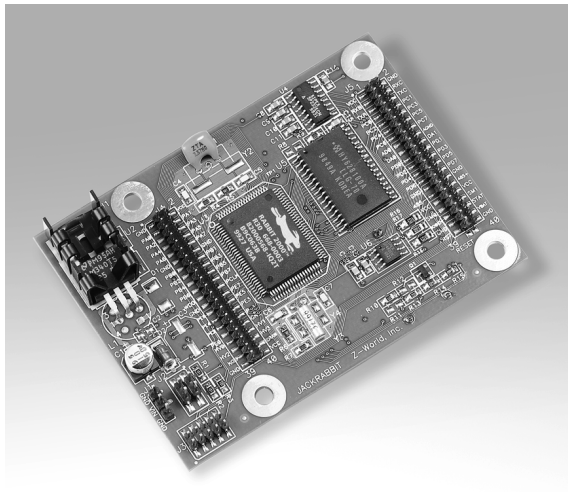




Jackrabbit (BL1800)

C-Programmable Single-Board Computer

Getting Started Manual

019-0072 • 050525-F

Jackrabbit (BL1800) Getting Started Manual

Part Number 019-0072 • 050525-F • Printed in U.S.A.

©2000–2005 Z-World Inc. • All rights reserved.

Z-World reserves the right to make changes and improvements to its products without providing notice.

Trademarks

Rabbit is a registered trademark of Rabbit Semiconductor.

Rabbit 2000 is a trademark of Rabbit Semiconductor.

Dynamic C and Z-World are registered trademarks of Z-World Inc.

Z-World, Inc.

2900 Spafford Street
Davis, California 95616-6800
USA

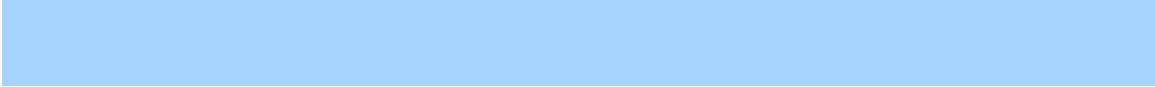
Telephone: (530) 757-3737
Fax: (530) 753-5141

www.zworld.com

TABLE OF CONTENTS

Chapter 1. Introduction	1
1.1 Features.....	1
1.1.1 Physical & Electrical Specifications	3
1.2 Development Software	4
1.3 How to Use This Manual	4
1.3.1 Additional Product Information	4
1.3.2 Additional Reference Information	4
1.3.3 Using Online Documentation.....	5
1.4 CE Compliance	6
1.4.1 Design Guidelines	7
1.4.2 Interfacing the Jackrabbit to Other Devices.....	7
 Chapter 2. Hardware Setup	 9
2.1 Development Kit Contents.....	9
2.2 Prototyping Board.....	10
2.2.1 Prototyping Board Features	11
2.3 Development Hardware Connections	12
2.3.1 Attach Jackrabbit to Prototyping Board.....	13
2.3.2 Connect Programming Cable	14
2.3.3 Connect Power	15
2.3.3.1 Overseas Development Kits	15
2.4 Run a Sample Program	16
2.4.1 Troubleshooting	16
2.5 Where Do I Go From Here?	17
2.5.1 Technical Support	17
 Chapter 3. Software Installation & Overview	 19
3.1 An Overview of Dynamic C	19
3.2 Installing Dynamic C	21
3.3 Switching Between Program Mode and Run Mode	22
3.3.1 Changing from Program Mode to Run Mode	22
3.3.2 Changing from Run Mode to Program Mode	22
3.4 Sample Programs	23
3.4.1 DEMOJR1.C	24
3.4.2 Other Sample Programs Illustrating Digital I/O	28
3.4.3 RS-232 Serial Communication Sample Programs	30
3.4.4 RS-485 Serial Communication Sample Program.....	31
3.5 Cooperative Multitasking	32
3.5.1 Advantages of Cooperative Multitasking	34

3.6 Jackrabbit Function Calls	35
3.6.1 I/O Drivers	35
3.6.1.1 Initialization	35
3.6.1.2 Digital Output	35
3.6.1.3 Analog Output	36
3.6.1.4 Analog Input	38
3.6.2 Serial Communication Drivers	39
3.6.2.1 RS-485 Serial Communication Drivers	39
3.7 Upgrading Dynamic C	40
3.7.1 Add-On Modules	40
Notice to Users	41
Index	43
Schematics	45



1. INTRODUCTION

The Jackrabbit is a high-performance, C-programmable single-board computer with a compact form factor. A Rabbit[®] 2000 microprocessor operating at 29.5 MHz provides fast data processing.

1.1 Features

- 29.5 MHz clock
- 24 CMOS-compatible I/O
- 3 analog channels: 1 A/D input, 2 PWM D/A outputs
- 4 high-power outputs (factory-configured as 3 sinking and 1 sourcing)
- 4 serial ports (2 RS-232 or 1 RS-232 with RTS/CTS, 1 RS-485, and 1 CMOS-compatible)
- 6 timers (five 8-bit timers and one 10-bit timer)
- 128K SRAM, 256K flash EPROM
- Real-time clock
- Watchdog supervisor
- Voltage regulator
- Backup battery

Appendix A provides detailed specifications for the Jackrabbit.

Three versions of the Jackrabbit are available. Their standard features are summarized in Table 1.

Table 1. Jackrabbit Features

Model	Features
BL1800	Full-featured controller with switching voltage regulator.
BL1810	BL1800 with 14.7 MHz clock, 128K flash EPROM, linear voltage regulator, sinking outputs sink up to 200 mA, sourcing output sources up to 100 mA, RS-232 serial ports rated for 1 kV ESD
BL1820	BL1810 with 3 additional digital I/O, no RS-485, no backup battery, 32K SRAM.

Throughout this manual, the term Jackrabbit refers to all three Jackrabbit models in Table 1; individual models are referred to specifically according to the model number in Table 1.

1.1.1 Physical & Electrical Specifications

Table 2 lists the basic specifications for the Jackrabbit.

Table 2. Jackrabbit Specifications

Specification	Data
Power Supply	7.5 V to 25 V DC, 100 mA typical
Size	2.50" × 3.50" × 0.94" (64 mm × 89 mm × 24 mm)
Environmental	–40°C to 70°C, 5–95% humidity, noncondensing

NOTE: For complete product specifications, see Appendix A in the *Jackrabbit (BL1800) User's Manual*.

Figure 1 shows the pinout for headers J4 and J5, which carry the signals associated with the Jackrabbit subsystems.

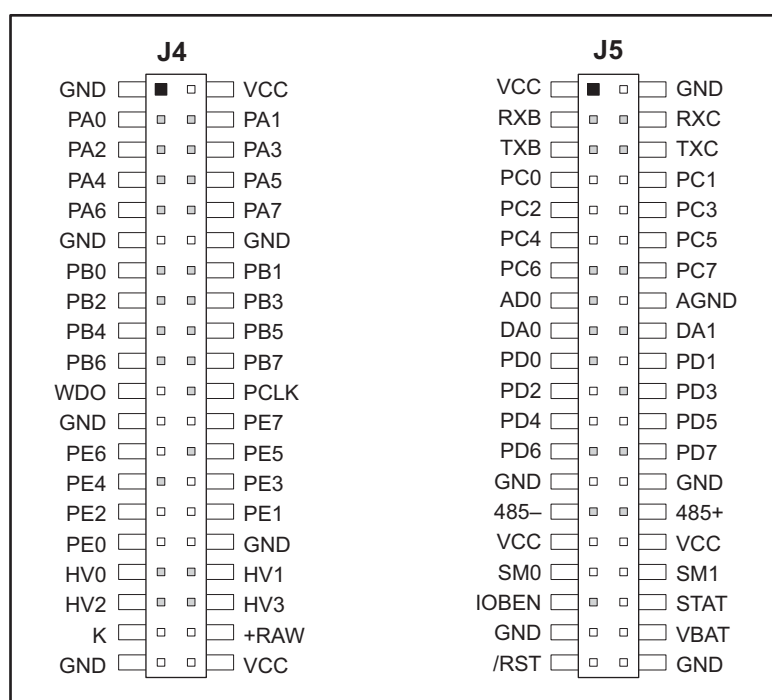


Figure 1. Pinout for Jackrabbit Headers J4 and J5

1.2 Development Software

The Jackrabbit uses the Dynamic C development environment for rapid creation and debugging of runtime applications. Dynamic C provides a complete development environment with integrated editor, compiler and source-level debugger. It interfaces directly with the target system, eliminating the need for complex and unreliable in-circuit emulators.

Dynamic C must be installed on a Windows workstation with at least one free serial USB or COM port for communication with the target system. See Chapter 3., “Software Installation & Overview,” for complete information on installing Dynamic C.

1.3 How to Use This Manual

This *Getting Started* manual is intended to give users a quick but solid start with the Jackrabbit. It does not contain detailed information on the hardware capabilities or the Dynamic C development environment. Most users will want more detailed information on some or all of these topics in order to put the Jackrabbit to effective use.

1.3.1 Additional Product Information

Detailed information about the Jackrabbit will be found in the *Jackrabbit (BL1800) User's Manual*, provided on the accompanying CD-ROM in both HTML and Adobe PDF format.

TIP: We recommend that anyone not thoroughly familiar with Z-World embedded control systems at least read through the rest of this manual to gain the necessary familiarity to make use of the more advanced information.

1.3.2 Additional Reference Information

In addition to the product-specific information contained in the *Jackrabbit (BL1800) User's Manual*, several higher level reference manuals are provided in HTML and PDF form on the accompanying CD-ROM. Advanced users will find these references valuable in developing systems based on the Jackrabbit:

- *Dynamic C User's Manual*
- *Dynamic C Function Reference Manual*
- *Rabbit 2000 Microprocessor User's Manual*

1.3.3 Using Online Documentation

We provide the bulk of our user and reference documentation in two electronic formats, HTML and Adobe PDF. We do this for several reasons.

We believe that providing all users with our complete library of product and reference manuals is a useful convenience. However, printed manuals are expensive to print, stock and ship. Rather than include and charge for manuals that every user may not want, or provide only product-specific manuals, we choose to provide our complete documentation and reference library in electronic form with every development kit and with our Dynamic C development environment.

NOTE: The most current version of Adobe Acrobat Reader can always be downloaded from Adobe's web site at <http://www.adobe.com>. We recommend that you use version 4.0 or later.

Providing this documentation in electronic form saves an enormous amount of paper by not printing copies of manuals that users don't need.

Finding Online Documents

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, create a new desktop icon that points to **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.

Printing Electronic Manuals

We recognize that many users prefer printed manuals for some uses. Users can easily print all or parts of those manuals provided in electronic form. The following guidelines may be helpful:

- Print from the Adobe PDF versions of the files, not the HTML versions.
- If your printer supports duplex printing, print pages double-sided.
- If you do not have a suitable printer or do not want to print the manual yourself, most retail copy shops (e.g. Kinkos, AlphaGraphics, etc.) will print the manual from the PDF file and bind it for a reasonable charge—about what we would have to charge for a printed and bound manual.

1.4 CE Compliance

Equipment is generally divided into two classes.

CLASS A	CLASS B
Digital equipment meant for light industrial use	Digital equipment meant for home use
Less restrictive emissions requirement: less than 40 dB $\mu\text{V/m}$ at 10 m (40 dB relative to 1 $\mu\text{V/m}$) or 300 $\mu\text{V/m}$	More restrictive emissions requirement: 30 dB $\mu\text{V/m}$ at 10 m or 100 $\mu\text{V/m}$

These limits apply over the range of 30–230 MHz. The limits are 7 dB higher for frequencies above 230 MHz. Although the test range goes to 1 GHz, the emissions from Rabbit-based systems at frequencies above 300 MHz are generally well below background noise levels.

The Jackrabbit BL1800 single-board computer has been tested and was found to be in conformity with the following applicable immunity and emission standards. The BL1810 and BL1820 single-board computers are also CE qualified as they are sub-versions of the BL1800 single-board computer. Boards that are CE-compliant have the CE mark.



NOTE: Earlier versions of the BL1800 sold before 2002 that do not have the CE mark are **not** CE-complaint.

Immunity

The Jackrabbit series of single-board computers meets the following EN55024/1998 immunity standards.

- EN61000-4-3 (Radiated Immunity)
- EN61000-4-4 (EFT)
- EN61000-4-6 (Conducted Immunity)

Additional shielding or filtering may be required for a heavy industrial environment.

Emissions

The Jackrabbit series of single-board computers meets the following emission standards with the Rabbit 2000 spectrum spreader turned on and set to the normal mode. The spectrum spreader is only available with Rev. C or higher of the Rabbit 2000 microprocessor. This microprocessor is used in all Jackrabbit series boards that carry the CE mark.

- EN55022:1998 Class B
- FCC Part 15 Class B

In order for the Jackrabbit boards to meet these EN55022:1998 Class B standards, you must add ferrite absorbers to the serial I/O cables used for RS-232 and RS-485 serial communication. Depending on your application, you may need to add ferrite absorbers to the

digital I/O cables. Your results may vary, depending on your application, so additional shielding or filtering may be needed to maintain the Class B emission qualification.

NOTE: If no ferrite absorbers are fitted, the Jackrabbit boards will still meet EN55022:1998 Class A requirements as long as the spectrum spreader is turned on.

The spectrum spreader is on by default for Jackrabbit models BL1810 and BL1820. The spectrum spreader is off by default for the Jackrabbit model BL1800, and must be turned on with at least one wait state in order for the BL1800 model to be CE-compliant. Section 2.7.3 in the *Jackrabbit (BL1800) User's Manual* provides further information about the spectrum spreader and its use, and includes information on how to add a wait state.

1.4.1 Design Guidelines

Note the following requirements for incorporating the Jackrabbit series of single-board computers into your application to comply with CE requirements.

General

- The power supply provided with the Development Kit is for development purposes only. It is the customer's responsibility to provide a CE-compliant power supply for the end-product application.
- When connecting the Jackrabbit single-board computer to outdoor cables, the customer is responsible for providing CE-approved surge/lightning protection.
- Z-World recommends placing digital I/O or analog cables that are 3 m or longer in a metal conduit to assist in maintaining CE compliance and to conform to good cable design practices. Z-World also recommends using properly shielded I/O cables in noisy electromagnetic environments.
- When installing or servicing the Jackrabbit, it is the responsibility of the end-user to use proper ESD precautions to prevent ESD damage to the Jackrabbit.

Safety

- For personal safety, all inputs and outputs to and from the Jackrabbit series of single-board computers must not be connected to voltages exceeding SELV levels (42.4 V AC peak, or 60 V DC). Damage to the Rabbit 2000 microprocessor may result if voltages outside the design range of 0 V to 5.5 V DC are applied directly to any of its digital inputs.
- The lithium backup battery circuit on the Jackrabbit single-board computer has been designed to protect the battery from hazardous conditions such as reverse charging and excessive current flows. Do not disable the safety features of the design.

1.4.2 Interfacing the Jackrabbit to Other Devices

Since the Jackrabbit series of single-board computers is designed to be connected to other devices, good EMC practices should be followed to ensure compliance. CE compliance is ultimately the responsibility of the integrator. Additional information, tips, and technical assistance are available from your authorized Z-World distributor, and are also available on our Web site at www.zworld.com.

2. HARDWARE SETUP

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

NOTE: This chapter (and this manual) assume that you have the Jackrabbit Development Kit. If you purchased a Jackrabbit board by itself, you will have to adapt the information in this chapter and elsewhere to your test and development setup.

2.1 Development Kit Contents

The Jackrabbit Development Kit contains the following items:

- BL1810 single-board computer.
- Prototyping Board.
- AC adapter, 12 V DC, 1 A. (Included only with Development Kits sold for the North American market. A header plug leading to bare leads is provided to allow overseas users to connect their own power supply with a DC output of 7.5–25 V.)

NOTE: The linear voltage regulator becomes rather hot for voltages above 15 V.

- 10-pin header to DE9 programming cable with integrated level-matching circuitry.
- *Dynamic C* CD-ROM, with complete product documentation on disk.
- This *Getting Started* manual.
- A bag of accessory parts for use on the Prototyping Board.
- Screwdriver.
- *Rabbit 2000 Processor Easy Reference* poster.
- Registration card.

2.2 Prototyping Board

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

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

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

The Prototyping Board is shown below in Figure 2, with its main features identified.

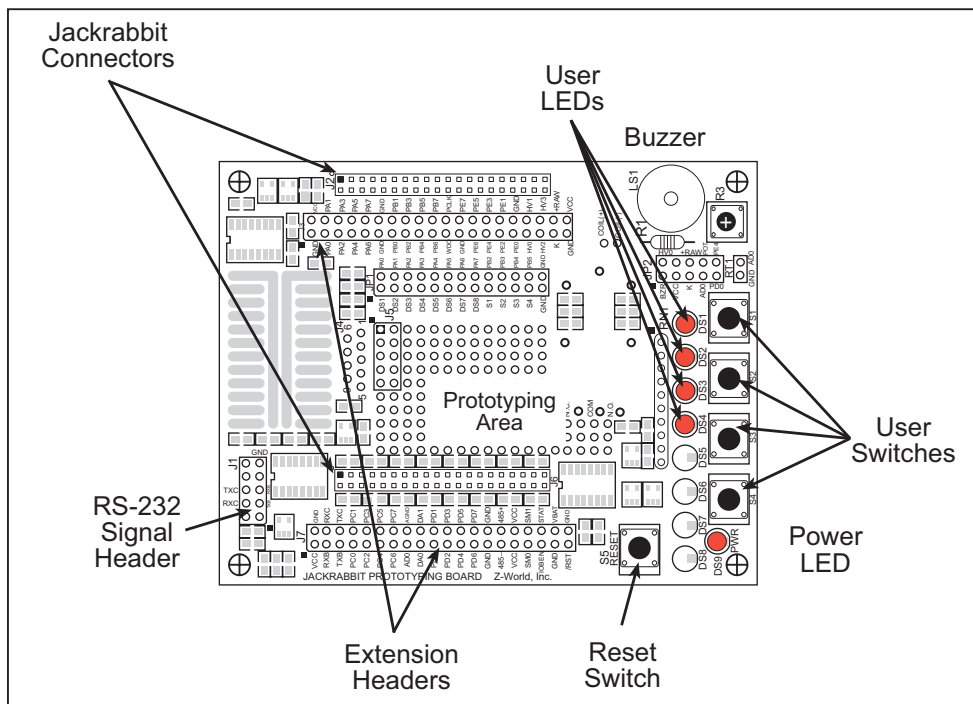


Figure 2. Prototyping Board

2.2.1 Prototyping Board Features

- **Power LED**—The power LED lights whenever power is connected to the Prototyping Board.
- **Reset Switch**—A momentary-contact, normally open switch is connected directly to the Jackrabbit's **/RESET_IN** pin. Pressing the switch forces a hardware reset of the system.
- **I/O Switches and LEDs**—Four momentary-contact, normally open switches are connected to the PB2–PB5 pins of the Rabbit 2000 microprocessor on the Jackrabbit, and may be read as inputs by sample applications.

Four LEDs are connected to the PA1–PA4 pins of the of the Rabbit 2000 microprocessor on the Jackrabbit, and may be driven as output indicators by sample applications.

- **Prototyping Area**—A generous prototyping area has been provided for the installation of through-hole components. Several areas for surface-mount devices are also available. (Note that there are SMT device pads on both top and bottom of the Prototyping Board.)
- **Extension Headers**—The complete pin sets of the Jackrabbit are duplicated at these two sets of headers. Developers can solder wires directly into the appropriate holes, or, for more flexible development, 40-pin header strips can be soldered into place. See Figure 1 for the header pinouts.
- **RS-232**—Two 3-wire or one 5-wire RS-232 serial port are available on the Prototyping Board. Refer to the Prototyping Board schematic (090-0088) for additional details.

A 10-pin 0.1-inch spacing header strip is installed at J1 to permit connection of a ribbon cable leading to a standard DE-9 serial connector.

2.3 Development Hardware Connections

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

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

2.3.1 Attach Jackrabbit to Prototyping Board

To attach the Jackrabbit board to the Prototyping Board, turn the Jackrabbit board over so that the battery is facing up. Plug the pins from headers J4 and J5 on the bottom side of the Jackrabbit board into the header sockets at J2 and J6 on the Prototyping Board as indicated in Figure 3.

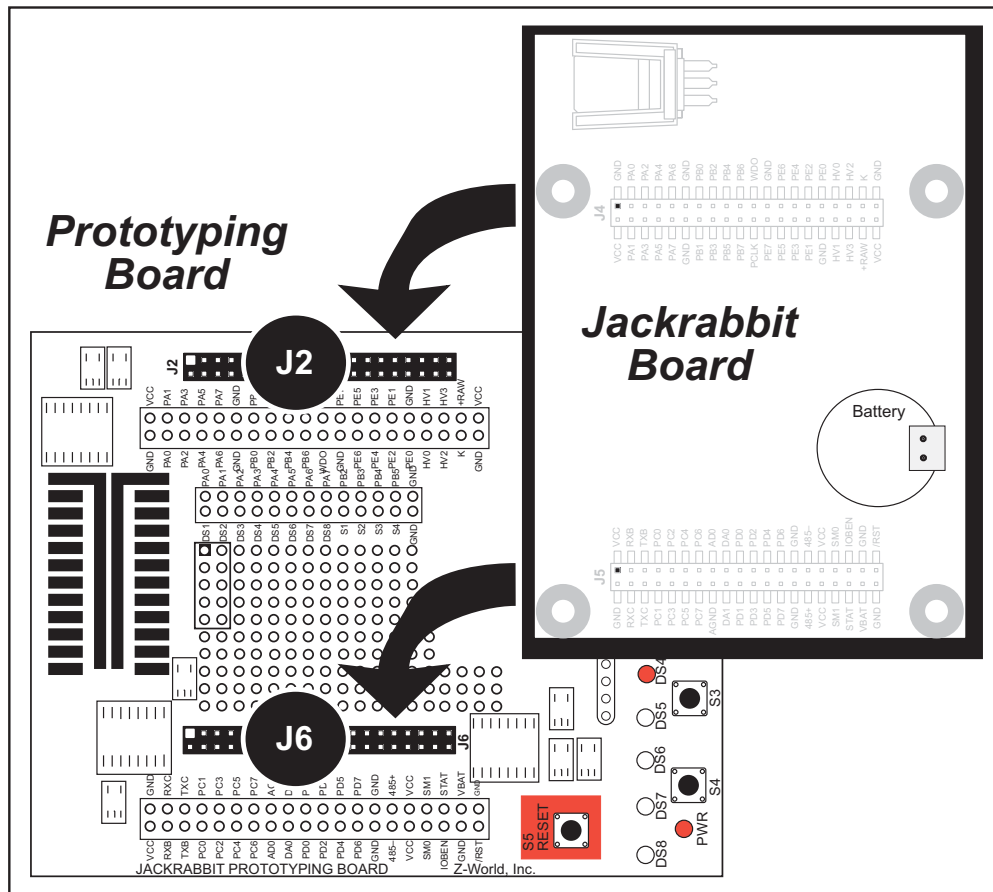


Figure 3. Attach Jackrabbit Board to Prototyping Board

NOTE: It is important that you line up the pins on headers J4 and J5 of the Jackrabbit board exactly with the corresponding pins of header sockets J2 and J6 on the Prototyping Board. The header pins may become bent or damaged if the pin alignment is offset, and the Jackrabbit might not work. Permanent electrical damage to the may also result if a misaligned Jackrabbit is powered up.

Press the Jackrabbit's pins firmly into the Prototyping Board headers.

2.3.2 Connect Programming Cable

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

Connect the 10-pin connector of the programming cable labeled **PROG** to header J3 on the Jackrabbit board as shown in Figure 4. Be sure to orient the marked (usually red) edge of the cable towards pin 1 of the connector. (Do not use the **DIAG** connector, which is used for a normal serial connection.)

NOTE: Use only the programming cable that has a red shrink wrap around the RS-232 level converter (Z-World part number 101-0513), which is supplied with the Development Kit. Other Z-World programming cables might not be voltage-compatible or their connector sizes may be different.

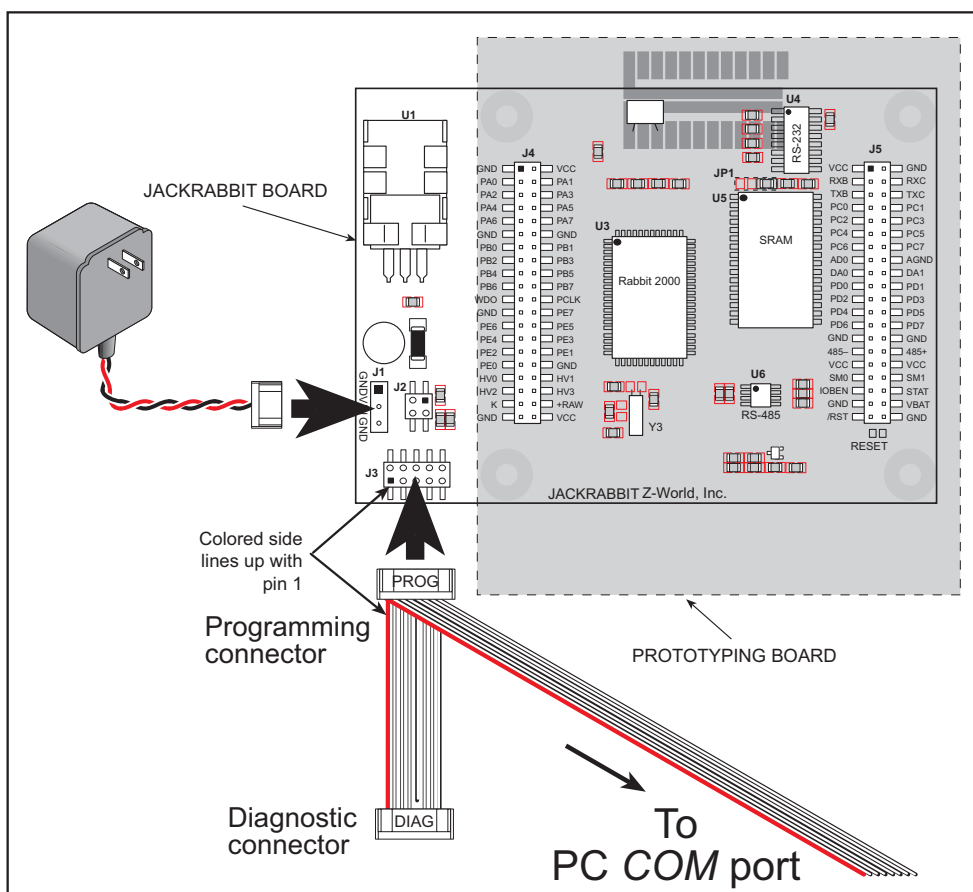


Figure 4. Power and Programming Cable Connections to Jackrabbit Board

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

NOTE: Some PCs now come equipped only with a USB port. It may be possible to use an RS-232/USB converter with the programming cable supplied with the Jackrabbit Development Kit. An RS-232/USB converter is available through the Z-World Web store.

2.3.3 Connect Power

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

Hook up the connector from the wall transformer to header J1 on the Jackrabbit board as shown in Figure 4. The orientation of this connector is not important since the VIN (positive) voltage is the middle pin, and GND is available on both ends of the three-pin header J1.

Plug in the wall transformer. The Jackrabbit board and the Prototyping Board are ready to be used.

NOTE: A **RESET** button is provided on the Prototyping Board (see Figure 3) to allow hardware reset without disconnecting power.

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

2.3.3.1 Overseas Development Kits

Development kits sold outside North America include a header connector that may be connected to 3-pin header J1 on the Jackrabbit. The connector may be attached either way as long as it is not offset to one side. The red and black wires from the connector can then be connected to the positive and negative connections on your power supply. The power supply should deliver 7.5 V–25 V DC at 5 W.

2.4 Run a Sample Program

If you already have Dynamic C installed, you are now ready to test your programming connections by running a sample program.

If you are using a USB port to connect your computer to the Jackrabbit, choose **Options > Project Options** and select “Use USB to Serial Converter” under the **Communications** tab.

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

This program shows that the CPU is working.

2.4.1 Troubleshooting

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

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

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

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

If there are any other problems:

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

2.5 Where Do I Go From Here?

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

1. Run all of the sample programs described in Section 3.4 to get a basic familiarity with Dynamic C and the Jackrabbit's capabilities.
2. For further development, refer to the *Jackrabbit (BL1800) User's Manual* for details of the board's hardware components.

A documentation icon should have been installed on your workstation's desktop; click on it to reach the documentation menu. You can create a new desktop icon that points to **default.htm** in the **docs** folder in the Dynamic C installation folder.

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

2.5.1 Technical Support

NOTE: If you purchased your Jackrabbit through a distributor or through a Z-World partner, contact the distributor or partner first for technical support.

If there are any problems at this point:

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

3. SOFTWARE INSTALLATION & OVERVIEW

To develop and debug programs for the Jackrabbit (and for all other Z-World and Rabbit Semiconductor hardware), you must install and use Dynamic C. It runs on an IBM-compatible PC and is designed for use with Z-World single-board computers and other devices based on the Rabbit microprocessor. This chapter takes you through the installation of Dynamic C, and then provides a tour of its major features with respect to the Jackrabbit.

3.1 An Overview of 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 the choice of doing your software development in the flash memory or in the data SRAM included on the Jackrabbit. The flash memory and SRAM options are selected with the **Options > Project Options > Compiler** menu.

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

NOTE: An application can be developed in RAM, but cannot run standalone from RAM after the programming cable is disconnected. All standalone applications can only run from flash memory.

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

Developing software with Dynamic C is simple. Users can write, compile, and test C and assembly code without leaving the Dynamic C development environment. Debugging occurs while the application runs on the target. Alternatively, users can compile a program to an image file for later loading. Dynamic C runs on PCs under Windows 95, 98, 2000, NT, Me, and XP. Programs can be downloaded at baud rates of up to 460,800 bps after the program compiles.

Dynamic C has a number of standard features:

- Full-feature source and/or assembly-level debugger, no in-circuit emulator required.
- Royalty-free TCP/IP stack with source code and most common protocols.
- Hundreds of functions in source-code libraries and sample programs:
 - ▶ Exceptionally fast support for floating-point arithmetic and transcendental functions.
 - ▶ RS-232 and RS-485 serial communication.
 - ▶ Analog and digital I/O drivers.
 - ▶ I²C, 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 Z-World targets in the absence of Dynamic C.
- Provision for customers to create their own source code libraries and augment on-line help by creating “function description” block comments using a special format for library functions.
- Standard debugging features:
 - ▶ Breakpoints—Set breakpoints that can disable interrupts.
 - ▶ Single-stepping—Step into or over functions at a source or machine code level, μ C/OS-II aware.
 - ▶ Code disassembly—The disassembly window displays addresses, opcodes, mnemonics, and machine cycle times. Switch between debugging at machine-code level and source-code level by simply opening or closing the disassembly window.
 - ▶ Watch expressions—Watch expressions are compiled when defined, so complex expressions including function calls may be placed into watch expressions. Watch expressions can be updated with or without stopping program execution.
 - ▶ Register window—All processor registers and flags are displayed. The contents of general registers may be modified in the window by the user.
 - ▶ Stack window—shows the contents of the top of the stack.
 - ▶ Hex memory dump—displays the contents of memory at any address.
 - ▶ **STDIO** window—`printf` outputs to this window and keyboard input on the host PC can be detected for debugging purposes. `printf` output may also be sent to a serial port or file.

3.2 Installing Dynamic C

Insert the Dynamic C CD from the Development Kit in your PC's CD-ROM drive. If the installation does not auto-start, run the **setup.exe** program in the root directory of the Dynamic C CD. Install any Dynamic C modules after you install Dynamic C.

Dynamic C has two components that can be installed together or separately. One component is Dynamic C itself, with the development environment, support files and libraries. The other component is the documentation library in HTML and PDF formats, which may be left uninstalled to save hard drive space or installed elsewhere (on a separate or network drive, for example).

The installation type is selected in the installation menu shown above. The options are:

- **Typical Installation** — Both Dynamic C and the documentation library will be installed in the specified folder (default).
- **Compact Installation** — Only Dynamic C will be installed.
- **Custom Installation** — You will be allowed to choose which components are installed. This choice is useful to install or reinstall just the documentation.

3.3 Switching Between Program Mode and Run Mode

The Jackrabbit is automatically in Program Mode when the programming cable is attached, and is automatically in Run Mode when no programming cable is attached. See Figure 5.

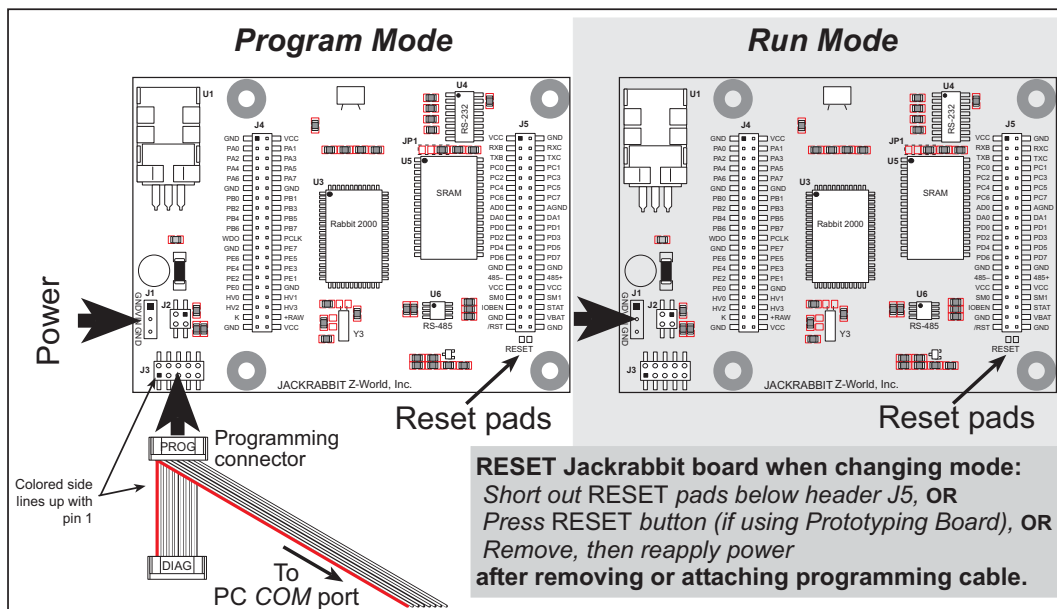


Figure 5. Jackrabbit Program Mode and Run Mode Setup

3.3.1 Changing from Program Mode to Run Mode

1. Disconnect the programming cable from header J3 of the Jackrabbit board.
2. Reset the Jackrabbit board. You may do this as explained in Figure 5. Figure 6 shows the location of the RESET button on the Prototyping Board.

The Jackrabbit is now ready to operate in the Run Mode.

3.3.2 Changing from Run Mode to Program Mode

1. Attach the programming cable to header J3 on the Jackrabbit board.
2. Reset the Jackrabbit board. You may do this as explained in Figure 5. Figure 6 shows the location of the RESET button on the Prototyping Board.

The Jackrabbit is now ready to operate in the Program Mode.

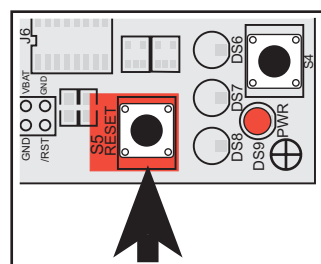


Figure 6. Location of Prototyping Board Reset Button

3.4 Sample Programs

A series of sample programs is provided in the Dynamic C **SAMPLES/JACKRAB** folder to help familiarize you with the Jackrabbit. Loading, executing and studying these programs will give you a solid hands-on overview of the Jackrabbit's capabilities, as well as a quick start with Dynamic C as an application development tool. These programs are intended to serve as tutorials, but then can also be used as starting points or building blocks for your own applications.

NOTE: It is assumed in this section that you have at least an elementary grasp of ANSI C. If you do not, see the introductory pages of the *Dynamic C User's Manual* for a suggested reading list.

Each sample program has comments that describe the purpose and function of the program.

Before running any of these sample program, make sure that your Jackrabbit is connected to the Prototyping Board and to your PC as described in Section 2.3, "Development Hardware Connections."

To run a sample program, open it with the **File** menu (if it is not already open), then compile and run it by pressing **F9** or by selecting **Run** in the **Run** menu.

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

The sample programs are listed in Table 3.

Table 3. Jackrabbit Sample Programs

DEMOJR1.C
DEMOJR2.C
DEMOJR3.C
JRIOTEST.C
JR_FLOWCONTROL.C
JR_PARITY.C
JRIO_COF.C
LCD_DEMO.C
RABDB01.C
RABDB02.C

The first five sample programs provide a step-by-step introduction to the Jackrabbit board. Additional sample programs illustrate more advanced topics.

3.4.1 DEMOJR1.C

This sample program can be used to illustrate some of the functions of Dynamic C.

First, open the file **DEMOJR1.C**, which is in the **SAMPLES/JACKRAB** folder. The program will appear in a window, as shown in Figure 7 below (minus some comments). Use the mouse to place the cursor on the function name **WrPortI** in the program and type **<Ctrl-H>**. This will bring up a documentation box for the function **WrPortI**. In general, you can do this with all functions in Dynamic C libraries, including libraries you write yourself. Close the documentation box and continue.

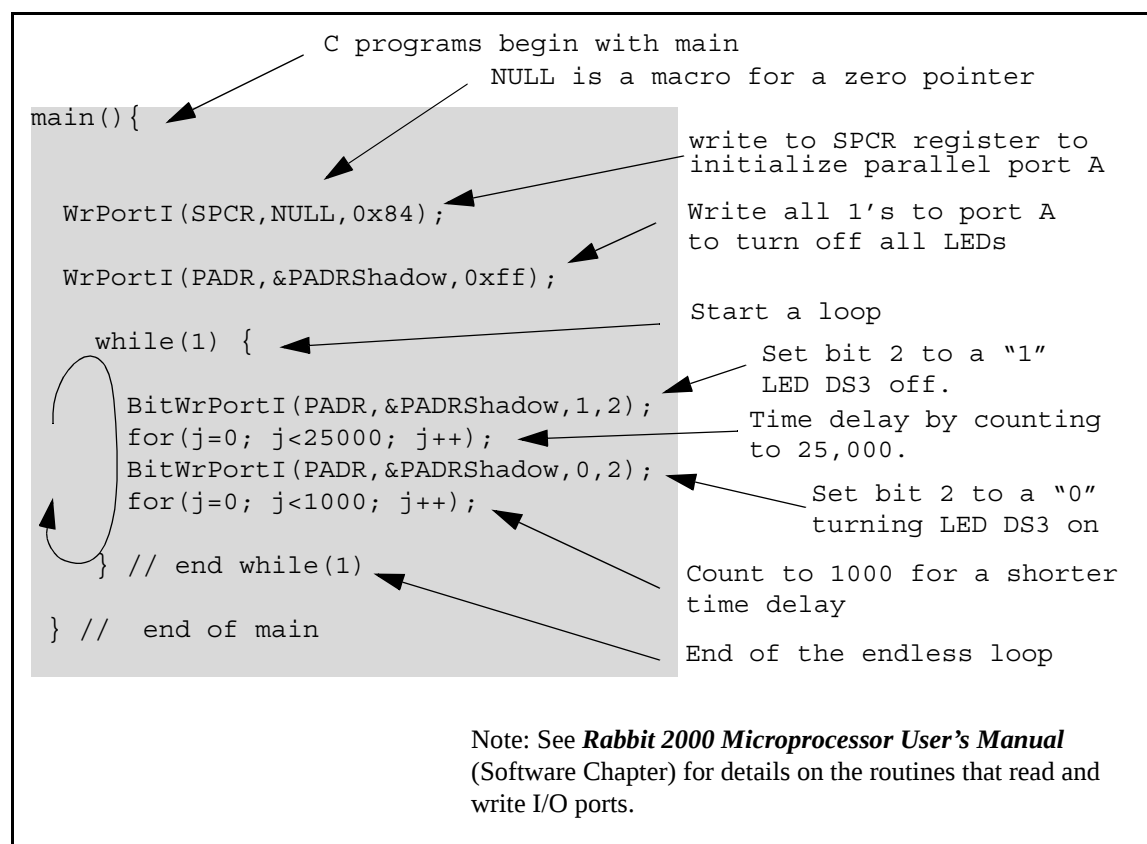


Figure 7. Sample Program DEMOJR1.C

To run the program **DEMOJR1.C**, load it with the **File > Open** menu, then compile and run it by pressing **F9** or by selecting **Run** in the **Run** menu. The LED on the Development Board should start flashing if everything went well. If this doesn't work review the following points.

- The target should be ready, which is indicated by the message "BIOS successfully compiled..." If you did not receive this message or you get a communication error, recompile the BIOS by typing **<Ctrl-Y>** or select **Recompile BIOS** from the **Compile** menu.
- A message reports that "No Rabbit processor detected" in cases where the Jackrabbit and Prototyping Board are not connected together, the wall transformer is not connected, or is not plugged in. (The red power LED lights whenever power is connected.)

- The programming cable must be connected to the Jackrabbit board. (The colored wire on the programming cable is closest to pin 1 on header J3 on the Jackrabbit board, as shown in Figure 4.) The other end of the programming cable must be connected to the PC serial port. The COM port specified in the Dynamic C **Options** menu must be the same as the one the programming cable is connected to.
- To check if you have the correct serial port, select **Compile**, then **Compile BIOS**, or type **<Ctrl-Y>**. If the “BIOS successfully compiled ...” message does not display, try a different serial port using the Dynamic C **Options** menu until you find the one you are plugged into. Don’t change anything in this menu except the COM number. The baud rate should be 115,200 bps and the stop bits should be 1.

Single-Stepping

Compile or re-compile **DEMOJR1.C** by clicking the **Compile** button on the task bar. The program will compile and the screen will come up with a highlighted character (green) at the first executable statement of the program. Use the **F8** key to single-step. Each time the **F8** key is pressed, the cursor will advance one statement. When you get to the **for (j=0, j< ...** statement, it becomes impractical to single-step further because you would have to press **F8** thousands of times. We will use this statement to illustrate watch expressions.

Watch Expression

Type **<Ctrl-W>** or chose **Add/Del Watch Expression** in the **Inspect** menu. A box will come up. Type the lower case letter **j** and click on *add to top* and *close*. Now continue single-stepping with **F8**. Each time you step, the watch expression (**j**) will be evaluated and printed in the watch window. Note how the value of **j** advances when the statement **j++** is executed.

Break Point

Move the cursor to the start of the statement:

```
for(j=0; j<1000; j++);
```

To set a break point on this statement, type **F2** or select **Breakpoint** from the **Run** menu. A red highlight will appear on the first character of the statement. To get the program running at full speed, type **F9** or select **Run** on the **Run** menu. The program will advance until it hits the break point. Then the break point will start flashing and show both red and green colors. Note that LED DS3 is now solidly turned on. This is because we have passed the statement turning on LED DS3. Note that **j** in the watch window has the value 25000. This is because the loop above terminated when **j** reached 25000.

To remove the break point, type **F2** or select **Toggle Breakpoint** on the **Run** menu. To continue program execution, type **F9** or select **Run** from the **Run** menu. Now the LED should be flashing again since the program is running at full speed.

You can set break points while the program is running by positioning the cursor to a statement and using the **F2** key. If the execution thread hits the break point, a break point will take place. You can toggle the break point off with the **F2** key and continue execution with the **F9** key. Try this a few times to get the feel of things.

Editing the Program

Click on the **Edit** box on the task bar. This will set Dynamic C into the edit mode so that you can change the program. Use the **Save as** choice on the **File** menu to save the file with a new name so as not to change the demo program. Save the file as **MYTEST.C**. Now change the number 25000 in the **for** (. . statement to 10000. Then use the **F9** key to recompile and run the program. The LED will start flashing, but it will flash much faster than before because you have changed the loop counter terminal value from 25000 to 10000.

Watching Variables Dynamically

Go back to edit mode (select edit) and load the program **DEMOJR2.C** using the **File** menu **Open** command. This program is the same as the first program, except that a variable **k** has been added along with a statement to increment **k** each time around the endless loop. The statement:

```
runwatch();
```

has been added. This is a debugging statement that makes it possible to view variables while the program is running.

Use the **F9** key to compile and run **DEMOJR2.C**. Now type **<Ctrl-W>** to open the watch window and add the watch expression **k** to the top of the list of watch expressions. Now type **<Ctrl-U>**. Each time you type **<Ctrl-U>**, you will see the current value of **k**, which is incrementing about 5 times a second.

As an experiment add another expression to the watch window:

```
k*5
```

Then type **<Ctrl-U>** several times to observe the watch expressions **k** and **k*5**.

Summary of Features

So far you have practiced using the following features of Dynamic C.

- Loading, compiling and running a program. When you load a program it appears in an edit window. You can compile by selecting **Compile** on the task bar or from the **Compile** menu. When you compile the program, it is compiled into machine language and downloaded to the target over the serial port. The execution proceeds to the first statement of main where it pauses, waiting for you to command the program to run, which you can do with the **F9** key or by selecting **Run** on the **Run** menu. If want to compile and start the program running with one keystroke, use **F9**, the run command. If the program is not already compiled, the run command will compile it first.
- Single-stepping. This is done with the **F8** key. The **F7** key can also be used for single-stepping. If the **F7** key is used, then descent into subroutines will take place. With the **F8** key the subroutine is executed at full speed when the statement that calls it is stepped over.

- Setting break points. The **F2** key is used to turn on or turn off (toggle) a break point at the cursor position if the program has already been compiled. You can set a break point if the program is paused at a break point. You can also set a break point in a program that is running at full speed. This will cause the program to break if the execution thread hits your break point.
- Watch expressions. A watch expression is a C expression that is evaluated on command in the watch window. An expression is basically any type of C formula that can include operators, variables and function calls, but not statements that require multiple lines such as *for* or *switch*. You can have a list of watch expressions in the watch window. If you are single-stepping, then they are all evaluated on each step. You can also command the watch expression to be evaluated by using the **<Ctrl-U>** command. When a watch expression is evaluated at a break point, it is evaluated as if the statement was at the beginning of the function where you are single-stepping. If your program is running you can also evaluate watch expressions with a **<Ctrl-U>** if your program has a **run-watch()** command that is frequently executed. In this case, only expressions involving global variables can be evaluated, and the expression is evaluated as if it were in a separate function with no local variables.

3.4.2 Other Sample Programs Illustrating Digital I/O

- **DEMOJR2.C**—repeatedly flashes LED DS3 (which is controlled by PA2) on the Prototyping Board.

This sample program also illustrates the use of the `runwatch()` function to allow Dynamic C to update watch expressions while running. To test this:

1. Add a watch expression for "k" under "Inspect:Add/Del Watch Expression."
2. Click "Add to top" so that it will be permanently in the watch list.
3. While the program is running, type **<Ctrl+U>** to update the watch window.

- **DEMOJR3.C**—demonstrates the use of costatements to LED DS4 (which is controlled by PA3) on the Prototyping Board. This sample program will also watch button S1 (PB2) and toggle LED DS1 (which is controlled by PA0) on/off when pressed. Note that S1 presses are debounced by the software.

The pins on Parallel Port A can all be set as either outputs or as inputs via the slave port control register (SPCTR). Do not use Parallel Port A if the slave port is being used.

Bits 0–5 on Parallel Port B are always inputs, and bits 6–7 are always outputs. Do not use Parallel Port B if the slave port is being used.

- **JRIOTEST.C**—exercises the JackRabbit's four digital output channels, the one analog input channel, and the two analog output channels.
- **JRIO_COF.C**—demonstrates the use of cofunctions with the analog input driver. Before you run this sample program, connect DA1 to AD0 on header J7 of the Prototyping Board to provide an input voltage. Once the sample program is running, it will read the input voltage ten times while another costatement is executed concurrently. The values will be printed out in the Dynamic C **STDIO** window at the end of the program.

Before running the **RABDB01.C** and the **RABDB02.C** sample programs, you will need to install 3 mm LEDs such as the Vishay Telefunken TLUR4400 at DS5–DS8 on the Jackrabbit Prototyping Board. These LEDs are included with the Jackrabbit Development Kit.

- **RABDB01.C**—flashes LEDs DS5–DS8 on the Jackrabbit Prototyping Board (which are connected to PA4–PA7) when corresponding switches S1–S4 (which are connected to PB2–PB5) are pressed. The buzzer, which is driven by HV0 from PE0, will also sound whenever switch S1 switch is pressed.
- **RABDB02.C**—flashes LEDs DS5–DS8 on the Jackrabbit Prototyping Board (which are connected to PA4–PA7) when corresponding switches S1–S4 (which are connected to PB2–PB5) are pressed. The buzzer, which is driven by HV0 from PE0, will also sound whenever switch S1 switch is pressed.

Before running the `LCD_DEMO.C` sample program, you will need an LCD based on the HD44780 (or an equivalent) controller.

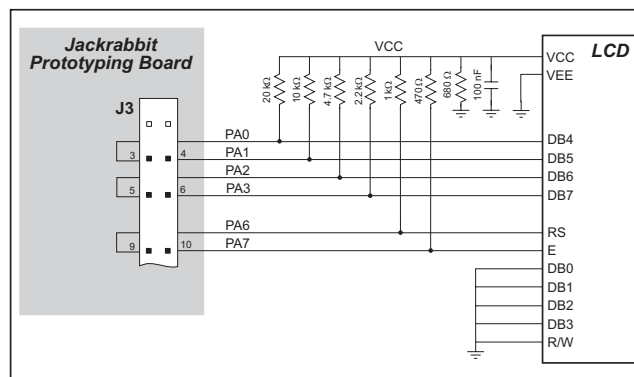
- `LCD_DEMO.C`—demonstrates a 4-bit interface to an LCD based on the HD44780 (or an equivalent) controller.

Connect the LCD to Parallel Port A.

PA0—LCD DB4
PA1—LCD DB5
PA2—LCD DB6
PA3—LCD DB7
PA6—LCD RS Register Select
(0 = command, 1 = data)
PA7—LCD E: normally low, latches on high to low transition

The R/W pin and DB0–DB3 on the LCD are grounded. DB0–DB3 are not used with a 4-bit interface.

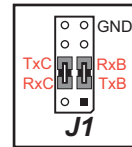
This sample program only involves writing, and that is why we ground the R/W pin. VEE is used to control brightness—the simplest thing to do is ground it. If you need to control the brightness, you can connect a potentiometer between ground and -5 V, with the arm going to the VEE pin. Check the specs on the LCD before doing this as some LCDs may require a different connection. VCC is +5 V and VSS is ground.



3.4.3 RS-232 Serial Communication Sample Programs

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

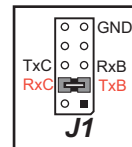
To set up the Prototyping Board, you will need to tie PC4 and PC5 (TxB and RxB) together at header J1, and you will also tie PC2 and PC3 (TxC and RxC) together as shown in the diagram.



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

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

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



To set up the Prototyping Board, you will need to tie PC4 and PC3 (TxB and RxC) together at header J1 as shown in the diagram.

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

3.4.4 RS-485 Serial Communication Sample Program

The following sample program illustrates the use of the RS-485 serial drivers. The sample program shows a byte being transmitted, and then the RS-485 transceiver waits for a reply.

```
#define DINBUFSIZE 15
#define DOUTBUFSIZE 15

void main( void ){
    int nEcho,nReply;
    char cChar;
    Jr485Init (); // Init RS485 Control (PD5)
    serDopen ( 9600 ); // Open Serial Port D
    for (;;) { // Forever
        for (cChar='a';cChar<='z';++cChar){
            // Send Alphabet
            Jr485Tx (); // Enable RS485 Transmitter
            serDputc ( cChar ); // Send Byte
            while ((nEcho = serDgetc ()) == -1);
            // Wait for Echo
            Jr485Rx (); // Disable RS485 Transmitter
            while ((nReply = serDgetc ()) == -1);
            // Wait for Reply
            printf ( "%02x -> %02x\n",nEcho,nReply );
        }
    }
}
```

NOTE: If your version of Dynamic C is earlier than 6.55, see Technical Note 117, *Jack-rabbit (BL1800 Series) RS-485 Bulletin*, for information on restrictions on using both DA1 and RS-485 at the same time.

3.5 Cooperative Multitasking

Cooperative multitasking is a convenient way to perform several different tasks at the same time. An example would be to step a machine through a sequence of steps and at the same time independently carry on a dialog with the operator via a human interface. Cooperative multitasking differs from a different approach called preemptive multitasking. Dynamic C supports both types of multitasking. In cooperative multitasking each separate task voluntarily surrenders its compute time when it does not need to perform any more activity immediately. In preemptive multitasking control is forcibly removed from the task via an interrupt.

Dynamic C has language extensions to support multitasking. The major C constructs are called *costatements*, *cofunctions*, and *slicing*. These are described more completely in the *Dynamic C User's Manual*. The example below, sample program **DEMOJR3.C**, uses costatements. A costatement is a way to perform a sequence of operations that involve pauses or waits for some external event to take place. A complete description of costatements is in the *Dynamic C User's Manual*. The **DEMOJR3.C** sample program has two independent tasks. The first task flashes LED DS4 once a second. The second task uses button S1 on the Prototyping Board to toggle the logical value of a virtual switch, **vswitch**, and flash DS1 each time the button is pressed. This task also debounces button S1.

```

int vswitch;          // state of virtual switch controlled by button
S1
main(){               // begin main program
                        // set up parallel port A as output
    WrPortI(SPCR,NULL,0x84);
    WrPortI(PADR,&PADRShadow,0xff);    // turn off all LEDs
    vswitch=0;                // initialize virtual switch off
(1) while (1) {            // Endless loop
    BigLoopTop();          // Begin a big endless loop

    // first task flash LED DS4 every second for 200 milliseconds

(2)    costate {           // begin a costatement
        BitWrPortI(PADR,&PADRShadow,0,3); // LED DS4 on
(3)    waitfor(DelayMs(200)); // light on for 200 ms
        BitWrPortI(PADR,&PADRShadow,1,3); // LED DS4 off
        waitfor(DelayMs(800)); // light off for 800 ms
(4)    }                  // end of costatement

    // second task - debounce switch #1 and toggle virtual switch vswitch

    // check button 1 and toggle vswitch on or off

    costate {
(5)    if(BitRdPortI(PBDR,2)) abort; // if button not down skip out
        waitfor(DelayMs(50)); // wait 50 ms
        if(BitRdPortI(PBDR,2)) abort; // if button not still down skip
out
        vswitch=!vswitch; // toggle virtual switch- button was down 50
ms
        while (1) { // wait for button to be off 200 ms
            waitfor(BitRdPortI(PBDR,2)); // wait for button to go up
            waitfor(DelayMs(200)); // wait for 200 milliseconds
            if(BitRdPortI(PBDR,2)) break; // if button up break
        } // end of while(1)
    } // end of costatement

    // make LED agree with vswitch if vswitch has changed

(6)    if( (PADRShadow & 1) == vswitch) {
        BitWrPortI(PADR,&PADRShadow,!vswitch,0);
    }
(7) } // end of while loop, go back to start
    } // end of main, never come here

```

The numbers in the left margin are reference indicators and are not a part of the code. Load and run the program. Note that LED DS4 flashes once per second. Push button S1 several times and note how LED DS1 is toggled.

The flashing of LED DS4 is performed by the costatement starting at the line marked (2). Costatements need to be executed regularly, often at least every 25 ms. To accomplish this, the costatements are enclosed in a *while* loop. The term while loop is used as a handy way to describe a style of real-time programming in which most operations are done in one loop. The while loop starts at (1) and ends at (7). The function **BigLoopTop()** is

used to collect some operations that are helpful to do once on every pass through the loop. Place the cursor on this function name **BigLoopTop()** and hit **<Ctrl-H>** to learn more.

The statement at (3) waits for a time delay, in this case 200 ms. The costatement is being executed on each pass through the big loop. When a **waitfor** condition is encountered the first time, the current value of **MS_TIMER** is saved and then on each subsequent pass the saved value is compared to the current value. If a **waitfor** condition is not encountered, then a jump is made to the end of the costatement (4), and on the next pass of the loop, when the execution thread reaches the beginning of the costatement, execution passes directly to the **waitfor** statement. Once 200 ms has passed, the statement after the **waitfor** is executed. The costatement has the property that it can wait for long periods of time, but not use a lot of execution time. Each costatement is a little program with its own statement pointer that advances in response to conditions. On each pass through the big loop, as little as one statement in the costatement is executed, starting at the current position of the costatement's statement pointer. Consult the *Dynamic C User's Manual* for more details.

The second costatement in the program debounces the switch and maintains the variable **vswitch**. Debouncing is performed by making sure that the switch is either on or off for a long enough period of time to ensure that high-frequency electrical hash generated when the switch contacts open or close does not affect the state of the switch. The **abort** statement is illustrated at (5). If executed, the internal statement pointer is set back to the first statement within the costatement, and a jump to the closing brace of the costatement is made.

At (6) a use for a shadow register is illustrated. A shadow register is used to keep track of the contents of an I/O port that is write only - it can't be read back. If every time a write is made to the port the same bits are set in the shadow register, then the shadow register has the same data as the port register. In this case a test is made to see the state of the LED and make it agree with the state of **vswitch**. This test is not strictly necessary, the output register could be set every time to agree with **vswitch**, but it is placed here to illustrate the concept of a shadow register.

To illustrate the use of snooping, use the watch window to observe **vswitch** while the program is running. Add the variable **vswitch** to the list of watch expressions. Then toggle **vswitch** and the LED. Then type **<Ctrl-U>** to observe **vswitch** again.

3.5.1 Advantages of Cooperative Multitasking

Cooperative multitasking, as implemented with language extensions, has the advantage of being intuitive. Unlike preemptive multitasking, variables can be shared between different tasks without having to take elaborate precautions. Sharing variables between tasks is the greatest cause of bugs in programs that use preemptive multitasking. It might seem that the biggest problem would be response time because of the big loop time becoming long as the program grows. Our solution for that is a device called slicing that is further described in the *Dynamic C User's Manual*.

3.6 Jackrabbit Function Calls

3.6.1 I/O Drivers

The Jackrabbit board contains four high-power digital output channels, two D/A converter output channels, and one A/D converter input channel. These I/O channels can be accessed using the functions found in the `JRIO.LIB` library.

3.6.1.1 Initialization

The function `jrIoInit()` must be called before any other function from the `JRIO.LIB` library. This function initializes the digital outputs and sets up the driver for the analog input/outputs. The digital outputs correspond to the Rabbit processor's port E bits 0–3, and the analog I/O uses timer B; bits 1, 2, and 4 of port D; and bits 6 and 7 of port E.

The function void `jrIoInit()` initializes the I/O drivers for Jackrabbit. In particular, it sets up parallel port D bits 1, 2, and 4 for analog output, port E bits 0–3 for digital output, and starts up the pulse-width modulation routines for the A/D and D/A channels. Note that these routines can consume up to 20% of the CPU's processing power; the routines use timer B and the B1 and B2 match registers.

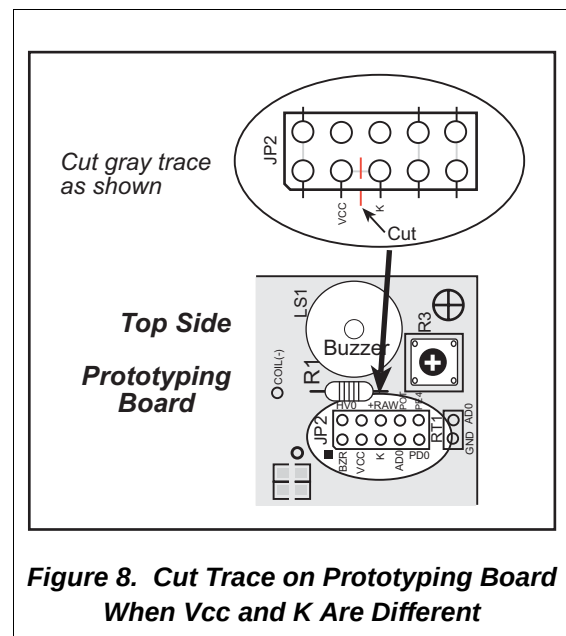
3.6.1.2 Digital Output

The Jackrabbit board contains four high-power digital output drivers, HV0–HV3, on header J4. These can be turned on and off with the following functions from the library `JRIO.LIB`.

HV0, HV1, and HV2 are open-collector sinking outputs, and are able to sink up to 1 A (200 mA for the BL1810 and BL1820) from a 30 V source connected to the K line on header J4. HV3 is a sourcing output that is able to source up to 500 mA (100 mA for the BL1810 and BL1820) from a 30 V source connected to the K line.

CAUTION: Remember to cut the trace between K and Vcc inside the outline for header JP2 on the top side of the Prototyping Board if you are supplying K from a separate power supply. An exacto knife, a precision grinder tool, or a screwdriver may be used to cut through the traces as shown in Figure 8.

NOTE: Failure to do this could lead to the destruction of the Rabbit 2000 microprocessor and other components once the Jackrabbit is connected to the Prototyping Board.



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

sets the state of a digital output bit.

jrioInit must be called first.

channel is the output channel number (0-3 on the Jackrabbit).

value is the output value (0 or 1).

```
void digOn(int channel);
```

sets the state of a digital output bit to on (1).

jrioInit must be called first.

channel is the output channel number (0-3 on the Jackrabbit).

```
void digOff(int channel);
```

sets the state of a digital output bit to off (0).

jrioInit must be called first.

channel is the output channel number (0-3 on the Jackrabbit).

NOTE: See the sample program **JRIOTEST.C** for an example of using the digital output functions.

3.6.1.3 Analog Output

The two analog output channels on the Jackrabbit (DA0 and DA1 on header J5) are controlled by a pulse-width modulation (PWM) driver. This requires the use of some fraction of the CPU cycles when the driver is running (up to 20% when both D/A channels are used). A voltage is selected by giving a value from 0 to 1024 to the driver, corresponding roughly to 0.1 V to 3.5 V on DA0. Because of the PWM interrupt frequency, the PWM driver can provide a continuous range of voltage output in the range from 0.1 V to 3.0 V for DA0, and 0.6 V to 3.6 V for DA1. These ranges can be specified with the constants **PWM_MIN**, **PWM_MAX0**, and **PWM_MAX1**. In other words, setting channel DA0 to the value **PWM_MIN** will output 0.1 V, and setting it to **PWM_MAX0** will output 3.0 V. Similarly, setting DA1 to **PWM_MIN** will output 0.6 V, and setting it to **PWM_MAX1** will output 3.6 V. Values below **PWM_MIN** will be rounded down to 0, and values above **PWM_MAX0** (**PWM_MAX1** for DA1) will be rounded up to 1024.

The output channels can also be set in an “always on” or “always off” mode, which does not require CPU cycles. The “always on” mode is set by requesting an output value of 1024, and will provide about 3.4 V on channel DA0, and 3.6 V on DA1. The “always off” mode is selected by asking for a value of 0, and provides an output of around 0.1 V on DA0 and 0.0 V on DA1.

Table 4 provides a summary of the possible analog output voltages corresponding to values given in the `anaOut` function.

Table 4. Typical Analog Output Voltages Corresponding to Values in `anaOut` Function

Channel	0	PWM_MIN	PWM_MAX	1024
DA0	0.08 V	0.08 V	2.875 V	3.4 V
DA1	0.004 V	0.63 V	3.6 V	3.6 V

The output value is set using the following function.

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

sets the state of an analog output channel.

`jrioInit` must be called first.

`channel` is the output channel number (0 or 1 on the Jackrabbit).

`value` is an integer from 0–1024 that corresponds to an output voltage as shown in Table 4.

NOTE: See the sample program `JRIOTEST.C` for examples of using the `anaOut` function.

Effect of Interrupts on Analog I/O

The stability of the voltage output (and hence the voltage input determination as well) depends on the ability of the driver to respond quickly to interrupt requests. Dynamic C debugging, use of the `printf` function, or any serial communications can disrupt the pulse-width modulation utilized by the driver and cause fluctuations in the voltage outputs. Avoid using serial communications or `printf` statements during portions of your program where the voltage must remain steady. Also be aware that debugging and running Dynamic C in polling mode will cause fluctuations. Finally, be certain to disable the PWM drivers by setting the output values to 0 or 1024 when you are done using them to free up the CPU.

Calibration of Values to Voltages

The analog output channels on the Jackrabbit board can be more accurately calibrated for each individual Jackrabbit board in the following manner (calibration of DA0 is assumed in this example, calibration of DA1 would proceed similarly):

- Set desired channel output to `PWM_MIN`.
- Measure voltage V_{min} on DA0.
- Set desired channel output to `PWM_MAX`.
- Measure voltage V_{max} on DA0.
- A linear relation between input value and voltage can now be calculated:

$$m = \frac{V_{max} - V_{min}}{PWM_MAX0 - PWM_MIN}$$

$$b = V_{max} - m \times PWM_MAX0$$

$$\text{voltage} = m \times \text{value} + b$$

3.6.1.4 Analog Input

The analog input channel on the Jackrabbit (AD0 on header J5) works by varying analog output channel DA0 until its voltage matches the input voltage on AD0. DA0 obviously cannot be used while an input voltage is being measured, although channel DA0 is still available. The value returned corresponds to the value that DA0 required to match the input voltage (you would call `anaOut(0,value)` for DA0 to provide that same voltage). If the value returned is negative, then the function considers the value suspect for some reason (most likely a failure of the DA0 voltage to settle quickly). The value can be taken as is, or another measurement can be done.

```
void anaIn(int channel, int *value);
```

Analog input for the Jackrabbit analog input channel (AD0).

`jrioInit` must be called first.

`channel` is the input channel number (0 only on the Jackrabbit).

An integer between 0 and 1024 will be returned in `value`, corresponding to a voltage obtained if output channel DA0 was set to that value. If a value is found, but the voltage has not appeared to fully settle, the value will be negative (but equal in magnitude to the found voltage) to allow remeasurement if desired.

NOTE: See sample program `JRIOTEST.C` for an example of the use of `anaIn`.

Two versions of the analog input function are available: the standard function, listed above, that does not return until the measurement has been made, and a cofunction version that can be called from within a costatement. This cofunction version allows other tasks to be performed while the voltage match is being made. The voltage measurement will take ten calls of the cofunction version to make a measurement.

```
void cof_anaIn(int channel, int *value);
```

The parameters are identical to those described above for `anaIn`.

NOTE: See sample program `JRIO_COF.C` for an example of the use of `cof_anaIn`.

3.6.2 Serial Communication Drivers

Library files included with Dynamic C provide a full range of serial communications support. The **RS232.LIB** library provides a set of circular-buffer-based serial functions. The **PACKET.LIB** library provides packet-based serial functions where packets can be delimited by the 9th bit, by transmission gaps, or with user-defined special characters. Both libraries provide blocking functions, which do not return until they are finished transmitting or receiving, and nonblocking functions, which must be called repeatedly until they are finished. For more information, see the *Dynamic C Function Reference Manual* and Technical Note 213, *Rabbit 2000 Serial Port Software*.

3.6.2.1 RS-485 Serial Communication Drivers

The **JR485.LIB** library in the Dynamic C **LIB/JRABLIB** directory contains three RS-485 drivers for use with the Jackrabbit. These drivers are used with the drivers for Serial Port D in the **RS232.LIB** library because **serDopen** uses PC0 (TXD) and PC1 (RXD), which are connected to pin 4 and pin 1 of the SP483EN RS-485 chip at U6. This chip is half duplex, requiring pin 3 (Data Enable) to be high for pins 6 and 7 to act as outputs, and low for those pins to act as inputs.

Parallel Ports D and E on the Rabbit 2000 are double-buffered to provide precisely timed updating of the output pins. Each port is divided into an upper and a lower nibble. All bits of each nibble must be updated simultaneously. Each nibble may be updated constantly at a rate of **perclk/2** or on a match of a selected timer (Timer A1, B1, or B2).

The bits used to select the update rate for each nibble are left random at power-up. If a mode other than **perclk/2** is selected, the bits of a particular port will not update on a simple writing to the port's data register. In particular, PD5, the RS-485 transmitter control, will not set the RS-485 transmitter enable unless the upper nibble of Port D is configured properly.

The **JR485Init** function in Dynamic C release 6.16 has provision to disable the special clocking features associated with the high nibble of Port D. This effectively disables digital-to-analog (D/A) converter output channel DA1, the low-resolution D/A converter channel, which also uses PD4. Channel DA0 has its PWM output clocked separately with the low nibble, and so is not affected. Because the analog-to-digital converter uses D/A channel DA0, analog-to-digital conversion is not affected.

There are three RS-485 serial drivers.

```
void Jr485Init();
```

Sets up parallel port D pins for RS-485 use.

```
void Jr485Tx();
```

Sets pin 3 (DE) of the SP483EN chip high to disable Rx and enable Tx.

```
void Jr485Rx();
```

Resets pin 3 (DE) of the SP483EN chip low to disable Tx and enable Rx.

3.7 Upgrading Dynamic C

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

- www.zworld.com/support/

or

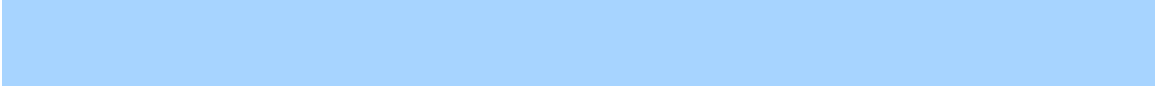
- www.rabbitsemiconductor.com/support/

for the latest patches, workarounds, and bug fixes.

3.7.1 Add-On Modules

Dynamic C installations are designed for use with the board they are included with, and are included at no charge as part of our low-cost kits. Z-World offers add-on Dynamic C modules for purchase, including the popular μ C/OS-II real-time operating system, as well as PPP, Advanced Encryption Standard (AES), and other select libraries.

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



NOTICE TO USERS

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

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

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

INDEX

A

additional information
 online documentation 5
 references 4

C

C language 19
 CE compliance 6
 design guidelines 7

D

Development Kit 9
 Dynamic C 4, 19
 installing 21
 standard features 20
 debugging 20
 telephone-based technical
 support 40
 upgrades and patches 40
 USB port settings 16

F

features
 Jackrabbit 1
 Prototyping Board 10, 11

H

hardware connections 12
 power supply 15
 programming cable 14
 hardware reset 15

J

Jackrabbit
 features 1
 models 2

M

models 2

P

Program Mode 22
 programming cable
 Jackrabbit connections 14
 setting modes 22
 Prototyping Board 10
 expansion area 11
 features 10, 11

R

reset 15
 Run Mode 22

S

sample programs
 DEMOJR1.C 23, 24
 DEMOJR2.C 23, 28
 DEMOJR3.C 23, 28, 32
 Dynamic C
 break point 25
 cooperative multitasking 32
 editing 26
 single-stepping 25
 watch expression 25
 watching variables dynami-
 cally 26
 JRIO_COF.C 23, 28
 JRIO_TEST.C 23, 28
 LCD_DEMO.C 23, 29
 PONG.C 16
 RABDB01.C 23, 28
 RABDB02.C 23, 28
 RS-232 serial communication
 JR_FLOWCONTROL.C
 23, 30
 JR_PARITY.C 23, 30
 RS-485 serial communica-
 tion 31

software

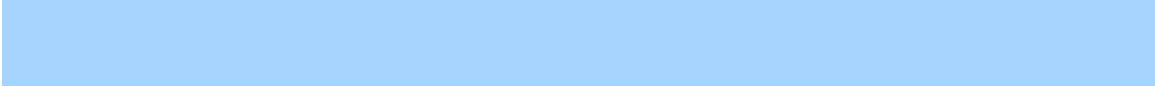
analog input 38
 anaIn 38
 cof_anaIn 38
 analog output 36
 anaOut 37
 board initialization 35
 jrioInit 35
 digital output 35
 digOff 36
 digOn 36
 digOut 36
 I/O drivers 35
 libraries
 JRIO.LIB 35
 PACKET.LIB 39
 RS232.LIB 39
 RS-232 serial communica-
 tion 39
 RS-485 serial communica-
 tion 39
 serial communication
 Jr485Init 39
 Jr485Rx 39
 Jr485Tx 39
 specifications
 physical and electrical 3

T

technical support 17

U

USB/serial port converter 14
 Dynamic C settings 16



SCHEMATICS

090-0092 Jackrabbit Schematic

www.zworld.com/documentation/schemat/090-0092.pdf

090-0088 Jackrabbit Prototyping Board Schematic

www.zworld.com/documentation/schemat/090-0088.pdf

090-0128 Programming Cable Schematic

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

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

