Digi - 20-101-1094 Datasheet

Welcome to E-XFL.COM

Understanding Embedded - Microcontroller, Microprocessor, FPGA Modules

Embedded - Microcontroller, Microprocessor, and FPGA Modules are fundamental components in modern electronic systems, offering a wide range of functionalities and capabilities. Microcontrollers are compact integrated circuits designed to execute specific control tasks within an embedded system. They typically include a processor, memory, and input/output peripherals on a single chip. Microprocessors, on the other hand, are more powerful processing units used in complex computing tasks, often requiring external memory and peripherals. FPGAs (Field Programmable Gate Arrays) are highly flexible devices that can be configured by the user to perform specific logic functions, making them invaluable in applications requiring customization and adaptability.

Applications of Embedded - Microcontroller,

Details	
Product Status	Obsolete
Module/Board Type	MPU Core
Core Processor	Rabbit 4000
Co-Processor	-
Speed	58.98MHz
Flash Size	512KB (Internal), 32MB (External)
RAM Size	512KB
Connector Type	IDC Header 2x25, 2x5
Size / Dimension	1.84" x 2.42" (47mm x 61mm)
Operating Temperature	0°C ~ 70°C
Purchase URL	https://www.e-xfl.com/product-detail/digi-international/20-101-1094

- **`LOW_POWER.C`**—demonstrates how to implement a function in RAM to reduce power consumption by the Rabbit microprocessor. There are four features that lead to the lowest possible power draw by the microprocessor.

  1. Run the CPU from the 32 kHz crystal.
  2. Turn off the high-frequency crystal oscillator.
  3. Run from RAM.
  4. Ensure that internal I/O instructions do not use CS0.

  Once you are ready to compile and run this sample program, use **<Alt-F9>** instead of just **F9**. This will disable polling, which will allow Dynamic C to continue debugging once the target starts running off the 32 kHz oscillator.

  This sample program will toggle LEDs DS2 and DS3 on the Prototyping Board. You may use an oscilloscope. DS2 will blink the fastest. After switching to low power, both LEDs will blink together.

- **`TAMPERDETECTION.C`**—demonstrates how to detect an attempt to enter the bootstrap mode. When an attempt is detected, the battery-backed onchip-encryption RAM on the Rabbit 4000 is erased. This battery-backed onchip-encryption RAM can be useful to store data such as an AES encryption key from a remote location.

  This sample program shows how to load and read the battery-backed onchip-encryption RAM and how to enable a visual indicator.

  Once this sample is compiled running (you have pressed the **F9** key while the sample program is open), remove the programming cable and press the reset button on the Prototyping Board to reset the module. LEDs DS2 and DS3 will be flashing on and off.

  Now press switch S2 to load the battery-backed RAM with the encryption key. The LEDs are now on continuously. Notice that the LEDs will stay on even when you press the reset button on the Prototyping Board.

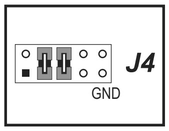
  Reconnect the programming cable briefly and unplug it again. The LEDs will be flashing because the battery-backed onchip-encryption RAM has been erased. Notice that the LEDs will continue flashing even when you press the reset button on the Prototyping Board.

  You may press switch S2 again and repeat the last steps to watch the LEDs.

- **`TOGGLESWITCH.C`**—demonstrates the use of costatements to detect switch presses using the press-and-release method of debouncing. LEDs DS2 and DS3 on the Prototyping Board are turned on and off when you press switches S2 and S3. S2 and S3 are controlled by PB4 and PB5 respectively.

Once you have loaded and executed these five programs and have an understanding of how Dynamic C and the RCM4000 modules interact, you can move on and try the other sample programs, or begin building your own.

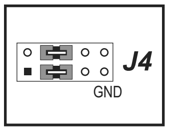
- **SIMPLE3WIRE.C**—This program demonstrates basic RS-232 serial communication. Lower case characters are sent by TxC, and are received by RxD. The characters are converted to upper case and are sent out by TxD, are received by RxC, and are displayed in the Dynamic C **STDIO** window.



  To set up the Prototyping Board, you will need to tie TxD and RxC together on the RS-232 header at J4, and you will also tie RxD and TxC together using the jumpers supplied in the Development Kit as shown in the diagram.

- **SIMPLE5WIRE.C**—This program demonstrates 5-wire RS-232 serial communication with flow control on Serial Port D and data flow on Serial Port C.

  To set up the Prototyping Board, you will need to tie TxD and RxD together on the RS-232 header at J4, and you will also tie TxC and RxC together using the jumpers supplied in the Development Kit as shown in the diagram.

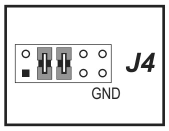


  Once you have compiled and run this program, you can test flow control by disconnecting TxD from RxD while the program is running. Characters will no longer appear in the **STDIO** window, and will display again once TxD is connected back to RxD.

  If you have two Prototyping Boards with modules, run this sample program on the sending board, then disconnect the programming cable and reset the sending board so that the module is operating in the Run mode. Connect TxC, TxD, and GND on the sending board to RxC, RxD, and GND on the other board, then, with the programming cable attached to the other module, run the sample program. Once you have compiled and run this program, you can test flow control by disconnecting TxD from RxD as before while the program is running.

- **SWITCHCHAR.C**—This program demonstrates transmitting and then receiving an ASCII string on Serial Ports C and D. It also displays the serial data received from both ports in the **STDIO** window.

  To set up the Prototyping Board, you will need to tie TxD and RxC together on the RS-232 header at J4, and you will also tie RxD and TxC together using the jumpers supplied in the Development Kit as shown in the diagram.



  Once you have compiled and run this program, press and release switches S2 and S3 on the Prototyping Board. The data sent between the serial ports will be displayed in the **STDIO** window.

- **IOCONFIG_SWITCHECHO.C**—This program demonstrates how to set up Serial Port F, which then transmits or receives an ASCII string to/from Serial Port D when switch S2 or S3 is pressed. The echoed serial data are displayed in the Dynamic C **STDIO** window.

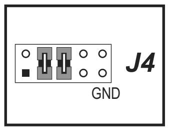
  Note that the I/O lines that carry the Serial Port F signals are not the Rabbit 4000 defaults. The Serial Port F I/O lines are configured by calling the library function **ser-Fconfig()** that was generated by the Rabbit 4000 **IOCONFIG.EXE** utility program.

  Serial Port F is configured to use Parallel Port C bits PC2 and PC3. These signals are available on the Prototyping Board's RS-232 connector (header J4).

  Serial Port D is left in its default configuration, using Parallel Port C bits PC0 and PC1. These signals are available on the Prototyping Board's RS-232 connector (header J4).

  Also note that there is one library generated by **IOCONFIG.EXE** in the Dynamic C **SAMPLES\RCM4000\SERIAL** folder for the 58 MHz RCM4100 and RCM4010.

  To set up the Prototyping Board, you will need to tie TxD and RxC together and tie TxC and RxD together on the RS-232 header at J4 using the jumpers supplied in the Development Kit. (Remember that RxC and TxC now are actually RxF and TxF.)

  Once you have compiled and run this program, press and release switches S2 or S3 on the Prototyping Board. The data echoed between the serial ports will be displayed in the **STDIO** window.

- Baud rate 19,200 bps, 8 bits, no parity, 1 stop bit
- Enable **Local Echo** option
- Feed options — **Receive = CR**, **Transmit = CR + LF**

Follow the remaining steps carefully in Tera Term to avoid overwriting previously saved calibration data when using same the file name.

- Enable the **File APPEND** option at the bottom of the dialog box
- Select the **OPEN** option at the right-hand side of the dialog box

Tera Term is now ready to log all data received on the serial port to the file you specified.

You are now ready to compile and run this sample program. A message will be displayed in the Tera Term display window once the sample program is running.

Enter the serial number of your RabbitCore module in the Tera Term display window, then press the **ENTER** key. The Tera Term display window will now display the calibration data.

Now select **CLOSE** from within the Tera Term LOG window, which will likely be a separate pop-up window minimized at the bottom of your PC screen. This finishes the logging and closes the file.

Open your data file and verify that the calibration data have been written properly. A sample is shown below.

```
Serial port transmission
========================
Uploading calibration table . . .
Enter the serial number of your controller = 9MN234
SN9MN234
ADSE
0
float_gain,float_offset,float_gain,float_offset,float_gain,float_offset,float_gain,float_offset,
float_gain,float_offset,float_gain,float_offset,float_gain,float_offset,float_gain,float_offset,
1
float_gain,float_offset,float_gain,float_offset,float_gain,float_offset,float_gain,float_offset,
float_gain,float_offset,float_gain,float_offset,float_gain,float_offset,float_gain,float_offset,
          |
          |
ADDF
0
float_gain,float_offset,float_gain,float_offset,float_gain,float_offset,float_gain,float_offset,
float_gain,float_offset,float_gain,float_offset,float_gain,float_offset,float_gain,float_offset,
2
float_gain,float_offset,float_gain,float_offset,float_gain,float_offset,float_gain,float_offset,
float_gain,float_offset,float_gain,float_offset,float_gain,float_offset,float_gain,float_offset,
          |
          |
ADMA
3
float_gain,float_offset,
4
float_gain,float_offset,
          |
          |
END
```

4.1 RCM4000 Digital Inputs and Outputs

Figure 6 shows the RCM4000 pinouts for header J3.

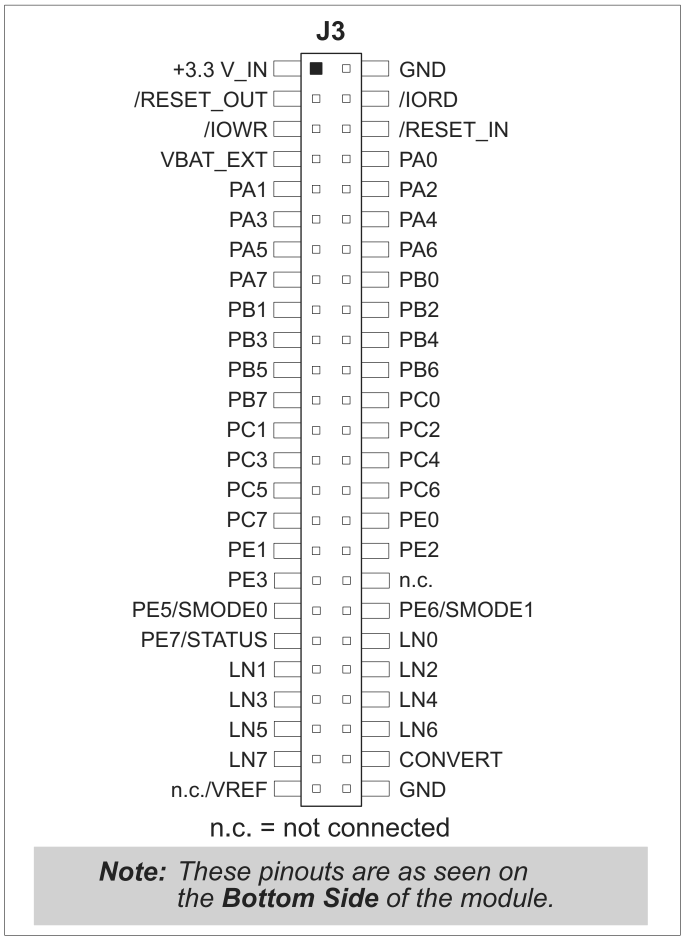


Figure 6. RCM4000 Pinout

Headers J3 is a standard 2 × 25 IDC header with a nominal 1.27 mm pitch.

Figure 7 shows the use of the Rabbit 4000 microprocessor ports in the RCM4000 modules.

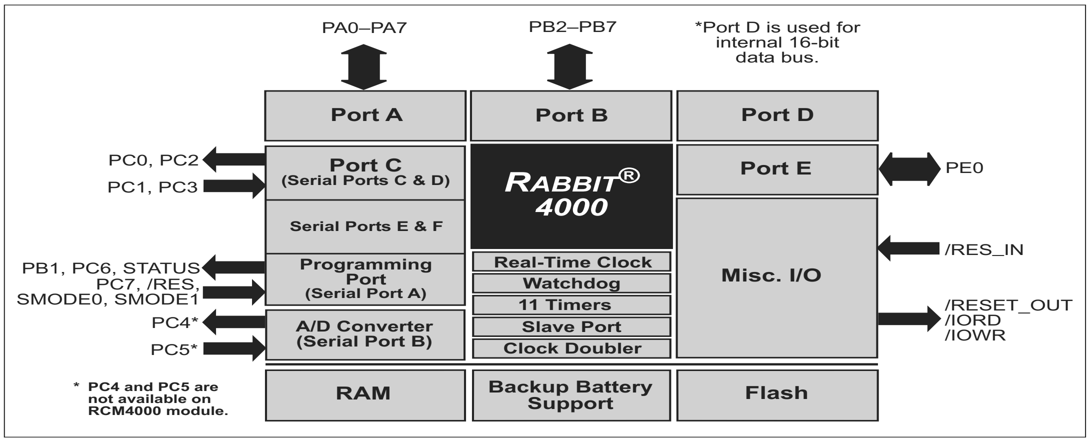


Figure 7. Use of Rabbit 4000 Ports

The ports on the Rabbit 4000 microprocessor used in the RCM4000 are configurable, and so the factory defaults can be reconfigured. Table 2 lists the Rabbit 4000 factory defaults and the alternate configurations.

Table 2. RCM4000 Pinout Configurations (continued)

Pin	Pin Name	Default Use	Alternate Use	Notes
49	VREF	Analog reference voltage		1.15 V/2.048 V/2.500 V on-chip ref. voltage (RCM4000 only)
50	GND	Ground		Ground

* PE5, PE6, and PE7 are used for the Ethernet clock and I/O signals, which ordinarily would not be routed to a general-purpose I/O header to minimize noise. Therefore, the RCM4000 RabbitCore modules present the SMODE and STATUS lines to header J3.

4.1.1 Memory I/O Interface

The Rabbit 4000 address lines (A0–A19) and all the data lines (D0–D7) are routed internally to the onboard flash memory and SRAM chips. I/0 write (/IOWR) and I/0 read (/IORD) are available for interfacing to external devices. Parallel Port D is used for the upper byte of the 16-bit memories.

Parallel Port A can also be used as an external I/O data bus to isolate external I/O from the main data bus. Parallel Port B pins PB2–PB7 can also be used as an auxiliary address bus.

When using the auxiliary I/O bus for any reason, you must add the following line at the beginning of your program.

```
#define PORTA_AUX_IO    // required to enable auxiliary I/O bus
```

Selected pins on Parallel Port E as specified in Table 2 may be used for input capture, quadrature decoder, DMA, and pulse-width modulator purposes.

4.1.2 Other Inputs and Outputs

The status and the two SMODE pins, SMODE0 and SMODE1, can be brought out to header J3 instead of PE5–PE7 as explained in Appendix A.6.

/RESET_IN is normally associated with the programming port, but may be used as an external input to reset the Rabbit 4000 microprocessor and the RCM4000 memory. /RESET_OUT is an output from the reset circuitry that can be used to reset other peripheral devices.

5. Software Reference

Dynamic C is an integrated development system for writing embedded software. It runs on an IBM-compatible PC and is designed for use with single-board computers and other devices based on the Rabbit microprocessor. Chapter 5 describes the libraries and function calls related to the RCM4000.

5.1 More About Dynamic C

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

You have a choice of doing your software development in the flash memory or in the static SRAM included on the RCM4000. The flash memory and SRAM options are selected with the **Options > Program Options > Compiler** menu.

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

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

NOTE: Do not depend on the flash memory sector size or type in your program logic. The RCM4000 and Dynamic C were designed to accommodate flash devices with various sector sizes in response to the volatility of the flash-memory market.

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

5.2 Dynamic C Function Calls

5.2.1 Digital I/O

The RCM4000 was designed to interface with other systems, and so there are no drivers written specifically for the I/O. The general Dynamic C read and write functions allow you to customize the parallel I/O to meet your specific needs. For example, use

```
WrPortI(PEDDR, &PEDDRShadow, 0x00);
```

to set all the Port E bits as inputs, or use

```
WrPortI(PEDDR, &PEDDRShadow, 0xFF);
```

to set all the Port E bits as outputs.

When using the auxiliary I/O bus on the Rabbit 4000 chip, add the line

```
#define PORTA_AUX_IO    // required to enable auxiliary I/O bus
```

to the beginning of any programs using the auxiliary I/O bus.

The sample programs in the Dynamic C **`SAMPLES/RCM4000`** folder provide further examples.

5.2.2 Serial Communication Drivers

Library files included with Dynamic C provide a full range of serial communications support. The **`RS232.LIB`** library provides a set of circular-buffer-based serial functions. The **`PACKET.LIB`** library provides packet-based serial functions where packets can be delimited by the 9th bit, by transmission gaps, or with user-defined special characters. Both libraries provide blocking functions, which do not return until they are finished transmitting or receiving, and nonblocking functions, which must be called repeatedly until they are finished, allowing other functions to be performed between calls. For more information, see the ***Dynamic C Function Reference Manual*** and Rabbit's Technical Note TN213, ***Rabbit Serial Port Software***.

5.2.3 SRAM Use

The RCM4000 module has a battery-backed data SRAM. Dynamic C provides the **`protected`** keyword to identify variables that are to be placed into the battery-backed SRAM. Such a variable is protected against loss in case of a power failure or other system reset because the compiler generates code that creates a backup copy of a protected variable before the variable is modified. If the system resets while the protected variable is being modified, the variable's value can be restored when the system restarts. This operation requires battery-backed RAM and the main system clock. If you are using the 32 kHz clock you must switch back to the main system clock to use protected variables because the atomicity of the write cannot be ensured when the 32 kHz clock is being used.

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

```
main() {
   protected int state1, state2, state3;
   ...
   _sysIsSoftReset();    // restore any protected variables
```

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

`anaInConfig (continued)`

PARAMETERS

`instructionbyte` the instruction byte that will initiate a read or write operation at 8 or 16 bits on the designated register address. For example,

```
checkid = anaInConfig(0x5F, 0, 9600);
   // read ID and set baud rate
```

`cmd` the command data that configure the registers addressed by the instruction byte. Enter 0 if you are performing a read operation. For example,

```
i = anaInConfig(0x07, 0x3b, 0);
   // write ref/osc reg and enable
```

`brate` the serial clock transfer rate of 9600 to 115,200 bytes per second. **`brate`** must be set the first time this function is called. Enter 0 for this parameter thereafter, for example,

```
anaInConfig(0x00, 0x00, 9600);
   // resets device and sets byte rate
```

RETURN VALUE

0 on write operations

data value on read operations

SEE ALSO

`anaInDriver`, **`anaIn`**, **`brdInit`**

anaInDiff (continued)

RETURN VALUE

A voltage value corresponding to the voltage differential on the analog input channel.

`ADTIMEOUT` (-4095) if the conversion is incomplete or busy bit timeout.

`ADOVERFLOW` (-4096) for overflow or out of range.

SEE ALSO

`anaInCalib`, **`anaIn`**, **`anaInmAmps`**, **`brdInit`**

anaInEERd

```
root int anaInEERd(unsigned int channel, unsigned int opmode,
    unsigned int gaincode);
```

DESCRIPTION

Reads the calibration constants, gain, and offset for an input based on their designated position in the flash memory, and places them into global tables **`_adcCalibS`**, **`_adcCalibD`**, and **`_adcCalibM`** for analog inputs. Depending on the flash size, the following macros can be used to identify the starting address for these locations.

`ADC_CALIB_ADDRS`, address start of single-ended analog input channels

`ADC_CALIB_ADDRD`, address start of differential analog input channels

`ADC_CALIB_ADDRM`, address start of milliamp analog input channels

NOTE: This function cannot be run in RAM.

PARAMETER

`channel`	the channel number (0 to 7) corresponding to LN0 to LN7.
`opmode`	the mode of operation: **`SINGLE`**—single-ended input **`DIFF`**—differential input **`mAMP`**—4–20 mA input

`channel`	**`SINGLE`**	**`DIFF`**	**`mAMP`**
0	+AIN0	+AIN0 -AIN1	+AIN0*
1	+AIN1	+AIN1 -AIN0*	+AIN1*
2	+AIN2	+AIN2 -AIN3	+AIN2*
3	+AIN3	+AIN3 -AIN2*	+AIN3
4	+AIN4	+AIN4 -AIN5	+AIN4
5	+AIN5	+AIN5 -AIN4*	+AIN5
6	+AIN6	+AIN6 -AIN7*	+AIN6
7	+AIN7	+AIN7 -AIN6*	+AIN7*
`ALLCHAN`	read all channels for selected **`opmode`**		

* Not accessible on Prototyping Board.

6.5 Run the `PINGME.C` Sample Program

Connect the crossover cable from your computer's Ethernet port to the RCM4000 module's RJ-45 Ethernet connector. Open this sample program from the **`SAMPLES\TCPIP\ICMP`** folder, compile the program, and start it running under Dynamic C. The crossover cable is connected from your computer's Ethernet adapter to the RCM4000 module's RJ-45 Ethernet connector. When the program starts running, the green **`LINK`** light on the RCM4000 module should be on to indicate an Ethernet connection is made. (Note: If the **`LNK`** light does not light, you may not be using a crossover cable, or if you are using a hub with straight-through cables perhaps the power is off on the hub.)

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

```
ping 10.10.6.101
```

or by **Start > Run**

and typing the entry

```
ping 10.10.6.101
```

Notice that the yellow **`ACT`** light flashes on the RCM4000 module while the ping is taking place, and indicates the transfer of data. The ping routine will ping the module four times and write a summary message on the screen describing the operation.

6.6 Running Additional Sample Programs With Direct Connect

The following sample programs are in the Dynamic C **`SAMPLES\RCM4000\TCPIP\`** folder.

- **`BROWSELED.C`**—This program demonstrates a basic controller running a Web page. Two "device LEDs" are created along with two buttons to toggle them. Users can use their Web browser to change the status of the lights. The DS2 and DS3 LEDs on the Prototyping Board will match those on the Web page. As long as you have not modified the **`TCPCONFIG 1`** macro in the sample program, enter the following server address in your Web browser to bring up the Web page served by the sample program.

  http://10.10.6.100.

  Otherwise use the TCP/IP settings you entered in the **`TCP_CONFIG.LIB`** library.
- **`PINGLED.C`**—This program demonstrates ICMP by pinging a remote host. It will flash LEDs DS2 and DS3 on the Prototyping Board when a ping is sent and received.
- **`SMTP.C`**—This program demonstrates using the SMTP library to send an e-mail when the S2 and S3 switches on the Prototyping Board are pressed. LEDs DS2 and DS3 on the Prototyping Board will light up when e-mail is being sent.

A.6 Jumper Configurations

Figure A-6 shows the header locations used to configure the various RCM4000 options via jumpers.

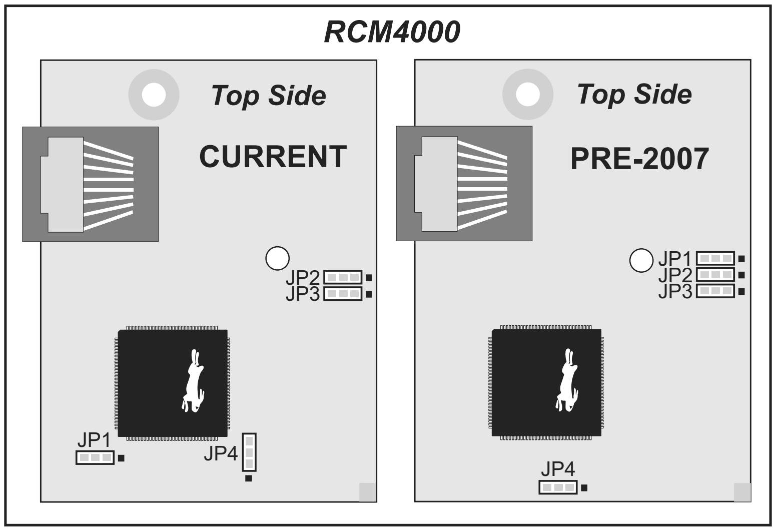


Figure A-6. Location of RCM4000 Configurable Positions

Table A-9 lists the configuration options.

Table A-9. RCM4000 Jumper Configurations

Header	Description	Pins Connected		Factory Default
JP1	PE6 or SMODE1 Output on J3[*]	1–2	SMODE1	×
		2–3	PE6	
JP2	PE5 or SMODE0 Output on J3	1–2	SMODE0	×
		2–3	PE5	
JP3	PE7 or STATUS Output on J3	1–2	STATUS	×
		2–3	PE7	
JP4	Battery Backup for Real-Time Clock	1–2	Battery Backup	×
		2–3	No Battery Backup	

* PE5–PE7 are used for the Ethernet clock and I/O signals, which ordinarily would not be routed to a general-purpose I/O header to minimize noise. Therefore, the RCM4000 RabbitCore modules present the SMODE and STATUS lines to header J3.

NOTE: The jumper connections are made using 0 Ω surface-mounted resistors.

B.1 Introduction

The Prototyping Board included in the Development Kit makes it easy to connect an RCM4000 module to a power supply and a PC workstation for development. It also provides some basic I/O peripherals (RS-232, LEDs, and switches), as well as a prototyping area for more advanced hardware development.

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

As you progress to more sophisticated experimentation and hardware development, modifications and additions can be made to the board without modifying the RCM4000 module.

The Prototyping Board is shown below in Figure B-1, with its main features identified.

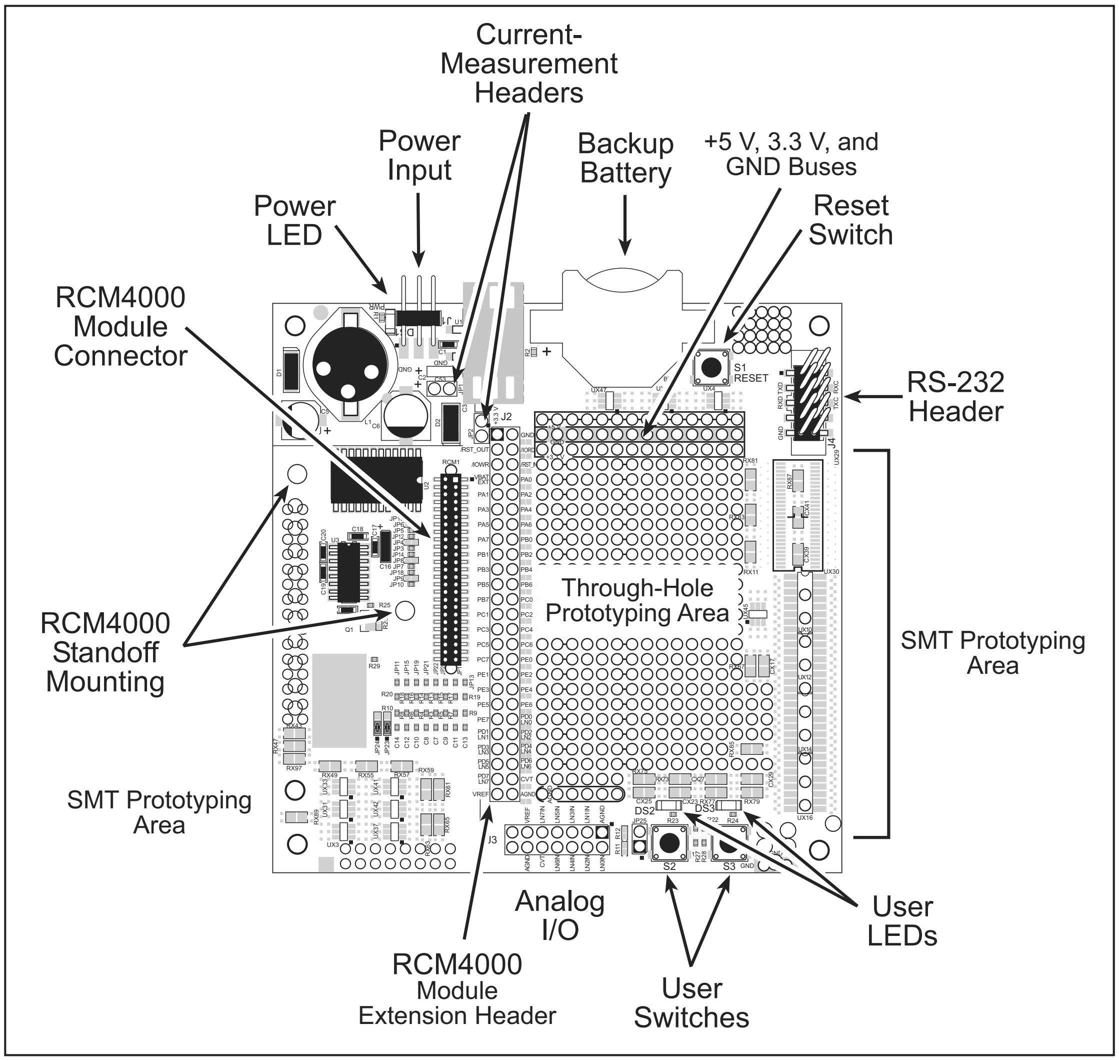


Figure B-1. Prototyping Board

INDEX