# RabbitCore RCM3600

C-Programmable Core Module

## User's Manual

019–0135 • 070831–E

# RabbitCore RCM3600 User's Manual

Part Number 019-0135 • 070831–E • Printed in U.S.A.

©2003–2007 Rabbit Semiconductor Inc. • All rights reserved.

Rabbit Semiconductor reserves the right to make changes and
improvements to its products without providing notice.

## 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 RCM3600 is a compact module that incorporates the powerful Rabbit® 3000 microprocessor, flash memory, static RAM, and digital I/O ports.

The Development Kit has what you need to design your own microprocessor-based system: a complete Dynamic C software development system and a Prototyping Board that acts as a motherboard to allow you to evaluate the RCM3600 and to prototype circuits that interface to the RCM3600 module.

The RCM3600 has a Rabbit 3000 microprocessor operating at 22.1 MHz, static RAM, flash memory, two clocks (main oscillator and real-time clock), and the circuitry necessary for reset and management of battery backup of the Rabbit 3000's internal real-time clock and the static RAM. One 40-pin header brings out the Rabbit 3000 I/O bus lines, parallel ports, and serial ports.

The RCM3600 receives its +5 V power from the customer-supplied motherboard on which it is mounted. The RCM3600 can interface with all kinds of CMOS-compatible digital devices through the motherboard.

## 1.1 RCM3600 Features

- Small size: 1.23" x 2.11" x 0.62"
  (31 mm x 54 mm x 16 mm)

- Microprocessor: Rabbit 3000 running at 22.1 MHz

- 33 parallel 5 V tolerant I/O lines: 31 configurable for I/O, 2 fixed outputs

- External reset I/O

- Alternate I/O bus can be configured for 8 data lines and 5 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

- 512K flash memory, 512K SRAM (options for 256K flash memory and 128K SRAM)

- Real-time clock

- Watchdog supervisor

- Connections via header J1 for customer-supplied backup battery

- 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 modules

- Four available 3.3 V CMOS-compatible serial ports with a maximum asynchronous baud rate of 2.76 Mbps. Three ports are configurable as a clocked serial port (SPI), and one port is configurable as an HDLC serial port. Shared connections to the Rabbit microprocessor make a second HDLC serial port available at the expense of two of the SPI configurable ports, giving you two HDLC ports and one asynchronous/SPI serial port.

- Supports 1.15 Mbps IrDA transceiver

There are two RCM3600 production models. If the standard models do not serve your needs, variations can be specified and ordered in production quantities. Contact your Rabbit Semiconductor sales representative for details.

Table 1 below summarizes the main features of the RCM3600.

### Table 1.  RCM3600 Features

| Feature | RCM3600 | RCM3610 |
|---|---|---|
| Microprocessor | Rabbit 3000 running at 22.1 MHz | |
| Flash Memory | 512K | 256K |
| SRAM | 512K | 128K |
| Serial Ports | 4 shared high-speed, 3.3 V CMOS-compatible ports: all 4 are configurable as asynchronous serial ports; 3 are configurable as a clocked serial port (SPI) and 1 is configurable as an HDLC serial port; option for second HDLC serial port at the expense of 2 clocked serial ports (SPI) | |

The RCM3600 can be programed through a USB port with an RS-232/USB converter, or over an Ethernet with the RabbitLink.

Appendix A provides detailed specifications for the RCM3600.

## 1.2  Advantages of the RCM3600

- Fast time to market using a fully engineered, "ready-to-run/ready-to-program" micro-processor core.

- Competitive pricing when compared with the alternative of purchasing and assembling individual components.

- Easy C-language program development and debugging

- Rabbit Field Utility to download compiled Dynamic C .bin files, and cloning board options for rapid production loading of programs.

- Generous memory size allows large programs with tens of thousands of lines of code, and substantial data storage.

## 1.3  Development and Evaluation Tools

### 1.3.1  Development Kit

The Development Kit contains the hardware you need to use your RCM3600 module.

- RCM3600 module.

- Prototyping Board.

- AC adapter, 12 V DC, 500 mA (included only with Development Kits sold for the North American market). A header plug leading to bare leads is provided to allow overseas users to connect their own power supply with a DC output of 7.5–30 V.

- Programming cable with 10-pin header and DB9 connections, and integrated level-matching circuitry.

- Cable kits to access RS-485 and analog input connectors on Prototyping Board.

- *Dynamic C* CD-ROM, with complete product documentation on disk.

- *Getting Started* instructions.

- Accessory parts for use on the Prototyping Board.

- *Rabbit 3000 Processor Easy Reference* poster.

- Registration card.
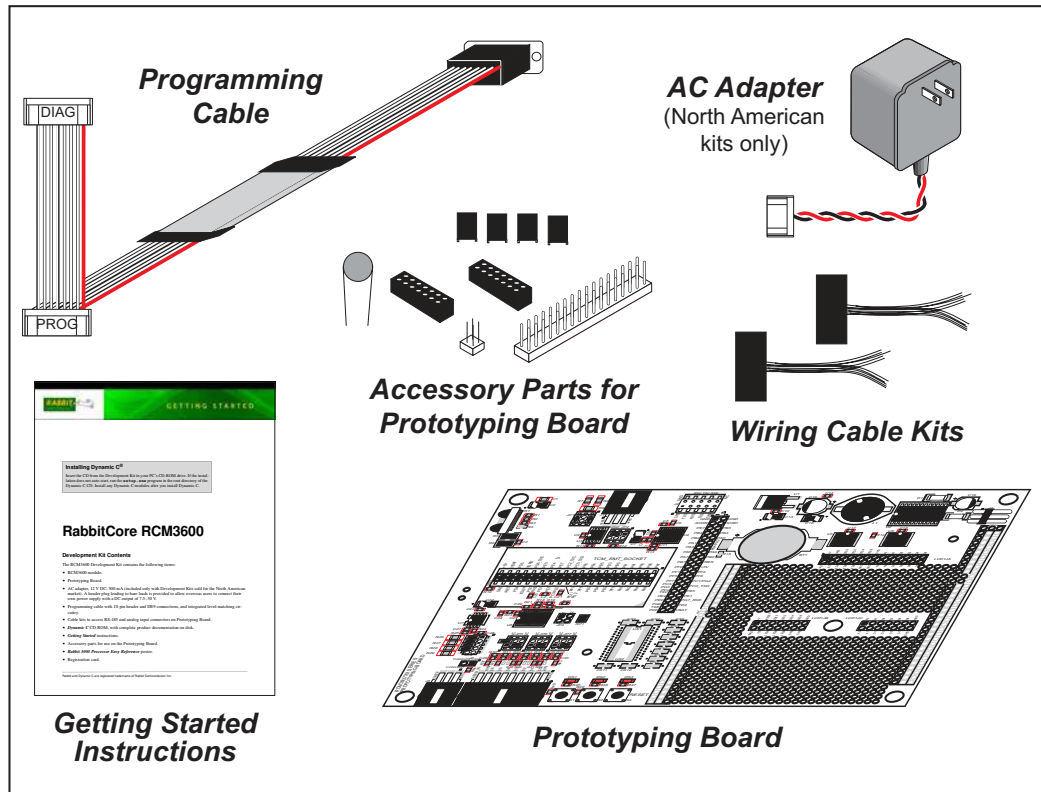


*Figure 1.  RCM3600 Development Kit*

### 1.3.2  Software

The RCM3600 is programmed using version 8.11 or later of Dynamic C. A compatible version is included on the Development Kit CD-ROM.

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

### 1.3.3  Connectivity Interface Kits

Rabbit Semiconductor has available an interface kit to allow you to provide a wireless interface to the RCM3600.

- 802.11b Wi-Fi Add-On Kit (Part No. 101-0999)—The Wi-Fi Add-On Kit for the RCM3600/RCM3700 footprint consists of an RCM3600/RCM3700 Interposer Board, a Wi-Fi CompactFlash card with a CompactFlash Wi-Fi Board, a ribbon interconnecting cable, and the software drivers and sample programs to help you enable your RCM3600 module with Wi-Fi capabilities. The RCM3600/RCM3700 Interposer Board is placed between the RCM3600 module and the Prototyping Board so that the CompactFlash Wi-Fi Board, which holds the Wi-Fi CompactFlash card, can be connected to the RCM3600-based system via the ribbon cable provided.

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

### 1.3.4  Online Documentation

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

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

# 2. GETTING STARTED

This chapter describes the RCM3600 hardware in more detail, and explains how to set up and use the accompanying Prototyping Board.

> **NOTE:** It is assumed that you have the RCM3600 Development Kit. If you purchased an RCM3600 module by itself or with another kit, you will have to adapt the information in this chapter and elsewhere to your test and development setup.

## 2.1 Install Dynamic C

To develop and debug programs for the RCM3600 (and for all other Rabbit Semiconductor hardware), you must install and use Dynamic C.

If you have not yet installed Dynamic C version 8.11 (or a later version), do so now by inserting the Dynamic C CD from the RCM3600 Development Kit in your PC's CD-ROM drive. If autorun is enabled, the CD installation will begin automatically.

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

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

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

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

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

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

## 2.2  Hardware Connections

There are three steps to prepare the RCM3600 for use with Dynamic C and the sample programs:

1.  Attach the RCM3600 module to the Prototyping Board.

2.  Connect the programming cable between the RCM3600 and the COM port on the workstation PC.

3.  Connect the power supply to the Prototyping Board.

### 2.2.1  Attach Module to Prototyping Board

Turn the RCM3600 module so that the Rabbit 3000 chip is facing up as shown in Figure 2 below. Insert the pins from the module's J1 header on the bottom side of the RCM3600 into the TCM_SMT_SOCKET socket on the Prototyping Board. The shaded corner notch at the bottom right corner of the RCM3600 module should face the same direction as the corresponding notch below it on the Prototyping Board.
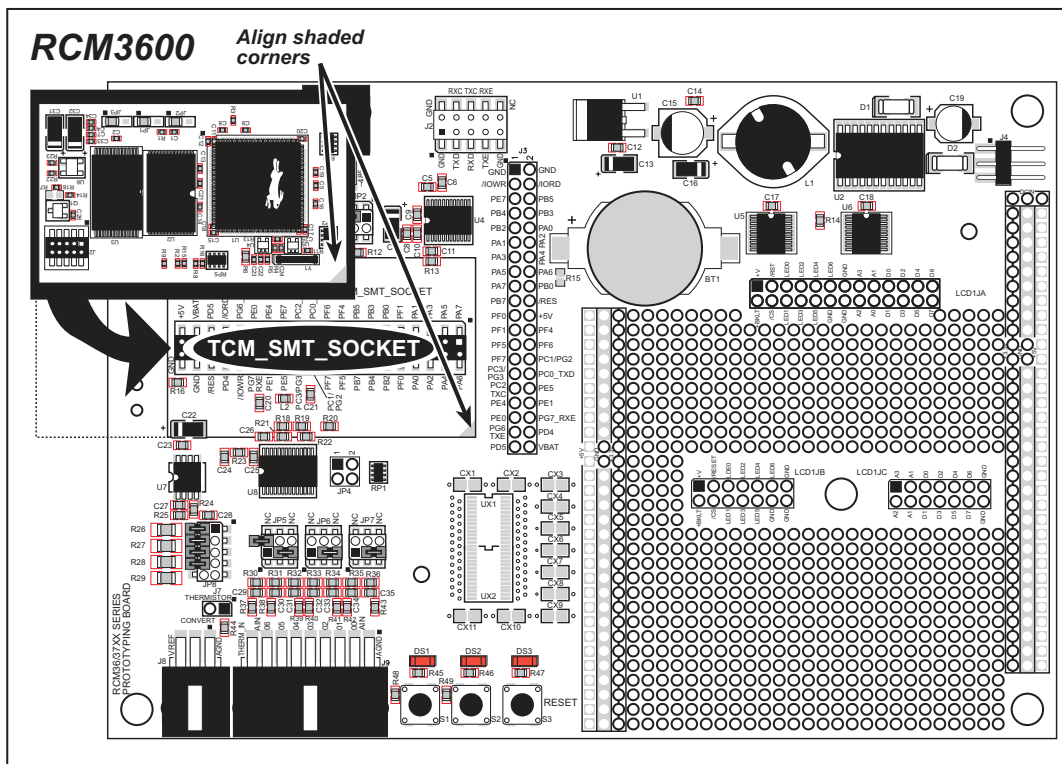


**Figure 2.  Install the RCM3600 Series on the Prototyping Board**

> **NOTE:**  It is important that you line up the pins on header J1 of the RCM3600 module exactly with the corresponding pins of the TCM_SMT_SOCKET socket 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 headers.

## 2.2.2 Connect Programming Cable

The programming cable connects the RCM3600 to the PC running Dynamic C to download programs and to monitor the RCM3600 module during debugging.

Connect the 10-pin connector of the programming cable labeled **PROG** to header J2 on the RCM3600 as shown in Figure 3. Be sure to orient the marked (usually red) edge of the cable towards pin 1 of the connector. (Do not use the **DIAG** connector, which is used for a normal serial connection.)
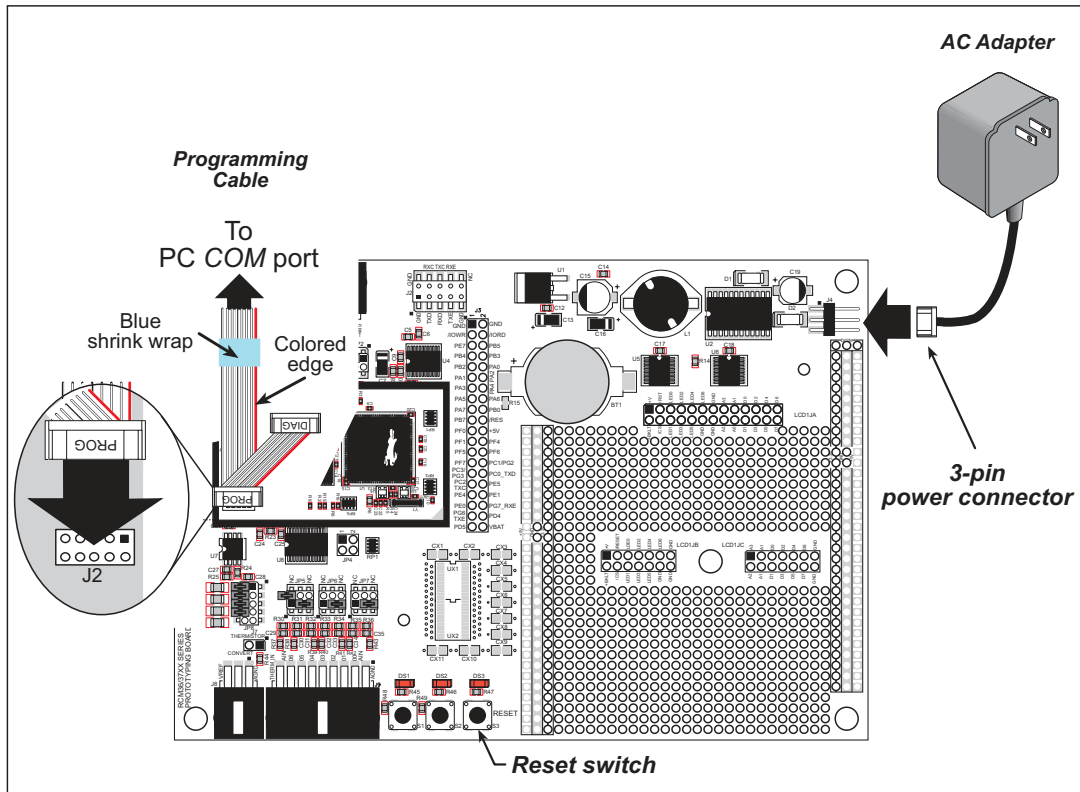


*Figure 3.  Connect Programming Cable and Power Supply*

> **NOTE:** Be sure to use the programming cable (Part No. 101-0542) supplied with this Development Kit—the programming cable has blue shrink wrap around the RS-232 converter section located in the middle of the cable. Programming cables from other Rabbit Semiconductor kits are not designed to work with RCM3600 modules.

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

> **NOTE:** Some PCs now come equipped only with a USB port. It may be possible to use an RS-232/USB converter (Part No. 540-0070) with the programming cable supplied with the RCM3600 Development Kit. Note that not all RS-232/USB converters work with Dynamic C.

### 2.2.3  Connect Power

When all other connections have been made, connect the wall transformer to 3-pin header J4 on the Prototyping Board as shown in Figure 3. The connector may be attached either way as long as it is not offset to one side.

Plug in the wall transformer. The LED above the **RESET** button on the Prototyping Board should light up. The RCM3600 and the Prototyping Board are now ready to be used.

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

#### 2.2.3.1  Overseas Development Kits

Development kits sold outside North America include a header connector that may be connected to 3-pin header J4 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 7.5 V–30 V DC at 500 mA.

## 2.3  Starting Dynamic C

Once the RCM3600 is connected as described in the preceding pages, start Dynamic C by double-clicking on the Dynamic C icon or by double-clicking on `dcrabXXXX.exe` in the Dynamic C root directory, where `XXXX` are version-specific characters. Dynamic C uses the serial COM port on your PC that you specified during installation.

If you are using a USB port to connect your computer to the RCM3600 module, choose **Options > Project Options** and select "Use USB to Serial Converter."

## 2.4  Run a Sample Program

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

### 2.4.1  Troubleshooting

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 that the RCM3600 is powered correctly — the power LED above the **RESET** button on the Prototyping Board should be lit.

- 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 RCM3600.

- Ensure that the RCM3600 module is firmly and correctly installed in its connectors on the Prototyping Board.

- Select a different COM port within Dynamic C. From the **Options** menu, select **Project Options**, then select **Communications**. Select another COM port from the list, then click OK. Press **<Ctrl-Y>** to force Dynamic C to recompile the BIOS. If Dynamic C still reports it is unable to locate the target system, repeat the above steps until you locate the active COM port.

## 2.5  Where Do I Go From Here?

If the sample program ran fine, you are now ready to go on to other sample programs and to develop your own applications. The source code for the sample programs is provided to allow you to modify them for your own use. The *RCM3600 User's Manual* also provides complete hardware reference information and describes the software function calls for the RCM3600, the Prototyping Board, and the optional LCD/keypad module.

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

### 2.5.1  Technical Support

> **NOTE:**  If you purchased your RCM3600 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 RCM3600 (and for all other Rabbit Semiconductor hardware), you must install and use Dynamic C.

## 3.1 Introduction

To help familiarize you with the RCM3600 modules, Dynamic C includes several sample programs. Loading, executing and studying these programs will give you a solid hands-on overview of the RCM3600'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.

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

1. Your RCM3600 must be plugged in to the Prototyping Board as described in Chapter 2, "Getting Started."

2. Dynamic C must be installed and running on your PC.

3. The programming cable must connect the programming header (J2) on the RCM3600 to your PC.

4. Power must be applied to the RCM3600 through the Prototyping Board.

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

To run a sample program, open it with the **File** menu, then compile and run it by pressing **F9** or by selecting **Run** in the **Run** menu. The RCM3600 must be in Program Mode (see Figure 8) and must be connected to a PC using the programming cable.
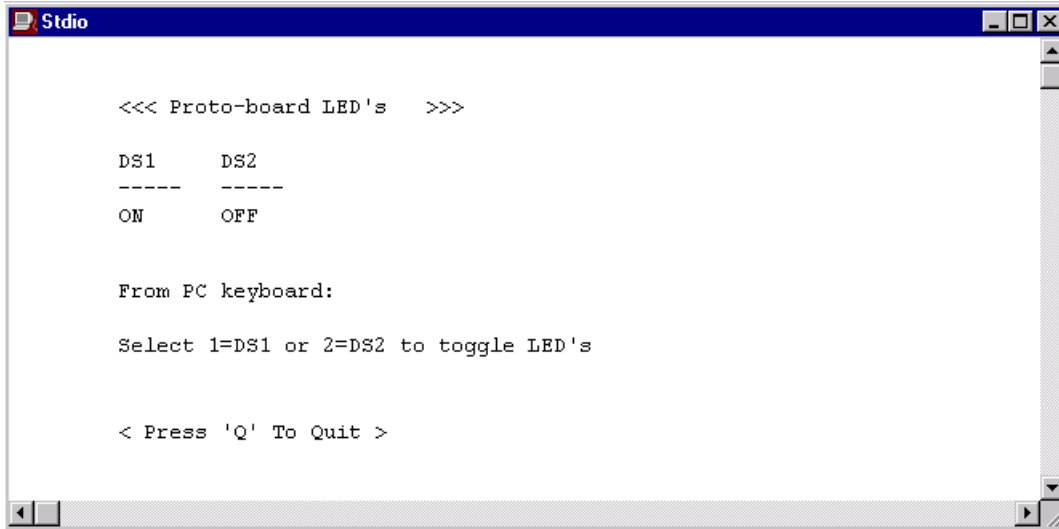
Complete information on Dynamic C is provided in the *Dynamic C User's Manual*.

## 3.2  Sample Programs

Of the many sample programs included with Dynamic C, several are specific to the RCM3600. Sample programs illustrating the general operation of the RCM3600, serial communication, and the A/D converter on the Prototyping Board are provided in the `SAMPLES\RCM3600` folder. Each sample program has comments that describe the purpose and function of the program. Follow the instructions at the beginning of the sample program. Note that the RCM3600 must be installed on the Prototyping Board when using these sample programs. Sample programs for the optional LCD/keypad module are described in Appendix C.

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

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

```
Stdio                                                    _ □ ×

        <<< Proto-board LED's    >>>

        DS1      DS2
        -----    -----
        ON       OFF


        From PC keyboard:

        Select 1=DS1 or 2=DS2 to toggle LED's


        < Press 'Q' To Quit >
```

  Press "1" or "2" on your keyboard to select LED DS1 or DS2 on the Prototyping Board. Then follow the prompt in the Dynamic C **STDIO** window to turn the LED on  or  off.

- `FLASHLED.c`—Demonstrates the use of assembly language 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.

- `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 requires a second Prototyping Board or Rabbit Semiconductor single-board computer that has an IrDA chip and is running the `IR_DEMO.C` sample program associated with it.

  First, compile and run the `IR_DEMO.C` sample program from the `SAMPLES` folder specific to the other system on the second system, then remove the programming cable and press the **RESET** button so that the first assembly is operating in the **Run** mode. Then connect the programming cable to the RCM3600 module, and compile and run the `IR_DEMO.C` sample program from the `SAMPLES\RCM3600` folder on the RCM3600 system. With the two IrDA transceivers facing each other, press switch S1 on the RCM3600 Prototyping Board to transmit a packet. The other system will return a response packet that will then appear in the Dynamic C **STDIO** window. The test packets and response packets have different codes.

- `DIO.c`—Demonstrates the digital I/O capabilities of the A/D converter on the Prototyping Board by configuring two lines to outputs and two lines as inputs on Prototyping Board header JP4.

  Install a 2 × 2 header at JP4 on the Prototyping Board and connect pins 1–3 and pins 2–4 on header JP4 before running this sample program.

  Once the sample program is compiled and running, it will prompt you in the **STDIO** window to select either pin 1 of header JP4 or pin 2 of header JP4 for the output. Once you have made that selection, you will be prompted to enter a logic 0 or 1. The specified logic level will then be output on pins 1–3 or pins 2–4 on header JP4.

- `TOGGLESWITCH.c`—Uses costatements to detect switches using debouncing. The corresponding LEDs (DS1 and DS2) will turn on or off. LEDs DS1 and DS2 on the Prototyping Board are turned on and off when you press switches S1 and S2. S1 and S2 are controlled by PF4 and PB7 respectively.

Once you have loaded and executed these five programs and have an understanding of how Dynamic C and the RCM3600 modules interact, you can move on and try the other sample programs, or begin developing your own.

### 3.2.1 Serial Communication

The following sample programs can be found in the Dynamic C `SAMPLES\RCM3600\`
`SERIAL` folder.

> **NOTE:** PE5 is set up to enable/disable the RS-232 chip on the Prototyping Board. This pin will also be toggled when you run RS-232 sample programs on the Prototyping Board. If you plan to use this pin for something else while you are running any of the RS-232 sample programs, comment out the following line.
>
> `BitWrPortI(PEDR, &PEDRShadow, 0, 5);//set low to enable rs232 device`

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

  To set up the Prototyping Board, you will need to tie TxD and RxD together on the RS-232 header at J2, and you will also tie TxC and RxC together using the jumpers supplied in the Development Kit as shown in the diagram.
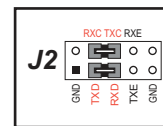
  A repeating triangular pattern should print out in the **STDIO** window.
  The program will periodically switch flow control on or off to demonstrate the effect of no flow control.

  Refer to the function description for `serDflowcontrolOn()` in the *Dynamic C Function Reference Manual* for a general description on how to set up flow-control lines.

- **`PARITY.C`**—This program demonstrates the use of parity modes by repeatedly sending byte values 0–127 from Serial Port D to Serial Port C. The program will switch between generating parity or not on Serial Port D. Serial Port C will always be checking parity, so parity errors should occur during every other sequence.
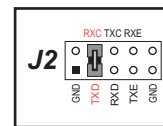
  To set up the Prototyping Board, you will need to tie TxD and RxC together on the RS-232 header at J2 using the 0.1" jumpers supplied in the Development Kit as shown in the diagram.

  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 RxD. The characters are converted to upper case and are sent out by TxD, are received by RxC, and are displayed in the Dynamic C **STDIO** window.
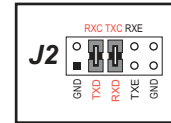
  To set up the Prototyping Board, you will need to tie TxD and RxC together on the RS-232 header at J2, and you will also tie RxD and TxC together using the 0.1" jumpers supplied in the Development Kit as shown in the diagram.

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

  To set up the Prototyping Board, you will need to tie TxD and RxD together on the RS-232 header at J2, and you will also tie TxC and RxC together using the 0.1" jumpers supplied in the Development Kit as shown in the diagram.
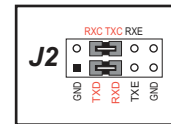
  Once you have compiled and run this program, you can test flow control by disconnecting TxC from RxC while the program is running. Characters will no longer appear in the **STDIO** window, and will display again once TxC is connected back to RxC.

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

  Before running this sample program, check to make sure that Serial Port E is set up as an RS-232 serial port—pins 1–3 and pins 2–4 on header JP2 must be jumpered together using the 2 mm jumpers supplied in the Development Kit. Then connect TxC to RxE and connect RxC to TxE on the RS-232 header at J2 using the 0.1" jumpers supplied in the Development Kit as shown in the diagram.
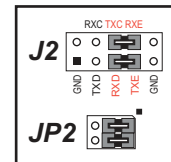
  > **NOTE:** The following two sample programs illustrating RS-485 serial communication will only work with the RCM3600/RCM3700 Prototyping Board.

- **SIMPLE485MASTER.C**—This program demonstrates a simple RS-485 transmission of lower case letters to a slave RCM3600. The slave will send back converted upper case letters back to the master RCM3600 and display them in the **STDIO** window. Use **SIMPLE485SLAVE.C** to program the slave RCM3600, and check to make sure that Serial Port E is set up as an RS-485 serial port—pins 3–5 and pins 4–6 on header JP2 must be jumpered together using the 2 mm jumpers supplied in the Development Kit.

- **SIMPLE485SLAVE.C**—This program demonstrates a simple RS-485 transmission of lower case letters to a master RCM3600. The slave will send back converted upper case letters back to the master RCM3600 and display them in the **STDIO** window. Use **SIMPLE485MASTER.C** to program the master RCM3600, and check to make sure that Serial Port E is set up as an RS-485 serial port—pins 3–5 and pins 4–6 on header JP2 must be jumpered together using the 2 mm jumpers supplied in the Development Kit.

### 3.2.2 A/D Converter Inputs

The following sample programs are found in the Dynamic C `SAMPLES\RCM3600\ADC` folder.

- `AD_CALDIFF_CH.C`—Demonstrates how to recalibrate one differential analog input channel using two known voltages to generate the calibration constants for that channel. Constants will be rewritten into user block data area.

  Before running this program, make sure that pins 1–3 are connected on headers JP5, JP6, and JP7 on the Prototyping Board. No pins are connected on header JP8.

- `AD_CALMA_CH.C`—Demonstrates how to recalibrate an A/D input channel being used to convert analog current measurements to generate the calibration constants for that channel.

  Before running this program, make sure that pins 3–5 are connected on headers JP5, JP6, and JP7 on the Prototyping Board. Connect pins 1–2, 3–4, 5–6, 7–8 on header JP8.

- `AD_CALSE_ALL.C`—Demonstrates how to recalibrate all single-ended analog input channels for one gain, using two known voltages to generate the calibration constants for each channel. Constants will be rewritten into the user block data area.

  Before running this program, make sure that pins 3–5 are connected on headers JP5, JP6, and JP7 on the Prototyping Board. No pins are connected on header JP8.

- `AD_CALSE_CHAN.C`—Demonstrates how to recalibrate one single-ended analog input channel with one gain using two known voltages to generate the calibration constants for that channel. Constants will be rewritten into user block data area.

  Before running this program, make sure that pins 3–5 are connected on headers JP5, JP6, and JP7 on the Prototyping Board. No pins are connected on header JP8.

  > **NOTE:** The above sample programs will overwrite any existing calibration constants.

- `AD_RDDIFF_CH.C`—Demonstrates how to read an A/D input channel being used for a differential input using previously defined calibration constants.

  Before running this program, make sure that pins 1–3 are connected on headers JP5, JP6, and JP7 on the Prototyping Board. No pins are connected on header JP8.

- `AD_RDMA_CH.C`—Demonstrates how to read an A/D input channel being used to convert analog current measurements using previously defined calibration constants for that channel.

  Before running this program, make sure that pins 3–5 are connected on headers JP5, JP6, and JP7 on the Prototyping Board. Connect pins 1–2, 3–4, 5–6, 7–8 on header JP8.

- `AD_RDSE_ALL.C`—Demonstrates how to read all single-ended A/D input channels using previously defined calibration constants.

  Before running this program, make sure that pins 3–5 are connected on headers JP5, JP6, and JP7 on the Prototyping Board. No pins are connected on header JP8.

- **AD_SAMPLE.C**—Demonstrates how to use a low-level driver on single-ended inputs. The program will continuously display the voltage (average of 10 samples) that is present on the A/D channels.

  Before running this program, make sure that pins 3–5 are connected on headers JP5, JP6, and JP7 on the Prototyping Board. No pins are connected on header JP8.

- **ANAINCONFIG.C**—Demonstrates how to use the Register Mode method to read single-ended analog input values for display as voltages. The sample program uses the function call **anaInConfig()** and the ADS7870 CONVERT line to accomplish this task.

  Before running this program, make sure that pins 3–5 are connected on headers JP5, JP6, and JP7 on the Prototyping Board. No pins are connected on header JP8. Also connect PE4 on header J3 on the Prototyping Board to the CNVRT terminal on header J8; if you are using this sample program as a template for your own program, be aware that PE4 is also used as the IrDA FIR_SEL pin.

- **THERMISTOR.C**—Demonstrates how to use analog input THERM_IN7 to calculate temperature for display to the **STDIO** window. This sample program assumes that the thermistor is the one included in the Development Kit whose values for beta, series resistance, and resistance at standard temperature are given in the part specification.

  Before running this program, install the thermistor into the AIN7 and AGND holes at location J7 on the Prototyping Board.

Before running the next two sample programs, **DNLOADCALIB.C** or **UPLOADCALIB.C**, connect your PC serial COM port to header J2 on the Prototyping Board as follows.

- Tx to RxE

- Rx to TxE

- GND to GND

Then connect pins 1–3 and 2–4 on header JP2 on the Prototyping Board.

Now start Tera Term on your PC. Once Tera Term is running, configure the serial parameters as follows:

- Baud rate 19200, 8 bits, no parity, and 1 stop bit.

- Enable the "Local Echo" option.

- Set the line feed options to Receive = CR and Transmit = CR + LF.

Now press **F9** to compile and run this program. Verify that the message "Waiting, Please Send Data file" is being display in Tera Term display window before proceeding. From within Tera Term, select **File > Send File > Path and filename**, then select the OPEN option within the dialog box. Once the data file has been downloaded, it will indicate whether the calibration data were written successfully.

- **DNLOADCALIB.C**—Demonstrates how to retrieve analog calibration data to rewrite it back to simulated EEPROM in flash with using a serial utility such as Tera Term.

- **UPLOADCALIB.C**—Demonstrates how to read calibrations constants from the user block in flash memory and then transmit the file using a serial port and a PC serial utility such as Tera Term. Use **DNLOADCALIB.C** to download the calibration constants created by this program.

# 4. HARDWARE REFERENCE

Chapter 4 describes the hardware components and principal hardware subsystems of the RCM3600. Appendix A, "RCM3600 Specifications," provides complete physical and electrical specifications.

Figure 4 shows the Rabbit-based subsystems designed into the RCM3600.
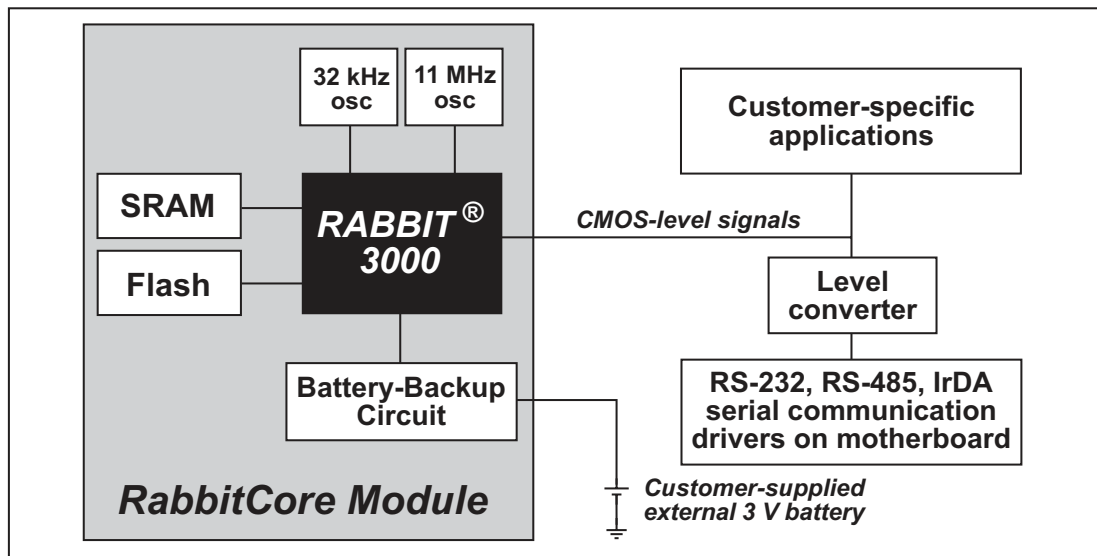


**Figure 4. RCM3600 Subsystems**

## 4.1  RCM3600 Digital Inputs and Outputs

Figure 5 shows the RCM3600 pinouts for header J1.



**J1**

| | | |
|---|---|---|
| PA6 | □ ■ | PA7 |
| PA4 | □ □ | PA5 |
| PA2 | □ □ | PA3 |
| PA0 | □ □ | PA1 |
| PF0 | □ □ | PF1 |
| PB2 | □ □ | PB0 |
| PB4 | □ □ | PB3 |
| PB7 | □ □ | PB5 |
| PF5 | □ □ | PF4 |
| PF7 | □ □ | PF6 |
| PC1/PG2 | □ □ | PC0 |
| PC3/PG3 | □ □ | PC2 |
| PE5 | □ □ | PE7 |
| PE1 | □ □ | PE4 |
| PG7 | □ □ | PE0 |
| /IOWR | □ □ | PG6 |
| PD4 | □ □ | /IORD |
| /RES | □ □ | PD5 |
| GND | □ □ | VBAT_EXT |
| GND | □ □ | VIN |

*Note:* These pinouts are as seen on
the **Bottom Side** of the module.
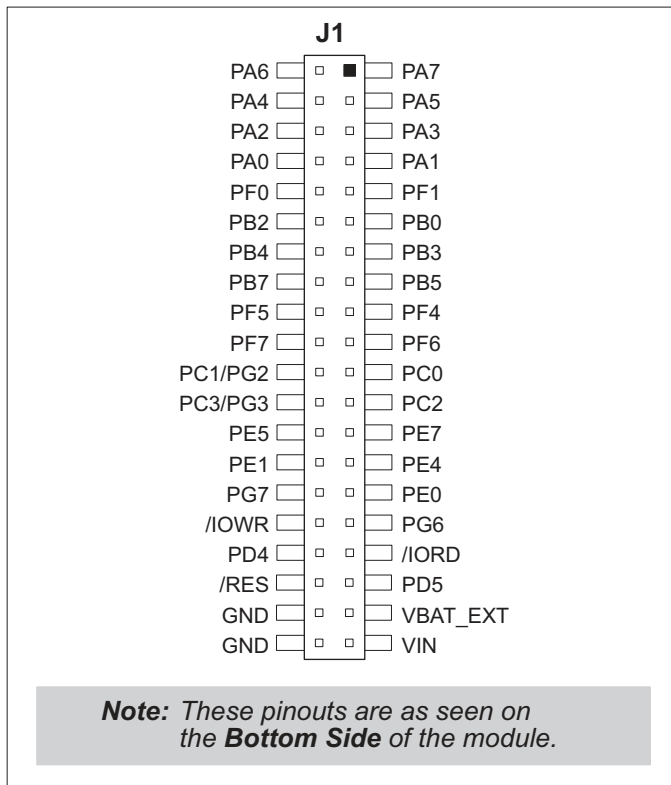
*Figure 5.  RCM3600 Pinouts*

Header J1 is a standard 2 x 20 IDC header with a nominal 0.1" pitch.

Figure 6 shows the use of the Rabbit 3000 microprocessor ports in the RCM3600 modules.
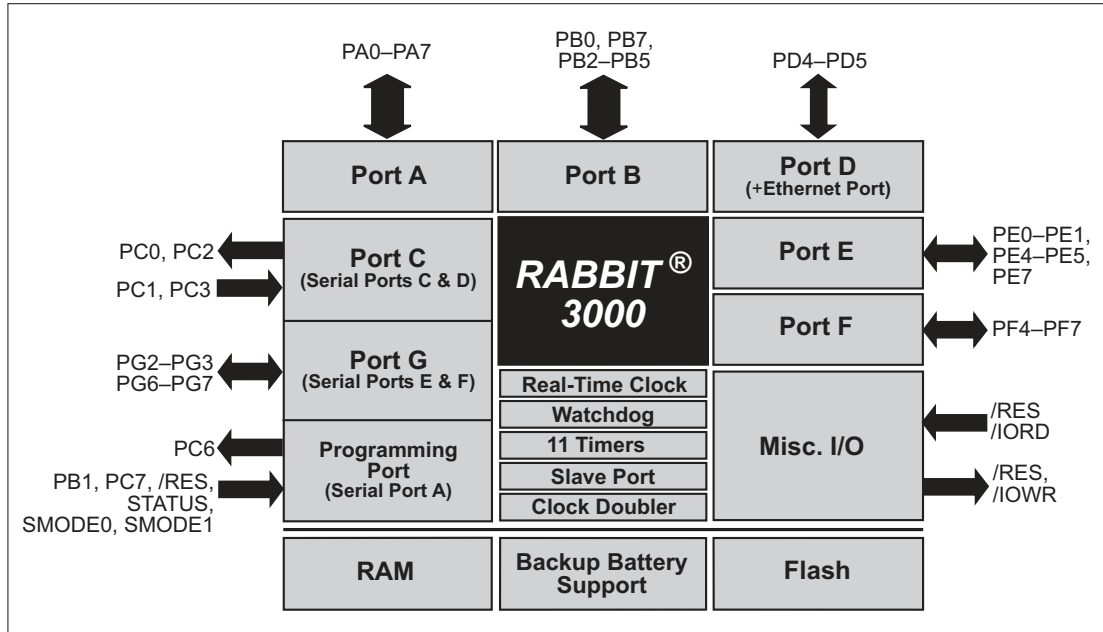


*Figure 6.  Use of Rabbit 3000 Ports*

The ports on the Rabbit 3000 microprocessor used in the RCM3600 are configurable, and so the factory defaults can be reconfigured. Table 2 lists the Rabbit 3000 factory defaults and the alternate configurations.

### Table 2.  RCM3600 Pinout Configurations

| Pin | | Pin Name | Default Use | Alternate Use | Notes |
|---|---|---|---|---|---|
| Header J1 | 1–8 | PA[7:0] | Parallel I/O | External data bus (ID0–ID7) Slave port data bus (SD0–SD7) | External Data Bus |
| | 9 | PF1 | Input/Output | QD1A CLKC | |
| | 10 | PF0 | Input/Output | QD1B CLKD | |
| | 11 | PB0 | Input/Output | CLKB | |
| | 12 | PB2 | Input/Output | IA0 /SWR | External Address 0 Slave port write |
| | 13 | PB3 | Input/Output | IA1 /SRD | External Address 1 Slave port read |
| | 14 | PB4 | Input/Output | IA2 SA0 | External Address 2 Slave Port Address 0 |
| | 15 | PB5 | Input/Output | IA3 SA1 | External Address 3 Slave Port Address 1 |
| | 16 | PB7 | Input/Output | IA5 /SLAVEATTN | External Address 5 Slave Port Attention |
| | 17 | PF4 | Input/Output | AQD1B PWM0 | |
| | 18 | PF5 | Input/Output | AQD1A PWM1 | |
| | 19 | PF6 | Input/Output | AQD2B PWM2 | |
| | 20 | PF7 | Input/Output | AQD2A PWM3 | |
| | 21 | PC0 | Output | TXD | Serial Port D |
| | 22 | PC1/PG2 | Input/Output | RXD/TXF | Serial Port D Serial Port F |
| | 23 | PC2 | Output | TXC | Serial Port C |
| | 24 | PC3/PG3 | Input/Output | RXC/RXF | Serial Port C Serial Port F |
| | 25 | PE7 | Input/Output | I7 /SCS | I/O Strobe 7 Slave Port Chip Select |

**Table 2. RCM3600 Pinout Configurations (continued)**

| | Pin | Pin Name | Default Use | Alternate Use | Notes |
|---|---|---|---|---|---|
| Header J1 | 26 | PE5 | Input/Output | I5<br>INT1B | I/O Strobe 5<br>Interrupt 1B |
| | 27 | PE4 | Input/Output | I4<br>INT0B | I/O Strobe 4<br>Interrupt 0B |
| | 28 | PE1 | Input/Output | I1<br>INT1A | I/O Strobe 1<br>Interrupt 1A |
| | 29 | PE0 | Input/Output | I0<br>INT0A | I/O Strobe 0<br>Interrupt 0A |
| | 30 | PG7 | Input/Output | RXE | Serial Port E |
| | 31 | PG6 | Input/Output | TXE | |
| | 32 | /IOWR | Output | | External write strobe |
| | 33 | /IORD | Input | | External read strobe |
| | 34 | PD4 | Input/Output | ATXB | Alternate Serial Port B |
| | 35 | PD5 | Input/Output | ARXB | |
| | 36 | /RES | Reset output | Reset input | Reset output from Reset Generator |
| | 37 | VBAT | | | |
| | 38 | GND | | | |
| | 39 | +5 V | | | |
| | 40 | GND | | | |

### 4.1.1  Memory I/O Interface

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

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

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

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

### 4.1.2  Other Inputs and Outputs

/RES is an output from the reset circuitry that can be used to reset other peripheral devices. This pin can also be used to reset the microprocessor.

## 4.2 Serial Communication

The RCM3600 board does not have any serial transceivers directly on the board. However, a serial interface may be incorporated on the board the RCM3600 is mounted on. For example, the Prototyping Board has RS-232, RS-485 and IrDA transceiver chips.

### 4.2.1 Serial Ports

There are five serial ports designated as Serial Ports A, C, D, E, and F. All five serial ports can operate in an asynchronous mode up to the baud rate of the system clock divided by 8. An asynchronous port can handle 7 or 8 data bits. A 9th bit address scheme, where an additional bit is sent to mark the first byte of a message, is also supported.

Serial Port A is normally used as a programming port, but may be used either as an asynchronous or as a clocked serial port once application development has been completed and the RCM3600 is operating in the Run Mode.

Serial Ports C and D can also be operated in the clocked serial mode. In this mode, a clock line synchronously clocks the data in or out. Either of the two communicating devices can supply the clock.

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

Serial Port F shares its pins with Serial Ports C and D on header J1, as shown in Figure 7. The selection of port(s) depends on your need for two clocked serial ports (Serial Ports C and D) vs. a second HDLC serial port (Serial Port F).
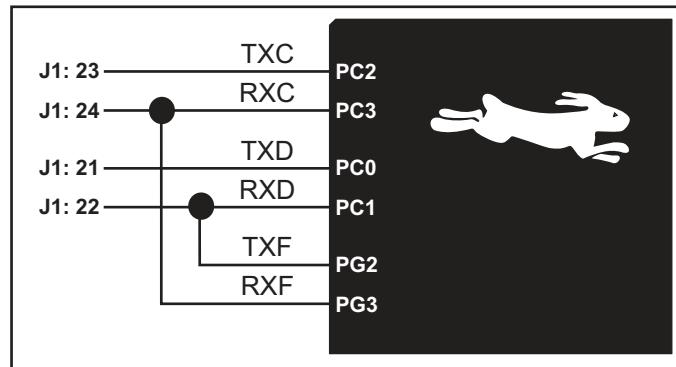


*Figure 7. RCM3600 Serial Ports C, D, and F*

The serial ports used are selected with the **serXOpen** function call, where X is the serial port (C, D, or F). Remember that the RxC and RxD on Serial Ports C and D cannot be used if Serial Port F is being used

### 4.2.2  Serial Programming Port

The RCM3600 programming port is accessed through header J2 or over an Ethernet connection via the RabbitLink EG2110. The programming port uses the Rabbit 3000's Serial Port A for communication. Dynamic C uses the programming port to download and debug programs.

The programming port is also used for the following operations.

- Cold-boot the Rabbit 3000 on the RCM3600 after a reset.

- Remotely download and debug a program over an Ethernet connection using the RabbitLink EG2110.

- Fast copy designated portions of flash memory from one Rabbit-based board (the master) to another (the slave) using the Rabbit Cloning Board.

**Alternate Uses of the Programming Port**

All three clocked Serial Port A signals are available as

- a synchronous serial port

- an asynchronous serial port, with the clock line usable as a general CMOS 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 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 CMOS output.

The reset pin is an external input that is used to reset the Rabbit 3000. The serial programming port can be used to force a hard reset on the RCM3600 by asserting the reset 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 programming port of the RCM3600 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 RCM3600 programming port, programs can be downloaded and debugged over the serial interface.

The **DIAG** connector of the programming cable may be used on header J2 of the RCM3600 with the RCM3600 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 RCM3600 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.
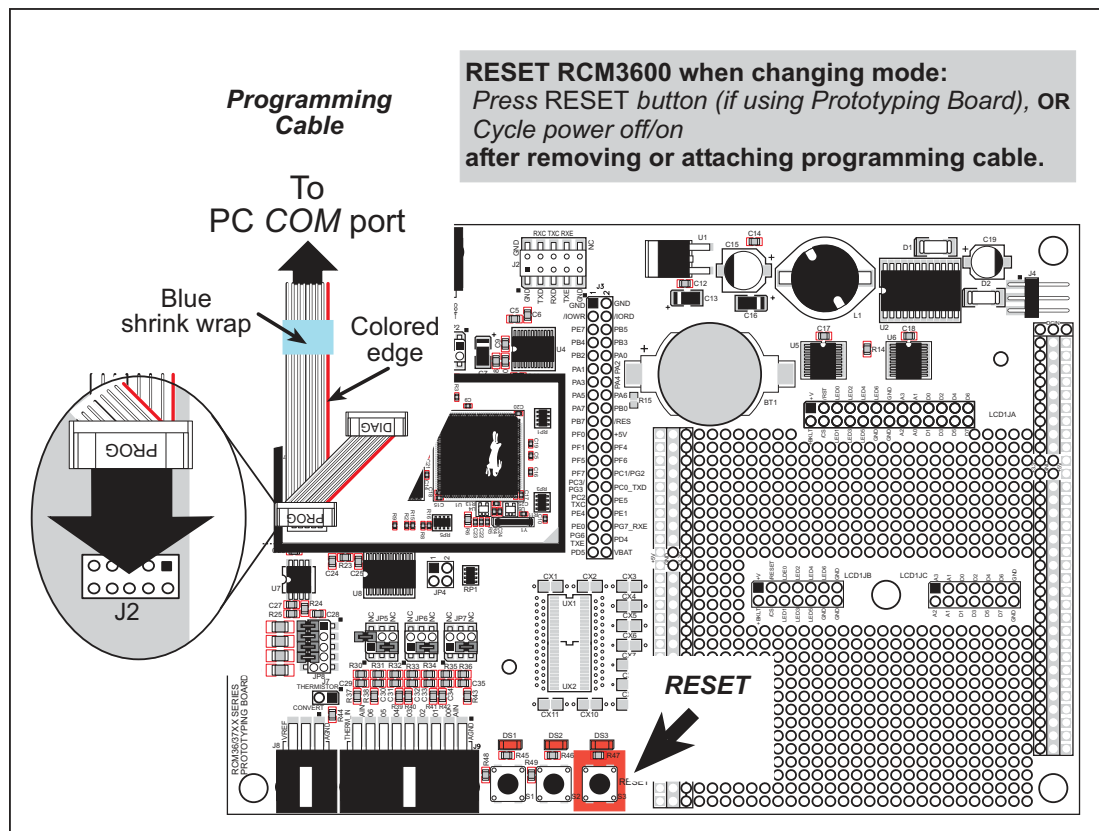


*Figure 8.  Switching Between Program Mode and Run Mode*

A program "runs" in either mode, but can only be downloaded and debugged when the RCM3600 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 RCM3600

The RCM3600 must be programmed via the RCM3600 Prototyping Board or via a similar arrangement on a customer-supplied board. Once the RCM3600 has been programmed successfully, remove the programming cable from the programming connector and reset the RCM3600. The RCM3600 may be reset by cycling the power off/on or by pressing the **RESET** button on the Prototyping Board. The RCM3600 module may now be removed from the Prototyping Board for end-use installation.

> **CAUTION:**  Power to the Prototyping Board or other boards should be disconnected when removing or installing your RCM3600 module to protect against inadvertent shorts across the pins or damage to the RCM3600 if the pins are not plugged in correctly. Do not reapply power until you have verified that the RCM3600 module is plugged in correctly.

## 4.4  Other Hardware

### 4.4.1  Clock Doubler

The RCM3600 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 22.1 MHz frequency specified for the RCM3600 is generated using a 11.06  MHz resonator.

The clock doubler may be disabled if 22.1 MHz clock speeds are not required. This 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 not recommended since it may limit the maximum clock speed or the maximum baud rate. It is unlikely that the strong setting will be used in a real application.

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

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

## 4.5  Memory

### 4.5.1  SRAM

RCM3600 series boards have 256K–512K of SRAM.

### 4.5.2  Flash EPROM

RCM3600 series boards also have 256K–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, use a portion of the "user block" area to store persistent data. The function calls **writeUser-Block** and **readUserBlock** are provided for this. Refer to the ***Rabbit 3000 Microprocessor Designer's Handbook*** for additional information.

A Flash Memory Bank Select jumper configuration option based on 0  Ω surface-mounted resistors exists at header JP1 on the RCM3600 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 512K 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 TN218, ***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 single-board computers and other single-board computers based on the Rabbit microprocessor. Chapter 5 describes the libraries and function calls related to the RCM3600.

## 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* and in the *Dynamic C Function Reference Manual*.

You have a choice of doing your software development in the flash memory or in the SRAM included on the RCM3600. The flash memory and SRAM options are selected with the **Options > Compiler** menu.

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

> **NOTE:**  An application can be compiled 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 in your program logic. The RCM3600 and Dynamic C were designed to accommodate flash devices with various sector sizes in response to the volatility of the flash-memory market.

Developing software with Dynamic C is simple. Users can write, compile, and test C and assembly code without leaving the Dynamic C development environment. Debugging occurs while the application runs on the target. Alternatively, users can compile a program to an image file for later loading. Dynamic C runs on PCs under Windows 95 and 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. Some of these standard features are listed below.

- Full-feature source and 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 (Rabbit Field Utility) to load binary images to Rabbit-based targets without the presence 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 Functions

The functions described in this section are for use with the Prototyping Board features. The source code is in the `RCM36xx.LIB` library in the Dynamic C `SAMPLES\RCM3600` 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*.

### 5.2.1 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 RCM3600 module and its Prototyping Board.

**Summary of Initialization**

1. I/O port pins are configured for Prototyping Board operation.
2. Unused configurable I/O are set as tied inputs or outputs.
3. The LCD/keypad module is disabled.
4. RS-485 is not enabled.
5. RS-232 is not enabled.
6. The IrDA transceiver is disabled.
7. LEDs are off.
8. The A/D converter is reset and SCLKB is to 57,600 bps.
9. The A/D converter calibration constants are read (this function cannot run in RAM).

   **CAUTION:** Pin PB7 is connected as both switch S2 and as an external I/O bus on the Prototyping Board. Do not use S2 when the LCD/keypad module is installed.

   **CAUTION:** Pins PC1 and PG2 are tied together, and pins PC3 and PG3 are tied together. Both pairs of pins are connected to the IrDA transceiver and to the RS-232 transceiver via serial ports on the Prototyping Board. Do *not* enable both transceivers on the Prototyping Board at the same time.

**RETURN VALUE**

None.

## 5.2.2  Analog Inputs

```
unsigned int anaInConfig(unsigned int
    instructionbyte, unsigned int cmd, long baud);
```

Use this function to configure the ADS7870 A/D converter. This function will address the ADS7870 in Register Mode only, and will return error if you try the Direct Mode. Section B.4.2 provides additional addressing and command information for the ADS7870 A/D converter.

| ADS7870 Signal | ADS7870 State | RCM3600 Function/State |
|:---:|:---:|:---|
| LN0 | Input | AIN0 |
| LN1 | Input | AIN1 |
| LN2 | Input | AIN2 |
| LN3 | Input | AIN3 |
| LN4 | Input | AIN4 |
| LN5 | Input | AIN5 |
| LN6 | Input | AIN6 |
| LN7 | Input | AIN7 |
| /RESET | Input | Board reset device |
| RISE/FALL | Input | Pulled up for SCLK active on rising edge |
| PIO0 | Input | Pulled down |
| PIO1 | Input | Pulled down |
| PIO2 | Input | Pulled down |
| PIO3 | Input | Pulled down |
| CONVERT | Input | Pulled down |
| BUSY | Output | PD1 pulled down; logic high state converter is busy |
| CCLKCNTRL | Input | Pulled down; 0 state sets CCLK as input |
| CCLK | Input | Pulled down; external conversion clock |
| SCLK | Input | PB0; serial data transfer clock |
| SDI | Input | PD4; 3-wire mode for serial data input |
| SDO | Output | PD5; serial data output /CS driven |
| /CS | Input | PD2 pulled up; active-low enables serial interface |
| BUFIN | Input | Driven by VREF; reference buffer amplifier |
| VREF | Output | Connected to BUFIN |
| BUFOUT | Output | VREF output |

**PARAMETERS**

**`instructionbyte`** is the instruction byte that will initiate a read or write operation at 8 or 16 bits on the designated register address. For example,

```
checkid = anaInConfig(0x5F, 0, 9600);  // read ID and set baud rate
```

**`cmd`** refers to the command data that configure the registers addressed by the instruction byte. Enter 0 if you are performing a read operation. For example,

```
i = anaInConfig(0x07, 0x3b, 0);  // write ref/osc reg and enable
```

**`baud`** is the serial clock transfer rate of 9600 to 57,600 bps. **`baud`** must be set the first time this function is called. Enter 0 for this parameter thereafter, for example,

```
anaInConfig(0x00, 0x00, 9600);  // resets device and sets baud
```

**RETURN VALUE**

0 on write operations,
data value on read operations

**SEE ALSO**

```
anaInDriver, anaIn, brdInit
```

```
unsigned int anaInDriver(unsigned int cmd,
    unsigned int len);
```

Reads the voltage of an analog input channel by serial-clocking an 8-bit command to the ADS7870 A/D converter by the Direct Mode method. This function assumes that Mode1 (most significant byte first) and the A/D converter oscillator have been enabled. See **anaInConfig()** for the setup.

The conversion begins immediately after the last data bit has been transferred. An exception error will occur if Direct Mode bit D7 is not set.

**PARAMETERS**

**cmd** contains a gain code and a channel code as follows.

D7—1; D6–D4—Gain Code; D3–D0—Channel Code

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

```
cmd = 0x80 | (gain_code*16) + channel_code
```

| Gain Code | Multiplier |
|-----------|------------|
| 0 | x1 |
| 1 | x2 |
| 2 | x4 |
| 3 | x5 |
| 4 | x8 |
| 5 | x10 |
| 6 | x16 |
| 7 | x20 |

| Channel Code | Differential Input Lines | Channel Code | Single-Ended Input Lines* | 4–20 mA Lines |
|--------------|--------------------------|--------------|---------------------------|---------------|
| 0 | +AIN0 -AIN1 | 8 | AIN0 | AIN0* |
| 1 | +AIN2 -AIN3 | 9 | AIN1 | AIN1* |
| 2 | +AIN4 -AIN5 | 10 | AIN2 | AIN2* |
| 3[†] | +AIN6 -AIN7 | 11 | AIN3 | AIN3 |
| 4 | -AIN0 +AIN1 | 12 | AIN4 | AIN4 |
| 5 | -AIN2 +AIN3 | 13 | AIN5 | AIN5 |
| 6 | -AIN4 +AIN5 | 14 | AIN6 | AIN6 |
| 7* | -AIN6 +AIN7 | 15 | AIN7 | AIN7* |

    *   Negative input is ground.

    †   Not accessible on RCM3600 Prototyping Board

**len**, the output bit length, is always 12 for 11-bit conversions

**RETURN VALUE**

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

0–2047 for 11-bit conversions (bit 12 for sign)

-1 overflow or out of range

-2 conversion incomplete, busy bit timeout

**SEE ALSO**

`anaInConfig, anaIn, brdInit`

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

Reads the value of an analog input channel using the direct method of addressing the ADS7870 A/D converter. The A/D converter is enabled the first time this function is called—this will take approximately 1 second to ensure that the A/D converter capacitor is fully charged.

**PARAMETERS**

**channel** is the channel number (0 to 7) corresponding to ADC_IN0 to ADC_IN7

**opmode** is the mode of operation:

**SINGLE**—single-ended input
**DIFF**—differential input
**mAMP**—4–20 mA input

| channel | SINGLE | DIFF | mAMP |
|---------|--------|------|------|
| 0 | +AIN0 | +AIN0 -AIN1 | +AIN0* |
| 1 | +AIN1 | +AIN1 -AIN0* | +AIN1* |
| 2 | +AIN2 | +AIN2 -AIN3 | +AIN2* |
| 3 | +AIN3 | +AIN3 -AIN2* | +AIN3 |
| 4 | +AIN4 | +AIN4 -AIN5 | +AIN4 |
| 5 | +AIN5 | +AIN5 -AIN4* | +AIN5 |
| 6 | +AIN6 | +AIN6 -AIN7* | +AIN6 |
| 7 | +AIN7 | +AIN7 -AIN6* | +AIN7* |

\* Not accessible on RCM3600 Prototyping Board.

**gaincode** is the gain code of 0 to 7

| Gain Code | Multiplier | Voltage Range* (V) |
|-----------|------------|--------------------|
| 0 | x1 | 0–20 |
| 1 | x2 | 0–10 |
| 2 | x4 | 0–5 |
| 3 | x5 | 0–4 |
| 4 | x8 | 0–2.5 |
| 5 | x10 | 0–2 |
| 6 | x16 | 0–1.25 |
| 7 | x20 | 0–1 |

\* Applies to RCM3600 Prototyping Board.

**RETURN VALUE**

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

0–2047 for 11-bit A/D conversions (signed 12th bit)
**ADOVERFLOW** (defined macro = -4096) if overflow or out of range
-4095 if conversion is incomplete or busy-bit timeout

**SEE ALSO**

`anaIn, anaInConfig, anaInDriver`

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

Calibrates the response of the desired A/D converter channel as a linear function using the two conversion points provided. Four values are calculated and placed into global tables to be later stored into simulated EEPROM using the function `anaInEEWr()`. Each channel will have a linear constant and a voltage offset.

**PARAMETERS**

**channel** is the analog input channel number (0 to 7) corresponding to ADC_IN0 to ADC_IN7

**opmode** is the mode of operation:

**SINGLE**—single-ended input
**DIFF**—differential input
**mAMP**—milliamp input

| channel | SINGLE | DIFF | mAMP |
|---------|--------|------|------|
| 0 | +AIN0 | +AIN0 -AIN1 | +AIN0* |
| 1 | +AIN1 | +AIN1 -AIN0* | +AIN1* |
| 2 | +AIN2 | +AIN2 -AIN3 | +AIN2* |
| 3 | +AIN3 | +AIN3 -AIN2* | +AIN3 |
| 4 | +AIN4 | +AIN4 -AIN5 | +AIN4 |
| 5 | +AIN5 | +AIN5 -AIN4* | +AIN5 |
| 6 | +AIN6 | +AIN6 -AIN7* | +AIN6 |
| 7 | +AIN7 | +AIN7 -AIN6* | +AIN7* |

    \*  Not accessible on Prototyping Board.

**gaincode** is the gain code of 0 to 7

| Gain Code | Multiplier | Voltage Range* (V) |
|-----------|------------|--------------------|
| 0 | x1 | 0–20 |
| 1 | x2 | 0–10 |
| 2 | x4 | 0–5 |
| 3 | x5 | 0–4 |
| 4 | x8 | 0–2.5 |
| 5 | x10 | 0–2 |
| 6 | x16 | 0–1.25 |
| 7 | x20 | 0–1 |

    \*  Applies to RCM3600 Prototyping Board.

**value1** is the first A/D converter channel value (0–2047)

**volts1** is the voltage or current corresponding to the first A/D converter channel value (0 to +20 V or 4 to 20 mA)

**value2** is the second A/D converter channel value (0–2047)

**volts2** is the voltage or current corresponding to the first A/D converter channel value (0 to +20 V or 4 to 20 mA)

**RETURN VALUE**

0 if successful.

-1 if not able to make calibration constants.

**SEE ALSO**

```
anaIn, anaInVolts, anaInmAmps, anaInDiff, anaInCalib, brdInit
```

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

Reads the state of a single-ended analog input channel and uses the calibration constants previously set using **anaInCalib** to convert it to volts.

**PARAMETERS**

**channel** is the channel number (0–7)

| Channel Code | Single-Ended Input Lines[*] | Voltage Range[†] (V) |
|---|---|---|
| 0 | +AIN0 | 0–20 |
| 1 | +AIN1 | 0–20 |
| 2 | +AIN2 | 0–20 |
| 3 | +AIN3 | 0–20 |
| 4 | +AIN4 | 0–20 |
| 5 | +AIN5 | 0–20 |
| 6 | +AIN6 | 0–20 |
| 7 | +AIN7 | 0–2[‡] |

  * Negative input is ground.
  † Applies to RCM3600 Prototyping Board.
  ‡ Used for thermistor in sample program.

**gaincode** is the gain code of 0 to 7

| Gain Code | Multiplier | Voltage Range[*] (V) |
|---|---|---|
| 0 | ×1 | 0–20 |
| 1 | ×2 | 0–10 |
| 2 | ×4 | 0–5 |
| 3 | ×5 | 0–4 |
| 4 | ×8 | 0–2.5 |
| 5 | ×10 | 0–2 |
| 6 | ×16 | 0–1.25 |
| 7 | ×20 | 0–1 |

  * Applies to RCM3600 Prototyping Board.

**RETURN VALUE**

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

**ADOVERFLOW** (defined macro = -4096) if overflow or out of range.

**SEE ALSO**

  **anaInCalib, anaIn, anaInmAmps, brdInit**

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

Reads the state of differential analog input channels and uses the calibration constants previously set using **anaInCalib** to convert it to volts.

**PARAMETERS**

**channel** is the analog input channel number (0 to 7) corresponding to ADC_IN0 to ADC_IN7

| channel | DIFF | Voltage Range (V) |
|---------|------|-------------------|
| 0 | +AIN0 -AIN1 | -20 to +20[*] |
| 1 | +AIN1 -AIN0 | — |
| 2 | +AIN2 -AIN3 | -20 to +20* |
| 3 | +AIN3 -AIN2 | — |
| 4 | +AIN4 -AIN5 | -20 to +20* |
| 5 | +AIN5 -AIN4 | — |
| 6 | +AIN6 -AIN7 | — |
| 7 | +AIN7 -AIN6 | — |

    \* Applies to RCM3600 Prototyping Board.

**gaincode** is the gain code of 0 to 7

| Gain Code | Multiplier | Voltage Range[*] (V) |
|-----------|------------|----------------------|
| 0 | x1 | 0–20 |
| 1 | x2 | 0–10 |
| 2 | x4 | 0–5 |
| 3 | x5 | 0–4 |
| 4 | x8 | 0–2.5 |
| 5 | x10 | 0–2 |
| 6 | x16 | 0–1.25 |
| 7 | x20 | 0–1 |

    \* Applies to RCM3600 Prototyping Board.

**RETURN VALUE**

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

**ADOVERFLOW** (defined macro = -4096) if overflow or out of range.

**SEE ALSO**

    **anaInCalib, anaIn, anaInmAmps, brdInit**

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

Reads the state of an analog input channel and uses the calibration constants previously set using **anaInCalib** to convert it to current.

**PARAMETERS**

**channel** is the channel number (0–7)

| Channel Code | 4–20 mA Input Lines* |
|:---:|:---:|
| 0 | +AIN0 |
| 1 | +AIN1 |
| 2 | +AIN2 |
| 3 | +AIN3[†] |
| 4 | +AIN4* |
| 5 | +AIN5* |
| 6 | +AIN6* |
| 7 | +AIN7 |

* Negative input is ground.
† Applies to RCM3600 Prototyping Board.

**RETURN VALUE**

A current value between 4.00 and 20.00 mA corresponding to the current on the analog input channel.

**ADOVERFLOW** (defined macro = -4096) if overflow or out of range.

**SEE ALSO**

**anaInCalib, anaIn, anaInVolts**

```
root int anaInEERd(unsigned int channel,
   unsigned int opmode, unsigned int gaincode);
```

Reads the calibration constants, gain, and offset for an input based on their designated position in the simulated EEPROM area of the flash memory, and places them into global tables for analog inputs. The constants are stored in the top 2K of the reserved user block memory area 0x1C00–0x1FFF. Depending on the flash size, the following macros can be used to identify the starting address for these locations.

**ADC_CALIB_ADDRS**, address start of single-ended analog input channels

**ADC_CALIB_ADDRD**, address start of differential analog input channels

**ADC_CALIB_ADDRM**, address start of milliamp analog input channels

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

**PARAMETER**

**channel** is the analog input channel number (0 to 7) corresponding to ADC_IN0 to ADC_IN7

**opmode** is the mode of operation:

**SINGLE**—single-ended input line
**DIFF**—differential input line
**mAMP**—milliamp input line

| channel | SINGLE | DIFF | mAMP |
|---------|--------|------|------|
| 0 | +AIN0 | +AIN0 -AIN1 | +AIN0$^*$ |
| 1 | +AIN1 | +AIN1 -AIN0* | +AIN1* |
| 2 | +AIN2 | +AIN2 -AIN3 | +AIN2* |
| 3 | +AIN3 | +AIN3 -AIN2* | +AIN3 |
| 4 | +AIN4 | +AIN4 -AIN5 | +AIN4 |
| 5 | +AIN5 | +AIN5 -AIN4* | +AIN5 |
| 6 | +AIN6 | +AIN6 -AIN7* | +AIN6 |
| 7 | +AIN7 | +AIN7 -AIN6* | +AIN7* |
| **ALLCHAN** | read all channels for selected **opmode** | | |

\* Not accessible on Prototyping Board.

**gaincode** is the gain code of 0 to 7. The **gaincode** parameter is ignored when **channel** is **ALLCHAN**.

| Gain Code | Voltage Range* (V) |
|:---:|:---:|
| 0 | 0–20 |
| 1 | 0–10 |
| 2 | 0–5 |
| 3 | 0–4 |
| 4 | 0–2.5 |
| 5 | 0–2 |
| 6 | 0–1.25 |
| 7 | 0–1 |

\* Applies to RCM3600 Prototyping Board.

**RETURN VALUE**

0 if successful.
-1 if address is invalid or out of range.
-2 if there is no valid ID block.

**SEE ALSO**

 **anaInEEWr, anaInCalib**

## int anaInEEWr(unsigned int channel, unsigned int opmode, unsigned int gaincode);

Writes the calibration constants, gain, and offset for an input based from global tables to designated positions in the simulated EEPROM area of the flash memory. The constants are stored in the top 2K of the reserved user block memory area 0x1C00–0x1FFF. Depending on the flash size, the following macros can be used to identify the starting address for these locations.

**ADC_CALIB_ADDRS**, address start of single-ended analog input channels

**ADC_CALIB_ADDRD**, address start of differential analog input channels

**ADC_CALIB_ADDRM**, address start of milliamp analog input channels

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

### PARAMETER

**channel** is the analog input channel number (0 to 7) corresponding to ADC_IN0–ADC_IN7

**opmode** is the mode of operation:

**SINGLE**—single-ended input line
**DIFF**—differential input line
**mAMP**—milliamp input line

| channel | SINGLE | DIFF | mAMP |
|---------|--------|------|------|
| 0 | +AIN0 | +AIN0 -AIN1 | +AIN0$^*$ |
| 1 | +AIN1 | +AIN1 -AIN0* | +AIN1* |
| 2 | +AIN2 | +AIN2 -AIN3 | +AIN2* |
| 3 | +AIN3 | +AIN3 -AIN2* | +AIN3 |
| 4 | +AIN4 | +AIN4 -AIN5 | +AIN4 |
| 5 | +AIN5 | +AIN5 -AIN4* | +AIN5 |
| 6 | +AIN6 | +AIN6 -AIN7* | +AIN6 |
| 7 | +AIN7 | +AIN7 -AIN6* | +AIN7* |
| **ALLCHAN** | read all channels for selected **opmode** | | |

\* Not accessible on RCM3600 Prototyping Board.

**gaincode** is the gain code of 0 to 7. The **gaincode** parameter is ignored when **channel** is **ALLCHAN**.

| Gain Code | Voltage Range[*] (V) |
|:---:|:---:|
| 0 | 0–20 |
| 1 | 0–10 |
| 2 | 0–5 |
| 3 | 0–4 |
| 4 | 0–2.5 |
| 5 | 0–2 |
| 6 | 0–1.25 |
| 7 | 0–1 |

\* Applies to Prototyping Board.

**RETURN VALUE**

0 if successful

-1 if address is invalid or out of range.

-2 if there is no valid ID block.

-3 if there is an error writing to flash memory.

**SEE ALSO**

    anaInEEWr, anaInCalib

## void digConfig(char statemask);

Configures channels PIO0 to PIO3 on the A/D converter to allow them to be used as digital I/O via header JP4 on the RCM3600 Prototyping Board.

Remember to execute the **brdInit** function before calling this function to prevent a runtime error.

**PARAMETER**

**statemask** is a bitwise mask representing JP4 channels 1 to 4. Use logic 0 for inputs and logic 1 for outputs in these bit positions:

    bits 7–5—0
    bit 4—JP4:4
    bit 3—JP4:3
    bit 2—JP4:2
    bit 1—JP4:1
    bit 0—0

**RETURN VALUE**

None.

**SEE ALSO**

    digOut, digIn

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

Writes a state to a digital output channel on header JP4 of the RCM3600 Prototyping Board. The PIO0 to PIO3 channels on the A/D converter chip are accessed via header JP4 on the RCM3600 Prototyping Board.

A runtime error will occur if the **brdInit** function was not executed before calling this function or if the channel is out of range.

**PARAMETERS**

**channel** is channel 1 to 4 for JP4:1 to JP4:4

**state** is a logic state of 0 or 1

**RETURN VALUE**

None.

**SEE ALSO**

> **brdInit, digIn**

## int digIn(int channel);

Reads the state of a digital input channel on header JP4 of the RCM3600 Prototyping Board. The PIO0 to PIO3 channels on the A/D converter chip are accessed via header JP4 on the RCM3600 Prototyping Board.

A runtime error will occur if the **brdInit** function was not executed before calling this function or if the channel is out of range.

**PARAMETERS**

**channel** is channel 1 to 4 for JP4:1 to JP4:4

**state** is a logic state of 0 or 1

**RETURN VALUE**

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

**SEE ALSO**

> **brdInit, digOut**

### 5.2.3  Digital I/O

The RCM3600 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\RCM3600` folder provide further examples.

---

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

This function is used to poll the real-time clock until the specified timeout occurs. The RCM3600 will operate in a low-power mode with a clock speed of 2.048 kHz until the timeout occurs. Once the timeout has ended, the RCM3600 will resume operating at 22.1 MHz. The analog device oscillator will be disabled until the timeout occurs and will then be enabled as well.

**PARAMETERS**

`timeout` is the length of the timeout in seconds

**RETURN VALUE**

None.

---

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

This function is used to poll a digital input for a certain value or until the specified timeout occurs. The RCM3600 will operate in a low-power mode with a clock speed of 2.048 kHz until the correct bit is received or the timeout occurs. Once this happens, the RCM3600 will resume operating at 22.1 MHz. The analog device oscillator will be disabled until the timeout occurs and will then be enabled as well.

**PARAMETERS**

`dataport` is the input port data register corresponding to the channel to poll (e.g., PADR)

`portbit` is the input port bit to poll

`value` is the input value (0 or 1) to receive

`timeout` is the length of the timeout in seconds if an input value is not received on the specified channel; enter 0 for no timeout

**RETURN VALUE**

None.

---

### 5.2.4  Serial Communication Drivers

Library files included with Dynamic C provide a full range of serial communications support. The `RS232.LIB` library provides a set of circular-buffer-based serial functions. The `PACKET.LIB` library provides packet-based serial functions where packets can be delimited by the 9th bit, by transmission gaps, or with user-defined special characters. Both libraries provide blocking functions, which do not return until they are finished transmitting or receiving, and nonblocking functions, which must be called repeatedly until they are finished, allowing other functions to be performed between calls. For more information, see the *Dynamic C Function Reference Manual* and Technical Note TN213, *Rabbit Serial Port Software*.

## 5.3  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.

### 5.3.1  Add-On Modules

Dynamic C installations are designed for use with the board they are included with, and are included at no charge as part of our low-cost kits. Rabbit Semiconductor offers add-on Dynamic C modules including 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.  RCM3600 SPECIFICATIONS

Appendix A provides the specifications for the RCM3600, and describes the conformal coating.

## A.1  Electrical and Mechanical Characteristics

Figure A-1 shows the mechanical dimensions for the RCM3600.



*Please refer to the RCM3600 footprint diagram later in this appendix for precise header locations.*
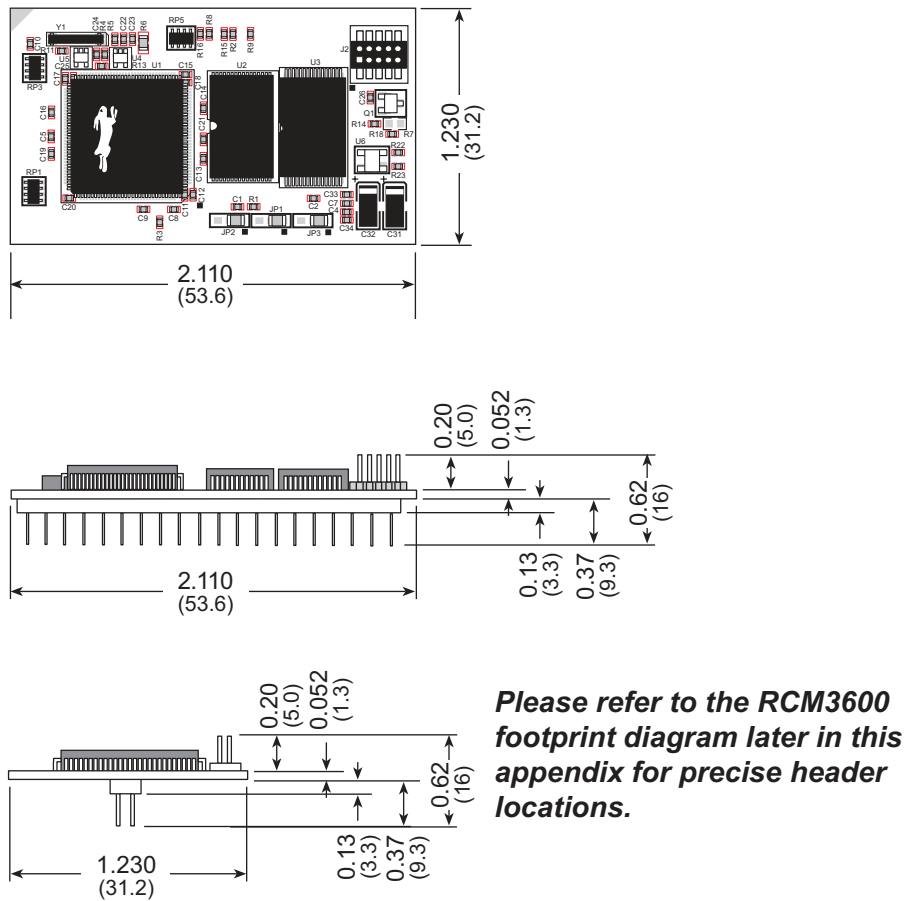
**Figure A-1.  RCM3600 Dimensions**

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

It is recommended that you allow for an "exclusion zone" of 0.04" (1 mm) around the RCM3600 in all directions when the RCM3600 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 RCM3600 when the RCM3600 is plugged into another assembly using the shortest connectors for header J1. Figure A-2 shows this "exclusion zone."
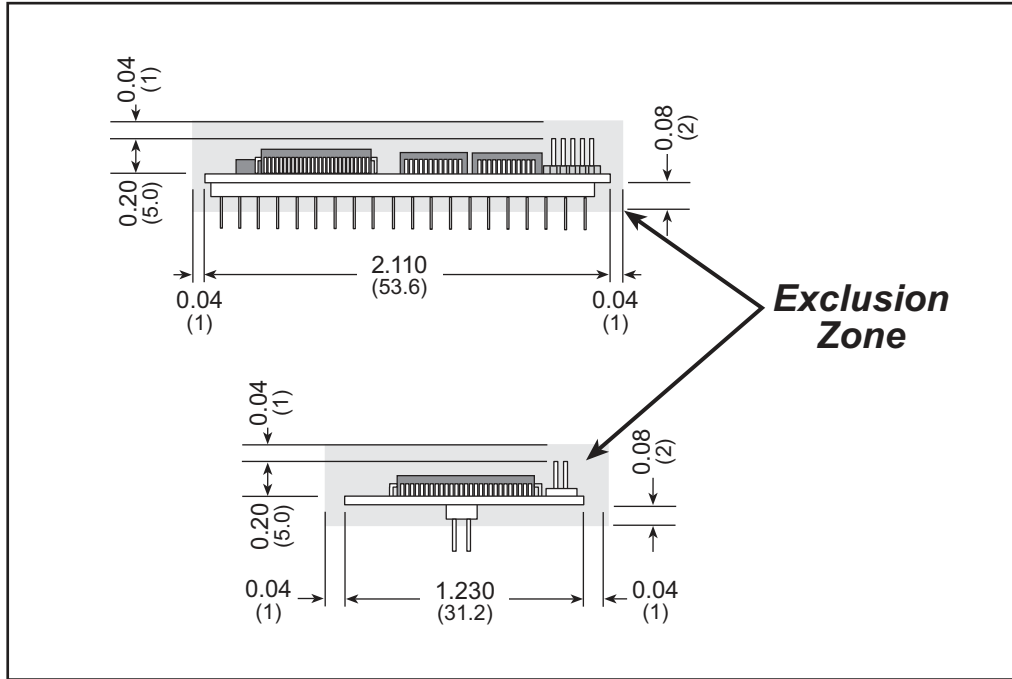


*Figure A-2.  RCM3600 "Exclusion Zone"*

Table A-1 lists the electrical, mechanical, and environmental specifications for the RCM3600.

*Table A-1.  RabbitCore RCM3600 Specifications*

| Parameter | RCM3600 | RCM3610 |
|---|---|---|
| Microprocessor | Low-EMI Rabbit 3000® at 22.1 MHz | |
| Flash Memory | 512K | 256K |
| SRAM | 512K | 128K |
| Backup Battery | Connection for user-supplied backup battery (to support RTC and SRAM) | |
| General-Purpose I/O | 33 parallel digital I/0 lines: <br> • 31 configurable I/O <br> • 2 fixed outputs | |
| Additional I/O | Reset | |
| Auxiliary I/O Bus | Can be configured for 8 data lines and 5 address lines (shared with parallel I/O lines), plus I/O read/write | |
| Serial Ports | Four 3.3 V CMOS-compatible ports configurable as: <br> • 4 asynchronous serial ports (with IrDA) or <br> • 3 clocked serial ports (SPI) plus 1 HDLC (with IrDA) or <br> • 1 clocked serial port (SPI) plus 2 HDLC serial ports (with IrDA) | |
| Serial Rate | Maximum asynchronous baud rate = CLK/8 | |
| Slave Interface | A slave port allows the RCM3600 to be used as an intelligent peripheral device slaved to a master processor, which may either be another Rabbit 3000 or any other type of processor | |
| Real-Time Clock | Yes | |
| Timers | Ten  8-bit timers (6 cascadable), one 10-bit timer with 2 match registers | |
| Watchdog/Supervisor | Yes | |
| Pulse-Width Modulators | 4 PWM output channels with 10-bit free-running counter and priority interrupts | |
| Input Capture/ Quadrature Decoder | 2-channel input capture can be used to time input signals from various port pins <br> • 1 quadrature decoder unit accepts inputs from external incremental encoder modules or <br> • 1 quadrature decoder unit shared with 2 PWM channels | |
| Power | 5 V ±0.25 V DC <br> 60 mA @ 22.1 MHz, 5 V; 38 mA @ 11.06 MHz, 5 V | |
| Operating Temperature | –40°C to +85°C | |
| Humidity | 5% to 95%, noncondensing | |
| Connectors | One 2 x 20, 0.1" pitch | |
| Board Size | 1.23" × 2.11" × 0.62" (31 mm × 54 mm × 16 mm) | |

## A.1.1 Headers

The RCM3600 uses one header at J1 for physical connection to other boards. J1 is a
2 × 20 SMT header with a 0.1" pin spacing.

Figure A-3 shows the layout of another board for the RCM3600 to be plugged into. These
values are relative to the designated fiducial (reference point).
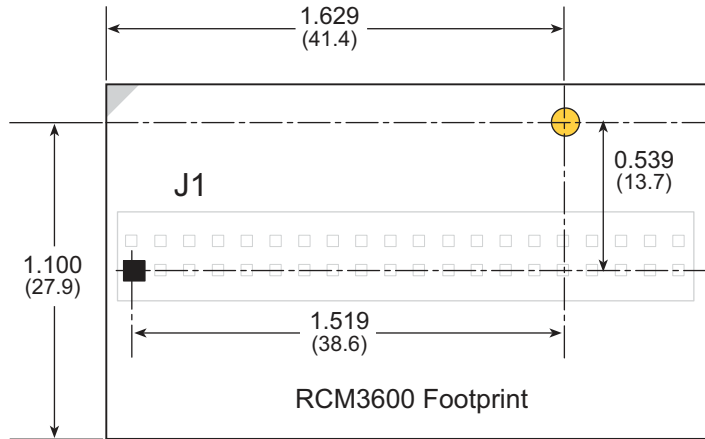


*Figure A-3.  User Board Footprint for RCM3600*

## A.2  Bus Loading

Pay careful attention to bus loading when designing an interface to the RCM3600. This section provides bus loading information for external devices.

Table A-2 lists the capacitance for the various RCM3600 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 RCM3600 output ports. Be sure to add the loads for the devices you are using in your custom system and verify that they do not exceed the values in Table A-3.

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

| Output Port | Clock Speed (MHz) | Maximum External Capacitive Loading (pF) |
|---|---|---|
| All I/O lines with clock doubler enabled | 22.1 | 100 |

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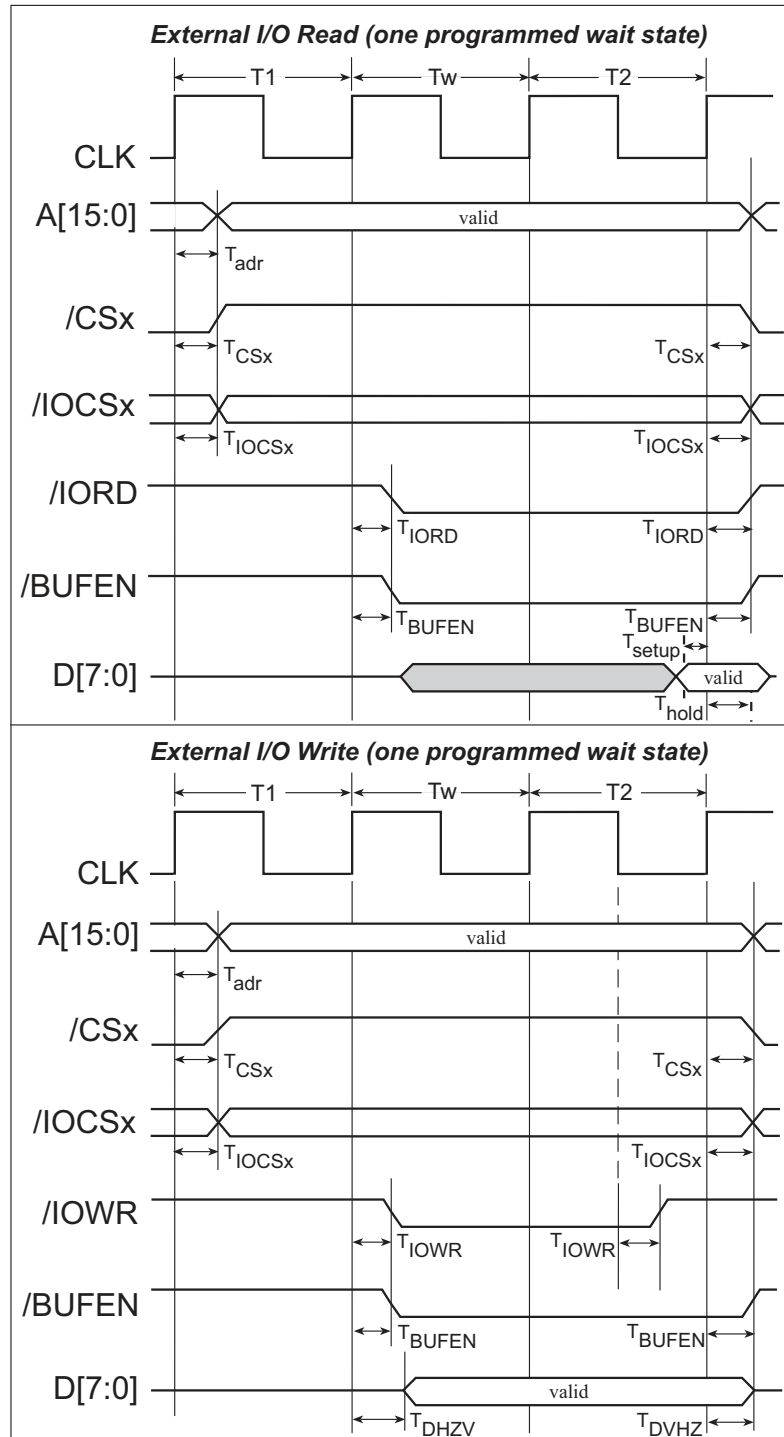


**Figure A-4. I/O Read and Write Cycles—No Extra Wait States**

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

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

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

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

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

- T = -40°C to 85°C, V = $V_{DD}$ ±10%

- Internal clock to nonloaded CLK pin delay ≤ 1 ns @ 85°C/3.0 V

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

- $T_{adr}$, the clock to address delay

- $T_{CSx}$, the clock to memory chip select delay

- $T_{IOCSx}$, the clock to I/O chip select delay

- $T_{IORD}$, the clock to I/O read strobe delay

- $T_{IOWR}$, the clock to I/O write strobe delay

- $T_{BUFEN}$, the clock to I/O buffer enable delay

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

When the spectrum spreader and the clock doubler are both enabled, every other clock cycle is shortened (sometimes lengthened) by a maximum amount given in Table A-4 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 Table A-4.

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

## A.3 Rabbit 3000 DC Characteristics

*Table A-5. Rabbit 3000 Absolute Maximum Ratings*

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

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

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

*Table A-6. 3.3 Volt DC Characteristics*

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

## A.4  I/O Buffer Sourcing and Sinking Limit

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

Table A-7 shows the AC and DC output drive limits of the parallel I/O buffers when the Rabbit 3000 is used in the RCM3600.

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

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

## A.5 Conformal Coating

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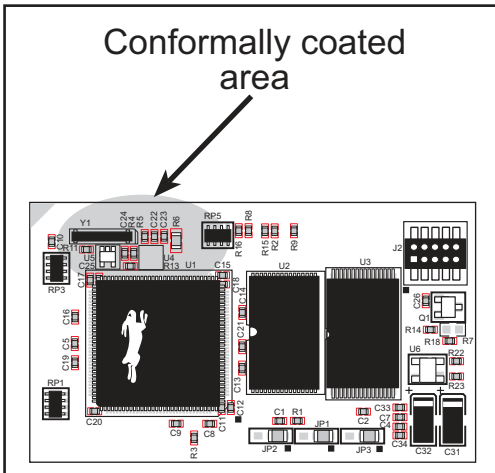


***Figure A-5. RCM3600 Areas Receiving Conformal Coating***

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 RCM3600 options via jumpers.
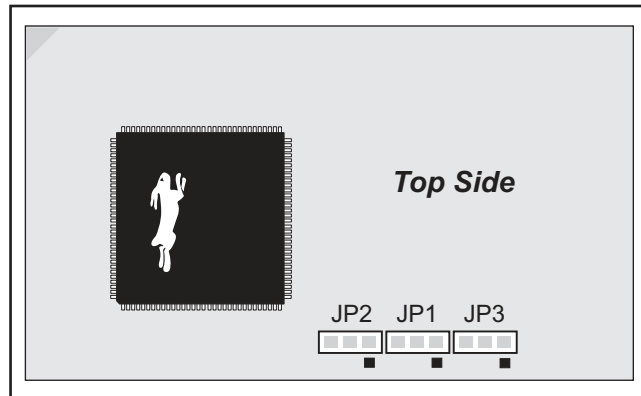


*Figure A-6.  Location of RCM3600 Configurable Positions*

Table A-8 lists the configuration options.

*Table A-8.  RCM3600 Jumper Configurations*

| Header | Description | Pins Connected | | Factory Default |
|---|---|---|---|---|
| JP1 | Flash Memory Bank Select | 1–2 | Normal Mode | ✕ |
| | | 2–3 | Bank Mode | |
| JP2 | SRAM Size | 1–2 | 128K–256K | RCM3610 |
| | | 2–3 | 512K | RCM3600 |
| JP3 | Flash Memory Size | 1–2 | 256K | RCM3610 |
| | | 2–3 | 512K | RCM3600 |

**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 Proto-typing Board.

# B.1  Introduction

The Prototyping Board included in the Development Kit makes it easy to connect an RCM3600 module to a power supply and a PC workstation for development. It also provides some basic I/O peripherals (RS-232, RS-485, an IrDA transceiver, LEDs, and switches), as well as a prototyping area for more advanced hardware development.

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

As you progress to more sophisticated experimentation and hardware development, modifications and additions can be made to the board without modifying or damaging the RCM3600 module itself.

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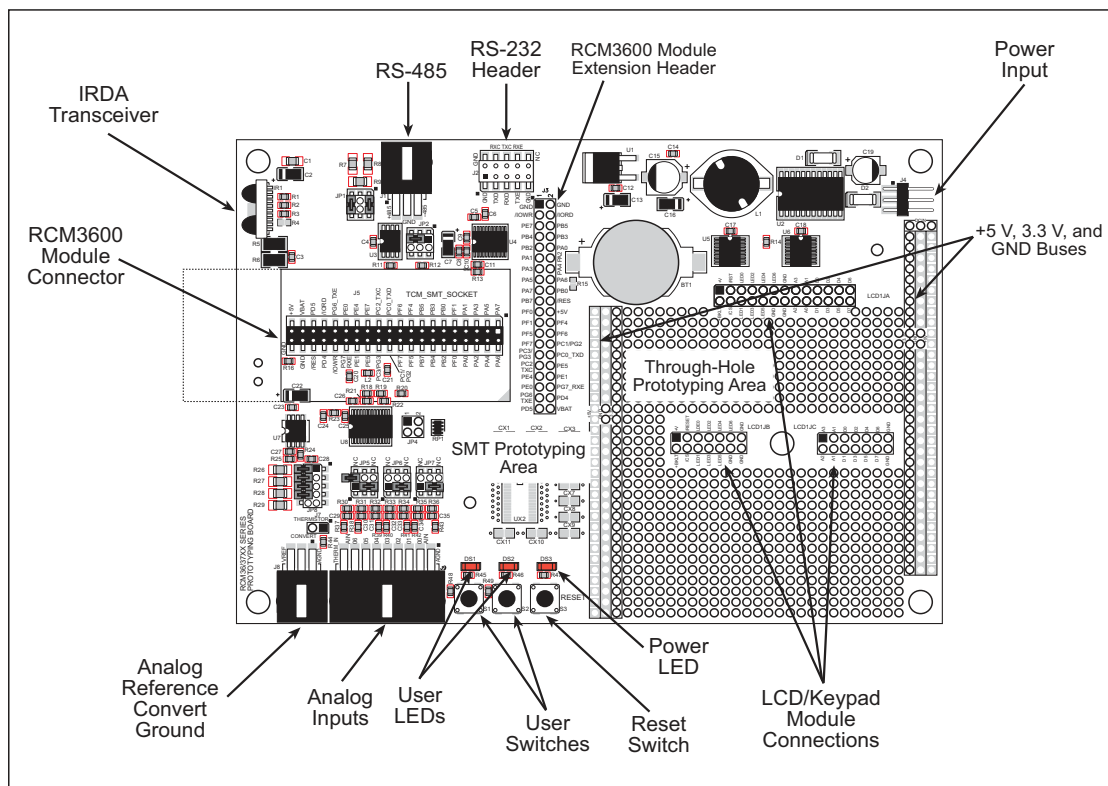


*Figure B-1.  Prototyping Board*

### B.1.1 Prototyping Board Features

- **Power Connection**—A 3-pin header is provided for connection to the power supply. Note that the 3-pin header is symmetrical, with both outer pins connected to ground and the center pin connected to the raw DCIN 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 header, and can be connected to the 3-pin header in either orientation. A similar header plug leading to bare leads is provided for overseas customers.

  Users providing their own power supply should ensure that it delivers 7.5–30 V DC at 500 mA. The voltage regulators will get warm while in use.

- **Regulated Power Supply**—The raw DC voltage provided at the POWER IN power-input 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 RCM3600 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 RCM3600'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 PF4 and PB7 pins of the RCM3600 module and may be read as inputs by sample applications.

  Two LEDs are connected to the PF6 and PF7 pins of the RCM3600 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 at both edges 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.

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

- **Module Extension Headers**—The complete non-analog pin set of the RCM3600 module is duplicated at header J3. Developers can solder wires directly into the appropriate holes, or, for more flexible development, a 2 x 20 header strip with a 0.1" pitch can be soldered into place. See Figure B-4 for the header pinouts.

- **Analog I/O Shrouded Headers**—The complete analog pin set of the RCM3600 Prototyping Board is available on shrouded headers J8 and J9. See Figure B-4 for the header pinouts.

- **RS-232**—Three 3-wire serial ports or one 5-wire RS-232 serial port and one 3-wire serial port are available on the Prototyping Board at header J2. A jumper on header JP2 is used to select the drivers for Serial Port E, which can be set either as a 3-wire RS-232 serial port or as an RS-485 serial port. Serial Ports C and D are not available while the IrDA transceiver is in use.

  A 10-pin 0.1-inch spacing header strip is installed at J2 allows you to connect a ribbon cable that leads to a standard DE9 serial connector.

- **RS-485**—One RS-485 serial port is available on the Prototyping Board at shrouded header J1. A 3-pin shrouded header is installed at J1. A jumper on header JP2 enables the RS-485 output for Serial Port E.

- **IrDA**—An infrared transceiver is included on the Prototyping Board, and is capable of handling link distances up to 1.5 m. The IrDA uses Serial Port F—Serial Ports C and D are unavailable while Serial Port F is in use.

# B.2  Mechanical Dimensions and Layout

Figure B-2 shows the mechanical dimensions and layout for the RCM3600 Prototyping Board.
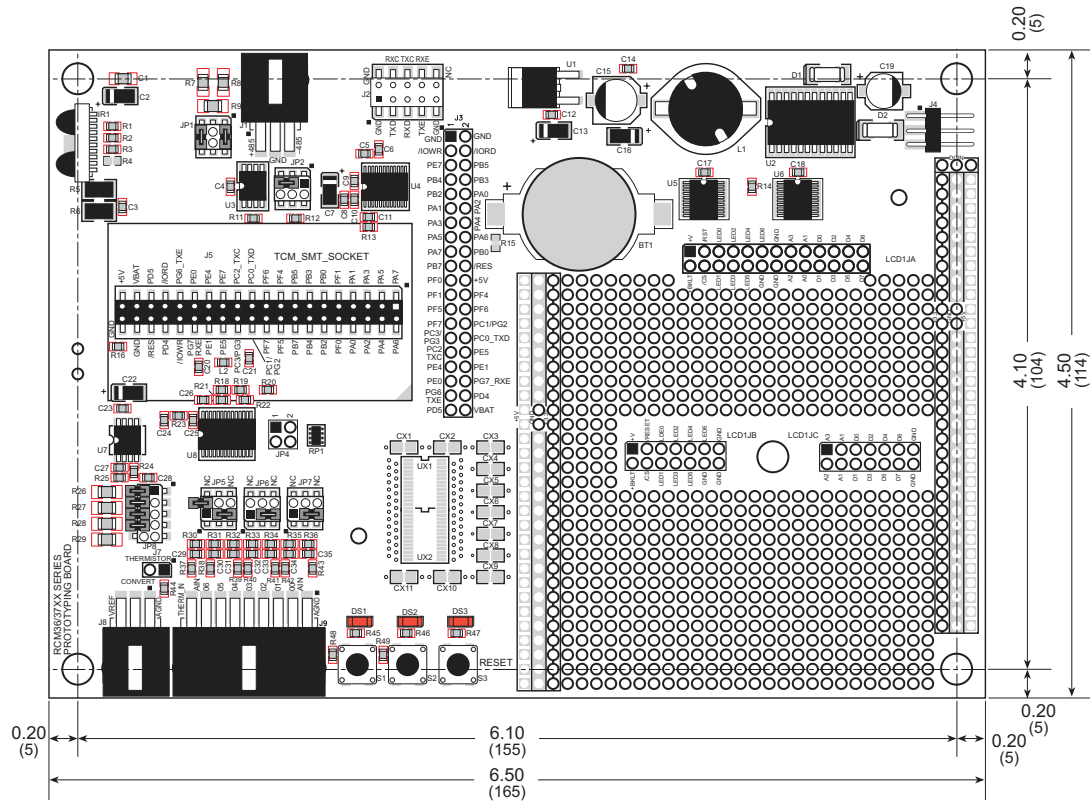


**Figure B-2.  Prototyping Board Dimensions**

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

<p align="center">*Table B-1. Prototyping Board Specifications*</p>

| Parameter | Specification |
|---|---|
| Board Size | 4.50" x 6.50" x 0.75" (114 mm x 165 mm x 19 mm) |
| Operating Temperature | –20°C to +60°C |
| Humidity | 5% to 95%, noncondensing |
| Input Voltage | 7.5 V to 30 V DC |
| Maximum Current Draw (including user-added circuits) | 800 mA max. for +3.3 V supply, 1 A total +3.3 V and +5 V combined |
| A/D Converter | 8-channel ADS7870 with programmable gain configurable for 11-bit single-ended, 12-bit differential, and 4–20 mA inputs<br>• Input impedance 6–7 MΩ<br>• A/D conversion time (including 120 µs raw count and Dynamic C) 180 µs |
| IrDA Transceiver | HSDL-3602, link distances up to 1.5 m |
| Prototyping Area | 2.5" x 3" (64 mm x 76 mm) throughhole, 0.1" spacing, additional space for SMT components |
| Standoffs/Spacers | 5, accept 4-40 x 1/2 screws |

## B.3  Power Supply

The RCM3600 requires a regulated 4.0 V to 12.6 V DC power source to operate. Depending on the amount of current required by the application, different regulators can be used to supply this voltage.

The Prototyping Board has an onboard +5 V switching power regulator from which a +3.3  V linear regulator draws its supply. Thus both +5 V and +3.3 V are available on the Prototyping Board.

The Prototyping Board itself is protected against reverse polarity by a Shottky diode at D2 as shown in Figure B-3.
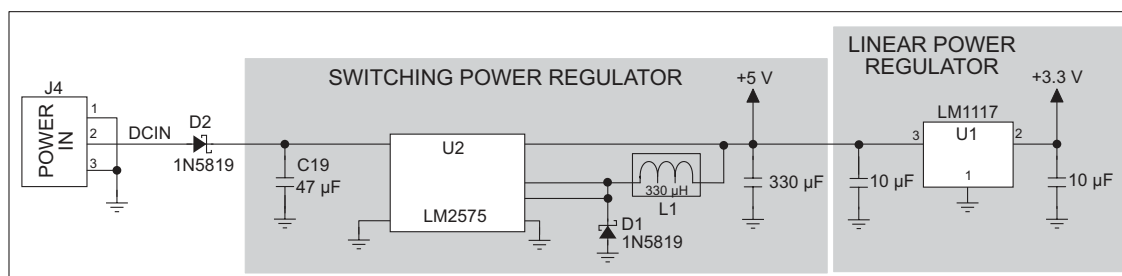


<p align="center">*Figure B-3.  Prototyping Board Power Supply*</p>

# 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 RCM3600 right out of the box without any modifications.

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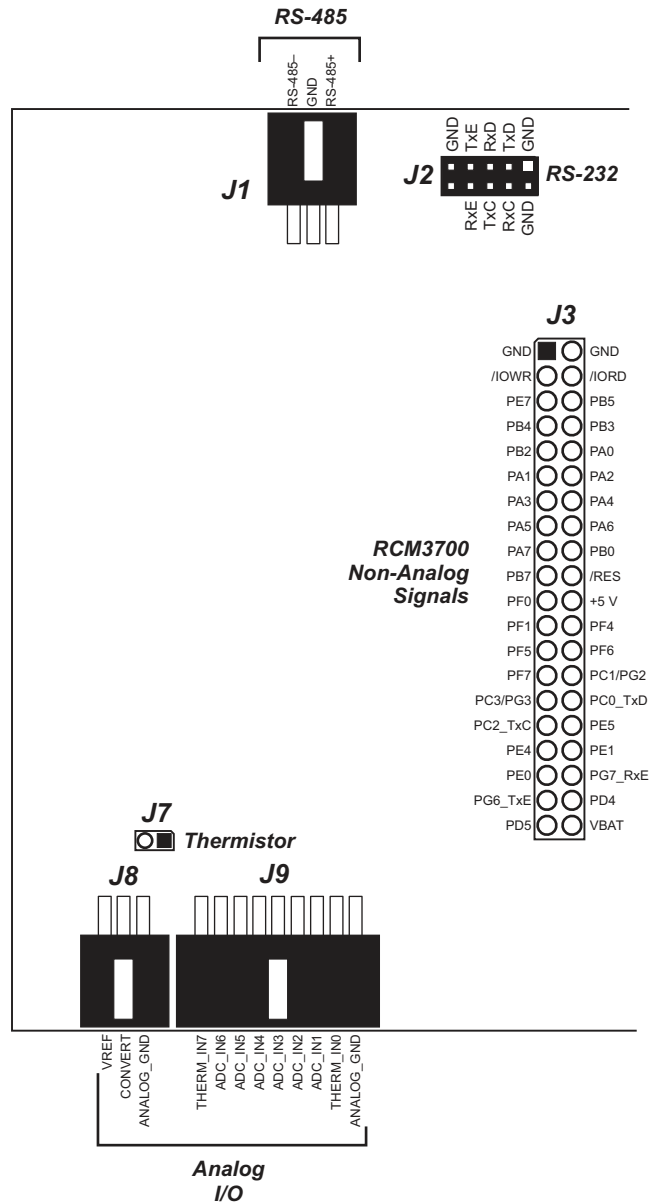


**Figure B-4.  Prototyping Board Pinout**

The Prototyping Board comes with the basic components necessary to demonstrate the operation of the RCM3600. Two LEDs (DS1 and DS2) are connected to PF6 and PF7, and two switches (S1 and S2) are connected to PF4 and PB7 to demonstrate the interface to the Rabbit 3000 microprocessor. Reset switch S3 is the hardware reset for the RCM3600.

The Prototyping Board provides the user with RCM3600 connection points brought out conveniently to labeled points at header J3 on the Prototyping Board. Although header J3 is unstuffed, a 2 x 20 header is included in the bag of parts. RS-485 signals are available on shrouded header J1, and RS-232 signals (Serial Ports C, D, and E) are available on header J2. A header strip at J2 allows you to connect a ribbon cable. A shrouded header connector and wiring harness are included with the Development Kit parts to help you access the RS-485 signals on shrouded header J1.

There is a 2.5" x 3" through-hole prototyping space available on the Prototyping Board. The holes in the prototyping area are spaced at 0.1" (2.5 mm). +3.3 V, +5 V, and GND traces run along both edges of the prototyping area for easy access. Small to medium circuits can be prototyped using point-to-point wiring with 20 to 30 AWG wire between the prototyping area, the +3.3 V, +5 V, and GND traces, and the surrounding area where surface-mount components may be installed. Small holes are provided around the surface-mounted components that may be installed around the prototyping area.

## B.4.1 Adding Other Components

There are two sets of pads for 28-pin devices that can be used for surface-mount prototyping SOIC devices. (Although the adjacent sets of pads could accommodate up to a 56-pin device, they do not allow for the overlap between two 28-pin devices.) There are also pads that can be used for SMT resistors and capacitors in an 0805 SMT package. Each component has every one of its pin pads connected to a hole in which a 30 AWG wire can be soldered (standard wire wrap wire can be soldered in for point-to-point wiring on the Prototyping Board). Because the traces are very thin, carefully determine which set of holes is connected to which surface-mount pad.

## B.4.2 Analog Features

The RCM3600 Prototyping Board has an onboard ADS7870 A/D converter to demonstrate the interface capabilities of the Rabbit 3000. The A/D converter multiplexes converted signals from eight single-ended or three differential inputs to alternate Serial Port B on the Rabbit 3000 (Parallel Port pins PD4 and PD5).

### B.4.2.1 A/D Converter Inputs

Figure B-5 shows a pair of A/D converter input circuits. The resistors form an approximately 10:1 attenuator, and the capacitor filters noise pulses from the A/D converter input.
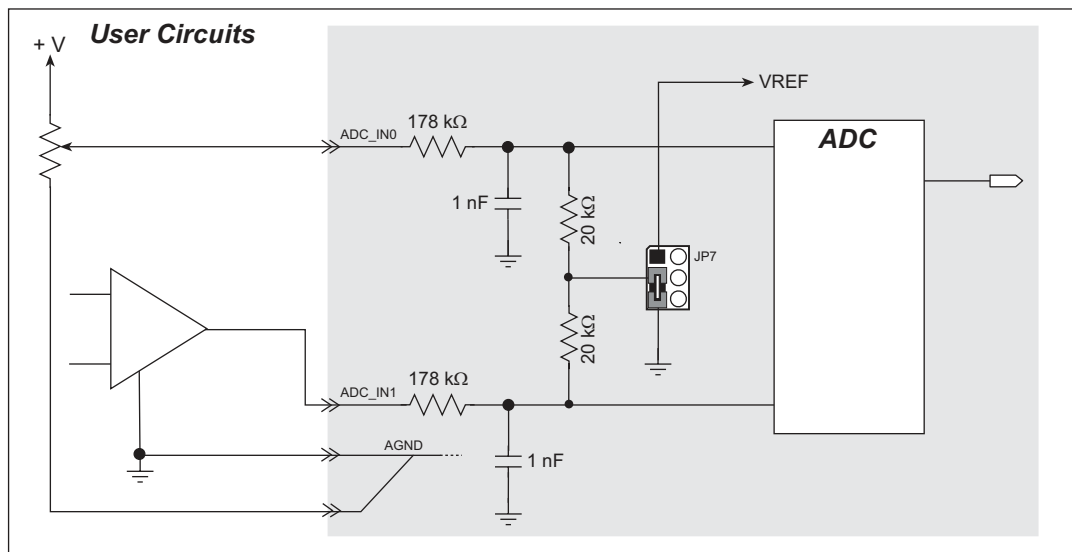


*Figure B-5. A/D Converter Inputs*

The A/D converter chip can make either single-ended or differential measurements depending on the value of the **opmode** parameter in the software function call. Adjacent A/D converter inputs can be paired to make differential measurements. The default setup on the Prototyping Board is to measure only positive voltages for the ranges listed in Table B-2.

*Table B-2.  Positive A/D Converter Input Voltage Ranges*

| Min. Voltage (V) | Max. Voltage (V) | Amplifier Gain | mV per Count |
|---|---|---|---|
| 0.0 | +20.0 | 1 | 10 |
| 0.0 | +10.0 | 2 | 5 |
| 0.0 | +5.0 | 4 | 2.5 |
| 0.0 | +4.0 | 5 | 2.0 |
| 0.0 | +2.5 | 8 | 1.25 |
| 0.0 | +2.0 | 10 | 1.0 |
| 0.0 | +1.25 | 16 | 0.625 |
| 0.0 | +1.0 | 20 | 0.500 |

Other possible ranges are possible by physically changing the resistor values that make up the attenuator circuit.

It is also possible to read a negative voltage on ADC_IN0 to ADC_IN5 by moving the jumper (see Figure B-5) on header JP7, JP6, or JP5 associated with the A/D converter input from analog ground to VREF, the reference voltage generated and buffered by the A/D converter. Adjacent input channels are paired so that moving a particular jumper changes both of the paired channels. At the present time Rabbit Semiconductor does not offer the software drivers to work with single-ended negative voltages, but the differential mode described below may be used to measure negative voltages.

> **NOTE:**  THERM_IN7 was configured to illustrate the use of a thermistor with the sample program, and so is not available for use as a differential input. There is also no resistor attenuator for THERM_IN7, so its input voltage range is limited.

Differential measurements require two channels. As the name *differential* implies, the difference in voltage between the two adjacent channels is measured rather than the difference between the input and analog ground. Voltage measurements taken in differential mode have a resolution of 12 bits, with the 12th bit indicating whether the difference is positive or negative.

The A/D converter chip can only accept positive voltages. Both differential inputs must be referenced to analog ground, and *both inputs must be positive with respect to analog ground*. Table B-3 provides the differential voltage ranges for this setup.

*Table B-3.  Differential Voltage Ranges*

| Min. Differential Voltage (V) | Max. Differential Voltage (V) | Amplifier Gain | mV per Count |
|---|---|---|---|
| 0 | ±20.0 | x1 | 10 |
| 0 | ±10.0 | x2 | 5 |
| 0 | ±5.0 | x4 | 2.5 |
| 0 | ±4.0 | x5 | 2.0 |
| 0 | ±2.5 | x8 | 1.25 |
| 0 | ±2.0 | x10 | 1.00 |
| 0 | ±1.25 | x16 | 0.625 |
| 0 | ±1.0 | x20 | 0.500 |

The A/D converter inputs can also be used with 4–20 mA current sources by measuring the resulting analog voltage drop across 249 Ω 1% precision resistors placed between the analog input and analog ground for ADC_IN3 to ADC_IN6. Be sure to reconfigure the jumper positions on header JP8 as shown in Section B.5 using the slip-on jumpers included with the spare parts in the Development Kit.

### B.4.2.2  Thermistor Input

Analog input THERM_IN7 on the Prototyping Board was designed specifically for use with a thermistor in conjunction with the **THERMISTOR.C** sample program, which demonstrates how to use analog input THERM_IN7 to calculate temperature for display to the Dynamic C **STDIO** window. The sample program is targeted specifically for the thermistor included with the Development Kit with $R_0$ @ 25°C = 3 kΩ and β 25/85 = 3965. Be sure to use the applicable $R_0$ and β values for your thermistor if you use another thermistor. Install the thermistor at location J7, which is shown in Figure B-4.
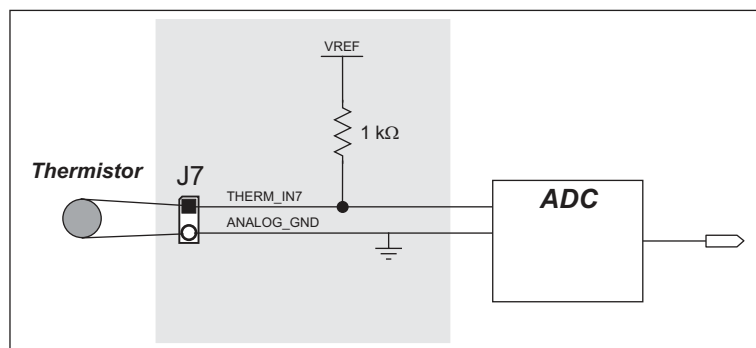


*Figure B-6.  Prototyping Board Thermistor Input*

### B.4.2.3  Other A/D Converter Features

The A/D converter's internal reference voltage is software-configurable for 1.15 V, 2.048 V, or 2.5 V using the `#define AD_OSC_ENABLE` macro in the Dynamic C `RCM36xx.LIB` library. The scaling circuitry on the Prototyping Board and the sample programs are optimized for an internal reference voltage of 2.048 V. This internal reference voltage is available on pin 3 of shrouded header J8 as VREF, and allows you to convert analog input voltages that are negative with respect to analog ground.

> **NOTE:** The amplifier inside the A/D converter's internal voltage reference circuit has a very limited output-current capability. The internal buffer can source up to 20 mA and sink only up to 20 µA. A separate buffer amplifier at U7 supplies the load current.

The A/D converter's CONVERT pin is available on pin 2 of shrouded header J8, and can be used as a hardware means of forcing the A/D converter to start a conversion cycle. The CONVERT signal is an edge-triggered event and has a hold time of two CCLK periods for debounce.

A conversion is started by an active (rising) edge on the CONVERT pin. The CONVERT pin must stay low for at least two CCLK periods before going high for at least two CCLK periods. Figure B-7 shows the timing of a conversion start. The double falling arrow on CCLK indicates the actual start of the conversion cycle.
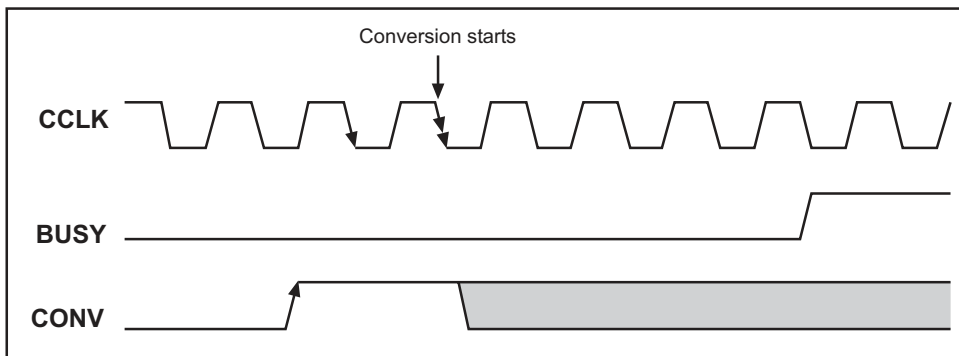


*Figure B-7.  Timing Diagram for Conversion Start Using CONVERT Pin*

### B.4.2.4 A/D Converter Calibration

To get the best results from the A/D converter, it is necessary to calibrate each mode (single-ended, differential, and current) for each of its gains. It is imperative that you calibrate each of the A/D converter inputs in the same manner as they are to be used in the application. For example, if you will be performing floating differential measurements or differential measurements using a common analog ground, then calibrate the A/D converter in the corresponding manner. The calibration must be done with the attenuator reference selection jumper in the desired position (see Figure B-5). If a calibration is performed and the jumper is subsequently moved, the corresponding input(s) must be recalibrated. The calibration table in software only holds calibration constants based on mode, channel, and gain. ***Other factors affecting the calibration must be taken into account by calibrating using the same mode and gain setup as in the intended use.***

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

| Mode | Read | Calibrate |
|---|---|---|
| Single-Ended, one channel | — | `AD_CALSE_CH.C` |
| Single-Ended, all channels | `AD_RDSE_ALL.C` | `AD_CALSE_ALL.C` |
| Milliamp, one channel | `AD_RDMA_CH.C` | `AD_CALMA_CH.C` |
| Differential, analog ground | `AD_RDDIFF_CH.C` | `AD_CALDIFF_CH.C` |

These sample programs are found in the Dynamic C `SAMPLES\RCM3600\ADC` subdirectory. See Section 3.2.2 for more information on these sample programs and how to use them.

## B.4.3  Serial Communication

The RCM3600 Prototyping Board allows you to access five of the serial ports from the RCM3600 module. Table B-4 summarizes the configuration options.

*Table B-4.  RCM3600 Prototyping Board Serial Port Configurations*

| Serial Port | Signal Header | Configured via Header | Default Use | Alternate Use |
|:---:|:---:|:---:|:---:|:---:|
| C | J2 | JP2 | RS-232 | — |
| D | J2 | JP2 | RS-232 | — |
| E | J1, J2 | JP1, JP2 | RS-485 (J1) | RS-232 (J2) |

Serial Port E is configured in hardware for RS-232 or RS-485 via jumpers on header JP2 as shown in Section B.5. Serial Port F is configured in software for the IrDA transceiver in lieu of Serial Ports C and D.

### B.4.3.1 RS-232

RS-232 serial communication on the RCM3600 Prototyping Board is supported by an RS-232 transceiver installed at U4. This transceiver provides the voltage output, slew rate, and input voltage immunity required to meet the RS-232 serial communication protocol. Basically, the chip translates the Rabbit 3000's signals to RS-232 signal levels. Note that the polarity is reversed in an RS-232 circuit so that a +5 V output becomes approximately -10 V and 0 V is output as +10  V. The RS-232 transceiver also provides the proper line loading for reliable communication.

RS-232 can be used effectively at the RCM3600 module's maximum baud rate for distances of up to 15 m.

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

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

**SERA_RTS_SHADOW**—Shadow register for the RTS line's parallel port (e.g., PCDRShadow).

**SERA_RTS_BIT**—The bit number for the RTS line.

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

**SERA_CTS_BIT**—The bit number for the CTS line.

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

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

#define DINBUFSIZE  15
#define DOUTBUFSIZE 15

#define MYBAUD 115200   // set baud rate
#endif

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

**B.4.3.2  RS-485**

The RCM3600 Prototyping Board has one RS-485 serial channel, which is connected to the Rabbit 3000 Serial Port E through an RS-485 transceiver. The half-duplex communication uses an output from PF5 on the Rabbit 3000 to control the transmit enable on the communication line. Using this scheme a strict master/slave relationship must exist between devices to insure that no two devices attempt to drive the bus simultaneously.

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

```
#define ser485open serEopen
#define ser485close serEclose
#define ser485wrFlush serEwrFlush
#define ser485rdFlush serErdFlush
#define ser485putc serEputc
#define ser485getc serEgetc

#define EINBUFSIZE  15
#define EOUTBUFSIZE 15
```

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

The RCM3600 Prototyping Boards with RCM3600 modules installed can be used in an RS-485 multidrop network spanning up to 1200 m (4000 ft), and there can be as many as 32 attached devices. Connect the 485+ to 485+ and 485– to 485– using single twisted-pair wires as shown in Figure B-8. Note that a common ground is recommended.
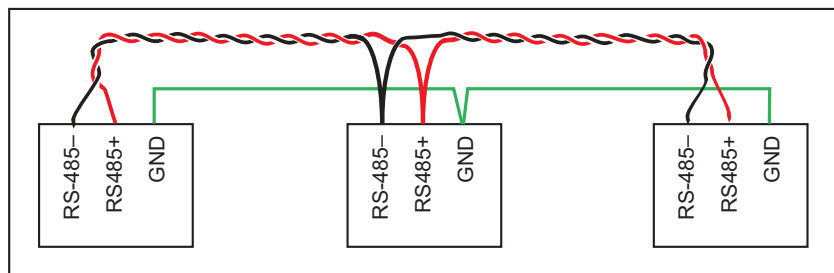


***Figure B-8.  RCM3600 Multidrop Network***

The RCM3600 Prototyping Board comes with a 220 Ω termination resistor and two 681 Ω bias resistors installed and enabled with jumpers across pins 1–2 and 5–6 on header JP1, as shown in Figure B-9.
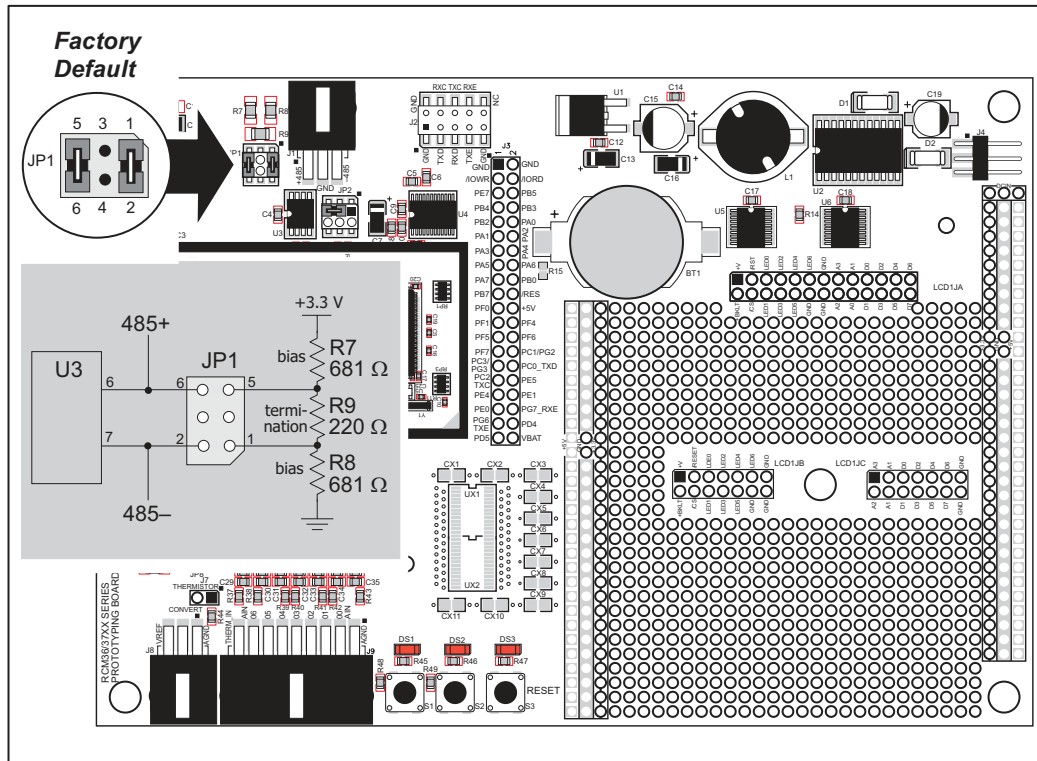


**Figure B-9. RS-485 Termination and Bias Resistors**

For best performance, the termination resistors in a multidrop network should be enabled only on the end nodes of the network, but *not* on the intervening nodes. Jumpers on boards whose termination resistors are not enabled may be stored across pins 1–3 and 4–6 of header JP1.

## B.4.4 Other Prototyping Board Modules

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

> **CAUTION:** Pin PB7 is connected as both switch S2 and as an external I/O bus on the Prototyping Board. Do not use S2 when the LCD/keypad module is installed.

## B.5  RCM3600 Prototyping Board Jumper Configurations

Figure B-10 shows the header locations used to configure the various RCM3600 Prototyping Board options via jumpers.
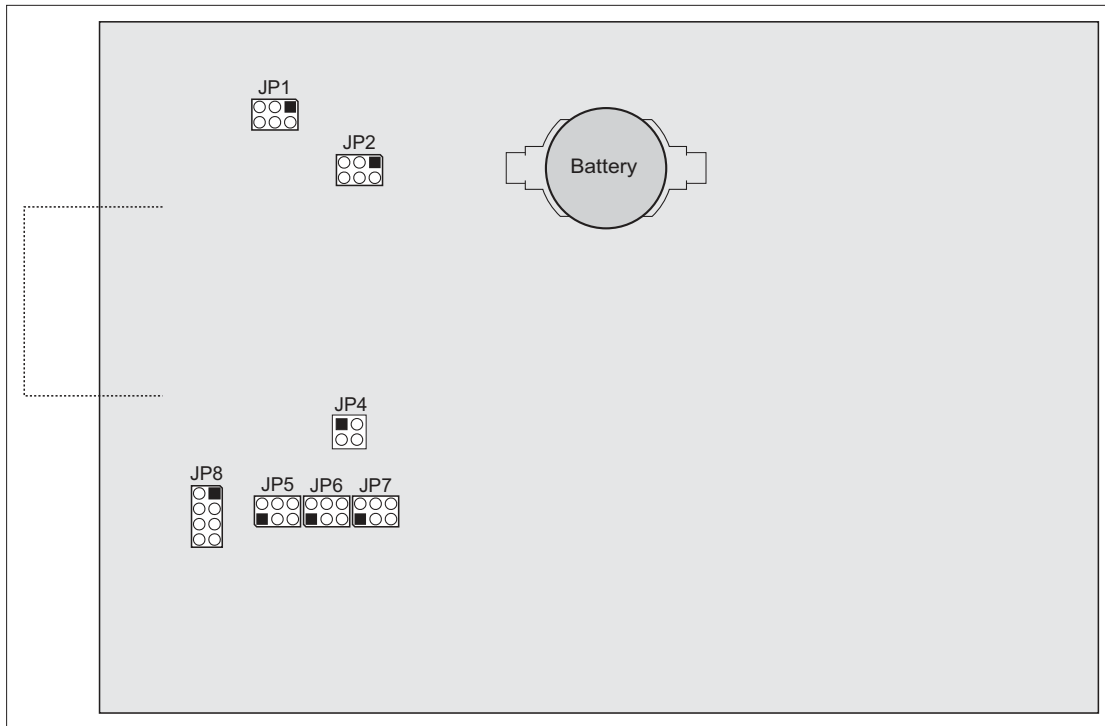


*Figure B-10.  Location of RCM3600 Configurable Positions*

Table B-5 lists the configuration options using jumpers.

*Table B-5.  RCM3600 Jumper Configurations*

| Header | Description | Pins Connected | | Factory Default |
|--------|-------------|----------------|---|-----------------|
| JP1 | RS-485 Bias and Termination Resistors | 1–2 5–6 | Bias and termination resistors connected | ✕ |
| | | 1–3 4–6 | Bias and termination resistors *not* connected (parking position for jumpers) | |
| JP2 | RS-232/RS-485 on Serial Port E | 1–3 2–4 | RS-232 | |
| | | 3–5 4–6 | RS-485 | ✕ |
| JP4 | A/D Converter Outputs | 1 | PIO_0 | n.c. |
| | | 2 | PIO_1 | n.c. |
| | | 3 | PIO_2 | n.c. |
| | | 4 | PIO_3 | n.c. |
| JP5 | ADC_IN4–ADC_IN5 | 1–2 | Tied to VREF | |
| | | 2–3 | Tied to analog ground | ✕ |
| JP6 | ADC_IN2–ADC_IN3 | 1–2 | Tied to VREF | |
| | | 2–3 | Tied to analog ground | ✕ |
| JP7 | ADC_IN0–ADC_IN1 | 1–2 | Tied to VREF | |
| | | 2–3 | Tied to analog ground | ✕ |
| JP8 | Analog Voltage/4–20 mA Options | 1–2 | Connect for 4–20 mA option on ADC_IN3 | n.c. |
| | | 3–4 | Connect for 4–20 mA option on ADC_IN4 | n.c. |
| | | 5–6 | Connect for 4–20 mA option on ADC_IN5 | n.c. |
| | | 7–8 | Connect for 4–20 mA option on ADC_IN6 | n.c. |

# APPENDIX C.  LCD/KEYPAD MODULE

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

## C.1  Specifications

Two optional LCD/keypad modules—with or without a panel-mounted NEMA 4 water-resistant bezel—are available for use with the Prototyping Board. They are shown in Figure C-1.
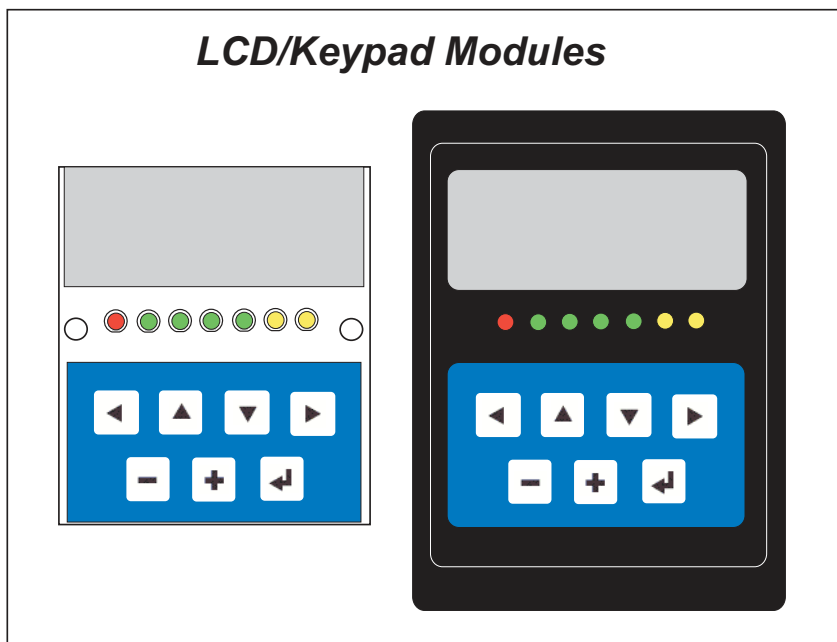


*Figure C-1.  LCD/Keypad Modules Versions*

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

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

*Table C-1.  LCD/Keypad Specifications*

| Parameter | Specification |
|---|---|
| Board Size | 2.60" x 3.00" x 0.75"<br>(66 mm x 76 mm x 19 mm) |
| Bezel Size | 4.50" × 3.60" × 0.30"<br>(114 mm × 91 mm × 7.6 mm) |
| Temperature | Operating Range: 0°C to +50°C<br>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 x 32 graphic display |
| Keypad | 7-key keypad |
| LEDs | Seven user-programmable LEDs |

&ast;   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 dimen-
> sions have a manufacturing toler-
> ance of ±0.01" (0.25 mm).
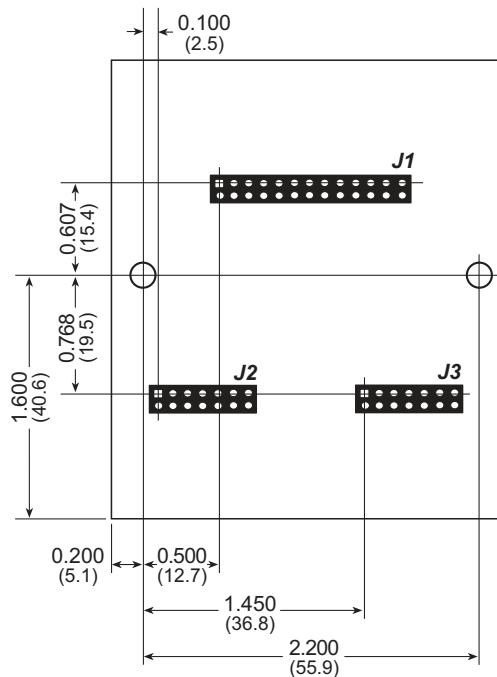


*Figure C-2.  User Board Footprint for
LCD/Keypad Module*

## 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 KDU3V LCD/keypad module for use with the RCM3600 Prototyping Board — these modules operate at 3.3 V. You may adjust the contrast using the potentiometer at R2 as shown in Figure C-3. LCD/keypad modules configured for 5 V may be used with the 3.3 V RCM3600 Prototyping Board, but the backlight will be dim.



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

*LCD/Keypad Module Jumper Configurations*
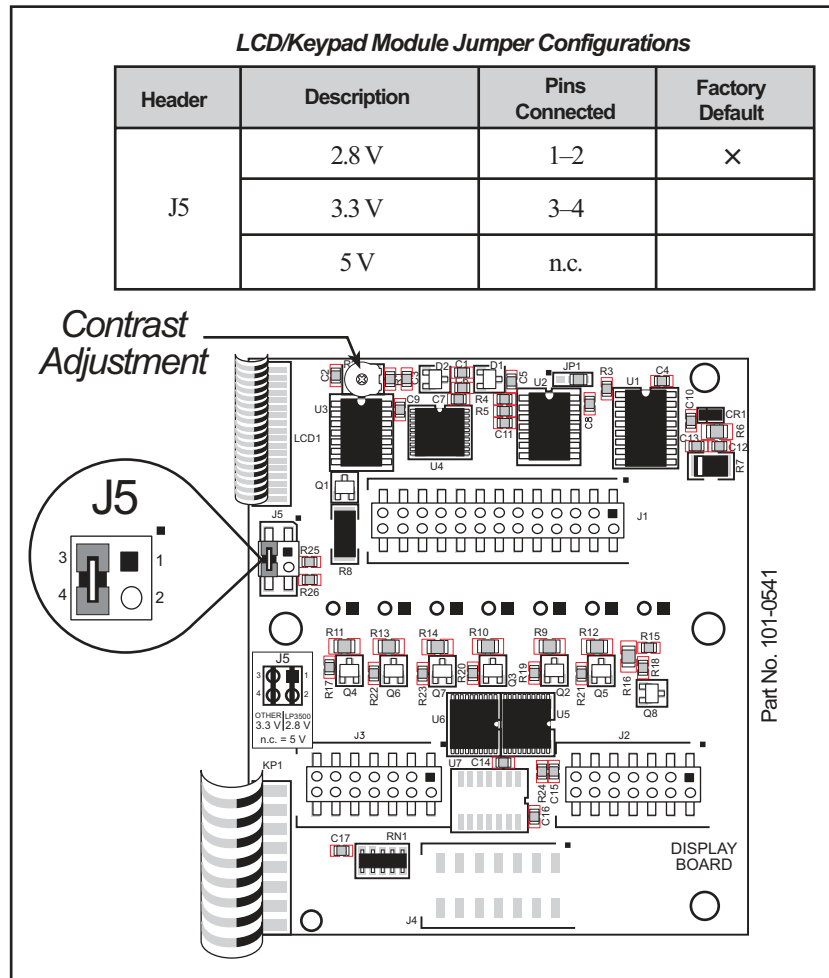
*Figure C-3.  LCD/Keypad Module Voltage Settings*

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 3.3 V by connecting the jumper across pins 3–4 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 not work with the RCM3600 Prototyping Board. The older LCD/keypad modules are no longer being sold.

## C.3  Keypad Labeling

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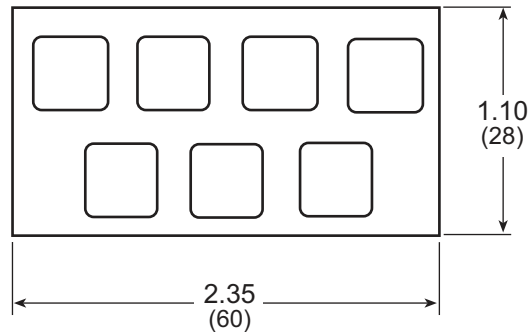


*Figure C-4.  Keypad Template*

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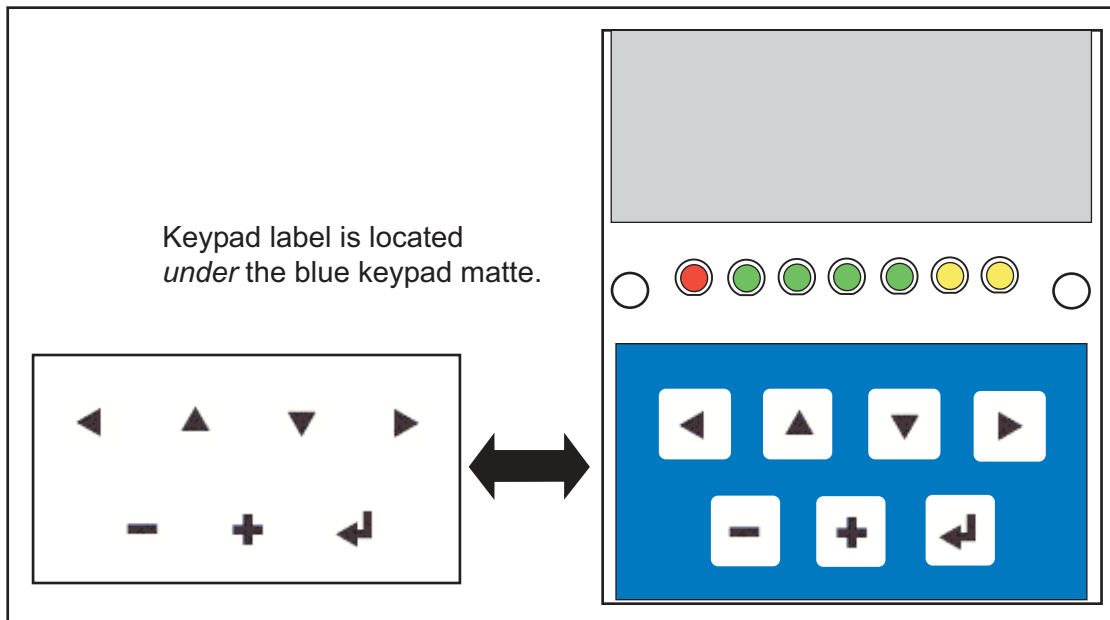


*Figure C-5.  Removing and Inserting Keypad Label*

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

## C.4  Header Pinouts

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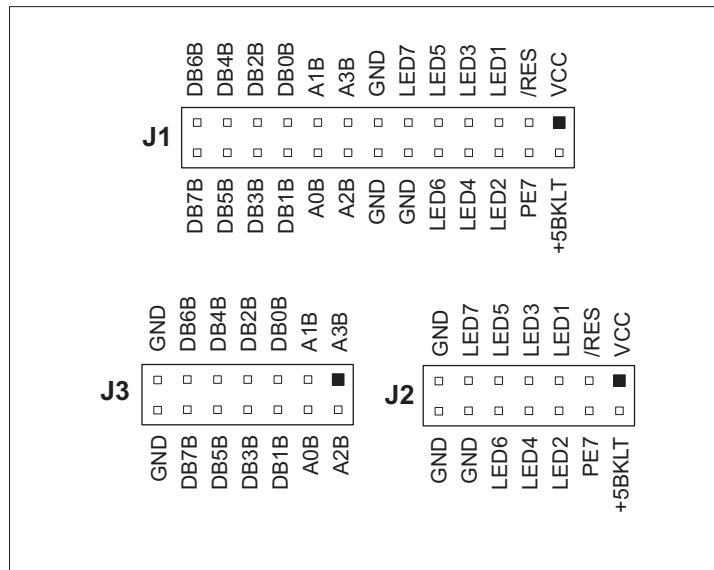


*Figure C-6.  LCD/Keypad Module Pinouts*

## C.4.1  I/O Address Assignments

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

*Table C-2.  LCD/Keypad Module Address Assignment*

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

## C.5 Install Connectors on Prototyping Board

Before you can use the LCD/keypad module with the RCM3600 Prototyping Board, you will need to install connectors to attach the LCD/keypad module to the RCM3600 Prototyping Board. These connectors are included with the RCM3600 Development Kit.

First solder the 2 x 13 connector to location LCD1JA on the RCM3600 Prototyping Board as shown in Figure C-7.

- If you plan to bezel-mount the LCD/keypad module, continue with the bezel-mounting instructions in Section C.7, "Bezel-Mount Installation."

- If you plan to mount the LCD/keypad module directly on the RCM3600 Prototyping Board, solder two additional 2 x 7 connectors at locations LCD1JB and LCD1JC on the RCM3600 Prototyping Board. Section C.6, "Mounting LCD/Keypad Module on the Prototyping Board," explains how to mount the LCD/keypad module on the RCM3600 Prototyping Board.
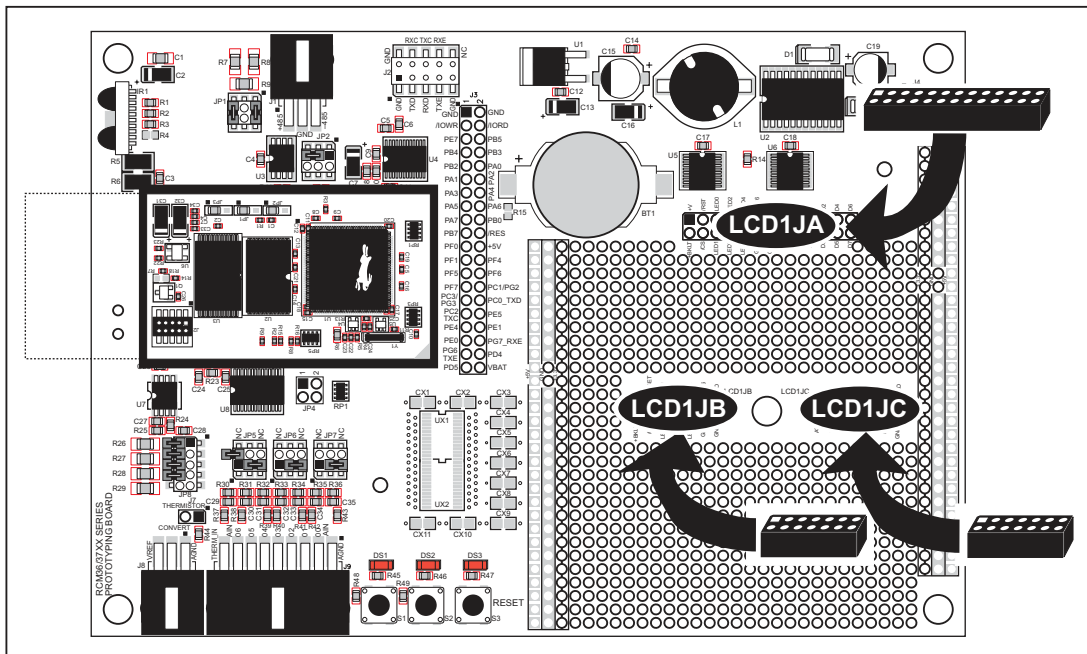


*Figure C-7. Solder Connectors to RCM3600 Prototyping Board*

## C.6 Mounting LCD/Keypad Module on the Prototyping Board

Install the LCD/keypad module on header sockets LCD1JA, LCD1JB, and LCD1JC of the Prototyping Board as shown in Figure C-8. 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.
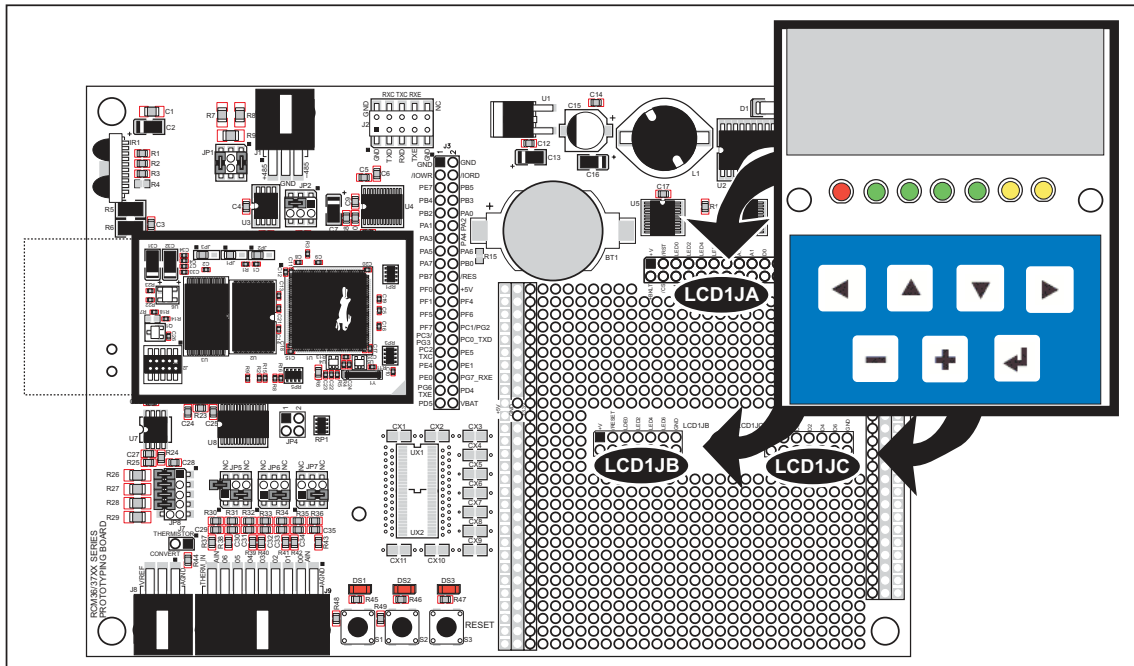


*Figure C-8.  Install LCD/Keypad Module on Prototyping Board*

> **CAUTION:** Pin PB7 is connected as both switch S2 and as an external I/O bus on the Prototyping Board. Do not use S2 when the LCD/keypad module is installed.

## C.7 Bezel-Mount Installation

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

1. Cut mounting holes in the mounting panel in accordance with the recommended dimensions in Figure C-9, then use the bezel faceplate to mount the LCD/keypad module onto the panel.
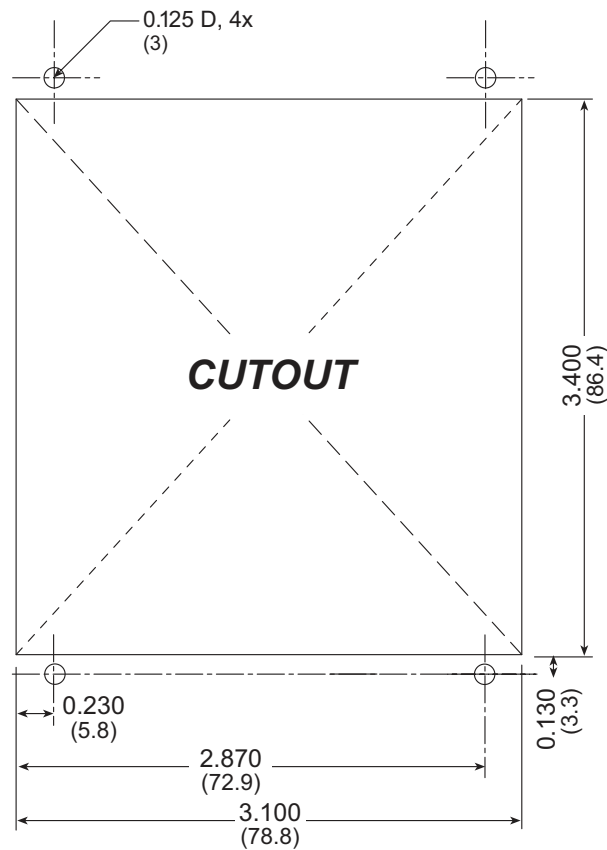
**Figure C-9.  Recommended Cutout Dimensions**

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.
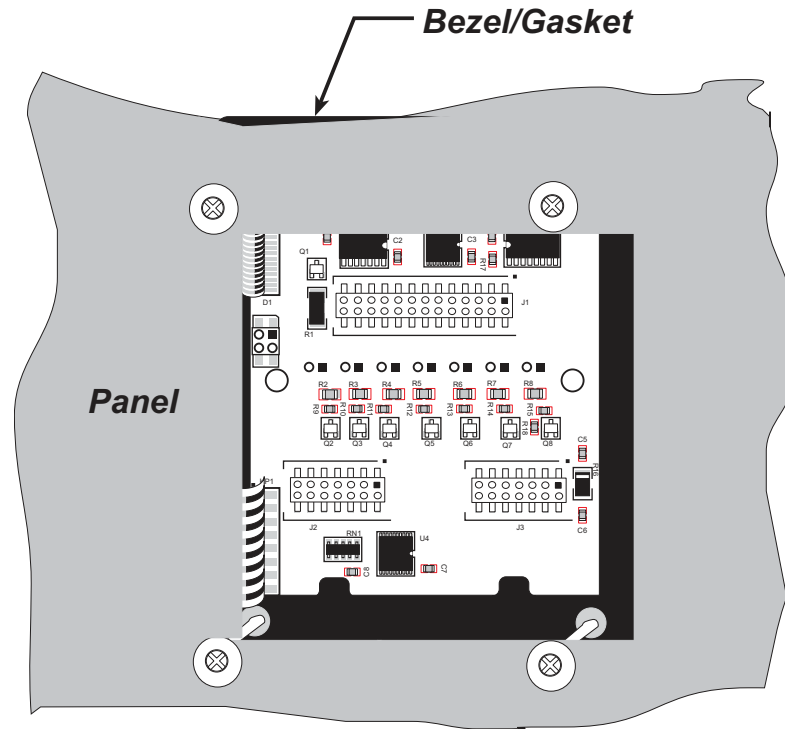


*Figure C-10. LCD/Keypad Module Mounted in Panel (rear view)*

Carefully tighten the screws until the gasket is compressed and the plastic bezel faceplate 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.7.1 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 RCM3600 Prototyping Board, and is connected via a ribbon cable as shown in Figure C-11.
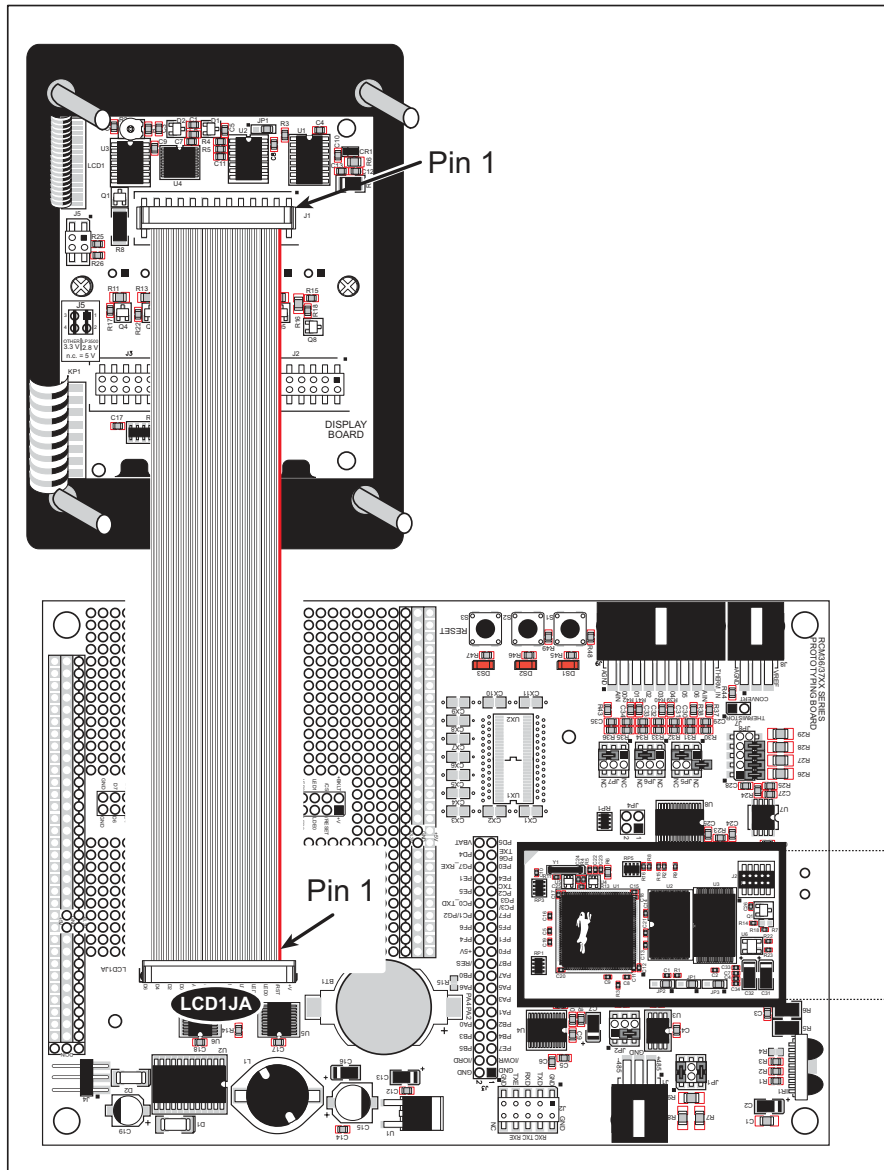


*Figure C-11.  Connecting LCD/Keypad Module to RCM3600 Prototyping Board*

Note the locations and connections relative to pin 1 on both the RCM3600 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.8 Sample Programs

Sample programs illustrating the use of the LCD/keypad module with the Prototyping Board are provided in the **SAMPLES\RCM3600\LCD_KEYPAD** folder.

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

Each sample program has comments that describe the purpose and function of the program. Follow the instructions at the beginning of the sample program. To run a sample program, open it with the **File** menu (if it is not still open), then compile and run it by pressing **F9**. The RCM3600 must be connected to a PC using the programming cable as described in Chapter 2, "Getting Started."

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

- **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.

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

## C.9  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.9.1  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.9.2  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 Dynamic C `LIB\DISPLAYS\LCD122KEY7_LIB` library.

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

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

**PARAMETERS**

**led** is the LED to control.

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

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

> 0 = off
> 1 = on

**RETURN VALUE**

None.

---

### C.9.3 LCD Display

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

## void glInit(void);

Initializes the display devices, clears the screen.

**RETURN VALUE**
None.

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

## void glBackLight(int onOff);

Turns the display backlight on or off.

**PARAMETER**
**onOff** turns the backlight on or off

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

**RETURN VALUE**
None.

**SEE ALSO**
glInit, glDispOnoff, glSetContrast

## void glDispOnOff(int onOff);

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

**PARAMETER**
**onOff** turns the LCD screen on or off

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

**RETURN VALUE**
None.

**SEE ALSO**
glInit, glSetContrast, glBackLight

## void glSetContrast(unsigned level);

Sets display contrast.

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

## void glFillScreen(int 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 glFillRegion(int left, int top, int width, int height, char pattern);

Fills a rectangular block in the LCD buffer with the pattern specified. Any portion of the block that is outside the LCD display area will be clipped.

**PARAMETERS**

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

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

`width` is the width of the block.

`height` is the height of the block.

`pattern` is the bit pattern to display (all black if `pattern` is 0xFF, all white if `pattern` is 0x00, and vertical stripes for any other pattern).

**RETURN VALUE**

None.

**SEE ALSO**

glFillScreen, glBlankScreen, glBlock, glBlankRegion

```
void glFastFillRegion(int left, int top, int width,
    int height, char pattern);
```

Fills a rectangular block in the LCD buffer with the pattern specified. The block left and width parameters must be byte-aligned. Any portion of the block that is outside the LCD display area will be clipped.

**PARAMETERS**

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

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

**width** is the width of the block.

**height** is the height of the block.

**pattern** is the bit pattern to display (all black if **pattern** is 0xFF, all white if **pattern** is 0x00, and vertical stripes for any other pattern).

**RETURN VALUE**

None.

**SEE ALSO**

**glFillScreen, glBlankScreen, glBlock, glBlankRegion**

```
void glBlankRegion(int left, int top, int width,
    int height);
```

Clears a region on the LCD display. The block left and width parameters must be byte-aligned. Any portion of the block that is outside the LCD display area will be clipped.

**PARAMETERS**

**left** is the *x* coordinate of the top left corner of the block (*x* must be evenly divisible by 8).

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

**width** is the width of the block (must be evenly divisible by 8).

**height** is the height of the block.

**RETURN VALUE**

None.

**SEE ALSO**

**glFillScreen, glBlankScreen, glBlock**

## void glBlock(int left, int top, int width, int height);

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**

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

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

**width** is the width of the block.

**height** 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**. Each font character's bitmap is column major and byte aligned.

**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 pointer to the font descriptor.

**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 upper left corner of the text.

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

`pInfo` is a pointer to the font descriptor.

`*fmt` is a formatted string.

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

**EXAMPLE**

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

**RETURN VALUE**

None.

**SEE ALSO**

`glXFontInit`

## void glBuffLock(void);

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

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

**RETURN VALUE**

None.

**SEE ALSO**

`glBuffUnlock, glSwap`

## void glBuffUnlock(void);

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

**RETURN VALUE**

None.

**SEE ALSO**

`glBuffLock, glSwap`

## void glSwap(void);

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

**RETURN VALUE**

None.

**SEE ALSO**

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

## void glSetBrushType(int type);

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

**PARAMETER**

`type` value can be one of the following macros.

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

**RETURN VALUE**

None.

**SEE ALSO**

`glGetBrushType`

## `int glGetBrushType(void);`

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

**RETURN VALUE**

The current brush type.

**SEE ALSO**

    `glSetBrushType`

## `void glXGetBitmap(int x, int y, int bmWidth,`
## `int bmHeight, unsigned long xBm);`

Gets a bitmap from the LCD page buffer and stores it in `xmem` RAM. This function automatically calls `glXGetFastmap` if the left edge of the bitmap is byte-aligned and the left edge and width are each evenly divisible by 8.

This function call is intended for use only when a graphic engine is used to interface with the LCD/keypad module.

**PARAMETERS**

`x` is the *x* coordinate in pixels of the top left corner of the bitmap (*x* must be evenly divisible by 8).

`y` is the *y* coordinate in pixels of the top left corner of the bitmap.

`bmWidth` is the width in pixels of the bitmap (must be evenly divisible by 8).

`bmHeight` is the height in pixels of the bitmap.

`xBm` is the `xmem` RAM storage address of the bitmap.

**RETURN VALUE**

None.

## `void glXGetFastmap(int left, int top, int width,`
## `int height, unsigned long xmemptr);`

Draws bitmap in the specified space. The data for the bitmap are stored in `xmem`. This function is similar to `glXPutBitmap`, except that it's faster. The bitmap must be byte-aligned. Any portion of a bitmap image or character that is outside the LCD display area will be clipped.

This function call is intended for use only when a graphic engine is used to interface with the LCD/keypad module.

**PARAMETERS**

`left` is the *x* coordinate of the top left corner of the bitmap (*x* must be evenly divisible by 8).

`top` is the *y* coordinate in pixels of the top left corner of the bitmap.

`width` is the width of the bitmap (must be evenly divisible by 8).

`height` is the height of the bitmap.

`xmemptr` is the `xmem` RAM storage address of the bitmap.

**RETURN VALUE**

None.

**SEE ALSO**

    `glXPutBitmap, glPrintf`

## 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 pointer to the window frame descriptor.

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

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

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

**winWidth** is the width of the text window frame.

**winHeight** is the height of the text window frame.

**RETURN VALUE**

  0—window frame was successfully created.

 -1—*x* coordinate + width has exceeded the display boundary.

-2—*y* coordinate + height has exceeded the display boundary.

-3—Invalid winHeight and/or winWidth parameter value.

## void TextBorderInit(windowFrame *wPtr, int border, char *title);

This function initializes the window frame structure with the border and title information.

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

**PARAMETERS**

**wPtr** is a pointer to the window frame descriptor.

**border** is the border style:

**SINGLE_LINE**—The function will draw a single-line border around the text window.
**DOUBLE_LINE**—The function will draw a double-line border around the text window.

**title** is a pointer to the title information:

If a **NULL** string is detected, then no title is written to the text menu.
If a string is detected, then it will be written center-aligned to the top of the text menu box.

**RETURN VALUE**

None.

**SEE ALSO**

TextBorder, TextGotoXY, TextPutChar, TextWindowFrame, TextCursorLocation

## void TextBorder(windowFrame *wPtr);

This function displays the border for a given window frame. This function will automatically adjust the text window parameters to accommodate the space taken by the text border. This adjustment will only occur once after the **TextBorderInit** function executes.

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

**PARAMETERS**

**wPtr** is a pointer to the window frame descriptor.

**RETURN VALUE**

None.

**SEE ALSO**

TextBorderInit, TextGotoXY, TextPutChar, TextWindowFrame, TextCursorLocation

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

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

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

**PARAMETERS**

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

**col** is a character column location.

**row** is a character row location.

**RETURN VALUE**

None.

**SEE ALSO**

`TextPutChar, TextPrintf, TextWindowFrame`

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

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

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

**PARAMETERS**

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

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

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

**RETURN VALUE**

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

**SEE ALSO**

`TextGotoXY, TextPrintf, TextWindowFrame, TextCursorLocation`

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

Displays a character on the display where the cursor is currently pointing. Once a character is displayed, the cursor will be incremented to the next character position. 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; escape sequences '\r' and '\n' are also recognized. All other escape sequences will be skipped over; for example, \b and \t will cause nothing to be displayed.

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

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

**PARAMETERS**

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

**\*fmt** is a formatted string.

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

**EXAMPLE**

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

**RETURN VALUE**

None.

**SEE ALSO**

    TextGotoXY, TextPutChar, TextWindowFrame, TextCursorLocation

## int TextMaxChars(windowFrame *wPtr);

This function returns the maximum number of characters that can be displayed within the text window.

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

**PARAMETERS**

**wPtr** is a pointer to the window frame descriptor.

**RETURN VALUE**

The maximum number of characters that can be displayed within the text window.

**SEE ALSO**

`TextGotoXY, TextPrintf, TextWindowFrame, TextCursorLocation`

## void TextWinClear(windowFrame *wPtr);

This functions clears the entire area within the specified text window.

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

**PARAMETERS**

**wPtr** is a pointer to the window frame descriptor.

**RETURN VALUE**

None.

**SEE ALSO**

`TextGotoXY, TextPrintf, TextWindowFrame, TextCursorLocation`

## C.9.4 Keypad

The functions used to control the keypad are contained in the Dynamic C `LIB\KEY-PADS\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, which is approximately one debounce period or 5 μs.

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

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

How many times to repeat.
0 = None.

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

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

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

How many times to repeat after low speed repeat.

0 = None.

**RETURN VALUE**

None.

**SEE ALSO**

`keyProcess, keyGet, keypadDef`

## `void keyProcess(void);`

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

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

**RETURN VALUE**

None

**SEE ALSO**

`keyConfig, keyGet, keypadDef`

## `char keyGet(void);`

Get next keypress.

**RETURN VALUE**

The next keypress, or 0 if none

**SEE ALSO**

`keyConfig, keyProcess, keypadDef`

## `int keyUnget(char cKey);`

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

**PARAMETER**

`cKey`

**RETURN VALUE**

None.

**SEE ALSO**

`keyGet`

## void keypadDef();

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

Keypad physical mapping 1 x 7

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

where

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

**Example:** Do the following for the above physical vs. ASCII return key codes.

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

Characters are returned upon keypress with no repeat.

**RETURN VALUE**

None.

**SEE ALSO**

```
keyConfig, keyGet, keyProcess
```

## void keyScan(char *pcKeys);

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

PARAMETER

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

**RETURN VALUE**

None.

**SEE ALSO**

```
keyConfig, keyGet, keypadDef, keyProcess
```

# APPENDIX D. POWER SUPPLY

Appendix D provides information on the current requirements of the RCM3600, and includes some background on the chip select circuit used in power management.

## D.1 Power Supplies

Power is supplied from the motherboard to which the RCM3600 is connected via header J1. The RCM3600 has an onboard +3.3 V linear power regulator that provides the +3.3 V supply to operate the RCM3600. Figure D-1 shows the power-supply circuit.
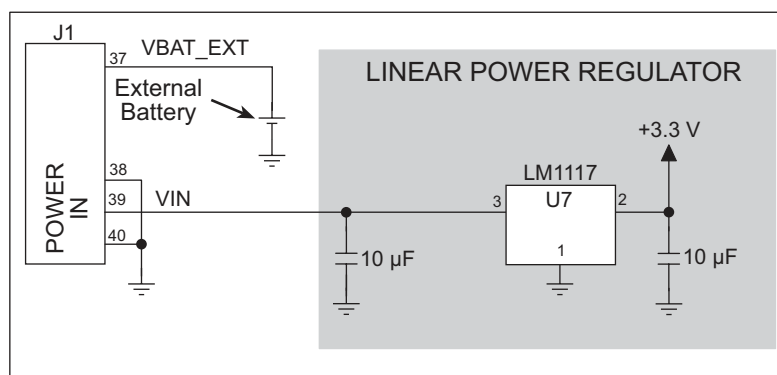


***Figure D-1. RCM3600 Power Supply***

The input voltage should be 5 V ± 0.25 V DC. An RCM3600 with no loading at the outputs typically draws 60 mA when operating at 22.1 MHz. Take care that any DC loading (for example, sourcing digital outputs) does not increase the overall current to more than 190 mA to keep the +3.3 V linear regulator from overheating.

### D.1.1 Battery-Backup Circuits

The RCM3600 does not have a battery, but there is provision for a customer-supplied battery to back up the data SRAM and keep the internal Rabbit 3000 real-time clock running.

Header J1, shown in Figure D-1, allows access to the external battery. This header makes it possible to connect an external 3 V battery. This allows the SRAM and the internal Rabbit 3000 real-time clock to retain data with the RCM3600 powered down.

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 RCM3600 is typically 6 µA when no other power is supplied. If a 235 mA·h battery is used, the battery can last about 4.5 years:

$$\frac{235 \text{ mA·h}}{6 \text{ µA}} = 4.5 \text{ years.}$$

The actual life in your application will depend on the current drawn by components not on the RCM3600 and the storage capacity of the battery. The RCM3600 does not drain the battery while it is powered up normally.

Cycle the main power off/on on the RCM3600 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 RCM3600 experience a loss of main power.

> **NOTE:** Remember to cycle the main power off/on any time the RCM3600 is removed from the Protoyping Board or motherboard since that is where the backup battery would be located.

## D.1.2  Reset Generator

The RCM3600 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 RCM3600 has a reset pin, pin 36 on header J1. This pin provides access to the reset output from the reset generator, and is also connected to the reset input of the Rabbit 3000 to allow you to reset the microprocessor externally. A resistor divider consisting of R21 and R22 attenuates the signal associated with an externally applied reset to prevent it from affecting the reset generator.

# INDEX

# SCHEMATICS

## 090-0176 RCM3600 Schematic

www.rabbit.com/documentation/schemat/090-0176.pdf

## 090-0180 Prototyping Board Schematic

www.rabbit.com/documentation/schemat/090-0180.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.