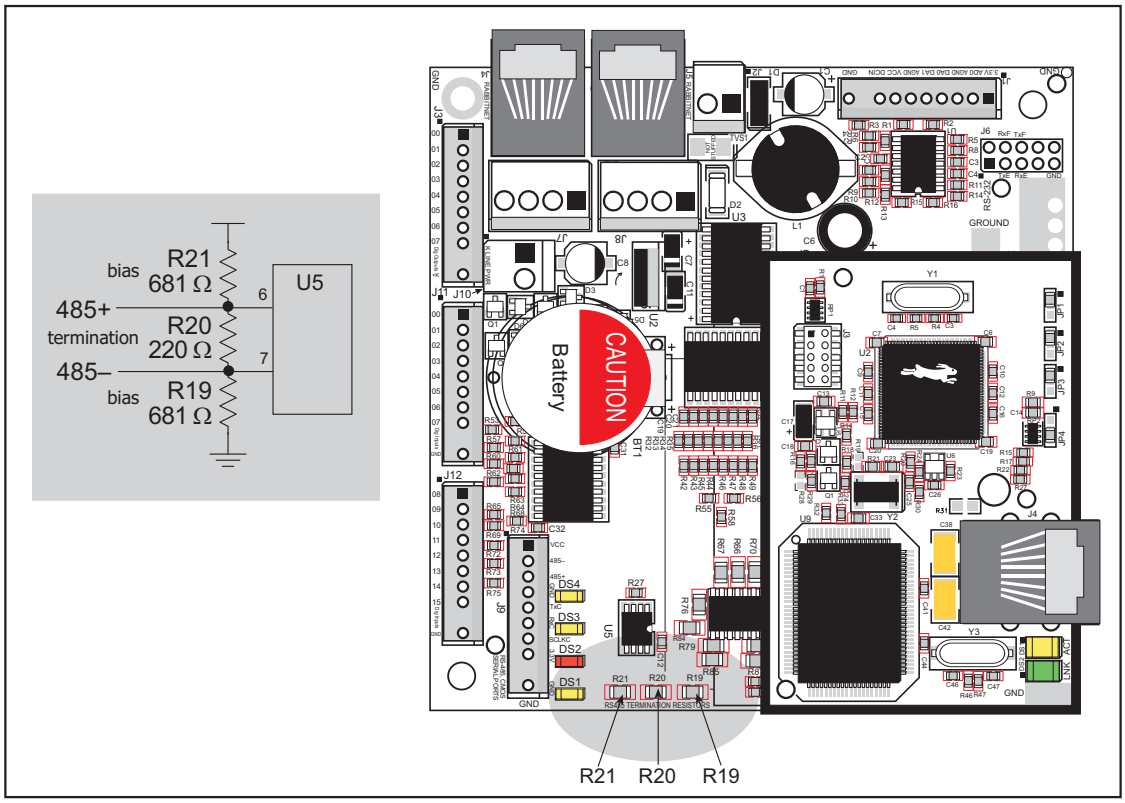


Figure 16. RS-485 Termination and Bias Resistors

3.5.3 Programming Port

The Coyote's serial programming port is accessed via the 10-pin programming header on the RabbitCore module or over an Ethernet connection via the RabbitLink EG2110. The programming port uses the Rabbit 3000's Serial Port A for communication. Dynamic C uses the programming port to download and debug programs.

The programming port is also used for the following operations.

- Cold-boot the Rabbit 3000 on the RabbitCore module after a reset.
- Remotely download and debug a program over an Ethernet connection using the RabbitLink EG2110.
- Fast copy designated portions of flash memory from one Rabbit-based board (the master) to another (the slave) using the Rabbit Cloning Board.

Alternate Uses of the Programming Port

All three clocked Serial Port A signals are available as

- a synchronous serial port
- an asynchronous serial port, with the clock line usable as a general CMOS I/O pin

The programming port may also be used as a serial port via the **DIAG** connector on the programming cable.

In addition to Serial Port A, the Rabbit 3000 startup-mode (SMODE0, SMODE1), status, and reset pins are available on the programming port.

The two startup mode pins determine what happens after a reset—the Rabbit 3000 is either cold-booted or the program begins executing at address 0x0000.

The status pin is used by Dynamic C to determine whether a Rabbit microprocessor is present. The status output has three different programmable functions:

1. It can be driven low on the first op code fetch cycle.
2. It can be driven low during an interrupt acknowledge cycle.
3. It can also serve as a general-purpose output.

The /RESET_IN pin is an external input that is used to reset the Rabbit 3000 and the RCM3400 onboard peripheral circuits. The serial programming port can be used to force a hard reset on the RCM3400 by asserting the /RESET_IN signal.

Refer to the ***Rabbit 3000 Microprocessor User's Manual*** for more information.

3.5.4 RabbitNet Ports

The RJ-45 jacks labeled ***RabbitNet*** are multiplexed clocked SPI RS-422 serial I/O expansion ports for use with peripheral cards currently being developed. The ***RabbitNet*** jack does ***not*** support Ethernet connections.

3.5.5 Ethernet Port

Figure 17 shows the pinout for the RJ-45 Ethernet port (header J4 on the RabbitCore module). Note that some Ethernet connectors are numbered in reverse to the order used here.

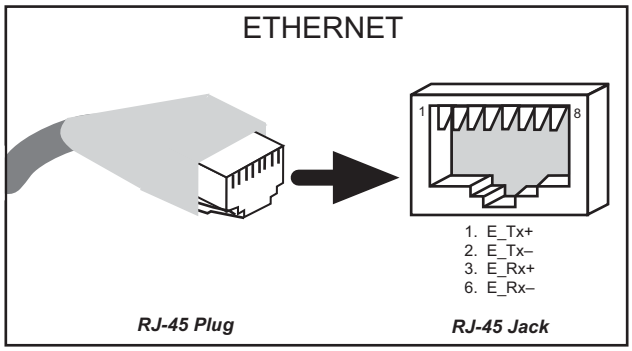


Figure 17. RJ-45 Ethernet Port Pinout

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

The transformer/connector assembly ground is connected to the RabbitCore module printed circuit board digital ground via a 0 Ω resistor, R31, as shown in Figure 18.

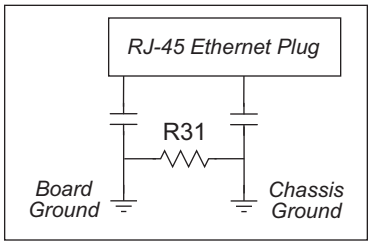


Figure 18. Isolation Resistor R31 on RabbitCore Module

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

3.6 Serial Programming Cable

The programming cable is used to connect the serial programming port of the Coyote to a PC serial COM port. The programming cable converts the RS-232 voltage levels used by the PC serial port to the CMOS voltage levels used by the Rabbit 3000.

When the **PROG** connector on the programming cable is connected to the programming header on the Coyote'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 Coyote's RabbitCore module with the Coyote operating in the Run Mode. This allows the programming port to be used as a regular serial port.

3.6.1 Changing Between Program Mode and Run Mode

The Coyote 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 3000 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 3000 in the Program Mode. When the programming cable's **PROG** connector is not attached, the SMODE pins are pulled low, causing the Rabbit 3000 to operate in the Run Mode.

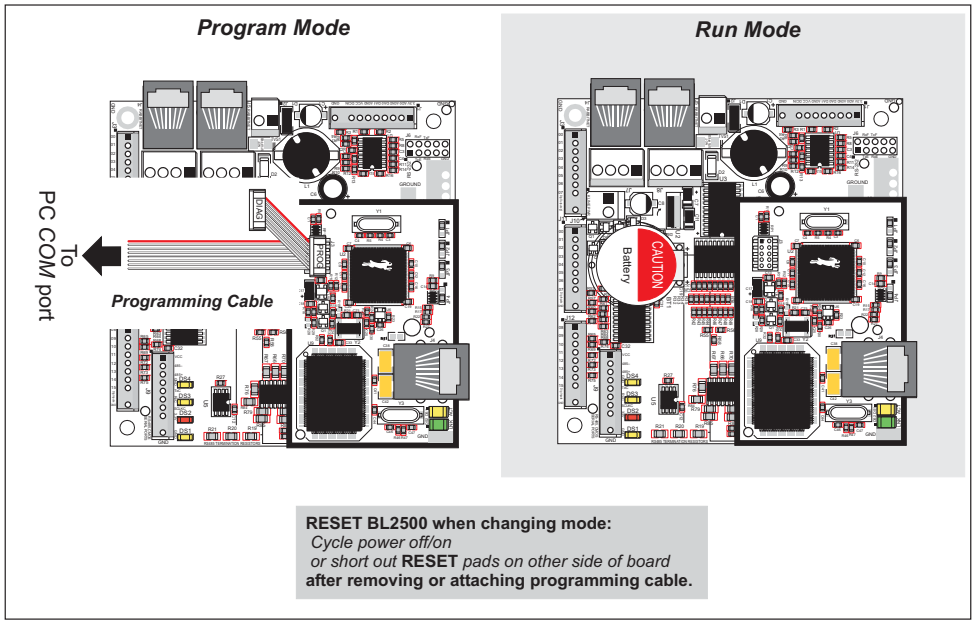


Figure 19. Coyote Program Mode and Run Mode Setup

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

Refer to the ***Rabbit 3000 Microprocessor User's Manual*** for more information on the programming port and the programming cable.

3.7 Other Hardware

3.7.1 Clock Doubler

The Coyote takes advantage of the Rabbit 3000 microprocessor's internal clock doubler. A built-in clock doubler allows half-frequency crystals to be used to reduce radiated emissions. The 29.4 MHz frequency specified for the Coyote is generated using a 14.7456 MHz crystal. The clock doubler will not work for crystals with a frequency above 26.7264 MHz.

The clock doubler may be disabled if 29.4 MHz clock speeds are not required. Disabling the Rabbit 3000 microprocessor's internal 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: Disabling the clock doubler will degrade the A/D and D/A conversion.

3.7.2 Spectrum Spreader

The Rabbit 3000 features a spectrum spreader, which helps 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 now remain off whenever you are in the project file where you defined the macro.

NOTE: Refer to the ***Rabbit 3000 Microprocessor User's Manual*** for more information on the spectrum-spreading setting and the maximum clock speed.

3.8 Memory

3.8.1 SRAM

The Coyote's RabbitCore module is designed to accept 128K to 512K of SRAM packaged in an SOIC case. The standard Coyote's RabbitCore modules come with 128K of SRAM.

3.8.2 Flash Memory

The Coyote is also designed to accept 128K to 512K of flash memory. The standard Coyote's RabbitCore modules comes with one 256K 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, use a portion of the "user block" area to store persistent data. The functions **`writeUserBlock`** and **`readUserBlock`** are provided for this. Refer to the ***Rabbit 3000 Microprocessor Designer's Handbook*** for additional information.

A Flash Memory Bank Select jumper configuration option based on 0 Ω surface-mounted resistors exists at header JP2 on the RabbitCore module. This option, used in conjunction with some configuration macros, allows Dynamic C to compile two different co-resident programs for the upper and lower halves of the 256K flash in such a way that both programs start at logical address 0000. This is useful for applications that require a resident download manager and a separate downloaded program. See Technical Note 218, ***Implementing a Serial Download Manager for a 256K Flash***, for details.

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 Coyote.

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 Coyote. 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: 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 Coyote 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 95, 98, 2000, NT, Me, and XP. 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, encryption, 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 Bug Fixes

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

The default installation of a patch or bug fix 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 without overwriting 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 a bug fix. Once you are sure the new patch works entirely to your satisfaction, you may retire the existing installation, but keep it available to handle legacy applications.

4.1.1.2 Upgrades

Dynamic C installations are designed for use with the board they are included with, and are included at no charge as part of our low-cost kits. Dynamic C is a complete software development system, but does not include all the Dynamic C features. Rabbit also offers add-on Dynamic C modules containing the popular μC/OS-II real-time operating system, as well as PPP, Advanced Encryption Standard (AES), and other select libraries. In addition to the Web-based technical support included at no extra charge, a one-year telephone-based technical support module is also available for purchase.

4.1.2 Accessing and Downloading Dynamic C Libraries

The libraries needed to run the Coyote are available on the CD included with the Development Kit, or they may be downloaded from http://www.rabbit.com/support/downloads/ on Rabbit's Web site. You may need to download upgraded or additional libraries to run selected RabbitNet peripheral cards.

When downloading the libraries from the Web site, click on the product-specific links until you reach the links for the BL2500 download. Once you have downloaded the self-extracting ZIP file, the following instructions will help you to add the libraries and sample programs to your existing Dynamic C installation.

1. Double-click on the file name of the self-extracting ZIP file.
2. The extracting program will prompt you for a folder in which to place the files.
3. Enter the drive letter and the name of the Dynamic C folder where the libraries and samples are to be added, for example, **C:\DCRabbit801**.
4. Click the **UnZip** button.

The files from the ZIP directory will then load automatically in your Dynamic C folder. Additional folders will be created as needed, and the **LIB.DIR**, **DEFAULT.H**, and **BOARDTYPES.LIB** files will be overwritten with the information needed to use the Coyote.

You will be able to use the revamped Dynamic C installation with the Coyote and you will continue to be able to use this installation with all the other Rabbit products you were able to use before.

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\BL2500** folder provides sample programs specific to the Coyote. 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), compile it using the **Compile** menu, and then run it by selecting **Run** in the **Run** menu. The Coyote must be in **Program** mode (see Section 3.6, "Serial Programming Cable") and must be connected to a PC using the programming cable as described in Section 2.2, "BL2500 Connections."

More complete information on Dynamic C is provided in the ***Dynamic C User's Manual***.

4.2.1 General Coyote Operation

The following sample programs are found in the **SAMPLES\BL2500**.

- **CONTROLLED.C**—Uses the D/A converters to vary the brightness of the LEDs on the Demonstration Board.
- **FLASHLEDS.C**—Uses cofunctions and costatements to flash LEDs on the Coyote at different intervals.
- **TOGGLESWITCH.C**—Uses costatements to detect switches presses on the Demonstration Board with press and release debouncing. Corresponding LEDs will turn on or off.

4.2.2 Digital I/O

The following sample programs are found in the **IO** subdirectory in **SAMPLES\BL2500**.

- **DIGIN.C**—This program demonstrates the use of the digital inputs and the function call **digIn()** using the Demonstration Board to see an input channel toggle from HIGH to LOW when pressing a pushbutton on the Demonstration Board.
- **DIGOUT.C**—This program demonstrates the use of the digital outputs and the function call **digOut()** using the Demonstration Board to see the logic levels of output channels in the **STDIO** window and the state of the corresponding LEDs on the Demonstration Board.

4.2.3 Serial Communication

The following sample programs are found in the **SERIAL** subdirectory in **SAMPLES\BL2500**.

- **FLOWCONTROL.C**—Demonstrates hardware flow control by sending a pattern of * characters out of Serial Port E (PG6) at115,200 bps. One character at a time is received from PG6 and is displayed. In this example, PG3 is configured as the CTS input, detecting a clear to send condition, and PG2 is configured as the RTS output, signaling a ready condition. This demonstration can be performed with either one or two boards.

- **SIMPLE3WIRE.C**—Demonstrates basic initialization for a simple RS-232 3-wire loopback displayed in the **STDIO** window.
- **SWITCHCHAR.C**—This program transmits and then receives an ASCII string on Serial Ports E and F when a switch is pressed. It also displays the serial data received from both ports in the **STDIO** window.
- **SIMPLE485MASTER.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 Coyote and display them in the **STDIO** window. Use **SIMPLESLAVE.C** to program the slave.
- **SIMPLE485SLAVE.C**—This program demonstrates a simple RS-485 transmission of lower case letters to a master Coyote. The slave will send back converted upper case letters back to the master Coyote and display them in the **STDIO** window. Use **SIMPLEMASTER.C** to program the master Coyote.

4.2.4 A/D Converter Inputs

The following sample programs are found in the **ADC** subdirectory in **SAMPLES\BL2500**.

- **AD0.C**—This program reads and displays the voltage on channel AD0 and its equivalent value to the **STDIO** window.
- **ADCCALIB.C**—This program demonstrates how to recalibrate one single-ended A/D converter channel using two known voltages to generate constants that are then rewritten into the user block data area.
- **COF_ANAIN.C**—This program demonstrates the use of the analog input driver as a cofunction. Connect DA1 to AD0 to provide an input voltage. When the program runs, it will read the input voltage ten times while another costate is executed concurrently. The values will be printed out at the end of the program.
- **DA2AD.C**—This program allows the user to input a voltage in the Dynamic C **STDIO** window for DA1 to output. The user needs to connect DA1 to AD0. The program will display the voltage read in the **STDIO** window.

4.2.5 D/A Converter Outputs

The following sample program is found in the **DAC** subdirectory in **SAMPLES\BL2500**.

- **DAC.C**—Demonstrates pulse-width modulation as an analog output voltage by displaying the voltage entered and measuring the voltage output.
- **DACCALIB.C**—Demonstrates how to recalibrate one single-ended analog output channel using two known voltages to generate constants for that channel that are then rewritten into the user block data area.
- **PWM.C**—Demonstrates pulse-width modulation as an analog output.

4.2.6 Using System Information from the RabbitCore Module

Calibration constants for the A/D converter are stored in the simulated EEPROM area of the flash memory. You may find it useful to retrieve the calibration constants and save them for future use, for example, if you should need to replace the RabbitCore module on the Coyote.

The following sample programs, found in the **ADC** subdirectory in **SAMPLES\BL2500**, illustrate how to save or retrieve the calibration constants. Note that both sample programs prompt you to use a serial number for the Coyote. This serial number can be any 5-digit number of your choice, and will be unique to a particular Coyote. Do ***not*** use the MAC address on the bar code label of the RabbitCore module attached to the Coyote since you may at some later time use that particular RabbitCore module on another Coyote, and the previously saved calibration data would no longer apply.

- **UPLOADCALIB.C**—This program demonstrates reading calibration constants from a controller's user block in flash memory and transmitting the file using a serial port with a PC serial utility such as Tera Term.

  NOTE: Use the sample program **DNLOADCALIB.C** to retrieve the data and rewrite it to the single-board computer.

- **DNLOADCALIB.C**—This program demonstrates how to retrieve your analog calibration data to rewrite them back to simulated EEPROM in flash using a serial utility such as Tera Term.

  NOTE: Calibration data must be saved previously in a file by the sample program **UPLOADCALIB.C**.

  NOTE: In addition to loading the calibration constants on the replacement RabbitCore module, you will also have to add the product information for the Coyote to the ID block associated with the RabbitCore module. The sample program **WRITE_IDBLOCK.C**, available on the Rabbit Web site at www.rabbit.com/support/feature_downloads.shtml, provides specific instructions and an example.

4.2.7 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. Set the real-time clock using the **SETRTCKB.C** sample program from the Dynamic C **SAMPLES\RTCLOCK** folder, using the onscreen prompts. 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.

4.3 Coyote Libraries

With Dynamic C running, click **File** > **Open**, and select **Lib**. The following list of Dynamic C libraries and library directories will be displayed. Two library directories provide libraries of function calls that are used to develop applications for the Coyote.

- **BL2500**—libraries associated with features specific to the Coyote. The functions in the **BL25xx.LIB** library are described in Section 4.4, "Coyote Function Calls.".
- **RN_CFG_BL25.LIB**—used to configure the BL2500 for use with RabbitNet peripheral cards.
- **TCPIP**—libraries specific to using TCP/IP functions.

Other generic functions applicable to all devices based on the Rabbit 3000 microprocessor are described in the ***Dynamic C Function Reference Manual***.

4.4 Coyote Function Calls

4.4.1 Board Initialization

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

Call this function at the beginning of your program. This function initializes Parallel Ports A through G for use with the Coyote. The ports are initialized according to Table A-3.

Summary of initialization

1. RS-485 is not initialized.
2. RS-232 is not initialized.
3. Unused configurable I/O are either pulled-up inputs or outputs set high.
4. PWM for DA0 and DA1 set to 57,600 Hz, and output voltage is zero. Uses functions **`pwm_init()`**, **`pwmOutConfig()`**, and **`pwmOut()`**.
5. Calibration constants for analog channels AD0, DA0, and DA1 are read from flash user block.

4.4.2 Digital I/O

void digOut(int channel, int value);

Sets the state of digital outputs OUT0–OUT7, whereOUT0–OUT7 are sinking outputs.

A run-time error will occur for the following conditions:

1. **channel** or **value** is out of range.
2. **brdInit** was not called first.

PARAMETERS

channel is the digital output channels (0–7)

value is the output value (0 or 1)

RETURN VALUE

None.

SEE ALSO

digIn, digBankOut

void digBankOut(int bank, int value);

Writes the state of a block of designated digital output channels. The bank consists of OUT0–OUT7. This call is faster than setting the individual channels, but does not output states simultaneously. States are written in succession from OUT7–OUT0.

A run-time error will occur for the following conditions:

1. **bank** or **value** is out of range.
2. **brdInit** was not called first.

PARAMETERS

bank is 0 for the bank of digital output channels (0–7)

value is an 8-bit output value, where each bit corresponds to one channel. OUT0 is the least significant bit 0

RETURN VALUE

None.

EXAMPLE

To send out odd channels high:

```
void digBankOut(0, 0xaa);
```

SEE ALSO

digOut, digBankIn

int digIn(int channel);

Reads the state of an input channel (IN00–IN15).

A run-time error will occur for the following conditions:

1. **channel** out of range.
2. **brdInit** was not executed before executing **digIn**.

PARAMETER

channel is the input channel number (0–15)

RETURN VALUE

The logic state of the input (0 or 1).

SEE ALSO

digOut, digBankIn

int digBankIn(int bank);

Reads the state of a block of designated digital input channels. One bank consists of IN0–IN07, and the other bank consists of IN08–IN15. This call is faster than reading the individual channels, but does not read the states simultaneously. States are read in succession from IN15–IN08 or from IN07–IN00.

A run-time error will occur for the following conditions:

1. **bank** is out of range.
2. **brdInit** was not called first.

PARAMETER

bank is 0 for the bank of digital inputs IN00–IN07, 1 for the bank of digital inputs IN08–IN15

RETURN VALUE

An input value in the lower byte, where each bit corresponds to one channel. IN00 and IN08 are in the bit 0 place.

EXAMPLE

To read inputs 8 to 15:

```
int digBankIn(1);
```

SEE ALSO

digIn, digBankOut

4.4.3 LEDs

```
void ledOut(int led, int value);
```

LED on/off control.

PARAMETERS

`led` is the LED to control

0 = LED DS1
1 = LED DS2
2 = LED DS3
3 = LED DS4

`value` is used to control whether the LED is on or off

0 = OFF
1 = ON

RETURN VALUE

None.

4.4.4 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 Function Reference Manual*** and Technical Note 213, ***Rabbit Serial Port Software***.

Use the following function calls with the Coyote.

void ser485Tx(void);

Enables the RS485 transmitter. Transmitted data get echo'ed back into the receive data buffer. These echo'ed data could be used to know when to disable the transmitter by using one of the following methods:

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

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

RETURN VALUE

None.

SEE ALSO

ser485Rx, serXopen

void ser485Rx(void);

Disables the RS-485 transmitter. This puts the Coyote in listen mode, which allows it to receive data from the RS-485 interface.

RETURN VALUE

None.

SEE ALSO

ser485Tx, serXopen

4.4.5 Analog Inputs

unsigned int anaIn(unsigned int channel);

Uses D/A converter channel DA0 to search through the full voltage range for a match to the input voltage on channel AD0. This is done using a 10-step successive-approximation binary search, which nominally takes 86 ms.

Call **pwmOutConfig()** and **pwm_init()** before using this function. An exception error will occur if these functions were not been called previously.

NOTE: DA0 should not be used when AD0 is in use.

PARAMETER

channel is 0 for channel AD0.

RETURN VALUE

An integer value between 0 and 1023 that corresponds to a voltage between 0.0 and 3.3 V on the analog input channel.

-1 if the return value is out of range.

SEE ALSO

cof_anaIn, anaInVolts

void cof_anaIn(int channel);

This function is the cofunction version of the analog input for analog input channel AD0. This version will "yield" on each step approximation in a costate, and will take 10 steps to complete the A/D conversion. The function will also process costates while waiting for each approximation to settle.

NOTE: All the restrictions for **anaIn** apply to **cof_anaIn**.

PARAMETERS

channel is 0 for channel AD0

RETURN VALUE

An integer value between 0 and 1023 that corresponds to a voltage between 0.0 and 3.3 V on the analog input channel.

-1 if the return value is out of range.

SEE ALSO

anaIn

float anaInVolts(unsigned int channel);

Reads the voltage of a single-ended analog input channel using D/A channel DA0 for comparison to find a match to the input voltage on channel AD0. This is done using a 10-step successive-approximation binary search, which nominally takes 86 ms.

Call **pwmOutConfig()** and **pwm_init()** before using this function. An exception error will occur if these functions were not been called previously.

NOTE: DA0 should not be used when AD0 is in use.

PARAMETER

channel is 0 for channel AD0

RETURN VALUE

A voltage value between 0 and 3.1 V for the analog input channel.

ADOVERFLOW is returned (defined macro = -4096) on overflow or if the return value is out of range.

SEE ALSO

anaIn, pwmOutConfig, pwm_init

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

Calibrates the response of the A/D converter channel as a linear function using the two conversion points provided. Values are calculated and placed into global table **_adcCalibS** for analog inputs to be stored later into simulated EEPROM using the function **anaInEEWr()**.

Each channel will have a linear constant and a voltage offset.

PARAMETERS

channel is 0 for channel AD0

value1 is the first A/D converter value (0–1023), usually a value of 310 that corresponds to 1.0 V

volts1 is the voltage corresponding to the first A/D converter value (0–3.3 V)

value2 is the second A/D converter value (0–1023), usually a value of 930 that corresponds to 3.0 V

volts2 is the voltage corresponding to the second A/D converter value (0–3.3 V)

RETURN VALUE

0 if successful

-1 if not able to make calibration constants

SEE ALSO

anaIn, anaInEERd, anaInEEWr

int anaInEERd(unsigned int channel);

Reads the calibration constants, gain, and offset for an input based on its designated channel code position into global table **_adcCalibS**. Use the sample program **USERBLOCK_INFO.C** in **SAMPLES\USERBLOCK** to get the addresses reserved for the calibration data constants and the addresses available for use in your program.

NOTE: This function cannot be run in RAM.

PARAMETERS

channel is 0 for channel AD0

RETURN VALUE

0 if successful

-1 if invalid address or range

SEE ALSO

anaInEEWr, anaInCalib

int anaInEEWr(unsigned int channel);

Writes the calibration constants, gain, and offset for an input based on its designated channel code position into global table **_adcCalibS**. Use the sample program **USERBLOCK_INFO.C** in **SAMPLES\USERBLOCK** to get the addresses reserved for the calibration data constants and the addresses available for use in your program.

NOTE: This function cannot be run in RAM.

PARAMETER

channel is 0 for channel AD0

RETURN VALUE

0 if successful

-1 if invalid address or range

SEE ALSO

anaInEERd, anaInCalib

4.4.6 Analog Outputs

```
unsigned long pwm_init(unsigned long frequency);
```

This function from the **R3000.LIB** library in **Lib\Rabbit3000** sets the base frequency for the PWM pulses and enables the PWM driver on all four channels. The base frequency is the frequency without pulse spreading. Pulse spreading will increase the frequency by a factor of 4.

PARAMETER

frequency is the frequency (in Hz)

RETURN VALUE

Actual frequency set. This will be the closest possible match to the requested frequency.

```
void pwmOutConfig(unsigned int channel,
   int pwmoption);
```

Option flags are used to enable features on an individual PWM channels. Use **pwm_init()** to set the frequency.

An exception error will occur if **brdInit()** has not been called previously.

PARAMETERS

channel is the PWM output channel to set: 0 for DA0, 1 for DA1.

pwmoption is used to set the PWM options as a combination of the following bit masks:

PWM_NORMAL—sets normal push-pull logic output.

PWM_SPREAD—Set pulse spreading. The duty cycle is spread over four separate pulses to increase the pulse frequency. Use this option for A/D and D/A conversions.

PWM_OPENDRAIN—sets the PWM output pin to be open-drain. This mask is usually not used.

RETURN VALUE

None.

SEE ALSO

pwm_init, brdInit

int pwmOut(unsigned int channel, int rawdata);

Sets a voltage (0 to V_{dd}) on an analog output channel given a data point on the 1024 clock count cycle.

Call **pwmOutConfig()** and **pwm_init()** before using this function. (An exception error will occur if these functions were not been called previously.)

PARAMETERS

channel is the PWM output channel to write: 0 for DA0, 1 for DA1

rawdata is data value (0–1024) for a 1024 clock count cycle. The value may be calculated using the percent duty cycle value (percentage that is on or high) of the 1024 clock count cycle, for example, 0.25*1024.

RETURN VALUE

0 if successful

SEE ALSO

pwmOutConfig, pwm_init

void pwmOutVolts(unsigned int channel, float voltage);

Sets the voltage of an analog output channel by using the previously set calibration constants to calculate the correct data values.

Call **pwmOutConfig()** and **pwm_init()** before using this function. (An exception error will occur if these functions were not been called previously.)

PARAMETERS

channel is the output channel 0 or 1 to write: 0 for DA0, 1 for DA1

voltage is the voltage desired on the output channel (0–3.3 V)

RETURN VALUE

None.

SEE ALSO

pwmOut, pwmOutConfig, pwm_init

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

Calibrates the response of the D/A converter channel as a linear function using the two conversion points provided. Values are calculated and placed into global table **_dacCalibS** for analog inputs to be stored later into simulated EEPROM using the function **anaOutEEWr()**.

Each channel will have a linear constant and a voltage offset.

PARAMETERS

channel is the output channel 0 or 1: 0 for DA0, 1 for DA1

value1 is the first D/A converter value (0–1023), usually a value of 310 that corresponds to 1.0 V

volts1 is the voltage corresponding to the first D/A converter value (0–3.3 V or V_{ref})

value2 is the second D/A converter value (0–1023), usually a value of 930 that corresponds to 3.0 V

volts2 is the voltage corresponding to the second D/A converter value (0–3.3 V or V_{ref})

RETURN VALUE

0 if successful

-1 if not able to make calibration constants

SEE ALSO

pwmOut, anaOutEERd, anaOutEEWr

int anaOutEERd(unsigned int channel);

Reads the calibration constants, gain, and offset for an output based on its designated channel code position into global table **_adcCalibS**. Use the sample program **USERBLOCK_INFO.C** in **SAMPLES\USERBLOCK** to get the addresses reserved for the calibration data constants and the addresses available for use in your program.

NOTE: This function cannot be run in RAM.

PARAMETERS

channel is the output channel 0 or 1: 0 for DA0, 1 for DA1

RETURN VALUE

0 if successful

-1 if invalid address or range

SEE ALSO

anaOutEEWr, anaOutCalib

int anaOutEEWr(unsigned int channel);

Writes the calibration constants, gain, and offset for an output based on its designated channel code position into global table **_adcCalibS**. Use the sample program **USERBLOCK_INFO.C** in **SAMPLES\USERBLOCK** to get the addresses reserved for the calibration data constants and the addresses available for use in your program.

> **NOTE:** This function cannot be run in RAM.

PARAMETER

channel is the output channel 0 or 1: 0 for DA0, 1 for DA1

RETURN VALUE

0 if successful

-1 if invalid address or range

SEE ALSO

anaOutEERd, anaOutCalib

4.4.7 RabbitNet Port

The function calls described in this section are used to configure the BL2500 for use with RabbitNet peripheral cards. The user's manual for the specific peripheral card you are using contains additional function calls related to the RabbitNet protocol and the individual peripheral card.

Add the following lines at the start of your program.

```
#define RN_MAX_DEV 10  // max number of devices
#define RN_MAX_DATA 16 // max number of data bytes in any transaction
#define RN_MAX_PORT 2  // max number of serial ports
```

Set the following bits in **RNSTATUSABORT** to abort transmitting data after the status byte is returned. This does not affect the status byte and still can be interpreted. Set any bit combination to abort:

bit 7—device busy is hard-coded into driver
bit 5—identifies router or slave
bits 4,3,2—peripheral-board-specific bits
bit 1—command rejected
bit 0—watchdog timeout

```
#define RNSTATUSABORT 0x80
     // hard-coded driver default to abort if the peripheral card is busy
```

void rn_sp_info();

Provides **rn_init()** with the serial port control information needed for BL2500 series controllers.

RETURN VALUE

None.

void rn_sp_close(int port);

Deactivates the BL2500 RabbitNet port as a clocked serial port. This call is also used by **rn_init()**.

PARAMETERS

portnum = 0

RETURN VALUE

None

void rn_sp_enable(int portnum);

This is a macro that enables or asserts the BL2500 RabbitNet port select prior to data transfer.

PARAMETERS

portnum = 0

RETURN VALUE

None

void rn_sp_disable(int portnum);

This is a macro that disables or deasserts the BL2500 RabbitNet port select to invalidate data transfer.

PARAMETERS

portnum = 0

RETURN VALUE

None.

5. USING THE TCP/IP FEATURES

Chapter 5 discusses using the TCP/IP features on the Coyote boards.

5.1 TCP/IP Connections

Before proceeding you will need to have the following items.

- If you don't have an Ethernet connection, you will need to install a 10Base-T Ethernet card (available from your favorite computer supplier) in your PC.
- Two RJ-45 straight-through Ethernet cables and a hub, or an RJ-45 crossover Ethernet cable.

The Ethernet cables and Ethernet hub are available from Rabbit in a TCP/IP tool kit. More information is available at www.rabbit.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 Ethernet cable to connect the Coyote to a PC that at least has a 10Base-T Ethernet card.
- If you have an Ethernet connection, use a straight-through Ethernet cable to establish an Ethernet connection to the Coyote from an Ethernet hub. These connections are shown in Figure 20.

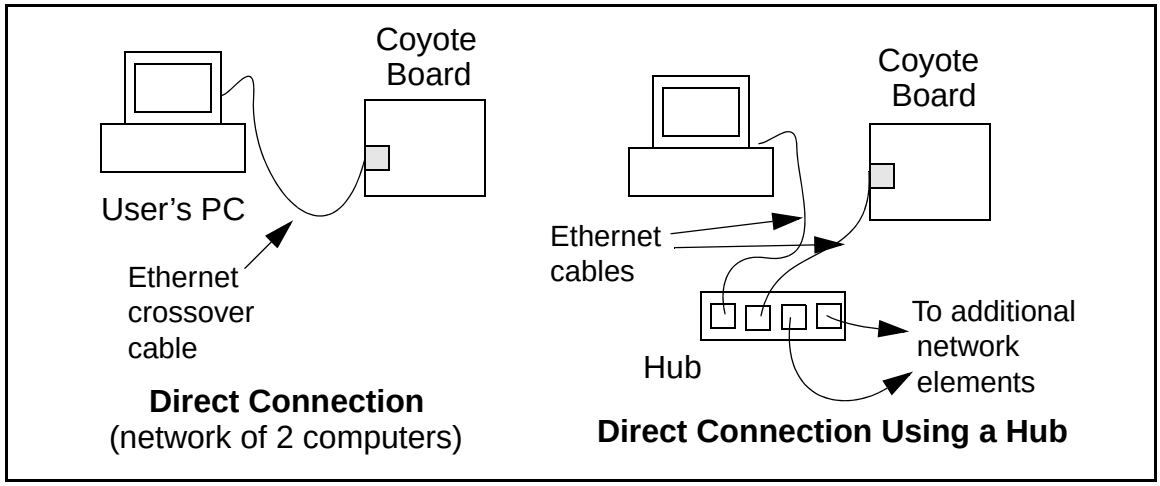


Figure 20. Ethernet Connections

3. Apply Power

Plug in the AC adapter. The Coyote 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 reset pins on the back of the Coyote.
>
> When the **PROG** connector of the programming cable connects the Coyote to your PC, and Dynamic C is running, a RESET occurs when you press **<Ctrl-Y>**.

The green **LNK** light on the Coyote's RabbitCore module is on when the Coyote 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 Coyote 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 Coyote 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***.

IP Addresses Before Dynamic C 7.30

Most of the sample programs use macros to define the IP address assigned to the board and the IP address of the gateway, if there is a gateway. Instead of the **TCPCONFIG** macro, you will see a **MY_IP_ADDRESS** macro and other macros.

```
#define MY_IP_ADDRESS "10.10.6.170"
#define MY_NETMASK "255.255.255.0"
#define MY_GATEWAY "10.10.6.1"
#define MY_NAMESERVER "10.10.6.1"
```

In order to do a direct connection, the following IP addresses can be used for the Coyote:

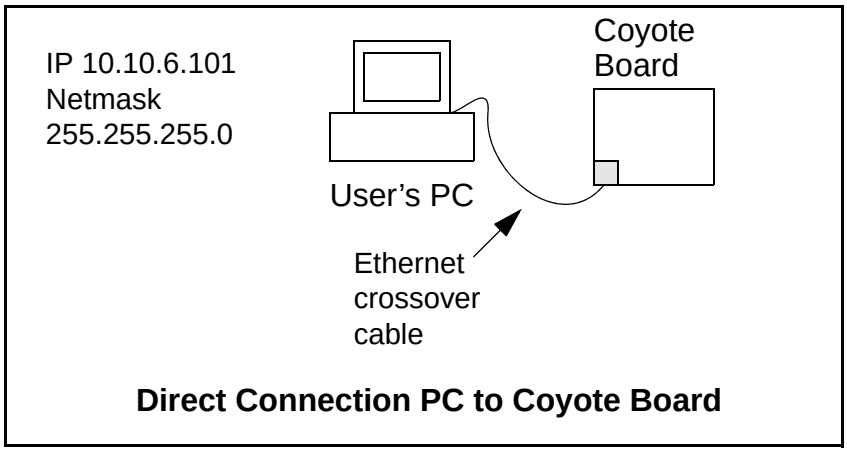
```
#define MY_IP_ADDRESS "10.1.1.2"
#define MY_NETMASK "255.255.255.0"
// #define MY_GATEWAY "10.10.6.1"
// #define MY_NAMESERVER "10.10.6.1"
```

In this case, the gateway and nameserver are not used, and are commented out. The IP address of the board is defined to be **10.1.1.2**. The IP address of you PC can be defined as **10.1.1.1**.

5.2.2 How to Set Up your Computer's IP Address for a Direct Connection

When your computer is connected directly to the Coyote via an Ethernet connection, you need to assign an IP address to your computer. To assign the PC the address **`10.10.6.101`** with the netmask **`255.255.255.0`**, do the following.

Click on **Start > Settings > Control Panel** to bring up the Control Panel, and then double-click the Network icon. Depending on which version of Windows you are using, look for the **TCP/IP Protocol**/**Network > Dial-Up Connections**/**Network** line or tab. Double-click on this line or select **Properties** or **Local Area Connection > Properties** to bring up the TCP/IP properties dialog box. You can edit the IP address and the subnet mask directly. (Disable "obtain an IP address automatically.") You may want to write down the existing values in case you have to restore them later. It is not necessary to edit the gateway address since the gateway is not used with direct connect.



Direct Connection PC to Coyote Board

5.2.3 Run the PINGME.C Demo

Connect the crossover cable from your computer's Ethernet port to the Coyote'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 Coyote 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 Coyote 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 sample programs discussed in this section use the Demonstration Board from the BL2500/OEM2500 Development Kit to illustrate their operation. Appendix C, "Demonstration Board Connections," contains diagrams of typical connections between the Coyote and the Demonstration Board used to run these sample programs.

The program **SMPTP.C** (**SAMPLES\BL2500\TCPIP**) uses the SMTP library to send an e-mail when a switch on the Demonstration Board is pressed.

The program **BROWSELED.C** (**SAMPLES\BL2500\TCPIP**) demonstrates a basic controller running a Web page. Two "LEDs" are created on the Web page, and two buttons on the Demonstration Board then toggle them. Users can change the status of the lights from the Web browser. The LEDs on the Demonstration Board match the ones on the Web page. 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 program **PINGLED.C** (**SAMPLES\BL2500\TCPIP**) demonstrates ICMP by pinging a remote host. It will flash LEDs DS1 and DS2 on the Demonstration Board when a ping is sent and received.

5.3 Where Do I Go From Here?

NOTE: If you purchased your Coyote 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.rabbit.com/support/bb/ 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.

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.

APPENDIX A. SPECIFICATIONS

Appendix A provides the specifications for the Coyote.

A.1 Electrical and Mechanical Specifications

Figure A-1 shows the mechanical dimensions for the Coyote.

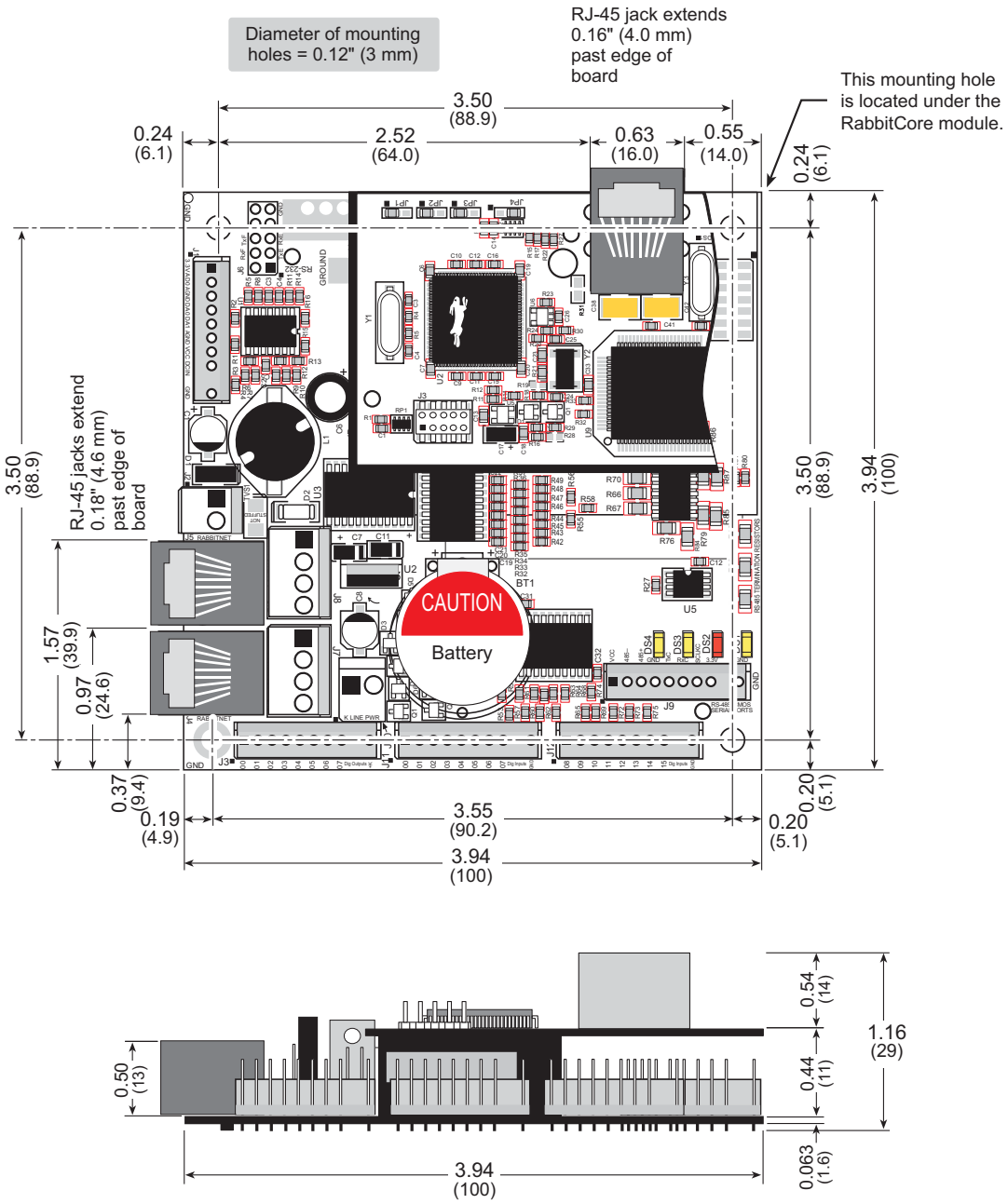


Figure A-1. Coyote Dimensions

NOTE: All measurements are in inches followed by millimeters enclosed in parentheses.

Table A-1 lists the electrical, mechanical, and environmental specifications for the Coyote.

Table A-1. Coyote Specifications

Feature	BL2500	BL2510
Microprocessor	Rabbit 3000® at 29.4 MHz	
Ethernet Port	10/100-compatible with 10Base-T interface	—
Flash Memory	256K standard, 512K (2 × 256K) option	
SRAM	128K standard, 512K option	
Backup Battery	3 V lithium coin type, 1000 mA·h, supports RTC and SRAM*	
LEDs	4, user-programmable	
Digital Inputs	16†: 15 protected to ±36 V DC, 1 protected from -36 V to +5 V DC, switching threshold is 1.5 V nominal	
Digital Outputs	8: sink up to 200 mA each, 36 V DC max.	
Analog Inputs	One 10-bit resolution, 8-bit accuracy, input range 0.1–3.1 V, 10 samples/s	
Analog Outputs	Two 9-bit PWM, 0.1–3.1 V DC, worst-case 17 ms settling time to within 5 mV of final value (built-in RC settling time constant = 2.5 ms)	
Serial Ports	6 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 CMOS level asynchronous or clocked serial port • one serial port dedicated for programming/debug	
Serial Rate	Max. asynchronous rate = CLK/8, Max. synchronous rate = CLK/2	
Real-Time Clock	Yes	
Timers	Ten 8-bit timers (6 cascadable, 3 are reserved for internal peripherals), one 10-bit timer with 2 match registers	
Watchdog/Supervisor	Yes	
Power	8–40 V DC (RabbitNet peripheral cards are limited to 32 V DC max.)	
	1 W typ. with no load	0.8 W typ. with no load
Temperature	-40°C to +70°C	
Humidity	5% to 95%, noncondensing	
Connectors	Friction-lock connectors: five polarized 9-position terminals with 0.1" pitch two 2-position power terminals with 0.156" pitch two 4-position terminals with 0.156" pitch	
Unit Size	3.94" × 3.94" × 1.16" (100 mm × 100 mm × 29 mm)	3.94" × 3.94" × 0.80" (100 mm × 100 mm × 20 mm)

* not present on standard OEM versions

† only 8 protected inputs on standard OEM versions

A.1.1 Exclusion Zone

It is recommended that you allow for an “exclusion zone” of 0.25" (6 mm) around the Coyote in all directions when the Coyote is incorporated into an assembly that includes other components. An “exclusion zone” of 0.12" (3 mm) is recommended below the Coyote. Figure A-2 shows this “exclusion zone.”

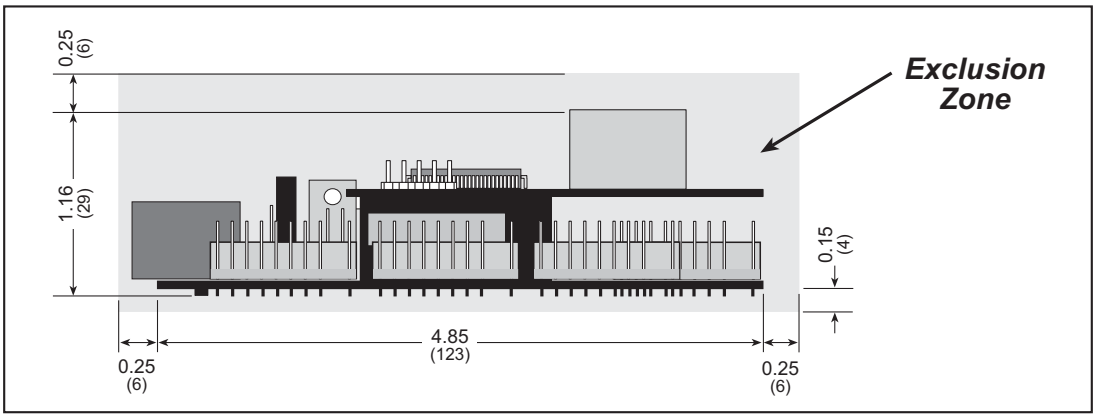


Figure A-2. Coyote “Exclusion Zone”

A.1.2 Physical Mounting

Figure A-3 shows position information to assist with interfacing other boards with the Coyote.

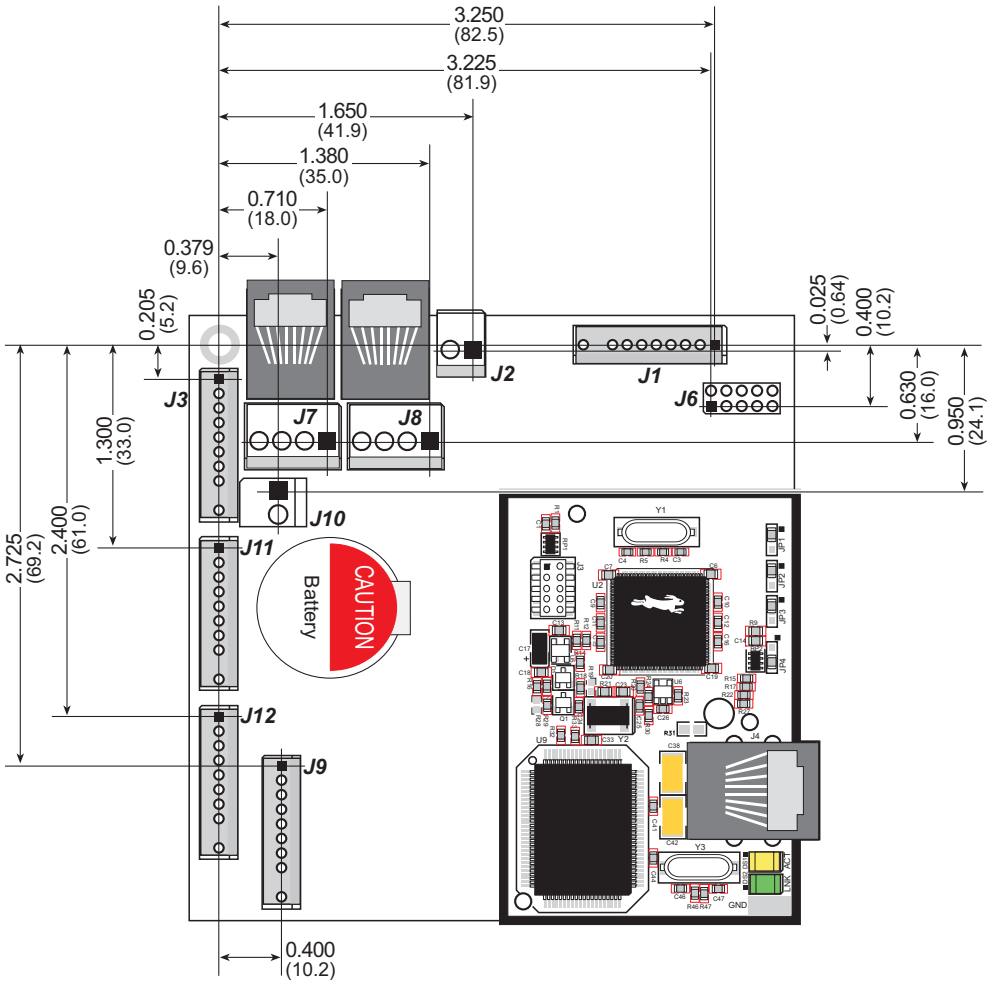


Figure A-3. User Board Footprint for Coyote

A.2 Conformal Coating

The areas around the crystal oscillator and the battery backup circuit on the Coyote's RabbitCore module have had the Dow Corning silicone-based 1-2620 conformal coating applied. The conformally coated areas are shown in Figure A-4. The conformal coating protects these high-impedance circuits from the effects of moisture and contaminants over time, and helps to maintain the accuracy of the real-time clock.

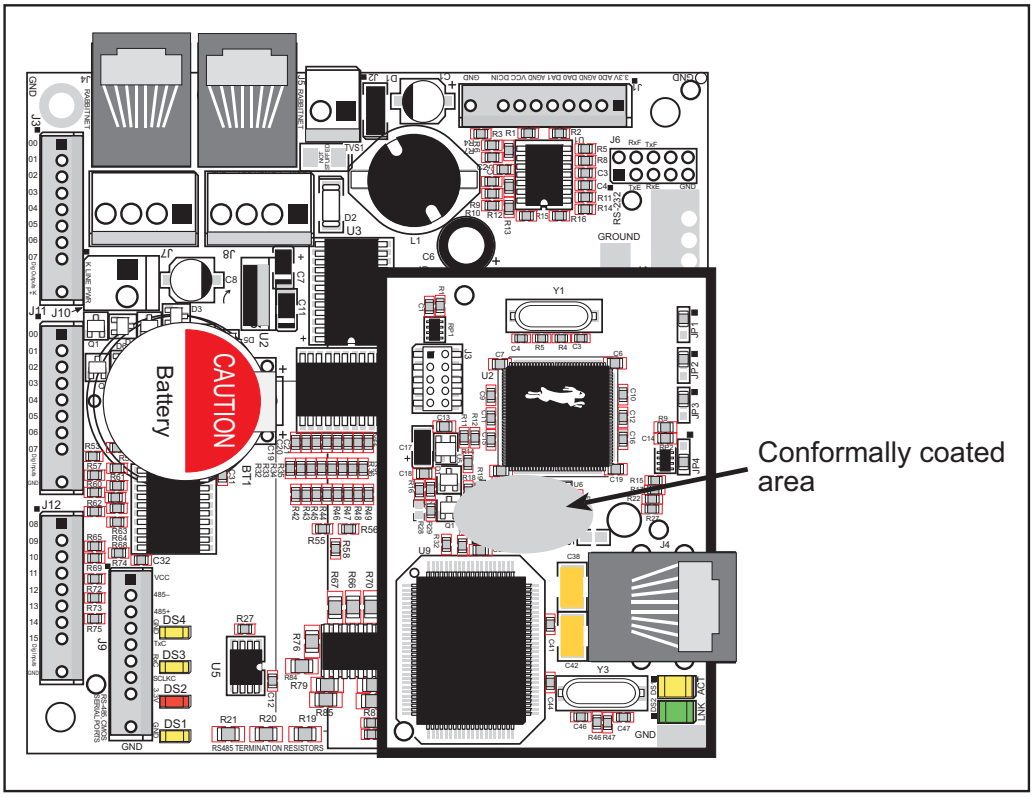


Figure A-4. Coyote's RabbitCore Module Areas Receiving Conformal Coating

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-5 shows the header and jumper locations used to configure the various Coyote options.

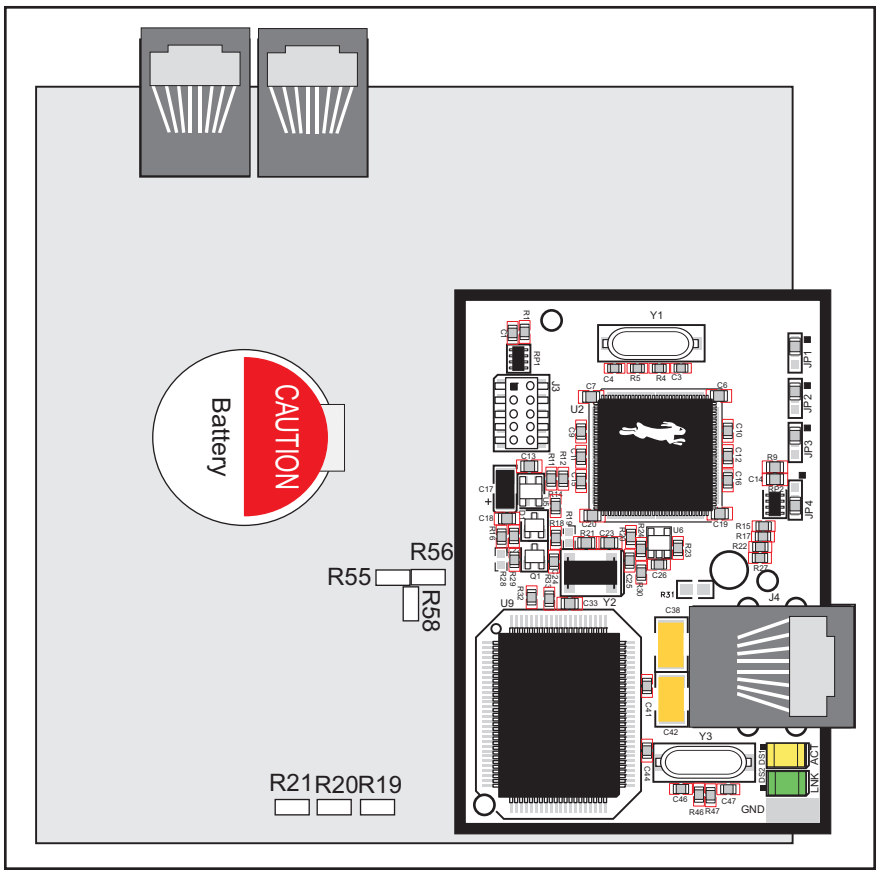


Figure A-5. Location of Coyote Configurable Positions (RabbitCore module is not shown)

Table A-2 lists the configuration options. 0 Ω surface mount resistors are used for all the positions except JP10 and J8, which use standard pluggable jumpers.

Table A-2. Coyote Jumper Configurations

Description	Pins Connected		Factory Default
IN00–IN07	R58	Pulled up to +3.3 V	✕
	R56	Pulled down	
	R55	Pulled up to +K	
RS-485 Bias and Termination Resistors	R20	Termination resistor	✕
	R19 R21	Bias resistors	✕

A.4 Use of Rabbit 3000 Parallel Ports

Figure A-6 shows the Rabbit 3000 parallel ports.

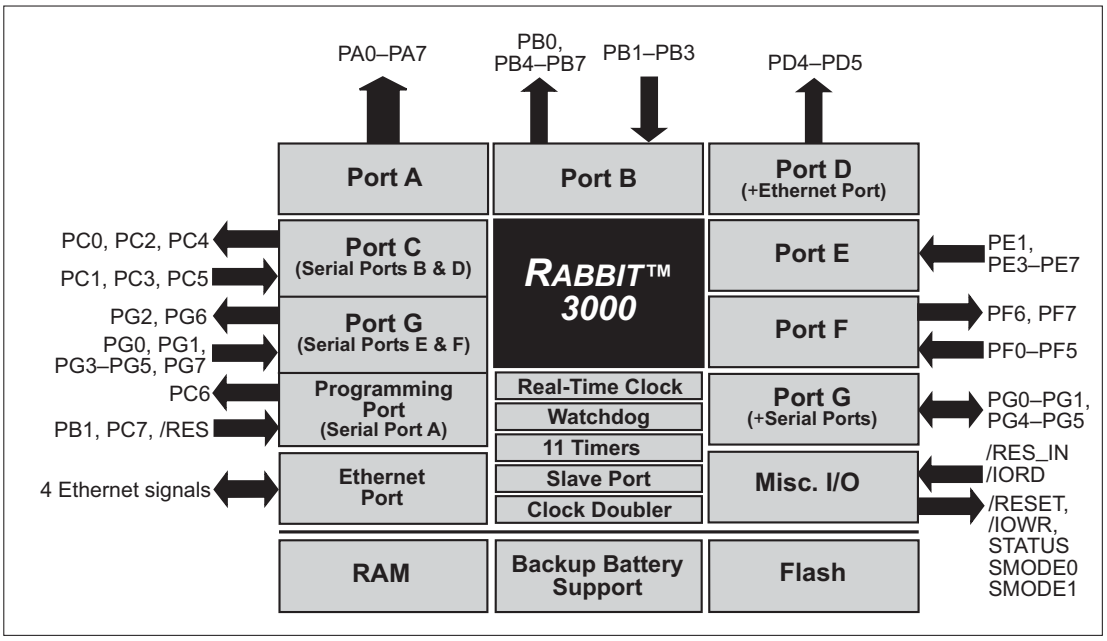


Figure A-6. Coyote Rabbit-Based Subsystems

Table A-3 lists the Rabbit 3000 parallel ports and their use in the Coyote.

Table A-3. Use of Rabbit 3000 Parallel Ports

Port	I/O	Signal	Initial State
PA0	Output	OUT0	Low
PA1	Output	OUT1	Low
PA2	Output	OUT2	Low
PA3	Output	OUT3	Low
PA4	Output	RS-485 Transmit Enable	Low (disables transmit)
PA5	Output	SPI Select	Low (= select SPI1) (high selects SPI2)
PA6	Output	LED DS4	High (disabled)
PA7	Output	LED DS3	High (disabled)
PB0	Output	CLKB SPI	High
PB1	Input	Programming Port Clock	High
PB2	Input	AD0 Low Comparator	Driven by comparator
PB3	Input	AD0 High Comparator	Driven by comparator

Table A-3. Use of Rabbit 3000 Parallel Ports (continued)

Port	I/O	Signal		Initial State
PB4	Output	OUT6		Low
PB5	Output	OUT7		Low
PB6	Output	LED DS1		High (disabled)
PB7	Output	LED DS2		High (disabled)
PC0	Output	TXD RS-485	Serial Port D	Inactive high
PC1	Input	RXD RS-485		Inactive high
PC2	Output	Configurable		Low
PC3	Input	IN14		Pulled up to 3.3 V
PC4	Output	TXB SPI	Serial Port B	Inactive high
PC5	Input	RXB SPI		Inactive high
PC6	Output	TXA Programming Port	Serial Port A	Inactive high
PC7	Input	RXA Programming Port		Inactive high
PD0	Output	Realtek RSTDRV		Inactive high
PD1	Output	Not Used		High
PD2	Output	Ethernet		High
PD3	Output	Ethernet		High
PD4	Output	OUT4		Low
PD5	Output	OUT5		Low
PD6	Output	Ethernet		High
PD7	Output	Ethernet		High
PE0	Output	Not Used		High
PE1	Input	IN00		Pulled up to 3.3 V
PE2	Output	Realtek AEN		High
PE3	Input	IN01		Pulled up to 3.3 V
PE4	Input	IN13		Pulled up to 3.3 V
PE5	Input	IN12		Pulled up to 3.3 V
PE6	Input	IN02		Pulled up to 3.3 V
PE7	Input	IN03		Pulled up to 3.3 V
PF0	Input	IN15		Pulled up to 3.3 V
PF1	Input	Configurable		Pulled up to 3.3 V
PF2	Input	IN08		Pulled up to 3.3 V

Table A-3. Use of Rabbit 3000 Parallel Ports (continued)

Port	I/O	Signal		Initial State
PF3	Input	IN09		Pulled up to 3.3 V
PF4	Input	IN10		Pulled up to 3.3 V
PF5	Input	IN11		Pulled up to 3.3 V
PF6	Output	DA0		High
PF7	Output	DA1		High
PG0	Input	IN04		Pulled up to 3.3 V
PG1	Input	IN05		Pulled up to 3.3 V
PG2	Output	TXF RS-232	Serial Port F	High
PG3	Input	RXF RS-232		High
PG4	Input	IN06		Pulled up to 3.3 V
PG5	Input	IN07		Pulled up to 3.3 V
PG6	Output	TXE RS-232	Serial Port E	High
PG7	Input	RXE RS-232		High

APPENDIX B. POWER SUPPLY

Appendix B describes the power circuitry provided on the Coyote.

B.1 Power Supplies

Power is supplied to the Coyote via the friction-lock connector terminal at J2. The Coyote has an onboard +5 V switching power regulator from which a +3.3 V linear regulator draws its supply. Thus both +5 V and +3.3 V are available. The Coyote is protected against reverse polarity by a diode at D1 as shown in Figure B-1.

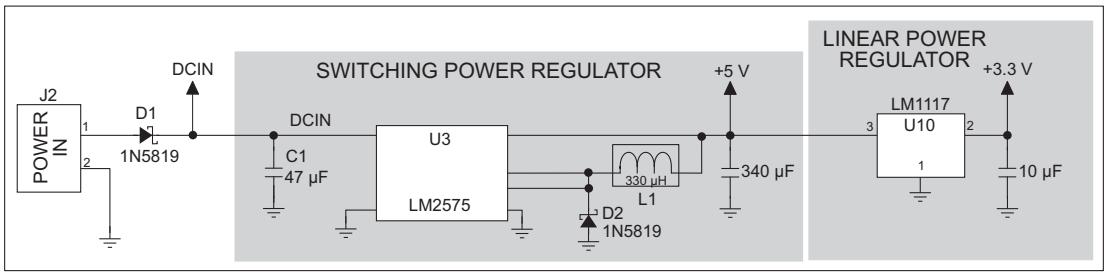


Figure B-1. Coyote Power Supply

The input voltage range is from 8 V to 40 V DC.

There is provision on the printed-circuit board for a transorb to be installed at TVS1 in parallel with C1 to provide suppression for positive noise pulses above 51 V. This part is only needed when the Coyote will be used in industrial environments where a clean source of power cannot be guaranteed, and is not part of the normal factory build.

B.2 Batteries and External Battery Connections

The SRAM and the real-time clock have battery backup. Power to the SRAM and the real-time clock (VRAM) on the Coyote's RabbitCore module is provided by two different sources, depending on whether the main part of the Coyote is powered or not. When the Coyote is powered normally, and Vcc is within operating limits, the SRAM and the real-time clock are powered from Vcc. 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 soldered-in 1000 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 less than 10 µA when there is no external power applied to the Coyote, and so the expected shelf life of the battery is more than

$$\frac{1000\ \text{mA}\cdot\text{h}}{10\ \mu\text{A}} = 11.4\ \text{years.}$$

The drain on the battery is typically less than 4 µA when external power ***is*** applied, and so the expected battery in-service life is

$$\frac{1000\ \text{mA}\cdot\text{h}}{4\ \mu\text{A}} = 28\ \text{years.}$$

Since the nominal shelf life of the lithium battery is 10–20 years, the in-service life should not be of concern.

> **NOTE:** The SRAM contents and the real-time clock settings will be lost if the battery is replaced with no power applied to the Coyote. Exercise care if you replace the battery while external power is applied to the Coyote.

B.2.1 Power to VRAM Switch

The VRAM switch on the Coyote's RabbitCore module, shown in Figure B-2, allows the battery backup to provide power when the external power goes off. The switch provides an isolation between Vcc and the battery when Vcc goes low. This prevents the Vcc line from draining the battery.

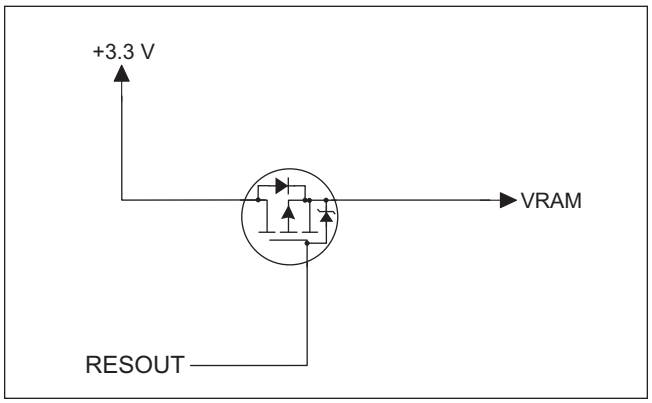


Figure B-2. VRAM Switch

The field-effect transistor provides a very small voltage drop between Vcc and VRAM (<100 mV, typically 10 mV) so that the board components powered by Vcc will not have a significantly different voltage than VRAM.

When the Coyote is ***not*** in reset, the **RESOUT** line will be high. This allows VRAM to nearly equal Vcc.

When the Coyote ***is*** in reset, the **RESOUT** line will go low. This provides an isolation between Vcc and VRAM.

B.2.2 Reset Generator

The Coyote's RabbitCore module uses a reset generator to reset the Rabbit 3000 microprocessor when the voltage drops below the voltage necessary for reliable operation. The reset typically occurs at 2.93 V (2.63 V for the BL2510).

B.3 Chip Select Circuit

The current drain on the battery in a battery-backed circuit must be kept at a minimum. When the Coyote is not powered, the battery keeps the SRAM memory contents and the real-time clock (RTC) going. The SRAM has a powerdown mode that greatly reduces power consumption. This powerdown mode is activated by raising the chip select (CS) signal line. Normally the SRAM requires Vcc to operate. However, only 2 V is required for data retention in powerdown mode. Thus, when power is removed from the circuit, the battery voltage needs to be provided to both the SRAM power pin and to the CS signal line. The CS control circuit accomplishes this task for the SRAM's chip select signal line.

B.4 Power to Peripheral Cards

DCIN and Vcc are available on friction-lock connector terminals J7 and J8 to power peripheral cards that may be used with the Coyote.

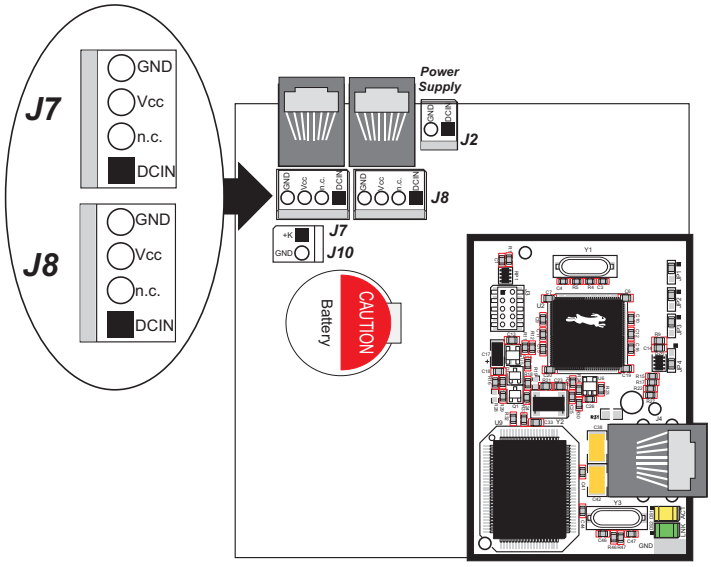


Figure B-3. Pinout Friction-Lock Connector Terminals J7 and J8

Keep in mind that the Coyote draws 377 mA from the Vcc supply, and that the diode at D1 (shown in Figure B-1) can handle at most 1 A at V_{RAW}, so that leaves the remaining current capacity to be shared among the DCIN and Vcc pins on friction-lock connector terminals J7 and J8. Table B-1 lists the available current at DCIN based on the current drawn at Vcc.

Table B-1. DCIN Current Available at J7 and J8 (in mA) Based on Power Supply and Vcc (= 5 V) Current Used at J7 and J8

V_{RAW} Power Supply Input at J2 (V)	Current at Vcc						
	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

APPENDIX C. DEMONSTRATION BOARD CONNECTIONS

Appendix C shows how to connect the Demonstration Board to the Coyote.

C.1 Assemble Wire Harness

Before you can connect the Demonstration Board to the Coyote to run the sample programs based on the Demonstration Board, you will need to assemble a wiring harness using the friction-lock connectors and crimp terminals supplied with the BL2500/OEM2500 Development Kit. In addition, you will need:

- Wire—22 to 30 AWG (0.33 mm^2 to 0.049 mm^2) for the 0.1" crimp terminals, 22 to 26 AWG (0.33 mm^2 to 0.13 mm^2) for the 0.156" crimp terminals
- Wire cutters and wire insulation stripper
- Crimp tool (pliers may be used, but a crimp tool provides a better crimp with a stronger force)

Rabbit sells a crimp tool and a Connectivity Kit that contains additional friction-lock connectors and crimp terminals. Table 3 in Chapter 3 provides information on specific friction-lock connectors and crimp terminals to be used with the various headers on the BL2500. Contact your authorized Rabbit distributor or your sales representative for more information.

Follow these steps to build your wire harness.

1. Prepare a few lengths of wire about 30 cm (12") long. The wires should have different colors of insulation to facilitate identifying the connections.
2. Trim about 2–3 mm (0.1") of insulation from your wire.
3. Position the wire in the crimp terminal as shown in Figure C-1.

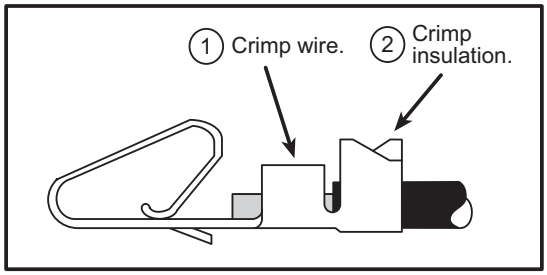


Figure C-1. Crimp Wire in Crimp Terminal

4. Use a crimp tool or pliers to first crimp the bare wire, then the insulation as shown in Figure C-1.
5. Insert the crimp terminals with wires into the friction-lock connector with the tab on the crimp terminal facing the opening on the side of the friction-lock connector. Insert the crimp terminal until the tab snaps into place in the side opening.

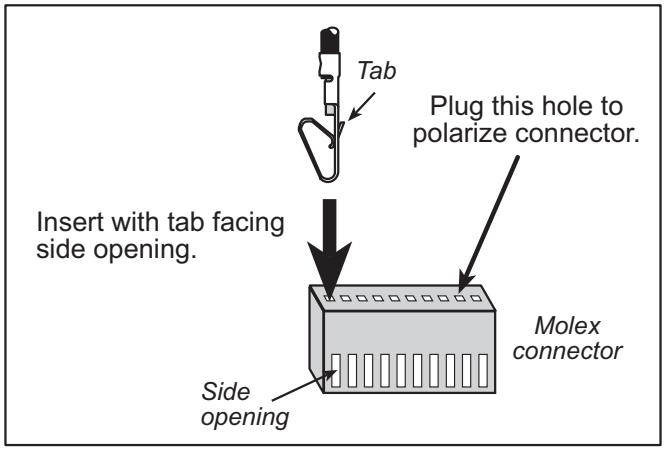


Figure C-2. Insert Crimp Terminals Into Friction-Lock Connector

6. Repeat these steps until all the wires and crimp terminals have been assembled.

TIP: Use different wire colors to help you color-code your harness.

TIP: On 10-pin friction-lock connectors, insert a plug into the hole indicated in Figure C-2 to polarize your connector to help prevent offsetting the connector by one pin when you attach it to your Coyote. Polarizing plugs are not included in Rabbit's Connectivity Kit.

C.2 Connecting Demonstration Board

Before running sample programs based on the Demonstration Board, you will have to connect the Demonstration Board from the BL2500/OEM2500 Development Kit to the Coyote board. Proceed as follows.

1. Use one of the wiring harnesses you have built to connect header J1 on the Demonstration Board to the Coyote. The connections are shown in Figure C-3 for sample program **`DIGIN.C`**, in Figure C-4 for sample program **`DIGOUT.C`**, and in Figure C-5 for the **`BL2500\TCPIP`** TCP/IP sample programs.
2. Make sure that your Coyote is connected to your PC and that the power supply is connected to the Coyote and plugged in as described in Chapter 2, "Getting Started."

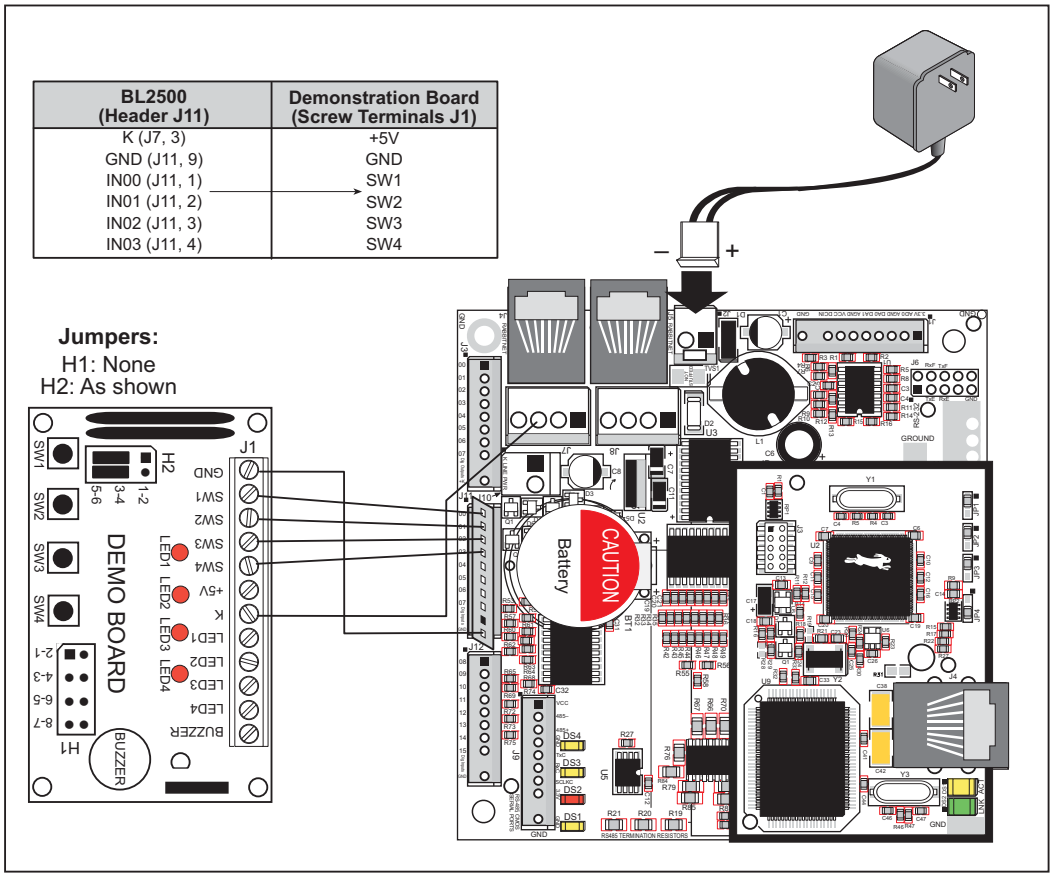


Figure C-3. Connections Between Coyote and Demonstration Board for `DIGIN.C` Sample Program

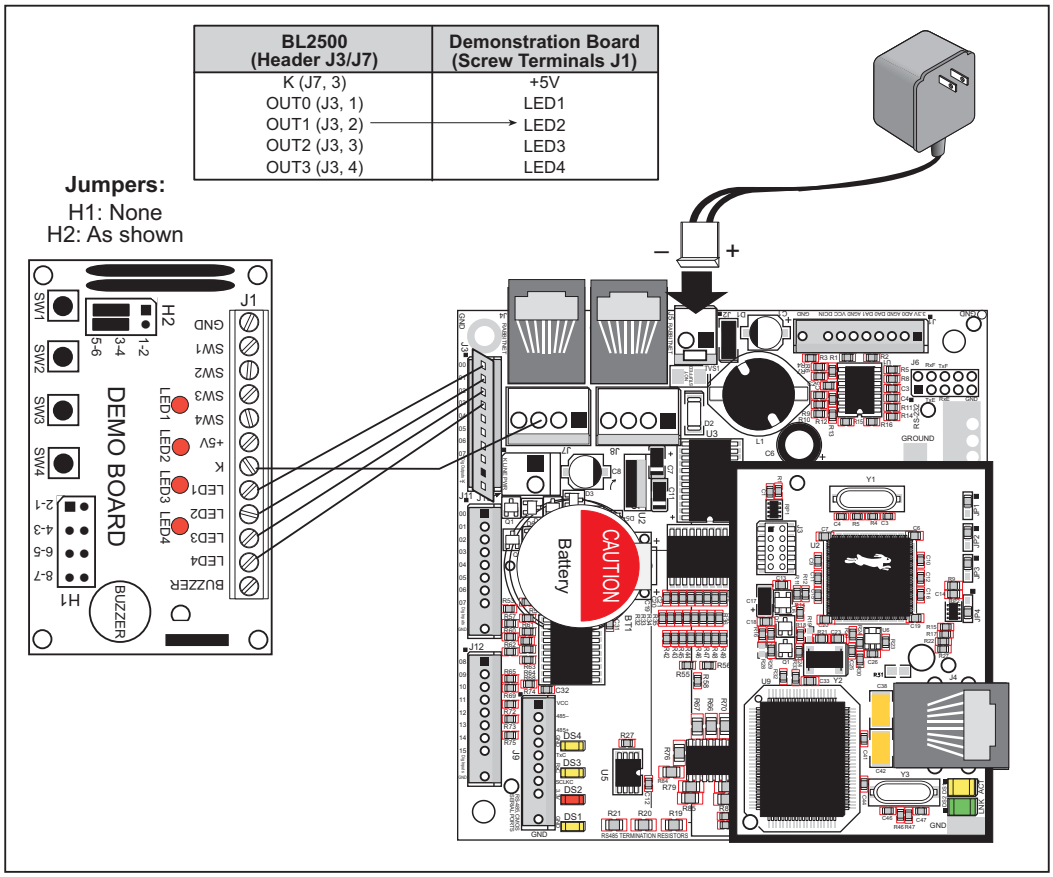


Figure C-4. Connections Between Coyote and Demonstration Board for `DIGOUT.C` Sample Program

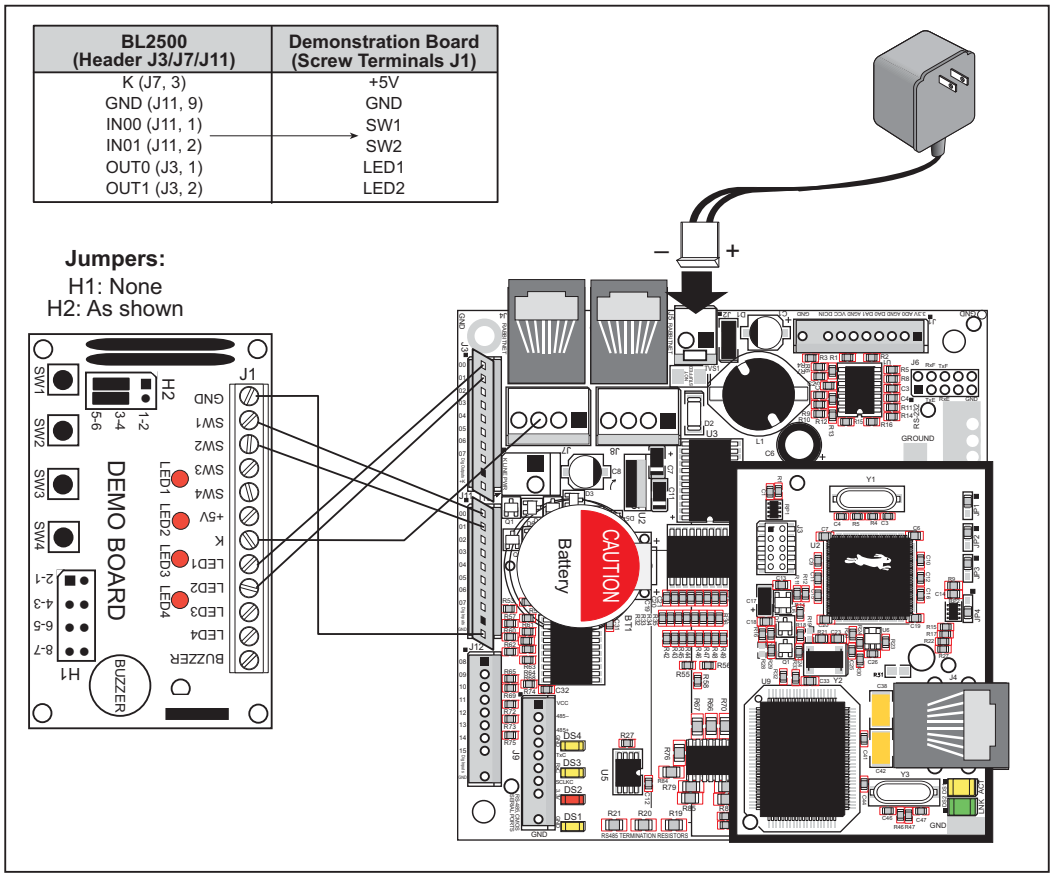


Figure C-5. Connections Between Coyote and Demonstration Board for TCP/IP* SMPT.C *Sample Program

APPENDIX D. RABBITNET

D.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.

D.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.

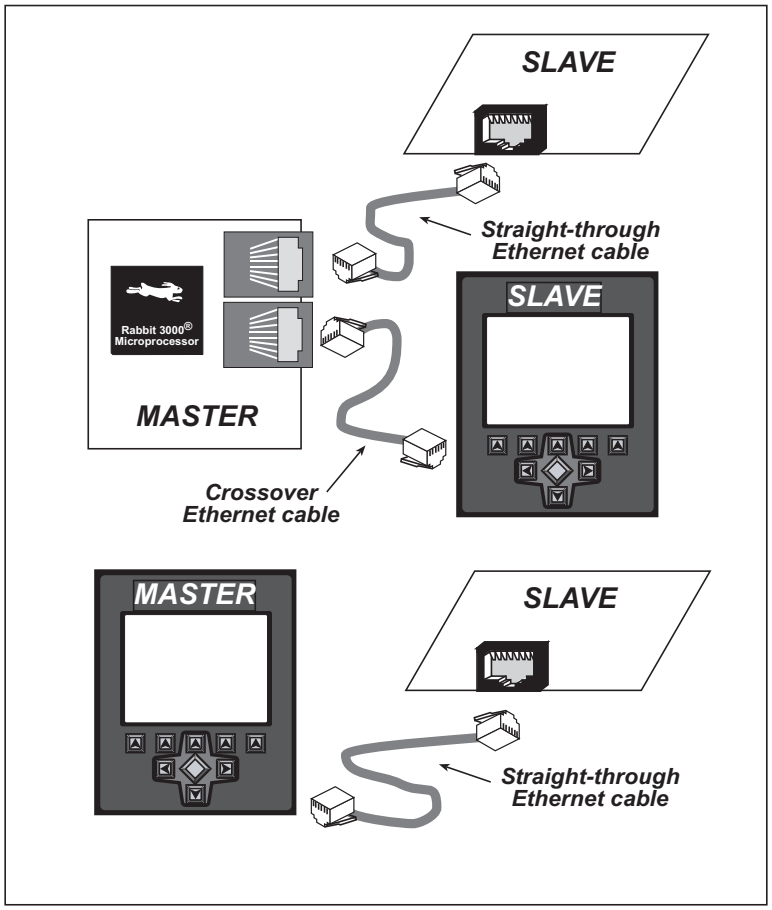


Figure D-1. Connecting Peripheral Cards to a Master

Use a straight-through 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 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.

D.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.

D.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 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.

D.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.

D.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.

int rn_init(char portflag, char servicetype);

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 is a 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

int rn_device(char pna);

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 is 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

int rn_find(rn_search *srch);

Locates the first active device that matches the search criteria.

PARAMETER

srch is the 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

int rn_echo(int handle, char sendecho, char *recdata);

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 is an address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.

sendecho is the character to echo back.

recdata is a 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.

int rn_write(int handle, int regno, char *data, int datalen);

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 is an address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.

regno is the command register number as designated by each device.

data is a pointer to the address of the string to write to the device.

datalen is the 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

int rn_read(int handle, int regno, char *recdata, int datalen);

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 is an address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.

regno is the command register number as designated by each device.

recdata is a pointer to the address of the string to read from the device.

datalen is the 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

int rn_reset(int handle, int resettype);

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 is an 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.

int rn_sw_wdt(int handle, float timeout);

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 is an address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.

timeout is a 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.

int rn_enable_wdt(int handle, int wdttype);

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 is an 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

int rn_hitwd(int handle, char *count);

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 is an address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.

count is a 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

int rn_rst_status(int handle, char *retdata);

Reads the status of which reset occurred and whether any watchdogs are enabled.

PARAMETERS

handle is an address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.

retdata is a 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.

int rn_comm_status(int handle, char *retdata);

PARAMETERS

handle is an address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.

retdata is a 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.

D.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.

INDEX

P

R

S

T

U

SCHEMATICS

090-0158 Coyote (BL2500) Schematic

www.rabbit.com/documentation/schemat/090-0158.pdf

090-0042 Demonstration Board Schematic

www.rabbit.com/documentation/schemat/090-0042.pdf

090-0128 Programming Cable Schematic

www.rabbit.com/documentation/schemat/090-0128.pdf

You may use the URL information provided above to access the latest schematics directly.

The Coyote consists of a main board with a RabbitCore module. Refer to the RabbitCore module manuals for more information on the RabbitCore modules, including their schematics.