



# **RabbitCore RCM4400W**

C-Programmable Wi-Fi Core Module

## **OEM User's Manual**

019-0160 • 090515-G

# **RabbitCore RCM4400W OEM User's Manual**

Part Number 019-0160 • 090515–G • Printed in U.S.A.

©2007–2009 Digi International Inc. • All rights reserved.

No part of the contents of this manual may be reproduced or transmitted in any form or by any means without the express written permission of Digi International.

Permission is granted to make one or more copies as long as the copyright page contained therein is included. These copies of the manuals may not be let or sold for any reason without the express written permission of Digi International.

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

## **Trademarks**

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

Wi-Fi is a registered trademark of the Wi-Fi Alliance.

Rabbit 4000 is a trademark of Digi International Inc.

The latest revision of this manual is available on the Rabbit Web site, [www.rabbit.com](http://www.rabbit.com), for free, unregistered download.

**Digi International Inc.**

[www.rabbit.com](http://www.rabbit.com)

# TABLE OF CONTENTS

<b>Chapter 1. Introduction</b>	<b>1</b>
1.1 RCM4400W Features .....	2
1.2 Advantages of the RCM4400W .....	3
1.3 Development and Evaluation Tools.....	4
1.3.1 RCM4400W Development Kit .....	4
1.3.2 Software .....	5
1.3.3 Online Documentation .....	5
1.4 Certifications.....	6
1.4.1 FCC Part 15 Class B .....	6
1.4.2 Industry Canada Labeling .....	7
1.4.3 Japan Labeling .....	8
1.4.4 Europe .....	8
<b>Chapter 2. Getting Started</b>	<b>9</b>
2.1 Install Dynamic C .....	9
2.2 Hardware Connections.....	10
2.2.1 Step 1 — Prepare the Prototyping Board for Development.....	10
2.2.2 Step 2 — Attach the Antenna to the RCM4400W Module .....	11
2.2.3 Step 3 — Attach Module to Prototyping Board.....	12
2.2.4 Step 4 — Connect Programming Cable.....	13
2.2.5 Step 5 — Connect Power .....	14
2.3 Run a Sample Program .....	15
2.3.1 Troubleshooting .....	16
2.4 Where Do I Go From Here? .....	17
2.4.1 Technical Support .....	17
<b>Chapter 3. Running Sample Programs</b>	<b>19</b>
3.1 Introduction.....	19
3.2 Sample Programs .....	20
3.2.1 Serial Communication.....	22
3.2.2 Real-Time Clock .....	25
3.2.3 Use of Serial Flash (Dynamic C v. 10.54 and later) .....	25
<b>Chapter 4. Hardware Reference</b>	<b>27</b>
4.1 RCM4400W Digital Inputs and Outputs .....	28
4.1.1 Memory I/O Interface .....	35
4.1.2 Other Inputs and Outputs .....	35
4.2 Serial Communication .....	36
4.2.1 Serial Ports .....	36
4.2.1.1 Using the Serial Ports.....	37
4.2.2 Wi-Fi .....	38
4.2.3 Programming Port.....	40
4.3 Programming Cable .....	41
4.3.1 Changing Between Program Mode and Run Mode .....	41
4.3.2 Standalone Operation of the RCM4400W .....	42

4.4 Other Hardware .....	43
4.4.1 Clock Doubler .....	43
4.4.2 Spectrum Spreader .....	43
4.5 Memory .....	44
4.5.1 SRAM .....	44
4.5.2 Flash EPROM .....	44
4.5.3 Serial Flash .....	44
<b>Chapter 5. Software Reference</b> .....	<b>45</b>
5.1 More About Dynamic C .....	45
5.2 Dynamic C Function Calls .....	47
5.2.1 Digital I/O .....	47
5.2.2 Serial Communication Drivers .....	47
5.2.3 User Block .....	47
5.2.4 SRAM Use .....	48
5.2.5 Wi-Fi Drivers .....	48
5.2.6 Serial Flash Drivers .....	48
5.2.7 Prototyping Board Function Calls .....	50
5.2.7.1 Board Initialization .....	50
5.2.7.2 Alerts .....	51
5.3 Upgrading Dynamic C .....	52
5.3.1 Add-On Modules .....	52
<b>Chapter 6. Using the Wi-Fi Features</b> .....	<b>53</b>
6.1 Introduction to Wi-Fi .....	53
6.1.1 Infrastructure Mode .....	53
6.1.2 Ad-Hoc Mode .....	54
6.1.3 Additional Information .....	54
6.2 Running Wi-Fi Sample Programs .....	55
6.2.1 Wi-Fi Setup .....	56
6.2.2 What Else You Will Need .....	57
6.2.3 Configuration Information .....	58
6.2.3.1 Network/Wi-Fi Configuration .....	58
6.2.3.2 PC/Laptop/PDA Configuration .....	59
6.2.4 Wi-Fi Sample Programs .....	61
6.2.4.1 Wi-Fi Operating Region Configuration .....	61
6.2.4.2 Wi-Fi Operation .....	63
6.2.5 RCM4400W Sample Programs .....	65
6.3 Dynamic C Wi-Fi Configurations .....	67
6.3.1 Configuring Dynamic C at Compile Time .....	67
6.3.2 Configuring Dynamic C at Run Time .....	71
6.3.3 Other Key Function Calls .....	81
6.4 Where Do I Go From Here? .....	82
<b>Appendix A. RCM4400W Specifications</b> .....	<b>83</b>
A.1 Electrical and Mechanical Characteristics .....	84
A.1.1 Antenna .....	88
A.1.2 Headers .....	89
A.2 Rabbit 4000 DC Characteristics .....	90
A.3 I/O Buffer Sourcing and Sinking Limit .....	91
A.4 Bus Loading .....	91
A.5 Conformal Coating .....	94
A.6 Jumper Configurations .....	95
<b>Appendix B. Prototyping Board</b> .....	<b>97</b>
B.1 Introduction .....	98
B.1.1 Prototyping Board Features .....	99
B.2 Mechanical Dimensions and Layout .....	101

B.3 Power Supply .....	102
B.4 Using the Prototyping Board.....	103
B.4.1 Adding Other Components.....	105
B.4.2 Measuring Current Draw .....	105
B.4.3 Analog Features.....	106
B.4.4 Serial Communication.....	106
B.4.4.1 RS-232 .....	106
B.5 Prototyping Board Jumper Configurations .....	108
<b>Appendix C. Power Supply</b> .....	<b>111</b>
C.1 Power Supplies .....	111
C.1.1 Battery-Backup.....	111
C.1.2 Battery-Backup Circuit.....	112
C.1.3 Reset Generator .....	113
C.1.4 Onboard Power Supplies .....	113
<b>Index</b> .....	<b>115</b>
<b>Schematics</b> .....	<b>119</b>





# 1. INTRODUCTION

The RCM4400W RabbitCore modules adds Wi-Fi/802.11b functionality to the existing Rabbit® 4000 microprocessor features to allow you to create a low-cost, low-power, embedded wireless control and communications solution for your embedded control system. The Rabbit® 4000 microprocessor features include hardware DMA, clock speeds of up to 60 MHz, I/O lines shared with up to six serial ports and four levels of alternate pin functions that include variable-phase PWM, auxiliary I/O, quadrature decoder, and input capture. Coupled with more than 500 new opcode instructions that help to reduce code size and improve processing speed, this equates to a core module that is fast, efficient, and the ideal solution for a wide range of wireless embedded applications.

The Development Kit has the essentials that you need to design your own wireless microprocessor-based system, and includes a complete Dynamic C software development system. This Development Kit also contains a Prototyping Board that will allow you to evaluate the RCM4400W RabbitCore modules and to prototype circuits that interface to the RCM4400W modules. You will also be able to write and test software for these modules.

In addition to onboard Wi-Fi/802.11b functionality, the RCM4400W model has a Rabbit 4000 microprocessor operating at 58.98 MHz, static RAM, flash memory, two clocks (main oscillator and timekeeping), and the circuitry necessary for reset and management of battery backup of the Rabbit 4000's internal real-time clock and the static RAM. One 50-pin header brings out the Rabbit 4000 I/O bus lines, parallel ports, and serial ports.

The RCM4400W series receives its +3.3 V power from the customer-supplied motherboard on which it is mounted. The RCM4400W series can interface with many CMOS-compatible digital devices through the motherboard.

## 1.1 RCM4400W Features

- Small size: 1.84" × 2.85" × 0.50"  
(47 mm × 72 mm × 13 mm)
- Microprocessor: Rabbit 4000 running at 58.98 MHz
- Up to 35 general-purpose I/O lines configurable with up to four alternate functions
- 3.3 V I/O lines with low-power modes down to 2 kHz
- Six CMOS-compatible serial ports — four ports are configurable as a clocked serial port (SPI), and two ports are configurable as SDLC/HDLC serial ports.
- Alternate I/O bus can be configured for 8 data lines and 6 address lines (shared with parallel I/O lines), I/O read/write
- 512KB flash memory, 512KB data SRAM, 512KB fast program-execution SRAM
- UBEC single-chip 802.11b transceiver
- Real-time clock
- Watchdog supervisor

Currently there is one RCM4400W production model. Table 1 summarizes its main features.

**Table 1. RCM4400W Features**

Feature	RCM4400W
Microprocessor	Rabbit® 4000 at 58.98 MHz
Flash Memory	512KB
Data SRAM	512KB
Fast Program-Execution SRAM	512KB
Serial Ports	6 shared high-speed, CMOS-compatible ports: 6 are configurable as asynchronous serial ports; 4 are configurable as clocked serial ports (SPI); 2 are configurable as SDLC/HDLC serial ports; 1 asynchronous serial port is used during programming
Wi-Fi	802.11b standard, ISM 2.4 GHz

**NOTE:** There is a special version of the RCM4400W RabbitCore module for Japan. It is functionally identical to the standard RCM4400W module and uses the same components, but has been assembled to meet the Japan regulatory requirements. Be sure to order the correct version for the market where you plan to use the RCM4400W. The two versions can be distinguished by the labels on the RF shield as shown below.



**Standard Release Label**



**Japan Version Label**



The RCM4400W series is programmed over a standard PC USB port through a programming cable supplied with the Development Kit.

**NOTE:** The RabbitLink cannot be used to program RabbitCore modules based on the Rabbit 4000 microprocessor.

Appendix A provides detailed specifications for the RCM4400W.

## **1.2 Advantages of the RCM4400W**

- Fast time to market using a fully engineered, “ready-to-run/ready-to-program” microprocessor core module.
- Competitive pricing when compared with the alternative of purchasing and assembling individual components.
- Easy C-language program development and debugging
- Rabbit Field Utility to download compiled Dynamic C .bin files, 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.
- Easily scalable for commercial deployment applications

## 1.3 Development and Evaluation Tools

### 1.3.1 RCM4400W Development Kit

The RCM4400W Development Kit contains the hardware essentials you will need to use the RCM4400W module. The items in the Development Kit and their use are as follows.

- RCM4400W module with 2.4 GHz *bec* whip dipole antenna.
- Prototyping Board.
- Universal AC adapter, 12 V DC, 1 A (includes Canada/Japan/U.S., Australia/N.Z., U.K., and European style plugs). Development Kits sold in North America may contain an AC adapter with only a North American style plug.
- USB programming cable with 10-pin header.
- 10-pin header to DB9 serial cable.
- *Dynamic C*<sup>®</sup> CD-ROM, with complete product documentation on disk.
- *Getting Started* instructions.
- A bag of accessory parts for use on the Prototyping Board.
- *Rabbit 4000 Processor Easy Reference* poster.
- Registration card.

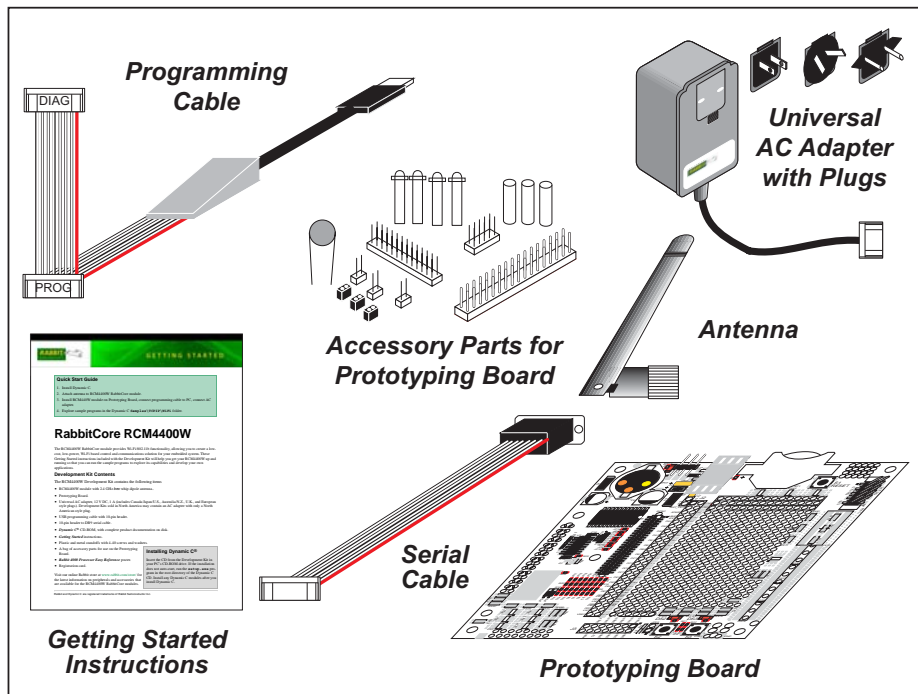


Figure 1. RCM4400W Development Kit

### 1.3.2 Software

The RCM4400W is programmed using version 10.21 or later of Dynamic C. A compatible version is included on the Development Kit CD-ROM.

RCM4400W RabbitCore modules labelled “For development use only” may be used with Dynamic C v. 10.11, but any applications developed using Dynamic C v. 10.11 will have to be recompiled with a future version of Dynamic C in order to run and meet regulatory requirements on production modules carrying the FCC certification markings. These “For development use only” modules were only sold in 2007.

Rabbit also offers add-on Dynamic C modules containing the popular  $\mu$ C/OS-II real-time operating system, the FAT file system, as well as PPP, Advanced Encryption Standard (AES), and other select libraries. In addition to the Web-based technical support included at no extra charge, a one-year telephone-based technical support module is also available for purchase. Visit our Web site at [www.rabbit.com](http://www.rabbit.com) or contact your Rabbit sales representative or authorized distributor for further information.

### 1.3.3 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.

## 1.4 Certifications

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

Only RCM4400W modules bearing the FCC certification are certified for use in Wi-Fi enabled end devices, and any applications must have been compiled using Dynamic C v. 10.21 or later. The certification is valid only for RCM4400W modules equipped with the dipole antenna that is included with the modules. Changes or modifications to this equipment not expressly approved by Rabbit may void the user's authority to operate this equipment.

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

**NOTE:** Any regulatory certification is voided if the RF shield on the RCM4400W module is removed.

### 1.4.1 FCC Part 15 Class B

The RCM4400W RabbitCore module has been tested and found to comply with the limits for Class B digital devices pursuant to Part 15 Subpart B, of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential environment. This equipment generates, uses, and can radiate radio frequency energy, and if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try and correct the interference by one or more of the following measures:

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

## Labeling Requirements (FCC 15.19)

FCC ID: VCB-540D144

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

- (1) this device may not cause harmful interference, and
- (2) this device must accept any interference received, including interference that may cause undesired operation.



The modular transmitter must be equipped with either a permanently affixed label or must be capable of displaying the FCC identification number electronically.

If using a permanently affixed label, the modular transmitter must be labeled with its own FCC identification number, and, if the FCC identification number is not visible when the module is installed inside another device, then the outside of the device into which the module is installed must also display a label referring to the enclosed module. This exterior label can use wording such as the following: “Contains Transmitter Module FCC ID: VCB-540D144” or “Contains FCC ID: VCB-540D144.” Any similar wording that expresses the same meaning may be used.

The following caution must be included with documentation for any device incorporating the RCM4400W RabbitCore module.

### **Caution — Exposure to Radio-Frequency Radiation.**

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

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

## 1.4.2 Industry Canada Labeling





Industry Industrie  
Canada Canada

ID: 7143A-540D144

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

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

### 1.4.3 Japan Labeling

		<b>Model Name</b> — <i>Use Your Company Model</i>
<b>ID Number</b> — <b>003WW071090000</b>		
<b>Company Name</b> — <i>Use Your Company Name</i>		



The logo mark diameter must be 5 mm or bigger.

If the equipment is 100 cm<sup>3</sup> or smaller in volume, the minimum size of the logo mark is 3 mm.

### 1.4.4 Europe

The marking shall include as a minimum:

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

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

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

### Regulatory Marking

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

## 2. GETTING STARTED

This chapter describes the RCM4400W hardware in more detail, and explains how to set up and use the accompanying Prototyping Board.

**NOTE:** This chapter (and this manual) assume that you have the RCM4400W Development Kit. If you purchased an RCM4400W 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 RCM4400W series of modules (and for all other Rabbit hardware), you must install and use Dynamic C.

If you have not yet installed Dynamic C version 10.11 (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 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 new icons on your PC desktop. One icon is for Dynamic C, another 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. Prepare the Prototyping Board for Development.
2. Attach the antenna to the RCM4400W module.
3. Attach the RCM4400W module to the Prototyping Board.
4. Connect the programming cable between the RCM4400W and the PC.
5. Connect the power supply to the Prototyping Board.



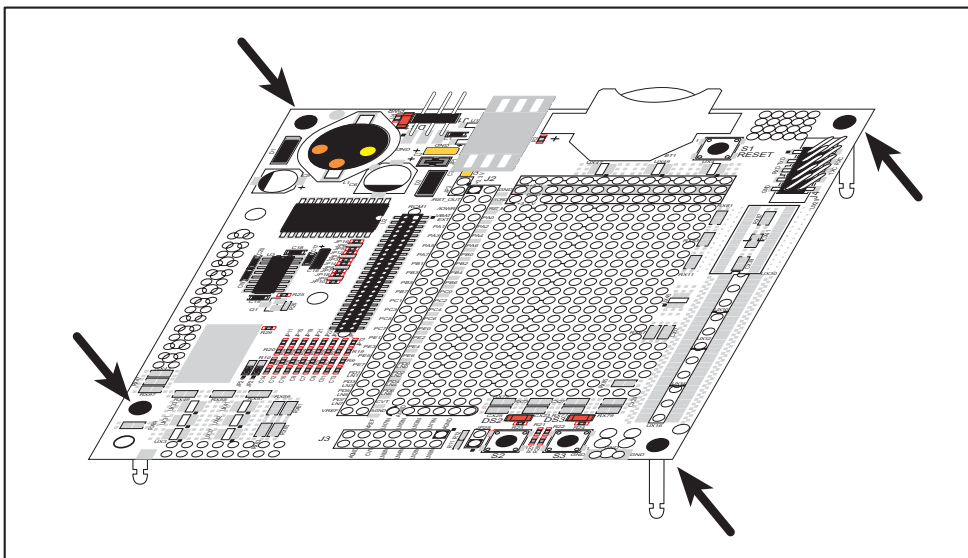
**CAUTION:** Provide ESD protection such as smocks and grounding straps on your footwear while assembling the RCM4400W module, installing it on another board, and while making or removing any connections.

Remember to use ESD protection regardless of whether you are working with the RCM4400W module on the Prototyping Board or in your own OEM application.

### 2.2.1 Step 1 — Prepare the Prototyping Board for Development

Snap in four of the plastic standoffs supplied in the bag of accessory parts from the Development Kit in the holes at the corners as shown in Figure 2.

**NOTE:** Pay attention to use the hole that is pointed out towards the bottom left of the Prototyping Board since the hole below it is used for a standoff when mounting the RCM4400W on the Prototyping Board.

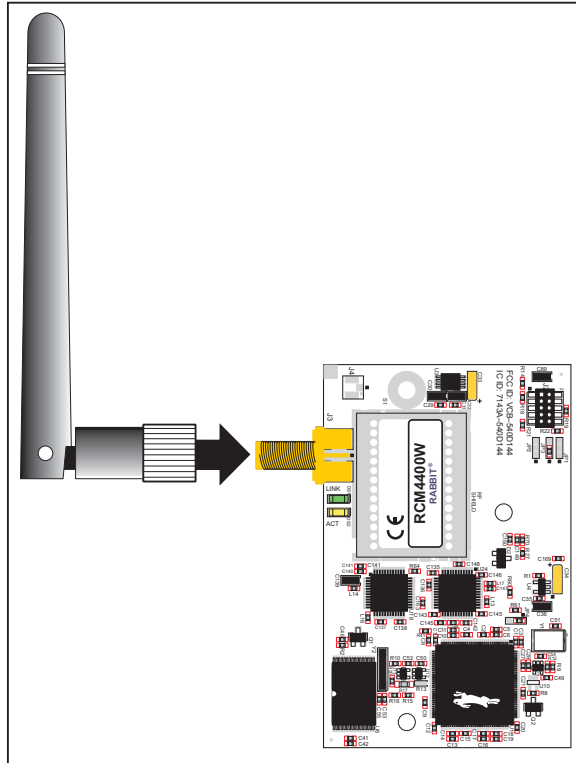


**Figure 2. Insert Standoffs**



### 2.2.2 Step 2 — Attach the Antenna to the RCM4400W Module

Attach the antenna to the antenna SMA connector on the RCM4400W as shown in Figure 3.



**Figure 3. Attach the Antenna to the RCM4400W Module**

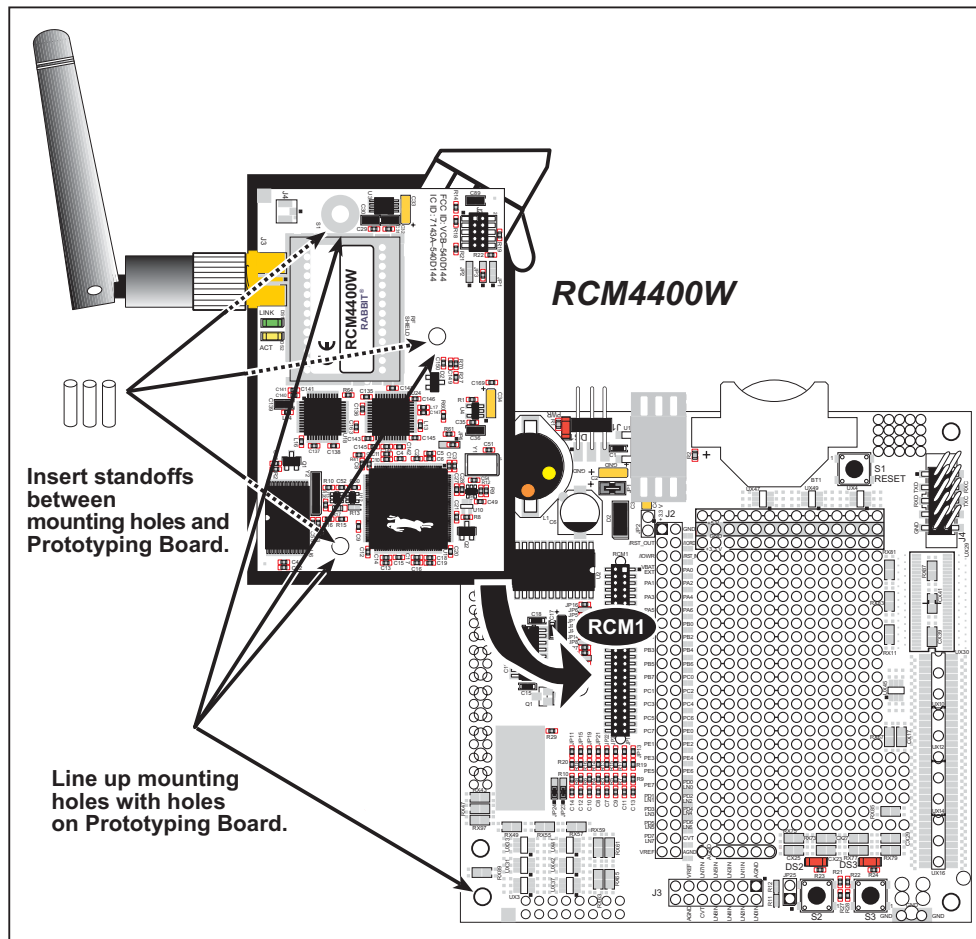


**CAUTION:** Do *not* remove the RF shield by the antenna since any attempt to remove the shield will damage the RF circuits underneath it.

Any regulatory certification is voided if the RF shield on the RCM4400W module is removed.

### 2.2.3 Step 3 — Attach Module to Prototyping Board

Turn the RCM4400W module so that the mounting holes line up with the corresponding holes on the Prototyping Board. Insert the metal standoffs as shown in Figure 4, secure them from the bottom using the 4-40 × 3/16 screws and washers, then insert the module's header J1 on the bottom side into socket RCM1 on the Prototyping Board.



**Figure 4. Install the Module on the Prototyping Board**

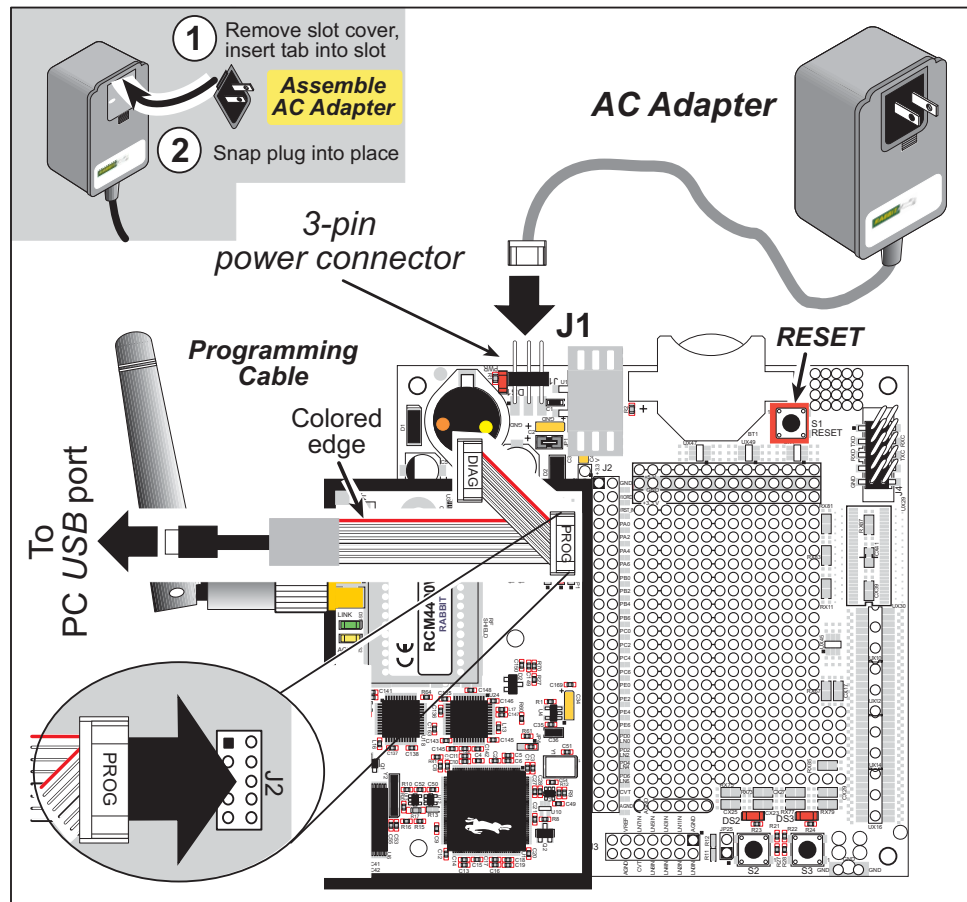
**NOTE:** It is important that you line up the pins on header J1 of the module exactly with socket RCM1 on the Prototyping Board. The header pins may become bent or damaged if the pin alignment is offset, and the module will not work. Permanent electrical damage to the module may also result if a misaligned module is powered up.

Press the module's pins gently into the Prototyping Board socket—press down in the area above the header pins. For additional integrity, you may secure the RCM4400W to the standoffs from the top using the remaining three screws and washers.

## 2.2.4 Step 4 — Connect Programming Cable

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

Connect the 10-pin connector of the programming cable labeled **PROG** to header J2 on the RCM4400W as shown in Figure 5. 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 normal serial connection.)



**Figure 5. Connect Programming Cable and Power Supply**

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

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

Your PC should recognize the new USB hardware, and the LEDs in the shrink-wrapped area of the USB programming cable will flash — if you get an error message, you will have to install USB drivers. Drivers for Windows XP are available in the Dynamic C **Drivers\Rabbit USB Programming Cable\WinXP\_2K** folder — double-click **DPInst.exe** to install the USB drivers. Drivers for other operating systems are available online at [www.ftdichip.com/Drivers/VCP.htm](http://www.ftdichip.com/Drivers/VCP.htm).

### 2.2.5 Step 5 — Connect Power

Once all the other connections have been made, you can connect power to the Prototyping Board.

If you have the universal AC adapter, prepare the AC adapter for the country where it will be used by selecting the appropriate plug. Snap in the top of the plug assembly into the slot at the top of the AC adapter as shown in Figure 5, then press down on the plug until it clicks into place.

Connect the AC adapter to 3-pin header J1 on the Prototyping Board as shown in Figure 5 above. The connector may be attached either way as long as it is not offset to one side—the center pin of J1 is always connected to the positive terminal, and either edge pin is ground.

Plug in the AC adapter. The **PWR** LED on the Prototyping Board next to the power connector at J1 should light up. The RCM4400W and the Prototyping Board are now ready to be used.

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

To power down the Prototyping Board, unplug the power connector from J1. You should disconnect power before making any circuit adjustments in the prototyping area, changing any connections to the board, or removing the RCM4400W from the Prototyping Board.

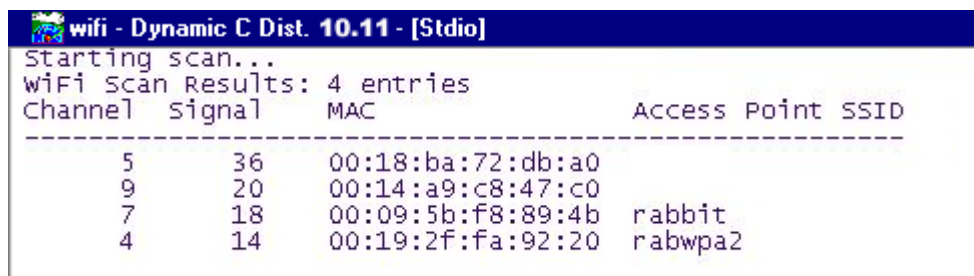
## 2.3 Run a Sample Program

If you already have Dynamic C installed, you are now ready to test your programming connections by running a sample program. Start Dynamic C by double-clicking on the Dynamic C icon on your desktop or in your **Start** menu. Select **Code and BIOS in Flash, Run in RAM** on the “Compiler” tab in the Dynamic C **Options > Project Options** menu. Then click on the “Communications” tab and verify that **Use USB to Serial Converter** is selected to support the USB programming cable. Click **OK**.

Determine which COM port was assigned to the USB programming cable on your PC. Open **Control Panel > System > Hardware > Device Manager > Ports** and identify which COM port is used for the USB connection. In Dynamic C, select **Options > Project Options**, then select this COM port on the **Communications** tab, then click **OK**. You may type the COM port number followed by **Enter** on your computer keyboard if the COM port number is outside the range on the dropdown menu.

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

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

The screenshot shows a window titled "wifi - Dynamic C Dist. 10.11 - [Stdio]". The text inside the window reads: "Starting scan...", "WiFi Scan Results: 4 entries", followed by a table with four columns: "Channel", "signal", "MAC", and "Access Point SSID". The table contains four rows of data.

Channel	signal	MAC	Access Point SSID
5	36	00:18:ba:72:db:a0	
9	20	00:14:a9:c8:47:c0	
7	18	00:09:5b:f8:89:4b	rabbit
4	14	00:19:2f:fa:92:20	rabwpa2

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

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

### 2.3.1 Troubleshooting

If you receive the message **Could Not Open Serial Port**, check that the COM port assigned to the USB programming cable was identified and set up in Dynamic C as described in the preceding section.

If you receive the message **No Rabbit Processor Detected**, the programming cable may be connected to the wrong COM port, a connection may be faulty, or the target system may not be powered up. First, check to see that the power LED on the Prototyping Board is lit. If the LED is lit, check both ends of the programming cable to ensure that it is firmly plugged into the PC and the programming header on the RCM4400W with the marked (colored) edge of the programming cable towards pin 1 of the programming header. Ensure that the module is firmly and correctly installed in its connectors on the Prototyping Board.

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

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

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

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

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

## 2.4 Where Do I Go From Here?

If the sample program ran fine, you are now ready to go on to the sample programs in Chapter 3 and to develop your own applications. The sample programs can be easily modified for your own use. The user's manual also provides complete hardware reference information and software function calls for the RCM4400W series of modules and the Prototyping Board.

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

### 2.4.1 Technical Support

**NOTE:** If you purchased your RCM4400W through a distributor or through a Rabbit partner, contact the distributor or partner first for technical support.

If there are any problems at this point:

- Use the Dynamic C **Help** menu to get further assistance with Dynamic C.
- Check the Rabbit Technical Bulletin Board and forums at [www.rabbit.com/support/bb/](http://www.rabbit.com/support/bb/) and at [www.rabbit.com/forums/](http://www.rabbit.com/forums/).
- Use the Technical Support e-mail form at [www.rabbit.com/support/](http://www.rabbit.com/support/).





## 3. RUNNING SAMPLE PROGRAMS

To develop and debug programs for the RCM4400W (and for all other Rabbit hardware), you must install and use Dynamic C. This chapter provides a tour of its major features with respect to the RCM4400W.

### 3.1 Introduction

To help familiarize you with the RCM4400W modules, Dynamic C includes several sample programs. Loading, executing and studying these programs will give you a solid hands-on overview of the RCM4400W's capabilities, as well as a quick start with Dynamic C as an application development tool.

This chapter provides sample programs that illustrate the digital I/O and serial capabilities of the RCM4400W RabbitCore module. Section 6.2.4 discusses the sample programs that illustrate the Wi-Fi features.

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

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

1. Your module 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 module to your PC.
4. Power must be applied to the module 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 compile and run it by pressing **F9**.

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

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

## 3.2 Sample Programs

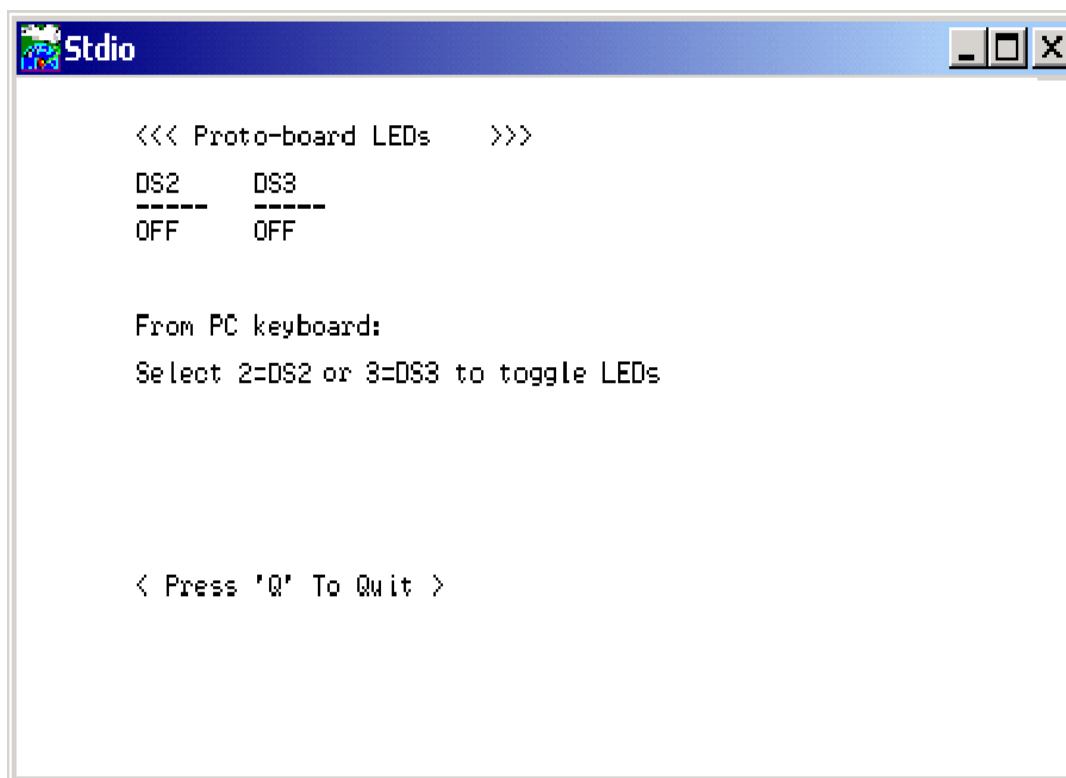
Of the many sample programs included with Dynamic C, several are specific to the RCM4400W modules. These programs will be found in the **SAMPLES\RCM4400W** folder.

- **CONTROLLED.C**—Demonstrates use of the digital outputs by having you turn LEDs DS2 and DS3 on the Prototyping Board on or off from the **STDIO** window on your PC.

Parallel Port B bit 2 = LED DS2

Parallel Port B bit 3 = LED DS3

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



Press “2” or “3” on your keyboard to select LED DS2 or DS3 on the Prototyping Board. Then follow the prompt in the Dynamic C **STDIO** window to turn the LED ON or OFF. A logic low will light up the LED you selected.

- **FLASHLED1.C**—demonstrates the use of assembly language to flash LEDs DS2 and DS3 on the Prototyping Board at different rates. Once you have compiled and run this program, LEDs DS2 and DS3 will flash on/off at different rates.
- **FLASHLED2.C**—demonstrates the use of cofunctions and costatements to flash LEDs DS2 and DS3 on the Prototyping Board at different rates. Once you have compiled and run this program, LEDs DS2 and DS3 will flash on/off at different rates.

- **TAMPERDETECTION.C**—demonstrates how to detect an attempt to enter the bootstrap mode. When an attempt is detected, the battery-backed onchip-encryption RAM on the Rabbit 4000 is erased. This battery-backed onchip-encryption RAM can be useful to store data such as an AES encryption key from a remote location.

This sample program shows how to load and read the battery-backed onchip-encryption RAM and how to enable a visual indicator.

Once this sample is compiled and running (you pressed the **F9** key while the sample program is open), remove the programming cable and press the reset button on the Prototyping Board to reset the module. LEDs DS2 and DS3 will be flashing on and off.

Now press switch S2 to load the battery-backed RAM with the encryption key. The LEDs are now on continuously. Notice that the LEDs will stay on even when you press the reset button on the Prototyping Board.

Reconnect the programming cable briefly and unplug it again to simulate an attempt to access the onchip-encryption RAM. The LEDs will be flashing because the battery-backed onchip-encryption RAM has been erased. Notice that the LEDs will continue flashing even when you press the reset button on the Prototyping Board.

You may press switch S2 again and repeat the last steps to watch the LEDs.

- **TOGGLESWITCH.C**—demonstrates the use of costatements to detect switch presses using the press-and-release method of debouncing. LEDs DS2 and DS3 on the Prototyping Board are turned on and off when you press switches S2 and S3. S2 and S3 are controlled by PB4 and PB5 respectively.

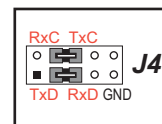
Once you have loaded and executed these five programs and have an understanding of how Dynamic C and the RCM4400W modules interact, you can move on and try the other sample programs, or begin building your own.

### 3.2.1 Serial Communication

The following sample programs are found in the **SAMPLES\RCM4400W\SERIAL** folder.

- **FLOWCONTROL.C**—This program demonstrates how to configure Serial Port D for CTS/RTS flow control with serial data coming from Serial Port C (TxC) at 115,200 bps. The serial data received are displayed in the **STDIO** window.

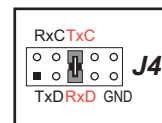
To set up the Prototyping Board, you will need to tie TxD and RxD together on the RS-232 header at J4, and you will also tie TxC and RxC together using the jumpers supplied in the Development Kit as shown in the diagram.



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 flow control.

If you have two Prototyping Boards with modules, run this sample program on the sending board, then disconnect the programming cable and reset the sending board so that the module is operating in the Run mode. Connect TxC, TxD, and GND on the sending board to RxC, RxD, and GND on the other board, then, with the programming cable attached to the other module, run the sample program.

- **PARITY.C**—This program demonstrates the use of parity modes by repeatedly sending byte values 0–127 from Serial Port C to Serial Port D. The program will switch between generating parity or not on Serial Port C. Serial Port D 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 TxC and RxD together on the RS-232 header at J4 using one of the jumpers supplied in the Development Kit as shown in the diagram.

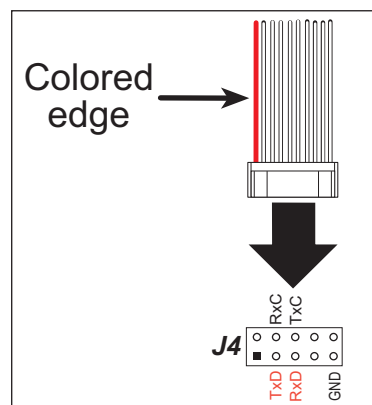
The Dynamic C **STDIO** window will display the error sequence.

- **SERDMA.C**—This program demonstrates using DMA to transfer data from a circular buffer to the serial port and vice versa. The Dynamic C **STDIO** window is used to view or clear the buffer.

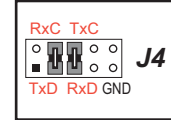
Before you compile and run the sample program, you will need to connect the RS-232 header at J4 to your PC as shown in the diagram using the serial to DB9 cable supplied in the Development Kit.

Once you have compiled and run the sample program, start Tera Term or another terminal emulation program to connect to the selected PC serial port at a baud rate of 115,200 bps. You can observe the output in the Dynamic C **STDIO** window as you type in Tera Term, and you can also use the Dynamic C **STDIO** window to clear the buffer.

The Tera Term utility can be downloaded from <http://vector.co.jp/authors/VA002416/teraterm.html>.



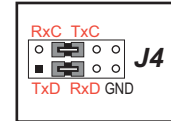
- **SIMPLE3WIRE.C**—This program demonstrates basic RS-232 serial communication. Lower case characters are sent on TxC, and are received by RxD. The received characters are converted to upper case and are sent out on TxD, are received on RxC, and are displayed in the Dynamic C **STDIO** window.



To set up the Prototyping Board, you will need to tie TxD and RxC together on the RS-232 header at J4, and you will also tie RxD and TxC together using the jumpers supplied in the Development Kit as shown in the diagram.

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

To set up the Prototyping Board, you will need to tie TxD and RxD together on the RS-232 header at J4, and you will also tie TxC and RxC together using the jumpers supplied in the Development Kit as shown in the diagram.

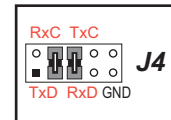


Once you have compiled and run this program, you can test flow control by disconnecting the TxD jumper from RxD while the program is running. Characters will no longer appear in the **STDIO** window, and will display again once TxD is connected back to RxD.

If you have two Prototyping Boards with modules, run this sample program on the sending board, then disconnect the programming cable and reset the sending board so that the module is operating in the Run mode. Connect TxC, TxD, and GND on the sending board to RxC, RxD, and GND on the other board, then, with the programming cable attached to the other module, run the sample program. Once you have compiled and run this program, you can test flow control by disconnecting TxD from RxD as before while the program is running. Since the J4 header locations on the two Prototyping Boards are connected with wires, there are no slip-on jumpers at J4 on either Prototyping Board.

- **SWITCHCHAR.C**—This program demonstrates transmitting and then receiving an ASCII string on Serial Ports C and D. 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 TxD and RxC together on the RS-232 header at J4, and you will also tie RxD and TxC together using the jumpers supplied in the Development Kit as shown in the diagram.



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

- **IOCONFIG\_SWITCHECHO.C**—This program demonstrates how to set up Serial Ports E and F, which then transmit and then receive an ASCII string when switch S2 or S3 is pressed. The echoed serial data are displayed in the Dynamic C **STDIO** window.

Note that the I/O lines that carry the Serial Port E and F signals are not the Rabbit 4000 defaults. The Serial Port E and F I/O lines are configured by calling the library function **serEFconfig()** that was generated by the Rabbit 4000 **IOCONFIG.EXE** utility program found in the Dynamic C **Utilities** folder.

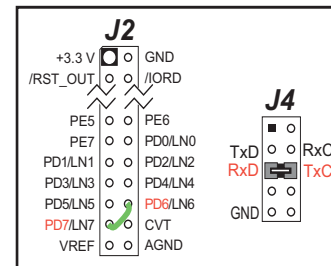
Note that the **RCM4400W\_IOCONFIG.LIB** library generated by **IOCONFIG.EXE** to support this sample program is provided in the Dynamic C **SAMPLES\RCM4400W\SERIAL** folder.

Serial Port E is configured to use Parallel Port D bits PD6 and PD7. These signals are available on the Prototyping Board's Module Extension Header (header J2).

Serial Port F is configured to use Parallel Port C bits PC2 and PC3. These signals are available on the Prototyping Board's RS-232 connector (header J4).

Serial Port D is left in its default configuration, using Parallel Port C bits PC0 and PC1. These signals are available on the Prototyping Board's RS-232 connector (header J4). Serial Port D transmits and then receives an ASCII string with Serial Port F when switch S3 is pressed.

To set up the Prototyping Board, you will need to tie Tx<sub>C</sub> and Rx<sub>D</sub> together on the RS-232 header at J4 using the jumpers supplied in the Development Kit; you will also tie Tx<sub>E</sub> (PD6) and Rx<sub>E</sub> (PD7) together with a soldered wire or with a wire jumper if you have soldered in the IDC header supplied with the accessory parts in the Development Kit.



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

### 3.2.2 Real-Time Clock

If you plan to use the real-time clock functionality in your application, you will need to set the real-time clock. Use the **SETRTCKB.C** sample program from the Dynamic C **SAMPLES\RTCKLOCK** folder, and follow the onscreen prompts. The **RTC\_TEST.C** sample program in the Dynamic C **SAMPLES\RTCKLOCK** folder provides additional examples of how to read and set the real-time clock.

### 3.2.3 Use of Serial Flash (Dynamic C v. 10.54 and later)

The following sample programs from the **SAMPLES\RCM4400W\Serial\_Flash** folder may be used with the RCM4400W as long as you are using Dynamic C v. 10.54 or later.

- **SERIAL\_FLASHLOG.C**—This program runs a simple Web server and stores a log of hits on the home page of the serial flash “server.” This log can be viewed and cleared from a browser at <http://10.10.6.100/>. You will likely have to first “configure” your network interface card for a “10Base-T Half-Duplex,” “100Base-T Half-Duplex,” or an “Auto-Negotiation” connection on the “Advanced” tab, which is accessed from the control panel (**Start > Settings > Control Panel**) by choosing **Network Connections**.

**NOTE:** This sample program accesses the serial flash directly and may overwrite other data stored in the user-accessible 800KB area, including the FAT file system. Do not run this sample program if you have any important data on the serial flash device.

- **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 five different commands to print out the contents of a specified page, set all bytes on the specified page to a single random value, clear (set to zero) all the bytes in a specified page, set all bytes on the specified page to a given value, or save user-specified text to a selected page.

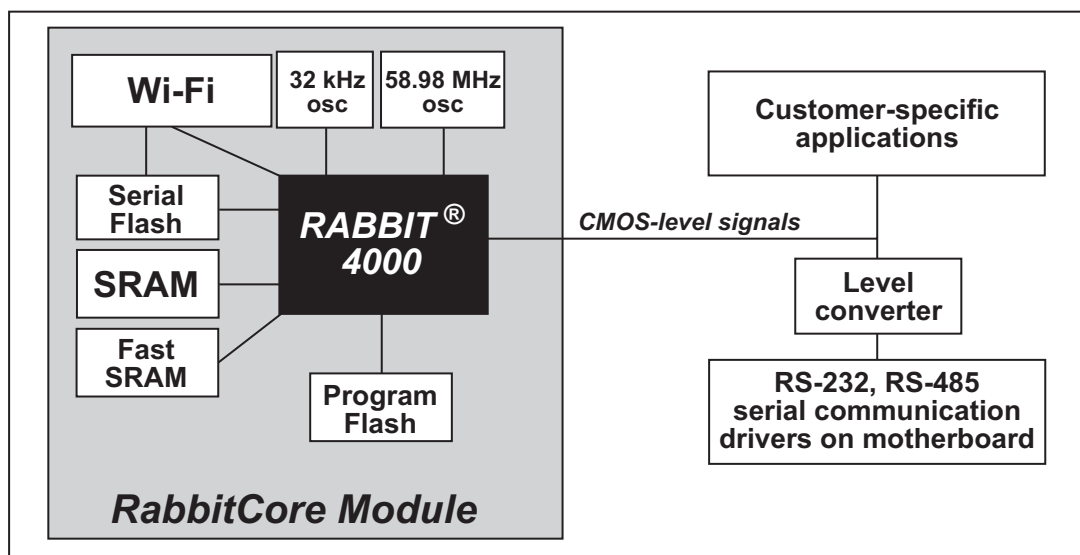




## 4. HARDWARE REFERENCE

Chapter 4 describes the hardware components and principal hardware subsystems of the RCM4400W. Appendix A, “RCM4400W Specifications,” provides complete physical and electrical specifications.

Figure 6 shows the Rabbit-based subsystems designed into the RCM4400W.

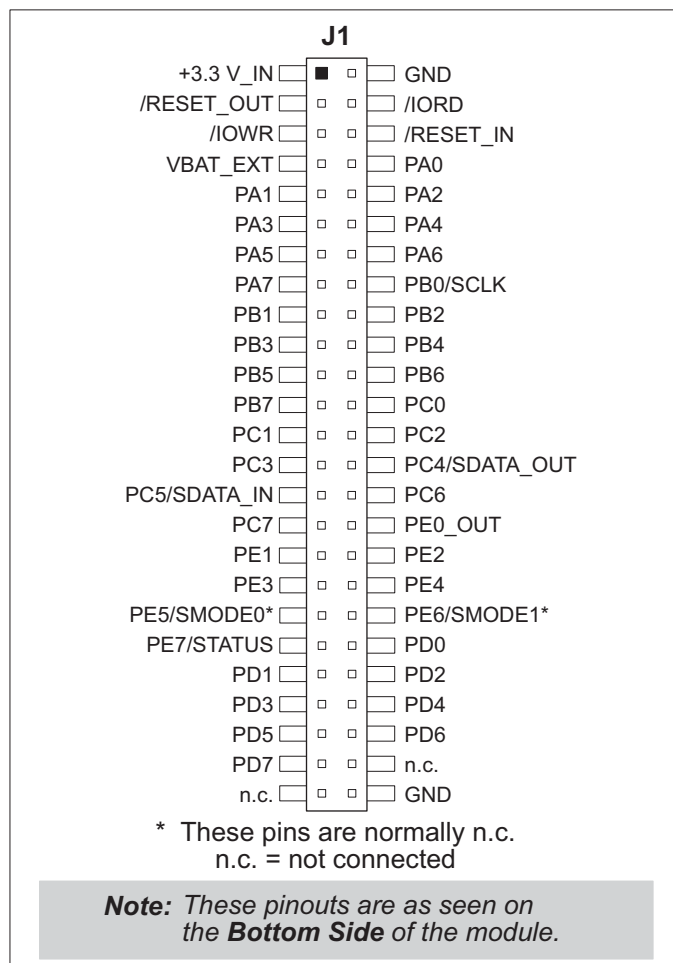


**Figure 6. RCM4400W Subsystems**

The 58.98 MHz frequency shown for the RCM4400W is generated using a 29.49 MHz crystal with the Rabbit 4000 clock doubler enabled.

## 4.1 RCM4400W Digital Inputs and Outputs

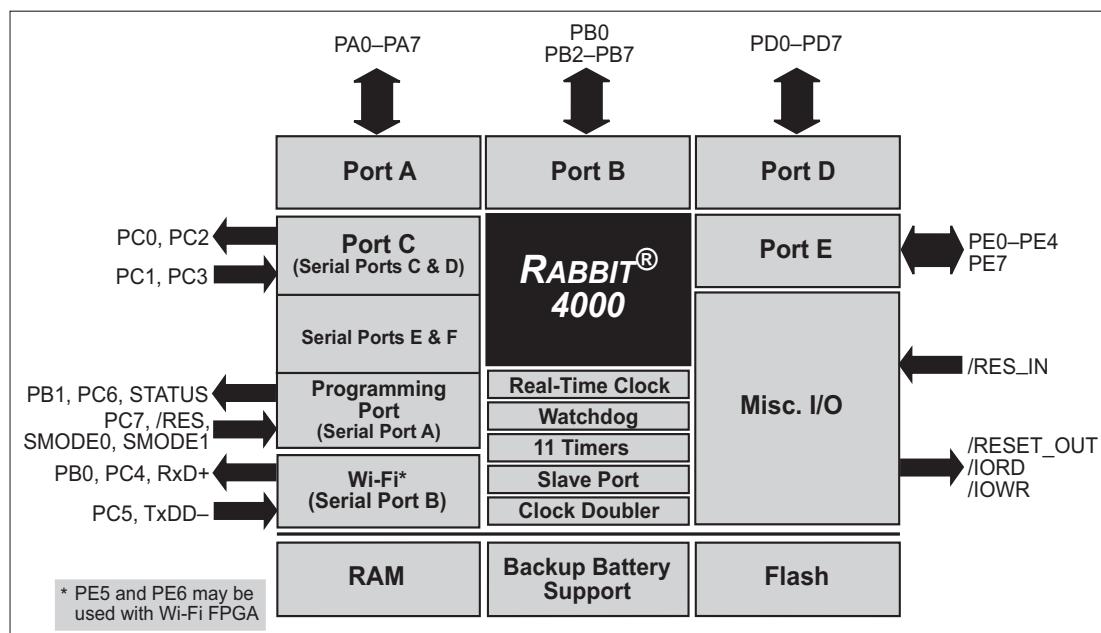
Figure 7 shows the RCM4400W pinouts for header J1.



**Figure 7. RCM4400W Pinout**

Headers J1 is a standard  $2 \times 25$  IDC header with a nominal 1.27 mm pitch.

Figure 8 shows the use of the Rabbit 4000 microprocessor ports in the RCM4400W modules.



**Figure 8. Use of Rabbit 4000 Ports**

The ports on the Rabbit 4000 microprocessor used in the RCM4400W are configurable, and so the factory defaults can be reconfigured. Table 2 lists the Rabbit 4000 factory defaults and the alternate configurations.

**Table 2. RCM4400W Pinout Configurations**

Pin		Pin Name	Default Use	Alternate Use	Notes
Header J1	1	+3.3 V_IN			
	2	GND			
	3	/RES_OUT	Reset output	Reset input	Reset output from Reset Generator or external reset input
	4	/IORD	Output		External I/O read strobe
	5	/IOWR	Output		External I/O write strobe
	6	/RESET_IN	Input		Input to Reset Generator
	7	VBAT_EXT	Battery input		
	8–15	PA[0:7]	Input/Output	Slave port data bus (SD0–SD7) External I/O data bus (ID0–ID7)	
	16	PB0	Input/Output	SCLK External I/O Address IA6	SCLKB (shared with serial flash)
	17	PB1	Input/Output	SCLKA External I/O Address IA7	Programming port CLKA
	18	PB2	Input/Output	/SWR External I/O Address IA0	
	19	PB3	Input/Output	/SRD External I/O Address IA1	
	20	PB4	Input/Output	SA0 External I/O Address IA2	
	21	PB5	Input/Output	SA1 External I/O Address IA3	
	22	PB6	Input/Output	/SCS External I/O Address IA4	
23	PB7	Input/Output	/SLAVATN External I/O Address IA5		

**Table 2. RCM4400W Pinout Configurations (continued)**

Pin	Pin Name	Default Use	Alternate Use	Notes
Header J1	24	PC0	Input/Output TXD I/O Strobe I0 Timer C0 TCLKF	Serial Port D
	25	PC1	Input/Output RXD/TXD I/O Strobe I1 Timer C1 RCLKF Input Capture	
	26	PC2	Input/Output TXC/TXF I/O Strobe I2 Timer C2	Serial Port C
	27	PC3	Input/Output RXC/TXC/RXF I/O Strobe I3 Timer C3 SCLKD Input Capture	
	28	PC4	Input/Output TXB I/O Strobe I4 PWM0 TCLKE	Serial Port B (shared with serial flash)
	29	PC5	Input/Output RXB/TXB I/O Strobe I5 PWM1 RCLKE Input Capture	
	30	PC6	Input/Output TXA/TXE I/O Strobe I6 PWM2	Programming port
	31	PC7	Input/Output RXA/TXA/RXE I/O Strobe I7 PWM3 SCLKC Input Capture	
	32	PE0	Input/Output I/O Strobe I0 A20 Timer C0 TCLKF INT0 QRD1B	

**Table 2. RCM4400W Pinout Configurations (continued)**

Pin		Pin Name	Default Use	Alternate Use	Notes
Header J1	33	PE1	Input/Output	I/O Strobe I1 A21 Timer C1 RXD/RCLKF INT1 QRD1A Input Capture	
	34	PE2	Input/Output	I/O Strobe I2 A22 Timer C2 TXF DREQ0 QRD2B	
	35	PE3	Input/Output	I/O Strobe I3 A23 Timer C3 RXC/RXF/SCLKD DREQ1 QRD2A Input Capture	
	36	PE4	Input/Output	I/O Strobe I4 /A0 INT0 PWM0 TCLKE	
	37	FPGA Interrupt Output/PE5/ SMODE0	Input/Output	I/O Strobe I5 INT1 PWM1 RXB/RCLKE Input Capture	Not connected
	38	FPGA Chip Select/PE6/ SMODE1	Input/Output	I/O Strobe I6 PWM2 TXE DREQ0	Not connected
	39	PE7/STATUS	Input/Output	I/O Strobe I7 PWM3 RXA/RXE/SCLKC DREQ1 Input Capture	PE7 is the default configuration

**Table 2. RCM4400W Pinout Configurations (continued)**

Pin	Pin Name	Default Use	Alternate Use	Notes
Header J1	40	PD0	Input/Output	I/O Strobe I0 Timer C0 D8 INT0 SCLKD/TCLKF QRD1B
	41	PD1	Input/Output	IA6 I/O Strobe I1 Timer C1 D9 INT1 RXD/RCLKF QRD1A Input Capture
	42	PD2	Input/Output	I/O Strobe I2 Timer C2 D10 DREQ0 TXF/SCLKC QRD2B
	43	PD3	Input/Output	IA7 I/O Strobe I3 Timer C3 D11 DREQ1 RXC/RXF QRD2A Input Capture
	44	PD4	Input/Output	I/O Strobe I4 D12 PWM0 TXB/TCLKE
	45	PD5	Input/Output	IA6 I/O Strobe I5 D13 PWM1 RXB/RCLKE Input Capture

**Table 2. RCM4400W Pinout Configurations (continued)**

Pin		Pin Name	Default Use	Alternate Use	Notes
Header J1	46	PD6	Input/Output	I/O Strobe I6 D14 PWM2 TXA/TXE	Serial Port E
	47	PD7	Input/Output	IA7 I/O Strobe I7 D15 PWM3 RXA/RXE Input Capture	
	48	Not Connected			
	49	Not Connected			
	50	GND			



### 4.1.1 Memory I/O Interface

The Rabbit 4000 address lines (A0–A19) and all the data lines (D0–D7) are routed internally to the onboard flash memory and SRAM chips. I/O write (/IOWR) and I/O read (/IORD) are available for interfacing to external devices, and are also used by the RCM4400W.

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–PB7 can also be used as an auxiliary address bus.

When using the auxiliary I/O bus for any reason, you must add the following line at the beginning of your program.

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

Selected pins on Parallel Ports D and E as specified in Table 2 may be used for input capture, quadrature decoder, DMA, and pulse-width modulator purposes.

### 4.1.2 Other Inputs and Outputs

The STATUS pin can be brought out to header J1 instead of the PE7 pin as explained in Appendix A.6.

/RESET\_IN is normally associated with the programming port, but may be used as an external input to reset the Rabbit 4000 microprocessor and the RCM4400W memory.

/RESET\_OUT is an output from the reset circuitry that can be used to reset other peripheral devices.

## 4.2 Serial Communication

The RCM4400W module does not have any serial driver or receiver chips directly on the board. However, a serial interface may be incorporated on the board the RCM4400W is mounted on. For example, the Prototyping Board has an RS-232 transceiver chip.

### 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 application development has been completed and the RCM4400W is operating in the Run Mode.

Serial Port B is shared with the serial flash, and is set up as a clocked serial port. PB0 provides the SCLKB output to the serial flash. Note that the serial flash is used to support the FPGA chip in the Wi-Fi circuit, and is not available for customer 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. Note that PD2 and PD0 provide the SCLKC and SCLKD outputs automatically when Serial Ports C and D are set up as clocked serial ports.

Serial Ports E and F can also be configured as SDLC/HDLC serial ports. The IrDA protocol is also supported in SDLC format by these two ports. Serial Ports E and F must be configured before they can be used. The sample program `IOCONFIG_SWITCHCHO.C` in the Dynamic C `SAMPLES\RCM4400W\SERIAL` folder shows how to configure Serial Ports E and F.

Table 3 summarizes the possible parallel port pins for the serial ports and their clocks.

**Table 3. Rabbit 4000 Serial Port and Clock Pins**

Serial Port A	TXA	PC6, PC7, PD6	Serial Port E	TXE	PD6, PE6, PC6
	RXA	PC7, PD7, PE7		RXE	PD7, PE7, PC7
	SCLKA	PB1		RCLKE	PD5, PE5, PC5
Serial Port B	TXB	PC4, PC5, PD4	Serial Port F	TCLKE	PD4, PE4, PC4
	RXB	PC5, PD5, PE5		TXF	PD2, PE2, PC2
	SCLKB	PB0		RXF	PD3, PE3, PC3
Serial Port C	TXC	PC2, PC3		RCLKF	PD1, PE1, PC1
	RXC	PC3, PD3, PE3		TCLKF	PD0, PE0, PC0
	SCLKC	PD2, PE2, PE7, PC7	RCLKE and RCLKF must be selected to be on the same parallel port as TXE and TXF respectively.		
Serial Port D	TXD	PC0, PC1			
	RXD	PC1, PD1, PE1			
	SCLKD	PD0, PE0, PE3, PC3			

#### 4.2.1.1 Using the Serial Ports

The receive lines on the RCM4400W serial ports do not have pull-up resistors. If you are using the serial ports without a receiver chip (for example, for RS-422, RS-232, or RS-485 serial communication), the absence of a pull-up resistor on the receive line will likely lead to line breaks being generated since line breaks are normally generated whenever the receive line is pulled low. If you are operating a serial port asynchronously, you can inhibit character assembly during breaks by setting bit 1 in the corresponding Serial Port Extended Register to 1. Should you need line breaks, you will have to either add a pull-up resistor on your motherboard or use a receiver that incorporates the circuits to have the output default to the nonbreak levels.

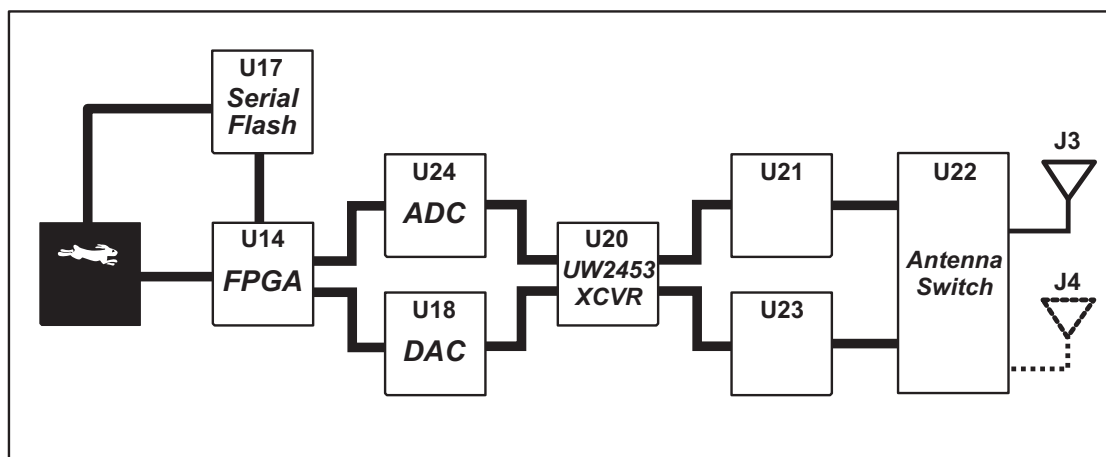
The Dynamic C **RS232.LIB** library requires you to define the macro **RS232\_NOCHARASSYINBRK** to inhibit break-character assembly for all the serial ports.

```
#define RS232_NOCHARASSYINBRK
```

This macro is already defined so that it is the default behavior for the sample programs in the Dynamic C **SAMPLES\RCM4400W\SERIAL** folder.

### 4.2.2 Wi-Fi

Figure 9 shows a functional block diagram for the Wi-Fi circuits.



**Figure 9. RCM4400W Wi-Fi Block Diagram**

The Wi-Fi transmission is controlled by the onboard FPGA chip at U14. The primary functions of this FPGA are to implement the 802.11b baseband Media Access Control (MAC) functionality, and to control the 802.11b integrated UBEC UW2453 transceiver.

The serial flash programs the FPGA automatically whenever power is applied. Once configured, the FPGA performs all of the 802.11b data encoding and decoding, radio configuration, and radio control functions.

The data interface between the FPGA and the UBEC UW2453 based 802.11b radio section consists of a D/A converter and an A/D converter. Both devices convert “I” and “Q” data samples at a rate of 40 MHz.

The UBEC UW2453 is a single-chip transceiver with integrated power amplifier for the 2.4 GHz Industrial, Scientific, and Medical (ISM) band. It is configured and controlled by the FPGA via a 3-wire serial data bus. The UW2453 contains the entire receiver, transmitter, VCO, PLL, and power amplifier necessary to implement an 802.11b radio.

The UW2453 can transmit and receive data at up to 11Mbits/s in the 802.11b mode. It supports 802.11b channels 1–13 (2.401 GHz to 2.472 GHz). The data modulate the channel carrier in such a way so as to produce a spread spectrum signal within the 22 MHz channel bandwidth of the selected channel. The channel numbers and associated frequencies are listed below in Table 4.

The Wi-Fi channels have a certain amount of overlap with each other. The further apart two channel numbers are, the less the likelihood of interference. If you encounter interference with a neighboring WLAN, change to a different channel. For example, use channels 1, 6, and 11 to minimize any overlap.

**Table 4. Wi-Fi Channel Allocations**

Channel	Center Frequency (GHz)	Frequency Spread (GHz)
1	2.412	2.401–2.423
2	2.417	2.406–2.428
3	2.422	2.411–2.433
4	2.427	2.416–2.438
5	2.432	2.421–2.443
6	2.437	2.426–2.448
7	2.442	2.431–2.453
8	2.447	2.436–2.458
9	2.452	2.441–2.463
10	2.457	2.446–2.468
11	2.462	2.451–2.473
12*	2.467	2.456–2.478
13*	2.472	2.461–2.483
14 (not used)	2.484	2.473–2.495

\* These channels are disabled for units delivered for sale in the United States and Canada.

Many countries specify the channel range and power limits for Wi-Fi devices operated within their borders, and these limits are set automatically in the RCM4400W in firmware according to the country or region. For example, only channels 1–11 are authorized for use in the United States or Canada, and so channels 12 and 13 are disabled. See Section 6.2.4.1 for additional information and sample programs demonstrating how to configure an end device to meet the regulatory channel range and power limit requirements. Table 5 provides additional information on which channels are allowed in selected countries. *Any attempt to operate a device outside the allowed channel range or power limits will void your regulatory approval to operate the device in that country.*

U21 and U23 are bandpass filters to reduce the transmit and receive sideband noise levels.

The same antenna is used to transmit and receive the 802.11b RF signal. An antenna switch isolates the high-power RF Tx signal path from the RF Rx signal path. The antenna switch works by alternately connecting the antennas to either the UW2453 Tx output or to the UW2453 Rx input. In order to support this antenna-sharing scheme, the RCM4400W module operates the radio in a half-duplex mode so that receive and transmit operations never occur at the same time. The chip at U22 switches the receive/transmit functionality between the outputs at J3 and J4 (not stuffed) so that J3 is transmitting while J4 would be receiving and vice versa. Dynamic C does not support a J4 output.

There are two LEDs close to the RP-SMA antenna connector at J3, a green LED at DS1 (**LINK**) to indicate association with the Wi-Fi access point, and a yellow LED at DS2 (**ACT**) to indicate activity.

An RG316 coaxial cable may be used to extend the antenna up to 30 cm (1 ft). This coaxial cable has an impedance of 50  $\Omega$  and experiences 1.65 dB/m signal loss at 2.4 GHz.

### 4.2.3 Programming Port

The RCM4400W is programmed via the 10-pin header labeled J2. The programming port uses the Rabbit 4000's Serial Port A for communication. Dynamic C uses the programming port to download and debug programs.

Serial Port A is also used for the following operations.

- Cold-boot the Rabbit 4000 on the RCM4400W after a reset.
- Fast copy designated portions of flash memory from one Rabbit-based board (the master) to another (the slave) using the Rabbit Cloning Board.

#### Alternate Uses of the Programming Port

All three Serial Port A signals are available as

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

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

In addition to Serial Port A, the Rabbit 4000 startup-mode (SMODE0, SMODE1), STATUS, and reset pins are available on the programming port.

The two startup-mode pins determine what happens after a reset—the Rabbit 4000 is either cold-booted or the program begins executing at address 0x0000.

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

1. It can be driven low on the first op code fetch cycle.
2. It can be driven low during an interrupt acknowledge cycle.
3. It can also serve as a general-purpose output once a program has been downloaded and is running.

The reset pin is an external input that is used to reset the Rabbit 4000.

Refer to the *Rabbit 4000 Microprocessor User's Manual* for more information.

## 4.3 Programming Cable

The programming cable is used to connect the programming port (header J2) of the RCM4400W to a PC serial COM port. The programming cable converts the RS-232 voltage levels used by the PC serial port to the CMOS voltage levels used by the Rabbit 4000.

When the **PROG** connector on the programming cable is connected to the RCM4400W programming port, programs can be downloaded and debugged over the serial interface.

The **DIAG** connector of the programming cable may be used on header J2 of the RCM4400W with the RCM4400W operating in the Run Mode. This allows the programming port to be used as a regular serial port.

### 4.3.1 Changing Between Program Mode and Run Mode

The RCM4400W is automatically in Program Mode when the **PROG** connector on the programming cable is attached, and is automatically in Run Mode when no programming cable is attached. When the Rabbit 4000 is reset, the operating mode is determined by the status of the SMODE pins. When the programming cable's **PROG** connector is attached, the SMODE pins are pulled high, placing the Rabbit 4000 in the Program Mode. When the programming cable's **PROG** connector is not attached, the SMODE pins are pulled low, causing the Rabbit 4000 to operate in the Run Mode.

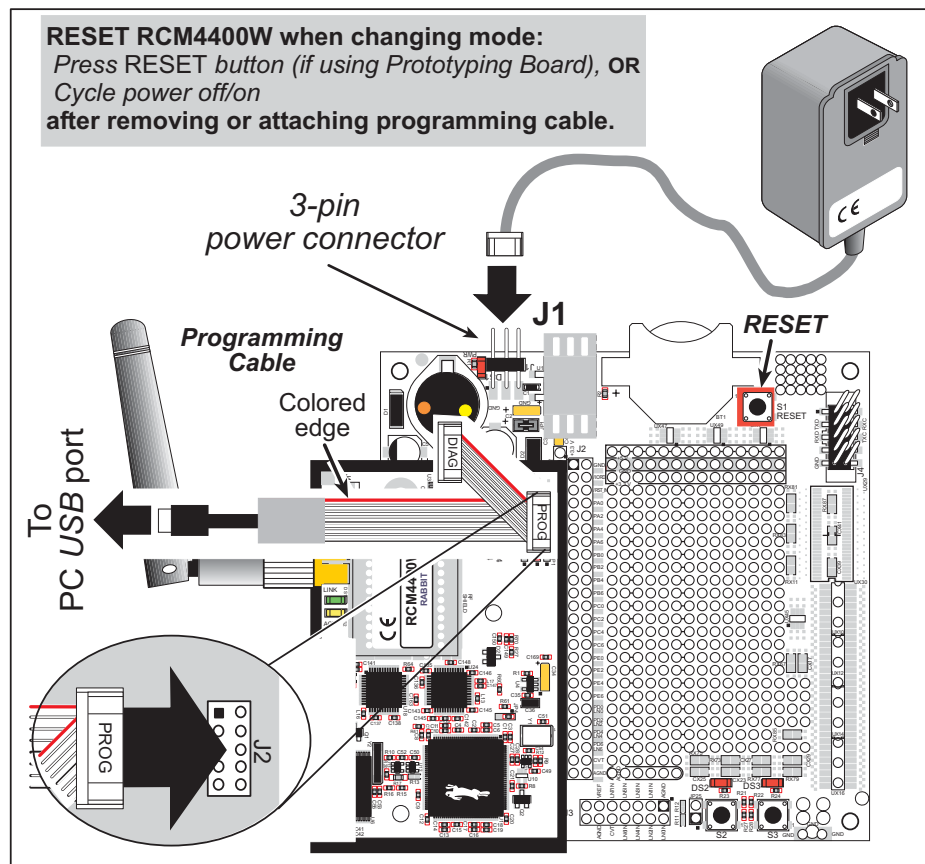


Figure 10. Switching Between Program Mode and Run Mode

A program “runs” in either mode, but can only be downloaded and debugged when the RCM4400W is in the Program Mode.

Refer to the *Rabbit 4000 Microprocessor User’s Manual* for more information on the programming port.

#### **4.3.2 Standalone Operation of the RCM4400W**

Once the RCM4400W has been programmed successfully, remove the programming cable from the programming connector and reset the RCM4400W. The RCM4400W may be reset by cycling, the power off/on or by pressing the **RESET** button on the Prototyping Board. The RCM4400W module may now be removed from the Prototyping Board for end-use installation.

**CAUTION:** Power to the Prototyping Board or other boards should be disconnected when removing or installing your RCM4400W module to protect against inadvertent shorts across the pins or damage to the RCM4400W if the pins are not plugged in correctly. Do not reapply power until you have verified that the RCM4400W module is plugged in correctly.



## 4.4 Other Hardware

### 4.4.1 Clock Doubler

The RCM4400W takes advantage of the Rabbit 4000 microprocessor's internal clock doubler. A built-in clock doubler allows half-frequency crystals to be used to reduce radiated emissions. The 58.98 MHz frequency specified for the RCM4400W is generated using a 29.49 MHz crystal.

The clock doubler should not be disabled since Wi-Fi operations depend highly on CPU resources.

### 4.4.2 Spectrum Spreader

The Rabbit 4000 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. The spectrum spreader settings may be changed through a simple configuration macro as shown below.

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

```
ENABLE_SPREADER=1
```

For strong spreading, add the line

```
ENABLE_SPREADER=2
```

To disable the spectrum spreader, add the line

```
ENABLE_SPREADER=0
```

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

3. Click **OK** to save the macro. The spectrum spreader will now remain off whenever you are in the project file where you defined the macro.

**NOTE:** Refer to the *Rabbit 4000 Microprocessor User's Manual* for more information on the spectrum-spreading setting and the maximum clock speed.

## 4.5 Memory

### 4.5.1 SRAM

All RCM4400W modules have 512KB of battery-backed data SRAM installed at U6, and 512KB of fast SRAM are installed at U7.

### 4.5.2 Flash EPROM

All RCM4400W modules also have 512KB of flash EPROM installed at U5.

**NOTE:** Rabbit 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 discouraged. Instead, define a “user block” area to store persistent data. The functions **writeUserBlock** and **readUserBlock** are provided for this. Refer to the *Rabbit 4000 Microprocessor Designer’s Handbook* for additional information.

### 4.5.3 Serial Flash

The 1MB serial flash memory on the RCM4400W is used to bootstrap the FPGA for the Wi-Fi circuits, and was not available for customer use. Starting with Dynamic C v. 10.54, it is possible to access 800KB of the serial flash for customer use as long as your application does not try to access the serial flash during the first call to **sock\_init()**.

## 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 single-board computers and other devices based on the Rabbit microprocessor. Chapter 5 describes the libraries and function calls related to the RCM4400W.

### 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 RCM4400W. 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 on the RCM4400W module, but should be run from the fast SRAM after the serial programming cable is disconnected. Your final code must always be stored in flash memory for reliable operation. RCM4400W modules 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 RCM4400W modules running at 58.98 MHz.

**NOTE:** Do not depend on the flash memory sector size or type in your program logic. The RCM4400W 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 NT and later—see Rabbit's Technical Note TN257, *Running Dynamic C® With Windows Vista®*, for additional information if you are using a Dynamic C under Windows Vista. Programs can be downloaded at baud rates of up to 460,800 bps after the program compiles.

Dynamic C has a number of standard features.

- Full-feature source and/or assembly-level debugger, no in-circuit emulator required.
- Royalty-free TCP/IP stack with source code and most common protocols.
- Hundreds of functions in source-code libraries and sample programs:
  - ▶ Exceptionally fast support for floating-point arithmetic and transcendental functions.
  - ▶ RS-232 and RS-485 serial communication.
  - ▶ Analog and digital I/O drivers.
  - ▶ I<sup>2</sup>C, SPI, GPS, file system.
  - ▶ LCD display and keypad drivers.
- Powerful language extensions for cooperative or preemptive multitasking
- Loader utility program to load binary images into Rabbit targets in the absence of Dynamic C.
- Provision for customers to create their own source code libraries and augment on-line help by creating “function description” block comments using a special format for library functions.
- Standard debugging features:
  - ▶ Breakpoints—Set breakpoints that can disable interrupts.
  - ▶ Single-stepping—Step into or over functions at a source or machine code level,  $\mu$ C/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.

## 5.2 Dynamic C Function Calls

### 5.2.1 Digital I/O

The RCM4400W was designed to interface with other systems, and so there are no drivers written specifically for the Rabbit 4000 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 4000 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/RCM4400W** 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 Rabbit's Technical Note TN213, *Rabbit Serial Port Software*, both included with the online documentation.

### 5.2.3 User Block

Certain function calls involve reading and storing calibration constants from/to the simulated EEPROM in flash memory located at the top 2K of the reserved user block memory area (3800–39FF). This leaves the address range 0–37FF in the user block available for your application.

These address ranges may change in the future in response to the volatility in the flash memory market, in particular sector size. The sample program **USERBLOCK\_INFO.C** in the Dynamic C **SAMPLES\USERBLOCK** folder can be used to determine the version of the ID block, the size of the ID and user blocks, whether or not the ID/user blocks are mirrored, the total amount of flash memory used by the ID and user blocks, and the area of the user block available for your application.

The `USERBLOCK_CLEAR.C` sample program shows you how to clear and write the contents of the user block that you are using in your application (the calibration constants in the reserved area and the ID block are protected).

## 5.2.4 SRAM Use

The RCM4400W module has a battery-backed data SRAM and a program-execution SRAM. Dynamic C provides the `protected` keyword to identify variables that are to be placed into the battery-backed SRAM. The compiler generates code that maintains two copies of each protected variable in the battery-backed SRAM. The compiler also generates a flag to indicate which copy of the protected variable is valid at the current time. This flag is also stored in the battery-backed SRAM. When a protected variable is updated, the “inactive” copy is modified, and is made “active” only when the update is 100% complete. This assures the integrity of the data in case a reset or a power failure occurs during the update process. At power-on the application program uses the active copy of the variable pointed to by its associated flag.

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

```
main() {
    protected int state1, state2, state3;
    ...

    _sysIsSoftReset();    // restore any protected variables
```

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

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

## 5.2.5 Wi-Fi Drivers

The Wi-Fi drivers are located in the `Rabbit4000\LIB\TCPIP` folder. Complete information on the Wi-Fi libraries and function calls is provided in Chapter 6. Additional information on TCP/IP is provided in the *Dynamic C TCP/IP User's Manual*.

## 5.2.6 Serial Flash Drivers

The 1MB serial flash memory on the RCM4400W is used to bootstrap the FPGA for the Wi-Fi circuits, and was not available for customer use. Starting with Dynamic C v. 10.54, it is possible to access 800KB of the serial flash for customer use.

The Dynamic C `Rabbit4000\LIB\SerialFlash\SFLASH.LIB` and `Rabbit4000\LIB\SerialFlash\SFLASH_FAT.LIB` libraries provide the function calls needed to use the serial flash. The FAT file system function calls are in the Dynamic C `Rabbit4000\LIB\FileSystem\FAT_CONFIG.LIB` library.

Since the RCM4400W uses part of the serial flash to bootstrap its FPGA, you must ensure that your application does not try to access the serial flash during the first call to `sock_init()`. If your application has written to the flash before calling `sock_init()`, be sure to spin on `sf_isWriting()` as shown in the sample code to ensure that the write is complete.

```
while (sf_isWriting()) {  
    // waiting for write to complete before calling sock_init  
}  
sock_init();
```

## 5.2.7 Prototyping Board Function Calls

The function calls described in this section are for use with the Prototyping Board features. The source code is in the Dynamic C `Rabbit4000\LIB\RCM4xxx\RCM44xxW.LIB` library if you need to modify it for your own board design.

The sample programs in the Dynamic C `SAMPLES\RCM4400W` folder illustrate the use of the function calls.

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

### 5.2.7.1 Board Initialization

---

---

#### `brdInit`

---

---

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

#### DESCRIPTION

Call this function at the beginning of your program. This function initializes Parallel Ports A through E for use with the Prototyping Board. This function call is intended for demonstration purposes only, and can be modified for your applications.

#### Summary of Initialization

1. I/O port pins are configured for Prototyping Board operation.
2. Unused configurable I/O are set as tied outputs.
3. RS-232 is not enabled.
4. LEDs are off.
5. The slave port is disabled.

#### RETURN VALUE

None.



### 5.2.7.2 Alerts

These function calls can be found in the Dynamic C `Rabbit4000\LIB\RCM4xxx\RCM4xxx.LIB` library.

---

---

#### **timedAlert**

---

---

```
void timedAlert(unsigned long timeout);
```

##### **DESCRIPTION**

Polls the real-time clock until a timeout occurs. The RCM4400W will be in a low-power mode during this time. Once the timeout occurs, this function call will enable the normal power source.

##### **PARAMETER**

<b>timeout</b>	the duration of the timeout in seconds
----------------	--

##### **RETURN VALUE**

None.

---

---

#### **digInAlert**

---

---

```
void digInAlert(int dataport, int portbit, int value,  
                unsigned long timeout);
```

##### **DESCRIPTION**

Polls a digital input for a set value or until a timeout occurs. The RCM4400W will be in a low-power mode during this time. Once a timeout occurs or the correct byte is received, this function call will enable the normal power source and exit.

##### **PARAMETERS**

<b>dataport</b>	the input port data register to poll (e.g., PADR)
<b>portbit</b>	the input port bit (0–7) to poll
<b>value</b>	the value of 0 or 1 to receive
<b>timeout</b>	the duration of the timeout in seconds (enter 0 for no timeout)

##### **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 site [www.rabbit.com/support/](http://www.rabbit.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. Rabbit offers for purchase add-on Dynamic C modules including the popular  $\mu$ C/OS-II real-time operating system, as well as PPP, Advanced Encryption Standard (AES), 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.rabbit.com](http://www.rabbit.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 WI-FI FEATURES

### 6.1 Introduction to Wi-Fi

Wi-Fi, a popular name for 802.11b, refers to the underlying technology for wireless local area networks (WLAN) based on the IEEE 802.11 suite of specifications conforming to standards defined by IEEE. IEEE 802.11b describes the media access and link layer control for a 2.4 GHz implementation, which can communicate at a top bit-rate of 11 Mbits/s. Other standards describe a faster implementation (54 Mbits/s) in the 2.4 GHz (802.11g) and a 54 Mbits/s implementation in the 5.6 GHz band (802.11a). The adoption of 802.11 has been fast because it's easy to use and the performance is comparable to wire-based LANs. Things look pretty much like a wireless LAN.

Wi-Fi (802.11b) is the most common and cost-effective implementation currently available. This is the implementation that is used with the RCM4400W RabbitCore module. A variety of Wi-Fi hardware exists, from wireless access points (WAPs), various Wi-Fi access devices with PCI, PCMCIA, CompactFlash, USB and SD/MMC interfaces, and Wi-Fi devices such as Web-based cameras and print servers.

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

#### 6.1.1 Infrastructure Mode

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

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

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

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

### **6.1.2 Ad-Hoc Mode**

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

### **6.1.3 Additional Information**

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

## 6.2 Running Wi-Fi Sample Programs

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

1. Your module must be 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 module to your PC.
4. Power must be applied to the module through the Prototyping Board.

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

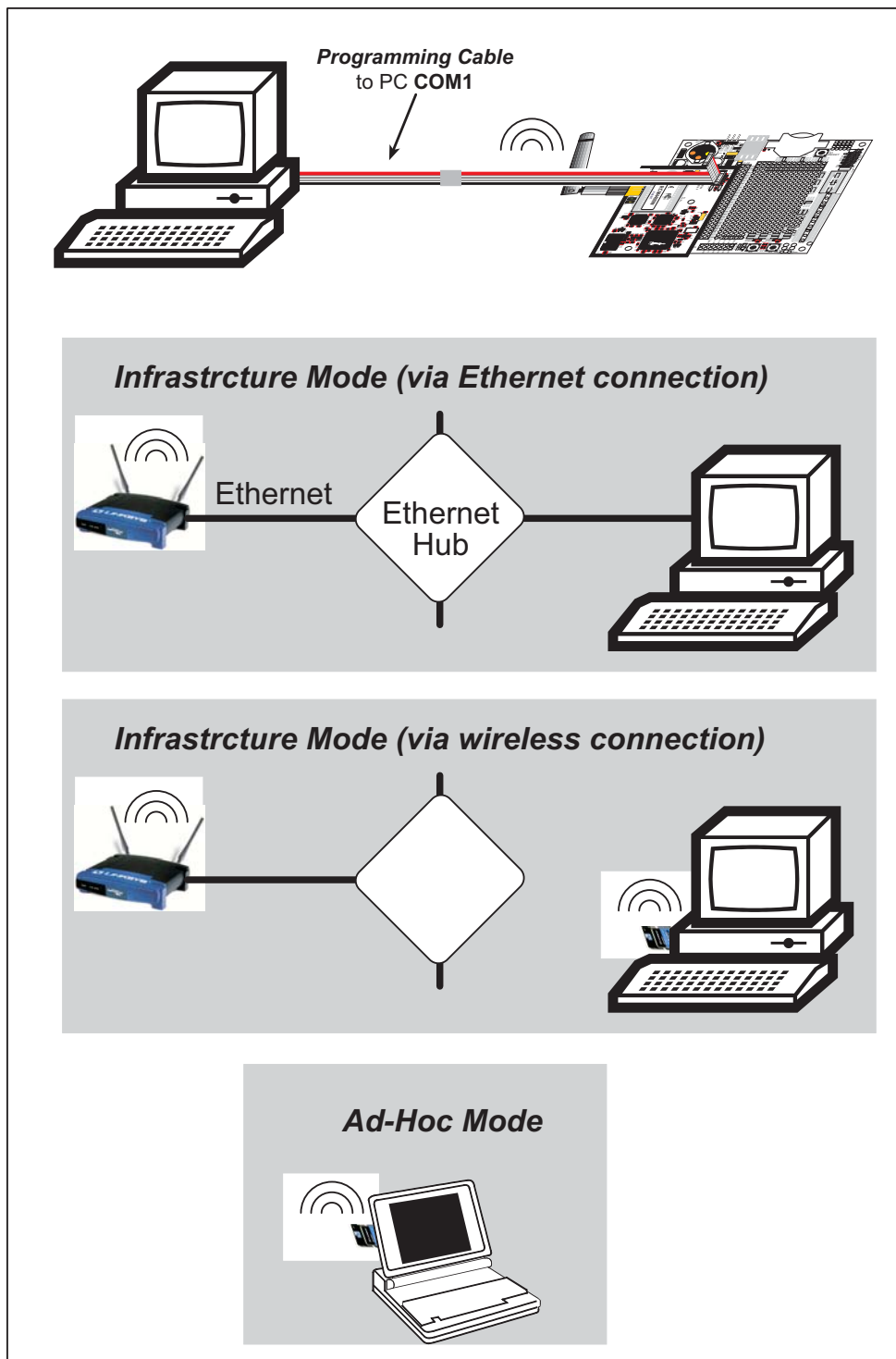
To run a sample program, open it with the **File** menu, then compile and run it by pressing **F9**.

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

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

## 6.2.1 Wi-Fi Setup

Figure 11 shows how your development setup might look once you're ready to proceed.



**Figure 11. Wi-Fi Host Setup**

### **6.2.2 What Else You Will Need**

Besides what is supplied with the RCM4400W Development Kit, you will need a PC with an available USB port to program the RCM4400W module. You will need either an access point for an existing Wi-Fi network that you are allowed to access and have a PC or notebook connected to that network (infrastructure mode), or you will need at least a PDA or PC with Wi-Fi to use the ad-hoc mode.

## 6.2.3 Configuration Information

### 6.2.3.1 Network/Wi-Fi Configuration

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

The sample programs configure the RCM4400W modules with a default **TCPCONFIG** macro from the **Rabbit4000\LIB\TCPIP\TCP\_CONFIG.LIB** library. This macro allows specific IP address, netmask, gateway, and Wi-Fi parameters to be set at compile time. Change the network settings to configure your RCM4400W module with your own Ethernet settings only if that is necessary to run the sample programs; you will likely need to change some of the Wi-Fi settings.

- Network Parameters

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

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

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

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

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

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



### 6.2.3.2 PC/Laptop/PDA Configuration

This section shows how to configure your PC or notebook to run the sample programs. Here we're mainly interested in the PC or notebook that will be communicating wirelessly, which is not necessarily the PC that is being used to compile and run the sample program on the RCM4400W module.

This section provides configuration information for the three possible Wi-Fi setups shown in Figure 11. Start by going to the control panel (**Start > Settings > Control Panel**) and click on **Network Connections**. The screen shots shown here are from Windows 2000, and the interface is similar for other versions of Windows.

Check with your administrator if you are unable to change the settings as described here since you may need administrator privileges.

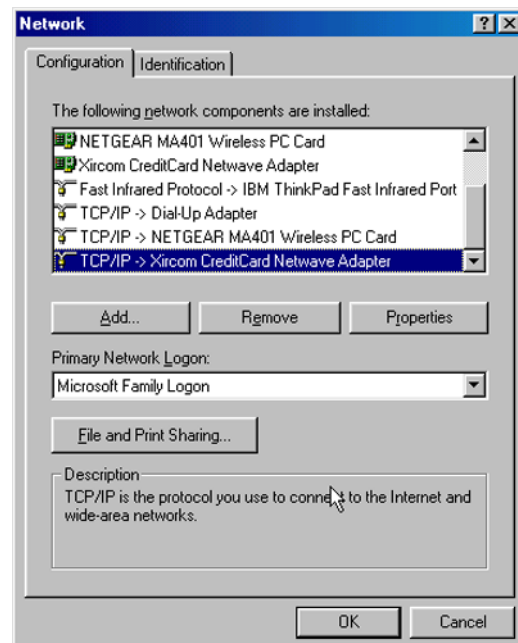


When you are using an access point with your setup in the infrastructure mode, you will also have to set the IP address and netmask (e.g., 10.10.6.99 and 255.255.255.0) for the access point. Check the documentation for the access point for information on how to do this.

#### Infrastructure Mode (via Ethernet connection)

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

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



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

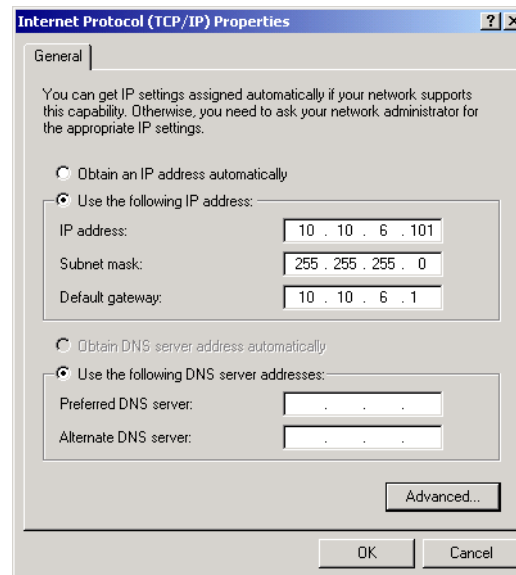
IP Address : 10.10.6.101

Netmask : 255.255.255.0

Default gateway : 10.10.6.1

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

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



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

### Infrastructure Mode (via wireless connection)

Set the IP address and netmask for your wireless-enabled PC or notebook as described in Step 2 for **Infrastructure Mode (via Ethernet connection)** by clicking on **Network Connections**, then on **Local Area Connection**. Now click on **Wireless Network Connection** to select the wireless network you will be connecting to. Once a sample program is running, you will be able to select the network from a list of available networks. You will have set your wireless network name with the `_WIFI_SSID` macro for the infrastructure mode as explained in Section 6.2.3.1, “Network/Wi-Fi Configuration.”

### Ad-Hoc Mode

Set the IP address and netmask for your wireless-enabled PC or notebook as described in Step 2 for **Infrastructure Mode (via Ethernet connection)** by clicking on **Network Connections**, then on **Local Area Connection**. Now click on **Wireless Network Connection** to select the wireless network you will be connecting to. Once a sample program is running, you will be able to select the network from a list of available networks. You will have set your wireless network name with the `_WIFI_OWNCHANNEL` macros for the ad-hoc mode as explained in Section 6.2.3.1, “Network/Wi-Fi Configuration.”

Once the PC or notebook is set up, we're ready to communicate. You can use Telnet or a Web browser such as Internet Explorer, which come with most Windows installations, to use the network interface, and you can use HyperTerminal to view the serial port when these are called for in some of the later sample programs.

Now we're ready to run the sample programs in the Dynamic C `Samples\TCPIP\WiFi` folder. The sample programs should run as is in most cases.

## 6.2.4 Wi-Fi Sample Programs

The sample programs in Section 6.2.4.1 show how to set up the country- or region-specific attributes, but do not show the basic setup of a wireless network. The sample programs in Section 6.2.4.2 show the setup and operation of a wireless network — the `WIFISCAN.C` sample program is ideal to demonstrate that the RCM4400W has been hooked up correctly and that the Wi-Fi setup is correct so that an access point can be found.

### 6.2.4.1 Wi-Fi Operating Region Configuration

The country or region you select will automatically set the power and channel requirements to operate the RCM4400W module. The following three options are available.

1. *Country or region is set at compile time.* This option is ideal when the end device is intended to be sold and used only in a single region. If the end device is to be deployed across multiple regions, this method would require an application image to be created for each region. This option is the only approved option for RCM4400W modules in Japan.
2. *Country or region is set via the 802.11d feature of the access point.* This option uses beacons from an access point to configure the RCM4400W country or region automatically. The end user is responsible for enabling 802.11d on the access point and then selecting the correct country to be broadcast in the beacon packets.

**NOTE:** This option sets the power limit for RCM4400W to the maximum level permitted in the region or the capability of the RCM4400W, whichever is less. Since the beacons are being sent continuously, the `wifi_ioctl WIFI_TX_POWER` function cannot be used with this option.

3. *Country or region is set at run time.* This is a convenient option when the end devices will be deployed in multiple regions. A serial user interface would allow the RCM4400W module to be configured via a Web page. Systems integrators would still have to make sure the end devices operate within the regulatory requirements of the country or region where the units are being deployed.

These options may be used alone or in any combination. The three sample programs in the Dynamic C `Samples\TCPIP\WiFi\Regulatory` folder illustrate the use of these three options.

- **REGION\_COMPILETIME.C**—demonstrates how you can set up your RCM4400W-based system at compile time to operate in a given country or region to meet power and channel requirements.

The country or region you select will automatically set the power and channel requirements to operate the RCM4400W module. Rabbit recommends that you check the reg-

ulations for the country where your system incorporating the RCM4400W will be deployed for any other requirements. *Any attempt to operate a device outside the allowed channel range or power limits will void your regulatory approval to operate the device in that country.*

Before you compile and run this sample program, uncomment the `#define _WIFI_REGION_REQ` line corresponding to the region where your system will be deployed. The Americas region will be used by default if one of these lines is not uncommented. Now compile and run this sample program. The Dynamic C **STDIO** window will display the region you selected.

The sample program also allows you to set up the TCP/IP configuration, and set the IP address and SSID as shown in the sample code below.

```
#define TCPCONFIG 1
#define _PRIMARY_STATIC_IP "10.10.6.170"
#define _WIFI_SSID "olmtest"
```

- **REGION\_MULTI\_DOMAIN.C**—demonstrates how the multi-domain options from the access point can be used to configure your RCM4400W-based system to meet regional regulations. The sample program includes pings to indicate that the RCM4400W-based system has successfully received country information from your access point.

The country or region you select will automatically set the power and channel requirements to operate the RCM4400W module. Rabbit recommends that you check the regulations for the country where your system incorporating the RCM4400W will be deployed for any other requirements.

Before you compile and run this sample program, verify that the access point has the 802.11d option enabled and is set for the correct region or country. Check the TCP/IP configuration parameters, the IP address, and the SSID in the macros, which are reproduced below.

```
#define TCPCONFIG 1
#define WIFI_REGION_VERBOSE
#define _PRIMARY_STATIC_IP "10.10.6.170"
#define _WIFI_SSID "deanap"
```

Now compile and run this sample program. The `#define WIFI_REGION_VERBOSE` macro will display the channel and power limit settings. The Dynamic C **STDIO** window will then display a menu that allows you to complete the configuration of the user interface.

- **REGION\_RUNTIME\_PING.C**—demonstrates how the region or country can be set at run time to configure your RCM4400W-based system to meet regional regulations. The sample program also shows how to save and retrieve the region setting from nonvolatile memory. Once the region/country is set, this sample program sends pings using the limits you set.

The country or region you select will automatically set the power and channel requirements to operate the RCM4400W module. Rabbit recommends that you check the regulations for the country where your system incorporating the RCM4400W will be deployed for any other requirements.

Before you compile and run this sample program, check the TCP/IP configuration parameters, the IP address, and the SSID in the macros, which are reproduced below.

```
#define TCPCONFIG 1
#define WIFI_REGION_VERBOSE
#define PING_WHO "10.10.6.1"
#define _PRIMARY_STATIC_IP "10.10.6.170"
#define _WIFI_SSID "deanap"
```

Now compile and run this sample program. The `#define WIFI_REGION_VERBOSE` macro will display the channel and power limit settings. The Dynamic C **STDIO** window will then display a menu that allows you to complete the configuration of the user interface.

#### 6.2.4.2 Wi-Fi Operation

- **WIFIPINGYOU.C**—sends out a series of pings to a RabbitCore module on an ad-hoc Wi-Fi network.

This sample program uses some predefined macros. The first macro specifies the default TCP/IP configuration from the Dynamic C `Lib\TCPIP\TCP_CONFIG.LIB` library.

```
#define TCPCONFIG 1
```

Use the next macro unchanged as long as you have only one RCM4400W RabbitCore module. Otherwise use this macro unchanged for the first RabbitCore module.

```
#define NODE 1
```

Then change the macro to `#define NODE 2` before you compile and run this sample program on the second RCM4400W RabbitCore module.

The next macros assign an SSID name and a channel number to the Wi-Fi network.

```
#define _WIFI_SSID "rab-hoc"
#define _WIFI_OWNCHANNEL "5"
```

Finally, IP addresses are assigned to the RabbitCore modules.

```
#define IPADDR_1 "10.10.8.1"
#define IPADDR_2 "10.10.8.2"
```

As long as you have only one RCM4400W RabbitCore module, the Dynamic C **STDIO** window will display the pings sent out by the module. You may set up a Wi-Fi enabled laptop with the IP address in `IPADDR_2` to get the pings.

If you have two RCM4400W RabbitCore modules, they will ping each other, and the Dynamic C **STDIO** window will display the pings.

- **WIFISCAN.C**—initializes the RCM4400W and scans for other Wi-Fi devices that are operating in either the ad-hoc mode or through access points in the infrastructure mode. No network parameter settings are needed since the RCM4400W does not actually join an 802.11b network. This program outputs the results of the scan to the Dynamic C **STDIO** window.

- **WIFISCANASSOCIATE.C**— demonstrates how to scan Wi-Fi channels for SSIDs using the `wifi_ioctl()` function call with **WIFI\_SCAN**. This takes a while to complete, so `wifi_ioctl()` calls a callback function when it is done. The callback function is specified using an `wifi_ioctl()` **WIFI\_SCANCB** function call.

Before you run this sample program, configure the Dynamic C **TCP\_CONFIG.LIB** library and your **TCPCONFIG** macro.

1. Use macro definitions in the “Defines” tab in the Dynamic C **Options > Project Options** menu to modify any parameter settings.

If you are not using DHCP, set the IP parameters to values appropriate to your network.

```
_PRIMARY_STATIC_IP = "10.10.6.100"
_PRIMARY_NETMASK = "255.255.255.0"
_MY_NAMESERVER = "10.10.6.1"
_MY_GATEWAY = "10.10.6.1"
```

Set the macro **IFC\_WIFI\_SSID** to define a C-style string to set the SSID of your access point as, for example,

```
IFC_WIFI_SSID = "My Access Point"
```

or use an empty string, "", to associate with the strongest BSS available.

Alternatively, you may create your own **CUSTOM\_CONFIG.LIB** library modeled on the Dynamic C **TCP\_CONFIG.LIB** library. Then use a **TCPCONFIG** macro greater than or equal to 100, which will invoke your **CUSTOM\_CONFIG.LIB** library to be used. Remember to add the **CUSTOM\_CONFIG.LIB** library to **LIB.DIR**.

2. If you are using DHCP, change the definition of the **TCPCONFIG** macro to 5. The default value of 1 indicates Wi-Fi with a static IP address.

Now compile and run the sample program. Follow the menu options displayed in the Dynamic C **STDIO** window.

```
Press s to scan available access points
Press a to scan access points and associate
Press m to print WIFI MAC status
```

Note that `wifi_ioctl()` function calls with **WIFI\_SCAN** do not return data directly since the scan takes a fair amount of time. Instead, callback functions are used. The callback function is set with an earlier `wifi_ioctl()` function call.

```
wifi_ioctl(IF_WIFI0, WIFI_SCANCB, scan_callback, 0);
wifi_ioctl(IF_WIFI0, WIFI_SCAN, "0", 0);
```

The data passed to the callback function are ephemeral since another scan may occur. Thus, the data need to be used (or copied) during the callback function.

While waiting for user input, it is important to keep the network alive by calling `tcp_tick(NULL)` regularly.

## 6.2.5 RCM4400W Sample Programs

The following sample programs are in the Dynamic C **SAMPLES\RCM4400W\TCP/IP\** 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 DS2 and DS3 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. Remember to configure the access point to match the default settings of the **TCPCONFIG 1** macro.

`http://10.10.6.100.`

Otherwise use the TCP/IP settings you entered in the in the “Defines” tab in the Dynamic C **Options > Project Options** menu.

- **PINGLED.C**—This program demonstrates ICMP by pinging a remote host. It will flash LED DS2 on the Prototyping Board when a ping is sent and it will flash LED DS3 when a ping is received.

Before you compile and run this sample program, change **PING\_WHO** to the host you want to ping. You may modify **PING\_DELAY** define to change the amount of time in milliseconds between the outgoing pings.

Uncomment the **VERBOSE** define to see the incoming ping replies.

- **PINGLED\_STATS.C**—This program is similar to **PINGLED.C**, but it also displays receiver/transmitter statistics in the Dynamic C **STDIO** window.

Before you compile and run this sample program, change **PING\_WHO** to the host you want to ping. You may modify **PING\_DELAY** define to change the amount of time in milliseconds between the outgoing pings.

Modify the value in the **MOVING\_AVERAGE** macro to change the moving average filtering of the statistics. Also review the **GATHER\_INTERVAL** and **GRAPHICAL** macros, which affect the number of samples to gather and create a bar graph display instead of a numeric display.

Uncomment the **VERBOSE** define to see the incoming ping replies.

- **PINGLED\_WPA\_PSK.C**—This program demonstrates the use of WPA PSK (Wi-Fi Protected Access with Pre-Shared Key). WPA is a more secure replacement for WEP. The implementation in the sample program supports use of the TKIP (Temporal Key Integrity Protocol) cypher suite.

The sample program uses macros to configure the access point for WPA PSK, specify the TKIP cypher suite, assign the access point SSID, and set the passphrase.

```
#define WIFI_USE_WPA // Bring in WPA support
#define _WIFI_WEP_FLAG WIFICONF_WEP_TKIP // Define cypher suite

#define _WIFI_SSID "parvati"

#define _WIFI_PSK_PASSPHRASE "now is the time"
```

The next macro specifies a suitable pre-shared key. The key may be entered either as 64 hexadecimal digits or as an ASCII string of up to 63 characters.

```
#define _WIFI_PSK_HEX
```

When you assign your own key, there is a good chance of typos since the key is long. It is advisable to enter the key in this macro first, then copy and paste into your access point to ensure that both the RCM4400W and the access point have the same key.

Initially, it may be easier to use the 64 hexadecimal digits form of the key rather than the ASCII passphrase. A passphrase requires considerable computation effort, which delays the startup of the sample by about 40 seconds.

If you want to add authentication, set the authentication to “open system,” which basically means that knowing the key is sufficient to allow access.

```
#define WIFI_AUTH WIFICONF_AUTH_OPEN_SYS
```

Change **PING\_WHO** to the host you want to ping. You may modify **PING\_DELAY** to change the amount of time in milliseconds between the outgoing pings.

Uncomment the **VERBOSE** define to see the incoming ping replies.

Once you have compiled the sample program and it is running, LED DS2 will flash when a ping is sent, and LED DS3 will flash when a ping is received.

- **POWERDOWN.C**—This program demonstrates how to power down the FPGA chip in the Wi-Fi circuit to reduce power consumption. Note that powering down the Wi-Fi portion of the RCM4400W module results in a loss of the network interface (unlike an Ethernet connection), and so is only suitable for applications such as data logging where only intermittent network connectivity is required.

The sample program demonstrates the powerdown operation as a simple sequential state machine. LED DS2 on the Prototyping Board will be on when the network interface is up, and LED DS3 will be on when the Wi-Fi circuit is powered up.

Before you compile and run this sample program, modify the configuration macros, including the **DOWNTIME** and the **UPTIME** values. The interface will be powered up and down for these intervals.

- **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 DS2 and DS3 on the Prototyping Board will light up when e-mail is being sent.



## 6.3 Dynamic C Wi-Fi Configurations

Rabbit has implemented a packet driver for the RCM4400W that functions much like an Ethernet driver for the Dynamic C implementation of the TCP/IP protocol stack. In addition to functioning like an Ethernet packet driver, this driver implements a function call to access the functions implemented on the 802.11b interface, and to mask channels that are not available in the region where the RCM4400W will be used.

The Wi-Fi interface may be used either at compile time using macro statements or at run time with the `wifi_ioctl()` function call from the Dynamic C `Rabbit4000\LIB\TCPIP\WIFI\WIFI_WLN_API.LIB` library.

### 6.3.1 Configuring Dynamic C at Compile Time

Rabbit has made it easy for you to set up the parameter configuration using already-defined `TCPCONFIG` macros from the Dynamic C `Rabbit4000\LIB\TCPIP\TCP_CONFIG.LIB` library at the beginning of your program as in the example below.

```
#define TCPCONFIG 1
```

There are two `TCPCONFIG` macros specifically set up for Wi-Fi applications with the RCM4400W module. (`TCPCONFIG 0` is not supported for Wi-Fi applications.)

<code>TCPCONFIG 1</code>	No DHCP
<code>TCPCONFIG 5</code>	DHCP enabled

These default IP address, netmask, nameserver, and gateway network parameters are set up for the `TCPCONFIG` macros.

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

The use of quotation marks in the examples described in this chapter is important since the absence of quotation marks will be flagged with warning messages when encrypted libraries such as the `WIFI_WLN_API.LIB` library are used.

### Wi-Fi Parameters

- Access Point SSID—`_WIFI_SSID`. This is the only mandatory parameter. Define the `_WIFI_SSID` macro to a string for the SSID of the access point in the infrastructure (BSS) mode, or the SSID of the ad-hoc network in the ad-hoc (IBSS) mode.

The default is shown below.

```
#define _WIFI_SSID "rabbitTest"
```

- Mode—`_WIFI_MODE` determines the mode:  
`WIFICONF_INFRASTRUCT` for the infrastructure mode, or `WIFICONF_ADHOC` for the ad-hoc mode.

The default is shown below.

```
#define _WIFI_MODE WIFICONF_INFRASTRUCT
```

- Your Own Channel—`_WIFI_OWNCCHANNEL` determines the channel on which to operate.

The default is shown below.

```
#define _WIFI_OWNCCHANNEL "0"
```

The default "0" means that any valid channel may be used by the requested SSID. This parameter is mandatory when creating an ad-hoc network. While it is optional for the infrastructure mode, it is usually best left at the default "0".

Note that there are restrictions on which channels may be used in certain countries. These are provided in Table 5 for some countries.

- Region/Country—`_WIFI_REGION_REQ` sets the channel range and maximum power limit to match the region selected. Table 5 lists the regions that are supported and their corresponding macros.

The region selected *must* match the region where the RCM4400W RabbitCore module will be used.

The default is shown below.

```
#define _WIFI_REGION_REQ _AMERICAS_REGION
```

- Disable/enable WEP encryption—`_WIFI_WEP_FLAG` indicates whether or not WEP encryption is being used.

The default (WEP encryption disabled) is shown below.

```
#define _WIFI_WEP_FLAG WIFICONF_WEP_DISABLE
```

The following WEP encryption options are available.

- `WIFICONF_WEP_DISABLE` — no WEP encryption is used.
- `WIFICONF_WEP_ENABLE` — use WEP encryption. You will need to define at least one WEP key (see below).
- `WIFICONF_WEP_TKIP` — use TKIP or WPA encryption. You will need to define a passphrase or a key for TKIP encryption, as well as define the `WIFI_USE_WPA` macro (see below).
- The following four encryption keys are provided. If WEP encryption is enabled, at least one key should be specified — do not use the defaults. You will have to modify these keys according to the encryption keys in effect for the Wi-Fi network you wish to access. A key is specified as either 5 or 13 comma-separated byte values.

```
#define _WIFI_KEY0 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef, 0x01,
0x23, 0x45, 0x67, 0x89
```

```
#define _WIFI_KEY1 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef, 0x01,
0x23, 0x45, 0x67, 0x89
```

```
#define _WIFI_KEY2 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef, 0x01,
0x23, 0x45, 0x67, 0x89
```

```
#define _WIFI_KEY3 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef, 0x01,
0x23, 0x45, 0x67, 0x89
```

- Select encryption key—`_WIFI_USEKEY` indicates which `_WIFI_KEYn` key to use.

The default shown below indicates that key 0, defined by `_WIFI_KEY0`, will be used.

```
#define _WIFI_USEKEY "0"
```

- Use WPA encryption.

The following macro must also be used with WPA encryption.

```
#define WIFI_USE_WPA
```

When using WPA encryption, `_WIFI_WEP_FLAG` must be defined as `WIFICONF_WEP_TKIP`, and you must define a WPA key using `_WIFI_PSK_PASSPHRASE` or `_WIFI_PSK_HEX`:

- Set WPA passphrase—`_WIFI_PSK_PASSPHRASE` is a string that matches the passphrase on your access point. It may also point to a variable.

Define an ASCII passphrase here, from 1 to 63 characters long. This passphrase is only used if you did not specify a hexadecimal key for the `_WIFI_PSK_HEX` macro. The insecure default is shown below.

```
#define _WIFI_PSK_PASSPHRASE "now is the time"
```

- Set WPA hexadecimal key—`_WIFI_PSK_HEX` is a string of hexadecimal digits that matches the 256-bit (64-byte) hexadecimal key used by your access point.

Specify a 64 hexadecimal digit (32 bytes) key here. This key will be used and will override any passphrase set with the `_WIFI_PSK_PASSPHRASE` macro. The example hex key shown below

```
#define _WIFI_PSK_HEX \  
"57A12204B7B350C4A86A507A8AF23C0E81D0319F4C4C4AE83CE3299EFE1FCD27"
```

is valid for the SSID `"rabbitTest"` and the passphrase `"now is the time"`.

Using a passphrase is rather slow. It takes a Rabbit 4000 more than 20 seconds to generate the actual 256-bit key from the passphrase. If you use a passphrase and `#define WIFI_VERBOSE_PASSPHRASE`, the Wi-Fi library will helpfully print out the hex key corresponding to that passphrase and SSID.

If both `_WIFI_PSK_HEX` and `_WIFI_PSK_PASSPHRASE` are defined, `_WIFI_PSK_HEX` will be used, and `_WIFI_PSK_PASSPHRASE` will be ignored.

- Authentication algorithm—`_WIFI_AUTH_MODE` can be used to limit the authentication modes used.

The default shown below allows enables both open-system authentication and shared-key authentication.

```
#define _WIFI_AUTH_MODE WIFICONF_AUTH_ALL
```

The following authentication options are available.

- `WIFICONF_AUTH_OPEN_SYS` — only use open authentication.
- `WIFICONF_AUTH_SHARED_KEY` — only use shared-key authentication (useful for WEP only).

- Fragmentation threshold—`_WIFI_FRAG_THRESH` sets the fragmentation threshold. Frames (or packets) that are larger than this threshold are split into multiple fragments. This can be useful on busy or noisy networks. The value can be between "256" and "2346".

The default, "0", means no fragmentation.

```
#define _WIFI_FRAG_THRESH "0"
```

- RTS threshold—`_WIFI_RTS_THRESH` sets the RTS threshold, the frame size at which the RTS/CTS mechanism is used. This is sometimes useful on busy or noisy networks. Its range is "1" to "2347".

The default, "0", means no RTS/CTS.

```
#define _WIFI_RTS_THRESH "0"
```

Examples are available within Dynamic C. Select “Function Lookup” from the **Help** menu, or press **<ctrl-H>**. Type “TCPCONFIG” in the Function Search field, and hit **<Enter>**. Scroll down to the section on “Wi-Fi Configuration.” The *Dynamic C TCP/IP User’s Manual*. (Volume 1) provides additional information about these macros and Wi-Fi.

It is also possible to redefine any of the above parameters dynamically using the `wifi_ioctl()` function call. Macros for alternative Wi-Fi configurations are provided with the `wifi_ioctl()` function call, and may be used to change the above default macros or configurations.

### 6.3.2 Configuring Dynamic C at Run Time

There is one basic function call used to configure the Wi-Fi settings.

---

---

#### wifi\_ioctl

---

---

```
int wifi_ioctl(int iface, int cmd, char* data, int len);
```

#### DESCRIPTION

This function call is used to configure the Wi-Fi interface, including setting the SSID, the mode, WEP keys, etc. It can also be used to get status information and to request a Wi-Fi scan.

Note that the Wi-Fi interface must be down when you are using the following commands that change the configuration — **WIFI\_SSID**, **WIFI\_MULTI\_DOMAIN**, **WIFI\_COUNTRY\_SET**, **WIFI\_MODE**, **WIFI\_OWNCHAN**, and **WIFI\_WEP\_FLAG**. The **wifi\_ioctl()** function description in the **WIFI\_WLN\_API.LIB** library provides sample code to demonstrate how to bring down the Wi-Fi interface to change these configurations.

#### PARAMETERS

**iface** specifies the Wi-Fi interface number for the RCM4400W (use **IF\_WIFIO** or **IF\_DEFAULT**)

The **cmd**, **data**, and **len** parameters are described in detail below. Each **cmd** (command) has different requirements for the **data** and **len** parameters. Note that these parameters are strings in all cases, even for “numeric” parameters. The Wi-Fi interface must be down when you are using the shaded commands that change the configuration.

cmd	data	len	Description
WIFI_SSID	char*	0–32	Sets SSID string
WIFI_MULTI_DOMAIN	char*	0	"0"—disable 802.11d country info capability "1"—enable 802.11d country info capability
WIFI_COUNTRY_SET	int*	0	0 through 9 to set channel range and power limits for selected country (see Table 5)
WIFI_COUNTRY_GET	_wifi_country*	0	Data structure with country-specific information
WIFI_MODE	char*	0	WIFICONF_INFRASTRUCT or WIFICONF_ADHOC
WIFI_OWNCHAN	char*	0	"0" through "13" decimal-coded string
WIFI_WEP_FLAG	char*	0	WIFICONF_WEP_DISABLE, WIFICONF_WEP_ENABLE, or WIFICONF_WEP_TKIP

cmd	data	len	Description
WIFI_WEP_USEKEY	char*	0	"0" through "3"
WIFI_WEP_KEY0	char []	5 or 13	64-bit or 128-bit key
WIFI_WEP_KEY1	char []	5 or 13	64-bit or 128-bit key
WIFI_WEP_KEY2	char []	5 or 13	64-bit or 128-bit key
WIFI_WEP_KEY3	char []	5 or 13	64-bit or 128-bit key
WIFI_AUTH	char*	0	WIFICONF_AUTH_OPEN_SYS, WIFICONF_AUTH_SHARED_KEY, or WIFICONF_AUTH_ALL
WIFI_WPA_PSK_PASSPHRASE	char*	0	ASCII string of 1 to 63 characters, null terminated, sets a key for the previously specified WIFI_SSID value
WIFI_WPA_PSK_HEX	char*	0	ASCII string of exactly 64 hexadecimal characters, null terminated, sets the WPA_PSK master key
WIFI_TX_RATE	char*	0	WIFICONF_RATE_1MBPS, WIFICONF_RATE_2MBPS, WIFICONF_RATE_5_5MBPS, WIFICONF_RATE_11MBPS or WIFICONF_RATE_ANY
WIFI_TX_POWER	char*	0	"0" through "15" (the actual range used depends on the country setting)
WIFI_FRAG_THRESH	char*	0	"0" (off) or "256" through "2346"
WIFI_RTS_THRESH	char*	0	"0" through "2347"
WIFI_SCANCB	void*	0	Pointer to the scan callback function call
WIFI_SCAN	NULL	0	Initiates a Wi-Fi scan
WIFI_STATUSGET	wifi_status*	0	Returns status information

In the data column:

**char\*** indicates that data argument is a string, and the **len** field is ignored

**char []** indicates that the argument is a character array, and **len** indicates the size

If you don't want encryption enabled, do not execute the **WIFI\_WEP\_FLAG** command in the table.

#### RETURN VALUE

0 = success

-1 = error (invalid command or parameter)

Use each command macro in its own `wifi_ioctl()` function call. For example, to name the “rabbit” access point and set a transmit rate of 11 Mbits/s, you would have these two lines of code in your program.

```
int wifi_ioctl(IF_WIFIO, WIFI_SSID, "rabbit", 0);
int wifi_ioctl(IF_WIFIO, WIFI_TX_RATE, WIFICONF_RATE_11MBPS, 0);
```

Let’s look at the individual `wifi_ioctl()` commands and their macro options.

### **WIFI\_SSID**

An SSID (service set identifier) names a specific wireless LAN (WLAN). All devices on a single WLAN must share a common SSID. Set this value to your WLAN’s SSID. If you leave the SSID blank, the Rabbit-based device will associate automatically with the access point that has the strongest signal. Generally, it is best to set the SSID explicitly so that the device does not join a WLAN that you were not expecting it to join.

For an infrastructure network (one that uses an access point), this is the name of the network as configured on the access point.

For an ad-hoc network, this is the name that you want to give the network you created. All devices on the ad-hoc network must use the same SSID.

### **WIFI\_MULTI\_DOMAIN**

This command enables or disables your device to be configured by an access point that is capable of supporting multiple domains according to the 802.11d standard. When your device is enabled, the access point will provide country information to your device to identify the regulatory domain in which it is located and to configure its PHY for operation in that regulatory domain.

**NOTE:** The access point must have the 802.11d option enabled with the country selected according to where your wireless device is deployed.

## WIFI\_COUNTRY\_SET

This command sets the channel range and maximum power limit for the country selected. The country you select will set the maximum power limit and channel range automatically, Rabbit strongly recommends checking the regulations for the country where your wireless devices will be deployed for any specific requirements. *Any attempt to operate a device outside the allowed channel range or power limits will void your regulatory approval to operate the device in that country.*

The following regions have macros and region numbers defined for convenience.

**Table 5. Worldwide Wi-Fi Macros and Region Numbers**

Region	Macro	Region Number	Channel Range
Americas	<code>_AMERICAS_REGION</code>	0	1–11
Mexico	<code>_MEXICO_REGION_INDOORS</code>	1	1–11 (indoors)
	<code>_MEXICO_REGION_OUTDOORS</code>	2	9–11 (outdoors)
Canada	<code>_CANADA_REGION</code>	3	1–11
Europe, Middle East, Africa, except France	<code>_EMEA_REGION</code>	4	1–13
France	<code>_FRANCE_REGION</code>	5	10–13
Israel	<code>_ISRAEL_REGION</code>	6	3–11
China	<code>_CHINA_REGION</code>	7	1–11
Japan	<code>_JAPAN_REGION</code>	8	1–13
Australia	<code>_AUSTRALIA_REGION</code>	9	1–11

The following sample code shows how to set Australia.

```
auto int country;
country = _AUSTRALIA_REGION;
wifi_ioctl(IF_WIFIO, WIFI_COUNTRY_SET, &country, 0);
```



## WIFI\_COUNTRY\_GET

This command returns country-specific information into the user-supplied buffer (or data structure) area. Accordingly, you must ensure there is enough space in the buffer for the entire data structure. Be sure the data pointer points to a buffer that is large enough to hold `sizeof(_wifi_country)`.

The `wifi_status` structure has the following definition.

```
typedef struct {
    char id;                // Country ID
    char country[16];       // Country name
    int first_channel;      // First channel
    int last_channel;       // Last channel
    unsigned int channel_mask; // Channel mask
    int max_pwr_dBm;        // Max power, dBm
    int max_pwr_index;      // Max Power index
    _wifi_country;
```

## WIFI\_MODE

Sets whether the Wi-Fi device should attach to an infrastructure network (**WIFICONF\_INFRASTRUCT**), which is the most common configuration, or an ad-hoc network (**WIFICONF\_ADHOC**). Access points are used with infrastructure networks, and coordinates communication among all the associated devices. No wireless access points are associated with the ad-hoc mode. This allows devices (such as Rabbit-based devices and notebooks) to communicate with each other directly as peer devices without an access point.

## WIFI\_OWNCCHAN

This parameter specifies the channel the Wi-Fi device uses in your network when operating in the ad-hoc mode. Set this parameter to "0" in an infrastructure network to allow the Wi-Fi driver to pick the channel automatically for the given SSID. For an ad-hoc network, this channel must be set to "1" through "13". Use the **WIFI\_COUNTRY\_GET** command to get the valid range of channels for the country where the device will be used.

**NOTE:** Regional regulations may not allow some channels to be used.

## WIFI\_WEP\_FLAG

The encryption flag can have one of three values—disabled (**WIFICONF\_WEP\_DISABLE**), WEP encryption enabled (**WIFICONF\_WEP\_ENABLE**), or TKIP/WPA encryption enabled (**WIFICONF\_WEP\_TKIP**). You can use either 40-bit (5-byte) or 104-bit (13-byte) keys for WEP (Wired Equivalent Privacy).

## WIFI\_WEP\_USEKEY

Indicates which key ("0"—"3") is the default transmission key. The setting may be left at the "0" default. The setting of the WEP keys is described below.

## WIFI\_WEP\_KEY0–3

These are the secret keys that are programmed into each device on a WLAN to use WEP (Wired Equivalent Privacy). Each of these keys must be entered correctly in order for WEP to work.

Each of the four WEP keys is an array of either 5 or 13 *binary bytes*, not an ASCII string. Set `len` to 5 for a 40-bit key, or 13 for a 104-bit key. Marketing literature sometimes refers to these as 64-bit or 128-bit keys. The 24 “extra” bits that are included in the marketing description serve as a cryptographic initialization vector.

## WIFI\_AUTH

The authentication option is used to configure different types of authentication that the Wi-Fi device supports. There are three types of authentication that are supported—open-system authentication (`WIFICONF_AUTH_OPEN_SYS`), shared-key authentication (`WIFICONF_AUTH_SHARED_KEY`), or both (`WIFICONF_AUTH_ALL`). The most important consideration is to use the same type of authentication as the access point you are planning on using; hence, `WIFICONF_AUTH_ALL` is the most flexible value.

## WIFI\_WPA\_PSK\_PASSPHRASE

This WPA option is only available if the `WIFI_USE_WPA` macro has been defined.

The command sets a key for the previously specified `WIFI_SSID` value. The key is computed as a hash of the passphrase and the target SSID, which could potentially take a long time to run. See the `PASSPHRASE.C` sample program for alternatives.

If your program (or TCP configuration) defines `_WIFI_PSK_PASSPHRASE` to a quoted string, then that string will be used automatically as a pass phrase, unless `_WIFI_PSK_HEX` is also defined (see the following command description).

## WIFI\_WPA\_PSK\_HEX

This WPA option is only available if the `WIFI_USE_WPA` macro has been defined.

The command sets a hexadecimal `WPA_PSK` master key. The string must be exactly 64 hexadecimal digits (using the characters 0–9 and a–f or A–F). This is interpreted as a byte string and parsed into the appropriate 32-byte binary key.

If your program (or TCP configuration) defines `_WIFI_PSK_HEX` to a quoted string of 64 hex digits, then that string will be used automatically as the PSK master key.

## WIFI\_TX\_RATE

This command macro specifies the maximum transmit rate for the Wi-Fi device. This rate is reduced as necessary depending on the quality of the wireless connection. The options are:

1 Mbits/s (**WIFICONF\_RATE\_1MBPS**)

2 Mbits/s (**WIFICONF\_RATE\_2MBPS**)

5.5 Mbits/s (**WIFICONF\_RATE\_5\_5MBPS**)

11 Mbits/s (**WIFICONF\_RATE\_11MBPS**)

**WIFICONF\_RATE\_ANY** to use the highest data rate available.

## WIFI\_TX\_POWER

Sets the transmit power for the Wi-Fi device. A higher transmit power will result in higher dBm. Use the **WIFI\_COUNTRY\_GET** command to get the power limit setting for the country where the device will be used.

**NOTE:** Regional regulations may not allow the full range of possible power settings to be used.

## WIFI\_FRAG\_THRESH

Sets the threshold (in bytes) beyond which a frame must be fragmented when transmitted. This can be useful on a very busy or noisy network, since frame corruption will be limited to the size of a fragment rather than the whole frame. This means that only the fragment will need to be retransmitted. To be effective, the fragmentation threshold will need to be set on all wireless devices on the network as well as on the access point.

## WIFI\_RTS\_THRESH

Sets the threshold (in bytes) beyond which an RTS (request to send) frame must be sent before the data frame can be sent. This can sometimes help performance with busy networks, although it is not used frequently.

## WIFI\_SCANCB

Sets up a user callback function that will be called when a user-requested scan has completed. The callback function must have the following function prototype. (The name of the function may be different.)

```
root void scan_callback(far wifi_scan_data* data);
```

The scan data will be provided in the data parameter. This structure has the following definition.

```
#define _WIFI_SCAN_NUM

typedef struct {
    int count;
    _wifi_wln_scan_bss bss[_WIFI_SCAN_NUM];
} wifi_scan_data;
```

**count** will have the number of access points that were detected.

**bss** is an array where each element corresponds to a detected access point.

**\_wifi\_wln\_scan\_bss** is a structure that has the following definition.

```
typedef struct {
    uint8 ssid[WLN_SSID_SIZE];
    int ssid_len;
    int channel;
    mac_addr bss_addr;
    uint16 bss_caps;
    uint8 wpa_info[WLN_WPAIE_SIZE];
    uint8 erp_info;
    uint16 rates;
    uint16 rates_basic;
    uint16 atim;
    int tx_rate;
    int rx_signal;
} _wifi_wln_scan_bss;
```

The structure elements have the following definitions:

**ssid** = service set ID (max. length 32)

**ssid\_len** = SSID length in bytes

**channel** = channel number (1–13)

**bss\_addr** = BSS ID (access point MAC address)

**bss\_caps** reserved

**wpa\_info** reserved

**erp\_info** reserved

**rates** reserved

**rates\_basic** reserved

**atim** reserved

**tx\_rate** = maximum transmit rate (in 100 kbps)

**rx\_signal** = received signal strength (0–107)

## WIFI\_SCAN

Initiates a Wi-Fi scan. When the scan has been completed, the configured scan callback function (see above) will be called. The callback function must have already been configured before using this command. A Wi-Fi scan will interrupt the network connectivity briefly since the scan must iterate through the channels on the wireless network.

## WIFI\_STATUSGET

When using this command, you must ensure there is enough space for the entire data structure. Be sure the data pointer points to a buffer that is large enough to hold `sizeof(wifi_status)`.

This command returns status information into the user-supplied buffer (or data structure) area. The `wifi_status` structure has the following definition.

```
typedef struct {
    wln_state state;
    uint8 ssid[WLN_SSID_SIZE];
    int ssid_len;
    int channel;
    mac_addr bss_addr;
    uint16 bss_caps;
    uint8 wpa_info[WLN_WPAIE_SIZE];
    uint32 authen;
    uint32 encrypt;
    int tx_rate;
    int rx_rate;
    int rx_signal;
    int tx_power;
    uint8 country_info[WLN_COUNTRY_STRLEN];
} wifi_status;
```

The structure elements have the following definitions.

**state** = association state: one of **WLN\_ST\_XXX** (see below)

**ssid** = current service set ID (SSID)

**ssid\_len** = service set ID length

**channel** = current channel (1–13)

**bss\_addr** = BSS ID (access point MAC address)

**bss\_caps** reserved

**wpa\_info** reserved

**authen** reserved

**encrypt** reserved

**tx\_rate** = current transmit rate (in 100 kbps)

**rx\_rate** = last received rate (in 100 kbps)

**rx\_signal** = last received signal strength (0–107)

**tx\_power** reserved

**country\_info** reserved

The **state** structure element can provide more information on the current state of the Wi-Fi driver. It can have the following values.

**WLN\_ST\_STOPPED** = Wi-Fi driver is stopped

**WLN\_ST\_SCANNING** = currently performing a scan

**WLN\_ST\_ASSOC\_ESS** = associated with an access point

**WLN\_ST\_AUTH\_ESS** = authenticated with an access point

**WLN\_ST\_JOIN\_IBSS** = joined an existing ad-hoc network

**WLN\_ST\_START\_IBSS** = started an ad-hoc network

### 6.3.3 Other Key Function Calls

Remember to call `sock_init()` after all the Wi-Fi parameters have been defined. The Wi-Fi interface will be up automatically as long as you configured Dynamic C at compile time with one of the `TCPCONFIG` macros. Otherwise the Wi-Fi interface is neither up nor down, and must be brought up explicitly by calling either `ifup(IF_WIFI0)` or `ifconfig(IF_WIFI0,...)`. You must bring the interface down when you configure Dynamic C at run time before modifying any parameters that require the interface to be down (see Section 6.3.2) by calling `ifdown(IF_WIFI0)` or `ifconfig(IF_WIFI0,...)`. Then bring the interface back up.

Finally, no radio transmission occurs until you call `tcp_tick(NULL)`.

Instead of executing the above sequence based on `sock_init()`, you could use `sock_init_or_exit(1)` as a debugging tool to transmit packets (ARP, DHCP, association, and authentication) while bringing up the interface and to get the IP address.

## 6.4 Where Do I Go From Here?

**NOTE:** If you purchased your RCM4400W through a distributor or through a Rabbit partner, contact the distributor or partner first for technical support.

If there are any problems at this point:

- Use the Dynamic C **Help** menu to get further assistance with Dynamic C.
- Check the Rabbit Technical Bulletin Board and forums at [www.rabbit.com/support/bb/](http://www.rabbit.com/support/bb/) and at [www.rabbitcom/forums/](http://www.rabbitcom/forums/).
- Use the Technical Support e-mail form at [www.rabbit.com/support/](http://www.rabbit.com/support/).

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

*An Introduction to TCP/IP* and the *Dynamic C TCP/IP User's Manual* provide background and reference information on TCP/IP, and are available on the CD and on our [Web site](#).



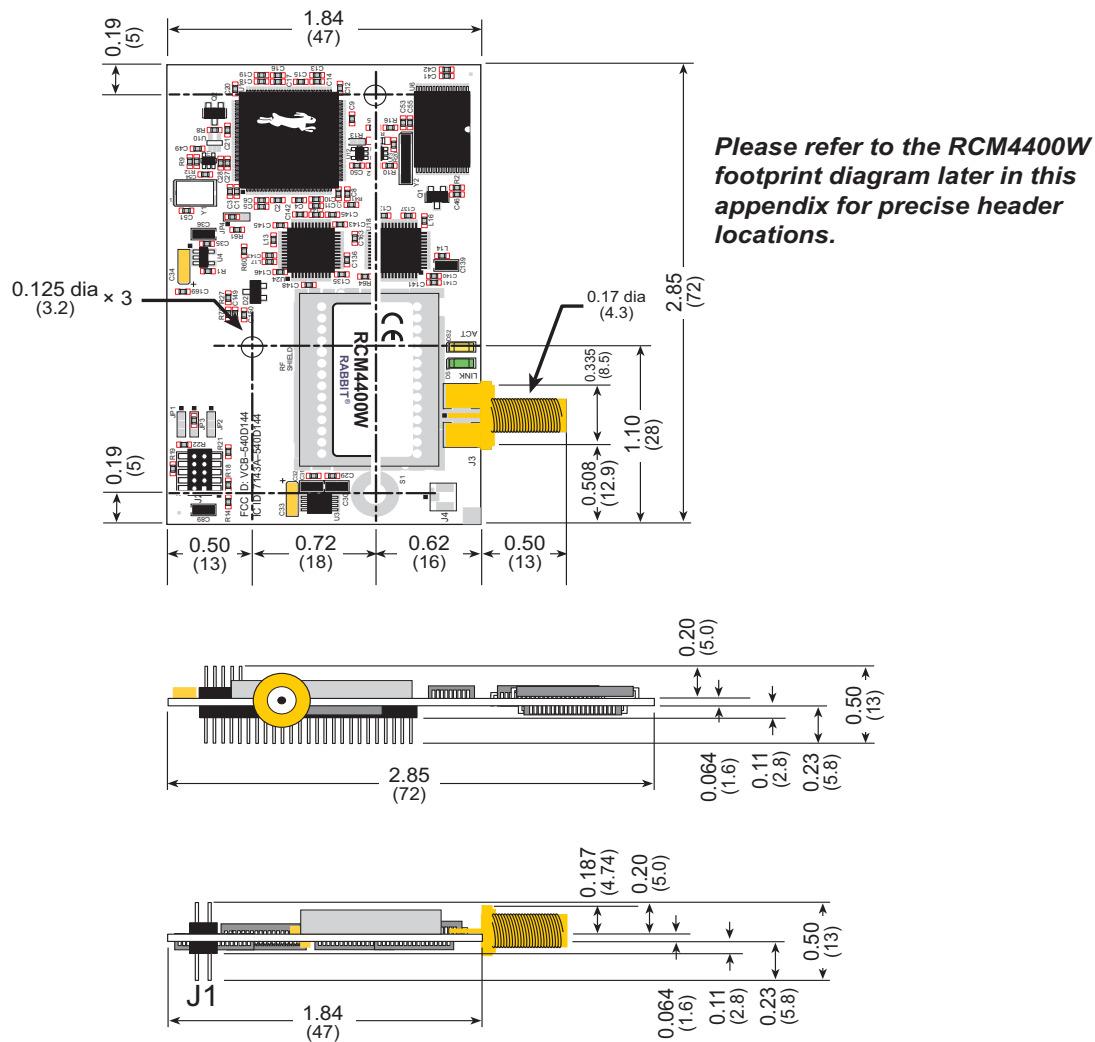


## **APPENDIX A. RCM4400W SPECIFICATIONS**

Appendix A provides the specifications for the RCM4400W, and describes the conformal coating.

## A.1 Electrical and Mechanical Characteristics

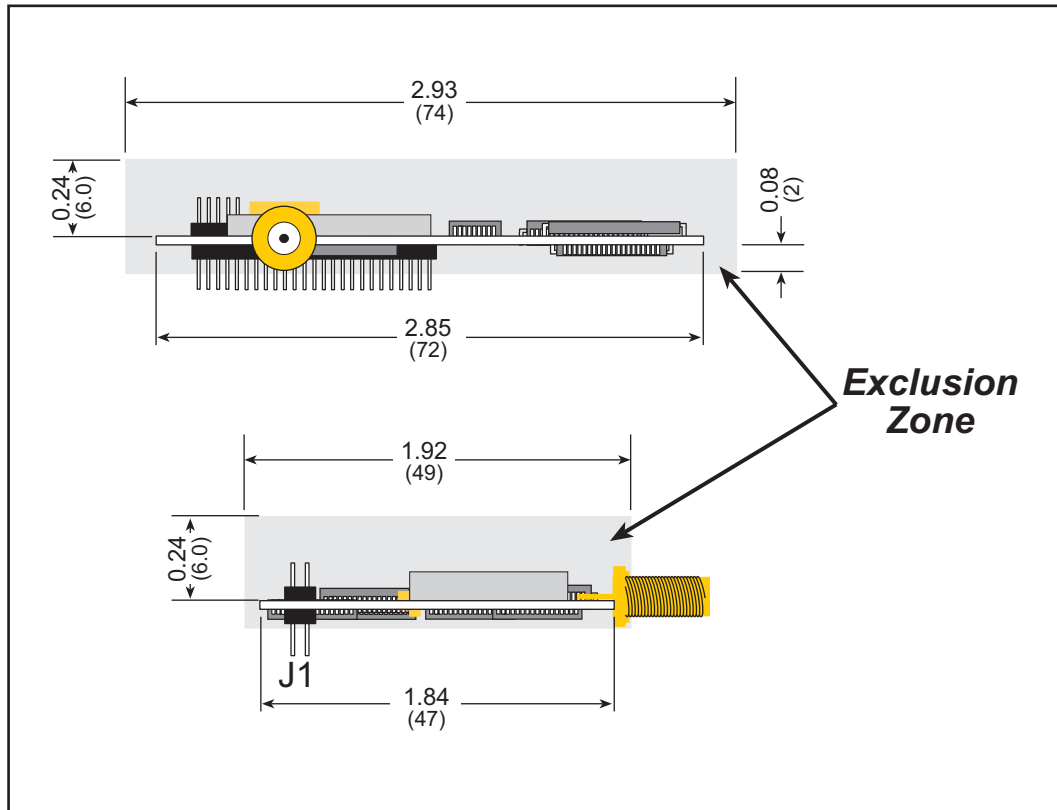
Figure A-1 shows the mechanical dimensions for the RCM4400W.



**Figure A-1. RCM4400W Dimensions**

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

It is recommended that you allow for an “exclusion zone” of 0.04" (1 mm) around the RCM4400W in all directions when the RCM4400W is incorporated into an assembly that includes other printed circuit boards. An “exclusion zone” of 0.08" (2 mm) is recommended below the RCM4400W when the RCM4400W is plugged into another assembly. Figure A-2 shows this “exclusion zone.”



**Figure A-2. RCM4400W “Exclusion Zone”**

**NOTE:** There is an antenna associated with the RCM4400W RabbitCore modules. Do not use any RF-absorbing materials in these vicinities in order to realize the maximum range.

If you are planning to mount your RCM4400W directly in a panel-mounted enclosure, the RP-SMA antenna connector will extend outside the enclosure. Keep the thickness of the enclosure plus washer and lock nut to less than 0.2" (5 mm) to make sure that the antenna can be mounted securely in the RP-SMA antenna connector.

Table A-1 lists the electrical, mechanical, and environmental specifications for the RCM4400W.

**Table A-1. RCM4400W Specifications**

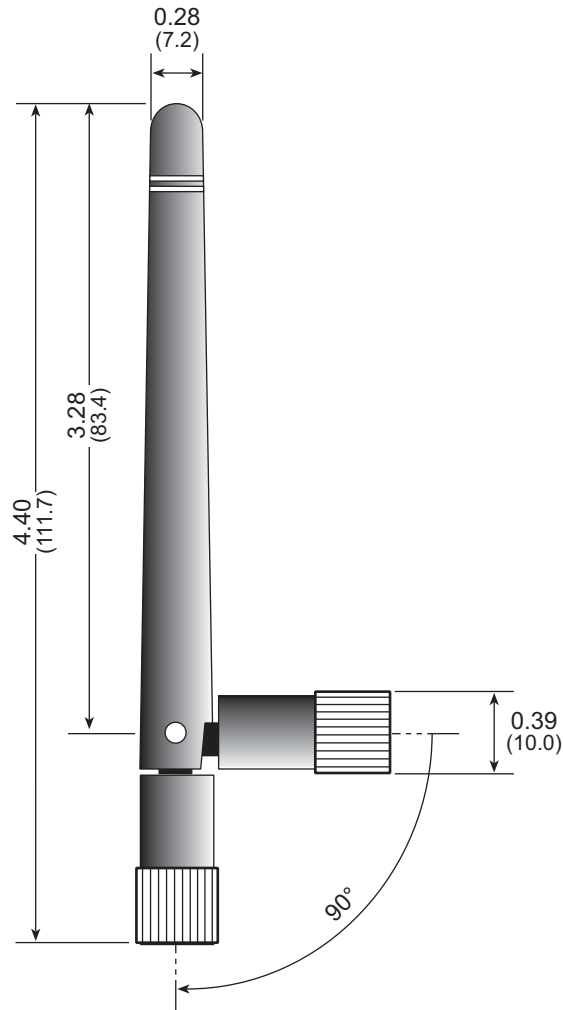
Parameter	RCM4400W
Microprocessor	Rabbit® 4000 at 58.98 MHz
Data SRAM	512KB
Program Execution Fast SRAM	512KB
Flash Memory	512KB
Serial Flash Memory	1MB (reserved for Wi-Fi FPGA, 800KB available when using Dynamic C v. 10.54 or later)
Backup Battery	Connection for user-supplied backup battery (to support RTC and data SRAM)
General Purpose I/O	up to 38 parallel digital I/O lines configurable with four layers of alternate functions
Additional Inputs	Reset in
Additional Outputs	Reset out
Auxiliary 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
Serial Ports	6 high-speed, CMOS-compatible ports: <ul style="list-style-type: none"> <li>all 6 configurable as asynchronous (with IrDA), 4 as clocked serial (SPI), and 2 as SDLC/HDLC</li> <li>1 asynchronous clocked serial port shared with programming port</li> <li>1 clocked serial port shared with serial flash</li> </ul>
Serial Rate	Maximum asynchronous baud rate = CLK/8
Slave Interface	Slave port allows the RCM4400W to be used as an intelligent peripheral device slaved to a master processor
Real Time Clock	Yes
Timers	Ten 8-bit timers (6 cascadable from the first), one 10-bit timer with 2 match registers, and one 16-bit timer with 4 outputs and 8 set/reset registers
Watchdog/Supervisor	Yes
Pulse-Width Modulators	4 channels synchronized PWM with 10-bit counter 4 channels variable-phase or synchronized PWM with 16-bit counter
Input Capture	2-channel input capture can be used to time input signals from various port pins

**Table A-1. RCM4400W Specifications (continued)**

Parameter	RCM4400W
Quadrature Decoder	2-channel quadrature decoder accepts inputs from external incremental encoder modules
Power (pins unloaded)	3.3 V.DC $\pm 5\%$
	450 mA @ 3.3 V while transmitting/receiving 80 mA @ 3.3 V while not transmitting/receiving
Operating Temperature	-30°C to +75°C
Humidity	5% to 95%, noncondensing
Connectors	One RP-SMA antenna connector One 2 $\times$ 25, 1.27 mm pitch IDC signal header One 2 $\times$ 5, 1.27 mm pitch IDC programming header
Board Size	1.84" $\times$ 2.85" $\times$ 0.50" (47 mm $\times$ 72 mm $\times$ 13 mm)
<b>Wi-Fi</b>	
Antenna Power Output	40 mW (16 dBm)
Compliance	802.11b, 2.4 GHz

### A.1.1 Antenna

The RCM4400W Development Kit includes a 2.4 GHz (+2 dB) dipole antenna whose dimensions are shown in Figure A-3.



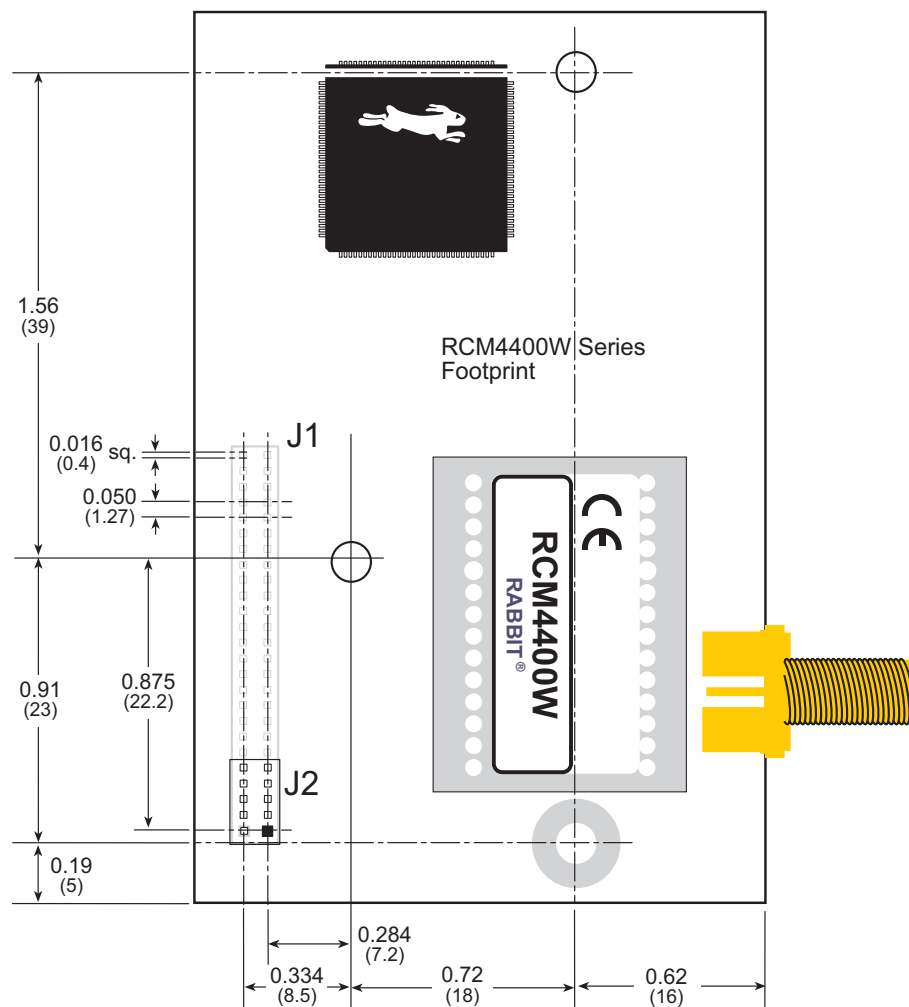
**Figure A-3. RCM4400W Development Kit Dipole Antenna**

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

### A.1.2 Headers

The RCM4400W uses a header at J1 for physical connection to other boards. J1 is a  $2 \times 25$  SMT header with a 1.27 mm pin spacing. J2, the programming port, is a  $2 \times 5$  header with a 1.27 mm pin spacing

Figure A-4 shows the layout of another board for the RCM4400W to be plugged into. These reference design values are relative to one of the mounting holes.



**Figure A-4. User Board Footprint for RCM4400W**

## A.2 Rabbit 4000 DC Characteristics

**Table A-2. Rabbit 4000 Absolute Maximum Ratings**

Symbol	Parameter	Maximum Rating
$T_A$	Operating Temperature	-40° to +85°C
$T_S$	Storage Temperature	-55° to +125°C
$V_{IH}$	Maximum Input Voltage	$V_{DDIO} + 0.3 \text{ V}$ (max. 3.6 V)
$V_{DDIO}$	Maximum Operating Voltage	3.6 V

Stresses beyond those listed in Table A-2 may cause permanent damage. The ratings are stress ratings only, and functional operation of the Rabbit 4000 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 4000 chip.

Table A-3 outlines the DC characteristics for the Rabbit 4000 at 3.3 V over the recommended operating temperature range from  $T_A = -40^\circ\text{C}$  to  $+85^\circ\text{C}$ ,  $V_{DDIO} = 3.0 \text{ V}$  to  $3.6 \text{ V}$ .

**Table A-3. 3.3 Volt DC Characteristics**

Symbol	Parameter	Min	Typ	Max
$V_{DDIO}$	I/O Ring Supply Voltage, 3.3 V	3.0 V	3.3 V	3.6 V
	I/O Ring Supply Voltage, 1.8 V	1.65 V	1.8 V	1.90 V
$V_{IH}$	High-Level Input Voltage ( $V_{DDIO} = 3.3 \text{ V}$ )		2.0 V	
$V_{IL}$	Low-Level Input Voltage ( $V_{DDIO} = 3.3 \text{ V}$ )		0.8 V	
$V_{OH}$	High-Level Output Voltage ( $V_{DDIO} = 3.3 \text{ V}$ )		2.4 V	
$V_{OL}$	Low-Level Output Voltage ( $V_{DDIO} = 3.3 \text{ V}$ )		0.4 V	
$I_{IO}$	I/O Ring Current @ 29.4912 MHz, 3.3 V, 25°C			12.2 mA
$I_{DRIVE}$	All other I/O (except TXD+, TXDD+, TXD-, TXDD-)			8 mA



### A.3 I/O Buffer Sourcing and Sinking Limit

Unless otherwise specified, the Rabbit I/O buffers are capable of sourcing and sinking 8 mA of current per pin at full AC switching speed. Full AC switching assumes a 29.4 MHz CPU clock with the clock doubler enabled and capacitive loading on address and data lines of less than 70 pF per pin. The absolute maximum operating voltage on all I/O is 3.3 V  $\pm 5\%$ .

### A.4 Bus Loading

You must pay careful attention to bus loading when designing an interface to the RCM4400W. This section provides bus loading information for external devices.

Table A-4 lists the capacitance for the various RCM4400W I/O ports.

**Table A-4. Capacitance of Rabbit 4000 I/O Ports**

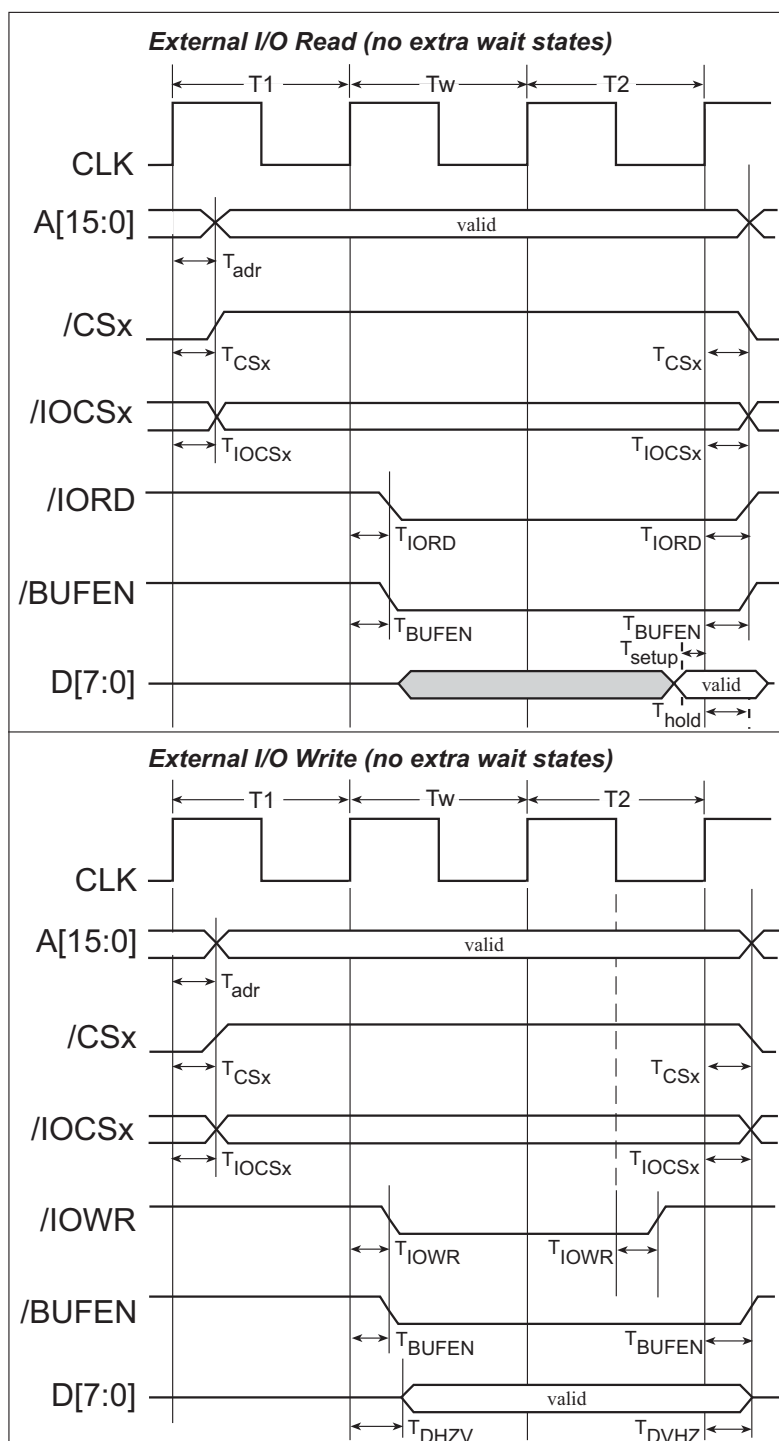
I/O Ports	Input Capacitance (pF)	Output Capacitance (pF)
Parallel Ports A to E	12	14

Table A-5 lists the external capacitive bus loading for the various RCM4400W 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-5.

**Table A-5. External Capacitive Bus Loading -20°C to +85°C**

Output Port	Clock Speed (MHz)	Maximum External Capacitive Loading (pF)
All I/O lines with clock doubler enabled	58.98	100

Figure A-5 shows a typical timing diagram for the Rabbit 4000 microprocessor external I/O read and write cycles.



**Figure A-5. External 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-6 lists the delays in gross memory access time for several values of  $VDD_{IO}$ .

**Table A-6. Preliminary Data and Clock Delays**

$VDD_{IO}$ (V)	Clock to Address Output Delay (ns)			Data Setup Time Delay (ns)	Worst-Case Spectrum Spreader Delay (ns)		
	30 pF	60 pF	90 pF		0.5 ns setting no dbl / dbl	1 ns setting no dbl / dbl	2 ns setting no dbl / dbl
3.3	6	8	11	1	2.3 / 2.3	3 / 4.5	4.5 / 9
1.8	18	24	33	3	7 / 6.5	8 / 12	11 / 22

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

- $T = -20^{\circ}\text{C}$  to  $85^{\circ}\text{C}$ ,  $V = VDD_{IO} \pm 10\%$
- Internal clock to nonloaded CLK pin delay  $\leq 1$  ns @  $85^{\circ}\text{C}/3.0$  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

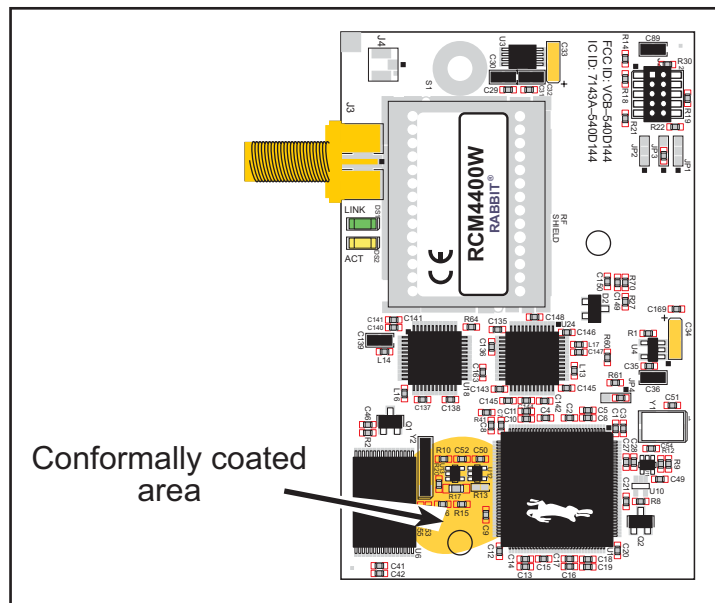
The data setup time delays are similar for both  $T_{setup}$  and  $T_{hold}$ .

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.

Rabbit's Technical Note TN227, *Interfacing External I/O with Rabbit Microprocessor Designs*, which is included with the online documentation, contains suggestions for interfacing I/O devices to the Rabbit 4000 microprocessors.

## A.5 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.



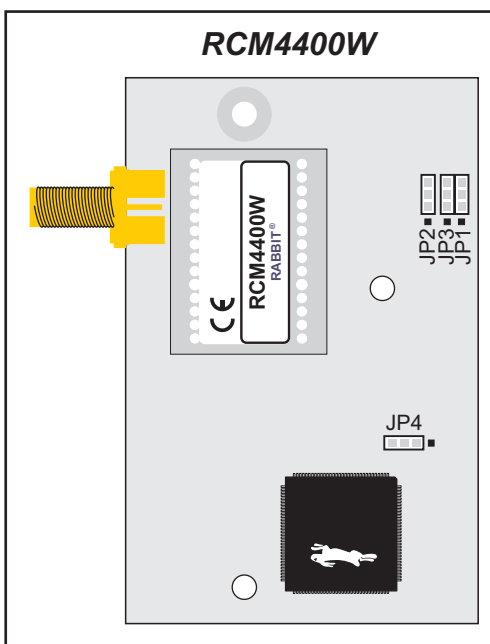
**Figure A-6. RCM4400W Areas Receiving Conformal Coating**

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 Rabbit's Technical Note TN303, *Conformal Coatings*, which is included with the online documentation.

## A.6 Jumper Configurations

Figure A-7 shows the header locations used to configure the various RCM4400W options via jumpers.



**Figure A-7. Location of RCM4400W Configurable Positions**

Table A-7 lists the configuration options.

**Table A-7. RCM4400W Jumper Configurations**

Header	Description	Pins Connected		Factory Default
JP1	PE6, FPGA Chip Select, or SMODE1 Output on J1 pin 38	1–2	PE6	Not stuffed.
		2–3	SMODE1	
JP2	PE5, FPGA Interrupt Output, or SMODE0 Output on J1 pin 37	1–2	PE5	Not stuffed.
		2–3	SMODE0	
JP3	PE7 or STATUS Output on J1 pin 39	1–2	PE7	×
		2–3	STATUS	
JP4	Reserved for future use.	1–2	PE0	×
		2–3	A20	

**NOTE:** The jumper connections are made using 0  $\Omega$  surface-mounted resistors.





## **APPENDIX B. PROTOTYPING BOARD**

Appendix B describes the features and accessories of the Prototyping Board, and explains the use of the Prototyping Board to demonstrate the RCM4400W and to build prototypes of your own circuits. The Prototyping Board has power-supply connections and also provides some basic I/O peripherals (RS-232, LEDs, and switches), as well as a prototyping area for more advanced hardware development.

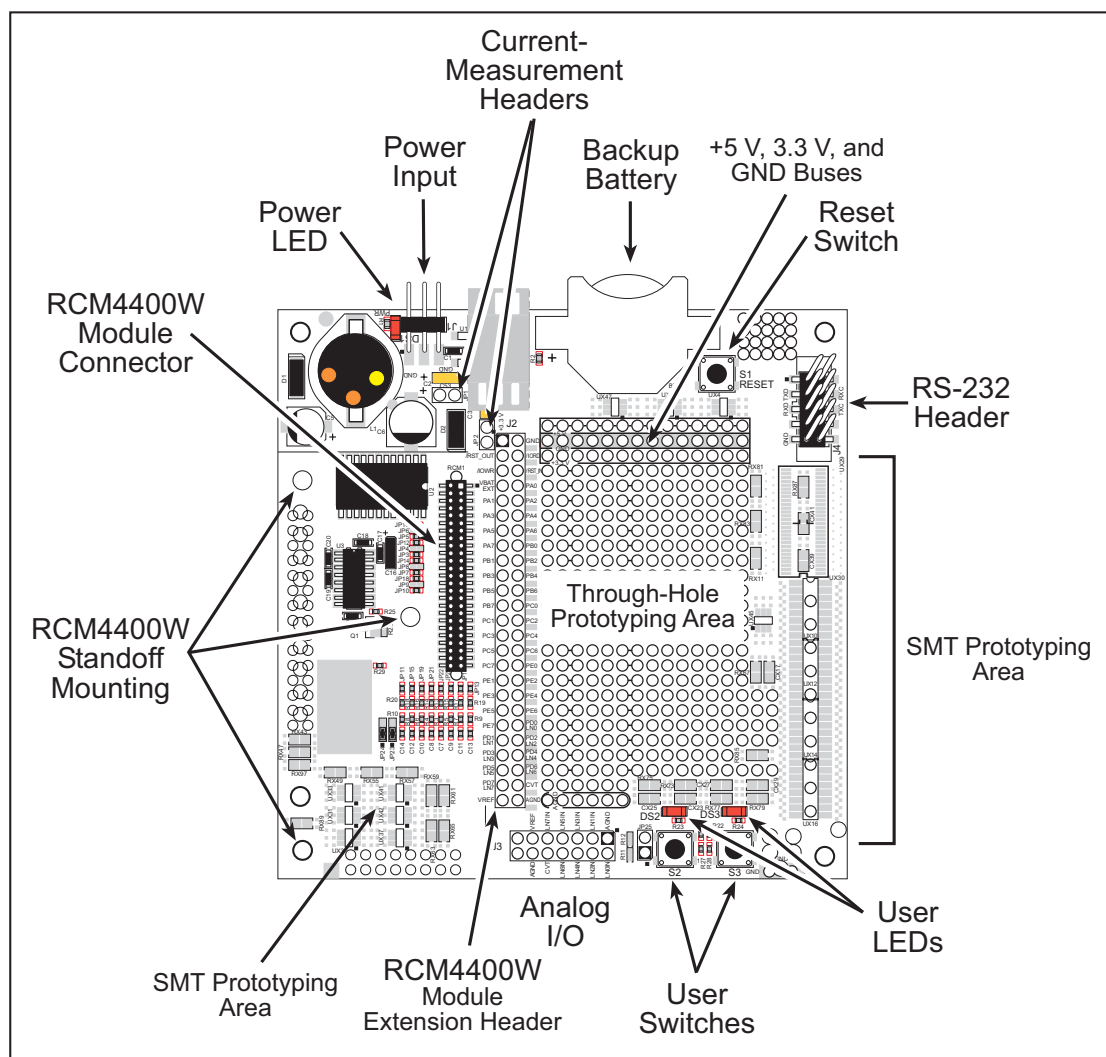
## B.1 Introduction

The Prototyping Board included in the Development Kit makes it easy to connect an RCM4400W module to a power supply and a PC workstation for development. It also provides some basic I/O peripherals (RS-232, 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 the RCM4400W module.

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 3-pin header is 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 is terminated with a header plug that connects to the 3-pin header in either orientation. The header plug leading to bare leads provided for overseas customers can be connected to the 3-pin header in either orientation.

Users providing their own power supply should ensure that it delivers 8–24 V DC at 8 W. The voltage regulators will get warm while in use.

- **Regulated Power Supply**—The raw DC voltage provided at the 3-pin header 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 RCM4400W module and the Prototyping Board.
- **Power LED**—The power LED lights whenever power is connected to the Prototyping Board.
- **Reset Switch**—A momentary-contact, normally open switch is connected directly to the RCM4400W'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 PB4 and PB5 pins of the RCM4400W module and may be read as inputs by sample applications.

Two LEDs are connected to the PB2 and PB3 pins of the RCM4400W module, and may be driven as output indicators by 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 around the edge of this area. Several areas for surface-mount devices are also available. (Note that there are SMT device pads on both top and bottom of the Prototyping Board.) Each SMT pad is connected to a hole designed to accept a 30 AWG solid wire.
- **Module Extension Header**—The complete pin set of the RCM4400W module is duplicated at header J2. Developers can solder wires directly into the appropriate holes, or, for more flexible development, a 2 × 25 header strip with a 0.1" pitch can be soldered into place. See Figure B-4 for the header pinouts.

**NOTE:** The same Prototyping Board can be used for several series of RabbitCore modules, and so the signals at J2 depend on the signals available on the specific RabbitCore module.

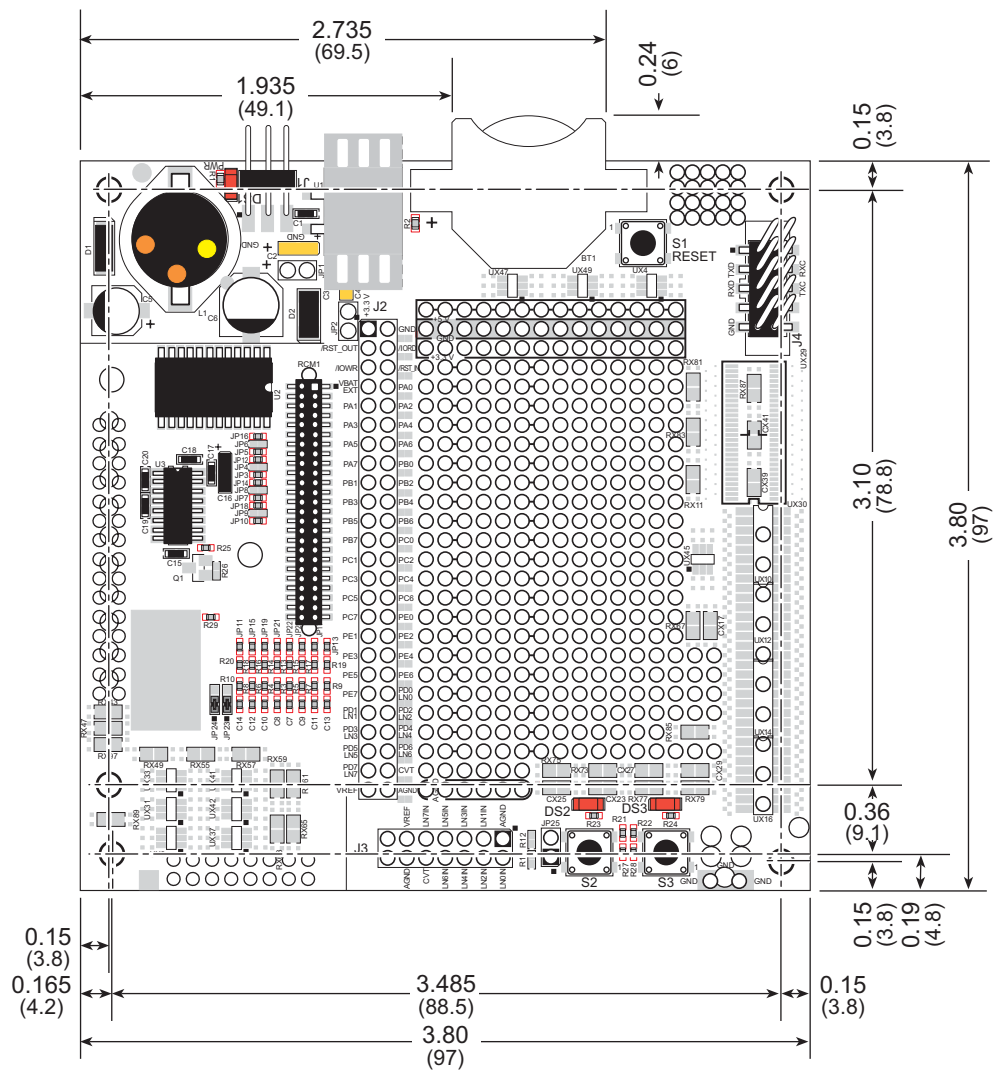
- **Analog Inputs Header**—The analog signals from a RabbitCore module are presented at header J3 on the Prototyping Board. These analog signals are connected via attenuator/filter circuits on the Prototyping Board to the corresponding analog inputs on the RabbitCore module.

**NOTE:** No analog signals are available on the Prototyping Board with the RCM4400W RabbitCore module installed since no analog signals are present on the RCM4400W's header J1.

- **RS-232**—Two 3-wire or one 5-wire RS-232 serial ports are available on the Prototyping Board at header J4. A 10-pin 0.1" pitch header strip installed at J4 allows you to connect a ribbon cable that leads to a standard DE-9 serial connector.
- **Current Measurement Option**—You may cut the trace below header JP1 on the bottom side of the Prototyping Board and install a 1 × 2 header strip from the Development Kit to allow you to use an ammeter across the pins to measure the current drawn from the +5 V supply. Similarly, you may cut the trace below header JP2 on the bottom side of the Prototyping Board and install a 1 × 2 header strip from the Development Kit to allow you to use an ammeter across the pins to measure the current drawn from the +3.3 V supply.
- **Backup Battery**—A 2032 lithium-ion battery rated at 3.0 V, 220 mA·h, provides battery backup for the RCM4400W data SRAM and real-time clock.

## 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. All dimensions have a manufacturing tolerance of  $\pm 0.01$ " (0.25 mm).

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

**Table B-1. Prototyping Board Specifications**

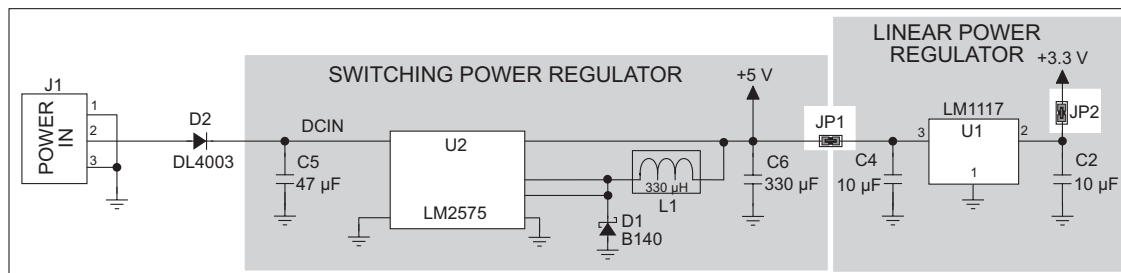
Parameter	Specification
Board Size	3.80" × 3.80" × 0.48" (97 mm × 97 mm × 12 mm)
Operating Temperature	0°C to +70°C
Humidity	5% to 95%, noncondensing
Input Voltage	8 V to 24 V DC
Maximum Current Draw (including user-added circuits)	800 mA max. for +3.3 V supply, 1 A total +3.3 V and +5 V combined
Prototyping Area	1.3" × 2.0" (33 mm × 50 mm) throughhole, 0.1" spacing, additional space for SMT components
Connectors	One 2 × 25 header socket, 1.27 mm pitch, to accept RCM4400W One 1 × 3 IDC header for power-supply connection One 2 × 5 IDC RS-232 header, 0.1" pitch Two unstuffed header locations for analog and RCM4400W signals 25 unstuffed 2-pin header locations for optional configurations

## B.3 Power Supply

The RCM4400W requires a regulated 3.0 V – 3.6 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 Schottky diode at D2 as shown in Figure B-3.



**Figure B-3. Prototyping Board Power Supply**

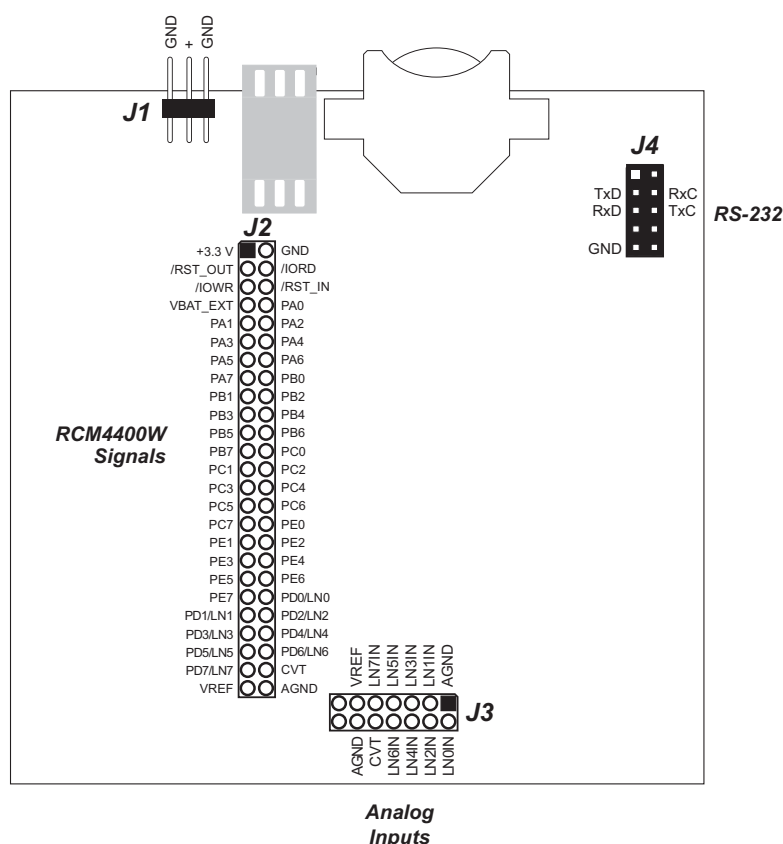
**TIP:** When you lay out your own power-supply circuit, place the switching voltage regulator as far away from the RCM4400W as possible to minimize RF noise, and use low-noise components such as a toroid coil.

## 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 to demonstrate the functionality of the RCM4400W right out of the box without any modifications to either board.

The Prototyping Board comes with the basic components necessary to demonstrate the operation of the RCM4400W. Two LEDs (DS2 and DS3) are connected to PB2 and PB3, and two switches (S2 and S3) are connected to PB4 and PB5 to demonstrate the interface to the Rabbit 4000 microprocessor. Reset switch S1 is the hardware reset for the RCM4400W.

The Prototyping Board provides the user with RCM4400W connection points brought out conveniently to labeled points at header J2 on the Prototyping Board. Although header J2 is unstuffed, a  $2 \times 25$  header is included in the bag of parts. RS-232 signals (Serial Ports C and D) are available on header J4. A header strip at J4 allows you to connect a ribbon cable, and a ribbon cable to DB9 connector is included with the Development Kit. The pinouts for these locations are shown in Figure B-4.



**Figure B-4. Prototyping Board Pinout**

The analog signals are brought out to labeled points at header location J3 on the Prototyping Board. Although header J3 is unstuffed, a  $2 \times 7$  header can be added. Note that analog signals are not available when the RCM4400W included in the Development Kit installed.

All signals from the RCM4400W module are available on header J2 of the Prototyping Board. The remaining ports on the Rabbit 4000 microprocessor are used for RS-232 serial communication. Table B-2 lists the signals on header J2 as configured by the `brdInit()` function call where applicable, and explains how they are used on the Prototyping Board.

**Table B-2. Use of RCM4400W Signals on the Prototyping Board**

Pin	Pin Name	Prototyping Board Use
1	+3.3 V	+3.3 V power supply
2	GND	
3	/RST_OUT	Reset output from reset generator
4	/IORD	External read strobe
5	/IOWR	External write strobe
6	/RESET_IN	Input to reset generator
8–15	PA0–PA7	Output, low
16	PB0	Serial flash SCLK
17	PB1	Output, high (programming port CLKA)
18	PB2	LED DS2 (output normally high/off)
19	PB3	LED DS3 (output normally high/off)
20	PB4	Switch S2 (input normally open/pulled up)
21	PB5	Switch S3 (input normally open/pulled up)
22–23	PB6–PB7	Output, high
24–25	PC0–PC1	Serial Port D (RS-232, header J4) (high)
26–27	PC2–PC3	Serial Port C (RS-232, header J4) (high)
28–29	PC4–PC5	Serial Port B (used by serial flash on RCM4400W)
30–31	PC6–PC7	Serial Port A (programming port) (high)
32–36	PE0–PE4	Output, high
37–38	PE5–PE6	Not available
39	PE7	Output, pulled high
48–49	N/A	Not available
50	GND	

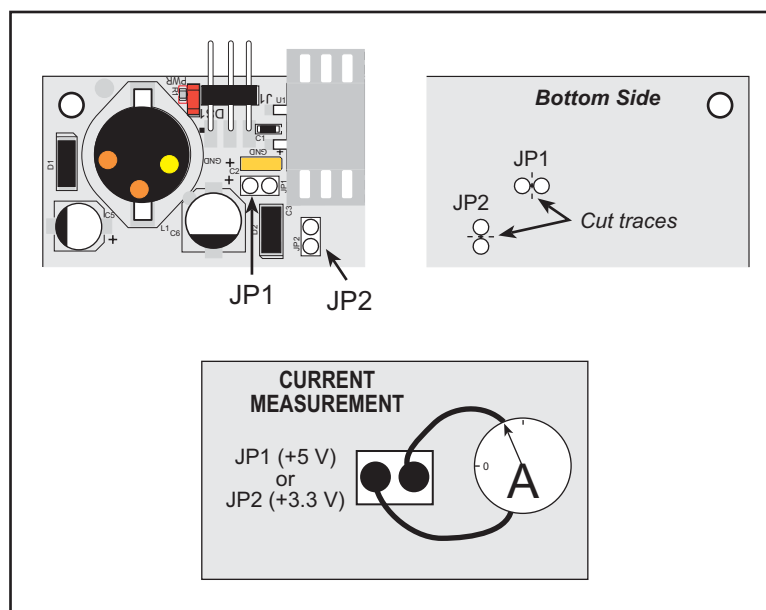
There is a 1.3" × 2" through-hole prototyping space available on the Prototyping Board. The holes in the prototyping area are spaced at 0.1" (2.5 mm). +3.3 V, +5 V, and GND traces run along the top edge of the prototyping area for easy access. Small to medium circuits can be prototyped using point-to-point 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 pads for 28-pin TSSOP devices, 16-pin SOIC devices, and 6-pin SOT devices that can be used for surface-mount prototyping with these 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 Measuring Current Draw

The Prototyping Board has a current-measurement feature available at header locations JP1 and JP2 for the +5 V and +3.3 V supplies respectively. To measure current, you will have to cut the trace on the bottom side of the Prototyping Board corresponding to the power supply or power supplies whose current draw you will be measuring. Header locations JP1 and JP2 are shown in Figure B-5. Then install a 1 × 2 header strip from the Development Kit on the top side of the Prototyping Board at the header location(s) whose trace(s) you cut. The header strip(s) will allow you to use an ammeter across their pins to measure the current drawn from that supply. Once you are done measuring the current, place a jumper across the header pins to resume normal operation.



**Figure B-5. Prototyping Board Current-Measurement Option**

**NOTE:** Once you have cut the trace below header location JP1 or JP2, you must either be using the ammeter or have a jumper in place in order for power to be delivered to the Prototyping Board.

### B.4.3 Analog Features

The Prototyping Board has typical support circuitry installed to complement the ADS7870 A/D converter chip, which is available on other RabbitCore modules based on the Rabbit 4000 microprocessor, but is not installed on the RCM4400W. Since the RCM4400W RabbitCore module does not have the ADS7870 A/D converter chip, the Prototyping Board will not provide A/D converter capability with the RCM4400W RabbitCore module.

### B.4.4 Serial Communication

The Prototyping Board allows you to access the serial ports from the RCM4400W module. Table B-3 summarizes the configuration options.

**Table B-3. Prototyping Board Serial Port Configurations**

Serial Port	Header	Default Use	Alternate Use
A	J2	Programming Port	RS-232
B	J2	Serial Flash	RS-232
C	J2, J4	RS-232	—
D	J2, J4	RS-232	—
E	J2	—	—
F	J2	—	—

Serial Ports E and F may be used as serial ports, or the corresponding pins at header location J2 may be used as parallel ports.

#### B.4.4.1 RS-232

RS-232 serial communication on header J4 on both Prototyping Boards is supported by an RS-232 transceiver installed at U3. 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 4000's signals to RS-232 signal levels. Note that the polarity is reversed in an RS-232 circuit so that a +3.3 V output becomes approximately -10 V and 0 V is output as +10 V. The RS-232 transceiver also provides the proper line loading for reliable communication.

RS-232 can be used effectively at the RCM4400W 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 (C or D). 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., PCDR).

**SERA\_RTS\_SHADOW**—Shadow register for the RTS line's parallel port (e.g., PCDRShadow).

**SERA\_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).

**SERA\_CTS\_BIT**—The bit number for the CTS line.

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

```
#define CINBUFSIZE 15    // set size of circular buffers in bytes
#define COUTBUFSIZE 15

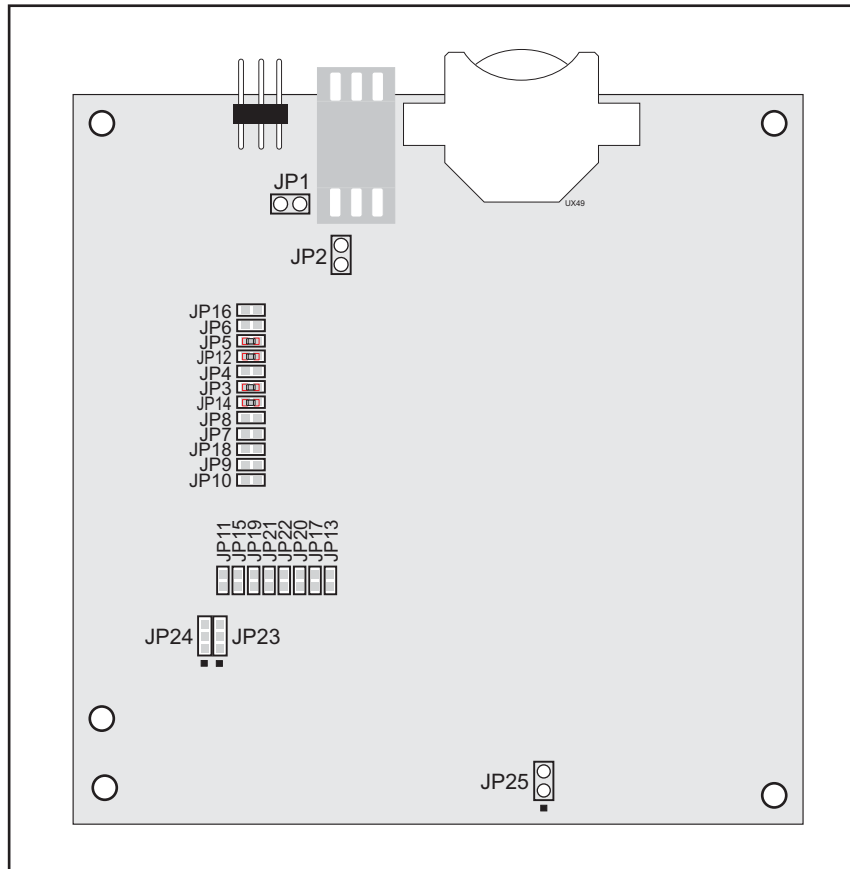
#define DINBUFSIZE 15
#define DOUTBUFSIZE 15

#define MYBAUD 115200    // set baud rate
#endif

main() {
    serCopen( _MYBAUD);    // open Serial Ports C and D
    serDopen( _MYBAUD);
    serCwrFlush();        // flush their input and transmit buffers
    serCrdFlush();
    serDwrFlush();
    serDrdFlush();
    serCclose( _MYBAUD);  // close Serial Ports C and D
    serDclose( _MYBAUD);
}
```

## B.5 Prototyping Board Jumper Configurations

Figure B-6 shows the header locations used to configure the various Prototyping Board options via jumpers.



**Figure B-6. Location of Configurable Jumpers on Prototyping Board**

Table B-4 lists the configuration options using either jumpers or 0  $\Omega$  surface-mount resistors.

**Table B-4. RCM4400W Prototyping Board Jumper Configurations**

Header	Description	Pins Connected		Factory Default
JP1	+5 V Current Measurement	1–2	Via trace or jumper	Connected
JP2	+3.3 V Current Measurement	1–2	Via trace or jumper	Connected
JP3 JP4	PC0/TxD/LED DS2	JP3 1–2	TxD on header J4	×
		JP4 1–2	PC0 to LED DS2	
		n.c.	PC0 available on header J2	

**Table B-4. RCM4400W Prototyping Board Jumper Configurations (continued)**

Header	Description	Pins Connected		Factory Default
JP5 JP6	PC1/RxD/Switch S2	JP5 1–2	RxD on header J4	×
		JP6 1–2	PC1 to Switch S2	
		n.c.	PC1 available on header J2	
JP7 JP8	PC2/TxC/LED DS3	JP7 1–2	TxC on header J4	×
		JP6 1–2	PC2 to LED DS3	
		n.c.	PC2 available on header J2	
JP9 JP10	PC3/RxC/Switch S3	JP9 1–2	PC3 to Switch S3	
		JP10 1–2	RxC on header J4	×
		n.c.	PC3 available on header J2	
JP11	LN0 buffer/filter to RCM4400W	1–2		Connected
JP12	PB2/LED DS2	1–2	Connected: PB2 to LED DS2	×
		n.c.	PB2 available on header J2	
JP13	LN1 buffer/filter to RCM4400W	1–2		Connected
JP14	PB3/LED DS3	1–2	Connected: PB3 to LED DS3	×
		n.c.	PB3 available on header J2	
JP15	LN2 buffer/filter to RCM4400W	1–2		Connected
JP16	PB4/Switch S2	1–2	Connected: PB4 to Switch S2	×
		n.c.	PB4 available on header J2	
JP17	LN3 buffer/filter to RCM4400W	1–2		Connected
JP18	PB5/Switch S3	1–2	Connected: PB5 to Switch S3	×
		n.c.	PB5 available on header J2	
JP19	LN4 buffer/filter to RCM4400W	1–2		Connected

**Table B-4. RCM4400W Prototyping Board Jumper Configurations (continued)**

Header	Description	Pins Connected		Factory Default
JP20	LN5 buffer/filter to RCM4400W	1–2		Connected
JP21	LN6 buffer/filter to RCM4400W	1–2		Connected
JP22	LN7 buffer/filter to RCM4400W	1–2		Connected
JP23	LN4_IN–LN6_IN	1–2	Tied to analog ground	×
		2–3	Tied to VREF	
JP24	LN0_IN–LN3_IN	1–2	Tied to analog ground	×
		2–3	Tied to VREF	
JP25	Thermistor Location	1–2		n.c.

**NOTE:** Jumper connections JP3–JP10, JP12, JP14, JP16, JP18, JP23, and JP24 are made using 0  $\Omega$  surface-mounted resistors. Jumper connections JP11, JP13, JP15, JP17, and JP19–JP22 are made using 470  $\Omega$  surface-mounted resistors.

## APPENDIX C. POWER SUPPLY

Appendix C provides information on the current requirements of the RCM4400W, and includes some background on the chip select circuit used in power management.

### C.1 Power Supplies

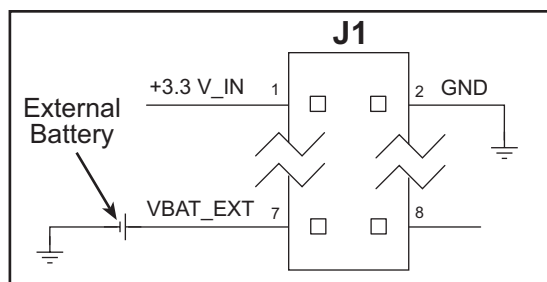
The RCM4400W requires a regulated 3.3 V DC  $\pm 5\%$  power source. The RabbitCore design presumes that the voltage regulator is on the user board, and that the power is made available to the RCM4400W board through header J1.

An RCM4400W with no loading at the outputs operating at 58.98 MHz typically draws 80 mA, and may draw up to 450 mA while the Wi-Fi circuit is transmitting or receiving.

#### C.1.1 Battery-Backup

The RCM4400W 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 4000 real-time clock running.

Header J1, shown in Figure C-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 4000 real-time clock to retain data with the RCM4400W powered down.



**Figure C-1. External Battery Connections at Header J1**

A 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 RCM4400W is typically 7.5  $\mu\text{A}$  when no other power is supplied. If a 165 mA·h battery is used, the battery can last about 2.5 years:

$$\frac{165 \text{ mA}\cdot\text{h}}{7.5 \mu\text{A}} = 2.5 \text{ years.}$$

The actual battery life in your application will depend on the current drawn by components not on the RCM4400W and on the storage capacity of the battery. The RCM4400W does not drain the battery while it is powered up normally.

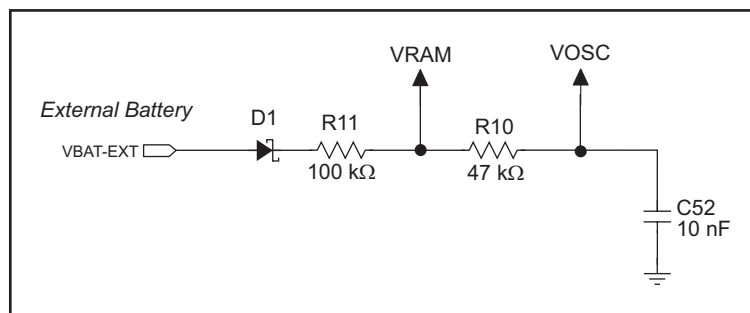
Cycle the main power off/on after you install a backup battery for the first time, and whenever you replace the battery. This step will minimize the current drawn by the real-time clock oscillator circuit from the backup battery should the RCM4400W experience a loss of main power.

**NOTE:** Remember to cycle the main power off/on any time the RCM4400W is removed from the Prototyping Board or motherboard since that is where the backup battery would be located.

Rabbit's Technical Note TN235, *External 32.768 kHz Oscillator Circuits*, provides additional information about the current draw by the real-time clock oscillator circuit.

### C.1.2 Battery-Backup Circuit

Figure C-2 shows the battery-backup circuit.



**Figure C-2. RCM4400W Backup Battery Circuit**

The battery-backup circuit serves three purposes:

- It reduces the battery voltage to the SRAM and to the real-time clock, thereby limiting the current consumed by the real-time clock and lengthening the battery life.
- It ensures that current can flow only *out* of the battery to prevent charging the battery.
- A voltage, VOSC, is supplied to U13, which keeps the 32.768 kHz oscillator working when the voltage begins to drop.

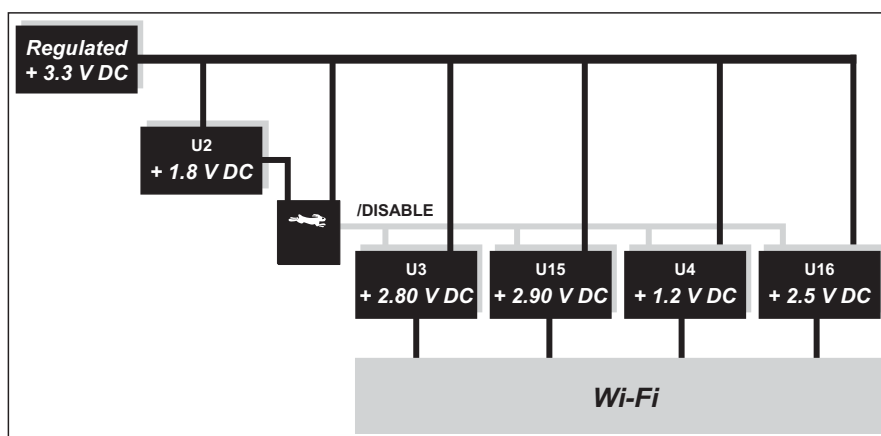
### C.1.3 Reset Generator

The RCM4400W uses a reset generator to reset the Rabbit 4000 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 RCM4400W has a reset output, pin 3 on header J1.

### C.1.4 Onboard Power Supplies

The +3.3 V supplied to the RCM4400W via header J1 powers most of the onboard circuits. In addition, there is a +1.8 V DC linear regulator that provides the core voltage to the Rabbit 4000 microprocessor. Other linear regulators supply the additional voltage levels needed by the Wi-Fi circuits. A /DISABLE line from the Rabbit 4000 can be used to disable the Wi-Fi linear regulators, essentially turning off the Wi-Fi circuits. The **POWERDOWN.C** sample program in the **SAMPLES\RCM4400W\TCPIP\** folder demonstrates this functionality.



**Figure C-3. RCM4400W Onboard Power Supplies**

Voltage	Power Supply Use
+2.90 V DC	VDD_PA
+2.80 V DC	VDD_VCO VDD_XCVR
+2.50 V DC	FPGA VCCAUX
+1.2 V DC	FPGA VCCINT





# INDEX

## A

additional information  
     online documentation ..... 5  
 antenna ..... 88  
     extension ..... 40  
 auxiliary I/O bus ..... 35

## B

battery backup  
     battery life ..... 112  
     circuit ..... 112  
     external battery connections ..... 111  
     real-time clock ..... 112  
     reset generator ..... 113  
     use of battery-backed SRAM ..... 48  
 board initialization  
     function calls ..... 50  
     brdInit() ..... 50  
 bus loading ..... 91

## C

certifications ..... 6  
     Europe ..... 8  
     FCC ..... 6  
     Industry Canada ..... 7  
     Japan ..... 8  
     labeling requirements ..... 7  
 clock doubler ..... 43  
 conformal coating ..... 94

## D

Development Kit ..... 4  
     AC adapter ..... 4  
     Getting Started instructions ..... 4  
     programming cable ..... 4

digital I/O ..... 28  
     function calls ..... 47  
     digInAlert() ..... 51  
     timedAlert() ..... 51  
 I/O buffer sourcing and sinking limits ..... 91  
 memory interface ..... 35  
 SMODE0 ..... 40  
 SMODE1 ..... 40  
 dimensions  
     Prototyping Board ..... 101  
     RCM4400W ..... 84  
 Dynamic C ..... 5, 9, 15, 45  
     add-on modules ..... 9, 52  
     installation ..... 9  
 battery-backed SRAM ..... 48  
 libraries  
     RCM44xxW.LIB ..... 50  
 protected variables ..... 48  
 regulatory compliance ..... 5  
 sample programs ..... 20  
 standard features  
     debugging ..... 46  
     telephone-based technical support ..... 5, 52  
     upgrades and patches ..... 52

## E

exclusion zone ..... 85

## F

features ..... 2  
     Prototyping Boards ..... 98, 99  
 flash memory addresses  
     user blocks ..... 44

## H

hardware connections  
     install RCM4400W on Prototyping Board ..... 12  
     power supply ..... 14  
     programming cable ..... 13

## I

I/O buffer sourcing and sinking limits ..... 91

## J

jumper configurations  
     Prototyping Board ..... 108  
     JP1 (+5 V current measurement) ..... 108  
     JP11 (LN0 buffer/filter to RCM4400W) ..... 109  
     JP12 (PB2/LED DS2) ..... 109  
     JP13 (LN1 buffer/filter to RCM4400W) ..... 109  
     JP14 (PB3/LED DS3) ..... 109  
     JP15 (LN2 buffer/filter to RCM4400W) ..... 109  
     JP16 (PB4/Switch S2) ..... 109  
     JP17 (LN3 buffer/filter to RCM4400W) ..... 109  
     JP18 (PB5/Switch S2) ..... 109  
     JP19 (LN4 buffer/filter to RCM4400W) ..... 109  
     JP2 (+ 3.3 V current measurement) ..... 108  
     JP20 (LN5 buffer/filter to RCM4400W) ..... 110  
     JP21 (LN6 buffer/filter to RCM4400W) ..... 110  
     JP22 (LN7 buffer/filter to RCM4400W) ..... 110  
     JP23 (analog inputs LN4–LN6 configuration) .. 110  
     JP24 (analog inputs LN0–LN3 configuration) .. 110  
     JP3–JP4 (PC0/TxD/LED DS2) ..... 108  
     JP5–JP6 (PC1/RxD/Switch S2) ..... 109  
     JP7–JP8 (PC2/TxC/LED DS3) ..... 109  
     JP9–JP10 (PC3/RxC/Switch S3) ..... 109

jumper configurations (cont'd)	
RCM4400W .....	95
JP1 (FPGA chip select, PE6, or SMODE1 output on J1)	95
JP2 (FPGA interrupt output, PE5, or SMODE0 output on J2) .....	95
JP3 (PE7 or STATUS output on J1) .....	95
JP4 .....	95
jumper locations .....	95

## L

labeling requirements .....	7
LEDs	
Wi-Fi association and activity .....	40

## O

onchip-encryption RAM	
how to use .....	21
operating region configura- tion .....	61

## P

pinout	
Prototyping Board .....	103
RCM4400W	
alternate configurations ..	30
header .....	28
power supplies	
+3.3 V .....	111
battery backup .....	111
battery-backup circuit .....	112
Program Mode .....	41
switching modes .....	41
programming cable	
PROG connector .....	41
RCM4400W connections ..	13
programming port .....	40
Prototyping Board .....	98
access to analog inputs ....	100
adding components .....	105
dimensions .....	101
expansion area .....	99
features .....	98, 99
jumper configurations ....	108
jumper locations .....	108
mounting RCM4400W .....	12
pinout .....	103
power supply .....	102
prototyping area .....	104
specifications .....	102
use of Rabbit 4000 signals	104

## R

Rabbit 4000	
spectrum spreader time delays .....	93
Rabbit subsystems .....	29
RCM4400W	
"development use only"	
version .....	5
mounting on Prototyping Board .....	12
real-time clock	
battery backup .....	112
RP-SMA connector .....	40
Run Mode .....	41
switching modes .....	41

## S

sample programs .....	20
getting to know the	
RCM4400W	
CONTROLLED.C .....	20
FLASHLED1.C .....	20
FLASHLED2.C .....	20
TAMPERDETECTION.C .....	21
TOGGLESWITCH.C ....	21
onboard serial flash	
SERIAL_FLASHLOG.C ..	25
SFLASH_INSPECT.C ..	25
PC/notebook configuration	59
real-time clock	
RTC_TEST.C .....	25
SETRTCKB.C .....	25
serial communication	
FLOWCONTROL.C ....	22
IOCONFIG_ SWITCHECHO.C .....	24
PARITY.C .....	22
SERDMA.C .....	22
SIMPLE3WIRE.C .....	23
SIMPLE5WIRE.C .....	23
SWITCHCHAR.C .....	23
TCP_CONFIG.LIB .....	58
USERBLOCK_CLEAR.C ..	48
USERBLOCK_INFO.C ....	47
Wi-Fi	
BROWSELED.C .....	65
PINGLED.C .....	65
PINGLED_STATS.C ....	65
PINGLED_WPA_PSK.C .....	65
POWERDOWN.C .....	66
SMTP.C .....	66

WIFI_SCAN.C .....	61, 63
WIFI_SCANASSOCI- ATE.C .....	64
WIFIPINGYOU.C .....	63
Wi-Fi configuration macros .....	58
Wi-Fi network configura- tion .....	58
Wi-Fi regulatory setup	
operating region configura- tion .....	61
REGION_COMPILE- TIME.C .....	61
REGION_MULTI_ DOMAIN.C .....	62
REGION_RUNTIME_ PING.C .....	62
serial communication .....	36
function calls .....	47
Prototyping Board	
RS-232 .....	106
software	
PACKET.LIB .....	47
RS232.LIB .....	47
serial flash	
software	
FAT_CONFIG.LIB .....	48
SFLASH.LIB .....	48
SFLASH_FAT.LIB .....	48
serial ports .....	36
programming port .....	40
receive line not pulled up ..	37
Serial Port B (serial flash) ..	36
Serial Port E	
configuration informa- tion .....	24, 36
Serial Port F	
configuration informa- tion .....	24, 36
software .....	5
auxiliary I/O bus .....	35, 47
I/O drivers .....	47
libraries	
TCP_CONFIG.LIB .....	67
regulatory compliance .....	5
serial communication drivers .....	47
serial flash .....	48
Wi-Fi configuration at	
compile time .....	67
configuration macros .....	67
access point SSID .....	67
authentication .....	69
enable/disable WEP encryption .....	68

software	
Wi-Fi configuration at	
compile time	
configuration macros	
(continued)	
encryption keys .....	68
fragmentation threshold	
.....	70
mode .....	67
other macros .....	70
region/country .....	68
RTS threshold .....	70
select encryption key ..	68
set WPA hex key .....	69
set WPA passphrase ..	69
WPA encryption .....	69
your own channel .....	68
network configuration ...	67
TCPCONFIG macro .....	67
Wi-Fi configuration at run	
time .....	71
Wi-Fi drivers .....	48
specifications	
bus loading .....	91
digital I/O buffer sourcing and	
sinking limits .....	91
exclusion zone .....	85
header footprint .....	89
Prototyping Board .....	102
Rabbit 4000 DC characteris-	
tics .....	90
Rabbit 4000 timing diagram	
.....	92
RCM4400W .....	83
antenna .....	88
dimensions .....	84
electrical, mechanical, and	
environmental .....	86
relative pin 1 locations .....	89
spectrum spreader .....	93
settings .....	43
subsystems	
digital inputs and outputs ..	28
switching modes .....	41

## T

technical support .....	17
-------------------------	----

## U

user block	
determining size .....	47
function calls .....	47
readUserBlock() .....	44
writeUserBlock() .....	44
reserved area for calibration	
constants .....	47

## W

Wi-Fi	
additional resources .....	82
bring interface down .....	81
bring interface up .....	81
circuit description .....	38
function calls	
ifconfig(IF_WIFIO,...) ..	81
ifdown(IF_WIFIO) .....	81
ifup(IF_WIFIO) .....	81
sock_init() .....	81
sock_init_or_exit(1) .....	81
tcp_tick(NULL) .....	81
wifi_ioctl commands .....	71
WIFI_AUTH .....	76
WIFI_COUNTRY_GET	
.....	75
WIFI_COUNTRY_SET	
.....	74
WIFI_FRAG_THRESH	
.....	77
WIFI_MODE .....	75
WIFI_MULTI_	
DOMAIN .....	73
WIFI_OWNCHAN ...	75
WIFI_OWNSSID .....	75
WIFI_RTS_THRESH	77
WIFI_SCAN .....	79
WIFI_SCANCB .....	78
WIFI_SSID .....	73
WIFI_STATUSGET ..	79
WIFI_TX_POWER ...	77
WIFI_TX_RATE .....	77
WIFI_WEP_FLAG ...	75
WIFI_WEP_KEY0-3	76
WIFI_WEP_USEKEY	
.....	75
WIFI_WPA_PSK_HEX	
.....	76
WIFI_WPA_PSK_	
PASSPHRASE .....	76
wifi_ioctl() .....	67, 71
sample programs .....	61
software libraries .....	48
software libraries .....	46





# SCHEMATICS

## **090-0239 RCM4400W Schematic**

[www.rabbit.com/documentation/schemat/090-0239.pdf](http://www.rabbit.com/documentation/schemat/090-0239.pdf)

## **090-0230 Prototyping Board Schematic**

[www.rabbit.com/documentation/schemat/090-0230.pdf](http://www.rabbit.com/documentation/schemat/090-0230.pdf)

## **090-0128 Programming Cable Schematic**

[www.rabbit.com/documentation/schemat/090-0128.pdf](http://www.rabbit.com/documentation/schemat/090-0128.pdf)

## **090-0252 USB Programming Cable Schematic**

[www.rabbit.com/documentation/schemat/090-0252.pdf](http://www.rabbit.com/documentation/schemat/090-0252.pdf)

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



Компания «Океан Электроники» предлагает заключение долгосрочных отношений при поставках импортных электронных компонентов на взаимовыгодных условиях!

Наши преимущества:

- Поставка оригинальных импортных электронных компонентов напрямую с производств Америки, Европы и Азии, а так же с крупнейших складов мира;
- Широкая линейка поставок активных и пассивных импортных электронных компонентов (более 30 млн. наименований);
- Поставка сложных, дефицитных, либо снятых с производства позиций;
- Оперативные сроки поставки под заказ (от 5 рабочих дней);
- Экспресс доставка в любую точку России;
- Помощь Конструкторского Отдела и консультации квалифицированных инженеров;
- Техническая поддержка проекта, помощь в подборе аналогов, поставка прототипов;
- Поставка электронных компонентов под контролем ВП;
- Система менеджмента качества сертифицирована по Международному стандарту ISO 9001;
- При необходимости вся продукция военного и аэрокосмического назначения проходит испытания и сертификацию в лаборатории (по согласованию с заказчиком);
- Поставка специализированных компонентов военного и аэрокосмического уровня качества (Xilinx, Altera, Analog Devices, Intersil, Interpoint, Microsemi, Actel, Aeroflex, Peregrine, VPT, Syfer, Eurofarad, Texas Instruments, MS Kennedy, Miteq, Cobham, E2V, MA-COM, Hittite, Mini-Circuits, General Dynamics и др.);

Компания «Океан Электроники» является официальным дистрибьютором и эксклюзивным представителем в России одного из крупнейших производителей разъемов военного и аэрокосмического назначения «JONHON», а так же официальным дистрибьютором и эксклюзивным представителем в России производителя высокотехнологичных и надежных решений для передачи СВЧ сигналов «FORSTAR».



## JONHON

«JONHON» (основан в 1970 г.)

Разъемы специального, военного и аэрокосмического назначения:

(Применяются в военной, авиационной, аэрокосмической, морской, железнодорожной, горно- и нефтедобывающей отраслях промышленности)

«FORSTAR» (основан в 1998 г.)

ВЧ соединители, коаксиальные кабели,  
кабельные сборки и микроволновые компоненты:

(Применяются в телекоммуникациях гражданского и специального назначения, в средствах связи, РЛС, а так же военной, авиационной и аэрокосмической отраслях промышленности).



Телефон: 8 (812) 309-75-97 (многоканальный)

Факс: 8 (812) 320-03-32

Электронная почта: [ocean@oceanchips.ru](mailto:ocean@oceanchips.ru)

Web: <http://oceanchips.ru/>

Адрес: 198099, г. Санкт-Петербург, ул. Калинина, д. 2, корп. 4, лит. А