PRODUCT MANUAL

PowerCore FLEX™

C-Programmable PowerCore Module
with Mass Storage and Ethernet

User's Manual

019–0141 • 070831–E

PowerCore FLEX™ User's Manual

Part Number 019-0141 • 070831–E • Printed in U.S.A.

Trademarks

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

Rabbit 3000 and PowerCore FLEX 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

Rabbit Semiconductor Inc.

www.rabbit.com

Rabbit Semiconductor Inc.

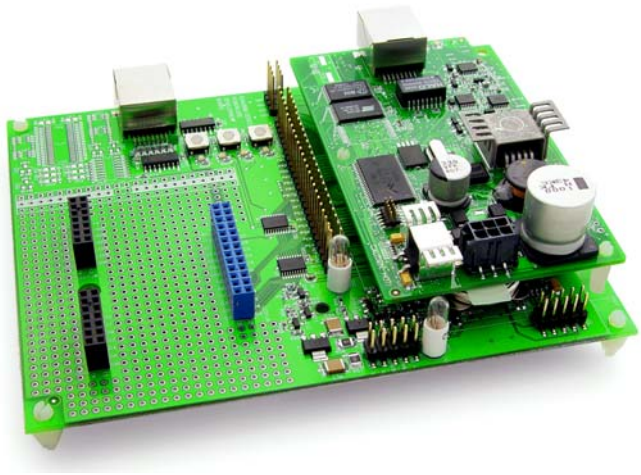
www.rabbit.com

TABLE OF CONTENTS

1. INTRODUCTION

The PowerCore is an easy-to-use core module with a networkable microprocessor system that has an optional onboard configurable power supply. In addition to two standard preconfigured models, PowerCore FLEX™ modules can be built on a quick-turn basis based on customer-selected options. Customers can select memory, Ethernet, power supply, and other features to suit their individual needs without having to pay for unneeded features.

The PowerCore is designed to plug into a motherboard designed by the customer. A 50-pin connector brings the various I/Os, the I/O bus, and the power supplies to the customer's motherboard. Three snap-in plastic standoffs provide sturdy mechanical support.



The PowerCore is supported by powerful Dynamic C development platform that includes extensive libraries to support networking and the Internet.

PowerCore modules are programmed over a standard PC serial port through a programming cable supplied with the Tool Kit, and can also be programmed through a USB port with an RS-232/USB converter, or directly over an Ethernet link via a RabbitLink.

1.1 PowerCore Features

1.1.1 Basic Features

- Small size: 2.35" × 4.00" × 1.08"
  (60 mm × 102 mm × 28 mm)
- 39 configurable 5 V tolerant general-purpose I/O lines
- Three additional digital inputs, two additional digital outputs
- Five 3.3 V CMOS-compatible serial ports with a maximum asynchronous baud rate up to 6.45 Mbps. Three ports are configurable as a clocked serial port (SPI), two ports are configurable as HDLC serial ports, and one ports is configurable as an SDLC serial port. One of the serial ports is normally dedicated as a programming port.
- 512K flash memory for storing instructions
- 256K static RAM for data
- Rabbit 3000® microprocessor running at 25.8 MHz. The Rabbit 3000 includes many powerful I/O devices such as serial ports, and precision pulse generation and measurement.
- 50-pin connector brings I/O and I/O bus to customer's motherboard
- Battery backable time/date clock

1.1.2 Options

- Ethernet including RJ-45 connector (10/100 compatible)
- Clock speed 51.6 MHz
- Analog precision ramp generator that can be used in conjunction with low-cost comparators to create rugged A/D converter inputs
- Larger 512K SRAM memory for data
- Second 512K flash memory
- 1 Mbyte serial flash memory that implements file storage
- Coin cell battery to back up the SRAM and the onboard time/date clock
- Onboard +5 V DC switching power supply rated at 1 A or 2 A
- Triac support with AC zero-crossover detection
- Wi-Fi Add-On Kit to enable Wi-Fi interface

- External power-supply options for PowerCore module and motherboard
  - ▶ user-supplied regulated +5 V DC is supplied to the PowerCore module from the motherboard—the +5 V is regulated down to +3.45 V for driving the nominal 3.3 V components on the PowerCore; the motherboard can draw from the 3.45 V supply
  - ▶ user-supplied unregulated DC (8–40 V) is supplied to the PowerCore module from the motherboard or via the locking power connector at J3—requires onboard +5 V switching regulator @ 1 A or 2 A maximum output; the +5 V is regulated down to +3.45 V for driving the nominal 3.3 V components on the PowerCore; the motherboard can draw from both the 3.45 V and the +5 V regulated supplies
  - ▶ AC from two-lead transformer is supplied to the PowerCore module that has half-wave rectifier and filtering capacitor, includes zero-crossover detection to synchronize software drivers for triac support, two-lead transformer with untapped secondary winding may be used—the resulting DC then passes to the onboard +5 V switching regulator @ 1 A or 2 A maximum output, and is further regulated to +3.45 V; additional filtering capacitors may be needed on motherboard if high unregulated DC current will be drawn
  - ▶ AC from three-lead center-tapped transformer is supplied to the PowerCore module that has full-wave rectifier and filtering capacitor, includes zero-crossover detection to synchronize software drivers for triac support, transformer with tapped secondary winding required—the resulting DC then passes to the onboard +5 V switching regulator @ 1 A or 2 A maximum output, and is further regulated to +3.45 V; additional filtering capacitors may be needed on motherboard if high unregulated DC current will be drawn
  - ▶ AC from two-lead transformer is supplied to the PowerCore module that has full-wave bridge rectifier and filtering capacitor, no zero-crossover detection or triac support, transformer with untapped secondary winding may be used—the resulting DC then passes to the onboard +5 V switching regulator @ 1 A or 2 A maximum output, and is further regulated to +3.45 V; additional filtering capacitors may be needed on motherboard if high unregulated DC current will be drawn

1.2 Standard Configurations and PowerCore FLEX Options

There are two preconfigured PowerCore models. Table 1 below summarizes their main features.

Table 1. Standard PowerCore Production Models

Feature	PowerCore 3800	PowerCore 3810
Microprocessor	Rabbit 3000 running at 51.6 MHz	Rabbit 3000 running at 25.8 MHz
Ethernet	10/100 compatible 10Base-T interface	—
SRAM	512K program (fast SRAM) + 512K data	256K data
Flash Memory (program)	512K	512K
Flash Memory (mass data storage)	1 Mbyte (serial flash)	—
Current Limits for Onboard +5 V DC Voltage Regulators	2 A	1 A

If the standard models do not serve your needs, flexible PowerCore FLEX options can be configured to meet your needs.

Appendix A provides detailed specifications for the PowerCore modules.

1.3 PowerCore FLEX Advantages

- Fast time-to-market using a fully engineered, “ready-to-run/ready-to-program” microprocessor core.
- Competitive pricing when compared with the alternative of purchasing and assembling individual components.
- Easy C-language program development and real-time debugging with integrated Dynamic C® environment.
- Onboard regulated power supply, which can be used to power external circuits.
- Program download utility (Rabbit Field Utility) and cloning board options for rapid production loading of programs.
- Generous memory size allows large programs with tens of thousands of lines of code, and substantial data storage.
- Integrated Ethernet port for network connectivity, with royalty-free TCP/IP software.
- Ideal for network-enabling security and access systems, home automation, HVAC systems, and industrial controls.
- Can use either AC or DC power sources.
- Onboard analog circuits (AC zero-crossover detection allows triac control, ramp generator allows 10-bit A/D conversion with temperature sensor to allow for temperature compensation).

1.4 Development and Evaluation Tools

The PowerCore Tool Kit contains the hardware you need to use your PowerCore module.

- PowerCore Prototyping Board.
- 48 V AC, 1 A center-tapped transformer.
- Programming cable with 10-pin header and DE9 connections, and integrated level-matching circuitry.
- Mounting standoffs.
- ***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.

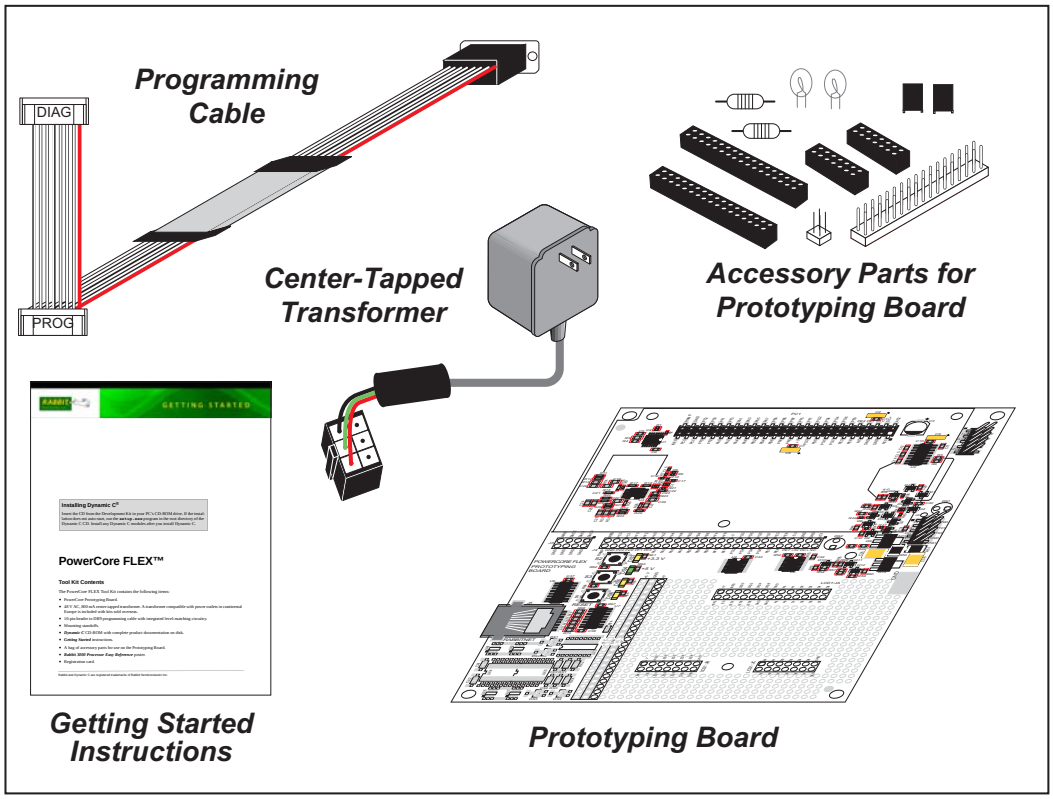


Figure 1. PowerCore Tool Kit

1.4.1 Software

PowerCore FLEX modules are programmed using version 9.20 or later of Rabbit Semiconductor's Dynamic C. A compatible version is included on the Tool Kit CD-ROM.

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

1.4.2 Wi-Fi Add-On Kit

Rabbit Semiconductor also offers a Wi-Fi Add-On Kit for PowerCore FLEX modules consisting of a PowerCore 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 PowerCore module with Wi-Fi capabilities. The PowerCore Interposer Board is placed between the PowerCore module and the PowerCore Prototyping Board so that the CompactFlash Wi-Fi Board, which holds the Wi-Fi CompactFlash card, can be connected to the PowerCore-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.

1.4.3 Online Documentation

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

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

2. GETTING STARTED

This chapter explains how to set up and use a PowerCore module with a PowerCore Prototyping Board.

NOTE: It is assumed that you have a Tool Kit. If you purchased a PowerCore FLEX module by itself, you will have to adapt the information in this chapter and elsewhere to your test and development setup.

2.1 Install Dynamic C

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

If you have not yet installed Dynamic C version 9.20 (or a later version), do so now by inserting the Dynamic C CD from the Tool Kit in your PC's CD-ROM drive. If autorun is enabled, the CD installation will begin automatically.

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

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

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

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

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

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

2.2 Hardware Connections

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

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

2.2.1 Attach Module to Prototyping Board

Turn the PowerCore module so that the notched corner is on the top left as shown in Figure 2 below. Snap in at least one standoff as shown below and then insert the module's J4 header into the PC1 socket on the Prototyping Board. The notched corner at the top left corner of the PowerCore module should face the same direction as the corresponding notch outline below it on the Prototyping Board

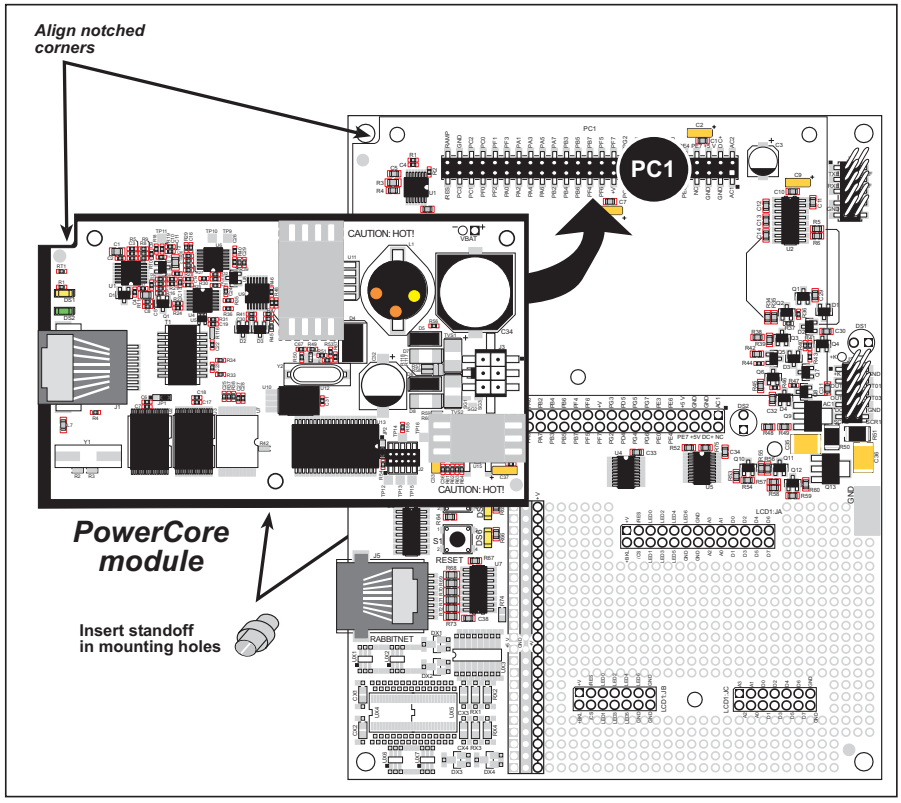


Figure 2. Install the PowerCore Module on the Prototyping Board

NOTE: It is important that you line up the pins on header J4 of the PowerCore module exactly with the corresponding pins of the PC1 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 mis-aligned module is powered up.

Press the standoff into its corresponding hole and press the module's pins firmly into the Prototyping Board socket.

2.2.2 Connect Programming Cable

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

Connect the 10-pin connector of the programming cable labeled **PROG** to header J2 on the PowerCore module as shown in Figure 3. There is a small dot on the circuit board next to pin 1 of header J2. 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 nonprogramming serial connection.)

Attach the DE9 connector end of the programming cable to a COM (serial) port on the PC.

NOTE: Be sure to use the programming cable (part number 101-0542) supplied with this Tool 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 might not work with PowerCore modules.

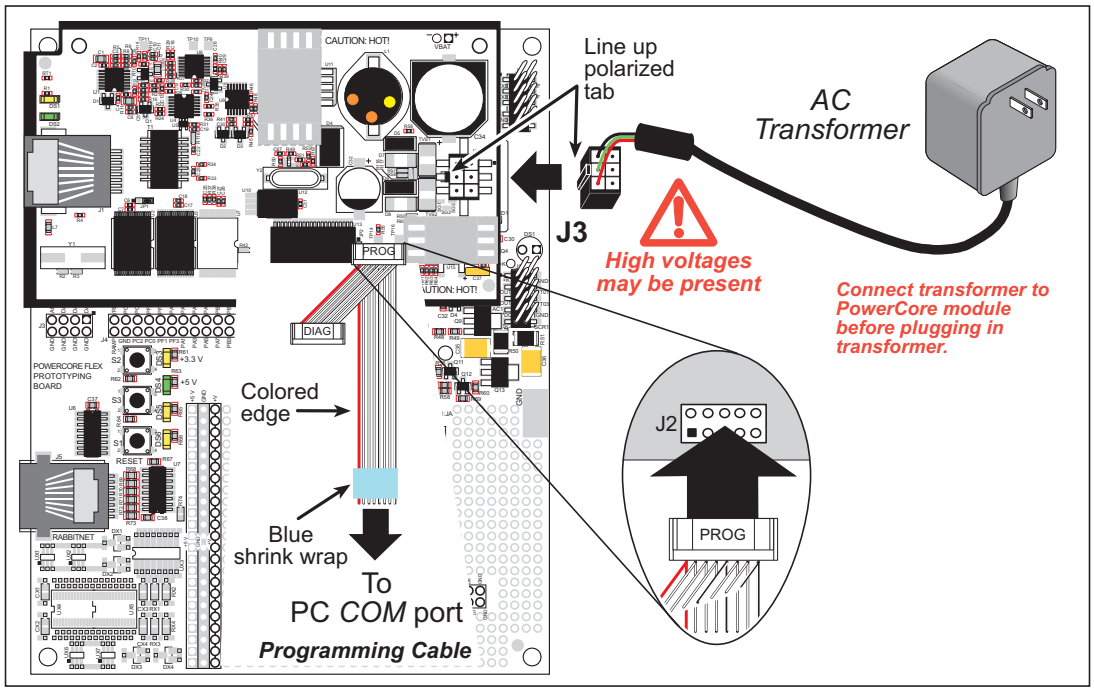


Figure 3. Connect Programming Cable and Power Supply

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 PowerCore FLEX 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, you can connect power to the PowerCore module. In most cases where a locking power connector is stuffed at J3, connect the locking plug from the wall transformer to the J3 locking connector on the PowerCore module, taking care to line up the polarized tab on the locking plug with the locking connector as shown in Figure 3.

Plug in the wall transformer. The DS3 and DS4 yellow and green LEDs on the Prototyping Board located near the **RESET** button should light up. The PowerCore module and the Prototyping Board are now ready to be used.

CAUTION: It is important that power is connected in the order specified in these instructions—connect the locking plug from the wall transformer to the J3 locking connector on the PowerCore module ***before*** you plug in the wall transformer. This is important for your safety because of the high AC voltages present and to avoid possible damage to your PowerCore module.

CAUTION: The heat sinks on the PowerCore module can become very hot. Avoid any contact with them.

If you selected one of the FLEX configuration options (Options 1 and 2 in Appendix D.1.1) for regulated or unregulated DC power supplied from outside the PowerCore module, no locking connector is stuffed at J3 on the PowerCore module. Instead, either regulated +5 V DC is supplied to the PowerCore from your motherboard via pins 5 and 6 of header J4, or unregulated DC power is supplied via pins 1 and 5 of header J4, depending on which option you selected. The voltage range for unregulated DC power is established by your selection of a voltage regulator in Option 2 as explained in Appendix D.1.1.

NOTE: This provision of external DC power is not possible when you are using the PowerCore Prototyping Board, which has no provision to supply powerto the PowerCore module. Any external DC power must come from a motherboard of your own design for use with one of these FLEX options.

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

2.3 Starting Dynamic C

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

If you are using a USB port to connect your computer to the PowerCore module, go to the **Options > Project Options** dialog box and select "Use USB to Serial Converter" on the **Communications** tab.

2.4 Run a Sample Program

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

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

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

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

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

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

- Check that the PowerCore module is powered correctly — the red and yellow LEDs on the Prototyping Board should be lit when the PowerCore module is mounted on the Prototyping Board and the wall transformer is plugged in.
- Check to make sure you are using the **PROG** connector, not the **DIAG** connector, on the programming cable.
- Check both ends of the programming cable to ensure that they are firmly plugged into the PC and the programming port on the PowerCore module.
- Ensure that the PowerCore module is firmly and correctly installed in its socket 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. This manual also provides complete hardware reference information and describes the software function calls for the PowerCore FLEX modules, the Prototyping Board, and the optional LCD/keypad module.

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

2.5.1 Standalone Operation of the PowerCore Module

Once the PowerCore module has been programmed successfully, remove the programming cable from the programming connector and reset the PowerCore module. The PowerCore module may be reset by removing, then reapplying power, or by pressing the **RESET** button on the Prototyping Board if it is connected to the Prototyping Board. The PowerCore module may now be removed from the Prototyping Board for end-use installation.

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

2.5.2 Technical Support

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

3.1 Introduction

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

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

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

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

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, and then compile and run the sample program by selecting **Run** in the **Run** menu (or press **F9**). The PowerCore module must be in Program Mode (see Figure 8) and must be connected to a PC using the programming cable.

More 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 PowerCore FLEX modules. Sample programs illustrating the general operation of the PowerCore FLEX modules, serial communication, the serial flash, A/D conversion, D/A conversion, and hardware accessories such as a temperature sensor and triacs are provided in the **SAMPLES\PowerCoreFLEX** 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 PowerCore module must be installed on the Prototyping Board when using the sample programs described in this chapter. TCP/IP sample programs are described in Chapter 6, "Using the TCP/IP Features," and sample programs for the optional LCD/keypad module are described in Appendix C.

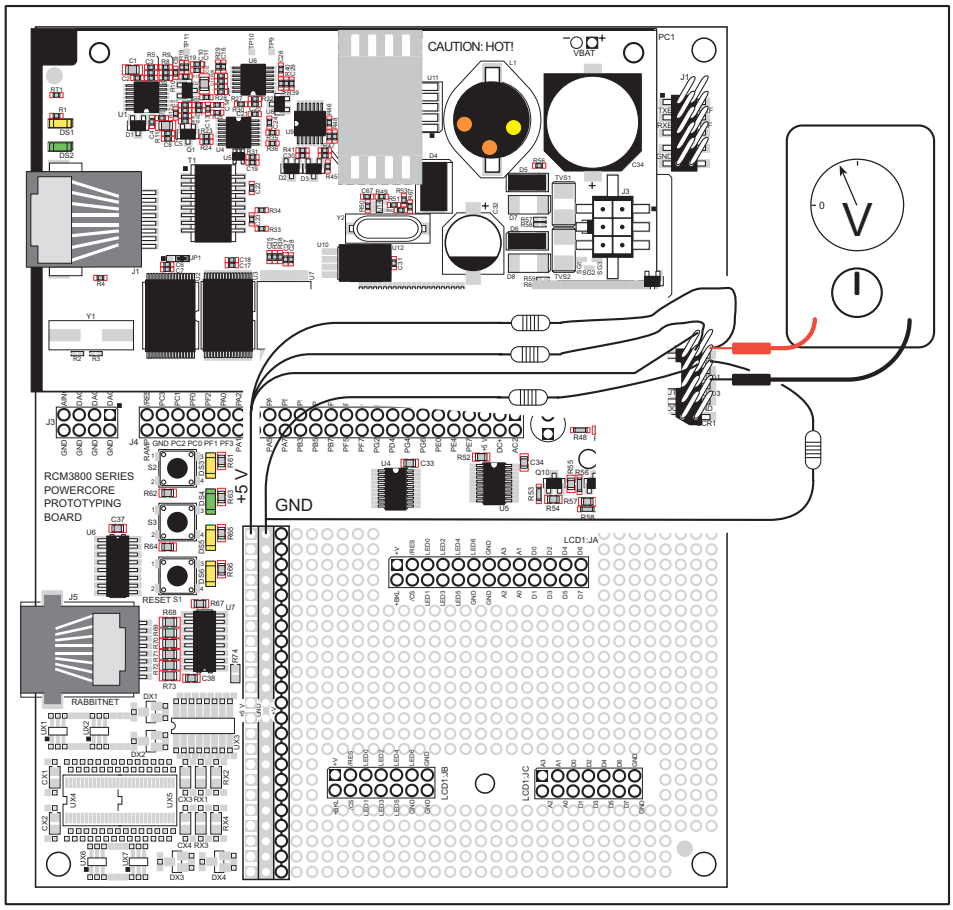
3.2.1 I/O

The following sample programs can be found in the **SAMPLES\PowerCoreFLEX\IO** folder.

- **DIGIN.c**—Demonstrates how to read digital inputs using the **digIn** function call. Press switches S2 and S3 on the Prototyping Board to change the logic levels for IN0 and IN1 as displayed in the Dynamic C **STDIO** window.
- **DIGOUT.c**—Demonstrates the control of sinking and sourcing digital outputs. Connect 100 Ω load resistors from the +5 V supply at pin 6 on header J4 or from the ground at pin 2 on header J2 as indicated in the table below.

Digital Output		Load Resistor	+K
Sinking	OUT0	+5 V to OUT0 (+5 V to J2 pin 3)	+K to +5 V (J2 pin 1 to +5 V)
	OUT1	+5 V to OUT1 (+5 V to J2 pin 4)	
Sourcing	OUT2	GND to OUT2 (GND to J2 pin 5)	+K to + 5 V (J2 pin 1 to +5 V)
	OUT3	GND to OUT3 (GND to J2 pin 6)	

These connections are illustrated below.



After you compile and run this program, select an output channel and logic level via the Dynamic C **STDIO** window. You can see the logic level with a voltmeter connected to the output channel you selected.

- **`LED.c`**—Demonstrates the use of the digital inputs by turning LEDs DS5 and DS6 on the Prototyping Board on and off.

3.2.2 A/D Converter

The following sample programs can be found in the **SAMPLES\PowerCoreFLEX\ADC** folder.

- **ADC_CALIB_EXTERNAL.c**—Demonstrates how to calibrate an external A/D channel using two known voltages to generate two coefficients, gain and offset, which will be written to the simulated EEPROM in the flash memory. The program will then display the voltage that has been calibrated.

  In order to run this sample program, you will need a separate power supply. Connect the positive lead from the power supply to the A/D channel input located on pin 7 of header J3 on the Prototyping Board. Connect the negative lead from the power supply to the GND located on pin 8 of J3 on the Prototyping Board.

- **ADC_CALIB_RAMP.c**—Demonstrates how to calibrate the A/D ramp-generator circuit. The A/D ramp-generator circuit is calibrated using the 2.5 V voltage reference and the end-of-ramp voltage point located on the PowerCore module. The calibration constants generated by this sample program will be written to the simulated EEPROM in the flash memory, and can then be used to calculate the voltage for the following A/D channels.

  Channel 0—2.5 V reference voltage
  Channel 1—end-of-ramp voltage
  Channel 2—thermistor

  The program will then display the voltage of these A/D channels.

  In order to run this sample program, you will need a voltmeter to measure the A/D 2.5 V voltage reference at TP10 and the end-of-ramp voltage at TP9; both test points are located between header J4 and the edge on the bottom side of the PowerCore module.

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

- **ADC_MUX_EXTERNAL1.c**—Demonstrates how to enable the A/D interrupt MUX routine for designs with multiple external A/D channels. This sample program will also read and display the voltage of the external A/D channel that is located on pin 7 of header J3 on the PowerCore Prototyping Board.

  Since there is only one A/D converter channel on the Prototyping Board, the MUX control is simulated by using LED DS5 on the Prototyping Board, which is controlled by PD5.

  LED = OFF = A/D channel 0
  LED = ON = A/D channel 1.

  The following steps explain how to implement round-robin MUX control.

  1. Define the function prototype.

     ```
     void adc_mux(void);
     ```

  2. Define **ADC_MUX_CNTRL** as follows to call the **adc_mux** routine.

     ```
     #define ADC_MUX_CNTRL call adc_mux
     ```

  3. Define the number of A/D channels. Change the default from 1 to the number of A/D channels in your design.

     ```
     #define MAX_ADCHANNELS <Number Channels>
     ```

  4. Create the **adc_mux** assembly routine as follows.

     ```
     #asm root nodebug
     adc_mux::
     // Insert Mux control code
     ret
     #endasm
     ```

     This routine will be called automatically by the low-level A/D converter driver.

  5. Add I/O initialization code at the beginning of your application program for the port pins used for MUX control.

  In order to run this sample program, you will need a separate power supply. Connect the positive lead from the power supply to the A/D channel input located on pin 7 of header J3 on the Prototyping Board. Connect the negative lead from the power supply to the GND located on pin 8 of J3 on the Prototyping Board.

- **ADC_MUX_EXTERNAL2.c**—Demonstrates how to implement MUX control to read external A/D channels randomly in your application program. This sample program will also read and display the voltage of the external A/D channel that is located on pin 7 of header J3 on the PowerCore Prototyping Board.

  Since there is only one A/D converter channel on the Prototyping Board, the MUX control is simulated by using LED DS5 on the Prototyping Board, which is controlled by PD5.

  LED = OFF = A/D channel 0
  LED = ON = A/D channel 1.

  The following steps explain how to implement the MUX routine for random A/D channel selection.

  1. Define the number of A/D channels. Change the default from 1 to the number of A/D channels in your design.

     ```
     #define MAX_ADCHANNELS <Number Channels>
     ```

  2. Provide an application MUX function that does the following:
     - A/D converter channel selection via your hardware MUX circuit.
     - Change the **_adc_mux_channel** index to the channel selected.
     - Add delay for the channel switching settling time.
     - Set the **adc_conversion_done** flag to FALSE.
  3. Add I/O initialization code at the beginning of your application program for the port pins used for MUX control.
  4. Then:
     - Call your routine with the A/D converter channel selected.
     - Wait in a nonblocking wait routine for the **adc_conversion_done** flag to become TRUE.
     - Read the A/D converter.
     - Repeat the sequence.

  In order to run this sample program, you will need a separate power supply. Connect the positive lead from the power supply to the A/D channel input located on pin 7 of header J3 on the Prototyping Board. Connect the negative lead from the power supply to the GND located on pin 8 of J3 on the Prototyping Board.

- **ADC_RD_EXTERNAL.c**—Demonstrates how to read and display the voltage of the external A/D converter located on the Prototyping Board. The voltage is calculated from coefficients read from the simulated EEPROM in flash memory. The external A/D channel is located on pin 7 of header J3 on the PowerCore Prototyping Board.

  In order to run this sample program, you will need a separate power supply. Connect the positive lead from the power supply to the A/D channel input located on pin 7 of header J3 on the Prototyping Board. Connect the negative lead from the power supply to the GND located on pin 8 of J3 on the Prototyping Board.

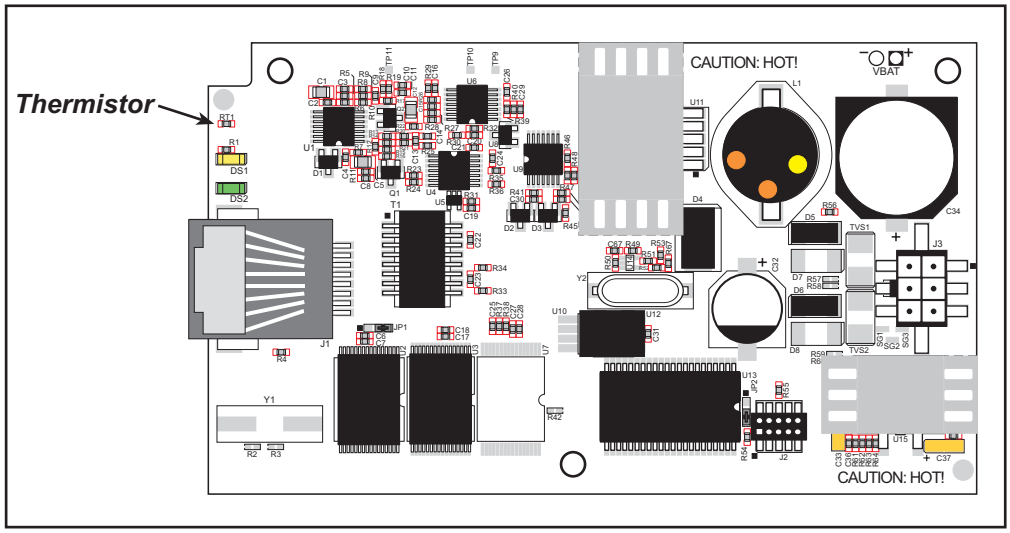
- **ADC_RD_RAMP.c**—Demonstrates how to read and display the voltage of the following PowerCore ramp-generator A/D channels.

  Channel 0—2.5 V reference voltage
  Channel 1—end-of-ramp voltage
  Channel 2—thermistor

  The voltage is calculated using coefficients read from the simulated EEPROM in flash memory that were written during the calibration process.

The following sample programs can be found in the **SAMPLES\PowerCoreFLEX\TEMPERATURE** folder.

- **THERMISTOR.c**—Demonstrates how to read the thermistor sensor on the PowerCore module. Once you are running the sample program, you will be able to use the Dynamic C **STDIO** window to observe the temperature change across the thermistor as you apply hot or cold air to the thermistor. Note that the thermistor is on the PowerCore circuit board, and so it will actually be measuring the temperature of the circuit board and the air immediately above it. This temperature will likely be higher than the ambient temperature a short distance from the board.



- **THERMOFFSET.c**—Demonstrates how to add an offset to set the thermistor sensor on the PowerCore module to an external reference temperature.

3.2.3 D/A Converter

The following sample programs can be found in the **SAMPLES\PowerCoreFLEX\DAC** folder.

- **DAC_CAL.c**—Demonstrates how to calibrate a D/A converter channel using two known voltages to generate the two coefficients, gain and offset, which will be written to the simulated EEPROM in flash memory. This sample program must be compiled to flash memory.

  In order to run this sample program, you will need a voltmeter that is connected to the D/A converter output channels and ground located at header position J3 on the Prototyping Board.

  NOTE: This sample program will overwrite any existing calibration constants.

- **DAC_VOLT.c**—Outputs a voltage that can be read with a voltmeter. The output voltage is computed using the calibration constants that are read from the simulated EEPROM in flash memory. This sample program must be compiled to flash memory.

  In order to run this sample program, you will need a voltmeter that is connected to the selected D/A converter output channel and ground located at header position J3 on the Prototyping Board.

3.2.4 Use of Serial Flash

The following sample programs can be found in the **SAMPLES\PowerCoreFLEX\SERIAL_FLASH** folder.

- **SFLASH_PATTERN_INSPECT.c**—When the sample program starts running, it attempts to initialize a serial flash chip on the PowerCore module. Once a serial flash chip is found, the sample program writes a pattern to the first 100 pages. The Dynamic C **STDIO** window will then display information about the page size. The user can then either print out the contents of a specified page or clear (set to zero) all the bytes in a specified page.

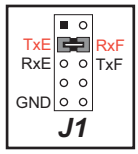
- **SFLASH_TEST.c**—This sample program tests a serial flash chip by performing a write and a read. The results of the test are displayed in the Dynamic C **STDIO** window.

3.2.5 Serial Communication

The following sample programs can be found in the **SAMPLES\PowerCoreFLEX\RS232** folder.

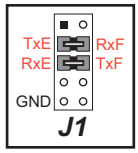
- **PARITY.C**—This program demonstrates the use of parity modes by repeatedly sending byte values 0–127 from Serial Port E to Serial Port F. The program will switch between generating parity or not on Serial Port E. Serial Port F will always be checking parity, so parity errors should occur during every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

  Before running this sample program, jumper TXE to RXF located on pins 3 and 4 on header J1 of the Prototyping Board.

  NOTE: For the sequence that does get parity errors, the errors won't occur for each byte received. This is because certain byte patterns and the stop bit will appear to generate the correct parity for the UART.

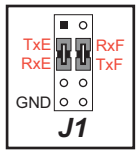
- **SIMPLE3WIRE.C**—This program demonstrates basic RS-232 serial communication whose loopback is displayed in the Dynamic C **STDIO** window.

  Before running this sample program, jumper TXE to RXF located on pins 3 and 4 on header J1 of the Prototyping Board. Then jumper RXE to TXF located on pins 5 and 6 on header J1 of the Prototyping Board.

- **SIMPLE5WIRE.C**—This program demonstrates 5-wire RS-232 serial communication whose loopback is displayed in the Dynamic C **STDIO** window.

  Before running this sample program, jumper TXE to RXE located on pins 3 and 5 on header J1 of the Prototyping Board. Then jumper RXF to TXF located on pins 4 and 6 on header J1 of the Prototyping Board.

  Once you compile and run this sample program, TXF and RXF will become the RTS and CTS flow control. To test the flow control, remove the jumper on J1 pins 4 and 6, which will cause the characters to stop printing in the Dynamic C **STDIO** window, and will resume printing when you reinstall the jumper.

3.2.6 Triacs

The following sample programs can be found in the **SAMPLES\PowerCoreFLEX\TRIAC** folder.

3.2.6.1 Phase-Angle Triac Control

The sample programs demonstrate phase-angle triac control for the PowerCore module and its Prototyping Board. Phase-angle triac control provides you with the ability to fire a triac at a given phase angle of a 50/60 Hz sine wave to provide the desired control for your hardware application.

Once one of the following sample programs is compiled and is running, select the desired triac and the phase angle of where to fire the selected triac in the Dynamic C **STDIO** window.

You may use an oscilloscope or some other load circuit such as the incandescent lamps provided in the Tool Kit to monitor the output. If you are using an oscilloscope, monitor the triac control pin to see the control pin going active at the phase angle from 0 to 180 that you selected. The control pins are located on header J4 of the Prototyping Board:

Pin 41—PF2_SCR-0
Pin 40—PF3_SCR-1

If you are not using an oscilloscope, solder in the incandescent lamps provided in the Tool Kit at DS1 and DS2 on the Prototyping Board before running any of these sample programs. Select option 7 (ramping triacs) from the Dynamic C **STDIO** window to monitor the lamps visually.

DS1—PF2_SCR-0
DS2—PF3_SCR-1

You should see lamps go from being fully ON, then dimming down to fully OFF.

- **TRIAC_PHASE.c**—This program demonstrates basic phase-angle triac control.
- **TRIAC_PHASE_ADC.c**—This program demonstrates phase-angle triac control and reading the A/D ramp circuit when you use the PowerCore module and its Prototyping Board. The Dynamic C **STDIO** window will show the A/D ramp output voltages for the 2.5 V reference voltage, the end-of-ramp-voltage, and the thermistor.
- **TRIAC_PHASE_FLASH.c**—This program demonstrates phase-angle triac control by writing to the flash memory (but ***not*** the serial flash memory).

3.2.6.2 Time-Proportional Triac Control

The sample programs demonstrate time-proportional triac control for the PowerCore module and its Prototyping Board. Time-proportional triac control provides you with the ability to control a triac over a fixed number of 50/60 Hz cycles. By setting the triac ON/OFF ratio you will be able to achieve the desired control for your hardware application.

Once one of the following sample programs is compiled and is running, set the ON/OFF ratio of and select the triac as prompted in the Dynamic C **STDIO** window.

You may use an oscilloscope or some other load circuit such as the incandescent lamps provided in the Tool Kit to monitor the output. If you are using an oscilloscope, monitor the triac control pin to see the triacs cycle according to the ON/OFF ratio you selected. The control pins are located on header J4 of the Prototyping Board:

Pin 41—PF2_SCR-0
Pin 40—PF3_SCR-1

If you are not using an oscilloscope, solder in the incandescent lamps provided in the Tool Kit at DS1 and DS2 on the Prototyping Board before running any of these sample programs. Select option 1 (ramping triacs) from the Dynamic C **STDIO** window to monitor the lamps visually.

DS1—PF2_SCR-0
DS2—PF3_SCR-1

- **`TRIAC_RATIO.c`**—This program demonstrates basic time-proportional triac control.
- **`TRIAC_RATIO_ADC.c`**—This program demonstrates time-proportional triac control and reading the A/D ramp circuit when you use the PowerCore module and its Prototyping Board.

  If you installed the incandescent lamps, you should see the lamps cycle ON/OFF according to the ON/OFF ratio you selected. When you first activate the lamps, DS2 will blink twice as fast as DS1.

  The Dynamic C **STDIO** window will show the A/D ramp output voltages for the 2.5 V reference voltage, the end-of-ramp-voltage, and the thermistor.
- **`TRIAC_RATIO_FLASH.c`**—This program demonstrates time-proportional triac control by writing to the flash memory (but ***not*** the serial flash memory).

  When you run this sample program, you will be able to monitor the triacs cycling OFF and ON, incrementing automatically over ratio ranges of 0/5 (= OFF) to 5/5 (= ON, either DS1 or pin 41 of Prototyping Board header J4) and 0/10 (= OFF) to 10/10 (= ON, either DS2 or pin 40 of Prototyping Board header J4).

3.2.7 TCP/IP

TCP/IP sample programs and complete instructions on how to run them are provided in Chapter 6, “Using the TCP/IP Features.”

3.2.8 LCD/Keypad Module

Appendix C, “LCD/Keypad Module,” provides sample programs for the optional LCD/keypad module, and includes instructions on how to run them.

4. HARDWARE REFERENCE

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

Figure 4 shows the Rabbit-based subsystems designed into the PowerCore FLEX modules.

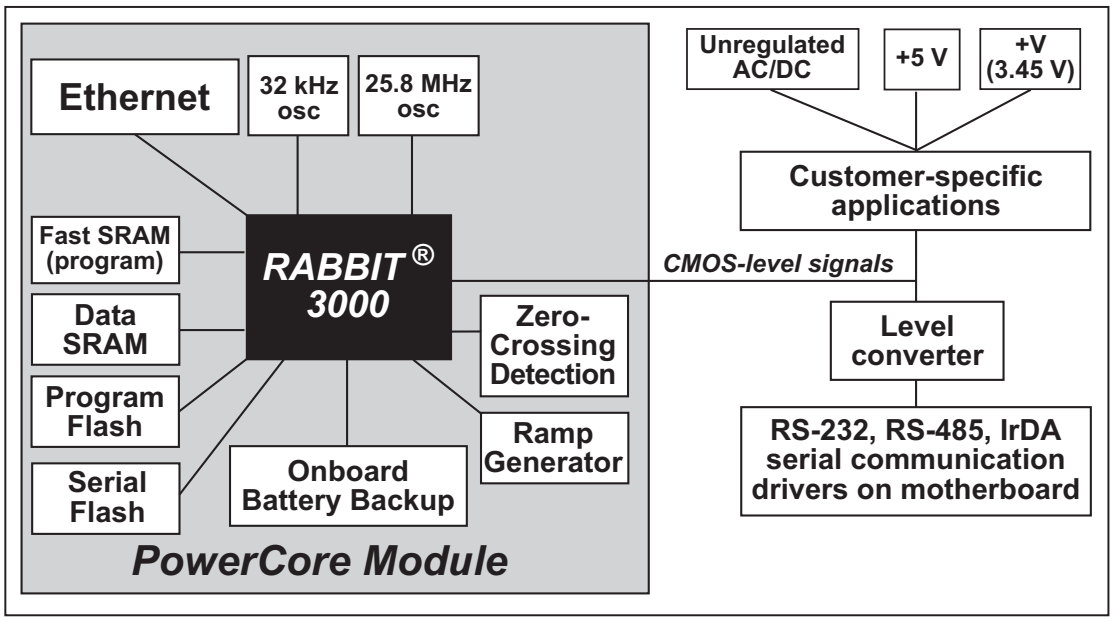


Figure 4. PowerCore Subsystems

4.1 PowerCore Digital Inputs and Outputs

Figure 5 shows the PowerCore pinout for header J4.

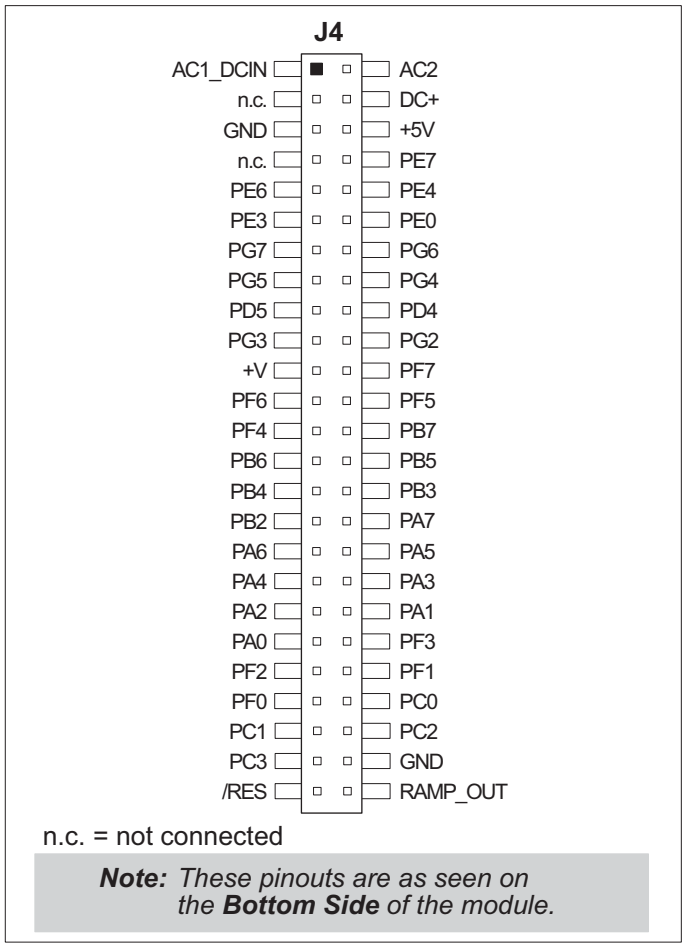


Figure 5. PowerCore Pinout

Header J4 is a standard 2 x 25 IDC header with a nominal 0.1" pitch.

Figure 6 shows the use of the Rabbit 3000 microprocessor ports in the PowerCore module.

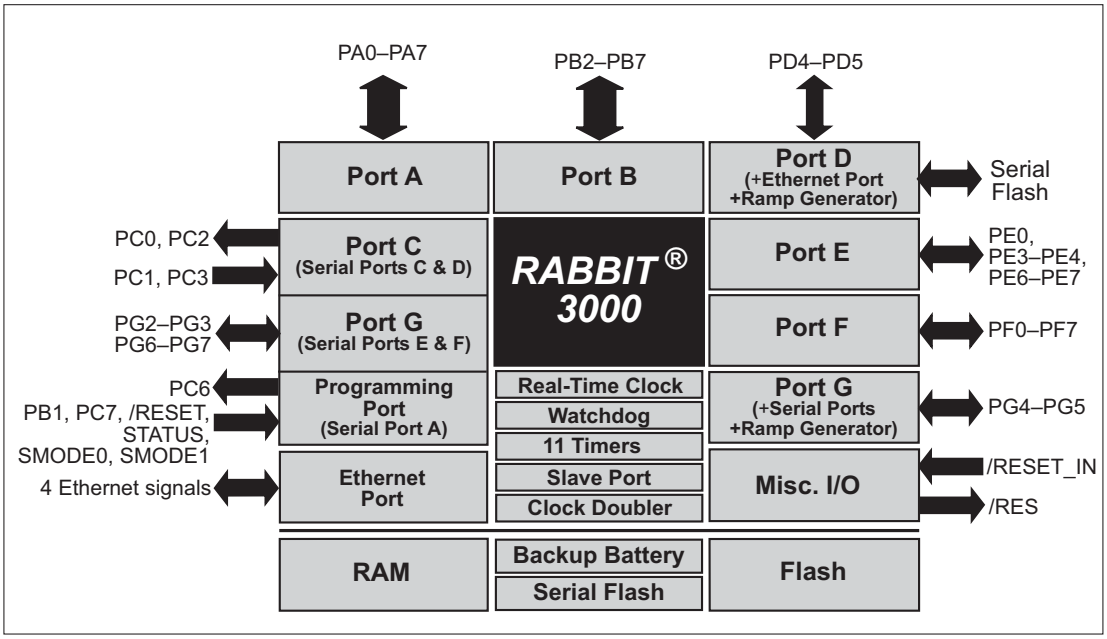


Figure 6. Use of Rabbit 3000 Ports

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

Table 2. PowerCore Pinout Configurations

Pin		Pin Name	Default Use	Alternate Use		Notes
Header J4	1	AC1_DCIN				
	2	AC2				
	3	not connected				
	4	DC+				
	5	GND				
	6	+5 V				
	7	not connected				
	8	PE7	Input/Output	I7 /SCS		I/O Strobe 7 Slave Port Chip Select
	9	PE6	Input/Output	I6		I/O Strobe 6
	10	PE4	Input/Output	I4 INT0B		I/O Strobe 4 Interrupt 0B
	11	PE3	Input/Output	I3		I/O Strobe 3
	12	PE0	Input/Output	I0 INT0A		I/O Strobe 0 Interrupt 0A
	13	PG7*	Input/Output	RXE	Serial Port E	
	14	PG6	Input/Output	TXE		
	15	PG5*	Input/Output	RCLKE		Serial Clock E input
	16	PG4	Input/Output	TCLKE		Serial Clock E ouput
	17	PD5	Input/Output	ARXB	Alternate Serial Port B	
	18	PD4	Input/Output	ATXB		
	19	PG3	Input/Output	RXF	Serial Port F	
	20	PG2	Input/Output	TXF		
	21	+V				3.45 V
	22	PF7*	Input/Output	AQD2A PWM3		
	23	PF6	Input/Output	AQD2B PWM2		
	24	PF5*	Input/Output	AQD1A PWM1		
	25	PF4	Input/Output	AQD1B PWM0		

* This pin may be used for input capture if not used for anything else.

Table 2. PowerCore Pinout Configurations (continued)

Pin		Pin Name	Default Use	Alternate Use		Notes
Header J4	26	PB7	Input/Output	IA5 /SLAVEATTN		External Address 5 Slave Attention
	27	PB6	Input/Output	IA4		External Address 4
	28	PB5	Input/Output	IA3 SA1		External Address 3 Slave port Address 1
	29	PB4	Input/Output	IA2 SA0		External Address 2 Slave port Address 0
	30	PB3	Input/Output	IA1 /SRD		External Address 1 Slave port read
	31	PB2	Input/Output	IA0 /SWR		External Address 0 Slave port write
	32–39	PA[7:0]	Parallel I/O	External data bus (ID0–ID7) Slave port data bus (SD0–SD7)		External Data Bus
	40	PF3*	Input/Output	QD2A		
	41	PF2	Input/Output	QD2B		
	42	PF1*	Input/Output	QD1A CLKC		
	43	PF0	Input/Output	QD1B CLKD		
	44	PC0	Output	TXD	Serial Port D	
	45	PC1	Input	RXD		
	46	PC2	Output	TXC	Serial Port C	
	47	PC3	Input	RXC		
	48	GND				
	49	/RES	Reset input			Reset output from Reset Generator
	50	RAMP_OUT	Ramp output			Ramp generator output

* This pin may be used for input capture if not used for anything else.

4.1.1 Internal and External Buses

The Rabbit 3000 address lines (A0–A19) and all the data lines (D0–D7) are routed internally to the onboard flash memory, SRAM, and Ethernet chips. These lines do not appear on header J4. It would be undesirable (and unnecessary) to run these lines to the motherboard as they must operate at high frequencies and any additional load capacitance would be undesirable.

A completely separate I/O bus is implemented using eight data lines and six address lines. Various strobes can be implemented to clock data to or from the bus. The I/O bus is an option that can be enabled by the user's program as explained below. The eight bidirectional I/O lines share pins with Parallel Port A, and the six address lines share pins with part of Parallel Port B. Although only 64 read or write addresses are available directly, it easy to expand the register space of the bus to be as large as desired by adding additional address bits implemented using a register loadable as one of the 64 registers. Another approach is to use additional separate strobe lines to create additional 64-register spaces. Rabbit I/O instructions are used to access the registers created on the I/O bus. Strobes are enabled by software setup of individual pins on Parallel Port E. More details are available in the ***Rabbit 3000 Users' Manual***.

Parallel Port A can also be used as an external I/O data bus to isolate external I/O from the main data bus. The pins on Parallel Port B used as I/O bus address lines can be used as individual lines when the I/O bus is not enabled. Parallel Port B pins PB2–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
```

4.1.1.1 Handling Stateful I/O Registers

I/O registers are often either readable or writable, but not both in order to save hardware expense or because the functionality does not fit a read/write model. For example, if writing a certain bit in a register causes a momentary action to take place, the bit in the register does not really exist and there is nothing to read back.

If an I/O register is stateful, it is a write register that holds bits that have a continuing meaning over time. If this information has to be changed and restored at different priority levels, for example, in a main program and in an interrupt routine, there must be a way to read the contents of the register and restore it. One approach is to make the register readable, but this requires extra hardware. The other approach is to establish a shadow register in memory that holds an echo of the register contents—the shadow register is loaded each time the register is loaded. Care must be taken with shadow registers to ensure that an interrupt cannot take place when the contents of the shadow register differ from those of the I/O register. The ***Rabbit 3000 Users' Manual*** provides additional information.

4.1.2 Other Inputs and Outputs

The status, /RESET_IN, SMODE0, and SMODE1 I/O are normally associated with the programming port. Since the status pin is not used by the system once a program has been downloaded and is running, the status pin can then be used as a general-purpose CMOS output. The programming port is described in more detail in Section 4.2.3.

The /RES pin on header J4 is a bidirectional signal that can be used to reset the microprocessor and external peripheral devices. The ramp-generator output on header J4 can be used for A/D measurements.

PF1, PF3, PF5, PF7, PG5, and PG7 (pins 42, 40, 24, 22, 15, and 13 on header J4) may be used for two input capture channels if they are not being otherwise used.

4.1.3 LEDs

The PowerCore module has two status LEDs located beside the RJ-45 Ethernet jack.

The yellow LED at DS1 indicates network activity.

The green LED at DS2 indicates that the PowerCore module is connected to a working network.

4.2 Serial Communication

The PowerCore module does not have any serial transceivers directly on the board. However, a serial interface may be incorporated into the board the PowerCore module is mounted on. For example, the PowerCore Prototyping Board has RS-422 and RS-232 transceiver chips.

4.2.1 Serial Ports

There are six serial ports on the Rabbit 3000—Serial Ports A, B, C, D, E, and F. All six serial ports can operate in an asynchronous mode up to the baud rate of the system clock divided by 8. An asynchronous port can handle 7 or 8 data bits. A 9th bit address scheme, where an additional bit is sent to mark the first byte of a message, is also supported.

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

Serial Port B is used to communicate with the serial flash on the PowerCore module, and so is not available for any other applications.

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 E can also be configured for SDLC.

The three serial ports that support clocked serial communications, Serial Ports A, C, and D, are suitable for interfacing with “SPI” devices.

4.2.2 Ethernet Port

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

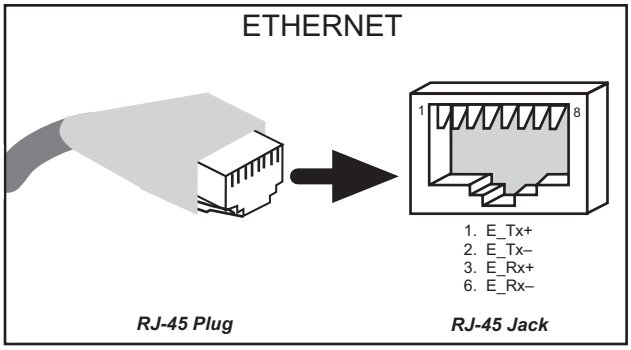


Figure 7. RJ-45 Ethernet Port Pinout

Two LEDs are placed next to the RJ-45 Ethernet jack, one to indicate a live Ethernet link (DS2 green) and one to indicate Ethernet activity (DS1 yellow).

4.2.3 Programming Port

The PowerCore module's programming port is accessed using header J2 or when programming with a RabbitLink EG2110 through the Ethernet jack. 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 PowerCore module after a reset.
- Remotely download and debug a program over an Ethernet connection using the RabbitLink EG2110.
- Fast copy designated portions of flash memory from one Rabbit-based board (the master) to another (the slave) using the Rabbit Cloning Board.

Alternate Uses of the Programming Port

All three clocked Serial Port A signals are available as

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

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

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

The two startup mode pins determine what happens after a reset—the Rabbit 3000 is either cold-booted or the program begins executing at address 0x0000. These two SMODE pins can be used as general inputs once the cold boot is complete.

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_IN pin is an external input that is used to reset the Rabbit 3000 and the PowerCore module's onboard peripheral circuits.

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

4.3 Programming Cable

A special programming cable is used to connect the PowerCore module's programming port to a serial COM port on the PC that hosts Dynamic C. The programming cable has a small circuit board in the cable that converts the RS-232 voltage levels used by the PC serial port to the CMOS voltage levels used by the Rabbit 3000. An additional adapter and software is available so that a PC USB port can be used instead of a serial port.

When the **PROG** connector on the programming cable is connected to header J2 on the PowerCore module, 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 PowerCore module with the PowerCore module 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 PowerCore module 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.

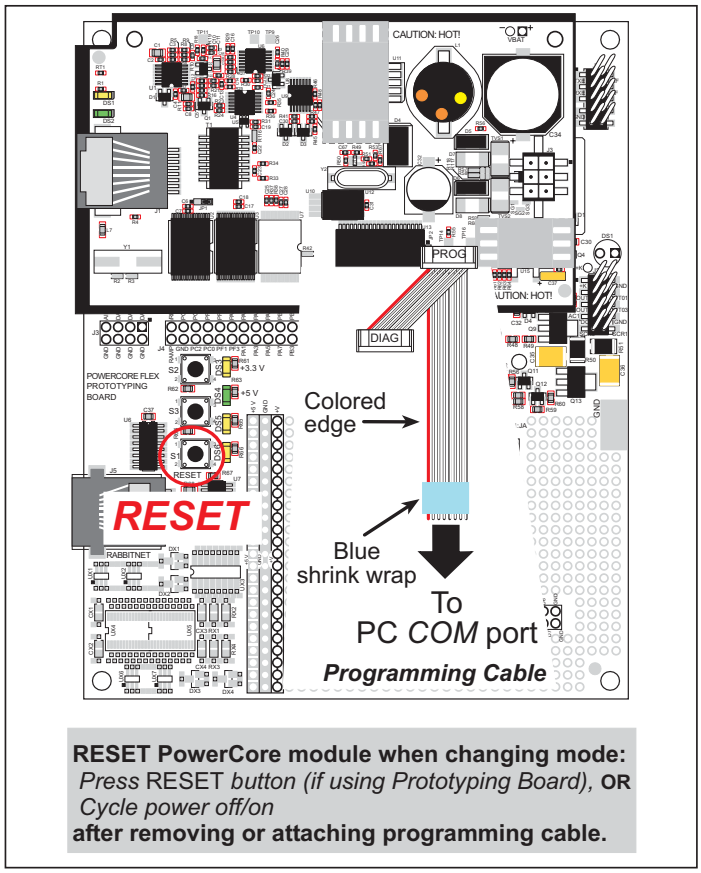


Figure 8. Switching Between Program Mode and Run Mode

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

4.4 Ramp Generator

The PowerCore module has an onboard ramp generator that provides a a continuous sawtooth function with a precision rising ramp. The calibration of the ramp is tied to an onboard 2.5 V voltage reference. The 400 Hz ramp has a linear rise time from 0 to 3.1 V of approximately 1.9 ms, and ramps down in approximately 0.45 ms. (The ramp actually starts at a slightly negative voltage of approximately -0.05 V.) The ramp output has a linearity of about 0.1% and is available for external circuits on pin 50 of header J4.

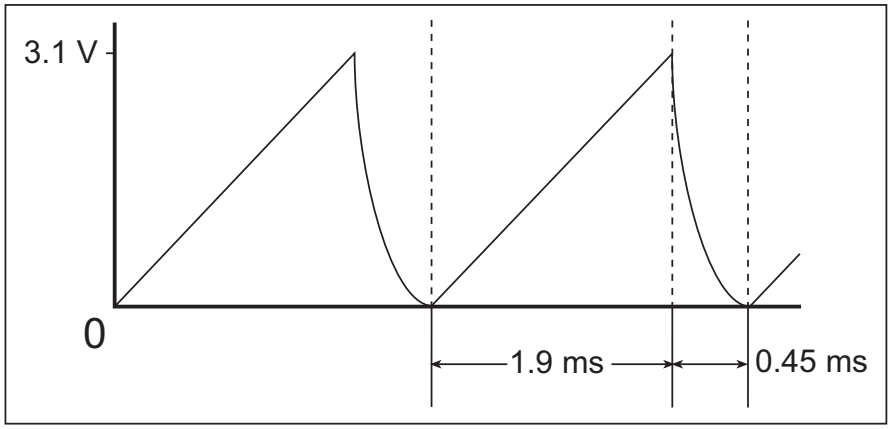


Figure 9. PowerCore Ramp Generator Signal

The onboard 2.5 V reference circuit is shown in Figure 10.

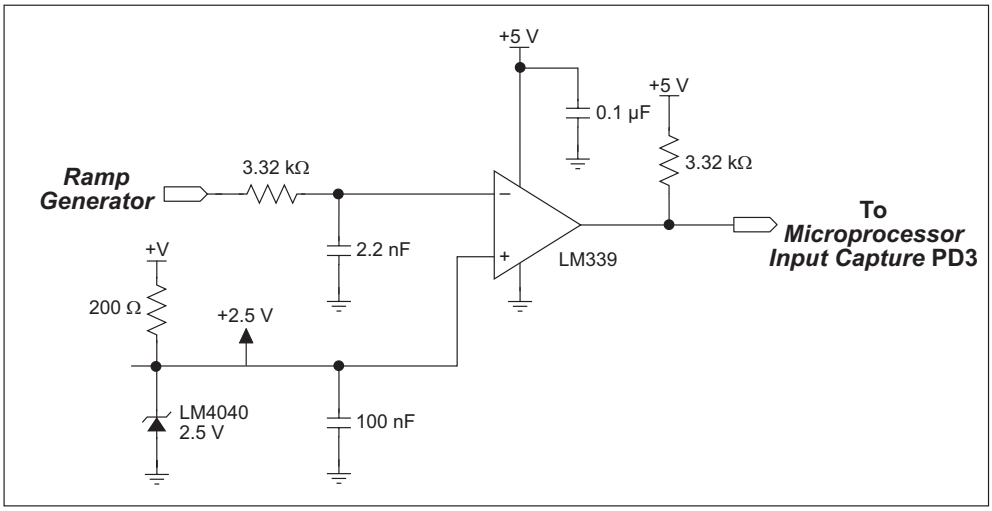


Figure 10. Onboard 2.5 V Reference

The ramp generator makes it possible to measure analog voltages using LM339 comparators and the pulse capture capabilities of the Rabbit 3000 microprocessor to convert time into voltage. One example of this analog measurement capability is the onboard thermistor, which allows temperature measurements of the PowerCore circuit board.

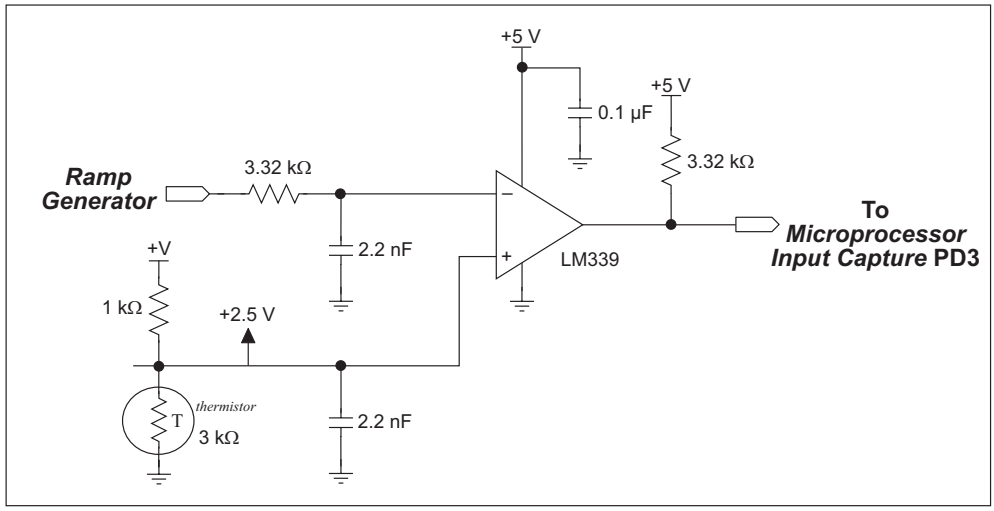


Figure 11. PowerCore Temperature-Measurement Circuit

A similar, more general A/D measurement circuit can be designed. An example is the A/D measurement circuit on the PowerCore Prototyping Board, which is shown in Figure 12. The circuit is designed to accept input voltages of 0–10 V.

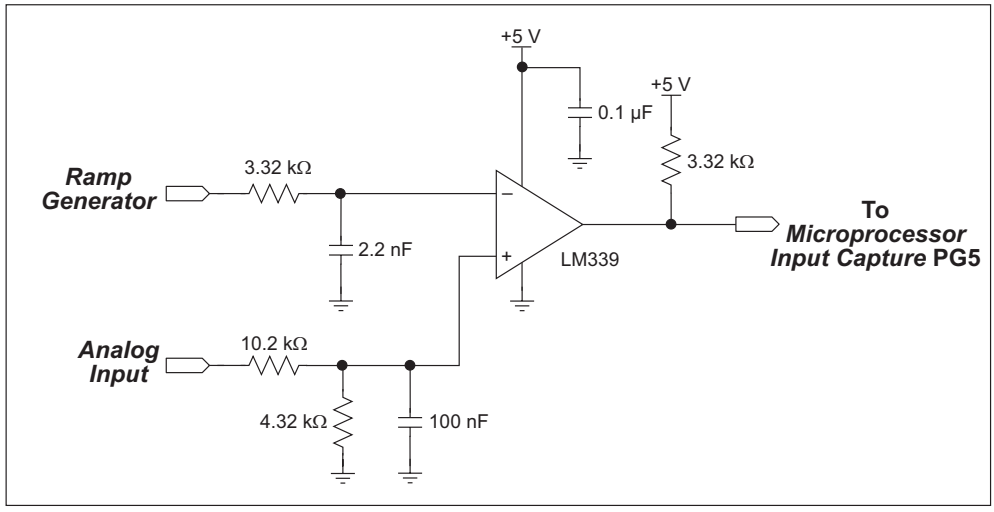


Figure 12. PowerCore Prototyping Board A/D Acquisition Circuit

An A/D converter measurement is implemented when a signal transition from the comparator is routed to an input capture on the Rabbit 3000 (for example, PD3 for the temperature-measurement circuit and PG5 for the A/D measurement circuit on the Prototyping Board). The counter in the Rabbit 3000 starts counting at the beginning of the ramp (PG1), and stops when the ramp crosses the input signal (PG5). The full scale is approximately 4095 counts, which yields a measurement resolution of 12 bits. The end of the ramp activates an interrupt to the Rabbit 3000, which then retrieves the count and stores it so that it can be accessed by a software function call. Generally, additional channels will be measured in

succession. The interrupt routine can be set up to average inputs and to detect out-of-range signals.

A/D conversion can be provided sequentially on additional channels by adding a multiplexer between the comparators and the Rabbit 3000 PG5 input capture pin.

At the same time that an A/D conversion takes place, the PowerCore module exercises one-third of a calibration routine where the 2.5 V reference is used to recalibrate the A/D converter every 7.5 ms. Part of the calibration routine reads the thermistor, whose temperature reading is also available.

This A/D conversion can be used to monitor slow-moving analog voltage outputs such as those from sensors or potentiometers.

4.4.1 Ramp Generator Theory of Operation

Figure 13 shows the comparator outputs for the ramp generator and the synchronizing signals.

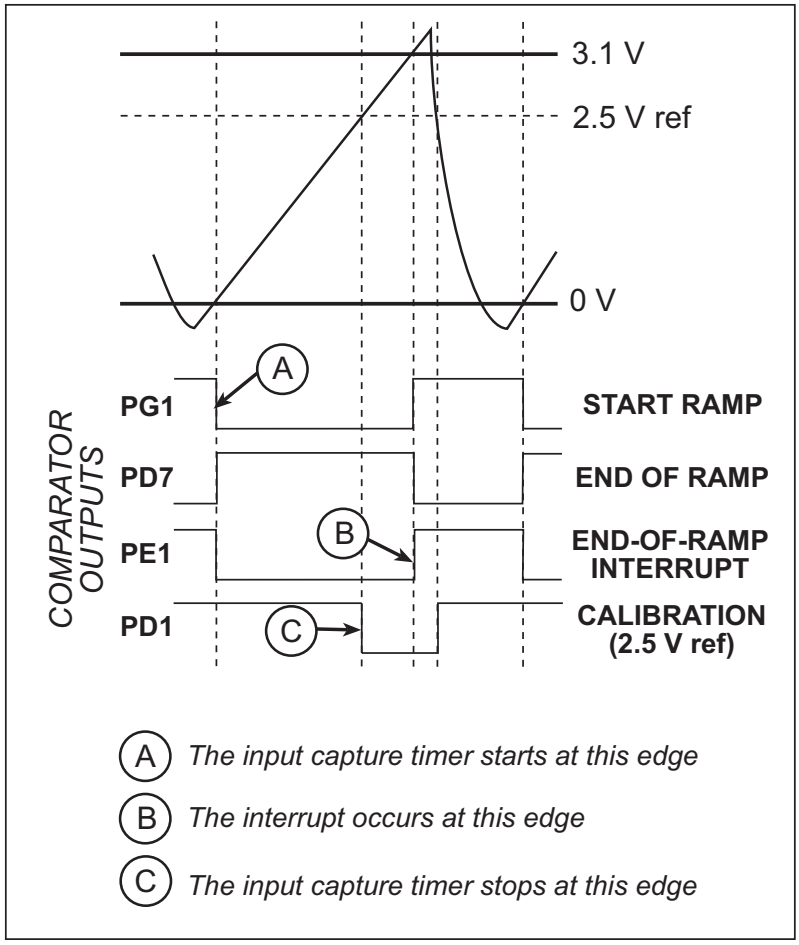


Figure 13. Ramp Generator and Synchronizing Signals from Comparators

The input capture software runs as an interrupt routine that is triggered by the end-of-ramp interrupt. The interrupt routine will read the time delay from the start of the ramp to the signal measured and store it in a memory table. The software will then set up the next measurement, generally by switching an *n* to 1 multiplexer to direct the next comparator output to the pulse capture stop input. The reference voltage is measured to calibrate the slope of the ramp. The pulse capture feature of the Rabbit 3000 microprocessor is used to measure the time between the start of the ramp and when the ramp reaches the analog input voltage that is being measured.

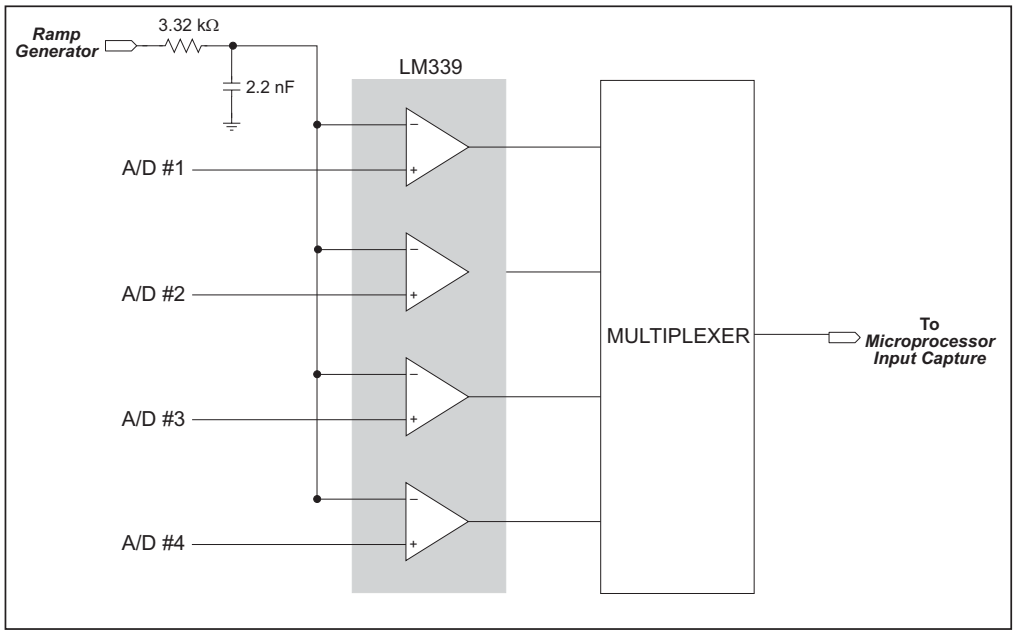


Figure 14. Using Multiplexer for A/D Converter Channels

In order to filter out noise picked up in the ramp signal that results from coupling of the PCB traces to nearby fast-switching digital signals, the ramp is filtered by an RC circuit at each comparator package by an RC circuit consisting of a 3.3 kΩ resistor and a 2.2 nF capacitor. This RC filter delays the ramp by about 7 µs, which is the same as 10 mV of ramp movement. The 3 db point of the filter is approximately 25 kHz. This is adequate to eliminate high-frequency spikes coupled to the PCB trace. The same filter is used for the start-of-ramp pulse and the reference pulse, so the delay of the ramp is canceled out. When the ramp switches direction from downward to upward, the ramp has sufficient negative swing to allow the transient to die down before the start-of-ramp pulse is generated.

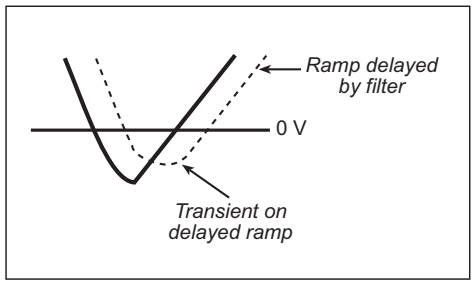


Figure 15. Ramp Signal Offset from RC Filter

4.5 Other Hardware

4.5.1 Clock Doubler

The PowerCore 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 51.6 MHz frequency specified for the PowerCore is generated using a 25.8 MHz crystal.

The clock doubler may be disabled if 51.6 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.

The clock doubler is automatically off for 25.8 MHz modules. Fast SRAM is needed to run programs when clock speeds are above 30 MHz.

4.5.2 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.

TIP: If you wish to add more than one macro definition to the Global Macro Definitions box, the macros should be separated by semi-colons.

4.6 Memory

4.6.1 SRAM

PowerCore FLEX modules running at 51.6 MHz need 512K of program-execution SRAM at U13, which is not battery-backed. The battery-backed data SRAM at U2 on all PowerCore modules is 256K–512K.

4.6.2 Flash EPROM

PowerCore modules also have 512K of flash EPROM installed at U3.

> **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 functions **`writeUserBlock`** and **`readUserBlock`** are provided for this. Refer to the ***Rabbit 3000 Microprocessor Designer's Handbook*** for additional information.

A Flash Memory Bank Select jumper configuration option based on 0 Ω surface-mounted resistors exists at JP2 on the PowerCore module. This option is reserved for future use.

4.6.3 Serial Flash

A serial flash is an available option on PowerCore FLEX modules to store data and Web pages. Sample programs in the **`SAMPLES\PowerCoreFLEX`** folder illustrate the use of the serial flash. These sample programs are described in Section 3.2.4, "Use of Serial Flash."

4.6.4 Dynamic C BIOS Source Files

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

4.7 Power Supply Options and Requirements

Appendix D provides a complete description of the onboard power supply, and the options and requirements for both onboard and external power supplies used with PowerCore modules.

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 devices based on the Rabbit microprocessor. Chapter 5 describes the libraries and function calls related to the PowerCore FLEX modules.

5.1 More About Dynamic C

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

You have a choice of doing your software development in the flash memory or in the static SRAM included on PowerCore FLEX modules. 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 directly to battery-backed RAM, but cannot be run reliably out of RAM after the programming cable is disconnected. Your final code must always be stored in flash memory for reliable operation. If your board has a fast SRAM, which is not battery-backed, you should select the "Code and BIOS in Flash, Run in RAM" compiler option, which stores the code in flash and copies it to the fast SRAM at run-time to take advantage of the faster clock speed. This option optimizes the performance of PowerCore FLEX modules running at 51.6 MHz.

NOTE: Do not depend on the flash memory sector size or type in your program logic. PowerCore modules 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 running Windows 95 or later. Programs can be downloaded at baud rates of up to 460,800 bps after the program compiles.

Dynamic C has a number of standard features:

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

5.1.1 Compile Options

Dynamic C offers three compile options:

- Compile to attached target (default, used when a PowerCore module is connected to a PC running Dynamic C via the programming cable)
- Compile defined target configuration to .bin file (see the additional information below)
- Compile to .bin file using attached target

Since the PowerCore FLEX modules permit many different configurations, a special project file for targetless compile should be set up with the configuration specific to your PowerCore FLEX board if you plan to compile a defined target configuration to a.bin file. Follow the instructions included with the **PowerCoreFLEX_BOARD_OPTIONS.c** sample program in the **SAMPLES\PowerCoreFLEX** folder to determine and set up the targetless compile options for your PowerCore module.

5.1.2 Using Dynamic C with Interrupts

You must disable all interrupts you have turned on before you exit Dynamic C to avoid getting target communication error messages. Use the **exit()** function call to do so as in some of the sample programs such as **ADC_RD_EXTERNAL.C**. The **exit(exitcode)** function call returns the **exitcode** parameter to Dynamic C and stops the program. 0 is used for this parameter to indicate that the function executed successfully; other values correspond to run-time errors.

5.1.3 User Block

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

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

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

5.2 Dynamic C Functions

5.2.1 Digital I/O

PowerCore FLEX modules were designed to interface with other systems, and so there are no drivers written specifically for the digital I/O; there are sample drivers for the triacs and the A/D converter on the Prototyping Board described in Section 5.2.7.

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.

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

5.2.2 External I/O

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.

5.2.3 SRAM Use

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

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

```
protected nf_device nandFlash;
int main() {
   ...
   _sysIsSoftReset();    // restore any protected variables
```

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

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

5.2.4 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***.

Section 5.2.8.5 provides sample drivers for serial communication when using the Prototyping Board.

The sample programs in the Dynamic C **SAMPLES/PowerCoreFLEX/RS232** folder provide further examples.

5.2.5 TCP/IP Drivers

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

5.2.6 Serial Flash Drivers

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

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

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

5.2.7 A/D Converter Ramp-Generator Drivers

The functions described in this section support the PowerCore ramp generator. The source code is in the Dynamic C **LIB\PowerCoreFLEX\ADCRAMP.LIB** library. This library provides the functions for the onboard A/D converter circuit to read the PowerCore thermistor, 2.5 V reference, and end-of-ramp voltages. The library also has a function call that can also be used to read an external A/D converter voltage.

The sample programs in the Dynamic C **SAMPLES/PowerCoreFLEX/ADC** and **SAMPLES/PowerCoreFLEX/DAC** folders provide further examples.

anaInRampInit

```
void anaInRampInit(void);
```

DESCRIPTION

Initializes the A/D converter ramp low-level driver for internal A/D operation to read the thermistor, 2.5 V reference, and end-of-ramp voltages. Call the **anaInExternalInit** function to enable reading an external A/D converter.

RETURN VALUE

None.

SEE ALSO

anaInRamp, anaInRampVolts, anaInCalibRamp, anaInEERdRamp, anaInEEWrRamp

anaInRamp

```
int anaInRamp(int channel);
```

DESCRIPTION

Reads the state of a ramp A/D channel. The state being read is placed in memory, and is updated by the A/D converter interrupt. The update interval is 7.5 ms for each A/D converter channel.

Use the following flags to read the ramp channels synchronously.

ref_conversion_done—2.5 V reference
ramp_conversion_done—end-of-ramp voltage
temp_conversion_done—thermistor

The **ADC_RD_RAMP.c** sample program provides an example.

PARAMETER

channel the analog input channel (0–2) to read.

0 = 2.5 V reference
1 = end-of-ramp voltage
2 = thermistor

RETURN VALUE

A value (0 to 4095) corresponding to the voltage on the analog input channel.

SEE ALSO

anaInRampInit, anaInCalibRamp, anaInRampVolts, anaInEERdRamp, anaInEEWrRamp

anaInRampVolts

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

DERSCRIPTION

Reads the state of a ramp A/D channel and uses the previously set calibration constants to convert it to volts. The state being read is placed in memory, and is updated by the A/D converter interrupt. The update interval is 7.5 ms for each A/D converter channel.

Use the following flags to read the ramp channels synchronously.

ref_conversion_done—2.5 V reference
ramp_conversion_done—end-of-ramp voltage
temp_conversion_done—thermistor

The **ADC_RD_RAMP.c** sample program provides an example.

PARAMETER

channel the analog input channel (0–2) to read.

0 = 2.5 V reference
1 = end-of-ramp voltage
2 = thermistor

RETURN VALUE

A value corresponding to the voltage on the analog input channel (0 to 3 V).

SEE ALSO

anaInRampInit, anaInCalibRamp, anaInRamp, anaInEERdRamp, anaInEEWrRamp

anaInCalibRamp

```
void anaInCalibRamp(int value1, float volts1, int value2,
   float volts2);
```

DESCRIPTION

Calibrates the response of the analog ramp circuit as a linear function using the two conversion points provided. The gain, offset and A/D converter resolution constants are calculated and placed into the global table **_adcCalibRamp**.

PARAMETERS

value1	the first A/D converter value.
volts1	the voltage corresponding to the first A/D converter value.
value2	the second A/D converter value.
volts2	the voltage corresponding to the second A/D converter value.

RETURN VALUE

0 if successful.
-1 if not able to make calibration constants.

SEE ALSO

anaInRampInit, anaInRamp, anaInRampVolts, anaInEERdRamp, anaInEEWrRamp

anaInEERdRamp

```
int anaInEERdRamp(void);
```

DESCRIPTION

Reads the gain and offset calibration constants for the ramp A/D converter from the simulated EEPROM in flash memory located at the top 2K of the reserved user block memory area.

RETURN VALUE

0 if successful.
-1 if address is invalid or is out of range.

SEE ALSO

anaInEEWrRamp

anaInEEWrRamp

```
int anaInEEWrRamp(void);
```

DESCRIPTION

Writes the gain and offset calibration constants for the ramp A/D converter to the simulated EEPROM in flash memory located at the top 2K of the reserved user block memory area.

RETURN VALUE

0 if successful.
-1 if address is invalid or is out of range.

SEE ALSO

anaInEERdRamp

thermReading

```
float thermReading(int units);
```

DESCRIPTION

Reads the voltage across the PowerCore thermistor via the ramp A/D converter. The value read from the A/D converter is converted to temperature in the units specified by the **units** parameter.

PARAMETER

units the temperature units.

0 = Celsius
1 = Fahrenheit
2 = Kelvin

RETURN VALUE

Temperature calculated from the thermistor located on the PowerCore. The value is in the units specified by the **units** parameter.

SEE ALSO

anaInRampInit, thermOffset

thermOffset

```
void thermOffset(int units, float reading);
```

DESCRIPTION

Creates an offset for any temperature difference between the thermistor and the external reference temperature, and writes the offset data to the simulated EEPROM in flash memory located at the top 2K of the reserved user block memory area. This offset value is then used by the **thermReading** function to calculate the temperature at the thermistor within the resolution specified for the thermistor's A/D converter circuit.

PARAMETERS

units the temperature units.

0 = Celsius
1 = Fahrenheit
2 = Kelvin

reading the value of the external temperature reference in the units specified by the **units** parameter.

RETURN VALUE

None.

SEE ALSO

anaInRampInit, thermReading

anaInDisable

```
void anaInDisable(void);
```

DESCRIPTION

Disables A/D ramp interrupt when writing to flash memory and to exit applications while using Dynamic C for a user interface.

RETURN VALUE

None.

SEE ALSO

anaInRamp, anaInRampVolts, anaInCalibRamp, anaInEERdRamp, anaInEEWrRamp, anaInEnable

anaInEnable

```
void anaInEnable(void);
```

DESCRIPTION

Re-enables A/D ramp interrupt after the interrupt was disabled by the **anaInDisable** function call.

RETURN VALUE

None.

SEE ALSO

anaInRamp, anaInRampVolts, anaInCalibRamp, anaInEERdRamp, anaInEEWrRamp, anaInDisable

anaInExternalInit

```
void anaInExternalInit(int port, int pin, int edge);
```

DESCRIPTION

Initializes the A/D converter low-level driver to read an external A/D converter. You must call the **anaInRampInit** function to initialize the A/D converter ramp circuit before you call **anaInExternalInit**.

PARAMETERS

port identifies the Rabbit I/O port used for the external A/D comparator.

- 0 = Parallel Port C
- 1 = Parallel Port D
- 2 = Parallel Port F
- 3 = Parallel Port G

pin identifies the specific Rabbit parallel port pin used for the external A/D comparator.

- 0 = pin 1
- 1 = pin 3
- 2 = pin 5
- 3 = pin 7

edge indicates which signal edge to use for the external A/D comparator.

- 0 = rising edge
- 1 = falling edge

RETURN VALUE

None.

anaIn

```
int anaIn(int channel);
```

DESCRIPTION

Reads the state of an external analog channel. To read the ramp channels synchronously, use the **ref_conversion_done** flag. The **adc_rd_external** function provides an example.

The state being read is located in memory, and is updated by the A/D converter interrupt. The update interval is as follows:

- For single-channel designs, the state of the analog channel will be updated approximately every 2.5 ms.
- For multiple-channel designs, the update interval will depend on the number of channels you have and the way you have implemented your MUX control (round-robin vs. random channel selection):

  the round-robin update interval will be the number of channels times 2.5 ms.

  the random-channel-selection update interval will be >2.5 ms for the initial channel selection and 2.5 ms thereafter.

There is a macro/hook in the interrupt service routine for you add your A/D converter MUX control code. The following steps explain how to implement round-robin MUX control.

1. Set the maximum number of external A/D converter channels:

   ```
   #define MAX_ADCHANNELS <new value>
   ```

2. Create a pointer to your custom MUX control routine by defining

   ```
   #define ADC_MUX_CNTRL call adcmux
   ```

   This step will insert a call to the **adcmux** routine at the end of interrupt service routine. This will allow the channel to be ready for the next conversion cycle.

3. Write your **adcmux** routine to switch the hardware and **adc_mux_channel** index to the next A/D converter channel. Make sure you initialize your I/O before executing the **anaInExternalInit** function.

The following steps explain how to implement random-channel-selection MUX control.

1. Set the maximum number of external A/D converter channels:

   ```
   #define MAX_ADCHANNELS <new value>
   ```

2. Provide an application function that will do the following:
   - Select the A/D converter channel via the hardware MUX circuit. Be sure to initialize your I/O before executing the **anaInExternalInit** and the new MUX function.
   - Change the **_adc_mux_channel** index to the channel selected.
   - Set the **adc_conversion_done** flag to FALSE.
   - Then call your routine with the A/D converter channel selected.
   - Wait in a nonblocking wait routine for the **adc_conversion_done** flag to become TRUE.
   - Read the A/D converter.
   - Repeat the sequence by calling your routine with the A/D converter channel selected.

PARAMETER

channel — the analog input channel (0 to **MAX_ADCHANNELS** - 1) to read. The **MAX_ADCHANNELS** macro is set to a default value of 1, which can be changed by adding the following line to your program.

```
#define MAX_ADCHANNELS <new value>
```

RETURN VALUE

A value (0–4095) corresponding to the voltage on the analog input channel. A value of -4096 indicates an overflow or an out-of-range condition.

SEE ALSO

anaInExternalInit, anaInVolts, anaInCalib

anaInVolts

```
float anaInVolts(int channel);
```

DESCRIPTION

Reads the state of an external analog channel and uses the previously set calibration constants to convert the value to volts.

The state being read is located in memory, and is updated by the A/D converter interrupt. The update interval is as follows:

- For single-channel designs, the state of the analog channel will be updated approximately every 2.5 ms.
- For multiple-channel designs, the update interval will depend on the number of channels you have and the way you have implemented your MUX control (round-robin vs. random channel selection):

  the round-robin update interval will be the number of channels times 2.5 ms.

  the random-channel-selection update interval will be >2.5 ms for the initial channel selection and 2.5 ms thereafter.

There is a macro/hook in the interrupt service routine for you add your A/D converter MUX control code. The following steps explain how to implement round-robin MUX control.

1. Set the maximum number of external A/D converter channels:

   `#define MAX_ADCHANNELS <new value>`

2. Create a pointer to your custom MUX control routine by defining

   `#define ADC_MUX_CNTRL call adcmux`

   This step will insert a call to the **`adcmux`** routine at the end of interrupt service routine. This will allow the channel to be ready for the next conversion cycle.

3. Write your **`adcmux`** routine to switch the hardware and **`adc_mux_channel`** index to the next A/D converter channel. Make sure you initialize your I/O before executing the **`anaInExternalInit`** function.

The following steps explain how to implement random-channel-selection MUX control.

1. Set the maximum number of external A/D converter channels:

   #define MAX_ADCHANNELS <new value>

2. Provide an application function that will do the following:
   - Select the A/D converter channel via the hardware MUX circuit. Be sure to initialize your I/O before executing the **anaInExternalInit** and the new MUX function.
   - Change the **_adc_mux_channel** index to the channel selected.
   - Set the **adc_conversion_done** flag to FALSE.
   - Then call your routine with the A/D converter channel selected.
   - Wait in a nonblocking wait routine for the **adc_conversion_done** flag to become TRUE.
   - Read the A/D converter.
   - Repeat the sequence by calling your routine with the A/D converter channel selected.

PARAMETER

channel — the analog input channel (0 to **MAX_ADCHANNELS** - 1) to read. The **MAX_ADCHANNELS** macro is set to a default value of 1, which can be changed by adding the following line to your program.

#define MAX_ADCHANNELS <new value>

RETURN VALUE

A voltage value corresponding to the voltage on the analog input channel. A value of -4096 indicates an overflow or an out-of-range condition.

SEE ALSO

anaInExternalInit, anaIn, anaInCalib

anaInCalib

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

DESCRIPTION

Calibrates the response of the selected external A/D converter channel as a linear function using the two conversion points provided. The gain and offset constants are calculated and placed into the global table **_adcInCalib**.

PARAMETERS

channel	the analog input channel (0 to **MAX_ADCHANNELS** - 1) to read. The **MAX_ADCHANNELS** macro is set to a default value of 1, which can be changed by adding the following line to your program. **#define MAX_ADCHANNELS <new value>**
value1	the first A/D converter value.
volts1	the voltage corresponding to the first A/D converter value.
value2	the second A/D converter value.
volts2	the voltage corresponding to the second A/D converter value.

RETURN VALUE

0 if successful.
-1 if not able to make calibration constants.

SEE ALSO

anaInExternalInit, anaInVolts, anaIn

anaInEERd

```
int anaInEERd(int channel);
```

DESCRIPTION

Reads the gain and offset calibration constants for an external A/D converter from the simulated EEPROM in flash memory located at the top 2K of the reserved user block memory area.

PARAMETER

channel the analog input channel (0 to **MAX_ADCHANNELS** - 1) to read. The **MAX_ADCHANNELS** macro is set to a default value of 1, which can be changed by adding the following line to your program.

```
#define MAX_ADCHANNELS <new value>
```

RETURN VALUE

0 if successful.
-1 if address is invalid or is out of range.

SEE ALSO

anaInExternalInit, anaInCalib, anaInEEWr, anaIn, anaInVolts

anaInEEWr

```
int anaInEEWr(int channel);
```

DESCRIPTION

Writes the gain and offset calibration constants for an external A/D converter to the simulated EEPROM in flash memory located at the top 2K of the reserved user block memory area.

PARAMETER

channel the analog input channel (0 to **MAX_ADCHANNELS** - 1) to read. The **MAX_ADCHANNELS** macro is set to a default value of 1, which can be changed by adding the following line to your program.

```
#define MAX_ADCHANNELS <new value>
```

RETURN VALUE

0 if successful.
-1 if address is invalid or is out of range.

SEE ALSO

anaInExternalInit, anaInCalib, anaInEERd, anaIn, anaInVolts

5.2.8 Prototyping Board Functions

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

The **PowerCoreFLEX.LIB** library automatically uses the **RN_CFG_PowerCoreFLEX.LIB** library, which is used to configure the PowerCore for use with RabbitNet peripheral boards on the PowerCore Prototyping Board.

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

5.2.8.1 Board Initialization

---

brdInit

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

DESCRIPTION

Call this function at the beginning of your program. This function initializes Parallel Ports A through G for use with the PowerCore Prototyping Board.

The Ethernet signals are configured according to the specific PowerCore FLEX module—PD0, PD2, and PE2 are initialized by the Ethernet driver for PowerCore FLEX modules with Ethernet, and are configured as outputs for PowerCore FLEX modules without Ethernet.

RETURN VALUE

None.

digIn

```
int digIn(int channel);
```

DESCRIPTION

Reads the input state of a digital input on the PowerCore Prototyping Board.

PARAMETER

channel the channel number corresponding to the digital input channel:

0—IN0 (pin 16 on header J4 or switch SW2 on the Prototyping Board)

1—IN1 (pin 47 on header J4 or switch SW3 on the Prototyping Board)

RETURN VALUE

The logic state (0 or 1) of the input. A run-time error will occur if the **channel** parameter is out of range.

SEE ALSO

brdInit, digOut

digOut

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

DESCRIPTION

Sets the state of digital outputs OUT00–OUT03 on Prototyping Board header J2.

PARAMETERS

channel the digital output channel OUT00–OUT03:

0 = OUT00 (sinking type output)
1 = OUT01 (sinking type output)
2 = OUT02 (sourcing type output)
3 = OUT03 (sourcing type output)

state the output logic value (0 or 1) to output:

Sinking driver
0 = connects the load to GND
1 = puts the output in a high-impedance state.

Sourcing driver
0 = puts the output in a high-impedance state
1 = connects the load to +K.

RETURN VALUE

None.

SEE ALSO

`brdInit, digIn`

ledOut

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

DESCRIPTION

Controls LEDs DS3 and DS4 on the PowerCore Prototyping Board.

PARAMETERS

led the LED to control:

0 = DS4
1 = DS3

value the value used to control the LED:

0 = off
1 = on

RETURN VALUE

None.

SEE ALSO

brdInit

anaOut

```
void anaOut(int channel, int rawdata);
```

DESCRIPTION

Sets the voltage on a given analog output channel on the Prototyping Board.

PARAMETERS

channel	the analog output channel (0–2).
rawdata	a value corresponding to the voltage on the analog output. Valid range = 0–1024.

RETURN VALUE

None.

SEE ALSO

`anaOutVolts, anaOutCalib, brdInit`

anaOutVolts

```
void anaOutVolts(int channel, float voltage);
```

DESCRIPTION

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

PARAMETERS

channel	the analog output channel (0–2).
voltage	the voltage desired on the analog output.

RETURN VALUE

None.

SEE ALSO

`anaOut, anaOutCalib, brdInit`

anaOutCalib

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

DESCRIPTION

Calibrates the response of the specified D/A converter channel on the Prototyping Board as a linear function using the two calibration points provided. Gain and offset constants are calculated and placed into the global table **_dacCalib**.

PARAMETERS

channel	the analog output channel to be calibrated (0–2).
value1	the first D/A converter value (0–1024).
volts1	the voltage (volts) corresponding to the first D/A converter value.
value2	the second D/A converter value (0–1024).
volts2	the voltage (volts) corresponding to the second D/A converter value.

RETURN VALUE

0 if successful.
-1 if not able to make calibration constants.

SEE ALSO

anaOut, anaOutVolts, brdInit

anaOutEERd

```
int anaOutEERd(int channel);
```

DESCRIPTION

Reads the gain and offset calibration constants from the simulated EEPROM in flash memory located at the top 2K of the reserved user block memory area.

NOTE: This function cannot be run in RAM.

PARAMETERS

channel THE ANALOG OUTPUT CHANNEL (0–2) WHOSE CALIBRATION CONSTANTS ARE TO BE READ.

RETURN VALUE

0 if successful.
-1 if address is invalid or is out of range.

SEE ALSO

anaOutEEWr, brdInit

anaOutEEWr

```
int anaOutEEWr(int channel);
```

DESCRIPTION

Writes the gain and offset calibration constants from the simulated EEPROM in flash memory located at the top 2K of the reserved user block memory area.

PARAMETER

channel	the analog output channel (0–2) whose calibration constants are to be written.

RETURN VALUE

0 if successful.
-1 if address is invalid or is out of range.

SEE ALSO

anaOutEERd, brdInit

serMode

```
void serMode(int mode);
```

DESCRIPTION

Sets up serial communication lines for the PowerCore FLEX modules.

Remember to call the **serXopen()** function before running this function before you start using any of the serial ports.

PARAMETER

mode the defined serial port configuration.

Mode	Serial Port	
	E	F
0	RS-232, 3-wire	RS-232, 3-wire
1	RS-232, 5-wire	RTS/CTS

RETURN VALUE

0 if valid mode; 1 if not.

SEE ALSO

serX functions located in **SERIAL.LIB**.

5.2.8.6 RabbitNet Port

The function calls described in this section are used to configure the RabbitNet port on the PowerCore Prototyping Board for use with RabbitNet peripheral cards. The user's manual for the specific peripheral card you are using contains additional function calls related to the RabbitNet protocol and the individual peripheral card. Appendix E provides additional information about the RabbitNet.

These RabbitNet peripheral cards are available at the present time.

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

Before using the RabbitNet port, add the following lines at the start of your program.

```
#define RN_MAX_DEV 10  // max number of devices
#define RN_MAX_DATA 16 // max number of data bytes in any transaction
#define RN_MAX_PORT 1  // max number of serial ports
```

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

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

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

rn_sp_info

```
void rn_sp_info();
```

DESCRIPTION

Provides **rn_init()** with the serial port control information needed for PowerCore FLEX modules.

RETURN VALUE

None.

rn_sp_close

```
void rn_sp_close(int port);
```

DESCRIPTION

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

PARAMETER

portnum = 0

RETURN VALUE

None

rn_sp_enable

```
void rn_sp_enable(int portnum);
```

DESCRIPTION

This is a macro that enables or asserts the RabbitNet port chip select on the PowerCore Prototyping Board prior to data transfer.

PARAMETER

portnum = 0

RETURN VALUE

None

rn_sp_disable

```
void rn_sp_disable(int portnum);
```

DESCRIPTION

This is a macro that disables or deasserts the RabbitNet port chip select on the Power-Core Prototyping Board to invalidate data transfer.

PARAMETER

portnum = 0

RETURN VALUE

None.

5.2.8.7 Triac Control

The functions described in this section support the triacs on the PowerCore Prototyping Board. The library can also be used in conjunction with other triac installations that incorporate a zero-crossing crossover detection circuit. The source code is in the Dynamic C **LIB\PowerCoreFLEX\TRIAC.LIB** library.

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

triac_PhaseInit

```
void triac_PhaseInit(int ext_interrupt, int interrupt_pin);
```

DESCRIPTION

Initializes the triac phase-angle control interrupt. Phase-angle triac control provides you with the ability to fire a triac at a given phase angle of a positive and negative 50/60 Hz A/C sine wave, thus providing you with the control required by your application.

A run-time error will occur if the **triac_PhaseInit** function has not executed.

To initialize the triac driver completely, you must also run the **triac_PhaseCntrlPin** function for each triac to be used in your application. You will need to initialize the port pin(s) you select for triac control before calling any of the triac API functions when using the **triac_PhaseCntrlPin** function. For multiple triacs, the control pins must be on the same port—this limits the number of triacs to 8, the maximum number of pins on a port. You will also need to write a custom I/O driver for the control pins. (See the **triac_gate_on** and **triac_gate_off** routines in the **TRIAC_PHASE.c** sample program for an example.) The triac driver will calibrate automatically to the incoming 50/60 Hz A/C waveform being used. Note that high-frequency calibration occurs 100% of the time, whereas low-frequency calibration occurs every n'th time specified by the **TRIAC_LOW_FREQ_CAL** macro.

Add the following lines in your application to set up the driver properly.

```
// Define phase-angle triac control method for proper
// library compilation.
#define PHASECONTROL
// Set the max number of triacs (max. is 8) for your application.
#define MAX_TRIACS <number of triacs>
// Define the triac control function names.
#define TRIAC_GATE_ON triac_gate_on
#define TRIAC_GATE_OFF triac_gate_off
// #use the triac lib
#use "triac.lib"
```

Note that the interrupt priority level is preset to level 3.

PARAMETERS

ext_interrupt selects the external interrupt vector.

0 = external interrupt 0
1 = external interrupt 1

interrupt_pin selects the external interrupt I/O pin.

0 = I/O pin PE0, only valid for external interrupt 0
1 = I/O pin PE4, only valid for external interrupt 0
2 = I/O pin PE1, only valid for external interrupt 1
3 = I/O pin PE5, only valid for external interrupt 1

RETURN VALUE

None.

SEE ALSO

triac_PhaseCntrlPin, triac_PhaseCntrl, triac_PhaseLock, triac_PhaseUnlock, triac_PhaseEnable, triac_PhaseDisable

triac_PhaseInitPWM

```
int triac_PhaseInitPWM(int channel, int pwm_level, int duty_
   cycle, int options, unsigned long frequency);
```

DESCRIPTION

Initializes a PWM channel for triac gate-signal power reduction.

PARAMETERS

channel the PWM channel (0–3) to use for triac gate-power reduction.

pwm_level the PWM static output state for the triac gate signal. When triac gate-power reduction is disabled,

0 = sets PWM output low
1 = sets PWM output high.

duty_cycle selects the duty cycle (0–1024) to be used for triac gate-power reduction.

options sets the PWM control options. Use the following macro bit masks to enable the selected option:

PWM_SPREAD sets pulse spreading. The duty cycle is spread over four separate pulses, and will increase the frequency by a factor of 4.

PWM_OPENDRAIN sets the PWM output pin to be an open-drain output instead of a normal push-pull logic output.

0 sets the PWM for a normal push-pull logic output with no pulse spreading.

frequency the PWM base frequency (in Hz). The base frequency is the frequency without pulse spreading. Pulse spreading (see **options** parameter) will increase the base frequency by a factor of 4.

RETURN VALUE

0 = OK.
-1 = an invalid channel number is used.
-2 = an invalid duty cycle was requested.
-3 = frequency requested is out of range or invalid.

triac_PhaseCntrlPin

```
void triac_PhaseCntrlPin(int triac, int port, int bit, int pin_
   state);
```

DESCRIPTION

Initializes the port and the I/O pin that is going to be used by the specified triac.

For multiple triacs, the control pins must be on the same port—the maximum number of pins is 8. You will also need to write a custom I/O driver for the control pins. (See the **triac_gate_on** and **triac_gate_off** routines in the **TRIAC_PHASE.c** sample program for an example.)

A run-time error will occur if the **triac_PhaseInit** function has not executed or if the maximum number of 8 triacs is exceeded.

PARAMETERS

triac	selects the triac (0 to **MAX_TRIACS** - 1).
port	specifies the I/O port used to control the triac. Use one of these predefined I/O macros.
	PADR, **PBDR**, **PCDR**, **PDDR**, **PEDR**, **PFDR**, or **PGDR**
io_pin	the bit number of the I/O pin to be used for triac control.
pin_state	the value used to set the I/O pin to its initial state.

RETURN VALUE

None.

SEE ALSO

triac_PhaseInit, triac_PhaseCntrl, triac_PhaseLock, triac_PhaseUnlock, triac_PhaseEnable, triac_PhaseDisable

triac_PhaseLock

```
void triac_PhaseLock(void);
```

DESCRIPTION

Locks the triac update buffer for synchronous operation of multiple triacs. The sequence is as follows.

1. Lock buffer via **triac_PhaseLock**—a low-level driver will use the last triac state until the buffer is unlocked via a call to **triac_PhaseUnlock**.
2. Update triacs via a call to **triac_PhaseCntrl**.
3. Unlock the buffer via **triac_PhaseUnlock**. The new triac setting will take effect at this time.

RETURN VALUE

None.

SEE ALSO

triac_PhaseInit, triac_PhaseCntrlPin, triac_PhaseCntrl, triac_PhaseUnlock, triac_PhaseEnable, triac_PhaseDisable

triac_PhaseUnlock

```
void triac_PhaseUnlock(void);
```

DESCRIPTION

Unlocks the triac update buffer for synchronous operation of multiple triacs. The sequence is as follows.

1. Lock buffer via **triac_PhaseLock**—a low-level driver will use the last triac state until the buffer is unlocked via a call to **triac_PhaseUnlock**.
2. Update triacs with a call to **triac_PhaseCntrl**.
3. Unlock the buffer via **triac_PhaseUnlock**. The new triac setting will take effect at this time.

RETURN VALUE

None.

SEE ALSO

triac_PhaseInit, triac_PhaseCntrlPin, triac_PhaseCntrl, triac_PhaseLock, triac_PhaseEnable, triac_PhaseDisable

triac_PhaseDisable

```
int triac_PhaseDisable(void);
```

DESCRIPTION

Disables the triac control interrupt to allow your application to do flash-write operations. Call the **triac_PhaseInit** function before calling this function.

Remember to call **triac_PhaseDisable** before doing any flash-write operations. Once you have completed the flash-write operations, you must call **triac_PhaseEnable** to restart the triac driver. This requirement also applies to any other operation that disables all interrupts

RETURN VALUE

0 = triac driver is in the process of being disabled.
1 = triac driver is disabled.

SEE ALSO

triac_PhaseInit, triac_PhaseCntrlPin, triac_PhaseCntrl, triac_PhaseLock, triac_PhaseUnLock, triac_PhaseEnable

triac_PhaseEnable

```
int triac_PhaseEnable(void);
```

DESCRIPTION

Re-enables the triac control interrupt after a flash-write operation has been completed. Call the **triac_PhaseInit** and the **triac_PhaseDisable** functions before calling this function.

RETURN VALUE

0 = triac driver is not ready.
1 = triac driver is ready.

SEE ALSO

triac_PhaseInit, triac_PhaseCntrlPin, triac_PhaseCntrl, triac_PhaseLock, triac_PhaseUnLock, triac_PhaseDisable

triac_PhaseCntrl

```
void triac_PhaseCntrl(int triac, int onOff, int phaseAngle,
   int pwm_cntrl);
```

DESCRIPTION

Sets the sine-wave phase angle at which to fire the specified triac. Phase-angle triac control provides you with the ability to fire a triac at a given phase angle of a positive and negative 50/60 Hz A/C sine wave, thus providing you with the control required by your application.

A run-time error will occur if the **triac_PhaseInit** or the **triac_PhaseCntrlPin** functions have not executed.

PARAMETERS

triac selects the triac (0 to **MAX_TRIACS** - 1).

onOff enables/disables the triac.

0 = disable triac
1 = enable triac.

phaseAngle sets the sine-wave phase angle of when to fire the triac (valid range = 0–179°).

pwm_ctrl sets the triac gate-signal PWM power-reduction option.

0 = disable PWM for gate signal
1 = enable PWM for gate signal

RETURN VALUE

None.

SEE ALSO

triac_PhaseInit, triac_PhaseCntrlPin, triac_PhaseLock, triac_PhaseUnLock, triac_PhaseEnable, triac_PhaseDisable

triac_TimePropInit

```
void triac_TimePropInit(int ext_interrupt, int interrupt_pin);
```

DESCRIPTION

Initializes the triac time-proportional control interrupt. Time-proportional triac control provides control of a triac for a fixed period of time, with the application setting the ON and OFF times within this fixed time period to provide the desired ratio of ON/OFF times.

To initialize the triac driver completely, you must also run the **triac_TimePropCntrlPin** function for each triac to be used in your application. You will need to initialize the port pin(s) you select for triac control before calling any of the triac API functions when using the **triac_TimePropCntrlPin** function. For multiple triacs, the control pins can be on any combination of ports. You will also need to write a custom I/O driver for the control pins. (See the **triac_gate_on** and **triac_gate_off** routines in the **TRIAC_RATIO.c** sample program for an example.)

Add the following lines in your application to set up the driver properly.

```
// Define time-proportional triac control method for
// proper library compilation.

#define TIMEPROPORTIONAL

// Set the max number of triacs for control

#define MAX_TRIACS <number of triacs>

// Define the triac custom function ON/OFF function
// names for proper library compilation. (see sample
program triac_ratio.c for an example)

#define TRIAC_GATE_ON triac_gate_on

#define TRIAC_GATE_OFF triac_gate_off

// Use the common ISR library for the triac and the
// ADC ramp circuit.

#use "adctriac_isr.lib"

// Use the triac library

#use "triac.lib"
```

Note that the interrupt priority level is preset to level 1.

PARAMETERS

ext_interrupt selects the external interrupt vector.

0 = external interrupt 0
1 = external interrupt 1

interrupt_pin selects the external interrupt I/O pin.

0 = I/O pin PE0, only valid for external interrupt 0
1 = I/O pin PE4, only valid for external interrupt 0
2 = I/O pin PE1, only valid for external interrupt 1
3 = I/O pin PE5, only valid for external interrupt 1

RETURN VALUE

None.

SEE ALSO

triac_TimePropCntrl

triac_TimePropCntrlPin

```
void triac_TimePropCntrlPin(int triac, int port, int io_pin,
   int pin_state);
```

DESCRIPTION

Initializes the port and the I/O pin that is going to be used by the specified triac.

For multiple triacs, the control pins can be on any combination of ports. You will also need to write a custom I/O driver for the control pins. (See the **triac_gate_on** and **triac_gate_off** routines in the **TRIAC_RATIO.c** sample program for an example.)

A run-time error will occur if the **triac_TimePropInit** function has not executed or if the maximum triac limit is exceeded.

PARAMETERS

triac	selects the triac (0 to **MAX_TRIACS** - 1).
port	specifies the I/O port used to control the triac. Use one of these predefined I/O macros.
	PADR, **PBDR**, **PCDR**, **PDDR**, **PEDR**, **PFDR**, or **PGDR**
io_pin	the bit number of the I/O pin to be used for triac control.
pin_state	the value used to set the I/O pin to its initial state.

RETURN VALUE

0 if successful.
-1 if address is invalid or is out of range.

SEE ALSO

triac_TimePropCntrl

triac_TimePropDisable

```
int triac_TimePropDisable(void);
```

DESCRIPTION

Disables the triac control interrupt to allow your application to do flash-write operations. Call the **triac_TimePropInit** function before calling this function.

Remember to call **triac_TimePropDisable** before doing any flash-write operations. Once you have completed the flash-write operations, you must call **triac_TimePropEnable** to restart the triac driver. This requirement also applies to any other operation that disables all interrupts.

RETURN VALUE

0 = triac driver is in the process of being disabled.
1 = triac driver is disabled.

triac_TimePropEnable

```
int triac_TimePropEnable(void);
```

DESCRIPTION

Re-enables the triac control interrupt after a flash-write operation has been completed. Call the **triac_TimePropInit** and the **triac_PhaseDisable** functions before calling this function.

RETURN VALUE

0 = triac driver is not ready.
1 = triac driver is ready.

triac_TimePropCntrl

```
void triac_TimePropCntrl(int triac, int onCycles,
   int totalCycles);
```

DESCRIPTION

Sets the on time of the specified triac initialized for time-proportional control. The specified triac will be turned on at the zero-crossing point of the 50/60 Hz A/C power cycles, and will remain on for the number of cycles you have specified.

The off-time is calculated by finding the difference between the total cycles and the on-time provided in this function.

A run-time error will occur if the **triac_TimePropInit** function has not executed.

PARAMETERS

triac	selects the triac (0 to **MAX_TRIACS** - 1).
onCycles	the number of 50/60 Hz cycles of on-time for the selected triac. Valid range = 0–32767.
totalCycles	sets the total number of 50/60 Hz cycles needed for control. Valid range = 0–32767.

RETURN VALUE

None.

SEE ALSO

triac_TimePropInit

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

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

In addition to the Web-based technical support included at no extra charge, a one-year telephone-based technical support module is also available for purchase.

6. USING THE TCP/IP FEATURES

6.1 TCP/IP Connections

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

Before proceeding you will need to have the following items.

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

The Ethernet cables and a 10Base-T Ethernet hub are available from Rabbit Semiconductor in a TCP/IP tool kit. More information is available at www.rabbit.com.

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

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

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

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

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

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

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

3. Apply Power

   Plug in the AC transformer. The PowerCore module and its Prototyping Board are now ready to be used.

6.2 TCP/IP Primer on IP Addresses

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

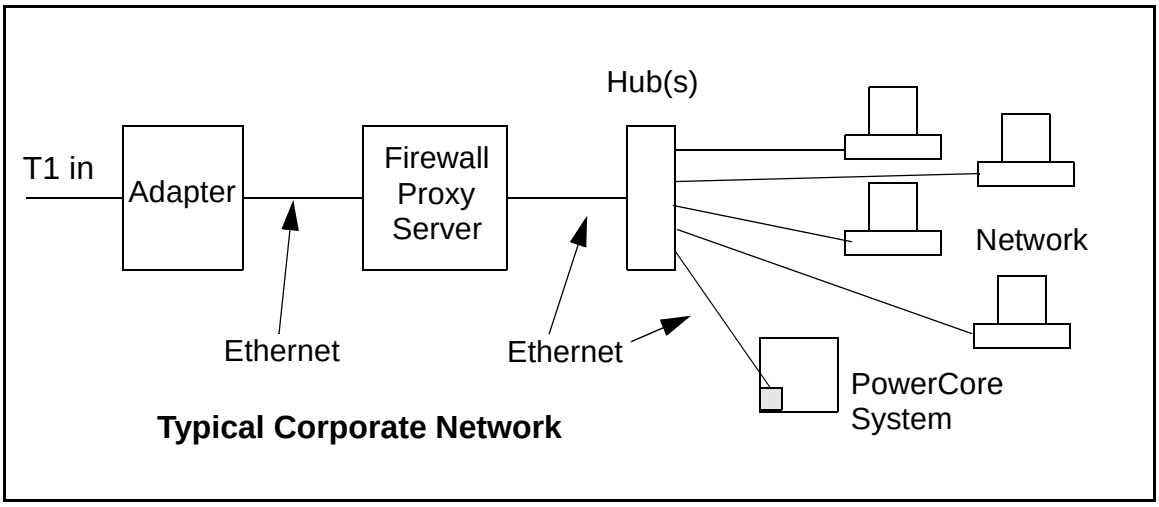
In order to set up this direct connection, the user will have to use a PC without networking, or disconnect a PC from the corporate network, or install a second Ethernet adapter and set up a separate private network attached to the second Ethernet adapter. Disconnecting your PC from the corporate network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

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

The PowerCore module uses a 10/100-compatible 10Base-T Ethernet interface, which is the most common scheme. The RJ-45 connectors are similar to U.S. style telephone connectors, except they are larger and have 8 contacts.

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

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



Typical Corporate Network

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

6.2.1 IP Addresses Explained

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

216.103.126.155

10.1.1.6

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

255.255.255.0

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

216.103.126.0

216.103.126.1

216.103.126.2

etc.

216.103.126.254

216.103.126.255

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

6.2.2 How IP Addresses are Used

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

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

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

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

6.2.3 Dynamically Assigned Internet Addresses

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

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

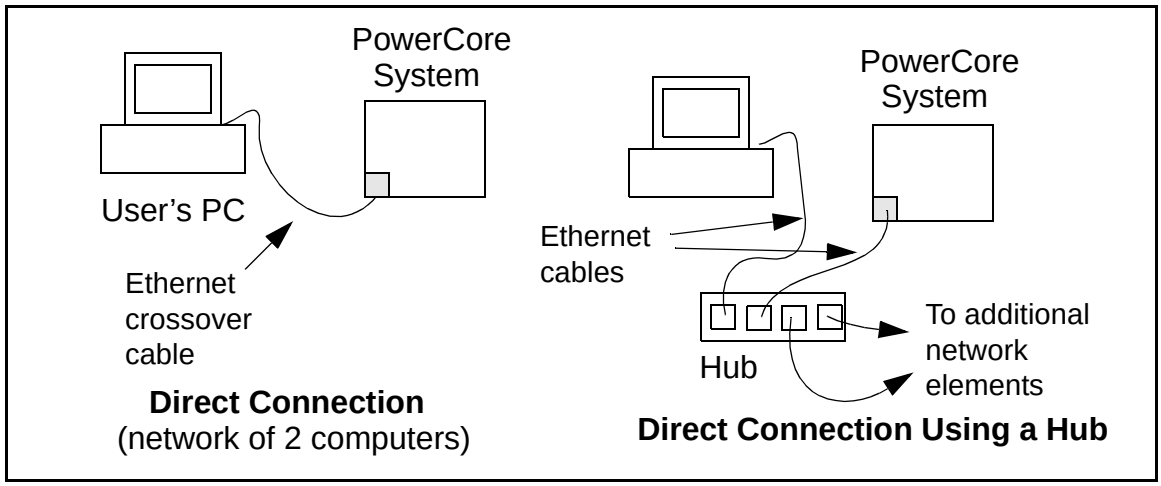
If you are not concerned about accessing the PowerCore module from the Internet, you can place the PowerCore module on the internal network using an IP address assigned either statically or through DHCP.

6.3 Placing Your Device on the Network

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

6.4 Running TCP/IP Sample Programs

We have provided a number of sample programs demonstrating various uses of TCP/IP for networking embedded systems. These programs require you to connect your PC and the PowerCore module together on the same network. This network can be a local private network (preferred for initial experimentation and debugging), or a connection via the Internet.



6.4.1 How to Set IP Addresses in the Sample Programs

With the introduction of Dynamic C 7.30 we have taken steps to make it easier to run many of our sample programs. You will see a **TCPCONFIG** macro. This macro tells Dynamic C to select your configuration from a list of default configurations. You will have three choices when you encounter a sample program with the **TCPCONFIG** macro.

1. You can replace the **TCPCONFIG** macro with individual **MY_IP_ADDRESS**, **MY_NETMASK**, **MY_GATEWAY**, and **MY_NAMESERVER** macros in each program.
2. You can leave **TCPCONFIG** at the usual default of 1, which will set the IP configurations to **10.10.6.100**, the netmask to **255.255.255.0**, and the nameserver and gateway to **10.10.6.1**. If you would like to change the default values, for example, to use an IP address of **10.1.1.2** for the PowerCore module, and **10.1.1.1** for your PC, you can edit the values in the section that directly follows the "General Configuration" comment in the **TCP_CONFIG.LIB** library. You will find this library in the **LIB\TCPIP** directory.
3. You can create a **CUSTOM_CONFIG.LIB** library and use a **TCPCONFIG** value greater than 100. Instructions for doing this are at the beginning of the **TCP_CONFIG.LIB** library in the **LIB\TCPIP** directory.

There are some other "standard" configurations for **TCPCONFIG** that let you select different features such as DHCP. Their values are documented at the top of the **TCP_CONFIG.LIB** library in the **LIB\TCPIP** directory. More information is available in the ***Dynamic C TCP/IP User's Manual***.

6.4.2 How to Set Up Your Computer for Direct Connect

Follow these instructions to set up your PC or notebook. Check with your administrator if you are unable to change the settings as described here since you may need administrator privileges. The instructions are specifically for Windows 2000, but the interface is similar for other versions of Windows.

> **TIP:** If you are using a PC that is already on a network, you will disconnect the PC from that network to run these sample programs. Write down the existing settings before changing them to facilitate restoring them when you are finished with the sample programs and reconnect your PC to the network.

1. Go to the control panel (**Start > Settings > Control Panel**), and then double-click the **Network** icon.
2. Select the network interface card used for the Ethernet interface you intend to use (e.g., **TCP/IP Xircom Credit Card Network Adapter**) and click on the "Properties" button. Depending on which version of Windows your PC is running, you may have to select the "Local Area Connection" first, and then click on the "Properties" button to bring up the Ethernet interface dialog. Then "Configure" your interface card for a "10Base-T Half-Duplex" or an "Auto-Negotiation" connection on the "Advanced" tab.

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

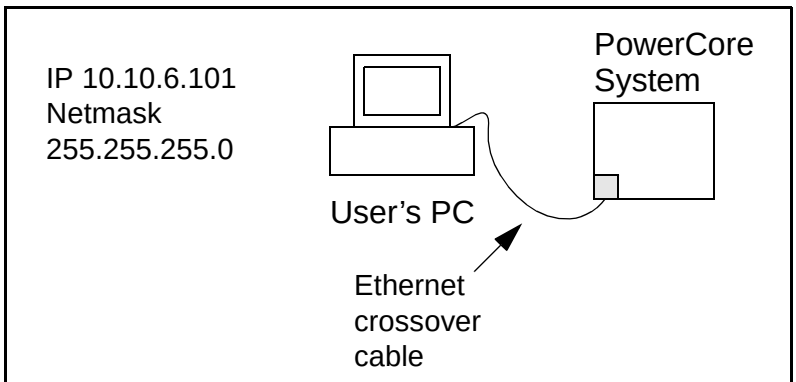
3. Now select the **IP Address** tab, and check **Specify an IP Address**, or select TCP/IP and click on "Properties" to assign an IP address to your computer (this will disable "obtain an IP address automatically"):

   IP Address : 10.10.6.101

   Netmask : 255.255.255.0

   Default gateway : 10.10.6.1

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



Direct Connection PC to PowerCore Module

6.5 Run the PINGME.C Sample Program

Connect the crossover cable from your computer's Ethernet port to the PowerCore module's RJ-45 Ethernet connector. Open this sample program from the **SAMPLES\TCPIP\ICMP** folder, compile the program, and start it running under Dynamic C. The crossover cable is connected from your computer's Ethernet adapter to the PowerCore module's RJ-45 Ethernet connector. When the program starts running, the green DS2 LED on the PowerCore module should be on to indicate an Ethernet connection is made. (Note: If the green LED does not light, you may not be using a crossover cable, or if you are using a hub perhaps the power is off on the hub.)

The next step is to ping the board from your PC. This can be done by bringing up the MS-DOS window and running the pingme program:

```
ping 10.10.6.101
```

or by **Start > Run**

and typing the entry

```
ping 10.10.6.101
```

Notice that the yellow DS1 LED flashes on the PowerCore module while the ping is taking place, and indicates the transfer of data. The ping routine will ping the board four times and write a summary message on the screen describing the operation.

6.6 Running Additional Sample Programs

Many generic TCP/IP sample programs are available in the Dynamic C **SAMPLES\TCPIP** folder. The following sample programs specific to the PowerCore FLEX modules are in the Dynamic C **SAMPLES\PowerCoreFLEX\TCPIP** folder.

- **SMTP.c**—This program demonstrates using the SMTP library to send an e-mail when either switch S2 or S3 on the Prototyping Board is pressed. If you are using a direct connection, you will need an SMTP server on your host machine.
- **SSI.c**—This program demonstrates using a Web page to control LEDs DS5 and DS6 (LED0 and LED1) from the PowerCore Prototyping Board. Two "device LEDs" are created along with two buttons to toggle them. Users can use their Web browser to change the status of the LEDs to match the status on the Prototyping Board.

  As long as you have not modified the **TCPCONFIG 1** macro in the sample program, enter the following server address in your Web browser to bring up the Web page served by the sample program.

  http://10.10.6.100.

  Otherwise use the TCP/IP settings you entered in the **TCP_CONFIG.LIB** library.

- **TELNET.C**—This sample program takes anything that comes in on a port and sends it out Serial Port E. It uses a digital input to indicate that the TCP/IP connection should be closed and a digital output to toggle an LED to indicate that there is an active connection.

  PC Setup

  1. Start up a PC serial utility such as Tera Term or Hyperterminal.
  2. Configure the serial parameters for a baud rate of 19200, 8 bits, no parity and 1 stop bit.
  3. Enable the "Local Echo" option.
  4. Enable the option to append linefeeds to incoming and outgoing messages.

  Connect PC to PowerCore Prototyping Board

  1. Connect PC Tx to pin 4 on header J1 of the Prototyping Board (RXF).
  2. Connect PC Rx to pin 6 on header J1 of the Prototyping Board (TXF).
  3. Connect PC GND to pin 9 on header J1 of the Prototyping Board (GND).

  Once you have compiled and run the sample program, start a Telnet session on your PC (**Start > Run** `telnet 10.10.6.100`). As long as you have not modified the `TCPCONFIG 1` macro in the sample program, the IP address is 10.10.6.100 as shown; otherwise use the TCP/IP settings you entered in the **TCP_CONFIG.LIB** library.

  Now look at LED DS5 on the Prototyping Board—a blinking LED indicates that there is an active Telnet connection. You can type a message in the Telnet window, then view the message using a PC serial utility such as Tera Term. Press switch S2 on the Prototyping Board to close the Telnet connection; DS5 should stop blinking, indicating that the Telnet connection has been closed.

6.7 Where Do I Go From Here?

> **NOTE:** If you purchased your PowerCore module 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/.

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

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

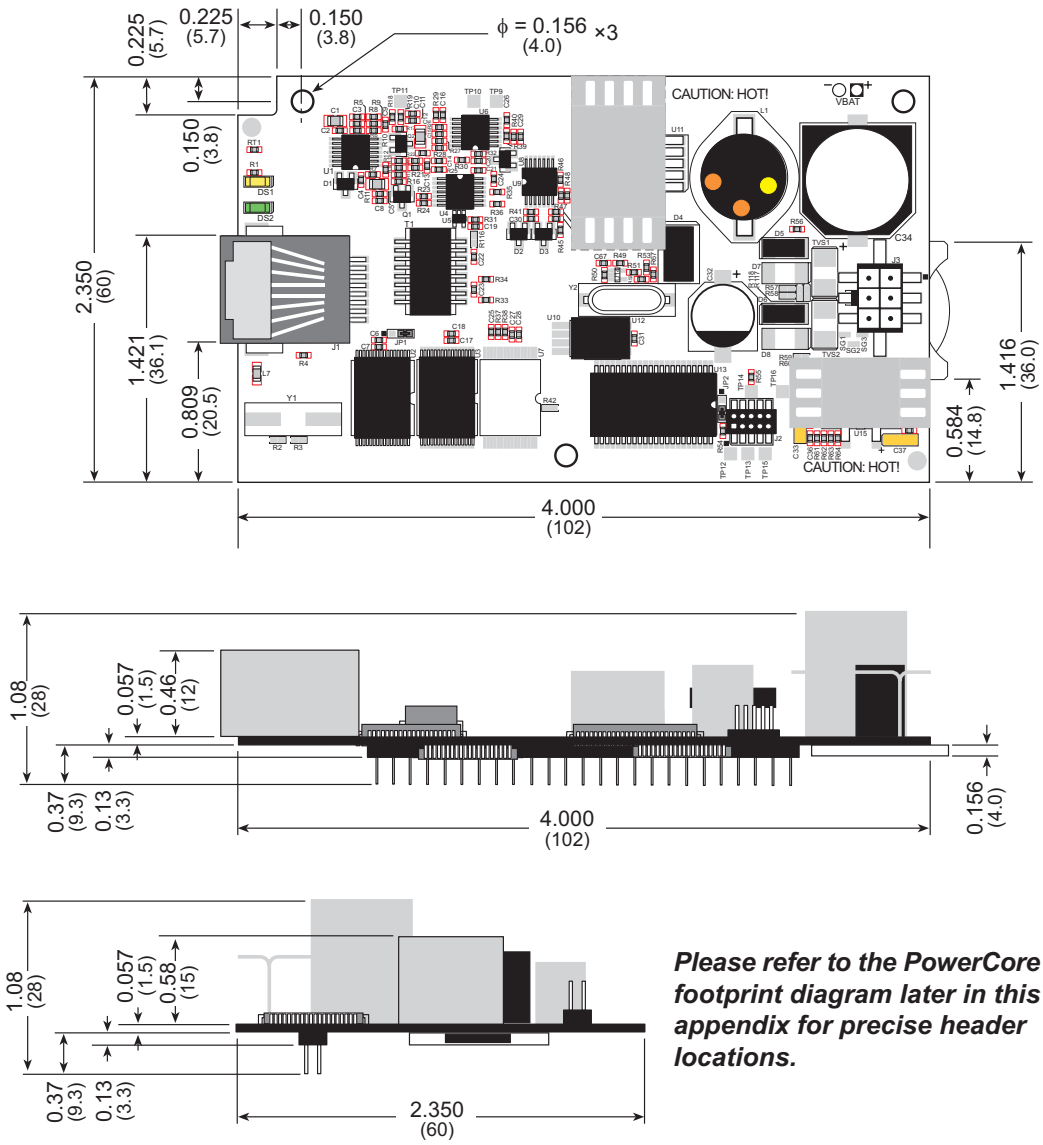
Please refer to the ***Dynamic C TCP/IP User's Manual*** to develop your own applications. ***An Introduction to TCP/IP*** provides background information on TCP/IP, and is available on the CD and on our Web site.

APPENDIX A. POWERCORE SPECIFICATIONS

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

A.1 Electrical and Mechanical Characteristics

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



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

Figure A-1. PowerCore 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).

NOTE: The battery holder and the RJ-45 jack both extend 0.1" (2.5 mm) beyond the edge of the board.

It is recommended that you allow for an “exclusion zone” of 0.04" (1 mm) around the PowerCore module in all directions when the PowerCore module is incorporated into an assembly that includes other printed circuit boards. An “exclusion zone” of 0.16" (4 mm) is recommended below the PowerCore module when the PowerCore module is plugged into another assembly. Figure A-2 shows this “exclusion zone.”

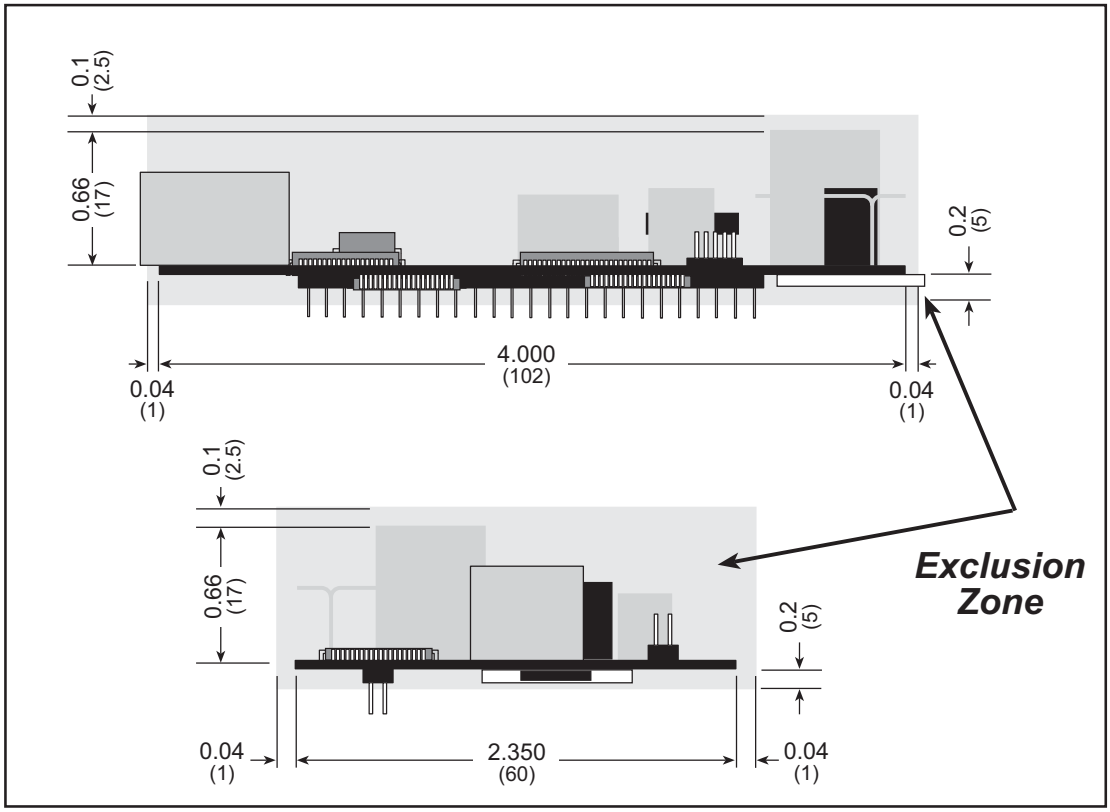


Figure A-2. PowerCore “Exclusion Zone”

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

The PowerCore modules were tested for heat dissipation over the specified operating temperature range, and normal heat dissipation by convection was found to be adequate. If you plan to use PowerCore module in a tightly enclosed space, additional forced-air cooling will likely be needed.

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

Table A-1. PowerCore Specifications

Parameter	PowerCore 3800	PowerCore 3810
Microprocessor	Rabbit 3000® at 51.6 MHz	Rabbit 3000® at 25.8 MHz
EMI Reduction	Spectrum spreader for reduced EMI (radiated emissions)	
Ethernet Port	10/100 compatible 10Base-T interface, RJ-45, 2 LEDs	—
SRAM	512K program (fast SRAM) + 512K data	256K data
Flash Memory (program)	512K	512K
Serial Flash Memory	1 Mbyte	—
Backup Battery	3 V lithium coin type 2032, 220 mA·h (to support RTC and data SRAM)	
General-Purpose I/O	39 parallel configurable digital I/0 lines	
Additional Inputs	2 startup mode, reset input	
Additional Outputs	Status, reset	
Analog Output	Ramp-generator for A/D conversion measurements	
Auxiliary I/O Bus	Can be configured for 8 data lines and 6 address lines (shared with parallel I/O lines)	
Serial Ports	Five 3.3 V, CMOS-compatible ports (shared with I/O) • all 5 configurable as asynchronous • 3 configurable as clocked serial (SPI) • 2 configurable as HDLC • 1 configurable as SDLC • 1 asynchronous serial port dedicated for programming	
Serial Rate	Maximum asynchronous baud rate = CLK/8	
Slave Interface	A slave port allows the PowerCore module to be used as an intelligent peripheral device slaved to a master processor, which may either be another Rabbit 3000 or any other type of processor	
Real-Time Clock	Yes	
Timers	Ten 8-bit timers (6 cascadable, 3 reserved for internal peripherals), one 10-bit timer with 2 match registers	
Watchdog/Supervisor	Yes	
Pulse-Width Modulators	4 PWM registers with 10-bit free-running counter and priority interrupts	
Input Capture	2-channel input capture can be used to time input signals from various port pins	
Quadrature Decoder	2-channel quadrature decoder accepts inputs from external incremental encoder modules	

Table A-1. PowerCore Specifications (continued)

Parameter		PowerCore 3800	PowerCore 3810
Input Power Options*	DC	Unregulated 8–43 V DC (draws 13.3 W)	Unregulated 8–40 V DC (draws 6.7 W)
	AC	24–60 V AC with center-tapped transformer (draws 13.3 W)	19–57 V AC with center-tapped transformer (draws 6.7 W)
		12–36 V AC with untapped standard transformer (draws 13.3 W)	10–29 V AC with untapped standard transformer (draws 6.7 W)
Current Limits for Onboard +5 V DC Voltage Regulators		2 A	1 A
Current Draw by Onboard Circuits		400 mA	150 mA
AC/DC Outputs†	+3.45 V DC	I_{3VoutM} = 350 mA	I_{3VoutM} = 550 mA
	+5 V DC	$I_{5VoutM} = [1600\ mA - I_{3VoutA}]$	$I_{5VoutM} = [850\ mA - I_{3VoutA}]$
	Unregulated AC/DC	$I_{unregM} = 1600\ mA - (I_{5VoutA} + I_{3VoutA}) \times \left(\frac{6.7\ V}{V_{IN}}\right)$	$I_{unregM} = 1850\ mA - (I_{5VoutA} + I_{3VoutA}) \times \left(\frac{6.7\ V}{V_{IN}}\right)$
Operating Temperature		–40°C to +70°C	
Humidity		5% to 95%, noncondensing	
Connectors		One 2 × 25, 0.1" pitch one 6-pin 3 mm locking one 2 × 5 for programming with 1.27 mm pitch	
Standoffs/Spacers		Provision for 3	
Board Size (without wiring harness)		2.350" × 4.000" × 1.08" (60 mm × 102 mm × 28 mm)	

* Additional power-supply options are available for PowerCore FLEX boards configured to your specifications. These are described in Section D.1.1.

† **NOTES**

+3.45 V DC power supply tolerance is ±150 mV; +5 V DC power supply tolerance is ±250 mV

I_{3VoutM} = maximum +3.45 V output current available to user's circuit

I_{3VoutA} = actual +3.45 V output current used by user's circuit

I_{5VoutM} = maximum +5 V output current available to user's circuit

I_{5VoutA} = actual +5 V output current used by user's circuit

I_{unregM} = maximum unregulated AC/DC output current available to user's circuit—note that you may need additional current capacity from your input power source to deliver this output current

The 6.7 V constant in the current calculation for I_{unregM} is derived from the 75% efficiency of the +5 V switching regulator.

An example is provided below to illustrate a typical calculation of the various output currents.

Example Current Calculation

Let's look at a 24 V AC power supply where the user's circuit consumes 300 mA @ +3.45 V, 550 mA @ +5 V, and 1500 mA @ 24 V AC. Can a PowerCore 3810 supply these needs?

I_{3VoutA} = 300 mA

I_{5VoutA} = 550 mA

I_{unregA} = 1500 mA = actual AC output current used by user's circuit

Now let's examine each of these currents in turn.

- I_{3VoutM} = 350 mA; since the user's circuit only needs 300 mA @ +3.45 V, this is **OK**.
- I_{5VoutM} = [850 mA - I_{3VoutA}] = [850 mA - 300 mA] = 550 mA; since the user's circuit needs 550 mA @ +5 V, this is **OK**.
- I_{unregM}

  $= 1850\ \text{mA} - (I_{5VoutA} + I_{3VoutA}) \times \left(\frac{6.7\ \text{V}}{V_{IN}}\right)$

  $= 1850\ \text{mA} - (550\text{mA} + 300\ \text{mA}) \times \left(\frac{6.7\ \text{V}}{24\ \text{V}}\right)$

  $= 1613\ \text{mA}$

  Since the user's circuit only needs 1500 mA @ +24 V AC, this is **OK**.

A.1.1 Headers and Spacers

The PowerCore module uses one header at J4 for physical connection to other boards. J4 is a 2 × 25 SMT header with a 0.1" pin spacing. J2, the programming port, is a 2 × 5 header with a 1.27 mm pin spacing. J3 is a 2 × 3 locking connector used for power-supply connections.

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

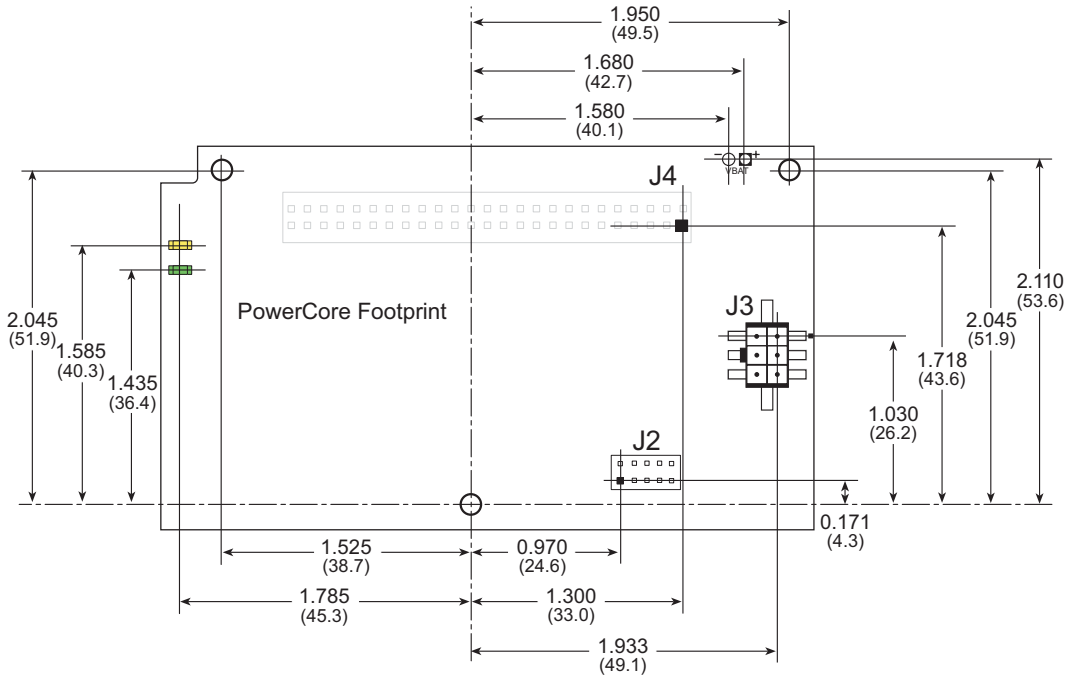


Figure A-3. User Board Footprint for PowerCore Module

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

The PowerCore module has three mounting holes whose diameter is 0.125" (3.2 mm). These holes can be used with spacers and mounting screws to secure the PowerCore module for use on a high-vibration environment.

A.2 Bus Loading

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

Table A-2 lists the capacitance for the various PowerCore module 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 PowerCore module 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	51.6	50
	25.8	70

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

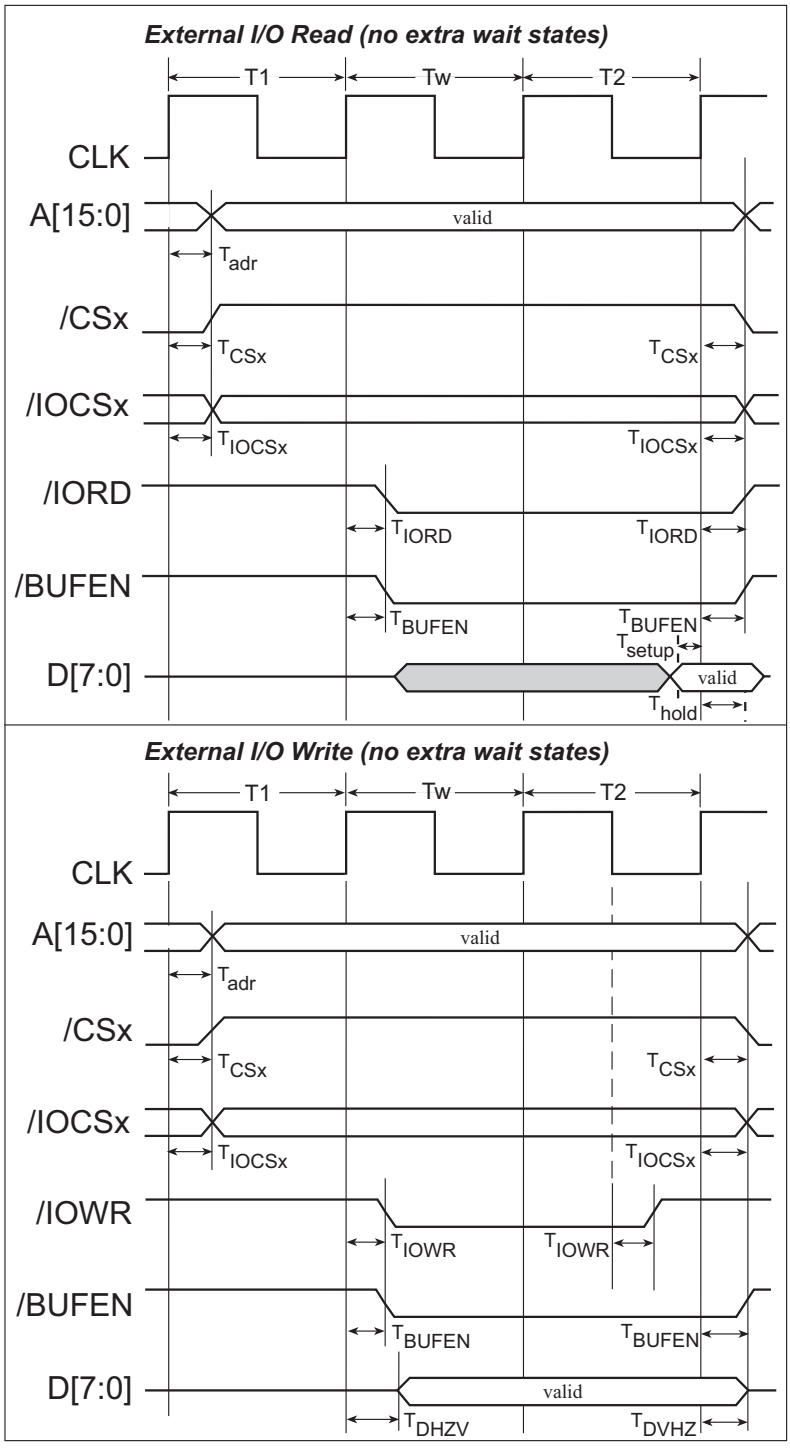


Figure A-4. I/O Read and Write Cycles—No Extra Wait States

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

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

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

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

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

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

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

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

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

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

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

A.3 Rabbit 3000 DC Characteristics

Table A-5. Rabbit 3000 Absolute Maximum Ratings

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

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

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

Table A-6. 3.3 Volt DC Characteristics

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

NOTE: PowerCore modules operate at a nominal +3.45 V DC.

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 PowerCore.

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	6.8	6.8

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

A.5 Jumper Configurations

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

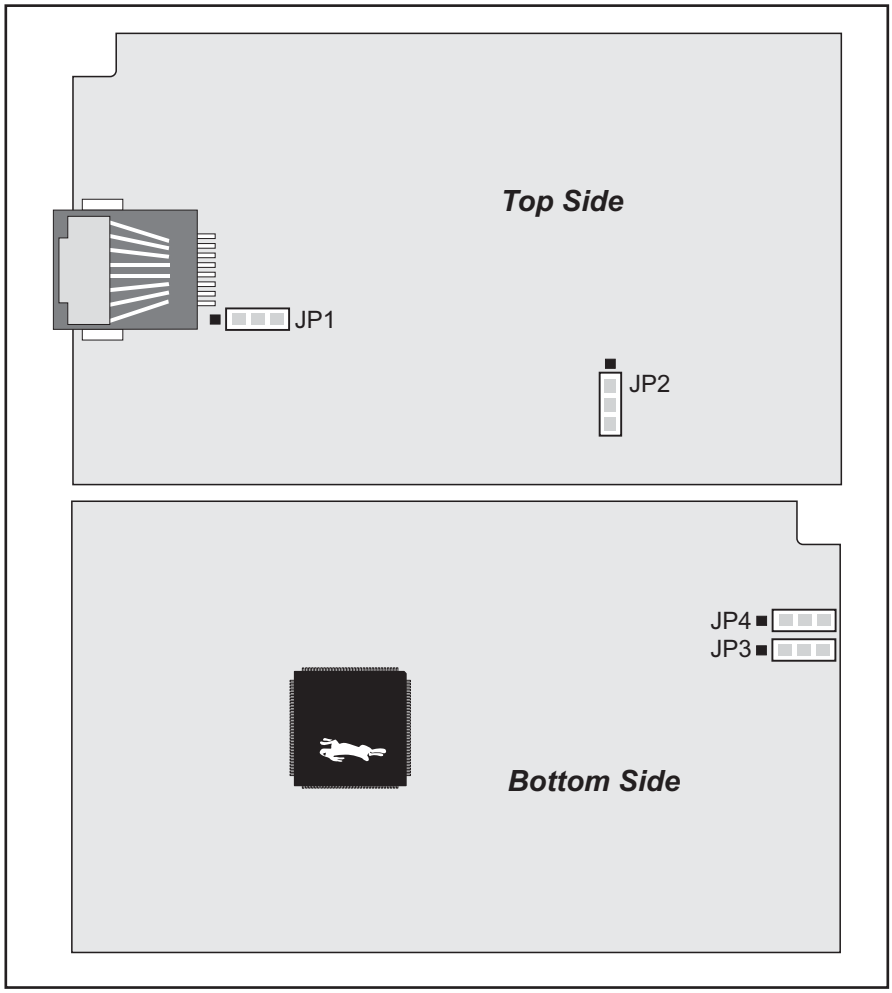


Figure A-5. Location of PowerCore Configurable Positions

Table A-8 lists the configuration options.

Table A-8. PowerCore Jumper Configurations

Header	Description	Pins Connected		Factory Default
JP1	Data SRAM Size	1–2	256K	PowerCore 3810
		2–3	512K	PowerCore 3800
JP2	Flash Memory Bank Select	1–2	Bank Mode	
		2–3	Normal Mode	✕
JP3	Ethernet LEDs	n.c.	No Ethernet	PowerCore 3810
		1–2	10/100-compatible 10Base-T Ethernet interface	PowerCore 3800
		2–3	100Base-T Ethernet interface	
JP4	Ethernet LEDs	n.c.	No Ethernet	PowerCore 3810
		1–2	10/100-compatible 10Base-T Ethernet interface	PowerCore 3800
		2–3	100Base-T Ethernet interface	

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

A.6 Conformal Coating

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

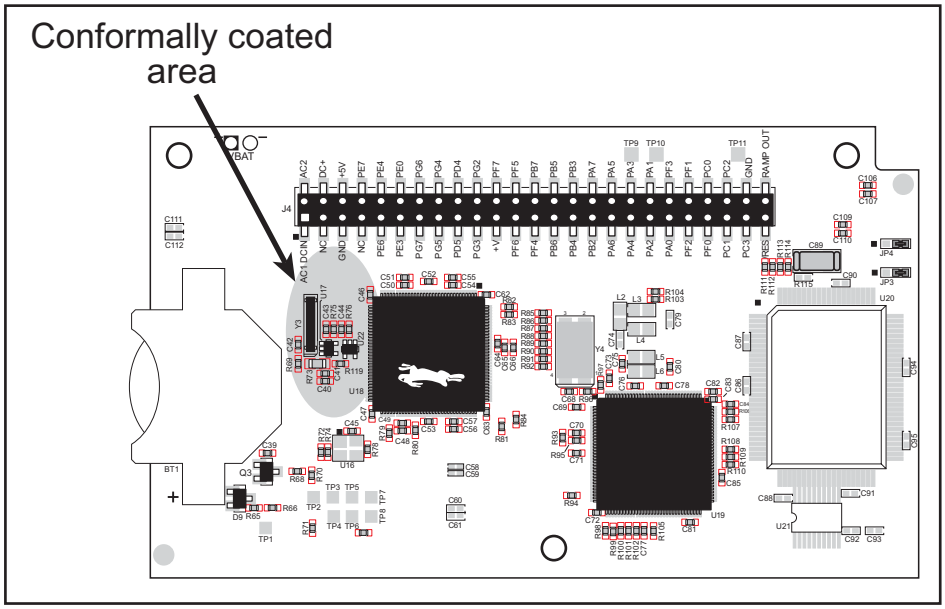


Figure A-6. PowerCore Areas Receiving Conformal Coating

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

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

APPENDIX B. PROTOTYPING BOARD

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

B.1 Introduction

The Prototyping Board included in the Development Kit provides some basic I/O peripherals (RS-232, triacs, 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 PowerCore module itself.

Figure B-1 shows the Prototyping Board and identifies its main features.

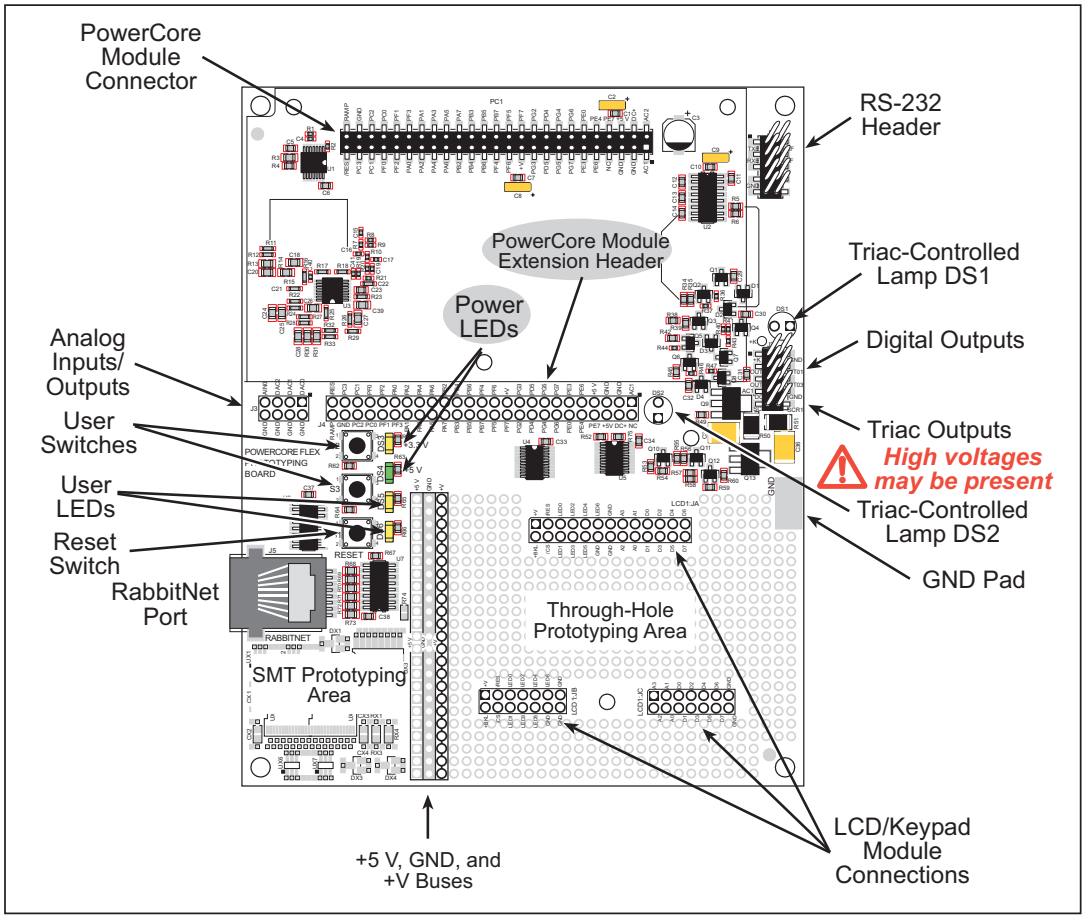


Figure B-1. Prototyping Board

CAUTION: High AC voltages may be present on pin 7 of header J2 and on the triac outputs on header J2.

B.1.1 Prototyping Board Features

- **Power Connection**—+5 V, +3.45 V, and AC1_DCIN voltages are supplied to the Prototyping Board by the PowerCore module. These voltages can be used to power customer-installed parts in the prototyping area.
- **Power LEDs**—The two power LEDs at DS3 and DS4 light whenever +3.45 V and + 5 V is being supplied to the Prototyping Board from the PowerCore module.
- **Prototyping Area**—A generous prototyping area has been provided for the installation of through-hole components. +5 V, +3.45 V, and Ground buses run along one edge of this area. Several areas for surface-mount devices are also available. Each SMT pad is connected to a hole designed to accept a 30 AWG solid wire.
- **Reset Switch**—A momentary-contact, normally open switch is connected directly to the PowerCore module'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 PC3 and PG4 pins of the PowerCore module and may be read as inputs by sample applications.

  Two user LEDs (DS5–DS6) are connected to I/O pins PD5 and PC2 of the PowerCore module.
- **LCD/Keypad Module**—Rabbit Semiconductor's LCD/keypad module may be plugged in directly to headers LCD1:JA, LCD1:JB, and LCD1:JC. The signals on headers LCD1:JB and LCD1:JC will be available only if the LCD/keypad module is plugged in to header LCD1:JA. Appendix C provides complete information for mounting and using the LCD/keypad module.
- **Module Extension Header**—The PowerCore module's pin set is duplicated at header J4. Developers can solder wires directly into the appropriate holes, or, for more flexible development, a 2 × 25 header strip with a 0.1" pitch can be soldered into place. See Figure B-3 for the header pinouts.
- **Digital I/O**—Four digital outputs and a +K connection are available on header J2. Two digital inputs are available at header location J4. See Figure B-3 for the header pinouts.
- **Triacs**—Two Z0107MN triacs are installed on the Prototyping Board, and can be used for general-purpose AC switching applications such as electrovalves, pumps, door locks, small lamp control, and fan-speed control. Their outputs are available on header J2. Miniature incandescent lamps may also be installed on the Prototyping Board to observe the triac operation.
- **RS-232**—Two 3-wire serial ports or one 5-wire RS-232 serial port are available on the Prototyping Board at header J1.
- **RabbitNet Port**—One RS-422 RabbitNet port is available to allow RabbitNet peripheral cards to be used with the PowerCore Prototyping Board.

B.2 Mechanical Dimensions and Layout

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

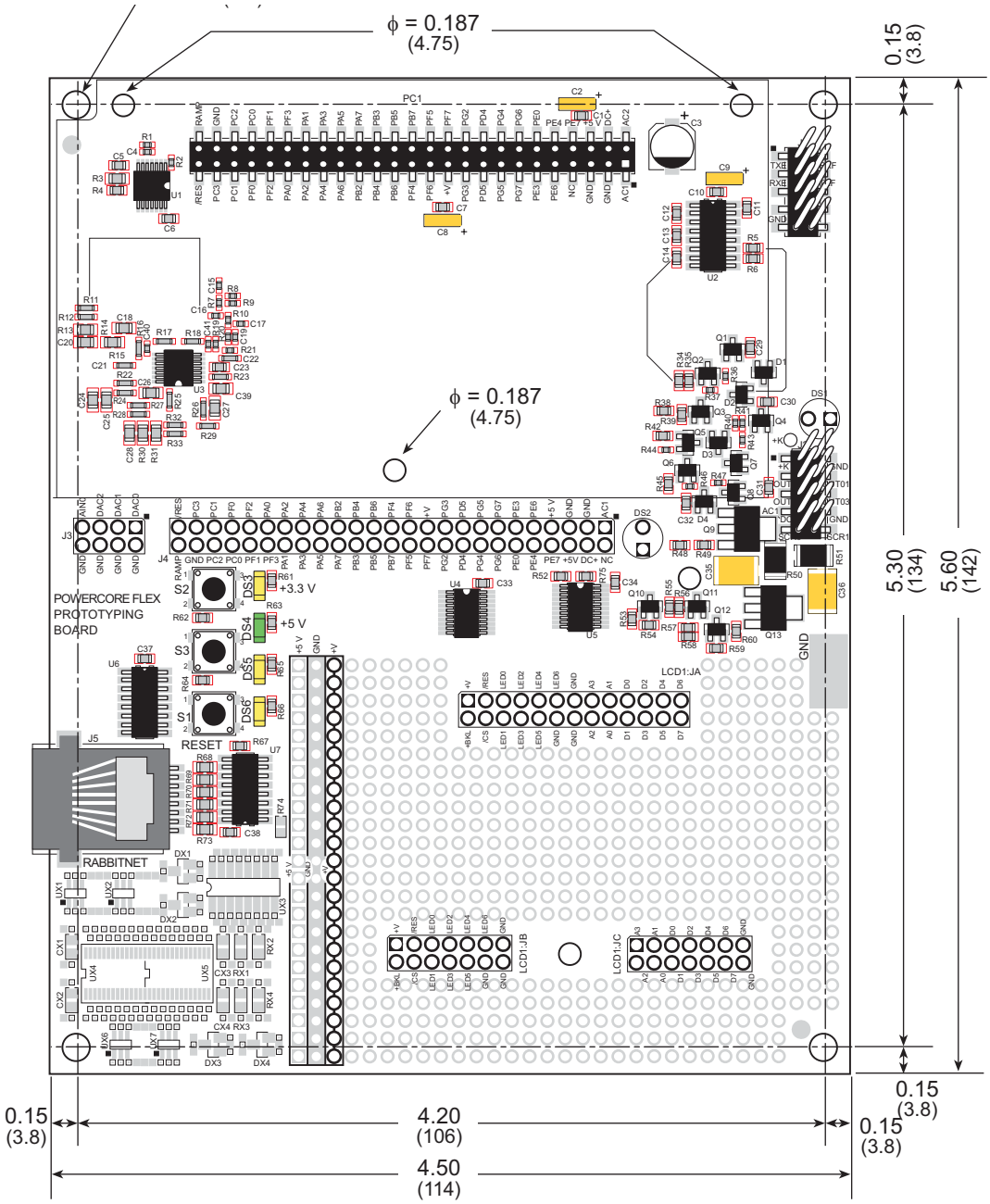


Figure B-2. Prototyping Board Dimensions

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

NOTE: The RJ-45 RabbitNet jack extends 0.1" (2.5 mm) beyond the edge of the board.

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

Table B-1. PowerCore Prototyping Board Specifications

Parameter	Specification
Board Size	4.50" × 5.60" × 0.88" (114 mm × 142 mm × 22 mm)
Operating Temperature	–40°C to +70°C
Humidity	5% to 95%, noncondensing
Power from PowerCore Module (max. current draw for user-added circuits)	• I_{3Vout} = 350 mA (max.) (PowerCore 3800) I_{3Vout} = 550 mA (max.) (PowerCore 3810) • I_{5Vout} = [1600 mA - I_{3Vout}] (PowerCore 3800) I_{5Vout} = [850 mA - I_{3Vout}] (PowerCore 3810)*
Digital Inputs	4 inputs pulled up, ± 36 V DC, switching threshold 0.9–2.3 V typical
Digital Outputs	2 sinking outputs,+40 V DC, 1 A maximum per channel, 2 sourcing outputs,+40 V DC, 500 mA maximum per channel
Analog Input	One 12-bit resolution, 10-bit accuracy, input range 0–10 V, 400 samples/s
Analog Outputs	3.0 V with typical 10 kΩ load, 10-bit resolution • DAC0—settling time 0.4 ms • DAC1—settling time 0.9 ms • DAC2—settling time 2.5 ms
Triacs	Two Z0107MN triacs† • 1 A on-state rms current, 30 V AC max. @ –40°C to +50°C • 0.8 A on-state rms current, 30 V AC max. @ 50°C to +70°C
Serial Ports	Two 3-wire RS-232 *or* one RS-232 with RTS/CTS
Other Serial Interfaces	RabbitNet RS-422 SPI port
Other Interfaces	LCD/keypad module
LEDs	Four LEDs • two power-supply indicators (+3.45 V and + 5 V) • two user-configurable LEDs
Prototyping Area	Throughhole, 0.1" spacing, additional space for SMT components
Connectors	• one 2 × 25, 0.1" pitch socket for PowerCore module • two 2 × 5, 0.1" pitch headers for serial ports, digital I/O, and triac outputs • one RJ-45 RabbitNet jack
Standoffs/Spacers	5 plastic standoffs

* Additional current can be supplied by the 5 V regulator if the 3.45 V regulator is not supplying its maximum current.

† Only 300 mA (rms) current is available when using the AC transformer from the PowerCore Tool Kit when the maximum current is being drawn from the PowerCore module's regulated power supplies.

B.3 Power Supply

The PowerCore Prototyping Board uses the regulated and unregulated power from the PowerCore module.

- regulated +3.45 V (+V)
- regulated +5 V
- unregulated DC voltage (DC+)
- AC voltage (AC1_DCIN)

B.4 Using the Prototyping Board

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

The Prototyping Board pinouts are shown in Figure B-3.

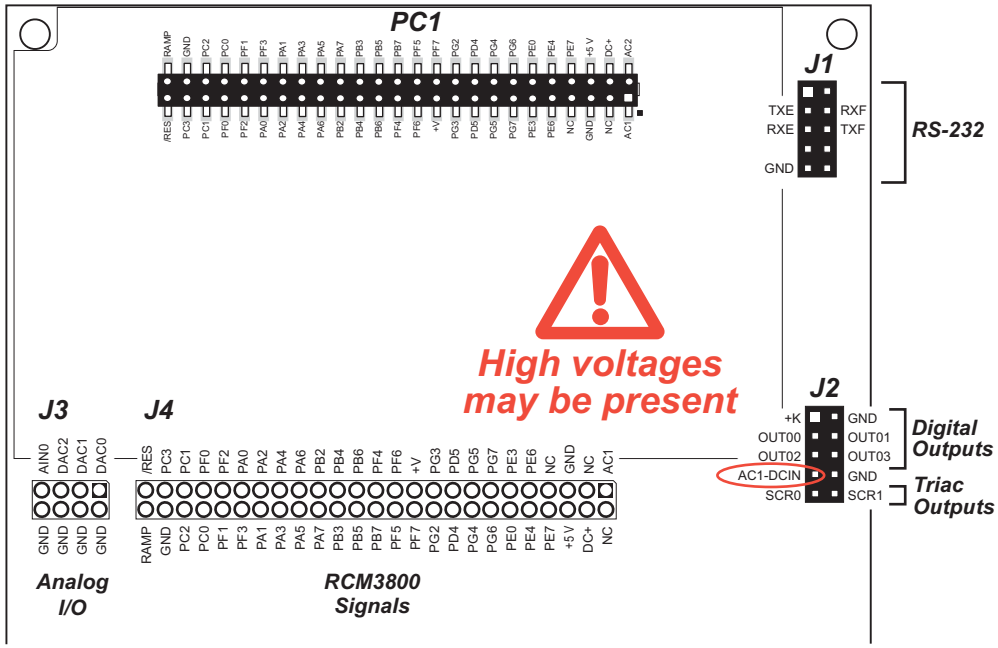


Figure B-3. PowerCore Prototyping Board Pinout

CAUTION: High AC voltages may be present on pin 7 of header J2 and on the triac outputs on header J2. Exercise extreme care to make sure that you do not inadvertently connect one of the triac outputs or pin 7 to other pins on header J2

The Prototyping Board comes with the basic components necessary to demonstrate the operation of the PowerCore module. Two user LEDs (DS5 and DS6) are connected to PowerCore module's pins PC2 and PD5, and may be driven as output indicators as shown in the sample applications. Two switches (S2 and S3) are connected to PG4 and PC3 to demonstrate the interface to the Rabbit 3000 microprocessor. Reset switch S1 is the hardware reset for the PowerCore module.

The Prototyping Board provides the user with the PowerCore module's connection points brought out conveniently to labeled points at J4 on the Prototyping Board. Although J4 is unstuffed, a 2 × 25 header is included in the bag of parts.

Analog signals are available at J3 on the Prototyping Board. Although J3 is unstuffed, a 2 × 4 header is included in the bag of parts.

RS-232 signals are available on header J1, and digital and triac outputs are available on header J2.

There is a through-hole prototyping space available on the Prototyping Board. The holes in the prototyping area are spaced at 0.1" (2.5 mm). +3.45 V, +5 V, and GND traces run along one edges of the prototyping area. Small to medium circuits can be prototyped using point-to-point wiring with 20 to 30 AWG wire between the prototyping area, the +3.45 V, +5 V, and GND traces, and the surrounding area where surface-mount components may be installed. Small holes are provided around the surface-mounted components that may be installed around the prototyping area.

B.4.1 Adding Other Components

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

B.4.2 Digital I/O

B.4.2.1 Digital Inputs

The PowerCore Prototyping Board has two digital inputs connected to a switch, IN0 and IN2. The inputs are pulled up to +3.45 V as shown in Figure B-4.

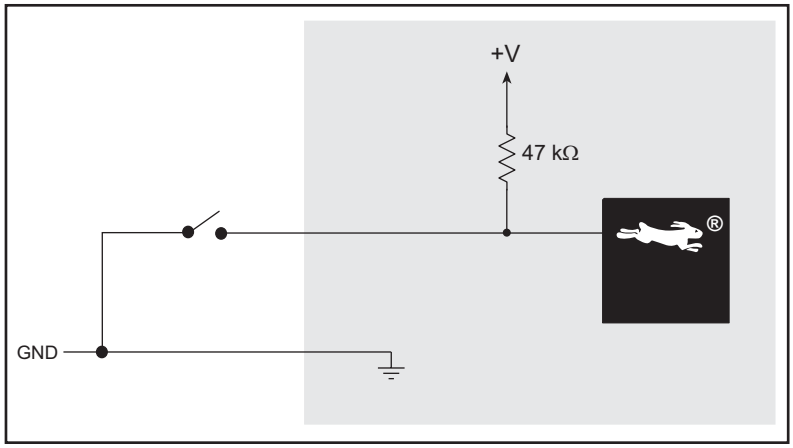


Figure B-4. PowerCore Prototyping Board Digital Inputs

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

B.4.3 Digital Outputs

Four digital outputs, two sinking and two sourcing, are available on header J2. The sinking outputs, OUT00 and OUT01, can each sink up to 1 A; the sourcing outputs, OUT02 and OUT03, can each source up to 500 mA. Figure B-5 shows a schematic diagram for both types of output.

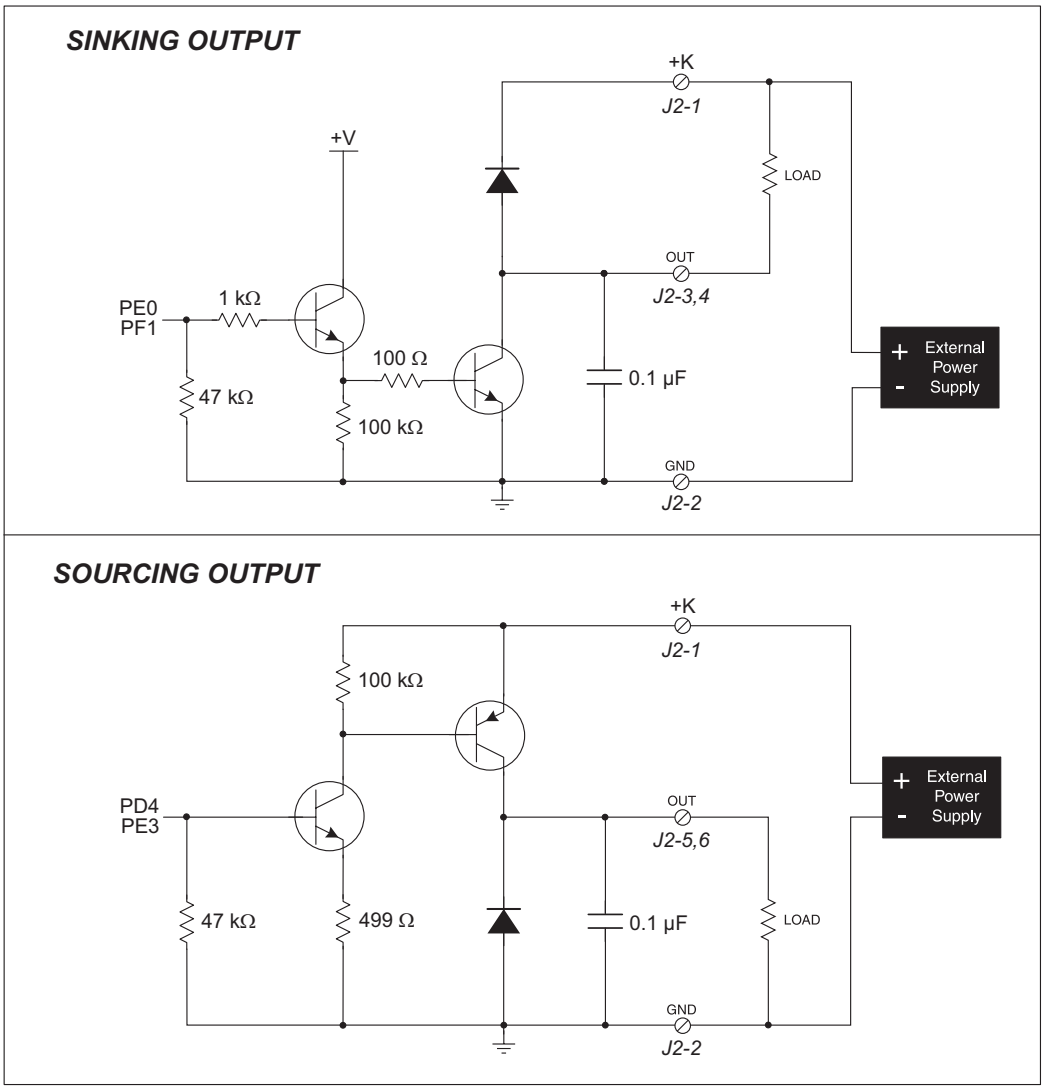


Figure B-5. PowerCore Prototyping Board Digital Outputs

B.4.4 Triac Outputs

The Prototyping Board has two triacs, each of which can handle up to 1 A. The Rabbit 3000 can enable or disable the triacs via software calls. Figure B-6 shows a schematic diagram for the two triac outputs.

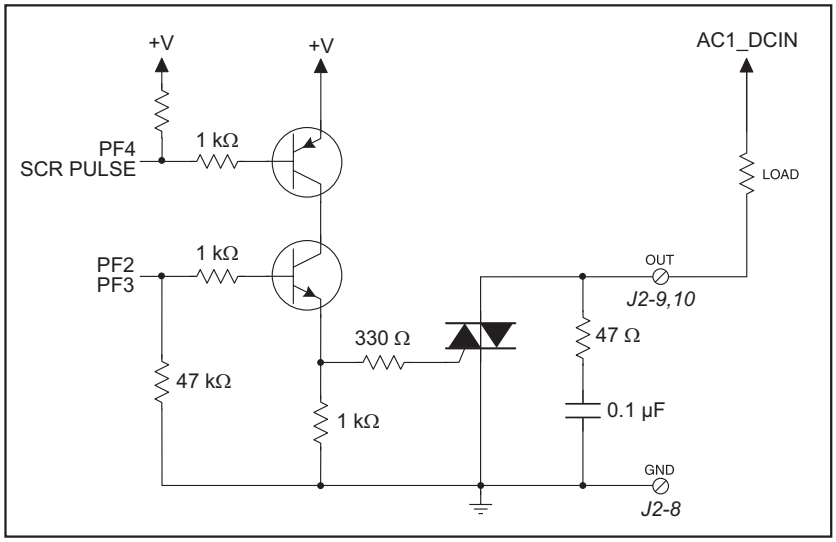
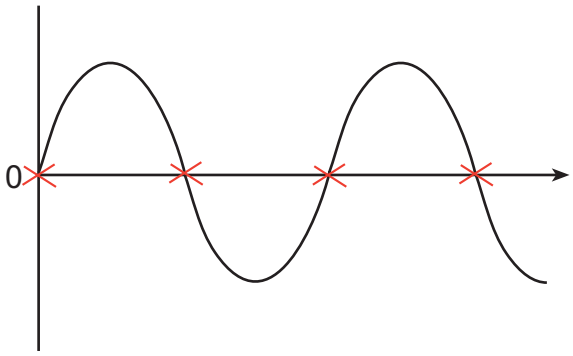


Figure B-6. PowerCore Prototyping Board Triac Outputs

When the triac outputs are being used, the AC must be supplied by a transformer with one side of the transformer connected to logic ground. This connection is made automatically when you use the center-tapped transformer supplied with the PowerCore Tool Kit, or you can use a half-wave rectified power supply.

The zero-crossover interrupt circuit on the PowerCore module, shown below in Figure B-7, allows the turning on of the triac output to be synchronized with the AC waveform. An interrupt is sent to the microprocessor when the AC voltage crosses zero—this allows the software to respond to the event. The triac output is turned off automatically at the next crossover after the gate is disabled.



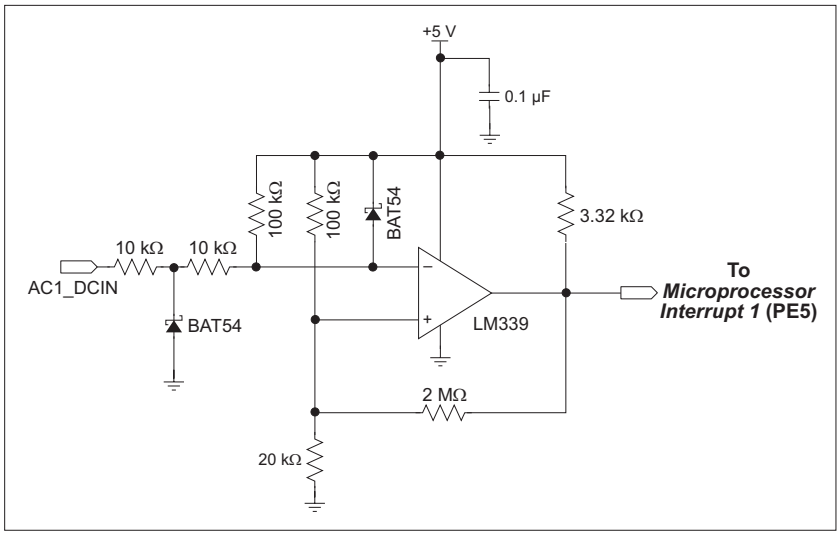
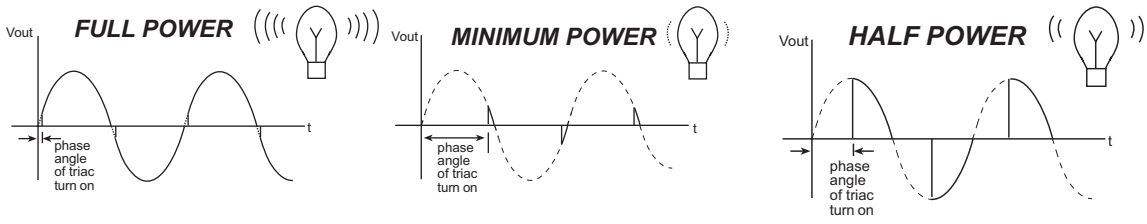


Figure B-7. PowerCore Crossover Detection Circuit

Since the triacs automatically turn themselves off at each zero crossing, power can be applied just after a zero crossing for full power to the other device, or most of the way between zero crossings for minimum power to the other device, or anywhere across the wave phase.



B.4.5 Analog I/O

B.4.5.1 A/D Converter Input

The Prototyping Board has one A/D converter circuit, shown below in Figure B-8.

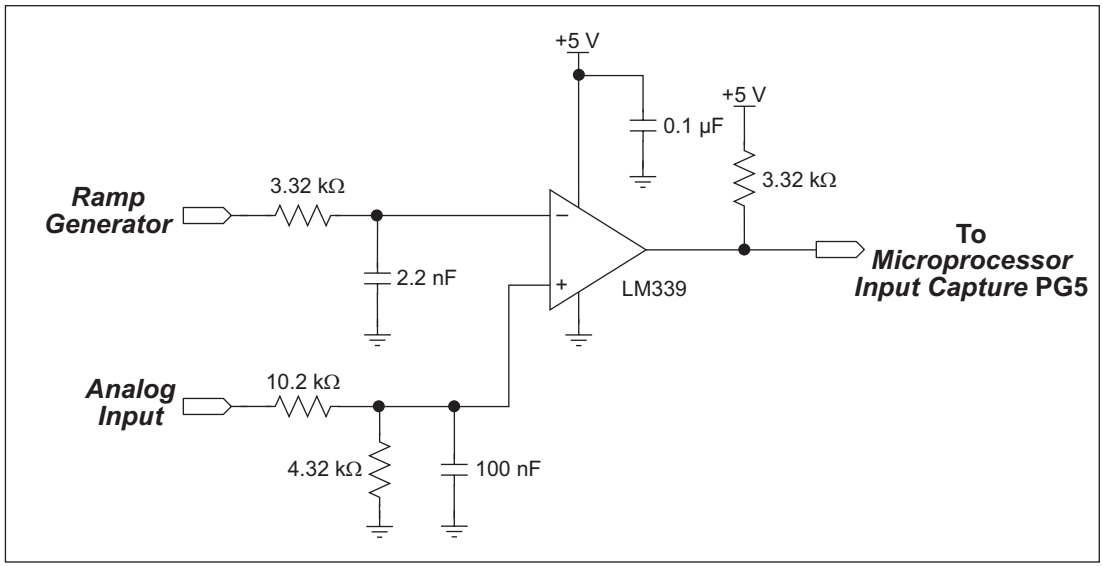


Figure B-8. PowerCore Prototyping Board A/D Converter Circuit

The ramp generator makes it possible to measure analog voltages using LM339 comparators and the pulse capture capabilities of the Rabbit 3000 microprocessor to convert time into voltage. One example of this analog measurement capability is the A/D measurement circuit on the PowerCore Prototyping Board.

The circuit is designed to accept input voltages of 0–10 V. An A/D converter measurement is implemented when a pulse from the comparator is routed to an input capture, PG5, on the Rabbit 3000. The counter in the Rabbit 3000 starts at the beginning of the ramp, and stops when the ramp crosses the input signal. The full scale is approximately 5000 counts, which yields a measurement resolution of at least 12 bits. The end of the ramp drives an interrupt in the Rabbit 3000, which then retrieves the count and stores it so that it can be accessed by a software function call. The interrupt routine can be set up to average inputs and to detect out-of-range signals.

The absolute accuracy of the voltage measurements is typically better than ± 5 mV.

B.4.5.2 D/A Converter Circuits

The Prototyping Board has three D/A converter circuits, each with a different settling time:

- DAC0—settling time 0.4 ms
- DAC1—settling time 0.9 ms
- DAC2—settling time 2.5 ms

Figure B-9 shows a schematic for the D/A converter circuits used for DAC1 and DAC2.

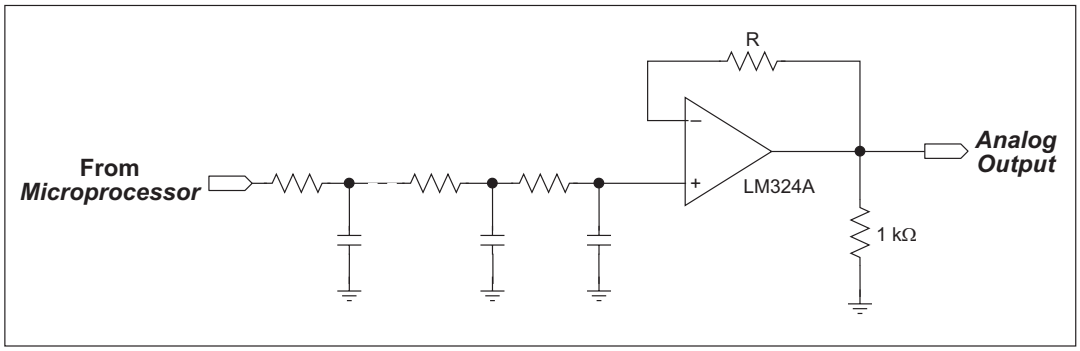


Figure B-9. PowerCore Prototyping Board D/A Converter Circuits DAC1 and DAC2

The three D/A converters differ in the number and kind of RC filtering stages.

- DAC0 has a faster response filter that is still able to filter out PWM noise. It uses a 5-pole active filter configuration with two op-amps.
- DAC1 has a less expensive 5-pole passive filter that has a slower response, but still filters out PWM noise.
- DAC2 has the least expensive filter, a 3-pole passive filter. The trade-off is a slower response time for the D/A converter.

Each D/A converter outputs 3.0 V into a typical 10 kΩ load.

B.4.6 Serial Communication

The PowerCore Prototyping Board allows you to access three of the serial ports on the PowerCore module. Table B-2 summarizes the use of the three serial ports.

Table B-2. PowerCore Prototyping Board Serial Ports

Serial Port	Signal Header	Use
D	J5 (RJ-45 jack)	RabbitNet
E	J1	RS-232
F	J1	RS-232

B.4.6.1 RS-232

RS-232 serial communication on the PowerCore Prototyping Board is supported by an RS-232 transceiver installed at U2. 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 +3.3 V input becomes approximately -7 V and 0 V is output as +7 V. The RS-232 transceiver also provides the proper line loading for reliable communication.

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

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

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

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

`SERX_RTS_BIT`—The bit number for the RTS line.

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

`SERX_CTS_BIT`—The bit number for the CTS line.

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

```
#define EINBUFSIZE  15
#define EOUTBUFSIZE 15

#define FINBUFSIZE  15
#define FOUTBUFSIZE 15

#ifndef _232BAUD
#define _232BAUD 115200
#endif

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

B.4.6.2 RabbitNet Ports

The RJ-45 jack labeled ***RabbitNet*** is a clocked SPI RS-422 serial I/O expansion port for use with RabbitNet peripheral boards. The ***RabbitNet*** jack does ***not*** support Ethernet connections. Additional information on RabbitNet peripheral boards is available in Appendix E and on the Web at www.rabbit.com/products/ExpSystems/.

B.4.7 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.

B.5 Use of Rabbit 3000 Parallel Ports

Table B-3 lists the Rabbit 3000 parallel ports and their use for the PowerCore Prototyping Board.

Table B-3. PowerCore Prototyping Board Use of Rabbit 3000 Parallel Ports

Port	I/O	Use		Initial State
PA0–PA7	Input/Output	ID0–ID7		Pulled up
PB2–PB5	Output	IA0–IA3		High
PB6–PB7	Output	Spare pins		High
PC0	Output	TXD SPI	Serial Port D	High (disabled)
PC1	Input	RXD SPI		Pulled up
PC2	Output	LED1 (DS6)		High (disabled)
PC3	Input	Switch S3		Pulled up
PD4	Output	Sourcing output (OUT02)		Low
PD5	Output	LED0 (DS5)		High
PE0	Output	Sinking output (OUT01)		Low
PE1	Input	A/D converter interrupt		—
PE3	Output	Sourcing output (OUT03)		Low
PE4	Output	RabbitNet chip select		High
PE6	Output	LCD/keypad module		High
PE7	Output	LCD/keypad module		High
PF0	Output	RabbitNet CLKD		Low (disabled)
PF1	Output	Sinking output (OUT00)		Low (disabled)
PF2	Output	Triac output (SCR0)		Low (disabled)
PF3	Output	Triac output (SCR1)		Low (disabled)
PF4	Output	Triac pulse		High (disabled)
PF5	Output	D/A converter output (DAC2)		Low (disabled)
PF6	Output	D/A converter output (DAC1)		Low (disabled)
PF7	Output	D/A converter output (DAC0)		Low (disabled)
PG2	Output	TXF RS-232	Serial Port F	High
PG3	Input	RXF RS-232		Pulled up
PG4	Input	Switch S2		Pulled up
PG5	Input	A/D converter input (AIN0)		—
PG6	Output	TXE RS-232	Serial Port E	High
PG7	Input	RXE RS-232		Pulled up

APPENDIX C. LCD/KEYPAD MODULE

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

C.1 Specifications

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.

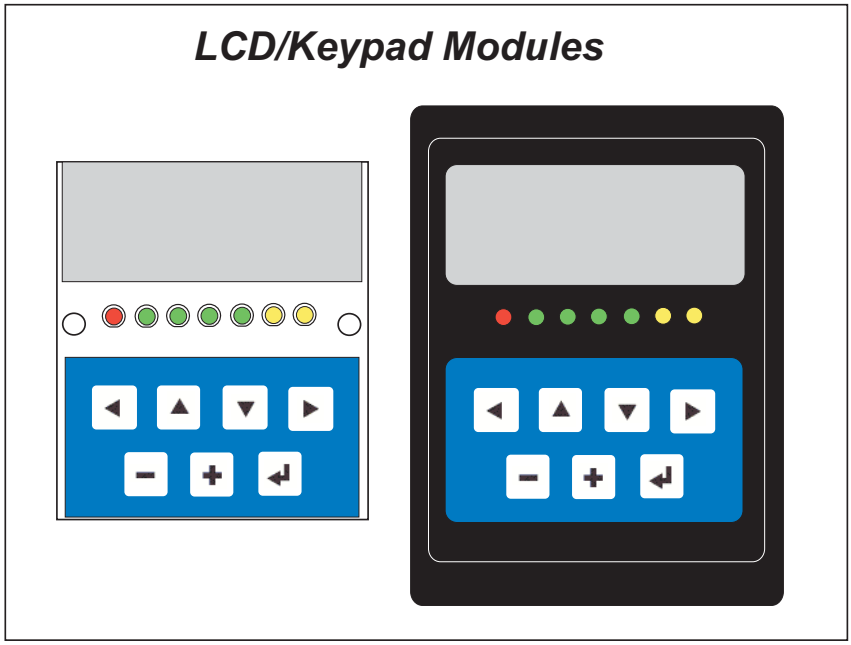


Figure C-1. LCD/Keypad Modules Versions

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 Rabbit Semiconductor sales representative or your authorized 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 Semiconuctor sales representative or authorized distributor.

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

Table C-1. LCD/Keypad Specifications

Parameter	Specification
Board Size	2.60" x 3.00" x 0.75" (66 mm x 76 mm x 19 mm)
Bezel Size	4.50" × 3.60" × 0.23" (114 mm × 91 mm × 6 mm)
Temperature	Operating Range: 0°C to +50°C Storage Range: –40°C to +85°C
Humidity	5% to 95%, noncondensing
Power Consumption	1.5 W without backlight*
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

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

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

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

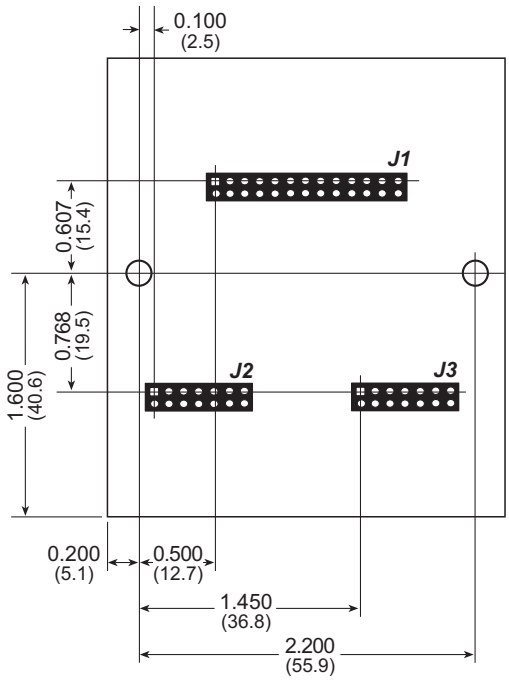


Figure C-2. User Board Footprint for LCD/Keypad Module

C.2 Contrast Adjustments for All Boards

Starting in 2005, LCD/keypad modules were factory-configured to optimize their contrast based on the voltage of the system they would be used in. Be sure to select a KDU3V LCD/keypad module for use with the PowerCore 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. While LCD/keypad modules configured for 5 V may be used with the 3.3 V PowerCore Prototyping Board by using the potentiometer to adjust the contrast, the backlight is likely to be dim.

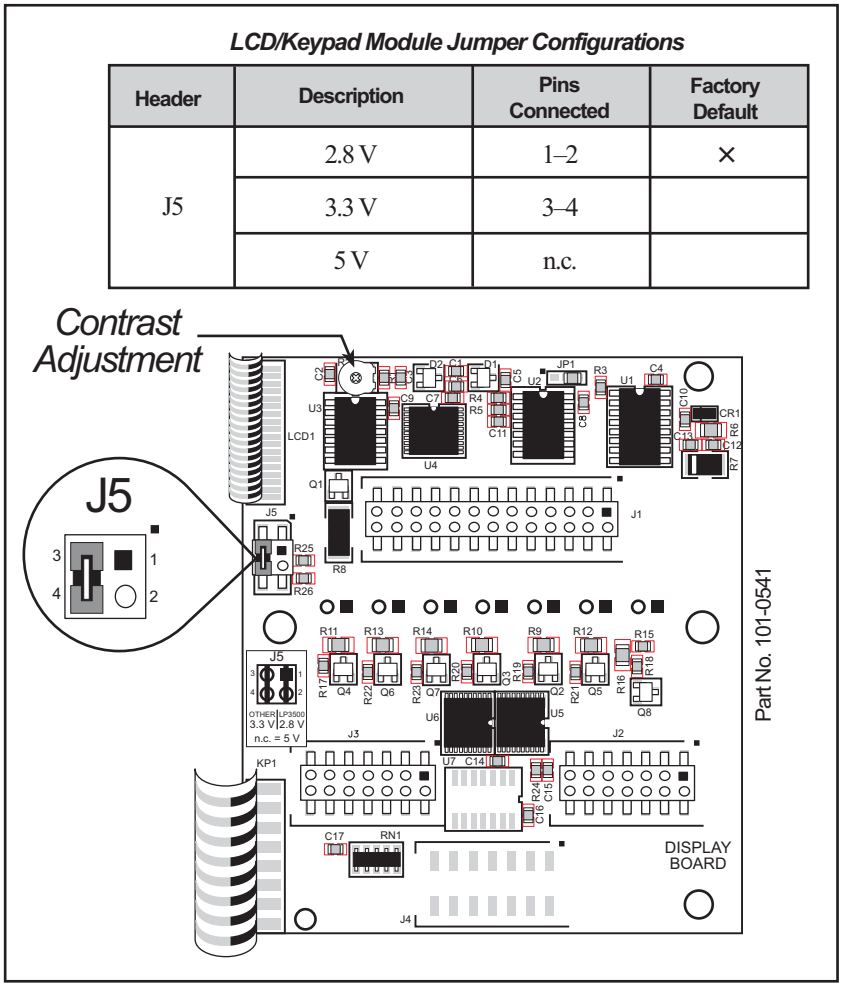


Figure C-3. LCD/Keypad Module Contrast Adjustments

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 setting the jumper across pins 3–4 on header J5 as shown in Figure C-3. Only one of these two options is available on 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 PowerCore Prototyping Board. These 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.

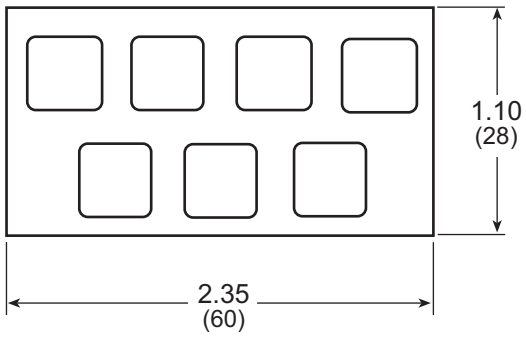


Figure C-4. Keypad Template

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.

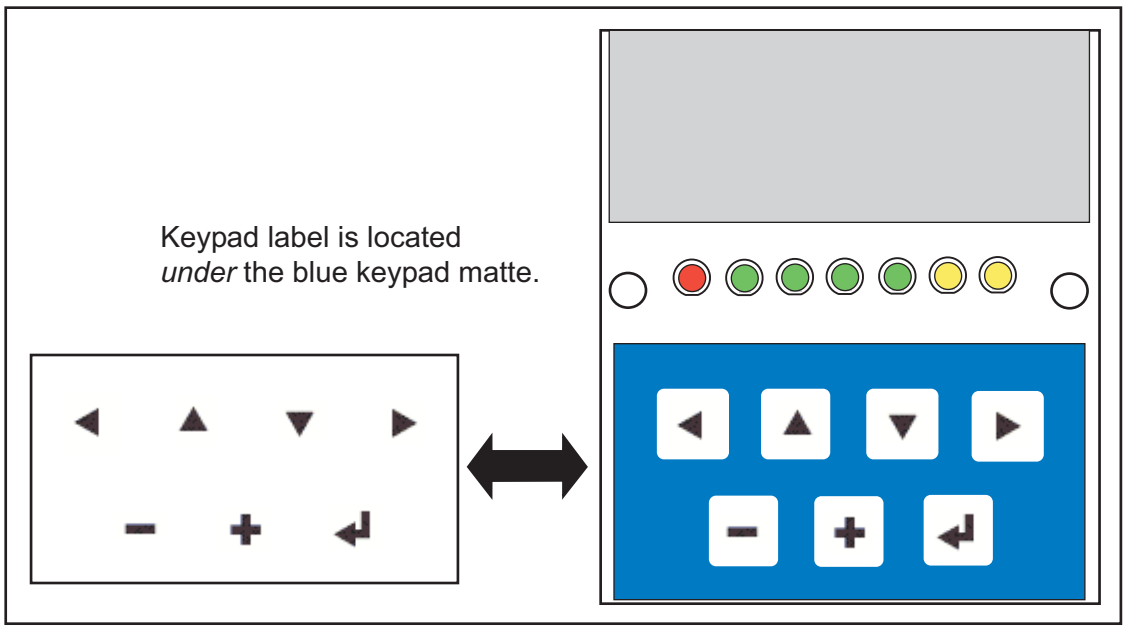


Figure C-5. Removing and Inserting Keypad Label

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

C.4 Header Pinouts

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

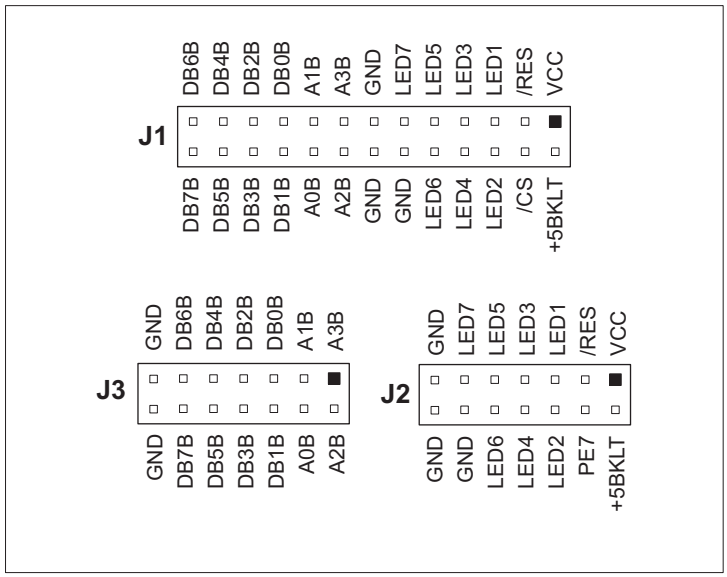


Figure C-6. LCD/Keypad Module Pinouts

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 PowerCore Prototyping Board, you will need to install connectors to attach the LCD/keypad module to the Prototyping Board. These connectors are included with the PowerCore Tool Kit.

First solder the 2 × 13 connector to location LCD1:JA on the PowerCore 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 Prototyping Board, solder two additional 2 × 7 connectors at locations LCD1:JB and LCD1:JC on the Prototyping Board. Section C.6, “Mounting LCD/Keypad Module on the Prototyping Board,” explains how to mount the LCD/keypad module on the Prototyping Board.

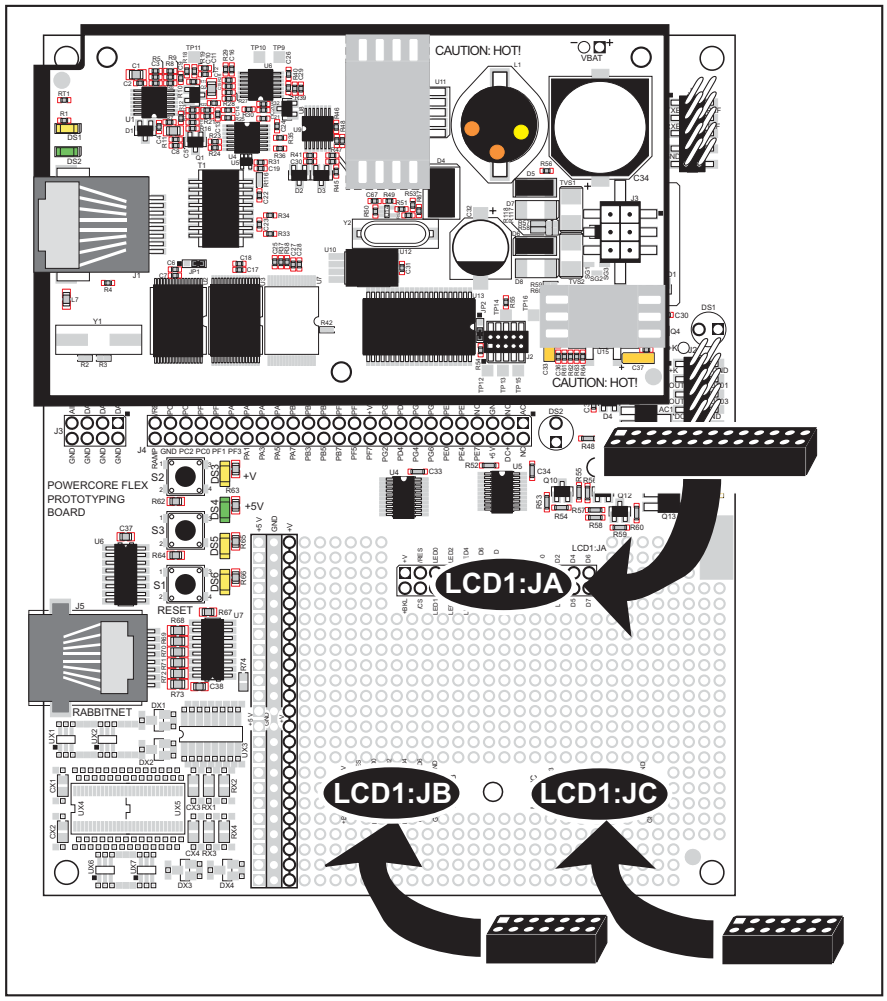


Figure C-7. Solder Connectors to RC3800 Prototyping Board

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

Install the LCD/keypad module on header sockets LCD1:JA, LCD1:JB, and LCD1:JC 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.

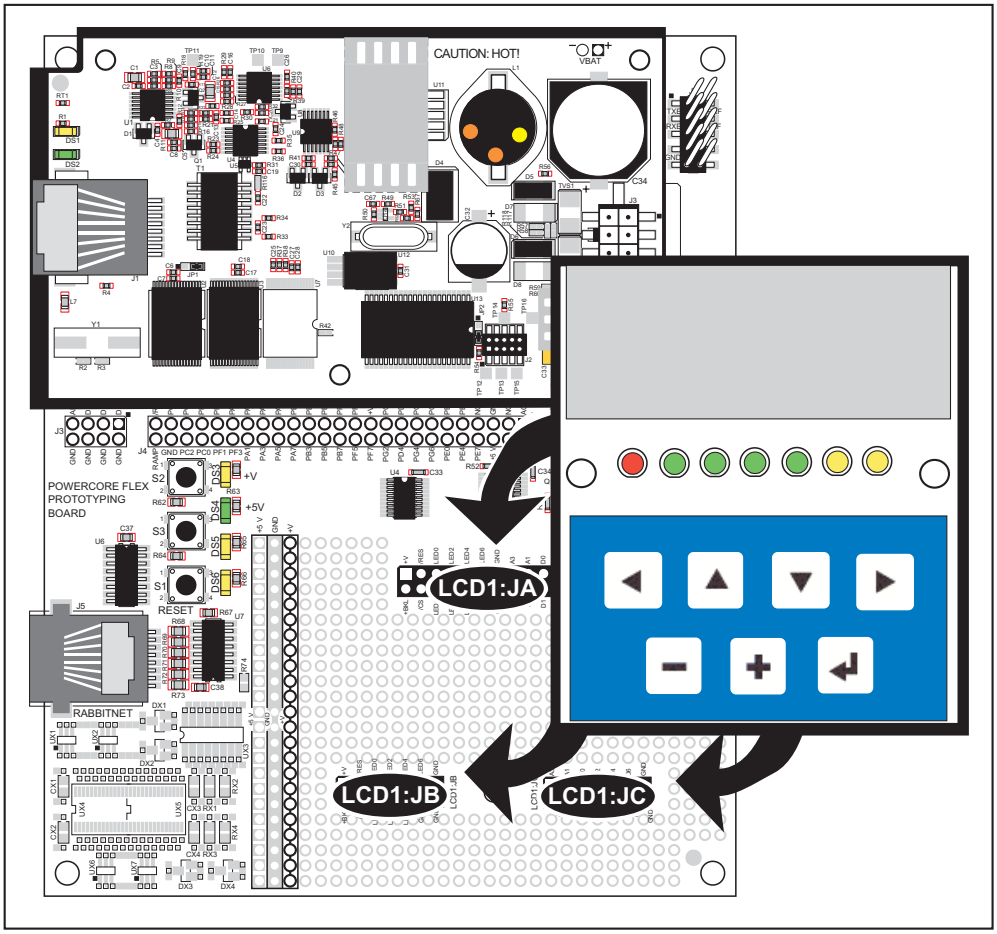


Figure C-8. Install LCD/Keypad Module on Prototyping Board

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.

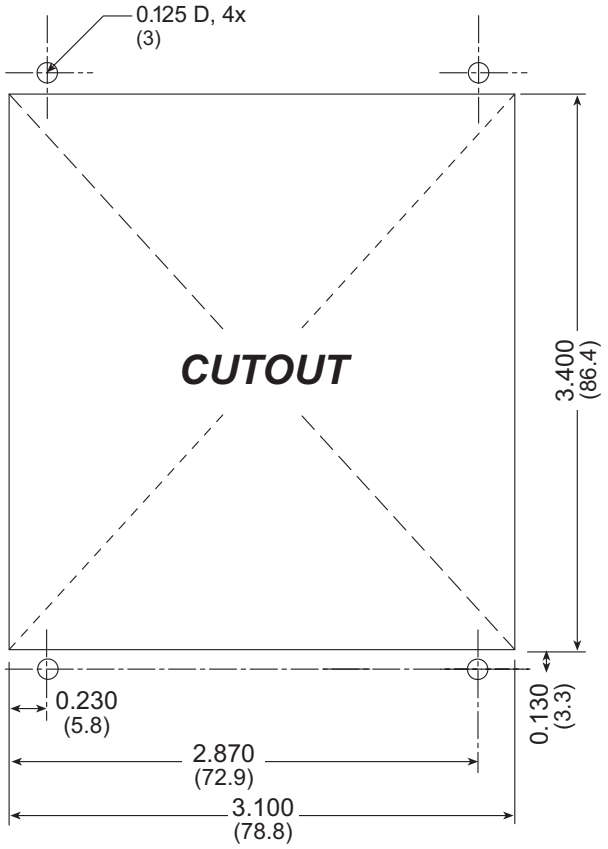


Figure C-9. Recommended Cutout Dimensions

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.

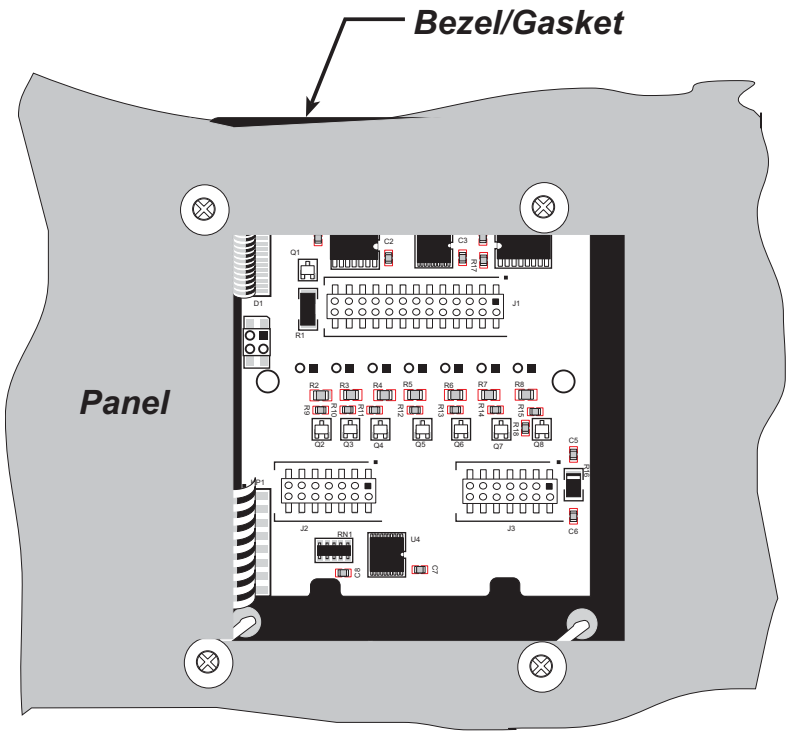


Figure C-10. LCD/Keypad Module Mounted in Panel (rear view)

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

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

C.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 PowerCore Prototyping Board, and is connected via a ribbon cable as shown in Figure C-11.

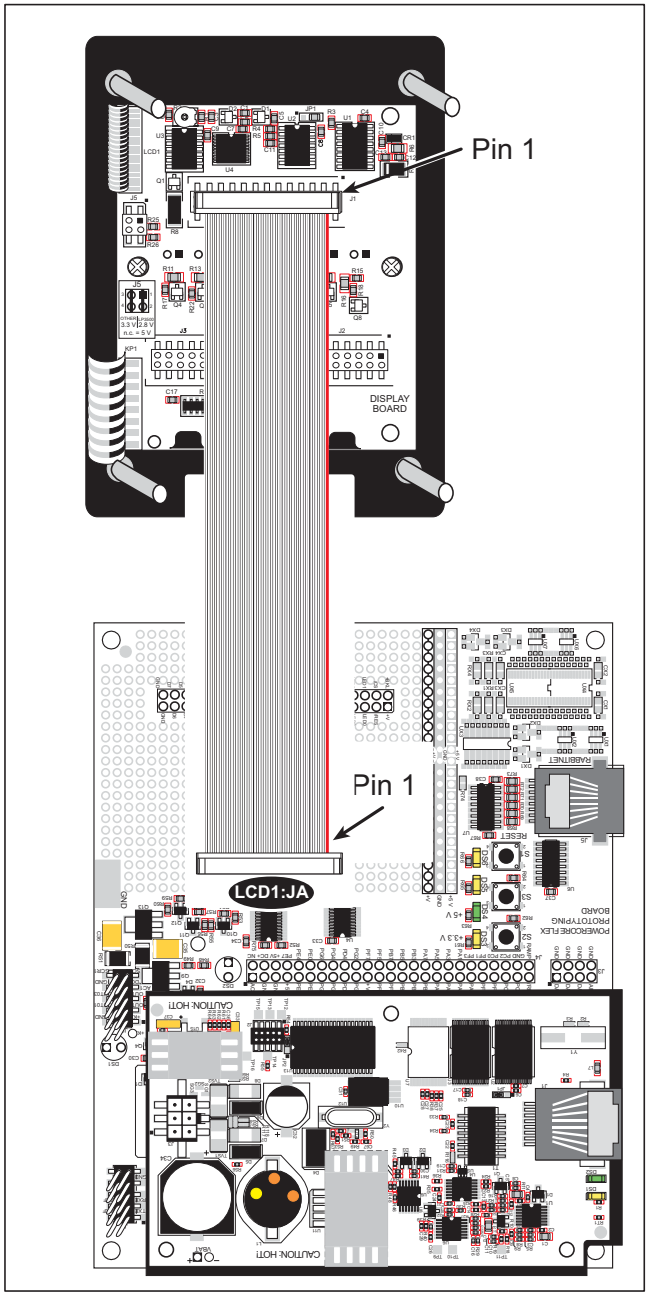


Figure C-11. Connecting LCD/Keypad Module to PowerCore Prototyping Board

Note the locations and connections relative to pin 1 on both the 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\PowerCoreFLEX\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), compile it using the **Compile** menu, and then run it by selecting **Run** in the **Run** menu. The PowerCore module must be in **Program** mode (see Section 4.3, "Programming Cable"), and must be connected to a PC using the programming cable as described in Chapter 2, "Getting Started."

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

- **KEYPAD_LED.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 keypress is detected.
- **LCDKEY_FUN.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.

  Once you have compiled the program and are running it, you can watch the LCD display as it goes through the various graphic demonstrations. Press a key on the LCD/keypad module to see the corresponding LED light up. Press S2 on the Prototyping Board to light up LED DS5 (LED0) on the Prototyping Board, or press S3 on the Prototyping Board to light up LED DS6 (LED1) on the Prototyping Board.
- **SWITCH_LCD.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. LEDs DS5 or DS6 on the Prototyping Board will also light up when switch S2 or S3 is pressed.

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

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.

---

dispInit

```
void dispInit();
```

DESCRIPTION

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.

---

dispLedOut

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

DESCRIPTION

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.

glInit

```
void glInit(void);
```

DESCRIPTION

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

glBackLight

```
void glBackLight(int onOff);
```

DESCRIPTION

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

glDispOnOff

```
void glDispOnOff(int onOff);
```

DESCRIPTION

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`

glSetContrast

```
void glSetContrast(unsigned level);
```

DESCRIPTION

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.

glFillScreen

```
void glFillScreen(int pattern);
```

DESCRIPTION

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

glBlankScreen

```
void glBlankScreen(void);
```

DESCRIPTION

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

RETURN VALUE

None.

SEE ALSO

glFillScreen, glBlock, glPlotPolygon, glPlotCircle

glFillRegion

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

DESCRIPTION

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	the *x* coordinate of the top left corner of the block.
top	the *y* coordinate of the top left corner of the block.
width	the width of the block.
height	the height of the block.
pattern	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

glFastFillRegion

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

DESCRIPTION

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	the *x* coordinate of the top left corner of the block.
top	the *y* coordinate of the top left corner of the block.
width	the width of the block.
height	the height of the block.
pattern	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

glBlankRegion

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

DESCRIPTION

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

RETURN VALUE

None.

SEE ALSO

glFillScreen, glBlankScreen, glBlock

glBlock

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

DESCRIPTION

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	the *x* coordinate of the top left corner of the block.
top	the *y* coordinate of the top left corner of the block.
width	the width of the block.
height	the height of the block.

RETURN VALUE

None.

SEE ALSO

glFillScreen, glBlankScreen, glPlotPolygon, glPlotCircle

glPlotVPolygon

```
void glPlotVPolygon(int n, int *pFirstCoord);
```

DESCRIPTION

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

RETURN VALUE

None.

SEE ALSO

glPlotPolygon, glFillPolygon, glFillVPolygon

glPlotPolygon

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

DESCRIPTION

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	the number of vertices.
y1	the *y* coordinate of the first vertex.
x1	the *x* coordinate of the first vertex.
y2	the *y* coordinate of the second vertex.
x2	the *x* coordinate of the second vertex.
...	the coordinates of additional vertices.

RETURN VALUE

None.

SEE ALSO

glPlotVPolygon, glFillPolygon, glFillVPolygon

glFillVPolygon

```
void glFillVPolygon(int n, int *pFirstCoord);
```

DESCRIPTION

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

RETURN VALUE

None.

SEE ALSO

glFillPolygon, glPlotPolygon, glPlotVPolygon

glFillPolygon

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

DESCRIPTION

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	the number of vertices.
x1	the *x* coordinate of the first vertex.
y1	the *y* coordinate of the first vertex.
x2	the *x* coordinate of the second vertex.
y2	the *y* coordinate of the second vertex.
...	the coordinates of additional vertices.

RETURN VALUE

None.

SEE ALSO

glFillVPolygon, glPlotPolygon, glPlotVPolygon

glPlotCircle

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

DESCRIPTION

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	the *x* coordinate of the center of the circle.
yc	the *y* coordinate of the center of the circle.
rad	the radius of the center of the circle (in pixels).

RETURN VALUE

None.

SEE ALSO

glFillCircle, glPlotPolygon, glFillPolygon

glFillCircle

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

DESCRIPTION

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	the *x* coordinate of the center of the circle.
yc	the *y* coordinate of the center of the circle.
rad	the radius of the center of the circle (in pixels).

RETURN VALUE

None.

SEE ALSO

glPlotCircle, glPlotPolygon, glFillPolygon

glXFontInit

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

DESCRIPTION

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	a pointer to the font descriptor to be initialized.
pixWidth	the width (in pixels) of each font item.
pixHeight	the height (in pixels) of each font item.
startChar	the value of the first printable character in the font character set.
endChar	the value of the last printable character in the font character set.
xmemBuffer	the **xmem** pointer to a linear array of font bitmaps.

RETURN VALUE

None.

SEE ALSO

glPrinf

glFontCharAddr

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

DESCRIPTION

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

PARAMETERS

***pInfo**	the **xmem** address of the bitmap font set.
letter	an ASCII character.

RETURN VALUE

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

SEE ALSO

glPutFont, glPrintf

glPutFont

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

DESCRIPTION

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	the *x* coordinate (column) of the top left corner of the text.
y	the *y* coordinate (row) of the top left corner of the text.
pInfo	a pointer to the font descriptor.
code	the ASCII character to display.

RETURN VALUE

None.

SEE ALSO

glFontCharAddr, glPrintf

glSetPfStep

```
void glSetPfStep(int stepX, int stepY);
```

DESCRIPTION

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	the **glPrintf** *x* step value
stepY	the **glPrintf** *y* step value

RETURN VALUE

None.

SEE ALSO

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

glGetPfStep

```
int glGetPfStep(void);
```

DESCRIPTION

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.

glPutChar

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

DESCRIPTION

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	the character to be displayed on the LCD.
***ptr**	not used, but is a place holder for **STDIO** string functions.
***cnt**	not used, is a place holder for **STDIO** string functions.
pInst	a pointer to the font descriptor.

RETURN VALUE

None.

SEE ALSO

glPrintf, glPutFont, doprnt

glPrintf

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

DESCRIPTION

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	the *x* coordinate (column) of the upper left corner of the text.
y	the *y* coordinate (row) of the upper left corner of the text.
pInfo	a pointer to the font descriptor.
***fmt**	a formatted string.
...	formatted string conversion parameter(s).

EXAMPLE

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

RETURN VALUE

None.

SEE ALSO

glXFontInit

glBuffLock

```
void glBuffLock(void);
```

DESCRIPTION

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`

glBuffUnlock

```
void glBuffUnlock(void);
```

DESCRIPTION

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`

glSwap

```
void glSwap(void);
```

DESCRIPTION

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)

glSetBrushType

```
void glSetBrushType(int type);
```

DESCRIPTION

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

glGetBrushType

```
int glGetBrushType(void);
```

DESCRIPTION

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

RETURN VALUE

The current brush type.

SEE ALSO

glSetBrushType

glXGetBitmap

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

DESCRIPTION

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	the *x* coordinate in pixels of the top left corner of the bitmap (*x* must be evenly divisible by 8).
y	the *y* coordinate in pixels of the top left corner of the bitmap.
bmWidth	the width in pixels of the bitmap (must be evenly divisible by 8).
bmHeight	the height in pixels of the bitmap.
xBm	the **xmem** RAM storage address of the bitmap.

RETURN VALUE

None.

glXGetFastmap

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

DESCRIPTION

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	the *x* coordinate of the top left corner of the bitmap (*x* must be evenly divisible by 8).
top	the *y* coordinate in pixels of the top left corner of the bitmap.
width	the width of the bitmap (must be evenly divisible by 8).
height	the height of the bitmap.
xmemptr	the **xmem** RAM storage address of the bitmap.

RETURN VALUE

None.

SEE ALSO

glXPutBitmap, glPrintf

glPlotDot

```
void glPlotDot(int x, int y);
```

DESCRIPTION

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	the *x* coordinate of the dot.
y	the *y* coordinate of the dot.

RETURN VALUE

None.

SEE ALSO

glPlotline, glPlotPolygon, glPlotCircle

glPlotLine

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

DESCRIPTION

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	the *x* coordinate of one endpoint of the line.
y0	the *y* coordinate of one endpoint of the line.
x1	the *x* coordinate of the other endpoint of the line.
y1	the *y* coordinate of the other endpoint of the line.

RETURN VALUE

None.

SEE ALSO

glPlotDot, glPlotPolygon, glPlotCircle

glLeft1

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

DESCRIPTION

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

PARAMETERS

left	the top left corner of bitmap, must be evenly divisible by 8, otherwise truncates.
top	the top left corner of the bitmap.
cols	the number of columns in the window, must be evenly divisible by 8, otherwise truncates.
rows	the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glHScroll, glRight1

glRight1

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

DESCRIPTION

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

PARAMETERS

left	the top left corner of bitmap, must be evenly divisible by 8, otherwise truncates.
top	the top left corner of the bitmap.
cols	the number of columns in the window, must be evenly divisible by 8, otherwise truncates.
rows	the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glHScroll, glLeft1

glUp1

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

DESCRIPTION

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

PARAMETERS

left	the top left corner of bitmap, must be evenly divisible by 8, otherwise truncates.
top	the top left corner of the bitmap.
cols	the number of columns in the window, must be evenly divisible by 8, otherwise truncates.
rows	the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glVScroll, glDown1

glDown1

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

DESCRIPTION

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

PARAMETERS

left	the top left corner of bitmap, must be evenly divisible by 8, otherwise truncates.
top	the top left corner of the bitmap.
cols	the number of columns in the window, must be evenly divisible by 8, otherwise truncates.
rows	the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glVScroll, glUp1

glHScroll

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

DESCRIPTION

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	the top left corner of bitmap, must be evenly divisible by 8.
top	the top left corner of the bitmap.
cols	the number of columns in the window, must be evenly divisible by 8.
rows	the number of rows in the window.
nPix	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

glVScroll

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

DESCRIPTION

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	the top left corner of bitmap, must be evenly divisible by 8.
top	the top left corner of the bitmap.
cols	the number of columns in the window, must be evenly divisible by 8.
rows	the number of rows in the window.
nPix	the number of pixels to scroll within the defined window (a negative value will produce a scroll up).

RETURN VALUE

None.

SEE ALSO

glHScroll

glXPutBitmap

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

DESCRIPTION

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	the top left corner of the bitmap.
top	the top left corner of the bitmap.
width	the width of the bitmap.
height	the height of the bitmap.
bitmap	the address of the bitmap in **xmem**.

RETURN VALUE

None.

SEE ALSO

glXPutFastmap, glPrintf

glXPutFastmap

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

DESCRIPTION

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	the top left corner of the bitmap, must be evenly divisible by 8, otherwise truncates.
top	the top left corner of the bitmap.
width	the width of the bitmap, must be evenly divisible by 8, otherwise truncates.
height	the height of the bitmap.
bitmap	the address of the bitmap in **xmem**.

RETURN VALUE

None.

SEE ALSO

glXPutBitmap, glPrintf

TextWindowFrame

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

DESCRIPTION

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`	a pointer to the window frame descriptor.
`pFont`	a pointer to the font descriptor.
`x`	the *x* coordinate of the top left corner of the text window frame.
`y`	the *y* coordinate of the top left corner of the text window frame.
`winWidth`	the width of the text window frame.
`winHeight`	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.

TextBorderInit

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

DESCRIPTION

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

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

PARAMETERS

wPtr a pointer to the window frame descriptor.

border 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 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

TextBorder

```
void TextBorder(windowFrame *wPtr);
```

DESCRIPTION

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.

PARAMETER

wPtr a pointer to the window frame descriptor.

RETURN VALUE

None.

SEE ALSO

TextBorderInit, TextGotoXY, TextPutChar, TextWindowFrame, TextCursorLocation

TextGotoXY

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

DESCRIPTION

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 a pointer to a font descriptor.

col a character column location.

row a character row location.

RETURN VALUE

None.

SEE ALSO

TextPutChar, TextPrintf, TextWindowFrame

TextCursorLocation

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

DESCRIPTION

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

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

PARAMETERS

`window`	a pointer to a font descriptor.
`col`	a pointer to cursor column variable.
`row`	a pointer to cursor row variable.

RETURN VALUE

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

SEE ALSO

`TextGotoXY, TextPrintf, TextWindowFrame, TextCursorLocation`

TextPutChar

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

DESCRIPTION

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`	a pointer to a font descriptor.
`ch`	a character to be displayed on the LCD.

RETURN VALUE

None.

SEE ALSO

`TextGotoXY, TextPrintf, TextWindowFrame, TextCursorLocation`

TextPrintf

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

DESCRIPTION

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 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	a pointer to a font descriptor.
***fmt**	a formatted string.
...	formatted string conversion parameter(s).

EXAMPLE

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

RETURN VALUE

None.

SEE ALSO

TextGotoXY, TextPutChar, TextWindowFrame, TextCursorLocation

TextMaxChars

```
int TextMaxChars(windowFrame *wPtr);
```

DESCRIPTION

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.

PARAMETER

wPtr 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

TextWinClear

```
void TextWinClear(windowFrame *wPtr);
```

DESCRIPTION

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

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

PARAMETERS

wPtr 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\KEYPADS\KEYPAD7.LIB** library.

---

keyInit

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

DESCRIPTION

Initializes keypad process.

RETURN VALUE

None.

SEE ALSO

brdInit

keyConfig

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

DESCRIPTION

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

PARAMETERS

cRaw a raw key code index.

1 × 7 keypad matrix with raw key code index assignments (in brackets):

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

User Keypad Interface

cPress a keypress code

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

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

cRelease a key release code.

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

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

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

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

How many times to repeat.
0 = None.

cCntLo 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 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

keyProcess

```
void keyProcess(void);
```

DESCRIPTION

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

keyGet

```
char keyGet(void);
```

DESCRIPTION

Get next keypress.

RETURN VALUE

The next keypress, or 0 if none.

SEE ALSO

keyConfig, keyProcess, keypadDef

keyUnget

```
int keyUnget(char cKey);
```

DESCRIPTION

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

keypadDef

```
void keypadDef();
```

DESCRIPTION

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

1 × 7 keypad physical mapping:

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

where
'L' represents Left Scroll
'U' represents Up Scroll
'D' represents Down Scroll
'R' represents Right 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

keyScan

```
void keyScan(char *pcKeys);
```

DESCRIPTION

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 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 for PowerCore power supplies and battery backup. Information on the reset generator used in power management is also included.

D.1 Power Supplies

The PowerCore has an optional 6-pin locking connector on the top side at J3 designed to accept a wire harness that brings in power. The same power connections and some additional connections are available on the 50-pin motherboard connector J4.

The PowerCore receives power in the form of unregulated AC, DC, or regulated +5 V DC. When an unregulated power supply is used, the PowerCore has an onboard +5 V DC switching regulator that is available in 1 A and 2 A versions. The +5 V DC is regulated down to 3.45 V by a linear regulator on the PowerCore board.

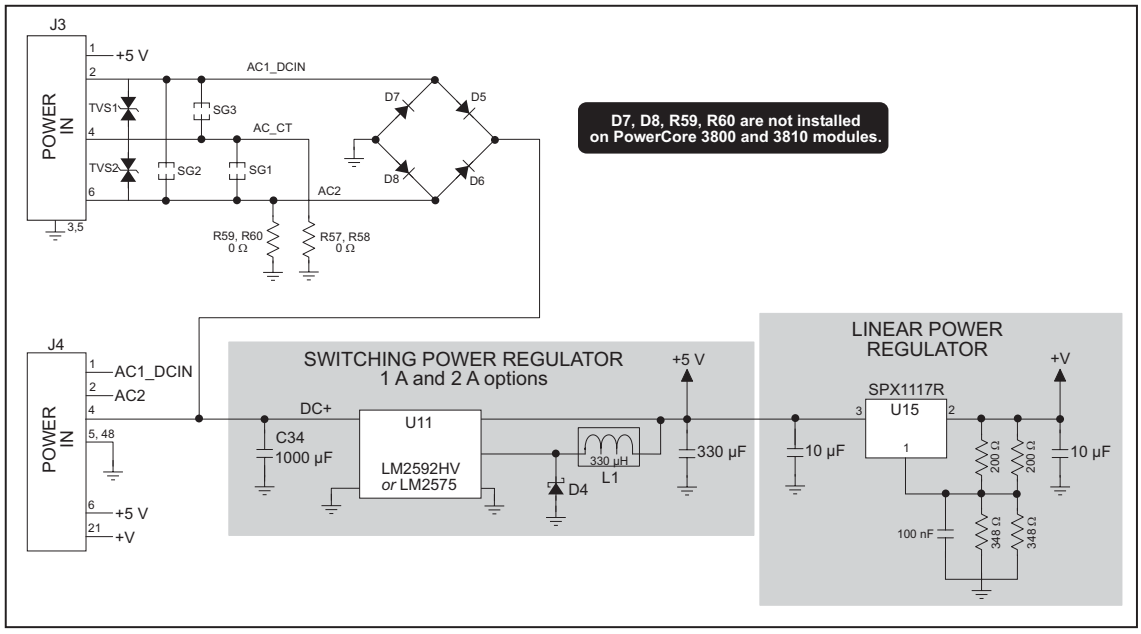


Figure D-1. PowerCore Module Power Supplies

The two preconfigured PowerCore models are built for use with a full-wave rectifier and center-tapped AC transformer. Power may be supplied directly to the PowerCore module via the locking connector at header J3:

- via pins 2, 4, and 6 (AC); note that center tap of the transformer is connected to ground through pin 4 and resistors R57 and R58
- via pins 2 and 3 (DC, unregulated)

Power may also be supplied to the PowerCore module from the motherboard to which the PowerCore module is plugged in:

- via pins 1 and 2 of header J4 (AC); note that center tap of the transformer should be connected to ground
- via pins 4 and 5 of header J4 (DC, unregulated)

The PowerCore module also provides for the regulated voltages to be output for use on the motherboard or elsewhere:

- +5 V DC on pin 6, header J4 (connection to motherboard), and pin 1, header J3 (locking connector)
- +3.45 V DC (+V) on pin 21, header J4 (connection to motherboard)

You can draw up to 550 mA from the +3.45 V supply, depending on the PowerCore options and the operating temperature. If there is a +5 V regulator on the PowerCore board, you can draw as much as 850 mA or 1850 mA from the +5 V supply, depending on what options are actually installed on the PowerCore module, the input voltage to the switching regulator, and the operating temperature. The current from the various power supplies that is available for use on the motherboard or elsewhere is specified in Table A-1.

There is an additional design consideration for the switching power supply with regard to AC ripple at 50 kHz or 150 kHz. The ripple can be reduced by adding additional filtering capacitors on the motherboard or by using local filters where reduced ripple is required. Ripple increases as more current is drawn from the power supply.

Ripple in the rectified AC can also be a particular concern for the half-wave rectifier option. This ripple can be reduced by adding additional capacitance to the unregulated DC between the DC+ line (from pin 4 of J4) and ground on the motherboard.

The operating life of electrolytic filter capacitors is reduced at higher temperatures and with more ripple. For this reason it is important to use high-quality capacitors and to follow standard engineering guidelines for ripple at higher operating temperatures to extend capacitor life and to avoid premature failures.

If you plan to operate your PowerCore module from a low-voltage AC input ***and*** draw significant amounts of current from the PowerCore for external circuits ***and*** operate your PowerCore module at very high ambient temperatures, adding an external 1000 µF capacitor to the motherboard can double the lifetime of the AC rectifier filter circuit.

Use these guidelines to define the thresholds when each element needs to be considered.

- Voltage type—AC only.
- Voltage threshold—AC input voltages within 3 V of the minimum specified input voltage for Options 3 or 4, within 5 V of the minimum specified input voltage for Option 5.
- Significant current

  more than 1 A to user circuits for 2 A regulator
  more than 250 mA to user circuits for 1 A regulator
- High ambient temperatures—If all three of the above factors are present, then look at Table D-1 to determine whether the operating temperature creates an issue. The capacitor lifetimes are listed at the specified temperatures for the existing 1000 µF capacitor at position C34 on the PowerCore module by itself and with an additional user-supplied 1000 µF capacitor installed between the DC+ line (from pin 4 of J4) and ground on the motherboard:

Table D-1. Effects of Additional 1000 µF Filtering Capacitor

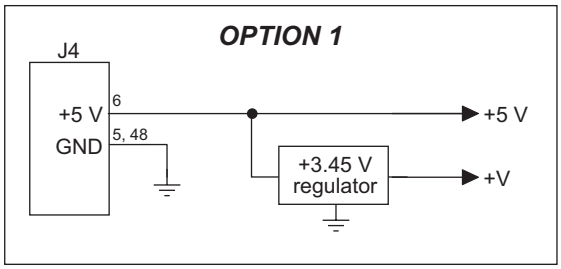
Ambient Temperature)	Lifetime with One 1000 µF Capacitor on PowerCore Module	Lifetime with Additional 1000 µF Capacitor on Motherboard
25°C	Capacitor lifetime not a factor	
40°C	Capacitor lifetime not a factor	
50°C	18 years	36 years
60°C	9 years	18 years
70°C	4.5 years	9 years

D.1.1 Power-Supply Options

Five power-supply options are available for the PowerCore modules. In two of the options, either regulated or unregulated DC power comes to header J4 on the PowerCore from the user's motherboard. In the three remaining options, AC power comes either to header J4 on the PowerCore from the user's motherboard, or the AC power comes directly to the PowerCore module via the polarized locking connector at J3.

Option 1—Regulated +5 V DC

Regulated +5 V (±5%) DC power must be supplied to the PowerCore module via pins 5 and 6 of header J4 (the locking connector at J3 is not stuffed when you select this option). There is always a +3.45 V DC linear voltage regulator on PowerCore modules to supply +3.45 V DC.

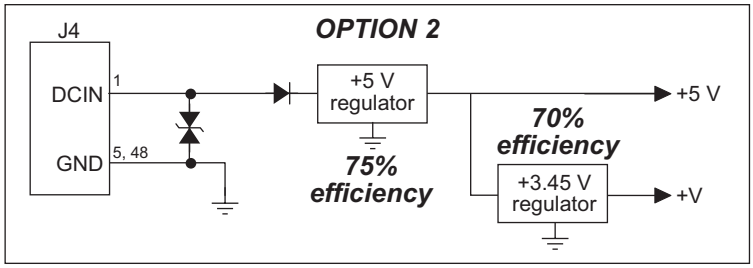


With this option, the maximum power is limited by the traces on the PowerCore module to 2 A at 5 V, which includes the power consumed by the +3.45 V and +5 V onboard PowerCore circuits. The 3.45 V regulator can pass a maximum of 700 mA. The PowerCore will require 150–400 mA at 3.45 V depending on the clock speed and other options.

This power-supply option does not have the AC zero-crossover detection circuit used for triac support.

Option 2—Unregulated DC

Unregulated DC power is supplied to the PowerCore module via pins 1 and 5 of header J4 (the locking connector at J3 is not stuffed when you select this option).



The selection of the +5 V regulator determines the voltage range for the unregulated DC input power as follows.

- 8–43 V DC for 2 A regulator option
- 9–40 V DC for 1 A regulator option

The power consumed by the onboard PowerCore circuits is 150–400 mA at +3.45 V depending on the clock speed and other options. This power-supply option does not support the AC zero-crossover detection circuit.

You can calculate the current that is available to your circuits that are external to the PowerCore module.

The current required is calculated based on the power consumed by the PowerCore as optionally configured. I = Power/Vin. The PowerCore will consume between 1 W and 2.7 W, depending on the PowerCore options. Additional power may be needed to provide for other user's circuitry external to the PowerCore. That additional power needed from the input power supply is calculated as follows.

- For the user's 5 V circuitry, additional power from supply = (5Vpower_to_user's_circuit) × 1.33.
- For the user's 3.45 V circuitry, additional power from supply = (3.45Vpower_to_user's_circuit) × 1.92.

Thus, as an example, if the user's 5 V circuitry consumes 2.5 W and the user's 3.45 V circuitry consumes 0.5 W, then the total power needed from the power supply is (2.5 W × 1.33 + 0.5 W × 1.92 + (PowerCore consumption as optionally configured)) = 5.3 W to 7.0 W. If that external power supply is a 12 V supply, then it will need to source I = Power/Voltage current. Bear in mind that voltage presented to the regulators has a 0.7 V diode drop. Thus the voltage in this formula must be reduced by 0.7 V, so current = Power/11.3 V amps current. 5.3 W/11.3 V = 469 mA; 7.0 W/11.3 V = 619 mA.

This power-supply option does not have the AC zero-crossover detection circuit used for triac support.

External AC Power Supplies

Three external AC power-supply options are available.

- Option 3—AC via locking connector at J3, full-wave bridge rectifier, onboard +5 V switching regulator @ 1 A or 2 A, no zero-crossover detection or triac support, transformer with untapped secondary winding may be used
- Option 4—AC via locking connector at J3, full-wave center-tapped rectifier, onboard +5 V switching regulator @ 1 A or 2 A, includes zero-crossover detection for triac support, transformer with tapped secondary winding required
- Option 5—AC via locking connector at J3, half-wave rectifier, onboard +5 V switching regulator @ 1 A or 2 A, includes zero-crossover detection for triac support, transformer with untapped secondary winding may be used

The operation and features of each these options are discussed below.

Option 3—Full-Wave Bridge

Option 3 requires unregulated AC or DC. This option places a full-wave bridge and filter before the regulators. The full wave bridge is necessary to rectify AC voltage into DC voltage. This option has a friction lock connector installed on the PowerCore to accept external power.

The full-wave bridge causes a 1.4 V voltage drop before voltage is applied to the regulators. This means that the 1.4 V drop needs to be taken into account.

The selection of the +5 V regulator determines the voltage range for the unregulated DC input power as follows.

- 9–51 V DC for 2 A regulator option
- 10–41 V DC for 1 A regulator option

With this option, unregulated DC voltage is input to pin 2 of locking connector J3. The negative side of the input voltage is applied to pin 6 of locking connector J3.

The unregulated DC connection allows a higher voltage to be applied with the 2 A regulator option, but it does not provide for a common power-supply ground and circuit ground. If the DC voltage does not exceed 43 V with a 2 A regulator, or 40 V with a 1 A regulator, then the DC power-input ground can be connected to both J3 pins 3 and 4— this will provide for a common power-supply and circuit ground, and will also provide 43 V transorb protection from transients.

AC voltage is applied to the same pins (between pins 2 and 6 of J3). Because the peak AC voltages are 1.42 times higher than the rated AC voltages, the AC voltage applied will have lower upper limits than those specified for the DC input. Also, because AC voltage inputs go down to zero voltage during each cycle, a higher minimum voltage than the DC voltage will be required. The selection of the +5 V regulator determines the voltage range for the AC input power as follows.

- 12–36 V AC for 2 A regulator option
- 10–29 V AC for 1 A regulator option

The current required is calculated based on the power consumed by the PowerCore as optionally configured: I = Power/Vin. The PowerCore will consume between 1 W and 2.7 W, depending on the PowerCore options. Additional power may be needed to provide for other user's circuitry external to the PowerCore. That additional power needed from the input power supply is calculated as follows.

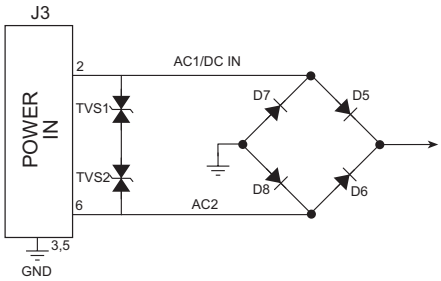
- For the user's 5 V circuitry, additional power from supply = (5Vpower_to_user's_circuit) × 1.33.
- For the user's 3.45 V circuitry, additional power from supply = (3.45Vpower_to_user's_circuit) × 1.92.

Additional power is taken by the input diodes.

Thus, as an example, if the user's 5 V circuitry consumes 2.5 W and the user's 3.45 V circuitry consumes 0.5 W, the total power needed from the power supply is (2.5 W × 1.33 + 0.5 W × 1.92 + (PowerCore consumption as optionally configured)) = 5.3 W to 7.0 W. If the external power supply is a 12 V supply, then it will need to source I = Power/Voltage current. Bear in mind that voltage presented to the regulators has a 1.4 V diode drop. Thus the voltage in this formula must be reduced by 1.4 V, so current = Power/10.6 V = 5.3 W/10.6 V = 500 mA; 7.0 W/10.6 V = 660 mA. This is the DC power supply current requirement, and would be the maximum current that an AC power-supply would have to deliver.

The input power circuit configuration is shown below for Option 3.

OPTION 3 FULL-WAVE BRIDGE

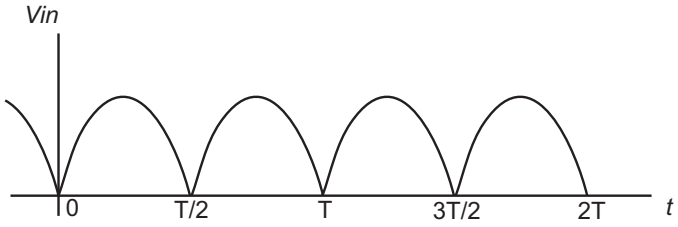


	2 A CONFIGURATION	1 A CONFIGURATION
Input Supply Requirements	12–36 V AC 9–51 V DC	10–29 V AC 10–41 V DC
Current Draw by Onboard Circuits	max I_Q = 150–400 mA @ +3.45 V	max I_Q = 150–400 mA @ +3.45 V
Output Current Available	700 mA - I_Q @ +3.45 V 2 A - I_Q - I_{3VMB} @ +5 V	700 mA - I_Q @ +3.45 V 1 A - I_Q - I_{3VMB} @ +5 V

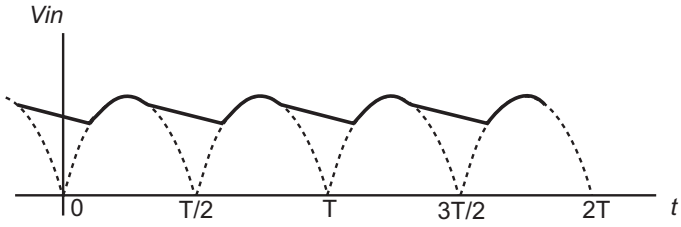
NOTE: I_{3VMB} = current consumed by user's board at +3.45 V

The full-wave bridge rectifier has no common ground between the power-supply transformer and the rest of the circuit. The full-wave bridge requires four diodes and therefore is a more expensive circuit than a full-wave center-tapped rectifier or a half wave rectifier.

The output from this full-wave bridge is shown at right.



After the bridge, the AC voltage is filtered with a large (1000 µF) capacitor. The final filtered voltage is shown by the solid line in the diagram at right.



For any current to flow, two of the diodes must be turned on at any given time, depending on the polarity of the AC sine wave at the transformer secondary winding. Because two diodes conduct at any given time, there are two diode voltage drops for the complete current to flow through. This means that the power dissipated by the full-wave bridge is 1.4 V × current, which is twice that of the full-wave center-tapped rectifier.

Since current is delivered to the load during both halves of the cycle and since there is no need for identical secondary windings on both sides of a center tap, the full-wave bridge requires the least expensive transformer. However, the full-wave bridge has twice the power dissipation of the full-wave center-tapped rectifier. The full-wave bridge cannot be used with triacs and other circuits based on zero-crossover detection because there is no common ground.

Option 4—Full-Wave CT

Option 4 requires unregulated AC or DC. This option uses a center-tapped-transformer full-wave bridge and filter before the regulators. The full-wave bridge converts AC voltage into DC voltage. This option has a friction lock connector installed on the PowerCore to accept external power. The full-wave rectifier of option 4 requires a center-tapped transformer—a center-tapped transformer has a third connection at the center of the secondary side for a connection to ground.

The full-wave rectifier causes a 0.7 V voltage drop before voltage is applied to the regulators. This means that the 0.7 V drop will need to be taken into account.

With this option, it is also possible to supply unregulated DC voltage to pin 2 of locking connector J3. The negative side of the input voltage is applied to pin 4 of locking connector J3.

The selection of the +5 V regulator determines the voltage range for the unregulated DC input power as follows.

- 8–43 V DC for 2 A regulator option
- 9–40 V DC for 1 A regulator option

AC voltage is applied to pins 2 and 6 of J3 with the center tap of the transformer connected to pin 4 of J3. Because the peak AC voltages are 1.42 times higher than the rated AC voltages, the AC voltage applied will have lower upper limits than that of an unregulated DC input. Also, because AC voltage inputs go down to zero voltage during each cycle, a higher minimum voltage than the DC voltage will be required. The selection of the +5 V regulator determines the voltage range for the AC input power as follows.

- 24–60 V AC for 2 A regulator option
- 19–57 V AC for 1 A regulator option

This is the voltage range between each AC output. The AC voltage to the center tap is half of the voltage specified above.

The current required is calculated based on the power consumed by the PowerCore as optionally configured: I = Power/Vin. The PowerCore will consume between 1 W and 2.7 W, depending on the PowerCore options. Additional power may be needed to provide for other user's circuitry external to the PowerCore. That additional power needed from the input power supply is calculated as follows.

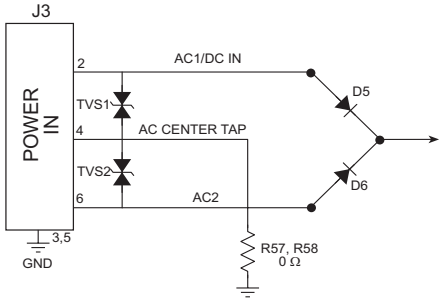
- For the user's 5 V circuitry, additional power from supply = (5Vpower_to_user's_circuit) × 1.33.
- For the user's 3.45 V circuitry, additional power from supply = (3.45Vpower_to_user's_circuit) × 1.92.

Additional power is taken by the input diodes.

Thus, as an example, if the user's 5 V circuitry consumes 2.5 W and the user's 3.45 V circuitry consumes 0.5 W, then total power needed from the power supply is (2.5 W × 1.33 + 0.5 W × 1.92 + (PowerCore consumption as optionally configured)) = 5.3 W to 7.0 W. If that external power supply is a 24 V center-tapped AC supply (giving 12 V on each side of the center tap), then it will need to source I = (Power/Voltage) current. Bear in mind that voltage presented to the regulators has a 0.7 V diode drop. Thus the voltage in this formula must be reduced by 0.7 V, so current = Power/11.3 V current (in amps). 5.3 W /11.3 V = 469 mA; 7.0 W/11.3 V = 619 mA. This is the maximum current the AC power supply will need to deliver. An AC supply may not need to deliver that much current since the filter capacitor may raise the DC voltage input into the regulator, allowing less current to be consumed. If a DC supply were used, this is the current the power supply would have to deliver.

The input power circuit configuration is shown below for Option 4.

OPTION 4 FULL-WAVE CENTER-TAPPED

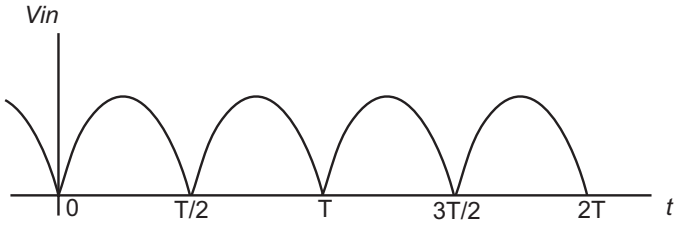


	2 A CONFIGURATION	1 A CONFIGURATION
Input Supply Requirements	24–60 V AC 8–43 V DC	19–57 V AC 9–40 V DC
Current Draw by Onboard Circuits	max I_Q = 150–400 mA @ +3.45 V	max I_Q = 150–400 mA @ +3.45 V
Output Current Available	700 mA - I_Q @ +3.45 V 2 A - I_Q - I_{3VMB} @ +5 V	700 mA - I_Q @ +3.45 V 1 A - I_Q - I_{3VMB} @ +5 V

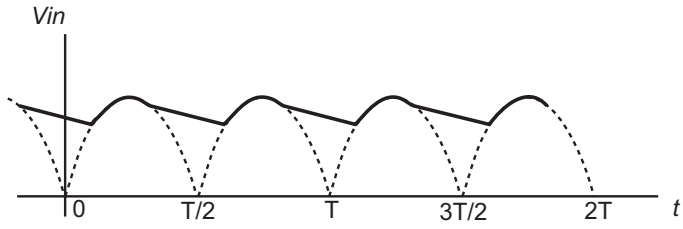
NOTE: I_{3VMB} = current consumed by user's board at +3.45 V

A full-wave center-tapped rectifier has a third connection on the secondary side of the AC transformer for ground. Both sides of the windings go separately through diodes as shown above. This is essentially two half-wave rectifier circuits with one inverted from the other. This configuration combines the advantages of full-wave and half-wave rectifiers.

The output from this full-wave center-tapped rectifier is shown at right.



After the rectifier, the AC voltage is filtered with a large (1000 µF) capacitor. The final filtered voltage is shown by the solid line in the diagram at right.



Since at any given time there is only current flowing in one of the diodes, there is only one diode voltage drop at a given time in the circuit. Because current flows during both half-cycles, only half the current required by an equivalent half-wave rectifier will be needed. The power dissipated by the full-wave center-tapped rectifier is 0.7 V × current. Since this rectifier circuit has only two diodes, it is less expensive than a full-wave bridge. Since the power-supply center tap is ground, there is a common ground between the power supply and the circuit. However, each of the secondary windings of the transformer must still pass the full current, and this along with the third center-tap connection could increase the physical size and cost of the transformer.

Because there is a common AC ground between the AC power transformer and the circuit, AC zero-crossing detection is possible. Zero-crossing detection allows accurate triac timing control. Thus, the zero-crossing detection circuit is stuffed and triac control is supported with this option.

Option 5—Half-Wave Rectifier

Option 5 requires unregulated AC or DC. This option places a half-wave rectifier and filter before the regulators. The half-wave rectifier is necessary to rectify AC voltage into DC voltage. This option has a friction lock connector installed on the PowerCore to accept external power.

The half-wave rectifier causes a 0.7 V voltage drop before voltage is applied to the regulators. This means that the 0.7 V drop will need to be taken into account.

With this option, unregulated DC voltage is input to pin 2 of friction lock connector J3. The negative side of the input voltage is applied to pin 4 and/or 6 of friction lock connector J3. The selection of the +5 V regulator determines the voltage range for the unregulated DC input power as follows.

- 8–43 V DC for 2 A regulator option
- 9–40 V DC for 1 A regulator option

AC voltage is applied to the same pins of J3. Because the peak AC voltages are 1.42 times higher than the rated AC voltages, the AC voltage applied will have lower upper limits. Also, because AC voltage inputs go down to zero voltage during each cycle, a higher minimum voltage than the DC voltage will be required. The selection of the +5 V regulator determines the voltage range for the AC input power as follows.

- 17–30 V AC for 2 A regulator option
- 14–28 V AC for 1 A regulator option

The current required is calculated based on the power consumed by the PowerCore as optionally configured: I = Power/Vin. The PowerCore will consume between 1 W and 2.7 W, depending on the PowerCore options. Additional power may be needed to provide for other user's circuitry external to the PowerCore. That additional power needed from the input power supply is calculated as follows.

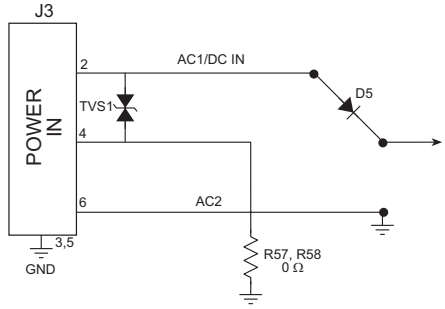
- For the user's 5 V circuitry, additional power from supply = (5Vpower_to_user's_circuit) × 1.33.
- For the user's 3.45 V circuitry, additional power from supply = (3.45Vpower_to_user's_circuit) × 1.92.

Additional power is taken by the input diodes.

Thus, as an example, if the user's 5 V circuitry consumes 2.5 W and the user's 3.45 V circuitry consumes 0.5 W, the total power needed from the power supply is (2.5 W × 1.33 + 0.5 W × 1.92 + (PowerCore consumption as optionally configured)) = 5.3 W to 7.0 W. If the external power supply is a 24 V supply, then it will need to source I = Power/Voltage current. Bear in mind that voltage presented to the regulators has a 0.7 V diode drop. Thus the voltage in this formula must be reduced by 0.7 V, so current = Power/23.3 V = 5.3 W/23.3 V = 227 mA; 7.0 W/23.3 V = 300 mA. An AC supply may not need to deliver that much current because the filter capacitor may raise the DC voltage input into the regulator, allowing less current to be consumed. For a 24 V DC input, this is what current the power supply would need to deliver.

The input power circuit configuration is shown below for Option 5.

OPTION 5 HALF-WAVE RECTIFIER

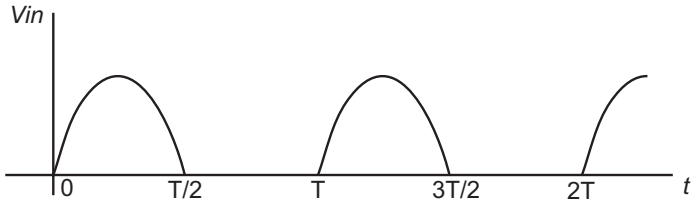


	2 A CONFIGURATION	1 A CONFIGURATION
Input Supply Requirements	17–30 V AC 8–43 V DC	14–28 V AC 9–40 V DC
Current Draw by Onboard Circuits	max I_Q = 150–400 mA @ +3.45 V	max I_Q = 150–400 mA @ +3.45 V
Output Current Available	700 mA - I_Q @ +3.45 V 2 A - I_Q - I_{3VMB} @ +5 V	700 mA - I_Q @ +3.45 V 1 A - I_Q - I_{3VMB} @ +5 V

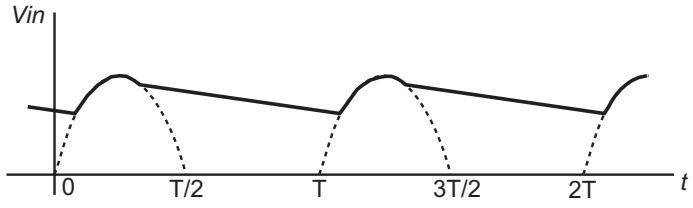
NOTE: I_{3VMB} = current consumed by user's board at +3.45 V

Half wave rectification allows a common ground between the power-supply ground and the circuit ground. Half-wave rectification only requires one diode, and so is less expensive than the other power-supply options.

The output from this half-wave rectifier is shown at right.



After the rectifier, the AC voltage is filtered with a large (1000 µF) capacitor. The final filtered voltage is shown by the solid line in the diagram at right.



A half-wave rectifier experiences only one diode voltage drop (0.7 V) during the rectification. Because current flows only during half the cycle, twice as much current will be required (albeit for only half of the wave cycle) compared to a full-wave rectifier. Since all the current going through the rectifier must go through a single diode drop, the power dissipated is 0.7 V × current. A larger, more expensive AC transformer will be needed to provide the current for a half-wave rectifier because twice the current is required during the positive half-cycle to fill the gaps.

Because there is a common AC ground between the AC power transformer and the circuit, AC zero-crossing detection is possible. Zero-crossing detection allows accurate triac timing control. Thus, the zero-crossing detection circuit is stuffed and triac control is supported with this option.

D.2 Battery-Backup Circuits

The data SRAM and the real-time clock on the PowerCore module have battery backup. Power to the SRAM and the real-time clock (VRAM) is provided by two different sources, depending on whether the PowerCore module is powered or not. When the PowerCore module is powered normally, and the +5 V supply is within operating limits, the SRAM and the real-time clock are powered from the +5 V supply. If power to the board is lost or falls below 4.38 V, the VRAM and real-time clock power will come from the battery. The reset generator circuit controls the source of power by way of its **/RESET** output signal.

A replaceable 220 mA·h lithium battery provides power to the real-time clock and SRAM when external power is removed from the circuit board. The drain on the battery is typically less than 6 µA when there is no external power applied to the PowerCore module, and so the expected shelf life of the battery is

$$\frac{220\ \text{mA}\cdot\text{h}}{6\ \mu\text{A}} = 4.2\ \text{years}.$$

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

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

D.2.1 Replacing the Backup Battery

The battery is user-replaceable, and is fitted in a battery holder. To replace the battery, slide out the old battery. Use only a 2032 or equivalent replacement lithium battery, and insert it into the battery holder with the + side facing away from the PowerCore module.

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

CAUTION: Be careful when replacing the battery with external AC power applied to the PowerCore module. AC voltages up to 60 V may be present on locking connector J3.

CAUTION: There is an explosion danger if the battery is short-circuited, recharged, or replaced incorrectly. Replace the battery only with the same type or an equivalent type recommended by the battery manufacturer. Dispose of used batteries according to the battery manufacturer's instructions.

D.3 Reset Generator

The PowerCore module uses a reset generator to reset the Rabbit 3000 microprocessor when the voltage drops below the voltage necessary for reliable operation. The reset typically occurs at 4.38 V.

The PowerCore module has a reset pin, pin 49 on header J4. This pin provides access to the reset output of the reset generator, which drives the reset input of the Rabbit 3000 and peripheral circuits. The /RESET output can be used to reset user-defined circuits on the motherboard on which the PowerCore module is mounted.

APPENDIX E. RABBITNET

E.1 General RabbitNet Description

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

E.1.1 RabbitNet Connections

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

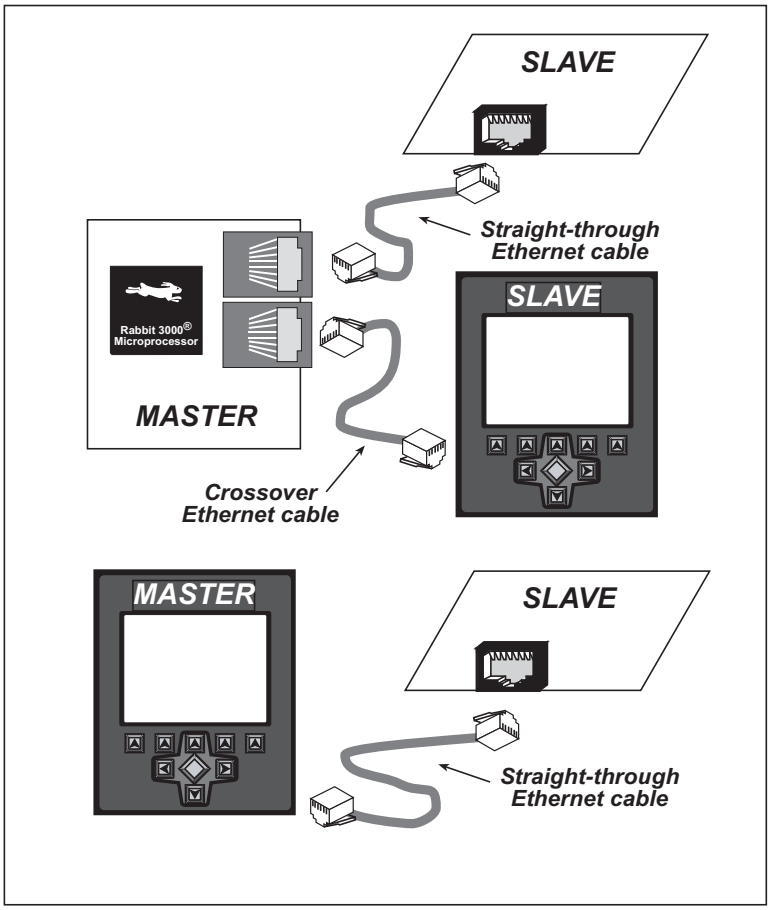


Figure E-1. Connecting Peripheral Cards to a Master

Use a straight-through Ethernet cable to connect the master to slave peripheral cards, unless you are using a device such as the OP7200 that could be used either as a master or a slave. In this case you would use a crossover cable to connect an OP7200 that is being used as a slave.

Distances between a master unit and peripheral cards can be up to 10 m or 33 ft.

E.1.2 RabbitNet Peripheral Cards

- Digital I/O

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

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

- A/D converter

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

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

- D/A converter

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

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

- Display/Keypad interface

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

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

- Relay card

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

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

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

E.2 Physical Implementation

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

The signals themselves are differential RS-422, which are series-terminated at the source. With this type of termination, the maximum frequency is limited by the round-trip delay time of the cable. Although a peripheral card could theoretically be up to 45 m (150 ft) from the master for a data rate of 1 MHz, Rabbit Semiconductor recommends a practical limit of 10 m (33 ft).

Connections between peripheral cards and masters are done using standard 8-conductor Ethernet cables. Masters and peripheral cards are equipped with RJ-45 8-pin female connectors. The cables may be swapped end for end without affecting functionality.

E.2.1 Control and Routing

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

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

E.3 Function Calls

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

rn_init

```
int rn_init(char portflag, char servicetype);
```

DESCRIPTION

Resets, initializes, or disables a specified RabbitNet port on the master single-board computer. During initialization, the network is enumerated and relevant tables are filled in. If the port is already initialized, calling this function forces a re-enumeration of all devices on that port.

Call this function first before using other RabbitNet functions.

PARAMETERS

portflag is a bit that represents a RabbitNet port on the master single-board computer (from 0 to the maximum number of ports). A set bit requires a service. If **portflag** = 0x03, both RabbitNet ports 0 and 1 will need to be serviced.

servicetype enables or disables each RabbitNet port as set by the port flags.

0 = disable port
1 = enable port

RETURN VALUE

0

rn_device

```
int rn_device(char pna);
```

DESCRIPTION

Returns an address index to device information from a given physical node address. This function will check device information to determine that the peripheral card is connected to a master.

PARAMETER

pna is the physical node address, indicated as a byte.

7,6—2-bit binary representation of the port number on the master
5,4,3—Level 1 router downstream port
2,1,0—Level 2 router downstream port

RETURN VALUE

Pointer to device information. -1 indicates that the peripheral card either cannot be identified or is not connected to the master.

SEE ALSO

rn_find

```
int rn_find(rn_search *srch);
```

DESCRIPTION

Locates the first active device that matches the search criteria.

PARAMETER

srch is the search criteria structure **rn_search:**

```
unsigned int flags;     // status flags see MATCH macros below
unsigned int ports;     // port bitmask
char productid;         // product id
char productrev;        // product rev
char coderev;           // code rev
long serialnum;         // serial number
```

Use a maximum of 3 macros for the search criteria:

```
RN_MATCH_PORT          // match port bitmask
RN_MATCH_PNA           // match physical node address
RN_MATCH_HANDLE        // match instance (reg 3)
RN_MATCH_PRDID         // match id/version (reg 1)
RN_MATCH_PRDREV        // match product revision
RN_MATCH_CODEREV       // match code revision
RN_MATCH_SN            // match serial number
```

For example:

```
rn_search newdev;
newdev.flags = RN_MATCH_PORT|RN_MATCH_SN;
newdev.ports = 0x03; //search ports 0 and 1
newdev.serialnum = E3446C01L;
handle = rn_find(&newdev);
```

RETURN VALUE

Returns the handle of the first device matching the criteria. 0 indicates no such devices were found.

SEE ALSO

rn_device

rn_echo

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

DESCRIPTION

The peripheral card sends back the character the master sent. This function will check device information to determine that the peripheral card is connected to a master.

PARAMETERS

handle	is an address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.
sendecho	is the character to echo back.
recdata	is a pointer to the return address of the character from the device.

RETURN VALUE

The status byte from the previous command. -1 means that device information indicates the peripheral card is not connected to the master.

rn_write

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

DESCRIPTION

Writes a string to the specified device and register. Waits for results. This function will check device information to determine that the peripheral card is connected to a master.

PARAMETERS

handle	is an address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.
regno	is the command register number as designated by each device.
data	is a pointer to the address of the string to write to the device.
datalen	is the number of bytes to write (0–15).

NOTE: A data length of 0 will transmit the one-byte command register number.

RETURN VALUE

The status byte from the previous command. -1 means that device information indicates the peripheral card is not connected to the master, and -2 means that the data length was greater than 15.

SEE ALSO

rn_read

rn_read

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

DESCRIPTION

Reads a string from the specified device and register. Waits for results. This function will check device information to determine that the peripheral card is connected to a master.

PARAMETERS

handle	is an address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.
regno	is the command register number as designated by each device.
recdata	is a pointer to the address of the string to read from the device.
datalen	is the number of bytes to read (0–15).

NOTE: A data length of 0 will transmit the one-byte command register number.

RETURN VALUE

The status byte from the previous command. -1 means that device information indicates the peripheral card is not connected to the master, and -2 means that the data length was greater than 15.

SEE ALSO

rn_write

rn_reset

```
int rn_reset(int handle, int resettype);
```

DESCRIPTION

Sends a reset sequence to the specified peripheral card. The reset takes approximately 25 ms before the peripheral card will once again execute the application. Allow 1.5 seconds after the reset has completed before accessing the peripheral card. This function will check peripheral card information to determine that the peripheral card is connected to a master.

PARAMETERS

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

resettype describes the type of reset.

0 = hard reset—equivalent to power-up. All logic is reset.
1 = soft reset—only the microprocessor logic is reset.

RETURN VALUE

The status byte from the previous command. -1 means that device information indicates the peripheral card is not connected to the master.

rn_sw_wdt

```
int rn_sw_wdt(int handle, float timeout);
```

DESCRIPTION

Sets software watchdog timeout period. Call this function prior to enabling the software watchdog timer. This function will check device information to determine that the peripheral card is connected to a master.

PARAMETERS

handle	is an address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.
timeout	is a timeout period from 0.025 to 6.375 seconds in increments of 0.025 seconds. Entering a zero value will disable the software watchdog timer.

RETURN VALUE

The status byte from the previous command. -1 means that device information indicates the peripheral card is not connected to the master.

rn_enable_wdt

```
int rn_enable_wdt(int handle, int wdttype);
```

DESCRIPTION

Enables the hardware and/or software watchdog timers on a peripheral card. The software on the peripheral card will keep the hardware watchdog timer updated, but will hard reset if the time expires. The hardware watchdog cannot be disabled except by a hard reset on the peripheral card. The software watchdog timer must be updated by software on the master. The peripheral card will soft reset if the timeout set by **rn_sw_wdt()** expires. This function will check device information to determine that the peripheral card is connected to a master.

PARAMETERS

handle	is an address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.
wdttype	0 enables both hardware and software watchdog timers 1 enables hardware watchdog timer 2 enables software watchdog timer

RETURN VALUE

The status byte from the previous command. -1 means that device information indicates the peripheral card is not connected to the master.

SEE ALSO

rn_hitwd, rn_sw_wdt

rn_hitwd

```
int rn_hitwd(int handle, char *count);
```

DESCRIPTION

Hits software watchdog. Set the timeout period and enable the software watchdog prior to using this function. This function will check device information to determine that the peripheral card is connected to a master.

PARAMETERS

handle	is an address index to device information. Use **rn_device()** or **rn_find()** to establish the handle.
count	is a pointer to return the present count of the software watchdog timer. The equivalent time left in seconds can be determined from **count** × 0.025 seconds.

RETURN VALUE

The status byte from the previous command. -1 means that device information indicates the peripheral card is not connected to the master.

SEE ALSO

rn_enable_wdt, rn_sw_wdt

rn_rst_status

```
int rn_rst_status(int handle, char *retdata);
```

DESCRIPTION

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

PARAMETERS

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

retdata is a pointer to the return address of the communication byte. A set bit indicates which error occurred. This register is cleared when read.

7—HW reset has occurred
6—SW reset has occurred
5—HW watchdog enabled
4—SW watchdog enabled
3,2,1,0—Reserved

RETURN VALUE

The status byte from the previous command.

rn_comm_status

```
int rn_comm_status(int handle, char *retdata);
```

PARAMETERS

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

retdata is a pointer to the return address of the communication byte. A set bit indicates which error occurred. This register is cleared when read.

7—Data available and waiting to be processed MOSI (master out, slave in)
6—Write collision MISO (master in, slave out)
5—Overrun MOSI (master out, slave in)
4—Mode fault, device detected hardware fault
3—Data compare error detected by device
2,1,0—Reserved

RETURN VALUE

The status byte from the previous command.

E.3.1 Status Byte

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

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

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

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

INDEX

R

S

T

U

W

SCHEMATICS

090-0193 PowerCore FLEX Module Schematic

www.rabbit.com/documentation/schemat/090-0193.pdf

090-0194 PowerCore Prototyping Board Schematic

www.rabbit.com/documentation/schemat/090-0194.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.