

Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!



Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832

Email & Skype: info@chipsmall.com Web: www.chipsmall.com

Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China



RabbitCore RCM2000

C-Programmable Module

User's Manual

019-0077 • 090417-K

RabbitCore RCM2000 User's Manual

Part Number 019-0077 • 090417-K • Printed in U.S.A.

©2001–2009 Digi International Inc. • All rights reserved.

No part of the contents of this manual may be reproduced or transmitted in any form or by any means without the express written permission of Digi International.

Permission is granted to make one or more copies as long as the copyright page contained therein is included. These copies of the manuals may not be let or sold for any reason without the express written permission of Digi International.

Digi International reserves the right to make changes and improvements to its products without providing notice.

Trademarks

Rabbit and Dynamic C are registered trademarks of Digi International Inc.

Rabbit 2000 and RabbitCore are trademarks of Digi International Inc.

The latest revision of this manual is available on the Rabbit Web site, www.rabbit.com, for free, unregistered download.

Digi International Inc.

www.rabbit.com

TABLE OF CONTENTS

Chapter 1. Introduction	1
1.1 Features	1
1.2 Advantages of Using the RCM2000	2
1.3 Development and Evaluation Tools	3
1.3.1 Development Kit	3
1.3.2 Development Kit Contents	3
1.3.3 Development Software	3
1.4 How to Use This Manual	4
1.4.1 Additional Product Information	4
1.4.2 Online Documentation	4
Chapter 2. Hardware Setup	5
2.1 Connections	6
2.1.1 Alternate Power Supply Connections	8
2.2 Run a Sample Program	9
2.2.1 Troubleshooting	9
2.3 Where Do I Go From Here?	10
2.3.1 Technical Support	10
Chapter 3. Running Sample Programs	11
3.1 Sample Programs	11
3.1.1 Running Sample Program FLASHLED.C	12
3.1.1.1 Single-Stepping	13
3.1.1.2 Watch Expressions	13
3.1.1.3 Break Point	13
3.1.1.4 Editing the Program	14
3.1.1.5 Watching Variables Dynamically	14
3.1.1.6 Summary of Features	14
3.1.1.7 Cooperative Multitasking	15
3.1.1.8 Advantages of Cooperative Multitasking	17
3.1.2 Getting to Know the RCM2000	18
3.1.3 Serial Communication	21
Chapter 4. Hardware Reference	23
4.1 RCM2000 Digital Inputs and Outputs	23
4.1.1 Dedicated Inputs	27
4.1.2 Dedicated Outputs	27
4.2 Memory I/O Interface	28
4.2.1 Additional I/O	28
4.3 Serial Communication	28
4.3.1 Serial Ports	28
4.3.2 Programming Port	29
4.4 Serial Programming Cable	30
4.4.1 Changing Between Program Mode and Run Mode	30
4.4.2 Standalone Operation of the RCM2000	31

4.5 Other Hardware	32
4.5.1 Clock Doubler	32
4.5.2 Spectrum Spreader	33
4.6 Memory	34
4.6.1 SRAM	34
4.6.2 Flash EPROM	34
4.6.3 Dynamic C BIOS Source Files	34
Chapter 5. Software Reference	35
5.1 More About Dynamic C	35
5.1.1 Using Dynamic C	36
5.2 I/O	38
5.2.1 PCLK Output	38
5.3 Serial Communication Drivers	39
5.4 Upgrading Dynamic C	40
5.4.1 Extras	40
Appendix A. Specifications	41
A.1 Electrical and Mechanical Specifications	42
A.1.1 Headers	45
A.2 Bus Loading	46
A.3 Rabbit 2000 DC Characteristics	48
A.4 I/O Buffer Sourcing and Sinking Limit	49
A.5 Conformal Coating	50
A.6 Jumper Configurations	51
Appendix B. Prototyping Board	53
B.1 Overview of the Prototyping Board	54
B.2 Mechanical Dimensions and Layout	55
B.3 Power Supply	57
B.4 Using the Prototyping Board	58
B.4.1 Adding Other Components	61
Appendix C. Power Management	63
C.1 Power Supplies	63
C.1.1 Batteries and External Battery Connections	63
C.1.2 Battery-Backup Circuit	64
C.1.3 Power to VRAM Switch	65
C.1.4 Reset Generator	65
C.2 Chip Select Circuit	66
Appendix D. Sample Circuits	67
D.1 RS-232/RS-485 Serial Communication	68
D.2 Keypad and LCD Connections	69
D.3 LCD Connections	70
D.4 External Memory	71
D.5 Simple D/A Converter	72
Index	73
Schematics	75

1. INTRODUCTION

The RabbitCore RCM2000 series is a family of microprocessor modules designed to be the heart of embedded control systems, providing an array of I/O and addressing.

Throughout this manual, the term RCM2000 refers to the complete series of RCM2000 RabbitCore modules unless other production models are referred to specifically.

The RCM2000 is a core module designed to be the heart of your own controller built around the plug-in module. Data processing is done by a Rabbit 2000 microprocessor operating at up to 25.8 MHz (RCM2000 and RCM2010).

The RCM2000 has a Rabbit 2000 microprocessor, a static RAM, a flash memory, two quartz crystals (main oscillator and timekeeping), and the circuitry necessary for reset and management of battery backup of the Rabbit 2000's internal real-time clock and the static RAM. Two 40-pin headers bring out the Rabbit 2000 I/O bus, address lines, data lines, parallel ports, and serial ports.

The RCM2000 receives its +5 V power from the user board on which it is mounted. The RCM2000 can interface with all kinds of digital devices through the user board.

The RCM2000 Development Kit comes with a Prototyping Board that can be used to demonstrate the operation of the RCM2000 and to prototype new circuits.

1.1 Features

- Small size: 1.90" × 2.30" (48.3 mm × 58.4 mm)
- Microprocessor: Rabbit 2000 running at 25.8 MHz (RCM2000 and RCM2010)
- 40 CMOS-compatible parallel I/O lines grouped in five 8-bit ports (shared with serial ports)
- 8 data lines (D0–D7)
- 13 address lines (A0–A12)
- I/O read, write, buffer enable
- Status, watchdog and clock outputs
- Two startup mode inputs for master/slave configuration

- External reset input
- Reset output
- Five 8-bit timers, two 10-bit timers; five timers are cascadable in pairs
- 256K flash EPROM, 512K SRAM
- Real-time clock
- Watchdog supervisor
- Provision for customer-supplied backup battery via connections on header J2
- Four CMOS-compatible serial ports: maximum asynchronous baud rate of 806,400 bps, maximum synchronous baud rate of 6.45 Mbps. Two ports are configurable as clocked ports.

Appendix A, “Specifications,” provides detailed specifications for the RCM2000.

Three versions of the RCM2000 are available. Their standard features are summarized in Table 1.

Table 1. RCM2000 Models and Features

Model	Features
RCM2000	Full-featured RCM2000 module with 25.8 MHz clock, 256K flash memory, and 512K SRAM
RCM2010	RCM2000 with 25.8 MHz clock and 128K SRAM
RCM2020	RCM2000 with 18.432 MHz clock and 128K SRAM

1.2 Advantages of Using the RCM2000

- Fast design time for your project since the basic core has already been designed and built.
- Competitive pricing compared with purchasing and assembling the individual components.
- Easy programming, including production installation of a program.
- Generous memory size allows large C programs with tens of thousands of lines of code, and substantial data storage.

1.3 Development and Evaluation Tools

1.3.1 Development Kit

A complete Development Kit, including a Prototyping Board and Dynamic C development software, is available for the RCM2000. The Development Kit puts together the essentials you need to design an embedded microprocessor-based system rapidly and efficiently.

1.3.2 Development Kit Contents

The RCM2000 Development Kit contains the following items:

- RCM2020 module with 256K flash memory and 128K SRAM.
- RCM2000 Prototyping Board with accessory hardware and components.
- Universal AC adapter, 12 V DC, 1 A (includes Canada/Japan/U.S., Australia/N.Z., U.K., and European style plugs).
- 10-pin header to DB9 programming cable with integrated level-matching circuitry.
- *Dynamic C* CD-ROM, with complete product documentation on disk.
- *Getting Started* instructions.
- Registration card.

1.3.3 Development Software

The RCM2000 modules use the Dynamic C development environment for rapid creation and debugging of runtime applications. Dynamic C provides a complete development environment with integrated editor, compiler and source-level debugger. It interfaces directly with the target system, eliminating the need for complex and unreliable in-circuit emulators.

1.4 How to Use This Manual

This user's manual is intended to give users detailed information on the RCM2000 module. It does not contain detailed information on the Dynamic C development environment.

1.4.1 Additional Product Information

In addition to the product-specific information contained in the *RabbitCore RCM2000 User's Manual* (this manual), several higher level reference manuals are provided in HTML and PDF form on the accompanying CD-ROM. Advanced users will find these references valuable in developing systems based on the RCM3100 modules:

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

1.4.2 Online Documentation

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

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



2. HARDWARE SETUP

This chapter describes the RCM2000 hardware in more detail, and explains how to set up the accompanying Prototyping Board.

NOTE: This chapter (and this manual) assume that you have the RabbitCore RCM2000 Development Kit. If you purchased an RCM2000 module by itself, you will have to adapt the information in this chapter and elsewhere to your test and development setup.

2. Connect RCM2000 to PC

Connect the 10-pin connector of the programming cable labeled **PROG** to header J3 on the RCM2000 module as shown in Figure 2 below. Be sure to orient the red edge of the cable towards pin 1 of the connector. (Do not use the **DIAG** connector, which is used for a normal serial connection.)

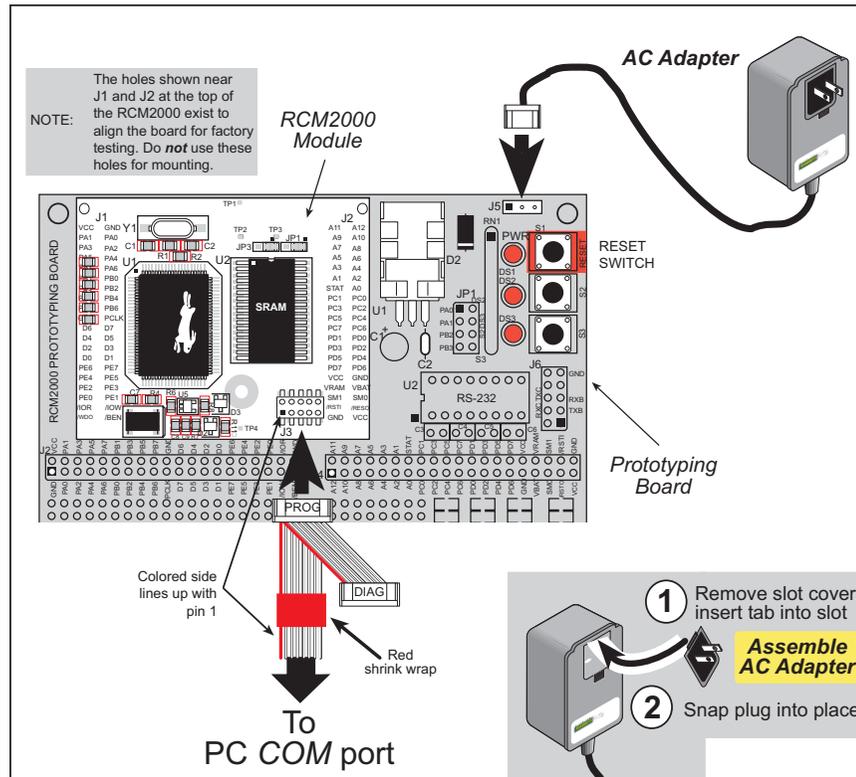


Figure 2. RCM2000 Power and Programming Connections

NOTE: Some PCs now come equipped only with a USB port. It may be possible to use an RS-232/USB converter (Part No. 20-151-0178) with the programming cable supplied with the RCM2000 Development Kit. Note that not all RS-232/USB converters work with Dynamic C.

3. Power Supply Connections

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

First, prepare the AC adapter for the country where it will be used by selecting the plug. The RCM2000 Development Kit presently includes Canada/Japan/U.S., Australia/N.Z., U.K., and European style plugs. Snap in the top of the plug assembly into the slot at the top of the AC adapter as shown in Figure 2, then press down on the spring-loaded clip below the plug assembly to allow the plug assembly to click into place.

Connect the AC adapter to 3-pin header J5 on the Prototyping Board. The connector may be attached either way as long as it is not offset to one side.

Plug in the AC adapter. The power LED on the Prototyping Board should light up. The RCM2000 and the Prototyping Board are now ready to be used.

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

To power down the Prototyping Board, unplug the power connector from J5. You should disconnect power before making any circuit adjustments in the prototyping area, changing any connections to the board, or removing the RCM2020 from the Prototyping Board.

2.1.1 Alternate Power Supply Connections

Development kits sold outside North America before 2009 included a header connector that could be connected to 3-pin header J5 on the Prototyping Board. The red and black wires from the connector could then be connected to the positive and negative connections on your power supply. The power supply should deliver 8 V–24 V DC at 8 W.

2.2 Run a Sample Program

Once the RCM2000 is connected as described in the preceding pages, start Dynamic C by double-clicking on the Dynamic C icon on your desktop or in your **Start** menu. Dynamic C uses the serial port specified during installation.

If you are using a USB port to connect your computer to the RCM2000 module, choose **Options > Project Options** and select “Use USB to Serial Converter” under the **Communications** tab, then click **OK**.

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), then compile and run it by pressing **F9** or by selecting **Run** in the **Run** menu. The **STDIO** window will open and will display a small square bouncing around in a box.

2.2.1 Troubleshooting

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

- Check that the RCM2000 is powered correctly — the red power LED on the Prototyping Board should be lit when the RCM2000 is mounted on the Prototyping Board and the AC adapter is plugged in.
- Check both ends of the programming cable to ensure that they are firmly plugged into the PC and the **PROG** connector, not the **DIAG** connector, is plugged in to the programming port on the RCM2000 with the marked (colored) edge of the programming cable towards pin 1 of the programming header.
- Ensure that the RCM2000 module is firmly and correctly installed in its connectors on the Prototyping Board.
- Dynamic C uses the COM port specified during installation. Select a different COM port within Dynamic C. From the **Options** menu, select **Project Options**, then select **Communications**. Select another COM port from the list, then click **OK**. Press **<Ctrl-Y>** to force Dynamic C to recompile the BIOS. If Dynamic C still reports it is unable to locate the target system, repeat the above steps until you locate the COM port used by the programming cable.

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

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

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

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

2.3 Where Do I Go From Here?

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

1. Run all of the sample programs described in Chapter 3 to get a basic familiarity with Dynamic C and the RCM2000's capabilities.
2. For further development, refer to the *RabbitCore RCM2000 User's Manual* for details of the module's hardware and software components.

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

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

2.3.1 Technical Support

NOTE: If you purchased your RCM2000 through a distributor or through a Rabbit partner, contact the distributor or partner first for technical support.

- Use the Dynamic C **Help** menu to get further assistance with Dynamic C.
- Check the Rabbit Technical Bulletin Board and forums at www.rabbit.com/support/bb/ and at www.rabbit.com/forums/.
- Use the Technical Support e-mail form at www.rabbit.com/support/.

3. RUNNING SAMPLE PROGRAMS

To develop and debug programs for the RCM2000 (and for all other Rabbit hardware), you must install and use 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 Rabbit single-board computers and other single-board computers based on the Rabbit microprocessor. Chapter 3 provides the sample programs related to the RCM2000.

3.1 Sample Programs

To help familiarize you with the RCM2000 modules, Dynamic C includes several sample programs in the Dynamic C `SAMPLES\RCM2000` directory. Loading, executing and studying these programs will give you a solid hands-on overview of the RCM2000's capabilities, as well as a quick start with Dynamic C as an application development tool. These programs are intended to serve as tutorials, but then can also be used as starting points or building blocks for your own applications.

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

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

Before running any of these sample program, make sure that your RCM2000 is connected to the Prototyping Board and to your PC as described in Section 2.1, "Connections."

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

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

3.1.1 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/RCM2000** 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.

```
main() {  
    int j;  
    WrPortI (SPCR, &SPCRShadow, 0x84);  
    WrPortI (PADR, &PADRShadow, 0xFF);  
    while(1) {  
        BitWrPortI (PADR, &PADRShadow, 1, 1);  
        for(j=0; j<32000; j++);  
        BitWrPortI (PADR, &PADRShadow, 0, 1);  
        for(j=0; j<25000; j++);  
    } // end while  
} // end of main
```

C programs begin with main

Set up Port A to output to LED DS2 and DS3

Start a loop

Turn LED DS3 off

Time delay by counting to 32,000

Turn LED DS3 on

Time delay by counting to 25,000

End of the endless loop

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

Figure 3. Sample Program FLASHLED.C

To run the program **FLASHLED.C**, open it with the **File** menu (if it is not already open), then compile and run it by pressing **F9** or by selecting **Run** in the **Run** menu. 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 RCM2000 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 RCM2000. (The colored wire on the programming cable is closest to pin 1 on header J3 on the RCM2000, 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.

3.1.1.1 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.

3.1.1.2 Watch Expressions

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.

3.1.1.3 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.

3.1.1.4 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.

3.1.1.5 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**.

3.1.1.6 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 **runwatch()** 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.

3.1.1.7 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 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 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

```

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 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.

3.1.1.8 Advantages of Cooperative Multitasking

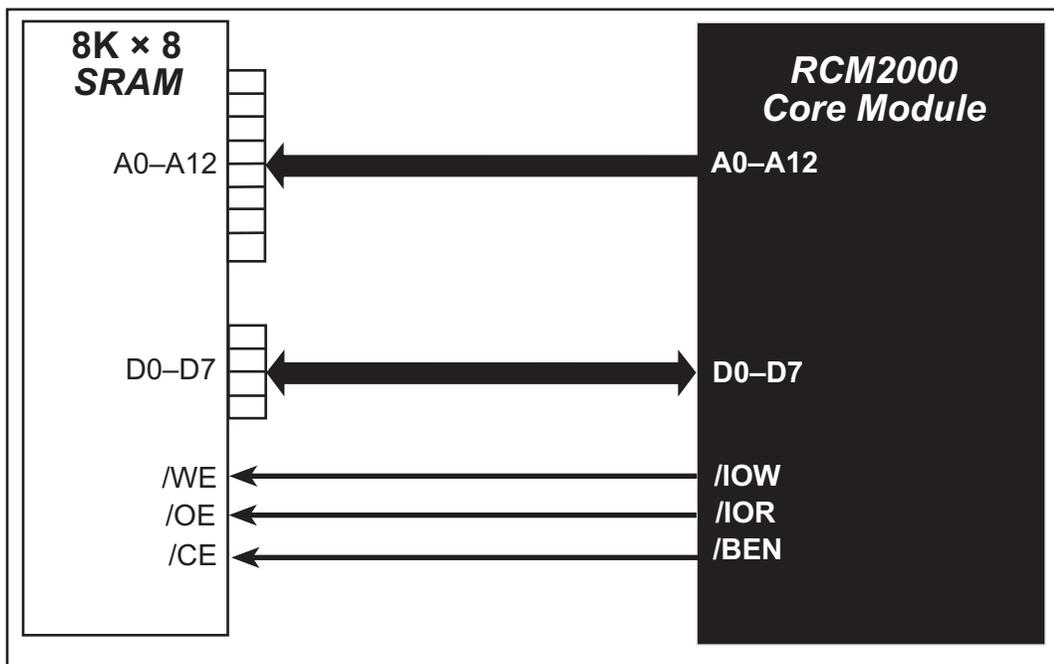
Cooperative multitasking, as implemented with language extensions, has the advantage of being intuitive. Unlike preemptive multitasking, variables can be shared between different tasks without having to take elaborate precautions. Sharing variables between tasks is the greatest cause of bugs in programs 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 User's Manual*.

3.1.2 Getting to Know the RCM2000

The following sample programs can be found in the `SAMPLES\RCM2000` folder.

- **EXTSRAM.C**—demonstrates the setup and simple addressing to an external SRAM. This program first maps the external SRAM to the I/O Bank 0 register with a maximum of 15 wait states, chip select strobe (which is ignored because of the circuitry), and allows writes. The first 256 bytes of SRAM are cleared and read back. Values are then written to the same area and are read back. The Dynamic C **STDIO** window will indicate if writes and reads did not occur

Connect an external SRAM as shown below before you run this sample program.



- **FLASHLED.C**—repeatedly flashes LED DS3 on the Prototyping Board on and off. LED DS3 is controlled by Parallel Port A bit 1 (PA1).
- **FLASHLED2.C**—repeatedly flashes LED DS3 on the Prototyping Board on and off. LED DS3 is controlled by Parallel Port A bit 1 (PA1).

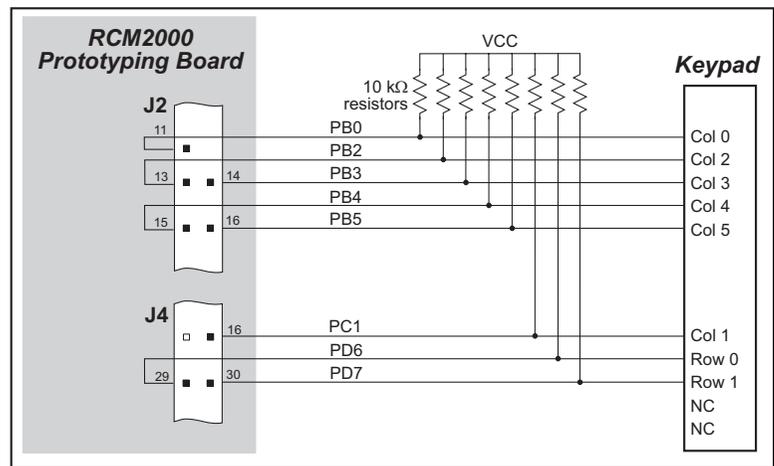
This sample program also shows the use of the `runwatch()` function to allow Dynamic C to update watch expressions while running. The following steps explain how to do this.

1. Add a watch expression for "k" in the **Inspect > Add Watch** dialog box.
2. Click "Add" or "Add to top" so that it will be in the watch list permanently.
3. Click **OK** to close the dialog box.
4. Press **<Ctrl+U>** while the program is running. This will update the watch window

- **FLASHLEDS.C**—demonstrates the use of coding with assembly instructions, cofunctions, and costatements to flash LEDs DS2 and DS3 on the Prototyping Board on and off. LEDs DS2 and DS3 are controlled by Parallel Port A bit 0 (PA0) and Parallel Port A bit 1 (PA1). Once you have compile this program and it is running, LEDs DS2 and DS3 will flash on/off at different rates.
- **FLASHLEDS2.C**—demonstrates the use of cofunctions and costatements to flash LEDs DS2 and DS3 on the Prototyping Board on and off. LEDs DS2 and DS3 are controlled by Parallel Port A bit 0 (PA0) and Parallel Port A bit 1 (PA1). Once you have compile this program and it is running, LEDs DS2 and DS3 will flash on/off at different rates.
- **KEYLCD.C**—demonstrates a simple setup for a 2 × 6 keypad and a 2 × 20 LCD.

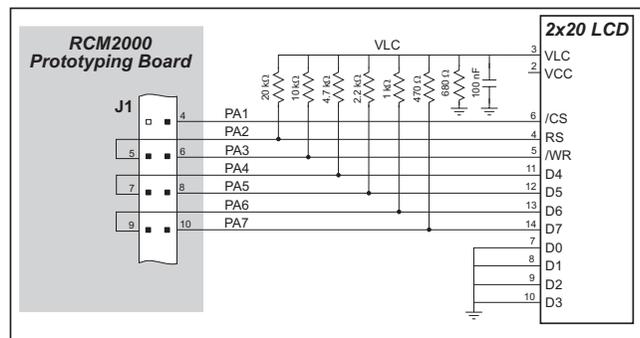
Connect the keypad to Parallel Ports B, C, and D.

- PB0—Keypad Col 0
- PC1—Keypad Col 1
- PB2—Keypad Col 2
- PB3—Keypad Col 3
- PB4—Keypad Col 4
- PB5—Keypad Col 5
- PD6—Keypad Row 0
- PD7—Keypad Row 1



Connect the LCD to Parallel Port A.

- PA0—backlight (if connected)
- PA1—LCD /CS
- PA2—LCD RS (High = Control, Low = Data) / LCD Contrast 0
- PA3—LCD /WR / LCD Contrast 1
- PA4—LCD D4 / LCD Contrast 2
- PA5—LCD D5 / LCD Contrast 3
- PA6—LCD D6 / LCD Contrast 4
- PA7—LCD D7 / LCD Contrast 5

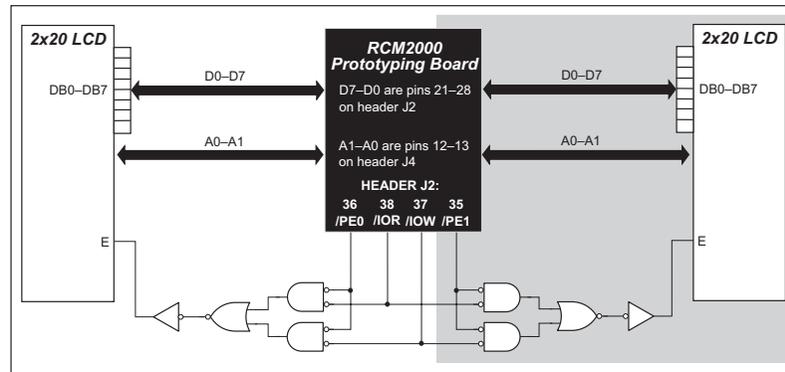


Once the connections have been made and the sample program is running, the LCD will display two rows of 6 dots, each dot representing the corresponding key. When a key is pressed, the corresponding dot will become an asterisk.

- **LCD_DEMO.C**—demonstrates a simple setup for an LCD that uses the HD44780 controller or an equivalent.

Connect the LCD to the RCM2000 address and data lines on the Prototyping Board.

D0—DB0
 D1—DB1
 D2—DB2
 D3—DB3
 D4—DB4
 D5—DB5
 D6—DB6
 D7—DB7



A0—RS (Register Select: 0 = command, 1 = data)

A1—R/W (0=write, 1=read)

*—E (normally low: latches on high-to-low transition)

- **SWTEST.C**—demonstrates the use of pushbutton switches S2 and S3 to toggle LEDs DS2 and DS3 on the Prototyping Board on and off.

Parallel Port A bit 0 = LED DS2

Parallel Port A bit 1 = LED DS3

Parallel Port B bit 2 = switch S2

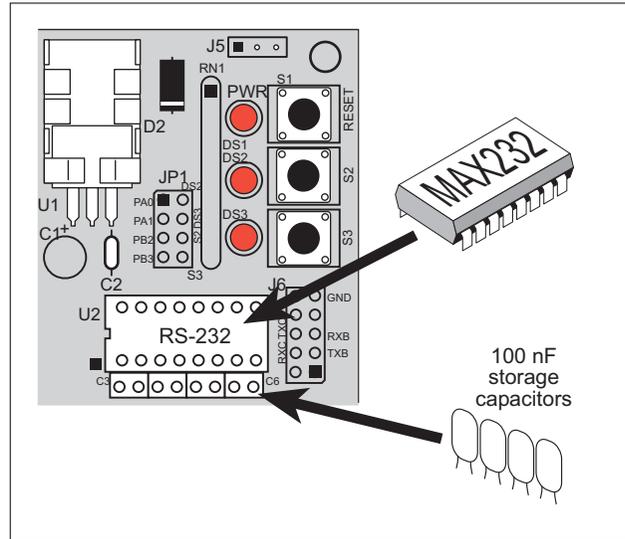
Parallel Port B bit 3 = switch S3

- **TOGGLELED.C**—demonstrates the use of costatements to detect switch presses using the press-and-release method of debouncing. As soon as the sample program starts running, LED DS3 on the Prototyping Board (which is controlled by PA1) starts flashing once per second. Press switch S2 on the Prototyping Board (which is connected to PB2) to toggle LED DS2 on the Prototyping Board (which is controlled by PA0). The push-button switch is debounced by the software.

3.1.3 Serial Communication

The following sample programs can be found in the **SAMPLES\RCM2000** folder.

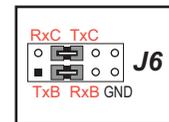
Two sample programs, **CORE_FLOWCONTROL.C** and **CORE_PARITY.C**, are available to illustrate RS-232 communication. To run these sample programs, you will have to add an RS-232 transceiver such as the MAX232 at location U2 and four 100 nF charge-storage capacitors at C3–C6 on the Prototyping Board. Also install the 2 × 5 IDC header included with the Prototyping Board accessory parts at J6 to interface the RS-232 signals.



The diagram shows the connections.

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

To set up the Prototyping Board, you will need to tie PC4 and PC5 (TxB and RxB) together at header J4, and you will also tie PC2 and PC3 (TxC and RxC) together as shown in the diagram.

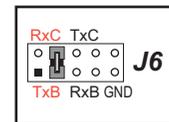


A repeating triangular pattern should print out in the **STDIO** window. The program will periodically switch flow control on or off to demonstrate the effect of no flow control.

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

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

To set up the Prototyping Board, you will need to tie PC4 and PC3 (TxB and RxC) together at header J4 as shown in the diagram.



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