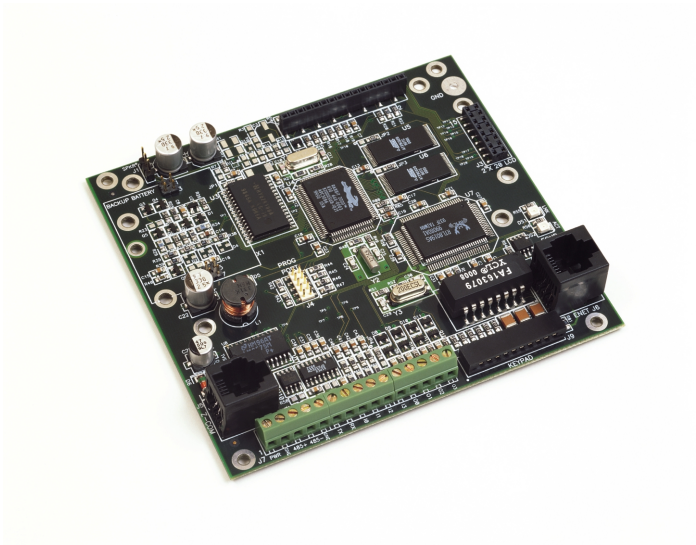


Getting Started

Integrated C Development System

TCP/IP Development Kit

000915 - B

Getting Started (TCP/IP Development Kit)

Part Number 019-0079 • 000915 - B • Printed in U.S.A.

Copyright

Trademarks

- Dynamic C® is a registered trademark of Z-World, Inc.
- Windows® is a registered trademark of Microsoft Corporation

Notice to Users

When a system failure may cause serious consequences, protecting life and property against such consequences with a backup system or safety device is essential. The buyer agrees that protection against consequences resulting from system failure is the buyer's responsibility.

This device is not approved for life-support or medical systems.

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. Rabbit Semiconductor 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.

Company Address

Rabbit Semiconductor
2932 Spafford Street
Davis, California 95616-6800
USA

Telephone: (530) 757-8400

Facsimile: (530) 757-8402

Web site: http://www.rabbitsemiconductor.com

Table of Contents

About This Manual

Rabbit Semiconductor and Z-World customers develop software for their programmable controllers using Z-World's Dynamic C Premier development system running on an IBM-compatible PC. Dynamic C Premier provides an interactive compiler, editor, and source-level debugger. The controller is connected to a COM port on the PC (COM1 by default) whose default operation is at 115,200 bps.

This manual introduces the Dynamic C Premier SE development system to write software for a TCP/IP board based on the Rabbit microprocessor. The Rabbit 2000 microprocessor is a new high-performance 8-bit microprocessor developed by Rabbit Semiconductor, a company affiliated with Z-World. The Rabbit 2000 can handle C language applications of approximately 1 megabyte (50,000+ C statements).

To view documentation included on the CD-ROM, close your browser (if already open), then click on the "TCP IP DEV KIT DOCS" icon on your desktop. This will bring up a list of all documentation contained on the CD-ROM. Refer to the Rabbit Semiconductor Web site, www.rabbitsemiconductor.com, for updates to manuals and application notes.

Conventions

Table 1 lists and defines the typographic conventions that may be encountered in Dynamic C.

Table 1: Typographic Conventions

Example	Description
`while`	Bold Courier font indicates a program, a fragment of a program, or a Dynamic C keyword or phrase.
`// IN-01…`	Program comments are in normal Courier font.
`Italics`	Courier italics indicate that something should be typed instead of the italicized words (e.g., type a file name where *`filename`* is shown).
Edit	Bold sans serif font indicates a menu or menu selection.
…	An ellipsis indicates that (1) irrelevant program text is omitted for brevity, or that (2) the preceding program text may be repeated indefinitely.
`[ ]`	Square brackets in a C function's definition or program segment indicate that the enclosed directive is optional.
`< >`	Angle brackets are used to enclose classes of terms.
a | b | c	A vertical bar indicates that a choice should be made from among the items listed.

About the TCP/IP Development Kit

The TCP/IP Development Kit is a more advanced kit than other Rabbit Semiconductor development kits that demonstrate the use of the Rabbit 2000 microprocessor. You should have some knowledge of the C programming language and microprocessor-based systems. If you don't already have a working knowledge of how to use Dynamic C Premier, then you should learn this first by doing the exercises in the ***TCP/IP Development Kit Getting Started*** manual. Once you are comfortable with Dynamic C Premier, you can begin trying out other TCP/IP sample programs.

If you are a real novice you might consider starting out with one of the easier Rabbit Semiconductor kits.

1. INSTALLING DYNAMIC C PREMIER

1.1 Requirements

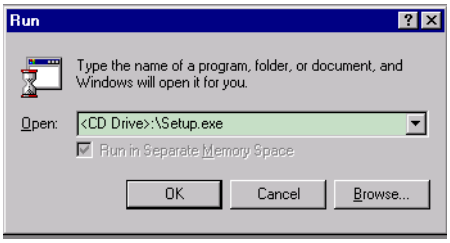
Dynamic C software comes on CD. To install Dynamic C, your system must be running one of the following.

- Windows 95
- Windows 98
- Windows NT
- Windows 2000
- Windows Me

Your PC should have at least one free COM port.

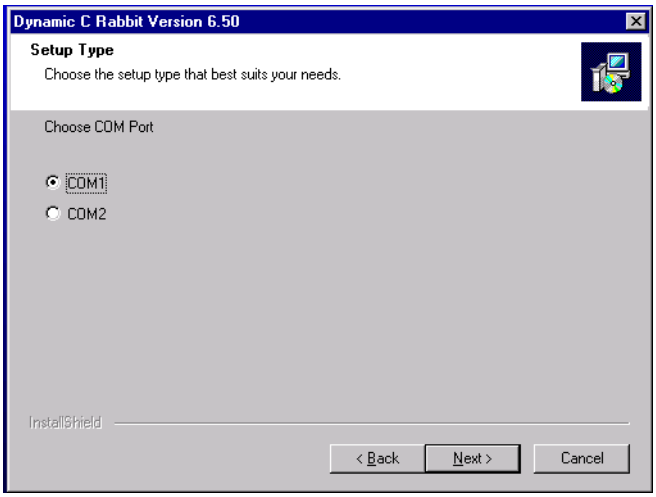
1.2 Installation

Insert the CD in the CD-ROM disk drive on your PC. As long as auto-install is enabled, the CD installation will begin automatically. If not, issue the Windows **Start** > **Run...** command and type the following, using your CD drive for **<CD Drive>**.



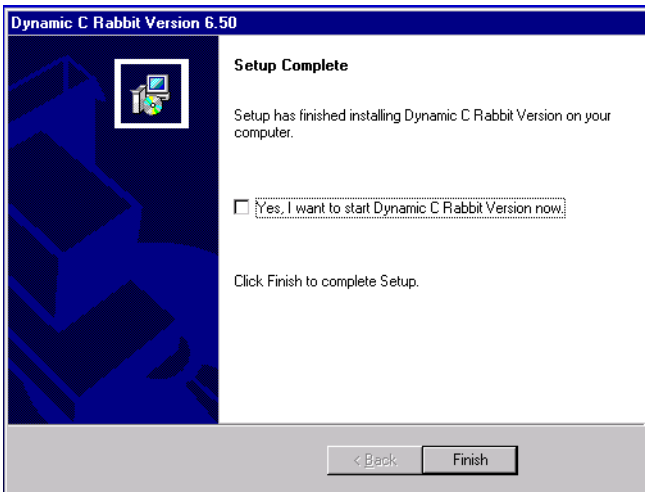
The installation program will then guide you through the installation process.

When selecting the PC COM port, the default PC COM port is COM1.



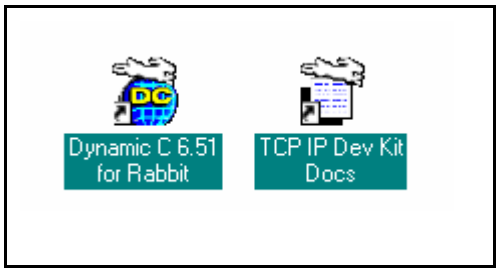
Before the installation is complete, the installation wizard will ask you what icons to display on your PC desktop. Separate icons are available for Dynamic C itself and for the manuals and other documents.

Click the **Finish** button to end the installation. Notice that there is a check mark option to start Dynamic C immediately once the installation is complete.



1.3 Desktop Icons

Once your installation of Dynamic C and the documentation is complete, you will have two icons on your PC desktop: one for Dynamic C and one for the documentation. Double-click the corresponding icon start Dynamic C or to access the documentation.



It is also possible to start Dynamic C or access the documentation by double-clicking the corresponding launch file on the drive where you installed Dynamic C and the documentation. The default file locations for a typical installation are shown.

- **C:\DCRABBIT_6...exe** to start Dynamic C
- **C:\DCRABBIT_6...\Docs\default** to display the documentation screen.

2. INTRODUCTION TO DYNAMIC C

Dynamic C is an integrated development system for writing embedded software. It runs on an IBM-compatible PC and is designed for use with Z-World controllers and other controllers based on the Rabbit microprocessor.

2.1 The Nature of Dynamic C

Dynamic C integrates the following development functions

- Editing
- Compiling
- Linking
- Loading
- Debugging

into one program. In fact, compiling, linking and loading are one function. Dynamic C has an easy-to-use built-in text editor. Programs can be executed and debugged interactively at the source-code or machine-code level. Pull-down menus and keyboard shortcuts for most commands make Dynamic C easy to use.

Dynamic C also supports assembly language programming. It is not necessary to leave C or the development system to write assembly language code. C and assembly language may be mixed together.

Debugging under Dynamic C includes the ability to use **`printf`** commands, watch expressions, breakpoints and other advanced debugging features. Watch expressions can be used to compute C expressions involving the target's program variables or functions. Watch expressions can be evaluated while stopped at a breakpoint or while the target is running its program.

Dynamic C provides extensions to the C language (such as *shared and protected* variables, costatements and cofunctions) that support real-world embedded system development. Interrupt service routines may be written in C. Dynamic C supports cooperative and preemptive multi-tasking.

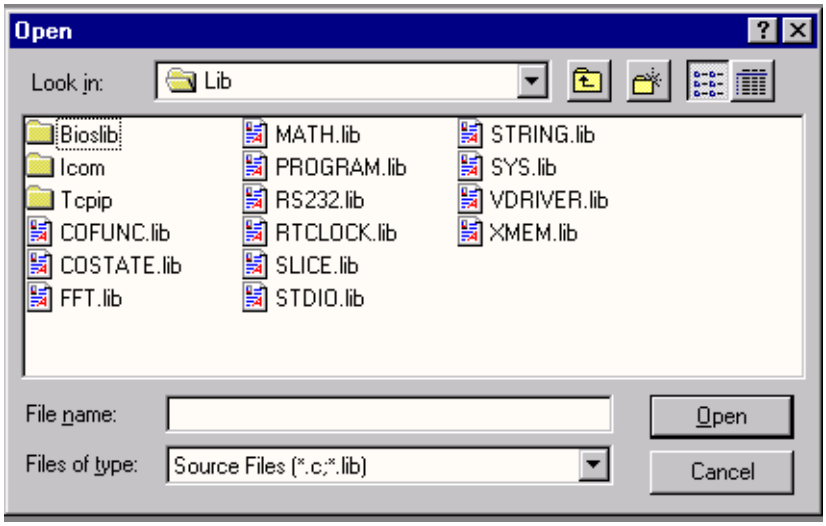
Dynamic C comes with many function libraries, all in source code. These libraries support real-time programming, machine level I/O, and provide standard string and math functions.

2.1.1 Speed

Dynamic C compiles directly to memory. Functions and libraries are compiled and linked and downloaded on-the-fly. On a fast PC, Dynamic C might load 30,000 bytes of code in 5 seconds at a baud rate of 115,200 bps.

2.2 Dynamic C Libraries

With Dynamic C running, click **File** > **Open**, and select **Lib**. The following list of Dynamic C libraries will be displayed.

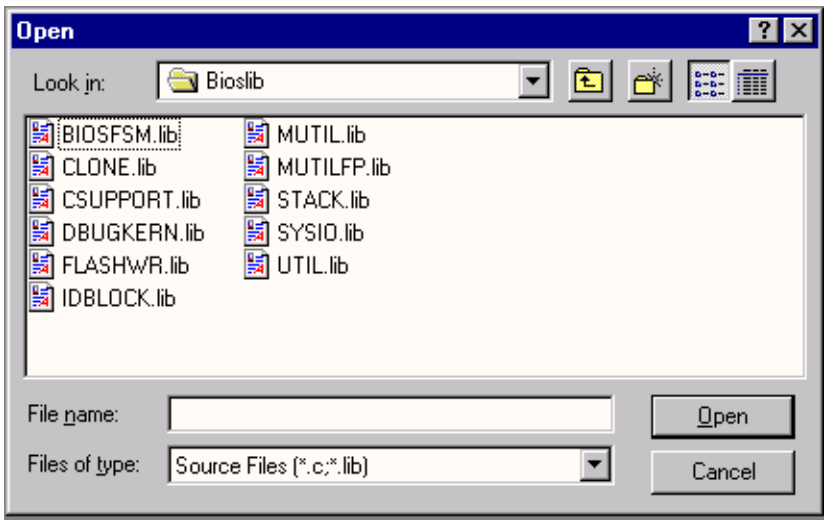


Let's examine the libraries.

- **Bioslib**—libraries specific to running a BIOS, applies to all controllers. Although the functions in these libraries are required by the BIOS, they are not exclusive to the BIOS.
- **Icom**—libraries specific to the TCP/IP Development Board.
- **Tcpip**—libraries specific to using TCP/IP with the TCP/IP Development Board.
- **COFUNC.lib**—enables multitasking cofunctions to be defined starting with **cofunc**. Cofunctions may be nested within costatements.
- **COSTATE.lib**—enables multitasking costatements to be defined starting with **costate**. Also contains a library of commonly used costatements.
- **FFT.lib**—fast Fourier transform functions.
- **MATH.lib**—math functions.
- **PROGRAM.lib**—does program initialization before calling **main**.
- **RS232.lib**—interface designed to provide users with a set of functions that send and receive data without yielding to other tasks, and a set of single-user cofunctions that send and receive data but yield to other tasks.
- **RTCLOCK.lib**—real-time clock drivers.
- **SLICE.lib**—library functions that allow multitasking.
- **STDIO.lib**—standard Dynamic C terminal window I/O functions.
- **STRING.lib**—string operations.

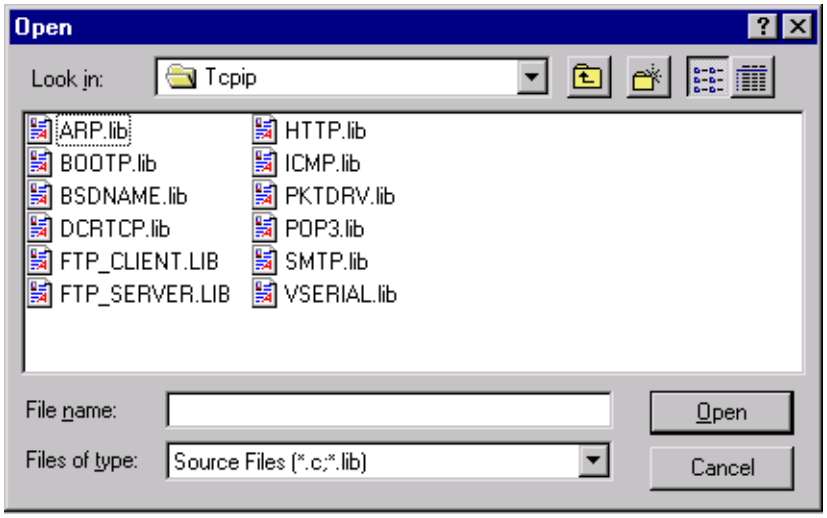
- **SYS.lib**—support libraries.
- **VDRIVER.lib**—generic virtual drivers.
- **XMEM.lib**—extended memory support functions.

The **Bioslib** folder contains libraries required by the BIOS, but not exclusive to the BIOS.



- **BIOSFSM.lib**—support libraries.
- **CLONE.lib**—functions used to “clone” boards by copying BIOS and programs from one board to another via a special cloning cable.
- **CSUPPORT.lib**—support libraries.
- **DBUGKERN.lib**—debugging kernel support functions.
- **FLASHWR.lib**—utility functions for writing to flash EPROM.
- **IDBLOCK.lib**—functions to access the ID block in Z-World product flash devices, also contains general CRC checking functions.
- **MUTIL.lib**—integer math utility functions.
- **MUTILFP.lib**—floating-point math utility functions.
- **STACK.lib**—base data structure for maintaining stack allocation information.
- **SYSIO.lib**—support libraries.
- **UTIL.lib**—utility functions.

The **Tcpip** folder contains libraries specific to using TCP/IP with the TCP/IP Development Board.



- **ARP.lib**—address resolution protocol functions.
- **BOOTP.lib**—bootstrap protocol functions.
- **BSDNAME.lib**—BSD-style socket routines.
- **DCRTCP.lib**—TCP/IP functions.
- **FTP_CLIENT.LIB**—FTP client functions.
- **FTP_SERVER.LIB**—FTP server functions.
- **HTTP.lib**—HTTP handler.
- **ICMP.lib**—ICMP handler.
- **PKTDRV.lib**—packet driver functions.
- **POP3.lib**—POP3 functions.
- **SMTP.lib**—SMTP handler.
- **VSERIAL.lib**—virtual Telnet functions.

2.3 Using Dynamic C

More complete information on Dynamic C is provided in the ***Dynamic C Premier User's Manual***. Functions specific to the TCP/IP Development Board are described in the ***TCP/IP Application Frameworks Manual*** and the ***TCP/IP Function Reference Manual***. ***An Introduction to TCP/IP*** provides background information on TCP/IP, and is included on the CD.

2.4 Upgrading Dynamic C

Dynamic C upgrades and patches are available from time to time. An upgrade may either enhance the features and libraries, or it may focus on bug fixes. Check the Web sites

www.zworld.com/support/supportcenter.html

or

www.rabbitsemiconductor.com

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

2.4.1 Workarounds

Workarounds describe problems and recommended ways around them. The figure below shows a typical workaround panel from one of the two Web sites.

Reference Number	Description	Work-Around	Version(s) Affected
3	When a jp instruction in embedded assembly code is the next to be executed, the DC debugger doesn't highlight the jp instruction in the source window.	Look at the assembly window when single stepping through assembly code.	6.0x, 6.1x
6	Run-time math exceptions in watch expressions cause the target program to crash when debugging.	None at this time. Avoid evaluating floating point watch expressions with bad domain arguments.	6.0x, 6.1x
12	Behavior similar to bug #3 with call instructions.	Look at the assembly window when single stepping through assembly code.	6.0x, 6.1x

17	Compiler doesn't catch all attempted uses of void functions in expressions.
	e.g. the follow program compiles successfully but shouldn't:
	void x(){}
	main(){
	char y;
	y = y + x();
	}
		None at this time.	6.0x, 6.1x

2.4.2 Upgrades

Upgrades are also available on the Web site, and are first downloaded to your PC. The downloaded application is then run, much like an installation would be.

The default installation of an upgrade is to install the new release of Dynamic C in a directory (folder) different from that of the original installation. Z-World recommends using a different directory so that you can verify the operation of the new release without overwriting the previous release. If you have made any changes to the BIOS or to libraries, or if you have programs in the old directory (folder), make these same changes to the BIOS or libraries in the new directory containing the upgraded release of Dynamic C. Do ***not*** simply copy over an entire file since you may overwrite a bug fix; of course, you may copy over any programs you have written. Once you are sure the new release works entirely to your satisfaction, you may retire the older release, but keep it available to handle legacy applications.

3. HARDWARE CONNECTIONS

3.1 Connections

Before proceeding you will need to have the following items.

- If you don't have Ethernet access, you will need at least a 10BaseT 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 Ethernet hub are available from Rabbit Semiconductor in a TCP/IP tool kit. More information is available at www.rabbitsemiconductor.com.

An AC adapter is ***not*** included with TCP/IP Development Kits sold outside North America. Rabbit Semiconductor recommends that an AC adpater with a minimum rating of 200 mA at 12 VDC or 100 mA at 24 VDC be used to supply power to the TCP/IP Development Board.

1. Attach the rubber feet to the bottom corners of the TCP/IP Development Board.

2. Connect the Programming Cable to the TCP/IP Development Board.

Turn the Rabbit 2000 TCP/IP Development Board so that the Rabbit 2000 microprocessor is facing as shown below. Connect the 10-pin connector of the programming cable to header J4 on the TCP/IP Development Board as shown below. Connect the other end of the programming cable to a COM port on your PC. Note that COM1 on the PC is the default COM port used by Dynamic C.

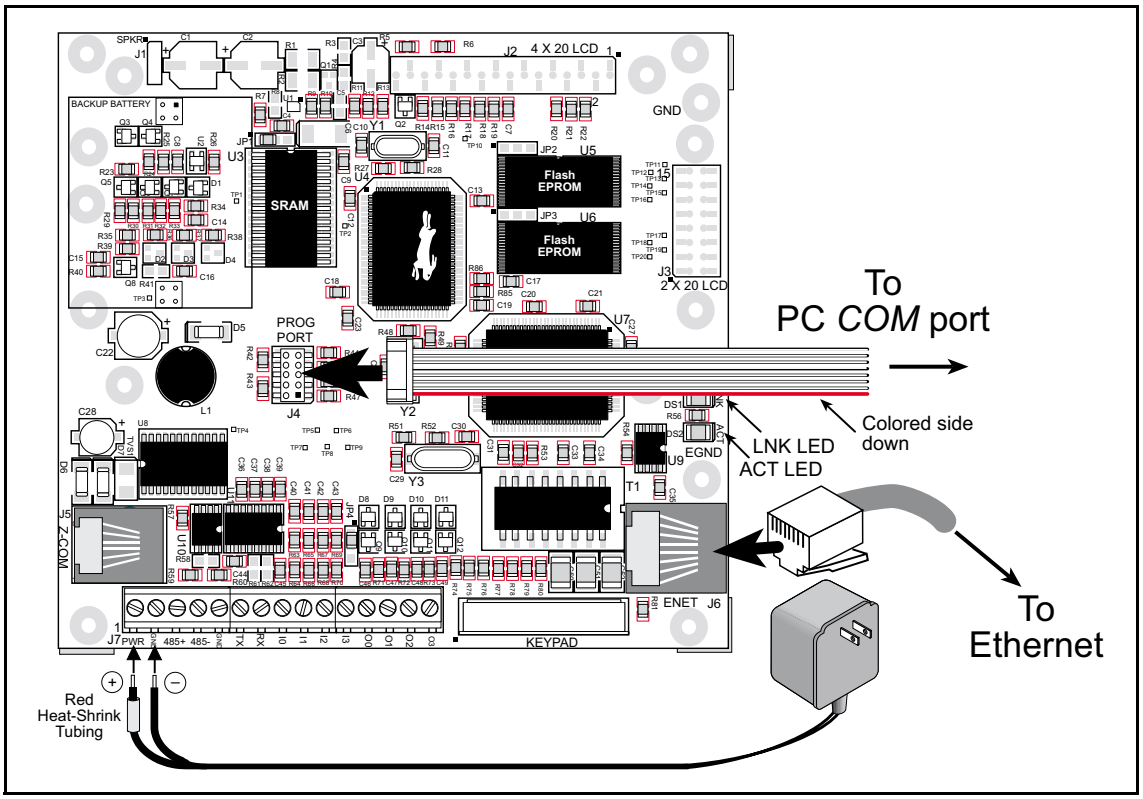


Figure 1. Connecting Power and PC to Rabbit 2000 TCP/IP Development Board

3. Connect Power Supply to TCP/IP Development Board

Connect the positive lead (indicated with red heat-shrink tubing on the AC adapter included with the Development Kit) to the PWR connector on header J7 on the TCP/IP Development Board, and connect the negative lead to GND on header J7 as shown in Figure 2.

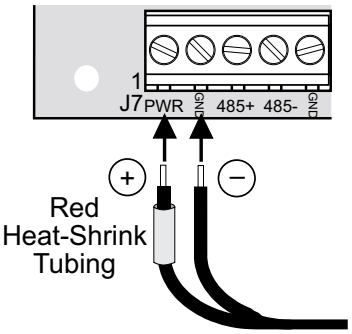


Figure 2. Power Supply Connections

Be careful to hook up the positive and negative power leads ***exactly*** as described. Otherwise, the TCP/IP Development Board will not function.

4. Ethernet Connections

If you do not have access to an Ethernet network, use a crossover Ethernet cable to connect the TCP/IP Development Board to a PC with at least a 10BaseT Ethernet card.

If you have Ethernet access, use a straight Ethernet cable to establish an Ethernet connection to the TCP/IP Development Board from an Ethernet hub. These connections are shown in Figure 3.

The PC running Dynamic C through the serial port need not be the same as the PC with the Ethernet card.

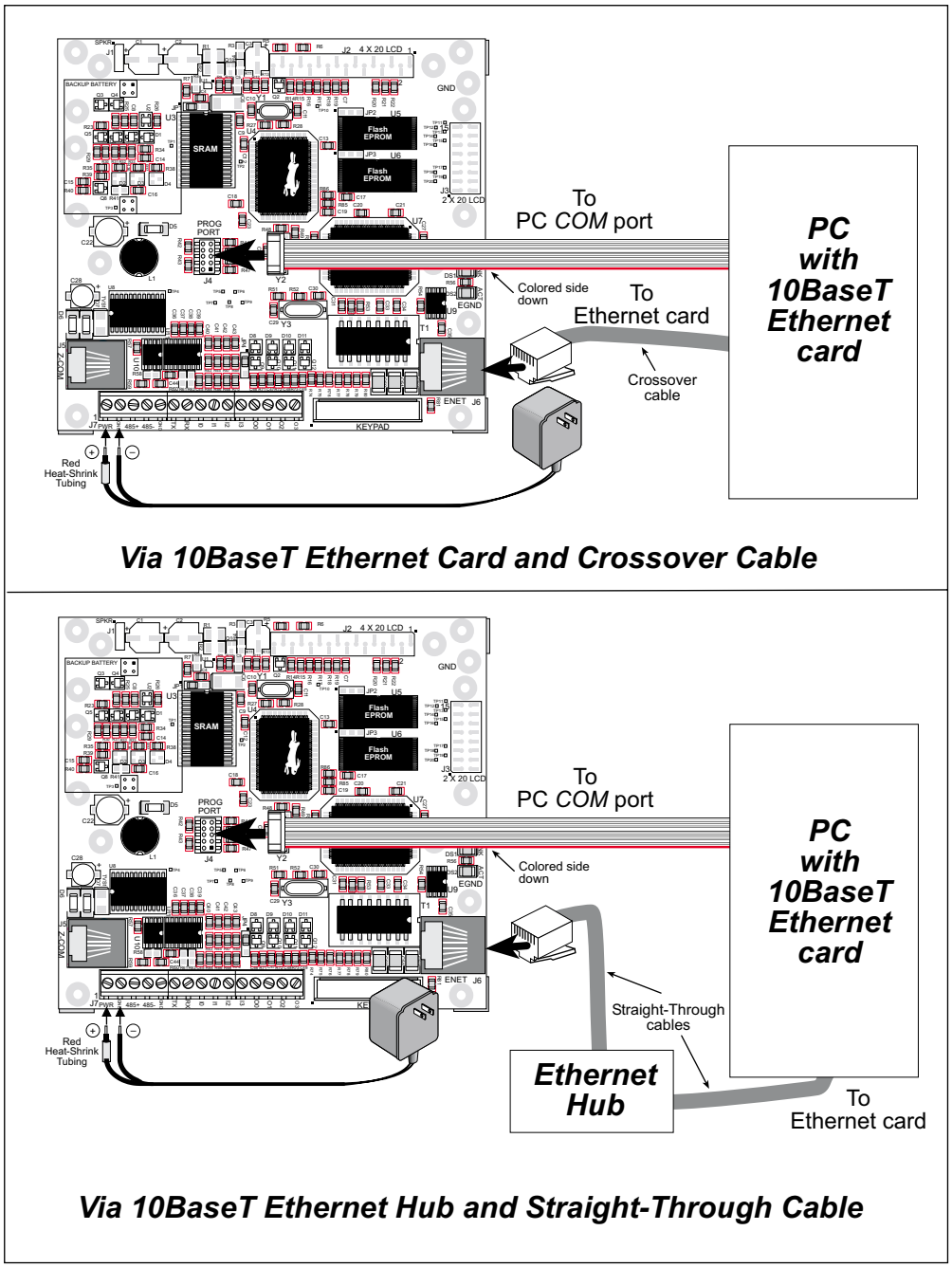


Figure 3. Ethernet Connections

5. Apply Power

Plug in the wall transformer. The TCP/IP Development Board is now ready to be used.

A hardware RESET is accomplished by unplugging the wall transformer, then plugging it back in.

When working with the TCP/IP Development Board, the green LNK light is on when a program is running and the board is properly connected either to an Ethernet hub or to an active Ethernet card. The red ACT light flashes each time a packet is received.

3.2 Installing Dynamic C

If you have not yet installed Dynamic C, you may do so by inserting the CD from the Development Kit in your PC's CD-ROM drive. The CD will auto-install unless you have disabled auto-install on your PC.

Chapter 1 provides detailed instructions for the installation of Dynamic C and any future upgrades.

3.3 Starting Dynamic C

Once the Rabbit 2000 TCP/IP Development Board is connected as described above, start Dynamic C by double-clicking on the Dynamic C icon or by double-clicking on the **.exe** file associated with **DcRab** in the Dynamic C directory.

Dynamic C assumes, by default, that you are using serial port COM1 on your PC. If you are using COM1, then Dynamic C should detect the TCP/IP Development Board and go through a sequence of steps to cold-boot the TCP/IP Development Board and to compile the BIOS. If the error message "Rabbit Processor Not Detected" appears, you have probably connected to a different PC serial port such as COM2, COM3, or COM4. You can change the serial port used by Dynamic C with the **OPTIONS** menu, then try to get Dynamic C to recognize the Rabbit 2000 TCP/IP Development Board by selecting **Recompile BIOS** on the **Compile** menu. Try the different COM ports in the **OPTIONS** menu until you find the one you are connected to. If you can't get Dynamic C to recognize the target on any port, then the hookup may be wrong or the COM port is not working on your PC.

If you receive the "BIOS successfully compiled …" message after pressing **<Ctrl-Y>** or starting Dynamic C, and this message is followed by "Target not responding," it is possible that your PC cannot handle the 115,200 bps baud rate. Try changing the baud rate to 57,600 bps as follows.

1. Open the BIOS source code file named **RABBITBIOS.C**, which can be found in the **BIOS** directory.
2. Change the line

```
#define USE115KBAUD 1   // set to 0 to use 57600 baud
```

   to read as follows.

```
#define USE115KBAUD 0   // set to 0 to use 57600 baud
```

3. Locate the **Serial options** dialog in the Dynamic C **Options** menu. Change the baud rate to 57,600 bps, then press **<Ctrl-Y>**.

When you receive the "BIOS successfully compiled …" message and do not receive a "Target not responding" message, the target is now ready to compile a program.

3.4 PONG.C

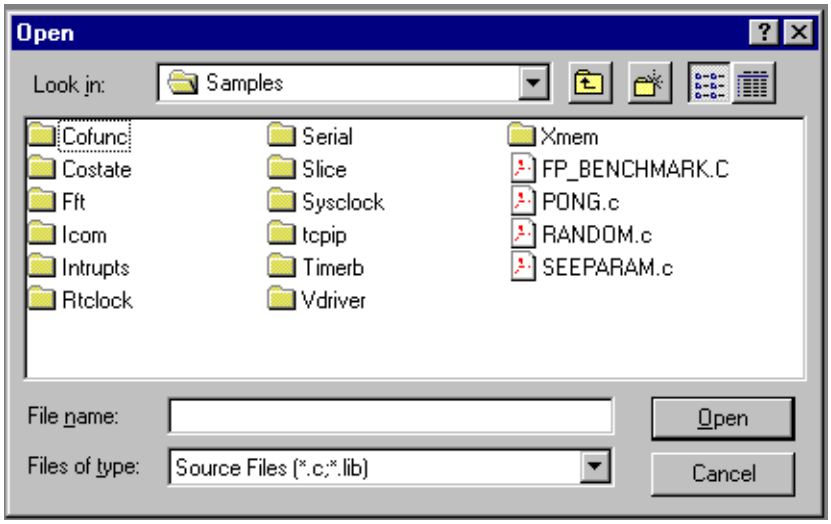
You are now ready to test your set-up by running a sample program.

Find the file **PONG.C**, which is in the Dynamic C **SAMPLES** folder. To run the program, open it with the **File** menu (if it is not still open), 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.

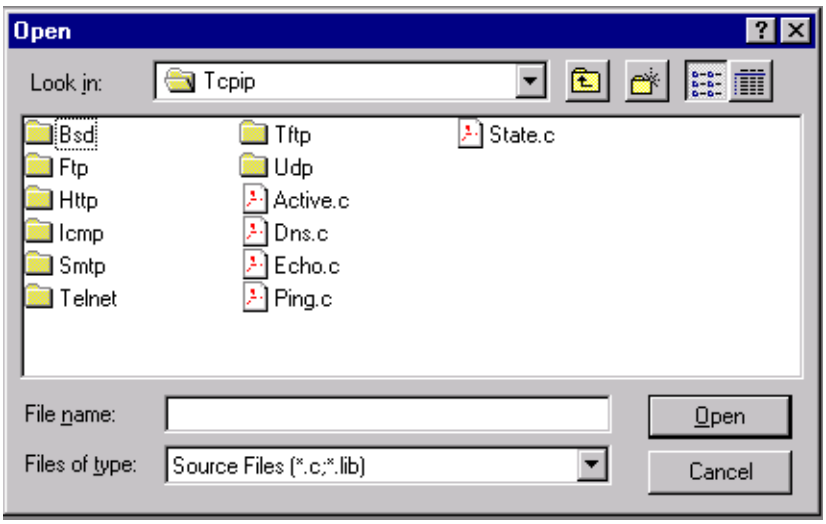
This program does not test the serial ports, the I/O, or the TCP/IP part of the board, but does ensure that the board is basically functional. The sample program in the next chapter tests the TCP/IP portion of the board.

3.5 Sample Programs

Other sample programs are provided in the Dynamic C **Samples** folder, which is shown below.



The various folders contain specific sample programs that illustrate the use of the corresponding Dynamic C libraries. The sample program **PONG.C** demonstrates the output to the **STDIO** window. The **ICOM** and **TCPIP** folders provide sample programs specific to the TCP/IP Development Board. Let's take a look at the **TCPIP** folder.



The various folders contain sample programs to illustrate the topics associated with the TCP/IP Development Board. See ***An Introduction to TCP/IP*** for more information on these topics.

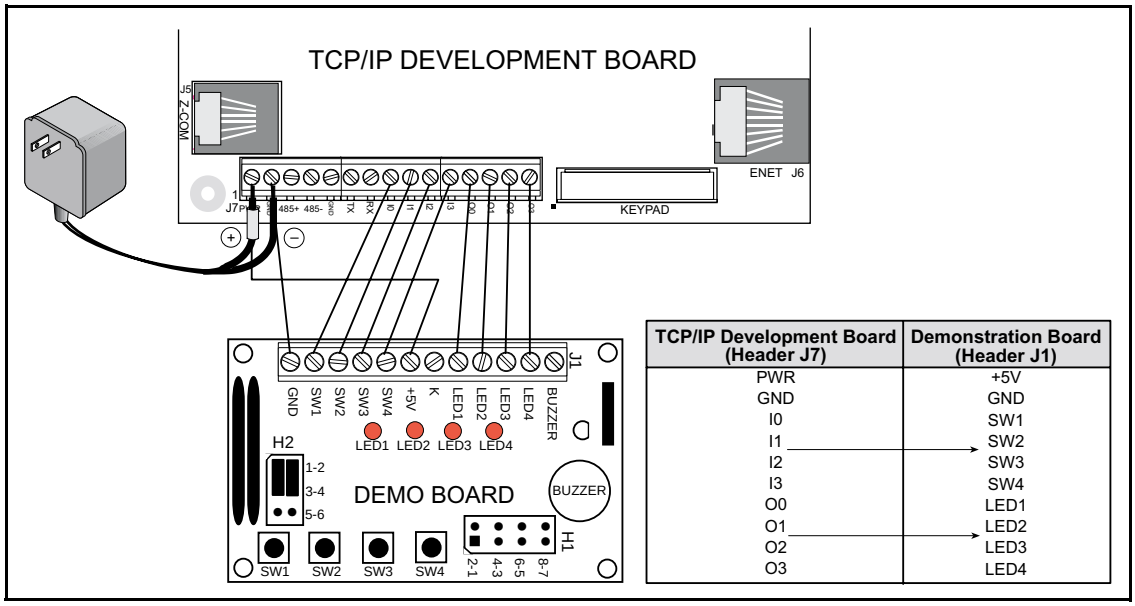
Each sample program has comments that describe the purpose and function of the program.

3.5.1 Running Sample Program DEMOBRD1.C

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

Before running this sample program, you will have to connect the Demonstration Board from the TCP/IP Development Kit to the TCP/IP Development Board. Proceed as follows.

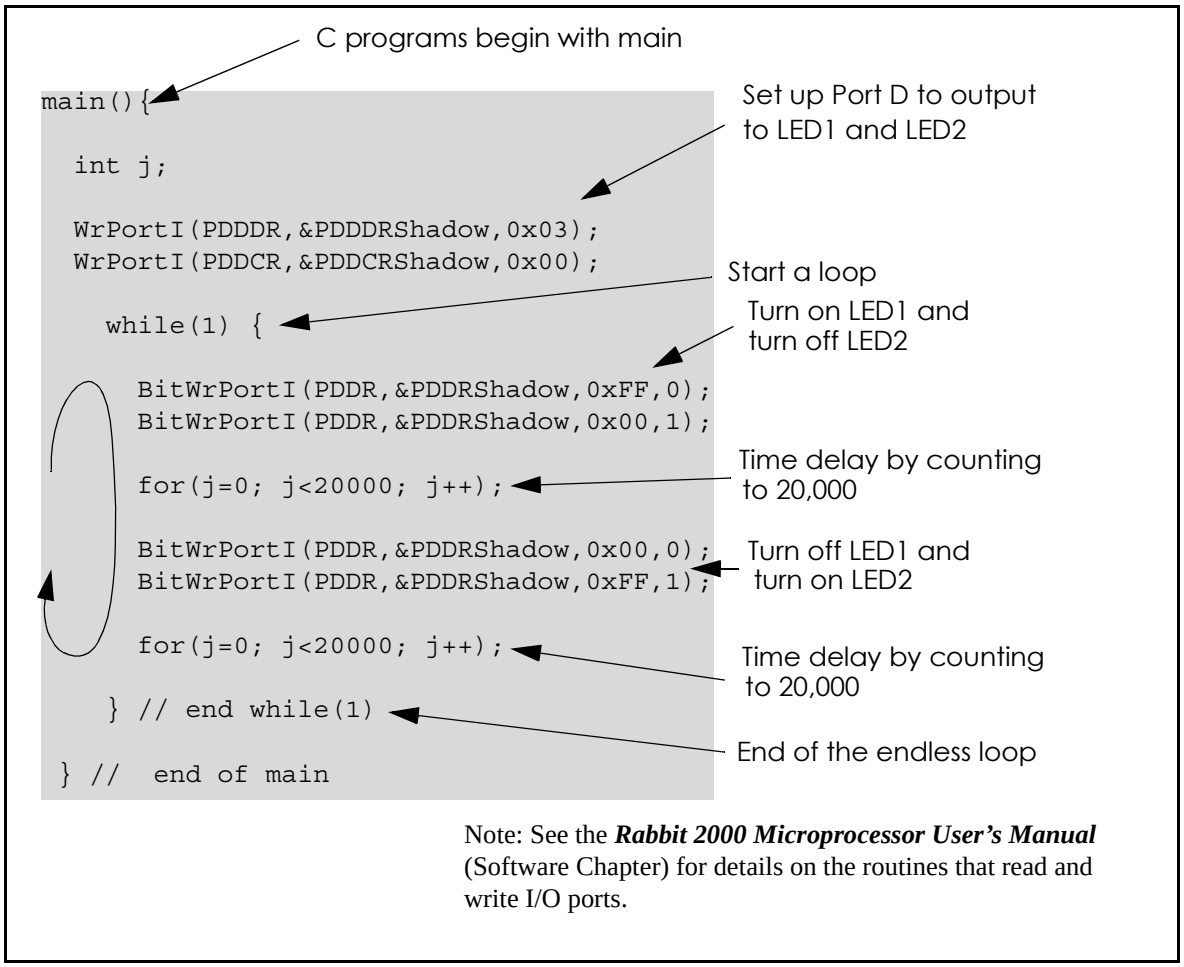
1. Use the wires included in the TCP/IP Development Kit to connect header J1 on the Demonstration Board to header J7 on the TCP/IP Development Board. The connections are shown in Figure 4.
2. Make sure that your TCP/IP Development Board is connected to your PC and that the power supply is connected to the TCP/IP Development Board and plugged in as described in Section 3.1.



TCP/IP Development Board (Header J7)	Demonstration Board (Header J1)
PWR	+5V
GND	GND
I0	SW1
I1 →	SW2
I2	SW3
I3	SW4
O0	LED1
O1 →	LED2
O2	LED3
O3	LED4

Figure 4. Connections Between TCP/IP Development Board and Demonstration Board

Now, open the file **DEMOBRD1.C**, which is in the **SAMPLES/ICOM** folder. The program will appear in a window, as shown in Figure 5 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.



Note: See the ***Rabbit 2000 Microprocessor User's Manual*** (Software Chapter) for details on the routines that read and write I/O ports.

Figure 5. Sample Program DEMOBRD1.C

To run the program **DEMOTCP1.C**, load it with the **File** menu, compile it using the **Compile** menu, and then run it by selecting **Run** in the **Run** menu. LED1 and LED2 on the Demonstration Board should start going on and off 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 "No Rabbit Processor Detected" in cases where the wall transformer is either not connected or is not plugged in.

- The programming cable must be connected to the TCP/IP Development Board. (The colored wire on the programming cable is closest to pin 1 on header J4 on the TCP/IP Development Board, as shown in Figure 1.) 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 serial port 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.

3.5.2 Single-Stepping

Compile or re-compile **DEMOBRD1.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.

3.5.2.1 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.

3.5.2.2 Break Point

Move the cursor to the start of the statement:

```
for(j=0; j<20000; 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. The break point will start flashing both red and green colors. Note that LED1 on the Demonstration Board is now solidly turned on. This is because we have passed the statement turning on LED1.

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 LED1 should be flashing again because 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.

3.5.2.3 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 20000 in the **for (..** statement to 10000. Then use the **F9** key to recompile and run the program. The LEDs will start flashing, but it will flash much faster than before because you have changed the loop counter terminal value from 20000 to 10000.

3.5.2.4 Watching Variables Dynamically

Go back to edit mode (select edit) and load the program **DEMOBRD2.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 **DEMOTCP2.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**.

3.5.2.5 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 **`rundwatch()`** 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.

3.5.3 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 Premier User's Manual***. The example below, sample program **`DEMOTCP3.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 Premier User's Manual***. The **`DEMOTCP3.C`** sample program has two independent tasks. The first task flashes LED2 once a second. The second task uses button SW1 on the Demonstration Board to toggle the logical value of a virtual switch, **`vswitch`**, and flash LED1 each time the button is pressed. This task also debounces button SW1.

Note that the Demonstration Board has to be connected to the TCP/IP Development Board as described in Section 3.5.1 to be able to run **`DEMOBRD3.C`**.

```
main() {
    int vswitch;        // state of virtual switch controlled by button S1

    WrPortI(PDDDR, &PDDDRShadow, 0x03); // set port D bits 0-1 as outputs
    WrPortI(PDDCR, &PDDCRShadow, 0x00); // set port D to not open drain mode
    vswitch = 0;                        // initialize virtual switch as off

(1) while (1) {                         // endless loop
        BigLoopTop();                   // begin a big endless loop

        // First task will flash LED4 for 200 ms once per second.

(2)     costate {
            BitWrPortI(PDDR, &PDDRShadow, 0xFF, 1);    // turn LED on
(3)         waitfor(DelayMs(200));                     // wait 200 ms
            BitWrPortI(PDDR, &PDDRShadow, 0x00, 1);    // turn LED off
            waitfor(DelayMs(800));                     // wait 800 ms
(4)     }

        // Second task - debounce SW1 and toggle vswitch

        costate {
(5)         if (!BitRdPortI(PDDR, 2)) abort; // if button not down skip out
            waitfor(DelayMs(50));            // wait 50 ms
            if(!BitRdPortI(PDDR, 2))  abort; // if button not still down exit

            vswitch = !vswitch;       // toggle since button was down 50 ms

            while (1) {
                waitfor(!BitRdPortI(PDDR, 2));  // wait for button to go up
                waitfor(DelayMs(200));          // wait additional 200 ms
                if (!BitRdPortI(PDDR, 2))
                    break;    // if button still up break out of while loop
            }
        }                     // end of costate

        // make LED1 agree with vswitch

(6)     BitWrPortI(PDDR, &PDDRShadow, vswitch, 0);

(7) }   // end of while loop
}       // end of main
```

The numbers in the left margin are reference indicators, and are not a part of the code. Load and run the program. Note that LED2 flashes once per second. Push button SW1 several times and note how LED1 is toggled.

The flashing of LED2 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 Premier 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.

3.5.4 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 caused slicing that is further described in the ***Dynamic C Premier User's Manual***.

4. RUNNING YOUR FIRST TCP/IP SAMPLE PROGRAM

4.1 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 that the user connect his PC and the TCP/IP Development Board 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.

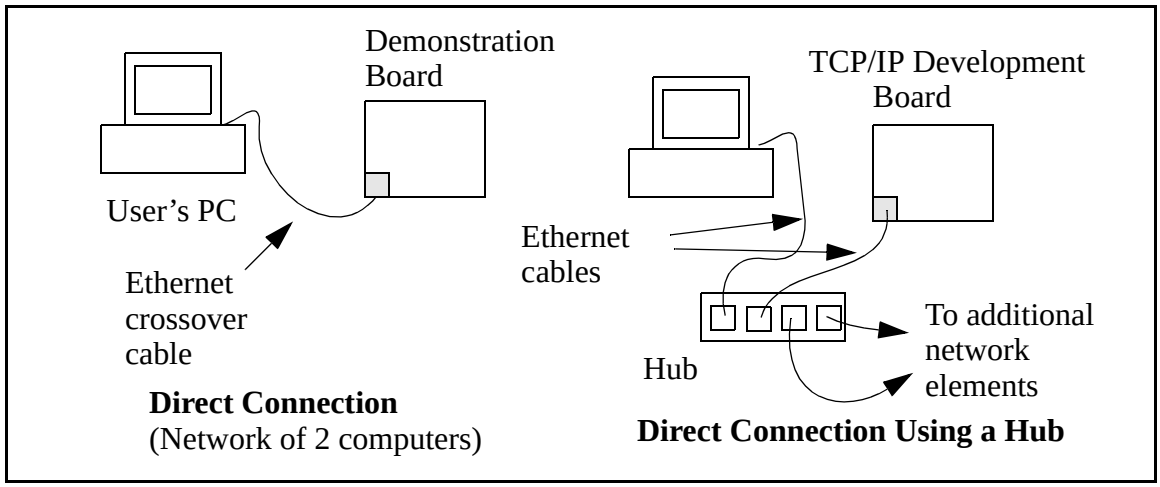
Obtaining IP addresses to interact over an existing, operating, network can involve a number of complications, and must usually be done with cooperation from your ISP and/or network systems administrator (if your company has one). For this reason it is suggested that the user begin instead by using a direct connection between his PC and the TCP/IP Development Board using an Ethernet crossover cable or a simple arrangement with a hub. (A crossover cable can be purchased at computer stores, but should not be confused with regular straight through cables.) The hub and a wide variety of cables can also be purchased from a local computer store.

In order to set up this direct connection, the user will have to use a virgin computer (right out of the box), or disconnect a computer from the corporate network, or as yet another approach install a second Ethernet adapter and setup a separate private network attached to the second Ethernet adapter. Disconnecting your computer from the corporate network may be easy or nearly impossible, depending on how it is set up. Mobile computers, such as laptops, are designed to be connected and disconnected and will present the least problem. If you computer 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 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.

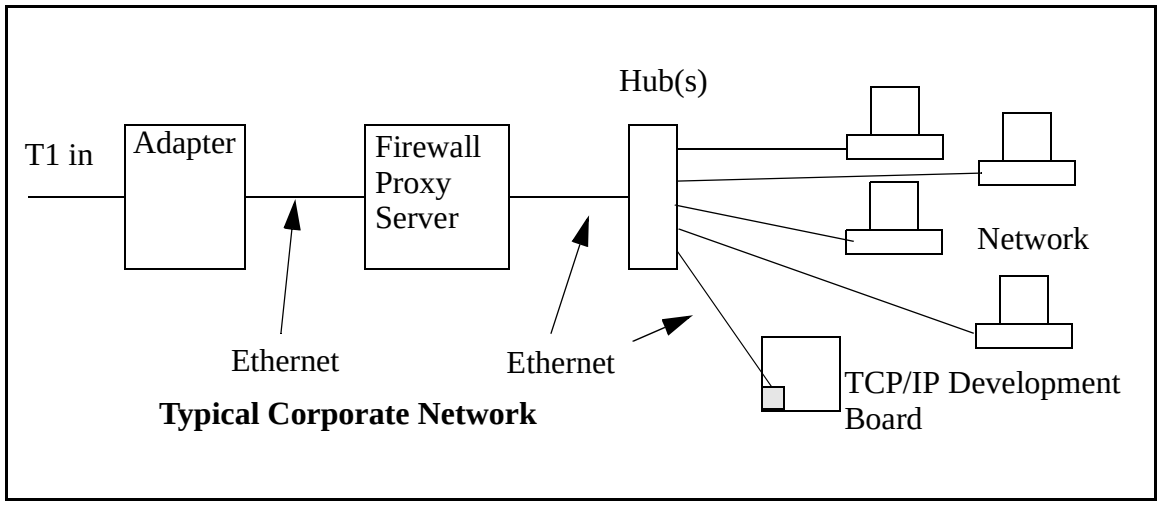
The following IP addresses are set aside for local networks and are not allowed on the Internet: 10.0.0.0 to 10.255.255.255, 172.16.0.0 to 172.31.255.255, and 192.168.0.0 to 192.168.255.255.

The TCP/IP Development Board uses a 10BaseT type of Ethernet connection, which is the most common scheme. Connectors are similar to U.S. style telephone connectors except larger and with 8 contacts (an RJ-45 connector).

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 readily available. The TCP/IP Development Board uses 10 Mbps Ethernet, so the hub or Ethernet adapter must be either a 10 Mbps unit or a 10/100 unit which adapts to either 10 or 100 Mbps.



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 include a combination of proxy servers and firewalls that filter and multiplex Internet traffic. In the configuration below, the TCP/IP Development Board 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 of the configuration complications by sacrificing some security.



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 demos without having to setup a direct connection between your computer and the TCP/IP Development Board. 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 demos.

4.2 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.

4.3 How IP Addresses are Used

The actual hardware connection via 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 IP address on the local network to which the packet must be sent. A table is maintained which allows 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.

IP addresses are arbitrary and can be allocated as desired provided that they don't conflict with other IP addresses. However, if they are to be used with the Internet, then they must be numbers that are assigned to your connection by proper authorities generally by delegation via your service provider.

4.4 Dynamically Assigned Internet Addresses

In many instances, IP addresses are assigned temporarily. This the normal procedure when you use a dial up internet service provider (ISP). Your system will be provided with an IP address which it can use to send and receive packets. This IP address will only be valid for the duration of the call and further may not actually be a real Internet IP address. Such an address works for browsing the Web, but cannot be used for transactions originating elsewhere since no other system has any way to know the internet address except by first receiving a packet from you. (If you want to find the IP address assigned by a dial up ISP run the program **`winipcfg`** while connected and look at the address for the ppp adapter under Windows 98.)

In a typical corporate network that is isolated from the Internet by a firewall and/or proxy server using address translation, the IP addresses are not usually actual Internet addresses and may be statically or dynamically assigned. If they are assigned statically, you only have to get an unused IP address and assign it to the TCP/IP Development Board. If the IP addresses are assigned dynamically, they you will have to get an IP address that is valid but outside of the range of IP addresses that are dynamically assigned. This will enable you to communicate from a computer on the network to the TCP/IP Development Board. If you want to communicate to the TCP/IP Development Board from the external Internet, then an actual Internet IP address must be assigned to the TCP/IP Development Board. It may be possible to setup the firewall to pass a real IP address, or it may be necessary to connect the TCP/IP Development Board in front of the firewall to accomplish this.

4.5 How to Set IP Addresses in the Demo Programs

Most of the demo programs such as shown in the example below use macros to define the IP address assigned to the board and the IP address of the gateway, if there is a gateway.

```
#define MY_IP_ADDRESS "216.112.116.155"
#define MY_NETMASK "255.255.255.248"
#define MY_GATEWAY "216.112.116.153"
```

In order to do a direct connection the following IP addresses can be used for the TCP/IP Development Board:

```
#define MY_IP_ADDRESS "10.1.1.2"
#define MY_NETMASK "255.255.255.248"
// #define MY_GATEWAY "216.112.116.153"
```

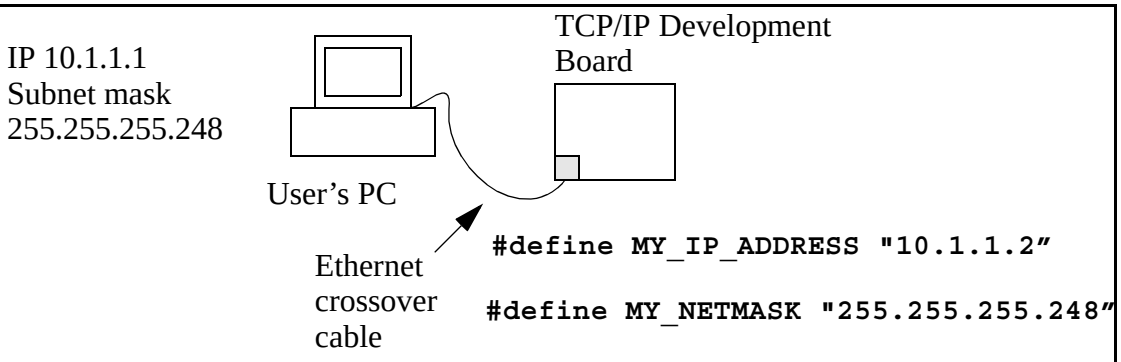
In this case, the gateway is not used and is commented out. The IP address of the board is defined to be 10.1.1.2. The IP address of you computer can be defined as 10.1.1.1.

4.6 How to Set Up your Computer's IP Address For Direct Connect

When your computer is connected directly to the TCP/IP Development Board via an Ethernet connection, you need to assign an IP address to your computer. To assign the computer the address 10.1.1.1 with the subnetmask 255.255.255.248 under Windows 98, do the following.

Click on **Start > Settings > Control Panel** to bring up the control panel, and then double-click the network icon. In the window find the line of the form TCP/IP -> Ethernet adapter name. Double-click on this line to bring up the TCP/IP properties dialog box. You can edit the IP address directly and the subnet mask. (Disable "obtain an IP address automatically.") You may want to write down the existing values in case you have to restore them later. It is not necessary to edit the gateway address since the gateway is not used with direct connect.

The method of setting the IP address may differ for different versions of Windows, such as 95, NT or 2000, .



Direct Connection PC to TCP/IP Development Board

4.7 Run the PINGME.C Demo

In order to run this program edit the IP address and netmask in the **PINGME.C** program (**SAMPLES\TCPIP\ICMP**) to the values given above (10.1.1.2 and 255.255.255.248). Compile the program and start it running under Dynamic C. The crossover cable is connected from your computer's Ethernet adapter to the TCP/IP Development Board's RJ-45 Ethernet connector. When the program starts running, the green LNK light on the Development Board should be on to indicate an Ethernet connection is made. (Note: If the LNK light does not light, you may not have 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 computer. This can be done by bringing up the MS-DOS window and running the pingme program:

```
ping 10.1.1.2
```

or by **Start** > **Run**

and typing the entry

```
ping 10.1.1.2
```

Notice that the red ACT light flashes on the Development Board while the ping is taking place indicating the transfer of data. The ping routine will ping the board four times and write a summary message on the screen describing the operation.

4.8 Running More Demo Programs With Direct Connect

The programs **STATIC.C** and **SSI3.C** (**SAMPLES/TCPIP/HTTP**) demonstrate how to make the TCP/IP Development Board be a Web server. This program allows you to turn on and off the LED's on the attached LED board from a remote Web browser. In order to run these demos edit the IP address as for the pingme program, compile the program and start it executing. Then bring up your Web browser and enter the following server address: http://10.1.1.2.

This should bring up the Web page served by the demo program. The demo program **static.c** is a static Web page. The demo program **ssi3.c** allows you to control the TCP/IP Development Board from the Web browser, turning on and off the led indicators on the small board attached to the main TCP/IP Development Board.

The demo program **RXSAMPLE.C** (**SAMPLES/TELNET**) allows you to communicate with the TCP/IP Development Board using the telnet protocol. To run this program, edit the IP address and compile it and start it running. Run the telnet program on your PC (**Start > Run** `telnet 10.1.1.2`). Each character you type will be printed in Dynamic C's **STDIO** window, indicating that the board is receiving the characters typed via TCP/IP.

4.9 Where Do I Go From Here?

If there are any problems at this point, call Rabbit Semiconductor Technical Support at (530)757-8400.

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

Additional sample programs are described in the ***TCP/IP High-Level Protocols*** manual

Please refer to the ***TCP/IP High-Level Protocols*** manual and the ***TCP/IP Function Reference*** manual (click the documentation icon on your PC) to develop your own applications. ***An Introduction to TCP/IP*** provides background information on TCP/IP, and is included on the CD.

5. SERIAL PORTS AND DIGITAL I/O

Chapter 5 describes how to set up TCP/IP Development boards for serial communication, and how to use the digital I/O.

The TCP/IP Development Board has 15 pins on header J7, one RJ-12 jack for RS-232 or RS-485 serial communication, and one Ethernet jack. The pinouts are shown in Figure 6.

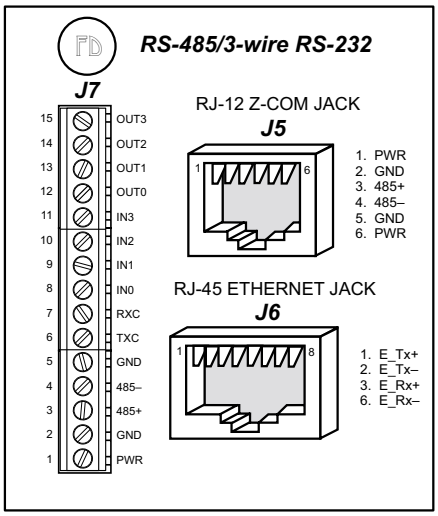


Figure 6. TCP/IP Development Board I/O Pinout

RJ-45 pinouts are sometimes numbered opposite to the way shown in Figure 6. Regardless of the numbering convention followed, the pin positions relative to the spring tab position (located at the bottom of the RJ-45 jack in Figure 6) are always absolute, and the RJ-45 connector will work properly with off-the-shelf Ethernet cables.

5.1 Serial Communication

In the factory-default configuration, the TCP/IP Development Board has one RS-232 (3-wire) serial channel, one RS-485 serial channel, and one synchronous CMOS serial channel.

5.1.1 RS-232

The TCP/IP Development Board's RS-232 serial channel is connected to an RS-232 transceiver, U11. U11 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 2000's 0 V to +Vcc signals to RS-232 signal levels. Note that the polarity is reversed in an RS-232 circuit so that +5 V is output as approximately -10 V and 0 V is output as approximately +10 V. U11 also provides the proper line loading for reliable communication.

The maximum baud rate is 115,200 bps. RS-232 can be used effectively at this baud rate for distances up to 15 m.

5.1.2 RS-485

The TCP/IP Development Board has one RS-485 serial channel, which is connected to the Rabbit 2000 serial port D through U10, an RS-485 transceiver. U10 supports the RS-485 serial communication protocol. The chip's slew rate limiters provide for a maximum baud rate

of 250,000 bps, which allows for a network of up to 300 m (or 1000 ft). The half-duplex communication uses the Rabbit 2000's PC0 pin to control the data enable on the communication line.

The RS-485 signals are available on pins 3 and 4 of header J7, and on J5, the RJ-12 jack.

The TCP/IP Development Board can be used in an RS-485 multidrop network. Connect the 485+ to 485+ and 485– to 485– using single twisted-pair wires (nonstranded, tinned).

Alternatively, the RS-485 multidrop network may be hooked up using cables with RJ-12 plugs. Note that the RJ-12 jack has +RAW_485 and GND, which means that only ***one*** Intellicom needs to be connected to an external power source via an AC adapter. When doing so, ensure that the AC adapter has sufficient capacity for the network — a TCP/IP Development Board nominally draws 100 mA at 12 VDC.

If you plan to connect a power supply to more than one TCP/IP Development Board in an RS-485 network using the RJ-12 jacks, rework the RS-485 cables so they do not connect +RAW_RS485 through the RJ-12 jack to the boards in the network.

The TCP/IP Development Board comes with a 220 Ω termination resistor and 680 Ω bias resistors already installed, as shown in Figure 7.

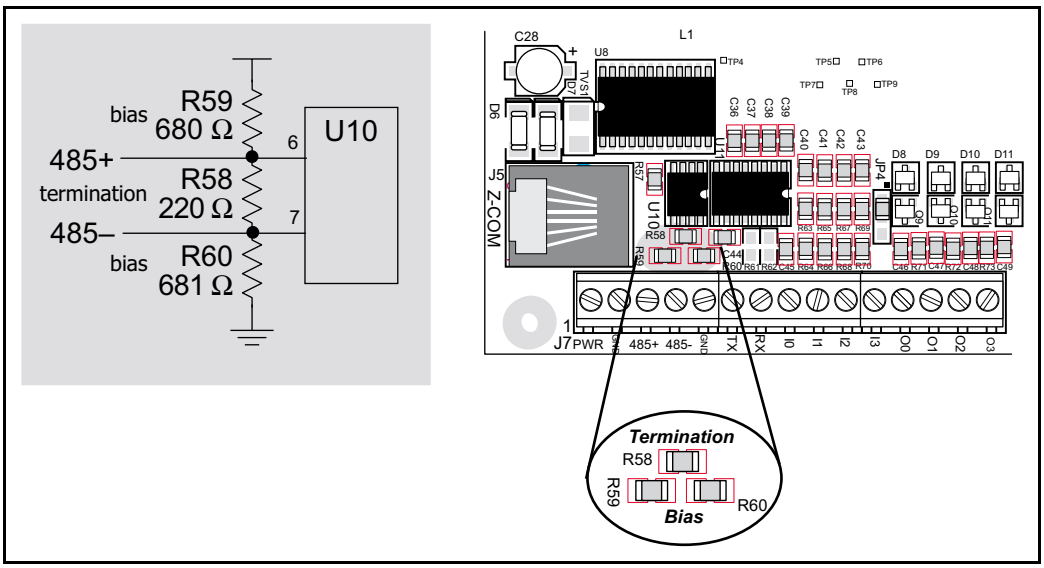


Figure 7. RS-485 Termination and Bias Resistors

The load these bias and termination resistors present to the RS-485 transceiver (U10) limits the number of TCP/IP Development Boards in a multidrop network to one master and nine slaves, unless the bias and termination resistors are removed. When using more than 10 TCP/IP Development Boards in a multidrop network, leave the 680 Ω bias resistors in place on the master TCP/IP Development Board, and leave the 220 Ω termination resistors in place on the TCP/IP Development Board at each end of the network.

5.1.3 Programming Port

The TCP/IP Development Board has a 10-pin programming header labeled J4. The programming port uses the Rabbit 2000's serial port A for communication. The Rabbit 2000 startup-mode pins (SMODE0, SMODE1) are presented to the programming port so that an externally connected device can force the Intellicom to start up in an external bootstrap mode.

Refer to the ***Rabbit 2000 Microprocessor User's Manual*** for more information related to the bootstrap mode.

The programming port is used to start the TCP/IP Development Board in a mode where the TCP/IP Development Board will download a program from the port and then execute the program. The programming port transmits information to and from a PC while a program is being debugged.

The TCP/IP Development Board can be reset from the programming port.

The Rabbit 2000 status pin is also presented to the programming port. The status pin is an output that can be used to send a general digital signal.

The clock line for serial port A is presented to the programming port, which makes fast serial communication possible.

5.1.4 Serial Communication Software

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

User interface to set up up serial communication lines for the Intellicom board. Call this function after **serXOpen()**.

Parameters

mode is the defined serial port configuration of the devices installed.

Mode	Serial Port	
	B	C
0	RS-485	RS-232, 3-wire
1	RS-232, 3-wire	RS-232, 3-wire
2	RS-232, 5-wire	CTS/RTS

Return Value

0 if correct mode, 1 if not.

See Also

serB485Tx, serB485Rx

void serB485Tx();

Sets pin 3 (DE) high to disable Rx and enable Tx.

See Also

`serMode, serB485Rx`

void serB485Rx();

Resets pin 3 (DE) low to enable Rx and disable Tx.

See Also

`serMode, serB485Tx`

5.1.4.1 Sample Serial Communication Programs

RS-232

1. Connect RX to TX as shown in Figure 8 below.

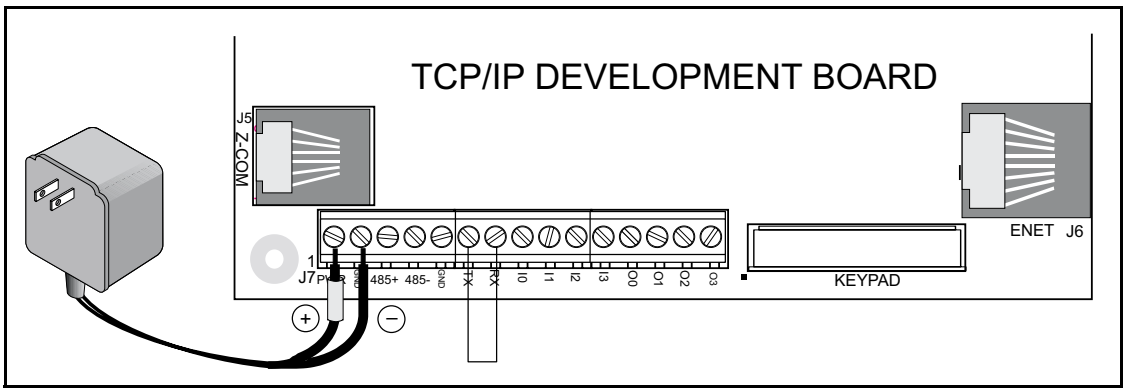


Figure 8. TCP/IP Development Board Setup for RS-232 Serial Communication Demonstration

2. Connect the programming cable to header J4 on the TCP/IP Development Board. Apply power to the TCP/IP Development Board.
3. Open the sample program **`SAMPLES\ICOM\ICOM232.C`** and press **F9**.

This program demonstrates a simple RS-232 loopback displayed in the **STDIO** window.

RS-485

1. Connect 485+ to 485+, 485– to 485–, and GND to GND as shown in Figure 9 below. If you do not have a separate wall transformer for the other board, also connect PWR to PWR as shown in Figure 9.

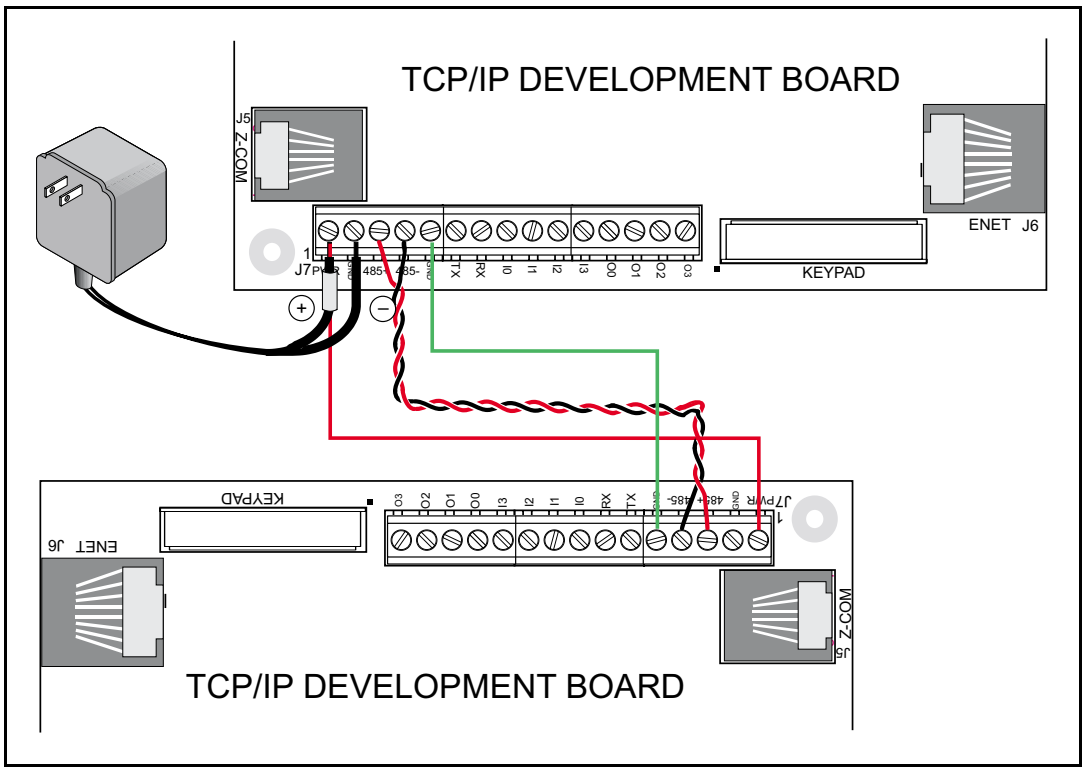


Figure 9. TCP/IP Development Board Setup for RS-485 Serial Communication Demonstration

2. Connect the programming cable to header J4 on one TCP/IP Development Board. This will be the slave, the other board will be the master. Apply power to the TCP/IP Development Boards.
3. Open the sample program **`SAMPLES\ICOM\ICOM485.C`**. You will find some code for the master, and some code for the slave. Copy and save the master and slave versions separately.
4. Open the sample slave program and press **F9**.
5. Connect the programming cable to header J4 on the master TCP/IP Development Board.
6. Open the master program and press **F9**.

This program demonstrates a simple RS-485 transmission of lower-case letters to a slave. The slave will send back converted upper case letters back to the master, which then displays them in the **STDIO** window.

5.2 Digital I/O

5.2.1 Digital Inputs

Pins 8–11 on header J7 have the four digital inputs IN0–IN3. Each of the four digital 0 V to 5 V inputs is protected over a range of –36 V to +36 V. The TCP/IP Development Board is factory-configured for the digital inputs to be pulled up to +5 V, but the digital inputs can also be pulled down by moving the surface-mounted jumper at JP4. The jumper settings and the location of JP4 are shown in Figure 10.

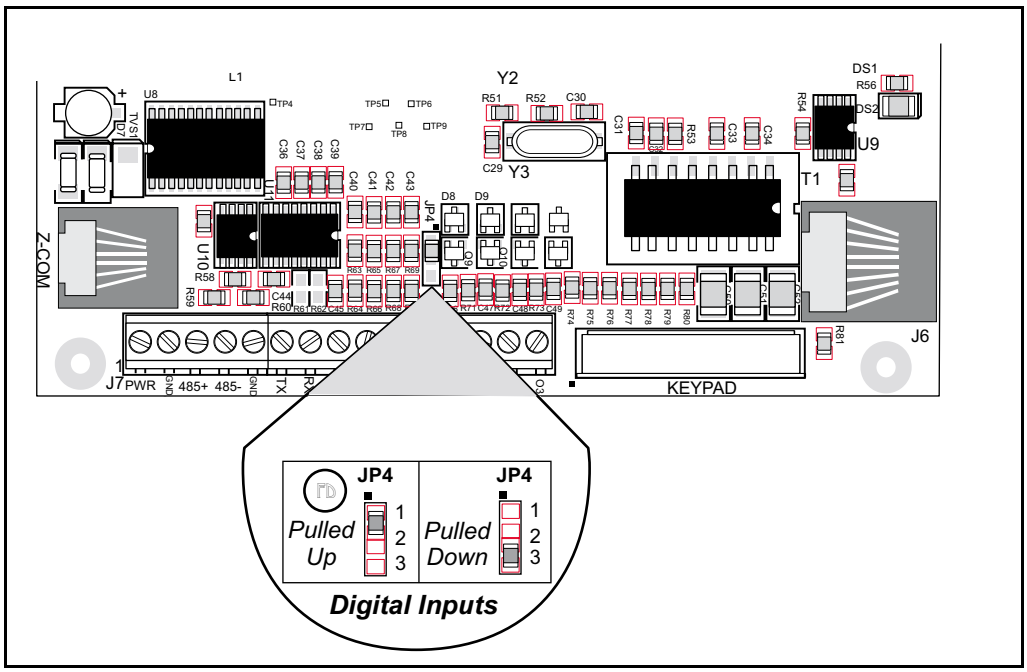


Figure 10. Surface-Mounted Jumper Configurations for Selecting Pullup/Pulldown on the Digital Inputs

5.2.2 Digital Outputs

Pins 12–15 on header J7 have the four digital outputs OUT0–OUT3. Each of the four open-collector digital outputs can sink up to 200 mA at 40 VDC.

5.2.3 Digital I/O Software

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

Sets the state of a digital output.

Parameters

`channel` is the output channel number (0, 1, 2, or 3).

`value` is the output value (0 or 1).

Return Value

None.

See Also

`digIn`

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

Reads the state of a digital input.

Parameters

`channel` is the input channel number (0, 1, 2, or 3).

Return Value

The state of the input (0 or 1).

See Also

`digOut`

5.2.4 Sample Digital I/O Programs

1. Connect the programming cable to header J4 on the TCP/IP Development Board. Apply power to the TCP/IP Development Board.
2. Open the sample program **`SAMPLES\ICOM\ICOMIO.C`** and press **F9**.

This program demonstrates how to turn the I/O on and off.

APPENDIX A. TCP/IP DEVELOPMENT BOARD SPECIFICATIONS

A.1 Electrical and Mechanical Specifications

Figure A-1 shows the mechanical dimensions for the TCP/IP Development Board.

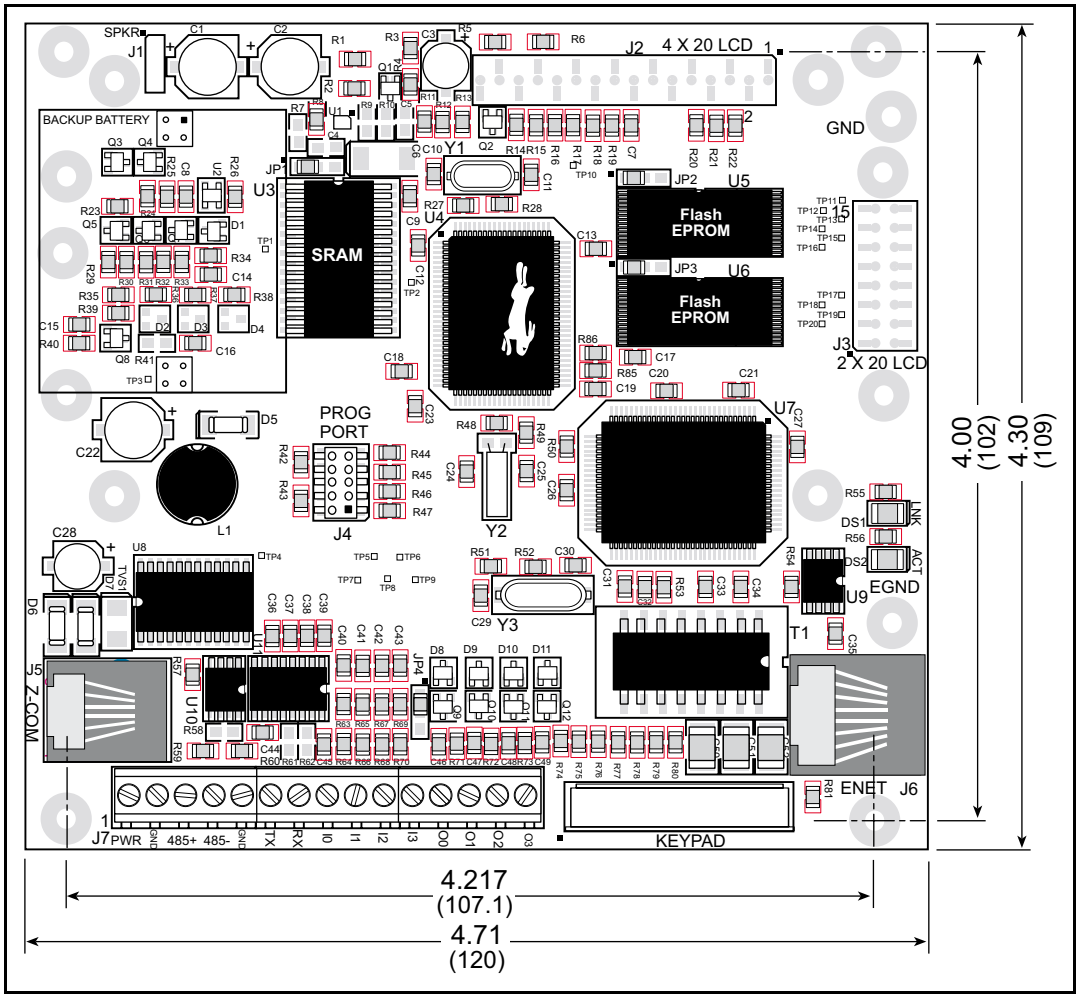


Figure A-1. TCP/IP Development Board Dimensions

Table A-1 lists the electrical, mechanical, and environmental specifications for the TCP/IP Development Board.

Table A-1. TCP/IP Development Board Specifications

Parameter	Specification
Board Size (with backup battery board)	4.30" × 4.71" × 0.79" (109 mm × 120 mm × 20 mm)
Connectors	15 screw terminals, 1 RJ-12, and 1 RJ-45
Operating Temperature	–20°C to +70°C
Humidity	5% to 95%, noncondensing
Input Voltage	9 V to 40 V DC
Current	100 mA @ 12 VDC
Ethernet Interface	Direct connection to 10BaseT Ethernet networks via RJ-45 connection
Digital Inputs	4 protected, 0 V to 5 V DC (protection from –36 V to + 36 VDC max.)
Digital Outputs	4 open collector, sinking (200 mA, 40 V DC max.)
Microprocessor	Rabbit 2000™
Clock	18.432 MHz
SRAM	128K, surface mount (supports 32K–512K)
Flash EPROM	256K for program and data plus 256K for file storage (supports 128K–512K)
Timers	7 eight-bit timers available
Serial Ports	• 1 RS-232 (3-wire), 1 RS-485, and 1 RS-232 programming port • RS-232 (3-wire) and RS-485 may be reconfigured for 1 RS-232 (5-wire) or 2 RS-232 (3-wire)
Serial Rate	Maximum asynchronous 115,200 bps for both serial ports
Watchdog/Supervisor	Yes
Time/Date Clock	Yes
Backup Battery	On backup battery board (not included)

APPENDIX B. POWER MANAGEMENT

B.1 Power Supplies

Power is supplied to the TCP/IP Development Board from an external source either through header J7 or from another TCP/IP Development Board through header J5, the RJ-12 jack.

The TCP/IP Development Board itself is protected against reverse polarity by Shottky diodes at D6 and D7 as shown in Figure B-1. The Shottky diode has a low forward voltage drop, 0.3 V, which keeps the minimum DCIN required to power the TCP/IP Development Board lower than a normal silicon diode would allow.

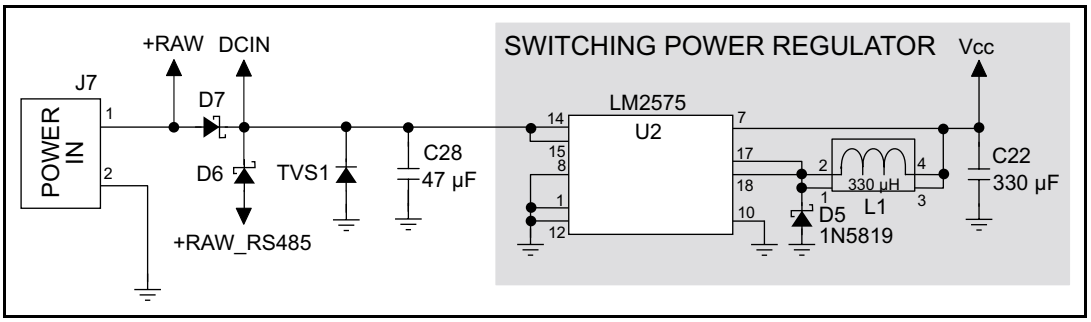


Figure B-1. TCP/IP Development Board Power Supply Schematic

Capacitor C28 provides surge current protection for the voltage regulator, and allows the external power supply to be located some distance away from the TCP/IP Development Board. A switching power regulator is used. The input voltage range is from 9 V to 40 V.

B.2 Batteries and External Battery Connections

A battery board with a 1000 mA·h lithium coin cell is available to provide power to the real-time clock and SRAM when external power is removed from the circuit. This allows the TCP/IP Development Board to continue to keep track of time and preserves the SRAM memory contents.

Figure B-2 shows the battery board circuit.

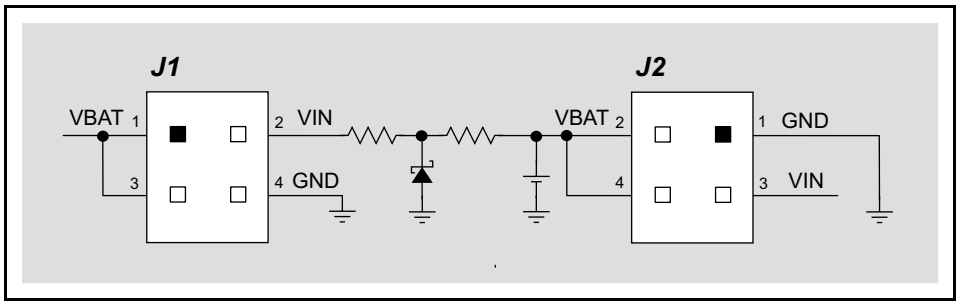


Figure B-2. Battery Board Circuit

The drain on the battery is typically less than 20 µA when there is no external power applied. The battery can last more than 5 years:

$$\frac{1000\ \text{mA}\cdot\text{h}}{20\ \mu\text{A}} = 5.7\ \text{years}.$$

The drain on the battery is typically less than 4 µA when external power ***is*** applied. The battery can last for its full shelf life:

$$\frac{1000\ \text{mA}\cdot\text{h}}{4\ \mu\text{A}} = 28.5\ \text{years (shelf life = 10 years)}.$$

Since the shelf life of the battery is 10 years, the battery can last for its full shelf life when external power is applied to the TCP/IP Development Board.

B.2.1 Battery Backup Circuit

The battery-backup circuit serves two purposes:

- It reduces the battery voltage to the real-time clock, thereby reducing the current consumed by the real-time clock and lengthening the battery life.
- It ensures that current can flow only ***out*** of the battery to prevent charging the battery.

Figure B-3 shows the TCP/IP Development Board battery backup circuitry on the TCP/IP Development Board.

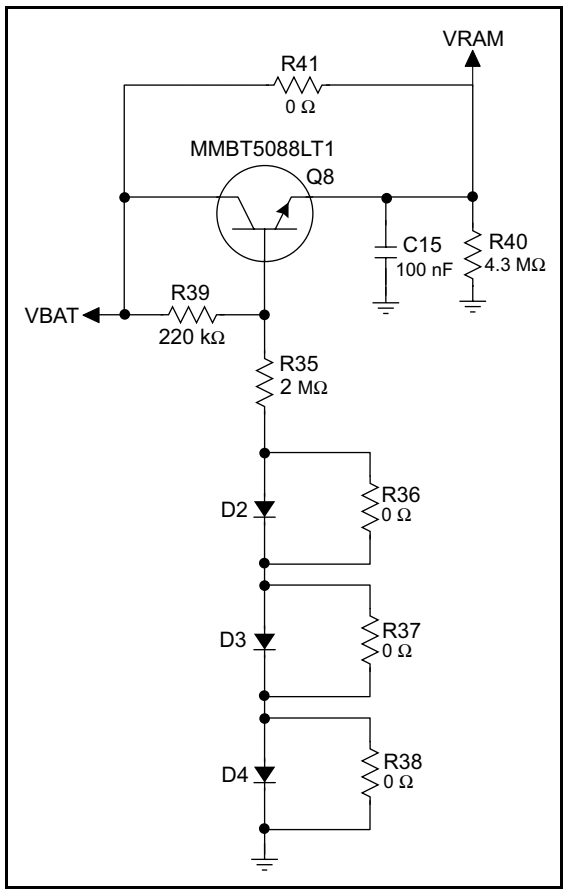


Figure B-3. TCP/IP Development Board Battery Backup Circuit

Resistor R41, shown in Figure B-3, is typically not stuffed on the TCP/IP Development Board. VRAM and Vcc are nearly equal (<100 mV, typically 10 mV) when power is supplied to the TCP/IP Development Board. R14 on the backup battery board prevents any catastrophic failure of Q8 by limiting current from the battery.

Resistors R35 and R39 make up a voltage divider between the battery voltage and the temperature-compensation voltage at the anode of diode D2. This voltage divider biases the base of Q8 to about 0.9 × VBAT. V_{BE} on Q8 is about 0.55 V. Therefore, VRAM is about 0.9 × VBAT - 0.55 V, or about 2.15 V for a 3 V battery.

These voltages vary with temperature. VRAM varies the least because temperature-compensation diodes D2–D4 will offset the variation with temperature of Q8's V_{BE}. R36–R38 may be stuffed instead of the corresponding D2–D4 to provide the optimum temperature compensation.

Resistor R40 provides a minimum load to the regulator circuit.

VRAM is also available on pin 34 of header J2 to facilitate battery backup of the external circuit. Note that the recommended minimum resistive load at VRAM is 100 kΩ, and new battery life calculations should be done to take external loading into account.

B.2.2 Power to VRAM Switch

The VRAM switch, shown in Figure B-4, allows the battery backup to provide power when the external power goes off. The switch provides an isolation between Vcc and the battery when Vcc goes low. This prevents the Vcc line from draining the battery.

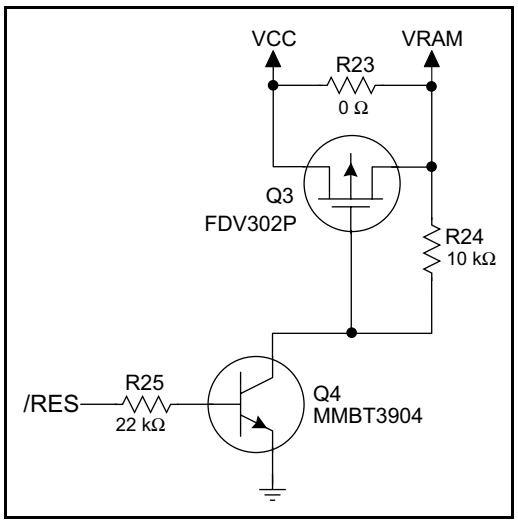


Figure B-4. VRAM Switch

Transistor Q3 is needed to provide a very small voltage drop between Vcc and VRAM (<100 mV, typically 10 mV) so that the processor lines powered by Vcc will not have a significantly different voltage than VRAM.

When the TCP/IP Development Board is ***not*** resetting (pin 2 on U4 is high), the /RES line will be high. This turns on Q4, causing its collector to go low. This turns on Q3, allowing VRAM to nearly equal Vcc.

When the TCP/IP Development Board ***is*** resetting, the /RES line will go low. This turns off Q3 and Q4, providing an isolation between Vcc and VRAM.

The battery backup circuit keeps VRAM from dropping below 2 V.

B.2.3 Reset Generator

The TCP/IP Development Board uses a reset generator, U2, to reset the Rabbit 2000 microprocessor when the voltage drops below the voltage necessary for reliable operation. The reset occurs between 4.50 V and 4.75 V, typically 4.63 V.

B.2.4 Installing/Replacing the Backup Battery Board

An optional pluggable backup battery board is available from Z-World.

To install the backup battery board, align the battery board over the outline as shown in Figure B-5, and plug it in. Be careful to align the connectors and the backup battery board. Fasten the backup board using a 4-40 × 3/16 screw and lockwasher.

Before replacing the backup battery board, make sure that the TCP/IP Development Board is receiving power from the standard power supply. This makes sure that data in RAM are not lost when the battery backup board is removed temporarily.

To replace the backup battery board, remove the screw and unplug the old battery board. Then install a replacement backup battery board.

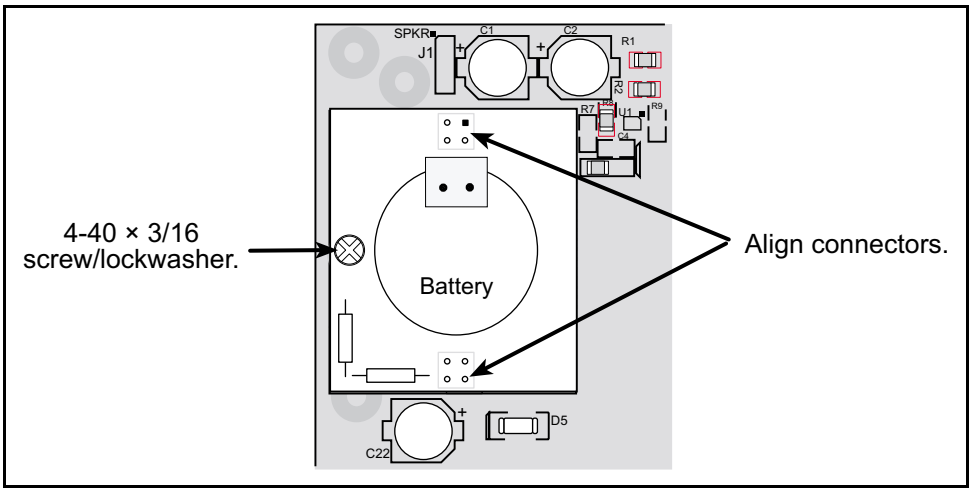


Figure B-5. Installing Backup Battery Board

Do ***not*** attempt to recharge the old battery and do ***not*** dispose of it in regular trash to avoid any risk of explosion or fire. You may either return the old backup battery board to Z-World for recycling or send the battery yourself to an approved recycling facility.

B.3 Chip Select Circuit

Figure B-6 shows a schematic of the chip select circuit.

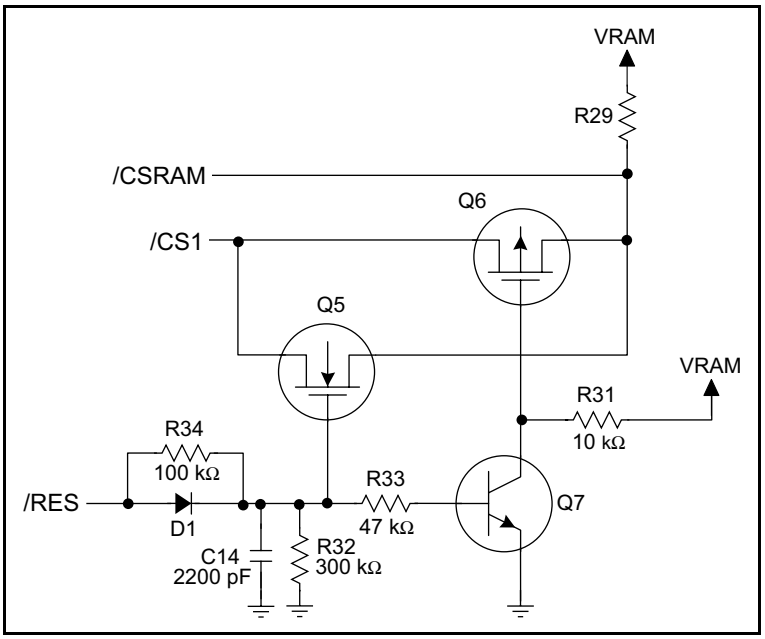


Figure B-6. Chip Select Circuit

The current drain on the battery in a battery-backed circuit must be kept at a minimum. When the TCP/IP Development Board is not powered, the battery keeps the SRAM memory contents and the real-time clock (RTC) going. The SRAM has a powerdown mode that greatly reduces power consumption. This powerdown mode is activated by raising the chip select (CS) signal line. Normally the SRAM requires Vcc to operate. However, only 2 V is required for data retention in powerdown mode. Thus, when power is removed from the circuit, the battery voltage needs to be provided to both the SRAM power pin and to the CS signal line. The CS control circuit accomplishes this task for the CS signal line.

In a powered-up condition, the CS control circuit must allow the processor's chip select signal /CS1 to control the SRAM's CS signal /CSRAM. So, with power applied, /CSRAM must be the same signal as /CS1, and with power removed, /CSRAM must be held high (but only needs to be battery voltage high). Q5 and Q6 are MOSFET transistors with opposing polarity. They are both turned on when power is applied to the circuit. They allow the CS signal to pass from the processor to the SRAM so that the processor can periodically access the SRAM. When power is removed from the circuit, the transistors will turn off and isolate /CSRAM from the processor. The isolated /CSRAM line has a 100 kΩ pullup resistor to VRAM (R29). This pullup resistor keeps /CSRAM at the VRAM voltage level (which under no power condition is the backup battery's regulated voltage at a little more than 2 V).

Transistors Q5 and Q6 are of opposite polarity so that a rail-to-rail voltage can be passed. When the /CS1 voltage is low, Q5 will conduct. When the /CS1 voltage is high, Q6 will conduct. It takes time for the transistors to turn on, creating a propagation delay. This delay is typically very small, about 10 ns to 15 ns.

The signal that turns the transistors on is a high on the processor's reset line, /RES. When the TCP/IP Development Board is not in reset, the reset line will be high, turning on n-channel Q5 and Q7. Q7 is a simple inverter needed to turn on Q6, a p-channel MOSFET. When a reset occurs, the /RES line will go low. This will cause C14 to discharge through R32 and R34. This small delay (about 160 µs) ensures that there is adequate time for the processor to write any last byte pending to the SRAM before the processor puts itself into a reset state. When coming out of reset, CS will be enabled very quickly because D1 conducts to charge capacitor C14.

Index

T

SCHEMATICS