RabbitCore RCM3300/RCM3310

C-Programmable Core Module
with Serial Flash Mass Storage and Ethernet

User's Manual

019–0121 • 050131–D

RabbitCore RCM3300/RCM3310 User's Manual

Part Number 019-0121 • 050131–D • Printed in U.S.A.

Trademarks

Rabbit and Rabbit 3000 are registered trademarks of Rabbit Semiconductor.

RabbitCore is a trademark of Rabbit Semiconductor.

Z-World is a registered trademark of Z-World Inc.

Dynamic C is a registered trademark of Z-World Inc.

xD-Picture Card is a trademark of Fuji Photo Film Co., Olympus Corporation, and Toshiba Corporation.

Z-World, Inc.

2900 Spafford Street
Davis, California 95616-6800
USA

Telephone: (530) 757-3737
Fax: (530) 757-3792

www.zworld.com

Rabbit Semiconductor

2932 Spafford Street
Davis, California 95616-6800
USA

Telephone: (530) 757-8400
Fax: (530) 757-8402

www.rabbitsemiconductor.com

TABLE OF CONTENTS

1. INTRODUCTION

The RCM3300 and RCM3310 RabbitCore modules feature a compact module that incorporates the latest revision of the powerful Rabbit 3000® microprocessor, flash memory, mass storage (serial flash), static RAM, and digital I/O ports. The RCM3300 and RCM3310 feature an integrated 10/100Base-T Ethernet port, and provide for LAN and Internet-enabled systems to be built as easily as serial-communication systems.

In addition to the features already mentioned above, the RCM3300 and RCM3310 have two clocks (main oscillator and real-time clock), reset circuitry, and the circuitry necessary for management of battery backup of the Rabbit 3000's internal real-time clock and the static RAM. Two 34-pin headers bring out the Rabbit 3000 I/O bus lines, parallel ports, and serial ports.

The RCM3300's and the RCM3310's mass-storage capabilities make them suited to running the optional Dynamic C FAT file system module and the featured remote application update where data are stored and handled using the same directory file structure commonly used on PCs.

The RCM3300 or RCM3310 receives +3.3 V power from the customer-supplied motherboard on which it is mounted. The RCM3300 and RCM3310 can interface with all kinds of CMOS-compatible digital devices through the motherboard.

The Development Kit has what you need to design your own microprocessor-based system: a complete Dynamic C software development system and a Prototyping Board that allows you to evaluate the RCM3300 or RCM3310, and to prototype circuits that interface to the RCM3300 or RCM3310 module.

1.1 RCM3300/RCM3310 Features

- Small size: 1.85" x 2.73" x 0.86"
  (47 mm x 69 mm x 22 mm)
- Microprocessor: Rabbit 3000 running at 44.2 MHz
- 49 parallel 5 V tolerant I/O lines: 43 configurable for I/O, 3 fixed inputs, 3 fixed outputs
- Three additional digital inputs, two additional digital outputs
- External reset
- Alternate I/O bus can be configured for 8 data lines and 6 address lines (shared with parallel I/O lines), plus I/O read/write
- Ten 8-bit timers (six cascadable) and one 10-bit timer with two match registers
- 512K flash memory, 512K program execution SRAM, 512K data SRAM
- Various mass-storage flash-memory options, which are required to run the optional Dynamic C FAT file system module and the featured remote application update.
- Real-time clock
- Watchdog supervisor
- Provision for customer-supplied backup battery via connections on header J4
- 10-bit free-running PWM counter and four pulse-width registers
- Two-channel Input Capture (shared with parallel I/O ports) can be used to time input signals from various port pins
- Two-channel Quadrature Decoder accepts inputs from external incremental encoder modules
- Five or six 3.3 V CMOS-compatible serial ports with a maximum asynchronous baud rate of 5.525 Mbps. Three ports are configurable as a clocked serial port (SPI), and two ports are configurable as SDLC/HDLC serial ports (shared with parallel I/O ports).
- Supports 1.15 Mbps IrDA transceiver

The RCM3360 and RCM3370 RabbitCore modules are similar to the RCM3300 and RCM3310, but they use fixed or removable NAND flash for their mass-storage flash memories instead of the fixed serial flash options of the RCM3300 and RCM3310.

Table 1 below summarizes the main features of the RCM3300 and the RCM3310 modules.

Table 1. RCM3300/RCM3310 Features

Feature	RCM3300	RCM3310
Microprocessor	Rabbit 3000 running at 44.2 MHz	
SRAM	512K program (fast SRAM) + 512K data	
Flash Memory (program)	512K	
Flash Memory (mass data storage)	8 Mbytes (serial flash)	4 Mbytes (serial flash)
Serial Ports	5 shared high-speed, 3.3 V CMOS-compatible ports: all 5 are configurable as asynchronous serial ports; 3 are configurable as a clocked serial port (SPI) and 1 is configurable as an HDLC serial port; option for second HDLC serial port at the expense of 2 clocked serial ports (SPI)	

The RCM3300 and RCM3310 are programmed over a standard PC serial port through a programming cable supplied with the Development Kit, and can also be programed through a USB port with an RS-232/USB converter, or directly over an Ethernet link using the featured remote application update or the Dynamic C download manager with or without a RabbitLink.

Appendix A provides detailed specifications for the RCM3300 and the RCM3310.

1.2 Advantages of the RCM3300 and RCM3310

- Fast time to market using a fully engineered, “ready-to-run/ready-to-program” microprocessor core.
- Competitive pricing when compared with the alternative of purchasing and assembling individual components.
- Easy C-language program development and debugging
- Program download utility (Rabbit Field Utility) and cloning board options for rapid production loading of programs.
- Generous memory size allows large programs with tens of thousands of lines of code, and substantial data storage.
- Integrated Ethernet port for network connectivity, with royalty-free TCP/IP software.
- Ideal for network-enabling security and access systems, home automation, HVAC systems, and industrial controls

1.3 Development and Evaluation Tools

1.3.1 RCM3300/RCM3310 Development Kit

The RCM3300/RCM3310 Development Kit contains the hardware you need to use your RCM3300 or RCM3310 module.

- RCM3300 module.
- Prototyping Board.
- AC adapter, 12 V DC, 1 A (included only with Development Kits sold for the North American market). A header plug leading to bare leads is provided to allow overseas users to connect their own power supply with a DC output of 8–30 V.)
- Programming cable with 10-pin header and DE9 connections, and integrated level-matching circuitry.
- ***Dynamic C*** CD-ROM, with complete product documentation on disk.
- ***Getting Started*** instructions.
- Accessory parts for use on the Prototyping Board.
- Screwdriver and Ethernet cables.
- ***Rabbit 3000 Processor Easy Reference*** poster.
- Registration card.

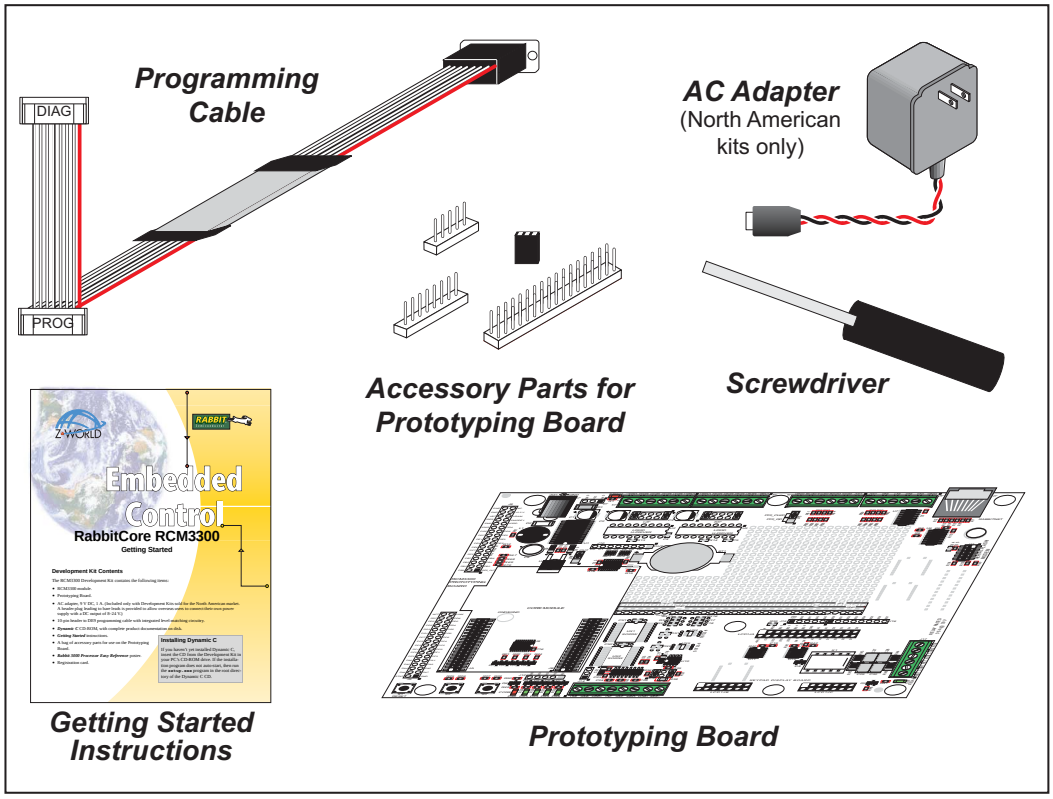


Figure 1. RCM3300/RCM3310 Development Kit

1.3.2 Software

The RCM3300 and the RCM3310 are programmed using version 8.51 or later of Z-World's Dynamic C. A compatible version is included on the Development Kit CD-ROM.

Z-World also offers for purchase add-on Dynamic C modules including the popular µC/OS-II real-time operating system, as well as point-to-point protocol (PPP), Advanced Encryption Standard (AES), FAT file system, Secure Sockets Layer (SSL), RabbitWeb, 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. Visit our Web site at www.zworld.com for further information and complete documentation for each module, or contact your Z-World sales representative or authorized distributor.

1.3.3 Connectivity Interface Kits

Z-World has available additional interface kits to allow you to provide a wireless interface to the RCM3300/RCM3310 and to use the RCM3300/RCM3310 with header sockets that have a 0.1" pitch.

- Wi-Fi Add-On Kit (Z-World Part No. 101-0997)—The Wi-Fi Add-On Kit for the RCM3000–RCM3300 footprint consists of an RCM3000–RCM3300 Interposer Board, a Wi-Fi CompactFlash card with a CompactFlash Wi-Fi Board, a ribbon interconnecting cable, and the software drivers and sample programs to help you enable your RCM3300/RCM3310 module with Wi-Fi capabilities. The RCM3000–RCM3300 Interposer Board is placed between the RCM3300/RCM3310 module and the RCM3300 Prototyping Board so that the CompactFlash Wi-Fi Board, which holds the Wi-Fi CompactFlash card, can be connected to the RCM3300/RCM3310-based system via the ribbon cable provided.
- Connector Adapter Board (Z-World Part No. 151-0114)—allows you to plug the RCM3300/RCM3310 whose headers have a 2 mm pitch into header sockets with a 0.1" pitch.

Visit our Web site at www.zworld.com or contact your Z-World sales representative or authorized distributor for further information.

1.3.4 Online Documentation

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, use your browser to find and load **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.

2. GETTING STARTED

This chapter explains how to set up and use the RCM3300/ RCM3310 modules with the accompanying Prototyping Board.

NOTE: It is assumed that you have a Development Kit. If you purchased an RCM3300 or RCM3310 module by itself, you will have to adapt the information in this chapter and elsewhere to your test and development setup.

2.1 Install Dynamic C

To develop and debug programs for the RCM3300/RCM3310 (and for all other Z-World and Rabbit Semiconductor hardware), you must install and use Dynamic C.

If you have not yet installed Dynamic C version 8.51 (or a later version), do so now by inserting the Dynamic C CD from the Development Kit in your PC's CD-ROM drive. If autorun is enabled, the CD installation will begin automatically.

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

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

Dynamic C uses a COM (serial) port to communicate with the target development system. The installation allows you to choose the COM port that will be used. The default selection is COM1. You may select any available port for Dynamic C's use. If you are not certain which port is available, select COM1. This selection can be changed later within Dynamic C.

NOTE: The installation utility does not check the selected COM port in any way. Specifying a port in use by another device (mouse, modem, etc.) may lead to a message such as **`"could not open serial port"`** when Dynamic C is started.

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

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

2.2 Hardware Connections

There are three steps to connecting the Prototyping Board for use with Dynamic C and the sample programs:

1. Attach the RCM3300/RCM3310 module to the Prototyping Board.
2. Connect the programming cable between the RCM3300/RCM3310 and the workstation PC.
3. Connect the power supply to the Prototyping Board.

2.2.1 Attach Module to Prototyping Board

Turn the RCM3300/RCM3310 module so that the Ethernet jack is facing the direction shown in Figure 2 below. Align the pins from headers J3 and J4 on the bottom side of the module into header sockets JA and JB on the Prototyping Board.

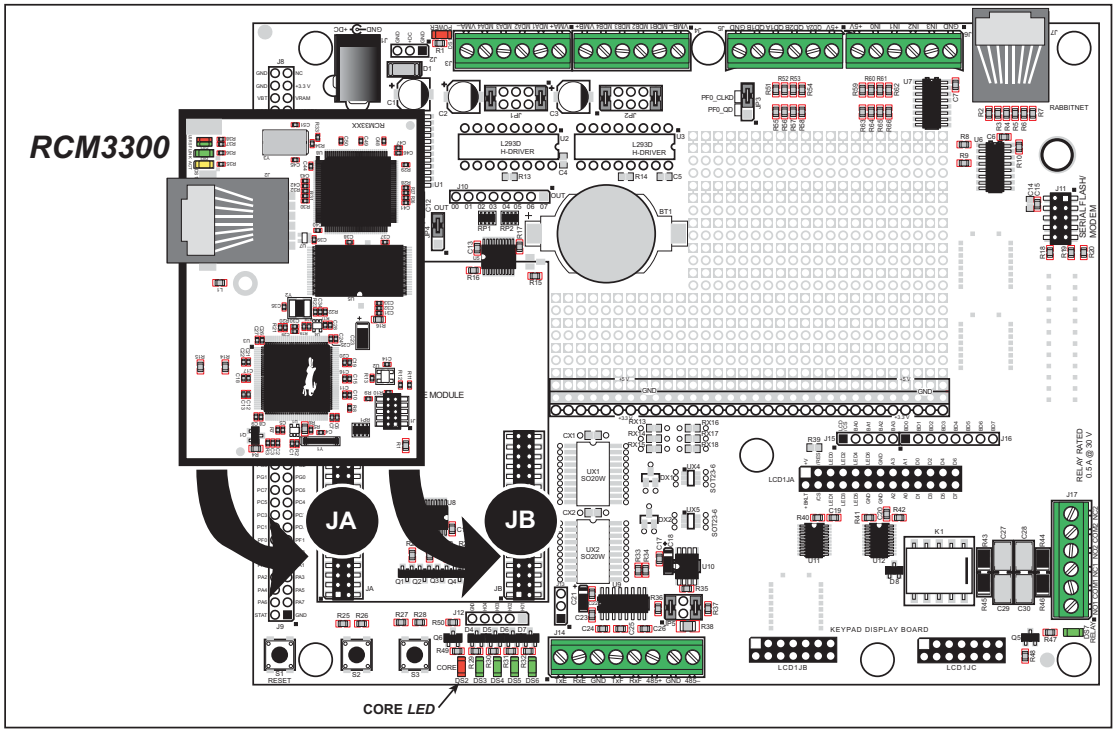


Figure 2. Install the RCM3300/RCM3310 Module on the Prototyping Board

NOTE: It is important that you line up the pins on headers J3 and J4 of the RCM3300/RCM3310 module exactly with the corresponding pins of header sockets JA and JB on the Prototyping Board. The header pins may become bent or damaged if the pin alignment is offset, and the module will not work. Permanent electrical damage to the module may also result if a misaligned module is powered up.

Press the module's pins firmly into the Prototyping Board header sockets.

2.2.2 Connect Programming Cable

The programming cable connects the RCM3300/RCM3310 to the PC running Dynamic C to download programs and to monitor the RCM3300/RCM3310 module during debugging.

Connect the 10-pin connector of the programming cable labeled **PROG** to header J1 on the RCM3300/RCM3310 as shown in Figure 3. There is a small dot on the circuit board next to pin 1 of header J1. Be sure to orient the marked (usually red) edge of the cable towards pin 1 of the connector. (Do not use the **DIAG** connector, which is used for a non-programming serial connection.)

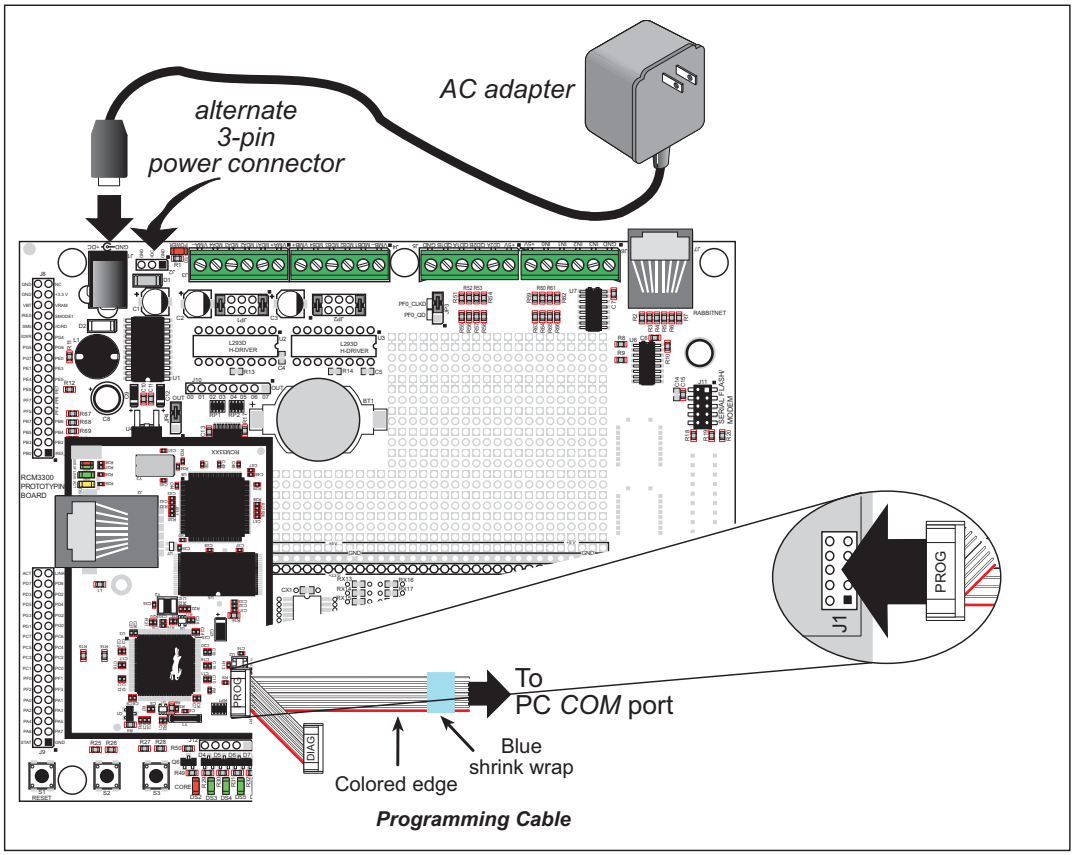


Figure 3. Connect Programming Cable and Power Supply

NOTE: Be sure to use the programming cable (part number 101-0542) supplied with this Development Kit—the programming cable has blue shrink wrap around the RS-232 converter section located in the middle of the cable. Programming cables with clear or red shrink wrap from other Z-World or Rabbit Semiconductor kits were not designed to work with RCM3300/RCM3310 modules.

Connect the other end of the programming cable to a COM port on your PC.

NOTE: Some PCs now come equipped only with a USB port. It may be possible to use an RS-232/USB converter (Z-World Part No. 540-0070) with the programming cable supplied with the Development Kit. An RS-232/USB converter is available through the Z-World Web store.

2.2.3 Connect Power

When all other connections have been made, you can connect power to the Prototyping Board. Connect the wall transformer to jack J1 on the Prototyping Board as shown in Figure 3.

Plug in the wall transformer. The core LED on the Prototyping Board should light up. The RCM3300/RCM3310 and the Prototyping Board are now ready to be used.

> **NOTE:** A **RESET** button is provided on the Prototyping Board to allow a hardware reset without disconnecting power.

2.2.3.1 Alternate Power-Supply Connections

All Development Kits include a header connector that may be used to connect your power supply to 3-pin header J2 on the Prototyping Board. The connector may be attached either way as long as it is not offset to one side—the center pin of J2 is always connected to the positive terminal, and either edge pin is negative. The power supply should deliver 8 V to 30 V DC at 8 W.

2.3 Starting Dynamic C

Once the RCM3300/RCM3310 is connected as described in the preceding pages, start Dynamic C by double-clicking on the Dynamic C icon or by double-clicking on **dcrabXXXX.exe** in the Dynamic C root directory, where **XXXX** are version-specific characters.

Dynamic C uses the serial port on your PC that you specified during installation.

If you are using a USB port to connect your computer to the RCM3300/RCM3310 module, choose **Options > Project Options** and select "Use USB to Serial Converter" on the **Communications** tab.

2.4 Run a Sample Program

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

This program shows that the CPU is working. The sample program described in Section 6.5, "Run the PINGME.C Sample Program," tests the TCP/IP portion of the board.

2.4.1 Troubleshooting

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 > Project 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 > Project Options > Communications** menu. Choose a lower debug baud rate.

If Dynamic C cannot find the target system (error message **"No Rabbit Processor Detected."**):

- Check that the RCM3300/RCM3310 is powered correctly — the red core LED on the Prototyping Board should be lit when the RCM3300/RCM3310 is mounted on the Prototyping Board and the AC adapter is plugged in.
- Check to make sure you are using the **PROG** connector, not the **DIAG** connector, on the programming cable.
- Check both ends of the programming cable to ensure that they are firmly plugged into the PC and the programming port on the RCM3300/RCM3310.

- Ensure that the RCM3300/RCM3310 module is firmly and correctly installed in its connectors on the Prototyping Board.
- Select a different COM port within Dynamic C. From the **Options** menu, select **Project Options**, then 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.

2.5 Where Do I Go From Here?

If the sample program ran fine, you are now ready to go on to other sample programs and to develop your own applications. The source code for the sample programs is provided to allow you to modify them for your own use. The ***RCM3300/RCM3310 User's Manual*** also provides complete hardware reference information and describes the software function calls for the RCM3300 and the RCM3310, the Prototyping Board, and the optional LCD/keypad module.

For advanced development topics, refer to the ***Dynamic C User's Manual*** and the ***Dynamic C TCP/IP User's Manual***, also in the online documentation set.

2.5.1 Standalone Operation of the RCM3300/RCM3310

The RCM3300/RCM3310 must be programmed via the Prototyping Board or via a similar arrangement on a customer-supplied board. Once the RCM3300/RCM3310 has been programmed successfully, remove the programming cable from the programming connector and reset the RCM3300/RCM3310. The RCM3300/RCM3310 may be reset by removing, then reapplying power, or by pressing the **RESET** button on the Prototyping Board. The RCM3300/RCM3310 module may now be removed from the Prototyping Board for end-use installation.

CAUTION: Disconnect power to the Prototyping Board or other boards when removing or installing your RCM3300/RCM3310 module to protect against inadvertent shorts across the pins or damage to the RCM3300/RCM3310 if the pins are not plugged in correctly. Do not reapply power until you have verified that the RCM3300/RCM3310 module is plugged in correctly.

2.5.2 Technical Support

NOTE: If you purchased your RCM3300/RCM3310 through a distributor or through a Z-World or Rabbit Semiconductor 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 Z-World/Rabbit Semiconductor Technical Bulletin Board at www.zworld.com/support/bb/.
- Use the Technical Support e-mail form at www.zworld.com/support/questionSubmit.shtml.

3. RUNNING SAMPLE PROGRAMS

To develop and debug programs for the RCM3300/RCM3310 (and for all other Z-World and Rabbit Semiconductor hardware), you must install and use Dynamic C.

3.1 Introduction

To help familiarize you with the RCM3300 and RCM3310 modules, Dynamic C includes several sample programs. Loading, executing and studying these programs will give you a solid hands-on overview of the RCM3300/RCM3310's capabilities, as well as a quick start using Dynamic C as an application development tool.

> **NOTE:** The sample programs assume that you have at least an elementary grasp of the C programming language. If you do not, see the introductory pages of the ***Dynamic C User's Manual*** for a suggested reading list.

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

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

1. Your RCM3300/RCM3310 must be plugged in to the Prototyping Board as described in Chapter 2, "Getting Started."
2. Dynamic C must be installed and running on your PC.
3. The programming cable must connect the programming header on the RCM3300/RCM3310 to your PC.
4. Power must be applied to the RCM3300/RCM3310 through the Prototyping Board.

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

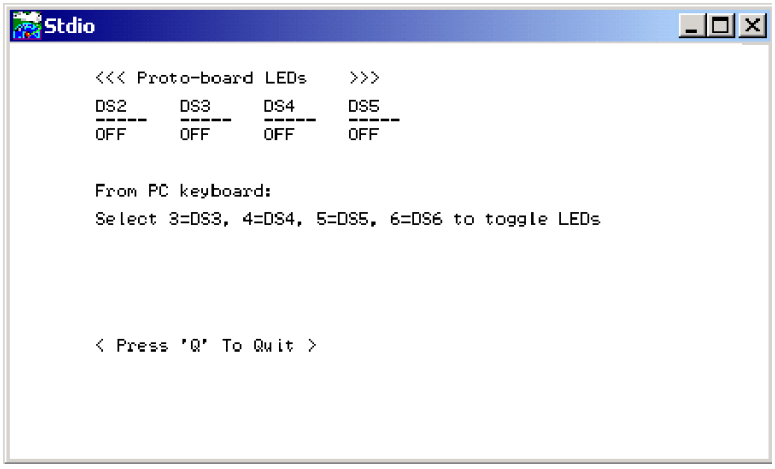
To run a sample program, open it with the **File** menu, then press function key **F9** to compile and run the program. The RCM3300/RCM3310 must be in Program Mode (see Figure 8) and must be connected to a PC using the programming cable.

3.2 Sample Programs

Of the many sample programs included with Dynamic C, several are specific to the RCM3300 and the RCM3310. Sample programs illustrating the general operation of the RCM3300/RCM3310, serial communication, and the serial flash are provided in the **SAMPLES\RCM3300** folder. Each sample program has comments that describe the purpose and function of the program. Follow the instructions at the beginning of the sample program. Note that the RCM3300/RCM3310 must be installed on the Prototyping Board when using the sample programs described in this chapter.

- **CONTROLLED.C**—Demonstrates use of the digital inputs by having you turn the LEDs on the Prototyping Board on or off from the **STDIO** window on your PC.

  Once you compile and run **CONTROLLED.C**, the following display will appear in the Dynamic C **STDIO** window.



  Press “2” or “3” or “4”or “5”on your keyboard to select LED DS3 or DS4 or DS5 or DS6 on the Prototyping Board. Then follow the prompt in the Dynamic C **STDIO** window to turn the LED on or off.

- **FLASHLED.C**—Demonstrates assembly-language program by flashing the USR LED on the RCM3300/RCM3310 and LEDs DS3, DS4, DS5, and DS6 on the Prototyping Board.
- **SWRELAY.C**—Demonstrates the relay-switching function call using the relay installed on the Prototyping Board through screw-terminal header J17.
- **TOGGLESWITCH.C**—Uses costatements to detect switches S2 and S3 using debouncing. The corresponding LEDs (DS3 and DS4) will turn on or off.

Once you have loaded and executed these five programs and have an understanding of how Dynamic C and the RCM3300/RCM3310 modules interact, you can move on and try the other sample programs, or begin building your own.

3.2.1 Use of Serial Flash

3.2.1.1 Onboard Serial Flash

The following sample programs can be found in the **SAMPLES\RCM3300\SerialFlash** folder.

- **SFLASH_INSPECT.C**—This program is a handy utility for inspecting the contents of a serial flash chip. When the sample program starts running, it attempts to initialize a serial flash chip on Serial Port B. Once a serial flash chip is found, the user can perform two different commands to either print out the contents of a specified page or clear (set to zero) all the bytes in a specified page.
- **SFLASH_LOG.C**—This program runs a simple Web server and stores a log of hits in the serial flash. This log can be viewed and cleared from a browser.

3.2.1.2 SF1000 Serial Flash Card

The following sample program can be found in the **SAMPLES\RCM3300\SF1000** folder.

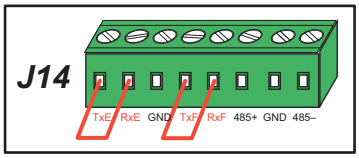
- **SERFLASHTEST.C**—An optional SF1000 Serial Flash card is required to run this demonstration. Install the Serial Flash card into socket J11 on the Prototyping Board. This sample program demonstrates how to read and write from/to the Serial Flash card.

3.2.2 Serial Communication

The following sample programs can be found in the **SAMPLES\RCM3300\SERIAL** folder.

- **FLOWCONTROL.C**—This program demonstrates hardware flow control by configuring Serial Port F for CTS/RTS with serial data coming from TxE (Serial Port E) at 115,200 bps. One character at a time is received and is displayed in the **STDIO** window.

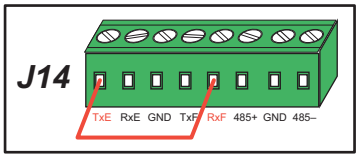
  To set up the Prototyping Board, you will need to tie TxE and RxE together on the RS-232 header at J14, and you will also tie TxF and RxF together as shown in the diagram.

  A repeating triangular pattern should print out in the **STDIO** window. The program will periodically switch flow control on or off to demonstrate the effect of no flow control.

- **PARITY.C**—This program demonstrates the use of parity modes by repeatedly sending byte values 0–127 from Serial Port E to Serial Port F. The program will switch between generating parity or not on Serial Port E. Serial Port F will always be checking parity, so parity errors should occur during every other sequence.

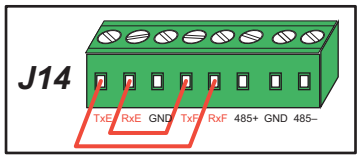
  To set up the Prototyping Board, you will need to tie TxE and RxF together on the RS-232 header at J14 as shown in the diagram.

  The Dynamic C **STDIO** window will display the error sequence.

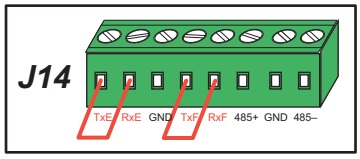
- **SIMPLE3WIRE.C**—This program demonstrates basic RS-232 serial communication. Lower case characters are sent by TxE, and are received by RxF. The characters are converted to upper case and are sent out by TxF, are received by RxE, and are displayed in the Dynamic C **STDIO** window.

  To set up the Prototyping Board, you will need to tie TxE and RxF together on the RS-232 header at J14, and you will also tie RxE and TxF together as shown in the diagram.



- **SIMPLE5WIRE.C**—This program demonstrates 5-wire RS-232 serial communication with flow control on Serial Port F and data flow on Serial Port E.

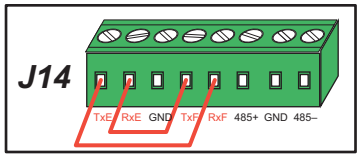
  To set up the Prototyping Board, you will need to tie TxE and RxE together on the RS-232 header at J14, and you will also tie TxF and RxF together as shown in the diagram.



  Once you have compiled and run this program, you can test flow control by disconnecting TxF from RxF while the program is running. Characters will no longer appear in the **STDIO** window, and will display again once TxF is connected back to RxF.

- **SWITCHCHAR.C**—This program transmits and then receives an ASCII string on Serial Ports E and F. It also displays the serial data received from both ports in the **STDIO** window.

  To set up the Prototyping Board, you will need to tie TxE and RxF together on the RS-232 header at J14, and you will also tie RxE and TxF together as shown in the diagram.



  Once you have compiled and run this program, press and release S2 and S3 on the Prototyping Board. The data sent between the serial ports will be displayed in the **STDIO** window.

Two sample programs, **SIMPLE485MASTER.C** and **SIMPLE485SLAVE.C**, are available to illustrate RS-485 master/slave communication. To run these sample programs, you will need a second Rabbit-based system with RS-485—another Z-World single-board computer or RabbitCore module may be used as long as you use the master or slave sample program associated with that board.

Before running either of these sample programs on the RCM3300/RCM3310 assembly, make sure pins 1–2 and pins 5–6 are jumpered together on header JP5 to use the RS-485 bias and termination resistors. The sample programs use Serial Port C as the RS-485 serial port, and they use PD7 to enable/disable the RS-485 transmitter.

The RS-485 connections between the slave and master devices are as follows.

- RS485+ to RS485+
- RS485– to RS485–
- GND to GND

- **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 and display them in the **STDIO** window. Use **SIMPLE485SLAVE.C** to program the slave.
- **SIMPLE485SLAVE.C**—This program demonstrates a simple RS-485 transmission of lower case letters to a master. The slave will send back converted upper case letters back to the master and display them in the **STDIO** window. Use **SIMPLE485MASTER.C** to program the master.

3.2.3 RabbitNet

Sample programs are available for each RabbitNet peripheral card, and can be found in the Dynamic C **SAMPLES\RabbitNet** folder. When you run any of these sample programs in conjunction with the RCM3300/RCM3310 and the Prototyping Board, you need to add the line

```
#use rcm33xx.lib
```

at the beginning of the sample program.

> **TIP:** You need to add **#use rcm33xx.lib** at the beginning of any sample program that is not in the Dynamic C **SAMPLES\RCM3300** folder.

3.2.4 Other Sample Programs

Section 6.6 describes the TCP/IP sample programs, and Appendix C.7 provides sample programs for the optional LCD/keypad module that can be installed on the Prototyping Board.

4. HARDWARE REFERENCE

Chapter 4 describes the hardware components and principal hardware subsystems of the RCM3300/RCM3310 modules. Appendix A, "RCM3300/RCM3310 Specifications," provides complete physical and electrical specifications.

Figure 4 shows the Rabbit-based subsystems designed into the RCM3300/RCM3310.

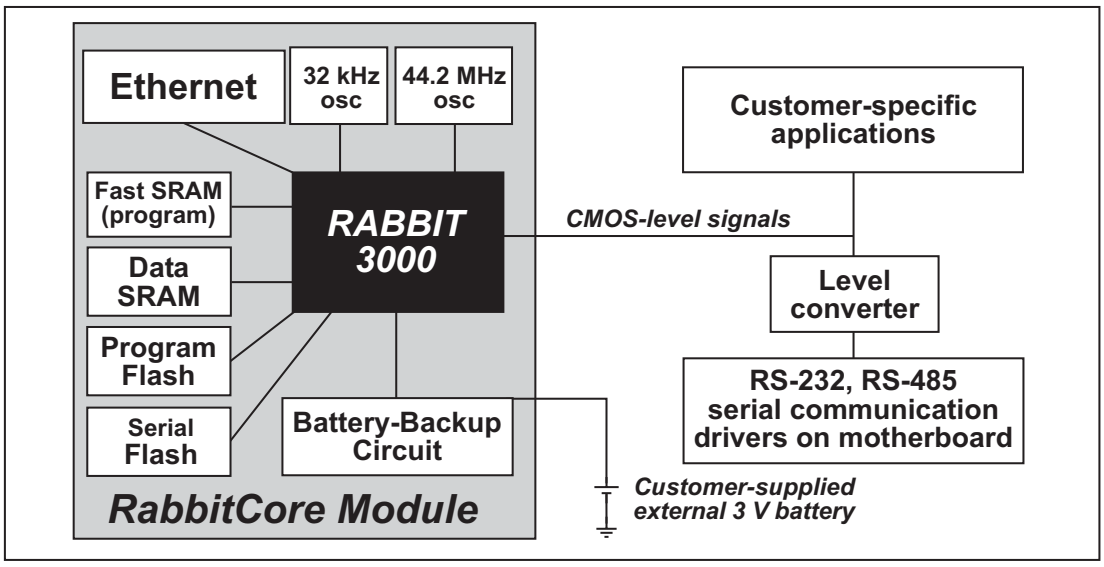


Figure 4. RCM3300/RCM3310 Subsystems

4.1 RCM3300/RCM3310 Digital Inputs and Outputs

Figure 5 shows the RCM3300/RCM3310 pinouts for headers J3 and J4.

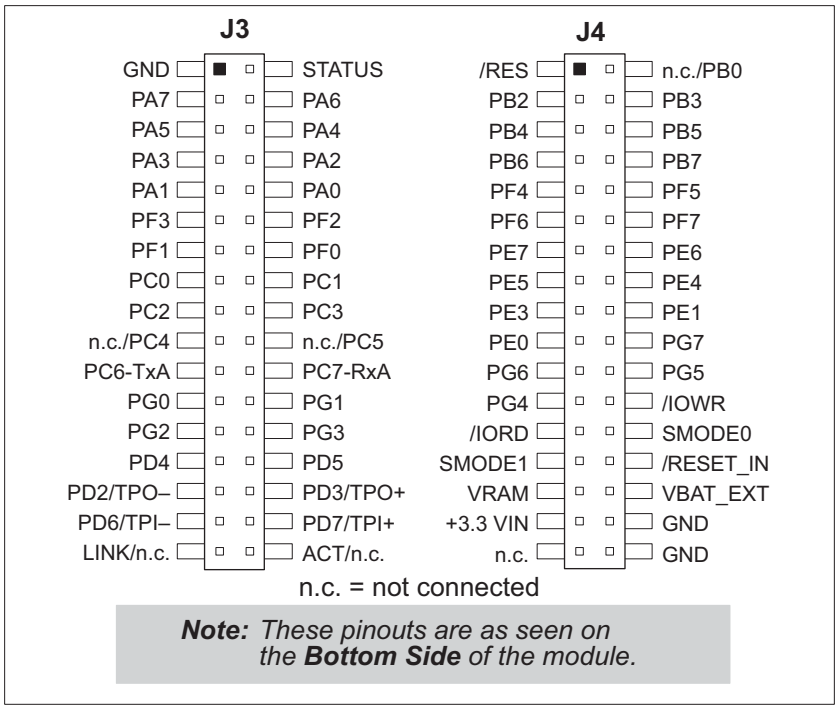


Figure 5. RCM3300/RCM3310 Pinouts

The pinouts for the RCM3000, RCM3100, RCM3200, RCM3300/RCM3310, and RCM3360/RCM3370 are almost compatible, except signals PB0, PC4, and PC5. PB0, PC4, and PC5 are used for the SPI interface to the serial flash on the RCM3300 and the RCM3310. Visit the Web site for further information.

Headers J3 and J4 are standard 2 × 34 headers with a nominal 2 mm pitch. An RJ-45 Ethernet port is also included with the RCM3300/RCM3310.

Pins 29–32 on header J3 are configured using 0 Ω resistors at locations JP4, JP5, JP6, and JP7 to be PD2, PD3, PD6, and PD7 respectively. They may also be reconfigured to carry the Ethernet signals TPI+, TPI–, TPO+, and TPO–.

Pins 33 and 34 on header J3 are wired to carry the **LINK** and **ACT** signals that illuminated the corresponding LEDs on the RCM3300/RCM3310 module. These signals may be "disconnected" by removing 0 Ω surface-mount resistors R41 and R42.

See Appendix A.5 for more information about the locations of these surface-mount resistors.

Figure 6 shows the use of the Rabbit 3000 microprocessor ports in the RCM3300/ RCM3310 modules.

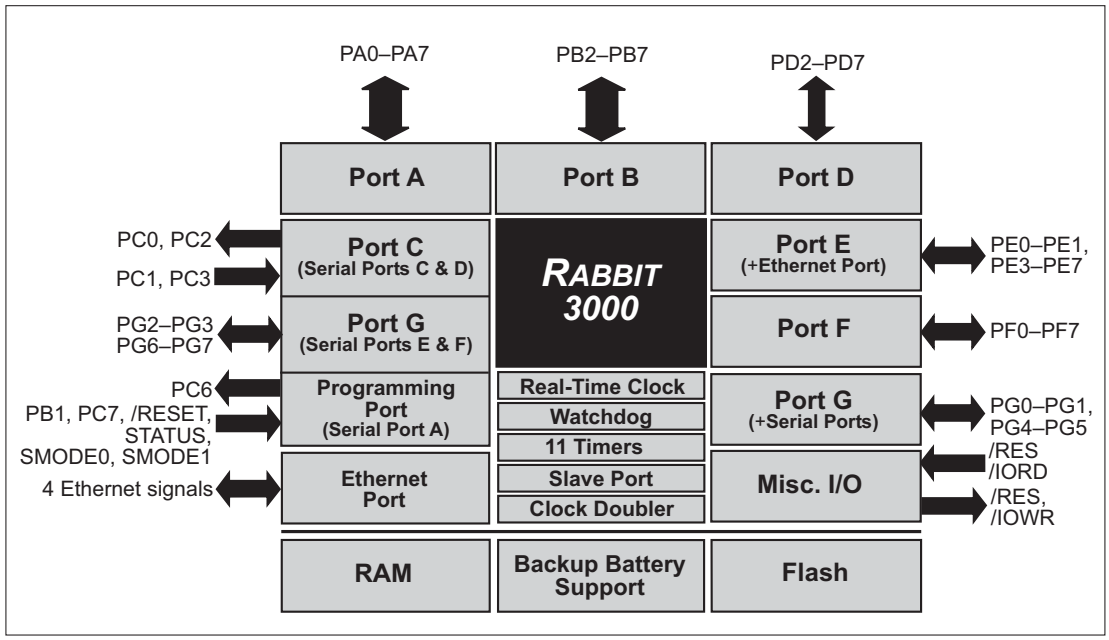


Figure 6. Use of Rabbit 3000 Ports

The ports on the Rabbit 3000 microprocessor used in the RCM3300/RCM3310 are configurable, and so the factory defaults can be reconfigured. Table 2 lists the Rabbit 3000 factory defaults and the alternate configurations.

Table 2. RCM3300/RCM3310 Pinout Configurations

Pin		Pin Name	Default Use	Alternate Use		Notes
Header J3	1	GND				
	2	STATUS	Output (Status)	Output		
	3–10	PA[7:0]	Parallel I/O	External data bus (ID0–ID7) Slave port data bus (SD0–SD7)		External Data Bus
	11	PF3	Input/Output	QD2A		
	12	PF2	Input/Output	QD2B		
	13	PF1	Input/Output	QD1A CLKC		
	14	PF0	Input/Output	QD1B CLKD		
	15	PC0	Output	TXD		Serial Port D
	16	PC1	Input	RXD		
	17	PC2	Output	TXC		Serial Port C
	18	PC3	Input	RXC		
	19	PC4	Output	TXB	Serial Port B	RCM3300/RCM3310—Not Connected (used for onboard serial flash)
	20	PC5	Input	RXB		
	21	PC6	Output	TXA		Serial Port A (programming port)
	22	PC7	Input	RXA		
	23	PG0	Input/Output	TCLKF		Serial Clock F output
	24	PG1	Input/Output	RCLKF		Serial Clock F input
	25	PG2	Input/Output	TXF		Serial Port F
	26	PG3	Input/Output	RXF		
	27	PD4	Input/Output	ATXB		
	28	PD5	Input/Output	ARXB		
	29	PD2/TPO–	Input/Output	TPOUT– *		Optional Ethernet transmit port
	30	PD3/TPO+	Input/Output	TPOUT+ *		
	31	PD6/TPI–	Input/Output	TPIN– *		Optional Ethernet receive port
	32	PD7/TPI+	Input/Output	TPIN+ *		
	33	LINK	Output			Max. sinking current draw 1 mA (see Note 1)
	34	ACT	Output			

* Pins 29–32 are configured with 0 Ω surface-mount resistors at JP4, JP5, JP7, and JP8.

Table 2. RCM3300/RCM3310 Pinout Configurations (continued)

Pin		Pin Name	Default Use	Alternate Use	Notes
Header J4	1	/RES	Reset output		Reset output from Reset Generator
	2	PB0	Input/Output	CLKB	RCM3300/RCM3310—Not Connected (used for onboard serial flash)
	3	PB2	Input/Output	IA0 /SWR	External Address 0 Slave port write
	4	PB3	Input/Output	IA1 /SRD	External Address 1 Slave port read
	5	PB4	Input/Output	IA2 SA0	External Address 2 Slave port Address 0
	6	PB5	Input/Output	IA3 SA1	External Address 3 Slave port Address 1
	7	PB6	Input/Output	IA4	External Address 4
	8	PB7	Input/Output	IA5 /SLAVEATTN	External Address 5 Slave Attention
	9	PF4	Input/Output	AQD1B PWM0	
	10	PF5	Input/Output	AQD1A PWM1	
	11	PF6	Input/Output	AQD2B PWM2	
	12	PF7	Input/Output	AQD2A PWM3	
	13	PE7	Input/Output	I7 /SCS	
	14	PE6	Input/Output	I6	
	15	PE5	Input/Output	I5 INT1B	
	16	PE4	Input/Output	I4 INT0B	
	17	PE3	Input/Output	I3	
	18	PE1	Input/Output	I1 INT1A	I/O Strobe 1 Interrupt 1A
	19	PE0	Input/Output	I0 INT0A	I/O Strobe 0 Interrupt 0A

Table 2. RCM3300/RCM3310 Pinout Configurations (continued)

Pin		Pin Name	Default Use	Alternate Use	Notes
Header J4	20	PG7	Input/Output	RXE	Serial Port E
	21	PG6	Input/Output	TXE	
	22	PG5	Input/Output	RCLKE	Serial Clock E input
	23	PG4	Input/Output	TCLKE	Serial Clock E ouput
	24	/IOWR	Output		External write strobe
	25	/IORD	Input		External read strobe
	26–27	SMODE0, SMODE1	(0,0)—start executing at address zero (0,1)—cold boot from slave port (1,0)—cold boot from clocked Serial Port A SMODE0 =1, SMODE1 = 1 Cold boot from asynchronous Serial Port A at 2400 bps (programming cable connected)		Also connected to programming cable
	28	/RESET_IN	Input		Input to Reset Generator
	29	VRAM	Output		See **Notes** below table
	30	VBAT_EXT	3 V battery Input		Minimum battery voltage 2.85 V
	31	+3.3 VIN	Power Input		3.15–3.45 V DC
	32	GND			
	33	n.c.			Reserved for future use
	34	GND			

Notes

1. When using pins 33–34 on header J3 to drive LEDs, these pins can handle a sinking current of up to 8 mA.
2. The VRAM voltage is temperature-dependent. If the VRAM voltage drops below about 1.2 V to 1.5 V, the contents of the battery-backed SRAM may be lost. If VRAM drops below 1.0 V, the 32 kHz oscillator could stop running. Pay careful attention to this voltage if you draw any current from this pin.

4.1.1 Memory I/O Interface

The Rabbit 3000 address lines (A0–A18) and all the data lines (D0–D7) are routed internally to the onboard flash memory and SRAM chips. I/0 write (/IOWR) and I/0 read (/IORD) are available for interfacing to external devices.

Parallel Port A can also be used as an external I/O data bus to isolate external I/O from the main data bus. Parallel Port B pins PB2–PB5 and PB7 can also be used as an auxiliary address bus.

When using the auxiliary I/O bus for a digital output or the LCD/keypad module on the Prototyping Board, or for any other reason, you must add the following line at the beginning of your program.

```
#define PORTA_AUX_IO    // required to enable auxiliary I/O bus
```

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.

4.1.2 Other Inputs and Outputs

Two status mode pins, SMODE0 and SMODE1, are available as inputs. The logic state of these two pins determines the startup procedure after a reset.

/RESET_IN is an external input used to reset the Rabbit 3000 microprocessor and the RCM3300/RCM3310 onboard peripheral circuits. /RES is an output from the reset circuitry that can be used to reset external peripheral devices.

4.1.3 LEDs

The RCM3300/RCM3310 has four status LEDs located beside the RJ-45 Ethernet jack: **ACT**, **LINK**, **SF** or **FM**, and **USR**.

The yellow **ACT** LED at DS1 indicates network activity.

The green **LINK** LED at DS2 indicates that the RCM3300/RCM3310 is connected to a working network.

The **SF** LED at DS3 blinks when data are being written to or read from the flash mass-storage device.

The red **USR** LED at DS3 is a user-programmable LED, which is controlled by PD0 on the Rabbit 3000's Parallel Port D. The sample program **FLASHLED.C** provided in the Dynamic C **SAMPLES\RCM3300** folder shows how to set up and use this user-programmable LED.

4.2 Serial Communication

The RCM3300/RCM3310 does not have any serial transceivers directly on the board. However, a serial interface may be incorporated into the board the RCM3300/RCM3310 is mounted on. For example, the Prototyping Board has RS-232 and RS-485 transceiver chips.

4.2.1 Serial Ports

There are six serial ports designated as Serial Ports A, B, C, D, E, and F. All six serial ports can 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.

Serial Port A is normally used as a programming port, but may be used either as an asynchronous or as a clocked serial port once the RCM3300/RCM3310 has been programmed and is operating in the Run Mode.

Serial Port B is used to communicate with the serial flash on the RCM3300/RCM3310 and is not available for other use.

Serial Ports C and D 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.

Serial Ports E and F can also be configured as HDLC serial ports. The IrDA protocol is also supported in SDLC format by these two ports.

4.2.2 Ethernet Port

Figure 7 shows the pinout for the RJ-45 Ethernet port (J2). Note that some Ethernet connectors are numbered in reverse to the order used here.

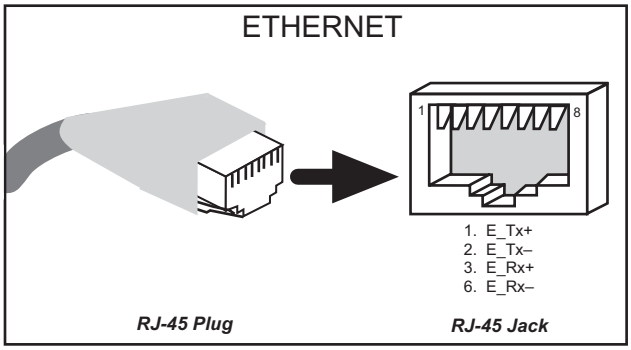


Figure 7. RJ-45 Ethernet Port Pinout

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

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

4.2.3 Programming Port

Serial Port A has special features that allow it to cold-boot the system after reset. Serial Port A is also the port that is used for software development under Dynamic C.

The RCM3300/RCM3310 is accessed using a 10-pin program header labeled J1 or through the Ethernet jack. The programming port uses the Rabbit 3000's Serial Port A for communication, and is used for the following operations.

- Programming/debugging
- Cloning
- Remote program download/debug over an Ethernet connection

When you use header J1, the Rabbit 3000 startup-mode pins (SMODE0, SMODE1) are presented to the programming port so that an externally connected device can force the RCM3300/RCM3310 to start up in an external bootstrap mode. The RCM3300/RCM3310 can be reset by Dynamic C via the **/RESET** line on header J1.

The Rabbit 3000 status pin is also presented to the programming port. The status pin is an output that can be used to send a general digital signal.

The clock line for Serial Port A is presented to the programming port, which makes synchronous serial communication possible.

The programming port is used to start the RCM3300/RCM3310 in a mode where the RCM3300/RCM3310 will download a program from the port and then execute the program. The programming port transmits information to and from a PC while a program is being debugged.

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

4.2.3.1 Alternate Uses of the Programming Port

The programming port may also be used as an application port via header J1 by using the **DIAG** connector on the programming cable. The programming port can be used as

- a synchronous serial port
- an asynchronous serial port, with the clock line usable as a general CMOS input
- two general-purpose CMOS inputs and one general-purpose CMOS output.

Two startup-mode pins, SMODE0 and SMODE1, are available as general CMOS inputs after they are read during the initial boot-up. The logic state of these two pins determines the startup procedure after a reset.

/RESET is an external input used to reset the Rabbit 3000 microprocessor and onboard peripheral circuits.

The status pin may also be used as a general-purpose CMOS output.

4.3 Programming Cable

The RCM3300/RCM3310 is placed automatically in program mode when the **PROG** connector on the programming cable is attached, and is placed automatically in run mode when no programming cable is attached.

The **DIAG** connector of the programming cable may be used on header J1 of the Prototyping Board with the RCM3300/RCM3310 operating in the run mode. This allows the programming port to be used as a regular serial port.

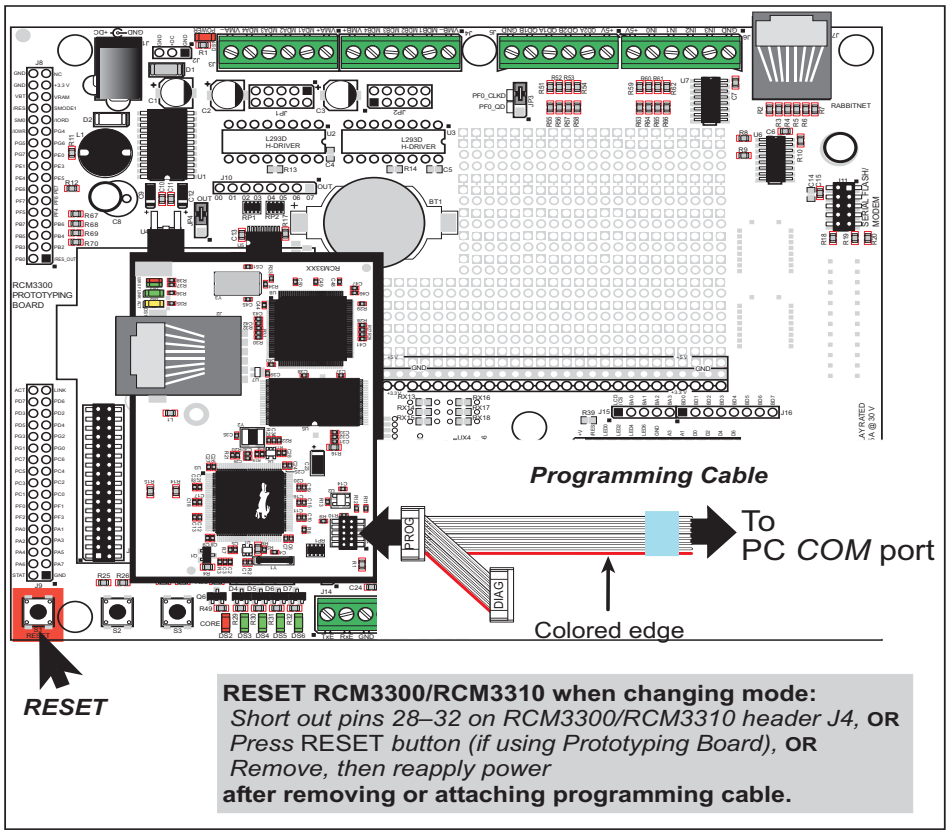


Figure 8. Switching Between Program Mode and Run Mode

4.3.1 Changing from Program Mode to Run Mode

1. Disconnect the programming cable from header J1 on the RCM3300/RCM3310.
2. Reset the RCM3300/RCM3310. You may do this as explained in Figure 8.

The RCM3300/RCM3310 is now ready to operate in the run mode.

4.3.2 Changing from Run Mode to Program Mode

1. Attach the programming cable to header J1 on the RCM3300/RCM3310.
2. Reset the RCM3300/RCM3310. You may do this as explained in Figure 8.

The RCM3300/RCM3310 is now ready to operate in the program mode.

4.4 Other Hardware

4.4.1 Clock Doubler

The RCM3300/RCM3310 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 44.2 MHz frequency specified for the RCM3300/RCM3310 is generated using a 22.12 MHz resonator.

The clock doubler may be disabled if 44.2 MHz clock speeds are not required. This will reduce power consumption and further reduce radiated emissions. The clock doubler is disabled with a simple change to the BIOS as described below.

1. Open the BIOS source code file, **RABBITBIOS.C** in the **BIOS** directory.
2. Change the line

```
#define CLOCK_DOUBLED 1 // set to 1 to double clock if
                        // Rabbit 2000: crystal <= 12.9024 MHz,
                        // Rabbit 3000: crystal <= 26.7264 MHz,
                        // or to 0 to always disable clock doubler
```

   to read as follows.

```
#define CLOCK_DOUBLED 0
```

3. Save the change using **File > Save**.

4.4.2 Spectrum Spreader

The Rabbit 3000 features a spectrum spreader, which helps to mitigate EMI problems. The spectrum spreader is on by default, but it may also be turned off or set to a stronger setting by changing the following macro in the BIOS.

```
#define ENABLE_SPREADER 1   // Set to 0 to disable spectrum spreader,
                            // 1 to enable normal spreading, or
                            // 2 to enable strong spreading.
```

NOTE: The strong spectrum-spreading setting is not recommended since it may limit the maximum clock speed or the maximum baud rate.

4.5 Memory

4.5.1 SRAM

RCM3300/RCM3310 boards have 512K of program-execution fast SRAM at U11. The program-execution SRAM is not battery-backed. There are 512K of battery-backed data SRAM installed at U10.

4.5.2 Flash EPROM

RCM3300/RCM3310 boards also have 512K of flash EPROM packaged in a 32-pin sTSOP case.

> **NOTE:** Z-World recommends that any customer applications should not be constrained by the sector size of the flash EPROM 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.

4.5.3 Serial Flash

A serial flash is supplied on the RCM3300 and the RCM3310 to store data and Web pages. Sample programs in the **SAMPLES\RCM3300** folder illustrate the use of the serial flash. These sample programs are described in Section 3.2.1, "Use of Serial Flash."

4.5.4 Dynamic C BIOS Source Files

The Dynamic C BIOS source files handle different standard RAM and flash EPROM sizes automatically.

5. SOFTWARE REFERENCE

Dynamic C is an integrated development system for writing embedded software. It runs on an IBM-compatible PC and is designed for use with Z-World controllers and other controllers based on the Rabbit microprocessor. Chapter 5 describes the libraries and function calls related to the RCM3300/RCM3310.

5.1 More About Dynamic C

Dynamic C has been in use worldwide since 1989. It is specially designed for programming embedded systems, and features quick compile and interactive debugging. A complete reference guide to Dynamic C is contained in the ***Dynamic C User's Manual***.

You have a choice of doing your software development in the flash memory or in the static SRAM included on the RCM3300/RCM3310. The flash memory and SRAM options are selected with the **Options > Program 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 compiled directly to the battery-backed data SRAM, but should be run from the program execution SRAM after the programming cable is disconnected. Your final code must always be stored in flash memory for reliable operation. RCM3300/RCM3310 modules running at 44.2 MHz have a fast program execution SRAM that is not battery-backed. Select **Code and BIOS in Flash, Run in RAM** from the Dynamic C **Options > Project Options > Compiler** menu to store the code in flash and copy it to the fast program execution SRAM at run-time to take advantage of the faster clock speed. This option optimizes the performance of RCM3300/RCM3310 modules running at 44.2 MHz.

NOTE: Do not depend on the flash memory sector size or type in your program logic. The RCM3300/RCM3310 and Dynamic C were designed to accommodate flash devices with various sector sizes in response to the volatility of the flash-memory market.

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 Z-World 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.

5.1.1 Developing Programs Remotely with Dynamic C

Dynamic C is an integrated development environment that allows you to edit, compile, and debug your programs. Dynamic C has the ability to allow programming over the Internet or local Ethernet. This is accomplished in one of two ways.

1. Via the Z-World RabbitLink, which allows a Rabbit-based target to have programs downloaded to it and debugged with the same ease as exists when the target is connected directly to a PC.
2. The RCM3300/RCM3310 has a featured remote application update written specifically to allow the RCM3300/RCM3310 to be programmed over the Internet or local Ethernet. These programs, **DLP_STATIC.C** and **DLP_WEB.C**, are available in the Dynamic C **SAMPLES\RCM3300\RemoteApplicationUpdate** folder. Complete information on the use of these programs is provided in the ***Remote Application Update*** instructions, which are available with the online documentation.

Dynamic C provides sample programs to illustrate the use of a download manager.

5.2 Dynamic C Functions

5.2.1 Digital I/O

The RCM3300/RCM3310 was designed to interface with other systems, and so there are no drivers written specifically for the I/O. The general Dynamic C read and write functions allow you to customize the parallel I/O to meet your specific needs. For example, use

```
WrPortI(PEDDR, &PEDDRShadow, 0x00);
```

to set all the Port E bits as inputs, or use

```
WrPortI(PEDDR, &PEDDRShadow, 0xFF);
```

to set all the Port E bits as outputs.

When using the auxiliary I/O bus on the Rabbit 3000 chip, add the line

```
#define PORTA_AUX_IO    // required to enable auxiliary I/O bus
```

to the beginning of any programs using the auxiliary I/O bus.

The sample programs in the Dynamic C **SAMPLES/RCM3300** folder provide further examples.

5.2.2 Serial Communication Drivers

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, allowing other functions to be performed between calls. For more information, see the ***Dynamic C Function Reference Manual*** and Technical Note TN213, ***Rabbit Serial Port Software***.

5.2.3 TCP/IP Drivers

The TCP/IP drivers are located in the **LIB\TCPIP** folder. Complete information on these libraries and the TCP/IP functions is provided in the ***Dynamic C TCP/IP User's Manual***.

5.2.4 Serial Flash Drivers

The Dynamic C **SerialFlash\SFLASH.LIB** library is used to interface to serial flash memory devices on an SPI bus such as the serial flash on board the RCM3300 and the RCM3310, which use Serial Port B as an SPI port. The library has two sets of function calls—the first is maintained for compatibility with previous versions of the **SFLASH.LIB** library. The functions are all blocking and only work for single flash devices. The new functions, which should be used for the RCM3300/RCM3310, make use of an **sf_device** structure as a handle for a specific serial flash device. This allows multiple devices to be used by an application.

More information on these function calls is available in the ***Dynamic C Function Reference Manual***.

5.2.5 Prototyping Board Functions

The functions described in this section are for use with the Prototyping Board features. The source code is in the Dynamic C **SAMPLES\RCM3300\RCM33xx.LIB** library if you need to modify it for your own board design.

The **RCM33xx.LIB** library is supported by the **RN_CFG_RCM33.LIB**—library, which is used to configure the RCM3300/RCM3310 for use with RabbitNet peripheral boards on the Prototyping Board.

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

5.2.5.1 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 Prototyping Board.

Summary of Initialization

1. I/O port pins are configured for Prototyping Board operation.
2. Unused configurable I/O are set as tied inputs or outputs.
3. External I/O are disabled.
4. The LCD/keypad module is disabled.
5. RS-485 is not enabled.
6. RS-232 is not enabled.
7. LEDs are off.
8. Ethernet select is disabled.
9. Mass-storage flash select is disabled.
10. Motor control is disabled.
11. The RabbitNet SPI interface is disabled.
12. The relay is set to normally closed positions.

RETURN VALUE

None.

5.2.5.2 Digital I/O

int digIn(int channel);

Reads the input state of inputs on Prototyping Board headers J5 and J6. Do not use this function if you configure these pins for alternate use after **brdInit()** is called.

PARAMETERS

channels is the channel number corresponding to the signal on header J5 or J6

0—IN0
1—IN1
2—IN2
3—IN3
4—QD1B
5—QD1A
6—QD2B
7—QD2A

RETURN VALUE

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

SEE ALSO

brdInit

void digOut(int channel, int value);

Writes a value to an output channel on Prototyping Board header J10. Do not use this function if you have installed the stepper motor chips at U2 and U3.

PARAMETERS

channel is output channel 0–7 (OUT00–OUT07).

value is the value (0 or 1) to output.

RETURN VALUE

None.

SEE ALSO

brdInit

5.2.5.3 Switches, LEDs, and Relay

int switchIn(int swin);

Reads the state of a switch input.

PARAMETERS

swin is the switch input to read:

2—S2
3—S3

RETURN VALUE

State of the switch input:

1 = open
0 = closed

SEE ALSO

brdInit

void ledOut(int led, int value);

Controls LEDs on the Prototyping Board and on the RCM3300/RCM3310.

PARAMETERS

led is the LED to control:

0 = red User LED on RCM3300/RCM3310
3 = DS3 on Prototyping Board
4 = DS4 on Prototyping Board
5 = DS5 on Prototyping Board
6 = DS6 on Prototyping Board

value is the value used to control the LED:

0 = off
1 = on

RETURN VALUE

None.

SEE ALSO

brdInit

void relayOut(int relay, int value);

Sets the position for the relay common contact. The default position is for normally closed contacts.

PARAMETERS

relay is the one relay (1)

value is the value used to connect the relay common contact:

0 = normally closed positions (NC1 and NC2)
1 = normally open positions (NO1 and NO2)

RETURN VALUE

None.

SEE ALSO

brdInit

5.2.5.4 Serial Communication

void ser485Tx(void);

Enables the RS-485 transmitter. Transmitted data are echoed back into the receive data buffer. The echoed data may be used as an indicator for disabling 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 are detected in the receive data buffer.

Remember to call the **serXopen()** function before running this function.

SEE ALSO

ser485Rx

void ser485Rx(void);

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

Remember to call the **serXopen()** function before running this function.

SEE ALSO

ser485Tx

5.2.5.5 RabbitNet Port

The function calls described in this section are used to configure the RabbitNet port on the Prototyping Board 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. Appendix E provides additional information about the RabbitNet.

These RabbitNet peripheral cards are available at the present time.

- Digital I/O Card (RN1100)
- A/D Converter Card (RN1200)
- D/A Converter Card (RN1300)
- Relay Card (RN1400)
- Keypad/Display Interface (RN1600)

Before using the RabbitNet port, 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 board is busy
```

void rn_sp_info();

Provides **rn_init()** with the serial port control information needed for RCM3300/RCM3310 modules.

RETURN VALUE

None.

void rn_sp_close(int port);

Deactivates the RCM3300/RCM3310 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 RCM3300/RCM3310 RabbitNet port chip 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 RCM3300/RCM3310 RabbitNet port chip select to invalidate data transfer.

PARAMETERS

portnum = 0

RETURN VALUE

None.

5.3 Upgrading Dynamic C

Dynamic C patches that focus on bug fixes are available from time to time. Check the Web sites

- www.zworld.com/support/

or

- www.rabbitsemiconductor.com/support/

for the latest patches, workarounds, and bug fixes.

5.3.1 Add-On Modules

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. Z-World offers for purchase add-on Dynamic C modules including the popular µC/OS-II real-time operating system, as well as PPP, Advanced Encryption Standard (AES), FAT file system, Secure Sockets Layer (SSL), RabbitWeb, and other select libraries.

Each Dynamic C add-on module has complete documentation and sample programs to illustrate the functionality of the software calls in the module. Visit our Web site at www.zworld.com for further information and complete documentation for each module.

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.

6. USING THE TCP/IP FEATURES

6.1 TCP/IP Connections

Programming and development can be done with the RCM3300/RCM3310 modules without connecting the Ethernet port to a network. However, if you will be running the sample programs that use the Ethernet capability or will be doing Ethernet-enabled development, you should connect the RCM3300/RCM3310 module's Ethernet port at this time.

Before proceeding you will need to have the following items.

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

A straight-through and a crossover Ethernet cable are included in both the RCM3300/RCM3310 Development Kit. Figure 9 shows how to identify the two cables based on the wires in the transparent RJ-45 connectors.

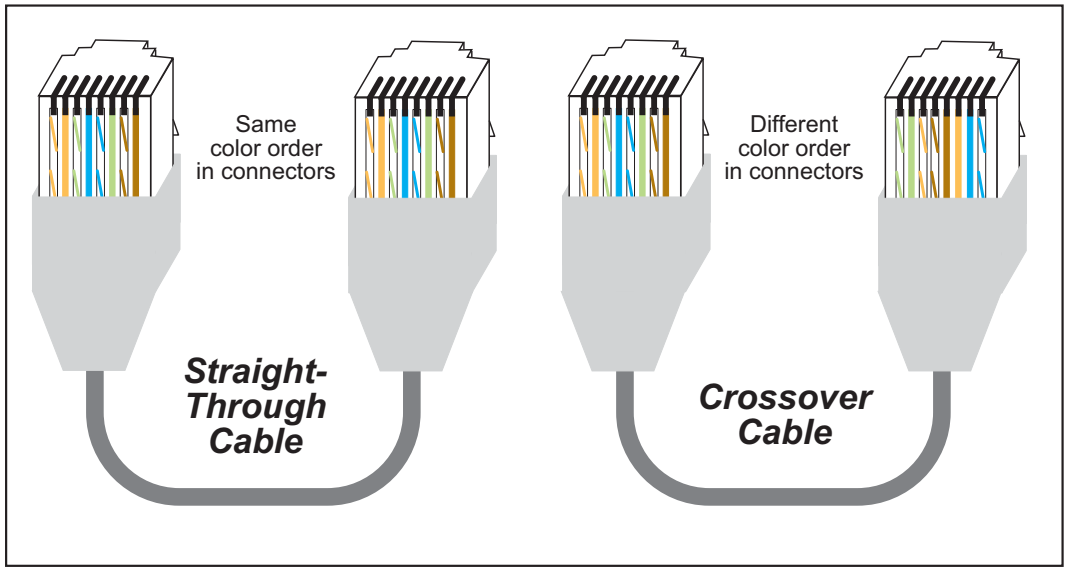


Figure 9. How to Identify Straight-Through and Crossover Ethernet Cables

Ethernet cables and a 10Base-T Ethernet hub are available from Z-World in a TCP/IP tool kit. More information is available at www.zworld.com.

Now you should be able to make your connections.

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

   There are four options for connecting the RCM3300/RCM3310 module to a network for development and runtime purposes. The first two options permit total freedom of action in selecting network addresses and use of the "network," as no action can interfere with other users. We recommend one of these options for initial development.

- **No LAN —** The simplest alternative for desktop development. Connect the RCM3300/RCM3310 module's Ethernet port directly to the PC's network interface card using an RJ-45 *crossover cable*. A crossover cable is a special cable that flips some connections between the two connectors and permits direct connection of two client systems. A standard RJ-45 network cable will not work for this purpose.
- **Micro-LAN —** Another simple alternative for desktop development. Use a small Ethernet 10Base-T hub and connect both the PC's network interface card and the RCM3300/RCM3310 module's Ethernet port to it using standard network cables.

The following options require more care in address selection and testing actions, as conflicts with other users, servers and systems can occur:

- **LAN —** Connect the RCM3300/RCM3310 module's Ethernet port to an existing LAN, preferably one to which the development PC is already connected. You will need to obtain IP addressing information from your network administrator.
- **WAN —** The RCM3300/RCM3310 is capable of direct connection to the Internet and other Wide Area Networks, but exceptional care should be used with IP address settings and all network-related programming and development. We recommend that development and debugging be done on a local network before connecting a RabbitCore system to the Internet.

> **TIP:** Checking and debugging the initial setup on a micro-LAN is recommended before connecting the system to a LAN or WAN.

The PC running Dynamic C does not need to be the PC with the Ethernet card.

3. Apply Power

   Plug in the AC adapter. The RCM3300/RCM3310 module and Prototyping Board are now ready to be used.

6.2 TCP/IP Primer on IP Addresses

Obtaining IP addresses to interact over an existing, operating, network can involve a number of complications, and must usually be done with cooperation from your ISP and/or network systems administrator. For this reason, it is suggested that the user begin instead by using a direct connection between a PC and the RCM3300/RCM3310 using an Ethernet crossover cable or a simple arrangement with a hub. (A crossover cable should not be confused with regular straight through cables.)

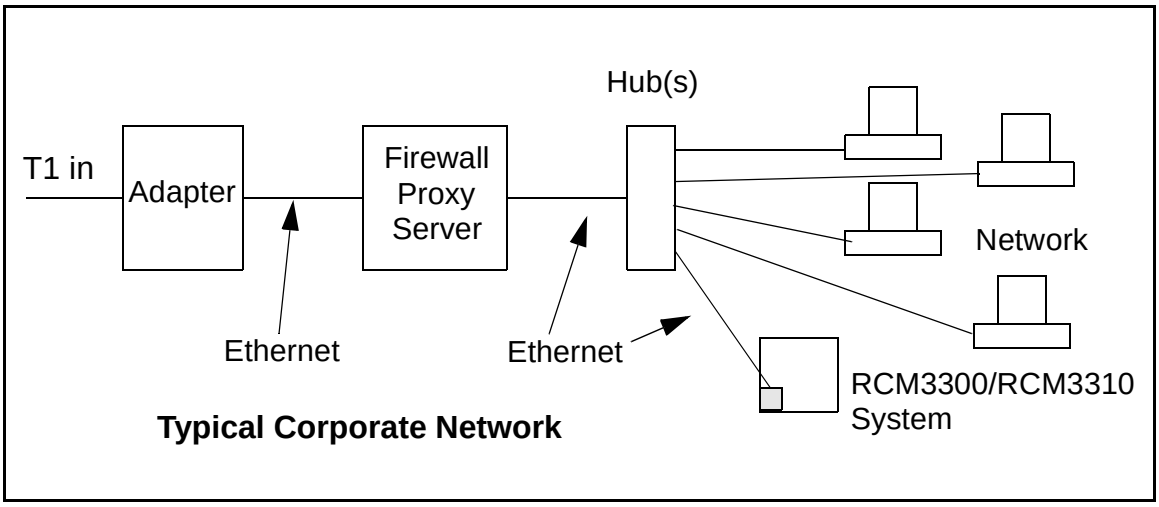
In order to set up this direct connection, the user will have to use a PC without networking, or disconnect a PC from the corporate network, or install a second Ethernet adapter and set up a separate private network attached to the second Ethernet adapter. Disconnecting your PC from the corporate network may be easy or nearly impossible, depending on how it is set up. If your PC boots from the network or is dependent on the network for some or all of its disks, then it probably should not be disconnected. If a second Ethernet adapter is used, be aware that Windows TCP/IP will send messages to one adapter or the other, depending on the IP address and the binding order in Microsoft products. Thus you should have different ranges of IP addresses on your private network from those used on the corporate network. If both networks service the same IP address, then Windows may send a packet intended for your private network to the corporate network. A similar situation will take place if you use a dial-up line to send a packet to the Internet. Windows may try to send it via the local Ethernet network if it is also valid for that network.

The following IP addresses are set aside for local networks and are not allowed on the Internet: 10.0.0.0 to 10.255.255.255, 172.16.0.0 to 172.31.255.255, and 192.168.0.0 to 192.168.255.255.

The RCM3300/RCM3310 uses a 10/100Base-T type of Ethernet connection, which is the most common scheme. The RJ-45 connectors are similar to U.S. style telephone connectors, except they are larger and have 8 contacts.

An alternative to the direct connection using a crossover cable is a direct connection using a hub. The hub relays packets received on any port to all of the ports on the hub. Hubs are low in cost and are readily available. The RCM3300/RCM3310 uses 10/100 Mbps Ethernet, so the hub or Ethernet adapter can be a 10 Mbps unit, a 100 Mbps unit, or a 10/100 Mbps unit.

In a corporate setting where the Internet is brought in via a high-speed line, there are typically machines between the outside Internet and the internal network. These machines include a combination of proxy servers and firewalls that filter and multiplex Internet traffic. In the configuration below, the RCM3300/RCM3310 could be given a fixed address so any of the computers on the local network would be able to contact it. It may be possible to configure the firewall or proxy server to allow hosts on the Internet to directly contact the controller, but it would probably be easier to place the controller directly on the external network outside of the firewall. This avoids some of the configuration complications by sacrificing some security.



Typical Corporate Network

If your system administrator can give you an Ethernet cable along with its IP address, the netmask and the gateway address, then you may be able to run the sample programs without having to setup a direct connection between your computer and the RCM3300/RCM3310. You will also need the IP address of the nameserver, the name or IP address of your mail server, and your domain name for some of the sample programs.

6.2.1 IP Addresses Explained

IP (Internet Protocol) addresses are expressed as 4 decimal numbers separated by periods, for example:

216.103.126.155

10.1.1.6

Each decimal number must be between 0 and 255. The total IP address is a 32-bit number consisting of the 4 bytes expressed as shown above. A local network uses a group of adjacent IP addresses. There are always 2^N IP addresses in a local network. The netmask (also called subnet mask) determines how many IP addresses belong to the local network. The netmask is also a 32-bit address expressed in the same form as the IP address. An example netmask is:

255.255.255.0

This netmask has 8 zero bits in the least significant portion, and this means that 2^8 addresses are a part of the local network. Applied to the IP address above (216.103.126.155), this netmask would indicate that the following IP addresses belong to the local network:

216.103.126.0

216.103.126.1

216.103.126.2

etc.

216.103.126.254

216.103.126.255

The lowest and highest address are reserved for special purposes. The lowest address (216.102.126.0) is used to identify the local network. The highest address (216.102.126.255) is used as a broadcast address. Usually one other address is used for the address of the gateway out of the network. This leaves 256 - 3 = 253 available IP addresses for the example given.

6.2.2 How IP Addresses are Used

The actual hardware connection via an Ethernet uses Ethernet adapter addresses (also called MAC addresses). These are 48-bit addresses and are unique for every Ethernet adapter manufactured. In order to send a packet to another computer, given the IP address of the other computer, it is first determined if the packet needs to be sent directly to the other computer or to the gateway. In either case, there is an Ethernet address on the local network to which the packet must be sent. A table is maintained to allow the protocol driver to determine the MAC address corresponding to a particular IP address. If the table is empty, the MAC address is determined by sending an Ethernet broadcast packet to all devices on the local network asking the device with the desired IP address to answer with its MAC address. In this way, the table entry can be filled in. If no device answers, then the device is nonexistent or inoperative, and the packet cannot be sent.

Some IP address ranges are reserved for use on internal networks, and can be allocated freely as long as no two internal hosts have the same IP address. These internal IP addresses are not routed to the Internet, and any internal hosts using one of these reserved IP addresses cannot communicate on the external Internet without being connected to a host that has a valid Internet IP address. The host would either translate the data, or it would act as a proxy.

Each RCM3300/RCM3310 RabbitCore module has its own unique MAC address, which consists of the prefix 0090C2 followed by a code that is unique to each RCM3300/RCM3310 module. For example, a MAC address might be 0090C2C002C0.

> **TIP:** You can always obtain the MAC address on your board by running the sample program **DISPLAY_MAC.C** from the **SAMPLES\TCPIP** folder.

6.2.3 Dynamically Assigned Internet Addresses

In many instances, devices on a network do not have fixed IP addresses. This is the case when, for example, you are assigned an IP address dynamically by your dial-up Internet service provider (ISP) or when you have a device that provides your IP addresses using the Dynamic Host Configuration Protocol (DHCP). The RCM3300/RCM3310 modules can use such IP addresses to send and receive packets on the Internet, but you must take into account that this IP address may only be valid for the duration of the call or for a period of time, and could be a private IP address that is not directly accessible to others on the Internet. These addresses can be used to perform some Internet tasks such as sending e-mail or browsing the Web, but it is more difficult to participate in conversations that originate elsewhere on the Internet. If you want to find out this dynamically assigned IP address, under Windows 98 you can run the **winipcfg** program while you are connected and look at the interface used to connect to the Internet.

Many networks use IP addresses that are assigned using DHCP. When your computer comes up, and periodically after that, it requests its networking information from a DHCP server. The DHCP server may try to give you the same address each time, but a fixed IP address is usually not guaranteed.

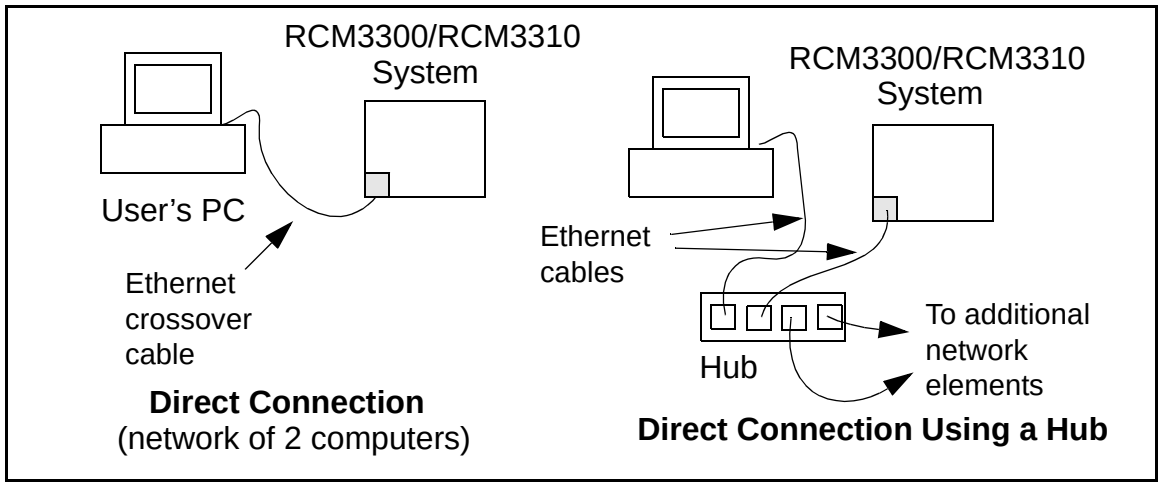
If you are not concerned about accessing the RCM3300/RCM3310 from the Internet, you can place the RCM3300/RCM3310 on the internal network using an IP address assigned either statically or through DHCP.

6.3 Placing Your Device on the Network

In many corporate settings, users are isolated from the Internet by a firewall and/or a proxy server. These devices attempt to secure the company from unauthorized network traffic, and usually work by disallowing traffic that did not originate from inside the network. If you want users on the Internet to communicate with your RCM3300/RCM3310, you have several options. You can either place the RCM3300/RCM3310 directly on the Internet with a real Internet address or place it behind the firewall. If you place the RCM3300/RCM3310 behind the firewall, you need to configure the firewall to translate and forward packets from the Internet to the RCM3300/RCM3310.

6.4 Running 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 you to connect your PC and the RCM3300/RCM3310 board 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.



6.4.1 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.

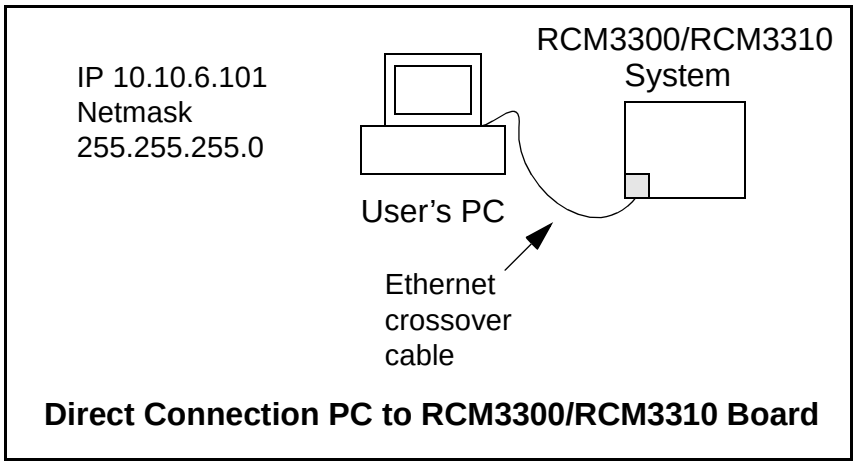
1. You can replace the **TCPCONFIG** macro with individual **MY_IP_ADDRESS**, **MY_NETMASK**, **MY_GATEWAY**, and **MY_NAMESERVER** macros in each program.
2. You can leave **TCPCONFIG** at the usual default of 1, which will set the IP configurations to **10.10.6.100**, the netmask to **255.255.255.0**, 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 RCM3300/RCM3310 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***.

6.4.2 How to Set Up your Computer's IP Address for Direct Connect

When your computer is connected directly to the RCM3300/RCM3310 module 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 RCM3300/RCM3310 Board

6.5 Run the PINGME.C Sample Program

Connect the crossover cable from your computer's Ethernet port to the RCM3300/RCM3310 board'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. The crossover cable is connected from your computer's Ethernet adapter to the RCM3300/RCM3310 board's RJ-45 Ethernet connector. When the program starts running, the green **LINK** light on the RCM3300/RCM3310 module should be on to indicate an Ethernet connection is made. (Note: If the **LNK** light does not light, you may not be using 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 pingme program:

```
ping 10.10.6.101
```

or by **Start > Run**

and typing the entry

```
ping 10.10.6.101
```

Notice that the yellow **ACT** light flashes on the RCM3300/RCM3310 module 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.

6.6 Running Additional Sample Programs With Direct Connect

The following sample programs are in the Dynamic C **SAMPLES\RCM3300\TCPIP** folder.

- **BROWSELED.C**—This program demonstrates a basic controller running a Web page. Two "device LEDs" are created along with two buttons to toggle them. Users can use their Web browser to change the status of the lights. The DS3 and DS4 LEDs on the Prototyping Board will match those 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.

- **MBOXDEMO.C**—The optional LCD/keypad module (see Appendix C) must be plugged in to the Prototyping Board when using this sample program. This program demonstrates sending e-mail messages that are then shown on the LCD/keypad module display. The keypad is used to scroll through a menu to view the messages, flip to other messages, mark messages as read, and delete messages. When a new e-mail arrives, an LED on the LCD/keypad module turns on, and then turns off once the message has been marked as read. A log of all e-mail actions is kept, and can be displayed in the Web browser. All current e-mails can also be read with the Web browser.

- **PINGLED.C**—This program demonstrates ICMP by pinging a remote host. It will flash LEDs DS3 and DS4 on the Prototyping Board when a ping is sent and received.

- **SMTP.C**—This program demonstrates using the SMTP library to send an e-mail when the S2 and S3 switches on the Prototyping Board are pressed. LEDs DS3 and DS4 on the Prototyping Board will light up when e-mail is being sent.

6.6.1 RabbitWeb Sample Programs

You will need to have the Dynamic C RabbitWeb module installed before you run the sample programs described in this section. The following sample programs are in the Dynamic C **SAMPLES\RCM3300\TCPIP\RABBITWEB** folder.

- **BLINKLEDS.C**—This program demonstrates a basic example to change the rate at which the DS3 and DS4 LEDs on the Prototyping Board blink.
- **DOORMONITOR.C**—The optional LCD/keypad module (see Appendix C) must be plugged in to the Prototyping Board when using this sample program. This program demonstrates adding and monitoring passwords entered via the LCD/keypad module.
- **SPRINKLER.C**—This program demonstrates how to schedule times for the relay and digital outputs in a 24-hour period.

6.6.2 Remote Application Update

The following programs that make up the featured application for the RCM3300/ RCM3310 can be found in the **SAMPLES\RCM3300\RemoteApplicationUpdate** folder.

- **DLP_STATIC.C**—This program uses the TCP/IP **HTTP.LIB** library, and outputs a basic static Web page.
- **DLP_WEB.C**—This program outlines a basic download program with a Web interface.

Complete information on the use of these programs is provided in the ***Remote Application Update*** instructions, which are available with the online documentation.

6.6.3 Dynamic C FAT File System, RabbitWeb, and SSL Modules

The Dynamic C FAT File System, RabbitWeb, and Secure Sockets Layer (SSL) modules have been integrated into a sample program for the RCM3300 and the RCM3310. The sample program requires that you have installed the Dynamic C FAT File System, RabbitWeb, and SSL modules.

> **TIP:** Before running any of the sample programs described in this section, you should look at and run sample programs for the TCP/IP **ZSERVER.LIB** library, the FAT file system, RabbitWeb, SSL, the download manager, and HTTP upload to become more familiar with their operation.

The **INTEGRATION.C** sample program in the **SAMPLES\RCM3300\Module_Integration** folder demonstrates the use of the TCP/IP **ZSERVER.LIB** library and FAT file system functionality with RabbitWeb dynamic HTML content, all secured using SSL. The sample program also supports dynamic updates of both the application and its resources using the Rabbit Download Manager (DLM) and HTTP upload capability, respectively—note that neither of these currently supports SSL security.

First, you need to format and partition the serial flash. Find the **FMT_DEVICE.C** sample program in the Dynamic C **SAMPLES\FileSystem** folder. Open this sample program with the **File > Open** menu, then compile and run it by pressing **F9**. **FMT_DEVICE.C** formats the mass storage device for use with the FAT file system. If the serial flash or NAND flash is already formatted, **FMT_DEVICE.C** gives you the option of erasing the mass storage flash and reformatting it with a single large partition. This erasure does not check for non-FAT partitions and will destroy *all* existing partitions.

Next, run the **INTEGRATION_FAT_SETUP.C** sample program in the Dynamic C **SAMPLES\RCM3300\Module_Integration** folder. Open this sample program with the **File > Open** menu, then compile and run it by pressing **F9**. **INTEGRATION_FAT_SETUP.C** will copy some **#ximported** files into the FAT file system.

The last step to complete before you can run the **INTEGRATION.C** sample program is to create an SSL certificate. The SSL walkthrough in the online documentation for the Dynamic C SSL module explains how to do this.

Now you are ready to run the **INTEGRATION.C** sample program in the Dynamic C **SAMPLES\RCM3300\Module_Integration** folder. Open this sample program with the **File > Open** menu, then compile and run it by pressing **F9**.

> **NOTE:** Since HTTP upload and the Dynamic C SSL module currently do not work together, compiling the **INTEGRATION.C** sample program will generate a serious warning. Ignore the warning because we are not using HTTP upload over SSL. A macro (**HTTP_UPLOAD_SSL_SUPRESS_WARNING**) is available to suppress the warning message.

Open a Web browser, and browse to the device using the IP address from the **TCP_CONFIG.LIB** library or the URL you assigned to the device. The humidity monitor will be displayed in your Web browser. This page is accessible via plain HTTP or over SSL-secured HTTPS. Click on the administrator link to bring up the admin page, which is secured automatically using SSL with a user name and a password. Use **myadmin** for user name and use **myadmin** for the password.

The admin page demonstrates some RabbitWeb capabilities and provides access to the HTTP upload page. Click the upload link to bring up the HTTP upload page, which allows you to choose new files for both the humidity monitor and the admin page. If your browser prompts you again for your user name and password, they are the same as above.

Note that the upload page is a static page included in the program flash, and can only be updated by recompiling and downloading the application. This page is protected so that you cannot accidentally change the upload page, possibly restricting yourself from performing future updates.

To try out the update capability, click the upload link on the admin page and choose a simple text file to replace **monitor.ztm**. Open another browser window and load the main page. You will see that your text file has replaced the humidity monitor. To restore the monitor, go back to the other window, click back to go to the upload page again, and choose **HUMIDITY_MONITOR.ZHTML** to replace **monitor.ztm** and click **Upload**.

When you refresh the page in your browser, you will see that the page has been restored. You have successfully updated and restored your application's files remotely!

When you are finished with the **INTEGRATION.C** sample program, you need to follow a special shutdown procedure before powering off to prevent any possible corruption of the FAT file system. Press and hold switch S2 on the Prototyping Board until LED DS3 blinks rapidly to indicate that it is now safe to turn the RCM3300/RCM3310 off. This procedure can be modified by the user to provide other application-specific shutdown tasks.

6.7 Where Do I Go From Here?

> **NOTE:** If you purchased your RCM3300/RCM3310 through a distributor or through a Z-World or Rabbit Semiconductor partner, contact the distributor or Z-World 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 Z-World/Rabbit Semiconductor Technical Bulletin Board at www.zworld.com/support/bb/.
- Use the Technical Support e-mail form at www.zworld.com/support/questionSubmit.shtml.

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

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

Please 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 CD and on Z-World's Web site.

APPENDIX A. RCM3300/RCM3310 SPECIFICATIONS

Appendix A provides the specifications for the RCM3300/ RCM3310, and describes the conformal coating.

A.1 Electrical and Mechanical Characteristics

Figure A-1 shows the mechanical dimensions for the RCM3300/RCM3310.

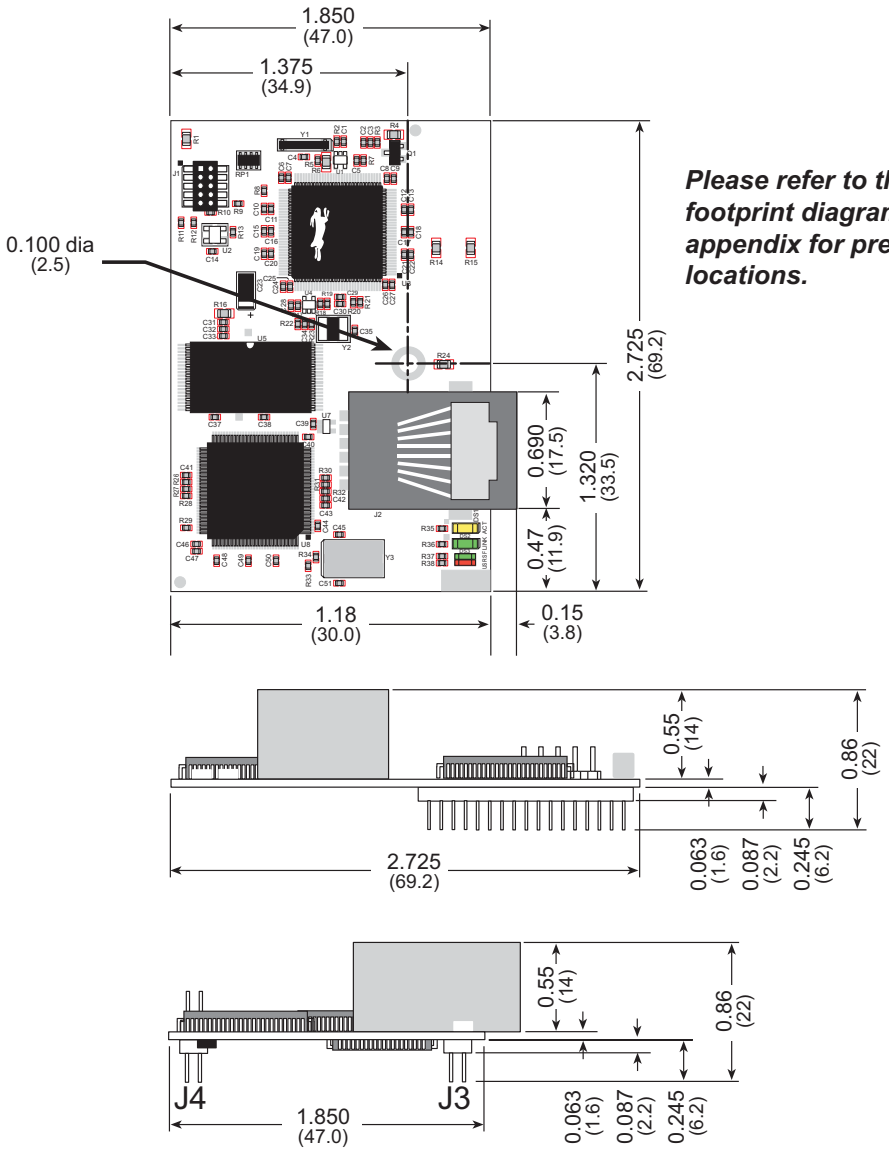


Figure A-1. RCM3300/RCM3310 Dimensions

NOTE: All measurements are in inches followed by millimeters enclosed in parentheses. All dimensions have a manufacturing tolerance of ±0.01" (0.25 mm).

It is recommended that you allow for an “exclusion zone” of 0.04" (1 mm) around the RCM3300/RCM3310 in all directions when the RCM3300/RCM3310 is incorporated into an assembly that includes other printed circuit boards. An “exclusion zone” of 0.08" (2 mm) is recommended below the RCM3300/RCM3310 when the RCM3300/RCM3310 is plugged into another assembly. Figure A-2 shows this “exclusion zone.”

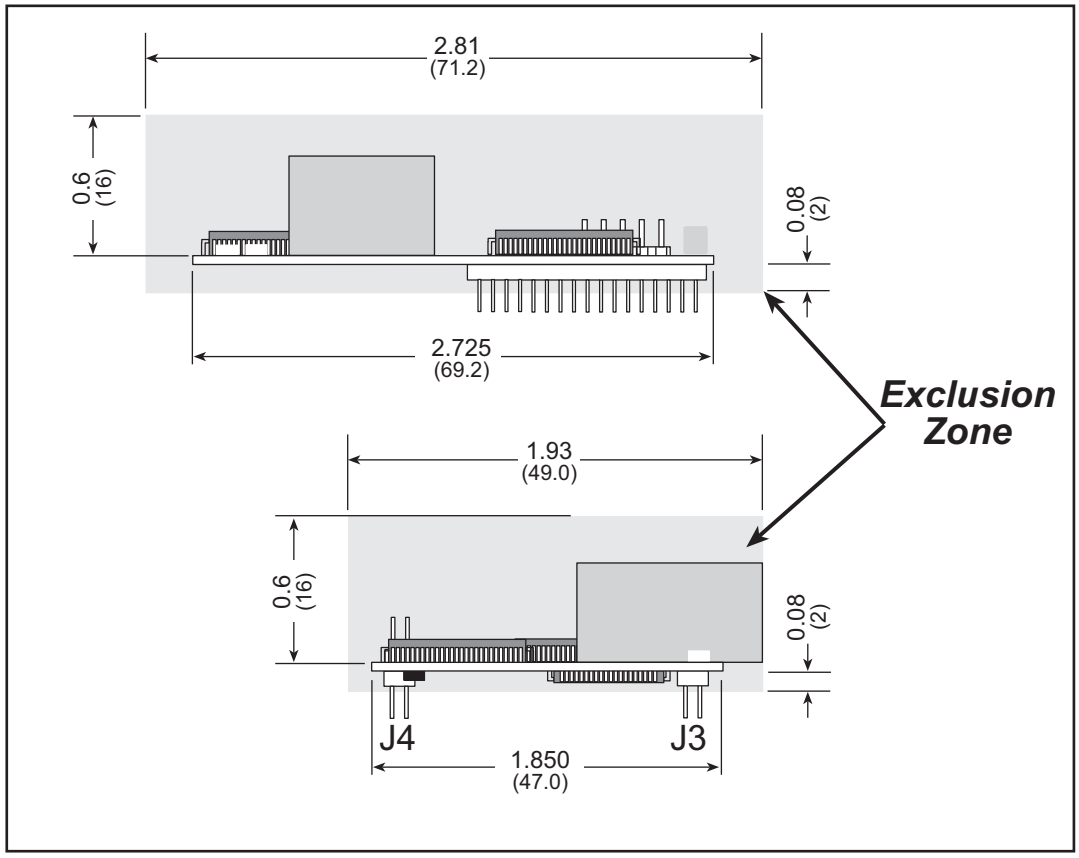


Figure A-2. RCM3300/RCM3310 “Exclusion Zone”

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

Table A-1 lists the electrical, mechanical, and environmental specifications for the RCM3300/ RCM3310.

Table A-1. RCM3300/RCM3310 Specifications

Parameter	RCM3300	RCM3310
Microprocessor	Low-EMI Rabbit 3000® at 44.2 MHz	
EMI Reduction	Spectrum spreader for reduced EMI (radiated emissions)	
Ethernet Port	10/100Base-T, RJ-45, 2 LEDs	
SRAM	512K program (fast SRAM) + 512K data	
Flash Memory (program)	512K	
Flash Memory (mass data storage)	8 Mbytes (serial flash)	4 Mbytes (serial flash)
LED Indicators	ACT (activity) LINK (link) SF (serial flash) USR (user-programmable)	
Backup Battery	Connection for user-supplied backup battery (to support RTC and data SRAM)	
General-Purpose I/O	49 parallel digital I/0 lines: • 43 configurable I/O • 3 fixed inputs • 3 fixed outputs	
Additional Inputs	Startup mode (2), reset in	
Additional Outputs	Status, reset out	
Auxiliary I/O Bus	Can be configured for 8 data lines and 5 address lines (shared with parallel I/O lines), plus I/O read/write	
Serial Ports	Five 3.3 V, CMOS-compatible ports (shared with I/O) • all 5 configurable as asynchronous (with IrDA) • 3 configurable as clocked serial (SPI) • 2 configurable as SDLC/HDLC • 1 asynchronous serial port dedicated for programming	
Serial Rate	Maximum asynchronous baud rate = CLK/8	
Slave Interface	A slave port allows the RCM3300/RCM3310 to be used as an intelligent peripheral device slaved to a master processor, which may either be another Rabbit 3000 or any other type of processor	
Real-Time Clock	Yes	
Timers	Ten 8-bit timers (6 cascadable, 3 reserved for internal peripherals), one 10-bit timer with 2 match registers	
Watchdog/ Supervisor	Yes	

Table A-1. RCM3300/RCM3310 Specifications (continued)

Parameter	RCM3300	RCM3310
Pulse-Width Modulators	4 PWM registers with 10-bit free-running counter and priority interrupts	
Input Capture	2-channel input capture can be used to time input signals from various port pins	
Quadrature Decoder	2-channel quadrature decoder accepts inputs from external incremental encoder modules	
Power	3.15–3.45 V DC 350 mA @ 44.2 MHz, 3.3 V	
Operating Temperature	–40°C to +70°C	
Humidity	5% to 95%, noncondensing	
Connectors	Two 2 × 17, 2 mm pitch one 2 × 5 for programming with 1.27 mm pitch	
Board Size	1.850" × 2.725" × 0.86" (47 mm × 69 mm × 22 mm)	

A.1.1 Headers

The RCM3300/RCM3310 uses headers at J3 and J4 for physical connection to other boards. J3 and J4 are 2 × 17 SMT headers with a 2 mm pin spacing. J1, the programming port, is a 2 × 5 header with a 1.27 mm pin spacing.

Figure A-3 shows the layout of another board for the RCM3300/RCM3310 to be plugged into. These values are relative to the designated fiducial (reference point).

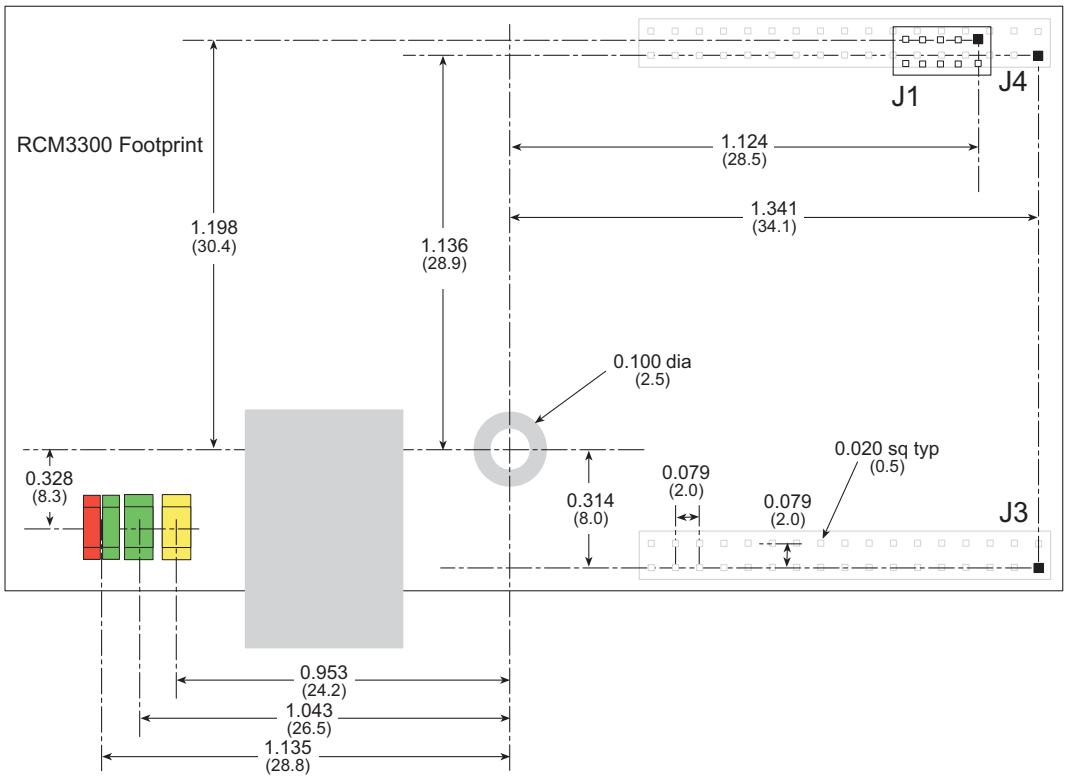


Figure A-3. User Board Footprint for RCM3300/RCM3310

A.2 Bus Loading

You must pay careful attention to bus loading when designing an interface to the RCM3300/RCM3310. This section provides bus loading information for external devices.

Table A-2 lists the capacitance for the various RCM3300/RCM3310 I/O ports.

Table A-2. Capacitance of Rabbit 3000 I/O Ports

I/O Ports	Input Capacitance (pF)	Output Capacitance (pF)
Parallel Ports A to G	12	14

Table A-3 lists the external capacitive bus loading for the various RCM3300/RCM3310 output ports. Be sure to add the loads for the devices you are using in your custom system and verify that they do not exceed the values in Table A-3.

Table A-3. External Capacitive Bus Loading -40°C to +85°C

Output Port	Clock Speed (MHz)	Maximum External Capacitive Loading (pF)
All I/O lines with clock doubler enabled	44.2	100

Figure A-4 shows a typical timing diagram for the Rabbit 3000 microprocessor external I/O read and write cycles.

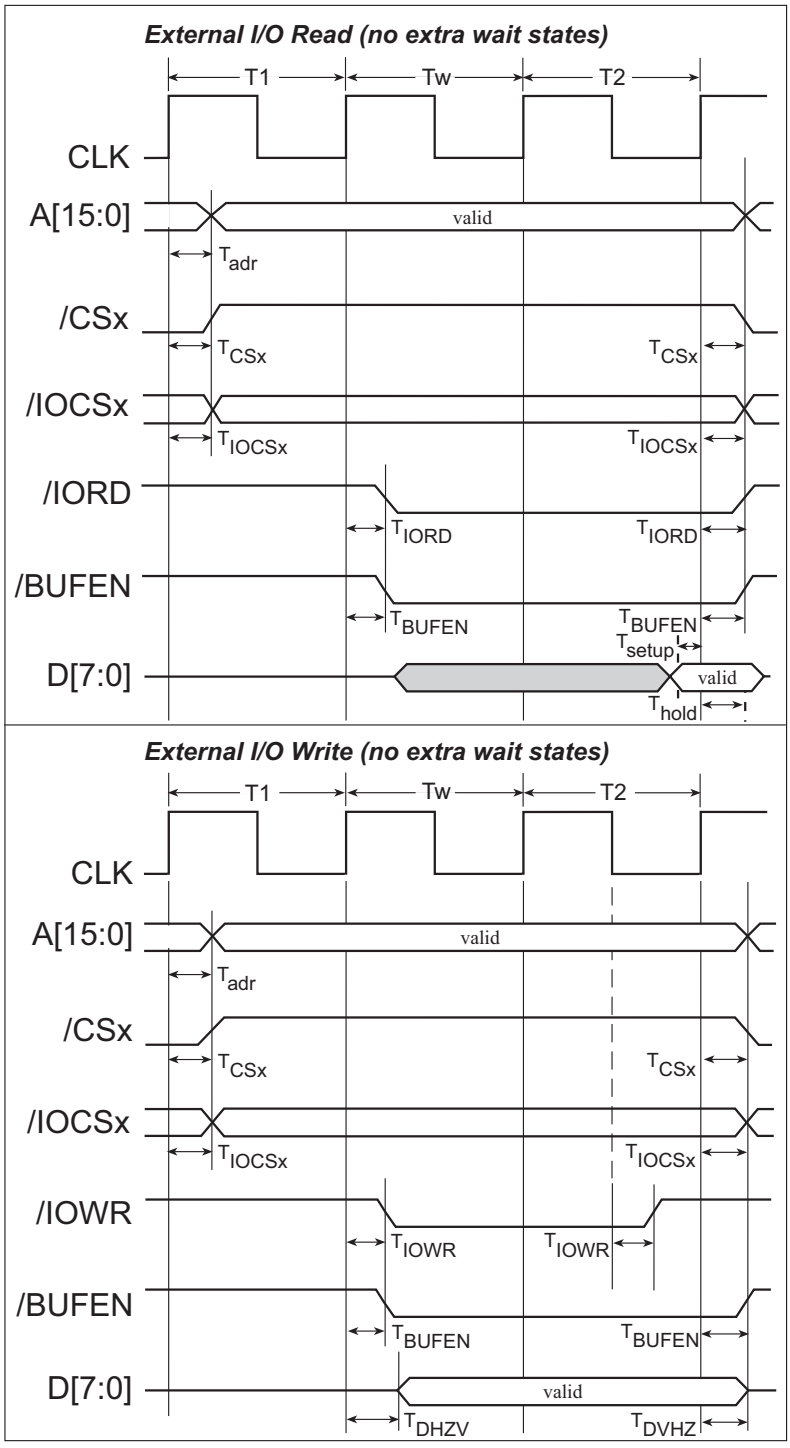


Figure A-4. I/O Read and Write Cycles—No Extra Wait States

NOTE: **/IOCSx** can be programmed to be active low (default) or active high.

Table A-4 lists the delays in gross memory access time at 3.3 V.

Table A-4. Data and Clock Delays VIN ±10%, Temp, -40°C–+85°C (maximum)

VIN	Clock to Address Output Delay (ns)			Data Setup Time Delay (ns)	Spectrum Spreader Delay (ns)	
	30 pF	60 pF	90 pF		Normal no dbl/dbl	Strong no dbl/dbl
3.3 V	6	8	11	1	3/4.5	4.5/9

The measurements are taken at the 50% points under the following conditions.

- T = -40°C to 85°C, V = V_{DD} ±10%
- Internal clock to nonloaded CLK pin delay ≤ 1 ns @ 85°C/3.0 V

The clock to address output delays are similar, and apply to the following delays.

- T_{adr}, the clock to address delay
- T_{CSx}, the clock to memory chip select delay
- T_{IOCSx}, the clock to I/O chip select delay
- T_{IORD}, the clock to I/O read strobe delay
- T_{IOWR}, the clock to I/O write strobe delay
- T_{BUFEN}, the clock to I/O buffer enable delay

The data setup time delays are similar for both T_{setup} and T_{hold}.

When the spectrum spreader is enabled with the clock doubler, every other clock cycle is shortened (sometimes lengthened) by a maximum amount given in the table above. The shortening takes place by shortening the high part of the clock. If the doubler is not enabled, then every clock is shortened during the low part of the clock period. The maximum shortening for a pair of clocks combined is shown in the table.

Technical Note TN227, ***Interfacing External I/O with Rabbit 2000/3000 Designs***, contains suggestions for interfacing I/O devices to the Rabbit 3000 microprocessors.

A.3 Rabbit 3000 DC Characteristics

Table A-5. Rabbit 3000 Absolute Maximum Ratings

Symbol	Parameter	Maximum Rating
T_A	Operating Temperature	-55° to +85°C
T_S	Storage Temperature	-65° to +150°C
	Maximum Input Voltage: • Oscillator Buffer Input • 5-V-tolerant I/O	 V_{DD} + 0.5 V 5.5 V
V_{DD}	Maximum Operating Voltage	3.6 V

Stresses beyond those listed in Table A-5 may cause permanent damage. The ratings are stress ratings only, and functional operation of the Rabbit 3000 chip at these or any other conditions beyond those indicated in this section is not implied. Exposure to the absolute maximum rating conditions for extended periods may affect the reliability of the Rabbit 3000 chip.

Table A-6 outlines the DC characteristics for the Rabbit 3000 at 3.3 V over the recommended operating temperature range from T_A = –55°C to +85°C, V_{DD} = 3.0 V to 3.6 V.

Table A-6. 3.3 Volt DC Characteristics

Symbol	Parameter	Test Conditions	Min	Typ	Max	Units
V_{DD}	Supply Voltage		3.0	3.3	3.6	V
V_{IH}	High-Level Input Voltage		2.0			V
V_{IL}	Low-Level Input Voltage				0.8	V
V_{OH}	High-Level Output Voltage	I_{OH} = 6.8 mA, V_{DD} = V_{DD} (min)	0.7 x V_{DD}			V
V_{OL}	Low-Level Output Voltage	I_{OL} = 6.8 mA, V_{DD} = V_{DD} (min)			0.4	V
I_{IH}	High-Level Input Current (absolute worst case, all buffers)	V_{IN} = V_{DD}, V_{DD} = V_{DD} (max)			10	µA
I_{IL}	Low-Level Input Current (absolute worst case, all buffers)	V_{IN} = V_{SS}, V_{DD} = V_{DD} (max)	-10			µA
I_{OZ}	High-Impedance State Output Current (absolute worst case, all buffers)	V_{IN} = V_{DD} or V_{SS}, V_{DD} = V_{DD} (max), no pull-up	-10		10	µA

A.4 I/O Buffer Sourcing and Sinking Limit

Unless otherwise specified, the Rabbit I/O buffers are capable of sourcing and sinking 6.8 mA of current per pin at full AC switching speed. Full AC switching assumes a 22.1 MHz CPU clock and capacitive loading on address and data lines of less than 100 pF per pin. The absolute maximum operating voltage on all I/O is 5.5 V.

Table A-7 shows the AC and DC output drive limits of the parallel I/O buffers when the Rabbit 3000 is used in the RCM3300/RCM3310.

Table A-7. I/O Buffer Sourcing and Sinking Capability

Pin Name	Output Drive (Full AC Switching) Sourcing/Sinking Limits (mA)	
	Sourcing	Sinking
All data, address, and I/O lines with clock doubler enabled	6.8	6.8

Under certain conditions, you can exceed the limits outlined in Table A-7. See the ***Rabbit 3000 Microprocessor User's Manual*** for additional information.

A.5 Jumper Configurations

Figure A-5 shows the jumper locations used to configure the various RCM3300/ RCM3310 options. The black square indicates pin 1.

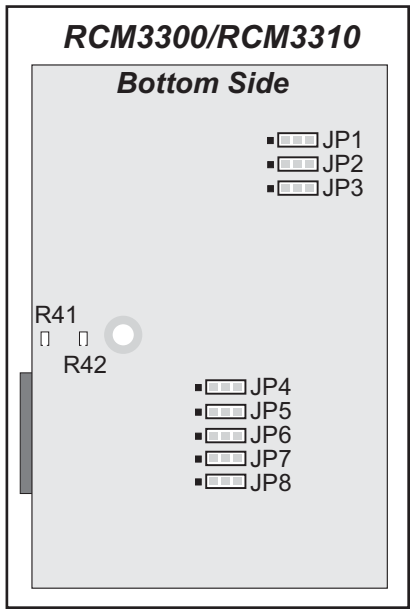


Figure A-5. Location of RCM3300/RCM3310 Configurable Positions

Table A-8 lists the configuration options.

Table A-8. RCM3300/RCM3310 Jumper Configurations

Header	Description	Pins Connected		Factory Default
JP1	Flash Memory Size	1–2	128K/256K	
		2–3	512K	✕
JP2	Flash Memory Bank Select	1–2	Reserved for future use	
		2–3	Normal Mode	✕
JP3	Data SRAM Size	1–2	128K/256K	
		2–3	512K	✕
JP4	Ethernet or I/O Output on Header J3	1–2	TPO+	
		2–3	PD3	✕
JP5	Ethernet or I/O Output on Header J3	1–2	TPO–	
		2–3	PD2	✕
JP6	Ethernet or I/O Output on Header J3	1–2	ENET_INT	
		2–3	PE0	✕
JP7	Ethernet or I/O Output on Header J3	1–2	TPI+	
		2–3	PD6	✕
JP8	Ethernet or I/O Output on Header J3	1–2	TPI–	
		2–3	PD3	✕

NOTE: The jumper connections are made using 0 Ω surface-mounted resistors.

A.6 Conformal Coating

The areas around the 32 kHz real-time clock crystal oscillator have had the Dow Corning silicone-based 1-2620 conformal coating applied. The conformally coated area is shown in Figure A-6. The conformal coating protects these high-impedance circuits from the effects of moisture and contaminants over time.

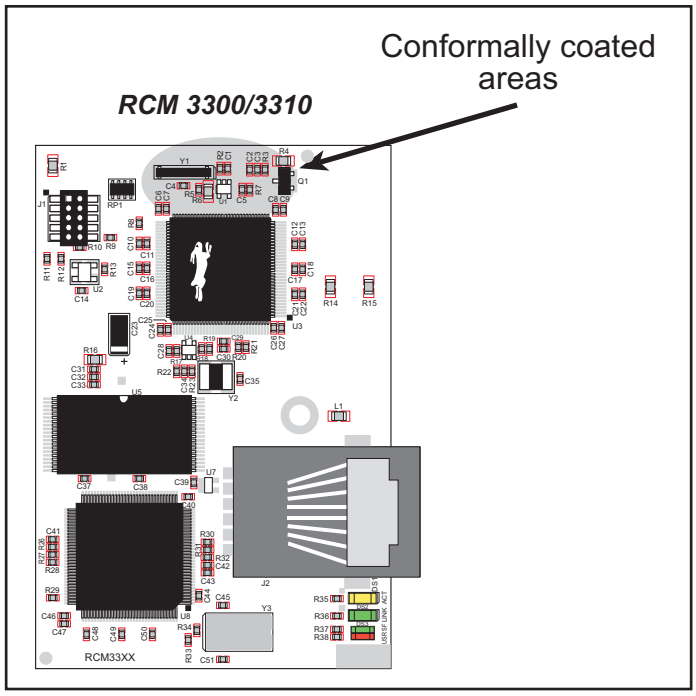


Figure A-6. RCM3300/RCM3310 Areas Receiving Conformal Coating

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***.

APPENDIX B. PROTOTYPING BOARD

Appendix B describes the features and accessories of the Prototyping Board.

B.1 Introduction

The Prototyping Board included in the Development Kit makes it easy to connect an RCM3300/RCM3310 module to a power supply and a PC workstation for development. It also provides some basic I/O peripherals (RS-232, RS-485, a relay, LEDs, and switches), as well as a prototyping area for more advanced hardware development.

For the most basic level of evaluation and development, the Prototyping Board can be used without modification.

As you progress to more sophisticated experimentation and hardware development, modifications and additions can be made to the board without modifying or damaging the RCM3300/RCM3310 module itself.

The Prototyping Board is shown below in Figure B-1, with its main features identified.

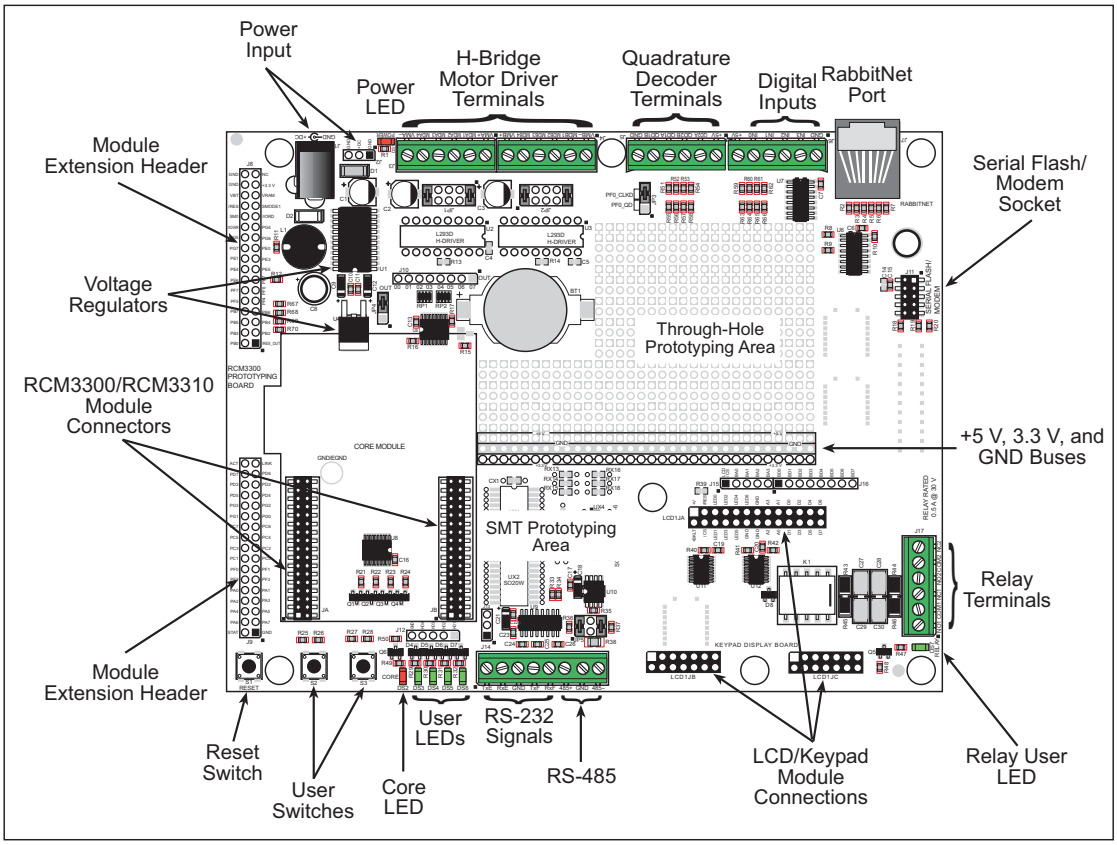


Figure B-1. Prototyping Board

B.1.1 Prototyping Board Features

- **Power Connection**—A power-supply jack and a 3-pin header are provided for connection to the power supply. Note that the 3-pin header is symmetrical, with both outer pins connected to ground and the center pin connected to the raw V+ input. The cable of the AC adapter provided with the North American version of the Development Kit ends in a plug that connects to the power-supply jack (J1). A header plug leading to bare leads is provided for overseas customers to connect their power supply to the 3-pin header (J2)—the center pin of J2 is always connected to the positive terminal, and either edge pin is negative.

  Users providing their own power supply should ensure that it delivers 8–30 V DC at 1 A.

- **Regulated Power Supply**—The raw DC voltage provided at the POWER IN jack is routed to a 5 V switching voltage regulator, then to a separate 3.3 V linear regulator. The regulators provide stable power to the RCM3300/RCM3310 module and the Prototyping Board. The voltage regulators will get warm while in use.

- **Power LED**—The power LED lights whenever power is connected to the Prototyping Board.

- **Core LED**—The core LED lights whenever an RCM3300/RCM3310 module is plugged in correctly on the Prototyping Board and the RCM3300/RCM3310 module is not being reset.

- **Relay LED**—The relay LED lights whenever the Prototyping Board relay is energized.

- **Reset Switch**—A momentary-contact, normally open switch is connected directly to the RCM3300/RCM3310's **/RESET_IN** pin. Pressing the switch forces a hardware reset of the system.

- **I/O Switches and LEDs**—Two momentary-contact, normally open switches are connected to the PG0 and PG1 pins of the RCM3300/RCM3310 module and may be read as inputs by sample applications.

  Four user LEDs (DS3–DS6) are connected to alternate I/O bus pins PA0–PA3 pins of the RCM3300/RCM3310 module via U8, and may be driven as output indicators. PE7 and PG5 control the registers in U8 as shown in the sample applications.

- **Prototyping Area**—A generous prototyping area has been provided for the installation of through-hole components. +3.3 V, +5 V, and Ground buses run along one edge of this area. Several areas for surface-mount devices are also available. Each SMT pad is connected to a hole designed to accept a 30 AWG solid wire.

- **LCD/Keypad Module**—Z-World's LCD/keypad module may be plugged in directly to headers LCD1JA, LCD1JB, and LCD1JC. The signals on headers LCD1JB and LCD1JC will be available only if the LCD/keypad module is plugged in to header LCD1JA. Appendix C provides complete information for mounting and using the LCD/keypad module.

- **Module Extension Headers**—The complete pin set of the RCM3300/RCM3310 module is duplicated at headers J8 and J9. Developers can solder wires directly into the appropriate holes, or, for more flexible development, 2 × 17 header strips with a 0.1" pitch can be soldered into place. See Figure B-4 for the header pinouts.
- **Digital I/O**—Four digital inputs are available on screw-terminal header J6. See Figure B-4 for the header pinouts.
- **RS-232**—Two 3-wire serial ports or one 5-wire RS-232 serial port are available on the Prototyping Board at screw-terminal header J14.
- **RS-485**—One RS-485 serial port is available on the Prototyping Board at screw-terminal header J14.
- **Quadrature Decoder**—Four quadrature decoder outputs (PF0–PF3) from the Rabbit 3000 chip are available on screw-terminal header J5. See Figure B-4 for the header pinouts.
- **H-Bridge Motor Driver**—Two pairs of H-bridge motor drivers are supported using screw-terminal headers J3 and J4 on the Prototyping Board for stepper-motor control. See Figure B-4 for the header pinouts.
- **RabbitNet Port**—One RS-422 RabbitNet port (shared with the serial flash interface) is available to allow RabbitNet peripheral cards to be used with the Prototyping Board.
- **Serial Flash Interface**—One serial flash/modem interface (shared with the RabbitNet port) is available to allow Z-World's SF1000 series serial flash to be used on the Prototyping Board.

B.2 Mechanical Dimensions and Layout

Figure B-2 shows the mechanical dimensions and layout for the Prototyping Board.

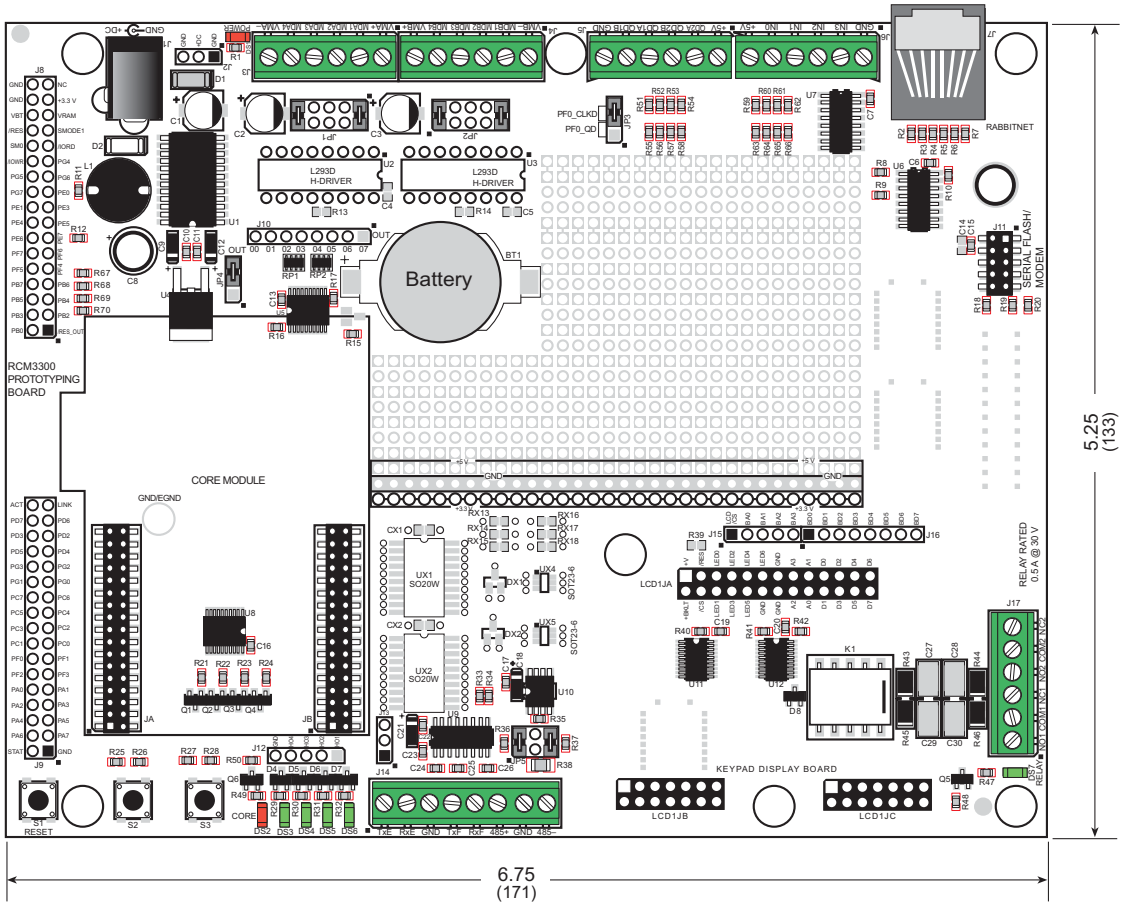


Figure B-2. Prototyping Board Dimensions

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

Table B-1 lists the electrical, mechanical, and environmental specifications for the Prototyping Board.

Table B-1. Prototyping Board Specifications

Parameter	Specification
Board Size	5.25" × 6.75" × 1.00" (133 mm × 171 mm × 25 mm)
Operating Temperature	–20°C to +70°C
Humidity	5% to 95%, noncondensing
Input Voltage	8 V to 30 V DC
Maximum Current Draw (including user-added circuits)	800 mA max. for +3.3 V supply, 1 A total +3.3 V and +5 V combined
Backup Battery	CR2032, 3 V lithium coin-type
Digital Inputs	4 inputs pulled up, ± 36 V DC, switching threshold 0.9–2.3 V typical
Digital Outputs	4 sinking outputs,+30 V DC, 500 mA maximum per channel 8 CMOS-level outputs if stepper motor not installed
Relay	SPDT relay, 500 mA @ 30 V
Serial Ports	• two 3-wire RS-232 *or* one RS-232 with RTS/CTS • one RS-485
Other Serial Interfaces	RabbitNet RS-422 port *or* serial flash/modem interface
Other Interfaces	• stepper motor control • quadrature decoder • LCD/keypad module
LEDs	Seven LEDs • one power on indicator • one RCM3300/RCM3310 module indicator • four user-configurable LEDs • one relay indicator
Prototyping Area	Throughhole, 0.1" spacing, additional space for SMT components
Connectors	• two 2 × 17, 2 mm pitch sockets for RCM3300/RCM3310 module • one 2 × 5, 2 mm pitch socket for serial flash/modem module • six screw-terminal headers for serial ports, digital inputs, stepper motor control, quadrature decoder, and relay contacts • one RJ-45 RabbitNet jack
Standoffs/Spacers	7, accept 4-40 x 1/2 screws

B.3 Power Supply

The RCM3300/RCM3310 requires a regulated 3.15 V to 3.45 V DC power source to operate. Depending on the amount of current required by the application, different regulators can be used to supply this voltage.

The Prototyping Board 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 on the Prototyping Board.

The Prototyping Board itself is protected against reverse polarity by a diode at D1 as shown in Figure B-3.

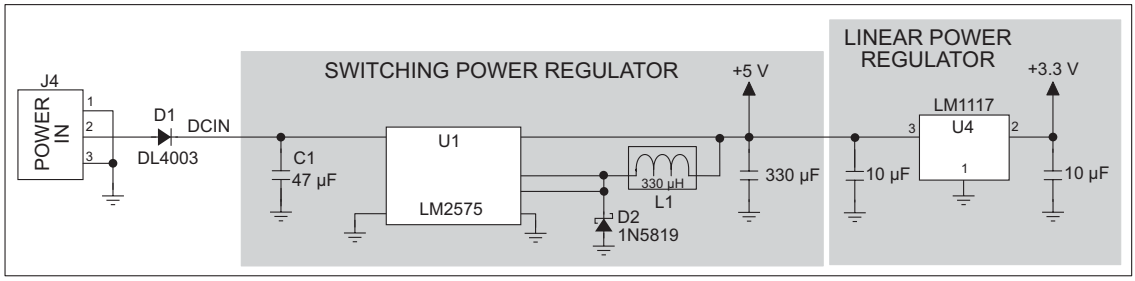


Figure B-3. Prototyping Board Power Supply

B.4 Using the Prototyping Board

The Prototyping Board is actually both a demonstration board and a prototyping board. As a demonstration board, it can be used with the sample programs to demonstrate the functionality of the RCM3300/RCM3310 right out of the box without any modifications.

The Prototyping Board pinouts are shown in Figure B-4.

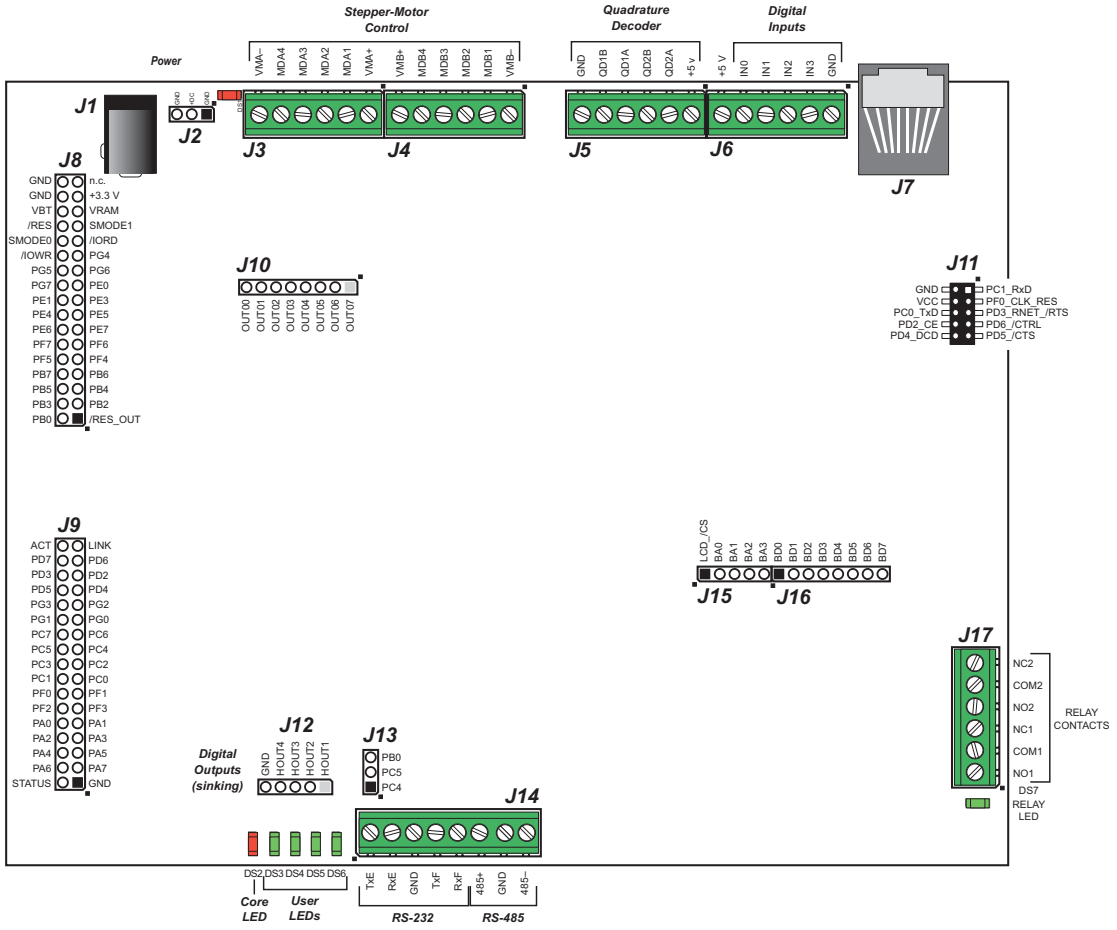


Figure B-4. Prototyping Board Pinout

The Prototyping Board comes with the basic components necessary to demonstrate the operation of the RCM3300/RCM3310. Four user LEDs (DS3–DS6) are connected to alternate I/O bus pins PA0–PA3 pins of the RCM3300/RCM3310 module via U8, and may be driven as output indicators when controlled by PE7 and PG5 as shown in the sample applications. Two switches (S2 and S3) are connected to PG0 and PG1 to demonstrate the interface to the Rabbit 3000 microprocessor. Reset switch S1 is the hardware reset for the RCM3300/RCM3310.

The Prototyping Board provides the user with RCM3300/RCM3310 connection points brought out conveniently to labeled points at J8 and J9 on the Prototyping Board. Although locations J8 and J9 are unstuffed, 2 × 17 headers are included in the bag of parts.

RS-232 and RS-485 signals are available on screw-terminal header J14, quadrature decoder outputs are available on screw-terminal header J5, and digital inputs are available on screw-terminal header J6. A 1 × 5 header strip from the bag of parts may be installed at J12 for four sinking digital outputs. A 1 × 3 header strip from the bag of parts may be installed at J13 to access selected signals from RCM3000, RCM3100, RCM3200, RCM3300/RCM3310, and RCM3360/RCM3370 RabbitCore modules (J13 cannot be used with the RCM3300/RCM3310 and the RCM3360/RCM3370).

If you don’t plan to use the LCD/keypad module, additional signals may be brought out on 1 × 5 and 1 × 8 headers from the bag of parts that you install at J15 and J16. If you don’t plan to use the stepper-motor control option, additional CMOS outputs are available via a 1 × 8 header that you install at J10.

There is a through-hole prototyping space available on the Prototyping Board. The holes in the prototyping area are spaced at 0.1" (2.5 mm). +3.3 V, +5 V, and GND traces run along one edges of the prototyping area. Small to medium circuits can be prototyped using point-to-point wiring with 20 to 30 AWG wire between the prototyping area, the +3.3 V, +5 V, and GND traces, and the surrounding area where surface-mount components may be installed. Small holes are provided around the surface-mounted components that may be installed around the prototyping area.

B.4.1 Adding Other Components

There are two sets of pads for 6-pin, 16-pin, and 28-pin devices that can be used for surface-mount prototyping devices. There are also pads that can be used for SMT resistors and capacitors in an 0805 SMT package. Each component has every one of its pin pads connected to a hole in which a 30 AWG wire can be soldered (standard wire wrap wire can be soldered in for point-to-point wiring on the Prototyping Board). Because the traces are very thin, carefully determine which set of holes is connected to which surface-mount pad.

B.4.2 Digital I/O

B.4.2.1 Digital Inputs

The Prototyping Board has four digital inputs, IN0–IN3, each of which is protected over a range of –36 V to +36 V. The inputs are pulled up to +3.3 V as shown in Figure B-5.

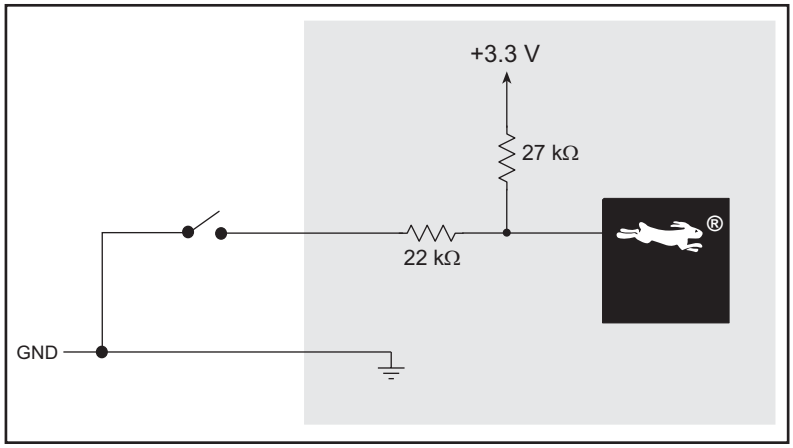


Figure B-5. Prototyping Board Digital Inputs

The actual switching threshold is between 0.9 V and 2.3 V. Anything below this value is a logic 0, and anything above is a logic 1.

The digital inputs are each fully protected over a range of -36 V to +36 V, and can handle short spikes of ±40 V.

B.4.3 CMOS Digital Outputs

If the stepper-motor option is not used, eight CMOS-level digital outputs are available at J10, and can each handle up to 25 mA.

B.4.4 Sinking Digital Outputs

Four sinking digital outputs shared with LEDs DS3–DS6 are available at J12, and can each handle up to 500 mA. Figure B-6 shows a wiring diagram for a typical sinking output.

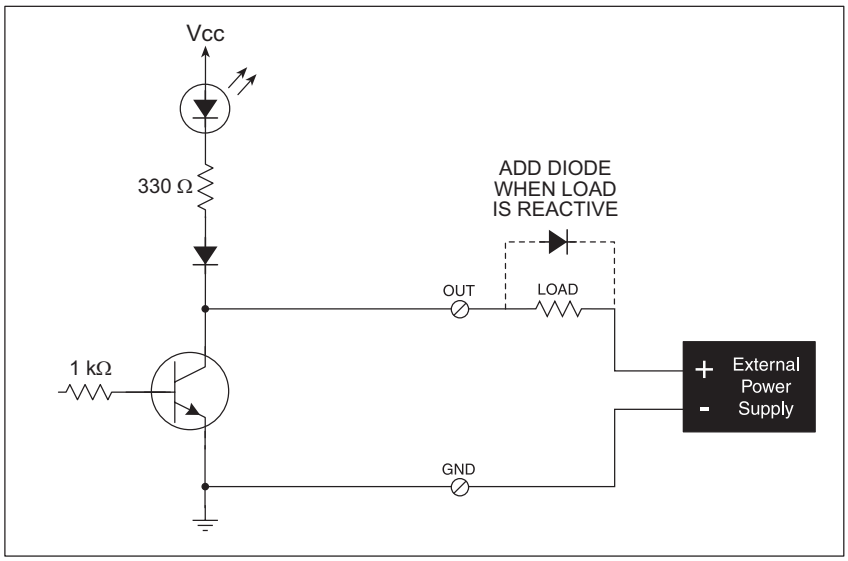


Figure B-6. Prototyping Board Sinking Digital Outputs

B.4.5 Relay Outputs

Figure B-7 shows the contact connections for the relay on the Prototyping Board. A diode across the coil provides a return path for inductive spikes, and snubbers across the relay contacts protect the relay contacts from inductive spikes.

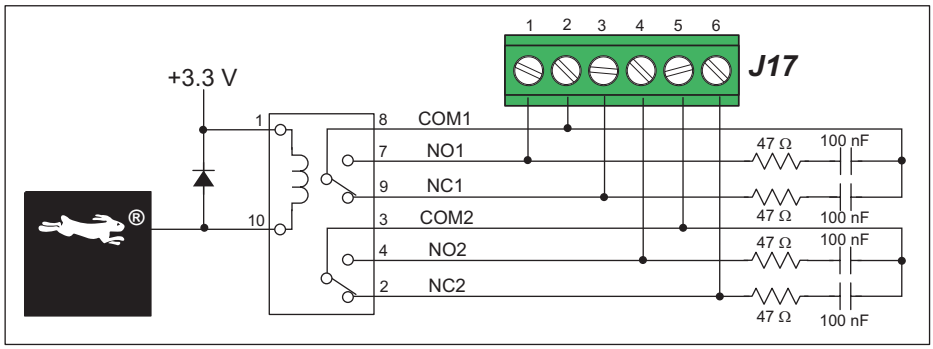


Figure B-7. Prototyping Board Relay Output Contact Connections

The relay is driven by pin PA4 of the RCM3300/RCM3310 module via U8, and is controlled by PE7 and PG5 as shown in the sample applications.

B.4.6 Serial Communication

The Prototyping Board allows you to access four of the serial ports from the RCM3300/ RCM3310 module. Table B-2 summarizes the configuration options.

Table B-2. Prototyping Board Serial Port Configurations

Serial Port	Signal Header	Configured via	Default Use	Alternate Use
C	J14	JP5*	RS-485	—
D	J7	JP3	RabbitNet (PD2 = 1)	Rabbit 3000 quadrature decoder
	J11		SF1000/modem (PD2 = 0)	
E	J14	—	RS-232	—
F	J14	—	RS-232	—

* RS-485 termination and bias resistors are configured via header JP5.

Serial Port D is configured in software either to allow J7 to be used as a RabbitNet port or to allow J11 to be used as a serial interface for either the SF1000 series serial flash or the modem module.

B.4.6.1 RS-232

RS-232 serial communication on the Prototyping Board is supported by an RS-232 transceiver installed at U9. 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 signals to RS-232 signal levels. Note that the polarity is reversed in an RS-232 circuit so that a +5 V output becomes approximately -10 V and 0 V is output as +10 V. The RS-232 transceiver also provides the proper line loading for reliable communication.

RS-232 can be used effectively at the RCM3300/RCM3310 module'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();
}
```

B.4.6.2 RS-485

The Prototyping Board has one RS-485 serial channel, which is connected to the Rabbit 3000 Serial Port C through an RS-485 transceiver. The half-duplex communication uses an output from PD7 on the Rabbit 3000 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 C is configured in software for RS-485 as follows.

```
#define ser485open serCopen
#define ser485close serCclose
#define ser485wrFlush serCwrFlush
#define ser485rdFlush serCrdFlush
#define ser485putc serCputc
#define ser485getc serCgetc

#define CINBUFSIZE  15
#define COUTBUFSIZE 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 Prototyping Boards with RCM3300/RCM3310 modules installed 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 B-8. Note that a common ground is recommended.

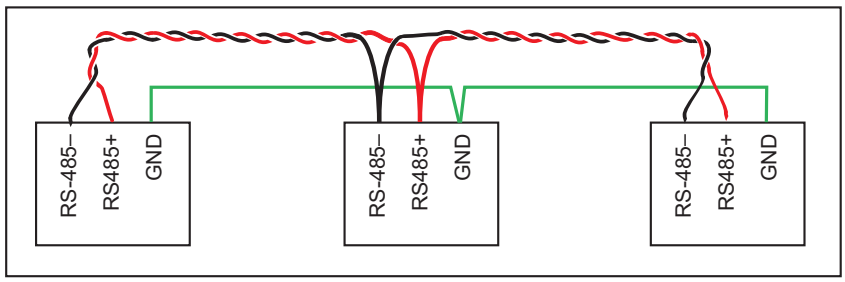


Figure B-8. Multidrop Network

The Prototyping Board comes with a 220 Ω termination resistor and two 681 Ω bias resistors installed and enabled with jumpers across pins 1–2 and 5–6 on header JP5, as shown in Figure B-9.

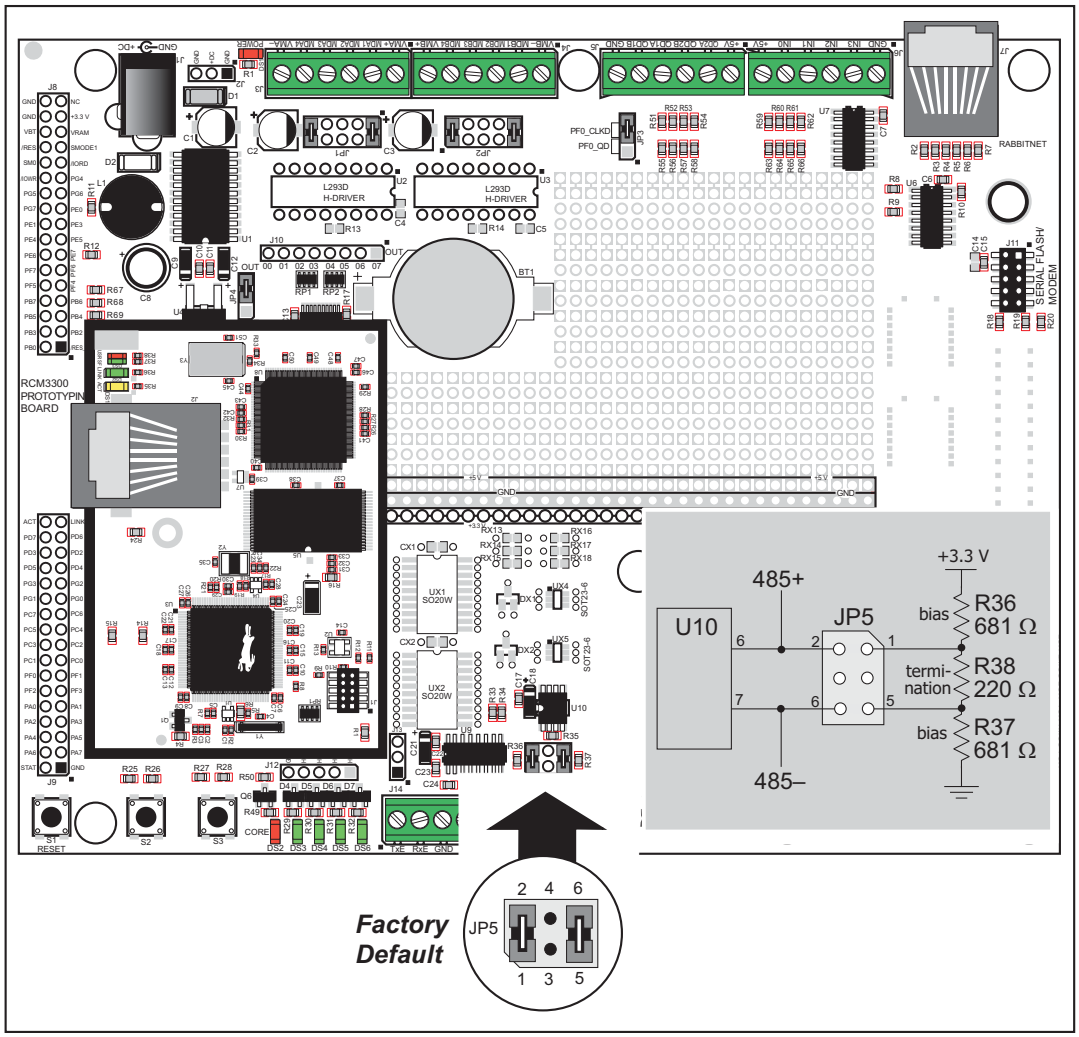


Figure B-9. RS-485 Termination and Bias Resistors

For best performance, the termination resistors in a multidrop network should be enabled only on the end nodes of the network, but ***not*** on the intervening nodes. Jumpers on boards whose termination resistors are not enabled may be stored across pins 1–3 and 4–6 of header JP5.

B.4.7 RabbitNet Ports

The RJ-45 jack labeled ***RabbitNet*** is a clocked SPI RS-422 serial I/O expansion port for use with RabbitNet peripheral boards. The ***RabbitNet*** jack does ***not*** support Ethernet connections. Header JP3 must have pins 2–3 jumpered when using the RabbitNet port.

The RabbitNet port is enabled in software by setting PD2 = 1. Note that the RabbitNet port and the J11 interface cannot be used simultaneously.

B.4.8 Other Prototyping Board Modules

An optional LCD/keypad module is available that can be mounted on the Prototyping Board. The signals on headers LCD1JB and LCD1JC will be available only if the LCD/keypad module is installed. Refer to Appendix C, "LCD/Keypad Module," for complete information.

Either Z-World's SF1000 series serial flash or modem module currently being developed may be installed in the socket labeled J11. The J11 interface is enabled in software by setting PD2 = 0. Header JP3 must have pins 2–3 jumpered when using the J11 interface. Note that the RabbitNet port and the J11 interface cannot be used simultaneously.

B.4.9 Quadrature Encoder

Four quadrature encoder outputs are available on screw-terminal header J5. To use the PF0 output from the Rabbit microprocessor, which goes to the QD1B output, remember to reconfigure the jumper on header JP3 to jumper pins 1–2.

Additional information on the use of the quadrature encoders on Parallel Port F is provided in the ***Rabbit 3000 Microprocessor User's Manual***.

B.4.10 Stepper-Motor Control

The Prototyping Board can be used to demonstrate the use of the RCM3300/RCM3310 to control a stepper motor. Stepper motor control typically directs moves in two orthogonal directions, and so two sets of stepper-motor control circuits are provided for via screw-terminal headers J3 and J4.

In order to use the stepper-motor control, install two Texas Instruments L293DN chips at locations U2 and U3 (shown in Figure B-10). These chips are readily available from your favorite electronics parts source, and may be purchased through Z-World's Web store as Z-World part number 660-0205.

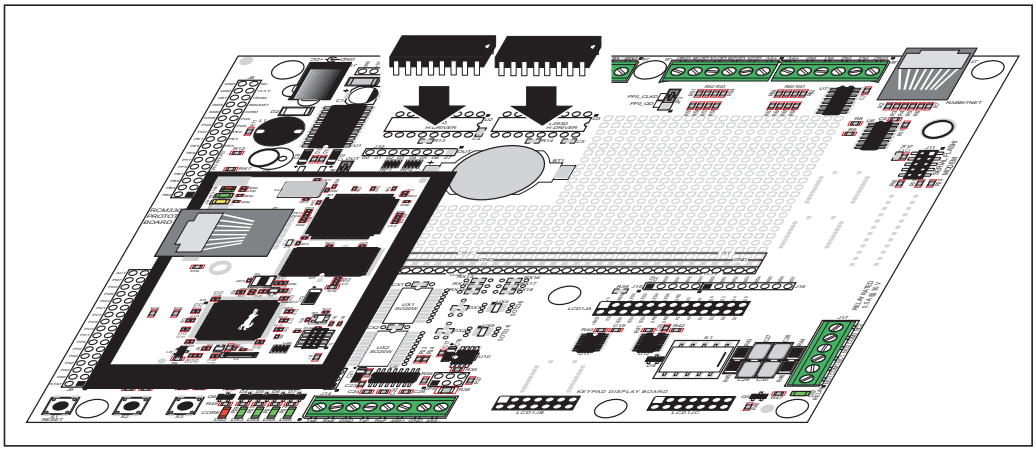


Figure B-10. Install Four-Channel Push-Pull Driver Chips

Figure B-11 shows the stepper-motor driver circuit.

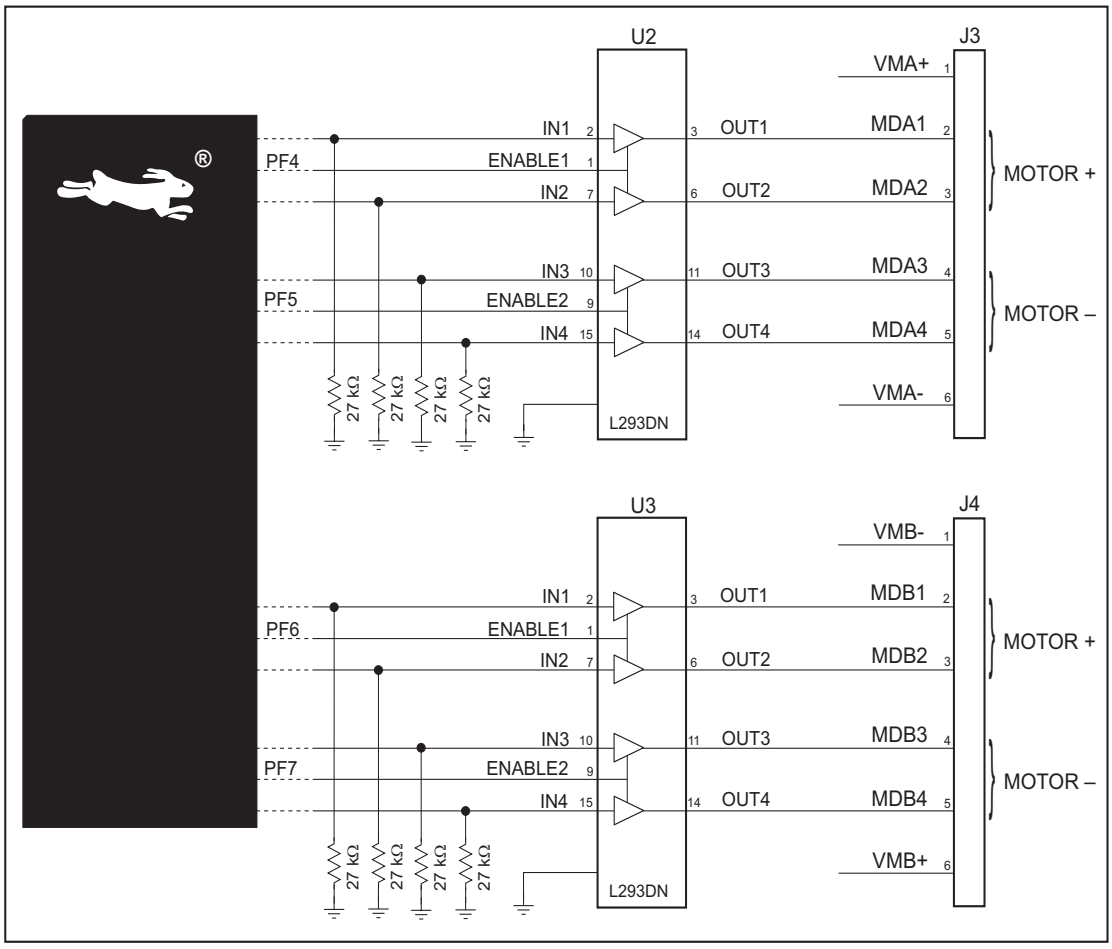


Figure B-11. Stepper-Motor Driver Circuit

The stepper motor(s) can be powered either from the onboard power supply or from an external power based on the jumper settings on headers JP1 and JP2.

Table B-3. Stepper Motor Power-Supply Options

Header	Pins Connected		Factory Default
JP1	1–2 9–10	Onboard power supply to U2	✕
	3–4 7–8	External power supply to U2	
JP2	1–2 9–10	Onboard power supply to U3	✕
	3–4 7–8	External power supply to U3	

B.5 Prototyping Board Jumper Configurations

Figure B-12 shows the header locations used to configure the various Prototyping Board options via jumpers.

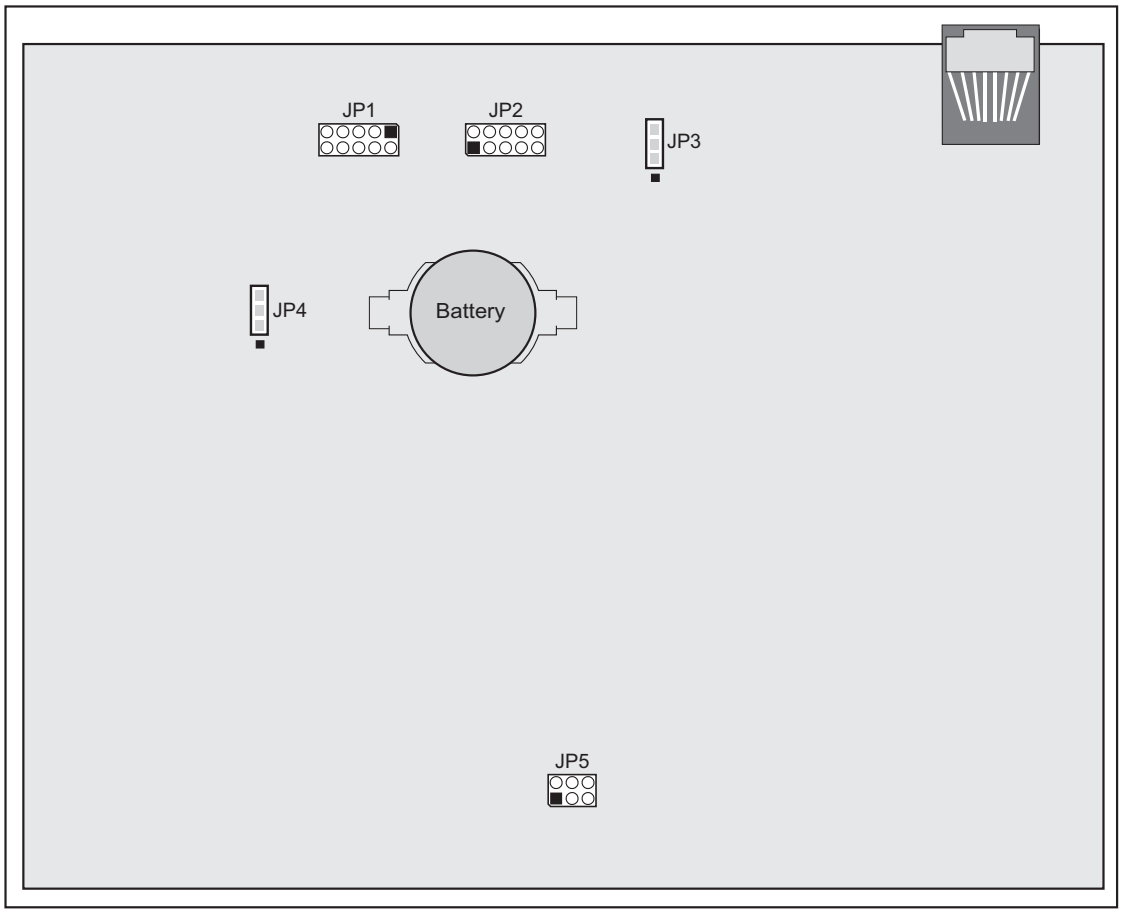


Figure B-12. Location of Prototyping Board Configurable Positions

Table B-4 lists the configuration options using jumpers.

Table B-4. Prototyping Board Jumper Configurations

Header	Description	Pins Connected		Factory Default
JP1	Stepper Motor Power-Supply Options (U2)	1–2 9–10	Onboard power supply	✕
		3–4 7–8	External power supply	
JP2	Stepper Motor Power-Supply Options (U3)	1–2 9–10	Onboard power supply	✕
		3–4 7–8	External power supply	
JP3	PF0 Option	1–2	Quadrature decoder outputs enabled	
		2–3	RabbitNet/Serial Flash interface enabled	✕
JP4	RCM3300/RCM3310 Power Supply	2–3	RCM3300/RCM3310 powered via Prototyping Board	✕
JP5	RS-485 Bias and Termination Resistors	1–2 5–6	Bias and termination resistors connected	✕
		1–3 4–6	Bias and termination resistors ***not*** connected (parking position for jumpers)	

B.6 Use of Rabbit 3000 Parallel Ports

Table B-5 lists the Rabbit 3000 parallel ports and their use for the Prototyping Board.

Table B-5. Prototyping Board Use of Rabbit 3000 Parallel Ports

Port	I/O	Use		Initial State
PA0–PA3	Data Bus	LCD/keypad module, motor driver, LEDs, J7		Active high
PA4	Data Bus	LCD/keypad module, motor driver, relay, J7		Active high
PA5–PA7	Data Bus	LCD/keypad module, motor control, J7		Active high
PB0	Input	CLKB, Serial Flash SCLK		High
PB1	Input	CLKA Programming Port		High (when not driven by CLKA)
PB2–PB5	Address Bus	LCD/keypad module, J6		High
PB6–PB7	Address Bus	J6		High
PC0	Output	TXD SPI, serial flash, J7	Serial Port D	High (disabled)
PC1	Input	RXD SPI, serial flash, J7		High (disabled)
PC2	Output	TXC RS-485 J7	Serial Port C	High (disabled)
PC3	Input	RXC RS-485 J7		High (disabled)
PC4	Output	TXB RCM3300 serial flash	Serial Port B*	High (disabled)
PC5	Input	RXB RCM3300 serial flash		High (disabled)
PC6	Output	TXA Programming Port	Serial Port A	High
PC7	Input	RXA Programming Port		High
PD0†	Output	RCM3300 **USR** LED		High
PD1†	Output	RCM3300 onboard serial flash select		High (disabled)
PD2	Output	SPI, serial flash, J7		Low (SPI disabled)
PD3	Output	SPI, serial flash, J7		High (SPI CS disabled)
PD4–PD6	Input	Serial flash, J7		High (disabled)
PD7	Output	RS-485 Tx enable		Low (disabled)
PE0–PE1	Input	IN0–IN1, J6		High
PE2†	Output	Ethernet AEN		Low (disabled)
PE3	Output	Motor driver A clock pulse		Low (disabled)
PE4–PE5	Input	IN2–IN3, J6		High
PE6	Output	LCD/keypad module		High (disabled)
PE7	Output	Motor driver B clock pulse		High (disabled)

Table B-5. Prototyping Board Use of Rabbit 3000 Parallel Ports (continued)

Port	I/O	Use		Initial State
PF0	Input	SPI, serial flash, quadrature decoder, J7		High
PF1–PF3	Input	Quadrature decoder, J7		High
PF4–PF7	Output	Motor 1–4 control		Low (disabled)
PG0	Input	Switch S1		High
PG1	Input	Switch S2		High
PG2	Input	TXF RS-232	Serial Port F	High (disabled)
PG3	Input	RXF RS-232		High (disabled)
PG4	Output	Motor driver A enable		High (disabled)
PG5	Output	Motor driver B enable		High (disabled)
PG6	Input	TXE RS-232	Serial Port E	High (disabled)
PG7	Input	RXE RS-232		High (disabled)

* Serial Port B is not available on the Prototyping Board when the RCM3300/RCM3310 is plugged in.

† PD0, PD1, and PE2 are not normally available on the Prototyping Board because they are not brought out on RCM3300 headers J3 and J4.

APPENDIX C. LCD/KEYPAD MODULE

An optional LCD/keypad is available for the Prototyping Board. Appendix C describes the LCD/keypad and provides the software APIs to make full use of the LCD/keypad.

C.1 Specifications

The LCD/keypad module comes with or without a panel-mounted NEMA 4 water-resistant bezel as shown in Figure C-1.

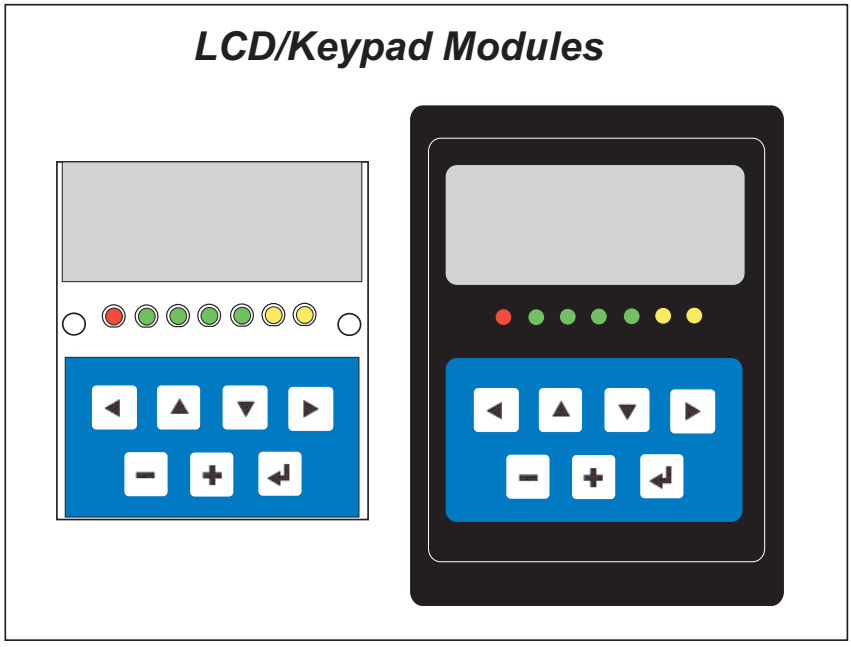


Figure C-1. LCD/Keypad Modules Versions

Only the version without the bezel can mount directly on the Prototyping Board; if you have the version with a bezel, you will have to remove the bezel to be able to mount the LCD/keypad module on the Prototyping Board. Either version of the LCD/keypad module can be installed at a remote location up to 60 cm (24") away. Contact your Z-World sales representative or your authorized Z-World distributor for further assistance in purchasing an LCD/keypad module.

Mounting hardware and a 60 cm (24") extension cable are also available for the LCD/keypad module through your Z-World sales representative or authorized distributor.

Table C-1 lists the electrical, mechanical, and environmental specifications for the LCD/keypad module.

Table C-1. LCD/Keypad Specifications

Parameter	Specification
Board Size	2.60" x 3.00" x 0.75" (66 mm x 76 mm x 19 mm)
Bezel Size	4.50" × 3.60" × 0.30" (114 mm × 91 mm × 7.6 mm)
Temperature	Operating Range: 0°C to +50°C Storage Range: –40°C to +85°C
Humidity	5% to 95%, noncondensing
Power Consumption	1.5 W maximum*
Connections	Connects to high-rise header sockets on the Prototyping Board
LCD Panel Size	122 × 32 graphic display
Keypad	7-key keypad
LEDs	Seven user-programmable LEDs

* The backlight adds approximately 650 mW to the power consumption.

The LCD/keypad module has 0.1" IDC headers at J1, J2, and J3 for physical connection to other boards or ribbon cables. Figure C-2 shows the LCD/keypad module footprint. These values are relative to one of the mounting holes.

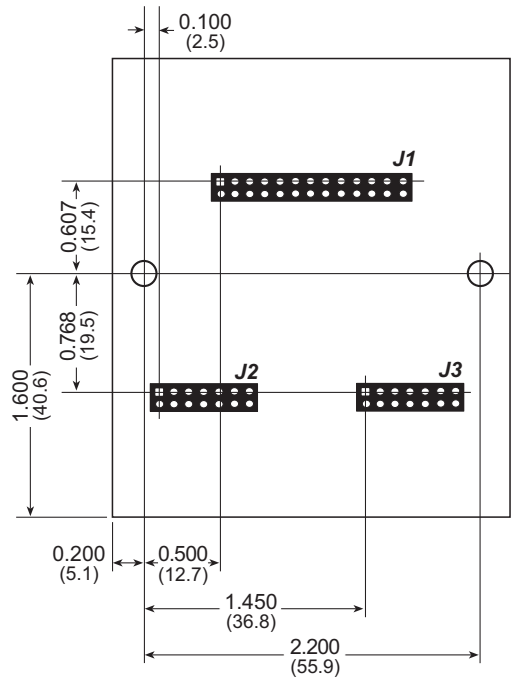


Figure C-2. User Board Footprint for LCD/Keypad Module

C.2 Contrast Adjustments for All Boards

Depending on when you acquired your LCD/keypad module, you will be able to set the contrast on the LCD display by adjusting the potentiometer at R2 or by setting the voltage for 3.3 V by setting the jumper across pins 3–4 on header J5 as shown in Figure C-3. Only one of these two options is available on a given LCD/keypad module.

LCD/Keypad Module Jumper Configurations

Header	Description	Pins Connected	Factory Default
J5	2.8 V	1–2	×
	3.3 V	3–4	
	5 V	n.c.	

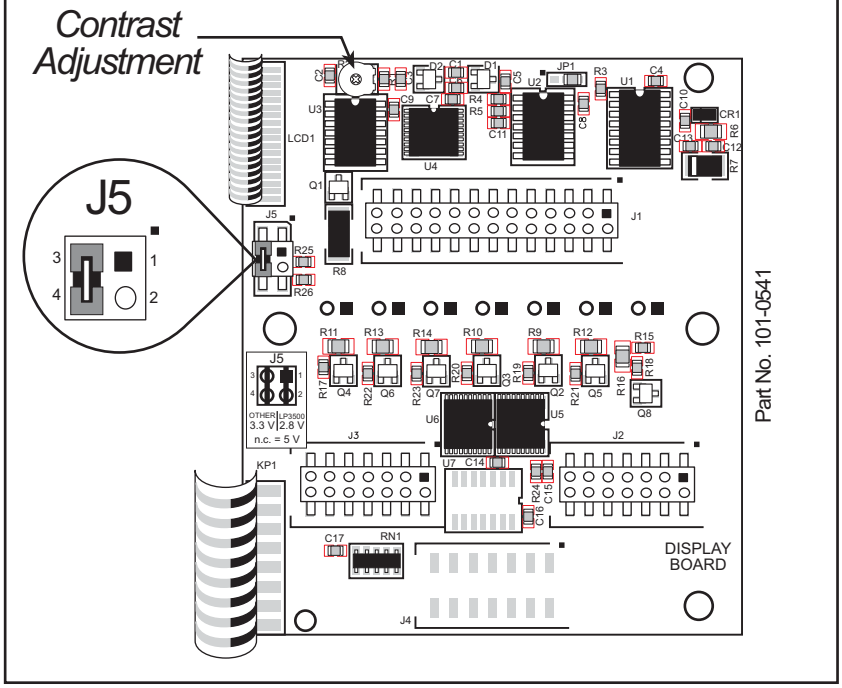


Figure C-3. LCD/Keypad Module Contrast Adjustments

NOTE: Older LCD/keypad modules that do not have a header at J5 or a contrast adjustment potentiometer at R2 are limited to operate only at 5 V, and will not work with the Prototyping Board. The older LCD/keypad modules are no longer being sold.

C.3 Keypad Labeling

The keypad may be labeled according to your needs. A template is provided in Figure C-4 to allow you to design your own keypad label insert.

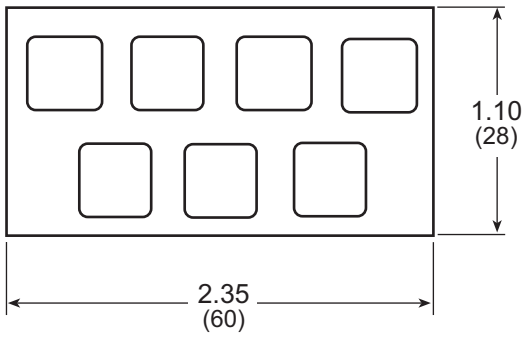


Figure C-4. Keypad Template

To replace the keypad legend, remove the old legend and insert your new legend prepared according to the template in Figure C-4. The keypad legend is located under the blue keypad matte, and is accessible from the left only as shown in Figure C-5.

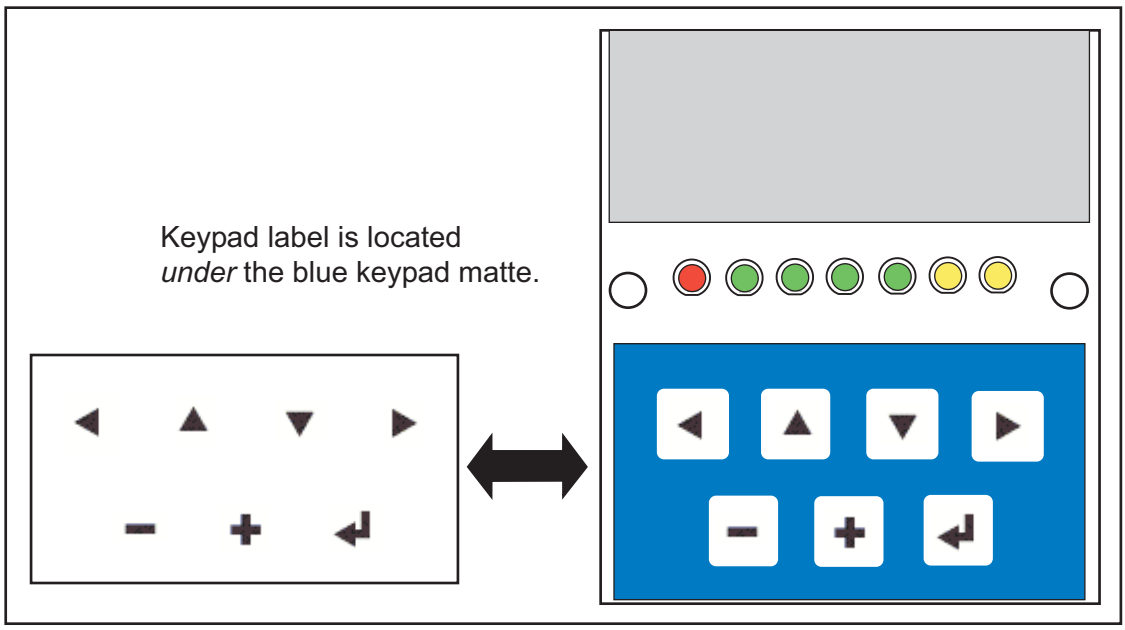


Figure C-5. Removing and Inserting Keypad Label

The sample program **KEYBASIC.C** in the **122x32_1x7** folder in **SAMPLES\LCD_KEYPAD** shows how to reconfigure the keypad for different applications.

C.4 Header Pinouts

Figure C-6 shows the pinouts for the LCD/keypad module.

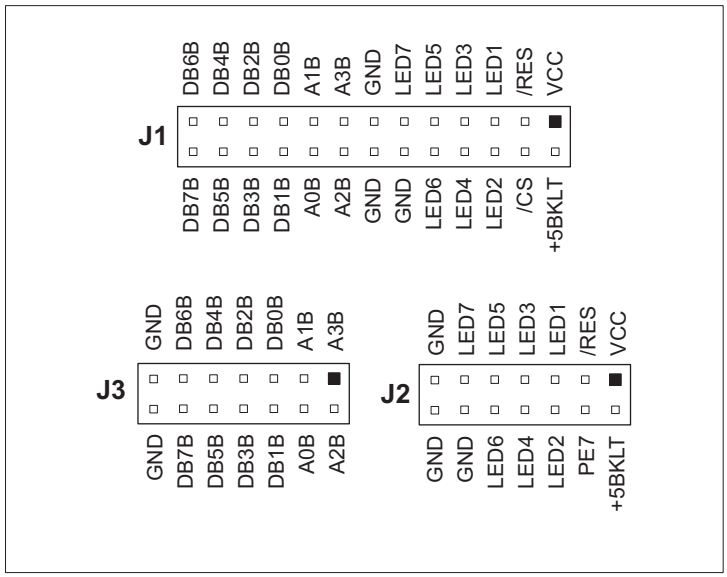


Figure C-6. LCD/Keypad Module Pinouts

C.4.1 I/O Address Assignments

The LCD and keypad on the LCD/keypad module are addressed by the /CS strobe as explained in Table C-2.

Table C-2. LCD/Keypad Module Address Assignment

Address	Function
0xE000	Device select base address (/CS)
0xExx0–0xExx7	LCD control
0xExx8	LED enable
0xExx9	Not used
0xExxA	7-key keypad
0xExxB (bits 0–6)	7-LED driver
0xExxB (bit 7)	LCD backlight on/off
0xExxC–ExxF	Not used

C.5 Mounting LCD/Keypad Module on the Prototyping Board

Install the LCD/keypad module on header sockets LCD1JA, LCD1JB, and LCD1JC of the Prototyping Board as shown in Figure C-7. Be careful to align the pins over the headers, and do not bend them as you press down to mate the LCD/keypad module with the Prototyping Board.

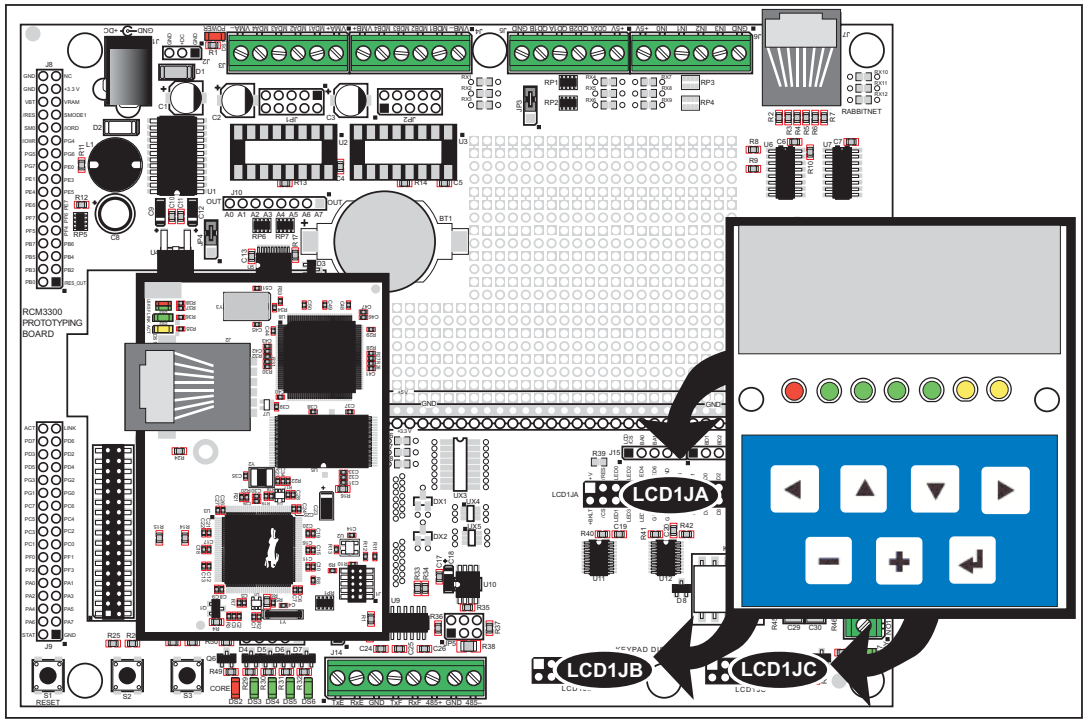


Figure C-7. Install LCD/Keypad Module on Prototyping Board

C.6 Bezel-Mount Installation

This section describes and illustrates how to bezel-mount the LCD/keypad module designed for remote installation. Follow these steps for bezel-mount installation.

1. Cut mounting holes in the mounting panel in accordance with the recommended dimensions in Figure C-8, then use the bezel faceplate to mount the LCD/keypad module onto the panel.

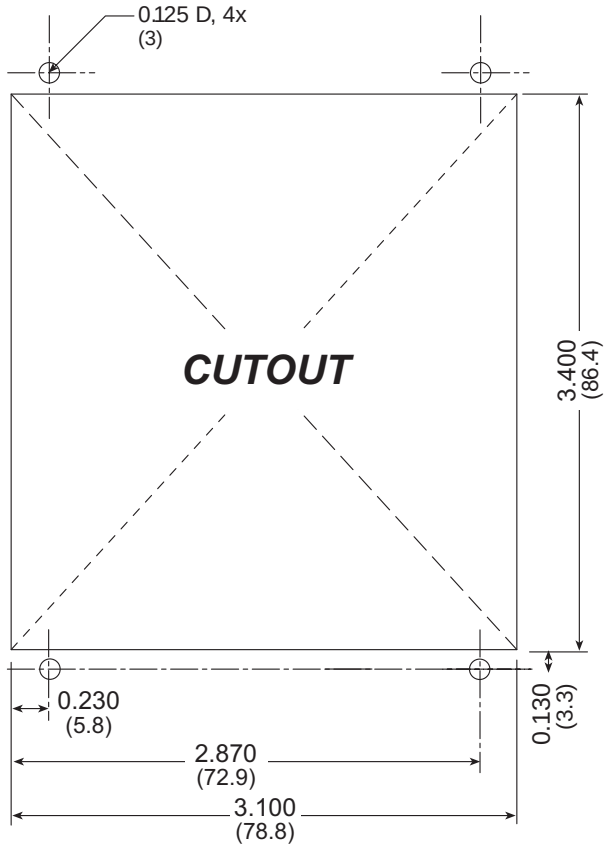


Figure C-8. Recommended Cutout Dimensions

2. Carefully “drop in” the LCD/keypad module with the bezel and gasket attached.

3. Fasten the unit with the four 4-40 screws and washers included with the LCD/keypad module. If your panel is thick, use a 4-40 screw that is approximately 3/16" (5 mm) longer than the thickness of the panel.

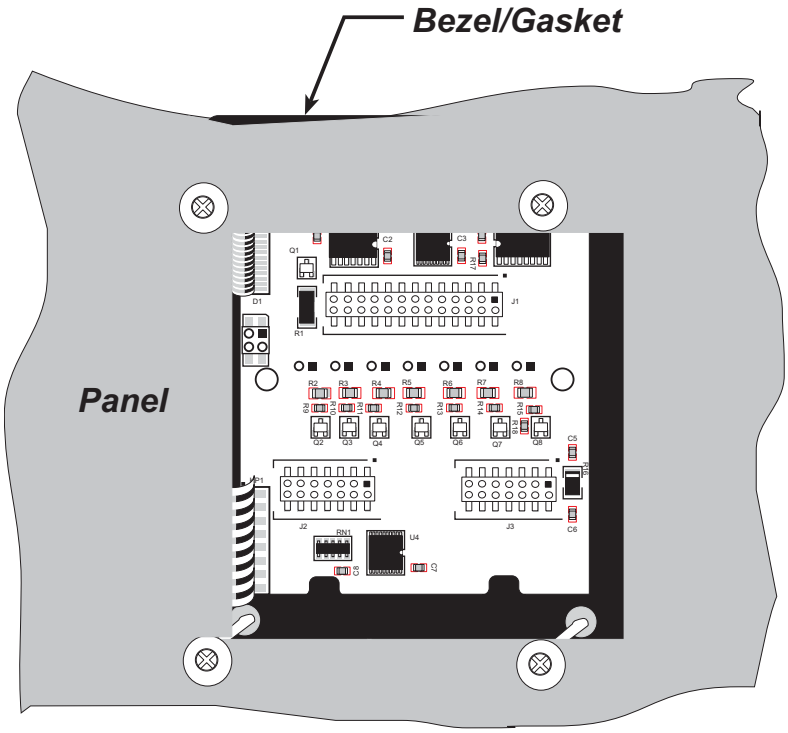


Figure C-9. LCD/Keypad Module Mounted in Panel (rear view)

Carefully tighten the screws until the gasket is compressed and the plastic bezel face-plate is touching the panel.

Do not tighten each screw fully before moving on to the next screw. Apply only one or two turns to each screw in sequence until all are tightened manually as far as they can be so that the gasket is compressed and the plastic bezel faceplate is touching the panel.

C.6.1 Connect the LCD/Keypad Module to Your Prototyping Board

The LCD/keypad module can be located as far as 2 ft. (60 cm) away from the Prototyping Board, and is connected via a ribbon cable as shown in Figure C-10.

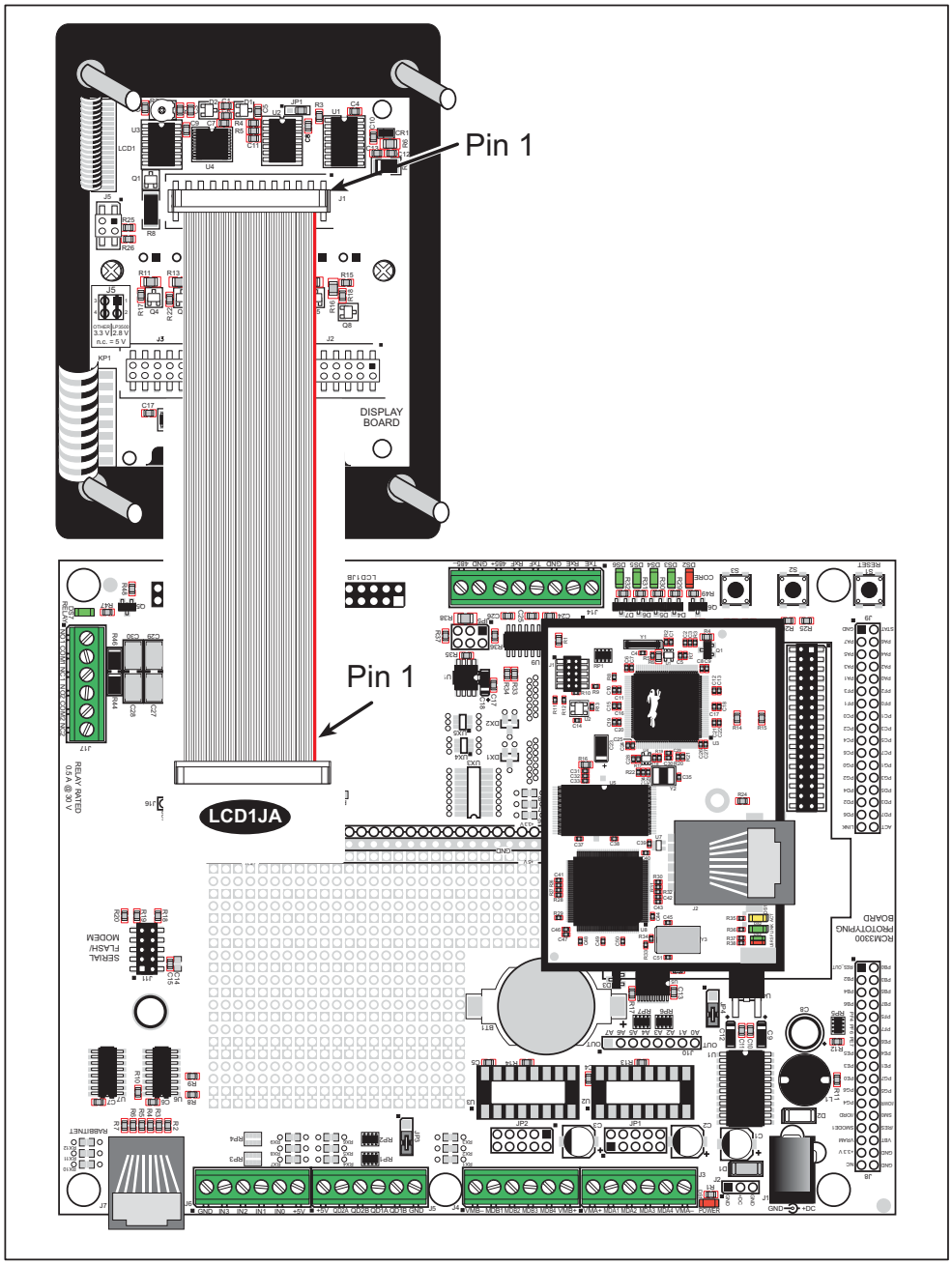


Figure C-10. Connecting LCD/Keypad Module to Prototyping Board

Note the locations and connections relative to pin 1 on both the Prototyping Board and the LCD/keypad module.

Z-World offers 2 ft. (60 cm) extension cables. Contact your authorized Z-World distributor or a Z-World sales representative at +1(530)757-3737 for more information.

C.7 Sample Programs

Sample programs illustrating the use of the LCD/keypad module with the Prototyping Board are provided in the **SAMPLES\RCM3300** folder.

These sample programs use the auxiliary I/O bus on the Rabbit 3000 chip, and so the **#define PORTA_AUX_IO** line is already included in the sample programs.

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 RCM3300/RCM3310 must be in **Program** mode (see Section 4.3, "Programming Cable"), and must be connected to a PC using the programming cable as described in Chapter 2, "Getting Started."

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

The following sample programs are found in the **SAMPLES\RCM3300\LCD_KEYPAD** folder.

- **KEYPADTOLED.C**—This program demonstrates the use of the external I/O bus. The program will light up an LED on the LCD/keypad module and will display a message on the LCD when a key press is detected. The DS3, DS4, DS5, and DS6 LEDs on the Prototyping Board will also light up.
- **LCDKEYFUN.C**—This program demonstrates how to draw primitive features from the graphic library (lines, circles, polygons), and also demonstrates the keypad with the key release option.
- **SWITCHTOLCD.C**—This program demonstrates the use of the external I/O bus. The program will light up an LED on the LCD/keypad module and will display a message on the LCD when a switch press is detected. The DS1 and DS2 LEDs on the Prototyping Board will also light up.

Additional sample programs are available in the **SAMPLES\LCD_KEYPAD\122x32_1x7** folder.

C.8 LCD/Keypad Module Function Calls

When mounted on the Prototyping Board, the LCD/keypad module uses the auxiliary I/O bus on the Rabbit 3000 chip. Remember to add the line

```
#define PORTA_AUX_IO
```

to the beginning of any programs using the auxiliary I/O bus.

C.8.1 LCD/Keypad Module Initialization

The initialization function is found in the **LCD122KEY7.LIB** library in the Dynamic C **DISPLAYS** folder.

```
void dispInit();
```

Initializes the LCD/keypad module. The keypad is set up using **keypadDef()** or **keyConfig()** after this function call.

RETURN VALUE

None.

C.8.2 LEDs

When power is applied to the LCD/keypad module for the first time, the red LED (DS1) will come on, indicating that power is being applied to the LCD/keypad module. The red LED is turned off when the **brdInit** function executes.

One function is available to control the LEDs, and can be found in the **LCD122KEY7.LIB** library in the Dynamic C **DISPLAYS** folder.

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

LED on/off control. This function will only work when the LCD/keypad module is installed on the Prototyping Board.

PARAMETERS

led is the LED to control.

0 = LED DS1
1 = LED DS2
2 = LED DS3
3 = LED DS4
4 = LED DS5
5 = LED DS6
6 = LED DS7

value is the value used to control whether the LED is on or off (0 or 1).

0 = off
1 = on

RETURN VALUE

None.

C.8.3 LCD Display

The functions used to control the LCD display are contained in the **GRAPHIC.LIB** library located in the Dynamic C **DISPLAYS\GRAPHIC** library folder. When *x* and *y* coordinates on the display screen are specified, *x* can range from 0 to 121, and *y* can range from 0 to 31. These numbers represent pixels from the top left corner of the display.

void glInit(void);

Initializes the display devices, clears the screen.

RETURN VALUE

None.

SEE ALSO

glDispOnOFF, glBacklight, glSetContrast, glPlotDot, glBlock, glPlotDot, glPlotPolygon, glPlotCircle, glHScroll, glVScroll, glXFontInit, glPrintf, glPutChar, glSetBrushType, glBuffLock, glBuffUnlock, glPlotLine

void glBackLight(int onOff);

Turns the display backlight on or off.

PARAMETER

onOff turns the backlight on or off

1—turn the backlight on
0—turn the backlight off

RETURN VALUE

None.

SEE ALSO

glInit, glDispOnoff, glSetContrast

void glDispOnOff(int onOff);

Sets the LCD screen on or off. Data will not be cleared from the screen.

PARAMETER

onOff turns the LCD screen on or off

1—turn the LCD screen on
0—turn the LCD screen off

RETURN VALUE

None.

SEE ALSO

glInit, glSetContrast, glBackLight

void glSetContrast(unsigned level);

Sets display contrast.

NOTE: This function is not used with the LCD/keypad module since the support circuits are not available on the LCD/keypad module.

void glFillScreen(char pattern);

Fills the LCD display screen with a pattern.

PARAMETER

The screen will be set to all black if **pattern** is 0xFF, all white if **pattern** is 0x00, and vertical stripes for any other pattern.

RETURN VALUE

None.

SEE ALSO

glBlock, glBlankScreen, glPlotPolygon, glPlotCircle

void glBlankScreen(void);

Blanks the LCD display screen (sets LCD display screen to white).

RETURN VALUE

None.

SEE ALSO

glFillScreen, glBlock, glPlotPolygon, glPlotCircle

void glBlock(int x, int y, int bmWidth, int bmHeight);

Draws a rectangular block in the page buffer and on the LCD if the buffer is unlocked. Any portion of the block that is outside the LCD display area will be clipped.

PARAMETERS

x is the *x* coordinate of the top left corner of the block.

y is the *y* coordinate of the top left corner of the block.

bmWidth is the width of the block.

bmWidth is the height of the block.

RETURN VALUE

None.

SEE ALSO

glFillScreen, glBlankScreen, glPlotPolygon, glPlotCircle

void glPlotVPolygon(int n, int *pFirstCoord);

Plots the outline of a polygon in the LCD page buffer, and on the LCD if the buffer is unlocked. Any portion of the polygon that is outside the LCD display area will be clipped. If fewer than 3 vertices are specified, the function will return without doing anything.

PARAMETERS

n is the number of vertices.

***pFirstCoord** is a pointer to array of vertex coordinates: **x1,y1, x2,y2, x3,y3,...**

RETURN VALUE

None.

SEE ALSO

glPlotPolygon, glFillPolygon, glFillVPolygon

void glPlotPolygon(int n, int y1, int x2, int y2, ...);

Plots the outline of a polygon in the LCD page buffer and on the LCD if the buffer is unlocked. Any portion of the polygon that is outside the LCD display area will be clipped. If fewer than 3 vertices are specified, the function will return without doing anything.

PARAMETERS

n is the number of vertices.

y1 is the *y* coordinate of the first vertex.

x1 is the *x* coordinate of the first vertex.

y2 is the *y* coordinate of the second vertex.

x2 is the *x* coordinate of the second vertex.

... are the coordinates of additional vertices.

RETURN VALUE

None.

SEE ALSO

glPlotVPolygon, glFillPolygon, glFillVPolygon

void glFillVPolygon(int n, int *pFirstCoord);

Fills a polygon in the LCD page buffer and on the LCD screen if the buffer is unlocked. Any portion of the polygon that is outside the LCD display area will be clipped. If fewer than 3 vertices are specified, the function will return without doing anything.

PARAMETERS

n is the number of vertices.

***pFirstCoord** is a pointer to array of vertex coordinates: **x1,y1, x2,y2, x3,y3,...**

RETURN VALUE

None.

SEE ALSO

glFillPolygon, glPlotPolygon, glPlotVPolygon

void glFillPolygon(int n, int x1, int y1, int x2, int y2, ...);

Fills a polygon in the LCD page buffer and on the LCD if the buffer is unlocked. Any portion of the polygon that is outside the LCD display area will be clipped. If fewer than 3 vertices are specified, the function will return without doing anything.

PARAMETERS

n is the number of vertices.

x1 is the *x* coordinate of the first vertex.

y1 is the *y* coordinate of the first vertex.

x2 is the *x* coordinate of the second vertex.

y2 is the *y* coordinate of the second vertex.

... are the coordinates of additional vertices.

RETURN VALUE

None.

SEE ALSO

glFillVPolygon, glPlotPolygon, glPlotVPolygon

void glPlotCircle(int xc, int yc, int rad);

Draws the outline of a circle in the LCD page buffer and on the LCD if the buffer is unlocked. Any portion of the circle that is outside the LCD display area will be clipped.

PARAMETERS

xc is the *x* coordinate of the center of the circle.

yc is the *y* coordinate of the center of the circle.

rad is the radius of the center of the circle (in pixels).

RETURN VALUE

None.

SEE ALSO

glFillCircle, glPlotPolygon, glFillPolygon

void glFillCircle(int xc, int yc, int rad);

Draws a filled circle in the LCD page buffer and on the LCD if the buffer is unlocked. Any portion of the circle that is outside the LCD display area will be clipped.

PARAMETERS

xc is the *x* coordinate of the center of the circle.

yc is the *y* coordinate of the center of the circle.

rad is the radius of the center of the circle (in pixels).

RETURN VALUE

None.

SEE ALSO

glPlotCircle, glPlotPolygon, glFillPolygon

void glXFontInit(fontInfo *pInfo, char pixWidth, char pixHeight, unsigned startChar, unsigned endChar, unsigned long xmemBuffer);

Initializes the font descriptor structure, where the font is stored in **xmem**.

PARAMETERS

***pInfo** is a pointer to the font descriptor to be initialized.

pixWidth is the width (in pixels) of each font item.

pixHeight is the height (in pixels) of each font item.

startChar is the value of the first printable character in the font character set.

endChar is the value of the last printable character in the font character set.

xmemBuffer is the **xmem** pointer to a linear array of font bitmaps.

RETURN VALUE

None.

SEE ALSO

glPrinf

unsigned long glFontCharAddr(fontInfo *pInfo, char letter);

Returns the **xmem** address of the character from the specified font set.

PARAMETERS

***pInfo** is the **xmem** address of the bitmap font set.

letter is an ASCII character.

RETURN VALUE

xmem address of bitmap character font, column major, and byte-aligned.

SEE ALSO

glPutFont, glPrintf

void glPutFont(int x, int y, fontInfo *pInfo, char code);

Puts an entry from the font table to the page buffer and on the LCD if the buffer is unlocked. Each font character's bitmap is column major and byte-aligned. Any portion of the bitmap character that is outside the LCD display area will be clipped.

PARAMETERS

x is the *x* coordinate (column) of the top left corner of the text.

y is the *y* coordinate (row) of the top left corner of the text.

***pInfo** is a pointer to the font descriptor.

code is the ASCII character to display.

RETURN VALUE

None.

SEE ALSO

glFontCharAddr, glPrintf

void glSetPfStep(int stepX, int stepY);

Sets the **glPrintf()** printing step direction. The *x* and *y* step directions are independent signed values. The actual step increments depend on the height and width of the font being displayed, which are multiplied by the step values.

PARAMETERS

stepX is the **glPrintf** *x* step value

stepY is the **glPrintf** *y* step value

RETURN VALUE

None.

SEE ALSO

Use **glGetPfStep()** to examine the current *x* and *y* printing step direction.

int glGetPfStep(void);

Gets the current **glPrintf()** printing step direction. Each step direction is independent of the other, and is treated as an 8-bit signed value. The actual step increments depends on the height and width of the font being displayed, which are multiplied by the step values.

RETURN VALUE

The *x* step is returned in the MSB, and the *y* step is returned in the LSB of the integer result.

SEE ALSO

Use **glGetPfStep()** to control the *x* and *y* printing step direction.

void glPutChar(char ch, char *ptr, int *cnt, glPutCharInst *pInst)

Provides an interface between the **STDIO** string-handling functions and the graphic library. The **STDIO** string-formatting function will call this function, one character at a time, until the entire formatted string has been parsed. Any portion of the bitmap character that is outside the LCD display area will be clipped.

PARAMETERS

ch is the character to be displayed on the LCD.

***ptr** is not used, but is a place holder for **STDIO** string functions.

***cnt** is not used, is a place holder for **STDIO** string functions.

***pInst** is a font descriptor pointer.

RETURN VALUE

None.

SEE ALSO

glPrintf, glPutFont, doprnt

void glPrintf(int x, int y, fontInfo *pInfo, char *fmt, ...);

Prints a formatted string (much like **printf**) on the LCD screen. Only the character codes that exist in the font set are printed, all others are skipped. For example, '\b', '\t', '\n' and '\r' (ASCII backspace, tab, new line, and carriage return, respectively) will be printed if they exist in the font set, but will not have any effect as control characters. Any portion of the bitmap character that is outside the LCD display area will be clipped.

PARAMETERS

x is the *x* coordinate (column) of the top left corner of the text.

y is the *y* coordinate (row) of the top left corner of the text.

***pInfo** is a font descriptor pointer.

***fmt** is a formatted string.

... are formatted string conversion parameter(s).

EXAMPLE

```
glprintf(0,0, &fi12x16, "Test %d\n", count);
```

RETURN VALUE

None.

SEE ALSO

glXFontInit

void glBuffLock(void);

Increments LCD screen locking counter. Graphic calls are recorded in the LCD memory buffer and are not transferred to the LCD if the counter is non-zero.

> **NOTE:** **glBuffLock()** and **glBuffUnlock()** can be nested up to a level of 255, but be sure to balance the calls. It is not a requirement to use these procedures, but a set of **glBuffLock()** and **glBuffUnlock()** bracketing a set of related graphic calls speeds up the rendering significantly.

RETURN VALUE

None.

SEE ALSO

glBuffUnlock, glSwap

void glBuffUnlock(void);

Decrements the LCD screen locking counter. The contents of the LCD buffer are transferred to the LCD if the counter goes to zero.

RETURN VALUE

None.

SEE ALSO

glBuffLock, glSwap

void glSwap(void);

Checks the LCD screen locking counter. The contents of the LCD buffer are transferred to the LCD if the counter is zero.

RETURN VALUE

None.

SEE ALSO

glBuffUnlock, glBuffLock, _glSwapData (located in the library specifically for the LCD that you are using)

void glSetBrushType(int type);

Sets the drawing method (or color) of pixels drawn by subsequent graphic calls.

PARAMETER

type value can be one of the following macros.

PIXBLACK draws black pixels (turns pixel on).
PIXWHITE draws white pixels (turns pixel off).
PIXXOR draws old pixel XOR'ed with the new pixel.

RETURN VALUE

None.

SEE ALSO

glGetBrushType

int glGetBrushType(void);

Gets the current method (or color) of pixels drawn by subsequent graphic calls.

RETURN VALUE

The current brush type.

SEE ALSO

glSetBrushType

void glPlotDot(int x, int y);

Draws a single pixel in the LCD buffer, and on the LCD if the buffer is unlocked. If the coordinates are outside the LCD display area, the dot will not be plotted.

PARAMETERS

x is the *x* coordinate of the dot.

y is the *y* coordinate of the dot.

RETURN VALUE

None.

SEE ALSO

glPlotline, glPlotPolygon, glPlotCircle

void glPlotLine(int x0, int y0, int x1, int y1);

Draws a line in the LCD buffer, and on the LCD if the buffer is unlocked. Any portion of the line that is beyond the LCD display area will be clipped.

PARAMETERS

x0 is the *x* coordinate of one endpoint of the line.

y0 is the *y* coordinate of one endpoint of the line.

x1 is the *x* coordinate of the other endpoint of the line.

y1 is the *y* coordinate of the other endpoint of the line.

RETURN VALUE

None.

SEE ALSO

glPlotDot, glPlotPolygon, glPlotCircle

void glLeft1(int left, int top, int cols, int rows);

Scrolls byte-aligned window left one pixel, right column is filled by current pixel type (color).

PARAMETERS

left is the top left corner of bitmap, must be evenly divisible by 8, otherwise truncates.

top is the top left corner of the bitmap.

cols is the number of columns in the window, must be evenly divisible by 8, otherwise truncates.

rows is the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glHScroll, glRight1

void glRight1(int left, int top, int cols, int rows);

Scrolls byte-aligned window right one pixel, left column is filled by current pixel type (color).

PARAMETERS

left is the top left corner of bitmap, must be evenly divisible by 8, otherwise truncates.

top is the top left corner of the bitmap.

cols is the number of columns in the window, must be evenly divisible by 8, otherwise truncates.

rows is the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glHScroll, glLeft1

void glUp1(int left, int top, int cols, int rows);

Scrolls byte-aligned window up one pixel, bottom column is filled by current pixel type (color).

PARAMETERS

left is the top left corner of bitmap, must be evenly divisible by 8, otherwise truncates.

top is the top left corner of the bitmap.

cols is the number of columns in the window, must be evenly divisible by 8, otherwise truncates.

rows is the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glVScroll, glDown1

void glDown1(int left, int top, int cols, int rows);

Scrolls byte-aligned window down one pixel, top column is filled by current pixel type (color).

PARAMETERS

left is the top left corner of bitmap, must be evenly divisible by 8, otherwise truncates.

top is the top left corner of the bitmap.

cols is the number of columns in the window, must be evenly divisible by 8, otherwise truncates.

rows is the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glVScroll, glUp1

```
void glHScroll(int left, int top, int cols,
   int rows, int nPix);
```

Scrolls right or left, within the defined window by *x* number of pixels. The opposite edge of the scrolled window will be filled in with white pixels. The window must be byte-aligned.

Parameters will be verified for the following:

1. The **`left`** and **`cols`** parameters will be verified that they are evenly divisible by 8. If not, they will be truncated to a value that is a multiple of 8.
2. Parameters will be checked to verify that the scrolling area is valid. The minimum scrolling area is a width of 8 pixels and a height of one row.

PARAMETERS

`left` is the top left corner of bitmap, must be evenly divisible by 8.

`top` is the top left corner of the bitmap.

`cols` is the number of columns in the window, must be evenly divisible by 8.

`rows` is the number of rows in the window.

`nPix` is the number of pixels to scroll within the defined window (a negative value will produce a scroll to the left).

RETURN VALUE

None.

SEE ALSO

`glVScroll`

void glVScroll(int left, int top, int cols, int rows, int nPix);

Scrolls up or down, within the defined window by *x* number of pixels. The opposite edge of the scrolled window will be filled in with white pixels. The window must be byte-aligned.

Parameters will be verified for the following:

1. The **left** and **cols** parameters will be verified that they are evenly divisible by 8. If not, they will be truncated to a value that is a multiple of 8.
2. Parameters will be checked to verify that the scrolling area is valid. The minimum scrolling area is a width of 8 pixels and a height of one row.

PARAMETERS

left is the top left corner of bitmap, must be evenly divisible by 8.

top is the top left corner of the bitmap.

cols is the number of columns in the window, must be evenly divisible by 8.

rows is the number of rows in the window.

nPix is the number of pixels to scroll within the defined window (a negative value will produce a scroll up).

RETURN VALUE

None.

SEE ALSO

glHScroll

void glXPutBitmap(int left, int top, int width, int height, unsigned long bitmap);

Draws bitmap in the specified space. The data for the bitmap are stored in **xmem**. This function calls **glXPutFastmap** automatically if the bitmap is byte-aligned (the left edge and the width are each evenly divisible by 8).

Any portion of a bitmap image or character that is outside the LCD display area will be clipped.

PARAMETERS

left is the top left corner of the bitmap.

top is the top left corner of the bitmap.

width is the width of the bitmap.

height is the height of the bitmap.

bitmap is the address of the bitmap in **xmem**.

RETURN VALUE

None.

SEE ALSO

glXPutFastmap, glPrintf

```
void glXPutFastmap(int left, int top, int width,
   int height, unsigned long bitmap);
```

Draws bitmap in the specified space. The data for the bitmap are stored in **xmem**. This function is like **glXPutBitmap**, except that it is faster. The restriction is that the bitmap must be byte-aligned.

Any portion of a bitmap image or character that is outside the LCD display area will be clipped.

PARAMETERS

left is the top left corner of the bitmap, must be evenly divisible by 8, otherwise truncates.

top is the top left corner of the bitmap.

width is the width of the bitmap, must be evenly divisible by 8, otherwise truncates.

height is the height of the bitmap.

bitmap is the address of the bitmap in **xmem**.

RETURN VALUE

None.

SEE ALSO

glXPutBitmap, glPrintf

```
int TextWindowFrame(windowFrame *window,
   fontInfo *pFont, int x, int y, int winWidth,
   int winHeight)
```

Defines a text-only display window. This function provides a way to display characters within the text window using only character row and column coordinates. The text window feature provides end-of-line wrapping and clipping after the character in the last column and row is displayed.

NOTE: Execute the **TextWindowFrame** function before other **Text...** functions.

PARAMETERS

***window** is a window frame descriptor pointer.

***pFont** is a font descriptor pointer.

x is the *x* coordinate of the top left corner of the text window frame.

y is the *y* coordinate of the top left corner of the text window frame.

winWidth is the width of the text window frame.

winHeight is the height of the text window frame.

RETURN VALUE

0—window frame was successfully created.
-1—*x* coordinate + width has exceeded the display boundary.
-2—*y* coordinate + height has exceeded the display boundary.

void TextGotoXY(windowFrame *window, int col, int row);

Sets the cursor location to display the next character. The display location is based on the height and width of the character to be displayed.

NOTE: Execute the **TextWindowFrame** function before using this function.

PARAMETERS

***window** is a pointer to a font descriptor.

col is a character column location.

row is a character row location.

RETURN VALUE

None.

SEE ALSO

TextPutChar, TextPrintf, TextWindowFrame

void TextCursorLocation(windowFrame *window, int *col, int *row);

Gets the current cursor location that was set by a Graphic **Text...** function.

NOTE: Execute the **TextWindowFrame** function before using this function.

PARAMETERS

***window** is a pointer to a font descriptor.

***col** is a pointer to cursor column variable.

***row** is a pointer to cursor row variable.

RETURN VALUE

Lower word = Cursor Row location
Upper word = Cursor Column location

SEE ALSO

TextGotoXY, TextPrintf, TextWindowFrame, TextCursorLocation

void TextPutChar(struct windowFrame *window, char ch);

Displays a character on the display where the cursor is currently pointing. If any portion of a bitmap character is outside the LCD display area, the character will not be displayed. The cursor increments its position as needed.

NOTE: Execute the **TextWindowFrame** function before using this function.

PARAMETERS

***window** is a pointer to a font descriptor.

ch is a character to be displayed on the LCD.

RETURN VALUE

None.

SEE ALSO

TextGotoXY, TextPrintf, TextWindowFrame, TextCursorLocation

void TextPrintf(struct windowFrame *window, char *fmt, ...);

Prints a formatted string (much like **printf**) on the LCD screen. Only printable characters in the font set are printed, also escape sequences, '\r' and '\n' are recognized. All other escape sequences will be skipped over; for example, '\b' and 't' will print if they exist in the font set, but will not have any effect as control characters.

The text window feature provides end-of-line wrapping and clipping after the character in the last column and row is displayed. The cursor then remains at the end of the string.

NOTE: Execute the **TextWindowFrame** function before using this function.

PARAMETERS

***window** is a pointer to a font descriptor.

***fmt** is a formatted string.

... are formatted string conversion parameter(s).

EXAMPLE

```
TextPrintf(&TextWindow, "Test %d\n", count);
```

RETURN VALUE

None.

SEE ALSO

TextGotoXY, TextPutChar, TextWindowFrame, TextCursorLocation

C.8.4 Keypad

The functions used to control the keypad are contained in the **KEYPAD7.LIB** library located in the Dynamic C **KEYPADS** library folder.

```
void keyInit(void);
```

Initializes keypad process

RETURN VALUE

None.

SEE ALSO

brdInit

```
void keyConfig(char cRaw, char cPress,
   char cRelease, char cCntHold, char cSpdLo,
   char cCntLo, char cSpdHi);
```

Assigns each key with key press and release codes, and hold and repeat ticks for auto repeat and debouncing.

PARAMETERS

cRaw is a raw key code index.

1x7 keypad matrix with raw key code index assignments (in brackets):

[0]		[1]		[2]		[3]
	[4]		[5]		[6]	

User Keypad Interface

cPress is a key press code

An 8-bit value is returned when a key is pressed.
0 = Unused.

See **keypadDef()** for default press codes.

cRelease is a key release code.

An 8-bit value is returned when a key is pressed.
0 = Unused.

cCntHold is a hold tick, which is approximately one debounce period or 5 µs.

How long to hold before repeating.
0 = No Repeat.

cSpdLo is a low-speed repeat tick, which is approximately one debounce period or 5 µs.

How many times to repeat.
0 = None.

cCntLo is a low-speed hold tick, which is approximately one debounce period or 5 µs.

How long to hold before going to high-speed repeat.
0 = Slow Only.

cSpdHi is a high-speed repeat tick, which is approximately one debounce period or 5 µs.

How many times to repeat after low speed repeat.
0 = None.

RETURN VALUE

None.

SEE ALSO

keyProcess, keyGet, keypadDef

void keyProcess(void);

Scans and processes keypad data for key assignment, debouncing, press and release, and repeat.

NOTE: This function is also able to process an 8 x 8 matrix keypad.

RETURN VALUE

None

SEE ALSO

keyConfig, keyGet, keypadDef

char keyGet(void);

Get next keypress.

RETURN VALUE

The next keypress, or 0 if none

SEE ALSO

keyConfig, keyProcess, keypadDef

int keyUnget(char cKey);

Pushes the value of **cKey** to the top of the input queue, which is 16 bytes deep.

PARAMETER

cKey

RETURN VALUE

None.

SEE ALSO

keyGet

void keypadDef();

Configures the physical layout of the keypad with the default ASCII return key codes.

Keypad physical mapping 1 x 7

0	4	1	5	2	6	3
['L']		['U']		['D']		['R']
	['–']		['+']		['E']	

where

'D' represents Down Scroll
'U' represents Up Scroll
'R' represents Right Scroll
'L' represents Left Scroll
'–' represents Page Down
'+' represents Page Up
'E' represents the ENTER key

Example: Do the followingfor the above physical vs. ASCII return key codes.

```
keyConfig ( 3,'R',0, 0, 0, 0, 0 );
keyConfig ( 6,'E',0, 0, 0, 0, 0 );
keyConfig ( 2,'D',0, 0, 0, 0, 0 );
keyConfig ( 4,'-',0, 0, 0, 0, 0 );
keyConfig ( 1,'U',0, 0, 0, 0, 0 );
keyConfig ( 5,'+',0, 0, 0, 0, 0 );
keyConfig ( 0,'L',0, 0, 0, 0, 0 );
```

Characters are returned upon keypress with no repeat.

RETURN VALUE

None.

SEE ALSO

keyConfig, keyGet, keyProcess

void keyScan(char *pcKeys);

Writes "1" to each row and reads the value. The position of a keypress is indicated by a zero value in a bit position.

PARAMETER

***pcKeys** is a pointer to the address of the value read.

RETURN VALUE

None.

SEE ALSO

keyConfig, keyGet, keypadDef, keyProcess

APPENDIX D. POWER SUPPLY

Appendix D provides information on the current requirements of the RCM3300/RCM3310, and includes some background on the reset generator.

D.1 Power Supplies

Power is supplied from the motherboard to which the RCM3300/RCM3310 is connected via header J4. The RCM3300/RCM3310 requires a regulated 3.15 V to 3.45 V DC power source. An RCM3300/RCM3310 with no loading at the outputs operating at 44.2 MHz typically draws 350 mA.

D.1.1 Battery-Backup Circuits

The RCM3300/RCM3310 does not have a battery, but there is provision for a customer-supplied battery to back up the data SRAM and keep the internal Rabbit 3000 real-time clock running.

Header J4, shown in Figure D-1, allows access to the external battery. This header makes it possible to connect an external 3 V power supply. This allows the SRAM and the internal Rabbit 3000 real-time clock to retain data with the RCM3300/RCM3310 powered down.

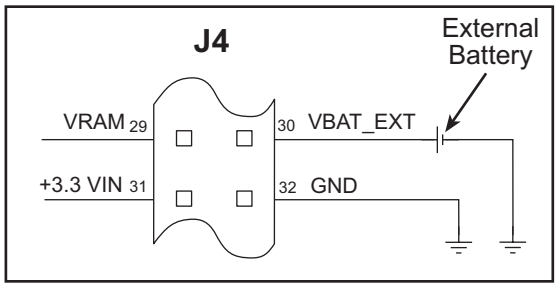


Figure D-1. External Battery Connections at Header J4

A lithium battery with a nominal voltage of 3 V and a minimum capacity of 165 mA·h is recommended. A lithium battery is strongly recommended because of its nearly constant nominal voltage over most of its life.

The drain on the battery by the RCM3300/RCM3310 is typically 6 µA when no other power is supplied. If a 165 mA·h battery is used, the battery can last about 3 years:

$$\frac{165\ \text{mA}\cdot\text{h}}{6\ \mu\text{A}} = 3.1\ \text{years}.$$

Note that the shelf life of a lithium ion battery is ultimately 10 years. The RCM3300/RCM3310 does not drain the battery while it is powered up normally.

D.1.2 Reset Generator

The RCM3300/RCM3310 uses a reset generator to reset the Rabbit 3000 microprocessor when the voltage drops below the voltage necessary for reliable operation. The reset occurs between 2.85 V and 3.00 V, typically 2.93 V.

The RCM3300/RCM3310 has a reset pin, pin 28 on header J4. This pin provides access to the reset input of the reset generator, whose output drives the reset input of the Rabbit 3000 and peripheral circuits. The /RESET output from the reset generator is available on pin 1 of header J4 on the RCM3300/RCM3310, and can be used to reset user-defined circuits on the motherboard on which the RCM3300/RCM3310 module is mounted.

APPENDIX E. RABBITNET

E.1 General RabbitNet Description

RabbitNet is a high-speed synchronous protocol developed by Z-World to connect peripheral cards to a master and to allow them to communicate with each other.

E.1.1 RabbitNet Connections

All RabbitNet connections are made point to point. A RabbitNet master port can only be connected directly to a peripheral card, and the number of peripheral cards is limited by the number of available RabbitNet ports on the master.

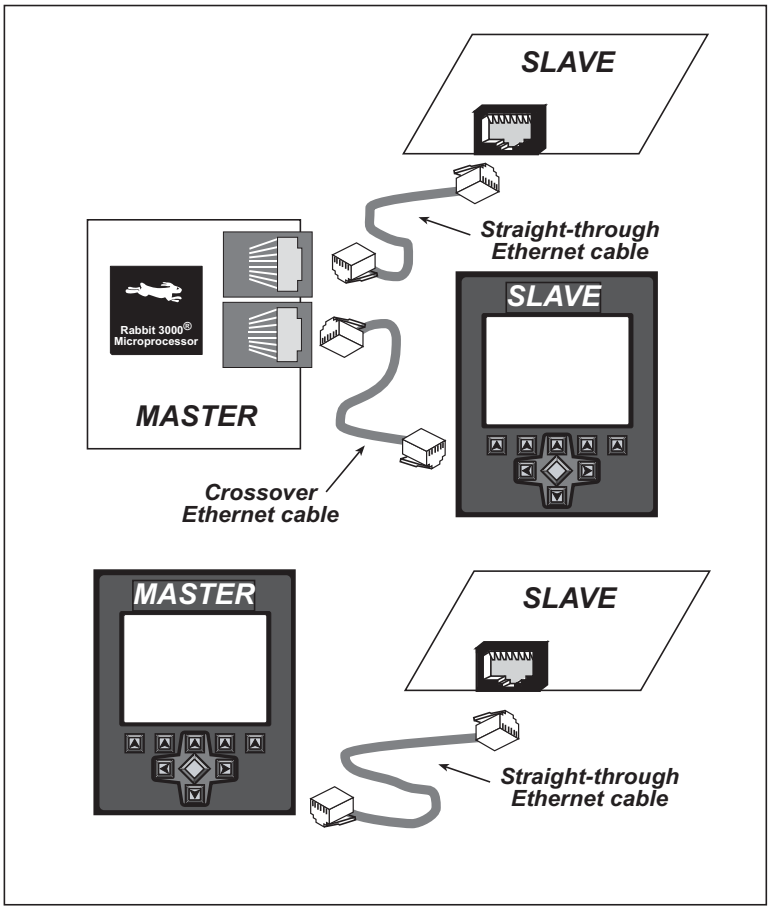


Figure E-1. Connecting Peripheral Cards to a Master

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.

E.1.2 RabbitNet Peripheral Cards

- Digital I/O

  24 inputs, 16 push/pull outputs, 4 channels of 10-bit A/D conversion with ranges of 0 to 10 V, 0 to 1 V, and -0.25 to +0.25 V. The following connectors are used:

  Signal = 0.1" friction-lock connectors
  Power = 0.156" friction-lock connectors
  RabbitNet = RJ-45 connector

- A/D converter

  8 channels of programmable-gain 12-bit A/D conversion, configurable as current measurement and differential-input pairs. 2.5 V reference voltage is available on the connector. The following connectors are used:

  Signal = 0.1" friction-lock connectors
  Power = 0.156" friction-lock connectors
  RabbitNet = RJ-45 connector

- D/A converter

  8 channels of 0–10 V 12-bit D/A conversion. The following connectors are used:

  Signal = 0.1" friction-lock connectors
  Power = 0.156" friction-lock connectors
  RabbitNet = RJ-45 connector

- Display/Keypad interface

  allows you to connect your own keypad with up to 64 keys and one character liquid crystal display from 1 × 8 to 4 × 40 characters with or without backlight using standard 1 × 16 or 2 × 8 connectors. The following connectors are used:

  Signal = 0.1" headers or sockets
  Power = 0.156" friction-lock connectors
  RabbitNet = RJ-45 connector

- Relay card

  6 relays rated at 250 V AC, 1200 V·A or 100 V DC up to 240 W. The following connectors are used:

  Relay contacts = screw-terminal connectors
  Power = 0.156" friction-lock connectors
  RabbitNet = RJ-45 connector

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

E.2 Physical Implementation

There are four signaling functions associated with a RabbitNet connection. From the master's point of view, the transmit function carries information and commands to the peripheral card. The receive function is used to read back information sent to the master by the peripheral card. A clock is used to synchronize data going between the two devices at high speed. The master is the source of this clock. A slave select (SS) function originates at the master, and when detected by a peripheral card causes it to become selected and respond to commands received from the master.

The signals themselves are differential RS-422, which are series-terminated at the source. With this type of termination, the maximum frequency is limited by the round-trip delay time of the cable. Although a peripheral card could theoretically be up to 45 m (150 ft) from the master for a data rate of 1 MHz, Z-World 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.

E.2.1 Control and Routing

Control starts at the master when the master asserts the slave select signal (SS). Then it simultaneously sends a serial command and clock. The first byte of a command contains the address of the peripheral card if more than one peripheral card is connected.

A peripheral card assumes it is selected as soon as it receives the select signal. For direct master-to-peripheral-card connections, this is as soon as the master asserts the select signal. The connection is established once the select signal reaches the addressed slave. At this point communication between the master and the selected peripheral card is established, and data can flow in both directions simultaneously. The connection is maintained so long as the master asserts the select signal.

E.3 Function Calls

The function calls described in this section are used with all RabbitNet peripheral cards, and are available in the **RNET.LIB** library in the Dynamic C **RABBITNET** folder.

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.

E.3.1 Status Byte

Unless otherwise specified, functions returning a status byte will have the following format for each designated bit.

7	6	5	4	3	2	1	0	
×	×							00 = Reserved 01 = Ready 10 = Busy 11 = Device not connected
		×						0 = Device 1 = Router
			×					0 = No error 1 = Communication error*
				×				Reserved for individual peripheral cards
					×			Reserved for individual peripheral cards
						×		0 = Last command accepted 1 = Last command unexecuted
							×	0 = Not expired 1 = HW or SW watchdog timer expired†

* Use the function **rn_comm_status()** to determine which error occurred.

† Use the function **rn_rst_status()** to determine which timer expired.

NOTICE TO USERS

Z-WORLD PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE-SUPPORT DEVICES OR SYSTEMS UNLESS A SPECIFIC WRITTEN AGREEMENT REGARDING SUCH INTENDED USE IS ENTERED INTO BETWEEN THE CUSTOMER AND Z-WORLD PRIOR TO USE. Life-support devices or systems are devices or systems intended for surgical implantation into the body or to sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling and user's manual, can be reasonably expected to result in significant injury.

No complex software or hardware system is perfect. Bugs are always present in a system of any size. In order to prevent danger to life or property, it is the responsibility of the system designer to incorporate redundant protective mechanisms appropriate to the risk involved.

All Z-World products are 100 percent functionally tested. Additional testing may include visual quality control inspections or mechanical defects analyzer inspections. Specifications are based on characterization of tested sample units rather than testing over temperature and voltage of each unit. Z-World products may qualify components to operate within a range of parameters that is different from the manufacturer's recommended range. This strategy is believed to be more economical and effective. Additional testing or burn-in of an individual unit is available by special arrangement.

INDEX

K

L

M

P

R

S

T

U

W

SCHEMATICS

090-0186 RCM3300/RCM3310 Schematic

www.rabbitsemiconductor.com/documentation/schemat/090-0186.pdf

090-0188 Prototyping Board Schematic

www.rabbitsemiconductor.com/documentation/schemat/090-0188.pdf

090-0156 LCD/Keypad Module Schematic

www.rabbitsemiconductor.com/documentation/schemat/090-0156.pdf

090-0128 Programming Cable Schematic

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

The schematics included with the printed manual were the latest revisions available at the time the manual was last revised. The online versions of the manual contain links to the latest revised schematic on the Web site. You may also use the URL information provided above to access the latest schematics directly.