APPLICATION NOTE

AN415

RIO Programmable I/O Kit

Introduction

The Rabbit RIO is a peripheral chip designed to be used with any microprocessor-based system to provide up to 32 additional digital I/O lines and a broader range of other features. The Rabbit RIO communicates with the microprocessor in either a serial mode via the SPI and RabbitNet protocols or a parallel mode. The multiple communication modes allow the Rabbit RIO to be a part of a wide variety of systems that use any one of these communication methods.

The RIO Programmable I/O Kit is intended to demonstrate how to expand an embedded control system design by adding additional I/O and other features. The Kit provides a Prototyping Board with a Rabbit RIO chip already installed and configurable header locations to allow you to develop your own application using the Dynamic C function calls and sample programs included with the Kit.

The sample programs included with the RIO Programmable I/O Kit can serve as a template for your application.

Application Kit Features

- RCM4110 RabbitCore module with Rabbit RIO Prototyping Board
- AC adapter, programming cable, and assorted parts for use with the Rabbit RIO Prototyping Board
- Complete Dynamic C software CD and supplemental CD with sample programs and reference information related to the RIO Programmable I/O Kit

What Else You Will Need

Dynamic C must be installed on your PC. Insert the Dynamic C CD from the RIO Programmable I/O Kit in your PC's CD-ROM drive. If the installation does not auto-start, run the **`setup.exe`** program in the root directory of the Dynamic C CD. Install the software from the supplemental CD and any Dynamic C software modules after you install Dynamic C.

Besides what is supplied with the RIO Programmable I/O Kit, you will need a PC with an available COM or USB port to program the RCM4110 in the RIO Programmable I/O Kit. If your PC only has a USB port, you will also need an RS-232/USB converter (Rabbit Semiconductor Part No. 540-0070), which is not included with this kit. Note that not all RS-232/USB converters work with Dynamic C.

The ***Rabbit RIO User's Manual***, which can be accessed from the documentation page icon that was placed on your desktop during the installation of the supplemental CD, provides complete information on using the Rabbit RIO.

Hardware Setup

The ***RIO Programmable I/O Kit Getting Started*** instructions included with the RIO Programmable I/O Kit shows how to set up and program the RCM4110 with the Prototyping Board.

The following steps from the ***RIO Programmable I/O Kit Getting Started*** instructions summarize the setup process once Dynamic C and the software from the supplemental CD have been installed on your PC.

1. Prepare the Prototyping Board for Development

Snap in four of the plastic standoffs supplied in the bag of accessory parts from the RIO Programmable I/O Kit in the holes at the corners as shown.

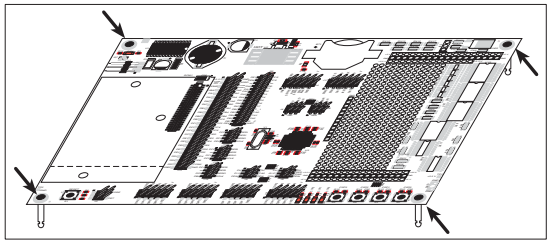


Figure 1. Insert Plastic Standoffs

2. Attach Module to Prototyping Board

Turn the RCM4110 module so that the mounting holes of the RCM4110 line up with the corresponding holes on the Prototyping Board. Insert the metal standoffs as shown in Figure 2, secure them from the bottom of the Prototyping Board using the 4-40 × 3/16 screws and washers, then insert the module's header J3 on the bottom side into header socket RCM1 on the Prototyping Board.

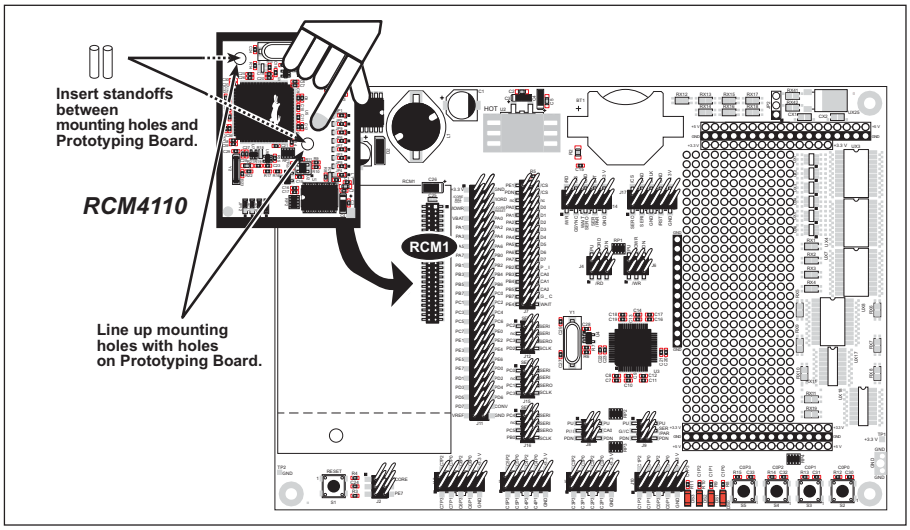


Figure 2. Install the RCM4110 Module on the Prototyping Board

NOTE: It is important that you line up the pins on header J3 of the RCM4110 module exactly with socket RCM1 on the Prototyping Board. The header pins may become bent or damaged if the pin alignment is offset, and the module will not work. Permanent electrical damage to the module may also result if a misaligned module is powered up.

Press the module's pins gently into the Prototyping Board header socket—press down in the area above the header pins. For additional integrity, you may secure the RCM4110 to the standoffs from the top using the remaining two 4-40 × 3/16 screws.

3. Set Jumpers

Jumpers were placed at the factory on Prototyping Board headers J2, J7, J8, J9, and J15 as shown in Figure 3. These jumper settings are the default settings for use with the RCM4110 RabbitCore module in the SPI serial mode using Serial Port D, which is the default serial port used by the sample programs. Table 1 describes the jumper settings for other communication modes and serial ports.

4. Connect Programming Cable

The programming cable connects the RCM4110 to the PC running Dynamic C to download programs and to monitor the RCM4110 module during debugging.

Connect the 10-pin connector of the programming cable labeled **PROG** to header J1 on the RCM4110 as shown in Figure 3. Be sure to orient the marked (usually red) edge of the cable towards pin 1 of the connector. (Do not use the **DIAG** connector, which is used for a normal serial connection to Serial Port A.)

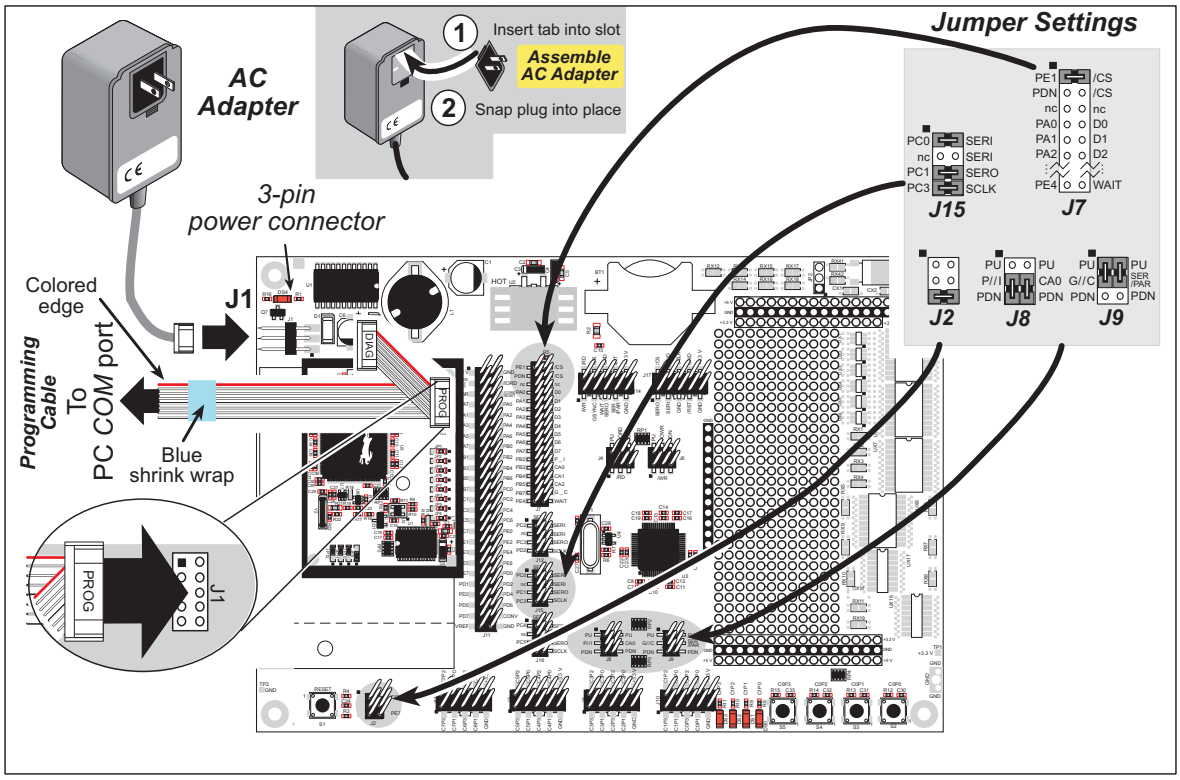


Figure 3. Prototyping Board and Programming Cable Setup

NOTE: Be sure to use the programming cable (part number 101-0542) supplied with the RIO Programmable I/O Kit—the programming cable has blue shrink wrap around the RS-232 converter section located in the middle of the cable. Programming cables with red or clear shrink wrap from other Rabbit Semiconductor kits are not designed to work with RCM4110 modules.

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

5. Connect Power

Once all the 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 appropriate plug. The RIO Programmable I/O 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 3, then press down on the plug until it clicks into place.

Connect the AC adapter to 3-pin header J1 on the Prototyping Board as shown in Figure 3 above. The connector may be attached either way as long as it is not offset to one side—the center pin of J1 is always connected to the positive terminal, and either edge pin is ground.

Plug in the AC adapter. The power LED on the Prototyping Board above the power connector at J1 should light up. The RCM4110 and the Prototyping Board are now ready to be used.

NOTE: If you need to reset the RCM4110, disconnect, then reconnect the power supply.

Using the Prototyping Board

This section describes the features and accessories of the Prototyping Board, and explains the use of the Prototyping Board to demonstrate the Rabbit RIO and to build prototypes of your own circuits. The Prototyping Board has power-supply connections and also provides some basic I/O peripherals (LEDs and switches), as well as a prototyping area for more advanced hardware development.

The Prototyping Board makes it easy to connect a Rabbit RIO to a RabbitCore module based on the Rabbit 4000 microprocessor and to a power supply and a PC workstation for development. It also provides some basic I/O peripherals (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 using the jumper configurations set at the factory.

The Prototyping Board is shown below in Figure 4, with its main features identified.

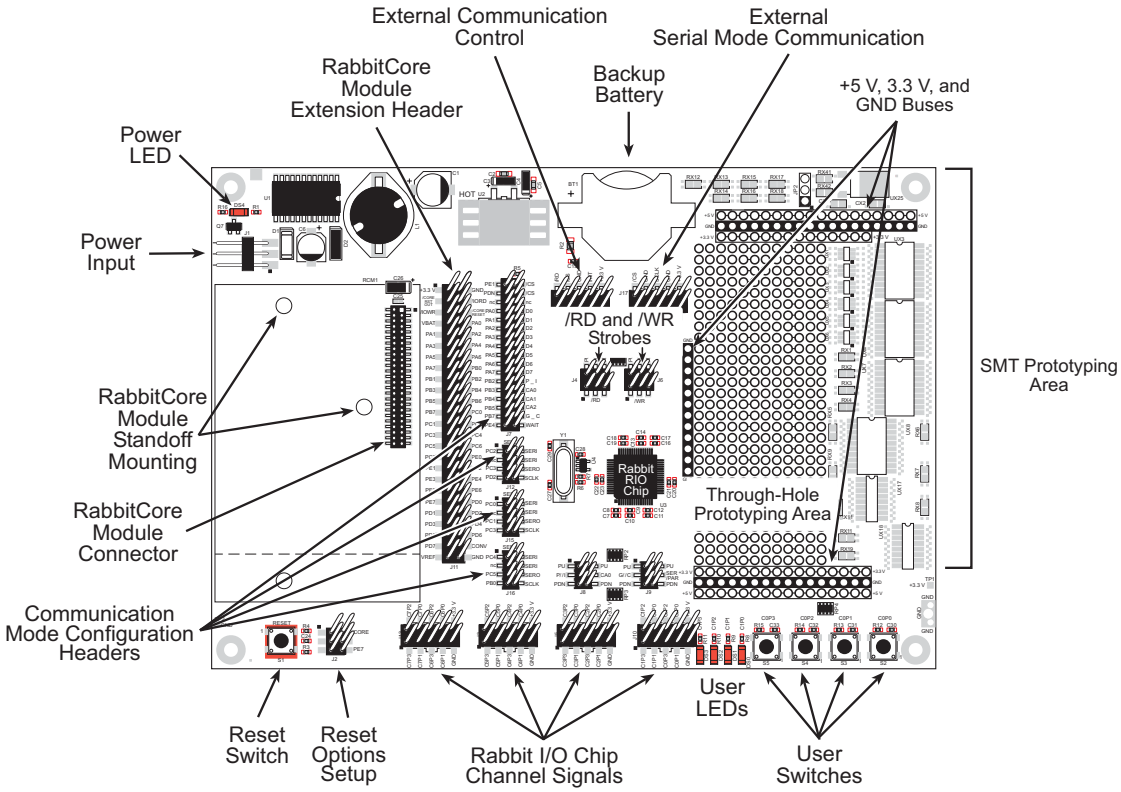


Figure 4. Prototyping Board

Prototyping Board Features

- **Power Connection**—A 3-pin header is provided for connection to the power supply. Note that the 3-pin header is symmetrical, with both outer pins connected to ground and the center pin connected to the raw V+ input. The cable of the AC adapter provided with the North American version of the RIO Programmable I/O Kit is terminated with a header plug that connects to the 3-pin header in either orientation. The header plug leading to bare leads provided for overseas customers can be connected to the 3-pin header in either orientation.

  Users providing their own power supply should ensure that it delivers 8–30 V DC at 8 W. The voltage regulators will get warm while in use.

- **Regulated Power Supply**—The raw DC voltage provided at the 3-pin header is routed to a 5 V switching voltage regulator, then to a separate 3.3 V linear regulator. The regulators provide stable power to the RabbitCore module, the Rabbit RIO, and the Prototyping Board.
- **Reset Switch**—A momentary-contact, normally open switch is connected directly to the Rabbit RIO chip's **/RESET** pin. Pressing the switch forces a hardware reset of the Rabbit RIO chip. This switch may also be configured to reset the RabbitCore module on the Prototyping Board by jumpering pins 1–2 on header J2 instead of pins 5–6.
- **I/O Switches and LEDs**—Four momentary-contact, normally open switches are connected to the four Channel 0 pins of the Rabbit RIO and may be read as inputs by sample applications.

  Four LEDs are connected to the four Channel 1 pins of the Rabbit RIO, and may be driven as output indicators by sample applications.
- **Prototyping Area**—A generous prototyping area has been provided for the installation of through-hole components. +3.3 V, +5 V, and Ground buses run around the edge of this area. Several areas for surface-mount devices are also available. (Note that there are SMT device pads on both top and bottom of the Prototyping Board.) Each SMT pad is connected to a hole designed to accept a 30 AWG solid wire.
- **Module Extension Header**—The complete non-analog pin set of the RabbitCore module is duplicated at header J11. See Figure 6 for the header pinouts.

  NOTE: The same Prototyping Board may be used for several series of RabbitCore modules, and so the signals at J11 depend on the signals available on the specific RabbitCore module.
- **Rabbit RIO Channel Signals**—Four headers with 10-pin 0.1" pitch header strip installed at J3, J5, J10, and J13 each carry the signals from two channels of the Rabbit RIO. These headers allow you to connect a ribbon cable that leads to a standard DB9 serial connector.
- **External Communication Control**—The 10-pin 0.1" pitch header strip installed at J14 allows external communication controls. This header can be used to provide common command signals to the Rabbit RIO if a master other than a Rabbit 4000 based RabbitCore module is used since such a master cannot be connected to RCM1 on the Prototyping Board.
- **External Serial Mode Communication**—Header J17 brings the signals needed for external serial mode communication. Header J17 is also a 10-pin 0.1" pitch header strip.
- **Backup Battery**—A 2032 lithium-ion battery rated at 3.0 V, 220 mA·h, provides battery backup for the RabbitCore module's SRAM and real-time clock.

Power Supply

The Prototyping Board has an onboard +5 V switching power regulator from which a +3.3 V linear regulator draws its supply. Thus both +5 V and +3.3 V are available on the Prototyping Board. The Prototyping Board is protected against reverse polarity by a Shottky diode at D1 as shown in Figure 5.

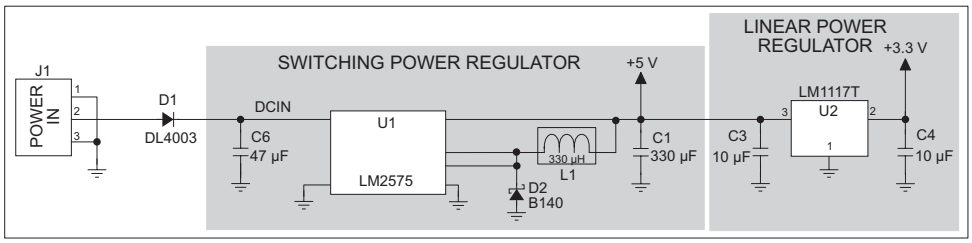


Figure 5. Prototyping Board Power Supply

Prototyping Board Description

The Prototyping Board acts as both a demonstration board and a prototyping board. As a demonstration board, it can be used to demonstrate the functionality of the Rabbit RIO right out of the box with simple jumper settings to select the communication mode.

The Prototyping Board comes with the basic components necessary to demonstrate the operation of the Rabbit RIO. Four LEDs (DS1–DS4) are connected to the Channel 1 pins, and four switches (S2–S5) are connected to the Channel 0 Pins to demonstrate the interface to the Rabbit RIO. Reset switch S1 is the hardware reset for the microprocessor and the Rabbit RIO.

The Prototyping Board provides the user with RabbitCore module connection points brought out conveniently to labeled points at header J11 on the Prototyping Board. The Rabbit RIO channel signals are available on headers J3, J5, J10, and J13. A ribbon cable may be connected to these headers. Figure 6 shows the pinouts for these locations.

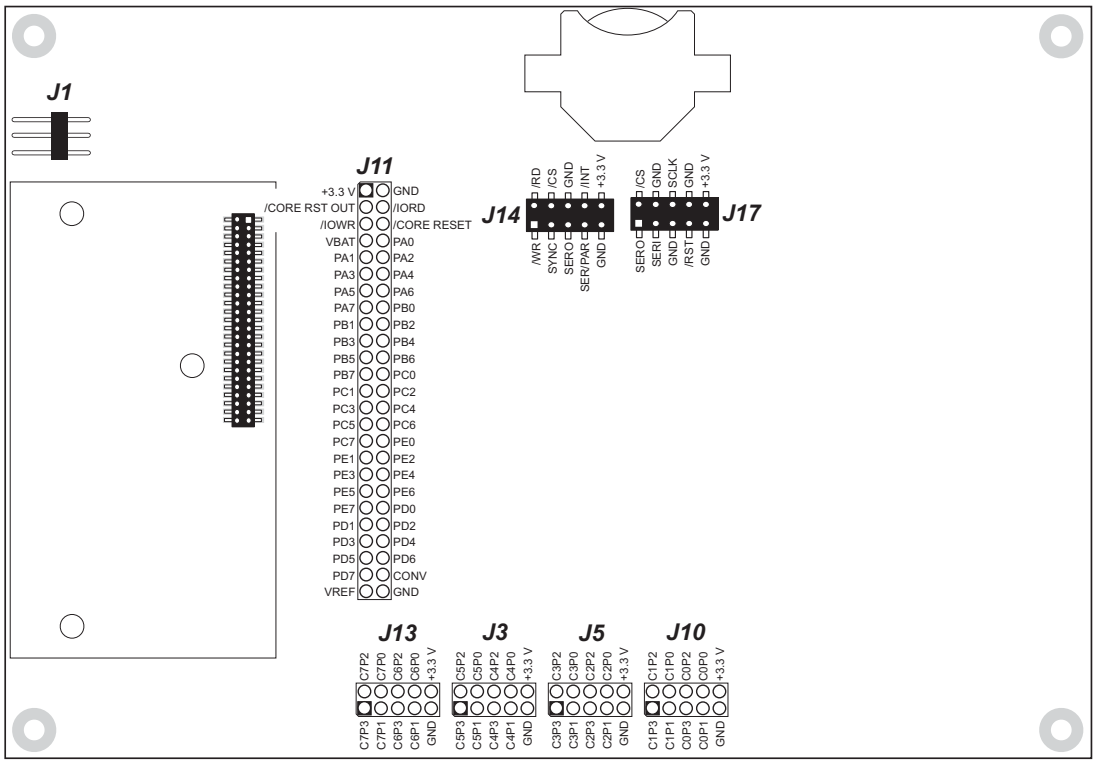


Figure 6. Prototyping Board Pinout

There is a 2.2" × 2.7" through-hole prototyping space available on the Prototyping Board. The holes in the prototyping area are spaced at 0.1" (2.5 mm). +3.3 V, +5 V, and GND traces run along the top and bottom edges of the prototyping area for easy access. Small to medium circuits can be prototyped using point-to-point wiring with 20 to 30 AWG wire between the prototyping area, the +3.3 V, +5 V, and GND traces, and the surrounding area where surface-mount components may be installed. Small holes are provided around the surface-mounted components that may be installed around the prototyping area.

Adding Other Components

There are pads for 28-pin TSSOP devices, 16-pin SOIC devices, and 20-pin SOT devices that can be used for surface-mount prototyping with these devices. There are also pads that can be used for SMT resistors and capacitors in an 0805 SMT package. Each component has every one of its pin pads connected to a hole in which a 30 AWG wire can be soldered (standard wire wrap wire can be soldered in for point-to-point wiring on the Prototyping Board). Because the traces are very thin, carefully determine which set of holes is connected to which surface-mount pad.

Header location JP2 is used to connect an additional linear voltage regulator at UX25 for your prototyping needs. Neither the optional voltage regulator nor header JP2 is stuffed. See the documentation supplied by the manufacturer of the voltage regulator for additional information on determining the resistor and capacitor values to be used with the circuit.

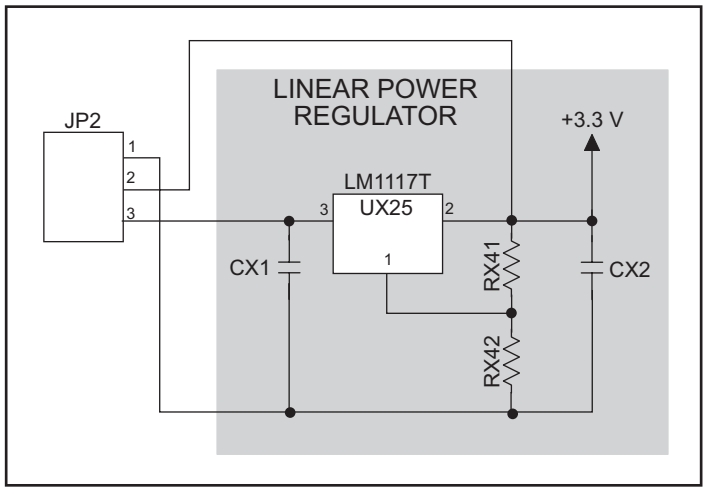


Figure 7. Optional +3.3 V Voltage Regulator Circuit

Prototyping Board Jumper Configurations

Figure 8 shows the header locations used to configure the various Prototyping Board options via jumpers.

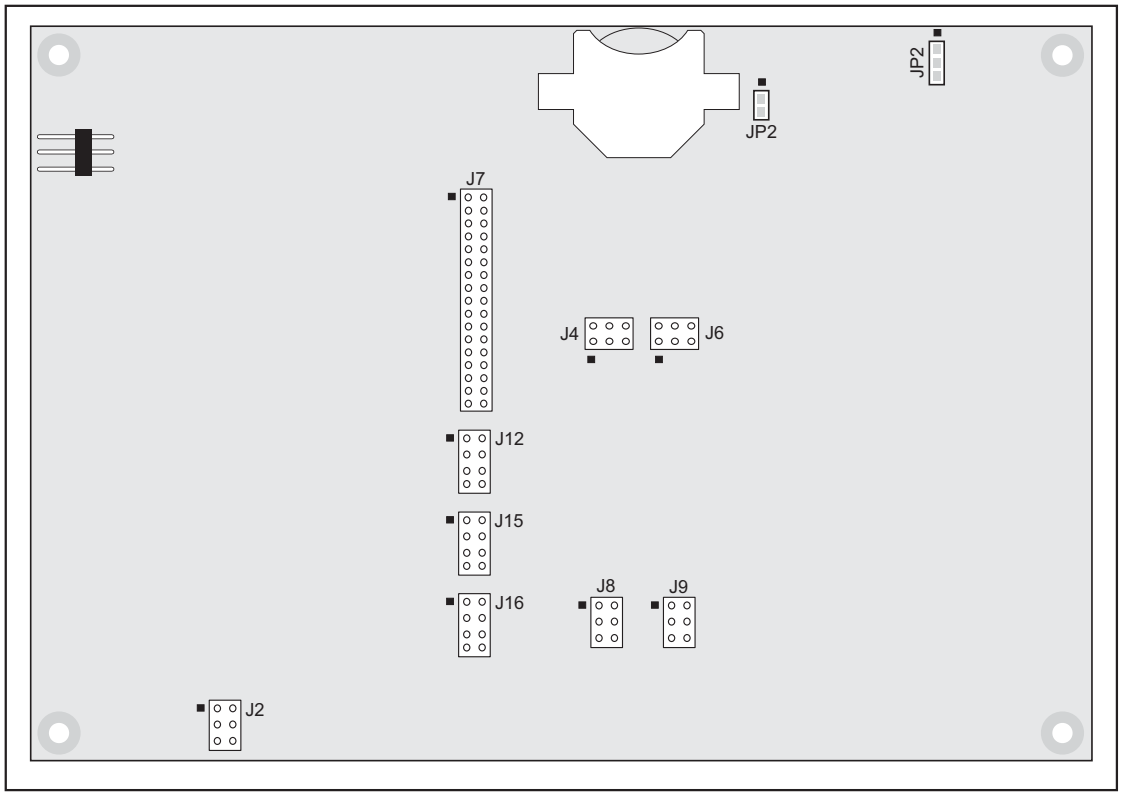


Figure 8. Configurable Jumper Locations on Prototyping Board

Table 1 lists the configuration options for the RCM4110 RabbitCore module using the slip-on jumpers that are already installed on the Prototyping Board. Be sure to place jumpers only at the configuration locations specified for the mode you are configuring.

The settings on header J15 for the serial communication modes reflect using Serial Port D with the LSB first. If you use Serial Port B or Serial Port C for serial communication, then jumper the same pins on header J16 or header J12 that you would have jumpered on header J15 as shown in Table 1.

Table 1. Rabbit RIO Prototyping Board Jumper Configurations

Configuration	Option	Header	Pins Connected
Chip Select	(applies all to all communication modes)	J7	1–2 (Rabbit RIO /CS on PE1*)
			3–4 (Rabbit RIO /CS pulled low)
Communication Mode	Parallel	J7	7–8 9–10 11–12 13–14 15–16 17–18 19–20 21–22 23–24 25–26 27–28 29–30 31–32 33–34
		J9	4–6
	Read Enable	J4	3–4
	Write Enable	J6	3–4
	RabbitNet Device	J8	3–5
		J9	2–4 3–5
		J15	1–2 5–6 7–8
	RabbitNet Hub	J8	1–3
		J9	2–4 3–5
		J15	1–2 5–6 7–8

Table 1. Rabbit RIO Prototyping Board Jumper Configurations (continued)

Configuration	Option	Header	Pins Connected
Communication Mode (continued)	SPI (LSB first)†	J8	3–5 4–6
		J9	1–3 2–4
		J15	1–2 5–6 7–8
	SPI (MSB first)†	J8	2–4 3–5
		J9	1–3 2–4
		J15	1–2 5–6 7–8
	Bidirectional Serial Data Flow on One Wire (LSB first)	J8	1–3 4–6
		J9	1–3 2–4
		J15	1–2 7–8
	Bidirectional Serial Data Flow on One Wire (MSB first)	J8	1–3 2–4
		J9	1–3 2–4
		J15	1–2 7–8
Microprocessor Reset of Rabbit RIO	via /CORE_RESET (all modes except SPI)	J2	1–2
	via PE7‡ (SPI modes)		5–6

* The RCM4000 and RCM4020 require a chip select other than the default PE1. The only /CS available on these RabbitCore modules and in the library is PE0. Instead of the slip-on jumper across pins 1–2 on header J3, use a wire jumper to connect PE0 (pin 32 on header J11) to /CS (pin 4 on header J14), and use **`RIO_CS_PE0`** as part of the **`iobase`** parameter for **`rio_init()`**.

† SPI is the unidirectional serial data flow on two wires.

‡ Use this reset setting via PE7 for the RCM4100, RCM4110, RCM4120, RCM4300, RCM4310, RCM4400W, and RCM4510W RabbitCore modules. The hardware reset signal for other RabbitCore modules will need to come out on a different pin. Remove the slip-on jumper from header J2 for the RCM4000, RCM4010, RCM4020, RCM4200, and RCM4210 RabbitCore modules, and connect a wire jumper from PB7 (pin 23 on header J11) to /RST (pin 7 on header J17).

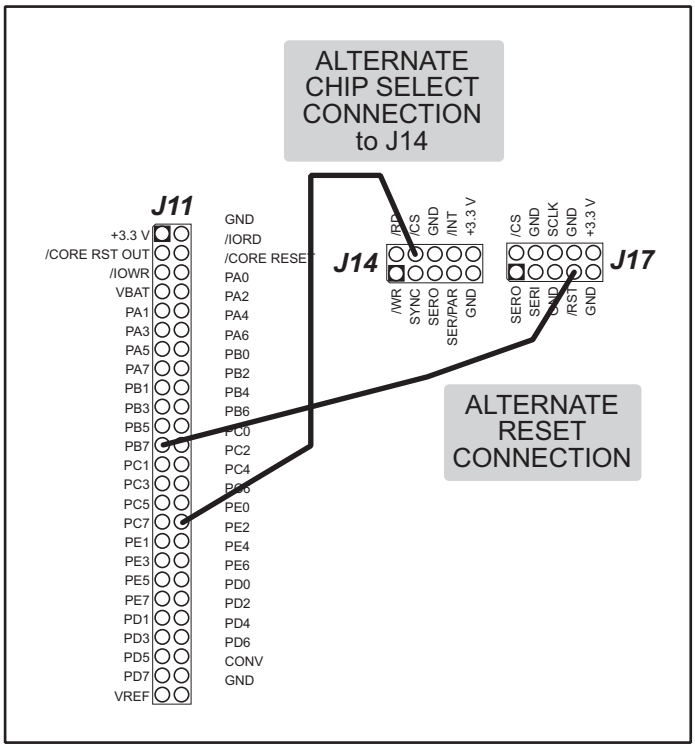


Figure 9. Alternate Chip Select and Reset Connections

Sample Programs

To help familiarize you with the Rabbit RIO, several sample programs were developed around the Prototyping Board and the RCM4110 RabbitCore module included with the RIO Programmable I/O Kit. Loading, executing and studying these programs will give you a solid hands-on overview of the Rabbit RIO's capabilities, as well as a quick start with Dynamic C as an application development tool.

> **NOTE:** The sample programs assume 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.

In order to run a sample program,

1. Your module must be plugged into the Prototyping Board as described in the Getting Started instructions or the "Hardware Setup" section in this application note.
2. Dynamic C must be installed and running on your PC.
3. The programming cable must connect the programming header on the module to your PC.
4. Power must be applied to the module through the Prototyping Board.

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

Each sample program has comments that describe the purpose and function of the program. Follow the instructions at the beginning of the sample program.

The sample programs for the Rabbit RIO are in the Dynamic C **SAMPLES\RIO** folder. The sample programs may be run without any further modifications when using the RCM4110 RabbitCore module included with the RIO Programmable I/O Kit once the Prototyping Board is jumpered as described in the preceding "Header location JP2 is used to connect an additional linear voltage regulator at UX25 for your prototyping needs. Neither the optional voltage regulator nor header JP2 is stuffed. See the documentation supplied by the manufacturer of the voltage regulator for additional information on determining the resistor and capacitor values to be used with the circuit." section.

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

Configuration Macros

The default communication mode the sample programs operate in is the SPI mode. This is established in the sample program with the following macro.

```
#define RIO_IFACE RIO_INTERFACE_SPI
```

Serial Port D is the default serial port for the SPI communication mode in the sample programs. This is established with the following macro.

```
#define RIO_SPI_PORT SDDR
```

Change **SDDR** to **SBDR** or **SCDR** if you will be using Serial Port B or Serial Port C respectively.

Remember to jumper the pins on header J16, header J12, or header J15 as shown in Table 1 depending on whether you are using Serial Port B, Serial Port C, or Serial Port D respectively.

The sample programs have one additional configuration macro, which is commented out for the default LSB first.

```
//#define MSB_FIRST
```

Uncomment the macro and set the jumpers as shown in Table 1 on headers J8, J9, and J16/J12/J15 for MSB first.

The hardware reset in the SPI mode for the RCM4100, RCM4110, RCM4120, RCM4300, RCM4310, RCM4400W, and RCM4510W RabbitCore modules requires the following macros, which are included in the sample programs before the **#use rio.lib** statement.

```
#define RIO_ENABLE_HW_RESET
#define RIO_HWRST_PORT 'E' // Parallel Port E is the default
#define RIO_HWRST_PIN 7    // Pin 7 provides the default reset bit
```

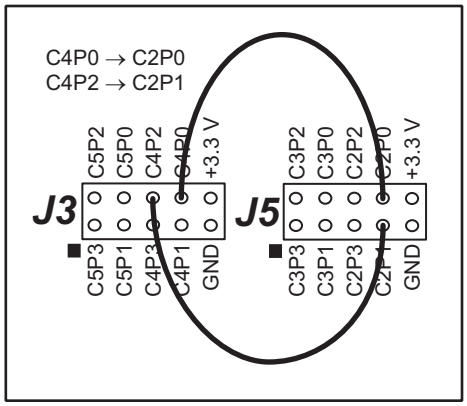
The hardware reset signal will need to come out on a different pin when you are using other RabbitCore modules. Use PB7 for a hardware reset in the SPI mode with the RCM4000, RCM4010, RCM4020, RCM4200, and RCM4210 RabbitCore modules. See Table 1 for the physical jumper settings on the Prototyping Board and change the second macro to **#define RIO_HWRST_PORT 'B'**.

Sample Programs for Rabbit-Based Boards

- **EVENT_COUNTER.C**—Demonstrates the event counter functionality of the Rabbit RIO chip.
  - C0P0 is the countdown event detector. Pressing switch S2 will make the counter count down.
  - C0P1 is the countup event detector. Pressing switch S3 will make the counter count up.

  Register values for Channel 0 are displayed in the Dynamic C **STDIO** window.
  - CVLR and CVMR (Count Value LSB/MSB Register) hold the current count.
  - CBLR and CBMR (Count Begin LSB/MSB Register) hold the value of CVLR on the rising edge of the increment switch.
  - CELR and CEMR (Count End LSB/MSB Register) hold the value of CVLR on the rising edge of the decrement switch.
- **INPUT_CAPTURE.C**—Demonstrates the input capture functionality of the Rabbit RIO chip.

  C2P0 and C2P1 are input capture pins that will measure the time between the rising edge of C4P2 and C4P0. C4P2 and C4P0 are PWM signals with different duty cycles — C4P0 is a variable-phase PWM signal (PPM) whose rising and falling edges can be changed by inputs from switches S2, S3, and S4.

  Before you compile and run this sample program, connect C4P0 to C2P0 and connect C4P2 to C2P1 on the Prototyping Board. C0P0, C0P1, and C0P2 are already connected to switches S2, S3, and S4 on the Prototyping Board.

  The time between the PPM edges will be:

  200 µs = no switch pressed
  400 µs = switch S2 pressed
  600 µs = switch S3 pressed
  800 µs = switch S4 pressed

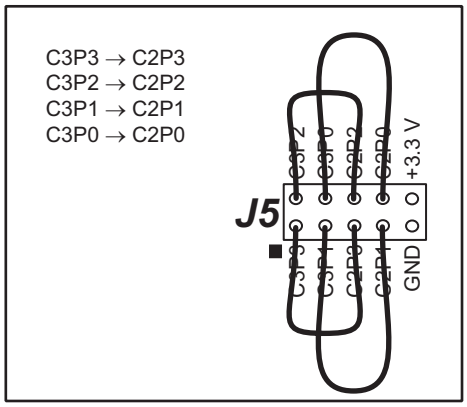
- **PIN_PAIR.C**—Demonstrates the pin-pair protection functionality of the Rabbit RIO chip.

  Channel 2 will output to its protected pins. The "unsafe" state is the complement of what the state of those pins is when the protection register is set. Channel 3 will read from Channel 2. The input should equal the output, unless Channel 2 is outputting the "unsafe" state. In this case, the output will be the "safe" state instead.

  For example, if **SAFE_STATE** is set to 01b, then the unsafe state is 10b. Even when the Rabbit microprocessor tells the Rabbit RIO to write 10b out, the Rabbit RIO will not. The original "safe" state, 01b, is read back into Channel 1. Since only 2 bits are used to represent the "safe" state, the value can be from 0 to 3 (00b to 11b).

  Before you compile and run this sample program, connect the Channel 2 pins to the corresponding pins on Channel 3 (header J5 pins 1–5, 2–6, 3–7, and 4–8). Change the **#define** statements to change which pins to protect:

  - **UPPER** macro protects the upper 2 bits of Channel 2
  - **LOWER** macro protects the lower 2 bits of Channel 2
  - **SAFE_STATE** macro specifies what pin combination to avoid

  The sample program uses the following macro setup.

```
#define SAFE_STATE 1
#define UPPER 1
#define LOWER 0
#define PROTECT_BITS UPPER
```

  Since **SAFE_STATE** is set to 1 (= 01b), the unsafe state is 2 (= 10b). The **#define PROTECT_BITS** macro is set to **UPPER**, which protects the upper 2 bits of Channel 2.

  When you compile and run this sample program, the Dynamic C **STDIO** window will open, and will scroll through the various outputs of Channels 2 and 3 to demonstrate what happens when an unsafe value is output.

- **PWM_MEASURE.C**—Demonstrates the PWM measurement functionality of the Rabbit RIO chip.

  C2P0 reads the pulse-width modulator and measures the length of time (in ms) the signal is high. C4P0 is a variable-phase PWM signal (PPM) whose rising and falling edges can be changed by inputs from switches S2, S3, and S4. The time between the PPM edges will be:

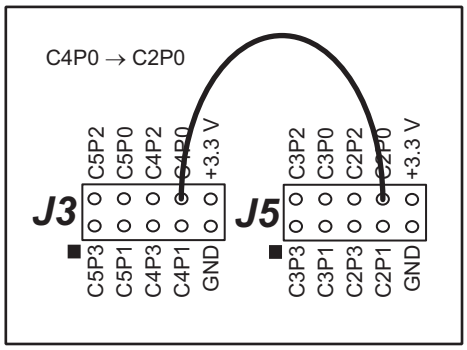
  - 200 µs when no switches are pressed
  - 400 µs when switch S2 is pressed
  - 600 µs when switch S3 is pressed
  - 800 µs when switch S4 is pressed

  C4P1 pulse width is 400 µs

  C4P2 pulse width is 600 µs

  C4P3 pulse width is 800 µs

  Before you compile and run this sample program, connect C2P0 to C4P0. Switches S2, S3, and S4 are already attached to C0P0, C0P1, and C0P2.

  The Dynamic C **STDIO** window will display the time the pulse is high as "count." It will also display the value of the input from the switches as "input."

- **QDECODE.C**—Demonstrates the quadrature decoder functionality of the Rabbit RIO chip.

  C0P0 (S2) is the in-phase signal, and C0P2 (S4) is the quadrature signal. The count register will increment and decrement depending on the order in which the switches go high.

  00 → 01 → 11 → 10 → 00 = increment

  00 → 10 → 11 → 01 -→ 00 = decrement

  This sample program may be run in any one of the communication modes as described in the instructions at the beginning of the sample program.

- **ROTATELED.C**—Demonstrates the Channel 1 outputs to the four LEDs on the Prototyping Board.

  The four LEDs on the Prototyping Board will flash on and off in a rotating sequence, and the Dynamic C **STDIO** window will open on your PC to display the status of the Rabbit RIO's Channel 1.

- **SIMPLEIO.C**—Demonstrates the parallel I/O functionality of the Rabbit RIO chip.

  Before you compile and run this sample program, connect the pins of Channel 4 to the corresponding pins of Channel 5 on header J3.

  Channel 4 outputs a nibble to Channel 5. This sample program will write directly to the data output register of Channel 4, all in one nibble. The Channel 5 input should show a counter that counts from 0 to 15.

- **SIMPLEIO2.C**—Demonstrates the parallel I/O functionality of the Rabbit RIO chip.

  Before you compile and run this sample program, connect the pins of Channel 4 to the corresponding pins of Channel 5 on header J3.

  Channel 4 outputs a nibble to Channel 5. Instead of writing directly to the Channel 4 pins' data output registers, this sample program will write to each pin's output register, bit by bit. The Channel 5 input should show a shifting bit pattern with the sequence 1, 2, 4, 8.

- **SYNC.C**—Demonstrates the forced increment, decrement, and synch functionality of the Rabbit RIO chip.

  This sample program will count up and down using the forced increment and decrement register bits. The only human input is the synch signal to reset the counter. This synch signal will come from the four switches (S2–S5) attached to Channel 0.

  The synch signal will go into C0P0, C0P1, C0P2, C0P3, or GSYNC. For example, if C0P0 is selected as the synch signal, toggling switch S2 will cause a counter reset to the channel. Use the following line in the sample program to select the synch signal.

  ```
  #define SYNC_SELECT P0_SYNC
  ```

  The table below lists the macros for selecting the switches for the synch signal.

G_SYNC	touch GSYNC (pin 3, header J14) to GND
P0_SYNC	C0P0 (S2)
P1_SYNC	C0P1 (S3)
P2_SYNC	C0P2 (S4)
P3_SYNC	C0P3 (S5)

Generic Sample Programs for All Boards

Since the Rabbit RIO can be used with other 8-bit microprocessors, these two sample programs illustrate how to program the Rabbit RIO without Dynamic C for other microprocessors.

- **RIOBASICLED.C**—Demonstrates how to toggle a single Rabbit RIO output pin. The sample program uses basic I/O functions to set up and control the output pin. There is a basic SPI handler in this sample program.
- **RIOBASICPWM.C**—Demonstrates how to generate PWM signals on a single RIO channel. The sample program uses basic I/O functions to set up and control the PWM channel. There is a basic SPI handler in this sample program.

Appendix A — Software Reference

The Dynamic C **LIB\RIO\RIO.LIB** library provides the function calls for controlling the Rabbit RIO. One or more Rabbit RIO chips may be used with a Rabbit 3000 or a Rabbit 4000 microprocessor. This library supports these three communication modes.

- Parallel (via Parallel Ports A and B in the auxiliary I/O bus mode)
- SPI (clocked serial interface over Serial Ports B, C, or D)
- RabbitNet (clocked serial bus architecture)

Table A-1 summarizes the serial port connections defined in the Dynamic C **RIO.LIB** library:

Table A-1. Default Serial Port Connections for* RIO.LIB *Library

Serial Port	SCLK	SERI	SERO
B	PB0	PC4	PC5
C	PD2	PC2	PC3
D	PC3	PC0	PC1

The Dynamic C **LIB\RN_CFG_GENERIC.LIB** library provides the configuration function calls for the RabbitNet mode.

Configuration Macros

The following configuration macros are available, and those you choose to use must be defined before the **#use rio.lib** statement.

Select a macro to identify which of the three communication modes you will be using for the Rabbit RIO interface.

```
#define RIO_IFACE RIO_INTERFACE_PARALLEL
#define RIO_IFACE RIO_INTERFACE_SPI
#define RIO_IFACE RIO_INTERFACE_RNET
```

This next macro allows you to use hardware interrupts.

```
#define RIO_ENABLE_INTERRUPTS <which>
```

When you use hardware interrupts, you will need to connect the Rabbit RIO's interrupt request (hardware signal) to PE0, PE1, PE4, or PE5. If the macro is not defined (the default), Rabbit RIO interrupts can only be serviced by software polling using the **rio_tick()** function call.

<which> is set to

0x01 — enable external interrupt 0 (PE0 and/or PE4 pins)
0x02 — enable external interrupt 1 (PE1 and/or PE5 pins)
0x03 — both of the above

Note that you may call **rio_set_handler()** for all Rabbit RIO channels that are to have a channel event handler. The handlers may also be called from **rio_tick()** if you are not using interrupts.

Interrupts are ***not*** supported when using Rabbit RIO chips in the RabbitNet mode.

The following macro allows you to select a slower SPI clock rate. The default is 8.

```
#define RIO_SPI_DIV <divider>
```

If the macro is defined, the SPI clock will be 1/<divider+1> of the CPU clock frequency (e.g.,. **#define RIO_SPI_DIV 15** will give 1/16 of the maximum rate).

The following hardware reset macro is useful for SPI communication modes to use a hardware (wired) reset instead of a software (communication channel) reset when **rio_reset()** is called. This macro is especially useful for resetting the SPI clock.

```
#define RIO_ENABLE_HW_RESET
```

The Rabbit hardware reset pin is PE7 by default, but can be changed by defining the following macros.

```
#define RIO_HWRST_PORT 'E' // Parallel Port E is the default
#define RIO_HWRST_PIN 7 // Pin 7 provides the default reset bit
```

The library provides some macros that are useful during code development and debugging.

```
#define RIO_DEBUG
```

This macro enables single-stepping within C code in the Dynamic C **LIB\RIO\RIO.LIB** library.

```
#define RIO_ASMDEBUG
```

This macro enables single-stepping within assembler code in the Dynamic C **LIB\RIO\RIO.LIB** library.

```
#define RIO_VERBOSE
```

This macro enables verbose error and debug messages to be displayed in the Dynamic C **STDIO** window. Use this macro only while developing your application.

rio_init

```
int rio_init(RIO *r, word flags, word iobase);
```

DESCRIPTION

Initializes a context structure for the Rabbit RIO. This structure is passed to the other Rabbit RIO function calls.

This function call allows the application to specify the communication mode (parallel, SPI, or RabbitNet) for the Rabbit RIO using a configuration macro.

PARAMETERS

r — pointer to device context to be initialized. The initial contents are overwritten. The application should not modify this structure directly once it is initialized. If you have defined **RIO_ENABLE_INTERRUPTS**, and you have specified one of the **RIO_INTR_PE<n>** flags, then this parameter ***must*** point to static storage.

flags — initialization flags as follows:

RIO_F_SPI — SPI (serial) mode access, otherwise parallel
RIO_F_BIDIR — **F_SPI** indicates bidirectional data mode (Rabbit side will need to tie Tx and Rx together)
RIO_F_RNET — use RabbitNet interface
RIO_F_IOHS — if parallel, use I/O handshake (Rabbit 4000 only)

If the flags are zero, then an ordinary parallel data bus is assumed. In this case, you must **#define PORTA_AUX_IO** before **#use rio.lib**.

If you have **#define RIO_ENABLE_INTERRUPTS**, you can also OR in one of the following flags:

RIO_INTR_PE0
RIO_INTR_PE1
RIO_INTR_PE4
RIO_INTR_PE5

to specify which hardware external interrupt pin is to be used. This automatically sets up an interrupt service routine to invoke any handlers defined via **rio_set_handler()** when a channel event occurs. Of course, this depends on a link between the Rabbit RIO's /INT pin and the appropriate Parallel Port E pin on the Rabbit microprocessor.

NOTE: The application must call **rio_interrupt_enable()** to actually enable interrupts from the Rabbit RIO.

iobase I/O base address of the device; the interpretation of this parameter depends on the mode selected

- in the parallel mode **iobase** should be a multiple of 0x2000, which defines the external I/O base address to use. In this case, the appropriate Parallel Port E pin will be used automatically as a chip select.
- in the SPI mode **iobase** must be one of SBDR, SCCR or SDDR (as appropriate for the serial port being used) ORed with one of **RIO_CS_PE0**, **RIO_CS_PE1**, … **RIO_CS_PE7** to specify the Parallel Port E pin to use as a chip select. Note that SCDR is not available on RabbitCore modules with 16-bit memory, such as the RCM4000 series.
- in the RabbitNet mode **iobase** is a handle created by the appropriate RabbitNet library functions.

RETURN VALUE

0 — success

-ENODEV — bad parameter, or mode not supported

rio_protect

```
void rio_protect(RIO *r, word chan, word pair, word pre)
```

DESCRIPTION

Sets pin-pair protection for the given channel/pair.

Protection may only be set once for any channel/pair, until the next reset or power-up of the Rabbit RIO. When set, it ensures that the pins are never driven to a forbidden state. The forbidden state is defined as both pins being in the opposite state from their state at the time this function was called. For example, if **rio_protect()** is called for Channel 1, Pair 0; when Channel 1 Pin 0 is low, and Channel 1 Pin 1 is high, then [ch1p0,ch1p1] will never be driven subsequently to [high,low]. All other states are allowed.

The four possible states may be defined into three categories:

"safe" — both pins are in the state they were in at the time this function was called

"forbidden" — both pins are inverted with respect to the "safe" state

"active" — the remaining two states.

In addition to this prevention of "overlap," the Rabbit RIO will enforce a "dead time" between the two possible active states. During this dead time, the pins are forced into a safe state. The duration of the dead time is specified by the **pre** parameter, which gives a Rabbit RIO clock count for the dead time.

PARAMETERS

r	pointer to Rabbit RIO context as set up by **rio_init()**
chan	Rabbit RIO channel number (0–7)
pair	if 0, specifies channel pins 0 and 1 as the pin pair, otherwise, specifies channel pins 2 and 3
pre	"dead time" time constant, a number between 0 and 255 inclusive; the main Rabbit RIO clock will be divided by (**pre** + 1) to give the "dead time" time constant

NOTE: The actual dead time will vary between (**pre** + 1) and (2***pre** + 2) Rabbit RIO clocks, depending on the exact timing of the output drive signals. This **pre** value will be used for all protected pairs once it is set, that is, only the last programmed value will be used.

rio_reset

```
void rio_reset(RIO *r);
```

DESCRIPTION

Resets a Rabbit RIO. This performs the same function as a power-up reset. The Rabbit RIO will need to be set up again using the appropriate function calls in this library.

If the **RIO_ENABLE_HW_RESET** macro is defined, it is assumed that a Rabbit parallel port pin is connected to /RESET on the Rabbit RIO chip, and a hardware reset is performed. Otherwise, a software reset is performed by setting bit 7 of the master control register. By default, PE7 is used, but can be changed with the **RIO_HWRST_PORT** and **RIO_HWRST_PIN** macros.

PARAMETER

r	pointer to Rabbit RIO context to be reset, as set up by **rio_init()**.

rio_set_prescale

```
void rio_set_prescale(RIO *r, word pr);
```

DESCRIPTION

Sets the global clock prescaler. Each channel may individually choose either to use the prescaler output or the main (undivided) clock.

This function may be called at any time. Channels that are using the prescaled clock (if any) will be affected immediately.

PARAMETER

r	pointer to Rabbit RIO context to be reset, as set up by **rio_init()**
pr	prescaler value, a number between 0 and 255 inclusive; the main RIO clock will be divided by (**pr** + 1)

rio_set_chan_funcs

```
int rio_set_chan_funcs(RIO *r, word chan, word channel_opts, word
   pin0_func, word pin1_func, word pin2_func,
   word pin3_func, word limit);
```

DESCRIPTION

Sets the functionality for a specified Rabbit RIO channel.

The Rabbit RIO has eight channels (0–7). Each channel has four I/O pins, along with numerous options for performing various I/O functions such as PWM outputs and input counters. This function call provides an easy way to set up common functionality on a pin-by-pin basis.

Each pin's functionality may be specified individually using the **pin<n>_func** parameters, however not all combinations are possible. See the table following the function call for additional information.

It is always possible to set any pin to a normal "parallel" input or output. Other options do not generally allow mixing. In general, you cannot have one pin performing a special output function (such as PWM) while another pin performs a special input function (such as quadrature counter). To ensure correct operation, you may temporarily **#define RIO_VERBOSE** to get a printout in the Dynamic C **STDIO** window of any errors that are encountered.

PARAMETERS

r pointer to Rabbit RIO context as set up by **rio_init()**

chan Rabbit RIO channel number (0–7)

channel_opts overall options for this channel, a combination of the following values (ORed together):

RIO_OPTS_SYNC — use the GSYNC (global synch input pin) as the channel synch; if you specify this flag in this parameter, you must not set it on any of the individual pin functions

RIO_OPTS_INVERT — invert the GSYNC input sense; if not set, then a rising edge or high level is used; if set, a falling edge or low level is considered active

RIO_OPTS_LEVEL — specify level-sensitive GSYNC input, else defaults to single edge sensitivity

RIO_OPTS_PRESCALE — use the (global) prescaler for this channel, otherwise uses the undivided Rabbit RIO clock; the prescaler period is set using **rio_prescale()**

RIO_OPTS_SYNC_RESETS — forces the SYNC signal (if any defined) to reset the counter in this channel; this only has an effect if a synch signal is actually specified

pin0_func

pin1_func

pin2_func

pin3_func

These 4 parameters specify functionality, and possibly some flags, for each pin. The desired functionality should be specified using one of the following **RIO_PF_*** macros. Note that "Mn" refers to a match register setting (see **rio_set_match()**), and "n" refers to the pin number being set (0–3).

RIO_PF_PARIN — parallel input, always allowed

RIO_PF_PAROUT — parallel output, always allowed

RIO_PF_PAROUT_SEQ — parallel output sequenced over PxCR[7:4], always allowed

RIO_PF_PWM — PWM goes high at counter rollover, low on Mn

RIO_PF_PPM1 — PPM goes high on Mn, low on M(n + 1) mod 4

RIO_PF_PPM2 — PPM goes high on Mn, low on M(n + 2) mod 4

RIO_PF_PPM3 — PPM goes high on Mn, low on M(n + 3) mod 4

The PWM and PPM functionality may be used in any combination on any pins, however, PPMs require two match registers, so practical considerations may not allow more than two PPMs per channel, and those two should be both on odd or both on even pins. The use of PWM or PPM precludes the use of any of the following functionality.

RIO_PF_ONESHOT_T — defines this pin to be the output of a retriggerable monostable; another pin should be specified with **RIO_OPTS_SYNC** to define the oneshot trigger input, or GSYNC may be used for this purpose. The one-shot is retriggerable in that the output active time will be extended if triggered again while active. If the SYNC input pin has the **RIO_OPTS_LEVEL** flag, then the oneshot timer does not start until the trigger goes inactive (however, the output goes active at the start of the trigger).

RIO_PF_ONESHOT_C — similar to **RIO_PF_ONESHOT_T**, except that rather than using the internal clock to perform the timeout, it requires another pin to be defined as a clock input (i.e., this is a oneshot counter rather than a oneshot timer). Another pin in this channel must be defined as the counter input using **RIO_PF_CTR_UP**.

RIO_PF_IC_START — input capture start pin

RIO_PF_IC_STOP — input capture stop pin

The same pin cannot be used for start and stop. Instead, use **CTR_RUN**, etc.

RIO_PF_CTR_UP — increment the counter

RIO_PF_CTR_DOWN — decrement the counter

RIO_PF_CTR_RUN — counter increments while this pin is active, and forces **RIO_OPTS_LEVEL** to be in effect for this pin.

RIO_PF_CTR_MEAS_WIDTH — same as **RIO_PF_CTR_RUN**, except the counter is reset automatically at the end of the run. The final time value is stored in the "begin" register, which may be read back using **rio_get_begin()**.

RIO_PF_QD_I — quadrature counter in-phase (A)

RIO_PF_QD_Q — quadrature counter quadrature (B)

Two input pins must be defined for the Quadrature Decoder, one with QD_I and the other with QD_Q. Any other combination will not work as expected. In addition to the above basic functionality, each **pin<n>_func** parameter may have the following options ORed in:

RIO_OPTS_SYNC — select this pin as a SYNC input. If this flag is set with INVERT and/or LEVEL, then the polarity applies to both the synch functionality and the input (begin/end/inc/dec) functionality on this pin. Only one pin or **channel_opts** may specify a SYNC option. SYNC is implicitly used for the oneshot macros.

RIO_OPTS_INVERT — inverts the input/output sense. If not set, then a rising edge or high level is used. If set, a falling edge or low level is considered active.

RIO_OPTS_LEVEL— specifies level-sensitive input function, otherwise defaults to single-edge sensitivity.

RIO_OPTS_BOTHEDGES — specifies edge sensitivity for input functionality, with both rising and falling edges considered active.

RIO_OPTS_TOGGLE — enable the toggle function for output functionality. This imposes a fixed frequency/duty cycle "mask" over the output when it would otherwise be continuously high. The duty cycle may be set to 50%, 25%, 12.5%, etc., using the **rio_set_toggle()** function call. This function call reduces the driver current (e.g., relay coils or a thyristor gate drive). When using this option, it is wise to set the desired toggle duty via **rio_set_toggle()** before invoking this macro. Note that INVERT, LEVEl and BOTHEDGES are ignored for quadrature inputs.

limit	specifies a counter rollover limit — you may change this parameter later using **rio_set_count_limit()**

RETURN VALUE

Currently always 0.

If the Rabbit RIO is not behaving as expected, use **#define RIO_VERBOSE** to help check for error messages issued when this function is called.

SEE ALSO

rio_set_count_limit()

Certain pin functions may be ORed with one or more flags, as shown in the following examples. The options for using the flags are described in the table that follows.

```
RIO_PF_PARIN | RIO_OPTS_SYNC
RIO_PF_PWM | RIO_OPTS_INVERT | RIO_OPTS_TOGGLE
```

Function	pin0_func pin1_func pin2_func pin3_func	RIO_OPTS_SYNC	RIO_OPTS_INVERT	RIO_OPTS_LEVEL	RIO_OPTS_ BOTHEDGES	RIO_OPTS_TOGGLE
Parallel input (up to four per channel)	RIO_PF_PARIN	Yes	N/A	Yes, if also with RIO_ OPTS_SYNC	N/A	N/A
Parallel output (up to four per channel)	RIO_PF_ PAROUT	N/A	N/A	N/A	N/A	N/A
Sequenced parallel output (up to four per channel)	RIO_PF_ PAROUT_SEQ	N/A	N/A	N/A	N/A	N/A
PWM (up to four per channel)	RIO_PF_PWM	N/A	Yes	N/A	N/A	Yes
PPM (up to two per channel)	RIO_PF_PPM1	N/A	Yes	N/A	N/A	Yes
	RIO_PF_PPM2	N/A	Yes	N/A	N/A	Yes
	RIO_PF_PPM3	N/A	Yes	N/A	N/A	Yes
Triac driver (up to one per channel; requires another pin to be SYNC, and another to be RIO_ PF_CTR_UP)	RIO_PF_ ONESHOT_T	N/A	Yes, but not with RIO_ OPTS_BOTHEDGES	N/A	N/A	Yes
	RIO_PF_ ONESHOT_C	N/A	Yes, but not with RIO_ OPTS_BOTHEDGES	Yes, but not with RIO_ OPTS_BOTHEDGES	Yes, but not with RIO_ OPTS_LEVEL or RIO_ OPTS_INVERT	N/A
Input capture between two pulses (at most one per channel; requires two pins)	RIO_PF_IC_ START	Yes	Yes	Yes, but not with RIO_ OPTS_BOTHEDGES	Yes, but not with RIO_ OPTS_LEVEL	N/A
	RIO_PF_IC_ STOP	Yes	Yes	Yes, but not with RIO_ OPTS_BOTHEDGES	Yes, but not with RIO_ OPTS_LEVEL	N/A
Event counters (at most one per channel; requires two pins)	RIO_PF_CTR_ UP	Yes	Yes	Yes, but not with RIO_ OPTS_BOTHEDGES	Yes, but not with RIO_ OPTS_LEVEL	N/A
	RIO_PF_CTR_ DOWN	Yes	Yes	Yes, but not with RIO_ OPTS_BOTHEDGES	Yes, but not with RIO_ OPTS_LEVEL	N/A
Input capture (at most one per channel; requires one pin)	RIO_PF_CTR_ RUN	Yes	Yes	N/A (already in effect)	N/A	N/A
	RIO_PF_CTR_ MEAS_WIDTH	Yes	Yes	N/A (already in effect)	N/A	N/A
Quadrature Decoder (at most one per channel; requires two pins	RIO_PF_QD_I	Yes	N/A	N/A	N/A	N/A
	RIO_PF_QD_Q	Yes	N/A	N/A	N/A	N/A

rio_set_count_limit

```
void rio_set_count_limit(RIO *r, word chan, word limit);
```

DESCRIPTION

Sets the counter rollover limit for a given channel. This affects all special functions. The value must be 0–65535 inclusive. Values 1–65535 specify the rollover count directly. 0 means the counter rolls over at 65536 (i.e., has the full 16-bit range).

In general, this should be set to 0 for counter/timer functions, but may be set to other values for PWMs and PPMs to modify the fundamental output frequency. Also, quadrature counters can sometimes benefit from restricting the counter range to be the same as the number of lines on the encoder wheel (times 4).

The total cycle time for any RIO channel may be computed as

T = Tclk * P * L

where Tclk is the Rabbit RIO's main clock period,

P is 1 if the global prescaler is not used by this channel, or is the value provided to **rio_set_prescale()** plus 1

L is the value provided by "limit" in this function call (except that 0 is interpreted as 65536)

T gives the maximum timer interval (without wraparound), the fundamental PWM or PPM period, or the maximum count.

PARAMETERS

r	pointer to Rabbit RIO context as set up by **rio_init()**
chan	Rabbit RIO channel number (0–7)
limit	counter rollover limit

rio_set_toggle

```
void rio_set_toggle(RIO *r, word chan, word duty);
```

DESCRIPTION

Sets the toggle duty cycle and rate for a channel. This affects output functions when using a "special function" for the pin such as a PWM output. It imposes an additional high-frequency switching waveform over the normal channel output level. It does not affect pins used for parallel output.

NOTE: "Duty" is not given as a percentage.

The toggle function uses the LSBs of the channel counter. The highest toggle rate will be equal to the rate at which the counter LSB changes, i.e., the counter "clock" divided by 2.

For a given duty setting, the maximum toggle frequency will be determined by the most significant bit set in the duty parameter.

PARAMETER

r pointer to Rabbit RIO context as set up by **rio_init()**

chan Rabbit RIO channel number (0–7)

duty duty cycle setting as follows (F is the counter clock rate)

Parameter Value	Duty	Frequency	Maximum Left Shift
0	100%	—	—
1	50%	F/2	7
3	25%	F/4	6
7	12.5%	F/8	5
15	6.25%	F/16	4
31	3.125%	F/32	3
63	1.5625%	F/64	2
127	0.78125%	F/128	1
255	0.390625%	F/256	0

In addition, the "duty" values may be shifted left by one or more positions. This will reduce the toggling rate by the corresponding power of 2. For example, "3" gives a 25% duty at F/4, whereas "12" (= = 3<<2) also gives a 25% duty, but at F/16. Note that shifting left such that the parameter value exceeds 255 is not valid, so, for example, the maximum allowable shift for a 25% duty is 6 positions (giving 0xC0 or 192). There is only one allowable setting for 0.390625% duty. Because of this, the minimum frequency is always F/256.

Other values (i.e., with a noncontiguous set of 1-bits) should not normally be used, since the output waveform will exhibit subharmonics.

rio_set_match

```
void rio_set_match(RIO *r, word chan, word mask, word *mx);
```

DESCRIPTION

Sets any or all of the match values for a channel.

This affects some special functions that use match registers (usually the PWM and PPM functions).

The duty cycle of a PWM on pin n is set by loading a value into match register n. The duty cycle will be

100% x (mx[n] / count_limit)

If mx[n] is greater than the limit, the duty will be 100%.

PPMs are similar, except two match registers are used in order to specify both the phase and the duty cycle.

PARAMETERS

r	pointer to Rabbit RIO context as set up by **rio_init()**
chan	Rabbit RIO channel number (0–7)
mask	a bit set (bits 0,1,2,3) corresponding to the match register(s) that are to be changed
mx	pointer to four word values — for each bit set in "mask" the corresponding value will be loaded into the match register; mx[n] is ignored if bit n is not set in the mask.

rio_chan_cmd

```
void rio_chan_cmd(RIO *r, word chan, word cmds);
```

DESCRIPTION

Issues a channel command.

PARAMETERS

r pointer to Rabbit RIO context as set up by **rio_init()**

chan Rabbit RIO channel number (0–7)

cmds a bit mask of commands to issue (ORed together):

RIO_CR_ZERO —reset counter to zeros — this may be used to reset a counter/timer after it hits an "end" condition.

RIO_CR_SYNC — force a SYNC condition

RIO_CR_INC or **RIO_CR_BEGIN** — force increment or begin

RIO_CR_DEC or **RIO_CR_END** — force decrement or end

RIO_RESTART — restarts the counter/timer after it hits an "end" condition, without resetting it to zeros.

rio_get_begin

```
word rio_get_begin(RIO *r, word chan);
```

DESCRIPTION

Reads the channel "begin" register. This is a snapshot of the counter value at the time of the capture start condition.

PARAMETERS

r pointer to Rabbit RIO context as set up by **rio_init()**

chan Rabbit RIO channel number (0–7)

RETURN VALUE

Counter begin value (0–65535).

rio_get_end

```
word rio_get_end(RIO *r, word chan);
```

DESCRIPTION

Reads the channel “end” register. This is a snapshot of the counter value at the time of the capture stop condition.

PARAMETERS

r	pointer to Rabbit RIO context as set up by **rio_init()**
chan	Rabbit RIO channel number (0–7)

RETURN VALUE

Counter end value (0–65535).

rio_get_count

```
word rio_get_count(RIO *r, word chan);
```

DESCRIPTION

Reads the current channel counter value.

PARAMETERS

r	pointer to Rabbit RIO context as set up by **rio_init()**
chan	Rabbit RIO channel number (0–7)

RETURN VALUE

Channel count (0–65535).

rio_get_all

```
void rio_get_all(RIO *r, word chan, word *begin_end_count);
```

DESCRIPTION

Reads the current channel begin, end and count values.

This performs the same function as **rio_get_begin()**, **rio_get_end()**, and **rio_get_count()** in sequence, except that it is more efficient if all three values are required.

PARAMETERS

r pointer to Rabbit RIO context as set up by **rio_init()**

chan Rabbit RIO channel number (0–7)

begin_end_count pointer to array of three words (**unsigned int**). Words 0, 1, and 2 will be filled with the begin, end, and current count respectively.

rio_get_inputs

```
word rio_get_inputs(RIO *r, word chan);
```

DESCRIPTION

Reads the current channel pin state.

PARAMETERS

r pointer to Rabbit RIO context as set up by **rio_init()**

chan Rabbit RIO channel number (0–7)

RETURN VALUE

The input pin state. Each of the four channel pins states is read, and is returned in bits 0–3 of the return value. Note that all pins are sampled, including those currently set to outputs. The bit is 1 if the input is high (Vdd), 0 if low (Vss).

rio_get_status

```
word rio_get_status(RIO *r, word chan);
```

DESCRIPTION

Reads the current channel status register. This shows the pending channel events.

PARAMETERS

r	pointer to Rabbit RIO context as set up by **rio_init()**
chan	Rabbit RIO channel number (0–7)

RETURN VALUE

Status register. Bit mapping is as per the **RIO_SR_*** definitions documented with **rio_set_handler()**.

rio_get_mode

```
word rio_get_mode(RIO *r, word chan);
```

DESCRIPTION

Reads the current channel mode register.

PARAMETERS

r	pointer to Rabbit RIO context as set up by **rio_init()**
chan	Rabbit RIO channel number (0–7)

RETURN VALUE

Mode register. Bit mapping is as per the **RIO_SR_*** definitions.

rio_get_ier

```
word rio_get_ier(RIO *r, word chan);
```

DESCRIPTION

Reads the current channel interrupt enable register.

PARAMETERS

r	pointer to Rabbit RIO context as set up by **rio_init()**
chan	Rabbit RIO channel number (0–7)

RETURN VALUE

Interrupt enable register. Bit mapping is as per the **RIO_SR_*** definitions.

rio_get_pxcr

```
word rio_get_pxcr(RIO *r, word chan);
```

DESCRIPTION

Reads the channel pin control registers, and packs the results into a 16-bit return value.

PARAMETERS

r	pointer to Rabbit RIO context as set up by **rio_init()**
chan	Rabbit RIO channel number (0–7)

RETURN VALUE

4 nibbles of the **RIO_FUNC_*** value, with the low nibble from P0CR and the high nibble from P3CR.

rio_set_outputs

```
void rio_set_outputs(RIO *r, word chan, word parout);
```

DESCRIPTION

Sets the current channel parallel output pin state.

This does not affect the pin unless the pin function is set to **RIO_PF_PAROUT** or **RIO_PF_PAROUT_SEQ**. All parallel output pins are set to the value in the **parout** parameter with bit 0 applied to Pin 0, etc. If the bit is 1, the output is set high (Vdd), if 0 the output is set low (Vss).

PARAMETERS

r	pointer to Rabbit RIO context as set up by **rio_init()**
chan	Rabbit RIO channel number (0–7)
parout	parallel output state

rio_set_handler

```
void rio_set_handler(RIO *r, word chan, void (*h)(), word ier);
```

DESCRIPTION

Sets a channel event handler.

Each Rabbit RIO channel has eight different events defined. This function call allows the application to register a different handler function for each channel.

The handler function “h” should have the following prototype:

```
void my_handler(RIO * r, word chan, word sr);
```

When called, **r** and **chan** will indicate the Rabbit RIO instance and channel number. **sr** will contain a bit mask of the current event codes:

RIO_SR_DQE — Decrement/Quadrature/End

RIO_SR_IIB — Increment/In-Phase/Begin

RIO_SR_ROLL_D — counter rollover on decrement

RIO_SR_ROLL_I — counter rollover on increment

RIO_SR_MATCH3 — Match 3 condition

RIO_SR_MATCH2 — Match 2 condition

RIO_SR_MATCH1 — Match 1 condition

RIO_SR_MATCH0 — Match 0 condition

The handler may call most of the documented functions in this library, except **rio_init()**.

NOTE: The handler may be called in an interrupt context. Thus, it should not use **printf()** or other **STDIO** library functions. In general, you should be very careful about calling any non-reentrant functions. If you have not defined **RIO_ENABLE_INTERRUPTS**, then the above restriction does not apply.

NOTE: The handler should not attempt to reset the interrupt (by writing to the Rabbit RIO channel status register). This is done automatically.

PARAMETERS

r	pointer to Rabbit RIO context as set up by **rio_init()**
chan	Rabbit RIO channel number (0–7)
h	pointer to handler function as described above. Pass NULL to remove this handler.
ier	channel event interrupt enable mask. This should be a bitwise OR of the **RIO_SR_*** constants (of 0xFF for all). If an event mask bit is not set, then an interrupt will not occur for that event, however the handler may still get invoked for such events if **rio_tick()** is used to poll the Rabbit RIO. For efficiency, if you are turning off a channel handler by passing a null handler function, you should set this parameter to zero in order to prevent unnecessary interrupts from occurring. If using **rio_tick()** to poll for events, the handler will only be invoked if at least one of the specified events actually occurs.

rio_interrupt_enable

```
void rio_interrupt_enable(RIO *r, int enab);
```

DESCRIPTION

Enables or disables the hardware interrupt request line on a Rabbit RIO.

NOTE: This function must be called after **rio_init()** or **rio_reset()** since the hardware default is to disable interrupts.

PARAMETERS

r	pointer to Rabbit RIO context as set up by **rio_init()**
enab	if non-zero, enable interrupts, otherwise disable them

rio_tick

```
void rio_tick(RIO *r);
```

DESCRIPTION

Polls a Rabbit RIO and executes handlers for all channel events.

If you are not using interrupts (e.g., because the Rabbit RIO is connected via RabbitNet), then you can call this function from the application to poll for channel events, and call any registered handlers automatically.

PARAMETER

r	pointer to Rabbit RIO context as set up by **rio_init()**

printChanStat

```
void printChanStat(int RIO_channel);
```

DESCRIPTION

Prints the internal register values for a specific RIO channel to the Dynamic C **STDIO** window. The registers are printed in the following order:

MODE REGISTER
INTERRUPT ENABLE REGISTER
STATUS REGISTER
PORT 0 CONTROL REGISTER
PORT 1 CONTROL REGISTER
PORT 2 CONTROL REGISTER
PORT 3 CONTROL REGISTER
COUNT BEGIN MSB REGISTER
COUNT BEGIN LSB REGISTER
COUNT END MSB REGISTER
COUNT END LSB REGISTER
COUNT VALUE MSB REGISTER
COUNT VALUE LSB REGISTER

PARAMETER

RIO_channel Rabbit RIO channel number (0–7) of the registers to print:

0x1n = print heading info, 'n' is logical or of the following
1 = position the cursor at the top of the **STDIO** window
2 = print the register abbreviations

RabbitNet Configuration Function Calls

NOTE: The Dynamic C support for RabbitNet has not been tested completely for systems based on the Rabbit 4000 such as the sample RCM4110 RabbitCore module included with this Kit. A beta version is presently available on request through Technical Support,. See the ***RIO Programmable I/O Kit Getting Started*** instructions for Technical Support options.

Appendix B — Specifications

Figure B-1 shows the mechanical dimensions and layout for the Prototyping Board.

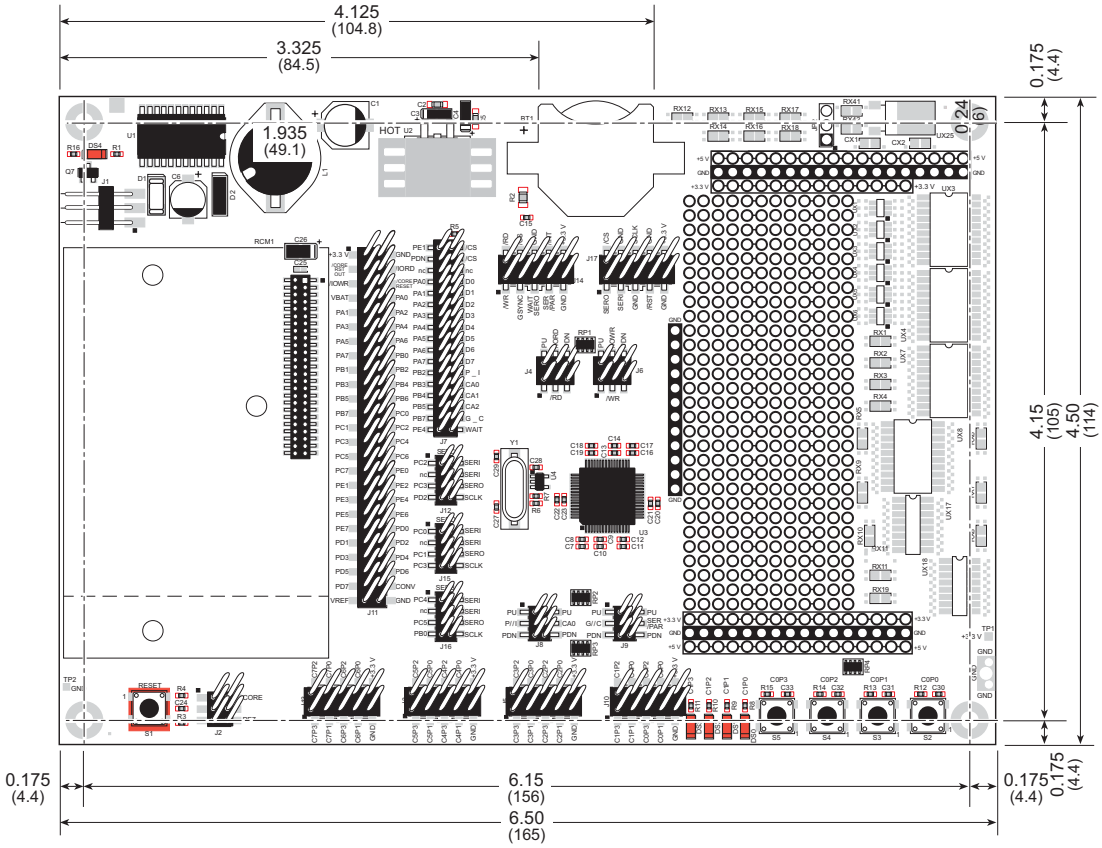


Figure B-1. Prototyping Board Dimensions

NOTE: All measurements are in inches followed by millimeters enclosed in parentheses. All dimensions have a manufacturing tolerance of ±0.01" (0.25 mm).

Table B-1 lists the electrical, mechanical, and environmental specifications for the Prototyping Board.

Table B-1. Prototyping Board Specifications

Parameter	Specification
Board Size	4.50" × 6.60" × 0.48" (114 mm × 165 mm × 12 mm)
Operating Temperature	0°C to +70°C
Humidity	5% to 95%, noncondensing
Input Voltage	8 V to 24 V DC
Maximum Current Draw (including user-added circuits)	800 mA max. for +3.3 V supply, 1 A total +3.3 V and +5 V combined
Prototyping Area	2.2" × 2.7" (56 mm × 68 mm) throughhole, 0.1" spacing, additional space for SMT components
Connectors	One 2 × 25 header socket, 1.27 mm pitch, to accept RabbitCore module One 1 × 3 IDC header, 0.1" pitch, for power-supply connection Seven 2 × 5 IDC headers, 0.1" pitch, for Rabbit RIO channels and serial signals Two 1 × 2 IDC headers, 0.1" pitch, for optional reset configuration Four 1 × 3 IDC headers, 0.1" pitch, for Rabbit RIO mode configuration One 2 × 4 IDC header, 2 mm pitch, for Rabbit RIO mode configuration One 2 × 17 IDC header, 2 mm pitch, for Rabbit RIO mode configuration

Appendix C — SPI Communication Mode

The initial release of the Rabbit RIO chip has a communications limitation when used in SPI mode—the communication state machine does not reset to the initial state when the /CS signal is taken away, which means that it is possible for communications with the Rabbit RIO to get out of synch if noise causes spurious edges on the clock line, or if the /CS line goes high before the end of the byte being transferred and the host starts a new packet the next time /CS goes low (instead of sending the remainder of the previous packet). As always, care should be taken to prevent noise on the communications lines to the Rabbit RIO. If this is still a concern, either the RabbitNet serial interface or the parallel interface can be used since this issue is not present in either of those modes (the communications state machine properly resets when /CS goes high in the RabbitNet mode).

Appendix D — Schematics

090-0228 RCM4100 Schematic

www.rabbit.com/documentation/schemat/090-0228.pdf

090-0231 Prototyping Board Schematic

www.rabbit.com/documentation/schemat/090-0231.pdf

090-0128 Programming Cable Schematic

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

You may use the URL information provided above to access the latest schematics directly.