RabbitCore 2000

C-Programmable Core Module

Getting Started

001004 - C

RabbitCore 2000 Getting Started

Part Number 019-0080 • 001004 - C • 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

Z-World, Inc.
2900 Spafford Street
Davis, California 95616-6800
USA

Telephone: (530) 757-3737
Facsimile: (530) 753-5141
Web site: http://www.zworld.com
E-mail: zworld@zworld.com

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

Telephone: (530) 757-8400
Facsimile: (530) 757-8402
Web site: http://www.rabbitsemiconductor.com
E-mail: sales@rabbitsemiconductor.com

Table of Contents

About This Manual

Z-World customers develop software for their programmable controllers using Z-World's Dynamic C development system running on an IBM-compatible PC. Dynamic C 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 development system to write software for a RabbitCore 2000 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).

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.

Development and Evaluation Tools

Development Kit

The Development Kit has the essentials that you need to design your own a microprocessor-based system, and includes a complete software development system (Dynamic C).

The items in the Development Kit and their use are as follows:

- CD-ROM with Dynamic C® SE software, RabbitCore 2000, and Rabbit™ 2000 microprocessor documentation. You may install this software by inserting the disk into your CD-ROM drive. If it doesn't start automatically, click on "setup.exe." This software runs under Windows '95, Windows '98, Windows 2000, and Windows NT. We suggest taking the option to load the documentation to your hard disk. The documentation is in both HTML and Adobe PDF format, and may be viewed with a browser.
- RabbitCore 2000 (RCM2020). This is a complete controller board that includes a Rabbit 2000 processor, 256K of flash memory, 128K of SRAM.
- Prototyping Board. The RabbitCore 2000 can be plugged into this board. The Protoyping Board includes a 5 V supply for powering the RabbitCore 2000, and various accessories such as pushbutton switches, and LEDs. In addition, you can add your own circuitry using through-hole or surface mount parts in the prototyping space provided.
- Programming cable. The programming cable is used to connect your PC serial port directly to the RabbitCore 2000 to write and debug C programs that run on the Rabbit 2000.
- AC adapter. The AC adapter is used to power the Prototyping Board and the RabbitCore 2000. The wall transformer is supplied only for Development Kits sold for the North American market. The RabbitCore 2000 can also be powered from any DC voltage source between 7.5 V and 25 V, but 12 V is recommended. The linear regulator becomes rather hot for voltages above 15 V.

Documentation

- Our documentation is provided in paperless form on the CD-ROM included in the Development Kit. (A paper copy of the "Getting Started" page *is* included.) Most documents, including this comprehensive ***RabbitCore 2000 User's Manual***, are provided in two formats: HTML and PDF. HTML documents can be viewed with an Internet browser, either *Netscape Navigator* or *Internet Explorer*. HTML documents are very convenient because all the documents are hyperlinked together, and it is easy to navigate from one place to another. PDF documents can be viewed using the Adobe Acrobat reader, which is automatically invoked from the browser. The PDF format is best suited for documents requiring high resolution, such as schematics, or if you want to print the document. Don't print a hard copy from the HTML version because the HTML version has no page numbers and the cross-references and table of contents links only work if viewed on line. The PDF versions contain page number references to allow navigation when reading a paper version of the manual. To view the online documentation with a browser, open the file **`default.htm`** in the **`docs`** folder.

1. INSTALLING DYNAMIC C

1.1 Requirements

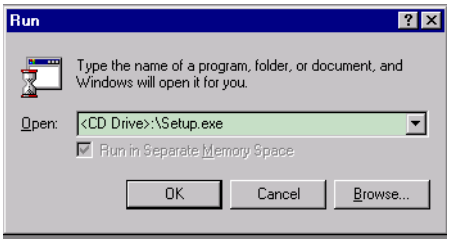
Dynamic C software comes on CD. To install Dynamic C, your system must be running one of the following.

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

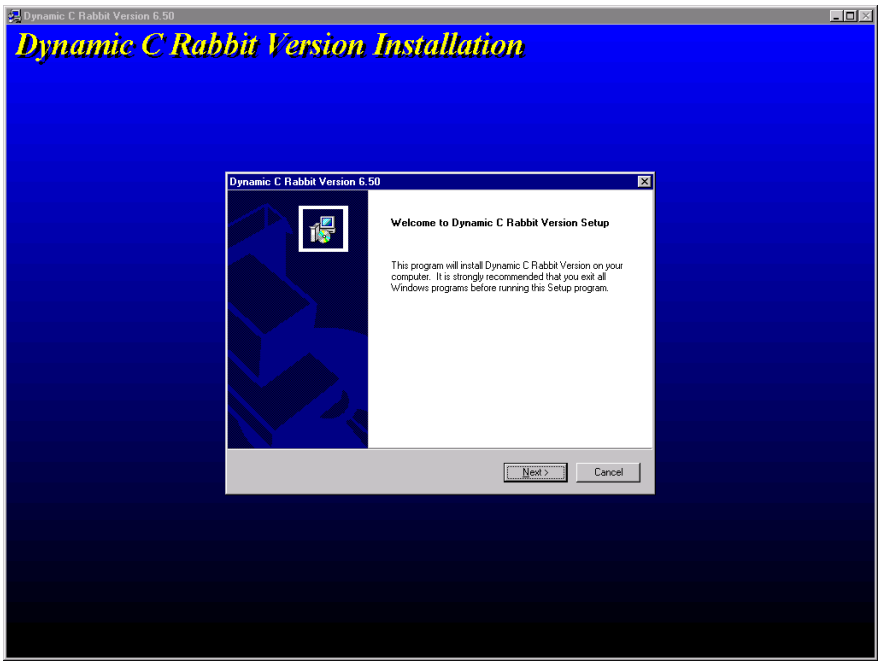
Your PC should have at least one free COM port

1.2 Installation

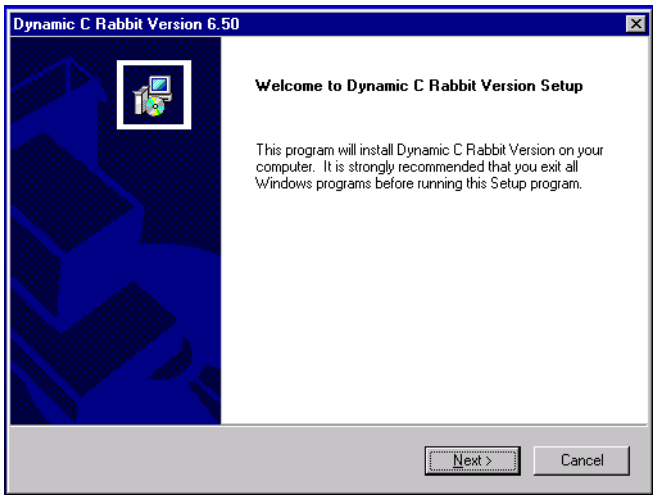
Insert the CD in the CD-ROM disk drive on your PC. As long as auto-install is enabled, the CD installation will begin automatically. If not, issue the Windows **Start** > **Run...** command and type the following.



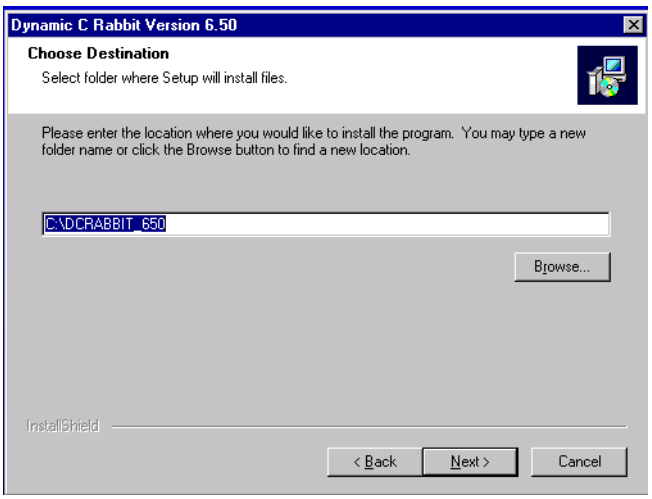
The installation program will then guide you through the installation process described below.



Click the **Next >** button to continue to the license agreement.

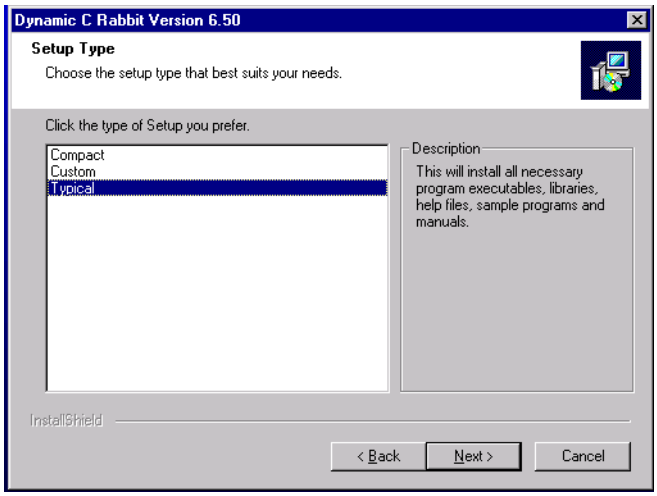


After reading and agreeing to the terms of the license, continue with the **Next >** button to select the destination folder where the files will be installed.

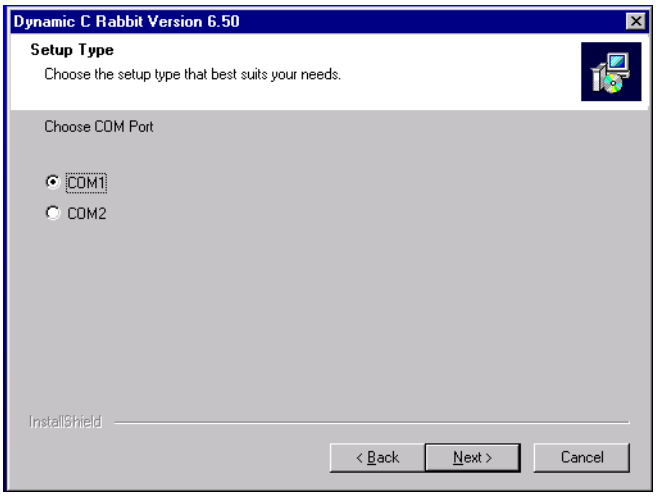


Click the **Next >** button to continue the installation. The Installation Wizard will prompt you to select a Compact, a Custom, or a Typical installation.

- Compact Installation—Dynamic C files only, no documents
- Custom Installation—your choice of Dynamic C files and documents
- Typical Installation—all Dynamic C files and all documents

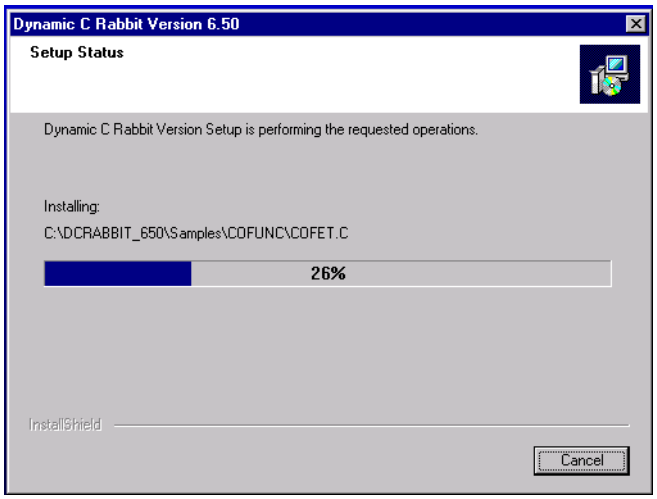


After choosing the installation, click the **Next >** button to continue. The files selected for installation are check-marked. Now select the PC COM port, usually, COM1.



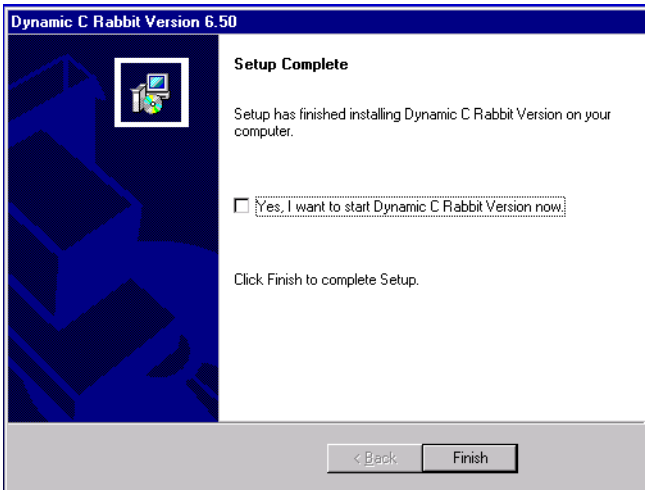
Click the **Next >** button to continue.

A status indicator shows the progress of the installation.



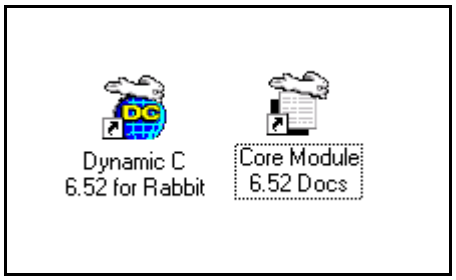
Before the installation is complete, the installation wizard will ask you what icons to display on your PC desktop. Separate icons are available for Dynamic C itself and for the manuals and other documents.

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_652\DcRab652.exe** to start Dynamic C
- **C:\DCRABBIT_652\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.

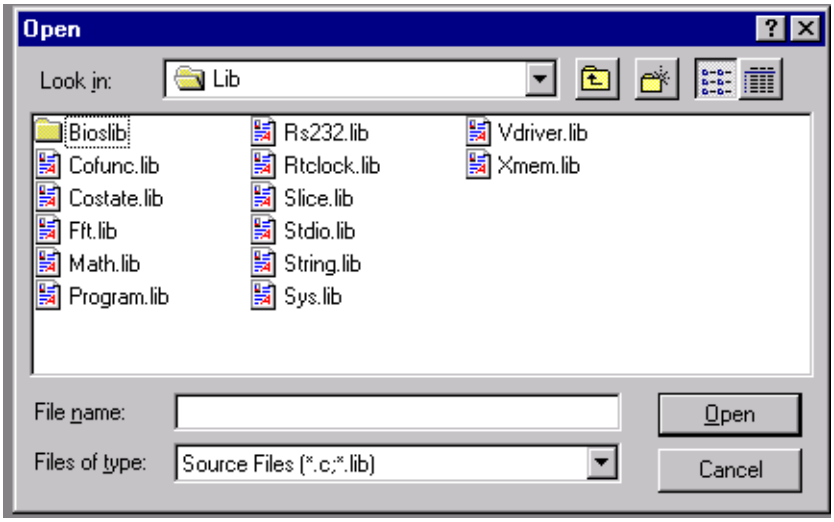
Dynamic C comes with many function libraries, all in source code. These libraries support real-time programming, machine level I/O, and provide standard string and math functions.

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.

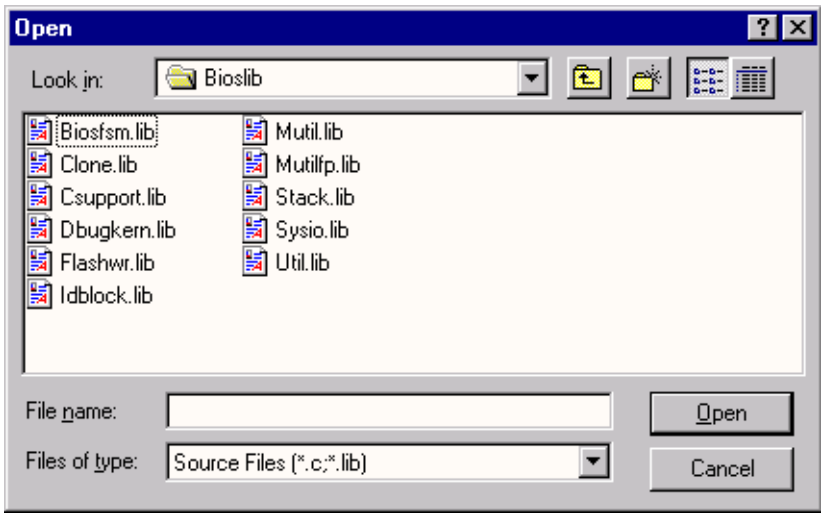


Let's examine the libraries.

- **Bioslib**—libraries specific to running a BIOS, apply to all controllers. Although the functions in these libraries are required by the BIOS, they are not exclusive to the BIOS.
- **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.

2.3 Using Dynamic C

Chapter 4., "Sample Programs," provides sample programs and explains how to use the basic features of Dynamic C.

More complete information on Dynamic C is provided in the ***Dynamic C (Rabbit Version) User's Manual***. Functions specific to the RabbitCore 2000 are described in the ***RabbitCore 2000 User's Manual***.

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/support.html

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.

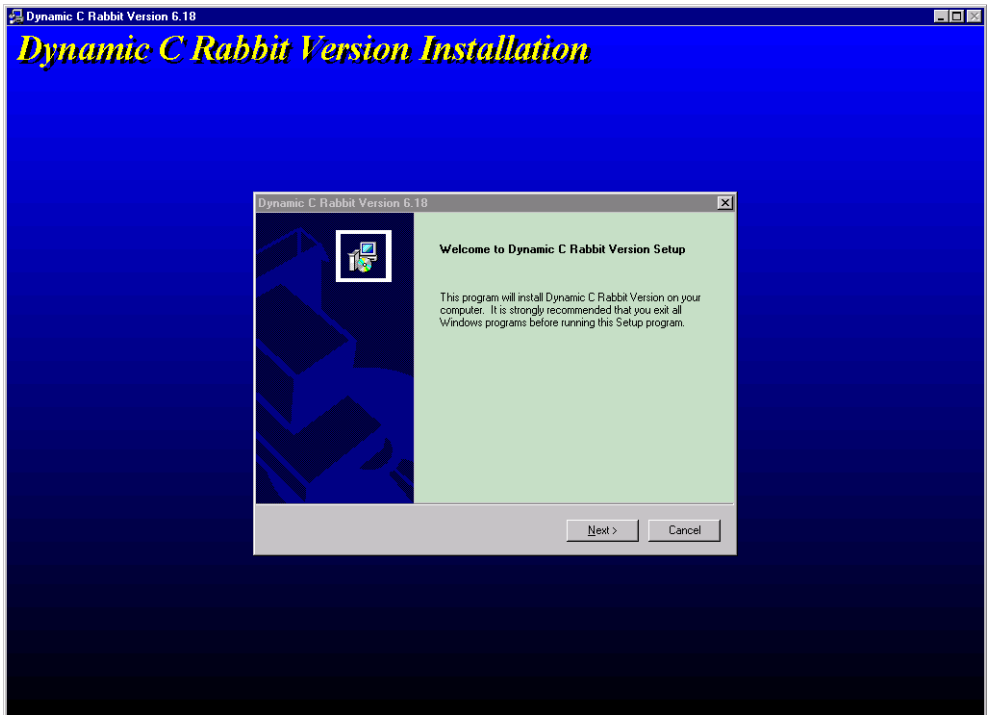
Reference Number	Description	Work-Around	Version(s) Affected
3	When a jp instruction in embedded assembly code is the next to be executed, the DC debugger doesn't highlight the jp instruction in the source window.	Look at the assembly window when single stepping through assembly code.	6.0x, 6.1x
6	Run-time 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

Before proceeding with the hardware connections described in this chapter, locate the following items.

- RabbitCore 2000 (model RCM2020)
- RCM2000 Prototyping Board
- Power supply (a 12 V, 500 mA power supply is included with Development Kits sold for the North American market)
- 10-pin to DE9 programming cable

3.1 Connections

1. Attach RabbitCore 2000 to Prototyping Board

Turn the RabbitCore 2000 so that the Rabbit 2000 microprocessor is facing as shown below. Plug RabbitCore 2000 Headers J1 and J2 into the sockets of headers J1 and J3 on the Prototyping Board.

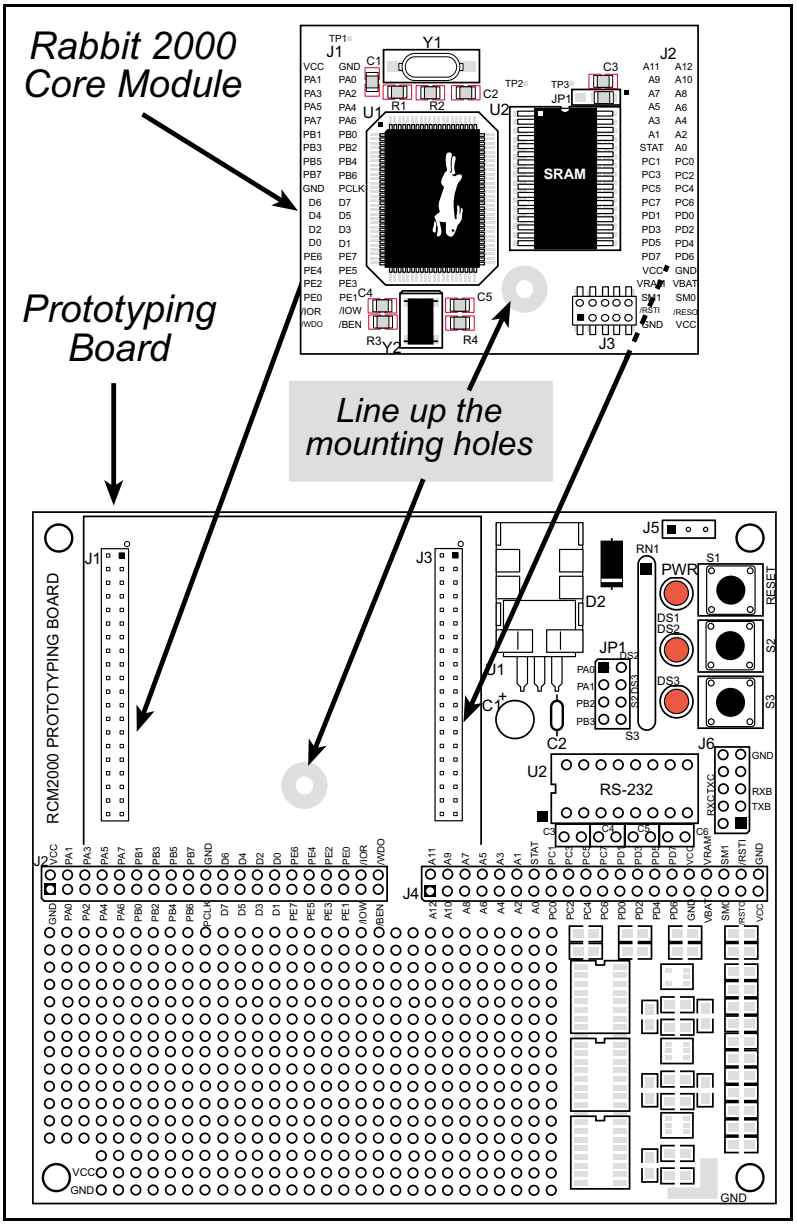


Figure 1. Attaching RabbitCore 2000 to Prototyping Board

It is important that you line up the pins on the RabbitCore 2000 headers J1 and J2 exactly with the corresponding pins of headers J1 and J3 on the Prototyping Board. The header pins may become bent or damaged if the pin alignment is offset, and the RabbitCore 2000 will not work.

2. Connect RabbitCore 2000 to PC

Connect the 10-pin connector of the programming cable to header J3 on the RabbitCore 2000 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.

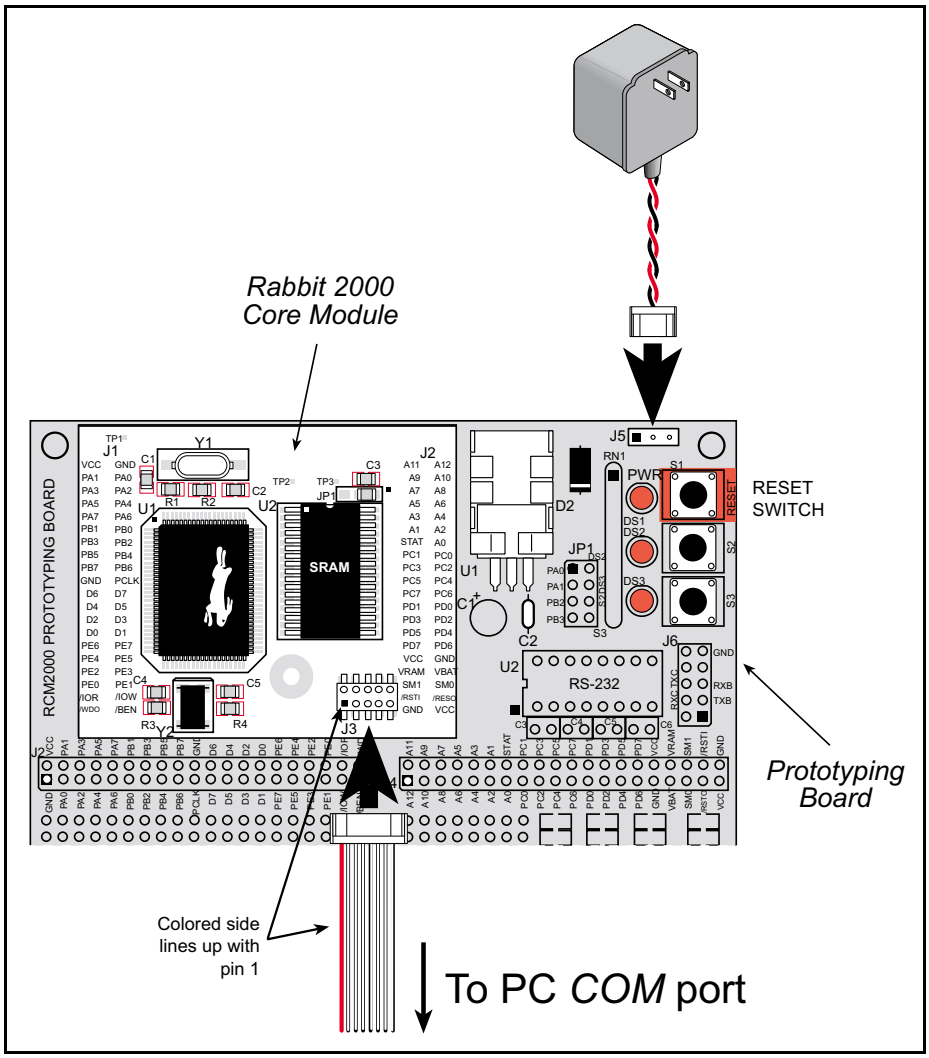


Figure 2. Power and Programming Connections to RabbitCore 2000

3. Power Supply Connections

Hook up the connector from the wall transformer to header J5 on the Prototyping Board as shown above. 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 J5.

Plug in the wall transformer. The power LED on the Prototyping Board should light up. The RabbitCore 2000 and the Prototyping Board are now ready to be used.

A RESET button is provided on the Prototyping Board to allow a hardware reset.

3.2 Starting Dynamic C

Once the RabbitCore 2000 is connected as described in the preceding pages, 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.

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.

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 RabbitCore 2000 and go through a sequence of steps to cold-boot the RabbitCore 2000 and to compile the BIOS. If an error message 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 RabbitCore 2000 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, **RABBITBIOS.C**.
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>**.

If 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.

3.3 Run a Sample Program

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

Find the file **FLASHLEDS.C**, which is in the Dynamic C **Samples/COREMODULE** folder. To run the program, open it using **File > Open**, compile it with **Debug Compile to Target** in the **Compile** menu, and then run it by selecting **Run** in the **Run** menu. LEDs DS2 and DS3 on the Prototyping Board should start flashing.

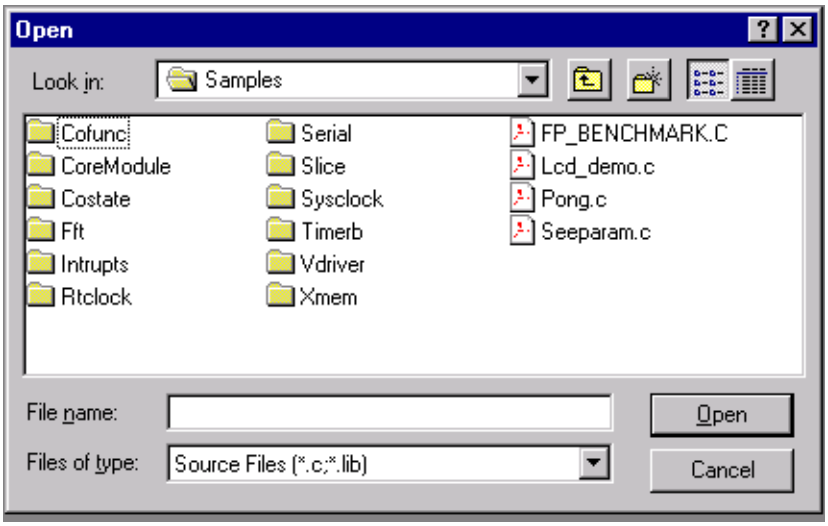
3.4 Where Do I Go From Here?

If there are any problems at this point, call Z-World Technical Support at (530)757-3737 or Rabbit Semiconductor Technical Support at (530)757-8400.

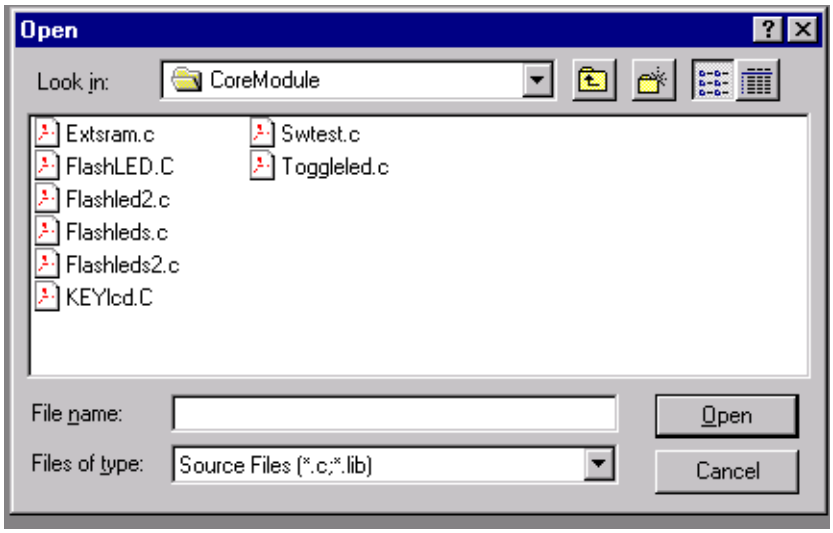
If the sample program ran fine, you are now ready to go on to other sample programs in the next chapter or to the ***RabbitCore 2000 User's Manual*** (click the documentation icon on your PC desktop or on **`Docs\default.htm`** in the Dynamic C directory).

4. SAMPLE PROGRAMS

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 **CoreModule** folder provides sample programs specific to the RabbitCore 2000. Let's take a look at the **CoreModule** folder.



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 RabbitCore 2000 is connected to the Prototyping Board and to your PC as described in the ***RabbitCore 2000 Getting Started*** manual.

4.1 Running Sample Program FLASHLED.C

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

First, open the file **FLASHLED.C**, which is in the **Samples/CoreModule** folder. The program will appear in a window, as shown in Figure 3 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.

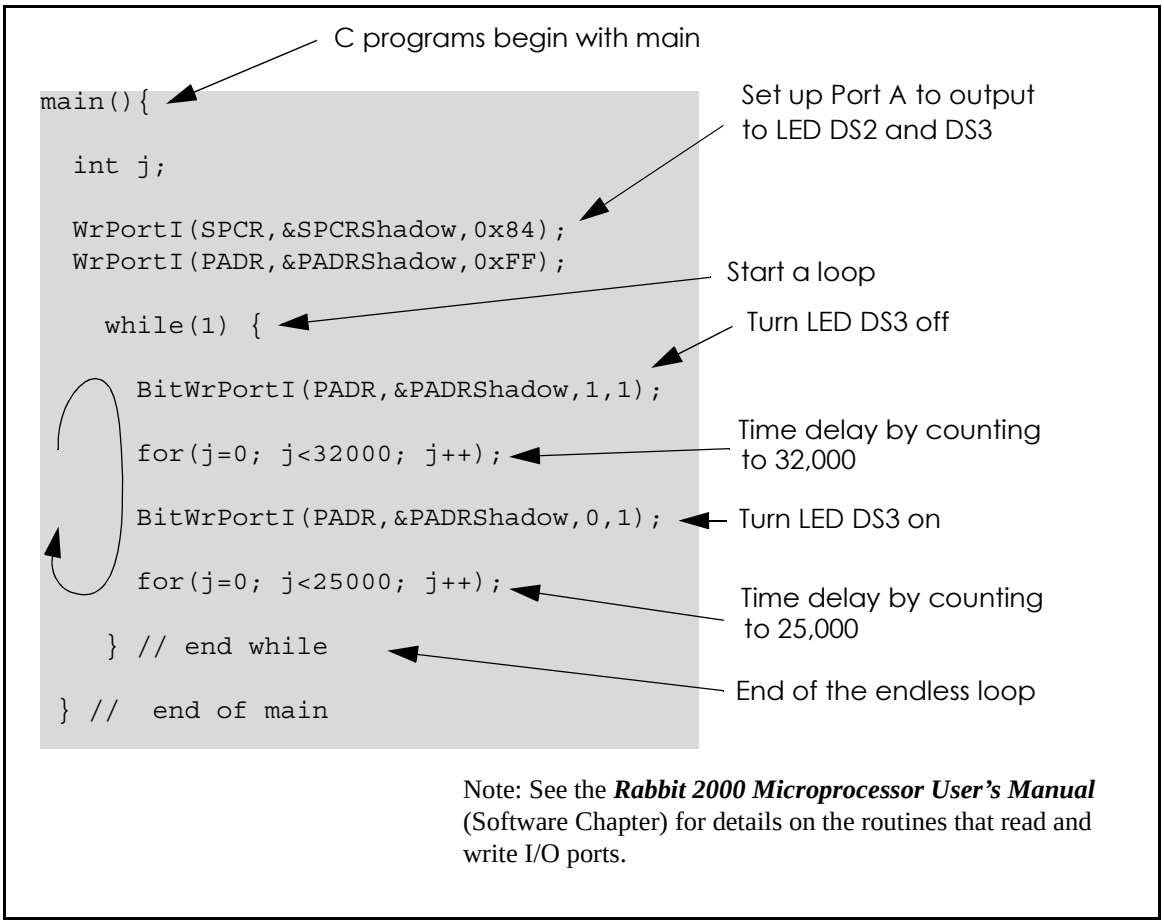


Figure 3. Sample Program FLASHLED.C

To run the program **FLASHLED.C**, load it with the **File** menu, compile it using the **Compile** menu, and then run it by selecting **Run** in the **Run** menu. The LED on the Prototyping 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 “No Rabbit Processor Detected” in cases where the RabbitCore 2000 and the 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 RabbitCore 2000. (The colored wire on the programming cable is closest to pin 1 on header J3 on the RabbitCore 2000, as shown in Figure 2.) 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.

4.2 Single-Stepping

Compile or re-compile **FLASHLED.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.

4.2.1 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.

4.2.2 Break Point

Move the cursor to the start of the statement:

```
for(j=0; j<25000; j++);
```

To set a break point on this statement, type **F2** or select **Toggle 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 32000. This is because the loop above terminated when `j` reached 32000.

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.

4.2.3 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 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.

4.2.4 Watching Variables Dynamically

Go back to edit mode (select edit) and load the program **FLASHLED2.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 **FLASHLED2.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**.

4.2.5 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.

4.3 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 another 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 (Rabbit Version) User's Manual***. The example below, sample program **`FLASHLEDS2.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 (Rabbit Version) User's Manual***. The **`FLASHLEDS2.C`** sample program has two independent tasks. The first task flashes LED DS2 2.5 times a second. The second task flashes DS3 every 1.5 seconds.

```
#define DS2 0           // predefine for LED DS2
#define DS3 1           // predefine for LED DS3

//  This cofunction flashes LED on for ontime, then off for offtime
cofunc flashled[4](int led, int ontime, int offtime) {
    for(;;) {
        waitfor(DelayMs(ontime));                       // on delay
        WrPortI(PADR,&PADRShadow,(1<<led)|PADR);        // turn LED off
        waitfor(DelayMs(offtime);                       // off delay
        WrPortI(PADR,&PADRShadow,(1<<led)^0xff&PADR);   // turn LED on
    }
}

main {
    // Initialize ports
    WrPortI(SPCR,&SPCRShadow,0x84);    // Set Port A all outputs, LEDs on
    WrPortI(PEFR,&PEFRShadow,0x00);    // Set Port E normal I/O
    WrPortI(PEDDR,&PEDDRShadow,0x01);  // Set Port E bits 7…1 input, 0 output
    WrPortI(PECR,&PECRShadow,0x00);    // Set transfer clock as pclk/2

    for(;;) {                          // run forever
        costate {                      // start costatement
            wfd {                 // use wfd (waitfordone) with cofunctions
              flashled[0](DS2,200,200); // flash DS2 on 200 ms, off 200 ms
              flashled[1](DS3,1000,500);// flash DS3 on 1000 ms, off 500 ms
            }
        }                               // end costatement
    }                                   // end for loop
}                                        // end of main, never come here
```

Load and run the program.

The flashing of the LEDs is performed by the costatement. Costatements need to be executed regularly, often at least every 25 ms. To accomplish this, the costatements are enclosed in a **while** loop or a **for** 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 costatement is executed on each pass through the big loop. When a **waitfor** or a **wfd** 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, 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. 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 (Rabbit Version) User's Manual*** for more details.

This program also illustrates a use for a shadow register. 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.

4.4 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 called slicing, which is further described in the ***Dynamic C (Rabbit Version) User's Manual***.

SCHEMATICS