

LP3100

C-Programmable Controller

User's Manual

Revision E



LP3100 User's Manual

Part Number 019-0049 • Revision E
Last revised on February 3, 2000 • Printed in U.S.A.

Copyright

© 1999 Z-World, Inc. • All rights reserved.

Z-World reserves the right to make changes and improvements to its products without providing notice.

Trademarks

- Dynamic C[®] is a registered trademark of Z-World, Inc.
 - Windows[®] is a registered trademark of Microsoft Corporation
 - Hayes Smart Modem[®] is a registered trademark of Hayes Microcomputer Products, Inc.
-

Notice to Users

When a system failure may cause serious consequences, protecting life and property against such consequences with a backup system or safety device is essential. The buyer agrees that protection against consequences resulting from system failure is the buyer's responsibility.

This device is not approved for life-support or medical systems.

All Z-World products are 100 percent functionally tested. Additional testing may include visual quality control inspections or mechanical defects analyzer inspections. Specifications are based on characterization of tested sample units rather than testing over temperature and voltage of each unit. Z-World may qualify components to operate within a range of parameters that is different from the manufacturer's recommended range. This strategy is believed to be more economical and effective. Additional testing or burn-in of an individual unit is available by special arrangement.

Company Address

Z-World, Inc.

2900 Spafford Street
Davis, California 95616-6800
USA

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

TABLE OF CONTENTS

About This Manual

Chapter 1: Overview	1
1.1 Features	2
1.2 Standard Models	3
1.3 Flexibility and Customization	3
1.4 Subsystems	4
1.4.1 Microprocessor, Memory, and Support Circuits	5
1.4.2 Power Control	5
1.4.3 Digital Inputs/Outputs	5
1.4.4 Analog Inputs	6
1.4.5 Serial Channels	6
1.4.6 Real-Time Clock	6
1.4.7 LPBus	6
1.4.8 LED	6
1.5 Development and Evaluation Tools	7
1.5.1 Development Kit Packing List	7
1.6 Software	7
1.7 CE Compliance	8
Chapter 2: Getting Started	9
2.1 Operating Modes	10
2.1.1 Changing Operating Modes	11
2.1.2 Using a SIB2	11
2.2 Connecting an LP3100 to a PC	12
2.3 Establishing Communication with an LP3100	15
2.4 Running a Sample Program	15
Chapter 3: Subsystems	17
3.1 Subsystems Overview	18
3.2 Microprocessor, Memory, and Support Circuits	19
3.2.1 Microprocessor Supervisor	19
3.2.2 Flash EPROM	19
3.3 Power Control	21
3.3.1 Power Supplies	21
3.3.2 Power Supply Control	23
3.3.3 Shutting Down VCC	24
3.3.4 Clock Speed	24
3.3.5 Microprocessor Operating Modes and Shutdown Mode	25
3.3.6 Component Shutdown	27
3.4 Digital Input/Output	28
3.4.1 Digital I/O Operating Modes and Configuration	28
3.4.2 Digital Inputs	28
3.4.3 Digital Outputs	30
3.4.4 Digital Inputs/Outputs	33
3.5 Analog Inputs	37
3.5.1 Scaling Input Range	40
3.6 Operation	42
3.6.1 The VOFF Voltage Divider	42
3.6.2 DC Gain	43
3.6.3 Finding VOFF	44
3.6.4 Practical Considerations	45
3.6.5 Input Impedance	46
3.6.6 Frequency Response	46

3.6.7 Using the ADC	47
3.6.8 Using the Analog Inputs	47
3.7 Serial Communication	48
3.7.1 Operation	48
3.7.2 RS-232 Communication	50
3.7.3 RS-485 Communication	50
3.7.4 Software	51
3.8 Real-Time Clock	51
3.8.1 Real-Time Clock Interrupts	52
3.8.2 Periodic Interrupts	53
3.8.3 Alarm Interrupts	54
3.8.4 Wake Up VCC	54
3.9 LPBus	55
3.9.1 LPBus Signals	57
3.9.2 Board ID	59
Chapter 4: Software Reference	61
4.1 Using Dynamic C Drivers	62
4.2 Digital Input/Output Functions	62
4.3 Analog Input Functions	68
4.4 Serial Communication Functions	72
4.4.1 RS-485 Functions	72
4.4.2 RS-232 Functions	73
4.5 Power Control Functions	74
4.6 RTC Functions	79
4.7 Flash EPROM Functions	82
4.8 LED Functions	82
4.9 LCD Functions	82
4.10 Keypad Functions	85
4.11 Additional Software	87
Appendix A: Troubleshooting	89
A.1 Out of the Box	90
A.2 Dynamic C Does Not Start	90
A.3 LP3100 Repeatedly Resets	91
A.4 Dynamic C Loses Link with Application Program	91
Appendix B: Specifications	93
B.1 General Specifications	94
B.2 Analog Inputs	97
B.3 Mechanical Specifications	98
B.3.1 LP3100 Mounting Plate Dimensions	99
B.4 Header Pinouts	100
2.4.1 LCD Connections	101
B.5 Jumper Settings	102
Appendix C: Serial Interface Board	103
C.1 Features	104
C.2 External Dimensions	105
Appendix D: Development Board	107
D.1 Overview	108
D.1.1 LCD Interface	109
D.1.2 Keypad Interface	110
D.1.3 RS-232 Channel Connectors	111
D.1.4 I/O Header	112

D.1.5 Power Supply Input Connector	112
D.1.6 Reset Switch	112
D.1.7 Prototyping Area	112
D.2 Dimensions	113
Appendix E: LPBus Prototyping Board	115
E.1 Overview	116
E.1.1 Installation of LPBus Prototyping Board	117
E.1.2 Prototyping Area	117
E.1.3 LPBus Signals	118
E.2 Design Considerations	118
E.2.1 Electrical	118
E.2.2 Board ID	119
E.2.3 No Connect Pins	121
E.2.4 Use of DS Lines	121
E.2.5 DMA	121
E.3 LPBus Timing	122
E.4 LPBus DMA	123
E.5 Dimensions	125
Appendix F: Power Management	127
F.1 Input Voltage	128
F.2 Power-Failure Detection	130
F.2.1 Power Failure Sequence of Events	130
Appendix G: Interrupts	133
G.1 Enabling/Disabling Interrupts	134
G.2 Interrupt Service Routines	134
G.3 Interrupt Vectors	135
G.4 Jump Vectors	136
Appendix H: Addresses	137
H.1 Simulated EEPROM Addresses	138
H.2 Microprocessor Register Addresses	138
H.3 LP3100 Peripheral Addresses	138
Appendix I: Optional Second Flash EPROM	141
I.1 Optional Flash EPROM	142
I.2 Sample Program	143
Index	145
Schematics	

ABOUT THIS MANUAL


This manual provides instructions for installing, testing, configuring, and interconnecting the LP3100 low-power controller.

Instructions to get started using Dynamic C software programming functions as well as complete C and Dynamic C references and programming resources are referenced when necessary.

Assumptions

Assumptions are made regarding the user's knowledge and experience in the following areas:

- Ability to design and engineer the target system that an LP3100 will control.
- Understanding of the basics of operating a software program and editing files under Windows on a PC.
- Knowledge of the basics of C programming.


 For a full treatment of C, refer to the following texts:

The C Programming Language by Kernighan and Ritchie (published by Prentice-Hall).

and/or

C: A Reference Manual by Harbison and Steel (published by Prentice-Hall).

- Knowledge of basic Z80 assembly language and architecture..

 For documentation from Zilog, refer to the following texts:

Z180 MPU User's Manual

Z180 Serial Communication Controllers

Z80 Microprocessor Family User's Manual.

Acronyms

Table 1 lists and defines the acronyms that may be used in this manual.







Table 1. Acronyms

Acronym	Meaning
EPROM	Erasable Programmable Read-Only Memory
EEPROM	Electrically Erasable Programmable Read-Only Memory
LCD	Liquid Crystal Display
LED	Light-Emitting Diode
NMI	Nonmaskable interrupt
PIO	Parallel Input/Output (Individually programmable input/output)
PRT	Programmable Reload Timer
RAM	Random Access Memory
RTC	Real-Time Clock
SIB	Serial Interface Board
SRAM	Static Random Access Memory
UART	Universal Asynchronous Receiver Transmitter

Icons

Table 2 displays and defines icons that may be used in this manual.

Table 2. Icons

Icon	Meaning	Icon	Meaning
	Refer to or see		Note
	Please contact	Tip	Tip
	Caution		High Voltage
	Factory Default		

Conventions

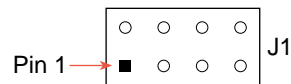
Table 3 lists and defines the typographic conventions that may be used in this manual.

Table 3. Typographic Conventions

Example	Description
while	Bold Courier font indicates a program, a fragment of a program, or a Dynamic C keyword or phrase.
// IN-01...	Program comments are in normal Courier font.
<i>Italics</i>	Courier italics indicate that something should be typed instead of the italicized words (e.g., type a file name where <i>filename</i> is shown).
Edit	Bold sans serif font indicates a menu or menu selection.
...	An ellipsis indicates that (1) irrelevant program text is omitted for brevity, or that (2) the preceding program text may be repeated indefinitely.
[]	Square brackets in a C function's definition or program segment indicate that the enclosed directive is optional.
< >	Angle brackets are used to enclose classes of terms.
a b c	A vertical bar indicates that a choice should be made from among the items listed.

Pin Number 1

A black square indicates pin 1 of all headers.



Measurements

All diagram and graphic measurements are in inches followed by millimeters enclosed in parenthesis.



CHAPTER 1. OVERVIEW

Chapter 1 provides an overview and description of the LP3100 low-power controllers. The following sections are included.

- Features
- Standard Models
- Flexibility and Customization
- Subsystems
- Development and Evaluation Tools
- Software
- CE Compliance

The LP3100 is a low-power controller designed for use in low-power and battery-powered embedded applications.

Typical applications for the LP3100 include the following:

- Remote data acquisition
- Portable instrumentation
- Handheld equipment
- Solar powered instrumentation
- Battery powered systems

1.1 Features

The LP3100 has many features, including the following:

- Low-voltage design (3.3 V operation)
- Flexible power control system
- 6.144 MHz or 3.072 MHz system clock (software selectable)
- Flash EPROM (up to 512K)
- Static RAM with battery backup capability (up to 512K)
- 2 serial channels (RS-232 and RS-485)
- 4-channel 12-bit analog inputs with signal conditioning
- 4 digital inputs
- 8 digital outputs
- 8 configurable digital inputs/outputs (software configurable)
- Real-time clock
- Microprocessor supervisor IC
- Expansion bus (LPBus)
- Onboard diagnostic LED
- Dedicated programming port for Dynamic C



For details on customizing the LP3100, call a Z-World Sales Representative at (530) 757-3737.

1.2 Standard Models

The LP3100 Series currently has four members: LP3100, LP3110, LP3120, and LP3130. Table 1 lists the features of each model.

Table 1. LP3100 Series Model Features

Model	Features
LP3100	6.144 MHz clock, 512K flash EPROM, 128K SRAM, 2 RS-232 channels, RS-485 driver, 4-channel 12-bit A/D converter, 20 digital input/output lines, real-time clock, LPBus expansion port.
LP3110	LP3100 with 256K flash EPROM, 32K SRAM, no RS-485 driver.
LP3120	LP3100 with 256K flash EPROM, 32K SRAM, no RS-485 driver, no A/D converter.
LP3130	LP3100 with 256K flash EPROM, 32K SRAM, no RS-485 driver, no A/D converter, no real-time clock, no 5 V regulator, no LPBus expansion port.

1.3 Flexibility and Customization

The LP3100 was designed for application flexibility. Two levels of flexibility allow appropriate selection of the I/O for a specific application's needs:

- **Flexibility Level 1 — Out of the Box**

Jumper-configurable serial communication channels.

Jumper-selectable supply voltage to output latches allows control of the output voltage level (3.3 V or 5 V).

Bias and gain for conditioned analog input channel conditioning circuits configured with resistors.

Flexible digital input/output channels (eight channels) configurable as inputs or outputs under software control.

Software-controlled subsystem power (analog inputs, RS-232, memory and microprocessor, real-time clock) allows selective power-down for maximum power savings.

- **Flexibility Level 2 — Customization**

For quantity orders, Z-World can tailor the LP3100 to meet custom specifications.

Once the application prototype is defined, Z-World's automated surface-mount manufacturing facility can build the LP3100 with the exact hardware a specific application requires.

1.4 Subsystems

The LP3100 consists of several subsystems, including a microprocessor, memory, power control, digital inputs and outputs, analog inputs, serial communication channels, a real-time clock, and an LPBus expansion port. Figure 1 shows the board layout and Figure 2 illustrates the LP3100 subsystems.

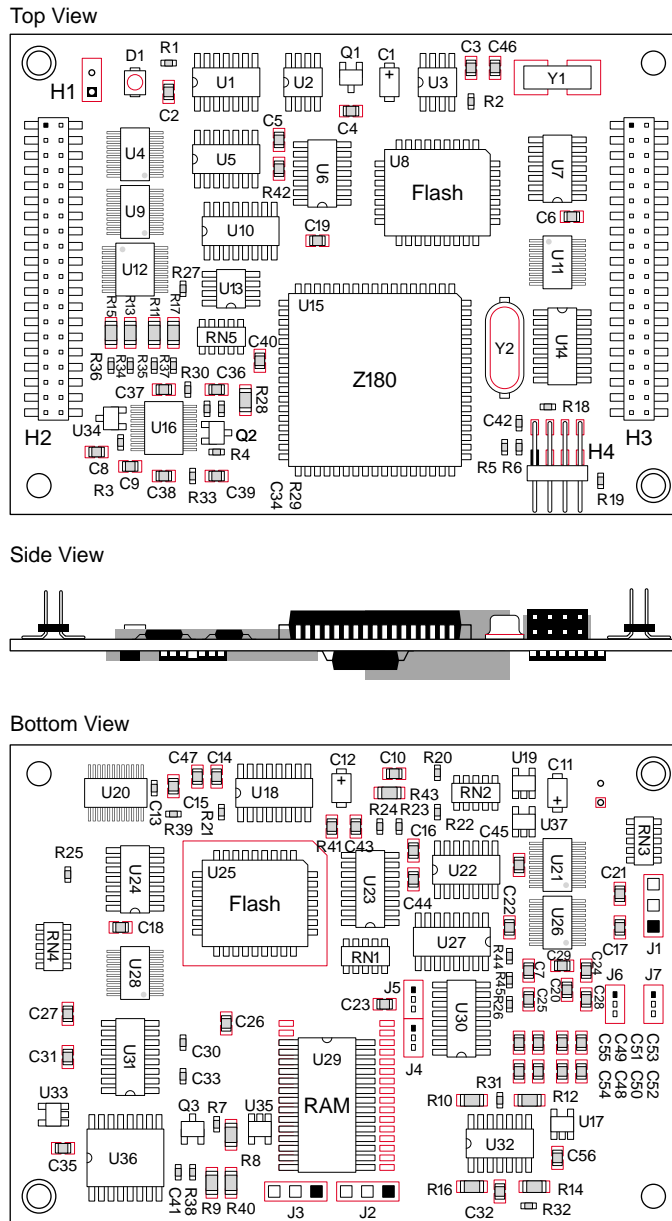


Figure 1. LP3100 Board Layout

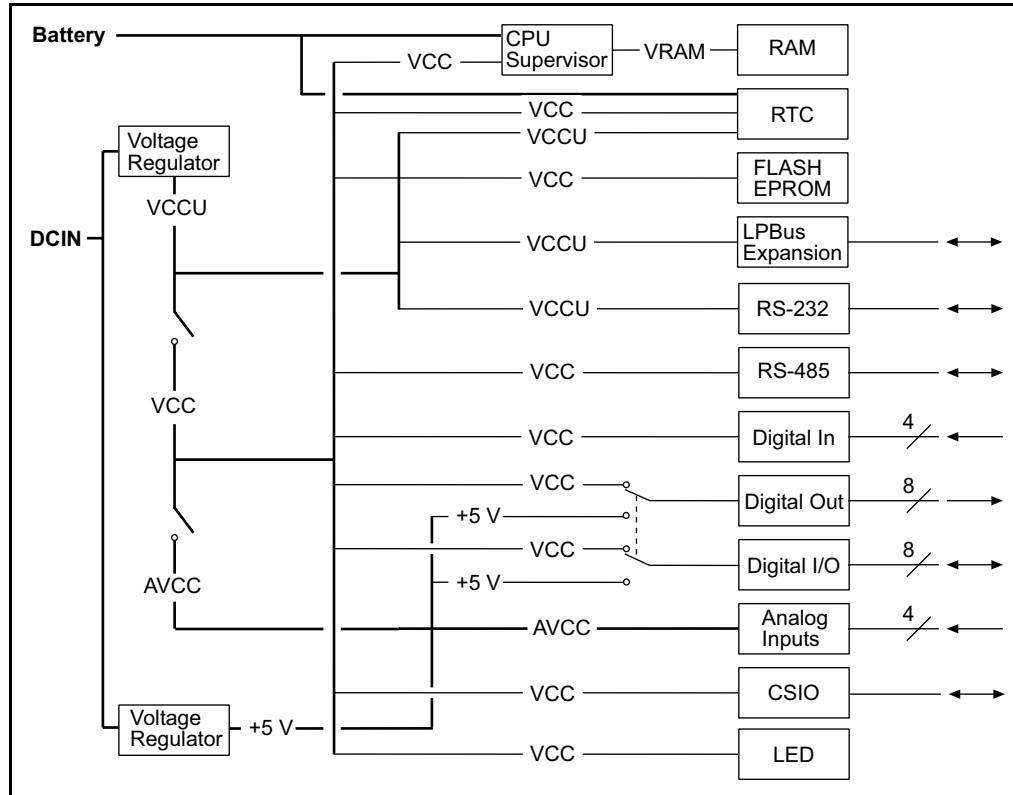


Figure 2. Subsystems Block Diagram

1.4.1 Microprocessor, Memory, and Support Circuits

The LP3100 uses a low-voltage microprocessor running at either 6.144 MHz or 3.072 MHz. Up to 512K of flash EPROM and 512K of static RAM are available. Up to 256K of the flash EPROM can be used for data-logging. A microprocessor supervisor IC provides protection against software bugs and glitches in addition to providing a battery-backup capability and brownout protection for the static RAM.

1.4.2 Power Control

Controlling the power supplies for the LP3100 subsystems enhances low-power operation. Most subsystems on the LP3100, including the microprocessor and memory, can be powered down or operated in low-power modes. The low-power mode virtually eliminates current draw for a subsystem that is not being used.

1.4.3 Digital Inputs/Outputs

The LP3100 series has the following three types of digital inputs/outputs.

1. **Digital Inputs** - Four dedicated digital input channels that include pull-up resistors. DIN0 can be used to wake the board from sleep mode.
2. **Digital Outputs** - Eight dedicated digital outputs that can be configured for either 3.3 V or 5 V output and can sink or source up to 8 mA.

3. **Configurable Digital Inputs/Outputs** - Eight digital channels that can be configured as either inputs or outputs with the same specifications as the dedicated inputs and outputs.

1.4.4 Analog Inputs

The LP3100 has four conditioned analog inputs. The inputs are multiplexed into a 12-bit ADC.

By default, the analog inputs have an input range of 0–10 volts. The inputs can be configured for almost any arbitrary range by replacing two configuration resistors.

1.4.5 Serial Channels

The LP3100 has two serial channels. One channel can be configured as either a 5-wire RS-232 or as a 3-wire RS-232. The other channel can be configured as a 3-wire RS-232 or as a 2-wire RS-485. The serial channels operate at speeds up to 38,400 bps using a 6.072 MHz system clock.

1.4.6 Real-Time Clock

Date and time-of-day information can be read from the real-time clock (RTC). The RTC provides the date and the day of the week in addition to the hour, minute, and second. The RTC can also be used to wake the LP3100 from sleep mode or to provide a periodic alarm.

1.4.7 LPBus

The LPBus provides an expansion capability for the LP3100. The LPBus provides address, data, and control signals for controlling add-on expansion modules.

1.4.8 LED

The onboard LED can be turned on or off under software control and can be used to indicate the status of the diagnostics or the system.

1.5 Development and Evaluation Tools

Z-World offers a Development Kit to help simplify prototyping with the LP3100.

1.5.1 Development Kit Packing List

- LP3100 Development Board
- LPBus Prototyping Board
- 2 × 20 character LCD with cable to connect to the LP3100 Development Board.
- Aluminum mounting plate
- Serial cable for software development
- DB25 to DB9 serial adapter
- 9 V DC power supply
- Battery holder and four AA size batteries
- Cable kit for 2 mm mass-termination connector
- Manual and schematics for all LP3100-related products


The Development Board, the LPBus Prototyping Board, the LCD, and the aluminum mounting plate are available for purchase separately. The optional SIB2 allows full use of all serial channels during development.



For information on the SIB2, or to order these items, call Z-World at (530) 757-3737.

1.6 Software

The LP3100 is easily programmed with Z-World's Dynamic C, an integrated development environment that includes an editor, optimizing C compiler, downloader, and debugger. Dynamic C provides a large number of easy-to-use software drivers for the LP3100.

 Refer to the Z-World catalog for more information about Dynamic C.

1.7 CE Compliance

The LP3100 has been tested by an approved competent body, and was found to be in conformity with applicable EN and equivalent standards. Note the following requirements for incorporating the LP3100 in your application to comply with CE requirements.



- The power supply provided with the Development Kit is for development purposes only. It is the customer's responsibility to provide a clean DC supply to the controller for all applications in end-products.
- The LP3100 has been tested to Light Industrial Immunity standards. Additional shielding or filtering may be required for an industrial environment.
- The LP3100 has been tested to EN55022 Class A emission standards. Additional shielding or filtering may be required to meet Class B emission standards.



Visit the "Technical Reference" pages of the Z-World Web site at <http://www.zworld.com> for more information on shielding and filtering.



CHAPTER 2. GETTING STARTED

Chapter 2 provides instructions for connecting an LP3100 series controller to a PC and running a sample program. The following sections are included:

- Operating Modes
- Connecting an LP3100 to a PC
- Establishing Communication with an LP3100
- Running a Sample Program

2.1 Operating Modes

The LP3100 has two operating modes, program mode and run mode.

- **Program Mode**

In Program Mode, the LP3100 runs under the control of Dynamic C. The LP3100 must be in program mode to compile and debug code on the LP3100.

In Program Mode, the LP3100 matches the baud rate of the PC's COM port up to 38,400 bps.

- **Run Mode**

In Run Mode, the LP3100 checks to see if the onboard memory contains a program. If a program exists, the LP3100 executes the program immediately after powerup.

In Run Mode, the LP3100 does not respond to Dynamic C running on the PC. Programs cannot be compiled or debugged while the LP3100 is in Run Mode.

The operating mode is determined by jumper settings on header H4. Header H4, illustrated in Figure 3, is the eight-pin right-angle header near header H3.

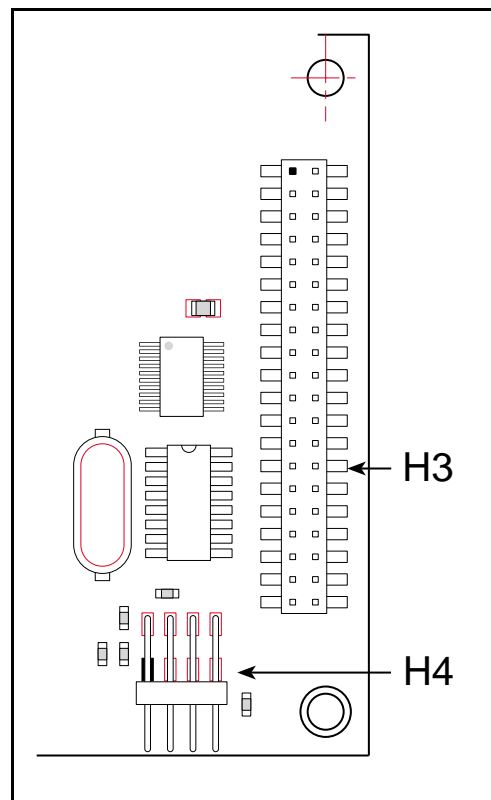
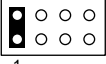
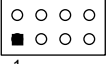


Figure 3. LP3100 Header H4 Location

Table 2 shows the LP3100 operating mode jumper settings.

Table 2. LP3100 Operating Mode Jumper Settings

Operating Mode	Header H4	Permissible Activities
Program Mode Connect pins 1 and 2	H4  1 Program Mode	Compile a program Run a program under debugger control Run a program without “polling.” See the Dynamic C manuals for a description of polling.
Run Mode No jumper connected	H4  1 Run Mode	Run program in memory

2.1.1 Changing Operating Modes

To change the operating mode, place or remove the jumper on H4. Then press the reset button on the LP3100 Development Board or cycle power to the LP3100 (remove and reapply power).

2.1.2 Using a SIB2

Since the SIB2 uses header H4 for communicating with the LP3100, it is not possible to change the operating mode via the jumper on header H4. Connecting the SIB2 to H4 automatically places the LP3100 into Program Mode. The SIB2 communicates with the PC at baud rates up to 57,600 bps. The SIB2 automatically sets its baud rate to match the PC’s serial port.

2.2 Connecting an LP3100 to a PC

The LP3100 can be programmed with a PC through an RS-232 port using the programming cable and Development Board provided in the Developer's Kit or by using a SIB2. Using the SIB2 frees all of the serial channels for the application during development.

The following steps describe how to connect the LP3100 to a PC.

1. Make sure that Dynamic C is installed on the PC as described in the *Dynamic C Technical Reference Manual*.
2. Make sure that power is not connected to either the LP3100 or to the Development Board.
3. Mate the LP3100 Development Board to the LP3100. Make sure that H7, on the bottom side of the Development Board, mates with H2 on the LP3100. Figure 4 illustrates the correct placement.

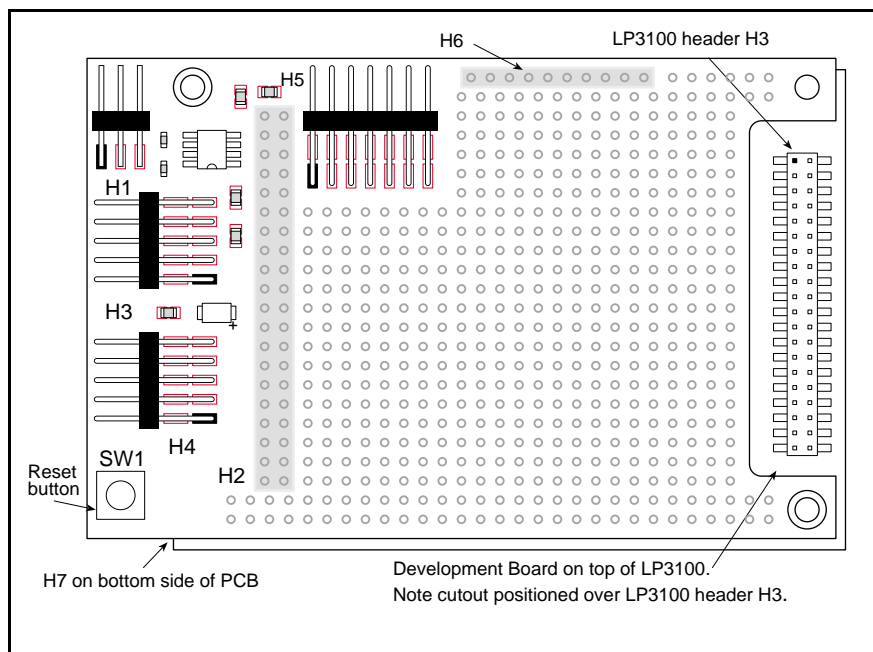


Figure 4. Development Board Placement



Note that the cutout in the Development Board should be positioned over the LP3100's header H3. Both the LP3100 and the Development Board may be damaged if the Development Board is installed incorrectly.

4. Establish a serial communication link.

Method 1—Directly to LP3100 RS-232 serial port. Place a jumper across pins 1 and 2 on header H4 to place the LP3100 in Program Mode. Connect the 10-conductor serial programming cable from header H4 on the LP3100 Development Board to an available COM port on the PC. Make sure that pin 1 on the ribbon cable connector (indicated by a small triangle on the connector) matches up with pin 1 (indicated by a small white circle near the corner of the connector) on header H4. Figure 5 illustrates the RS-232 programming connection.

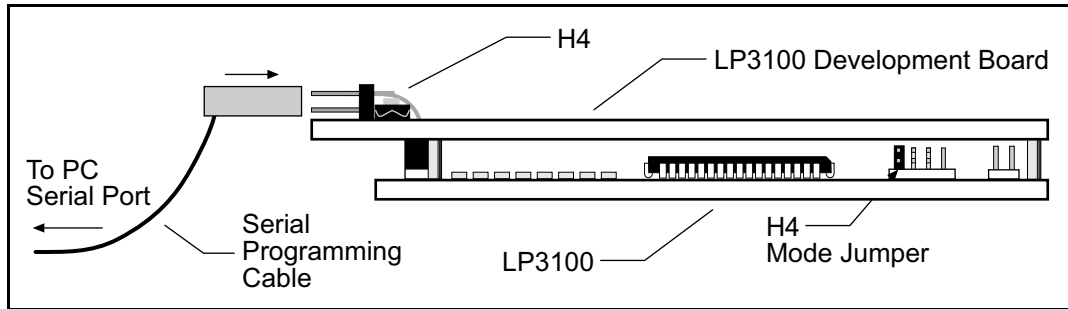


Figure 5. LP3100 RS-232 Programming Connection



Use only the supplied programming cables and adapters.

Method 2—Via SIB2. Connect the telephone-style cable supplied in the Developer's Kit to the RJ-12/DB-9 adapter. Connect the adapter to an available serial port on the PC. Connect the other end of the telephone-style cable to the SIB2.

Plug the SIB2's 8-pin connector onto header H4 on the LP3100 as illustrated in Figure 6. Make sure that pin 1 on the ribbon cable connector (on the striped side) matches up with pin 1 on H4 (indicated by a small white dot next to the header).

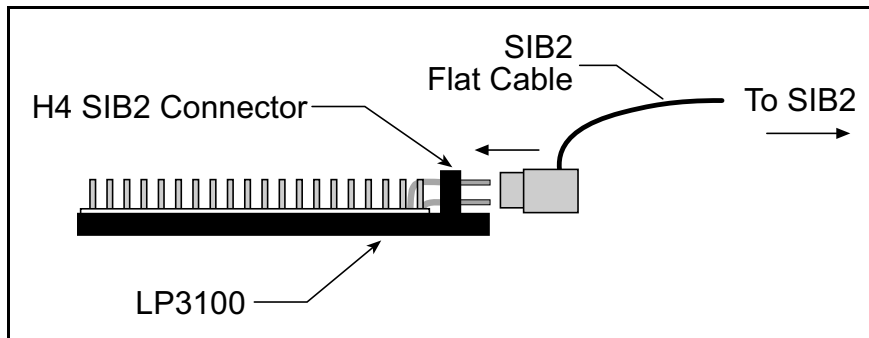


Figure 6. SIB2 Connection (End View)

Figure 7 illustrates a top view of the SIB2 connection.

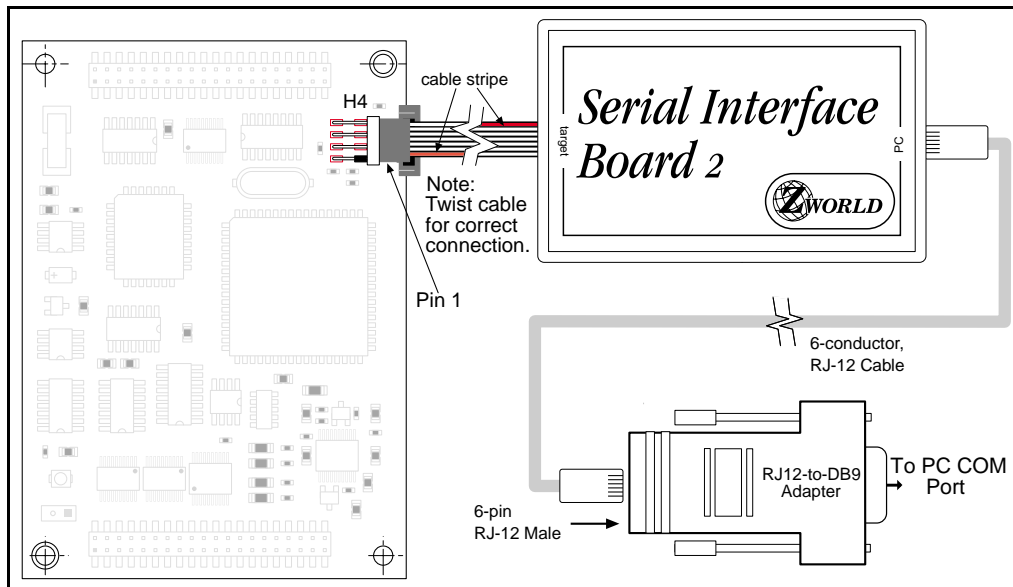


Figure 7. SIB2 Connection



Use only the 3.3 V SIB2 with the supplied adapter and programming cable.

Do not use the 5 V SIB with an LP3100.


5. Connect the 9 V DC power supply to the LP3100 Development Board power connector H1. The orientation of pin 1 is not important, but make sure that all three pins of H1 line up with the plug before connecting.
6. Plug the power supply into a wall socket.



Never connect or disconnect the SIB2 from the LP3100 while the LP3100 is powered. The SIB2 and the LP3100 may both be damaged.

2.3 Establishing Communication with an LP3100

1. Double-click the Dynamic C icon to start the software. Each time Dynamic C starts, it attempts communication with the LP3100.
2. If the communication attempt is successful, no error messages are displayed.

 If an error message such as **Target Not Responding** or **Communication Error** is shown, see Appendix A: “Troubleshooting.”

After making necessary changes to establish communication between the PC and the LP3100, use the Dynamic C shortcut **<Ctrl Y>** to reset the controller and initialize communication.

2.4 Running a Sample Program

The following steps compile and run a sample program:

1. Open the sample program **HELLO.C** located in the Dynamic C **SAMPLES\LP31XX** directory.

```
HELLO.C

/* Everybody's first program . . . */

void main( void ){
    printf("Hello world.\n");
    while (1) hitwd();
}
```

2. Compile the program by pressing **F3** or by choosing **Compile** from the **COMPILE** menu. Dynamic C compiles and downloads the program into the LP3100's flash memory.
3. During compilation, Dynamic C rapidly displays several messages in the compiling window. This condition is normal.

 If an error message such as **Target Not Responding** or **Communication Error** appears, see Appendix A: “Troubleshooting.”

4. Run the program by pressing **F9** or by choosing **Run** from the **RUN** Menu.
5. To stop program execution, press **<Ctrl Z>**. To restart program execution, press **F9**.



CHAPTER 3. SUBSYSTEMS

Chapter 3 discusses the LP3100 subsystems. The following sections are included:

- Subsystems Overview
- Microprocessor, Memory, and Support Circuits
- Power Control
- Digital Input/Output
- Analog Inputs
- Operation
- Serial Communication
- Real-Time Clock
- LPBus

3.1 Subsystems Overview

The LP3100 consists of several subsystems including a microprocessor, memory, power control, digital inputs and outputs, analog inputs, serial communication channels, a RTC, and an LPBus expansion port. Figure 8 illustrates the LP3100 subsystems.

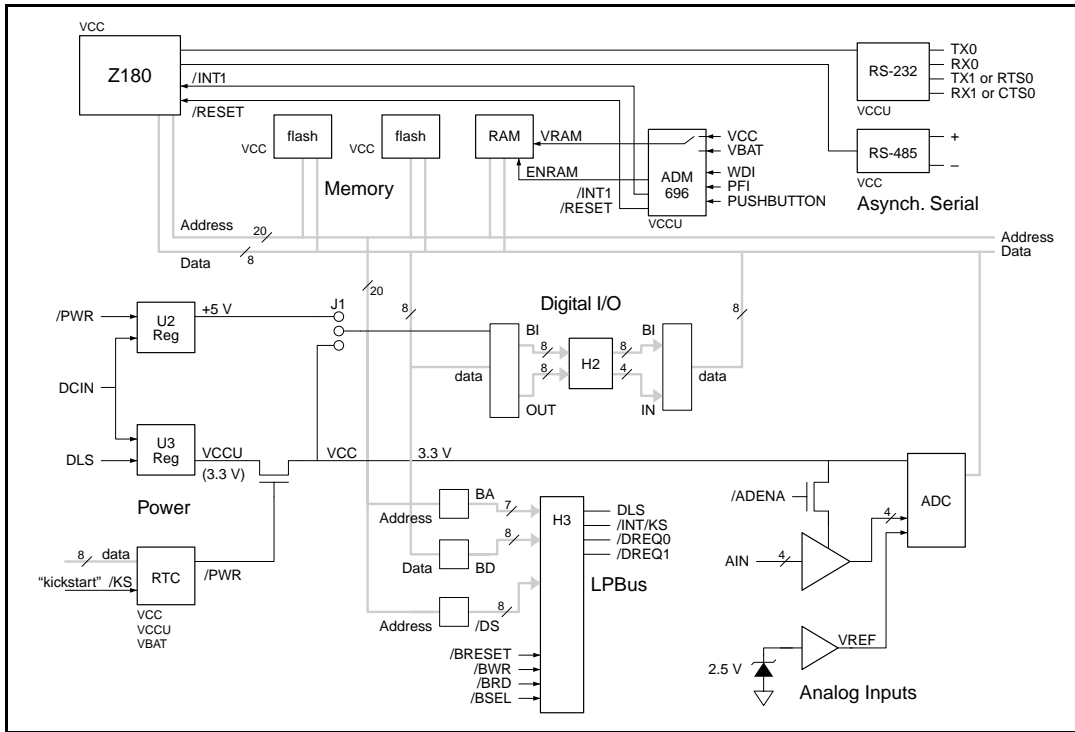


Figure 8. Subsystems Block Diagram

3.2 Microprocessor, Memory, and Support Circuits

The LP3100 was designed for low-voltage, low-power operation. The LP3100 uses an enhanced, low-voltage Zilog Z180 microprocessor designed for 3.3 V operation. The use of this microprocessor reduces power consumption and lowers the supply voltage requirements. The system clock speed can be set at 3.072 MHz or 6.144 MHz under software control. Lowering the system clock speed reduces the amount of power required by the LP3100.

The microprocessor also has several sleep modes that further decrease power requirements. The microprocessor has a Clocked Serial Input/Output (CSI/O) port for use with the SIB2 as a dedicated programming port. Using the SIB2 for development allows full use of both serial channels during development.

3.2.1 Microprocessor Supervisor

The LP3100 has an Analog Devices ADM696 microprocessor supervisor IC to provide reset control, memory protection, battery backup, and watchdog timer functions.

The watchdog timer is configured for a timeout period of 300 ms. The watchdog timer will reset the LP3100 if the watchdog is not “hit” by a software call function every 300 ms.

In addition to providing watchdog functions, the ADM696 monitors VCC. The ADM696 prevents spurious writes to the SRAM when VCC falls below a safe level.

The ADM696 also monitors the unregulated DCIN. As DCIN drops below a predefined threshold, the ADM696 generates a microprocessor interrupt to alert the microprocessor of imminent power failure. The microprocessor can then take precautions against data corruption.

3.2.2 Flash EPROM

The LP3100 has locations for two flash EPROMs. The flash EPROMs are either Atmel 29LV010 (128K x 8) or 29LV020 (256K x 8). U8, the flash EPROM on the top (connector) side, is soldered directly to the PCB and is used to store programs.

U25, the other flash EPROM, is located in a socket on the bottom side of the PCB. U25 is used for data-logging. U25 is socketed so that it can be easily replaced. When installing the flash EPROM, make sure that the flat corner of the IC fits into the flat corner of the socket as shown in Figure 9.

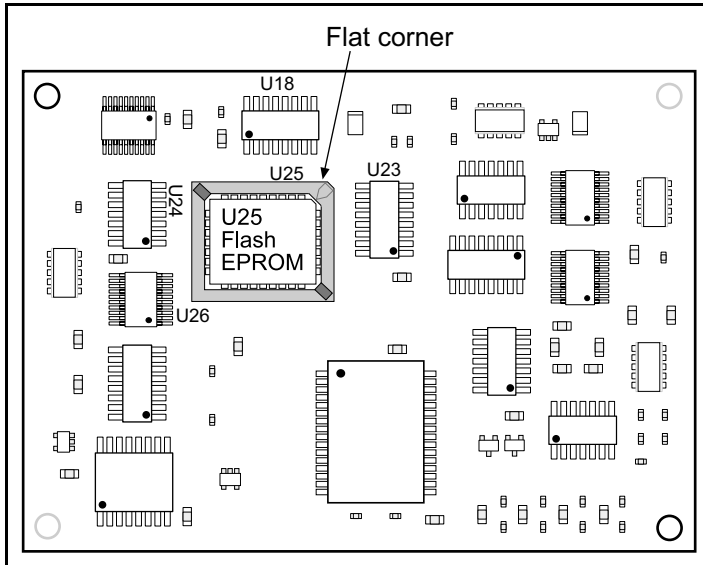


Figure 9. Flash EPROM Installation

The flash EPROM is specified for a maximum of 10,000 write cycles per sector. There is no limitation on reads. Evenly distributing the writes throughout the sectors will maximize the life of the flash EPROM.

Example: A data-logging application requires a data record 1K long with a 128K flash EPROM. Using a fixed location in flash EPROM, 1K records can be written 10,000 times before reaching the sector write limit. However, if the location of the record is alternated in an even distribution through the flash EPROM, then the record could be written 10,000 times in 128 different locations, allowing 1,280,000 writes.

3.3 Power Control

There are several ways to reduce power consumption by the LP3100. Power supplies, system clock speed, sleep modes, and shutdown modes for LP3100 peripherals and subsystems are all under software control, allowing flexible control of power consumption.

3.3.1 Power Supplies

Table 3 lists the subsystems and the name of each subsystem's power supply. Figure 10 illustrates the power supplies.

Table 3. LP3100 Power Supplies

Subsystem	Power Supply Name
Power control, LPBus, RS-232, RTC	VCCU
Microprocessor, MPU supervisor, RS-485, flash EPROM, RTC, support circuits	VCC
Analog-to-Digital converter	ADVCC
Analog input	AVCC
5 V for external digital interface	+5 V
SRAM	VRAM
SRAM and RTC backup source	VBAK

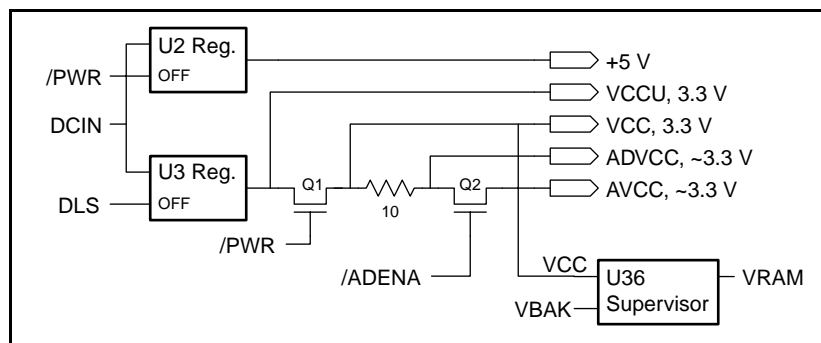


Figure 10. Power Supplies

- VCCU is the primary power supply for the LP3100. Voltage regulator U3 provides VCCU from the unregulated DCIN supply. In addition to supplying current for VCC and ADVCC (via VCC), VCCU powers the LPBus, the RS-232 interface, and the RTC. VCCU can be turned on and off by a device on the LPBus via the DLS signal. All circuits on the LP3100, except the RTC and SRAM, are unpowered while VCCU is off (provided that backup power is supplied to the LP3100).

- VCC is a secondary power supply that provides power to most of the LP3100 circuits including the microprocessor, the microprocessor supervisor, the flash EPROM, and the RTC. VCC is derived from VCCU. A software-controlled switch connects or disconnects the VCC supply from VCCU. VCC can be turned off under software control. With VCCU is still active, restore VCC by any of the following actions:
 - The RTC can generate an alarm (VCCU must still be providing power to the RTC) and restore VCC and reset the Z180.
 - VCCU can be shut down and brought back up. This cycling of VCCU will bring VCC up and reset the Z180.
 - The LPBus signal /INT/KS can be pulled low. This will restore VCC and reset the Z180.
 - If the /EN_DKS bit is asserted 0, a logic low on DIN0 will restore VCC and reset the Z180.



The microprocessor stops running once VCCU or VCC is turned off.

- AVCC is the power supply for the A/D converter and signal conditioning circuits. AVCC is derived from VCC. A software-controlled switch connects or disconnects the ADVCC supply from VCC.
- The +5 V supply can be used to supply 5 V to the external digital interface. Voltage regulator U2 provides +5 V DC from the unregulated DCIN supply.
- VRAM is the SRAM power supply. During normal operation, the microprocessor supervisor IC provides power to the SRAM from VCC. If VCC is turned off, the microprocessor supervisor powers the SRAM from VBAK. If there is no VBAK supply voltage, the contents of the SRAM are lost when VCC is turned off.
- VBAK is the backup voltage for the SRAM and RTC. The VBAK supply voltage should be approximately 3 V (typically a battery or a supercapacitor). VBAK supplies power to the SRAM when VCC is turned off and to the RTC when VCCU is turned off.

3.3.2 Power Supply Control

The LP3100 Series is designed to consume less power than other Z-World controllers. In addition to a lower power consumption during normal operation, an LP3100 series controller also permits further power saving through the following special features:

- **Shutdown Mode** — VCC is turned off. ADVCC is also off since it is supplied by VCC. VCCU is still active. The current draw in this mode varies from 300 μ A to 400 μ A. The LP3100 can be brought out of the shutdown mode by an external interrupt event (/RESET, DIN0, LPBus, RTC) or by cycling power.
- **Alternate External Supply** — Allows the LP3100 to be powered by an external power supply connected to the LPBus. The VCCU regulator is controlled via the DLS signal on the LPBus. A high on DLS shuts down the VCCU regulator.
- **Digital Output Supply** — The digital outputs can be powered by either a 3.3 V or a 5 V supply. The +5 V supply for the digital outputs consumes an additional 2 mA. Use the +5 V supply only if external devices require a 5 V interface. Jumper J1 selects the digital output voltage supply. The +5 V supply can be turned on or off under software control.
- **Analog Input Supply** — The analog input supply voltage, AVCC, can be turned off if the analog inputs are not being used. The analog input section consumes approximately 1.5 mA. The analog input supply can be turned on or off under software control.

When deciding which power-saving features to use, consider the following important criteria.

- **Overall power consumption requirement** — This average power consumption requirement is often related to the power supply constraints, such as the rating of batteries. If the power supply system can constantly supply 25 mA, there is no need for any power conservation methods.
- **Power-up/resumption time requirement** — Some applications require the controller to respond quickly to external or internal events, while others have more relaxed requirements. Any requirement with a reaction time less than 300 μ s rules out shutting down VCC.
- **Power-up/resumption source options** — Most power conservation schemes discussed in this manual require the suspension of execution (the only exception is shutting down individual components). Different power conservation schemes provide different options to resume execution. The application engineer should make sure the options match the requirements of the application. For example, if the application is to be “awakened” by serial communication, the standby mode is not a power conservation option.
- **Application-specific requirements** — The application may have requirements that do not seem related to power conservation schemes. For example, if the application needs to maintain timing accuracy of less than one second, shutting down VCC will no longer be a useable alternative. The application engineer must evaluate the actual resources on an LP3100 that are required by the application, then determine if the chosen power conservation scheme may affect the required resources.

- **Application complexity** - Some power conservation schemes, such as shutting down VCC and the standby mode, present more issues for software design and development. If there is more than one power conservation option available, the application engineer should balance between power conservation and software complexity.



If you have further questions about special low-power considerations, contact Z-World Technical Support at (530)757-3737.

3.3.3 Shutting Down VCC

LP3100 series controllers are equipped with VCCU (unconditional VCC), VCC (conditional VCC), and provision for a back-up battery. Some chips are powered by VCCU, but most are powered by VCC. The RTC can resume VCC as long as VCCU is available. More information is provided about the RTC features in a following subsection.

The most important advantage of shutting down VCC is that it minimizes power consumption. However, VCC can only be resumed from two software configurable sources. VCC can also be resumed by system reset and VCCU power cycling. The two software configurable VCC resumption sources are time-based wake up and kick start. The time-based wake up feature relies on a pre-set date/time in the RTC to restart VCC. The kick start approach relies on a negative edge on the kick start pin (DIN0) to resume VCC.

When VCC is resumed, the stack is rewound and control is passed to the main function. If it is necessary to resume the program thread, the application engineer must not assume the stack is the same as before VCC is shut down. The granularity of program thread resumption is coarse.

Furthermore, when VCC resumes, the system requires some overhead time even before control is passed to the main function. Therefore, shutting down VCC is not suitable for applications that require fast recovery time.

In addition, the RTC can wake up VCC at a maximum frequency of 1 Hz. Where periodic power-up of more than 1 Hz is needed, the application must depend on external circuitry that pulses the kickstart line (DIN0). If external circuitry is not available, do not shut down VCC.

3.3.4 Clock Speed

The LP3100 can switch the system clock speed between 3.072 MHz and 6.144 MHz. A lower clock speeds reduces computing power, but also reduces power consumption. The clock speed can be changed on the fly without affecting the state of the system. The 6.144 MHz clock speed is selected for computation-intensive operation. The system clock can be slowed to 3.072 MHz when the additional speed is not needed. The LP3100 requires an additional 4 mA when running at 6.144 MHz.

Changing Clock Speed

The application can change the clock speed to reduce power consumption. However, changing the clock speed has many side effects. All peripherals that depend on the clock speed will be affected by a change of clock speed. In particular, the Z180 PRTs (programmable reloadable timers), ASCIs (asynchronous serial communication interfaces) and CSI/O devices will be affected.

Dynamic C (5.25 and later versions) provides support to change the clock speed. The application can call `lp31clk3MHz()` to change the clock speed to 3 MHz, or call `lp31clk6MHz()` to change the clock speed to 6 MHz. When developing software via communication port 0 (instead of the SIB2 port), these functions help to adjust communication port 0 so you can continue to debug the software after changing the clock speed. If you plan to change clock speed in the application, it is best to use the SIB2 or use communication port 0 at 19,200 bps. Note that these functions do not adjust the PRT or ASCIs not used for development purposes. If the application uses a PRT or an ASCII port, the application program must adjust the PRT or ASCII port to maintain functionality after clock speed change.



Changing system clock speed can affect many system resources including timers, serial channels, DMA channels, and LPBus. Be careful to consider all of the possible effects of changing the system clock speed when developing an application.

3.3.5 Microprocessor Operating Modes and Shutdown Mode

The microprocessor used on the LP3100 has several low-power modes. Each mode has a unique set of tradeoffs. Following is a brief description of each mode:

- **Standby Mode** – `void sysStandby()`

The system clock is shut down. Uses minimum power consumption relative to the other two modes. On-chip peripherals are also stopped in this mode. The only methods to get out of standby mode are a system reset, power cycling, and external interrupts (INT0, INT1, INT2 and NMI). The longest recovery time and the greatest power savings occur when the MPU is in Standby Mode. Since on-chip devices are not running, only external sources (/RESET, DIN0, LPBus, RTC) can bring the MPU out of Standby Mode. The recovery time of 64 cycles is greater than other modes because the oscillator needs to be restarted.

- **Sleep Mode** – `void sysSleep()`

The system clock continues to operate, but is blocked from the CPU core and DMA channels. All other on-chip peripherals continue to operate. System reset, power cycling and any interrupt resumes execution immediately. Current draw is less than Halt Mode and greater than Standby Mode. Recovery time is longer than Halt Mode and shorter than Standby Mode. The LP3100 will leave Sleep Mode in response to a logic low signal on the /RESET line, an interrupt from an internal source, or an interrupt from an external source (DIN0, LPBus, RTC).

- **Halt Mode** – `void sysHalt()`

The system clock, the CPU core, and DMA channels continue to operate. However, the CPU does not fetch the next instruction. This mode does not save much power since it draws the most current, but it has the shortest recovery time. The LP3100 will leave Halt Mode in response to a logic low signal on the /RESET line, an interrupt from an internal source, or an interrupt from an external source (DIN0, LPBus, RTC).

- **Shutdown Mode** - VCC is turned off. ADVCC is also off since it is supplied by VCC. VCCU is still active. The current draw in this mode varies from 300 μ A to 400 μ A. The LP3100 can be brought out of the shutdown mode by an external interrupt event (/RESET, DIN0, LPBus, RTC) or by cycling power.

When the CPU is interrupted in any of the low-power operating modes (except the shutdown mode), the CPU executes the appropriate interrupt handling routine, then returns to the instruction after the sleep or halt instruction. In other words, execution of the main thread is resumed after the interrupt handler returns.

If the Standby Mode is used, the on-chip timer can no longer keep track of the time in `SEC_TIMER` and `MS_TIMER`. The application should reinitialize `SEC_TIMER` immediately after `sysStandby()` if `SEC_TIMER` is used either explicitly or implicitly.

The following lines of code define how to reinstate `SEC_TIMER`.

```
struct tm t;
sysStandby();           // enter standby mode
                        // some external interrupt
                        // resumes execution
tm_rd(&t);              // get the real time
SEC_TIMER = mktime(&t); // reinitialize SEC_TIMER
```

It is impossible to resynchronize `MS_TIMER` when execution resumes after the Standby Mode is engaged. Furthermore, the potential jump of `SEC_TIMER` and the discontinuity of `MS_TIMER` may cause concurrent timing logic in the application to fail. The programmer should make sure that all concurrent threads are prepared for entering the Standby Mode.

One interrupt source (an external interrupt to the Z180) to resume execution from the Standby Mode is the RTC. The RTC can generate interrupts (via INT0) at up to 8 kHz using the periodic interrupt feature, or down to once a month using the alarm feature.

When using DIN0 to bring the microprocessor out of the Halt, Sleep, Standby, or Shutdown mode, set the `/EN_DKS` bit and apply a logic low signal to DIN0.



The Standby, Sleep, and Halt modes are supported by the Zilog enhanced Z180 (SL1919) processor. Refer to the *Zilog Z180 Databook* for more details about these modes.

3.3.6 Component Shutdown

Some components on an LP3100 series controller can be shut down by software. All shutdown controls are mapped to the digital output map (use `eioBrdDO` to control the shutdown features). Table 4 lists each shutdown feature, the `eioBrdDO` map, and actual I/O address.

Table 4. Shutfown Control Map

Feature ("/" signals are active low)		eioBrdDO Index	Actual Port No.
/232TEN	RS-232 transmit driver enable	18	0x4082
232EN	RS-232 receive driver enable	19	0x4083
/CTSSEN	RS-232 CTS enable	20	0x4084
LED	LED enable	24	0x40A0
485TE	RS-485 transmit enable	25	0x40A1
/485RE	RS-485 receive enable	26	0x40A2
DOE	Digital output enable	28	0x40A4
/ADENA	ADC reference voltage eneable	29	0x40A5

To assert active low signals, pass zero for the state argument in the call to `eioBrdDO`, or pass zero to the output byte argument in `outport`. Pass non-zero for the state argument in the call to `eioBrdDO` to turn off an active low signal, or pass 1 to the output byte argument in `outport`. For example, to turn off the ADC reference voltage, call `eioBrdDO(29,1)` or `outport(0x40a5,1)`. Note that if you use the `outport` approach, bit 0 and only bit 0 of the output byte is used to control the feature. In other words, `outport(0x40a5,2)` enables the ADC reference voltage.

The RS-232 transceiver has two software-selectable low-power modes.

- **Driver Disable Mode** — Current draw is reduced by 70 μ A from normal operating mode. The charge pumps are turned off and the driver outputs are placed in a high-impedance mode. Both receivers are still active.
- **Shutdown Mode** — Current draw is about 200 μ A. The entire transceiver is turned off.

3.4 Digital Input/Output

The LP3100 has 20 digital input and output channels. Four channels are dedicated inputs, eight channels are dedicated outputs, and eight channels can be configured as either inputs or outputs. All digital input/output channels are brought out on header H2.

3.4.1 Digital I/O Operating Modes and Configuration

The eight dedicated digital outputs can be configured for 5 V or 3 V output signal levels. This configuration applies to all eight channels. The output voltage is jumper configurable.

The eight digital input/output channels can be configured as either digital inputs or digital outputs. If the input/output channels are configured as outputs, they can also be configured for 5 V or 3.3 V operation. The input/output mode is software configurable and can be changed at any time. The output voltage is jumper configurable.



The 3.3 V or 5 V configuration applies to the DOUT and DINOUT (when in output mode) signals. All digital outputs are either 3.3 V or 5 V.

3.4.2 Digital Inputs

There are four dedicated digital inputs available on header H2: DIN0, DIN1, DIN2, and DIN3. DIN0 can also be used to wake the system from a powerdown condition. Table 5 lists the H2 pin number and software channel for the digital input channels.

Table 5. Digital Inputs

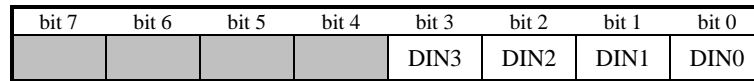
Pin No.	Signal	Dynamic C Channel No.
22	DIN0	8
23	DIN1	9
24	DIN2	10
25	DIN3	11

Each of the four digital inputs has a 10 k Ω pull-up resistor. DIN1 through DIN3 are pulled up to VCC. DIN0 is pulled up to VCCU, allowing it to remain pulled high even if VCC is turned off. This allows kick start resetting of an LP3100 when VCC is turned off.

The four digital inputs are connected to a 74VHC541 buffer. The 74VHC541 is a high-speed, low-power CMOS device connected to a 3.3 V supply rail.

The buffer will tolerate a maximum input voltage of 7 V. Thus, reading 5 V logic signals requires no additional interfacing. The minimum input high voltage is 2.3 V. The maximum input low voltage is 1.0 V.

The buffer is mapped into the Z180 I/O space at 4040H. DIN0 to DIN3 are mapped to the low order nibble. DIN0 is the LSB.



The inputs can be read using two methods: (1) Z-World drivers or (2) the `inport` function. Z-World drivers provide the easiest method of reading the inputs and will help make software compatible with future versions of the LP3100 as well as other Z-World products.

Program 3-1 illustrates the use of Dynamic C functions for reading the digital inputs.

Program 3-1. Reading Digital Inputs with Functions

```

/* Read the Digital inputs using Z-World drivers*/
/* note this is a bitwise access to the port */

#include eziolp31.lib          // use the correct library
#define INPUTCHAN 8          // Read DIN0
void main( void ){
    auto int result;
    eioBrdInit(0);          // initialize the board
    while(1) {
        hitwd();
        result = eioBrdDI( INPUTCHAN );
        switch (result) {
            case -1: printf("digital input channel doesn't
                           exist!\n", INPUTCHAN);
                break;
            case 0: printf("digital input channel %d reads
                           low\n", INPUTCHAN);
                break;
            case 1: printf("digital input channel %d reads
                           high\n", INPUTCHAN);
                break;
        }
    }
}

```



printf statements must be on one continuous line in an executable program. In this sample program, the **printf** statement is shown as more than one line only for display.

Program 3-2 shows how to use the `inport` function for reading the digital inputs.

Program 3-2. Reading Digital Inputs with inport Function

```

/* Read the Digital inputs using inport()*/

void main (void) {
    while(1) {
        printf("\nDIN=%X", (int)inport(0x4040));
        hitwd();
    }
}

```

3.4.3 Digital Outputs

There are eight dedicated digital outputs, DOUT0 through DOUT7. These eight outputs are available on header H2. Table 6 lists the H2 pin numbers and software channels for the digital outputs.

Table 6. Digital Outputs

Pin No.	Signal	Dynamic C Channel No.
14	DOUT0	8
15	DOUT1	9
16	DOUT2	10
17	DOUT3	11
18	DOUT4	12
19	DOUT5	13
20	DOUT6	14
21	DOUT7	15

The eight digital outputs are connected to a 74VHCT574 latch. The 74VHCT574 is a high-speed low-power CMOS device that will sink or source up to 8 mA. The output voltage will decrease as the output current increases. Table 7 shows the minimum output high voltage, maximum output low voltage, and output current for both 3.3 V and 5 V supplies.

Table 7. Digital Output Specifications

Parameter	3.3 V Supply	5.0 V Supply
Minimum High-Voltage Output*	3.2 V	4.9 V
Maximum Low-Voltage Output<Superscript>*	0.1 V	0.1 V
Maximum Output Current (sinking or sourcing)	4 mA	8 mA

* With loads < 50 μA

A jumper on header J1 determines whether the output latch is supplied by the 3.3 V or the 5 V supply. J1 controls the supply voltage of both the DOUT latch and the DINOUT latch. Use the supply voltage that is appropriate for the devices connected to the digital outputs.

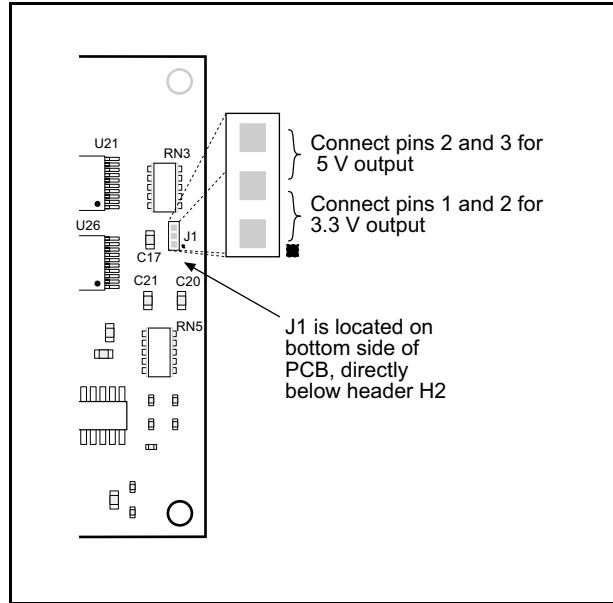


Figure 11. J1 Jumper Settings



When changing the J1 jumper setting, make sure that only pins 1 and 2 *or* pins 2 and 3 are connected. If pin 1, pin 2, and pin 3 are all connected, the LP3100 may be damaged.

The DOUT latch is mapped to the Z180 I/O space at 4060H. DOUT0 is the least-significant bit.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
DOUT7	DOUT6	DOUT5	DOUT4	DOUT3	DOUT2	DOUT1	DOUT0


The outputs can be written using two methods: Z-World drivers or the `outport` function. Z-World drivers provide the easiest method of writing to the outputs, and will help make the software more compatible with future versions of the LP3100 as well as other Z-World products.

The code segment in Program 3-3 illustrates the use of Dynamic C functions to write digital outputs. Program 3-4 shows how to use the **outputport** function to read digital inputs.

Program 3-3. Writing Digital Outputs with Z-World Drivers

```
/* Write a Digital output using Z-World drivers*/
/* note this is a bitwise operation */

#include eziolp31.lib
#define OUTPUTCHAN 8          // Use DOUT0
void main( void ){
    auto int result;
    auto unsigned char value;
    while(1) {
        hitwd();
        eioBrdInit(0);
        value = 0;           // Write a zero to output channel
        result = eioBrdDO(OUTPUTCHAN, value);
        if (result == -1)
            printf("digital output channel %d doesn't
                    exist!\n", INPUTCHAN);
    }
}
```

 **printf** statements must be on one continuous line in an executable program. In this sample program, the **printf** statement is shown as more than one line only for display.

Program 3-4. Read Digital Inputs Using outputport

```
/* Read the Digital inputs using outputport()*/

void main( void ){
    unsigned char c;
    c=0;
    while(1) {
        outputport(0x04060, c++);
        printf("\nwrote %d", (int) c);
        hitwd();
    }
}
```

3.4.4 Digital Inputs/Outputs

There are eight combination digital input/output channels, DINOUT0 through DINOUT7. The digital inputs/outputs are brought out on header H2. Table 8 lists the H2 pins and software channel numbers for the digital inputs/outputs.

Table 8. Digital Input/Output Channel Numbers

Pin No.	Signal	Dynamic C Channel No.
6	DINOUT0	0
7	DINOUT1	1
8	DINOUT2	2
9	DINOUT3	3
10	DINOUT4	4
11	DINOUT5	5
12	DINOUT6	6
13	DINOUT7	7

The eight DINOUT pins are connected to a 74VHC541 buffer for the inputs and a 74VHCT574 latch for the outputs. The buffers and latches are high-speed, low-power CMOS devices. Each digital input/output has a 10 k Ω pull-up resistor to VCC. The input buffer will accept input voltages up to 7 V. Thus, reading 5 V logic signals requires no additional interfacing. The output latches source or sink up to 8 mA with a 5 V supply and up to 4 mA with a 3.3 V supply.

Table 9 lists input and output logic threshold values as well as the output current. Output voltages are shown for both 3.3 V and 5 V supplies. Output voltages listed in Table 9 are for output currents of 50 μ A or less. The output voltages will drop as the output current increases..

Table 9. Digital I/O Channel Specifications

Parameter	3.3 V Supply	5.0 V Supply
Input Minimum High Voltage	2.3 V	2.3 V
Input Maximum Low Voltage	1.0 V	1.0 V
Output Minimum High Voltage	3.2 V	4.9 V
Output Maximum Low Voltage	0.1 V	0.1 V
Maximum Current (sinking or sourcing)	4 mA	8 mA

The input buffer is mapped into the Z180 I/O space at 4020H. DINOUT0 is the least-significant bit. The output latch is mapped into the I/O space at 4040H. DINOUT0 is the least-significant bit.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
DINOUT 7	DINOUT 6	DINOUT 5	DINOUT 4	DINOUT 3	DINOUT 2	DINOUT 1	DINOUT 0

During power-up or reset, the bit-addressable latch, U30, resets, and the output latch (U4) outputs are enabled momentarily even if DINOUT0–DINOUT7 will be used as inputs.

The DINOUT channels can be used as outputs and inputs simultaneously. Excessive loading on the output lines can be detected by reading an input while an output is active.

Configuring as Outputs

1. Enable the output latch by writing a 0 to the bit-addressable latch, U30, at address 40A4H (/DOE).
2. The status of the output latch signals is indeterminate at startup. Write an initialization routine to place the outputs in the desired default state.



Do not connect DINOUT0–DINOUT7 to inputs or machinery that could be damaged or cause injury because the output state cannot be guaranteed on power-up or reset.

The outputs can be written by using Z-World drivers or the **outport** function and read by using the drivers or the **inport** function.

Program 3-5 illustrates the use of **inport** function for writing the digital input/outputs.

Program 3-5. Exercising DINOUT Channel Using inport

```

/* Read the Digital inputs using inport()*/
void main( void ){
    unsigned char c;
    c=0;
    outport( 0x40A4, 0 );    /* enable output latch */

    /* outport( 0x40A4, 1 ) is the complementary function used to
    put output latch outputs into a high Z mode. */

    while(1) {
        outport( 0x04040, c++);
        printf("\nwrote %d", (int) inport( 0x04020 );
        hitwd();
    }
}

```

Program 3-6 illustrates the use of the Z-World driver for the DINOUT lines.

Program 3-6: Exercising DINOUT CHannel with Z-World Drivers

```
/* Use the DINOUT lines as outputs with read back verification. Interface using Z-World drivers*/

#include eziolp31.lib

#define OUTPUTCHAN 0 // DINOUT0
#define INPUTCHAN 0 // DINOUT0

void main( void ){
    auto unsigned char value;
    auto int result;
    eioBrdInit(0); // initialize the board
    while(1) {
        hitwd();
        eioBrdInit(0);
        value = 0; // Write a zero to output channel
        result = eioBrdDO(OUTPUTCHAN, value);
        if (result == -1)
            printf("digital output channel %d doesn't
                    exist!\n", INPUTCHAN);
        result = eioBrdDI( INPUTCHAN );
        switch (result) {
            case -1: printf("digital input channel %d
                            doesn't exist!\n", INPUTCHAN);
                    break;
            case 0: printf("digital input channel %d reads
                            low\n", INPUTCHAN);
                    break;
            case 1: printf("digital input channel %d reads
                            high/n", INPUTCHAN);
                    break;
        }
    }
}
```



printf statements must be on one continuous line in an executable program. In this sample program, the **printf** statement is shown as more than one line only for display.

Configuring as Inputs

Z-World recommends removing the output latch, U4, when DINOUT0–DINOUT7 are to be used as inputs. If U4 is not removed, place the output latch into a high-impedance state by writing a 1 to the bit-addressable latch, U30, at address 40A4H (/DOE).



If U4 is not removed, DINOUT0–DINOUT7 become enabled as outputs at power-up or reset before the initialization routine configures them. Exercise care when connecting DINOUT0–DINOUT7 to outputs or machinery that could be damaged or cause injury when momentarily connected to an output signal.

3.5 Analog Inputs

The LP3100 has four analog input channels that are brought out on header H2. The analog inputs are designated AIN0 to AIN3, and are listed in Table 10.

Table 10. Analog Input Channel Numbers

Pin No.	Signal	Dynamic C Channel No.
35	AIN0	0
36	AIN1	1
37	AIN2	2
38	AIN3	3

The analog input channels can be configured for a wide range of input voltages. The default input range of the analog inputs is 0 V to 10 V. The analog inputs can be configured for almost any input range by replacing two resistors in the input amplifier circuit.

Each analog channel consists of an inverting amplifier referenced to a user-defined offset voltage, as shown in Figure 12.

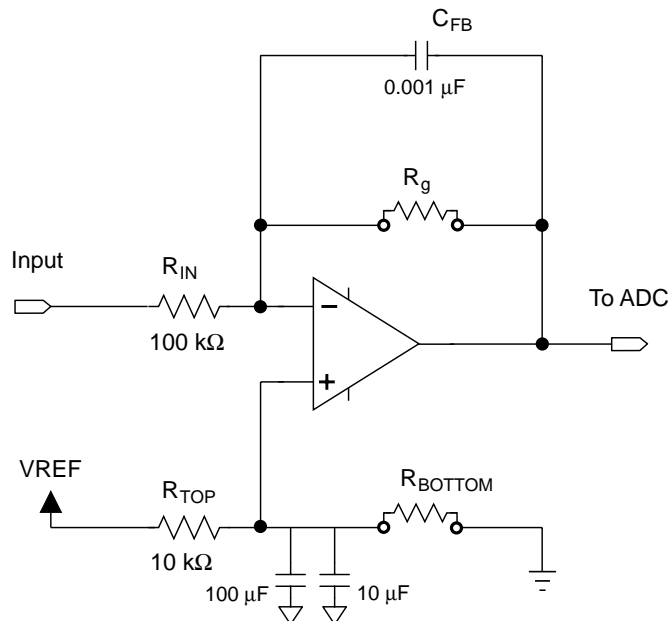


Figure 12. Analog Input Amplifier

Table 11 lists the analog input amplifier components.

Table 11. Analog Input Component References

Channel	R _{IN}	R _g	R _{TOP}	R _{BOTTOM}	Capacitor
AIN0	R30	R10	R34	R11	C36
AIN1	R31	R12	R35	R13	C37
AIN2	R32	R14	R36	R15	C38
AIN3	R33	R16	R37	R17	C39

Changing the values of R_g and R_{bottom} sets the gain and offset of the channel. Figure 13 illustrates the locations for R_g and R_{bottom} on an LP3100 printed circuit board.

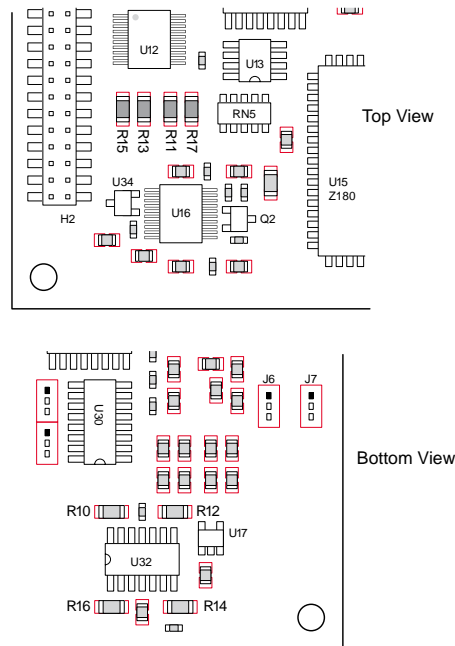


Figure 13. Location of R_g and R_{bottom} Resistors

Table 12 lists resistor values for some common analog input voltage ranges.

Table 12. Resistor Values for Common Input Ranges

Channel Input Range (V)	R_g (k Ω)	R_{BOTTOM} (k Ω)
-10.0 to +10.0	11.8	8.06
-5.0 to +5.0	23.2	6.65
-2.5 to +2.5	47.5	4.99
-2.0 to +2.0	59.0	4.53
-1.0 to +1.0	118	2.87
-0.5 to +0.5	237	1.69
-0.25 to +0.25	464	0.953
-0.1 to +0.1	1180	0.392
* 0 to +10.0	23.7	39.2
0 to +5.0	46.4	19.6
0 to 2.5	93.1	10.0
0 to 1.0	226	4.02
+1.0 to +2.0	237	13.3
+2.0 to +7.0	47.5	140.0
+0.020 to +0.100	3010	0.402

* Default values

If the application requires an input voltage range not shown in Table 12, the correct values of R_g and R_{bottom} can be determined using the formulas presented in the next section.

R_g and R_{bottom} are both 1206-size SMT resistors.



Exercise caution when replacing any of the analog input resistors or capacitors. Use the proper SMT rework equipment for removing and replacing these parts.

3.5.1 Scaling Input Range

Once the input range has been determined, the appropriate resistor values must be selected. The following steps enumerate how to achieve the appropriate register values.

1. Choose a gain resistor. Use Equation (3-1) to determine the value of the gain resistor.

$$R_g = \frac{2.5 \times 10^5}{V_{IN_{max}} - V_{IN_{min}}} \quad (3-1)$$

2. Select R_{BOTTOM} using Equation (3-2).

$$R_{BOTTOM} = \frac{R_g \cdot V_{IN_{max}} \cdot 10^4}{R_g \cdot (2.5 - V_{IN_{max}}) + (2.5 \times 10^5)} \quad (3-2)$$

3. Select the appropriate resistor values. Standard resistor values can be found in many electronics references and catalogs. Using 1% resistors will give better accuracy and a greater number of choices than using 5% resistors.

- For R_g , select the next standard value less than the standard value closest to the computed value. Choosing a lower value helps insure that the input signal does not exceed the 2.5 V reference voltage for the ADC.
- For R_{BOTTOM} , select the nearest standard value.



If the computed values exceed 3 M Ω , it may be necessary to change the values of R_{TOP} and R_{IN} . Calculations for these resistors are presented in the next section.

4. To verify that the calculated values provide the correct gain and offset, plug $V_{IN_{min}}$ and $V_{IN_{max}}$ into Equation (3-3).

$$V_{OUT} = \left(\frac{R_{BOTTOM}}{R_{BOTTOM} + 10^4} \cdot 2.5 - V_{IN} \right) \left(\frac{R_g}{10^5} \right) + \left(\frac{R_{BOTTOM}}{R_{BOTTOM} + 10^4} \cdot 2.5 \right) \quad (3-3)$$

$V_{IN_{min}}$ should yield a positive V_{OUT} just less (within 50 mV) than 2.5 V. $V_{IN_{max}}$ should yield a positive V_{OUT} just greater (within 50 mV) than zero.

5. Test the circuit to verify that it works properly.

Scaling Input Range: Example

Given a sensor with an output of -1 V to 2 V, V_{INmin} and V_{INmax} are as follows.

$$V_{INmax} = 2 \text{ V}, V_{INmin} = -1 \text{ V}$$

1. Determine the value of the gain resistor using Equation (3-1).

$$\begin{aligned} R_g &= \frac{2.5 \times 10^5}{2 - (-1)} \\ &= 83.3 \text{ k}\Omega \text{ (ideal value)} \end{aligned}$$

2. Select R_{BOTTOM} using Equation (3-2).

$$R_{BOTTOM} = \frac{83.3 \times 10^3 \cdot 2 \cdot 10^4}{(83.3 \times 10^3) \cdot (2.5 - 2) + 2.5 \cdot 10^5}$$

3. Select resistor values:

- The next lower standard 1% value for R_g is 80.6 k Ω .
- The closest standard 1% value for R_{BOTTOM} is 5.76 k Ω .

4. Now, check values by plugging V_{INmin} and V_{INmax} into Equation (3-3).

- Check V_{OUT} for V_{INmin} .

$$\begin{aligned} V_{OUT} &= \left(\frac{5.76 \times 10^3}{5.76 \times 10^3 + 10^4} \cdot 2.5 - (-1) \right) \left(\frac{80.6 \times 10^3}{10^5} \right) \\ &\quad + \left(\frac{5.76 \times 10^3}{5.76 \times 10^3 + 10^4} \right) \cdot 2.5 \\ &= 2.456 \text{ V} \end{aligned}$$

This value of V_{OUT} is within 50 mV of 2.5 V using V_{INmin} .

- Check V_{OUT} for V_{INmax} .

$$\begin{aligned}
 V_{OUT} &= \left(\frac{5.76 \times 10^3}{5.76 \times 10^3 + 10^4} \cdot 2.5 - (2) \right) \left(\frac{80.6 \times 10^3}{10^5} \right) \\
 &\quad + \left(\frac{5.76 \times 10^3}{5.76 \times 10^3 + 10^4} \right) \cdot 2.5 \\
 &= 0.038 \text{ V}
 \end{aligned}$$

This value of V_{OUT} is within 50 mV of 0 V using V_{INmax} .

V_{OUT} falls within the acceptable output range.

In this example, with $R_g = 80.6 \text{ k}\Omega$ and $R_{BOTTOM} = 5.76 \text{ k}\Omega$, the analog channel will accept inputs from -1 V to 2 V. The amplifier output is in the range 0 V to 2.5 V, with a little margin (~30 mV) for system error (e.g., ADC or op-amp offset).

3.6 Operation

This section presents a complete analysis of the analog input circuit (see Figure 12).

The first-order approximation of the op-amp assumes the following criteria:

- Infinite open-loop gain
- Zero output impedance
- Zero voltage offset
- Infinite input impedance
- Zero input bias currents
- Noiseless components

3.6.1 The V_{OFF} Voltage Divider

The voltage divider formed by R_{TOP} and R_{BOTTOM} provides an offset voltage for the amplifier. The offset voltage V_{OFF} is given by Equation (3-4).

$$V_{OFF} = \left(\frac{R_{BOTTOM}}{R_{BOTTOM} + R_{TOP}} \right) \cdot V_{REF} \tag{3-4}$$

V_{REF} for the LP3100 is 2.5 V.

3.6.2 DC Gain

Examine the DC gain of the circuit.

An amplifier in a negative feedback topology will force the error between the amplifier's inputs to zero.

This implies that $V_{(INVERTING)} = V_{(NONINVERTING)} = V_{OFF}$.

Since there is infinite impedance at the op-amp's inputs, the current through R_{IN} must equal the current through R_g . The current through R_{IN} is determined by Equation (3-5).

$$I_{R_{IN}} = \frac{V_{IN} - V_{OFF}}{R_{IN}} \quad (3-5)$$

The current through R_g , I_{R_g} , is determined by Equation (3-6).

$$I_{R_{IN}} = \frac{V_{OFF} - V_{OUT}}{R_g} \quad (3-6)$$

Setting Equation (3-5) and Equation (3-6) equal and solving for V_{OUT} yields Equation (3-7).

$$V_{OUT} = \left(\frac{V_{OFF} - V_{IN}}{R_{IN}} \right) \cdot R_g + V_{OFF} \quad (3-7)$$

When V_{OFF} is zero, the circuit scales V_{IN} by a factor of $-R_g/R_{IN}$. This is the DC gain. The DC gain can be determined by Equation (3-8).

$$g = \frac{-R_g}{R_{IN}} \Rightarrow |g| = \frac{R_g}{R_{IN}} \quad (3-8)$$

This result agrees with the gain of a classic op-amp inverting amplifier.

Given a desired input range, it is possible to compute the required circuit gain. The amplifier needs to map the input voltage range to the 0 V to V_{REF} input range of the ADC.

$$g = \frac{V_{REF}}{V_{IN_{max}} - V_{IN_{min}}} \quad (3-9)$$

The LP3100 has $V_{REF} = 2.5$ V and R_{IN} is shipped as 100 k Ω .

Once the gain has been computed, Equation (3-8) can be used to compute R_g .

3.6.3 Finding V_{OFF}

Once the gain is known, the V_{OFF} required to center the output in the range 0 V to V_{REF} can be determined.

Examine V_{OFF} when V_{IN} is at minimum and maximum.

First, take the case of a maximum V_{IN} . Because the circuit is an inverting amplifier, $V_{IN_{max}}$ must map V_{OUT} to zero.

Start with Equation (3-7) and substitute $V_{OUT} = 0$ and $V_{IN} = V_{IN_{max}}$ to get Equation (3-10).

$$\begin{aligned} 0 &= \left(\frac{V_{OFF} - V_{IN_{max}}}{R_{IN}} \right) \cdot R_g + V_{OFF} \\ &= (V_{OFF} - V_{IN_{max}}) \left(\frac{R_g}{R_{IN}} \right) + V_{OFF} \end{aligned} \quad (3-10)$$

Now, simplify using Equation (3-8). Substitute Equation (3-8) into Equation (3-10) to get Equation (3-11).

$$\begin{aligned} 0 &= (V_{OFF} - V_{IN_{max}})(-g) + V_{OFF} \\ &= (V_{IN_{max}} - V_{OFF})g + V_{OFF} \end{aligned} \quad (3-11)$$

Then,

$$V_{OFF} = V_{IN_{max}} \left(\frac{g}{g + 1} \right). \quad (3-12)$$

Do the same thing for $V_{IN} = V_{IN_{min}}$. When $V_{IN_{min}}$ is presented at V_{IN} , $V_{OUT} = V_{REF}$ has to be true. Start with Equation (3-7) and substitute $V_{OUT} = V_{REF}$ and $V_{IN} = V_{IN_{min}}$ to get Equation (3-13).

$$\begin{aligned}
 V_{REF} &= \left(\frac{V_{OFF} - V_{IN_{min}}}{R_{IN}} \right) (R_g) + V_{OFF} \\
 &= (V_{OFF} - V_{IN_{min}}) \left(\frac{R_g}{R_{IN}} \right) + V_{OFF} \\
 &= (V_{OFF} - V_{IN_{min}}) (-g) + V_{OFF} \tag{3-13} \\
 &= (V_{IN_{min}} - V_{OFF}) g + V_{OFF} \\
 &= V_{IN_{min}} g + V_{OFF} (1 - g)
 \end{aligned}$$

Rearrange and solve for V_{OFF} . Either Equation (3-12) or Equation (3-14) may be used to calculate V_{OFF} .

$$V_{OFF} = \frac{V_{IN_{min}} (g) - V_{REF}}{(g - 1)} \tag{3-14}$$

3.6.4 Practical Considerations

Resistors are available only in discrete standard values. Once ideal values are computed, a standard value must be selected. There are more 1% resistor values to choose from than there are 5% resistor values. Also, 1% resistors have a lower temperature drift. There is only a slight difference in cost between 1% and 5% resistors.

Resistors over 3 M Ω should be avoided. Two reasons to avoid large resistor values are the susceptibility of the circuit to noise and the availability of parts. Reducing R_{IN} or adding an external pre-amp are alternative methods of increasing gain if a large DC gain is needed.

The op-amp in the LP3100 has a ± 7.5 mV maximum offset voltage (at 25°C). This offset is multiplied by the DC gain and added to V_{OUT} . This means if the DC gain is 10, V_{OUT} may have a 75 mV offset in it.

The input bias currents of the op-amp will also produce error voltages at the inputs that get multiplied by the DC gain, and will show up as an offset in V_{OUT} .

To avoid these offsets pushing V_{OUT} beyond the 0 V to V_{REF} range of the ADC, select R_g to be a smaller standard value than computed. This will sacrifice some dynamic range of the ADC for improved reliability.

Selecting R_{BOTTOM} is a matter of picking the standard value closest to the computed value.

3.6.5 Input Impedance

The input impedance looking into the circuit from V_{IN} is just R_{IN} . Note also that R_{IN} is connected to the inverting input, which is maintained (by the op-amp's negative feedback) at V_{OFF} .

Gain can be increased by sacrificing input impedance. A fixed value of R_g will produce a larger DC gain if R_{IN} is reduced. However, a smaller R_{IN} will require the source of V_{IN} (often a transducer) to provide additional current, as shown in Equation (3-15).

$$I_{R_{IN}} = \frac{V_{IN} - V_{OFF}}{R_{IN}} \quad (3-15)$$

3.6.6 Frequency Response

The capacitor in the feedback loop fixes a pole as shown in Equation (3-16).

$$F_c = \frac{1}{2\pi \cdot R_g \cdot C_{FB}} \quad (3-16)$$

F_c , the 3 dB point for the single-pole filter, is in hertz, and C is capacitance. The filter will roll off at the 20 dB decade after F_c .

This low-pass filter helps eliminate noise in the channel. The pole should be set as low as possible for the application. The standard capacitor shipped is 0.001 μ F.

3.6.7 Using the ADC

The best way to use the ADC is with the Z-World Dynamic C drivers. Using the Dynamic C drivers helps ensure that the code will be compatible with future versions of the LP3100 as well as other Z-World products. Program 3-7 illustrates the use of the Dynamic C ADC functions.

Program 3-7. Reading Analog Inputs with Dynamic C Functions

```
/* Read the analog inputs using Dynamic C Functions */

#include eziolp31.lib

#define INPUTCHAN 0

main() {
    float raw, analog;
    eioBrdInit(0);           /* initialize the I/O driver */
    eioBrdDO(nADENA, 0);    /* enable analog inputs */
    eioErrorCode = 0;       // clear error flag
    raw = eioBrdAI(INPUTCHAN+16); /* read the raw chan.*/

    /* read channel, scale with calibration constants */

    eioErrorCode = 0;       // clear error flag
    analog = eioBrdAI(INPUTCHAN);
    if (eioErrorCode & EIO_NODEV) {
        printf("analog input channel %d doesn't exist!\n",
              INPUTCHAN);
    }
    else {
        printf("analog input channel %d reads 0x%04x,
              interpreted to %f\n", INPUTCHAN, (int)raw,
              analog);
    }
}
```



printf statements must be on one continuous line in an executable program. In this sample program, the **printf** statement is shown as more than one line only for display.

3.6.8 Using the Analog Inputs

The factory calibrates each LP3100, storing each unit's individual zero offset and actual gain for each channel in simulated EEPROM. The library function **eioBrdAI** uses these calibration values to provide adjusted readings for the analog inputs.

3.7 Serial Communication

The serial channels provide a simple method for connecting the LP3100 to other serial devices. The LP3100 has hardware drivers for both RS-232 and RS-485 signals. Speeds up to 38,400 bps are possible when the system clock is set to 6.144 MHz. The maximum speed is 19,200 bps when the clock is set to 3.072 MHz.

3.7.1 Operation

The LP3100 has two serial channels. These serial channels can be configured in the following three ways:

1. One 5-wire RS-232 channel and one RS-485 channel.
2. One 3-wire RS-232 channel and one RS-485 channel.
3. Two 3-wire RS-232 channels.

The serial channels, Z0 and Z1, originate from UARTs on the Z180 microprocessor used on the LP3100. Serial Channel Z0 has handshaking signals available when configured as a 5-wire RS-232 port. Serial Channel Z1 does not support hardware handshaking.

Table 13 summarizes the serial channel signals for channels Z0 and Z1.

Table 13. Serial Channel Signals

Serial Channel Z0	Serial Channel Z1
TXA0—Transmit channel 0	TXA1—Transmit channel 1
RXA0—Receive Channel 0	RXA1—Receive channel 1
RTS0—Request to send, channel 0	
CTS0—Clear to send, channel 0	

Serial Channel Z0 can be configured as either a 3- or 5-wire RS-232 interface. Z0 can only be configured for RS-232. Z1 can be configured either as a 3-wire RS-232 or as an RS-485 interface.

The RS-232 converter (LTC1385) has two inputs and two outputs. When Z0 is configured as a 5-wire interface, all four LTC1385 I/O lines are used. Table 14 lists how resources are allocated for a Z0 5-wire interface.

Table 14. 5-wire RS-232 Configuration Signals

Z0 Signal	RS-232 Converter (LTC1385)
TXA0—Transmit channel 0	TX0—RS-232 transmit
RXA0—Receive Channel 0	RX0—RS-232 receive
RTS0—Request to send, channel 0	TX1—RS-232 RTS
CTS0—Clear to send, channel 0	RX1—RS-232 CTS

When Z0 is configured as a 3-wire RS-232 interface, only two LTC1385 I/O lines are used. Table 15 lists the resource allocation for one 3-wire RS-232 interface.

Table 15. 3-wire RS-232 Configuration Signals

Z0 Signal	RS-232 Converter (LTC1385)
TXA0—Transmit channel 0	TX0—RS-232 transmit
RXA0—Receive Channel 0	RX0—RS-232 receive



When Z0 is configured as a 3-wire RS-232 interface, there are enough LTC1385 resources left to have a second 3-wire interface. Table 16 lists the resource allocation for two 3-wire RS-232 interfaces. Z0 is a dedicated RS-232 interface (3- or 5-wire). Z180 Channel Z1 can be either an RS-232 or an RS-485 interface.

Table 16. Two 3-wire RS-232 Configuration Signals

Channel	Signal	RS-232 Converter (LTC1385)
Z0	TXA0—Transmit channel 0	TX0—RS-232 transmit channel 0
	RXA0—Receive Channel 0	RX0—RS-232 receive channel 0
Z1	TXA1—Transmit channel 1	TX1—RS-232 transmit channel 1
	RXA1—Receive Channel 1	RX1—RS-232 receive channel 1

LP3100 resources are configured using both hardware and software. The jumper settings are shown in Table 17.

Table 17. Serial Channel Jumper Settings

Configuration	J2	J3
Two 3-wire RS-232	2-3 	2-3 
One 3-wire RS-232 and one RS-485	1-2	1-2
One 5-wire RS-232 and one RS-485	1-2	1-2

The jumper settings for the last two configurations are the same. Software drivers set up Channel Z0 for a 3- or 5-wire RS-232 interface. The jumpers are used to route the Z180 Channel Z1 signals to either the RS-232 transceiver or the RS-485 transceiver.

3.7.2 RS-232 Communication

The RS-232 channels and the supplied Dynamic C software allow the LP3100 to communicate with other computers or controllers. By adding a modem, remote communication can be achieved (including remote downloading) using the X-modem protocol. Software driver examples for RS-232 can be found in the Dynamic C `SAMPLES\AASC` subdirectory. For additional information on remote downloading, refer to the Dynamic C user's manuals.

Use the optional Z-World SIB2 to make all serial channels available to the application during software development.



See Chapter 2, "Getting Started," and Appendix C, "Serial Interface Board," for more information.

3.7.3 RS-485 Communication

The LP3100 can be configured for one RS-485 communication channel using Serial Channel Z1. The LP3100 provides a two-wire RS-485 interface. The + signal is available on pin 32 of H2 and the - signal is available on pin 31 of H2. Using RS-485, signals can be carried up to 4,000 feet, assuming the line is properly terminated and the baud rate is 9,600 bps or less. As a general rule, the maximum transmission distance is halved for each doubling of the baud rate.



Some manufacturers refer to + and - as A and B, although the assignment of A to + and B to - varies from company to company. Z-World uses + and - exclusively and consistently on all controllers that support RS-485.

With the RS-485 port, an LP3100 can be connected to other controllers, expansion boards, or a variety of third-party devices. The RS-485 transceiver of the LP3100 is capable of driving up to 31 other RS-485 devices. If more than 32 devices (including the LP3100) are required on a network, then RS-485 repeaters must be used.



RS-485 repeaters are available from companies such as Black Box and B&B Electronics.

Figure 14 illustrates the wiring of an RS-485 network.

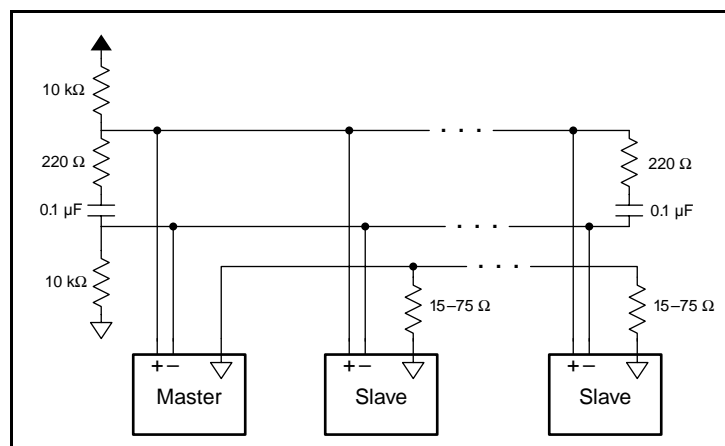


Figure 14. RS-485 Network Wiring

For most RS-485 systems, simply running two wires (+ and –) is sufficient. However, a third wire may be required over long distances and in electrically noisy systems. Rather than using a traditional ground for the third wire, connect a wire to the ground of the network master and then to the ground of each slave unit through a small resistor. The resistor value is typically 15 Ω to 75 Ω . This grounding technique eliminates significant grounding differences that may prevent the RS-485 transceiver from communicating. Another added benefit of this grounding technique is that it provides a return path for current sent by the RS-485 transmitter without forcing a common ground that could create other electrical problems for nodes on the network.

On a traditional RS-485 network, the master node provides network biasing that holds the RS-485 in an idle state when no transmitters are active. Typically, bias resistors are also included on slave units, and must be removed for the slave units to operate properly. However, the LP3100 uses a small pull-up resistor (10 k Ω), which can remain on the slave units and still allow proper operation.


Termination resistors are recommended in a multidrop network to reduce reflections. Termination resistors are typically installed on the master controller and on the slave unit located at the opposite end of the network. For termination, install a 220 Ω resistor across the RS-485+ and RS-485– connections on the LP3100 as shown in Figure 14.

3.7.4 Software

Serial channel drivers are located in the AASC library. Other drivers are also available. Comprehensive information on the serial channel software and programming can be found in the *Dynamic C Function Reference Manual* and the other Dynamic C manuals.

3.8 Real-Time Clock

The Dallas Semiconductor DS1685 RTC provides several functions for the LP3100. This RTC keeps track of time while providing an alarm function, system power control, a unique 64-bit serial number, and additional user RAM.

 For detailed information on the DS1685 RTC, see the Dallas Semiconductor *Timekeeping and NV SRAM Databook*.

During normal operation, the RTC is powered from VCC. If VCC is shut down, the RTC is powered by VCCU. If both VCC and VCCU are shut down, the RTC can be backed up with the battery that backs up the RAM. This allows the RTC to keep time even when the main power source is removed.

The RTC uses a 32,768 Hz crystal as a time base. This low frequency keeps radiated electromagnetic interference to a minimum.

Z-World provides drivers to access the most commonly used features. The functions `tm_rd()` and `tm_wr()` read and write data to the real-time clock. These functions are documented in the *Dynamic C Function Reference Manual*.

Refer to the DS1685 data sheet for more details if there is a need to access the RTC hardware directly. The RTC address space is accessed by a two-step operation. First, a RTC

address is latched into the RTC address latch. This is accomplished by a Z180 I/O write to 0x4020. For example, use the following function call to access REG A in the RTC address space (RTC 0x0A):

```
output(0x4020, 0x0A); // set up address for REG A
```

Second, the RTC data can be read or written with another I/O cycle to 0x4000. For example, use the following command to write 0x03 to the RTC address currently latched into the RTC:

```
output(0x4000, 0x03); // write byte to RTC
```

3.8.1 Real-Time Clock Interrupts

An important function is generating interrupts to support application features. The interrupts can be generated for the following causes:

- **Periodic** - The application may specify a frequency at which the RTC interrupts.
- **Alarm** - The application may specify a particular time (per second, per minute, per hour, per day and per month) at which the RTC interrupts.
- **Update ended** - The RTC can interrupt at the end of each update cycle. This is not used in the Dynamic C RTC drivers.
- **Wake up** - The RTC can reenale VCC of the controller and interrupt at a particular time (per second, per minute, per hour, per day and per month).
- **Kick start** - The RTC can reenale VCC of the controller and interrupt by monitoring a kick start line.
- **RAM clear** - Not used on the LP3100 series controllers.

The periodic, alarm, wake up and kick start interrupts are particularly useful on the LP3100 series controllers. If an application needs a high-frequency periodic interrupt (2 Hz to 8.192 kHz), the periodic interrupt feature can be used. If your application needs a low-frequency periodic interrupt (per second to per month), the alarm feature can be used. In order to save power, shut down VCC on the controller, and rely on either the wake up or kick start feature of the RTC to enable VCC at a pre-set time or when an external event happens.

Hardware Connection

The /IRQ line (active low) of the RTC is connected to /INT0 of the Z180.

Software Setup

The following procedure should be followed to set up the RTC for interrupt generation and servicing.

1. Write interrupt service routine. The ISR can be written in C or assembly. If it is a C routine, make sure the definition is qualified by **interrupt reti**. If it is an assembly routine, make sure the routine returns by the instruction **reti**.

2. Vector to the custom interrupt routine. This is normally accomplished by using the directive `#JUMP_VEC RST38_VEC myISR`, in which `myISR` should be replaced by the name of the interrupt routine.
3. `INT0` should be disabled when the interrupt is being set up. Note that by default, when the main function is called, `INT0` is disabled. `INT0` is maskable by resetting bit 0 in the I/O register `ITC` (address `0x34`) or via the global `DI` (disable interrupt) instruction. The application can call `rtcIRQ(0)` to reset bit 0 in register `ITC`.
4. If the RTC cannot be assumed initialized, call `rtcInit()` to initialize it. Also, use the following macros to disable all interrupt sources for initialization: `rtcSwAIE(0)`, `rtcSwKSE(0)`, `rtcSwPIE(0)`, `rtcSwUIE(0)`, `rtcSwRIE(0)`, and `rtcSwWIE(0)`.
5. Call `rtcClrIRQ()` to clear the interrupt flags for each interrupt source.
6. Set up the registers for each interrupt source. This is interrupt source dependent.
7. Enable `INT0` by calling `rtcIRQ(1)`.
8. Enable the interrupt source. This is interrupt source dependent.

3.8.2 Periodic Interrupts

Use the macro `rtcSetPIRate(x)` to specify the frequency of interrupts. The mapping between `x` and the period of the interrupt is listed in Table 18.

Table 18. Interrupt Mapping

Value of x	Interrupt Period	Value of x	Interrupt Period	Value of x	Interrupt Period
1	3.90625 ms	6	976.5625 μ s	11	31.25 ms
2	7.8125 ms	7	1.953125 ms	12	62.5 ms
3	122.070 μ s	8	3.90625 ms	13	125 ms
4	244.141 μ s	9	7.8125 ms	14	250 ms
5	488.281 μ s	10	15.625 ms	15	500 ms

Use the macro `rtcSwPIE(1)` to enable the interrupt source and `rtcSwPIE(0)` to disable the interrupt source. In the interrupt service routine, either read register C (use macro `rtcRdRegC()`) explicitly or use the macro `rtcChkPF()` to clear the interrupt flag. Note that if the flag is not cleared (by reading), the interrupt stays asserted. If you are also using the alarm or update-ended interrupts, you should read register C to determine the exact cause(s) of the interrupt. The periodic interrupt flag is bit 6 of register C. The following code illustrates the idea.

```

flags = rtcRdRegC();
if (flags & 0x40) {
    // handle periodic interrupt here
}
// check other bits of flag here if required

```

Refer to the sample program `SAMPLES/LP31XX/RTCPER.C`.

3.8.3 Alarm Interrupts

An alarm can be set using the `rtcSetAlmTime` function. The alarm will cause the RTC to generate an interrupt (/INT0) when the alarm expires. This can be used to wake the Z180 from a sleep mode or initiate periodic actions. Alarms are typically used in systems taking periodic samples over long periods of time (hours, days, weeks, or months). VCCU must be active in order for an alarm to wake the microprocessor.

The alarm has four fields to specify when to generate the alarm interrupt. There are fields to specify the second, minute, hour and month-day to generate the alarm interrupt. Each field, including the second field, can be initialized with a “don’t care” (0xc0) value. The interrupt is generated if the current time in the RTC matches all the alarm fields. A “don’t care” value in the alarm field matches all RTC values of the same field. For example, if you need an interrupt every hour, specify 00 for the second and minute alarm fields, and use 0xc0 for the hour and month-day fields.

The function `rtcSetAlmTime(struct tm *t)` is used to initialize the alarm fields. Initialize a `struct tm` variable first, then call the `rtcSetAlmTime` function to initialize fields in the RTC. Use the macro `rtcSwAIE(1)` to enable the interrupt source and `rtcSwAIE(0)` to disable the interrupt source. In the interrupt service routine, either read register C (use macro `rtcRdRegC()`) explicitly or use the macro `rtcChkAF()` to clear the interrupt flag. Note that if the flag is not cleared (by reading), the interrupt stays asserted. If you are also using the periodic or update-ended interrupts, you should read register C to determine the exact cause(s) of the interrupt. The periodic interrupt flag is bit 5 of register C. The following code illustrates the concept.

```
if (flags & 0x20) {
    // handle alarm interrupt here
}
```

Refer to the sample program `SAMPLES/LP31XX/RTCALM.C`.


3.8.4 Wake Up VCC

VCC on the LP3100 series controllers can be shut down by software. When VCC is shut down, the application can set up the RTC to power-up VCC again when either the kickstart pin is toggled or at a particular time. VCCU must be available for these operations.

Call `lp31shutdown` to shut down VCC. This function takes two arguments. The first argument is a flag to indicate whether kickstart should be enabled. If the parameter is non-zero, the kickstart feature is enabled. The second argument is a flag to indicate whether the wake-up feature should be enabled. If the parameter is non-zero, the wake-up feature is enabled. `lp31shutdown` does not return to the caller.

When power is resumed, the application’s main function will be called again. At this point, the application should check the kick start flag and wake up flag to determine the cause of the power-up. The kick start flag is checked with the macro `rtcChkKF()`, and the wake up flag is checked with the macro `rtcChkWF()`. As soon as the flags (kickstart and wake up) are checked, they should be cleared. To clear the kick start flag, use the

macro `rtcSwKF(0)`. To clear the wake up flag, use the macro `rtcSwWF(0)`. Furthermore, in order not to interfere with other sources of interrupts from the RTC, kick start and wake up should not generate interrupts. The application can disable kickstart and wake up from generating interrupts by using the macros `rtcSwKSE(0)` and `rtcSwWIE(0)` respectively.

 For a simple example, refer to the sample program `SAMPLES/LP31XX/RTCWAKE.C`.

3.9 LPBus

The LPBus is an expansion bus that allows boards to be added to a system to extend its I/O capabilities. The LPBus is designed to support as many as eight expansion boards, available from either Z-World or they may be a customer design. Appendix E, “LPBus Prototyping Board,” provides information on designing LPBus expansion boards.

The LPBus is based on the Z180’s microprocessor bus. One LPBus bus cycle corresponds to an I/O cycle on the Z180 bus. All of the LPBus signals are buffered so that loading on the expansion bus will not affect the microprocessor bus. LPBus control signals provide ample set-up and hold time to make design of expansion boards easy and reliable. The LPBus protocol permits each expansion board to have an address jumper that selects one of eight Device Selects (/DS0–/DS7). Each board on the bus must be assigned a unique Device Select.

The mechanical interface for the LPBus is H3, a 40-pin, 2 mm pitch header. There is another 40-pin header, H2, at the opposite end of the LP3100 board for other I/O signals. It is mechanically possible to plug an adapter board into the LP3100 incorrectly by rotating the board 180 degrees, swapping the position of the LPBus and I/O headers. This could damage the electronic components if power is applied. Note that headers H2 and H3 are offset from the center of the board by about 0.050 inches. If the boards are plugged in incorrectly, the board edges and mounting holes will not be aligned.

Figure 15 illustrates the location of header H3.

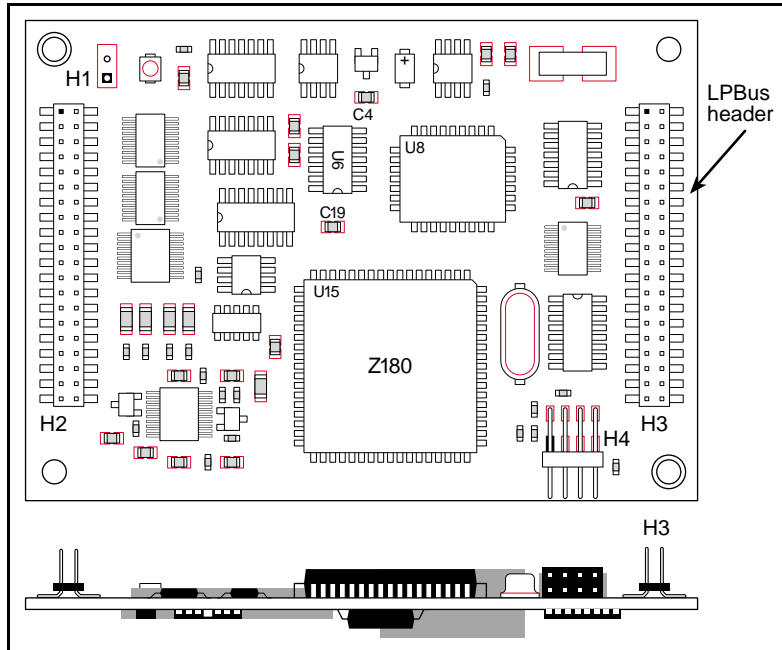


Figure 15. LPBus Header Connection Location

Expansion boards can be mounted on the LP3100 in one of two ways as shown in Figure 16.

1. **Flat** — Use a multidrop ribbon cable to connected adjacent boards.
2. **Vertical stacking** — Use female connectors on the bottom side of the expansion board.



Figure 16. LPBus Expansion Board Arrangements

The LPBus protocol has provision for slave board powerup and powerdown. This allows each expansion board to have independent power control. The buffer logic on the powered-down expansion board isolates the buses to prevent leakage between powered and unpowered subsystems. Also, any expansion board can remain powered while the LP3100 controller is powered down. The active expansion board can wake the controller via the /INT/KS pin.

3.9.1 LPBus Signals

Table 19 details the 40-pin LPBus header H3 and lists each pin function.

Table 19. H3 LPBus Pinout and Signals

H3 Pin	Signal	H3 Pin	Signal
1	VCCU	2	GROUND
3	DCIN	4	/PWR
5	/BRESET	6	DLS
7	/ENFSH	8	/INT/KS
9	/DREQ0	10	/DREQ1
11	GROUND	12	BD0
13	BD1	14	BD2
15	BD3	16	BD4
17	BD5	18	BD6
19	BD7	20	GROUND
21	BA0	22	BA1
23	BA2	24	BA3
25	BA4	26	BA5
27	BA6	28	/DS0
29	/DS1	30	/DS2
31	/DS3	32	/DS4
33	/DS5	34	/DS6
35	/DS7	36	/BRD
37	/BWR	38	BE
39	GROUND	40	VCCU

VCCU, GROUND, DCIN

GND (or GROUND) is tied to the LP3100's ground plane. VCCU is an unswitched power supply that is on whenever DCIN is in the acceptable input range. DCIN is the raw supply voltage supplied to the board.

/PWR

/PWR is the line that controls the switched VCC supply on the LP3100. It is driven low by the power control logic to turn on VCC and is pulled up to VCCU with a resistor. It can also be used by an expansion board to monitor the state of VCC on the LP3100. In some applications, an expansion board may need to have control of this line.

/BRESET

This active-low output is the Bus Reset line that can be used to reset expansion boards during a power-on reset or under software control. **/BRESET** is controlled by a bit-addressable latch at address 4081H.

DLS

By driving this input high, an expansion board can disable the LP3100's onboard regulators. It would normally only be driven high if an auxiliary power supply is installed in the system to supply power to the LP3100 in addition to supplying power to the rest of the system. For example, a switching boost regulator operating from a single 1.5 V battery cell may be installed. In that case, the 1.5 V source is too low to be used by the LP3100's regulators and they would need to be disabled by driving this line high. Leave this line unconnected if it is not used.

/ENFSH

/ENFSH is reserved for use by Z-World and should not be connected. This active-low input line is not used in normal LPBus operations, but is used to program a blank or corrupted flash memory on the LP3100. Access to the local flash memory is inhibited when this control line is externally driven low during a power-up cycle. Instead, memory read cycles are diverted to the LPBus so that data are supplied by an external memory board that causes the flash programming code programming the flash memory to execute.

/INT/KS

/INT/KS is an active-low input. By driving this input line low, an expansion board can either “awaken” or kick start the LP3100 out of a power-off “sleep” state, or if power is already on, generate an interrupt.

/DREQ0 and /DREQ1

These active-low inputs are DMA request lines for the Z180. The DMA channels can be used for high-speed I/O transfers to and from LPBus expansion boards.

BD0–BD7

These are a bidirectional buffered extension of the Z180 data bus.

BA0–BA6

These outputs are a buffered extension of the A0–A7 of the Z180 I/O address bus.

In addition to the eight decoded Device Select (**/DSx**) lines, there are seven address lines available on the LPBus. This allows up to 128 I/O addresses for each device select. BA6 has a dual purpose and can be used to implement power on/off control for expansion boards. If BA6 is used for this purpose, there are 64 I/O addresses per device select line.

The BA6 power control is active for any write to an I/O location corresponding to the addressed boards. When BA6 is used for power control, setting BA6 to 0 causes the addresses board to power up. Setting BA6 to 1 causes the board to power down.

/DS0–/DS7

The active-low Device Select outputs are asserted during an LPBus I/O cycle and reflect the portion of the I/O address map that the Z180 is addressing during that cycle. Typically an LPBus expansion board would use one of these device select lines together with the BA0–BA6 address lines to completely specify a particular I/O address.

/BRD, /BWR

These active-low outputs are the equivalent of the Z180 I/O read and write lines. /BRD and /BWR are buffered and ANDed with the Z180 Auxiliary strobe line “E” to insure plenty of setup time for decoding a board’s address.

BE

This output is a buffered copy of the Z180 auxiliary strobe line “E.”

3.9.2 Board ID

LPBus expansion boards supplied by Z-World have a jumper array that allows user selection of the device select line used for each board. Each board in a system would be jumpered to use a different line.

Table 20 lists the LPBus decoded addresses.

Table 20. LPBus Decoded Addresses

Select	I/O Address	BA0–BA5	BA0–BA6*
/DS0	8000–803F (8000–807F)	00–3F	00–7F
/DS1	8080–80BF (8080–80FF)	80–BF	80–FF
/DS2	8100–813F (8100–807F)	00–3F	00–7F
/DS3	8180–81BF (8180–81FF)	80–BF	80–FF
/DS4	8200–823F (8200–827F)	00–3F	00–7F
/DS5	8280–82BF (8280–82FF)	80–BF	80–FF
/DS6	8300–833F (8300–837F)	00–3F	00–7F
/DS7	8380–83BF (8380–83FF)	80–BF	80–FF

* When BA6 is not used for power control

Each LPBus expansion board type has a board ID register containing a unique code that can be read by the LP3100. This allows the LP3100 to determine what types of boards exist in a system. By Z-World convention this ID register can be read at the lowest I/O address within the address range corresponding to the Device Select line assigned to the board. For example, to read the Board ID of the board that has been jumpered to use /DS0, the LP3100 would read I/O address 8000H; for /DS1 it would read 8080H, and so on. Z-World suggests following this convention for any boards designed by the customer. Z-World has reserved a set of ID codes for user-designed LPBus expansion boards to avoid any conflict over the use of an ID code that Z-World may use in the future for its own designs. Appendix E, “LPBus Prototyping Board,” lists the current and reserved codes.



CHAPTER 4. SOFTWARE REFERENCE

Chapter 4 describes functions for controlling an LP3100 and peripherals. The following sections are included.

- Using Dynamic C Drivers
- Digital Input/Output Functions
- Analog Input Functions
- Serial Communication Functions
- Power Control Functions
- RTC Functions
- Flash EPROM Functions
- LED Functions
- LCD Functions
- Keypad Functions
- Additional Software

4.1 Using Dynamic C Drivers

To use the functions discussed in this chapter, the library containing that function must be “included” in the program source code. For example, the following line of code at the top of a program will allow use of all of the functions in the **EZIOLP31** library:

```
#use eziolp31.lib
```

See the Dynamic C manuals for more information on the **#use** directive and using libraries.

4.2 Digital Input/Output Functions

Table 21 lists channel numbers for the digital I/O channels and peripheral control registers. The channel numbers are for use with the Dynamic C drivers. The physical addresses are listed in Table 22 and in Appendix H.

Table 21. Digital Input/Output Channel Numbers

eioBrdDO	eioBrdDI	Channel No.	eioBrdDO	eioBrdDI	Channel No.
DINOUT0	DINOUT0	0	/EN_DKS	N/A	16
DINOUT1	DINOUT1	1	/MBR	N/A	17
DINOUT2	DINOUT2	2	/232TEN	N/A	18
DINOUT3	DINOUT3	3	232EN	N/A	19
DINOUT4	DINOUT4	4	/CTSEN	N/A	20
DINOUT5	DINOUT5	5	/PWREN	N/A	21
DINOUT6	DINOUT6	6	N/A	N/A	22
DINOUT7	DINOUT7	7	N/A	N/A	23
DOUT0	DIN0	8	/LED	N/A	24
DOUT1	DIN1	9	485TE	N/A	25
DOUT2	DIN2	10	/485RE	N/A	26
DOUT3	DIN3	11	/AD_CS	N/A	27
DOUT4	N/A	12	/DOE	N/A	28
DOUT5	N/A	13	/ADENA	N/A	29
DOUT6	N/A	14	N/A	N/A	30
DOUT7	N/A	15	N/A	N/A	31

Table 22 lists addresses for the digital input, digital output, and digital input/output channels. Use these addresses with the `inport` and `outport` functions.

Table 22. Digital Input, Digital Output, and Digital Input/Output Addresses

Channel	inport Address	outport Address
DIN 0-3	4040H	N/A
DOUT 0-7	N/A	4060H
DINOUT	4020H	4040H

`int eioBrdDI(unsigned int eioAddr)`

Reads the state of one of the digital inputs. Table 21 lists the channel numbers corresponding to specific LP3100 resources.

PARAMETER

eioAddr specifies the input to be read. Valid numbers are from 0 to 11: 0–7 represent the eight digital input/output channels, and 8–11 represent the four dedicated digital inputs.

RETURN VALUE

0 if input reads low

1 if input reads high

–1 if **eioAddr** is out of range

ERROR: If **eioAddr** is out of range, **eioErrorCode** is bit-ored with the constant **EIO_NODEV**. **eioErrorCode** is a global **int** defined in the library.

LIBRARY

EZIOLP31.LIB



eioErrorCode is not cleared by **eioBrdDI**.

EXAMPLE

```
void main (void) {
    int x;
    eioBrdInit(0); // reset all LP3100 resources to default states
    while(1) {
        hitwd(); // hit watchdog to prevent reset
        eioErrorCode = eioErrorCode & ~EIO_NODEV;
        // clear the error flag
        x = eioBrdDI(8); // read channel 8, which is DIN0
        if (eioErrorCode & EIO_NODEV) {
            printf("\n\rChannel requested is out of range");
            while(1) hitwd(); // wait forever
        }
        switch (x) {
            case 0: printf("\n\rDIN is low");
                // x was zero
            default: printf("\n\rDIN is high");
                // x was non-zero
        }
    }
}
```

```
int eioBrdDO( unsigned int eioAddr, int state )
```

Sets or clears a digital output. A channel number is passed to **eioBrdDO** via **eioAddr**. Table 21 lists the channel numbers corresponding to specific LP3100 resources.

eioBrdDO does not abstract the activation or assertion level for a resource. The programmer must be aware of the activation level of the device being controlled. For example, the onboard LED, which is an active high device, is turned on with **eioBrdDO(24,1)**. To turn on the power to the analog section, /ADENA is low and so use **eioBrdDO(29,0)**.

PARAMETERS

eioAddr specifies the digital output to be set. Valid numbers are from 0 to 31: 0–7 represent the eight digital input/output channels, 8–15 represent the eight digital outputs, and 16–31 represent the internal outputs listed in Table 21. Channel numbers 22, 23, 30, and 31 are unused.

state determines whether a one or a zero is written as the digital output state for the specified output ($-32,768 < \mathbf{state} \leq 32,767$). When a non-zero value is written to the digital output, the digital output assumes a high-voltage (VCC level) state and the digital output is on. A zero makes the digital output assume ground potential, and so the digital output is off.

RETURN VALUE

0 if successful

–1 if **eioAddr** is out of range

ERROR: If **eioAddr** is out of range, **eioErrorCode** is bit-ored with the constant **EIO_NODEV**. **eioErrorCode** is a global **int** defined in the library.

eioErrorCode is not cleared by **eioBrdDO**.

LIBRARY

EZIOLP31.LIB

EXAMPLE

```
void main (void) { // blink the LED using eioBrdDO
    int x;
    eioBrdInit(0); // reset all LP3100 resources to default states
    while(1) {
        hitwd(); // hit watchdog to prevent reset
        eioErrorCode = 0; // clear the error flag
        x = eioBrdDO(24,1); // turn LED on
        for (x = 0; x < 6000; x++) // wait for a while
        x = eioBrdDO(24,0); // turn LED off
        for (x = 0; x < 6000; x++) // wait for a while
    }
}
```

unsigned int inport(unsigned int port)

Reads a value from a Z180 I/O port. When **inport** reads from the address space, A16...A19 are held low—only the lower 16 bits of the address bus are significant. Table 22 lists the addresses for some of the digital resources on the LP3100.

The **inport** function is useful when more than one digital input needs to be read simultaneously. For example, if the DIN0...3 inputs are tied to DIP switches, and the binary combination of the DIP switches was meaningful, then the **inport** function would be the function of choice. If only a single bit needs to be sampled, **eioBrdDI** would suffice.

When using **inport**, be aware of the context of the data returned. **inport** is a low-level function, and the hardware determines the position of the bit values and their position in a byte. The returned byte needs to be anded with a mask to remove bits that are not of interest.

PARAMETER

port is the LP3100 port address to read ($0 \leq \text{port} \leq 65,535$).

RETURN VALUE

Byte value read from the port.

LIBRARY

BIOS.LIB

EXAMPLE

```
void main (void) {
    int x;
    eioBrdInit(0); // reset all LP3100 resources to default states
    while() {
        hitwd(); // hit watchdog to prevent reset
        x = inport(0x4040);
        x = x&0x000F; // mask all high-order meaningless bits
        switch (x) {
            case 0:    printf("\n\rDIN0..3 set to 0");
                       break;
            case 1:    printf("\n\rDIN0..3 set to 1");
                       break;
            case 2:    printf("\n\rDIN0..3 set to 2");
                       break;
            case 3:    printf("\n\rDIN0..3 set to 3");
                       break;
            case 4:    printf("\n\rDIN0..3 set to 4");
                       break;
            case 5:    printf("\n\rDIN0..3 set to 5");
                       break;
            case 6:    printf("\n\rDIN0..3 set to 6");
                       break;
            case 7:    printf("\n\rDIN0..3 set to 7");
                       break;
            default:   printf("\n\rNever should get here!");
                       break;
        }
    }
}
```

void outport(unsigned int port, unsigned int value)

Writes data to a Z180 I/O port. The address written to is specified by **port**. The lower 8 bits of the second parameter, **value**, are written to the I/O address. The upper four bits of the Z180 address bus, A16...A19, are held low during the write operation. Table 4-2 lists some of the I/O address spaces for the Z180.

This function is useful for manipulating digital I/O resources on a bitwise basis. The outport function enables the application to write bits directly to the hardware without the need to use other drivers. However, Z-World recommends using the other drivers whenever they are available to maintain code portability between products and product revisions.

PARAMETERS

port is the LP3100 port address to be written ($0 \leq \text{port} \leq 65,535$).

value is the data to be written to the port ($0 \leq \text{value} \leq 65,535$). Only the lower byte (F) **value** is written to the port.

LIBRARY

BIOS.LIB

EXAMPLE

```
void main (void) {
    eioBrdInit(0); // reset all LP3100 resources to default states
    outport(0x040A0, 0x01);
                // turn LED on (see table in schematic)
}
```

4.3 Analog Input Functions

`void lp31ADCInit(void)`

Initializes the analog inputs. This includes turning on the power supply for the analog section and resetting the ADC.

`eioBrdInit` calls `lp31ADCInit` to turn on and reset the analog section. `lp31ADCInit` affects only the analog section, whereas `eioBrdInit` also configures other LP3100 resources.

`lp31ADCDis` is used to shut down the power supply for the analog section (AVCC).

LIBRARY

`EZIOLP31.LIB`

EXAMPLE

```
void main (void) {
    int x;
    eioBrdInit(0); // reset all LP3100 resources to default states
    lp31ADCInit(); // turn on, reset, and reconfigure
                  // analog input section
    x = (int)eioBrdAI(16);
                // read the raw ADC value from AINO
    printf("\n\rAnalog channel 0 = %d",x);
    lp31ADCDis(); // shut down analog section
}
```

float eioBrdAI(unsigned int eioAddr)

Reads the analog-to-digital converter and returns the data. The parameter **eioAddr** determines which channel will be read and how the data will be returned. The mode is unipolar input, 12-bit data length, most significant bit first.

PARAMETER

eioAddr must be 0 to 3 or 16 to 19. **eioAddr** values 0 through 3 represent analog inputs AIN0–AIN3. The function returns a calibrated measurement. The LP3100 is factory calibrated to return volts. The customer may use **eioBrdACalib** to specify another unit such as degrees Celsius or grams. The calibration is a simple two-point interpolation/extrapolation.

eioAddr values 16 through 19 also represent physical channels AIN0–AIN3, but cause the function to return a 12-bit raw data value (0–4095) for the analog input. No scaling or offset compensation is done.

RETURN VALUE

For **eioAddr** values 0 through 3, the function will return the voltage read if the read is successful. For **eioAddr** values 16 through 19, the function returns the 12-bit raw data value read from the A/D converter if the read is successful.

ERROR: Sets bit 1 of **eioErrorCode** if **eioAddr** is out of range.

LIBRARY

EZIOCMN.LIB

EXAMPLE

```
void main (void) {
    int x;
    eioBrdInit(0); // reset all LP3100 resources to default states
    lp31ADCInit(); // turn on, reset, and reconfigure
                  // analog input section
    x = (int)eioBrdAI(16);
                  // read the raw ADC value from AIN0
    printf("\n\rAnalog channel 0 = %d",x);
    lp31ADCDis(); // shut down analog section
}
```

void eioBrdInit(int flags)

Sets all LP3100 hardware resources to their default states.

The following tasks are performed by `eioBrdInit`.

- Analog section powered up
- ADC reset
- DINOUT output latch set to 0xFF
- DOUT output latch set to 0xFF
- Stored ADC calibration constants are read

Call `eioBrdInit` whenever the LP3100 is recovering from a VCC power off. For `eioBrdDO` to perform correctly, the digital hardware resources must be synchronized with a software mask. `eioBrdInit` does this synchronization.

PARAMETER

`flags` is not used at this level and should be set to 0.

LIBRARY

`EZIOCMN.LIB`



Call `eioBrdInit` before calling `eioBrdAI`.

EXAMPLE

```
void main (void) {
    eioBrdInit(0); // reset all LP3100 resources to default states
    printf("\n\rYou are in a twisted maze of passages.");
}
```

int eioBrdACalib(int eioAddr, unsigned int d1, unsigned int d2, float v1, float v2)

Computes and stores calibration constants for an analog input channel. The calibration constants are stored in simulated EEPROM space allocated out of the onboard flash EPROM.

The calibration constants allow `eioBrdAI` to return data values in engineering units rather than simply as raw analog-to-digital converter codes. For example, a temperature sensor that produces a voltage output may be connected to an analog channel. The channel may be calibrated in units of temperature to simplify the measurement process.

PARAMETERS

`eioAddr` is the analog input channel (0–3).

`d1` is the raw data corresponding to `v1` ($0 \leq d1 \leq 65,535$)

`d2` is the raw data corresponding to `v2` ($0 \leq d2 \leq 65,535$)

`v1` (any `float`) is the known SI unit (voltage) corresponding to `d1`

`v2` (any `float`) is the known SI unit (voltage) corresponding to `d2`

RETURN VALUE

0 if successful, -1 if `eioAddr` is out of range.

LIBRARY

EZIOLP31.LIB



Since the LP3100 is factory-calibrated for volts, it is only necessary to use this function to recalibrate the LP3100 for other units.

EXAMPLE

```
#use eziolp31.lib
#define AINPUT 0          // adc input 0
#define LOVOLT 0.0       // voltage for first reading
#define HIVOLT 9.0       // voltage for second reading
#define SETRAW 16        // offset to get raw data

void AnyKeyPress (void);

main (){
    auto unsigned rawdata;
    auto unsigned data1, data2;
    float volt_equ, volt1, volt2;
    auto unsigned inputnum;

    VdInit();
    eioBrdInit(0);
    inputnum = AINPUT;
    volt1 = LOVOLT;
    volt2 = HIVOLT;

    // get raw data from two known voltages

    printf("Adjust for voltage to %.2fV for first reading
           and press spacebar to continue\n", LOVOLT);
    AnyKeyPress();
    data1 = eioBrdAI(inputnum+SETRAW);

    printf("Adjust for voltage to %.2fV for second reading
           and press spacebar to continue\n", HIVOLT);
    AnyKeyPress();
    data2 = eioBrdAI(inputnum+SETRAW);

    // define gain and offset and store in memory

    eioBrdACalib(inputnum, data1, data2, volt1, volt2);

    // use the stored values to compute equivalent voltage

    while (1){
        rawdata = eioBrdAI(inputnum+SETRAW);
        volt_equ = eioBrdAI(inputnum);
        printf("Voltage at AIN%d is %.2f from raw data
              %d\n", inputnum, volt_equ, rawdata);
    }
}

void AnyKeyPress (void){
    while (!kbhit())
        hitwd();
    getchar();
}
```

4.4 Serial Communication Functions

There are several libraries that can be used for serial communication. Serial communication functions, constants, and definitions are described in the *Dynamic C Function Reference Manual* and the *Dynamic C Application Frameworks Manual*. `AASC.LIB` is the recommended serial communication library.

4.4.1 RS-485 Functions

`void on_485(void)`

Turns on the RS-485 transmitter for Z180 Port 1. This must be done before sending data on an RS-485 network.

LIBRARY

`DRIVERS.LIB`

EXAMPLE

```
void main (void) {
    eioBrdInit(0); // reset all LP3100 resources to default states
    on_485();      // turn on RS-485 transmitter
}
```

`void off_485(void)`

Turns off the RS-485 transmitter for Z180 Port 1. This needs to be done to allow other devices on the RS-485 network to send data.

LIBRARY

`DRIVERS.LIB`

EXAMPLE

```
void main (void) {
    eioBrdInit(0); // reset all LP3100 resources to default states
    off_485();     // turn off RS-485 transmitter
}
```

4.4.2 RS-232 Functions

`void on_232(void)`

The macro `on_232` turns on the LTC1385 charge pump and enables the RS-232 drivers. Be sure to use this function before sending or receiving data via an RS-232 port. Shutting down the charge pump will reduce the current draw by 70 μ A and conserve a small amount of battery power.

LIBRARY

`EZIOLP31.LIB`

The RS-232 drivers are on by default when the LP3100 is in program mode, and are off when the LP3100 is in run mode. The RS-232 drivers must be turned on if the RS-232 channels are going to be used when the LP3100 is in run mode.

EXAMPLE

```
void main (void) {
    eioBrdInit(0); // reset all LP3100 resources to default states
    on_232();      // turn on RS-232 charge pump
}
```

`void off_232(void)`

The macro `off_232` turns off the LTC1385 charge pump and disables the RS-232 drivers, which draw 70 μ A. If no RS-232 communication is needed, use `off_232` to shut down the RS-232 circuits and save power.

LIBRARY

`EZIOLP31.LIB`

EXAMPLE

```
void main (void) {
    eioBrdInit(0); // reset all LP3100 resources to default states
    off_232();     // turn off RS-232 charge pump
}
```

4.5 Power Control Functions

`void lp31ADCDis(void)`

Turns off power to the analog section (AVCC) of the LP3100. The analog section draws about 1.1 mA.

When analog measurements are not being made, the analog section should be turned off to conserve power.

The preferred function to turn on and initialize the analog section is `lp31ADCInit`.

LIBRARY

`EZIOLP31.LIB`

EXAMPLE

```
void main (void) {
    int x;
    eioBrdInit(0); // reset all LP3100 resources to default states
    lp31ADCInit(); // turn on, reset, and reconfigure
                  // analog input section
    x = (int)eioBrdAI(16);
                // read the raw ADC value from AIN0
    printf("\n\rAnalog channel 0 = %d",x);
    lp31ADCDis(); // shut down analog section
}
```

`void sysHalt(void)`

Puts the Z180 into HALT mode. In HALT mode, the Z180 continuously fetches, but does not execute the instruction after the halt instruction. The Z180 clock continues running at full speed. The Halt(L) processor signal is held low. All on-chip I/O and DMA are enabled. Interrupts are active.

HALT is not a useful power-saving tool because the Z180 continues to run at full speed.

The Z180 can recover (wake up) from the HALT mode by a reset, an NMI, an on-chip IRQ, or an external IRQ.

LIBRARY

`SYS.LIB`

EXAMPLE

```
#use sys.lib
void main (void) {
    eioBrdInit(0); // reset all LP3100 resources to default states
    sysHalt();    // Enter HALT mode
    while(1)     // Get here only after wake-up event occurs
        hitwd();
}
```

void sysSleep(void)

Puts the Z180 into SLEEP mode. In SLEEP mode, the Z180 clock continues to run, but is blocked from the CPU core. All Z180 signals are held high, except Halt(L), which is held low. SLEEP is a useful power-saving tool.

The Z180 can recover (wake up) from the SLEEP mode by a reset, an NMI, or an external IRQ.

LIBRARY

SYS.LIB

EXAMPLE

```
#use sys.lib
void main (void) {
    eioBrdInit(0); // reset all LP3100 resources to default states
    sysSleep();   // Enter SLEEP mode
    while(1)      // Get here only after wake-up event occurs
        hitwd();
}
```

void sysStandby(void)

Puts the Z180 into STANDBY mode. In STANDBY mode, the Z180 clock is stopped and on-chip peripherals are turned off. This is the lowest power Z180 operating mode available.

The Z180 can recover (wake up) from the STANDBY mode by a reset, an NMI, or an external IRQ.

LIBRARY

SYS.LIB

EXAMPLE

```
#use sys.lib
void main (void) {
    eioBrdInit(0); // reset all LP3100 resources to default states
    sysStandby(); // Enter STANDBY mode
    while(1)      // Get here only after wake-up event occurs
        hitwd();
}
```

```
void lp31Shutdown( int enbKS, int enbWE )
```

This function shuts down the LP3100 (VCC is turned off). The Z180, flash EPROM, analog section, serial drivers, and digital I/O are all unpowered. The SRAM and RTC may be backed up by an external 3 V battery. The RTC may be programmed to draw its current from VCCU instead of the external battery. The DC current consumed in the shutdown mode is typically 300 μ A to 400 μ A. Note that this function does not return.

VCC may be restored in one of four ways based on the parameters **enbKS** and **enbWE**.

PARAMETERS

If **enbKS** is non-zero, pulling DIN0 low or an interrupt on the LPBus will restore VCC and reset the Z180.

If **enbWE** is non-zero, the RTC wakeup feature is enabled. Be sure to set up an alarm event (see **rtcSetAlmTime**) before shutting down VCC.

A third way to restore VCC is to remove and restore DCIN. This will always turn on VCC and reset the Z180.

The **PB_RST** signal on pin 40 of header H2 can be used to remove and restore DCIN when pin 40 is connected to a pushbutton reset like the one provided on the LP3100 Development Board.

LIBRARY


EZIOLP31.LIB

EXAMPLE

```
void main (void) {  
    eioBrdInit(0); // reset all LP3100 resources to default states  
    lp31Shutdown(); // Enter SHUTDOWN mode  
    while(1) // Never gets here—lp31Shutdown never returns  
        hitwd(); // On wake-up, execution starts at top of main()  
}
```

void lp31PFO(char onOff)

Enables or disables the power-fail interrupt (PFI). PFI is disabled on power-up. PFI must be enabled each time VCC is turned on if the PFI feature is being used. If the PFI is enabled, the power-fail output (PFO) from the ADM696 supervisor chip will trigger the PFI when DCIN drops below the trip point. The time available for the interrupt service routine to perform an orderly shutdown is determined by how fast DCIN drops.

 Do not enable the PFI if the nominal DCIN < 4.5 V. The DCIN trip point can be anywhere from 3.7 V to 4.5 V because of component tolerances (mainly the ADM696 supervisor chip).

PARAMETER

If **onOff** is zero, the PFI is disabled. Otherwise the PFI is enabled for $0 < \text{onOff} \leq 255$.

LIBRARY

EZIOLP31.LIB

EXAMPLE

```
void main (void) {
    eioBrdInit(0); // reset all LP3100 resources to default states
    lp31PFO(1);    // Enable power-fail interrupt
}
```

unsigned int lp31Clk3MHz(void)

Changes the CPU clock speed to 3 MHz, assuming an external 6 MHz crystal is in place. The Z180 has an internal clock divider that is under software control. This function invokes the Z180 clock divider. Running the CPU at half speed reduces the CPU current draw.

If the LP3100 is in program mode, **lp31Clk3MHz** will adjust the internal baud rate generator for Serial Port Z0 so as not to disrupt communication with the Dynamic C debugger.

If the LP3100 is in run mode, **lp31Clk3MHz** will *not* adjust any of the serial port parameters. For standalone applications, be sure to initialize the serial ports after the processor speed is set.

RETURN VALUE

0 if no change was made (the CPU was already running at 3 MHz); non-zero if the CPU speed was changed to 3 MHz.

LIBRARY

EZIOLP31.LIB

EXAMPLE

```
void main (void) {
    eioBrdInit(0); // reset all LP3100 resources to default states
    lp31Clk3MHz(); // Switch CPU clock speed to 3 MHz
}
```

unsigned int lp31Clk6MHz(void)

Changes the CPU clock speed to 6 MHz, assuming an external 6 MHz crystal is in place. The Z180 has an internal clock divider that is under software control. This function disables the Z180 clock divider.

If the LP3100 is in program mode, **lp31Clk6MHz** will adjust the internal baud rate generator for Serial Port Z0 so as not to disrupt communication with the Dynamic C debugger.

If the LP3100 is in run mode, **lp31Clk6MHz** will *not* adjust any of the serial port parameters. For standalone applications, be sure to initialize the serial ports after the processor speed is set.

RETURN VALUE

0 if no change was made (the CPU was already running at 6 MHz); non-zero if the CPU speed was changed to 6 MHz.

LIBRARY

EZIOLP31.LIB

EXAMPLE

```
void main (void) {
    eioBrdInit(0); // reset all LP3100 resources to default states
    lp31Clk6MHz(); // Switch CPU clock speed to 6 MHz
}
```


4.6 RTC Functions

```
void rtcInit( void )
```

Initializes the RTC in 24-hour mode. Call this function once before using the RTC.

LIBRARY

EZIOLP31.LIB

```
int tm_wr( struct tm *t )
```

Sets the RTC date, day of the week, and time. The tm structure is defined as follows.

```
struct tm {  
    char tm_sec;           // seconds 0-59  
    char tm_min;         // 0-59  
    char tm_hour;        // 0-59  
    char tm_mday;        // 1-31  
    char tm_mon;         // 1-12  
    char tm_year;        // 00-150 (1900-2050)  
    char tm_wday;        // 0-6 0==Sunday  
};
```

This structure is defined in the library **DRIVERS.LIB**. Do not redefine this structure in a program.

PARAMETER

Structure that will hold the time/date data.

RETURN VALUE

0 if successful; -1 if clock fails or is not installed.

LIBRARY

DRIVERS.LIB

```
int tm_rd( struct tm *t )
```

Reads the current system time into the structure **tm**.

PARAMETER

Structure that will hold the time/date data.

RETURN VALUE

0 if successful; -1 if clock fails or is not installed.

LIBRARY

DRIVERS.LIB

```
void rtcSetAlmTime( struct tm *t )
```

Sets the alarm for the RTC. Note that this function does not enable the interrupt from the RTC or enable alarm interrupts. See `rtcIRQ` and `rtcClrIRQ` for more information about creating and handling interrupts generated by the RTC.

In order for the RTC to wake up the system, the application must assert wake up interrupt enable and clear the wakeup flag before powering down. See `lp31Shutdown` for more details about shutting down the main power of the LP3100.

PARAMETER

Pointer structure that specifies the alarm time.

LIBRARY

`EZIOLP31.LIB`

```
void rtcSwAIE( int sw )
```

Enables or disables the alarm.

PARAMETER

If `sw` is 0, the alarm is disabled. If `sw` is non-zero, the alarm is enabled.

LIBRARY

`EZIOLP31.LIB`

```
int rtcChkAF( void )
```

Returns the alarm enable/disable status.

RETURN VALUE

0 if the alarm is disabled; non-zero otherwise.

LIBRARY

`EZIOLP31.LIB`

```
void rtcClrIRQ()
```

Clears all the interrupt flags that can cause the /IRQ line to assert. Note that this function does not enable or disable the interrupt from each source. Call

- `rtcSwAIE(x)` for alarm interrupt enable
- `rtcSwPIE(x)` for periodic interrupt enable
- `rtcSwUIE(x)` for update-ended interrupted enable
- `rtcSwWIE(x)` for wakeup interrupt enable
- `rtcSwKSE(x)` for kickstart interrupt enable
- `rtcSwRIE(x)` for RAM clear interrupt enable

In all these routines, pass non-zero to `x` to enable and pass zero to disable. See `rtcIRQ` for more information about causing and handling interrupts generated by the RTC.

LIBRARY

`EZIOLP31.LIB`

void rtcIRQ(int enable)

This function enables the Z180 to receive IRQ from the RTC. Mode 1 on INTO is supported. The programmer should write the custom ISR (interrupt service routine) and instruct Dynamic C to vector it using the following directive.

```
#JUMP_VEC RST38_VEC myInt0ISR
```

where *myInt0ISR* is the actual ISR name.

The ISR should return with **reti** instead of **ret**.

In the interrupt routine, use the following macros to check the cause of the interrupt.

```
rtcChkAF() returns the alarm flag  
rtcChkKF() returns the kickstart flag  
rtcChkPF() returns the periodic flag  
rtcChkRF() returns the RAM clear flag  
rtcChkUF() returns the update-ended flag  
rtcChkWF() returns the wakeup flag
```

Note that the periodic flag, alarm flag, and update-ended flag reside in the same physical register C. If more than one flag needs to be tested, the program should read register C to a variable using **rtcRdRegC()**, then examine bits 4, 5, and 6 respectively. Reading register C automatically clears all of the periodic, alarm, and update-ended flags.

To clear the kickstart, RAM clear, and wakeup flags, call the following macros.

```
rtcSwKF(x) switches the kickstart flag  
rtcSwRF(x) switches the RAM clear flag  
rtcSwWF(x) switches the wakeup flag
```

Pass 0 for **x** to turn the flag off and 1 to turn it on.

Reading register C via **rtcRdRegC()** will clear the alarm flag, periodic flag, and update-ended flag.

To enable a particular source of interrupt, call the following macros.

```
rtcSwAIE(x) for alarm interrupt enable  
rtcSwPIE(x) for periodic interrupt enable  
rtcSwUIE(x) for update-ended interrupt enable  
rtcSwWIE(x) for wake up interrupt enable  
rtcSwKSE(x) for kickstart interrupt enable  
rtcSwRIE(x) for RAM clear interrupt enable
```

In all these routines, pass non-zero to **x** to enable and zero to disable.



The /IRQ line remains asserted (low) as long as both the interrupt enable flag and the status flag are asserted for any of the six causes (alarm, periodic, update-ended, wake up, kickstart, and RAM clear). The IRS must ensure that /IRQ is not asserted when the ISR returns.

LIBRARY

EZIOLP31.LIB

4.7 Flash EPROM Functions

```
int WriteFlash( unsigned long physical_addr,  
char *buf, int count )
```

Writes **count** number of bytes pointed to by **buf** to the flash EPROM absolute data location **physical_addr**. Allocate the data location by declaring the byte arrays as initialized arrays or declare an initialized **xdata** array. If a byte array is declared, convert the logical memory to physical memory with **phy_adr(array)**. For initialized **xdata**, the array name can be passed directly.

PARAMETERS

physical_addr is the absolute data location in flash EPROM.

buf is a pointer to the bytes to write.

count is the number of bytes to write.

RETURN VALUE

0 if **WriteFlash** is successful.

-1 if flash EPROM is not used.

-2 if **physical_addr** is inside the BIOS area.

-3 if **physical_addr** is within the symbol area or the simulated EEPROM area.

-4 if the write to the flash EPROM times out.

LIBRARY

DRIVERS.LIB

4.8 LED Functions

```
int swLED( unsigned int state )
```

Turns the LED on or off.

PARAMETER

state is 1 to turn the LED on, 0 to turn it off.

RETURN VALUE

0 if state is valid, -1 otherwise.

LIBRARY

EZIOLP31.LIB

4.9 LCD Functions

```
void lcdEnCur( void )
```

Enables cursor.

LIBRARY

TL.LIB

```
void lcdDisCur( void )
```

Disables cursor.

LIBRARY

TL.LIB

```
void lcdEnBlink( void )
```

Enables blinking of the LCD cursor.

LIBRARY

TL.LIB

```
void lcdDisBlink( void )
```

Disables the blinking of the LCD cursor.

LIBRARY

TL.LIB

```
void lcdPos( char row, char col )
```

Positions the cursor on the LCD.

PARAMETERS

row is the row of the cursor (starts with 0).

col is the column of the cursor (starts with 0).

LIBRARY

TL.LIB

```
int lcdVprintf( char fmt, void *firstArg )
```

Equivalent to **vprintf**, but prints to the current cursor position on the LCD .

PARAMETERS

fmt is the format string. Note that tabs, line feeds, returns and other special characters will not be interpreted by the LCD.

firstArg points to the address of the first argument.

RETURN VALUE

0 if the string is sent successfully, -1 if the LCD times out.

LIBRARY

TL.LIB

```
int lcdPrintf( char fmt, ... )
```

Equivalent to `printf`, but prints to the current cursor position on the LCD.

PARAMETER

`fmt` is the format string. Note that tabs, linefeeds, returns and other special characters will not be interpreted by the LCD.

RETURN VALUE

0 if the string is sent successfully, -1 if the LCD times out.

LIBRARY

`TL.LIB`

```
int lcdLongWait(void)
```

Similar to `lcdWait`, but waits for four times as long for the longest operation on the LCD.

RETURN VALUE

0 if the LCD is free, -1 if the LCD times out.

LIBRARY

`TL.LIB`

```
int lcdInit( void )
```

Initializes the LCD hardware and library. Blanks the screen and sets cursor to blink on the first row at the first column.

RETURN VALUE

0 if the initialization is successful, -1 if the LCD times out.

LIBRARY

`TL.LIB`

```
int lcdRead( void )
```

Reads the status of the LCD.

RETURN VALUE

The status of the LCD. If the seventh bit is set, the LCD is not free yet.

LIBRARY

`TL_LP31.LIB`

```
void lcdWrite( char ch )
```

Writes one character to the LCD control register.

PARAMETER

`ch` is the bytes to write to the LCD control register.

LIBRARY

`TL_LP31.LIB`

```
int lcdWait( void )
```

Waits for LCD to be done. Times out if LCD is not responding.

RETURN VALUE

0 if all is fine (LCD is not free), -1 if LCD is always busy (timed out).

LIBRARY

TL_LP31.LIB

```
int lcdCtrl( char ch )
```

Sends a character to the LCD control register. This function waits for the LCD becomes free before sending the control byte.

PARAMETER

ch is the character to send.

RETURN VALUE

0 if all is fine, -1 if the LCD is always busy (timed out). Note that the character is sent even if the LCD times out.

LIBRARY

TL_LP31.LIB

4.10 Keypad Functions

```
void kpInit( int(*changeFn)() )
```

Initializes the keypad module. This function should be called before other functions of this module are called.

PARAMETER

changeFn points to a function that will be called when the driver detects a change (when **kpScanstate** is called). Two arguments are passed to the callback function. The first argument is a pointer to an array that indicates the current state of the keypad. The second pointer points to an array that indicates what keypad positions are changed and detected by **kpScanstate**. The byte offset in the array represents the line pulled high (row number), and the bits in a byte represents the positions (column number) read back.

LIBRARY

KP.LIB

```
int kpScanstate( void )
```

Scans the keypad and detects any changes to the keypad status. It returns non-zero if there is any change. If **kpInit** is called with a non-NULL function pointer, that function will be called with the state of the keypad. This function should be called periodically to scan for keypad activities.

RETURN VALUE

0 if there is no change to the keypad, non-zero if there is any change to the keypad.

LIBRARY

KP.LIB

```
int kpDefStChgFn( char *curState, char *changed )
```

This function is the default state change function for the default get key function **kpDefGetKey**. This function is called back by **kpScanstate** when there is a change in the keypad state. If the current key is not read by **kpDefGetKey**, the new key pressed will not be registered.

PARAMETERS

curState points to an array that holds the current state of the keypad (bitmapped, 1 indicates key is not currently pressed).

changed points to an array that reflects the change of keypad state from the previous scan (bitmapped, 1 indicates there was a change).

RETURN VALUE

-1 if no key is pressed. Otherwise it returns the normalized key number. The normalized key number is $8*\text{row}+\text{col}+\text{edge}*256$. **edge** is 1 if the key is released, and 0 if the key is pressed.

LIBRARY

KP.LIB

```
int kpDefGetKey(void)
```

This function is the default get key function. It returns the key previously pressed (i.e., from the one-keypress buffer). The key pressed is actually interpreted by **kpDefStChgFn**, which is called back by **kpScanstate**. **kpDefInit** should be used to initialize the module.

RETURN VALUE

-1 if no key is pressed. Otherwise it returns the normalized key number. The normalized key number is $8*\text{row}+\text{col}+\text{edge}*256$. **edge** is 1 if the key is released, and 0 if the key is pressed.

LIBRARY

KP.LIB

`int kpDefInit(void)`

This function initializes the module to use the default state change function to interpret key presses when `kpScanstate` is called. Use `kpDefGetKey` to get the code of the last key pressed.

LIBRARY

`KP.LIB`

4.11 Additional Software

- For watchdog information, refer to descriptions of the function `hitwd` in the Dynamic C manuals.
- For reading and writing simulated EEPROM data, refer to descriptions of the functions `ee_rd` and `ee_wr` in the Dynamic C manuals.
- For power fail flag information, refer to the descriptions of the functions `_sysIsPwrFail` and `sysIsPwrFail` in the Dynamic C manuals.
- For resetting the board information, refer to descriptions of the functions `sysForceSupRst`, `sysIsSuperReset`, `_sysIsSuperReset`, `sysForceReset`, `_sysIsWDTO`, and `sysIsWDTO` in the Dynamic C manuals.
- If an error message such as **Target Not Responding** or **Communication Error** appears, see Appendix A, “Troubleshooting.”



APPENDIX A. TROUBLESHOOTING

Appendix A provides procedures for troubleshooting system hardware and software.

A.1 Out of the Box

Check the items mentioned in this section before starting development.

- Do not connect any boards to the LPBus, RS-485, or any other I/O devices until verifying that the LP3100 runs standalone. If using the RS-232 channel on the Development Board, make sure that the Development Board is aligned correctly.
- Verify that the entire system has a good, low-impedance ground. The LP3100 is often connected between the PC and other devices. Any differences in ground potential from unit to unit can cause serious, hard-to-diagnose problems.
- Use the supplied Z-World cables. Double check the connecting cables to make sure they are properly connected.
- Verify that the PC's COM port actually works. Try connecting a known-good serial device to the COM port. Remember that COM1/COM3 and COM2/COM4 on a PC share interrupts. User shells and mouse software often interfere with proper COM-port operation. For example, a mouse running on COM1 may prevent running Dynamic C on COM3 unless the interrupt is changed.
- Use the supplied Z-World power supply. Verify that the power supply has enough capacity to support the LP3100 plus any attached devices and is adequately filtered.
- Experiment with each peripheral device connected to the LP3100 to determine how it affects LP3100 operation when it is powered up, powered down, when its connecting wiring is open, and when its connecting wiring is shorted.

A.2 Dynamic C Does Not Start

If Dynamic C does not start, an error message will usually appear on the Dynamic C screen (for example, "Target Not Responding" or "Communication Error"), announcing a communication failure.

There could be one or more of the following problems in series:

- The wrong COM port is selected.
- The wrong baud rate is selected. The LP3100 has a maximum baud rate of 38,400 bps. Make sure that the baud rate setting in Dynamic C is 9600 bps, 19,200 bps, or 38,400 bps when using the RS-232 port for programming. If the SIB2 is used for development, use one of the following baud rates: 9600 bps, 19,200 bps, 28,800 bps, or 57,600 bps. Choose the fastest setting first. Try lower baud rates if communication attempts fail.
- The LP3100 needs to be reset. Press the reset switch on the development board or cycle power.
- Serial communication wiring is incorrect or improperly connected.

The first thing to check is the hardware and software setup of the PC's COM port. Check the following two items.

1. Make sure that all wiring and cables are connected properly.
2. Make sure that the correct COM port is selected.

Most PCs have at least two COM ports (COM1 and COM2), and some computers have additional COM ports. Sometimes a PC assigns COM1 or COM2 to an internal modem, leaving the other COM port available on the back of the PC.



Some computers have special programs to reconfigure the port assignments. If necessary, run such a program to make a specific COM port appear at an external back-panel "D" connector.

Repeat the following procedure until a COM port that works with Dynamic C and your LP3100 is found.

1. Use the Serial command of Dynamic C's Options menu to try a different COM port.
2. Reset the LP3100 by pressing the reset switch SW1.
3. Select **Reset Target** from the **Run** menu. Dynamic C will try to establish communication again.

A.3 LP3100 Repeatedly Resets

If the program fails to hit the watchdog timer periodically, the watchdog timer causes a reset approximately every second. Dynamic C hits the watchdog timer when debugging a program using the Dynamic C debugger. If the program loaded in the LP3100 does not hit the watchdog timer, the watchdog will reset the board and the program will restart. (To hit the watchdog, make a call to the Dynamic C library function `hitwd`).

If there is no battery is connected to the LP3100, it may be necessary to short the VBAK line to ground. This can be done by shorting out the two pins on H1.

A.4 Dynamic C Loses Link with Application Program

Dynamic C may lose its link with the LP3100 if the program disables interrupts for more than 50 ms.



APPENDIX B. SPECIFICATIONS

Appendix B provides comprehensive LP3100 physical, electronic, and environmental specifications.

B.1 General Specifications

Table B-1 lists the electrical, mechanical, and environmental specifications for an LP3100.

Table B-1. LP3100 General Specifications

Parameter	Specification
Board Size	2.5" W × 3.5" L × 0.5" H (64 mm × 89 mm × 13 mm)
Operating Temperature	-40°C to +70°C
Humidity	5% to 95%, noncondensing
Input Voltage and Current	3.5 V to 24 V DC, 200 μA standby, 16 mA to 24 mA (19 mA typ) at 6.144 MHz, 11 mA typical at 3.072 MHz linear regulators
Digital Inputs	4 (accept 3.3 V or 5.0 V logic signals)
Digital Outputs	8 (generate 3.3 V or 5.0 V logic signals, jumper selectable)
Configurable I/O	8 digital lines form a bidirectional parallel I/O port (byte-wide programmable)
Analog Inputs	4 conditioned 12-bit inputs, default range 0 V to 10 V
Microprocessor	Z180
Clock	6.144/3.072 MHz, software selectable
SRAM	32K-512K
Flash EPROM	128K-512K
Serial Ports	One RS-232 (with or without RTS/CTS) configurable as RS-232 (3-wire) or RS-485
Serial Rate	Up to 38,400 bps
Watchdog/Supervisor	Yes
Time/Date Clock	Yes
Backup Battery	Connection provided for external 3 V battery

Tables B-2 and B-3 list the specifications for the digital inputs and outputs.

Table B-2. Digital Input Specifications

Parameter	Specification
Number of Inputs	4
3.3 V Compatible	Yes
5.0 V Compatible	Yes
Type of Input	3.3 V powered by VHC logic
Voltage Threshold Low (max)	1.0 V
Voltage Threshold High (min)	2.3 V
ESD Performance Human Body Model	>2000 V
ESD Performance Machine Model	>200 V

Table B-3. Digital Output Specifications

Parameter	Specification
Number of Outputs	8
3.3 V Compatible	Yes
5.0 V Compatible	Onboard regulator supplies optional 5.0 V to the output latch. This draws an additional 4 mA of quiescent current.
Type of Output	VHCT logic
Output Voltage Low (max) 3.3 V rail, < 50 μ A	0.1 V
Output Voltage High (min) 3.3 V rail, < 50 μ A	3.2 V
Output Voltage Low (max) 5.0 V rail, < 50 μ A	0.1 V
Output Voltage High (min) 5.0 V rail, < 50 μ A	4.5 V
Maximum Output Current (sourcing or sinking)	25 mA
ESD Performance	No specs available at this time

Table B-3. Digital Output Specifications

Parameter	Specification
Bit Addressable	Yes, only using Z-World's drivers. The physical latch is byte addressable only.

The specifications for the eight configurable digital input/outputs are identical to those for the digital inputs and outputs. This byte-wide port can be configured to be a 3.3 V or 5 V output or input port. The inputs/outputs are configured for inputs or outputs as a whole, and are not individually configurable.

B.2 Analog Inputs

Table B-4 lists the specifications for the analog input.

Table B-4. Analog Input Specifications

Parameter	Specification
Number of Inputs	4 conditioned channels
Resolution	12 bits — no missing codes
Effective Resolution	10 bits
Input Range	0 V to 10 V (user-configurable on per channel basis)
Conversion Time	10 μ s
Input Impedance	100 k Ω
Integral Nonlinearity	± 1 LSB
Differential Nonlinearity	± 1 LSB

B.3 Mechanical Specifications

Figure B-1 illustrates the mechanical dimensions of an LP3100.

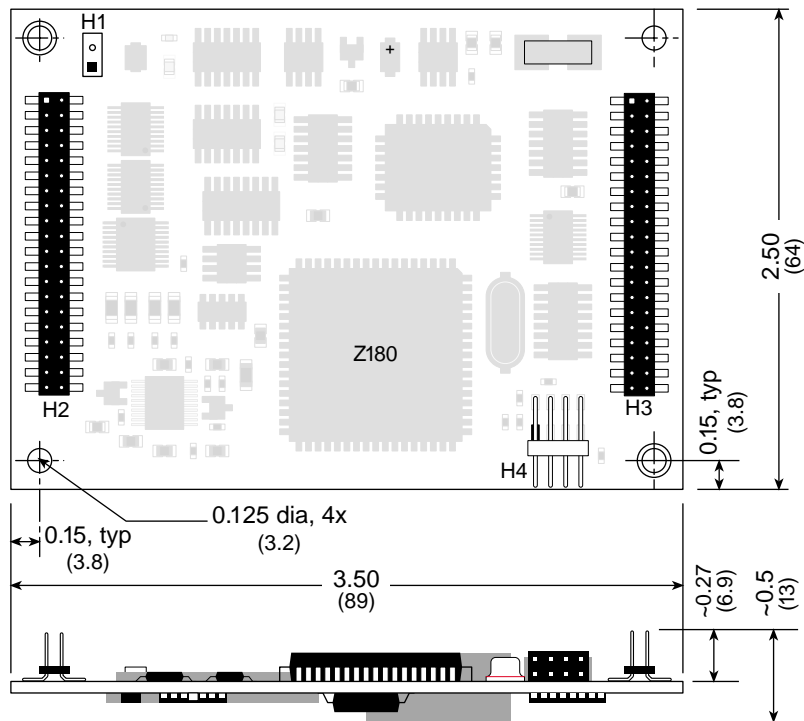


Figure B-1. LP3100 Dimensions

Table B-5 lists the functions of the LP3100 headers and the pin 1 locations.

Table B-5. LP3100 Header Functions and Pin 1 Locations (in inches)

Header	Function	Location
H1	Backup Battery	0.425, 2.200
H2	Application I/O	0.1955, 2.023
H3	LPBus Expansion Port	3.2255, 2.023
H4	SIB Programming Port	N/A

B.3.1 LP3100 Mounting Plate Dimensions

Figure B-2 illustrates the dimensions of an LP3100 aluminum mounting plate.

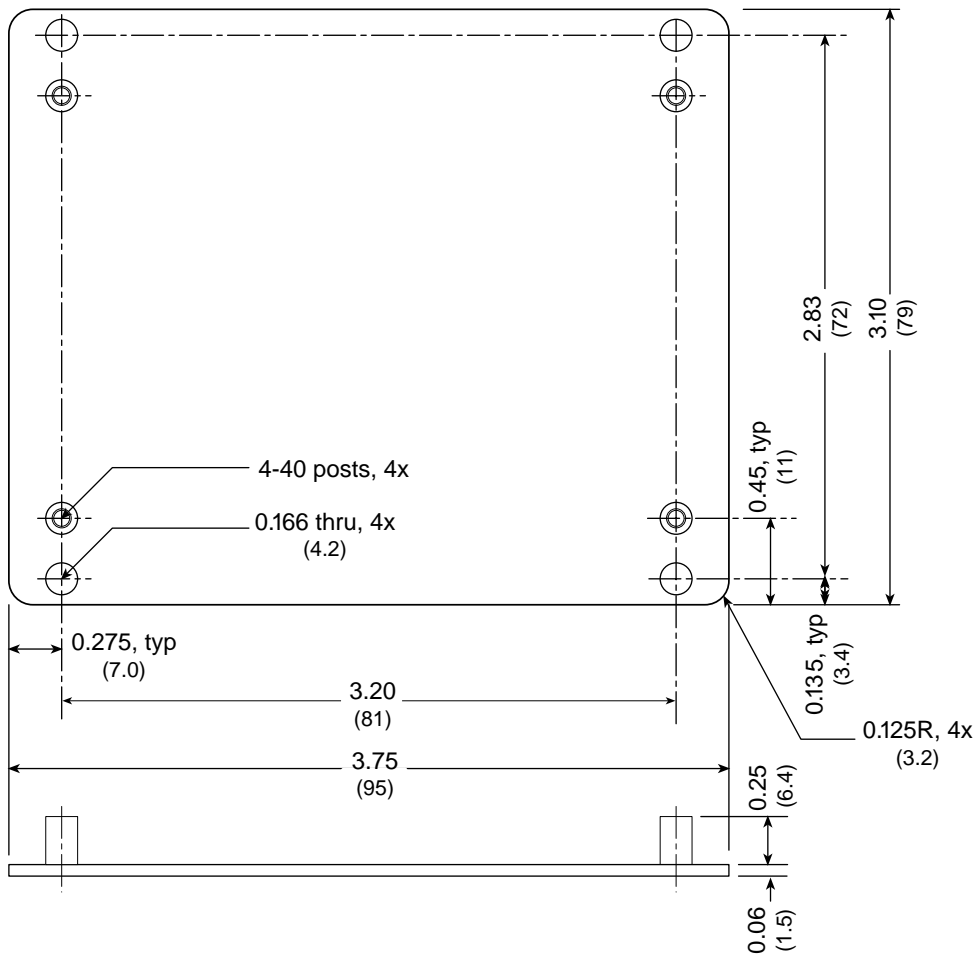


Figure B-2. Aluminum Mounting Plate

B.4 Header Pinouts

The LP3100's digital input/output, analog inputs, serial channels, and LPBus use headers H1, H2, and H3 for physical connection to the outside world. H1 is a 1 × 2 header with 0.100 inch spacing. H2 and H3 are 2 × 20 SMT headers with a 2 mm pin spacing as illustrated in Figure B-3.

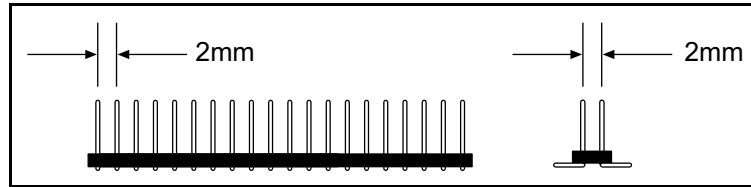


Figure B-3. H2 and H3 Pin Spacing

H1 does not have a header installed on standard LP3100 controllers. This allows the user to solder wires directly to the LP3100 or to install an appropriate header for the backup battery. H1 is able to accommodate a 2 × 1 header with 0.100 inch spacing. Pin 1 of H1 is the positive input for the backup battery. Pin 2 is the ground connection.

Connections to headers H2 and H3 can be made using mass-termination connectors, PCB-mount header receptacles, or pin-insertable housings. Figure B-4 shows the pinouts for headers H2 and H3.

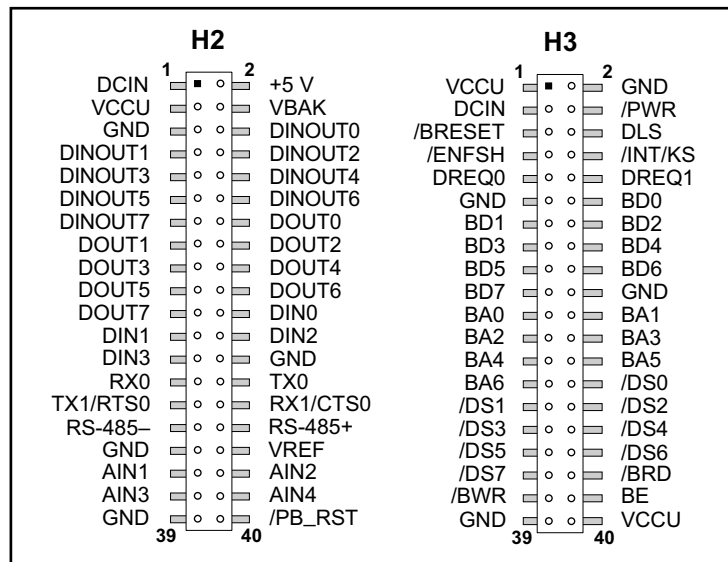


Figure B-4. H2 and H3 Pinouts

2.4.1 LCD Connections

Although there is no dedicated LCD port on the LP3100, the LP3100 can be interfaced to a LCD by using the digital I/O channels. The LP3100 Development Board provides a connector designed to interface to a 2 × 20 LCD. A 4 × 20 LCD can also be attached to the LCD connector on the Development Board, but might require a special cable (refer to the LCD's specifications for wiring information). Z-World provides software drivers that will support 2 × 20 and 4 × 20 LCDs.

The LP3100 supports LCDs with 10- or 11-wire parallel interfaces, using the DOUT and DINOUT pins. The software must simulate a read or write cycle for the LCD using the output latches. Table B-6 lists the wire connections necessary to use the LCDs with Z-World drivers.

Table B-6. LP3100 to LCD Wiring Connections

H2 Pin No.	LP3100 Signal	LCD Module Signal	4 × 20 LCD Module Pin No.	2 × 20 LCD Module Pin No.	2 × 20 LCD Connector* Pin No.
39	GND	GND	1	1	2
2	+5 V	V _{IN}	2	2	1
33	GND	Contrast [†]	3	3	4
21	DOUT7	RS	4	4	3
20	DOUT6	R/W	5	5	6
17	DOUT3	E	6	6	5
6	DINOUT0	D0	7	7	8
7	DINOUT1	D1	8	8	7
8	DINOUT2	D2	9	9	10
9	DINOUT3	D3	10	10	9
10	DINOUT4	D4	11	11	12
11	DINOUT5	D5	12	12	11
12	DINOUT6	D6	13	13	14
13	DINOUT7	D7	14	14	13

* The right-angle header on the 2 × 20 module is numbered in a nonstandard fashion. The odd and even positions are swapped.

† Connect the contrast line to a source between 0 V and 1 V for variable contrast.

B.5 Jumper Settings

Three of the seven jumpers control the digital output latch voltage and the serial channel configuration. Table B-7 lists the jumper configurations.

Table B-7. LP3100 Jumper Settings

Header	Pins 1–2 Connected		Pins 2–3 Connected	
J1	Digital output latch supplied with 3.3 V	FD	Digital output latch supplied with 5.0 V	
J2	Selects /RTS0 for channel Z0, 5-wire RS-232 mode		Selects TXA1 for channel Z1, 3-wire RS-232 mode	FD
J3	Selects RS-485 driver for channel Z1 RXA1, channel Z1 RS-485 mode		Selects RS-232 driver for channel Z0, two channel 3-wire RS-232 mode	FD
J4, J5, J6, J7	All four headers are reserved for Z-World factory defaults.			FD

The headers are located on the bottom side of the LP3100, as shown in Figure B-5. The jumpers are standard 0402 size SMT zero-ohm resistors. Jumper wire or solder may be used.

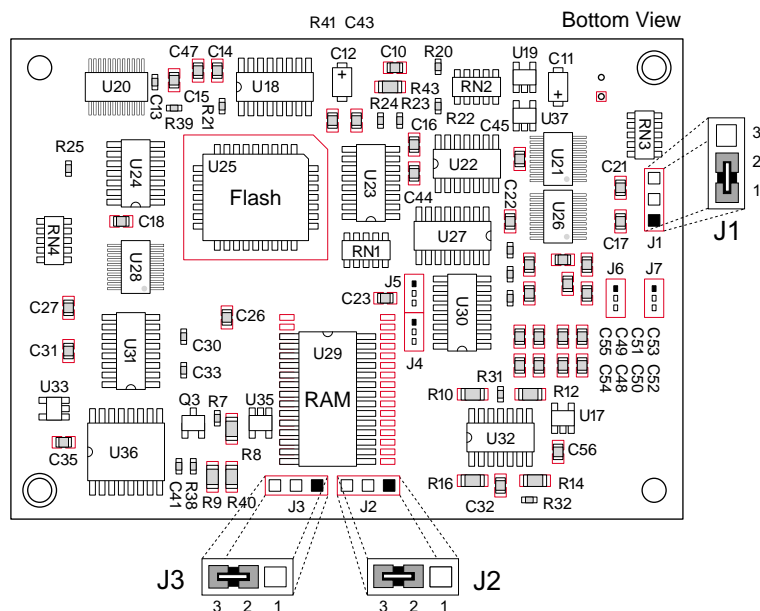


Figure B-5. LP3100 Header Locations



APPENDIX C. SERIAL INTERFACE BOARD

Appendix C provides technical details and baud rate configuration data for Z-World's SIB2.

C.1 Features

The SIB2 is an interface adapter used to program the LP3100. The SIB2 is contained in an ABS plastic enclosure, making it rugged and reliable. The SIB2 enables the LP3100 to communicate with Dynamic C via the Z180's clocked serial I/O (CSI/O) port, freeing the LP3100's serial ports for use by the application during programming and debugging.

The SIB2's 8-pin cable plugs into the target LP3100's processor through an aperture in the backplate, and a 6-conductor RJ-12 phone cable connects the SIB2 to the host PC. The SIB2 automatically selects its baud rate to match the communication rates established by the host PC (9600, 19,200, or 57,600 bps). However, the SIB2 determines the host's communication baud rate only on the first communication after reset. To change baud rates, change the COM baud rate, reset the target LP3100 (which also resets the SIB2), then select **Reset Target** from Dynamic C.



Chapter 2 provides detailed information on connecting the SIB2 to the LP3100.

The SIB2 receives power and resets from the target LP3100 via the 8-pin connector J1. Therefore, do not unplug the SIB2 from the target while power is applied. To do so could damage both the LP3100 and the SIB2; additionally, the target may reset.



Never connect or disconnect the SIB2 with power applied to the LP3100.

The SIB2 consumes approximately 60 mA from the +5 V supply. The target-system current consumption therefore increases by this amount while the SIB2 is connected to the LP3100.

C.2 External Dimensions

Figure C-1 illustrates the external dimensions for the SIB2.

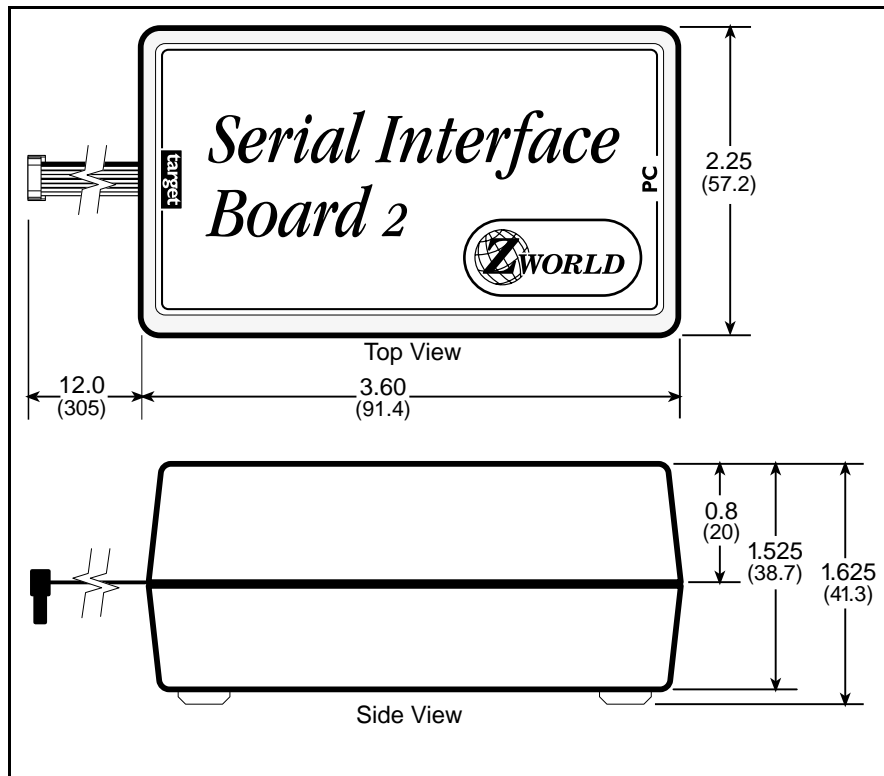


Figure C-1. SIB2 External Dimensions



APPENDIX D. DEVELOPMENT BOARD

D.1 Overview

The LP3100 Development Board provides a quick and easy method of developing and evaluating simple LP3100 applications. The Development Board provides an LCD connector, a keypad connector, two RS-232 channel connectors, an I/O header, a power-supply connector, a reset switch, and a prototyping area. All headers on the LP3100 development board use 0.025" square pins on 0.100" centers.

Figure D-1 illustrates the top view of the LP3100 Development Board and identifies the headers discussed in this appendix.

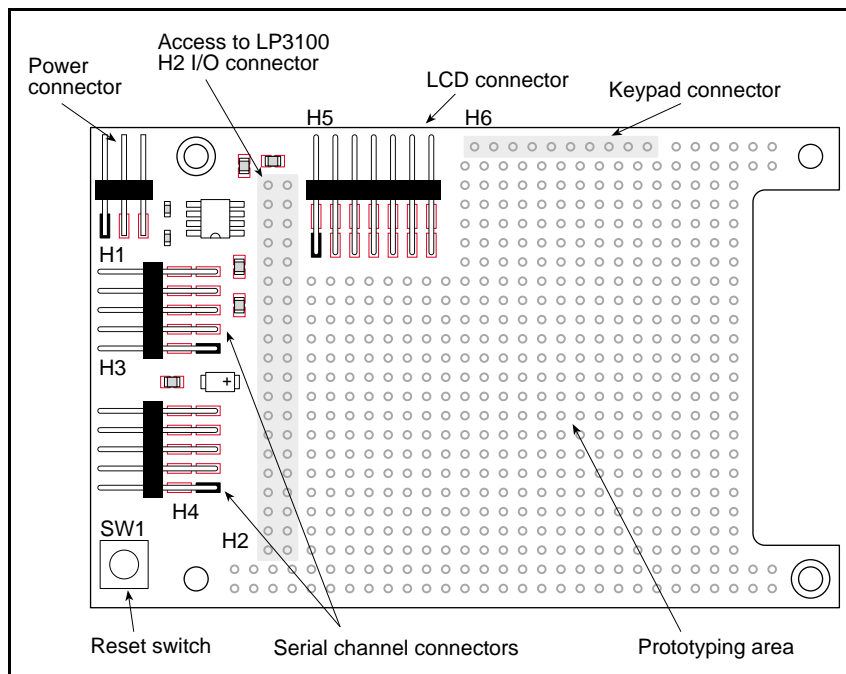


Figure D-1. Development Board Layout

D.1.1 LCD Interface

Table D-1 lists the pinouts for the LCD connector (H5) on the LP3100 Development Board.

Table D-1. LP3100 Development Board LCD Connector

H5 Pin No.	LP3100 Signal	LCD Module Signal
1	+5 V	V _{IN}
2	GND	GND
3	DOUT7	RS
4	N/A	Contrast
5	DOUT3	E
6	DOUT6	R/W
7	DINOUT1	D1
8	DINOUT0	D0
9	DINOUT3	D3
10	DINOUT2	D2
11	DINOUT5	D5
12	DINOUT4	D4
13	DINOUT7	D7
14	DINOUT6	D6



A 2 × 20 LCD can be connected to the Development Board with a 14-conductor ribbon cable.



Drivers for the LCD are listed in Chapter 4, “Software Reference.” Sample programs for the LCD are located in the Dynamic C **SAMPLES\LP31XX** subdirectory.

D.1.2 Keypad Interface

The LP3100 Development Board has a through-hole connector (H6) for connecting a keypad to the LP3100. Table D-2 lists the pinout for H6. The Development Board will support keypads sizes up to 4 rows \times 4 columns.

Table D-2. LP3100 Development Board Keypad Connector

H6 Pin No.	LP3100 Signal	Keypad Signal
1	DIN0	Sense 0
2	DIN1	Sense 1
3	DIN2	Sense 2
4	DIN3	Sense 3
5	DOUT4	Scan 0
6	DOUT5	Scan 1
7	DOUT6	Scan 2
8	DOUT7	Scan 3
9	not connected	N/A
10	not connected	N/A



Drivers for the keypad are listed in Chapter 4, “Software Reference.” Sample programs for the keypad are located in the Dynamic C **SAMPLES\LP31XX** subdirectory.

D.1.3 RS-232 Channel Connectors

The two LP3100 RS-232 channels are brought out on 10-pin headers (H3 and H4) on the Development Board. H4 is connected to Serial Channel Z0. H3 provides connections to either Serial Channel Z1 or the handshaking signals RTS0 and CTS0 from Serial Channel Z0.

Table D-3 lists an LP3100 Development Board's serial channel connections.

Table D-3. LP3100 Development Board Serial Channel Connections

H3 and H4 Pin No.	H3 Signal	H4 Signal
1	not connected	not connected
2	not connected	not connected
3	TX1/RTS0	TX0
4	not connected	not connected
5	RX1/CTS0	RX0
6	+5 V pullup	+5 V pullup
7	not connected	not connected
8	not connected	not connected
9	GND	GND
10	not connected	not connected

D.1.4 I/O Header

The LP3100 Development Board provides access to the LP3100 I/O header H2. The Development Board I/O header (H2) is a 2 × 20 header. The pinout is identical to the pinout for LP3100 header H2 (see Figure D-2).

D.1.5 Power Supply Input Connector

The power supply input connector H1 is used for connecting an external power supply to the LP3100 and to the Development Board. H1 is a three-pin single-row header. Pins 1 and 3 are ground, and pin 2 is input voltage.

D.1.6 Reset Switch

Reset switch SW1 is connected to the PB_RST signal on the LP3100. Pressing the switch causes the PB_RST signal to go low.

D.1.7 Prototyping Area

The Development Board has a prototyping area consisting of a grid of plated through-holes with 0.100" spacing to accommodate standard DIP ICs and through-hole components. Copper areas, covered with a solder mask, surround the holes in the grid. The copper on the top (component) side is connected to a 3.3 V DC power supply. The copper on the bottom side is connected to ground. Connections to the 3.3 V supply or GND can be made by scratching the solder mask and soldering to the copper area.

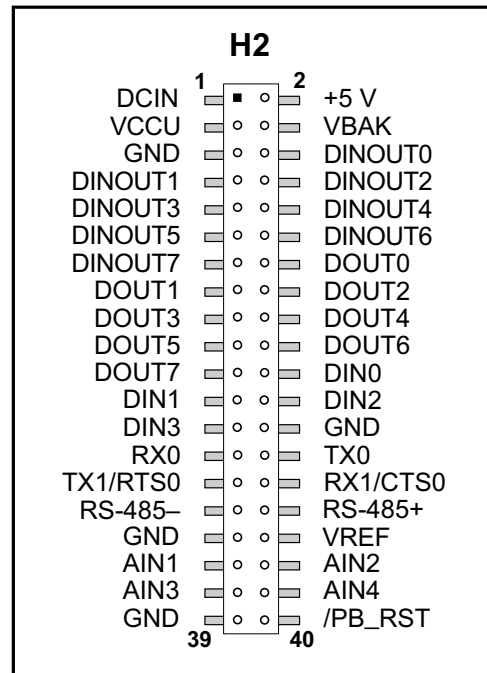
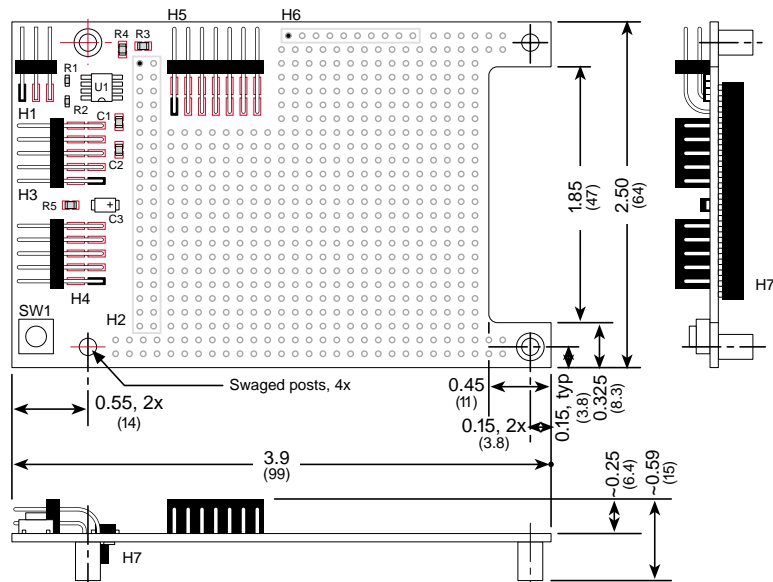


Figure D-2. LP3100 Header H2 Pinout

D.2 Dimensions

Figure D-3 illustrates the dimensions of an LP3100 Development Board.



1. Header H7 is not centered on the board, but is offset by 0.050" (1.3 mm).

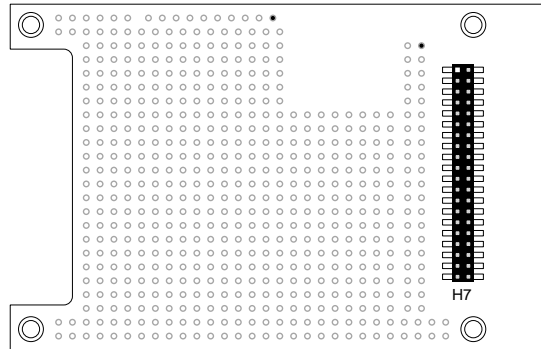


Figure D-3. LP3100 Development Board Dimensions



APPENDIX E. LPBUS PROTOTYPING BOARD

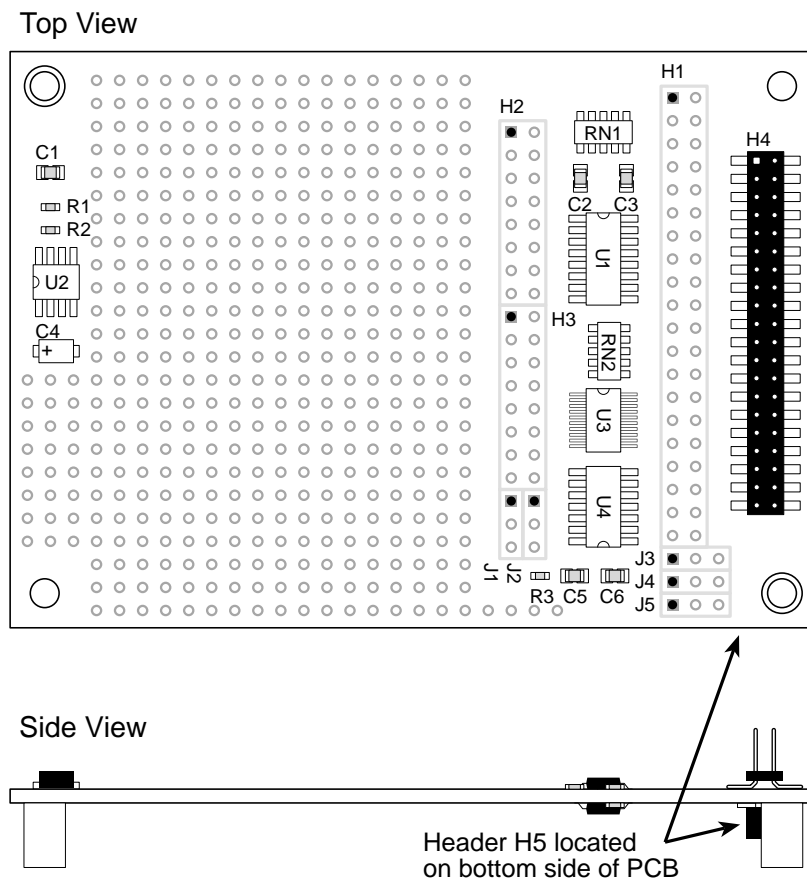
Appendix E provides information about using the LPBus Prototyping Board to design LPBus expansion boards.

E.1 Overview

The LPBus is an expansion bus that provides a simple method of expanding the I/O capability of the LP3100 and other Z-World controllers. Z-World expansion boards or customer designed boards can be easily connected to the LPBus via ribbon cable or stacking connectors. This appendix outlines some of the electrical and mechanical considerations of designing LPBus expansion boards. Following the recommendations in this appendix will increase the reliability of expansion boards and maintain compatibility with other Z-World expansion boards.

The LPBus Prototyping Board provides a quick and easy method of developing and evaluating expansion boards for the LP3100. The LPBus Prototyping Board provides the LPBus signals and a prototyping area.

Figure E-1 illustrates the top view of the LPBus Prototyping Board and identifies the location of the headers with signals that are discussed in this appendix.



Headers H4 and H5 are not centered, but are offset .050" (1.3 mm)

Figure E-1. LPBus Prototyping Board Layout

E.1.1 Installation of LPBus Prototyping Board

Mate the LPBus Prototyping Board to the LP3100. Make sure that header H5 on the bottom side of the Prototyping Board mates with header H3 on the LP3100. Figure E-2 illustrates the correct placement.

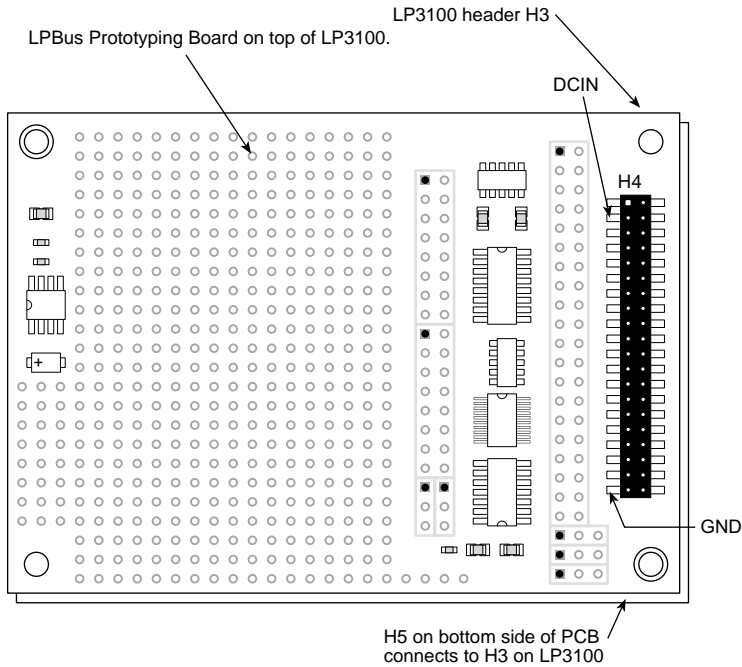


Figure E-2. LPBus Prototyping Board Alignment

DCIN can be connected to pin 3 of header H4 on the LPBus Prototyping Board, and GND can be connected to pin 39 of header H4 as shown in Figure E-2.

E.1.2 Prototyping Area

The Development Board has a prototyping area consisting of a grid of plated through-holes with 0.100" spacing to accommodate standard DIP ICs and through-hole components. Copper areas, covered with a solder mask, surround the holes in the grid. The copper on the top (component) side is connected to a 5 V DC power supply. The copper on the bottom side is connected to ground. Connections to the 5 V supply or GND can be made by scratching the solder mask and soldering to the copper area.

E.1.3 LPBus Signals

The LPBus signals are brought to the LPBus Prototyping Board via headers H5 and H4 from header H3 on the LP3100. Figure E-3 shows the pinout for the various signals available on the LPBus Prototyping Board.

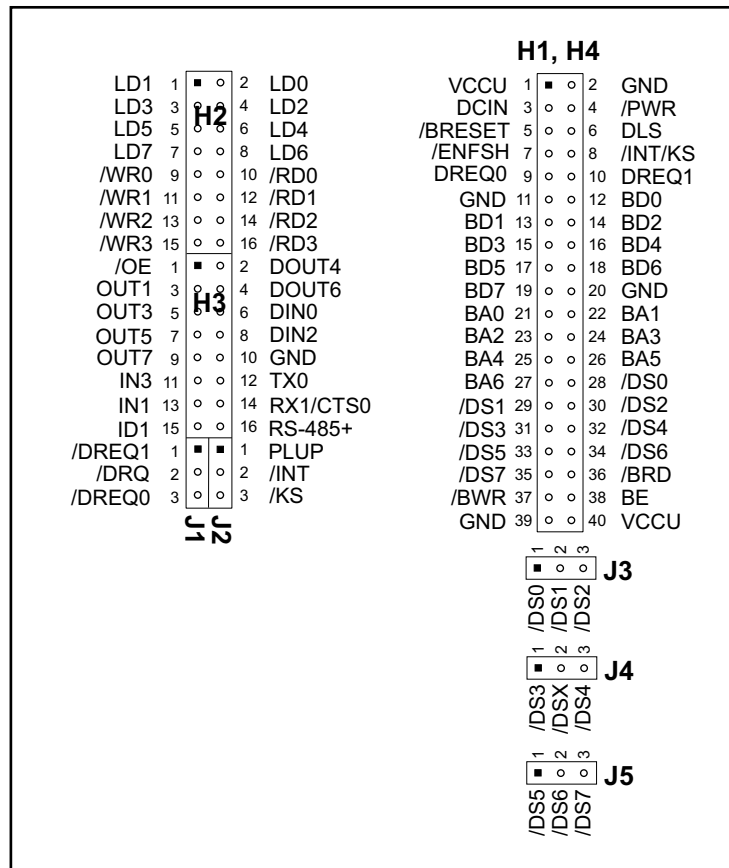


Figure E-3. LPBus Signal Pinouts on Prototyping Board

E.2 Design Considerations

E.2.1 Electrical

To avoid excessive loading of the LPBus, do not load any bus line with more than one input. If a bus line must drive more than one input, the line should be buffered. Since the /DS0–/DS7 lines are usually used by only a single board in each system, they do not usually need to be buffered. It is also important to keep connector and cable lengths as short as possible. The LPBus was designed to work reliably with up to eight expansion boards in the system, as long as all boards meet the specified design parameters.

The VHC logic family is well suited for use on LPBus expansion boards. VHC logic has the following advantages:

- Available from multiple sources (National Semiconductor, Motorola).

- Operation is well specified for both 3.3 V and 5 V supplies.
- Tolerant of inputs of up to 7 V.
- Low cost.

HC logic may be used, but its operation is not specified at 3.3 V. Programmable logic devices such as PALs may also be used, although their power consumption may be higher than VHC logic.

E.2.2 Board ID

Since different types of expansion boards can reside at different I/O addresses, it is useful for the software to be able to identify the type of board residing at a particular I/O slot (selected from one of eight /DS lines). The Board ID is read from the lowest address within the selected board's I/O space.

A board can have either a 4-bit or 7-bit Board ID. D7 identifies the width of the Board ID.

Table E-1 lists the LPBus ID types.

Table E-1. LPBus Device ID Types

D7 of Board ID Location	Mode	Board ID Data Bits
0	4-bit Board ID	D[3..0]
1	7-bit Board ID	D[6..0]

If a board is present, the Board ID defined below will be read. If no board is present, 255 decimal (binary 11111111) will be read.

Table E-2 lists the 4-bit Board IDs.

Table E-2. 4-bit Board IDs

Board ID, Decimal (Binary)	Reserved
0 (0000)	Customer use
1 (0001)	Customer use
2 (0010)	Customer use
3 (0011)	Customer use
4 (0100)	Z-World future expansion board
5 (0101)	Z-World future expansion board
6 (0110)	Z-World future expansion board
7 (0111)	Z-World future expansion board
8 (1000)	Z-World future expansion board
9 (1001)	Z-World future expansion board
10 (1010)	Z-World future expansion board
11 (1011)	Z-World future expansion board
12 (1100)	Z-World LPBus Prototyping Board
13 (1101)	Z-World LPBus Prototyping Board
14 (1110)	Z-World LPBus Prototyping Board
15 (1111)	Z-World LPBus Prototyping Board

7-bit Board IDs

The use of 7-bit Board IDs has not yet been defined by Z-World.

The 7-bit Board ID 255 (binary 11111111) is not a useable Board ID since this is what is read when no Board ID is found.



If a 7-bit Board ID is required, contact Z-World at (530) 757-3737.

E.2.3 No Connect Pins

Signals not used on the expansion board should be left unconnected. However, all 40 signals should be routed through if a pass-through connector is used. Always leave /ENFSH unconnected.

E.2.4 Use of DS Lines

The LPBus provides eight Device Selects for selecting slave boards. It is possible to have LPBus boards share the same DS address if additional qualification is done on other address lines.

E.2.5 DMA

Direct memory access is available on the LPBus. There are two DMA request lines on the LPBus: /DREQ0 and /DREQ1.

A typical application of DMA would be a high-speed sampling analog-to-digital conversion board. In sample mode, the A/D would make a conversion, then request a transfer of converted data directly to memory via DMA. This approach permits faster data transfer than using I/O instructions does. This is particularly important if the microprocessor is running at the slower 3 MHz speed. In addition to the improved speed, DMA transfers also uses less microprocessor time.

E.3 LPBus Timing

Figure E-4 shows the read-cycle timing for the LPBus with a 6.144 MHz system clock.

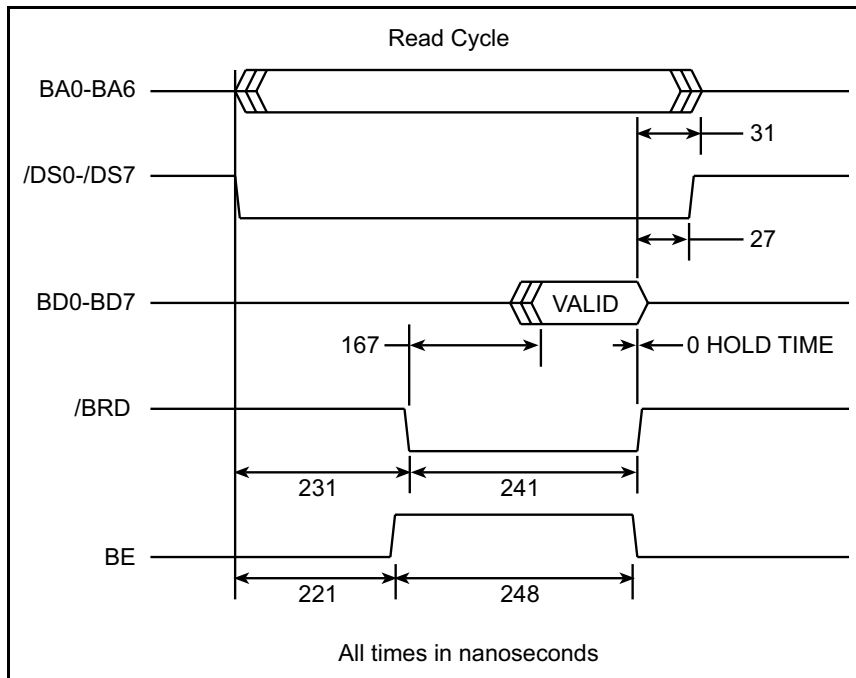


Figure E-4. LPBus Read-Cycle Timing

Figure E-5 shows the write-cycle timing for the LPBus 6.144 MHz system clock.

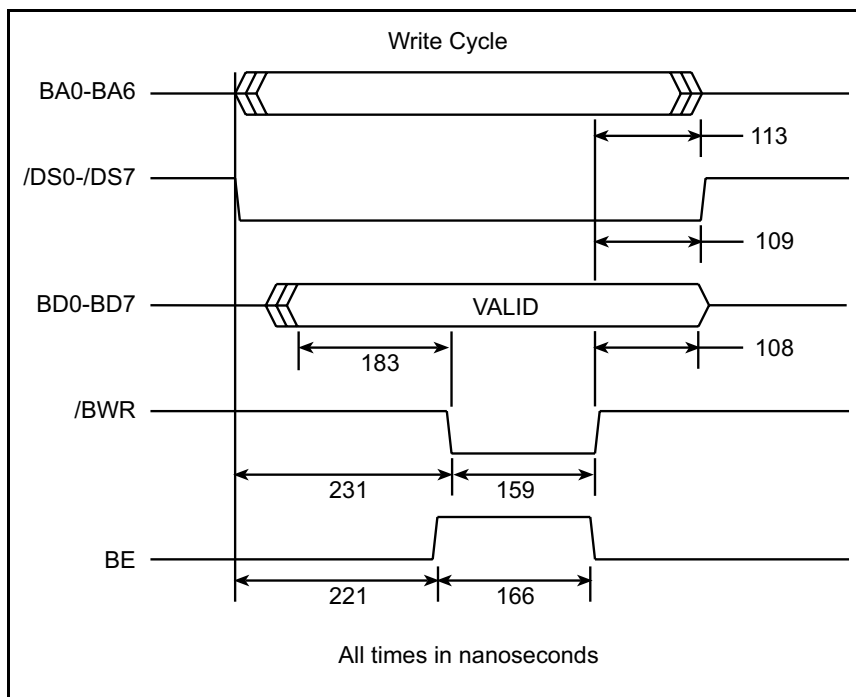


Figure E-5. LPBus Write-Cycle Timing

The LPBus timing shown in Figures E-4 and E-5 is based on calculations using worst-case delays over the entire operating temperature range. A 50 pF load is assumed for the LPBus driver IC's for the worst-case timing calculations. A 50 pF capacitance per line would represent a load of three expansion boards and their associated cables. Each additional expansion board and length of interconnecting cable adds approximately 15 pF to the load, and increases the bus delay by approximately 1 ns.

There is a 231 ns delay for write cycles after a device select line is lowered until the /BWR line goes low. This is the time available for decoding the device select and I/O address to qualify an internal write line on an expansion board. For a read operation, the selected board must present valid data on BD0-BD7 no later than 167 ns after the assertion of /BRD. The data hold time is 0 ns.

E.4 LPBus DMA

DMA can be used to provide high-speed I/O operations to or from the expansion boards. The DMA request lines, /DREQ0 and /DREQ1, are request lines for the Z180's two DMA channels.

/DREQ0 and /DREQ1 are active low and have a resistor pull-up to VCC. Since /DREQ0 and /DREQ1 can be driven by more than one source, they should only be driven low. If more than one board uses a DMA request line, ensure that multiple boards do not attempt DMA transfers simultaneously. /DREQ0 has priority over /DREQ1 in case of simultaneous requests.

The Z180 microprocessor may be programmed to respond to DMA requests on level or edge sense. Once a board has been initialized by software, the board can request a DMA transfer by pulling one of the request lines low.

The Z180 microprocessor samples the DMA request lines on the rising edge of the clock cycle just prior to state T3 (i.e., either T2 or Tw) of an instruction cycle. Since the expansion boards have no means to determine T-states directly, the request line should be held low until a DMA cycle results to insure that it has been recognized. What happens then depends on the request sense mode that has been programmed.

Once the requested I/O cycle has started, the point at which the Z180 microprocessor samples the request line for a subsequent cycle occurs just prior to the assertion of the /BRD or /BWR line. For the edge-sense mode, the board can use the assertion of the /BRD or /BWR lines to release the /DREQ_n line. Another cycle will not be initiated until the /DREQ_n line is lowered again.

For the level-sense mode, the request line can be held low until the next to last I/O cycle. During that cycle, the /BRD or /BWR line can be used to release the request line. Having sampled the line still low just prior to the assertion of /BRD or /BWR, the DMA channel will execute one more, final cycle. The /DREQ_n line can be raised and lowered during a level-sense DMA transfer if the transfer needs to be synchronized with some other event on your board, for example, as a buffer fills and empties. In that case the same timing situation applies regarding the assertion and release of the request line: hold it after assertion

to ensure that the Z180 has sampled it, and one additional cycle will occur upon releasing it after /BRD and /BWR .

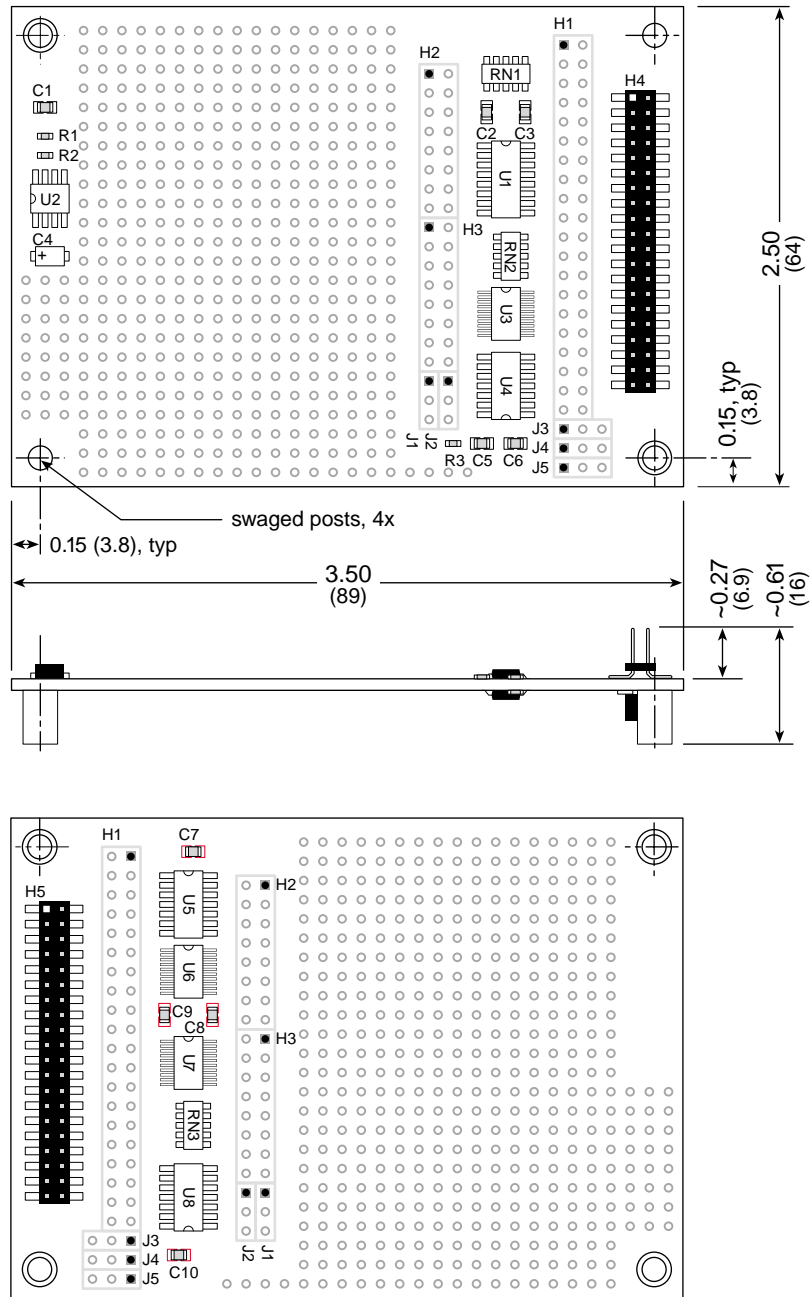
For the board to control the release of the /DREQ n line, it is possible to use the assertion of the device select line, /DS n . In that case, the assertion of /DS n could force the release of DREQ n before the assertion of /BRD or /BWR, which would cause no additional DMA cycles to occur after the cycle in which the request line was released. For this case to work correctly, the design would have to implicitly know the I/O address of the source or destination of the DMA transfer since the device select alone does not specify a particular address. The device select line is safe to use in this way since it is gated by /IORQ on the LP3100 before being placed on the LPBus. It is risky to try to decode an I/O address using BA0-BA6 together with the leading edge of /DS n as a condition to release /DREQ n since the address lines are in state of transition at that time. A false decode is possible because of uneven delays and races conditions in the decode logic. This is not a reliable design approach unless a synchronous design is used to delay and sample the /DS n by a couple of clock periods.

The DMA controller can be programmed to operate in either the burst mode or the cycle-steal mode. In the cycle-steal mode, DMA cycles are interleaved with normal program cycles. Since the expansion bus does not provide a means to determine whether or not an I/O cycle is a DMA cycle, it should be assumed to be a DMA cycle once a board is appropriately initialized by software and until the end of the transfer. In other words, the software should perform the initialization needed to set up the board to participate in a DMA transfer before the final I/O access that “arms” it. The software should not poll the board’s status by reading from the board while a DMA transfer is taking place. These restrictions do not apply if the I/O address programmed for the DMA transfer is different from the setup and status registers and if the address is fully decoded during the DMA cycle.

The Z180’s /TEND0 and /TEND1, which signal the last cycle of a DMA transfer, are not available on the LPBus. Therefore, a programmable counter or similar mechanism must be included on an expansion board to terminate the DMA requests (i.e., stop asserting /DREQ n) after the correct number of I/O cycles has taken place.

E.5 Dimensions

Figure E-6 illustrates the dimensions of an LPBus Prototyping Board.



Headers H4 and H5 are not centered, but are offset 0.050" (1.3 mm).

Figure E-6. LPBus Prototyping Board Dimensions



APPENDIX F. POWER MANAGEMENT

Appendix F provides information about hardware and software specific to power management.

F.1 Input Voltage

The minimum direct current input voltage to the LP3100 is 5.2 V if the +5 V supply is used for the digital outputs. If the +5 V supply is not used, the minimum input voltage is 3.5 V. The maximum input voltage depends on the ambient air temperature and current draw from the 3.3 V regulator, but should never exceed 24 V.

When calculating the maximum input voltage, measure the temperature of the LP3100 exposed to the same conditions and environment that will exist in the field. Temperature of the LP3100 should be measured on the surface of voltage regulators U2 and U3. If the LP3100 is enclosed, heat may be retained within the enclosure and cause the ambient air temperature to rise. If the operating temperature is near or below the ambient temperature, the air acts as an insulator and device's temperature will rise. Try to simulate a worst case scenario. If a worst-case scenario cannot be simulated, determine the rise in temperature within the enclosure and add this to the expected maximum temperature outside the enclosure.

Determine the current used by the circuits external to the LP3100. Add this to the current draw of the LP3100. The current draw for various conditions is shown in Table F-1.

Table F-1. LP3100 Resource Current Draw

LP3100 Resource	Current Draw
Microprocessor (3.072 MHz)	11 mA
Microprocessor (6.144 MHz)	15 mA
Analog Input Section	1.5 mA
RS-232 Transceiver	1.0 mA
Diagnostic LED	5.0 mA
5 V Supply for Output Latches	4.0 mA

Compute the maximum DC input voltage by using Equation (F-1).

$$V_{IN} = 3.3 + \left(\frac{1}{I_{MAX} \cdot \theta_{JA}} \right) \cdot (T_{J_{max}} - T_{ambient}) \quad (F-1)$$

$T_{J_{max}}$ is 125°C. This is the maximum allowable junction (die) temperature for the regulators.



A regulator's life expectancy doubles with every 10°C reduction in junction temperature.

θ_{JA} is the thermal resistance between the junction and the ambient air around the regulators. This is a function of the PCB construction and the air movement across the regulator. Assuming static air movement, use a value of 160° C per watt.

Example:

A data logging system that uses an LP3100, a sensor, and a seven-segment LED display is placed inside a small enclosure with no air flow.

The LP3100 is running at 3.072 MHz and is using the analog input section. The current draw in this configuration is about 12.5 mA. The sensor draws about 3 mA and the seven-segment LED draws 35 mA (seven segments at 5 mA). The total current draw is 50.5 mA. The worst case ambient temperature inside the enclosure is 85° C. Add a safety margin, about 20 percent in this case, to get 61 mA. Substituting these values into Equation (F-1) yields

$$\begin{aligned} V_{IN} &= 3.3 \text{ V} + (125^{\circ}\text{C} - 85^{\circ}\text{C}) \left(\frac{1}{61 \text{ mA} \cdot 160^{\circ}\text{C/W}} \right) \\ &= 3.3 \text{ V} + 0.10 \text{ V/}^{\circ}\text{C} \cdot 40^{\circ}\text{C} \\ &= 7.4 \text{ V} \end{aligned} \tag{F-2}$$

The maximum input voltage for this system would be 7.4 V. Lowering the current draw or the ambient temperature will increase the input voltage limit. Increasing the current draw or the ambient temperature will decrease the input voltage limit. See Figure F-1.

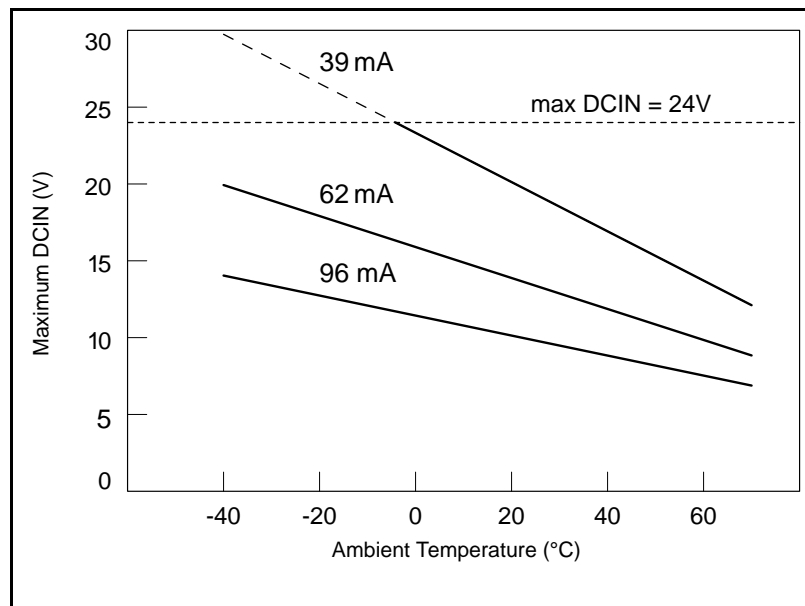


Figure F-1. LP3100 Input Voltage vs. Ambient Temperature

F.2 Power-Failure Detection

Figure F-2 shows the power-fail detection circuitry of an LP3100.

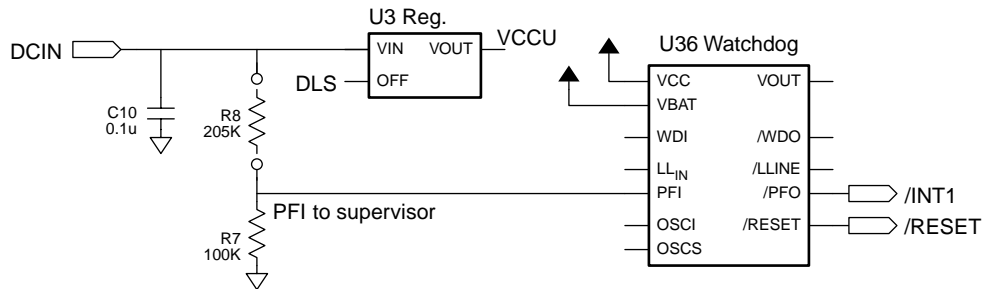


Figure F-2. LP3100 Power-Failure Detection Circuit

F.2.1 Power Failure Sequence of Events

The following events occur as the input power fails.

1. The power-management IC triggers a power-fail interrupt (/INT1) when the DC input voltage falls within the range of 4.33 V to 3.61 V DC.
2. At some point, the raw input voltage will not be sufficient for the regulator to provide 3.3 V DC to the LP3100 because of the dropout voltage. At that point the regulated output begins to drop.

Use a power supply with large capacitance to increase the holdup time. This will provide additional time for the LP3100 to execute a safe shutdown.

3. The power management IC switches power for SRAM to the backup battery when the regulated voltage falls below the battery input voltage.
4. The power management IC keeps the system in reset until the regulated voltage drops below 1 V DC. At this point the power-management IC ceases operating. By this time, the portion of the circuitry not battery-backed has already ceased functioning.

The ratio of the power supply's output capacitor value to the circuit's current draw determines the actual holdup time.

A situation similar to a continuous low input (“brownout”) can occur if the power supply is overloaded. For example, when a high-current device such as a relay turns on, the raw voltage supplied to the LP3100 may dip below the power-failure threshold. The interrupt routine performs a shutdown. This shutdown turns off the relay, clearing the problem. However, if the cause of the overload persists, the system oscillates, alternately experiencing an overload and then resetting. Using a power supply with a greater capacity will correct this problem.

If the power source is removed abruptly from the LP3100, then only the capacitors on the board provide power, reducing computing time to a few microseconds. These times can vary considerably depending on the system configuration and loads on the LP3100 power

supplies. External power supply capacitance determines the time it takes for power to completely fail.

Figure F-3 summarizes the power-failure sequence.

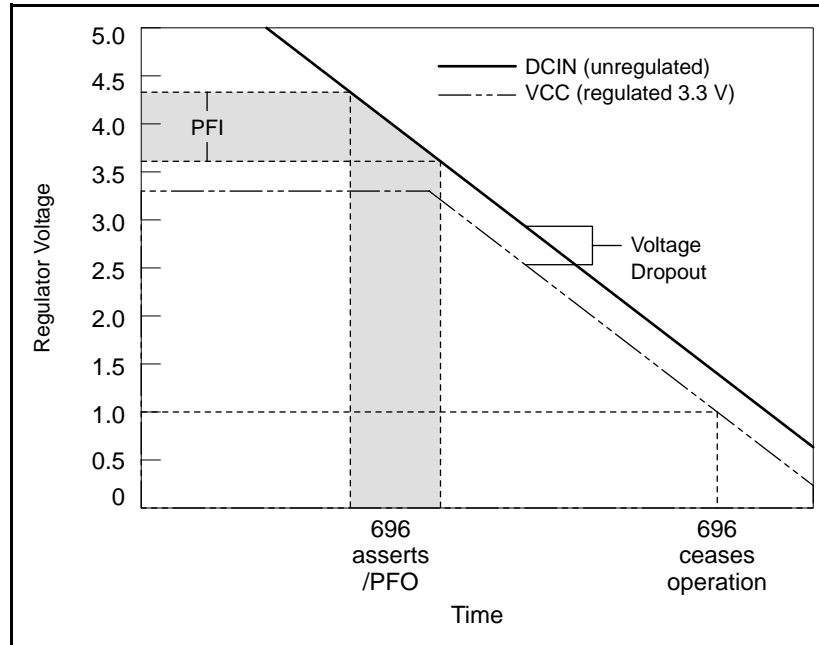


Figure F-3. Power-Fail Sequence

The supervisor switches VRAM to VBAK or VCC, whichever is greater. Since the supervisor has some hysteresis, it does not keep switching. At some time during a power failure, VCC falls Below VBAT.

When VCC falls to approximately 3.0 V, a voltage divider trips the LLIN input to the supervisor, causing it to assert RESET and /RESET.



APPENDIX G. INTERRUPTS

Appendix G presents a suggested interrupt vector map. Most of these interrupt vectors can be altered under program control. The addresses are given here in hex, relative to the start of the interrupt vector page, as determined by the contents of the I-register. These are the default interrupt vectors.

G.1 Enabling/Disabling Interrupts

Interrupts can be enabled or disabled by including the following commands in the code:

- Enable “ON” Interrupt 0

```
outport(ITC,inport(ITC) | 1)
```

- Enable “ON” Interrupt 1

```
outport(ITC,inport(ITC) | 2)
```

- Disabled “OFF” Interrupt 0

```
outport(ITC,inport(ITC) & 0xFE)
```

- Disabled “OFF” Interrupt 1

```
outport(ITC,inport(ITC) & 0xFD)
```

G.2 Interrupt Service Routines

Interrupt service routines (ISRs) are packets of code that the microprocessor jumps to and executes when it receives an interrupt request.



Refer to the *Dynamic C Technical Reference Manual* for instructions on writing interrupt service routines.

Refer to the *Zilog Z180/180 User's Manual* (available from Z-World) for complete details on using Z180 interrupts.

G.3 Interrupt Vectors

To “vector” an interrupt to a user function in Dynamic C, use a directive such as the following:

```
#INT_VEC 0x10 myfunction
```

The above example causes the interrupt at offset 10H (serial port 1 of the Z180) to invoke the function `myfunction()`. The function must be declared with the interrupt keyword:

```
interrupt myfunction() {  
    ...  
}
```

Table G-1 lists the Z180 internal interrupt vectors.

Table G-1. Z180 Internal Device Interrupt Vectors

Address	Name	Description
0x00	INT1_VEC	/INT1
0x02	INT2_VEC	/INT2
0x04	PRT0_VEC	Programmable Reload Timer Channel 0
0x06	PRT1_VEC	Programmable Reload Timer Channel 1
0x08	DMA0_VEC	DMA Channel 0
0x0A	DMA1_VEC	DMA Channel 1
0x0C	CSIO_VEC	Clocked Serial I/O
0x0E	SER0_VEC	Serial 0
0x10	SER1_VEC	Serial 1

G.4 Jump Vectors

Jump vectors are similar to interrupt vectors, except that instead of loading the address of the interrupt routine from the interrupt vector, these interrupts cause a jump directly to the address of the vector, which contains a jump instruction to the interrupt routine. Below is an example of a jump vector:

```
0x66non-maskable power-failure interrupt
```

Because nonmaskable interrupts (NMI) can be used for Dynamic C communication, the interrupt vector for power failure is normally stored just in front of the Dynamic C program. Store a vector there by using the following compiler directive.

```
#JUMP_VEC NMI_VEC name
```

The Dynamic C communication routines jump to the NMI vector when a power failure causes the NMI rather than a serial interrupt.

Table G-2 lists the interrupt priorities.

Table G-2. Interrupt Priorities

Interrupt Priorities	
(Highest Priority)	Trap (Illegal Instruction)
	NMI (Nonmaskable Interrupt)
	INT0 (Maskable Interrupt, Level 0)
	INT1 (Maskable Interrupt, Level 1)
	INT2 (Maskable Interrupt, Level 2)
	Programmable Reload Timer, Channel 0
	Programmable Reload Timer, Channel 1
	DMA Channel 0
	DMA Channel 1
	Clocked Serial I/O
	Serial Channel Z0
(Lowest Priority)	Serial Channel Z1



APPENDIX H. ADDRESSES

Appendix H presents information on EEPROM, microprocessor registers, and LP3100 peripheral addresses.

H.1 Simulated EEPROM Addresses

The LP3100 uses a section of the flash EPROM to simulate EEPROM. The size of the simulated EEPROM is 512 bytes (not 512K). Locations 0x02 through 0x19 are used for storing the analog input calibration constants. The rest of the simulated EEPROM is free for use by the application.

H.2 Microprocessor Register Addresses

The Z180's internal I/O registers occupy the first 40_n addresses. Refer to the Zilog Z80180/Z180 MPU internal I/O register map.

H.3 LP3100 Peripheral Addresses

Table H-1 lists the addresses that control the I/O devices external to the Z180 microprocessor.

Table H-1. LP3100 External I/O Device Registers

Address	R/W	Name	Function
4000H	R	/RDRTC	RTC read
4000H	W	/WRRTC	RTC write
4020H	R	/CS1R	DINOUT read
4020H	W	ALERTC	RTC address latch enable
4040H	R	/CS2R	DIN read, DLS and watchdog monitor, ADC
4040H	W	/CS2W	DINOUT write
4060H	R	HITWD	Watchdog timer
4060H	W	/CS3W	DOUT write
4080H	R	/EN_DKS	DINO kickstart enable
4081H	W	/BRESET	LPBus reset
4082H	W	/232TEN	RS-232 transmit enable
4083H	W	232EN	RS-232 driver enable
4084H	W	/CTSEN	5-wire RS-232
4085H	W	/PWREN	Power supply enable
40A0H	W	LED	LED control
40A1H	W	485TE	RS-485 transmit enable
40A2H	W	/485RE	RS-485 receive enable
40A3H	W	/AD_CS	ADC select
40A4H	W	/DOE	DINOUT output enable
40A5H	W	/ADENA	AVCC power supply enable

Table H-1. LP3100 External I/O Device Registers

Address	R/W	Name	Function
8000H–807FH	W	/DS0	LPBus Device Select 0
8080H–80FFH	W	/DS1	LPBus Device Select 1
8100H–817FH	W	/DS2	LPBus Device Select 2
8180H–81FFH	W	/DS3	LPBus Device Select 3
8200H–827FH	W	/DS4	LPBus Device Select 4
8280H–82FFH	W	/DS5	LPBus Device Select 5
8300H–837FH	W	/DS6	LPBus Device Select 6
8380H–83FFH	W	/DS7	LPBus Device Select 7



APPENDIX I. OPTIONAL SECOND FLASH EPROM

I.1 Optional Flash EPROM

The LP3100 series controllers can have an optional second flash EPROM installed. If an application needs to use flash EPROM storage to log information or store information that can be updated at run-time, a second flash EPROM is better than using the only flash EPROM on the system. With a second flash EPROM, the system does not have to disable interrupts when writing to the flash EPROM. Writing to the flash EPROM can potentially take up to 10 ms. Furthermore, since the second flash EPROM is not used to store code or the debugger kernel, a software bug in the application using the second flash EPROM is less likely to corrupt the application itself or the debugger kernel.

The Dynamic C `sys.LIB` and `xmem.LIB` libraries provide functions to check for the existence and type of the second flash EPROM as well as functions to access it. The following functions are provided.

```
int sysChk2ndFlash( struct _flashInfo *pInfo )
```

Checks for the existence and configuration of the second flash EPROM mapped to memory space.

PARAMETER

`*pInfo` is a pointer to a `struct _flashInfo` that stores the configuration of the flash EPROM.

RETURN VALUE

0 if the second flash EPROM exists and the configuration is valid. Otherwise, a negative number is returned.

```
void sysRoot2FXmem( struct _flashInfo *pInfo,  
void *src, unsigned long int dest, unsigned int len )
```

Copies memory content from root memory to second flash EPROM.

PARAMETERS

`*pInfo` is a pointer to a `struct _flashInfo` (initialized by `sysChk2ndFlash`).

`*src` is a pointer to the beginning of the block in root memory to be copied to the second flash EPROM.

`dest` is the physical address that points to the beginning of the block in the second flash mapped to memory space.

`len` is the length of the block to be copied.

```
void xmem2root( unsigned long int src,  
void *dest, unsigned int len )
```

Stores `len` characters from physical address `src` to logical address `dest`.

I.2 Sample Program

The following sample program, **WRFLASH.C**, writes the literal string **Test 2nd flash\r\n** to fixed locations in the second flash EPROM, then reads back the locations and displays them in the **STDIO** window.

WRFLASH.C

```
/*This program demonstrates how to write information into the second flash
EPROM (if equipped) and read the information back. To initialize, call
sysChk2ndFlash to initialize parameters of the second flash. To write
information to the flash, use sysRoot2FXmem (much like root2xmem, but
writes to flash space instead of RAM space). To read information from the
flash to root space, use xmem2root.*/

#include eziolp31.lib
#include xmem.lib

main() {
    char strbuf[50];
    struct _flashInfo pInfo;
    unsigned long cur_addr;
    int i;

    _GLOBAL_INIT();

    // check to see that 2nd flash exists

    if ( sysChk2ndFlash ( &pInfo ) != 0 ) {
        printf( "No 2nd Flash - cannot log data!!" );
    }

    hitwd();
    cur_addr = 0x40000;
    sprintf ( strbuf, "Test 2nd flash\r\n" );
    printf ( "Starting logging test\n" );
    for ( i = 0; i < 10; i++ ) {
        sysRoot2FXmem( &pInfo, strbuf, cur_addr, 16 );
        printf ( "\ncur_addr = %ld", cur_addr );
        cur_addr += 16;
        hitwd();
    }
    hitwd();
    printf ( "\nreading from 2nd flash\n" );
    hitwd();
    cur_addr = 0x40000;
    for ( i = 0; i < 10; i++ ) {
        xmem2root ( cur_addr, strbuf, 16 );
        printf ( "\n%3d %7ld : %s", i, cur_addr, strbuf );
        cur_addr += 16;
        hitwd();
    }
}
```


Symbols

+5 V	22
/DREQ0	123
/DREQ1	123
/DS0_S7	118
/DSx	58
/EN_DKS	22
/ENFSH	58
/INT/KS	22, 56, 58
/PWR	57

A

addresses	
EEPROM (simulated)	138
microprocessor registers	138
peripheral	138
ADVCC	22
analog inputs	6
calibration constants	70
configuration	37, 38
frequency response	46
gain calculation	40
gain resistors	38, 40
initializing	70
input impedance	46
Input ranges	39
offset resistors	38
offset voltage	44
calculation	40
reading	47, 69
sample program	47
scaling inputs	40
signal conditioning	37
applications	2

B

battery backup	21, 100
board layout	
Development Board	108
LPBus Prototyping Board	116

C

CE compliance	8
clock speed	5, 19, 24
change 3 MHz	
lp31Clk3MHz	25
change 6 MHz	
lp31Clk6MHz	25

clocked serial I/O	
interrupt priorities	136
Compile	
sample program	15
component shutdown	27
CSI/O	25

D

data logging	5, 19
DCIN	19, 57, 130, 131
Development Board	12, 108
device select	
lines	58
outputs	59
digital I/O	
configuring	34
input voltage	33
output voltage	33
sample program	35
Digital inputs	
sample program	29
digital inputs	
input voltage	28
pull-up resistors	28
reading	29, 64, 66
digital outputs	
output voltage	30
sample program	32
writing	65, 67
DIN0	22
pull-up resistor	28
DIN0&N3	28
DLS	21
DMA Channel 0	
interrupt priorities	136
DMA Channel 1	
interrupt priorities	136
DMA request	58
driver disable mode	
RS-232	27
DS1685	51

E

EEPROM (simulated)	
addresses	138
eioBrdACalib	70
eioBrdAI	69
eioBrdDI	64
eioBrdDO	27, 65

eioBrdInit	70
EZIOLP31.LIB	62

F

flash EPROM	5, 138
write cycle limit	20

G

GND	57
-----	----

H

H1 (LP3100)	100
H2 (LP3100)	28
pinout	100
H3 (LP3100)	
pinout	100
halt mode	26
header locations	
Development Board	108
LPBus Prototyping Board	116
holdup time	130

I

inport	66
input voltage	
digital inputs	28
digital inputs/outputs	33
INT0	
interrupt priorities	136
INT1	
interrupt priorities	136
INT2	
interrupt priorities	136
interrupt priorities	
clocked serial I/O	136
DMA Channel 0	136
DMA Channel 1	136
INT0	136
INT1	136
INT2	136
nonmaskable interrupt	136
PRT Timer Channel 0	136
Serial Channel Z0	136
Serial Channel Z1	136
trap	136
interrupts	
alarm	54
power failure	130

J			
J1			
output latch	31		
jump vectors	136		
K			
keypad			
detecting keypresses	86		
initialization	87		
kpDefGetKey	86		
kpDefInit	87		
L			
LCD			
delay	84		
disabling blinking cursor ...	83		
enabling blinking cursor	83		
initialization	84		
positioning cursor	83		
printing text	83, 84		
reading status	84		
wait	85		
writing control characters .	84,		
85			
lcdCtrl	85		
lcdDisBlink	83		
lcdEnBlink	83		
lcdInit	84		
lcdLongWait	84		
lcdPos	83		
lcdPrintf	84		
lcdRead	84		
lcdVprintf	83		
lcdWait	85		
lcdWrite	84		
low-power modes	75		
LP3100 versions	3		
lp31Clk3MHz	77		
lp31Clk6MHz	78		
lp31PFO	77		
lp31Shutdown	76		
LPBus	6		
/ENFSH	121		
addresses	58		
BA6 power control	58		
board ID	59		
capacitive loading	123		
device select	55, 121		
DMA	121		
expansion board			
design	116		
mounting	55		
LPBus (continued)			
interface logic	118		
kick start	58		
mechanical interface	55		
power control	57		
read	59		
reset	58		
write	59		
LPBus Prototyping Board ...	116		
LPBus Prototyping Board Di-			
mensions	125		
M			
microprocessor	5		
operating modes			
halt	26		
sleep	25		
standby	25		
register addresses	138		
supervisor	5		
MS_TIMER	26		
multidrop network	51		
N			
nonmaskable interrupts	136		
interrupt priorities	136		
O			
operating modes	25		
output	67		
output voltage			
digital inputs/outputs	33		
digital outputs	30		
P			
periodic interrupts			
flag	54		
pinouts			
H3 (LP3100)	100		
power			
SIB2	104		
power control	5, 57		
power failure			
detection circuit	130		
interrupt	130		
power monitoring	19		
power supply control			
analog inputs	23		
digital outputs	23		
features	23		
powerdown mode	23, 26		
shutdown mode	23		
powerdown mode	23, 26		
wake up	28		
PRT Channel 0			
interrupt priorities	136		
PRT Channel 1			
interrupt priorities	136		
R			
RBOTTOM	38		
Rg	38		
RJ-12	104		
RS-232 serial communication ...			
48, 49, 50			
AASC.LIB	50		
low-power modes			
shutdown	27		
RS-485 serial communication ...			
48, 49, 50			
Serial Channel Z1	50		
termination resistors	51		
RTC	6		
alarm	54		
alarm enabling/disabling ...	80		
alarm fields	54		
"don't care"	54		
alarm status	80		
battery backup	51		
drivers	51		
interrupts	52		
periodic interrupts	53		
software	51		
rtcChkAF	80		
rtcSwAIE	80		
S			
SEC_TIMER	26		
Serial Channel Z0	48, 49		
interrupt priorities	136		
Serial Channel Z1	48, 49, 50		
interrupt priorities	136		
serial channels	6		
baud rates	48		
configuration	48		
handshaking	48		
RS-232	50		
sample programs			
AASC	50		
software	51		
shutdown mode			
RS-232	27		
SIB2	12, 13, 19		
baud rate	104		
power	104		

sleep mode	25
SRAM	5
standby mode	25
static RAM	5
sysHalt	26
sysSleep	25, 75
sysStandby	26, 75
system clock speed	19

T

trap	
interrupt priorities	136
troubleshooting	
board resets	91
cables	90
communication error	90
grounding	90
PC COM ports	90

V

VBAK	22
VCC	22
restoring	22
shutdown	24
VCC resumption	
kick start	24
time-based	24
VCCU	21
alarm interrupts	54
VRAM	22

W

wakeup	
kick start	24
time-based	24
watchdog	
timeout	19

SCHEMATICS
