PRODUCT MANUAL

Fox (LP3500)

C-Programmable Single-Board Computer

User's Manual

019–0111 • 081121–L

Fox (LP3500) User's Manual

Part Number 019-0111 • 081121–L • Printed in U.S.A.

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

1. INTRODUCTION

The LP3500 is a low-power single-board computer with built-in analog and digital I/O. Although the LP3500 was designed specifically for low-power applications and data logging, it has a host of features that make it attractive for other applications as well. Low power is often required in portable equipment operating from batteries or from solar power. The LP3500 is ideal for monitoring equipment or processes that are far-removed from a power supply, remote telemetry (RTUs), pipeline control and monitoring, well-head monitoring; and use on mobile equipment such as refrigeration trucks. An optional plastic enclosure and an LCD/keypad module are available.

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

1.1 LP3500 Description

The LP3500 is a low-power single-board computer that incorporates the powerful and low-EMI Rabbit 3000 microprocessor, flash memory, static RAM, digital I/O ports, A/D converter inputs, PWM outputs, RS-232/RS-485 serial ports, and both parallel and serial interfaces that allow other devices to be connected to the LP3500.

All aspects of the LP3500 are designed for low power consumption and operates at a variety of power levels, including a power-save mode, to fit customer-specified conditions at any given time. The CPU runs at a nominal speed of 7.4 MHz, and operates at 2.8 V to conserve power. The LP3500 consumes less than 20 mA when fully operational, and less than 100 µA when in the power-save mode. A replaceable coin-type battery will allow the LP3500 to operate in sleep mode for over 3 years. The LP3500 is normally powered from an external battery or power supply. When the unit is in the power-save mode, it can be awakened by an internal timer, an RS-232 signal, or via polling of an external input. The LP3500 can be switched from the power-save mode to full operation and back under program control. In addition, various sections of circuitry (such as the RS-232 ports) can be switched off under program control to further conserve power when not in use.

1.2 LP3500 Features

- Rabbit 3000® microprocessor operating at up to 7.4 MHz.
- 512K/128K static RAM and 512K/256K flash memory options.
- 26 digital I/O: 16 protected digital inputs and 10 high-current digital outputs provide sinking and sourcing outputs.
- 8 single-ended or 4 differential analog channels with Vcc monitoring option: 11-bit single-ended or 12-bit differential channels.
- 3 PWM outputs.
- Six serial ports

  1 RS-485

  3 RS-232 (one 5-wire and one 3-wire or three 3-wire), jumper option for logic-level outputs; Serial Port E has a “listen” and “wake-up” capability

  1 logic-level serial interface for optional add-ons

  1 asynchronous clocked serial port dedicated for programming
- Battery-backed real-time clock.
- Watchdog supervisor.

Two LP3500 models are available. Their standard features are summarized in Table 1.

Table 1. LP3500 Models

Feature	LP3500	LP3510
Microprocessor	Rabbit 3000 running at 7.4 MHz	
Static RAM	512K	128K
Flash Memory	512K	256K
A/D Converter Inputs (ranges from 0–1 V DC to 0–20 V DC, 4 channels may be individually configured for 4–20 mA)	Yes	No
C-form Bistable Relay	Yes	No

Appendix A provides detailed specifications.

The LP3500 can be mounted in two ways. It can be mounted to a panel or on a plastic-enclosure base, which allows I/O connections to be made using traditional connectors with 0.1" spacing. The LP3500 can also be inverted and mounted directly to mating connectors on a motherboard of the customer's design. The first approach is appropriate where I/O connections go directly to devices and switches. The second approach is appropriate where additional circuitry is incorporated on the motherboard.

1.3 Optional Add-Ons

- Plastic enclosure (can be wall-mounted or panel-mounted), which consists of a base and a cover for either the LP3500 by itself or an assembly made up of the LP3500 and the LP3500 Prototyping Board. The base is also available separately.
- The Prototyping Board included with the Tool Kit is a convenient means of interfacing to the LP3500 via the screw-terminal headers on the Prototyping Board. The Prototyping Board is also available for separate purchase.
- 4M and 8M SF1000 serial flash expansion cards.
- LCD/keypad module with 7-key keypad and seven LEDs.

Further details on the Prototyping Board, the plastic enclosure, and the LCD/keypad module are provided in Appendix B, Appendix C, and Appendix D.

Visit our Web site for up-to-date information about additional add-ons and features as they become available. The Web site also has the latest revision of this user's manual and schematics.

1.4 Development and Evaluation Tools

1.4.1 Tool Kit

A Tool Kit contains the hardware essentials you will need to develop applications with the LP3500 single-board computer. The items in the Tool Kit and their use are as follows.

- ***LP3500 Getting Started*** instructions.
- ***Dynamic C*** CD-ROM, with complete product documentation on disk.
- Programming cable, used to connect your PC serial port to the LP3500.
- Universal AC adapter, 12 V DC, 1 A (includes Canada/Japan/U.S., Australia/N.Z., U.K., and European style plugs). If you are using another power supply, it must provide 3 to 30 V DC.
- Prototyping Board with pushbutton switches, LEDs, and screw-terminal headers. The Prototyping Board can be hooked up to the LP3500 to demonstrate the I/O capabilities of the LP3500 and to provide a prototyping area for you to develop your own add-on circuits. The screw-terminal headers extend the LP3500's headers for development, and can also be used in a production environment.
- Plastic enclosure with four screws.
- Four standoffs with mounting screws.
- Screwdriver.
- ***Rabbit 3000 Processor Easy Reference*** poster.
- Registration card.

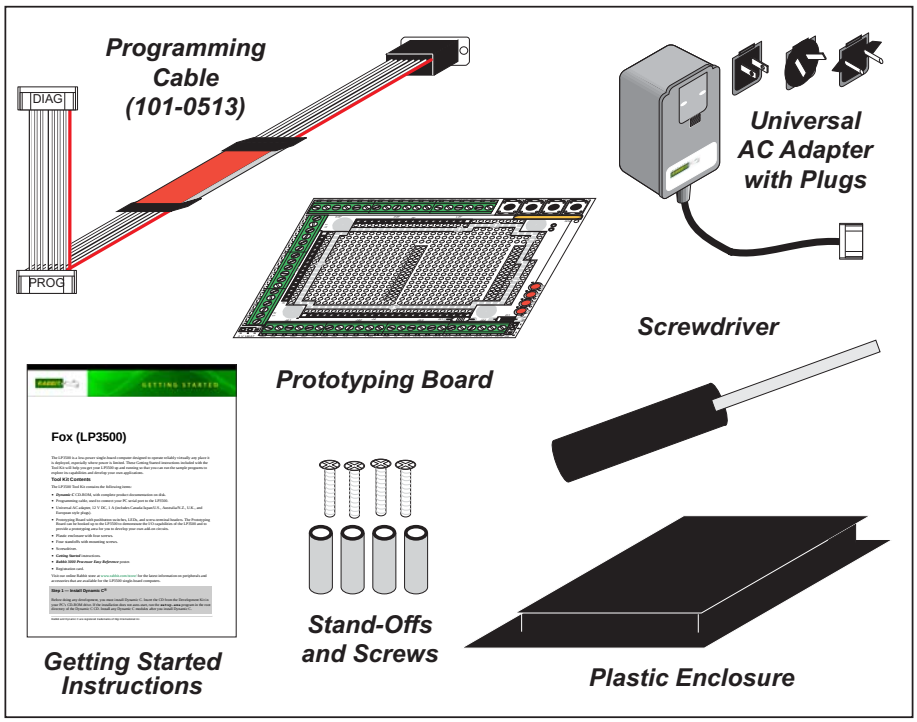


Figure 1. LP3500 Tool Kit

1.4.2 Software

The LP3500 is programmed using version 7.26P or later of Rabbit's Dynamic C. A compatible version is included on the Tool Kit CD-ROM. Library functions provide an easy-to-use interface for the LP3500. Software drivers for digital and analog I/O, and for serial communication are included with Dynamic C. Web-based technical support is included at no extra charge.

Starting with Dynamic C version 9.60, Dynamic C includes the popular µC/OS-II real-time operating system, point-to-point protocol (PPP), FAT file system, RabbitWeb, and other select libraries. Rabbit also offers for purchase the Rabbit Embedded Security Pack featuring the Secure Sockets Layer (SSL) and a specific Advanced Encryption Standard (AES) library.

In addition to the Web-based technical support included at no extra charge, a one-year telephone-based technical support subscription is also available for purchase.

Visit our Web site at www.rabbit.com for further information and complete documentation.

1.5 CE Compliance

Equipment is generally divided into two classes.

CLASS A	CLASS B
Digital equipment meant for light industrial use	Digital equipment meant for home use
Less restrictive emissions requirement: less than 40 dB µV/m at 10 m (40 dB relative to 1 µV/m) or 300 µV/m	More restrictive emissions requirement: 30 dB µV/m at 10 m or 100 µV/m

These limits apply over the range of 30–230 MHz. The limits are 7 dB higher for frequencies above 230 MHz. Although the test range goes to 1 GHz, the emissions from Rabbit-based systems at frequencies above 300 MHz are generally well below background noise levels.

The LP3500 has been tested and was found to be in conformity with the following applicable immunity and emission standards. The LP3510 is also CE qualified as it is a sub-version of the LP3500. Boards that are CE-compliant have the CE mark.

NOTE: Earlier versions of the LP3500 sold before 2003 that do not have the CE mark are ***not*** CE-complaint.

Immunity

The LP3500 series of single-board computers meets the following EN55024/1998 immunity standards.

- EN61000-4-3 (Radiated Immunity)
- EN61000-4-4 (EFT)
- EN61000-4-6 (Conducted Immunity)

Additional shielding or filtering may be required for a heavy industrial environment.

Emissions

The LP3500 series of single-board computers meets the following emission standards emission standards with the Rabbit 3000 spectrum spreader turned on and set to the normal mode.

- EN55022:1998 Class B
- FCC Part 15 Class B

Your results may vary, depending on your application, so additional shielding or filtering may be needed to maintain the Class B emission qualification.

1.5.1 Design Guidelines

Note the following requirements for incorporating the LP3500 series of single-board computers into your application to comply with CE requirements.

General

- The power supply provided with the Tool Kit is for development purposes only. It is the customer's responsibility to provide a CE-compliant power supply for the end-product application.
- When connecting the LP3500 to outdoor cables, the customer is responsible for providing CE-approved surge/lightning protection.
- Rabbit recommends placing digital I/O or analog cables that are 3 m or longer in a metal conduit to assist in maintaining CE compliance and to conform to good cable design practices. Rabbit also recommends using properly shielded I/O cables in noisy electromagnetic environments.

Safety

- For personal safety, all inputs and outputs to and from the LP3500 must not be connected to voltages exceeding SELV levels (42.4 V AC peak, or 60 V DC). Damage to the Rabbit 3000 microprocessor may result if voltages outside the design range of 0 V to 40 V DC are applied directly to any of its digital inputs.
- The lithium backup battery circuit on the LP3500 has been designed to protect the battery from hazardous conditions such as reverse charging and excessive current flows. Do not disable the safety features of the design.

1.5.2 Interfacing the LP3500 to Other Devices

There are two versions of the LCD/keypad module that may be used with the LP3500: without a bezel (Part No. 101-0601), and a remote panel-mounted version with bezel (Part No. 101-0541). The cable used to connect the LCD/keypad module should be less than 30 cm (12") to maintain CE compliance. Appendix C provides complete information for mounting and using the LCD/keypad module.

Since the LP3500 series of single-board computers is designed to be connected to other devices, good EMC practices should be followed to ensure compliance. CE compliance is ultimately the responsibility of the integrator. Additional information, tips, and technical assistance are available from your authorized Rabbit distributor, and are also available on our Web site at www.rabbit.com.

2. GETTING STARTED

Chapter 2 explains how to connect the programming cable and power supply to the LP3500.

2.1 LP3500 Connections

1. Use the 4-40 screws supplied with the Tool Kit to attach the metal standoffs to your LP3500 series board as shown in Figure 2.

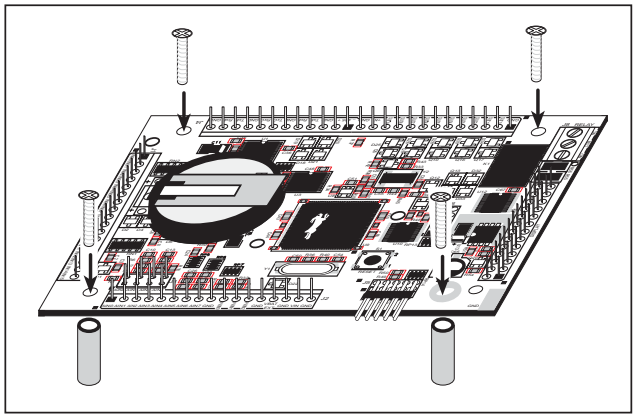


Figure 2. Attach Stand-Offs and Remove Battery Cap

2. Attach the LP3500 main board to the Prototyping Board as shown in Figure 3.

Press the pins from the headers on the bottom side of the LP3500 board firmly into the corresponding header sockets located at J1, J2, and J4 on the Prototyping Board.

NOTE: It is important that you line up the header pins on the LP3500 exactly with the corresponding header sockets J1, J2, and J4 on the Prototyping Board. The header pins may become bent or damaged if the pin alignment is offset, and the LP3500 will not work. Permanent electrical damage may also result if a misaligned LP3500 is powered up.

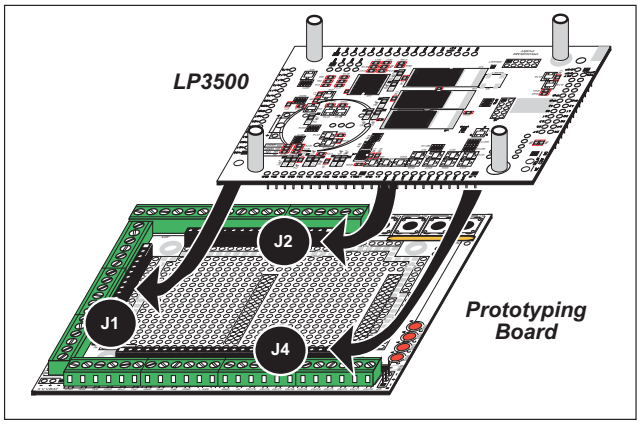


Figure 3. Attach LP3500 Main Board to Prototyping Board

3. Connect the programming cable to download programs from your PC and to program and debug the LP3500.

   NOTE: Use only the programming cable that has a red shrink wrap around the RS-232 level converter (Part No. 101-0513), which is supplied with the LP3500 Tool Kit. Other Rabbit programming cables with clear or blue shrink wrap might not be voltage-compatible or their connector sizes may be different.

Connect the 10-pin **PROG** connector of the programming cable to header J5 on the LP3500 board. Ensure that the colored edge lines up with pin 1 as shown. There is a small dot on the circuit board next to pin 1 of header J5. (Do not use the **DIAG** connector, which is used for monitoring only.) Connect the other end of the programming cable to a COM port on your PC. Make a note of the port to which you connect the cable, as Dynamic C will need to have this parameter configured. Note that COM1 on the PC is the default COM port used by Dynamic C.

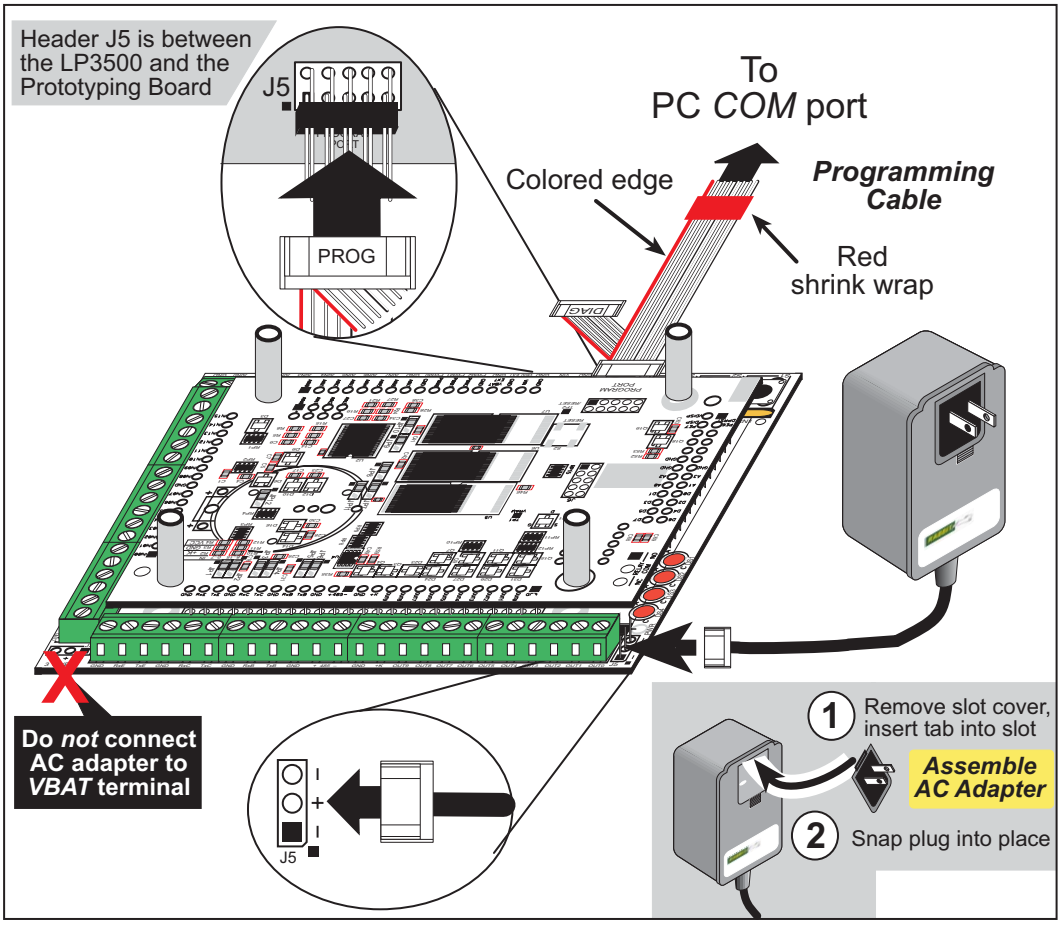


Figure 4. Programming Cable and Power Supply Connections

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 LP3500 Tool Kit. Note that not all RS-232/USB converters work with Dynamic C.

4. Connect the power supply.

First, prepare the AC adapter for the country where it will be used by selecting the plug. The LP3500 Tool 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 4, then press down on the spring-loaded clip below the plug assembly to allow the plug assembly to click into place.

Hook up the connector from the wall transformer to header J5 on the Prototyping Board as shown in Figure 4. The orientation of this connector is not important since the VIN (positive) voltage is the middle pin, and GND is available on both ends of the three-pin header J5.

> **NOTE:** Do ***not*** connect the AC adapter to the **VBAT** terminal on the Prototyping Board. The **VBAT** terminal supplies the backup battery voltage of 3 V, and the LP3500 may be damaged if subjected to the raw DC voltage from the AC adapter through the **VBAT** terminal.

5. Apply power.

Plug in the AC adapter. If you are using your own power supply, it must provide 3 V to 30 V DC—voltages outside this range could damage the LP3500.

> **NOTE:** A hardware reset may be done by pressing the RESET switch on the LP3500. The LP3500 may also be reset by unplugging the AC adapter, then plugging it back in. However, when the LP3500 is operating in the power-save mode, the backup battery will provide sufficient voltage to prevent a reset from happening, in which case you will have to press the RESET switch on the LP3500.

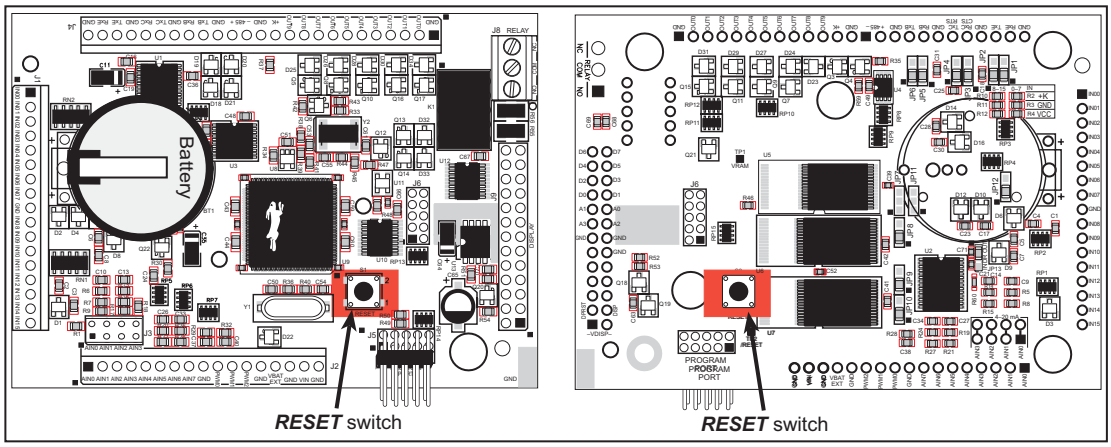


Figure 5. Locations of LP3500 RESET Switches

Reset switches are located on both sides of the LP3500 board.

2.2 Remove Battery Tab

The backup battery on the LP3500 has a plastic tab to protect the battery against discharging before the LP3500 is placed into service.

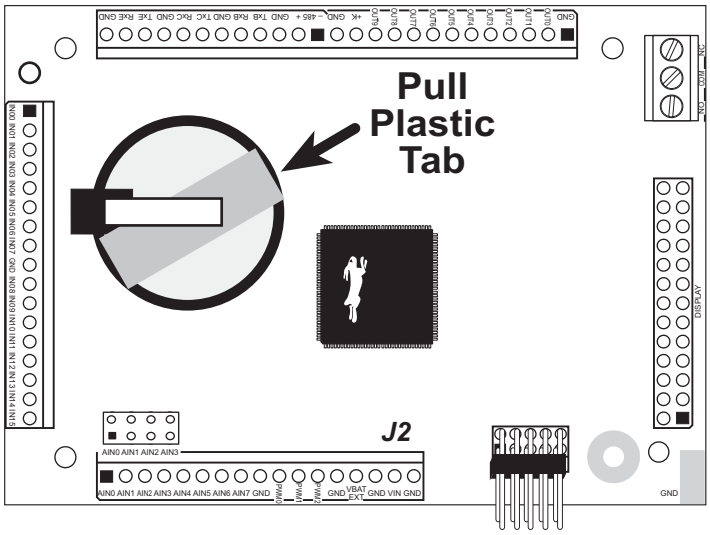


Figure 6. Remove Battery Tab

NOTE: Rabbit recommends that the battery tab not be removed until you are ready to place the LP3500 in normal service with regular power connected through header J2.

The backup battery protects the contents of the SRAM and keeps the real-time clock running when regular power to the LP3500 is interrupted. If you plan to use the real-time clock functionality in your application, you will need to set the real-time clock once you remove the plastic tab. Set the real-time clock using the onscreen prompts in the demonstration program. Alternatively, you may set the real-time clock using the **SETRTCKB.C** sample program from the Dynamic C **SAMPLES\RTCLOCK** folder. The **RTC_TEST.C** sample program in the Dynamic C **SAMPLES\RTCLOCK** folder provides additional examples of how to read and set the real-time clock.

2.3 Installing Dynamic C

If you have not yet installed Dynamic C version 7.26P (or a later version), do so now by inserting the Dynamic C CD in your PC's CD-ROM drive. The CD will auto-install unless you have disabled auto-install on your PC.

If the CD does not auto-install, click **Start > Run** from the Windows **Start** button and browse for the Dynamic C **`setup.exe`** file on your CD drive. Click **OK** to begin the installation once you have selected the **`setup.exe`** file.

The ***Dynamic C User's Manual*** provides detailed instructions for the installation of Dynamic C and any future upgrades.

> **NOTE:** If you have an earlier version of Dynamic C already installed, the default installation of the later version will be in a different folder, and a separate icon will appear on your desktop.

2.4 Starting Dynamic C

Once the LP3500 is connected to your PC and to a power source, start Dynamic C by double-clicking on the Dynamic C icon on your desktop or in your **Start** menu.

If you are using a USB port to connect your computer to the LP3500, choose **Options > Project Options** and select "Use USB to Serial Converter" on the **Communications** tab. Click **OK**.

Dynamic C assumes, by default, that you are using serial port COM1 on your PC when you are running a program. If you ***are*** using COM1, then Dynamic C should detect the LP3500 and go through a sequence of steps to cold-boot the LP3500 and to compile the BIOS. If the error message "Rabbit Processor Not Detected" appears, you have probably connected to a different PC serial port such as COM2, COM3, or COM4. You can change the serial port used by Dynamic C with the **OPTIONS** menu, then try to get Dynamic C to recognize the LP3500 by selecting **Reset Target/Compile BIOS** on the **Compile** menu. Try the different COM ports in the **OPTIONS** menu until you find the one you are connected to. If you still can't get Dynamic C to recognize the target on any port, then the hookup may be wrong or the COM port might not working on your PC.

Dynamic C automatically uses a maximum debug baud rate of 38,400 bps when an LP3500 series board is in use.

2.5 PONG.C

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

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

This program shows that the CPU is working.

2.6 Where Do I Go From Here?

> **NOTE:** If you purchased your LP3500 through a distributor or Rabbit partner, contact the distributor or partner first for technical support.

If there are any problems at this point:

- 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/.

If the sample program ran fine, you are now ready to go on to explore other LP3500 features and develop your own applications.

Chapter 3, "Subsystems," provides a description of the LP3500's features, Chapter 4, "Software," describes the Dynamic C software libraries and introduces some sample programs. These sample programs can be used as templates for applications you may wish to develop.

3. SUBSYSTEMS

Chapter 3 describes the principal subsystems for the LP3500.

- Power Modes
- Digital I/O
- Serial Communication
- A/D Converter Inputs (LP3500 only)
- PWM Outputs
- Relay Output Circuit (LP3500 only)
- Memory

Figure 7 shows these Rabbit-based subsystems designed into the LP3500.

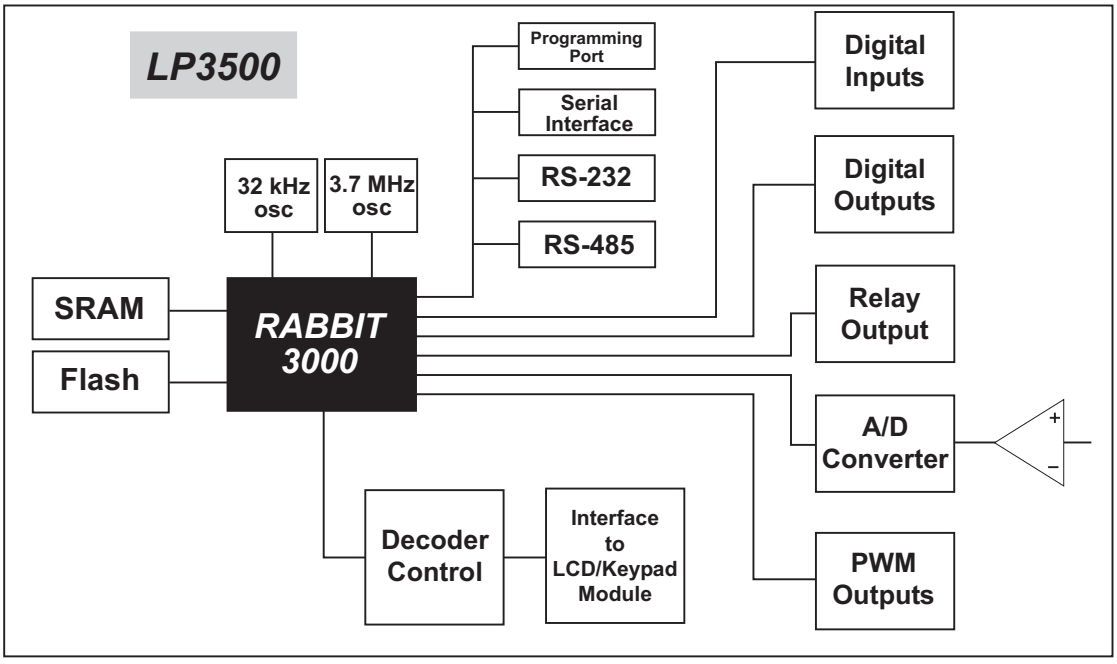


Figure 7. LP3500 Subsystems

3.1 LP3500 Pinouts

The LP3500 pinouts are shown in Figure 8.

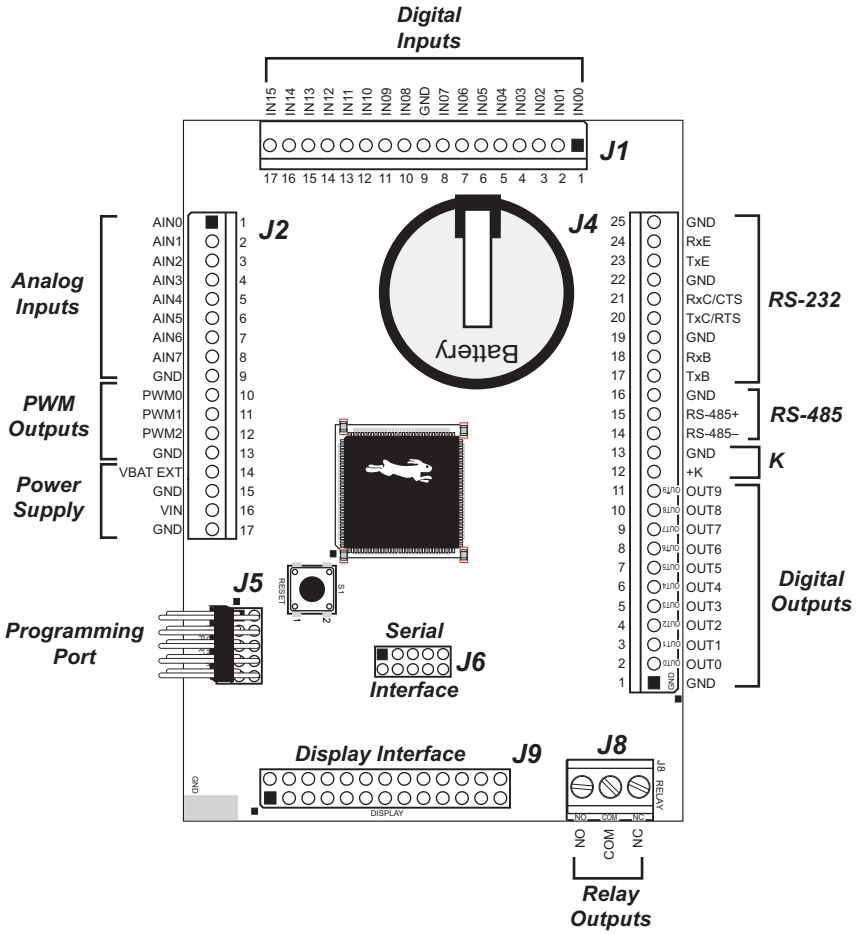


Figure 8. LP3500 Pinouts

NOTE: Although header J2 is installed on the LP3510, the associated analog I/O are not available on the LP3510. The relay screw-terminal header at J8 is also not installed on the LP3510. The power supply inputs on header J2 are still available.

3.1.1 Headers and Screw Terminals

Standard LP3500 models are equipped with two 1 × 17 headers (J1 and J2) with a pitch of 0.1", one 1 × 25 header (J4) with a pitch of 0.1", and one 1 × 3 screw terminal strip (J8). The Display Interface (J9) is a 2 × 13 header with a pitch of 0.1", and the Serial Interface is a 2 mm 2 × 5 socket.

A variety of commercially available connectors with a 0.1" pitch can be used to interface to the signals on headers J1, J2, and J4, or the Prototyping Board may be used to access these signals via screw-terminal headers.

Table 2. Software-Defined Power Modes

<table>
<tr><th colspan="2">Mode</th><th>Clock Frequency</th><th>Debug Capability</th><th>Code Restrictions</th><th>Power Source</th><th>Supports Relay Output</th><th>Supports Subsystems from Table 3</th><th>Typical Current Consumption</th></tr>
<tr><td>1</td><td rowspan="5">Normal Modes</td><td>7.3728 MHz</td><td rowspan="3">Normal</td><td rowspan="10">None</td><td rowspan="10">VIN</td><td rowspan="10">Yes</td><td rowspan="10">Yes</td><td rowspan="5">16 mA
↓
5 mA</td></tr>
<tr><td>2</td><td>3.6864 MHz</td></tr>
<tr><td>3</td><td>1.8423 MHz</td></tr>
<tr><td>4</td><td>1.2288 MHz</td><td rowspan="7">None</td></tr>
<tr><td>5</td><td>0.9216 MHz</td></tr>
<tr><td>6</td><td rowspan="5">Low-Power Modes</td><td>32.768 kHz</td><td rowspan="5">1.0 mA
↓ See note
140 µA</td></tr>
<tr><td>7</td><td>16.384 kHz</td></tr>
<tr><td>8</td><td>8.192 kHz</td></tr>
<tr><td>9</td><td>4.096 kHz</td></tr>
<tr><td>10</td><td>2.048 kHz</td></tr>
<tr><td colspan="2" rowspan="2">Power-Save Mode</td><td rowspan="2">2.048 kHz</td><td rowspan="2">See Section 3.2.1 and Section 3.2.2</td><td>VIN or external battery</td><td rowspan="2">No</td><td rowspan="2">No</td><td>140 µA</td></tr>
<tr><td>External battery (with linear regulator turned off)</td><td>70 µA</td></tr>
<tr><td colspan="3">Processor halted</td><td>None</td><td>SRAM and RTC updates</td><td>Onboard battery or ext. battery</td><td>No</td><td>No</td><td>46 µA</td></tr>
</table>

NOTE: The actual current consumption depends heavily on the SRAM writes in the user's program. See Section 3.2.2 for more information.

Table 2 lists the power modes based on clock frequency that can be defined in software using the **`powerMode`** function.

The LP3500 can operate at various power levels, depending on the clock frequency and on which subsystems on the board are turned off using the **`devPowerSet`** function. Table 3 lists the LP3500 subsystems that can be turned off with the **`devPowerSet`** function.

Table 3. LP3500 Subsystems That Can Be Turned Off

LP3500 Section	Description
RS-232	Receivers and transmitters are disabled, RxE remains active.
RS-485	Transmitter is disabled.
A/D Converter (LP3500 model only)	ADS7870 internal oscillator is turned off.
LCD/Keypad Module	LCD/keypad module is turned off.
All of the Above	All sections are disabled as described above.

NOTE: RxE always remains active when the above systems are turned off to allow the LP3500 to "listen" while it is in the power-save mode.

The LP3500 processor turns off automatically when VIN is removed, and the processor will not operate again until VIN is restored. The onboard battery provides backup for the SRAM and the real-time clock. VIN must be applied to the LP3500 in order to run or set the processor in any of the numbered modes listed in Table 2.

3.2.1 Setting the Power-Save Mode

The LP3500 can be placed in the power-save mode using one of three different software calls,

`serCommAlert`,

`timedAlert`, or

`digInAlert`,

depending on whether you wish to use Serial Port E, a simple timeout, or a digital input to trigger the LP3500 to resume operation in one of the other power modes.

If you call **`serCommAlert`**, then any activity on Serial Port E will trigger the LP3500 out of the low-power mode. If you call **`timedAlert`**, then the LP3500 is triggered out of the power-save mode when the specified time has elapsed. If you call **`digInAlert`**, then the LP3500 is triggered out of the power-save mode when the specified channel is activated.

In addition, **`digInAlert`** and **`serCommAlert`** have "backup" timeout parameters associated with them to wake up the LP3500 after a specified period even in the absence of the digital or serial triggers.

3.2.2 Operating in the Power-Save Mode

VIN may be removed to allow the LP3500 to operate using the external battery once the LP3500 is in Mode 10. At this point, the LP3500 will draw 200 µA after the subsystems listed in Table 3 are turned off. The LP3500's linear regulator may then be turned off using the **`setpowersource`** function call, and this will lower the current draw to 100 µA.

The LP3500 digital I/O can continue to operate (remember that an independent +K source is required for the digital outputs) using special software routines.

Here are some tips for when the LP3500 is in the power-save mode.

1. Do not write to the SRAM while the LP3500 is in power-save mode and you are relying solely on the onboard backup battery.
2. When the linear regulator is turned off, watch your current consumption carefully since too high a current draw could trigger a system reset and turn off the processor.

3.2.3 Resuming Normal-Power or Low-Power Operation

As long as VIN is still connected and the linear regulator has not been turned off, the LP3500 will return automatically to the previous power mode once the non-zero timeout specified in **`serCommAlert`**, **`timedAlert`**, or **`digInAlert`** has elapsed.

> **NOTE:** The processor will turn off if VIN is not available at the expiration of the timeout or if VIN is not available when a wake-up signal comes in through Serial Port E or the selected digital input.

When the timeout is set to 0, which corresponds to an indefinite timeout, the LP3500 may still be restored to a normal power mode from the power-save mode.

1. Make sure that raw DC power is available at VIN.
2. Turn the linear regulator back on using the **`setpowersource`** function call.
3. Use the **`rdPowerState`** function call to establish that the LP3500 is now operating from VIN. Note that this function only works with LP3500 models, which have the A/D converter.
4. Use the **`powerMode`** function call to set the desired power mode.

3.3 Digital I/O

3.3.1 Digital Inputs

The LP3500 has 16 digital inputs, IN00–IN15. The inputs are factory-configured to be pulled up to +K in banks of eight, but they can also be pulled up to Vcc or down to 0 V in banks of eight by changing a surface-mounted 0 Ω resistor as shown in Figure 9.

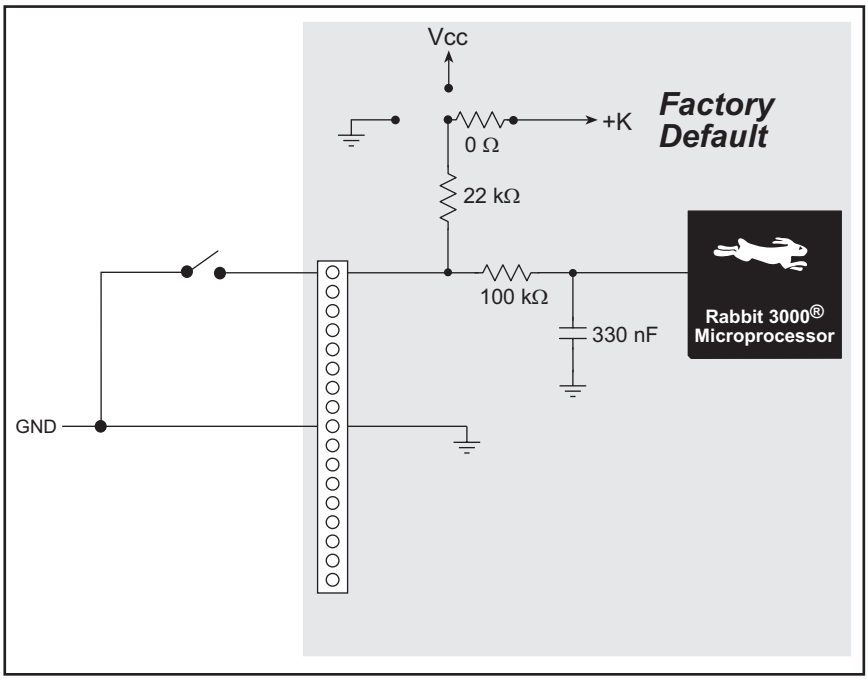


Figure 9. LP3500 Digital Inputs [Pulled Up—Factory Default]

The digital inputs are each fully protected over a range of 0 V to +36 V, and can handle short spikes of ±40 V. The actual switching threshold is approximately 1.40 V. Anything below this value is a logic 0, and anything above is a logic 1.

Pulling the digital inputs to Vcc will increase the current consumption by about 300 µA for each digital input.

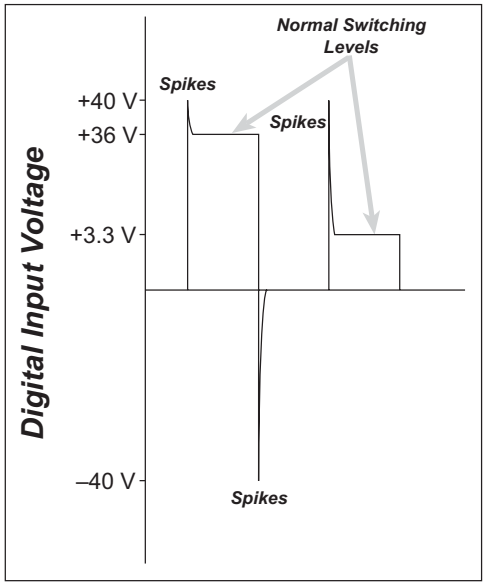


Figure 10. LP3500 Digital Input Protected Range

3.3.2 Digital Outputs

The LP3500 has 10 digital outputs: OUT0–OUT7 can each sink up to 200 mA, and OUT8–OUT9 can each source up to 200 mA at 36 V. Figure 11 shows a wiring diagram for using the digital outputs in a sinking or a souring configuration.

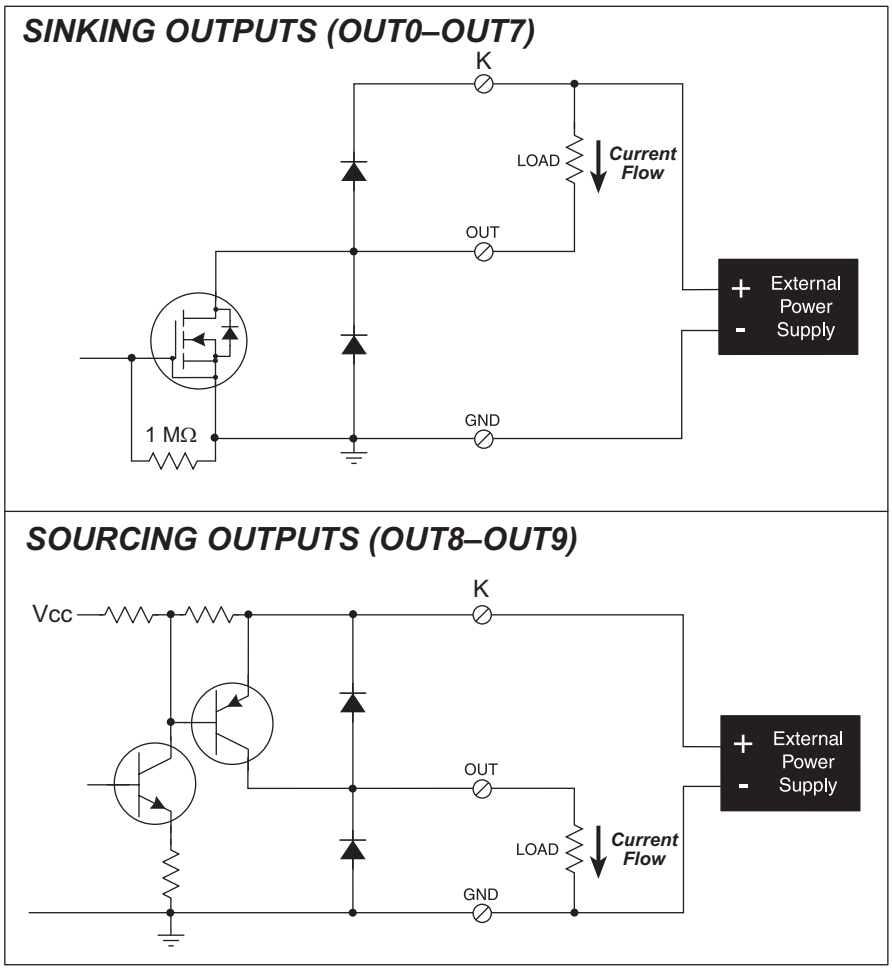


Figure 11. LP3500 Digital Outputs

TIP: Turn the outputs off (high for sourcing outputs and low for sinking outputs) to reduce current consumption.

When the LP3500 is connected to the Prototyping Board, a 0 Ω resistor on the Prototyping Board (R1) ties +K to VIN, the raw DC input voltage. Figure 12 shows the location of this 0 Ω resistor on the Prototyping Board.

NOTE: R1 on the Prototyping Broad must be removed to avoid damage to the power supplies *if* you are using the Prototyping Board with the LP3500 *and* you are using separate power supplies for VIN and K.

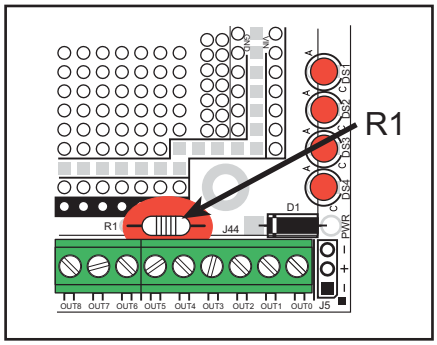


Figure 12. Location of R1 on Prototyping Board

When the LP3500 is used alone, remember to connect a power supply to +K (pin 12 on header J7). Your +K supply may be up to +36 V DC, and should be capable of delivering up to 2.0 A.

NOTE: If +K is not connected, the digital inputs may float, which may increase your current consumption.

3.4 Serial Communication

The LP3500 has three RS-232 serial ports that can set using the **`serMode`** software function call as one RS-232 serial channel (with RTS/CTS) and one 3-wire channel, or they may be set as three RS-232 (3-wire) channels. Table 4 summarizes the options.

Table 4. RS-232 Serial Communication Configurations

`serMode`	Serial Port		
	B	C	E
0	RS-232, 3-wire	RS-232, 3-wire	RS-232, 3-wire
1	RS-232, 5-wire	CTS/RTS	RS-232, 3-wire

The LP3500 also has one RS-485 serial channel (Serial Port F), one CMOS-level serial interface port (Serial Port D), and one CMOS-level serial channel that serves as the programming port (Serial Port A).

All six serial ports operate in an asynchronous mode. An asynchronous port can handle 7 or 8 data bits. A 9th bit address scheme, where an additional bit is sent to mark the first byte of a message, is also supported. Serial Port D and Serial Port A, the programming port, can be operated alternately in the clocked serial mode. In this mode, a clock line synchronously clocks the data in or out. Either of the two communicating devices can supply the clock. The LP3500 uses a 3.6864 MHz crystal, which is doubled to 7.3728 MHz. At this frequency, the LP3500 supports standard asynchronous baud rates up to a maximum of 921,600 bps.

Table 5 lists the use and the capabilities of the six serial ports.

Table 5. LP3500 Serial Port Uses and Capabilities

Serial Port	Use	Header Location	Synchronous Capability
A	Programming port or logic-level serial port	J5	Yes
B	3-wire RS-232	J4	No
C	3-wire RS-232 or RTS/CTS flow control for Serial Port B	J4	No
D	Serial interface port supports SPI device, also used by A/D converter on LP3500	J6	Yes
E	RS-232	J4	No
F	RS-485	J4	No

3.4.1 RS-232

The LP3500 RS-232 serial communication is supported by an RS-232 transceiver. This transceiver provides the voltage output, slew rate, and input voltage immunity required to meet the RS-232 serial communication protocol. Basically, the chip translates the Rabbit 3000's logic-level signals to RS-232 signal levels. Note that the polarity is reversed in an RS-232 circuit so that a +2.8 V output becomes approximately -7 V and 0 V is output as +7 V. The RS-232 transceiver also provides the proper line loading for reliable communication.

RS-232 can be used effectively at the LP3500's maximum baud rate for distances of up to 15 m.

Logic-level signals are also possible on Serial Ports B, C, and E by changing the 0 Ω surface-mounted resistor jumper settings at locations JP1–JP6.

Serial Port E can be set to "listen" and "wake up" the LP3500 when the unit is in a low-power mode.

3.4.2 RS-485

The LP3500 has one RS-485 serial channel, which is connected to Serial Port F on the Rabbit 3000 through an RS-485 transceiver. The half-duplex communication uses the Rabbit 3000's PG0 pin to control the transmit enable on the communication line.

The RS-485 transceiver used on the LP3500 is only capable of supporting a maximum baud rate of 64,000 bits/s.

The LP3500 can be used in an RS-485 multidrop network. Connect the 485+ to 485+ and 485– to 485– using single twisted-pair wires (nonstranded, tinned) as shown in Figure 13. Note that a common ground is recommended.

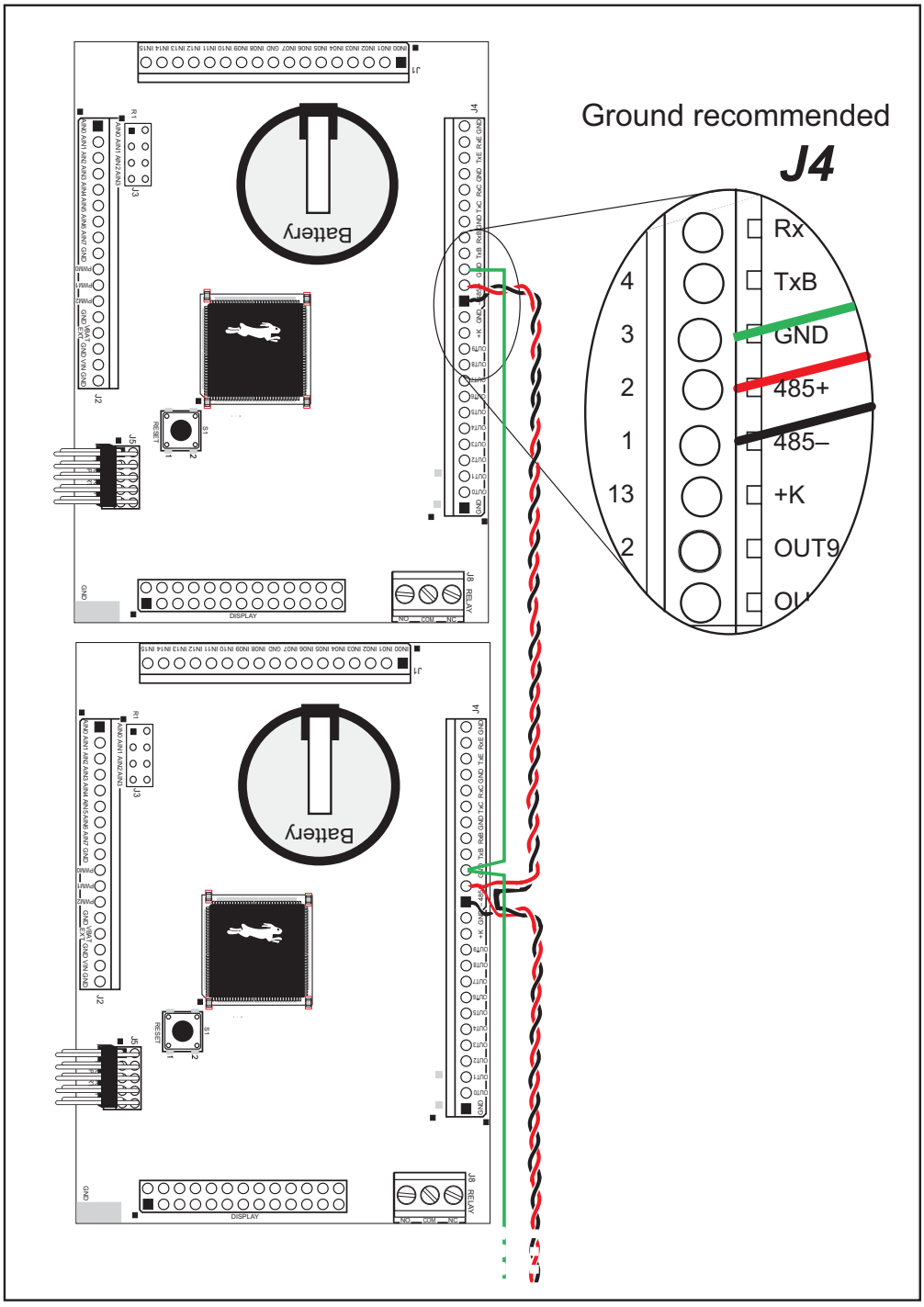


Figure 13. LP3500 Multidrop Network

Zener diodes are used in lieu of termination and bias resistors to minimize power consumption.

3.4.3 Serial Interface Port

The LP3500 offers a serial interface port at header J6, a 2 mm 2 × 5 socket. This port may be used to connect serial logic-level devices such as Rabbit's SF1000 serial flash expansion cards to Serial Port D on the Rabbit 3000. The PIO_0, PIO_1, PIO_2, and PF2_SRST signals are not used by the SF1000 serial flash expansion cards.

Figure 14 provides the pinout for the serial interface port.

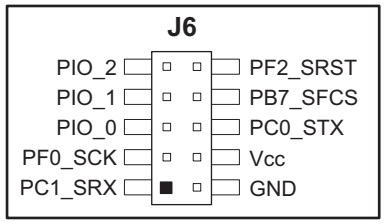


Figure 14. Serial Interface Port (Header J6) Pinout

3.4.4 Programming Port

The LP3500's serial programming port is accessed using header J5. The programming port uses the Rabbit 3000's Serial Port A for communication. Dynamic C uses the programming port to download and debug programs.

The programming port is also used for the following operations.

- Cold-boot the Rabbit 3000 on the LP3500 after a reset.
- Remotely download and debug a program over an Ethernet connection using the RabbitLink EG2110.
- Fast copy designated portions of flash memory from one Rabbit-based board (the master) to another (the slave) using the Rabbit Cloning Board.

In addition to Serial Port A, the Rabbit 3000 startup-mode (SMODE0, SMODE1), status, and reset pins are available on the programming port.

The two startup mode pins determine what happens after a reset—the Rabbit 3000 is either cold-booted or the program begins executing at address 0x0000.

The status pin is used by Dynamic C to determine whether a Rabbit microprocessor is present. The status output has three different programmable functions:

1. It can be driven low on the first op code fetch cycle.
2. It can be driven low during an interrupt acknowledge cycle.
3. It can also serve as a general-purpose CMOS output.

The /RESET_IN pin is an external input that is used to reset the Rabbit 3000 and the LP3500 onboard peripheral circuits. The serial programming port can be used to force a hard reset on the LP3500 by asserting the /RESET_IN signal.

Alternate Uses of the Serial Programming Port

All three clocked Serial Port A signals are available as

- a synchronous serial port
- an asynchronous serial port, with the clock line usable as a general CMOS input

The programming port may also be used as a serial port once the application is running. The SMODE pins may then be used as inputs and the status pin may be used as an output.

Refer to the ***Rabbit 3000 Microprocessor User's Manual*** for more information.

3.5 Display Interface

The LP3500 supports an interface with the parallel ports on the Rabbit 3000 via the Display Interface at header J9. The Display Interface may be used with Rabbit's LCD/keypad module, which offers an operator interface with seven keys and a 122 × 32 graphic display.

Figure 15 provides the pinout for the Display Interface.

Appendix C, "LCD/Keypad Module," provides further information on the LCD/keypad module.

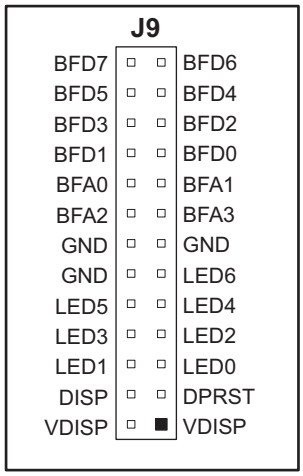


Figure 15. Display Interface (Header J9) Pinout

3.6 A/D Converter Inputs (LP3500 only)

The single 8-channel A/D converter chip used in the LP3500 (the LP3510 does not have analog capabilities) has a resolution of 12 bits for differential measurements or 11 bits for single-ended measurements. Four of the channels can be jumpered individually for 4–20 mA using jumpers across pins on header J3, and all 8 channels can be used over several software-scaled voltage ranges.

The A/D converter chip has an internal amplifier that works with the resistor divider network on the analog inputs as shown in Figure 16.

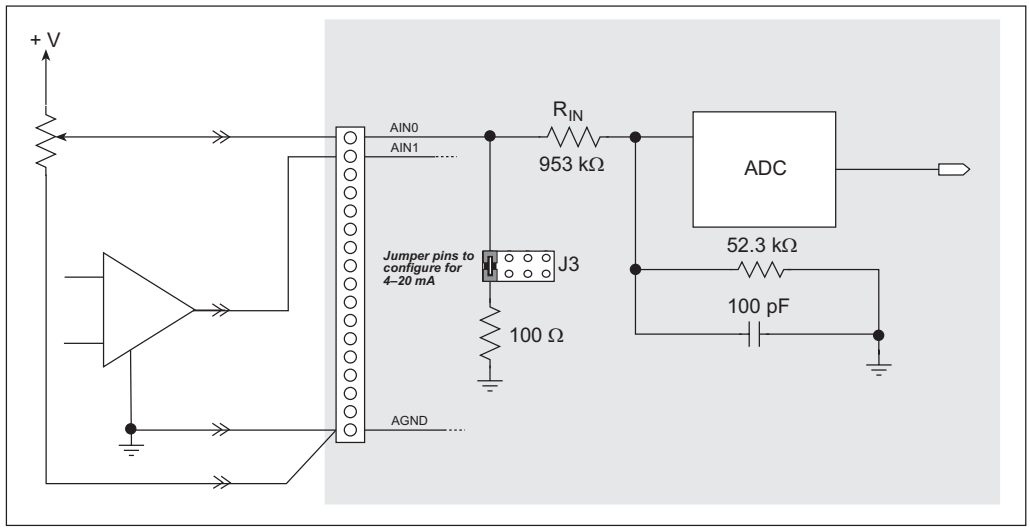


Figure 16. Buffered A/D Converter Inputs

The A/D converter chip can be programmed in software to operate over the voltage ranges shown in Table 6.

Table 6. A/D Converter Input Ranges

Gain Code	Multiplier	Voltage Range
0	1	0–20 V
1	2	0–10 V
2	4	0–5 V
3	5	0–4 V
4	8	0–2.5 V
5	10	0–2 V
6	16	0–1.25 V
7	20	0–1 V

Single-ended measurements are made by connecting the analog signal between an analog input channel (AIN0–AIN7) and AGND. Differential measurements are made by connecting a pair of differential analog signals to an adjacent pair of analog input channels (AIN0–AIN1, …, AIN6–AIN7). The A/D converter is only capable of converting positive voltages, and so will convert the *difference* between an adjacent pair of input channels, and must be scaled for a voltage range appropriate for the voltage differences.

Table 7 lists the jumper configurations for header J3 used to set the 4–20 mA and the voltage measurement options.

Table 7. Header J3 Configuration for Analog I/O Options

Analog Input Channel	Voltage Option (Factory Default)	4–20 mA Option
AIN0	Jumper “parked” on pin 2	Pins 1–2 connected
AIN1	Jumper “parked” on pin 4	Pins 3–4 connected
AIN2	Jumper “parked” on pin 6	Pins 5–6 connected
AIN3	Jumper “parked” on pin 8	Pins 7–8 connected

CAUTION: If you have enabled the 4–20 mA current option on any of the AIN0–AIN3 channels, be careful with any ***voltage*** sources that you might connect to these inputs. The voltage must be less than 2.5 V to keep the current across the 100 Ω resistor below the maximum allowed current.

The A/D converter inputs are factory-calibrated, and the calibration constants are stored in flash memory. You may calibrate the A/D converter inputs at a later time using the software functions described in Section 4.4.5, “A/D Converter Inputs.”

NOTE: If you are using a fixed voltage range, you should recalibrate your LP3500 at that range.

AIN7 can be used to monitor Vcc using the **`VccMonitorInit`** function. While Vcc can be monitored in all the power modes, Vcc monitoring is particularly useful when the LP3500 is being operated from an external battery to monitor the voltage being supplied by the battery.

The **`VccMonitorInit`** function requires the operation of the A/D converter, which consumes about 500 µA. The Vcc monitoring circuit itself consumes about 15 µA while it is engaged. Turn off **`VccMonitorInit()`** (and the A/D converter if it is not going to be used) when the test is done to extend your battery life.

3.7 PWM Outputs

The D/A conversion outputs are pulse-width modulated and scaled to provide an output from 0 V to Vcc (approx. 2.8 V).

Figure 17 shows the PWM outputs.

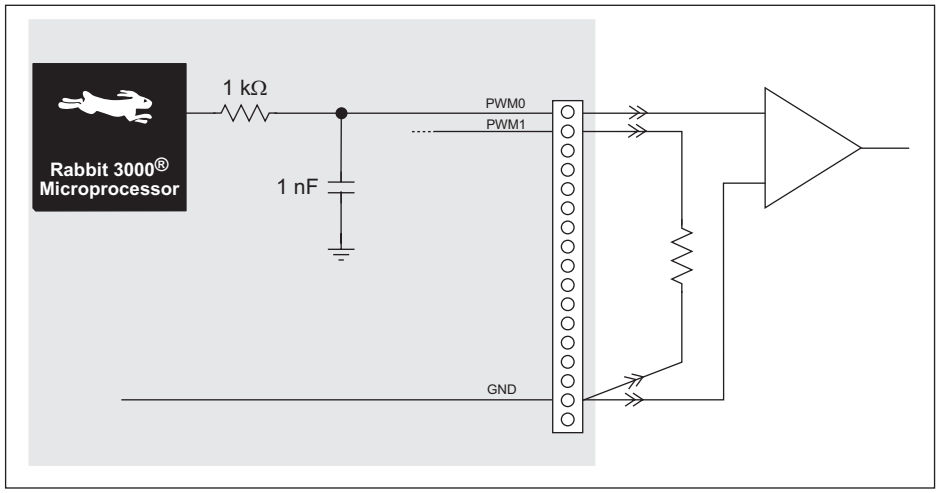


Figure 17. PWM Outputs

3.8 Relay Output Circuit (LP3500 only)

A bistable relay is stuffed on LP3500 models only at position K1, and the relay contacts are accessed via screw-terminal header J8. The relay can switch up to 1 A at 30 V DC.

The relay is set via Parallel Port PG4 on the Rabbit 3000, and is reset via Parallel Port PG5 by a 10 ms pulse. The relay resets when the LP3500 resets, and operates only in the normal power modes.

NOTE: The relay ***does not reset automatically*** when power is removed from the LP3500.

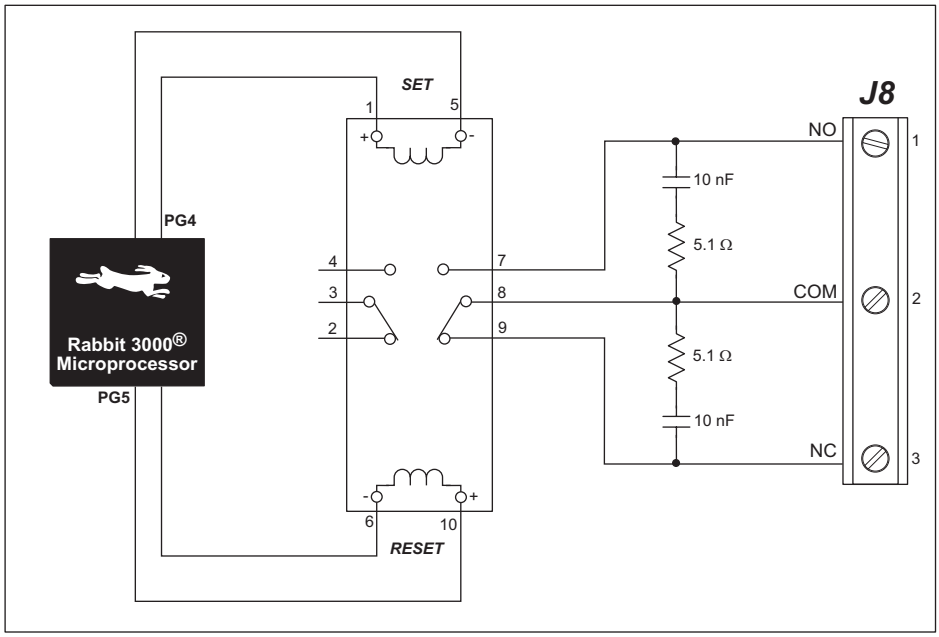


Figure 18. Relay Output Circuit

NOTE: Switching the relay may consume up to 120 mA during the roughly 10 ms that it takes for the relay to switch. Make sure that your power supply has sufficient capacity to handle this surge current to avoid putting the LP3500 into the power-save mode. The relay does not consume any current while it is in the NO or the NC position and is not switching.

3.9 Serial Programming Cable

The programming cable is used to connect the LP3500's serial programming port to a PC serial COM port. The programming cable converts the RS-232 voltage levels used by the PC serial port to the CMOS voltage levels used by the Rabbit 3000.

When the **PROG** connector on the programming cable is connected to the LP3500's serial programming port at header J5, programs can be downloaded and debugged over the serial interface.

The **DIAG** connector of the programming cable may be used on header J5 of the LP3500 with the LP3500 operating in the Run Mode. This allows the programming port to be used as a regular serial port.

3.9.1 Changing Between Program Mode and Run Mode

The LP3500 is automatically in Program Mode when the **PROG** connector on the programming cable is attached, and is automatically in Run Mode when no programming cable is attached. When the Rabbit 3000 is reset, the operating mode is determined by the state of the SMODE pins. When the programming cable's **PROG** connector is attached, the SMODE pins are pulled high, placing the Rabbit 3000 in the Program Mode. When the programming cable's **PROG** connector is not attached, the SMODE pins are pulled low, causing the Rabbit 3000 to operate in the Run Mode. See Figure 19.

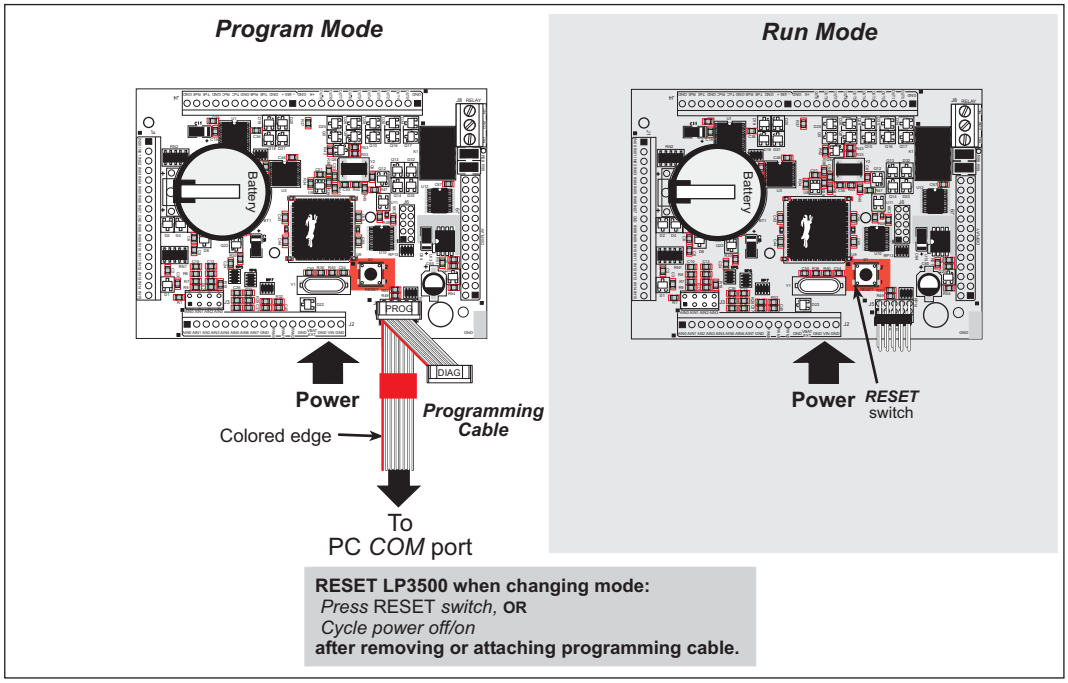


Figure 19. LP3500 Program Mode and Run Mode Set-Up

A program "runs" in either mode, but can only be downloaded and debugged when the LP3500 is in the Program Mode.

Refer to the ***Rabbit 3000 Microprocessor User's Manual*** for more information on the programming port and the programming cable.

3.9.2 Standalone Operation of the LP3500

The LP3500 must be programmed via the Prototyping Board or via a similar arrangement on a customer-supplied board. Once the LP3500 has been programmed successfully, remove the serial programming cable from the programming connector and reset the LP3500. The LP3500 may be reset by cycling the power off/on or by pressing the **RESET** button on the Prototyping Board. The LP3500 module may now be removed from the Prototyping Board for end-use installation.

CAUTION: Disconnect power to the Prototyping Board or other boards when removing or installing your LP3500 to protect against inadvertent shorts across the pins or damage to the LP3500 if the pins are not plugged in correctly. Do not reapply power until you have verified that the LP3500 is plugged in correctly.

3.10 Other Hardware

3.10.1 Spectrum Spreader

The Rabbit 3000 features a spectrum spreader, which helps to mitigate EMI problems. By default, the spectrum spreader is on automatically, but it may also be turned off or set to a stronger setting. The means for doing so is through a simple global macro as shown below.

1. Select the "Defines" tab from the Dynamic C **Options > Project Options** menu.
2. Normal spreading is the default, and usually no entry is needed. If you need to specify normal spreading, add the line

   ```
   ENABLE_SPREADER=1
   ```

   For strong spreading, add the line

   ```
   ENABLE_SPREADER=2
   ```

   To disable the spectrum spreader, add the line

   ```
   ENABLE_SPREADER=0
   ```

   NOTE: The strong spectrum-spreading setting is not recommended since it may limit the maximum clock speed or the maximum baud rate. It is unlikely that the strong setting will be used in a real application.
3. Click **OK** to save the macro. The spectrum spreader will now be set to the state specified by the macro value whenever you are in the project file where you defined the macro.

NOTE: Refer to the ***Rabbit 3000 Microprocessor User's Manual*** for more information on the spectrum-spreading setting and the maximum clock speed.

3.11 Memory

3.11.1 SRAM

The LP3500 module is designed to accept 128K to 512K of SRAM at U5. The standard LP3500 modules come with 512K of SRAM.

3.11.2 Flash Memory

The LP3500 is also designed to accept 256K to 512K of flash memory at U6 and U7. The standard LP3500 modules comes with two 256K flash memory chips.

> **NOTE:** Rabbit recommends that any customer applications should not be constrained by the sector size of the flash memory since it may be necessary to change the sector size in the future.

A Flash Memory Bank Select jumper configuration option based on 0 Ω surface-mounted resistors exists at header JP10. This option, used in conjunction with some configuration macros, allows Dynamic C to compile two different co-resident programs for the upper and lower halves of the 256K flash in such a way that both programs start at logical address 0000. This is useful for applications that require a resident download manager and a separate downloaded program. See Rabbit's Technical Note 218, ***Implementing a Serial Download Manager for a 256K Flash***, in the online documentation for details.

4. SOFTWARE

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 4 provides the libraries, function calls, and sample programs related to the LP3500.

You have a choice of doing your software development in the flash memory or in the static RAM included on the LP3500. The flash memory and SRAM options are selected with the **Options > Project 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 developed 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. Due to the volatility of the flash memory market, the LP3500 and Dynamic C were designed to accommodate flash devices with various sector sizes.

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 2000/NT and later. Rabbit's Technical Note TN257, ***Running Dynamic C® With Windows Vista®***, in the online documentation set provides additional information about using Windows Vista® with versions of Dynamic C prior to v. 9.60.

Programs can be downloaded at baud rates of up to 460,800 bps after the program compiles.

Dynamic C has a number of standard features.

- Full-feature source and/or assembly-level debugger, no in-circuit emulator required.
- Royalty-free TCP/IP stack with source code and most common protocols.
- Hundreds of functions in source-code libraries and sample programs:
  - Exceptionally fast support for floating-point arithmetic and transcendental functions.
  - RS-232 and RS-485 serial communication.
  - Analog and digital I/O drivers.
  - I^2C, SPI, GPS, encryption, file system.
  - LCD display and keypad drivers.
- Powerful language extensions for cooperative or preemptive multitasking
- Loader utility program to load binary images into Rabbit targets in the absence of Dynamic C.
- Provision for customers to create their own source code libraries and augment on-line help by creating "function description" block comments using a special format for library functions.
- Standard debugging features:
  - Breakpoints—Set breakpoints that can disable interrupts.
  - Single-stepping—Step into or over functions at a source or machine code level, µC/OS-II aware.
  - Code disassembly—The disassembly window displays addresses, opcodes, mnemonics, and machine cycle times. Switch between debugging at machine-code level and source-code level by simply opening or closing the disassembly window.
  - Watch expressions—Watch expressions are compiled when defined, so complex expressions including function calls may be placed into watch expressions. Watch expressions can be updated with or without stopping program execution.
  - Register window—All processor registers and flags are displayed. The contents of general registers may be modified in the window by the user.
  - Stack window—shows the contents of the top of the stack.
  - Hex memory dump—displays the contents of memory at any address.
  - **STDIO** window—`printf` outputs to this window and keyboard input on the host PC can be detected for debugging purposes. `printf` output may also be sent to a serial port or file.

4.1 Upgrading Dynamic C

4.1.1 Patches and Bug Fixes

Dynamic C patches that focus on bug fixes are available from time to time. Check the Web site www.rabbit.com/support/ for the latest patches, workarounds, and bug fixes.

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

4.1.2 Extras

Dynamic C installations are designed for use with the board they are included with, and are included at no charge as part of our low-cost kits.

Starting with Dynamic C version 9.60, Dynamic C includes the popular µC/OS-II real-time operating system, point-to-point protocol (PPP), FAT file system, RabbitWeb, and other select libraries. Rabbit also offers for purchase the Rabbit Embedded Security Pack featuring the Secure Sockets Layer (SSL) and a specific Advanced Encryption Standard (AES) library.

In addition to the Web-based technical support included at no extra charge, a one-year telephone-based technical support subscription is also available for purchase.

Visit our Web site at www.rabbit.com for further information and complete documentation.

4.2 Sample Programs

Sample programs are provided in the Dynamic C **Samples** folder. The sample program **PONG.C** demonstrates the output to the **STDIO** window.

The various directories in the **Samples** folder contain specific sample programs that illustrate the use of the corresponding Dynamic C libraries.

The **LP3500** folder provides sample programs specific to the LP3500. Each sample program has comments that describe the purpose and function of the program. Follow the instructions at the beginning of the sample program.

To run a sample program, open it with the **File** menu (if it is not still open), compile it using the **Compile** menu, and then run it by selecting **Run** in the **Run** menu. The LP3500 must be in the **Program** mode (see Section 3.9, "Serial Programming Cable") and must be connected to a PC using the programming cable as described in Section 2.1, "LP3500 Connections."

Appendix F, "Running a Sample Program," takes you through the steps of running one of the sample programs.

4.2.1 Power Modes

The following sample program is found in the **POWER** subdirectory in **SAMPLES\LP3500**.

- **POWER.C**—This program demonstrates switching from the normal raw DC power source to an external battery using the Prototyping Board. Pressing a switch will change from the power source and will be displayed by flashing LEDs.
- **LOWPWRDEMO.C**—This program demonstrates a low-power mode with the normal power source connected to the LP3500.

4.2.2 Digital I/O

The following sample programs are found in the **IO** subdirectory in **SAMPLES\LP3500**.

- **DIGIN.C**—Demonstrates the use of the digital inputs. Using the Prototyping Board, you can see an input channel toggle from HIGH to LOW when pressing a pushbutton on the Prototyping Board.
- **DIGOUT.C**—Demonstrates the use of the high-current outputs configured as either sinking or sourcing outputs. Using the Prototyping Board, you can see an LED toggle on/off via a high-current output.
- **DIGBANKIN.C**—Demonstrates the use of the digital inputs. Using the Prototyping Board, you can see a bank of input channels toggle from HIGH to LOW when pressing a pushbutton on the Prototyping Board.
- **DIGBANKOUT.C**—Demonstrates the use of the high-current outputs configured as either sinking or sourcing outputs. Using the Prototyping Board, you can see a bank of channels toggle the corresponding LEDs on/off via high-current outputs.

4.2.3 Serial Communication

The following sample programs are found in the **RS232** subdirectory in **SAMPLES\LP3500**.

- **SIMPLE3WIRE.C**—This program demonstrates basic initialization for a simple RS-232 3-wire loopback displayed in the **STDIO** window.

The following sample programs are found in the **RS485** subdirectory in **SAMPLES\LP3500**.

- **SIMPLE485MASTER.C**—This program demonstrates a simple RS-485 transmission of lower case letters to a slave LP3500. The slave will send back converted upper case letters back to the master LP3500 and display them in the **STDIO** window. Use **SIMPLE485SLAVE.C** to program the slave LP3500.
- **SIMPLE485SLAVE.C**—This program demonstrates a simple RS-485 transmission of lower case letters to a slave LP3500. The slave will send back converted upper case letters back to the master LP3500 and display them in the **STDIO** window. Use **SIMPLE485MASTER.C** to program the master LP3500.

4.2.4 A/D Converter Inputs

The following sample programs are found in the **ADC** subdirectory in **SAMPLES\LP3500**.

- **AD_RDVOLT_ALL.C**—This program reads and displays the voltage and equivalent values of each single-ended A/D converter channel. Coefficients are read from the A/D converter's simulated EEPROM in flash memory to compute the equivalent voltages. Computed raw data and equivalent voltages are displayed in the **STDIO** window.
- **AD_RDVOLT_CH.C**—This program reads and displays the voltage and equivalent values of one single-ended A/D converter channel. Coefficients are read from the A/D converter's simulated EEPROM in flash memory to compute the equivalent voltages. Computed raw data and equivalent voltages are displayed in the **STDIO** window.
- **AD_RDDIFF_CH.C**—This program demonstrates reading one differential A/D converter channel. Coefficients are read from the A/D converter's simulated EEPROM in flash memory to compute the equivalent voltages. Computed raw data and equivalent voltages are displayed in the **STDIO** window.
- **AD_RDMA_CH.C**—This program demonstrates reading one milliampere A/D converter channel. Coefficients are read from the A/D converter's simulated EEPROM in flash memory to compute the equivalent currents. Computed raw data and equivalent currents are displayed in the **STDIO** window.
- **AD_SAMPLE.C**—This program demonstrates how to use the A/D low-level driver. The program will display the average voltage that is present on an A/D converter channel. The particular channel and the number of samples may be changed by the user.

4.2.5 PWM Outputs

The following sample program is found in the **IO** subdirectory in **SAMPLES\LP3500**.

- **PWMOUT.C**—This program demonstrates the PWM functions. It will set the PWM channels, PWM0–PWM2, to the following duty cycles:

  PWM Channel 0 to 10%
  PWM Channel 1 to 25%
  PWM Channel 2 to 50%

  All activity will be displayed in the **STDIO** window.

4.2.6 Relay Output

The following sample program is found in the **RELAY** subdirectory in **SAMPLES\LP3500**.

- **SWRELAY.C**—This program demonstrates the relay-switching function call operating on normal power source. Use the pushbutton switches on the Prototyping Board to switch the relay between the SET (NO) and RESET (NC) positions. All activity will be displayed with the LEDs.

4.2.7 Vcc Monitoring

The following sample program is found in the **POWER** subdirectory in **SAMPLES\LP3500**.

- **VCCMONITOR.C**—This program demonstrates the Vcc monitoring function on AIN7. All activity will be displayed in the **STDIO** window

4.2.8 LP3500 Calibration

The following sample programs are found in the **ADC** subdirectory in **SAMPLES\LP3500**.

- **AD_CAL_ALL.C**—This program demonstrates how to recalibrate all single-ended A/D converter channels using two known voltages to generate constants for each channel, and will be written into the user block data area. The program uses the **STDIO** window to display the voltage that is being monitored.

  NOTE: This sample program will overwrite the calibration constants set at the factory.

- **AD_CAL_CHAN.C**—This program demonstrates how to recalibrate one single-ended A/D converter channel using two known voltages to generate constants for each channel, and will be written into the user block data area. The program uses the **STDIO** window to display the voltage that is being monitored.

  NOTE: This sample program will overwrite the calibration constants set at the factory.

- **AD_CALDIFF_CH.C**—This sample program demonstrates how to recalibrate one differential A/D converter channel using two known voltages to generate constants for that channel and rewrite the constants into the user block data area. The program uses the **STDIO** window to display the voltage that is being monitored.

  NOTE: This sample program will overwrite the calibration constants set at the factory.

- **AD_CALMA_CH.C**—This sample program demonstrates how to recalibrate one A/D converter channel operating in the 4–20 mA current mode using two known currents to generate two coefficients, gain and offset, which are rewritten into the user block data area. The program uses the **STDIO** window to display the current that is being monitored.

  NOTE: This sample program will overwrite the calibration constants set at the factory.

4.2.9 LCD/Keypad Module Sample Programs

Sample programs for the LCD/keypad module are described in Section C.8.

4.3 LP3500 Libraries

One library directory provides the function calls that are used to develop applications for the LP3500.

- **LP3500**—libraries associated with features specific to the LP3500. The functions in the **LIB\Rabbit3000\LP35xx.LIB** library are described in Section 4.4, "LP3500 Function Calls."

The LCD/keypad module functions are described in Section C.7. Other generic functions applicable to all devices based on the Rabbit 3000 microprocessor are described in the ***Dynamic C Function Reference Manual***.

4.4 LP3500 Function Calls

4.4.1 LP3500 Power Modes

```
int devPowerSet(int devices, int state);
```

Sets individual devices to low-power or fully active states in the order listed below.

PARAMETERS

devices is a list of the following macros, which are OR'ed together, that will be affected by the **state** parameter, e.g., **RS232DEV** | **ADCDEV**.

RS232DEV—RS-232 devices

RS485DEV—RS-485 devices

ADCDEV—ADS7870 A/D converter devices

DISPDEV—LCD/keypad module

ALLDEVICES—all devices

state

0 = shuts or powers down listed devices

1 = activates listed devices

Macro	Description `state` = 0	`state` after Board Initialization
RS232DEV	Receivers and transmitters are disabled, RxE remains active	1
RS485DEV	Transmitter is disabled	0
ADCDEV	ADS7870 internal oscillator is turned off	0
DISPDEV	LCD/keypad module is turned off.	0
ALLDEVICES	All devices are disabled as described above	—

Table E-1 provides further information about the power consumption associated with each section.

RETURN VALUE

0 if valid parameter

-1 otherwise

SEE ALSO

powerMode, anaInConfig, brdInit

`int powerMode(int mode);`

Sets the LP3500 operating power.

PARAMETERS

`mode` is the operating mode based on the following macros.

Mode	Description	Typical Current Consumption	Debug Capable?
1	CCLK = PCLK = MainOsc = 7.3728 MHz	5–16 mA	Yes
2	CCLK = PCLK = MainOsc/2 = 3.6864 MHz		
3	CCLK = PCLK = MainOsc/4 = 1.8423 MHz		
4	CCLK = PCLK = MainOsc/6 = 1.2288 MHz		No
5	CCLK = PCLK = MainOsc/8 = 0.9216 MHz		
6	CCLK = PCLK = 32.768 kHz	2 mA	
7	CCLK = PCLK = 32.768 kHz/2 = 16.384 kHz		
8	CCLK = PCLK = 32.768 kHz/4 = 8.192 kHz		
9	CCLK = PCLK = 32.768 kHz/8 = 4.096 kHz		
10	CCLK = PCLK = 32.768 kHz/16 = 2.048 kHz		

NOTE: When using modes 6–10, be sure to call **`hitwd()`** explicitly since periodic interrupts, which incorporate a virtual watchdog, are disabled in these modes.

Table 2 provides more specific information on the LP3500's capabilities associated with these and other software-defined modes.

RETURN VALUE

0 if valid parameter

-1 if invalid parameter

SEE ALSO

`devPowerSet, rdPowerState, setPowerState, VccMonitor`

void serCommAlert(int lowpowermode, int maxpowermode, int powersource, unsigned long timeout);

Use this function to poll Serial Port E for any activity or until a timeout occurs. The function call forces the LP3500 to enter the low-power mode using the battery for polling. Upon expiration of the timeout or the receipt of a serial byte, this function will enable the normal power mode and exit.

Use **devPowerSet(ALLDEVICES, int state)** before and after this function call to deactivate and activate all devices to operate at less power.

PARAMETERS

lowpowermode is the low-power mode setting to enter, usually 10 (see **powerMode()**)

maxpowermode is the maximum operating power mode setting to enter, usually 1 (see **powerMode()**)

powersource

1 = battery
2 = raw DC power

timeout is the timeout in seconds if no activity is detected on the RxE receiver line. Enter 0 for no timeout

SEE ALSO

powerMode, digInAlert, timedAlert, devPowerSet

void timedAlert(int lowpowermode, int maxpowermode, int powersource, unsigned long timeout)

Use this function to poll the real-time clock until a timeout occurs. The function call forces the LP3500 to enter the low-power mode, disables the normal power source, and may enable the external battery for polling. Upon expiration of the timeout this function will enable the normal power mode and exit. If the normal power source is not available, the LP3500 will not be able to resume operation at the maximum-power mode, and may reset.

Use **devPowerSet(ALLDEVICES, int state)** before and after this function call to deactivate and activate all devices to operate at less power.

PARAMETERS

lowpowermode is the low-power mode setting to enter, usually 10 (see **powerMode()**)

maxpowermode is the maximum operating power mode setting to enter, usually 1 (see **powerMode()**)

powersource

1 = battery
2 = raw DC power

timeout is the timeout in seconds if an input is not received.

SEE ALSO

powerMode, digInAlert, serCommAlert, devPowerSet

void digInAlert(int channel, int value, int lowpowermode, int maxpowermode, int powersource, unsigned long timeout)

Use this function to poll a selected digital input until a timeout occurs. The function call forces the LP3500 to enter the low-power mode using the battery for polling. Upon activation of the channel or expiration of the timeout, this function will enable the normal power mode and exit.

Use **devPowerSet(ALLDEVICES, int state)** before and after this function call to deactivate and activate all devices to operate at less power.

PARAMETERS

channel is the digital input channel (IN00– IN15) to poll

value is the input value of 0 or 1 to receive

lowpowermode is the low-power mode setting to enter, usually 10 (see **powerMode()**)

maxpowermode is the maximum operating power mode setting to enter, usually 1 (see **powerMode()**)

powersource
1 = battery
2 = raw DC power

timeout is the timeout in seconds if an input is not received. Enter 0 for no timeout.

SEE ALSO

powerMode, serCommAlert, timedAlert, devPowerSet

int rdPowerState(void);

Determines if the LP3500 is running under battery power or a raw DC power source.

RETURN VALUE

0 if on raw DC power source

1 if on battery power

SEE ALSO

powerMode, setPowerState

int setPowerSource(int state);

Turns the linear regulator “off” or “on.”

PARAMETER

0 for normal power source

1 for battery

RETURN VALUE

0 if successful

-1 if raw DC power source is not available

-2 if battery is not available

SEE ALSO

powerMode, rdPowerState

4.4.2 Board Initialization

```
void brdInit (void);
```

Call this function at the beginning of your program. This function initializes the system I/O ports and loads all the A/D converter and D/A converter calibration constants from flash memory into SRAM for use by your program. If the LCD/keypad module is installed, this function will turn off LED DS1 to indicate that the initialization was successful.

Summary of Initialization

- LP3500 uses main oscillator
- LCD/keypad module buffer is disabled
- RS-485 serial communication is ***not*** enabled
- RS-232 serial communication ***is*** enabled
- Unused configurable inputs are tied and unused configurable outputs are set low
- Self-timed chip select is set to 109 ns
- If A/D converter chip is installed, chip is reset and SCLKD is set to 19,200 bps
- If A/D converter chip is installed, calibration constants are read
- If relay is installed, relay is set to NC or RESET position

The ports are initialized according to Table A-3.

4.4.3 Digital I/O

void digOut(int channel, int value);

Sets the state of a digital output (OUT0–OUT9).

Remember to call **brdInit** before executing this function.

A runtime error will occur for the following conditions:

1. **channel** or **value** out of range.
2. **brdInit** was not executed before executing **digOut**.

PARAMETERS

channel is the output channel number (0–9).

value is the output value (0 or 1).

SEE ALSO

brdInit, digIn, digBankOut

void digBankOut(int bank, int value);

Writes the state of a block of designated digital output channels. The first bank consists of OUT0–OUT7, the second bank consists of OUT8–OUT9.

A run-time error will occur for the following conditions:

1. **channel** or **value** out of range.
2. **brdInit** was not executed before executing **digOut**.

PARAMETER

bank is 0 for OUT0–OUT7, 1 for OUT8–OUT9.

value is an 8-bit output value, where each bit corresponds to one channel. OUT0 and OUT8 are the least significant bit 0.

RETURN VALUE

None.

SEE ALSO

brdInit, digOut, digBankIn

int digIn(int channel);

Reads the state of an input channel (IN00–IN15).

A run-time error will occur for the following conditions:

1. **channel** out of range.
2. **brdInit** was not executed before executing **digIn**.

PARAMETER

channel is the input channel number (0–15)

RETURN VALUE

The logic state of the input (0 or 1).

SEE ALSO

brdInit, digOut, digBankIn

void digBankIn(int bank);

Reads the state of a block of designated digital input channels. The first bank consists of IN0–IN07, the second bank consists of IN08–IN15.

A run-time error will occur for the following conditions:

1. **bank** out of range.
2. **brdInit** was not executed before executing **digIn**.

PARAMETER

bank is 0 for IN00–IN07, 1 for IN08–IN15.

RETURN VALUE

An input value in the lower byte, where each bit corresponds to one channel. IN00 and IN08 are in the bit 0 place.

SEE ALSO

brdInit, digOut, digBankOut

4.4.4 Serial Communication

Library files included with Dynamic C provide a full range of serial communications support. The **LIB\Rabbit3000\RS232.LIB** library provides a set of circular-buffer-based serial functions. The **LIB\Rabbit3000\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. For more information, see the ***Dynamic C User's Manual*** and Rabbit's Technical Note TN213, ***Rabbit 2000 Serial Port Software***.

Use the following function calls with the LP3500.

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

User interface to set up LP3500 serial communication lines. Call this function after **serXOpen()**.

Whether you are opening one or multiple serial ports, this function must be executed after executing the last **serXOpen** function AND before you start using any of the serial ports. This function is non-reentrant.

If Mode 1 is selected, CTS/RTS flow control is exercised using the **serBflowcontrolOn** and **serBflowcontrolOff** functions from the **RS232.LIB** library.

PARAMETER

mode is the defined serial port configuration.

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

RETURN VALUE

0 if valid mode, 1 if not.

SEE ALSO

ser485Tx, ser485Rx

void ser485Tx(void);

Enables the RS-485 transmitter. Transmitted data get echo'ed back into the receive data buffer. These echo'ed data could be used to know when to disable the transmitter by using one of the following methods:

Byte mode—disable the transmitter after the same byte that is transmitted is detected in the receive data buffer.

Block data mode—disable the transmitter after the same number of bytes transmitted is detected in the receive data buffer.

`serMode()` must be executed before running this function.

SEE ALSO

`serMode, ser485Rx`

void ser485Rx(void);

Disables the RS-485 transmitter. This puts the LP3500 in listen mode, which allows it to receive data from the RS-485 interface. **`serMode()`** must be executed before running this function.

SEE ALSO

`serMode, ser485Tx`

NOTE: The RS-485 transceiver used on the LP3500 is only capable of supporting a maximum baud rate of 64,000 bits/s. The baud rate is set by the Dynamic C **`_485BAUD`** macro. For example, add the following line on the **Defines** tab in the Dynamic C **Options > Project Options** to set a baud rate of 57,600 bits/s, then click **OK**.

```
_485BAUD=57600
```

4.4.5 A/D Converter Inputs

The functions in this section apply only to the LP3500 model.

```
unsigned int anaInConfig(unsigned int
  instructionbyte, unsigned int cmd, long baud);
```

Use this function to configure the ADS7870 A/D converter. This function will address the ADS7870 in Register Mode only, and will report an error if you try to use it in Direct Mode. Refer to ADS7870 specification for proper addressing and commands.

ADS7870 Signal	ADS7870 State	LP3500 Function/State
LN0	Input	AIN0
LN1	Input	AIN1
LN2	Input	AIN2
LN3	Input	AIN3
LN4	Input	AIN4
LN5	Input	AIN5
LN6	Input	AIN6
LN7	Input	AIN7
/RESET	Input	Board reset device
RISE/FALL	Input	Tied up for SCLK active on rising edge
PIO_0	Input	Pulled down unless driven by serial interface connection
PIO_1	Input	Pulled down unless driven by serial interface connection
PIO_2	Input	Pulled down unless driven by serial interface connection
PIO_3	Input	Pulled up unless driven by Vcc monitor
CONVERT	Input	Pulled down, not used
BUSY	Output	PF1 pulled down; 1 state converter is busy
CCLKCNTRL	Input	Tied down; 0 state sets CCLK as input
CCLK	Input	Tied down; external conversion clock
SCLK	Input	PF0; serial data transfer clock
SDI	Input	PC0; 3-wire mode for serial data input
SDO	Output	PC1; serial data output /CS driven
/CS	Input	PF3 pulled up; active-low enables serial interface
BUFIN	Input	Tied down; reference buffer amplifier

PARAMETERS

instructionbyte 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 is the command data that configure the registers addressed by the instruction byte. Enter 0 if performing a read operation.

```
i = anaInConfig(0x07, 0x3a, 0);  //  write ref/osc reg and enable
```

baud is the serial clock transfer rate of 9600 to 57,600 bps. **baud** must be set on the first call to this function. Enter 0 in this parameter thereafter.

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

RETURN VALUE

0 on write operations, data value on read operations.

SEE ALSO

anaInDriver, anaIn, brdInit

```
unsigned int anaInDriver(unsigned int cmd,
   unsigned int len);
```

Reads the voltage of an analog input channel by serial-clocking an 8-bit command to the ADS7870 device by its Direct Mode method. The conversion begins as soon as the last data bit is transferred.

An exception error will occur if Direct Mode bit D7 is not set.

PARAMETER

`cmd` contains a gain code and a channel code as follows.

D7—1; D6–D4—Gain Code; D3–D0—Channel Code

Use the following calculation and the tables below to determine **`cmd`**:

```
cmd = 0x80 | (gain_code*16) + channel_code
```

Gain Code	Multiplier	Voltage Range
0	1	0–20 V
1	2	0–10 V
2	4	0–5 V
3	5	0–4 V
4	8	0–2.5 V
5	10	0–2 V
6	16	0–1.25 V
7	20	0–1 V

Channel Code	Differential Input Lines	Channel Code	Single-Ended Input Lines*	4–20 mA Lines
0	+AIN0 -AIN1	8	AIN0	AIN0
1	+AIN2 -AIN3	9	AIN1	AIN1
2	+AIN4 -AIN5	10	AIN2	AIN2
3	+AIN6 -AIN7	11	AIN3	AIN3
4	Reserved	12	AIN4	Reserved
5	Reserved	13	AIN5	Reserved
6	Reserved	14	AIN6	Reserved
7	Reserved	15	AIN7	Reserved

* Negative input is ground.

`len`, the output bit length, is always 12 bits.

RETURN VALUE

A value corresponding to the voltage on the analog input channel, which will be:

0–2047 for 11-bit A/D conversions (bit 12 for sign)

-1 for overflow

SEE ALSO

`anaInConfig, anaIn`

int anaIn(unsigned int channel, int opmode, int gaincode);

Reads the value of an analog input channel using the direct method of addressing the ADS7870 A/D converter.

PARAMETERS

channel is the analog input channel number (0 to 7) corresponding to AIN0–AIN7

opmode is the mode of operation:

SINGLE—single-ended input line
DIFF—differential input line
mAMP—milliamp input line

channel	SINGLE	DIFF	mAMP
0	+AIN0	+AIN0 -AIN1	+AIN0
1	+AIN1	—	+AIN1
2	+AIN2	+AIN2 -AIN3	+AIN2
3	+AIN3	—	+AIN3
4	+AIN4	+AIN4 -AIN5	—
5	+AIN5	—	—
6	+AIN6	+AIN6 -AIN7	—
7	+AIN7	—	—

gaincode is the gain code of 0 to 7:

Gain Code	Voltage Range
0	0–20 V
1	0–10 V
2	0–5 V
3	0–4 V
4	0–2.5 V
5	0–2 V
6	0–1.25 V
7	0–1 V

RETURN VALUE

A value corresponding to the voltage on the analog input channel, which will be:

0–2047 for 11-bit A/D conversions (signed 12th bit)

ADOVERFLOW (defined macro = -4096) if overflow or out of range

SEE ALSO

anaIn, anaInConfig, anaInDriver

```
int anaInCalib(int channel, int opmode,
   int gaincode, int value1, float volts1,
   int value2, float volts2);
```

Calibrates the response of the A/D converter channel as a linear function using the two conversion points provided. Four values are calculated and placed into global table **_adcCalib** to be stored later store into simulated EEPROM using the function **anaInEEWr()**. Each channel will have the following information:

a linear constant,

a voltage offset,

a calculation gain code used to calculate calibrations, and

a user gain code to set voltage range (defaults to the calculation gain code).

NOTE: Vcc monitoring is disabled when **anaInCalib** is running.

PARAMETERS

channel is the analog input channel number (0 to 7) corresponding to AIN0–AIN7

opmode is the mode of operation:

SINGLE—single-ended input line
DIFF—differential input line
mAMP—milliamp input line

channel	SINGLE	DIFF	mAMP
0	+AIN0	+AIN0 -AIN1	+AIN0
1	+AIN1	—	+AIN1
2	+AIN2	+AIN2 -AIN3	+AIN2
3	+AIN3	—	+AIN3
4	+AIN4	+AIN4 -AIN5	—
5	+AIN5	—	—
6	+AIN6	+AIN6 -AIN7	—
7	+AIN7	—	—

gaincode is the gain code of 0 to 7:

Gain Code	Voltage Range
0	0–20 V
1	0–10 V
2	0–5 V
3	0–4 V
4	0–2.5 V
5	0–2 V
6	0–1.25 V
7	0–1 V

value1 is the first A/D converter channel value (0–2047).

volts1 is the voltage or current corresponding to the first A/D converter channel value (0 to +10 V or 4 to 20 mA).

value2 is the second A/D converter channel value (0–2047).

volts2 is the voltage or current corresponding to the first A/D converter channel value (0 to +10 V or 4 to 20 mA).

RETURN VALUE

0 if successful.

-1 if not able to make calibration constants.

SEE ALSO

anaIn, anaInVolts, anaInmAmps, anaInDiff, anaInSetRange, anaInVoltXGain, anaInCalib, brdInit

`float anaInVolts(unsigned int channel, unsigned int gaincode);`

Reads the state of a single-ended analog input channel and uses the previously set calibration constants to convert it to volts.

PARAMETER

`channel` is the channel number (0–7):

Channel Code	Single-Ended Input Lines*
0	+AIN0
1	+AIN1
2	+AIN2
3	+AIN3
4	+AIN4
5	+AIN5
6	+AIN6
7	+AIN7

* Negative input is ground.

`gaincode` is the gain code of 0 to 7.

Gain Code	Voltage Range
0	0–20 V
1	0–10 V
2	0–5 V
3	0–4 V
4	0–2.5 V
5	0–2 V
6	0–1.25 V
7	0–1 V

RETURN VALUE

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

ADOVERFLOW (defined macro = -4096) if overflow or out of range.

SEE ALSO

`anaInCalib, anaIn, anaInmAmps, brdInit`

`float anaInmAmps(unsigned int channel);`

Reads the state of an analog input channel and uses the previously set calibration constants to convert it to current.

PARAMETER

`channel` is 0–3:

Channel	4–20 mA Input Lines*
0	AIN0
1	AIN1
2	AIN2
3	AIN3

* Negative input is ground.

RETURN VALUE

A current value between 4.00 and 20.00 mA corresponding to the current on the analog input channel.

`ADOVERFLOW` (defined macro = -4096) if overflow or out of range.

SEE ALSO

`anaInCalib, anaIn, anaInVolts`

`float anaInDiff(unsigned int channel, unsigned int gaincode);`

Reads the state of a differential analog input channel and uses the previously set calibration constants to convert it to volts.

PARAMETER

`channel` is the channel number (0, 2, 4, 6):

Channel	Differential Input Lines
0	+AIN0 -AIN1
2	+AIN2 -AIN3
4	+AIN4 -AIN5
6	+AIN6 -AIN7

`gaincode` is the gain code of 0 to 7.

Gain Code	Voltage Range
0	0–20 V
1	0–10 V
2	0–5 V
3	0–4 V
4	0–2.5 V
5	0–2 V
6	0–1.25 V
7	0–1 V

RETURN VALUE

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

ADOVERFLOW (defined macro = -4096) if overflow or out of range.

SEE ALSO

`anaInCalib, anaIn, anaInmAmps, brdInit`

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

Reads the calibration constants, gain, and offset for an input based on its designated channel code position into global table **_adcCalib**. The constants are stored in the top 1K of the reserved user block memory area 0x1C00–0x1FFF.

NOTE: This function cannot be run in RAM.

PARAMETER

channel is the analog input channel number (0 to 7) corresponding to AIN0–AIN7.

opmode is the mode of operation:

SINGLE—single-ended input line
DIFF—differential input line
mAMP—milliamp input line

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

gaincode is the gain code of 0 to 7. The **gaincode** parameter is ignored when **channel** is **ALLCHAN**.

Gain Code	Voltage Range
0	0–20 V
1	0–10 V
2	0–5 V
3	0–4 V
4	0–2.5 V
5	0–2 V
6	0–1.25 V
7	0–1 V

RETURN VALUE

0 if successful.

-1 if address is invalid or out of range.

SEE ALSO

anaInEEWr, anaInCalib

int anaInEEWr(unsigned int channel, int opmode unsigned int gaincode);

Writes the calibration constants, gain, and offset for an input based on its designated channel code position from global table **_adcCalib**. The constants are stored in the top 1K of the reserved user block memory area 0x1C00–0x1FFF.

NOTE: This function cannot be run in RAM.

PARAMETER

channel is the analog input channel number (0 to 7) corresponding to AIN0–AIN7.

opmode is the mode of operation:

SINGLE—single-ended input line
DIFF—differential input line
mAMP—milliamp input line

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

gaincode is the gain code of 0 to 7. The **gaincode** parameter is ignored when **channel** is **ALLCHAN**.

Gain Code	Voltage Range
0	0–20 V
1	0–10 V
2	0–5 V
3	0–4 V
4	0–2.5 V
5	0–2 V
6	0–1.25 V
7	0–1 V

RETURN VALUE

0 if successful.

-1 if address is invalid or out of range.

SEE ALSO

anaInEEWr, anaInCalib

4.4.6 Vcc Monitoring (LP3500 only)

void VccMonitorInit(int state);

PIO3 on the ADS7870 A/D converter enables or disables Vcc monitoring. If monitoring is enabled, analog input channel AIN7 is not available.

PARAMETER

state

1 = enable Vcc monitor

0 = disable Vcc monitor

SEE ALSO

VccMonitor, anaInConfig, brdInit

float VccMonitor(void);

If Vcc monitoring is enabled, the Vcc level is read by the ADS7870 A/D converter and is converted to a voltage value.

RETURN VALUE

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

SEE ALSO

VccMonitorInit, anaInVolts, brdInit

4.4.7 PWM Outputs

The PWM functions in this section can be used to operate the analog outputs on the LP3500 model.

```
int pwmOutConfig(unsigned long frequency);
```

Sets the base frequency for the PWM pulses and enables the PWM driver on all four channels. The base frequency is the frequency without pulse spreading. Pulse spreading (see **`pwm_set`**) will increase the frequency by a factor of 4.

PARAMETERS

`frequency` is the frequency (in Hz).

RETURN VALUE

Actual frequency set. This will be the closest possible match to the requested frequency.

SEE ALSO

`pwmOut`

```
int pwmOut(unsigned int channel, float dutycycle);
```

Sets a voltage (0 to VDD on an analog output channel according to the percent duty cycle of the 1024-clock-count cycle.)

PARAMETERS

`channel` is the output channel to write to (0–3).

`dutycycle` is the percent duty (or on) cycle value of the 1024-clock-count cycle (i.e., 0.25).

RETURN VALUE

0 if successful

-1 if an invalid channel number is used

-2 if an invalid duty cycle was requested

SEE ALSO

`pwmOutConfig`

4.5 Relay Output (LP3500 only)

`int relayOut(int relay, int value)`

A 10 ms low-to-high pulse sets the state of a relay. On power-up or **`brdInit()`** the relay contact will go to the normally closed (NC) RESET contact.

PARAMETERS

`relay`

0 = the one relay

`value` is a value used to connect the relay common contact:

0 = relay normally closed (NC or RESET) (Parallel Port PG5)

1 = relay normally open (NO or SET) (Parallel Port PG4)

RETURN VALUE

0 if successful

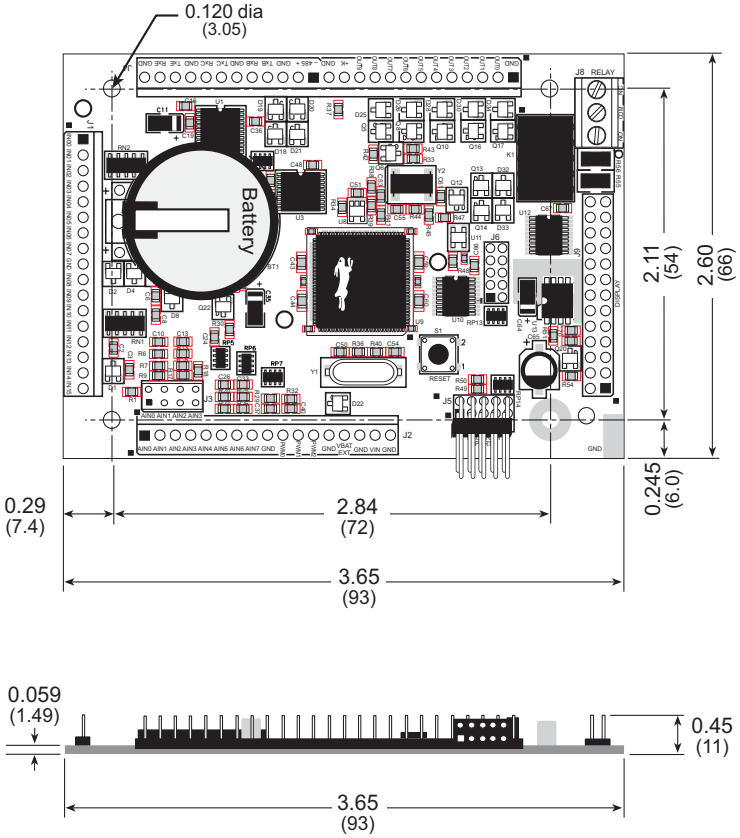
-1 if the normal power source is not available

APPENDIX A. LP3500 SPECIFICATIONS

Appendix A provides the specifications for the LP3500, and describes the conformal coating.

A.1 Electrical and Mechanical Characteristics

Figure A-1 shows the mechanical dimensions for the LP3500.



Please refer to the LP3500 footprint diagram later in this appendix for precise header locations.

Figure A-1. LP3500 Dimensions

Table A-1 lists the electrical, mechanical, and environmental specifications for the LP3500.

Table A-1. LP3500 Specifications

Feature	LP3500	LP3510
Microprocessor	Rabbit 3000® at up to 7.4 MHz	
EMI Reduction	Spectrum spreader for ultra-low EMI (radiated emissions)	
Flash Memory	512K (2 × 256K)	256K
SRAM	512K	128K
Backup Battery	Socketed 3 V lithium coin Panasonic CR2330, 265 mA·h, supports RTC and SRAM, connection for user-supplied external battery	
Keypad/Display	Supports optional LCD/keypad module with 7 keys and 122 × 32 graphic display	
Digital Inputs	16: fully protected 0–36 V DC, can handle short spikes ±40 V	
Digital Outputs	10: 8 sink up to 200 mA each, 36 V DC max.; 2 source up to 200 mA each, 36 V DC max.	
Relay Output	1 C-form, 1 A, 30 V DC	None
Analog Inputs	• Eight single-ended or four differential inputs • 1 MΩ input impedance • Sampling rate up to 200 samples/s • Eight software-controlled ranges from 0–1 V to 0–20 V DC **Single-Ended Inputs** • Resolution: 11 bits • Accuracy: 8 bits • 4 channels can be set individually for 4–20 mA with plug-in jumpers • 1 channel has software-selectable voltage-monitoring option **Differential Inputs** • Resolution: 12 bits • Accuracy: 9 bits	None
Analog Outputs	3 unfiltered pulse-width modulated, 1 kΩ output impedance	None

Table A-1. LP3500 Specifications (continued)

Feature	LP3500	LP3510
Serial Ports	6 shared high-speed, CMOS-compatible ports: • 1 RS-485 • 3 RS-232 (one 5-wire and one 3-wire or three 3-wire), jumper option for logic-level outputs; one RS-232 port needs to have wake-up capability • 1 logic-level serial interface for optional add-ons • 1 asynchronous clocked serial port dedicated for programming	
Serial Rate	Max. asynchronous baud rate = CLK/8	
Real-Time Clock	Yes	
Timers	Ten 8-bit timers (6 cascadable from the first), one 10-bit timer with 2 match registers	
Watchdog/Supervisor	Yes	
Pulse-Width Modulators	10-bit free-running counter and four pulse-width registers	
Power	3 V to 30 V DC 20 mA (max.) @ 7.4 MHz, 100 µA max. @ 2 kHz (with linear regulator turned off)	
Operating Temperature	–40°C to +70°C	
Humidity	5% to 95%, noncondensing	
Connectors	• 0.1" headers I/O and misc. signals: one 1 × 25, two 1 × 17 headers Display: one 2 × 13 header • 2 mm headers Programming Port: one 2 × 5 header Serial Interface: one 2 × 4 socket • Screw-terminal headers Relay: one 3-position screw-terminal header	
Board Size	2.60" × 3.65" × 0.45" (66 mm × 93 mm × 11 mm)	

A.1.1 Exclusion Zone

It is recommended that you allow for an “exclusion zone” of 0.25" (6 mm) around the LP3500 in all directions when the LP3500 is incorporated into an assembly that includes other printed circuit boards. This “exclusion zone” that you keep free of other components and boards will allow for sufficient air flow, and will help to minimize any electrical or electromagnetic interference between adjacent boards. An “exclusion zone” of 0.12" (3 mm) is recommended below the LP3500. Figure A-2 shows this “exclusion zone.”

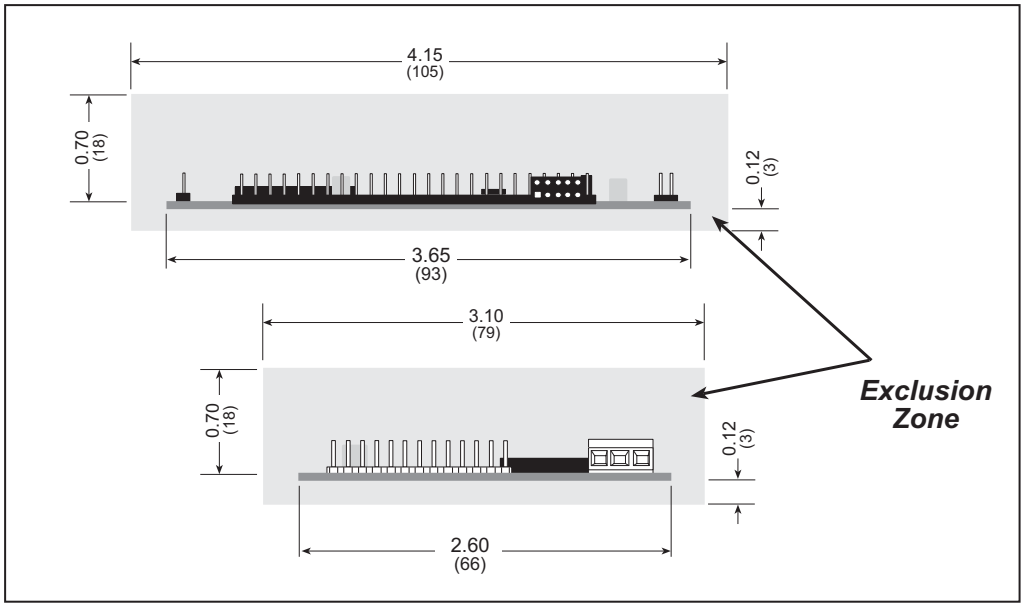


Figure A-2. LP3500 “Exclusion Zone”

When using the LP3500 with the Prototyping Board, do not install any components in the prototyping area on the Prototyping Board between the LP3500 and the Prototyping Board.

A.1.2 Headers

The LP3500 uses 0.1" IDC headers at J1–J4 for physical connection to other boards. J5, the programming port, is a 2 × 5 header with a 2 mm pin spacing.

Figure A-3 shows the LP3500 footprint. These values are relative to the mounting hole.

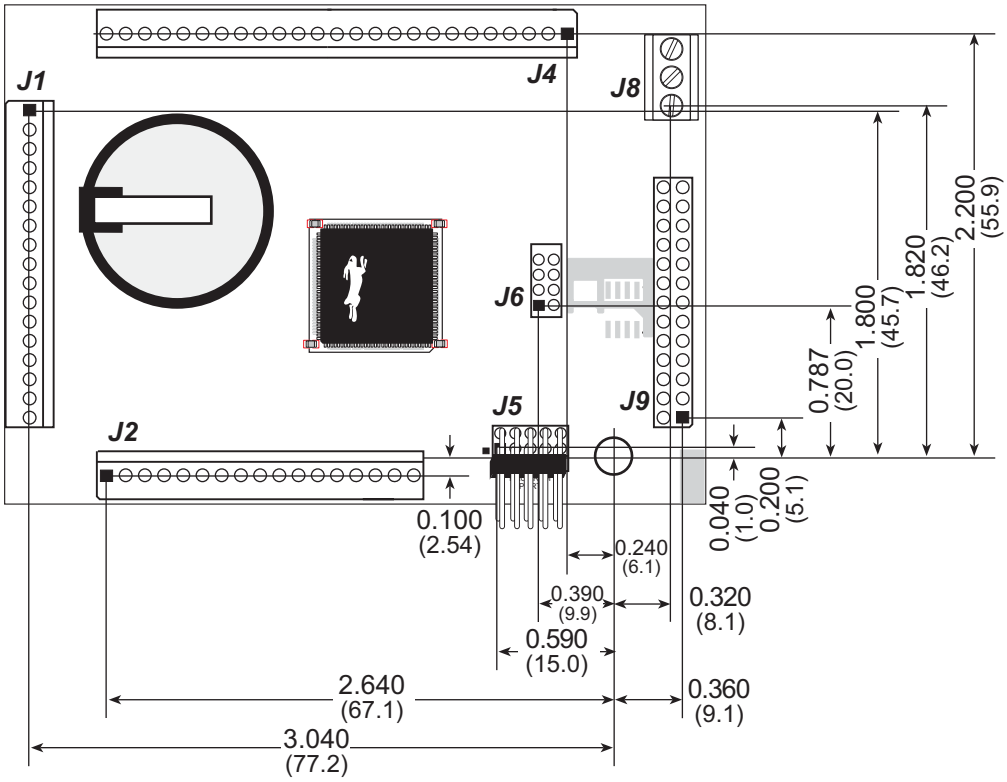


Figure A-3. User Board Footprint for LP3500

A.2 Conformal Coating

The areas around the 32 kHz real-time clock crystal oscillator has had the Dow Corning silicone-based 1-2620 conformal coating applied. The conformally coated area is shown in Figure A-4. The conformal coating protects these high-impedance circuits from the effects of moisture and contaminants over time.

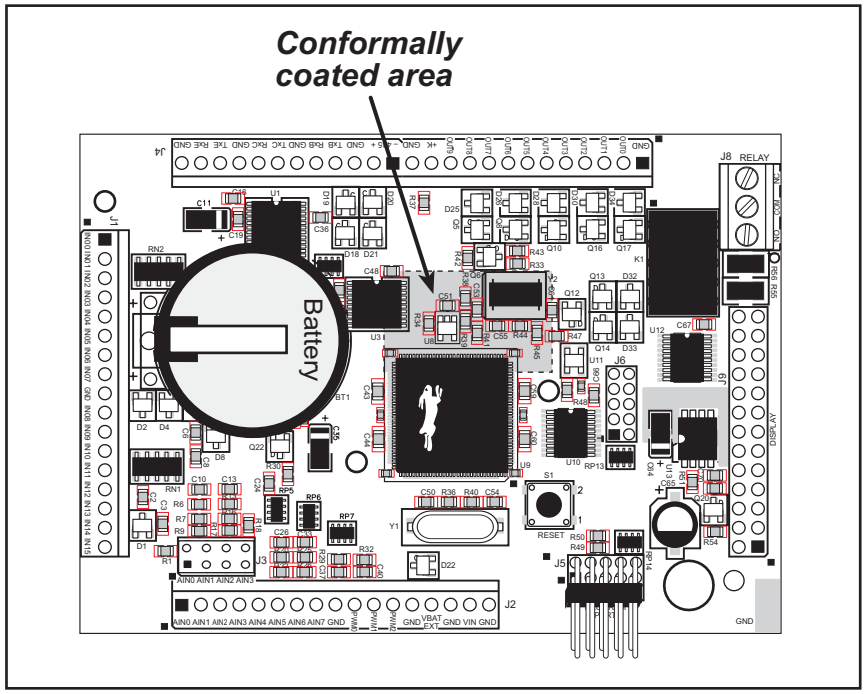


Figure A-4. LP3500 Areas Receiving Conformal Coating

Any components in the conformally coated area may be replaced using standard soldering procedures for surface-mounted components. A new conformal coating should then be applied to offer continuing protection against the effects of moisture and contaminants.

> **NOTE:** For more information on conformal coatings, refer to Technical Note 303, ***Conformal Coatings***.

A.3 Jumper Configurations

Figure A-5 shows the header locations used to configure the various LP3500 options via jumpers.

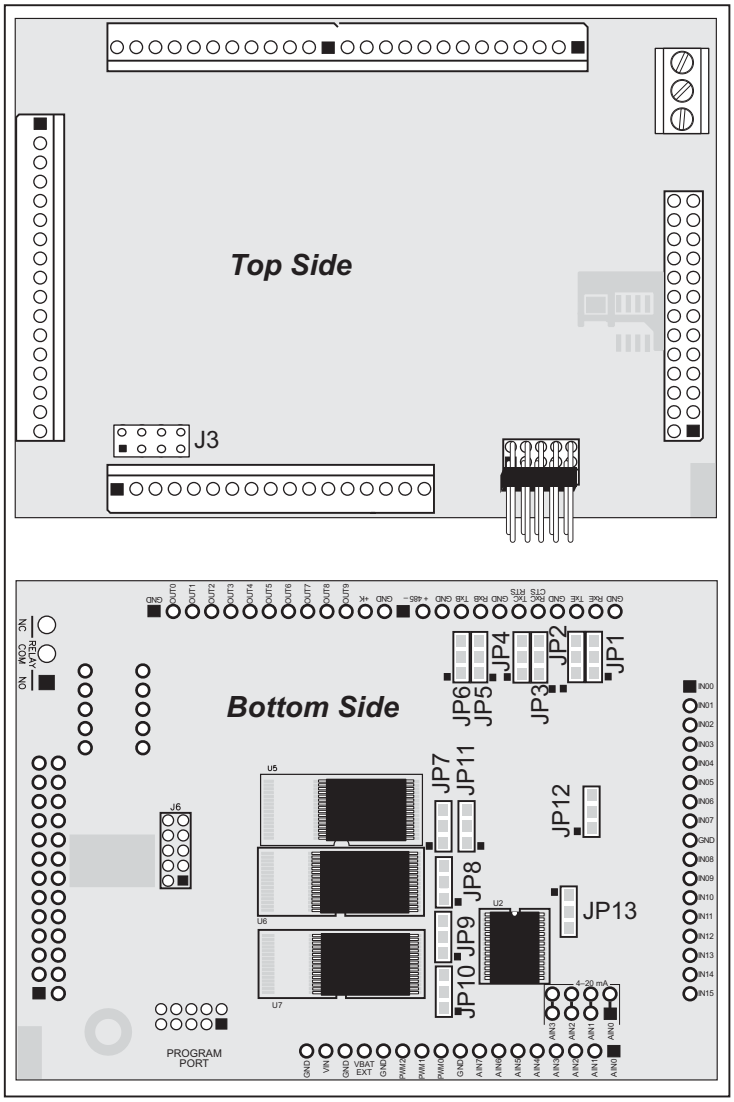


Figure A-5. Location of LP3500 Configurable Positions

Table A-2 lists the configuration options.

Table A-2. LP3500 Jumper Configurations

Header	Description	Pins Connected		Factory Default
J3	A/D Converter Voltage/Current Measurement Options	None	Voltage Option	✕
		1–2	AIN0 4–20 mA Option	
		3–4	AIN1 4–20 mA Option	
		5–6	AIN2 4–20 mA Option	
		7–8	AIN3 4–20 mA Option	
JP1	RxE RS-232/Logic Level Select	1–2	RS-232 Level	✕
		2–3	Logic Level	
JP2	TxE RS-232/Logic Level Select	1–2	RS-232 Level	✕
		2–3	Logic Level	
JP3	RxC RS-232/Logic Level Select	1–2	RS-232 Level	✕
		2–3	Logic Level	
JP4	TxC RS-232/Logic Level Select	1–2	RS-232 Level	✕
		2–3	Logic Level	
JP5	RxB RS-232/Logic Level Select	1–2	RS-232 Level	✕
		2–3	Logic Level	
JP6	TxB RS-232/Logic Level Select	1–2	RS-232 Level	✕
		2–3	Logic Level	
JP7	SRAM Size	1–2	128K	LP3510
		2–3	512K	LP3500
JP8	Flash Memory Size	1–2	128K/256K	✕
		2–3	512K	
JP9	Flash Memory Size	1–2	128K/256K	LP3500
		2–3	512K	
JP10	Flash Memory Bank Select	1–2	Normal Mode	✕
		2–3	Bank Mode	

Table A-2. LP3500 Jumper Configurations (continued)

Header	Description	Pins Connected		Factory Default
JP11	Manufacturing Use	1–2		×
JP12	Manufacturing Use	1–2		×
JP13	Manufacturing Use	1–2		×

NOTE: The jumper connections on header J3 are made using standard slip-on jumpers. All other jumper connections except those across JP11 and JP12 are made using 0 Ω surface-mounted resistors. 390 Ω current-limiting resistors are used on JP11 and JP12.

A.4 Use of Rabbit 3000 Parallel Ports

Figure A-6 shows the Rabbit 3000 parallel ports.

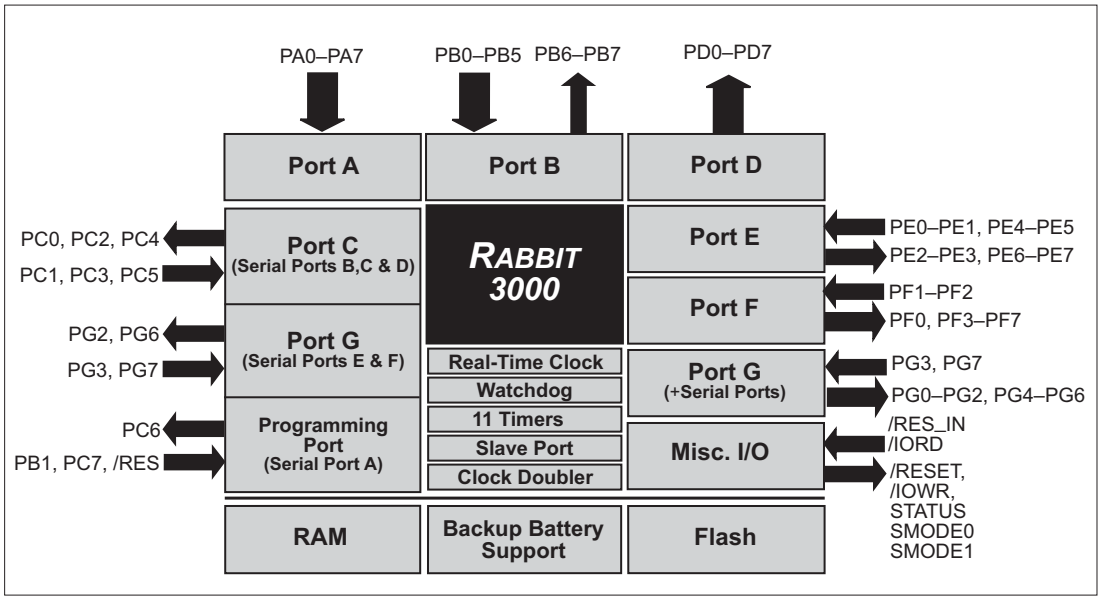


Figure A-6. LP3500 Rabbit-Based Subsystems

Table A-3 lists the Rabbit 3000 parallel ports and their use in the LP3500.

Table A-3. Use of Rabbit 3000 Parallel Ports

Port	I/O	Signal	Output Function State
PA0	Input	IN08	Pulled up
PA1	Input	IN09	Pulled up
PA2	Input	IN10	Pulled up
PA3	Input	IN11	Pulled up
PA4	Input	IN12	Pulled up
PA5	Input	IN12	Pulled up
PA6	Input	IN14	Pulled up
PA7	Input	IN15	Pulled up
PB0	Input	Power Input Detect	Low when external power source is connected; high for battery
PB1	Input	CLKA	Pulled up when not driven by programming port
PB2	Input	IN04	Pulled up

Table A-3. Use of Rabbit 3000 Parallel Ports (continued)

Port	I/O	Signal		Output Function State
PB3	Input	IN05		Pulled up
PB4	Input	IN06		Pulled up
PB5	Input	IN07		Pulled up
PB6	Output	LCD Buffer Enable		Inactive high
PB7	Output	Serial Device Select		Inactive high
PC0	Output	TXD Serial Device Int.	Serial Port D	Inactive high
PC1	Input	RXD Serial Device Int.		Inactive high
PC2	Output	RTS/TxC RS-232	Serial Port C	Inactive high
PC3	Input	CTS/RxC RS-232		Inactive high
PC4	Output	TxB RS-232	Serial Port B	Inactive high
PC5	Input	RxB RS-232		Inactive high
PC6	Output	TxA Programming Port	Serial Port A	Inactive high
PC7	Input	RxA Programming Port		Inactive high
PD0	Output	OUT0		Inactive low
PD1	Output	OUT1		Inactive low
PD2	Output	OUT2		Inactive low
PD3	Output	OUT3		Inactive low
PD4	Output	OUT4		Inactive low
PD5	Output	OUT5		Inactive low
PD6	Output	OUT6		Inactive low
PD7	Output	OUT7		Inactive low
PE0	Input	IN00		Pulled up
PE1	Input	IN01		Pulled up
PE2	Output	OUT8		Inactive low
PE3	Output	OUT9		Inactive low
PE4	Input	IN02		Pulled up
PE5	Input	IN03		Pulled up
PE6	Output	LCD/Keypad Module Reset Line		Inactive high
PE7	Output	LCD/Keypad Module Device Select		Inactive high

Table A-3. Use of Rabbit 3000 Parallel Ports (continued)

Port	I/O	Signal		Output Function State
PF0	Output	ADC Serial Clock		Inactive high
PF1	Input	ADC Busy		Inactive low
PF2	Input	Not used		Pulled up
PF3	Output	ADC Device Select		Inactive high
PF4	Output	PWM0		Inactive high
PF5	Output	PWM1		Inactive high
PF6	Output	PWM2		Inactive high
PF7	Output	Power Enable Control		Low when external power source is connected; high for battery
PG0	Output	RS-485 Transmit Enable		Inactive low
PG1	Output	RS-232 Shutdown Control		Inactive high
PG2	Output	TxF RS-485	Serial Port F	Inactive high
PG3	Input	RxF RS-485		Inactive high
PG4	Output	Relay Set		Inactive low
PG5	Output	Relay Reset		Inactive low
PG6	Output	TxE RS-232	Serial Port E	Inactive high
PG7	Input	RxE RS-232		Inactive high

APPENDIX B. PROTOTYPING BOARD

Appendix B describes the features and accessories of the Prototyping Board included with the LP3500 Tool Kit, and explains the use of the Prototyping Board to demonstrate the LP3500 and to build prototypes of your own circuits.

The screw-terminal headers on the Prototyping Board facilitate access to the LP3500 connector pins, and the Prototyping Board is available for purchase separately.

B.1 Mechanical Dimensions and Layout

Figure B-1 shows the mechanical dimensions and layout for the LP3500 Prototyping Board.

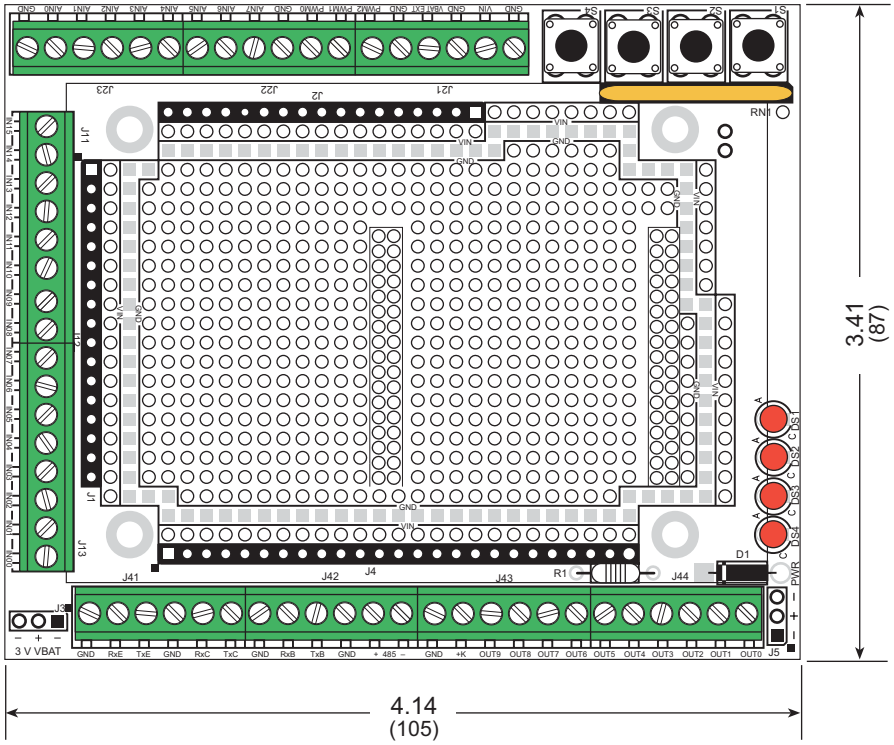


Figure B-1. LP3500 Prototyping Board Dimensions

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

Table B-1. LP3500 Prototyping Board Specifications

Parameter	Specification
Board Size	3.41" × 4.14" × 0.45" (87 mm × 105 mm × 11 mm)
Operating Temperature	–40°C to +70°C
Humidity	5% to 95%, noncondensing
Prototyping Area	2.2" × 3.4" (56 mm × 86 mm) throughhole, 0.1" spacing

B.2 Using the Prototyping Board

B.2.1 Interface to LP3500

The Prototyping Board serves as a convenient interface for the LP3500, extending the IDC headers to convenient screw-terminal connectors, and provides interfaces to the AC adapter included with the Tool Kit and to a user-supplied external battery.

Figure B-2 shows the pinouts for the Prototyping Board.

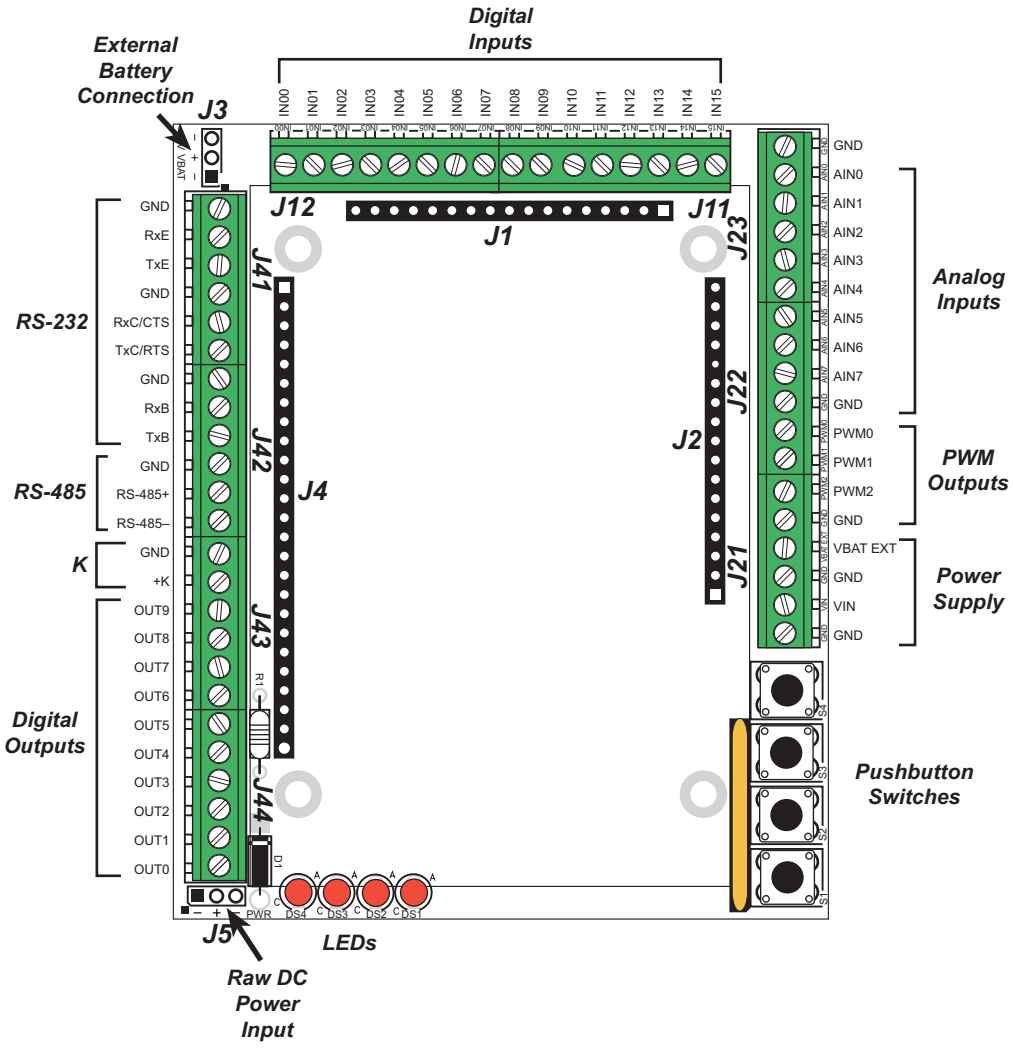


Figure B-2. LP3500 Prototyping Board Pinout

NOTE: The LP3500 must be plugged in to the Prototyping Board as described in Chapter 2, “Getting Started,” for these signals to be available.

B.2.2 Demonstration Board

The Prototyping Board is actually both a demonstration board and a prototyping board. As a demonstration board, it can be used to demonstrate the functionality of the LP3500 right out of the box without any modifications to either board. There are no jumpers or dip switches to configure or misconfigure on the Prototyping Board so that the initial setup is very straightforward.

The Prototyping Board comes with the basic components necessary to demonstrate the operation of the LP3500. Four LEDs (DS1–DS4) are connected to PD0–PD3, and four switches (S1–S4) are connected to PE0, PE1, PE4, and PE5 to demonstrate the interface to the Rabbit 3000 microprocessor.

> **NOTE:** Before running sample programs based on the LP3500, you will have to plug in the LP3500 to the Prototyping Board as described in Chapter 2, "Getting Started."

B.2.3 Prototyping Area

Small to medium circuits can be prototyped using point-to-point wiring with 20 to 30 AWG wire on the prototyping area. Raw DC input, VIN, and GND lines surround the prototyping area.

The resistor pack located next to the pushbutton switches may be removed to disconnect the LEDs and pushbutton switches from the Prototyping Board circuits, giving your LP3500 exclusive access to what you may develop in the prototyping area.

+K and VIN are tied together by resistor R1 located beside header J44. Cut off R1 if you intend to supply a separate +K.

APPENDIX C. LCD/KEYPAD MODULE

An optional LCD/keypad is available for the LP3500. Appendix C describes the LCD/keypad module and provides the software calls to make full use of the LCD/keypad module.

C.1 Specifications

Two optional LCD/keypad modules—with or without a panel-mounted NEMA 4 water-resistant bezel—are available for use with the LP3500. They are shown in Figure C-1.

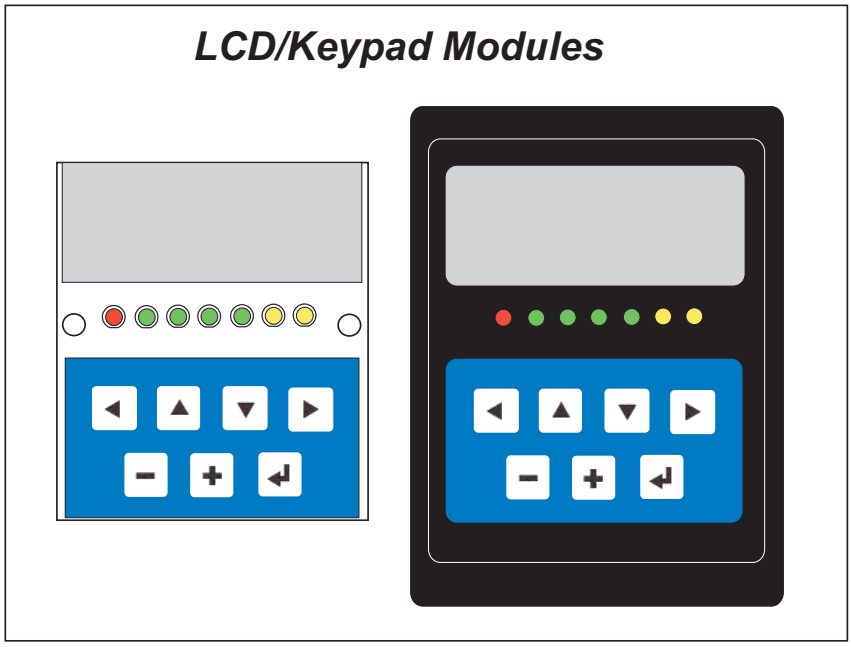


Figure C-1. LCD/Keypad Module Models

LCD/keypad modules sold prior to the launch of the LP3500 might not be voltage-compatible with the LP3500. Contact your Rabbit sales representative or your authorized distributor for further assistance in purchasing an LCD/keypad module.

Mounting hardware and a 127 mm (5") or 60 cm (24") extension cable are also available for the LCD/keypad module through your Rabbit sales representative or authorized distributor.

Table C-1 lists the electrical, mechanical, and environmental specifications for the LCD/keypad module.

Table C-1. LCD/Keypad Specifications

Parameter	Specification
Board Size	2.60" × 3.00" × 0.75" (66 mm × 76 mm × 19 mm)
Bezel Size	4.50" × 3.60" × 0.30" (114 mm × 91 mm × 7.6 mm)
Temperature	Operating Range: 0°C to +50°C Storage Range: –40°C to +85°C
Humidity	5% to 95%, noncondensing
Power Consumption	1.5 W maximum with backlight on*
Connections	Connects to header J9 (Display Interface) on LP3500
LCD Panel Size	122 × 32 graphic display
Keypad	7-key keypad
LEDs	Seven user-programmable LEDs

* The backlight adds approximately 650 mW to the power consumption.

The LCD/keypad module has 0.1" IDC header sockets at J1, J2, and J3 for physical connection to other boards or ribbon cables. Figure C-2 shows the LCD/keypad module footprint. These values are relative to one of the mounting holes.

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

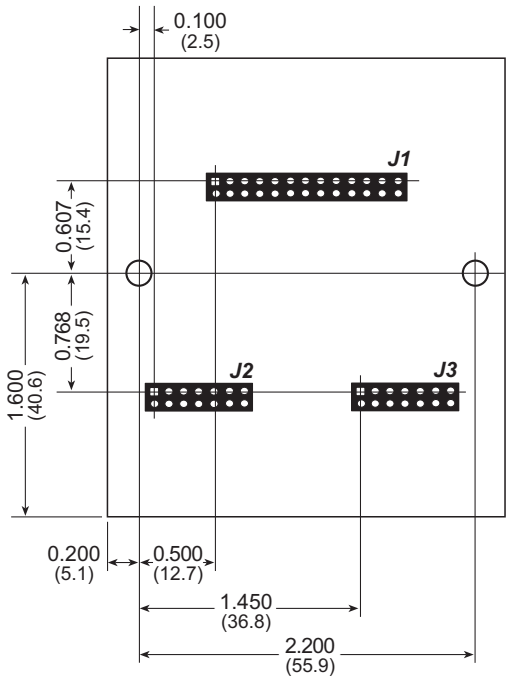


Figure C-2. User Board Footprint for LCD/Keypad Module

C.2 Contrast Adjustment

Starting in 2005, LCD/keypad modules were factory-configured to optimize their contrast based on the voltage of the system they would be used in. Be sure to select a KDU3V LCD/keypad module for use with the LP3500 — these modules operate at 3.3 V. You may adjust the contrast using the potentiometer at R2 as shown in Figure C-3. KDU5V LCD/keypad modules configured for 5 V may be used with the LP3500, but the backlight will be dim.

LCD/Keypad Module Jumper Configurations

Header	Description	Pins Connected	Factory Default
J5	2.8 V	1–2	×
	3.3 V	3–4	
	5 V	n.c.	

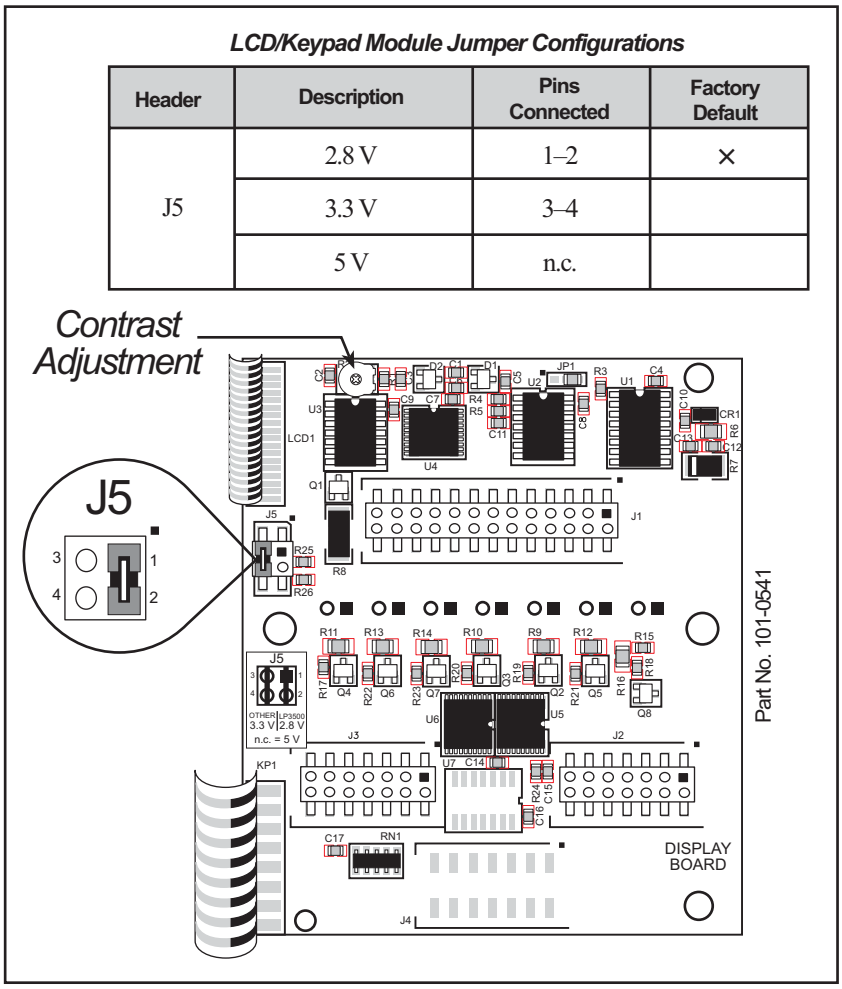


Figure C-3. LCD/Keypad Module Contrast Adjustment

You can set the contrast on the LCD display of pre-2005 LCD/keypad modules by adjusting the potentiometer at R2 or by setting the voltage for 2.8 V by connecting the jumper across pins 1–2 on header J5 as shown in Figure C-3. Only one of these two options is available on these older LCD/keypad modules.

NOTE: Older LCD/keypad modules that do not have a header at J5 or a contrast adjustment potentiometer at R2 are limited to operate only at 5 V, and will not work with the LP3500. The older LCD/keypad modules are no longer being sold.

C.3 Keypad Labeling

The keypad may be labeled according to your needs. A template is provided in Figure C-4 to allow you to design your own keypad label insert.

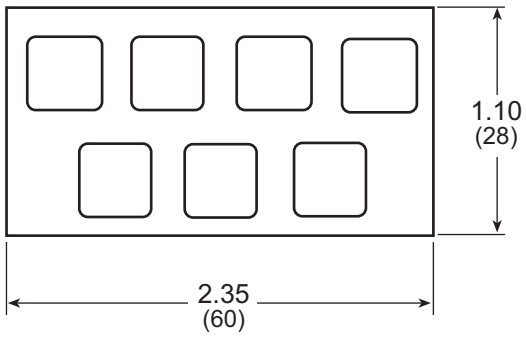


Figure C-4. Keypad Template

To replace the keypad legend, remove the old legend and insert your new legend prepared according to the template in Figure C-4. The keypad legend is located under the blue keypad matte, and is accessible from the left only as shown in Figure C-5.

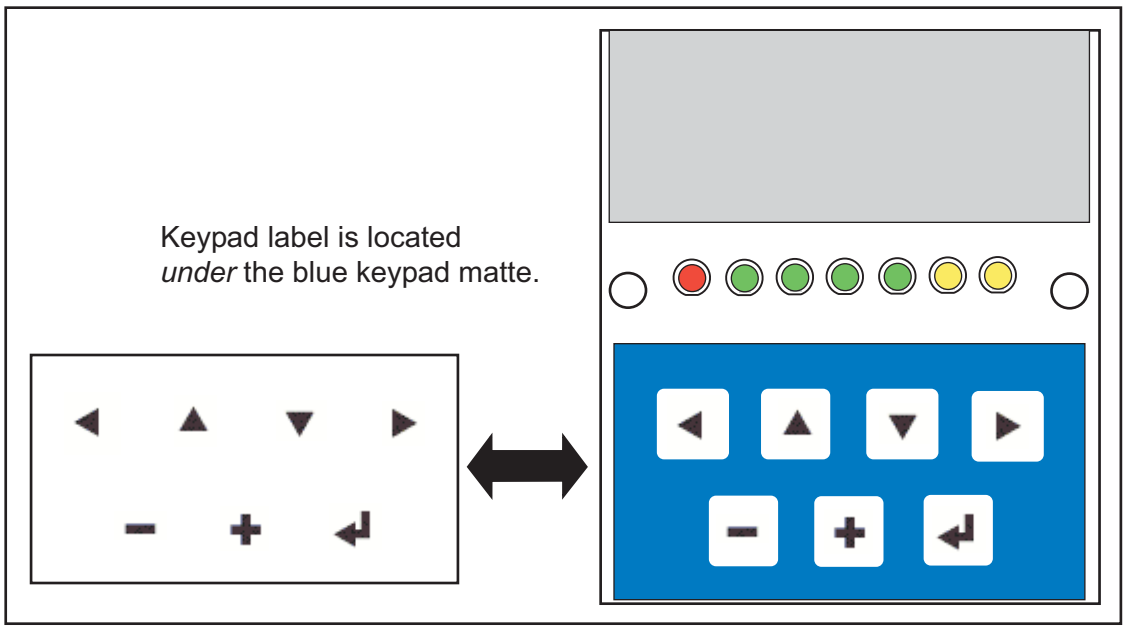


Figure C-5. Removing and Inserting Keypad Label

The sample program **`KEYBASIC.C`** in the **`122x32_1x7`** folder in **`SAMPLES\LCD_KEYPAD`** shows how to reconfigure the keypad for different applications.

C.4 Header Pinouts

Figure C-6 shows the pinouts for the LCD/keypad module.

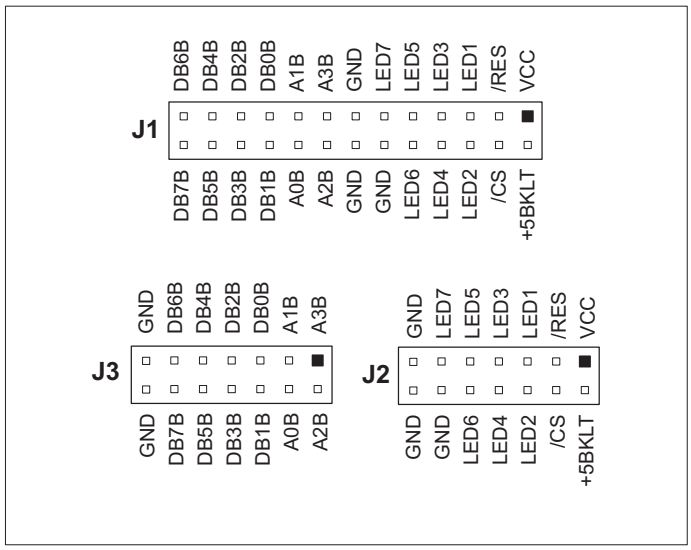


Figure C-6. LCD/Keypad Module Pinouts

NOTE: Note that there are no connections from headers J2 and J3 of the LCD/keypad module to the LP3500. These headers interface to the keypad and to the LEDs on the LCD/keypad module, and need to be interfaced to the digital I/O on the LP3500 if you need keypad or LED functionality. The LEDs may be driven by an active signal either in software or in hardware.

C.4.1 I/O Address Assignments

The LCD and keypad on the LCD/keypad module are addressed by the /CS strobe as explained in Table C-2.

Table C-2. LCD/Keypad Module Address Assignment

Address	Function
0xE000	Device select base address (/CS)
0xExx0–0xExx7	LCD control
0xExx8	LED enable
0xExx9	Not used
0xExxA	7-key keypad
0xExxB (bits 0–6)	7-LED driver
0xExxB (bit 7)	LCD backlight on/off
0xExxC–ExxF	Not used

C.5 Bezel-Mount Installation

This section describes and illustrates how to bezel-mount the LCD/keypad module. Follow these steps for bezel-mount installation.

1. Cut mounting holes in the mounting panel in accordance with the recommended dimensions in Figure C-7, then use the bezel faceplate to mount the LCD/keypad module onto the panel.

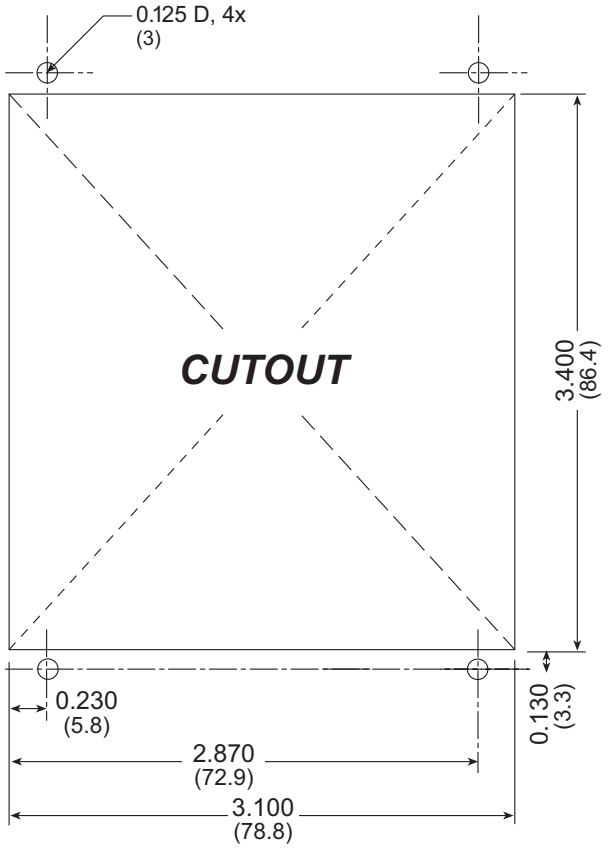


Figure C-7. Recommended Cutout Dimensions

2. Carefully “drop in” the LCD/keypad module with the bezel and gasket attached.

3. Fasten the unit with the four 4-40 screws and washers included with the LCD/keypad module. If your panel is thick, use a 4-40 screw that is approximately 3/16" (5 mm) longer than the thickness of the panel.

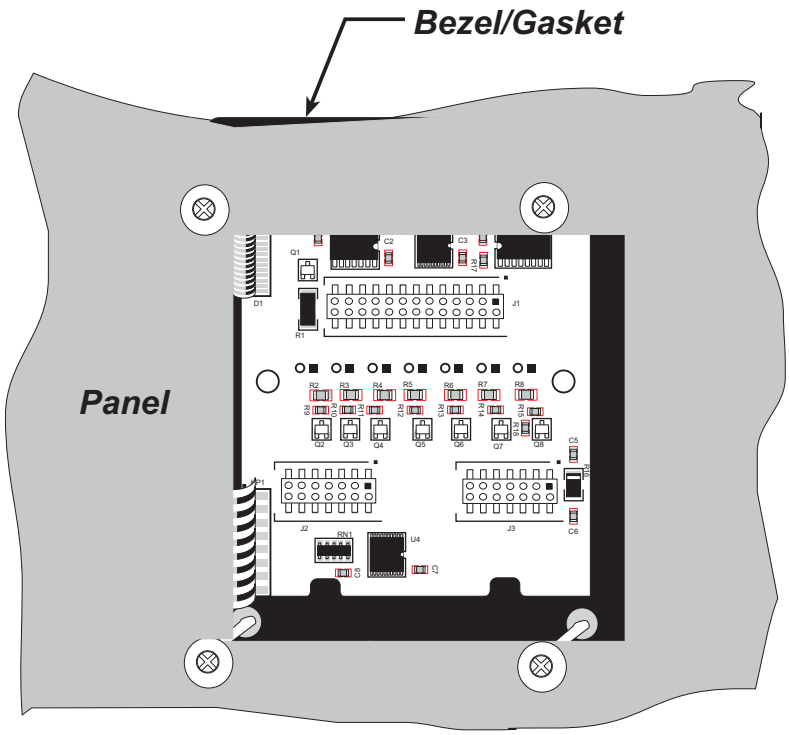


Figure C-8. LCD/Keypad Module Mounted in Panel (rear view)

Carefully tighten the screws until the gasket is compressed and the plastic bezel face-plate is touching the panel.

Do not tighten each screw fully before moving on to the next screw. Apply only one or two turns to each screw in sequence until all are tightened manually as far as they can be so that the gasket is compressed and the plastic bezel faceplate is touching the panel.

C.6 Connect the LCD/Keypad Module to Your LP3500

The LCD/keypad module can be located as far as 2 ft. (60 cm) away from the LP3500, and is connected via a ribbon cable as shown in Figure C-9.

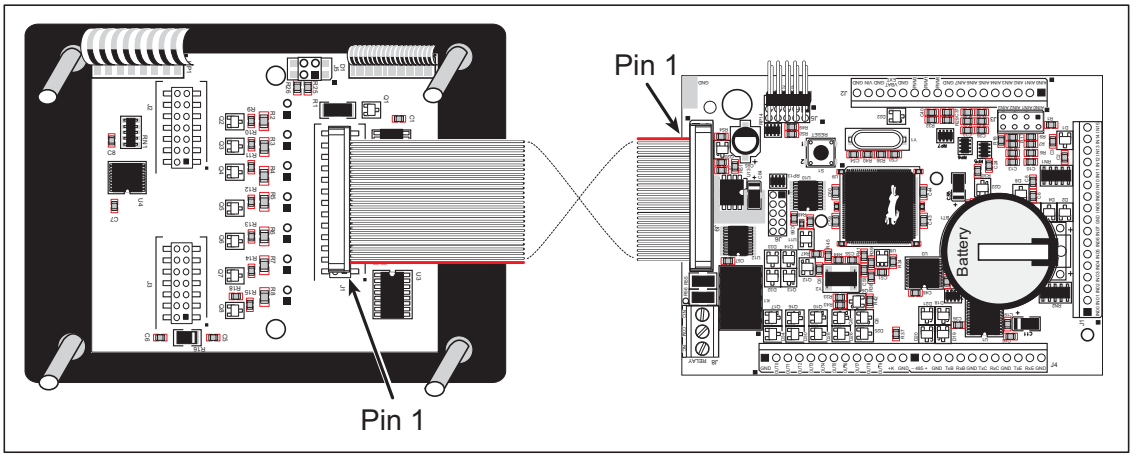


Figure C-9. Connecting LCD/Keypad Module to LP3500

Note the locations and connections relative to pin 1 on both the LP3500 and the LCD/keypad module.

Rabbit offers two different lengths of ribbon cable–5" (127 mm) and 2 ft. (60 cm). Contact your authorized distributor or a Rabbit sales representative for more information.

C.7 LCD/Keypad Module Function Calls

The LCD/keypad module is normally off. Add the **devPowerSet** function call after **brdInit** to turn on the LCD/keypad buffers.

```
brdInit();                  // Initialize the LP3500
devPowerSet(DISPDEV, 1);    // Enable LCD/keypad buffer
```

C.7.1 LEDs

When power is applied to the LCD/keypad module for the first time, the red LED (DS1) will come on, indicating that power is being applied to the LCD/keypad module. The red LED is turned off when the **brdInit** function executes.

One function is available to control the LEDs, and can be found in the **LIB\Rabbit3000\DISPLAYS\LCD122KEY7.LIB** library.

void dispLedOut(int led, int value);

LED on/off control. This function will only work when the LCD/keypad module is connected to the LP3500.

PARAMETERS

led is the LED to control.

0 = LED DS1
1 = LED DS2
2 = LED DS3
3 = LED DS4
4 = LED DS5
5 = LED DS6
6 = LED DS7

value is the value used to control whether the LED is on or off (0 or 1).

0 = off
1 = on

RETURN VALUE

None.

SEE ALSO

brdInit

C.7.2 LCD Display

The functions used to control the LCD display are contained in the **LIB\Rabbit3000\DISPLAYS\GRAPHIC.LIB** library. When *x* and *y* coordinates on the display screen are specified, *x* can range from 0 to 121, and *y* can range from 0 to 31. These numbers represent pixels from the top left corner of the display.

void glInit(void);

Initializes the display devices, clears the screen.

RETURN VALUE

None.

SEE ALSO

glDispOnOFF, glBacklight, glSetContrast, glPlotDot, glBlock, glPlotDot, glPlotPolygon, glPlotCircle, glHScroll, glVScroll, glXFontInit, glPrintf, glPutChar, glSetBrushType, glBuffLock, glBuffUnlock, glPlotLine

void glBackLight(int onOff);

Turns the display backlight on or off.

PARAMETER

onOff turns the backlight on or off

1—turn the backlight on
0—turn the backlight off

RETURN VALUE

None.

SEE ALSO

glInit, glDispOnoff, glSetContrast

void glDispOnOff(int onOff);

Sets the LCD screen on or off. Data will not be cleared from the screen.

PARAMETER

onOff turns the LCD screen on or off

1—turn the LCD screen on
0—turn the LCD screen off

RETURN VALUE

None.

SEE ALSO

glInit, glSetContrast, glBackLight

void glSetContrast(unsigned level);

Sets display contrast.

NOTE: This function is not used with the LCD/keypad module since the support circuits are not available on the LCD/keypad module.

void glFillScreen(char pattern);

Fills the LCD display screen with a pattern.

PARAMETER

The screen will be set to all black if **pattern** is 0xFF, all white if **pattern** is 0x00, and vertical stripes for any other pattern.

RETURN VALUE

None.

SEE ALSO

glBlock, glBlankScreen, glPlotPolygon, glPlotCircle

void glBlankScreen(void);

Blanks the LCD display screen (sets LCD display screen to white).

RETURN VALUE

None.

SEE ALSO

glFillScreen, glBlock, glPlotPolygon, glPlotCircle

void glBlock(int x, int y, int bmWidth, int bmHeight);

Draws a rectangular block in the page buffer and on the LCD if the buffer is unlocked. Any portion of the block that is outside the LCD display area will be clipped.

PARAMETERS

x is the *x* coordinate of the top left corner of the block.

y is the *y* coordinate of the top left corner of the block.

bmWidth is the width of the block.

bmWidth is the height of the block.

RETURN VALUE

None.

SEE ALSO

glFillScreen, glBlankScreen, glPlotPolygon, glPlotCircle

void glPlotVPolygon(int n, int *pFirstCoord);

Plots the outline of a polygon in the LCD page buffer, and on the LCD if the buffer is unlocked. Any portion of the polygon that is outside the LCD display area will be clipped. If fewer than 3 vertices are specified, the function will return without doing anything.

PARAMETERS

n is the number of vertices.

***pFirstCoord** is a pointer to array of vertex coordinates: **x1,y1, x2,y2, x3,y3,...**

RETURN VALUE

None.

SEE ALSO

glPlotPolygon, glFillPolygon, glFillVPolygon

void glPlotPolygon(int n, int y1, int x2, int y2, ...);

Plots the outline of a polygon in the LCD page buffer and on the LCD if the buffer is unlocked. Any portion of the polygon that is outside the LCD display area will be clipped. If fewer than 3 vertices are specified, the function will return without doing anything.

PARAMETERS

n is the number of vertices.

y1 is the *y* coordinate of the first vertex.

x1 is the *x* coordinate of the first vertex.

y2 is the *y* coordinate of the second vertex.

x2 is the *x* coordinate of the second vertex.

... are the coordinates of additional vertices.

RETURN VALUE

None.

SEE ALSO

glPlotVPolygon, glFillPolygon, glFillVPolygon

void glFillVPolygon(int n, int *pFirstCoord);

Fills a polygon in the LCD page buffer and on the LCD screen if the buffer is unlocked. Any portion of the polygon that is outside the LCD display area will be clipped. If fewer than 3 vertices are specified, the function will return without doing anything.

PARAMETERS

n is the number of vertices.

***pFirstCoord** is a pointer to array of vertex coordinates: **x1,y1, x2,y2, x3,y3,...**

RETURN VALUE

None.

SEE ALSO

glFillPolygon, glPlotPolygon, glPlotVPolygon

void glFillPolygon(int n, int x1, int y1, int x2, int y2, ...);

Fills a polygon in the LCD page buffer and on the LCD if the buffer is unlocked. Any portion of the polygon that is outside the LCD display area will be clipped. If fewer than 3 vertices are specified, the function will return without doing anything.

PARAMETERS

n is the number of vertices.

x1 is the *x* coordinate of the first vertex.

y1 is the *y* coordinate of the first vertex.

x2 is the *x* coordinate of the second vertex.

y2 is the *y* coordinate of the second vertex.

... are the coordinates of additional vertices.

RETURN VALUE

None.

SEE ALSO

glFillVPolygon, glPlotPolygon, glPlotVPolygon

void glPlotCircle(int xc, int yc, int rad);

Draws the outline of a circle in the LCD page buffer and on the LCD if the buffer is unlocked. Any portion of the circle that is outside the LCD display area will be clipped.

PARAMETERS

xc is the *x* coordinate of the center of the circle.

yc is the *y* coordinate of the center of the circle.

rad is the radius of the center of the circle (in pixels).

RETURN VALUE

None.

SEE ALSO

glFillCircle, glPlotPolygon, glFillPolygon

void glFillCircle(int xc, int yc, int rad);

Draws a filled circle in the LCD page buffer and on the LCD if the buffer is unlocked. Any portion of the circle that is outside the LCD display area will be clipped.

PARAMETERS

xc is the *x* coordinate of the center of the circle.

yc is the *y* coordinate of the center of the circle.

rad is the radius of the center of the circle (in pixels).

RETURN VALUE

None.

SEE ALSO

glPlotCircle, glPlotPolygon, glFillPolygon

void glXFontInit(fontInfo *pInfo, char pixWidth, char pixHeight, unsigned startChar, unsigned endChar, unsigned long xmemBuffer);

Initializes the font descriptor structure, where the font is stored in **xmem**.

PARAMETERS

***pInfo** is a pointer to the font descriptor to be initialized.

pixWidth is the width (in pixels) of each font item.

pixHeight is the height (in pixels) of each font item.

startChar is the value of the first printable character in the font character set.

endChar is the value of the last printable character in the font character set.

xmemBuffer is the **xmem** pointer to a linear array of font bitmaps.

RETURN VALUE

None.

SEE ALSO

glPrinf

unsigned long glFontCharAddr(fontInfo *pInfo, char letter);

Returns the **xmem** address of the character from the specified font set.

PARAMETERS

***pInfo** is the **xmem** address of the bitmap font set.

letter is an ASCII character.

RETURN VALUE

xmem address of bitmap character font, column major, and byte-aligned.

SEE ALSO

glPutFont, glPrintf

void glPutFont(int x, int y, fontInfo *pInfo, char code);

Puts an entry from the font table to the page buffer and on the LCD if the buffer is unlocked. Each font character's bitmap is column major and byte-aligned. Any portion of the bitmap character that is outside the LCD display area will be clipped.

PARAMETERS

x is the *x* coordinate (column) of the top left corner of the text.

y is the *y* coordinate (row) of the top left corner of the text.

***pInfo** is a pointer to the font descriptor.

code is the ASCII character to display.

RETURN VALUE

None.

SEE ALSO

glFontCharAddr, glPrintf

void glSetPfStep(int stepX, int stepY);

Sets the **glPrintf()** printing step direction. The *x* and *y* step directions are independent signed values. The actual step increments depend on the height and width of the font being displayed, which are multiplied by the step values.

PARAMETERS

stepX is the **glPrintf** *x* step value

stepY is the **glPrintf** *y* step value

RETURN VALUE

None.

SEE ALSO

Use **glGetPfStep()** to examine the current *x* and *y* printing step direction.

int glGetPfStep(void);

Gets the current **glPrintf()** printing step direction. Each step direction is independent of the other, and is treated as an 8-bit signed value. The actual step increments depends on the height and width of the font being displayed, which are multiplied by the step values.

RETURN VALUE

The *x* step is returned in the MSB, and the *y* step is returned in the LSB of the integer result.

SEE ALSO

Use **glGetPfStep()** to control the *x* and *y* printing step direction.

void glPutChar(char ch, char *ptr, int *cnt, glPutCharInst *pInst)

Provides an interface between the **STDIO** string-handling functions and the graphic library. The **STDIO** string-formatting function will call this function, one character at a time, until the entire formatted string has been parsed. Any portion of the bitmap character that is outside the LCD display area will be clipped.

PARAMETERS

ch is the character to be displayed on the LCD.

***ptr** is not used, but is a place holder for **STDIO** string functions.

***cnt** is not used, is a place holder for **STDIO** string functions.

***pInst** is a font descriptor pointer.

RETURN VALUE

None.

SEE ALSO

glPrintf, glPutFont, doprnt

void glPrintf(int x, int y, fontInfo *pInfo, char *fmt, ...);

Prints a formatted string (much like **printf**) on the LCD screen. Only the character codes that exist in the font set are printed, all others are skipped. For example, '\b', '\t', '\n' and '\r' (ASCII backspace, tab, new line, and carriage return, respectively) will be printed if they exist in the font set, but will not have any effect as control characters. Any portion of the bitmap character that is outside the LCD display area will be clipped.

PARAMETERS

x is the *x* coordinate (column) of the top left corner of the text.

y is the *y* coordinate (row) of the top left corner of the text.

***pInfo** is a font descriptor pointer.

***fmt** is a formatted string.

... are formatted string conversion parameter(s).

EXAMPLE

```
glprintf(0,0, &fi12x16, "Test %d\n", count);
```

RETURN VALUE

None.

SEE ALSO

glXFontInit

void glBuffLock(void);

Increments LCD screen locking counter. Graphic calls are recorded in the LCD memory buffer and are not transferred to the LCD if the counter is non-zero.

> **NOTE:** **glBuffLock()** and **glBuffUnlock()** can be nested up to a level of 255, but be sure to balance the calls. It is not a requirement to use these procedures, but a set of **glBuffLock()** and **glBuffUnlock()** bracketing a set of related graphic calls speeds up the rendering significantly.

RETURN VALUE

None.

SEE ALSO

glBuffUnlock, glSwap

void glBuffUnlock(void);

Decrements the LCD screen locking counter. The contents of the LCD buffer are transferred to the LCD if the counter goes to zero.

RETURN VALUE

None.

SEE ALSO

glBuffLock, glSwap

void glSwap(void);

Checks the LCD screen locking counter. The contents of the LCD buffer are transferred to the LCD if the counter is zero.

RETURN VALUE

None.

SEE ALSO

glBuffUnlock, glBuffLock, _glSwapData (located in the library specifically for the LCD that you are using)

void glSetBrushType(int type);

Sets the drawing method (or color) of pixels drawn by subsequent graphic calls.

PARAMETER

type value can be one of the following macros.

PIXBLACK draws black pixels (turns pixel on).
PIXWHITE draws white pixels (turns pixel off).
PIXXOR draws old pixel XOR'ed with the new pixel.

RETURN VALUE

None.

SEE ALSO

glGetBrushType

int glGetBrushType(void);

Gets the current method (or color) of pixels drawn by subsequent graphic calls.

RETURN VALUE

The current brush type.

SEE ALSO

glSetBrushType

void glPlotDot(int x, int y);

Draws a single pixel in the LCD buffer, and on the LCD if the buffer is unlocked. If the coordinates are outside the LCD display area, the dot will not be plotted.

PARAMETERS

x is the *x* coordinate of the dot.

y is the *y* coordinate of the dot.

RETURN VALUE

None.

SEE ALSO

glPlotline, glPlotPolygon, glPlotCircle

void glPlotLine(int x0, int y0, int x1, int y1);

Draws a line in the LCD buffer, and on the LCD if the buffer is unlocked. Any portion of the line that is beyond the LCD display area will be clipped.

PARAMETERS

x0 is the *x* coordinate of one endpoint of the line.

y0 is the *y* coordinate of one endpoint of the line.

x1 is the *x* coordinate of the other endpoint of the line.

y1 is the *y* coordinate of the other endpoint of the line.

RETURN VALUE

None.

SEE ALSO

glPlotDot, glPlotPolygon, glPlotCircle

void glLeft1(int left, int top, int cols, int rows);

Scrolls byte-aligned window left one pixel, right column is filled by current pixel type (color).

PARAMETERS

left is the top left corner of bitmap, must be evenly divisible by 8, otherwise truncates.

top is the top left corner of the bitmap.

cols is the number of columns in the window, must be evenly divisible by 8, otherwise truncates.

rows is the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glHScroll, glRight1

void glRight1(int left, int top, int cols, int rows);

Scrolls byte-aligned window right one pixel, left column is filled by current pixel type (color).

PARAMETERS

left is the top left corner of bitmap, must be evenly divisible by 8, otherwise truncates.

top is the top left corner of the bitmap.

cols is the number of columns in the window, must be evenly divisible by 8, otherwise truncates.

rows is the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glHScroll, glLeft1

void glUp1(int left, int top, int cols, int rows);

Scrolls byte-aligned window up one pixel, bottom column is filled by current pixel type (color).

PARAMETERS

left is the top left corner of bitmap, must be evenly divisible by 8, otherwise truncates.

top is the top left corner of the bitmap.

cols is the number of columns in the window, must be evenly divisible by 8, otherwise truncates.

rows is the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glVScroll, glDown1

void glDown1(int left, int top, int cols, int rows);

Scrolls byte-aligned window down one pixel, top column is filled by current pixel type (color).

PARAMETERS

left is the top left corner of bitmap, must be evenly divisible by 8, otherwise truncates.

top is the top left corner of the bitmap.

cols is the number of columns in the window, must be evenly divisible by 8, otherwise truncates.

rows is the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glVScroll, glUp1

```
void glHScroll(int left, int top, int cols,
   int rows, int nPix);
```

Scrolls right or left, within the defined window by *x* number of pixels. The opposite edge of the scrolled window will be filled in with white pixels. The window must be byte-aligned.

Parameters will be verified for the following:

1. The **`left`** and **`cols`** parameters will be verified that they are evenly divisible by 8. If not, they will be truncated to a value that is a multiple of 8.
2. Parameters will be checked to verify that the scrolling area is valid. The minimum scrolling area is a width of 8 pixels and a height of one row.

PARAMETERS

`left` is the top left corner of bitmap, must be evenly divisible by 8.

`top` is the top left corner of the bitmap.

`cols` is the number of columns in the window, must be evenly divisible by 8.

`rows` is the number of rows in the window.

`nPix` is the number of pixels to scroll within the defined window (a negative value will produce a scroll to the left).

RETURN VALUE

None.

SEE ALSO

`glVScroll`

void glVScroll(int left, int top, int cols, int rows, int nPix);

Scrolls up or down, within the defined window by *x* number of pixels. The opposite edge of the scrolled window will be filled in with white pixels. The window must be byte-aligned.

Parameters will be verified for the following:

1. The **left** and **cols** parameters will be verified that they are evenly divisible by 8. If not, they will be truncated to a value that is a multiple of 8.
2. Parameters will be checked to verify that the scrolling area is valid. The minimum scrolling area is a width of 8 pixels and a height of one row.

PARAMETERS

left is the top left corner of bitmap, must be evenly divisible by 8.

top is the top left corner of the bitmap.

cols is the number of columns in the window, must be evenly divisible by 8.

rows is the number of rows in the window.

nPix is the number of pixels to scroll within the defined window (a negative value will produce a scroll up).

RETURN VALUE

None.

SEE ALSO

glHScroll

void glXPutBitmap(int left, int top, int width, int height, unsigned long bitmap);

Draws bitmap in the specified space. The data for the bitmap are stored in **xmem**. This function calls **glXPutFastmap** automatically if the bitmap is byte-aligned (the left edge and the width are each evenly divisible by 8).

Any portion of a bitmap image or character that is outside the LCD display area will be clipped.

PARAMETERS

left is the top left corner of the bitmap.

top is the top left corner of the bitmap.

width is the width of the bitmap.

height is the height of the bitmap.

bitmap is the address of the bitmap in **xmem**.

RETURN VALUE

None.

SEE ALSO

glXPutFastmap, glPrintf

```
void glXPutFastmap(int left, int top, int width,
   int height, unsigned long bitmap);
```

Draws bitmap in the specified space. The data for the bitmap are stored in **xmem**. This function is like **glXPutBitmap**, except that it is faster. The restriction is that the bitmap must be byte-aligned.

Any portion of a bitmap image or character that is outside the LCD display area will be clipped.

PARAMETERS

left is the top left corner of the bitmap, must be evenly divisible by 8, otherwise truncates.

top is the top left corner of the bitmap.

width is the width of the bitmap, must be evenly divisible by 8, otherwise truncates.

height is the height of the bitmap.

bitmap is the address of the bitmap in **xmem**.

RETURN VALUE

None.

SEE ALSO

glXPutBitmap, glPrintf

```
int TextWindowFrame(windowFrame *window,
   fontInfo *pFont, int x, int y, int winWidth,
   int winHeight)
```

Defines a text-only display window. This function provides a way to display characters within the text window using only character row and column coordinates. The text window feature provides end-of-line wrapping and clipping after the character in the last column and row is displayed.

NOTE: Execute the **TextWindowFrame** function before other **Text...** functions.

PARAMETERS

***window** is a window frame descriptor pointer.

***pFont** is a font descriptor pointer.

x is the *x* coordinate of the top left corner of the text window frame.

y is the *y* coordinate of the top left corner of the text window frame.

winWidth is the width of the text window frame.

winHeight is the height of the text window frame.

RETURN VALUE

0—window frame was successfully created.
-1—*x* coordinate + width has exceeded the display boundary.
-2—*y* coordinate + height has exceeded the display boundary.

```
void TextGotoXY(windowFrame *window, int col,
   int row);
```

Sets the cursor location to display the next character. The display location is based on the height and width of the character to be displayed.

NOTE: Execute the **`TextWindowFrame`** function before using this function.

PARAMETERS

`*window` is a pointer to a font descriptor.

`col` is a character column location.

`row` is a character row location.

RETURN VALUE

None.

SEE ALSO

`TextPutChar, TextPrintf, TextWindowFrame`

```
void TextCursorLocation(windowFrame *window,
   int *col, int *row);
```

Gets the current cursor location that was set by a Graphic **`Text...`** function.

NOTE: Execute the **`TextWindowFrame`** function before using this function.

PARAMETERS

`*window` is a pointer to a font descriptor.

`*col` is a pointer to cursor column variable.

`*row` is a pointer to cursor row variable.

RETURN VALUE

Lower word = Cursor Row location
Upper word = Cursor Column location

SEE ALSO

`TextGotoXY, TextPrintf, TextWindowFrame, TextCursorLocation`

void TextPutChar(struct windowFrame *window, char ch);

Displays a character on the display where the cursor is currently pointing. If any portion of a bitmap character is outside the LCD display area, the character will not be displayed. The cursor increments its position as needed.

NOTE: Execute the **TextWindowFrame** function before using this function.

PARAMETERS

***window** is a pointer to a font descriptor.

ch is a character to be displayed on the LCD.

RETURN VALUE

None.

SEE ALSO

TextGotoXY, TextPrintf, TextWindowFrame, TextCursorLocation

void TextPrintf(struct windowFrame *window, char *fmt, ...);

Prints a formatted string (much like **printf**) on the LCD screen. Only printable characters in the font set are printed, also escape sequences, '\r' and '\n' are recognized. All other escape sequences will be skipped over; for example, '\b' and 't' will print if they exist in the font set, but will not have any effect as control characters.

The text window feature provides end-of-line wrapping and clipping after the character in the last column and row is displayed. The cursor then remains at the end of the string.

NOTE: Execute the **TextWindowFrame** function before using this function.

PARAMETERS

***window** is a pointer to a font descriptor.

***fmt** is a formatted string.

... are formatted string conversion parameter(s).

EXAMPLE

```
TextPrintf(&TextWindow, "Test %d\n", count);
```

RETURN VALUE

None.

SEE ALSO

TextGotoXY, TextPutChar, TextWindowFrame, TextCursorLocation

C.7.3 Keypad

The functions used to control the keypad are contained in the Dynamic C **LIB\Rabbit3000\DISPLAYS\KEYPADS\KEYPAD7.LIB** library.

```
void keyInit(void);
```

Initializes keypad process

RETURN VALUE

None.

SEE ALSO

brdInit

```
void keyConfig(char cRaw, char cPress,
   char cRelease, char cCntHold, char cSpdLo,
   char cCntLo, char cSpdHi);
```

Assigns each key with key press and release codes, and hold and repeat ticks for auto repeat and debouncing.

PARAMETERS

cRaw is a raw key code index.

1x7 keypad matrix with raw key code index assignments (in brackets):

[0]	[1]	[2]	[3]
[4]	[5]	[6]	

User Keypad Interface

cPress is a key press code

An 8-bit value is returned when a key is pressed.
0 = Unused.

See **keypadDef()** for default press codes.

cRelease is a key release code.

An 8-bit value is returned when a key is pressed.
0 = Unused.

cCntHold is a hold tick, which is approximately one debounce period or 5 µs.

How long to hold before repeating.
0 = No Repeat.

cSpdLo is a low-speed repeat tick, which is approximately one debounce period or 5 µs.

How many times to repeat.
0 = None.

cCntLo is a low-speed hold tick, which is approximately one debounce period or 5 µs.

How long to hold before going to high-speed repeat.
0 = Slow Only.

cSpdHi is a high-speed repeat tick, which is approximately one debounce period or 5 µs.

How many times to repeat after low speed repeat.
0 = None.

RETURN VALUE

None.

SEE ALSO

keyProcess, keyGet, keypadDef

void keyProcess(void);

Scans and processes keypad data for key assignment, debouncing, press and release, and repeat.

NOTE: This function is also able to process an 8 x 8 matrix keypad.

RETURN VALUE

None

SEE ALSO

keyConfig, keyGet, keypadDef

char keyGet(void);

Get next keypress.

RETURN VALUE

The next keypress, or 0 if none

SEE ALSO

keyConfig, keyProcess, keypadDef

int keyUnget(char cKey);

Pushes the value of **cKey** to the top of the input queue, which is 16 bytes deep.

PARAMETER

cKey

RETURN VALUE

None.

SEE ALSO

keyGet

void keypadDef();

Configures the physical layout of the keypad with the default ASCII return key codes.

Keypad physical mapping 1 x 7

0	4	1	5	2	6	3
['L']		['U']		['D']		['R']
	['–']		['+']		['E']	

where

'D' represents Down Scroll
'U' represents Up Scroll
'R' represents Right Scroll
'L' represents Left Scroll
'–' represents Page Down
'+' represents Page Up
'E' represents the ENTER key

Example: Do the followingfor the above physical vs. ASCII return key codes.

```
keyConfig ( 3,'R',0, 0, 0, 0, 0 );
keyConfig ( 6,'E',0, 0, 0, 0, 0 );
keyConfig ( 2,'D',0, 0, 0, 0, 0 );
keyConfig ( 4,'-',0, 0, 0, 0, 0 );
keyConfig ( 1,'U',0, 0, 0, 0, 0 );
keyConfig ( 5,'+',0, 0, 0, 0, 0 );
keyConfig ( 0,'L',0, 0, 0, 0, 0 );
```

Characters are returned upon keypress with no repeat.

RETURN VALUE

None.

SEE ALSO

keyConfig, keyGet, keyProcess

void keyScan(char *pcKeys);

Writes "1" to each row and reads the value. The position of a keypress is indicated by a zero value in a bit position.

PARAMETER

***pcKeys** is a pointer to the address of the value read.

RETURN VALUE

None.

SEE ALSO

keyConfig, keyGet, keypadDef, keyProcess

C.8 Sample Programs

Sample programs illustrating the use of the LCD/keypad module with the LP3500 board are provided in the **`SAMPLES\LP3500\Display_Keypad`** directory.

To run a sample program, open it with the **File** menu (if it is not still open), compile it using the **Compile** menu, and then run it by selecting **Run** in the **Run** menu. The LP3500 must be connected to a PC using the programming cable as described in Section 2.1, “LP3500 Connections.” Each sample program contains detailed instructions for running it.

APPENDIX D. PLASTIC ENCLOSURE

The plastic enclosure provides a secure way to protect your LP3500. The enclosure itself may be mounted on any flat surface.

The complete plastic enclosure consists of a base and a cover. The base alone is a convenient surface on which to mount the LP3500, and also provides a means to mount the LP3500 on any flat surface.

Appendix D describes how to mount the LP3500 inside the plastic enclosure, and provides details on mounting the assembly. The plastic enclosure is able to accommodate the following LP3500 combinations.

- LP3500 board only
- LP3500 Prototyping Board only
- LP3500 mounted on LP3500 Prototyping Board

D.1 Assembly Instructions

1. Remove any stand-offs on the LP3500 board or LP3500 board/Prototyping Board combination to be enclosed.

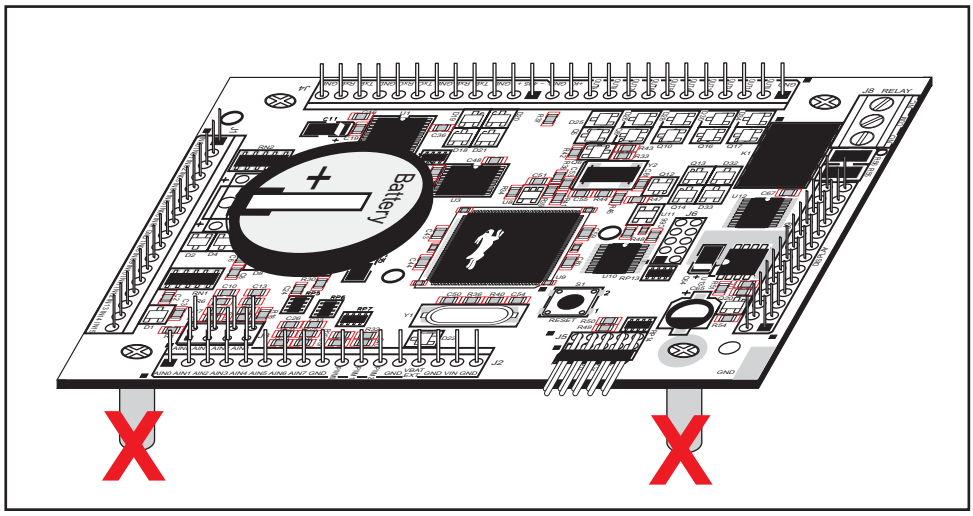


Figure D-1. Remove Stand-Offs from LP3500 Board

2. Attach the LP3500, the LP3500 Prototyping Board, or the board combination to the plastic enclosure base.

Position the board(s) over the plastic enclosure base as shown below in Figure D-2. Attach the board(s) to the base using the two 4-40 × ¼ screws supplied with the enclosure base.

NOTE: You will need longer 4-40 screws if you are mounting the combination with the LP3500 installed on the Prototyping Board.

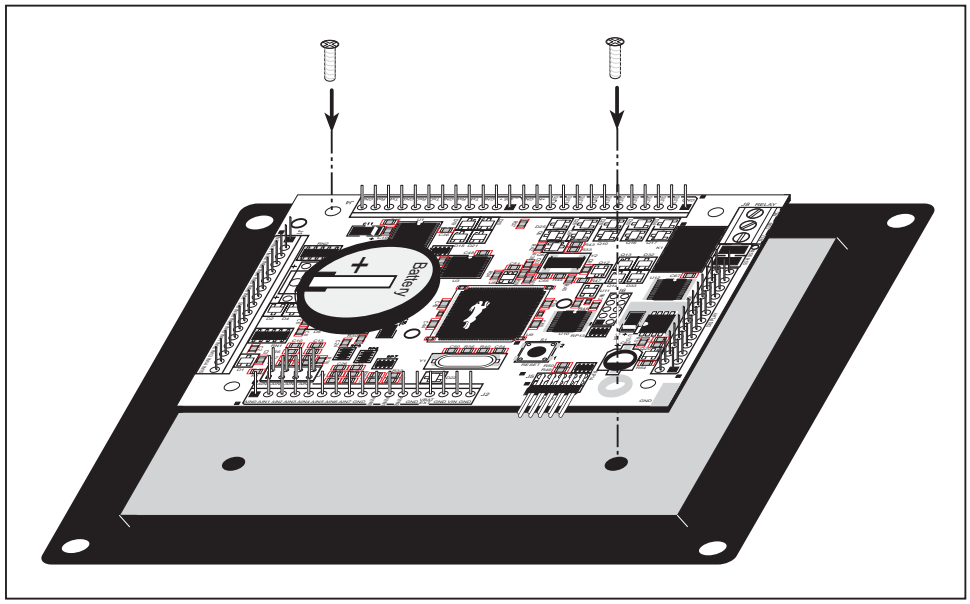


Figure D-2. Attach Board(s) to Plastic Enclosure Base

3. Mount plastic enclosure (optional).

Use four #10 screws to attach the plastic enclosure at the four outer corner mounting holes to the surface on which it will be mounted. This step will be most suitable to production versions of LP3500 units once development has been completed.

4. Attach the enclosure cover to the base.

Position the cover over the plastic enclosure base as shown below in Figure D-3. Attach the cover to the base using the two 4-40 × 7/8 screws supplied.

> **NOTE:** You will need longer 4-40 screws if you are mounting the combination with the LP3500 installed on the Prototyping Board.

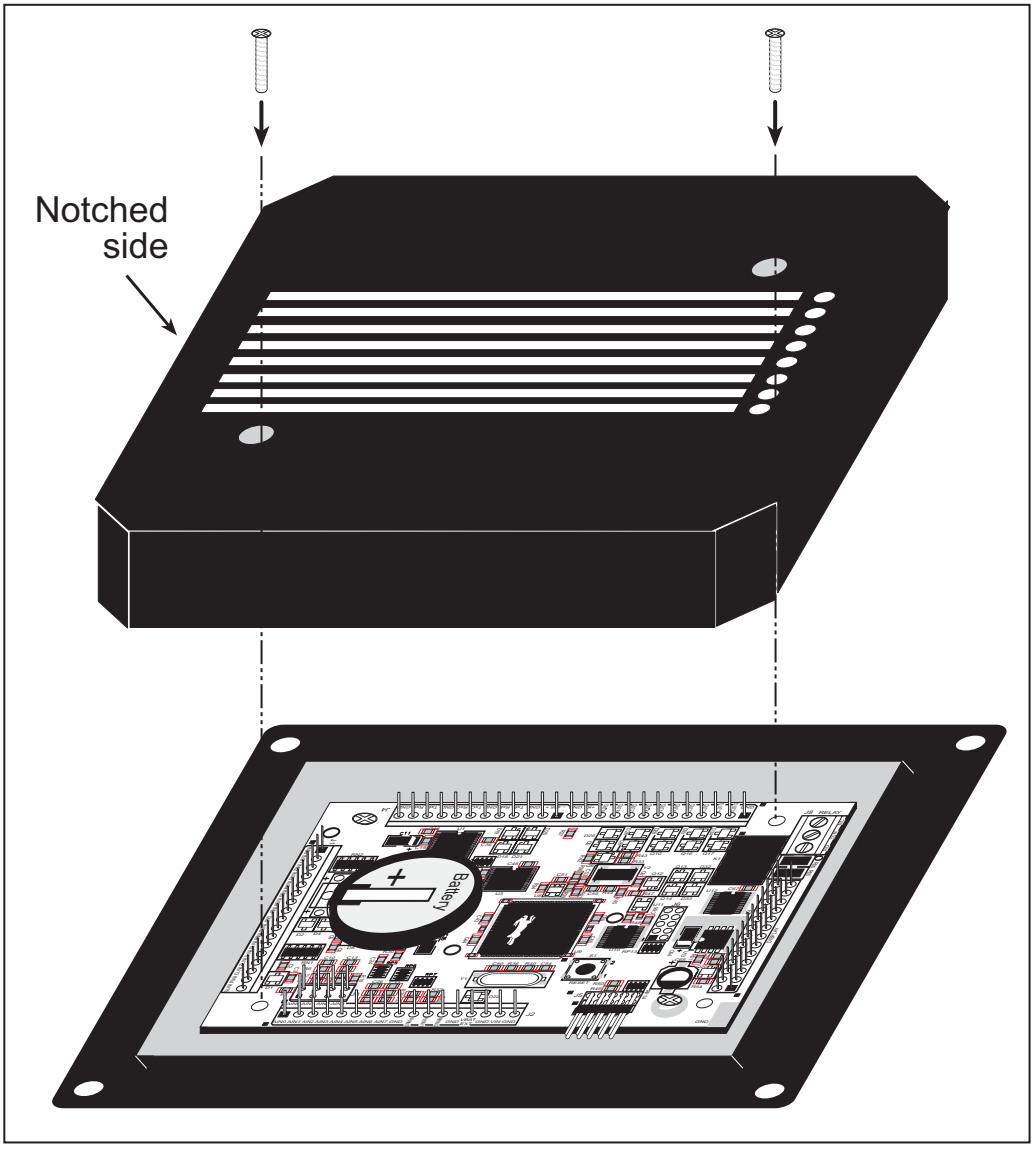


Figure D-3. Attach Enclosure Top

D.2 Dimensions

Figure D-4 shows the dimensions for the plastic enclosure.

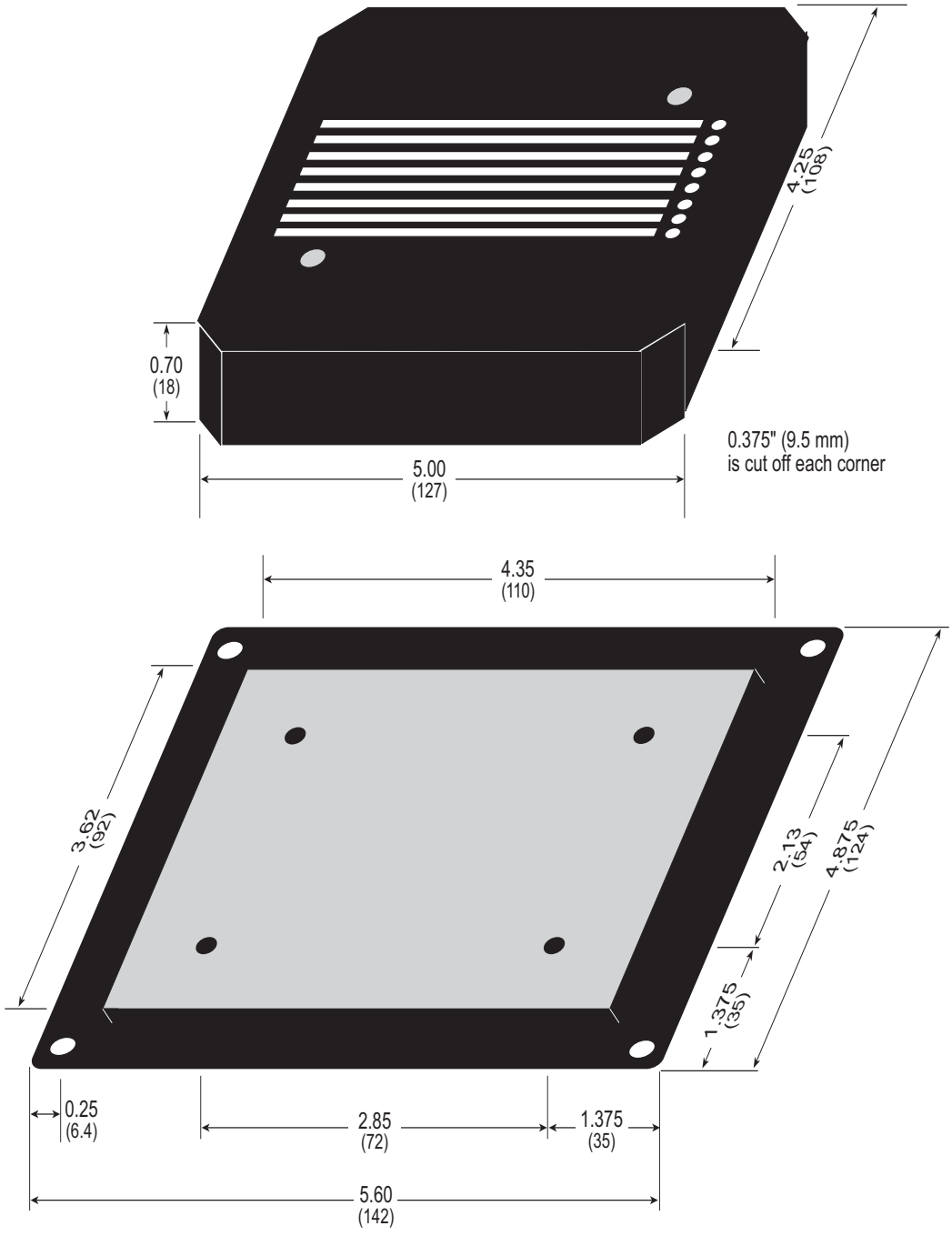


Figure D-4. Plastic Enclosure Dimensions

APPENDIX E. POWER MANAGEMENT

Appendix E describes the power circuitry provided on the LP3500. The LP3500 can operate from an unregulated external power source, or from an external battery. There is onboard battery backup for the SRAM and the real-time clock.

E.1 External Power Supply

Power is normally supplied to the LP3500 via pins 16 and 17 of header J2 on the LP3500. The Prototyping Board provides a convenient header plug for use with the AC adapter included with the LP3500 Tool Kit. The Prototyping Board includes a Shottky diode for protection against reverse polarity.

The raw DC power, VIN, goes through a linear regulator as shown in Figure E-1. The linear regulator outputs a Vcc of 2.8 V DC.

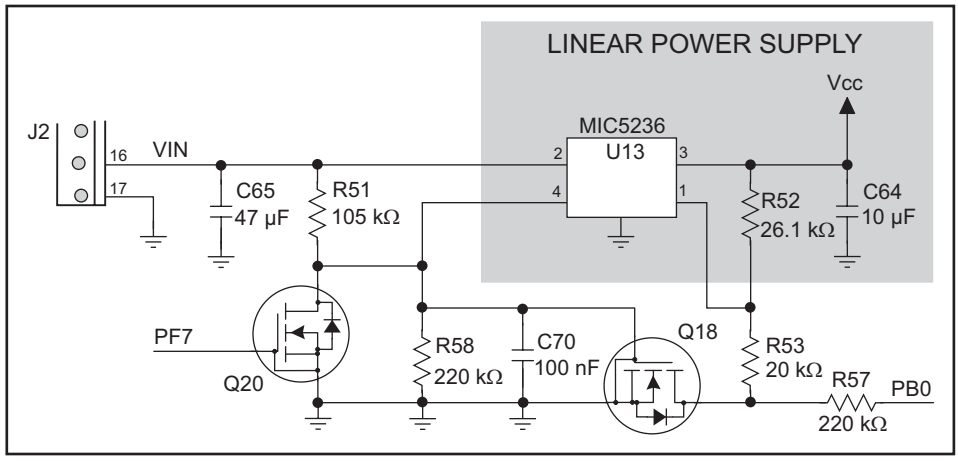


Figure E-1. LP3500 Linear Regulator Circuit

The power necessarily dissipated by the regulator can be calculated if both the external input voltage and the current drawn by the LP3500 are known. The current provided by the high-power output drivers does not have to be included if a separate power supply is connected to K so that power does not come from Vcc.

The linear regulator maintains its output voltage to within ±5% as long as the linear regulator is dissipating less than 0.75 W. Thermal shutdown turns the regulator off when it overheats. Figure E-1 shows the power operating curves for the specified VIN range of 3–30 V DC. Note that while a VIN range of 3–30 V is possible, 3–15 V is recommended to allow reasonable current.

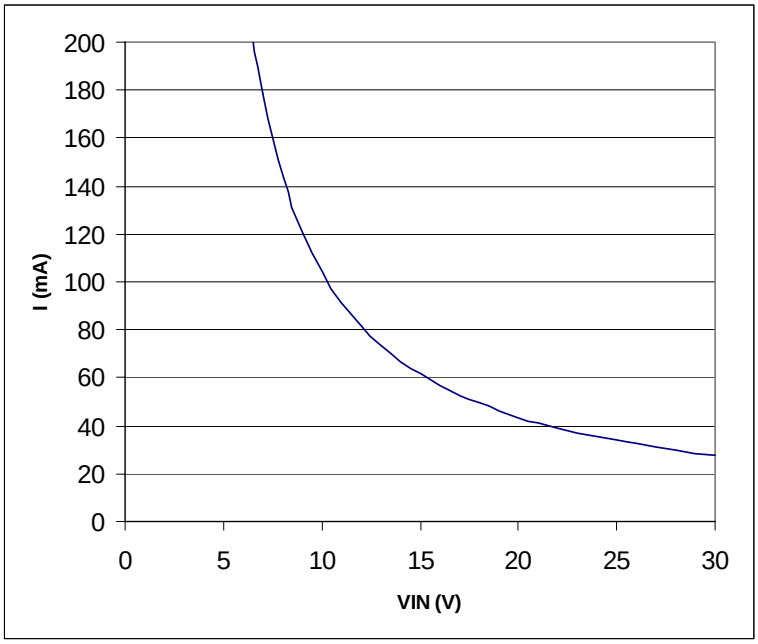


Figure E-2. Linear Regulator Power Operating Curve

The LP3500 can operate at various different power levels, depending on which sections of the board are turned off using the **`devPowerSet`** function. Table E-1 lists the sections.

Table E-1. LP3500 Sections That Can Be Turned Off

LP3500 Section	Power Consumption	Controlled by Rabbit 3000 Port
RS-232	8 mA with load	PG1
RS-485	0.1 mA with no load	PG0
A/D Converter (LP3500 model only)	0.45 mA at 1.5 ksample/s sampling rate	PB6
Relay (LP3500 model only)	120 mA for 10 ms per switch	PG4, PG5
LCD/Keypad Module	5 mA without backlight, 50 mA for backlight, 5 mA for each LED (max. 7 LEDs)	PB0

NOTE: RxE always remains active to allow the LP3500 to "listen" while it is in the power-save mode.

Parallel Port PF7 on the Rabbit 3000 chip controls whether the linear regulator is on or off. Parallel Port PB0 senses whether there is an output from the linear regulator, and shuts off the RS-232 (except RxE, which is used to "listen"), RS-485, A/D converter, and relay sections via Parallel Ports PB6, PG0, PG1, PG4, and PG5 to conserve power.

E.2 Batteries and External Battery Connections

The SRAM and the real-time clock have battery backup. Power to the SRAM and the real-time clock (VRAM) is provided by two different sources, depending on whether the main part of the LP3500 is powered or not. When the LP3500 is powered normally, and Vcc is within operating limits, the SRAM and the real-time clock are powered from Vcc. If power to the board is lost or falls below 2.75 V, the VRAM and real-time clock power will come from either the onboard or the external battery. The reset generator circuit controls the source of power by way of its **/RESET** output signal.

A replaceable onboard 265 mA·h lithium battery provides power to the real-time clock and SRAM when all external power is removed from the circuit board and the LP3500 processor is off. The drain on the battery is typically 46 µA under these worst-case conditions, and so the expected life of the onboard battery is

$$\frac{265\ \text{mA}\cdot\text{h}}{46\ \mu\text{A}} = 240\ \text{days}.$$

The drain on the battery is typically less than 4 µA when external power ***is*** applied, and so the expected LP3500 battery in-service life is

$$\frac{265\ \text{mA}\cdot\text{h}}{4\ \mu\text{A}} = 7.5\ \text{years}.$$

The primary role of the onboard battery is to keep the SRAM and the real-time clock functional when the LP3500 processor is off. Even though there is limited capacity in the onboard battery, and its circuit is in parallel with that of the external battery, Rabbit strongly recommends against relying on the onboard battery for any role besides that of keeping the SRAM and the real-time clock functional when the LP3500 processor is off. The short in-service life of the onboard battery highlights the importance of an external battery, especially if the LP3500 is to operate in the power-save mode. A 2.8–3.3 V external battery or equivalent "regulated" voltage is recommended.

There is provision for an external battery connection via pins 14 and 15 on header J2 on the LP3500, and the LP3500 Prototyping Board has a plug-in header at J6 that can be used to connect an external battery to the LP3500/Prototyping Board combination.

Cycle the main power off/on on the LP3500 after you install a backup battery for the first time, and whenever you replace the battery. This step will minimize the current drawn by the real-time clock oscillator circuit from the backup battery should the LP3500 experience a loss of main power.

E.2.1 Replacing the Backup Battery

The battery is user-replaceable, and is fitted in a battery holder. To replace the battery, lift up on the spring clip and slide out the old battery. Use only a Panasonic CR2330 or equivalent replacement battery, and insert it into the battery holder with the + side facing up.

NOTE: The SRAM contents and the real-time clock settings will be lost if the battery is replaced with no power applied to the LP3500. Exercise care if you replace the battery while external power is applied to the LP3500.

NOTE: Should it be more convenient, it is also possible to install the onboard battery on the other side of the LP3500 board.

CAUTION: There is an explosion danger if the battery is short-circuited, recharged, or replaced incorrectly. Replace the battery only with the same type or an equivalent type recommended by the battery manufacturer. Dispose of used batteries according to the battery manufacturer's instructions.

E.2.2 Power to VRAM Switch

The VRAM switch on the LP3500 module, shown in Figure E-3, allows the battery backup to provide power when the external power goes off. The switch provides an isolation between Vcc and the battery when Vcc goes low. This prevents the Vcc line from draining the battery.

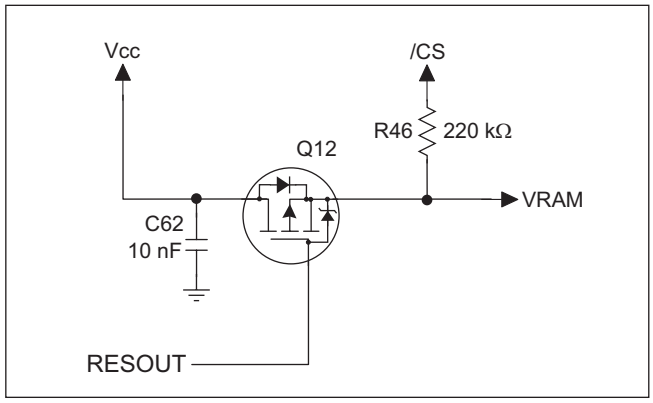


Figure E-3. VRAM Switch

Field-effect transistor Q12 provides a very small voltage drop between Vcc and VRAM (<100 mV, typically 10 mV) so that the board components powered by Vcc will not have a significantly different voltage than VRAM.

When the LP3500 is ***not*** in reset, the **/RESOUT** line will be high. This allows VRAM to nearly equal Vcc.

When the LP3500 ***is*** in reset, the **/RESOUT** line will go low. This provides an isolation between Vcc and VRAM.

E.2.3 Reset Generator

The LP3500 module uses a reset generator on the module, U11, to reset the Rabbit 3000 microprocessor when the voltage drops below the voltage necessary for reliable operation. The reset occurs between 2.55 V and 2.75 V, typically 2.63 V.

E.3 Chip Select Circuit

The current drain on the battery in a battery-backed circuit must be kept at a minimum. When the LP3500 is not powered, the battery keeps the SRAM memory contents and the real-time clock (RTC) going. The SRAM has a powerdown mode that greatly reduces power consumption. This powerdown mode is activated by raising the chip select (CS) signal line. Normally the SRAM requires Vcc to operate. However, only 2 V is required for data retention in powerdown mode. Thus, when power is removed from the circuit, the battery voltage needs to be provided to both the SRAM power pin and to the CS signal line. The CS control circuit accomplishes this task for the SRAM's chip select signal line.

In a powered-up condition, the CS control circuit must allow the processor's chip select signal /CS1 to control the SRAM's CS signal /CSRAM. So, with power applied, /CSRAM must be the same signal as /CS1, and with power removed, /CSRAM must be held high (but only needs to be battery voltage high).The isolated /CSRAM line has a 220 kΩ pullup resistor to VRAM (R46). This pullup resistor keeps /CSRAM at the VRAM voltage level (which under no power condition is the backup battery's regulated voltage at a little more than 2 V).

APPENDIX F. RUNNING A SAMPLE PROGRAM

Appendix G goes through the steps of running a sample program with the LP3500 connected to the Prototyping Board.

Sample programs are provided in the Dynamic C **Samples** folder. The various directories in the **Samples** folder contain specific sample programs that illustrate the use of the corresponding Dynamic C libraries.

The **LP3500** folder provides sample programs specific to the LP3500. Each sample program has comments that describe the purpose and function of the program. Follow the instructions at the beginning of the sample program.

Let's look at the sample program **DIGIN.C** in the **Samples/LP3500/IO** folder.

1. Connect the LP3500 to a PC using the programming cable as described in Section 2.1, "LP3500 Connections," and connect the AC adapter to header J5 on the Prototyping Board.

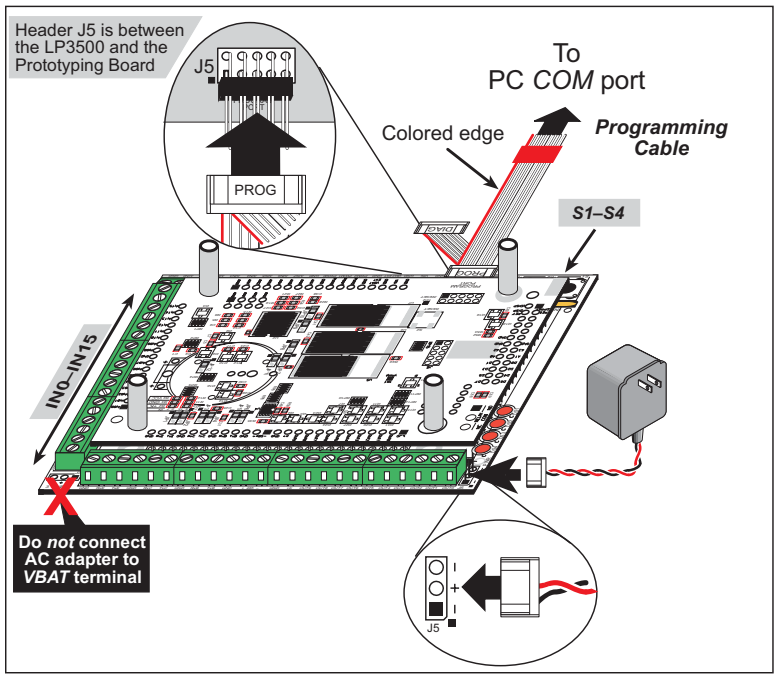
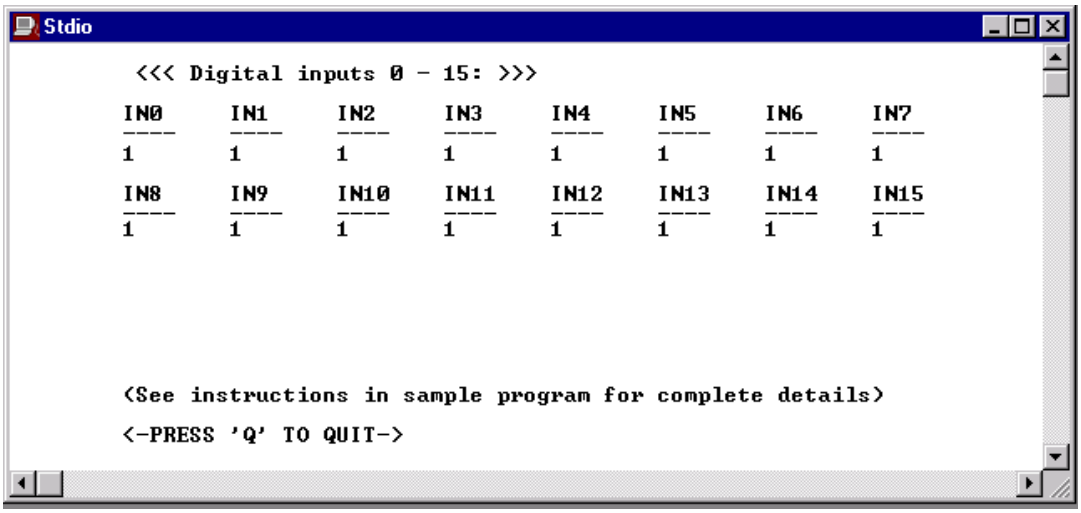


Figure F-1. Programming Cable and Power Supply Connections

2. +K must be connected to an external power supply. A 0 Ω resistor on the Prototyping Board (R1) ties +K to VIN from the AC adapter and thus satisfies this requirement as long as the LP3500 is connected to the Prototyping Board.
3. Open **DIGIN.C** with the **File** menu, compile it using the **Compile** menu, and then run it by selecting **Run** in the **Run** menu.
4. The following display will appear in the Dynamic C **STDIO** window.



5. When pressing pushbutton switches S1–S4 on the Prototyping Board you can change the inputs for IN0–IN3 from a “1” to a “0.”
6. Similarly you can view a change to the inputs for IN0–IN15 from a “1” to a “0” when you touch a wire connected to ground to IN0–IN15.

INDEX

M

O

P

R

S

T

U

SCHEMATICS

090-0150 LP3500 Schematic

www.rabbit.com/documentation/schemat/090-0150.pdf

090-0151 LP3500 Prototyping Board Schematic

www.rabbit.com/documentation/schemat/090-0151.pdf

090-0156 LCD/Keypad Module Schematic

www.rabbit.com/documentation/schemat/090-0156.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.