PRODUCT MANUAL

RabbitCore RCM3100

C-Programmable Module

User's Manual

019–0115 • 070831–H

RabbitCore RCM3100 User's Manual

Part Number 019-0115 • 070831–H • Printed in U.S.A.

Trademarks

Rabbit and Dynamic C are registered trademarks of Rabbit Semiconductor Inc.

Rabbit 3000 and RabbitCore are trademarks of Rabbit Semiconductor Inc.

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

Rabbit Semiconductor Inc.

www.rabbit.com

TABLE OF CONTENTS

1. INTRODUCTION

The RCM3100 RabbitCore module is designed to be the heart of embedded control systems.

Throughout this manual, the term RCM3100 refers to the complete series of RCM3100 RabbitCore modules unless other production models are referred to specifically.

The RCM3100 has a Rabbit 3000 microprocessor operating at 29.4 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 3000's internal real-time clock and the static RAM. Two 34-pin headers bring out the Rabbit 3000 I/O bus lines, parallel ports, and serial ports.

The RCM3100 receives its +3.3 V power from the customer-supplied motherboard on which it is mounted. The RabbitCore RCM3100 can interface with all kinds of CMOS-compatible digital devices through the motherboard.

1.1 RCM3100 Features

- Small size: 1.65" × 1.85" × 0.55"
  (42 mm × 47 mm × 14 mm)
- Microprocessor: Rabbit 3000 running at 29.4 MHz
- 54 parallel 5 V tolerant I/O lines: 46 configurable for I/O, 4 fixed inputs, 4 fixed outputs
- Two additional digital inputs, two additional digital outputs
- External reset input
- 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
- Ten 8-bit timers (six cascadable) and one 10-bit timer with two match registers
- 256K–512K flash memory, 128K–512K SRAM
- Real-time clock
- Watchdog supervisor
- Provision for customer-supplied backup battery via connections on header J2
- 10-bit free-running PWM counter and four pulse-width registers

- Two-channel Input Capture can be used to time input signals from various port pins
- Two-channel Quadrature Decoder accepts inputs from external incremental encoder devices
- Six CMOS-compatible serial ports: maximum asynchronous baud rate of 3.68 Mbps, maximum synchronous baud rate of 7.35 Mbps. Four ports are configurable as a clocked serial port (SPI), and two ports are configurable as SDLC/HDLC serial ports.
- Supports 1.15 Mbps IrDA transceiver

There are two production models in the RCM3100 series. If the standard models do not serve your needs, other variations can be specified and ordered in production quantities. Contact your Rabbit Semiconductor sales representative for details.

Table 1 below highlights the differences between the two models in the RCM3100 family.

Table 1. RCM3100 Versions

Feature	RCM3100	RCM3110
Microprocessor	Rabbit 3000 running at 29.4 MHz	
Flash Memory	2 × 256K	256K
Static RAM	512K	128K
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 clocked serial port is dedicated for programming	

In addition, there is an RCM3000 series of RabbitCore modules that includes Ethernet connectivity.

The RabbitCore modules can be programed locally, remotely, or via a network using appropriate interface hardware.

Appendix A, “RabbitCore RCM3100 Specifications,” provides detailed specifications for the RCM3100.

1.2 Advantages of the RCM3100

- Fast time to market using a fully engineered, “ready to run” microprocessor core.
- Competitive pricing when compared with the alternative of purchasing and assembling individual components.
- Easy C-language program development and debugging
- Utility programs for rapid production loading of programs.
- Generous memory size allows large programs with tens of thousands of lines of code, and substantial data storage.

1.3 Development and Evaluation Tools

A complete Development Kit, including a Prototyping Board and Dynamic C development software, is available for the RCM3100. The Development Kit puts together the essentials you need to design an embedded microprocessor-based system rapidly and efficiently.

1.4 How to Use This Manual

This user’s manual is intended to give users detailed information on the RCM3100 module. It does not contain detailed information on the Dynamic C development environment.

1.4.1 Additional Product Information

Information about the RCM3100 and its associated Development Kit and Prototyping Board can be found in the ***RabbitCore RCM3100 User’s Manual,*** which is provided on the accompanying CD-ROM in both HTML and Adobe PDF format.

In addition to the product-specific information contained in the ***RabbitCore RCM3100 User’s Manual*** (this manual), several higher level reference manuals are provided in HTML and PDF form on the accompanying CD-ROM. Advanced users will find these references valuable in developing systems based on the RCM3100 modules:

- ***Dynamic C User’s Manual***
- ***Dynamic C Function Reference Manual***
- ***Rabbit 3000 Microprocessor User’s Manual***

1.4.2 Online Documentation

The online documentation is installed along with Dynamic C, and an icon for the documentation menu is placed on the workstation’s desktop. Double-click this icon to reach the menu. If the icon is missing, use your browser to find and load **default.htm** in the **docs** folder, found in the Dynamic C installation folder.

The latest versions of all documents are always available for free, unregistered download from our Web sites as well.

2. HARDWARE SETUP

This chapter describes the RCM3100 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 RCM3100 Development Kit. If you purchased an RCM3100 module by itself, you will have to adapt the information in this chapter and elsewhere to your test and development setup.

2.1 Development Kit Contents

The RCM3100 Development Kit contains the following items:

- RCM3100 module, 512K flash memory, and 512K SRAM.
- RCM30/31/32XX Prototyping Board.
- AC adapter, 12 V DC, 1 A. (Included only with Development Kits sold for the North American market. A header plug leading to bare leads is provided to allow overseas users to connect a power supply compatible with their local mains power.)
- 10-pin header to DB9 programming cable with integrated level-matching circuitry.
- ***Dynamic C*** CD-ROM, with complete product documentation on disk.
- ***Getting Started*** instructions.
- A bag of accessory parts for use on the Prototyping Board.
- ***Rabbit 3000 Processor Easy Reference*** poster.
- Registration card.

2.2 Hardware Connections

2.2.1 Attach Module to Prototyping Board

Turn the RCM3100 module so that the mounting holes on the RCM3100 and on the Prototyping Board line up, as shown in Figure 1 below. Align the pins from headers J1 and J2 on the bottom side of the module into header sockets RCM2JA and RCM2JB on the Prototyping Board (these sockets were labeled J12 and J13 on earlier versions of the Prototyping Board).

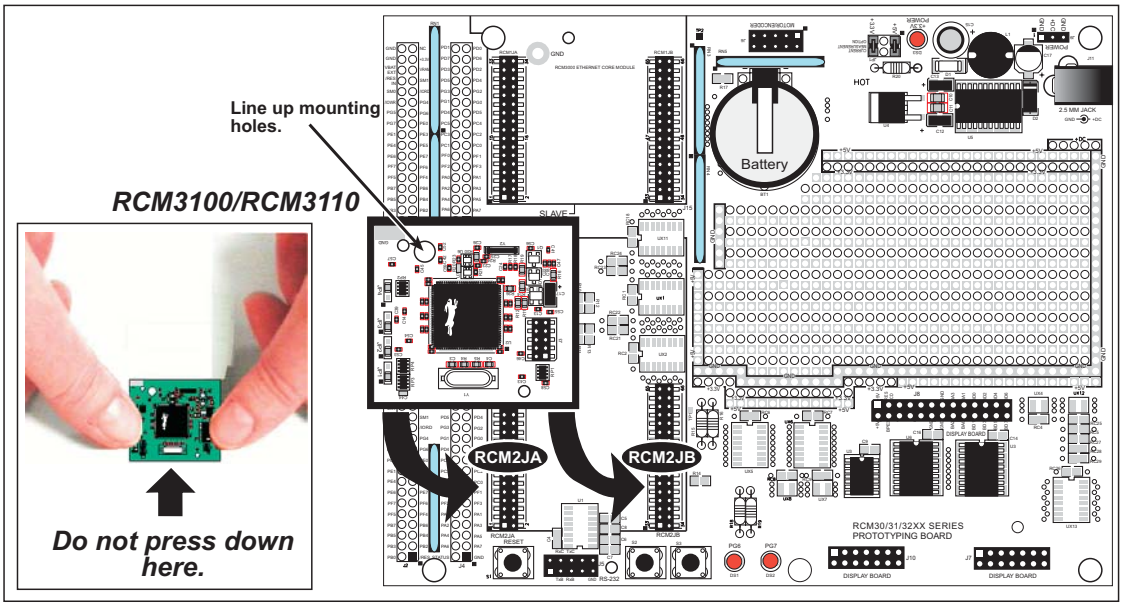


Figure 1. Installing the RCM3100 Module on the Prototyping Board

Although you can install a single module into either the **MASTER** or the **SLAVE** position on the Prototyping Board, all the Prototyping Board features (switches, LEDs, serial port drivers, etc.) are connected to the **MASTER** position — install a single module in the **MASTER** position.

> **NOTE:** It is important that you line up the pins on headers J1 and J2 of the RCM3100 module exactly with the corresponding pins of header sockets RCM2JA and RCM2JB on the Prototyping Board. The header pins may become bent or damaged if the pin alignment is offset, and the module will not work. Permanent electrical damage to the module may also result if a misaligned module is powered up.

Press the module's pins firmly into the Prototyping Board header sockets—press down in the area above the header pins using your thumbs or fingers over the connectors as shown in Figure 1. Do ***not*** press down on the middle of the RCM3100 module to avoid flexing the module, which could damage the module or the components on the module.

Should you need to remove the RCM3100 module, grasp it with your fingers along the sides by the connectors and gently work the module up to pull the pins away from the sockets where they are installed. Do ***not*** remove the module by grasping it at the top and bottom.

2.2.2 Connect Programming Cable

The programming cable connects the RCM3100 module to the PC running Dynamic C to download programs and to monitor the module for debugging.

Connect the 10-pin connector of the programming cable labeled **PROG** to header J3 on the RCM3100 module as shown in Figure 2. 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.)

NOTE: Be sure to use the programming cable (part number 101-0513) supplied with this Development Kit—the programming cable has red shrink wrap around the RS-232 converter section located in the middle of the cable. Programming cables with blue or clear shrink wrap from other from other Rabbit Semiconductor kits are not designed to work with RCM3100 modules.

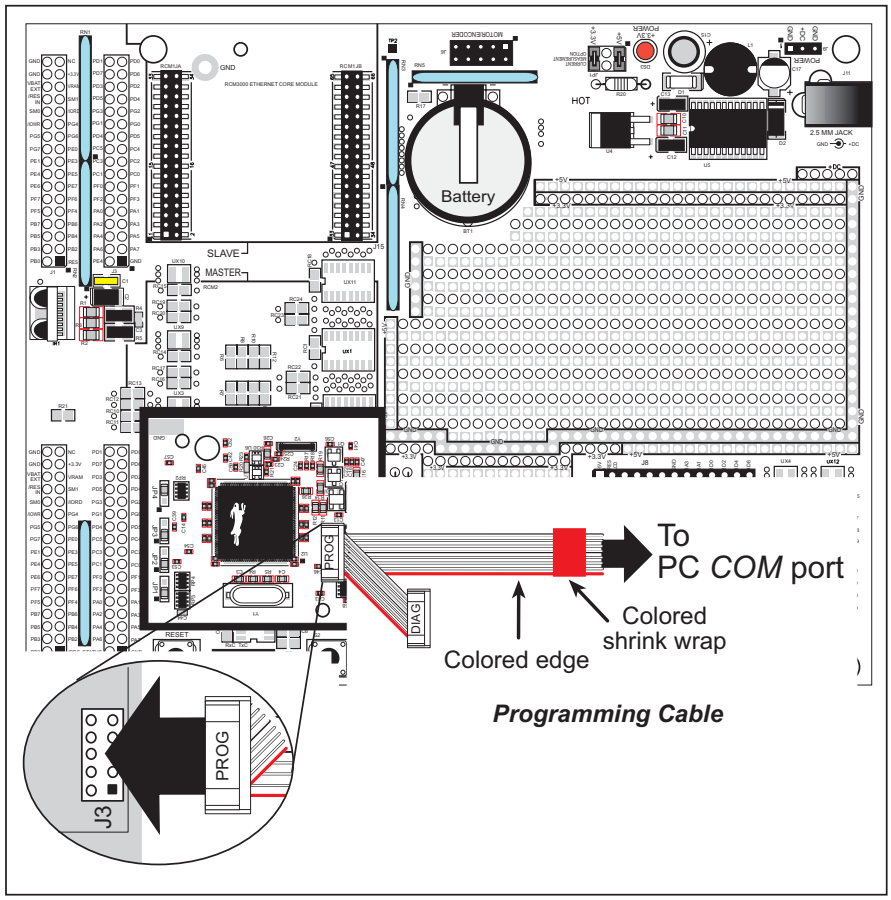


Figure 2. Connect Programming Cable to RCM3100

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

NOTE: Some PCs now come equipped only with a USB port. It may be possible to use an RS-232/USB converter (Part No. 540-0070) with the programming cables mentioned above. Note that not all RS-232/USB converters work with Dynamic C.

2.2.3 Connect Power

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

Connect the wall transformer to jack J11 on the Prototyping Board as shown in Figure 3 below.

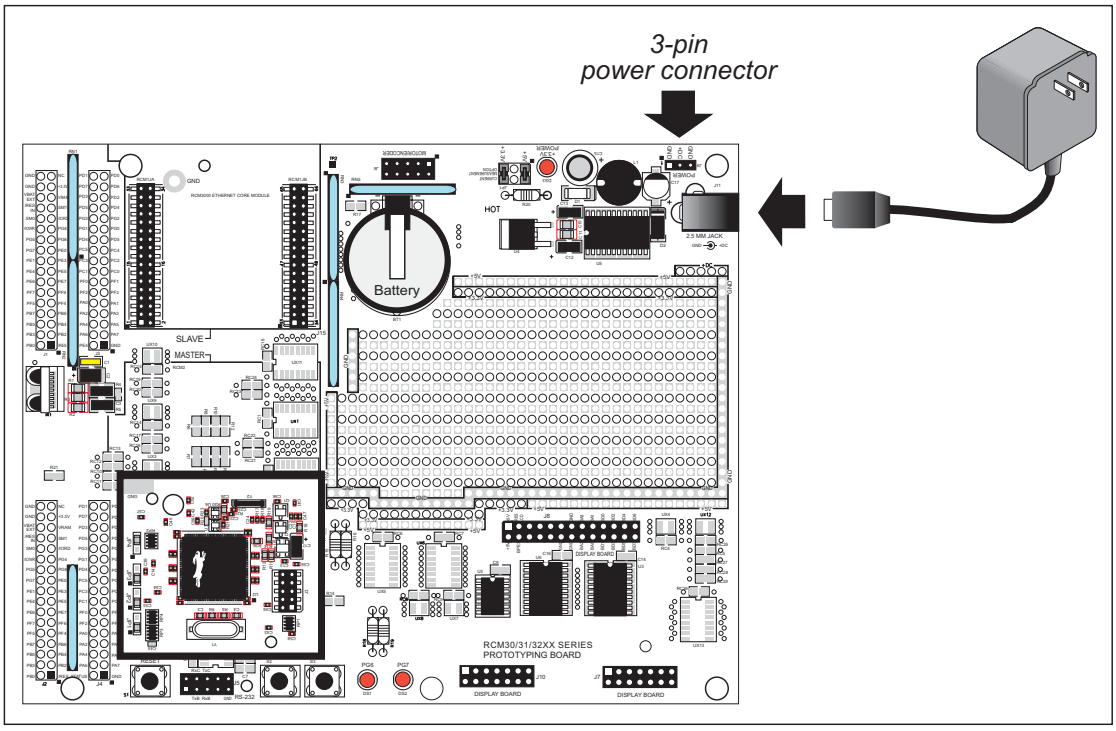


Figure 3. Power Supply Connections

Plug in the wall transformer. The power LED on the Prototyping Board should light up. The RCM3100 and the Prototyping Board are now ready to be used.

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

To power down the Prototyping Board, unplug the power connector from J11. You should disconnect power before making any circuit adjustments in the prototyping area, changing any connections to the board, or removing the RCM3100 from the Prototyping Board.

2.2.3.1 Overseas Development Kits

Development kits sold outside North America include a header connector that may be connected to 3-pin header J9 on the Prototyping Board. The connector may be attached either way as long as it is not offset to one side. The red and black wires from the connector can then be connected to the positive and negative connections on your power supply. The power supply should deliver 8 V–24 V DC at 8 W.

2.3 Starting Dynamic C

Once the RCM3100 is connected as described in the preceding pages, start Dynamic C by double-clicking on the Dynamic C icon or by double-clicking on **dcrab_XXXX.exe** in the Dynamic C root directory, where **XXXX** are version-specific characters. Dynamic C uses the serial port specified during installation.

If you are using a USB port to connect your computer to the RCM3100, choose **Options > Project Options** and select "Use USB to Serial Converter." You may have to determine which COM port was assigned to the RS-232/USB converter.

2.4 Run a Sample Program

Find the file **PONG.C**, which is in the Dynamic C **SAMPLES** folder. To run the program, open it with the **File** menu (if it is not still open), then compile and run it by pressing **F9**. The **STDIO** window will open and will display a small square bouncing around in a box.

This program shows that the CPU is working.

2.4.1 Troubleshooting

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

- Locate the **Serial Options** dialog in the Dynamic C **Options > Project Options > Communications** menu. Select a slower Max download baud rate.

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

- Locate the **Serial Options** dialog in the Dynamic C **Options > Project Options > Communications** menu. Choose a lower debug baud rate.

If there are any other problems:

- Check to make sure you are using the **PROG** connector, not the **DIAG** connector, on the programming cable.
- Check both ends of the programming cable to ensure that they are firmly plugged into the PC and the programming port on the RCM3100.
- Ensure that the RCM3100 module is firmly and correctly installed in its connectors on the Prototyping Board.
- Select a different COM port within Dynamic C. From the **Options** menu, select **Project Options**, then select **Communications**. Select another COM port from the list, then click OK. Press **<Ctrl-Y>** to force Dynamic C to recompile the BIOS. If Dynamic C still reports it is unable to locate the target system, repeat the above steps until you locate the active COM port.

2.5 Where Do I Go From Here?

We recommend that you proceed to the next chapter and install Dynamic C (if you do not already have it installed), then run the **PONG.C** sample program to verify that the RCM3100 module and the Prototyping Board are set up and functioning correctly.

If everything appears to be working, we recommend the following sequence of action:

1. Run all of the sample programs described in Chapter 3 to get a basic familiarity with Dynamic C and the RCM3100 module's capabilities.
2. For further development, refer to the ***RabbitCore RCM3100 User's Manual*** for details of the module's hardware and software components.
3. For advanced development topics, refer to the ***Dynamic C User's Manual***, also in the online documentation set.

2.5.1 Technical Support

NOTE: If you purchased your RCM3100 through a distributor or through a Rabbit Semiconductor partner, contact the distributor or partner first for technical support.

If there are any problems at this point:

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

3. RUNNING SAMPLE PROGRAMS

To develop and debug programs for the RCM3100 (and for all other Rabbit Semiconductor hardware), you must install and use Dynamic C. Chapter 3 walks you through the sample programs associated with the RCM3100.

3.1 Introduction

To help familiarize you with the RCM3100 modules, Dynamic C includes several sample programs. Loading, executing and studying these programs will give you a solid hands-on overview of the RCM3100's capabilities, as well as a quick start with Dynamic C as an application development tool.

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

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

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

1. Your RCM3100 module must be plugged in to the Prototyping Board as described in Chapter 2, "Hardware Setup."
2. Dynamic C must be installed and running on your PC.
3. The RCM3100 module must be connected to your PC through the serial programming cable.
4. Power must be applied to the RCM3100 through the Prototyping Board.

Refer to Chapter 2, "Hardware Setup," if you need further information on these steps.

To run a sample program, open it with the **File** menu, then press function key **F9** to compile and run the program.

3.2 Sample Programs

Of the many sample programs included with Dynamic C, several are specific to the RCM3100. Sample programs illustrating the general operation of the RCM3100, and serial communication are provided in the **SAMPLES\RCM3100** folder. Each sample program has comments that describe the purpose and function of the program. Follow the instructions at the beginning of the sample program.

- **CONTROLLED.C**—uses the **STDIO** window to demonstrate digital outputs by toggling LEDs DS1 and DS2 on the Prototyping Board on and off.

  Parallel Port G bit 6 = LED DS1
  Parallel Port G bit 7 = LED DS2

  Once you have compiled and run this program, you will be prompted via the Dynamic C **STDIO** window to select LED DS1 or DS2. Use your PC keyboard to make your selection.

  Once you have selected the LED, you will be prompted to select to turn the LED either ON or OFF. A logic low will light up the LED you selected.

- **FLASHLED1.C**—demonstrates the use of costatements to flash LEDs DS1 and DS2 on the Prototyping Board at different rates. Once you have compiled and run this program, LEDs DS1 and DS2 will flash on/off at different rates.

- **FLASHLED2.C**—demonstrates the use of cofunctions and costatements to flash LEDs DS1 and DS2 on the Prototyping Board at different rates. Once you have compiled and run this program, LEDs DS1 and DS2 will flash on/off at different rates.

- **TOGGLESWITCH.C**—demonstrates the use of costatements to detect switches using the press-and-release method of debouncing. LEDs DS1 and DS2 on the Prototyping Board are turned on and off when you press switches S2 and S3.

- **IR_DEMO.C**—Demonstrates sending Modbus ASCII packets between two Prototyping Board assemblies via the IrDA transceivers with the IrDA transceivers facing each other. Note that this sample program will only work with the RCM30/31/32XX Prototyping Board.

  First, compile and run this program on one Prototyping Board assembly, then remove the programming cable and press the **RESET** button on the Prototyping Board so that the first RabbitCore module is operating in the **Run** mode. Then connect the programming cable to the second Prototyping Board assembly with the RCM3100 and compile and run the same sample program. With the programming cable still connected to the second Prototyping Board assembly, press switch S2 on the second Prototyping Board to transmit a packet. Once the first Prototyping Board assembly receives a test packet, it will send back a response packet that will be displayed in the Dynamic C **STDIO** window. The test packets and response packets have different codes.

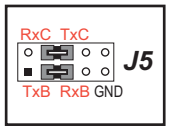
Once you have loaded and executed these sample programs and have an understanding of how Dynamic C and the RCM3100 modules interact, you can move on and try the other sample programs, or begin building your own.

3.2.1 Serial Communication

The following sample programs can be found in the **SAMPLES\RCM3100\SERIAL** folder.

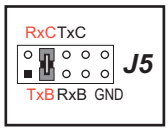
- **FLOWCONTROL.C**—This program demonstrates hardware flow control by configuring Serial Port C (PC3/PC2) for CTS/RTS with serial data coming from TxB at 115,200 bps. One character at a time is received and is displayed in the **STDIO** window.

  To set up the Prototyping Board, you will need to tie TxB and RxB together on the RS-232 header at J5, 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 no flow control.

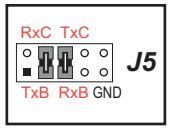
- **PARITY.C**—This program demonstrates the use of parity modes by repeatedly sending byte values 0–127 from Serial Port B to Serial Port C. The program will switch between generating parity or not on Serial Port B. Serial Port C will always be checking parity, so parity errors should occur during every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

  To set up the Prototyping Board, you will need to tie TxB and RxC together on the RS-232 header at J5 using the jumpers supplied in the Development Kit as shown in the diagram.

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

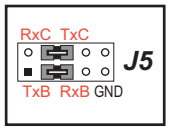
- **SIMPLE3WIRE.C**—This program demonstrates basic RS-232 serial communication. Lower case characters are sent by TxC, and are received by RxB. The characters are converted to upper case and are sent out by TxB, are received by RxC, and are displayed in the Dynamic C **STDIO** window.

  To set up the Prototyping Board, you will need to tie TxB and RxC together on the RS-232 header at J5, and you will also tie RxB 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 C and data flow on Serial Port B.

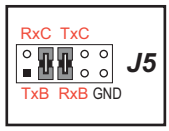
  To set up the Prototyping Board, you will need to tie TxB and RxB together on the RS-232 header at J5, and you will also tie TxC and RxC together using the jumpers supplied in the Development Kit as shown in the diagram.

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

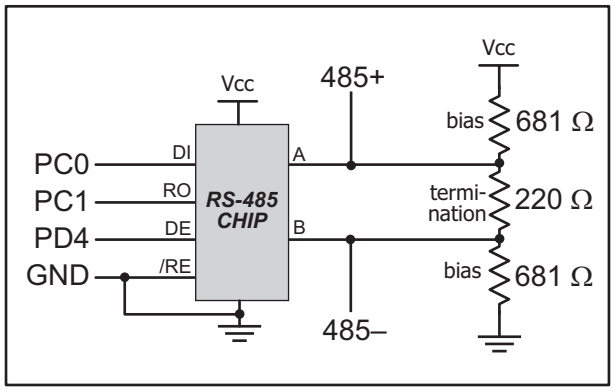
- **SWITCHCHAR.C**—This program demonstrates transmitting and then receiving an ASCII string on Serial Ports B and C. It also displays the serial data received from both ports in the **STDIO** window.

  To set up the Prototyping Board, you will need to tie TxB and RxC together on the RS-232 header at J5, and you will also tie RxB 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 S2 and S3 on the Prototyping Board. The data sent between the serial ports will be displayed in the **STDIO** window.

Two sample programs, **SIMPLE485MASTER.C** and **SIMPLE485SLAVE.C**, are available to illustrate RS-485 master/slave communication. To run these sample programs, you will need a second Rabbit-based system with RS-485, and you will also have to add an RS-485 transceiver such as the SP483E and bias resistors to the RCM30/31/32XX Prototyping Board.



The diagram shows the connections. You will have to connect PC0 and PC1 (Serial Port D) on the RCM30/31/32XX Prototyping Board to the RS-485 transceiver, and you will connect PD4 to the RS-485 transceiver to enable or disable the RS-485 transmitter.

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

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

- **SIMPLE485MASTER.C**—This program demonstrates a simple RS-485 transmission of lower case letters to a slave RCM3100. The slave will send back converted upper case letters to the master RCM3100 and display them in the **STDIO** window. Use **SIMPLE485SLAVE.C** to program the slave RCM3100.

- **SIMPLE485SLAVE.C**—This program demonstrates a simple RS-485 transmission of lower case letters to a master RCM3100. The slave will send back converted upper case letters to the master RCM3100 and display them in the **STDIO** window. Use **SIMPLE485MASTER.C** to program the master RCM3100.

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. Set the real-time clock using the **SETRTCKB.C** sample program from the Dynamic C **SAMPLES\RTCLOCK** folder, using the onscreen prompts. The **RTC_TEST.C** sample program in the Dynamic C **SAMPLES\RTCLOCK** folder provides additional examples of how to read and set the real-time clock.

3.2.3 Other Sample Programs

Appendix C.8 provides sample programs for the optional LCD/keypad module that can be installed on the Prototyping Board.

4. HARDWARE REFERENCE

Chapter 2 describes the hardware components and principal hardware subsystems of the RCM3100. Appendix A, "RabbitCore RCM3100 Specifications," provides complete physical and electrical specifications.

Figure 4 shows these Rabbit-based subsystems designed into the RCM3100.

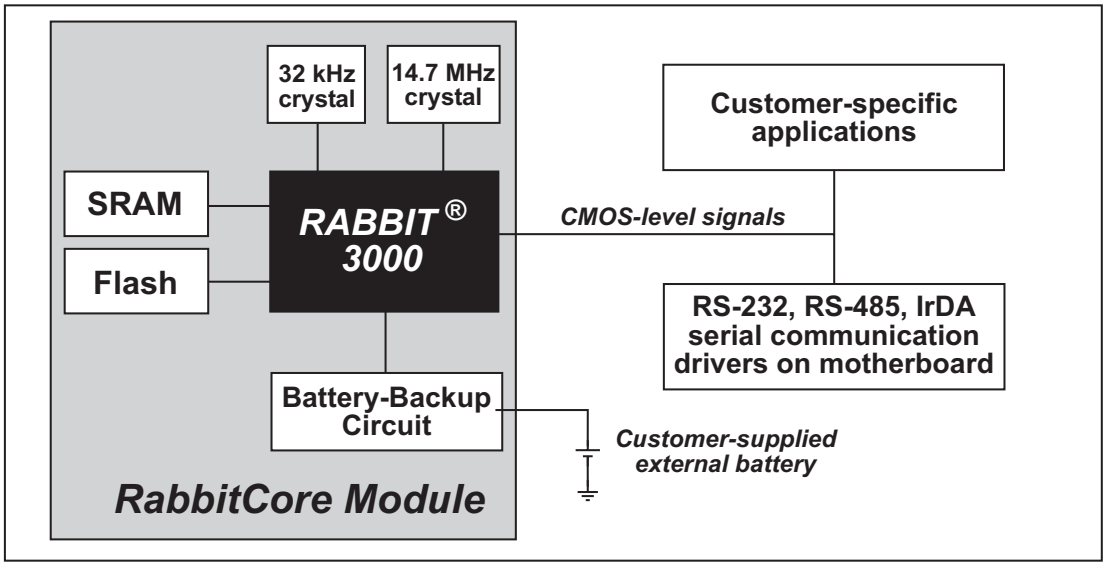


Figure 4. RCM3100 Subsystems

4.1 RCM3100 Digital Inputs and Outputs

The RCM3100 has 54 parallel I/O lines grouped in seven 8-bit ports available on headers J1 and J2. The 46 bidirectional I/O lines are located on pins PA0–PA7, PB0, PB2–PB7, PD0–PD7, PE0–PE1, PE3–PE7, PF0–PF7, and PG0–PG7.

Figure 5 shows the RCM3100 RabbitCore pinouts for headers J1 and J2.

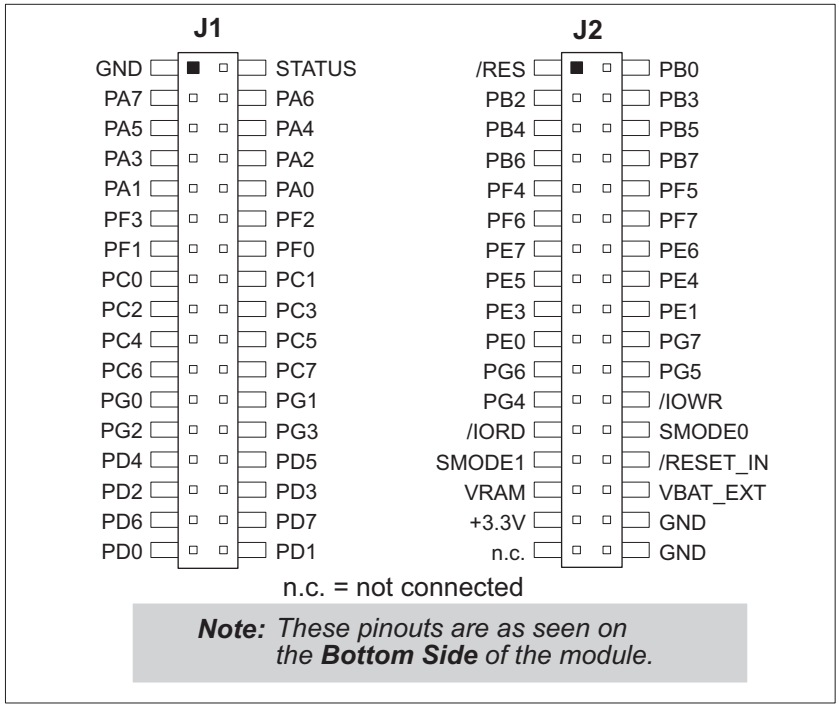


Figure 5. RCM3100 Pinouts

Headers J1 and J2 are standard 2 × 34 headers with a nominal 2 mm pitch.

The signals labeled PD0–PD3, PD6, and PD7 on header J1 (pins 29–34) and the pin that is not connected (pin 33 on header J2) are reserved for future use on other RabbitCore modules.

Figure 6 shows the use of the Rabbit 3000 ports in the RCM3100 RabbitCore modules.

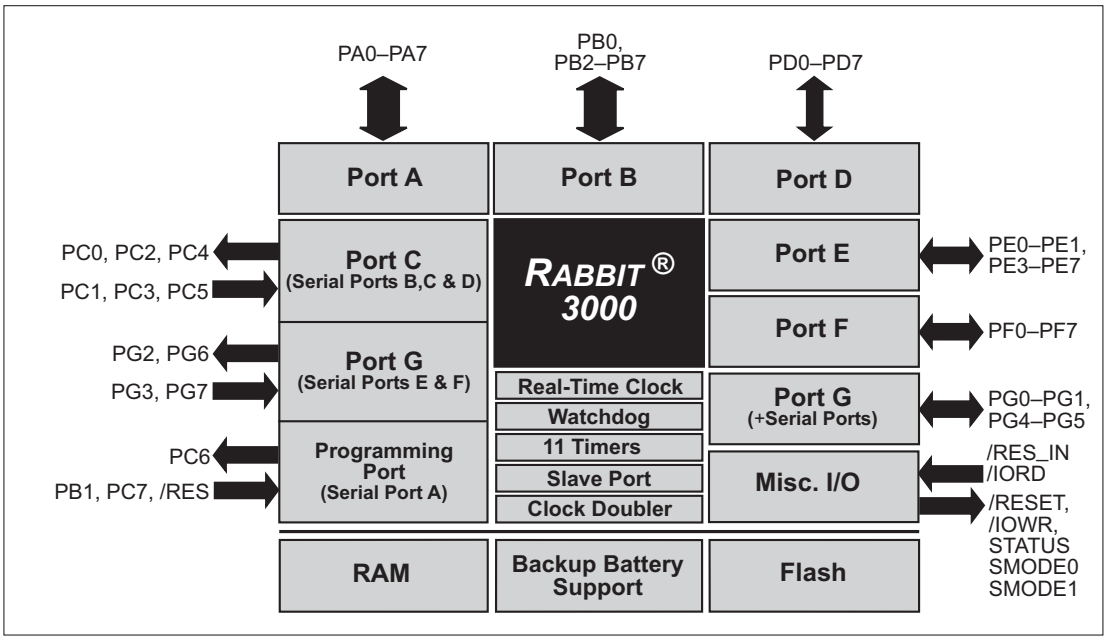


Figure 6. Use of Rabbit 3000 Ports

The ports on the Rabbit 3000 microprocessor used in the RCM3100 are configurable, and so the factory defaults can be reconfigured. Table 2 lists the Rabbit 3000 factory defaults and the alternate configurations.

Table 2. RCM3100 Pinout Configurations

Pin		Pin Name	Default Use	Alternate Use	Notes
Header J1	1	GND			
	2	STATUS	Output (Status)	Output	
	3–10	PA[7:0]	Parallel I/O	External data bus (ID0–ID7) Slave port data bus (SD0–SD7)	
	11	PF3	Input/Output	QD2A	
	12	PF2	Input/Output	QD2B	
	13	PF1	Input/Output	QD1A CLKC	
	14	PF0	Input/Output	QD1B CLKD	
	15	PC0	Output	TXD	Serial Port D
	16	PC1	Input	RXD	
	17	PC2	Output	TXC	Serial Port C
	18	PC3	Input	RXC	
	19	PC4	Output	TXB	Serial Port B
	20	PC5	Input	RXB	
	21	PC6	Output	TXA	Serial Port A (programming port)
	22	PC7	Input	RXA	
	23	PG0	Input/Output	TCLKF	Serial Clock F output
	24	PG1	Input/Output	RCLKF	Serial Clock F input
	25	PG2	Output	TXF	Serial Port F
	26	PG3	Input	RXF	
	27	PD4	Input/Output	ATXB	
	28	PD5	Input/Output	ARXB	
	29*	PD2	Input/Output		
	30*	PD3	Input/Output		
	31*	PD6	Input/Output		
	32*	PD7	Input/Output		
	33*	PD0	Input/Output		
	34*	PD1	Input/Output		

* Pins 29–34 are reserved for future RCM3100 RabbitCore modules.

Table 2. RCM3100 Pinout Configurations (continued)

Pin		Pin Name	Default Use	Alternate Use	Notes
Header J2	1	/RES	Reset output	Reset input	Reset output from Reset Generator
	2	PB0	Input/Output	CLKB	
	3	PB2	Input/Output	IA0 /SWR	External Address 0 Slave port write
	4	PB3	Input/Output	IA1 /SRD	External Address 1 Slave port read
	5	PB4	Input/Output	IA2 SA0	External Address 2 Slave port Address 0
	6	PB5	Input/Output	IA3 SA1	External Address 3 Slave port Address 1
	7	PB6	Input/Output	IA4	External Address 4
	8	PB7	Input/Output	IA5 /SLAVEATTN	External Address 5 Slave Attention
	9	PF4	Input/Output	AQD1B PWM0	
	10	PF5	Input/Output	AQD1A PWM1	
	11	PF6	Input/Output	AQD2B PWM2	
	12	PF7	Input/Output	AQD2A PWM3	
	13	PE7	Input/Output	I7 /SCS	
	14	PE6	Input/Output	I6	
	15	PE5	Input/Output	I5 INT1B	
	16	PE4	Input/Output	I4 INT0B	
	17	PE3	Input/Output	I3	
	18	PE1	Input/Output	I1 INT1A	I/O Strobe 1 Interrupt 1A
	19	PE0	Input/Output	I0 INT0A	I/O Strobe 0 Interrupt 0A

Table 2. RCM3100 Pinout Configurations (continued)

Pin		Pin Name	Default Use	Alternate Use	Notes
Header J2	20	PG7	Input/Output	RXE	Serial Port E
	21	PG6	Input/Output	TXE	
	22	PG5	Input/Output	RCLKE	Serial Clock E input
	23	PG4	Input/Output	TCLKE	Serial Clock E ouput
	24	/IOWR	Output		External write strobe
	25	/IORD	Input		External read strobe
	26–27	SMODE0, SMODE1	(0,0)—start executing at address zero (0,1)—cold boot from slave port (1,0)—cold boot from clocked Serial Port A SMODE0 =1, SMODE1 = 1 Cold boot from asynchronous Serial Port A at 2400 bps (programming cable connected)		Also connected to programming cable
	28	/RESET_IN	Input		Input to Reset Generator
	29	VRAM	Output		Maximum Current Draw 15 µA
	30	VBAT_EXT	3 V battery Input		Minimum battery voltage 2.8 V
	31	+3.3V	Input		3.15–3.45 V DC
	32	GND			
	33	n.c.			
	34	GND			

4.1.1 Memory I/O Interface

The Rabbit 3000 address lines (A0–A19) and all the data lines (D0–D7) are routed internally to the onboard flash memory and SRAM chips. I/0 write (/IOWR) and I/0 read (/IORD) are available for interfacing to external devices.

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

When using the auxiliary I/O bus instead of the default address bus, you must add the following line at the beginning of your program.

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

The STATUS output has three different programmable functions:

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

4.1.2 Other Inputs and Outputs

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

/RESET_IN is an external input used to reset the Rabbit 3000 microprocessor and the RabbitCore RCM3100 memory. /RES is an output from the reset circuitry that can be used to reset other peripheral devices.

4.1.3 5 V Tolerant Inputs

The RCM3100 operates over a voltage from 3.15 V to 3.45 V, but most RCM3100 input pins, except /RESET_IN, VRAM, VBAT_EXT, and the power-supply pins, are 5 V tolerant. When a 5 V signal is applied to 5 V tolerant pins, they present a high impedance even if the Rabbit power is off. The 5 V tolerant feature allows 5 V devices that have a suitable switching threshold to be connected directly to the RCM3100. This includes HCT family parts operated at 5 V that have an input threshold between 0.8 and 2 V.

> **NOTE:** CMOS devices operated at 5 V that have a threshold at 2.5 V are not suitable for direct connection because the Rabbit 3000 outputs do not rise above VDD, and is often specified as 3.3 V. Although a CMOS input with a 2.5 V threshold may switch at 3.3 V, it will consume excessive current and switch slowly.

In order to translate between 5 V and 3.3 V, HCT family parts powered from 5 V can be used, and are often the best solution. There is also the “LVT” family of parts that operate from 2.0 V to 3.3 V, but that have 5 V tolerant inputs and are available from many suppliers. True level-translating parts are available with separate 3.3 V and 5 V supply pins, but these parts are not usually needed, and have design traps involving power sequencing.

4.2 Serial Communication

The RCM3100 board does not have an RS-232 or an RS-485 transceiver directly on the board. However, an RS-232 or RS-485 interface may be incorporated on the board the RCM3100 is mounted on. For example, the Prototyping Board has a standard 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 16. 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 Ports A, B, 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. When the Rabbit 3000 provides the clock, the baud rate can be up to 80% of the system clock frequency divided by 128, or 183,750 bps for a 29.4 MHz clock speed.

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 Port A is available only on the programming port.

4.2.2 Serial Programming Port

The RCM3100 serial programming port is accessed using header J3. The programming port uses the Rabbit 3000's Serial Port A for communication. Dynamic C uses the programming port to download and debug programs.

The programming port is also used for the following operations.

- Cold-boot the Rabbit 3000 on the RCM3100 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.

Programming may also be initiated through the motherboard to which the RCM3100 series module is plugged in to since the Serial Port A (PC6 and PC7), SMODE0, SMODE1, and /RESET_IN are available on headers J1 and J2 (see Table 2).

Alternate Uses of the Serial Programming Port

All three clocked Serial Port A signals are available as

- a synchronous serial port
- an asynchronous serial port, with the clock line usable as a general CMOS input

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

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

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

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

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

The /RESET_IN pin is an external input that is used to reset the Rabbit 3000 and the RCM3100 onboard peripheral circuits. The serial programming port can be used to force a hard reset on the RCM3100 by asserting the /RESET_IN signal.

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

4.3 Serial Programming Cable

The programming cable is used to connect the serial programming port of the RCM3100 to a PC serial COM port. The programming cable converts the RS-232 voltage levels used by the PC serial port to the CMOS voltage levels used by the Rabbit 3000.

When the **PROG** connector on the programming cable is connected to the RCM3100 serial programming port at header J3, programs can be downloaded and debugged over the serial interface.

The **DIAG** connector of the programming cable may be used on header J3 of the RCM3100 with the RCM3100 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 RCM3100 is automatically in Program Mode when the **PROG** connector on the programming cable is attached, and is automatically in Run Mode when no programming cable is attached. When the Rabbit 3000 is reset, the operating mode is determined by the status of the SMODE pins. When the programming cable's **PROG** connector is attached, the SMODE pins are pulled high, placing the Rabbit 3000 in the Program Mode. When the programming cable's **PROG** connector is not attached, the SMODE pins are pulled low, causing the Rabbit 3000 to operate in the Run Mode.

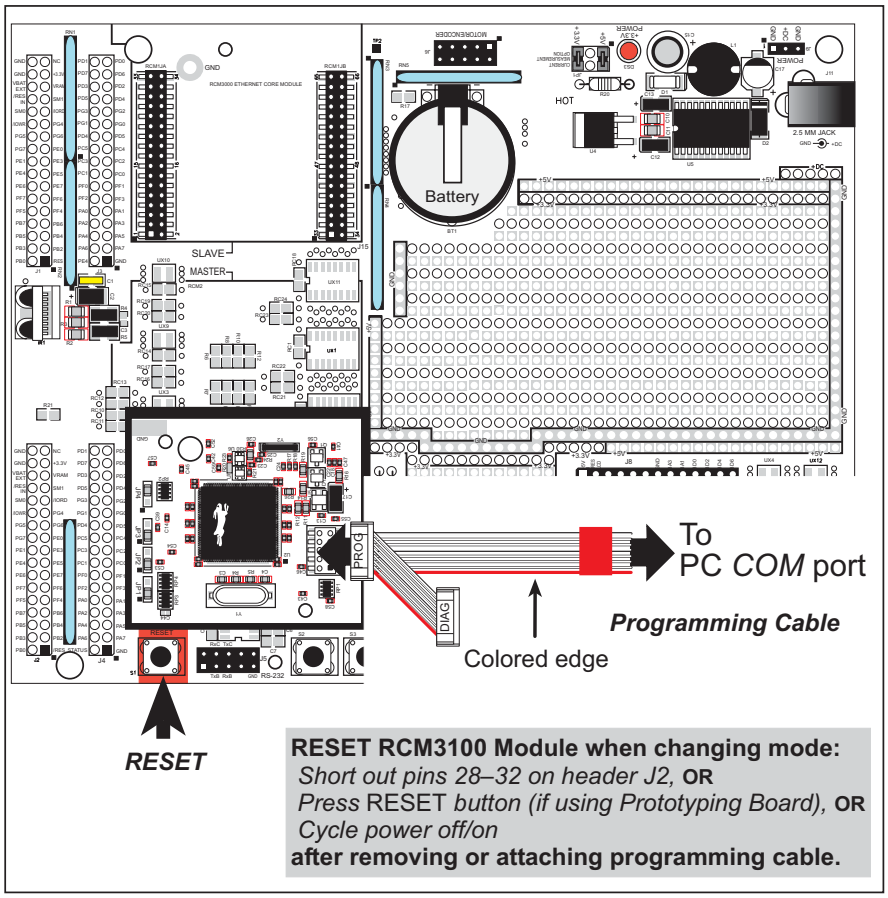


Figure 7. Switching Between Program Mode and Run Mode

A program "runs" in either mode, but can only be downloaded and debugged when the RCM3100 is in the Program Mode.

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

4.3.2 Standalone Operation of the RCM3100

The RCM3100 must be programmed via the Prototyping Board or via a similar arrangement on a customer-supplied board. Once the RCM3100 has been programmed successfully, remove the serial programming cable from the programming connector and reset the RCM3100. The RCM3100 may be reset by cycling the power off/on or by pressing the **RESET** button on the Prototyping Board. The RCM3100 module may now be removed from the Prototyping Board for end-use installation.

CAUTION: Disconnect power to the Prototyping Board or other boards when removing or installing your RCM3100 module to protect against inadvertent shorts across the pins or damage to the RCM3100 if the pins are not plugged in correctly. Do not reapply power until you have verified that the RCM3100 module is plugged in correctly.

4.4 Other Hardware

4.4.1 Clock Doubler

The RCM3100 takes advantage of the Rabbit 3000 microprocessor's internal clock doubler. A built-in clock doubler allows half-frequency crystals to be used to reduce radiated emissions. The 29.4 MHz frequency specified for the RCM3100 is generated using a 14.7456 MHz crystal. The clock doubler will not work for crystals with a frequency above 26.7264 MHz.

The clock doubler may be disabled if 29.4 MHz clock speeds are not required. Disabling the Rabbit 3000 microprocessor's internal clock doubler will reduce power consumption and further reduce radiated emissions. The clock doubler is disabled with a simple configuration macro as shown below.

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

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

4.4.2 Spectrum Spreader

The Rabbit 3000 features a spectrum spreader, which helps to mitigate EMI problems. By default, the spectrum spreader is on automatically, but it may also be turned off or set to a stronger setting. The means for doing so is through a simple configuration macro as shown below.

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

   ```
   ENABLE_SPREADER=1
   ```

   For strong spreading, add the line

   ```
   ENABLE_SPREADER=2
   ```

   To disable the spectrum spreader, add the line

   ```
   ENABLE_SPREADER=0
   ```

   NOTE: The strong spectrum-spreading setting is unnecessary for the BL2000.
3. Click **OK** to save the macro. The spectrum spreader will now remain off whenever you are in the project file where you defined the macro.

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

4.5 Memory

4.5.1 SRAM

The RCM3100 can accept 128K to 512K of SRAM at U4.

4.5.2 Flash EPROM

The RCM3100 can accept 256K to 512K of flash EPROM.

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

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

A Flash Memory Bank Select jumper configuration option based on 0 Ω surface-mounted resistors exists at header JP1 on the RCM3100 RabbitCore modules. This option, used in conjunction with some configuration macros, allows Dynamic C to compile two different co-resident programs for the upper and lower halves of the 256K flash in such a way that both programs start at logical address 0000. This is useful for applications that require a resident download manager and a separate downloaded program. See Technical Note 218, ***Implementing a Serial Download Manager for a 256K Flash***, for details.

4.5.3 Dynamic C BIOS Source Files

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

5. SOFTWARE REFERENCE

Dynamic C is an integrated development system for writing embedded software. It runs on an IBM-compatible PC and is designed for use with Rabbit Semiconductor devices and other devices based on the Rabbit microprocessor. Chapter 4 provides the libraries and function calls related to the RCM3100.

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 RAM included on the RCM3100. The flash memory and SRAM options are selected via the "BIOS Memory Setting" in 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 developed in RAM, but cannot run standalone from RAM after the programming cable is disconnected. All standalone applications can only run from flash memory.

> **NOTE:** Do not depend on the flash memory sector size or type. Due to the volatility of the flash memory market, the RCM3100 and Dynamic C were designed to accommodate flash devices with various sector sizes.

The RCM3100 model has two 256K flash memories. By default, Dynamic C will use only the first flash memory for program code in the RCM3100 model. Uncomment he BIOS **`USE_2NDFLASH_CODE`** macro to allow the second flash memory to hold any program code that is in excess of the available memory in the first flash.

Developing software with Dynamic C is simple. Users can write, compile, and test C and assembly code without leaving the Dynamic C development environment. Debugging occurs while the application runs on the target. Alternatively, users can compile a program to an image file for later loading. Dynamic C runs on PCs under Windows 95 or later. Programs can be downloaded at baud rates of up to 460,800 bps after the program compiles.

Dynamic C has a number of standard features.

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

5.2 Dynamic C Function Calls

5.2.1 I/O

The RCM3100 was designed to interface with other systems, and so there are no drivers written specifically for the I/O. The general Dynamic C read and write functions allow you to customize the parallel I/O to meet your specific needs. For example, use

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

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

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

to set all the Port E bits as outputs.

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

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

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

The sample programs in the Dynamic C **SAMPLES/RCM3100** directory 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. For more information, see the ***Dynamic C User's Manual*** and Technical Note 213, ***Rabbit 2000 Serial Port Software***.

5.2.3 Prototyping Board Functions

The function described in this section is for use with the Prototyping Board. The source code is in the **RCM3100.LIB** library in the Dynamic C **SAMPLES\RCM3100** folder if you need to modify it for your own board design.

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

5.2.3.1 Board Initialization

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

Call this function at the beginning of your program. This function initializes Parallel Ports A through G for use with the RCM3000/31/32XX Prototyping Board.

This function also sets any unused configurable port pins as outputs with a high output, and assumes that only one RCM3100 module is installed in the **MASTER** position on the Prototyping Board.

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/ for the latest patches, workarounds, and bug fixes.

The default installation of a patch or bug fix is to install the file in a directory (folder) different from that of the original Dynamic C installation. Rabbit Semiconductor recommends using a different directory so that you can verify the operation of the patch without overwriting the existing Dynamic C installation. If you have made any changes to the BIOS or to libraries, or if you have programs in the old directory (folder), make these same changes to the BIOS or libraries in the new directory containing the patch. Do ***not*** simply copy over an entire file since you may overwrite a bug fix; of course, you may copy over any programs you have written.

5.3.1 Upgrades

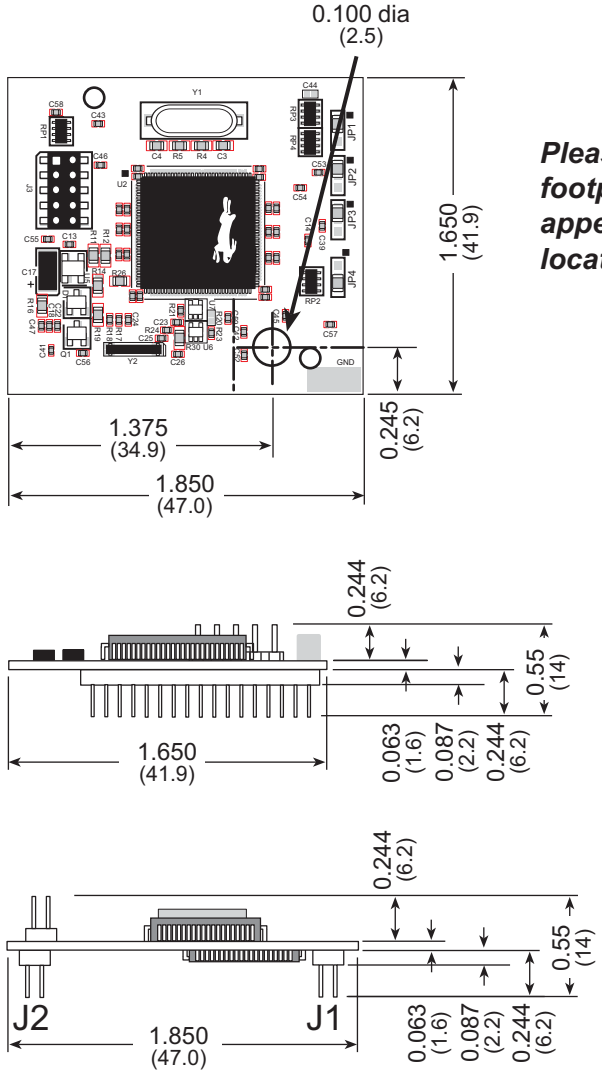
Dynamic C installations are designed for use with the board they are included with, and are included at no charge as part of our low-cost kits. Dynamic C is a complete software development system, but does not include all the Dynamic C features. Rabbit Semiconductor also offers add-on Dynamic C modules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

APPENDIX A. RABBITCORE RCM3100 SPECIFICATIONS

Appendix A provides the specifications for the RCM3100, and describes the conformal coating.

A.1 Electrical and Mechanical Characteristics

Figure A-1 shows the mechanical dimensions for the RCM3100.



Please refer to the RCM3100 footprint diagram later in this appendix for precise header locations.

Figure A-1. RCM3100 Dimensions

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

Table A-1 lists the electrical, mechanical, and environmental specifications for the RCM3100.

Table A-1. RabbitCore RCM3100 Specifications

Feature	RCM3100	RCM3110
Microprocessor	Rabbit 3000® at 29.4 MHz	
EMI Reduction	Spectrum spreader for reduced EMI (radiated emissions)	
Flash Memory	512K (2 × 256K)	256K
SRAM	512K	128K
Backup Battery	Connection for user-supplied backup battery to support RTC and SRAM)	
General-Purpose I/O	54 parallel digital I/0 lines: • 46 configurable I/O, • 4 fixed inputs, • 4 fixed outputs	
Additional Digital Inputs	2 startup mode, reset in	
Additional Digital Outputs	Status, reset out	
Auxiliary I/O Bus	8 data lines and 6 address lines (shared with I/O) plus I/O read/write	
Serial Ports	6 shared high-speed, CMOS-compatible ports: • 6 configurable as asynchronous (with IrDA), 4 as clocked serial (SPI), and 2 as SDLC/HDLC (with IrDA) • 1 asynchronous clocked serial port dedicated for programming • support for MIR/SIR IrDA transceiver	
Serial Rate	Max. asynchronous baud rate = CLK/8	
Slave Interface	A slave port allows the RCM3100 to be used as a master or as an intelligent peripheral device with Rabbit-based or any other type of processor	
Real-Time Clock	Yes	
Timers	Ten 8-bit timers (6 cascadable from the first), one 10-bit timer with 2 match registers	
Watchdog/Supervisor	Yes	
Pulse-Width Modulators	10-bit free-running counter and four pulse-width registers	
Input Capture	2- channel input capture can be used to time input signals from various port pins	
Quadrature Decoder	2-channel quadrature decoder accepts inputs from external incremental encoder modules	
Power	3.15 V to 3.45 V DC 75 mA @ 3.3 V	

Table A-1. RabbitCore RCM3100 Specifications (continued)

Feature	RCM3100	RCM3110
Operating Temperature	–40°C to +85°C	
Humidity	5% to 95%, noncondensing	
Connectors (for connection to headers J4 and J5)	Two 2 ×17, 2 mm pitch	
Board Size	1.850" × 1.650" × 0.55" (47 mm × 42 mm × 14 mm)	

A.1.1 Exclusion Zone

It is recommended that you allow for an “exclusion zone” of 0.04" (1 mm) around the RCM3100 in all directions when the RCM3100 is incorporated into an assembly that includes other printed circuit boards. This “exclusion zone” that you keep free of other components and boards will allow for sufficient air flow, and will help to minimize any electrical or electromagnetic interference between adjacent boards. An “exclusion zone” of 0.08" (2 mm) is recommended below the RCM3100 when the RCM3100 is plugged into another assembly using the shortest connectors for headers J1 and J2. Figure A-2 shows this “exclusion zone.”

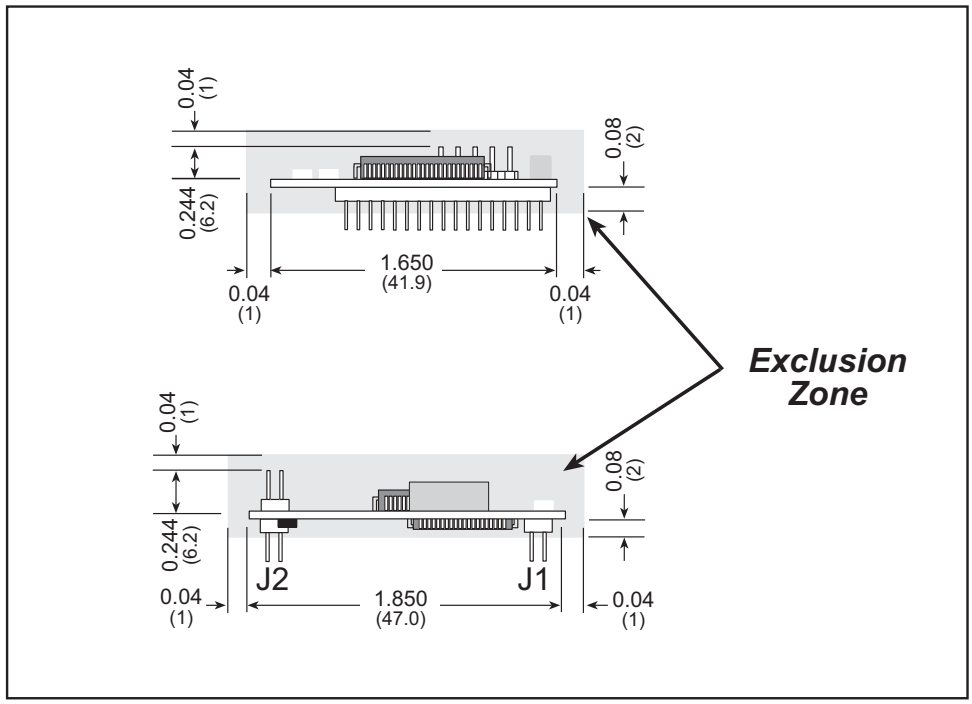


Figure A-2. RCM3100 “Exclusion Zone”

A.1.2 Headers

The RCM3100 uses headers at J1 and J2 for physical connection to other boards. J1 and J2 are 2 × 17 SMT headers with a 2 mm pin spacing. J3, the programming port, is a 2 × 5 header with a 2 mm pin spacing.

Figure A-3 shows the layout of another board for the RCM3100 to be plugged into. These values are relative to the mounting hole.

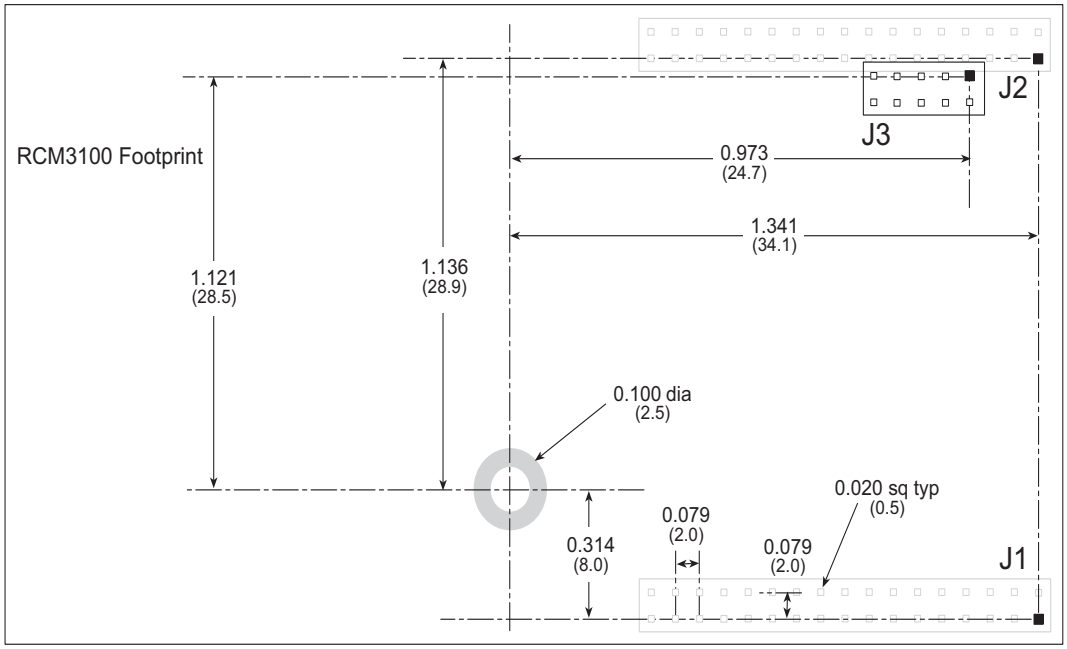


Figure A-3. User Board Footprint for RCM3100

A.1.3 Physical Mounting

A standoff with a 2-56 screw is recommended to attach the RCM3100 to a user board at the hole position shown in Figure A-3.

A.2 Bus Loading

You must pay careful attention to bus loading when designing an interface to the RCM3100. This section provides bus loading information for external devices.

Table A-2 lists the capacitance for the various RCM3100 I/O ports.

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

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

Table A-3 lists the external capacitive bus loading for the various RCM3100 output ports. Be sure to add the loads for the devices you are using in your custom system and verify that they do not exceed the values in Table A-3.

Table A-3. External Capacitive Bus Loading -40°C to +70°C

Output Port	Clock Speed (MHz)	Maximum External Capacitive Loading (pF)
All I/O lines with clock doubler enabled	29.4	30–70
All I/O lines with clock doubler disabled	14.7456	100

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

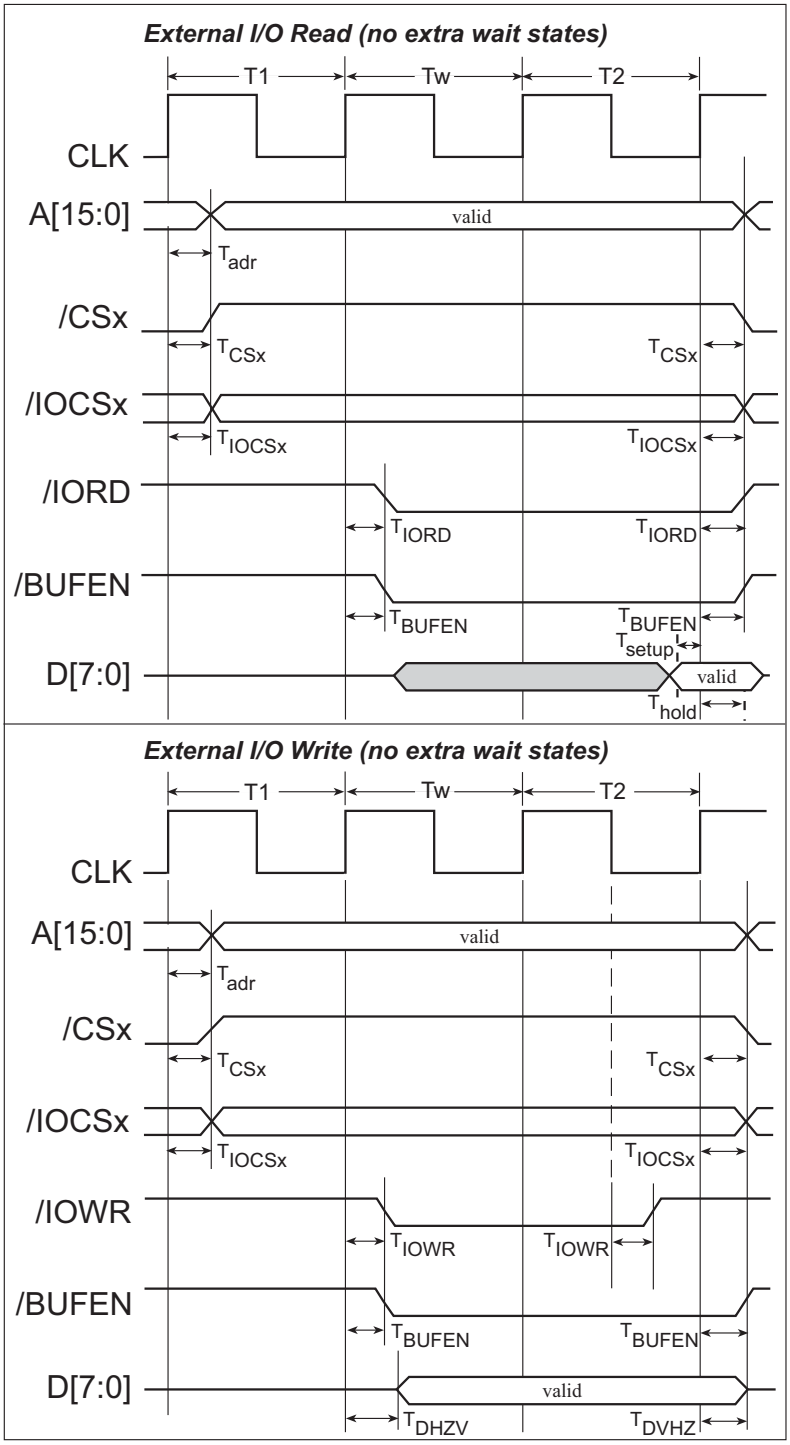


Figure A-4. External I/O Read and Write Cycles—No Extra Wait States

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

Table A-4 lists the delays in gross memory access time for V_{DD} = 3.3 V.

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

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

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

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

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

- T_{adr}, the clock to address delay
- T_{CSx}, the clock to memory chip select delay
- T_{WEx}, the clock to memory write strobe 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.

A.3 Rabbit 3000 DC Characteristics

Table A-5 outlines the DC characteristics for the Rabbit 3000 at 3.3 V over the recommended operating temperature range from T_a = –55°C to +125°C. Note that while the Rabbit 3000 is rated to operate over a voltage range from 3.0–3.6 V, the RCM3100 has a more restrictive operating voltage range of 3.15–3.45 V DC.

Table A-5. 3.3 Volt DC Characteristics

Symbol	Parameter	Test Conditions	Min	Typ	Max	Units
I_{IH}	Input Leakage High	$V_{IN} = V_{DD}$, V_{DD} = 3.3 V			1	µA
I_{IL}	Input Leakage Low (no pull-up)	$V_{IN} = V_{SS}$, V_{DD} = 3.3 V	-1			µA
I_{OZ}	Output Leakage (no pull-up)	$V_{IN} = V_{DD}$ or V_{SS}, V_{DD} = 3.3 V	-1		1	µA
V_{IL}	CMOS Input Low Voltage				$0.3 \times V_{DD}$	V
V_{IH}	CMOS Input High Voltage		$0.7 \times V_{DD}$			V
V_T	CMOS Switching Threshold	V_{DD} = 3.3 V, 25°C		1.65		V
V_{OL}	Low-Level Output Voltage	I_{OL} = 6 mA			0.4	V
V_{OH}	High-Level Output Voltage	I_{OH} =6 mA	$0.7 \times V_{DD}$			V

A.4 I/O Buffer Sourcing and Sinking Limit

Unless otherwise specified, the Rabbit 3000 I/O buffers are capable of sourcing and sinking 6.8 mA of current per pin at full AC switching speed. Full AC switching assumes a 29.4 MHz CPU clock and capacitive loading on address and data lines of less than 70 pF per pin. The maximum Vcc is 3.6 V, and the absolute maximum operating voltage on all parallel I/O is 5.5 V.

Table A-6 shows the AC and DC output drive limits of the parallel I/O buffers when the Rabbit 3000 is used in the RCM3100.

Table A-6. I/O Buffer Sourcing and Sinking Capability

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

Under certain conditions, the maximum instantaneous AC/DC sourcing or sinking current may be greater than the limits outlined in Table A-6. The maximum AC/DC sourcing current can be as high as 12.5 mA per buffer as long as the number of sourcing buffers does not exceed three per V_{DD} or V_{SS} pad, or up to six outputs between pads. Similarly, the maximum AC/DC sinking current can be as high as 8.5 mA per buffer as long as the number of sinking buffers does not exceed three per V_{DD} or V_{SS} pad, or up to six outputs between pads. The V_{DD} bus can handle up to 35 mA, and the V_{SS} bus can handle up to 28 mA. All these analyses were measured at 100°C.

A.5 Conformal Coating

The areas around the 32 kHz real-time clock crystal oscillator has had the Dow Corning silicone-based 1-2620 conformal coating applied. The conformally coated area is shown in Figure A-5. The conformal coating protects these high-impedance circuits from the effects of moisture and contaminants over time.

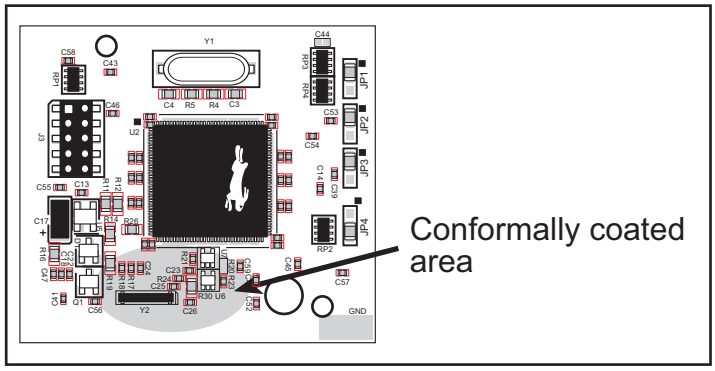


Figure A-5. RCM3100 Areas Receiving Conformal Coating

Any components in the conformally coated area may be replaced using standard soldering procedures for surface-mounted components. A new conformal coating should then be applied to offer continuing protection against the effects of moisture and contaminants.

NOTE: For more information on conformal coatings, refer to Technical Note 303, ***Conformal Coatings***.

A.6 Jumper Configurations

Figure A-6 shows the header locations used to configure the various RCM3100 options via jumpers.

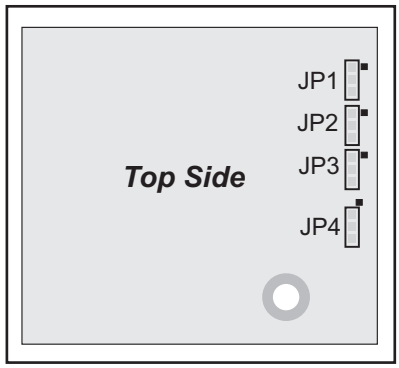


Figure A-6. Location of RCM3100 Configurable Positions

Table A-7 lists the configuration options.

Table A-7. RCM3100 Jumper Configurations

Header	Description	Pins Connected		Factory Default
JP1	Flash Memory Bank Select	1–2	Normal Mode	×
		2–3	Bank Mode	
JP2	Flash Memory Size	1–2	128K/256K	×
		2–3	512K	
JP3	Flash Memory Size	1–2	128K/256K	RCM3100
		2–3	512K	
JP4	SRAM Size	1–2	128K	RCM3110
		2–3	512K	RCM3100

NOTE: The jumper connections are made using 0 Ω 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 RCM3100 and to build prototypes of your own circuits.

B.1 Introduction

The Prototyping Board included in the Development Kit makes it easy to connect an RCM3100 module to a power supply and a PC workstation for development. It also provides some basic I/O peripherals (switches and LEDs), as well as a prototyping area for more advanced hardware development.

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

As you progress to more sophisticated experimentation and hardware development, modifications and additions can be made to the board without modifying or damaging the RCM3100 module itself.

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

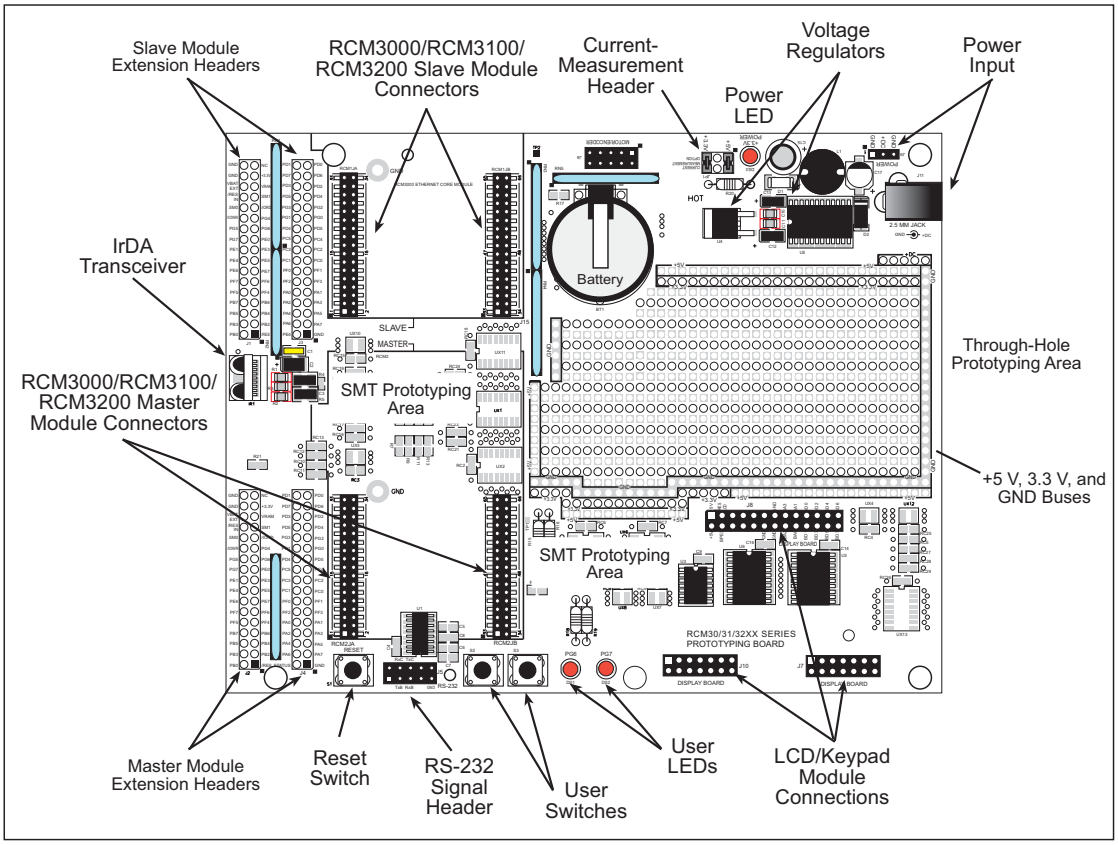


Figure B-1. RCM30/31/32XX Prototyping Board

B.1.1 Prototyping Board Features

- **Power Connection**—A power-supply jack and a 3-pin header are provided for connection to the power supply. Note that the 3-pin header is symmetrical, with both outer pins connected to ground and the center pin connected to the raw V+ input. The cable of the AC adapter provided with the North American version of the Development Kit ends in a plug that connects to the power-supply jack. 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 POWER IN jack is routed to a 5 V switching voltage regulator, then to a separate 3.3 V linear regulator. The regulators provide stable power to the RCM3100 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 RCM3100's **/RESET_IN** pin. Pressing the switch forces a hardware reset of the system.

- **I/O Switches and LEDs**—Two momentary-contact, normally open switches are connected to the PG0 and PG1 pins of the master RCM3100 module and may be read as inputs by sample applications.

  Two LEDs are connected to the PG6 and PG7 pins of the master 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.

- **Master Module Connectors**—A set of connectors is pre-wired to permit installation of the first RCM3000, RCM3100, or RCM3200 module that serves as the primary or "master module."

- **Slave Module Connectors**—A second set of connectors is pre-wired to permit installation of a second, slave RCM3000, RCM3100, or RCM3200 module. This capability is reserved for future use, although the schematics in this manual contain all of the details an experienced developer will need to implement a master-slave system.

- **Module Extension Headers**—The complete pin sets of both the **MASTER** and **SLAVE** RabbitCore modules are duplicated at these two sets of headers. Developers can solder wires directly into the appropriate holes, or, for more flexible development, 26-pin header strips can be soldered into place. See Figure B-4 for the header pinouts.

- **RS-232**—Two 3-wire or one 5-wire RS-232 serial port are available on the Prototyping Board. Refer to the Prototyping Board schematic (090-0137) for additional details.

  A 10-pin 0.1-inch spacing header strip is installed at J5 to permit connection of a ribbon cable leading to a standard DE-9 serial connector.

- **Current Measurement Option**—Jumpers across pins 1–2 and 5–6 on header JP1 can be removed and replaced with an ammeter across the pins to measure the current drawn from the +5 V or the +3.3 V supplies, respectively.

- **Motor Encoder**—A motor/encoder header is provided at header J6 for future use.

- **LCD/Keypad Module**—Rabbit Semiconductor's LCD/keypad module may be plugged in directly to headers J7, J8, and J10.

B.2 Mechanical Dimensions and Layout

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

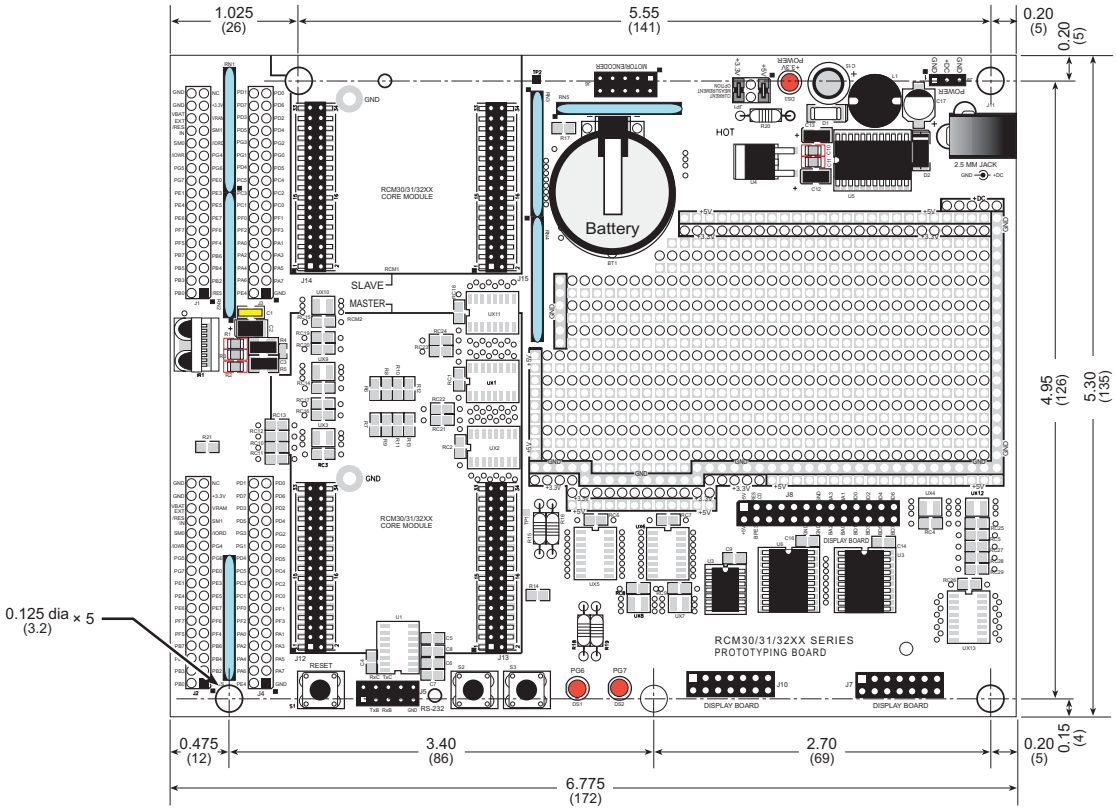


Figure B-2. RCM30/31/32XX Prototyping Board Dimensions

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

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

Table B-1. RCM30/31/32XX Prototyping Board Specifications

Parameter	Specification
Board Size	5.30" × 6.775" × 1.00" (135 mm × 172 mm × 25 mm)
Operating Temperature	–20°C to +60°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	2.0" × 3.5" (50 mm × 90 mm) throughhole, 0.1" spacing, additional space for SMT components
Standoffs/Spacers	5, accept 4-40 × 3/8 screws

B.3 Power Supply

The RCM3100 requires a regulated 3.3 V ± 0.15 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 AC adapter supplied with the RCM3100 Development Kit provides 12 V at up to 1 A as the input to the voltage regulator on the Prototyping Board. 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 Shottky diode at D2 as shown in Figure B-3.

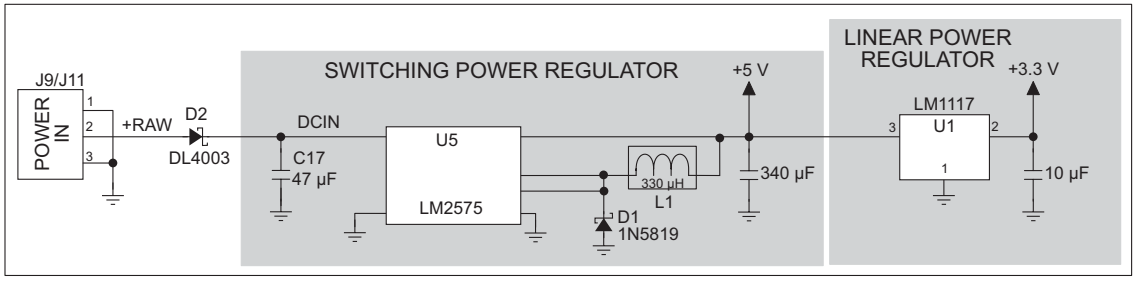


Figure B-3. Prototyping Board Power Supply

B.4 Using the Prototyping Board

The Prototyping Board is actually both a demonstration board and a prototyping board. As a demonstration board, it can be used to demonstrate the functionality of the RCM3100 right out of the box without any modifications to either board. There are no jumpers or dip switches to configure or misconfigure on the Prototyping Board so that the initial setup is very straightforward.

The Prototyping Board comes with the basic components necessary to demonstrate the operation of the RCM3100. Two LEDs (DS1 and DS2) are connected to PG6 and PG7, and two switches (S2 and S3) are connected to PG1 and PG0 to demonstrate the interface to the Rabbit 3000 microprocessor. Reset switch S1 is the hardware reset for the RCM3100.

The Prototyping Board provides the user with RCM3100 connection points brought out conveniently to labeled points at headers J2 and J4 on the Prototyping Board. Small to medium circuits can be prototyped using point-to-point wiring with 20 to 30 AWG wire between the prototyping area and the holes at locations J2 and J4. The holes are spaced at 0.1" (2.5 mm), and 40-pin headers or sockets may be installed at J2 and J4. The pinouts for locations J2 and J4, which correspond to headers J1 and J2, are shown in Figure B-4.

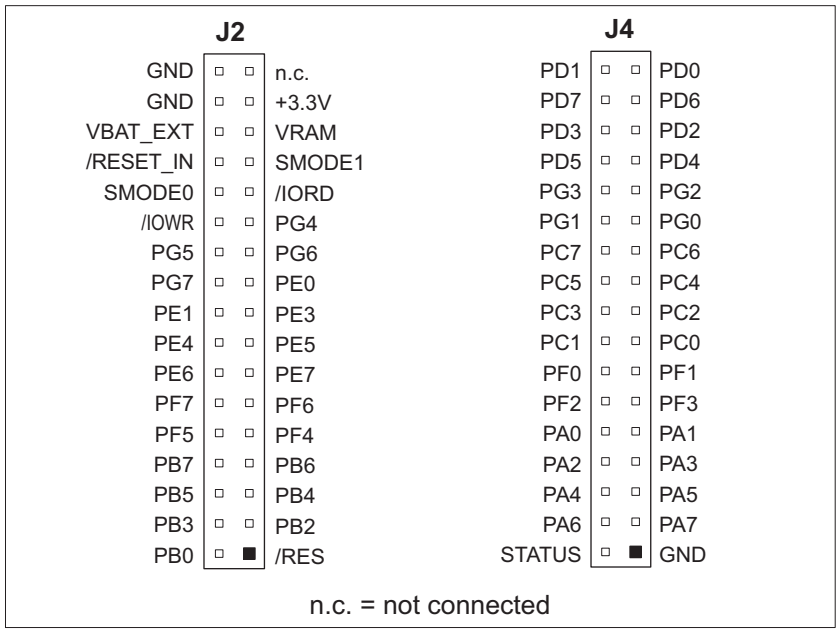


Figure B-4. RCM30/31/32XX Prototyping Board Pinout (Top View)

The small holes are also provided for surface-mounted components that may be installed around the prototyping area.

There is a 2.0" × 3.5" through-hole prototyping space available on the Prototyping Board. +3.3 V, +5 V, and GND traces run along the edge of the Prototyping Board for easy access.

B.4.1 Adding Other Components

There are pads that can be used for surface-mount prototyping involving SOIC devices. There is provision for seven 16-pin devices (six on one side, one on the other side). There are 10 sets of pads that can be used for 3- to 6-pin SOT23 packages. 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 on header JP1. Normally, a jumper connects pins 1–2 and pins 5–6 on header JP1, which provide jumper connections for the +5 V and the +3.3 V regulated voltages respectively. You may remove a jumper and place an ammeter across the pins instead, as shown in the example in Figure B-5, to measure the current being drawn.

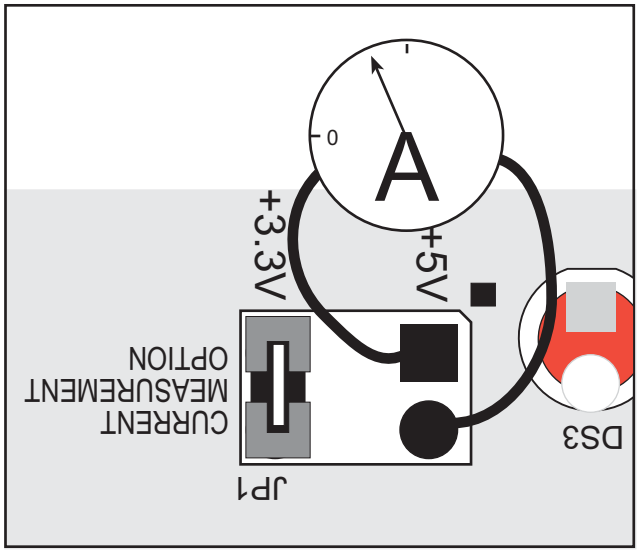


Figure B-5. Prototyping Board Current-Measurement Option

B.4.3 Other Prototyping Board Modules and Options

With the RCM3100 plugged into the **MASTER** slots, it has full access to the RS-232 transceiver, and can act as the "master" relative to another RabbitCore RCM3000, RCM3100, or RCM3200 plugged into the **SLAVE** slots, which acts as the "slave."

An optional LCD/keypad module is available that can be mounted on the Prototyping Board. Refer to Appendix C, "LCD/Keypad Module," for complete information.

The RCM3100 has a 2-channel quadrature decoder and a 10-bit free-running PWM counter with four pulse-width registers. These features allow the RCM3100 to be used in a motor control application, although Rabbit Semiconductor does not offer the drivers or a compatible stepper motor control board at this time.

The Prototyping Board has a header at J6 to which a customer-developed motor encoder may be connected. Figure B-6 shows the motor encoder pinout at header J6.

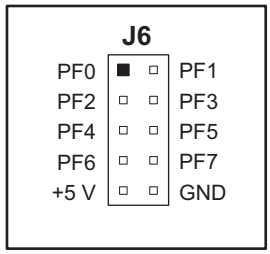


Figure B-6. Prototyping Board Motor Encoder Connector Pinout

Refer to Appendix E, "Motor Control Features," for complete information on using the Rabbit 3000's Parallel Port F in conjunction with this application.

B.5 Use of Rabbit 3000 Parallel Ports

Table B-2 lists the Rabbit 3000 parallel ports and their use for the RCM30/31/32XX Prototyping Board.

Table B-2. RCM30/31/32XX Prototyping Board Use of Rabbit 3000 Parallel Ports

Port	I/O	Use		Initial State
PA0–PA7	Output	Configurable external I/O bus		High when not driven by I/O bus
PB0–PB1	Input	Not used		Pulled up on RCM3100
PB2–PB5	Input	Configurable external I/O bus		High when not driven by I/O bus
PB6–PB7	Output	Not used		Pulled up on RCM3100
PC0	Output	Not used		High (disabled)
PC1	Input	Not used		Pulled up on RCM3100
PC2	Output	TXC	Serial Port C	High (disabled)
PC3	Input	RXC		Pulled up on RCM3100
PC4	Output	TXB	Serial Port B	High (disabled)
PC5	Input	RXB		Pulled up on RCM3100
PC6	Output	TXA Programming Port	Serial Port A	High (disabled)
PC7	Input	RXA Programming Port		Pulled up on RCM3100
PD0–PD4	Output	Not used		High
PD5	Input	Not used		Pulled up on Prototyping Board
PD6–PD7	Output	Not used		High
PE0–PE1	Output	Not used		High
PE2	Input	Not used		Pulled up on Prototyping Board
PE3	Output	LCD device select		Low (disabled)
PE4	Output	IrDA speed select		Low (disabled)
PE5	Output	Not used		High
PE6	Output	External I/O strobe		High (disabled)
PE7	Output	Not used		High (disabled)
PF0–PF7	Input	Reserved for future use		Pulled up on Prototyping Board

Table B-2. RCM30/31/32XX Prototyping Board Use of Rabbit 3000 Parallel Ports (continued)

Port	I/O	Use		Initial State
PG0	Input	Switch S3 (normally open)		High
PG1	Input	Switch S2 (normally open)		High
PG2	Output	TXF IrDA	Serial Port F	Pulled down
PG3	Input	RXF IrDA		Driven by IrDA driver
PG4	Input	IrDA MD1		Pulled up on Prototyping Board
PG5	Input	IrDA MD0		Pulled down on Prototyping Board
PG6	Output	LED DS1		High (disabled)
PG7	Output	LED DS2		High (disabled)

APPENDIX C. LCD/KEYPAD MODULE

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

C.1 Specifications

Two optional LCD/keypad modules—with or without a panel-mounted bezel—are available for use with the Prototyping Board. They are shown in Figure C-1.

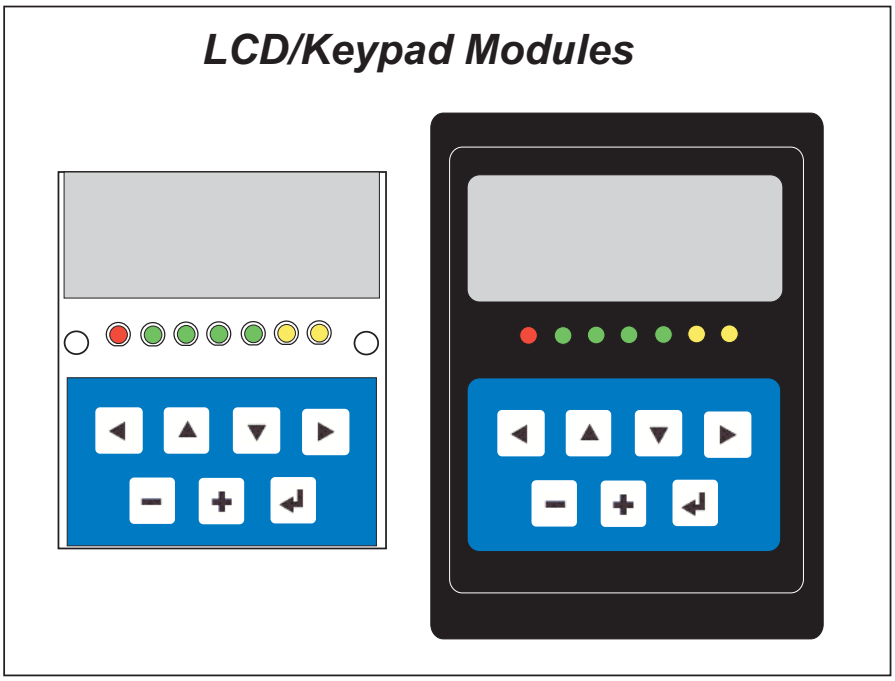


Figure C-1. LCD/Keypad Modules Models

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

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

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

Table C-1. LCD/Keypad Specifications

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

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

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

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

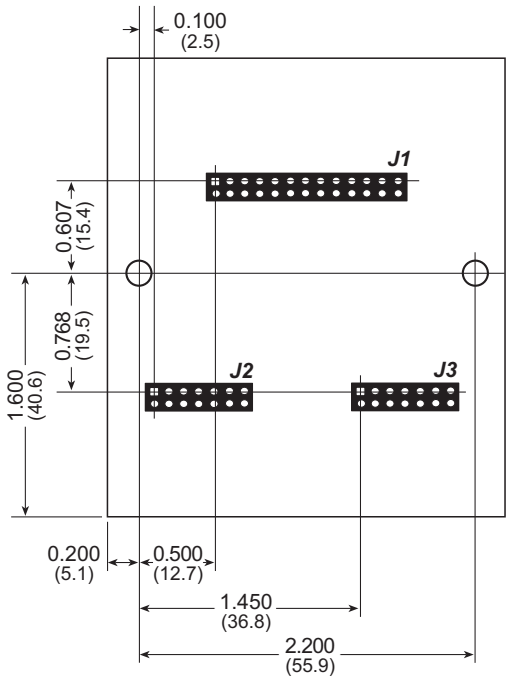


Figure C-2. User Board Footprint for LCD/Keypad Module

C.2 Contrast Adjustments for All Boards

Starting in 2005, LCD/keypad modules were factory-configured to optimize their contrast based on the voltage of the system they would be used in. Be sure to select a KDU5V LCD/keypad module for use with the RCM3000/3100/3200 Prototyping Board — these modules operate at 5 V. You may adjust the contrast using the potentiometer at R2 as shown in Figure C-3. LCD/keypad modules configured for 3.3 V should not be used with the 5 V RCM3000/3100/3200 Prototyping Board because the higher voltage will reduce the backlight service life dramatically.

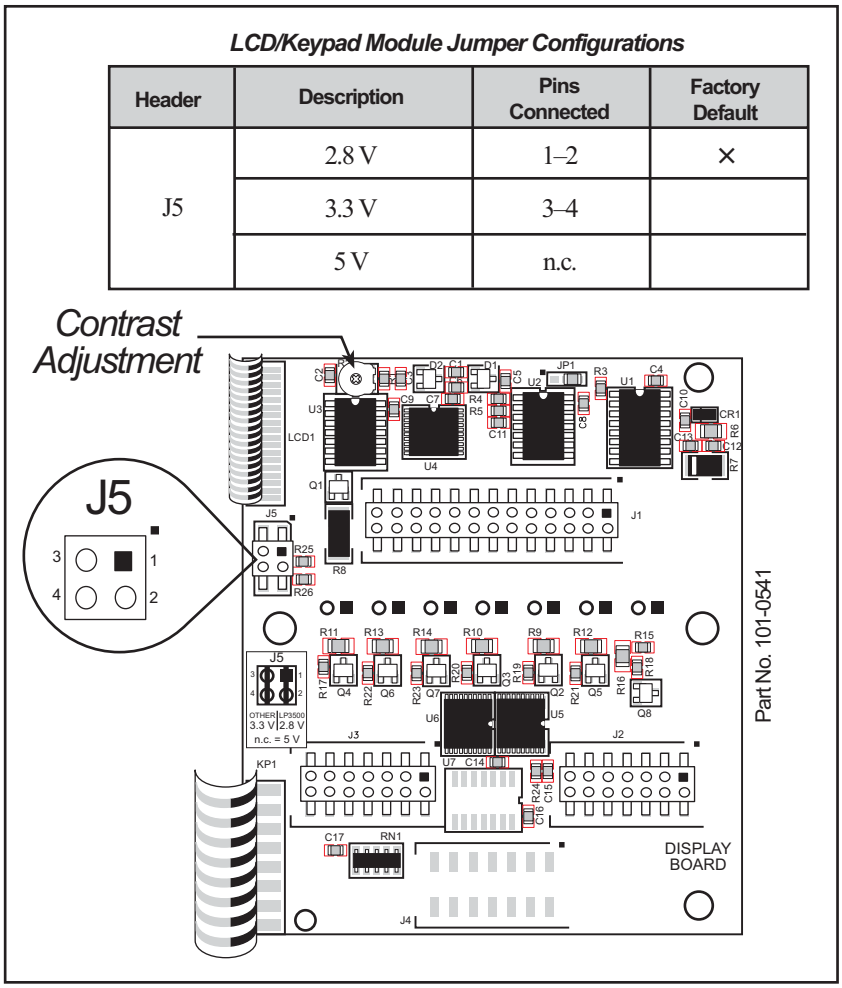


Figure C-3. LCD/Keypad Module Voltage Settings

You can set the contrast on the LCD display of pre-2005 LCD/keypad modules by adjusting the potentiometer at R2 or by setting the voltage for 5 V by removing the jumper across pins 1–2 on header J5 as shown in Figure C-3. Only one of these two options is available on these LCD/keypad modules.

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

C.3 Keypad Labeling

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

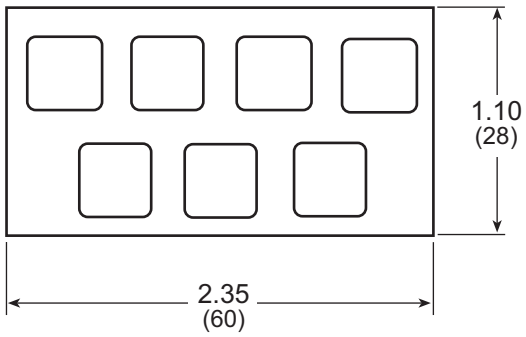


Figure C-4. Keypad Template

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

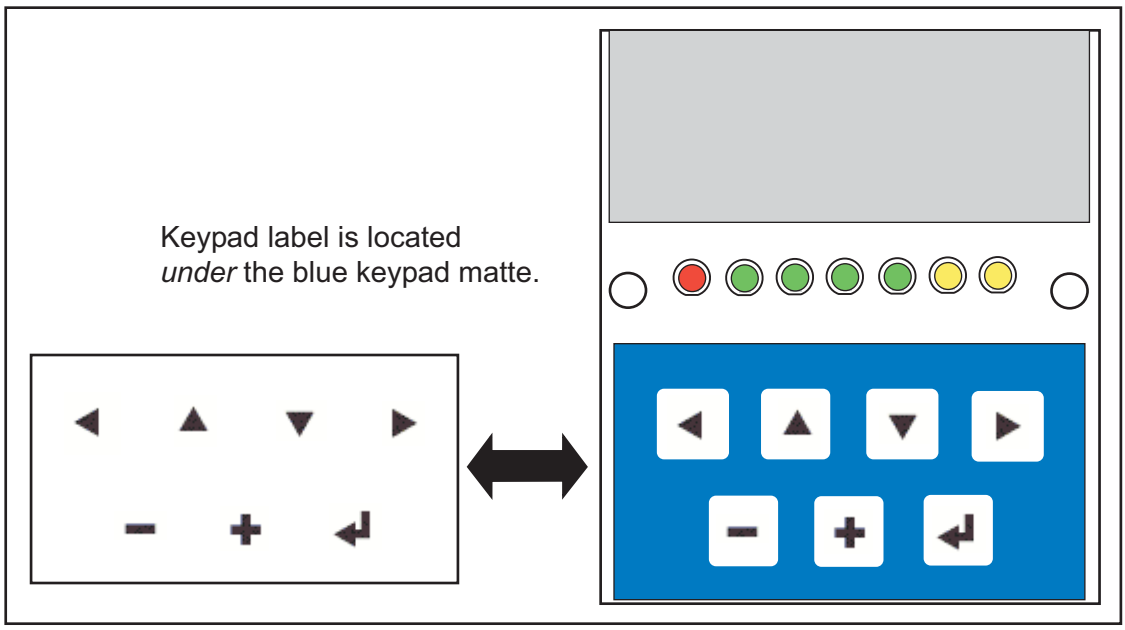


Figure C-5. Removing and Inserting Keypad Label

C.4 Header Pinouts

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

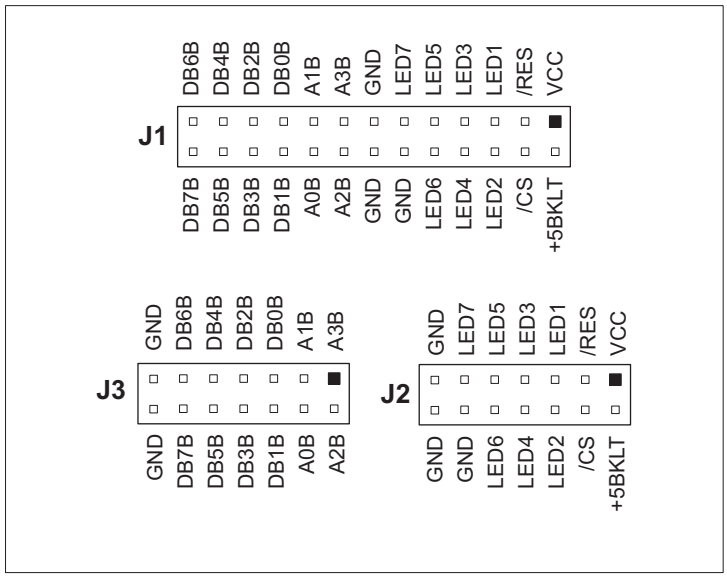


Figure C-6. LCD/Keypad Module Pinouts

C.4.1 I/O Address Assignments

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

Table C-2. LCD/Keypad Module Address Assignment

Address	Function
0xC000	Device select base address (/CS)
0xCxx0–0xCxx7	LCD control
0xCxx8	LED enable
0xCxx9	Not used
0xCxxA	7-key keypad
0xCxxB (bits 0–6)	7-LED driver
0xCxxB (bit 7)	LCD backlight on/off
0xCxxC–CxxF	Not used

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

Install the LCD/keypad module on header sockets J7, J8, and J10 of the Prototyping Board as shown in Figure C-7. Be careful to align the pins over the headers, and do not bend them as you press down to mate the LCD/keypad module with the Prototyping Board.

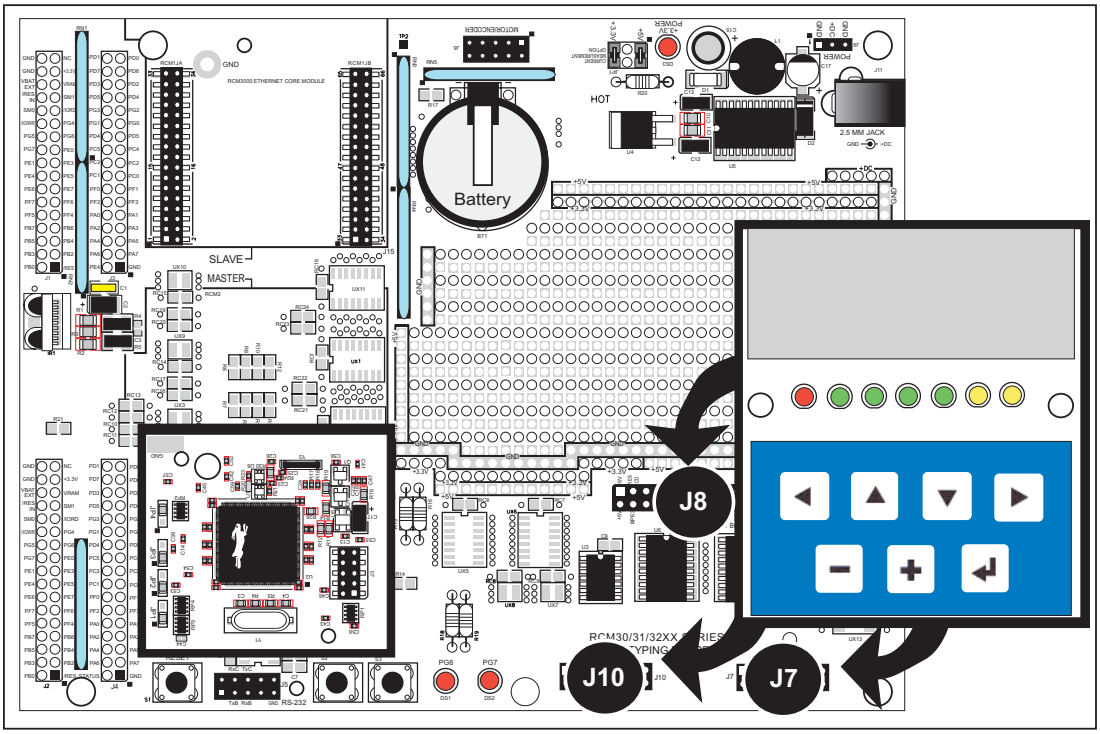


Figure C-7. Install LCD/Keypad Module on Prototyping Board

C.6 Bezel-Mount Installation

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

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

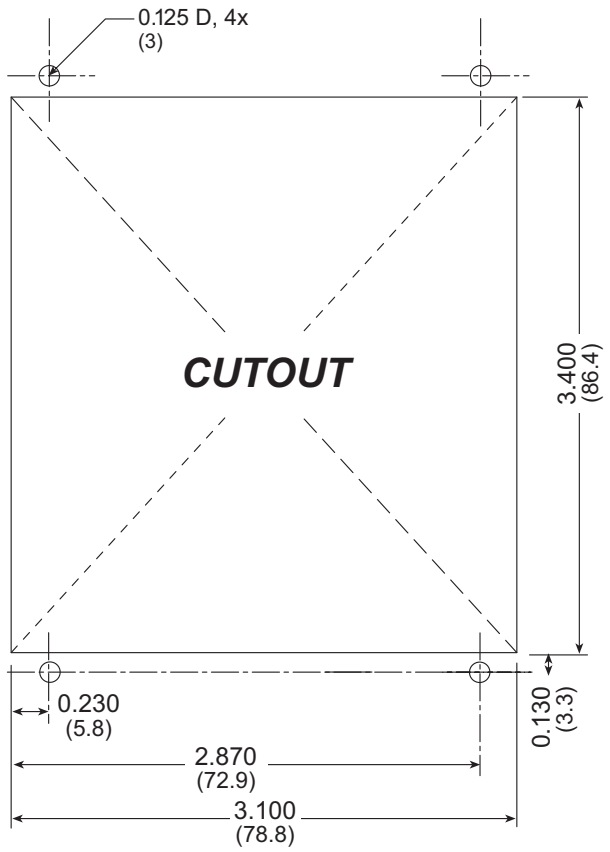


Figure C-8. Recommended Cutout Dimensions

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

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

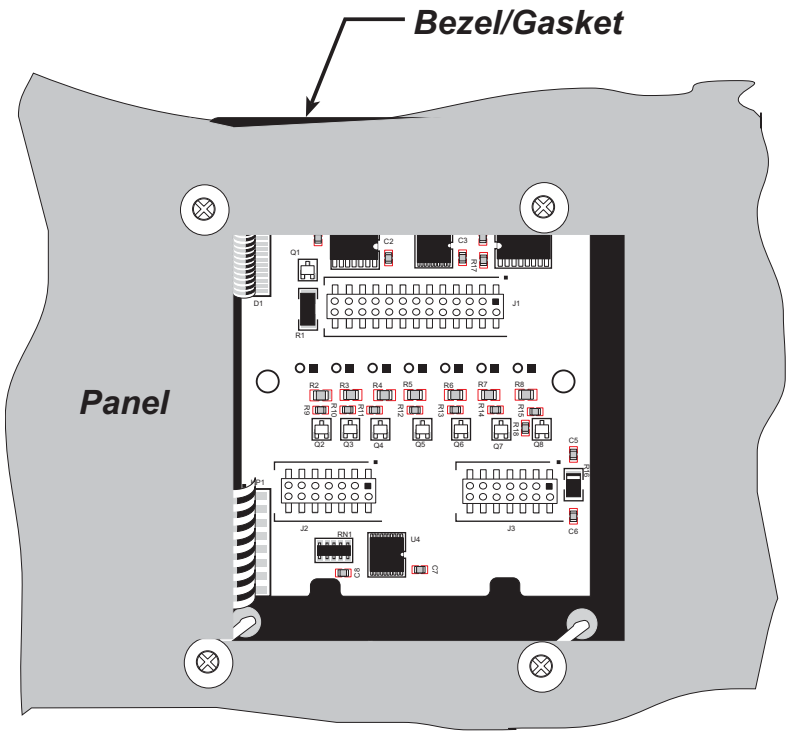


Figure C-9. LCD/Keypad Module Mounted in Panel (rear view)

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

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

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

The LCD/keypad module can be located as far as 2 ft. (60 cm) away from the RCM30/31/32XX Prototyping Board, and is connected via a ribbon cable as shown in Figure C-10.

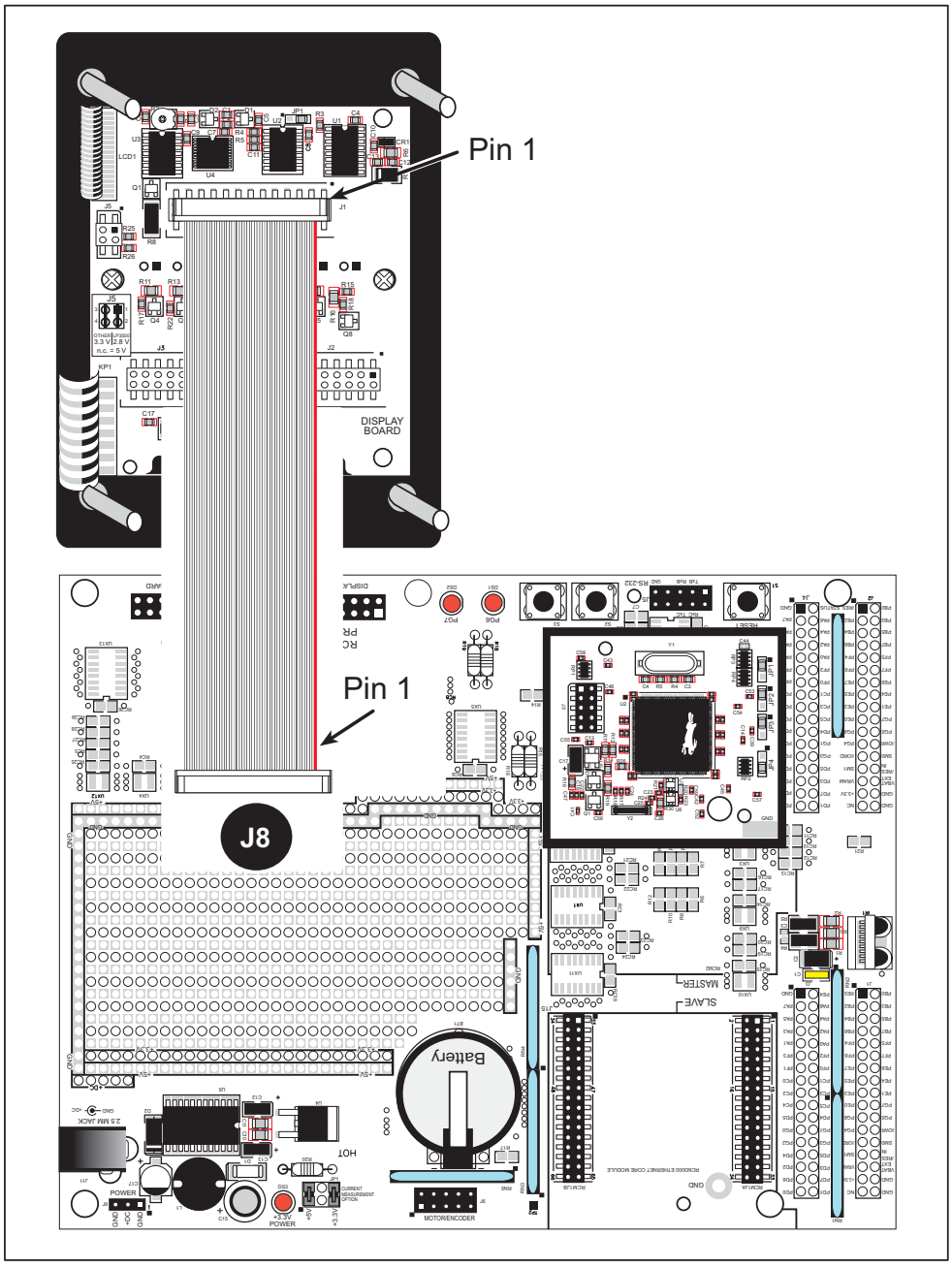


Figure C-10. Connecting LCD/Keypad Module to RCM30/31/32XX Prototyping Board

Note the locations and connections relative to pin 1 on both the RCM30/31/32XX Prototyping Board and the LCD/keypad module.

Rabbit Semiconductor offers 2 ft. (60 cm) extension cables. Contact your authorized distributor or a Rabbit Semiconductor sales representative for more information.

C.7 LCD/Keypad Module Function Calls

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

```
#define PORTA_AUX_IO
```

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

C.7.1 LCD/Keypad Module Initialization

The function used to initialize the LCD/keypad module can be found in the Dynamic C **LIB\DISPLAYS\LCD122KEY7.LIB** library.

```
void dispInit();
```

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

RETURN VALUE

None.

C.7.2 LEDs

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

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

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

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

PARAMETERS

led is the LED to control.

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

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

0 = off
1 = on

RETURN VALUE

None.

C.7.3 LCD Display

The functions used to control the LCD display are contained in the **GRAPHIC.LIB** library located in the Dynamic C **DISPLAYS\GRAPHIC** library directory.

void glInit(void);

Initializes the display devices, clears the screen.

RETURN VALUE

None.

SEE ALSO

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

void glBackLight(int onOff);

Turns the display backlight on or off.

PARAMETER

onOff turns the backlight on or off

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

RETURN VALUE

None.

SEE ALSO

glInit, glDispOnoff, glSetContrast

void glDispOnOff(int onOff);

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

PARAMETER

onOff turns the LCD screen on or off

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

RETURN VALUE

None.

SEE ALSO

glInit, glSetContrast, glBackLight

void glSetContrast(unsigned level);

Sets display contrast.

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

void glFillScreen(char pattern);

Fills the LCD display screen with a pattern.

PARAMETER

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

RETURN VALUE

None.

SEE ALSO

glBlock, glBlankScreen, glPlotPolygon, glPlotCircle

void glBlankScreen(void);

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

RETURN VALUE

None.

SEE ALSO

glFillScreen, glBlock, glPlotPolygon, glPlotCircle

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

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

PARAMETERS

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

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

bmWidth is the width of the block.

bmWidth is the height of the block.

RETURN VALUE

None.

SEE ALSO

glFillScreen, glBlankScreen, glPlotPolygon, glPlotCircle

void glPlotVPolygon(int n, int *pFirstCoord);

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

PARAMETERS

n is the number of vertices.

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

RETURN VALUE

None.

SEE ALSO

glPlotPolygon, glFillPolygon, glFillVPolygon

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

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

PARAMETERS

n is the number of vertices.

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

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

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

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

... are the coordinates of additional vertices.

RETURN VALUE

None.

SEE ALSO

glPlotVPolygon, glFillPolygon, glFillVPolygon

void glFillVPolygon(int n, int *pFirstCoord);

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

PARAMETERS

n is the number of vertices.

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

RETURN VALUE

None.

SEE ALSO

glFillPolygon, glPlotPolygon, glPlotVPolygon

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

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

PARAMETERS

n is the number of vertices.

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

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

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

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

... are the coordinates of additional vertices.

RETURN VALUE

None.

SEE ALSO

glFillVPolygon, glPlotPolygon, glPlotVPolygon

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

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

PARAMETERS

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

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

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

RETURN VALUE

None.

SEE ALSO

glFillCircle, glPlotPolygon, glFillPolygon

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

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

PARAMETERS

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

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

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

RETURN VALUE

None.

SEE ALSO

glPlotCircle, glPlotPolygon, glFillPolygon

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

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

PARAMETERS

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

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

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

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

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

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

RETURN VALUE

None.

SEE ALSO

glPrinf

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

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

PARAMETERS

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

letter is an ASCII character.

RETURN VALUE

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

SEE ALSO

glPutFont, glPrintf

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

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

PARAMETERS

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

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

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

code is the ASCII character to display.

RETURN VALUE

None.

SEE ALSO

glFontCharAddr, glPrintf

void glSetPfStep(int stepX, int stepY);

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

PARAMETERS

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

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

RETURN VALUE

None.

SEE ALSO

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

int glGetPfStep(void);

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

RETURN VALUE

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

SEE ALSO

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

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

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

PARAMETERS

ch is the character to be displayed on the LCD.

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

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

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

RETURN VALUE

None.

SEE ALSO

glPrintf, glPutFont, doprnt

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

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

PARAMETERS

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

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

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

***fmt** is a formatted string.

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

EXAMPLE

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

RETURN VALUE

None.

SEE ALSO

glXFontInit

void glBuffLock(void);

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

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

RETURN VALUE

None.

SEE ALSO

glBuffUnlock, glSwap

void glBuffUnlock(void);

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

RETURN VALUE

None.

SEE ALSO

glBuffLock, glSwap

void glSwap(void);

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

RETURN VALUE

None.

SEE ALSO

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

void glSetBrushType(int type);

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

PARAMETER

type value can be one of the following macros.

PIXBLACK draws black pixels.
PIXWHITE draws white pixels.
PIXXOR draws old pixel XOR'ed with the new pixel.

RETURN VALUE

None.

SEE ALSO

glGetBrushType

int glGetBrushType(void);

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

RETURN VALUE

The current brush type.

SEE ALSO

`glSetBrushType`

void glPlotDot(int x, int y);

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

PARAMETERS

`x` is the *x* coordinate of the dot.

`y` is the *y* coordinate of the dot.

RETURN VALUE

None.

SEE ALSO

`glPlotline, glPlotPolygon, glPlotCircle`

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

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

PARAMETERS

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

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

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

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

RETURN VALUE

None.

SEE ALSO

`glPlotDot, glPlotPolygon, glPlotCircle`

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

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

PARAMETERS

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

top is the top left corner of the bitmap.

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

rows is the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glHScroll, glRight1

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

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

PARAMETERS

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

top is the top left corner of the bitmap.

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

rows is the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glHScroll, glLeft1

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

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

PARAMETERS

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

top is the top left corner of the bitmap.

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

rows is the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glVScroll, glDown1

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

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

PARAMETERS

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

top is the top left corner of the bitmap.

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

rows is the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glVScroll, glUp1

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

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

Parameters will be verified for the following:

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

PARAMETERS

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

top is the top left corner of the bitmap.

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

rows is the number of rows in the window.

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

RETURN VALUE

None.

SEE ALSO

glVScroll

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

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

Parameters will be verified for the following:

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

PARAMETERS

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

top is the top left corner of the bitmap.

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

rows is the number of rows in the window.

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

RETURN VALUE

None.

SEE ALSO

glHScroll

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

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

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

PARAMETERS

left is the top left corner of the bitmap.

top is the top left corner of the bitmap.

width is the width of the bitmap.

height is the height of the bitmap.

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

RETURN VALUE

None.

SEE ALSO

glXPutFastmap, glPrintf

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

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

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

PARAMETERS

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

top is the top left corner of the bitmap.

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

height is the height of the bitmap.

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

RETURN VALUE

None.

SEE ALSO

glXPutBitmap, glPrintf

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

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

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

PARAMETERS

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

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

x is the *x* coordinate of where the text window frame is to start.

y is the *y* coordinate of where the text window frame is to start.

winWidth is the width of the text window frame.

winHeight is the height of the text window frame.

RETURN VALUE

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

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

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

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

PARAMETERS

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

col is a character column location.

row is a character row location.

RETURN VALUE

None.

SEE ALSO

TextPutChar, TextPrintf, TextWindowFrame

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

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

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

PARAMETERS

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

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

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

RETURN VALUE

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

SEE ALSO

TextGotoXY, TextPrintf, TextWindowFrame, TextCursorLocation

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

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

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

PARAMETERS

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

ch is a character to be displayed on the LCD.

RETURN VALUE

None.

SEE ALSO

TextGotoXY, TextPrintf, TextWindowFrame, TextCursorLocation

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

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

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

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

PARAMETERS

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

***fmt** is a formatted string.

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

EXAMPLE

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

RETURN VALUE

None.

SEE ALSO

TextGotoXY, TextPutChar, TextWindowFrame, TextCursorLocation

C.7.4 Keypad

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

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

Initializes keypad process

RETURN VALUE

None.

SEE ALSO

brdInit

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

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

PARAMETERS

cRaw is a raw key code index.

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

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

User Keypad Interface

cPress is a key press code

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

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

cRelease is a key release code.

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

cCntHold is a hold tick.

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

cSpdLo is a low-speed repeat tick.

How many times to repeat.
0 = None.

cCntLo is a low-speed hold tick.

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

cSpdHi is a high-speed repeat tick.

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

RETURN VALUE

None.

SEE ALSO

keyProcess, keyGet, keypadDef

void keyProcess(void);

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

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

RETURN VALUE

None

SEE ALSO

keyConfig, keyGet, keypadDef

char keyGet(void);

Get next keypress

RETURN VALUE

The next keypress, or 0 if none

SEE ALSO

keyConfig, keyProcess, keypadDef

int keyUnget(char cKey);

Push keypress on top of input queue

PARAMETER

cKey

RETURN VALUE

None.

SEE ALSO

keyGet

void keypadDef();

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

Keypad physical mapping 1 × 7

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

where

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

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

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

Characters are returned upon keypress with no repeat.

RETURN VALUE

None.

SEE ALSO

`keyConfig, keyGet, keyProcess`

void keyScan(char *pcKeys);

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

PARAMETER

`*pcKeys` is the address of the value read.

RETURN VALUE

None.

SEE ALSO

`keyConfig, keyGet, keypadDef, keyProcess`

C.8 Sample Programs

Sample programs illustrating the use of the LCD/keypad module with the Prototyping Board are provided in the **SAMPLES\RCM3100** directory.

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

Each sample program has comments that describe the purpose and function of the program. Follow the instructions at the beginning of the sample program. To run a sample program, open it with the **File** menu (if it is not still open), compile it using the **Compile** menu, and then run it by selecting **Run** in the **Run** menu. The RCM3100 must be in **Program** mode (see Section 4.3, "Serial Programming Cable"), and must be connected to a PC using the programming cable as described in Chapter 2.

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

The following sample programs are found in the **SAMPLES\RCM3100\LCD_KEYPAD** folder.

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

APPENDIX D. POWER SUPPLY

Appendix D provides information on the current requirements of the RCM3100, and includes some background on the chip select circuit used in power management.

D.1 Power Supplies

The RCM3100 requires a regulated 3.3 V ± 0.15 V DC power source. The RabbitCore design presumes that the voltage regulator is on the user board, and that the power is made available to the RCM3100 board through header J2.

An RCM3100 with no loading at the outputs operating at 29.4 MHz typically draws 75 mA. The RCM3100 will consume an additional 10 mA when the programming cable is used to connect the programming header, J3, to a PC.

D.1.1 Battery-Backup Circuits

The RCM3100 does not have a battery, but there is provision for a customer-supplied battery to back up SRAM and keep the internal Rabbit 3000 real-time clock running.

Header J2, shown in Figure D-1, allows access to the external battery. This header makes it possible to connect an external 3 V power supply. This allows the SRAM and the internal Rabbit 3000 real-time clock to retain data with the RCM3100 powered down.

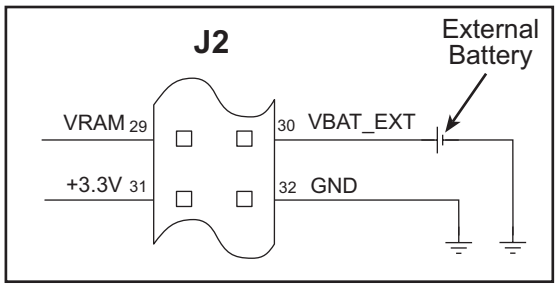


Figure D-1. External Battery Connections at Header J5

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

The drain on the battery by the RCM3100 is typically 7.1 µA when no other power is supplied. If a 165 mA·h battery is used, the battery can last almost 3 years:

$$\frac{165\ \text{mA}\cdot\text{h}}{7\ \mu\text{A}} = 2.7\ \text{years}.$$

The actual life in your application will depend on the current drawn by components not on the RCM3100 and the storage capacity of the battery. The RCM3100 does not drain the battery while it is powered up normally.

Cycle the main power off/on on the RCM3100 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 RCM3100 experience a loss of main power.

D.1.2 Reset Generator

The RCM3100 uses a reset generator to reset the Rabbit 3000 microprocessor when the voltage drops below the voltage necessary for reliable operation. The reset occurs between 2.55 V and 2.70 V, typically 2.63 V. The RCM3100 has a reset output, pin 1 on header J2.

APPENDIX E. MOTOR CONTROL FEATURES

The RCM30/31/32XX Prototyping Board has a header at J6 for a motor control connection. While Rabbit Semiconductor does not have the drivers or a compatible stepper motor control board at this time, this appendix provides additional information about Parallel Port F on the Rabbit 3000 microprocessor to enable you to develop your own application.

E.1 Overview

The Parallel Port F connector on the Prototyping Board, J6, gives access to all 8 pins of Parallel Port F, along with +5 V. This appendix describes the function of each pin, and the ways they may be used for motion-control applications. It should be read in conjunction with the ***Rabbit 3000 Microprocessor User's Manual*** and the RCM3100 and the RCM3000/RCM3100/RCM3200 Prototyping Board schematics.

E.2 Header J6

The connector is a 2 × 5, 0.1" pitch header suitable for connecting to an IDC receptacle with the following pin allocations.

Table E-1. RCM30/31/32XX Prototyping Board Header J6 Pinout

Pin	Rabbit 3000	Primary Function	Alternate Function 1	Alternate Function 2
1	Parallel Port F, bit 0	General-purpose I/O port	Quadrature decoder 1 Q input	SCLK_D
2	Parallel Port F, bit 1	General-purpose I/O port	Quadrature decoder 1 I input	SCLK_C
3	Parallel Port F, bit 2	General-purpose I/O port	Quadrature decoder 2 Q input	-
4	Parallel Port F, bit 3	General-purpose I/O port	Quadrature decoder 2 I input	-
5	Parallel Port F, bit 4	General-purpose I/O port	PWM[0] output	Quadrature decoder 1 Q input
6	Parallel Port F, bit 5	General-purpose I/O port	PWM[1] output	Quadrature decoder 1 I input
7	Parallel Port F, bit 6	General-purpose I/O port	PWM[2] output	Quadrature decoder 2 Q input
8	Parallel Port F, bit 7	General-purpose I/O port	PWM[3] output	Quadrature decoder 2 I input
9	+5 V	External buffer logic supply		
10	0 V	Common		

All eight Parallel Port F lines are pulled up internally to +3.3 V via 100 kΩ resistors. When used as outputs, the port pins will sink up to 6 mA at a V_{OL} of 0.4 V max. (0.2 V typ), and source up to 6 mA at a V_{OH} of 2.2 V typ. When used as inputs, all pins are 5 V tolerant.

As the outputs from Parallel Port F are compatible with 3.3 V logic, buffers may be needed when the external circuit drive requirements exceed the 2.2 V typ logic high and/or the 6 mA maximum from the Rabbit 3000. The +5 V supply output is provided for supplying interface logic. When used as inputs, the pins on header J6 do not require buffers unless the input voltage will exceed the 5 V tolerance of the processor pins. Usually, a simple resistive divider with catching diodes will suffice if higher voltage inputs are required. If the outputs are configured for open-drain operation, they may be pulled up to +5 V (while observing the maximum current, of course).

E.3 Using Parallel Port F

Parallel Port F is a byte-wide port with each bit programmable for data direction and drive. These are simple inputs and outputs controlled and reported in the Port F Data Register. As outputs, the bits of the port are buffered, with the data written to the Port F Data Register transferred to the output pins on a selected timing edge. The outputs of Timer A1, Timer B1, or Timer B2 can be used for this function, with each nibble of the port having a separate select field to control this timing. These inputs and outputs are also used for access to other peripherals on the chip.

As outputs, Parallel Port F can carry the four Pulse Width Modulator outputs on PF4–PF7 (J6 pins 5–8). As inputs, Parallel Port F can carry the inputs to the quadrature decoders on PF0–PF3 (J6 pins 1–4). When Serial Port C or Serial Port D is used in the clocked serial mode, two pins of Port F (PF0 / J6:1 and PF1 / J6:2) are used to carry the serial clock signals. When the internal clock is selected in these serial ports, the corresponding bit of Parallel Port F is set as an output.

E.3.1 Parallel Port F Registers

Data Direction Register—**PFDDR**, address 00111111 (0x3F), write-only, default value on reset 00000000. For each bit position, write a 1 to make the corresponding port line an output, or 0 to produce an input.

Drive Control Register—**PFDCR**, address 00111110 (0x3E), Write-only, no default on reset (port defaults to all inputs). Effective only if the corresponding port bits are set as outputs, each bit set to 1 configures the corresponding port bit as open drain. Setting the bit to 0 configures that output as active high or low.

Function Register—**PFFR**, address 00111101 (0x3D), Write-only, no default on reset. This register sets the alternate output function assigned to each of the pins of the port. When set to 0, the corresponding port pin functions normally as an output (if configured to be an output in **PFDDR**). When set to 1, each bit sets the corresponding pin to have the alternate output function as shown in the summary table at the end of this section.

Control Register—**PFCR**, address 00111100 (0x3C), Write-only, default on reset xx00xx00. This register sets the transfer clock, which controls the timing of the outputs on each nibble of the output ports to allow close synchronization with other events. The summary table at the end of this section shows the settings for this register. The default values on reset transfer the output values on **CLK**/2.

Data Register—**PFDR**, address 00111000 (0x38), Read or Write, no default value on reset. On read, the current state of the pins is reported. On write, the output buffer is written with the value for transfer to the output port register on the next rising edge of the transfer clock, set in the **PFCR**.

Table E-2. Parallel Port F Registers

Register Name	Mnemonic	I/O Address	R/W	Reset Value
Port F Data Register	PFDR	00111000 (0x38)	R/W	xxxxxxxx
Bits	**Value**	**Description**		
0:7	Read	Current state of pins		
	Write	Port buffer. Value transferred to O/P register on next rising edge of transfer clock.		
Port F Control Register	PFCR	00111100 (0x3C)	W only	xx00xx00
Bits	**Value**	**Description**		
0:1	00	Lower nibble transfer clock is CLK/2		
	01	Lower nibble transfer clock is Timer A1		
	10	Lower nibble transfer clock is Timer B1		
	11	Lower nibble transfer clock is Timer B2		
2:3	xx	These bits are ignored		
4:5	00	Upper nibble transfer clock is CLK/2		
	01	Upper nibble transfer clock is Timer A1		
	10	Upper nibble transfer clock is Timer B1		
	11	Upper nibble transfer clock is Timer B2		
6:7	xx	These bits are ignored		
Port F Function Register	PFFR	00111101 (0x3D)	W	xxxxxxxx
Bits	**Value**	**Description**		
0:7	0	Corresponding port bits function normally		
0	1	Bit 0 carries SCLK_D		
1	1	Bit 1 carries SCLK_C		
2:3	x	No effect		
4	1	Bit 4 carries PWM[0] output		
5	1	Bit 5 carries PWM[1] output		
6	1	Bit 6 carries PWM[2] output		
7	1	Bit 7 carries PWM[3] output		
Port F Drive Control Register	PFDCR	00111110 (0x3E)	W	xxxxxxxx
Bits	**Value**	**Description**		
0:7	0	Corresponding port bit is active high or low		
	1	Corresponding port bit is open drain		

Table E-2. Parallel Port F Registers (continued)

Register Name	Mnemonic	I/O Address	R/W	Reset Value
Port F Data Direction Register	PFDDR	00111111 (0x3F)	W	00000000
Bits	**Value**	**Description**		
0:7	0	Corresponding port bit is an input		
	1	Corresponding port bit is an output		

E.4 PWM Outputs

The Pulse-Width Modulator consists of a 10-bit free-running counter and four width registers. Each PWM output is high for $n + 1$ counts out of the 1024-clock count cycle, where n is the value held in the width register. The PWM output high time can optionally be spread throughout the cycle to reduce ripple on the externally filtered PWM output. The PWM is clocked by the output of Timer A9. The spreading function is implemented by dividing each 1024-clock cycle into four quadrants of 256 clocks each. Within each quadrant, the Pulse-Width Modulator uses the eight MSBs of each pulse-width register to select the base width in each of the quadrants. This is the equivalent to dividing the contents of the pulse-width register by four and using this value in each quadrant. To get the exact high time, the Pulse-Width Modulator uses the two LSBs of the pulse-width register to modify the high time in each quadrant according to Table E-3 below. The "n/4" term is the base count, and is formed from the eight MSBs of the pulse-width register.

Table E-3. PWM Outputs

Pulse Width LSBs	1st	2nd	3rd	4th
00	n/4 + 1	n/4	n/4	n/4
01	n/4 + 1	n/4	n/4 + 1	n/4
10	n/4 + 1	n/4 + 1	n/4 + 1	n/4
11	n/4 + 1	n/4 + 1	n/4 + 1	n/4 + 1

The diagram below shows a PWM output for several different width values for both modes of operation. Operation in the spread mode reduces the filtering requirements on the PWM output in most cases.

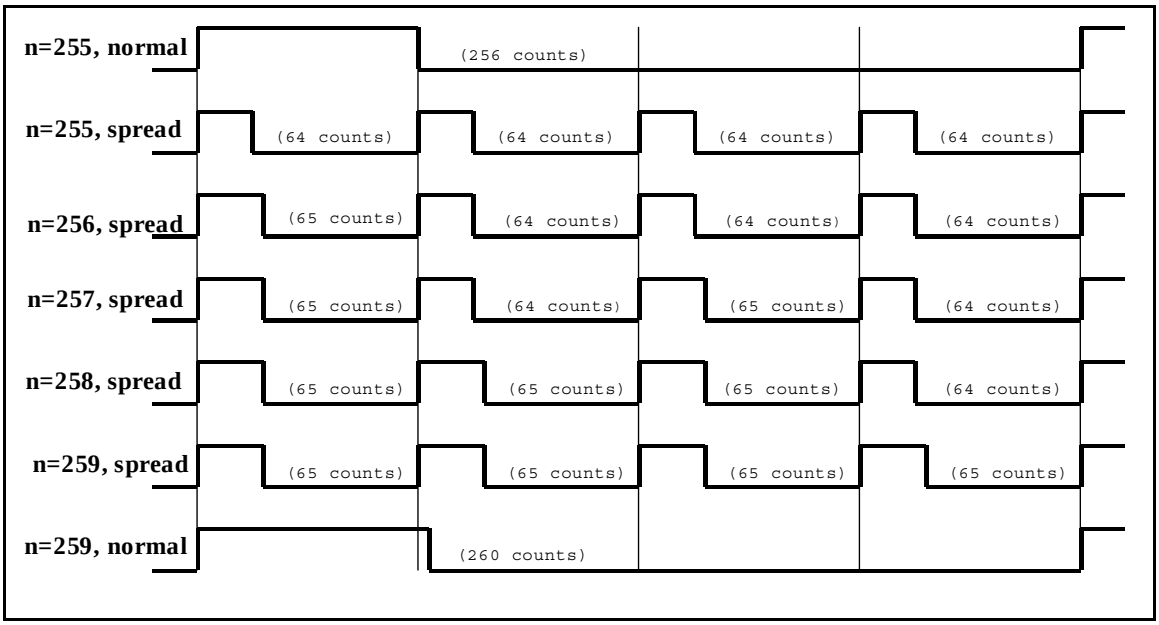


Figure E-1. PWM Outputs for Various Normal and Spread Modes

E.5 PWM Registers

There are no default values on reset for any of the PWM registers.

Table E-4. PWM Registers

PWM LSBs	Register	Address
	PWL0R	10001000 (0x88)
	PWL1R	10001010 (0x8A)
	PWL2R	10001100 (0x8C)
	PWL3R	10001110 (0x8E)
Bit(s)	**Value**	**Description**
7:6	Write	The least significant two bits for the Pulse Width Modulator count are stored
5:1		These bits are ignored.
0	0	PWM output High for single block.
	1	Spread PWM output throughout the cycle
PWM MSB x	**Register**	**Address**
	PWM0R	Address = 10001001 (0x89)
	PWM1R	Address = 10001011 (0x8B)
	PWM2R	Address = 10001101 (0x8D)
	PWM3R	Address = 10001111 (0x8F)
Bit(s)	**Value**	**Description**
7:0	write	The most significant eight bits for the Pulse-Width Modulator count are stored With a count of *n*, the PWM output will be high for *n* +1 clocks out of the 1024 clocks of the PWM counter.

E.6 Quadrature Decoder

The two-channel Quadrature Decoder accepts inputs via Parallel Port F from two external optical incremental encoder modules. Each channel of the Quadrature Decoder accepts an in-phase (I) and a quadrature-phase (Q) signal, and provides 8-bit counters to track shaft rotation and provide interrupts when the count goes through the zero count in either direction. The Quadrature Decoder contains digital filters on the inputs to prevent false counts and is clocked by the output of Timer A10. Each Quadrature Decoder channel accepts inputs from either the upper nibble or lower nibble of Parallel Port F. The I signal is input on an odd-numbered port bit, while the Q signal is input on an even-numbered port bit. There is also a disable selection, which is guaranteed not to generate a count increment or decrement on either entering or exiting the disable state. The operation of the counter as a function of the I and Q inputs is shown below.

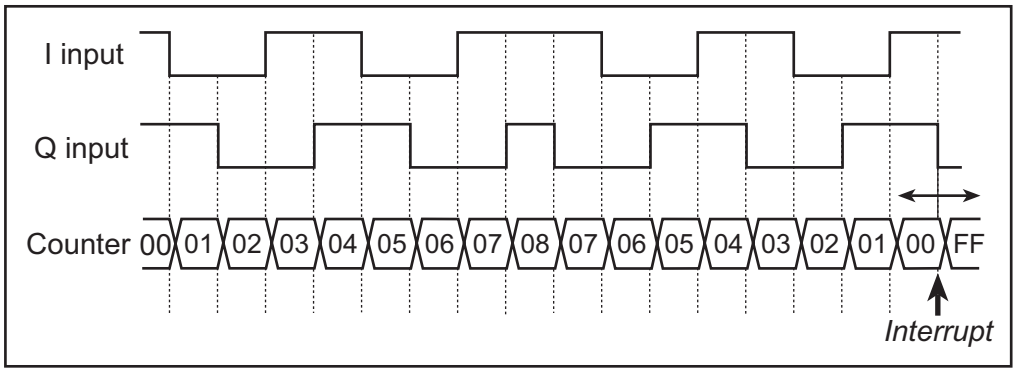
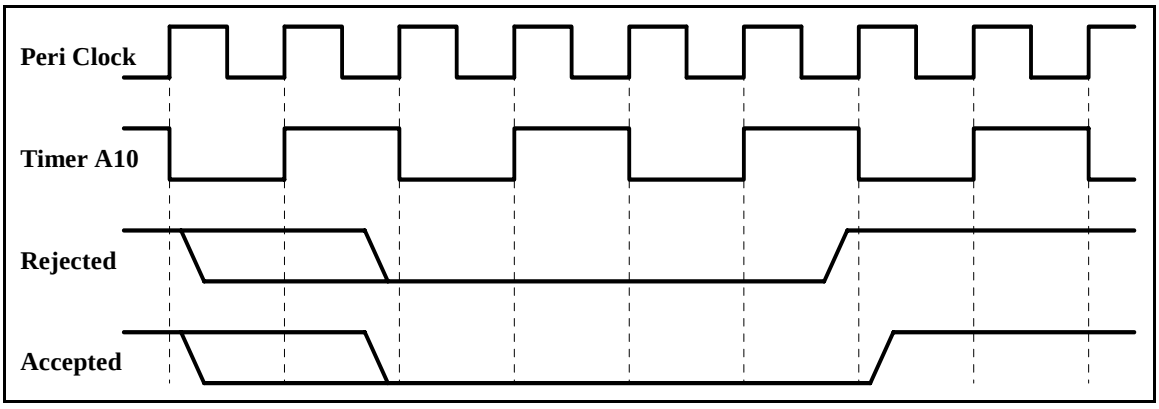


Figure E-2. Operation of Quadrature Decoder Counter

The Quadrature Decoders are clocked by the output of Timer A10, giving a maximum clock rate of one-half of the peripheral clock rate. The time constant of Timer A10 must be fast enough to sample the inputs properly. Both the I and Q inputs go through a digital filter that rejects pulses shorter than two clock periods wide. In addition, the clock rate must be high enough that transitions on the I and Q inputs are sampled in different clock cycles. The Input Capture (see the ***Rabbit 3000 Microprocessor Users Manual***) may be used to measure the pulse width on the I inputs because they come from the odd-numbered port bits. The operation of the digital filter is shown below.



The Quadrature Decoder generates an interrupt when the counter increments from 0x00 to 0x01 or when the counter decrements from 0x00 to 0xFF. Note that the status bits in the QDCSR are set coincident with the interrupt, and the interrupt (and status bits) are cleared by reading the QDCSR.

Table E-5. Quadrature Decoder Registers

Register Name	Mnemonic	Address
Quad Decode Control/Status Register	QDCSR	10010000 (0x90)
Bit	**Value**	**Description**
7 (rd-only)	0	Quadrature Decoder 2 did not increment from 0xFF.
	1	Quadrature Decoder 2 incremented from 0xFF to 0x00. This bit is cleared by a read of this register.
6 (rd-only)	0	Quadrature Decoder 2 did not decrement from 0x00.
	1	Quadrature Decoder 2 decremented from 0x00 to 0xFF. This bit is cleared by a read of this register
5	0	This bit always reads as zero.
4 (wr-only)	0	No effect on the Quadrature Decoder 2.
	1	Reset Quadrature Decoder 2 to 0x00, without causing an interrupt.
3 (rd-only)	0	Quadrature Decoder 1 did not increment from 0xFF.
	1	Quadrature Decoder 1 incremented from 0xFF to 0x00. This bit is cleared by a read of this register.
2 (rd-only)	0	Quadrature Decoder 1 did not decrement from 0x00.
	1	Quadrature Decoder 1 decremented from 0x00 to 0xFF. This bit is cleared by a read of this register.
1	0	This bit always reads as zero.
Bit	**Value**	**Description**
0 (wr-only)	0	No effect on the Quadrature Decoder 1.
	1	Reset Quadrature Decoder 1 to 0x00, without causing an interrupt.

Table E-5. Quadrature Decoder Registers (continued)

Register Name	Mnemonic	Address
Quad Decode Control Register	QDCR	Address = 10010001 (0x91)
Bit	**Value**	**Description**
7:6	0x	Disable Quadrature Decoder 2 inputs. Writing a new value to these bits will not cause Quadrature Decoder 2 to increment or decrement.
	10	Quadrature Decoder 2 inputs from Port F bits 3 and 2.
	11	Quadrature Decoder 2 inputs from Port F bits 7 and 6.
5:4	xx	These bits are ignored.
3:2	0x	Disable Quadrature Decoder 1 inputs. Writing a new value to these bits will not cause Quadrature Decoder 1 to increment or decrement.
	10	Quadrature Decoder 1 inputs from Port F bits 1 and 0.
	11	Quadrature Decoder 1 inputs from Port F bits 5 and 4.
1:0	0	Quadrature Decoder interrupts are disabled.
	1	Quadrature Decoder interrupt use Interrupt Priority 1.
	10	Quadrature Decoder interrupt use Interrupt Priority 2.
	11	Quadrature Decoder interrupt use Interrupt Priority 3.
Quad Decode Count Register	QDC1R	Address = 10010100 (0x94)
	(QDC2R)	Address = 10010110 (0x96)
Bit(s)	**Value**	**Description**
7:0	read	The current value of the Quadrature Decoder counter is reported.

INDEX

M

P

Q

R

S

T

U

SCHEMATICS

090-0144 RCM3100 Schematic

www.rabbit.com/documentation/schemat/090-0144.pdf

090-0137 RCM3000/RCM3100/RCM3200 Prototyping Board Schematic

www.rabbit.com/documentation/schemat/090-0137.pdf

090-0156 LCD/Keypad Module Schematic

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

090-0128 Programming Cable Schematic

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

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